робМіністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-
МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Мобільні додатки з технологією доповненої реальності, з використанням мови Swift та фреймворків ARKit та Core ML

Текстова частина до курсової роботи
за спеціальністю «Інженерія програмного забезпечення» — 121

Керівник курсової роботи
к-т фіз.-мат. наук, доцент
Гороховський С.С.

_____

(Підпис)
«___» _____ 2020 року
Виконав студент БП ІПЗ-4
Маргаль Н. М.
«___» _____ 2020 року

Київ 2020

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики технологій факультету інформатики
ЗАТВЕРДЖУЮ
Викладач кафедри інформатики,
канд. фіз-мат. наук, доц. _____ Гороховський С.С.
(підпис)
«____» _____2019р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу
студенту Маргалю Никандру Максимовичу
Факультету інформатики 4 р.н. бакалаврської програми
ТЕМА: Мобільні додатки з технологією доповненої реальності, з використанням мови Swift та фреймворків ARKit та Core ML

Зміст текстової частини до курсової роботи:
Індивідуальне завдання
Вступ
Огляд теоретичного матеріалу і здійснення дослідження
Висновки
Список літератури
Додатки (за необхідністю)
Список посилань

Дата видачі «___» _____ 2019 р. Керівник_____
(підпис)
Завдання отримав _____
(підпис)

Тема: **Мобільні додатки з технологією доповненої реальності, з використанням мови Swift та фреймворків ARKit та Core ML**

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	13.10.2019	
2.	Огляд літератури за темою роботи.	10.02.2020	
3.	Проведення досліджень	20.02.2020	
3.	Опис результатів дослідження	27.02.2020	
4.	Аналіз отриманих результатів з керівником,	19.03.2020	
6.	Корегування роботи	13.04.2020	
7.	Створення презентації та написання доповіді.	16.04.2020	
8.	Остаточне оформлення пояснювальної роботи та слайдів.	18.04.2020	
9.	Здача курсової роботи на перевірку	19.04.2020	

Студент Маргаль Н. М.

Керівник Гороховський С. С.

“_____”_____

ЗМІСТ

Вступ

Актуальність та практичне значення обраної теми

Станом на 2020-й рік, смартфони через свою поширеність та доступність є чи не головною платформою для поширення програмного забезпечення до кінцевого користувача. Постійно існує попит на сервіси «в кишені», які полегшують життя людям, вирішують буденні задачі та допомагають відволіктися.

Не зважаючи на те, що багато мобільних додатків пропонують своїм користувачам дуже достойний рівень UX (user experience — з англ. дослівно «користувацький досвід»), розробники постійно змушені переглядати відповіді на питання: «як утримати користувача від видалення цього додатку?» і «як змусити нових клієнтів встановити його?». Це, фактично, є одним з головних рушіїв прогресу користувацьких технологій, що спонукає компанії розробляти все нові способи зацікавити користувача.

Серед технологій, що з’явилися на ринку користувацької електроніки не так давно і відкрили надзвичайно багато цікавих рішень у різноманітних сферах виділю дві — машинне навчання (Machine learning, ML) та доповнена реальність (Augmented reality, AR). Компанії, як от Apple чи Google, витрачають дуже багато ресурсів на проєктування та створення нових застосувань для цих технологіям. Це може стосуватися власних застосунків компаній, наприклад створення 3D-моделей, що доступні для перегляду багатьом користувачам через камеру, вимірювання довжини чи площі реальних об’єктів з використанням камери і гіроскопу, навіть можливість побачити фізичні розміри нового продукту компанії на власному столі та роздивитися його. З іншого боку, ці ж самі компанії зацікавлені у створенні зручних у використанні API (application programming interface — з англ. Прикладний програмний інтерфейс), щоб

надати можливість стороннім розробникам створювати власні застосування цих технологій.

Існує велика кількість прикладів, коли застосування таких технологій як машинне навчання чи доповнена реальність в мобільних додатках не лише виправдана, а й відкриває купу цікавих та неочікуваних переваг як для користувачів, так і для розробників. Перш за все, створюється абсолютно новий спосіб взаємодії користувача з інтерфейсом додатку. Відбувається перехід від кнопок, жестів та інших «звичних» нам елементів інтерфейсу до більш інстинктивних та, певною мірою, природніх — взаємодія з реальними об’єктами через смартфон, використання фізичних жестів над самим смартфоном для роботи з додатком, іноді навіть суттєве спрощення та переосмислення «класичних», хоча, насправді, дуже складних інтерфейсів. Прикладом останнього є Grib [1] — його розробники створили 3D-редактор, що керується лише двома кнопками та просторовими жестами пристрою.

Хотілося б згадати про сегмент додатків, які слугують своєрідними «примірочними» як для одягу, так і для більш складних і громіздких предметів декору. Вони дозволяють побачити різноманітні речі в звичному для користувача середовищі без необхідності їх купувати. Найвідомішими прикладами таких додатків є IKEA Place [2], що надає можливість розмістити віртуальні меблі IKEA у себе вдома та Wanna Kicks [3] — додаток для примірки модного взуття.

І, звісно ж, не забудьмо про мобільні ігри, які використовують доповнену реальність для створення нових і цікавих геймплейних рішень, а також для покращення якості зображення та фізики.

Хоча машинне навчання важче «помітити» в додатку, цією технологією підкріплені багато з тих, якими ми користуємося щодня. Це, перш за все — нотатки, що розпізнають рукописний ввід, перекладачі, як от Google Translate, що перекладають в реальному текст, на який «дивиться»

камера, голосові асистенти — Google Assistant, Siri, Alexa та інші, соціальні мережі з їхніми адаптованими стрічками і рекламою тощо.

Технології машинного навчання та доповненої реальності мають велику кількість застосувань. Часто ці застосування виникають у досить неочікуваних місцях. Проте, завдяки наявності потужного інструментарію, впровадження технологій в нові та існуючі додатки є досить нескладною задачею. Власне, рішення створення додатку в рамках курсової вмотивоване доведенням попереднього твердження.

Структура роботи

Робота складається зі вступу, трьох розділів, висновків, списку літератури та додатків.

Постановка задачі

1. Проаналізувати принципи роботи машинного навчання та доповненої реальності в рамках мобільних операційних систем.
2. Розглянути і дослідити технології для роботи з машинним навчанням (Core ML) та доповненою реальністю (ARKit) в рамках операційної системи iOS.
3. Використовуючи мову Swift, фреймворки Core ML та ARKit, створити iOS-додаток для інтерактивних виставок.

Розділ 1. Аналіз предметної області. Постановка завдання курсової роботи

1.1. Базові поняття про машинне навчання

Машинне навчання — це наукова галузь, яку часто класифікують як підгалузь штучного інтелекту, яка вивчає методи надання комп'ютерним системам «навчатися» для розв'язування певної задачі. Машинне навчання, на противагу традиційному підходу до розв'язання задачі, не вимагає безпосереднього створення та програмування алгоритму, який би її вирішував. Навпаки створюється певна математична модель, що «навчається» на великій кількості даних (тренувальний набір), що складаються з вхідних параметрів та результату, який модель повинна повернути після обрахунків. Проаналізувавши такий набір даних, модель отримує змогу з певною точністю вірно оперувати над даними, що не асоційовані ні з якою формою «правильної відповіді». Такі дані ще називають тестувальним набором.

Взагалі кажучи, підхід, що описаний вище, вирішує лише частину задач машинного навчання. Такий підхід називається «навчання з учителем», або ж «supervised learning». Він полягає в тому, що система отримує множину прикладів («стимул-реакція»), тренується на цих прикладах та отримує в подальшому змогу прогнозувати реакції отримуючи стимули, що перед тим не надходили до системи.

Прикладом навчання з учителем є задача розпізнавання об'єкту на зображенні. В такій задачі моделі, що навчається, надають набір зображень, кожне з яких містить з-поміж усіх інших об'єкт, який необхідно розпізнати і виділити. Кожне зображення з тренувального набору якимось (але обов'язково однаковим для всіх прикладів) чином розмічене так, щоб можна було зрозуміти, в якій області зображення знаходиться цільовий

об’єкт. Такі позначення ще називають лейблами (від англ. label — позначка).

Іншим підходом є «навчання без учителя» (unsupervised learning). Воно полягає в тому, що тренувальний набір містить дані, що не асоційовані ні з якими лейблами. Такий підхід, як правило, використовують для розв’язування задач, де потрібно знайти внутрішні взаємозв’язки, залежності чи закономірності в об’єктах.

Одним з прикладів такого підходу є задача кластеризіції. Це не одна конкретна задача, адже для кожного окремого випадку розуміння сутності кластеру — певної однорідної за якимись ознаками групи об’єктів — може суттєво відрізнятися.

Наостанок, принцип навчання з підкріпленням (англ. reinforcement learning) полягає у пошуку якоїсь дії, що максимізує винагороду в певній ситуації. Навчання з підкріпленням не передбачає ніякого тренувального набору даних. Навпаки, агент моделі вимушений керуватися своїм «попередні досвідом» для вирішення, яка дія на поточному кроці буде оптимальною для отримання винагороди.

Хорошим прикладом застосування такого підходу є гра в комп’ютерну гру «змійка». Програмний агент, що керує діями змійки, змушений в кожну мить часу вирішувати — продовжити шлях у поточному напрямку, повернути ліворуч, або ж праворуч. Винагородою в цьому контексті є поїдання їжі та збільшення загального ігрового рахунку. Якщо ж дії агента призведуть до закінчення гри — тобто змійка «з’їсть» себе або удариться в стіну — агент отримає негативну винагороду. В подальшому ці дані будуть враховані для оптимізації підходу до керування змійкою.

Для вирішення задач машинного навчання різних класів використовуються різнотипні програмовані моделі. В рамках цієї роботи, науковий інтерес становить лише клас моделей, що відомі як «штучні нейронні мережі».

Нейронна мережа — це обчислювальна система, що концептуально натхненна будовою мозку тварин. Така система, вирішуючи задачу, поступово покращує свою продуктивність обробляючи дані, що їй приходять на вхід, проте не потребує специфічної розробки під задачу. Наприклад, була розроблена нейронна мережа, що вміє класифікувати зображення — себто віднести зображення до одного з двох класів — «хот-дог» та «не хот-дог». В процесі навчання модель отримала тренувальний набір, що складався з картинок, що були вручну відмічені та віднесені до одного з двох перелічених класів. Базуючись на даних, що проаналізувала модель, вона навчилася відрізняти хот-дог від не-хот-догу не маючи жодної уяви про його сутність. Навпаки, характеристики для ідентифікації будуть створені автоматично, базуючись на прикладах, які модель обробить.

Очевидно, що всередині моделі відбуваються операції над скалярами, векторами та матрицями, що є математичним представленням для об’єктів реального світу — нехай то звук, текст, чи, як в даному випадку, зображення. Отож характеристики, що створить модель для ідентифікації хот-догів матимуть виняткова числовий характер й будуть базуватися на шаблонах, що модель ідентифікує на зображенні.

Нейронні мережі використовуються для розв’язання широкого спектру задач штучного інтелекту. Серед таких: комп’ютерне бачення (класифікація зображень, знаходження об’єктів на зображенні, кластеризація регіонів зображень, розпізнавання символів тощо), розпізнавання мовлення, обробка та розуміння природньої мови, машинний переклад, знаходження помилок в тексті, знаходження ідентичності між документами, стеження за фондовими біржами та передбачення динаміки їх зміни та багато інших. Як було згадано раніше, люди щодня звертаються до застосувань на базі нейронних мереж, коли користуються голосовими асистентами, перекладачами, додатками для обробки зображень та соціальними мережами.

Для подальшого встановлення проблематики використання нейронних мереж для навчання систем, що виконуються на мобільних пристроях, необхідно познайомитися з їхньою будовою та принципами роботи. Один з ключових елементів нейронної мережі — нейрон. З його будовою можна ознайомитися на наступному рисунку.

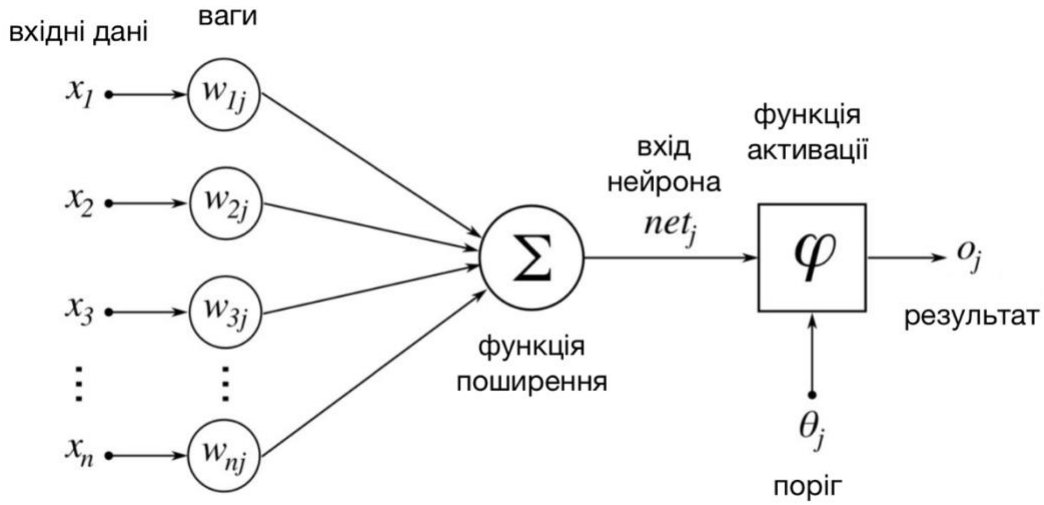

Рис. 1.1 Схема нейрона нейронної мережі [26]

Нейрон сполучений з іншими нейронами мережі та приймає на вхід перелік значень, кожне з яких має певний коефіцієнт (вагу), тобто ступінь впливу на результат обчислення. Ці значення надходять до функції поширення, що обчислює зважену суму всіх входів і повертає так званий вхід нейрона. Далі функція активації визначає, чи достатнім є значення входу, щоб передати подразнення далі, порівнюючи його з деяким порогом. Якщо поріг пройдено, нейрон сигналізує наступнику і процес повторюється.

В мережі нейрони організовані таким чином, що їх можливо розділити на так звані шари. Кожен шар складається з довільної кількості нейронів — це вирішується на стадії проєктування мережі і не може бути змінене в подальшому. Ефективність використання тієї чи іншої архітектури мережі дуже залежить від конкретної задачі і є досить поширеною темою для написання наукових статей та робіт. Шари бувають

трьох видів: вхідними, вихідними та прихованими. Нейрони вхідного шару не мають попередників і визначають модель взаємодії вхідних даних з усією мережею. На противагу, вихідні дані нейронів вихідного шару не передаються іншим нейронам; ці дані є результатом роботи всієї мережі. Приховані шари мають вхід і вихід і слугують винятково для обчислень даних в мережі.

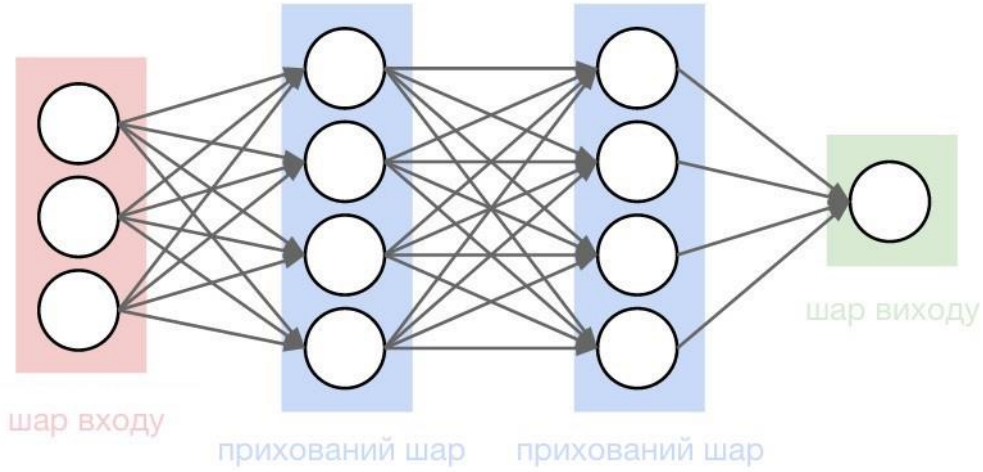

Рис. 1.2 Будова шарів нейронної мережі [26]

Також мережа складається з деяких гіперпараметрів. Це значення, що визначаються перед початком навчання і в його процесі ніяк не змінюються. До таких параметрів відносять кількість прихованих шарів мережі та кількість нейронів у кожному з них, функцію втрат, що формально визначає відстань між очікуваним та отриманим результатом та темп навчання — числовий коефіцієнт, що визначає розмір кроку кожної ітерації на зменшення показнику функції втрат.

Навчання нейронної мережі — це процес, під час якого вагові коефіцієнти зазнають змін, збільшуючи чи зменшуючи вплив певних властивостей (з англ. features) на результат, що повертає мережа. Перед першою ітерацією роботи мережі, усім вагам надають випадкових значень. Здавалося б, що можна ініціювати їх однаковими значеннями, як от 0 чи 1, але тут створюється небезпека, що модель буде вчитися повільніше, або ж не вчитиметься взагалі. З кожною ітерацією навчання результат кожного

входу надходить до функції втрат, яка допомагає визначити, як сильно модель помилилася у своїх припущеннях. Відповідно до показників функції, відбувається корекція вагових коефіцієнтів і процес навчання повторюється знову. Взагалі кажучи, для успішного тренування нейронної мережі необхідно виконати дуже багато ітерації — іноді ці числа вимірюються сотнями тисяч, а подекуди й більше. [4]

Враховуючи специфіку деяких задач, що вирішують нейронні мережі, на практиці часто доводиться мати справу з моделями, що оперують не скалярами у своїх обрахунках, а векторами чи навіть кількавимірними матрицями високих розмірностей. Наприклад, модель, що класифікує зображення і асоціює його з одним з двох класів, про яку було згадано раніше, постійно отримує на вхід матрицю, кожен елемент якої є пікселем зображення. Залежно від архітектури моделі, але в будь-якому випадку, вхідні дані зазнають великої кількості перетворень, що передбачає велику кількість операцій над матрицями. І це є однією з найбільших проблем щодо використання машинного навчання на пристроях з обмеженими ресурсами пам’яті, процесорного часу та ємності батареї. Адже такі процеси мають бути серйозно оптимізовані та в жодному разі не погіршувати досвіду користування пристроєм.

1.2. Технологія доповненої реальності

Доповнена реальність — це вдосконалена версія фізичної, дійсної реальності, на елементи якої накладаються створені комп’ютером звук, відео, графіка або дотикові відчуття [5]. В більш поширеному значенні цього терміну доповненою реальністю називають саме досвід взаємодії користувача з генерованою комп’ютером перцептивною інформацією. Іноді така інформація може характеризуватися кількома способами взаємодії з органами чуття, зокрема зоровими, слуховими, нюховими, дотиковими тощо.

Доповнена реальність певною мірою є наступником іншої технології, що має назву віртуальна реальність. Під час користування віртуальною реальністю, користувач не має жодної змоги бачити чи відчувати нічого, що відбувається довкола нього. Для користувача створюється окремий повністю віртуальний світ разом з інтерфейсом для керування цим світом. Зазвичай таким інтерфейсом є спеціальна гарнітура, що складається з непрозорих окулярів, обладнаних моніторами та навушниками та, іноді, додатковими пультами з гіроскопом. Очевидно, що такий підхід може бути дещо небезпечним і вимагає особливих і досить комфортних умов для реалізації для використання. По-перше, якщо людина користується гарнітурою в тісному чи людному приміщенні, вона ризикує зіткнутися зі своїм оточенням, пошкодити дорогий пристрій чи завдати шкоди самій собі. По-друге, існує ризик для людей з чутливим вестибулярним апаратом, адже події у віртуальному світі (наприклад, швидкий рух), ніяк не супроводжуються віддаччею на інші органи руху, окрім очей та вух, отож у деяких людей може виникати нудота чи запаморочення. По-третє, використання гарнітури для віртуальної реальності вимагає постійного перегляду відеоряду на дуже близькій відстані від очей. Це або сильно обмежує час використання пристрою, або загрожує стати причиною проблем з зором. До того ж, далеко не всі компанії, що виробляють гарнітури, піклуються про забезпечення достатньо високої роздільної здатності задля зниження ціни на свою продукцію. Хоча, заради справедливості варто сказати, що віртуальна реальність має величезний спектр застосувань у військовій справі, медицині, моделюванні, сфері розваг та інших і є досить популярною, отож постійно розвивається.

Види доповненої реальності можна класифікувати наступним чином.

Проєкційна доповнена реальність (англ. Projection-based) не потребує використання сторонніх пристроїв з боку користувача. Принцип технології полягає у проєктуванні зображення на реально поверхню і використання

численних сенсорів, що дають змогу користувачу взаємодіяти з ним. Взаємодія визначається через зміну відстані до поверхні проєктування, коли користувач, наприклад, дотикається до елементу взаємодії. Такий підхід наразі не є поширеним, проте він поступово знаходить свою сферу застосування. Наприклад, компанія Light Guide Systems розробила інструмент, що позбавляє підприємства необхідності випускати і зберігати громіздкі інструкції для персоналу. Натомість, робочі місця обладнані спеціальними приладами, що проєктують усі необхідні інструкції на поверхню робочого місця. [6]

Рис. 1.3 Проєкційна доповнена реальність [6]

Маркерна (англ. Marker-based) доповнена реальність, або ж розпізнавальна створює проєкцію віртуальних об’єктів на певні мітки (маркери), базуючись на алгоритмах комп’ютерного зору. Система визначає положення об’єкта-мітки та його кут повороту в двох чи трьох площинах і керується цими даними для зміни положення віртуального об’єкта. В більш простіших варіантах систем, поворот на площині маркера не впливає на відповідний поворот віртуальної моделі. Такий підхід використовується в багатьох мобільних та ігрових застосунках. Зокрема, згаданий вище Google-перекладач спроможний використовувати слова як маркери і проєктувати на їхнє місце перекладені іншою мовою. Ще у 2012

році Sony PlayStation Vita підтримувала цю технологію з використанням спеціальних карток, що використовувалися для запуску спеціальних ігор з доповненою реальністю. Також технологію використовують для побудови віртуальних виставок, а також у сфері освіти — наприклад для вивчення іноземних мов шляхом використання іншомовних слів як маркерів для об’єктів, що знайомі учням у побуті.

Рис. 1.4 Маркерна доповнена реальність

Безмаркерна (англ. Marker-less) доповнена реальність працює базуючись на положенні пристрою користувача у тривимірному просторі. Для цього використовуються GPS, акселерометр та гіроскоп пристрою. На такому принципі побудовані згадані вище додатки для віртуального приміряння одягу та меблів у кімнаті. Деякі додатки для навігації дають змогу використовувати камеру пристрою, через яку проєктується маршрут до точки призначення.

Рис. 1.5 Безмаркерна доповнена реальність [27]

Суперімпозиційна (англ. Superimposition) доповнена реальність працює за принципом заміни реальних об’єктів на віртуальні. Причому об’єкти можуть бути замінені повністю, або ж лише деякі їхні частини. Хоча принцип роботи технології і є дещо подібним до маркерної — все ж, без розпізнавання об’єктів не обійтися — суперімпозиційна доповнена реальність є концептуально більш складною. Адже «маркерами» можуть виступати не лише плоскі зображення, а й повноцінні об’єкти реального світу. Технологія активно використовується в розважальній індустрії та промисловості.

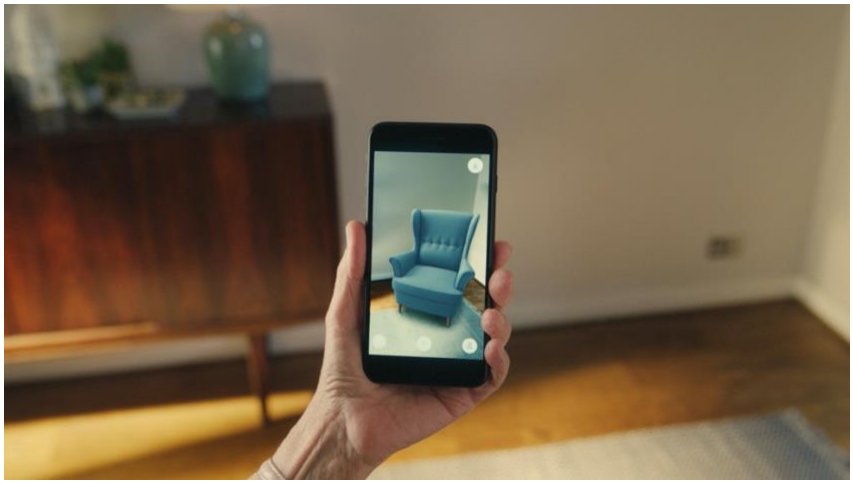

Рис. 6 Суперімпозиційна доповнена реальність [2]

Використання підходів доповненої реальності потребує постійного стеження за показниками камери, гіроскопа та акселерометра пристрою та постійного їх аналізу. Додаткової складності додає необхідність роботи з цими даними в реальному часі.

Перед початком роботи застосування, відбувається ініціювання ключових параметрів, таких як тривимірне положення пристрою у просторі (зазвичай, цей параметр набуває початкового нуль-значення), а також положення пристрою відносно власної системи координат. Під час роботи застосування, камера неперервно обробляє потік кадрів, що надходять застосуванню на подальшу обробку. Перед початком проєктування віртуальних об’єктів на сцену, користуючись показниками гіроскопа так акселерометра, відбувається калібрація початкового положення пристрою. Для цього деякі застосунки просять користувача встановити пристрій в бажане положення, яке буде вважатися стартовим, або ж пообертати його, чи провести його над поверхнею, на яку відбуватиметься проєктування, камерою донизу — це залежить від використаного підходу та рушія доповненої реальності. Щойно усі параметри проініційовано, відбувається синхронне порівняння показників датчиків. Для кожного кадру існує певне положення пристрою в різних системах координат (просторових та

відносно власних осей). Маючи частоту оновлення камери та показання датчиків для кожного кадру, можна встановити вектор переходу та обертання зображення для будь-яких двох сусідніх моментів часу. Таким чином, маючи проєкцію віртуального об’єкту в кадрі, можна змінювати його положення відповідно до цих векторів, виконавши відповідні перетворення. Загальні принципи одного з підходів описані у наступній публікації [7].

Звісно, описаний вище підхід є лише одним з відомих і не використовується усіма виробниками рушіїв доповненої реальності. Більше того, він не завжди успішно застосовний залежно від виду доповненої реальності, з яким працює розробник.

1.3. Висновки до розділу 1

У першому розділі було розглянуто та проаналізовано дві сучасні технології, що активно використовуються багатьма розробниками — машинне навчання та доповнена реальність. Було описано базові принципи роботи цих технологій, введено їхню класифікацію на підвиди а також наведено приклади успішного використання цих технологій у наявних програмованих додатках.

Розділ 2. Теоретичні відомості про Core ML та ARKit

2.1. Особливості використання машинного навчання для пристроїв з обмеженими ресурсами

Впродовж розвитку технології машинного навчання було виявлено, що існує багато повсякденних задач, в розв’язанні яких його досить зручно використовувати. Проте, враховуючи кількість складних обрахунків, якими оперують застосування, що використовують принципи машинного навчання — багатократне множення матриць — створити прикладне застосування технології для кінцевого користувача було неможливим.

Користувачі смартфонів не схильні користуватися додатками, що мають погану швидкодію. Навіть якщо додаток виконує якусь важливу задачу, але при цьому не реагує на дії користувача через складну фонову задачу, це значно погіршує досвід використання; ймовірніше за все, користувач вийде з додатка, або й видалить його. Також варто зауважити, що фонові задачі з обчисленнями, взагалі кажучи, не мають отримувати пріоритет під час розподілу процесорного часу. Адже на побудову адаптивного графічного інтерфейсу з його анімаціями та складними за структурою відображеннями, попри оптимізації з боку системи, витрачається досить багато ресурсів системи. Отож якість цього самого інтерфейсу дуже сильно залежить від загальної швидкодії пристрою. А «зовнішній вигляд і відчуття» додатку, як встановлено, має бути пріоритетом для кожного розробника.

Іншим проблемним місцем є батарея пристрою. Очевидно, що ефективне використання енергії в смартфоні ніяк не менш важливе, аніж швидкодія. Швидке збільшення кількості циклів зарядки погано впливає на максимальний об’єм акумулятора. Це, в свою чергу, змушує користуватися зовнішніми накопичувачами, аби лише телефон не відімкнувся впродовж дня. Очевидно, це значно погіршує комфорт від користування пристроєм.

До того ж, смартфони з надто використаною батареєю частіше потрапляють на звалище, що негативно впливає на навколишнє середовище.

Досить важливим фактором є використання мобільних даних для підключення до мережі. Тарифи для мобільного інтернету значно дорожчі від домашніх, що використовують Wi-Fi для підключення. Отож, якщо додаток вимагає частого підключення і використовує багато даних, користувач зможе повноцінно ним користуватися лише під час підключення до стаціонарної мережі, що не завжди зручно. Ба більше, часте використання мобільних даних витрачає більше заряду акумулятора.

Хоча використання віддаленого сервера для тренування моделі машинного навчання і спілкування з ним через якийсь протокол може звучати цікаво, проте з цим є певні проблеми. Перш за все це стосується систем, що вимагають постійної передачі потоку даних. Окрім очевидної низької ефективності використання інтернет-трафіку, таке підключення може легко стати здобиччю для кіберзлочинців. Припустимо, користувач пише листа в нотатнику, що перевіряє коректність введення тексту та аналізує його стиль (як, наприклад, Grammarly). Якби цей нотатник постійно зв’язувався з сервером і дані б перехопили, до зловмисників міг би потрапити вміст листа з усіма «чутливими» даними та особистою інформацією. Незалежно від контексту написаного, витік даних спровокував би обурення великої кількості користувачів додатку, що негативно вплинуло б на репутацію розробника.

Наостанок хотілося б зазначити, що будь-яка програмна технологія, щоб її легко було впровадити в існуючі чи нові додатки, повинна мати широкий спектр готових бібліотек, що надають високорівневий інтерфейс користування нею. Необхідність створювати таку бібліотеку з нуля може призвести до кадрових витрат, що для маленьких компаній є нерентабельним. Особливо, якщо йде мова про повноцінний фреймворк, що

вимагає постійної підтримки, стабільного графіку виходу версій, надійності та оптимізованості.

Ось чому програмна та апаратна підтримка технологічних гігантів сприяла стрімкому розвитку технології машинного навчання на мобільних пристроях. Вони не лише взяли на себе розробку високоякісних і продуктивних фреймворків, а й займаються апаратним вдосконаленням своїх пристроїв, оптимізовуючи їх безпосередньо як платформу для використання підходів машинного навчання. Більше того, на їхні ресурси ведеться постійна робота на дослідження технології машинного навчання вцілому, що також має позитивний вплив на ситуацію.

2.2. Використання фреймворку Core ML для побудови додатків iOS

На щорічній конференції для розробників Apple WWDC (Worldwide Developer Conference) було представлено кілька ключових нововведень, що сильно вплинули на розробку мобільних додатків для системи iOS і, непрямо, для інших систем. До цих нововведень належали також фреймворки для машинного навчання.

Інструменти, що були розроблені інженерами Apple, чудово вирішували усі проблеми, описані в підрозділі 2.1 і ось чому.

По-перше, будь-яке програмне забезпечення виробництва Apple перед презентацією і виходом проходить багато стадій оптимізації і вдосконалення. Завдяки цьому нівелювалися проблеми, пов’язані з енергоефективністю та ефективністю використання процесорного часу. Окрім програмних оптимізацій, відбулися також важливі апаратні. Так, починаючи з 2017 року чипи серії A (A11 Bionic і вище) об’єднані спеціальним компонентом Neural Engine. Він дозволяє дуже швидко й ефективно виконувати операції, що є поширеними для машинного навчання.

По-друге, завдяки тому, що стало можливим використовувати прийоми машинного навчання на пристроях, відпала й проблема постійного підключення до інтернету й надмірного використання трафіку. Ба більше, пристрої Apple славляться своєю захищеністю та складністю зламу, а відсутність необхідності підключення до мережі з метою використання технології машинного навчання лише зменшує кількість можливих вразливостей системи.

По-третє, нарешті була вирішена проблема наявності потужного інструментарію промислового рівня, яким могли б користуватися розробники і впроваджувати свіжі ідеї щодо машинного навчання в свої додатки. Важливим є й те, що цей інструментарій є доступним не лише для iPhone, а й для будь-якого пристрою виробництва Apple на будь-якій операційній системі — iOS, macOS, watchOS і, навіть, tvOS.

Core ML — це фреймворк, що дозволяє інтегрувати машинне навчання в будь-який додаток. Він є ключовою ланкою в архітектурі більш спеціалізовних фреймворків, що поєднує між собою низькорівневі і високорівневі інструменти для розробки.

Розглянемо більш детально архітектуру фреймворку.

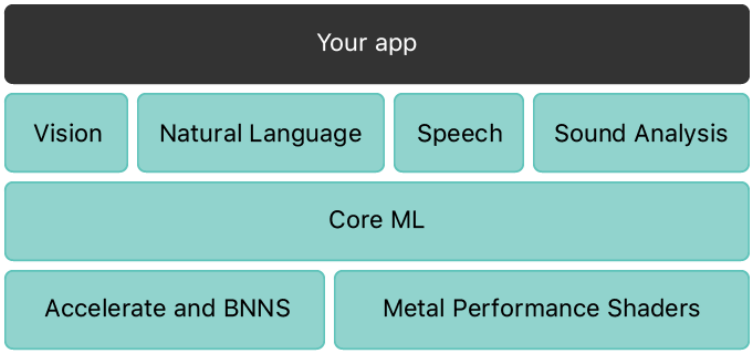

Рис. 2.7 Архітектура Core ML [11]

Accelerate — це фреймворк, що дозволяє ефективно виконувати складні обрахунки на процесорі, наприклад, ключові для машинного навчання множення матриць, використовуючи на максимум його

можливості з обрахунку векторів. Кожна операційна система має власну його реалізацію, що оптимізована для її архітектури [8].

BNNS (англ. basic neural network subroutines) — це набір функцій, що використовуються для побудови та запуску нейронних мереж, з використанням попередньо отриманих тренувальних даних. Важливим є те, що цей фреймворк не виконує тренування мереж. Його мета — забезпечити максимальну продуктивність виконанню мережі, що була попередньо натренована. У BNNS концептуально закладена робота з цілими шарами, а не окремими нейронами, фільтрами, що забезпечують попередню обробку вхідних даних до шару та функцією активації. BNNS містить функції для створення, використання та знищення трьох типів шарів.

1. Шар згортки (convolution layer) для кожного пікселя у вхідному зображенні, отримує цей піксель і його сусідів та поєднує їхні значення з ваговими коефіцієнтами з навчальних даних для обчислення відповідного пікселя на вихідному зображенні.
2. Шар пулінгу (pooling layer) створює зменшене вихідне зображення з вхідного, розбиваючи його на менші прямокутні зображення. Кожний піксель на виході є найбільшим, або ж середнім серед пікселів у відповідному зображенні. Цей шар не використовує навчальні дані.
3. Повністю з’єднаний шар приймає вектор на вхід. Цей вектор множиться на матрицю з вагами з навчальних даних. Результуючий вектор оновлюється функцією активації [9].

Metal Performance Shaders — це фреймворк, що містить набір оптимізованих обчислювальних і графічних шейдерів, які легко інтегрувати у відповідні додатки. Ці примітиви (з точки зору розпаралелювання) спеціально налаштовані з урахуванням особливостей усіх сімейств

графічних процесорів, задля забезпечення максимальної ефективності й швидкості роботи. Також використання фреймворку позбавляє розробника необхідність створювати шейдери власноруч [10].

Core ML побудований на базі описаних вище фреймворків. Можна виділити дві головні задачі, які виконує Core ML. Перша — абстрагування від низькорівневих задач та надання зручного інтерфейсу для впровадження методів машинного навчання в додатку, або ж для побудови більш високорівневих фреймворків. До цього також відноситься можливість портування моделей машинного навчання безпосередньо в код програми, з подальшою можливістю використання їх як звичайних класів. Друга — прийняття рішення, який фреймворк буде відповідати за обрахунки: або Accelerate з використанням головного процесора, або Metal Performance Shaders з використанням графічного процесора [11].

На базі Core ML побудовані ще чотири фреймворки: Vision, Natural Language, Speech та SoundAnalysis.

Vision використовується для виявлення обличчя та його рис, тексту, розпізнавання штрих-кодів, зіставлення зображень та відстеження різних рис. Vision також дозволяє використовувати самописні моделі Core ML для класифікації чи розпізнавання об’єктів на зображенні [12].

Natural Language дає змогу розпізнавати почерк, виконувати токенізацію, лематизацію, визначення частин мови та розпізнавання названих сутностей. Через цей фреймворк також можна працювати з власними моделями, або ж сторонніх розробників [13].

Фреймворк Speech використовують для розпізнавання розмовних слів у записаному аудіо або тому, що транслюється. Він надає можливість користуватися функціоналом, схожим до того, що використовує клавіатура під час перетворення звуку в текст, лише, звісно, без наявності клавіатури. Розпізнавання доступне для кількох мов, проте робота фреймворку

залежить і можлива лише за наявності прямого підключення до мережі. [14].

SoundAnalysis надає можливість аналізувати аудіо та виділяти з нього такі елементи, як, наприклад, сміх, оплески чи інші. Цей аналіз може застосовуватися до попередньо записаного аудіо, або ж під час живого транслювання. Під час роботи можна користуватися власними моделями, але обов'язково тими, що виконують класифікацію.[15].

Приємною особливістю фреймворку Core ML є те, що для нього створений потужний інструментарій, що дозволяє легко створювати моделі машинного навчання й одразу інтегрувати їх в мобільні додатки.

Create ML — це додаток для операційної системи macOS, який встановлюється безкоштовно. Він приймає дані у вигляді тексту, зображень чи таблиць, дозволяє розмітити їх відповідним чином та подати на вхід нейронній мережі, яка попередньо запрограмована. Перед початком роботи необхідно вказати, яка саме мережа буде використовуватися, адже існують моделі для роботою з текстом, звуком, зображеннями, таблицями та навіть активностями. Сам процес навчання прихований від користувача; неможливо також модифікувати мережу програмно. Єдний спосіб з нею взаємодіяти — подавати нові дані на вхід. По закінченню навчання, програма підготує до експорту файли .mlmodel, які можна імпортувати в середовище розробки Xcode і використовувати для розробки додатку з використанням Core ML [16].

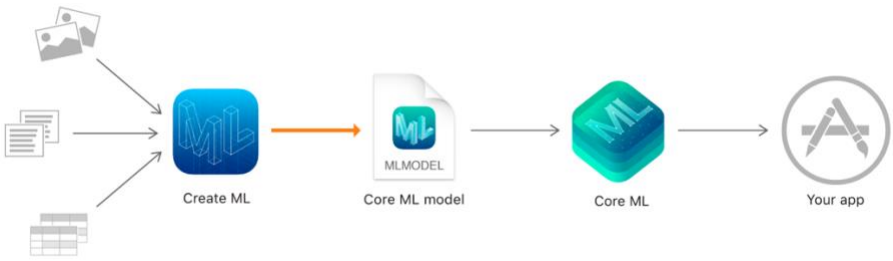

Рис. 2.8 Створення та використання моделі за допомогою Create ML [16]

Також був розроблений спосіб використання вже створених моделей машинного навчання з фреймворком Core ML. Для цього було створено Python-утиліту Core ML Tools. Вона доступна для встановлення через менеджер пакетів pip і існує у відкритому доступі, а також активно підтримується спільнотою. Утиліта дає змогу конвертувати моделі з форматів TensorFlow, Keras, Caffe, LIBSVB, scikit-learn, XGBoost, що є найпоширенішими у сфері машинного навчання.

2.2 Особливості застосування підходів доповненої реальності для мобільних пристроїв

Враховуючи, що технологія доповненої реальності значною мірою покладається на неперервний потік відео, можна зробити висновок, що деякі проблеми, що ускладнювали розвиток мобільної доповненої реальності, є спільними для проблем мобільного машинного навчання. А саме, це обчислювальна потужність, енергоефективність та відсутність відповідних фреймворків.

Як було сказано вище, виробники смартфонів та супровідного програмного забезпечення постійно працюють над покращенням технології машинного навчання, вирішуючи проблему енергоефективності та створюючи апаратну підтримку технології машинного навчання. Враховуючи, що концептуально технологія доповненої реальності частково, а подекуди й повністю, використовує ті самі інструменти для обрахунків, це надає їй потужного поштовху у розвитку.

З огляду на особливості роботи технології доповненої реальності, можна зробити висновок, що смартфони дуже добре підходять для використання їх як пристроїв для виконання відповідний AR-додатків.

По-перше, більшість сучасних смартфонів обладнані усіма необхідними датчиками для роботи з доповненою реальністю. Акселерометри та гіроскопи вже давно вбудовані в смартфони і за цей час

стали дуже точними. Більшість смартфонів мають якісну камеру з можливістю стабілізації, а також дисплей з високою роздільною здатністю.

По-друге, цінова категорія смартфонів значно нижча, аніж у профільних пристроїв для доповненої реальностіЗвісно, задля кращого досвіду користування рекомендують користуватися смартфонами з вищої цінової категорії, проте навіть вони виявляються значно дешевшими. Це пов’язано з тим, що доповнена реальність не досягла піку свого розвитку та популярності, а тому більшість таких пристроїв виготовляють на замовлення невеликими партіями для бізнесів, а не окремих користувачів. До того ж, смартфони є значно більш гнучкими у використанні і не зав’язані під два чи три AR-додатка.

По-третє, як було сказано вище, технологічні гіганти піклуються про те, щоб їхні пристрої відмінно підтримували популярні технології. Яскравим прикладом такого піклування є фреймворк виробництва Apple для роботи з доповненою реальністю на пристроях на базі iOS та iPadOS — ARKit.

2.4. Використання ARKit для створення додатків з доповненою реальністю

Разом з фреймоврками для машинного навчання, на WWDC 2017 була представлена перша версія рушія для доповненої реальності ARKit. Фреймворк дав змогу стороннім розробникам впроваджувати технологію доповненої реальності у власні додатки. Вже на момент релізу він надавав широкий інструментарій для реалізації як простих так і складних додатків. В подальших версіях відбувалося розширення сценаріїв використання, покращилася енергоефективність та додалися інші цікаві функції. Наразі доступна версія ARKit 3.5.

ARKit вирішує дуже складну задачу знаходження відповідності між дійсним і уявним простором. Сесії доповненої реальності з використанням рушія використовують підхід, що називається віртуально-інерційною

одометрією. Він полягає у комбінуванні аналізу показників датчиків руху та комп’ютерного бачення для відслідковування положення й орієнтації пристрою у просторі. Важливо звертати увагу на те, що якість доповнення реальності сильно залежить від освітлення та якості й деталізації зображення. Погане освітлення може спричинити помилки у знаходженні площин чи об’єктів — про це детальніше далі в цьому розділі —, а тому й погіршувати досвід користування. Причому йдеться не лише про низький рівень освітлення — надмірна кількість світла так само знижує деталізацію об’єктів і площин, що ускладнює або й унеможливлює коректну роботу рушія.

Однією з механік і головних функцій ARKit є аналіз та розуміння вмісту так званої сцени — простору, який є майданчиком для розміщення віртуальних об’єктів. Це може бути стіл, стіна, або ж ціла кімната чи подвір’я. За допомогою фреймворку можна знаходити площини за допомогою лише зображення та поступальних рухів смартфоном вздовж осей площини. Щойно площина буде знайдена, рушій повідомить про її розміри, віддаленість та кут нахилу — тобто усі необхідні дані для розміщення об’єктів та подальшою взаємодією з ними.

Вся інформація про відстеження сцени аналізується та передається користувачу. Оскільки відстеження сцени відбувається з одночасним використанням сенсорів для відстеження руху та камери, рух пристрою, навіть незначний, дуже сильно покращує якість роботи рушія. Проте, якщо рух є надто інтенсивним, може відбуватися «змазування» зображення а також десинхронізація показників камери й гіроскопа з акселерометром. У такому разі відбудеться порушення сцени — якщо на ній були присутні віртуальні об’єкти, їхні позиції можуть зміститися у просторі.

Також важливим моментом є правильне використання ресурсів для знаходження площини. Якщо з моменту початку виявлення площини пройшло небагато часу, її положення може бути неточним. Проте рушій

постійно проводить додаткові вимірювання для покращення точності результатів знаходження площини. Повільних рухів вздовж площини достатньо для покращення показників вимірювання. Проте, щойно точність стає задовільною, варто вручну вимикати цей процес, адже, по-перше існує ризик, що дрібні деталі площини після довгого аналізу негативно вплинуть на точність і по-друге, цей процес є досить енергозатратним [17].

Надзвичайно важливим нововведенням другої версії ARKit є можливість створювати віртуальні сесії для двох чи більше користувачів. Єдиною вимогою є наявність пристрою з iOS 12 чи новішою. Такі віртуальні сесії працюють за принципом Multipeer Connectivity: один учасник зв'язку створює та ініціює «карту світу» (ARWorldMap) та відправляє її іншому користувачу. Той, у свою чергу, приймає карту та отримує змогу бачити об'єкти в доповненій реальності на тих самих фізичних місцях, де їх встановив перший користувач. Сценарії використання цього підходу є дуже гнучкими. Передусім — це ігри, спільні презентації, віртуальні відео-конференції тощо [18].

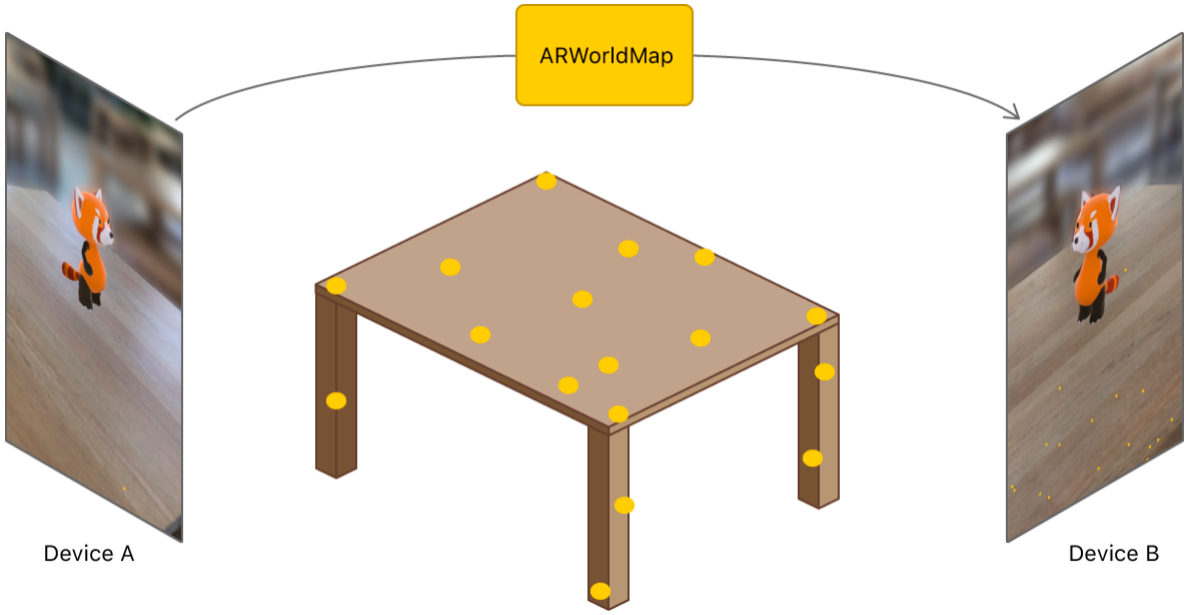

Рис. 2.9 Спільна AR-сесія двох користувачів [18]

«Хорошим тоном» з боку розробника є надання можливості продовжити в програмі з того ж місця, на якому роботу було закінчено. Це можу стосуватися налаштувань, положення відповідних елементів інтерфейсу тощо. Фреймворк ARKit слідує цьому принципу, адже усі сесії доповненої реальності, з певними обмеженнями, можна зберегти до файлу та відновити, навіть якщо додаток був закритий. Звісно, обмеження стосуються, в першу чергу, освітлення. Якщо сцена була створена вдень, а повторно відтворена вночі, то брак освітлення може змістити віртуальні об’єкти, або ж поступово прибрати їх зі сцени цілком.

Для розуміння роботи механізму відновлення сесії, необхідно розглянути діаграму її життєвого циклу.

Рис. 2.10 Діаграма життєвого циклу сесії доповненої реальності [19]

Як видно з діаграми (рис. 2.10), одразу після запуску існує часовий проміжок, де функціональність рушія зовсім недоступна, а з часом стає доступною, проте обмеженою. Це пояснюється особливістю будови рушія а також процесами, що виконуються перед ініціалізацією головних функцій, доступних користувачу. Якщо під час роботи рушія стаються певні перебої, наприклад механізм пошуку площини не оновлює результатів під час надходження нових даних, або ж глибинний пошук не повертає результатів, рушій також може входити до стану з обмеженими функціями.

У випадку переривання сесії (наприклад, якщо додаток було закрито), діаграма життєвого циклу є дещо складнішою:

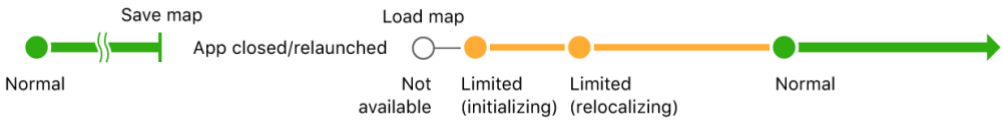

Рис. 2.11 Діаграма життєвого циклу ARKit під час переривання сесії [19]

Як видно, перед поверненням до нормального стану сесії відбувається три етапи. Спочатку рушій готується до процесу ініціалізації. Під час процесу ініціалізації відбувається зчитування файла, що зберігав сесію та створення відповідних об’єктів у пам’яті пристрою, а також сканування площини для встановлення об’єктів. Останнім етапом є розміщення об’єктів на визначеній площині. Після цього додаток починає працювати у звичному режимі. Згідно з документацією Apple, такий процес вартий впровадження лише за умови, що дані сесії є ключовими для збереження. Наприклад, користувач встановив віртуальні об’єкти на сцені і їхні положення важливо зберегти [19].

Останнім нововведенням, що стало доступним з версією ARKit 3 став RealityKit. Цей високорівневий фреймворк дуже тісно інтегрований з ARKit і надає змогу виконувати дуже ефективний і продуктивний рендеринг 3D-об’єктів та інтегрувати їх у навколишній простір. Ключовою відмінністю використання саме цього фреймворку є високий ступінь реалізму. Окрім дуже деталізованих моделей, він надає можливість впроваджувати звуки та анімовані об’єкти (навіть з симуляцією фізики), а також реагувати на зміни навколишнього середовища та синхронізувати все це між кількома пристроями.

2.5. Висновки до розділу 2

Обидва фреймворка — Core ML та ARKit — надають високоякісний і потужний функціонал для впровадження технологій машинного навчання та доповненої реальності в пристрої виробництва Apple. З огляду на енергоефективність, захищеність, продуктивність та зручність використання, ці фреймворки є чудовим, а можливо і найкращим вибором для створення сучасних iOS-додатків.

Розділ 3. Опис реалізації програмного продукту

3.1. Аналіз технічного завдання

3.2. Огляд засобів розробки та обґрунтування їх вибору

Для розробки клієнтської частини застосунку було використано стандартні засоби розробки для системи iOS.

Swift [20] — сучасна мова програмування, представлена у 2014 році на конференції WWDC. Головними особливостями мови є висока продуктивність, автоматичне керування пам’яттю з використанням механізму автоматичного обрахунку посилань (Automatic reference counting, ARC), а також наявність великої кількості синтаксичних конструкцій, що сильно покращують читабельність коду та, місцями, дозволяють значно зменшити його кількість. Мова є строго та статично типізованою, проте містить механізм автоматичного виведення типів з виразу. Тобто не обов’язково явно вказувати тип виразу — компілятор на етапі компіляції сам його встановить. Мова є мультипарадигмальною і підтримує об’єктно-орієнтований, функціональний та прототипно-орієнтований стилі програмування. Останній є винаходом розробників мови і дозволяє значно скоротити кількість класів в коді на користь структурам. Такий підхід вважається більш безпечним для роботи з пам’яттю а також дозволяє більш чітко групувати сутності в програмі і слідувати хорошому стилю програмування.

UIKit [21] — фреймворк, за допомогою якого будують графічний інтерфейс додатків для iOS. Донедавна він був єдиною альтернативою, проте у 2019 році Apple представити SwiftUI [22] — новий фреймворк, що дозволяє краще структурувати код інтерфейсу. Проте, вибір на користь саме UIKit пояснюється тим, що SwiftUI підтримується лише з 13 версії операційної системи, а отже кількість пристроїв, які могли б запустити

додаток є дещо меншою. Також UIKit досі є чудовим фреймворком і дозволяє будувати інтерфейси будь-якої складності.

Alamofire [23] — бібліотека для Swift, що дозволяє легко обробляти HTTP-запити. Бібліотека є безкоштовною та наявна у відкритому доступі, а також має велику кількість співавторів та користувачів, що постійно покращують її ефективність. Наразі бібліотека немає достойних альтернатив.

Розробка серверної частини виконана з використанням фреймворку Ruby on Rails.

Ruby on Rails [24] — популярний безкоштовний фреймворк, що за час існування став дуже потужним інструментом для веб-розробки. Окрім базового функціоналу, до фреймворку можна долучати велику кількість модулів, створених спільнотою за допомогою менеджера пакетів gem.

PostgreSQL [25] — база даних з відкритим кодом, що легко інтегрується у будь-які застосування. Переважно це пов’язано з тим, що спільнота розробників постійно створює та оновлює спеціальні модулі, що дозволяють користуватися базою даних у різних програмних середовищах. База даних є дуже продуктивною та активно використовується великими компаніями для своїх продуктів.

3.3. Опис основних підходів до розробки

Система побудована з використанням клієнт-серверної архітектури. У даному випадку така архітектура передбачає наявність одного монолітного застосунку-сервера, що обробляє вхідні запити та генерує відповіді та багатьох клієнтів, що є додатками для операційної системи iOS. Серверна частина побудована з використанням фреймворку Ruby on Rails. Це потужний фреймворк, що дозволяє будувати застосування з високою пропускною здатністю, а також дозволяє програмісту фокусуватися на написанні головної логіки. Адже велика частина «рутинної роботи», як от

написання SQL-запитів, створення маршрутизації та реалізація прав доступу, залишається за лаштунками і виконується фреймворком автоматично.

Особливістю архітектури є універсальність сервера. Можна побудувати додатки для інших операційних систем, або навіть створити веб-сайт, проте сторону сервера можна буде взагалі не змінювати. Клієнт та сервер взаємодіють через протокол HTTP — hypertext transfer protocol (з англ. — протокол для передачі гіпертексту). У роботі системи використовуються GET запити для отримання даних про сутностей в базі даних, POST — для створення нових, PUT — для оновлення та DELETE — для видалення. Після відправлення кожного запиту клієнтові повертається JSON-рядок, що є результатом виконання запиту. Таким чином також відбувається сповіщення про ті чи інакші помилки, що виникають під час його обробки.

Клієнт та сервер побудовані з дотриманням парадигми ООП. Усі дані чітко погруповані по класах, які належать тим чи іншим модулям. Також для створення клієнта та сервера був використаний шаблон проєктування MVC — Model-View-Controller (з англ. — модель, відображення, контролер). Цей підхід передбачає класифікацію усіх сутностей в програмі на три відповідні категорії. Модель відповідає за представлення даних з бази даних та основну бізнес-логіку застосунку. Відображення — це компоненти, що характеризують дані, які виходять «назовні»; у випадку сервера — це класи, які відправляють JSON у відповідь на запити; для клієнта — це файли інтерфейсу користувача. Контролер займається «зв’язуванням» моделі та відображення — форматує дані до належного вигляду, обробляє події всередині програми тощо.

Окрему увагу варто звернути на шаблон «делегат», який дуже широко використовується у частині клієнта. Він дає змогу певному компоненту сповіщати про якісь події, які виникають під час його роботи. Наприклад,

користувач натиснув на певну клітинку таблиці. Щоб отримати доступ до таких сповіщень, необхідно реалізувати відповідний протокол для класу, який хоче отримувати ці самі сповіщення. Після цього з’являється доступ до перезапису методів, як, наприклад didSelectRowAtIndexPath. У них можна реалізувати будь-яку потрібну логіку та реагувати на відповідні події так, як цього вимагає завдання.

Обидва фреймворки — Ruby on Rails для Ruby та UIKit для Swift надають потужну підтримку для розробника, що дозволяє фокусуватися переважно на побудові бізнес-логіки. Наприклад, UIKit містить величезний стандартний набір різних елементів графічного інтерфейсу, анімацій, засобів для навігації між екранами та відслідковування дій користувача додатку. Також варто відмітити, що керування пам’яттю в додатку відбувається автоматично, отож ймовірність витоку пам’яті в додатку є надзвичайно низькою.

3.4. Опис даних додатку та його інтерфейсу

Одним з головних класів додатку є AppSession, який слідкує за статусом автентифікації користувача. Він має два стани, що описані властивістю state: користувач або авторизований, або ні. Якщо. Користувач авторизований, ми можемо доступитися до його даних з будь-якої частини додатку завдяки шаблону «одинак». Також цей клас зберігає дані про сесію користувача у вигляді JWT-токена та надає до них доступ в усій програмі. Також до цього класу винесені методи авторизації, реєстрації та оновлення даних користувача.

```swift
class AppSession {
    enum State {
        case loggedOut
        case loggedIn(User)
    }

    private(set) var state: State
    static let shared = AppSession()


    func logIn(withUsername username: String, password: String, completion: ((Result<User, AFError>) → Void)?) {
        APIClient.login(login: username, password: password) { [weak self] result in
            guard let self = self else { return }
            switch result {
            case .success(let user):
                self.state = .loggedIn(user)
                self.updateUserSession()
                completion?(.success(user))
            case .failure(let error):
                completion?(.failure(error))
            }
        }
    }

    func signUp(withName name: String, username: String, email: String, password: String, isOrganizer: Bool, completion: ((Result<User, AFError>) → Void)?) {
        APIClient.signUp(name: name, username: username, password: password, isOrganizer: isOrganizer, email: email) { [weak self] result in
            guard let self = self else { return }
            switch result {
            case .success(let user):
                self.state = .loggedIn(user)
                self.updateUserSession()
                completion?(.success(user))
            case .failure(let error):
                completion?(.failure(error))
            }
        }
    }
```

Лістинг 3.1 Клас AppSession

Іншим важливим компонентом є структура APIClient, що дає змогу звертатися до сервера через прописані HTTP-запити. Взагалі, він є лише загальнодоступним інтерфейсом для роботи з сервером і частиною складної архітектури для роботи у мережі. Клас має статичний метод request, що приймає HTTP-маршрут у вигляді протоколу EndPointType та функцію, що буде виконана щойно надійде відповідь. Всередині функції відбувається декодування JSON-рядка в одну з п’яти моделей додатку — User, Expo, ARModel, UserToExpo, Comment для подальшого використання в програмі. Ці моделі є прикладом частини Model, що описана раніше в цьому розділі. Статичний метод request використовується як основа для усіх інших методів, що відправляють запити в усьому додатку. Взагалі кажучи, інші методи — лише виклики request з певними параметрами маршрутів.

```
@discardableResult
private static func request<ResponseType: Decodable>(route: EndPointType,
                                                     decoder: JSONDecoder = JSONDecoder(),
                                                     completion: @escaping (Result<ResponseType, AFError>) → Void) → DataRequest {
    decoder.dateDecodingStrategy = .formatted(.iso8601Full)
    return AF.request(route)
        .validate(statusCode: 200 ..< 600)
        .responseDecodable(decoder: decoder) { (response: DataResponse<ResponseType, AFError>) in
            debugPrint(response)
            if let auth = response.response?.allHeaderFields[HTTPHeaderField.auth.rawValue] as? String {
                UserDefaults.standard.setValue(value: auth, forKey: .jwt)
            }
            completion(response.result)
        }
}
```

Лістинг 3.2 Метод request

Протокол EndPointType має властивості baseURL: URL — базова адреса для доступу до сервера; method: HTTPMethod — набуває чотирьох значень в програмі: .get, .post, .put, .delete; path: String — безпосередні маршрути до різних енд-поінтів; task: HTTPTask — конфігураційна властивість, що дає змогу визначити, як обробляти той чи інший запит. Наприклад, з додатковими параметрами, заголовками чи без, або ж чи потрібна додаткова авторизація для виконання запиту. За допомогою розширення протоколу реалізований метод asURLRequest, що дає змоугу перетворити об’єкт типу EndPointType у повноцінний об’єкт типу URLRequest з можливістю його відправки на сервер.

```
protocol EndPointType: URLRequestConvertible {
    var baseURL: URL { get }
    var method: HTTPMethod { get }
    var path: String { get }
    var task: HTTPTask { get }
}
```

Лістинг 3.3 Протокол EndPointType

Для кожної моделі створена відповідна структура, що реалізовує EndPointType. Тут прописані усі HTTP-маршрути з відповідними типами запитів. Також тут задаються параметри, які необхідно додати до запиту для отримання правильної відповіді. Параметри бувають двох типів — параметри «тіла» (у вигляді JSON) та URL-параметри, що задаються безпосередньо у рядку запита. Для кодування параметрів створені дві

структури, що реалізують протокол ParameterEncoder з єдиним методом encode — JSONParameterEncoder та URLParamterEncoder.

Окрім частини, що працює з сервером додаток, очевидно, містить графічну частину. Тут вона представлена у вигляді двох типів сутностей. Перший — файл Storyboard, в якому за допомогою графічного редагування можна створювати елементи інтерфейсу додатку та файли розширень UIView, за допомогою яких описується деяка додаткова поведінка класів відображень, як-от наявність додаткових елементів в тому чи іншому компоненті вікна.

На Storyboard знаходиться граф, вершинами якого є екрани застосунку, а ребрами — об’єкти з’єднань (segue), які дозволяють не лише вказати системі, куди має відбуватися наступний перехід, а й передавати важливі дані з одного екрану на інший. Щоб досягнути такої поведінки необхідно у дочірньому класі ViewController реалізувати метод prepareForSegue. Його буде викликано одразу перед тим, як відбудуться перехід від одного екрану до іншого. В цьому методі необхідно отримати екземпляр класу-наступника (segue.destination) та присвоїти відповідним властивостям потрібні значення. Зазвичай необхідно використати механізм приведення типів, адже destination має тип ViewController, якому невідомо нічого про додаткову логіку, яку розробник закладає у свої сутності в програмі.

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    switch segueIdentifier(for: segue) {
    case .editProfile:
        guard case .loggedIn(let user) = session.state else { fatalError("user not found") }
        let destination = segue.destination as! ProfileViewController
        destination.user = user
    default:
        break
    }
}
```

Лістинг 3.4 Приклад передачі даних між екранами

Кожному екрану в додатку відповідає клас, який наслідує ViewController. У таких класах відбувається обробка дій, наприклад — натискання на кнопку чи введення тексту у текстове поле, створення логіки для навігації до інших частин додатку а також відображення моделей застосунку, тобто приведення їх до такого вигляду, щоб їхній вміст можна було відобразити як текст, зображення чи інші дані, які може побачити користувач.

Ключовим елементом додатку є клас ARViewController. Він відповідає за відображення екрану сцени SCNSceneView, на яку відбувається проєктування 3D-моделей на відповідні маркери. Перед ініціалізацією класу відбувається завантаження усіх моделей з відповідними маркерами. Після цього за допомогою об’єкту класу ARImageTrackingConfiguration задаються так звані tracking images (англ. — щображення для відслідковування) — саме це і є тими маркерами, на які буде відбуватися проєктування. За допомогою метода делегату ARSCNViewDelegate renderer(_:, nodeFor:) -> SCNNode до кожної моделі, які, до слова, зберігаються у спеціальному форматі .scn, прив’язується відповідний маркер. В процесі роботи ARViewController відбувається розпізнавання зображення, пошук відповідної 3D-моделі та, власне, проєктування для подальшого перегляду з боку користувача.

```
extension ARViewController: ARSCNViewDelegate {
    func renderer(_ renderer: SCNSceneRenderer, nodeFor anchor: ARAnchor) -> SCNNode? {
        let node = SCNNode()
        guard
            let anchor = anchor as? ARImageAnchor,
            let modelNode = createModel(for: anchor.referenceImage)
        else { return nil }

        let overlayPlane = SCNPlane(width: anchor.referenceImage.physicalSize.width,
                                    height: anchor.referenceImage.physicalSize.height)
        overlayPlane.firstMaterial?.diffuse.contents = UIColor(white: 1, alpha: 0.7)
        let markerNode = SCNNode(geometry: overlayPlane)
        markerNode.eulerAngles.x = -.pi / 2

        node.addChildNode(markerNode)
        node.addChildNode(modelNode)

        return node
    }
```

Лістинг 3.5 Розширення класу ARViewController та метод renderer

3.5. Тестування додатку

Точкою входу у додаток є процедура автентифікації. Для користувачів, що мають обліковий запис в системі необхідно ввести свої логін та пароль. Для нових користувачів необхідно спочатку створити обліковий запис з такими даними: повне ім’я, унікальний в системі логін, унікальна в системі електронна пошта, пароль. Також потрібно вказати, чи бажає користувач створювати власні виставки, чи буде лише глядачем. Після створення облікового запису користувач повинен увійти зі своїми даними для користування. Також варто зазначити, що про успішну реєстрацію користувача буде повідомлено відповідним листом на електронну пошту.

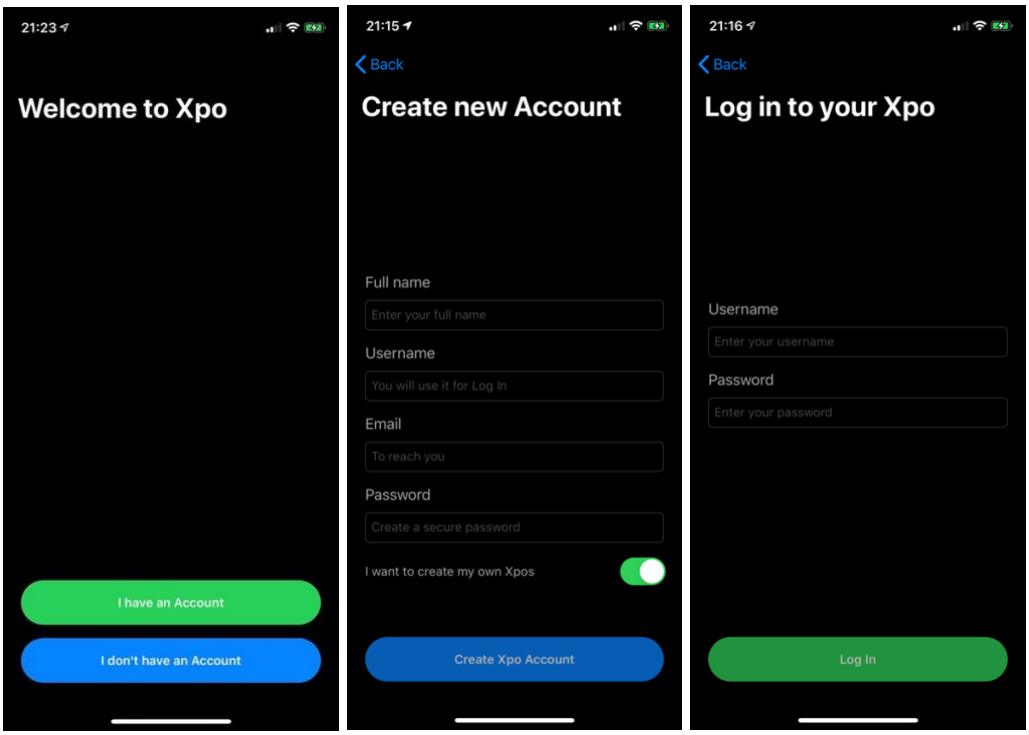

Рис. 3.1 Авторизація та реєстрація

Після успішної авторизації користувач потрапляє на головний екран, де відображається список усіх виставок, створених усіма користувачами. Кожна виставка містить таку інформацію: назва, опис, час початку та кінця,

ім’я організатора, кількість переглядів та вподобань. Також кожна «картка» виставки містить мініатюрне титульне зображення.

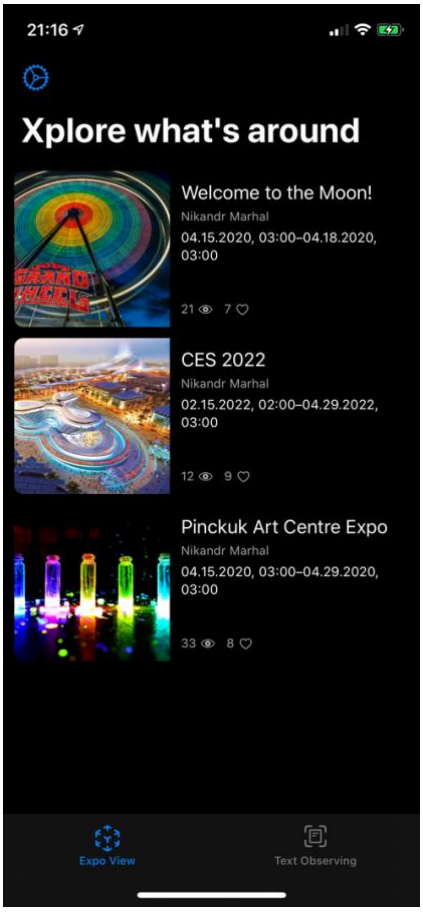

Рис. 3.2 Головний екран

При натисканні на будь-яку з «карток» відкриється екран виставки. На екрані виставки можна побачити більш детальний опис та іншу інформацію, збільшене титульне зображення. Під час відкриття виставки, у користувача та конкретної виставки автоматично створюється зв’язок в базі даних. Після цього можна написати коментар після натиску відповідної кнопки, натиснути кнопку «подобається», або ж прибрати попереднє вподобання. Важливим елементом цього екрану є кнопка взаємодії з виставкою. Після першого натискання на неї, усі 3D-моделі та маркери, до яких вони приєднані почнуть завантажуватися з віддаленого сервера на пристрій користувача. Про статус завантаження повідомить індикатор

завантаження, що з’явиться над кнопкою. Щойно завантаження буде завершене, користувач зможе відкрити екран перегляду AR виставки і спостерігати моделі при наведенні камери на відповідні маркери.

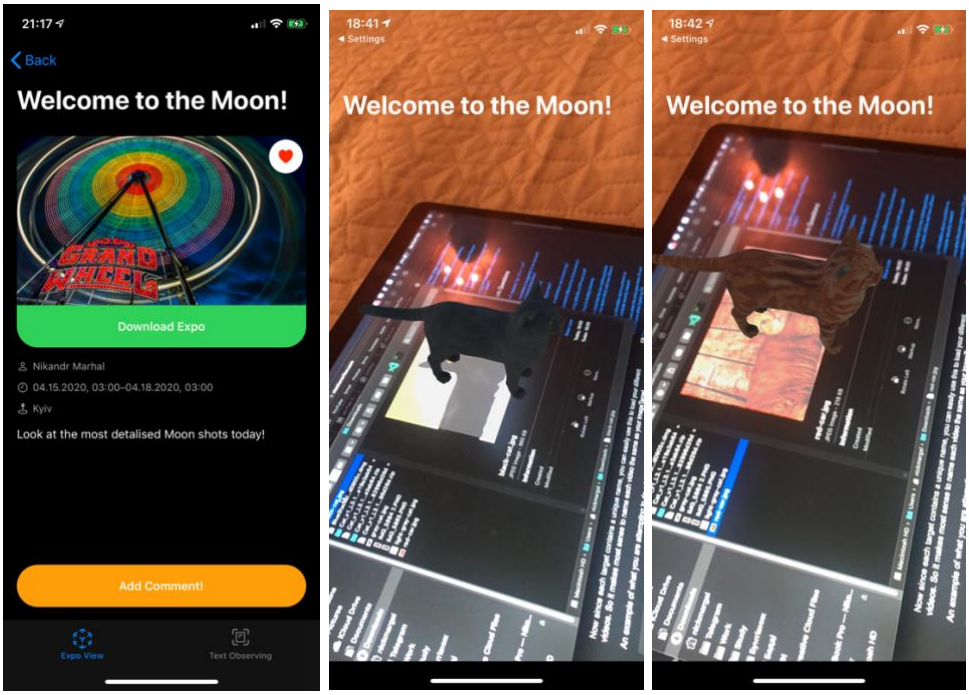

Рис 3.3 Екран перегляду виставки

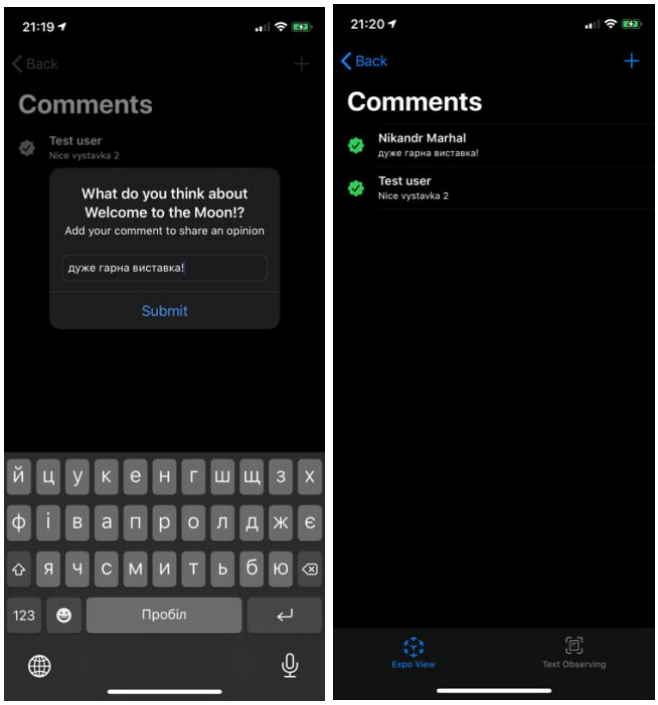

Рис 3.4 Екран коментарів

Також з головного екрану можна потрапити на екран налаштувань, натиснувши на кнопку у верхньому лівому кутку. На екрані, залежно від

ролі користувача, розташовані дві або три кнопки. Для звичайного користувача — «профіль», де можна змінити дані, вказані при реєстрації та «вийти» для завершення сесії та повернення на екран автентифікації. Для користувачів, що є організаторами також є екран «мої виставки». На ньому можна переглянути усі виставки, що були створені цим користувачем. При натисканні на кожну з них відкриється екран редагування з можливістю змінити дані про назву, опис, розташування, дату початку і кінця. Також на екрані з усіма виставками користувача можна створити нову виставку. Разом з тим необхідно з файлів на пристрої вказати пари «маркер – модель», які будуть завантажені на віддалений сервер.

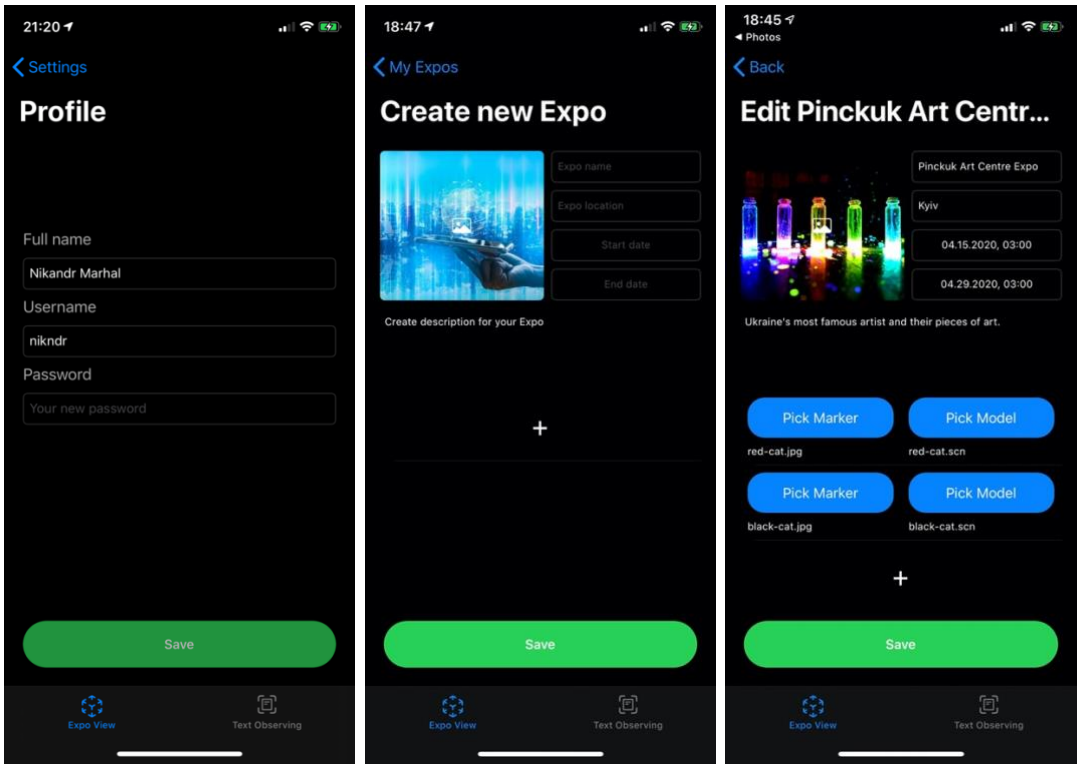

Рис 3.5 Профіль, екран створення та редагування виставки

Висновки

Отож, поставлені задачі були виконані. В ході роботи було проведено аналіз технологій машинного навчання та доповненої реальності в сучасних реаліях. Також було проаналізовано дві сучасні імплементації цих технологій — Core ML та ARKit з точки зору особливостей їхньої реалізації та можливостей. Також було детально проаналізовано структуру фреймворків та описані основні підходи й засоби до їхнього використання.

З іншого боку, під час виконання практичної частини роботи були використані не лише сучасні технології, а й основні методології та підходи для розробки. Зокрема, були використані підходи об’єктно-орієнтованого програмування та декілька ключових шаблонів проєктування, що є невід’ємною частиною якісної розробки для мобільних пристроїв.

В результаті був створений клієнт-серверний застосунок з можливістю створення, перегляду та редагування інтерактивних виставок в доповненій реальності. З додаткового функціоналу, без якого важко уявити подібний застосунок — авторизація та реєстрація, можливість змінювати персональні дані та надавати зворотній зв’язок організаторам виставки через вподобання та коментарі.

Список використаної літератури

1. Grib — World’s first 3D modelling software for Augmented Reality [Електронний ресурс] https://grib3d.com
2. IKEA Place [Електронний ресурс] https://apps.apple.com/au/app/ikea-place/id1279244498
3. Wanna Kicks [Електронний ресурс] https://apps.apple.com/app/id1444049305
4. Christopher M. Bishop — Pattern Recognition and Machine Learning — Springer: C. 225-284
5. Patrick Schueffel — The consice fintech compendium
6. Projector based augmented reality [Електронний ресурс] https://lightguidesys.com/blog/projector-based-augmented-reality-new-form-enterprise-ar/
7. Guo Hong-jie, Du Fu-zhou — Real-time Projection Method for Augmented Reality Assisted Assembley, 2018
8. Accelerate [Електронний ресурс] https://developer.apple.com/documentation/accelerate
9. BNNS [Електронний ресурс] https://developer.apple.com/documentation/accelerate/bnns
10. Metal performance shaders [Електронний ресурс] https://developer.apple.com/documentation/metalperformanceshaders
11. Core ML [Електронний ресурс] https://developer.apple.com/documentation/coreml
12. Vision [Електронний ресурс] https://developer.apple.com/documentation/vision
13. Natural Language [Електронний ресурс] https://developer.apple.com/documentation/naturallanguage

14. Speech [Електронний ресурс]
https://developer.apple.com/documentation/speech
15. Sound Analysis [Електронний ресурс]
https://developer.apple.com/documentation/soundanalysis
16. Create ML [Електронний ресурс]
https://developer.apple.com/machine-learning/create-ml/
https://developer.apple.com/documentation/createml
17. World Tracking [Електронний ресурс]
https://developer.apple.com/documentation/arkit/world_tracking
18. Multi-user experience [Електронний ресурс]
https://developer.apple.com/documentation/arkit/creating_a_multiuser_ar_experience
19. Managing session lifecycle and tracking quality [Електронний ресурс]
https://developer.apple.com/documentation/arkit/managing_session_lifecycle_and_tracking_quality
20. Swift programming language [Електронний ресурс]
https://swift.org
21. UIKit Framework [Електронний ресурс]
https://developer.apple.com/documentation/uikit
22. SwiftUI [Електронний ресурс]
https://developer.apple.com/xcode/swiftui/
23. Alamofire [Електронний ресурс]
https://github.com/Alamofire/Alamofire
24. Ruby on Rails [Електронний ресурс]
https://rubyonrails.org
25. PostgreSQL [Електронний ресурс]
https://www.postgresql.org

26. Neural networks [Електронний ресурс] https://towardsdatascience.com/neural-networks-for-beginners-by-beginners-6bfc002e13a2
27. Augmented Reality vs Virtual Reality [Електронний ресурс] https://www.feedsfloor.com/technology/augmented-reality-vs-virtual-reality-choose-one-right-mobile-app-development-company

Додаток А. Код сесії користувача

```
class AppSession {
   enum State {
      case loggedOut
      case loggedIn(User)
   }

   private(set) var state: State
   static let shared = AppSession()


   func logIn(withUsername username: String, password: String, completion: ((Result<User, AFError>) ->
Void)?) {
      APIClient.login(login: username, password: password) { [weak self] result in
         guard let self = self else { return }
         switch result {
         case .success(let user):
            self.state = .loggedIn(user)
            self.updateUserSession()
            completion?(.success(user))
         case .failure(let error):
            completion?(.failure(error))
         }
      }
   }

   func signUp(withName name: String, username: String, email: String, password: String, isOrganizer: Bool,
completion: ((Result<User, AFError>) -> Void)?) {
      APIClient.signUp(name: name, username: username, password: password, isOrganizer: isOrganizer, email:
email) { [weak self] result in
         guard let self = self else { return }
         switch result {
         case .success(let user):
            self.state = .loggedIn(user)
            self.updateUserSession()
            completion?(.success(user))
         case .failure(let error):
            completion?(.failure(error))
         }
      }
   }

   func logOut(completion: (() -> Void)?) {
      state = .loggedOut
      updateUserSession()
      completion?()
   }

   func updateUser(newName: String?, newLogin: String?, newPassword: String?, completion: ((Result<User,
AFError>) -> Void)?) {
      if case .loggedIn(let user) = state {
         APIClient.updateUser(login: user.login, newLogin: newLogin, newName: newName, newPassword:
newPassword) { [weak self] result in
            guard let self = self else { return }
```

```
            switch result {
            case .success(let user):
               self.state = .loggedIn(user)
               self.updateUserSession()
               completion?(.success(user))
            case .failure(let error):
               completion?(.failure(error))
            }
         }
      }
   }

   func synchronize(completion: @escaping () -> Void) {
      guard case .loggedIn(let user) = state else { return }
      APIClient.getUser(login: user.login) { [weak self] response in
         guard let self = self else { return }
         switch response {
         case .success(let user):
            self.state = .loggedIn(user)
         case .failure(let error):
            debugPrint(error)
            // TODO: maybe logout?
         }
         completion()
      }
   }

   private func updateUserSession() {
      let defaults = UserDefaults.standard
      switch state {
      case .loggedIn(let user):
         defaults.setValue(value: user.id, forKey: .userID)
         defaults.setValue(value: user.name, forKey: .name)
         defaults.setValue(value: user.login, forKey: .username)
         defaults.setValue(value: user.email, forKey: .email)
         defaults.setValue(value: user.isOrganizer, forKey: .isOrganizer)
      case .loggedOut:
         defaults.setValue(value: nil, forKey: .userID)
         defaults.setValue(value: nil, forKey: .name)
         defaults.setValue(value: nil, forKey: .username)
         defaults.setValue(value: nil, forKey: .email)
         defaults.setValue(value: nil, forKey: .isOrganizer)
         defaults.setValue(value: nil, forKey: .jwt)
      }
   }

   private init() {
      self.state = .loggedOut
   }
}
```

Додаток Б. Код реалізації перегляду доповненої реальності

```
import ARKit
import SceneKit
import UIKit

struct SCNResources {
   static let rootAssets = "art.scnassets"
   static let basicSceneName = "Scene"
   static let sceneExtension = ".scn"

   static let basicScenePath = fullPath(for: SCNResources.basicSceneName)

   static func fullPath(for resource: String, with fileExtension: String? = SCNResources.sceneExtension) -> String
{
      return "\(SCNResources.rootAssets)/\(resource)\(fileExtension ?? "")"
   }
}

class ARViewController: UIViewController {
   // MARK: - Properties

   var expo: Expo!

   // MARK: - Outlets

   @IBOutlet var sceneView: ARSCNView!

   // MARK: - Actions

   // MARK: - Lifecycle methods

   override func viewDidLoad() {
      super.viewDidLoad()
      configureUIElements()

      sceneView.delegate = self

      let scene = SCNScene(named: "art.scnassets/Scene.scn")!

      sceneView.scene = scene
   }

   override func viewWillAppear(_ animated: Bool) {
      super.viewWillAppear(animated)

      let configuration = ARImageTrackingConfiguration()

      guard let trackingImages = ARReferenceImage.referenceImages(inGroupNamed: "ARRes", bundle:
Bundle.main) else {
         debugPrint("no reference images found")
         return
      }

      configuration.trackingImages = trackingImages
```

```
        sceneView.session.run(configuration)
    }

    // MARK: - UI configuration

    /// Put your custom UI code here:
    /// rounded corners, shadows, corders etc.
    func configureUIElements() {
        navigationItem.title = expo.name
    }

    // MARK: - Navigation

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {}
}

extension ARViewController: ARSCNViewDelegate {
    func renderer(_ renderer: SCNSceneRenderer, nodeFor anchor: ARAnchor) -> SCNNode? {
        let node = SCNNode()
        guard
            let anchor = anchor as? ARImageAnchor,
            let modelNode = createModel(for: anchor.referenceImage)
        else { return nil }

        let overlayPlane = SCNPlane(width: anchor.referenceImage.physicalSize.width,
                                    height: anchor.referenceImage.physicalSize.height)
        overlayPlane.firstMaterial?.diffuse.contents = UIColor(white: 1, alpha: 0.7)
        let markerNode = SCNNode(geometry: overlayPlane)
        markerNode.eulerAngles.x = -.pi / 2

        node.addChildNode(markerNode)
        node.addChildNode(modelNode)

        return node
    }

    func createModel(for reference: ARReferenceImage) -> SCNNode? {
        guard
            let sceneName = sceneName(for: reference),
            let modelScene = SCNScene(named: sceneName),
            let modelNode = modelScene.rootNode.childNodes.first
        else { return nil }

        modelNode.position = SCNVector3Zero
        return modelNode
    }

    func sceneName(for reference: ARReferenceImage) -> String? {
        expo.arModels.filter { $0.markerFilePath == reference.name }.first?.modelFilePath
    }
}
```