АНОТАЦІЯ

Магістерська робота «Розробка переглядача зображень з можливістю керування жестами на платформу iOS для мультимедійного простору» Коцулим Валерій Євгенійович, Тернопільський національний технічний університет імені І. Пулюя, факультет комп'ютерно–інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПм–61, Тернопіль, 2019.

Пояснювальна записка містить: 102 с. 18 рис., 3 табл., 4 дод..

Метою роботи є розробка мобільного додатку та дослідження використання штучного інтелекту в повсякденному житті та задачах.

В ході роботи досліджено та проаналізовано предметну область, визначено ключовий функціонал додатку.

Розроблена система написана на мові програмування Swift.

Ключові слова: ФРЕЙМВОРК, SWIFT, ШТУЧНИЙ ІНТЕЛЕКТ, ЖИТТЄВИЙ ЦИКЛ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, МОБІЛЬНІ ДОДАТКИ.

ABSTRACT

Master thesis «Development of image viewer with gesture control option for a platform iOS for multimedia field» Kotsulym Valerii, I. Pulyu Ternopil National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, SPM–61 Group, Ternopil, 2019 .

The explanatory note contains: 102 p. 18 Figure, 3 Table, 4 Add ..

The purpose of the work is to develop mobile application and researching of usage AI for everyday usage.

In the course of the work the subject area was researched and analyzed, key system specifications were identified.

The system is written in Swift and JavaScript programming language.

Keywords: FRAMEWORK, SWIFT, ARITIFICIAL INTELLIGENCE, SOFTWARE DEVELOPMENT LIFE CYCLE, MOBILE APPLICATIONS

ВСТУП 7
1. ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ 9
1.1 Аналіз вимог до програмної системи 9
1.1.1 Аналіз предметної області 9
1.1.1.1 Штучний інтелект 13
1.1.1.2 Машинне навчання 15
1.1.1.3 Нейронні мережі 16
1.1.1.4 Мобільні пристрої 19
1.1.1.5 Мобільні операційні системи 22
1.1.2 Постановка задачі 24
1.1.3 Пошук акторів та варіантів використання 26
1.2 Проектування програмної системи. 28
1.2.1 Вибір моделі розробки 28
1.2.2 Вибір архітектурного шаблону 29
1.2.3 Вибір технології штучного інтелекту. 31
1.3 Конструювання програмної системи 35
1.3.1 Вибір мови програмування та середовища розробки 35
1.3.1.1 Swift 36
1.3.1.2 C++. 37
1.3.1.3 C# 38
1.3.1.4 JavaScript 39
1.3.1.5 Xcode. 40
1.4 Використання програмної системи. 45
1.4.1 Розгортання програмної системи та системні вимоги. 45
2. ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ 49
2.1 Загальний опис тестування. 49
2.2 Тестування створеного додатку. 51
2.3 Результат тестування. 51
3. ОРГАНІЗАЦІЙНО ЕКОНОМІЧНА ЧАСТИНА 52
3.1 Загальний підхід до визначення економічної ефективності розробки 52
3.2 Визначення ключових витрат. 53
4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ 62
4.1 Охорона праці. 62
4.2 Безпека в надзвичайних ситуаціях. 64

4.2.1 Освітлення виробничих приміщень для роботи з ВДТ. 64
ВИСНОВКИ 73
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 74
ДОДАТКИ 76
ДОДАТОК А - ТЕХНІЧНЕ ЗАВДАННЯ 77
ДОДАТОК Б - ТЕЗИ 78
ДОДАТОК В - ЛІСТИНГ КОДУ 80
ДОДАТОК Г - ДИСК 102

ВСТУП

З огляду на те, скільки часу людина проводить у гаджетах — світ уже поневолений технологіями. Але, деякі вчені всерйоз обговорюють можливість повстання машин і появи штучного інтелекту (далі — ШІ), який ступить на стежку війни з людьми.

Просунутий інтелектуальний комп'ютер може набути свідомості і здогадатися, що саме людина є головною причиною всіх бід на Землі. Незважаючи на те, що такий сценарій буде ненайпозитивнішим для долі людства, сучасні дослідники пнуться геть зі шкіри, щоб нагородити комп'ютери дедалі більшою кількістю людських рис.

Одним з останніх відкриттів стала робота нейробіологів зі США, які заявили, що ШІ можна вселити почуття власної «смертності» і буквально пояснити комп'ютеру, що його існування досить крихке.

У зв'язку з цим, НВ розбиралася, навіщо потрібно наділяти комп'ютер почуттями, коли з'явиться повноцінний ШІ і чи може людство бути симуляцією комп'ютерної програми, яку розробила одна з надцивілізацій.

Навіщо потрібен ШІ?

Оскільки інтелектуальні системи на кшталт нейромереж працюють з інформацією — їх застосування актуальне в усіх сферах людської діяльності. Логістика транспортних перевезень, обслуговування медичної сфери, банкінг, фінансові операції, оптимізація промисловості, автономне водіння, інфраструктура міст, — це лише мала частка того, де можна застосувати і де вже застосовують різні нейромережі.

Так, подібні системи взяли на себе частину роботи людини. Навчена робототехніка може керувати літаками, розбирати юридичні справи, створювати журналістські тексти і навіть проводити медичні операції. Звісно, все це поки лише перспективні сфери для повного застосування ШІ, і їхня діяльність все одно строго контролюється людиною.

Реальне застосування, скажімо, нейромереж актуальне в рутинній роботі, яка пов’язана з обробкою інформації. Через створення інтелектуальних систем можна оптимізувати роботу багатьох офісних співробітників, клерків, секретарів, бухгалтерів, аудиторів, поштових службовців тощо. Загалом, всі, хто зараз займається документообігом, математичними підрахунками, збором і обробкою інформації мають бути готові до того, що вже завтра їхнє місце займе залізяка, якій не потрібен відпочинок і соцпакет.

Втім, не варто боятися, що роботи знищать усі професії і заберуть роботу у людей. Нам, як і раніше, потрібні аналітики, фахівці з маркетингу, продажів, різні ідеологи, політологи, філософи, вчителі, юристи, — взагалі всі, хто може робити свою роботу краще, ніж комп’ютер. Не кажучи вже про творців цих комп’ютерів, вчених, інженерів, ІТ-фахівців, розробників ПЗ тощо.

Поки, на жаль чи на щастя, комп’ютер програє людині в більшості сфер, і те, що ми навчили нейромережі моментально виконувати конкретні, але складні завдання і вчитися на них — не означає, що такі системи розумніші за людину.

Примітивний ШІ, який сьогодні реалізований у вигляді інтелектуальних голосових помічників, машинного перекладу з однієї мови на іншу, створення текстів чи визначення об'єктів на фото — все ще дуже слабкий, робить багато помилок і не може обійтися без допомоги людини.

1. ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

1.1 Аналіз вимог до програмної системи

1.1.1 Аналіз предметної області

Штучний інтелект це технологія яка дозволяє машині навчатися та покращувати свої ж результати. Ця технологія має декілька важливих складових, а саме: [1]

- Бізнес вимоги.
- Роботу з даними.
- Алгоритми.

Ці складові зображені на Рисунку 1.1.

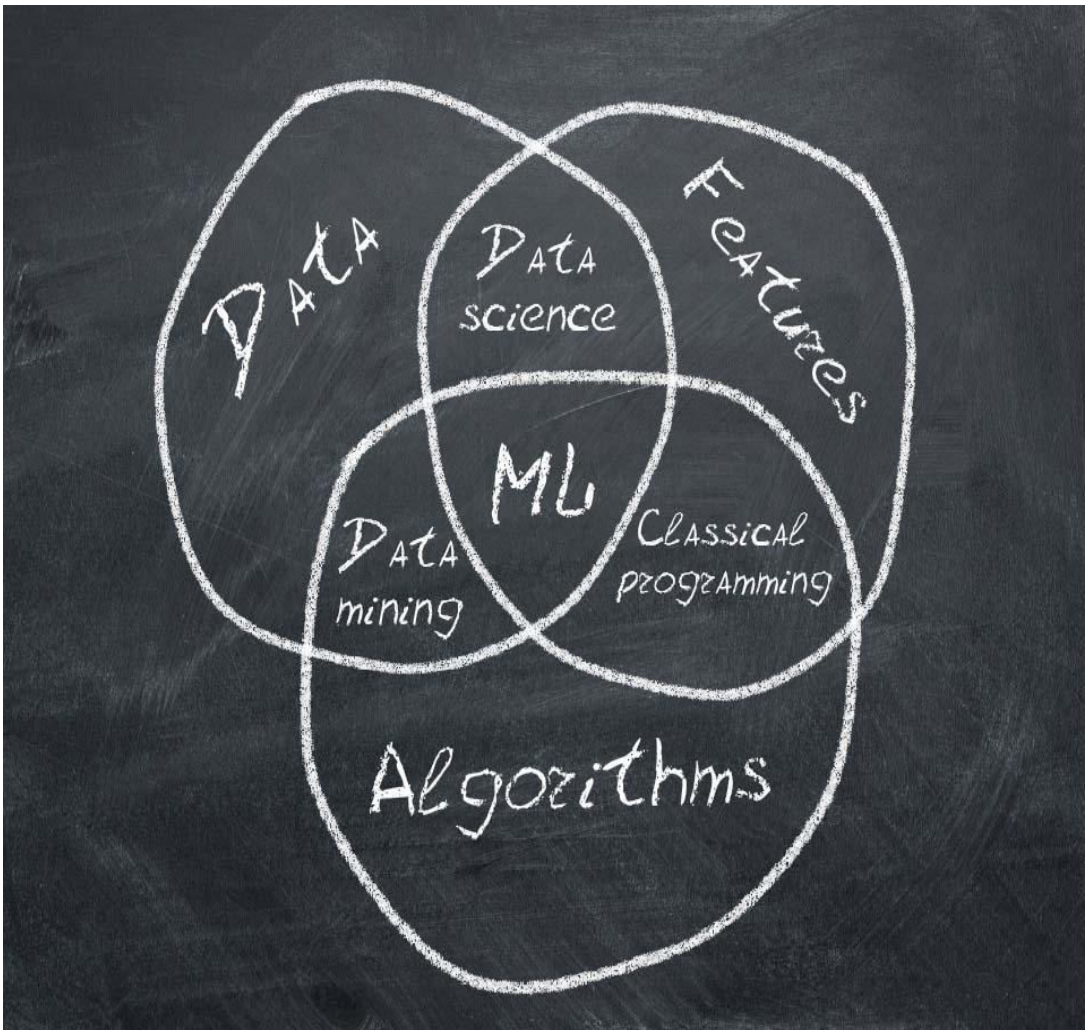

Рисунок 1.1. Складові штучного інтелекту.

Бізнес вимоги – це функціональні вимоги до програмного продукту, те яким він має бути (для якої мети, що він має робити, яку проблему вирішувати). Приклади бізнес вимог:

- Перегляд зображень.
- Перегляд пошти.
- Програвання музики.
- Аудіо/відео зв'язок.
- Показ відео.
- Пошук книг.

Це ще не весь список прикладів, сюди може увійти будь яка вимога із реального життя.

Всі характеристики повинні бути описанні якомога детальніше та уточнені ще до процесу розробки, це дозволить уникнути непорозумінь між замовником програмного продукту та виконавцем цього проекту та створити надійну і швидку програмну систему. Щодо машинного навчання – всі вимоги до програмного продукту повинні бути якомога точнішими до процесу розробки, тільки в такому випадку буде створена система яка робитиме точні прогнози та працюватиме як потрібно. [2]

Дані – це набір інформації на основі якої програмна система виконуватиме свою функцію. Для більшої точності прогнозів потрібно дати на вхід до системи, яка навчається, якомога більше правильних даних. Наприклад для фільтрації спаму потрібно дати багато повідомлень які містять конкретні ключові слова, такі як «купити». «дохід», «кредит» і т.д., позначеними як спам. Це дозволить системі навчитися на основі цих даних та робити правильніші прогнози в майбутньому. Найскладніша робота в розробці таких машин це збір правильних даних, на основі яких машина вчитиметься. В. даний час такі збірки даних можна купити, однак це може коштувати досить дорого, але це зекономить багато часу на зборі цих даних. Також важливо давати дані які мають конкретні ознаки, оскільки машина може бачити більше ніж людське око, і для користувача це виглядатиме як не доцільні дані.

Алгоритми – це послідовність дій які приводять до конкретного результату. Важливо правильно визначати та оцінювати алгоритми, які використовуватимуться в програмному продукті, оскільки від них буде залежати велика частина програмного продукту. В машинному навчанні алгоритми відіграють велику роль, оскільки саме від цієї характеристики залежить швидкодія та точність обробки даних. Для початку роботи над алгоритмами, які використовуватимуться в продукті, необхідно дослідити мету алгоритму та як користувач з ним буде взаємодіяти.

Існують такі види навчання:

- Навчання із вчителем.
- Навчання без вчителя.
- Навчання із підкріпленням.

Навчання із вчителем - передбачає підбір даних, які вказуватимуть конкретно на об’єкт та описом до нього. Це забезпечує кращу відповідність, бо саме людина вказує що це за об’єкт, так як вона його бачить. Такий спосіб більш підходящий в роботі із реальними людьми, оскільки виключає варіанти, коли машинне бачить якусь відповідність, яку не може побачити користувач, що в свою чергу може спантеличити кінцевого користувача продукту. Цей спосіб роботи вважається найнадійнішим, тому що весь процес навчання контролюється людиною.

Навчання без вчителя – це опис якогось об’єкта на початковому етапі та підготовка збірки даних в наступних, наприклад характеристика дверей:

- Прямокутні.
- З ручкою.
- Можлива присутність вікна.

На основі цих характеристик – система спробує знаходити відповідність на вхідних даних. Цей варіант допускає деяку не точність, з точки зору людини, оскільки машина може бачити відповідність в об’єктах які не бачить людське око.

Навчання із підкріпленням – спосіб навчання який полягає в тому, що машині ставиться конкретна задача, наприклад знайти зображення

повітряних кульок, і вона в свою чергу пробує сама знайти опис характеристик в зовнішньому середовищі, наприклад в інтернеті. Такий спосіб може бути актуальним для роботи із великими масивами даних, які можуть бути змінено в режимі реального часу, прикладом такого використання може бути бот для купівля/продажу інструментів, валюти на таких ринках як Forex. Штучний інтелект може збирати інформацію про існуючі транзакції і операції та співставити успішні операції із не успішними, вивчати новини та на основі цієї інформації робити операції на ринку. Ще одним прикладом може бути бот для тоталізатора, на прикладі футболу, та алгоритм його дій:

- Пошук інформації про команду
- Пошук інформації про команду суперників
- Вивчення новин та інформації про майбутній матч
- Вирахування ймовірність виграшу, програшу, певного рахунку
- Пропозиція користувачеві зробити ставку на певний матч

Недоліками систем штучного інтелекту є те, що всі вони вузькоспеціалізовані та вміють робити тільки те що їх навчили. Нехай в нас є чат бот який вміє показувати статус доставки посилки. У випадку якщо ми попросимо в нього щось інше – він просто відповість що не зрозумів запитання або ж щось інше, на що його запрограмували. Це вказує на те що він не може робити все і не може бути застосованим у всіх сферах одночасно. Кожна сфера потребує розробки своє системи, що не дозволяє замінити людини. Також ці системи не можуть думати логічно, та робити вибір виходячи з логіки, як це робить людина.

Проте штучний інтелект також має ряд переваг, на відміну від людини:

- не піддається зовнішньому впливу
- не втомлюється
- не страждає від розладів та неуважності
- є більш точним та набагато швидшим

Сьогодні є декілька основних напрямків розвитку таких технологій

- штучний інтелект (AI)
- машинне навчання (ML)
- нейронні мережі

Останнім часом технологія штучного інтелекту набула високої популярності та заполонила світ, також вона присутня майже у всіх сферах. Її дослідження проводяться з метою вивчення технології та знаходження дедалі більше застосувань. На сьогоднішній день - розвиток штучного інтелекту опирається на застосування теперішніх результатів, яких вдалося досягнути, в інших сферах науки, медицини, промисловості, бізнесу та навіть у повсякденному житті. Головною особливістю такої системи є здатність навчатися, та використовувати здобуті навички в майбутньому. Така програмна система здатна швидше справлятися із поставленою задачею і робити складніші та точніші прогнози на майбутнє.

Для того щоб правильно оцінити потреби користувачів та можливості майбутньої системи – необхідно дослідити цю область та виявити потенційні недоліки.

На даний момент є досить багато додатків які використовують штучний інтелект, та переглядачів зображень, однак жоден з цих продуктів не поєднує дані технології. Багато користувачів віддають перевагу стандартним переглядачам, що ускладнює вихід програмного додатку на ринок. Для того щоб зацікавити користувачів, потрібно запропонувати для них дещо не звичайне, те чого вони ще не зустрічали, та те що дозволить їм вирішувати проблеми.

Використання штучного інтелекту в сучасних мобільних додатках є досить популярним, та приваблює потенційних користувачів. На даний момент багато пристроїв використовують штучний інтелект для покращення зображень, застосування різноманітних ефектів та інших схожих речей. Також штучний інтелект використовують для розблокування пристроїв за допомогою обличчя.

1.1.1.1 Штучний інтелект

Штучний інтелект – галузь досліджень вмінь машин думати та діяти як людина а також приймати рішення на основі статистичних даних попередніх результатів. Така особливість дозволяє створювати технології та системи які виконують людську роботу якісніше та швидше. Таку машина порівнюють із «дитиною», тому що так само як і діти вона навчається, робить вибір та використовує здобуті раніше навички в поставлених задачах. Перші згадки про цю галузь були в 1956 році. Сам термін «Штучний інтелект» був придуманим Дартмутським Професором Джоном МакКарті. Він вперше зібрав групу для досліджень цієї технології, та спробував створити систему яка зможе навчатися, як дитина, за допомогою методу спроб та помилок і самовдосконалюватися. Це було понад 60 років тому і того часу ця технологія не досягнула загального признання, та була використана тільки для досліджень в наукових лабораторіях.

Останні роки ця технологія набула високої та заполонила сучасний світ і присутня майже у всіх сферах життя. Багато досліджень проводяться з метою вивчення цієї технології та знаходження для неї дедалі більше застосувань. На сьогоднішній день - розвиток штучного інтелекту опирається на застосування результатів, яких вдалося досягнути, в інших сферах науки, медицини, промисловості, бізнесу та навіть у повсякденному житті. Головною особливістю такої системи є здатність навчатися, та використовувати здобуті навички в майбутньому. Така програмна система здатна швидше справлятися із поставленою задачею і робити складніші та точніші прогнози на майбутнє.

Сучасний штучний інтелект є досить розвинутий та використовується в більшості систем, які потребують обробку великих об’ємів даних найшвидшим способом, в тому числі він використовується і в мобільних пристроях. Багато розробників використовують штучний інтелект в якості реклами, наприклад більшість сучасних мобільних телефонів мають вмонтовану камеру, яка має можливість пост обробки зображень за

допомогою штучного інтелекту. Також такі телефони містять вбудованих віртуальних асистентів, які за допомогою штучного інтелекту вивчають поведінку користувача в тих чи інших умовах і пробують вгадати побажання користувача в даних умовах, наприклад після виходу із певної локації – користувач телефонує певному номеру телефону в більшості випадків. Такий віртуальний асистент вивчає цю поведінку та при наступному виході із цієї локації – він запропонує саме дзвінок цьому контакту. Яскравим прикладом цього є віртуальний асистент на платформі iOS, якого звати Siri, може підказувати користувачеві що зробити, після того як він взяв телефон до рук. Часто такі підказки і справді відповідають дійсності, та допомагають користувачам такого пристрою робити менше дій для досягнення необхідного результату.

Темою магістерської роботи є «Розробка переглядача зображень з можливістю керування жестами на платформу iOS для мультимедійного простору». В основі цього додатку лежатиме штучний інтелект, який дозволить за допомогою жестів та виразів обличчя переключати картинки, приближувати та віддаляти зображення. Основною вимогою для програмного продукту це стабільна робота, та зручний інтерфейс. Цей продукт буде корисним у випадку неможливості безпосереднього контакту із пристроєм (прикладом можуть бути кухари, в яких немає можливості безпосереднього контакту із пристроєм, оскільки використовують засоби для захисту рук). В подальшому цю розробку можна розширити, та використовувати для відео, музики або інших цілей.

Багато додатків вимагають значних системних ресурсів однак сучасні пристрої є достатньо потужними для обробки цих даних, що і дозволяє використовувати таку технологію як штучний інтелект в сучасних смартфонах.

1.1.1.2 Машинне навчання

Машинне навчання – автоматизація побудови аналітичної моделі, що дозволяє збирати та аналізувати статистику даних. Таку систему можна використовувати для різних задач, таких як розпізнавання та класифікація наборів даних, наприклад для визначення спаму на пошті, рекомендації друзів в соціальних мережах, підбір відео для перегляду на відео хостингах, підбір реклами для користувачів і т.д..

Метою машинного навчання є автоматизація та спрощення людської роботи в аналізі різних статистичних даних. В першу чергу це принесе користь для бізнесу та спростить прогнозування росту для власників бізнесу, маркетологів та співробітників.

Машинне навчання об’єднує такі складові як бізнес вимоги, роботу з даними та алгоритми.

В даний час технологія машинного навчання охоплює велику кількість програмних додатків в різних сферах (банки, заправки, астрономія, роботи для виробництва і т.д.). Щодня виникає все більше напрямків та сфер застосування для цієї технології.

1.1.1.3 Нейронні мережі

Нейронні мережі – це система яка моделює роботу мозку (нейронів), та вирішує певні задачі враховуючи свій попередній досвід. Це дозволяє робити точні прогнози статистичних даних і в наступних обрахуваннях робить все менше помилок. Також це один з напрямів досліджень в галузі штучного інтелекту, основою якого є вміння імітувати нейронну систему людини. Така система має можливість виправляти існуючі помилки та вміє самонавчатися. Така система також моделює роботу людського мозку та дозволяє їй навчатися використовуючи попередній досвід.

Модель нейронних мереж зображено на Рисунку 1.2.

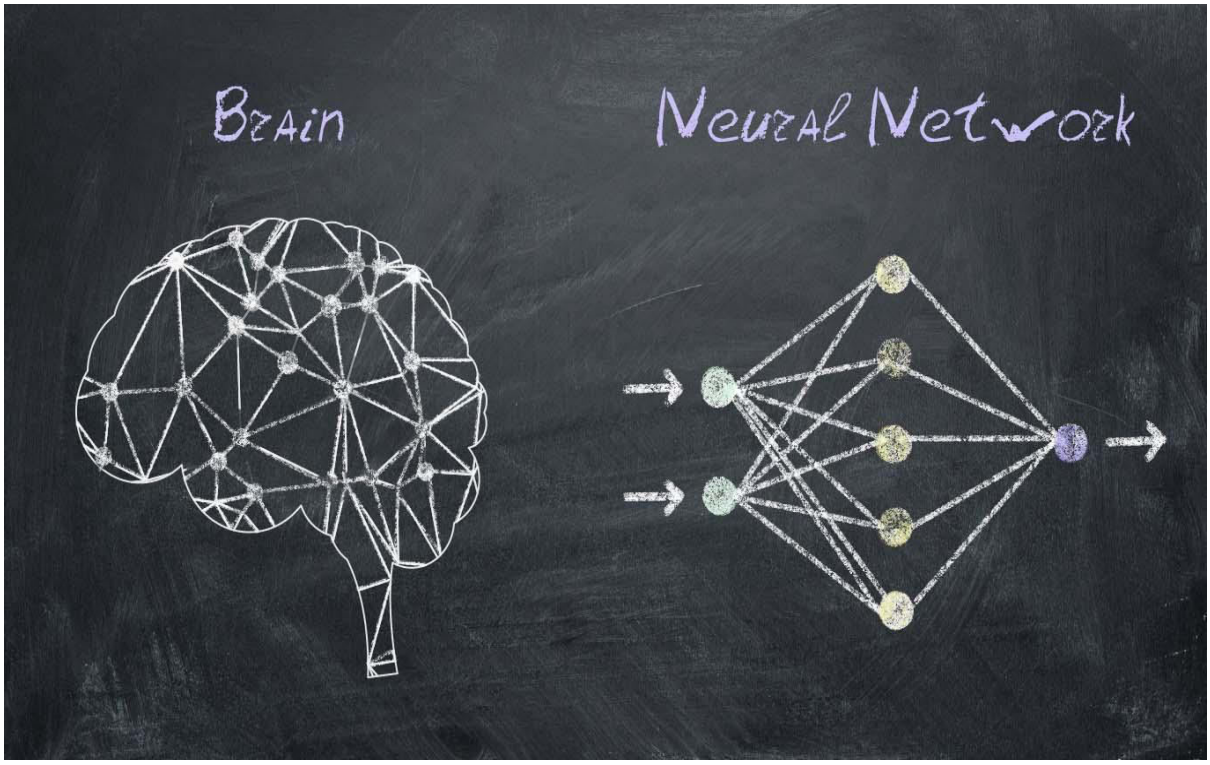

Рисунок 1.2. Нейронні мережі. [2]

З появою нової електроніки – почалися перші спроби відтворення людського мислення в електронних системах. Перше з досліджень було зроблено в 1943 році, після виходу статті математика Уолтера Піттса та Уоррена Маккалоха, про представлення моделі нейронних мереж в електричних схемах. Перші успіхи в дослідженні технології надихнули на нові дослідження та обрахування майбутніх можливостей таких систем. Починаючи з 1980х років інтерес до таких технологій зріс, багато фахівців почали дослідження в цій сфері та були натхнені вражаючими результатами попередніх дослідників.

Причиною такого ажіотажу навколо цієї технології були думки про те, що комп'ютери здатні працювати набагато швидше та вміти пристосовуватися до зовнішніх умов. На таку думку вчених надихнув людський мозок із його мережею нейронів. Причиною такої ідеї є те – що комп'ютери є набагато швидшими за людський мозок, мають кращу реакцію та швидкодію, однак в них є один значний мінус, вони не вміють пристосовуватися до зовнішніх умов та змінювати свої пріоритети на основі цих спостережень.

Під визначенням нейронної мережі мається на увазі об’єднання нейронів за допомогою зв’язків, які називаються синапсами. Ці нейрони потім об’єднуються в шари, що в свою чергу утворює нейронну системну мережу.

Одною із перших компаній яка почала застосовувати нейронні мережі – це всесвітньо відома компанія Google. Саме вона вперше використала нейронні мережі у своєму продукті. Ця технологія була використана компанією для того щоб оптимізувати процес пошуку в своїй системі та рекомендацій саме тих результатів які необхідні користувачеві. Також це допомагає краще зрозуміти запит користувача.

Ця технологія зараз активно використовується в різних сферах. Основною метою застосування це обробка великих об’ємів даних. Прикладом такої системи може бути пошукова система, яку було згадано раніше, для пошуку схожих зображень. Це вимагає обробку великого об’єму даних, однак за допомогою технології нейронних мереж це стає можливим та доступним.

Обчислення в таких системах виконуються наступним чином:

- Нейрон отримує дані
- Робить обчислення на їхній основі
- Повертає якийсь результат на вхід іншому нейрону
- Процес повторюється до тих пір, допоки останній нейрон не зробить свої обчислення

Технологія нейронних мереж також використовується в написанні сильних ботів, наприклад для ігри в покер, які можуть з легкістю обіграти сучасних чемпіонів у цьому виді спорту.

. Сучасний штучний інтелект є досить розвинутий та використовується в більшості систем, які потребують обробку великих об’ємів даних найшвидшим способом, в тому числі він використовується і в мобільних пристроях. Багато розробників використовують штучний інтелект в якості реклами, наприклад більшість сучасних мобільних телефонів мають вмонтовану камеру, яка має можливість пост обробки

зображень за допомогою штучного інтелекту. Також такі телефони містять вбудованих віртуальних асистентів, які за допомогою штучного інтелекту вивчають поведінку користувача в тих чи інших умовах і пробують вгадати побажання користувача в даних умовах, наприклад після виходу із певної локації – користувач телефонує певному номеру телефону в більшості випадків. Такий віртуальний асистент вивчає цю поведінку та при наступному виході із цієї локації – він запропонує саме дзвінок цьому контакту. Яскравим прикладом цього є віртуальний асистент на платформі iOS, якого звати Siri, може підказувати користувачеві що зробити, після того як він взяв телефон до рук. Часто такі підказки і справді відповідають дійсності, та допомагають користувачам такого пристрою робити менше дій для досягнення необхідного результату.

1.1.1.4 Мобільні пристрої

Історія мобільних пристроїв розпочалася ще в середині XX століття. В 1947 році народилася ідея створення телефону, яким можна буде користуватися в дорозі, а саме в автомобілі. Вага такого апарату була близько 40кг та, звичайно ж, використання такого пристрою поза межами автомобіля було неможливим. Після 30 років досліджень все таки вдалося значно зменшити вагу пристрою, та наблизитися до позначки в 12-14 кг. Однак вони вимагали постійного живлення та досі були «прив’язані» до автомобілів.

В 1967 році компанія «Motorola», під керівництвом Мартіна Купера, забезпечила поліцейських із Чикаго портативними раціями. І тільки в 1973 році відбулося випробування першого стільникового телефону в Нью-Йорку. Вага цього пристрою була близько 1,15кг, він містив 12 клавіш і 2 для прийняття/скидання виклику. Тільки після цього випробування всі зрозуміли що можливо створити портативний та автономний пристрій зв’язку, без безпосереднього джерела живлення. Масове виробництво цих

пристроїв розпочалося тільки через 10 років та його вартість становила 4000 доларів за пристрій. Цей пристрій зображено на Рисунку 1.3.

Рисунок 1.3. Перший стільниковий телефон. [3]

З тих часів багато чого змінилося і цей пристрій став звичним для нас, всі використовують його для спілкування, пошуку інформації, перегляду зображень та відео, прослуховування музики та інших потреб.

Протягом останніх років спостерігається тенденція росту ринку мобільних пристроїв. Збільшується не тільки кількість користувачі а й кількість виробників. Мобільний пристрій зараз – це не тільки засіб спілкування та комунікації, також він використовується для розваг, пошуку інформації перегляду чи прослуховування медіа файлів. Також на даний момент мобільні телефони майже викорінили стаціонарні. Зараз майже немає приватних будинків, чи квартир, в яких є стаціонарний пристрій, однак вони досі використовуються в компаніях, державних установах.

Сучасні мобільні телефони є досить потужними та зручними, з точки зору ергономіки, що приваблює все більше і більше користувачів. Сучасне населення веде досить напружений спосіб життя, в якому є постійна необхідність зв'язку, спілкування та навіть роботи. Ось декілька основних переваг сучасних мобільних телефонів:

- Активна взаємодія з іншими людьми
- Можливість відео/аудіо зв'язку
- Доступ до великого об'єму інформації, завдяки глобальній мережі інтернет
- Різного роду дослідження
- Навігація
- Розваги
- Розумні асистенти які, за допомогою штучного інтелекту, підказують користувачам наступні дії та допомагають у користуванні пристроєм
- Зйомка Фото та Відео на рівні із фотоапаратами, а часом навіть вище

Сучасні технології дозволяють використовувати свій мобільний пристрій для інтеграції із багатьма іншими технологіями, наприклад:

1) Розумний будинок. Саме так, мобільні пристрої дозволяють керувати різною електронікою в будинку навіть не встаючи з дивану. Наприклад ви можете, за допомогою мобільного телефону, включити світло, музику, подивитися через камеру на подвір'ї хто до вас прийшов і т.д..
2) Автомобілі. Зараз набирають популярності електричні автомобілі, які оснащенні високотехнологічними комп'ютерними системи, що дозволяє керувати автомобілем навіть через свій смартфон. Ось приклад дій які вже можна робити з автомобілями, обладнаними такими комп'ютерами:
    a. Ввімкнення вимкнення двигуна.

b. Подати сигнал з автомобіля, якщо ви залишили його десь на великій парковці і забули де саме.
c. Підкликати його до себе. Ця технологія з’явилася в автомобілях компанії Tesla. Вона дозволяє за допомогою лише однієї кнопки на телефоні дати команди автомобілю під’їхати до вас.
d. Ввімкнути/Вимкнути аудіо система.
e. Почати прогрів двигуна.

3) Різні електронні прилади такі як ваги, термометр, тренажери і т.д.. Всі ці прилади вміють показувати необхідну інформацію користувачам на екрані їхнього смартфону.

1.1.1.5 Мобільні операційні системи

Мобільна система iOS була розроблена компанією Apple ще в 2007 році, яка називалася iPhone OS. Перша версія операційної системи була досить «сирою» та підтримувала тільки веб додатки. Тільки в 2008 році з’явилася перша версія iOS SDK для розробників та почали з’являтися перші додатки від сторонніх розробників. Протягом 12 років система покращувалася і зараз вона налічує безліч користувачів та вважається найзручнішою для більшості користувачів. Також одною із переваг є AppStore (магазин додатків для iOS), який налічує безліч якісних додатків. В платформі iOS є безліч плюсів, таких як:

- Оновлення системи та тривала підтримка старих пристроїв. Саме компанія Apple ввела поняття «оновлення мобільної системи» що дозволяє обновляти не тільки додатки, а і саму систему. Саме це дозволяє підтримувати старі пристрої до сих пір. Для прикладу iPhone SE, випущений весною 2016 року – обновився із версії 9.03 до 13.2.3, що робить його актуальним пристроєм до сих пір.

- замовників – хочуть розробляти додатки спершу на платформу iOS – оскільки саме її користувачі вважаються більш платоспроможним користувачами.
- «Екосистема» Apple. Компанія Apple робить синхронізацію всіх своїх пристроїв, від годинника до ноутбука. Це дозволяє отримувати доступ до своїх даних із будь якого пристрою, за допомогою хмарного сервісу iCloud.
- . Компанія Apple розробила свої алгоритми анімації, що робить ілюзію того, що операційна система передбачає ваші дії та робить анімації набагато плавнішими.
- Безпека. Компанія стверджує що їхня операційна система є найбезпечнішою та її неможливо «зламати». Було декілька гучних новин, які вказували на вразливість системи, однак компанія швидко виправляла їх та на даний момент немає схожих проблем.

Однак і в системі Android їх достатньо:

- Відкритий програмний код. Оскільки ця операційна система є відкритою для модифікацій та перегляду (AOSP) – це дозволяє робити системою дуже гнучкою та можливість реалізувати те, що потрібно саме у твоєму випадку. Саме це дозволяє виробникам робити свої графічні оболонки, такі як:
    - TouchWiz(Samsung)
    - MIUI(Xiaomi)
    - FlameOS(Meizu)
    - EMUI(Huawei, Honor)

Однак поруч із цими оболонками є так звана «чиста оболонка», яка використовується деякими виробниками пристроїв в межах програми «Android One». В цій програмі беруть участь виробники таких брендів як:

- Xiaomi (MI a1, MI a2, MI a2 lite і MI a3)
- Nokia (Більшість пристроїв)
- Motorola (Серія Motorola One)

- Також самі власники системи Android – Google та модельний ряд пристроїв Google Pixel.
- Більшість пристроїв мають набагато кращі характеристики ніж пристрої компанії Apple:
    - пам'ять.
    - Процесор.
    - (Hardware).
    - Пам'ять пристрою.
- Гнучкість самої системи. Всі Android пристроїв мають можливість налаштування системи під себе. Наприклад:
    - Налаштування робочого стола
    - Кольорової схеми всього пристрою (на відміну від платформи iOS, яка має всього два стилі, чорний та білий).
    - Можливість додавання різноманітних розширень на робочий стіл.

Дослідження та порівняння систем Android та iOS провела компанія Norton, та переглянути результати можна за посиланням [7]

1.1.2 Постановка задачі

Завданням на магістерську роботу є розробка переглядача зображень із підтримкою керування жестами на платформу iOS. Основною задачею цієх роботи – це підтримка керування жестами та мімікою обличчя, без безпосереднього контакту із пристроєм. Результатом роботи повинен бути додаток на платформу iOS.

Керування додатком повинно бути зручним для користувача та не приносити незручностей. Також важливо розробити сучасний інтерфейс додатку та із хорошим користувацьким досвідом. Основною функцією такого додатку буде розпізнавання жестів та міміки обличчя і для реалізації цього функціоналу необхідно використати штучний інтелект.

Візуальні компоненти додатку повинні добре продуманими та зрозумілими. Вони повинні буду виконаними із сучасним дизайном та стандартами. Застосунок повинен працювати плавно, та без жодних підвисань та доставляти користувачеві якісний користувацький досвід. Порівняння значення користувацького досвіду та користувацького інтерфейсу – можна переглянути на Рисунку 1.4.

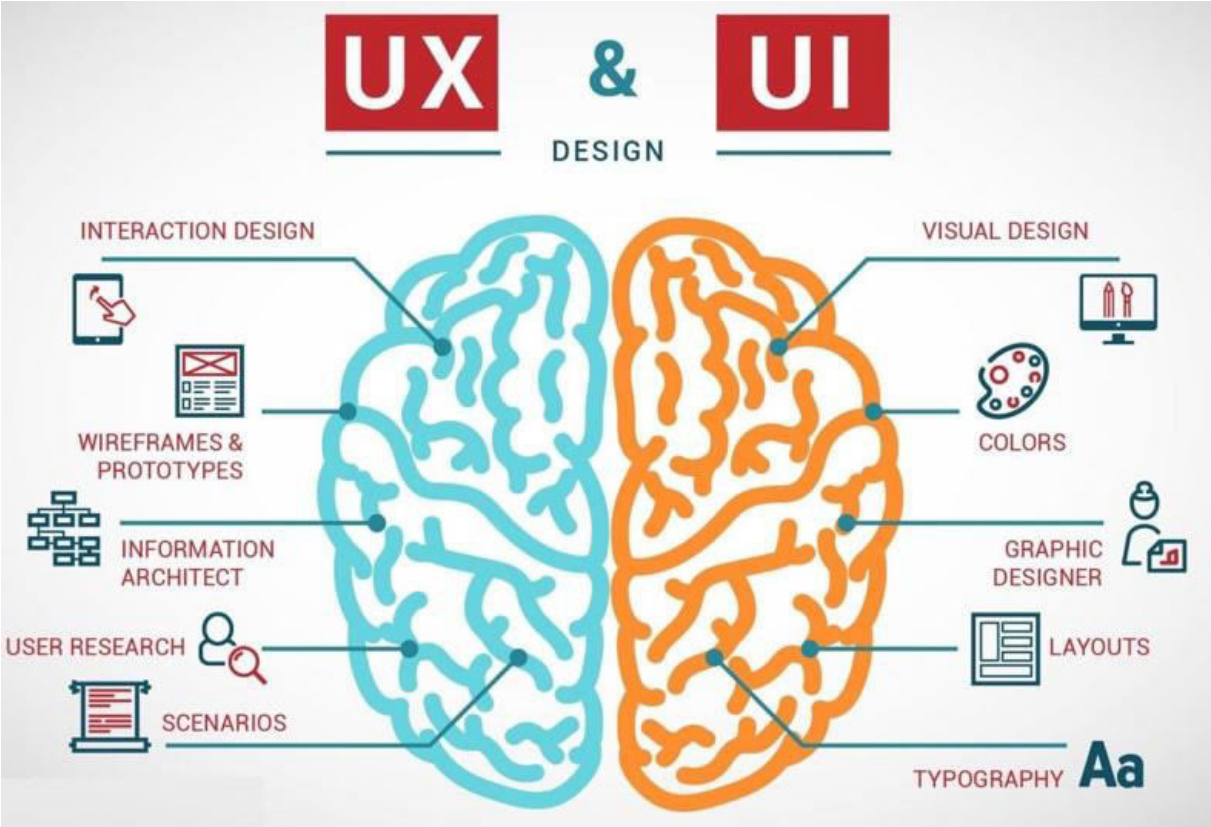

Рисунок 1.4. UI та UX

Для вирішення проблеми із розпізнавання жестів – необхідно застосувати штучний інтелект, який вирішить цю проблему.

Додаток повинен працювати плавно, без зависань

Ключовим модулем цього додатку буде модуль розпізнавання команд користувача за його жестами, використовуючи для цього камеру. Всі користувачі повинні знати про те, що за ними слідкує камера, повинні бути попередженими про це та згодні на такі умови. Також одним із елементів функціоналу будуть налаштування різного роду. Інструмент за допомогою якого вестиметься розробка та який використовуватиметься в додатку буде

визначено пізніше, шляхом досліджень. Порядок задач які необхідно визначити для подальшої розробки:

- Вибір інструментів за допомогою яких вестиметься розробка майбутнього додатку.
- Вибір технології яка використовуватиметься в процесі розробки.
- Обрання архітектурного шаблону для майбутнього мобільного додатку.
- Обрання інструментів для роботи із машинним навчанням.
- Побудувати діаграму класів.
- Реалізація системи.
- Тестування системи.

1.1.3 Пошук акторів та варіантів використання

Ознайомившись із поставленою задачею та вимогами для програмної системи було знайдено всього одного актора системи, користувач, тому що технічне завдання не передбачає будь-якого іншого користувача.

Ось основні варіанти використання для майбутнього додатку:

- Перегляд списку альбомів
- Перегляд інформації про альбом
- Перегляд фото із альбому
- Перегляд фото на повний екран
- Перегляд інформації про фото
- Приближення віддалення фото.

Перегляд альбомів – перегляд альбомів користувача із пристрою. Вигляд альбомів буде у вигляді списку.

Відкриття альбому з фотографіями – ця дія дозволить користувачеві побачити список фотографій із альбому у вигляді так званого grid view.

Відкриття фото на повний екран – дозволяє відкривати фото на весь екран для перегляду, цей екран матиме можливість переключання зображень, за допомогою жестів.

Перегортання фото – дозволяє виконувати переключення фото.

Приближення/Віддалення фото – дозволяє виконувати дії над зображенням, на повний екран, такі як ZoomIn та ZoomOut.

Налаштування додатку – оскільки додаток має достатньо функціоналу – він потрібує різного роду налаштувань. Ось перелік деяких з них:

- Доступ до камери
- Контроль та налаштування варіантів керування
- Налаштування відображення зображень

Варіанти використання штучного інтелекту для розпізнавання жестів користувача:

- Довге моргання правим оком. Стосується повноекранного перегляду фото. Дозволяє переключити зображення вправо.
- Довге моргання лівим оком. Стосується повноекранного перегляду фото. Дозволяє переключити зображення вліво.

Ключовими варіантами використання буде виконання можливих дій, за допомогою жестів та штучного інтелекту. Одним із таких варіантів – буде перегортання зображень за допомогою погляду. Також доступними будуть функції звичайного переглядача, такі як перегляд фото на пристрої та інші дії. Основним функціоналом додатку є розпізнавання жестів. Це буде досягнуто за допомогою штучного інтелекту. Основні варіанти використання зображено на рисунку 1.5

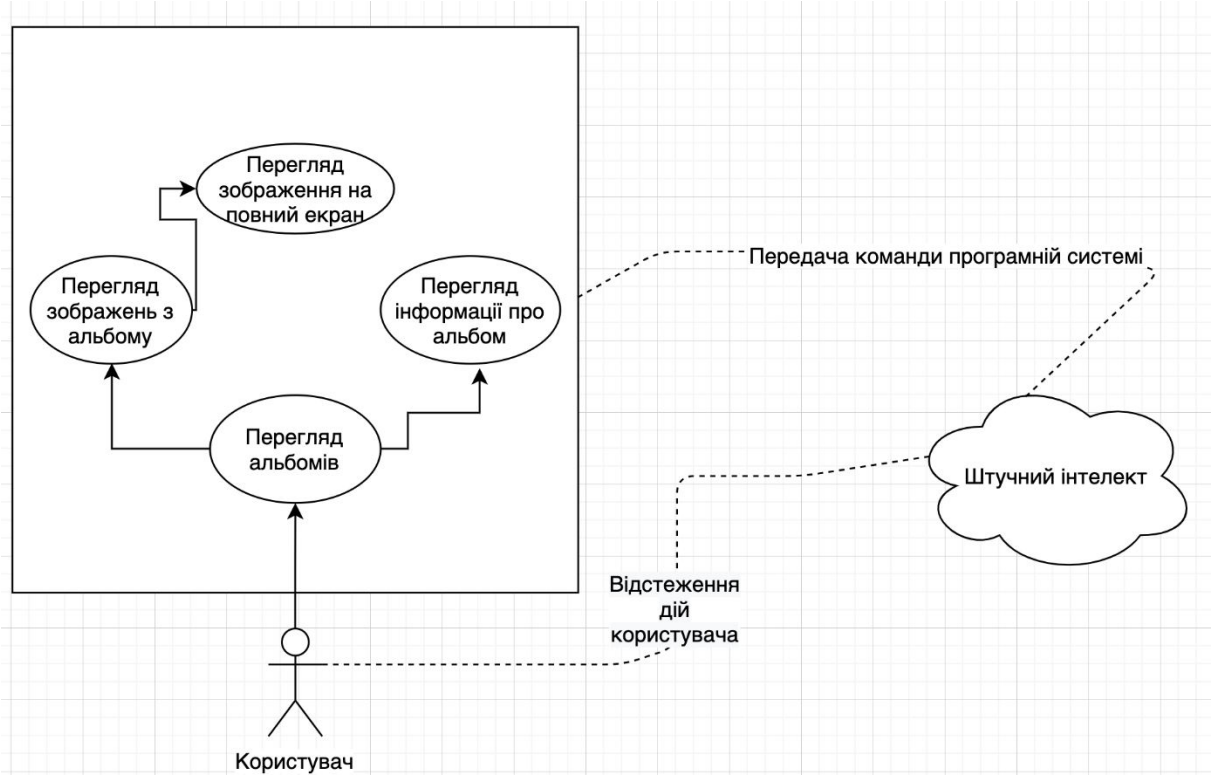

Рисунок 1.5. Зображення загальної картини варіантів використання в додатку.

1.2 Проектування програмної системи.

1.2.1 Вибір моделі розробки

Існує декілька моделей розробки програмного забезпечення.

- Водоспадна (каскадна) модель. Ця модель є однією із найпопулярніших на даний момент, оскільки вона є простою і зрозумілою. Основна ідея якої є в тому, що кожний етап розробки має свою мету та певний результат. Основним недоліком цієї моделі є те що вона не є гнучкою, та з нею можуть виникати проблеми, якщо замовник вирішить щось змінити в робочому процесі, або кінцевому продукті. У такому випадку доведеться все починати заново. Також ця модель не дає повного уявлення про систему.
- Ітераційна спіральна модель. Основною ідеєю є те, що весь етап розробки розбивається на декілька етапів:
    - Планування

- Розробка
- Тестування

І ці етапи повторюються до тих пір, доки не буде отримано результат, який бажає замовник. Із явних плюсів цього методу є те, що кожний крок контролюється замовником, та це дозволяє йому вносити зміни на будь-якому етапі розробки, що є більш привабливою моделлю.

Для розробки програмної системи було обрано ітераційну модель.

1.2.2 Вибір архітектурного шаблону

Для розробки на платформу iOS доступно декілька архітектурних моделей:

- MVC (Model View Controller) - стандартна архітектурна модель для платформи iOS, і є рекомендованою компанією Apple. Цей шаблон складається із трьох основних частин, таких як Model(M) View(V) Controller(C). Зв'язок між шарами додатку зображено на Рисунку 1.6.

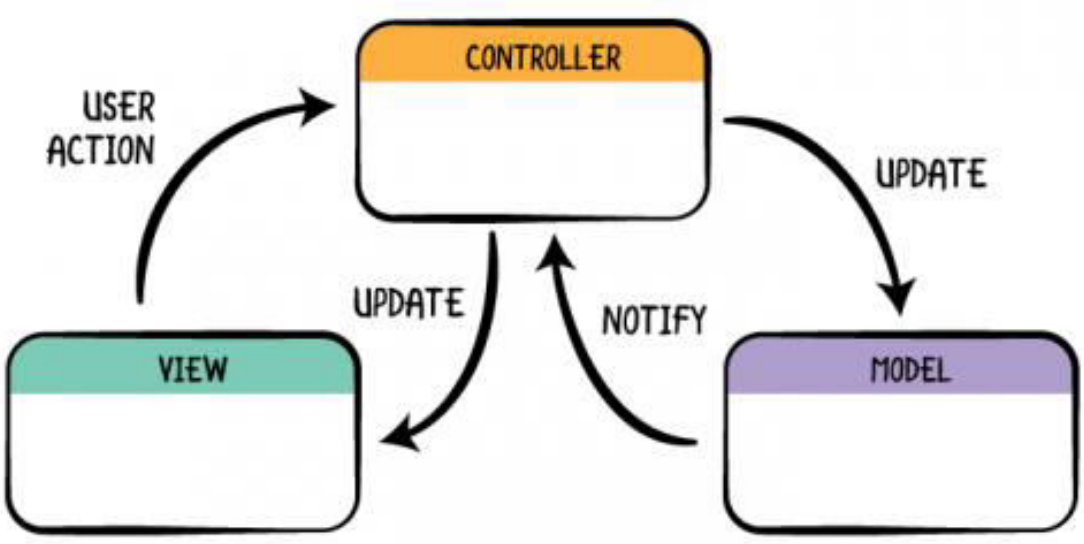

Рисунок 1.6. Зв'язок шарів додатку в архітектурній моделі MVC.

- Model – представляє моделі всього проекту та логіку їхньої обробки. Він містить:

        - Відправку/Отримання даних із сервера за допомогою HTTP/HTTPS протоколів, та налаштування правил для цього.
        - Зберігання отриманих даних в локальній базі даних, кеші та інших типах сховища.
        - Конвертацію отриманих даних в локальні моделі.
        - Різного типу менеджери та абстрактні реалізації.
        - Абстрактні джерела даних, який дозволяють спростити логіку отримання та обробки даних.
        - Сталі значення. Кожний проект має свої сталі та ключі, які необхідні для роботи із певними даними.
    - View – містить все те що бачить користувач, а саме кнопки, картинки, текст, списки та їхню розташування
    - Controller – це проміжний шар між View та Model. Він відповідає за те, як співпрацюватимуть між собою View та Model, а саме, як відображатимуться дані з моделі. Це дозволяє розділити та спростити логіку роботи та зв'язок цих шарів. Його робота полягає в отриманні даних та показ їх користувачеві, а також в обробці цих даних.
- MVVM (Model View ViewModel) – на даний момент є найбільш популярною моделлю, основною ідеєю якої є поведінковий шаблон «спостерігач». Перші згадки про цей шаблон були в 2005 році та були орієнтовані на технологію компанії Microsoft, яка називається WPF (Windows Presentation Foundation). Model і View відіграє таку саму роль як і в шаблоні MVC. Єдиною різницею є Controller, його роль виконує ViewModel.
    - ViewModel – це проміжний шар, як і в MVC, однак має одну значну різницю. На відміну від Controller він не запитує дані а підписується на них та отримує всі обновлення. В iOS це передбачає створення додаткового класу та розділяє ViewController і ViewModel.

ViewController це сутність iOS додатку який показує дані на UI та передає зміни користувача у ViewModel, також він підписується на зміни із ViewModel. ViewModel відповідає за обробку подій користувача та отримання і обробку даних, також повідомляє UI шар про зміни.

- MVP (Model View Presenter) – загальною моделлю цей шаблон дуже схожий на інші, однак Controller/ViewModel заміняється такою сутністю як Presenter. Presenter – це сутність, яка дозволяє користувачу працювати із моделями та розмежовує ці сутності. Він містить посилання на UI та може змінювати її відносно своїх потреб.

Вирахувавши всі плюси та мінуси дослідження можливих архітектурних рішень – прийнято рішення використовувати MVVM в подальшій розробці. Це підпадає під сучасний тренд розробки мобільних iOS додатків та дозволить з меншими затратами часу та ресурсів виконати поставлене завдання.

1.2.3 Вибір технології штучного інтелекту.

Основною задачею цього програмного продукту є забезпечення користувачам якісного додатку, який вирішить їхні проблеми із традиційними схожими додатками. Додаток повинен працювати швидко, без зависань, та повинен мати хороший програмний дизайн. Для вирішення проблеми із безконтактним керуванням додатку було прийняте рішення використати штучний інтелект, який вміє розрізняти жести та вирази обличчя, а саме Firebase. Оскільки ця компанія є комерційною, отже послуги компанії є платними. Деталі вартості послуг компанії Firebase можна знайти по посиланню. Ось екран вартості деяких з них, ціни зображені на рисунку 1.7:

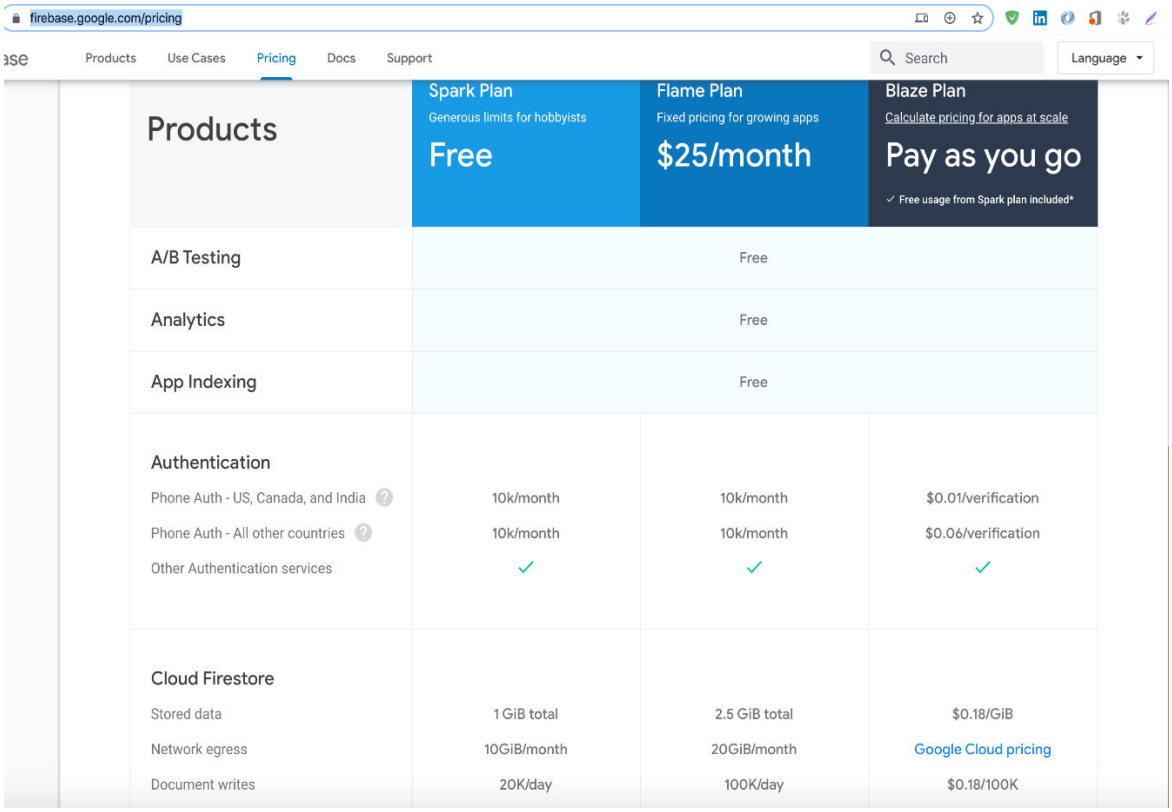

Рисунок 1.7. Вартість послуг сервісів Firebase [5]

Firebase – це розробка компанії Firebase Inc, та була викуплена компанією Google у 2014 році. Спочатку ця розробка була спрямована на вирішенні проблем із розробкою чатів та інших веб додатків, однак після придбання компанією Google функціонал був значно розширеним і зараз цей проект включає в себе досить багато модулів, одним з яких є Firebase ML (Machine learning) Kit. Найбільші та найпопулярніші сервіси які включає в себе Firebase:

- Firebase Auth
- Realtime Database
- Firebase Storage
- Firebase Analytics

Firebase Auth – сервіс який спрощує авторизацію користувачів в додатках для платформ Android, iOS та веб додатків. Для кожної із цих платформ випущена «бібліотека» яка спрощує роботу із цією технологією та реалізує функції, які притаманні якійсь конкретній платформі (наприклад робота із Keychain для платформи iOS).

Realtime Database – сервіс який надає доступ до хмарної бази даних, яка знаходиться на віддаленому сервері, що спрощує та пришвидшує розробку простих додатків як чат, та схожі на них. Це дозволяє розробити мобільні клієнти без розробки серверної частини, що економить ресурси при замовленні розробки таких додатків. Однак використання цього сервісу є платним, у випадку перевищення лімітів на використання серверних ресурсів. Актуальну інформації стосовно вартості та оплати RealTime Database можна знайти на їхньому офіційному сайті.

Firebase Storage – дозволяє відвантажувати файли користувача на віддалений сервер та завантажувати їх в подальшому. Цей сервіс є умовно безкоштовним, та має ліміти при яких користування є безкоштовним.

Firebase Analytics – рішення для збирання даних про користувачів для досліджень. Дозволяє отримувати таку інформацію як вік користувача, категорію, інформацію про його пристрій і т.д.. Зараз використання цього сервісу є досить популярним та використовується багатьма компаніями.

Платформою для розробки була обрана мобільна платформа iOS. Оскільки вона є однією із найпопулярніших на даний час та займає друге місця після платформи Android.

Сьогодні розробити мобільний додаток із підтримкою такої технології – не викликає значних проблем, оскільки є багато варіантів бібліотек, які допомагають вирішити дану проблему, та містять інструменти для роботи із цією технологією. Ось перелік декількох бібліотек для розробки програмної системи із підтримкою штучного інтелекту на платформу iOS:

- YCML
- MAChineLearning
- Multi-Perceptron-NeuralNetwork
- BPN-NeuralNetwork
- Multi-Perceptron-NeuralNetwork
- KRKmeans-Algorithm
- KRFuzzyCMeans-Algorithm
- KRHebbian-Algorithm

- MLPNeuralNet
- Bender
- Swift AI
- Swift for Tensorflow
- BrainCore
- Swix
- AIToolbox
- MLKit
- Swift Brain
- Perfect TensorFlow
- PredictionBuilder
- Awesome CoreML
- Awesome Core ML Models
- Firebase MLKit
- Core ML

Більшість з них написані невеликими командами, та не набули достатньо популярності. Розглянемо декілька з них детальніше та спробуємо зробити правильний вибір для майбутнього додатку.

Firebase MLKit.

Розробка компанії Firebase, яка була викуплена та належить компанії Google, починаючи з 2014 року. Ця бібліотека для роботи із штучним інтелектом та дозволяє робити багато дій з отриманими даними, також ця бібліотека має багато функціоналу зв’язаного із штучним інтелектом:

- Text recognition. Розпізнавання рукописного вводу, розпізнавання тексту із зображень. Швидше за все саме ця технологія використовується в додатку перекладача, для мобільний пристроїв Android та iOS, від компанії Google.
- Face detection. Дозволяє розпізнавати обличчя із фото або із камери. Також має можливість розпізнавання міміки та подальшої її обробки. Саме це є необхідним функціоналом для додатку та буде використовуватися в майбутньому.

- Barcode scanning. Ця функція дозволяє розпізнавати штрих коди, QR коди. Це може бути використаним для пошуку товарів за штрих кодом, розшифрування QR кодів.
- Image labeling. Використовується для розпізнавання об’єктів зображених на фото або з камери.
- Landmark recognition. Технологія розпізнавання локацій, дозволяє визначати локацію за фото.

Темою магістерської роботи є «Розробка переглядача зображень з можливістю керування жестами на платформу iOS для мультимедійного простору». В основі цього додатку лежатиме штучний інтелект, який дозволить за допомогою жестів та виразів обличчя переключати картинки, приближувати та віддаляти зображення. Основною вимогою для програмного продукту це стабільна робота, та зручний інтерфейс. Цей продукт буде корисним у випадку неможливості безпосереднього контакту із пристроєм (прикладом можуть бути кухари, в яких немає можливості безпосереднього контакту із пристроєм, оскільки використовують засоби для захисту рук). В подальшому цю розробку можна розширити, та використовувати для відео, музики або інших цілей.

Багато додатків вимагають значних системних ресурсів однак сучасні пристрої є достатньо потужними для обробки цих даних, що і дозволяє використовувати таку технологію як штучний інтелект в сучасних смартфонах.

1.3 Конструювання програмної системи

1.3.1 Вибір мови програмування та середовища розробки

Історія розробки сторонніх додатків для платформи iOS починається з 2008 року, саме тоді з’явилася можливість розробки сторонніх додатків для неї, а сама платформа називалася iPhoneOS 2. Саме тоді з’явилася перша версія Xcode та можливість розробки сторонніми розробниками.

Також тоді з’явився додаток AppStore, який був встановлений відразу «із коробки» на всіх мобільних пристроях (Apple), та дозволив завантажувати необхідні додатки.

Із виходом AppStore – компанія Apple заборонила установку додатків іншим способом, однак така можливість залишилася за умови використання Jailbreak версії iOS, що є незаконним. Для завантаження додатку в AppStore – кожний розробник повинен згенерувати власний сертифікат за допомогою Xcode, що підтверджує належність прав на додатку конкретному розробнику. В процесі розробки є можливість встановлювати додатки прямо на пристрій за допомогою безкоштовного сертифікату, який дозволяє встановити його на обмежену кількість пристроїв.

Першою мовою для розробки iOS додатку була ObjectiveC, за основу якої було взято об’єктно-орієнтовану мову c++. Для створення класу в ObjectiveC необхідно створити 2 файли:

- із розширенням .h, який описує загальну логіку цього класу, основні його методи та змінні
- із розширенням .m. Цей файл потрібний для реалізації класу, та написання необхідної логіки

Розробка цією мовою передбачає ручне очищення створених змінних та необхідність самому керувати пам’яттю, що було причиною деякої складності при розробці. Ця проблема була вирішена компанією в 2011 році, саме тоді було запроваджено таку технологію як ARC (Automatic Reference Counter). Основною метою цієї технології спростити роботу із пам’яттю за допомогою підраховування посилань на кожний об’єкт. У випадку якщо немає посилань – цей об’єкт видаляється із пам’яті автоматично. Синтаксис цієї мови програмування є досить складним та не є схожим на інші мови програмування, тому перейти на цю мову із інших досить складно. На сьогодні ця мова програмування майже не використовується та більшість проектів розробляється мовою Swift.

1.3.1.1 Swift.

Проблемою мови ObjectiveC було те – що розробник повинен був контролювати все що він пише – самостійно, та збільшує ризик фатальної помилки, після якої додаток завершиться аварійно. Компанія Apple розробила іншу мову на заміну, яка мала багато нових можливостей та вирішували існуючі проблеми. Ось декілька нововведень які з'явилися в мові swift:

- Розширення. Можливість написання коду до вже існуючих класів, що додало багато нових можливостей.
- Автоматичне очищення пам'яті.
- Більш дружелюбний синтаксис.
- Автоматичне визначення типу даних.

За основу для цієї мови були взяті такі мови програмування як:

- Objective-C
- Rust
- Haskell
- Ruby
- Python
- C#
- Та інші

Це дозволило створити свою, унікальну, мову програмування врахувавши особливості інших.

На відміну від Objective-C вона не вимагає MacOS для розробки, та має можливість запуску окремих класів на операційній системі Linux. На даний момент саме ця мова використовується для розробки мобільних додатків.

1.3.1.2 C++.

Мова для розробки програмного функціоналу, який передбачає використання протоколу доступу до розширених функцій операційної

системи. Вона дозволяє працювати із таким функціоналом та технологіями як:

- Open GL. Графічна бібліотека яка дозволяє створювати 2D та 3D компоненти. Ця технологія забезпечує API для прямої роботи із графічним ядром пристрою, та приховує складнощі в роботі із графічним ядром. Саме вона використовується для розробки більшості ігор та додатків які працюють із 3D графікою.

- Робота із hardware пристрою. Дозволяє працювати із окремим модулями пристрою, такі як Bluetooth, камера, динаміки, мікрофон та іншими компонентами.

Саме ця мова використовується для таких додатків як ігри, отримання детальної інформації про пристрій (потужність сигналу WiFi і т.д.)

1.3.1.3 C#.

Мова для розробки мобільних програмних додатків, яка передбачає створення за допомогою технології Xamarin. Ця технологія дозволяє робити додатки, які запускатимуться на обох мобільних платформах, на iOS, Android а також це єдиний спосіб розробки на ще одну мобільну платформу, яка вже вийшла з ринку декілька років назад, Windows Phone. Це дозволяє зекономити багато часу у випадку розробки простого додатку на декілька платформ, однак в такому випадку необхідно пожертвувати деяким функціоналом, оскільки розробка на декількох платформах одночасно - може використовувати технології які не доступні на інших платформах. На даний момент вона не є настільки популярною як і «native» розробка, однак досі тримає свої позиції на ринку та користується популярністю. Ця розробка належить компанії Microsoft, але декілька років назад вона була закрита та більше не підтримується.

Мова C# дозволяє розробляти ігри за допомогою Unity технології. Ця технологія дозволяє розробляти ігри, використовуючи багато вже описаних компонентів та має можливість використання реалістичної

фізики в іграх, такої як падіння, зупинка, тертя і т.д. Досить багато сучасних ігор використовують Unity у якості ядра. У випадку необхідності розробки ігри – ця технологія дозволяє зекономити багато часу та результати справді вражають. Ось декілька ігор розроблених за допомогою Unity: [6]

- Monument Valley 2
- Hearthstone

Повний список ігор написаних за допомогою ядра Unity – можна переглянути за посиланням [11]

1.3.1.4 JavaScript.

Javascript – на даний момент є найпопулярнішою мовою, оскільки використовується в багатьох проектах, від серверів до клієнтів та мобільних платформ. Основною його перевагою є кросплатформеність, що дозволяє використовувати її в безлічі проектах та в багатьох сферах. Не виключенням є і мобільна розробка. Ось декілька технологій які використовується в мобільній розробці:

- Cordova. Розробка компанії Apache. Дозволяє розробляти мобільні додатки використовуючи HTML5, CSS, JavaScript. Вона дозволяє запускати веб додатки на мобільному телефоні, які виглядатимуть як звичайні, однак вони, все таки, мають достатньо відмінностей, однією з них є те, що ця технологія запускає додаток в так званому контейнері, який перетворює команди із додатку на команди системи. Саме це дозволяє виконувати додатки на будь-якій платформі. Ця технологія має і мінуси, одним з них є запуск контейнера, який перетворює код (Рисунок 1.8) в системний, що призводить до уповільнення роботи програми.

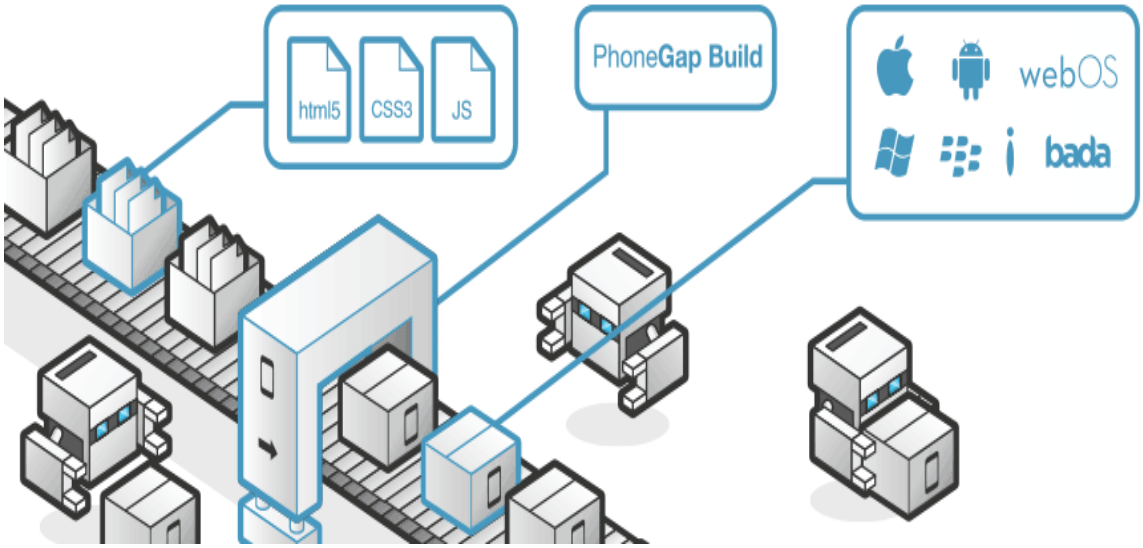

Рисунок 1.8. Схема роботи технології Cordova

- React Native. Є однією із найпопулярніших JavaScript технологій для мобільної розробки, а також найрозвинутішою у цій сфері. На відміну від попередньої технологія – вона дозволяє перетворювати код в команди системі на етапі компіляції, що значно пришвидшує його виконання після установки. Ця розробка належить компанії Facebook, яка використовує її в своїх продуктах. У випадку необхідності простого клієнта для сервера на Android та iOS – ця технологія підходить і здатна значно пришвидшити процес розробки.
- NativeScript. За своєю ідеєю – ця технологія дуже схожа до попередньої, компанії Facebook. Ці технології конкурують між собою та займають приблизно одинакові місця в рейтингах популярності в сучасній розробці кросплатформенних додатків.

Будь яка із цих мов та технологій може бути використана для реалізація ідеї додатку

1.3.1.5 Xcode.

Xcode – це програмний додаток, на платформу MacOS, для розробки додатків мовою Objective-C або Swift, також він дозволяє розробляти додатки мовою C++.

Основним компонентом iOS додатку є такі компоненти:

UIApplicationDelegate. Основний клас який відповідає за життєвий цикл додатку. Дозволяє додатку реагувати на такі події як перехід в background та вихід з нього. Також він так званою точкою входу для додатку, що дозволяє налаштувати додаток та його компоненти.[8]

UIViewController. Клас який відповідає за конкретний екран програми та дозволяє налаштувати його під свої потреби. Він є зв'язним компонентом для UI та ViewModel. Цей клас дозволяє опрацювати різні події зв'язані із життєвим циклом екрану. Також він дозволяє здійснювати переходи між додатку та налаштування для цього.[9]

- Storyboard. Файл який містить намальований інтерфейс додатку, включаючи всі переходи та інші UI компоненти.
- Info.plist. В цьому файлі містяться налаштування проекту, такі як необхідні дозволи додатку, версію додатку, налаштування локалізації, підтримувані режими орієнтації екрану і т.д.
- Assets.xcassets – містить зображення, які використовуються в додатку.

Інтерфейс Xcode.

Основний екран програми Xcode зображено на Рисунку 1.9.

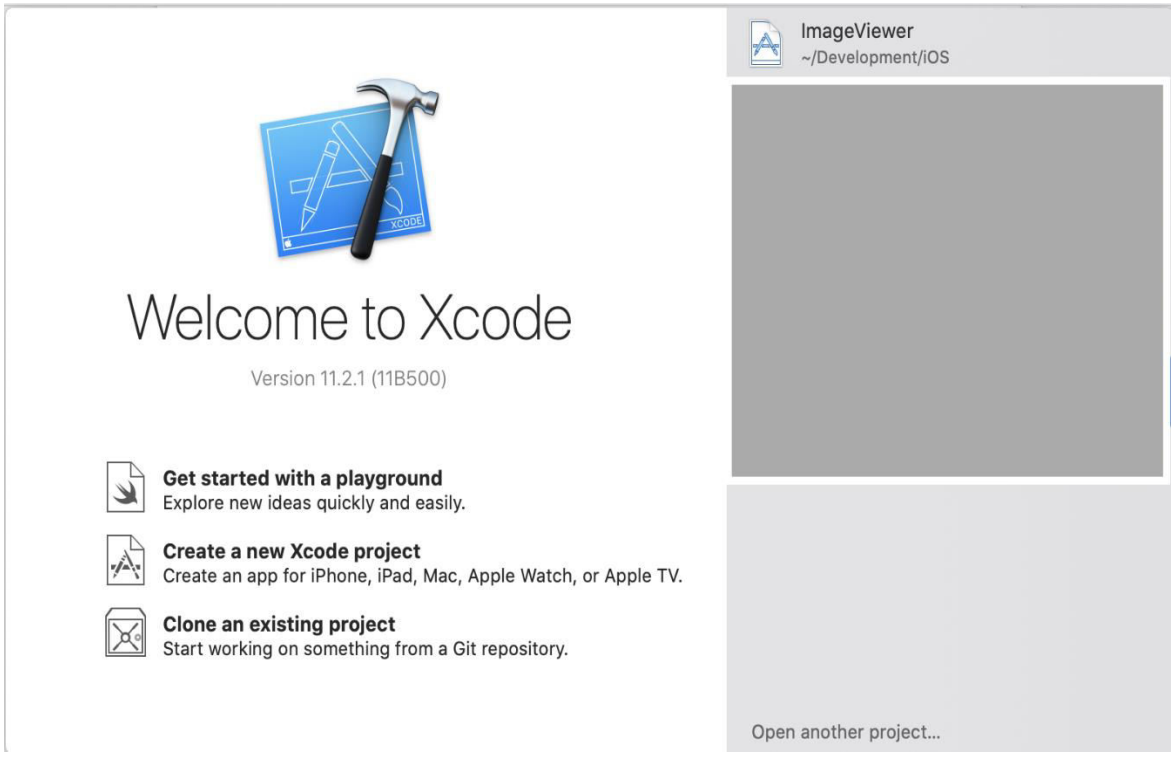

Рисунок 1.9. Головний екран програми Xcode.

Рисунок. Головний екран програми Xcode.

Головний екран дає доступ до таких функцій:

1) Запуск та перевірка певних можливостей мови Swift (на рисунку цей пункт називається «Get started with playground»).
2) Створення нового проекту («Create a new Xcode project»). Переходить на екран створення нового проекту із подальшим його налаштуванням.
3) Отримання проекту із сховища Git («Clone an existing project»). Дозволяє завантажити проект із сховища Git за допомогою графічного інтерфейсу Xcode.
4) Справа знаходиться пункт із списком існуючих проектів, який дозволяє відкрити раніше створені або завантажені проекти.

При кліку на пункт 2 – відкриється екран, зображений на Рисунку 1.10.

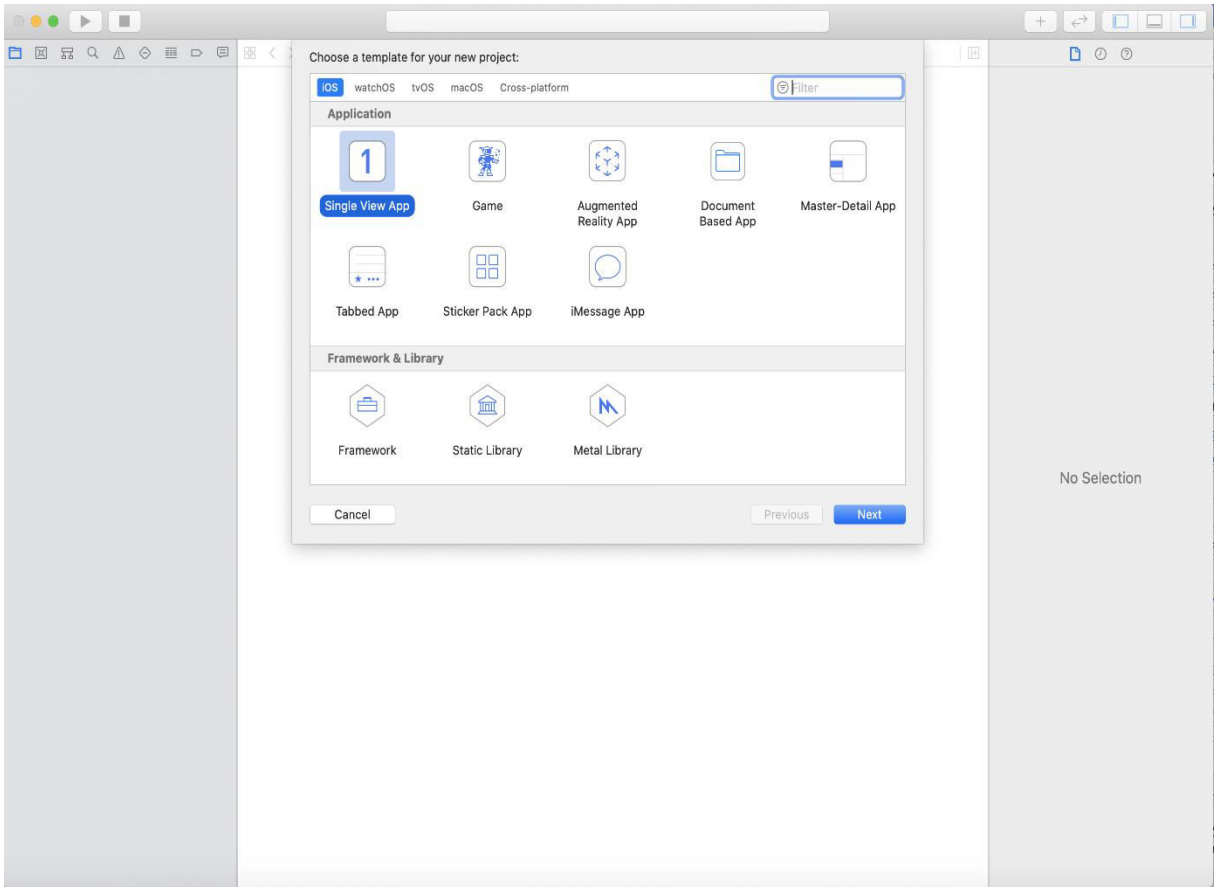

Рисунок 1.10. Екран створення нового проекту.

Цей екран дозволяє обрати необхідні налаштування для майбутнього проекту. Для обирання необхідно визначитися із функціоналом проекту. В користувача є вибір із підготовлених наперед шаблонів проектів, для iOS, watchOS, tvOS, macOS та Cross-Platform. Це описує всі можливі варіанти платформ від Apple.

Розглянемо варіанти налаштувань проекту для платформи iOS:

1) Single View App. Створення проекту, який має наперед створені файли для роботи із простим додатком (приклад створеного проекту на Рисунку 1.11). Цей варіант налаштувань є одним із найпопулярнішим, оскільки передбачає створення додатку без лишньої інформації. Це дозволяє швидше приступити до розробки швидше та розпочати роботу із майбутнім додатком.

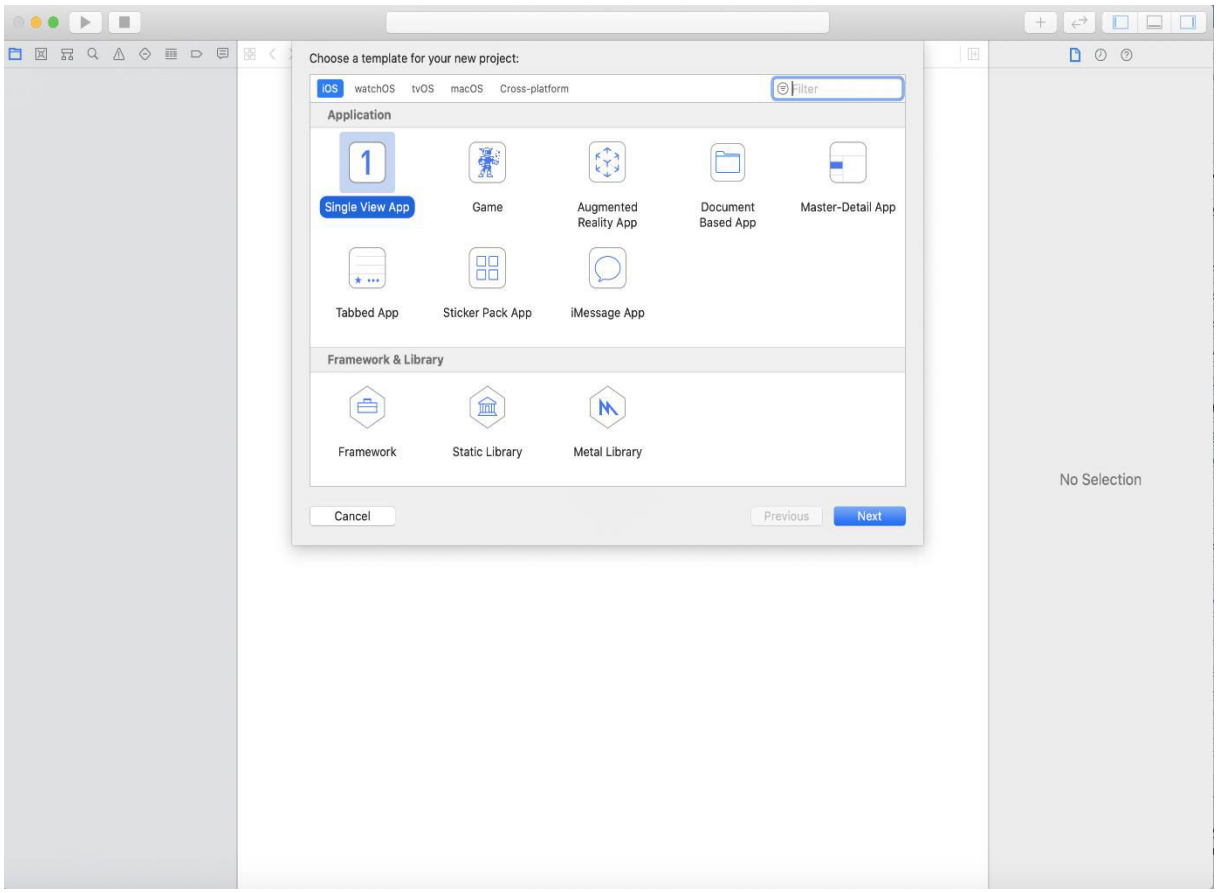

Рисунок 1.11. Приклад створеного Single View App проекту для iOS.

Як видно на рисунку – проект включає простий набір файлів, таких як AppDelegate, ViewController, Storyboard, Assets та info.plist.

1) Game. Створення проекту, який містить необхідні компоненти для майбутньої ігри.

2) Augmented reality. Проект із налаштуваннями для розробки додатку, із можливістю доповненої реальності.

3) Інші варіанти схожі між собою та також створюють проект із початковими налаштуваннями під кожний випадок.

Після вибору необхідного варіанту – відкривається екран, який дозволяє вибрати місце, де буде створений проект на локальному диску, команду, яка розроблятиме додаток, назву проекту, вибір мови на якій вестиметься розробка, вибір чи буде використовуватися CoreData (iOS база даних), та, з новою версією Xcode, можливість вибору інструменту для розробки користувацького інтерфейсу (Storyboard, SwiftUI) .

Цей екран показано на Рисунку 1.12

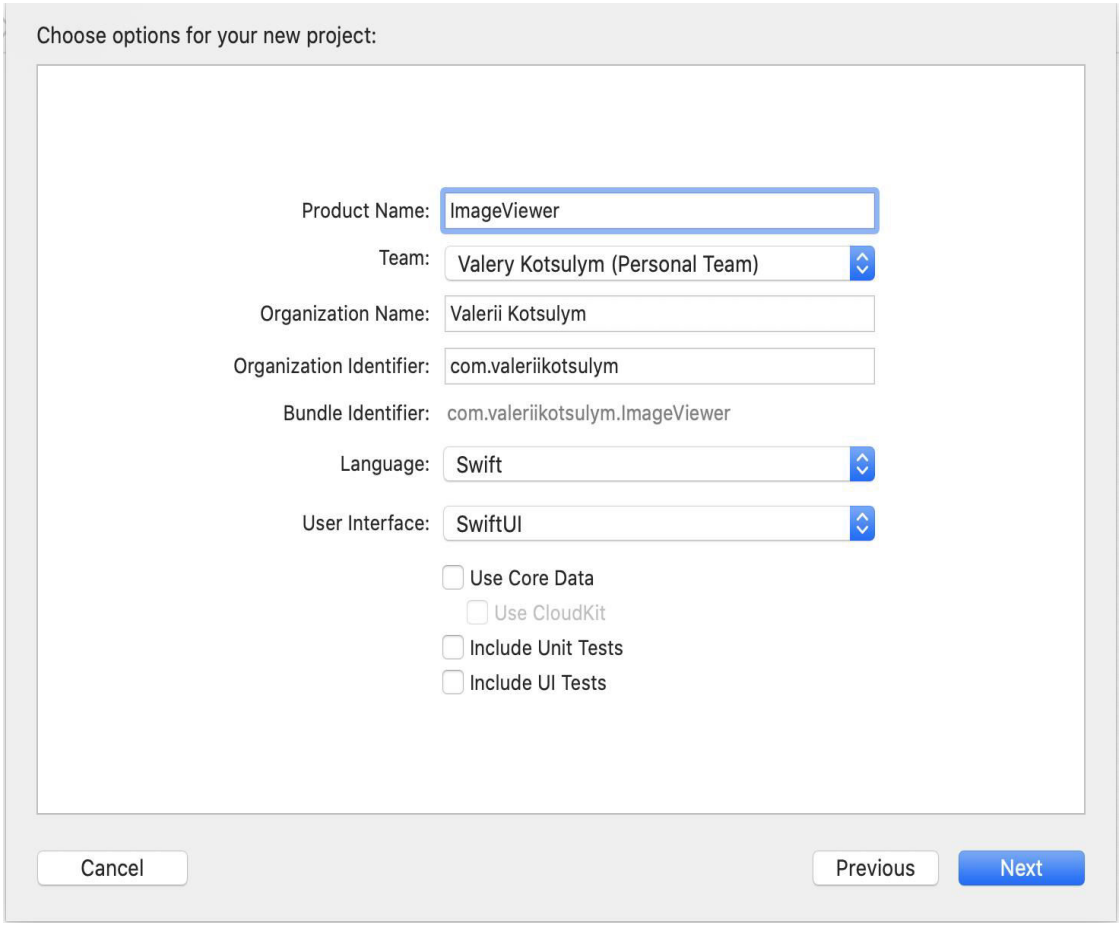

Рисунок 1.12. Наступний екран створення проекту для iOS.

За допомогою цих кроків – можна створити локальний проект, який в подальшому можна завантажити до віддаленого сховища Git.

На жаль даний момент не має інших варіантів для розробки під платформу iOS, та доступним є тільки Xcode.

1.4 Використання програмної системи.

Основними системними вимогами є платформа iOS версії не нижче 12.0 та будь-який мобільний телефон/планшет. Для установки тестової версії додатку – необхідно спершу додати пристрій до сертифікату, яким підписується додаток, або оплатити вартість облікового запису розробника у Apple developer program. Вартість такого облікового запису дорівнює 100$, та потребує оновлення підписки щороку. Оскільки цей додаток розроблений в навчальних цілях – немає гострої необхідності оплачувати та використовувати його.

1.4.1 Розгортання програмної системи та системні вимоги.

Основною вимогою для пристрою – використання платформи iOS версії 13.+, що є оптимальним для сучасних додатків. Пристрій повинен бути із передньою камерою, для можливості розпізнавання жестів. На даний момент підтримка iPad та Apple Watch не доступна, однак в майбутньому можлива інтеграція. Ось інструкція завантаження додатку в магазин AppStore:

1) Авторизуватися на веб сервісі https://developer.apple.com/ (Рис). Для того щоб продовжити необхідно обрати пункт Account із наступного Рисунку 1.13.

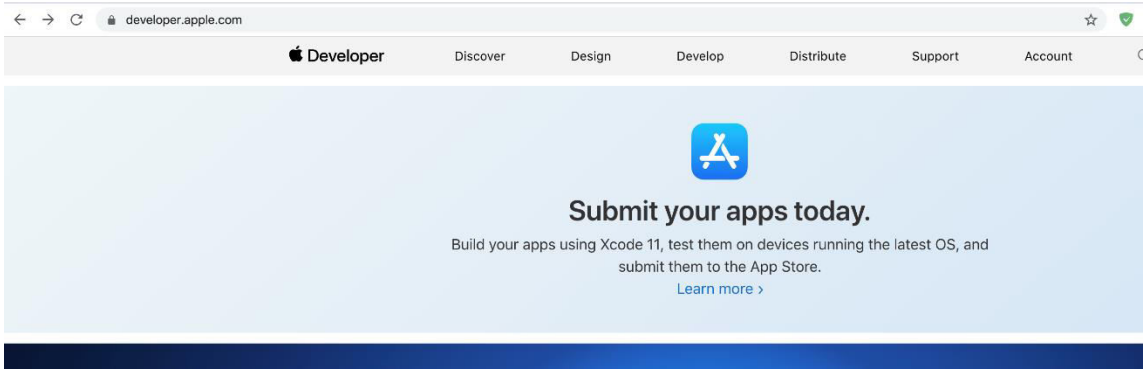

Рисунок 1.13. Головна сторінка сервісу для розробника додатків на платформи від компанії Apple

2) Ввести дані для входу або реєстрації нового профілю розробника.
3) Головний екран сервісу для розробників від компанії Apple зображено на Рисунку 1.14.

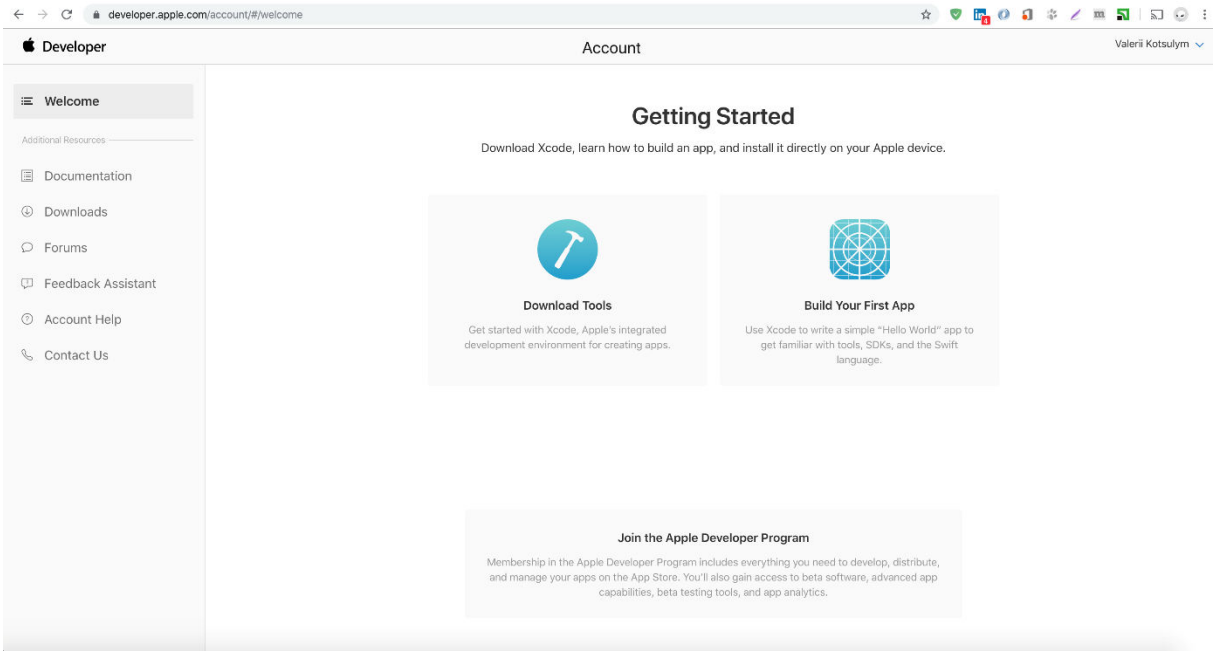

Рисунок 1.14. Головний екран сервісу для розробників від компанії Apple.

Після кліку по пункту Join the Apple Developer Program користувач побачить сторінку для заповнення інформації про себею Цей екран зображено на Рисунку 1.15

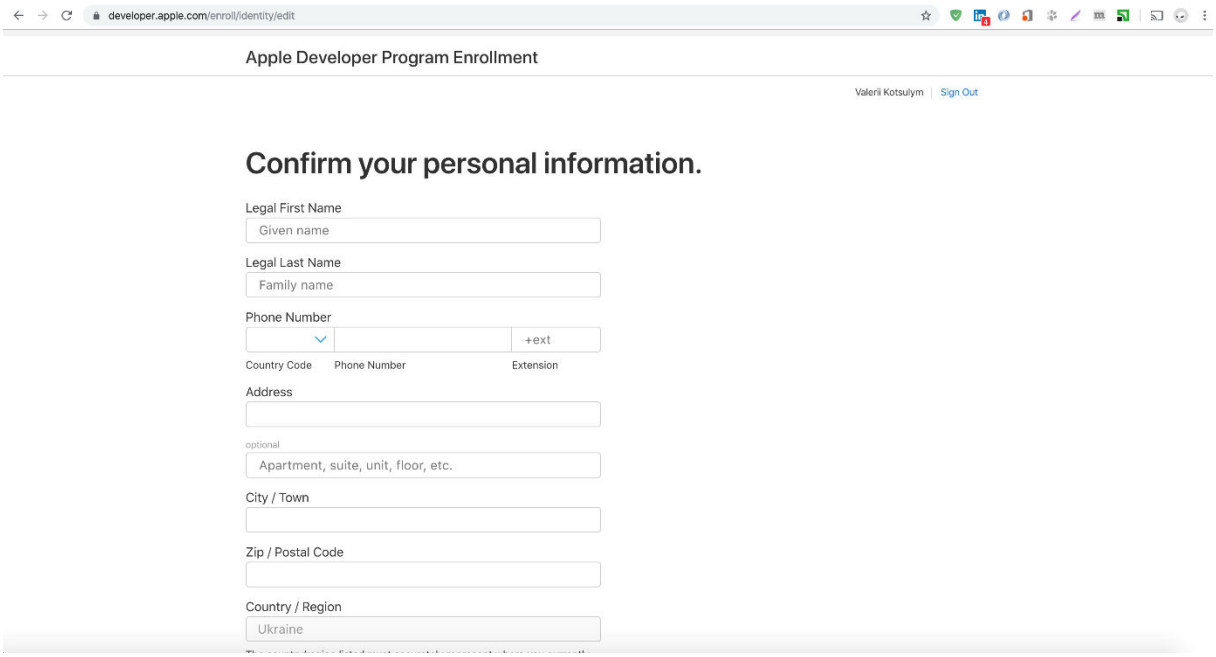

Рисунок 1.15. Екран для реєстрації в програмі для розробників.

4) Наступним кроком буде вибір типу майбутнього профілю, зображено на Рисунку 1.16

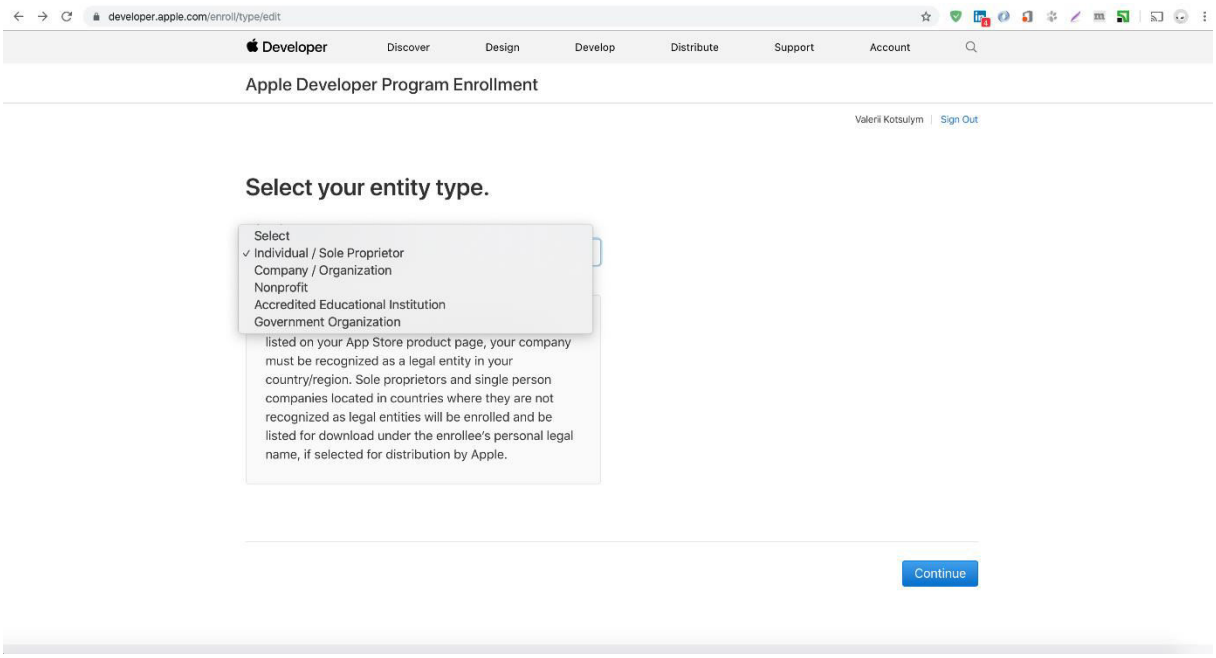

Рисунок 1.16. Екран вибору типу для майбутнього профілю.

5) Після цих дій необхідно переглянути умови та продовжити у випадку згоди. Можна переглянути на рисунку 1.17

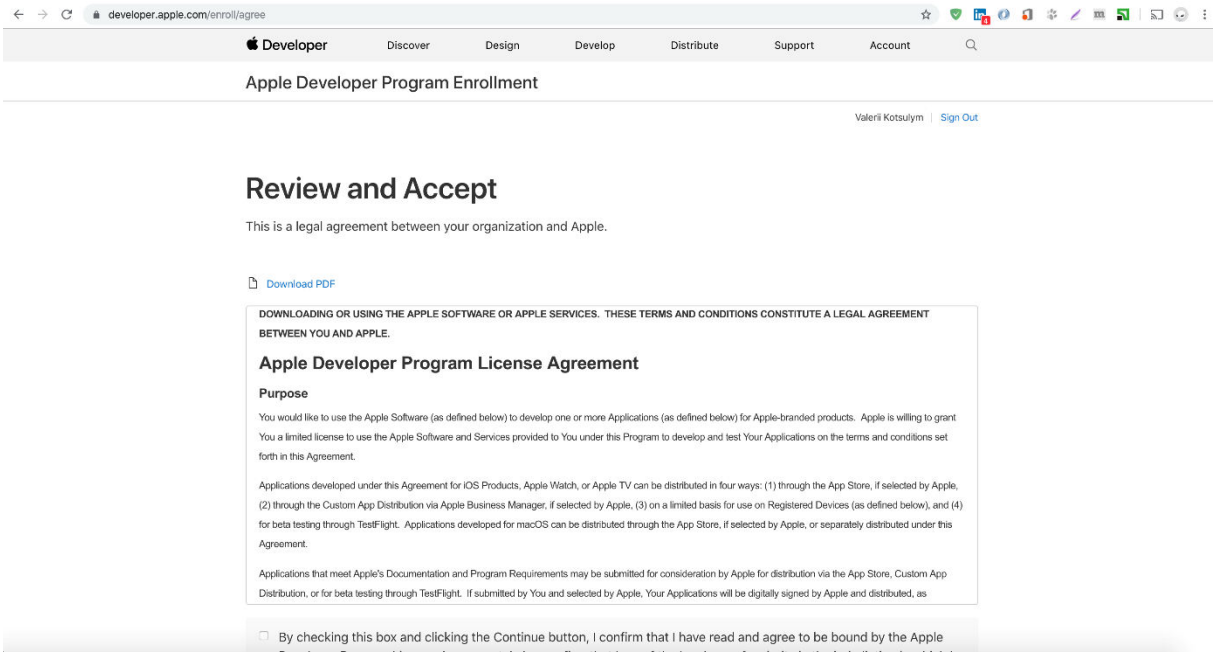

Рисунок 1.17. Ліцензія та умови участі в програмі для розробників.

6) Останній екран до якого є доступ це екран оплати участі в програмі для розробників Apple (Рисунку 1.18).

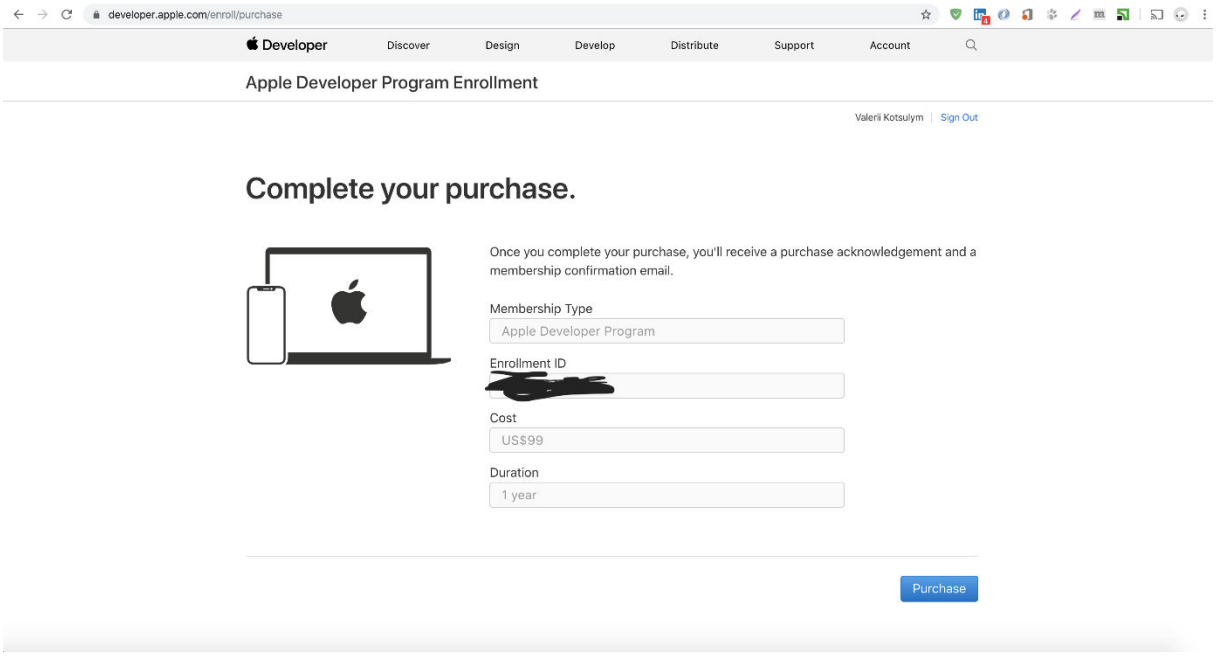

Рисунок 1.18. Екран завершення оплати участі в програмі

2. ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Загальний опис тестування.

Для пришвидшення тестування програмної системи було прийнято рішення написати тести для автоматизації тестування. XCode та мова swift дозволяє створювати тести як для тестування користувацького інтерфейсу, так і для тестування функціоналу. Це дозволяє покривати тестами як малі одиниці додатку так і більшу комплексну річ, таку як feature.

В 2019 році – компанія випустила новий фреймворк для роботи з тестами, який називається XCTest.

До Xcode 11 тестова конфігурація була частиною Xcode схем. Розробники повинні були конфігурувати різні набори тестів для різних умов, та редагували існуючу схему під певні тестові необхідності. Тестова конфігурація була тісно пов'язана з Xcode схемами, тому доводилося створювати безліч схем для для різних завдань при тестуванні.

Нова функціональність в Xcode 11 дозволяє розробникам і QA інженерів конфігурувати тести відповідно до своїх потреб. Test Plan дозволяє визначити:

- які тести запустити в збірнику;
- як запустити ці тести (наприклад, в випадковому або алфавітному порядку);
- як управляти тестовими артефактами (наприклад, аттачментами і скріншотами)
- як використовувати runtime інструменти як частина тестування.

З Xcode Test Plan можна конфігурувати тест інструменти незалежно від Xcode схем. Це значне поліпшення в технологіях тестування, яке може вивести тести на новий рівень.

Писати одиничні тести для вашого коду вважається корисним підходом в програмуванні. Ви знаєте, що це добре для вас - і всі експерти кажуть, що це правильно робити - але іноді на вашому шляху можуть виникати всілякі перешкоди.

Для того, щоб перетворити написання тестів на звичку, необхідний хороший набір інструментів. Розробляючи для платформ Apple, ви можете скористатися багатьма вбудованими тестовими рамками Xcode для початку роботи. Якщо вам зручно з цими інструментами, ви можете почати розвивати їх і поступово збільшувати кількість написаних тестів, роблячи їх частиною вашого нормального робочого процесу розвитку.

Лише думка про необхідність написати цілий набір тестів може бути приголомшливою для деяких розробників. Тести не обов'язково є "цікавою" частиною написання програми. Однак можуть виникнути можливості, коли ви робите більше, ніж прості перевірки додатку у своїх тестах. Для вирішення цієї проблеми необхідно спробувати розглядати тестовий код як звичайний код.

Коли почнемо розглядати свої тести як код, який можна оптимізувати та упорядкувати, працювати над ними стає цікавіше. Можна знайти розумні способи тестування складних умов в тестах інтеграції, придумати способи ініціалізації загального стану або додати цікаві умови помилок за допомогою глузливих помилок або викликів мережевих заглушок.

Одним зручним трюком для зменшення кількості написаного коду є створення buildSut()методу, який повертає об'єкт, що тестується одиницею ("Sut" означає "System Under Test" або "Subject Under Test", залежно від програмного забезпечення). Наявність такого об’єкта, створеного в одному місці, значно спрощує оновлення тестів у (майже неминучому) випадку, якщо до ініціалізатора додаються нові аргументи, або потрібно зробити додаткові кроки для правильної настройки вашого Sut.

Також можна використовувати відладчик всередині Xcode під час роботи через невдалі тести так само, як і з кодом програми. Точки перерви як у тестовому коді, так і у вашому цільовому додатку будуть дотримані, а також ви можете додати заяви про реєстрацію, де це необхідно.

Реєстрація операторів може бути простішим способом налагодження асинхронних або залежних від часу тестів. Якщо ви використовуєте точки перерви разом із очікуванням очікувань, не забудьте збільшити тривалість

очікування для будь-яких запитів. Часто змінювати щось із тайм-ауту 1.0на 10000.0 - це швидкий і простий спосіб переконатися, що ваші тести не закінчяться передчасно, оскільки ви сидите в налагоджувачі.

2.2 Тестування створеного додатку.

Основним функціоналом створеного додатку є використання штучного інтелекту для керування жестами. Отже всі тести та перевірки повинні бути націлені саме на цей фукнціонал.

Для впевненості в тому, що додаток працюватиме правильно – необхідно перевірити всі основні те не основні варіанти використання, а саме:

- Перегляд зображення на повний екран
- Перегляд альбому
- Перегляд інформації про альбом
- Перегляд зображень з альбому
- Керування жестами на екрані перегляду альбому
- Керування жестами на екрані повноекранного перегляду зображення.

Кожна ітерація розробки додатку має етап тестування, як заключний етап перед закінченням, отже весь функціонал був добре перевірений та протестований.

2.3 Результат тестування.

Кожний етап був перевіреним, та більшість знайдених проблем було виправлено в ході розробки. Однією із не виправлених проблем залишилася проблема із обробкою декількох жестів одночасно (наприклад zoom + переключення картинки). Ця проблема взята до уваги та розпочато дослідження щодо шляхів її виправлення.

3. ОРГАНІЗАЦІЙНО ЕКОНОМІЧНА ЧАСТИНА

3.1 Загальний підхід до визначення економічної ефективності розробки

Розробка будь-якого проекту неможлива без фінансових витрат, які необхідні для фінансування кожного з етапів розробки. Одним з основних аспектів, які відображають успішність проекту є його економічна ефективність. Оцінка економічної ефективності розробленого програмного продукту, розрахунок передбачених витрат та оцінка ризиків непередбачуваних витрат є головною метою даного розділу.

Для виконання проекту може буде залучено 1 керівника, 2 розробники та 1 тестувальник. На проекти такою складності виділяється 160 робочих годин, тому в подальшому будемо орієнтуватися саме на таку часову одинцю.

Витрати часу по окремих етапах розробки програмного забезпечення відображено в таблиці 3.1.

Таблиця 3.1 – Операції процесу розробки ПЗ і часові затрати

	Місячна зарплата грн.	Денна зарплата грн	Трудомісткість, людино-днів		Основна заробітня плата, грн	
			Розробка мовою Swift	Розробка мовою Objective-C	Розробка мовою Swift	Розробка мовою Objective-C
Керівник	9000	450	1	1	450	450
Розробник	12500	625	14	20	8750	12500

Продовження до табл 3.1

Тестувальник	8000	400	5	5	2000	2000
Всього			20	26	11200	14950

При розробці програмного забезпечення на платформу iOS немає високої необхідності в керівнику, оскільки його роль полягає в налагоджені робочого процесу. На малих проектах та з малою командою - налагодження процесу є швидкою процедурою.

3.2 Визначення ключових витрат.

Першою ключовою витратою є заробітна плата.

Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно-ділових якостей працівника, результатів його праці та господарської діяльності підприємства. Заробітна плата складається з основної та додаткової оплати праці.

Основна заробітна плата нараховується на виконану роботу за тарифними ставками, відрядними розцінками чи посадовими окладами і не залежить від результатів господарської діяльності підприємства.

$ЗП_{осн1} = 11200$ грн;

$ЗП_{осн2} = 14950$ грн;

Додаткова заробітна плата – це складова заробітної плати працівників, до якої включають витрати на оплату праці, не пов’язані з виплатами за фактично відпрацьований час. Нараховують додаткову заробітну плату залежно від досягнутих і запланованих показників, умов виробництва, кваліфікації виконавців. Джерелом додаткової оплати праці є фонд матеріального стимулювання, який створюється за рахунок прибутку.

$$ЗП_{дод} = 0{,}2 \cdot ЗП_{осн} \quad (3.1)$$

$ЗП_{дод1} = 0{,}2 \cdot ЗП_{осн1} = 2240$ грн.;

$ЗП_{дод2} = 0{,}2 \cdot ЗП_{осн2} = 2990$ грн.;

Таким чином загальний фонд заробітної плати, що обчислюється за формулою:

$$ФЗП = ЗП_{осн} + ЗП_{дод}. \quad (3.2)$$

$$ФЗП_1 = ЗП_{осн1} + ЗП_{дод1} = 13440 \text{ грн.};$$

$$ФЗП_2 = ЗП_{осн2} + ЗП_{дод2} = 17940 \text{ грн.};$$

Крім того, слід визначити відрахування на соціальні заходи:

- єдиний соціальний внесок – 3,6 %;
- військовий збір – 1,5 %;
- ПДФО (прибутковий податок) – 15 %.

Отже, сума відрахувань на соціальні заходи буде становити:

1) Єдиний соціальний внесок:

$$Відр_{ЄСВ1} = 0{,}036 \cdot ФЗП_1 = 483{,}84 \text{ грн.};$$

$$Відр_{ЄСВ2} = 0{,}036 \cdot ФЗП_2 = 645{,}84 \text{ грн.};$$

2) Військовий збір:

$$Відр_{ВЗ1} = 0{,}015 \cdot ФЗП_1 = 201{,}6 \text{ грн.};$$

$$Відр_{ВЗ2} = 0{,}015 \cdot ФЗП_2 = 269{,}1 \text{ грн.};$$

3) Прибутковий податок (ПДФО):

$$Відр_{ПДФО1} = 0{,}15 \cdot ФЗП_1 = 2016 \text{ грн.};$$

$$Відр_{ПДФО2} = 0{,}15 \cdot ФЗП_2 = 2691 \text{ грн.};$$

Нарахування на фонд оплати праці, які включають відрахування до Пенсійного фонду, фонду з тимчасової втрати працездатності, фонду з безробіття і фонду страхування від нещасних випадків на виробництві; для бюджетної організації тариф на фонд оплати праці встановлено на рівні 36,3%.

Зокрема, видання програмного забезпечення – 36,77%.

Нарахування на Фонд оплати праці (ФОП):

$$ФОП_{ЄСВ} = 0{,}3677 \cdot ФЗП \quad (3.3)$$

$$ФОП_{ЄСВ1} = 0{,}3677 \cdot ФЗП_1 = 4941{,}89 \text{ грн.};$$

$$ФОП_{ЄСВ2} = 0{,}3677 \cdot ФЗП_2 = 6596{,}54 \text{ грн.};$$

Всього витрат:

$$В_{ЗП1} = ФЗП_1 + ФОП_{ЄСВ1} + Відр_{ЄСВ1} + Відр_{вз1} + Відр_{ПДФО1} = 21083{,}33 \text{ грн.};$$

$$В_{ЗП2} = ФЗП_2 + ФОП_{ЄСВ2} + Відр_{ЄСВ2} + Відр_{вз2} + Відр_{ПДФО2} = 28142{,}48 \text{ грн.};$$

Також важливою складовою витрат є матеріальні витрати. Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни:

$$М_{ві} = q_i \cdot p_i, \quad (3.4)$$

де: q_i – кількість витраченого матеріалу і–го виду;

p_i – ціна матеріалу і–го виду.

Звідси, загальні матеріальні витрати можна визначити:

$$З_{м.в.} = \sum М_{ві}. \quad (3.5)$$

Таблиця 3.2 – Зведені розрахунки матеріальних витрат.

Найменування матеріальних ресурсів	Один. Виміру	Фактично витрачено матеріалів	Ціна 1-ці., грн.	Загальна сума витрат, грн
Папір формату А4	листів	500	0,5	250
Маркер	шт	10	6	60
Тонер для принтера	шт	2	80	160
Флеш-накопичувач	шт	4	80	320
Всього				790

Отже, загальна сума матеріальних витрат становить 790 гривень.

В багатьох випадках існує ряд додаткових витрат які пов’язані із реалізацією проекту, але в більшості випадків їхня сума не перевищує десяти відсотків від загальної собівартості реалізації проекту.

Також варто врахувати електроенергію. Затрати на електроенергію використану 1-цею обладнання визначаються за формулою:

$$З_е = W \cdot T \cdot S, \quad (3.6)$$

де *W* – необхідна потужність, кВт;

T – кількість годин роботи обладнання;

S – вартість кіловат-години електроенергії.

Вартість кіловат-години електроенергії слід приймати згідно існуючих на даний час тарифів. Отже, 1 кВт з ПДВ коштує 2,50 грн.

Потужність комп’ютера для створення проекту – 150 Вт, кількість годин роботи необхідних для проекту – 160 годин.

$$З_{е1} = 0{,}15 \cdot 160 \cdot 2{,}50 = 60$$

$$З_{е3} = 0{,}15 \cdot 208 \cdot 2{,}50 = 78$$

Характерною особливістю застосування основних фондів у процесі виробництва є їх відновлення. Для відновлення засобів праці у натуральному виразі необхідне їх відшкодування у вартісній формі, яке здійснюється шляхом амортизації.

Для визначення амортизаційних відрахувань застосовуємо формулу:

$$A = \frac{С_Б \cdot N_A \cdot T_{ФАК}}{T_{год}} \quad (3.7)$$

де $С_Б$ – балансова вартість обладнання, грн;

N_A – норма амортизаційних відрахувань в рік, %;

$T_{год}$ – річний робочий фонд часу, год;

$T_{ФАК}$ –фактичний час роботи обладнання по написанню програми, год.

Комп’ютери та оргтехніка належать до четвертої групи основних фондів. Для цієї групи річна норма амортизації дорівнює 60 % (квартальна – 15 %).

Отже, використовуючи в роботі 1 комп’ютер балансовою вартістю 20000 грн. Отже, амортизаційні відрахування будуть рівні:

A_1 =(20000 ·0,6·160/2080 = 923,08 грн.

A_2 =(20000 ·0,6·208)/2080 = 1200 грн.

Варто врахувати і накладні витрати, адже вони пов’язані з обслуговуванням виробництва, утриманням апарату управління спілкою та створення необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 20-60 % від суми основної та додаткової заробітної плати працівників.

$$НВ = 0{,}5 \cdot ЗП_{осн} \quad (3.8)$$

де $H_в$ – накладні витрати.

Отже, накладні витрати:

$Н_{в1} = 11200 \cdot 0{,}5 = 5600$ грн.

$Н_{в2} = 14950 \cdot 0{,}5 = 7475$ грн.

4.3 Визначення періоду окупності та собівартості

Проведемо розрахунок вартості створюваного програмного продукту. Вартість продукції включає у собі собівартість і планований прибуток.

Собівартість продукції – це сума грошових витрат підприємства (фірми) на виробництво і збут одиниці продукції, виконання робіт та надання послуг.

Повна собівартість програмного продукту дорівнює сумі усіх витрат на його виробництво: 27666.41 грн (Св1) використовуючи Swift, 36895.48 грн (Св2) використовуючи Objective-C.

Приймемо прибуток на рівні 30%. Для нових інноваційних продуктів, що користуються високим попитом на ринку, ринкову вартість Вр можна встановити вищу.

Отже, вартість розробленого програмного забезпечення:

$Вр_1 = Св_1 + 0{,}3 \cdot Св_1 = 35966{,}33$ грн.

$Вр_2 = Св_2 + 0{,}3 \cdot Св_2 = 47964{,}11$ грн.

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів

праці на підприємстві за певний проміжок часу. Економічна ефективність ($E_р$) полягає у відношенні результату виробництва до затрачених ресурсів:

$$E_р = \frac{П}{C_В} \tag{3.9}$$

де $П$ – прибуток;

$C_в$ – собівартість.

Плановий прибуток ($П_{пл}$) знаходимо за формулою:

$$П_{пл} = В_р - C_в \tag{3.10}$$

Розраховуємо плановий прибуток:

$П_{пл1} = В_{р1} - С_{в1} = 35966{,}33 - 27666{,}41 = 8299{,}92$ грн.

$П_{пл2} = В_{р2} - С_{в2} = 47964{,}11 - 36895{,}48 = 11068{,}63$ грн.

Отже, формула для визначення економічної ефективності набуде вигляду:

$$E_р = \frac{П_{пл}}{C_в}. \tag{3.11}$$

$Е_{р1} = 8299{,}92 / 27666{,}41 = 0{,}3$

$Е_{р2} = 11068{,}63 / 36895{,}48 = 0{,}3$

Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень ($T_р$):

$$Т_{ок} = \frac{1}{Е} \tag{3.12}$$

Термін окупності дорівнює:

$Т_{ок} = 1 / 0{,}3 = 3{,}3$ роки

У нашому випадку Ток1= Ток2 =1/0,30=3,33 років, що є нормальним, оскільки допустимим вважається термін окупності до 5 років.

Даний розрахунок виконаний у розрахунку на 1 екземпляр програмного продукту без врахування його тиражування.

Загальна вартість пропонованих робіт по розробці програмного продукту становить 20500 грн. Оскільки ефективність для обидвох проектів відповідно до встановленого рівня прибутку становить 0,3, що є високим показником, то проводити дані роботи варто і вкладені кошти

окупляться за 3 роки та три місяці. Також слід врахувати можливість не одиничного замовлення програми, відповідно її ціна в такому випадку значно понизиться, а при продажі понад план прибуток зросте.

Виходячи із експертних оцінок і складності програми, приймемо величину витрат на супровід і модернізацію програмного забезпечення, створеного за допомогою мови Swift 20% від початкових витрат, а за Objective-C 50%.

Собівартість модернізації:

$С_в М_1 = 0{,}2 \cdot С_{в1} = 0{,}2 \cdot 27666{,}41 = 5533{,}29$ грн.,

$С_в М_2 = 0{,}5 \cdot С_{в2} = 0{,}5 \cdot 36895{,}48 = 18447{,}74$ грн.

Для споживача вартість модернізації:

$М_1 = 0{,}2 \cdot В_1 = 0{,}2 \cdot 35966{,}33 = 7193{,}27$ грн;

$М_2 = 0{,}5 \cdot В_2 = 0{,}5 \cdot 47964{,}11 = 23982{,}06$ грн.

Таким чином, уже після першої модернізації, загальні витрати на створення і супровід ПЗ для виробника який використовує мову Swift менші ніж для виробника який використовує Objective-C.

$ЗВ_{1\,(вир)} = 27666{,}41 + 5533{,}29 = 33199{,}7$ грн.;

$ЗВ_{2\,(вир)} = 36895{,}48 + 18447{,}74 = 55343{,}22$ грн..

Як і для споживача:

$ЗВ_1 = 35966{,}33 + 7193{,}27 = 43159{,}6$ грн.;

$ЗВ_2 = 47964{,}11 + 23982{,}06 = 71946{,}17$ грн..

Річна економія витрат за всіма можливими напрямами і додатковими витратами, пов'язаними з супроводом і тільки одноразовою модернізацією (у розрахунку на одиницю продукції) при розробці різними мовами програмування:

$ΔС_{(вир)} = ЗВ_{2\,(вир)} - ЗВ_{1\,(вир)} = 55343{,}22 - 33199{,}7 = 22143{,}52$

$ΔС = ЗВ_2 - ЗВ_1 = 71964{,}17 - 43159{,}6 = 28804{,}57$ грн..

Чистий приведений дохід (ЧПД) визначається як різниця між сукупними доходами (сукупний грошовий потік) і сукупними витратами

(сукупними інвестиціями) взятими за весь період життя інвестицій і дисконтова ними в кожному році на фактор часу.

Для розрахунку коефіцієнта дисконтування (коефіцієнта приведення) грошових потоків за роками періоду економічного життя інвестицій використовується формула:

$$\alpha = \frac{1}{(1+\mathrm{i})^n}; \qquad (3.13)$$

де i – ставка дисконтування або норма дисконту, i = 0,2;

n – час або кількість періодів (років), протягом якого планується отримання доходу.

$\alpha_0 = 1, \alpha_1 = \frac{1}{1+0,2} = 0,60.$

Вважатимемо, що обидва програмних продукти однаково забезпечують потреби і вимоги споживача, і тому придбання першої чи другої програми однаково вплинуть на розмір його додаткових доходів на вкладений капітал. Тому приймемо цю величину за постійну, а порівняння дохідності двох проектів проведемо тільки за витратами. Чим менші витрати, тим більша дохідність проекту.

$ЗВ_1 = 19500 + 7090 = 26590$ грн.;

$ЗВ_2 = 19500 + 3900 = 23400$ грн.

Таблиця 3.3 – Техніко–економічні показники програмного продукту

Показник	Розробка мовою Swift	Розробка мовою Objective-C
Зарплата основна, грн	11200	14950
Зарплата додаткова, грн	2240	2990
Фонд заробітньої плати, грн	13440	17940
Відрахування на ФОП, грн	4941,88	6596,54
Військовий збір 1,5%	201,6	269,1
Єдиний соціальний внесок 3.6%	483,84	645,84
ПДВ, 15%	2016	2691

Разом на виплату праці, грн	21083,33	28142,48
Матеріальні витрати, грн	790	790

Продовження до таблиці 3.3

Електроенергія, грн	60	78
Амортизація, грн	923,08	1200
Накладні витрати, грн	5600	7475
Разом на ін.витрати, грн	7373,08	9543
Собівартість	27666,41	36895.48
Прибуток	8299,92	11068,63
Вартість розробленого ПЗ	35966,33	47964,11
Економічна ефективність	0,30	0,30
Термін окупності, років	3,33	3,33
Собівартість модернізації	5533,29	18447,74
Загальні витрати на розробку	41499,62	66411,85
Економія	24912,23	-

Економія витрат у випадку придбання, супроводу і одноразової модернізації програмного продукту, створеного допомогою мови Swift, становить 24912,23 грн.

Оскільки ефективність для обидвох проектів відповідно до встановленого рівня прибутку становить 0,3, що є високим показником, то проводити дані роботи варто і вкладені кошти окупляться за 3 роки та два місяці, бо нормальним терміном окупності є термін, який коливається від 1 до 3 років, в даному випадку допуск 2 місяці можна вважати допустим, тоді розробка вважається доцільною і економічно вигідною.

При використанні Swift зменшуються витрати на реалізацію проекту, але для підтримки проекту і його подальшої модернізації.

Отже, програмний продукт може бути впроваджений та мати подальший розвиток, оскільки він є економічно вигідною за всіма основними техніко-економічними показниками.

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці.

Сучасний етап розвитку суспільства неможливо розглядати окремо від інформаційних систем. Кожен нещасний випадок, порушення охорони праці чи інформаційні помилки мають одну спільну причину-це людський фактор. Безпека суспільства належить не технологіям чи техніці, а людині. Людський фактор це основа соціальних якостей людини, які історично склалися в суспільстві.

На даний час в Україні зареєстровано більше мільйона компаній. Тому кожне підприємство має свої інструкції. Кожна компанія для покращення робочого процесу, повинна забезпечити безпечну працю громадянам, гігієну праці та здорового виробничого середовища.

Відповідно до закону України, а саме статті 13 Закону України “Про охорону праці” передбачається вимога, щодо впровадження заходів з охорони праці .Кожна компанія відповідно до нормативно правових актів в рамках якої реалізуються трудові відносини, зобов’язана вживати всіх необхідних заходів з охорони праці та використовують відповідну документацію.

У нормативно правових актах передбачено:

- положення про охорону працю;
- інструкції з охорони праці по кожній професії;
- накази з охорони праці;
- журнал реєстрації інструктажів.

Тому однією з найбільш важливих подій 2018 року стало впровадження в Європейському союзі Загального регламенту про захист даних(GDPR).

Норми охорони праці в Україні

Наказом Мінсоцполітики 14.02.2018 № 207, який вступив в силу 18.05.2018 р., затверджені Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями.

Ці Вимоги поширюються на всіх суб'єктів господарювання незалежно від форм власності, організаційно-правової форми і видів діяльності та встановлюють мінімальні вимоги безпеки та захисту здоров'я під час здійснення роботи, пов'язаної з використанням екранних пристроїв незалежно від їхнього типу та моделі.

Вимоги до приміщень.

Приміщення, в яких планується установка та подальша робота з комп'ютером, повинні відповідати проектній документації будинку, погодженій з уповноваженими державними органами. Крім того, роботодавець повинен враховувати санітарні нормативи освітлення, вимоги до параметрів мікроклімату (температура, відносна вологість), ступеня і сили вібрації, звукового шуму і вогнестійкості приміщення, а також характеристики електромагнітного, ультрафіолетового та інфрачервоного полів. Конкретні показники зазначених санітарних норм див. в Державних санітарих правилах і нормах роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПІН 3.3.2.007-98, затверджених Постановою Головного державного санітарного лікаря України №7 від 10 грудня 1998 року.

У кожній кімнаті, де обладнуватимуться робочі місця співробітників, що працюватимуть на комп'ютері, повинні бути наявні елементи природного та штучного освітлення. При цьому, на вікнах слід встановити легко регульовані жалюзі чи штори, які дозволять працівникам коригувати рівень освітлення в приміщенні.

4.2 Безпека в надзвичайних ситуаціях.

4.2.1 Освітлення виробничих приміщень для роботи з ВДТ.

В сучасному світі, у сфері діяльності людини відіграють велику і важливу роль інформаційні технології які базуються на використанні електронно-обчислювальної техніки і телекомунікаційних засобів. Основою цих технологій є інтегральний режим робити з візуальними дисплейними терміналами(ВДТ), інша назва ВДТ-комп'ютери. ВДТ бувають суспільного (колективного) використання та персональні. ВДТ є або об'єктом праці, або головним її засобом, або робочим інструментом. Суспільство широко використовує персональні комп'ютери.

Комп'ютери широко використовуються на підприємствах, наукових лабораторіях адже вони забезпечують надійну інформацію. Також зараз все більше і більше комп'ютерів використовують навчальні заклади, наприклад у дистанційному навчанні. Але і комп'ютери знайшли своє місце і повсякденному житті. Розроблено велику кількість розважальних програм, комп'ютерних енциклопедій, комп'ютерних ігор тощо…

Сьогодні в США за підрахунками використовують більше 100млн комп'ютерів. Отже комп'ютери використовують люди з із різним рівнем освіти, досвідом роботи, станом здоров'я і ступенем нейропсихічної стійкості.

Але через використання комп'ютерів зросли проблеми людей зі здоров'ям. Для збереження власного і суспільного здоров'я, вимагає удосконалення та розробки нових підходів для організації робочих місць, також проведення профілактичних заходів для запобігання розвитку негативних наслідків впливу ПК на здоров'я користувачів і працівників.

Негативні наслідки комп'ютерних технологій включають в себе:

- інтенсифікації темпу роботи й монотонності;

- ізоляція працівника у виробничому середовищі, обмеженні його контактів з іншими людьми;
- розвитку несприятливих психічних станів;
- великих нервових навантаженнях при незначних фізичних;
- перенапруження органу зору;

Розладу стану здоров'я спричиненого шкідливими факторами, джерелом якого є ВДТ.

У зв'язку з цим праця професійних користувачів ВДТ має свої особливості.

Наслідками праці за комп'ютерами є функціональні розлади аналізаторів, захворювання опорно-рухового апарату, нервової, серцево-судинної та інших систем організму.

Економічні втрати від таких хвороб оператора ВДТ, за підрахунками американських фахівців, можуть обійтися у 100 тисяч доларів.

Тому робота з ВДТ у багатьох країнах віднесена до списку шкідливих і небезпечних, а захворювання які виникають при роботі ВДТ називаються професійними.

Також значного поширення набуло специфічне захворювання, яке отримало назву синдром комп'ютерного стресу. СКС супроводжується головним болем, запалення очей, алергією, роздратованістю, млявістю, і депресією.

Захворювання очей та порушення зору є одним найбільш поширеними скаргами персоналу ВДТ. Частота порушень зору за підрахунками складає більше 15-20%.ніж серед працівників які не використовують комп'ютери. В США був уведений новий термін "астенопія". Симптоми класифікуються на очні (біль, печія, та різь в очах, почервоніння повік та очних яблук) та зорові(пелена перед очима. мерехтіння, швидка втома під час зорової роботи…).

Виникнення та розвиток патології зорової функції зумовлені:

- умовами зорової роботи на ВДТ;

- світлотехнічною різнорідністю об’єктів зорової роботи, багаторазове переведення лінії зору від одного об’єкту на інший;
- осліплююча дія світильників, які освітлюють приміщення на робочому місці.

Є чотири групи факторів які впливають на порушення зорових функцій:

1) параметри освітлення робочого місця;
2) характеристиками дисплея;
3) специфікою роботи на ВДТ;
4) неправильною організацією робочого місця.

Для зменшення шкідливого впливу ПК широко використовують профілактичні методи. Значну роль відводиться медицині.

Також зараз у нашій країні проводиться розробка національних документів, спрямованих на охорону праці користувачів ПК. Одним з них є “Державні санітарні правила і норми робити з візуальними дисплейними терміналами електронно-обчислювальних машин”.

Виконуючи ці правила можна знизити несприятливу дію на працівників шкідливих і небезпечних факторів, які супроводжують роботу з відео-дисплейними терміналами.

У Правилах викладені гігієнічні й ергономічні вимоги до організації робочих приміщень та робочих місці, параметрів робочого середовища, дотримання яких дає змогу запобігти порушенню стану здоров’я користувачів ПК.

Вимоги до виробничих приміщень для експлуатації ВДТ:

- розміщення робочих місць з ВДТ у підвальних приміщення, на цокольних поверхах заборонено;
- площа на одне робоче місце становить не менше 6,0м2, а об’єм - не менше ніж 20.0м2;

- приміщення для роботи повинно мати природне і штучне освітлення;

Природне освітлення повинно здійснюватися через світлові прорізи, орієнтовані на північ, а коефіцієнт природної освітленості повинен становити не нижче, ніж 1,5%.

- виробничі приміщення повинні бути облаштованими шафами для зберігання документів, магнітних дисків, полицями, стелажами, тумбочками, тощо..
- у приміщення з ВДТ слід щоденно робити вологе прибирання.
- повинні бути оснащенні аптечкою невідкладної допомоги;
- мають бути обладнані приміщення для відпочинку під час роботи, кімната психологічного розвантаження.

Гігієнічні вимоги до параметрів виробничого середовища включають вимоги до параметрів мікроклімату, освітлення, шуму і вібраціях, рівнів електромагнітного та іонізуючого випромінювання.

Освітлення це використання енергії сонця і штучних джерел світла для забезпечення зорового сприйняття оточуючого світу.

Це один з найважливіших чинників, який значно впливає на продуктивність праці, рівень травматизму і професійних захворювань.

Потрібно щоб рівень освітлення відповідав нормам, не створював сліпучої дії від джерела світла, так і від предметів.

До функції зору, що відіграють найбільш важливу роль у трудовому процесі, належить:

- контрастна чутливість;
- гострота зору;
- швидкість розрізнення деталей;
- стійкість ясного бачення.

Контрастна чутливість - це здатність ока розпізнавати мінімальні рівні яскравості об’єкту і фону. Максимальна контрастна чутливість забезпечується яскравістю фону в межах 100-2200 нт(ніт), а за їх межами чутливість зменшується.

Розрізняють три види освітлення:

- природне(джерелом його є сонце);
- штучне(джерелом є штучне освітлення);
- суміщене або інтегроване (поєднання природного і штучного).

Є дві системи штучного освітлення:

- система загального освітлення(світло розподіляється на всю площа приміщення);
- Система комбінованого освітлення(там, де проводяться роботи високої точності з напругою зорового аналізатора)

Природне освітлення - це поєднання світла від прямих сонячних променів і дифузного світла небосхилу.

Природне освітлення є біологічно і гігієнічно найбільш цінним видом освітлення, до якого найкраще пристосовані очі людини, Воно виявляє позитивний психофізіологічний вплив на людину безпосереднім зв'язком з навколишнім середовищем через віконні отвори.

Приміщення для роботи з ВДТ повинні мати природне та штучне освітлення

Природне освітлення має здійснюватися через світлові прорізи, орієнтовані переважно на північ чи північний захід, коефіцієнт природного освітленості не нижче ніж1,5%.

За виробничої потреби дозволяється експлуатувати у приміщеннях без природного освітлення за узгодженням з органами державного нагляду за охороною праці та органами і установами санітарно-епідеміологічної служби.

Вікна приміщень з ВДТ повинні мати регулювальні пристрої для відкривання, а також жалюзі, штори, зовні козирки тощо.

Штучне освітлення приміщення з робочими місцями, обладнаними ВДТ загального та персонального користування, має бути обладнане системою загального рівного освітлення.

Загальне освітлення має бути виконане у вигляді суцільних або переривчастих ліній світильників, що розміщуються збоку працівників.

Допускається застосовувати світильники таких класів світлорозподілу:

- світильники прямого світла-П
- переважно прямого світла-Н
- переважно відбитого світла-В.

Світильники штучного освітлення повинні розміщуватися локально над робочими місцями.

Для загального освітлення необхідно застосовувати світильники із розсіювачами та дзеркальними екранними сітками або віддзеркалювачами. укомплектовані високочастотними пускорегулювальними апаратами. Використовувати світильники без розсіювачів та екранних сіток забороняється.

Як джерело світла при штучному освітленні повинні застосовуватися, люмінесцентні лампи типу ЛБ. При обладнанні відбивного освітлення у виробничих та адміністративно-громадських приміщеннях можуть застосовуватися метало-галогенні лампи потужністю до 250 Вт.

Допускається використовувати лампи розжарювання.

Яскравість світильників загального освітлення в зоні кутів випромінювання від 50* до 90* відносно вертикалів подовжній і поперечній площинах повинна складати не більше 200кд/м2,а захисний кут світильників повинен бути не більше за 40*.

Коефіцієнт запасу для освітлювальної установки загального освітлення слід приймати рівним 1.4.

Коефіцієнт пульсації повинен не перевищувати 5%і забезпечуватися застосуванням газорозрядних ламп у світильниках загального і місцевого освітлення.

За відсутності світильників з ВЧ ПРА - лампи багатоламповых світильників або розташовані поруч світильники загального освітлення необхідно підключати до різних фаз трифазної мережі.

Рівень освітленості на робочому столі в зоні розташування документів має бути в межах 300..500лк.У разі неможливості забезпечити даний рівень освітленості системою загального освітлення допускається застосування світильників місцевого освітлення, але при цьому не повинно бути відблисків на поверхні екрану та збільшення освітленості екрану більше ніж до 300 лк.

Світильники місцевого освітлення повинні мати напівпрозорий відбивач світла з захисним кутом не менше за 40*.

Необхідно передбачити обмеження прямої близькості від джерела природного та штучного освітлення, при цьому яскравість поверхонь, що світиться (вікна, джерела штучного світла) і перебувають в полі зору, повинна бути не більшою за 200кд/м2.

Необхідно обмежувати нерівномірність розподілу яскравості в полі зору осіб, що працюють з відео-терміналом, при цьому відношення значить яскравості робочих поверхонь не повинно перевищувати 3:1,а робочих поверхонь і навколишніх предметів(стіни, обладнання)-5:1.

Необхідно використовувати систему вимикачів, що дозволяю регулювати інтенсивність природного, а також дозволяє освітлювати тільки потрібні для роботи зони приміщення.

Для забезпечення нормованих значень освітлення в приміщеннях з відео-терміналами ЕОМ загального та персонального користування необхідно очищати віконне скло та світильники не рідше ніж 2 рази на рік та своєчасно проводити заміну ламп, що перегоріли.

Вимоги до організації робочих місць користувачів ПК.

При розміщенні робочих місць з персональними комп'ютерами відстань між робочими столами з відеомоніторами повинно бути не менше 2,0 м, а відстань між бічними поверхнями відеомоніторів - не менше 1,2 м.

Робочі місця з персональними комп'ютерами в приміщеннях з джерелами шкідливих виробничих факторів розміщуються в ізольованих кабінах з організованим повітрообміном. При виконанні творчої роботи, що вимагає значного розумового напруження або високої концентрації уваги, робочі місця з персональними комп'ютерами рекомендується ізолювати один від одного перегородками висотою 1,5-2,0 м.. Екран відеомонітора повинен знаходитися від очей користувача на відстані 600-700 мм, але не ближче 500 мм з урахуванням розмірів алфавітно-цифрових знаків і символів.

Конструкція робочого столу повинна забезпечувати оптимальне розміщення на робочій поверхні використовуваного обладнання з урахуванням його кількості і конструктивних особливостей, характеру виконуваної роботи. При цьому допускається використання робочих столів різних конструкцій, що відповідають сучасним вимогам ергономіки. Поверхня робочого столу повинна мати коефіцієнт відбиття 0,5-0,7.Висота робочої поверхні столу для дорослих користувачів повинна регулюватися в межах 680-800 мм; при відсутності такої можливості висота робочої поверхні столу повинна складати 725 мм. Модульними розмірами робочої поверхні столу для ПК, на підставі яких повинні розраховуватися конструктивні розміри, слід вважати: ширину - 800, 1000, 1200 і 1400 мм, глибину - 800 і 1000 мм при нерегульованої його висоті, рівній 725 мм.

Робочий стіл повинен мати простір для ніг висотою не менше 600 мм, шириною - не менше 500 мм, глибиною на рівні колін - не менше 450 мм, на рівні витягнутої ноги - не менше 650 мм.

Конструкція робочого стільця (крісла) повинна забезпечувати підтримку раціональної робочої пози під час роботи на персональному комп'ютері, дозволяти змінювати позу з метою зниження статичного напруження м'язів шийно-плечової області і спини для попередження розвитку втоми. Тип робочого стільця (крісла) слід вибирати з урахуванням зростання користувача, характеру та тривалості роботи з ПК. Робочий стілець (крісло) повинен бути підйомно-поворотним,

регульованим по висоті і кутам нахилу сидіння і спинки, а також відстані спинки від переднього краю сидіння, при цьому регулювання кожного параметра повинні бути незалежною, легко здійснюваної мати надійну фіксацію. Поверхня сидіння, спинки та інших елементів стільця (крісла) повинна бути напівм'якої, з нековзним, слабо електризується і повітропроникним покриттям, що забезпечує легке очищення від забруднень. Робоче місце користувача ПК слід обладнати підставкою для ніг, має ширину не менше 300 мм, глибину не менше 400 мм, регулювання по висоті в межах до 150 мм і за кутом нахилу опорної поверхні підставки до 20 град. Поверхня підставки повинна бути рифленою і мати по передньому краю бортик висотою 10 мм.

Клавіатуру слід розташовувати на поверхні столу на відстані 100-300 мм від краю, зверненого до користувача, або на спеціальній, регульованою по висоті робочої поверхні, відокремленої від основної стільниці.

ВИСНОВКИ

Отже дослідивши тему штучного інтелекту, та розробивши додаток який використовуватиме цю технологію – було визначено потенційні проблеми та переваги даної технології.

Особливістю цієї сфери є вміння комп’ютера імітувати роботу людського мозку та вміння технологій навчатися на основі своїх попередніх результатів. Сьогодні є багато сфер досліджень та використання даної технології.

Мобільні пристрою також мають можливість використовувати цю технологію. Це відкриває широкий спектр можливостей в сучасному світі.

Метою цією роботи було дослідити можливості технології штучного інтелекту та застовувати в реальному світі.

В результаті роботи було розроблено додаток на задану тему та досліджено різні випадки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Як працює машинне навчання та AI. [Електронний ресурс] Режим доступу http://thefuture.news/ai
2) Штучний інтелект, машинне навчання та нейронні мережі – у чому їх різниця і для чого їх використовують. [Електронний ресурс] Режим доступу https://evergreens.com.ua/ua/articles/machine-learning-overview.html
3) Історія мобільних пристрій та їх використання \. [Електронний ресурс] Режим доступу https://westele.com.ua/ua/blog/135_istoria-mobilnogo-telefona.html
4) Keychain servicer [Електронний ресурс] Режим доступу https://developer.apple.com/documentation/security/keychain_services
5) Firebase Pricing [Електронний ресурс] Режим доступу https://firebase.google.com/pricing
6) Games made by unity [Електронний ресурс] Режим доступу https://unity3d.com/games-made-with-unity
7) Android vs. iOS: Which is more secure? [Електронний ресурс] Режим доступу https://us.norton.com/internetsecurity-mobile-android-vs-ios-which-is-more-secure.html
8) Богданюк В.Є. Методичні вказівки до виконання організаційно-економічного розділу дипломних проектів / Богданюк В.Є., Березовський К.В., Пашін В.П. – К.: НТУУ "КПІ", 1999. – 66 с.
9) Методичні вказівки до виконання магістерської роботи освітнього рівня “магістр” студентами усіх форм навчання для напряму підготовки 121 – “Інженерія програмного забезпечення” / Укладачі : Петрик М.Р., Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 26 с.
10) Методичні рекомендації по виконанню розділу техніко-економічного обґрунтування дипломних робіт студентами технічних спеціальностей напряму підготовки 8.05010302 «Інженерія програмного забезпечення» освітньо-кваліфікаційного

рівня «Магістр» / Укладачі: Петрик М.Р., Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль: Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с.

ДОДАТКИ

ДОДАТОК А - ТЕХНІЧНЕ ЗАВДАННЯ

Міністерство освіти і науки України

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп’ютерно-інформаційних систем і програмної інженерії

Кафедра програмної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою

програмної інженерії

к.т.н. Петрик М.Р.

“___ “_____ 2019 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання магістерської роботи

на тему: «Розробка переглядача зображень з можливістю керування жестами на платформу iOS для мультимедійного простору»

Коцулиму Валерію Євгенійовичу

Керівник роботи:

к.т.н., доц., проректор з НМР, Дячук С.Ф.,

“___” _____ 2019 р.

_____

Виконавець:

студент групи СПм-61

Коцулим Валерій Євгенійович
“___” _____ 2019 р.

_____

м. Тернопіль – 2019

УДК 004.415.5

В. Коцулим ст. гр. СПм-61

Тернопільський національний технічний університет імені Івана Пулюя

Проблеми використання технології машинного навчання

Машинне навчання - це вміння програми самовдосконалюватися через деякий час використовуючи “досвід” здобутий на попередніх етапах. Це дозволяє програмам показувати кращі результати та вищої точності розрахунків. Одна із сфер застосування цієї технології є розпізнавання обличчя, що дозволяє керування програмними продуктами за допомогою міміки.

Сьогодні є дуже багато додатків які працюють із використанням машинного навчання, однак більшість з них використовуються в розважальних цілях, та не приносять конкретної користі. Також всі вони використовують багато процесорних та акумуляторних ресурсів.

Машинне навчання — це технологія, перші про застосування якого датуються 60 рокам XX століття. В той час було випущено першу “розумну” машину Perceptron, яка вчилася розрізняти прості математичні фігури як прямокутники, трикутники тощо... На даний момент ця технологія знаходиться на високому рівні розвитку та використовується, в основному, для таких цілей:

- Обслуговування клієнтів;
- Автопілоти;
- Виявлення шахрайства;
- Аналіз та прогноз статистичних даних;
- Безпека доступу до даних.

Багато сучасних програмних систем виходять з ладу при обробці великих збірок

даних, причиною чого є неправильні дії системи внаслідок перенавчання та перенасичення системи даними, що може похитнути довіру до таких систем.

Більшість компаній використовують штучний інтелект як “ярлик” для своїх продуктів, збільшуючи таким чином свої продажі та рейтинги, однак часто це не відповідає дійсності.

Такі програмні продукти, в більшості, використовують штучний інтелект там — де без цього можна обійтися використовуючи стандартні технології та методи.

Жоден із продуктів, які використовують машинний інтелект, не можуть робити прогнози із 100% ймовірністю та приймати тільки правильні рішення, вони можуть мати тільки наближену до цієї цифри точність прийнятих рішень та не зможуть замінити всіх людей на робочих місцях.

Список використаної літератури:

1. ЯК НАВЧАЮТЬСЯ МАШИНИ І ЧОМУ ШТУЧНИЙ ІНТЕЛЕКТ НЕ
РОЗУМНИЙ? [Електронний ресурс] – Режим доступу: https://cybercalm.org/novyny/yak-navchayutsya-mashyny-i-chomu-shtuchnyj-intelekt-ne-rozumnyj/ – Назва з екрану.

2. 8 шляхів, як машинне навчання вдосконалить робочі процеси компаній [Електронний ресурс] – Режим доступу: https://www.imena.ua/blog/ai-for-better-work/ –Назва з екрану.

3. Штучний інтелект рекламують як вирішення усіх проблем, насправді ж вінчасто помиляється й не може замінити людину [Електронний ресурс] – Режим доступу:
https://thebabel.com.ua/texts/38487-shtuchniy-intelekt-reklamuyut-yak-virishennya-usih-problem-naspravdi-vin-chasto-pomilyayetsya-i-ne-mozhe-zaminiti-lyudinu-inzhener-z-prinstona-poyasnyu-chomu – Назва з екрану.

```
import AVFoundation
import UIKit

import Firebase

/// Defines UI-related utilitiy methods for vision detection.
public class CameraMotionRecognizer {

  // MARK: - Public

    private static var motionRecognizer:
CameraMotionRecognizer?

    public static func addMotionRecognizer(motionRecognizer:
CameraMotionRecognizer) {
        self.motionRecognizer = motionRecognizer
    }
    public static func removeMotionRecognizer() {
        self.motionRecognizer = nil
    }

  public static func addCircle(
    atPoint point: CGPoint,
    to view: UIView,
    color: UIColor,
    radius: CGFloat
    ) {
    let divisor: CGFloat = 2.0
    let xCoord = point.x - radius / divisor
    let yCoord = point.y - radius / divisor
    let circleRect = CGRect(x: xCoord, y: yCoord, width:
radius, height: radius)
    let circleView = UIView(frame: circleRect)
    circleView.layer.cornerRadius = radius / divisor
    circleView.alpha = Constants.circleViewAlpha
    circleView.backgroundColor = color
    view.addSubview(circleView)a
  }

  public static func addRectangle(_ rectangle: CGRect, to
view: UIView, color: UIColor) {
    let rectangleView = UIView(frame: rectangle)
    rectangleView.layer.cornerRadius =
Constants.rectangleViewCornerRadius
    rectangleView.alpha = Constants.rectangleViewAlpha
    rectangleView.backgroundColor = color
    view.addSubview(rectangleView)
```

```
  }

  public static func addShape(withPoints points: [NSValue]?,
to view: UIView, color: UIColor) {
    guard let points = points else { return }
    let path = UIBezierPath()
    for (index, value) in points.enumerated() {
      let point = value.cgPointValue
      if index == 0 {
        path.move(to: point)
      } else {
        path.addLine(to: point)
      }
      if index == points.count - 1 {
        path.close()
      }
    }
    let shapeLayer = CAShapeLayer()
    shapeLayer.path = path.cgPath
    shapeLayer.fillColor = color.cgColor
    let rect = CGRect(x: 0, y: 0, width:
view.frame.size.width, height: view.frame.size.height)
    let shapeView = UIView(frame: rect)
    shapeView.alpha = Constants.shapeViewAlpha
    shapeView.layer.addSublayer(shapeLayer)
    view.addSubview(shapeView)
  }

  public static func imageOrientation(
    fromDevicePosition devicePosition:
AVCaptureDevice.Position = .back
    ) -> UIImage.Orientation {
    var deviceOrientation = UIDevice.current.orientation
    if deviceOrientation == .faceDown || deviceOrientation ==
.faceUp ||
      deviceOrientation == .unknown {
      deviceOrientation = currentUIOrientation()
    }
    switch deviceOrientation {
    case .portrait:
      return devicePosition == .front ? .leftMirrored : .right
    case .landscapeLeft:
      return devicePosition == .front ? .downMirrored : .up
    case .portraitUpsideDown:
      return devicePosition == .front ? .rightMirrored : .left
    case .landscapeRight:
      return devicePosition == .front ? .upMirrored : .down
    case .faceDown, .faceUp, .unknown:
      return .up
    }
  }

  public static func visionImageOrientation(
```

```
    from imageOrientation: UIImage.Orientation
    ) -> VisionDetectorImageOrientation {
    switch imageOrientation {
    case .up:
      return .topLeft
    case .down:
      return .bottomRight
    case .left:
      return .leftBottom
    case .right:
      return .rightTop
    case .upMirrored:
      return .topRight
    case .downMirrored:
      return .bottomLeft
    case .leftMirrored:
      return .leftTop
    case .rightMirrored:
      return .rightBottom
    }
  }

  // MARK: - Private

  private static func currentUIOrientation() ->
UIDeviceOrientation {
    let deviceOrientation = { () -> UIDeviceOrientation in
      switch UIApplication.shared.statusBarOrientation {
      case .landscapeLeft:
        return .landscapeRight
      case .landscapeRight:
        return .landscapeLeft
      case .portraitUpsideDown:
        return .portraitUpsideDown
      case .portrait, .unknown:
        return .portrait
      }
    }
    guard Thread.isMainThread else {
      var currentOrientation: UIDeviceOrientation = .portrait
      DispatchQueue.main.sync {
        currentOrientation = deviceOrientation()
      }
      return currentOrientation
    }
    return deviceOrientation()
  }
}

// MARK: - Constants

private enum Constants {
  static let circleViewAlpha: CGFloat = 0.7
```

```
    static let rectangleViewAlpha: CGFloat = 0.3
    static let shapeViewAlpha: CGFloat = 0.3
    static let rectangleViewCornerRadius: CGFloat = 10.0
}
```

```
import AVFoundation
import CoreVideo

import Firebase

class CameraViewController: UIViewController {
  private let detectors: [Detector] =
[.onDeviceAutoMLImageLabeler,
                                       .onDeviceFace,
                                       .onDeviceText,

.onDeviceObjectProminentNoClassifier,

.onDeviceObjectProminentWithClassifier,

.onDeviceObjectMultipleNoClassifier,

.onDeviceObjectMultipleWithClassifier]
  private var currentDetector: Detector = .onDeviceFace
  private var isUsingFrontCamera = true
  private var previewLayer: AVCaptureVideoPreviewLayer!
  private lazy var captureSession = AVCaptureSession()
  private lazy var sessionQueue = DispatchQueue(label:
Constant.sessionQueueLabel)
  private lazy var vision = Vision.vision()
  private var lastFrame: CMSampleBuffer?
  private var areAutoMLModelsRegistered = false
  private lazy var modelManager = ModelManager.modelManager()
  @IBOutlet var downloadProgressView: UIProgressView!

  private lazy var previewOverlayView: UIImageView = {

    precondition(isViewLoaded)
    let previewOverlayView = UIImageView(frame: .zero)
    previewOverlayView.contentMode =
UIView.ContentMode.scaleAspectFill

previewOverlayView.translatesAutoresizingMaskIntoConstraints =
false
    return previewOverlayView
  }()

  private lazy var annotationOverlayView: UIView = {
    precondition(isViewLoaded)
    let annotationOverlayView = UIView(frame: .zero)

annotationOverlayView.translatesAutoresizingMaskIntoConstraint
s = false
    return annotationOverlayView
  }()

  // MARK: - IBOutlets
```

```
  @IBOutlet private weak var cameraView: UIView!

  // MARK: - UIViewController

  override func viewDidLoad() {
    super.viewDidLoad()

    previewLayer = AVCaptureVideoPreviewLayer(session:
captureSession)
    setUpPreviewOverlayView()
    setUpAnnotationOverlayView()
    setUpCaptureSessionOutput()
    setUpCaptureSessionInput()
  }

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

    startSession()
  }

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

    stopSession()
  }

  override func viewDidLayoutSubviews() {
    super.viewDidLayoutSubviews()

    previewLayer.frame = cameraView.frame
  }

  // MARK: - IBActions

  @IBAction func selectDetector(_ sender: Any) {
    presentDetectorsAlertController()
  }

  @IBAction func switchCamera(_ sender: Any) {
    isUsingFrontCamera = !isUsingFrontCamera
    removeDetectionAnnotations()
    setUpCaptureSessionInput()
  }

  // MARK: - On-Device AutoML Detections

  private func detectImageLabelsAutoMLOndevice(
    in visionImage: VisionImage,
    width: CGFloat,
    height: CGFloat) {
    registerAutoMLModelsIfNeeded()
```

```
    let options = VisionOnDeviceAutoMLImageLabelerOptions(
      remoteModelName: Constant.remoteAutoMLModelName,
      localModelName: Constant.localAutoMLModelName
    )
    options.confidenceThreshold =
Constant.labelConfidenceThreshold
    let autoMLOnDeviceLabeler =
vision.onDeviceAutoMLImageLabeler(options: options)
    print("labeler: \(autoMLOnDeviceLabeler)\n")

    let group = DispatchGroup()
    group.enter()

    autoMLOnDeviceLabeler.process(visionImage) {
detectedLabels, error in
      defer { group.leave() }

      self.updatePreviewOverlayView()
      self.removeDetectionAnnotations()

      if let error = error {
        print("Failed to detect labels with error:
\(error.localizedDescription).")
        return
      }

      guard let labels = detectedLabels, !labels.isEmpty else
{
        return
      }

      let annotationFrame = self.annotationOverlayView.frame
      let resultsRect = CGRect(
        x: annotationFrame.origin.x + Constant.padding,
        y: annotationFrame.size.height - Constant.padding -
Constant.resultsLabelHeight,
        width: annotationFrame.width - 2 * Constant.padding,
        height: Constant.resultsLabelHeight
      )
      let resultsLabel = UILabel(frame: resultsRect)
      resultsLabel.textColor = .yellow
      resultsLabel.text = labels.map { label -> String in
        return "Label: \(label.text), Confidence:
\(label.confidence ?? 0)"
        }.joined(separator: "\n")
      resultsLabel.adjustsFontSizeToFitWidth = true
      resultsLabel.numberOfLines = Constant.resultsLabelLines
      self.annotationOverlayView.addSubview(resultsLabel)
    }

    group.wait()
  }
```

```
  private func registerAutoMLModelsIfNeeded() {
    if areAutoMLModelsRegistered {
      return
    }

    let initialConditions = ModelDownloadConditions()
    let updateConditions = ModelDownloadConditions(
      allowsCellularAccess: false,
      allowsBackgroundDownloading: true
    )
    let remoteModel = RemoteModel(
      name: Constant.remoteAutoMLModelName,
      allowsModelUpdates: true,
      initialConditions: initialConditions,
      updateConditions: updateConditions
    )
    modelManager.register(remoteModel)
    NotificationCenter.default.addObserver(
      self,
      selector: #selector(remoteModelDownloadDidSucceed(_:)),
      name: .firebaseMLModelDownloadDidSucceed,
      object: nil
    )
    NotificationCenter.default.addObserver(
      self,
      selector: #selector(remoteModelDownloadDidFail(_:)),
      name: .firebaseMLModelDownloadDidFail,
      object: nil
    )
    DispatchQueue.main.async {
      self.downloadProgressView.isHidden = false
      self.downloadProgressView.observedProgress =
self.modelManager.download(remoteModel)
    }

    guard let localModelFilePath = Bundle.main.path(
      forResource: Constant.localModelManifestFileName,
      ofType: Constant.autoMLManifestFileType
      ) else {
        print("Failed to find AutoML local model manifest
file.")
        return
    }
    let localModel = LocalModel(name:
Constant.localAutoMLModelName, path: localModelFilePath)
    modelManager.register(localModel)
    areAutoMLModelsRegistered = true
  }

  // MARK: - Notifications

  @objc
```

```
  private func remoteModelDownloadDidSucceed(_ notification:
Notification) {
    let notificationHandler = {
      self.downloadProgressView.isHidden = true
      guard let userInfo = notification.userInfo,
        let remoteModel =

userInfo[ModelDownloadUserInfoKey.remoteModel.rawValue] as?
RemoteModel
        else {
          print("firebaseMLModelDownloadDidSucceed
notification posted without a RemoteModel instance.")
          return
      }
      print("Successfully downloaded the remote model with
name: \(remoteModel.name). The model is ready for detection.")
    }
    if Thread.isMainThread { notificationHandler(); return }
    DispatchQueue.main.async { notificationHandler() }
  }

  @objc
  private func remoteModelDownloadDidFail(_ notification:
Notification) {
    let notificationHandler = {
      self.downloadProgressView.isHidden = true
      guard let userInfo = notification.userInfo,
        let remoteModel =

userInfo[ModelDownloadUserInfoKey.remoteModel.rawValue] as?
RemoteModel,
        let error =
userInfo[ModelDownloadUserInfoKey.error.rawValue] as? NSError
        else {
          print("firebaseMLModelDownloadDidFail notification
posted without a RemoteModel instance or error.")
          return
      }
      print("Failed to download the remote model with name:
\(remoteModel.name), error: \(error).")
    }
    if Thread.isMainThread { notificationHandler(); return }
    DispatchQueue.main.async { notificationHandler() }
  }

  // MARK: Other On-Device Detections

  private func detectFacesOnDevice(in image: VisionImage,
width: CGFloat, height: CGFloat) {
    let options = VisionFaceDetectorOptions()

    // When performing latency tests to determine ideal
detection settings,
```

```
    // run the app in 'release' mode to get accurate
performance metrics
    options.landmarkMode = .none
    options.contourMode = .all
    options.classificationMode = .none

    options.performanceMode = .fast
    let faceDetector = vision.faceDetector(options: options)

    var detectedFaces: [VisionFace]? = nil
    do {
      detectedFaces = try faceDetector.results(in: image)
    } catch let error {
      print("Failed to detect faces with error:
\(error.localizedDescription).")
    }
    guard let faces = detectedFaces, !faces.isEmpty else {
      print("On-Device face detector returned no results.")
      DispatchQueue.main.sync {
        self.updatePreviewOverlayView()
        self.removeDetectionAnnotations()
      }
      return
    }

    DispatchQueue.main.sync {
      self.updatePreviewOverlayView()
      self.removeDetectionAnnotations()
      for face in faces {
        let normalizedRect = CGRect(
          x: face.frame.origin.x / width,
          y: face.frame.origin.y / height,
          width: face.frame.size.width / width,
          height: face.frame.size.height / height
        )
        let standardizedRect =

self.previewLayer.layerRectConverted(fromMetadataOutputRect:
normalizedRect).standardized
        UIUtilities.addRectangle(
          standardizedRect,
          to: self.annotationOverlayView,
          color: UIColor.green
        )
        self.addContours(for: face, width: width, height:
height)
      }
    }
  }

  private func recognizeTextOnDevice(in image: VisionImage,
width: CGFloat, height: CGFloat) {
    let textRecognizer = vision.onDeviceTextRecognizer()
```

```
    textRecognizer.process(image) { text, error in
      self.removeDetectionAnnotations()
      self.updatePreviewOverlayView()
      guard error == nil, let text = text else {
        print("On-Device text recognizer error: " +
          "\(error?.localizedDescription ??
Constant.noResultsMessage)")
        return
      }
      // Blocks.
      for block in text.blocks {
        let points = self.convertedPoints(from:
block.cornerPoints, width: width, height: height)
        UIUtilities.addShape(
          withPoints: points,
          to: self.annotationOverlayView,
          color: UIColor.purple
        )

        // Lines.
        for line in block.lines {
          let points = self.convertedPoints(from:
line.cornerPoints, width: width, height: height)
          UIUtilities.addShape(
            withPoints: points,
            to: self.annotationOverlayView,
            color: UIColor.orange
          )

          // Elements.
          for element in line.elements {
            let normalizedRect = CGRect(
              x: element.frame.origin.x / width,
              y: element.frame.origin.y / height,
              width: element.frame.size.width / width,
              height: element.frame.size.height / height
            )
            let convertedRect =
self.previewLayer.layerRectConverted(
              fromMetadataOutputRect: normalizedRect
            )
            UIUtilities.addRectangle(
              convertedRect,
              to: self.annotationOverlayView,
              color: UIColor.green
            )
            let label = UILabel(frame: convertedRect)
            label.text = element.text
            label.adjustsFontSizeToFitWidth = true
            self.annotationOverlayView.addSubview(label)
          }
        }
      }
```

```
    }
  }

  // MARK: Object Detection

  private func detectObjectsOnDevice(in image: VisionImage,
                                     width: CGFloat,
                                     height: CGFloat,
                                     options:
VisionObjectDetectorOptions) {
    let detector = vision.objectDetector(options: options)

    var detectedObjects: [VisionObject]? = nil
    do {
      detectedObjects = try detector.results(in: image)
    } catch let error {
      print("Failed to detect object with error:
\(error.localizedDescription).")
      return
    }
    guard let objects = detectedObjects, !objects.isEmpty else
{
      print("On-Device object detector returned no results.")
      DispatchQueue.main.sync {
        self.updatePreviewOverlayView()
        self.removeDetectionAnnotations()
      }
      return
    }

    DispatchQueue.main.sync {
      self.updatePreviewOverlayView()
      self.removeDetectionAnnotations()
      for object in objects {
        let normalizedRect = CGRect(
          x: object.frame.origin.x / width,
          y: object.frame.origin.y / height,
          width: object.frame.size.width / width,
          height: object.frame.size.height / height
        )
        let standardizedRect =

self.previewLayer.layerRectConverted(fromMetadataOutputRect:
normalizedRect).standardized
        UIUtilities.addRectangle(
          standardizedRect,
          to: self.annotationOverlayView,
          color: UIColor.green
        )
        let label = UILabel(frame: standardizedRect)
        label.numberOfLines = 2
        var description = ""
        if let trackingID = object.trackingID {
```

```
          description = "ID:" + trackingID.stringValue + "\n"
        }
        description = description + "
Class:\(object.classificationCategory.rawValue)"
        label.text = description

        label.adjustsFontSizeToFitWidth = true
        self.annotationOverlayView.addSubview(label)
      }
    }
  }

  // MARK: - Private

  private func setUpCaptureSessionOutput() {
    sessionQueue.async {
      self.captureSession.beginConfiguration()
      // When performing latency tests to determine ideal
capture settings,
      // run the app in 'release' mode to get accurate
performance metrics
      self.captureSession.sessionPreset =
AVCaptureSession.Preset.medium

      let output = AVCaptureVideoDataOutput()
      output.videoSettings =
        [(kCVPixelBufferPixelFormatTypeKey as String):
kCVPixelFormatType_32BGRA]
      let outputQueue = DispatchQueue(label:
Constant.videoDataOutputQueueLabel)
      output.setSampleBufferDelegate(self, queue: outputQueue)
      guard self.captureSession.canAddOutput(output) else {
        print("Failed to add capture session output.")
        return
      }
      self.captureSession.addOutput(output)
      self.captureSession.commitConfiguration()
    }
  }

  private func setUpCaptureSessionInput() {
    sessionQueue.async {
      let cameraPosition: AVCaptureDevice.Position =
self.isUsingFrontCamera ? .front : .back
      guard let device = self.captureDevice(forPosition:
cameraPosition) else {
        print("Failed to get capture device for camera
position: \(cameraPosition)")
        return
      }
      do {
        self.captureSession.beginConfiguration()
        let currentInputs = self.captureSession.inputs
```

```
        for input in currentInputs {
          self.captureSession.removeInput(input)
        }

        let input = try AVCaptureDeviceInput(device: device)
        guard self.captureSession.canAddInput(input) else {
          print("Failed to add capture session input.")
          return
        }
        self.captureSession.addInput(input)
        self.captureSession.commitConfiguration()
      } catch {
        print("Failed to create capture device input:
\(error.localizedDescription)")
      }
    }
  }

  private func startSession() {
    sessionQueue.async {
      self.captureSession.startRunning()
    }
  }

  private func stopSession() {
    sessionQueue.async {
      self.captureSession.stopRunning()
    }
  }

  private func setUpPreviewOverlayView() {
    cameraView.addSubview(previewOverlayView)
    NSLayoutConstraint.activate([
      previewOverlayView.centerXAnchor.constraint(equalTo:
cameraView.centerXAnchor),
      previewOverlayView.centerYAnchor.constraint(equalTo:
cameraView.centerYAnchor),
      previewOverlayView.leadingAnchor.constraint(equalTo:
cameraView.leadingAnchor),
      previewOverlayView.trailingAnchor.constraint(equalTo:
cameraView.trailingAnchor),

      ])
  }

  private func setUpAnnotationOverlayView() {
    cameraView.addSubview(annotationOverlayView)
    NSLayoutConstraint.activate([
      annotationOverlayView.topAnchor.constraint(equalTo:
cameraView.topAnchor),
      annotationOverlayView.leadingAnchor.constraint(equalTo:
cameraView.leadingAnchor),
```

```
      annotationOverlayView.trailingAnchor.constraint(equalTo:
cameraView.trailingAnchor),
      annotationOverlayView.bottomAnchor.constraint(equalTo:
cameraView.bottomAnchor),
      ])
  }

  private func captureDevice(forPosition position:
AVCaptureDevice.Position) -> AVCaptureDevice? {
    if #available(iOS 10.0, *) {
      let discoverySession = AVCaptureDevice.DiscoverySession(
        deviceTypes: [.builtInWideAngleCamera],
        mediaType: .video,
        position: .unspecified
      )
      return discoverySession.devices.first { $0.position ==
position }
    }
    return nil
  }

  private func presentDetectorsAlertController() {
    let alertController = UIAlertController(
      title: Constant.alertControllerTitle,
      message: Constant.alertControllerMessage,
      preferredStyle: .alert
    )
    detectors.forEach { detectorType in
      let action = UIAlertAction(title: detectorType.rawValue,
style: .default) {
        [unowned self] (action) in
        guard let value = action.title else { return }
        guard let detector = Detector(rawValue: value) else {
return }
        self.currentDetector = detector
        self.removeDetectionAnnotations()
      }
      if detectorType.rawValue == currentDetector.rawValue {
action.isEnabled = false }
      alertController.addAction(action)
    }
    alertController.addAction(UIAlertAction(title:
Constant.cancelActionTitleText, style: .cancel))
    present(alertController, animated: true)
  }

  private func removeDetectionAnnotations() {
    for annotationView in annotationOverlayView.subviews {
      annotationView.removeFromSuperview()
    }
  }

  private func updatePreviewOverlayView() {
```

```
    guard let lastFrame = lastFrame,
          let imageBuffer =
CMSampleBufferGetImageBuffer(lastFrame)
    else {
      return
    }
    let ciImage = CIImage(cvPixelBuffer: imageBuffer)
    let context = CIContext(options: nil)
    guard let cgImage = context.createCGImage(ciImage, from:
ciImage.extent) else {
      return
    }
    let rotatedImage =
      UIImage(cgImage: cgImage, scale: Constant.originalScale,
orientation: .right)
    if isUsingFrontCamera {
      guard let rotatedCGImage = rotatedImage.cgImage else {
        return
      }
      let mirroredImage = UIImage(
        cgImage: rotatedCGImage, scale:
Constant.originalScale, orientation: .leftMirrored)
      previewOverlayView.image = mirroredImage
    } else {
      previewOverlayView.image = rotatedImage
    }
  }

  private func convertedPoints(
    from points: [NSValue]?,
    width: CGFloat,
    height: CGFloat
  ) -> [NSValue]? {
    return points?.map {
      let cgPointValue = $0.cgPointValue
      let normalizedPoint = CGPoint(x: cgPointValue.x / width,
y: cgPointValue.y / height)
      let cgPoint =
previewLayer.layerPointConverted(fromCaptureDevicePoint:
normalizedPoint)
      let value = NSValue(cgPoint: cgPoint)
      return value
    }
  }

  private func normalizedPoint(
    fromVisionPoint point: VisionPoint,
    width: CGFloat,
    height: CGFloat
  ) -> CGPoint {
    let cgPoint = CGPoint(x: CGFloat(point.x.floatValue), y:
CGFloat(point.y.floatValue))
```

```
    var normalizedPoint = CGPoint(x: cgPoint.x / width, y:
cgPoint.y / height)
    normalizedPoint =
previewLayer.layerPointConverted(fromCaptureDevicePoint:
normalizedPoint)
    return normalizedPoint
  }

  private func addContours(for face: VisionFace, width:
CGFloat, height: CGFloat) {
    // Face
    if let faceContour = face.contour(ofType: .face) {
      for point in faceContour.points {
        let cgPoint = normalizedPoint(fromVisionPoint: point,
width: width, height: height)
        UIUtilities.addCircle(
          atPoint: cgPoint,
          to: annotationOverlayView,
          color: UIColor.blue,
          radius: Constant.smallDotRadius
        )
      }
    }

    // Eyebrows
    if let topLeftEyebrowContour = face.contour(ofType:
.leftEyebrowTop) {
      for point in topLeftEyebrowContour.points {
        let cgPoint = normalizedPoint(fromVisionPoint: point,
width: width, height: height)
        UIUtilities.addCircle(
          atPoint: cgPoint,
          to: annotationOverlayView,
          color: UIColor.orange,
          radius: Constant.smallDotRadius
        )
      }
    }
    if let bottomLeftEyebrowContour = face.contour(ofType:
.leftEyebrowBottom) {
      for point in bottomLeftEyebrowContour.points {
        let cgPoint = normalizedPoint(fromVisionPoint: point,
width: width, height: height)
        UIUtilities.addCircle(
          atPoint: cgPoint,
          to: annotationOverlayView,
          color: UIColor.orange,
          radius: Constant.smallDotRadius
        )
      }
    }
    if let topRightEyebrowContour = face.contour(ofType:
.rightEyebrowTop) {
```

```
      for point in topRightEyebrowContour.points {
        let cgPoint = normalizedPoint(fromVisionPoint: point,
width: width, height: height)
        UIUtilities.addCircle(
          atPoint: cgPoint,
          to: annotationOverlayView,
          color: UIColor.orange,
          radius: Constant.smallDotRadius
        )
      }
    }
    if let bottomRightEyebrowContour = face.contour(ofType:
.rightEyebrowBottom) {
      for point in bottomRightEyebrowContour.points {
        let cgPoint = normalizedPoint(fromVisionPoint: point,
width: width, height: height)
        UIUtilities.addCircle(
          atPoint: cgPoint,
          to: annotationOverlayView,
          color: UIColor.orange,
          radius: Constant.smallDotRadius
        )
      }
    }

    // Eyes
    if let leftEyeContour = face.contour(ofType: .leftEye) {
      for point in leftEyeContour.points {
        let cgPoint = normalizedPoint(fromVisionPoint: point,
width: width, height: height)
        UIUtilities.addCircle(
          atPoint: cgPoint,
          to: annotationOverlayView,
          color: UIColor.cyan,
          radius: Constant.smallDotRadius
        )
      }
    }
    if let rightEyeContour = face.contour(ofType: .rightEye) {
      for point in rightEyeContour.points {
        let cgPoint = normalizedPoint(fromVisionPoint: point,
width: width, height: height)
        UIUtilities.addCircle(
          atPoint: cgPoint,
          to: annotationOverlayView,
          color: UIColor.cyan,
          radius: Constant.smallDotRadius
        )
      }
    }

    // Lips
```

```
    if let topUpperLipContour = face.contour(ofType:
.upperLipTop) {
      for point in topUpperLipContour.points {
        let cgPoint = normalizedPoint(fromVisionPoint: point,
width: width, height: height)
        UIUtilities.addCircle(
          atPoint: cgPoint,
          to: annotationOverlayView,
          color: UIColor.red,
          radius: Constant.smallDotRadius
        )
      }
    }
    if let bottomUpperLipContour = face.contour(ofType:
.upperLipBottom) {
      for point in bottomUpperLipContour.points {
        let cgPoint = normalizedPoint(fromVisionPoint: point,
width: width, height: height)
        UIUtilities.addCircle(
          atPoint: cgPoint,
          to: annotationOverlayView,
          color: UIColor.red,
          radius: Constant.smallDotRadius
        )
      }
    }
    if let topLowerLipContour = face.contour(ofType:
.lowerLipTop) {
      for point in topLowerLipContour.points {
        let cgPoint = normalizedPoint(fromVisionPoint: point,
width: width, height: height)
        UIUtilities.addCircle(
          atPoint: cgPoint,
          to: annotationOverlayView,
          color: UIColor.red,
          radius: Constant.smallDotRadius
        )
      }
    }
    if let bottomLowerLipContour = face.contour(ofType:
.lowerLipBottom) {
      for point in bottomLowerLipContour.points {
        let cgPoint = normalizedPoint(fromVisionPoint: point,
width: width, height: height)
        UIUtilities.addCircle(
          atPoint: cgPoint,
          to: annotationOverlayView,
          color: UIColor.red,
          radius: Constant.smallDotRadius
        )
      }
    }
```

```
    // Nose
    if let noseBridgeContour = face.contour(ofType:
.noseBridge) {
      for point in noseBridgeContour.points {
        let cgPoint = normalizedPoint(fromVisionPoint: point,
width: width, height: height)
        UIUtilities.addCircle(
          atPoint: cgPoint,
          to: annotationOverlayView,
          color: UIColor.yellow,
          radius: Constant.smallDotRadius
        )
      }
    }
    if let noseBottomContour = face.contour(ofType:
.noseBottom) {
      for point in noseBottomContour.points {
        let cgPoint = normalizedPoint(fromVisionPoint: point,
width: width, height: height)
        UIUtilities.addCircle(
          atPoint: cgPoint,
          to: annotationOverlayView,
          color: UIColor.yellow,
          radius: Constant.smallDotRadius
        )
      }
    }
  }
}

// MARK: AVCaptureVideoDataOutputSampleBufferDelegate

extension CameraViewController:
AVCaptureVideoDataOutputSampleBufferDelegate {

  func captureOutput(
    _ output: AVCaptureOutput,
    didOutput sampleBuffer: CMSampleBuffer,
    from connection: AVCaptureConnection
    ) {
    guard let imageBuffer =
CMSampleBufferGetImageBuffer(sampleBuffer) else {
      print("Failed to get image buffer from sample buffer.")
      return
    }
    lastFrame = sampleBuffer
    let visionImage = VisionImage(buffer: sampleBuffer)
    let metadata = VisionImageMetadata()
    let orientation = UIUtilities.imageOrientation(
      fromDevicePosition: isUsingFrontCamera ? .front : .back
    )
```

```
    let visionOrientation =
UIUtilities.visionImageOrientation(from: orientation)
    metadata.orientation = visionOrientation
    visionImage.metadata = metadata
    let imageWidth =
CGFloat(CVPixelBufferGetWidth(imageBuffer))
    let imageHeight =
CGFloat(CVPixelBufferGetHeight(imageBuffer))
    var shouldEnableClassification = false
    var shouldEnableMultipleObjects = false
    switch currentDetector {
    case .onDeviceObjectProminentWithClassifier,
.onDeviceObjectMultipleWithClassifier:
      shouldEnableClassification = true
    default:
      break
    }
    switch currentDetector {
    case .onDeviceObjectMultipleNoClassifier,
.onDeviceObjectMultipleWithClassifier:
      shouldEnableMultipleObjects = true
    default:
      break
    }

    switch currentDetector {
    case .onDeviceAutoMLImageLabeler:
      detectImageLabelsAutoMLOndevice(in: visionImage, width:
imageWidth, height: imageHeight)
    case .onDeviceFace:
      detectFacesOnDevice(in: visionImage, width: imageWidth,
height: imageHeight)
    case .onDeviceText:
      recognizeTextOnDevice(in: visionImage, width:
imageWidth, height: imageHeight)
    case .onDeviceObjectProminentNoClassifier,
.onDeviceObjectProminentWithClassifier,
         .onDeviceObjectMultipleNoClassifier,
.onDeviceObjectMultipleWithClassifier:
      let options = VisionObjectDetectorOptions()
      options.shouldEnableClassification =
shouldEnableClassification
      options.shouldEnableMultipleObjects =
shouldEnableMultipleObjects
      options.detectorMode = .stream
      detectObjectsOnDevice(in: visionImage,
                            width: imageWidth,
                            height: imageHeight,
                            options: options)
    }
  }
}
```

```
// MARK: - Constants

public enum Detector: String {
  case onDeviceAutoMLImageLabeler = "On-Device AutoML Image
Labeler"
  case onDeviceFace = "On-Device Face Detection"
  case onDeviceText = "On-Device Text Recognition"
  case onDeviceObjectProminentNoClassifier = "ODT for
prominent object, only tracking"
  case onDeviceObjectProminentWithClassifier = "ODT for
prominent object with classification"
  case onDeviceObjectMultipleNoClassifier = "ODT for multiple
objects, only tracking"
  case onDeviceObjectMultipleWithClassifier = "ODT for
multiple objects with classification"
}

private enum Constant {
  static let alertControllerTitle = "Vision Detectors"
  static let alertControllerMessage = "Select a detector"
  static let cancelActionTitleText = "Cancel"
  static let videoDataOutputQueueLabel =
"com.google.firebaseml.visiondetector.VideoDataOutputQueue"
  static let sessionQueueLabel =
"com.google.firebaseml.visiondetector.SessionQueue"
  static let noResultsMessage = "No Results"
  static let localAutoMLModelName = "local_automl_model"
  static let remoteAutoMLModelName = "remote_automl_model"
  static let localModelManifestFileName =
"automl_labeler_manifest"
  static let autoMLManifestFileType = "json"
  static let labelConfidenceThreshold: Float = 0.75
  static let smallDotRadius: CGFloat = 4.0
  static let originalScale: CGFloat = 1.0
  static let padding: CGFloat = 10.0
  static let resultsLabelHeight: CGFloat = 200.0
  static let resultsLabelLines = 5
}
```

ДОДАТОК Г - ДИСК