МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ
ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«___»_____2022 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТР
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
«ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ»

Тема: «Система розпізнавання обличчя на базі технології Face Recognition з використання Vision API»

Виконавець: Максим ВІЦЕНКО

Керівник: к.т.н., доцент Наталія КІРХАР

Нормоконтролер: к.т.н., доцент Олена ТОЛСТІКОВА

КИЇВ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *кібербезпеки, комп'ютерної та програмної інженерії*
Кафедра *Комп'ютерних інформаційних технологій*
Спеціальність *122 «Комп'ютерні науки»*
Освітньо-професійна програма *«Інформаційні технології проектування»*

ЗАТВЕРДЖУЮ:
завідувач кафедри КІТ
_____Аліна САВЧЕНКО
(підпис)
«____»_____2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
Віценко Максима Миколайовича
(ПІБ випускника)

1. Тема роботи: «Система розпізнавання обличчя на базі технології Face Recognition з використання Vision API» затверджена наказом ректора № 1774/ст від 28.09.2022р.
2. Термін виконання роботи: з 26 вересня 2022 року по 27 листопада 2022 року.
3. Вихідні дані до роботи: Застосунок на мові програмування SWIFT для демонстрації технології розпізнавання обличчя.
4. Зміст пояснювальної записки: 1. Аналіз та поняття технології. 2. Проектування застосунку. 3. Розробка та тестування застосунку.
5. Перелік обов'язкового ілюстративного матеріалу: 6. Поняття системи розпізнавання обличчя. 7. Методи розпізнавання обличчя 13. Дизайн та функціонал застосунку. 14. Підготовка даних. 15. Реалізація функції «Маска обличчя». 16. Реалізація функції «Знаходження облич». 17. Реалізація функції «Розпізнавання обличчя». 18. Блок схема принципу роботи застосунку.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1.	Написання 1 розділу. Аналіз джерел та поняття технології	26.09.2022-16.10.2022	
2.	Написання 2 розділу. Проектування застосунку	17.10.2022-30.10.2022	
3.	Написання 3 розділу. Розробка та тестування застосунку	31.10.2022-14.11.2022	
4.	Загальне редагування та друк пояснювальної записки	15.11.2022-20.11.2022	
5.	Проходження нормоконтролю, перепліт пояснювальної записки.	16.11.2022-20.10.2022	
6.	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації	20.11.2022-22.11.2022	

7. Дата видачі завдання 26.09.2022р.

Керівник кваліфікаційної роботи _____ Наталія КІРХАР
(підпис керівника)

Завдання прийняв до виконання _____ Максим ВІЦЕНКО
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему: «Система розпізнавання обличчя на базі технології Face Recognition з використання Vision API» містить: 95 сторінок, 61 рисунків, 22 інформаційних джерела, 1 додаток

Об'єктом досліджень – є процес розпізнавання обличчя людини.

Предметом досліджень – є методи та засоби опрацювання зображень за допомогою технологиї Face Recognition з використанням Vision.api.

Мета кваліфікаційної роботи – отримати готову програму – застосунок для демонстрації технології розпізнавання обличчя.

Методи дослідження – мова програмування SWIFT, інтегроване середовище розробки Xcode**.**

Результати кваліфікаційної роботи рекомендується використовувати для демонстраціх роботи технології розпізнавання обличчя, та в подальному інтеграції у компанії та підприємства.

ЗАСТОСУНОК, РОЗПІЗНАВАННЯ ОБЛИЧЧЯ, ЗОБРАЖЕННЯ, БЕЗПЕКА , SWIFT, МОДЕЛЬ ,VISION.API, IOS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

SDK *(Software development kit)*	–	Набір для розробки програмного забезпечення
IDE *(Integrated development. environment)*	–	Інтегроване середовище розробки
API (Application programming interface)	–	Програмний інтерфейс
URL *(Uniform Resource Locator*	–	Визначник місцезнаходження сайту в мережі Інтернет

ВСТУП

Біометрія як термінологія відноситься до поєднання грецьких слів (біо) і (метрика), що означає «вимірювання життя». Це стає дуже важливим у світі комп'ютерної безпеки. Метою біометрії є або автоматична ідентифікація, або перевірка особистості людей. Це означає, що вхідні дані (зображення, мова або відео) передаються системі, і вона порівнює ці вхідні дані з базою даних.

Біометрія може бути фізіологічною або поведінковою. Фізіологічна біометрія визначає фізичні особливості людини та використовує такі ресурси, як (сканування сітківки ока, райдужної оболонки ока, відбитки пальців і розпізнавання обличчя). Поведінкова біометрія базується на поведінці користувача та включає в себе аналіз інформації, такої як форма та потік власного почерку, час натискання клавіш.

Розпізнавання обличчя використовується для ідентифікації особи за її/її вхідним зображенням. Розпізнавання обличчя адаптується для застосування на мобільних пристроях завдяки їх гнучкості. Основні проблеми розпізнавання обличчя на мобільних пристроях пов’язані з обмеженнями обчислювальної потужності, обмеженою пам’яттю мобільних пристроїв, обмеженою пропускною здатністю мережі, проблемами конфіденційності та безпеки.

Розпізнавання обличчя на мобільних пристроях можна використовувати для автентифікації користувача, розрізнення людини від іншої особи для ідентифікації сайтів соціальних мереж. Він широко використовується в безпеці і маркетингу.

Існує багато традиційних додатків безпеки, таких як додатки для імені користувача (на основі ідентичності) і додатки для пароля (облікові дані). Мобільні пристрої з традиційними програмами легко викрасти, оскільки пароль можна зламати або виявити. Однак програми розпізнавання обличчя є більш ефективними, ніж традиційні програми, оскільки вони безпечніші та гнучкіші, і користувачеві не потрібно запам’ятовувати паролі.

Програми для розпізнавання обличчя є найбільш підходящою біометрією, оскільки ці пристрої мають камери.

У цій роботі першим кроком є розпізнавання обличчя за допомогою Vision.api. Процес визначення обличчя виконується для визначення того, які пікселі на зображенні відображають частину обличчя, а які ні. Він заснований на досягненні високої швидкості виявлення обличчя за найменший проміжок часу.

Другим кроком є процес виділення ознак для визначення основних орієнтирів кожної людини. Такими орієнтирами є очі, рот або ніс.

Також розроблено можливість пошуку облич на фото, а не тільки через камеру.

Метою роботи є дослідження напрямків і технологій розпізнавання обличчя для цього буде розробленна система розпізнавання обличчя на базі технології Face Recognition з використання Vision.api для платформи IOS.

Для досягнення поставленої мети необхідне рішення наступних завдань:

1. Огляд та аналіз існуючих методів розпізнавання обличчя;
2. Проведення огляду методологій розробки програмних застосунків, систем розпізнавання обличчя, архітектурних рішень, SDK та технологій;
3. Розробка мобільного застосунку для демонстрації роботи технології розпізнавання обличчя;
4. Описання принципів роботи мобільного застосунку, та демонстрація його роботи.

Об'єктом досліджень є процес розпізнавання обличчя людини.

Предметом досліджень є методи та засоби опрацювання зображень за допомогою технології Face Recognition з використанням Vision.api.

Актуальність теми кваліфікаційної роботи «Система розпізнавання обличчя на базі технології Face Recognition з використання Vision API» ґрунтується на тому, що за наш теперішній непростий час, на підприємствах, організаціях та компаніях часто страждає безпека даних, постійно зливають

конфіденційну інформацію, напрацювання чи технології які б не слід було розповсюджувати по інтернету. Ризик втрати стабільної роботи підприємства, моральної втрати або взагалі використання напрацювань конкурентом що призведе до матеріальних втрат. За для цього існує один із найбезпечніших методів безпеки даних, це доступ до них за біометричними даними.

Для досягнення поставленої мети й виконання завдань використано метод системно-структурного аналізу наукової літератури, який дав змогу показати особливості систем розпізнавання обличчя та розробки мобільних застосунків мовою програмування Swift. Для формулювання і систематизації висновків використано методи аналізу, формалізації, абстрагування та узагальнення.

Наукова новизна роботи полягає у створенні мобільного та портативного доступу до системи розпізнавання обличчя, яку в майбутньому можливо інтегрувати у будь які пристрої з камерою від смартфона до камери на дверях в кімнату.

Практична цінність роботи полягає у тому, що програмне забезпечення має перспективу користуватись великим попитом у зв’язку з широкою доступністю, зручністю та зрозумілістю для користувача. Цей інструмент, допоможе підприємтсвам та компаніям вдосконалити свою кібербезпеку та унеможливити витік конфіденційної інформації з підприємства у мережу.

РОЗДІЛ 1
АНАЛІЗ ТА ПОНЯТТЯ ТЕХНОЛОГІЇ

1.1. Поняття системи розпізнавання обличчя

Розпізнавання обличчя — це спосіб ідентифікації або підтвердження особи людини за її обличчям. Системи розпізнавання облич можна використовувати для ідентифікації людей на фотографіях, відео або в реальному часі.[3]

Розпізнавання обличчя є категорією біометричного захисту. Інші форми біометричного програмного забезпечення включають розпізнавання голосу, розпізнавання відбитків пальців і розпізнавання сітківки або райдужної оболонки ока. Технологія в основному використовується для забезпечення безпеки та правоохоронних органів, хоча зростає інтерес до інших сфер використання.

Як правило, розпізнавання обличчя не покладається на величезну базу даних фотографій для визначення особистості — воно просто визначає та розпізнає одну особу як єдиного власника пристрою, обмежуючи доступ для інших.

Крім розблокування телефонів, розпізнавання облич працює шляхом зіставлення облич людей, які проходять повз спеціальні камери, із зображеннями людей зі списку спостереження. [4]

Списки спостереження можуть містити фотографії будь-кого, включно з людьми, яких не підозрюють у будь-яких правопорушеннях, і зображення можуть надходити звідки завгодно — навіть із наших акаунтів у соціальних мережах. Системи обробки обличчя можуть відрізнятися, але загалом вони працюють таким чином:

Крок 1: Розпізнавання обличчя

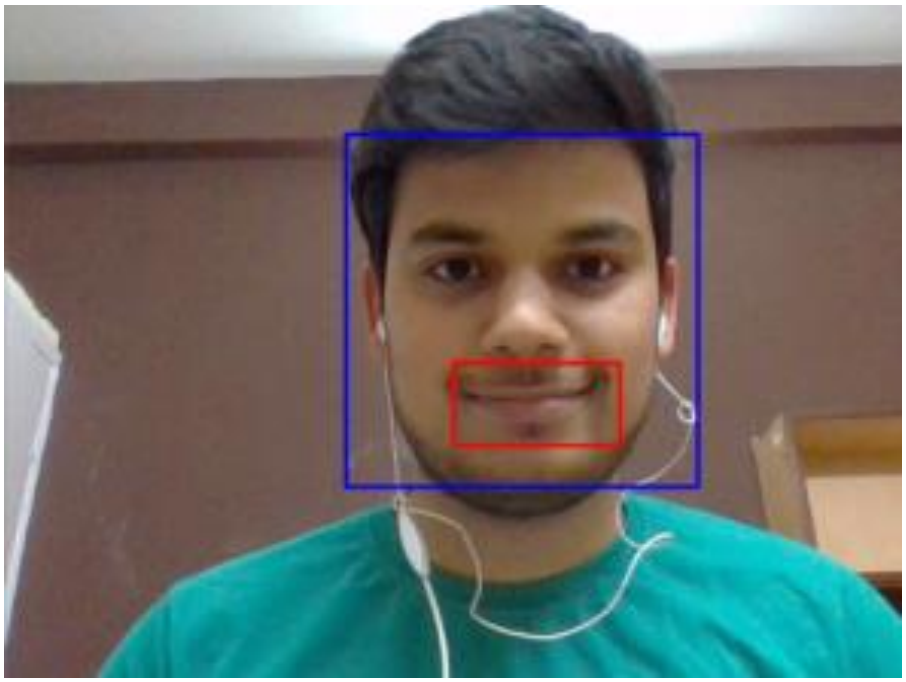

Рис. 1.1. Розпізнавання обличчя

Камера виявляє та знаходить зображення обличчя, як одного, так і в натовпі. На зображенні може бути зображена особа, яка дивиться прямо вперед або в профіль.

Крок 2: Аналіз обличчя

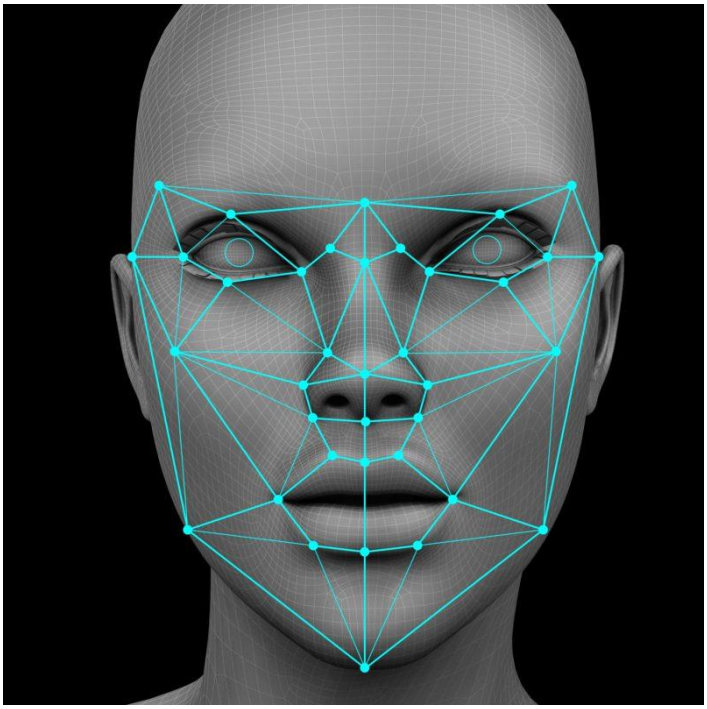

Рис. 1.2. Аналіз обличчя

Далі знімається та аналізується зображення обличчя. Більшість технологій розпізнавання обличчя покладаються на 2D-зображення, а не на 3D-зображення, оскільки вони можуть зручніше зіставляти 2D-зображення з загальнодоступними фотографіями або фотографіями в базі даних.

Програма читає геометрію обличчя. До ключових факторів відносяться відстань між очима, глибина очних западин, відстань від чола до підборіддя, форма скул, контур губ, вух і підборіддя. Мета полягає в тому, щоб визначити орієнтири обличчя, які є ключовими для розрізнення вашого обличчя.

Крок 3: Перетворення зображення на дані

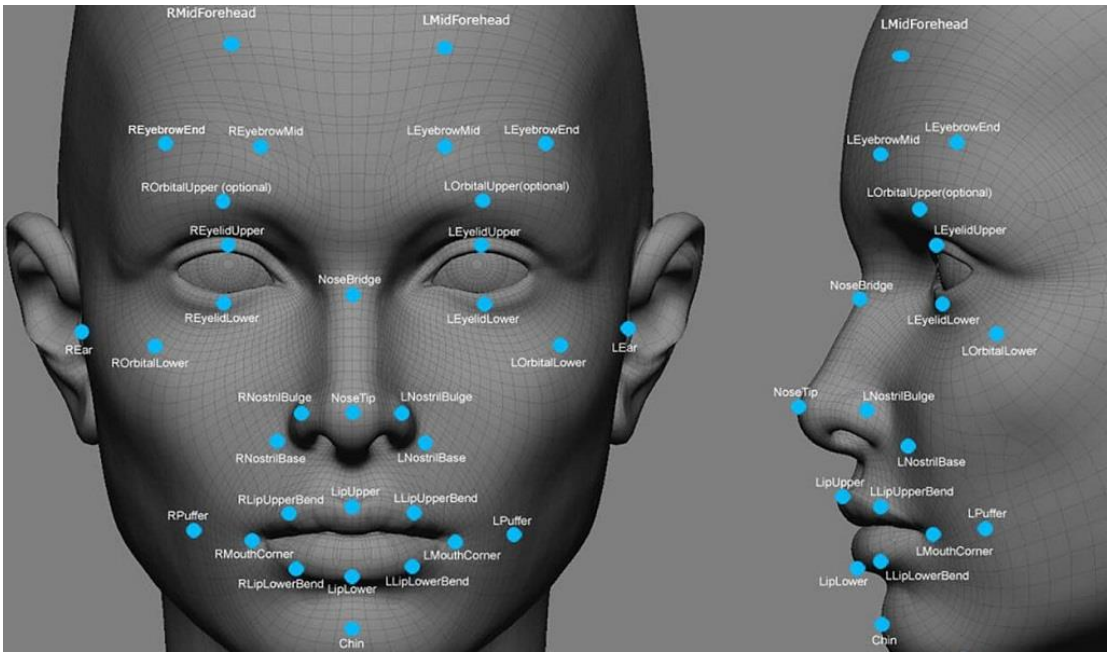

Рис. 1.3. Перетворення зображення на дані

Процес захоплення обличчя перетворює аналогову інформацію (обличчя) у набір цифрової інформації (даних) на основі рис обличчя людини. Аналіз вашого обличчя по суті перетворюється на математичну формулу. Цифровий код називається відбитком обличчя. Подібно до того, як відбитки пальців унікальні, кожна людина має власний відбиток обличчя.

Крок 4: Пошук відповідності

Рис. 1.4. Пошук відповідності

Потім ваш відбиток обличчя порівнюється з базою даних інших відомих облич. Наприклад, ФБР США має доступ до 650 мільйонів фотографій, взятих із різних державних баз даних.

У Facebook будь-яка фотографія, позначена тегом імені людини, стає частиною бази даних Facebook, яка також може використовуватися для розпізнавання обличчя. Якщо ваш відбиток обличчя збігається із зображенням у базі даних розпізнавання обличчя, робиться визначення.

З усіх біометричних вимірювань розпізнавання обличчя вважається найбільш природним. Інтуїтивно це має сенс, оскільки ми зазвичай впізнаємо себе та інших, дивлячись на обличчя, а не на відбитки пальців і райдужну оболонку очей. За оцінками, більше половини населення світу регулярно стикається з технологією розпізнавання облич.

Одна з найпопулярніших розробок технологічного лідера світу доказ що технологія інноваційна. У вересні 2017 року на Apple Special Event була представлена технологія FaceID. Ця технологія розпізнавання активно впроваджена у соціальних мереж для позначення людей на фотографіях.

Результатами широкого використання Face ID стало створення різного роду бібліотек та Application program interface (API) для розпізнавання обличь. Розроблені рішення поділяються на спеціально для якоїсь області чи

універсальні, написані під певну мову програмування чи з підтримкою всіх популярних мов. Часто серед усього різноманіття засобів важко зрозуміти, який з них підходить найкраще для вирішення конкретної проблеми.

Робота присвячена проблематиці систем розпізнавання обличчя, а саме розпізнавання обличь на основі Vision.api.

1.2. Аналіз технології розпізнавання обличчя

Проаналізуємо відомі рішення для задачі розпізнавання обличчя на основі машинного навчання, такі як:

- Apple Face ID;
- Microsoft Face API;
- Aware Nexa|Face;
- Samsung Face Recognition.

Система Apple Face ID призначена для авторизації власника телефону.

Face ID від Apple – це технологія розпізнавання обличчя, яка була запущена на iPhone X у 2017 році. Ця технологія замінила систему сканування відбитків пальців Apple Touch ID для останніх iPhone компанії, включаючи iPhone 13 mini, 13, 13 Pro та 13 Pro Max, і вона буде ймовірно, також буде знайдено на майбутніх iPhone.[5]

Face ID використовує «систему камер TrueDepth», яка складається з датчиків, камер і точкового проектора у верхній частині дисплея iPhone у вирізі для створення детальної 3D-карти вашого обличчя. Кожного разу, коли ви дивитесь на свій телефон, система проводить безпечну перевірку автентифікації, дозволяючи розблокувати ваш пристрій або авторизувати платіж швидко й інтуїтивно, якщо вона вас розпізнає.

Рис. 1.5. Демонстрація роботи точкового проектора

У Face ID бере участь ряд апаратних факторів, таких як система камери TrueDepth, нейронні мережі та чіпи Bionic. Face ID також адаптується до змін вашої зовнішності, таких як використання косметичного макіяжу або зростання волосся на обличчі.

Якщо у вашій зовнішності є більш значні відмінності, як-от гоління бороди, Face ID підтвердить вашу особу за допомогою пароля, перш ніж оновити дані вашого обличчя.

Рис. 1.6. Основні етапи роботи системи Face ID

Особливості системи: апаратна розробка, нативна робота системи з камерою та датчиками

Переваги:

- висока ефективність розпізнавання;
- розроблене спеціальне апаратне забезпечення;
- захищеність системи від обману;
- можливість розпізнавати при поганому освітлені.

Недоліки:

- технологія запатентована, та може використовуватись лише на продукції компанії Apple.

Сервіс Microsoft Face API. Служба Azure Face надає алгоритми штучного інтелекту, які виявляють, розпізнають і аналізують людські обличчя на зображеннях. Програмне забезпечення для розпізнавання обличчя є важливим у багатьох різних сценаріях, таких як підтвердження особи, безконтактний контроль доступу та розмивання обличчя для конфіденційності.[6]

Службу Face API можна використовувати через SDK клієнтської бібліотеки або напряму викликавши REST API. Дотримуйтесь інструкцій швидкого старту, щоб почати.

Ідентифікація обличчя може відповідати типу "один-до-багатьох" одного обличчя на зображенні набору облич у безпечному сховищі. Кандидати на збіги повертаються на основі того, наскільки точно дані про їхні обличчя відповідають обличчю запиту. Цей сценарій використовується для надання доступу до будівлі чи аеропорту певній групі людей або перевірки користувача пристрою.

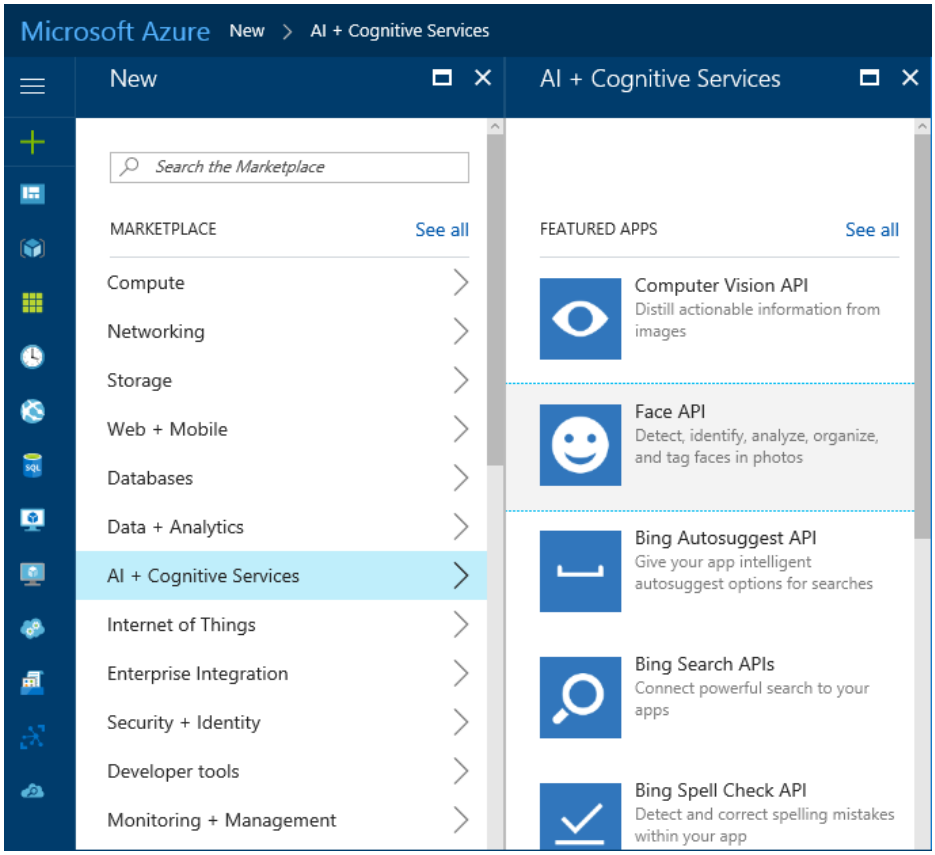

Рис. 1.7. Головне вікно системи Face API

Приклад бази даних облич яку використовує Face API під назвою «myfriends» показано на рис. 1.8. Кожна група може містити до 1 мільйона об’єктів різних осіб. Кожен об’єкт людини може мати до 248 зареєстрованих облич.

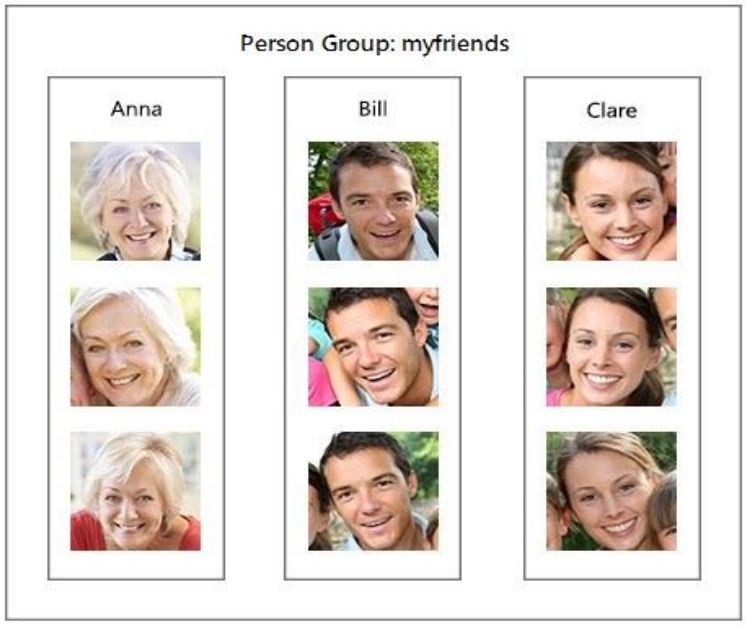

Рис. 1.8. Приклад бази даних облич яку використовує Face API

Особливості системи: задумувалась як інструмент, на основі якого розробники можуть будувати системи розпізнавання обличчя. У зв'язку з цим був створений у вигляді API.

Переваги:

- хороша ефективність розпізнавання;
- наявність величезної кількості засобів і методів для розпізнавання;
- доступність;
- усі операції відбуваються на стороні серверів Microsoft.

Недоліки:

- не є закінченим рішенням, тобто це тільки інструмент, а не готовий продукт;
- відсутня можливості додаткових налаштувань;
- залежність розробки від стороннього API.

Система Aware Nexa|Face. AwareABIS™ — це автоматизована система біометричної ідентифікації (ABIS), яка використовується для великомасштабної біометричної ідентифікації та дедуплікації з підтримкою модальностей відбитків пальців, обличчя та райдужної оболонки ока. [7]

Його високомодульна архітектура дозволяє налаштувати та оптимізувати його для цивільних або кримінальних застосувань.

Він має гнучкість для використання високопродуктивних алгоритмів Aware, перевірених NIST Nexa™, зіставлення обличчя і райдужної оболонки ока, а також алгоритмів відбитків пальців найвищого рівня від сторонніх постачальників.

Разом ці функції роблять його одним з найкращих постачальників цієї технології на ринку не лише завдяки широким можливостям налаштування, але й запобіганню прив'язці до постачальника. На рис. 1.9. показано принцип роботи програми

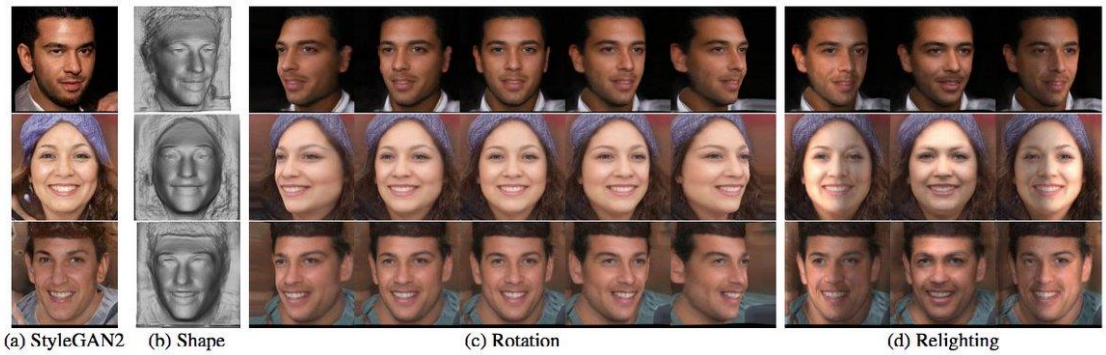

Рис. 1.9. Принцип роботи системи Nexa|Face

Особливості системи: комплексна біометрична системи з можливістю впровадження сторонніх модулів, а також кросплатформність цієї розробки.

Переваги:

- висока ефективність розпізнавання;
- наявність додаткових систем розпізнавання;
- висока надійність і захищеність системи;
- кросплатформність.

Недоліки:

- комплексність системи;
- висока вартість.

Система Samsung Face Recognition

Samsung також використовує схожу систему з Apple включає два компоненти: інфрачервоний світлодіод та ірисову камеру. Перший, можна сказати, схожий на заливний освітлювач найновішого iPhone. [8]

ІЧ-світлодіод працює так само, як ілюмінатор потоку. Однак він показує різницю, лише фокусуючи інфрачервоне світло на очах, на відміну від заливного освітлювача, який фокусується на всьому обличчі.

Ірис-камера — це не що інше, як інфрачервона камера, завдання якої — зробити знімок вашої райдужної оболонки.

Процес починається з інфрачервоного світлодіода, який допомагає камері райдужної оболонки, освітлюючи очі, щоб камера могла правильно захопити малюнок райдужної оболонки. У Samsung є можливість вибору між обома скануваннями райдужної оболонки або лише одним.

Рис. 1.10. Головне вікно системи Samsung Face Recognition

Особливості системи: розроблена системи отримала свій подальший розвиток у системі розпізнавання зіниці.

Переваги:

- висока ефективність розпізнавання;
- наявність розпізнавання за допомогою зіниці;
- додатковий шар захисту за допомогою Knox системи і захищеної папки.

Недоліки:

- можливість хибного розпізнавання;
- проблема з розпізнаванням при поганому освітленні;
- закрита технологія тільки для продукції Samsung.

1.3. Сфери використання Face Recognition

Переваги безпеки спонукали багато галузей до впровадження технології розпізнавання облич у свою повсякденну роботу. Ось де можна застосувати цю технологію.[9]

Уряд та держава

Правоохоронні органи у США, такі як ФБР, використовують технологію розпізнавання облич, щоб шукати списки злочинців і допомагати остаточно ідентифікувати особу, яка вас цікавить. Вони можуть зіставити нещодавно зроблене фото чи відео з базою даних наявних фотографій, щоб допомогти ідентифікувати цю особу.

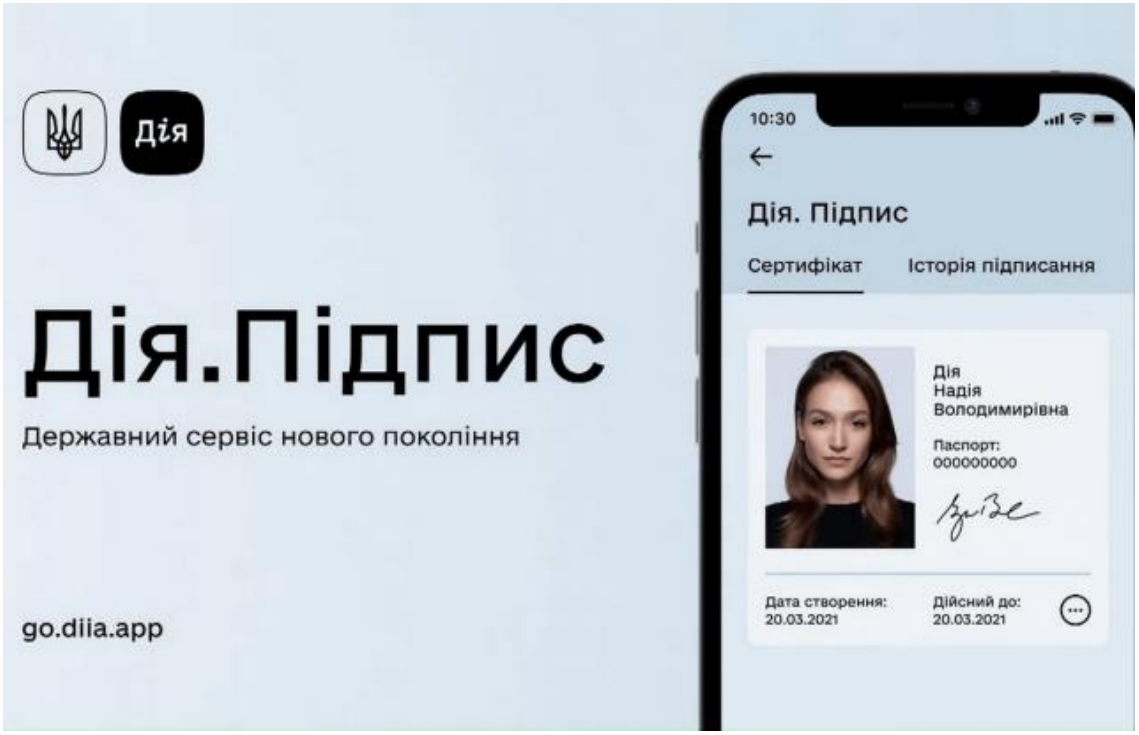

Рис. 1.11. Приклад використання технології у сфері уряду та держави

Автомобільна безпека

Вантажівки часто перевозять важливі предмети, чи то інформацію, чи готівку, і покладаються на технологію розпізнавання обличчя, щоб запобігти крадіжці або навіть переконатися, що очі водія дивляться на дорогу.

З іншого боку, технологія розпізнавання обличчя іноді використовується додатками для обміну поїздками, щоб підтвердити, що даний пасажир є тим, ким він себе видає. Або ж, та сама технологія може гарантувати, що пасажир наближається до потрібного водія.

Рис. 1.12. Приклад використання у сфері автомобільної безпеки

Управління доступом

Крім автомобілів і смартфонів, розпізнавання обличчя можна використовувати вдома, щоб надати доступ до певних пристроїв розумного дому на додаток до входу в сам будинок. Оскільки ця технологія стає все більш досконалою, люди почуватимуться краще захищеними від вторгнень у житло та пограбувань.

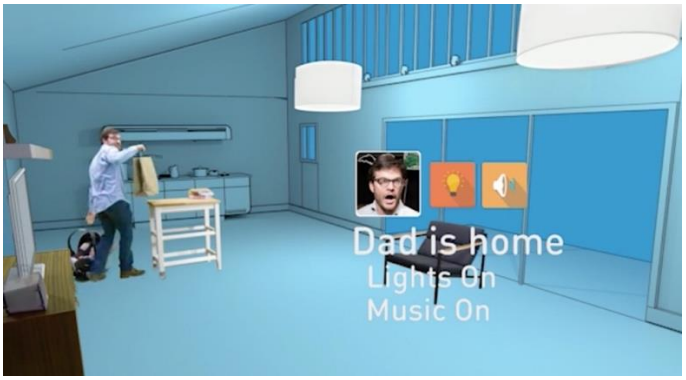

Рис. 1.13. Приклад використання технології у сфері управління доступом

Імміграція

Імміграційні офіси існують як розширення більш відомих урядових сегментів. Технологія розпізнавання облич використовується для посилення контролю на кордоні, особливо коли йдеться про злочинців і зацікавлених осіб, які намагаються перетнути кордон.

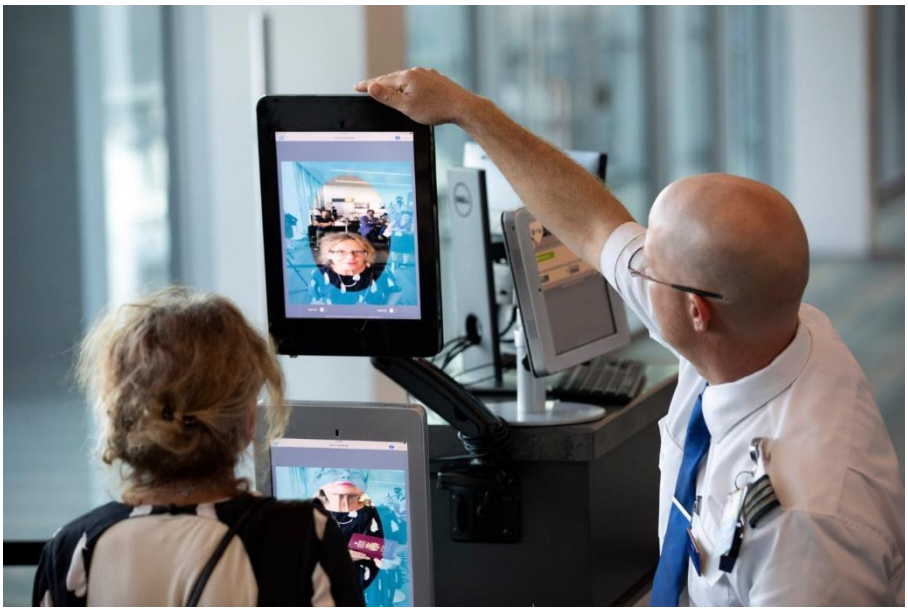

Рис. 1.14. Приклад використання технології у сфері імміграції

Освіта

Крім федеральної та місцевої безпеки, програми розпізнавання обличчя можуть бути найбільш помітними в освітньому секторі.

Все більше шкіл уже використовують камери, які використовують програмне забезпечення для розпізнавання облич, щоб ідентифікувати учнів, персонал, неавторизованих осіб і навіть поведінку, яка може становити загрозу безпеці. Це одна з багатьох нових технологічних тенденцій, які трансформують освіту.

Для шкіл, які використовують цю технологію, головною перевагою, яку вони бачать, є відстеження відвідуваності студентів, а також підтримка безпеки свого кампусу. На жаль, технологія може бути дуже упередженою, і дослідження показали докази заборони програмного забезпечення.

Рис. 1.15. Приклад використання технології у сфері уряду та держави

Роздрібна торгівля

Хоча Україна відстає у використанні розпізнавання обличчя для роздрібної торгівлі, інші країни, як-от Японія, роблять це вже досить давно. Наприклад, торговельні автомати в Японії можуть рекомендувати споживачеві напої за допомогою технології розпізнавання обличчя, щоб приблизно визначити стать і вік клієнта.

Наприклад в США Amazon відкрив свій перший магазин Amazon Go у 2018 році. Тут немає виписки, і магазин повністю залежить від датчиків, щоб визначити, що клієнт бере та купує.

Враховуючи це, можна з упевненістю припустити, що Amazon має достатньо розуму, щоб почати відображати поведінку покупців у найближчому майбутньому, коли онлайн-покупці зв’яжуться з офлайн-обличчями.

Також слід зауважити що українська мережа «Сільпо» заявила що працює над схожою технологією як і у Amazon, проте через війну процес розробки припинився.

Рис. 1.16. Приклад використання технології у сфері роздрібної торгівлі

Охорона здоров'я

Застосування технології розпізнавання облич використовується в лікарнях, особливо в тих, хто працює в інтернатах. Програмне забезпечення слугує для відстеження всього, що відбувається в лікарні, забезпечуючи безпеку пацієнтів і безпеку приміщення.

Якщо пацієнт покидає лікувальний заклад без ідентифікації, розпізнавання обличчя може допомогти швидко ідентифікувати та знайти його, щоб запобігти будь-якій шкоді.

Рис. 1.17. Приклад використання технології у сфері охорони здоров’я

1.4. Взлом системи розпізнавання облич

Найпопулярнішим способом перевірки біометрії обличчя є створення підпису, що складається з геометрії та основних рис, включаючи очі, ніс, губи та їх розподіл на обличчі. Виявлення схожості двох облич можливо шляхом порівняння підписів, створених за допомогою математичних формул, щоб вони не реагували на старіння користувача, зміну його бороди, зачіски чи ваги тіла.[10]

Механізми автентифікації використовують алгоритм порівняння підписів, щоб на основі ступеня подібності визначити, чи справді ми схожі на людину, за яку себе видаємо.

Через складність задачі неможливо чітко відповісти (так / ні). Системою-виконавцем приймається рішення про прийнятність певної схожості обличчя з урахуванням можливих відмінностей у зовнішності. Ключем до ефективності системи є встановлення межі, за якої можна знайти найменшу кількість неправильно класифікованих випадків (співвідношення хибно-позитивних і хибно-негативних результатів). Немає відомого ідеального алгоритму, який точно визначив би, чи обличчя на двох фотографіях насправді належить одній людині.

Є багато випадків, коли загальнодоступні системи на основі біометричних даних виходили з ладу, наприклад, автентифікація голосу, запроваджена HSBC, яку ошукав репортер BBC та його брат, який змінив свій голос, щоб отримати авторизацію від системи.

Основною метою зловмисників біометричних систем є видати себе за іншу або неіснуючу особу, щоб отримати автентифікований доступ до ресурсів, не розкриваючи свою справжню особу. Найпоширенішим випадком крадіжки особистих даних є отримання короткострокової позики в невеликих компаніях, у яких процес перевірки короткий і вимагає лише пред’явлення дійсного посвідчення особи.

Простої маски, надрукованої на папері, достатньо, щоб представити задане обличчя в системі, не змінюючи програму чи пристрій. Щоб обійти

перевірку живучості, іноді вимагалися деякі додаткові дії, наприклад закочування очей або згинання роздруківки.

Незважаючи на те, що це виглядає смішно, навіть цієї процедури було достатньо, щоб обдурити деякі системи розпізнавання облич. Використання більш складних і випадкових жестів у перевірці живості вимагає можливості маніпуляції обличчям у відеопотоці.

Проєкт із відкритим кодом під назвою faceswap, дозволив замінити існуючі обличчя на цифрові, згенеровані за допомогою технології deepfake. Початкові випробування показали, що якість і різноманітність фотографій людини є вирішальними для успішного вивчення нашої моделі AI, яка зосереджена на зміні визначених облич.

Для найпростіших випадків, як-от коротке відео людини, яка стоїть перед обличчям, запис випадкової розмови в кафе за допомогою камери iPhone надасть достатньо навчального матеріалу.

Надання недостатньої кількості зображень із різними кутами обличчя на етапі навчання може призвести до дуже поганої якості вихідного відео під час процесу генерації.

Ще одна можливість досягти тієї ж мети — використовувати програмне забезпечення для захоплення руху та анімації обличчя, яке професійно використовується в кіноіндустрії.

Вищезазначена вразливість стала можливою через відсутність або неправильну реалізацію механізму визначення того, чи є одна й та сама особа на всьому записі. У разі використання це може обійти механізми перевірки особи та потенційно пройти процес із підробленим посвідченням особи та лише кількома кадрами жертви.

Вчені з Університету Меріленда винайшли светр, який може приховати свого власника від систем розпізнавання облич. Нове дослідження показує, що можна створити одяг, який може блокувати розпізнавання обличчя за допомогою камер, розміщених у громадських місцях.

Експерти створили заплутану картину для систем розпізнавання обличчя на основі ШІ. Зображення, про яке йде мова, нагадує фотографію

людей на овочевому ринку з-за прилавка, але насправді мова йде про абсолютно синтетичне зображення. Демонстрція роботи светру зображена на рис. 1.18.

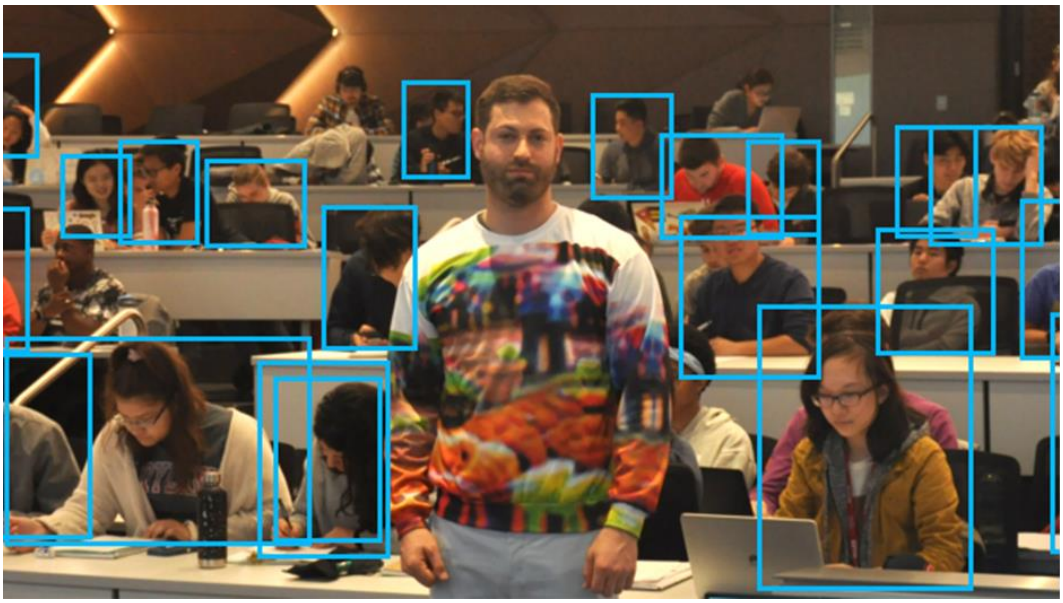

Рис. 1.18. Демонстрація роботи светру

Алгоритм YOLOv2 був обдурений, ей конкретний візерунок був нанесений на светр. У результаті людину, яка носить цей светр, не вдалося виявити системами розпізнавання обличчя на основі штучного інтелекту. Пост в Twitter ентузіастів з роботою светру (рис. 1.19.)

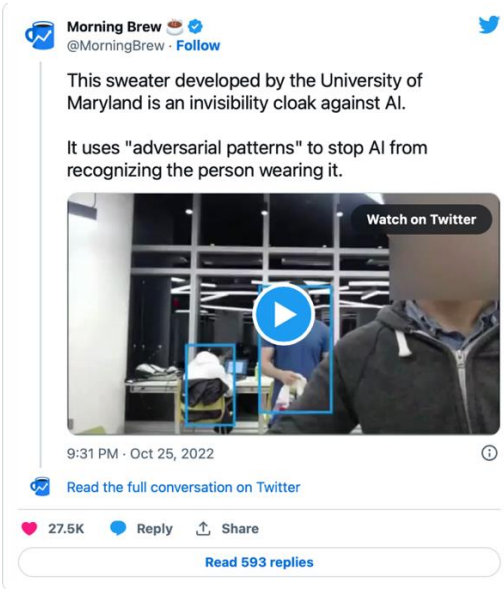

Рис. 1.19. Пост в Twitter ентузіастів з роботю светру

Дослідники описують свій новий винахід так: «Цей стильний светр ідеально зігріє вас взимку в офісі чи в дорозі. Його водонепроникна підкладка з мікрофлісу, сучасний крій і візерунки без штучного інтелекту допоможуть вам сховатися від детекторів об’єктів».

1.5. Використання системи розпізнвання облич в Китаї

Китайська система розпізнавання облич реєструє майже кожного громадянина країни, маючи широку мережу камер по всій країні. Витік бази даних(рис 1.20.) у 2019 році дав уявлення про те, наскільки розповсюдженими є інструменти стеження в Китаї: понад 6,8 мільйона записів за один день, зроблених з камер, розташованих навколо готелів, парків, туристичних місць і мечетей, містять відомості про людей віком від 9 років. днів.

Рис. 1.20. Витік бази даних китаю

Уряд Китаю звинувачують у використанні розпізнавання облич для скоєння звірств проти мусульман-уйгурів, покладаючись на технологію для проведення «найбільшого масового ув'язнення меншин у світі сьогодні».

«Китай використовує розпізнавання облич для визначення профілю уйгурів, класифікації їх за етнічною приналежністю та виділення для відстеження, жорстокого поводження та затримання», — заявила двопартійна група з 17 сенаторів у листі до держсекретаря Майка Помпео в березні. 11. «І ці технології розгорнуті на службу антиутопічному баченню управління технологіями, яке використовує економічні переваги Інтернету за відсутності політичної свободи та розглядає технологічні компанії як інструменти державної влади».

Агресивний розвиток і використання в Китаї розпізнавання облич відкриває вікно в те, як технологію, яка може бути водночас безпечною та корисною (скажімо, Face ID вашого iPhone), можна також перекрутити, щоб уможливити придушення дій, які звичайна людина може навіть не розглядати як злочинність. Китайські чиновники використовували засоби стеження, щоб публічно присоромити людей, які носять одяг для сну в громадських місцях, назвавши це «нецивілізованою поведінкою».

Покарання за ці дрібні правопорушення є задуманим, кажуть експерти зі спостереження. Загроза публічного приниження через розпізнавання облич допомагає китайським чиновникам орієнтувати понад мільярд людей на те, що вони вважають прийнятною поведінкою, від того, що ви одягаєте, до того, як ви переходите вулицю.

«Ідея полягає в тому, що влада намагається запровадити комплексне спостереження та поведінкову інженерію в масовому масштабі», — сказала Майя Ван, старший науковий співробітник Human Rights Watch з Китаю. «Влада хоче створити таке суспільство, яким їй було б дуже легко керувати».

Ідея про те, що система розпізнавання облич у Китаї автоматично розміщуватиме ваше ім'я та фотографію на рекламному щиті, а також приватно надсилатиме вам штраф, вселяє в людей страх поводитися певним чином.

Технології часто мали таку здатність впливати на поведінку, навіть за межами Китаю.

Поведінкова інженерія — це концепція поєднання технологій і психології, щоб спонукати людей діяти певним чином, і це те, що ми бачимо щодня. Це очевидно в «темних шаблонах», дизайні інтерфейсу, наприклад, прихованих кнопках відмови, які обманом змушують людей надати свої особисті дані.

Але є ключова різниця в тому, як поведінкова інженерія здійснюється в США порівняно з Китаєм і його розпізнаванням обличчя.

У США це виражається в комерційній манері, тоді як у Китаї це зусилля держави, динаміка різна, і кількість потужності різна, але в них є разюча схожість.

У США поведінкову інженерію можна здійснити шляхом накопичення даних про людей і просування або виключення контенту для них на основі передбачуваних рис особистості.

Скандал Facebook у 2018 році з Cambridge Analytica(Рис. 1.21.) виник через те, що нині неіснуюча британська компанія з аналізу даних використовувала дані мільйонів людей для цільової реклами, яка спонукала людей голосувати певним чином.

Рис. 1.20. Скандал Facebook у 2018 році з Cambridge Analytica

У той час як цей тип поведінкової інженерії здебільшого спрямований на продаж продуктів і отримання прибутку, Китай намагається нагнітати страх і контролювати своє населення, і розпізнавання облич відіграє в цьому ключову роль.(Рис. 1.21.)

Рис. 1.21. Використання системи розпізнавння обличчя в повсякденному житті

«Це обумовлює ідею «так влаштовано суспільство», що за вами завжди спостерігають», — сказав Джейк Лаперрук, старший радник Constitution Project. «Це схоже на спробу підкреслити людям, що «ми весь час спостерігаємо, якщо ви коли-небудь зробите щось, щоб нас розлютити, ми побачимо і знайдемо спосіб збентежити вас».

У Китаї ніхто не застрахований від розпізнавання обличчя чи публічного ганьблення, яке супроводжується ним.

Камери, які встановлюють на пішохідних переходах, щоб розпізнавати та розміщувати фотографії тих, хто ходить на вулицю, є звичним явищем, і звіт Abacus у травні 2019 року показав фотографії дітей, які ходять на

цифровому білборді. У місцевій ДАІ заявили, що «з дітьми потрібно поводитись так само, як з дорослими», повідомляє видання.

Китайський парк також покладався на розпізнавання облич, щоб запобігти тому, щоб люди брали занадто багато туалетного паперу, скануючи обличчя людей перед тим, як видавати рулони.

Це щось накшталт системи «все допускається», незалежно від того, наскільки незначною вона є, будь-яке порушення стосується такого роду технологій.

І в той час як у США розпізнавання облич переживає розрахунок через расові упередження та проблеми з правами людини, у Китаї постачальники технологій спостереження пишаються її здатністю виділяти людей різних етнічних груп.

Китайська модель розпізнавання облич, починаючи від репресивного стеження за мусульманами-уйгурами і закінчуючи повсякденним контролем над населенням у 1,4 мільярда людей, також викликає занепокоєння для міжнародної спільноти, заявили американські законодавці.

У вересні минулого року Фонд Карнегі за міжнародний мир виявив, що Китай є основним постачальником систем стеження зі штучним інтелектом(Рис. 1.22.), надаючи цю технологію в 63 країни.

Наступним найближчим постачальником технологій стеження зі штучним інтелектом була японська корпорація NEC, яка продає розпізнавання облич у 14 країн.

У квітні агентство Reuters повідомило, що Amazon купила камери у Dahua, китайської компанії спостереження, внесеної США до чорного списку через звинувачення в тому, що вона допомагає Китаю затримувати та стежити за мусульманами-уйгурами.

Згідно зі звітом, угода на суму 10 мільйонів доларів передбачала 1500 камер для моніторингу поширення COVID-19.

У листі сенаторів від 11 березня вони підняли питання щодо впливу Китаю на те, як слід використовувати розпізнавання облич, оскільки він є найбільшим експортером цієї технології.

Ван зазначив, що хоча розпізнавання облич часто є дочірньою компанією американських технологічних гігантів, таких як Amazon Rekognition або Microsoft, у Китаї є кілька компаній, які вже домінують у галузі. Прийняття країною режиму стеження дозволяє постачальникам розпізнавання облич просувати цю технологію навіть у найтривіальніших цілях.

Рис. 1.22. Система стеження зі штучним інтелектом

З таким рівнем впливу існує занепокоєння, що китайська модель використання розпізнавання облич – широко розповсюдженої мережі, призначеної для публічного ганьби та контролю – може поширитися на решту світу.

«На жаль, Китай продемонстрував готовність використовувати органи, що встановлюють стандарти, у збочений спосіб, щоб нормалізувати глобальну думку про оруеллівську технологію стеження», — заявила група законодавців. [11]

ВИСНОВКИ ДО РОЗДІЛУ 1

Використання можливих уразливостей у методах віддаленої автентифікації у фінансових установах може мати трагічні наслідки, наприклад, крадіжки грошей з рахунків або отримання позик іншими людьми, не виходячи з дому. Одним із найімовірніших сценаріїв шахрайства може бути купівля крадених посвідчень особи на нелегальних сайтах у даркнеті та спроба взяти кредити.

В першому розділі було розкрито поняття системи розпізнавання обличчя та стан досліджень сьогодення. Проведено аналіз технологій розпізнавання обличчя, визначено переваги та недоліки кожної системи. Обрано технологію для реалізації, та запропоновано проект системи розпізнавання обличчя на мобільному пристрої під управлінням IOS. Система досягає високої продуктивності в порівнянні з іншими системами на мобільних пристроях з Android. Крім того, алгоритм API забезпечує високу продуктивність у виявленні та виділенні обличчя.

У майбутньому запропонована система розпізнавання облич буде вдосконалена, щоб розпізнавати людей у багатьох сферах та на вхідних зображеннях незалежно від напрямку та розмірів зображень

РОЗДІЛ 2

ПРОЕКТУВАННЯ ЗАСТОСУНКУ

2.1. Опис вимог до застосунку

Застосунок для демонстрації роботи технології розпізнавання обличчя повинен відповідати наступним функціональним вимогам:

– застосунок повинен дозволяти користувачам швидко використовувати всі функції без реєстрації.

– застосунок повинен відображати інформацію про при зчитування даних обличчя

– застосунок повинен мати функцію розпізнавання обличчя.

– застосунок повинен мати функцію пошуку облич на зображенні/камері.

– застосунок повинен мати функцію побудови маски обличчя.

– застосунок повинен дозволяти користувачу розпізнавати обличчя

У результаті аналізу функціональних вимог та огляду аналогів, сформовано наступні нефункціональні вимоги:

– застосунок повинен мати зручний та мінімалістичний інтерфейс;

– застосунок повинен бути розроблений на мові програмування SWIFT;

– застосунок повинен працювати на IOS 11 та вище.

2.2. Вибір шаблону проектування

Розробники завжди віддають перевагу розробці програми із застосуванням шаблону архітектури програмного забезпечення. Архітектурний шаблон надає модульність файлам проекту та гарантує, що всі коди будуть охоплені модульним тестуванням.

Це спрощує розробникам завдання підтримувати програмне забезпечення та розширювати функції програми в майбутньому. MVC (Model — View — Controller), MVP (Model — View — Presenter) і MVVM (Model — View — ViewModel) є найпопулярнішим і визнаним у галузі шаблоном архітектури серед розробників.

Шаблон «Модель—Вид—Контролер» (MVC) (рис. 2.1.)

Шаблон MVC пропонує розділити код на 3 компоненти. Під час створення класу/файлу програми розробник повинен класифікувати її за одним із наступних трьох рівнів:

Модель: цей компонент зберігає дані програми. Він не знає про інтерфейс. Модель відповідає за обробку логіки домену (справжні бізнес-правила) і зв’язок із базою даних і мережевими рівнями.

Перегляд: це рівень інтерфейсу користувача (інтерфейс користувача), який містить компоненти, видимі на екрані. Крім того, він забезпечує візуалізацію даних, що зберігаються в моделі, і пропонує взаємодію з користувачем.

Контролер: цей компонент встановлює зв’язок між представленням і моделлю. Він містить основну логіку програми, отримує інформацію про відповідь користувача та оновлює модель відповідно до потреб.

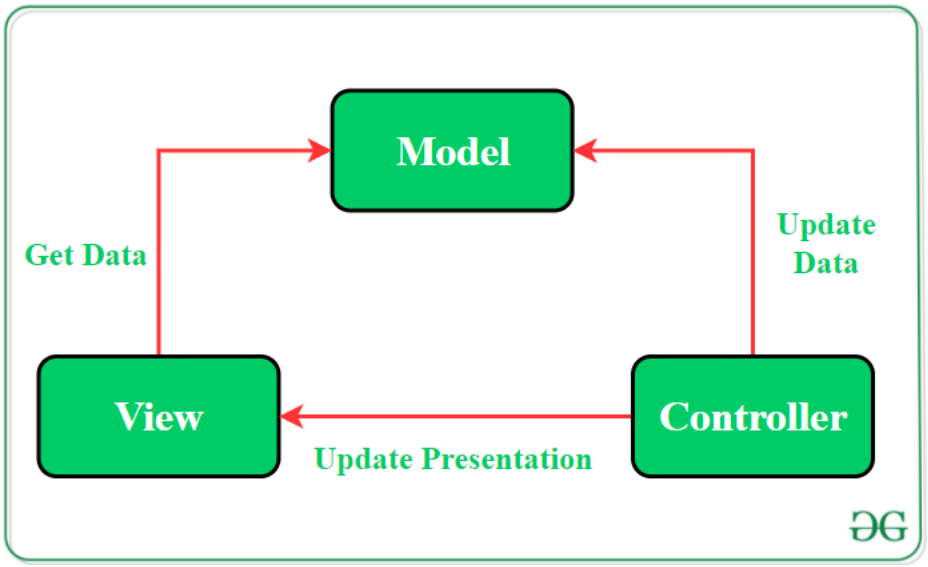

Рис. 2.1. Опис шаблону «Модель—Вид—Контролер»

Шаблон «Модель—Перегляд—Презентатор» (MVP) (рис. 2.2.)

Шаблон MVP долає труднощі MVC і забезпечує простий спосіб структурування кодів проекту. Причина, чому MVP отримав широке визнання, полягає в тому, що він забезпечує модульність, можливість тестування та більш чисту кодову базу, яку можна підтримувати. Він складається з наступних трьох компонентів:

Модель: Рівень для зберігання даних. Він відповідає за обробку логіки домену (справжні бізнес-правила) і зв'язок із базою даних і мережевими рівнями.

Перегляд: рівень інтерфейсу користувача (інтерфейс користувача). Він забезпечує візуалізацію даних і відстеження дій користувача з метою сповіщення Презентатора.

Доповідач: отримує дані з моделі та застосовує логіку інтерфейсу користувача, щоб вирішити, що відображати. Він керує станом представлення та виконує дії відповідно до сповіщень, введених користувачем із представлення.

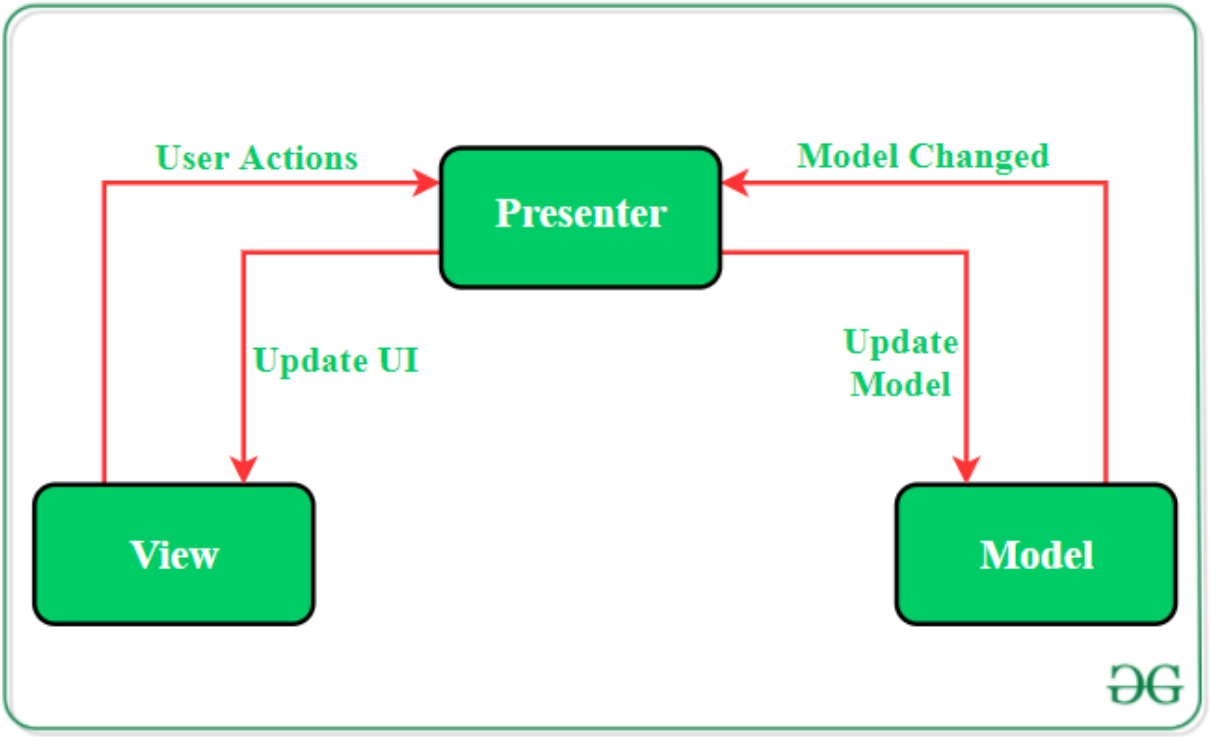

Рис. 2.2. Опис шаблону «Модель—Перегляд—Презентатор»

Шаблон «Модель — Вигляд — Модель перегляду» (MVVM) (рис. 2.3.)

Шаблон MVVM має певну схожість із шаблоном проектування MVP (Model — View — Presenter), оскільки роль Presenter виконує ViewModel. Однак недоліки шаблону MVP були вирішені за допомогою MVVM. Він пропонує відокремити логіку представлення даних (Views або UI) від основної частини бізнес-логіки програми. Окремі рівні коду MVVM:

Модель: цей рівень відповідає за абстракцію джерел даних. Модель і ViewModel працюють разом, щоб отримати та зберегти дані.

Перегляд: мета цього шару — повідомити ViewModel про дії користувача. Цей рівень спостерігає за ViewModel і не містить жодної логіки програми.

Доповідач: відкриває ті потоки даних, які мають відношення до View. Крім того, він служить сполучною ланкою між моделлю та представленням.

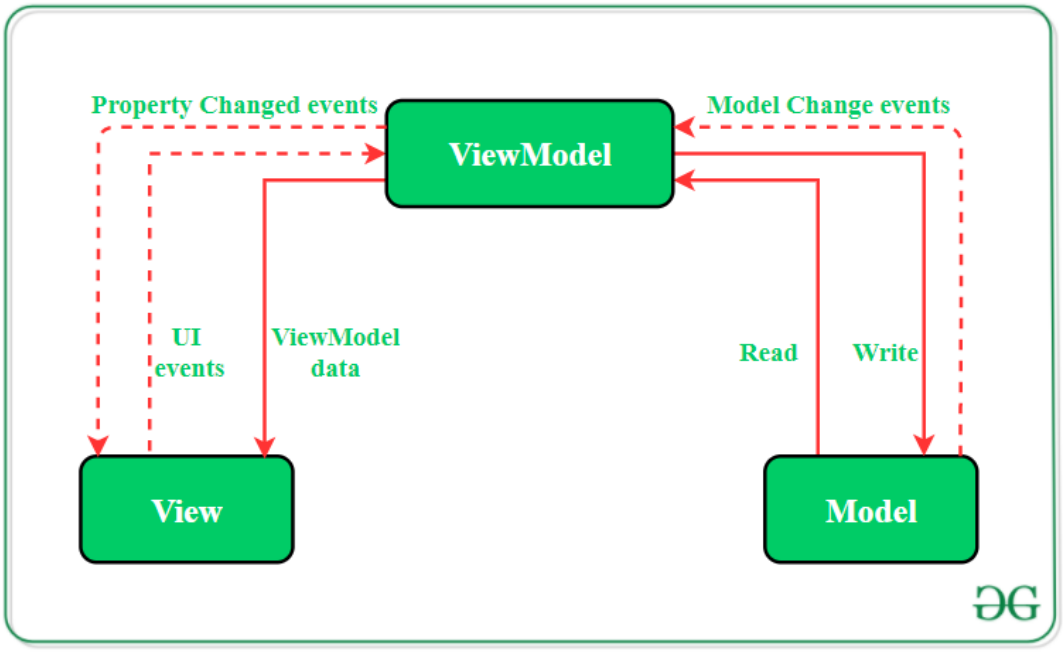

Рис. 2.3. Опис шаблону «Модель—Вигляд—Модель перегляду»

2.3. Вибір та використання середовища розробки

Xcode — це інтегроване середовище розробки (IDE) від Apple для macOS, яке використовується для розробки програмного забезпечення для macOS, iOS, iPadOS, watchOS і tvOS. Спочатку він був випущений наприкінці 2003 року; остання стабільна версія – версія 14.0.1, випущена 26 вересня 2022 року через Mac App Store з macOS Monterey. [12]

Набір програмного забезпечення надається безкоштовно. Зареєстровані розробники можуть завантажити попередні випуски та попередні версії набору через веб-сайт Apple Developer. Xcode містить інструменти командного рядка, які дозволяють розробку в стилі UNIX через програму Terminal у macOS. Їх також можна завантажити та встановити без графічного інтерфейсу користувача.

Рис. 2.4. Іконка застосунку Xcode

Основні особливості

Xcode підтримує вихідний код для мов програмування: C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, ResEdit (Rez) і Swift, з різними моделями програмування, включаючи, але не обмежуючись Cocoa , Carbon і Java. Треті розробники додали підтримку GNU Pascal, Free Pascal, Ada, C#, Go, Perl і D.

Xcode може створювати повні двійкові (універсальні двійкові) файли, що містять код для кількох архітектур із виконуваним форматом Mach-O. Це допомогло полегшити перехід від 32-розрядної версії PowerPC до 64-розрядної версії PowerPC, від PowerPC до Intel x86, від 32-розрядної до 64-розрядної версії Intel і від x86 до Apple Silicon, дозволивши розробникам поширювати одну програму серед користувачів і операційна система автоматично вибирає відповідну архітектуру під час виконання.

Використовуючи iOS SDK, tvOS SDK і watchOS SDK, Xcode також можна використовувати для компіляції та налагодження програм для iOS, iPadOS, tvOS і watchOS.

Xcode містить інструмент GUI Instruments, який працює на основі динамічної системи трасування DTrace, створеної Sun Microsystems і випущеної як частина OpenSolaris.

Xcode також інтегрує вбудовану підтримку керування вихідним кодом за допомогою системи та протоколу контролю версій Git, що дозволяє користувачеві створювати та клонувати сховища Git (які можна розміщувати на сайтах розміщення сховищ вихідного коду, таких як GitHub, Bitbucket і Perforce або самостійно розміщений за допомогою програмного забезпечення з відкритим вихідним кодом, такого як GitLab), а також для фіксації, надсилання та вилучення змін, усе з Xcode, автоматизуючи завдання, які традиційно виконувалися б за допомогою Git з командного рядка.

Основним застосуванням пакету є інтегроване середовище розробки (IDE), яке також називається Xcode. Набір Xcode включає більшу частину документації для розробників Apple і вбудований Interface Builder, програму, яка використовується для створення графічних інтерфейсів користувача.

До Xcode 4.1 набір Xcode включав модифіковану версію колекції компіляторів GNU. У Xcode 3.1 до Xcode 4.6.3 він включав компілятор LLVM-GCC із зовнішніми частинами з колекції компіляторів GNU та генератором коду на основі LLVM.

У Xcode 3.2 і пізніших він включав компілятор Clang C/C++/Objective-C із нещодавно написаними інтерфейсами та генератором коду на основі LLVM, а також статичний аналізатор Clang. Починаючи з Xcode 4.2, компілятор Clang став компілятором за замовчуванням, починаючи з Xcode 5.0, Clang був єдиним компілятором.

До Xcode 4.6.3 набір Xcode використовував GNU Debugger (GDB) як серверну частину для налагоджувача IDE. Починаючи з Xcode 4.3, налагоджувач LLDB також надавався; починаючи з Xcode 4.5 LLDB замінив

GDB як серверну частину за замовчуванням для налагоджувача IDE.Починаючи з Xcode 5.0, GDB більше не постачався.

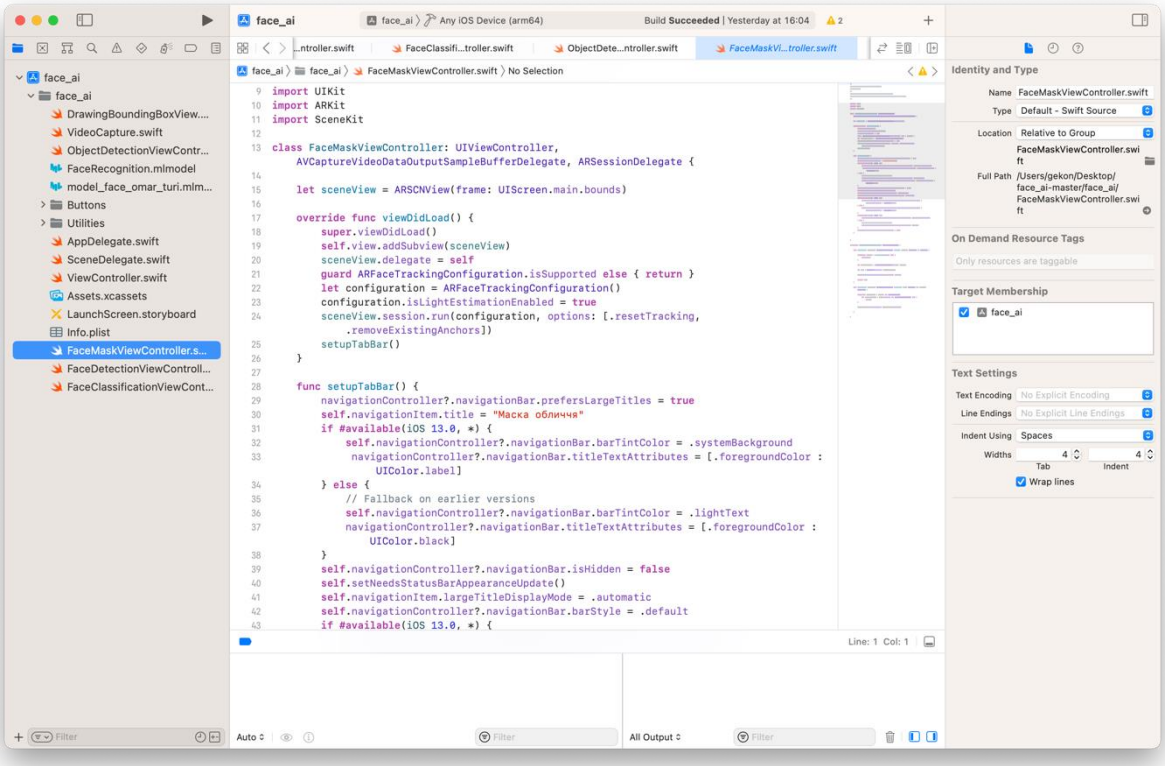

Рис. 2.5. Вид Xcode на прикладі коду мого застосунку

2.4. **Використання гайдлайнів дизайну від Apple (**Apple Human Guidelines)

Розміри екрана iPhone

Протягом перших 5 або 6 років випуску iPhone розмір екрану був цілком прийнятним. Якщо дизайн працював на екрані 320x480, це було золотим стандартом. На сьогоднішній день, кожен рік при виході нової моделі iPhone, приходить і інший розмір екрану.[13]

Ось повний список для довідки:

*дисплей на телефоні технічно 2,61x

DEVICE	FRAME SIZE (E.G. FOR FIGMA)	EXPORT SCALING
13 Pro Max, 12 Pro Max	428 x 926	@3x
13, 13 Pro, 12, 12 Pro	390 x 844	@3x
13 Mini, 12 Mini	375 x 812	@3x
11 Pro Max, XS Max	414 x 896	@3x
11 Pro, X, XS	375 x 812	@3x
11, XR	414 x 896	@2x
8+, 7+, 6+, 6S+	414 x 736	@3x*
SE (gen 2), 7, 6, 6s	375 x 667	@2x
5, 5s, 5c, SE	320 x 568	@2x
4, 4s	320 x 480	@2x
1, 2, 3	320 x 480	@1x

Рис. 2.6. Список розміру екрану смартфонів Apple

Розмір екранної області – Це «розмір точки» або «@1x» розмір певного пристрою.

Масштабування експорту – Наскільки більшим потрібно зробити растрове зображення (PNG, JPG) під час експорту, щоб максимально використати вищу роздільну здатність деяких пристроїв.

Для розробників та особисті розміри що були обрані на широку аудиторію є найпопулярніший розмір екрана iPhone: 375x812 pt або 375x667 pt але в Xcode є інструменти для уніфікованого дизайну за для того щоб контент виглядав на всіх пристроях Apple однаково гарно.

Саме через це дизайн, який добре працює на вужчому екрані (375 pt), майже напевно буде добре працювати на трохи ширшому екрані (414 pt), але навпаки, ні. Тому завжди краще спочатку розробити дизайн для менших екранів, а потім ще раз перевірити та налаштувати для більших екранів. Оскільки висота є меншим обмеженням, не має значення, скажімо, 667 чи 812 пікселів у висоту.

iOS Points проти пікселів

«Point» — це міра, за допомогою якої дизайнери порівнюють розміри шрифтів і елементів інтерфейсу на пристроях iOS. «Піксель» — це крихітний квадрат світла, з якого складається екран iPhone та і взагалі будь якого смартфону.

Менші пікселі означають чіткіше зображення, що чудово. Але якщо просто зменшити свої пікселі, усе на екрані також стане меншим! Щоб збалансувати це, слід вимірювати розмір елементів на екрані в балах.

Коли пікселі стали вдвічі меншими за висоту/ширину, можливо просто використовувати квадрат пікселів 2x2 для кожної точки (це називається @2x).

І як тільки пікселі стали приблизно на третину нижчими за висоту/ширину, слід використовувати квадрат пікселів 3x3 для кожної точки. Візуальний приклад відображення різної щільності пікселів зображено на Рис. 2.7

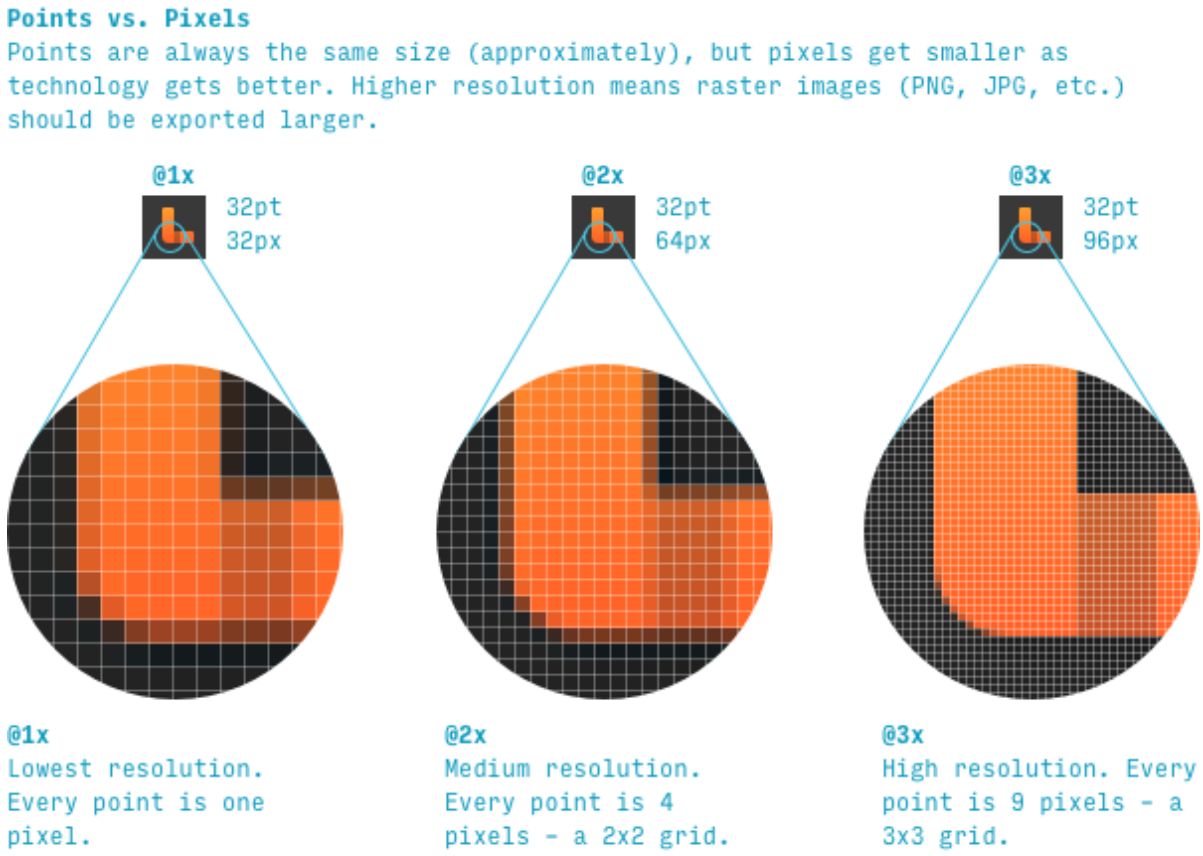

Рис. 2.7. Візуальний приклад відображення різної щільності пікселів

Points — це одиниця, яка дозволяє нам мати екрани з вищою роздільною здатністю без того, щоб усі елементи на сторінці просто зменшилися.

Макет сторінки iPhone

Хоча різні програми для iOS мають різні макети, багато стандартних сторінок як і мій застосунок матимуть приблизно такий макет(Рис. 2.8):

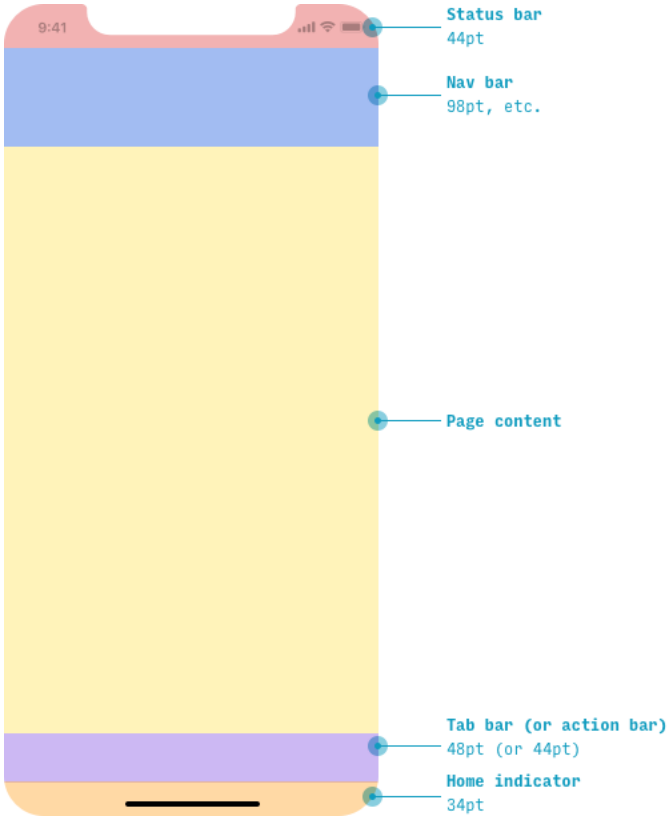

Рис. 2.8. Макетр сторінки застосунку

Модальні аркуші iOS

Деякі завдання включають один екран – або лінійну серію екранів – які розробник може хотіти щоб користувачі завершили, не виходячи з контексту, у якому вони були.

В iOS 15 з’явився ідеальний елемент інтерфейсу користувача для цього:

Модальний аркуш — це звичайна сторінка, яка (а) ковзає знизу вгору, покриваючи майже всю попередню сторінку, але (б) залишає попередню сторінку видимою, але заглибленою, на задньому плані. Приклад вигляду Модального аркушу зобоажено на рис. 2.9.

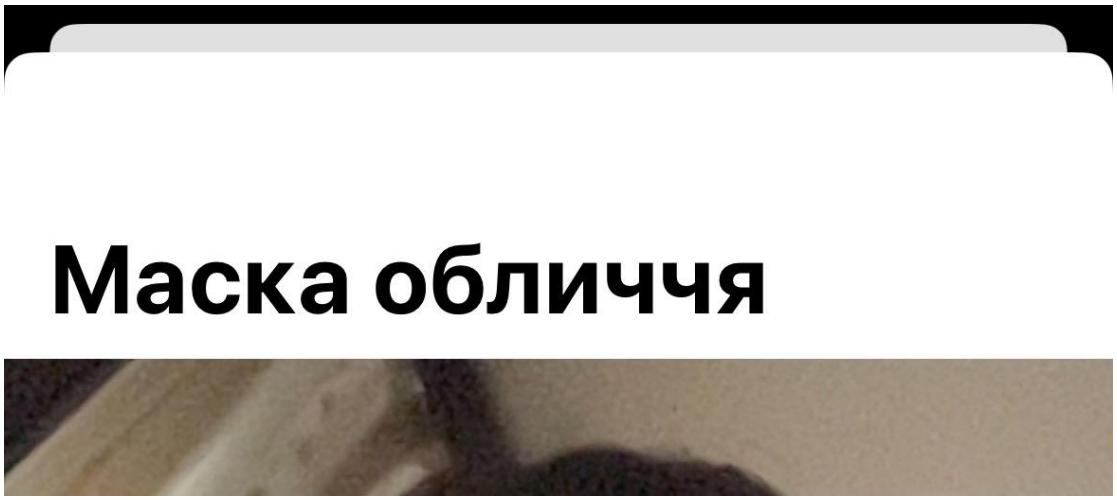

Рис. 2.9. Приклад вигляду Модального аркушу

Модальний аркуш можна закрити:

1) Натисканням кнопки «закрити» вгорі (вгорі це «Скасувати» у верхньому правому куті)

2)Провести пальцем вниз по самій модальній картці (саме цей метод використовую я у застосунку)

Mac Catalyst

Одним із останніх оновлень є додавання розділу «Catalyst», якого не було минулі роки.

По-перше, визначає деякі конкретні речі, на які слід звернути увагу, як-от підтримка багатозадачності, підтримка перетягування та реагування на комбінації клавіш. Надає усі види вдосконалень, які можливо отримати безкоштовно з програмою для Mac. Такі речі, як «Системні параметри», «Розділений перегляд», «Файловий браузер» тощо. Навіть містить інформацію про звичайні речі, як-от завантаження (це вкаже, як правильно обробляти стани завантаження вашої програми).[14]

Групує умовності та шаблони проектування(рис 2.10.), які мають найбільший вплив на адаптацію, у 4 ключових областях:

- навігація;
- введення та взаємодія користувача;
- меню;

- масштабування вмісту.

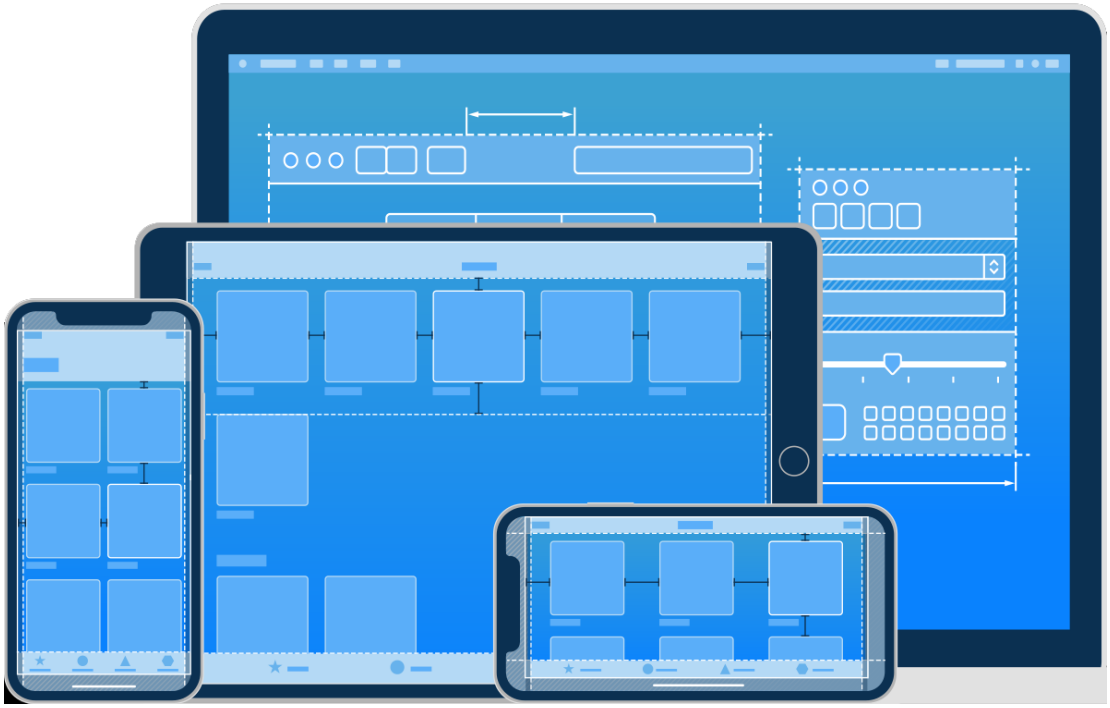

Рис. 2.10. Шаблони проектування

Пристрої iOS мають різні розміри екрана, і люди можуть використовувати їх у книжковій або альбомній орієнтації. У пристроях від краю до краю, як-от iPhone X і iPad Pro, дисплей має округлені кути, які точно відповідають загальним розмірам пристрою. Інші пристрої — такі як iPhone SE та iPad Air — мають прямокутний дисплей.

Якщо програма працює на певному пристрої, інструкції у Visual Design гарантуватимуть, що вона працює на екрані будь-якого розміру для цього пристрою. Іншими словами, програма лише для iPhone має працювати на кожному розмірі екрана iPhone, а програма лише для iPad має працювати на кожному розмірі екрана iPad.

Інструкції щодо візуального дизайну охоплюють усе в таких категоріях, як адаптивність і макет, анімація, брендинг, колір, темний режим, екрани запуску, матеріали, термінологія, типографіка тощо.

Архітектура програми описує шаблони та методи, які використовуються для проектування та створення програми. Архітектура

додатків Apple дає вам дорожню карту та найкращі практики, яких слід дотримуватися під час створення додатка, щоб у кінцевому результаті отримати добре структурований додаток.

Вся формула, що лежить в основі Рекомендацій щодо людського інтерфейсу, полягає в досягненні узгодженості кожного інтерфейсу та дизайну та адаптації до екосистеми Apple показано на рис. 2.11. Хороша архітектура мобільних додатків є основою всього добре розробленого програмного забезпечення та може забезпечити чудову взаємодію з користувачем, і Apple завжди прагне до цього.

Рис. 2.11. Взаємодія девайсів Apple

Інструкції щодо інтерфейсу користувача диктуватимуть набір правил для загального використання. Взаємодія з користувачем підпадає під це. Взаємодія з користувачем — це те, як користувач діє на систему та як система діє на користувача.

Взаємодія з користувачем відіграє важливу роль у визначенні взаємодії з користувачем, і обидва йдуть рука об руку, щоб забезпечити користувачам бездоганний досвід роботи з програмою. Сама сторінка інструкцій Apple містить інтерактивні елементи, оскільки ви можете побачити приклади всього в дуже інтуїтивно зрозумілій манері. Рекомендації Apple також охоплюють такі речі, як те, як краще проектувати будь-що.

Apple десятиліттями домінувала на ринку побутової електроніки та програмного забезпечення. Вони постійно розробляють технології, які неможливо уявити, що відрізняє їх від інших компаній, які прямо конкурують з ними.

Ви знайдете все, що пов’язано з AirPlay, Apple Pay, доповненою реальністю, CarPlay, Game center, Health Kit, iCloud, покупками в додатку, живими фотографіями, машинним навчанням, картами, Siri тощо в інструкціях Apple щодо людського інтерфейсу в дуже детально та добре викладено.

Apple відома в усьому світі тим, що встановлює стандарти в кожній галузі, в якій вони беруть участь, будь то дизайн, виробництво, розробка чи технології.

2.5. Екосистема продуктів Apple та їх взаємодія

Apple Ecosystem(Рис. 2.12.)не є продуктом Apple. Ви не можете піти в Apple Store і «купити» екосистему Apple, але це досвід користувача та спосіб життя, який є побічним продуктом володіння кількома пристроями Apple. Екосистема в основному така. Спеціальні функції, доступні для ваших пристроїв Apple, які виводять вашу роботу на новий рівень завдяки взаємодії між пристроями Apple.

Рис. 2.12. Apple Ecosystem

Apple може створити екосистему, оскільки Apple не проектує навколо одного пристрою. Вони створюють продукти навколо екосистеми. Раніше він розроблявся навколо одного пристрою, але десь в епоху iPad і iPhone з'явилася екосистема, яка тепер мовчки стала домінуючою рисою досвіду Apple. Крім того, Apple рекламує екосистему не відкрито, а непомітно.

Основою всієї екосистеми Apple є Apple ID(Рис. 2.13.). Apple ID використовується для реєстрації всіх ваших пристроїв Apple. З цього моменту Apple не знатиме, скільки пристроїв ви знаєте, але перевірятиме, чи є вони поруч і чи є пристрої поблизу вашого кола контактів. Така інформація означає, що пристрій буде доступний для вас або когось із ваших близьких.

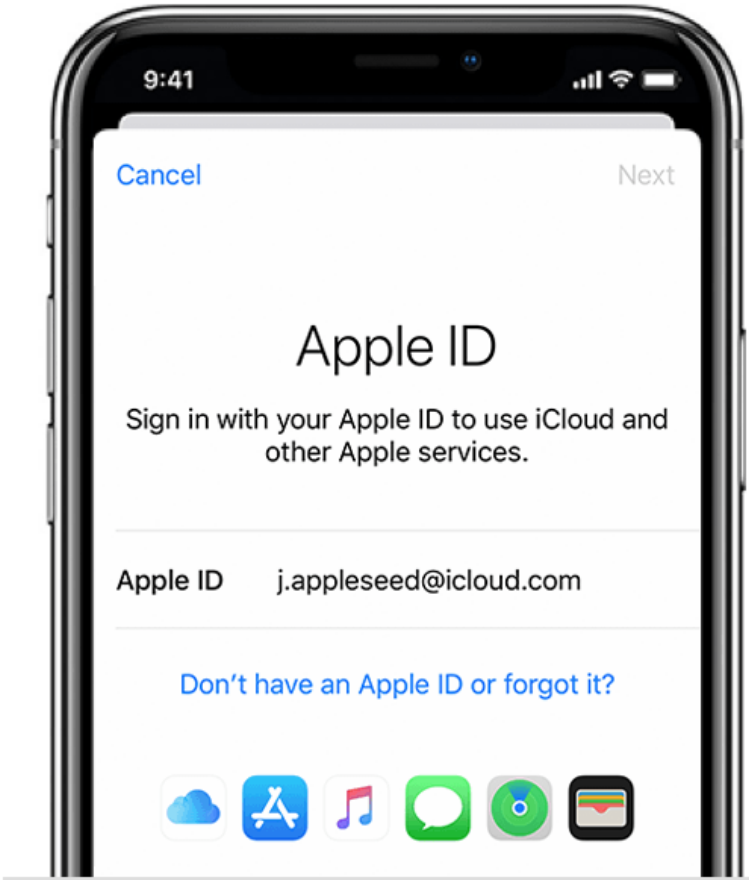

Рис. 2.13 Apple ID

Зайти в екосистему Apple — це як потрапити у світ, де все працює. Наприклад, iPhone працює не тільки сам по собі, але й з усіма іншими пристроями Apple, якими ви володієте.

Програми в iOS також доступні в iPadOS і macOS. Мало того, завдяки синхронізації даних iCloud можливо продовжувати з одного пристрою на інший там, де зупинилися. Ось така екосистема на роботі.

В екосистемі Apple повільно, але впевнено створювала загальні служби та програми для всієї своєї лінійки протягом 10 років(Рис 2.14.). Наприклад, до 2012 року нотатки були частиною поштової програми macOS, яка синхронізувалась із програмою iOS Notes. Це насправді не працює між обома пристроями.

Потім Apple випустила Mountain Lion, який представив окремі нотатки та нагадування, а iChat змінився на повідомлення. Це перші зерна екосистеми Apple. Відтоді macOS, iPadOS та iOS мають спільні програми, такі як Weather, FaceTime, Safari (яка може синхронізувати вкладки між пристроями), Photos і останнє доповнення: Shortcut (яке в майбутньому замінить Automator у macOS).

Рис.2.14. Загальні служби(програми) які вже встановлено автоматично

Завдяки тому, що всі пристрої мають однакові додатки, користувач може очікувати, що всі пристрої працюватимуть майже однаково на різних пристроях. Крім того, дані, які використовують програми, зберігаються в iCloud (хмарні служби Apple), вміст кожної програми буде майже синхронізовано до такої міри, що дає змогу продовжити свою роботу на іншому пристрої, який ви знімете.

Ця функція під назвою Continuity забезпечує безперебійну взаємодію між пристроями. Саме такий бездоганний досвід і є основою екосистеми Apple.

Існують також платні послуги, які надає Apple, як-от Music, TV+, Fitness+, News+ та інші, які знаходяться в розробці (за чутками, це Podcast+), які не тільки безперебійно працюють на кожному пристрої, але й доступні на більшості пристроїв. Таким чином, ви платите єдину щомісячну плату, і всі ваші пристрої можуть мати такі послуги.

Екосистема Apple перевершує інші пропозиції від Android або Microsoft, оскільки жодна інша компанія не охоплює весь пакет. Microsoft може боротися з Apple на настільному фронті, Google на мобільному і, можливо, планшетному фронті, і Sony на ігровому фронті, але Apple має вертикальну інтеграцію, надаючи інтерфейси для настільних комп'ютерів, мобільних пристроїв, годинників, столів і телевізорів.

З такою різноманітністю платформ Apple прийняла правильне рішення оптимізувати зовнішній вигляд усіх платформ. Переходячи від однієї платформи до іншої, немає великої різниці з точки зору зовнішнього вигляду та відчуття, і це навмисно.

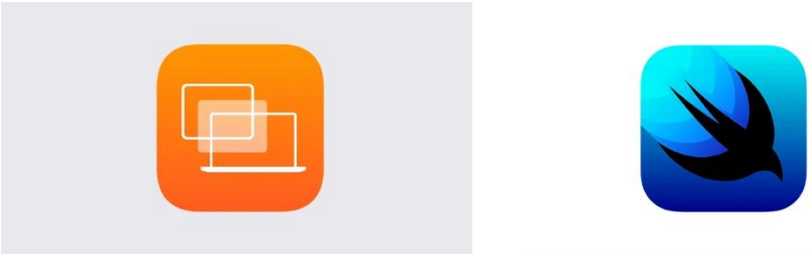

Рис.2.15. Програми для навчання розробки під девайси Apple

З точки зору програмного забезпечення, Apple зробила iCloud основою екосистеми. Більшість поширених програм Apple підключено до iCloud, тому компанія дала доступ синхронізування однакових даних між пристроями. Приклад роботи синхронізації показано на рис.2.16.

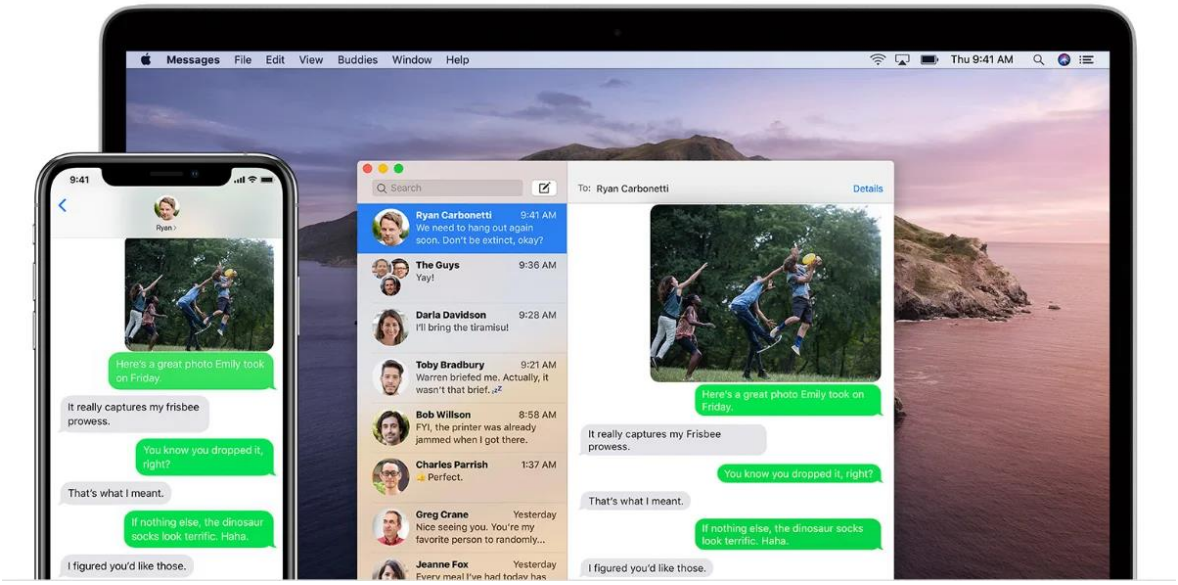

Рис. 2.16. Приклад роботи синхронізації

Універсальне керування в macOS Monterey. Максимально чудова інтеграція iPadOS і macOS(Рис. 2.17.)

Крім того, контролюючи апаратне забезпечення, Apple робить ще один крок далі у взаємодії з кількома пристроями. Він також може виявити, що ваші пристрої знаходяться ближче один до одного, і діяти належним чином.

Такі функції, як Universal Control (з'являється в macOS Monterey на підтримуваних пристроях), дозволяють вам керувати сусідніми комп'ютерами Mac і iPad, якими володієте.

Він розпізнає не тільки ваші пристрої, але й пристрої ваших друзів. Люди у вашому списку контактів, їхні пристрої можуть спілкуватися з вашими пристроями, як-от обмін паролями Wi-Fi, AirDrop і новіша функція SharePlay, яка дозволяє кожному ділитися програмами на кількох пристроях.

Рис. 2.17. Універсальне керування в macOS Monterey та інтеграція iPadOS і macOS

iMessage є ще одним найкращим прикладом ідеальної роботи екосистеми Apple. Усі ваші повідомлення доступні на ваших пристроях, а інші люди, які перебувають у тій самій екосистемі, мають дуже багатий набір функцій, як-от відеодзвінки, повідомлення, які надходять у вигляді тексту, аудіо чи відео, і навіть перегляд того самого за допомогою SharePlay.

Apple SharePlay(Рис.2.18.): приклад використання потужності екосистеми. Спілкування з друзями на iPhone, і коли ви отримаєте спільний відеозапис від друзів у вашій групі, дає зможливість відтворити відео на своєму Apple TV.

Рис. 2.18. Приклад роботи Apple SharePlay

Однією з унікальних особливостей екосистеми Apple є унікальний робочий процес для виконання завдань, будь то настільки легкий, як перенесення файлів з одного дизайну в інший, до чогось такого складного, як створення художнього фільму від зйомки до редагування, а потім прискорення таких завдань за допомогою спеціально створених обладнання.

Створення та редагування відео є складним і вимогливим завданням. Постає питання про зйомку відеоматеріалів (або кількох), а потім перенесення їх на комп'ютер для редагування та експортування для споживання.

Проблеми для виконання такого завдання полягають у тому, що необхідно виконати величезний обсяг зберігання та обробки даних. І якщо у вас є корекція кольору або загальні ефекти для відео (що робить кожен професіонал). Тут Apple визначила такі завдання та розробила рішення для таких завдань і використовуючи апаратні прискорювачі для покращення досвіду.

Apple створила апаратні прискорювачі, які роблять її неперевершеною для таких завдань. Якщо ваш робочий процес ідеально вписується в екосистему, як-от редагування відео ProRes у Final Cut Pro, жодне інше рішення не наближається(Рис. 2.19.)

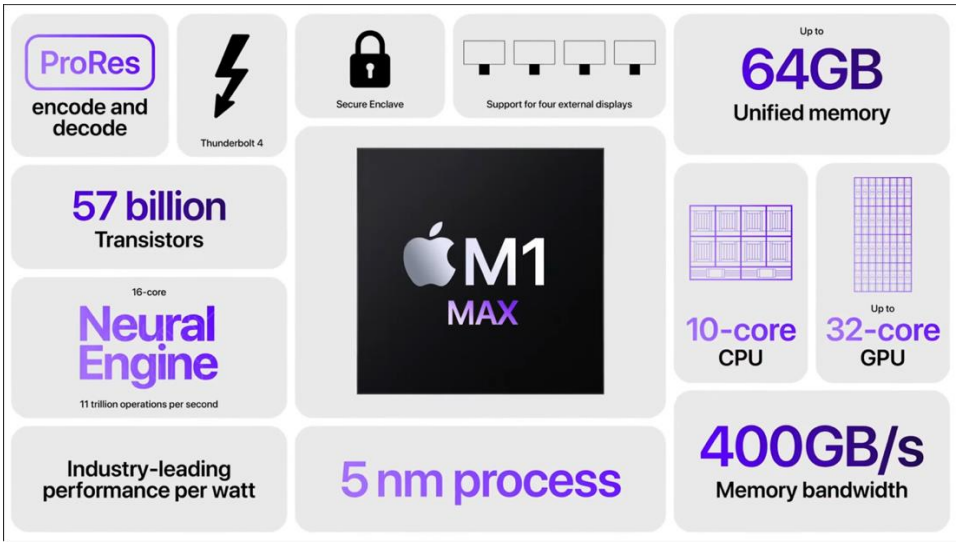

Рис. 2.19. Технічні характеристики Apple M1 Max

Сьогодні можливо записувати відео в ProRes зі швидкістю 1 гігабайт на секунду для відео 4K зі швидкістю 60 кадрів на секунду на iPhone 13 Pro. На щастя, чіп A15 в iPhone 13 Pro має прискорювач ProRes, щоб справлятися з таким навантаженням.

Потім ви використовуєте AirDrop, щоб перенести відзнятий матеріал на MacBook Pro з M1 Pro Max SOC. Цей SOC має спеціальний медіа-процесор для обробки потоків кадрів 4K60 ProRes, а Apple має Final Cut Pro, який оптимізований для апаратного забезпечення.

Ця досконала екосистема, створена Apple, гарантує, щокористувач отримає оптимальний досвід редагування високоякісного відеоматеріалу ProRes, який іншим машинам важко впоратися через неоптимізовану природу.

2.6. Структура та розробка застосунку з використанням Vision.api

Apple випустила Core ML та Vision.api у iOS 11. Core ML дає розробникам можливість використовувати моделі машинного навчання у своїх додатках. Це дозволяє створювати інтелектуальні фічі на пристрої, наприклад, визначення будь-яких об'єктів.[15]

Core ML 3, починаючи з iOS 13, було додано on-device навчання і нові способи персоналізації виведення.

Машинне навчання або ML (machine learning) – це підмножина II, яке навчає машини виконувати певні завдання. Наприклад, можливо використовувати ML для навчання машини розпізнавати кішку на зображенні або перекладати текст з однієї мови на іншу.

Apple визначає модель як ”результат застосування алгоритму машинного навчання до набору навчальних даних”. Слід уявити модель як функцію, яка приймає вхідні дані, виконує певну операцію якнайкраще з цими вхідними даними, таку як навчання, а потім виконує прогнозування та класифікацію. попередніх етапів ця функція здійснює відповідний вихідний сигнал. Приклад роботи ML показано на рис. 2.20.

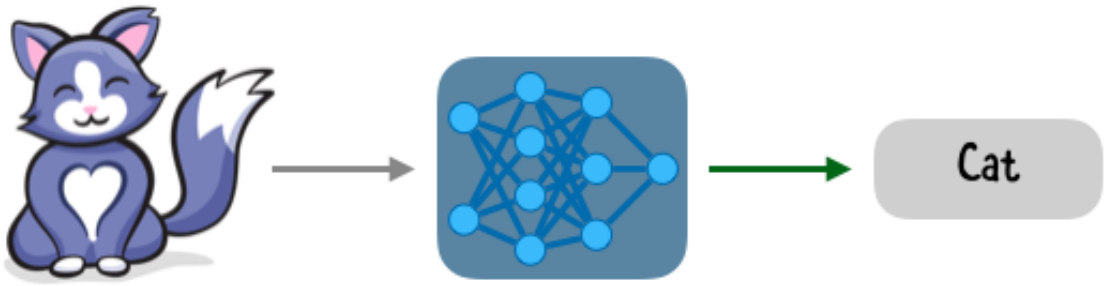

Рис. 2.20. Приклад роботи ML

За допомогою Core ML та Vision.api можливо точно налаштувати модель, що оновлюється на пристрої під час виконання. Це означає, що розробник може персоналізувати досвід використання кожного користувача.

On-device персоналізація - це ідея, що лежить в основі Face ID. Apple може надати загальну модель на пристрій, що розпізнає особу. Під час налаштування ID Face кожен користувач може точно налаштувати модель для розпізнавання саме свого обличчя.

Немає сенсу відправляти цю оновлену модель назад до Apple для інших користувачів. Це підкреслює перевагу конфіденційності, яку приносить on-device персоналізація.

Модель, що оновлюється - це базова модель ML, у якої є відповідне маркування.

Vision.api за допомогою моделі оновлюваного малюючого класифікатора розпізнає нові малюнки на основі k-Nearest Neighbors або k-NN.

k-NN алгоритм припускає, що схожі між собою об'єкти знаходяться близько один до одного.

Відбувається порівняння векторів ознак. Вектор ознаки або властивості містить важливу інформацію, яка описує характеристики об'єкта. Прикладом векторної ознаки є RGB колір, представлений R, G і B окремо.

Порівняння відстані між векторами ознак – це простий спосіб побачити, чи схожі два об'єкти. k-NN класифікує вхідні дані, використовуючи їх k-nearest neighbors (тобто найближчих сусідів).

Моделі k-NN прості та швидкі. Не потрібно багато прикладів, щоб навчити їх. Однак продуктивність може сповільнитися, якщо для аналізу буде надано безліч даних.

k-NN – це один з типів моделей, які Core ML та Vision.api підтримує для розробки технології.

Сам по собі Vision.api це збірка багатоьох менших api в собі. (Рис. 2.21.)

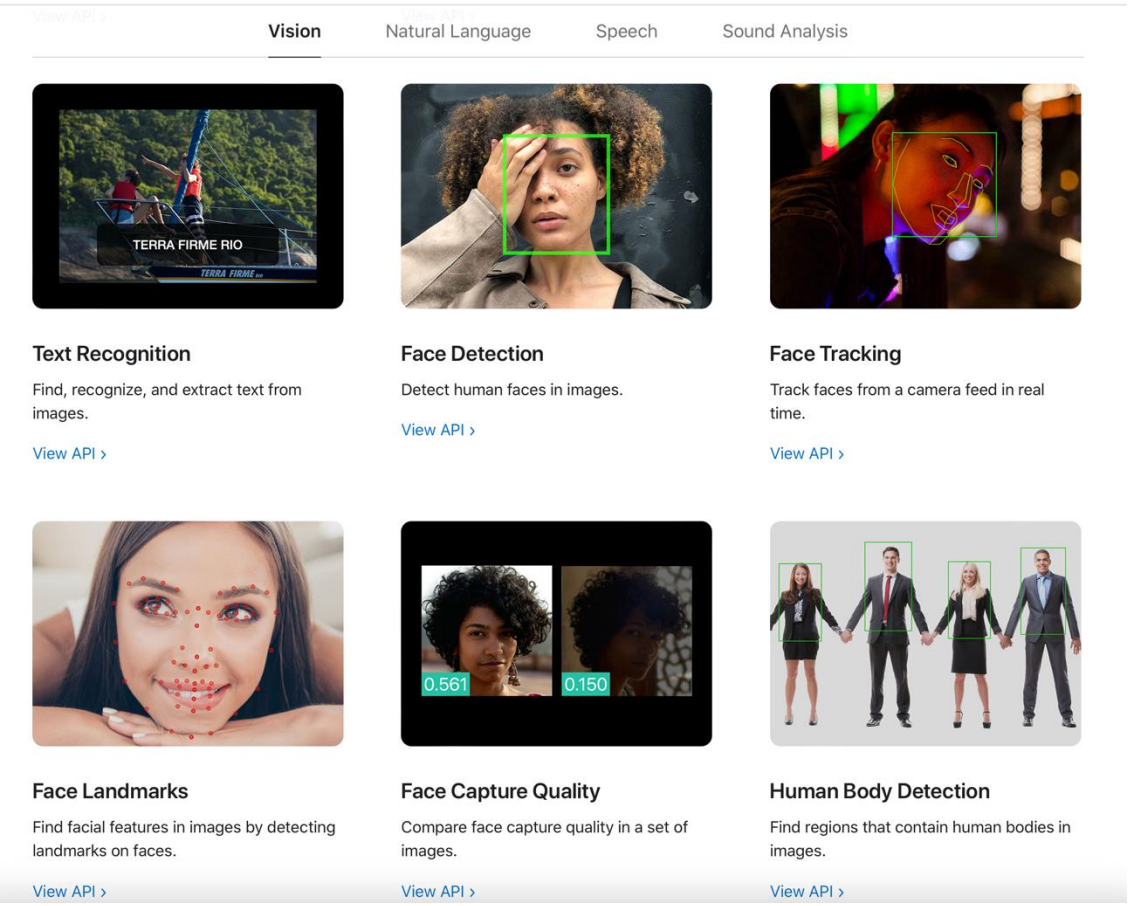

Рис. 2.21. Api що входять у Vision

Для розробки було використано лише декілька з них, такі як (Face Detection, Face Recognition та Face Tracking)

VNDetectFaceRectanglesRequest(Face Detection) – Запит, який знаходить обличчя на зображенні.

Огляд: клас VNDetectFaceRectanglesRequest : VNImageBasedRequest

Цей запит повертає обличчя у вигляді прямокутних обмежувальних рамок із зазначенням початку та розміру.

Результати: вар. результатів: [VNFaceObservation]?

Результати запиту на розпізнавання обличчя: лас VNFaceObservation

Інформація про обличчя або риси обличчя, яку виявляє запит на аналіз зображення.

VNDetectFaceRectanglesRequestRevision1: Int

Структура Vision може виявляти та відстежувати прямокутники, обличчя та інші помітні об’єкти в послідовності зображень.

Щоб візуалізувати геометрію спостережуваних рис обличчя, код малює малює маску із трикутників із використання датчиків FaceID смартфону.

Зразок програми застосовує алгоритми комп’ютерного зору, щоб знайти обличчя на наданому зображенні. Коли він знаходить обличчя, він намагається відстежити це обличчя в наступних кадрах відео.

Спочатку створюється новий сеанс AVCaptureSession для відображення захоплення відео.

Далі створюється послідовна черга розсилки. Ця черга гарантує, що відеокадри, отримані асинхронно через методи зворотного виклику делегату, доставляються в порядку. Встановлюється сеанс захоплення з відео AVMediaType його пристрій і роздільна здатність.

Коли застосунок отримає спостереження від розпізнавання обличчя обробника запиту зображення, він вноситься в обробник запиту послідовності:

try self.sequenceRequestHandler.perform(requests, on: pixelBuffer, orientation: exifOrientation)

Якщо детектор не знайшов обличчя, створюється обробник запиту зображення для виявлення обличчя. Коли це виявлення вдасться та буде спостереження за обличчям, відстежування займеться скрипт VNTrackObjectRequest.

Фреймворк Vision може виявляти прямокутники, обличчя, текст і штрих-коди в будь-якій орієнтації.

Vision обробляє запити на основі нерухомих зображень за допомогою VNImageRequestHandler і передбачає, що зображення орієнтовані вертикально, тому і передає зображення з урахуванням орієнтації. Об’єкти CGImage, CIImage та CVPixelBuffer не мають орієнтації, тому надаються як частина ініціалізатора.[16]

Ініціалізація VNImageRequestHandler із даних зображення доступна в таких форматах:

CGImage: формат зображення Core Graphics, який можна отримати з будь-якого UIImage через його допоміжний метод cgImage через ініціалізатор за допомогою CGImagePropertyOrientation.

CIImage: формат Core Image, який найкраще використовувати, якщо у

конвеєрі обробки зображень уже є Core Image. Об’єкти CIImage не містять орієнтації, тому вказуються і ініціалізаторі init(ciImage:orientation:options:).

CVPixelBuffer: формат зображення Core Video для даних із живої стрічки та фільмів. Об’єкти CVPixelBuffer не містять орієнтації, тому вказуються в ініціалізаторі init(cvPixelBuffer:orientation:options:).

NSData: дані зображення, стиснені або збережені в пам’яті, які ви можете отримати через мережеве з’єднання. Наприклад, до цієї категорії потрапляють фотографії, завантажені з веб-сайту чи хмари.

NSURL: URL-шлях до зображення на диску.

Система зору може не виявити належним чином перевернуті або перевернуті елементи, якщо прийме неправильну орієнтацію. Фотографії, вибрані в інструменті вибору зображень зразка, містять інформацію про орієнтацію. Отримати доступ до цих даних можливо через властивість UIImage imageOrientation.

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі були описані функціональні та нефункціональні вимоги до застосунку.

Було описано принцип дії, переваги та недоліки архітектурних рішень на основі Vision.api.

Програма була розроблена з використанням сучасного стеку технологій, що використовується при створенні мобільних додатків для системи IOS з використанням сучасного середовища розробки Xcode.

За допомогою розгорнутих у другому розділі шаблонів проектування, архітектурних рішень, технологій та принципів дизайну Apple Human Guidelines було розроблено стабільний, швидкий та зручний мобільний застосунок. Також слід зазначити, що програма відповідає як описаним, так і загальноприйнятим вимогам до мобільних застосунків.

РОЗДІЛ 3

РОЗРОБКА ТА ТЕСТУВАННЯ ЗАСТОСУНКУ

3.1. Принцип роботи мобільного застосунку

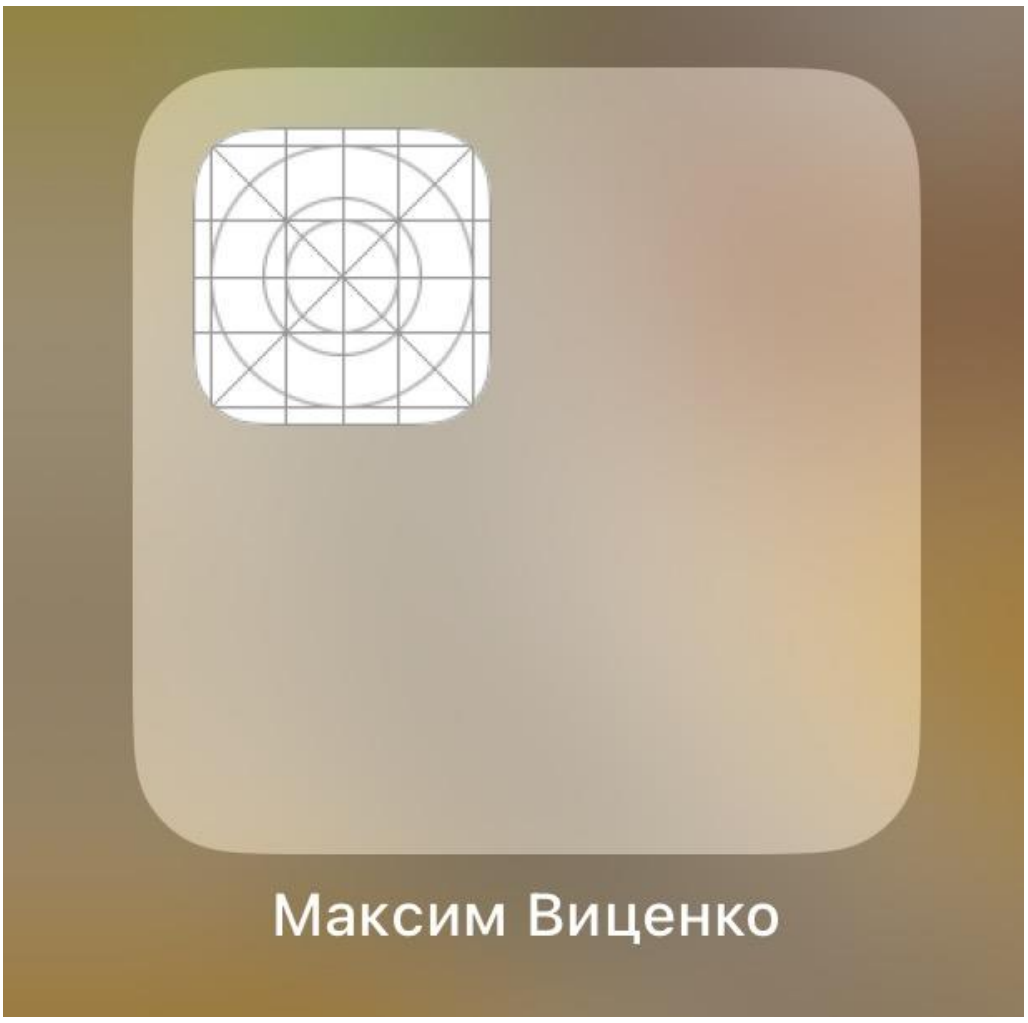

Рис. 3.1. Іконка застосунку

Коли користувач перший раз відкриває застосунок, перше, що він бачить – це головний екран на якому він може вибрати функцію для перевірки системи розпізнавання обличчя(рис. 3.2).

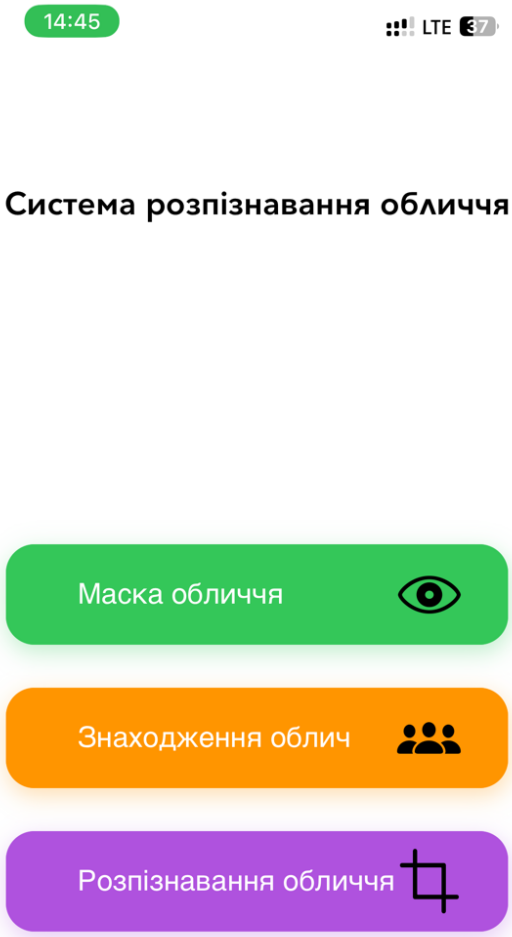

Рис. 3.2. Головний. Екран із вибором функцій

Після того як користувач обрав функцію натиснувши на відповідну кнопку, його перенаправляє у вікно відтворення функції.

Перша функція затосунку – це *«Маска обличчя»,* це інструмент що візуально демонструє користувачу як саме працює застосунок а саме зчитування даних з обличчя для розпізнавання методом побудови або маски на обличчі відповідних зон що в подальшому зчитуються. Екран демонстрції роботи «Маски обличчя» зображено на рис. 3.3.

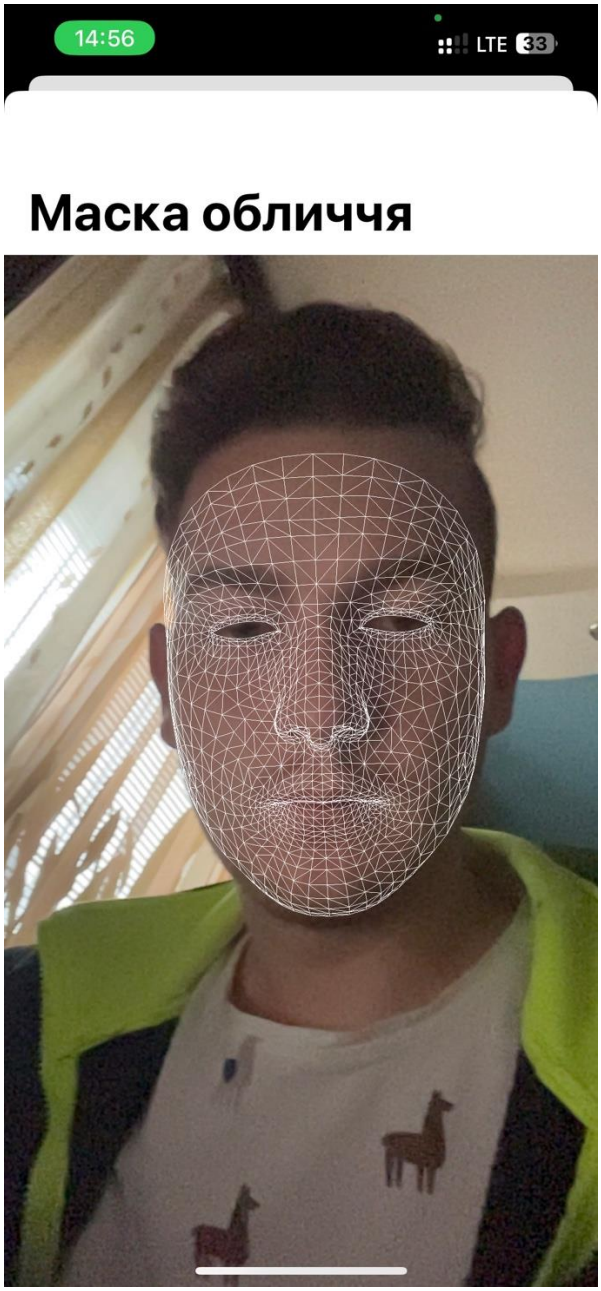

Рис. 3.3. Екран демонстрації роботи «Маски обличчя»

Створення екземпляра перегляду сцени

Прикоад використання ARKit створивши екземпляр ARSCNView, який автоматично рендерить живе відео з камери пристрою як фон сцени. Він також автоматично переміщує камеру SceneKit відповідно до реального руху пристрою, що означає, що нам не потрібен якір, щоб відстежувати положення об’єктів, які ми додаємо до сцени.

Оскільки ця функція використовує камеру TrueDepth, ми можемо використовувати лише передню камеру.

Вид сцени повинен мати межі всього екрана для сеансу камери:

```
let sceneView = ARSCNView(frame: UIScreen.main.bounds)
```

Запуск сеансу ARFaceTrackingConfiguration, який є сеансом AR, виявляє обличчя користувача (якщо його видно на зображенні передньої камери) і додає до свого списку прив’язок об’єкт ARFaceAnchor, що представляє обличчя.

Налаштування функції ViewDidLoad():

```
override func
viewDidLoad()
{
                    super.viewDidLoad()
                    self.view.addSubview(sceneView)
                    sceneView.delegate = self
                    guard ARFaceTrackingConfiguration.isSupported else { return }
                    let configuration = ARFaceTrackingConfiguration()
                    configuration.isLightEstimationEnabled = true
                    sceneView.session.run(configuration, options: [.resetTracking, .removeExistingAnchors])
                    setupTabBar()
                }
```

Приклад використання розширення для встановлення делегату сцени та візуалізації ARSCNFaceGeometry. Також було додано функцію, яка оновлює геометрію обличчя в реальному часі:

```
extension
FaceMaskViewController:
ARSCNViewDelegate {
```

```
    func renderer(_ renderer: SCNSceneRenderer, nodeFor anchor: ARAnchor) -
> SCNNode? {

        guard let device = sceneView.device else {
            return nil
        }

        let faceGeometry = ARSCNFaceGeometry(device: device)
        let node = SCNNode(geometry: faceGeometry)
        node.geometry?.firstMaterial?.fillMode = .lines

        return node
    }

    func renderer(_ renderer: SCNSceneRenderer, didUpdate node: SCNNode,
for anchor: ARAnchor) {

        guard let faceAnchor = anchor as? ARFaceAnchor,
            let faceGeometry = node.geometry as? ARSCNFaceGeometry else {
                return
        }
        faceGeometry.update(from: faceAnchor.geometry)
    }
}
```

Геометрія обличчя може бути надзвичайно корисною, якщо потрібно розмістити об’єкти на певній ділянці обличчя — наприклад нанести макіяж на ніс. У цьому випадку можливо знайти потрібні вершини, використовуючи об’єкт ARSCNFaceGeometry, а потім створити екземпляр об’єкта вузла.

Наступна послуга «*Знаходження облич»,* її мета розпізнати лиця на об’єкті куди користувач направить камеру, та порахувати кількість людей(їхніх облич) що знайшов застосунок, екран демонстрації роботи «Знаходження облич» зображено на рис. 3.4.

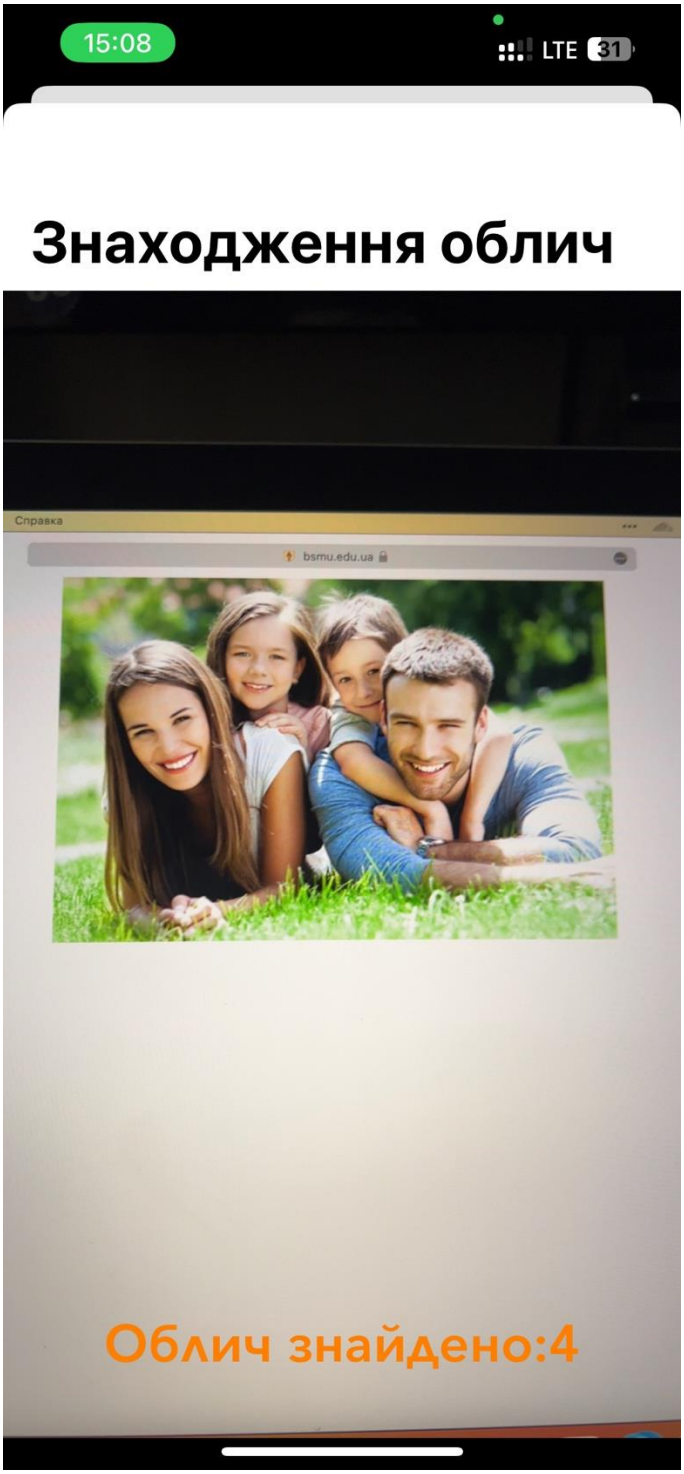

Рис. 3.4. Екран демонстрації роботи «Знаходження облич»

Для налаштування сеансу захоплення облич використовувався AVCaptureSession що дало змогу додати допис попереднього перегляду:

```
fileprivate func
setupCamera() {
                    let captureSession = AVCaptureSession()
                    captureSession.sessionPreset = .high
```

```
        guard let captureDevice = AVCaptureDevice.default(for: .video) else { return }
        guard let input = try? AVCaptureDeviceInput(device: captureDevice) else { return }
        captureSession.addInput(input)

        captureSession.startRunning()

        let previewLayer = AVCaptureVideoPreviewLayer(session: captureSession)
        view.layer.addSublayer(previewLayer)
        previewLayer.frame = view.frame

        let dataOutput = AVCaptureVideoDataOutput()
        dataOutput.setSampleBufferDelegate(self, queue: DispatchQueue(label:
    "videoQueue"))
        captureSession.addOutput(dataOutput)
    }
```

Метод виявлення обличчя є частиною фреймворку Vision.api, який є дуже швидким і досить точним. Метод VNDetectFaceRectangleRequest() повертає масив обмежувальних рамок для виявлених облич.

Щоб отримати кількість облич, створено скрипт який рахує кількість елементів у масиві, а потім відповідно оновлює мітку яка і відображає дані про кількість облич на екрані:

```
func captureOutput(_
output:
AVCaptureOutput,
didOutput
sampleBuffer:
CMSampleBuffer, from
connection:
AVCaptureConnection)
{

        guard let pixelBuffer: CVPixelBuffer =
    CMSampleBufferGetImageBuffer(sampleBuffer) else { return }
        let request = VNDetectFaceRectanglesRequest { (req, err) in

            if let err = err {
                print("Failed to detect faces:", err)
                return
```

```
            }
            DispatchQueue.main.async {
                if let results = req.results {
                    self.numberOfFaces.text = "\(results.count) face(s)"
                }
            }
        }

        DispatchQueue.global(qos: .userInteractive).async {
            let handler = VNImageRequestHandler(cvPixelBuffer: pixelBuffer, options: [:])
            do {
                try handler.perform([request])
            } catch let reqErr {
                print("Failed to perform request:", reqErr)
            }
        }
    }
```

Щоб оновити мітку, це було зроблено в головному потоці; інакше сеанс вийде з ладу.

У результаті API повертає масив об’єктів, який містить усі грані як прямокутні рамки.

Остання функція застосунку це *«Розпізнавання обличчя»,* з назви зрозуміло і її функціонал, застосунок креслить квадрат який позначає область лиця та над квадратом текст імені, порядкового номеру або нікнейму людини що сканують обличчя. Приклад роботи екрану демонстрації роботи «Розпізнавання обличчя» зображено на рис. 3.5.

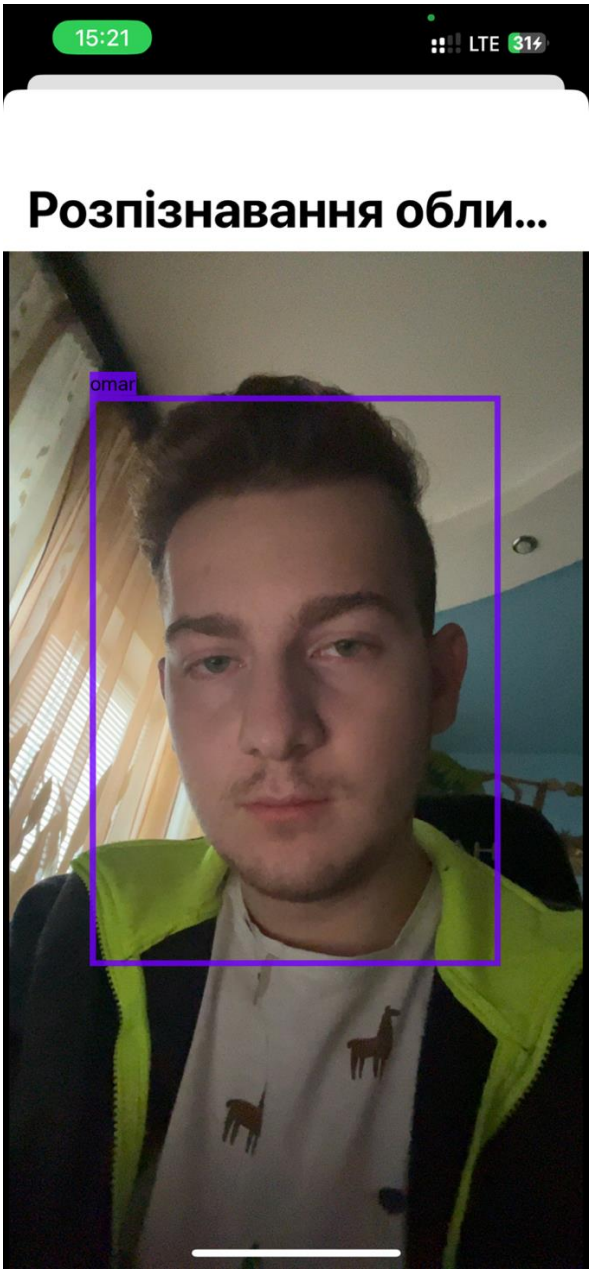

Рис. 3.5. Екран демонстрації роботи «Розпізнавання обличчя»

Підготовка даних

Тут знадобиться кілька сотень зображень, щоб побудувати точну модель, яка передбачить обмежувальну рамку.

Щоб навчити модель, потрібно анотувати кожне зображення , тому для частини анотацій використувався репозиторій GitHub Sebastian G. Perez’s.

Це проста програма Flask (Рис. 3.6.), яка обробляє анотації зображень і створює файл .csv, відформатований у спосіб, який можна легко використовувати з xCode для навчання моделі.

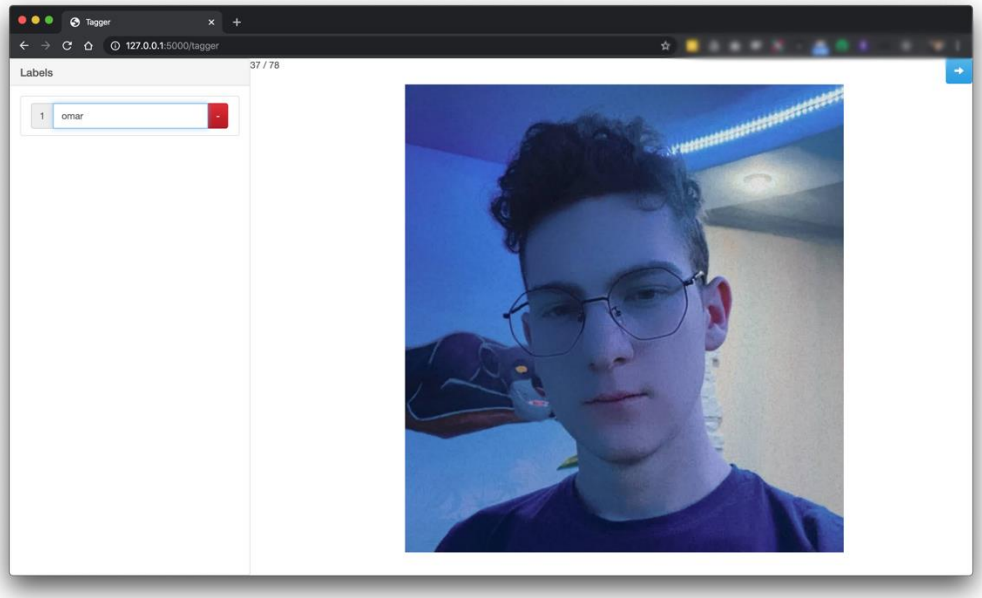

Рис. 3.6. Екран демонстрації роботи «Flask»

Для налаштування сеансу захоплення розроблено скрипт що використовує попередній перегляд з камери в прямому ефірі, щоб зробити висновки для кожного кадру та намалювати обмежувальну рамку.

```
var
videoCapture:
VideoCapture!
                    let semaphore = DispatchSemaphore(value: 1)

                    let videoPreview: UIView = {
                      let view = UIView()
                       view.translatesAutoresizingMaskIntoConstraints = false
                       return view
                    }()
```

```
    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        self.videoCapture.start()
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)
        self.videoCapture.stop()
    }

    // MARK: - SetUp Camera preview
    func setUpCamera() {
        videoCapture = VideoCapture()
        videoCapture.delegate = self
        videoCapture.fps = 30
        videoCapture.setUp(sessionPreset: .vga640x480) { success in

            if success {
                if let previewLayer = self.videoCapture.previewLayer {
                    self.videoPreview.layer.addSublayer(previewLayer)
                    self.resizePreviewLayer()
                }
                self.videoCapture.start()
            }
        }
    }

extension ViewController: VideoCaptureDelegate {
    func videoCapture(_ capture: VideoCapture, didCaptureVideoFrame pixelBuffer:
CVPixelBuffer?, timestamp: CMTime) {
        if !self.isInferencing, let pixelBuffer = pixelBuffer {
            self.isInferencing = true
            self.predictUsingVision(pixelBuffer: pixelBuffer)
        }
    }
}
```

Простими словами опис коду має такий вигляд:

1)Створює екземпляр UIView для розміщення VideoCapture.(Рис. 3.7, 3.8.)

2)Запускає VideoCapture у функції перегляду viewWillAppear

3)Зупиняє VideoCapture у функції перегляду viewWillDisappear

4)Налаштовує для VideoCapture кількість кадрів за секунду, а також якість відео. (Для меншої ресурсоємкості використовувався тип VGA 640x480, щоб отримати постійну частоту кадрів за секунду (FPS)).

5)Налаштовує делегат VideoCapture на відповідність AVCaptureVideoDataOutputSampleBufferDelegate.

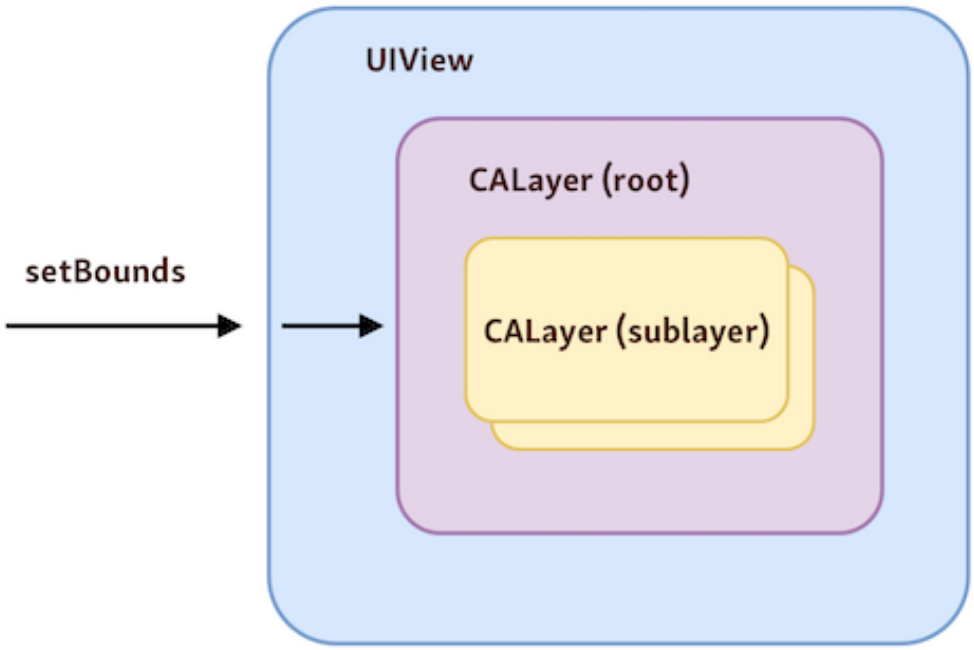

Рис. 3.7. Приклад створення екземпляру UIView

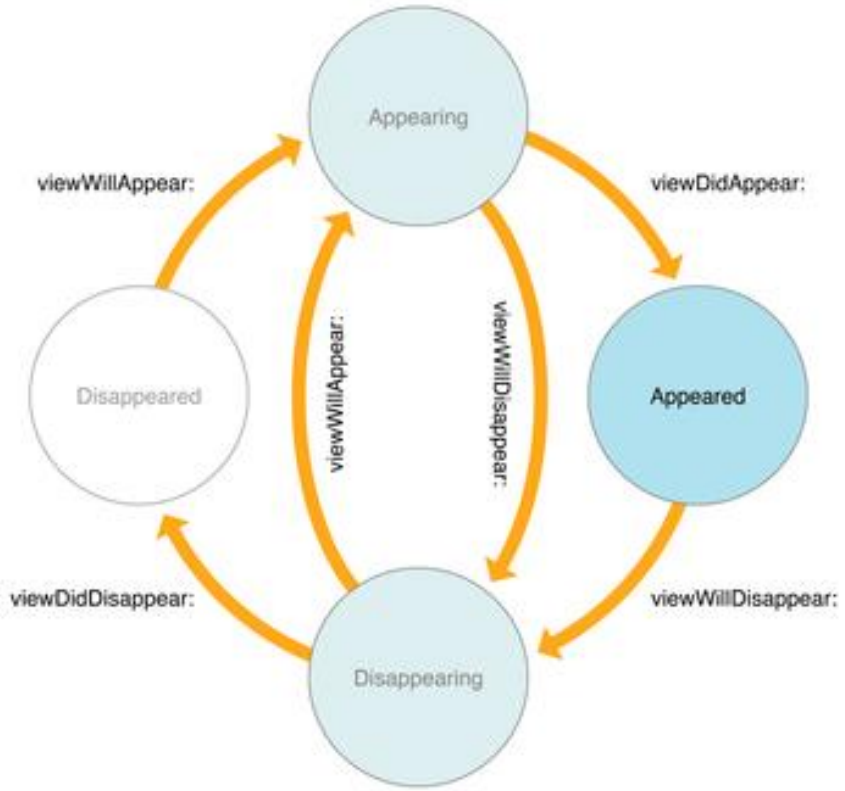

Рис. 3.8 Приклад розміщення VideoCapture

Після створення та налаштування попереднього перегляду, розроблено скрипт, який описує екземпляр моделі Core ML(Рис.3.9) і починає прогнозування кадрів:

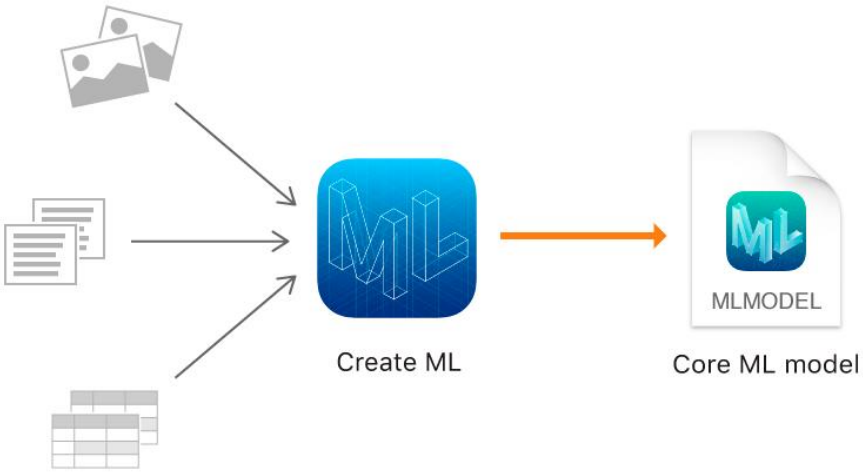

Рис. 3.9 Екземпляр моделі Core ML

```
func
setUpModel()
{
                    if let visionModel = try? VNCoreMLModel(for: model_face_omar_turi().model) {
                        self.visionModel = visionModel
                        request = VNCoreMLRequest(model: visionModel, completionHandler:
                visionRequestDidComplete)
                        request?.imageCropAndScaleOption = .scaleFill
                    } else {
                        fatalError("fail to create vision model")
                    }
                }
```

Модель тут називається model_face_omar_turi. Але це лише сеанс Vision — я кий все одно запускає модель, передавши їй кадри. Очікується, що модель повертатиме об’єкт MultiArray, який інкапсулює обмежувальну рамку. Останніми кроками будуть передбачення, аналіз об’єкта та малювання прямокутника навколо обличчя.

```
extension
ObjectDetectionViewController:
VideoCaptureDelegate {
                                    func videoCapture(_ capture: VideoCapture,
                                didCaptureVideoFrame pixelBuffer: CVPixelBuffer?, timestamp:
                                CMTime) {
                                        // the captured image from camera is contained on
                                pixelBuffer
                                        if !self.isInferencing, let pixelBuffer = pixelBuffer {
                                            self.isInferencing = true
                                            // predict!
                                            self.predictUsingVision(pixelBuffer: pixelBuffer)
                                        }
                                    }
                                }


                                extension ObjectDetectionViewController {
                                    func predictUsingVision(pixelBuffer: CVPixelBuffer) {
                                        guard let request = request else { fatalError() }
                                        self.semaphore.wait()
```

```
        let handler = VNImageRequestHandler(cvPixelBuffer:
pixelBuffer)
        try? handler.perform([request])
    }

    // MARK: - Post-processing
    func visionRequestDidComplete(request: VNRequest, error:
Error?) {
        if let predictions = request.results as?
[VNRecognizedObjectObservation] {
            DispatchQueue.main.async {
                self.BoundingBoxView.predictedObjects = predictions
                self.isInferencing = false
            }
        } else {

            self.isInferencing = false
        }
        self.semaphore.signal()
    }
}
```

Після чого застосунок спрацьовує та коректно розпізнає об’єкт на камері що в свою чергу доводить коректність роботи програми.

3.2. Розробка UI інтерфесу та отримання даних з камери для зчитування при роботі застосунку

Для побудови інтерфесу саме кодом було використано UIKit.API, так як це офійне API від компанії Apple, саме воно точно відповідає всім гайдлайнам компанії, та надає зручні інструменти для створення інтерфейсу.

Було створено Label та декілька кнопок для швидкого та зручного доступу дло основних функціх застосунку(Рис. 3.10, 3.11.), за дизайн

відповідає код:

```
import UIKit

class ViewController: UIViewController {

    let name: UILabel = {
       let text = UILabel()
        text.translatesAutoresizingMaskIntoConstraints = false
        text.font = UIFont(name: "Avenir-Heavy", size: 22)
        text.text = "Система розпізнавання обличчя"
        text.textColor = .label
        return text

    }()


    let faceMask: BtnPleinLarge = {
        let button = BtnPleinLarge()
        button.translatesAutoresizingMaskIntoConstraints = false
        button.addTarget(self, action: #selector(buttonToFaceMask(_:)), for:
.touchUpInside)
        button.setTitle("Маска обличчя", for: .normal)
        let icon = UIImage(systemName: "eye")?.resized(newSize: CGSize(width: 50,
height: 30))
        button.addRightImage(image: icon!, offset: 30)
        button.backgroundColor = .systemGreen
        button.layer.borderColor = UIColor.systemGreen.cgColor
        button.layer.shadowOpacity = 0.3
        button.layer.shadowColor = UIColor.systemGreen.cgColor

        return button
    }()

    let faceDetection: BtnPleinLarge = {
        let button = BtnPleinLarge()
        button.translatesAutoresizingMaskIntoConstraints = false
        button.addTarget(self, action: #selector(buttonToFaceDetection(_:)), for:
.touchUpInside)
        button.setTitle("Знаходження облич", for: .normal)
        let icon = UIImage(systemName: "person.3.fill")?.resized(newSize:
CGSize(width: 50, height: 25))
        button.addRightImage(image: icon!, offset: 30)
        button.backgroundColor = .systemOrange
        button.layer.borderColor = UIColor.systemOrange.cgColor
        button.layer.shadowOpacity = 0.3
        button.layer.shadowColor = UIColor.systemOrange.cgColor

        return button
```

```
}()
```

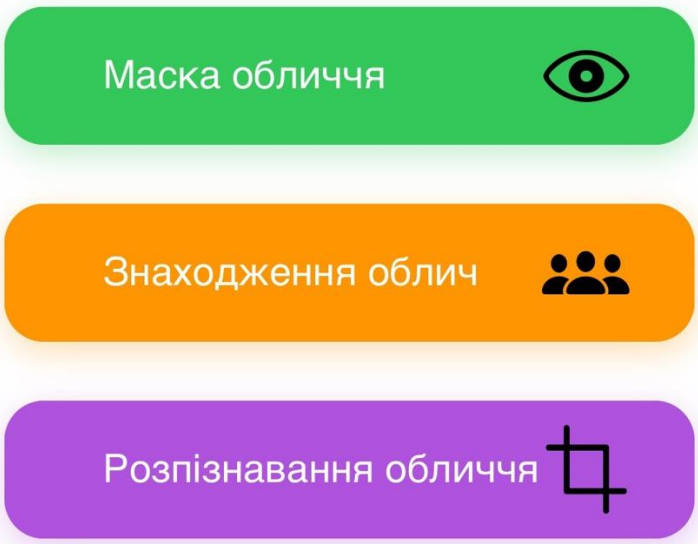

Рис. 3.10. Візуальний вигляд кнопок у застосунку

```
    let objectDetection: BtnPleinLarge = {
        let button = BtnPleinLarge()
        button.translatesAutoresizingMaskIntoConstraints = false
        button.addTarget(self, action: #selector(buttonToObjectDetection(_:)), for:
.touchUpInside)
        button.setTitle("Розпізнавання обличчя", for: .normal)
        let icon = UIImage(systemName: "crop")?.resized(newSize: CGSize(width: 50,
height: 50))
        button.addRightImage(image: icon!, offset: 30)
        button.backgroundColor = .systemPurple
        button.layer.borderColor = UIColor.systemPurple.cgColor
        button.layer.shadowOpacity = 0.3
        button.layer.shadowColor = UIColor.systemPurple.cgColor

        return button
    }()


    override func viewDidLoad() {
        super.viewDidLoad()
        view.backgroundColor = .systemBackground
        setupTabBar()
        setupLabel()
        setupButtons()
    }
```

Рис. 3.11. Візуальний вигляд Label у застосунку

В свою чергу за фунціональну можливість кнопок працювати та відповідати відповідним функціям застосунку відповідає код(Рис. 3.12, 3.13.):

```
    func setupTabBar() {
        navigationController?.navigationBar.prefersLargeTitles = true
        self.navigationItem.title = "Face AI"
        if #available(iOS 13.0, *) {
            self.navigationController?.navigationBar.barTintColor = .systemBackground
             navigationController?.navigationBar.titleTextAttributes = [.foregroundColor :
UIColor.label]
        } else {
            self.navigationController?.navigationBar.barTintColor = .lightText
            navigationController?.navigationBar.titleTextAttributes = [.foregroundColor :
UIColor.black]
        }
        self.navigationController?.navigationBar.isHidden = false
        self.setNeedsStatusBarAppearanceUpdate()
        self.navigationItem.largeTitleDisplayMode = .automatic
        self.navigationController?.navigationBar.barStyle = .default
navigationController?.navigationBar.backgroundColor = .systemBackground
        } else {
            navigationController?.navigationBar.backgroundColor = .white
        }
        self.tabBarController?.tabBar.isHidden = false
    }
```

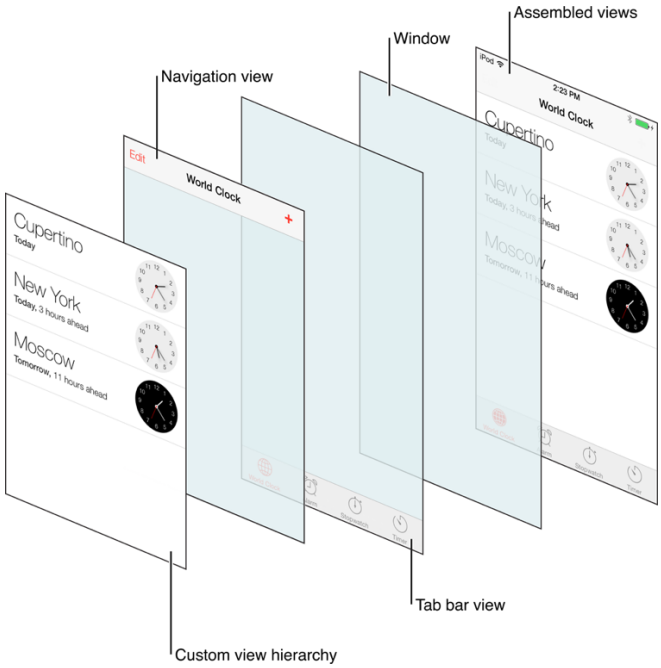

Рис. 3.12. Візуальний вигляд роботи navigationController

```
 private func setupLabel() {
        view.addSubview(name)
        name.topAnchor.constraint(equalTo: view.topAnchor, constant: 100).isActive =
true
        name.heightAnchor.constraint(equalToConstant: 100).isActive = true
        name.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive = true
        name.numberOfLines = 1
    }

    private func setupButtons() {

        view.addSubview(faceMask)
        view.addSubview(faceDetection)
        view.addSubview(objectDetection)

        faceMask.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive =
true
        faceMask.widthAnchor.constraint(equalToConstant: view.frame.width -
40).isActive = true
        faceMask.heightAnchor.constraint(equalToConstant: 70).isActive = true
        faceMask.centerYAnchor.constraint(equalTo: view.centerYAnchor).isActive =
true

        faceDetection.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive
= true
        faceDetection.widthAnchor.constraint(equalToConstant: view.frame.width -
40).isActive = true
        faceDetection.heightAnchor.constraint(equalToConstant: 70).isActive = true
```

```
        faceDetection.topAnchor.constraint(equalTo: faceMask.bottomAnchor,
constant: 30).isActive = true

        objectDetection.centerXAnchor.constraint(equalTo:
view.centerXAnchor).isActive = true
        objectDetection.widthAnchor.constraint(equalToConstant: view.frame.width -
40).isActive = true
        objectDetection.heightAnchor.constraint(equalToConstant: 70).isActive = true
        objectDetection.topAnchor.constraint(equalTo: faceDetection.bottomAnchor,
constant: 30).isActive = true
    }

    @objc func buttonToFaceMask(_ sender: BtnPleinLarge) {

        let controller = FaceMaskViewController()

        let navController = UINavigationController(rootViewController: controller)

        self.present(navController, animated: true, completion: nil)
    }

    @objc func buttonToFaceDetection(_ sender: BtnPleinLarge) {

      let controller = FaceDetectionViewController()

      let navController = UINavigationController(rootViewController: controller)

      self.present(navController, animated: true, completion: nil)
    }

    @objc func buttonToFaceClassification(_ sender: BtnPleinLarge) {

      let controller = FaceClassificationViewController()

      let navController = UINavigationController(rootViewController: controller)

      self.present(navController, animated: true, completion: nil)
    }

    @objc func buttonToObjectDetection(_ sender: BtnPleinLarge) {

      let controller = ObjectDetectionViewController()

      let navController = UINavigationController(rootViewController: controller)

      self.present(navController, animated: true, completion: nil)
    }

}
```

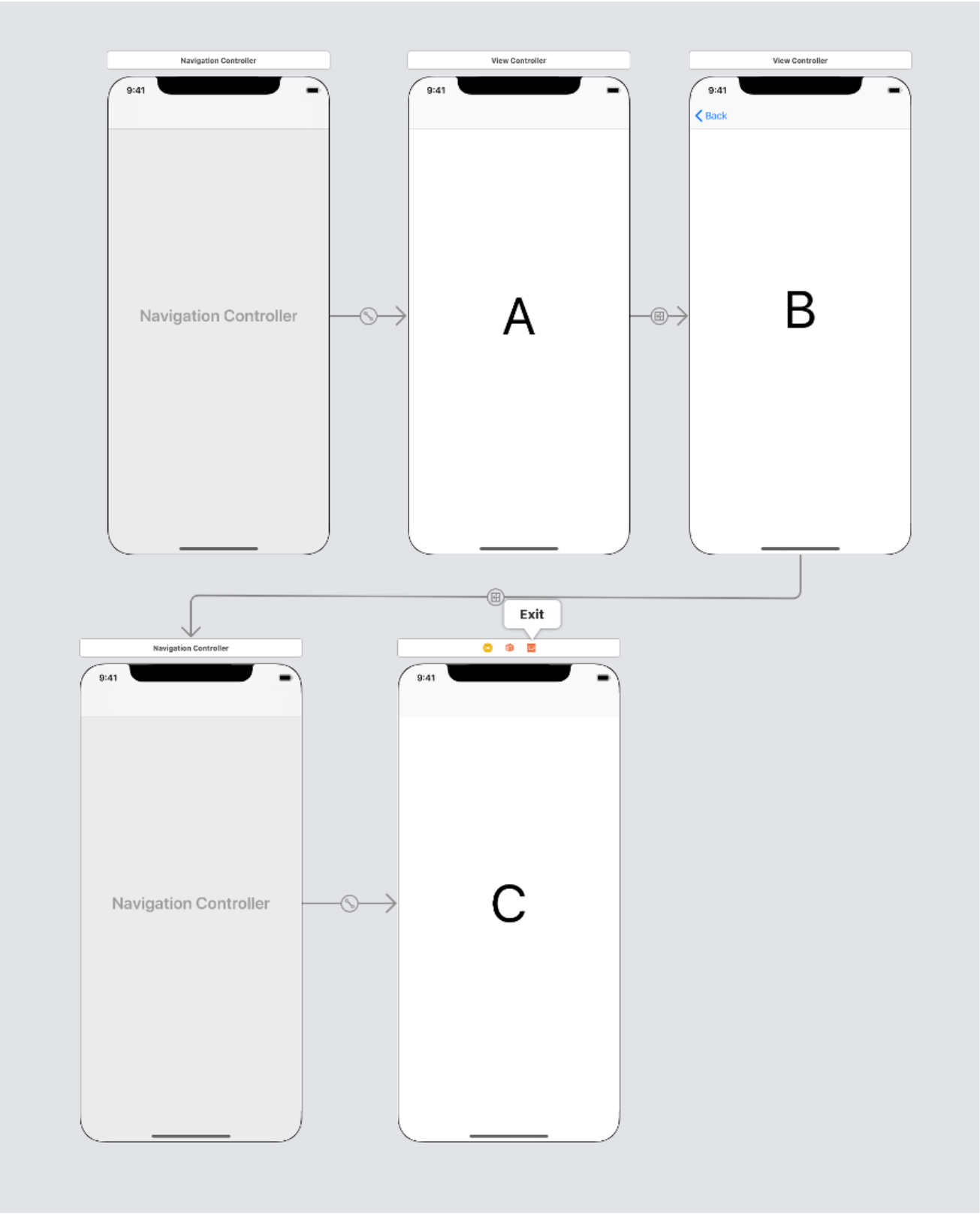

Рис. 3.13. Візуальний вигляд роботи viewController

Для захоплення та обробки відео з камери при обробці застосунком інформації було використано AVFoundation.API та CoreVideo.API. (Рис. 3.14, 3.15)

```
import UIKit
import AVFoundation
import CoreVideo

public protocol VideoCaptureDelegate: class {
   func videoCapture(_ capture: VideoCapture, didCaptureVideoFrame:
CVPixelBuffer?, timestamp: CMTime)
}

public class VideoCapture: NSObject {
   public var previewLayer: AVCaptureVideoPreviewLayer?
   public weak var delegate: VideoCaptureDelegate?
   public var fps = 15

   let captureSession = AVCaptureSession()
   let videoOutput = AVCaptureVideoDataOutput()
   let queue = DispatchQueue(label: "com.tucan9389.camera-queue")

   var lastTimestamp = CMTime()

   public func setUp(sessionPreset: AVCaptureSession.Preset = .vga640x480,
                     completion: @escaping (Bool) -> Void) {
      self.setUpCamera(sessionPreset: sessionPreset, completion: { success in
         completion(success)
      })
   }
```

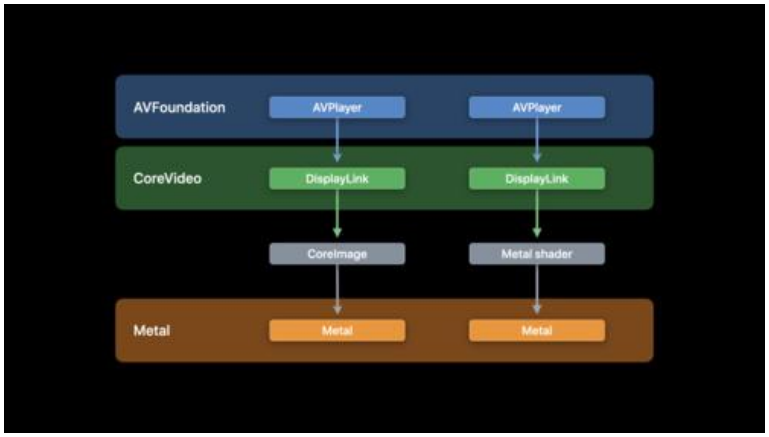

Рис. 3.14. Візуальний вигляд роботи CoreVideo.API

```
   func setUpCamera(sessionPreset: AVCaptureSession.Preset, completion:
@escaping (_ success: Bool) -> Void) {
```

```
        captureSession.beginConfiguration()
        captureSession.sessionPreset = sessionPreset

        guard let captureDevice = AVCaptureDevice.default(.builtInWideAngleCamera,
                                                          for: .video,
                                                          position: .front) else {

            print("Error: no video devices available")
            return
        }

        guard let videoInput = try? AVCaptureDeviceInput(device: captureDevice) else
{
            print("Error: could not create AVCaptureDeviceInput")
            return
        }

        if captureSession.canAddInput(videoInput) {
            captureSession.addInput(videoInput)
        }

        let previewLayer = AVCaptureVideoPreviewLayer(session: captureSession)
        previewLayer.videoGravity = AVLayerVideoGravity.resizeAspect
        previewLayer.connection?.videoOrientation = .portrait
        self.previewLayer = previewLayer

        let settings: [String : Any] = [
            kCVPixelBufferPixelFormatTypeKey as String: NSNumber(value:
kCVPixelFormatType_32BGRA),
        ]

        videoOutput.videoSettings = settings
        videoOutput.alwaysDiscardsLateVideoFrames = true
        videoOutput.setSampleBufferDelegate(self, queue: queue)
        if captureSession.canAddOutput(videoOutput) {
            captureSession.addOutput(videoOutput)
        }

        // We want the buffers to be in portrait orientation otherwise they are
        // rotated by 90 degrees. Need to set this _after_ addOutput()!
        videoOutput.connection(with: AVMediaType.video)?.videoOrientation = .portrait

        captureSession.commitConfiguration()

        let success = true
        completion(success)
    }

    public func start() {
```

```
        if !captureSession.isRunning {
            captureSession.startRunning()
        }
    }

    public func stop() {
        if captureSession.isRunning {
            captureSession.stopRunning()
        }
    }
}

extension VideoCapture: AVCaptureVideoDataOutputSampleBufferDelegate {
    public func captureOutput(_ output: AVCaptureOutput, didOutput sampleBuffer:
CMSampleBuffer, from connection: AVCaptureConnection) {
        // Because lowering the capture device's FPS looks ugly in the preview,
        // we capture at full speed but only call the delegate at its desired
        // framerate.
        let timestamp = CMSampleBufferGetPresentationTimeStamp(sampleBuffer)
        let deltaTime = timestamp - lastTimestamp
        if deltaTime >= CMTimeMake(value: 1, timescale: Int32(fps)) {
            lastTimestamp = timestamp
            let imageBuffer = CMSampleBufferGetImageBuffer(sampleBuffer)
            delegate?.videoCapture(self, didCaptureVideoFrame: imageBuffer,
timestamp: timestamp)
        }
    }

    public func captureOutput(_ output: AVCaptureOutput, didDrop sampleBuffer:
CMSampleBuffer, from connection: AVCaptureConnection) {
        //print("dropped frame")
    }
}
```

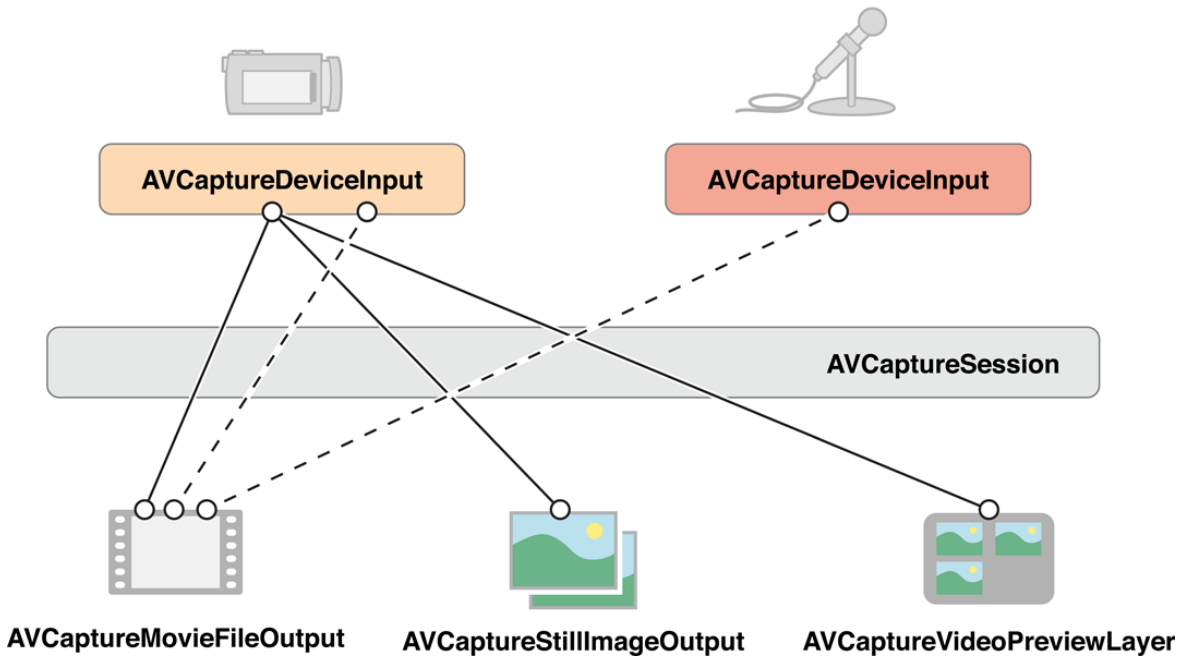

Рис. 3.15. Візуальний вигляд роботи AVFoundation.API

Для виконання програми слід її запустить на емуляторі або на пристрої. Вигляд емуляції пристрою що підключено (Рис. 3.16.):

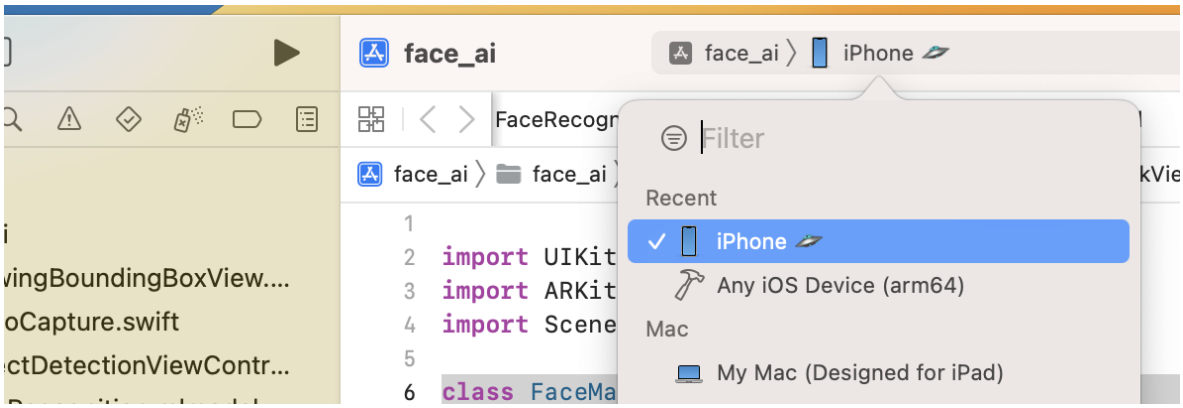

Рис. 3.16. Вигляд емуляції пристрою що підключено

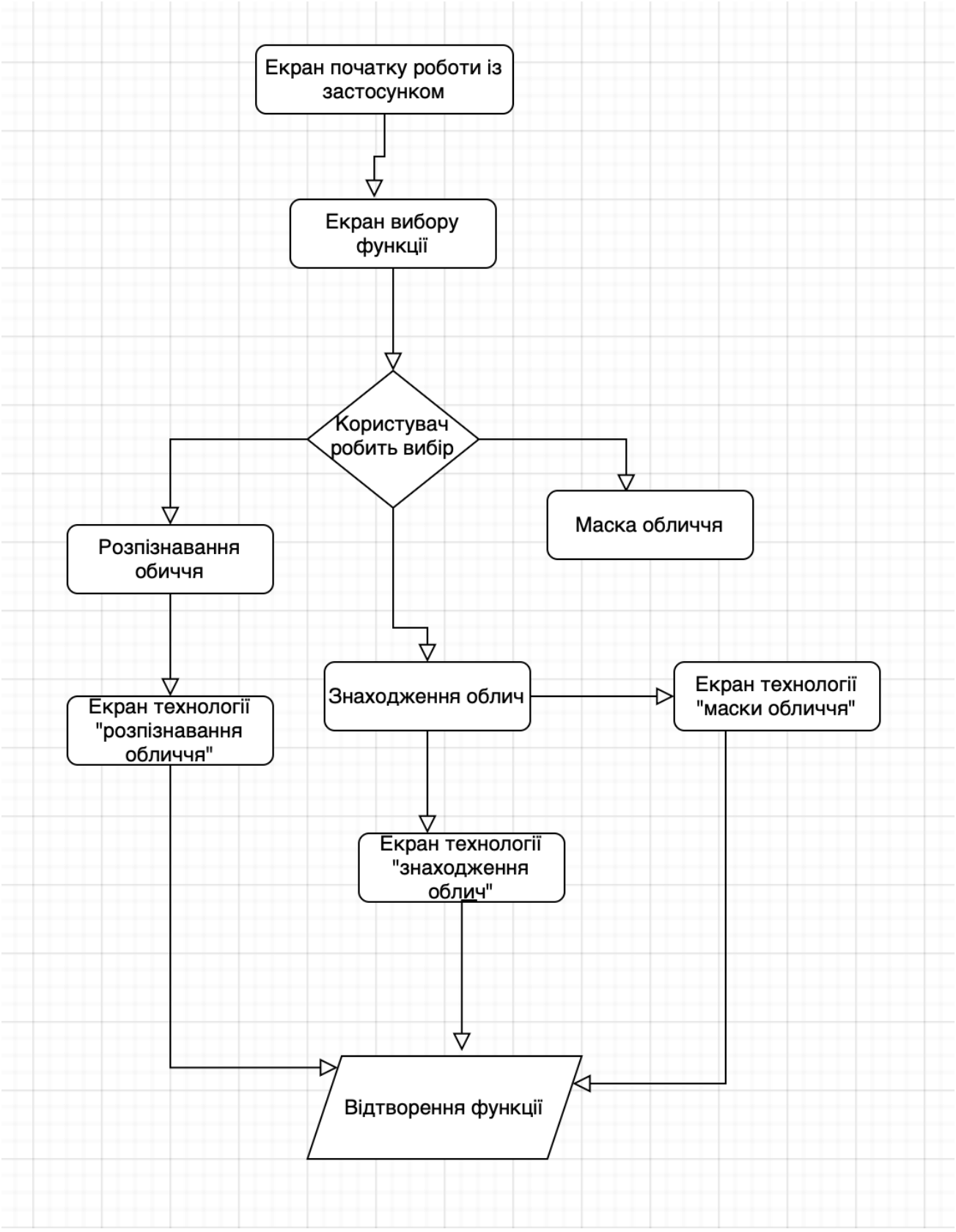

Рис. 3.18. блок схема принципу роботи застосунку

ВИСНОВКИ ДО РОЗДІЛУ 3

В даному розділі було описано алгоритм роботи та розроблено мобільний застосунок та його дизайн.

Для розробки користувацького інтерфейсу було використано принципи Apple Human Guidelines, це дозволило розробити зручний та інтуїтивно зрозумілий користувацький інтерфейс.

Написання застосунку здійснювалась за допомогою мови програмування swift у серидовищі розробки Xcode, для головної функції застосунку, а саме розпізнавання обличчя було використано Vision.api.

Це дало можливість реалізації зчитування даних з відео, фото та камери на будь яких сумісних із цією технологією сучасних смартфонах та пристроях.

Також застосунок було запущенно шляхом емуляції на особлистому пристрої підключеного до пк, що дало змогу переконатись у повній працездатності застосунку.

ВИСНОВКИ

Завдяки вбудованому Vision.api від Apple обробка зображень і створення потужних моделей ніколи не були такими простими. Хоча потужніші моделі можна створювати для розпізнавання обличчя за допомогою фреймворків, таких як PyTorch або TensorFlow, у цих моделей, як правило, не вистачає багатьох атрибутів, необхідних для роботи на пристрої, як-от швидкість і роздутий розмір, і тому їх набагато важче розмістити на мобільних пристроях. пристроїв.

Найважливішим аспектом роботи з нативними API є те, що в більшості випадків вони оптимізовані для роботи з багатьма пристроями, а також покращені за допомогою компонентів апаратного прискорення, таких як Neural Engine від Apple.

На сьогоднішній день, розпізнавання облич дозволяє ідентифікувати людські обличчя на зображеннях або відео, визначати, чи належить обличчя на двох зображеннях одній людині, або знаходити обличчя у великій колекції існуючих зображень. Біометричні системи безпеки використовують розпізнавання обличчя для унікальної ідентифікації людей під час реєстрації або входу в систему, а також для посилення процедур автентифікації користувачів.

Алгоритми розпізнавання обличчя мають практично ідеальну точність в ідеальних умовах. У контрольованих умовах спостерігається більш високий відсоток точності, але в реальному світі він, як правило, нижчий. Передбачити рівень точності цієї технології складно, оскільки жоден показник не дає повної картини.

Проаналізовано системи розпізнавання обличчя, порвіняння та проблематики цих систем, це дало змогу створити перелік вимог до функціоналу, який повинен бути доступним у застосунку для демострації технології.

Проаналізовано наукову та методичну літературу. Це допомогло поглибити знання з проектування та розробки мобільних додатків, мови програмування Swift.

Під час написання кваліфікаційної роботи досліджено сучасні шаблони проектування, архітектурні рішення написання додатків, шаблони проектування додатків, мову дизайну Apple Human Guidelines.

Для розробки мобільного додатка використано середу розробки застосунків Xcode, мову програмування Swift та Vision.api. Серед великої кількості архітектурних рішень використано та детально описано Face Detection, Face Recognition та Face Tracking. Використання даних шаблонів та архітектурних рішень дало можливість написати легко тестований та чітко структуризований код.

Описано принцип роботи мобільного застосунку та складено схему опису алгоритму його роботи.

Всі поставленні завдання виконано. Результатом виконання кваліфікаційної роботи є застосунок для демонстрації робтоти технології розпізнавання обличчя. Даний мобільний застосунок відповідає описаним та загальним вимогам сучасних мобільних застосунків, повністю працездатний та готовий до інтегрування в систему підприємств та компаній.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bach, F., Jordan, M.: ‘Kernel independent component analysis', J. Mach. Learn. Res., 2003, 3, pp. 1–48
2. Suruliandi, A., Meena, K., Reena, R.R.: ‘Local binary pattern and its derivatives for face recognition', IET Comput. Vis., 2012, 6, (5), pp. 480–488.
3. Facial recognition system [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Facial_recognition_system
4. Система розпізнавання облич [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Система_розпізнавання_облич
5. Use Face ID on your iPhone [Електронний ресурс] – Режим доступу до ресурсу: https://support.apple.com/en-us/HT208109
6. Azure Face service[Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/cognitive-services/computer-vision/overview-identity
7. Система Aware Nexa|Face [Електронний ресурс] – Режим доступу до ресурсу: https://www.aware.com/biometrics/nexa-facial-recognition/
8. Samsung Face Recognition [Електронний ресурс] – Режим доступу до ресурсу: https://www.samsung.com/us/support/answer/ANS00062630/
9. Attacking the face recognition authentication[Електронний ресурс] Режим доступу до ресурсу: https://www.securing.pl/en/attacking-the-face-recognition-authentication-how-easy-is-to-fool-it/

10. Invisibility Sweater[Електронний ресурс] – Режим доступу до ресурсу: https://expatguideturkey.com/invisibility-sweater-discovered/

11. HOW CHINA USES FACIAL RECOGNITION TO CONTROL HUMAN BEHAVIOR[Електронний ресурс] – Режим доступу до ресурсу: https://www.cnet.com/news/politics/in-china-facial-recognition-public-shaming-and-control-go-hand-in-hand/

12. Xcode [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Xcode

13. Human Design Guidelines [Еле ктронний ресурс] – Режим доступу до ресурсу: https://www.learnui.design/blog/ios-design-guidelines-templates.html

14. Apple ecosystem explained [Електронний ресурс] – Режим доступу до ресурсу: https://techjourneyman.com/blog/apple-ecosystem-explained/

15. On-device APIs [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/machine-learning/api/

16. Detect people, faces, and poses using Vision [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/videos/play/wwdc2021/10040/?time=40

17. Facial recognition system [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Facial_recognition_system

18. Facial recognition: top 7 trends (tech, vendors, use cases) [Електронний ресурс] – Режим доступу до ресурсу: https://www.thalesgroup.com/en/markets/digital-identity-and-security/government/biometrics/facial-recognition

19. Як знайти компроміс між зручністю та приватністю при розпізнаванні облич[Електронний ресурс] – Режим доступу до ресурсу: https://chas.news/future/starshii-brat-bilshe-ne-stezhit-za-toboyu-chomu-demokratichni-kraini-vidmovlyayutsya-vid-tehnologii-rozpiznavannya-oblich
20. World-leading Face Recognition for Multiple Industries [Електронний ресурс] – Режим доступу до ресурсу: https://www.nec.com/en/global/solutions/biometrics/face/index.html
21. Порівняння сервісів розпізнавання облич [Електронний ресурс] – Режим доступу до ресурсу: https://evergreens.com.ua/ua/articles/facial-recognition-services-comparison.html
22. Clearview Stole My Face and the EU Can't Do Anything About It [Електронний ресурс] – Режим доступу до ресурсу: https://www.wired.com/tag/face-recognition/

ДОДАТОК А «API ДОКУМЕНТАЦІЯ»

```
import UIKit
import Vision
import AVKit
import CoreMedia
import VideoToolbox
import SceneKit

class ObjectDetectionViewController: UIViewController,
AVCaptureVideoDataOutputSampleBufferDelegate {

    // MARK: - Vision Properties
    var request: VNCoreMLRequest?
    var OcrRequest: VNRecognizeTextRequest?
    var visionModel: VNCoreMLModel?
    var isInferencing = false

    // MARK: - AV Property
    var videoCapture: VideoCapture!
    let semaphore = DispatchSemaphore(value: 1)

    let videoPreview: UIView = {
      let view = UIView()
       view.translatesAutoresizingMaskIntoConstraints = false
       return view
    }()

    let BoundingBoxView: DrawingBoundingBoxView = {
      let boxView = DrawingBoundingBoxView()
       boxView.translatesAutoresizingMaskIntoConstraints = false
       return boxView
    }()
    override func viewDidLoad() {
       super.viewDidLoad()

       view.backgroundColor = .black
       setupTabBar()
       setUpModel()
       setupCameraView()
       setUpCamera()
       setupBoundingBoxView()
    }


          self.navigationController?.navigationBar.barTintColor = .lightText
          navigationController?.navigationBar.titleTextAttributes = [.foregroundColor :
UIColor.black]
       }
```

```
        self.navigationController?.navigationBar.isHidden = false
        self.setNeedsStatusBarAppearanceUpdate()
        self.navigationItem.largeTitleDisplayMode = .automatic
        self.navigationController?.navigationBar.barStyle = .default
        if #available(iOS 13.0, *) {
            navigationController?.navigationBar.largeTitleTextAttributes =
[.foregroundColor : UIColor.label]
        } else {
            navigationController?.navigationBar.largeTitleTextAttributes =
[.foregroundColor : UIColor.black]
        }
        if #available(iOS 13.0, *) {
            navigationController?.navigationBar.backgroundColor = .systemBackground
        } else {
            // Fallback on earlier versions
            navigationController?.navigationBar.backgroundColor = .white
        }
        self.tabBarController?.tabBar.isHidden = false
    }


    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        self.videoCapture.start()
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)
        self.videoCapture.stop()
    }

    override func viewDidLayoutSubviews() {
        super.viewDidLayoutSubviews()
        resizePreviewLayer()
    }

    func resizePreviewLayer() {
        videoCapture.previewLayer?.frame = videoPreview.bounds
    }

    // MARK: - Setup CoreML model and Text Request recognizer
    func setUpModel() {
        if let visionModel = try? VNCoreMLModel(for: model_face_omar_turi().model) {
            self.visionModel = visionModel
            request = VNCoreMLRequest(model: visionModel, completionHandler:
visionRequestDidComplete)
```

```
            request?.imageCropAndScaleOption = .scaleFill
        } else {
            fatalError("fail to create vision model")
        }
    }

    // MARK: - SetUp Camera preview
    func setUpCamera() {
        videoCapture = VideoCapture()
        videoCapture.delegate = self
        videoCapture.fps = 30
        videoCapture.setUp(sessionPreset: .high) { success in

            if success {
                // add preview view on the layer
                if let previewLayer = self.videoCapture.previewLayer {
                    self.videoPreview.layer.addSublayer(previewLayer)
                    self.resizePreviewLayer()
                }

                // start video preview when setup is done
                self.videoCapture.start()
            }
        }
    }

    fileprivate func setupCameraView() {
        view.addSubview(videoPreview)
        videoPreview.bottomAnchor.constraint(equalTo: view.bottomAnchor).isActive =
true
        videoPreview.leftAnchor.constraint(equalTo: view.leftAnchor).isActive = true
        videoPreview.rightAnchor.constraint(equalTo: view.rightAnchor).isActive = true
        videoPreview.topAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.topAnchor).isActive = true
    }

    fileprivate func setupBoundingBoxView() {
        view.addSubview(BoundingBoxView)
        BoundingBoxView.bottomAnchor.constraint(equalTo:
videoPreview.bottomAnchor).isActive = true
        BoundingBoxView.leftAnchor.constraint(equalTo:
videoPreview.leftAnchor).isActive = true
        BoundingBoxView.rightAnchor.constraint(equalTo:
videoPreview.rightAnchor).isActive = true
        BoundingBoxView.topAnchor.constraint(equalTo:
videoPreview.topAnchor).isActive = true
    }

}
```

```
extension ObjectDetectionViewController: VideoCaptureDelegate {
    func videoCapture(_ capture: VideoCapture, didCaptureVideoFrame pixelBuffer:
CVPixelBuffer?, timestamp: CMTime) {
        // the captured image from camera is contained on pixelBuffer
        if !self.isInferencing, let pixelBuffer = pixelBuffer {
            self.isInferencing = true
            // predict!
            self.predictUsingVision(pixelBuffer: pixelBuffer)
        }
    }
}

extension ObjectDetectionViewController {
    func predictUsingVision(pixelBuffer: CVPixelBuffer) {
        guard let request = request else { fatalError() }
        // vision framework configures the input size of image following our model's
input configuration automatically which is 416X416
        self.semaphore.wait()
        let handler = VNImageRequestHandler(cvPixelBuffer: pixelBuffer)
        try? handler.perform([request])
    }

    // MARK: - Post-processing
    func visionRequestDidComplete(request: VNRequest, error: Error?) {
        if let predictions = request.results as? [VNRecognizedObjectObservation] {
            DispatchQueue.main.async {
                self.BoundingBoxView.predictedObjects = predictions
                self.isInferencing = false
            }
        } else {

            self.isInferencing = false
        }
        self.semaphore.signal()
    }
}


class FaceClassificationViewController: UIViewController,
AVCaptureVideoDataOutputSampleBufferDelegate {

    let label: UILabel = {
        let label = UILabel()
        label.backgroundColor = .clear
        label.textAlignment = .center
        label.translatesAutoresizingMaskIntoConstraints = false
        label.textColor = .orange
        label.font = UIFont(name: "Avenir-Heavy", size: 30)
        label.text = "No face"
        return label
```

```
    }()


    override func viewDidLoad() {
        super.viewDidLoad()
        setupTabBar()
        setupCamera()
        setupLabel()
    }


self.navigationController?.navigationBar.barTintColor = .systemBackground
            navigationController?.navigationBar.titleTextAttributes = [.foregroundColor :
UIColor.label]
        } else {
            // Fallback on earlier versions
            self.navigationController?.navigationBar.barTintColor = .lightText
            navigationController?.navigationBar.titleTextAttributes = [.foregroundColor :
UIColor.black]
        }
        self.navigationController?.navigationBar.isHidden = false
        self.setNeedsStatusBarAppearanceUpdate()
        self.navigationItem.largeTitleDisplayMode = .automatic
        self.navigationController?.navigationBar.barStyle = .default
        if #available(iOS 13.0, *) {
            navigationController?.navigationBar.largeTitleTextAttributes =
[.foregroundColor : UIColor.label]
        } else {
            navigationController?.navigationBar.largeTitleTextAttributes =
[.foregroundColor : UIColor.black]
        }
        if #available(iOS 13.0, *) {
            navigationController?.navigationBar.backgroundColor = .systemBackground
        } else {
            // Fallback on earlier versions
            navigationController?.navigationBar.backgroundColor = .white
        }
        self.tabBarController?.tabBar.isHidden = false
    }

    fileprivate func setupCamera() {
        let captureSession = AVCaptureSession()
        captureSession.sessionPreset = .high

        guard let captureDevice = AVCaptureDevice.default(for: .video) else { return }
        guard let input = try? AVCaptureDeviceInput(device: captureDevice) else {
return }
        captureSession.addInput(input)

        captureSession.startRunning()
```

```
        let previewLayer = AVCaptureVideoPreviewLayer(session: captureSession)
        view.layer.addSublayer(previewLayer)
        previewLayer.frame = view.frame

        let dataOutput = AVCaptureVideoDataOutput()
        dataOutput.setSampleBufferDelegate(self, queue: DispatchQueue(label:
"videoQueue"))
        captureSession.addOutput(dataOutput)
    }

    fileprivate func setupLabel() {
        view.addSubview(label)
        label.bottomAnchor.constraint(equalTo: view.bottomAnchor, constant: -
32).isActive = true
        label.leftAnchor.constraint(equalTo: view.leftAnchor).isActive = true
        label.rightAnchor.constraint(equalTo: view.rightAnchor).isActive = true
        label.heightAnchor.constraint(equalToConstant: 80).isActive = true
    }

    func captureOutput(_ output: AVCaptureOutput, didOutput sampleBuffer:
CMSampleBuffer, from connection: AVCaptureConnection) {

        guard let pixelBuffer: CVPixelBuffer =
CMSampleBufferGetImageBuffer(sampleBuffer) else { return }

        guard let model = try? VNCoreMLModel(for: FaceRecognition().model) else {
                fatalError("Unable to load model")
            }

        let coreMlRequest = VNCoreMLRequest(model: model) {[weak self] request,
error in
            guard let results = request.results as? [VNClassificationObservation],
                let topResult = results.first
                else {
                fatalError("Unexpected results")
            }

            DispatchQueue.main.async {[weak self] in
                self?.label.text = topResult.identifier
            }
        }

        let handler = VNImageRequestHandler(cvPixelBuffer: pixelBuffer, options: [:])
        DispatchQueue.global().async {
            do {
                try handler.perform([coreMlRequest])
            } catch {
                print(error)
            }
```

```
        }

    }

}

class FaceMaskViewController: UIViewController,
AVCaptureVideoDataOutputSampleBufferDelegate, ARSessionDelegate {

    let sceneView = ARSCNView(frame: UIScreen.main.bounds)

    override func viewDidLoad() {
        super.viewDidLoad()
        self.view.addSubview(sceneView)
        sceneView.delegate = self
        guard ARFaceTrackingConfiguration.isSupported else { return }
        let configuration = ARFaceTrackingConfiguration()
        configuration.isLightEstimationEnabled = true
        sceneView.session.run(configuration, options: [.resetTracking,
.removeExistingAnchors])
        setupTabBar()
    }


            self.navigationController?.navigationBar.barTintColor = .systemBackground
             navigationController?.navigationBar.titleTextAttributes = [.foregroundColor :
UIColor.label]
        } else {
            // Fallback on earlier versions
            self.navigationController?.navigationBar.barTintColor = .lightText
            navigationController?.navigationBar.titleTextAttributes = [.foregroundColor :
UIColor.black]
        }
        self.navigationController?.navigationBar.isHidden = false
        self.setNeedsStatusBarAppearanceUpdate()
        self.navigationItem.largeTitleDisplayMode = .automatic
        self.navigationController?.navigationBar.barStyle = .default
        if #available(iOS 13.0, *) {
            navigationController?.navigationBar.largeTitleTextAttributes =
[.foregroundColor : UIColor.label]
        } else {
            navigationController?.navigationBar.largeTitleTextAttributes =
[.foregroundColor : UIColor.black]
        }
        if #available(iOS 13.0, *) {
            navigationController?.navigationBar.backgroundColor = .systemBackground
        } else {
            // Fallback on earlier versions
            navigationController?.navigationBar.backgroundColor = .white
        }
```

```
        self.tabBarController?.tabBar.isHidden = false
    }


}

extension FaceMaskViewController: ARSCNViewDelegate {

    func renderer(_ renderer: SCNSceneRenderer, nodeFor anchor: ARAnchor) ->
SCNNode? {

        guard let device = sceneView.device else {
            return nil
        }

        let faceGeometry = ARSCNFaceGeometry(device: device)

        let node = SCNNode(geometry: faceGeometry)

        node.geometry?.firstMaterial?.fillMode = .lines

        return node
    }

    func renderer(_ renderer: SCNSceneRenderer, didUpdate node: SCNNode, for
anchor: ARAnchor) {

        guard let faceAnchor = anchor as? ARFaceAnchor,
            let faceGeometry = node.geometry as? ARSCNFaceGeometry else {
                return
        }

        faceGeometry.update(from: faceAnchor.geometry)

    }

}
```