МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ З АВТОРИЗАЦІЄЮ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ»**

Виконав: студент 2 курсу, групи 8.1212-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
В.С. Левада
(ініціали та прізвище)

Керівник завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Леваді Владиславу Сергійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка мобільного застосунку з авторизацією на основі штучного інтелекту

керівник роботи Гребенюк Сергій Миколайович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 642-с

2. Строк подання студентом роботи 27.11.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Опис програмної реалізації мобільного застосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 03.05.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	17.05.2023	
2.	Збір вихідних даних.	07.06.2023	
3.	Обробка методичних та теоретичних	28.06.2023	
	джерел.		
4.	Розробка першого та другого розділу.	30.08.2023	
5.	Розробка третього розділу.	08.11.2023	
6.	Оформлення та нормоконтроль	20.11.2023	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	14.12.2023	

Студент _____ В.С. Левада
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.М. Гребенюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка мобільного застосунку з авторизацією на основі штучного інтелекту»: 60 с., 32 рис., 11 джерел.

КОМП'ЮТЕРНИЙ ЗІР, РОЗПІЗНАВАННЯ ОБЛИЧЧЯ, ШТУЧНИЙ ІНТЕЛЕКТ, MTCNN, OPENCV, TENSORFLOW.

Об'єкт дослідження – процес розпізнавання обличчя за допомогою штучного інтелекту.

Предмет дослідження – автентифікація на основі розпізнавання обличчя.

Мета роботи – дослідити методи які використовуються для розпізнавання обличчя, втілити їх у програмну систему авторизації.

В результаті роботи була створена клієнт-серверна програмна система з функціоналом авторизації.

На протязі виконання магістерської кваліфікаційної роботи були досліджені методи комп'ютерного зору які дозволяють аналізувати зображення виявляючи об'єкти. Також була досліджена модель MTCNN яка дозволяє виявляти та розпізнавати обличчя.

Втілення можливостей комп'ютерного зору та розпізнавання обличчя у серверний додаток відбувалося за допомогою мови програмування Python, та бібліотеки TensorFlow. Для представлення зображень використовувались бібліотеки numpy та OpenCV.

В результаті роботи була створена клієнт-серверна програмна система наділена можливостями штучного інтелекту для розпізнавання обличчя та подальшої авторизації.

SUMMARY

Master’s qualifying paper «Development of the Mobile Application with Authorization Based on the Artificial Intelligence»: 60 pages, 32 figures, 11 references.

COMPUTER VISION, FACE RECOGNITION, ARTIFICIAL INTELLIGENCE, MTCNN, OPENCV, TENSORFLOW.

Object of the study – facial recognition process with an artificial intelligence.

Subject of the study – authentication by the facial recognition.

Aim of the study is facial recognition methods research, bring them into authorisation system.

The client-server system with authorization was developed as a result of the work.

In the course of the master’s qualification work, computer vision methods were studied, which allow analyzing images by detecting objects.

The MTCNN model was also investigated, which allows detecting and recognizing faces.

The implementation of computer vision and face recognition capabilities in the server application was done using the Python programming language and the TensorFlow library. The numpy and OpenCV libraries were used to represent the images.

A client-server software system was created as a result of the work. The system has artificial intelligence capabilities for facial recognition and further authorization.

ЗМІСТ

ВСТУП

Обчислювальна техніка вже давно займає значне місце як для життя окремої людини, так і для суспільства в цілому. Різноманітність цифрових пристроїв дозволяє оптимізувати велику кількість галузей людської діяльності. Сьогодні неможливо уявити життя без комп'ютерів у будь-якому їхньому вигляді. Ноутбуки, сервери, мейнфрейми, смартфони, планшети, інтернет речей формують основний список комп'ютерів, з якими безпосередньо або опосередковано контактує кожна людина.

Використовуючи комп'ютер для роботи, навчання, перевірки повідомлень, банківського рахунку, інтернет-покупок та іншого користувач повинен перш за все отримати доступ до відповідних ресурсів. Авторизація користувачів – це дуже важлива складова інформаційної безпеки. Якщо зловмисник отримає доступ, наприклад, до банківського рахунку, наслідки будуть дуже неприємними.

У роботі поставлено за мету створити програмну систему, яка реалізує надійний механізм доступу. Система складається з серверної та клієнтської частин. Сервер прослуховує та виконує запити на реєстрацію або авторизацію. Клієнт – це мобільний Андроїд додаток, в якому користувач може зареєструватись або пройти авторизацію.

Важлива складова розробленої системи – це технологія штучного інтелекту яка використовується для контролю доступу. Окрім введення пароля користувач може отримати доступ за допомогою свого обличчя. Отримавши зображення сервер за допомогою бібліотеки TensorFlow активізує процес машинного навчання для розпізнавання обличчя. Після цього формується посилання та ідентифікатор користувача які будуть ключем доступу.

Розпізнавання обличчя, як одна з похідних технологій на основі штучного інтелекту, дозволяє створювати надійний контроль доступу у

поєднанні з ергономічністю. Користувачеві достатньо навести камеру пристрою на своє обличчя без необхідності введення пароля. Надійність такого способу авторизації посилюється тим, що спроба піддивитись пароль зловмисником втрачає сенс.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Контроль доступу у кіберпросторі

Контроль доступу може бути реалізований різними способами. Засоби контролю доступу можуть базуватись на основі фізичних атрибутів, набору правил, списків осіб чи систем, або більш складних факторів. Конкретний тип контролю доступу часто залежить від середовища, в якому він буде використовуватися.

У багатьох програмах та операційних системах реалізовані базові засоби контролю доступу. Військові структури та державні органи використовують програмні системи з підвищеною безпекою. Такі системи можуть мати багаторівневу архітектуру захисту.

Надання доступу полягає у створенні можливості використовувати той чи інший ресурс. Наприклад, це може бути доступ до файлу для відповідного користувача або групи користувачів. Після отримання доступу користувач може читати, редагувати, або видалити файл. Також доступ до файлу, оперативної пам’яті, або бази даних може отримати певний процес.

Відмова в доступі – це діаметрально протилежна процедура наданню доступу. У даному контексті виконується запобігання доступу до відповідного ресурсу. Заборона доступу може бути направлена на певну особу, яка намагається ввійти до програмної системи залежно від часу доби.

Багато систем контролю доступу налаштовано на заборону за замовчуванням, причому доступ надається лише авторизованим користувачам. Надійно захищена система сформує відмову в доступі для зловмисника. Користувач буде повідомлений про спробу несанкціонованого доступу до його даних в системі через електрону пошту або інші способи зв’язку.

1.2 Механізми контролю доступу

Механізми контролю доступу – це логічні компоненти, які визначають спосіб отримання запиту від суб’єкта на доступ до об’єкта та ухвалення або відхилення цього доступу. Суб’єктом може бути користувач або інша сутність (наприклад, процес). На місці об’єкта може бути файл, база даних, апаратні ресурси (такі, як оперативна пам’ять).

До механізмів доступу належать ідентифікація, автентифікація й авторизація які виконуються у зазначеній послідовності.

1.2.1 Ідентифікація

Ідентифікація – це процедура в результаті якої визначається ідентифікатор суб’єкта. Ідентифікацію можна також розуміти як твердження про те, ким є суб’єкт. Ідентифікатором може бути ім’я, логін, або число яке однозначно ідентифікує суб’єкт. Під суб’єктом слід розуміти користувача або процес в інформаційній системі. Призначення ідентифікатору відбувається на етапі реєстрації користувача або створення процесу.

Ідентифікація важлива не лише для захисту конфіденційності, але й для ефективного керування доступом. Системи управління доступом базуються на точній ідентифікації користувачів, щоб надати їм відповідні рівні доступу до різних ресурсів. Це особливо важливо в організаціях, де різні користувачі мають різні ролі та обов’язки.

З розвитком інтернету речей (Internet of Things, IoT) та великих даних, аспекти ідентифікації стають ще складнішими. Важливо розглядати нові методи індивідуалізації та безпеки, щоб уникнути вразливостей у підключених пристроях та забезпечити стійкість інформаційних систем.

Процедура ідентифікації в інформаційних системах є ключовим фактором для запобігання несанкціонованому доступу, зловживанню даними та іншим видам кіберзлочинності.

1.2.2 Автентифікація

Автентифікація – це процес перевірки ідентичності користувача для надання або відмови у доступі до певних ресурсів чи послуг. Процес автентифікації виконується після ідентифікації. Логіка автентифікації визначає, чи є суб’єкт тим, за кого він себе видає.

Автентифікація може засновуватись на факторах. Кожен фактор налічує певну кількість методів. Коли система намагається автентифікувати заяву про ідентифікацію, чим більше факторів використовуються, тим кращий буде результат [1]. У контексті автентифікації користувача факторами є:

– те, що знає користувач;
– те, чим є користувач;
– чим володіє користувач;
– що робить користувач.

Найбільш поширений фактор автентифікації – це те, що знає користувач. Цей фактор включає паролі, пін-коди, парольні фрази, або іншу інформацію яку особа може запам’ятати.

Фактор «те, чим є користувач» заснований на відносно унікальних фізичних характеристиках людини. Ці характеристики часто називають біометрією. До цього фактору відносяться зріст, вага, колір очей або волосся. Але такі атрибути не є достатньо унікальними. Тому для автентифікації на основі цього фактору використовують складніші біометричні дані: відбитки пальців, райдужна оболонка або сітківка ока, характеристики обличчя. Цей фактор більш надійний, оскільки підробка чи викрадення копії фізичного ідентифікатора є дещо складнішим, але, все ж таки можливим завданням.

Фактор «чим володіє користувач» базується на спеціальних речах, котрі може мати особа при собі. Це може бути банківська картка або апаратний токен авторизації. Банки та інші установи у якості автентифікації використовують доступ до мобільних телефонів та скриньок електронної

пошти. Надійність цього фактору може бути різною залежно від реалізації. У випадку апаратного токена зловмиснику потрібно викрасти певний пристрій, щоб підробити метод автентифікації. У разі використання доступу до адреси електронної пошти надійність буде меншою.

Фактор «те, що робить користувач» заснований на діях або поведінці окремої людини. У даному контексті можуть розглядатись хода людини, її почерк, часові інтервали між натисканнями клавіш коли виконується введення пароля та інше. Такий фактор вважається сильним методом автентифікації, який дуже важко підробити [1]. Але у цьому випадку мають місце досить скрупульозні вимоги до самої системи яка реалізовує даний фактор.

Багато систем використовує багатофакторну автентифікацію. Прикладом багатофакторної автентифікації може бути використання банкомата. У цьому випадку людина використовує те, що знає і те, чим володіє: пін-код, банківська картка. Слід зазначити, що банківська картка може виконувати функцію ідентифікації та автентифікації.

Залежно від певних факторів можна побудувати більш, або менш надійну автентифікацію. У деяких випадках використовувати багатофакторну автентифікацію недоцільно. Створення безпеки в інформаційній системі повинно бути пропорційне захищеному об’єкту.

1.2.3 Авторизація

Авторизація є наступним кроком після ідентифікації та автентифікації. Авторизація – це процес надання користувачеві права на доступ до певних ресурсів чи функцій системи.

Авторизація визначає, які саме ресурси чи дії є доступними для конкретного користувача після успішної автентифікації. Логіка надання доступу повинна засновуватись на концепції, яка називається принципом

найменших привілеїв. Відповідно даній концепції, суб’єкту, яким може бути користувач або процес, надається мінімальний доступ до ресурсів необхідних для виконання певної функції. Порушення принципу найменших привілеїв є основною проблемою систем інформаційної безпеки.

Одним із найпоширеніших способів неналежного застосування принципу найменших привілеїв є дозволи, які надаються обліковим записам користувачів операційної системи. Ці дозволи найчастіше порушуються користувачами та адміністраторами операційних систем Microsoft.

В операційних системах Microsoft звичайні користувачі ОС, які виконують такі завдання, як створення документів у текстових процесорах та обмін електронною поштою, мають налаштований адміністративний доступ, що дозволяє їм виконувати будь-які завдання, які дозволяє ОС. Як наслідок цього користувач із надлишковими правами відкриває електронну пошту, що містить зловмисне програмне забезпечення, або стикається з вебсайтом, який надсилає код атаки на клієнтський комп’ютер. Ці атаки мають повну владу над системою, оскільки вони діють як користувач, який в свою чергу наділений адміністративними можливостями. Завдяки цьому робота зловмисників значно полегшується, оскільки вони можуть просто вимкнути інструменти захисту від зловмисного програмного забезпечення, встановити будь-які додаткові інструменти для атак, які їм потрібні.

Така ж сама проблема спостерігається в службах або процесах, які працюють на більш привілейованому рівні, ніж це потрібно для виконання передбачених функцій. Наприклад, якщо в системі є служба, на якій працює вебсервер, цій службі потрібен дозвіл для доступу до файлів і сценаріїв, які безпосередньо стосуються вебконтенту, який вона обслуговує, і нічого більше. Якщо дозволити вебслужбі отримати доступ до додаткових файлів у файловій системі, зловмисник потенційно може прочитати або змінити ці файли, щоб отримати несанкціонований доступ до більш конфіденційної інформації.

1.3 Штучний інтелект та контроль доступу

Штучний інтелект (ШІ) значно трансформує інформаційні технології. Інтеграція ШІ в системи контролю доступу може сформувати новий рівень захищеності. Системи контролю доступу відіграють ключову роль у захисті конфіденційної інформації та фізичних просторів. Використання технології штучного інтелекту в цих системах надає безпрецедентні можливості, підвищуючи як точність, так і адаптивність керування доступом. За допомогою ШІ в системах контролю доступу набувають сенс такі функції:

– біометрична автентифікація;
– прогнозний аналіз для запобігання загрозам;
– політики адаптивного доступу.

Біометрична автентифікація включаючи розпізнавання обличчя та поведінкову біометрію, забезпечує підвищену точність і стійкість проти спроб шахрайства, встановлюючи новий стандарт перевірки особи.

Алгоритми штучного інтелекту аналізують моделі поведінки користувачів, уможливлюючи прогнозний аналіз загроз. Це дає змогу системам контролю доступу передбачати та запобігати порушення безпеки до того, як воно відбудеться.

Прогнозний аналіз виконується шляхом отримання внутрішніх зв’язків даних, отриманих шляхом добування корисної інформації [2].

Методи прогнозної аналітики є перспективним напрямком досліджень кібербезпеки, який дозволить більш доцільно підходити до створення безпеки. Прогнози можуть слугувати раннім попередженням, щоб фахівці могли заздалегідь дізнатися про загрози, налаштувати відповідні контрзаходи та завчасно пом’якшити або повністю запобігти небажаним інцидентам [2].

Алгоритми штучного інтелекту раціонально створюють політики адаптивного доступу на основі зміни поведінки користувачів і контекстних факторів, забезпечуючи рівень адаптивності, якого важко досягти традиційним системам на основі правил.

У даній кваліфікаційній роботі розглядається контроль доступу на основі розпізнавання обличчя, який належить до біометричної автентифікації.

1.3.1 Біометрія

Біометричне розпізнавання можна визначити як науку про встановлення ідентифікації людини спираючись на її фізичних даних, характеристику поведінки. Таке розпізнавання повністю або частково автоматизовано. Біометрія широко розповсюджується у сучасному світі.

Люди зазвичай спираються на такі характеристики тіла, як обличчя, голос і хода, а також іншу контекстну інформацію (наприклад, місцезнаходження та одяг), щоб впізнавати один одного. Набір атрибутів, пов’язаних з людиною, становить її особисту ідентичність.

Біометрія пропонує доволі надійне вирішення проблеми розпізнавання особи. Оскільки біометричні ідентифікатори притаманні людині, важче маніпулювати цими рисами, ділитися ними або забути. Отже, біометричні ознаки становлять міцний і досить сильний зв’язок між людиною та її ідентичністю.

Користувачем системи може бути будь-яка особа, яка надає свій біометричний ідентифікатор з метою розпізнавання. Біометричне розпізнавання все частіше застосовується у ряді державних та цивільних системах керування доступом, щоб замінити або доповнити існуючі механізми на основі знань і токенів.

Біометрична система аналізує одну або кілька фізичних, поведінкових характеристик, включаючи відбитки пальців, долоні, обличчя, райдужну оболонку, сітківку ока, голос, підпис, ходу або інформацію про ДНК особи, щоб ідентифікувати користувача [3].

Виявлення та розпізнавання осіб на заданому зображенні на основі

обличчя є класичними проблемами розпізнавання об’єктів, яким приділяється велика увага в літературі з комп’ютерного зору. Хоча вважається, що люди добре впізнають знайомі обличчя, точні когнітивні процеси, пов’язані з цією діяльністю, до кінця не вивчені. Тому навчити машину розпізнавати обличчя, як це роблять люди, є складним завданням [3].

1.3.2 Поняття штучного інтелекту

Штучний інтелект викликає надзвичайну зацікавленість серед науковців та інших людей. Єдиного визначення поняття ШІ не існує. Деякі формулювання ШІ стосуються розумових процесів та способів міркування, деякі належать до поведінки. Використовуючи технологію штучного інтелекту фахівці ставлять за мету відтворення здібностей людини з максимальною точністю. Система заснована на технології ШІ може виконувати складні нетривіальні задачі та самостійно поповнювати свою базу знань.

Також можна навести визначення ШІ з авторитетних наукових робіт:

Спосіб навчити комп’ютери робити те, в чому люди на зараз мають перевагу [4].

Обчислювальний інтелект – це наука, яка вивчає спосіб проєктування інтелектуальних агентів [5].

Вивчення таких обчислень, які дозволяють відчувати, розмірковувати та діяти [6].

Основна суть штучного інтелекту полягає у створенні інтелектуальних агентів, суб’єктів, наповнених здатністю сприймати навколишнє середовище та стратегічно діяти для досягнення максимального успіху. Ця фундаментальна концепція є коренем для систем штучного інтелекту, де теорія та раціональне прийняття рішень відіграють ключову роль у формуванні поведінки цих інтелектуальних агентів.

Дослідження алгоритмів які розв’язують задачі займає важливе місце навколо ШІ. В основі розв’язання задач у ШІ лежать традиційні, евристичні алгоритми пошуку. Коли в системі присутня багатоагентна середа, де цілі агентів конфліктують, виникають задачі пошуку в умовах протистояння. Такі задачі називають іграми [6].

Важливе поняття яке лежить в основі штучного інтелекту – це машинне навчання. Технологія машинного навчання (МН) може використовуватись у таких задачах, як розпізнавання образів. МН полягає у використанні даних й алгоритмів для імітації способу яким навчаються люди. Процес машинного навчання виконується багато раз, щоб поступово покращувати точність обробки даних. Існує декілька підходів машинного навчання включаючи кероване навчання, некероване навчання, напівкероване навчання, навчання з підкріпленням. Перед виконанням процесу МН можуть створюватись спеціальні моделі які тренуються на заздалегідь підготовлених даних. Такими моделями можуть бути штучні нейронні мережі. Машинне навчання дає змогу ШІ збирати інформацію, робити прогнози та динамічно пристосовуватися до середовища, що розвивається.

Дослідження плавно поширюється на область робототехніки, де принципи штучного інтелекту диктують проєктування та керування інтелектуальними роботизованими системами. Від сприйняття роботом навколишньої середи до планування руху, система ШІ глибоко інтегрується в робототехніку, розкриваючи потенціал для безперебійної взаємодії автономних систем із фізичним світом.

1.3.3 Комп’ютерний зір

Одна з теорій та технологій заснованих на штучному інтелекті це комп’ютерний зір. Задача комп’ютерного зору або комп’ютерного бачення полягає у створенні машин, які здатні класифікувати та розпізнавати об’єкти

на основі вхідного зображення цифрового формату. Елементами цифрового зображення є пікселі. Кожен піксель має свою координату та числове значення. Виходячи з цього, цифрове зображення – це багатовимірний масив чисел який зберігається у комп’ютерній пам’яті.

На вхід системі комп’ютерного бачення може подаватись відеопослідовність, дво- або тривимірні зображення з однієї або декількох камер. Для розуміння візуальної інформації, комп’ютерне бачення може виконувати наступні процеси:

– отримання зображень;
– початкова обробка;
– сегментація;
– ідентифікація особливостей зображень;
– виявлення об’єктів;
– розпізнавання об’єктів;
– прийняття рішень.

Процес отримання зображень полягає у захоплені візуальних даних з різних джерел, таких, як камери, сенсори або набори даних.

Початкова обробка передбачає покращення якості зображення, зміна розміру, нормалізація, колірна корекція.

Процес сегментації виконує розподіл зображення на окремі частинки засновуючись на таких критеріях як колір, текстура або інтенсивність.

Ідентифікація особливостей зображень характеризувати об’єкти. Особливостями зображень можуть бути краї, кути, текстури чи інші характерні елементи.

Процес виявлення об’єктів відповідає своїй назві.

Процес розпізнавання детальніший ніж процес виявлення об’єктів. Цей процес ідентифікує об’єкти. Тут передбачається зіставлення витягнутих функцій із заздалегідь визначеними шаблонами або використання алгоритмів машинного навчання для класифікації.

Процес прийняття рішень передбачає використання інформації,

отриманої з попередніх процесів, для прийняття рішень або виконання дій. Цей процес застосовується в таких додатках, як автономні системи та робототехніка.

1.3.4 MTCNN

При застосуванні системи розпізнавання обличчя необхідно визначитись з її кореневою частиною. Система розпізнавання обличчя може базуватись на моделі MTCNN. Розроблена як алгоритм для виявлення, розпізнавання обличчя у безперешкодних умовах, MTCNN дозволяє створювати системи комп'ютерного зору з великою точністю сприйняття навколишньої середи [7]. MTCNN (Multi-task Cascaded Convolutional Neural Networks) створена з урахуванням значних варіацій та оклюзій.

MTCNN має наступний алгоритм. Коли надходить зображення, спочатку змінюється розмір у різних масштабах для побудови піраміди візуальних даних. Наступним кроком сформовані дані віддаються трирівневій обробці.

На першому рівні залучена конвуляційна мережа, а саме мережа пропозицій P-Net (proposal network). Ця мережа розбиває вхідне зображення на багато частин різного розміру. На основі отриманих частин обчислюються вектори регресії крайових місць зображення [7]. Далі використовуються оцінені вектори регресії крайових місць для калібрування усіх частин зображення. Ці частини також можна назвати кандидатами. Після цього використовується немаксимальне придушення, щоб об'єднати кандидатів які сильно перекриваються [7].

Далі отримані кандидати передаються іншій конвуляційній мережі, яка називається Уточнювальна мережа (Refine Network) R-Net. Ця мережа відхиляє кандидатів які втратили сенс, виконує калібрування з регресією крайових частин та об'єднує частини-кандидати.

На останньому третьому етапі виконуються схожі дії для більш детальної обробки даних зображення. На цьому етапі залучається вихідна мережа (Output Network, O-Net).

Над моделлю MTCNN проводились експерименти, які показали задовільні результати [7].

1.4 Висновки до першого розділу

На етапі вивчення предметної галузі було вивчено особливості надання доступу у комп’ютерних системах. Користувач може запам’ятати складний пароль для входу та бути впевненим у надійності. Складні паролі дійсно можуть підвищити безпеку захищених даних та ресурсів.

Однак крім штучних ключів є природні ідентифікатори якими наділена кожна людина. Якщо оцифрувати відбитки пальців, особливості обличчя, ДНК або інші природні характеристики, можна визначити, що отримані дані настільки унікальні, що їх штучне дублювання не може бути можливим.

Такі обставини можуть гарантувати можливість створення досить надійних інформаційних систем. З іншого боку оцифровка природних ідентифікаторів являє собою доволі складний, а іноді неможливий процес.

Технологія штучного інтелекту дозволяє створювати надійні системи ідентифікації на основі біометричних даних. При цьому надійні системи на основі ШІ відрізняються деякою складністю та потребують ретельного тестування на кожному рівні розробки.

2 ІНСТРУМЕНТАРІЙ РОЗРОБКИ

2.1 Інструментарій розробки сервера

Серверна частина повинна швидко реагувати на запити від клієнта, формувати та відправляти належні відповіді. Важливим фактором у роботі систем з клієнт-серверною архітектурою є затримка між запитом та відповіддю. Якщо сервер дуже довго формує відповіді, він може бути непридатним, та звести нанівець усю систему. У реальних проєктах сервер обробляє велику кількість одночасних запитів. Множина запитів обробляється сервером, який не залишає без уваги жодне звернення. Коли надходить запит, сервер повинен активізувати логіку, яка виконує всі необхідні функції з найменшою кількістю елементарних дій. Саме такий сервер буде сприятливим для клієнтів та придатним для системи в цілому.

Коли сервер стає доступним для великої множини клієнтів, виникає загроза конфіденційності даних. Конфігурація сервера повинна вказувати на використання криптографічних алгоритмів. Доступ до бази даних й інших ресурсів слід налаштовувати належним чином.

У поточному проєкті сервер активізовує технологію штучного інтелекту, яка своєю чергою виконує складну задачу розпізнавання обличчя. У запитах до сервера містяться об’ємні дані, які слід обробляти дуже ретельно.

Для того, щоб підібрати доцільні інструменти розробки серверної частини потрібно спиратися на вищезазначені міркування.

2.1.1 Операційна система Ubuntu

На етапі створення серверної частини перш за все необхідно обрати операційну систему. Виконуючи роль основного рушія, операційна система

надає відповідні ресурси апаратного забезпечення для належної роботи стека програм. Перед тим, як запит надійде до цільової програми, він потрапляє в абстракцію операційної системи. Операційна система (ОС) створює основну середу для кожної програми, які відповідають за обробку запитів на певних етапах. Коли необхідно щось змінити, налаштувати, або видалити на сервері, адміністратор використовує засоби ОС.

Обираючи операційну систему для сервера керуються такими характеристиками:

– підтримка необхідного програмного забезпечення;
– стабільність і безпека;
– легкість у використанні;
– підтримка хмарних сервісів.

Вищезазначені міркування зробили вибір на користь операційної системи Ubuntu. Комплекс корисних програм синтезованих у дистрибутив Ubuntu, представляє флагманський проєкт компанії Canonical. Ubuntu заснована на дистрибутиві Debian, та розповсюджується як вільне програмне забезпечення з відкритим початковим кодом. Це дозволяє її використовувати безкоштовно та аналізувати кожну особливість ОС.

В ОС Ubuntu вбудовано декілька систем управління пакетами, що дозволяє розширювати функціональність, використовувати необхідні програми та інструменти.

В екосистемі Ubuntu відбуваються регулярні випуски оновлень та патчів, які допомагають у підтримці високого рівня безпеки. Спільнота Ubuntu активно працює над вдосконаленням своєї системи, та швидко реагує на виявлені вразливості. Ubuntu використовує механізм sudo для адміністрування системи, уникаючи облікового запису root. Оновлення безпеки гарантуються щонайменше на п’ять років для рекомендованих випусків LTS і є безкоштовними.

Один рядок командної строки може задати необхідні інструкції з налаштування системи. То ж Ubuntu доволі легка у використанні.

Ubuntu взаємодіє з різними хмарними платформами, такими як Amazon Web Services (AWS) і Microsoft Azure. Це полегшує розгортання та управління віртуальними серверами в хмарному середовищі.

2.1.2 Обслуговування запитів за допомогою вебсервера Nginx

Клієнт-серверна архітектура розробленого проєкту передбачає взаємодію вузлів системи. Логіка цієї взаємодії специфікується протоколом HTTP/S. Тому на стороні сервера повинна бути спеціальна програма – вебсервер, який керує прийманням запитів та видачою відповідей. Оберемо вебсервер, який відповідає наступним вимогам:

– ефективність та продуктивність роботи;
– підтримка протоколів TLS/SSL;
– легка конфігурація;
– модульність;
– розвинена спільнота й підтримка.

Nginx є одним із найбільш широко використовуваних вебсерверів, доступних сьогодні, частково, через його можливості балансування навантаження та зворотного проксі-сервера для HTTP та інших мережевих протоколів.

Nginx спроєктований для ефективної обробки множини одночасних з’єднань, що робить його ідеальним для високозавантажених вебсайтів та додатків. Він ефективно витрачає ресурси та має низьку витрату пам’яті.

В основі Nginx лежить асинхронна архітектура, що дозволяє обробляти багато одночасних з’єднань без створення окремого потоку для кожного з’єднання.

Nginx підтримує протоколи TLS/SSL, беручи на себе розшифровку шифрованого трафіку.

Конфігурація Nginx відносно проста, що сприяє швидкому

впровадженню та обслуговуванню.

Nginx побудований на модульній архітектурі, що дозволяє динамічно додавати та відключати функціональність за допомогою модулів.

Широка спільнота користувачів та підтримка з боку керуючої компанії дозволяють отримати відповіді на можливі питання та надають вичерпну документацію.

Одна з характерних функцій Nginx це розподілення навантаження [8]. Інформаційні системи, які працюють в інтернет-просторі повинні показувати високу продуктивність та безвідмовність роботи. Щоб задовольнити цю потребу створюється кілька копій однієї системи, і навантаження розподіляється між ними. В міру збільшення навантаження інша копія системи може бути виведена в режим онлайн. Така архітектурна техніка має назву горизонтальне масштабування [8]. В контексті цієї архітектури існує потреба балансування навантаження на інформаційну систему. Nginx задовольняє цю потребу декількома засобами, наприклад за допомогою HTTP, TCP та UDP [8]. Функція розподілення навантаження робить використання Nginx доцільним у складних системах які мають справу з великими даними.

2.1.3 Інтеграція uWSGI до Nginx

Серверна частина запрограмована на мові Python, тому до стека програм сервера входить uWSGI. Робота вебсервера uWSGI дозволяє розгортати серверний додаток на мові Python. uWSGI підтримує різні протоколи, такі як HTTP, FastCGI, WebSocket, і є універсальним інтерфейсом для спілкування між вебсервером і серверним додатком. Архітектура uWSGI спроєктована так, щоб обробляти багато одночасних з’єднань асинхронно, це поліпшує продуктивність і ефективність. uWSGI наділений здатністю масштабування, що дозволяє використовувати його для розгортання значної

множини одночасних з'єднань та обробки великого трафіку.

Інтеграція uWSGI з Nginx дозволяє використовувати переваги обох вебсерверів, надаючи швидку й ефективну платформу для розгортання Python-додатків у високопродуктивному середовищі. Серверна конфігурація з інтегрованим uWSGI представлена на рисунку 2.1.

Рисунок 2.1 – Сервер з uWSGI інтегрований до Nginx

На зображені видно, що з'єднання між Nginx та uWSGI відбувається за допомогою абстракції сокета Юнікс. uWSGI з'єднується з додатком за допомогою викликаючого об'єкту. Такий ланцюг з'єднання дозволяє стеку програм сервера виконуватись належним чином.

2.1.4 Мова програмування Python

Додаток сервера повинен залучити технологію машинного навчання для розпізнавання обличчя. Сучасні мови програмування здатні інтерпретувати будь-яку логіку включаючи інструкції пов'язані з технологією машинного навчання та штучного інтелекту. Однак не всі формальні мови мають лаконічну семантику та набір бібліотек машинного навчання (МН).

Проаналізувавши теперішню кон'юнктуру мов програмування можна прийняти об'єктивне рішення використовувати Python для реалізації

серверного додатка.

Python відрізняється лаконічним синтаксисом від інших мов програмування. Python має величезну екосистему з бібліотеками та фреймворками, що дозволяє використовувати цю мову в різних проєктах включаючи технології МН. До арсеналу Python належать такі бібліотеки: TensorFlow, PyTorch, scikit-learn. Вони дозволяють використовувати потужні інструменти для створення моделей ШІ та піддавати їх процесу машинного навчання.

Python має активну спільноту розробників, що сприяє швидкому вирішенню проблем й надає багато ресурсів для вивчення та підтримки.

2.1.5 Мікрофреймворк Flask

Для того, щоб організувати зручну архітектуру серверного додатка був обраний мікрофреймворк Flask. Гнучкий та простий у користуванні інструмент дозволяє легко будувати серверні додатки та вебсайт.

Серед основних характеристик цього мікрофреймворку можна привести наступне:

– легковаговий;
– простий;
– модульний;
– здатний легко розширювати функціональність;
– вбудована підтримка юніт-тестування.

У мікрофреймворк не входить рівень абстракції бази даних. Однак Flask був розроблений таким чином, щоб до нього можна було легко підключати необхідні компоненти, у тому числі системи керування базами даних.

Побудова серверного додатка була значно спростована завдяки каркасу Flask.

2.1.6 Бібліотека машинного навчання TensorFlow

На етапі розробки проєкту, необхідно проаналізувати особливості технології машинного навчання, та розпізнавання обличчя. Важливим етапом є визначення інструментів, за допомогою яких відбувається розпізнавання обличчя. Серед різних технологій обираємо бібліотеку TensorFlow.

TensorFlow – це бібліотека чисельної обробки, яку використовують дослідники та фахівці дисципліни машинного навчання. Хоча за допомогою TensorFlow можна виконувати будь-які числові операції, бібліотека здебільшого використовується для навчання та запуску глибинних нейронних мереж [9].

TensorFlow насамперед пропонує спростити розгортання рішень машинного навчання на різних платформах та компонентах – центральних, графічних процесорах, мобільних пристроях, а також в браузерах [9].

У TensorFlow реалізовано декілька рівнів абстракції. На рисунку 2.2 зображена архітектура TensorFlow.

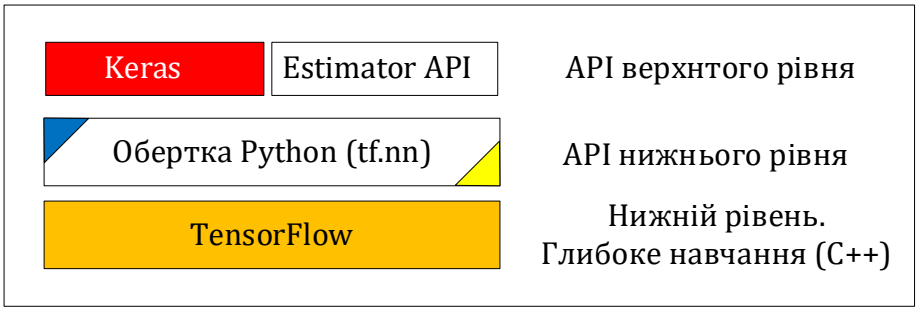

Рисунок 2.2 – Архітектура TensorFlow

На самому початку архітектури розташована логіка, яка запрограмована на мові C++. Далі йде рівень на Python, що обгортає логіку C++.

У верхній частині архітектури розташовані два компоненти, Keras та Estimate API.

Keras – це високорівневий інтерфейс для машинного навчання, що

дозволяє швидко та ефективно створювати, навчати та розгортати моделі глибокого навчання. Основна мета Keras – надати простий та інтуїтивний інтерфейс для роботи з нейронними мережами, що дозволяє швидко експериментувати з різними моделями та архітектурами.

Estimator API – це високорівневий інтерфейс для створення, навчання та оцінювання моделей машинного навчання. Цей інтерфейс надає спрощений шлях для розробки різноманітних моделей та використовується для стандартизації роботи з моделями у фреймворку TensorFlow. Estimator API дозволяє легко реалізовувати власні моделі та використовувати готові алгоритми, роблячи процес розробки та навчання більш ефективним та структурованим.

Така архітектура дозволяє реалізовувати необхідний функціонал.

Завдяки своїй здатності ефективно обробляти великі об’єми даних та тренувати складні моделі нейронних мереж, TensorFlow здобув велику популярність. У контексті комп’ютерного зору, TensorFlow використовується для створення потужних моделей, які здатні визначати та розпізнавати обличчя. Також важливою вимогою є здатність системи розпізнавання обличчя виконувати свої дії за дуже короткий час. Клієнт не повинен бачити значну затримку під час авторизації.

2.2 Інструментарій розробки клієнта

Окрім серверної частини система передбачає клієнтський додаток, який встановлюється на мобільний телефон під керуванням ОС Android. На відміну від серверної програми, кількість клієнтських додатків обмежується здатністю сервера обробляти запити. В ідеальних умовах система з клієнт-серверною архітектурою не повинна накладати обмеження на кількість клієнтів. Розроблений мобільний додаток повинен бути легким у використанні та задовольняти різним екранам можливих пристроїв.

Основна задача клієнтського додатка полягає у формуванні запитів, прийманні відповідей та оновленні екрана в залежності від отриманих даних. Інтерфейс додатка повинен бути ергономічним, не містити зайвих, неналежних елементів керування. Перед розробкою клієнта було проаналізовано всі його задачі та сформульовано необхідний інструментарій розробки.

2.2.1 Операційна система Android

Android – відносно нова операційна система, призначена для роботи на мобільних пристроях. Заснована на ядрі Linux та налаштована на такі особливості мобільних пристроїв, як сенсорний екран, акселерометр, гіроскоп та інше. Основна частина системи Android написана на Java, а логіка деякої частини ядра та бібліотек написані на C та C++.

Однією з ключових особливостей Android є відкритий початковий код, що сприяє активній співпраці розробників з усього світу. Представляючи частину проєкта Android Open Source Project (AOSP), операційна система надає можливість розробникам створювати власні модифікації та власні версії ОС для різних пристроїв.

Дизайн системи Android значною мірою обертається навколо ізоляції процесів між програмами, а також між різними частинами самої системи. Це викликає необхідність у міжпроцесному зв’язку для координації дій між різними процесами. Тому Android має багатофункціональний засіб міжпроцесного зв’язку під назвою Android Binder.

Архітектура Binder розділена на три рівні. У нижній частині стека знаходиться модуль ядра, який реалізує фактичну міжпроцесну взаємодію та надає її через функцію ioctl ядра.

ioctl – це виклик ядра загального призначення для надсилання спеціальних команд до драйверів і модулів ядра. На вершині модуля ядра є

базовий об’єктноорієнтований API простору користувача, який дозволяє програмам створювати кінцеві точки IPC і взаємодіяти з ними через типи IBinder і Binder. У верхній частині знаходиться модель програмування на основі інтерфейсу, де додатки декларують свої інтерфейси IPC, і їм не потрібно турбуватися про деталі того, як відбувається IPC на нижчих рівнях.

Значний внесок в поширення Android зробили виробники смартфонів, які вибрали цю операційну систему для своїх пристроїв. Багато відомих компаній, таких як Samsung, LG, HTC, та інші, випускають пристрої, що працюють під управлінням Android. Широкий вибір пристроїв різних цінових категорій дозволяє користувачам знаходити оптимальне рішення для своїх потреб та бюджету.

Окрім широкого вибору пристроїв, Android пропонує інтеграцію з великою кількістю сервісів Google. Від Gmail і Google Maps до Google Drive та Google Assistant – це надає користувачам зручний доступ до різноманітних інструментів і сервісів.

2.2.2 Мова програмування Java

Як мову програмування мобільного додатка клієнта було вирішено використовувати Java. Об’єктно-орієнтованість мови та наявність необхідного функціоналу в екосистемі Android дозволяє належним чином будувати продуктивні додатки.

Java, розроблена компанією Sun Microsystems (згодом придбаною Oracle Corporation), є однією з найбільш універсальних та популярних мов програмування у світі інформаційних технологій. Започаткована в 1995 році, Java швидко завоювала популярність завдяки своїй переносності, надійності та розширюваності.

Мова програмування Java відома своєю платформонезалежністю. Програми, написані на Java, можуть виконуватися на будь-якій віртуальній

машині Java (JVM), що робить їх переносними між різними операційними системами. Це дозволяє розробникам створити програму один раз і запускати будь-де. Тому Java також використовується для розробки кросплатформних додатків.

Однією з ключових особливостей Java є велика кількість бібліотек та фреймворків. Java Standard Edition (SE) та Java Enterprise Edition (EE) мають широкий спектр інструментів, які полегшують розробку від малих додатків до великих підприємницьких рішень.

Java також славиться своєю надійністю та безпекою. Механізм збирання сміття (garbage collection) допомагає автоматично керувати пам’яттю.

Java залишається однією з найпопулярніших мов програмування завдяки своїй переносності, розширюваності, надійності та різноманітній екосистемі, яка надає розробникам всі необхідні інструменти для творчості та інновацій.

2.2.3 Бібліотека комп’ютерного зору OpenCV

Клієнтський додаток виконує передачу зображення на сервер у контексті комп’ютерного зору. Тому для автентифікації на основі розпізнавання обличчя залучена бібліотека OpenCV.

OpenCV, або Open Source Computer Vision Library, є потужним інструментом для розробки комп’ютерного зору та обробки зображень. Заснована на відкритому початковому коді, OpenCV надає розробникам значну кількість функцій та алгоритмів для роботи з зображеннями, відео та потоковими даними.

Бібліотека написана мовою програмування C++. Вона має інтерфейси для більшості популярних мов програмування, включаючи, але не обмежуючись C/C++, Python та Java. OpenCV працює на різних ОС,

включаючи Windows, Android, Linux, macOS та інших Unix-подібних системах.

Бібліотека містить понад 2500 оптимізованих алгоритмів для завдань машинного навчання та комп'ютерного зору. Спільнота OpenCV налічує понад 47 000 професіоналів з комп'ютерного зору [10]. OpenCV широко використовується в наукових установах для навчання, дослідницьких, урядових організаціях і в різних сегментах промисловості. Відомі організації, такі як Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda та Toyota, використовують OpenCV [10].

OpenCV має вбудовані інструменти для розпізнавання обличчя, виявлення ключових точок та використання алгоритмів ідентифікації особливостей.

OpenCV легко інтегрується з іншими технологіями та бібліотеками. Наприклад, можна використовувати TensorFlow для глибшого розпізнавання об'єктів.

Важливим аспектом використання OpenCV є оптимізація для мобільних платформ. Існують методи оптимізації, які дозволяють прискорити обчислення та забезпечити ефективну роботу навіть на пристроях з обмеженими ресурсами.

2.2.4 Бібліотека Retrofit для реалізації логіки запитів

Клієнт активно відправляє HTTP запити, та отримає відповіді від сервера. Для створення логіки HTTP комунікацій клієнтська частина використовує бібліотеку Retrofit.

Бібліотека Retrofit перетворює дані із контексту REST API в інтерфейси Java. Для цього необхідно використовувати анотації для опису окремих кінцевих точок API та їхніх HTTP-запитів.

Підтримка заміни параметрів URL-адреси (наприклад, запиту та параметра шляху) інтегрована за замовчуванням, а також функціональні можливості для запитів із кодуванням форми та багатокомпонентних запитів.

Серед переваг використання Retrofit слід зазначити:

– спростування початкового коду;
– виконання запитів в асинхронному форматі;
– підтримка різних форматів даних.

Інтеграція з анотаціями дозволяє зменшити кількість коду, що підвищує читабельність та підтримку.

Retrofit автоматично використовує асинхронний підхід для виконання мережевих запитів, запобігаючи блокуванню основного потоку та покращуючи інтерактивність додатка.

Вбудована підтримка для різних форматів обміну даними, таких як JSON та XML, робить Retrofit універсальним інструментом для взаємодії з різноманітними серверами.

2.2.5 Бібліотека Moshi для перетворення типів

Активно взаємодіючи із сервером, додаток має справу з даними, представленими форматом JSON. Для того, щоб перетворити дані JSON у систему типів Java використовується бібліотека Moshi.

Розроблена компанією Square бібліотека Moshi доволі ефективна і легка у використанні. Можна також привести інші переваги цієї бібліотеки:

– автоматичне визначення типу;
– null-safety;
– значна швидкість.

Moshi може автоматично визначати типи даних, що забезпечує легку інтеграцію та уникнення додаткового коду для явного вказання типів.

2.3 Висновки до другого розділу

Підбір інструментарію розробки клієнтської та серверної частини дав змогу зрозуміти, як треба підійти до розробки програмної системи, та спланувати час на реалізацію кожного компонента. Деякі інструменти визначають характеристики майбутньої програми.

Популярність та підтримка того, чи іншого інструмента визначає доцільність його використання. Наприклад, якщо мова програмування має вичерпну документацію та широко використовується у промислових системах можна легко дізнатись про її особливості, та отримати відповіді на неоднозначні питання.

3 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

3.1 Опис проєкта

Після аналізу предметної галузі та інструментарію розпочнемо розробку самого проєкту.

На початку реалізації була створена діаграма класів та візуальне уявлення того, що собою представляє сервер, його структура.

Основна логіка системи виконується на сервері. Наявність серверної частини централізує систему та визначає задоволеність клієнтів. Сервер зберігає базу даних клієнтів. У серверний додаток інтегрована система керування базами даних SQLite. Тому представлення даних базується на реляційній моделі. Додаток сервера реалізує моделі для машинного навчання засновані на нейронних мережах MTCNN.

Після того, як загальна концепція сервера була створена, почнемо проєктування клієнтського додатка.

Клієнтська сторона – це мобільний додаток, який виконується під керуванням ОС Андроїд.

Потім було вирішено побудувати візуальне представлення клієнта. Клієнт має справу з комп’ютерним зором, тому використовує бібліотеку OpenCV. Бібліотека дозволяє інтерпретувати зображення у вигляді необхідному для розпізнавання обличчя. Візуальне представлення клієнта складається з діаграми класів та ілюстрації компонентів, які у своїй сукупності формують те, що люди сприймають мобільним додатком.

3.2 Загальна архітектура

Розроблена система має просту архітектуру клієнт-сервер. Один сервер обслуговує багатьох клієнтів. Тому в системі розгорнутий тільки один

екземпляр серверного додатка. Екземплярів клієнтського додатка в системі може бути довільно. На рисунку 3.1 зображено загальну архітектуру розробленого проєкту.

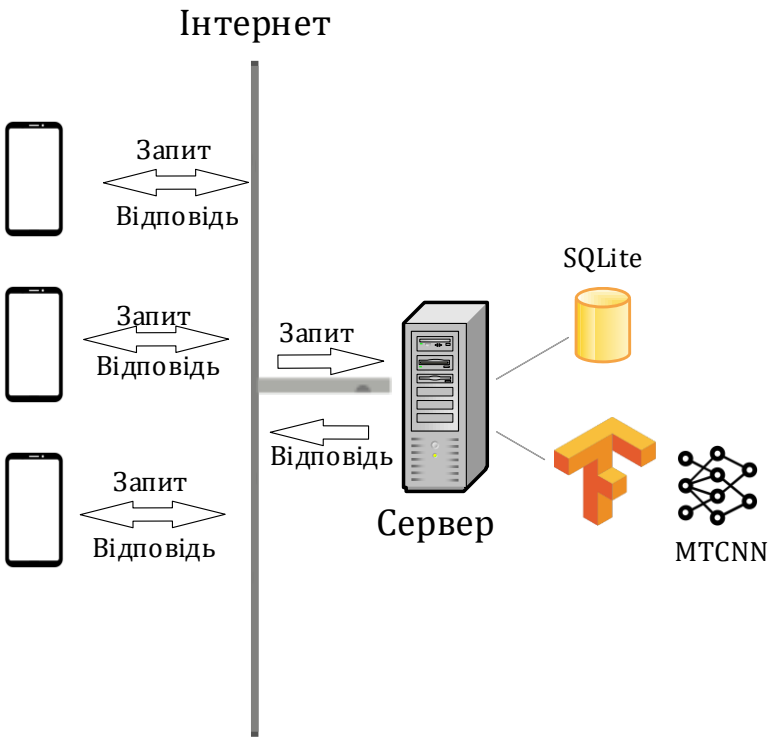

Рисунок 3.1 – Загальна архітектура системи

Запити від клієнтів посилаються до сервера, який, взаємодіючи з TensorFlow, формує відповіді.

У конфігурації ОС сервера можна визначити його пропускну здатність. Командний рядок сервера дозволяє змінити, або проаналізувати його конфігурацію. Наприклад, використавши утиліту ifconfig можна визначити найбільший розмір пакета даних переданого через мережевий інтерфейс без фрагментацій.

Наступною є команда, яка демонструє використання ifconfig: *sudo ifconfig eth0 mtu 1500.*

Ця команда задає границі розміру пакета даних до 1500 байтів.

Клієнт-серверна архітектура дозволяє легко визначати особливості взаємодії вузлів та впроваджувати налаштування належним чином.

3.3 Серверна частина

Сервер складається з багатьох частин, які дозволяють надавати клієнтам належні відповіді без зайвих затримок. Логіка роботи сервера має справу з достатньо об’ємними даними, які обробляються нейронними мережами в декілька етапів. У наступних підпунктах розглядається архітектура серверної частини, пояснюється, як саме сервер виконує свою роботу.

3.3.1 Архітектура сервера

Коли відбувається розробка програмного забезпечення, яке здається складним, у першу чергу необхідно спроєктувати архітектуру.

Створення архітектури для поточної системи було дуже важливим тому, що це визначало успішність подальшої реалізації та розуміння проєкту.

На рисунку 3.2 відображається архітектура всього сервера.

Сервер працює під керуванням ОС Ubuntu, з якою взаємодіють інші програми необхідні для належної роботи всієї системи. Приймаючи запит операційна система сповіщає вебсервер Nginx. Вебсервер повинен швидко опрацьовувати велику кількість запитів від клієнтів. Nginx працює на рівні протоколу HTTP.

У конфігурації сервера також є прошарок uWSGI необхідний для адаптування мережевої взаємодії під мову програмування Python. uWSGI безпосередньо зв’язаний з екземпляром мікрофреймворка Flask, який повертає логіка серверного додатка. Екземпляр Flask інкапсулює всі інструкції, які були закладені розробником серверної частини. Для того, щоб виконувати розпізнавання обличчя екземпляр додатка взаємодіє з нейронними мережами через бібліотеку TensorFlow.

Рисунок 3.2 – Архітектура сервера

Проілюстрована композиція веде логічний потік призначеним шляхом, що задовольняє потребам розробника.

3.3.2 Особливості серверного додатка

Серверний додаток складається з двох основних модулів: mod_facerecognition та mod_face_utils. Структура модулів зображена на рисунку 3.3.

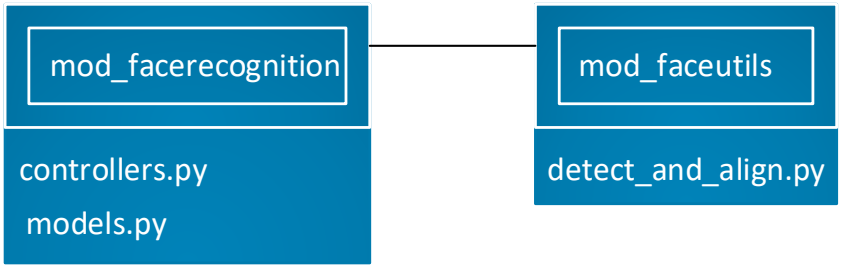

Рисунок 3.3 – Структура серверного додатка

У модулі mod_facerecognition визначається база даних та контролер, де виконуються основні функції, включаючи автентифікацію, зберігання користувача у базі, та інше.

На рисунку 3.4 відображається діаграма контролера.

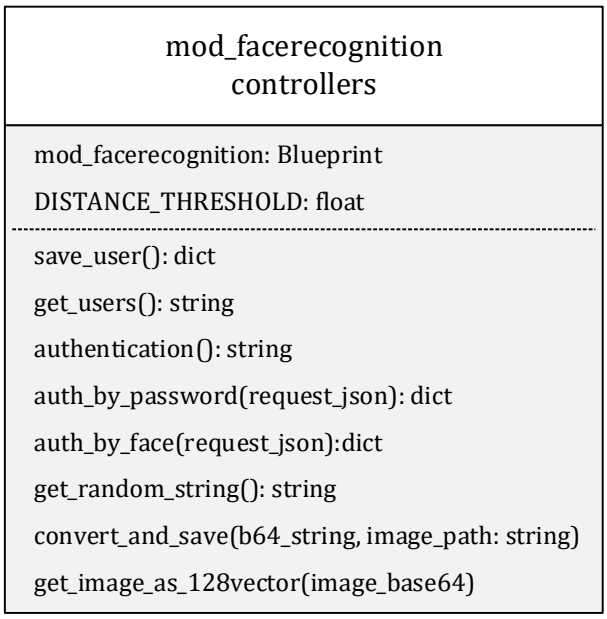

Рисунок 3.4 – Діаграма контролерів модуля mod_facerecognition

Основними функціями контролера цього модуля є: authentication(), auth_by_password(), auth_by_face(). На рисунку 3.5 зображено початковий код функції authentication().

```
@mod_facerecognition.route('/authenticate_user/', methods=['POST'])
def authentication():

    status = False
    response_message = "Unknown authentication request! " \
                       "Use either username and password OR username and face."

    request_json = request.get_json()

    if 'image' in request_json and 'username' in request_json:
        data = auth_by_face(request_json)
    elif 'password' in request_json and 'username' in request_json:
        data = auth_by_password(request_json)
    else:
        data = {"success": status, "message": response_message, "user": None}

    return jsonify(data)
```

Рисунок 3.5 – Функція authentication

Всередині функції перевіряється яким чином клієнт намагається автентифікуватись: через пароль або через зображення зі своїм обличчям. В залежності від цього викликається функція auth_by_password() або

auth_by_face(). Ці функції повертають словник «data» з відповіддю про успіх чи невдачу. Далі словник «data» перетворюється у формат JSON та повертається із поточної функції для подальшого відправлення сформованої відповіді клієнту.

До модуля mod_facerecognition також належить модель, яка специфікує користувача для серверного додатка та його бази даних. Діаграма класу User продемонстрована на рисунку 3.6.

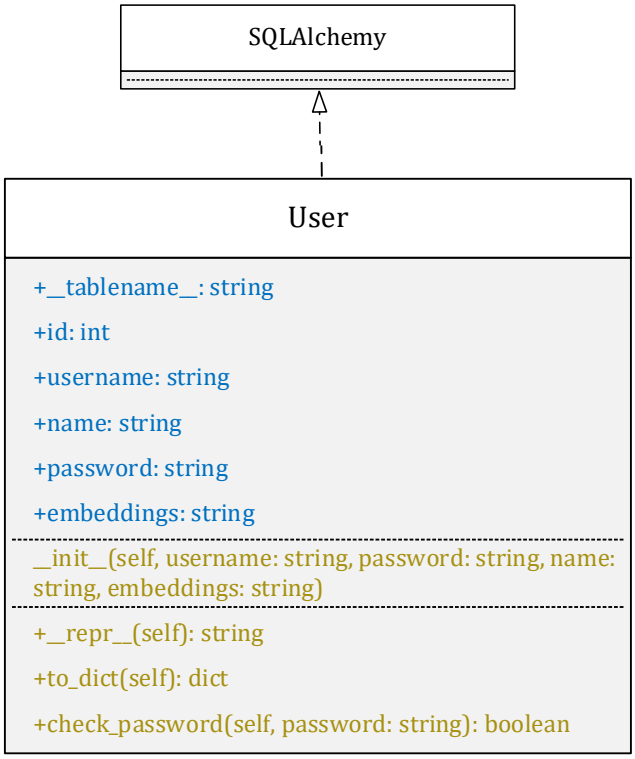

Рисунок 3.6 – Діаграма класу User

За допомогою цього типу можна легко записувати або видаляти користувачів з бази даних.

3.3.3 Логіка розпізнавання обличчя

Функція auth_by_face(), яка зустрічалась у попередньому підрозділі, веде програму до логіки розпізнавання обличчя. Цей шлях відображається на рисунку 3.7.

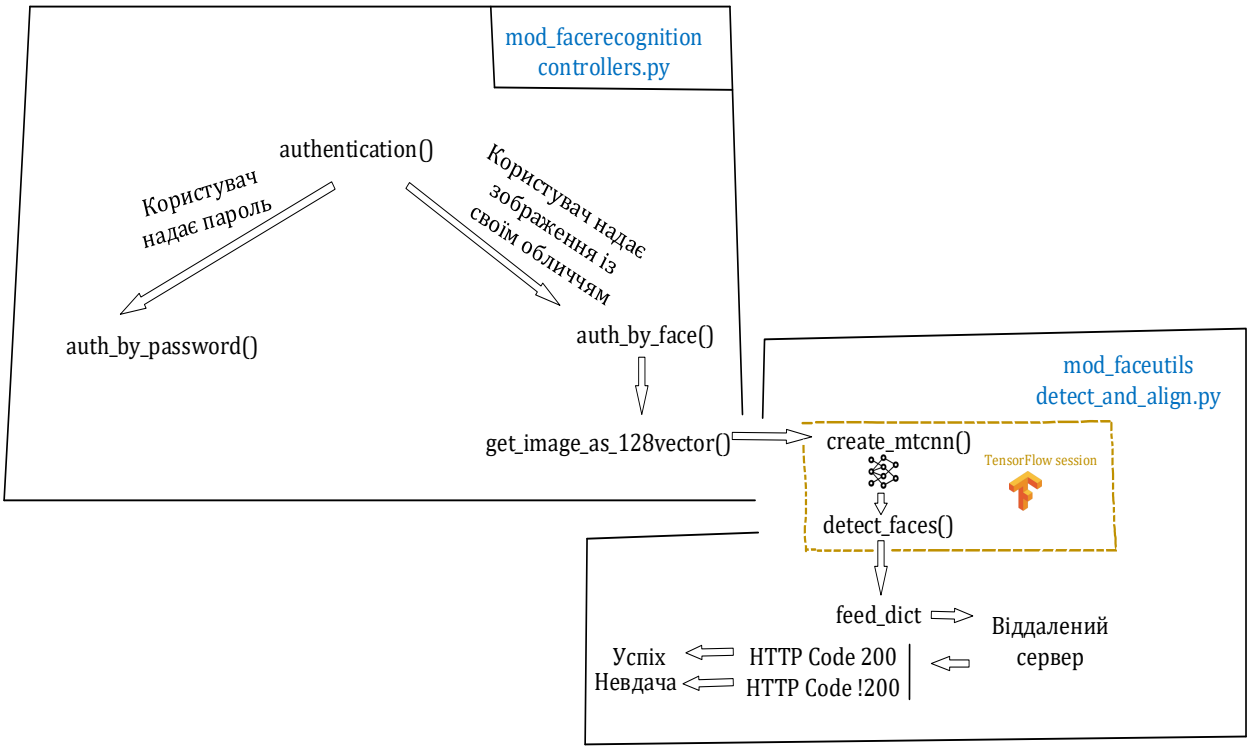

Рисунок 3.7 – Шлях залучення логіки розпізнавання обличчя

Функція auth_by_face() викликає get_image_as_128vector(), яка створює сесію TensorFlow. На момент існування сесії створюється нейронна мережа MTCNN.

Реалізуючи трирівневу структуру мережі, логіка створює екземпляри PNet, RNet, ONet.

Після того, як MTCNN готова, викликається функція detect_faces() для обробки зображення та виявлення облич у ньому.

Наприкінці своєї роботи detect_faces() повертає:

– face_patches – список нормалізованих патчів облич, готових для подальшої обробки;
– padded_bounding_boxes – множину крайових місць зображення;
– landmarks – колекцію ознак для кожного виявленого обличчя.

Після цього формується словник, який містить необхідні дані для подальшої обробки. Ці дані обробляються віддаленим сервером, який повертає остаточний результат про успіх або невдачу авторизації за допомогою технології комп’ютерного зору.

3.4 Клієнтська частина

Клієнтська програма являє собою невід'ємну частину проєкту. Клієнтська частина втілена у Андроїд додаток, який взаємодіє із сервером, має графічний інтерфейс користувача.

Для реалізації клієнта необхідно було створити логіку мережевої роботи, та відповідний інтерфейс користувача.

Перш за все була розроблена архітектура клієнтського додатка. Архітектура передбачує зв'язок між екранами. Кожен екран має відображати певні віджети для реєстрації, входу через пароль, або через розпізнавання обличчя.

Також на стороні клієнта має бути деяке представлення про користувача, який реєструється, або пройшов реєстрацію та входить під своїми реквізитами доступу.

Всі компоненти клієнтського додатка повинні узгоджено працювати, підлаштовуватись до змін конфігурації та мати коротку затримку в роботі.

3.4.1 Архітектура клієнта

Перш за все було вирішено створити архітектуру клієнтського додатка. Уявлення архітектури у вигляді діаграм та інших зображень дозволяє раціонально подивитись на всю програму наповнюючи її необхідним функціоналом.

Як архітектурний патерн використовувався MVVM (Model View ViewModel). На рисунку 3.8 зображена схема патерну MVVM.

Архітектурний компонент Model містить дані про користувача. Model також може описувати базу даних, але у поточному додатку це не має сенсу.

Компонент ViewModel виконує функцію прошарку між View та Model.

Основна робота ViewModel це підготовка даних у той вид, який буде використовуватись у View.

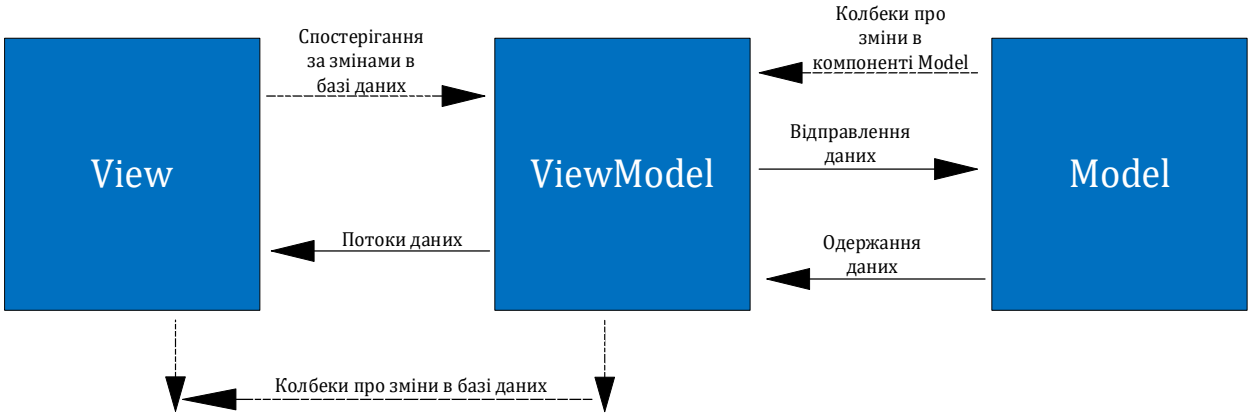

Рисунок 3.8 – Архітектурний патерн MVVM

Архітектурний компонент View являє собою верхній рівень додатка. На цьому рівні розташовується представлення графічної частини та її логіки. Це верхній рівень, який керує іншими рівнями, дозволяючи узгодити всю роботу клієнта.

Взаємодія між View та ViewModel заснована на патерні спостерігач (observer). Логіка View підписується до ViewModel. Якщо дані у Model змінюються, компонент ViewModel одразу направляє повідомлення до View. Отримавши повідомлення логіка View оновлює графічний інтерфейс належним чином.

Рисунок 3.9 дає більш детальне уявлення про архітектуру додатка.

Рівень Model архітектури додатка презентує зареєстрований користувач, або користувач, який пройшов авторизацію. Ці два типи користувачів специфікуються класами RegisteredUser та LoggedInUser. Діаграма цих класів зображена на рисунку 3.10.

Ці класи ідентичні один одному. Інкапсуляція зареєстрованого та авторизованого користувачів у різні класи дозволяє чітко їх розрізнити на етапі звернення.

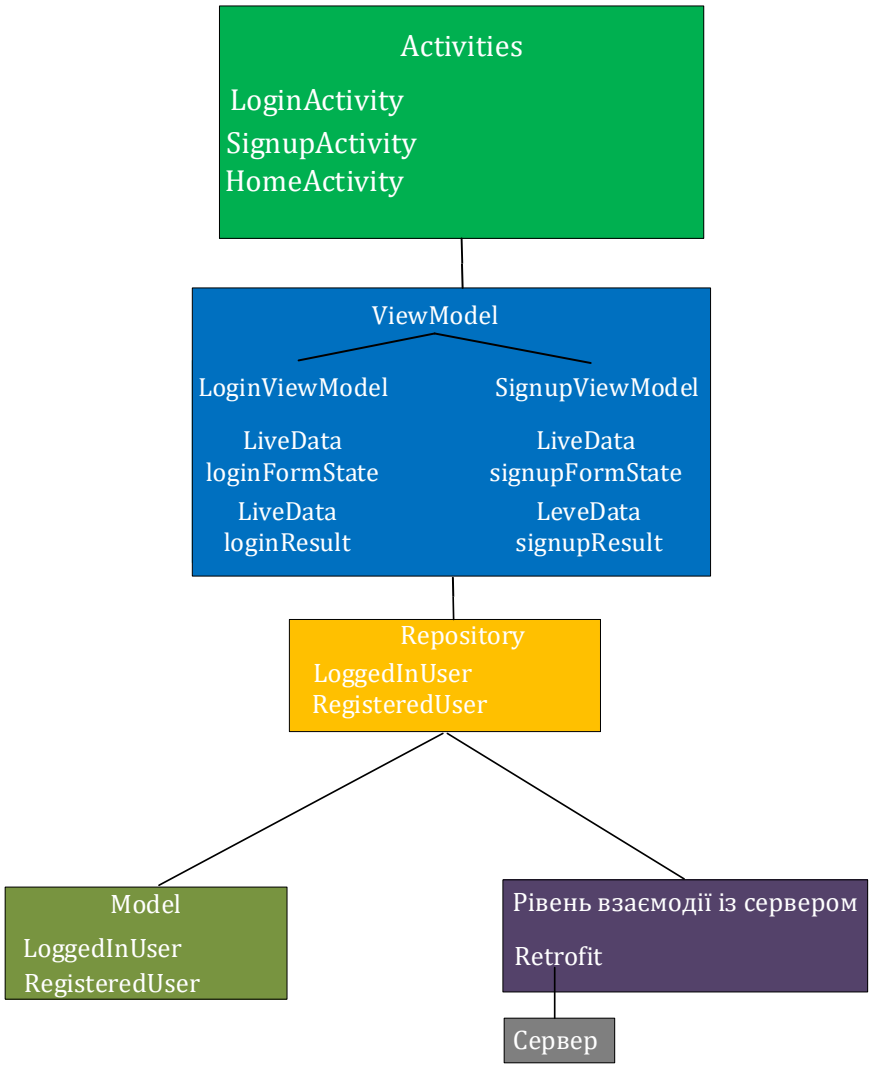

Рисунок 3.9 – Архітектура додатка

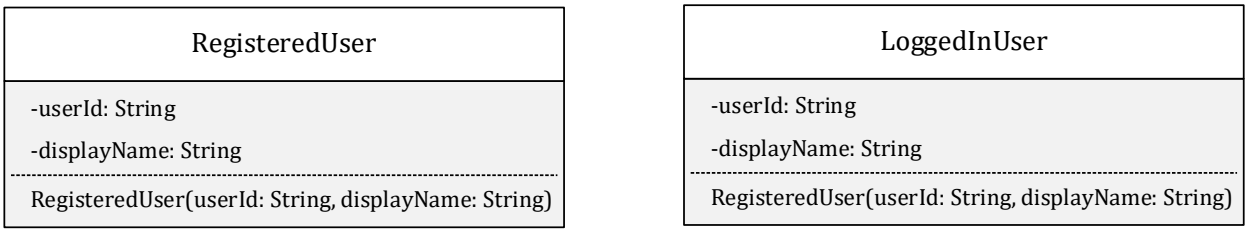

Рисунок 3.10 – Діаграма класів RegisteredUser та LoggedInUser

Рівень, вищий по відношенню до model – це репозиторій (Repository). Цей рівень добуває дані для рівня ViewModel, керує реєстрацією та аутентифікацією користувачів. У поточній програмі репозиторій втілений у двох класах LoginRepository та SignupRepository, проілюстрованих на рисунках 3.11 та 3.12.

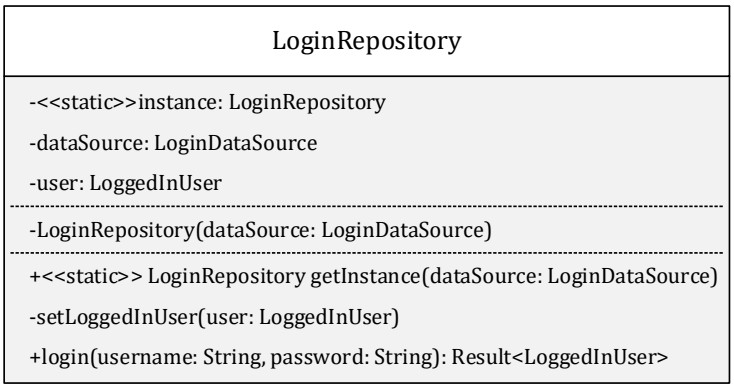

Рисунок 3.11 – Діаграма класу LoginRepository

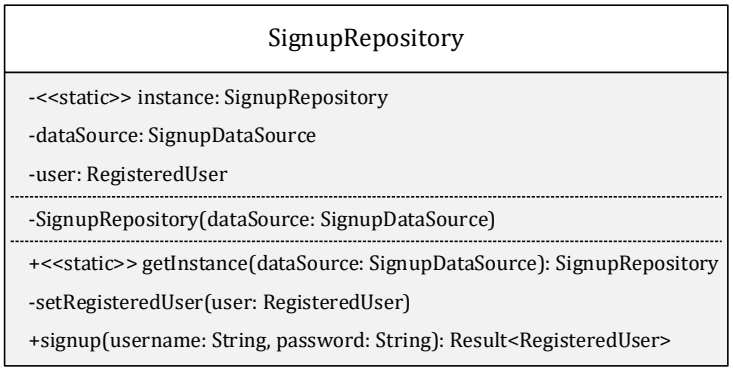

Рисунок 3.12 – Діаграма класу SignupRepository

Рівень ViewModel, який слідує за репозиторієм, специфікований класами SignupViewModel та LoginViewModel. Рисунок 3.13 ілюструє діаграму класів SignupViewModel та LoginViewModel.

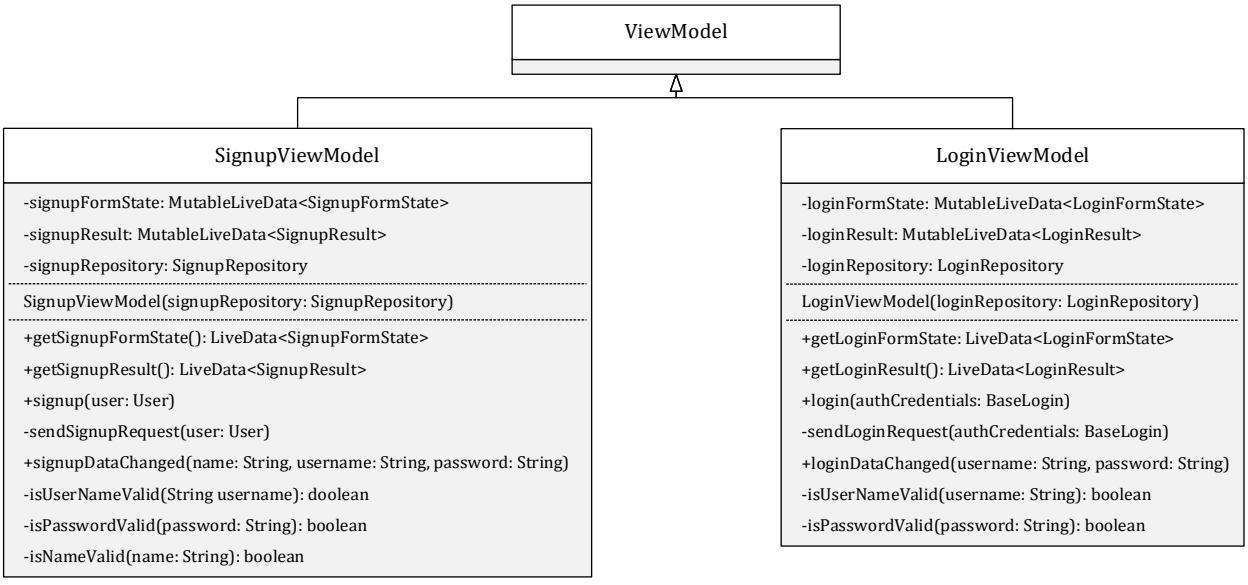

Рисунок 3.13 – Діаграма класів SignupViewModel та LoginViewModel

На верхньому рівні розташовані чотири активності. Перші три активності керують екраном входу, реєстрації та головним екраном. Інструкції з роботи цих екранів розташовані у класах LoginActivity, SignupActivity та HomeActivity. Рисунок 3.14 презентує діаграми цих класів.

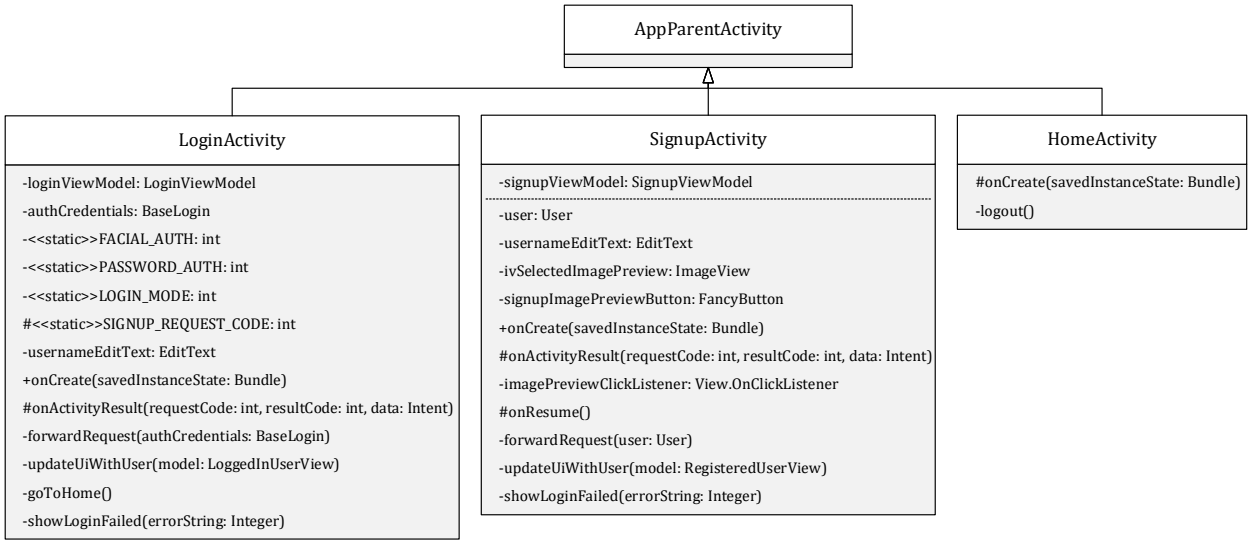

Рисунок 3.14 – Діаграми класів LoginActivity, SignupActivity та HomeActivity

Четверта активність представляє екран для автентифікації за допомогою розпізнавання обличчя.

У цій активності міститься логіка, яка за допомогою бібліотеки OpenCV, перетворює зображення у вигляд, необхідний для виконання операції розпізнавання.

Активність представлена класом FdActivity, який зображено на рисунку 3.15.

На рисунку 3.15 зображені не всі, але найбільш важливі методи класу FdActivity.

Також була розроблена частина додатка відповідальна за мережеву взаємодію із сервером. Для цього був створений інтерфейс APIInterface та клас APIClient.

Діаграма цих типів зображена на рисунку 3.16.

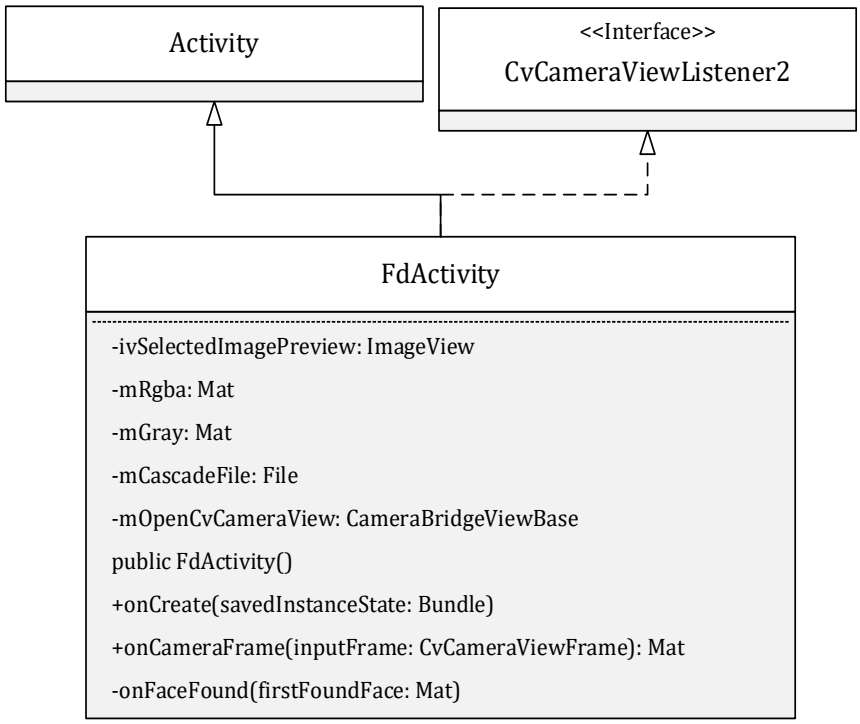

Рисунок 3.15 – Діаграма класу FdActivity

Рисунок 3.16 – Діаграма типів APIInterface та APIClient

Наведена архітектура та набір класів дозволяють створити додаток з необхідним функціоналом.

3.4.2 Інтеграція комп’ютерного зору

Додаток потребує використання технології комп’ютерного зору, для цього залучена бібліотека OpenCV. Тому інтеграція бібліотеки була необхідним процесом на етапі створення.

Інтеграція потребувала додавання нового модулю до проєкта. Після

аналізу документації OpenCV було вирішено завантажити набір інструментів OpenCV4Android SDK та додати до проєкту відповідну бібліотеку [11]. Структура бібліотеки OpenCV, інтегрованої до програми, зображена на рисунку 3.17.

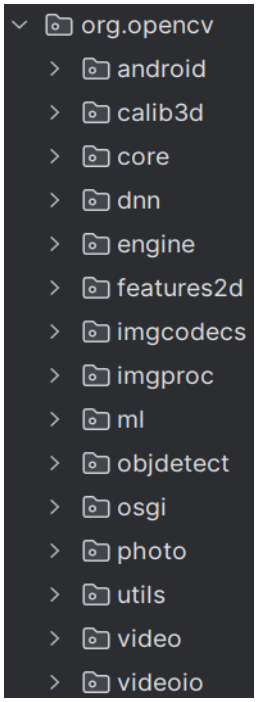

Рисунок 3.17 – Структура бібліотеки OpenCV

Також для виконання коду, пов’язаного з комп’ютерним зором, треба було мати додаткові бібліотеки. До складу OpenCV4Android SDK входять заголовки C++ та нативні Андроїд бібліотеки, які дозволяють адаптувати комп’ютерний зір під архітектури процесорів, які встановлюються на мобільні пристрої. Сюди входять такі архітектури, як: ARM-v5, ARM-v7a та x86. То ж до проєкту було додано бібліотеки, зображені на рисунку 3.18.

Доступ до функціоналу C++ стає можливим завдяки інтерфейсу JNI (Java native interface).

Після інтеграції OpenCV було створено окремий пакет org.opencv.samples. У цьому пакеті створено код із залученням власної логіки для обробки інформації, отриманої від камери пристрою.

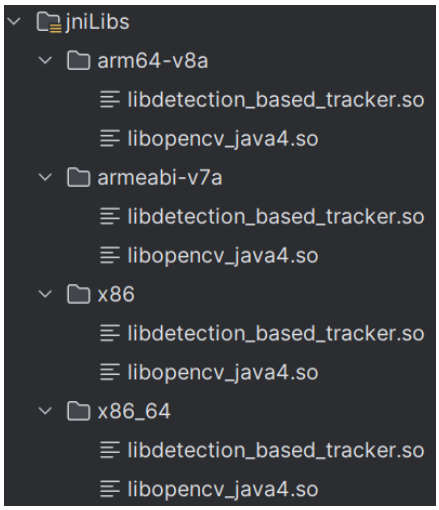

Рисунок 3.18 – Інтегрований функціонал для архітектур ARM-v5, ARM-v7a та x86

На рисунку 3.19 зображена структура клієнтського проєкту з адаптованим комп'ютерним зором.

Інтеграція комп'ютерного зору робить можливою подальшу розробку клієнтської частини, яка повинна мати необхідний функціонал.

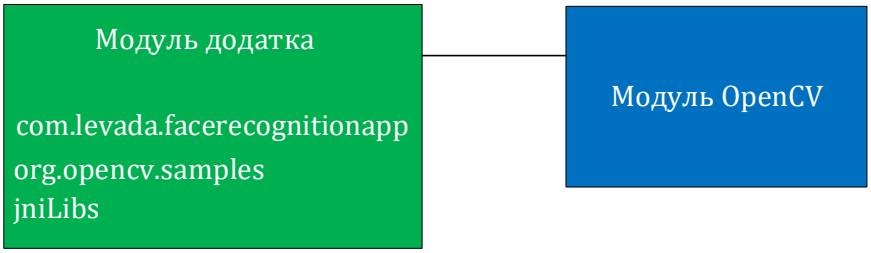

Рисунок 3.19 – Загальна структура клієнтського проєкта

3.4.3 Логіка роботи клієнта

Маючи архітектуру та інтегровану бібліотеку OpenCV, створимо логіку, яка деталізує кожну дію клієнтського додатка.

Коли додаток запускається вперше, лаунчер передає управління активності LoginActivity. Інтерфейс цієї активності зображено на рисунку 3.20.

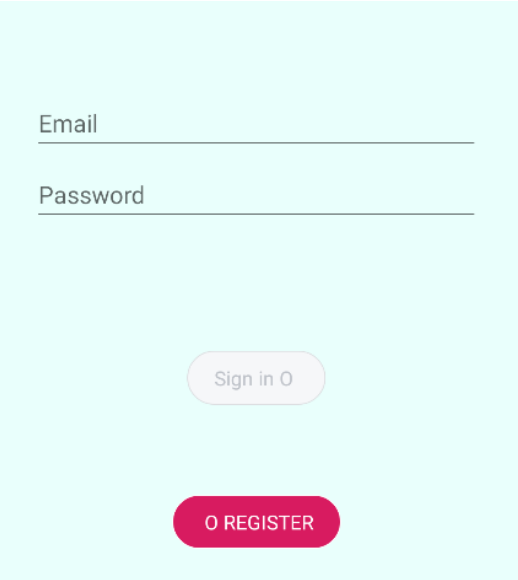

Рисунок 3.20 – Інтерфейс активності LoginActivity

Якщо користувач надав своє ім’я або email та натиснув кнопку «Sgin in», залишивши поле паролю пустим, додаток активізує послідовність дій для автентифікації на основі розпізнавання обличчя. Перш за все вступає в дію логіка обробки натискання на кнопку «Sign in», продемонстрована на рисунку 3.21.

```
loginButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (LOGIN_MODE == FACIAL_AUTH) {
            authCredentials = new FacialLogin();
            authCredentials.username = usernameEditText.getText().toString();
            startActivityForResult(new Intent( packageContext: LoginActivity.this, FdActivity.class),
                    GET_FACE_REQUEST_CODE);
        } else if (LOGIN_MODE == PASSWORD_AUTH) {
            authCredentials = new PasswordLogin();
            authCredentials.username = usernameEditText.getText().toString();
            ((PasswordLogin) authCredentials).password = passwordEditText.getText().toString();
            forwardRequest(authCredentials);
        }
    }
});
```

Рисунок 3.21 – Обробка натискання на кнопку «Sign in»

У цьому випадку управління буде йти у напрямку розгалуження, яке задовольняє умові LOGIN_MODE == FACIAL_AUTH.

Тоді логіка створить намір (об’єкт Intent) та розпочне запуск активності FdActivity, яка відповідає за активізацію камери пристрою та обробку отриманого зображення.

На рисунку 3.22 зображено макет активності FdActivity, розмітка якої сформована у файлі «fd_activity_surface_view.xml».

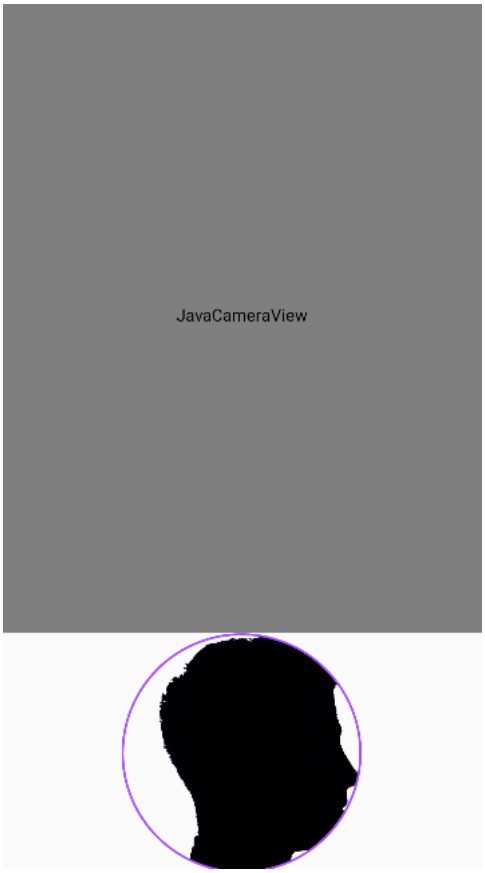

Рисунок 3.22 – Макет активності FdActivity

У верхній частині макета розміщено елемент JavaCameraView. Цей елемент представлений однойменним класом бібліотеки OpenCV. JavaCameraView виконує функцію моста між функціоналом OpenCV та об’єктом Camera, який входить до фреймворку Андроїд. Таким чином, зображення, отримане з камери пристрою, стане доступним для OpenCV.

На момент запуску FdActivity викликається метод життєвого циклу onCreate().

Серед основних інструкцій цього методу можна перерахувати

наступне:

– ініціалізація поля mOpenCvCameraView компонентом камери пристрою (це поле відноситься до типу CameraBridgeViewBase, який є базовим для JavaCameraView);
– переведення об’єкта mOpenCvCameraView у видимий стан, зв’язування його з фронтальною камерою пристрою;
– реєстрація слухача до mOpenCvCameraView для подальшої обробки потоку кадрів які будуть надходити до середовища додатка;
– зв’язування зображення попереднього перегляду з полем ivSelectedImagePreview.

Перераховані інструкції зображені на рисунку 3.23.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ///...
    mOpenCvCameraView = (CameraBridgeViewBase) findViewById(R.id.fd_activity_surface_view);
    mOpenCvCameraView.setVisibility(CameraBridgeViewBase.VISIBLE);
    int num = Camera.getNumberOfCameras();
    if (num > 1) {
        mOpenCvCameraView.setCameraIndex(1);
    }
    mOpenCvCameraView.setCvCameraViewListener(this);
    ivSelectedImagePreview = findViewById(R.id.iv_selected_image_preview);
}
```

Рисунок 3.23 – Основні інструкції методу FdActivity.onCreate()

Після зв’язування камери пристрою з OpenCV додаток здатний приймати та обробляти кожен кадр, що надходить. Таку поведінку забезпечує об’єкт JavaCameraView, який у окремому потоці, надсилає кадри за допомогою методу deliverAndDrawFrame(). Після опрацювання цього методу система викликає наступний, сигнатура якого зображена на рисунку 3.24.

```
Mat CameraBridgeViewBase.CvCameraViewListener2.onCameraFrame(CvCameraViewFrame inputFrame)
```

Рисунок 3.24 – сигнатура методу onCameraFrame

Цей метод стає доступним у класі FdActivity завдяки наслідуванню.

На початку роботи цього методу розглядається кожен піксель поточного кадра. Піксель містить чисельні значення RGBA та відтінків сірого. Такі дані збираються у багатовимірні матриці та піддаються процесу виявлення обличчя на поточному кадрі. Основні інструкції методу onCameraFrame() зображені на рисунку 3.25.

```
public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
    mRgba = inputFrame.rgba();
    mGray = inputFrame.gray();
    ///...
    firstFoundFace = null;
    Rect[] facesArray = faces.toArray();
    for (int i = 0; i < facesArray.length; i++) {
        Imgproc.rectangle(mRgba, facesArray[i].tl(), facesArray[i].br(),
        FACE_RECT_COLOR, 3);
        try {
            firstFoundFace = new Mat(mRgba, facesArray[i]);

        } catch (Exception e) {}
    }
    onFaceFound(firstFoundFace);
    return mRgba;
}
```

Рисунок 3.25 – Основні інструкції методу onCameraFrame()

Перед виконанням інструкції return викликається метод onFaceFound() з наступною сигнатурою: *private void onFaceFound(final Mat firstFoundFace)*.

Якщо обличчя було виявлено, як параметр метод отримує значення, відмінне від null. У протилежному випадку метод не виконає жодної інструкції. При відповідних умовах на початку методу виконується перетворення отриманої матриці до типу Bitmap – рідному для системи типів Андроїд. Далі пояснюються наступні інструкції цього методу, які вважаються основними. Стиснення зображення у формат PNG. Кодування зображення стандартом Base64, та представлення його як рядка. Збереження отриманих даних у класі StaticData. Створення інтента на повернення до викликаючої активності. На рисунку 3.26 відображається початковий код методу onFaceFound().

Після завершення методу onFaceFound() операційна система викликає метод onActivityResult(), щоб повернутись до LoginActivity. У методі ініціалізується поле authCredentials з обличчям користувача, далі викликається метод forwardRequest(authCredentials), який посилає сигнал на рівень ViewModel за допомогою методу login() об'єкта loginViewModel. Метод login() призводить до виклику sendLoginRequest().

```
private void onFaceFound(final Mat firstFoundFace) {
    if (firstFoundFace != null) {
        final Bitmap faceImageBitmap = OpenCVUtils.convertMatToBitMap(firstFoundFace);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                ivSelectedImagePreview.setImageBitmap(faceImageBitmap);
            }
        });
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        faceImageBitmap.compress(Bitmap.CompressFormat.PNG, 100, byteArrayOutputStream);
        byte[] byteArray = byteArrayOutputStream.toByteArray();
        String faceImageBase64 = Base64.encodeToString(byteArray, Base64.NO_WRAP);
        faceImageBase64 = "" + faceImageBase64;
        Intent resultData = new Intent();
        StaticData.base64ImageData = faceImageBase64;
        StaticData.bitmapImageData = faceImageBitmap;
        setResult(RESULT_OK, resultData);
        finish();
    }
}
```

Рисунок 3.26 – Початковий код методу onFaceFound()

У методі sendLoginRequest() реалізується логіка звернення до серверу. Будується екземпляр типу APIInterface та формується запит на вхід до системи з пред'явленими реквізитами. Також у цьому методі задаються інструкції, які будуть виконуватись у випадку вдалої відповіді від серверу. На рисунку 3.27 зображено початковий код методу sendLoginRequest().

У методі зворотного виклику onResponse встановлюється значення з відповіддю від серверу для об'єкта типу LiveData. Це змушує оповістити про зміни даних в об'єкті LiveData та викликати метод спостерігача. На рисунку 3.28 зображено початковий код, у якому створюється підписка на LiveData об'єкт компоненти ViewModel.

```
private void sendLoginRequest(BaseLogin authCredentials) {
    APIInterface apiInterface = APIClient.getClient().create(APIInterface.class);
    Call<ResponseMessage> signinAction = null;
    if (authCredentials instanceof FacialLogin) {
        signinAction = apiInterface.authenticateUserFace((FacialLogin) authCredentials);
    } else if (authCredentials instanceof PasswordLogin) {
        signinAction = apiInterface.authenticateUserPassword((PasswordLogin) authCredentials);
    }
    signinAction.enqueue(new Callback<ResponseMessage>() {
        @Override
        public void onResponse(Call<ResponseMessage> call, Response<ResponseMessage> response) {
            ResponseMessage responseMessage = response.body();
            if (responseMessage != null) {
                loginResult.setValue(new LoginResult(new LoggedInUserView(responseMessage.success,
                responseMessage.message)));
                Timber.d("RESPONSE MESSAGE : : " + responseMessage.message);
            } else {
                loginResult.setValue(new LoginResult(R.string.login_failed));
            }
        }
        @Override
        public void onFailure(Call<ResponseMessage> call, Throwable t) {
            loginResult.setValue(new LoginResult(R.string.connection_error));
        }
    });
}
```

Рисунок 3.27 – Початковий код методу sendLoginRequest()

```
loginViewModel.getLoginResult().observe( owner: this, new Observer<LoginResult>() {
    @Override
    public void onChanged(@Nullable LoginResult loginResult) {
        if (loginResult == null) {
            return;
        }
        sweetAlertDialog.hide();
        if (loginResult.getError() != null) {
            showLoginFailed(loginResult.getError());
        }
        if (loginResult.getSuccess() != null) {
            updateUiWithUser(loginResult.getSuccess());
        }
    }
});
```

Рисунок 3.28 – Підписка на зміни даних у loginViewModel

Якщо спроба автентифікації була вдалою система викликає метод updateUiWithUser().

Метод updateUiWithUser зберігає логін користувача та відомості про те, що вхід зроблено. Початковий код методу updateUiWithUser() зображено на рисунку 3.29.

```
private void updateUiWithUser(LoggedInUserView model) {
    String welcome = model.getDisplayName();
    // initiate successful logged in experience
    Toast.makeText(getApplicationContext(), welcome, Toast.LENGTH_LONG).show();
    if (model.getSuccess()) {
        SharedPreferences.Editor editor =
                PreferenceManager.getDefaultSharedPreferences( context: this).edit();
        editor.putBoolean(StaticData.IS_LOGGED_IN, true);
        editor.putString(StaticData.LOGGED_IN_USERNAME, model.getDisplayName());
        editor.commit();
        goToHome();
    }
}
```

Рисунок 3.29 – Початковий код методу updateUiWithUser()

Для зберігання даних використовується тип SharedPreferences. Наприкінці викликається метод goToHome(), який завантажує головну активність.

На рисунку 3.30 зображено графічне представлення логічного потоку автентифікації за допомогою розпізнавання обличчя.

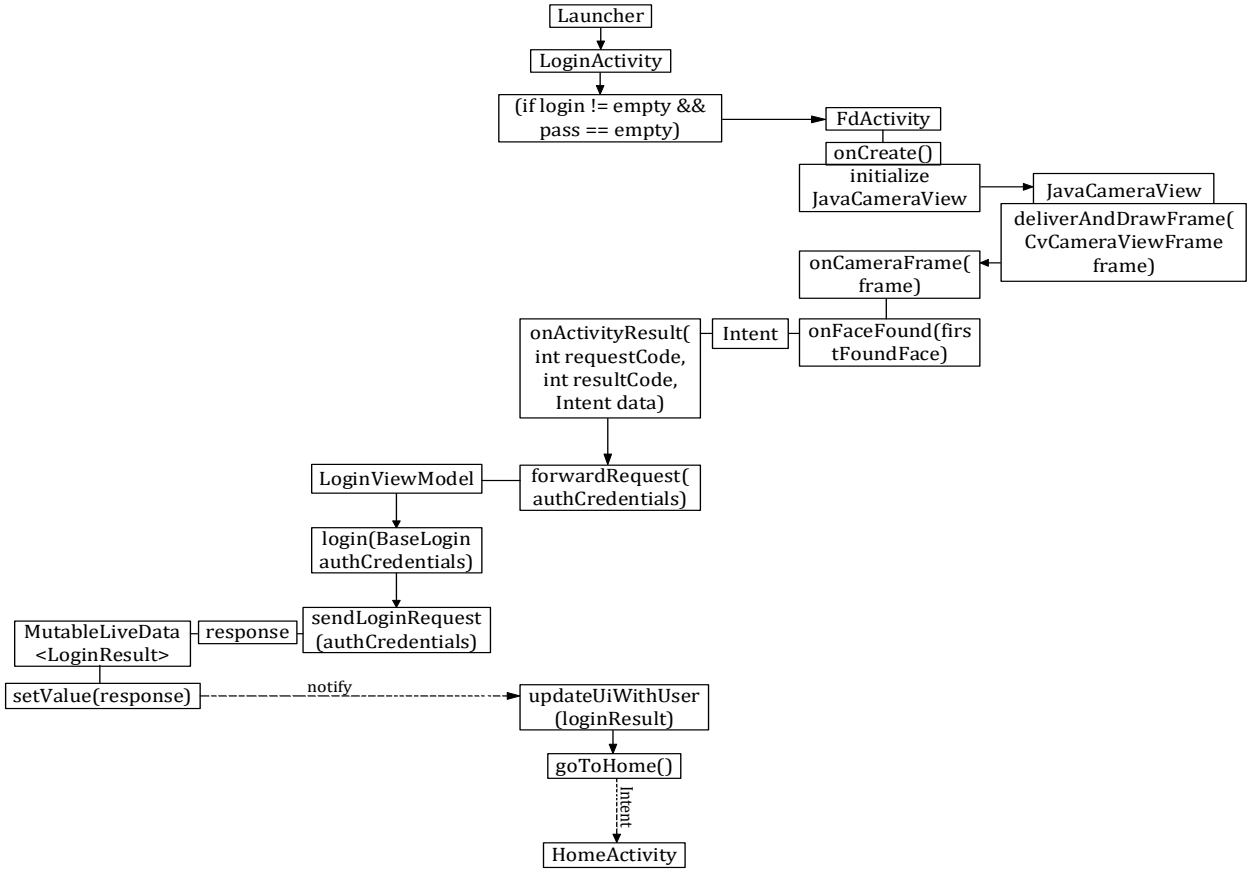

Рисунок 3.30 – Візуалізація логіки автентифікації за допомогою розпізнавання обличчя

Таким чином, клієнтський додаток реалізує автентифікацію на основі технології розпізнавання обличчя.

3.5 Висновки до третього розділу

Завантажуючи мобільний додаток вперше, користувач не повинен довго звикати до інтерфейсу та відчувати затримку в роботі. Тому інтерфейс додатка має бути ергономічним, а програмний код – без зайвих або складних інструкцій.

Створений додаток залучає функціонал бібліотеки OpenCV, яка створює певне навантаження на мобільну систему. Але все ж таки OpenCV для Андроїд може бути використаний належним чином та задовольняти потребам розробника.

У розробленому додатку бібліотека OpenCV додається як модуль. Це дозволяє відокремити власний проєкт від допоміжного та використовувати технологію комп’ютерного зору без порушення продуктивності.

Створення архітектурного уявлення клієнтської програми дозволило підійти до розробки раціональним шляхом та позбавитись від можливих помилок.

ВИСНОВКИ

Після виконання кваліфікаційної роботи було створено дві частини проєкту. Розробка серверної частини полягала у створенні програми, яка використовує технологію штучного інтелекту для надійної автентифікації. Серверний додаток наділений можливістю комп'ютерного зору та здатен обробляти оцифровані зображення спираючись на модель MTCNN. Розпізнавання обличчя на сервері також використовує функціонал бібліотеки TensorFlow.

У програмну систему втілений механізм контролю доступу, який складається із основних кроків: ідентифікація, автентифікація та авторизація. Процес автентифікації доволі складний тому що його логіка спирається на штучний інтелект. Складність для системи полягає в тому, що програма має справу з величезними масивами даних, які багаторазово піддаються обробці.

Реалізація моделі MTCNN тягне за собою деякі витрати апаратних ресурсів. Модель припускає залучення трьох типів PNetn, RNet, ONet, які виконують кропітку роботу для розпізнавання.

Клієнтський додаток також має справу з комп'ютерним зором. Інформація у вигляді зображення з обличчям спочатку потрапляє клієнту. То ж клієнтська програма першою повинна оцифрувати отримані дані. Якщо користувач бажає автентифікуватись за допомогою розпізнавання обличчя, клієнтський додаток отримує зображення та, за допомогою бібліотеки OpenCV, намагається знайти обличчя для подальшого створення запиту до сервера.

Бібліотека Retrofit дозволила створювати запити до сервера дуже лаконічно та полегшила розуміння коду.

Проєкт втілює клієнт-серверну архітектуру, яка працює за відношенням багато до одного. У своїй композиції серверний та клієнтський додаток повинні виконувати свою роботу без затримок.

ПЕРЕЛІК ПОСИЛАНЬ

1. Jason A. The basics of information security: understanding the fundamentals of InfoSec in theory and practice. Syngress, 2011. 190 p.

2. Husák M., Bartoš V., Sokol P., Gajdoš A. Predictive methods in cyber defense: Current experience and research challenges. ScienceDirect. URL: https://www.sciencedirect.com/science/article/abs/pii/S0167739X20329836 (дата звернення: 14.09.2023).

3. Jain A.K., Arun A.R., Nandakumar K. Introduction to biometrics. Springer, 2011. 328 p.

4. Rich E., Knight K. Artificial Intelligence, 2nd edition. McGraw Hill Higher Education, 1991. 640 p.

5. Poole D., Mackworth A., Goebel R. Computational Intelligence: A Logical Approach. Oxford University Press, 1998. 576 p.

6. Winston P.H. Artificial Intelligence, 3d edition. AddisonLWesley, 1992. 765 p.

7. Zhang K., Zhang Z., Li Z., Qiao Y. Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks. arXiv. URL: https://arxiv.org/ftp/arxiv/papers/1604/1604.02878.pdf (дата звернення: 26.09.2023).

8. DeJonghe D. NGINX cookbook. Advanced recipes for high-performance, load balancing. O'Reilly, 2022. 207 p.

9. Planche B., Andres E. Hands-On Computer Vision with TensorFlow 2. Packt, 2019. 363 p.

10. OpenCV About. opencv.org. URL: https://opencv.org/about/ (дата звернення: 21.09.2023).

11. OpenCV4Android SDK. opencv.org. URL: https://docs.opencv.org/2.4/doc/tutorials/introduction/android_binary_package/O4A_SDK.html (дата звернення: 11.10.2023).