МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

Система ідентифікації мешканців житлового комплексу

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

Освітня програма **«Комп'ютерні науки»**

Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи КН- 41

_____Кропта З.С._____
(прізвище та ініціали)

Керівник _____Гайна Г. А._____
(прізвище та ініціали)
__кандидат технічних наук, професор__
(науковий ступінь, звання)

Випускна кваліфікаційна робота бакалавра допущена до захисту
рішенням кафедри *інтелектуальних технологій*
Протокол № 11 від 06.06.2022 р.
зав. кафедри_____ доц. Іларіонов О.Є.

Київ - 2022

Анотація

Кропта Злата Сергіївна виконала випускну кваліфікаційну роботу на тему «Система ідентифікації мешканців житлового комплексу» за спеціальністю 122 – «Комп'ютерні науки».

У випускній кваліфікаційній роботі проаналізовано сучасний стан галузі контролю охорони об'єктів будівництва, проаналізовано сучасні системи, що використовуються для ідентифікації людини, розглянуто особливості обраних засобів, спроектована архітектура, реалізовано програму, що виконує ідентифікацію людей, як мешканців або чужинців у реальному часі.

Ключові слова: ідентифікація мешканців, веб-сайт, сервер, розпізнавання облич, база даних мешканців.

Summary

The degree project: «Identification system for residents of the residential complex» has completed by **Kropta Zlata** speciality 122 – «Computer Science».

In this graduation thesis the current state of control over the protection of construction sites and modern systems used for human identification are analysed, the features of selected tools are considered, the architecture is designed, the program that identifies people as residents or strangers in real time is realised.

Keywords: resident identification, website, server, face recognition, resident database.

ЗМІСТ

Перелік умовних позначень і скорочень

ЖК — житловий комплекс

БД — база даних

НМ — нейронні мережі

LFRT — технологія розпізнавання в реальному часі

ІЧ — інфрачервоний

CPOI — система розпізнавання облич Інтерполу

СУРБД — Системи управління реляційними базами данихё

CRUD - create, read, update and delete

SQL — мова структурованих запитів

ACID — атомарність, послідовність, ізоляція,довговічність

CPU

RAM

SSD

NoSQL

JSON

ORM — Object Relational Mapping

VAD — Value Added Chain Diagram

EPC — Event-Driven Process Chain

HOG — Histogram of oriented gradients

SVM — Support vector machines

ВСТУП

Дана випускна кваліфікаційна робота написана на тему розробки програмного рішення для ідентифікації мешканців житлових комлексів.

Основною метою даної роботи є запровадження технології для житлових комплексів, що дозволить ідентифікувати людей, як мешканців або як чужинців даного комплексу задля підвищення безпеки мешканців та задля спрощення процесу входу мешканців на територію ЖК.

Об’єкт дослідження – процес розпізнавання та ідентифікації облич у відеопотоці на вході житлових комплексів.

Предметом є використання та дослідження алгоритмів знаходження та розташування обличчя, оцінки пози обличчя, опису обличчя.

У ході виконання випускної кваліфікаційної роботи має бути реалізоване представлення роботи системи з ідентифікації людини та віднесення її до мешканця ЖК або ж до чужинця.

У даній роботі я хочу запропонувати новий підхід для забезпечення безпеки в житлових комплексах - це вхід до ЖК за допомогою розпізнавання облич. Цей метод не є чимось новим, та, звичайно він вже використовується у багатьох інших сферах. Але, досліджуючи дану предметну область, було помічено, що в Україні до сих пір не ввели такий спосіб охорони ЖК. З усіх схожих рішень було знайдено лише використання розпізнавання обличь на домофоні підїзду, але не самого ЖК.

Дана робота складається з трьох розділів, де перший розділ - це аналітичний огляд теми, другий - проектування системи, та останній розділ - написання програмного рішення.

Отже, в даній роботі за мету береться підвищення безпеки мешканців та спрощення процесу входу мешканців на територію ЖК.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД СИСТЕМИ ІДЕНТИФІКАЦІЇ МЕШКАНЦІВ ЖИТЛОВОГО КОМПЛЕКСУ

1.1 Опис предметної області технології ідентифікації облич людей

Область застосування розроблюваної системи

Хоча ідентифікація обличчя може здатися чимось новим та передовим, наразі вона вже використовується у багатьох сферах діяльності[1]. Перша з них - це використання даної технології для забезпечення безпеки гаджетів.

Гаджети

Різні програми використовують ідентифікацію обличчя, щоб захистити вашу інформацію. Насправді, навіть безпечний секретний ключ не може захистити ваші записи та дані від талановитих програмістів, тому люди пішли на особисте підтвердження. Ці програми вимагають, щоб ваше обличчя відкривало мобільний телефон або доступ до особистої інформації.

Медицина

Наприклад, існують програми для медичних послуг, наприклад, Face2Gene та програми, такі як DeepGestalt, які використовують підтвердження особи для розпізнавання спадкової проблеми. Вони досліджують обличчя та порівнюють їх із базою даних облич тих, які містять ці розлади.

Ідентифікація певної категорії людей(купівля алкоголю)

Деякі ринки та бари у Великобританії використовують ідентифікацію облич, щоб визначити, чи є люди достатньо зрілими, щоб купувати алкоголь. Супермаркети дозволяють клієнтам використовувати касу самостійної покупки алкоголю, не вимагаючи додаткового працівника для перевірки посвідчень.

Загальна безпека

Наприклад, у багатьох магазинах є системи ідентифікації облич, які позначають людей як небезпеку, якщо вони крадуть. Ця структура може розрізнити крадіїв і повідомити комірника про їхні минулі відмінності, незалежно від того, чи вони колись заходили в цей магазин чи ні.

Систему ідентифікації обличчя використовували як міру безпеки у найвищих установах і на робочих місцях, щоб гарантувати відсутність будь-

якого вандалізму. Цей тип програмного забезпечення не залишає місця для людських помилок і є важливим інструментом допомоги. За допомогою набору алгоритмів програмне забезпечення виконує геометричне та фотометричне розпізнавання за лічені секунди.

Розпізнавання обличчя зробило перевірку відносно простішою, без особливого оснащення та великої кількості інформації, доступної за лічені хвилини.

Існуючі методи ідентифікації людини[2]:

Технологія розпізнавання осіб довела свою цінність у різних сценаріях використання для різних організацій. Ось кілька способів використання розпізнавання осіб для ефективної та дієвої перевірки особистості сьогодні:

Безпека пристрою: Людина може використовуватися як фактор автентифікації для розблокування мобільних пристроїв. У цьому випадку пряме зображення порівнюється з раніше існуючим зображенням у приватній базі даних телефону. Цією функцією оснащені деякі моделі мобільних пристроїв Apple iPhone та Microsoft Android.

Заходи щодо запобігання крадіжкам: Правоохоронні органи можуть використовувати системи розпізнавання осіб для ідентифікації підозрюваних у відомих кримінальних базах даних після крадіжки.

Купівля алкоголю: Деякі бари та магазини спиртних напоїв використовують технологію розпізнавання осіб для виявлення підроблених посвідчень особи та водійських прав, одночасно покращуючи якість обслуговування клієнтів. Наприклад, за допомогою технології розпізнавання осіб клієнти, які погоджуються на зберігання їх даних у приватній базі даних, що належить магазину, можуть безпечно купувати алкоголь у касах самообслуговування.

Шкільна безпека: Система розпізнавання осіб може використовуватись у школі для виявлення відомих підозрілих осіб, таких як виключені учні, торговці наркотиками та інші загрози. Після цього адміністрація може звернутися до служби безпеки школи, щоб розібратися в ситуації.

Охорона аеропорту: В аеропортах вже використовується розпізнавання осіб, щоб прискорити процес безпеки та посадки. У деяких країнах ручна перевірка документів, що засвідчують особу, може виконуватися електронними воротами, які використовують розпізнавання осіб та порівняння осіб для перевірки посвідчення особи мандрівника. Крім того, державні установи використовували цю технологію для виявлення осіб, які прострочили візу або перебувають під слідством.

Правоохоронні органи: Використовуючи технологію розпізнавання осіб, правоохоронні органи можуть значно підвищити свою ефективність у боротьбі з такими загрозами, як підробка особистих даних. Наприклад, Департамент транспортних засобів Нью-Йорка використав програму розпізнавання осіб для ідентифікації 21 000 випадків потенційного шахрайства з використанням особистих даних з 2010 року.

Найпопулярніший приклад використання ідентифікації обличчя людини

В телефонах Apple використовується технологія Face ID, що дає змогу користувачам безпечно користоватися телефоном за допомогою їхнього обличчя[3].

За допомогою технології Face ID можна безпечно розблокувати iPhone та iPad, підтверджувати покупки, виконувати вхід у додатках та багато іншого одним тільки поглядом(див. рисунок 1.1).

Рис 1.1 – Робота Face ID в телефонах Iphone

До складу Face ID входить кілька компонентів(див. рис. 1.2.)[4]:

- Проектор точок TrueDepth, що створює «хмару», що проектується на скановану особу, на основі якої будується унікальна карта, яка вдосконалюється та доповнюється при кожному новому скануванні.
- ІЧ-камера, яка зчитує дані «хмари» і направляє їх у систему Secure Enclave, що входить до складу фірмового чіпсету Apple A11 Bionic Neural.
- Інфрачервоний випромінювач, що забезпечує стабільну авторизацію навіть у темряві.

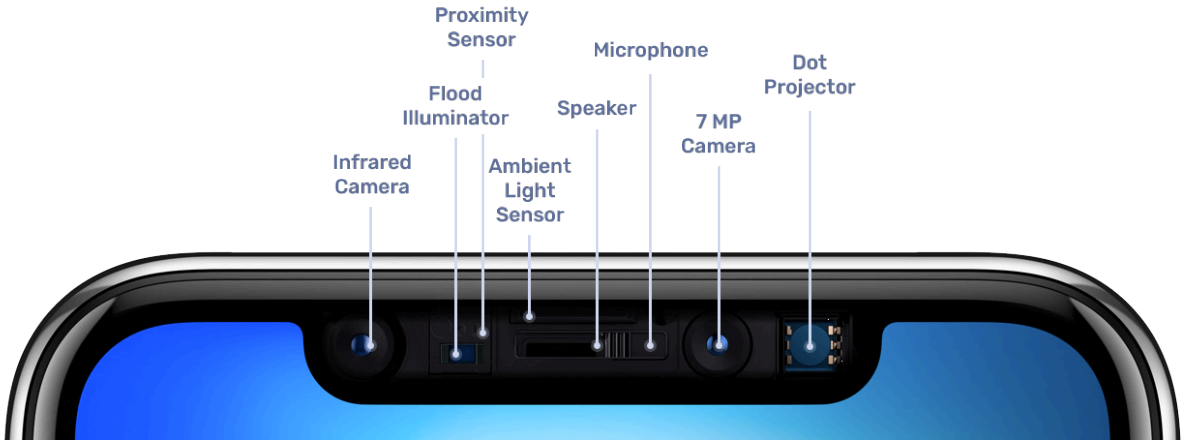

Рис 1.2 – Камера TrueDepth

Якщо дуже коротко, то Apple Face ID – це система, яка використовує розпізнавання облич для розблокування смартфона та підтвердження платежів[5]. Вона порівнює обличчя користувача з картинкою, що зберігається у пам'яті айфона. А робить ці зображення iPhone за допомогою спеціальної камери True Depth Camera.

У верхній частині передньої панелі iPhone є невеликий острівець, не закритий великим екраном і під зав'язку набитий різною електронікою і датчиками. Крім звичайного набору (фронтальна камера, мікрофон, динамік,

датчики наближення та освітленості), тут є інфрачервона камера, система підсвічування (flood illuminator) та проектор точок (dot projector).

Існуючі схожі підходи ідентифікації людини

Один із головних інструментів безпеки в сучасних будинках - це "розумний" домофон у під'їздах[6]. Такий, наприклад, встановлено у столичному ЖК EINSTEIN Concept House.

Девайс оснащений сенсорним екраном та камерою, яка вміє розпізнавати обличчя. Увійти в будинок можна трьома способами – за допомогою карти, сканування відбитка пальця або технології розпізнавання облич. Гості можуть відчинити двері, зателефонувавши в конкретну квартиру як на звичайних домофонах. При цьому господар завдяки камері бачить хто конкретно до нього прийшов(див. рисунок 1.3).

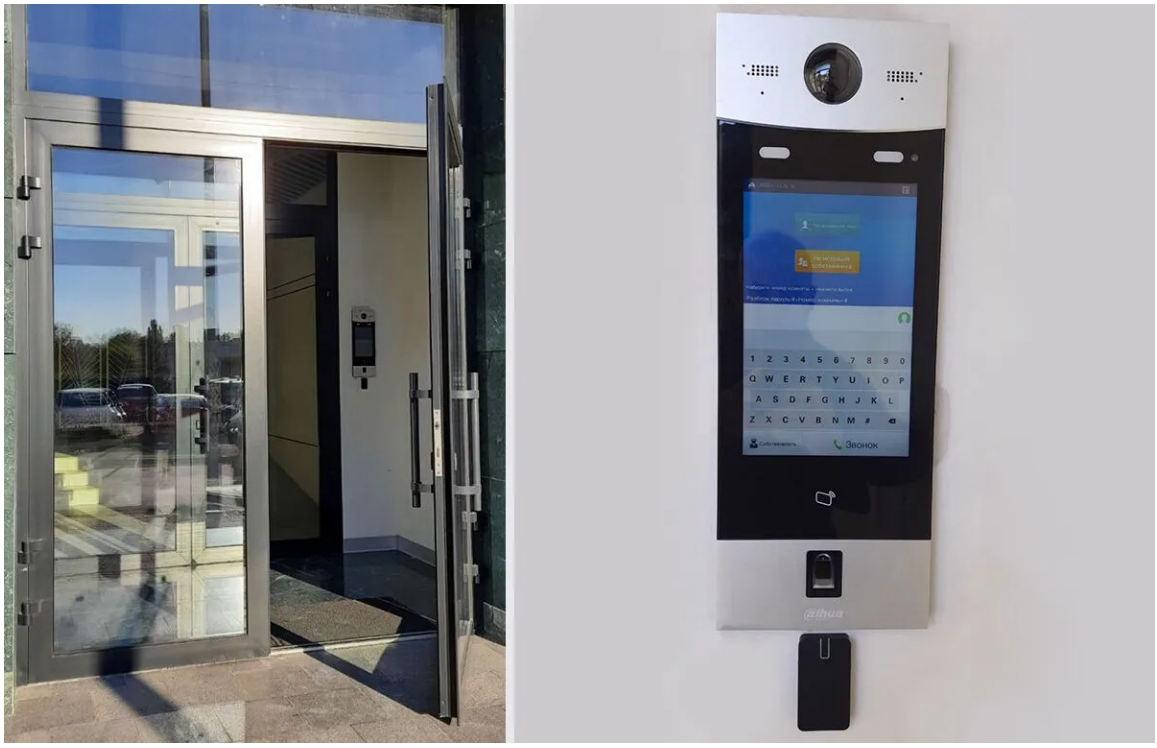

Рис 1.3 – "Розумний" домофон у під'їзді в українському ЖК

Система розпізнавання облич Інтерполу (CPOI) містить зображення облич, отримані з більш ніж 179 країн, що робить її унікальною глобальною базою даних про злочини[7].

У поєднанні з автоматизованим біометричним програмним забезпеченням ця система здатна ідентифікувати або перевіряти людину, порівнюючи та аналізуючи візерунки, форми та пропорції її рис і контурів обличчя.

Майже 1500 терористів, злочинців, втікачів, зацікавлених осіб або зниклих безвісти були ідентифіковані з моменту запуску системи розпізнавання облич Інтерполу наприкінці 2016 року(див. рисунок 1.4).

Рис 1.4 – Розпізнавання облич у правоохоронних органах

Коли зображення обличчя вводиться в систему, воно автоматично кодується за допомогою алгоритму та порівнюється з профілями, які вже зберігаються в системі. Це призводить до списку «кандидатів» із найбільш вірогідними збігами(див. рисунок 1.5).

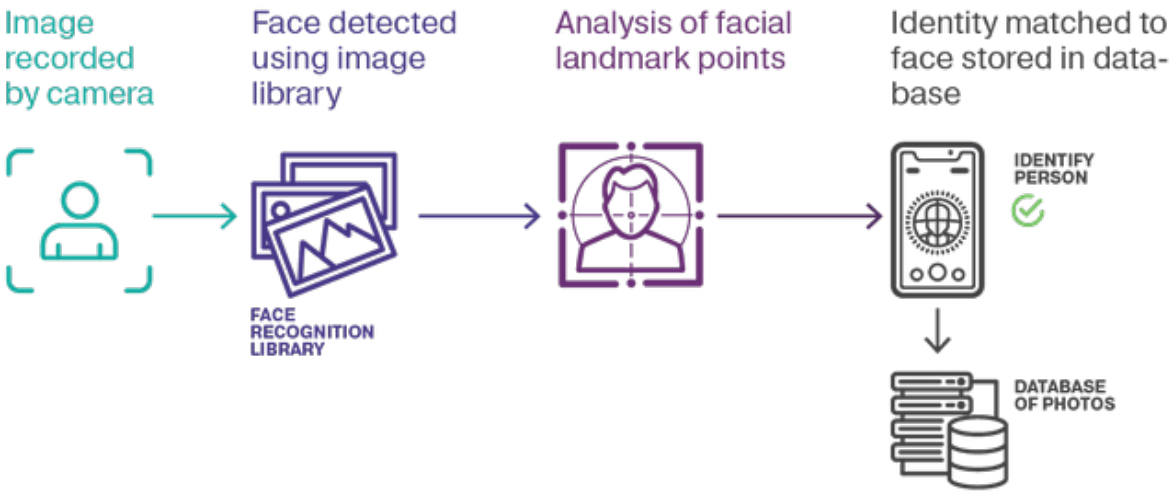

Рис. 1.5 – Схема роботи ідентифікації облич

Актуальність розробки системи ідентицікації мешканців ЖК

У більшості країн світу існує стійка тенденція до урбанізації. Все більшої популярності набувають житлові комплекси з власною соціальною інфраструктурою: фітнес-центрами, басейнами, кафе-барами, наземними та підземними паркінгами, дитячими майданчиками тощо.

У рейтингу країн світу за рівнем урбанізації, що випускається Департаментом Організації Об'єднаних Націй з економічних та соціальних питань, Україна посідає 64 місце – міське населення нашої країни становить 69,1 %(рис. 1.6).

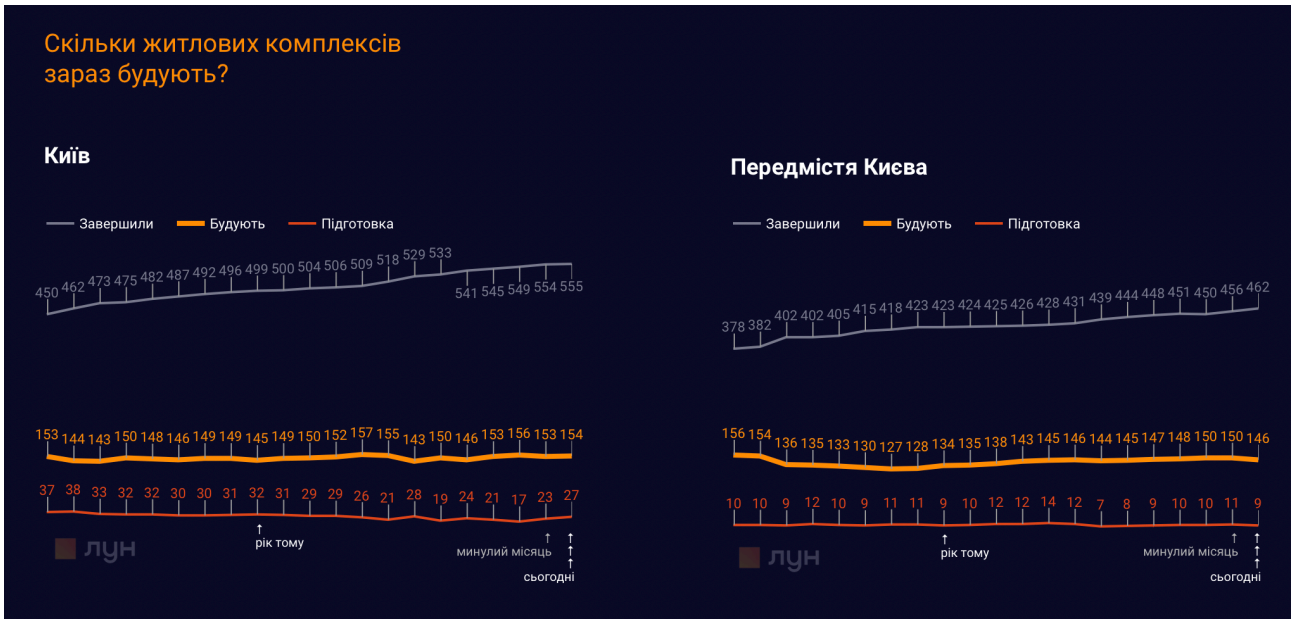

Рис. 1.6 – Статистика розмірів будівництва ЖК у Києві від ЛУН

Поряд із цим все гостріше постає питання безпеки сім'ї та майна мешканців. Справжня ціна комфорту та благополуччя у житлових комплексах визначається саме рівнем безпеки.

Безпека будинку є першочерговою турботою кожного. Для захисту наших речей та цінностей від будь-якого пограбування чи крадіжки потрібна система безпеки будинку. У Сполучених Штатах є квартирні крадіжки, які відбуваються кожні 13 секунд, 4 крадіжки на хвилину, 240 крадіжок на годину і майже 6000 на день! За деякими статистичними даними, 88% усіх крадіжок зі зломом носять житловий характер, 77% всіх злочинів становлять злочини проти власності, 38% усіх пограбувань здійснюються із застосуванням зброї, крадіжка особистих даних є найшвидшим злочином у США, Канаді та Великобританії. 3 з 4 будинків у США будуть зруйновані протягом наступних 20 років [8]. Традиційна система домашньої безпеки легко зламається і досить застаріла. Це, в свою чергу, призводить до пограбування, а також потребує встановлення дорогої системи безпеки. Дана запропонована система забезпечує економічне та енергоефективне рішення для домашньої безпеки за допомогою розпізнавання облич та їх ідентифікації.

Зацікавлені сторони (стейкхолдери)

Стейкхолдери — це ті, хто активно залучений до проекту чи бізнесу, ті, на чиї інтереси може вплинути успіх чи неуспіх проекту, а також ті, хто через свою посаду чи повноваження може сам вплинути на проект[9].

Для цього проекту були визначені наступні стейкхолдери:

1) Внутрішні зацікавлені сторони (офіційно залучені до функціонування системи):

- Інженери-розробники;
- Забудовники;
- Інвестори(клієнти);
- Акціонери;

2) Зовнішні зацікавлені сторони (не приймають безпосередньої участі, але можуть впливати на функціонування системи):

- Контролюючі органи (податкова служба, служба безпеки України і т.д.)
- Державні органи (міністерство економіки, національний банк України).

3) Прихильники (бенефіціари) і супротивники (антагоністи):

-Бенефіціари — ті сторони, які у результаті повинні отримати вигоду від виконання проекту, тому відкриті до взаємодії і переговорів. Як приклад, бенефіціарами можна назвати власників ЖК, клієнтів, що отримають користь від нової системи безпеки, інвесторів та акціонерів, інформаційні компанії інших регіонів, що можуть дослідити та впровадити розроблену систему, менеджмент та постачальники.

- Антагоністи — ті сторони, які потенційно або фактично ворожі по відношенню до проекту. Як приклад, можна назвати компанії-конкуренти, що пропонують свої аналоги на розгляд, прихильники традиційних способів охорони території.

Матриця зацікавлених сторін(рис.1.7) допомагає визначити тактику взаємодії з різними групами стейкхолдерів проекту[10].

"Зацікавленість" відображає рівень інтересу стейкхолдера до проекту.

"Впливовість" означає силу впливу стейкхолдера на проект.

У верхньому правому квадранті знаходиться зона позитивної (сприятливої) впливовості та зацікавленості. У ній розміщуються стейкхолдери, які налаштовані позитивно по відношенню до компанії, мають можливість впливу на результати діяльності.

Далі, за годинниковою стрілкою, у правому нижньому квадранті – зацікавлені стекйхолдери, що не мають змоги впливати.

Третій (лівий нижній) квадрант призначений для зацікавлених у негативному результаті, але безсилих проти організації чи її проекту.

Четвертий (лівий правий) – впливові супротивники. Це найнебезпечніший квадрант. Йому доведеться приділяти найпильнішу увагу.

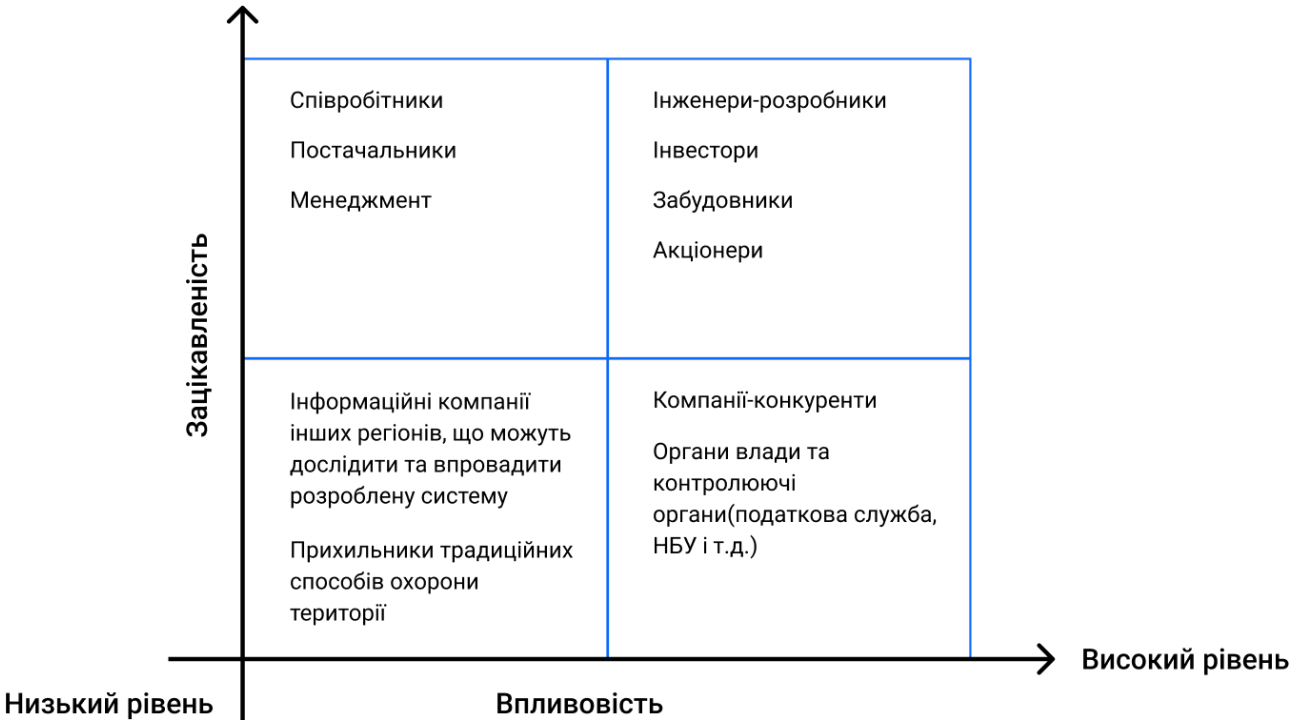

Рис 1.7 – Матриця стейкхолдерів

Проблемни випускної кваліфікаційної роботи

Проблемним моментом випускної кваліфікаційної роботи є відсутність прямого доступу до необхідних інструментів та обладнання для

функціонування повноцінної системи ідентифікації облич. Оскільки дана система повинна бути задіяна в житлових комплексах у вигляді камери, яка при вході в ЖК розпізнає кожного мешканця та приймає рішення про те, чи є ця людина мешканцем ЖК чи ні і чи можна впустити людину на територію ЖК, то було прийнято рішення змоделювати цю модель поведінки. Тобто, розробити систему, яка буде підключатися до камери пристрою, і, використовуючи базу даних з усіма мешканцями даного ЖК, порівнюватиме кожну особу, яка потраплятиме в поле зору камери, порівнюватиме з БД і прийматиме рішення, чи є ця людина мешканцем нашого ЖК.

Також у ході аналізу предметної області були виявлені наступні проблеми її функціонування:

- Погодні умови — певні невзгоди, що можуть заважати коректному функціонуванню даної системи.
- Зріст людини — проблема правильного розположення людини щодо камери, щоб її обличчя зчитувалося належним чином.
- Освітлення — погане освітлення обличчя людини може пошкодити належній ідентифікації її особистості.

Рішення проблем

Дані проблеми були проаналізовані та знайдені раціональні рішення. Отже, камера для розпізнавання облич буде знаходитися в невеликому приміщенні зі стабільним освітленням - людина заходить всередину, встає на певну мітку задля того, щоб її обличчя було гарно видно.

Схематично усе буде виглядати наступним чином(рис 1.8):

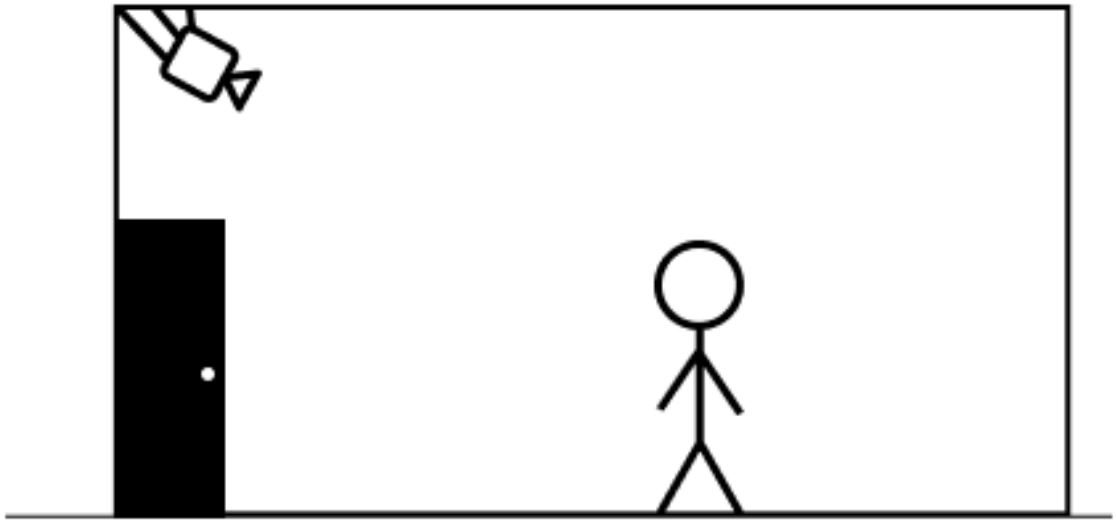

Рис 1.8 – Схема роботи системи охорони в ЖК

Переваги та недоліки ідентифікації облич людей

Ідентифікація людини за зображенням особи або через потік відео має ряд переваг у порівнянні з іншими методами ідентифікації людини:

1) не потрібне спеціальне чи дороге обладнання;

2) не потрібен фізичний контакт із пристроями. Не треба ні до чого торкатися або спеціально зупинятися і чекати на спрацювання системи. У більшості випадків досить просто пройти повз або затриматися перед камерою на невеликий час.

До недоліків ідентифікації людини слід віднести:

1) сама по собі така система не забезпечує 100% надійності ідентифікації. Там, де потрібна висока надійність, застосовують комбінування кількох біометричних методів;

2) розпізнавання особи неефективне тоді, коли є значні зміни, наприклад, унаслідок нещасного випадку унеможливлюють навіть людську візуалізацію;

Технології, що необхідні для виконання даної задачі

Технологія розпізнавання обличчя покладається на процес ідентифікації або перевірки особистості людини за допомогою її рис обличчя[11]. Програмне забезпечення, відоме як перевірка обличчя, можна використовувати для

встановлення чиєїсь особи, наприклад, як спосіб розблокувати електронні пристрої та допомогти при перевірці паспортів в аеропортах.

У контексті нерухомості він також часто зустрічається в системах входу, які автоматично надають доступ персоналу та авторизованим відвідувачам замість видачі брелока чи картки. Такі види використання зазвичай здійснюються з знанням суб'єкта і зазвичай працюють на свою користь, прискорюючи процеси та забезпечуючи підвищену безпеку.

Системи ідентифікації обличчя найчастіше зустрічаються на платформах соціальних мереж, де вони автоматично ідентифікують те саме обличчя на кількох фотографіях і можуть пов'язувати обличчя з профілем. Поліція також використовує ідентифікацію обличчя, наприклад, на фотографіях, знятих із камер відеоспостереження або соціальних мереж, щоб ідентифікувати когось у своїй базі даних зображень під вартою.

Однак технологія розпізнавання обличчя в реальному часі («LFRT») є більш суперечливою. Вона працює за допомогою програмного забезпечення для розпізнавання облич для сканування відеозапису в реальному часі, виявлення облич у кадрі, а потім їх перевірки на відповідність встановленим критеріям, таким як список спостереження. Будь-які збіги, які знімають попередньо встановлений поріг, потім оцінюються та відображаються. Він функціонує більше як відеоспостереження і іноді використовується без відома суб'єктів даних.

1.2 Постановка задачі технології ідентифікації облич людей

Задачі випускної кваліфікаційної роботи

Тема випускної кваліфікаційної роботи: розробка програмного рішення для ідентифікації мешканців житлових комлексів.

Основною метою даної роботи є запровадження технології для житлових комплексів, що дозволить ідентифікувати людей, як мешканців або як чужинців даного комплексу задля підвищення безпеки мешканців та задля спрощення процесу входу мешканців на територію ЖК.

Об’єкт дослідження – процес розпізнавання та ідентифікації облич у відеопотоці на вході житлових комплексів

Предметом є використання та дослідження алгоритмів знаходження та розташування обличчя, оцінки пози обличчя, опису обличчя.

У ході виконання випускної кваліфікаційної роботи має бути реалізоване представлення роботи системи з ідентифікації людини та віднесення її до мешканця ЖК або ж до чужинця.

Отже, для досягнення мети необхідно вирішити такі завдання:

- Створити базу даних мешканців ЖК, у якій будуть міститися дані облич кожного з мешканців, а також й всі інші важливі дані про мешканців, а саме: ПІБ та адреса.
- Розробити веб сервіс, на якому можна буде продемонструвати роботу даного проекту.
- Розробити функціонал, за допомогою якого можна буде зв’язати камеру пристроя з працюючою програмою.
- Розробити адмін панель з можливістю вводити нові дані про мешканців до БД, додавати нових мешканців, або змінювати поточні дані.
- Протестувати роботу веб сервісу на прикладі деякої кількості людей, занесених до бази даних, та також тих, що не занесені у БД.

У результаті буде отримана система ідентифікації облич мешканців ЖК із можливістю подальшого використання в існуючих ЖК.

Вимоги до розроблювальної системи

Системні вимоги до розроблюваної системи:

- наявність камери, що буде транслювати картинку.

Функціональні вимоги:

- розпізнавання та ідентифікація облич людей.
- доступ до бази даних мешканців через адмін-панель, можливість додати новий запис до БД, видалити або змінити існуючий.

- запис інформації до бази даних про прибуття того чи іншого мешканця.
- подання на вхід системи бази даних з даними про всіх мешканців.
- подання на вихід БД з записами усіх людей, що були розпізнані через камеру та їх даними.

Нефункціональні вимоги:

- забезпечувати високу швидкість обробки зображень, переданих з відео, у реальному часі.
- неперервність роботи.
- захищеність (програма не повинна надавати відкритий доступ до програмного коду третім особам).
- стабільність (програма не повинна допускати навіть тимчасових затримок під час своєї роботи).
- передбачати мінімальні витрати на впровадження програмного продукту.
- забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення.

Завдяки аналітичному огляду розглянутої предметної області були визначені мета та поставлене завдання випускної кваліфікаційної роботи. Тому можна переходити до функціонального та бізнес-аналізу, а далі й до проектування архітектури системи.

У першому розділі було проаналізовано сучасний стан галузі контролю охорони об'єктів будівництва. Було також проаналізовано сучасні системи, що використовуються для ідентифікації людини, було визначено зацікавлені сторони проекту, визначено проблеми та знайдено рішення, визначено функціональні та нефункціональні вимоги до проекту, були представлені цілі та задачі проекту.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ІДЕНТИФІКАЦІЇ МЕШКАНЦІВ ЖИТЛОВОГО КОМПЛЕКСУ

2.1 Функціональний та бізнес-аналіз системи ідентифікації мешканців ЖК

Функціональний аналіз

Система, що розглядається, складається з трьох підсистем: програма для розпізнавання та ідентифікації облич мешканців, веб-сайт з базою даних мешканців(з занесеними туди кодами їх облич).

Для програми мають бути реалізовані наступні функції:

- транслювання зображення у реальному часі;
- розпізнавання обличчя;
- порівняння зчитуваємого коду обличчя з наявними кодами облич у базі даних.

Для веб-сайту мають бути реалізовані наступні функції:

- адмін-панель з можливістю додавати, видалята та змінювати записи про мешканців;
- система зчитування обличчя мешканця та занесення його до БД у вигляді ембеддінгу;
- веб-сайт, що буде забезпечувати зручну взаємодію користувача із системою.

Після проведення функціонального аналізу можна перейти до декомпозиції вищерозглянутої контекстної діаграми процесу «Ідентифікація мешканця ЖК» (див. рис 2.1).

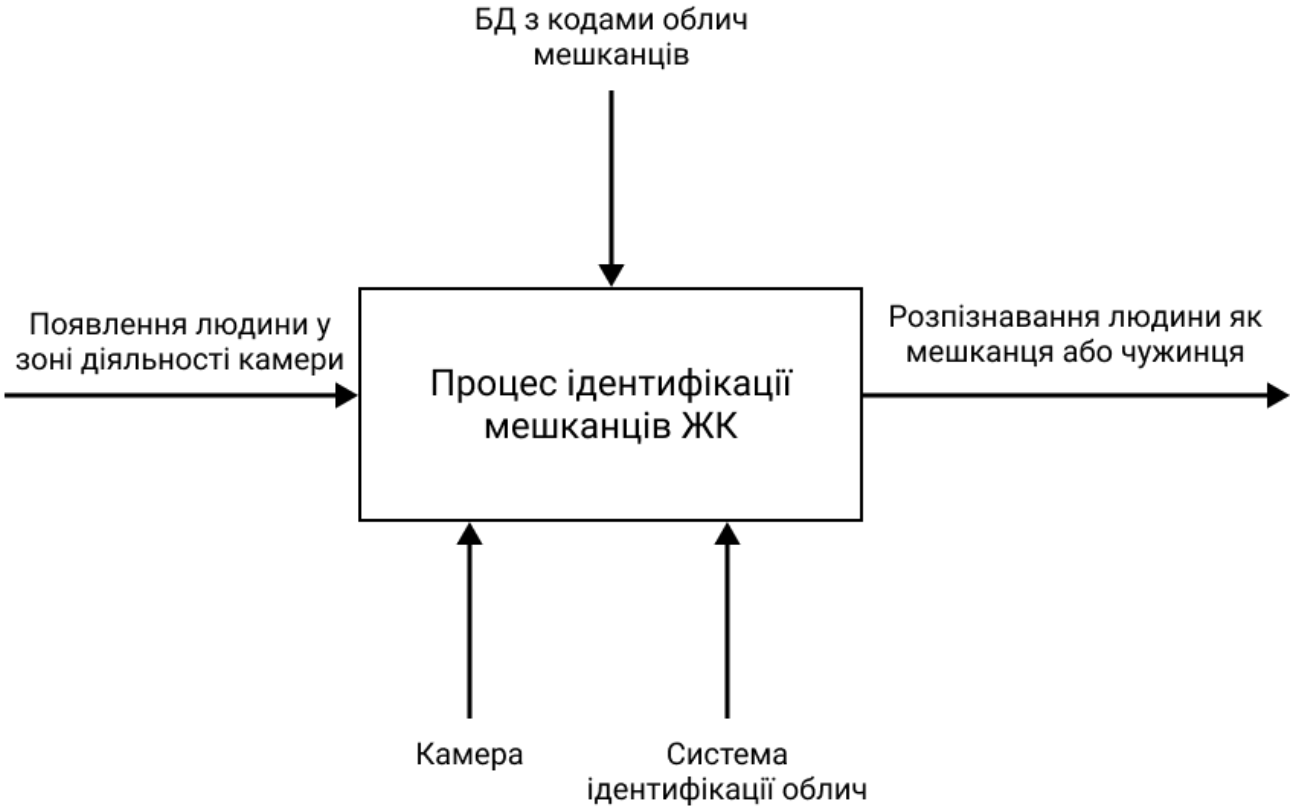

Рис 2.1 – Контекстна діаграма процесу «Ідентифікація мешканця ЖК»

Декомпозиція процесу «Система ідентифікації мешканців ЖК», що відображена на рис. 2.2. складається з наступних кроків:

Перший крок. Транслювання зображення з камери до прогами - зображення з камери постійно траслюється у реальному часі та постійно передає до програми зображення усього, що відбувається у її полі зору. Як тільки у кадрі з’являється людина, відбувається другий крок.

Другий крок. Розпізнавання положення облич на зображенні - після передачі зображення до програми, відбувається розпізнавання того, чи є на даному зображенні обличчя і якщо так, то знаходиться його точне місцеположення.

Третій крок. Переведення рис обличчя у ебеддінг. Ембеддінг (embedding vectors) - це процес, який безпосередньо вивчає відображення зображень обличчя в компактний евклідовий простір, де відстані безпосередньо представляють подібність обличчя[6], тобто це вектор чисел, що відображає унікальні параметри обличчя.

Четвертий крок. Далі, маючи ембеддінг щойно розпізнаного обличчя, програма здійснює порівняння даного ембеддінгу з ембеддінгами у наданій базі даних мешканців.

У результаті виконання усіх попередніх кроків, програма видає у як результат висновок про те, чи є дана людина мешканцем ЖК, або ні. Також робляться записи про всіх розпізнаних людей, чи то був мешканець, чи то був чужинець.

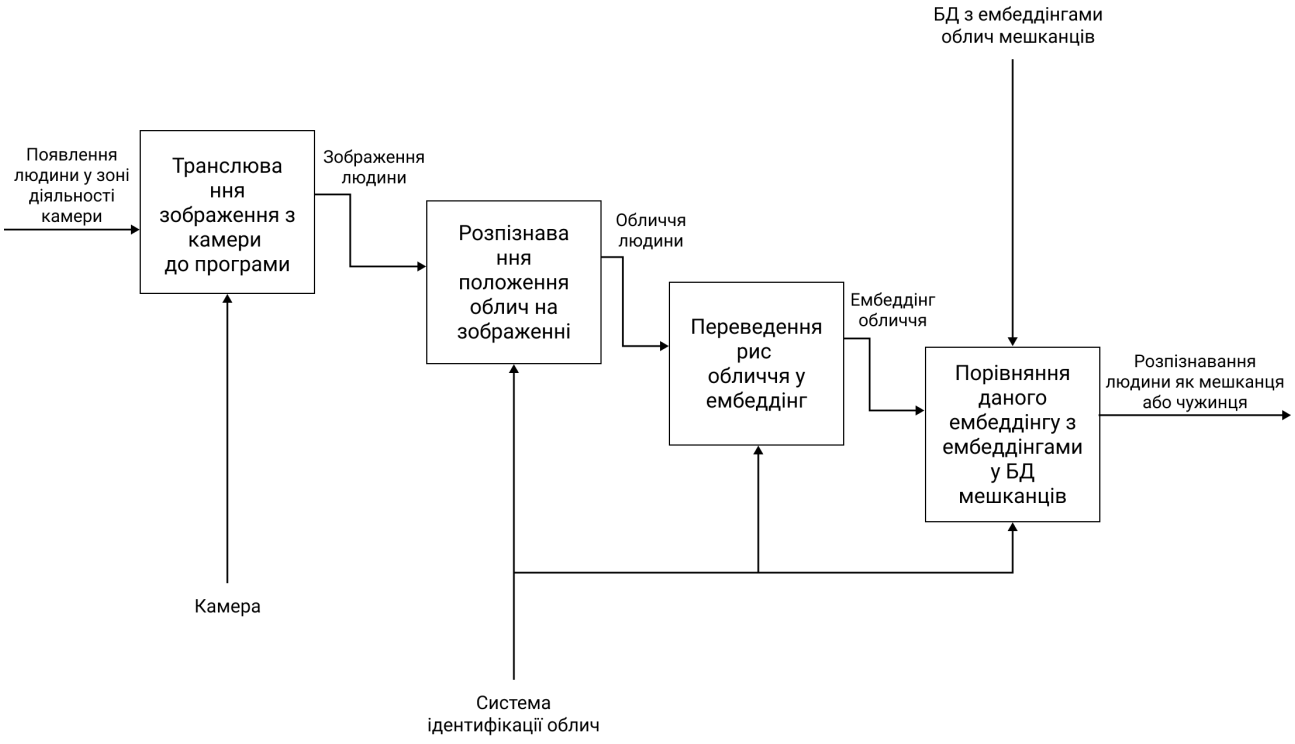

Рисунок 2.2 – Декомпозиція процесу «Система ідентифікації мешканців ЖК»

Відповідно до декомпозиції основного процесу програми можна визначити життєвий цикл ідентифікації облич мешканців ЖК (рис. 2.3).

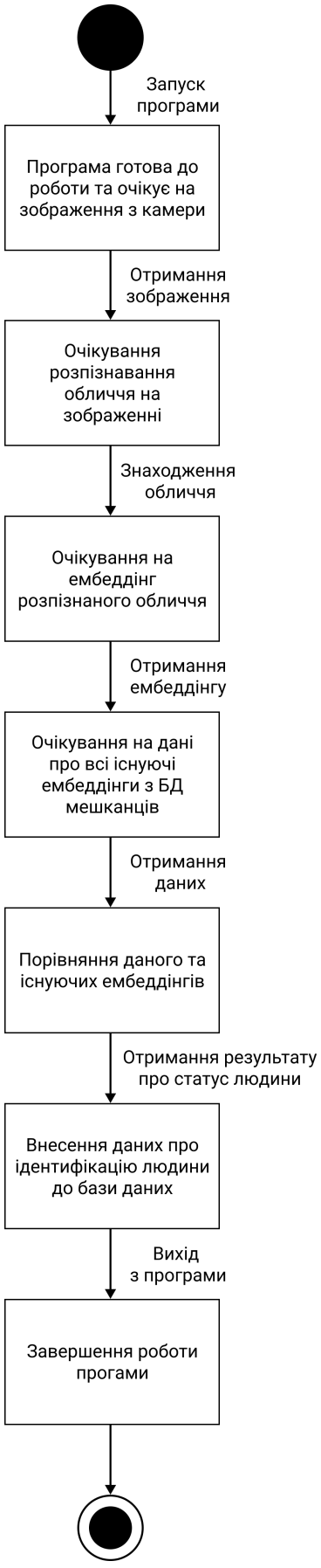

Рисунок 2.3 – Життєвий цикл програмного модулю

Процес ідентифікації обличчя людини складається із таких підпроцесів (див. рис. 2.4):

- Трансляція зображення
- Розпізнавання обличчя
- Ідентифікація обличчя
- Визначення статусу

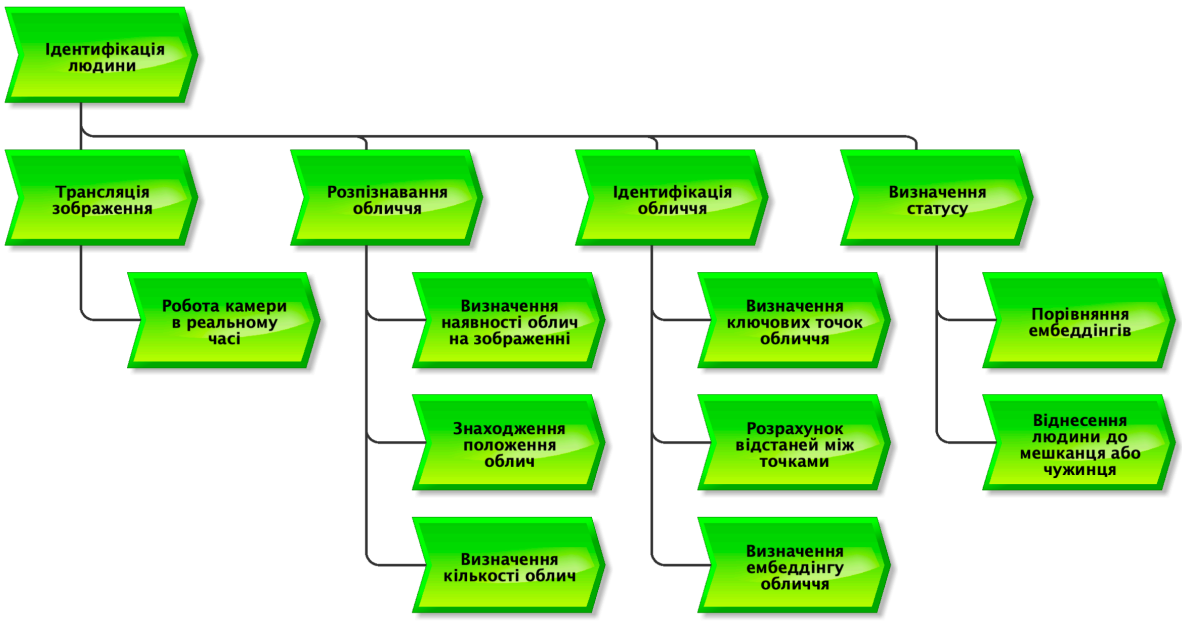

Рисунок 2.4 – Процес «Ідентифікації людини» у нотації VAD

Більш детальний процес роботи програмни у нотації EPC (Event-Driven Process Chain) відображено на рис. 2.5.

Бізнес-правила:

- програма працює тільки при наявності веб-камери;
- програма працює тільки при підкюченні веб-камери до ноутбуку.

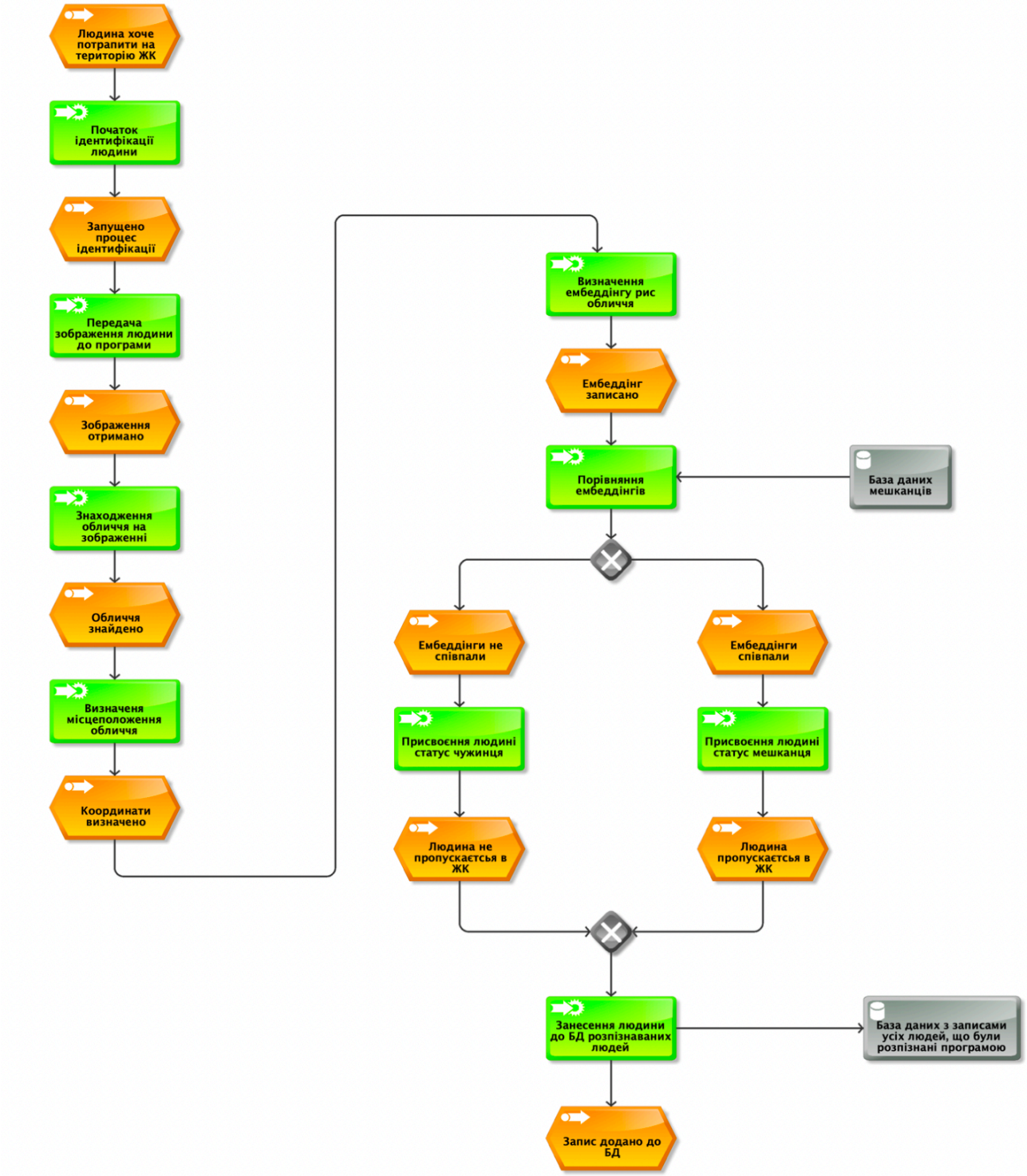

Рисунок 2.5 – Процес роботи програми у нотації EPC

2.2 Архітектура системи ідентифікації мешканців ЖК

Робота камери в реальному часі

Архітектура клієнт-сервер[12]

Архітектура «клієнт-сервер» визначає загальні принципи взаємодії в мережі, де є сервери, вузли-постачальники деяких специфічних функцій (сервісів) і клієнти (споживачі цих функцій).

Практичні реалізації такої архітектури називаються клієнт-серверними технологіями.

Дволанкова архітектура – розподіл трьох базових компонентів між двома вузлами (клієнтом та сервером). Дволанкова архітектура використовується у клієнт-серверних системах, де сервер відповідає на клієнтські запити безпосередньо та в повному обсязі (див. рис. 2.6).

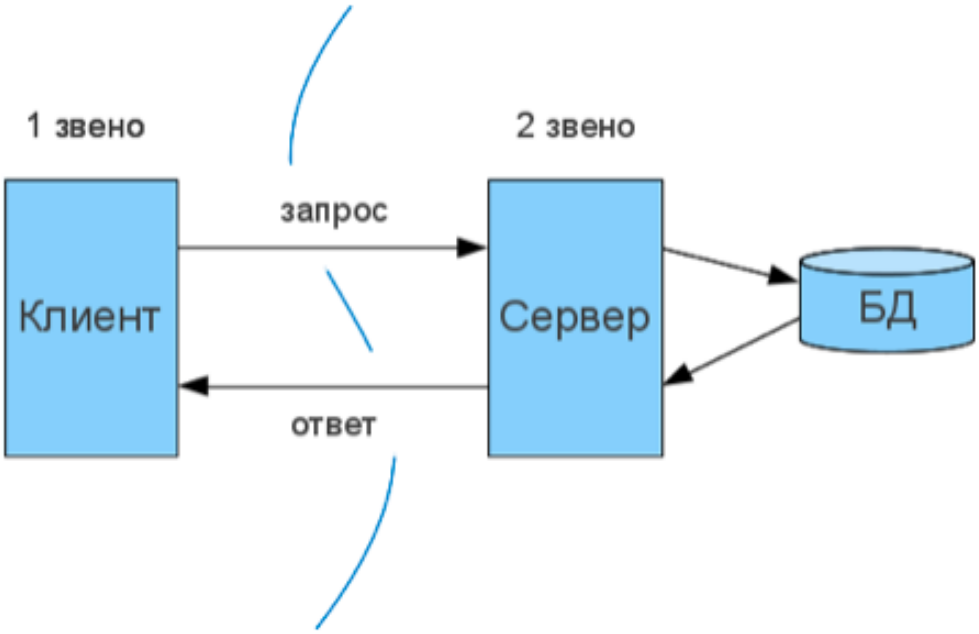

Рис 2.6 – Схема роботи дволанкової клієнт-серверної архітектури

Розташування компонентів на стороні клієнта або сервера визначає такі основні моделі їхньої взаємодії в рамках дволанкової архітектури:

- Сервер терміналів - розподілене подання даних.
- Файл-сервер — доступ до віддаленої бази даних та файлових ресурсів.

- Сервер БД - віддалене представлення даних.
- Сервер програм — віддалений додаток.
- Клієнт - це браузер, але зустрічаються і винятки (у тих випадках, коли один веб-сервер (ВС1) виконує запит до іншого (ВС2), роль клієнта грає веб-сервер ВС1). У класичній ситуації (коли роль клієнта виконує браузер) для того, щоб користувач побачив графічний інтерфейс програми у вікні браузера, останній повинен обробити отриману відповідь веб-сервера, в якому буде міститися інформація, реалізована із застосуванням HTML, CSS, JS). Саме ці технології "дають зрозуміти" браузеру, як саме необхідно "відмалювати" все, що він отримав у відповіді.

Веб-сервер – це сервер, який приймає HTTP-запити від клієнтів і видає HTTP-відповіді. Веб-сервер називають як програмне забезпечення, що виконує функції веб-сервера, так і безпосередньо комп'ютер, на якому це програмне забезпечення працює. Найбільш поширеними видами програмного забезпечення веб-серверів є Apache, IIS і NGINX. На веб-сервері функціонує додаток, що тестується, який може бути реалізований із застосуванням найрізноманітніших мов програмування: PHP, Python, Ruby, Java, Perl та ін.

База даних фактично не є частиною веб-сервера, але більшість програм просто не можуть виконувати всі покладені на них функції без неї, тому що саме в базі даних зберігається вся динамічна інформація програми (облікові, дані користувача та ін).

База даних — це інформаційна модель, що дозволяє впорядковано зберігати дані про об'єкт або групу об'єктів, які мають набір властивостей, які можна категоризувати. Бази даних функціонують під керівництвом систем управління базами даних (СУБД). Найпопулярнішими СУБД є MySQL, MS SQL Server, PostgreSQL, Oracle (все – клієнт-серверні).

Розпізнавання облич на зображенні[13][29](див. рис. 2.7).

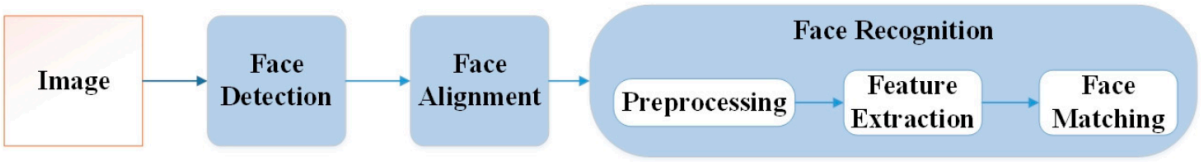

Рис 2.7 — Система розпізнавання обличчя

Програми розпізнавання облич використовують алгоритми та ML, щоб знаходити людські обличчя на більших зображеннях, які часто містять інші об’єкти, які не є обличчями, наприклад ландшафти, будівлі та інші частини людського тіла, як-от ноги чи руки. Алгоритми розпізнавання облич зазвичай починаються з пошуку людських очей – однієї з найпростіших функцій. Потім алгоритм може спробувати виявити брови, рот, ніс, ніздрі та райдужну оболонку. Як тільки алгоритм робить висновок, що він знайшов область обличчя, він застосовує додаткові тести, щоб підтвердити, що він насправді виявив обличчя.

Щоб забезпечити точність, алгоритми потрібно навчати на великих наборах даних, які містять сотні тисяч позитивних і негативних зображень. Навчання покращує здатність алгоритмів визначати, чи є обличчя на зображенні та де вони знаходяться.

Виявлення облич на знімках може бути складним через мінливість таких факторів, як поза, вираз обличчя, положення та орієнтація, колір шкіри та значення пікселів, наявність окулярів або волосся на обличчі, а також відмінності в посиленні камери, умовах освітлення та роздільній здатності зображення. Останні роки принесли прогрес у розпізнаванні облич за допомогою глибокого навчання, що дає перевагу значно перевершити традиційні методи комп’ютерного зору.

Гістограма спрямованих градієнтів (HOG)[14]

Навнет Далал і Білл Триггс представили функції гістограми орієнтованих градієнтів (HOG) у 2005 році. HOG — це простий і потужний дескриптор функцій. Він використовується не тільки для виявлення обличчя, але також широко використовується для виявлення об’єктів, таких як автомобілі, домашні тварини та фрукти. HOG надійний для виявлення об’єктів, оскільки форма об’єкта характеризується за допомогою локального розподілу градієнта інтенсивності та напрямку краю.

Принцип гістограми дескриптора орієнтованих градієнтів полягає в тому, що зовнішній вигляд і форму локального об’єкта в зображенні можна описати розподілом градієнтів інтенсивності або напрямків країв. Похідні зображення по x і y (градієнти) корисні, оскільки величина градієнтів велика навколо країв і кутів через різку зміну інтенсивності, і ми знаємо, що краї та кути містять набагато більше інформації про форму об’єкта, ніж плоскі області. Отже, гістограми напрямків градієнтів використовуються як ознаки в цьому дескрипторі.

Вектори ознак, отримані з дескриптора HOG, використовуються класифікатором, таким як SVM, для виявлення відповідного об’єкта (див. рис. 2.8).

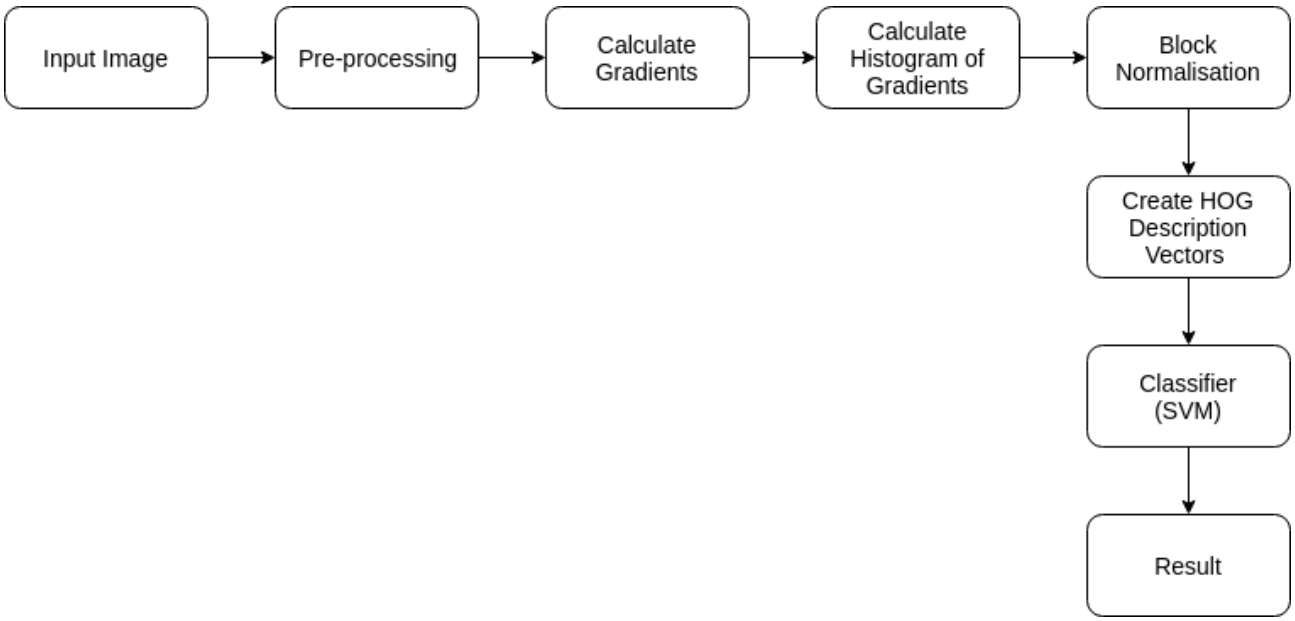

Рис 2.8 — Етапи виявлення об’єктів за допомогою HOG

Як працює гістограма орієнтованих градієнтів (HOG)?

Попередня обробка

Попередня обробка зображення включає нормалізацію зображення, але це абсолютно необов’язкове. Воно використовується для підвищення продуктивності дескриптора HOG.

Основна ідея HOG — поділ зображення на невеликі зв’язані клітинки (див. рис. 2.9).

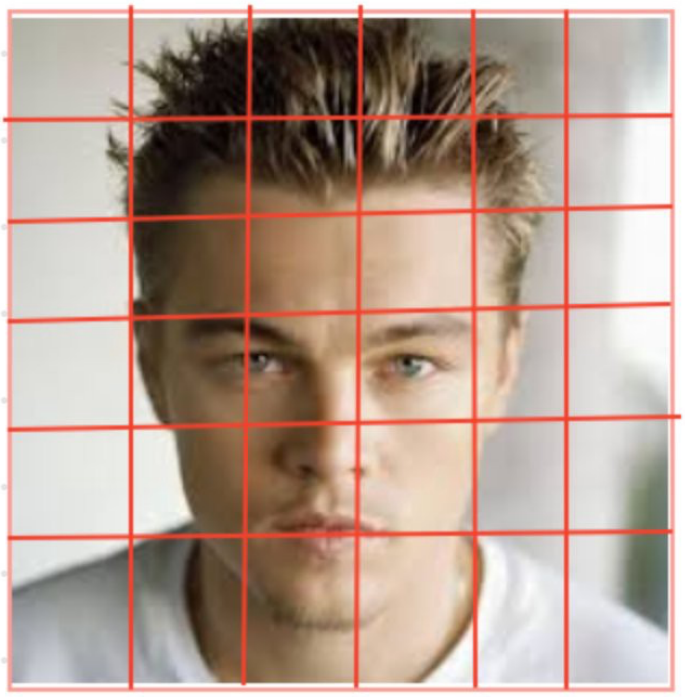

Рис 2.9 — Розділення зображення на невеликі клітинки

Обчислення гістограми для кожної клітинки (див. рис. 2.10).

Рис 2.10 — Обчислення гістограми кожної клітинки

Першим фактичним кроком у дескрипторі HOG є обчислення градієнта зображення в напрямку x і y.

Скажімо, піксель Q має значення навколо нього, як показано на рисунку 2.11.:

	100	
70	**Q**	120
	50	

Рис 2.11 — Приклад побудови градієнту навколо пікселя Q

Можна обчислити величину градієнта для Q в напрямках x і y таким чином (див. формули 2.1, 2.2):

$$G_x = 100 - 50 = 50 \quad (2.1)$$

$$G_y = 120 - 70 = 50 \quad (2.2)$$

Ми можемо отримати величину градієнта як (див. формули 2.3):

$$G = \sqrt{(G_x)^2 + G_y)^2} = 70.7 \quad (2.3)$$

І напрямок градієнта як (див. формули 2.4):

$$\theta = arctan(\frac{G_y}{G_x}) = 45^\circ \quad (2.4)$$

Розрахування гістограми градієнтів у клітинках 8×8:

- Зображення розбивається на блоки клітинок 8×8, і для кожного блоку клітинок 8×8 обчислюється гістограма градієнтів.
- По суті, гістограма являє собою вектор із 9 сегментів (чисел), що відповідають кутам від 0 до 180 градусів (з кроком 20 градусів).
- Значення цих 64 клітинок (8X8) збираються та сукупно додаються в ці 9 значень.
- Це по суті зменшує 64 значення до 9 значень.

На рисунку 2.12 показано, як це робиться. Обведений синім, піксель має кут 80 градусів і величину 2. Таким чином, він додає 2 до 5-ї комірки. Градієнт на пікселі, обведеному червоним кольором, має кут 10 градусів і величину 4. Оскільки 10 градусів є половиною від 0 до 20, піксель рівномірно розподіляється на два блоки.

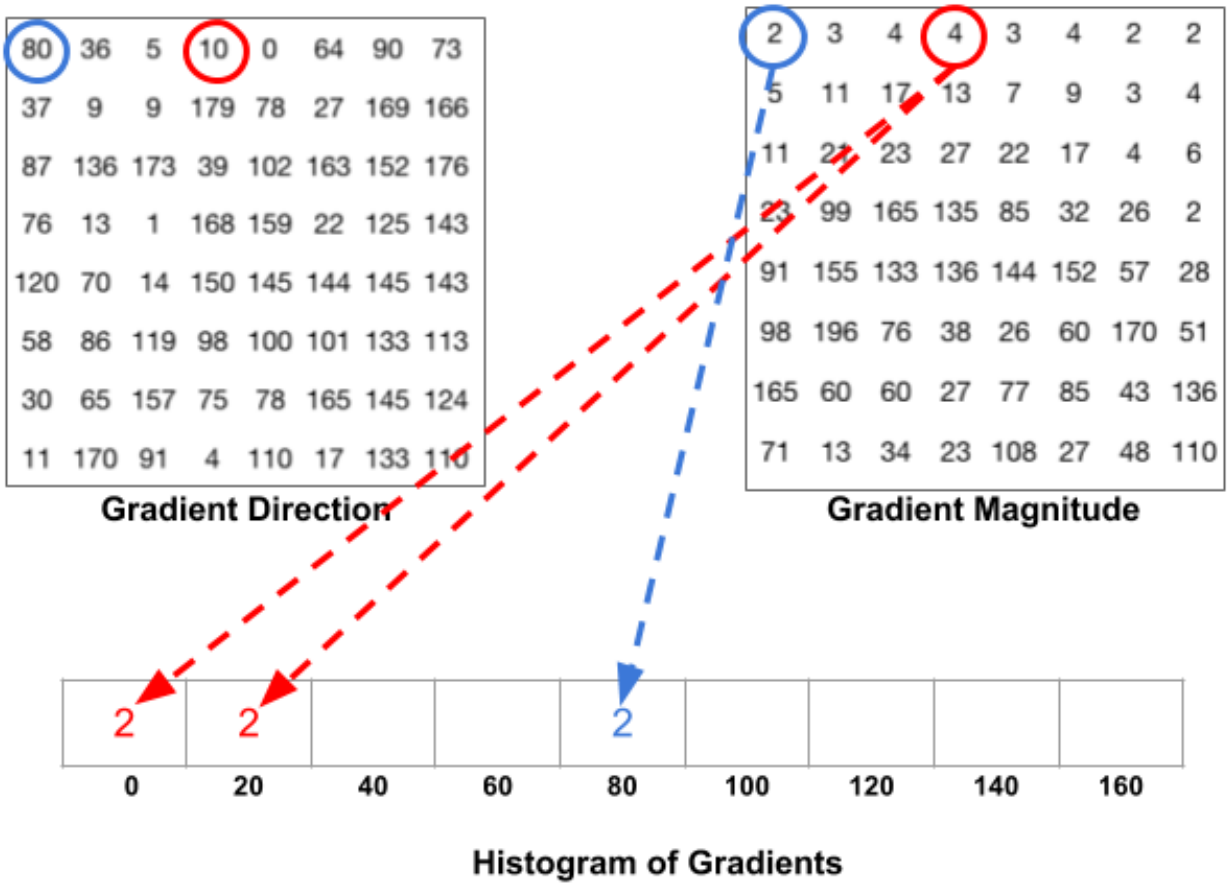

Рис 2.12 — Розбиття величини градієнта відповідно до напряму градієнта

Нормалізація блоку

Градієнт чутливий до загального освітлення. Якщо ми кажемо розділити/ помножити значення пікселя на деяку константу, щоб зробити його світлішим/ темнішим, величина градієнта зміниться, а також значення гістограми. Ми хочемо, щоб значення гістограми були незалежними від освітлення. Нормалізація здійснюється на векторі гістограми v всередині блоку. Можна використовувати одну з наступних норм:

- L1 норма
- L2 норма
- L2-Hys (обрізана норма L2 у низькому стилі)

Тепер ми могли б просто нормалізувати вектор гістограми 9×1, але краще нормалізувати блок більшого розміру 16×16. Блок розміром 16×16 має 4 гістограми (8×8 клітинок результатів для однієї гістограми), які можна об’єднати, щоб утворити вектор елемента розміром 36 x 1 і нормалізувати. Потім вікно розміром 16×16 переміщується на 8 пікселів, і для цього вікна обчислюється нормований вектор 36×1, і процес повторюється для зображення.

Розрахувати вектор дескриптора HOG

- Щоб обчислити остаточний вектор ознак для всього фрагмента зображення, вектори 36×1 об’єднуються в один гігантський вектор (див. рис. 2.13).
- Отже, скажімо, якщо було вхідне зображення розміром 64×64, то блок 16×16 має 7 позицій по горизонталі і 7 позицій по вертикалі.
- В одному блоці 16×16 ми маємо 4 гістограми, які після нормалізації об’єднуються, утворюючи вектор 36×1.
- Цей блок переміщується на 7 позицій по горизонталі та вертикалі, в сумі 7×7 = 49 позицій.
- Отже, коли ми об’єднуємо їх усі в один вектор посилення, ми отримуємо 36×49 = 1764 розмірний вектор.

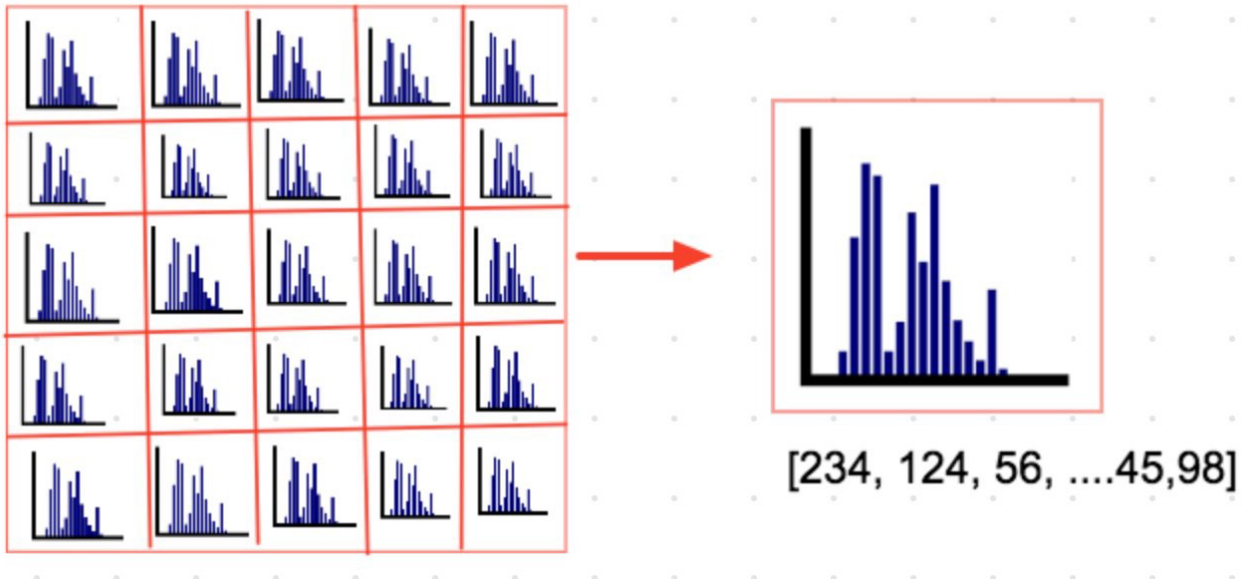

Рис 2.13 — Об’єднання невеликих гістограм в одну гістограму, яка є кінцевим вектором ознак

Цей вектор тепер використовується для навчання класифікаторів, таких як SVM, а потім для виявлення об’єктів (див. рис. 2.14).

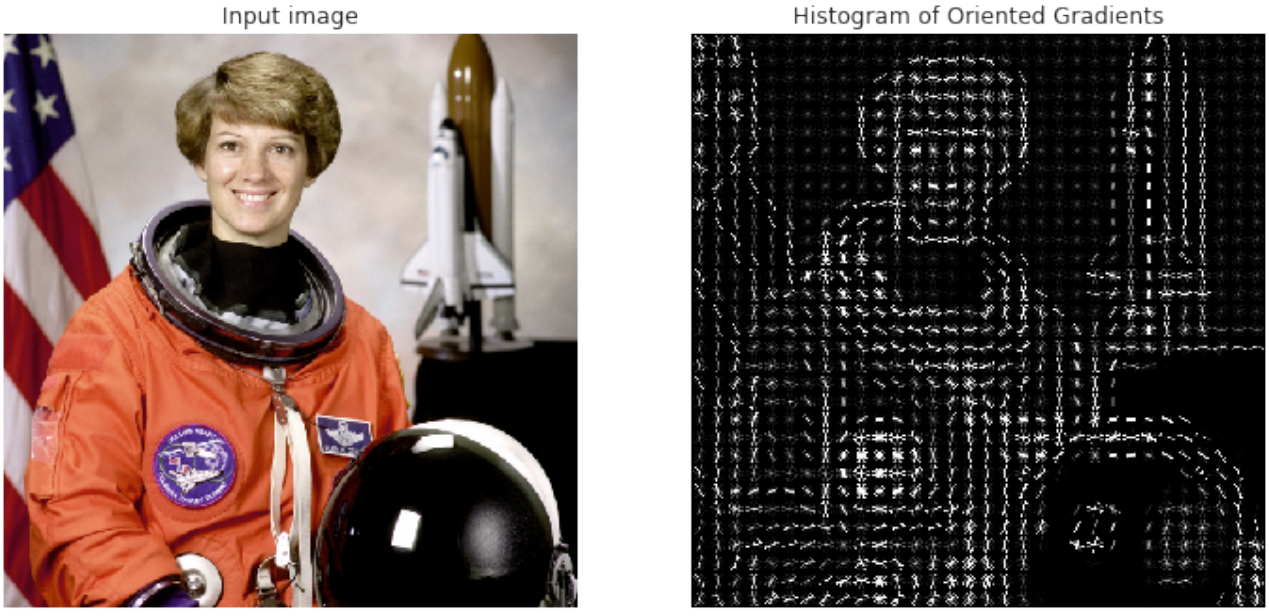

Рис 2.14 — Візуалізація особливостей HOG зображення астронавта Ейлін Коллінз

Визначення ембеддінгів обличчя[15]

На самому базовому рівні моделі розпізнавання обличчя виконують такі кроки:

1. Вхідне зображення подається в алгоритм.
2. Алгоритм створює ембеддінг обличчя для вхідного зображення.
3. Алгоритм порівнює ембеддінг обличчя вхідного зображення з ембеддінгом відомих облич у базі даних.

Кожен підхід має різний метод навчання, і дослідники часто коригують або додають елементи до встановлених методів у цій галузі. Однак більшість систем використовують триплетні втрати для навчання алгоритму. Що стосується розпізнавання обличчя, то втрата триплетів працює шляхом подачі в алгоритм трьох зображень.

Два зображення – це особа А, а решта – особа В. Алгоритм створює ембеддінг обличчя кожного зображення, а потім порівнює їх.

Після порівняння мережа буде трохи відкоригована, щоб ембеддінг особи А було більш схоже один на одного, ніж ембеддінг особи В. Згодом це навчає алгоритм використовувати вимірювання обличчя, які дозволяють йому точно класифікувати зображення однієї і тієї ж людини як схожі один на одного. Потім цей процес повторюється сотні тисяч або навіть мільйони разів. Нарешті, мережа повинна мати можливість створювати точні ембеддінги обличчя для облич, яких вона ніколи не бачила.

Перш ніж алгоритм зможе порівняти обличчя, необхідно перетворити зображення облич у дані, які може зрозуміти алгоритм. Для цього система обчислює вимірювання на основі рис обличчя та орієнтирів. На рис. 2.15 зображено 68 орієнтирів обличчя, також відомих як ключові точки обличчя.

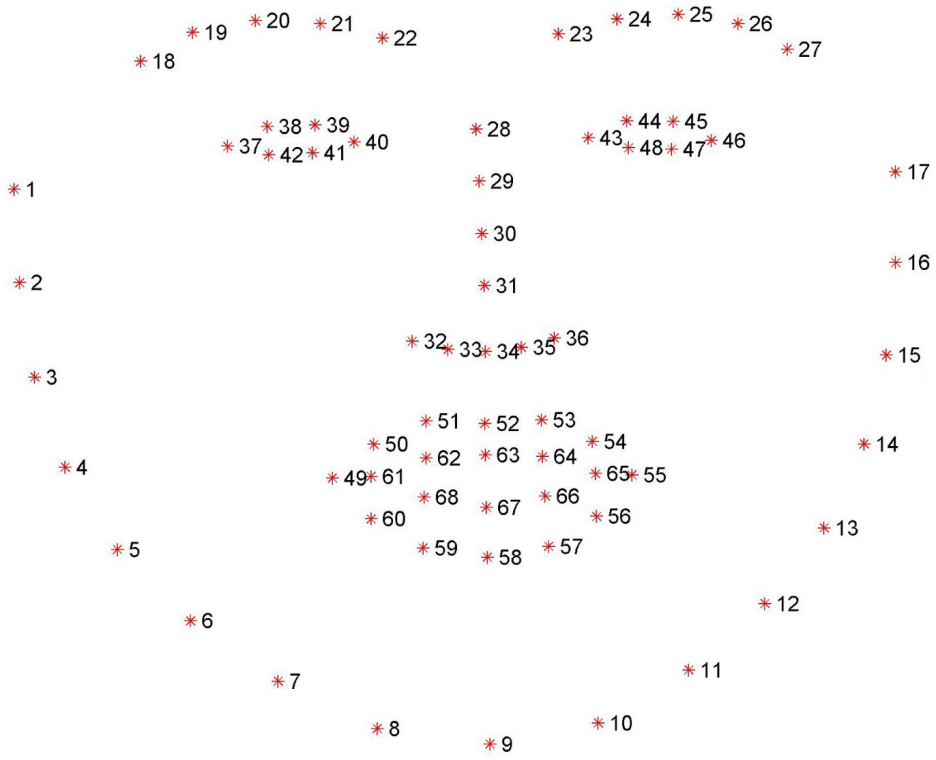

Рис 2.15 — Зображення ключових точок обличчя в ембеддінгу

Створюючи ембеддінг обличчя, зображення обличчя перетворюється в числові дані. Ці дані потім представляють у вигляді вектора в прихованому семантичному просторі. Чим ближче вбудовування знаходяться один до одного в латентному просторі, тим більша ймовірність, що вони належать одній людині.

Порівняння ембеддінгів обличчя

Для порівняння двох ембеддінгів використовується Евклідова відстань. Відстань показує, наскільки схожі обличчя.

У математиці евклідова відстань між двома точками в евклідовому просторі — це довжина відрізка прямої між двома точками. Його можна обчислити з декартових координат точок за допомогою теореми Піфагора, тому іноді його називають піфагоровою відстанню.

Нехай у n-мірному просторі задані дві точки: p(p1,p2,…pn) nf q(q1,q2,… qn). Тоді евклідова відстань між ними обчислюється за такою формулою:

$$d_{pq} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2},$$

$d_{pq} = /sqrt$

Приклад використання Евклюдової відстані можна побачити на рисунку 2.16.

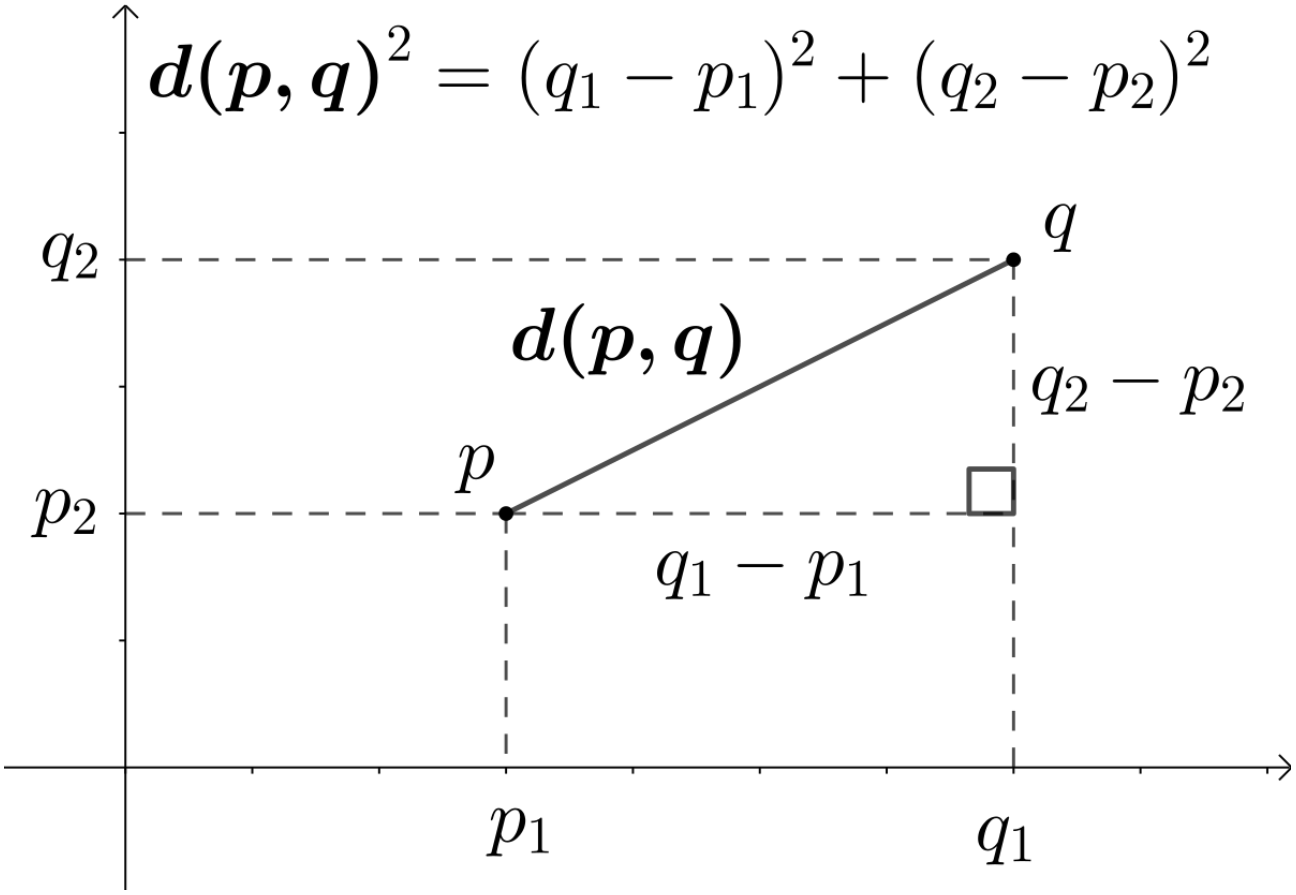

Рис 2.16 — Представлення Евклідової відстані між векторами p та q

2.3 Інформаційна складова проекту - база даних

Реляційні бази даних[16]

Реляційна база даних — це саме те, що випливає з її назви: інструмент для зберігання різних типів інформації, пов’язаних один з одним різними способами. Наприклад, реляційна база даних для інтернет-магазину може зберігати дані клієнтів, а також відповідну інформацію, таку як їхні різні адреси, списки побажань, замовлення тощо.

Реляційні бази даних існують протягом тривалого часу, використовуються в широкому діапазоні випадків використання і залишаються популярним вибором навіть сьогодні, оскільки вони пропонують зрілу, стабільну, перевірену технологію, яка стає все більш надійною та простою у використанні. Можна очікувати, що вони керують даними для додатків електронної комерції, управління запасами, нарахуванням заробітної плати, даними клієнтів тощо.

У перші роки розвитку обчислювальної техніки кожна програма зазвичай зберігала свої дані у своєму унікальному форматі. Зберігання даних різними способами ускладнювало підтримку систем та їх інтеграцію за потреби. Модель реляційної бази даних виникла як спосіб стандартизації того, як розробники можуть послідовно працювати з даними на високому рівні, залишаючи програмному забезпеченню деталі основних операцій і форматів зберігання.

Реляційні моделі, як правило, організують дані в таблиці, що складаються з рядків і стовпців, подібно до програм електронних таблиць. Нижче показано, як реляційна база даних може відстежувати дітей у літньому таборі (див. рис. 2.17). Кожен рядок містить інформацію для конкретного туриста, включаючи унікальний ідентифікаційний номер разом із ім’ям, віком та улюбленою їжею дитини.

ID	Name	Age	Favorite Food
1	Lisa	11	Pizza
2	Sarah	8	Ice cream
3	Jenna	9	Pizza

Рис 2.17 — Приклад реляційної бази даних

Призначення унікального ідентифікатора може бути не відразу очевидним, але саме це дає змогу іншим таблицям у базі даних відстежувати будь-які пов’язані дані — наприклад, дати відвідування, заходи, сертифікати, нагороди тощо — шляхом включення ідентифікаційного номера кемпера у підпорядкованих таблицях. Здатність об’єднувати багато різних таблиць за такими ідентифікаторами – це те, що робить реляційні бази даних реляційними.

На відміну від цього, системи управління реляційними базами даних (скорочено СУРБД) є програмними системами, які керують реляційними базами даних. У той час як сама реляційна база даних являє собою певну колекцію концептуальних таблиць, які зберігаються на носіях даних у різних форматах, СУРБД — це те, що дає змогу взаємодіяти з даними, не знаючи деталей. Наприклад, СУРБД дає змогу адмініструвати декілька реляційних баз даних на одному або кількох серверах і приймати команди для створення, читання, оновлення або видалення даних (так звані операції «CRUD») у таблицях.

Реляційні бази даних абстрагують операції через так звану мову структурованих запитів (SQL). SQL стандартизовано в системах СУРБД, щоб виконувати операції CRUD (та інші) узгодженим чином, тому ті самі оператори SQL можуть виконуватися будь-якою СУРБД, яка підтримує стандарт.

Переваги реляційних баз даних:

• З точки зору розвитку, реляційні бази даних представляють більш структурований погляд на світ. Реляційна структура добре вписується в моделювання реальних бізнес-суб’єктів (наприклад, клієнтів, замовлення, каталог, товари тощо) і ділових відносин, які існують між ними.

• Самі оператори SQL часто досить прості, щоб їх можна було прочитати людиною, і знижують бар’єр для входу для розробників додатків. . Враховуючи свій вік, SQL став звичайною мовою, знайомою багатьом розробникам.

• Тим не менш, однією з найбільш значущих переваг, які надають реляційні бази даних, є гарантії «ACID», головна цінність для критично важливих для бізнесу операцій. «ACID» означає атомарність, послідовність, ізоляцію та довговічність, усі вони відносяться до важливих властивостей транзакції, термін, який використовується для опису одиниці роботи з базою даних.

• Атомарність — це та властивість транзакції, яка гарантує, що це «все або нічого». Тобто транзакція або виконується як повна одиниця роботи, виконуючи різні операції CRUD (та інші) і реєструється як єдине фіксування в реляційній базі даних, або зазнає невдачі аналогічним чином. У разі збою атомарність вимагає, щоб будь-які частини вже виконаної транзакції були повністю відкатані, залишаючи реляційну базу даних точно такою, якою вона була до початку транзакції.

• Узгодженість йде поруч з атомарністю, оскільки вона вимагає, щоб дані завжди були в узгодженому внутрішньому стані, коли починається транзакція і коли вона закінчується. Усі зміни в базових даних для таблиць та інших структур мають бути повними та правильними, щоб кожна успішна транзакція «переміщувала» дані бази даних з одного чітко визначеного дійсного стану в інший без нічого не вирішеного або ще не зробленого.

• Наступні переваги - ізоляція та довговічность. Коли багато різних користувачів одночасно взаємодіють з однією і тією ж реляційною базою даних, кожен з яких намагається здійснити різні транзакції з потенційно однаковим набором таблиць, важливо, щоб кожен користувач бачив лише свій погляд на дані. Зчитування даних із таблиці, до якої одночасно намагається записати інший користувач, має відображати наявні дані, а не зміни, щоб уникнути «витоку» даних із транзакції, яка ще може завершитися невдачею. Ізоляція гарантує, що «погляд» у всій системі залишається послідовним і незмінним під час інших операцій, поки дана транзакція виконується, як ніби вона виконується сама по собі.

• Довговічність вимагає, щоб деталі будь-якої успішно здійсненої транзакції постійно записувалися після завершення. Зміни, що лежать в основі подання даних, мають бути застосовані повністю, щоб не відбулася втрата — навіть у разі збоїв у кешуванні, жорсткому диску чи інших системних збоях. Зазвичай це включає в себе незмінну реєстрацію деталей того, що, коли і ким було змінено, а також для майбутніх потреб аудиту.

Чому SQL?[17]

- Бази даних SQL є реляційними, бази даних NoSQL нереляційними.
- Бази даних SQL використовують структуровану мову запитів і мають попередньо визначену схему. Бази даних NoSQL мають динамічні схеми для неструктурованих даних.
- Бази даних SQL мають вертикальне масштабування, а бази даних NoSQL горизонтальне.
- Бази даних SQL засновані на таблицях, тоді як бази даних NoSQL є сховищами документів, ключів і значень, графіків або широких стовпців.
- Бази даних SQL краще для багаторядкових транзакцій, тоді як NoSQL краще для неструктурованих даних, таких як документи або JSON.

Бази даних SQL використовують структуровану мову запитів і мають попередньо визначену схему для визначення та маніпулювання даними. SQL є однією з найбільш універсальних і широко використовуваних мов запитів, що

робить його безпечним вибором для багатьох випадків використання. Він ідеально підходить для складних запитів. Однак SQL може бути занадто обмежуючим. Перш ніж працювати з ним, потрібно використовувати попередньо визначені схеми, щоб визначити структуру даних. Усі дані повинні мати однакову структуру. Цей процес вимагає серйозної попередньої підготовки.

Бази даних SQL у більшості ситуацій вертикально масштабовані. Тобто можна збільшити навантаження на один сервер, додавши більше ємності CPU, RAM або SSD.

Бази даних SQL засновані на таблицях, тоді як бази даних NoSQL є сховищами документів, ключів і значень, графіків або широких стовпців.

Деякі приклади баз даних SQL включають MySQL, Oracle, PostgreSQL і Microsoft SQL Server. Бази даних SQL краще для багаторядкових транзакцій, тоді як NoSQL краще для неструктурованих даних, таких як документи або JSON.

Отже, у даному розділі було проведено функціональний та бізнес-аналіз системи, були побудовані котекстні діаграми, діаграми у нотації VAD, EPC діаграми та визначено життєвий цикл програми. Було також проаналізовано та визначено методи та алгоритми для використання у подальшому при побудові архітектури даної програми.

РОЗДІЛ 3. ТЕХНОЛОГІЧНІ ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ СИСТЕМИ ІДЕНТИФІКАЦІЇ МЕШКАНЦІВ ЖИТЛОВОГО КОМПЛЕКСУ

3.1 Програмне забезпечення системи ідентифікації мешканців ЖК

Для коректної роботи даної системи ідентифікації мешканців ЖК потрібен комп'ютер з будь-якою операційною системою, наприклад Windows, MacOS або Linux. Такий комп'ютер буде виконувати функцію сервера після деяких налаштувань. Також до комп'ютера повинна бути підключена веб-камера.

Комп'ютер-сервер має бути добре оснащений з точки зору комплектуючих. Рекомендовані вимоги до компонентів: не менше 8 Гб оперативної пам'яті, процесор Intel I5 8000+ серії або схожі, а також жорсткий диск

Для реалізації серверу в проекті використовується мова програмування — Python. Для запуску серверу необхідно налаштувати віртуальне оточення з такими залежностями (потрібно встановити наступні бібліотеки):

- FastAPI[18] — це сучасний, швидкий (високопродуктивний) веб-фреймворк для створення API на Python 3.6+ на основі стандартних підказок типу Python. Він є швидким (дуже висока продуктивність, нарівні з NodeJS і Go (завдяки Starlette і Pydantic). Один з найшвидших доступних фреймворків Python), простий у використанні. Написаний на основі відкритих стандартів для API: OpenAPI (раніше відомих як Swagger) і JSON Schema.
- Uvicorn[19] — це реалізація Asynchronous Server Gateway Interface (ASGI) веб-сервера для Python. Донедавна Python не мав мінімального низькорівневого інтерфейсу сервера/програми для асинхронних фреймворків. Специфікація ASGI заповнює цей пробіл і означає, що тепер ми можемо розпочати створення загального набору інструментів, які можна використовувати в усіх асинхронних платформах. Зараз Uvicorn підтримує HTTP/1.1 і WebSockets.
- Starlette[20] — це зручний фреймворк/набір інструментів Asynchronous Server Gateway Interface (ASGI), який ідеально підходить для створення

асинхронних веб-сервісів на Python. Він готовий до великих проектив і дає вам можливість підтримки WebSocket, виконання фонові завдань, вікористовувати тестовий клієнт, створений на основі запитів, підтримки ORS, GZip, статичних файлів, підтримки сеансів і файлів cookie. Модульність, на якій розроблено Starlette, сприяє створенню компонентів для повторного використання, які можна спільно використовувати між будь-якою структурою ASGI. Це має створити екосистему спільного проміжного програмного забезпечення та програм, які можна монтувати. Чистий поділ API також означає, що легше зрозуміти кожен компонент окремо.

• Websockets[21] — це сучасна технологія, що дозволяє відкрити постійне двонаправлене з'єднання між браузером користувача і сервером. За допомогою його API ви можете надіслати повідомлення на сервер та отримати відповідь без виконання http запиту, причому цей процес буде подійно-керованим. Програмування веб-сокету віконується як на стороні веб-серверу, так і на стороні кліенту, так як обидві сторони повинні “розуміти” та виконувати прі надходенні повидомлення.

• OpenCV[22][28] — Python бібліотека для роботи з зображеннямі та відео. OpenCV створено для забезпечення загальної інфраструктури для додатків комп’ютерного зору та для прискорення використання машинного сприйняття в комерційних продуктах. Бібліотека має понад 2500 оптимізованих алгоритмів, які включають повний набір як класичних, так і найсучасніших алгоритмів комп’ютерного зору та машинного навчання. Ці алгоритми можна використовувати для ідентифікації об’єктів, класифікації дій людини на відео, відстеження рухів камери, відстеження рухомих об’єктів, вилучення 3D-моделей об’єктів, створення тривимірних хмар точок із стереокамер, з’єднання зображень разом для отримання високої роздільної здатності. зображення всієї сцени, знаходити схожі зображення з бази даних зображень, видаляти червоні очі із зображень, зроблених за допомогою спалаху, слідкувати за рухами очей, розпізнавати пейзаж і встановлювати маркери для накладання на них доповненої реальності тощо.

• Dlib/face_recognition[23] — це сучасний набір інструментів, що містить алгоритми машинного навчання та інструменти для створення складного програмного забезпечення для вирішення реальних проблем. Він використовується як у промисловості, так і в наукових колах у широкому діапазоні областей, включаючи робототехніку, вбудовані пристрої, мобільні телефони та великі високопродуктивні обчислювальні середовища. Ліцензування з відкритим кодом дозволяє використовувати його в будь-якій програмі безкоштовно.

• SQLAlchemy[24] — це набір інструментів Python SQL та Object Relational Mapper, який надає розробникам додатків повну потужність та гнучкість SQL. SQLAlchemy надає повний набір добре відомих шаблонів збереження на рівні випуску у продакшн, розроблених для ефективного та високопродуктивного доступу до бази даних, адаптованих до простої мови домену Pythonic. Реляційно-орієнтована система запитів, яка чітко розкриває весь спектр можливостей SQL, включаючи об’єднання, підзапити, кореляцію та майже все інше, з точки зору об’єктної моделі. Написання запитів за допомогою ORM використовує ті ж методи реляційної композиції, які ви використовуєте при написанні SQL.

• Jinja2[25] — це швидкий, розширюваний механізм створення HTML шаблонів. Спеціальні заповнювачі в шаблоні дозволяють писати код, схожий на синтаксис Python. Потім шаблону передаються дані для відтворення остаточного документа. Бібліотека реалізує такий функціонал: успадкування та включення шаблону, розширювані фільтри, тести, функції та навіть синтаксис. Філософія Jinja полягає в тому, що, хоча логіка програми належить Python, якщо це можливо, вона не повинна ускладнювати роботу дизайнера шаблонів, надто обмежуючи функціональність.

• Pydantic[26] — це корисна бібліотека для аналізу та перевірки даних. Вона приводить типи введення до оголошеного типу (використовуючи підказки типу), накопичує всі помилки за допомогою ValidationError, а також добре задокументована, завдяки чому вона зручна в освоєнні.

Окрім штучного оточення Python[27], необхідно встановити будь-який веб-браузер, наприклад Mozilla Firefox, Google Chrome або Opera. Це необхідно, тому що веб-браузер одразу після встановлення вміє корректно відображати HTML, CSS, а також виконувати JavaScript код.

Клієнта частина системи реалізована мовою програмування JavaScript, адже вона є сучаною та підтримує усі необхідні бібліотеки, серед них: підтримка WebSocket технології (все було описано вище), та можливість надсилання HTTP запросів, що дає можливість використовувати API серверу.

3.2 Функціонал та інтерфейс системи ідентифікації мешканців ЖК

Система ідентифікації мешканців ЖК працює на основі дволанкової клієнт-серверної архітектури. Це означає, що на сервері (комп'ютері) працює веб-сервер, сервер бази даних, а також знаходяться файли, що відповідають за поведінку клієнтної частини системи.

Веб-сервер має наступні функції:

- Регламентує відображення веб-сторінок за певними посиланнями
- Відповідає за роботу з транзакціями до бази даних
- Створює та типізує об'єкти для HTTP відповідей
- Оброблює та перенаправляє HTTP запити на API частину серверу
- Виконує запити, отримані від API
- Налаштовує поведінку WebSocket та управляє підключеннями до нього

Інтерфейс даної системи представляє собою веб-сайт з усім функціоналом, необхідним для коректної роботи продукту.

Сервер Системи ідентифікації мешканців ЖК (надалі просто Система) запускається використовуючи бібліотеку Uvicorn. За стандартними налаштуваннями сервер запускається на локальній IP адресі http://127.0.0.1:8000. Веб-платформа Системи ідентифікації мешканців ЖК складається з декількох сторінок.

Головна сторінка веб-сайту

На головній сторінці сервісу з ідентифікації мешканців ЖК знаходиться короткий опис щодо цілей проекту, а також можливість протестувати його у реальному часі. Головна сторінка виглядає наступним чином (див. рис. 3.1).

Головна сторінка містить у собі хедер із двома посиланнями - на "Головну сторінку", та на "Адмін панель". Далі на веб-сторінці дуже коротко описані головні принципи роботи всієї Системи. Знизу знаходиться вікно "Приклад ідентифікації", а також ще одне посилання на "Адмін панель", для того, аби додати мешканця у базу даних Системи. Код інтерфейсу веб-сервісу наведено у додатку Е.

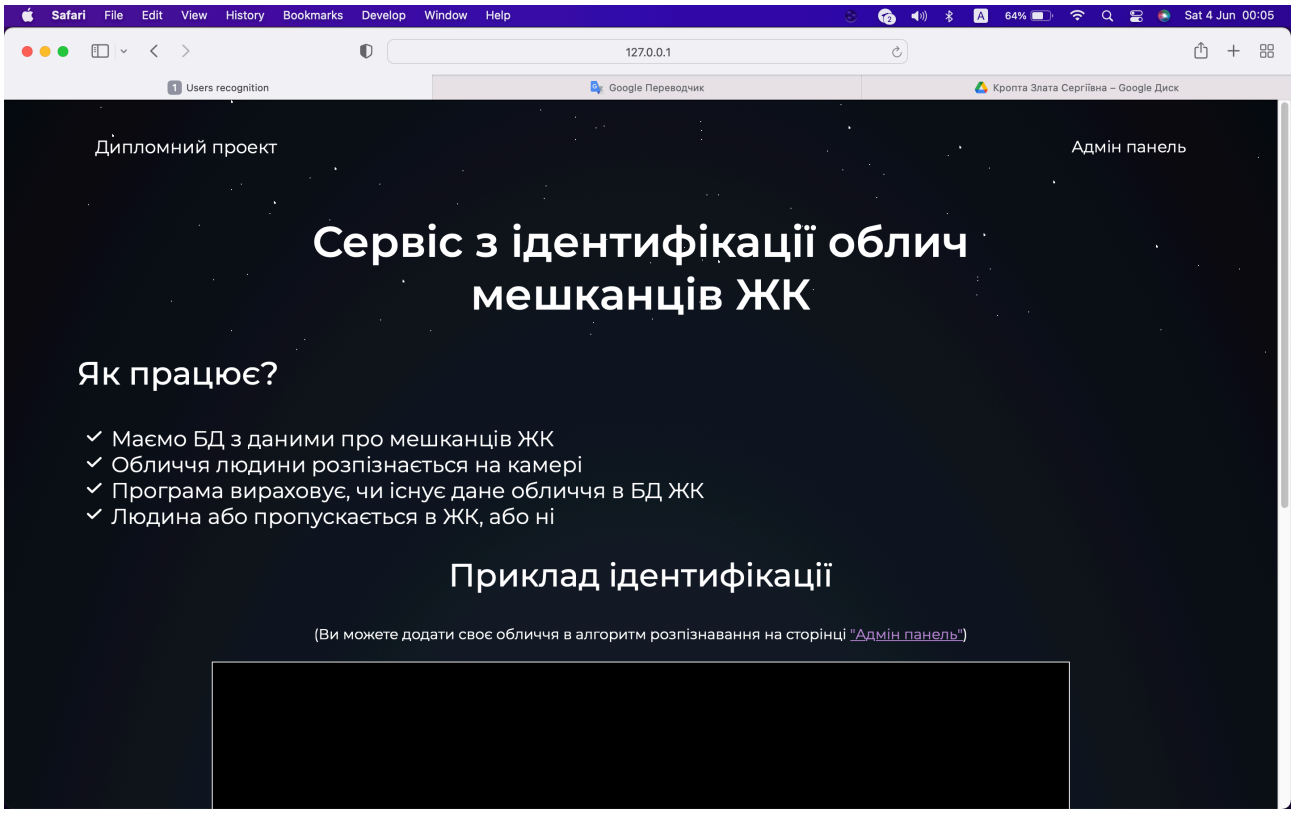

Рис 3.1 — Вигляд головної сторінки сервісу з ідентифікації

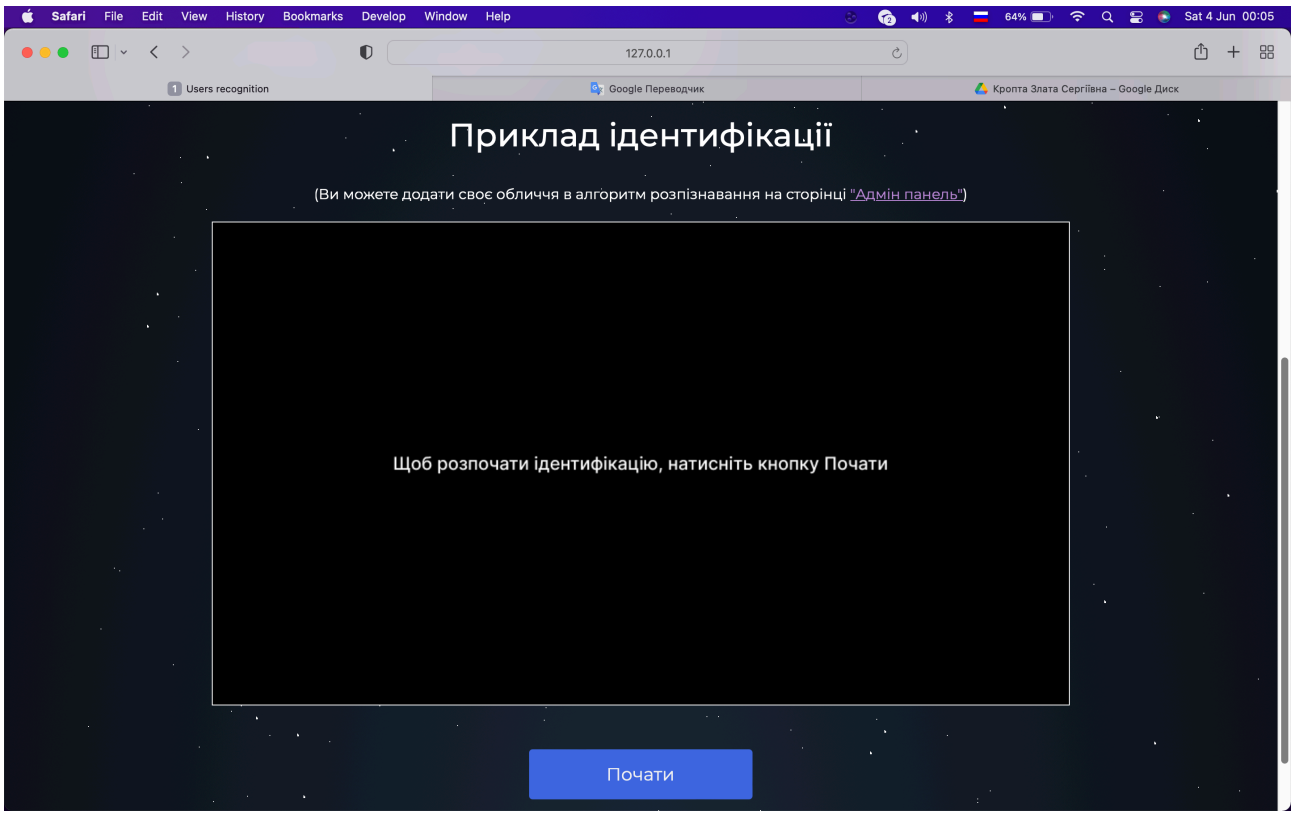

Рис 3.1 — Вигляд головної сторінки сервісу з ідентифікації(продовження)

При натисканні на кнопку "Почати" виконується JavaScript функція (код можна подивитися в додатку Д), що виконує наступні інструкції:

- Відкрити веб-сокет за посиланням ws://127.0.0.1:8000/ws_video/2. Це стандартне посиляння для веб-сокету, окрім останньої змінної. На місці двійкі також може бути одиниця, таким чином обирається функціонал вебсокету. При відкритті підключення, що закінчується одиницею, веб-сокет буде просто надсилати зображення з вебкамери. Якщо посилання закінчується на двійку, веб-сокет будет повертати зображення із розпізнаними обличчями на ньому. Саме тому, у даному випадку стоїть двійка, адже необхідно показати весь функціонал Системи.
- Створити івент-лісенер. Завдання лісенера - очікувати повідомлення від веб-сокета. В нашому випадку, веб-сокет буде надсилати декілька разів на секунду закодоване у строку байтів зображення з вебкамери. При отриманні повідомлення, а саме закодованого зображення, функція перетворює зображення функцією URL.createObjectURL() у формат посилання задля

того, щоб далі вставити такий об’єкт на html сторінку у тег <canvas>, як джерело зображення, тобто source. Саме таким чином, від 15 до 30 разів на секунду, відбувається передача зображення від вебкамери на сторінку веб-платформи.

Запустити даний севріс можна, натиснувши на кнопку «Почати» (додаток Д), тим самим увімкнеться камера і відразу запуститься алгоритм розпізнавання та ідентифікації облич (код знаходиться в додатках Б та Г).

У даному випадку є два варіанти перебігу подій:

1. Розпізнане обличчя наявне у базі даних, а отже, дана людина — це мешканець ЖК, алгоритм визнає її «своєю» та вона запускається до ЖК (див. рис. 3.2).

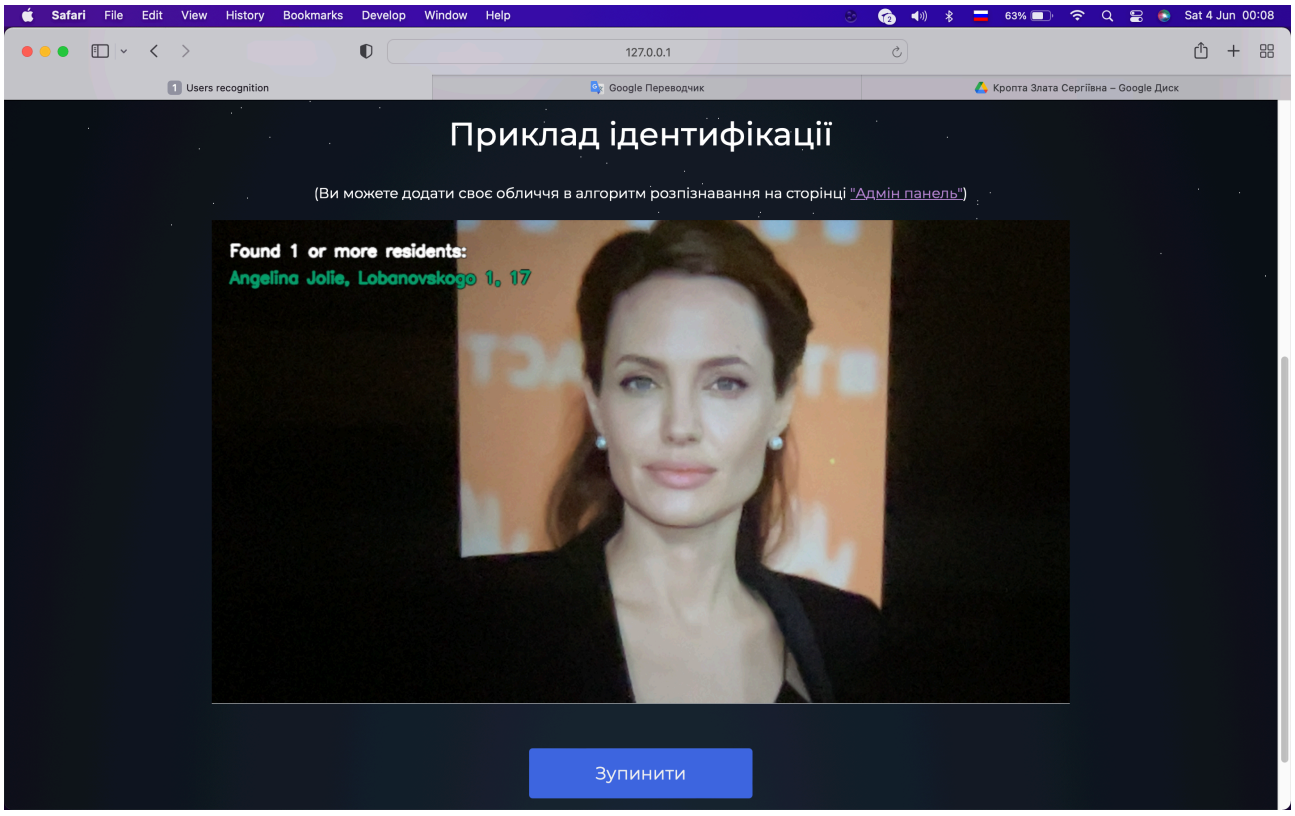

Рис 3.2 — Програма розпізнає мешканця ЖК

2. Розпізнаного обличчя немає у базі даних мешканців, а отже, алгоритм не ідентифікує дану людину як мешканця ЖК та не пропускає її (див. рис. 3.3).

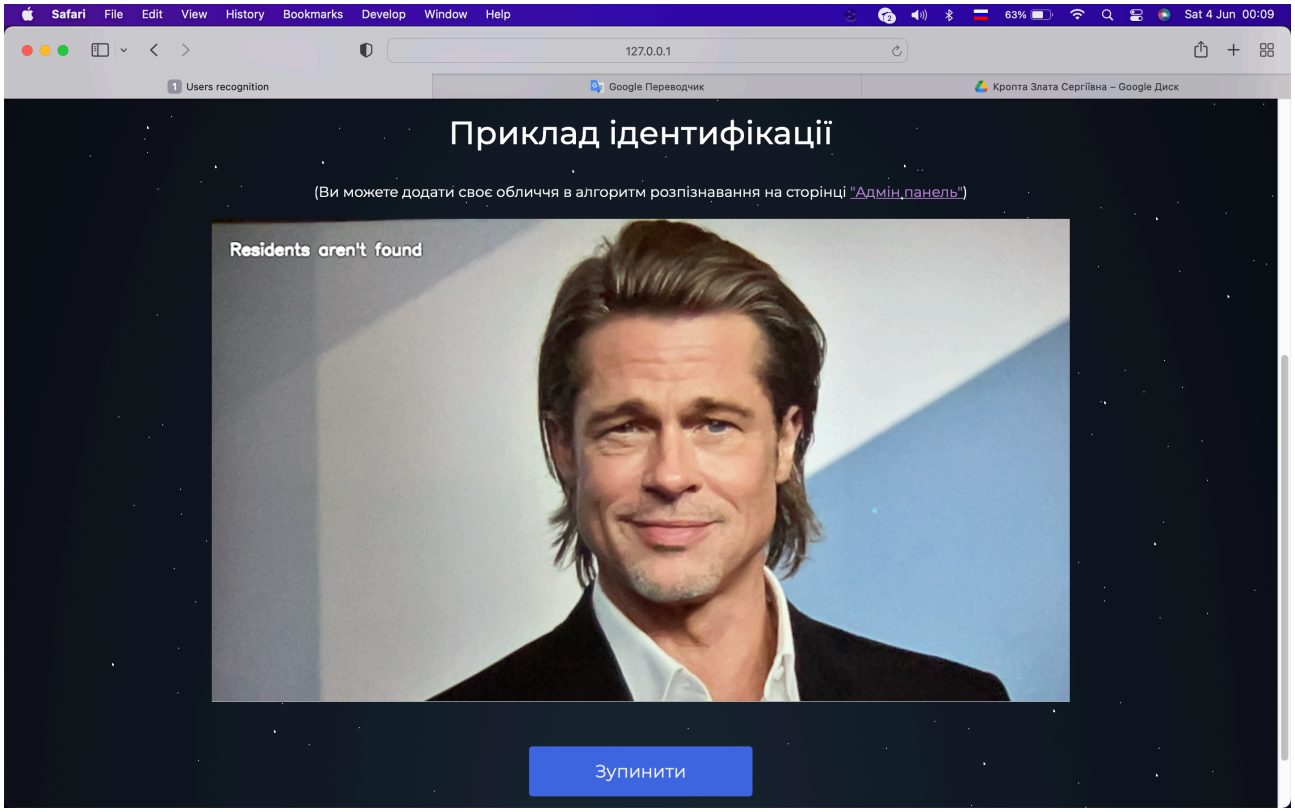

Рис 3.3 — Програма розпізнає чужинця

3. Якщо в об’єктив камери потрапило відразу декілька людей — алгоритм виводить дані про мешканців, яких він впізнав, інших же він ігнорує (див. рис. 3.4).

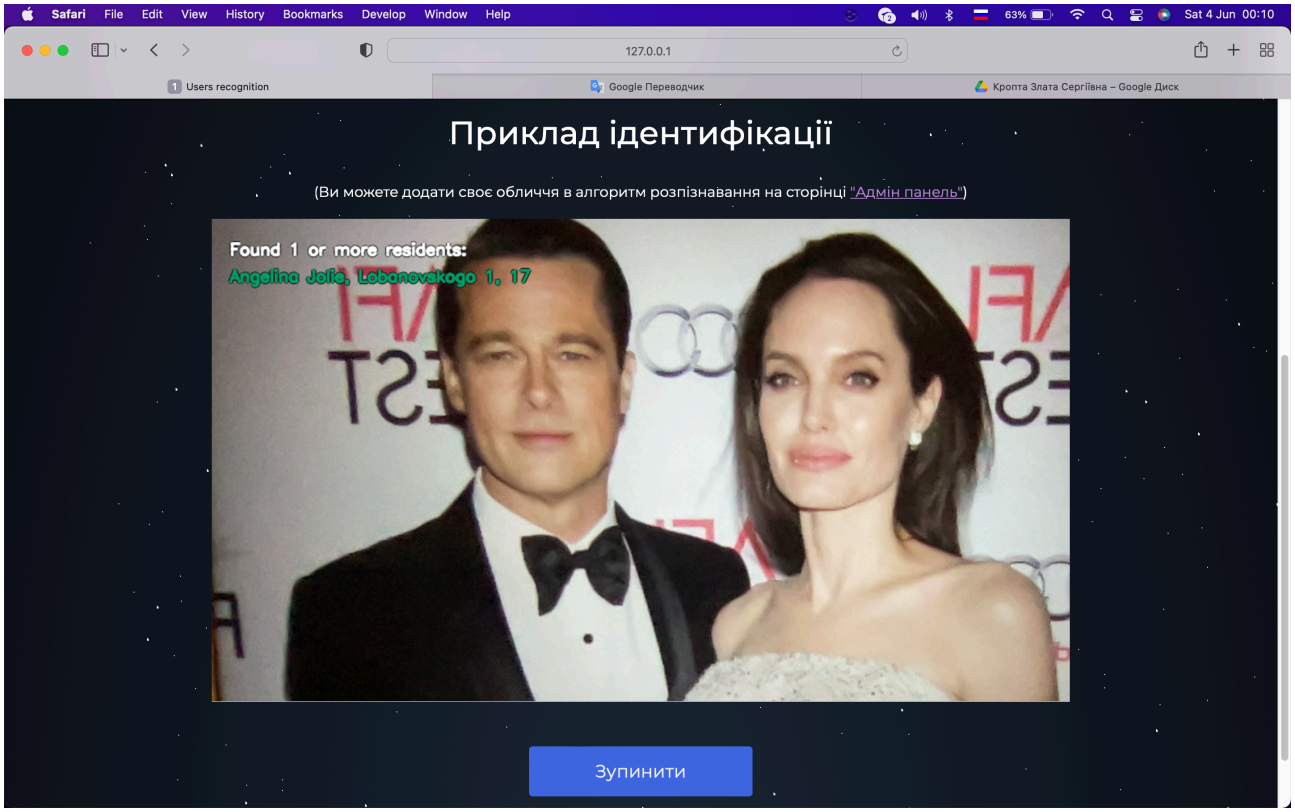

Рис 3.4 — Програма розпізнає чужинців та мешканців одночасно

Вище було описано, що відбувається при підключенні до веб-сокету зі сторони клієнта. Тепер варто розібрати, що відбувається на стороні серверу. Веб-сервер використовує бібліотеку FastAPI для створення ендпоінту веб-сокета. Також при старті сервера, ініціалізується клас ConnectionManager(), котрий відповідає за підключення до веб-сокету.

При підключенні клієнта до веб-сокета викликається інстанс класу ConnectionManager(), а саме його метод connect(), котрий приймає підключення клієнта до веб-сокету, а також додає підключення в Python масив. Далі на стороні серверу запускається нескінченний цикл, в котрому 30 разів на секунду виконується наступне:

- Якщо тип підключення до веб-сокету дорівнює одиниці, зображення з камери вилучається асинхронним методом generate_frame_bytes()

- Якщо тип підключення до веб-сокету дорівнює двійці, тоді зображення з камери вилучається асинхронним методом generate_frame_face_rec()

- Якщо тип підключення не зазначений, зображення дорівнює None об’єкту

- Якщо зображення дорівнює None об’єкту, підіймається WebSocketDisconnect помилка

- В залишившихся випадках, виконується метод broadcast() класу ConnectionManager(), що надсилає повідомлення в усі підключені клієнти до даного вебсокету.

Додатково, при виникненні WebSocketDisconnect помилки у консоль серверу виводиться відповідне повідомлення, закривається підключення кліента до веб-сокету і також виключається вебкамера.

Адмін сторінка веб-сайту

На сторінці адмін панелі є можливість додати мешканця до бази даних, натиснувши на кнопку «Додати запис». Також на цій же сторінці наявна таблиця - база даних з записами про усіх мешканців даного ЖК. В даній таблиці відображаються дані про мешканців, а саме: Ім’я, Адреса мешканця, Дата створення запису та кнопки з можливістю видалити запис та змінити його.

Адмін сторінка сайту може мати два стани:

1. Стан, коли в базі даних немає мешканців.

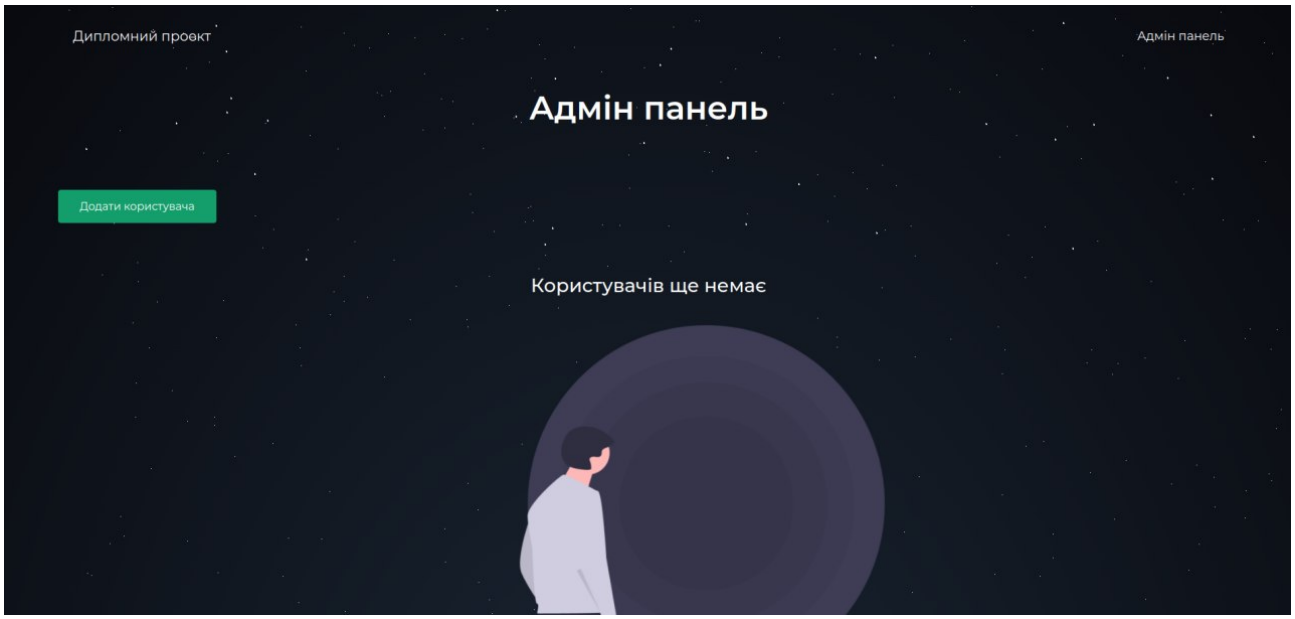

Рис 3.5 — Адмін панель, коли немає записів

2. Стан, коли таблиця база даних є заповненою.

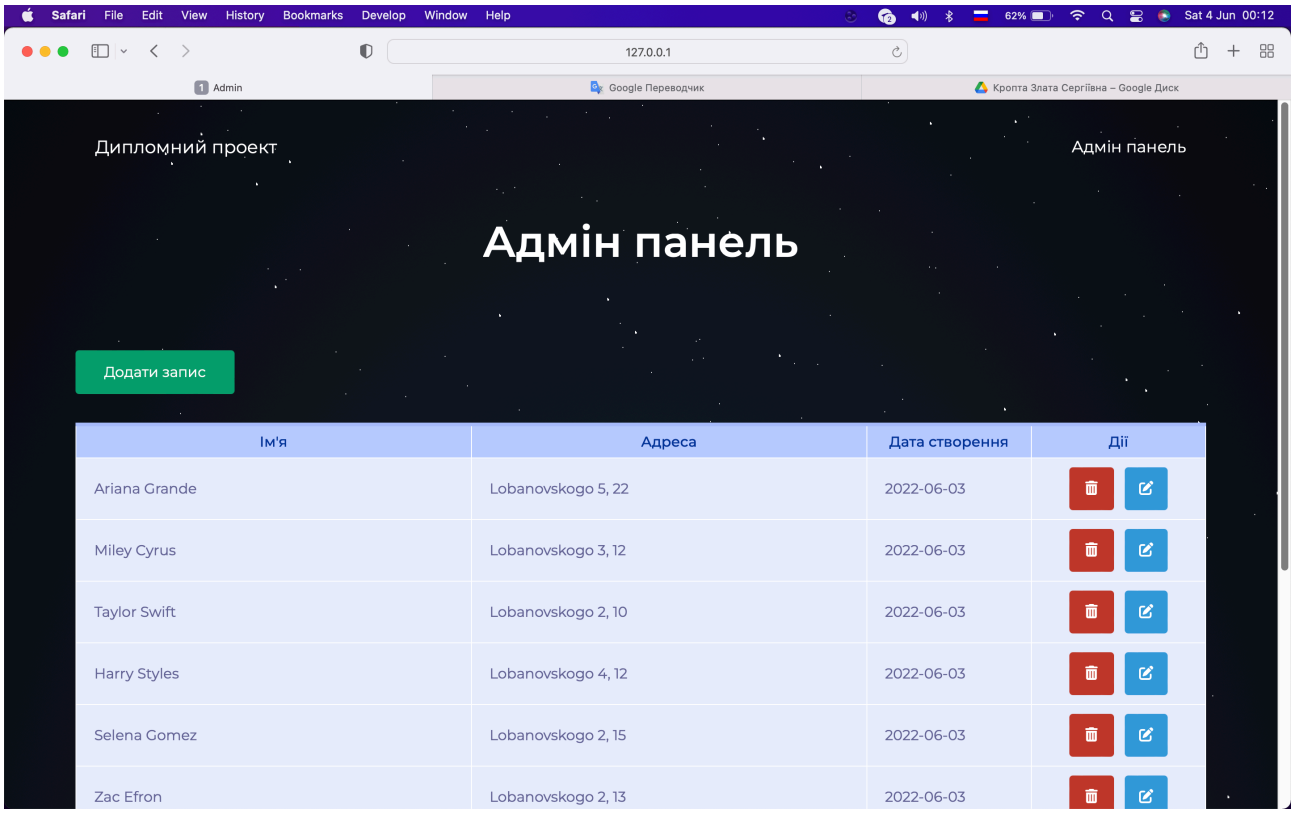

Рис 3.6 — Адмін панель, коли записи є

Таблиця на сторінці "Адмін панель" генерується за допомогою технології Jinja2 та базою даних. База даних складається з двох таблиць - User та FaceEncoding (код у додатку А). Таблиця User містить наступні колонки: айді мешканця, його ім’я, адреса та час створення запису у базі даних. Таблиця FaceEncoding зберігає в собі енкодінги облич користувачів. У ній регламентовані колонки: айді енкодінга, айді власника обличчя (що насправді є Foreign Key із посиланням на таблицю User), сам енкодінг у форматі строки та дата створення запису.

На етапі завантаження сторінки, веб-сервер робить запит у базу даних на всі записи у таблиці Users. Отримавши записи, сервер повертає їх клієнту. Технологія Jinja2 надає змогу відобразити кожний із записів, отриманих із бази даних, як окремий рядок у таблиці на веб-сторінці. У разі, якщо в базі даних немає жодного запису - на сторінці сайту буде відображен надпис "Користувачів ще немає" та тематичне зображення.

При натисканні на кнопку видалення запису із таблиці, насправді користувач переходить за відносним посиланням '/admin/users/delete/{{ user.id }}'. Ендпоінт веб-серверу обробляє це звернення, надсилаючи запит у базу даних на видалення запису із таблиці User, де співпадає id, з указаним у змінній посилання. Також видаляються усі енкодінги обличчя із таблиці FaceEncoding, що були пов’язані з вказаним мешканцем.

При натисканні на кнопку редагування запису, користувач потрапляє на сторінку, де можна змінити дані імені, адресу, а також оновити енкодінги обличчя. Така необхідність може виникнути, якщо користувач був підлітком і за кілька років дуже змінився у зовнішньому вигляді.

При натисненні на кнопку «Додати запис», даний сервіс відкриває форму з трьох кроків, які полягають у наступному (код можна подивитись у додатках В і Д):

1. Перший крок — введення даних про мешканця (див. рис. 3.7). Тут необхідно ввести ім’я та адресу людини. Якщо дані поля, що є

обов’язковими для заповнення не заповнюються — виводиться помилка та не можливо перейті до наступного кроку (див. рис. 3.8).

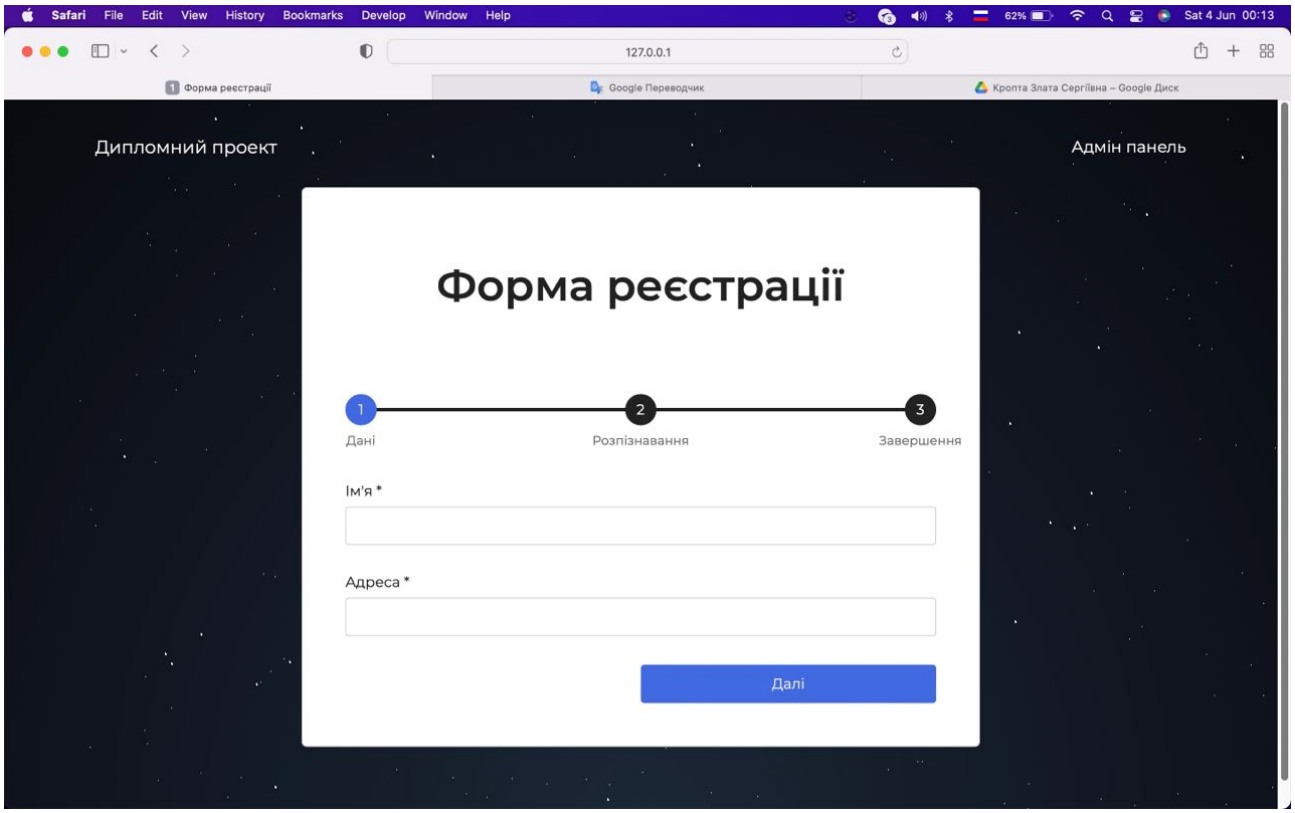

Рис 3.7 — Перший крок форми для додавання запису до БД

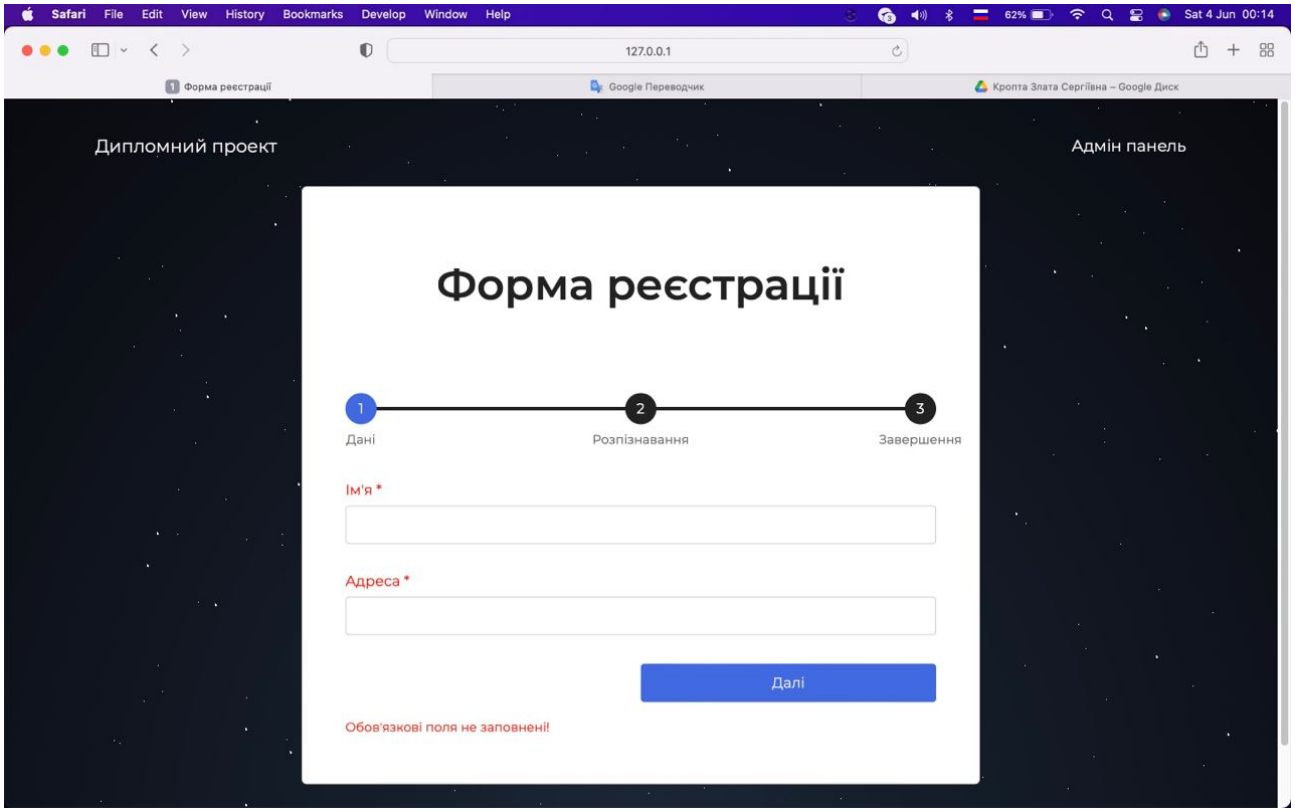

Рис 3.8 — Реакція форми при незаповненні обов’язкових полів

2. Другий крок — розпізнавання обличчя людини. До бази даних додається також код обличчя мешканця (ембеддінг). Для цього, необхідно для початку натиснути на кнопку «Почати», увімкнеться камера та розпочнеться процес розпізнавання обличчя, для цього користувачу необхідно зробити 5 фотографій згідно з инструкціями, натискаючи на кнопку «Зробити фото». Коли всі кроки інстукції виконано — зупиняється робота камери та натискаючи на кнопку «Підтвердити та відправити», до бази даних додається новий користувач (див. рис. 3.9). Коли користувач натискає кнопку "Почати", виконується JacaScript функція, яка відкриває веб-сокет за посиланням ws://127.0.0.1:8000/ws_video/1 та починає звичайну трансляцію відео з вебкамери.

Також після натискання на кнопку "Почати", з’являється кнопка "Зробити фото", а на зображенні вебкамери малюється певна рамка, у яку користувач має розмістити своє обличчя у вказаному положенні. Для коректної роботи алгоритма розпізнавання обличчя, кожен мешканець повинен зробити 5 фото із

різним положенням обличчя, а саме: прямий погляд у камеру, обличчя повернене вліво, повернене вправо, піднято вгору та опущене вниз.

У разі спроби зробити фото, на якому не буде обличчя, або їх буде декілька, фото не зарахується. Після п’ятого отриманого фото, кнопка "Зробити фото" стає неактивна.

При натисканні "Зробити фото" викликається JavaScript функція, яка робить запит за відносним посиланням '/take_photo'. Веб-сервер оброблює цей запит, отримуючи актуальне зображення з вебкамери. Використовуючи бібліотеку face_recognition, на зображенні розпізнаються обличчя. В кінці кінців, клієнту повертається ендкодінг обличчя у форматі строки і він одразу додається у JavaScript масив.

Коли користувач зробив 5 фото, він відкриває можливість відправити форму на сервер. При натисканні "Підтвердити та відправити" на стороні клієнта формуються введені дані у необхідний для відправки форми формат, після чого надсилаються як POST запрос за відносним посиланням '/admin/submit_form/'.

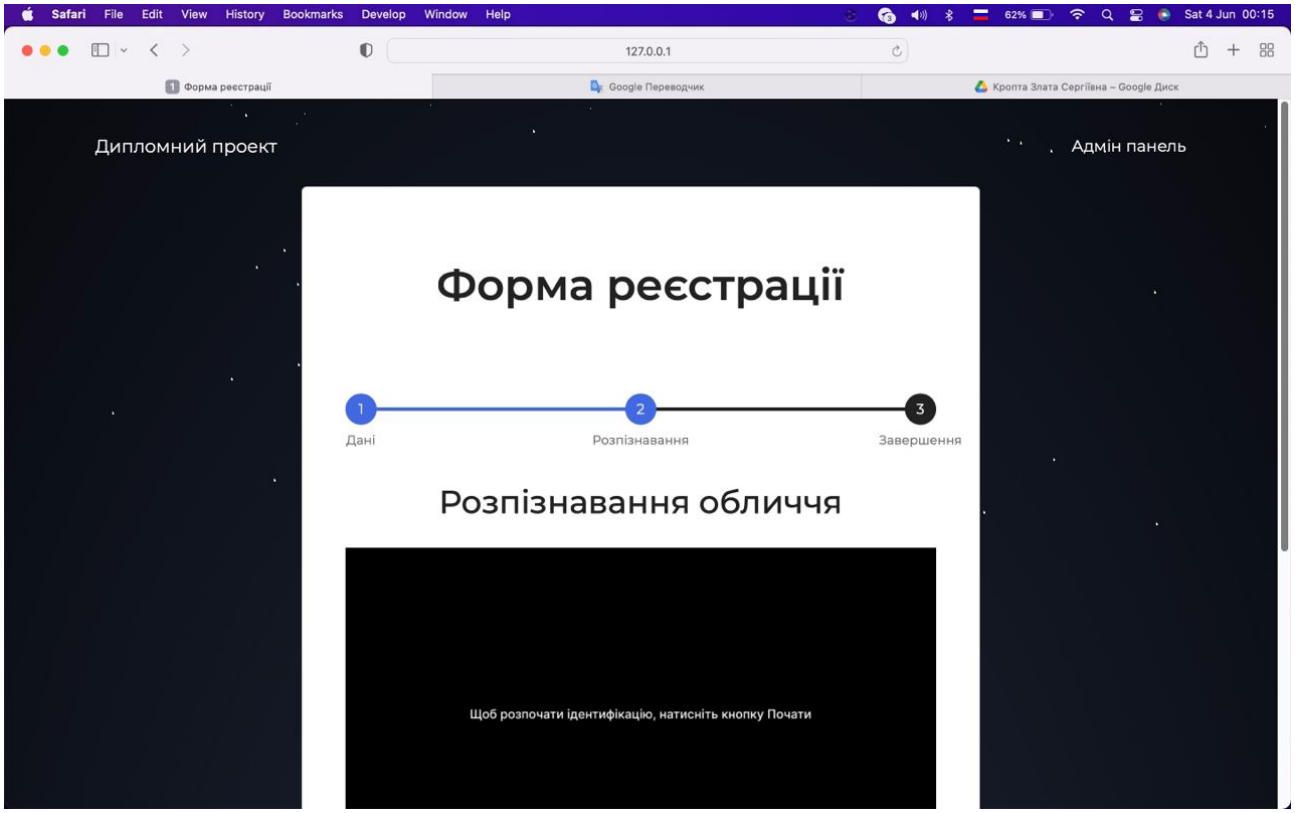

Рис 3.9 — Крок розпізнавання форми

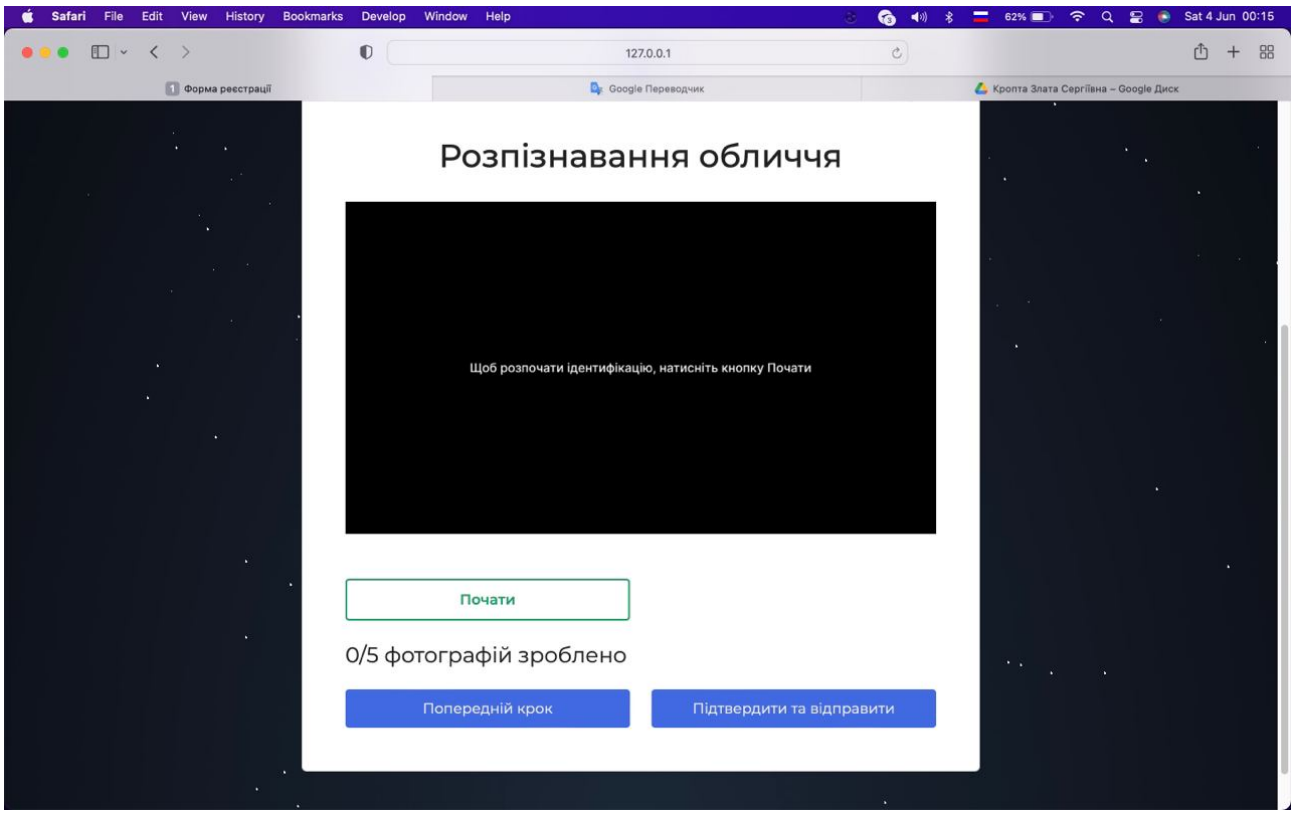

Рис 3.9 — Крок розпізнавання форми (продовження)

3. Останній крок — підтвердження додавання запису до бази даних(див. рис. 3.10.

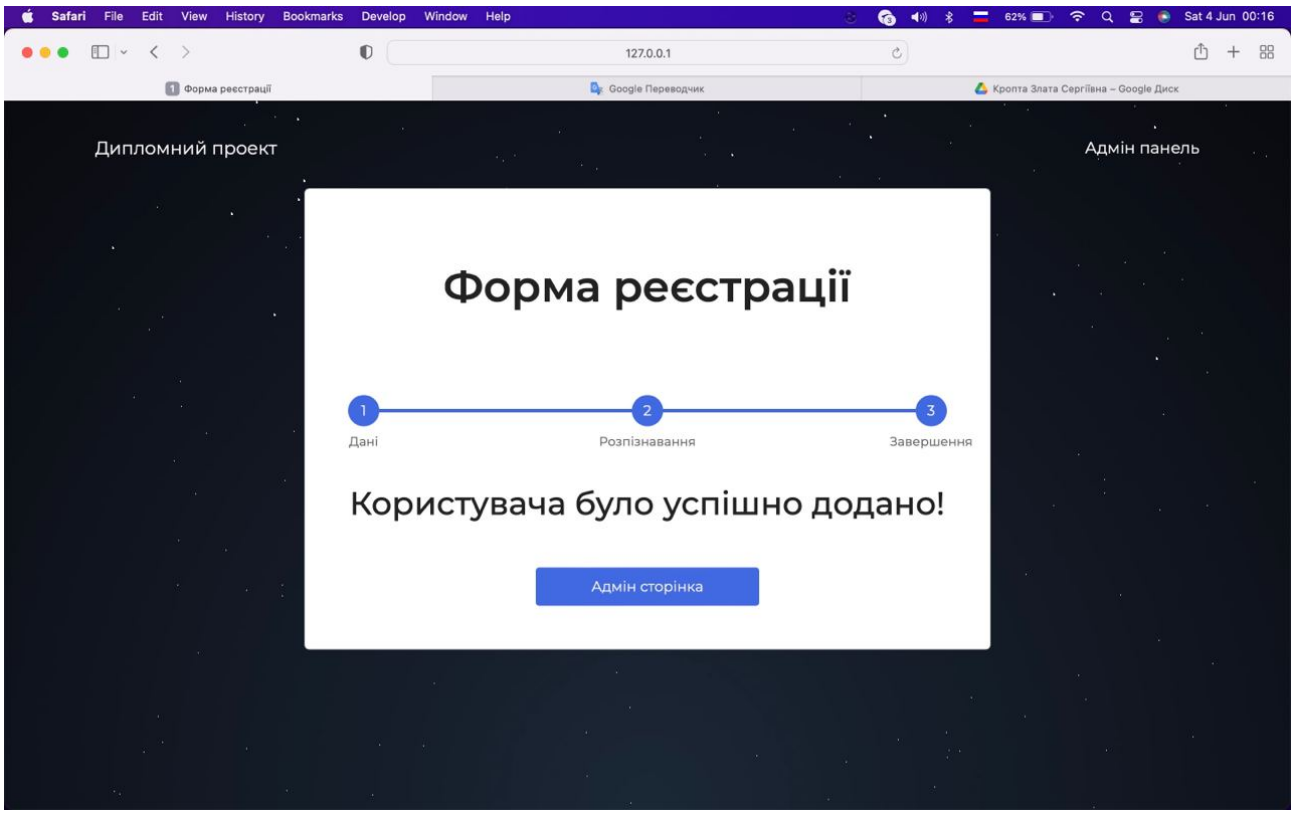

Рис 3.10 — Підтвердження успішного додавання запису

3.3 Структура бази даних системи ідентифікації мешканців ЖК

База даних складається з двох таблиць — таблиці мешканців та таблиці їх ембеддінгів. В даній роботі використовуються реляційна база даних на основі мови SQL[30].

В таблиці з мешканцями наявне їх унікальне ID, а також записано ім’я та адреса мешканця. В таблиці з ембеддінгами є унікальне ID мешканця та список з 5 ембеддінгів. Код бази даних наведено у додатку А.

Схема структури БД зображена на рисунку 3.11.

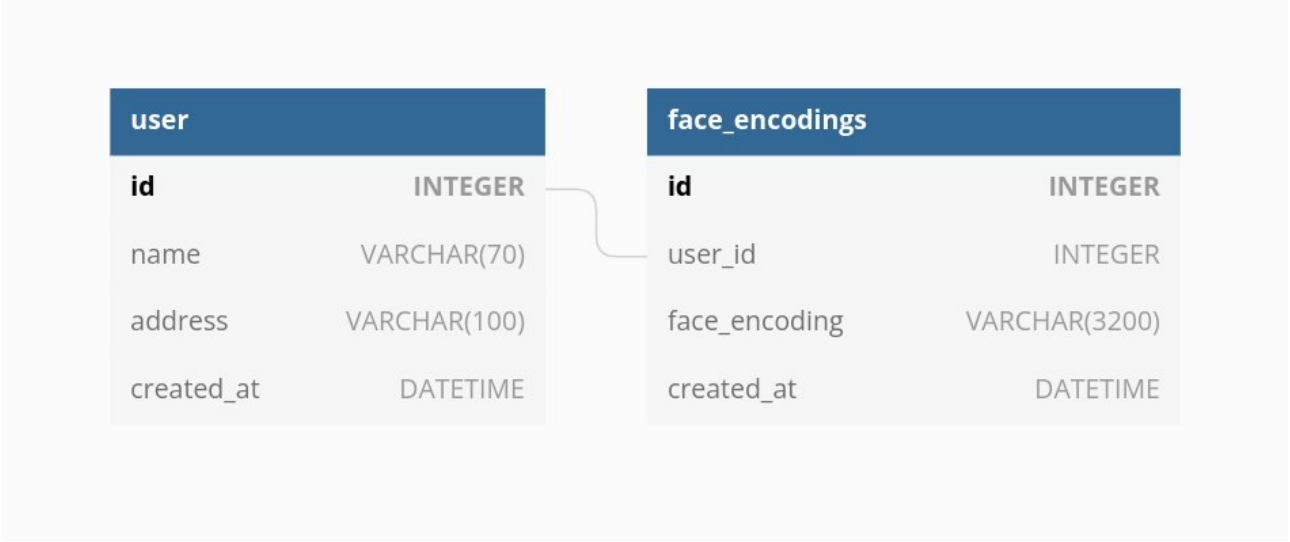

Рис 3.11 — Структура БД мешкнаців ЖК

3.4 Дослідження та налаштування алгоритму ідентифікації мешканців ЖК

Для того, щоб поліпшити точність порівняння облич, було прийнято рішення про проведення дослідження на датасеті з 10 000 людей. Даний датасет було взято з інтернету, він містить у собі 10 000 різних особистостей, з різною кількостю фотографій на кожну особу. Код дослідження наведено у додатку Ж.

Хід проведення дослідження

1. Підготовка датасету

Для початку необхідно зробити фільтрацію по всьому датасету, щоб відібрати всіх людей, які мають більше 7 фотографій. Це необхідно для подальших кроків. Далі створюється датафрейм, де кожен рядок є індексом особистості та відповідне ім’я файлу для кожної людини (див. рис. 3.12).

DataFrame has 9343 rows

	identity	filenames
0	2880	[000001.jpg, 000404.jpg, 003415.jpg, 004390.jp...
1	2937	[000002.jpg, 011437.jpg, 016335.jpg, 017121.jp...
2	8692	[000003.jpg, 015648.jpg, 033840.jpg, 038887.jp...
3	5805	[000004.jpg, 001778.jpg, 010191.jpg, 013676.jp...
4	9295	[000005.jpg, 008431.jpg, 014427.jpg, 016680.jp...

Рис 3.12 — Датафрейм з фотографіями людей

2. Створення структур даних для алгоритму розпізнавання облич:

• Датафрейм, де кожна людина має п’ять ембеддінгів обличчя. Беруться люди, які мають більше 7 фотографій, розраховуються їх ембеддінги, якщо на фотографії модель розпізнає більше 1 обличчя, або взагалі не розпізнає обличчя, то вона пропускається. Якщо ж фотографія виявилася не пригодною до розпізнавання, береться наступна фотографія даної людини, і так допоки не набереться 5 ембеддінгів. У результаті отримується датафрейм з індексом людини та списком ембеддінгів (див. рис. 3.13).

	identities	encodings
0	2880	[[-0.09392920881509781, 0.16680526733398438, 0...
1	2937	[[-0.13565079867839813, 0.04019104316830635, 0...
2	8692	[[-0.08772975206375122, 0.17790649831295013, 0...
3	5805	[[-0.207429900765419, 0.17357361316680908, -0....
4	9295	[[-0.11555222421884537, 0.05275091528892517, 0...

Рис 3.13 — Датафрейм з ембеддінгами людей

В результаті після усіх цих перетворень, отримаємо датасет з 9283 людей, які мають чіткі та не пошкоджені фотографії.

• Словник, де ключі - це індекс користувачів, а значення - відповідні імена/адреси. Будується словник, де у кожної особи є свій унікальний індекс, якому відповідають ім’я людини та її адреса.

• Датафрейм із зображеннями, які раніше не розглядалися алгоритмом. Будується тестова вибірка з тих зображень, які алгоритм ще не розглядав (оскільки для проекту потрібно лише 5 фотографій). По даній тестовій вибірці також будується датафрейм для кожної людини, який складається з унікального ідентифікатора людини та її ембеддінгів.

3. Перевірка ефективності різних підходів

Перший метод — при порівнянні ембеддінгів, обирається та людина, яка виконала умову порогу найпершою з усього списку ембеддінгів.

Цей метод є найпростішим. Його було протестовано на різних порогах і отримано такі результати (див. рис. 3.14):

```
Accuracy with 0.525 threshold: 0.72
Accuracy with 0.550 threshold: 0.77
Accuracy with 0.575 threshold: 0.76
```

Рис 3.14 — Точність роботи першого методу

Даний метод працює з точністю 77%, при порозі 0.55, даний показник необхідно покращити, отже переходимо до другого методу.

Другий метод — рахується середнє значення ембеддінгів.

Даний метод бере 5 ембеддінгів облич певної людини та знаходить їх середнє значення. Таким чином, отримується список, де кожній людини належить одне значення ембеддінгу. При порівнянні людей з цим методом також перебирається найкращий поріг (див. рис. 3.15):

```
Accuracy with 0.400 threshold: 0.65
Accuracy with 0.410 threshold: 0.69
Accuracy with 0.420 threshold: 0.72
Accuracy with 0.430 threshold: 0.74
Accuracy with 0.440 threshold: 0.74
Accuracy with 0.450 threshold: 0.73
Accuracy with 0.460 threshold: 0.71
Accuracy with 0.470 threshold: 0.65
```

Рис 3.15 — Точність роботи другого методу

Метод середнього значення спрацював гірше за перший метод, найкраща точність ідентифікації скаладає 74%, що на 3% менше за перший метод. Отже, даний метод слід відкинути та рухатися далі.

Третій метод — обирається та людина, у якої найбільша кількість ембеддінгів (а саме, 5) співпала, тобто вивелось значення True. Для підвищення точності було прийнято рішення вдосконалити перший метод, беручи не лише перше значення True зі списку ембеддінгів, але проходячи усі 5 значень і обираючи саме ту людину, у якої співпало найбільше ембеддінгів (див. рис. 3.16).

```
Accuracy with 0.480 threshold: 0.77
Accuracy with 0.490 threshold: 0.76
Accuracy with 0.500 threshold: 0.76
Accuracy with 0.510 threshold: 0.76
Accuracy with 0.520 threshold: 0.75
Accuracy with 0.530 threshold: 0.76
Accuracy with 0.540 threshold: 0.76
Accuracy with 0.550 threshold: 0.72
Accuracy with 0.560 threshold: 0.70
```

Рис 3.16 — Точність роботи третього методу

Четвертий метод — рахує евклідову відстань ембеддінгів та бере їх середнє значення. В результаті отримується список з середніми значеннями відстані. Далі обираємо перше значення, яке буде відповідати заданому порогу. Результати розрахунків точності наведені на рисунку 3.17.

```
Accuracy with 0.380 threshold: 0.52
Accuracy with 0.390 threshold: 0.57
Accuracy with 0.400 threshold: 0.61
Accuracy with 0.410 threshold: 0.62
Accuracy with 0.420 threshold: 0.62
Accuracy with 0.430 threshold: 0.65
Accuracy with 0.440 threshold: 0.63
Accuracy with 0.450 threshold: 0.57
```

Рис 3.17 — Точність роботи четвертого методу

Останній метод — порівняння ембеддінгів за допомогою знаходження мінімальної евклідової відстані. Суть даного методу - це знайти відстань(різницю) між даними п'ятьма ембеддінгами людини, що ідентифікується та даними ембеддінгів з загального списку наявних людей. Таким чином, отримаємо список різниць між ембеддінгами. В результаті з наявних даних обирається мінімальна різниця. Результат роботи методу наведено нижче (рис. 3.18):

```
print(f'{np.mean(ident_accs_min_dist):.2f}')
```

```
Accuracy = 0.9129607291247822
```

Рис 3.18 — Точність роботи п’ятого методу

Найкращу точність показав останній метод порівняння. Він обчислював евклідові відстані до кожного кодування, отримував середню відстань для кожної особи з набору даних і повертав особу з найменшим значенням відстані. Остаточна точність склала 91%. Точність може становити більше, якщо набір

даних точніший. Для більшої точності необхідно, щоб фотографії в базі даних мали гарне освітлення, ракурс і якість, тоді відсоток точності може підвищитись.

Отже, у даному розділі було детально описано алгоритм роботи програмии для ідентифікації мешканців ЖК, була описана структура БД, також було більш детально розглянуто дослідження для поліпшення точності розпізнавання та ідентифікації мешканців, виведено фінальний результат.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What Is Facial Recognition, How It Is Used & What Is It's Future Scope? [Електронний ресурс]. – Режим доступу: https://www.datatobiz.com/blog/facial-recognition-understanding/

2. Технология распознавания лиц [Електронний ресурс]. – Режим доступу: https://www.onespan.com/ru/topics/raspoznavanie-lic

3. Использование функции Face ID на iPhone и iPad Pro [Електронний ресурс]. – Режим доступу: https://support.apple.com/ru-ru/HT208109

4. Что такое Apple Face ID и камера True Depth [Електронний ресурс]. – Режим доступу: https://www.kaspersky.ru/blog/apple-face-id-security/18732/

5. Что такое Face ID в смартфоне и принцип работы [Електронний ресурс]. – Режим доступу: https://mob-mobile.ru/statya/9292-chto-takoe-face-id-v-smartfone-i-princip-raboty.html

6. FACE RECOGNITION: A MODERN WAY OF SECURITY [Електронний ресурс]. – Режим доступу: https://www.innefu.com/blog/face-recognition-security/\

7. Facial Recognition - Interpol [Електронний ресурс]. – Режим доступу: https://www.interpol.int/How-we-work/Forensics/Facial-Recognition

8. HomeBurglary Statistics 2021 [Електронний ресурс]. – Режим доступу: https://reolink.com/home- burglary-crime-statistics/

9. Кто такие стейкхолдеры и как ими управлять [Електронний ресурс]. – Режим доступу: https://blog.calltouch.ru/stejkholdery-kto-eto-takie-kakie-byvayut-vidy-stejkholderov-proekta/

10. Кто такие стейкхолдеры и как ими продуктивно управлять [Електронний ресурс]. – Режим доступу: https://anvilhook.ru/blog/kto-takie-stejkxoldery-i-kak-imi-produktivno-upravlyat

11. Анализ существующих подходов к распознаванию лиц [Електронний ресурс]. – Режим доступу: https://habr.com/ru/company/synesis/blog/238129/

12. Архитектура клиент-сервер [Електронний ресурс]. – Режим доступу: https://bugza.info/arxitektura-klient-server/

13. Face detection[Електронний ресурс]. – Режим доступу: https://www.techtarget.com/searchenterpriseai/definition/face-detection

14. Using Histogram of Oriented Gradients (HOG) for Object Detection[Електронний ресурс]. – Режим доступу: https://iq. genus.org/object-detection-with-histogram-of-oriented-gradients-hog/

15.A STUDY OF FACE EMBEDDING IN FACE RECOGNITION [Електронний ресурс]. – Режим доступу: https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=3377&context=theses

16. What is a Relational Database? [Електронний ресурс]. – Режим доступу: https://fauna.com/blog/relational-database

17. SQL vs NoSQL: 5 Critical Differences [Електронний ресурс]. – Режим доступ

18. FastAPI [Електронний ресурс]. – Режим доступу: https://fastapi.tiangolo.com

19. Uvicorn [Електронний ресурс]. – Режим доступу: https://www.uvicorn.org

20. Starlette [Електронний ресурс]. – Режим доступу: https://www.starlette.io

21. WebSockets [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/ru/docs/Web/API/WebSockets_API

22. OpenCV [Електронний ресурс]. – Режим доступу: https://opencv.org/about/

23. DLIB [Електронний ресурс]. – Режим доступу: http://dlib.net

24. SQLAlchemy [Електронний ресурс]. – Режим доступу: https://github.com/sqlalchemy/sqlalchemy

25. Jinja2 [Електронний ресурс]. – Режим доступу: https://pypi.org/project/Jinja2/

26. Pydantic [Електронний ресурс]. – Режим доступу: https://medium.com/swlh/cool-things-you-can-do-with-pydantic-fc1c948fbde0

27. Jan Erik Solem. Programming Computer Vision with Python. – Видавництво O'Reilly Media, Inc., 2012. – 300 с. – ISBN: 978-144-931-654-9

28. Joseph Howes, Joe Minichino. Learning OpenCV 4 Computer Vision with Python 3. – Видавництво Packt Publishing, 2020. – 372 с. – ISBN: 978-178-953-161-9

29. Asit Kumar Datta, Madhura Datta, Pradipta Kumar Banerjee. Computer Vision: Face Detection and Recognition. – Видавництво Chapman and Hall/CRC, 2015. – 321 с. – ISBN: 978-148-222-657-7

30. Josephine Bush. Learn SQL Database Programming. – Видавництво Packt Publishing, 2020. – 564 с. – ISBN: 978-183-898-476-2

ВИСНОВКИ

У даній роботі було зроблено дослідження задля підбору потрібного метода для найкращої точності розпізнавання та ідентифікації облич. Була проведена робота з базами даних, у результаті проект пов’язаний з реляційною БД мешканців, що складається з двох таблиць. Також було програму для розпізнавання та ідентифікації людей, розроблено фронт-енд та бек-енд частини.

У результаті даної випускної кваліфікаційної роботи була отримана система ідентифікації мешканців ЖК, що складається з серверу, який розпізнає та ідентифікує людину у реальному часі, звертаючись до бази даних мешканців даного ЖК.

ДОДАТОК А

database.py

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

SQLALCHEMY_DATABASE_URL = "sqlite:///./sql/sqlite.db"
engine = create_engine(SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False})
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()
```

models.py

```
from datetime import datetime
from sqlalchemy import Integer, String, Column, DateTime, ForeignKey
from sqlalchemy.orm import relationship

from .database import Base

class User(Base):
    __tablename__ = 'user'

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(70))
    address = Column(String(100))
    created_at = Column(DateTime, default=datetime.utcnow())

    face_encodings = relationship('FaceEncoding')
```

```
class FaceEncoding(Base):
    __tablename__ = 'face_encodings'

    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey('user.id'))
    face_encoding = Column(String(3200))
    created_at = Column(DateTime, default=datetime.utcnow())

    owner = relationship("User", back_populates="face_encodings")
```

schemas.py

```
from datetime import datetime
from pydantic import BaseModel
from typing import List, Optional


class FaceEncodingBase(BaseModel):
    face_encoding: str


class FaceEncodingCreate(FaceEncodingBase):
    pass


class FaceEncoding(FaceEncodingBase):
    id: int
    user_id: int
    created_at: Optional[datetime]
```

```
    class Config:
        orm_mode = True


class UserBase(BaseModel):
    name: str
    address: str


class UserCreate(UserBase):
    pass


class User(UserBase):
    id: int
    face_encodings: List[FaceEncoding] = []
    created_at: Optional[datetime]

    class Config:
        orm_mode = True
```

crud.py

```
from sqlalchemy.orm import Session
from . import models, schemas
from .database import SessionLocal

# Dependency
def get_db():
    db = SessionLocal()
```

```
    try:
        yield db
    finally:
        db.close()

def get_user(db: Session, user_id: int):
    return db.query(models.User).filter(models.User.id == user_id).first()

def get_user_by_name(db: Session, name: str):
    return db.query(models.User).filter(models.User.name == name).first()

def get_user_by_address(db: Session, address: str):
    return db.query(models.User).filter(models.User.address == address).first()

def get_user_by_name_and_address(db: Session, name: str, address: str):
        return db.query(models.User).filter(models.User.address == address, models.User.name == name).first()

def get_users(db: Session, skip: int = 0, limit: int = 100):
    return db.query(models.User).offset(skip).limit(limit).all()

def create_user(db: Session, user: schemas.UserCreate):
    db_user = models.User(name=user.name, address=user.address)
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user

def delete_user(db: Session, user_id: int):
    user = db.query(models.User).filter(models.User.id == user_id).first()
    db.delete(user)
```

```
        db.commit()
        return True

    def create_user_face_encoding(db: Session, face_encoding:
schemas.FaceEncodingCreate, user_id: int):
            db_face_encoding = models.FaceEncoding(**face_encoding.dict(),
user_id=user_id)
        db.add(db_face_encoding)
        db.commit()
        db.refresh(db_face_encoding)
        return db_face_encoding

    def get_face_encodings_by_user_id(db: Session, user_id: int):
                                        face_encodings =
db.query(models.FaceEncoding).filter(models.FaceEncoding.user_id ==
user_id).all()
        return face_encodings

    def delete_face_encoding_by_user_id(db: Session, user_id: int):
                                        face_encodings =
db.query(models.FaceEncoding).filter(models.FaceEncoding.user_id ==
user_id).all()
            print(f'sql.crud.delete_face_encoding_by_user_id(): Deleting
{len(face_encodings)} encodings of User with id {user_id}')
        for face_encoding in face_encodings:
          db.delete(face_encoding)
          db.commit()
        return True

    def get_face_encodings(db: Session, skip: int = 0, limit: int = 100):
        return db.query(models.FaceEncoding).offset(skip).limit(limit).all()
```

ДОДАТОК Б

service_get_webcam_stream.py

```
import time
import cv2
import numpy as np
import face_recognition
import json

from sql import models, crud
from sql.database import SessionLocal
from sqlalchemy.orm import Session



# Dependency
def get_db():
    db_ = SessionLocal()
    try:
        yield db_
    finally:
        db_.close()



class WebcamStream:

    def __init__(self):
        self.webcam = None
        self.TOLERANCE = 0.4
        self.FONT_SCALE = 0.6
        self.MODEL = 'hog'
```

```
            self.COLOR = [0, 255, 0]
            self.db_data = self.reload_database()
            self.time_from_last_face_rec_update = time.time()
            self.last_face_rec_results = None
              print(f'service_get_webcam_stream.generate_frame_face_rec(): Quantity of encodings in db_data -> {len(self.db_data["encodings"])}')
              print(f'service_get_webcam_stream.generate_frame_face_rec(): Quantity of encodings in the DB -> {len(crud.get_face_encodings(db=SessionLocal()))}')

        def start_stream(self) -> bool:
            """
            Starts the webcam stream
            """
            self.webcam = cv2.VideoCapture(0)
            self.webcam.set(cv2.CAP_PROP_FRAME_WIDTH, 960)
            self.webcam.set(cv2.CAP_PROP_FRAME_HEIGHT, 540)
            self.webcam.set(cv2.CAP_PROP_FPS, 30)
                print('service_get_webcam_stream.start_stream(): Webcam has turned on!')
            return True

        def stop_stream(self) -> bool:
            """
            Stops the webcam stream
            """
            self.webcam.release()
                print('service_get_webcam_stream.stop_stream(): Webcam has turned off!')
            return True

        def reload_database(self, db: Session = SessionLocal()) -> dict:
```

```
        """
        Reloads the database with faces that needed to be recognized and owners' data
        """
        all_encodings = db.query(models.FaceEncoding.face_encoding).all()
        all_names = db.query(models.User.name).all()
        all_addresses = db.query(models.User.address).all()
        fin_encodings = []
        fin_names = []
        fin_addresses = []
        for encoding in all_encodings:
            fin_encodings.append(json.loads(encoding[0]))
        for name in all_names:
            for i in range(5):
                fin_names.append(name[0])
        for address in all_addresses:
            for i in range(5):
                fin_addresses.append(address[0])
        db.close()
        print('service_get_webcam_stream.reload_database(): The data from the DB was updated!')
                print(f'service_get_webcam_stream.reload_database(): {len(fin_encodings)}')
        return {'encodings': fin_encodings, 'names': fin_names, 'addresses': fin_addresses}

    def put_text_(self, frame, text, org, color=(108, 158, 20)):
                cv2.putText(img=frame, text=text, org=org, fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=self.FONT_SCALE,
                    color=(0, 0, 0), lineType=cv2.LINE_AA, thickness=3)
```

```
                cv2.putText(img=frame, text=text, org=org,
fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=self.FONT_SCALE,
                    color=color, lineType=cv2.LINE_AA, thickness=2)

        async def generate_frame_bytes(self) -> bytes or None:
            """
            Returns the actual webcam frame encoded as png and then as bytes

            Is used in resource_webcam_stream.websocket_endpoint() if stream_type
== 1
            """
            try:
                success, frame = self.webcam.read()
                frame = cv2.flip(frame, 1)
                ret, png_frame = cv2.imencode('.png', frame)
                bytes_frame = png_frame.tobytes()
                return bytes_frame
            except:
                return None

            async def generate_frame_face_rec(self, db: Session = SessionLocal()) ->
bytes or None:
            """
            Returns the actual webcam frame encoded as png and then as bytes
            It also has names and addresses of recognized captured people

            Performance measurements on i5-8300H:
            - raw frame generation takes ~0.035 sec
            - frame with located face in it takes ~0.43 sec
            - frame with located and recognized from DB face takes ~0.6 sec
```

```
        Performance measurements on M1:
        - raw frame generation takes ~0.07 sec
        - frame with located face in it takes ~0.32 sec
        - frame with located and recognized from DB face takes ~0.33 sec

        Is used in resource_webcam_stream.websocket_endpoint() if stream_type
== 2
        """
        try:
            t1 = time.time()
            # trying to retrieve a frame from the webcam
            success, frame = self.webcam.read()
            frame = cv2.flip(frame, 1)

              # recognize faces if it was more than 1 seconds from the last face rec
update
             # print(f'service_get_webcam_stream.generate_frame_face_rec(): Time
from the last face rec update -> '
            #       f'{time.time() - self.time_from_last_face_rec_update}')
            if time.time() - self.time_from_last_face_rec_update > 1:
                                locations = face_recognition.face_locations(frame,
model=self.MODEL)
                encodings = face_recognition.face_encodings(frame, locations)
                self.last_face_rec_results = ['Residents aren\'t found', ]

                for face_encoding, face_location in zip(encodings, locations):
                                                                results =
face_recognition.compare_faces(self.db_data['encodings'], face_encoding,
self.TOLERANCE)
                    if True in results:
                        if self.last_face_rec_results[0] == 'Residents aren\'t found':
```

```
                        self.last_face_rec_results = ['Found 1 or more residents:', ]
                    name = self.db_data['names'][results.index(True)]
                                                                    address =
db.query(models.User.address).filter(models.User.name == name).first()
                     print(f'service_get_webcam_stream.generate_frame_face_rec():
User found: {name}, {address[0]}')
                    self.last_face_rec_results.append((name, address[0]))
                self.time_from_last_face_rec_update = time.time()

            padding = 0
            for person in self.last_face_rec_results:
                if isinstance(person, str):
                    text = person
                        self.put_text_(frame, text, (20, 40 + padding), color=(255, 255,
255))
                else:
                    text = f'{person[0]}, {person[1]}'
                    self.put_text_(frame, text, (20, 40 + padding))
                padding += 30

            ret, png_frame = cv2.imencode('.png', frame)
            bytes_frame = png_frame.tobytes()
            t2 = time.time()
                 # print(f'service_get_webcam_stream.generate_frame_face_rec():
Frame generation took {t2 - t1} sec')
            return bytes_frame
        except:
            return None

    def generated_frame(self) -> np.ndarray or None:
        """
```

```
Returns the actual webcam frame

Is used in resource_webcam_stream.take_photo()
"""
try:
    success, frame = self.webcam.read()
    frame = cv2.flip(frame, 1)
    return frame
except:
    return None
```

ДОДАТОК В

resource_add_user.py

```
from fastapi import APIRouter, Form, Depends, HTTPException
from fastapi.responses import HTMLResponse, PlainTextResponse
from fastapi.requests import Request
from fastapi.templating import Jinja2Templates
import json

from sqlalchemy.orm import Session
from sql import crud, schemas
from resources.resource_webcam_stream import update_face_rec_db

router_add_user = APIRouter()
templates = Jinja2Templates(directory="templates")


@router_add_user.get('/add_user/', response_class=HTMLResponse)
def get_add_user_page(request: Request):
    return templates.TemplateResponse('add_user.html', {'request': request})


@router_add_user.post('/add_user/')
def submit_form(name: str = Form(...), address: str = Form(...), encodings: str = Form(...),
                db: Session = Depends(crud.get_db)):
    # Adding new User
    user = schemas.UserCreate(name=name, address=address)
    db_user_name = crud.get_user_by_name(db=db, name=user.name)
    db_user_address = crud.get_user_by_address(db=db, address=user.address)
```

```
        if db_user_address and db_user_name:
                raise HTTPException(status_code=400, detail='Name and address is already registered')
        crud.create_user(db=db, user=user)

        # Adding Face Encodings
                user_id = crud.get_user_by_name_and_address(db=db, name=name, address=address)
        user_encodings = []
        for encoding in encodings.split('],['):
            if encoding[0] != '[':
                encoding = '[' + encoding
            if encoding[-1] != ']':
                encoding = encoding + ']'
            face_encoding = schemas.FaceEncodingCreate(face_encoding=encoding)
                crud.create_user_face_encoding(db=db, face_encoding=face_encoding, user_id=user_id.id)
            user_encodings.append(json.loads(encoding))

        # Updating Face Rec Algorithm's Data Base
        update_face_rec_db(user_encodings, name, address)

        return PlainTextResponse('User and encodings were succesfully added!')


    from fastapi import APIRouter, Form, Depends, HTTPException
    from fastapi.responses import HTMLResponse, PlainTextResponse
    from fastapi.requests import Request
    from fastapi.templating import Jinja2Templates
    import json

    from sqlalchemy.orm import Session
```

```
from sql import crud, schemas
from resources.resource_webcam_stream import update_face_rec_db

router_add_user = APIRouter()
templates = Jinja2Templates(directory="templates")


@router_add_user.get('/add_user/', response_class=HTMLResponse)
def get_add_user_page(request: Request):
    return templates.TemplateResponse('add_user.html', {'request': request})


@router_add_user.post('/add_user/')
def submit_form(name: str = Form(...), address: str = Form(...), encodings: str = Form(...),
                db: Session = Depends(crud.get_db)):
    # Adding new User
    user = schemas.UserCreate(name=name, address=address)
    db_user_name = crud.get_user_by_name(db=db, name=user.name)
    db_user_address = crud.get_user_by_address(db=db, address=user.address)
    if db_user_address and db_user_name:
        raise HTTPException(status_code=400, detail='Name and address is already registered')
    crud.create_user(db=db, user=user)

    # Adding Face Encodings
    user_id = crud.get_user_by_name_and_address(db=db, name=name, address=address)
    user_encodings = []
    for encoding in encodings.split('],['):
        if encoding[0] != '[':
```

```
                encoding = '[' + encoding
            if encoding[-1] != ']':
                encoding = encoding + ']'
            face_encoding = schemas.FaceEncodingCreate(face_encoding=encoding)
                crud.create_user_face_encoding(db=db, face_encoding=face_encoding,
user_id=user_id.id)
            user_encodings.append(json.loads(encoding))

        # Updating Face Rec Algorithm's Data Base
        update_face_rec_db(user_encodings, name, address)

        return PlainTextResponse('User and encodings were succesfully added!')
```

ДОДАТОК Г

resource_webcam_stream.py

```
import asyncio
import time
from typing import List
from fastapi import APIRouter, WebSocket
from fastapi.responses import PlainTextResponse, HTMLResponse
from starlette.websockets import WebSocketDisconnect

from services.service_get_webcam_stream import WebcamStream
from services.service_recognize_faces import RecognizeFaces

router_webcam_stream = APIRouter()

webcam_stream = WebcamStream()
recognize_faces = RecognizeFaces()



def update_face_rec_db(encodings, name, address) -> bool:
    for encoding in encodings:
        webcam_stream.db_data['encodings'].append(encoding)
    for i in range(5):
        webcam_stream.db_data['names'].append(name)
    for i in range(5):
        webcam_stream.db_data['addresses'].append(address)
    return True



def reload_face_rec_db() -> bool:
    t1 = time.time()
```

```
        webcam_stream.db_data = webcam_stream.reload_database()
                print(f'resource_webcam_stream.reload_face_rec_db(): db_data was reloaded successfully! It took {time.time() - t1:.3f} sec')
        return True


    class ConnectionManager:
        def __init__(self):
            self.active_connections: List[WebSocket] = []

        async def connect(self, websocket: WebSocket):
            await websocket.accept()
            self.active_connections.append(websocket)

        def disconnect(self, websocket: WebSocket):
            self.active_connections.remove(websocket)

        async def broadcast(self, frame: bytes):
            for connection in self.active_connections:
                await connection.send_bytes(frame)


    manager = ConnectionManager()


    # WebSocket
    @router_webcam_stream.websocket("/ws_video/{stream_type}")
    async def websocket_endpoint(websocket: WebSocket, stream_type: int):
        await manager.connect(websocket)
        try:
            while True:
```

```
                # Set displaying FPS here as: 1 / FPS
                await asyncio.sleep(0.03)

                if stream_type == 1:
                    frame = await webcam_stream.generate_frame_bytes()
                elif stream_type == 2:
                    frame = await webcam_stream.generate_frame_face_rec()
                else:
                    frame = None

                if frame is None:
                    raise WebSocketDisconnect
                else:
                    await manager.broadcast(frame)
            except WebSocketDisconnect:
                        print('resource_webcam_stream.websocket_endpoint():  Raised
WebSocketDisconnect')
                manager.disconnect(websocket)
                webcam_stream.stop_stream()


        @router_webcam_stream.get('/video_start')
        def video_start():
            webcam_stream.start_stream()
            return PlainTextResponse('You just turned on the camera')


        @router_webcam_stream.get('/video_stop')
        def video_stop():
            webcam_stream.stop_stream()
            return PlainTextResponse('You just turned off the camera')
```

```
@router_webcam_stream.get('/take_photo')
def take_photo() -> str or bool:
    photo = webcam_stream.generated_frame()
    if photo is None:
        print('resource_webcam_stream.recognize_face(): Couldn\'t take a photo, maybe webcam is turned off')
        return PlainTextResponse('')

    encodings = recognize_faces.recognize_faces(photo)

    if len(encodings) != 1:
        print('resource_webcam_stream.recognize_face(): Face not found or found more than 1 face!')
        return PlainTextResponse('')
    else:
        return str(list(encodings[0]))
```

ДОДАТОК Д

main.js

```
////////////////////////////// WEBCAM WEBSOCKET //////////////////////////////


let webcam_place = document.getElementById("webcam");
let msg = document.getElementById("video");
let context = msg.getContext("2d");

function openSocket() {
    // number 1 in the end of the link below means that the websocket will be returning pure webcam stream
    var websocket = new WebSocket("ws://127.0.0.1:8000/ws_video/1");
    websocket.onmessage = (event) => {
        let image = new Image(msg.width, msg.height);
        const urlObject = URL.createObjectURL(event.data);
        image.src = urlObject;
        image.onload = (event) => {
            context.drawImage(image, 0, 0);
            URL.revokeObjectURL(urlObject);
            delete event
        };
        delete image
        delete event
    }
}

function openSocketFaceRecognition() {
    // number 2 in the end of the link below means that the websocket will be returning webcam stream with face recoognition
```

```
            var websocket = new WebSocket("ws://127.0.0.1:8000/ws_video/2");
            websocket.onmessage = (event) => {
                let image = new Image(msg.width, msg.height);
                const urlObject = URL.createObjectURL(event.data);
                image.src = urlObject;
                image.onload = (event) => {
                    context.drawImage(image, 0, 0);
                    URL.revokeObjectURL(urlObject);
                    delete event
                };
                delete image
                delete event
            };
        }


        function showHide(id_1, id_2, flag) {
            var xmlHttp = new XMLHttpRequest();
            var button = document.getElementById('start');
            if (document.getElementById(id_1, id_2)){
                var obj = document.getElementById(id_1);
                var obj2 = document.getElementById(id_2);

                if (obj2.style.display == "none") {
                    xmlHttp.open( "GET", "/video_start", false ); // false for synchronous
request
                    xmlHttp.send( null );
                    if (flag === 1) {
                        openSocket();
                    }
                    if (flag === 2) {
```

```
                openSocketFaceRecognition();
            }
            obj.style.display = "none"; // hide preview
            obj2.style.display = "inline-block"; // show video
            button.innerText = 'Зупинити';
            button.dataset.trigger = false;
            btn_photo.style.display = "inline-block";
        }
        else {
            xmlHttp.open( "GET", "/video_stop", false ); // false for synchronous
request
            xmlHttp.send( null );
            obj.style.display = "block"; // show preview
            obj2.style.display = 'none'; // hide video
            button.innerText = 'Почати';
            button.dataset.trigger = true;
            btn_photo.style.display = "none";
        }
    }
    else alert("Элемент с id: " + element_id + " не найден!");
}

function showHideAddUser(id_1, id_2, id_3, flag) {
    var xmlHttp = new XMLHttpRequest();
    var button = document.getElementById('start');
    if (document.getElementById(id_1, id_2, id_3)){
        var obj = document.getElementById(id_1);
        var obj2 = document.getElementById(id_2);
        var obj3 = document.getElementById(id_3);

        if (obj2.style.display == "none") {
```

```
                xmlHttp.open( "GET", "/video_start", false ); // false for synchronous
request
                xmlHttp.send( null );
                if (flag === 1) {
                    openSocket();
                }
                if (flag === 2) {
                    openSocketFaceRecognition();
                }
                obj.style.display = "none"; // hide preview
                obj2.style.display = "inline-block"; // show video
                obj3.style.display = "inline-block";
                button.innerText = 'Зупинити';
                button.dataset.trigger = false;
                btn_photo.style.display = "inline-block";
            }
            else {
                xmlHttp.open( "GET", "/video_stop", false ); // false for synchronous
request
                xmlHttp.send( null );
                obj.style.display = "block"; // show preview
                obj2.style.display = 'none'; // hide video
                obj3.style.display = 'none';
                button.innerText = 'Почати';
                button.dataset.trigger = true;
                btn_photo.style.display = "none";
            }
        }
        else alert("Элемент с id: " + element_id + " не найден!");
    }
```

```
////////////////////////////// MULTI FORM //////////////////////////////

const prevBtns = document.querySelectorAll(".btn-prev");
const nextBtns = document.querySelectorAll(".btn-next");
const progress = document.getElementById("progress");
const formSteps = document.querySelectorAll(".form-step");
console.log(formSteps)
const progressSteps = document.querySelectorAll(".progress-step");

let formStepsNum = 0;

function updateFormSteps() {
//  console.log(formSteps)
  formSteps.forEach((formStep) => {
    formStep.classList.contains("form-step-active") &&
      formStep.classList.remove("form-step-active");
  });

  formSteps[formStepsNum].classList.add("form-step-active");
}

function updateProgressbar() {
  progressSteps.forEach((progressStep, idx) => {
    if (idx < formStepsNum + 1) {
      progressStep.classList.add("progress-step-active");
    }
    else {
      progressStep.classList.remove("progress-step-active");
    }
  });
```

```
  const progressActive = document.querySelectorAll(".progress-step-active");

  progress.style.width =
    ((progressActive.length - 1) / (progressSteps.length - 1)) * 100 + "%";
}

function buttonNext() {
    var name = document.getElementById("name").value;
    var address = document.getElementById("address").value;

    if (name != '' && address != '') {
        formStepsNum++;
        updateFormSteps();
        updateProgressbar();
        const labels = document.querySelectorAll('label');
        for (const label of labels) {
            label.classList.remove('red-text');
        }
    }
    else {
        document.getElementById('next-alert').style.display = 'inline-block'
        const labels = document.querySelectorAll('label');
        for (const label of labels) {
            label.classList.add('red-text');
        }
    }
}

function buttonPrevios() {
    formStepsNum--;
    updateFormSteps();
```

```
        updateProgressbar();

    }

    ////////////////////////////  FACE RECOGNITION  ////////////////////////////

    let encodings = []
    let taken_photos_span = document.getElementById("saved-photos");

    let btn_photo = document.getElementById('photo')
    let photo_text = document.getElementById('photo-text')

    function takePhoto() {
        if (encodings.length < 5) {
            var xmlHttp = new XMLHttpRequest();
            var path = '/take_photo';
            xmlHttp.open( "GET", path, false ); // false for synchronous request
            xmlHttp.send( null );
            if (xmlHttp.responseText == '') {
                console.log('Webcam is turned off or face isn\'t found!')
            }
            else {
                encodings.push(xmlHttp.responseText)
                                        taken_photos_span.innerHTML =
(parseInt(taken_photos_span.innerHTML) + 1).toString()
                console.log(encodings)
            }
            if (encodings.length == 5) {
                btn_photo.classList.remove('btn');
                btn_photo.classList.remove('btn-photo');
                btn_photo.classList.add('btn-disabled');
```

```
            }
        }
        else {
            console.log
        }
    }

    ////////////////////////////  CUSTOM FORM MANAGER  ////////////////////////////

    const btn = document.querySelector('button');

    function submitForm() {
        const XHR = new XMLHttpRequest();

        let urlEncodedData = "",
        urlEncodedDataPairs = [],
        name;

        if (encodings.length == 5 && name != '' && address != '') {
            var data = {'name': document.getElementById("name").value,
                    'address': document.getElementById("address").value,
                    'encodings': encodings}
            console.log(data)


            // Turn the data object into an array of URL-encoded key/value pairs.
            for ( name in data ) {
                    urlEncodedDataPairs.push( encodeURIComponent( name ) + '=' +
encodeURIComponent( data[name] ) );
            }
```

```
            console.log(urlEncodedDataPairs)

             // Combine the pairs into a single string and replace all %-encoded spaces
to
            // the '+' character; matches the behaviour of browser form submissions.
            urlEncodedData = urlEncodedDataPairs.join( '&' ).replace( /%20/g, '+' );

            // Define what happens on successful data submission
        //      XHR.addEventListener( 'load', function(event) {
        //      alert( 'Yeah! Data sent and response loaded.' );
        //      } );

            // Define what happens in case of error
            XHR.addEventListener( 'error', function(event) {
            alert( 'Oops! Something went wrong.' );
            } );

            // Set up our request
            XHR.open( 'POST', '/add_user/', false );

            // Add the required HTTP header for form data POST requests
                    XHR.setRequestHeader(  'Content-Type',  'application/x-www-form-
urlencoded' );

            // Finally, send our data.
            XHR.send( urlEncodedData );

            buttonNext();
          }
          else {
            console.log('Some inputs aren\'t filled or photos aren\'t taken')
```

```
        document.getElementById('submit-alert').style.display = 'inline-block'
    }
}
```

ДОДАТОК Е

add_user.html

```
{% extends "base.html" %}

{% block head %}
<title>Форма реєстрації</title>
{% endblock %}

{% block body %}
<form action="/admin/submit_form/" method="POST" class="form">
   <h1 class="text-center">Форма реєстрації</h1>
   <!-- Progress bar -->
   <div class="progressbar">
      <div class="progress" id="progress"></div>
      <div class="progress-step progress-step-active" data-title="Дані"></div>
      <div class="progress-step" data-title="Розпізнавання"></div>
      <div class="progress-step" data-title="Завершення"></div>
   </div>

   <!-- Steps -->
   <div class="form-step form-step-active">
      <div class="input-group">
         <label for="name" >Ім'я *</label>
         <input type="text" name="name" id="name" required/>
      </div>
      <div class="input-group">
         <label for="address" >Адреса *</label>
         <input type="text" name="address" id="address" required/>
      </div>
      <div class="">
```

```
                    <a class="btn btn-next width-50 ml-auto" onclick="buttonNext()">Далі</a>
                </div>
                    <p id="next-alert" style="display: none;">Обов'язкові поля не заповнені!</p>
            </div>

            <div class="form-step">
                <h2>Розпізнавання обличчя</h2>
                    <p>Для внесення нового користувача до бази даних, необхідно отримати дані про риси обличчя користувача.</p>
                <p>Слідуйте наступній інструкції:</p>
                <ul>
                    <li>Увімкніть камеру, натиснувши на кнопку “Почати”.</li>
                    <li>Зробіть 5 фотографій, натискаючи на кнопку "Зробити фото"</li>
        <!--            <li>Уважно читайте кроки, які вам необхідно виконати після натиснення на кнопку.</li>-->
        <!--            <li>Після кожного виконаного кроку, необхідно натиснути кнопку “Зробити фото”.</li>-->
                        <li>Коли ви зробите 5 фотографій, необхідно зупинити трансляцію камери, натиснувши кнопку “Зупинити”.</li>
                    <li>Натисніть кнопку “Підтвердити та відправити”.</li>
                </ul>
                <div class="rectangle rectangle-100">
                    <img id="preview" src="../images/identification.png"/>
                        <canvas id="video" width="960" height="540" style="display: none; margin-left: auto; margin-right: auto;"/>
                </div>

                <div class="btns-group">
```

```
                <a id="start" class="btn-start" onclick="showHide('preview', 'video', 1)">Почати</a>
                    <a id="photo" class="btn btn-photo" onclick="takePhoto()" style="display: none;">Зробити фото</a>
            </div>

              <p id="photo-text" ><span id="saved-photos">0</span>/5 фотографій зроблено</p>

            <div class="btns-group">
              <a class="btn" onclick="buttonPrevios()">Попередній крок</a>
              <a class="btn" onclick="submitForm()">Підтвердити та відправити</a>
            </div>
            <p id="submit-alert" style="display: none;">Фотографії не зроблені</p>
          </div>

          <div class="form-step">
            <h2>Користувача було успішно додано!</h2>
            <a class="btn btn-center" href="/admin">Адмін сторінка</a>
          </div>
        </form>
        {% endblock %}


        admin.html
        {% extends "base.html" %}

        {% block head %}
        <title>Admin</title>
```

```
    <script src="https://kit.fontawesome.com/bfb9c3600a.js"
crossorigin="anonymous"></script>
    {% endblock %}

    {% block body %}
    <div class="content">
      <h1 style="text-align: center">Адмін панель</h1>
      <a href="/add_user/" class="add_btn">Додати запис</a>
      {% if users|length < 1 %}
      <h4 style="text-align: center">Користувачів ще немає</h4>
        <img id="preview" src="../images/void.png" style="width:40%; margin:0
auto;"/>
      {% else %}
      <table>
        <tr>
          <th>Ім'я</th>
          <th>Адреса</th>
          <th>Дата створення</th>
          <th>Дії</th>
        </tr>
        {% for user in users %}
        <tr>
          <td class="name_address">{{ user.name }}</td>
          <td class="name_address">{{ user.address }}</td>
          <td class="acion_date">{{ user.created_at.date() }}</td>
          <td class="action_date">
              <a href="/admin/users/delete/{{ user.id }}" class="button primary
delete"><i
                  class="fa-solid fa-trash-can"></i></a>
                <a class="button primary edit"><i class="fa-solid fa-pen-to-
square"></i></a>
```

```
            </td>
          </tr>
          {% endfor %}
        </table>
        {% endif %}

    </div>
    {% endblock %}


    base.html
    <!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <link rel="stylesheet" type="text/css" href="/static/main.css">
        <link rel="preconnect" href="https://fonts.googleapis.com">
        <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
                <link href="https://fonts.googleapis.com/css2?
family=Montserrat:wght@400;500;600&display=swap" rel="stylesheet">
                <link href="https://use.fontawesome.com/releases/v5.6.1/css/all.css"
rel="stylesheet">
        {% block head %}{% endblock %}
    </head>

    <body>
        <div id="stars"></div>
        <div id="stars2"></div>

        <nav class="header">
            <a href="/" class="logo">Дипломний проект</a>
```

```
        <div class="header-right">
            <a href="/admin">Адмін панель</a>
        </div>
    </nav>

    {% block body %}{% endblock %}
    <script src="/static/main.js"></script>
</body>
</html>
```

index.html

```
{% extends "base.html" %}

{% block head %}
<title>Users recognition</title>
{% endblock %}

{% block body %}
<header>
    <div>
        <h1>Сервіс з ідентифікації облич мешканців ЖК</h1>
    </div>

    <div>
        <h3>Як працює?</h3>
        <ul class="arrows">
            <li>Маємо БД з даними про мешканців ЖК</li>
            <li>Обличчя людини розпізнається на камері</li>
            <li>Програма вираховує, чи існує дане обличчя в БД ЖК</li>
            <li>Людина або пропускається в ЖК, або ні</li>
```

```
            </ul>
        </div>

        <div>
            <h2>Приклад ідентифікації</h2>
                <p class="small-description">(Ви можете додати своє обличчя в алгоритм розпізнавання на сторінці <a href="/admin">"Адмін панель"</a>)</p>
        </div>

        <div style="text-align: center" class="rectangle">
            <img id="preview" src="../images/identification.png" style = "border: 1px solid white;"/>
    <!--        <img id="webcam" src="" style="display: none;">-->
                <canvas id="video" width="960" height="540" style="display: none; margin-left: auto; margin-right: auto;"/>
        </div>
         <a id="start" class="btn btn-center btn-main" onclick="showHide('preview', 'video', 2)">Почати</a>
    </header>
    {% endblock %}
```

ДОДАТОК Ж

```
researching the face recognition algorithm.ipynb
import os
import face_recognition
import cv2
import pandas as pd
import numpy as np
import json
# reading dataset 'identity_CelebA'
df_identity = pd.read_csv('dataset/identity_CelebA.txt', sep=' ', names=['filename', 'identity'])
df_identity.head(3)
distribution = df_identity['identity'].value_counts()  # counting how many images there are for each identity
distr = distribution.loc[distribution > 4]          # leaving identities, who has 7+ images
idx_list = distr.index                              # getting indexes of the left identities
len(idx_list)
def create_identities_dict(row):
    if row['identity'] in identities_dict:
        identities_dict[row['identity']][0].append(row['filename'])
    else:
        identities_dict[row['identity']] = [[row[,filename']]]
%%time
identities_dict = {}

for index, row in df_identity.iterrows():
    create_identities_dict(row)

# creating identities/filenames DataFrame
```

```
df_if = pd.DataFrame.from_dict(identities_dict, orient='index').reset_index().rename(columns={"index": 'identities', 0: ,filenames'})

def image_to_encodings(filename):
    """
        - Gets filename, reads the image, finds faces, creates face encodings
        - Returns face encoding, if a face was found on the image
    """
    img = cv2.imread('dataset/img_align_celeba/' + filename)
    locations = face_recognition.face_locations(img, model='hog')
    encodings_ = face_recognition.face_encodings(img, locations)
    return encodings_
def image_to_encodings(filename):
    """
        - Gets filename, reads the image, finds faces, creates face encodings
        - Returns face encoding, if a face was found on the image
    """
    img = cv2.imread('dataset/img_align_celeba/' + filename)
    locations = face_recognition.face_locations(img, model='hog')
    encodings_ = face_recognition.face_encodings(img, locations)
    return encodings_
%%time
encodings, errors, seen_images, data = filenames_into_encodings(df_if)

df_fin = pd.DataFrame(data=data, columns=['identities', 'encodings'])
df_fin.to_csv('identities_encodings_super_final.csv', index=False)
df_fin.head(5)
def check_enc_quant(row):
    if len(json.loads(row['encodings'])) != 5:
        damaged_rows.append(row[,identities'])
```

```
def match_encoding(encodings_list, encoding, np_ar, method):
    """
        - Takes the list of all encodings, encoding which is going to be compared,
            parameters for np.arange function, which will choose the threshold and
method name
            - Depends on method name proceeds different algorithm of comparing
face encodings
            - Returns a list of predicted identities. List length equals to amount of
iterated thresholds

    """
    threshold_results = []
    encodings_list = np.array(encodings_list)
    encoding = np.array(encoding)


    if method == 'distance_threshold':
        results = face_recognition.face_distance(encodings_list, encoding)
        batch_size = 5
        for threshold in np.arange(np_ar[0], np_ar[1], np_ar[2]):
            matched_identity_idx = None
            for i in range(0, len(results), batch_size):
                mean_distance = np.mean(results[i:i+batch_size])
                if mean_distance < threshold:
                    matched_identity_idx = i // 5
                    break
            if matched_identity_idx is None:
                threshold_results.append(None)
            else:
                identity_actual = users[matched_identity_idx]
                threshold_results.append(identity_actual)
```

```
            elif method == 'min_distance':
                results = face_recognition.face_distance(encodings_list, encoding)
                batch_size = 5
                distances = []
                for i in range(0, len(results), batch_size):
                    mean_distance = np.mean(results[i:i+batch_size])
                    distances.append(mean_distance)
                matched_identity_idx = distances.index(np.min(distances))
                identity_actual = users[matched_identity_idx]
                threshold_results.append(identity_actual)



            else:
                for i in np.arange(np_ar[0], np_ar[1], np_ar[2]):
                        results = face_recognition.compare_faces(encodings_list, encoding,
tolerance=i)
                    if True in results:
                        if method == '5inarow':
                            batch_size = 5
                            matched_identity_idx = None
                            truth_list = []
                            for index in range(0, len(results), batch_size):
                                true_num = results[index:index+batch_size].count(True)
                                truth_list.append(true_num)
        #                         if true_num == 5:
        #                             matched_identity_idx = index // 5
        #                             break
                            matched_identity_idx = truth_list.index(max(truth_list))
```

```
                if matched_identity_idx is None:
                    threshold_results.append(None)

                else:
                    identity_actual = users[matched_identity_idx]
                    threshold_results.append(identity_actual)


            else:
                matched_identity_idx = results.index(True)
                identity_actual = users[matched_identity_idx]
                threshold_results.append(identity_actual)
        else:
            threshold_results.append(None)


    return threshold_results


def calc_acc(array: list, true_value: int):
    """
        - Takes array of predictions and a `true_value`
        - Compares and counts accuracy
        - Returns accuracy for a list
    """
    right_preds = 0
    for el in array:
        if el == true_value:
            right_preds += 1
    accuracy = right_preds / len(array)
    return accuracy
```

```
def get_predictions(row, encodings_list, np_ar, method: str, limit: int = 5) ->
list:
    """
      - Takes a row of `df_test_encodings` DataFrame
      - Goes into a for loop for each encoding and matches it to `encodings` list
      - Calls `match_encoding` function
    """
    predictions_for_each_image = []
    for encoding in row['5_encodings']:
      results = match_encoding(encodings_list, encoding, np_ar, method)

      if results is None:
        continue
      predictions_for_each_image.append(results)
    return predictions_for_each_image



def get_accuracy_by_threshold(row, results_by_threshold):
    """
      - Takes the dict from `get_results_by_threshold`
      - Calculates accuracy for each threshold
      - Return accuracies list
    """
    acc_by_threshold = []
    for key in results_by_threshold:
      acc = calc_acc(results_by_threshold[key], row['identities'])
      acc_by_threshold.append(acc)
    return acc_by_threshold
```

```
    def get_accuracy(row, predictions, np_ar, method):
        """
            - Takes the dict from `get_results_by_threshold`
            - Calculates accuracy for each threshold
            - Return accuracies list
        """
        acc_by_threshold = []
        if method == 'min_distance':
            acc = calc_acc(predictions[:, 0], row['identities'])
            acc_by_threshold.append(acc)
        else:
            length = len(np.arange(np_ar[0], np_ar[1], np_ar[2]))
            for i in range(length):
                acc = calc_acc(predictions[:, i], row['identities'])
                acc_by_threshold.append(acc)
        return acc_by_threshold


    def create_accuracy_column(df, encodings_list, np_ar, method: str):
        """
            - Takes `df_test` DataFrame
                                - Calls `get_predictions`, `get_results_by_threshold`,
`get_accuracy_by_threshold`
             - Returns list of lists (for each person) with accuracies (for each tolerance
threshold)
        """
        accuracy_list = []
        for index, row in df.iloc[8000:8200].iterrows():
    #     for index, row in df.iterrows():
```

```
            predictions_for_each_image = get_predictions(row, encodings_list, np_ar, method)
            predictions = np.array(predictions_for_each_image)
        #       print(predictions)
            acc_by_threshold = get_accuracy(row, predictions, np_ar, method)
            accuracy_list.append(acc_by_threshold)

        return accuracy_list


    def print_acc_by_threshold(accuracies, np_ar):
        accuracies = np.array(accuracies)

        for i, threshold in enumerate(np.arange(np_ar[0], np_ar[1], np_ar[2])):
                print(f'Accuracy with {threshold:.3f} threshold: {accuracies[:, i].mean():.2f}')

    def match_encoding(encodings_list, encoding, np_ar, method):
        """
        - Takes the list of all encodings, encoding which is going to be compared,
            parameters for np.arange function, which will choose the threshold and method name
            - Depends on method name proceeds different algorithm of comparing face encodings
            - Returns a list of predicted identities. List length equals to amount of iterated thresholds

        """
        threshold_results = []
        encodings_list = np.array(encodings_list)
        encoding = np.array(encoding)
```

```
if method == 'distance_threshold':
    results = face_recognition.face_distance(encodings_list, encoding)
    batch_size = 5
    for threshold in np.arange(np_ar[0], np_ar[1], np_ar[2]):
        matched_identity_idx = None
        for i in range(0, len(results), batch_size):
            mean_distance = np.mean(results[i:i+batch_size])
            if mean_distance < threshold:
                matched_identity_idx = i // 5
                break
        if matched_identity_idx is None:
            threshold_results.append(None)
        else:
            identity_actual = users[matched_identity_idx]
            threshold_results.append(identity_actual)


elif method == 'min_distance':
    results = face_recognition.face_distance(encodings_list, encoding)
    batch_size = 5
    distances = []
    for i in range(0, len(results), batch_size):
        mean_distance = np.mean(results[i:i+batch_size])
        distances.append(mean_distance)
    matched_identity_idx = distances.index(np.min(distances))
    identity_actual = users[matched_identity_idx]
    threshold_results.append(identity_actual)
```

```
            else:
                for i in np.arange(np_ar[0], np_ar[1], np_ar[2]):
                    results = face_recognition.compare_faces(encodings_list, encoding,
tolerance=i)
                if True in results:
                    if method == '5inarow':
                        batch_size = 5
                        matched_identity_idx = None
                        truth_list = []
                        for index in range(0, len(results), batch_size):
                            true_num = results[index:index+batch_size].count(True)
                            truth_list.append(true_num)
        #                     if true_num == 5:
        #                         matched_identity_idx = index // 5
        #                         break
                        matched_identity_idx = truth_list.index(max(truth_list))

                        if matched_identity_idx is None:
                            threshold_results.append(None)

                        else:
                            identity_actual = users[matched_identity_idx]
                            threshold_results.append(identity_actual)



                    else:
                        matched_identity_idx = results.index(True)
                        identity_actual = users[matched_identity_idx]
                        threshold_results.append(identity_actual)
                else:
                    threshold_results.append(None)
```

```
    return threshold_results


def calc_acc(array: list, true_value: int):
    """
        - Takes array of predictions and a `true_value`
        - Compares and counts accuracy
        - Returns accuracy for a list
    """
    right_preds = 0
    for el in array:
        if el == true_value:
            right_preds += 1
    accuracy = right_preds / len(array)
    return accuracy


def get_predictions(row, encodings_list, np_ar, method: str, limit: int = 5) ->
list:
    """
        - Takes a row of `df_test_encodings` DataFrame
        - Goes into a for loop for each encoding and matches it to `encodings` list
        - Calls `match_encoding` function
    """
    predictions_for_each_image = []
    for encoding in row['5_encodings']:
        results = match_encoding(encodings_list, encoding, np_ar, method)

        if results is None:
```

```
            continue
        predictions_for_each_image.append(results)
    return predictions_for_each_image


def get_accuracy_by_threshold(row, results_by_threshold):
    """
        - Takes the dict from `get_results_by_threshold`
        - Calculates accuracy for each threshold
        - Return accuracies list
    """
    acc_by_threshold = []
    for key in results_by_threshold:
        acc = calc_acc(results_by_threshold[key], row['identities'])
        acc_by_threshold.append(acc)
    return acc_by_threshold


def get_accuracy(row, predictions, np_ar, method):
    """
        - Takes the dict from `get_results_by_threshold`
        - Calculates accuracy for each threshold
        - Return accuracies list
    """
    acc_by_threshold = []
    if method == 'min_distance':
        acc = calc_acc(predictions[:, 0], row['identities'])
        acc_by_threshold.append(acc)
    else:
        length = len(np.arange(np_ar[0], np_ar[1], np_ar[2]))
        for i in range(length):
```

```
            acc = calc_acc(predictions[:, i], row['identities'])
            acc_by_threshold.append(acc)
        return acc_by_threshold


    def create_accuracy_column(df, encodings_list, np_ar, method: str):
        """
        - Takes `df_test` DataFrame
                - Calls `get_predictions`, `get_results_by_threshold`, `get_accuracy_by_threshold`
        - Returns list of lists (for each person) with accuracies (for each tolerance threshold)
        """
        accuracy_list = []
        for index, row in df.iloc[8000:8200].iterrows():
    #     for index, row in df.iterrows():
            predictions_for_each_image = get_predictions(row, encodings_list, np_ar, method)
            predictions = np.array(predictions_for_each_image)
    #         print(predictions)
            acc_by_threshold = get_accuracy(row, predictions, np_ar, method)
            accuracy_list.append(acc_by_threshold)

        return accuracy_list


    def print_acc_by_threshold(accuracies, np_ar):
        accuracies = np.array(accuracies)

        for i, threshold in enumerate(np.arange(np_ar[0], np_ar[1], np_ar[2])):
```

```
                print(f'Accuracy with {threshold:.3f} threshold: {accuracies[:, i].mean():.2f}’)

    %%time
    ident_accs_min_dist = create_accuracy_column(df_test_read, encodings, np_ar=(0.38, 0.451, 0.01), method=,min_distance')
    print(f'{np.mean(ident_accs_min_dist):.2f}')
```