Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування вищого навчального закладу)
Навчально-науковий інститут інформаційних технологій та робототехніки
(повна назва факультету)
Кафедра комп'ютерних та інформаційних технологій і систем
(повна назва кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістра

(освітньо-кваліфікаційний рівень)

на тему

Інтелектуальна система обліку робочого часу на основі розпізнавання облич за допомогою штучного інтелекту

Виконав: студент 6 курсу, групи 601-ТН
спеціальності

Комп'ютерні науки
(шифр і назва напряму)

Аксьонов М.А.
(прізвище та ініціали)

Керівник Кулик В.А.
(прізвище та ініціали)

Рецензент Підяшенко В. Є.
(прізвище та ініціали)

Полтава – 2025 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ « ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

спеціальність 122 «Комп'ютерні науки» на тему

Інтелектуальна система обліку робочого часу на основі розпізнавання облич за допомогою штучного інтелекту»

Студента групи 601-ТН Аксьонова Максима Андрійовича

Керівник роботи
д.е.н., проф. Кулик В.А.

Консультант
кандидат технічних наук,
доцент Скакаліна О.В.

Завідувач кафедри
кандидат фізико-математичних наук,
доцент Двірна О.А.

Полтава – 2025

РЕФЕРАТ

Кваліфікаційна робота магістра: 101 с., 23 малюнки, 1 таблиця, 2 додаток, 21 джерело.

Об’єкт дослідження: Об'єктом дослідження у дипломній роботі є система розпізнавання обличчя – технологія, що використовує алгоритми комп'ютерного зору та машинного навчання для автоматизації процесу ідентифікації осіб у зображеннях або відео.

Мета роботи: Метою дипломної роботи є розробка та впровадження системи розпізнавання обличчя для автоматизації обліку відвідуваності. Головна мета полягає у створенні ефективної, безпечної та зручної системи, яка забезпечує точне розпізнавання осіб і задовольняє потреби користувачів у навчальних або робочих середовищах.

Методи: Аналіз літературних джерел і наукових статей, присвячених технологіям розпізнавання обличчя, архітектурі систем комп'ютерного зору, протоколам роботи з базами даних та методам обробки зображень. Проектування архітектури системи розпізнавання обличчя, включаючи визначення бази даних, логіки роботи алгоритмів та структури користувацького інтерфейсу. Реалізація системи за допомогою таких технологій, як Python, OpenCV, Firebase, DeepFace, Flask та бібліотека для роботи з нейронними мережами TensorFlow. Тестування функціональності системи, перевірка точності алгоритмів розпізнавання та їх продуктивності при різних умовах використання. Оцінка роботи системи, порівняння отриманих результатів із існуючими рішеннями, визначення переваг і перспектив використання розробленої системи.

Ключові слова: розпізнавання обличчя, комп’ютерний зір, машинне навчання, облік відвідуваності, архітектура, проектування, тестування, валідація, ефективність

ABSTRACT

Master's qualification thesis: 101 p., 23 figures, 1 table, 2 appendices, 21 sources.

Object of Research: The object of research in this thesis is the face recognition system – a technology that uses computer vision and machine learning algorithms to automate the process of identifying individuals in images or videos.

Purpose of the Work: The purpose of this thesis is to develop and implement a face recognition system for automating attendance tracking. The main goal is to create an efficient, secure, and user-friendly system that ensures accurate face recognition and meets the needs of users in educational or work environments.

Methods: Analysis of literature sources and scientific articles related to face recognition technologies, computer vision system architectures, database interaction protocols, and image processing methods. Designing the architecture of the face recognition system, including defining the database, logic of algorithm operations, and user interface structure. Implementation of the system using technologies such as Python, OpenCV, Firebase, DeepFace, Flask, and the TensorFlow neural network library. Testing the functionality of the system, verifying the accuracy of recognition algorithms and their performance under different usage conditions. Evaluating the system’s performance, comparing the results with existing solutions, and identifying the advantages and future prospects of using the developed system.

Keywords: face recognition, computer vision, machine learning, attendance tracking, architecture, design, testing, validation, efficiency

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

API — Інтерфейс програмування додатків.

DB — База даних.

GUI — Графічний інтерфейс користувача.

SDK — Набір для розробки програмного забезпечення.

AI — Штучний інтелект.

ML — Машинне навчання.

CNN — Згорткова нейронна мережа.

IoT — Інтернет речей.

JSON — Формат обміну даними.

URL — Адреса ресурсу в Інтернеті.

SQL — Мова структурованих запитів.

HTTPS — Захищений протокол передачі гіпертексту.

OCR — Оптичне розпізнавання символів.

UI — Інтерфейс користувача.

UX — Досвід користувача.

Firebase — Платформа для мобільних і веб-додатків.

AI/ML модель — Модель для вирішення завдань за допомогою AI/ML.

WebSocket — Протокол двостороннього зв'язку.

JWT — Токен для автентифікації.

SSL/TLS — Протоколи безпеки зв'язку.

BLOB — Великий двійковий об'єкт.

ВСТУП

Сучасний світ все частіше звертається до інтелектуальних технологій для оптимізації різних аспектів життя та бізнесу. Однією з таких інновацій є застосування штучного інтелекту для автоматизації обліку робочого часу, що дозволяє підвищити прозорість, точність і ефективність управління персоналом.

Інтелектуальні системи обліку робочого часу на основі розпізнавання облич забезпечують автоматичну реєстрацію присутності працівників, врахування понаднормової роботи та розрахунок заробітної плати. Це рішення мінімізує вплив людського фактора та сприяє підвищенню продуктивності організації.

Дипломна робота присвячена розробці системи обліку робочого часу на основі технології розпізнавання облич за допомогою штучного інтелекту. Основною метою є створення програмного рішення, яке автоматизує облік, забезпечить зручний інтерфейс користувача та інтеграцію з базами даних для збереження й аналізу інформації.

У роботі буде розглянуто алгоритми розпізнавання облич, захист і конфіденційність даних, а також ключові етапи створення системи, включаючи проектування, реалізацію та тестування. Основними технологіями є Python, Flask, OpenCV та бібліотека face_recognition, що забезпечують ефективність і надійність.

Ця робота демонструє зацікавлення у застосуванні штучного інтелекту для розв’язання прикладних задач. Її результатом стане інструмент, що автоматизує адміністративні завдання й покращить управління робочими процесами в організаціях.

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД ПОНЯТЬ СИСТЕМИ ОБЛІКУ

1.1 Опис предметної області

Автоматизація управлінських процесів є важливим завданням сучасного бізнесу. Однією з ключових складових цього процесу є облік робочого часу працівників, що дозволяє контролювати присутність, ефективність роботи та забезпечувати точний розрахунок заробітної плати. Традиційні методи, такі як табелі обліку чи магнітні картки, мають значні недоліки, включаючи залежність від людського фактора, ризик фальсифікації даних і складність інтеграції з іншими системами.

Інтелектуальні системи обліку робочого часу, які базуються на технології розпізнавання облич, вирішують ці проблеми завдяки автоматизації процесів та підвищенню точності збору даних. Такі системи використовують алгоритми штучного інтелекту для ідентифікації особи за унікальними біометричними характеристиками. Вони забезпечують швидке та зручне визначення присутності працівника без необхідності фізичного контакту з пристроями, що є важливим у сучасних умовах.

Основними перевагами таких систем є:

- прозорість та точність – автоматизовані алгоритми виключають можливість людських помилок або навмисного втручання;
- швидкість і зручність – технологія дозволяє здійснювати ідентифікацію за лічені секунди без додаткових дій від працівника;
- інтеграція – системи можуть взаємодіяти з іншими програмами для розрахунку заробітної плати, складання графіків роботи та аналітики продуктивності.

Використання систем обліку на основі розпізнавання облич актуальне для різних сфер, таких як:

- корпоративний сектор – контроль відвідування офісів, управління графіками працівників;
- промисловість – моніторинг присутності на виробництві, підрахунок понаднормових годин;
- освіта – облік відвідувань студентів чи викладачів;
- медицина – контроль роботи медичного персоналу в режимі 24/7.

У сучасних умовах важливим є також питання безпеки та конфіденційності. Біометричні дані є чутливими, тому системи повинні відповідати міжнародним стандартам захисту даних, забезпечуючи їх шифрування та обмежений доступ.

Розробка системи обліку робочого часу на основі розпізнавання облич відкриває широкі можливості для автоматизації рутинних процесів, зменшення витрат на адміністративні задачі та підвищення продуктивності компаній. У подальших розділах будуть детально розглянуті вимоги до системи, її архітектура та методи реалізації [1].

1.2 Огляд існуючих програмних рішень

На сьогоднішній день на ринку представлені численні програмні рішення для обліку робочого часу, кожне з яких має свої особливості, функціональність та цільову аудиторію. У цьому розділі розглянемо основні види таких систем, їхні можливості, переваги та недоліки, а також порівняємо популярні рішення, що використовуються в різних організаціях. Завдяки швидкому розвитку технологій, ці системи постійно вдосконалюються, що дозволяє знизити витрати часу та підвищити ефективність управління персоналом.

Існуючі рішення для обліку робочого часу можна умовно розділити на кілька категорій. Серед традиційних систем з фізичними пристроями, що використовують магнітні або RFID-картки для реєстрації часу, можна відзначити їхню простоту та доступність. Біометричні пристрої, що здійснюють

ідентифікацію за відбитками пальців або обличчям, надають високий рівень безпеки та зручності, оскільки не вимагають носіння додаткових карток. Хмарні сервіси, як інтернет-платформи для автоматизації обліку часу з доступом через веб-браузер або мобільний додаток, дозволяють зручно керувати обліком часу з будь-якої точки світу, зменшуючи залежність від фізичних пристроїв. Корпоративні системи з інтеграцією з HRM-системами або ERP-платформами дають можливість автоматизувати управління персоналом і забезпечити єдиний інформаційний простір для компанії. Останнім часом набирають популярності системи з використанням штучного інтелекту, які використовують розпізнавання облич для безконтактної реєстрації присутності, що забезпечує високу точність та зручність для користувачів.

Аналіз переваг і недоліків існуючих систем показує, що більшість рішень орієнтовані на автоматизацію процесів, однак вони мають свої обмеження. Вартість впровадження таких систем, як біометричні технології та платформи з використанням штучного інтелекту, може бути значною на початковому етапі, що обмежує доступність для малих і середніх підприємств. Хмарні сервіси вимагають стабільного інтернет-з'єднання, що може бути проблематичним у регіонах з поганим покриттям, тоді як локальні рішення потребують спеціалізованого обладнання. Технічні обмеження, зокрема системи на основі розпізнавання облич, можуть працювати зі збоями за поганого освітлення або значної зміни зовнішності, що знижує їх ефективність у реальних умовах. Використання біометричних даних також вимагає суворого дотримання законодавства про захист персональної інформації, що створює додаткові труднощі для компаній, що працюють на міжнародному рівні.

Існуючі програмні рішення для обліку робочого часу охоплюють широкий спектр можливостей, але водночас мають обмеження, що спонукають до розробки нових систем. Особливу увагу слід приділити використанню штучного інтелекту для покращення точності, зручності та ефективності таких платформ. Це обумовлює актуальність дослідження та розробки інтелектуальної системи, яка поєднуватиме переваги сучасних рішень [2].

Популярні рішення на ринку (табл. 1.1).

Таблиця 1.1 – Порівняльна таблиця мов програмування

Система	Ключові особливості	Переваги	Недоліки
TMetric	Хмарний сервіс для відстеження робочого часу через веб-інтерфейс. Ручний та автоматичний трекінг	Простота використання, гнучка система налаштувань.	Обмежена функціональність у безкоштовній версії, необхідність ручного введення деяких даних.
Kronos Workforce Ready	Комплексна HRM-система: облік робочого часу, управління графіками, аналіз продуктивності.	Потужні аналітичні інструменти, висока масштабованість.	Висока вартість впровадження та обслуговування, складність інтеграції.
WorkTime	Програмне забезпечення для моніторингу робочого часу в реальному часі.	Широкий функціонал для моніторингу ефективності, зручний інтерфейс.	Питання конфіденційності, може сприйматися як суворий контроль.
Face Recognition Attendance System	Використання технології розпізнавання облич для реєстрації часу прибуття та відбуття.	Висока точність, безконтактна ідентифікація, зменшення шахрайства.	Залежність від якості обладнання, зниження ефективності при поганому освітленні або зміні зовнішності.
BioTime	Біометрична система з відбитками пальців, обличчям, райдужкою ока. Інтеграція з базами даних.	Багатоваріантність ідентифікації, висока безпека.	Складність налаштування, вимоги до спеціалізованого обладнання.

1.3 Функціональні та нефункціональні вимоги

Для забезпечення ефективності та відповідності системи потребам користувачів необхідно визначити чіткі функціональні та нефункціональні вимоги до її розробки.

Функціональні вимоги охоплюють основну функціональність системи, необхідну для її роботи:

- система повинна розпізнавати обличчя працівників з використанням камер стандартної якості;
- реєстрація часу входу та виходу кожного працівника має здійснюватися автоматично з можливістю перегляду історії;
- забезпечення функції обчислення робочих годин, включаючи понаднормові, на основі заданих графіків;
- автоматизований розрахунок заробітної плати з урахуванням тарифної ставки та понаднормових;
- можливість реєстрації нових працівників у базі даних шляхом збереження їхніх фотографій та персональних даних;
- забезпечення експорту даних про робочий час та заробітну плату у форматах Excel або PDF для подальшого використання;
- налаштування прав доступу для адміністраторів, які можуть переглядати та змінювати дані працівників;
- інтеграція з базою даних для збереження інформації про облік робочого часу та заробітну плату.

Нефункціональні вимоги визначають якісні характеристики системи:

- система повинна працювати з високою точністю розпізнавання облич навіть у складних умовах освітлення;
- забезпечення захисту персональних даних через шифрування та обмежений доступ до них;

- інтерфейс системи має бути простим і зручним для користувачів із мінімальними технічними навичками;
- висока продуктивність системи, що дозволяє обробляти до 100 розпізнавань облич за хвилину;
- можливість масштабування для підтримки більших баз даних у разі збільшення кількості працівників;
- надійність роботи, що передбачає відсутність збоїв та мінімізацію ризиків втрати даних;
- відносно низькі вимоги до апаратного забезпечення, що дозволяють використовувати систему на стандартних офісних комп'ютерах або серверах.

Ці вимоги є основою для розробки системи, яка відповідатиме сучасним стандартам, забезпечуючи ефективність, зручність і надійність у використанні [3].

1.4 Вимоги до безпеки та захисту даних

Безпека та захист даних є критично важливими аспектами при розробці інтелектуальної системи обліку робочого часу. Оскільки система працює з персональними даними працівників та біометричною інформацією, необхідно врахувати такі вимоги до її безпеки:

Шифрування даних: Усі дані, що зберігаються у базі даних або передаються між клієнтською та серверною частинами, мають бути зашифровані за допомогою сучасних алгоритмів.

Контроль доступу: Доступ до даних системи повинен бути обмежений відповідно до ролей користувачів. Адміністратори мають доступ до всіх функцій, а звичайні працівники – лише до своїх особистих даних.

Автентифікація користувачів: Для забезпечення надійності автентифікації система повинна підтримувати багаторівневу перевірку, наприклад, через пароль і двофакторну аутентифікацію.

Захист біометричних даних: Біометричні дані (зображення облич) повинні зберігатися у вигляді хешованих або зашифрованих даних, що унеможливлює їхнє відновлення у первинному вигляді.

Логи доступу та дій: Система має вести записи про всі дії користувачів, включаючи спроби входу, перегляд та зміну даних, що дозволить відстежувати та аналізувати потенційно небезпечні дії.

Захист від несанкціонованого доступу: Серверна частина системи повинна бути захищена за допомогою міжмережевого екрану (Firewall) та сучасних методів виявлення загроз, таких як IDS/IPS (системи виявлення та запобігання вторгнень).

Резервне копіювання даних: Регулярне резервне копіювання має гарантувати збереження даних у разі збоїв або кібератак. Збереження резервних копій повинно здійснюватися у зашифрованому вигляді на окремих серверах або у хмарному середовищі.

Оновлення та підтримка: Система повинна регулярно оновлюватися для усунення вразливостей та забезпечення відповідності сучасним стандартам безпеки.

Відповідність регуляторним стандартам: Система має відповідати вимогам міжнародних і локальних стандартів щодо захисту персональних даних, таких як GDPR, CCPA або аналогічні.

Захист від атак типу "людина посередині" (MITM): Для захисту передачі даних система повинна використовувати протоколи HTTPS та SSL/TLS.

Дотримання цих вимог забезпечить надійний захист персональних та біометричних даних, мінімізуючи ризики несанкціонованого доступу, витоку або компрометації інформації [4].

1.5 Теоретичні основи розпізнавання облич за допомогою Python

Розпізнавання облич є однією з найактуальніших задач в області комп'ютерного зору та штучного інтелекту. Основною метою цієї задачі є ідентифікація або верифікація особи на зображеннях або відео. У сучасних системах розпізнавання облич використовуються різноманітні підходи та алгоритми, серед яких найбільш ефективними є методи машинного навчання, зокрема глибокі нейронні мережі.

Використання Python для розпізнавання облич стало стандартом у галузі завдяки потужним бібліотекам та інструментам, які спрощують розробку таких систем. Однією з найпопулярніших бібліотек для роботи з зображеннями та відео є OpenCV (Open Source Computer Vision Library), що забезпечує широкі можливості для обробки зображень та відео. Ця бібліотека підтримує численні алгоритми для детекції облич, таких як алгоритм Хаара (Haar cascades), що застосовується для виявлення облич на зображеннях.

Іншим важливим інструментом є бібліотека dlib, яка також має потужні алгоритми для розпізнавання облич та їхніх характеристик. Dlib реалізує методи, засновані на глибокому навчанні, зокрема використання нейронних мереж для виявлення облич, а також їхніх рис і відстаней між очима, носом та ротом, що дозволяє проводити точнішу ідентифікацію осіб [5].

Одним з ключових аспектів, який забезпечує високу точність розпізнавання, є використання алгоритмів глибокого навчання, таких як згорткові нейронні мережі (CNN). Ці алгоритми здатні вивчати особливості зображення на різних рівнях деталізації, від простих ознак (наприклад, контурів) до складних абстракцій (наприклад, форм обличчя). Завдяки цьому, вони можуть досягати високої точності при розпізнаванні навіть за умов поганого освітлення або при частковому перекритті обличчя.

Для вирішення задачі розпізнавання облич важливо також використовувати методи попередньої обробки зображень, такі як корекція яскравості та контрасту, фільтрація шумів, нормалізація розміру зображення та

інші. Вони дозволяють покращити якість вхідних даних, що в свою чергу підвищує ефективність алгоритмів розпізнавання.

Розпізнавання облич в реальному часі також є однією з ключових характеристик сучасних систем. Для цього використовуються оптимізовані алгоритми, здатні швидко обробляти потоки відео, не втрачаючи точності. Завдяки використанню Python та його потужних бібліотек, таких як OpenCV та dlib, можна реалізувати ефективні та масштабовані системи для реального часу [6].

Варто зазначити, що в останні роки популярними стали методи, що використовують попередньо натреновані моделі глибокого навчання для вирішення задачі розпізнавання облич. Наприклад, моделі, реалізовані в бібліотеці Face Recognition (яка є надбудовою над dlib), дозволяють значно спростити процес розпізнавання та досягти високої точності з мінімумом налаштувань.

Таким чином, використання Python та його бібліотек для розпізнавання облич є зручним і ефективним підходом для розробки систем, що використовують штучний інтелект для автоматизації таких процесів, як облік робочого часу, забезпечуючи високу точність, швидкість та масштабованість системи [7].

1.6 Постановка задачі

Метою дипломної роботи є розробка інтелектуальної системи обліку робочого часу на основі розпізнавання облич із використанням методів штучного інтелекту. Така система повинна забезпечувати автоматизацію обліку присутності працівників, розрахунок робочих годин, обчислення понаднормових та заробітної плати, а також гарантувати високий рівень безпеки даних.

Для досягнення цієї мети необхідно вирішити такі завдання:

- провести аналіз предметної області, зокрема вивчити сучасні підходи до обліку робочого часу та технології розпізнавання облич;
- ознайомитися з існуючими програмними рішеннями, що використовуються для подібних цілей, та виділити їхні переваги та недоліки;
- визначити функціональні та нефункціональні вимоги до розроблюваної системи, зокрема щодо її безпеки, точності та продуктивності;
- розробити архітектуру системи, яка включає серверну частину для обробки даних, клієнтську частину для взаємодії з користувачем та базу даних для зберігання інформації;
- реалізувати алгоритми розпізнавання облич з використанням технологій машинного навчання або готових бібліотек для роботи з комп'ютерним зором;
- забезпечити автоматизований облік робочого часу з інтеграцією з календарем або графіком змін працівників;
- розробити механізм обчислення заробітної плати, враховуючи тарифні ставки та понаднормові години;
- здійснити заходи щодо забезпечення безпеки системи, включаючи шифрування даних, захист доступу та резервне копіювання;
- провести тестування системи, включаючи перевірку точності розпізнавання облич, продуктивності та зручності використання;
- підготувати технічну документацію та рекомендації щодо впровадження системи в організаціях різного масштабу.

Розроблена система повинна бути інтегрованою, простою у використанні та здатною до масштабування відповідно до потреб підприємства. Вирішення поставлених завдань сприятиме підвищенню ефективності управління робочим часом, оптимізації витрат на адміністрування персоналу та забезпеченню прозорості в обліку праці [8].

РОЗДІЛ 2
ПРОЕКТУВАННЯ СИСТЕМИ ОБЛІКУ

2.1 Структура системи

Інтелектуальна система обліку робочого часу на основі розпізнавання облич використовує модульну архітектуру, що забезпечує високу гнучкість, масштабованість та зручність інтеграції. Основні компоненти системи включають веб-додаток, модуль обробки зображень, базу даних і інтерфейси для взаємодії з користувачем. Кожен компонент відповідає за окремий аспект роботи системи, що сприяє її ефективності та надійності.

Веб-додаток реалізований за допомогою Flask, легкого веб-фреймворка, який відповідає за обробку запитів від користувачів і керування веб-сервером. Завдяки використанню Flask система підтримує:

- обробку запитів на авторизацію адміністраторів;
- відображення записів про відвідуваність у зручному інтерфейсі;
- надання можливостей для керування списками працівників і відділів [9].

Додатково використовується бібліотека Werkzeug, яка дозволяє реалізувати завантаження файлів та управління сесіями.

Модуль обробки зображень виконує ключові завдання, пов'язані з розпізнаванням облич. Основні етапи роботи:

- захоплення зображень за допомогою вебкамери, реалізоване через бібліотеку OpenCV;
- попередня обробка зображень із використанням бібліотеки Pillow, яка додає можливості масштабування, обрізки та інших операцій із зображеннями;
- виявлення та розпізнавання облич із використанням алгоритмів штучного інтелекту;

- моделі тренуються для ідентифікації облич, забезпечуючи високу точність навіть у складних умовах освітлення [10].

Інтеграція з базою даних використовується для зберігання та обробки даних система використовує Firebase Realtime Database і Firebase Storage:

- у базі даних зберігаються записи про працівників, адміністраторів та результати розпізнавання;
- firebase Storage відповідає за зберігання зображень облич для подальшого навчання моделей або верифікації;
- використання Firebase забезпечує надійну і швидку синхронізацію даних у реальному часі між клієнтом і сервером.

Система пропонує такі функції:

- реальний час: Відмітка відвідуваності відбувається одразу після розпізнавання облич, а інформація синхронізується в базі даних;
- підтримка багатокласовості: Працівники можуть бути зареєстровані в кількох відділах, і система розділяє записи для кожного з них;
- безпечний доступ: Авторизований доступ забезпечує адміністраторам можливість переглядати дані без ризику несанкціонованого доступу.

Ключові бібліотеки та залежності. Система покладається на кілька важливих бібліотек, серед яких:

- numpy: Для роботи з багатовимірними масивами даних і виконання математичних операцій;
- firebase Admin SDK: Для взаємодії з хмарними сервісами Firebase;
- pillow: Для обробки зображень.

Інтерфейс користувача. Система оснащена зручним графічним інтерфейсом, який дозволяє адміністраторам та працівникам легко працювати з даними відвідуваності. Підтримка вебкамери забезпечує просте розгортання і використання системи в будь-якому середовищі [11].

Ця багаторівнева архітектура робить систему легкою для масштабування та інтеграції з іншими платформами, що відкриває широкі перспективи для її використання в корпоративних та інших професійних середовищах.

2.2 Архітектура системи обліку

Архітектура інтелектуальної системи обліку робочого часу побудована на модульній основі, що забезпечує чіткий поділ функціональності між компонентами та спрощує масштабування й підтримку. Взаємодія між компонентами системи організована через централізовану базу даних Firebase, яка виступає ключовою ланкою для зберігання та обміну даними (рис. 2.1).

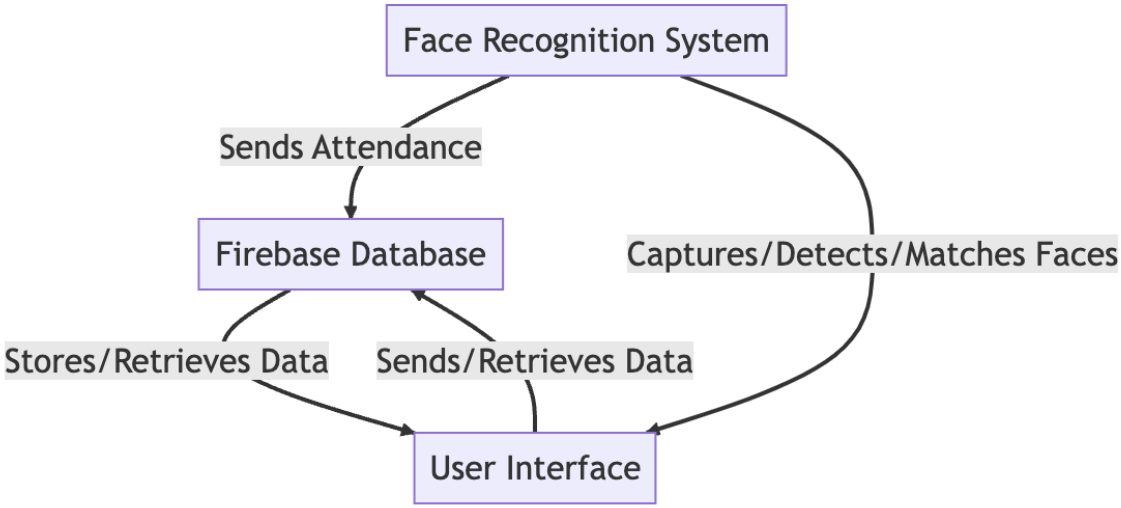

Рисунок 2.1 – Схема системи

Система розпізнавання облич

Цей модуль є ядром системи, побудованим за допомогою Python та OpenCV. Процес включає захоплення зображень або відеокадрів через вебкамеру, виявлення облич за допомогою класифікаторів типу Haar Cascade, вирівнювання облич для коректної орієнтації зображень, екстракцію ознак через DeepFace з моделлю FaceNet для створення унікальних векторних представлень, та порівняння цих облич з базою даних через обчислення схожості векторів.

Інтерфейс користувача

Веб-інтерфейс створений з використанням HTML, CSS та JavaScript для забезпечення доступу до функціоналу системи. Користувачі мають можливість авторизуватися, переглядати дані про відвідуваність, створювати нові групи, а також отримувати візуалізацію інформації з бази даних Firebase.

База даних Firebase

Firebase використовується як централізоване сховище для інформації про користувачів, їхні обличчя в вигляді векторних уявлень, а також відвідуваність. База даних синхронізує дані в реальному часі і надає API для взаємодії з іншими компонентами системи.

Взаємодія між компонентами

Система розпізнавання облич виявляє і розпізнає обличчя, після чого надсилає ідентифікатор користувача та часову мітку до Firebase для фіксації відвідуваності. Інтерфейс користувача здійснює запити до Firebase для отримання інформації про відвідуваність, списки працівників, їх розклад, а також для додавання нових записів. Firebase здійснює всі операції зберігання, оновлення та надання даних, зберігаючи зображення користувачів для подальшого використання.

Автентифікація та безпека

Система використовує Firebase Authentication для автентифікації користувачів і забезпечує доступ лише авторизованим особам.

Взаємодія з обчислювальними модулями

Обчислення і обробка ембеддінгів відбувається на серверній стороні для зниження навантаження на клієнтські пристрої. Всі необхідні дані для розпізнавання облич завантажуються з Firebase у вигляді збережених векторних уявлень.

Архітектура дозволяє легко додавати нові функції, такі як інтеграція з іншими системами обліку чи розширення аналітичних можливостей.

Ця архітектура забезпечує високу надійність, продуктивність та безпеку для ефективного функціонування системи в реальних умовах [12].

2.3 Вибір платформи для бази даних

Платформа Firebase була обрана для реалізації системи обліку робочого часу через її потужний функціонал, масштабованість та зручність інтеграції з іншими компонентами проєкту. Firebase пропонує повний набір інструментів для розробки сучасних веб- та мобільних додатків, включаючи базу даних у реальному часі, хмарне сховище, автентифікацію користувачів і можливості для аналізу даних.

1. Firebase Realtime Database

Основним компонентом для зберігання даних у системі є Realtime Database. Ця хмарна NoSQL-база даних забезпечує швидке та надійне збереження й обмін даними в реальному часі. Переваги використання Realtime Database.

Реальна синхронізація. Дані автоматично оновлюються на всіх підключених клієнтах без необхідності повторного запиту.

Простота інтеграції. Бібліотека Firebase SDK надає готові рішення для взаємодії з базою даних із використанням Python, що скорочує час розробки.

Структуроване зберігання. База даних дозволяє створювати вкладені структури, що спрощує організацію даних, таких як інформація про працівників, їх відвідуваність і метадані облич.

2. Firebase Storage

Для зберігання зображень облич працівників використовується Firebase Storage. Це дозволяє безпечно зберігати файли, забезпечуючи легкий доступ до них для подальшого використання системою розпізнавання облич. Основні переваги.

Гнучке управління файлами. Зображення зберігаються в структурованій ієрархії (наприклад, static/images/{student_id}.jpg), що забезпечує зручність доступу.

Шифрування та безпека. Усі дані автоматично шифруються під час зберігання та передачі, що гарантує їх конфіденційність.

Масштабованість. Firebase Storage адаптується до збільшення обсягу даних без необхідності внесення змін у код або налаштування.

3. Firebase Authentication

Для забезпечення безпеки системи та запобігання несанкціонованому доступу використовується Firebase Authentication. Цей компонент забезпечує.

Підтримку кількох методів входу. Працівники та адміністратори можуть входити в систему за допомогою електронної пошти та пароля.

Легку інтеграцію з іншими сервісами Firebase. Дані авторизації можуть бути пов'язані з записами в базі даних, що спрощує керування правами доступу.

Безпека. Firebase використовує сучасні методи шифрування і гарантує захист даних під час входу.

4. Висока доступність і продуктивність

Firebase працює на інфраструктурі Google, що гарантує.

Високу доступність. Дані доступні в будь-який час, що критично для систем, які працюють у реальному часі.

Швидкий доступ. Firebase оптимізований для низької затримки передачі даних, що дозволяє швидко обробляти запити та відображати результати у веб-інтерфейсі.

5. Інструменти для аналітики

Firebase також пропонує можливості для аналізу даних і моніторингу роботи системи. Firebase Analytics – це інструмент для аналізу поведінки користувачів і відстеження ключових показників продуктивності.

Firebase Performance Monitoring дозволяє відстежувати швидкість роботи системи, виявляти вузькі місця та оптимізувати їх.

6. Масштабованість і глобальна доступність

Платформа Firebase масштабована під час зростання навантаження на систему. Це важливо для підтримки великої кількості користувачів. Глобальна інфраструктура Firebase дозволяє зменшити затримки доступу до даних для користувачів у різних регіонах.

7. Зручність розробки

Інтеграція Firebase у проєкт значно спрощує процес розробки завдяки:

- докладній документації.
- підтримці бібліотек для основних мов програмування, включаючи Python.
- автоматичній обробці складних завдань, таких як синхронізація даних, кешування та управління файлами.

Вибір Firebase як платформи для реалізації системи обліку робочого часу є оптимальним завдяки її багатофункціональності, високій продуктивності, легкості інтеграції та безпеці. Всі компоненти Firebase гармонійно працюють разом, забезпечуючи простоту реалізації, ефективність і надійність системи [13].

2.4 Вибір мови програмування та середовища розробки

У розробці інтелектуальної системи обліку робочого часу з використанням технології розпізнавання облич було обрано Python як основну мову програмування та ряд додаткових інструментів для створення і підтримки системи. Вибір Python як мови програмування та середовища розробки має ряд обґрунтованих переваг, зокрема з точки зору ефективності, зручності використання бібліотек для обробки зображень і штучного інтелекту, а також доступності сучасних фреймворків для побудови веб-застосунків.

1. Переваги вибору мови Python

Python є однією з найпопулярніших мов програмування для розробки інтелектуальних систем, особливо в контексті обробки даних, штучного інтелекту та машинного навчання. Основні переваги Python для даного проєкту:

Простота синтаксису: Python має зрозумілу та легку для освоєння синтаксичну структуру, що дозволяє швидко розробляти ефективний код навіть для складних проєктів. Це дозволяє прискорити етапи розробки, тестування та налагодження програмного забезпечення.

Широка підтримка бібліотек для штучного інтелекту та комп'ютерного зору: Python є лідером на ринку бібліотек і фреймворків для машинного навчання та обробки зображень, таких як OpenCV, TensorFlow, Keras, PyTorch, scikit-learn, DeepFace тощо. Ці інструменти дозволяють ефективно реалізовувати систему розпізнавання облич та інші інтелектуальні функції без необхідності розробляти складні алгоритми з нуля.

Масштабованість та гнучкість: Python дозволяє створювати як прості, так і складні системи, що робить його універсальним інструментом для розробки від початкових прототипів до повноцінних промислових застосунків.

Велика спільнота розробників: Завдяки популярності мови, Python має велику спільноту розробників, що постійно створюють нові бібліотеки, документацію та інші ресурси. Це дозволяє швидко знаходити рішення для різних технічних проблем та отримувати допомогу при розробці [14].

Інтеграція з іншими мовами та інструментами: Python добре інтегрується з іншими мовами програмування та інструментами, що дозволяє використовувати різні технології в одному проєкті.

2. Особливості використання Python для розпізнавання облич

Python є ідеальним вибором для обробки зображень і відео, а також для розпізнавання облич завдяки великій кількості готових бібліотек для комп'ютерного зору. Наприклад:

OpenCV: Це одна з найпопулярніших бібліотек для роботи з зображеннями та відео. Вона дозволяє виконувати низку операцій, таких як обробка зображень, виявлення об'єктів, розпізнавання облич, визначення віку та статі людини, виявлення очей, тощо. В OpenCV є інструменти для роботи з різними форматами зображень і відео, що дозволяє ефективно обробляти дані у реальному часі.

DeepFace: Ця бібліотека для Python надає простий інтерфейс для використання декількох передових алгоритмів для розпізнавання облич, таких як FaceNet, VGG-Face, OpenFace, DeepID та інші. Вона дозволяє швидко інтегрувати систему розпізнавання облич в Python-аплікації, що важливо для реалізації функціоналу системи обліку робочого часу.

dlib: Це ще одна популярна бібліотека для обробки зображень, яка забезпечує високу точність і ефективність при розпізнаванні облич. Вона використовує алгоритми для детекції та векторизації облич, що робить її ефективною в задачах розпізнавання осіб [15].

3. Вибір середовища розробки

Для створення веб-частини системи обліку робочого часу обрано Flask — легковаговий фреймворк для веб-розробки на Python. Ось чому Flask є оптимальним вибором для цього проєкту:

Легкість та мінімалізм: Flask надає лише основні функціональні можливості, не навантажуючи систему зайвими компонентами. Це дозволяє розробнику мати повний контроль над структурою додатка та вибором інструментів.

Гнучкість та розширюваність: Flask дозволяє інтегрувати сторонні бібліотеки та компоненти для розширення функціоналу. Це ідеально підходить для проєкту, який має специфічні вимоги до роботи з базами даних, автентифікації користувачів, інтеграції з Firebase та розпізнавання облич.

Широке використання: Flask широко використовується для створення веб-додатків, що дозволяє знаходити готові рішення для багатьох типових завдань.

Для інтеграції з Firebase, у проєкті буде використовуватися бібліотека firebase-admin для Python. Вона дозволяє взаємодіяти з базою даних Firebase, виконувати операції з даними, зберігати зображення та отримувати інформацію про користувачів і їхній статус.

4. Додаткові інструменти та технології

Для обробки зображень та даних у системі використовуються й інші бібліотеки Python:

Pillow: Бібліотека для обробки зображень, яка дозволяє виконувати різноманітні операції, такі як зміна розміру, обрізка, конвертація форматів, фільтри тощо.

numpy: Основна бібліотека для обчислень з багатовимірними масивами та матрицями. Вона активно використовується в обробці даних для машинного навчання та при виконанні математичних операцій з зображеннями [16].

Werkzeug: Це потужний набір інструментів для веб-розробки на Python, який надає функції для обробки запитів, роботи з сесіями, а також для завантаження файлів, що є важливим для обробки та збереження зображень користувачів.

5. Підтримка та документація

Однією з важливих причин вибору Python є наявність широкої документації, яка охоплює всі аспекти роботи з бібліотеками для обробки зображень, машинного навчання та веб-розробки. Також Python має великий обсяг відкритих ресурсів, що дозволяє швидко знаходити рішення для конкретних завдань у процесі розробки.

Вибір мови програмування Python для розробки інтелектуальної системи обліку робочого часу з використанням технології розпізнавання облич обґрунтований численними перевагами цієї мови, зокрема її універсальністю, широкою підтримкою бібліотек для машинного навчання та комп'ютерного зору, а також наявністю потужних фреймворків для веб-розробки. Це дозволяє розробникам ефективно та швидко реалізувати функціонал, що необхідний для створення зручної та надійної системи обліку робочого часу, яка може бути масштабована та легко інтегрована з іншими системами в майбутньому [17].

2.5 Проектування функціональності

Проектування функціональності інтелектуальної системи обліку робочого часу на основі розпізнавання облич є критичним етапом у створенні цієї системи, оскільки від якості її функціональних можливостей залежить ефективність роботи всіх користувачів та цілісність даних. Система повинна бути зручна, надійна, безпечна та інтегруватися з іншими компонентами, такими як база

даних та веб-інтерфейс. Основні аспекти проектування функціональності охоплюють різні рівні взаємодії користувачів з системою, обробку та збереження даних, а також інтеграцію з інтелектуальними алгоритмами для розпізнавання облич (рис. 2.2).

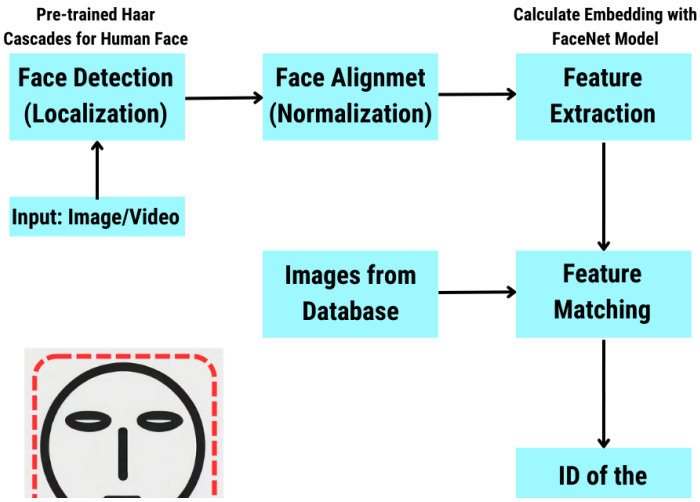

Рисунок 2.2 – Сторінка налаштування

Проектування функціональності системи передбачає реалізацію кількох ключових компонентів. Однією з основних функцій є розпізнавання облич, що дозволяє визначити особу користувача шляхом аналізу його обличчя за допомогою алгоритмів машинного навчання та комп'ютерного зору. Після успішного розпізнавання система автоматично фіксує час приходу користувача, створюючи запис у базі даних. Користувачі мають доступ до сторінок, де можуть переглядати свої записи, реєструватися або перевіряти результати відвідуваності. Безпека та автентифікація забезпечуються для доступу лише авторизованим користувачам до персональних даних та можливості внесення змін [18].

Основна частина функціональності системи полягає в розпізнаванні обличчя. Для цього використовуються етапи, такі як виявлення обличчя, коли система використовує алгоритми детекції, такі як Haar Cascade або Dlib, для виявлення обличчя на зображенні. Якість детекції має великий вплив на точність подальшого розпізнавання. Після виявлення обличчя система здійснює

покращення зображення для підвищення точності розпізнавання, включаючи корекцію освітлення та вирівнювання обличчя. Далі відбувається екстракція особливих ознак обличчя за допомогою таких методів, як FaceNet або DeepFace, які перетворюють обличчя на векторні представлення, що потім порівнюються з базою даних. Якщо система знаходить збіг, то реєструється успішний вхід і автоматично фіксується час відвідування.

Процес реєстрації користувачів є важливим для точного відстеження відвідуваності. Користувачі реєструються через веб-інтерфейс, надаючи свої персональні дані та фото для подальшого розпізнавання. Це дозволяє створити унікальний профіль для кожного користувача, що включає ідентифікатор користувача, фото обличчя та інформацію. Для забезпечення безпеки доступу до персональних даних система використовує механізм автентифікації за допомогою Firebase Authentication, що забезпечує надійну аутентифікацію працівників і адміністраторів. Використання токенів доступу допомагає обмежити доступ до функцій відповідно до ролі користувача.

Функціональність запису на тренінги є важливою для контролю відвідуваності. Працівники можуть реєструватися, при відвідуванні система фіксує час і дату, а також записує інформацію про присутність або відсутність працівника, залежно від результатів розпізнавання обличчя. Керівники мають змогу переглядати журнал відвідуваності в реальному часі та отримувати звіти про відсутніх або присутніх працівників.

Адміністратори або керівники мають доступ до панелі керування, де можуть створювати нові курси, визначати їх розклад, додавати або видаляти працівників. Вони також можуть переглядати звіти щодо відвідуваності та редагувати особисті дані працівників, змінювати налаштування їхніх курсів [19].

Firebase забезпечує збереження та обробку даних, зокрема інформації про користувачів, відвідуваність та фотографії облич. Дані зберігаються в Firebase Realtime Database, а фотографії працівників зберігаються в Firebase Storage. Для безпеки даних передбачено кілька рівнів захисту, зокрема шифрування даних та обмеження доступу до особистих записів лише авторизованими користувачами.

Інтерфейс користувача системи повинен бути інтуїтивно зрозумілим і зручним для працівників і керівників. Він включає панелі для реєстрації та входу, для перегляду розкладу, запису на тренінги та моніторингу відвідуваності.

Проектування функціональності системи обліку робочого часу на основі розпізнавання облич є складним і багатогранним процесом, що включає реалізацію ефективних алгоритмів для розпізнавання облич, зберігання даних у безпечній і масштабованій базі, а також створення зручного веб-інтерфейсу для користувачів. Основною метою є забезпечення точного, надійного та зручного механізму для автоматизації обліку робочого часу та підвищення ефективності роботи організацій.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ ОБЛІКУ

3.1 Розгортання хмарної інфраструктури

Для розробки програмного забезпечення, яке працює з даними, важливо створити стабільну, масштабовану та зручну інфраструктуру. У цій роботі для реалізації хмарних сервісів було обрано платформу Firebase, яка забезпечує функціональність зберігання даних, обміну файлами та автентифікації користувачів.

Спочатку було створено проєкт на платформі Firebase. Для цього використовували офіційний веб-інтерфейс, у якому виконали вхід до акаунта Google і вибрали опцію створення нового проєкту. Було вказано назву проєкту, наприклад, "face-recognition-app", після чого система автоматично налаштувала проєкт і підготувала його до роботи.

Наступним етапом стало налаштування бази даних у реальному часі (Realtime Database). Цей сервіс обрано для зберігання даних у структурованому вигляді з можливістю оновлення інформації в режимі реального часу. Базу даних було налаштовано в тестовому режимі, щоб під час розробки забезпечити відкритий доступ до зчитування та запису даних (рис. 3.1).

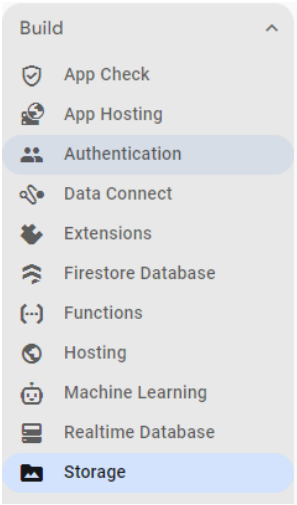

Рисунок 3.1 – Сторінка налаштування

Окрім цього, було активовано сервіс хмарного зберігання (Cloud Storage), який дозволяє зберігати файли, зокрема зображення. Для його активації використовувався розділ налаштувань платформи Firebase, де було обрано режим тестування, що спрощує інтеграцію на початкових етапах розробки [20].

Для підключення Firebase до програмного забезпечення було використано SDK (Software Development Kit). У конфігурації проєкту додано URL бази даних і хмарного зберігання, які надаються Firebase, а також згенеровано ключ доступу у форматі JSON. Цей ключ було імпортовано до проєкту для забезпечення зв'язку з хмарними сервісами (рис. 3.2).

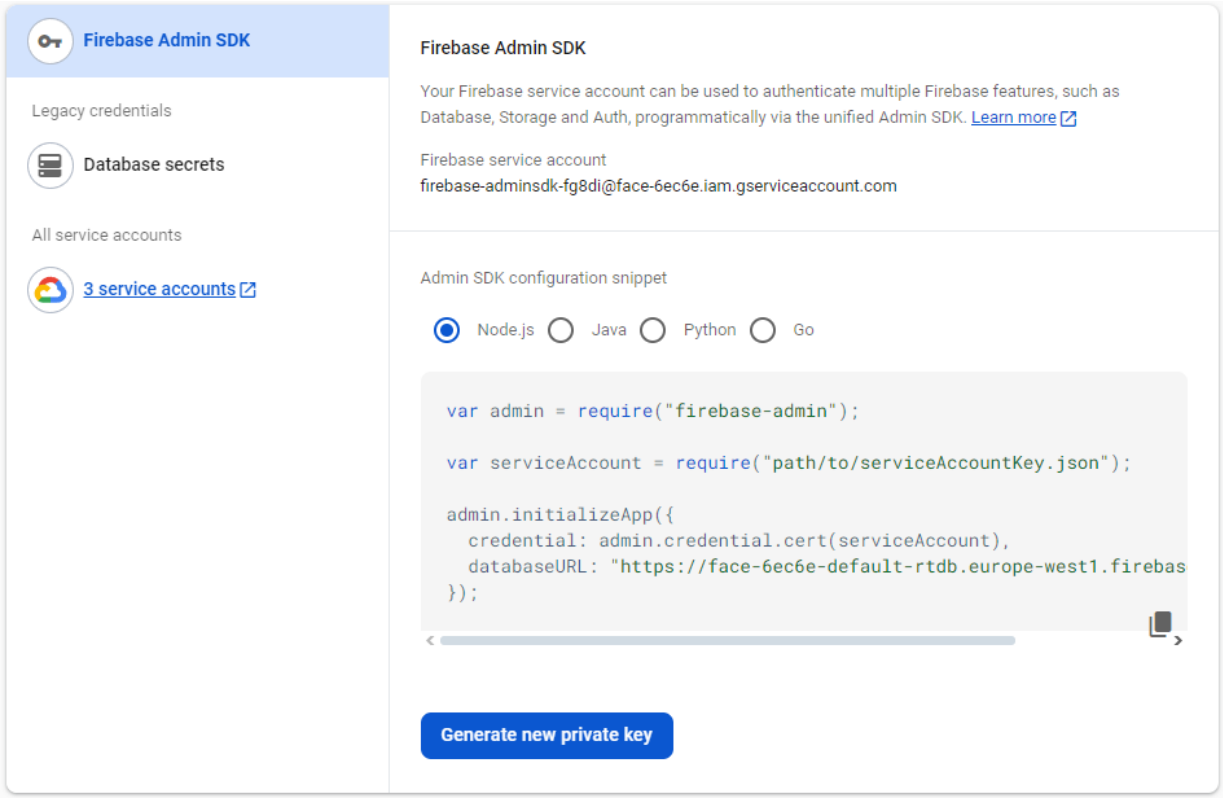

Рисунок 3.2 – Сторінка налаштування сервісного аккаунту

Особливу увагу було приділено забезпеченню безпеки. Після завершення основного етапу розробки були встановлені суворі правила доступу до бази даних та зберігання файлів. Наприклад, для доступу до даних тепер вимагається автентифікація користувачів, що значно підвищує рівень безпеки системи.

Таким чином, розгортання хмарної інфраструктури за допомогою Firebase стало важливим етапом роботи, який забезпечує зручне зберігання даних, їх

обробку та надійний обмін інформацією між клієнтом і сервером. Це рішення гарантує не лише ефективність роботи системи, але й дозволяє її масштабувати у майбутньому.

3.2 Конфігурація мережі та серверів

Після успішного реєстрації та розгортання, наступним кроком була конфігурація мережі та серверів. Цей процес включав в себе налаштування мережевих параметрів, доступу до серверів та забезпечення безпеки.

Налаштування IP-адрес та доступу: Спочатку я визначив, яким чином мої сервери будуть отримувати IP-адреси. Я використовував статичні або динамічні IP-адреси в залежності від вимог проекту. Крім того, я налаштував правила файрволу мережі та обмеження доступу до серверів.

Налаштував резервне копіювання бази даних. Це дозволяє регулярно створювати резервні копії даних та забезпечувати можливість відновлення системи в разі втрати або пошкодження даних.

Для забезпечення роботи програмного продукту було створено конфігураційний файл у форматі YAML, який містить ключові параметри для налаштування зв’язку з Firebase. Цей файл дозволяє зберігати необхідні дані для авторизації та взаємодії з базою даних і хмарним сховищем у зручному та безпечному вигляді (рис. 3.3).

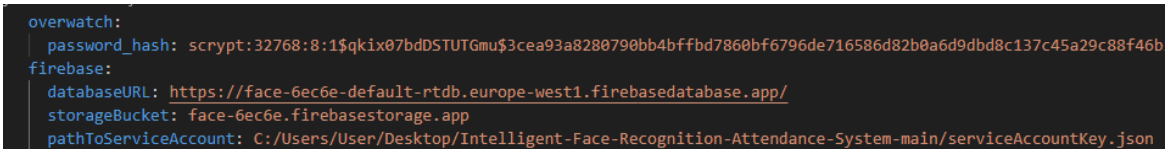

Рисунок 3.3 – Код з налаштуванням

У файлі конфігурації передбачено блок для збереження хешованого пароля. Цей пароль використовується для авторизації адміністративного доступу в системі. Його зберігання у вигляді хешу, створеного за допомогою

криптографічних алгоритмів, гарантує безпеку, адже навіть у разі доступу до файлу конфігурації зловмисник не зможе відновити пароль у відкритому вигляді.

Інша частина конфігурації призначена для роботи з Firebase. Тут зазначається посилання на базу даних Realtime Database, яке є основним джерелом даних для системи. Також вказується ідентифікатор хмарного сховища Firebase, що дозволяє зберігати файли, необхідні для роботи програми. Для забезпечення доступу до Firebase використовується JSON-файл із ключем сервісного акаунта. У файлі YAML міститься шлях до цього JSON-файлу, який система завантажує під час ініціалізації.

Під час запуску програмного продукту бібліотека для роботи з YAML зчитує конфігураційний файл, а отримані дані використовуються для налаштування підключення до Firebase. Це забезпечує автоматизацію процесу інтеграції, зменшує ризик помилок і спрощує процес налаштування.

Для реалізації підключення застосовується бібліотека Firebase Admin SDK. Після завантаження конфігураційного файлу система ініціалізує Firebase, використовуючи ключ сервісного акаунта та параметри бази даних і хмарного сховища. Наприклад, у середовищі Python цей процес включає зчитування файлу YAML, завантаження ключа та виклик методів для ініціалізації з’єднання з Firebase.

Щоб уникнути витоку конфіденційних даних, конфігураційний файл і ключ сервісного акаунта зберігаються локально і виключені з системи контролю версій. Окрім цього, у Firebase налаштовано політики безпеки, які обмежують доступ до ресурсів тільки для авторизованих користувачів [21].

Використання конфігураційного файлу надає можливість швидко змінювати налаштування підключення без внесення змін до коду програми, що сприяє зручності й адаптивності системи. Такий підхід не лише спрощує процес розгортання, але й забезпечує надійність і безпеку роботи програмного продукту.

3.3 Проектування бази даних

База даних матиме дві основні сутності: користувач і файл. Кожна сутність повинна мати унікальний ідентифікатор. Користувач буде авторизуватись через e-mail та пароль. Нам необхідно зберігати інформацію про розмір диска та кількість заповненого місця в байтах. Також не забуваємо про аватарку користувача. Колекція USERS містить інформацію про всіх користувачів, зокрема працівників і адміністраторів. Кожен користувач має унікальний ідентифікатор (userID), ім’я (name), електронну адресу (email) і пароль (password), що зберігається у зашифрованому вигляді для забезпечення безпеки. Поле userType визначає тип користувача (працівник чи адміністратор), а classes містить список курсів, до яких прив’язаний користувач, та кількість відвідувань. Окремо зберігаються embeddings – цифрові відбитки обличчя, необхідні для біометричної ідентифікації.

Тепер створимо другу сутність із інформацією про файл. Додамо обов'язковий параметр айді, ім'я та тип файлу. У типі ми будемо вказувати розширення файлу, або те, що файл є папкою. Також вкажемо розмір файлу та зовнішній ключ користувача (рис. 3.4).

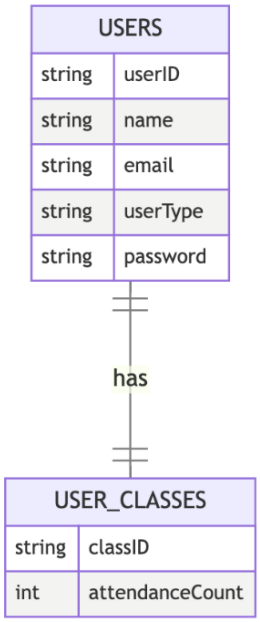

Рисунок 3.4 – Макет User

Колекція USER_CLASSES зберігає інформацію про класи, зокрема унікальний ідентифікатор класу (classID) та кількість відвідувань (attendanceCount). Ця колекція пов’язана з користувачами через відповідні класи.

Для зберігання зображень працівників використовується Firebase Storage, де файли зображень мають назви, що відповідають ідентифікаторам користувачів. Такий підхід дозволяє швидко інтегрувати медіафайли в систему.

Схема розроблена з акцентом на безпеку, гнучкість і масштабованість. Використання Firebase забезпечує швидкий доступ до даних, синхронізацію в реальному часі та ефективне управління зв’язками між об’єктами. Вона також відкриває можливості для впровадження штучного інтелекту, наприклад, для розпізнавання облич, що робить її придатною для подальшого розширення функціональності системи.

3.4 Розробка дизайну

Розроблено макети сторінок програмних модулів, які забезпечують зручний і зрозумілий процес взаємодії адміністраторів із системою. Наведені макети охоплюють ключові функціональні можливості, такі як реєстрація нових адміністраторів та перегляд інформації про їхню активність. Їх ціль полягає у створенні інтуїтивного інтерфейсу для роботи з базою даних і покращення взаємодії адміністраторів із системою.

Макет сторінки введення даних адміністратора (рис. 3.5) розроблено для реєстрації нового адміністратора в системі.

- заголовок сторінки, формулювання "Please Enter Your Informations" чітко визначає мету сторінки та орієнтує користувача на виконання завдання;
- форма введення даних, включає ключові поля, такі як ім'я адміністратора, електронна адреса, тип користувача (наприклад, "адміністратор") і пароль. Поле для вибору відділів чи напрямків дозволяє швидко вказати сферу відповідальності адміністратора;

- кнопка підтвердження ("Submit"), Використовується для завершення процесу реєстрації. Після її натискання дані адміністратора перевіряються та зберігаються в базі даних;
- ергономіка, елементи форми розташовані послідовно, що робить процес введення зручним і зрозумілим навіть для нових адміністраторів.

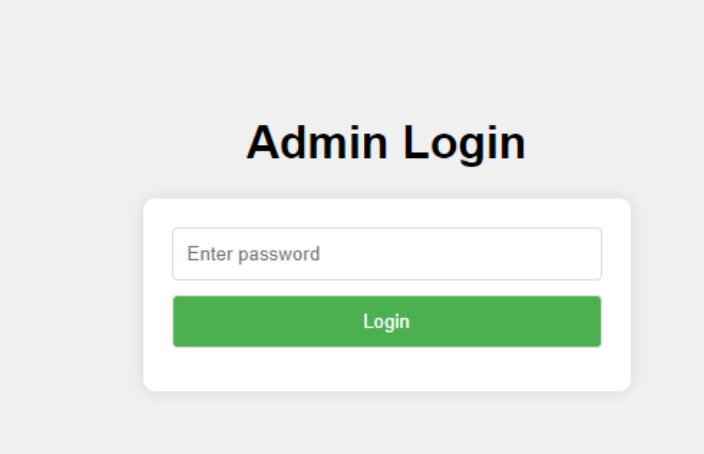

Рисунок 3.5 – Макет Admin login

Макет сторінки перегляду інформації про активність (рис. 3.6) забезпечує швидкий доступ до даних про робочих і їхню участь у проєктах чи виконаних завданнях.

- заголовок сторінки, формулювання "Attendance" (або "Активність") чітко визначає, що саме відображається на сторінці;
- заблиця даних, таблиця включає стовпці з основною інформацією про адміністраторів, такі як ім'я, електронна адреса, тип користувача ("адміністратор"), і деталі про їхню участь у проєктах або завданнях;
- організація даних, таблиця розроблена з урахуванням зручності читання та ефективного перегляду великого обсягу даних. Інформація розділена по рядках, що спрощує аналіз.

Макети створені з урахуванням вимог до зручності користування та можливостей масштабування. Вони демонструють логічну та послідовну організацію елементів, що полегшує виконання завдань у системі.

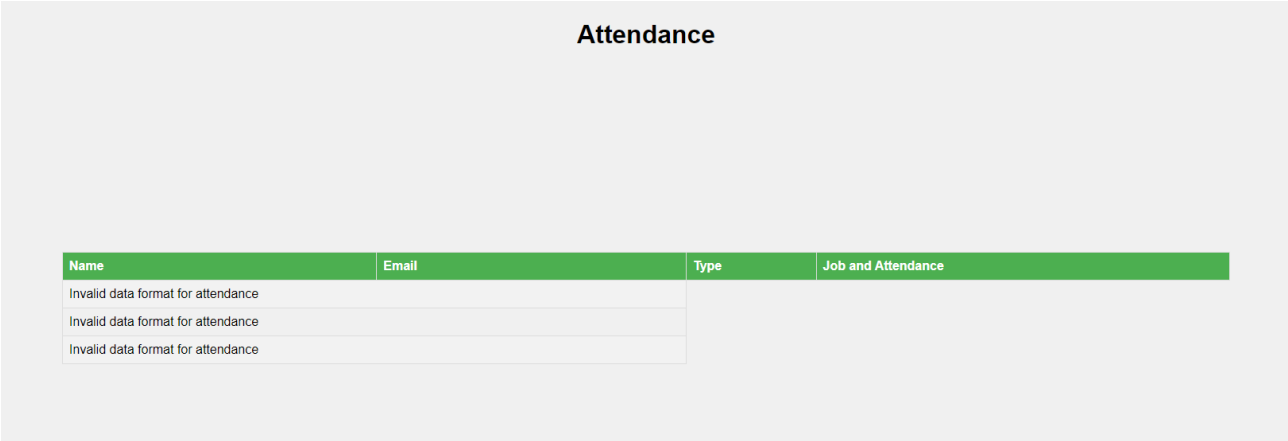

Рисунок 3.6 – Макет Attendance

Реалізував інтерфейс вибору посади, який представлений у вигляді випадаючого списку. Ця функція дозволяє користувачу обрати необхідну категорію або відділ із запропонованого списку. Заголовок форми — "Select job position", а вибір поточної категорії, наприклад "Information Technology", автоматично відображається в списку. Цей елемент сприяє зручному фільтруванню або перегляду даних, пов’язаних із конкретними посадами чи підрозділами, та є важливою частиною функціоналу системи (рис. 3.7).

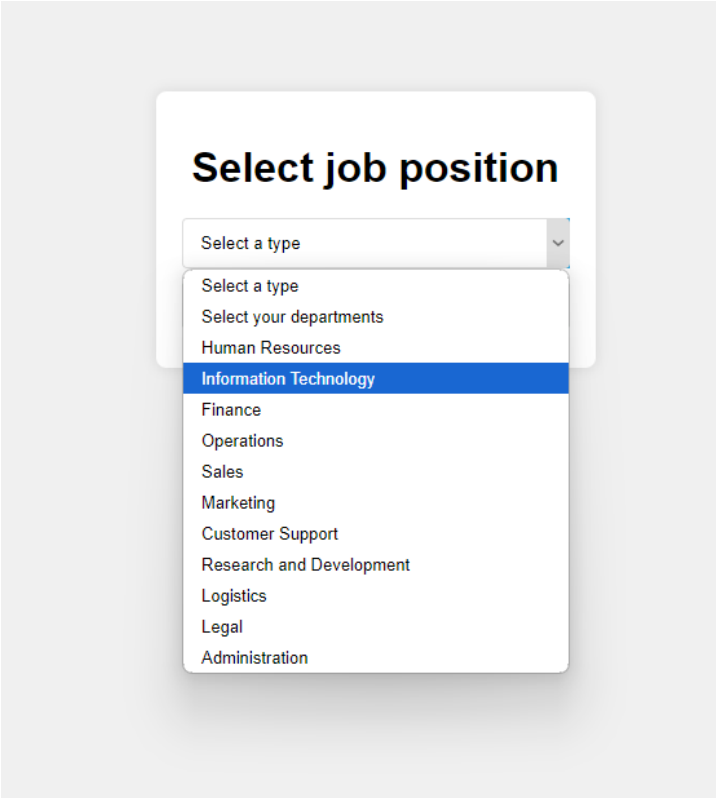

Рисунок 3.7 – Макет Select job position

На цій сторінці (рис. 3.8) я створив макет форми для введення інформації про користувачів у межах дипломної роботи. Форма складається з кількох обов’язкових полів:

- ім’я та прізвище, тут потрібно ввести повне ім’я працівника або адміністратора, яке ідентифікуватиме його в системі;
- електронна пошта, сюди вводиться корпоративна або персональна адреса, яка використовується для зв’язку або входу до системи;
- роль у системі, у випадаючому списку можна обрати роль, яку виконує користувач, наприклад, працівник чи адміністратор;
- підрозділ, у цьому полі необхідно вказати відділ, до якого належить користувач. Список включає різні опції, такі як "Інформаційні технології", "Фінанси", "Логістика" тощо;
- пароль, у полі для пароля користувач вводить унікальний набір символів, який забезпечує безпеку доступу до його облікового запису.

Після заповнення всіх полів користувач натискає кнопку "Submit", що надсилає введені дані до системи. Ця форма розроблена для швидкого й точного збору інформації про працівників та адміністраторів, забезпечуючи її подальше використання в системі управління.

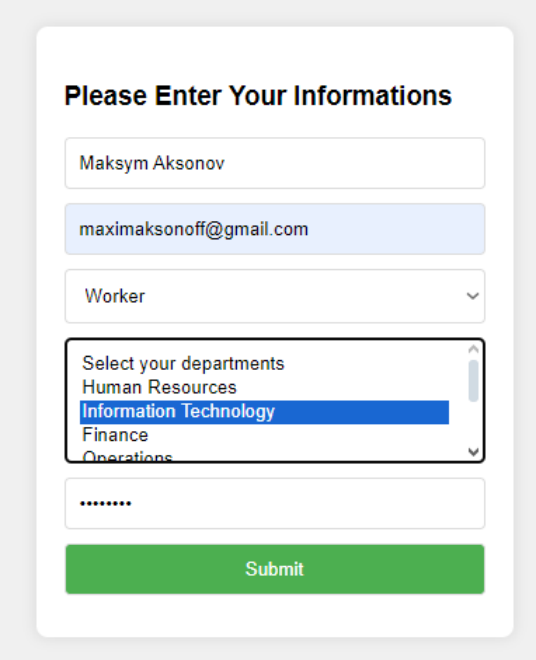

Рисунок 3.8 – Макет сторінки створення користувача

Сторінка обліку робочого часу повинна забезпечувати зручний та інтуїтивно зрозумілий спосіб управління інформацією про робочі години, що дозволяє користувачам ефективно відслідковувати та керувати своїм робочим часом (рис. 3.9).

Рисунок 3.9 – Макет сторінки авторизації користувача

3.5 Проектування графічного інтерфейсу користувача

Веб-сайт системи розпізнавання обличчя для обліку робочого часу має зручний інтерфейс, який дозволяє користувачам ефективно працювати з основними функціями. На сторінці є логотип і заголовок "Welcome to our Face Detection System for Workplace Attendance", що завжди залишається видимим при прокручуванні. Користувач може завантажити нове зображення обличчя, натиснувши кнопку для завантаження фотографії. Це зображення буде використано для розпізнавання облич. Є також функція для реєстрації присутності, яка після обробки зображення або відео підтверджує присутність користувача та перенаправляє на вибір класу або групи.

Крім того, є можливість для адміністраторів увійти в систему для налаштування доступу та управління її функціями. Інтерфейс також включає функцію захоплення обличчя за допомогою камери, що дозволяє в реальному часі спостерігати за процесом розпізнавання. Після успішного захоплення обличчя користувач буде перенаправлений для введення додаткової інформації.

Весь інтерфейс забезпечує простоту у використанні та зручність для користувачів, дозволяючи їм швидко виконувати необхідні дії (рис. 3.10).

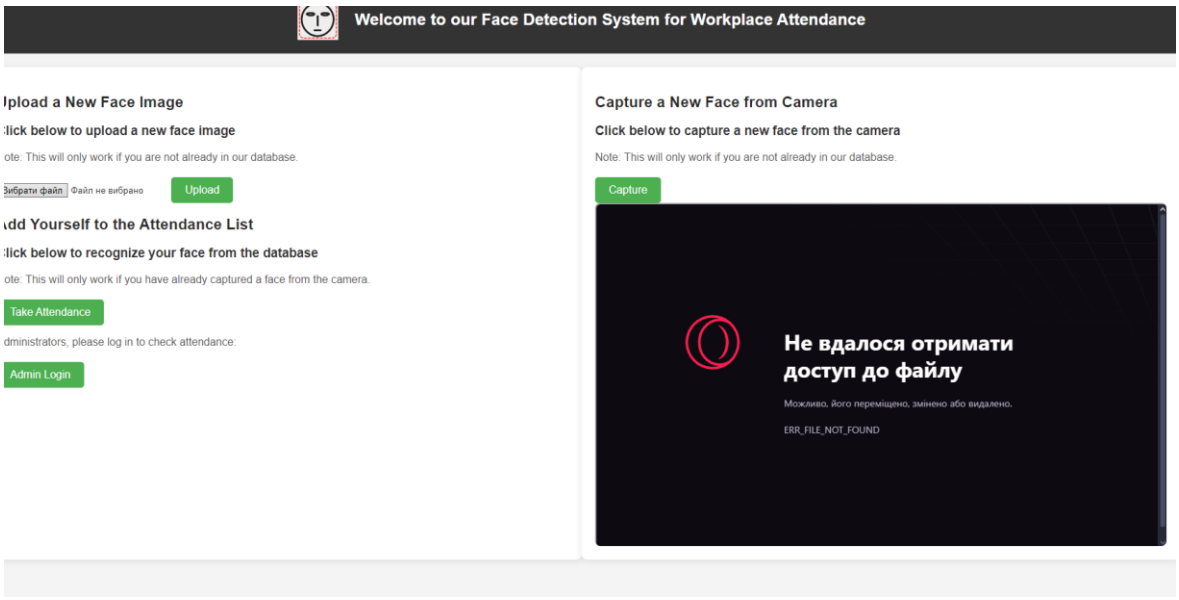

Рисунок 3.10 – Макет сторінки Home

В результаті розробки програмних модулів кінцевий продукт відповідає розробленим макетам (рис 3.5 – 3.10).

3.6 Розробка серверної частини

Функція upload_database(filename) виконує завдання перевірки і завантаження файлу в базу даних Firebase. По-перше, вона перевіряє, чи існує файл з таким іменем у Firebase Storage. Якщо існує, повертає помилку з відповідним повідомленням у HTML-форматі. По-друге, перевіряє, чи ім'я файлу, за винятком розширення, є числом, що гарантує структуру іменування файлів (ймовірно, прив'язану до ID працівників). Якщо ім'я файлу не відповідає умовам, повертає помилку. Якщо ж файл валідний, функція генерує шлях до файлу на сервері додатка, ініціалізує об'єкт blob в Firebase Storage, завантажує файл і очищає повідомлення про помилку. Повертає булевий статус валідності та можливу помилку.

Функція match_with_database(img, database) займається виявленням облич на переданому зображенні та їх зіставленням з базою даних. Спершу викликається detect_faces(img), яка повертає координати виявлених облич. Потім по черзі малюються червоні прямокутники навколо кожного обличчя на копії зображення, яке зберігається локально. Для кожного обличчя викликається align_face(img, face) для вирівнювання, потім extract_features(aligned_face) для отримання числового представлення (вектору ознак). Вектор ознак передається у match_face(embedding, database), яка шукає відповідність у базі. Якщо знайдено, повертає повідомлення про збіг; якщо ні, повідомляє про відсутність збігу. У разі виключення (наприклад, через відсутність обличчя), повертає повідомлення про невдачу.

Функція home() використовується для обробки запиту до кореневого маршруту вебдодатку ("/"). Вона рендерить HTML-шаблон home.html, що є основною сторінкою програми. Вона не містить додаткової логіки, тому її основне завдання — забезпечити користувачеві доступ до домашньої сторінки.

Функція add_info() аналогічно обробляє запити до маршруту "/add_info". Вона повертає шаблон add_info.html, який дозволяє адміністратору або користувачу додати нову інформацію до системи. Її логіка в коді відсутня, бо сама функціональність зосереджена на взаємодії з фронтендом.

Функція admin_login() забезпечує можливість входу адміністратора за допомогою пароля. Якщо запит методом "POST", тобто користувач ввів пароль, цей пароль звіряється з хешем, збереженим у змінній ADMIN_PASSWORD_HASH, за допомогою check_password_hash(). Якщо паролі співпадають, користувача перенаправляють на сторінку відвідуваності, інакше — показується повідомлення про невірний пароль через flash(). У всіх інших випадках (GET-запит) повертається шаблон для введення пароля.

Функція attendance() завантажує дані про працівників з Firebase Realtime Database через db.reference("Workers"). Дані витягуються у форматі словника, де кожен ключ відповідає ID працівника. Далі інформація про працівників,

включаючи їх ім'я, email, тип користувача та класи, додається до словника workers, який передається до шаблону attendance.html для рендерингу.

Функція upload() відповідає за завантаження файлів через форму. Перевіряє, чи файл був переданий, чи його розширення допустиме (перевіряється через allowed_file). Ім'я файлу генерується як ID працівника, отриманий з бази даних. Потім файл зберігається на сервері, а після цього завантажується до Firebase через upload_database. Якщо все успішно, користувач перенаправляється на сторінку додавання інформації.

Функція allowed_file(filename) перевіряє, чи файл має одне з дозволених розширень (png, jpg, jpeg, gif). Вона використовує метод rsplit для виділення розширення, після чого перевіряє його належність до множини дозволених.

3.7 Опис функціональності системи розпізнавання обличчя

Розглянемо ключові етапи та функції, що реалізують систему розпізнавання обличчя на основі Python. Використовуються такі бібліотеки, як OpenCV, dlib, DeepFace та SciPy для досягнення високої точності та ефективності в обробці зображень та розпізнаванні облич.

На першому етапі імпортуються необхідні бібліотеки, такі як cv2 для обробки зображень, dlib для виявлення облич і визначення їхніх ключових точок, а також DeepFace для створення ембедінгів (числових представлень) облич. Окрім того, вказуються шляхи до необхідних файлів, таких як модель для прогнозування ключових точок обличчя та класифікатор Хаара для виявлення облич на зображеннях.

Функція detect_faces призначена для виявлення облич на вхідному зображенні. Спочатку зображення перетворюється у відтінки сірого, що полегшує виявлення контурів облич. Потім за допомогою каскадного класифікатора, базованого на алгоритмі Хаара, виявляються обличчя, і повертається список координат знайдених облич.

Функція align_face здійснює вирівнювання обличчя. Після того як обличчя було виявлено, на зображенні виділяються ключові точки (очі, рот тощо), що дозволяє вирівняти обличчя так, щоб очі та рот перебували в зафіксованих позиціях. Це дозволяє усунути вплив повороту чи нахилу обличчя на точність розпізнавання. Крім того, зображення масштабується до розміру 256x256 пікселів, що є стандартним для подальшої обробки.

Функція extract_features відповідає за витягнення характеристик обличчя у вигляді ембедінгу. Після вирівнювання обличчя зображення конвертується з формату BGR у RGB, що є стандартом для роботи з DeepFace. За допомогою цієї моделі обличчя перетворюється в числове представлення, яке містить характеристики, що дозволяють відрізняти одне обличчя від іншого.

Функція match_face порівнює витягнуті характеристики обличчя з ембедінгами, що зберігаються в базі даних. Для порівняння використовується косинусна відстань, яка дозволяє визначити схожість між двома ембедінгами. Якщо відстань між ембедінгами менша за певний поріг (0.50), система вважає, що обличчя співпало, і повертає ім'я співпадаючої особи з бази даних. Якщо ж відстань більша, то повертається None, що означає відсутність співпадіння.

Основний процес роботи системи розпізнавання обличчя складається з кількох етапів. Спочатку, на першому етапі, система здійснює виявлення обличчя на зображенні. Це відбувається за допомогою методів, які дозволяють ефективно знаходити обличчя в реальному часі. Після цього, для коректної роботи, здійснюється вирівнювання обличчя, щоб компенсувати можливі нахили або повороти, які можуть впливати на точність розпізнавання. Третій етап полягає у витягуванні характеристик обличчя, для чого використовуються моделі, такі як DeepFace. Ця модель генерує числові представлення обличчя, які називаються ембедінгами. Отримані характеристики порівнюються з тими, що вже зберігаються в базі даних, для подальшої ідентифікації.

Технічні деталі реалізації включають кілька важливих компонентів. Для виявлення обличчя застосовується класичний метод каскадних класифікаторів Хаара, який добре зарекомендував себе у реальному часі. Для точного

визначення ключових точок на обличчі використовується бібліотека dlib. Вона дозволяє здійснювати точне вирівнювання обличчя, що важливо для збереження коректності результатів. Для створення ембедінгів застосовується модель DeepFace, яка генерує числові характеристики обличчя. Для порівняння облич використовується метод косинусної відстані, що дозволяє точно оцінити схожість між отриманими ембедінгами.

Особливістю реалізації є автоматичне вирівнювання обличчя, що дозволяє системі досягати високої точності, незалежно від того, під яким кутом було зроблене фото або відео. Алгоритм також оптимізований для роботи в умовах змінного освітлення та різних кутів зйомки, що робить його більш універсальним і стійким до змінних умов. Важливим аспектом є встановлення порогового значення для визначення, чи є співпадіння між обличчями, що дає змогу контролювати точність розпізнавання. Система також може працювати з базою даних відомих облич, що дозволяє здійснювати їх ідентифікацію в реальному часі, забезпечуючи точність і швидкість розпізнавання.

3.8 Опис коду для роботи з базою даних

Розглянемо ключові етапи та функції для інтеграції системи розпізнавання обличчя з базою даних і взаємодії з Firebase, щоб зберігати та обробляти дані в робочих середовищах. Спершу імпортуються необхідні бібліотеки для роботи з OpenCV, Firebase, а також для порівняння обличчя. Налаштовуються шляхи до файлів та підключення до бібліотек для розпізнавання облич, а також до Firebase для доступу до бази даних і сховища.

Основна функція match_with_database призначена для порівняння обличчя, захопленого з камери або завантаженого з файлу, з обличчями в базі даних. Процес порівняння складається з кількох етапів: спочатку на зображенні виявляються обличчя, потім для кожного обличчя виконується вирівнювання та витягнення характеристик, після чого ці характеристики порівнюються з

наявними в базі даних. Результати виводяться на зображеннях, з підписами до кожного обличчя.

Налаштування Firebase включає завантаження сертифікатів для аутентифікації, що дозволяє безпечно підключатися до сервісів Firebase. Також налаштовується база даних для зберігання характеристик облич та сховище для їх зберігання. Після ініціалізації з’єднання з Firebase, система може використовувати URL бази даних для взаємодії з даними.

Робота з базою даних передбачає підключення до Firebase, де зберігаються характеристики облич робочих та адміністраторів. Для кожного робочого або адміністратора отримуються його ембедінги — числові характеристики обличчя, що дозволяють зберігати та порівнювати ці обличчя з новими зображеннями. Локальна база даних формується для кожного робочого для подальшого порівняння з новими даними.

Інтерфейс користувача дозволяє вибирати між двома режимами: роботою з камерою або зображенням. В залежності від вибору, система захоплює відео з камери або завантажує зображення для обробки. У режимі роботи з камерою є можливість зберігати кадри, а в режимі зображення — обробляти зображення, завантажене з файлу.

Система інтегрована з Firebase, що дозволяє зберігати дані робочих та адміністраторів і їхні характеристики в реальному часі, що забезпечує точність порівняння. Живий перегляд з камери надає можливість зберігати кадри, а також відображає візуальні підказки для користувача, що робить інтерфейс більш зручним. Крім того, система підтримує різні джерела вводу: камера або файл, а результати розпізнавання візуалізуються за допомогою прямокутників навколо облич. Для забезпечення надійності, система передбачає обробку помилок при розпізнаванні та перевірку наявності обличчя на зображенні. Безпечне з'єднання з Firebase гарантує стабільність і безпеку роботи з даними.

Технічний процес роботи системи включає ініціалізацію з'єднання з Firebase, завантаження даних робочих та адміністраторів, вибір режиму роботи (камери або зображення), захоплення або завантаження зображення для обробки,

розпізнавання облич, витягнення характеристик та порівняння їх з базою даних, а також візуалізацію результатів, включаючи підписи з іменами знайдених осіб.

Можливі покращення системи включають додавання підтримки множинних облич на зображеннях для покращення функціональності, вдосконалення інтерфейсу користувача для зручнішого використання, додавання системи логування для фіксації результатів роботи, а також оптимізацію процесів взаємодії з базою даних для швидшої обробки великих обсягів даних.

Ця система є інтегрованим рішенням для автоматизації таких процесів, як облік робочого часу, що використовує технології розпізнавання облич та базу даних для ефективної роботи в реальному часі в корпоративних та професійних середовищах.

РОЗДІЛ 4
ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ СИСТЕМИ ОБЛІКУ

4.1 План тестування

План тестування є важливою частиною розробки програмного забезпечення, оскільки він визначає, як буде перевірятися правильність роботи системи, надійність її функцій, ефективність взаємодії компонентів та виявлення можливих помилок. У даному випадку, тестування зосереджене на компоненті системи розпізнавання облич, яка включає в себе детекцію облич, порівняння з базою даних, збереження даних у Firebase, а також інтеграцію з веб-камерою чи зображенням.

Під час перевірки роботи системи розпізнавання облич необхідно впевнитися, що вона виконує всі заявлені функції та здатна стабільно працювати у різних реальних умовах. Для цього розроблено план тестування, який охоплює всі ключові аспекти роботи системи.

На першому етапі потрібно перевірити базову функціональність розпізнавання облич. Підключивши веб-камеру, слід упевнитися, що система коректно ідентифікує обличчя, обрамляючи його рамкою. Важливо, щоб інформація про кількість облич у кадрі та швидкість обробки (FPS) відображалася в реальному часі без затримок.

Далі необхідно протестувати систему на здатність працювати з кількома обличчями одночасно. У кадрі розміщуються кілька осіб, і перевіряється, чи система правильно розпізнає кожне обличчя, рахує їхню кількість та коректно обробляє відповідні дані.

Для оцінки стійкості системи слід змінити зовнішні умови. Зокрема, перевіряється робота алгоритму за різного рівня освітлення: від яскравого світла до напівтемряви або за умов швидкої зміни освітлення. Система повинна

зберігати свою функціональність і точність. У випадку відсутності облич у кадрі потрібно перевірити, чи правильно вона фіксує 0 об'єктів.

Особливу увагу варто приділити тестуванню навантаження. У кадрі можуть з’являтися об’єкти, які потенційно здатні ввести систему в оману: фотографії, відео з екрана або манекени. Завдання системи – уникати хибних спрацьовувань і розпізнавати лише реальні обличчя.

Крім того, інтерфейс повинен забезпечувати зручність користувачів. Необхідно перевірити, чи коректно відображаються дані про кадр, кількість розпізнаних облич та швидкість обробки. Інформація має оновлюватися в режимі реального часу без затримок.

Під час тестування продуктивності потрібно оцінити швидкість обробки кадрів за різних умов – від одного до кількох облич у кадрі або за їхньої відсутності. Це дозволить визначити, чи відповідає система заявленим вимогам.

На завершення результати кожного тесту документуються, зазначаючи як успішні сценарії, так і виявлені помилки. На основі цих даних робляться висновки про функціональність, точність і стійкість системи. Такий підхід дозволяє виявити потенційні недоліки та забезпечити їхнє усунення перед впровадженням системи у практичне використання.

4.2 Валідація роботи системи

Валідація роботи системи є критичним етапом у процесі розробки програмного забезпечення, оскільки вона дозволяє перевірити, чи відповідає система вимогам і чи правильно вона виконує свою функціональність в умовах реального використання. У випадку із системою розпізнавання обличчя для контролю відвідуваності в навчальних закладах, валідація є особливо важливою, оскільки система має працювати в умовах різноманітних змінних факторів, таких як різне освітлення, кути нахилу обличчя, наявність декількох облич на

зображенні тощо. Тому валідація повинна охоплювати кілька ключових аспектів: функціональність, точність, продуктивність, безпеку та зручність використання.

Валідація роботи системи входу для адміністратора включає такі етапи (рис. 4.1):

- перевірка, чи користувач вводить пароль у поле, позначене як обов'язкове (required);
- перевірка процесу авторизації: при правильному паролі має здійснюватися вхід, при неправильному — виводитися повідомлення про помилку;
- перевірка коректності відображення повідомлень про помилки, використовуючи шаблон для виведення тексту.;
- оцінка дизайну форми для зручності користувача, включаючи розмір і доступність полів;
- перевірка роботи кнопки "Login", яка відправляє запит на сервер;
- тестування обробки помилок на сервері для коректної відповіді при невірному паролі.

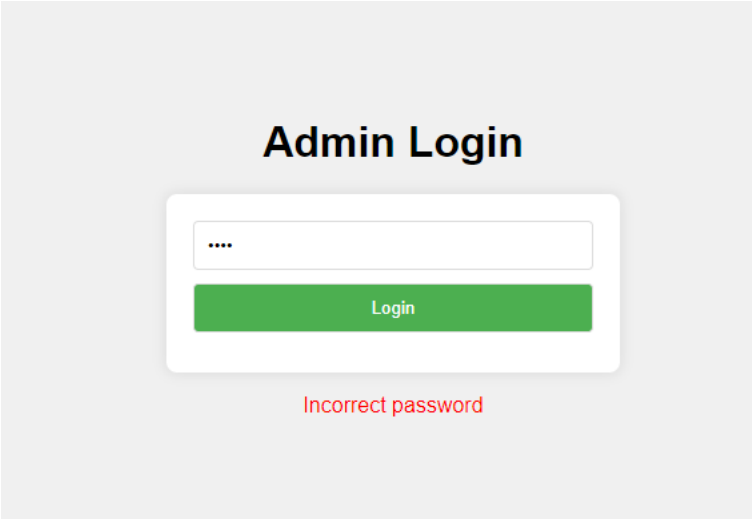

Рисунок 4.1 – Сторінка входу

Валідація системи вибору класу включає наступні етапи (рис. 4.2):

- перевірка обов'язковості вибору класу з випадаючого списку (required);

- перевірка, чи користувач може вибрати один з варіантів класів, доступних у списку;
- оцінка правильної роботи кнопки "Submit", яка надсилає вибір на сервер;
- перевірка відображення правильно обраного класу на сервері після підтвердження;
- тестування на правильність обробки порожнього вибору (коли користувач не вибрав жодного класу);
- перевірка дизайну форми для зручності користувача та доступності елементів.

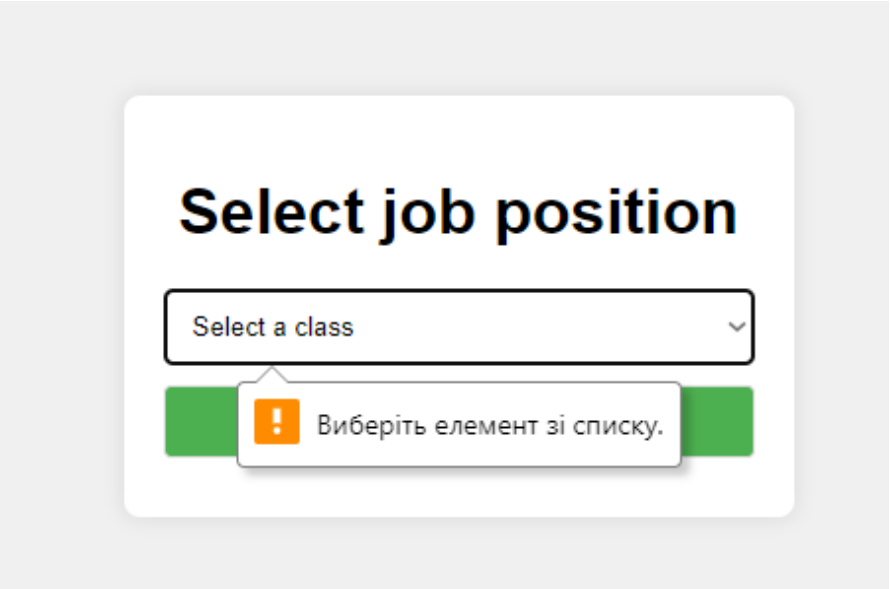

Рисунок 4.2 – Сторінка вибору

Валідація системи додавання інформації включає наступні етапи (рис. 4.3):

- перевірка обов'язковості введення імені, електронної пошти та пароля для користувача;
- перевірка обов'язковості вибору типу користувача з випадаючого списку ("Офісний працівник" або "Адміністратор");
- перевірка правильності вибору з переліку доступних робочих груп або проектів (наприклад, для "Розробки програмного забезпечення", "Аналізу даних" тощо);

- перевірка можливості введення кількох груп або проектів (для багатократного вибору);
- перевірка на коректність відображення введеної інформації на сервері після підтвердження;
- тестування на відсутність заповнених полів (порушення вимог);
- перевірка дизайну та зручності форми для швидкого заповнення та навігації користувачем.

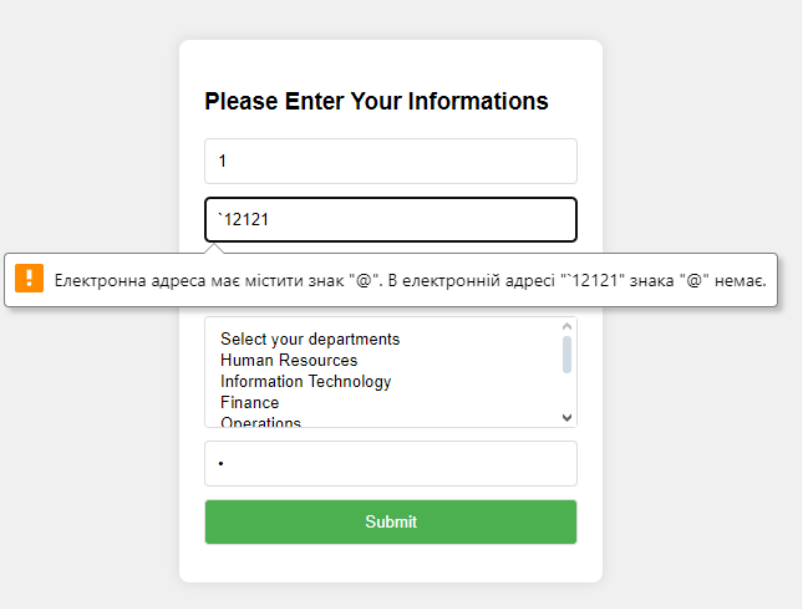

Рисунок 4.3 – Сторінка реєстрації

У процесі роботи з веб-застосунками важливу роль відіграють статусні коди відповіді сервера. Вони дають зрозуміти, як сервер обробив запит користувача і чи була операція успішною. Один з основних інструментів для діагностики роботи веб-сервісу — це статусні коди HTTP, які передають інформацію про стан запиту і результати виконання.

Два з найбільш часто використовуваних кодів відповіді, які часто з'являються в консолі при роботі з веб-додатками, — це код **200** та код **302**. Вони мають важливе значення для розуміння роботи сервера і коректності взаємодії з користувачем. (рис. 4.4)

- Код відповіді 200 вказує на успішне виконання запиту та правильну обробку інформації.

- Код відповіді 302 вказує на тимчасове перенаправлення користувача на іншу сторінку (наприклад, після успішної авторизації або відправлення форми).

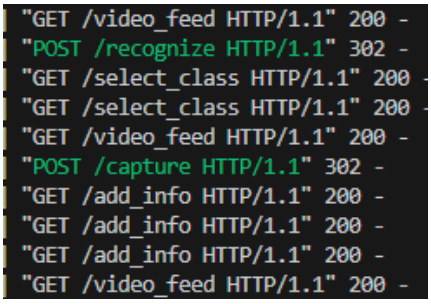

Рисунок 4.4 – Консоль програми

Зображення було успішно завантажено до бази даних. Тепер воно доступне для подальшого використання, і я можу звертатися до нього через базу даних за необхідності.

Повідомлення: "image upload successfully to the database"

Це повідомлення означає, що зображення було успішно отримано, оброблено та збережено в базі даних без жодних проблем.

У бекенді процес зазвичай включає отримання зображення, перетворення його в відповідний формат і збереження його в базі даних (рис. 4.5). Також можна зберігати шлях або URL зображення в базі даних для подальшого зручного доступу.

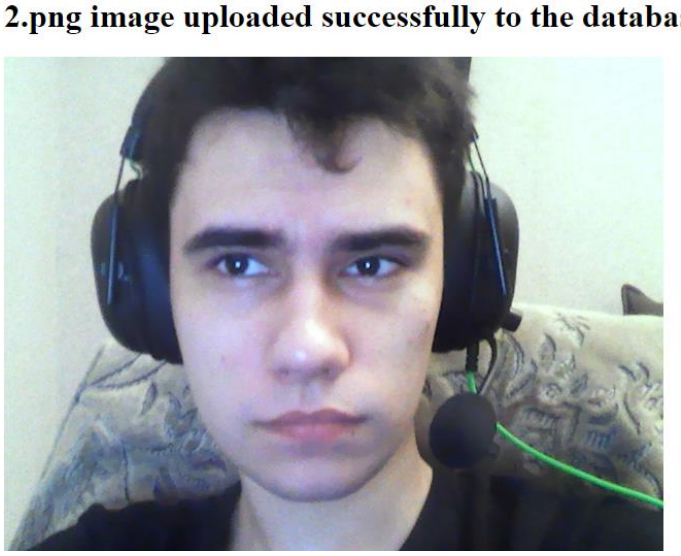

Рисунок 4.5 – Завантаження фото

Тестування відображення таблиці з даними про відвідуваність працівників перевіряє правильність форматування та коректність відображення даних. Таблиця повинна містити такі колонки: ім'я, email, посада, департамент, час входу, час виходу, кількість відпрацьованих годин.

Дані працівників мають коректно відображатися в таблиці, включаючи правильне чергування фону рядків та зміни кольору при наведенні на активний елемент. Перевіряється також динамічне заповнення даних з бази, щоб забезпечити актуальність інформації (рис. 4.6).

Рисунок 4.6 – Сторінка відвідування

4.3 Тестування функціональності та продуктивності

Тестування функціональності та продуктивності є важливими етапами валідації програмної системи, оскільки вони дозволяють не тільки перевірити, чи працює система відповідно до вимог, а й оцінити, наскільки ефективно вона виконує свої завдання під різними умовами навантаження. У випадку із системою розпізнавання обличчя для контролю доступу до робочих приміщень, ці етапи тестування допомагають виявити проблеми, пов'язані з коректністю роботи, швидкодією, а також здатністю системи обробляти різноманітні сценарії використання.

Тестування функціональності зосереджене на перевірці, чи всі заявлені функції системи виконуються правильно та чи система відповідає всім вимогам користувача. Для цього слід ретельно перевірити кожну з функцій:

Детекція обличчя. Перше, що потрібно перевірити — це чи система здатна правильно виявляти обличчя на зображеннях і у відео. Тестування повинно охоплювати різні умови освітлення (погане освітлення, яскраве світло, зворотне освітлення), а також різні кути нахилу голови та розміри облич. Також важливо протестувати здатність системи виявляти кілька облич одночасно.

Вирівнювання обличчя. Після виявлення обличчя система повинна коректно вирівняти його для подальшого розпізнавання. Важливо протестувати, чи система правильно обробляє нахилені обличчя, чи вона здатна виправляти перспективні спотворення (наприклад, при зйомці знизу або зверху).

Екстракція ознак. Перевірка точності екстракції ознак є критично важливою, оскільки саме ці ознаки використовуються для порівняння облич з базою даних. Для тестування цієї функціональності необхідно використовувати різні зображення однієї й тієї ж особи, щоб переконатися, що система правильно витягує унікальні ознаки, навіть якщо особа змінює вираз обличчя або знаходиться в іншому освітленні.

Порівняння з базою даних. Тестування порівняння повинно включати перевірку, чи система правильно знаходить співпадіння облич із зареєстрованими працівниками в базі даних. Важливо протестувати як позитивні випадки (коли система має знайти співпадіння), так і негативні (коли система не повинна знайти співпадіння).

Запис відвідуваності в базу даних. Перевірка коректності запису даних про присутність в Firebase є важливою частиною тестування. Необхідно перевірити, чи система коректно фіксує час присутності працівників, чи актуалізуються дані в базі, а також чи правильно відображаються результати при запитах на веб-інтерфейсі.

Інтерфейс користувача. Тестування інтерфейсу користувача полягає в перевірці зручності та інтуїтивності використання. Важливо протестувати, чи всі

кнопки і елементи управління працюють коректно, чи доступні всі необхідні функції.

Система розпізнавання облич реалізована на основі глибокого навчання, що дозволяє визначати обличчя в реальному часі з використанням веб-камери. Основні особливості та перевірені сценарії включають:

Розпізнавання обличчя в кадрі

Система аналізує відеопотік з камери, ідентифікуючи обличчя та відображаючи навколо нього рамку. Дані про кількість облич і швидкість обробки (FPS) відображаються в реальному часі (рис. 4.7, 4.8).

Рисунок 4.7 – Тест на наявність обличчя

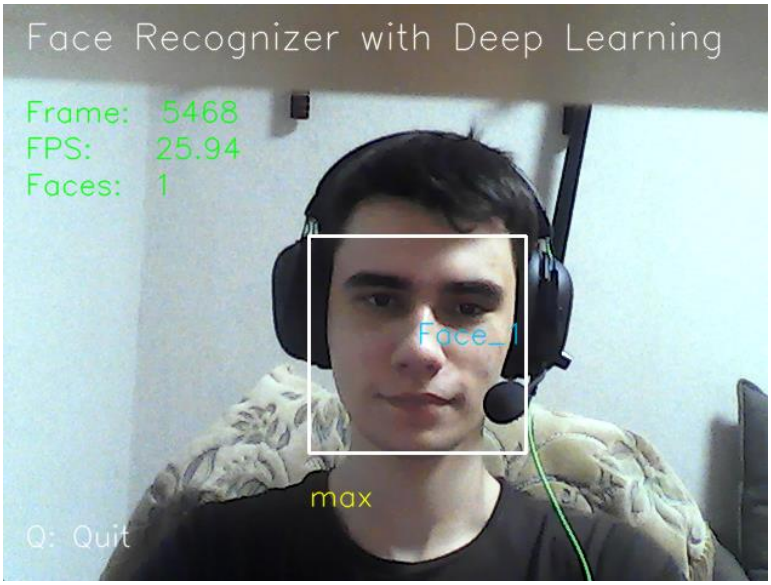

Рисунок 4.8 – Тест на розпізнавння

Підтримка багатьох облич

Система здатна одночасно визначати кілька облич у кадрі, коректно їх рахуючи та обробляючи (рис. 4.9).

Рисунок 4.9 – Тестування на навантаження та відсутність обличчя

Стійкість до зовнішніх факторів

Зміна освітлення: Алгоритм адаптується до змін рівня освітлення, зберігаючи функціональність у більшості стандартних умов (рис. 4.10).

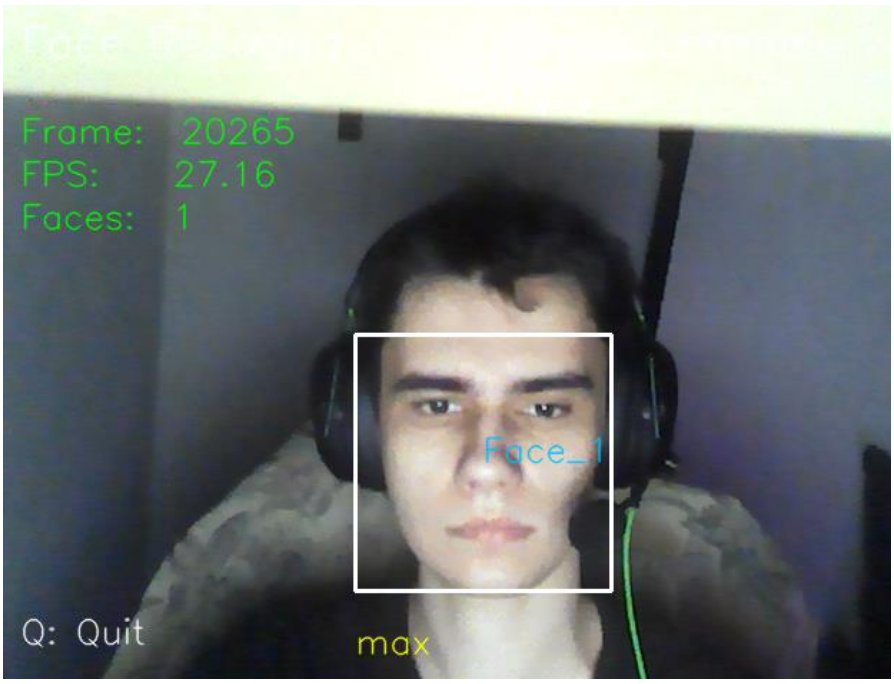

Рисунок 4.10 – Тестування на зміну освітлення

Тестування навантаження

Робота системи перевірялась при додаткових об’єктах у кадрі, таких як фотографії чи відео з екрана, для оцінки її здатності уникати помилкових спрацьовувань. У разі відсутності облич система коректно фіксує 0 об’єктів (рис. 4.11).

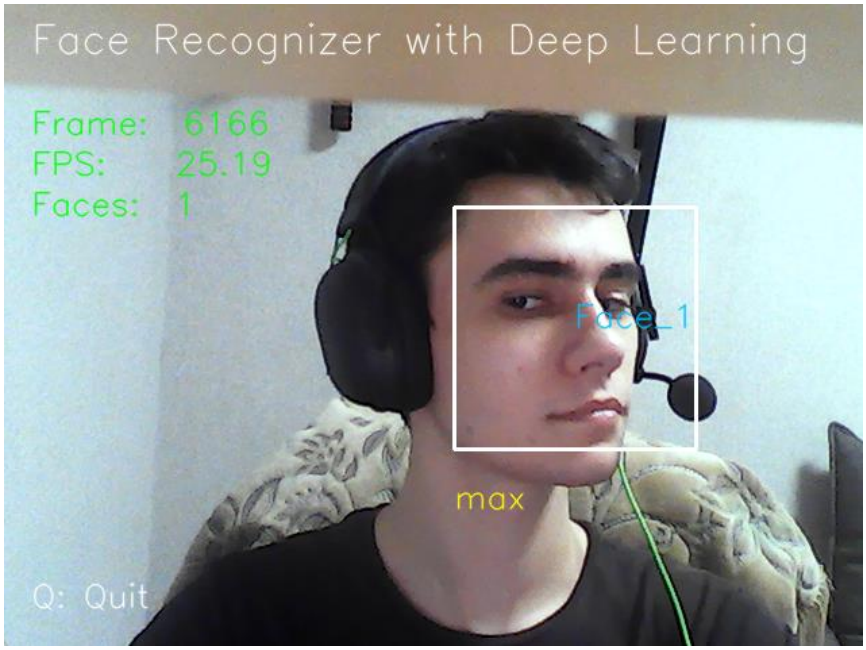

Рисунок 4.11 – Тест на Зміна пози обличчя

Інтерфейс відображає дані кадру, поточну швидкість обробки та кількість розпізнаних облич, що дозволяє проводити моніторинг продуктивності та точності.

Цей функціонал дозволяє забезпечити коректну роботу системи для різних умов, включаючи стандартне розпізнавання, перевірку стійкості та адаптацію до змін у навколишньому середовищі.

4.4 Робота під різними навантаженнями

Тестування системи в умовах різних навантажень є ключовим етапом для забезпечення її стабільності та ефективності, особливо якщо йдеться про системи, що виконують складні обчислення, як-от розпізнавання облич. Метою

таких перевірок є оцінка здатності системи працювати за реальних умов із високим навантаженням: обробляти численні запити, тривало функціонувати без перерв і працювати з великими обсягами даних.

Одним із основних аспектів тестування є перевірка роботи системи з великими базами даних. Вона повинна швидко обробляти тисячі записів і залишатися ефективною навіть за значного збільшення їх кількості. Важливо також оцінити час відповіді системи при високій кількості одночасних запитів, наприклад, коли багато працівників одночасно реєструються для доступу до корпоративних систем або сервісів. Крім того, система повинна масштабуватися для обробки кількох облич одночасно, що є важливим для роботи в великих конференц-залах або під час корпоративних заходів.

Ще одним критично важливим напрямом тестування є перевірка роботи системи при високих навантаженнях на сервер. Це включає оцінку її здатності обробляти великий обсяг запитів до бази даних, взаємодіяти з хмарним сервісом Firebase та коректно зберігати дані навіть за умови одночасної реєстрації кількох адміністраторів чи користувачів.

Не менш важливою є перевірка стабільності системи під тривалим навантаженням. Вона має залишатися продуктивною, без помилок у пам’яті чи збоїв, навіть після кількох годин безперервної роботи. Система також повинна правильно обробляти помилки, наприклад, тимчасову недоступність бази даних або збої у відеопотоці, й успішно відновлювати свою роботу.

Тестування під навантаженням дозволяє оцінити готовність системи до реальних умов використання, виявити потенційні проблеми та впевнитися, що вона працює стабільно й ефективно за будь-яких обставин.

4.5 Виявлення та виправлення помилок та дефектів

Виявлення та виправлення помилок і дефектів є важливим етапом у процесі тестування та оптимізації будь-якої програмної системи, особливо у

проекті, що включає складні технології, як-от розпізнавання облич за допомогою штучного інтелекту. У цьому підрозділі розглянемо процес виявлення, аналізу та усунення помилок і дефектів, що можуть виникнути під час роботи системи, а також методи для їх коригування та запобігання повторному виникненню.

Виявлення помилок

Процес виявлення помилок починається з детального тестування усіх компонентів системи. Ключовими етапами є:

Юзабіліті-тестування: Один з найбільш поширених способів виявлення помилок — це тестування інтерфейсу користувача (UI) і взаємодії з користувачем. Оскільки система включає веб-інтерфейс для взаємодії з користувачами, варто тестувати, як система реагує на різні типи введених даних (наприклад, неправильні або неповні дані, некоректний формат зображень).

Тестування камер та зображень: Основна задача системи — це точне розпізнавання облич. Для цього важливо протестувати систему з різними типами камер та умовами освітлення, щоб виявити помилки в розпізнаванні, які можуть виникнути через низьку якість зображень або неповні дані.

Функціональні тести: Кожна функція, що реалізується в системі, повинна бути перевірена в реальних умовах. Тестування повинно охоплювати перевірку роботи функцій з розпізнавання облич, коректного співставлення з базою даних, а також запису даних у Firebase. Виявлення помилок на цьому етапі дозволяє локалізувати дефекти в логіці або обробці запитів.

Логування помилок: Важливим інструментом для виявлення помилок є система логування, яка допомагає розробникам і тестувальникам відслідковувати помилки під час виконання програми. У коді можна додавати повідомлення про стан виконання коду, відстежуючи, на якому етапі виникають проблеми. Наприклад, використання try-except для обробки виключень дозволяє відловлювати помилки під час роботи з базою даних або розпізнаванням облич.

Тестування під високим навантаженням: Один із способів виявлення помилок — це тестування системи при високому навантаженні, коли одночасно відбуваються численні запити або велика кількість осіб намагається пройти

реєстрацію. Це дозволяє виявити помилки, пов'язані з нестабільністю сервера або бази даних.

Аналіз помилок

Після виявлення помилок необхідно здійснити їхній аналіз, щоб зрозуміти, чому вони виникли, і як їх виправити:

Аналіз помилок розпізнавання облич: Якщо система не може точно ідентифікувати обличчя, важливо проаналізувати, чому це сталося. Однією з причин може бути низька якість зображень або їхня розмитість, недостатнє освітлення чи неприродні кути зйомки. Система повинна бути перевірена на здатність обробляти різні варіанти освітлення і кутів, а також на можливість корекції цих факторів під час обробки.

Помилки у взаємодії з базою даних: Якщо виникають помилки під час взаємодії з Firebase (наприклад, проблеми з підключенням до бази даних або помилки запису), необхідно перевірити правильність налаштувань з'єднання та доступу до бази, а також забезпечити правильну обробку даних, що передаються.

Помилки в обробці відео та зображень: Оскільки система повинна працювати з відео потоком, можливі проблеми, такі як низька частота кадрів, помилки в синхронізації зображень, розмивання зображень тощо. Це може бути результатом неефективної роботи з бібліотекою OpenCV або неякісної роботи камери. Важливо перевірити налаштування відео та роздільну здатність для коректної обробки.

Виправлення помилок

Після аналізу помилок необхідно перейти до їх виправлення, яке включає кілька кроків:

Оптимізація алгоритму розпізнавання обличчя: Для покращення точності розпізнавання можна використовувати інші алгоритми машинного навчання або покращити існуючі методи, такі як використання більш ефективних моделей для обробки зображень. Наприклад, заміна простих методів розпізнавання (Haar Cascade) на більш складні, як-от застосування глибоких нейронних мереж для

екстракції характеристик обличчя, може значно підвищити точність розпізнавання.

Обробка помилок під час роботи з Firebase: Для виправлення проблем із базою даних необхідно перевірити коректність підключення до Firebase, а також забезпечити обробку помилок, пов'язаних із з'єднаннями або обмеженнями на кількість запитів.

Виправлення проблем із камерами та відео: Якщо зображення або відео мають низьку якість, можна впровадити покращення якості зображень за допомогою алгоритмів корекції освітлення або стабілізації відео. Це допоможе підвищити точність розпізнавання навіть у поганих умовах зйомки.

Налаштування тайм-аутів та обробка великих обсягів запитів: Важливо налаштувати тайм-аути та оптимізувати обробку великої кількості запитів, що надходять до сервера. Це дозволить запобігти зависанню системи під час високих навантажень та підвищити її стабільність.

Запобігання повторенню помилок

Після виправлення помилок важливо розробити стратегії, які запобігають їх повторенню:

Покращення тестування: Важливо розширити тести, додавши додаткові сценарії для тестування на різних пристроях, у різних умовах освітлення та при високому навантаженні. Це дозволить виявити потенційні помилки до того, як вони вплинуть на реальних користувачів.

Моніторинг і логування помилок: Встановлення системи моніторингу та логування дозволить відслідковувати помилки в реальному часі. Це дасть можливість швидко реагувати на проблеми та коригувати їх на ранніх етапах.

Покращення взаємодії з користувачами: Для запобігання помилок через неправильне введення даних чи неочевидні дії зі сторони користувача можна впровадити покращений UX/UI дизайн і налаштувати систему підказок для користувачів.

Виявлення, аналіз і виправлення помилок є критичними етапами в процесі тестування системи розпізнавання обличчя. Кожна помилка має бути правильно

виявлена, проаналізована і виправлена, щоб забезпечити безперебійну роботу системи та підвищити її ефективність. Окрім виправлення поточних дефектів, необхідно вжити заходів для запобігання їх повторному виникненню, що включає в себе покращення процесів тестування, моніторингу та взаємодії з користувачами.

4.6 Введення в експлуатацію

Введення в експлуатацію — це фінальний етап після завершення розробки та тестування програмного забезпечення, який включає в себе процес перенесення системи з етапу розробки в реальне середовище експлуатації. Це важливий етап, на якому система стає доступною для кінцевих користувачів. У цьому підрозділі розглянемо деталі процесу введення в експлуатацію для системи розпізнавання облич, враховуючи всі аспекти цього процесу, від підготовки до фактичного запуску та моніторингу роботи системи після запуску.

Підготовка до введення в експлуатацію

Перед тим як система стане доступною для користувачів, необхідно пройти низку підготовчих етапів, що забезпечать правильну роботу і мінімізують ймовірність виникнення неполадок в реальних умовах.

Завершення всіх тестів: На етапі підготовки важливо переконатися, що всі етапи тестування були завершені, а усі помилки та дефекти були виявлені і виправлені. Це включає функціональне тестування, тестування безпеки, тестування продуктивності, тестування на зручність використання (usability) та тестування під різними навантаженнями.

Оптимізація системи: Перед запуском необхідно провести оптимізацію коду та бази даних, щоб система працювала швидко і стабільно під час реальної експлуатації. Наприклад, оптимізація алгоритмів розпізнавання облич, зменшення часу обробки відео та зображень, а також налаштування ефективної

взаємодії з базою даних (Firebase) для забезпечення швидкого доступу до інформації.

Підготовка інфраструктури: Важливо підготувати сервери, на яких буде працювати система, а також налаштувати середовище розгортання. Це включає в себе налаштування серверів для обробки запитів, баз даних, хмарних сервісів (Firebase), а також будь-яких інших компонентів, необхідних для роботи системи. Також потрібно провести перевірку на належну роботу всіх служб.

Резервне копіювання: Оскільки система працює з великими обсягами даних, важливо налаштувати процедури для регулярного резервного копіювання. Це дозволить відновити дані у разі виникнення збоїв або інших непередбачуваних ситуацій.

Навчання користувачів: Для того, щоб кінцеві користувачі, такі як студенти та інструктори, могли ефективно використовувати систему, важливо створити інструкції або провести навчальні сесії. Цей крок забезпечить правильне використання системи та зменшить ймовірність виникнення помилок через неправильне введення даних чи незнання функцій.

Фактичний запуск системи

Після завершення підготовчих етапів можна приступити до фактичного запуску системи в експлуатацію. Цей етап включає кілька важливих кроків, спрямованих на впровадження програмного забезпечення в реальне середовище.

Розгортання програми на сервері: Першим кроком є розгортання програми на сервері або в хмарному середовищі. Враховуючи, що система використовує Firebase для зберігання даних, необхідно перевірити правильність підключення до хмарного сховища та бази даних. Це включає в себе налаштування доступу до сервісів, перевірку конфігураційних файлів та забезпечення надійного з’єднання між компонентами системи.

Перевірка доступності і роботи сервісів: Після розгортання системи потрібно перевірити, чи всі її компоненти працюють належним чином. Це включає перевірку інтерфейсу користувача (UI), алгоритмів розпізнавання облич, можливості запису та отримання даних з Firebase, а також ефективної

роботи системи при високому навантаженні. Важливо, щоб система була доступною для користувачів і правильно обробляла запити.

Інтеграція з іншими системами: Якщо система має інтеграцію з іншими системами, необхідно переконатися, що ця інтеграція працює безперебійно. Тестування всіх взаємодій між компонентами допоможе забезпечити цілісність та безпеку даних.

Моніторинг та підтримка в реальному часі: Після запуску важливо активувати моніторинг системи для виявлення потенційних проблем, таких як збої у розпізнаванні облич, затримки у записі даних, або інші технічні помилки. Система моніторингу повинна включати перевірку активності серверів, виконання запитів до бази даних і доступності компонентів.

Післязапускова підтримка та обслуговування

Після того як система запущена, необхідно забезпечити її належне функціонування в режимі реального часу. Це включає в себе кілька важливих аспектів.

Аналіз відгуків користувачів. Після введення в експлуатацію важливо отримувати відгуки від користувачів, щоб виявити проблеми, які можуть виникнути в процесі експлуатації. Це може включати як функціональні проблеми (наприклад, невірне розпізнавання обличчя), так і проблеми з інтерфейсом користувача.

Виправлення помилок. Виявлення та виправлення помилок, які можуть з’явитися після запуску, є важливим етапом післязапускового обслуговування. Для цього потрібно впровадити систему збору логів і помилок, що дозволить швидко реагувати на будь-які технічні проблеми та здійснювати корекцію.

Оновлення та покращення функцій. Важливо постійно працювати над вдосконаленням системи. Це може включати оновлення алгоритмів розпізнавання облич, поліпшення ефективності роботи з базою даних або додавання нових функцій, які відповідають потребам користувачів.

Резервне копіювання та відновлення. В процесі експлуатації важливо підтримувати регулярне резервне копіювання даних і забезпечувати можливість відновлення системи у випадку збоїв або непередбачених ситуацій.

Скалювання системи. Оскільки система може з часом зростати, важливо мати план для масштабування системи. Це може включати розширення ресурсів сервера, оптимізацію бази даних, а також впровадження нових технологій для обробки великих обсягів даних.

Введення в експлуатацію є важливим етапом, що завершує цикл розробки програмного забезпечення. Він передбачає ретельну підготовку системи, перевірку її працездатності в реальних умовах і забезпечення постійної підтримки після запуску. Забезпечення надійної роботи, зручності для користувачів і моніторинг роботи системи після її запуску допомагають забезпечити високий рівень якості та ефективності роботи системи розпізнавання облич.

ВИСНОВОК

У даній дипломній роботі було здійснено дослідження та розробку інтелектуальної системи обліку робочого часу на основі розпізнавання облич за допомогою штучного інтелекту. Метою розробки було створення ефективного, безконтактного та високоточного інструменту для автоматизації процесу обліку робочого часу, що враховує сучасні вимоги до безпеки та зручності користування.

Аналіз існуючих систем показав, що використання біометричних технологій, зокрема розпізнавання облич, дозволяє значно підвищити точність і ефективність обліку робочого часу, однак потребує вирішення таких проблем, як залежність від освітлення, змін зовнішності та конфіденційності даних. Створена система об'єднує переваги сучасних технологій і забезпечує зручний і надійний спосіб реєстрації робочого часу без необхідності фізичних контактів.

Розробка цієї інтелектуальної системи передбачає інтеграцію технологій машинного навчання для покращення точності та швидкості розпізнавання облич. Під час тестування система показала високу продуктивність та стабільність роботи при різних умовах, зокрема при змінному освітленні та різних позах облич.

Отже, розроблена система є ефективним та надійним рішенням для автоматизації обліку робочого часу, яке може бути адаптоване до потреб різних організацій. Її використання забезпечить зручний та безпечний процес обліку, зменшуючи людські помилки та підвищуючи загальну ефективність роботи. Результати цієї роботи можуть стати основою для подальших досліджень і вдосконалення системи, з урахуванням нових технологічних досягнень та вимог користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. DeepLearning.AI – Курс по використанню глибокого навчання для комп'ютерного зору [Електронний ресурс]. – Режим доступу: https://www.deeplearning.ai/ (дата звернення: 09.10.2024).

2. Medium – Створення системи для обліку робочого часу на основі розпізнавання облич [Електронний ресурс]. – Режим доступу: https://medium.com/ (дата звернення: 07.10.2024). – Назва з екрана.

3. Розробка системи візуалізації даних [Електронний ресурс]. – Режим доступу: https://chartjs.org/ (дата звернення: 13.09.2024).

4. Database Optimization for Efficiency [Електронний ресурс]. – Режим доступу: https://example.com/database-optimization (дата звернення: 12.10.2024). – Назва з екрана.

5. OpenCV – Офіційна документація [Електронний ресурс]. – Режим доступу: https://opencv.org/documentation/ (дата звернення: 01.09.2024).

6. Dlib – Офіційна документація [Електронний ресурс]. – Режим доступу: http://dlib.net/ (дата звернення: 15.09.2024).

7. Python – Офіційна документація [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/ (дата звернення: 12.09.2024). – Назва з екрана.

8. Real-time Updates for Attendance Data [Електронний ресурс]. – Режим доступу: https://example.com/real-time-updates (дата звернення: 10.09.2024). –

9. Flask Documentation [Електронний ресурс]. – Режим доступу: https://flask.palletsprojects.com/ (дата звернення: 10.10.2024). – Назва з екрана.

10. Werkzeug Documentation [Електронний ресурс]. – Режим доступу: https://werkzeug.palletsprojects.com/ (дата звернення: 12.10.2024).

11. Pillow Documentation [Електронний ресурс]. – Режим доступу: https://pillow.readthedocs.io/ (дата звернення: 15.09.2024).

12. Firebase Admin SDK [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs/admin (дата звернення: 11.10.2024). – Назва з екрана.

13. Архітектура клієнт-сервер для баз даних [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/client-server-architecture/ (дата звернення: 14.09.2024). – Назва з екрана.

14. Глибоке навчання та нейронні мережі. URL: https://foxminded.ua/deep-learning/ (дата звернення 11.10.2024).

15. Analytics Vidhya – Використання штучного інтелекту для розпізнавання облич [Електронний ресурс]. – Режим доступу: https://www.analyticsvidhya.com/ (дата звернення: 11.10.2024).

16. NumPy Documentation [Електронний ресурс]. – Режим доступу: https://numpy.org/doc/ (дата звернення: 13.09.2024). – Назва з екрана.

17. Coursera – Курс по комп'ютерному зору для розпізнавання облич [Електронний ресурс]. – Режим доступу: https://www.coursera.org/ (дата звернення: 13.10.2024).

18. Машинне навчання. [Електронний ресурс] URL: https://www.it.ua/knowledge-base/technology-innovation/machine-learning (дата звернення 05.10.2024).

19. Haar Cascade Classifier [Електронний ресурс]. – Режим доступу: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html (дата звернення: 14.09.2024). – Назва з екрана.

20. Machine Learning Mastery – Використання машинного навчання для комп'ютерного зору [Електронний ресурс]. – Режим доступу: https://machinelearningmastery.com (дата звернення: 16.10.2024). – Назва з екрана.

21. TensorFlow – Офіційна документація з використання TensorFlow для комп'ютерного зору [Електронний ресурс]. – Режим доступу: https://www.tensorflow.org (дата звернення: 22.10.2024).

ДОДАТОК А
ТЕСТ-КЕЙСИ

Нижче наведені тест-кейси для виявлення обличчя на зображенні.

Таблиця А.1 – Тест-кейс: виявлення обличчя на зображенні

Опис	Тест на наявність обличчя
Передумови	Зображення з одним обличчям
Кроки	1. Завантажити зображення з одним обличчям. 2. Виконати функцію detect_faces.
Очікуваний результат	Функція повинна виявити одне обличчя на зображенні.
Результат	Успішно

Таблиця А.2 – Тест-кейс: виявлення кількості облич на зображенні

Опис	Тест на кількість облич
Передумови	Зображення з кількома обличчями
Кроки	1. Завантажити зображення з кількома обличчями. 2. Виконати функцію detect_faces.
Очікуваний результат	Функція повинна виявити кілька облич.
Результат	Успішно

Таблиця А.3 – Тест-кейс: виявлення відсутності облич на зображенні

Опис	Тест на відсутність обличчя
Передумови	Зображення без обличчя
Кроки	1. Завантажити зображення без обличчя. 2. Виконати функцію detect_faces.
Очікуваний результат	Функція не повинна виявити облич.
Результат	Успішно

Таблиця А.4 – Тест-кейс: коректність вирівнювання обличчя

Опис	Тест на коректність вирівнювання обличчя
Передумови	Зображення з нахиленим або повернутим обличчям
Кроки	1. Завантажити зображення з нахиленим або повернутим обличчям. 2. Виконати функцію align_face.
Очікуваний результат	Обличчя має бути вирівняне так, щоб очі та рот розташовувалися в стандартних точках.
Результат	Успішно

Таблиця А.5 – Тест-кейс: правильне співпадіння обличчя з базою даних

Опис	Тест на правильне співпадіння
Передумови	Ознаки обличчя, що є в базі даних
Кроки	1. Виконати функцію match_face з ознаками
Очікуваний результат	Повинно бути знайдено співпадіння з обличчям у базі даних.
Результат	Успішно

Таблиця А.6 – Тест-кейс: коректність захоплення відео з камери

Опис	Тест на коректність захоплення відео
Передумови	Користувач має доступ до веб-камери
Кроки	1.Запуститивеб-камеру. 2. Перевірити, чи відео коректно захоплюється та відображається на екрані.
Очікуваний результат	Відео має бути збережене у відповідному форматі та відображене на екрані.
Результат	Успішно

Таблиця А.7 – Тест-кейс: збереження зображення після натискання кнопки

Опис	Тест на збереження зображення
Передумови	Користувач має доступ до веб-камери
Кроки	1. Натискання кнопки "s" для збереження зображення.
Очікуваний результат	Зображення повинно бути збережене у відповідній директорії.
Результат	Успішно

ДОДАТОК В
ВИХІДНИЙ КОД ОСНОВНИХ КОМПОНЕНТІВ

```
from flask import Flask, render_template
from flask import request, redirect, url_for, Response, flash
from werkzeug.security import check_password_hash
from werkzeug.utils import secure_filename
import os
import cv2
from datetime import datetime
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db
from firebase_admin import storage
from detection.face_matching import detect_faces, align_face
from detection.face_matching import extract_features, match_face
from utils.configuration import load_yaml

config_file_path = load_yaml("configs/database.yaml")

ADMIN_PASSWORD_HASH = config_file_path["admin"]["password_hash"]
# print

# Initialize Firebase
cred = credentials.Certificate(config_file_path["firebase"]["pathToServiceAccount"])
firebase_admin.initialize_app(
    cred,
    {
```

```
        "databaseURL": config_file_path["firebase"]["databaseURL"],
        "storageBucket": config_file_path["firebase"]["storageBucket"],
    },
)

def upload_database(filename):
    """
    Checks if a file with the given filename already exists in the
    database storage, and if not, uploads the file to the database.
    """
    valid = False
    # If the fileName exists in the database storage, then continue
    if storage.bucket().get_blob(filename):
        valid = True
        error = f"<h1>{filename} already exists in the database</h1>"

    # First check if the name of the file is a number
    if not filename[:-4].isdigit():
        valid = True
          error = f"<h1>Please make sure that the name of the {filename} is a number</h1>"

    if not valid:
        # Image to database
        filename = os.path.join(app.config["UPLOAD_FOLDER"], filename)
        bucket = storage.bucket()
        blob = bucket.blob(filename)
        blob.upload_from_filename(filename)
        error = None
```

```
        return valid, error

    def match_with_database(img, database):
        "'The function "match_with_database" takes an image and a database as input, detects faces in the
        image, aligns and extracts features from each face, and matches the face to a face in the database.
        '''
        global match
        # Detect faces in the frame
        faces = detect_faces(img)

        # Draw the rectangle around each face
        for x, y, w, h in faces:
            cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 4)

        # save the image
        cv2.imwrite("static/recognized/recognized.png", img)

        for face in faces:
            try:
                # Align the face
                aligned_face = align_face(img, face)

                # Extract features from the face
                embedding = extract_features(aligned_face)

                embedding = embedding[0]["embedding"]

                # Match the face to a face in the database
```

```
            match = match_face(embedding, database)

            if match is not None:
                return f"Match found: {match}"
            else:
                return "No match found"
        except:
            return "No face detected"
        # break # TODO: remove this line to detect all faces in the frame

app = Flask(__name__, template_folder="template")
app.secret_key = "123456"  # Add this line

# Specify the directory to save uploaded images
UPLOAD_FOLDER = "static/images"
app.config["UPLOAD_FOLDER"] = UPLOAD_FOLDER

@app.route("/")
def home():
    return render_template("home.html")

@app.route("/add_info")
def add_info():
    return render_template("add_info.html")

@app.route("/admin_login", methods=["GET", "POST"])
def admin_login():
    if request.method == "POST":
        password = request.form.get("password")
        if check_password_hash(ADMIN_PASSWORD_HASH, password):
```

```
            return redirect(url_for("attendance"))
        else:
            flash("Incorrect password")
    return render_template("admin_login.html")

@app.route("/attendance")
def attendance():
    ref = db.reference("Workers")
    number_workers = len(ref.get())
    workers = {}
    for i in range(1, number_workers):
        workerInfo = db.reference(f"Workers/{i}").get()
        workers[i] = [
            workerInfo["name"],
            workerInfo["email"],
            workerInfo["position"],
            workerInfo["departments"],
        ]
    return render_template("attendance.html", workers=workers)

@app.route("/upload", methods=["POST"])
def upload():
    global filename
    # Check if a file was uploaded
    if "file" not in request.files:
        return "No file uploaded", 400

    file = request.files["file"]

    # Check if the file is one of the allowed types/extensions
```

```
if file.filename == "":
    return "No selected file", 400

if file and allowed_file(file.filename):
    # Make the filename safe, remove unsupported chars
    filename = secure_filename(file.filename)
    # change the name of the file to the workerId
    # Information to database
    ref = db.reference("Workers")
    try:
        # Obtain the last workerId number from the database
        workerId = len(ref.get())
    except TypeError:
        workerId = 1

    filename = f"{workerId}.png"

    # Move the file from the temporal folder to
    # the upload folder we setup
    file.save(os.path.join(app.config["UPLOAD_FOLDER"], filename))

    # Upload the file to the database
    val, err = upload_database(filename)

    if val:
        return err

    # Redirect the user to the uploaded_file route, which
    # will basically show on the browser the uploaded file
    return redirect(url_for("add_info"))
```

```
            return "File upload failed", 400

        def allowed_file(filename):
            # Put your allowed file types here
            ALLOWED_EXTENSIONS = {"png", "jpg", "jpeg", "gif"}
                return "." in filename and filename.rsplit(".", 1)[1].lower() in
ALLOWED_EXTENSIONS

        @app.route("/uploads/<filename>")
        def uploaded_file(filename):
            timestamp = datetime.now().strftime("%Y%m%d%H%M%S")  # for browser
cache
            # Generate the URL of the image
            url = url_for("static", filename="images/" + filename, v=timestamp)
            # Return an HTML string that includes an <img> tag
                return f'<h1>File uploaded successfully</h1><img src="{url}"
alt="Uploaded image">'

        @app.route("/video_feed")
        def video_feed():
                return Response(gen_frames(), mimetype="multipart/x-mixed-replace;
boundary=frame")

        @app.route("/capture", methods=["POST"])
        def capture():
            global filename
            ret, frame = video.read()
            if ret:
                # Information to database
```

```
            ref = db.reference("Workers")

            try:
                # Obtain the last workerId number from the database
                workerId = len(ref.get())

            except TypeError:
                workerId = 1

            # Save the image
            filename = f"{workerId}.png"
            # Save the frame as an image
              cv2.imwrite(os.path.join(app.config["UPLOAD_FOLDER"],  filename),
frame)

            # Upload the file to the database
            val, err = upload_database(filename)

            if val:
                return err
        # Redirect to the success page
        return redirect(url_for("add_info"))

    @app.route("/success/<filename>")
    def success(filename):
        timestamp = datetime.now().strftime("%Y%m%d%H%M%S")  # for browser
cache
        # Generate the URL of the image
        url = url_for("static", filename="images/" + filename, v=timestamp)
        # Return an HTML string that includes an <img> tag
```

```
            return f'<h1>{filename} image uploaded successfully to the database</h1><img src="{url}" alt="Uploaded image">'

    @app.route("/submit_info", methods=["POST"])
    def submit_info():
        # Get the form data
        name = request.form.get("name")
        email = request.form.get("email")
        position = request.form.get("position")
          departments = request.form.getlist("departments")  # Get all selected departments
        password = request.form.get("password")

        # Get the last uploaded image
        workerId, _ = os.path.splitext(filename)
        fileName = os.path.join(app.config["UPLOAD_FOLDER"], filename)
        data = cv2.imread(fileName)

        # Detect faces in the image
        faces = detect_faces(data)

        for face in faces:
            # Align the face
            aligned_face = align_face(data, face)

            # Extract features from the face
            embedding = extract_features(aligned_face)
            break

        # Add the information to the database
```

```
    ref = db.reference("Workers")
    data = {
        str(workerId): {
            "name": name,
            "email": email,
            "position": position,
            "departments": {dept: int("0") for dept in departments},
            "password": password,
            "embeddings": embedding[0]["embedding"],
        }
    }

    for key, value in data.items():
        ref.child(key).set(value)

    return redirect(url_for("success", filename=filename))

@app.route("/recognize", methods=["GET", "POST"])
def recognize():
    global detection
    ret, frame = video.read()
    if ret:
        # Information to database
        ref = db.reference("Workers")
        # Obtain the last workerId number from the database
        number_workers = len(ref.get())
        print("There are", (number_workers - 1), "workers in the database")

        database = {}
        for i in range(1, number_workers):
```

```
            workerInfo = db.reference(f"Workers/{i}").get()
            workerName = workerInfo["name"]
            workerEmbedding = workerInfo["embeddings"]
            database[workerName] = workerEmbedding

        detection = match_with_database(frame, database)

    # Return a successful response
    return redirect(url_for("select_department"))

@app.route("/select_department", methods=["GET", "POST"])
def select_department():
    if request.method == "POST":
        selected_dept = request.form.get("departments")

        # Generate the URL of the image
            timestamp = datetime.now().strftime("%Y%m%d%H%M%S") # for browser cache
                url = url_for("static", filename="recognized/recognized.png", v=timestamp)

        # Information to database
        ref = db.reference("Workers")
        number_workers = len(ref.get())

        for i in range(1, number_workers):
            workerInfo = db.reference(f"Workers/{i}").get()
            if match == workerInfo["name"]:
                if selected_dept in workerInfo["departments"]:
                    ref.child(f"{i}/departments/{selected_dept}").set(
```

```
                    int(workerInfo.get("departments", {}).get(selected_dept)) + 1
                )
                    return f'<h2>Selected Department: {selected_dept} - {detection}</h2><img src="{url}" alt="Recognized face">'
            else:
                return f'<h2>Worker not in department - {detection}</h2><img src="{url}" alt="Recognized face">'
    else:
        return render_template("select_department.html")

def gen_frames():
    global video
    video = cv2.VideoCapture(0)
    while True:
        success, frame = video.read()
        if not success:
            break
        else:
            ret, buffer = cv2.imencode(".jpg", frame)
            frame = buffer.tobytes()
            yield (b"--frame\r\n" b"Content-Type: image/jpeg\r\n\r\n" + frame + b"\r\n")

if __name__ == "__main__":
    app.run(debug=True)

select_department.html
<!DOCTYPE html>
<html>
<head>
```

```
<title>Select Class</title>
<style>
    body {
        font-family: Arial, sans-serif;
        margin: 0;
        padding: 0;
        background-color: #f0f0f0;
        display: flex;
        justify-content: center;
        align-items: center;
        height: 100vh;
    }
    form {
        background-color: white;
        padding: 20px;
        border-radius: 8px;
        box-shadow: 0px 0px 10px 0px rgba(0,0,0,0.1);
        width: 300px;
    }
    h1 {
        text-align: center;
        margin-bottom: 20px;
    }
    select, input[type="submit"] {
        display: block;
        width: 100%;
        padding: 10px;
        margin-bottom: 10px;
        border-radius: 4px;
        border: 1px solid #ddd;
```

```
            box-sizing: border-box;
        }
        input[type="submit"] {
            background-color: #4CAF50;
            color: white;
            cursor: pointer;
        }
        input[type="submit"]:hover {
            background-color: #45a049;
        }
    </style>
</head>
<body>
    <form action="/select_class" method="post">
        <h1>Select the class you will attend now</h1>
        <select name="departments" required>
            <option value="">Select a department</option>
            <option value="IT">Information Technology</option>
            <option value="HR">Human Resources</option>
            <option value="FIN">Finance</option>
            <option value="MKT">Marketing</option>
            <option value="OPS">Operations</option>
            <option value="RND">Research and Development</option>
            <option value="SALES">Sales</option>
            <option value="ADMIN">Administration</option>
            <option value="LEGAL">Legal</option>
            <option value="EXEC">Executive</option>
        </select>
        <input type="submit" value="Submit">
    </form>
```

```
</body>
</html>
```

Home.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Face Recognition</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f0f0f0;
        }
        #header {
            background-color: #333;
            color: white;
            padding: 10px 0;
            text-align: center;
        }
        #header img {
            height: 60px;
        }
        #header h1 {
            display: inline;
            margin-left: 20px;
            vertical-align: middle;
        }
        #left-column {
```

```
    float: left;
    width: 25%;
    padding: 20px;
}
#right-column {
    float: right;
    width: 65%;
    padding: 20px;
}
.video-container {
    position: relative;
    width: 100%;
    padding-top: 60%;
    overflow: hidden;
}
#video {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
}
#footer {
    clear: both;
    background-color: #333;
    color: white;
    text-align: center;
    padding: 10px 0;
    position: fixed;
    width: 33.5%;
```

```
            bottom: 0;
        }
    </style>
</head>
<body>
    <div id="header">

                <h1>Welcome to our Face Detection System for Workplace Attendance</h1>
    </div>

    <div id="left-column">
        <h2>Upload a new face as image</h2>
        <h3>Click the button below to upload a new face as image</h3>
        <p>NOTE: This will only work if you are not in our database</p>
        <form action="/upload" method="post" enctype="multipart/form-data">
            <input type="file" name="file">
            <input type="submit" value="Upload">
        </form>
        <h2>Add yourself to the attendance list</h2>
        <h3>Click the button below to recognize your face from the database</h3>
        <p>NOTE: This will only work if you have already captured a face from the camera</p>
        <button id="recognize-button">Take Attendance</button>
        <p>NOTE: If you are an administrator, please click the button below to login to check attendance</p>
        <button  onclick="window.location.href='/admin_login'">Administrator Login</button>
    </div>
```

```
        <div id="right-column">
            <h2>Capture a new face from Camera</h2>
            <h3>Click the button below to capture a new face from the camera</h3>
            <p>NOTE: This will only work if you are not in our database</p>
            <button id="capture-button">Capture</button>
            <div class="video-container">
                <iframe id="video" src="/video_feed"></iframe>
            </div>
        </div>

        <script>
            document.getElementById('recognize-button').addEventListener('click',
function() {
                var xhr = new XMLHttpRequest();
                xhr.open('POST', '/recognize', true);
                xhr.onload = function() {
                    if (xhr.status === 200) {
                        // Redirect to the select department page
                        window.location.href = '/select_department';
                    }
                };
                xhr.send();
            });

        </script>

        <script>
            document.getElementById('capture-button').addEventListener('click',
function() {
                var xhr = new XMLHttpRequest();
```

```
            xhr.open('POST', '/capture', true);
            xhr.onload = function() {
                if (xhr.status === 200) {
                    // Change the location of the window object to the success page
                    window.location.href = '/add_info';
                }
            };
            xhr.send();
        });
    </script>
</body>
</html>
```

Add_info.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Add Worker Information</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f0f0f0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
        }
        form {
```

```
            background-color: white;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0px 0px 10px 0px rgba(0,0,0,0.1);
            width: 300px;
        }
        h1 {
            text-align: center;
            margin-bottom: 20px;
        }
        input, select {
            display: block;
            width: 100%;
            padding: 10px;
            margin-bottom: 10px;
            border-radius: 4px;
            border: 1px solid #ddd;
            box-sizing: border-box;
        }
        input[type="submit"] {
            background-color: #4CAF50;
            color: white;
            cursor: pointer;
        }
        input[type="submit"]:hover {
            background-color: #45a049;
        }
    </style>
</head>
<body>
```

```
        <h1>Add Worker Information</h1>
        <form action="/submit_info" method="post">
            <input type="text" name="name" placeholder="Full Name" required>
            <input type="email" name="email" placeholder="Email" required>
            <select name="position" required>
                <option value="">Select Position</option>
                <option value="Manager">Manager</option>
                <option value="Developer">Developer</option>
                <option value="Analyst">Analyst</option>
                <option value="HR">HR Professional</option>
                <option value="Administrator">Administrator</option>
                <option value="Executive">Executive</option>
            </select>
            <select name="departments" multiple required>
                <option value="IT">Information Technology</option>
                <option value="HR">Human Resources</option>
                <option value="FIN">Finance</option>
                <option value="MKT">Marketing</option>
                <option value="OPS">Operations</option>
                <option value="RND">Research and Development</option>
                <option value="SALES">Sales</option>
                <option value="ADMIN">Administration</option>
                <option value="LEGAL">Legal</option>
                <option value="EXEC">Executive</option>
            </select>
                <input type="password" name="password" placeholder="Password"
required>
            <input type="submit" value="Submit">
        </form>
    </body>
```

```
</html>
```

Adminlogin.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Administrator Login</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f0f0f0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            flex-direction: column;
        }
        h1 {
            text-align: center;
            margin-bottom: 20px;
        }
        form {
            background-color: white;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0px 0px 10px 0px rgba(0,0,0,0.1);
            width: 300px;
        }
```

```
            input {
                display: block;
                width: 100%;
                padding: 10px;
                margin-bottom: 10px;
                border-radius: 4px;
                border: 1px solid #ddd;
                box-sizing: border-box;
            }
            input[type="submit"] {
                background-color: #4CAF50;
                color: white;
                cursor: pointer;
            }
            input[type="submit"]:hover {
                background-color: #45a049;
            }
            p {
                color: red;
                text-align: center;
            }
        </style>
    </head>
    <body>
        <h1>Administrator Login</h1>
        <form action="/admin_login" method="post">
            <input type="password" name="password" placeholder="Enter password"
required>
            <input type="submit" value="Login">
        </form>
```

```
    {% with messages = get_flashed_messages() %}
        {% if messages %}
            <p>{{ messages[0] }}</p>
        {% endif %}
    {% endwith %}
</body>
</html>
```

face-detection.py

```
import sys
sys.path.append("../")

import cv2
from detection.face_matching import *

import firebase_admin
from firebase_admin import credentials
from firebase_admin import db
from firebase_admin import storage

def match_with_database(img, database):
    # Detect faces in the frame
    faces = detect_faces(img)

    for face in faces:
        try:
            # Align the face
            aligned_face = align_face(img, face)

            # Extract features from the face
```

```
            embedding = extract_features(aligned_face)

            embedding = embedding[0]["embedding"]

            # Match the face to a face in the database
            match = match_face(embedding, database)

            if match is not None:
                print(f"Match found: {match}")
            else:
                print("No match found")
        except:
            print("No face detected")
        # break # TODO: remove this line to detect all faces in the frame

    # Draw the rectangle around each face
    for x, y, w, h in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 4)
        print("Face detected")

    # Display the frame
    cv2.imshow("Detected Faces", img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

cred = credentials.Certificate("../database/serviceAccountKey.json")
firebase_admin.initialize_app(
    cred,
    {
```

```
                "databaseURL": "https://face-recognition-486cb-default-rtdb.firebaseio.com/",
                "storageBucket": "face-recognition-486cb.appspot.com",
            },
        )

        # Information to database
        ref = db.reference("Workers")
        # Obtain the last workerId number from the database
        number_workers = len(ref.get())
        print("There are", (number_workers - 1), "workers in the database")

        database = {}
        for i in range(1, number_workers):
            workerInfo = db.reference(f"Workers/{i}").get()
            workerName = workerInfo["name"]
            workerEmbedding = workerInfo["embeddings"]
            database[workerName] = workerEmbedding

        # print('Database:',database)
        camera_or_image = input("Enter (1) if you have camera\nEnter (2) if you have image: ")

        if camera_or_image == "1":
            # define a video capture object
            vid = cv2.VideoCapture(0)
            while True:
                # Capture the video frame
                # by frame
                ret, frame = vid.read()
```

```
            # Add beatiful text to the frame to say save the image with 's' button
            cv2.putText(
                frame,
                "Press 's' to save the image",
                (50, 50),
                cv2.FONT_HERSHEY_SIMPLEX,
                1,
                (0, 0, 255),
                2,
                cv2.LINE_AA,
            )

            # Display the resulting frame
            cv2.imshow("frame", frame)

            # the 'q' button is set as the
            # quitting button you may use any
            # desired button of your choice
            if cv2.waitKey(1) & 0xFF == ord("s"):
                break
        # After the loop release the cap object
        vid.release()
        # Destroy all the windows
        cv2.destroyAllWindows()

        faces = match_with_database(frame, database)

    elif camera_or_image == "2":
        # Read the image from the workplace camera
```

```
img = cv2.imread("../examples/office_photo.png")
faces = match_with_database(img, database)
```