КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації і інформаційних технологій

Кафедра управління проєктами

П О Я С Н Ю В А Л Ь Н А З А П И С К А
ДО АТЕСТАЦІЙНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

на тему:

Веб-орієнтована інформаційна система для ідентифікації особи за фото на документах

Виконав студент групи: ІСТ-ШІКТ-11м

Чикалов Євгеній Андрійович

(прізвище, ім'я та по батькові повністю)

Спеціальність: 126 "Інформаційні системи та технології»

Освітня програма: Штучний інтелект. Когнітивні технології

Керівник:___ Вікторія БУШУЄВА __

(прізвище, ініціали,)

_____к.т.н., доцент каф. УП___

науковий ступінь, вчене звання

Київ 2023 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: Автоматизації і інформаційних технологій

Кафедра: Управління проєктами

Освітній рівень: Магістр за освітньо-професійною програмою

Галузь знань: 12 Інформаційні технології

Спеціальність: 126 “Інформаційні системи та технології

Освітня програма: Штучний інтелект. Когнітивні технології

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Сергій БУШУЄВ

„___”_____2023 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ АТЕСТАЦІЙНОЇ РОБОТИ

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

Чикалов Євгеній Андрійович

(прізвище, ім’я та по батькові студента)

1. Тема роботи:

Веб-орієнтована інформаційна система для ідентифікації особи за фото на документах

затверджена наказом ректора КНУБА № 2385/2 від «11» листопада 2023 року

2. Керівник роботи:

к.т.н., Бушуєва Вікторія Борисівна

(прізвище, ім’я та по батькові, науковий ступінь, вчене звання)

3. Строк подання студентом роботи до захисту:

за 5 днів до захисту

4. Зміст пояснювальної записки (перелік питань, які слід розробити):

Провести аналіз актуальності, обрати технології для подальшої оптимальної розробки, розробити дизайн інформаційної системи, розробити веб-орієнтовану інформаційну систему.

5. Графічний матеріал за розділами:

Готові приклади розпізнавання, діаграма компонентів хмарних сервісів, скріншоти роботи системи

6. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Збір матеріалів обраного напряму роботи	20.10.2023
Опрацювання та аналіз матеріалів роботи	31.10.2023
Вступ	03.11.2023
Розділ 1. Актуальність	07.11.2023
Розділ 2. Обрання технологій для створення системи	11.11.2023
Розділ 3. Дизайн інформаційної системи	14.11.2023
Розділ 4.Розробка веб-орієнтованої інформаційної системи	23.11.2023
Висновки	25.11.2023
Остаточне оформлення роботи	30.11.2023
Перевірка роботи на плагіат	04.12.2023
Попередній захист роботи на кафедрі	06.12.2023
Направлення роботи на рецензування	08.12.2023

8. Дата видачі завдання _____

Зав. кафедри _____ Сергій БУШУЄВ
(підпис) (прізвище та ініціали)

Керівник _____ Вікторія БУШУЄВА
(підпис) (прізвище та ініціали)

Студент _____ Євгеній ЧИКАЛОВ
(підпис) (прізвище та ініціали)

РЕЗЮМЕ (summary) *до атестаційної випускної роботи студента:*			
ЗВО	Київський національний університет будівництва і архітектури		
Тема	Веб-орієнтована інформаційна система для ідентифікації особи за фото на документах		
Освітній ступінь	Магістр за освітньо-професійною програмою навчання		
Факультет	Автоматизації і інформаційних технологій		
Кафедра	Управління проєктами		
Спеціальність	126 «Інформаційні системи та технології»		
Освітня програма	Штучний інтелект. Когнітивні технології		
Керівник	к.т.н., Бушуєва Вікторія Борисівна		
Обсяг роботи:	*пояснювальна записка, стор.*	*розділів*	*слайдів презентації*
Розділ 1.	Проведено аналіз актуальності, приведені приклади використання розпізнавання зображень державою, описано проблеми які можливо вирішити за допомогою інструментів розпізнавання.		
Розділ 2.	Розглянуто перелік актуальних технологій. Обрано та проаналізовано аналоги на ринку провайдерів хмарних технологій. Обрана мова програмування, описана актуальність її використання та додаткові бібліотеки.		
Розділ 3.	Запропоновано дизайн веб-орієнтованої інформаційоної системи з використанням хмарних технологій. Показано актуальну схему використання та проектування додатку.		
Розділ 4.	Запропоновано процесс розробки веб-орієнтованої інформаційної системи з використанням новітніх технологій, хмарних технологій та технологій машинного навчання.		
Висновки по роботі:	Зроблені висновки щодо актуальності впровадження запропонованого аналогу інформаційної системи у вигляді веб-додатку з урахуванням усіх ризиків та проблем що можуть виникнути.		

Ключові слова: веб-орієнтована інформаційна система, розпізнавання зображень, розпізнавання обличча, архітектура веб-додатку
Keywords: web-oriented information system, image recognition, face recognition, web application architecture

Укладач: _____

Керівник: _____

« ___ » _____ 2023 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації і інформаційних технологій

Кафедра управління проєктами

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Сергій БУШУЄВ
“___” _____2023 року

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО АТЕСТАЦІЙНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

_____

_____
(назва)

Виконав студент групи: ІСТ-ШІКТ-11м

Чикалов Євгеній Андрійович
(прізвище, ім’я та по батькові повністю)

Спеціальність: 126 “Інформаційні системи та технології»

Освітня програма: Штучний інтелект. Когнітивні технології

Керівник:___ Вікторія БУШУЄВА __
(прізвище, ініціали,)
_____к.т.н., доцент каф. УП___
науковий ступінь, вчене звання
Рецензент:_____
(прізвище, ініціали,)

_____
науковий ступінь, вчене звання

Київ 2023р

ВСТУП

Мета атестаційної роботи – опис процесу розробки, конфігурації прототипу веб-орієнтованої інформаційної системи для ідентифікації особи за фото.

Об’єкт атестаційної роботи – веб-орієнтована інформаційна система для ідентифікації особи за обличчам з використанням хмарних технологій.

Предмет атестаційної роботи – використання хмарних технологій штучного інтелекту та машиного навчання для розпізнавання та порівняння зображень людини.

Методи дослідження – у роботі використовуються емпіричні методи дослідження, а саме: вивчення джерел інформації, аналіз отриманих відомостей, спостереження.

Штучний інтелект (AI) - це галузь комп'ютерних наук, яка зосереджена на створенні програм та систем, здатних виконувати завдання, які зазвичай вимагають людського розуму. Це включає такі аспекти, як машинне навчання, обробка природної мови, комп'ютерне зору, розпізнавання мови та експертні системи. Штучний інтелект допомагає нам у багатьох сферах діяльності людини[1].

AI може автоматизувати рутинні та часозатратні завдання, такі як обробка даних, аналіз тексту або відповіді на запитання користувачів.

Також за допомогою AI можна аналізувати великі обсяги даних та виявляти закономірності, які можуть допомогти у прийнятті рішень. Наприклад, AI може допомогти у прогнозуванні продажів, оптимізації логістики або виявленні шахрайства.

Завдяки штучному інтелекту ми можемо покращити медичну діагностику. AI аналізує медичні зображення та допомає у виявляти хвороби, такі як рак або серцево-судинні захворювання. Також AI може допомогти у розробці нових ліків та плануванні індивідуальних планів лікування.

За допомогою штучного інтелекту можливо адаптувати навчальний матеріал до індивідуальних потреб учнів, а також допомагати вчителям у виявленні слабких місць учнів та підтримці їх прогресу.

Навіть сфера розваг також використовує штучний інтелект. AI використовується у відеоіграх для створення реалістичних персонажів та інтерактивних сценаріїв, а також у рекомендаційних системах для підбору фільмів, музики або новин, які можуть вас зацікавити.

У сфері безпеки AI може допомогти у виявленні та запобіганні кібератак, а також у розробці систем безпеки для автономних транспортних засобів.

РОЗДІЛ 1. АКТУАЛЬНІСТЬ

1.1.Поняття комп’ютерного зору

Комп’ютерний зір - підгалузь штучного інтелекту. Він надає комп’ютерам здатність інтерпретувати та розуміти візуальні дані з навколишнього середовища. Ця область штучного інтелекту включає в себе обробку, аналіз і визначення значення – зображень, відео та інших форм графічної інформації[2].

Початковим етапом є збір даних. У системах комп’ютерного зору цей етап передбачає використання камер, датчиків та інших пристроїв для захоплення зображень і відео реального світу. Ці необроблені візуальні дані згодом служать вхідними даними для додаткової обробки - вирішального кроку до аналізу.

Перш ніж аналізувати візуальні дані, їх часто потрібно попередньо обробити, щоб підвищити якість і зменшити обчислювальну складність. Цей процес може включати кілька завдань - зменшення шуму, підвищення контрастності та яскравості, зміну розміру або навіть перетворення зображення на відтінки сірого. На цьому етапі алгоритми комп’ютерного зору беруть участь у вилученні ознак, вони ідентифікують і витягують відповідні особливості з візуальних даних - відмінні візерунки або характеристики, які зображують вміст зображення. Ці функції охоплюють краї, кути, текстури або кольори: усі елементи, які сприяють повному опису зображення.

Системи комп’ютерного зору, використовуючи алгоритми машинного навчання, ідентифікують і класифікують об’єкти в зображеннях шляхом виявлення та розпізнавання об’єктів. Цей процес може включати різні завдання, такі як виявлення обличчя, розпізнавання тексту, класифікація тварин або ідентифікація транспортного засобу. Великі набори даних позначених зображень навчають алгоритми розпізнавати шаблони та особливості, пов’язані з різними класами об’єктів.

Комп’ютерний зір у сфері аналізу відео активно відстежує рух об’єктів у часі, він розпізнає їхню траєкторію та швидкість — функція, яка виявляється корисною для різноманітних застосувань, таких як відеоспостереження, навігаційні можливості автономних транспортних засобів та складний спортивний аналіз.

Окрім простого розпізнавання окремих об’єктів, комп’ютерний зір може заглибитися в розуміння сцени: аналіз цілої сцени для виявлення зв’язків між об’єктами – їхніх атрибутів і дій. Це контекстне розуміння надає системам тонке розуміння взаємодії об’єктів.

Системи комп’ютерного зору після аналізу візуальних даних залучаються до додаткової обробки. Цей крок уточнює результати – він може включати фільтрування помилкових виявлень, або об’єднання інформації з різних джерел. Згодом користувачі можуть використовувати кінцевий результат для прийняття рішень, керування іншими системами, або отримання відповідної інформації.

Спектр застосуваннь дуже широкий та різноманітний. Це можуть бути такі сфери як - автономні транспортні засоби, медична діагностика, системи безпеки, робототехніка, навіть у рекламі та розвагах. Майже усі використовують потужність комп’ютерного зору. Постійний прогрес у технологіях і алгоритмах розширює його здатність сприймати візуальну інформацію з дедалі більшою універсальністю.

1.2.Використання розпінавання зображень державою

Українці дуже часто зустрічають у своєму повсякденному житті використання коп’ютерного зору. Найбільш розповсюдженними прикладами є: системи фотофіксації порушень ПДР, та додаток ДІЯ для ідентифікації особи та підпису документів.

У запропонованому державою мобільному додатку, ми повинні підтверджувати свою особу за біометричними данними (біометрією в цьому

випадку виступає обличча людини). Єдиним ньюансом для стабільної роботи моделі комп'ютерного зору є технічні можливості пристрою користувача (якість зображень створених на фронтальну камеру смартфону).

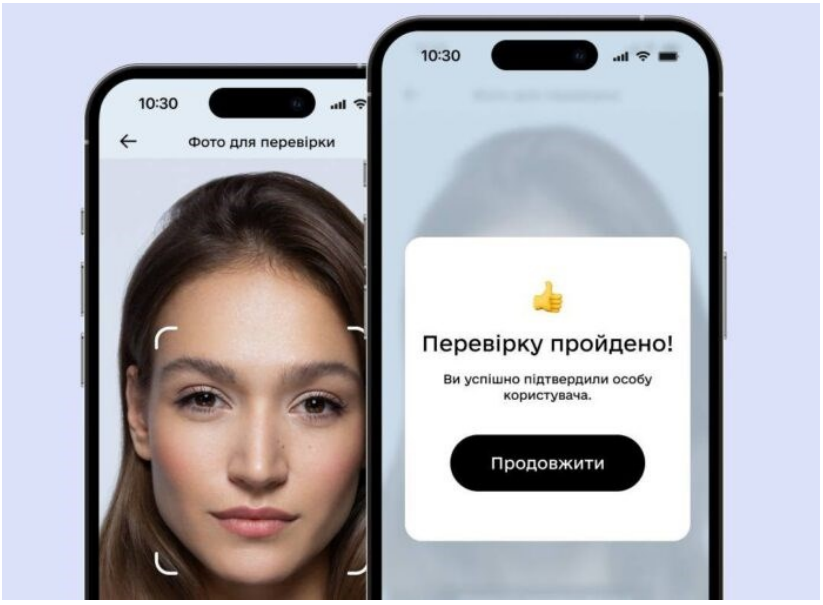

Рисунок 1.1. Приклад перевірки особи в додатку ДІЯ

В системах та програмному забезпеченні що використовує МВС України для ідентифікації порушників ПДР немає обробки зображень облич, але є можливість за допомогою камер що фіксують перевищення швидкості в автоматизованому порядку відокремити та розпізнати на фото автомобільні номери державного зразку[3].

Рисунок 1.2. Приклад фото порушника ПДР з розпізнаванням державного номерного знаку

У данній роботі описано як спроектувати систему з використанням комп’ютерного зору та використовувати у будь-якій сфері за потреби. Навіть небанківські фінансові установи (страхові компанії) починають використовувати розпізнавання облич людей для безпеки клієнтської частини.

1.3.Проблеми які держава може вирішувати за допомогою машинного навчання та розпізнавання зображень

У сьогоднішньому стрімко розвиваючомуся світі державам необхідно адаптуватися та інновувати, щоб вирішувати нагальні проблеми та надавати якісні послуги для громадян. Машинне навчання (ML) та комп'ютерний зір (CV) - потужні інструменти, що мають величезний потенціал для перетворення методів роботи урядів у рішенні питань та наданні послуг своїм громадянам. Уявіть зручність та ефективність, яких ми могли б досягти, застосувавши такі чудові інновації у різних секторах, як охорона здоров'я, транспорт та правоохоронні органи. У цій статті ми з

радістю розглянемо майбутні можливості машинного навчання та комп'ютерного зору революціонізувати багато напрямів роботи для держав.

З допомогою машинного навчання послуги охорони здоров'я можуть бути поліпшені. Вивчаючи закономірності попиту, ми можемо більш ефективно розподіляти ресурси, забезпечуючи оптимальне лікування громадян та скорочуючи час очікування. Більше того, комп'ютерний зір може стати незамінною допомогою для медичних фахівців у діагностиці захворювань та контролю за медичними зображеннями для раннього виявлення і лікування. Вигоди від таких нововведень неоціненні, і ми радіють цій революції в охороні здоров'я.

Щодо транспорту, машинне навчання та комп'ютерний зір можуть працювати разом для оптимізації руху, прогнозування пасажиропотіку та створення найбільше ефективних маршрутів громадського транспорту. Це призведе до швидкішої та більш надійної транспортної системи, а також зробить наш щоденний перевезення більш приємним. Більше того, використання таких технологічних досягнень на дорозі допоможе створити розумніші рішення щодо транспортного керування та організації руху.

З точки зору правоохоронної діяльності, машинне навчання та комп'ютерний зір можуть зміцнити наш боротьбу зі злочинами і забезпечити збільшення безпеки для наших громад. Уявіть технологію комп'ютерного зору, яка сканує відеоматеріали спостереження, виявляє можливу злочинну активність та відстежує підозрілі дії в режимі реального часу. Передбачуючи виникнення злочинів, правоохоронні органи зможуть працювати ефективніше, захищаючи громадян.

Також захоплює можливість застосування машинного навчання та комп'ютерного зору для покращення керування стихійними лихами та відновлення після них. Надаючи передбачення природних катастроф, уряди можуть діяти проактивно, врятувавши життя людей і зменшивши руйнівні наслідки для населених пунктів. Технології комп'ютерного зору

дають можливість отримувати детальне розуміння рівня пошкоджень через супутникові або повітряні зображення, дозволяючи урядам втілити належні міри запобігання.

Наостанок, машинне навчання та комп'ютерний зір можуть мати важливе значення у формуванні економічної політики, допомагаючи урядам приймати рішення, засновані на даних, з впевненістю в напрямках зміцнення державного фінансування.

Висновки до розділу 1

Використовуючи дану систему ми можемо звірити обличчя людини що подала заявлений ідентифікаційний документ з наявним фото (наприклад, id картку, паспорт, водійське посвідчення), що виключає доступ та використання чутливих даних третьою особою, але за умови правильно налаштованої бізнес логіки безпекового етапу на самому підприємстві.

Проаналізувавши наявний ринок фінансових установ, технічних рішень у сфері медицини, безпеки громадського порядку, можна стверджувати, що дана інформаційна система є дуже актуальною.

РОЗДІЛ 2. ОБРАННЯ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ СИСТЕМИ

2.1.Вибір технологій

Сьогодні неможливо уявити IT-сферу без застосування хмарних технологій, які розпочали свій розвиток ще у 1950-х роках. Тоді неможливість придбання дорогої комп'ютерної техніки для багатьох співробітників сприяла створенню рішень, коли декілька користувачів могли одночасно підключатися до одного процесора. Хмарні технології розвивалися завдяки інтернету, оскільки основний принцип таких технологій полягає у забезпеченні доступу користувачів до сервісів з будь-якої точки світу.

На сьогодні хмарні технології широко використовуються у різних сферах діяльності, таких як освіта, медицина, банківська справа, бізнес, торгівля, логістика та інше. Вони надають бізнесу вищу обчислювальну потужність, сучасні сервіси для обробки великої кількості даних та ефективне масштабування залежно від збільшення об'ємів даних. Список доступних послуг від провайдерів хмарних сервісів майже нескінченний і постійно оновлюється. Сучасні хмарні технології охоплюють сервери, сховища даних, бази даних, мережі, програмне забезпечення різних видів, системи контролю та моніторингу тощо.

Хмарні технології надаються користувачам у формі звичайного онлайн сервісу, при цьому виділяють три рівні хмарних сервісів - SaaS, PaaS та IaaS. Вони представляють собою моделі хмарних технологій, що формують піраміду за різним рівнем контролю над інформацією. На вершині піраміди знаходиться користувач сервісу, який працює з набором даних завдяки програмному забезпеченню з зручним інтерфейсом. Якщо програмне забезпечення розгорнуте на платформі, це відповідає другому рівню хмарних сервісів. Третій рівень, основа піраміди, відповідає за віртуальні сервери, обчислювальну потужність та сховища даних.

З розвитком інтернету хмарні сервіси типу SaaS (Software as a Service, "програмне забезпечення як послуга") стали широко розповсюдженими,

дозволяючи відрізнити онлайн-сервіси від десктопних програм, для встановлення яких потрібен персональний комп'ютер.

Коли в технологічному світі виникла потреба швидкого розгортання нових програм, з'явилася ідея PaaS (Platform as a Service, "Платформа як сервіс"). З часом еволюція хмарних технологій привела до ситуації, коли великі компанії мали безліч невикористаних обчислювальних ресурсів. Продаж цих ресурсів породив новий рівень хмарних технологій - IaaS (Infrastructure as a Service, "Інфраструктура як послуга")[4].

Схематично сервіси зображують так:

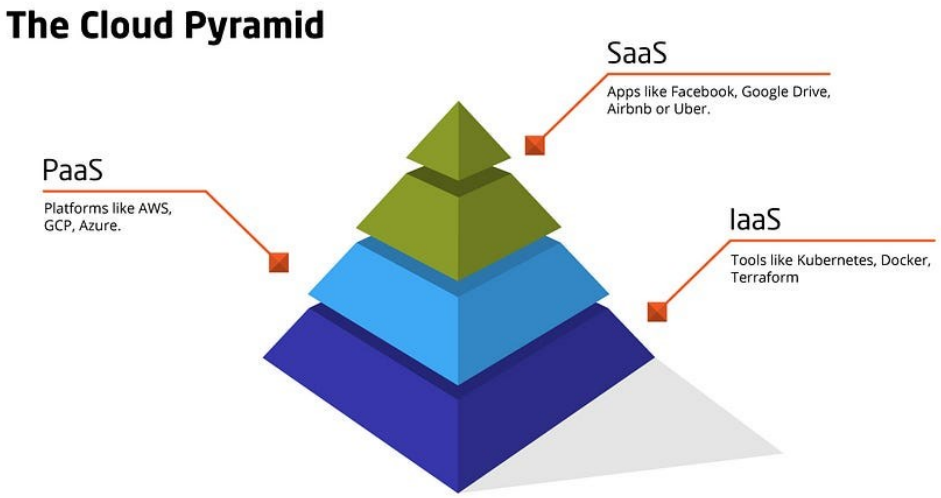

Рисунок 2.1. Хмарна піраміда

В сучасному світі численні компанії пропонують свої послуги у вигляді хмарних технологій. Однак, на даний момент лідерами у цій галузі є Amazon Web Services, Azure та Google Cloud.

Іноді вибір хмарного сервісу може бути більш розумним, ніж розробка власного рішення на фізичній машині, через багато переваг.

Хмарні сервіси дають можливість легко збільшувати або зменшувати ресурси відповідно до потреб. Ви можете швидко отримати більше ресурсів, коли є високий попит, або зменшити їх у спокійний період.

Використання хмарних служб допомагає заощадити витрати, пов’язані з купівлею, установкою та обслуговуванням фізичних пристроїв, а також електроенергією та охолодженням для серверів. Хмарні сервіси дають змогу налаштувати проект за лічені хвилини, а підготовка власної фізичної інфраструктури може зайняти кілька тижнів або місяців. Крім того, доступ до хмарних сервісів можна отримати з будь-якого місця, де є підключення до Інтернету.

Хмарні служби пропонують зручні панелі керування та інструменти, які підтримують автоматичне розгортання, моніторинг і застосування політик безпеки, а також інтеграцію зі сторонніми службами та API. Провадери провайдери витрачають багато ресурсів на підтримку надійності та безпеки інфраструктури, гарантуючи наявність резервного копіювання, тиражування даних, захист від DDoS-атак та інших загроз[5].

Використовуючи хмарні рішення ви отримуєте найновіші технології та оновлення безпеки, які автоматично встановлюються постачальником, звільняючи вашу команду від необхідності підтримувати та оновлювати фізичну інфраструктуру.

У результаті використання хмарних сервісів може бути вигідним для більшості сучасних організацій, залежно від їхніх потреб, програмного та апаратного забезпечення[6].

Нижче в таблиці задано перелік найбільш популярних сервісів

Таблиця 2.1

№	Назва провайдера
1	Amazon Web Services
2	Microsoft Azure
3	Google Cloud
4	IBM Cloud
5	Oracle Cloud

Рисунок 2.2. Логотипи хмарних

6	Alibaba Cloud
7	DigitalOcean
8	Rackspace Cloud
9	Linode
10	OVH Cloud

Але щоб обрати найкращій для вирішення задачі створення веб-орієнтованої інформаційної системи верифікації особи за фото потрібно провести аналіз хмарних провайдерів представлених у таблиці вище.

Таблиця 2.2

Порівняння хмарних провайдерів

№	Назва провайдера	Інновації	Масштаб	Ціни	Сфера застосування
1	Amazon Web Services	висока	Найбільша	середні	Універсальний: веб-сайти, аналітика, AI, IoT, сховища даних, комп'ютерний зір
2	Microsoft Azure	висока	середня	середні	Універсальний: веб-сайти, мобільні застосунки, IoT, віртуальна наука
3	Google Cloud	висока	низька	низькі і середні	Аналітика, машинне навчання, AI, веб-сайти
4	IBM Cloud	середня	середня	дорожчі	Штучний інтелект, машинне навчання, блокчейн

5	Oracle Cloud	середня	низька	дорожчі	Бази даних, Аплікації, архівування
6	Alibaba Cloud	середня	найменша	низькі	Алікації, веб-хостинг, безпека, моніторинг
7	DigitalOcean	низька	низька	низькі	Веб-хостинг, мобільні застосунки
8	Rackspace Cloud	низька	низька	середні	Веб-сервіси, хостинг, безпека
9	Linode	низька	низька	низькі	Веб-хостинг, віртуальні сервери
10	OVH Cloud	низька	середня	низькі в Європі	Хостинг, сервери, бази даних

Після проведенного порівняльного аналізу було обрано один із найбільш популярних хмарних провайдерів Amazon Web Services (AWS).

Успішно обравши провайдера, тепер потрібно проаналізувати наявні сервіси та обрати які саме ми можемо використовувати для задач розпізнавання зображень.

Список сервісів що використовують штучний інтелект:

Таблиця 2.3

Опис сервісів з використанням машинного навчання

№	Сервіс	Опис
1	Amazon SageMaker	Платформа, що спрощує побудову, навчання та розгортання моделей машинного навчання на будь-якому масштабі

2	Amazon Lex	Сервіс для створення розмовних інтерфейсів з інтелектуальним такими як чат-боти, при цьому використовує голосове та текстове розпізнавання
3	Amazon Polly	Сервіс, який перетворює текст на мовлення з використанням глибоких алгоритмів машинного навчання
4	Amazon Rekognition	Сервіс розпізнавання образів та відео, який базується на глибокому навчанні моделей з розпізнавання об'єктів, сцен, осіб, емоцій тощо
5	Amazon Translate	Автоматичний сервіс перекладу, який використовує нейромережі останнього покоління для перекладу тексту між 50+ мовами
6	Amazon Comprehend	Сервіс аналізу природної мови з можливістю виявлення емоцій, ключових фраз та тем у тексті
7	Amazon Transcribe	Сервіс перетворення голосу на текст, який підтримує розпізнавання різноманітних мов та акцентів, розпізнавання диктора та інші функції
8	AWS DeepLens	Камера для глибокого навчання з набором API, які дозволяють розробникам легко інтегрувати її в свої проекти використовуючи камеру
9	Amazon Forecast	Сервіс прогнозування, який допомагає передбачати такі різноманітні фактори, як попит, продажі та обсяги навантаження

Із переліку наявних сервісів ми бачимо що Amazon Rekognition відповідає нашим вимогам і ми можемо використати його. Тепер потрібно перевірити доцільність використання.

Amazon Rekognition — це служба що дозволяє програмам аналізувати зображення та відео за допомогою новітніх алгоритмів машинного навчання, глибокого навчання та комп’ютерного бачення. Його можна використовувати

для багатьох сценаріїв, таких як безпека, сортування вмісту, реклама, аналітика подій і розпізнавання об’єктів на фотографіях або відео[7].

Amazon Rekognition може автоматично виявляти тисячі об’єктів, сцен і загальних понять на зображеннях, надаючи інформацію, необхідну для сортування, фільтрації та пошуку вмісту. Цей сервіс може аналізувати обличчя людей на зображеннях, ідентифікуючи такі риси обличчя, як стать, вік та емоції, а також може зіставляти обличчя відомих людей із бази даних. Він також може відстежувати обличчя користувачів у відео в реальному часі.

Також Amazon Rekognition може виявляти текст на зображеннях і екранах із різноманітних джерел, включаючи знаки, брошури, документи, номерні знаки тощо[8]. Навіть за певної конфігурації Amazon Rekognition може автоматично ідентифікувати небажаний або небезпечний вміст у відео та зображеннях, уможливлюючи модерацію вмісту та дотримання політики.

Модель розпізнавання може визначити міжособистісні стосунки між людьми на зображенні, наприклад відстань між ними, чи дивляться вони один на одного, їхнє розташування тощо. Amazon Rekognition може відстежувати людей і об’єкти в реальному часі у відео, призначаючи їм унікальні ідентифікатори.

Цінова політика Amazon Rekognition максимально лояльна та може зберегти кошти простою віддаленого серверу (у випадку його оренди), бо ціни для зображень дуже низькі.

- Для перших 5 000 оброблених зображень на місяць сервіс є безкоштовним.
- За наступні 5 000 001 - 10 000 000 зображень, сплачуватимете $1,00 за кожні 1 000 зображень.
- За 10 000 001 - 50 000 000 зображень, сплачуватимете $0,80 за кожні 1 000 зображень.
- Вартість для 50 000 001 та більше зображень дорівнює $0,65 за кожні 1 000 зображень.

Для збереження зображень ми також будем використовува інші сервіси, зокрема Amazon S3.

Amazon S3 (Simple Storage Service) від Amazon Web Services (AWS) - це надійний, масштабований та безпечний сервіс зберігання даних у хмарі. Завдяки своїй гнучкості та низьким вартостям, S3 став популярним сервісом серед індивідуальних розробників та великих корпорацій для зберігання, архівації та аналізу даних[9].

Однією з ключових переваг Amazon S3 є його ємність, яка дозволяє масштабувати зберігання від декількох мегабайтів до екзабайтів даних. Завдяки його віртуалізації користувачі можуть налаштувати умови зберігання відповідно до своїх потреб, а також керувати доступом до даних, використовуючи ключі шифрування, групи доступу та інші механізми захисту.

Amazon S3 пропонує простий та зручний інтерфейс для завантаження, зберігання та управління файлами. З інтерфейсу користувача можна легко завантажувати файли, а також використовувати API для їх доступу, обробки та передачі. Це сприяє співпраці між розробниками та додатками.

Оскільки S3 базується на інфраструктурі Amazon Web Services, налагодженій на високу стійкість, сервіс забезпечує 99,99% доступності даних та можливість створення аварійних копій, що мінімізує ризик втрати даних. Більше того, з силами таких функцій, як перевірка на відповідність стандартам та автоматизоване керування життєвим циклом зберігання, Amazon S3 забезпечує безпеку даних та допомагає оптимізувати витрати.

Вартість S3 залежить від обсягу зберігання, пропускної здатності та операцій запитів, але загалом вважається одним з найбільш вигідних рішень для зберігання даних у хмарі. У додаток, AWS пропонує безкоштовний рівень для нових користувачів Amazon S3, який дозволяє їм спробувати сервіс перед використанням платних тарифів.

На сьогоднішній день Amazon S3 є одним з найбільш популярних, безпечних та гнучких рішень для зберігання даних у хмарі для всіх видів бізнесу та організацій. Завдяки своїм інноваційним характеристикам, він продовжує розвиватися та адаптуватися до нових викликів, які постає перед користувачами хмарних технологій.

Таблиця 2.4

Порівняння звичайних файлових серверів та Amazon S3

Критерій	Звичайні файлові сервери	Amazon S3
Вартість	Витрати на обладнання, обслуговування та відновлення інфраструктури. Високі первісні витрати	Оплата лише за використаний обсяг та пропускну здатність. Низькі первісні витрати
Масштабованість	Обмежена ємністю фізичного обладнання та необхідністю слідкувати за потребами у просторі зберігання	Легко збільшувати або зменшувати обсяг хмарного зберігання, від мегабайтів до екзабайтів
Надійність	Уразливість до відмови обладнання, природніх катастроф та помилок людського фактору	Висока надійність за рахунок реплікації даних та запобіжних заходів, прийнятих AWS
Життєвий цикл даних	Потрібне ручне управління із застосуванням політик архівації та резервного копіювання; витратні та трудомісткі процеси	Автоматизоване керування життєвим циклом зберігання та налаштування переходу від гарячих до холодних даних

Швидкість доступу	Залежить від розміщення користувача та доступності VPN. Можливі затримки при високому навантаженні на сервери	Фізичне місцезнаходження майже не впливає на доступність данних та пропускну здатність
Безпека	Вимагає детального планування і керування, щоб захиститися від атак та втрати даних	Інтегровані забезпечувальні засоби, такі як шифрування, контроль доступу і захист від DDoS-атак
Сумісність	Обмеження на підтримувані формати файлів, доступ та інтеграцію з програмами	Широкий спектр підтримуваних форматів файла, API для інтеграції з програмами, віддалений доступ через Інтернет

Аналізуючи порівняння також ми можемо дійти висновку що економічно вигідніше буде використати AWS S3 для зберігання зображень.

2.2.Мова програмування

Java - це високорівнева, всесвітньо відома мова програмування, створена компанією Sun Microsystems у 1995 році. З часом створення, Java стала однією з найпопулярніших мов програмування, завдяки своїй гнучкості, надійності та межам платформи[10].

Java була розроблена з метою покращення мобільності та безпеки програм, зокрема використовуючи принцип "пиши один раз, запускай увсячі". Ця ідея досягалася через використання віртуальної машини Java

(JVM), яка перетворює код на міжплатформену байт-код, що забезпечує незалежність від певної апаратної платформи. Таким чином, код, написаний на Java, може використовуватися на різних платформах та операційних системах без необхідності змінювати його для кожної системи (за умов правильної конфігурації віртуальної машини Java).

Ще однією важливою особливістю мови Java є її підтримка об'єктно-орієнтованого програмування, яка сприяє структурованому підходу до розробки програм, а також створенню гнучкого та повторно використовуваного коду. Java містить багато готових класів та бібліотек, що забезпечують широке використання об'єктно-орієнтованих парадигм.

Безпека та зручність користувачів також стали важливими розглядами при створенні Java. Java включає ряд механізмів безпеки, таких як брандмауер, який забезпечує контроль доступу та ізоляцію коду, а також механізми керування пам'яттю, які автоматично займаються очищенням пам'яті та покращують процес виявлення та збору сміття.

Java орієнтована на роботу з великими й складними ентерпрайз-проектами та при цьому простими щоденними програмами. Це забезпечило те, що Java використовується в широкому спектрі застосунків - від чат-ботів та онлайн-графічних редакторів до веб-сайтів, мобільних додатків та корпоративних систем. Java також є популярним вибором для забезпечення безперебійної роботи електронних керуючих систем, таких як автоматизовані навігаційні системи та системи промислового контролю.

За допомогою Java можна створювати потужні, масштабовані та безпечні програми для різних платформ за рік. З Constвіртуальніма машина (JVM) та різноманітних класів та бібліотек, доступних для розробників, Java продовжує залишатися однією з найпотужніших мов програмування, котрі активно використовуються й учаться всього світу[11].

Для написання Front-End частини було використанно HTML, CSS та JS. Для вдосконалення UI та UX, було піде додатково підключено

бібліотеку jQuery та використаємо Bootstrap для мінімальної адаптивності нашого веб-орієнтованого додатку.

Щоб уникнути можливості підставляти фото людей замість реального обличча ми повінні відфільтровувати ці дії на самому початку тому додамо ще бібліотеку з Face API.

У якості фреймворку для написання Back-End частини мовою Java буде обрано Spring екосистему.

Spring Framework - це потужний, гнучкий та відкритий фреймворк для створення Java-додатків різних масштабів, від невеликих веб-проектів до великих корпоративних систем. Створений Родом Джонсоном у 2003 році, Spring Framework розроблений з метою полегшення розробки, спрощення конфігурації та забезпечення оптимальної роботи різних компонентів Java-додатків[12].

Одним з ключових принципів Spring Framework є ін'єкція залежностей (Dependency Injection, DI), що дозволяє розробникам автоматично поєднувати взаємозв'язані компоненти для створення легкого, гнучкого та повторно використовуваного коду. За допомогою DI Spring Framework допомагає вирішити проблеми жорсткої зв'язаності (tight coupling) між класами, поліпшує тестування програм та підтримку коду.

Spring Framework пропонує модульну систему з гнучкими та взаємозамінними компонентами. Ці компоненти, відомі як Spring Projects, адресують різноманітні питання розробки Java, такі як безпека, доступ до даних, інтеграція з іншими системами, веб служби та багато іншого. Це робить Spring Framework першокласним механізмом для будь-якої Java-орієнтованої розробки, що може допомогти прискорити проект та полегшити роботу розробників.

Spring Boot - це підпроект Spring Framework, розроблений для полегшення створення та запуску автономних Java-додатків. Spring Boot

автоматично налаштовує ваші Spring-додатки на основі призначення для користувача, пропонує різноманітну підтримку спільноти та мінімізує кількість коду, необхідного для створення каркасів. Він також має багато вбудованих підтримок для популярних Java-технологій, таких як JDBC, JPA, RESTful сервіси, роботу з кешуванням та інтеграцією з актуальними системами розгортання.

Spring Framework та Spring Boot забезпечують широку підтримку різних платформ та середовищ розробки Java. Вони інтегровані з багатьма іншими наборами інструментів для розробки та надають розробникам єдиний временіні фрейм для прискореного розробки, швидкого та ефективного впровадження програм Java.

Отже, остаточний набір технологій що буде використано:

- Back-End: Java (Spring Framework)
- Front-End: HTML, CSS, JS та додатково (jQuery, Bootstrap, FaceAPI)
- Cloud services: AWS а саме AWS Rekognition та AWS S3

Висновки до розділу 2

Обрані технології є максимально актуальними, використовуються в повсякденні в багатьох комерційних проектах. Обрано мову програмування та хмарні технології що дають максимальну ефективність при мінімальній вартості на розробку, впровадження та підтримку системи.

РОЗДІЛ 3. ДИЗАЙН ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1.UML діаграма

У сфері розробки програмного забезпечення розробка чіткої та добре структурованої системи має вирішальне значення. Інструментом, який зазвичай використовується для цієї мети, є Уніфікована мова моделювання (UML), яка є стандартизованою візуальною мовою, яка представляє та передає аспекти програмної системи. Діаграми UML необхідні для ілюстрації структури, поведінки та взаємодії програмних компонентів, сприяючи більш ефективній розробці та співпраці.

Основними елементами є прецедент (варіант користування) та учасники (actor). Actor – це логічний зовнішній об'єкт, що безпосередньо взаємодіє з системою. Прецедент – це опис варіантів (послідовних дій учасника), що призводить до отриманням спостережуваного (очікуваного) результату.

Для нашої веб-орієнтованої інформаційної системи достатньо визначити лише 2 учасників що будуть взаємодіяти з системою та 3 прецедента.

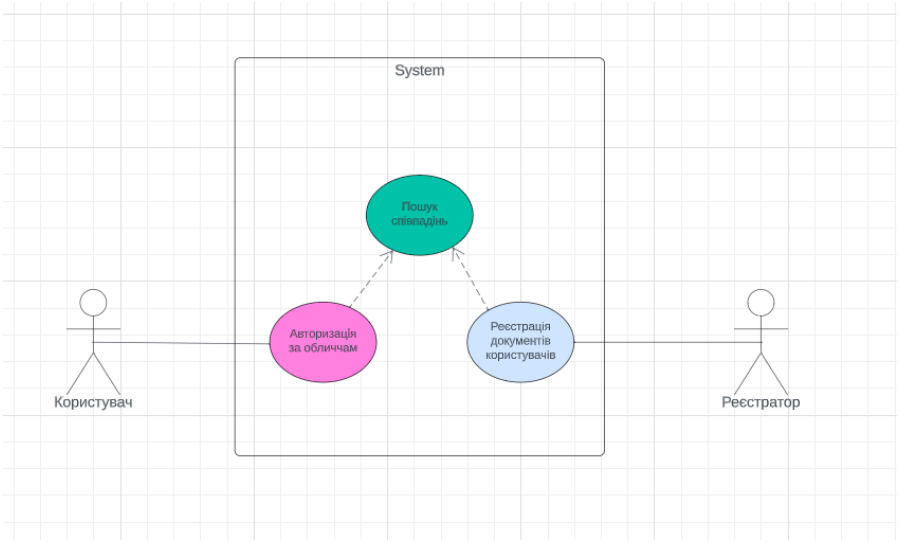

Рисунок 3.1. UML діаграма

Серед учасників (акторів) майбутньої інформаційної системи ми можемо виділити:

- Користувач – звичайний клієнт де потрібно буде інтегрувати цю інформаційну систему.

- Реєстратор – умовно особа зі сторони надавача послуг, що має можливість вносити реєстраційні данні про «Користувачів».

Користувач має доступ до функціоналу «Авторизація за обличчам»

Реєстратор має доступ до функціоналу «Реєстрація документів користувачів»

А ці два прецеденти фактично виконують функціонал «Пошук співпадінь»

3.2.Діаграма компонентів веб-орієнтованої системи

Для більшого розуміння роботи системи в цілому потрібно спроектувати архітектуру веб-орієнтованої інформаційної системи за обраними технологіями.

На рисунку ми можемо зобразити взаємодію користувача під час того як він намагається пройти авторизацію.

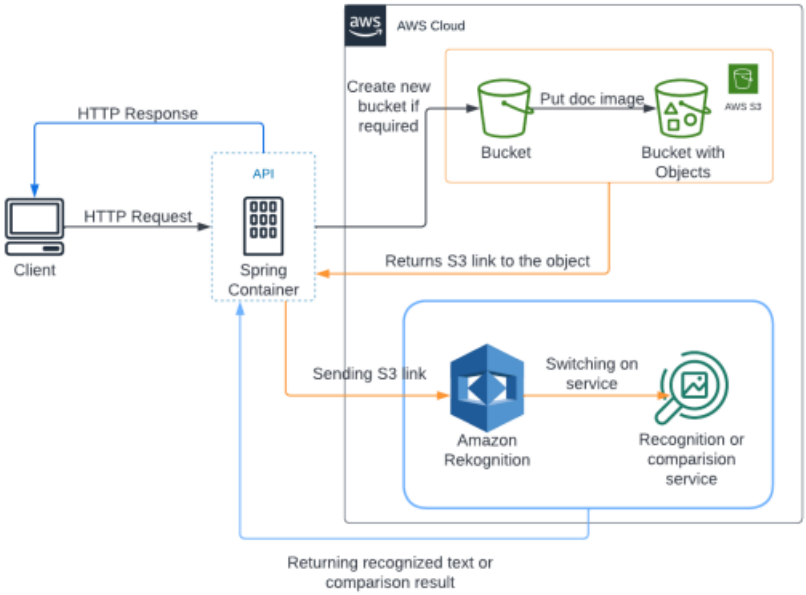

Рисунок 3.2 Діаграма вдаємодії хмарних компонентів

Клієнт або Реєстратор фактично будуть взаємодіяти з системою за аналогічними сценаріями, але у випадку «Реєстратора» система поверне текстовий результат, а у випадку «Клієнта» результат співпадіння.

Висновки до розділу 3

Описана діаграма компонентів веб-орієнтованої інформаційної системи з вказанням взаємодії компонентів та користувача, а також показано прототипізовану UML діаграму.

РОЗДІЛ 4. РОЗРОБКА ВЕБ-ОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

4.1.Архітектура додатку

Під час вибору основних технологій для проектування системи ми вибрали мову Java, але для взаємодії системи в цілому також потрібні й інші бібліотеки/імпортовані пакети.

У проекті використовується велика кількість імпортованих пакетів (бібліотек/залежностей) та фреймворків для скорочення часу розробки та покращення оптимізації серверної частини. Ми обрали Apache Maven для автоматизації процесу збірки, оскільки це один з найпопулярніших інструментів з упорядкованою моделлю для управління збіркою, яка базується на XML конфігурації. При створенні проекту та визначенні залежностей, що потрібно імпортувати, ми користуємось POM-файлом.

Цей файл містить список всіх імпортованих пакетів, які можна оглянути та дослідити.

Для зручності розробки було прийнято рішення використовувати паттерн MVC (Model-view-controller), щоб компоненти що мають певне смислове навантаження були максимально згруповані.

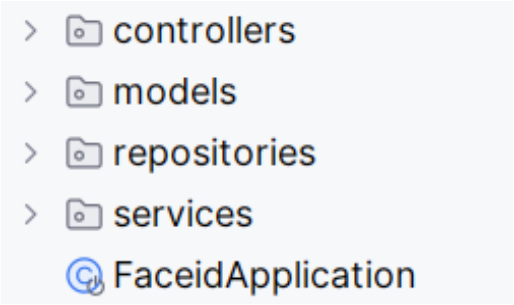

Рисунок 4.1. Приклад згрупованих компонентів

Як ми бачимо з рис. То основні компоненти розділені на 4 пакети. Кожен з них містить класи, що логічно пов’язані між собою:

- Controllers – містить функціонал що відповідає за HTTP запити. Приймає за певним ендпоінтом інформацію, викликає модель, та повертає значення оброблені моделлю.
- Models – містить класи в яких описана основна взаємодія з даними: парсинг файлів, генерація файлів, манупуляції з даними
- Repositories – фактично містить в собі лише інтерфейси, що дозволяють нам робити операції (запити) до бази даних
- Services – містить і класи і інтерфейси у вигляді обгорток над стандартними бібліотеками та імпортованими. Наприклад обгортка бібліотеки для роботи з AWS виглядає таким чином:

```
1 usage   Yevhenii.Chykalov_rft *
public float compare(Image source, Image target){
    CompareFacesRequest request = new CompareFacesRequest().withSourceImage(source).withTargetImage(target);
    List<CompareFacesMatch> faces = amazonConfig.rekognition().compareFaces(request).getFaceMatches();
    if(faces.isEmpty()){
        return 0;
    }
    return faces.get(0).getSimilarity();
}
```

Рисунок 4.2. Приклад обгортки сервісу

4.2.Розбір API частини

Для самого початку нам потрібно розробити Back-End частину. Фактично нам її буде достатньо для майбутніх інтеграцій у якості API. Розробивши цю частину ми зможемо вже взаємодіяти з системою без UI (User Interface) виконуючи запити за допомогою програми Postman, Insomnia, або взагалі cURL утиліти.

Першим кроком потрібно описати саме контролери, розділивши їх логічно, в нас їх буде декілька:

На авторизаційні та реєстраційні дії

1. @GetMapping(path = "/login")
2. @PostMapping(path = "/register", produces = "application/json")

На генерацію валідацію токенів (потрібно для безпеки використання сервісу)

3. @GetMapping("/token_generate")
4. @GetMapping("/token_verify")
5. @GetMapping("/token_get")

На порівняння інформації зображень

6. @GetMapping(path = "/compare", produces = "application/json")

Спочатку потрібно описати логіку для нашого реєстратора. Функціями реєстратора буде внесення документів до умовного реєстру/бази.

Реєстратор повинен викликати ендпоінт /register та передати туди зображення у вигляді рядку Base64 що фактично містить в собі масив байт зображення. Тобто зі сторони клієнта (Front-End) повинна бути виконана умова перетворення зображення в масив байт, а потім з масиву байт до Base64. Або можливо спростити використвши MultipartFile, але певно це викличе певні дефекти зі сторони швидкодії, тому цю реалізацію потрібно робити в залежності від доступних ресурсів.

Цей ендпоінт повинен зберегти зображення до Amazon S3 згідно схеми поданої в розділі дизайну інформаційної системи та порівняти вже з присутніми на схожість.

Після успішного збереження реєстратор може отримати інформацію з документа.

AWS Rekognition дозволяє це зробити, він знаходить абслютно всі текстові елементи та розбиває їх на певні блоки.

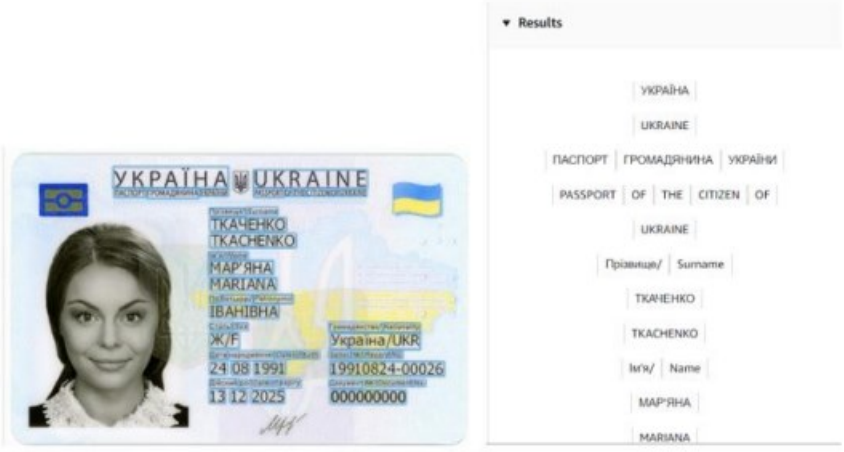

Рисунок 4.3. Приклад розпізнавання документу

Реалізувавши частину авторизації, нам обов’язково потрібно виконати мінімальні умови для зняття зображення з камери клієнта. На клієнтській стороні потрібно реалізувати функціонал що захоплює обличча та визначає емоції людини. Для більшої безпеки потрібно попросити 2 дії. Наприклад ми можемо попросити посміхнутися

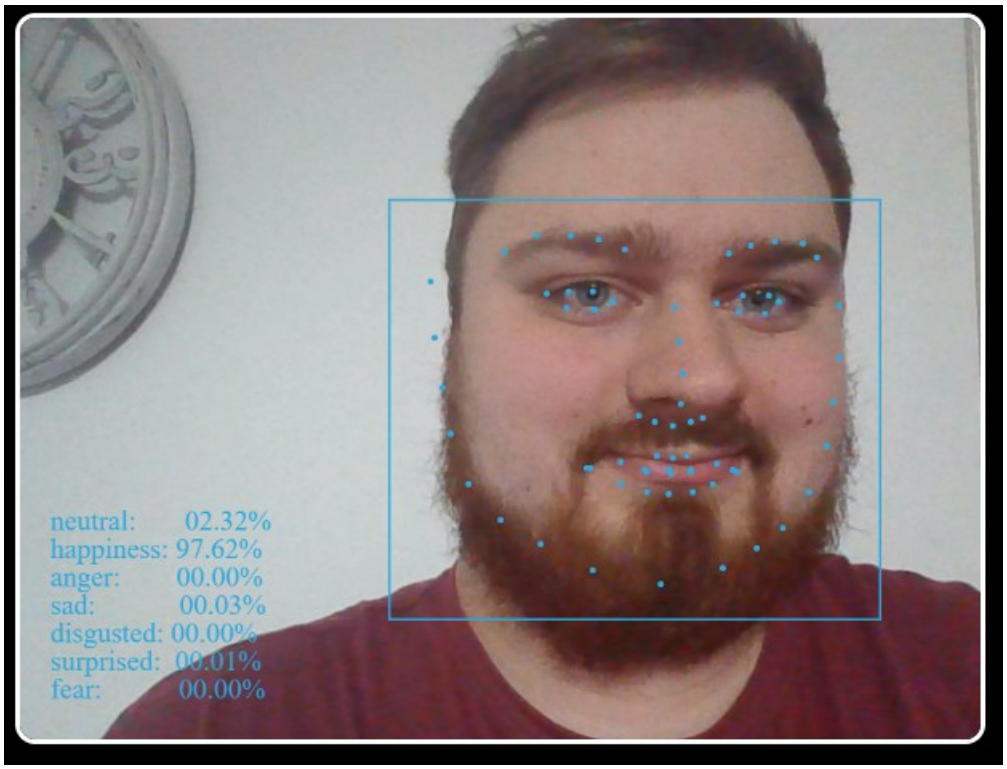

Рисунок 4.4. Приклад розпізнаної емоції «Щастя»

А потім попросити людину здивуватися

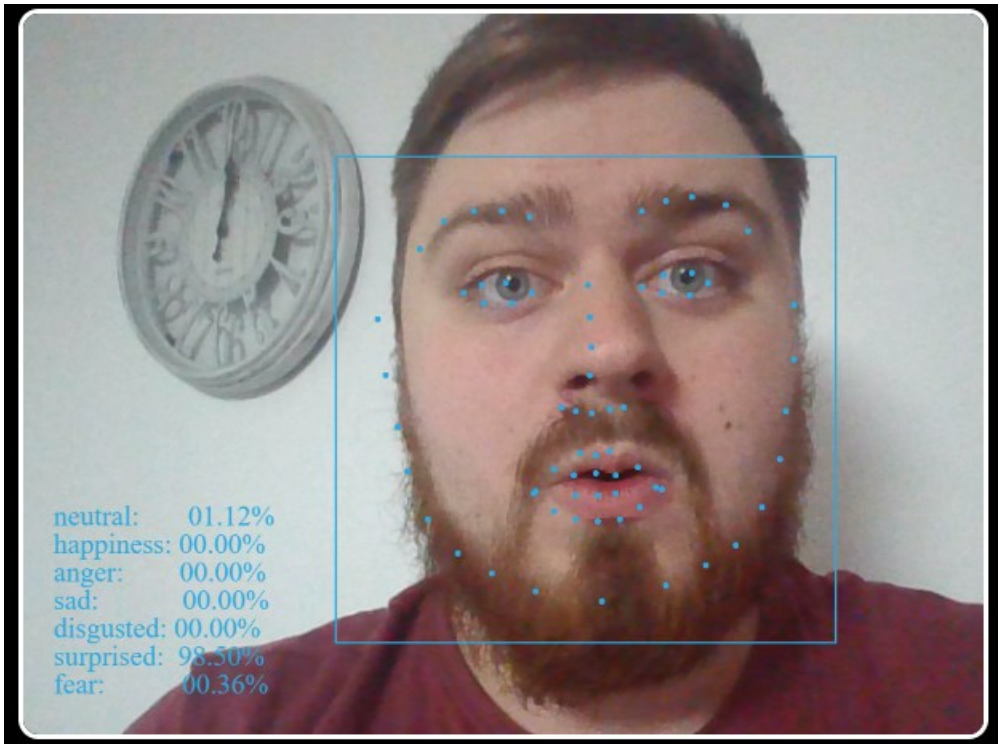

Рисунок 4.5. Приклад розпізнаної емоції «Здивування»

Лінії, крапки, та текст видно лише в режимі відкладки. Як можна помітити, то відсоток проти кожної зображеної емоції більший за 95, тому маркером наявності можна залишити це число.

Після того як людина відобразила 2 емоції ми повинні порівняти ці два зображення між собою, що людина на 2х вото одна й та ж сама. Для прикладу використаємо один із наявниї ендпоінтів що описані вище.

Викличемо метод /compare та побачимо результат:

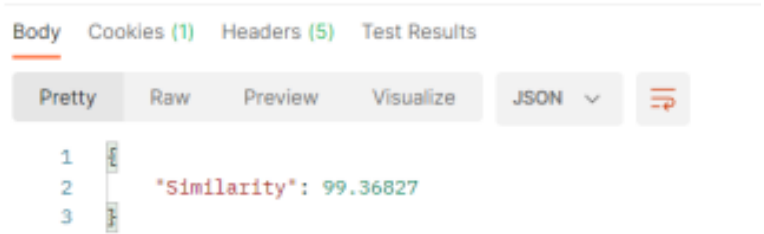

Рисунок 4.6. Результат порівнянна повернутий у вигляді Json об’єкта

За результатом ми бачимо що це однай й таж сама особа, тому ми можемо надсилати це зображення (будь яке з двох) до маршруту що відповідає за авторизацію.

```
function detectSecondRandomExpression(detections, emotion){
  if(detections.length > 0){
    if(detections[0].expressions == emotion){
       if(detections[0].expressions.value * 100 > 97){
          comparePhoto(first, second)
       }
    }
  }
}
```

Логіка авт оризації аналогічна до попереднього кроку. Ми дістаємо кожне зображення з бакету S3 та порівнюємо його з джерелом (клієнтським фото).

Для оптимізації пошуку краще вдосконалити розміщення зображень документів в бакеті, ми можемо їх партиціонувати за певними ознаками:

- Стать
- Колір очей
- Віковий діапазон

Але для таких рішень потрібна більш навчена модель розпізнавання образів, але й ми не можемо виключати що фотографія в документа була зроблена багато років тому, або те що людина наділа кольорові лінзи. Хоча наявна модель що існує в AWS Rekognition досить гарно розпізнає будь які образи не зважаючи на вік, та кольори очей та волосся.

Для прикладу можемо розглянути чорно-біле фото документів та реальну фотографію людини використовуючи не наш API, а функціонал для тестування в AWS.

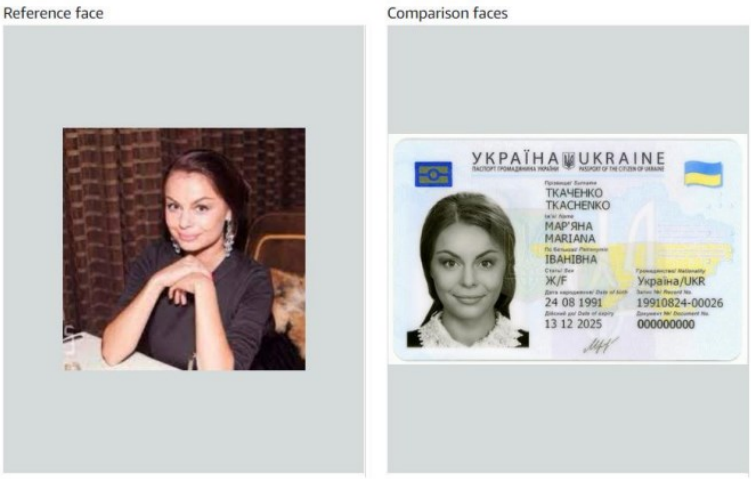

Рисунок 4.7. Приклад порівняння документа та реального зображення

Співпадіння досить високе, більше 99% незважаючи на колір зображення, що прямо впливає на те якого кольору певні частини людини на зображенні.

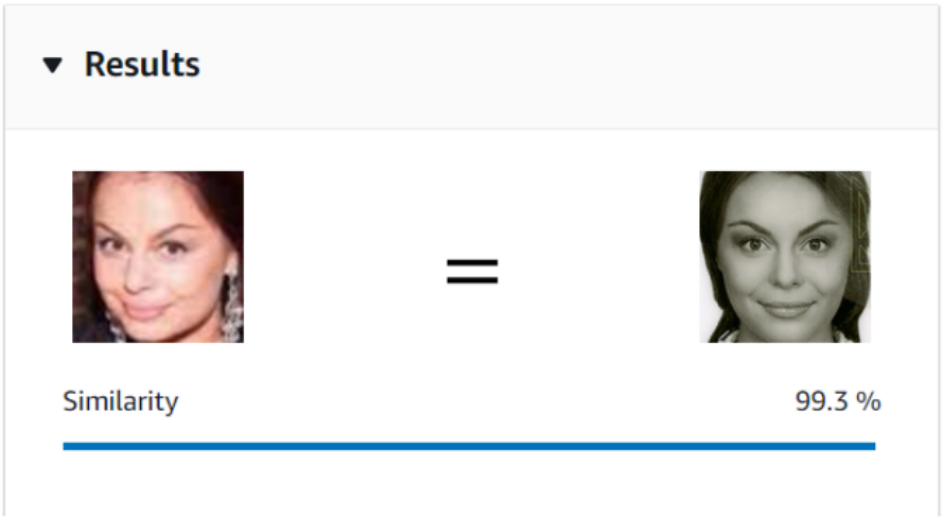

Рисунок 4.8. Результат порівняння

З відкритих даних нам відомо що модель розпізнавання образів незважає на колір, а фактично заміряє абсолютно всі параметри відносного розташування органів голови людини на зображенні, тобто ширину, довжину, відстань від ока до брів, відстань від верхньої губи до носа і таке інше. Також нам відомо, що під час використання – модель навчається, таким чином вона з часом лише вдосконалюється.

Самонавчання моделі (self-training model) стосується методів машинного навчання, де модель початково навчається на малих маркованих наборах даних, після чого використовує цю початкову модель для роботи з немаркованими

даними. Фактично модель навчається зі своїх попередніх передбачень і постійно вдосконалюється для отримання кращих результатів.

Цей підхід використовується в таких задачах, де маркування даних відбувається вручну, займає багато часу або коштів. Самонавчання моделі сприяє маркуванню цих даних з меншими зусиллями та може керувати великими наборами даних без їх повного маркування.

Один з підходів до самонавчання моделі полягає в тому, що модель здійснює передбачення на немаркованих даних, а потім мітки з найвищою впевненістю передбачення використовуються для розширення набору маркованих даних. Таким чином, модель знову навчається на розширених маркованих даних і цей процес повторюється кілька разів для досягнення кращого результату.

В умовах інтеграції данного прототипу верифікації особи за її фото є певні безпекові ризики. Але їх можливо уникнути запровадивши певні безпекові процеси в середині підприємства що обрало даний прототип як основу авторизації. Але все ж таки одну проблему вирішити неможливо, це – людський фактор. Сам «реєстратор» може помилитися та завантажити не ті документи, або зробити виток чутливої інформації.

У випадку якщо безпекові процеси пов’язані з людьми налаштовані добре то залишається декілька варіантів того як нам з технічної сторони створити максимально безпечні умови користування інформаційною системою[12].

4.3.Вимоги до безпеки додатку

Основні вимоги до безпеки з технічної сторони:

1. Використання токенів від AWS ін’єктивно, зберігаючи їх серед змінних середовища, або в конфігураційних файлах, наприклад в application.properties[13].

Рисунок 4.9. Приклад application.properties файлу

За допомогою цього файлу(при використанні Spring Framework), ми можемо неявно викликати ці змінні в коді, ось приклад:

Рисунок 4.10. Використання чутливих даних в коді

2. Використати для кожної ролі окремі «сховища» чутливих даних. Умовно ми можемо ще використати AWS Secrets Manager. Цей сервіс допоможе нам якомога найдініше зберігати дані та обмежити до них доступ за допомогою різних ролей.

Висновки до розділу 4

В цьому розділі було обрано архітектуру додатку, запропоновано методи для реалізації веб-орієнтованої інформаційної системи та розглянуто вимоги до безпеки цього додатку з використанням хмарних технологій.

ВИСНОВОК

У данній роботі було розроблено прототип веб-орієнтованої інформаційної системи для ідентифікації особи за фото на документах з використанням найсучасніших хмарних технологій для розпізнавання образів, тексту на зображеннях.

У ході виконання роботи було розглянуто рівні хмарних сервісів, а також проаналізовано який із хмарних провайдерів краще підходить під задачі розпізнавання та машинного навчання в цілому. Проаналізовано цінову політику, та наявні сервіси, що можуть допомогти максимально ефективно розробити веб-орієнтовану інформаційну систему.

Було запропоновано та обрано найбільш зручні інструменти для розробки інформаційної системи та прототипу. Також було виявлено та запропоновано рішення зі сторони клієнта про найкращий метод перевірки зображення обличча, що зменьшує ризики використання зображень замість реального обличча людини.

Розглянуто варіанти збереження чутливої інформації більш безпечними варіантами. Запропоновано варіанти зберігання даних та розмеження доступу до них.

Данну інформаційну систему можна використовувати як додатковий безпековий етап під час авторизації користувачів в підприємствах, державних установах та інших закладах, де потрібно обмежити доступ до певної інформації. Але є певні вимоги зі сторони державних закладів, щодо політики конфіденційності самих сервісів, проте під час аналізу недаремно був обраний провайдер саме Amazon Web Services.

Для бізнес клієнтів та клієнтів що будуть використовувати дуже часто сервіси цього провайдеру є можливість переписати угоди щодо конфіденційності даних на стільки, що навіть технічна підтримка не отримає можливість

переглядати, вносити виправлення та в цілому допомагати з виявленими проблемами під час використання сервісів.

Основною перевагою є швидка та легка інтеграція до інших веб-орієнтованих інформаційних систем. Можливість розробки за допомогою хмарних сервісів максимально покращує результат та зменьшує фінансові витрати на реалізацію та підтримку функціоналу.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bradski, Gary. Learning OpenCV: Computer Vision with the OpenCV Library / Gary Bradski, Adrian Kaehler. – Newton: O'Reilly Media, 2008. - 580 p.
2. Сігел, Ніколас П. Комп'ютерне зору: алгоритми та додатки / Ніколас П. Сігел. – Дніпро: Інфра-М, 2017. - 400 с.
3. Міністерство внутрішніх справ України. Розпізнавання номерних знаків транспортних засобів [Електронний ресурс]. – Режим доступу: https://mvs.gov.ua/38482-rozpiznavannya-nomernih-znakiv-transportnih-zasobiv
4. Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., and Ghalsasi, A. (2011). Cloud computing - The business perspective. Decision Support Systems, 51(1), 176-189.
5. Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, 25(6), 599-616.
6. Мишкін, Дмитро Валерійович. Основи роботи з Amazon Web Services: практичний підхід / Дмитро Валерійович Мишкін. – Одеса: Астропринт, 2019. - 280 с.
7. Росін, Валерій Ігорович. Amazon Web Services в дії / Валерій Ігорович Росін, Олег Сергійович Медведєв. – Київ: ВК "Навчальна книга – Богдан", 2020. - 480 с.
8. Сміт, Метт. Introduction to Amazon Rekognition / Метт Сміт, Колін Фріз. – Київ: Ліра, 2020. - 180 с.
9. Ґерт, Памела Фокс. Amazon S3: Вивчаємо основи веб-сервісів / Памела Фокс Ґерт. – Київ: Діалектика, 2019. - 250 с.
10. Deitel, Paul. Java How to Program, Early Objects (11th Edition) / Paul Deitel, Harvey Deitel. – Boston: Pearson, 2017. - 1234 p.

11. Sierra, Kathy. Head First Java (2nd Edition) / Kathy Sierra, Bert Bates. – Newton: O'Reilly Media, 2005. - 688 p.
12. Семерей, Віктор Максимович. Spring Boot. Конфігурація і захист додатків / Віктор Максимович Семерей. – Київ: Діалектика, 2021. - 420 с.
13. Вінник, Борис Миколайович. Секрети Java Spring: Безпека і збереження даних / Борис Миколайович Вінник. – Харків: Техніка, 2020. - 300 с.
14. Олифір, Григорій Сергійович. Java Spring: захист даних та криптографічні методи / Григорій Сергійович Олифір. – Львів: Новий Світ, 2019. - 280 с.

Додаток А

ЛІСТИНГ КЛАСІВ ТА ПРОГРАМНИХ ФАЙЛІВ ДЛЯ РЕАЛІЗАЦІЇ

```
package com.faceid;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication; import
org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfigura
tion;

@SpringBootApplication(exclude = { SecurityAutoConfiguration.class }) public
class FaceidApplication {

   public static void main(String[] args) {
      SpringApplication.run(FaceidApplication.class, args);
   } }
package com.faceid.controllers.user;
 import com.faceid.models.CookieManager;
import com.faceid.models.ResponseCreator;
import com.faceid.models.TokenManager;
import org.springframework.core.io.ByteArrayResource;
import org.springframework.core.io.Resource; import
com.faceid.models.entities.User;
import com.faceid.services.interfaces.RequestService; import
com.faceid.services.interfaces.UserService; import
org.springframework.beans.factory.annotation.Autowired; import
org.springframework.http.HttpHeaders; import
org.springframework.http.HttpStatus; import
org.springframework.http.MediaType; import
org.springframework.http.ResponseEntity; import
org.springframework.web.bind.annotation.GetMapping; import
org.springframework.web.bind.annotation.RestController;

import javax.servlet.http.HttpServletRequest; import
java.io.IOException;

@RestController
public class DetailsController {
    @Autowired
    private TokenManager tokenManager;
    @Autowired
    private CookieManager cookieManager;
    @Autowired
    private UserService userService;
    @Autowired
    private RequestService requestService;
    @Autowired
    private ResponseCreator responseCreator;
    @GetMapping("/graph")
    public ResponseEntity graph(HttpServletRequest request) {
User user = userService.findByUsername(
                tokenManager.decodeJWT(cookieManager.getCookieValue(request,
"auth-token"))
                        .getIssuer());
        return ResponseEntity.status(HttpStatus.OK)
.body(requestService.countRequestsAndFindDateByUsernam
e(user.getUsername()));
}
```

```
    @GetMapping("/stats")
    public ResponseEntity stats(HttpServletRequest request) {
User user = userService.findByUsername(
                tokenManager.decodeJWT(cookieManager.getCookieValue(request,
"auth-token"))
                        .getIssuer());

        String roleName = userService.getCurrentRole(user.getId()).getName();
if(requestService.getAmountPerMonth(user.getUsername()) == null) {
return ResponseEntity.status(HttpStatus.OK).body(new
ResponseCreator().getStatsData(roleName,0, null));
        }
        int used = requestService.getAmountPerMonth(user.getUsername());
        return ResponseEntity.status(HttpStatus.OK).body(new
ResponseCreator()
                .getStatsData(roleName, used,
requestService.countRequestsAndFindDateByUsername(user.getUsername()))
        );
    }

    @GetMapping("/download")
    public ResponseEntity<Resource> download(HttpServletRequest request) throws
IOException {
        User user = userService.findByUsername(
                tokenManager.decodeJWT(cookieManager.getCookieValue(request,
"auth-token"))
                        .getIssuer());

        HttpHeaders header = new HttpHeaders();
        header.add(HttpHeaders.CONTENT_DISPOSITION, "attachment;
filename=report.csv");
        header.add("Cache-Control", "no-cache, no-store, must-revalidate");
header.add("Pragma", "no-cache");          header.add("Expires", "0");

        byte[] file =
responseCreator.getFileReport(requestService.countRequestsAndFindDateByUserna
me(user.getUsername()));
        ByteArrayResource resource = new ByteArrayResource(file);
         return ResponseEntity.ok()
.headers(header)
                .contentLength(file.length)

.contentType(MediaType.parseMediaType("application/octetstream"))
                .body(resource);
    } }
package com.faceid.controllers.user;
 import com.faceid.models.TokenManager;
import com.faceid.models.entities.User;
import com.faceid.services.interfaces.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletResponse;
@Controller
public class LoginController {
    @Autowired
    private TokenManager tokenManager;
```

```
    @Autowired
    private UserService userService;

    @GetMapping(path = "/login")
public String login(
            @RequestParam String username,
            @RequestParam String password,
            HttpServletResponse response) {
        User user = userService.findByUsername(username);
if(user !=null){
            if(user.getPassword().equals(password)){
Cookie cookie = new Cookie("auth-token",
tokenManager.createJWT(user.getUsername()));
cookie.setMaxAge(60*60);
response.addCookie(cookie);
            }
}
        return "redirect:/home";
    }
 }
package com.faceid.controllers.user;
 import com.faceid.models.UserValidator; import
com.faceid.models.entities.User; import
com.faceid.services.interfaces.UserService; import
org.springframework.beans.factory.annotation.Autowired; import
org.springframework.stereotype.Controller; import
org.springframework.web.bind.annotation.PostMapping; import
org.springframework.web.bind.annotation.RequestParam;

@Controller
public class RegistrationController {
    @Autowired
    private UserService userService;
    @Autowired
    private UserValidator userValidator;

    @PostMapping(path="/registration")
public String registration(
            @RequestParam String username,
@RequestParam String email,
            @RequestParam String password,
@RequestParam String repassword){
if(!password.equals(repassword)){
            return "redirect:/registration?error=password";
        }else {
            if(userService.findByUsername(username) != null){

if(userService.findByUsername(username).getUsername().equals(username)){
return "redirect:/registration?error=username";
                }else
if(userService.findByUsername(username).getEmail().equals(email)){
return "redirect:/registration?error=email";                    }
            }
            User user = new User();
user.setUsername(username);             user.setEmail(email);
user.setPassword(password);             userService.save(user);
        }
        return "redirect:/";
    }
 }
package com.faceid.controllers.user;
```

```
 import com.faceid.controllers.HomeController;
import com.faceid.models.CookieManager; import
com.faceid.models.TokenManager; import
com.faceid.models.entities.User; import
com.faceid.services.interfaces.UserService; import
org.slf4j.Logger; import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus; import
org.springframework.http.ResponseEntity; import
org.springframework.web.bind.annotation.GetMapping; import
org.springframework.web.bind.annotation.RestController;

import javax.servlet.http.HttpServletRequest;

@RestController
public class TokenController {
    private Logger logger = LoggerFactory.getLogger(HomeController.class);
    @Autowired
    private TokenManager tokenManager;
    @Autowired
    private CookieManager cookieManager;
    @Autowired
    private UserService userService;

    @GetMapping("/token_generate")
    public ResponseEntity generateToken(HttpServletRequest request) {
User user = userService.findByUsername(
                tokenManager.decodeJWT(cookieManager.getCookieValue(request,
"auth-token"))
                .getIssuer());
if(user != null) {
            userService.setUserToken(user);
logger.info(user.toString());
            return
ResponseEntity.status(HttpStatus.OK).body(user.getToken());
        }else {
return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("Unauthorized");
        }

    }
    @GetMapping("/token_verify")
public String verifyToken() {
return "token_verify";
    }
    @GetMapping("/token_get")
public String getToken() {
return "token_get";
    } } package
com.faceid.controllers.validations;  import
com.faceid.models.TokenManager; import
com.faceid.models.data.Router; import
com.faceid.models.entities.Request; import
com.faceid.models.entities.Role; import
com.faceid.models.entities.User;
import com.faceid.services.interfaces.RequestService; import
com.faceid.services.interfaces.UserService; import
org.springframework.beans.factory.annotation.Autowired; import
org.springframework.http.HttpStatus; import
org.springframework.http.ResponseEntity; import
org.springframework.web.bind.annotation.GetMapping; import
```

```
org.springframework.web.bind.annotation.RequestHeader; import
org.springframework.web.bind.annotation.RequestMapping; import
org.springframework.web.bind.annotation.RequestMethod; import
org.springframework.web.bind.annotation.RequestParam; import
org.springframework.web.bind.annotation.RestController; import
org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletResponse;
import java.io.IOException; import
java.time.LocalDate;
import java.time.format.DateTimeFormatter; import
java.util.Map;

@RequestMapping(         value =
"/api",         method =
RequestMethod.GET,          produces =
"application/json"
)
@RestController
public class ValidationApiController {
    @Autowired
    protected Router router;
    @Autowired
    private UserService userService;
    @Autowired
    private RequestService requestService;
    @Autowired
    private TokenManager tokenManager;

    @GetMapping(path = "/validate", produces = "application/json")    public
ResponseEntity validate(@RequestHeader("Authorization") String token,
                                    @RequestParam("file") MultipartFile file,
                                    @RequestParam String firstname,
                                    @RequestParam String lastname,
@RequestParam String sex,
                                    @RequestParam String birthday,
                                    @RequestParam String passportNumber,
@RequestParam String idNumber) throws IOException {
        if (token == null || token.isEmpty()) {
            return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
}
        Request request = new Request();
        String username = userService.findUsernameByToken(token);
if (username == null) {
            return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
        }
        if (file == null || file.isEmpty()) {
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
        }
        Integer amount = requestService.getAmountPerMonth(username);
if (amount == null) {
            amount = 0;
}
        Role role =
userService.getCurrentRole(userService.findByUsername(username).getId());
switch (role.getName()) {             case "basic":                 if
(amount == 500) {                     return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("You have reached the limit
of requests per month");                 }                 break;
case "pro":                 if (amount == 5000) {                     return
```

```
ResponseEntity.status(HttpStatus.FORBIDDEN).body("You have reached the limit
of requests per month");                 }                 break;
case "enterprise":
                if (amount == 100000) {
                    return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("You have reached the limit
of requests per month");                 }                 break;
        }
        request.setUsername(username);

request.setDate(LocalDate.now().format(DateTimeFormatter.ofPattern("dd-
MMMyyyy")));
        requestService.save(request);
        return ResponseEntity.status(HttpStatus.OK)
                .body(router.validate(file, firstname, lastname, sex, birthday,
passportNumber, idNumber));
    }

    @GetMapping(path = "/compare", produces = "application/json")    public
ResponseEntity compare(@RequestHeader("Authorization") String token,
                                @RequestParam("document") MultipartFile doc,
                                @RequestParam("photo") MultipartFile photo)
throws IOException {
        if (token == null || token.isEmpty()) {
            return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
}
        Request request = new Request();
        String username = userService.findUsernameByToken(token);
if (username == null) {
            return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
        }
        if (doc == null || doc.isEmpty() || photo.isEmpty()) {
return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
        }
        Integer amount = requestService.getAmountPerMonth(username);
if (amount == null) {             amount = 0;         }
        Role role =
userService.getCurrentRole(userService.findByUsername(username).getId());
switch (role.getName()) {             case "basic":
                return ResponseEntity.status(HttpStatus.FORBIDDEN).body("You
are now allowed to use photo comparison API.");             case "pro":
if (amount == 5000) {                     return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("You have reached the limit
of requests per month");                 }                 break;
case "enterprise":
                if (amount == 100000) {
                    return
ResponseEntity.status(HttpStatus.FORBIDDEN).body("You have reached the limit
of requests per month");                 }                 break;
        }
        request.setUsername(username);

request.setDate(LocalDate.now().format(DateTimeFormatter.ofPattern("dd-
MMMyyyy")));
        requestService.save(request);
        return ResponseEntity.status(HttpStatus.OK)
.body(router.compare(photo, doc));
    } }
package com.faceid.controllers.validations;

import com.faceid.models.data.Router;
```

```
import com.faceid.services.interfaces.RequestService; import
com.faceid.services.interfaces.UserService; import
org.springframework.beans.factory.annotation.Autowired; import
org.springframework.stereotype.Controller; import
org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;

@Controller
public class ValidationController {
    @Autowired
    private Router router;
    @Autowired
    private UserService userService;
    @Autowired
    private RequestService requestService;

    @RequestMapping(value = "/validate", method = RequestMethod.POST)
public String singleFileUpload(
            @RequestParam("file") MultipartFile file,
            @RequestParam String firstname,
            @RequestParam String lastname,
            @RequestParam String sex,
            @RequestParam String birthday,
            @RequestParam String passportNumber,
            @RequestParam String idNumber,
Model model)             throws IOException
{

        model.addAttribute("users",
                router.validate(file, firstname, lastname, sex, birthday,
passportNumber, idNumber)
        );
        return "result";
    } }
package com.faceid.controllers;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class GeneralController {

    @RequestMapping("/")
public String index() {
        return "redirect:/index.html";
    } }
package com.faceid.controllers;
 import com.faceid.models.CookieManager;
import com.faceid.models.TokenManager;
 import com.faceid.models.entities.Role;
import com.faceid.models.entities.User;
import com.faceid.services.interfaces.UserService;

import org.slf4j.*;
```

```
import org.springframework.ui.Model; import
org.springframework.stereotype.Controller; import
org.springframework.web.bind.annotation.GetMapping; import
org.springframework.beans.factory.annotation.Autowired;

import javax.servlet.http.Cookie; import
javax.servlet.http.HttpServletRequest; import
javax.servlet.http.HttpServletResponse;

@Controller
public class HomeController {

    private Logger logger = LoggerFactory.getLogger(HomeController.class);
    @Autowired
    private TokenManager tokenManager;
    @Autowired
    private UserService userService;
    @Autowired
    private CookieManager cookieManager;

    @GetMapping("/details")
    public String getGraph(Model model, HttpServletRequest req) {
if(isCookieDoesNotExists(req)) {             logger.info("Cookie
does not exists");             logger.info("Redirecting to /");
return "redirect:/";
        }
        logger.info("Cookie exists");
logger.info("Adding token to model");
addModelUser(model, req);
logger.info("Redirecting to /graph");         return
"graph";
    }

    @GetMapping("/token")
    public String getToken(Model model, HttpServletRequest req) {
if(isCookieDoesNotExists(req)) {             logger.info("Cookie
does not exists");             logger.info("Redirecting to /");
return "redirect:/";
        }
        logger.info("Cookie exists");
logger.info("Adding token to model");
addModelUser(model, req);
logger.info("Redirecting to /token");         return
"token";
    }

    @GetMapping("/home")
    public String getHome(Model model, HttpServletRequest req) {
if(isCookieDoesNotExists(req)) {             logger.info("Cookie
does not exists");             logger.info("Redirecting to /");
return "redirect:/";
        }
        logger.info("Cookie exists");
logger.info("Adding token to model");
addModelUser(model, req);
logger.info("Redirecting to /home");         return
"home";
    }

    @GetMapping("/demo")
```

```
    public String getDemo(Model model, HttpServletRequest req) {
if(isCookieDoesNotExists(req)) {             logger.info("Cookie
does not exists");             logger.info("Redirecting to /");
return "redirect:/";
        }
        logger.info("Cookie exists");
logger.info("Adding token to model");
addModelUser(model, req);
logger.info("Redirecting to /demo");         return
"demo";
    }

    @GetMapping("/logout")
    public String getLogout(HttpServletResponse response) {
Cookie cookie = new Cookie("auth-token", null);
cookie.setMaxAge(0);         response.addCookie(cookie);
return "redirect:/";
    }      private void addModelUser(Model model, HttpServletRequest
req) {          User user = userService.findByUsername(
                tokenManager.decodeJWT(cookieManager.getCookieValue(req,
"auth-token")).getIssuer()
        );
        Role role = userService.getCurrentRole(user.getId());
model.addAttribute("user", user);         model.addAttribute("role",
role);
    }
    private   boolean   isCookieDoesNotExists(HttpServletRequest   req)   {
return req.getCookies() == null || req.getCookies().length <= 0;     }
}
package com.faceid.models.converters;

public interface ConvertableDetail {
    String getDate();
int getAmount();
}
package com.faceid.models.data;

public class Generator {
    private static final String AB =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz-_";
private static final int RANDOM_STRING_LENGTH = 30;

    public static String randomString() {
        StringBuilder sb = new StringBuilder(RANDOM_STRING_LENGTH);
for (int i = 0; i < RANDOM_STRING_LENGTH; i++) {             int
index = (int) (AB.length() * Math.random());
sb.append(AB.charAt(index));
        }
        return sb.toString();
    } }
package com.faceid.models.data;

import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.util.IOUtils; import
com.faceid.models.entities.Identity; import
com.faceid.models.enums.Names; import
com.faceid.services.amazon.RecognizeData; import
com.faceid.services.amazon.UploadToS3;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component; import
org.springframework.web.multipart.MultipartFile;
```

```
 import java.io.File; import
java.io.IOException; import
java.nio.ByteBuffer; import
java.nio.file.Files; import
java.nio.file.Path; import
java.nio.file.Paths; import
java.util.HashMap; import
java.util.Map;

@Component
public class Router {
@Autowired
    private Validator validator;
    @Autowired
    private UploadToS3 uploadToS3;
    @Autowired
    private RecognizeData recognizeData;
     public Map<String, Boolean> validate(MultipartFile file,
String firstname,
                                         String lastname,
                                         String sex,
                                         String birthday,
                                         String passportNumber,
                                         String idNumber) throws IOException
{
        byte[] bytes = file.getBytes();
        Path path = Paths.get(file.getOriginalFilename());
        Files.write(path, bytes);
uploadToS3.upload(
Names.BUCKET_NAME.getBucketName(),
file.getOriginalFilename(),                  new
File(file.getOriginalFilename())
        );
        Identity user = new Identity(firstname, lastname, birthday, sex,
passportNumber, idNumber);
        Identity recognized = recognizeData.recognize(
file.getOriginalFilename(),
                Names.BUCKET_NAME.getBucketName()
        );
        return validator.validate(user, recognized);
    }
    public Map<String, Float> compare(MultipartFile photo, MultipartFile doc)
throws IOException {
        ByteBuffer sourceBuffer =
ByteBuffer.wrap(IOUtils.toByteArray(photo.getInputStream()));
        ByteBuffer targetBuffer =
ByteBuffer.wrap(IOUtils.toByteArray(doc.getInputStream()));
        Image source = new Image().withBytes(sourceBuffer);
Image target = new Image().withBytes(targetBuffer);         return
new HashMap<>() {{
            put("Similarity", recognizeData.compare(source, target));
        }};
    } }
package com.faceid.models.data;

import com.amazonaws.services.rekognition.model.Image;
import com.faceid.models.StringFormatter; import
lombok.Data;
import org.springframework.stereotype.Component;

import com.faceid.models.entities.Identity;
```

```
 import java.util.Arrays; import
java.util.HashMap; import
java.util.LinkedHashMap; import
java.util.Map;

@Data @Component
public class Validator {
    private Map<String, Boolean> results = new LinkedHashMap<>();

    public Map<String, Boolean> validate(Identity requested, Identity
recognized){
        results.put("First name",
requested.getFirstName().toUpperCase().equals(recognized.getFirstName()));
results.put("Last name",
requested.getLastName().toUpperCase().equals(recognized.getLastName()));
results.put("Date of birth",
StringFormatter.removeDash(requested.getDate()).equals(recognized.getDate()))
;
        results.put("Sex",
requested.getSex().toUpperCase().equals(recognized.getSex()));
results.put("Identification Code",
isValidIdentificationCode(requested.getIdNumber()));
results.put("Passport number",
requested.getPassportNumber().equals(recognized.getPassportNumber()));
        System.out.println(requested);
System.out.println(recognized);         return
results;
    }

    public boolean isValidIdentificationCode(String code){
int[] numbers =
Arrays.stream(code.split("")).mapToInt(Integer::parseInt).toArray();
int checkSum = numbers[0] * -1 +                numbers[1] * 5 +
numbers[2] * 7 +                numbers[3] * 9 +
numbers[4] * 4 +                numbers[5] * 6 +
numbers[6] * 10 +                numbers[7] * 5 +
numbers[8] * 7;
        System.out.println(checkSum);
int   checkValue   =   checkSum   %   11;
System.out.println(checkValue);
if(checkValue  == 10){          checkValue
= 0;
        }
        return checkValue == numbers[9];
    } }
package com.faceid.models.entities;

import com.faceid.models.converters.ConvertableDetail;

public class Detail implements ConvertableDetail {
private String date;     private int amount;

    public String getDate() {
return date;
    }
    public void setDate(String date) {
this.date = date;
    }      public int
getAmount() {         return
amount;
```

```
    }      public void setAmount(int
amount) {          this.amount = amount;
    }

    @Override
    public String toString() {
return "Detail{" +
                "date='" + date + '\'' +
                ", amount='" + amount + '\'' +
                '}';
    } } package
com.faceid.models.entities;

import lombok.AllArgsConstructor;
import lombok.Data; import
lombok.EqualsAndHashCode; import
lombok.NoArgsConstructor;

@Data
@NoArgsConstructor @AllArgsConstructor
public class Identity {     private
String firstName;     private String
lastName;     private String date;
    @EqualsAndHashCode.Exclude
private String sex;     private
String passportNumber;
    @EqualsAndHashCode.Exclude
private String idNumber;
}
package com.faceid.models.entities;
 import javax.persistence.Entity; import
javax.persistence.GeneratedValue; import
javax.persistence.GenerationType; import
javax.persistence.Id; import
javax.persistence.ManyToOne; import
javax.persistence.Table;

@Entity
@Table(name = "request") public
class Request {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;     private String username;     private
String date;
     public Long getId() {
return id;
    }
    public void setId(Long id) {
this.id = id;
    }      public String
getUsername() {          return
username;
    }
    public void setUsername(String username) {
this.username = username;
    }
    public String getDate() {
return date;
    }
    public void setDate(String date) {
this.date = date;
```

```
    } }
package com.faceid.models.entities;

import javax.persistence.*; import
java.util.Set;

@Entity
@Table(name = "role") public
class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

    private String name;

    @ManyToMany(mappedBy = "roles")
private Set<User> users;
     public Long getId() {
return id;
    }
    public void setId(Long id) {
this.id = id;
    }      public String
getName() {          return
name;
    }
    public void setName(String name) {
this.name = name;
    }
    public Set<User> getUsers() {
return users;
    }
    public void setUsers(Set<User> users) {
this.users = users;
    } }
package com.faceid.models.entities;

import lombok.AllArgsConstructor; import
lombok.Data;
import org.springframework.stereotype.Component;

@Data
@AllArgsConstructor public
class StatsData {
private int limit;
private int used;
private int free;
private int average;
}
package com.faceid.models.entities;
 import javax.persistence.Entity; import
javax.persistence.GeneratedValue; import
javax.persistence.GenerationType; import
javax.persistence.Id; import
javax.persistence.ManyToMany; import
javax.persistence.OneToMany; import
javax.persistence.Table; import
java.util.Set;

@Entity
```

```
@Table(name = "users") public
class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;     private String username;     private
String email;     private String password;
    @ManyToMany
    private Set<Role> roles;
private String token;
public Long getId() {
return id;     }
    public void setId(Long id) {
this.id = id;
    }
    public String getUsername() {
return username;
    }
    public void setUsername(String username) {
this.username = username;
    }
    public String getEmail() {
return email;
    }
    public void setEmail(String email) {
this.email = email;
    }
    public String getPassword() {
return password;
    }
    public void setPassword(String password) {
this.password = password;
    }
    public Set<Role> getRoles() {
return roles;
    }
    public void setRoles(Set<Role> roles) {
this.roles = roles;
    }
    public String getToken() {
return token;
    }
    public void setToken(String token) {
this.token = token;
    }
     @Override     public String
toString() {         return
"User{" +                "id="
+ id +
                ", username='" + username + '\'' +
", email='" + email + '\'' +
                ", password='" + password + '\'' +
                ", roles=" + roles +
                ", token='" + token + '\'' +
                '}';
    }
}
package com.faceid.models.enums;
 import lombok.AllArgsConstructor;
import lombok.Getter;
```

```
@AllArgsConstructor
@Getter public enum
Names {
    BUCKET_NAME("faceid-documents");
private final String bucketName;
}
package com.faceid.models.enums;

import lombok.AllArgsConstructor; import
lombok.Getter;

@AllArgsConstructor
@Getter
public enum Roles {
    BASIC(1),
    PRO(2),
    ENTERPRISE(3),
ADMIN(4);
    private final int role_id;
}
package com.faceid.models;

import com.amazonaws.auth.AWSCredentials; import
com.amazonaws.auth.AWSStaticCredentialsProvider; import
com.amazonaws.auth.BasicAWSCredentials; import
com.amazonaws.services.rekognition.AmazonRekognition; import
com.amazonaws.services.rekognition.AmazonRekognitionClient; import
com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
 import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean; import
org.springframework.context.annotation.Configuration;

@Configuration public class
AmazonConfig {
@Value("${aws.accessKey}")
private String accessKey;
@Value("${aws.secretKey}")
private String secretKey;
     @Bean
    public AmazonS3 s3() {
        AWSCredentials awsCredentials =
                new BasicAWSCredentials(accessKey, secretKey);
System.out.println(awsCredentials.getAWSSecretKey() + " " +
awsCredentials.getAWSAccessKeyId());         return
AmazonS3ClientBuilder
                .standard()
                .withRegion("eu-west-1")
                .withCredentials(new
AWSStaticCredentialsProvider(awsCredentials))
                .build();
    }
    public AmazonRekognition rekognition(){
AWSCredentials awsCredentials =
                new BasicAWSCredentials(accessKey, secretKey);
return AmazonRekognitionClient
                .builder()
                .withRegion("eu-west-1")
                .withCredentials(new
AWSStaticCredentialsProvider(awsCredentials))
                .build();
```

```
    } }
package com.faceid.models;
 import org.springframework.stereotype.Component;
import javax.servlet.http.Cookie; import
javax.servlet.http.HttpServletRequest; import
java.util.Arrays;

@Component
public class CookieManager {
    public String getCookieValue(HttpServletRequest req, String cookieName) {
return Arrays.stream(req.getCookies())
                .filter(c -> c.getName().equals(cookieName))
                .findFirst()
                .map(Cookie::getValue)
.orElse(null);
    } }
package com.faceid.models;

import com.liqpay.LiqPay;
import org.springframework.stereotype.Component;

import java.time.LocalDateTime; import
java.time.format.DateTimeFormatter; import
java.util.HashMap; import java.util.Map;

@Component
public class PaymentCreator {
    public String generatePayment(String username, String role, String
publicKey, String privateKey){
        String date =
LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss"));
        HashMap<String, String> params = new HashMap<>();
params.put("action", "subscribe");         params.put("currency",
"USD");
        params.put("order_id", role + "/" + username);
params.put("version", "3");         params.put("language",
"en");         params.put("public_key", publicKey);
params.put("subscribe_periodicity", "month");
params.put("subscribe_date_start", date);
        params.put("result_url", "http://localhost:8080/successfully/" + role);
        if(role.equals("pro")){
params.put("amount", "150");
            params.put("description", "Upgrading to status PRO");
        }else if(role.equals("enterprise")){
params.put("amount", "750");
            params.put("description", "Upgrading to status ENTERPRISE");
}
        LiqPay liqpay = new LiqPay(publicKey, privateKey);
return liqpay.cnb_form(params);
    }      public Map<String, Object> receivePaymentStatus(String role,
String username, String publicKey, String privateKey) throws Exception {
HashMap<String, String> params = new HashMap<>();
params.put("action", "status");         params.put("version", "3");
        params.put("order_id", role + "/" + username);

        LiqPay liqpay = new LiqPay(publicKey, privateKey);
return liqpay.api("request", params);     }
}
package com.faceid.models;
```

```
import com.faceid.models.converters.ConvertableDetail;
import com.faceid.models.entities.StatsData; import
org.springframework.stereotype.Component;

import java.io.BufferedWriter;
import java.io.File; import
java.io.FileWriter; import
java.io.IOException; import
java.io.PrintWriter; import
java.nio.file.Files; import
java.util.List;

@Component
public class ResponseCreator {
    public StatsData getStatsData(String role, int used,
List<ConvertableDetail> values) {
int limit = 0;          switch
(role) {              case "basic":
limit = 500;                  break;
case "pro":                  limit =
5000;                 break;
case "enterprise":
limit = 100000;
break;              default:
break;
        }
        if(values != null)
        {
            int average =
values.stream().mapToInt(ConvertableDetail::getAmount).sum() / values.size();
return new StatsData(limit, used, limit-used, average);
        }
        return new StatsData(limit, used, limit-used, 0);
    }
    public byte[] getFileReport(List<ConvertableDetail> details) throws
IOException {
        File file = new File("report.csv");
try(FileWriter fw = new FileWriter(file)) {
fw.write("Date,Amount\n");
details.forEach(detail -> {                 try {
                    fw.write(detail.getDate() + "," + detail.getAmount() +
"\n");
                } catch (IOException e) {
throw new RuntimeException(e);
                }
            });
}
        return Files.readAllBytes(file.toPath());
} } package com.faceid.models;

import org.springframework.stereotype.Component;
@Component
public class StringFormatter {
    public static String removeDash(String string){
        return string.replace("-"," ");
    } }
package com.faceid.models;
 import io.jsonwebtoken.Claims;
import io.jsonwebtoken.JwtBuilder;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
```

```
import org.springframework.beans.factory.annotation.Value; import
org.springframework.stereotype.Component;
 import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;
import java.security.Key;

@Component
public class TokenManager {
@Value("${jwt.secret}")     private
String secret;

    public String createJWT(String issuer) {
        SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;

        byte[] apiKeySecretBytes =
DatatypeConverter.parseBase64Binary(secret);
        Key signingKey = new SecretKeySpec(apiKeySecretBytes,
signatureAlgorithm.getJcaName());

        JwtBuilder builder = Jwts.builder()
                .setIssuer(issuer)
                .signWith(signatureAlgorithm, signingKey);
return builder.compact();
    }
    public Claims decodeJWT(String jwt) {
return Jwts.parser()
                .setSigningKey(DatatypeConverter.parseBase64Binary(secret))
                .parseClaimsJws(jwt)
.getBody();
    } }
package com.faceid.models;

import com.faceid.models.entities.User; import
com.faceid.services.interfaces.UserService; import
org.springframework.beans.factory.annotation.Autowired; import
org.springframework.stereotype.Component; import
org.springframework.validation.Errors; import
org.springframework.validation.ValidationUtils; import
org.springframework.validation.Validator;

@Component
public class UserValidator implements Validator {
    @Autowired
    private UserService userService;
    @Override
    public boolean supports(Class<?> aClass) {
return User.class.equals(aClass);
    }
@Override
    public void validate(Object o, Errors errors) {
        User user = (User) o;
ValidationUtils.rejectIfEmptyOrWhitespace(errors, "username",
"NotEmpty");
        if (user.getUsername().length() < 6 || user.getUsername().length() >
32) {
            errors.rejectValue("username", "Size.userForm.username");
        }
        if (userService.findByUsername(user.getUsername()) != null) {
errors.rejectValue("username", "Duplicate.userForm.username");
        }
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "password",
```

```
"NotEmpty");
        if (user.getPassword().length() < 8 || user.getPassword().length() >
32) {
            errors.rejectValue("password", "Size.userForm.password");
        }
    } }
package com.faceid.repositories;

import com.faceid.models.converters.ConvertableDetail; import
com.faceid.models.entities.Request;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query; import
org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Repository
public interface RequestRepository extends JpaRepository<Request, Long> {
    List<Request> findAllByUsername(String username);
    List<Request> findAllByDate(String date);
    List<Request> findAllByUsernameAndDate(String username, String date);
    @Query(value = "SELECT date, COUNT(date) as amount from request where
username = ? group by date", nativeQuery = true)
    List<ConvertableDetail> countRequestsAndFindDateByUsername(String
username);

    @Query(value = "SELECT COUNT(id) AS amount FROM request where username =
? and DATE_TRUNC('month', date\\:\\:date) = DATE_TRUNC('month', CURRENT_DATE)
GROUP BY DATE_TRUNC('month', date\\:\\:date)", nativeQuery = true)
    Integer amountPerCurrentMonth(String username);
}
package com.faceid.repositories;

import com.faceid.models.entities.Role;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query; import
org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
@Repository
public interface RoleRepository extends JpaRepository<Role, Long>{
    Role findRoleById(Long id);

    @Transactional
    @Modifying
    @Query(value = "INSERT INTO users_roles(users_id, roles_id) values
(?1,?2)", nativeQuery = true)
    void insertRole(long userId, int roleId);
@Transactional
    @Modifying
    @Query(value = "UPDATE users_roles SET roles_id = ?2 where users_id =
?1", nativeQuery = true)
    void updateRoleForUser(long userId, int roleId);

    @Query(value = "SELECT * from role where id = (SELECT roles_id FROM
users_roles WHERE users_id = ?)", nativeQuery = true)
    Role getRoleByUserId(long userId);
}
```

```
package com.faceid.repositories;

import com.faceid.models.converters.ConvertableDetail; import
com.faceid.models.entities.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query; import
org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    User findByUsername(String username);
    User findByToken(String token);
    @Query(value = "SELECT date, COUNT(date) as amount from request where
username = ? group by date", nativeQuery = true)
    List<ConvertableDetail>    countRequestsAndFindDateByUsername(String
username); }
package com.faceid.services.amazon;

import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.CompareFacesMatch; import
com.amazonaws.services.rekognition.model.CompareFacesRequest; import
com.amazonaws.services.rekognition.model.DetectTextRequest; import
com.amazonaws.services.rekognition.model.DetectTextResult; import
com.amazonaws.services.rekognition.model.Image; import
com.amazonaws.services.rekognition.model.S3Object; import
com.amazonaws.services.rekognition.model.TextDetection; import
com.faceid.models.AmazonConfig; import com.faceid.models.entities.Identity;
import org.apache.commons.text.StringEscapeUtils; import
org.springframework.beans.factory.annotation.Autowired; import
org.springframework.stereotype.Service;

import java.util.List;
 @Service
public class RecognizeData {
    @Autowired
    private AmazonConfig amazonConfig;

    public Identity recognize(String name, String bucket) {
        Identity identity = new Identity();
        DetectTextRequest request = new DetectTextRequest()
                .withImage(new Image()
                        .withS3Object(new S3Object()
                                .withName(name)
.withBucket(bucket)));          try {
            DetectTextResult result =
amazonConfig.rekognition().detectText(request);
            List<TextDetection>  textDetections  =  result.getTextDetections();
System.out.println("Detected    lines    and    words    for    "    +    name);
identity.setLastName(textDetections.get(5).getDetectedText());
identity.setFirstName(textDetections.get(8).getDetectedText());
identity.setDate(textDetections.get(19).getDetectedText());

identity.setPassportNumber(textDetections.get(24).getDetectedText());

identity.setSex(textDetections.get(15).getDetectedText().replace("/", ""));
} catch (AmazonRekognitionException e) {
            e.printStackTrace();
}
        return identity;
```

```
    }
    public float compare(Image source, Image target){
        CompareFacesRequest request = new
CompareFacesRequest().withSourceImage(source).withTargetImage(target);
List<CompareFacesMatch> faces =
amazonConfig.rekognition().compareFaces(request).getFaceMatches();
if(faces.size() == 0){              return 0;
        }
        return faces.get(0).getSimilarity();
    } }
package com.faceid.services.amazon;
 import com.amazonaws.AmazonServiceException;
import com.faceid.models.AmazonConfig; import
lombok.AllArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.io.File;

@AllArgsConstructor
@Service
public class UploadToS3 {
    @Autowired
    private final AmazonConfig amazonS3;

    public void upload(String path, String fileName, File file) {
try {
            amazonS3.s3().putObject(path, fileName, file);
        } catch (AmazonServiceException e) {
            throw new IllegalStateException("Failed to upload the file", e);
        }
    } } package
com.faceid.services.interfaces;

import com.faceid.models.converters.ConvertableDetail; import
com.faceid.models.entities.Request;
 import java.util.List;
 public interface RequestService {
void save(Request request);
    Iterable<Request> findAllRequestsByUsernameAndDate(String username,
String date);
    Iterable<Request> findAll();
    Iterable<Request> findAllByUsername(String username);
    List<ConvertableDetail> countRequestsAndFindDateByUsername(String
username);
    Integer getAmountPerMonth(String username);
}
package com.faceid.services.interfaces;
 import com.faceid.models.entities.Role;
import com.faceid.models.entities.User;
 public interface UserService {
void save(User user);
    User findByUsername(String username);
    String findTokenByUsername(String username);
    String findUsernameByToken(String token);
    User setUserToken(User user);
    Role getCurrentRole(long userId);
    void updateRoleForUser(long userId, int roleId);
}
package com.faceid.services;
```

```
import com.faceid.models.converters.ConvertableDetail; import
com.faceid.models.entities.Request; import
com.faceid.repositories.RequestRepository; import
com.faceid.services.interfaces.RequestService; import
org.springframework.beans.factory.annotation.Autowired; import
org.springframework.stereotype.Service;

import java.util.List;

 @Service
public class RequestServiceImpl implements RequestService {
    @Autowired
    private RequestRepository requestRepository;

    @Override
    public void save(Request request) {
requestRepository.save(request);
    }

    @Override
    public Iterable<Request> findAllRequestsByUsernameAndDate(String username,
String date) {
        return requestRepository.findAllByUsernameAndDate(username, date);
}

    @Override
    public Iterable<Request> findAll() {
return requestRepository.findAll();
    }

    @Override
    public Iterable<Request> findAllByUsername(String username) {
return requestRepository.findAllByUsername(username);
    }

    @Override
    public List<ConvertableDetail> countRequestsAndFindDateByUsername(String
username) {          return
requestRepository.countRequestsAndFindDateByUsername(username);
    }

    @Override
    public Integer getAmountPerMonth(String username) {
return requestRepository.amountPerCurrentMonth(username);     } }
package com.faceid.services;
 import com.faceid.repositories.UserRepository;
import com.faceid.models.entities.Role; import
com.faceid.models.entities.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails; import
org.springframework.security.core.userdetails.UserDetailsService; import
org.springframework.security.core.userdetails.UsernameNotFoundException; import
org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.util.HashSet; import java.util.Set;
@Service
public class UserDetailsServiceImpl implements UserDetailsService{
    @Autowired
```

```
    private UserRepository userRepository;
    @Override
    @Transactional(readOnly = true)
    public UserDetails loadUserByUsername(String username) {
User user = userRepository.findByUsername(username);
        if (user == null) throw new UsernameNotFoundException(username);
Set<GrantedAuthority> grantedAuthorities = new HashSet<>();         for
(Role role : user.getRoles()){              grantedAuthorities.add(new
SimpleGrantedAuthority(role.getName()));
        }
return new
org.springframework.security.core.userdetails.User(user.getUsername(),
user.getPassword(), grantedAuthorities);
    } }


package com.faceid.services;
 import com.faceid.models.entities.Role; import
com.faceid.models.enums.Roles; import
com.faceid.repositories.RoleRepository; import
com.faceid.repositories.UserRepository; import
com.faceid.models.entities.User; import
com.faceid.services.interfaces.UserService; import
com.faceid.models.data.Generator;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;  @Service
public class UserServiceImpl implements UserService {
    @Autowired
    private UserRepository userRepository;
    @Autowired
    private RoleRepository roleRepository;

    @Override

    public void save(User user) {
        //todo: encrypt password before saving
        //user.setRoles(new HashSet<>(roleRepository.findAll()));
userRepository.save(user);

roleRepository.insertRole(userRepository.findByUsername(user.getUsername()).g
etId(), Roles.BASIC.getRole_id());
    }
@Override
    public User findByUsername(String username) {
return userRepository.findByUsername(username);
    }

    @Override
    public String findTokenByUsername(String username) {
return userRepository.findByUsername(username).getToken();
    }

    @Override
    public User setUserToken(User user) {
user.setToken(Generator.randomString());         userRepository.save(user);
        return userRepository.findByUsername(user.getUsername());
    }

    @Override
```

```
    public String findUsernameByToken(String token) {
return userRepository.findByToken(token).getUsername();
    }

    @Override
    public Role getCurrentRole(long userId) {
return roleRepository.getRoleByUserId(userId);
    }

    @Override
    public void updateRoleForUser(long userId, int roleId) {
roleRepository.updateRoleForUser(userId, roleId);
    }
}
```

```
let faceapi;
let detections = [];

let video;
let canvas;

function setup() {
  canvas = createCanvas(480, 360);
  canvas.id("canvas");

  video = createCapture(VIDEO);
  video.id("video");
  video.size(width, height);

  const faceOptions = {
    withLandmarks: true,
    withExpressions: true,
    withDescriptors: true,
    minConfidence: 0.5
  };

  faceapi = ml5.faceApi(video, faceOptions, faceReady);
}

function faceReady() {
  faceapi.detect(gotFaces);
}

function gotFaces(error, result) {
  if (error) {
    console.log(error);
    return;
  }

  detections = result;

  clear();
  drawBoxs(detections);
  drawLandmarks(detections);
```

```
  drawExpressions(detections, 20, 250, 14);

  faceapi.detect(gotFaces);
}

function drawBoxs(detections){
  if (detections.length > 0) {
    for (f=0; f < detections.length; f++){
      let {_x, _y, _width, _height} = detections[f].alignedRect._box;
      stroke(44, 169, 225);
      strokeWeight(1);
      noFill();
      rect(_x, _y, _width, _height);
    }
  }
}

function drawLandmarks(detections){
  if (detections.length > 0) {
    for (f=0; f < detections.length; f++){
      let points = detections[f].landmarks.positions;
      for (let i = 0; i < points.length; i++) {
        stroke(44, 169, 225);
        strokeWeight(3);
        point(points[i]._x, points[i]._y);
      }
    }
  }
}

function detectSecondRandomExpression(detections, emotion){
  if(detections.length > 0){
    if(detections[0].expressions == emotion){
        if(detections[0].expressions.value * 100 > 97){
            comparePhoto(first, second)
        }
    }
  }
}
body {
    background-color: #000;
  }

  #canvas{
    position: relative;
    top: 10px;
    left: 0px;
    z-index:1;
  }

  #video{
```

```
  position: absolute;
  top: 10px;
  left: 10px;
  z-index:0;
  border: 3px #fff solid;
    border-radius: 10px;
}
```