МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

Кваліфікаційна робота бакалавра

ІНТЕЛЕКТУАЛЬНА ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ ФОТО ВЕРИФІКАЦІЇ ОСОБИ З ВИКОРИСТАННЯМ ХМАРНИХ ТЕХНОЛОГІЙ

Здобувач освіти гр. ІН-81 Євгеній ЧИКАЛОВ

Науковий керівник,
кандидат технічних наук, Олег БЕРЕСТ
старший викладач

Завідувач кафедри Анатолій ДОВБИШ
доктор технічних наук,професор.

Суми 2022

РЕФЕРАТ

Записка: 31 стор., 22 рис., 2 табл., 2 додатки, 21 джерел.

Об’єкт дослідження (розробки) — інтелектуальна інформаційна система для фото верифікації особи.

Мета роботи — аналіз (актуальності, аналогів), проектування, прототипування та розробка інтелектуальної інформаційної системи для фото верифікації особи.

Методи дослідження (розробки) — розробка інтелектуальної системи у вигляді WEB-додатку з використанням таких технологій Java (Spring, Thymeleaf), PostgreSQL для серверної частини та HTML, CSS, Javascript (JQuery) для клієнтської частини.

Результати — на основі попередньо проведених аналізу та проектування, розроблений WEB-додаток у якості прототипу із використанням вищезазначених технологій. Було проаналізовано аналоги та актуальність. Було виявлено, що системи з подібним використанням технологій існують для верифікації особи, але конкретного доступного рішення у вигляді WEB-додатку з можливістю інтеграції для потеб власного підприємства, що включає в себе функціонал верифікації особи за її персональною фотокарткою та фото/сканом паспорту громадянина України і перевірки поточних даних, що відображені на фото/скані цього документу.

ІНТЕЛЕКТУВАЛЬНА ІНФОРМАЦІЙНА СИСТЕМА, ХМАРНІ ТЕХНОЛОГІЇ, AMAZON WEB SERVICES, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, РОЗПІЗНАВАННЯ ТЕКСТУ, РОЗПІЗНАВАННЯ ОСОБИ, ВЕРИФІКАЦІЯ, JAVA, POSTGRES, JAVASCRIPT, SPRING

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____
Зав. кафедри Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ

до кваліфікаційної роботи бакалавра

Студента 4-го курсу, групи ІН-81 спеціальності 122 -Комп'ютерні науки, денної форми навчання Чикалова Є.А.

Тема: «Інтелектуальна інформаційна система для фото верифікації особи з використанням хмарних технологій»

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) актуальність; 2) огляд аналогів; 3) вибір технологій; 4) дизайн інтелектуальної системи; 5) основна частина; 6) висновки.

Дата видачі завдання "_____"_____2022 р.
Керівник випускної роботи _____ Берест О.Б.
Завдання прийняв до виконання _____ Чикалов Є.А.

ЗМІСТ

ВСТУП

Верифікація особи за її паспортними даними є важливою та необхідною частиною при перевірці персональних даних, що надає особа певній установі, органу влади, підприємству, тощо [1].

В більшості випадків коли необхідна перевірка наданої інформації особою про себе проходить перевірку у ручному режимі. Тобто маємо людину, яка перевіряє на відповідність внесені дані, з фактичними, що відображені у особистих документах особи, що їх надає. При такому підході маємо досить ефективне розпізнавання тексту на документах, але при винесенні висновків, про таке розпізнавання маємо великі ризики саме помилки з боку людини, що виконує перевірку. Окрім ризиків помилки маємо ще питання витраченого часу на виконання таких дій.

Подана робота присвячена створенню інтелектуальної системи, що дозволяє її користувачеві, проводити такі дії з перевірки особи за її документами, з використанням алгоритмів машинного навчання та хмарних технологій для цілей, описаних вище. У роботі запропоновано використання створеної інформаційної системи для випадку, коли особа надає свій особистий паспорт громадянина України та свої дані персональні дані.

Подібні рішення для верифікації, що базується на відповідності особи її паспортним даним наявні, але не мають широкого застосування, через створення таких під свої власні локальні потреби, що орієнтуються лише на частину з вирішеної проблематики, якій була присвячена подана робота.

РОЗДІЛ 1 АКТУАЛЬНІСТЬ

У сучасному світі неможливо обійтися без використання хмарних технологій. Ми так чи інакше не можемо уникнути їх в повсякденному житті. Більшість веб, десктоп там мобільних додатків все одно їх використовують як у якості баз даних, хмарних сховищ та інструментів для обчислень, тощо [2].

Описана у роботі інтелектуальна система може бути впроваджена в будь якій сфері, де є необхідність перевірки введених особистих даних особи. До таких сфер належать, банківська, страхова, сфера бізнесу, тощо.

Якщо впровадити цю інтелектуальну інформаційну систему для фото верифікації особи на підприємстві то ми замінемо ручний труд людини коли вона самостійно, візуально звіряє актуальність інформації поданої клієнтом та зменшимо можливість впливу людського фактору на результати перевірки.

Використовуючи дану систему у якості безпекового етапу у фінансових установах ми також виграємо в якості поданої інформації, бо можливо звірити обличчя людини що подала заявлений ідентифікаційний документ (наприклад, id картку), що виключає подання документів третьою особою, але за умови правильно налаштованої бізнес логіки безпекового етапу на самому підприємстві [3].

Проаналізувавши наявний ринок України, можна стверджувати, що дана інформаційна система є доволі актуальної та дуже легкою в інтеграції та використанні на підприємствах будь-яких сфер.

РОЗДІЛ 2 ОГЛЯД АНАЛОГІВ

На сьогоднішній день майже кожна людина стикається з інтелектуальними інформаційними системами для верифікації її особи, наприклад при використання додатку, призначеному для електронного документообігу, що може частково або повністю заміти громадянину України паперовий варіант його документів, необхідно пройти верифікацію, для доказу, що доступ до особистих документів є санкціонованим [4].

Подібні інтелектуальні системи ми можемо спостерігати при використання мобільних додатків, наприклад при багаторазовому неправильно введеному паролі до особистого акаунту, додаток може запросити пройти верифікацію, де буде звірено обличчя користувача, який намагається потрапити у особистий кабінет рахунку з його паспортними даними.

У подібних системах є порівняння введених даних, що потрапляють до системи вводу у графічному вигляді (обличчя особи), потім відбувається порівняння з фото із офіційних документів.

Такі рішення розробляються окремою компаніями під свої власні потреби, та не мають широкого розповсюдження.

РОЗДІЛ 3 ВИБІР ТЕХНОЛОГІЙ

3.1 Мова програмування

В якості основної мови програмування було обрано мову Java. На сьогоднішній день створена у 1996 році компанією Sun Microsystems, що була згодом поглинута корпорацією Oracle, мова програмування Java, є однією з найпопулярніших мов програмування.

Мова програмування Java застосовується для створення програмного забезпечення для великого списку пристроїв [5]. Так, наприклад, для створення десктпопних програм, для смартфонів та мобільних телефонів, побутової техніки, тощо можна використовувати мову Java.

Java є об'єктно орієнтованою, строго типізованою мовою програмування для загального призначення. Програмне забезпечення, що було створене на мові Java, спочатку транслюється у байт код, це дає змогу програмі працювати на будь-якій архітектурі, залежно від типу якої створена окрема JVM. Після трансляції у байт код JVM виконує цей код. Фактично JVM є програмою, що обробляє байт код та передає подальші інструкції призначуваному обладнанню у якості інтерпретатора [6].

Для розробки клієнтської частини ми використовуємо такі технології як: HTML, CSS, JavaScript та бібліотеку jQuery.

Для опису бізнес логіки інтелектуальної інформаційної системи використали Java з фреймворками Spring, Hibernate та PostgreSQL у якості СУБД, а також додаткові бібліотеки для роботи з хмарним сервісом Amazon.

3.2 Хмарні технології

3.2.1 Загальні положення

Сучасну IT - сферу не можливо уявити без використання хмарних технологій. Самі хмарні технології почали свій розвиток ще з 1950-х років. Ситуація неможливості придбання комп'ютерної техніки високої вартості для великої кількості співробітників спонукала до створення рішень, коли

декілька користувачів можуть одночасно підключатися до одного процесору. Свій вклад у розвиток хмарних технологій був внесений розвитком мережі інтернет, так як принцип, що лежить у основі концепції хмарних технологій полягає саме у можливості користувача мати доступ до сервісу з будь-якої точки світу.

На сьогоднішній день хмарні технології дуже широко застосовані у різних сферах діяльності. А саме у освіті, медицині, банківській сфері, бізнесі, торгівлі, логістиці, тощо. Саме хмарні технології можуть надати бізнесу більш високу обчислювальну потужність та сучасні сервіси для обробки великої кількості даних, а також можливість ефективного масштабування відповідно до збільшення об’єму даних. Перелік доступних послуг від компаній, що надають доступ до своїх сервісів мають майже безкінечний перелік, котрий постійно доповнюється. На сьогоднішній день хмарні технологій включають сервери, сховища даних, бази даних, мережі, різного роду програмне забезпечення, системи для контролю та моніторингу системи, тощо.

3.2.2 Рівні хмарних технологій

Хмарні технології для користувача надаються користувачеві у вигляді звичайного онлайн сервісу. Всього виділяють три рівні хмарних технологій - SaaS, PaaS, IaaS. В загальному випадку їх можна називати моделями надання хмарних сервісів, які собою являють хмарну піраміду, відповідно до різного рівня контролю інформацією. На вершині піраміди завжди буд знаходитися користувач сервісом, що працює з деяким набором даних, через певне програмне забезпечення, що може мати зручний інтерфейс. У випадку, коли це програмне забезпечення, буде розгорнуте на якійсь платформі – можемо говорити вже про другий рівень піраміди хмарних сервісів. Коли для розгортання маємо певні віртуальні сервери, обчислювальну потужність та накопичувачі – маємо справу з третім рівнем, а саме з основою піраміди хмарних сервісів [7].

З розвитком мережі інтернет широке розповсюдження почали отримувати хмарні сервіси типу SaaS - Software as a Service, що в перекладі значить «програмне забезпечення як послуга», це дало можливість розмежування поняття онлайн-сервісів від десктопних, тобто тих, що треба встановлювати на персональний комп'ютер.

Коли у динамічно розвиваючогося технологічного світу з'явилася потреба швидкого розміщення нового програмного забезпечення - народилася ідея PaaS – Platform as a Service, що в перекладі «Платформа як сервіс». Народження нової концепції хмарних технологій з часом призвело до ситуації, коли великі компанії мали велику кількість невикористаних обчислювальних потужностей. Продаж нереалізованих ресурсів породив новий рівень хмарних технологій, а саме IaaS – Infrastructure as a Service, що в перекладі значить «Інфраструктура як послуга» [8].

Часто схематично сервіси зображують як на рис.3.1

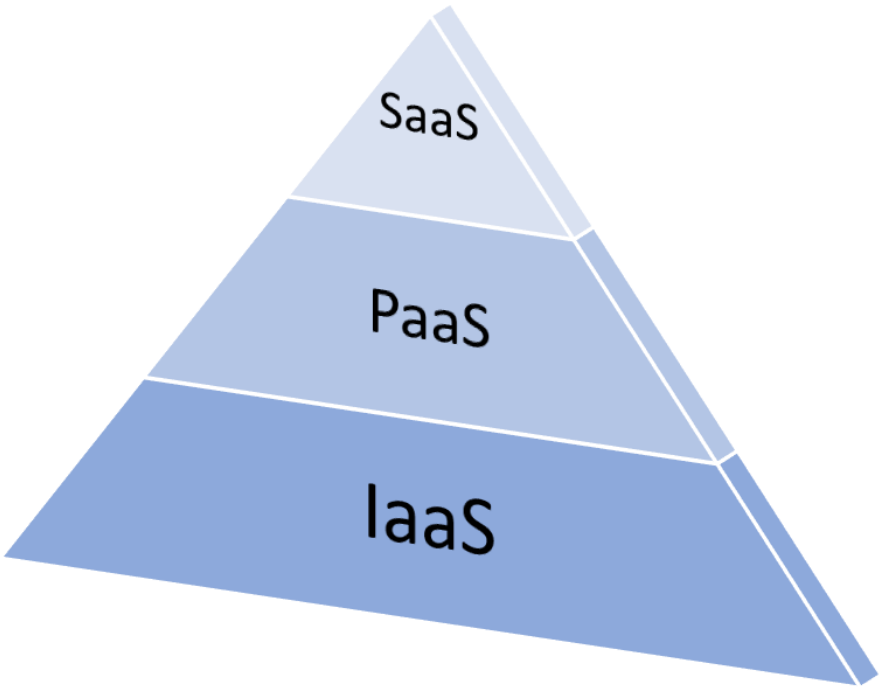

Рис.3.1 – Піраміда моделей надання послуг хмарними технологіями.

3.2.3 Різновид сервісів

На сьогоднішній день багато компаній надають свої послуги як хмарні технології, але до лідерів відносяться: Amazon Web Services, Azure, Google Cloud [9]. У таблиці наведені основні категорії сервісів, що надають лідери рисунку хмарних технологій.

Таблиця 3.1

Категорії сервісів, що надають найпотужніші постачальники хмарних технологій.

Платформи	Категорії сервісів
AWS	Compute; EC2 Instance Types; Storage; Database; Networking & Content Delivery; Migration & Transfer; Developer Tools; Management & Governance; Security, Identity & Compliance; Analytics; Machine Learning; Robotics; Mobile; Application Integration; Media Services; Business Application.; End User Computing; Internet of Things; Game Tech; General AWS; AR & VR.; AWS Cost Management; Blockchain; Customer Engagement; Satellite.
Azure	DevOps; Analytics; Database; Safety; Windows Virtual Desktop; Computing environment; Hybrid and multi-cloud environment; AI + machine learning; Integration; Internet; Internet of Things; Containers; Migration; Mobile applications; Multimedia; Networking; Mixed reality; Developer Tools; Certificate; Control; Storage
Google Cloud	API Platform & Ecosystems; Big Data; Cloud AI.; Compute; Data Transfer; Developer Tools; Identity & Security; Internet of Things; Management Tools; Networking; Storage & Databases.

Як бачимо, в цілому різні платформи надають схожі категорії сервісів, але варто зауважити, що AWS в цьому напрямку залишається лідером. Саме з використанням AWS було створено описану у роботі інтелектуальну систему. Для її створення було використано як центральний сервіс Amazon Rekognition, та як допоміжний Amazon S3.

РОЗДІЛ 4 ДИЗАЙН ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ

4.1 Entity Relationship Diagram

Entity Relationship Diagram (ERD) – це діаграма відношень (зв’язків), яка використовується для проектування бази даних (архітектури). ER діаграма складається за багатьох умовних позначок, що допомагають візуалізувати дві важливі особливості: взаємовідносини між об’єктами, та самі об’єкти [10].

Проаналізувавши проектування баз даних інформаційних інтелектуальних систем було створено діаграму для нашого додатку як на рис.4.1

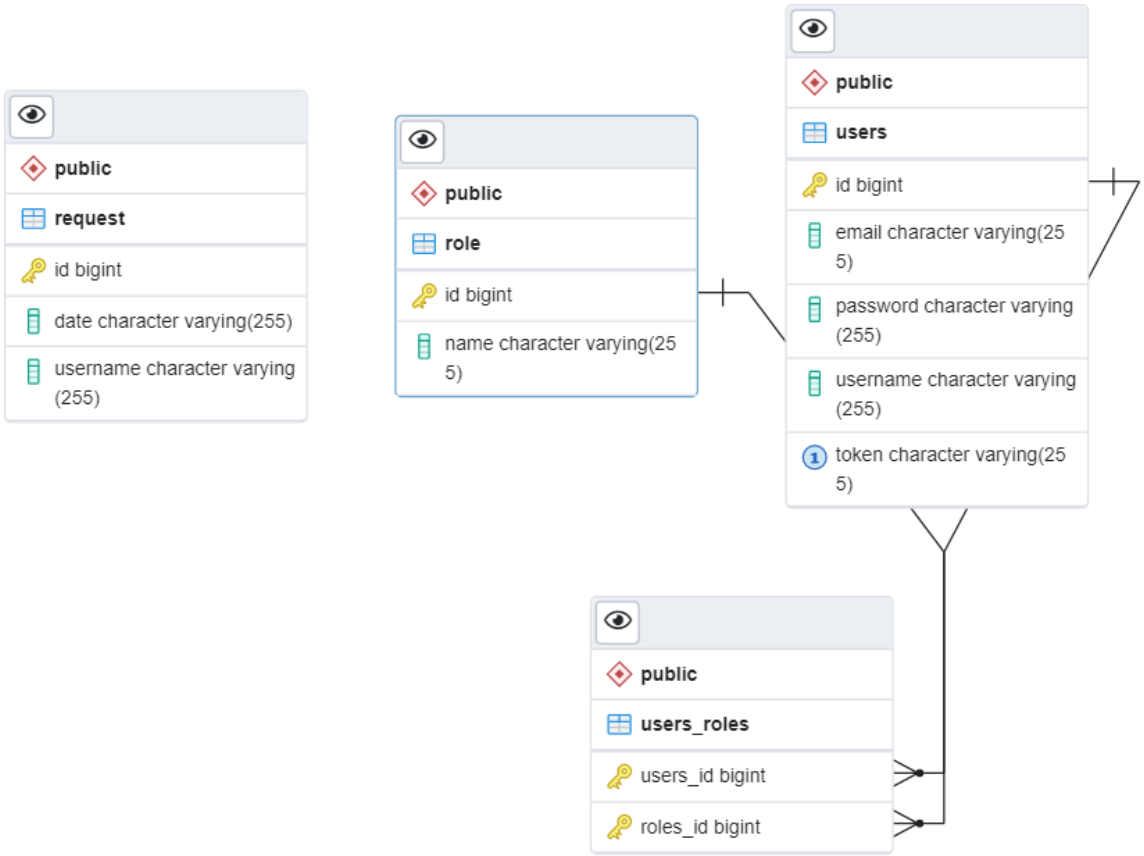

Рис.4.1 – ER діаграма.

Сутності які представлені на малюнку:

Таблиця 4.1

Опис сутностей бази даних.

	request	role	users	users_roles
Опис сутності	Описує сутність запиту, що містить в собі дату та ім'я користувача	Описує наявні ролі користувачів	Описує сутність користувача, що містить в собі електрону пошту, пароль та ім'я	Об’єднує таблиці користувачі то ролей за зв’язком один к багатьом
Атрибути	id (bigint) PK	id (bigint)	id (bigint)	user_id PK
	date (char var)	name (char var)	email (char var)	roles_id PK
	Username (char var)		password (char var)	
			username (char var)	
			token (char var) Unique	

4.2 Class Diagram

Class diagram UML - це графічне зображення архітектури додатку у вигляді діаграми, що використовується для візуалізації і побудови об'єктно-орієнтованих систем [11].

На діаграмі класів ми позначаємо:

- Класи, інтерфейси
- Атрибути класів
- Методи (функції, або операції)
- Вказуємо взаємодію між класами

Для створення інформаційної інтелектуальної системи було створено наступну діаграму класів, що представлена на рис.4.2.

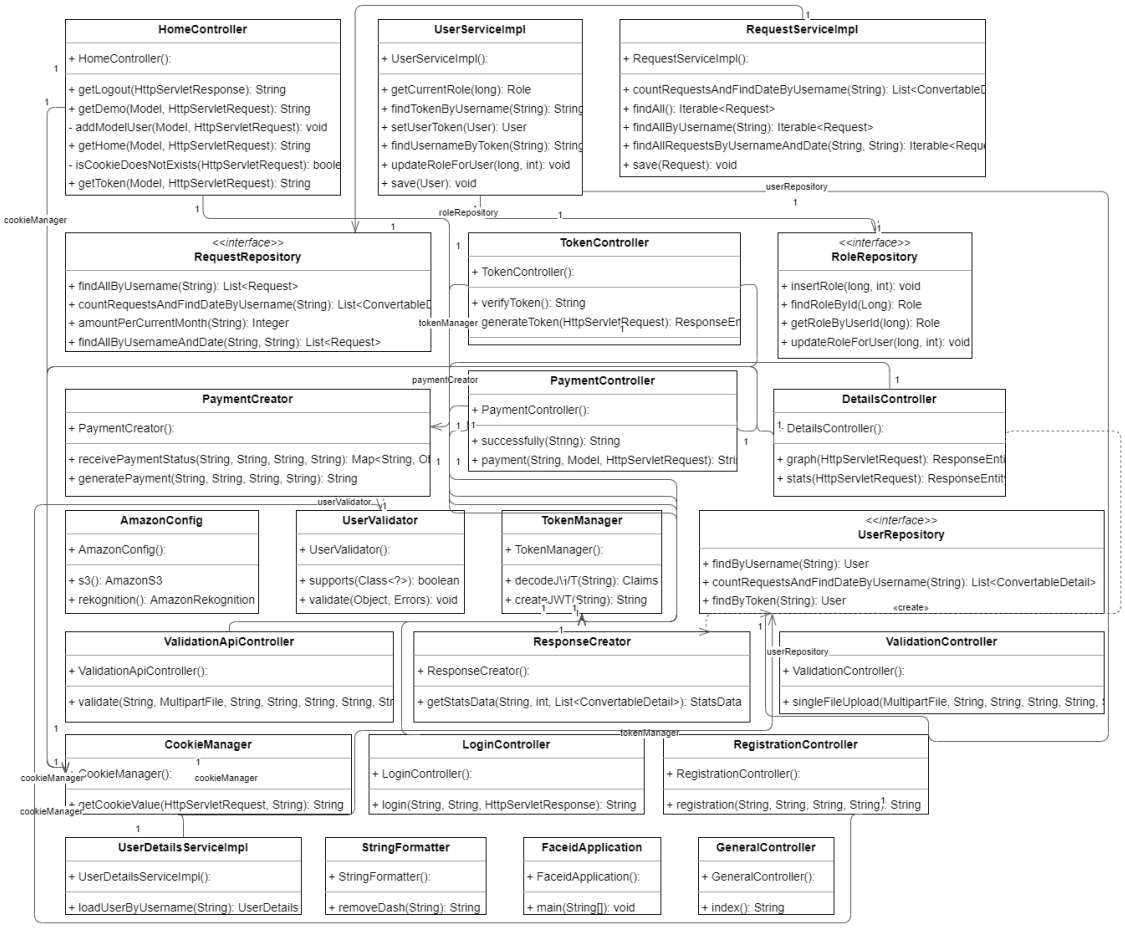

Рис.4.2 – Діаграма класів.

4.3 Use Case Diagram

UML use case diagram – вид діаграм, що показує саму систему, відношення між користувачами (акторами) та прецедентами [12]. Такий вид діаграм доцільно використовувати, щоб наглядно показати усі варіанти користування нашим додатком.

Основними елементами є учасники (actor) і прецедент (варіант користування). Actor – це логічний зовнішній об'єкт, що безпосередньо взаємодіє з системою. Зазвичай при графічному відображені ми використовуємо малюнок "чоловічка" [13]. Прецедент – це опис варіантів (послідовних дій учасника), що призводить до отриманням спостережуваного (очікуваного) результату.

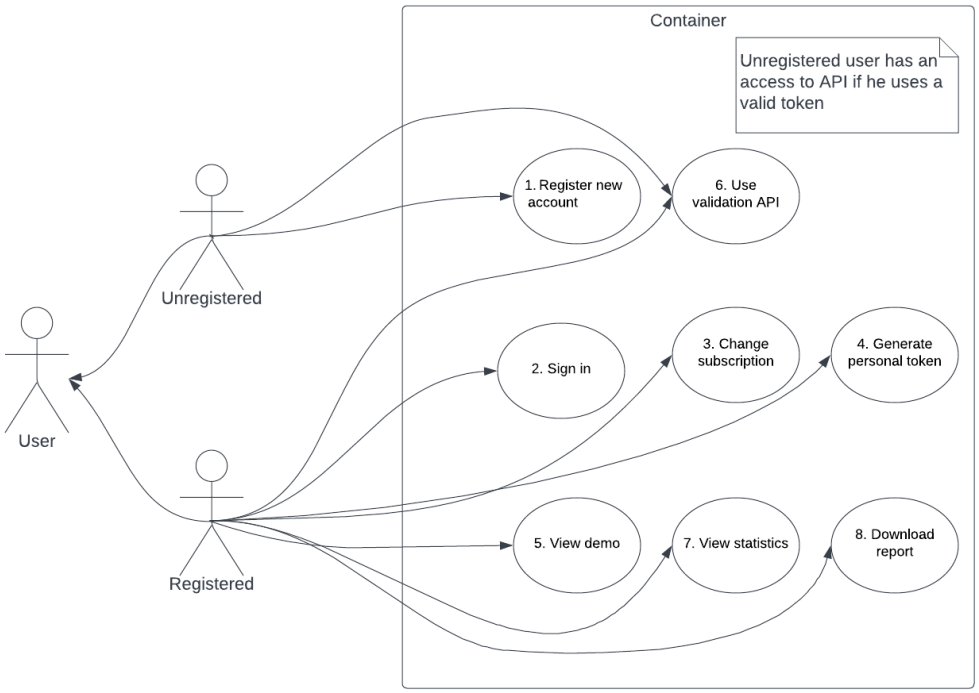

Рис.4.3 – Use case діаграма.

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

- User – поточний користувач системи
- Unregistered – незареєстрований користувач інформаційної системи, що не має доступу до функціоналу системи окрім реєстрації та використання приватного API за умови володіння ключем (токеном).
- Registered – користувач, що повноцінно має можливість

Use case які може використовувати учасник:

1. Register new account. Функція за допомогою якої Unregistered користувач може зареєструвати новий аккаунт.

2. Sign in. Функція що дозволяє авторизуватися в інформаційній системі у вигляді веб додатку.

3. Change subscription. Функція якою можливо змінити роль (підписку) для отримання доступу до функціоналу.

4. Generate personal token. Функція що дозволяє генерувати персональний ключ (токен) для авторизації в API.

5. View demo. Функція за допомогою якої можливий перегляд демонстративної версії інтеграції інтелектуальної інформаційної системи.

6. Use validation API. Функція використання приватного програмного інтерфейсу додатку.

7. View statistics. Функція перегляду статистичних даних щодо використання системи.

8. Download report. Функція що створює можливість завантажити звіт у вигляді CSV файлу.

РОЗДІЛ 5 ОСНОВНА ЧАСТИНА

5.1 Розроблені пакети

На рисунку відображена структура проекту який було створено в наслідок розробки інтелектуальної інформаційної системи у вигляді веб додатку.

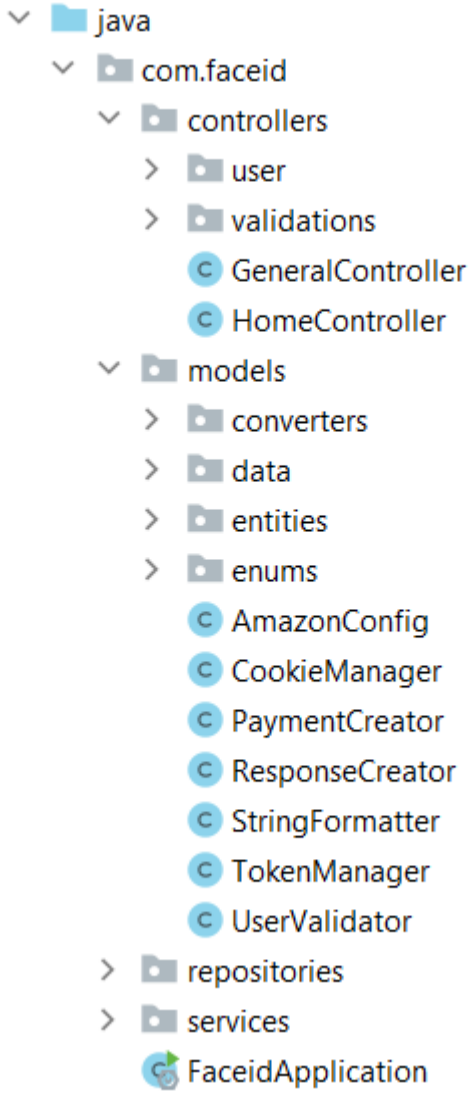

Рис.5.1 – Перелік директорій проекту.

В цій структурі:

- java – є пакетом найвищого рівня.
- com.faceid – є головним кореневим підпакетом пакету java, та є пакетом всього додатку, тут знаходяться всі загальні класи та підпакети, що будуть формувати функціонал нашого web-додатку.

- controllers – цей пакет містить всі контролери для web-view з використанням фреймворку Thymeleaf, а також всі контролери для REST API.

Сам Thymeleaf представляє собою шаблонізатор для веб сторінок описаних мовою розмітки HTML [14], завдяки кодовим вставкам ми можемо передавати з back-end сторони на front-end зміни в моделі та відображати їх на веб сторінках. Для створення контролерів було використано анотації @Controller та @RestController, щодо маніпуляцій над web-view та REST API відповідно [15].

Склад пакету controllers:

1. Підпакет users – містить контролери для роботи з користувачем.
   1.1. DetailsController – контролер, що надає деталі та статистику щодо користування сервісом.
   1.2. LoginController – контролер, який містить функціонал авторизації користувача.
   1.3. PaymentController – платіжний контролер, що відповідає за зміну плану підписки (ролей).
   1.4. RegistrationController – контролер, в якому створений функціонал реєстрації користувача.
   1.5. TokenController – функціональний контролер в якому створений функціонал менеджменту персональних токенів користувачів, для подальшого використання в якості авторизації для API.
2. Підпакет validations – містить верифікаційні контролери.
   2.1. ValidationApiController – головний функціональний контролер, що містить реалізацію для верифікації та валідації особи за фото (зображенням документу).
   2.2. ValidationController – дублюючий контролер для публічної демонстрації роботи інформаційної інтелектуальної системи.
3. GeneralController – контролер для індексації головної сторінки.

4. HomeController – контролер, що містить функціонал для навігації за сторінками.

Склад пакету models:

1. Підпакет converters – містить інтерфейс для конвертації сутностей.
   1.1. ConvertableDetail – функціональний інтерфейс для можливості конвертувати дані з використанням Native Query.
2. Підпакет data – складається з класів, що допомагають оброблювати дані.
   2.1. Generator – клас, що містить методи для генерації рядків.
   2.2. Router – розподіляє потік даних згідно відповідного контролеру.
   2.3. Validator – складається з методів для валідації даних користувача, їх перетворення та обробки.
3. Підпакет entities – містить класи-сутності.
   3.1. Detail – сутність для опису деталей використання сервісу (дата та кількість запитів до приватного API).
   3.2. Identity – сутність, що відображає модель об'єкта розпізнавання.
   3.3. Request – клас на основі якого створюються відповіді від сервера до клієнта.
   3.4. Role – сутність ролей користувача.
   3.5. StatsData – клас, що містить інформацію про статистику використання приватного API (ліміти, кількість використаного ліміту).
   3.6. User – сутність, що описує модель користувача.
4. Підпакет enums – містить класи перерахування.
   4.1. Names – клас з перерахуванням загальних фраз.
   4.2. Roles – клас з перерахуванням ролей користувачів.
5. AmazonConfig – клас з конфігурацією Amazon Web Services.
6. CookieManager – клас для маніпулювання cookies.
7. PaymentCreator – модель, що описує поведінку фінансових транзакцій (придбання підписки для зміни ролі) через API сервісу LiqPay.
8. ResponseCreator – сервісний клас, що формує відповідь клієнту.
9. StringFormatter – клас який має функціонал для форматування рядків.

10. TokenManager – клас, що генерує нові токени для використання приватного API.
11. UserValidation – сервісний клас, що містить методи для перевірки існування користувачів в інтелектуальній інформаційній системі у вигляді веб додатку.

repositories – пакет з інтерфейсами для взаємодії зі сховищем даних. Ці інтерфейси використовують JPA Entity (у якості Hibernate ORM) для взаємодії з базою даних, що забезпечує основні операції по пошуку, створенню, редагуванню та видаленню даних [16].

Склад пакету repositories:

1. RequestRepository – інтерфейс, що містить абстрактні методи для пошуку всіх запитів користувача за такими критеріями: нікнейм, дата, нікнейм та дата. В інтерфейсі наявні методи для підрахування кількості запитів та дати коли цей юзер виконав їх.
2. RoleRepository – інтерфейс, де є методи для додавання, зміни та отримання поточної ролі користувача.
3. UserRepository – інтерфейс з методами для пошуку користувачів за нікнеймом, або токеном.

services – пакет з класами, де реалізований функціонал репозиторіїв.

Склад пакету services:

1. Підпакет amazon – містить сервісні класи для роботи з Amazon Web Serices.
   1.1. RecognizeData – класс в якому створений функціонал роботи з розпізнаванням зображень за допомогою AWS SDK.
   1.2. UploadToS3 – сервісний клас з функціоналом завантаження зображень до хмарного сховища S3 (Amazon Simple Storage Service).
2. Підпакет interfaces – містить інтерфейси для опису дій над сутностями.
   2.1. RequestService – інтерфейс з методами збереження та отримання інформації про запити.
   2.2. UserService – інтерфейс з методами для збереження, редагування та видалення користувача.

3. RequestServiceImpl – клас, що реалізує інтерфейс RequestService.
4. UserServiceImpl – клас у якому імпліментовано методи з інтерфейсу UserService.
5. UserDetailsServiceImpl – клас з реалізацією функціоналу, що був запроваджений в інтерфейсі UserDetailsService.

5.2 Імпортовані пакети

У проекті використовується багато різноманітних імпортованих пакетів (бібліотек/залежностей), фреймворків для суттєвого зменшення часу розробки та підвищення оптимізації роботи серверної частини [17].

У якості автоматизації збірки проекту був обраний Maven. Apache Maven є одним з найпопулярніших інструментів для управління збіркою, з впорядкованою моделюю конфігурації на основі XML [18]. Для опису самого проекту та залежностей, що можуть бути імпортованими ми використовуємо pom файл.

В цьому файлі ми можемо переглянути увесь список імпортованих пакетів:

- spring-boot-starter-data-jpa.
- spring-boot-starter-thymeleaf
- spring-boot-starter-web
- spring-boot-devtools
- spring-boot-starter-test
- spring-boot-starter-security
- spring-boot-maven-plugin
- postgresql
- lombok
- jwt
- apache-commons
- jaxb-api

- json-simple
- aws-java-sdk

5.3 Прототип

В даному розділі ми розглянемо прототип нашої інтелектуальної інформаційної системи у вигляді веб додатку. Ми опишемо графічний дизайн, функціонал нашого додатку та продемонструємо його основні можливості.

На рис.5.2 зображена форма реєстрації з полями. Усі поля потребують заповнення, а саме: email, ім’я користувача, пароль, повтореній пароль. Реєстраційна форма має валідацію як зі сторони front-end (додаток Б) так і на back-end частині (додаток А). Під час перевірки з клієнтської сторони ми задовольняємо певні умови:

1. Текст введений в поле email відповідає формату login@domain.com.
2. Ім’я користувача не містить пробілів та спеціальних символів окрім символу підкреслення.
3. Пароль будь який, але мінімум 8 символів та повністю співпадає з повтореним паролем.

Після натискання кнопки «Register» починається перевірка на серверній частині. Робиться запит до бази даних на отримання інформації про користувача за 2 критеріями: email та username. Якщо хоча б за одним критерієм буде знайдений користувач – отримаємо помилку, а в разі відсутності відомостей – створюємо нового користувача.

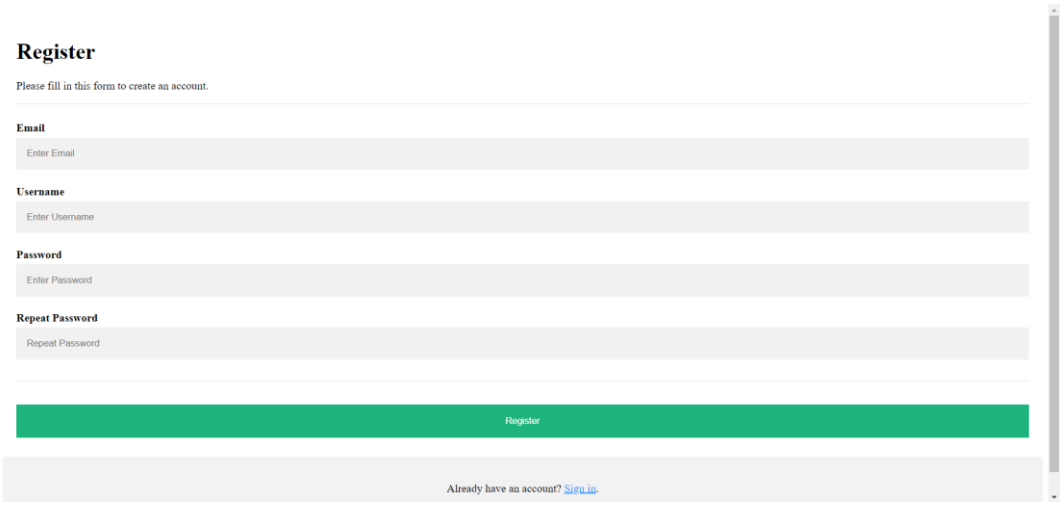

Рис.5.2 – Сторінка реєстрації.

Після успішної реєстрації користувач буде переадресований на сторінку авторизації, що зображена на рис.5.3. Для успішної авторизації користувач повинен ввести правильні дані, що були вказані при реєстрації. Під час авторизації також йде перевірка на спеціальні символи з клієнтської сторони та пошук за паролем та ім'ям користувача на серверній частині.

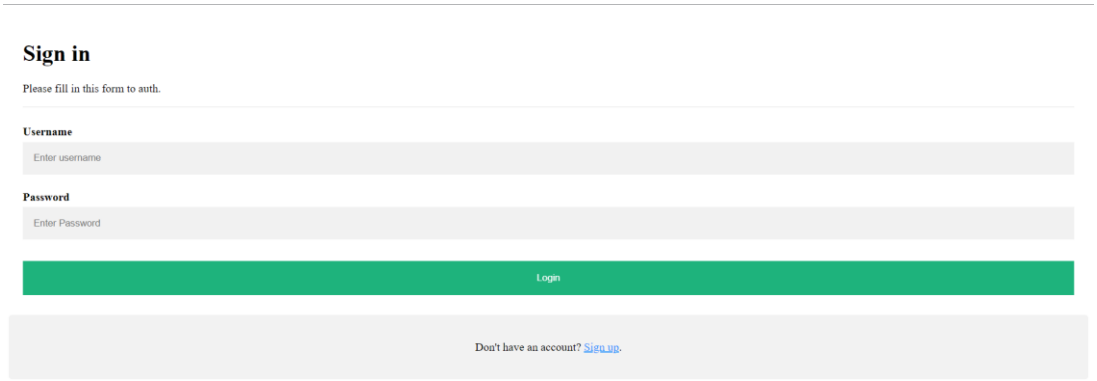

Рис.5.3 – Сторінка авторизації.

Коли користувач успішно авторизувався – він бачить перед собою навігаційну панель, що складається з таких пунктів як: Home, Token, Details, Demo та блок що дає можливість змінити роль (підписку) для збільшення ліміту користуванням приватного програмного інтерфейсу додатку. Щоб зрозуміти яка поточна роль користувача – достатньо подивитися на відсутність кнопки «Update» бо ця кнопка присутня тільки на тих ролях до яких можна підвищити свою.

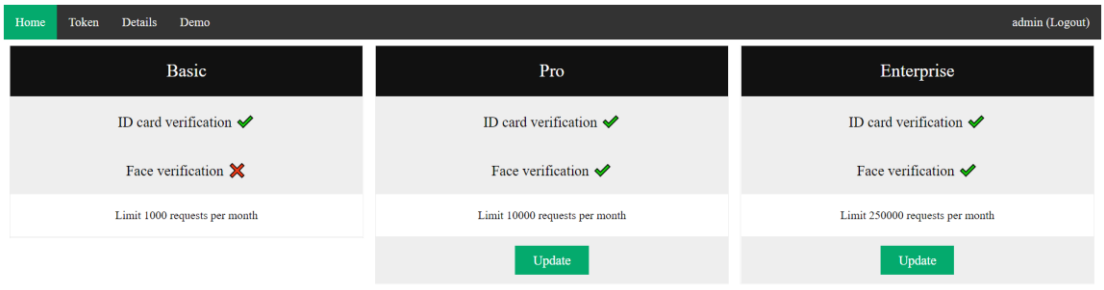

Рис.5.4 – Головна сторінка.

На сторінці Token (рис.5.5) – користувач має змогу згенерувати та змінити персональний токен для успішної авторизації під час використання API.

Рис.5.5 – Сторінка генерації токену.

Користувач завжди має змогу переглянути свій токен скориставшись кнопкою «Show token» як це показано на рис.5.6.

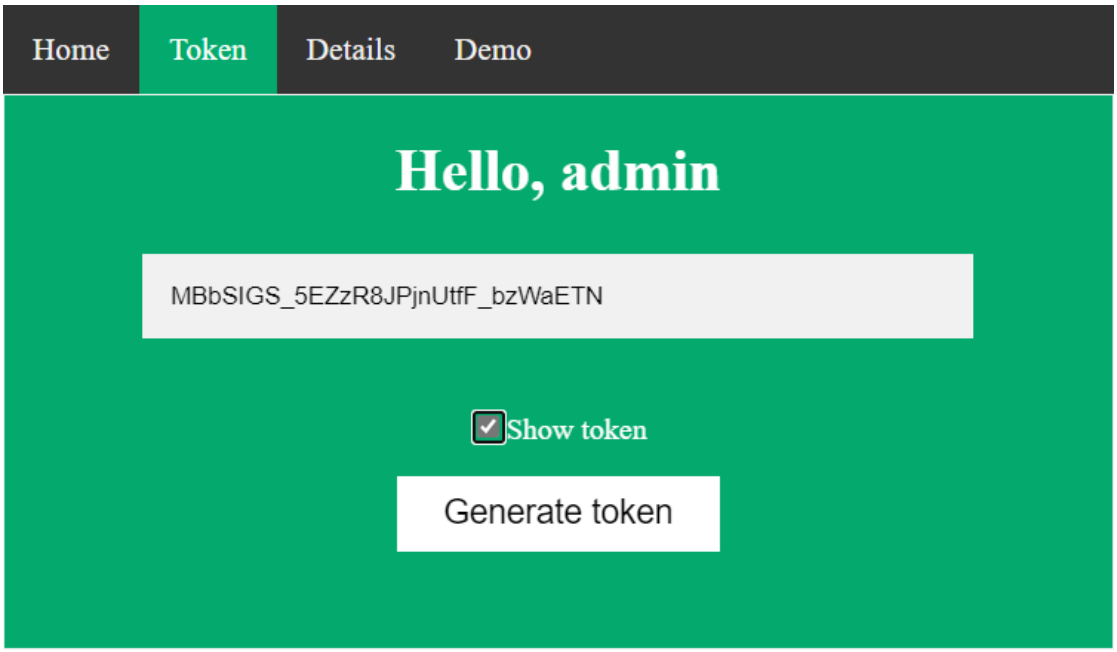

Рис5.6 – Блок генерації токену.

Якщо користувач переключиться до третьої сторінки «Details» - він матиме можливість передивитися власну статистику користування програмним інтерфейсом інтелектуальної інформаційної системи. Умовно цю сторінку можливо поділити на 2 блоки. Перший блок містить графік залежності кількості викликів API до дати, а другий містить статистику щодо вичерпаних лімітів. Там наявна інформація про поточний план користування (підписку), скільки використано відносно всього ліміту та який середній показник звернень до API.

Також у другому блоці наявні 2 кнопки:

1. Update – оновлює сторінку без перезавантаження за допомогою технології AJAX [19].

2. Download – при натисканні починається завантаження CSV файлу з інформацією про те скільки разів був виклик API відносно певного дня.

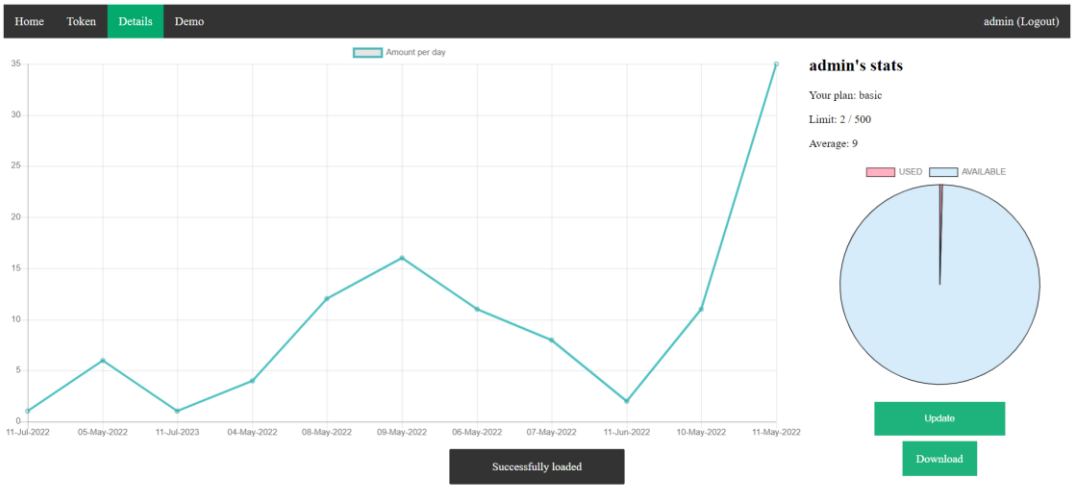

Рис.5.6 – Сторінка статистики використання.

Остання сторінка (рис.5.6), що може бути корисною для користувача – демонстрація успішної інтеграції інтелектуальної інформаційної системи. Наведений приклад допомагає користувачу спробувати самостійно без конкретних навичок у програмуванні використати сервіс.

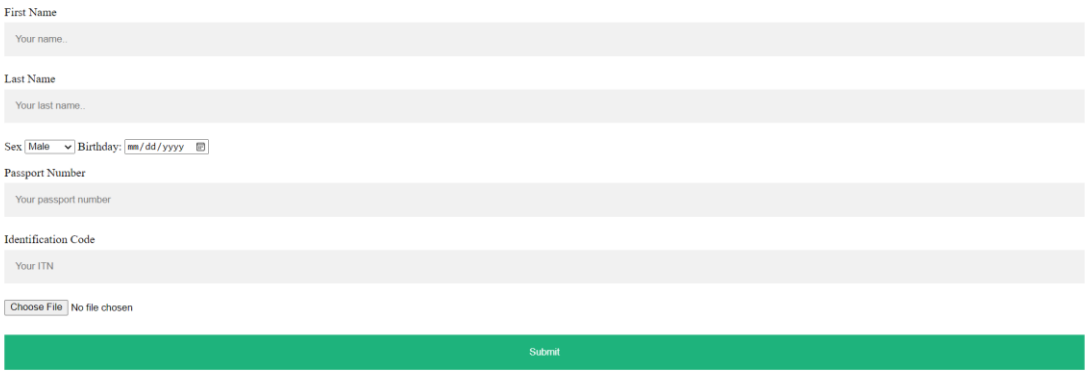

Рис.5.7 – Демонстраційна форма відправки на верифікацію.

Користувач має заповнити поля інформацією відповідно до вказаних на документі та завантажити файл-фото документу. Для демонстрації роботи сервісу можна використати приклад ID картки, що є у вільному доступі та окреме фото цієї особи.

Рис.5.8 – Заповнена демонстраційна форма.

Під час натискання кнопки «Submit» - серверна частина отримує інформацію та починається процес верифікації особи (рис.5.9).

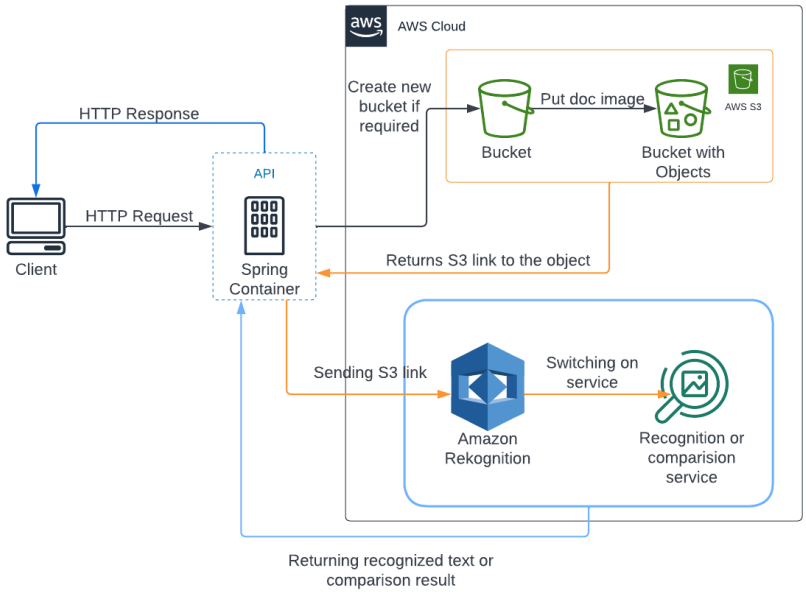

Рис.5.9 – Діаграма процесу верифікації.

Алгоритм роботи можливо описати наступним шляхом:

1. Клієнт надсилає запит до нашого API

2. Серверна частина за допомогою AWS SDK створює бакет під користувача (якщо його немає), та додає зображення документа в нього.

3. У відповідь AWS повертає S3 посилання на об'єкт в конкретному бакеті [20].

4. Надалі це посилання передається до сервісу Amazon Rekognition де в залежності від обратного сервісу (розпізнавання тексту на документі, або порівняння персонального фото та фото на документі) [21].

5. Повертаються розпізнані текстові блоки до нашого серверу.

6. Сервер співвідносить дані що були надані клієнтом з данними що отримані в результаті розпізнавання тексту та формує відповідь клієнту (рис.5.10).

Рис.5.10 – Оброблений результат відповіді клієнту.

Якщо клієнт побажає інтегрувати цю інтелектуальну інформаційну систему до своїх сервісів – йому потрібно буде створити GET запит на endpoint: /api/validatie вказавши в Headers ключ Authorization зі значенням токену, що можливо отримати на сторінці «Token» (рис.5.11) та в Body перерахувати всі потрібні поля (firstname, lastname, sex, birthday, passportNumber, idNumber, file) з заповненими значеннями (рис5.12.).

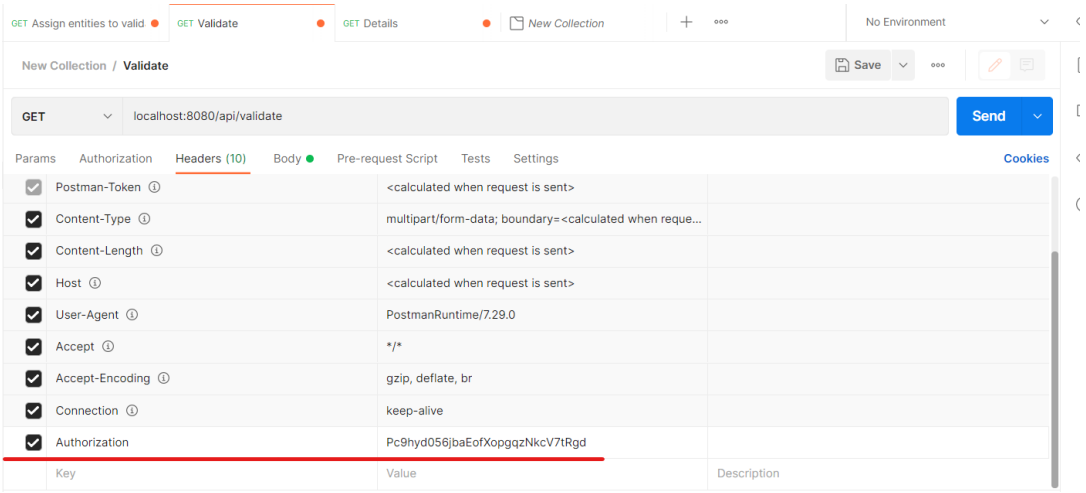

Рис.5.11 – Демонстрація заповненого Headers з авторизацією по токену.

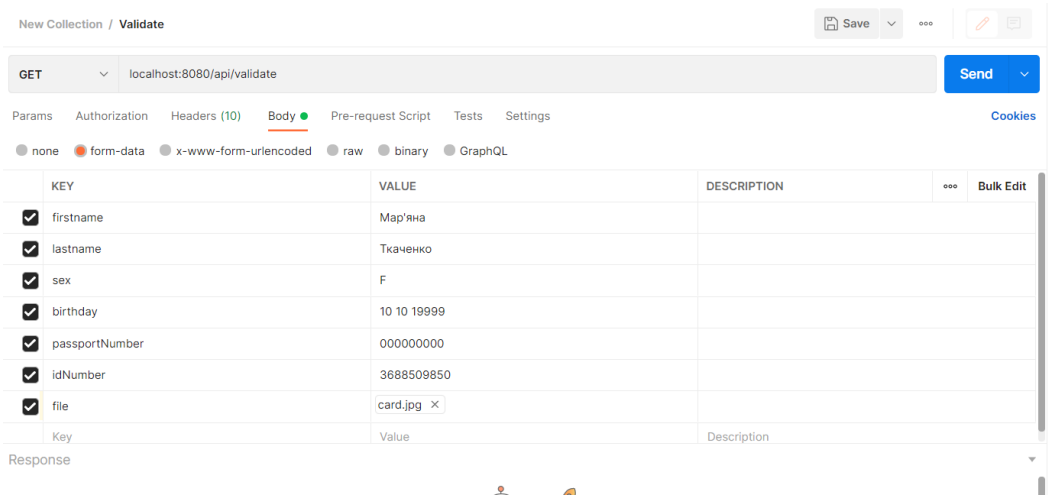

Рис.5.12 – Демонстрація заповненого Body.

В якості успішного виконання запиту ми отримуємо JSON з інформацією про те які значення співпали, а які ні (рис.5.13).

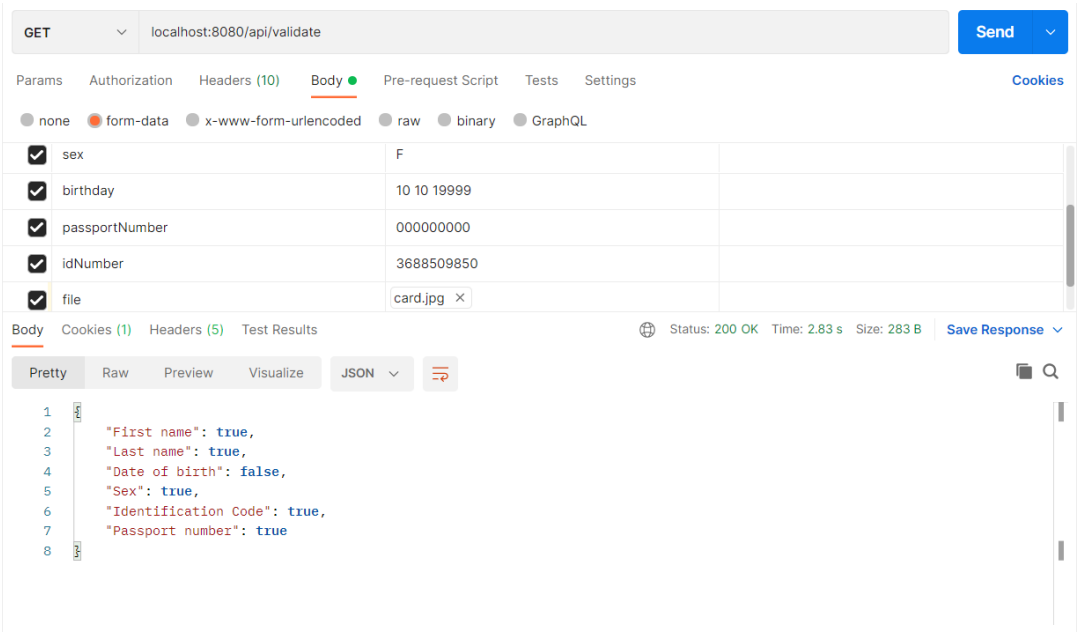

Рис.5.13 – Відповідь з результатами у вигляді JSON.

Наглядно зрозуміти як розпізнає сервіс AWS Rekognition текст на документі можливо переглянувши рис.5.14.

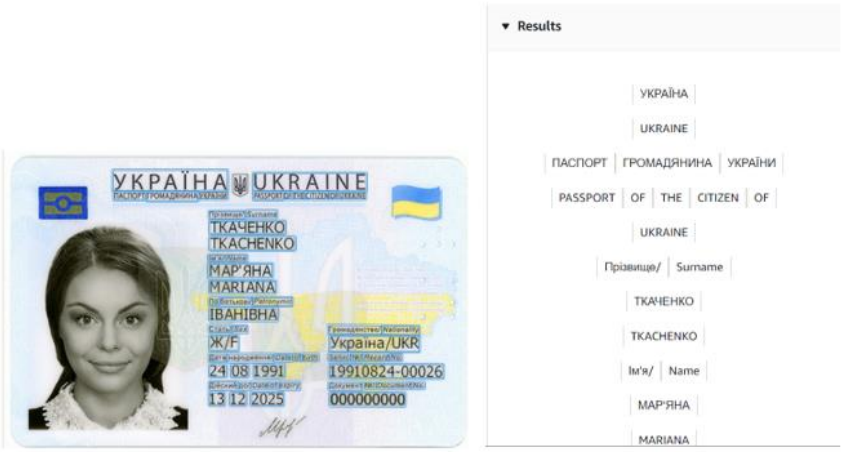

Рис.5.14 – Приклад розпізнаних блоків за допомогою AWS Rekognition.

Якщо користувач забажає інтегрувати систему для звірення персонального фото, та фото що знаходиться на документі – йому потрібно буде створити GET запит на endpoint: /api/compare де так само вказати персональний ключ в Headers та передати 2 зображення в Body (document, photo). У якості відповіді на запит – користувач отримує JSON лише з одним полем Similarity де вказано відсоток співпадання персонального фото та фото на документі (рис.5.15).

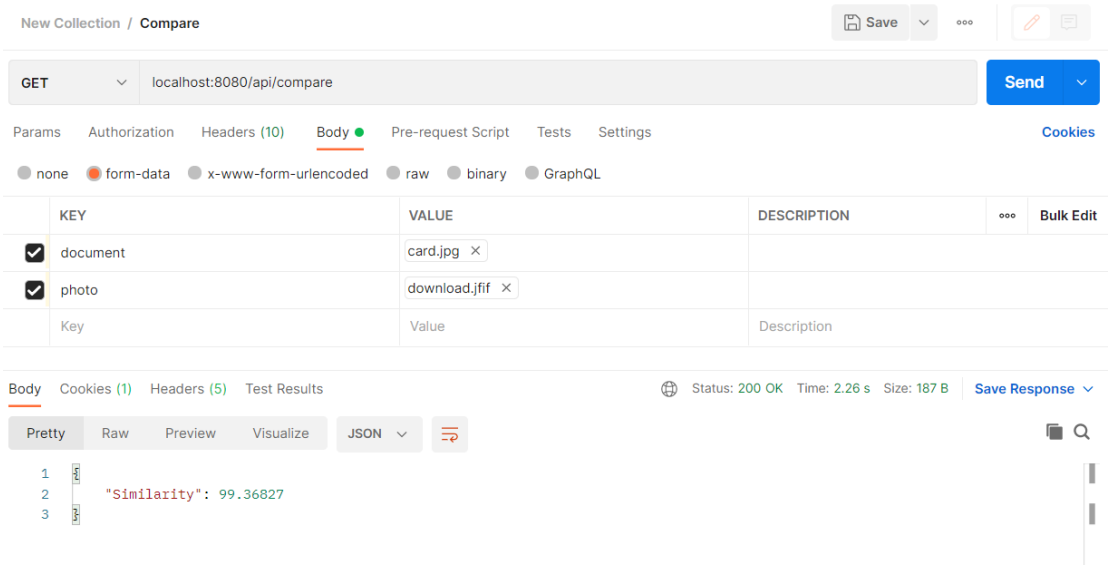

Рис.5.15 – Результат порівняння фотографій.

Наглядний приклад як сервіс AWS Rekognition порівнює фотографії можливо побачити на рис.5.16 та рис 5.17 де зображено вхідні зображення та результат порівняння відповідно.

Рис.5.16 – Приклад вхідних зображень для порівняння.

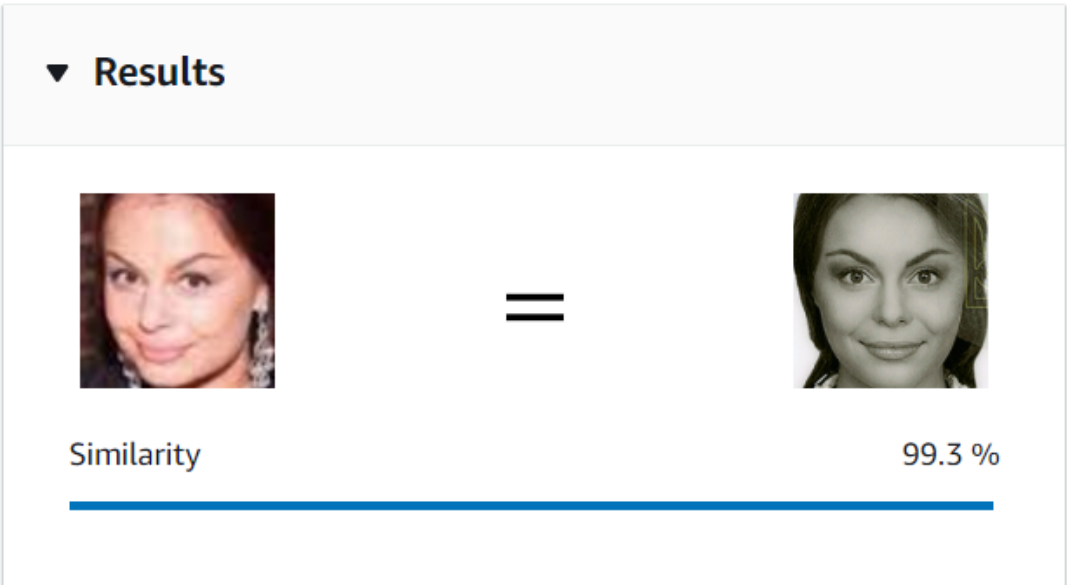

Рис.5.17 – Приклад результату порівняння фото за допомогою Amazon Rekognition

У випадку розпізнавання особистих даних з паспорту спочатку розпізнається текст, а потім відбувається порівняння з даними, що були введені про особу до інтелектуальної системи, через передбачену для

користувача систему вводу. Коли система ідентифікує особу, порівнюючи її зовнішність з фото у паспорті, ми отримуємо високий відсоток, у разі співпадіння.

ВИСНОВКИ

У даній роботі було розроблено інтелектуальну інформаційну систему у вигляді веб додатку з використанням хмарних технологій для фото верифікації особи.

У ході виконання роботи було розглянуто та пропрацьовано велику кількість джерел інформації щодо використання хмарних технологій, методологій проектування додатку та патернів проектування. Проаналізувавши зібрану інформацію було обрано найоптимальніший варіант стеку технологій для створення інформаційної системи.

В ході вибору технологічних рішень для використання при створення додатку було проаналізовано як аналоги, що присутні на ринку, так і аналоги серед постачальників хмарних сервісів. Було обрано найбільш зручні інструменти для розробки, що зменшують швидкість та складність написання додатку, підвищують ефективність роботи сервісу, а також інструменти для розробки клієнтської частини, також забезпечують масштабування, та можливість інтеграції з іншими сервісами, що може бути корисним при подальшому розвитку створеної системи.

В ході виконання роботи були розроблені діаграми, а саме: ERD, Use case та Class діаграми. Також було розроблено клієнтську частину та головні сервіси відповідно до бізнес логіки інтелектуальної інформаційної системи.

Створений прототип веб додатку з навігацією, головною сторінкою, сторінкою генерації персонального ключа, перегляду статистики використаних ресурсів та демонстраційної роботи інформаційної системи.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vnukova N. Financial Institutions Risks Management in the Field of Financial Monitoring // Sci. Notes Ostroh Acad. Natl. Univ. «Economics» Ser. 2018. T. 1, № 8 (36). С. 64–68.
2. Alzakholi O. и др. Comparison Among Cloud Technologies and Cloud Performance // J. Appl. Sci. Technol. Trends. 2020. Т. 1, № 2. С. 40–47.
3. Castelblanco A. и др. Machine learning techniques for identity document verification in uncontrolled environments: A case study // Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). Springer, 2020. Т. 12088 LNCS. С. 271–281.
4. Що таке хмарні технології і як вони можуть допомогти вашому підприємству? [Електронний ресурс]. URL: https://business.diia.gov.ua/cases/tehnologii/so-take-hmarni-tehnologii-i-ak-voni-mozut-dopomogti-vasomu-pidpriemstvu (дата обращения: 08.06.2022).
5. Object Oriented Prog With Java. McGraw-Hill Education (India) Pvt Limited, 2009.
6. Lindholm T. и др. The Java Virtual Machine Specification. Java SE 8. 2015.
7. Облачная пирамида: IaaS, PaaS и SaaS [Електронний ресурс]. URL: https://gigacloud.ua/ru/blog/navchannja/hmarna-piramida-iaas-paas-i-saas (дата звернення: 12.06.2022).
8. Що таке хмарні технології і як вони працюють | EDIN [Електронний ресурс]. URL: https://edin.ua/shho-take-xmarni-texnologiï-i-navishho-voni-potribni/ (дата звернення: 08.06.2022).
9. AWS, Azure, Google Cloud - что лучше для вашего бизнеса [Детальный разбор] | Dinarys [Електронний ресурс]. URL: https://dinarys.com/ru/blog/comparison-of-aws-vs-azure-vs-google-cloud (дата звернення: 08.06.2022).
10. Учимся проектированию Entity Relationship — диаграмм / Habr [Електронний ресурс]. URL: https://habr.com/en/post/440556/ (дата звернення: 08.06.2022).

11. Диаграмма классов — Википедия [Електронний ресурс]. URL: https://ru.wikipedia.org/wiki/Диаграмма_классов (дата звернення: 06.06.2022).
12. UML Use Case Diagram Tutorial | Lucidchart [Електронний ресурс]. URL: https://www.lucidchart.com/pages/uml-use-case-diagram (дата звернення: 05.06.2022).
13. Как и зачем писать Use Cases | Shark Develop [Електронний ресурс]. URL: https://sharkdevelop.com/use-cases/ (дата звернення: 11.06.2022).
14. Thymeleaf - Wikipedia [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Thymeleaf (дата звернення: 10.06.2022).
15. Difference Between @Controller and @RestController Annotation in Spring - GeeksforGeeks [Електронний ресурс]. URL: https://www.geeksforgeeks.org/difference-between-controller-and-restcontroller-annotation-in-spring/ (дата звернення: 09.06.2022).
16. Your relational data. Objectively. - Hibernate ORM [Електронний ресурс]. URL: https://hibernate.org/orm/ (дата звернення: 11.06.2022).
17. Oaks S. Java performance : the definitive guide. O'Reilly Media, 2014. С. 408.
18. Maven – Introduction [Електронний ресурс]. URL: https://maven.apache.org/what-is-maven.html (дата звернення: 11.06.2022).
19. Ajax: Отправить HTML-форму без перезагрузки страницы [Електронний ресурс]. URL: https://fructcode.com/ru/blog/how-to-send-html-form-with-ajax/ (дата звернення: 10.06.2022).
20. Creating, listing, and deleting Amazon S3 buckets - AWS SDK for Java [Електронний ресурс]. URL: https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/examples-s3-buckets.html (дата звернення: 06.06.2022).
21. Detecting text in an image - Amazon Rekognition [Електронний ресурс]. URL: https://docs.aws.amazon.com/rekognition/latest/dg/text-detecting-text-procedure.html (дата звернення: 06.06.2022).

Додаток А

ЛІСТИНГ JAVA КЛАСІВ ДЛЯ РЕАЛІЗАЦІЇ СЕРВЕРНОЇ ЧАСТИНИ

```
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.countRequestsAndFindDateByUsername(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/octet-
stream"))
                .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.CookieManager;
import com.faceid.models.PaymentCreator;
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.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import javax.servlet.http.HttpServletRequest;

@Controller
public class PaymentController {
    @Value("${liqpay.publicKey}")
    private String publicKey;
    @Value("${liqpay.privateKey}")
    private String privateKey;
    @Autowired
    private PaymentCreator paymentCreator;
    @Autowired
    private UserService userService;
    @Autowired
    private TokenManager tokenManager;
    @Autowired
    private CookieManager cookieManager;

    private User currentUser;
    private Role currentRole;

    @RequestMapping(value = "/subscribe/{role}", method = RequestMethod.GET)
```

```
    public String payment(@PathVariable("role") String role, Model model,
HttpServletRequest req) {
        currentUser = userService.findByUsername(
                tokenManager.decodeJWT(cookieManager.getCookieValue(req,
"auth-token")).getIssuer()
        );
        currentRole = userService.getCurrentRole(currentUser.getId());
        model.addAttribute("html",
paymentCreator.generatePayment(currentUser.getUsername(), role, publicKey,
privateKey));
        model.addAttribute("user", currentUser);
        model.addAttribute("role", currentRole);
        return "payment";
    }

    @PostMapping("/successfully/{role}")
    public String successfully(@PathVariable("role") String role) {
        int roleId = 1;
        if (role.equals("pro")) {
            roleId = 2;
        }else if(role.equals("enterprise")){
            roleId = 3;
        }
        userService.updateRoleForUser(currentUser.getId(), roleId);
        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-MMM-
yyyy")));
        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-MMM-
yyyy")));
        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);
    }
}
```

Додаток Б

ЛІСТИНГ HTML ТА CSS ФАЙЛІВ ДЛЯ РЕАЛІЗАЦІЇ КЛІЄНТСЬКОЇ ЧАСТИНИ

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>FaceID Validator Login</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
<form action="/login">
  <div class="container-1">
    <h1>Sign in</h1>
    <p>Please fill in this form to auth.</p>
    <hr>

    <label for="username"><b>Username</b></label>
    <input type="text" placeholder="Enter username" name="username"
id="username" required>

    <label for="psw"><b>Password</b></label>
    <input type="password" placeholder="Enter Password" name="password"
id="psw" required>

    <button type="submit" class="registerbtn">Login</button>
  </div>

  <div class="container signin">
    <p>Don't have an account? <a href="/signup.html">Sign up</a>.</p>
  </div>
</form>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>FaceID Validator Registration</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
<form action="/registration" method="post">
  <div class="container-1">
    <h1>Register</h1>
    <p>Please fill in this form to create an account.</p>
    <hr>

    <label for="email"><b>Email</b></label>
    <input type="text" placeholder="Enter Email" name="email" id="email"
required>

    <label for="username"><b>Username</b></label>
    <input type="text" placeholder="Enter Username" name="username"
id="username" required>

    <label for="psw"><b>Password</b></label>
    <input type="password" placeholder="Enter Password" name="password"
id="psw" required>
```

```
    <label for="psw-repeat"><b>Repeat Password</b></label>
    <input type="password" placeholder="Repeat Password" name="repassword"
id="psw-repeat" required>
    <hr>
    <button type="submit" class="registerbtn">Register</button>
  </div>

  <div class="container signin">
    <p>Already have an account? <a href="/index.html">Sign in</a>.</p>
  </div>
</form>
</body>
</html>
* {box-sizing: border-box}

/* Add padding to containers */
.container {
    padding: 16px;
}

/* Full-width input fields */
input[type=text], input[type=password] {
    width: 100%;
    padding: 15px;
    margin: 5px 0 22px 0;
    display: inline-block;
    border: none;
    background: #f1f1f1;
}

input[type=text]:focus, input[type=password]:focus {
    background-color: #ddd;
    outline: none;
}

/* Overwrite default styles of hr */
hr {
    border: 1px solid #f1f1f1;
    margin-bottom: 25px;
}

/* Set a style for the submit/register button */
.registerbtn {
    background-color: #04AA6D;
    color: white;
    padding: 16px 20px;
    margin: 8px 0;
    border: none;
    cursor: pointer;
    width: 100%;
    opacity: 0.9;
}

.registerbtn:hover {
    opacity:1;
}

/* Add a blue text color to links */
a {
    color: dodgerblue;
}

/* Set a grey background color and center the text of the "sign in" section
*/
```

```
.signin {
    background-color: #f1f1f1;
    text-align: center;
}
input[type=submit]:hover {
    background-color: #45a049;
}

.container {
    border-radius: 5px;
    background-color: #f2f2f2;
    padding: 20px;
}
.container-1 {
    border-radius: 5px;
    background-color: #ffffff;
    padding: 20px;
}
/* Style the table */
table {
    border-collapse: collapse;
    border-spacing: 0;
    width: 100%;
    border: 1px solid #ddd;
}

/* Style table headers and table data */
th, td {
    text-align: center;
    padding: 16px;
}

th:first-child, td:first-child {
    text-align: left;
}

/* Zebra-striped table rows */
tr:nth-child(even) {
    background-color: #f2f2f2
}

.fa-check {
    color: green;
}

.fa-remove {
    color: red;
}
#snackbar {
    visibility: hidden; /* Hidden by default. Visible on click */
    min-width: 250px; /* Set a default minimum width */
    margin-left: -125px; /* Divide value of min-width by 2 */
    background-color: #333; /* Black background color */
    color: #fff; /* White text color */
    text-align: center; /* Centered text */
    border-radius: 2px; /* Rounded borders */
    padding: 16px; /* Padding */
    position: fixed; /* Sit on top of the screen */
    z-index: 1; /* Add a z-index if needed */
    left: 50%; /* Center the snackbar */
    bottom: 30px; /* 30px from the bottom */
}

/* Show the snackbar when clicking on a button (class added with JavaScript)
```

```
*/
#snackbar.show {
    visibility: visible; /* Show the snackbar */
    /* Add animation: Take 0.5 seconds to fade in and out the snackbar.
    However, delay the fade out process for 2.5 seconds */
    -webkit-animation: fadein 0.5s, fadeout 0.5s 2.5s;
    animation: fadein 0.5s, fadeout 0.5s 2.5s;
}

/* Animations to fade the snackbar in and out */
@-webkit-keyframes fadein {
    from {bottom: 0; opacity: 0;}
    to {bottom: 30px; opacity: 1;}
}

@keyframes fadein {
    from {bottom: 0; opacity: 0;}
    to {bottom: 30px; opacity: 1;}
}

@-webkit-keyframes fadeout {
    from {bottom: 30px; opacity: 1;}
    to {bottom: 0; opacity: 0;}
}

@keyframes fadeout {
    from {bottom: 30px; opacity: 1;}
    to {bottom: 0; opacity: 0;}
}
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>FaceID Validator</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
<div class="container-1">
  <form method="post" action="/validate" enctype="multipart/form-data">

    <label for="fname">First Name</label>
    <input type="text" id="fname" name="firstname" placeholder="Your name..">

    <label for="lname">Last Name</label>
    <input type="text" id="lname" name="lastname" placeholder="Your last
name..">

    <label for="sex">Sex</label>
    <select id="sex" name="sex">
      <option value="M">Male</option>
      <option value="F">Female</option>
    </select>

    <label for="birthday">Birthday:</label>
    <input type="date" id="birthday" name="birthday" data-date-format="DD
MMMM YYYY">

    <p></p>
    <label for="passportNumber">Passport Number</label>
    <input type="text" id="passportNumber" name="passportNumber"
placeholder="Your passport number">

    <label for="idNumber">Identification Code</label>
    <input type="text" id="idNumber" name="idNumber" placeholder="Your ITN">
```

```
    <input type="file" name="file" /><br/><br/>
    <input class="registerbtn" type="submit" value="Submit" />
  </form>
</div>
</body>
</html>
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" >
<head>
    <meta charset="UTF-8">
    <title>Details</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
<style>
    .row {
        margin-left:-5px;
        margin-right:-5px;
    }

    .column-1 {
        float: left;
        width: 75%;
        padding: 5px;
    }
    .column-2 {
        float: left;
        width: 25%;
        padding: 5px;
    }
    /* Add a black background color to the top navigation */
    .topnav {
        background-color: #333;
        overflow: hidden;
    }

    /* Style the links inside the navigation bar */
    .topnav a {
        float: left;
        color: #f2f2f2;
        text-align: center;
        padding: 14px 16px;
        text-decoration: none;
        font-size: 17px;
    }
    .topnav .name {
        float: right;
        color: #f2f2f2;
        text-align: center;
        padding: 14px 16px;
        text-decoration: none;
        font-size: 17px;
    }

    /* Change the color of links on hover */
    .topnav a:hover {
        background-color: #ddd;
        color: black;
    }

    /* Add a color to the active/current link */
    .topnav a.active {
        background-color: #04AA6D;
```

```
        color: white;
    }
    * {
        box-sizing: border-box;
    }


    /* List items */
    .price li {
        border-bottom: 1px solid #eee;
        padding: 20px;
        text-align: center;
    }


    /* The "Sign Up" button */
    .button {
        background-color: #ffffff;
        border: none;
        color: #000000;
        padding: 10px 25px;
        text-align: center;
        text-decoration: none;
        font-size: 18px;
    }
    input[type=text], input[type=password] {
        width: 75%;
        padding: 15px;
        margin: 5px 0 22px 0;
        display: inline-block;
        border: none;
        background: #f1f1f1;
    }
</style>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></scri
pt>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.6.0/Chart.min.js"></sc
ript>
<script>
    function callSnackbar() {
        var x = document.getElementById("snackbar");
        x.className = "show";
        setTimeout(function(){ x.className = x.className.replace("show", "");
}, 3000);
    }
    function dataLoad(){
        $.ajax({
            method : 'GET',
            dataType: "text",
            url : '/graph',
            success: function(data) {
                var labels = JSON.parse(data).map(function(e) {
                    return e.date;
                });
                var data = JSON.parse(data).map(function(e) {
                    return e.amount;
                });
                var ctx = canvas.getContext('2d');
                var config = {
                    type: 'line',
                    data: {
                        labels: labels,
```

```
                        datasets: [{
                            label: 'Amount per day',
                            data: data,
                            fill: false,
                            borderColor: 'rgb(75, 192, 192)',
                            tension: 0
                        }]
                    }
                };
                var chart = new Chart(ctx, config);
            }
        });
        $.ajax({
            method : 'GET',
            dataType: "text",
            url : '/stats',
            success: function(data) {
                var data = JSON.parse(data);
                var average = data.average;
                var limit = data.limit;
                var used = data.used;
                var free = data.free;
                $('#average').text('Average: ' + average);
                $('#limit').text('Limit: ' + used + " / " + limit);
                var donut = document.getElementById("donut");
                var config = {
                    type: 'pie',
                    data: {
                        labels: ['USED', 'AVAILABLE'],
                        datasets: [{
                            label: '#',
                            data: [used, free],
                            backgroundColor: [
                                'rgba(255, 99, 132, 0.5)',
                                'rgba(54, 162, 235, 0.2)'
                            ],
                            borderColor: [
                                'rgb(2,2,2)',
                                'rgb(2,2,2)'
                            ],
                            borderWidth: 1
                        }]
                    },
                    options: {
                        responsive: true,
                    }
                };
                var myChart = new Chart(donut, config);
            }
        });
        callSnackbar();
    }
</script>
<div class="topnav">
    <a href="/home">Home</a>
    <a href="/token">Token</a>
    <a class="active" href="/details">Details</a>
    <a href="/demo">Demo</a>
    <a th:text="${user.getUsername()} + ' (Logout)'" href="/logout"
class="name"></a>
</div>
<div>
    <div class="row">
        <div class="column-1">
```

```
            <canvas id="canvas"></canvas>
        </div>
        <div class="column-2">
            <div>
                <h2 th:text="${user.getUsername()} + '\'s stats'"></h2>
                <p th:text="'Your plan: ' + ${role.getName()}"></p>
                <p id="limit"></p>
                <p id="average"></p>
                <center>
                    <canvas id="donut" style="height: 47%; width:
47%"></canvas>
                    <p><button onclick="dataLoad()" class="registerbtn"
style="position:center; width:50%; height:75%">Update</button></p>
                    <p><a href="javascript:window.location='download'"
class="registerbtn" style="position:center; width:50%; height:75%;text-
decoration:none">Download</a></p>
                </center>
            </div>
        </div>
    </div>
    <div id="snackbar">Successfully loaded</div>
</div>
<script type="text/javascript">
    $(document).ready(function() {
        dataLoad();
    });
</script>
</body>
</html>
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" >
<head>
    <meta charset="UTF-8">
    <title>Results of comparison</title>
</head>
<body>
<style>
    /* Add a black background color to the top navigation */
    .topnav {
        background-color: #333;
        overflow: hidden;
    }

    /* Style the links inside the navigation bar */
    .topnav a {
        float: left;
        color: #f2f2f2;
        text-align: center;
        padding: 14px 16px;
        text-decoration: none;
        font-size: 17px;
    }
    .topnav .name {
        float: right;
        color: #f2f2f2;
        text-align: center;
        padding: 14px 16px;
        text-decoration: none;
        font-size: 17px;
    }

    /* Change the color of links on hover */
    .topnav a:hover {
        background-color: #ddd;
```

```
    color: black;
}

/* Add a color to the active/current link */
.topnav a.active {
    background-color: #04AA6D;
    color: white;
}
* {
    box-sizing: border-box;
}

/* Create three columns of equal width */
.columns {
    float: left;
    width: 33.3%;
    padding: 8px;
}

/* Style the list */
.price {
    list-style-type: none;
    border: 1px solid #eee;
    margin: 0;
    padding: 0;
    -webkit-transition: 0.3s;
    transition: 0.3s;
}

/* Add shadows on hover */
.price:hover {
    box-shadow: 0 8px 12px 0 rgba(0,0,0,0.2)
}

/* Pricing header */
.price .header {
    background-color: #111;
    color: white;
    font-size: 25px;
}

/* List items */
.price li {
    border-bottom: 1px solid #eee;
    padding: 20px;
    text-align: center;
}

/* Grey list item */
.price .grey {
    background-color: #eee;
    font-size: 20px;
}

/* The "Sign Up" button */
.button {
    background-color: #04AA6D;
    border: none;
    color: white;
    padding: 10px 25px;
    text-align: center;
    text-decoration: none;
    font-size: 18px;
}
```

```
    /* Change the width of the three columns to 100%
    (to stack horizontally on small screens) */
    @media only screen and (max-width: 600px) {
        .columns {
            width: 100%;
        }
    }
</style>
<div class="topnav">
    <a class="active" href="/home">Home</a>
    <a href="/token">Token</a>
    <a href="/details">Details</a>
    <a href="/demo">Demo</a>
    <a th:text="${user.getUsername()} + ' (Logout)'" href="/logout"
class="name"></a>
</div>
<div class="columns">
    <ul class="price">
        <li class="header">Basic</li>
        <li class="grey">ID card verification ✔</li>
        <li class="grey">Face verification ✘ </li>
        <li class="green">Limit 1000 requests per month</li>
        <li class="grey" th:if="${role.getName()} != 'basic'"><a href="#"
class="button">Update</a></li>
    </ul>
</div>
<div class="columns">
    <ul class="price">
        <li class="header">Pro</li>
        <li class="grey">ID card verification ✔</li>
        <li class="grey">Face verification ✔</li>
        <li class="green">Limit 10000 requests per month</li>
        <li class="grey" th:if="${role.getName()} != 'pro'"><a
href="/subscribe/pro" class="button">Update</a></li>
    </ul>
</div>
<div class="columns">
    <ul class="price">
        <li class="header">Enterprise</li>
        <li class="grey">ID card verification ✔</li>
        <li class="grey">Face verification ✔</li>
        <li class="green">Limit 250000 requests per month</li>
        <li class="grey" th:if="${role.getName()} != 'enterprise'"><a
href="/subscribe/enterprise" class="button">Update</a></li>
    </ul>
</div>

</body>
</html>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Payment page</title>
</head>

<body>
<style>
    /* Add a black background color to the top navigation */
    .topnav {
        background-color: #333;
        overflow: hidden;
```

```
}

/* Style the links inside the navigation bar */
.topnav a {
    float: left;
    color: #f2f2f2;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
    font-size: 17px;
}
.topnav .name {
    float: right;
    color: #f2f2f2;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
    font-size: 17px;
}

/* Change the color of links on hover */
.topnav a:hover {
    background-color: #ddd;
    color: black;
}

/* Add a color to the active/current link */
.topnav a.active {
    background-color: #04AA6D;
    color: white;
}
* {
    box-sizing: border-box;
}

/* Create three columns of equal width */
.columns {
    float: left;
    width: 33.3%;
    padding: 8px;
}

/* Style the list */
.price {
    list-style-type: none;
    border: 1px solid #eee;
    margin: 0;
    padding: 0;
    -webkit-transition: 0.3s;
    transition: 0.3s;
}

/* Add shadows on hover */
.price:hover {
    box-shadow: 0 8px 12px 0 rgba(0,0,0,0.2)
}

/* Pricing header */
.price .header {
    background-color: #111;
    color: white;
    font-size: 25px;
}
```

```
    /* List items */
    .price li {
        border-bottom: 1px solid #eee;
        padding: 20px;
        text-align: center;
    }

    /* Grey list item */
    .price .grey {
        background-color: #eee;
        font-size: 20px;
    }

    /* The "Sign Up" button */
    .button {
        background-color: #04AA6D;
        border: none;
        color: white;
        padding: 10px 25px;
        text-align: center;
        text-decoration: none;
        font-size: 18px;
    }

    /* Change the width of the three columns to 100%
    (to stack horizontally on small screens) */
    @media only screen and (max-width: 600px) {
        .columns {
            width: 100%;
        }
    }
</style>
<div class="topnav">
    <a class="active" href="/home">Home</a>
    <a href="/token">Token</a>
    <a href="/details">Details</a>
    <a href="/demo">Demo</a>
    <a th:text="${user.getUsername()} + ' (Logout)'" href="/logout"
class="name"></a>
</div>
    <center>
        <p>Click the button under to proceed payment</p>
        <div th:remove="tag" th:utext="${html}"></div>
    </center>
</body>
</html>
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" >
<head>
    <meta charset="UTF-8">
    <title>Results of comparison</title>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css">
</head>
<body>

<h1>Validation results</h1>
<style>table {
    border-collapse: collapse;
    border-spacing: 0;
    width: 100%;
    border: 1px solid #ddd;
}
```

```
/* Style table headers and table data */
th, td {
    text-align: center;
    padding: 16px;
}

th:first-child, td:first-child {
    text-align: left;
}

/* Zebra-striped table rows */
tr:nth-child(even) {
    background-color: #f2f2f2
}

.fa-check {
    color: green;
}

.fa-remove {
    color: red;
}</style>
<table>
    <tr th:each="user : ${users.entrySet()}">
        <td th:text="${user.key}">keyvalue</td>
        <td>
            <i th:if="${user.value} == false" class="fa fa-remove"></i>
            <i th:if="${user.value} == true" class="fa fa-check"></i>
        </td>
    </tr>
</table>

</body>
</html>
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" >
<head>
    <meta charset="UTF-8">
    <title>Token generation</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
<style>
    .flip-card {
        background-color: transparent;
        width: 600px;
        height: 300px;
        border: 1px solid #f1f1f1;
        perspective: 1000px;    }

    .flip-card-inner {
        position: relative;
        width: 100%;
        height: 100%;
        text-align: center;
        transition: transform 0.8s;
        transform-style: preserve-3d;
    }


    .flip-card:hover .flip-card-inner {
        transform: rotateY(180deg);
    }
```

```
.flip-card-front, .flip-card-back {
    position: absolute;
    width: 100%;
    height: 100%;
    -webkit-backface-visibility: hidden; /* Safari */
    backface-visibility: hidden;
}

/* Style the front side (fallback if image is missing) */
.flip-card-front {
    background-color: #bbb;
    color: black;
}

/* Style the back side */
.flip-card-back {
    background-color: #04aa6d;
    color: white;
    transform: rotateY(180deg);
}
/* Add a black background color to the top navigation */
.topnav {
    background-color: #333;
    overflow: hidden;
}

/* Style the links inside the navigation bar */
.topnav a {
    float: left;
    color: #f2f2f2;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
    font-size: 17px;
}
.topnav .name {
    float: right;
    color: #f2f2f2;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
    font-size: 17px;
}

/* Change the color of links on hover */
.topnav a:hover {
    background-color: #ddd;
    color: black;
}

/* Add a color to the active/current link */
.topnav a.active {
    background-color: #04AA6D;
    color: white;
}
* {
    box-sizing: border-box;
}


/* List items */
.price li {
    border-bottom: 1px solid #eee;
```

```
        padding: 20px;
        text-align: center;
    }


    /* The "Sign Up" button */
    .button {
        background-color: #ffffff;
        border: none;
        color: #000000;
        padding: 10px 25px;
        text-align: center;
        text-decoration: none;
        font-size: 18px;
    }
    input[type=text], input[type=password] {
        width: 75%;
        padding: 15px;
        margin: 5px 0 22px 0;
        display: inline-block;
        border: none;
        background: #f1f1f1;
    }
</style>
<script>
    function myFunction() {
        var x = document.getElementById("myInput");
        if (x.type === "password") {
            x.type = "text";
        } else {
            x.type = "password";
        }
    }
    function callSnackbar() {
        var x = document.getElementById("snackbar");

        // Add the "show" class to DIV
        x.className = "show";

        // After 3 seconds, remove the show class from DIV
        setTimeout(function(){ x.className = x.className.replace("show", "");
}, 3000);
    }
</script>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></scri
pt>
<script type="text/javascript">
    $(document).ready(function() {
        $('button[id="generate"]').click(function() {
            $.ajax({
                method : 'GET',
                dataType: "text",
                url : '/token_generate',
                success: function(data) {
                    callSnackbar();
                    $('#myInput').val(data);
                }
            });
        });
    });
</script>
<div class="topnav">
    <a href="/home">Home</a>
```

```
    <a class="active" href="/token">Token</a>
    <a href="/details">Details</a>
    <a href="/demo">Demo</a>
    <a th:text="${user.getUsername()} + ' (Logout)'" href="/logout"
class="name"></a>
</div>
<div class="flip-card">
    <div class="flip-card-inner">
        <div class="flip-card-front">
            <center>
                <img src="https://icons.veryicon.com/png/o/system/system-
project/secret-key-2.png" alt="Avatar" style="width:30%;height:30%;padding-
top:10%">
            </center>
        </div>
        <div class="flip-card-back">
            <h1 th:text="'Hello, ' + ${user.getUsername()}"></h1>
            <p><input type="password" value="Empty token"
th:value="${user.getToken()}" id="myInput"></p>
            <p><input type="checkbox" onclick="myFunction()">Show token</p>
            <p><button id="generate" class="button">Generate
token</button></p>
            <div id="snackbar">Updated successfully</div>
        </div>
    </div>
</div>

</body>
</html>
```