Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Веб-орієнтована система гібридного механізму управління зображеннями для їх ефективної обробки та аналізу в умовах великих даних. Back-end
(тема)

Виконав:
студент 4 курсу, групи ПЗПІ-20-8

Мандзинець А.В.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник ст. викл. каф. ПІ Терещенко Г.Ю.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва)
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Мандзинцю Анатолію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Веб-орієнтована система гібридного механізму управління зображеннями для їх ефективної обробки та аналізу в умовах великих даних. Back-end»

Затверджена наказом університету від 20 травня 2024 р. № 471Ст

2. Термін подання студентом роботи до екзаменаційної комісії 17 червня 2024

3. Вихідні дані до роботи Розробити серверну частину для веб-орієнтованої системи гібридного механізму управління зображеннями для їх ефективної обробки та аналізу в умовах великих даних

4. Перелік питань, що потрібно опрацювати в роботі вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	08.04.2024	*виконано*
2	Створення специфікації ПЗ	15.04.2024	*виконано*
3	Проектування ПЗ	20.04.2024	*виконано*
4	Розробка ПЗ	17.05.2024	*виконано*
5	Тестування ПЗ	27.05.2024	*виконано*
6	Оформлення пояснювальної записки	29.05.2024	*виконано*
7	Підготовка презентації та доповіді	03.06.2024	*виконано*
8	Попередній захист	10.06.2024	*виконано*
9	Нормоконтроль, рецензування	12.06.2024	*виконано*
10	Здача роботи у електронний архів	13.06.2024	*виконано*
11	Допуск до захисту у зав. кафедри	14.06.2024	*виконано*

Дата видачі завдання 16 травня 2024р.

Студент (ка)_____ Мандзинець А.В.
(підпис)

Керівник роботи _____ ст. викл. каф. ПІ Терещенко Г.Ю.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра, 68 стор., 30 рис., 1 табл., 18 джерел.

АРХІТЕКТУРА, ВЕЛИКІ ДАНІ, ВЕБ, ЗОБРАЖЕННЯ, КЛАСИФІКАЦІЯ, КОМП'ЮТЕРНИЙ ЗІР, СХОВИЩЕ, AZURE, JAVASCRIPT.

Об'єкт розробки – веб-орієнтована система гібридного механізму управління зображеннями для їх ефективної обробки та аналізу в умовах великих даних (back-end).

Мета розробки – створити серверну частину для веб-орієнтованої системи гібридного механізму управління зображеннями для їх ефективної обробки та аналізу в умовах великих даних.

Метод рішення – мова програмування JavaScript, фреймворк Node.js, СУБД Azure Cosmos DB, сховище Azure Blob Storage.

В результаті розробки створено серверну частину для веб-орієнтованої системи гібридного механізму управління зображеннями для їх ефективної обробки та аналізу в умовах великих даних.

ARCHITECTURE, BIG DATA, WEB, IMAGES, CLASSIFICATION, COMPUTER VISION, STORAGE, AZURE, JAVASCRIPT.

The object of development is a web-oriented system of a hybrid image management mechanism for their effective processing and analysis in the context of big data (back-end).

The purpose of the development is to create a server part for a web-oriented system of a hybrid image management mechanism for their effective processing and analysis in the context of big data

Solution method – JavaScript programming language, Node.js framework, Azure Cosmos DB database, Azure Blob Storage.

As a result of the development, a server part was created for a web-oriented system of a hybrid image management mechanism for their effective processing and analysis in the context of big data.

Я, Мандзинець Анатолій Володимирович, студент гр. ПЗПІ-20-8, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Веб-орієнтована система гібридного механізму управління зображеннями для їх ефективної обробки та аналізу в умовах великих даних. Back-end», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Темою кваліфікаційної роботи є розробка back-end частини веб-орієнтованої системи гібридного механізму управління зображеннями для їх ефективної обробки та аналізу в умовах великих даних.

Фото та зображення давно стали невід'ємною частиною нашого життя. Розвиток камер на телефонах, покращення якості передачі даних в мережі Інтернет, поява соціальних мереж вплинули на те, що нині тільки в Інтернеті налічується близько 136 мільярдів зображень.

Крім цього в кожного з нас на власних пристроях зберігається значна кількість зображень, вони можуть зберігатися в нас для пам'яті, для роботи або освіти. Через це дуже часто нам стає важко знаходити потрібні фото або місце, яке вони займають, стає непомірним.

Виходом із такої ситуації стали сховища для зображень. Деякі з них працюють як хмарне сховище, які синхронізують дані на пристрої та в хмарі. Іншим варіантом є програмне забезпечення, що постачається з пристроями для локального зберігання зображень. Також для цього часто використовуються соціальні мережі або месенджери.

Проте всі вони окремо мають ряд недоліків:

– проблеми з конфіденційністю;

– погіршення якості зображення;

– повільна швидкість завантаження;

– неефективне використання пам'яті при зберіганні даних;

– відсутність інструментів для автоматичного аналізу зображень;

– незручний інтерфейс.

Метою роботи є розробка серверної частини програмної системи для гібридного механізму управління зображеннями для їх ефективної обробки та аналізу в умовах великих даних. Для реалізації цього використовуються мова програмування JavaScript, база даних Azure Cosmos DB, сервіси Azure, середовище розробки WebStorm.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Важко уявити наше сьогоднішнє життя без зображень, фото, картинок. Бажання залишити назавжди зображення якоїсь миті існувало у людства здавна. І фактично, тільки поява фотографії кілька століть тому зробила це можливим.

З цього часу фотографія стала невід'ємною частиною нашого життя, нашої культури. Буквально формуючи наше сприйняття світу та взаємодію одне між одним. Такий сильний вплив не є дивним, адже близько 85% всієї інформації ми сприймаємо за допомогою зору.

Тобто, сьогодні фото – це пам'ять про важливі моменти нашого життя, особистого та суспільного життя. Даючи на змогу знову й знову переживати ці моменти. Це спосіб самовираження та натхнення. Важливий елемент досліджень у різних науках – від мікроскопічних зображень у медицині до високоякісних зображень чорної діри у космосі. Також слід згадати соціальні мережі для яких зображення є важливим способом комунікації, обміну ідей та емоцій.

Значний вплив на зображення мав Інтернет. Мережею стали не тільки обмінюватися зображеннями, а й використовувати її як сховище для них.

Наприклад, у 2004 році Ludicorp, компанією з Ванкувера було запущено Flickr. Ранні версії Flickr були зосереджені на чаті під назвою FlickrLive з можливостями обміну фотографіями в реальному часі. Пізніше обмін фотографіями розвинувся в головну функцію. З'явилися додавання тегів до фотографій, позначення фотографій як вибраних, групові пули фотографій і аналіз «цікавості», на який було видано патент. Уже у 2005 сервіс було придбано компанією Yahoo! за $22-25 млн. [1].

В цей же час розвивався інший сервіс Photobucket, призначений для зберігання мультимедійних файлів з можливістю конвертації в будь-який формат. Спочатку був створений сервіс для обміну фотографіями. Однак виявилося, що користувачі воліють не стільки обмінюватися, скільки зберігати на ньому свої знімки і розміщувати посилання на них на інших сайтах. Звернувши увагу на подібну поведінку споживачів, засновники не стали вводити систему заборон, а,

вивчивши дії користувачів, дали їм можливість завантажувати фотографії і розміщувати посилання на них на будь-яких форумах [2].

Слід також згадати про сервіс Picasa створений у 2002 та придбаний компанією Google у 2004. Продукт дуже вдало розвивався та мав широкий функціонал для роботи з цифровими зображеннями. Сервіс дозволяв імпортувати та відстежувати файли, розпізнавати обличчя, додавати теги, колекціонувати та сортувати. Наявні також функції редагування фотографій, зокрема покращення кольору, зменшення ефекту червоних очей і кадрування, зменшення розміру, геотегування. Інші функції включають слайд-шоу, друк і графіки зображень. Фактично, сьогодні ми знаємо цей сервіс як Google Photo [3].

Тобто можна побачити, що потреба у сервісах для зберігання зображень з'явилася давно. Крім того вплив мали ще деякі фактори. Перш за все, розповсюдження смартфонів, оснащених якісними камерами, дозволило людям неймовірно легко знімати та миттєво ділитися фотографіями. Іншою важливою причиною став розвиток соціальних мереж таких як Instagram, Facebook, X (Twitter), засновані на публікації медіа-контенту значну частину якого складають зображення.

Користувачі таких сервісів завантажують в них зображення мільйонами. Для розуміння, за приблизними оцінками, Instagram щоденно збільшується на 95 мільйонів зображень, у Facebook та X цифри приблизно такі ж. Загалом на 2022 рік в Інтернеті нараховували близько 136 мільярдів зображень (реальна цифра може бути у 10 разів більша) [4].

При цьому розвиток хмарних технологій дозволив не турбуватися про обмеження фізичного простору. На світ з'явилися такі сервіси як Google Photo, Apple iCloud, Amazon Photo, що дають можливість працювати з власним хмарним сховищем.

Таким чином, ми маємо величезну кількість зображень, що постійно збільшується, а також потребу серед користувачів у зберіганні цих даних. Фактично, коли ми говоримо про мільярди зображень можна використати термін Великі дані (Big Data). Мається на увазі великий об'єм неструктурованих або

напівстуктурованих даних, якими є зображення. Обробка та анаілз таких зображень засобами штучного інтелекту для отримання певної інформації застосовується у різних сферах, таких як медицина, виробництво електроніки, освіта, океанографія [5].

1.2 Виявлення проблем та актуалізація рішень

Розглянемо спочатку згадані вище сервіси Google Photo, Apple iCloud, Amazon Photo, що дають можливість працювати з власним хмарним сховищем. Окрім зручного доступу та роботи з великою кількістю зображень, ці програмні системи також надають додаткові можливості такі як поділитися фото з іншими користувачами, створити резервні копії, синхронізувати дані, редагувати, використовувати штучний інтелект для пошуку обличь (див. рис. 1.1), класифікації та кращого пошуку зображень.

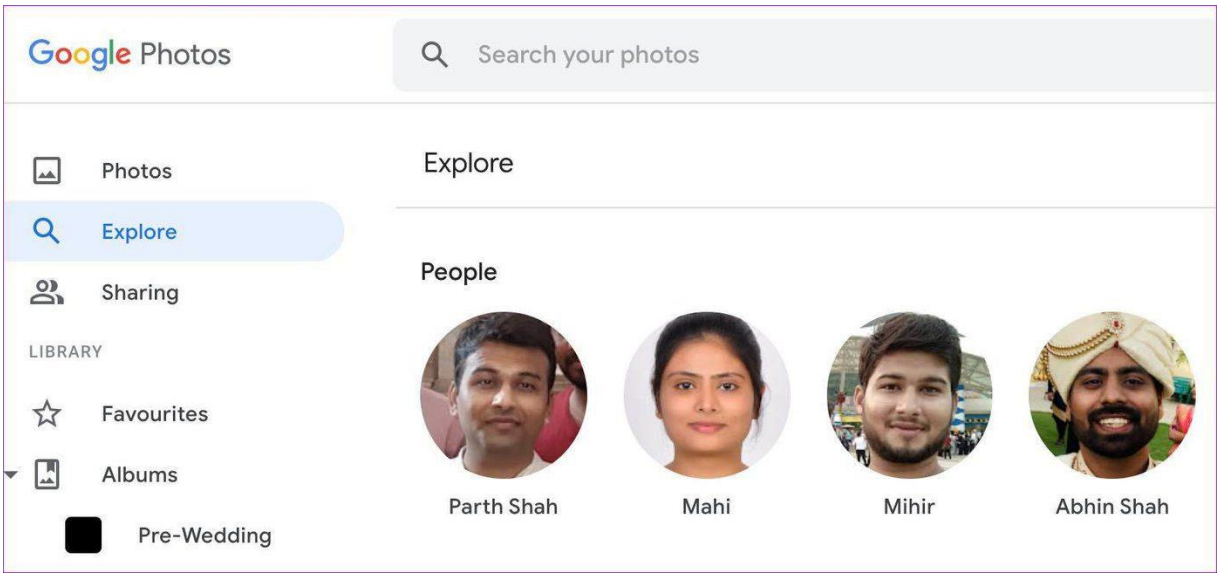


Рисунок 1.1 – Розпізнавання обличь у Google Photos (за даними [6])

Це дійсно вражаючі інструменти, проте користувачі мають повністю покладатися на провайдера цих послуг в плані безпеки або конфіденційності, наприклад, при використанні моделей, що розпізнають об’єкти на фото. Також завжди є певні втратити дані при зміні політик або припинення роботи сервісу. Наприклад, так трапилося з Yahoo! Photo, які змушували переходити користувачів на Flickr, бо сам сервіс припиняв свою роботу.

Якщо розглядати такі сервіси як Instagram, Facebook або Pinterest для власних зображень, часто можна помітити, що вони стискають зображення. Тобто якість вашого зображення може відчутно погіршиться.

Іншим вибором для збереження фото є власний хостинг NAS (Network attached storage). В цій ніші також наявні немало пристроїв, які постачаються із вбудованим програмним забезпеченням для роботи із зображеннями. Наприклад, Synology, що надає широкий функціонал роботи з фото, проте використовувати його можна виключно на мережевому обладнанні від Synology.

Більш доступними альтернативами є open source застосунки для локального сховища: Immich, PhotoPrism, Lychee, Librephotos. Очевидними перевагами такого підходу є повний контроль системи, конфіденційність та вартість. Проте й недоліків у цьому випадку немало.

Перш за все, необхідні технічні компетентності для установки та роботи з такою системою. По-друге, часто такі застосунки вимагають значних ресурсів для роботи, яких не завжди вистачає. Також слід розуміти, що оскільки це відкриті проекти, які підтримуються спільнотою, існує ймовірність, що їх в якийсь момент перестануть підтримувати. Крім цього, часто ці додатки мають деякі недоліки, наприклад, погана робота розпізнавання об’єктів на фото, повільне завантаження та пошук, неможливість поділитися даними з іншими користувачами, а інколи – складний або незрозумілий інтерфейс (див. рис. 1.2).

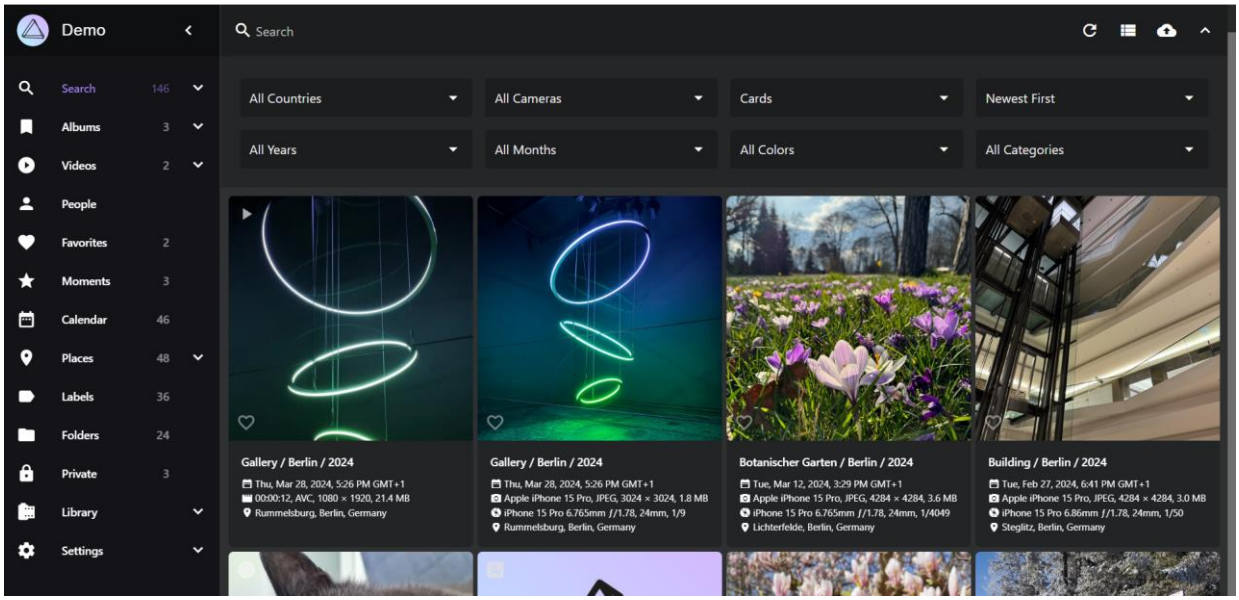


Рисунок 1.2 – Сторінка із зображеннями у PhotoPrism (за даними [7])

Крім цього дуже поширеною є проблема передачі великої кількості даних при початковому завантаженні зображень у застосунок. Це перш за все впливає на час завантаження. Слід звернути увагу на ефективність використання пам’яті при зберіганні такої кількості зображень. Це також має вплив на швидкість виконання пошуку, відображення зображень на клієнтській частині або завантаження (download) зображень на власний пристрій.

1.3 Постановка задачі

Для вирішення потреб користувачів пропонується розробити веб-орієнтовану систему гібридного механізму управління зображеннями для їх ефективної обробки та аналізу в умовах великих даних.

Основною цільовою аудиторією будуть:

– особи, які хочуть безпечно зберігати та впорядковувати свої особисті зображення;
– аматори і професійні фотографи, якім потрібна платформа для зберігання своїх зображень;
– компанії, яким потрібен репозиторій для зберігання та керування зображеннями документації або реклами.

Виходячи з аналізу предметної області та огляду існуючих аналогів можна зробити висновок, що завдання зберігання великої кількості зображень є актуальним та корисним. Ця сфера давно розвивається і сьогодні доступно багато готових рішень, як від великих компаній (Google, Amazon) так і open source. Проте в кожного з них можна знайти якісь недоліки:

– проблеми з конфіденційністю;
– вартість або обмеження у використанні;
– втрата якості зображення;
– повільна робота.

Для розробки нової системи потрібно зосередитися на можливості автоматичної обробки зображень, наприклад, стиснення без помітної втрати якості для:

– швидшого завантаження багатьох зображень;

– ефективного зберігання великої кількості даних.

При завантаженні слід додати функції аналізу зображень:

– присвоєння зображенню певний клас (класифікація);

– додавання тегів до зображення на основі вмісту;

– розпізнавання тексту на зображенні.

Оскільки робота буде вестися із великими даними, слід подбати про гібридний механізм управління даними. Мається на увазі підхід до зберігання зображень та даних про них в окремих сервісах: сховищі для бінарних об’єктів та базі даних. Цей гібридний підхід оптимізує роботу та продуктивність сервісів, використовуючи кожен з них за призначенням.

Вирішивши ці проблеми та запропонувавши користувачам якісь нові функції можна вдало конкурувати з готовими системами. Таким чином, для вирішення виявлених проблем функціональність майбутньої системи буде охоплювати:

– створення облікового запису користувача;

– автоматичну класифікацію зображень при завантаженні;

– пошук за назвою та тегами зображення;

– завантаження багатьох зображень;

– отримання даних про зайнятий простір;

– можливість поділитися альбомом з іншими користувачами;

Серед нефункціональних вимог можна виділити:

– дотримання безпеки даних;

– низька затримка відповіді системи;

– зручний та інтуїтивний інтерфейс.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Вимоги до серверної частини

Відповідно до задач, які має вирішувати сервіс та його функціоналу можемо сформувати наступні вимоги:

– дозволяти працювати з даними користувача;

– отримувати дані про зайняту зображеннями пам’ять;

– додавати багато зображень за один раз;

– обробляти зображення (стискати);

– виконувати аналіз зображення;

– змінювати дані про зображення;

– завантажувати зображення;

– отримувати дані про зображення;

– виконувати пошук зображень;

– видаляти дані про зображення та альбоми;

– завантажувати багато зображень архівом;

– забезпечувати доступ до альбомів різних користувачів;

– дотримуватися безпеки при зберіганні паролів;

– дотримуватися механізмів аутентифікації користувача;

– повідомляти про помилки.

Серверна частина програмної системи буде реалізована на мові програмування JavaScript з використання фреймворка Node.js [8], він є легким, але при цьому потужним інструментом, Реалізуючи асинхронну модель запитів, що керується подіями та неблокуючий ввід/вивід, Node.js чудово підходить для серверів, що отримують велику кількість запитів та потребують високої продуктивності та масштабованості. Крім цього Node.js має велику кількість бібліотек для різноманітних задач. Однією з таких є Express, яку буде використано для простішої роботи із запитами та проміжним програмним забезпеченням веб-сервера.

2.2 Вимоги до служб зберігання даних

Сформуємо перелік основних вимог до бази даних:

- продуктивність – забезпечення швидкого часу відповіді в умовах великих даних;
- доступність – база має бути доступною, коли це необхідно, з мінімальним часом простою;
- масштабованість – можливість витримувати високі навантаження використовуючи додаткові ресурси;
- підтримування нереляційної моделі зберігання даних;
- безпека даних.

Сьогодні доступний широкий вибір баз даних, що задовільними б цим вимогам. Ми ж можемо зупинитися на Azure Cosmos DB. Це глобально розподілена, багатомодельна (підтримує різні моделі даних) розроблена саме для забезпечення високої доступності, масштабованості та низької затримки [9].

Ця база даних доступна як частина сервісів хмарної платформи Microsoft Azure. Крім цього Azure надає ще великий вибір різноманітних інструментів, тому для кращої інтеграції та простоти при розробці скористаємося сервісами цього провайдера для побудови інфраструктури програмної системи.

Як уже було сказано, в гібридному підході велика даних буде зберігатися окремо. Тобто є потреба у сховищі для файлів, що дозволятиме швидко та ефективно завантажувати зображення, забезпечуватиме їх цілісність та безпеку, буде легким у використанні. Для цього чудово підходить Azure Blob Storage.

2.3 Вимоги до сервісів аналізу зображень

Під аналізом зображень розуміється класифікація, додавання тегів і розпізнавання тексту. Для цього програмна система використовуватиме штучний інтелект. Серед вимог слід назвати доступність, безпеку та зручність. Використання для таких задач натренованих моделей може потребувати значних ресурсів. Отож, для більш ефективної роботи та зменшення часу запиту можна скористатися сервісом Azure AI Vision, що надає всі перелічені можливості.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

Для проектування програмного забезпечення скористаємось Unified Modeling Language (UML). Це уніфікована мова моделювання, що є важливою частиною процесу розробки програмного забезпечення. Для роботи використовуються графічні позначення для створення абстрактної моделі системи [10].

Для початку розглянемо діаграму прецедентів. Вона демонструє відношення між акторами та прецедентами в системі. За допомогою неї можна більш детально аналізувати вимоги та визначати функціональність.

Основними компонентами такої діаграми являються актори, прецедент (use case), система та відношення. Актор – це хтось або щось, що взаємодіє із системою, але входить до неї. Прецедент можна описати, як поведінку або послуги, які надає система.

Відношення пов’язуються акторів із варіантами використання або варіанти використання між собою. Прийнято виділяти 4 основних типів відношень. Першим є асоціація найчастіше ним показують зв’язок між актором та варіантом використання. Другим видом зв’язку є включення (include) він показує, що один варіант включається як складовий компонент прецеденту. Виключення показує додаткову функціональність або можливий не обов’язковий варіант поведінки системи. І останній вид зв’язку – це узагальнення або успадкування, тобто батьківсько-дочірні відношення.

Ґрунтуючись на описане в попередніх розділах для проектування архітектури програмної системи розглянемо діаграму прецедентів (див. рис. 3.1).

Як можна побачити з діаграми, основними акторами у даній програмній системі є користувач та адміністратор. Основні дії користувача націлені на роботу з зображеннями та альбомами.

Користувач реєструється у застосунку, після цього має авторизуватися, щоб перейти до користування системою. Далі користувачу доступний перегляд та зміна власних даних. Також тепер користувач може завантажувати у застосунок власні

зображення, переглядати створені альбоми, виконувати пошук зображень, змінювати їх, поділитися альбомом з іншими користувачами, завантажити його або видалити.

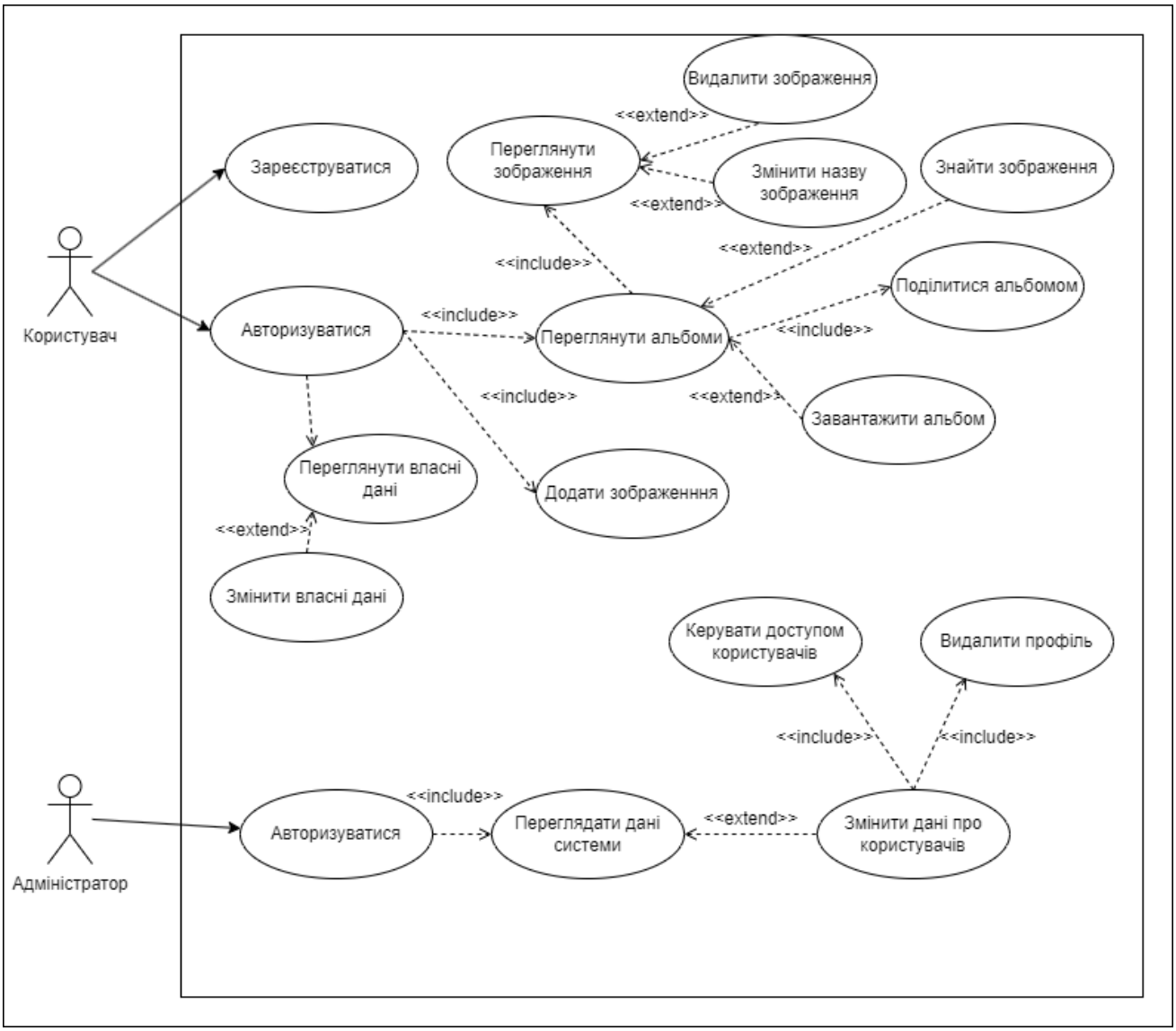


Рисунок 3.1 – Діаграма прецедентів (виконано самостійно)

Робота ж адміністратора полягає в перегляді даних про систему, видаленні та блокуванні користувачів. Дані про систему включають в себе список всіх користувачів програмної системи, кількість зображень та зайнятої пам’яті у кожного.

Скористаємось побудовою діаграми станів для проєктування програмного забезпечення. Ця діаграма визначає зміну станів об'єкту у часі та подається у вигляді автоматів зі стандартними умовними позначеннями. Фактично, стан

представляє собою набір значень атрибутів об’єкта якогось класу. Зміна значень таких атрибутів показує зміну станів об’єкта.

На діаграмі використовується декілька основних елементів. Перш за все це стан, він позначається прямокутником з округленими вершинами. Цей прямокутник завжди містить назву стану, а також може мати внутрішні дії. Стрілками позначаються переходи між станами.

Продемонструємо таку діаграму на рис. 3.2. В ній розглянуто основні стани системи при додаванні зображень користувачем. Тут показано процес обробки та аналізу, цей стан містить в собі окрему послідовність внутрішніх станів, які мають свій початок та кінець.

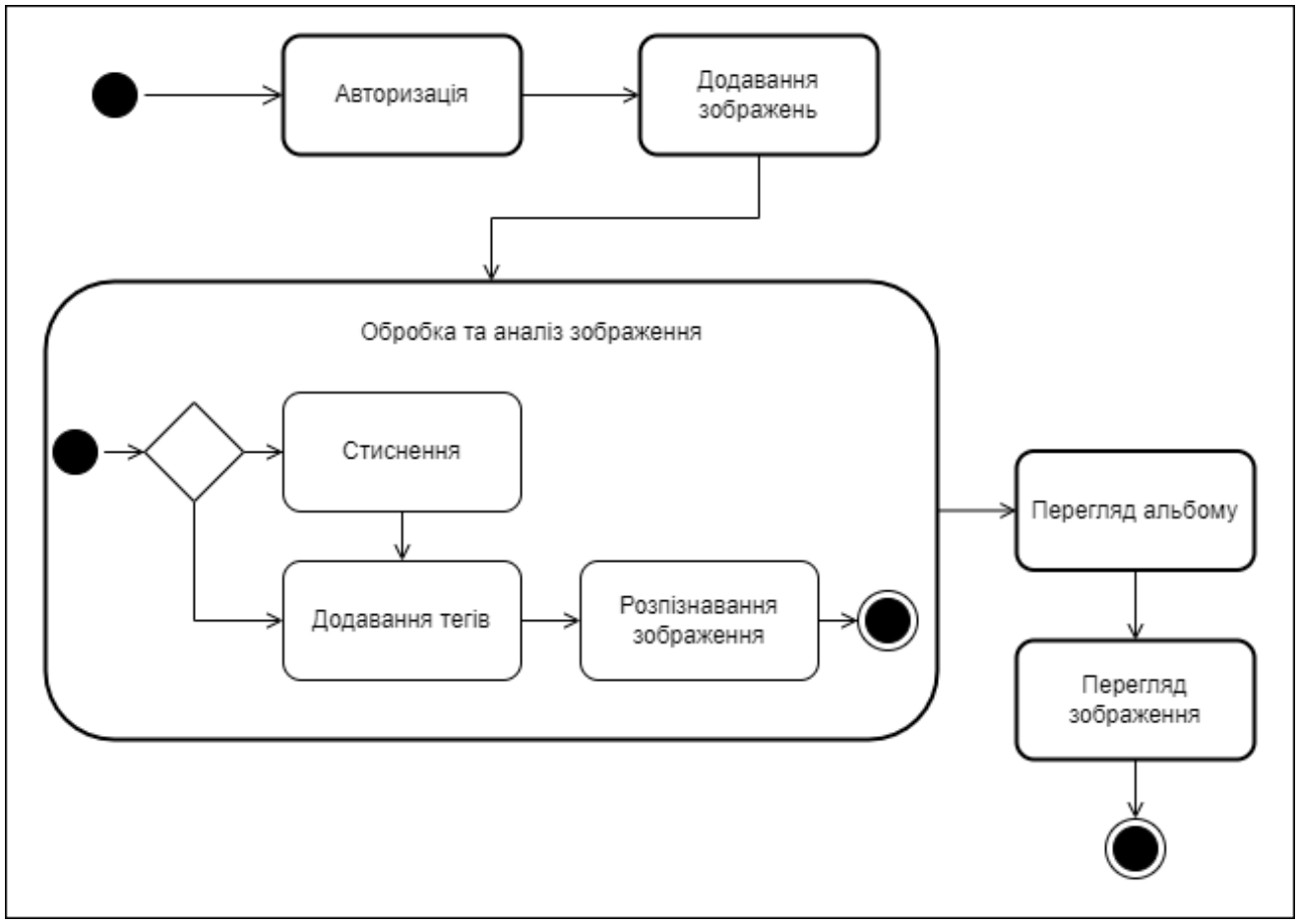


Рисунок 3.2 – Діаграма станів (виконано самостійно)

Тобто, із послідовності стану системи можна помітити, що для початку роботи кожен користувач має авторизуватися. Ні в якому іншому випадку він не

зможе почати працювати із зображеннями. Тільки після цього відбувається додавання зображення.

Додати можливо багато зображень, яким буде присвоєно теги, в порядку найбільш підходящих, перший з них буде обраний для позначення класу зображення. Також при наявності тексту на зображені він буде розпізнаний. Слід зауважити, що якщо зображення матиме великий розмір його буде стиснуто. Це пришвидшить завантаження зображення на сервер, його аналіз та дозволить ефективніше використовувати об’єм сховища.

Після цього користувач зможе перейти до альбомів, що створюються автоматично. Кожен з них міститиме один клас зображень, наприклад тварини, природа або будівлі, залежно від того, які значення поверне аналіз зображень.

3.2 Проєктування архітектури програмного забезпечення

Архітектура програмного забезпечення є важливим етапом під час проєктування. Вона має включати всі компоненти системи, їхні зв’язки та те, як вони взаємодіють. Правильний підхід до процесу проєктування дає більш чітке розуміння розроблюваної системи, визначення показників якості, гнучкості при розробці та легкості для підтримування коду у майбутньому.

До основних архітектурних стилів програмного забезпечення відносяться клієнт-сервер, компонентна, дизайн на основі предметної області, багато-рівнева архітектура. Кожна з них має свої переваги та недоліки і може знайти застосування у певній сфері залежно від заданих вимог.

Виходячи з вимог, описаних вище до нашої веб-орієнованої системи архітектура майбутньої програмної системи буде трирівнева (можна назвати підвидом багато-рівневої архітектури).

На рис. 3.3 схематично показано принцип за яким функціонує ця архітектура.

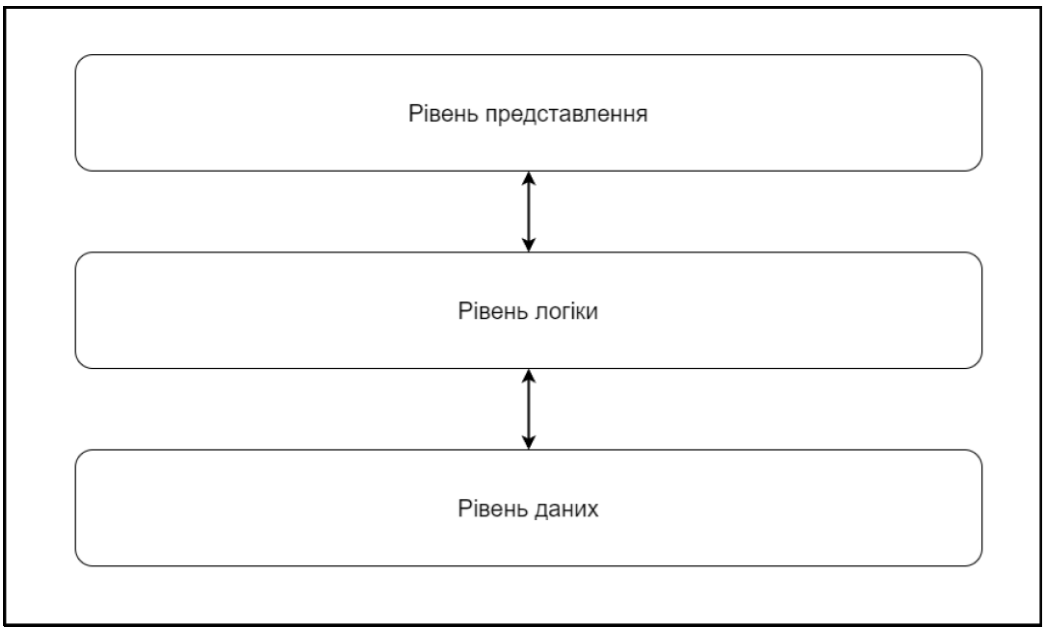


Рисунок 3.3 – Трирівнева архітектура (виконано самостійно)

Як можна побачити трирівнева архітектура – це модель побудови архітектури програмного забезпечення, що включає в себе три основних компоненти: клієнтський застосунок, серверну частину та сервер бази даних. Прийнято також розміщувати ці компоненти окремо на різних пристроях.

Ця архітектура має ряд переваг:

– масштабованість: очевидно, що розділивши один великий додаток на декілька менших частин, ми зможемо легко справлятися з навантаженням на окремі його компоненти, виділивши на них більше ресурсів. Наприклад, додати новий екземпляр бази даних, якщо навантаження на один буде перевищувати певне значення;
– модульність: надає легкий підхід до розробки, тестування та модифікації окремих частин системи. Наприклад, можна повністю замінити один із компонентів без зміни бізнес-логіки або рівня даних, доки інтерфейси між рівнями залишаються узгодженими;
– надійність: оскільки різні компоненти системи працюють незалежно, то вихід з ладу одного з них не стане критичним для всієї системи;
– безпека: рівень представлення (клієнт) не має прямого доступу до бази даних, тому зловмисникам стає важче завдати шкоди системі.

Виділимо тепер основні компоненти нашої веб-орієнтованої програмної системи:

– сервер бази даних або рівень даних: цей рівень відповідає за керування та зберігання даних програми;
– веб-сервер або рівень бізнес-логіки: він є посередником між рівнем презентації та рівнем даних, забезпечуючи цілісність і безпеку даних. Отримуючи запити від рівня представлення, обробляє їх відповідно до бізнес-логіки програми;
– інтерфейс або рівень представлення: цей рівень відповідає взаємодію з користувачами, представлення інтерфейсу програми та відправлення даних на рівень бізнес-логіки.

В нашому випадку окремо також буде виділено сервіс для роботи з класифікацією зображень – Azure AI Vision.

Використаємо діаграму розгортання (див. рис. 4.4), на якій зображуються основні компоненти програмної системи

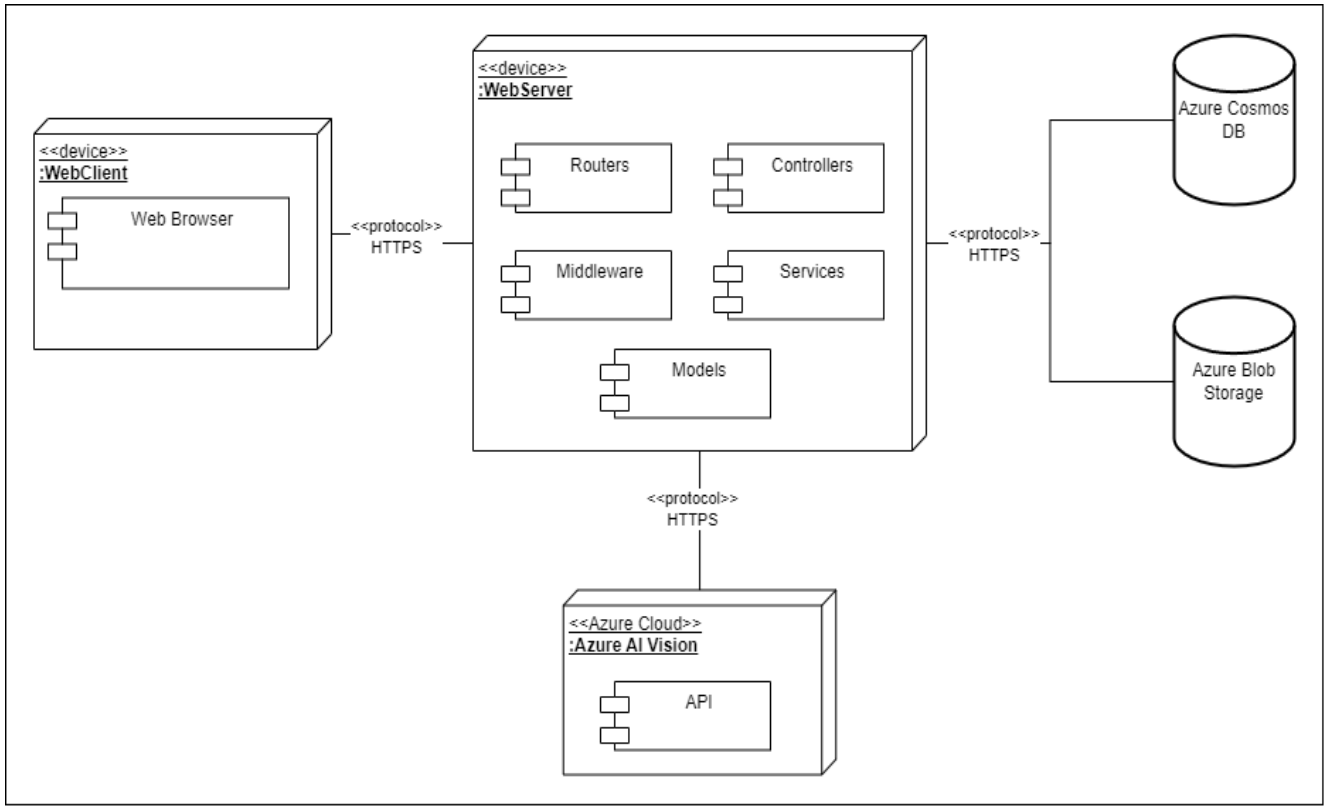


Рисунок 3.4 – Діаграма розгортання (виконано самостійно)

Діаграма покращує розуміння системи, планування розгортання, детальніше показує компоненти та способи комунікації між ними. На ній добре видно схематичне зображення компонентів, що були описані вище та їх взаємодію. Можна побачити всі компоненти розміщуються окремо та взаємодіють між собою через HTTP запити.

Серверна частина програмної системи буде реалізована на мові програмування JavaScript з використання фреймворку Node.js та Express. Для проєктування архітектури цього компонента системи буде використано багатошарову архітектуру. Вона є чудовим прикладом легкої архітектури, яка не перевантажена нічим лишнім, забезпечує відділення логічних шарів та просту взаємодію між ними.

Розглянемо основні компоненти такої архітектури:

– routers: приймає всі запити, що надходять на сервер та викликає необхідні функції у контролері;
– controllers: його основна функція – це делегування роботи необхідним сервісам та повернути результат;
– services: містить опис функцій пов’язаних з окремою сутністю або доменом;
– models: містить класи, через які відбувається взаємодія із базою даних.

Таке розділення та ізоляція шарів надає кращі можливості для тестування або зміни окремих шарів (наприклад, бази даних) не зачіпаючи інші.

3.3 Проєктування структури зберігання даних

Предметна область, задачі якої вирішує програмна система, включають в себе зберігання декількох типів даних: дані користувачів та їх файли (зображення). Загалом існує декілька підходів для зберігання зображень [11].

Перший це зберігання зображень у базі даних у вигляді двійкових файлів. Цей метод є простим і ефективним, але він може бути невигідним, коли ми говоримо про зображення в умовах великих даних.

Інший підхід – це зберігати зображення на пристрої, а шлях до файлу зберігати у базі даних. Цей метод забезпечує легкий доступ до зображень, але він дуже залежить від доступних ресурсів пристрою, через що може стати не ефективним.

Третій підхід це полягає у використанні служби хмарного зберігання, наприклад Amazon S3 або Azure Blob Storage. Це передбачає збереження зображень у хмарному сервісі, а потім збереження URL-адрес у базі даних. Такий метод має високу масштабованість і забезпечує легкий доступ до зображень з будь-якого місця. Хоча мінусом такого рішення може стати його вартість.

В нашій програмній системі оберемо гібридний підхід до зберігання таких даних. Для зберігання даних про користувачів та зображення використаємо нереляційну базу даних. NoSQL бази даних добре підходять роботи в умовах Big Data [12]. Самі файли ж файли будемо у хмарному сховищі.

В якості бази даних було обрано Azure Cosmos DB, вона задовольняє всі вимоги для роботи у високонавантажених системах. Це нереляційна база даних, що є одним із продуктів Azure. Вона є глобально поширеною, що забезпечує її високу доступність та синхронізацію у різних регіонах.

Cosmos DB є горизонтально масштабованою, за потреби їй можна легко виділити додаткові ресурси. База надає декілька рівнів узгодженості даних, що дозволяє тримати правильний баланс між узгодженістю та продуктивністю. Також, за замовчуванням Azure Cosmos DB автоматично індексує всі дані, що зберігаються у базі.

Azure Cosmos DB пропонує кілька API баз даних таких як NoSQL, MongoDB, PostgreSQL, Cassandra, Gremlin і Table. Для розроблюваної системи зупинимося на API для NoSQL, який є рідним для Azure Cosmos DB, а тому має кращу підтримку, інтерфейс для роботи з даними, інтеграцію з іншими сервісами Azure та мовами програмування.

Azure Blob Storage – це хмарний сервіс для з Microsoft Azure. Він призначений для зберігання величезних обсягів неструктурованих даних, таких як текстові або двійкові дані. Сховище забезпечує декілька рівнів зберігання (гарячі,

холодні та архівні) та має високу масштабованість, доступність та рівень безпеки, підтримує резервування [13].

Загалом ці технології разом чудово підходять для зберігання великих даних. Володіють широким функціоналом та можливостями, а також добре інтегровані з іншими технологіями.

Використаємо ER-діаграму, яка допомагає описати концептуальну схему предметної області. На основі неї пізніше буде створена база даних, яка використовує нереляційну модель даних.

На рис. 3.5 показано ER-діаграму програмної системи.

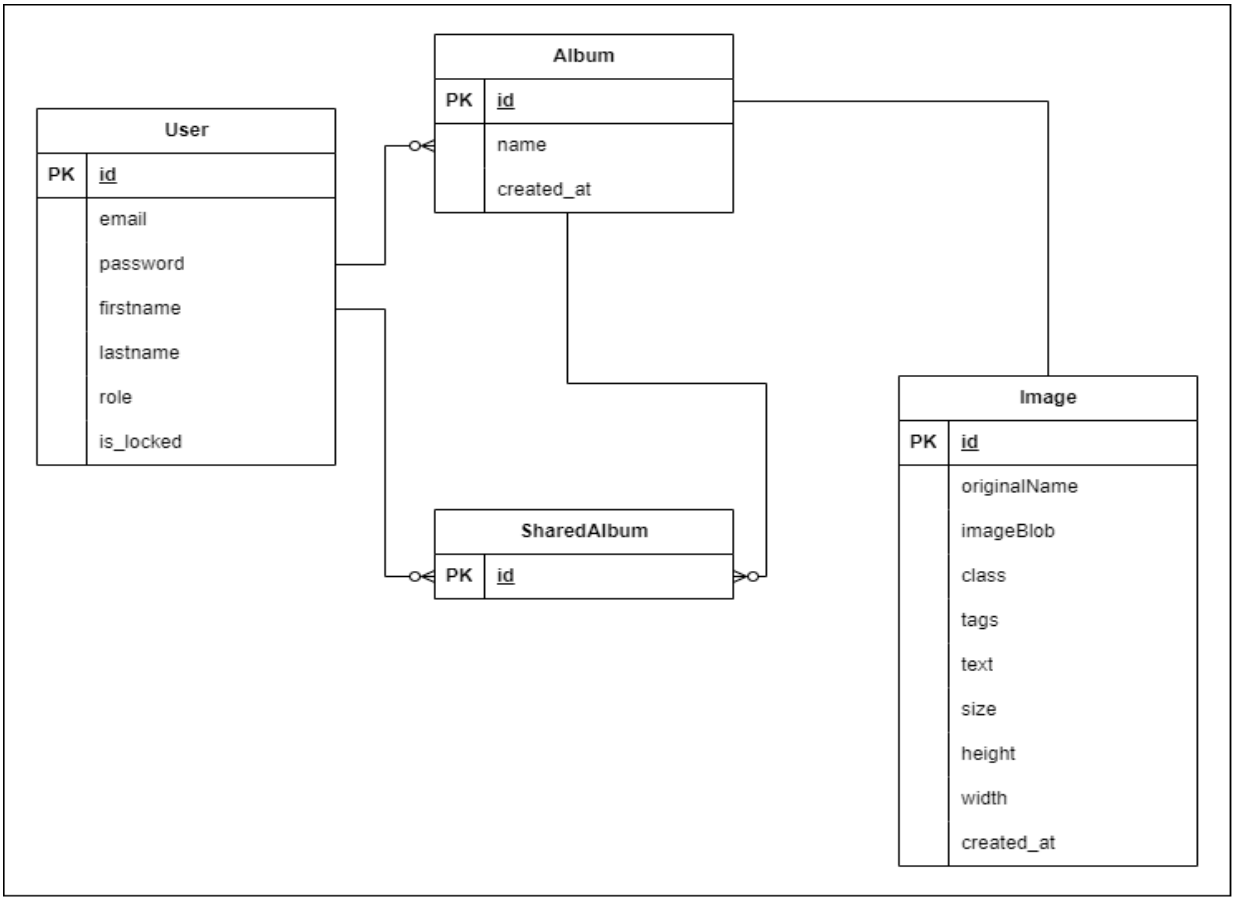


Рисунок 3.5 – ER-діаграма програмної системи (виконано самостійно)

Тепер розглянемо детальніше зображені сутності:

– Users: призначена для зберігання всіх даних про користувача;

– Albums: альбоми створені автоматично та містять зображення одного класу.

– Images: всі дані про зображення;

– SharedAlbum: інформація про альбоми, якими поділилися.

Між Users та Albums один до багатьох, бо один користувач може мати багато альбомів. Між Users та SharedAlbums один до багатьох, такий же зв’язок між Albums та SharedAlbums. Оскільки одному користувачу може бути доступно багато альбомів та один альбом може бути доступним багатьом користувачам. Одне зображення може належати тільки одному альбому (класу).

Оскільки для реалізації програмної системи було обрано нереляційну базу даних, то сутності міститимуть більше полів через денормалізацію та надмірність, яка допускається при роботі з нереляційними базами даних.

3.4 Приклади найцікавіших алгоритмів та методів

Одним із основних методів системи є обробка та аналіз зображень, тому опишемо його тут детальніше.

Все починається із отримання запиту у controller, який відповідає за обробку запитів пов’язаних із зображеннями. У випадку, коли запит валідний, тобто зображення необхідного формату та розміру були завантажені, запускається об’єкт класу service, що містить функції для роботи з даними зображення.

Далі відбувається обробка зображення у вигляді його стиснення при розмірі більше 2 Мб. Після чого зображення зберігається у сховищі – Azure Blob Storage.

Перед виконанням аналізу зображення слід звернути увагу на обмеження форматів для яких доступне розпізнавання тексту. Далі формується запит та відправляється до AI Vision.

Azure AI Vision надає доступ до передових алгоритмів, які обробляють зображення та повертають інформацію на основі візуальних характеристик, які цікавлять [14]. В нашому випадку це додавання тегів, класифікація та розпізнавання тексту.

Більш детально алгоритм цього методу показано на діаграмі діяльності на рис 3.6. Ці діаграми призначені для потоків управління та даних у програмі. Вони є особливо корисними при візуалізації складних процесів, які, наприклад містять паралельну обробку.

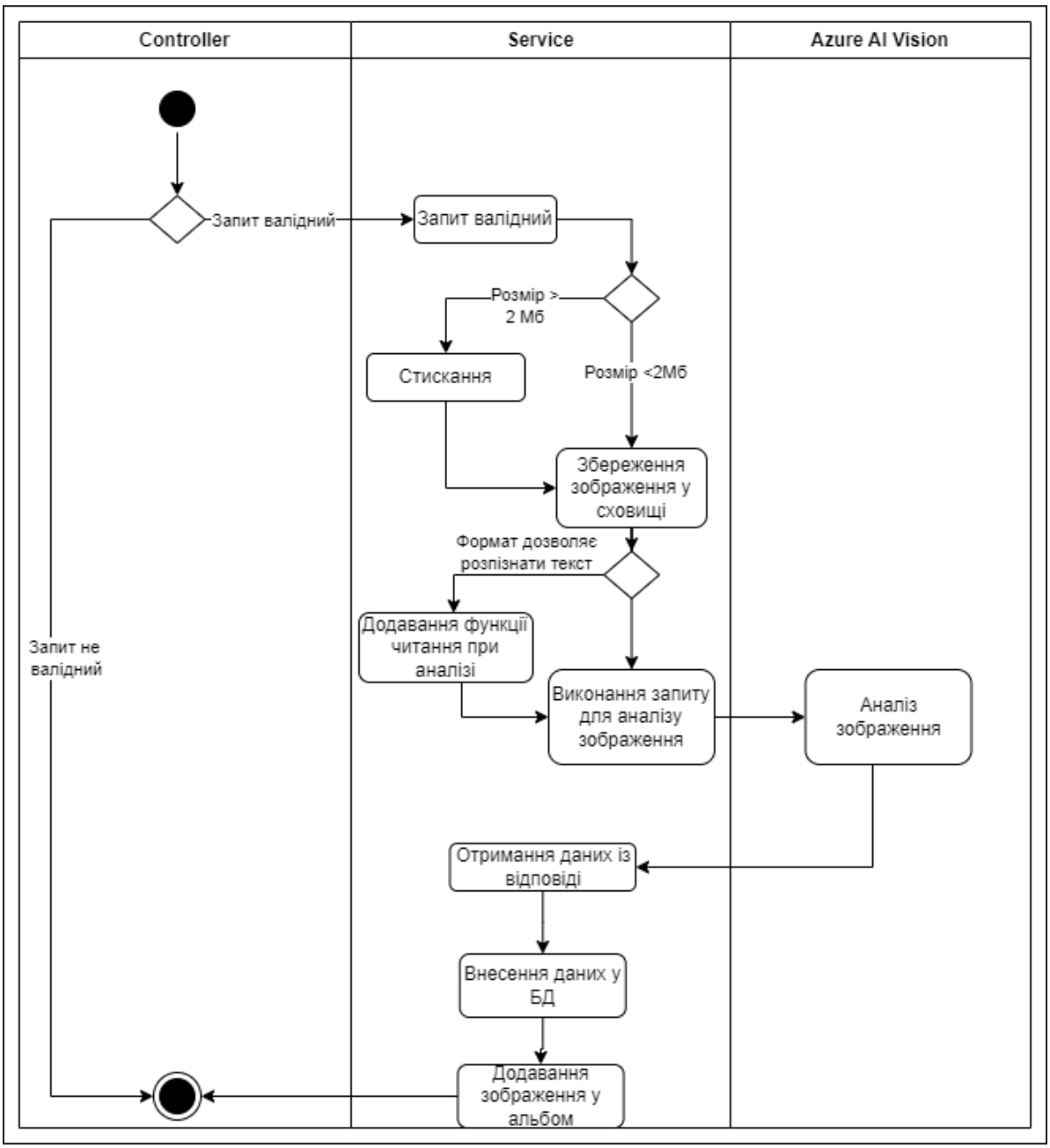


Рисунок 3.6 – Діаграма діяльності (виконано самостійно)

Як можна тепер побачити з діаграми після отримання відповіді про аналіз зображення від Azure AI Vision слід вийняти необхідні дані та зберегти їх у базі даних. Після чого зображення буде додано до нового або вже існуючого альбому залежно від визначеного класу, а користувачу повернеться повідомлення про успішне завантаження зображення.

Використання сервісів Azure, додаткових бібліотек Node.js та асинхронне виконання функцій значно підвищить ефективність виконання завантаження з обробкою та аналізом в умовах великих даних.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Створення моделей для бази даних

На основі створеної раніше ER-діаграми маємо, що в системі буде наявно три сутності: користувач, зображення, альбом. У Azure Cosmos DB дані зберігаються у вигляді контейнерів (колекцій) JSON документів. Створити їх можна при підключенні до бази. Код на створення такого контейнера наведено нижче:

```
const { container } = await
cosmosDatabase.containers.createIfNotExists({ id: 'Users' });
```

Для роботи з базою даних Azure Cosmos DB використано Azure Cosmos DB SDK для Node.js. Він надає ряд переваг:

– надає зручний API для полегшеного виконання основних CRUD (створення, читання, оновлення, видалення) операцій;

– підтримує складні запити, включаючи підтримку запитів SQL, запитів на основі JavaScript у збережених процедурах і тригерах;

– дозволяє ефективно масштабувати та обробляти великі обсяги даних із низькою затримкою.

Для початку для кожної сутності було розроблено моделі. Вони будуть корисними для роботи з базою даних, оскільки гарантуватимуть, що дані дотримуються певної структури.

Ці моделі реалізовані за допомогою класів. Переглянемо нижче код для моделі користувача:

```
class User {
    constructor(email, password, firstname, lastname, role='user') {
        this.email = email;
        this.hashPassword = password;
        this.firstname = firstname;
        this.lastname = lastname;
        this.role = role;
        this.locked = false
    }
```

В даному випадку він містить всі поля описані раніше крім ідентифікатора, який буде присвоєний базою автоматично.

Переглянемо тепер на код моделі зображення:

```
class Image {
    constructor(id, userId, originalName, blobName, imageUrl, 
className, tags, text, metadata, size, type='image') {
        this.id = id;
        this.type = type;
        this.userId = userId;
        this.originalName = originalName;
        this.blobName = blobName;
        this.imageUrl = imageUrl;
        this.className = className;
        this.tags = tags;
        this.text = text;
        this.metadata = metadata;
        this.date = new Date();
        this.size = size;
    }
```

В цьому випадку id зображеню буде присвоєно на бекенді за допомогою uuid v4 генератора. Як зв’язок із сутністю користувача використовується його ідентифікатор – userId. Cosmos DB автоматично індексує дані зображень, що дозволяє швидко шукати та отримувати інформацію про зображення на основі певного поля, наприклад, id або назви.

Код моделі альбомів:

```
class Album {
    constructor(className, images, userId, userEmail, sharedWith) {
        this.className = className;
        this.images = images;
        this.userId = userId;
        this.userEmail = userEmail;
        this.sharedWith = sharedWith;
        this.createdAt = new Date()
    }
```

Альбоми містять зображення, тому, скориставшись можливостями Azure Cosmos DB, промоделюємо цей зв’язок шляхом зберігання масиву ідентифікаторів зображень в альбомі (поле images). Аналогічно зроблено зі списком електронних пошт користувачів, яким доступний альбом (поле sharedWith). Слід зауважити, що

Azure Cosoms DB має обмеження розміру документа – 2 Мб. Проте цього достатньо вистачить, щоб зберігати ідентифікатори близько 40 000 зображень одного класу у одному альбомі одного користувача.

Також можна звернути увагу на те, що в альбомі, як і в зображенні, також зберігається id користувача, а також його пошта. Це приклад денормалізації, що є абсолютно допустимим, оскільки Azure Cosmos DB є нереляційною і, до прикладу, не підтримує звичний для реляційних баз JOIN.

Приклад формування запиту та його виконаня показано на рис 4.1.

```
1  async create(email, password, firstname, lastname) {
2      const hashedPassword = await bcrypt.hash(password, 5)
3      const user = new User(email, hashedPassword, firstname, lastname)
4      const querySpec = {
5          query: 'SELECT c.id FROM c WHERE c.email = @email',
6          parameters: [
7              {
8                  name: '@email',
9                  value: email
10             }
11         ]
12     };
13     const existUser = await user.find(querySpec)
14     if (existUser) {
15         throw new Error("User already exists");
16     }
17     await user.create()
18     const {hashPassword, ...userData} = user
19     return userData;
```

Рисунок 4.1 – Приклад виконання запиту (виконано самостійно)

На рисунку можна побачити створення (рядки 3-12), його виконання через об’єкті моделі (рядок 13) та повернення отриманих даних у відповідь на запит.

4.2 Створення функцій обробників для даних користувачів

При реалізації бізнес-логіки для роботи з даними користувачів зосередимось на описаних раніше функціях: авторизація, автентифікація, операції з даними (додавання, читання, зміна, видалення), перевірка ролі користувачів.

При реєстрації нового користувача серед даних, які вносяться, є пароль. Скористаємось поширеним методом – хешуванням паролю, тобто одностороннім

перетворенням без можливості інвертувати для захисту від загроз злому паролю. У Node.js для цього використаємо бібліотеку bcrypt, що надає таку можливість.

Автентифікація будується на використанні JSON Web Token (бібліотека jsonwebtoken), який є зручним способом для безпечної передачі даних між двома сторонами [15]. Токени створюються сервером, підписуються секретним ключем та передаються клієнту. В даних, що кодуються як JSON об'єкт, можна додати поле role, де вказується роль користувача (звичайний користувач, адміністратор). Це можна перевіряти при кожному запиті за допомогою middleware функції наведеній нижче:

```
return function (req, res, next) {
        if (req.method === "OPTIONS") {
            next()
        }
        try {
            const token = req.headers.authorization.split(' ')[1]
            if (!token) {
                return res.status(401).json({status: "error",
message: "Unauthorized!"})
            }
            const decoded = jwt.verify(token, process.env.SECRET_KEY)
            if (decoded.role !== role) {
                return res.status(403).json({status: "error",
message: "Access denied!"})
            }
            req.user = decoded;
            next()
        } catch (e) {
            res.status(401).json({status: "error", message:
"Unauthorized!"})
        }
    };
```

Тобто в даному випадку при відсутності токена, його недійсності або невідповідності ролі користувача доступ до запиту не відбудеться та повернеться помилка.

При внесенні даних про користувача в базу даних слід виконати валідацію переданих даних, а саме перевіряти коректний вигляд електронної пошти та довжина паролю. Скористаємось бібліотекою express-validator. Вона здатна перевіряти поля тіла запиту на відповідність певним вимогам.

Таким чином у разі невідповідності полів у відповіді на запит відправиться повідомлення про помилку.

Загалом всі відповіді на запити мають однаковий вигляд:

– поле статус: «error» або «success»;

– поле з даними або повідомлення про помилку.

На рис. 4.2 показано відповідь на запит з помилкою.

```
{
    "status": "error",
    "message": [
        {
            "type": "field",
            "value": "testgmail.com",
            "msg": "Invalid email format",
            "path": "email",
            "location": "body"
        }
    ]
}
```

Рисунок 4.2 – Відповідь на запит (виконано самостійно)

Такий підхід спростить обробку відповідей на запит на клієнтській частині, надаючи повідомлення при помилках.

4.3 Аналіз та обробка зображень

Основний функціонал програмної системи пов’язаний із зображеннями, тому для початку потрібно додати можливість завантажувати велику кількість зображень користувачем. Для цього скористаємось multer – це проміжне програмне забезпечення, яке дає можливість обробки завантажень файлів у програмах Node.js та Express.

Multer дозволяє зберігати файли на диску або в пам’яті, додати обмеження типу файлу, які дозволено завантажувати та максимальний розмір. Скористаємось збереженням зображення у пам’яті, врахуємо формати зображень та максимальний

розмір у 20 МБ дозволених для сервісу аналізу зображень. У результаті отримуємо такий код для перевірки завантажених файлів:

```
const upload = multer({
    storage: multer.memoryStorage(),
    limits: { fileSize: 20000000 }, // 20 MB
    fileFilter: function (req, file, cb) {
        checkFileType(file, cb);
    }}).array('images', 100);
function checkFileType(file, cb) {
    const filetypes = /jpeg|jpg|png|tiff|webp/;
    const extname =
filetypes.test(path.extname(file.originalname).toLowerCase());
    const mimetype = filetypes.test(file.mimetype);
    if (mimetype && extname) {
        return cb(null, true);
    } else {
        return cb(null, false);
    }
}
```

У разі вдалої перевірки та задовільнення всіх вимог, переходимо до обробки та аналізу.

Під обробкою розуміється стиснення зображення, тобто зменшення його розмірів у байтах. Це має на меті досягти більш ефективнішого зберігання зображень у сховищі, пришвидшити їх завантаження та покращити їх аналіз (в деяких випадках на стиснених зображення штучний інтелект краще розпізнає текст).

Для такої обробки використано бібліотеку sharp. Це швидка та ефективна бібліотека для роботи із зображеннями. Вона підтримує роботу з великою кількістю форматів зображень та надає різноманітний функціонал: зміна розміру, стиснення, трансформація, маніпуляції з кольором та отримання метаданих.

Слід врахувати, що використання стискання для файлів з початково малим розміром може мати негативний вплив. Оскільки при невеликому виграші у зекономленому розмірі (декілька кілобайт) збільшується час обробки. Також необхідно зауважити, що використання алгоритмів стискання для зображень малих розмірів в деяких випадках може викликати збільшення його розміру. Тому було обрано порогове значення 2 МБ нижче якого стискання не проводитиметься.

Для стиснення треба обрати значення quality від 1-100, яке за замовчуванням дорівнює 80. Можна самостійно підібрати ці значення для кожного формату для виконання стискання без помітних втрат. Нижче наведемо частину коду, де для різних типів вказано це значення:

```
const formatOptions = {
        'image/jpeg': sharp(file.buffer).jpeg({ quality: 85 }),
        'image/png': sharp(file.buffer).png({ quality: 85 }),
        'image/webp': sharp(file.buffer).webp({ quality: 90 }),
        'image/tiff': sharp(file.buffer).tiff({ quality: 90 })
    };
```

Для завантаження зображень у сховище Azure Blob використовується Azure Storage Blob SDK. Завантажуємо файл, надавши йому унікальну назву, після чого отримуємо посилання на нього.

Для аналізу зображення виконується запит до сервісу Azure AI Vision за допомогою SDK. Перед цим слід врахувати обмеження на формат файлів для виконання розпізнавання тексту. В результаті аналізу буде отримано тіло відповіді, з якого слід вийняти список тегів (перший тег буде присвоєно як клас зображення) та текст, якщо він присутній на зображенні. Нижче наведений код, який виконує все вище описане:

```
const result = await cvClient.path('/imageanalysis:analyze').post({
        body: {
            url: imageUrl
        },
        queryParameters: {
            features: features
        },
        contentType: 'application/json'
    });
    let tags = []
    const classResult = result.body.tagsResult.values[0]?.name ||
'unknown';
    result.body.tagsResult.values.forEach(value => {
        tags.push(value.name)
    })
    let resultText = []
    if (result.body.readResult) {
        result.body.readResult.blocks.forEach(block => {
            block.lines.forEach(line => {
                resultText.push(line.text)
            })
```

```
        })
    }
```

Дані про аналіз зображень вносяться в базу даних. Для покращення продуктивності запис у базу відбувається пакетами замість завантаження кожного результату окремо. Весь код для обробки зображень наведено у Додатку В.

В якості відповіді на запит повертається список назв зображень та присвоєні їм класи, як це показано на рис. 4.3.


```
"status": "success",
"data": [
    {
        "originalName": "DSC00141.JPG",
        "className": "building"
    },
    {
        "originalName": "test_read.PNG",
        "className": "text"
    },
    {
        "originalName": "thumbn01518878_ostrich.JPEG",
        "className": "bird"
    }
]
```


Рисунок 4.3 – Тіло відповіді на запит (виконано самостійно)

Ці зображення додаються в існуючі альбоми, або створюють нові. Один альбом містить файли одного класу. Для прикладу завантажимо зображення, яке продемонстроване на рис. 4.4.

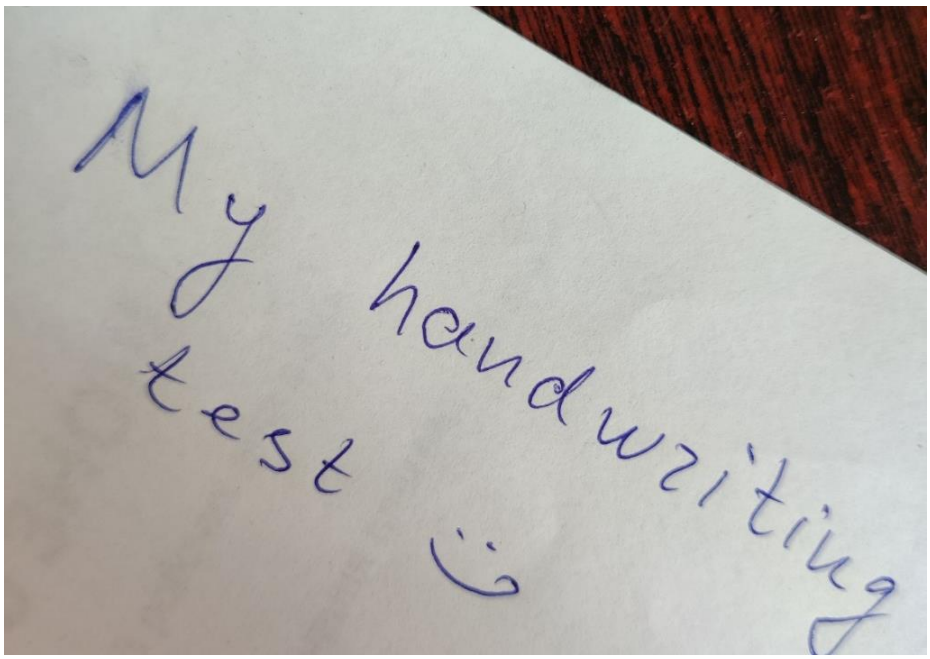


Рисунок 4.4 – Приклад зображення (виконано самостійно)

Результати аналізу можна отримати при запиті на перегляді конкретного зображення. На рис. 4.5 можна побачити частину JSON документа, що містить дані про результат аналізу завантаженого зображення (див. рис. 4.4).


```
"className": "text",
"tags": [
    "text",
    "handwriting",
    "letter",
    "paper",
    "paper product",
    "stationery",
    "calligraphy",
    "ink",
    "stationary",
    "envelope",
    "document"
],
"text": [
    "My",
    "handwriting",
    "test"
],
```


Рисунок 4.5 – Дані про зображення (виконано самостійно)

В даному прикладі можна побачити, що для зображення було додано 11 тегів, перший з яких обрано як назву класу для зображення. Крім цього, оскільки на зображені був присутній текст, його розпізнано та внесено у відповідне поле «text». Оскільки розмір зображення не перевищує 2 Мб, то стискання не відбувалося та розмір не змінювався.

Отож, використовуючи описану вище реалізацію, вдалося досягти економії пам’яті на 10-40% залежно від початкового розміру та формату зображення. При цьому час завантаження декількох десятків зображень незначно поступається таким сервісам як Google Photo. Така реалізація є доволі ефективною для роботи з великою кількістю зображень.

4.4 Створення функцій обробників для даних альбомів

Крім базових операцій перегляду, зміни, видалення, альбоми можуть завантажуватися як архів всіх зображень, що в них містяться. Розглянемо цю функцію детальніше.

Для початку потрібно отримати зображення із сховища. Це виконують дві функції downloadBlob та streamToBuffer, код яких зображено на рис. 4.6.

```
async function downloadBlob(blobName) {
    const blobClient = containerClient.getBlobClient(blobName);
    const downloadBlockBlobResponse = await blobClient.download(0);
    return await streamToBuffer(downloadBlockBlobResponse.readableStreamBody);
}
async function streamToBuffer(readableStream) {
    return new Promise((resolve, reject) => {
        const chunks = [];
        readableStream.on("data", (data) => {
            chunks.push(data);
        });
        readableStream.on("end", () => {
            resolve(Buffer.concat(chunks));
        });
        readableStream.on("error", reject);
    });
}
```

Рисунок 4.6 – Функції для завантаження зображень із сховища (виконано самостійно)

DownloadBlob завантажує blob як Radable Stream і перетворює його на Buffer за допомогою streamToBuffer. В даному випадку Buffer це об’єкт Node.js, який забезпечує спосіб безпосередньої обробки неструктурованих двійкових даних.

StreamToBuffer зчитує дані потоку, збирає їх у масив та об’єднує, використовуючи Promise (об’єкт, що представляє можливе завершення або помилку асинхронної операції). Тут використання await і async забезпечує асинхронне, неблокуюче виконання, що робить код ефективним і здатним обробляти потенційно великі blob-файли без блокування основного потоку.

Для формування архіву використовується бібліотека archiver. Нижче наведено код для створення та відправки архіву:

```
const archive = archiver('zip');
res.attachment('images.zip');
archive.pipe(res);
images.forEach((image, index) => {
archive.append(image, { name: imagesBlobs[index] });
});
await archive.finalize();
```

Archive.pipe(res) передає дані з архіву безпосередньо у відповіді, дозволяючи клієнту завантажувати архів під час його створення. В циклі зображення додаються у архів, а метод finalize сповіщає про завершення додавання файлів.

Такий підхід дозволяє легко та ефективно завантажувати велику кількість зображень різного розміру із сховища разом.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування серверної частини додатку є важливим аспектом перевірки роботи компонентів програмної системи: робота бази даних, сервісів та бізнес логіки. Існує багато підходів до тестування: ручне, автоматизоване, тестування білого або чорного ящика.

В роботі було використано модульне тестування. Його можна розглядати як процес забезпечення якості коду, яке спрямована на перевірку правильності роботи окремих одиниць коду, таких як функції, методи, класи [16]. Серед переваг такого типу тестування є швидкість знаходження помилок та виправлення їх на ранніх стадіях розробки.

Для тестування API на Node.js та Express доступна велика кількість фреймворків. Зупинимось на Jest, що є однією з найпопулярніших бібліотек для тестування коду на JavaScript, особливо у випадку модульного тестування. Додатково використаємо SuperTest для тестування ендпоінтів та HTTP маршрутів.

Структура тестів написаних за допомогою Jest складається з декількох компонентів:

– «describe» для опису модульного тесту;
– «test» описуємо, що має робити тест та вказуємо функцію зворотного виклику;
– функція зворотного виклику спочатку надсилає запит до ендпоінта, а потім порівнюються очікувана та отримана відповіді. Тест пройдено, якщо обидві відповіді збігаються.

Нижче наведемо приклад коду для тестування реєстрації користувача:

```
describe("POST /api/user/registration", () => {
    test("should register new user", async () => {
        const data = { firstname: "John", lastname: "Doe", email:
"john@doe.com", password: "123456" };
        const response = await
request(app).post("/api/user/registration").send(data);
        id = response.body.user.id;
        expect(response.statusCode).toBe(200);
    });});
```

Тестування має покрити головний функціонал системи пов’язаний з користувачами, зображеннями та альбомами. Основні тестові випадки та результати тестування описані у таблиці 5.1.

Таблиця 5.1 – Таблиця тестових випадків та результатів

№	Опис	Передані дані	Очікуваний результат	Висновок
1	Реєстрація користувача	Електронна, пошта, пароль, ім’я прізвище	Додано нового користувача (статус 200)	Пройдено
2	Перевірка електронної пошти при реєстрації	Не коректна електронна, пошта, пароль, ім’я прізвище	Повідомлення про помилку (статус 400)	Пройдено
3	Пустий пароль при реєстрації	Електронна, пошта, ім’я прізвище	Повідомлення про помилку (статус 400)	Пройдено
4	Авторизація користувача	Електронна, пошта, пароль	Отримано токен авторизації (статус 200)	Пройдено
5	Отримання даних про користувача	Токен авторизації	Отримано дані користувача (статус 200)	Пройдено
6	Отримання даних про користувача без авторизації	Токен авторизації	Повідомлення про помилку (статус 401)	Пройдено
7	Отримання даних про систему адміном	Токен авторизації	Отримано дані про систему (статус 200)	Пройдено

Кінець таблиці 5.1.

№	Опис	Передані дані	Очікуваний результат	Висновок
8	Видалення користувача та всіх його даних	Токен авторизації, id користувача	Отримано дані про систему (статус 200)	Пройдено
9	Отримання даних про систему не адміном	Токен авторизації	Повідомлення про помилку (статус 403)	Пройдено
10	Завантаження нового зображення	Токен авторизації, зображення	Додано нове зображення (статус 200)	Пройдено
11	Перегляд даних про зображення	Токен авторизації, ID зображення	Отримано всі дані про зображення (статус 200)	Пройдено
12	Перегляд альбому	Токен авторизації, ID альбому	Отримано всі зображення альбому (статус 200)	Пройдено
13	Видалення альбому	Токен авторизації, ID альбому	Видалено альбом та всі його зображення (статус 200)	Пройдено
14	Відкрити доступ для користувача	Токен авторизації, email	Додано нового користувача у список, з ким поділилися (статус 200)	Пройдено
15	Закрити доступ для користувача	Токен авторизації, email	Користувача із списку видалено (статус 200)	Пройдено

Отже, було сформовано та написано 15 модульних тестів. Повідомлення про успішне виконання всіх тестів можна побачити на рис. 5.1.

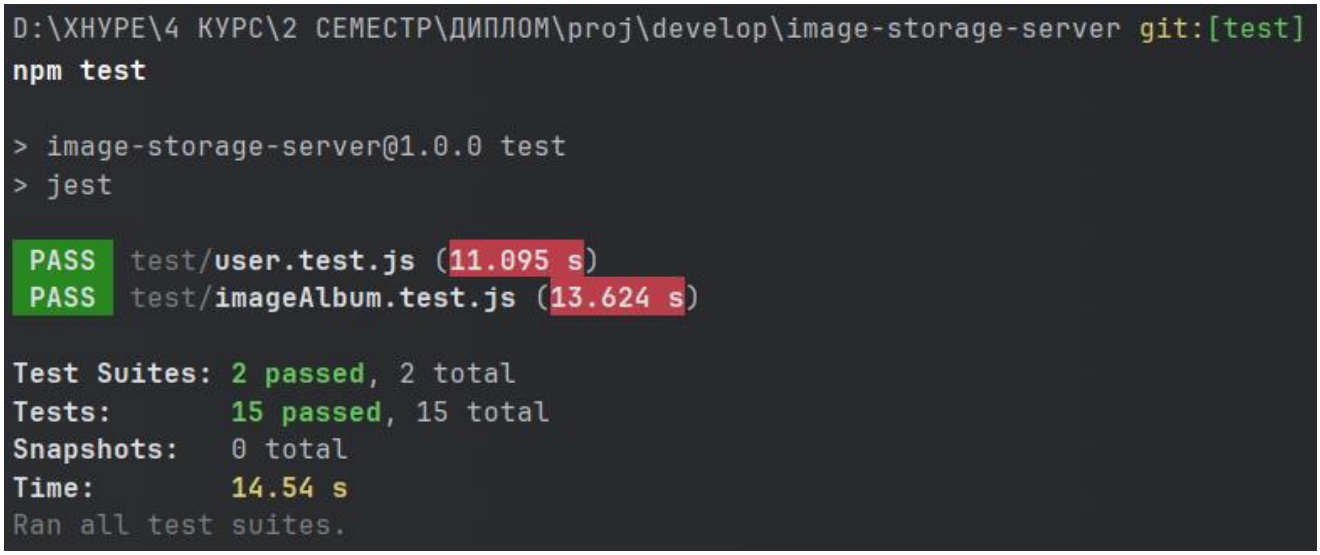


Рисунок 5.1 – Результат запуску тестів (виконано самостійно)

Як можна побачити з рисунку, тестування складалося з 2 наборів тестів (для користувача та альбомів з зображеннями), які всього містили 15 тестів. Всі вони були успішно пройдені приблизно за 15 секунд.

Таким чином за допомогою модульного тестування було успішно перевірено критично важливі компоненти для забезпечення надійності, продуктивності та безпеки серверної частини програмної системи.

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Наступним важливим етапом після розробки та тестування програмної системи є впровадження програмного забезпечення. На попередніх етапах було виконано аналіз предметної області, що дозволило оцінити актуальність та потребу в програмному забезпеченні. Після чого було сформовано перелік вимог до системи та вибір технологій під час проєктування. Програма написана та протестована під час розробки.

На основі зробленого можна перейти до визначення етапів та плану впровадження програмного забезпечення. Першим з них можна виділити етап розгортання програмного забезпечення, тобто надання доступу до використання. На цьому кроці потрібно звертати увагу на доступність, продуктивність та використання ресурсів, які задіяні у розгортанні. Також важливо виявити та повідомити про потенційні помилки.

Після цього йде етап більш комплексного та детального тестування, фактично його можна назвати альфа-тестуванням. Учасниками його є команда розробників. Таке тестування спрямоване на виявлення та виправлення помилок до того, як програмне забезпечення буде випущено для ширшої аудиторії, включає перевірку функціональності, зручності використання та безпеки [17].

У разі виявлення помилок відводиться етап для їх виправлення. Тільки після цього відбувається випуск програмного забезпечення для кінцевих користувачів. Цей етап є важливим і вимагає ретельного планування та координації, щоб гарантувати, що програмне забезпечення постачатиметься безперебійно та функціонуватиме правильно. Слід відстежувати всі можливі помилки та швидко реагувати на них.

Описаний підхід може забезпечити плавний випуск, вирішення виявлених проблем та надання доступу користувачам до якісного програмного забезпечення.

ВИСНОВКИ

Результатом кваліфікаційної роботи є реалізована серверна частина веб-орієнтованої системи гібридного механізму управління зображеннями для їх ефективної обробки та аналізу в умовах великих даних.

У ході даної роботи було проаналізовано предметну область, пов’язану зберіганням великої кількості зображень. Область налічує достатню кількість вже існуючих сервісів та застосунків, попри це було виявлено ряд їх недоліків, пов’язаних із конфіденційністю, обмеженим функціоналом, швидкістю завантаження та ефективністю зберігання великої кількості зображень.

В результаті виконання кваліфікаційної роботи, спроектовано архітектуру серверної частини застосунку та обрано технології, які дозволили повністю реалізувати запланований підхід та відповідати всім поставленим вимогам.

Веб-застосунок було написано з використанням мови програмування JavaScript та фреймворків Node.js і Express. За допомогою Node.js вдалося розробити продуктивний додаток, а також реалізувати різноманітний функціонал завдяки великій кількості наявних бібліотек. Для зберігання даних було обрано гібридний механізм.

Для зберігання зображень використано Azure Blob Storage, а дані про зображення, такі як назва, розмір, результати аналізу містяться у базі даних Azure Cosmos DB. Такий вибір технологій дозволив забезпечити високу доступність, безпеку та ефективність в умовах великих даних. Аналіз зображення реалізованою з використанням сервісу Azure AI Vision. Він дозволив легко та ефективно отримувати клас, теги та розпізнаний текст на основі зображення.

Програмна система була вдало протестована, вона є ефективним, безпечним та потужним інструментом готовим до використання в якості сховища зображень в умовах великих даних.

Матеріали кваліфікаційної роботи пройшли апробацію на VI Всеукраїнській студентській конференції «Експериментальні та теоретичні дослідження в контексті сучасної науки» (див. дод. Д). [18]

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Flickr [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Flickr (дата звернення: 05.05.2024).

2. Photobucket [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Photobucket (дата звернення 05.05.2024).

3. Picasa [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Picasa (дата звернення 05.05.2024).

4. How many pictures are there (2024): [Електронний ресурс] – Режим доступу до ресурсу: https://photutorial.com/photos-statistics (дата звернення 25.04.2024).

5. Терещенко Г.Ю., Стась Б.Л. Архітектура Сховища Зображень в Епоху Великих Даних. // XIII International Scientific and Practical Conference «Eurasian scientific discussions», (January 22-24, 2023), Barcelona, Spain, 2023. –Barca Academy Publishing,478 p. – PP. 191 - 195.

6. Google Photos [Електронний ресурс] – Режим доступу до ресурсу: https://www.google.com/photos/about (дата звернення 13.05.2024).

7. Документація PhotoPrism [Електронний ресурс] – Режим доступу до ресурсу: https://docs.photoprism.app (дата звернення 13.05.2024).

8. Документація Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/api/all.html (дата звернення 05.05.2024).

9. Документація Azure Cosmos DB [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/ru-ru/azure/cosmos-db (дата звернення 05.05.2024).

10. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3-тє вид. Print2print, 2016. 208 с.

11. Терещенко Г. Ю., Боцюра І. С. Image warehouse architecture in the era of big data // 25-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті», Збірник Матеріалів форуму., т. 6, - Харків (ХНУРЕ), 20-22 квітня 2021 р., с. 163 - 164.

12. Tereshchenko G., Kyrychenko I., Chetverykov G. Overview of image storage models in Big Data conditions // Комп’ютерна математика в науці, інженерії та освіті

(CMSEE-2020). – Матеріали V Всеукраїнської науково-технічної конференції – Полтава, 27 листопада 2020 р. – С. 18–21. УДК 00.89:004.043.

13. Документація Azure Blob Storage [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction (дата звернення 13.05.2024).

14. Документація Azure AI Vision [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/en-us/products/ai-services/ai-vision (дата звернення 13.05.2024).

15. Документація JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/jsonwebtoken (дата звернення 20.05.2024).

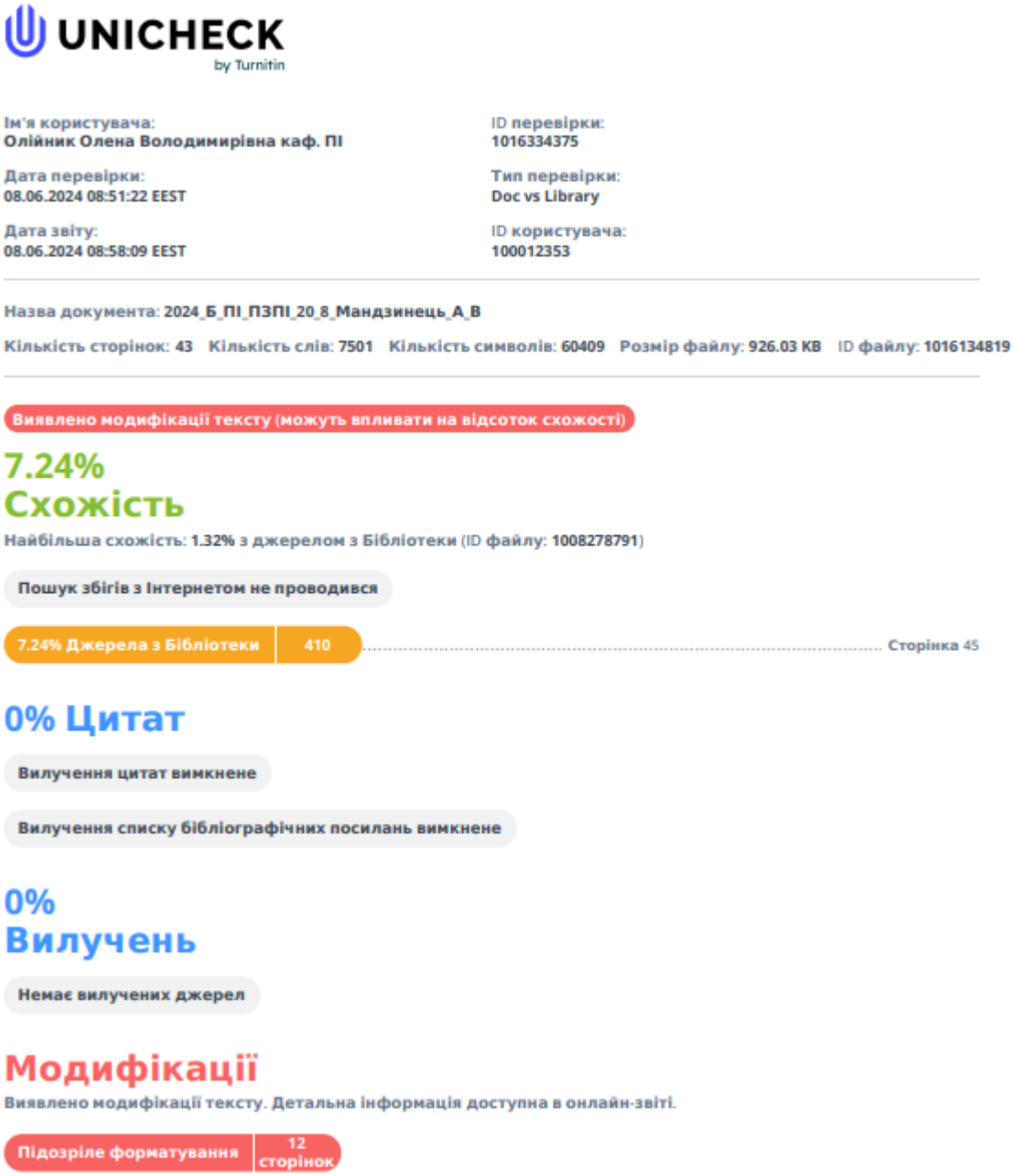
16. What is unit testing? [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/unit-testing (дата звернення: 20.05.2024).

17. Software release life cycle [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Software_release_life_cycle (дата звернення: 20.05.2024).

18. Терещенко Г. Ю., Мандзинець А. В., Долготер М. С. Проектування веб-орієнтованої системи гібридного механізму управління зображеннями для їх ефективної обробки та аналізу в умовах великих даних // VI Всеукраїнська студентська конференція «Експериментальні та теоретичні дослідження в контексті сучасної науки», Збірник Матеріалів конференції, - Рівне, 21 червня 2024 р., с. 18 - 22.

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Олійник Олена Володимирівна каф. ПІ

ID перевірки:
1016334375

Дата перевірки:
08.06.2024 08:51:22 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
08.06.2024 08:58:09 EEST

ID користувача:
100012353

Назва документа: 2024_Б_ПІ_ПЗПІ_20_8_Мандзинець_А_В

Кількість сторінок: 43 Кількість слів: 7501 Кількість символів: 60409 Розмір файлу: 926.03 KB ID файлу: 1016134819

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

7.24%
Схожість

Найбільша схожість: 1.32% з джерелом з Бібліотеки (ID файлу: 1008278791)

Пошук збігів з Інтернетом не проводився

7.24% Джерела з Бібліотеки 410 Сторінка 45

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування 12 сторінок

ДОДАТОК Б

Перелік джерел посилання за науковими напрямками керівника та науковців кафедри програмної інженерії

5. Терещенко Г.Ю., Стась Б.Л. Архітектура Сховища Зображень в Епоху Великих Даних. // XIII International Scientific and Practical Conference «Eurasian scientific discussions», (January 22-24, 2023), Barcelona, Spain, 2023. –Barca Academy Publishing,478 p. – PP. 191 - 195.

11. Терещенко Г. Ю., Боцюра І. С. Image warehouse architecture in the era of big data // 25-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті», Збірник Матеріалів форуму., т. 6, - Харків (ХНУРЕ), 20-22 квітня 2021 р., с. 163 - 164.

12. Tereshchenko G., Kyrychenko I., Chetverykov G. Overview of image storage models in Big Data conditions // Комп’ютерна математика в науці, інженерії та освіті (CMSEE-2020). – Матеріали V Всеукраїнської науково-технічної конференції – Полтава, 27 листопада 2020 р. – С. 18–21. УДК 00.89:004.043.

18. Терещенко Г. Ю., Мандзинець А. В., Долготер М. С. Проектування веб-орієнтованої системи гібридного механізму управління зображеннями для їх ефективної обробки та аналізу в умовах великих даних // VI Всеукраїнська студентська конференція «Експериментальні та теоретичні дослідження в контексті сучасної науки», Збірник Матеріалів конференції, - Рівне, 21 червня 2024 р., с. 18 - 22.

ДОДАТОК В

Приклад кодів програми

В.1 Функції для виконання обробки та аналізу зображень при завантажені

```
    async function uploadImages(req, res) {
        try {
            if (req.files === undefined || req.files.length === 0) {
            return  res.status(400).json({status:  'error',  message:  'No
files selected or unsupported format!'});
        }
        const id = req.user.id;
        const email = req.user.email
        const {dbResult, resultData} = await ImageService.upload(id, email,
req.files)
        res.status(200).json({status:'success', data: resultData});
        for (const item of dbResult) {
            await    AlbumService.addOrUpdate(item.resourceBody.className,
item.resourceBody.blobName, id, email);
        }
        } catch (e) {
            console.error(e);
            res.status(500).json({status: 'error', message: e.message})
        }
    }
    async function imageCompression(file) {
        const format = file.mimetype
        const formatOptions = {
            'image/jpeg': sharp(file.buffer).jpeg({ quality: 85 }),
            'image/png': sharp(file.buffer).png({ quality: 85 }),
            'image/webp': sharp(file.buffer).webp({ quality: 90 }),
            'image/tiff': sharp(file.buffer).tiff({ quality: 90 })
        };
        if (!formatOptions[format]) {
            throw new Error(`Unsupported image format: ${format}`);
        }
        return formatOptions[format].toBuffer();
    }
    async function imageProcessing(file) {
        let imageBuffer = file.buffer;
        let features;
        if (file.size > THRESHOLD_SIZE) {
            imageBuffer = await imageCompression(file)
        }
        if (READABLE_FORMATS.includes(file.mimetype)) {
            features = ['Tags', 'Read']
        } else {
            features = ['Tags']
        }
        return {imageBuffer,features}
    }
    async function upload(id, email, files) {
        const imageUploadPromises = files.map(async (file) => {
```

```
            const {imageBuffer,features} = await
this.imageProcessing(file)
            const size = imageBuffer.length
            const imageId = uuidv4().toString()
            const blobName = imageId + path.extname(file.originalname);
            const imageUrl = await uploadBlob(blobName, imageBuffer)
            const {tags, classResult, resultText, metadata} = await
imageAnalysis(imageUrl,features)
            return new Image(imageId, id, file.originalname, blobName,
imageUrl, classResult, tags, resultText, metadata, size)
        });
        const uploadAndCVResults = await Promise.all(imageUploadPromises);
        const imageModel = new Image()
        const dbResult = await imageModel.create(uploadAndCVResults)
        const resultData = uploadAndCVResults.map(({ id, originalName,
className }) => ({ id, originalName, className }));
        return {dbResult: dbResult, resultData: resultData}
    }
    async function create(data) {
        const container = await getImagesContainer()
        return await container.items.bulk(data.map(item =>
({operationType: 'Create', resourceBody: item})))
    }
```

ДОДАТОК Г

Специфікація програмного продукту

1 ВСТУП

1.1 Огляд проєкту

Фото та зображення давно стали невід’ємною частиною нашого життя. Розвиток камер на телефонах, покращення якості передачі даних в мережі Інтернет, поява соціальних мереж вплинули на те, що нині тільки в Інтернеті налічується близько 136 мільярдів зображень.

На наших власних пристроях зберігається значна кількість зображень. Через це дуже часто нам стає важко знаходити потрібні фото або місце, яке вони займають, стає непомірним.

Виходом із такої ситуації стали сховища для зображень. Деякі з них працюють як хмарне сховище, які синхронізують дані на пристрої та в хмарі або поставляються з пристроями для локального зберігання зображення. Також для цього часто використовуються соціальні мережі або месенджери.

1.2 Мета

Метою роботи є розробка серверної частини програмної системи для гібридного механізму управління зображеннями для їх ефективної обробки та аналізу в умовах великих даних.

1.3 Область застосування

Застосування програмної системи доступне на будь-якому пристрої, що дає можливсіть працювати з веб-ресурсами. Використання застосунку може бути у вигляді особистого сховища зображень, комерційне використання, освіта, медицина, креативна індустрія.

1.4 Короткий зміст

Система зосереджена на можливості автоматичної обробки зображень, наприклад, стиснення без помітної втрати якості для:

– швидшого завантаження багатьох зображень;

– ефективного зберігання великої кількості даних.

При цьому при завантаженні наявні функції аналізу зображень:

– присвоєння зображенню певний клас (класифікація);

– додавання тегів до зображення на основі вмісту;

– розпізнавання тексту на зображенні.

Крім цього застосунок охоплює додатково наступні функції:

– створення облікового запису користувача;

– автоматичну класифікацію зображень при завантаженні;

– пошук за назвою та тегами зображення;

– завантаження багатьох зображень;

– дані про зайнятий простір;

– можливість поділитися альбомом з іншими користувачами;

Серед нефункціональних вимог можна виділити:

– дотримання безпеки даних;

– низька затримка відповіді системи;

– зручний та інтуїтивний інтерфейс.

2 ЗАГАЛЬНИЙ ОПИС

2.1 Перспективи продукту

Розроблювана програмна система надає можливість користувачам створити обліковий запис та завантажувати у застосунок велику кількість власних зображень. При цьому виконається обробка та аналіз для стиснення, класифікації, присвоєння тегів та розпізнавання тексту. Це може задовільнити велику кількість користувачів.

2.2 Функції продукту

Функціями веб-застосунку є реєстрація, авторизація, завантаження фото, класифікація, присвоєння тегів та розпізнавання тексту, перегляд зображень у альбомах, зміна власних даних та даних про зображення, завантаження всіх зображень із альбому.

2.3 Характеристики користувачів

Основною цільовою аудиторією будуть:

– особи, які хочуть безпечно зберігати та впорядковувати свої особисті зображення;

–аматори і професійні фотографи, якім потрібна платформа для зберігання своїх зображень;

– компанії, яким потрібен репозиторій для зберігання та керування зображеннями документації або реклами.

2.4 Загальні обмеження

Обмеження системи пов’язані перш за все доступом до використання технологій та інструментів, які використовуються програмною системою.

Для роботи веб-сервера використовується мова програмування JavaScript з фреймворками Node.js та Express.

Для зберігання даних та аналізу зображень використовуються хмарні сервіси Azure. Максимальний розмір документа в базі даних Cosmos DB – 2 Мб. Допустимі формати зображень для AI Vision – jpeg, jpg, png, webp, tiff. Максимальний розмір зображення – 20 Мб.

2.5 Припущення й залежності

1. Припущення, що користувачі мають доступ до Інтернету.

2. Залежність від постачальників хмарних послуг Azure для зберігання даних та аналізу зображень.

3. Залежність від сторонніх бібліотек і інструментів Node.js для обробки зображень і безпеки.

3 КОНКРЕТНІ ВИМОГИ

3.1 Вимоги до інтерфейсів

3.1.1. Інтерфейс користувача

Для роботи із застосунком клієнта доступі публічні API. Вони виконують функції аутентифікації, роботи із зображеннями та іншими даними. Також наявні внутрішні API між сервісами веб-серверу, зберігання даних та аналізу зображень.

3.1.3. Комунікаційний протокол

В якості комунікаційних протоколів використано HTTP/HTTPS.

3.1.4. Вимоги до пам’яті

Вимоги до пам’яті базуються на основі вимог середовища Node.js, в якому працює застосунок. Далі наведено мінімальні вимоги:

– розрядність системи: 32 або 64 біт;

– оперативна пам’ять: 512 Мб;

– дискова пам’ять: 300 Мб.

3.2 Атрибути програмного продукту

3.2.1 Безпека

Атрибутами безпеки програмного продукту пов’язані перш за все з авторизацією та аутентифікацією користувача на основі JSON токена. На основі нього ж реалізований інтерфейси для рольового контролю доступу та керування дозволами. Під час зберігання та передачі даних відбувається шифрування даних.

3.2.2 Супроводжуваність

Супроводжуваність проєкту забезпечується з допомогою системи контролю версій Git та відкритого репозиторію на Github для співпраці з іншими розробниками. Робити регулярний рефакторинг коду, щоб покращити його

структуру та читабельність. Пошук проблем та тестування при додавання нових компонентів. Постійнк відстеження та оновлення бібліотек від яких залежить проєкт. Додано документацію опису проєкту, специфікації, коментарі в коді.

3.2.3 Переносимість

Для переносимості програмного продукту в різних середовищах дотримано стандарти методів кодування для реалізації REST API. Дотримано використання стандартних форматів даних та протоколів.

3.2.4 Продуктивність

Час відповіді на більшість запитів становить 1-2 с. Здатність справлятися зі збільшеним навантаженням шляхом горизонтального або вертикально масштабування.

3.2.5 Вимоги до бази даних

Для розгортання база даних Cosmos DB на хмарному провайдері Azure необхідна підписка. Пропускна здатність бази – 1000 запитів в секунду.Максимальний розмір – 25 ГБ. Час відповіді 10 мс. Здатність справлятися зі збільшеним навантаженням шляхом горизонтального або вертикально масштабування.

3.2.6 Інші вимоги

Для сховища зображень розгортається сервіс Azure Blob Storage, для аналізу Azure AI Vision. Для цього необхідна підписка на Azure. Створення облікового запису Azure Storage та Cognitive Services, щоб використовувати можливості Azure AI Vision.

ДОДАТОК Д

Тези VI Всеукраїнської студентської конференції «Експериментальні та теоретичні дослідження в контексті сучасної науки»

ПРОЕКТУВАННЯ ВЕБ-ОРІЄНТОВАНОЇ СИСТЕМИ ГІБРИДНОГО МЕХАНІЗМУ УПРАВЛІННЯ ЗОБРАЖЕННЯМИ ДЛЯ ЇХ ЕФЕКТИВНОЇ ОБРОБКИ ТА АНАЛІЗУ В УМОВАХ ВЕЛИКИХ ДАНИХ

Мандзинець Анатолій Володимирович
здобувач вищої освіти факультету комп'ютерних наук
Харківський національний університет радіоелектроніки, Україна
Долготер Михайло Сергійович
здобувач вищої освіти факультету комп'ютерних наук
Харківський національний університет радіоелектроніки, Україна
Науковий керівник: Терещенко Гліб Юрійович
ст. викладач кафедри програмної інженерії
Харківський національний університет радіоелектроніки, Україна

Кількість зображень, що створюють та розповсюджують люди постійно росте. Наприклад, Instagram щоденно збільшується на 95 мільйонів зображень. У 2022 році в Інтернеті було близько 136 мільярдів зображень. Але ж крім соціальних мереж зображення широко використовуються різних сферах від освіти та медицини, до океанографії та астрономії.

Розвиток хмарних технологій, таких як Google Photo, Apple iCloud, Amazon Photo, дозволив зручно працювати з великою кількістю зображень, забезпечуючи додаткові можливості, не турбуючись про обмеження пам'яті пристрою. Однак, користувачі повинні покладатися на провайдерів цих послуг щодо безпеки та конфіденційності.

Локальне сховище або власний хостинг, наприклад, Synology, Immich, PhotoPrism, Lychee, є альтернативою. Проте вони мають свої недоліки: погана робота розпізнавання об'єктів, повільне завантаження та пошук, складне налаштування.

Окрім цього, передача великої кількості даних при початковому завантаженні зображень впливає на час завантаження та ефективність

використання пам'яті, що позначається на швидкості пошуку, відображення та завантаження зображень на пристрій.

Тому створення системи гібридного механізму управління зображеннями та реалізація їх ефективної обробки та аналізу в умовах великих даних є актуальною задачею. Основною метою є вибір архітектури та стеку технологій для програмної системи, яка повинна виконувати ряд функцій: реєстрація авторизація користувача, завантаження великої кількості зображень, обробки та аналізу зображень, можливість поділитися зображеннями з іншими. Для пришвидшення завантаження та ефективнішого використання пам'яті виконаємо обробку зображення у вигляді стиснення без помітних втрат якості. В якості аналізу можна додати корисні для користувача функції класифікації зображень, виявлення об'єктів, розпізнавання тексту, автоматичне додавання тегів.

Проектована програмна система буде наслідувати трирівневу архітектуру. Клієнтом буде являтися Web-застосунок. Серверна частина, яка буде представлена у вигляді API, буде реалізована з використанням фреймворку Express, що побудований над Node.js [1]. Цей стек дозволяє доволі легко створювати масштабовані, високопродуктивні веб-додатки. При цьому Node.js надає доступ великої кількості бібліотек, що дозволяють додати готову функціональність у проєкт, наприклад, для стиснення зображень.

Оскільки робота відбувається в умовах великих даних, слід подбати про правильний підхід до управління великою кількістю зображень та даних про них. Загалом виділяють багато підходів для зберігання зображень [2].

Перший – це зберігання зображень у базі даних у вигляді двійкових файлів. Цей метод є простим і ефективним, але він може бути невигідним, коли ми говоримо про зображення в умовах великих даних.

Інший підхід – це зберігати зображення на пристрої, а шлях до файлу зберігати у базі даних. Цей метод забезпечує легкий доступ до зображень, але він дуже залежить від доступних ресурсів пристрою, через що може стати не ефективним.

Третій варіант полягає у використанні служби хмарного зберігання. Це передбачає збереження зображень у хмарному сервісі, а потім збереження URL-адрес у базі даних. Такий метод має високу масштабованість і забезпечує легкий доступ до зображень з будь-якого місця. Хоча мінусом такого рішення може стати його вартість.

У програмній системі, яка розроблюється, обрано гібридний підхід до зберігання таких даних. Для зберігання даних про користувачів та зображення використано нереляційну базу даних. NoSQL бази даних добре підходять для роботи в умовах Big Data [3]. Самі ж файли зберігаються у хмарному сховищі.

В якості бази даних обрано Azure Cosmos DB. Це глобально розподілена, багатомодельна база даних від Azure. Вона забезпечує високу доступність, є горизонтально масштабованою, тобто дозволяє виділяти додаткові ресурси за потреби, і надає декілька рівнів узгодженості даних для балансу між узгодженістю та продуктивністю. За замовчуванням автоматично індексує всі дані.

Також використано хмарний сервіс Azure Blob Storage. Він призначений для зберігання величезних обсягів неструктурованих даних, таких як текстові або двійкові дані. Сховище забезпечує декілька рівнів зберігання та має високу масштабованість, доступність та рівень безпеки, підтримує резервування.

Для аналізу зображень використано Azure AI Vision, що надає доступ до передових алгоритмів, які обробляють зображення та повертають інформацію на основі візуальних характеристик, які цікавлять. Аналіз зображень за допомогою цього сервісу охоплює читання тексту із зображень, додавання тегів, класифікацію, виявлення людей і об'єктів, і це лише невелика частина його можливостей.

Таким чином, при завантажені зображень користувача, вони можуть бути класифіковані, додано теги та розпізнано текст, якщо він там наявний. Після цього зображення одного класу будуть автоматично згруповані, а при перегляді кожного фото окремо, можна побачити всі дані про нього.

Загальний вигляд програмної системи можна побачити на діаграмі розгортання (рис. 1), що показує основні компоненти системи та способи комунікації між ними.

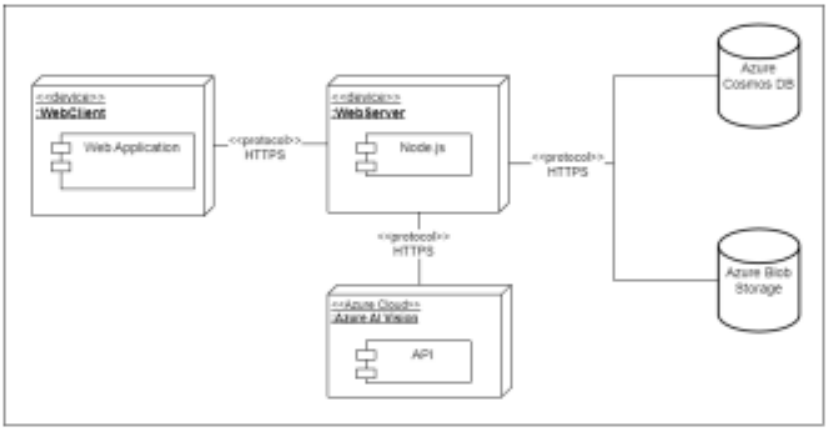


Рис. 1. **Діаграма розгортання програмної системи**

У роботі було представлено архітектуру та обґрунтовано вибір технологій для реалізації веб-орієнтованої системи гібридного механізму управління зображеннями для їх подальшої ефективної обробки та аналізу в умовах великих даних. Такий додаток дозволить користувачам легко працювати з великою кількістю власних та завантажених зображень.

Список використаних джерел:

1. Flanagan D. JavaScript: The Definitive Guide. 7th ed. O'Reilly Media, 2020. 706 p.

2. Терещенко Г. Ю., Боцюра І. С. Image warehouse architecture in the era of big data // 25-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті», Збірник Матеріалів форуму., т. 6, - Харків (ХНУРЕ), 20-22 квітня 2021 р., с. 163 - 164.

3. Tereshchenko G., Kyrychenko I., Chetverykov G. Overview of image storage models in Big Data conditions // Комп'ютерна математика в науці, інженерії та освіті (CMSEE-2020). – Матеріали V Всеукраїнської науково-технічної конференції – Полтава, 27 листопада 2020 р. – С. 18–21. УДК 00.89:004.043.

МОЛОДІЖНА НАУКОВА ЛІГА

Громадська організація «Молодіжна наукова ліга».
Номер запису в Реєстрі громадських об'єднань: 1508433.
Адреса: вул. Зодчих, буд. 40, офіс 103; м. Вінниця, Вінницька обл., 21037
Організація функціонує як відокремлений підрозділ ТОВ «UKRLOGOS Group».
ЄДРПОУ: 44574528
IBAN: UA783052990000026003018111950
Банк ВФ АТ Кб «ПриватБанк»; МФО 305299
Свідоцтво суб'єкта видавничої справи: ДК № 7172 від 21.10.2020.

ДОВІДКА

ПРО ПРИЙНЯТТЯ ТЕЗ ДО ПУБЛІКАЦІЇ

09.06.2024

Шановний(і) авторе(и):
Мандзинець Анатолій Володимирович,
Долготер Михайло Сергійович,

Організаційний комітет з радістю повідомляє, що тези «ПРОЕКТУВАННЯ ВЕБ-ОРІЄНТОВАНОЇ СИСТЕМИ ГІБРИДНОГО МЕХАНІЗМУ УПРАВЛІННЯ ЗОБРАЖЕННЯМИ ДЛЯ ЇХ ЕФЕКТИВНОЇ ОБРОБКИ ТА АНАЛІЗУ В УМОВАХ ВЕЛИКИХ ДАНИХ» прийняті до публікації в збірнику за матеріалами VI Всеукраїнської студентської наукової конференції «Експериментальні та теоретичні дослідження в контексті сучасної науки» (21.06.2024, м. Рівне, Україна).

Опубліковані тези будуть доступні з 21.06.2024 за посиланням:
https://archive.liga.science/index.php/conference-proceedings/issue/view/ukr-21.06.2024

. . . . . .

Електронні сертифікати учасників конференції та подяки науковим керівникам будуть доступні з 21 червня. Розсилка замовлених друкованих примірників, сертифікатів та подяк відбудеться до 5 липня.

Конференцію зареєстровано Державною науковою установою «УкрІНТЕІ» в базі даних науково-технічних заходів України та інформаційному бюлетені «План проведення наукових, науково-технічних заходів в Україні» (Посвідчення №34 від 05.01.2024).

З повагою,

Директор Молодіжної наукової ліги
Голова оргкомітету конференції
ІГОР КОРЕНЮК

МОЛОДІЖНА НАУКОВА ЛІГА

ДОДАТОК Е

Слайди презентації

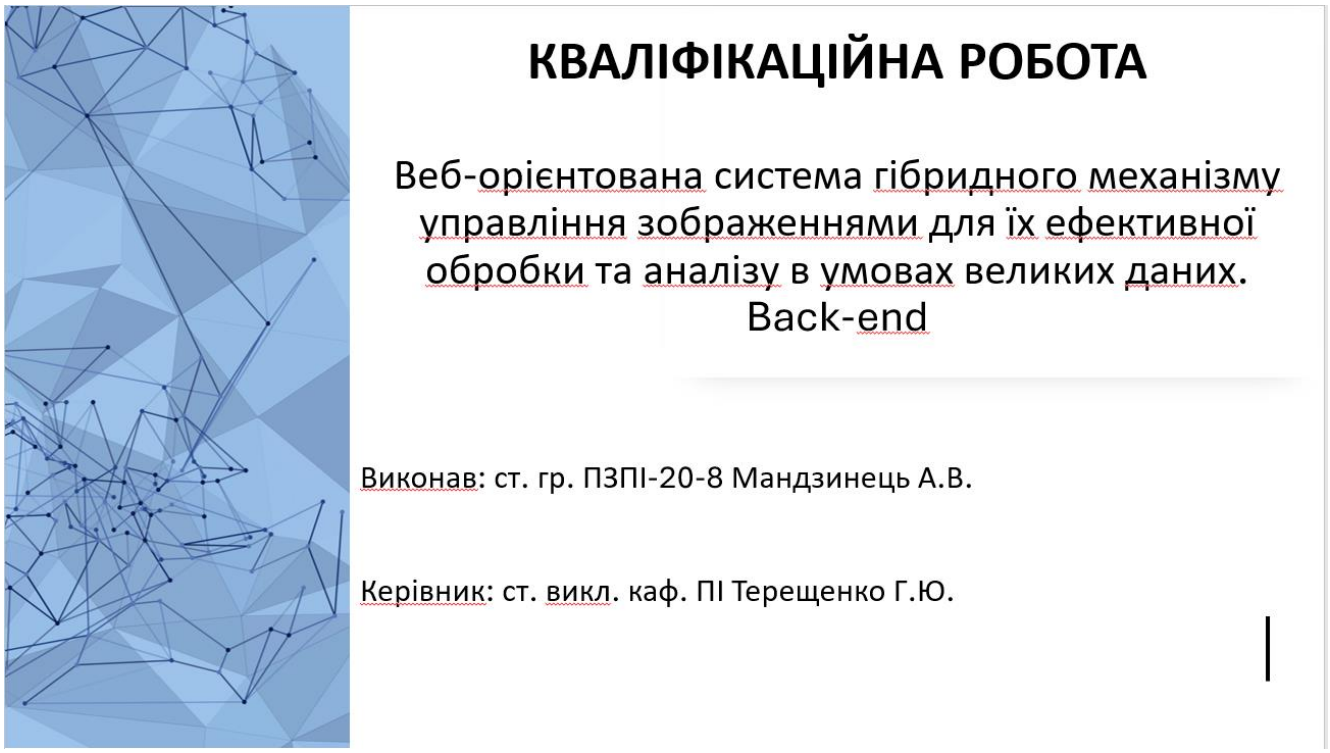


Рисунок Б.1 – Слайд № 1

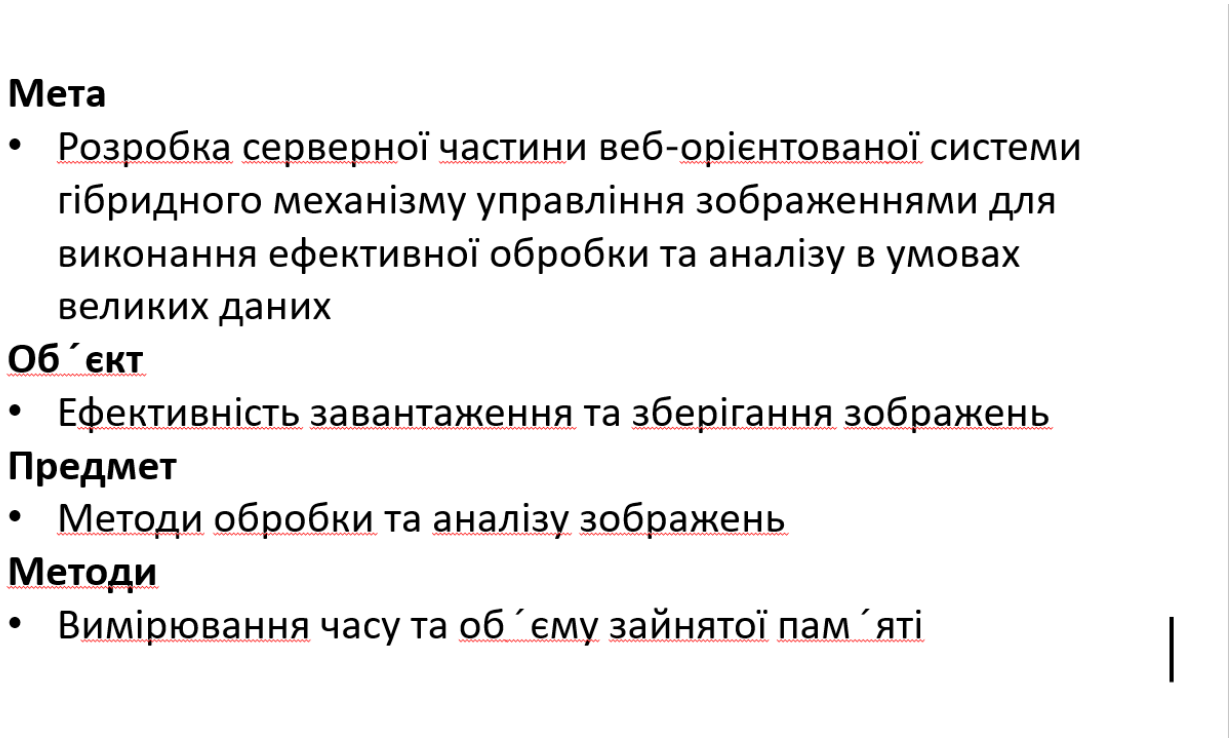


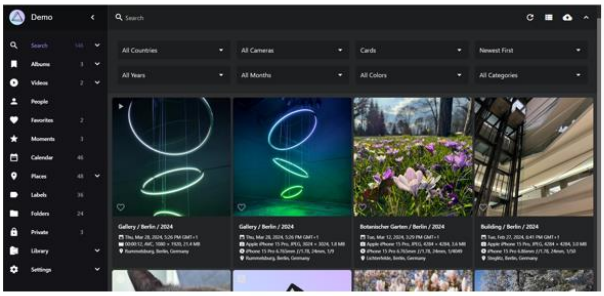
Рисунок Б.2 – Слайд № 2

Аналіз предметної галузі

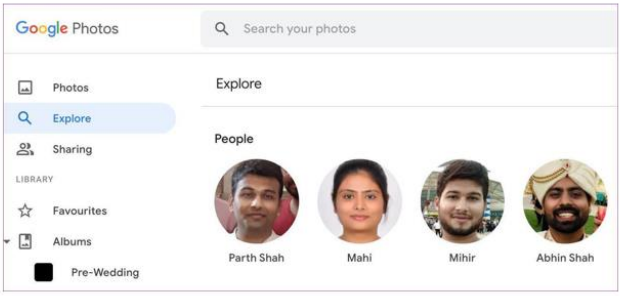
- Початок 2000-х – поява перших сервісів для зберігання зображень в Інтернеті
- Розвиток соціальних мереж та збільшення кількості зображень
- Поширення хмарних технологій

Рисунок Б.3 – Слайд № 3

Аналіз конкурентів



Приклад інтерфейсу PhotoPrism



Приклад інтерфейсу Google Photo

Рисунок Б.4 – Слайд № 4

Постановка задачі

Основні функції системи:

- Виконання обробки зображень
- Аналіз зображень
- Робота із класифікованими зображеннями
- Можливість поділитися зображеннями

Рисунок Б.5 – Слайд № 5

Використані технології

- Node.js
- Azure Cosmos DB
- Azure Blob Storage
- Azure AI Vision

Azure Blob Storage

Рисунок Б.6 – Слайд № 6

Архітектура проєкту

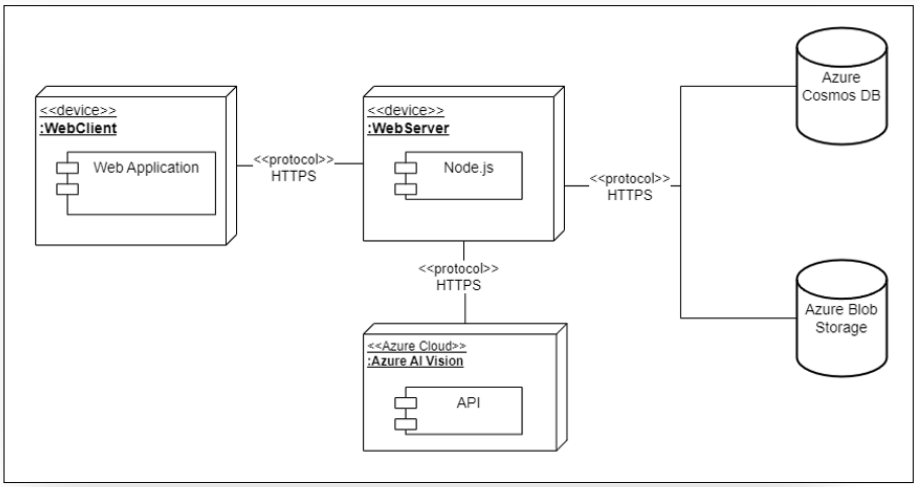


Рисунок Б.7 – Слайд № 7

Архітектура веб-сервера

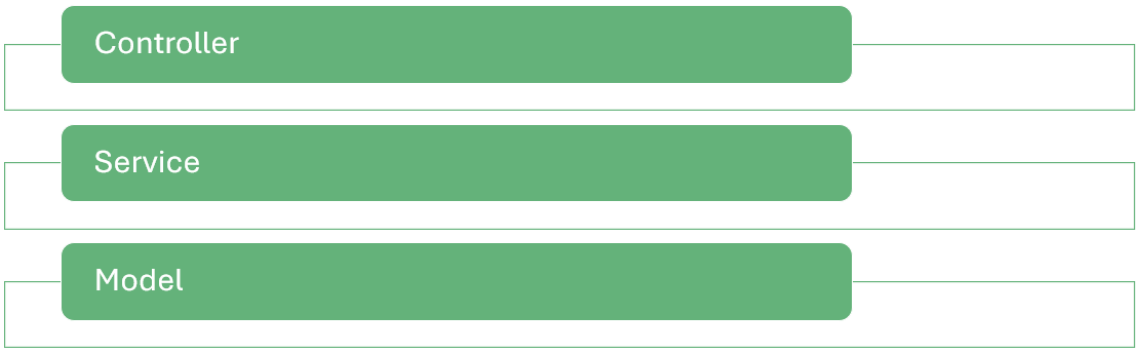


Рисунок Б.8 – Слайд № 8

ER-діаграма

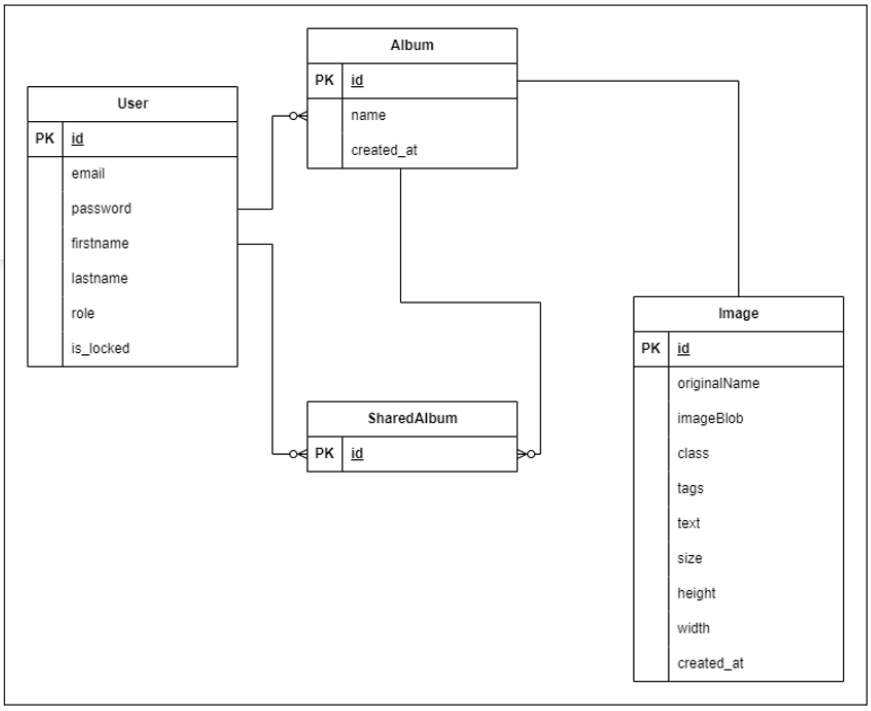


Рисунок Б.9 – Слайд № 9

Обробка та аналіз зображень

```
async function imageCompression(file) {
    const format = file.mimetype
    const formatOptions = {
        'image/jpeg': sharp(file.buffer).jpeg({ quality: 85 }),
        'image/png': sharp(file.buffer).png({ quality: 85 }),
        'image/webp': sharp(file.buffer).webp({ quality: 90 }),
        'image/tiff': sharp(file.buffer).tiff({ quality: 90 })
    };
    if (!formatOptions[format]) {
        throw new Error(`Unsupported image format: ${format}`);
    }
    return formatOptions[format].toBuffer();
}
```

Функція стиснення зображення

```
async function imageAnalysis(imageUrl,features) {
    const result = await cvClient.path('/imageanalysis:analyze').post({
        body: {
            url: imageUrl
        },
        queryParameters: {
            features: features
        },
        contentType: 'application/json'
    });
    let tags = []
    const classResult = result.body.tagsResult.values[0]?.name || 'unknown';
    result.body.tagsResult.values.forEach(value => {
        tags.push(value.name)
    })
    let resultText = []
    if (result.body.readResult) {
        result.body.readResult.blocks.forEach(block => {
            block.lines.forEach(line => {
                resultText.push(line.text)
            })
        })
    }
    const metadata = result.body.metadata
    return {tags, classResult, resultText, metadata}
}
```

Запит на виконання аналізу

Рисунок Б.10 – Слайд № 10

Дані користувача

```
async create(email, password, firstname, lastname) {
    const hashedPassword = await bcrypt.hash(password, 5)
    const user = new User(email, hashedPassword, firstname, lastname)
    const querySpec = {
        query: 'SELECT c.id FROM c WHERE c.email = @email',
        parameters: [
            {
                name: '@email',
                value: email
            }
        ]
    };
    const existUser = await user.find(querySpec)
    if (existUser) {
        throw new Error("User already exists");
    }
    await user.create()
    const {hashPassword, ...userData} = user
    return userData;
}
```

Використання хешування при зберіганні паролю

```
module.exports = function(role) {
    return function (req, res, next) {
        if (req.method === "OPTIONS") {
            next()
        }
        try {
            const token = req.headers.authorization.split(' ')[1]
            if (!token) {
                return res.status(401).json({
                    status: "error", message: "Unauthorized!"
                })
            }
            const decoded = jwt.verify(token, process.env.SECRET_KEY)
            if (decoded.role !== role) {
                return res.status(403).json({
                    status: "error", message: "Access denied!"
                })
            }
            req.user = decoded;
            next()
        } catch (e) {
            res.status(401).json({
                status: "error", message: "Unauthorized!"
            })
        }
    };
}
```

Використання JSON токена для визначення ролі користувача

Рисунок Б.11 – Слайд № 11

Тестування


```
describe("POST /api/user/registration", () => {
    test("should register new user", async () => {
        const data = {
            firstname: "John",
            lastname: "Doe",
            email: "john@doe.com",
            password: "123456"
        };
        const response = await request(app)
        .post("/api/user/registration")
        .send(data);
        id = response.body.user.id;
        expect(response.statusCode).toBe(200);
    });
});
```

Тестування реєстрації

```
PASS  test/user.test.js (6.207 s)
PASS  test/imageAlbum.test.js (9.24 s)

Test Suites: 2 passed, 2 total
Tests:       15 passed, 15 total
Snapshots:   0 total
Time:        9.675 s, estimated 11 s
Ran all test suites.
```

Повідомлення про успішне виконання тестів

Рисунок Б.12 – Слайд № 12

Висновки

- Аналіз предметної області
- Формування вимог
- Проектування програмної системи
- Розробка
- Тестування
- Тези конференції (наведено у додатку Г)

Рисунок Б.13 – Слайд № 13

ДЕМОНСТРАЦІЯ

Рисунок Б.14 – Слайд № 14

ДЯКУЮ ЗА УВАГУ!

Рисунок Б.15 – Слайд № 15