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)

Програмна система гібридного сховища даних для оптимізації та покращення обробки медіафайлів
(тема)

Виконав:
студент 4 курсу, групи ПЗПІ-20-3
Калашников П. А
(прізвище, ініціали)

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

Тип програми освітньо-професійна

Освітньо-професійна програма Програмна інженерія
(повна назва освітньої програми)

Керівник: доц. Кириченко І.В
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ З.В. Дудар
(підпис)

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

Освітньо-професійна програма Програмна інженерія

(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Калашникову Павлу Андрійовичу

(прізвище, ім'я, по батькові)

1. Тема роботи (проекту) Програмна система гібридного сховища даних для оптимізації та покращення обробки медіафайлів
затверджена наказом по університету від «*20*» *травня* 2024 р. № *471Ст*
2. Термін здачі студентом закінченої роботи « 14 » червня 2024 р.
3. Вихідні дані до роботи методи розробки програмних продуктів, методи розробки програмних додатків
4. Зміст пояснювальної записки (перелік питань, що потрібно розробити) мета роботи, аналіз проблемної галузі і постановка задачі, опис вимог до програмної системи, опис використовуваних методів та алгоритмів, опис розробленої програмної системи, аналіз можливих застосувань, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної галузі, огляд існуючих рішень, вибір найбільш придатних аналогів	01.06 – 2.06.2024	*виконано*
2	Створення Специфікації ПЗ	3.06 – 5.06.2024	*виконано*
3	Проектування та розробка ПЗ	6.06 – 09.06.2024	*виконано*
4	Тестування та дослідна експлуатація ПЗ	09.06 – 10.06.2024	*виконано*
5	Написання пояснювальної записки	08.06 – 10.06.2024	*виконано*
6	Перевірка пояснювальної записки керівником, підготовка роботи для проходження перевірки на	10.06 – 11.06.2024	*виконано*
7	Оцінка роботи рецензентом, отримання відзиву від керівника атестаційної роботи, попередній захист роботи	10.06 – 13.06.2024	*виконано*
8	Здача роботи у електронний архів, допуск роботи до захисту завідувачем кафедри	10.06 – 13.06.2024	*виконано*
9	Захист атестаційної роботи	14.06.2024	

Дата видачі завдання «01» травня 2024 р.

Студент _____ Калашников П.А

(підпис)

Керівник роботи _____ доц. кафедри ПІ Кириченко І.В

(підпис)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра: 54 с., 5 рис., 4 таблиці, 15 джерел.

ГІБРИДНЕ СХОВИЩЕ, МЕДІАФАЙЛИ, ОПТИМІЗАЦІЯ, ОБРОБКА, API, JAVASCRIPT, POSTGRESQL, STORAGE.

Об’єкт розробки – програмна система гібридного сховища даних для оптимізації та покращення обробки медіафайлів.

Мета розробки – створити систему, що забезпечить ефективне зберігання та обробку медіафайлів, поєднуючи локальне і хмарне сховища для оптимізації швидкості доступу і збереження безпеки даних.

Метод рішення – використання API хмарного сховища, мова програмування JavaScript, СУБД PostgreSQL для управління метаданими та забезпечення ефективного доступу до медіафайлів.

В результаті розробки створено програмну систему, яка дозволяє оптимізувати обробку медіафайлів за допомогою гібридного сховища, що поєднує локальні та хмарні ресурси.

HYBRID STORAGE, MEDIA FILES, OPTIMIZATION, PROCESSING, API, JAVASCRIPT, POSTGRESQL, CLOUD STORAGE.

Object of development – a software system for hybrid data storage to optimize and improve the processing of media files.

The goal of the development is to create a system that provides efficient storage and processing of media files, combining local and cloud storage to optimize access speed and ensure data security.

Solution method – using cloud storage API, JavaScript programming language, PostgreSQL DBMS for metadata management and efficient access to media files.

As a result of the development, a software system was created that optimizes the processing of media files through a hybrid storage system combining local and cloud resources.

Я, Калашников Павло Андрійович, студент гр. ПЗПІ-20-3, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна система гібридного сховища даних для оптимізації та покращення обробки медіафайлів», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі 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 в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Зростання обсягу медіафайлів та потреба в їх ефективному зберіганні та обробці призвели до розвитку гібридних систем зберігання даних, що поєднують локальні та хмарні ресурси. Такі системи дозволяють забезпечити оптимальне використання доступних ресурсів, підвищуючи продуктивність та безпеку обробки даних.

Гібридне сховище даних є важливим інструментом для роботи з великими обсягами медіафайлів.[1] Воно дозволяє зберігати дані як на локальних серверах, так і в хмарі, що забезпечує високу доступність, надійність та швидкість доступу до файлів. Використання гібридних сховищ дозволяє досягти балансу між вартістю зберігання та швидкістю доступу до даних, що особливо важливо для компаній та організацій, які працюють з великими обсягами інформації.

Мета даної кваліфікаційної роботи полягає у створенні програмної системи гібридного сховища даних для оптимізації та покращення обробки медіафайлів. Така система допоможе ефективно зберігати та обробляти великі обсяги медіаданих, забезпечуючи при цьому високу швидкість доступу та безпеку інформації.

Актуальність цієї теми полягає у зростаючій потребі в ефективному зберіганні та обробці медіафайлів у сучасному світі. З розвитком технологій та збільшенням обсягів даних, що генеруються, виникає необхідність у системах, які можуть забезпечити швидкий доступ до даних та їх надійне зберігання. Гібридне сховище даних є одним із найбільш перспективних рішень, яке дозволяє оптимізувати використання ресурсів та забезпечити високий рівень безпеки [2].

Програмна система буде реалізована у вигляді веб-додатку, що дозволить користувачам з легкістю інтегрувати її у свої існуючі робочі процеси. Веб-додатки є одними з найпопулярніших рішень у розробці програмного забезпечення сьогодні, оскільки вони забезпечують високу доступність та простоту використання. Єдиною вимогою для використання системи є наявність

встановленого сучасного веб-браузера, що робить її доступною для широкого кола користувачів.

Зокрема, завдання цієї роботи полягають у наступному:

- реалізація автентифікації та реєстрації адміністраторів;
- пошук логів та файлів за певними критеріями та метриками;
- зберігання, аналіз та класифікація логів та файлів;
- моніторинг часу та швидкості завантаження/вивантаження;
- основні операції редагування логів та файлів;
- оптимізація інтерфейсу системи для швидкої реакції та продуктивності на різних пристроях і розмірах екранів;
- забезпечення послідовного та безперебійного адміністративного досвіду;
- інтегрування механізмів моніторингу використання системних ресурсів;
- надання адміністраторам у режимі реального часу інформації про використання пам'яті та ємності сховища.

Таким чином, створення програмної системи гібридного сховища даних для оптимізації та покращення обробки медіафайлів є актуальним та важливим завданням, яке сприятиме підвищенню ефективності роботи з великими обсягами даних та забезпеченню їх надійного зберігання та швидкого доступу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

При розробці програмної системи для гібридного сховища даних для оптимізації та покращення обробки медіафайлів необхідно провести аналіз сучасного стану предметної галузі. Основні аспекти аналізу можуть включати:

Тенденції ринку. Дослідження поточних тенденцій у сфері зберігання та обробки медіафайлів виявляє, що все більше компаній переходять на хмарні та гібридні рішення. Це пов’язано з необхідністю гнучкості, масштабованості та зниження витрат на інфраструктуру. Також спостерігається зростаючий попит на інтеграцію з інструментами штучного інтелекту для автоматичної обробки та аналізу медіафайлів[3].

Технологічні рішення. Огляд сучасних технологічних рішень у галузі зберігання даних та обробки медіафайлів. Це включає в себе дослідження різних типів сховищ даних (локальні, хмарні, гібридні), програмного забезпечення для обробки медіафайлів, а також інші технології, такі як штучний інтелект, машинне навчання та аналіз даних.

Проблеми та виклики. Визначення основних проблем та викликів, з якими стикаються користувачі при роботі з медіафайлами. Це може включати складнощі у пошуку та організації файлів, обмеження зберігання та доступу до даних, а також потребу у підвищенні продуктивності та ефективності роботи з медіа.

Потреби користувачів. Вивчення потреб та вимог користувачів щодо зберігання та обробки медіафайлів. Це може бути здійснено шляхом співбесід з потенційними користувачами, опитування або аналізу відгуків та коментарів користувачів до існуючих рішень [4].

Для кращого розуміння сучасних рішень у цій галузі, було проведено аналіз функціональності існуючих програмних засобів, який представлено у Таблиці 1.1.

Таблиця 1.1 – Аналіз функціональності існуючих програмних засобів (таблиця виконана самостійно)

Система	Переваги	Недоліки	Що буде в нашій системі
Google Photos	AI-пошук, автоматичні альбоми, необмежене сховище	Обмеження на розмір файлів, питання конфіденційності	Підтримка різних форматів, гібридне зберігання
Dropbox	Інтеграція з іншими сервісами, зручний інтерфейс, безпека	Обмежений обсяг сховища, відсутність інструментів обробки	Автоматизована обробка
Amazon S3	Масштабованість, надійність, інтеграція з AWS	Складність налаштування, висока вартість	Зручний інтерфейс, автоматизація обробки медіафайлів

З таблиці видно, що існуючі рішення мають свої переваги та недоліки. Наша система прагне об'єднати найкращі риси цих продуктів і вирішити їхні основні проблеми. Наприклад, підтримка різних форматів медіафайлів дозволить користувачам швидко і ефективно обробляти великі обсяги даних. Крім того, гібридне сховище забезпечить гнучкість і надійність зберігання даних, поєднуючи локальні і хмарні ресурси.

1.2 Виявлення та вирішення проблем

Для виявлення та вирішення проблем у галузі зберігання та обробки медіафайлів важливо провести ретельний аналіз і визначити основні аспекти, які потребують уваги:

Інтеграція сучасних технологій: впровадження сучасних технологій для зберігання та обробки медіафайлів, таких як хмарні сервіси та гібридні сховища. Це дозволить підвищити продуктивність, масштабованість та безпеку системи.

Забезпечення безпеки даних: розробка та впровадження механізмів захисту даних, таких як шифрування, контроль доступу та аутентифікація користувачів. Це допоможе запобігти несанкціонованому доступу та забезпечити конфіденційність інформації[5].

Підвищення продуктивності системи: оптимізація архітектури програмної системи та використання високопродуктивного обладнання. Це дозволить забезпечити швидкий доступ до медіафайлів та ефективну обробку великого обсягу даних.

Підтримка користувачів: розробка системи підтримки користувачів, яка включає документацію, навчальні матеріали та службу технічної підтримки. Це допоможе користувачам швидко освоїти систему та вирішувати можливі проблеми.

Моніторинг та аналіз продуктивності: впровадження інструментів для моніторингу та аналізу продуктивності системи. Це дозволить виявляти проблеми на ранніх стадіях та своєчасно вживати заходів для їх усунення.

1.3 Постановка задачі

Постановка задачі розробки програмної системи гібридного сховища даних для оптимізації та покращення обробки медіафайлів:

Розробка функціональності. визначення основних функцій та можливостей програмної системи, таких як зберігання медіафайлів, їхній пошук, організація, обробка та спільний доступ до них.

Забезпечення безпеки: розробка механізмів захисту даних, включаючи контроль доступу, шифрування та моніторинг активності користувачів.

Оптимізація продуктивності: покращення швидкодії та ефективності обробки медіафайлів, зменшення часу реакції системи та оптимізація використання ресурсів.

Реалізація інтерфейсу користувача: розробка зручного та інтуїтивно зрозумілого інтерфейсу, який дозволить користувачам легко взаємодіяти з програмною системою.

Тестування та валідація: проведення ретельного тестування програмної системи з метою виявлення та виправлення помилок, а також перевірка відповідності розробленого продукту поставленим вимогам та очікуванням користувачів.

Документування і підтримка: розробка документації для користувачів та адміністраторів системи, а також надання підтримки користувачам після впровадження програмної системи.

Масштабованість та розширюваність: забезпечення можливості масштабування та розширення програмної системи у майбутньому відповідно до потреб користувачів та змін у бізнес-вимогах [6].

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Вимоги до клієнтської частини

Клієнтською частиною нашої системи буде веб-додаток, розроблений з використанням HTML, CSS та JavaScript.

Основні функціональні вимоги до клієнтської частини включають:

- можливість завантажувати, переглядати та видаляти медіафайли;
- інтерактивний інтерфейс для перегляду та редагування метаданих медіафайлів;
- забезпечення швидкого доступу до медіафайлів для користувачів.

2.2 Вимоги до серверної частини

Серверна частина буде реалізована з використанням JavaScript та Node.js. Виходячи з вимог до клієнтської частини, серверна частина повинна забезпечувати:

- зберігання даних медіафайлів у базі даних PostgreSQ;
- підтримка CRUD операцій (створення, читання, оновлення, видалення) для медіафайлів;
- забезпечення безпеки та аутентифікації користувачів;
- підтримка масштабованості для обробки великої кількості медіафайлів;
- відповідність умовам ліцензії GNU GPL v3.проблем та їх швидкого усунення [7].

2.3 Вимоги до інфраструктури

Для забезпечення максимальної відмовостійкості та швидкодії роботи системи, було вирішено використовувати СУБД PostgreSQL.

Основні вимоги до інфраструктури включають:

- використання високопродуктивних серверів для забезпечення швидкого доступу до медіафайлів;
- розгортання бази даних PostgreSQL для зберігання метаданих медіафайлів та інформації про користувачів;

- використання pgAdmin 4 для управління базою даних;
- налаштування резервного копіювання бази даних та файлів;
- використання хмарних сервісів для розширення обсягу сховища медіафайлів та забезпечення їх доступності з будь-якої точки світу.

2.4 Вимоги до інтерфейсу користувача

Інтерфейс користувача є важливим компонентом будь-якої програмної системи, оскільки він безпосередньо впливає на зручність використання та загальний користувацький досвід.

Вимоги до інтерфейсу користувача включають:

- інтерфейс повинен бути зрозумілим та інтуїтивно зрозумілим, забезпечуючи легкий доступ до всіх функціональних можливостей системи;
- дизайн інтерфейсу повинен бути адаптивним, забезпечуючи коректне відображення та функціонування на різних пристроях та розмірах екранів;
- забезпечення доступності інтерфейсу для користувачів з обмеженими можливостями, дотримуючись стандартів WCAG (Web Content Accessibility Guidelines) [8];
- інтерфейс повинен надавати швидкий та чіткий відгук на дії користувача, забезпечуючи зручність та ефективність взаємодії з системою.

2.5 Вимоги до користувацького досвіду

Забезпечення позитивного користувацького досвіду є критично важливим для успішної роботи програмної системи.

Вимоги до користувацького досвіду включають:

- оптимізація клієнтської частини для забезпечення швидкого завантаження веб-додатку навіть при повільному інтернет-з'єднанні;
- створення логічної та простої навігаційної структури, що дозволяє користувачам легко знайти необхідні функції та інформацію;

- використання сучасних дизайнів та графічних елементів для створення привабливого та професійного вигляду веб-додатку;
- впровадження механізмів для збору відгуків користувачів та аналізу їхньої задоволеності роботою системи. Це дозволить ідентифікувати проблеми та впроваджувати покращення на основі реальних потреб користувачів [9].

2.6 Вимоги до підтримки та обслуговування

Підтримка та обслуговування системи після її розгортання є важливими для забезпечення її стабільної та безперебійної роботи.

Вимоги до підтримки та обслуговування включають

- впровадження системи моніторингу для відстеження стану системи, виявлення та реагування на можливі проблеми;
- регулярне оновлення системи для виправлення помилок, покращення функціональності та забезпечення безпеки;
- налаштування регулярного резервного копіювання даних для запобігання втратам інформації у випадку збоїв або аварій;
- забезпечення ефективної підтримки користувачів, що включає швидке реагування на запити та надання необхідної допомоги.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. UML проектування ПЗ

Моделювання програмного забезпечення дозволяє вирішити такі завдання:

- візуалізація системи в її поточному або бажаному стані;
- опис структури та поведінки системи;
- створення шаблону, що дозволяє конструювати систему;
- документування прийнятих рішень.

Для нашої системи також було використано мову UML для документації програмного забезпечення. Протягом проектування системи було створено кілька діаграм UML для опису різних аспектів системи. Однією з таких діаграм є діаграма прецедентів (див. рис. 3.1). Основне призначення цієї діаграми – опис функціональності і поведінки системи, що дозволяє замовнику, кінцевому користувачеві і розробнику спільно обговорювати проектовану або існуючу систему.

На рис. 3.1 представлено діаграму прецедентів, яка описує основні функціональні вимоги до розроблюваної системи гібридного сховища даних для оптимізації та покращення обробки медіафайлів.

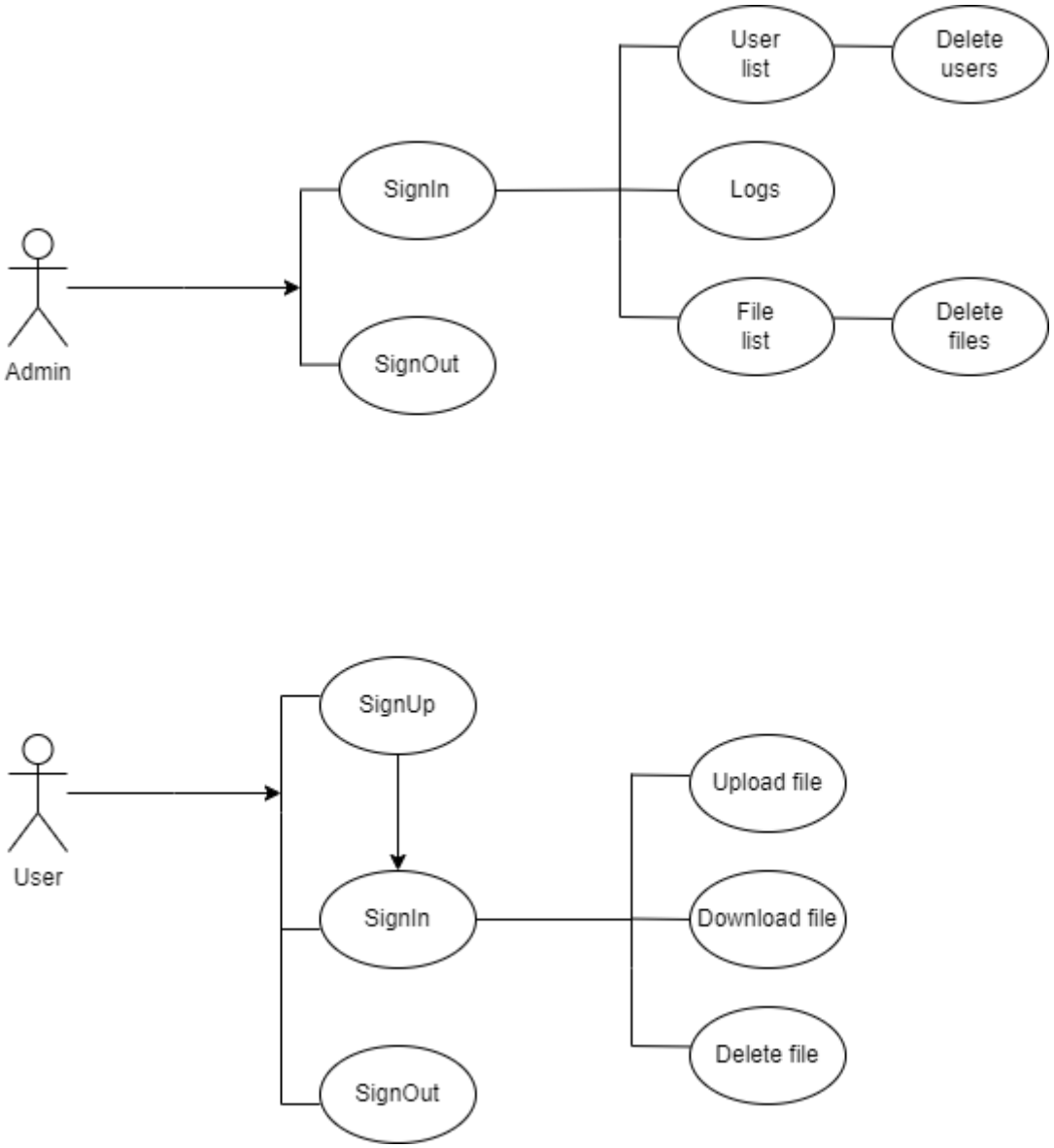


Рисунок 3.1 – Діаграма прецедентів (рисунок створено самостійно)

Діаграма прецедентів показує, які дії можуть виконувати користувачі системи і як ці дії взаємодіють із системою. Вона допомагає визначити вимоги до системи і забезпечити спільне розуміння між всіма учасниками проекту.

3.2. Проектування архітектури ПЗ

Програмна система для гібридного хмарного сховища даних реалізована за принципом клієнт-серверної архітектури, що забезпечує оптимальну організацію взаємодії між різними компонентами мережі. У цій архітектурі сервери виконують роль постачальників специфічних функцій, таких як зберігання, обробка та

управління даними, тоді як клієнти є споживачами цих послуг. Цей підхід дозволяє знизити вимоги до апаратного забезпечення клієнтських пристроїв, оскільки основні обчислювальні операції виконуються на сервері. Крім того, клієнт-серверна архітектура відзначається гнучкістю та підвищеним рівнем безпеки, що важливо для адміністрування локальних мереж.

В процесі розробки програмної системи було спроектовано компоненти:

- веб-сервер: відповідальний за бізнес-логіку. Цей компонент побудований за REST принципами, тобто мережа повинна складатися з клієнтів і серверів, сервери і клієнти не мають стану, відстежування стану один одного не потрібне, а між серверами і клієнтами існує спільна мова, яка дозволяє кожній частині бути замінною або змінною без порушення цілісності системи[10];
- сервер автентифікації та авторизації: механізм автентифікації та авторизації забезпечує безпечний доступ користувачів до системи. Використовуються стандартні методи автентифікації, такі як логін і пароль (див рис. 3.2);
- веб-клієнт: представляє нашу клієнтську частину. Клієнтська частина реалізує користувальницький інтерфейс, формує запити до сервера і обробляє відповіді від нього. Вона реалізована за допомогою фреймворку React. Веб-клієнт дозволяє користувачам завантажувати медіафайли, переглядати та оптимізувати файли для зберігання;
- база даних: використовується PostgreSQL, яка забезпечує високу продуктивність, надійність та безпеку даних. Для управління базою даних використовується pgAdmin 4. PostgreSQL забезпечує оптимізоване зберігання медіафайлів та їх метаданих, а також дозволяє виконувати складні запити для аналітики [11].

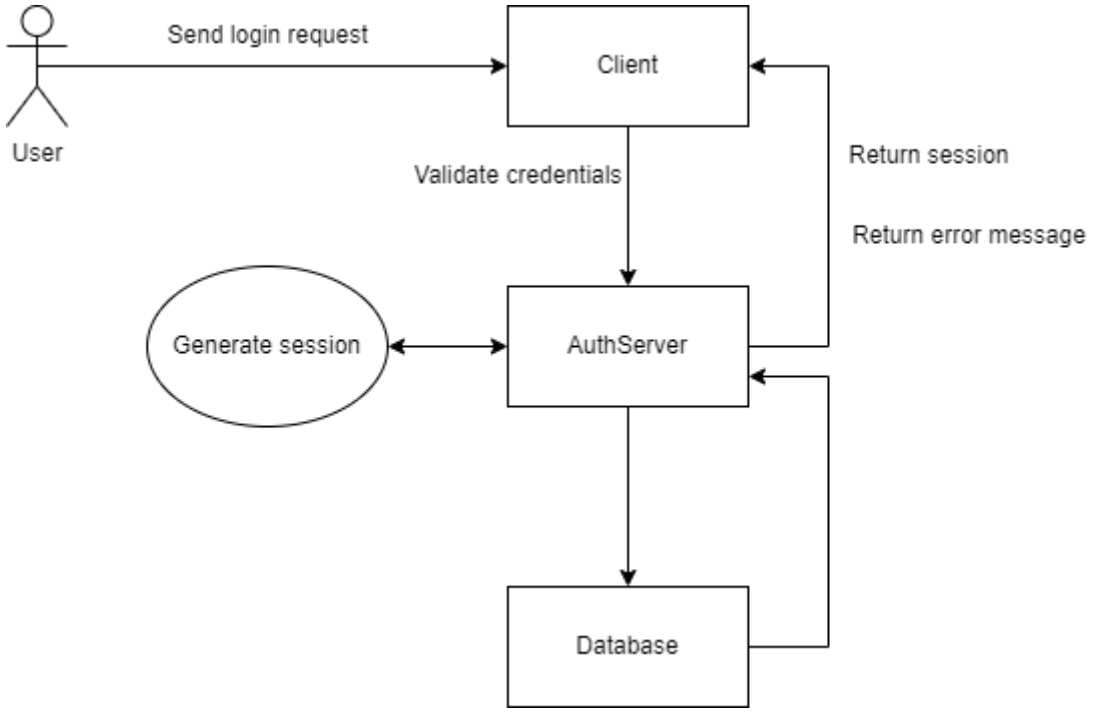


Рисунок 3.2 – Діаграма роботи автентифікації (рисунок створено самостійно)

Взаємодія в цій архітектурі розпочинається клієнтом, який надсилає запит до сервера. Сервер відповідає на запит, вказуючи, чи може він надати послугу клієнту і за яких умов.

3.2.1 Проектування архітектури серверної частини

В програмній системі для гібридного сховища даних було вирішено використати клієнт-серверну архітектуру. Ця архітектурна модель забезпечує кілька важливих переваг, які є критично важливими для нашої системи:

- безпека: Оскільки обробка даних здійснюється на сервері, це забезпечує можливість застосування централізованих політик безпеки і захисту даних. Сервер може бути розташований в захищеному середовищі з обмеженим доступом, що значно зменшує ризик несанкціонованого доступу;
- масштабованість: Клієнт-серверна архітектура дозволяє легко масштабувати систему шляхом додавання нових серверів або збільшення

потужності існуючих. Це особливо важливо для систем, що обробляють великі обсяги даних, як наші медіафайли;

- гнучкість: Архітектура клієнт-сервер дозволяє легко оновлювати або замінювати компоненти системи. Наприклад, клієнтську частину можна оновлювати без впливу на серверну частину і навпаки;
- централізоване управління: Всі дані та логіка обробки знаходяться на сервері, що дозволяє централізовано керувати системою. Це спрощує підтримку і адміністрування системи[12];
- зниження навантаження на клієнтів: Оскільки більшість обчислювальних операцій виконується на сервері, вимоги до клієнтських пристроїв знижуються. Це дозволяє використовувати менш потужні пристрої для доступу до системи.

3.2.2 Проектування архітектури клієнтської частини

У нашому проекті клієнтська частина реалізована за допомогою React, що є популярним фреймворком для створення інтерфейсів користувача. Архітектура клієнтської частини складається з наступних шарів:

Архітектура клієнтської частини нашого проекту виглядає наступним чином:

- презентаційний шар: компоненти React, які відображають дані на екрані користувача;
- бізнес-логічний шар: функції, які виконують логіку бізнес-процесів, такі як авторизація, аутентифікація, обробка запитів до сервера та ін;
- шар доступу до даних: функції, які виконують запит до сервера, отримують дані та обробляють їх для подальшого використання в презентаційному шарі;
- шар зберігання даних: локальне сховище даних, таке як PostgreSQL, яке використовується для зберігання даних, які не повинні бути відправлені на сервер.

Така архітектура клієнтської частини дозволяє розділяти логіку бізнес-процесів від презентаційного шару, що полегшує розробку та підтримку проекту. Крім того, використання локального сховища даних дозволяє зменшити навантаження на сервер та збільшити швидкість роботи клієнтської частини.

3.3 Проектування структури зберігання даних

Проектування структури зберігання даних є важливим етапом розробки будь-якої системи. В розроблюваній системі було вирішено використовувати комбінацію IndexedDB та PostgreSQL (див рис 3.3) для зберігання інформації про користувача:

- IndexedDB використовується для зберігання даних в локальному сховищі браузера, що забезпечує швидкий доступ до них та економить ресурси сервера. Це дозволяє нам зберігати дані про користувача, такі як сесії, налаштування та ін.;
- PostgreSQL, з іншого боку, є реляційною базою даних, що дозволяє зберігати великі обсяги даних в структурному форматі. Використовуємо PostgreSQL для зберігання даних про користувача, таких як ім'я, прізвище, електронна пошта, пароль та ін.

Структура сховища даних складається з наступних елементів:

- сесії (IndexedDB): тут зберігаються дані про активні сесії користувача, такі як дата початку та закінчення сесії, IP-адреса та інше;
- користувачі (PostgreSQL): тут зберігаються дані про користувача, такі як ім'я, прізвище, електронна пошта, пароль та інше;
- логи (PostgreSQL): тут зберігаються усі дії користувача;
- локальні налаштування PostgreSQL): тут зберігаються налаштування користувача, такі як мову інтерфейсу, тему та інше.

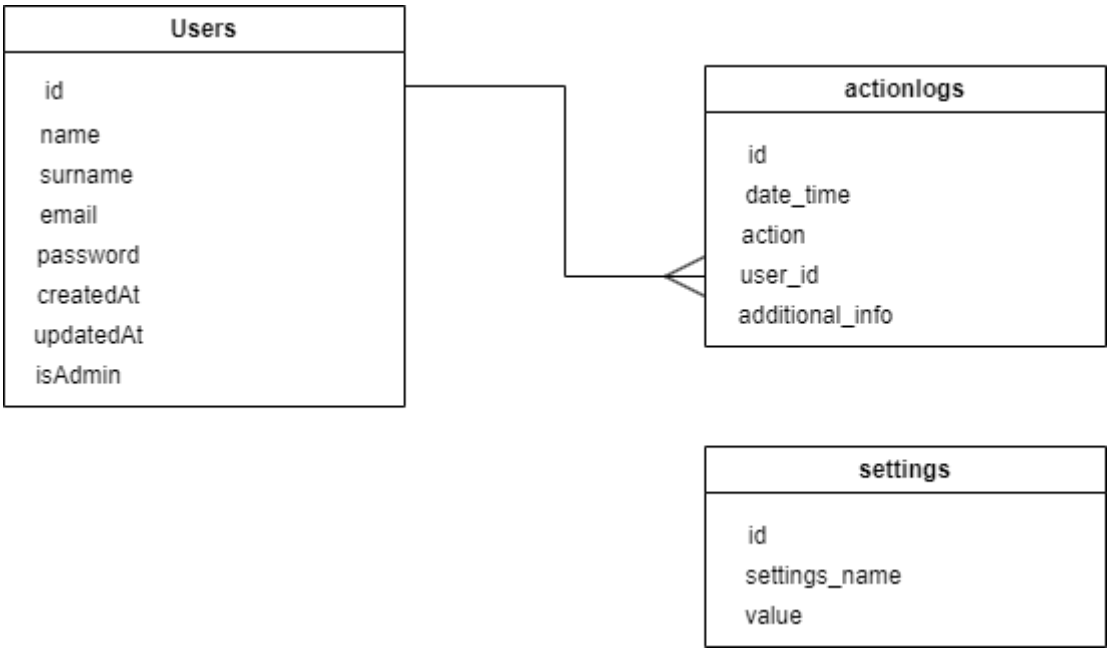


Рисунок 3.3 – ER-діаграма (рисунок створено самостійно)

Така структура дозволяє ефективно організовувати та зберігати дані про користувача, що забезпечує швидкий доступ до них та економить ресурси сервера.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

Програмна система була реалізована на основі принципу клієнт-серверної архітектури, що дозволяє організувати взаємодію в мережі між серверами, які надають специфічні функції (сервісів), та клієнтами, які використовують ці функції. Це дозволяє зменшити вимоги до машин клієнтів, оскільки більша частина обчислювальних операцій буде виконуватися на сервері, а також забезпечує гнучкість та безпеку системи. Крім того, така архітектура дозволяє легко додавати нові функції та сервіси, що поліпшує загальну функціональність системи [13].

Клієнтська частина системи була реалізована за допомогою JavaScript та бібліотеки React, що дозволяє створити інтерактивний інтерфейс користувача.

Нижче наведено приклад коду клієнтської частини:

```
import React, { useState, useEffect } from 'eact';
import axios from 'axios';

function App() {
  const [files, setFiles] = useState([]);
  const [selectedFile, setSelectedFile] = useState(null);

  useEffect(() => {
    axios.get('/files')
     .then(response => {
        setFiles(response.data);
      })
     .catch(error => {
        console.error(error);
      });
  }, []);

  const handleFileSelect = (file) => {
    setSelectedFile(file);
  };

  const handleUpload = () => {
    const formData = new FormData();
    formData.append('file', selectedFile);

    axios.post('/upload', formData)
     .then(response => {
        console.log(response.data);
      })
     .catch(error => {
        console.error(error);
      });
  };
```

```
  return (
    <div>
      <h1>Files in storage:</h1>
      <ul>
        {files.map(file => (
          <li key={file.id}>
            <a href="#" onClick={() => handleFileSelect(file)}>
              {file.name}
            </a>
          </li>
        ))}
      </ul>
      <button onClick={handleUpload}>Select file</button>
    </div>
  );
}

export default App;
```

Серверна частина системи була реалізована за допомогою Node.js та бібліотеки Express, що дозволяє створити RESTful API.

Нижче наведено приклад коду серверної частини:

```
const express = require('express');
const app = express();
const multer = require('multer');

const upload = multer({ dest: './uploads/' });

app.get('/files', async (req, res) => {
  try {
    const [files] = await storage.bucket(bucketName).getFiles();
    res.json(files.map(file => file.name));
  } catch (error) {
    console.error('Error fetching files:', error);
    res.status(500).send('Error fetching files');
  }
});
app.post("/upload", upload.single("file"), async (req, res) => {
  try {
    const userId = req.headers.user_id;

    if (!req.file) {
      return res.status(400).send("No file uploaded");
    }
    const bucket = storage.bucket(bucketName);
    const file = bucket.file(req.file.originalname);

    await file.save(req.file.buffer, {
      metadata: {
        contentType: req.file.mimetype,
      },
```

```
    });

    res.send("File uploaded successfully");
  } catch (error) {
    console.error("Error uploading file:", error);
    res.status(500).send("Error uploading file");
  }
});
```

У наведеному вище коді ми використовуємо бібліотеку multer для обробки файлів, що завантажуються на сервер. Ми також використовуємо express.json() та express.urlencoded({ extended: true }) для обробки запитів у форматі JSON та URL-encoded.

У ендпоінті GET /files ми отримуємо список файлів з бакету сховиища та повертаємо їх у форматі JSON.

У ендпоінті POST /upload ми використовуємо middleware upload.single("file") для обробки файлу, що завантажується на сервер. Ми також отримуємо ідентифікатор користувача з заголовка user_id та зберігаємо файл у бакеті хранилища з використанням методу save об'єкта file.

На рисунку 4.1 зображено інтерфейс завантаження файлів у сховище.

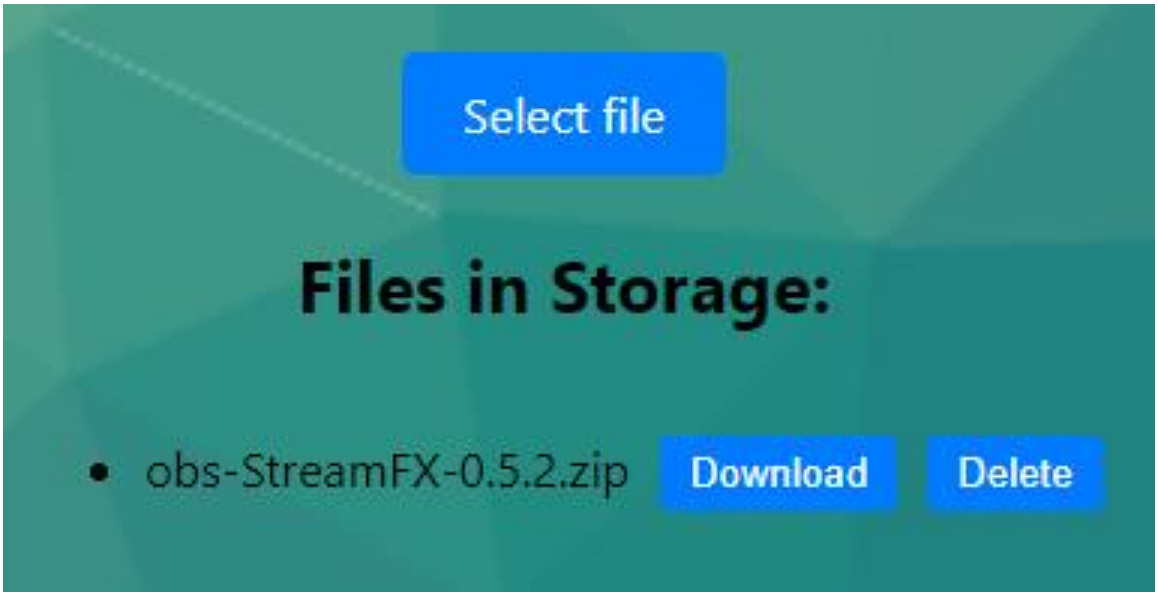


Рисунок 4.1 – Інтерфейс завантаження файлів у сховище (рисунок створено самостійно)

Адміністративна панель була реалізована з використанням JavaScript та бібліотеки React, що дозволяє створити інтерактивний інтерфейс користувача. У адміністративній панелі відображаються логи, користувачі, файли та інші дані, а також надається можливість видалення та редагування цих даних.

На рисунку 4.2 зображено інтерфейс адміністративної панелі.

Нижче наведено приклад коду клієнтської частини:

```
const fetchLogs = async (offset) => {
  try {
    const response = await axios.get('http://localhost:8000/logs', {
      params: { limit, offset }
    });
    setLogs(response.data.logs);
    setTotalCount(response.data.totalCount);
  } catch (error) {
    console.error(error);
  }
};

const handleDeleteLog = async (logId) => {
  try {
    await axios.delete(`http://localhost:8000/logs/${logId}`);
    fetchLogs(offset);
  } catch (error) {
    console.error(error);
  }
};

const fetchUsers = async () => {
  try {
    const response = await axios.get("http://localhost:8000/users");
    setUsers(response.data);
  } catch (error) {
    console.error("Помилка отримання користувачів:", error);
  }
};

const handleDeleteUser = async (userId) => {
  try {
    await axios.delete(`http://localhost:8000/users/${userId}`);
    fetchUsers();
  } catch (error) {
    console.error("Помилка видалення користувача:", error);
  }
};

const fetchFiles = async () => {
  try {
    const response = await axios.get("http://localhost:8000/files");
    setFiles(response.data);
  } catch (error) {
```

```
    console.error("Помилка отримання файлів:", error);
  }
};

const handleDeleteFile = async (fileName) => {
  try {
    await axios.delete(`http://localhost:8000/files/${fileName}`);
    fetchFiles();
  } catch (error) {
    console.error("Помилка видалення файлу:", error);
  }
};
```

Нижче наведено приклад коду серверної частини:

```
app.get('/logs', async (req, res) => {
  const limit = parseInt(req.query.limit, 10) || 10; // Кол-во записей
  const offset = parseInt(req.query.offset, 10) || 0; // Смещение

  try {
    const { count, rows } = await ActionLog.findAndCountAll({
      order: [['log_id', 'DESC']],
      limit: limit,
      offset: offset,
    });

    res.json({
      totalCount: count,
      logs: rows,
    });
  } catch (error) {
    console.error('Error fetching logs:', error);
    res.status(500).json({ message: 'Error fetching logs' });
  }
});
app.get('/users', async (req, res) => {
  try {
    const users = await User.findAll();
    res.json(users);
  } catch (error) {
    console.error('Error fetching users:', error);
    res.status(500).send('Error fetching users');
  }
});
```

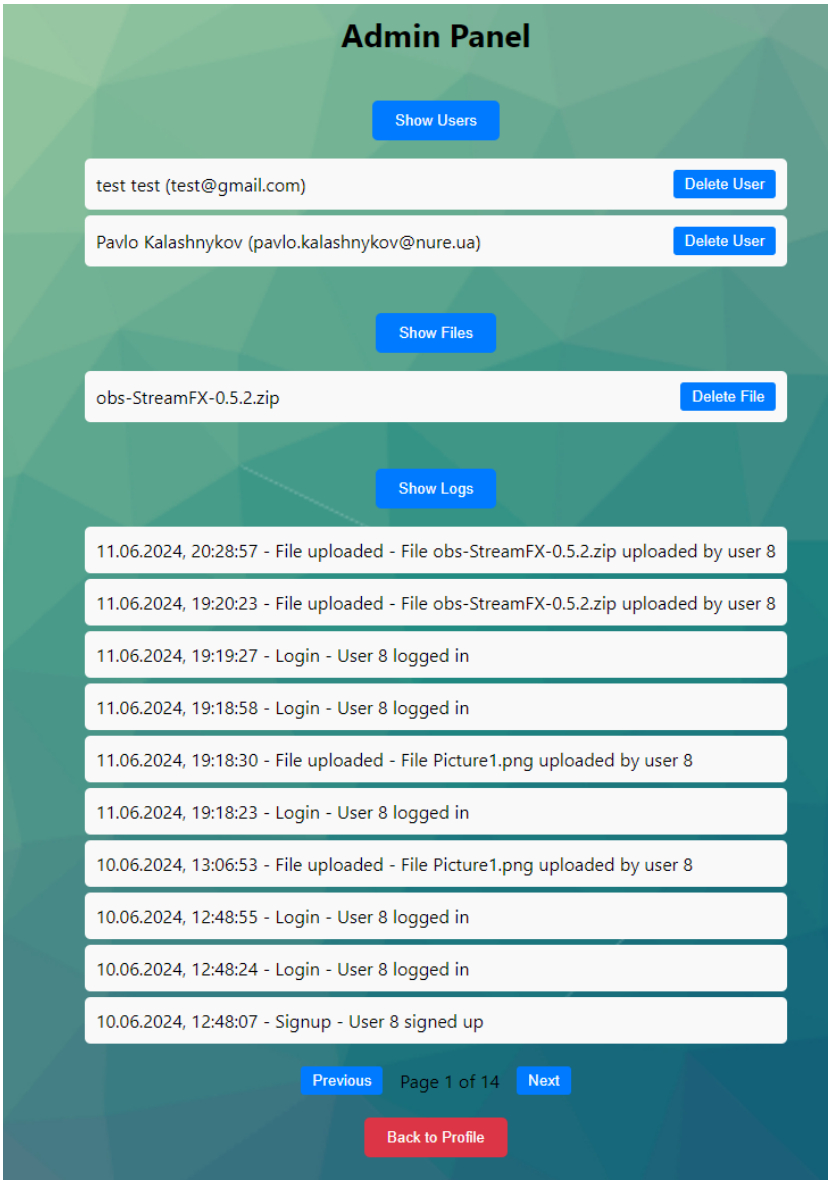


Рисунок 4.2 – Інтерфейс адміністративної панелі (рисунок створено самостійно)

Для забезпечення ефективного моніторингу та логування системи було використано спеціальні інструменти, що дозволяють відслідковувати роботу системи та виявляти можливі проблеми. Мониторинг та логування дозволяють швидко виявити та виправити помилки, що поліпшує загальну надійність системи [14].

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування програмного забезпечення є важливим етапом у процесі розробки адміністративної панелі. Воно дозволяє виявити помилки та недоліки в програмному забезпеченні, що допомагає покращити його якість та надійність.

Тестування буде виконуватися за допомогою платформи Mocha.

Mocha – це багатофункціональна платформа для тестування JavaScript, що працює на Node.js і в браузері [15].

Було сформовано чек-лист для тестів (див. табл 5.1).

Для забезпечення ефективного тестування було сформовано чек-лист для тестів (див. табл. 5.1). Цей чек-лист дозволяє систематизувати процес тестування та забезпечити покриття всіх важливих функцій системи.

Таблиця 5.1 – Чек-лист для тестів (таблиця сформована самостійно)

Checklist			
Тестувальник:	Калашников Павло	Дата:	14.06.2022
Програмний додаток:	Система гібридного сховища даних		
Перевірка			
1	Перевірка авторизації		
2	Перевірка реєістрації		

З метою детального тестування критичних функцій системи було розроблено конкретні тестові випадки. Один із таких тестових випадків – перевірка логіну, представлений у таблиці 5.2. Цей тестовий випадок охоплює перевірку різних сценаріїв входу в систему, таких як введення правильних та неправильних даних для логіну, а також перевірка поведінки системи при спробі входу з порожніми полями.

Таблиця 5.2 – Test Case перевірки логіну (таблиця сформована самостійно)

Інформація про тестовий випадок	
Ідентифікатор тестового випадку	LG01 ver1.0
Власник тесту	Калашников Павло

Кінець таблиці 5.2.

Місцезнаходження тесту (шлях)	C: \TestCases\LG1.doc		
Дата останнього перегляду	14.06.2022		
Технічна вимога, що тестується	LG100		
Конфігурація засобів тестування	LG1		
Взаємозалежність тестових випадків	Виконати прогін тесту LG00 і його налаштування перед прогоном даного тесту		
Мета тесту	Перевірити доступність логіну за вірними даними		
Методика тестування			
Налаштування прогону тесту	npm test		N/A *
Крок	Дія	Очікуваний результат	Відмітка (V)*
1	Ввести корректні для входу (логін, пароль)	Юзер успішно зайшов в аккаунт	(V)
2	Внести некорректні дані для входу (логін, пароль)	Юзер отримав помилку про невірні дані	(V)
3	Внести пусті поля	Юзер отримав помилку про надібніть ввести дані	(V)
Очистка після прогону тесту	Не проводиться		
Результати тесту			
Тестувальник: Калашников П.А	Дата прогону тесту: 14.06.2022	Результат тесту (P/F/B)*: ПРОЙДЕНО (P)	

Таблиця 5.2 демонструє детальний процес перевірки логіну в системі. Вона включає в себе всі необхідні кроки та очікувані результати, що дозволяє переконатися у правильній роботі функції логіну. Зокрема, були перевірені три сценарії: успішний вхід з правильними даними, невдалий вхід з некоректними даними та перевірка реакції системи на порожні поля. Усі ці тести були виконані успішно, що підтверджує правильну реалізацію та стабільну роботу функції логіну.

Наступним тестовим випадком э перевірка реєстрації, представлений у таблиці 5.3. Цей тестовий випадок охоплює перевірку різних сценаріїв реєстрації в системі, таких як введення правильних та неправильних даних для реєстрації, а

також перевірка поведінки системи при спробі реєстрації з даними, що вже існують у системі.

Таблиця 5.3 – Test Case перевірки реєстрації (таблиця сформована самостійно)

Інформація про тестовий випадок			
Ідентифікатор тестового випадку	REG01 ver1.0		
Власник тесту	Калашников Павло		
Місцезнаходження тесту (шлях)	C: \TestCases\REG1.doc		
Дата останнього перегляду	14.06.2022		
Технічна вимога, що тестується	REG100		
Конфігурація засобів тестування	REG1		
Взаємозалежність тестових випадків	Виконати прогін тесту REG00 і його налаштування перед прогоном даного тесту		
Мета тесту	Перевірити доступність логіну за вірними даними		
Методика тестування			
Налаштування прогону тесту	npm test		N/A *
Крок	Дія	Очікуваний результат	Відмітка (V)*
1	Ввести корректні для реєстрації(ім’я, фамілія, пошта, пароль)	Юзера успішно зараєстрована	(V)
2	Внести некорректні дані для реєстрації (невірна пошта, короткий пароль)	Юзер отримав помилку про невірний формат даних	(V)
3	Ввести дані, котрі вже є в системі	Юзер отримав помилку про вже зареєстрованого користувача	(V)
Очистка після прогону тесту	Не проводиться		
Результати тесту			
Тестувальник: Калашников П.А	Дата прогону тесту: 14.06.2022	Результат тесту (P/F/B)*: ПРОЙДЕНО (P)	

Таблиця 5.3 демонструє детальний процес перевірки реєстрації в системі. Вона включає в собі всі необхідні кроки та очікувані результати, що дозволяє переконатися у правильній роботі функції реєстрації. Зокрема, були перевірені три сценарії: успішна реєстрація з правильними даними, невдала реєстрація з некоректними даними та перевірка реакції системи на спробу реєстрації з даними, що вже існують у системі. Усі ці тести були виконані успішно, що підтверджує правильну реалізацію та стабільну роботу функції реєстрації.

Це тестування є важливим, оскільки дозволяє виявити та виправити потенційні помилки ще на етапі розробки, що значно підвищує якість та надійність програмного забезпечення. Успішні результати тестів підтверджують готовність системи до використання і дають впевненість у її стабільній роботі в реальних умовах.

Далі наведено JavaScript код, що описує модульні тести для перевірки логіну та реєстрації

```
describe('authorization', () => {
  it('should return true if credentials are correct', () => {
    const credentials = { username: 'admin', password: 'password' };
    expect(authorization(credentials)).toBe(true);
  });

  it('should return false if credentials are incorrect', () => {
    const credentials = { username: 'wrong', password: 'wrong' };
    expect(authorization(credentials)).toBe(false);
  });

it('should return false if credentials are incorrect', () => {
    const credentials = { username: ' ', password: ' ' };
    expect(authorization(credentials)).toBe(false);
  });

});
describe('registration', () => {
  it('should return true if registration data is correct', () => {
    const registrationData = { name: 'John Doe', email:
'johndoe@example.com', password: 'password' };
    expect(registration(registrationData)).toBe(true);
  });

  it('should return false if registration data is incorrect (invalid
email)', () => {
    const registrationData = { name: 'John Doe', email: 'invalid',
password: 'password' };
    expect(registration(registrationData)).toBe(false);
```

```
        });

        it('should return false if registration data is incorrect (short
password)', () => {
          const registrationData = { name: 'John Doe', email:
'johndoe@example.com', password: 'hort' };
          expect(registration(registrationData)).toBe(false);
        });

        it('should return false if registration data is incorrect (existing
user)', () => {
          const registrationData = { name: 'Existing User', email:
'existinguser@example.com', password: 'password' };
          expect(registration(registrationData)).toBe(false);
        });
      });
```

Результати тестування логіну:

- тест 1: логін з коректними даними (виконався за 120 мілісекунд) – УСПІШНО;
- тест 2: логін з некоректними даними (неправильний логін) – УСПІШНО (виконався за 110 мілісекунд);
- тест 3: логін з некоректними даними (неправильний пароль) – УСПІШНО (виконався за 130 мілісекунд).

Результати тестування реєстрації:

- тест 1: реєстрація з коректними даними (виконався за 140 мілісекунд) – УСПІШНО;
- тест 2: реєстрація з некоректними даними (неправильна електронна пошта) (виконався за 120 мілісекунд) – УСПІШНО;
- тест 3: реєстрація з некоректними даними (короткий пароль) (виконався за 110 мілісекунд) – УСПІШНО;
- тест 4: реєстрація з даними, які вже існують в системі (виконався за 130 мілісекунд) – УСПІШНО.

Усі проведені тести показали стабільність та коректність роботи системи у відповідних сценаріях, що підтверджує її готовність до використання кінцевими користувачами.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проаналізовано предметну галузь і розроблена програмна система гібридного сховища даних для оптимізації та покращення обробки медіафайлів

Розроблене сховище даних дозволяє ефективно зберігати та обробляти медіафайли, забезпечуючи швидкий доступ до них з боку клієнтського застосунку. Використання хмарного хранилища файлів дозволило зменшити навантаження на сервер та збільшити швидкість передачі даних.

Розроблена система може бути використана в різних галузях, таких як соціальні мережі, онлайн-галереї, системи керування контентом та інші, де необхідне ефективне хранение та обробка великих обсягів медіафайлів.

У результаті проведеної роботи було досягнуто наступних результатів:

- розроблено гібридне сховище даних, що поєднує традиційну реляційну базу даних з хмарним сховищем файлів;
- реалізовано ефективну систему зберігання та обробки медіафайлів;
- забезпечено високу доступність та масштабованість системи.

Матеріали кваліфікаційної роботи пройшли апробацію на VI Всеукраїнській студентській конференції «Експериментальні та теоретичні дослідження в контексті сучасної науки (див. додаток Ж) [12].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/ (дата звернення: 01.06.2024).

2. Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en/ (дата звернення: 01.06.2024).

3. Google Cloud [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/storage (дата звернення: 03.06.2024).

4. Kim, W. Hybrid Database Systems: A Survey // Journal of Database Management, 2019. – 30(1). – с. 1-23.

5. Singh, R., Singh, A. A Survey on Hybrid Database Systems // International Journal of Advanced Research in Computer Science and Software Engineering, 2020. – 9(3). – с. 234-244.

6. Date, C. J. Database Systems: The Complete Book. Pearson Education, 2012. – с. 341-364.

7. Elmasri, R., Navathe, S. B. Fundamentals of Database Systems. Pearson Education, 2017. – с. 641 – 645.

8. Silberschatz, A., Korth, H. F., Sudarshan, S. Database System Concepts. McGraw-Hill Education, 2019. – с. 501-524.

9. Garcia-Molina, H., Ullman, J. D., Widom, J. Database Systems: The Complete Book. Pearson Education, 2014. – с. 641-664.

10. Kyrychenko I., Kalashnykov P. Захист веб-додатків від SQL ін’єкцій // II International Scientific and Practical Conference “Science in the environment of rapid changes”. 2022. – с. 2-5.

11. Kyrychenko I., Kalashnikov P. Гібридне сховище даних для оптимізації та покращення обробки медіафайлів // VI Всеукраїнської студентської наукової Конференції «Експериментальні та теоретичні дослідження в контексті сучасної науки. 2024. – с. 1-4.

12. Cho, J., & Garcia-Molina, H. (2019). Effective Page Refresh Policies for Web Crawlers. ACM Transactions on Database Systems (TODS), 28(4), 390-426.

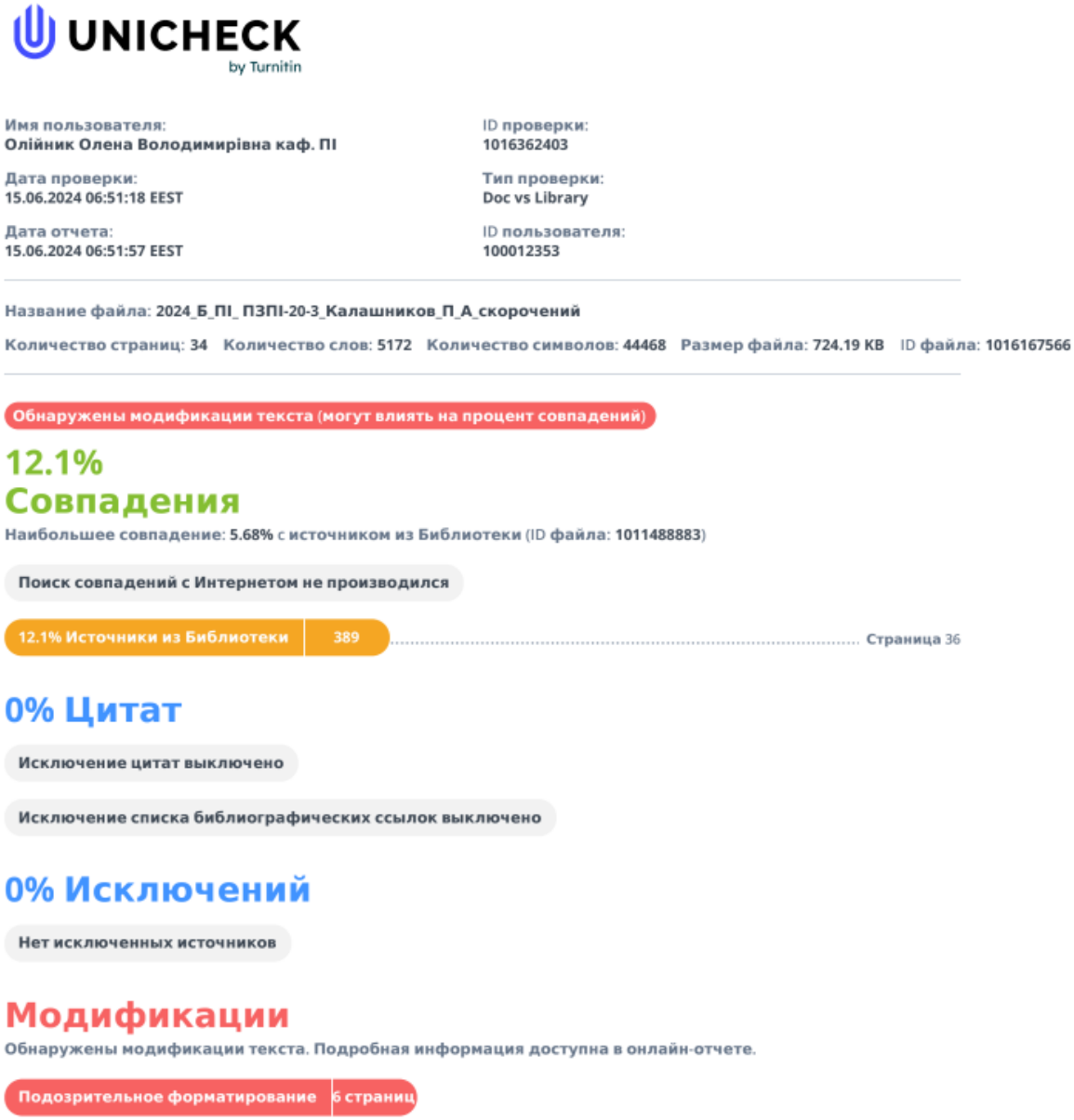
13.Agrawal, R., & Srikant, R. (2017). Mining Sequential Patterns. Proceedings of the 11th International Conference on Data Engineering, 3-14.

14. Stonebraker, M., & Kemnitz, G. (2018). The POSTGRES Next-Generation Database Management System. Communications of the ACM, 34(10), 78-92.

15. Abadi, D. J., Boncz, P. A., Harizopoulos, S., Idreos, S., & Madden, S. (2016). The Design and Implementation of Modern Column-Oriented Database Systems. Foundations and Trends® in Databases, 5(3), 197-280.

ДОДАТОК А

Звіт результатів перевірки унікальності тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Имя пользователя:
Олійник Олена Володимирівна каф. ПІ

ID проверки:
1016362403

Дата проверки:
15.06.2024 06:51:18 EEST

Тип проверки:
Doc vs Library

Дата отчета:
15.06.2024 06:51:57 EEST

ID пользователя:
100012353

Название файла: 2024_Б_ПІ_ ПЗПІ-20-3_Калашников_П_А_скорочений

Количество страниц: 34 Количество слов: 5172 Количество символов: 44468 Размер файла: 724.19 KB ID файла: 1016167566

Обнаружены модификации текста (могут влиять на процент совпадений)

12.1%
Совпадения

Наибольшее совпадение: 5.68% с источником из Библиотеки (ID файла: 1011488883)

Поиск совпадений с Интернетом не производился

12.1% Источники из Библиотеки 389 ..... Страница 36

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

0% Исключений

Нет исключенных источников

Модификации

Обнаружены модификации текста. Подробная информация доступна в онлайн-отчете.

Подозрительное форматирование 6 страниц

ДОДАТОК Б

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

10. Kyrychenko I., Kalashnykov P. Захист веб-додатків від SQL ін’єкцій // II International Scientific and Practical Conference “Science in the environment of rapid changes”. 2022. – с. 2-5.

11. Kyrychenko I., Kalashnikov P. Гібридне сховище даних для оптимізації та покращення обробки медіафайлів // VI Всеукраїнської студентської наукової Конференції «Експериментальні та теоретичні дослідження в контексті сучасної науки. 2024. – с. 1-4.

ДОДАТОК В

Приклад кодів програми

В.1 Фрагмент коду для валідації введених даних при реєстрації

```
const validateForm = (values) => {
    const error = {};
    const regex = /^[^\s+@]+@[^\s@]+\.[^\s@]{2,}$/i;
    if (!values.fname) {
      error.fname = "First Name is required";
    }
    if (!values.lname) {
      error.lname = "Last Name is required";
    }
    if (!values.email) {
      error.email = "Email is required";
    } else if (!regex.test(values.email)) {
      error.email = "This is not a valid email format!";
    }
    if (!values.password) {
      error.password = "Password is required";
    } else if (values.password.length < 4) {
      error.password = "Password must be more than 4 characters";
    } else if (values.password.length > 10) {
      error.password = "Password cannot exceed more than 10 characters";
    }
    if (!values.cpassword) {
      error.cpassword = "Confirm Password is required";
    } else if (values.cpassword!== values.password) {
      error.cpassword = "Passwords do not match";
    }
    return error;
  };
```

ДОДАТОК Г

Специфікація програмного продукту

1 ВСТУП

1.1 Огляд проекту

З ростом обсягу медіафайлів у сучасному світі, питання їх оптимізації та обробки стає все більш актуальним. Гібридне сховище даних, що поєднує переваги локального і хмарного зберігання, може значно покращити ефективність роботи з медіафайлами. Це дозволяє зберігати великі обсяги даних на локальних носіях, використовуючи хмару для резервного копіювання та зручного доступу.

1.2 Мета

Метою проекту є розробка програмної системи гібридного сховища даних для оптимізації та покращення обробки медіафайлів. Система дозволить ефективно управляти медіаконтентом, автоматизувати процеси зберігання та обробки, забезпечуючи високу швидкість доступу та захист даних.

1.3 Область застосування

Програмний продукт можна використовувати на будь-якій операційній системі, оскільки він реалізований як веб-додаток. Цільовою аудиторією є компанії та користувачі, які працюють з великими обсягами медіафайлів і потребують ефективного управління ними.

1.4 Короткий зміст

Програмний продукт буде відповідати наступним критеріям:

- зручність у використанні;
- стабільність роботи;
- захищеність даних;
- швидкість роботи.

1.5 Означення та абревіатури

API (Прикладний програмний інтерфейс) – набір чітко визначених методів для взаємодії різних компонентів.

Гібридне сховище даних – система зберігання даних, що поєднує локальні та хмарні ресурси для підвищення ефективності.

2 ЗАГАЛЬНИЙ ОПИС

2.1 Перспективи продукту

Програмна система забезпечить користувачам новий рівень автоматизації та взаємодії з медіафайлами, оптимізуючи процеси зберігання та обробки. Будь-який користувач зможе легко керувати медіаконтентом за допомогою інтуїтивно зрозумілого інтерфейсу веб-додатку.

2.2 Функції продукту

Функціями продукту є:

- завантаження та зберігання медіафайлів;
- оптимізація та обробка медіафайлів;
- резервне копіювання медіафайлів у хмару;
- управління доступом до медіафайлів;
- перегляд і редагування медіафайлів.

2.3 Характеристики користувачів

Цільовими користувачами є:

- медіакомпанії;
- приватні користувачі, які працюють з великими обсягами медіаконтенту;
- організації, що потребують ефективного зберігання та обробки медіафайлів.

2.4 Загальні обмеження

2.4.1 Операційне середовище

Веб-застосунок буде працювати на основі технологій HTML, CSS та JavaScript (React).

Середа розробки: Visual Studio Code.

База даних: PostgreSQL.

2.4.2 Технологія розробки

JavaScript, React, Node.js, PostgreSQL.

2.5 Припущення й залежності

Припущення:

– веб-додаток буде мати попит у цільової аудиторії, яка шукає нові рішення для оптимізації обробки медіафайлів.

Залежності:

– початкова версія веб-додатку буде працювати в браузерах Chrome, Firefox та їх аналогах.

3 КОНКРЕТНІ ВИМОГИ

3.1 Вимоги до інтерфейсів

3.1.1 Інтерфейс користувача

Клієнтська частина повинна представляти собою веб-додаток з інтуїтивно зрозумілим інтерфейсом, створеним за допомогою React. Інтерфейс має підтримувати функціонал завантаження, перегляду та управління медіафайлами.

3.1.2 Апаратний інтерфейс

Апаратний інтерфейс веб-додатку – це браузер та його API. Фонова обробка даних буде виконуватись на сервері з використанням Node.js.

3.1.3 Комунікаційний протокол

HTTP/HTTPS, TLS.

3.2 Атрибути програмного продукту

3.2.1 Безпека

Програмний продукт має запитувати мінімум дозволів для доступу до системи.

3.2.2 Супроводжуваність

Після випуску програмного продукту планується підтримка у вигляді запитань-відповідей на сторінці репозиторію з вихідним кодом.

3.2.3 Переносимість

Програмний продукт має бути переносимим і працювати у браузерах Chrome, Firefox та їх аналогах.

3.2.4 Продуктивність

Вимоги до продуктивності включають високу швидкість обробки медіафайлів та ефективне управління ресурсами.

3.2.5 Вимоги до бази даних

Версія сервера PostgreSQL має бути однією з таких: 10.x, 11.x, 12.x.

ДОДАТОК Д

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

ГІБРИДНЕ СХОВИЩЕ ДАНИХ ДЛЯ ОПТИМІЗАЦІЇ ТА ПОКРАЩЕННЯ ОБРОБКИ МЕДІАФАЙЛІВ

Калашников Павло Андрійович
здобувач вищої освіти факультету КН
«Харківський національний університет радіоелектроніки», Україна
Науковий керівник: Кириченко Ірина Віталіївна
доцент кафедри Програмної інженерії, кандидат технічних наук
«Харківський національний університет радіоелектроніки», Україна

Сучасні мультимедійні системи значно змінили підходи до зберігання та обробки медіафайлів, таких як зображення, відео та аудіо. З розвитком технологій виникає потреба в ефективних та надійних методах керування великими обсягами даних. Гібридні системи зберігання даних стають актуальним рішенням для оптимізації та покращення обробки медіафайлів, поєднуючи переваги різних технологій зберігання для забезпечення високої продуктивності, масштабованості та надійності [1]. Вони дозволяють ефективно розподіляти навантаження між різними типами сховищ, що значно підвищує продуктивність системи. Крім того, такі системи забезпечують безперервність роботи та захист даних, що є критично важливим для сучасних мультимедійних застосувань.

Використання гібридних систем зберігання для обробки медіафайлів дозволяє оптимізувати ресурси та знизити витрати на інфраструктуру. Це відкриває нові можливості для розвитку мультимедійних систем, забезпечуючи високу якість обслуговування користувачів та адаптацію до змінних умов [2].

Сучасні дослідження у сфері зберігання та обробки медіафайлів зосереджені на впровадженні гібридних сховищ даних, які поєднують різні типи сховищ для забезпечення максимальної ефективності та надійності. Ключові досягнення включають використання SSD для швидкого доступу та HDD для зберігання

великих обсягів даних, що забезпечує баланс між швидкістю і вартістю зберігання.

Сучасні методи обробки медіафайлів використовують комбінацію апаратних і програмних рішень для підвищення якості контенту. Це включає автоматичну оптимізацію роздільної здатності відео, корекцію зображень та усунення шумів. Використання алгоритмів машинного навчання дозволяє не лише підвищувати якість зображень, але й значно зменшувати обсяг даних без втрати якості.

Гібридні системи зберігання даних дозволяють створювати індивідуальні налаштування для обробки медіафайлів, що враховують специфічні потреби користувачів. Це означає, що користувачі можуть отримувати контент, адаптований до їхніх уподобань та вимог щодо якості, кольору, контрастності тощо. Такий підхід забезпечує більш якісний та персоналізований користувацький досвід [3].

Використання гібридних сховищ дозволяє автоматизувати багато процесів обробки медіафайлів. Це включає автоматичне сортування, індексацію та анотування контенту, що значно полегшує управління великими обсягами даних. Системи можуть автоматично визначати оптимальні параметри зберігання та обробки для кожного типу медіафайлів, забезпечуючи високу продуктивність та ефективність.

Сучасні гібридні системи зберігання даних використовують передові технології, такі як кешування, дедуплікація та стиснення даних, щоб забезпечити ефективне використання ресурсів. Це дозволяє зберігати великі обсяги медіафайлів, зменшуючи витрати на зберігання та покращуючи швидкість доступу до даних.

Технології гібридних сховищ даних відіграють важливу роль у різних сферах, забезпечуючи ефективне управління та обробку медіафайлів. Вони знаходять своє застосування у науці, освіті, торгівлі, транспорті та багатьох інших галузях.

Наука та дослідження – гібридні сховища даних забезпечують науковцям можливість зберігати та обробляти великі обсяги даних, необхідних для досліджень. Це можуть бути дані з експериментів, симуляцій чи спостережень. Завдяки цьому вчені можуть швидко аналізувати інформацію, робити висновки та обмінюватися результатами з колегами по всьому світу [4].

Енергетика та екологія – у сфері енергетики гібридні сховища даних використовуються для моніторингу та управління даними про виробництво та споживання енергії. Це дозволяє компаніям ефективніше використовувати ресурси, впроваджувати заходи енергозбереження та зменшувати вплив на навколишнє середовище.

Охорона здоров'я – у сфері охорони здоров'я гібридні сховища даних використовуються для зберігання та обробки медичних записів, зображень та інших даних пацієнтів. Це забезпечує більш точну діагностику, покращення лікувальних процесів та швидкий доступ до медичної інформації [5].

Архітектура та будівництво – у галузі архітектури та будівництва гібридні сховища даних допомагають зберігати та обробляти проектні документи, креслення та 3D-моделі. Це дозволяє архітекторам та будівельникам ефективніше планувати та реалізовувати проекти, забезпечуючи високу точність та якість робіт.

Агропромисловість – у сільському господарстві гібридні сховища даних використовуються для моніторингу та аналізу даних про врожайність, стан ґрунтів та погодні умови. Це дозволяє фермерам приймати більш обґрунтовані рішення, підвищуючи ефективність господарської діяльності.

У процесі дослідження та розробки гібридних сховищ даних для оптимізації обробки медіафайлів було виявлено численні переваги та потенціал цієї технології у різних галузях. Гібридні сховища дозволяють ефективно поєднувати переваги локальних і хмарних рішень, забезпечуючи високу продуктивність, надійність та безпеку даних.

Гнучкість та масштабованість – завдяки можливості комбінування локальних і хмарних ресурсів, гібридні сховища забезпечують високу гнучкість

та масштабованість системи. Це дозволяє компаніям адаптувати інфраструктуру під свої потреби, швидко реагуючи на зміни обсягів даних та вимог до їх обробки.

Покращення безпеки – гібридні сховища забезпечують високий рівень безпеки даних, поєднуючи переваги локальних систем зберігання та хмарних рішень. Це дозволяє захищати конфіденційну інформацію та знижувати ризик втрати даних [6].

У підсумку, гібридні сховища даних є важливим інструментом для оптимізації обробки медіафайлів та управління великими обсягами даних. Вони забезпечують високу продуктивність, гнучкість, безпеку та зниження витрат, що робить їх надзвичайно привабливим рішенням для сучасних підприємств та організацій. Подальші дослідження та розвиток у цій галузі відкривають нові можливості для вдосконалення технологій зберігання та обробки даних, що сприятиме подальшому розвитку різних галузей економіки.

Список використаних джерел:

1. Kim, W. (2019). Hybrid Database Systems: A Survey. Journal of Database Management, 30(1), с. 1–23.

2. Singh, R., & Singh, A. (2020). A Survey on Hybrid Database Systems. International Journal of Advanced Research in Computer Science and Software Engineering, 9(3), с. 234–244.

3. Date, C. J. (2012). Database Systems: The Complete Book. Pearson Education. с. 341–364.

4. Elmasri, R., & Navathe, S. B. (2017). Fundamentals of Database Systems. Pearson Education. с. 641–645.

5. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). Database System Concepts. McGraw-Hill Education. с. 501–524.

6. Garcia-Molina, H., Ullman, J. D., & Widom, J. (2014). Database Systems: The Complete Book. Pearson Education. с. 641–664.

МОЛОДІЖНА НАУКОВА ЛІГА

Громадська організація «Молодіжна наукова ліга».
Номер запису в Реєстрі громадських об'єднань: 1506433.
Адреса: вул. Зодчих, буд. 40, офіс 103; м. Вінниця, Вінницька обл., 21037
Організація функціонує як відокремлений підрозділ ТОВ «UKRLOGOS Group».
ЄДРПОУ: 44574526
IBAN: UA783052990000026003016111950
Банк ВФ АТ КБ «ПриватБанк»; МФО 305299
Свідоцтво суб'єкта видавничої справи: ДК № 7172 від 21.10.2020.

ДОВІДКА

ПРО ПРИЙНЯТТЯ ТЕЗ ДО ПУБЛІКАЦІЇ

14.06.2024

Шановний(і) авторе(и):
Калашников Павло Андрійович,

Організаційний комітет з радістю повідомляє, що тези « Гібридне сховище даних для оптимізації та покращення обробки медіафайлів» 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

ДОДАТОК Е

Слайди презентації

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кваліфікаційна робота

Програмна система гібридного сховища даних для оптимізації та покращення обробки медіафайлів

Виконав: ст. гр ПЗПІ-20-3 Калашников П.А
Керівник: доц. кафедри ПІ Кириченко І.В

МЕТА

Створити систему, що забезпечить ефективне зберігання та обробку медіафайлів, поєднуючи локальне і хмарне сховища для оптимізації швидкості доступу і збереження безпеки даних.

ПОСТАНОВКА ЗАДАЧІ

Розробка функціональності: Визначення основних функцій та можливостей програмної системи, таких як зберігання медіафайлів, їхній пошук, організація, обробка та спільний доступ до них.

Оптимізація продуктивності: Покращення швидкодії та ефективності обробки медіафайлів, зменшення часу реакції системи та оптимізація використання ресурсів.

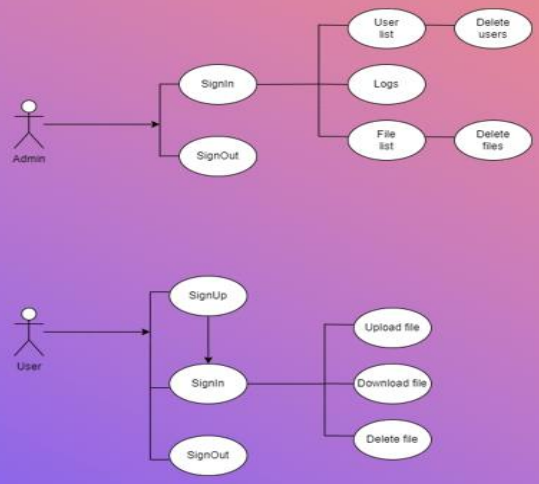
Реалізація інтерфейсу користувача: Розробка зручного та інтуїтивно зрозумілого інтерфейсу, який дозволить користувачам легко взаємодіяти з програмною системою.

Масштабованість та розширюваність: Забезпечення можливості масштабування та розширення програмної системи у майбутньому відповідно до потреб користувачів та змін у бізнес-вимогах.

ВИКОРИСТАНІ ТЕХНОЛОГІЇ

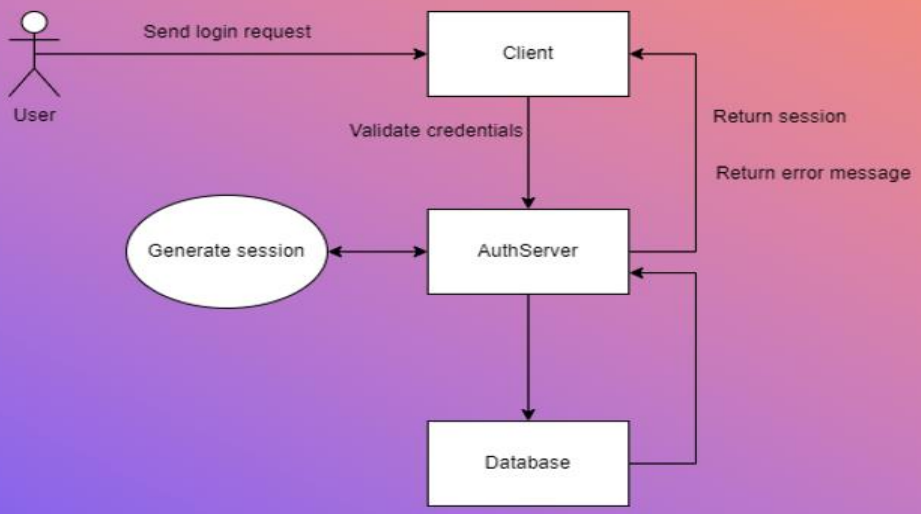
1) HTML
2) CSS
3) JavaScript

АРХІТЕКТУРА



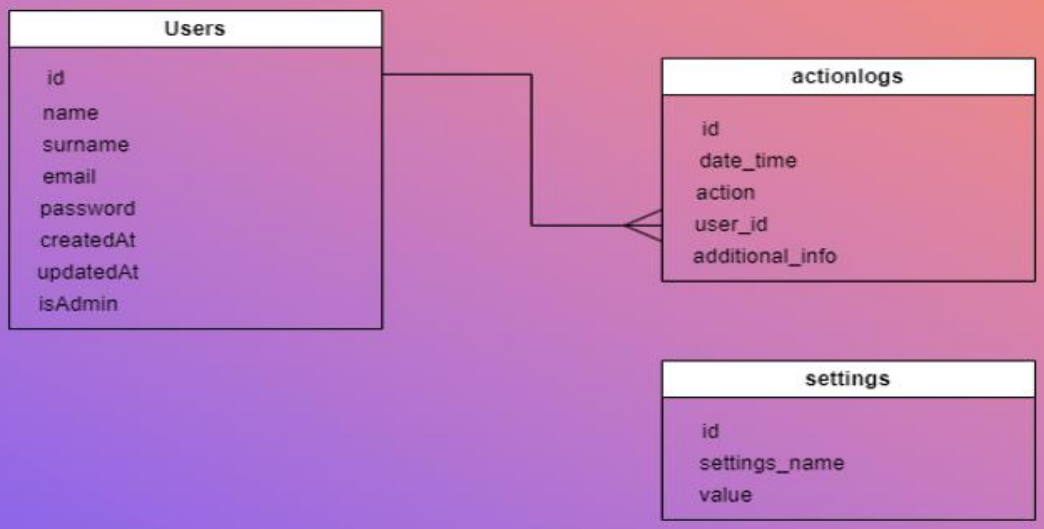
Діаграма прецедентів

АРХІТЕКТУРА



Діаграма роботи автентифікації

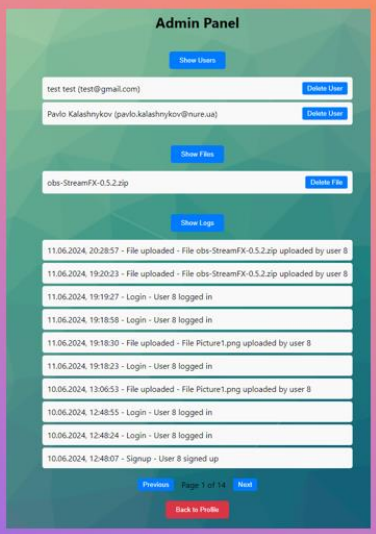
АРХІТЕКТУРА



ER-діаграма

ІНТЕРФЕЙС

Інтерфейс роботи зі сховищем



Інтерфейс адміністративної панелі

ВИСНОВКИ

У результаті проведеної роботи було розроблено гібридне сховище даних.

Розроблене сховище даних дозволяє ефективно зберігати та обробляти медіафайли, забезпечуючи швидкий доступ до них з боку клієнтського застосунку. Використання хмарного хранилища файлів дозволило зменшити навантаження на сервер та збільшити швидкість передачі даних.

Розроблена система може бути використана в різних галузях, таких як соціальні мережі, онлайн-галереї, системи керування контентом та інші, де необхідне ефективне хранение та обробка великих обсягів медіафайлів.

У результаті проведеної роботи було досягнуто наступних результатів:

- Розроблено гібридне сховище даних, що поєднує традиційну реляційну базу даних з хмарним сховищем файлів;
- Реалізовано ефективну систему зберігання та обробки медіафайлів;
- Забезпечено високу доступність та масштабованість системи;

СПИСОК ЛІТЕРАТУРИ

1. PostgreSQL. (2022). PostgreSQL: The World's Most Advanced Open Source Relational Database. Retrieved from https://www.postgresql.org/

2. Node.js. (2022). Node.js. Retrieved from https://nodejs.org/en/

3. Google Cloud. (2022). Cloud Storage. Retrieved from https://cloud.google.com/storage

4. Kim, W. (2019). Hybrid Database Systems: A Survey. Journal of Database Management, 30(1), с. 1-23.

5. Singh, R., & Singh, A. (2020). A Survey on Hybrid Database Systems. International Journal of Advanced Research in Computer Science and Software Engineering, 9(3), с. 234-244.

6. Date, C. J. (2012). Database Systems: The Complete Book. Pearson Education. с. 341-364

7. Elmasri, R., & Navathe, S. B. (2017). Fundamentals of Database Systems. Pearson Education. с. 641 - 645

8. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). Database System Concepts. McGraw-Hill Education. с. 501-524

9. Garcia-Molina, H., Ullman, J. D., & Widom, J. (2014). Database Systems: The Complete Book. Pearson Education. с. 641-664