Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Оптимізація пошуку в файлових сховищах засобами NLP моделей

(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-21-2

Гліб Коваль
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки

(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Штучний інтелект

(повна назва освітньої програми)
Керівник ст. викл. Олена Гриньова
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Ковалю Глібу Костянтиновичу
(прізвище, ім'я, по батькові)

1. Тема роботи Оптимізація пошуку в файлових сховищах засобами NLP моделей

затверджена наказом університету від 19 травня 20 25 р. № 378Ст
2. Термін подання студентом роботи до екзаменаційної комісії 25 червня 20 25 р.
3. Вихідні дані до роботи Матеріали з використання NLP для аналізу та обробки метаданих текстових документів у файлових системах. Python, API, HTML, CSS, JavaScript, FastAPI, Visual Studio Code

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі та постановка задачі
2) Проєктування та розробка хмарного середовища для зберігання файлів
3) Програмна реалізація та тестування вебсайту

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	19.05.2025	виконано
2	Аналіз предметної галузі	20.05 – 22.05	виконано
3	Постановка задачі	23.05 – 25.05	виконано
4	Обґрунтування вибору використаних технологій	26.05 – 29.05	виконано
5	Програмна реалізація	30.05 – 06.06	виконано
6	Оформлення пояснювальної записки	04.06 – 15.06	виконано
7	Підготовка до захисту	17.06 – 20.06	виконано
8	Захист перед ЕК	25.06.2025	виконано

Дата видачі завдання 19 травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Олена Гриньова
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 67 с., 38 рис., 5 табл., 4 дод., 21 джерело.

АВТОРИЗАЦІЯ, ЗБЕРЕЖЕННЯ ФАЙЛІВ, СХОВИЩЕ, ХМАРНЕ СЕРЕДОВИЩЕ, NLP.

Об’єкт дослідження – надання можливості штучному інтелекту шукати файли по змісту.

Предмет дослідження – застосування інноваційних технологій для розробки хмарного середовища для збереження файлів.

Мета роботи – створення хмарного середовища для зберігання файлів з елементами авторизації, пошуку файлів, улюблених файлів та можливості повернути видалений файл до 24 годин після його видалення і використання ШІ для пошуку файлів за змістом.

Методи дослідження – розробка базується на використанні сучасних технологій. Реалізація здійснюється за допомогою таких технологій як FastAPI, Next.js, Shadcn. Для підвищення продуктивності система розділена на 3 окремих мікросервіси, які відповідають тільки за певні задачі.

Одержані результати: розроблений хмарний середовище для збереження файлів виконує наступні завдання: автентифікація, авторизація, збереження файлів, пошук файлів, видалення та можливість відновлення видаленого файлу до 24 годин, можливість добавити файл до улюблених, можливість переглянути файл та можливість скачати файл.

ABSTRACT

Bachelor’s thesis contains: 67 pp., 38 fig., 5 tabl., 4 ann., 21 references.

AUTHORISATION, CLOUD ENVIRONMENT, FILE STORAGE, NLP, REPOSITORY.

The object of research is to enable AI (artificial intelligence) to search files by content.

The subject of research is the use of innovative technologies to develop a cloud environment for storing files.

Purpose – to create a cloud environment for storing files with elements of authorisation, file search, favourite files and the ability to return a deleted file up to 24 hours after it is deleted and use AI to search for files by content.

Research methods – the development is based on the use of modern technologies. The implementation is carried out with the help of such technologies as FastAPI, Next.js, Shadcn. To improve performance, the system is divided into 3 separate microservices that are responsible only for certain tasks.

Results obtained: the developed cloud environment for file storage performs the following tasks: authentication, authorisation, file saving, file search, file deletion and the ability to recover a deleted file up to 24 hours, the ability to add a file to favourites, the ability to view a file and the ability to download a file.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 7
Вступ 8
1 Аналіз предметної галузі та постановка задачі 10
1.1 Засоби для розробки хмарного сховища файлів 10
1.2 Огляд та аналіз існуючих програмних рішень 11
1.3 Аналіз існуючих технологій розробки 14
1.4 Вимоги до системи 18
2 Проєктування та розробка хмарного середовища для зберігання файлів 20
2.1 Специфікація вимог до системи 20
2.2 Розробка UML-діаграм 21
2.3 Вибір технологій розробки хмарного середовища 24
2.3.1 Вибір технологій розробки графічного інтерфейсу 24
2.3.2 Вибір технологій розробки backend частини хмарного сервісу для збереження файлів 25
2.4 Розробка графічного інтерфейсу вебсервісу 27
2.5 Проєктування бази даних 29
2.6 Семантичний пошук за допомогою NLP-моделей 35
3 Програмна реалізація та тестування вебдодатку 39
3.1 Опис модулів та компонентів серверної частини 39
3.2 Опис модулів та компонентів клієнтської частини 50
3.3 Інструкція до розробленого вебсайту 55
Висновки 57
Перелік джерел посилання 58
Додаток А Код Для Завантаження Файлів 60
Додаток Б Код компоненти Dashboard 63
Додаток В Код провайдера для автентифікації 65
Додаток Г Відомість кваліфікаційної роботи 67

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

СКБД – система керування базами даних;

ШІ – штучний інтелект;

AI – Artificial Intelligence – штучний інтелект;

CSS – Cascading Style Sheets – каскадні таблиці стилів;

HTML – HyperText Markup Language – мова розмітки гіпертексту;

HTTP – Hypertext Transfer Protocol – протокол передавання гіпертексту;

JSON – JavaScript Object Notation – запис об’єктів JavaScript;

NLP – Natural Language Processing – обробка природної мови;

RAG – Retrieval-Augmented Generation – генерація з доповненням через пошук;

XML – eXtensible Markup Language – розширювана мова розмітки.

ВСТУП

У сучасному цифровому світі обсяги інформації зростають з вражаючою швидкістю. Щодня створюються терабайти даних у вигляді текстових документів, електронного листування, звітів, технічної документації, юридичних файлів та інших типів контенту. Значна частина цієї інформації зберігається у структурованих файлових сховищах, які використовуються як у корпоративному середовищі, так і багатьох інших сферах. У зв’язку з цим пошук релевантної інформації стає все більш актуальним викликом. Стандартні методи пошуку, побудовані на принципах індексації за ключовими словами або простих синтаксичних відповідностей, дедалі частіше виявляються недостатньо ефективними, особливо коли мова йде про великі обсяги неструктурованих даних або складні запити з контекстуальним підтекстом.

Проблема полягає не тільки в зростанні обсягів даних, але й у складності їхньої природи. Більшість інформації у файлових сховищах представлена в неструктурованому форматі, що робить її недоступною для прямого запиту. Наприклад, пошук конкретного документа, що містить певну концепцію, але не прямі ключові слова, може бути вкрай складним.

Існуючі системи пошуку у файлових сховищах часто покладаються на індексування імен файлів, розширень, дат створення і тому інше. Хоча ці підходи є корисними для базової навігації, вони не дозволяють здійснювати пошук за змістом між документами. Це створює «інформаційний шум» та знижує ефективність роботи.

Останні роки ми стали свідками стрімкого розвитку технологій обробки природної мови NLP, які дозволяють глибше аналізувати зміст текстів, розуміти семантичні зв’язки між словами, фразами і цілими документами [1]. Інтеграція методів NLP у інформаційно-пошукові системи відкриває нові можливості для оптимізації пошуку у файлових сховищах. На відміну від класичних алгоритмів, NLP-моделі здатні враховувати

контекст, ідентифікувати сутності, аналізувати наміри користувача та здійснювати семантичний пошук, що підвищує як точність, так і повноту результатів. Такі моделі також можуть покращувати запити, доповнюючи їх синонімами або уточненнями, і забезпечувати релевантні відповіді навіть тоді, коли формулювання запиту значно відрізняється від формулювань у документах.

У цьому контексті особливе місце займають системи, побудовані за архітектурою Retrieval-Augmented Generation (RAG), які поєднують традиційний пошук з генеративними можливостями глибинних нейронних мереж. Такі системи використовують векторні представлення тексту, що дозволяють зіставляти документи на рівні змісту, а не лише за ключовими словами. На етапі пошуку запит користувача також перетворюється у вектор, і за допомогою спеціалізованих векторних сховищ здійснюється пошук найбільш релевантних відповідей.

Попри всі переваги, практичне впровадження NLP-моделей у пошукові системи має низку викликів. Одним із головних є необхідність балансувати між продуктивністю та точністю. Глибокі трансформерні моделі, зокрема архітектури на основі BERT, показують відмінні результати за точністю, проте потребують значних обчислювальних ресурсів. Ще одним викликом є ефективне зберігання великої кількості векторів у пошуковому індексі. У таких випадках застосовуються методи компресії векторів, побудова графів з малою латентністю, використання GPU-ресурсів, а також технологій, що забезпечують мінімізацію затримок при пошуку.

Тому дане дослідження полягає у фокусі на інтеграції NLP-моделей для вирішення конкретних проблем пошуку у файлових сховищах. На відміну від загальних пошукових систем, що орієнтовані на вебконтент. Зокрема, буде розглянуто, як різні NLP-моделі можуть бути адаптовані та оптимізовані для роботи з різними типами файлів та форматами даних, включаючи текстові документи, PDF-файли, презентації тощо [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Засоби для розробки хмарного сховища файлів

Сучасний світ вимагає зручного та ефективного зберігання і доступу до даних. Хмарні сховища стають все більш популярними завдяки своїм численним перевагам. Вони дозволяють зберігати файли онлайн, забезпечуючи швидкий доступ до них з будь-якого пристрою, підключеного до Інтернету.

Уявімо ситуацію, коли ваш товариш попросив вас поділитися важливим документом. Ви можете мати цей документ на своєму ноутбуці або жорсткому диску, але якщо у вас немає доступу до цих пристроїв, це може стати проблемою. Саме тут на допомогу приходять хмарні сховища. Завантаживши документ в хмару, ви зможете поділитися ним миттєво, надавши доступ через посилання або електронну пошту.

Інший приклад: уявімо, що ви працюєте над проєктом і потребуєте доступу до великих обсягів даних, але пам’ять вашого пристрою обмежена. Використовуючи хмарне сховище, ви можете зберігати свої файли на сервері, не займаючи місця на своєму пристрої. Це особливо корисно для мобільних пристроїв, де пам’ять часто є обмеженою.

Хмарні сховища також надають додаткові функції, такі як автоматичне резервне копіювання, синхронізація файлів між різними пристроями, можливість спільного доступу до файлів для командної роботи та високий рівень безпеки даних. Сервісні компанії, що надають такі послуги, забезпечують надійність зберігання даних, використовуючи передові технології шифрування та багаторівневий захист інформації.

Таким чином, хмарні сховища є незамінним інструментом у сучасному цифровому світі, дозволяючи зберігати, ділитися та захищати дані ефективно і зручно.

1.2 Огляд та аналіз існуючих програмних рішень

Перед розробкою проєкту було вирішено проаналізувати аналогічні системи. Для цього було обрано три найпопулярніші хмарні сервіси для зберігання файлів.

Першим сервісом для аналізу було обрано Google Drive – це хмарний сервіс для зберігання файлів, розроблений і підтримуваний компанією Google (рисунок 1.1). Запущений 24 квітня 2012 року, Google Drive надає користувачам можливість зберігати файли на серверах Google, синхронізувати файли між пристроями та ділитися файлами з іншими користувачами. Станом на 2025 рік сервіс має понад 2 мільярди активних користувачів.

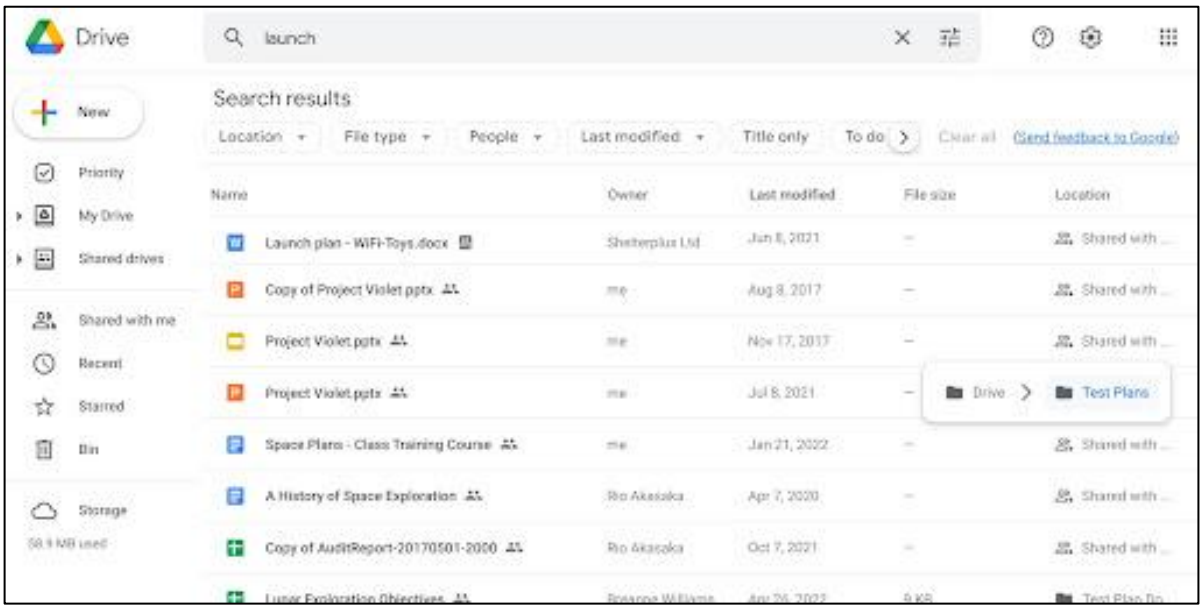

Рисунок 1.1 – Вигляд Google Drive

Google Drive пропонує користувачам безкоштовний обліковий запис із 15 ГБ простору для зберігання, який ділиться з іншими сервісами Google, такими як Gmail і Google Photos. Для тих, хто потребує більше місця, доступні платні плани, що дозволяють збільшити обсяг сховища до кількох

терабайтів. Основні функції Google Drive включають зберігання файлів будь-якого типу, доступ до файлів з будь-якого пристрою, автоматичне резервне копіювання та синхронізацію, а також інтеграцію з іншими сервісами Google, такими як Google Docs, Sheets та Slides. Це дозволяє створювати, редагувати та співпрацювати над документами в реальному часі.

Google Drive доступний на платформах Windows, macOS, Android, iOS та у веб-браузерах. Це забезпечує високий рівень доступності та зручності використання для користувачів з різними пристроями. Безпека даних на Google Drive забезпечується за допомогою шифрування даних як під час передачі, так і в стані спокою, а також двофакторної автентифікації для додаткового рівня захисту.

Наступним сервісом для аналізу є Dropbox (рисунок 1.2) – це один із перших і найпопулярніших хмарних сервісів для зберігання файлів, заснований у 2007 році Дрю Х'юстоном і Арашем Фердовсі. Dropbox дозволяє користувачам зберігати файли на хмарних серверах, синхронізувати їх між різними пристроями та ділитися файлами з іншими користувачами. На сьогоднішній день сервіс має понад 700 мільйонів зареєстрованих користувачів.

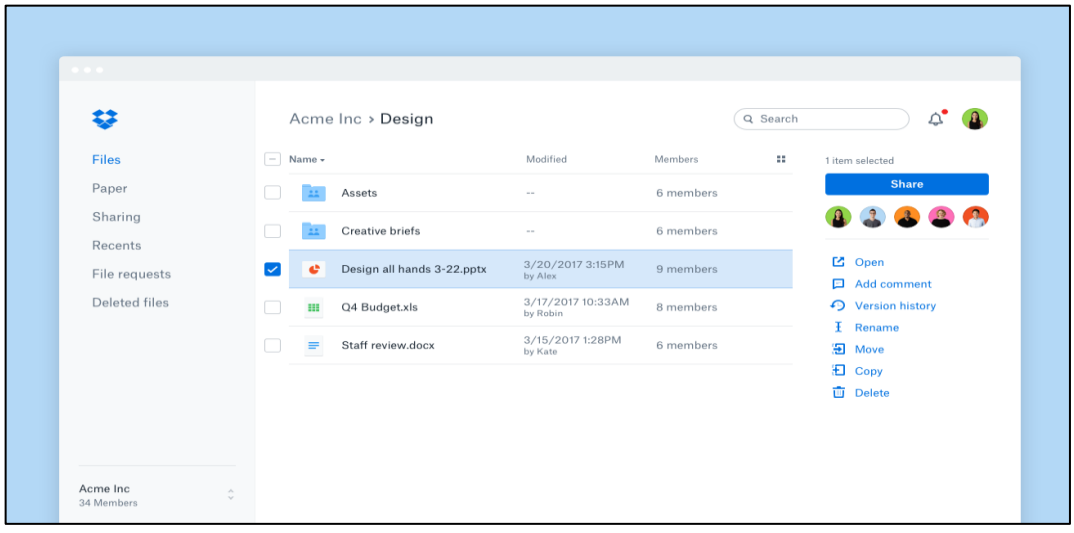

Рисунок 1.2 – Вигляд Dropbox

Dropbox пропонує безкоштовний обліковий запис із 2 ГБ простору для зберігання, а також кілька платних планів, що дозволяють збільшити обсяг сховища до декількох терабайтів. Ключові функції включають автоматичну синхронізацію файлів, зберігання версій файлів, спільний доступ до файлів і папок, а також інтеграцію з численними сторонніми додатками для підвищення продуктивності. Dropbox також підтримує функції для командної роботи, такі як коментарі до файлів і спільне редагування.

Dropbox доступний на платформах Windows, macOS, Linux, Android, iOS та через веб-браузери. Для забезпечення безпеки даних використовуються шифрування, двофакторна автентифікація та комплексні політики управління доступом.

Третім сервісом для аналізу є OneDrive (рисунок 1.3) – це хмарний сервіс для зберігання файлів, розроблений компанією Microsoft і інтегрований у пакет офісних програм Microsoft 365. Запущений у 2007 році, OneDrive надає користувачам можливість зберігати файли в хмарі, синхронізувати їх між пристроями та ділитися файлами з іншими користувачами.

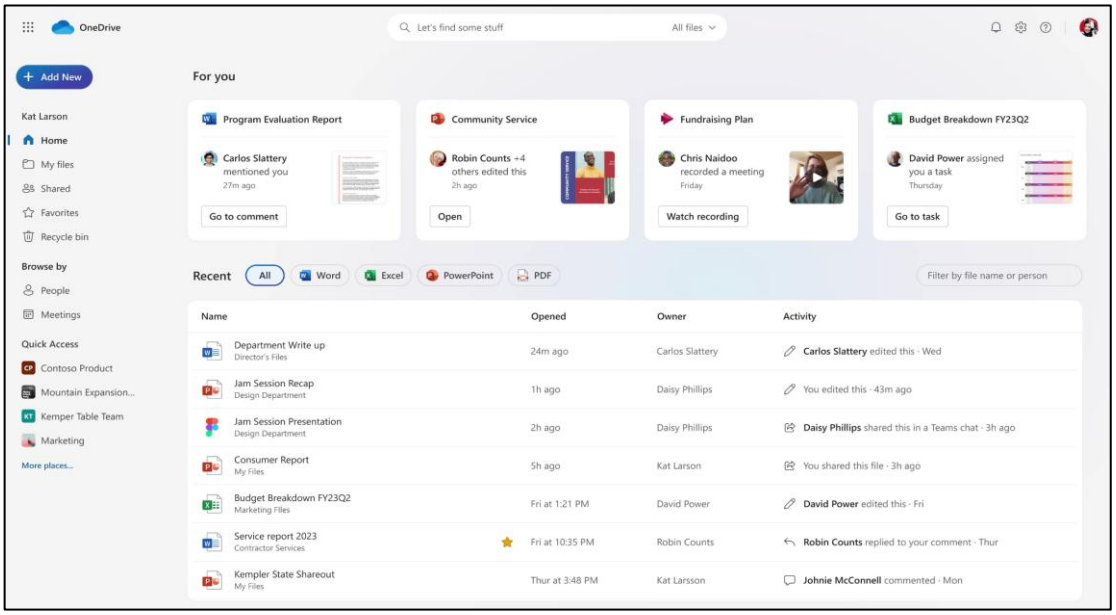

Рисунок 1.3 – Вигляд OneDrive

OneDrive пропонує безкоштовний обліковий запис із 5 ГБ простору для зберігання, а також платні плани, що дозволяють збільшити обсяг сховища до декількох терабайтів. Основні функції включають автоматичну синхронізацію файлів, інтеграцію з Microsoft Office для спільної роботи над документами, резервне копіювання та відновлення файлів, а також можливість створення альбомів для фотографій.

OneDrive доступний на платформах Windows, macOS, Android, iOS та через веб-браузери. Сервіс забезпечує високу безпеку даних за допомогою шифрування, двофакторної автентифікації та надійного управління доступом.

Таким чином, аналіз цих трьох популярних хмарних сервісів дозволяє зробити висновок про їх високу функціональність, зручність використання та надійність, що є важливими аспектами для розробки власного хмарного середовища для зберігання файлів.

1.3 Аналіз існуючих технологій розробки

Next.js – це JavaScript фреймворк [2]. Який має відкритий вихідний код для розробки серверних рендерингів і статичних вебдодатків на базі React [3]. Next.js розроблений і підтримується компанією Vercel [4]. Його мета – забезпечити розробникам засоби для створення високопродуктивних вебдодатків із мінімальними налаштуваннями.

Однією з ключових особливостей Next.js є підтримка рендерингу на сервері та статичної генерації. Це означає, що компоненти React можуть бути попередньо оброблені на сервері або згенеровані під час зборки, що суттєво підвищує швидкість завантаження сторінок. Крім того, Next.js забезпечує інтеграцію з API-інтерфейсами, що дозволяє розробникам легко реалізовувати серверну логіку без потреби у створенні окремого бекенду. Next.js також має вбудовану підтримку стилізації за допомогою CSS, що дозволяє реалізовувати гнучку і підтримувану систему оформлення без

додаткових бібліотек або складної конфігурації [5]. Для типізації коду фреймворк підтримує TypeScript «з коробки», що відповідає сучасним вимогам до розробки масштабованих і безпечних вебзастосунків. Проаналізувавши Next.js, стало зрозуміло, що цей фреймворк є популярним вибором для розробки сучасних вебдодатків. Його гнучкість і можливості серверного рендерингу роблять його ідеальним для створення SEO-оптимізованих та високопродуктивних вебдодатків.

FastAPI – це сучасний вебфреймворк для створення API на базі Python [6]. Він розроблений Себастьяном Раміресом і має відкритий вихідний код. FastAPI побудований на базі Starlette для вебфункціональності та Pydantic для валідації даних [7].

Як показали незалежні тести, FastAPI обладнає високую продуктивністю. Згідно з офіційною документацією FastAPI та результатами open-source тестувань, цей фреймворк забезпечує високу швидкодію завдяки нативній підтримці асинхронного програмування, що дозволяє ефективно обробляти тисячі одночасних запитів. Ще однією видатною особливістю FastAPI є автоматична генерація інтерактивної документації API. Документація створюється динамічно на основі анотацій типів Python, що дозволяє значно спростити процес тестування та інтеграції API. В основі валідації вхідних даних у FastAPI лежить бібліотека Pydantic, яка дозволяє створювати строгі моделі даних з автоматичним перетворенням типів, валідацією значень та наданням докладних повідомлень про помилки. Такий підхід гарантує високу надійність та безпечність API, особливо при роботі з користувацькими або сторонніми даними. FastAPI вважається одним з найшвидших вебфреймворків для Python, завдяки своїй простоті та високій продуктивності. Він ідеально підходить для створення високопродуктивних API з мінімальними зусиллями.

Shadcn – це колекція компонентів користувацького інтерфейсу, розроблена для використання з бібліотекою React [8]. Вона надає гнучкі та настроювані компоненти, які легко інтегруються в будь-який проєкт.

Плюси shadcn є велика кількість заздалегідь підготовлених компонентів, серед яких кнопки, інпути, випадаючі списки, тощо. Усі ці компоненти побудовані з використанням Tailwind CSS, що дозволяє легко налаштовувати стилі та адаптувати зовнішній вигляд до потреб конкретного застосунку [9]. Такий підхід відповідає принципам open-source інженерії, у якому контроль над компонуванням надається повністю в руки розробника. Інтеграція з React відбувається природно, без додаткових налаштувань [10]. Компоненти написані на TypeScript, що забезпечує безпеку, а також полегшує роботу з інтегрованими середовищами розробки. Документація до shadcn вирізняється своєю простотою, лаконічністю та доступністю. Shadcn забезпечує швидке створення інтерфейсів користувача, дозволяючи розробникам зосередитися на логіці додатка, а не на розробці UI компонентів з нуля.

PostgreSQL – це потужна, об'єктно-реляційна система керування базами даних з відкритим вихідним кодом [11]. Вона розроблена групою PostgreSQL Global Development Group і підтримується спільнотою розробників по всьому світу.

Головна перевага PostgreSQL є її суворе дотримання стандартів SQL, що дозволяє реалізовувати складні запити, вбудовані процедури та інші важливі засоби забезпечення цілісності даних [12]. Крім реляційної моделі, PostgreSQL підтримує напівструктуровані типи даних, такі як JSON, JSONB, XML, що робить її універсальним інструментом для побудови гібридних систем, де реляційні структури поєднуються з гнучкими підходами до зберігання даних [13]. Особливу увагу в PostgreSQL приділено розширюваності: користувачі можуть створювати власні функції на SQL, PL/pgSQL, Python, C або інших мовах, додавати агрегатні функції, нові типи даних, оператори, індекси, а також використовувати готові розширення.

Також PostgreSQL підтримує реплікацію – як потокову, так і логічну, що дозволяє будувати системи з високою доступністю. Окремо варто зазначити вбудовані засоби повнотекстового пошуку, які дозволяють індексувати великі текстові поля. Також PostgreSQL активно підтримується такими великими хмарними платформами, як Amazon RDS, Google Cloud SQL та Microsoft Azure Database for PostgreSQL, що свідчить про її адаптивність до сучасних DevOps-практик. PostgreSQL відома своєю надійністю, гнучкістю та багатими можливостями. Вона широко використовується в різних галузях, від фінансів до наукових досліджень.

MinIO – це високопродуктивне об'єктне сховище з відкритим вихідним кодом, сумісне з Amazon S3 [14]. Воно розроблене компанією MinIO, Inc. і забезпечує масштабоване зберігання даних для великих обсягів інформації.

Однією з важливих переваг MinIO є повна сумісність з API Amazon S3, що дозволяє використовувати існуючі клієнтські бібліотеки, інструменти та сервіси AWS без потреби у зміні архітектури додатку, зберігаючи при цьому контроль над інфраструктурою і даними. MinIO підтримує шифрування даних «на льоту» та «в спокої». Простота встановлення MinIO є ще однією ключовою перевагою: система може бути розгорнута у Docker-контейнері, Kubernetes-кластері або як звичайний серверний сервіс на будь-якій Unix-подібній ОС [15]. Окрім цього, MinIO підтримує багатокористувацьке середовище, надаючи можливість створення окремих просторів для зберігання (buckets) з ізольованими правами доступу, а також ведення журналів аудиту для відстеження всіх дій користувачів. MinIO ідеально підходить для зберігання великих обсягів даних у хмарних середовищах, забезпечуючи високу продуктивність та надійність. Він використовується для зберігання резервних копій, великих масивів даних та інших додатків, що потребують надійного об'єктного сховища.

1.4 Вимоги до системи

Завдання на розроблення програмного забезпечення – оптимізація в файлових сховищах засобами NLP.

Основні функції:

– авторизація та аутентифікація користувачів;

– завантаження, зберігання, видалення та файлів;

– можливість повернути файл після видалення;

– можливість додати файли до «улюблених»;

– пошук файлів.

Для того щоб увійти в систему, користувачеві потрібно вказати свій логін та пароль. Якщо користувача немає в базі даних, йому потрібно зареєструватися, вказавши логін, електронну пошту та пароль.

Необхідно розробити зручний та простий користувацький інтерфейс. Увійшовши на сайт, користувач повинен залогінитися або зареєструватися, якщо він не має облікового запису, щоб отримати подальший доступ до сайту. Після того як користувач залогінився, він зможе завантажувати файли, зберігати їх у папки, додавати до «улюблених», видаляти та редагувати файли, а також шукати необхідні файли.

У таблиці 1.1 показані основні вимоги до функціоналу.

Таблиця 1.1 – Основні вимоги до функціоналу

Авторизація та аутентифікація користувачів	Вхід за допомогою електронної пошти та пароля, реєстрація нового користувача із зазначенням логіна, електронної пошти та пароля
Файловий менеджмент	Завантаження файлів на сервер, зберігання файлів у хмарному середовищі, видалення файлів

Продовження таблиці 1.1

Улюблені файли	Додавання файлів до списку «улюблених», видалення файлів із списку «улюблених», перегляд списку «улюблених» файлів
Пошук файлів	Пошук файлів за назвою, фільтрація результатів пошуку

У таблиці 1.2 показані основні вимоги до інтерфейсу.

Таблиця 1.2 – Основні вимоги до інтерфейсу

Головна сторінка	Привітання користувача, кнопки для входу або реєстрації
Сторінка входу	Поля для введення логіну та паролю, кнопка для входу
Сторінка реєстрації	Відповідні поля для реєстрації, кнопка для реєстрації
Інтерфейс користувача після входу	Панель навігації з розділами: «Мої файли», «Улюблені», «Корзина», розділ для завантаження файлів, можливість додавання файлів до «улюблених», пошукове поле для швидкого знаходження файлів

Розробка хмарного середовища для зберігання файлів має забезпечити високу зручність для користувачів, надійність зберігання даних та ефективний доступ до них.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ХМАРНОГО СЕРЕДОВИЩА ДЛЯ ЗБЕРІГАННЯ ФАЙЛІВ

2.1 Специфікація вимог до системи

Вебсайт повинен мати інтуїтивно зрозумілий, зручний та надійний у використанні інтерфейс, що забезпечує комфортну роботу для користувачів різного рівня підготовки. Важливо, щоб він коректно функціонував на всіх сучасних браузерах і пристроях, забезпечуючи однаковий рівень продуктивності та зручності, незалежно від платформи.

До нефункціональних вимог вебсайту належать стабільність роботи, що гарантує безперебійну роботу навіть при великих навантаженнях, а також високий рівень безпеки, що захищає дані користувачів від несанкціонованого доступу. Вебресурс повинен працювати з оптимальною швидкістю, що сприяє позитивному користувацькому досвіду, а також мати зручний та приємний інтерфейс, який підтримує гармонійний вигляд і зрозумілу структуру інформації.

Функціональні вимоги включають можливість додавання та видалення файлів, що дозволяє користувачам легко керувати своїми даними. Важливою функцією є також можливість позначення файлів як улюблених для швидшого доступу. Система повинна забезпечувати ефективний пошук файлів, що спрощує навігацію та швидке знаходження необхідної інформації. Крім того, вебсайт має підтримувати функцію відновлення файлів після їх видалення, що додає рівень надійності та зручності у роботі з файлами.

Щодо вимог до інтерфейсу, він повинен бути зручним і інтуїтивним, з правильно організованим розміщенням інформації та гармонійною кольоровою гамою, що робить роботу комфортнішою та приємнішою для користувача.

2.2 Розробка UML-діаграм

UML – це спеціальна мова для графічного опису, призначається для об’єктного моделювання розробки різного програмного забезпечення [16]. Мова володіє широким профілем і має відкритий стандарт, в якому використовуються різні графічні позначення для того, щоб зробити абстрактну модель системи.

За допомогою UML розробники програмного забезпечення, вебсайтів, вебдодатків можуть зробити певну угоду, щоб представити загальні поняття за допомогою цих дій на проєкті досягається більша концентрація на архітектурі та проєктуванні.

Особливості:

– об’єктно орієнтована мова;

– опис системи з будь-якої точки зору;

– прості діаграми;

– можливість розширити та вводити власні графічні та текстові стереотипи;

– широке поширення;

– активний розвиток.

На рисунку 2.1 зображено варіанти користування вебсайтом. За допомогою діаграми зображено взаємодію користувачів з системою та основні функції системи.

Дана UML-діаграма демонструє логіку взаємодії користувача з системою хмарного зберігання файлів. Гість починає з головної сторінки, де може виконати вхід або зареєструватися, вказавши логін, email та пароль. Після авторизації користувач отримує доступ до панелі керування, яка відкриває функції перегляду файлів, додавання їх до улюблених, пошуку та роботи з корзиною. Реалізована можливість виходу з облікового запису забезпечує завершення сесії. Діаграма чітко розмежовує дії гостя і зареєстрованого користувача.

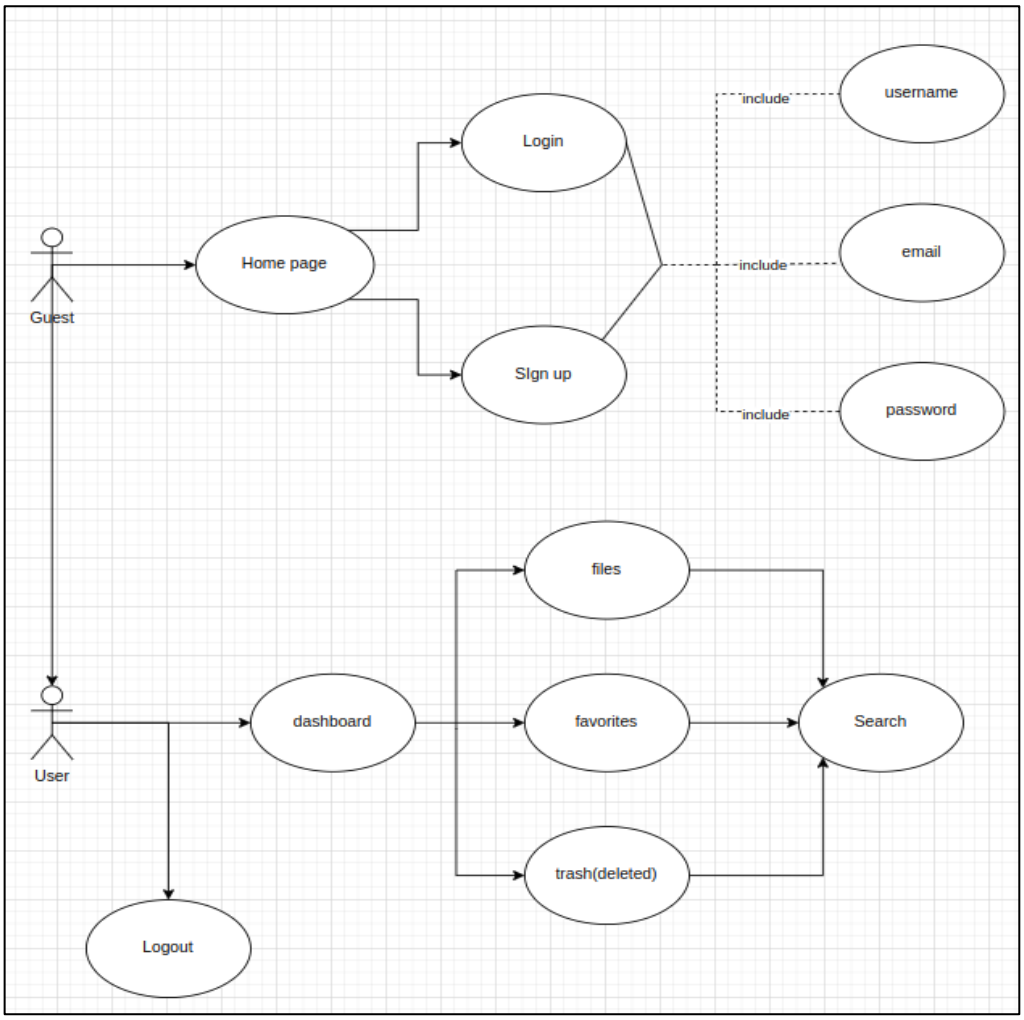

Рисунок 2.1 –Діаграма використання

У таблиці 2.1 представлена специфікація авторів, які зображені на UML діаграмі користування вебсайтом (рисунок 2.1).

Дана таблиця подає опис двох основних акторів, які взаємодіють із вебсайтом відповідно до UML-діаграми на рисунку 2.1. Першим є гість – користувач, який ще не авторизований у системі, отже, має доступ лише до функцій реєстрації або входу. Він не може взаємодіяти з функціоналом сайту, пов’язаним із файлами чи персональними даними. Другим актором є зареєстрований користувач, що вже створив обліковий запис і пройшов авторизацію. Такий користувач має повноцінний доступ до всіх функцій сайту: керування файлами, пошук, додавання до улюбленого, видалення та відновлення файлів у межах встановленого часу.

Таблиця 2.1 – Специфікація авторів

Актор	Опис
Гість	Гість – це людина, яка не увійшла в систему або її немає в базі даних, тобто вона не авторизована. Не зареєстрований користувач не має ніяких можливостей окрім реєстрації або входу в обліковий запис
Зареєстрований користувач	Зареєстрований користувач – це людина яка є в базі даних та виконала вхід у систему. Користувач має доступ до всіх функцій вебсайту. Він може завантажувати файли, шукати файли, додавати файли в список улюблених, видаляти файли, а також має можливість повернути видалений файл до 24 годин

У таблиці 2.2 представлений опис різних варіантів користування вебсайтом.

Таблиця 2.2 – Опис варіантів користування

Завантаження файлів	Зареєстрований користувач може завантажити будь-який файл
Перегляд файлів	Зареєстрований користувач може переглядати свої файли
Пошук фалів	Зареєстрований користувач може шукати будь-які файли

Продовження таблиці 2.2

Видалення файлів	Зареєстрований користувач може видалити файл, після чого він переміститься до корзини та автоматично видалиться якщо пройде 24 години
Повернення файлів	Зареєстрований користувач може повернути файл з корзини, якщо не пройшло 24 години
Добавляти файли до вподобаних	Зареєстрований користувач може добавляти файли у список «улюблених»

2.3 Вибір технологій розробки хмарного середовища

2.3.1 Вибір технологій розробки графічного інтерфейсу

Хмарне середовище для зберігання файлів побудовано за допомогою мікросервісної архітектури. Мікросервісна архітектура дозволяє розділити систему на невеликі, незалежно розроблювані та розгортанні компоненти, що взаємодіють між собою. У цьому середовищі буде реалізовано три основні мікросервіси: мікросервіс для автентифікації, мікросервіс для фронтенду та мікросервіс для взаємодії з файлами.

Для розробки інтерфейсу користувача хмарного середовища для зберігання файлів було обрано такі технології: Next.js, Shadcn та Tailwind CSS. Кожна з цих технологій відіграє ключову роль у створенні ефективного, зручного та привабливого користувацького інтерфейсу.

Next.js – це фреймворк для React, що дозволяє створювати серверні рендеринг додатки та статичні вебсайти з використанням React [10]. Його було обрано завдяки його можливостям, таким як попередня рендеринг та

створення статичних сайтів, що підвищує продуктивність і SEO. Next.js дозволяє легко створювати маршрутизацію та має підтримку API маршрутів для створення бекенд кінцевих точок. Основні переваги Next.js включають швидкість, простоту використання та масштабованість, що робить його ідеальним вибором для сучасних вебдодатків.

Shadcn – це бібліотека стилізованих компонентів для React, яка допомагає створювати естетично привабливі та функціональні інтерфейси користувача [8]. Вона пришвидшує процес розробки, надаючи готові до використання компоненти, які можна легко налаштовувати. Shadcn була обрана завдяки своїй здатності забезпечувати консистентний дизайн і прискорювати роботу над візуальною частиною проєкту.

Tailwind CSS – це сучасний CSS-фреймворк, який пропонує набір готових до використання класів, що відповідають за стилізацію елементів [9]. Він обраний за своєю простотою використання та можливістю швидко створювати стильовані компоненти без необхідності писати власний CSS [5].

2.3.2 Вибір технологій розробки backend частини хмарного сервісу для збереження файлів

Для розробки бекенду хмарного середовища для зберігання файлів були обрані такі технології: FastAPI, PostgreSQL та MinIO. Кожна з цих технологій відіграє ключову роль у створенні ефективної, масштабованої та надійної системи.

FastAPI – це сучасний, високопродуктивний вебфреймворк для створення API на основі Python 3.6+ [7]. Він є фреймворком, який забезпечує високу продуктивність завдяки підтримці асинхронного програмування, що дозволяє створювати швидкі та ефективні бекенд-сервіси, зі швидкодією, близькою до Node.js та Go. Він вирізняється простотою використання, оскільки дає змогу розробляти кінцеві точки API

з мінімальним обсягом коду, що пришвидшує процес розробки. Крім того, FastAPI автоматично генерує документацію API за допомогою OpenAPI та JSON Schema, що значно полегшує інтеграцію та тестування [17]. Завдяки використанню типів Python фреймворк забезпечує автоматичну валідацію вхідних даних і підвищений рівень безпеки.

FastAPI є чудовим вибором для побудови надійних та масштабованих бекенд-сервісів, які потребують швидкої обробки запитів та асинхронних операцій.

PostgreSQL – це, відкрита система керування базами даних з відкритим кодом. PostgreSQL є потужною СУБД, яка підтримує виконання складних SQL-запитів і роботу з різноманітними типами даних, що робить її оптимальним вибором для складних додатків [11]. Вона забезпечує високу масштабованість як у горизонтальному, так і у вертикальному напрямку, дозволяючи ефективно працювати зі зростаючими обсягами інформації. Крім того, PostgreSQL є розширюваною, підтримує численні розширення та дає змогу додавати нові функції без значних змін у структурі бази даних. PostgreSQL є надійною та гнучкою СУБД для зберігання даних у хмарному середовищі зберігання файлів, що забезпечує високий рівень продуктивності та надійності.

MinIO – це високопродуктивна система зберігання об'єктів з відкритим кодом, сумісна з Amazon S3 API (рисунок 2.2). Основні переваги MinIO включають:

– висока продуктивність. MinIO забезпечує швидкий доступ до даних, що важливо для хмарних систем зберігання файлів;

– сумісність з S3 API. Повна сумісність з Amazon S3 API, що дозволяє використовувати існуючі інструменти та бібліотеки для роботи з об'єктами;

– масштабованість. Підтримка розподілених та кластерних конфігурацій, що дозволяє масштабувати систему відповідно до потреб;

– безпека. Вбудовані механізми шифрування та підтримка аутентифікації та авторизації для забезпечення безпеки даних.

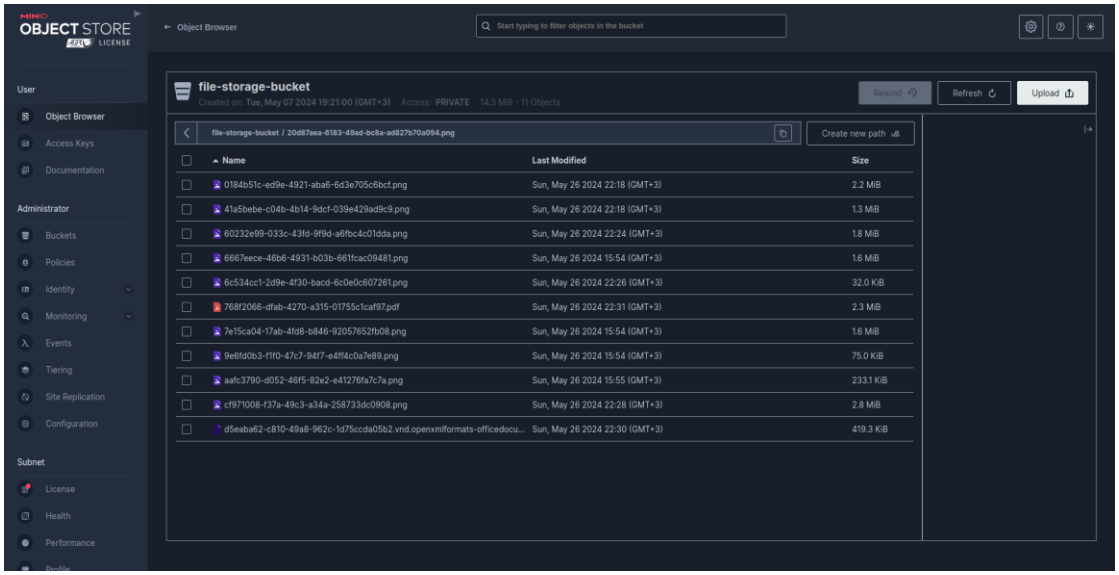

Рисунок 2.2 – MinIO bucket для зберігання файлів

MinIO є ідеальним вибором для зберігання великих обсягів даних у хмарному середовищі, забезпечуючи високу доступність та швидкий доступ до файлів [14].

2.4 Розробка графічного інтерфейсу вебсервісу

Для розробки інтерфейсу хмарного середовища для зберігання файлів були обрані такі технології: Next.js, Shadcn та TailwindCSS. Кожна з цих технологій грає важливу роль у створенні ефективного, привабливого та інтуїтивно зрозумілого інтерфейсу користувача.

Next.js – це фреймворк для React, що дозволяє створювати серверні рендеринг додатки та статичні вебсайти з використанням React [3]. Next.js підвищує продуктивність додатків і покращує SEO завдяки кращій індексації сторінок пошуковими системами. Фреймворк підтримує як статичне, так і динамічне попереднє рендеринг, що сприяє швидкому завантаженню сторінок. Вбудована маршрутизація на основі файлової структури спрощує організацію та навігацію в додатку. Крім того, Next.js

дозволяє створювати API маршрути без потреби в окремому сервері, що додає гнучкості та спрощує архітектуру. Next.js є чудовим вибором для створення сучасних вебдодатків з високою продуктивністю і зручною розробкою.

Shadcn – це бібліотека стилізованих компонентів для React, яка допомагає створювати естетично привабливі та функціональні інтерфейси користувача (рисунок 2.3). Основні переваги Shadcn включають:

– готові компоненти: бібліотека надає великий набір готових до використання компонентів, що дозволяє швидко створювати інтерфейси;

– налаштовуваність: компоненти легко налаштовуються для відповідності стилю вашого додатку;

– консистентність: використання Shadcn забезпечує консистентний дизайн у всьому додатку, що покращує користувацький досвід;

– інтеграція з React: оскільки Shadcn побудована для React, вона легко інтегрується з Next.js, що спрощує розробку інтерфейсу.

Рисунок 2.3 – Сторінка компонентів Shadcn

Shadcn є ідеальним вибором для швидкого створення привабливих та узгоджених інтерфейсів користувача [8].

TailwindCSS – це утилітарний CSS фреймворк, що дозволяє створювати інтерфейси користувача за допомогою набору класів, які можна легко комбінувати для створення унікальних дизайнів. TailwindCSS є чудовим інструментом для створення гнучких та адаптивних інтерфейсів. Він дозволяє будувати складний дизайн без потреби писати звичайний CSS. Використання готових класів значно прискорює розробку та полегшує зміну стилів у процесі роботи. Завдяки вбудованій підтримці адаптивного дизайну інтерфейси автоматично підлаштовуються під різні пристрої. TailwindCSS також забезпечує консистентність стилів у всьому додатку, що спрощує його підтримку та масштабування.

2.5 Проєктування бази даних

PostgreSQL дозволяє ефективно зберігати метаінформацію про файли – такі як назва, шлях, дата створення, тип, мітки, ключові слова, вектори подібності або ембедінги, згенеровані NLP моделями. Завдяки своїй підтримці розширених типів даних та індексації СУБД забезпечує швидкий пошук по текстових полях, ключових словах або семантичній схожості, що є критично важливим для реалізації розумного пошуку.

Крім того, PostgreSQL можна інтегрувати з модулями для обробки векторного пошуку, що дозволяє зберігати та шукати векторні подання тексту, які генеруються NLP моделями [1]. Це забезпечує можливість семантичного пошуку в межах файлового сховища, коли користувач може вводити запити природною мовою, а система знаходить релевантні файли не лише за ключовими словами, а й за змістом.

Таким чином, PostgreSQL є не просто сховищем даних, а й активним компонентом, який забезпечує швидкий, гнучкий та масштабований пошук, підтримуючи складні запити, обробку великих обсягів інформації та зберігання структурованих NLP-даних. Оскільки сайт побудований за допомогою мікросервісної архітектури, тому для правильної роботи

використовуються дві бази даних. Перша для аутентифікації та авторизації користувачів та друга для керування файлами.

На рисунку 2.4 зображено таблицю для збереження профілю користувачів.

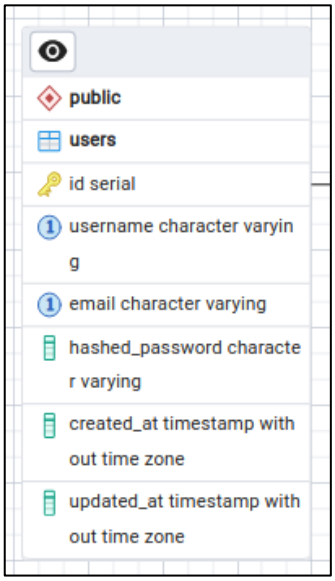

Рисунок 2.4 – Таблиця користувачів

База даних для аутентифікації та авторизації. Ця таблиця представляє собою такі поля:

– id – унікальний ідентифікатор;

– username – унікальне ім'я користувача, яке використовується для входу в систему та відображення у профілі;

– email – унікальна електронна адреса користувача, яка використовується для автентифікації та комунікації;

– hashed_password – захешований пароль користувача, що забезпечує безпечне зберігання пароля;

– created_at – час створення запису про користувача, використовується для відстеження дати реєстрації;

– updated_at – час останнього оновлення запису про користувача, використовується для відстеження змін у профілі користувача.

На рисунку 2.5 зображено таблицю для збереження refresh токену, який потрібен для поновлення access токену, тому на проєкті використовується технологія JWT токенів.

Рисунок 2.5 – Таблиця токенів

Ця таблиця представляє собою такі поля:

– id – унікальний ключ для ідентифікації кожного запису токену в таблиці;

– refresh_token – токен для оновлення доступу, що використовується для отримання нового токену доступу без повторного входу;

– user_id – посилання на користувача, якому належить цей токен, що зв'язує токен з відповідним записом у таблиці users.

На рисунку 2.6 зображено таблицю для збереження міграцій, тобто версій бази даних.

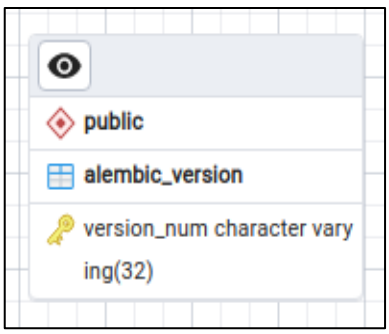

Рисунок – 2.6 Таблиця міграцій

База даних для керування файлами. На рисунку 2.7 зображено таблицю для збереження файлів.

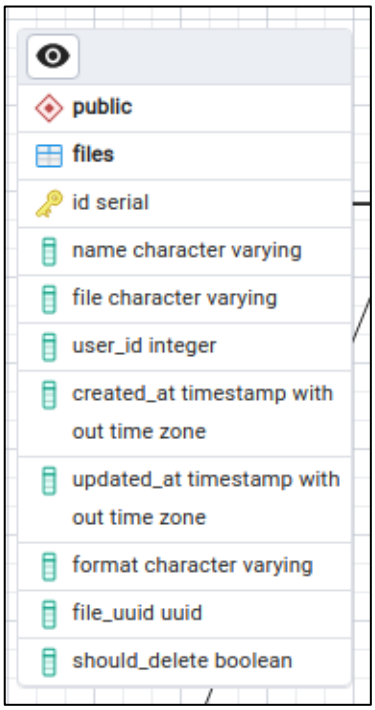

Рисунок 2.7 – Таблиця файлів

Ця таблиця представляє собою такі поля:

– id – унікальний ключ для ідентифікації кожного файлу в таблиці;

name – ім'я файлу, яке може бути відображене користувачеві;

– file – шлях до файлу або URL для доступу до файлу в системі зберігання;

– user_id – посилання на користувача, який завантажив цей файл, що зв'язує файл з відповідним записом у таблиці users;

– created_at – час створення запису про файл, використовується для відстеження дати завантаження;

– updated_at – час останнього оновлення запису про файл, використовується для відстеження змін у файлі;

– format – формат файлу, наприклад «jpg», «pdf», що дозволяє ідентифікувати тип вмісту;

– file_uuid – унікальний ідентифікатор файлу, який використовується для запобігання конфліктам і забезпечення унікальності файлів;

– should_delete – прапорець для позначення файлів, які повинні бути видалені. Використовується для управління життєвим циклом файлів, наприклад, для запланованого видалення.

Також повний програмний код для завантаження файлів наведено у лістингу А.1 додатку А.

На рисунку 2.8 зображено таблицю для збереження улюблених файлів.

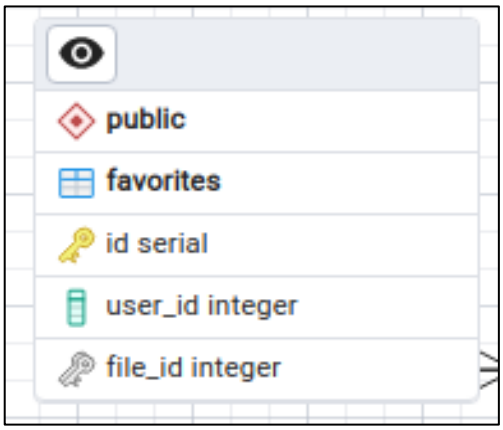

Рисунок 2.8 – Таблиця улюблених файлів

Ця таблиця представляє собою такі поля:

– id – унікальний ключ для ідентифікації кожного файлу в таблиці;

– user_id – посилання на користувача, який вподоба цей файл, що зв'язує файл з відповідним записом у таблиці users;

– file_id – посилання на файл, який вподобав користувач, що зв'язує запис з відповідним у таблиці files.

На рисунку 2.9 зображено таблицю для збереження запланованих завдань.

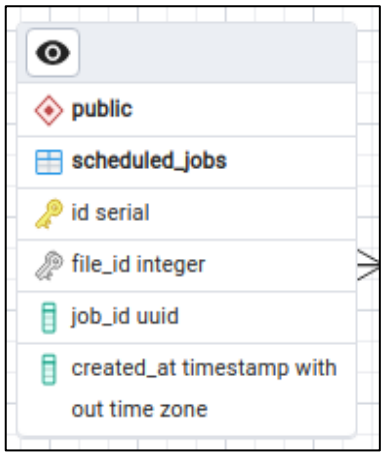

Рисунок 2.9 – Таблиця запланованих завдань

Ця таблиця представляє собою такі поля:

– id – унікальний ключ для ідентифікації кожного запису в таблиці;

– file_id – посилання на файл для якого виконується завдання, що зв'язує запис з відповідним у таблиці files;

– job_id – унікальний ідентифікатор для кожного завдання;

– created_at – час створення запису про файл, використовується для відстеження дати завантаження.

На рисунку. 2.10 зображено таблицю для збереження міграцій, тобто версій бази даних.

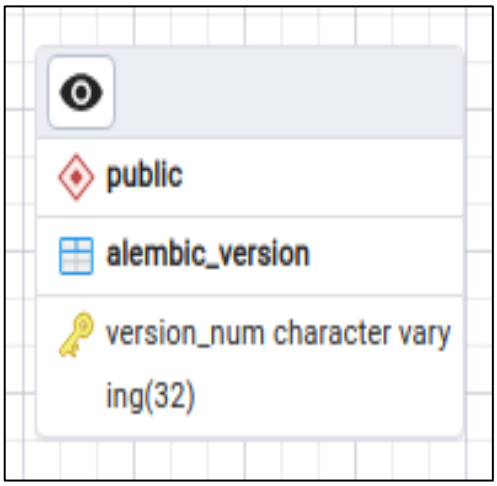

Рисунок 2.10 – Таблиця міграцій

2.6 Семантичний пошук за допомогою NLP-моделей

У дослідженні розглядаються два основні підходи до інтелектуального пошуку файлів: семантичний аналіз з використанням моделей обробки природної мови (NLP) та векторний пошук з перетворенням текстового вмісту у векторні представлення.

В роботі розглянуто два основні підходи до інтелектуального пошуку файлів. Один із методів передбачає пошук за ключовими словами та семантичний аналіз. Це можна зробити завдяки використання моделей обробки природної мови (NLP), таких як BERT або GPT, для аналізу текстового вмісту файлів і визначення їхньої релевантності до запиту. Це дозволяє знаходити файли навіть за неточними або контекстно пов’язаними запитами. Друга стратегія передбачає векторний пошук, що дозволяють знаходити документи не за точним збігом ключових слів, а за їхнім значенням у контексті. Для цього текстовий вміст файлів перетворюється у векторні представлення за допомогою моделей, таких як Word2Vec, FastText або Sentence Transformers. Ці вектори зберігаються в оптимізованих базах для швидкого пошуку, наприклад, FAISS або Pinecone, що дає змогу ефективно знаходити документи з близьким вмістом.

Технології та інструменти.

BERT (Bidirectional Encoder Representations from Transformers) – це модель штучного інтелекту (МШІ), розроблена Google, яка забезпечує двонаправлений аналіз контексту слів у реченні. Завдяки цьому МШІ ефективно виконує завдання семантичного аналізу та класифікації тексту. GPT (Generative Pre-trained Transformer) – це модель від OpenAI, яка працює в односпрямованому режимі (зліва направо) і використовується для генерації текстів, автоматичного доповнення запитів та інших завдань.

Запропоновано використати бібліотеку Sentence Transformers, яка призначена для створення компактних багатовимірних векторів

тексту (sentence embeddings), для зберігання змісту текстового документа. Це оптимальне рішення для семантичного пошуку.

Для зберігання та швидкого пошуку векторних представлень була обрана хмарна платформа Pinecone. Pinecone забезпечує масштабованість і високу швидкість роботи з великими наборами даних, що робить її ефективним рішенням для задач семантичного пошуку.

На рисунку 2.11 зображено пошуковий індекс Pinecone.

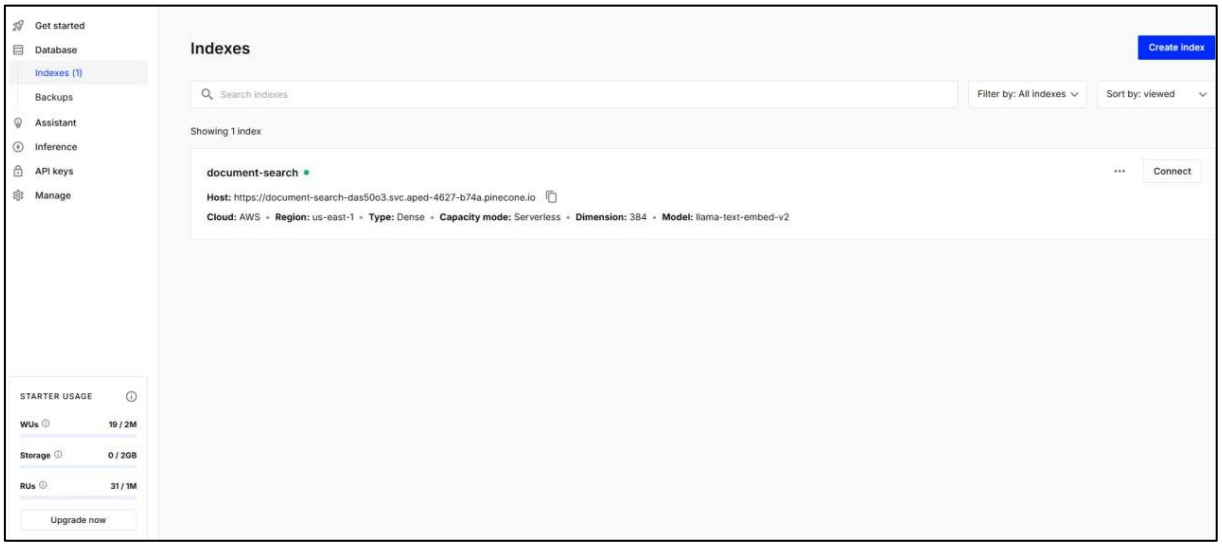

Рисунок 2.11 – Пошуковий індекс Pinecone

Для програмної реалізації інтелектуального пошуку файлів була застосована друга стратегія, а саме векторний пошук. Початковим етапом є вилучення контексту з документів за допомогою бібліотек python-pptx, python-docx, pdfplumber та стандартного контекстного менеджера мови Python [18]. Розбиття великих документів на менші фрагменти. Після вилучення контексту здійснюється перевірка його довжини. У разі, якщо документ містить велику кількість сторінок, його текстовий вміст розбивається на менші фрагменти (чанки). Конвертація фрагментів у векторні представлення. Ці фрагменти конвертуються у векторні

представлення за допомогою моделі all-MiniLM-L6-v2 із бібліотеки Sentence Transformers.

На рисунку 2.12 зображено векторні представлення тексту.

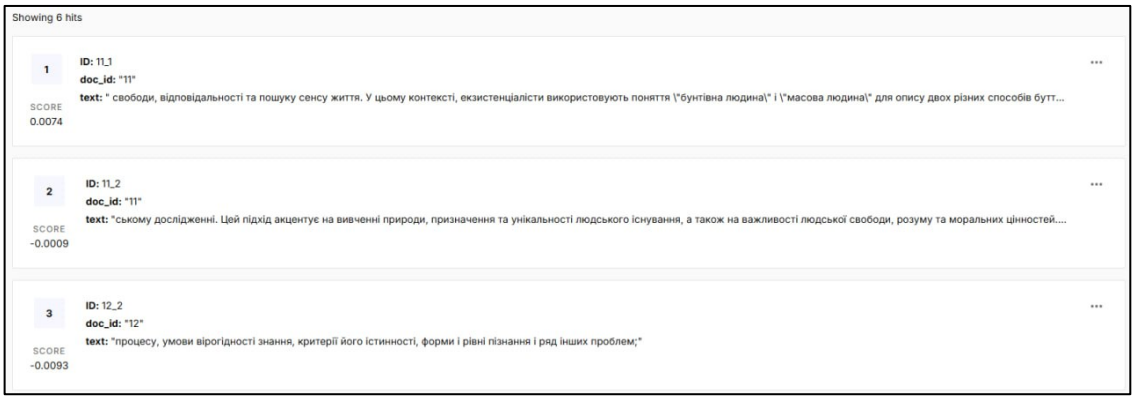

Рисунок 2.12 – Векторні представлення тексту

Зберігання векторів у базі даних Pinecone. Отримані векторні представлення зберігаються у векторній базі даних Pinecone, що забезпечує ефективний та швидкий пошук. Обробка користувацьких запитів та порівняння з наявними векторами. Фінальний етап передбачає обробку користувацького запиту, який також перетворюється у векторне представлення та надсилається до Pinecone для порівняння з наявними векторами. У результаті система повертає від 7 до 10 документів із найвищим рівнем відповідності або, за відсутності релевантних збігів, не знаходить жодного схожого документа.

Якщо порівнювати ці два підходи, то вони мають суттєві відмінності в принципах роботи та ефективності (таблиця 2.3). Перший підхід базується на класичному пошуковому індексуванні, де документи знаходяться за точним або частковим збігом слів, що забезпечує швидкість, але не гарантує розуміння контексту. Векторний пошук, навпаки, використовує нейромережеві моделі для перетворення тексту в багатовимірні вектори, що дозволяє знаходити документи за змістовною схожістю, навіть якщо ключові слова відсутні (рисунок 2.13).

Таблиця 2.3 – Порівняльний аналіз класичного індексування та векторного пошуку

Критерій	Класичне індексування	Векторний пошук
Розуміння контексту	Низьке	Високе
Точність	Середня	Дуже висока
Швидкість	Висока	Середня
Ресурсоємність	Низька	Висока
Простота впровадження	Висока	Низька

Рисунок 2.13 – Конфігурація пошукового індексу

Векторний пошук вимагає більше обчислювальних ресурсів і складніший у впровадженні. Тому для задач, де важлива швидкість і точний збіг, підходить перший метод, а коли потрібно враховувати семантику та контекст, краще використовувати векторний пошук, особливо для складних семантичних запитів та неструктурованих даних.

Використання моделей BERT, GPT у поєднанні з Pinecone, дозволяє знаходити документи навіть за нечіткими запитами або за змістовною схожістю. Ця робота має потенціал для значного покращення ефективності роботи з документами у різних галузях.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБДОДАТКУ

3.1 Опис модулів та компонентів серверної частини

На рисунку 3.1 побудовано структуру вебсайту.

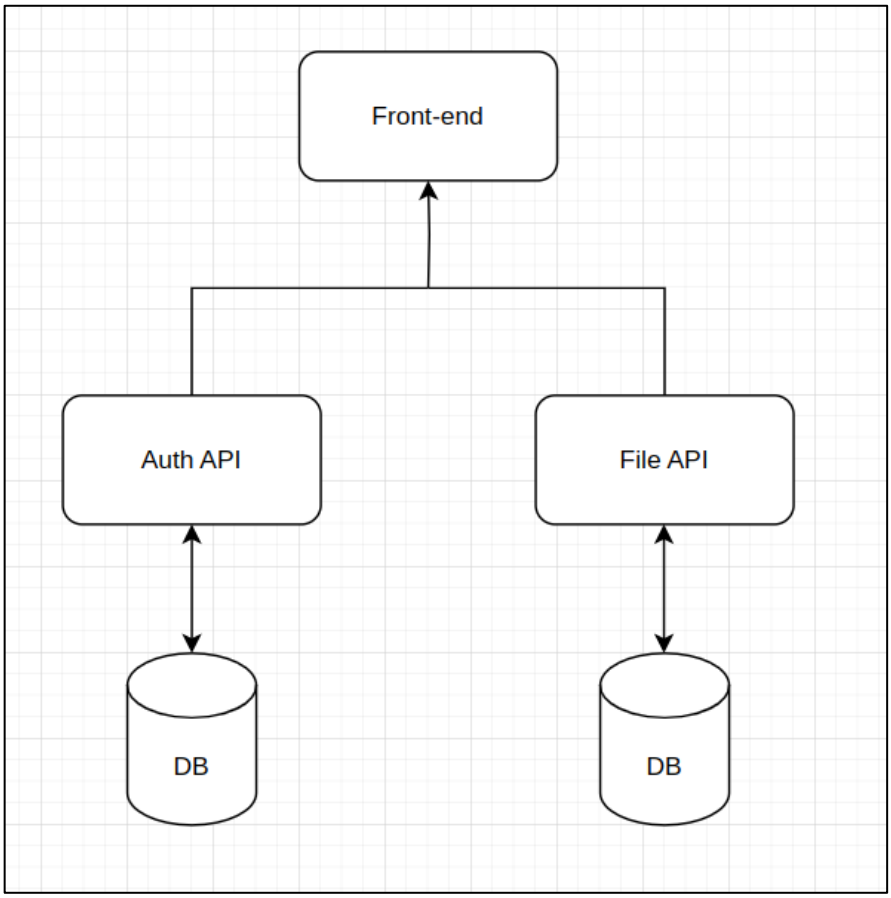

Рисунок 3.1 – Структура мікросервісів

Для початку розглянем реалізацію сервісу для файлів, які технології для нього потрібні та як їх використовувати.

Мікросервіси – набір незалежних сервісів, кожен з яких виконує окрему бізнес-функцію. Кожен мікросервіс має свою власну базу даних і може працювати автономно, що дозволяє їм бути незалежними від інших сервісів.

На рисунку 3.2 зображено головний файл програми в який підключено файл з маршрутами, тобто з API, яка виглядає наступний чином (рисунок 3.3 та рисунок 3.4).

```python
from fastapi import FastAPI
from .routers import files
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI()

origins = [
    "http://localhost:3000",
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(router=files.router, prefix="/api/v1")
```

Рисунок 3.2 – Головний файл проєкту

В FastAPI, маршрути (routes) визначають кінцеві точки (endpoints) для взаємодії клієнтів з вашим вебдодатком. Вони дозволяють визначати, які функції повинні бути виконані у відповідь на HTTP-запити до певних URL [19]. Основні завдання маршрутів включають:

– обробка HTTP-запитів. Кожен маршрут асоціюється з певним шляхом URL і методом HTTP (GET, POST, PUT, DELETE тощо). Коли сервер отримує запит на цей шлях з відповідним методом, він передає запит до визначеної функції обробника;

– визначення кінцевих точок API. Веб-API складається з різних кінцевих точок, які забезпечують доступ до різних ресурсів або функціональності. Кожен маршрут відповідає за конкретну кінцеву точку.

```python
@router.get(
    path: "/files",
    dependencies=[Depends(is_token_expired), Depends(is_authenticated)],
    summary="Get all files",
    response_model=list[FilesFavorite],
)
async def get_all_files(q: str | None = None, db: Session = Depends(get_db)):
    if q:
        files = (
            db.query(models.File, models.Favorite.id)
            .filter(
                *criterion: models.File.should_delete.is_(False),
                func.lower(models.File.name).contains(q),
            )
            .join(models.Favorite, isouter=True)
            .order_by(models.File.created_at.desc())
            .all()
        )
    else:
        files = (
            db.query(models.File, models.Favorite.id)
            .filter_by(should_delete=False)
            .join(models.Favorite, isouter=True)
            .order_by(models.File.created_at.desc())
            .all()
        )

    result = [{"data": file, "fav": fav_id} for file, fav_id in files]

    return result
```

Рисунок 3.3 – Ендпоінт для того щоб взяти всі файли

```python
@router.post(
    path: "/file/upload",
    dependencies=[Depends(is_token_expired), Depends(is_authenticated)],
    summary="Create file",
    response_model=File,
)
async def upload_file(
    file: UploadFile,
    user_id: Annotated[str, Depends(is_authenticated)],
    db: Session = Depends(get_db),
):
    extension = file.headers.get("content-type").split("/")[1]
    # Upload the file to storage minIO
    file_url, file_uuid = await upload(file, extension)

    filename = remove_extension(file.filename)

    db_file = models.File(
        name=filename,
        file=file_url,
        file_uuid=file_uuid,
        user_id=int(user_id),
        format=extension,
    )
    db.add(db_file)
    db.commit()
    db.refresh(db_file)

    uploaded_file = db.query(models.File).filter_by(id=db_file.id).first()

    return uploaded_file
```

Рисунок 3.4 – Ендпоінт для завантаження файлів

Ці два зображення це всього лиш 2 ендпоінта з багатьох присутніх у даному мікросервісі. Ось ще список додаткових ендпоінтів:

– /connection;

– /favorites;

– /deleted;

– /file/{file_id};

– /file-restore/{file_id};

– /favorites/add.

Тепер настав час перейти до мікросервісу пов’язаного з авторизацією та автентифікацією. Як ми розуміємо, наші ендпоінти для взаємодії з файлами повинні бути захищені від сторонніх людей, тобто вони повинні віддавати дані тільки для авторизованих користувачів, для цього і потрібен цей сервіс.

На рисунку 3.5 зображено головний файл програми в який підключено файл з маршрутами.

```python
from fastapi import FastAPI
from .routers import tokens
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI()

origins = [
    "http://localhost:3000",
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(router=tokens.router, prefix="/api/v1")
```

Рисунок 3.5 – Головний файл

На рисунку 3.6 зображено фрагмент коду бекенд-логіки авторизації користувача, він реалізованій за допомогою FastAPI. Конкретно тут описано обробник HTTP-запиту на маршрут /login, який виконує перевірку логіну й паролю та створює access і refresh токени. Тобто з API виглядає наступним чином (рисунок 3.7 та рисунок 3.8).

```
vitossss
@router.post(
    path: "/login",
    summary="Create access and refresh tokens for user",
    response_model=TokenUserSchema,
)
async def login(
        response: Response, form_data: OAuth2PasswordRequestForm = Depends(), db: Session = Depends(get_db)
):
    user = db.query(models.User).filter_by(username=form_data.username).first()
    if user is None:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Incorrect username or password",
        )

    if not verify_password(form_data.password, str(user.hashed_password)):
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Incorrect username or password",
        )

    refresh_token = create_refresh_token(user.email)
    instance = models.Token(refresh=refresh_token, user_id=user.id)
    db.add(instance)
    db.commit()

    response.set_cookie(key="refresh_token", value=refresh_token, httponly=True)
    return {
        "user": UserOut(id=user.id, username=user.username, email=user.email),
        "access_token": create_access_token(user.email, user.id),
        "refresh_token": refresh_token,
    }
```

Рисунок 3.6 – Ендпоінт для логіну

Для того, щоб створити API потрібно визначити маршрути, які відповідають на HTTP-запити, і додати до них відповідні функції-

обробники. Запуск додатку здійснюється за допомогою Uvicorn, який забезпечує виконання FastAPI додатку. Після запуску можна тестувати API через веб-браузер за адресою http://127.0.0.1:8000 або використовувати автоматично згенеровану документацію Swagger UI за адресою http://127.0.0.1:8000/docs. Додавання нових маршрутів і функціональності здійснюється шляхом визначення нових шляхів і методів обробки запитів, таких як GET, POST тощо.

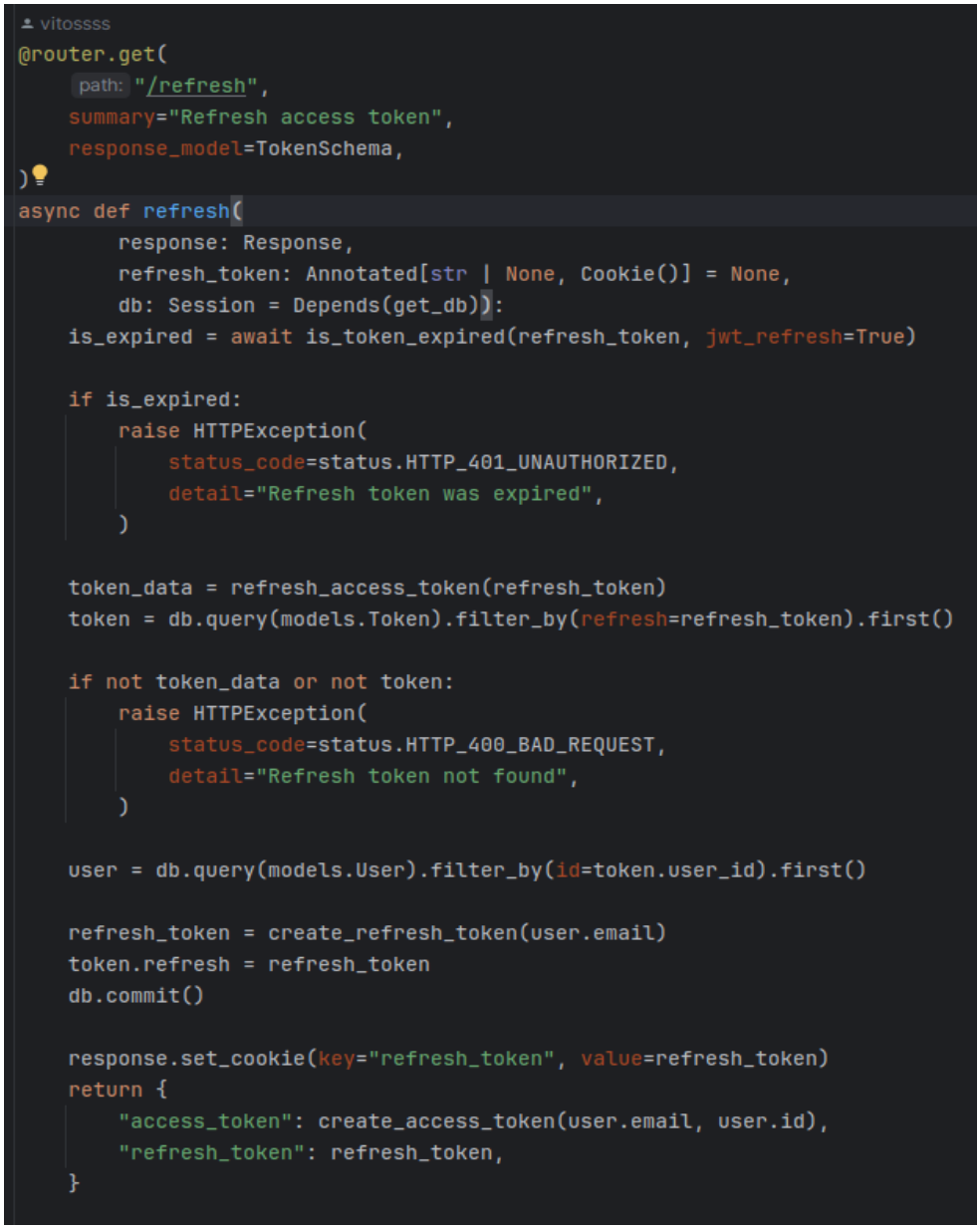

Рисунок 3.7 – Ендпоінт для перевірки токену та для створення нового, якщо той не дійсний

Окрім поточних ендпоінтів сервіс автентифікації та авторизації налічує наступні ендпоінти:

– /connection;

– /create_user;

– /get_user;

– /logout.

Також одним з найголовніших аспектів є база даних. Їх у нас дві штуки, як показано на серверній структурі (рисунок 3.1). Отож на рисунку 3.8 зображений файл в якому видно як здійснюється підключення бази даних.

```python
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

from app.config import (
    POSTGRES_USER,
    POSTGRES_PASSWORD,
    POSTGRES_HOST,
    POSTGRES_DB,
    POSTGRES_PORT,
)

SQLALCHEMY_DATABASE_URL = f"postgresql://{POSTGRES_USER}:{POSTGRES_PASSWORD}@{POSTGRES_HOST}:{POSTGRES_PORT}/{POSTGRES_DB}"

engine = create_engine(SQLALCHEMY_DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()
```

Рисунок 3.8 – Файл піключення бази даних

На наступних двох рисунках (рисунок 3.9 та рисунок 3.10) зображення моделі обох мікросервісів. Моделі в FastAPI використовуються для визначення структури даних, які передаються між клієнтом і сервером. Вони забезпечують валідацію вхідних даних, що дозволяє гарантувати, що дані відповідають очікуваному формату та типам [7].

```python
23 usages  vitossss
class File(Base):
    __tablename__ = "files"

    id = Column(Integer, primary_key=True)
    name = Column(String, nullable=False)
    file = Column(String, nullable=False)
    file_uuid = Column(UUID, nullable=False)
    user_id = Column(Integer, nullable=False)
    format = Column(String, nullable=False)
    created_at = Column(DateTime, default=datetime.utcnow)
    updated_at = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)
    should_delete = Column(Boolean, default=False)

    favorites = relationship(
        argument: "Favorite", back_populates="files", cascade="all, delete-orphan"
    )
    scheduled_jobs = relationship(
        argument: "ScheduledJob", back_populates="files", cascade="all, delete-orphan"
    )


8 usages  vitossss
class Favorite(Base):
    __tablename__ = "favorites"

    id = Column(Integer, primary_key=True)
    user_id = Column(Integer, nullable=False)
    file_id = Column(Integer, ForeignKey("files.id"), nullable=False)

    files = relationship( argument: "File", back_populates="favorites")


2 usages  vitossss
class ScheduledJob(Base):
    __tablename__ = "scheduled_jobs"

    id = Column(Integer, primary_key=True)
    file_id = Column(Integer, ForeignKey("files.id"), nullable=False)
    job_id = Column(UUID, nullable=False)
    created_at = Column(DateTime, default=datetime.utcnow)

    files = relationship( argument: "File", back_populates="scheduled_jobs")
```

Рисунок 3.9 – Моделі мікросервісу для файлів

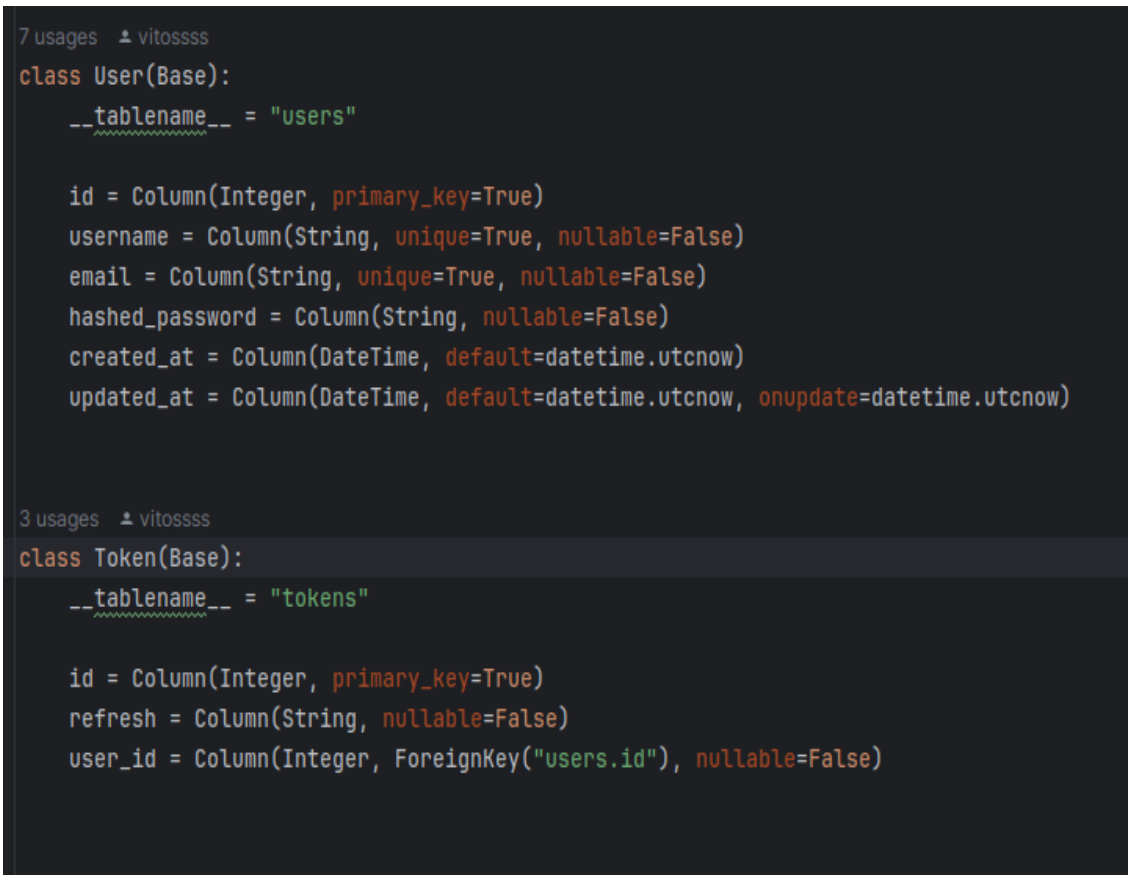

Рисунок 3.10 – Моделі мікросервісу для користувачів

Важливу роль так само відіграє безпека, тому на наступному рисунку 3.11 зображено функції безпеки. Використання моделей також спрощує роботу з даними всередині програми, оскільки дозволяє чітко визначити, які поля і типи даних очікуються. Крім того, моделі в FastAPI автоматично інтегруються в документацію API, надаючи розробникам і користувачам API чітке уявлення про структуру запитів і відповідей. Це підвищує зручність розробки і тестування, а також робить API більш зрозумілим і надійним.

Окремою частиною програми є настройка локального сховища файлів за допомогою MinIO та Docker. На рисунку 3.12 зображений docker-compose файл який дозволяє запустити MinIO сервіс локально на своїй машині.

```python
2 usages  vitossss
def get_hashed_password(password: str) -> str:
    return password_context.hash(password)


2 usages  vitossss
def verify_password(password: str, hashed_pass: str) -> bool:
    return password_context.verify(password, hashed_pass)


3 usages  vitossss
def decode_access_token(token: str) -> int:
    decoded_jwt = jwt.decode(token, JWT_SECRET_KEY, ALGORITHM)
    return decoded_jwt.get("user_id")


3 usages  vitossss
def create_access_token(subject: Union[str, Any], user_id: Union[str, Any]) -> str:
    expires_delta = datetime.utcnow() + timedelta(
        minutes=ACCESS_TOKEN_EXPIRE_MINUTES
    )

    to_encode = {"exp": expires_delta, "sub": str(subject), "user_id": str(user_id), "isAuth": True}
    encoded_jwt = jwt.encode(to_encode, JWT_SECRET_KEY, ALGORITHM)
    return encoded_jwt


3 usages  vitossss
def create_refresh_token(subject: Union[str, Any]) -> str:
    expires_delta = datetime.utcnow() + timedelta(
        minutes=REFRESH_TOKEN_EXPIRE_MINUTES
    )

    to_encode = {"exp": expires_delta, "sub": str(subject)}
    encoded_jwt = jwt.encode(to_encode, JWT_REFRESH_SECRET_KEY, ALGORITHM)
    return encoded_jwt


2 usages  vitossss
def refresh_access_token(refresh_token: str):
    encoded_jwt = jwt.decode(refresh_token, JWT_REFRESH_SECRET_KEY, ALGORITHM)
    return encoded_jwt
```

Рисунок 3.11 – Функції, які забезпечують безпеку та захист

```yaml
services:
  db:
    image: postgres
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    env_file:
      - .env
    ports:
      - "5431:5432"

  minio:
    image: minio/minio
    volumes:
      - minio_data:/data
    command: server --console-address ":9001" /data
    env_file:
      - .env
    ports:
      - "9000:9000"
      - "9001:9001"

volumes:
  postgres_data:
  minio_data:
```

Рисунок 3.12 – Докер файл для MinIO та бази даних

Docker Compose використовується для спрощення процесу налаштування і керування багатоконтейнерними Docker-додатками. За допомогою Docker Compose можна визначити всі сервіси, які складають додаток, у файлі docker-compose.yml, включаючи бази даних, кеші, вебсервіси та інші компоненти. Це дозволяє запускати весь додаток за допомогою однієї команди, спрощуючи управління залежностями та конфігураціями. Docker Compose також забезпечує ізольоване середовище для кожного сервісу, що полегшує розробку, тестування та розгортання додатків у різних середовищах. Це інструмент, який значно спрощує роботу з контейнерами і підвищує ефективність роботи з Docker.

На наступному рисунку 3.13 зображено підключення MinIO сервісу та функції для збереження файлів до локального сховища.

```python
client = Minio(
    MINIO_HOSTNAME,
    access_key=MINIO_ACCESS_KEY,
    secret_key=MINIO_SECRET_KEY,
    secure=False,
)
bucket_name = MINIO_BUCKET


1 usage  vitossss
async def upload_file(file, extension):
    found = client.bucket_exists(bucket_name)

    if not found:
        client.make_bucket(bucket_name)
    else:
        print(f"Bucket {bucket_name} already exists")

    # Generate a UUID for the file name
    file_uuid = str(uuid.uuid4())
    file_name = f"{file_uuid}.{extension}"

    file_content = await file.read()

    client.put_object(
        bucket_name,
        file_name,
        io.BytesIO(file_content),
        length=len(file_content),
    )

    # Warning! MinIO share the link of object only for 7 days!
    url = client.presigned_get_object(bucket_name, file_name)

    return url, file_uuid


1 usage  vitossss
def delete_file(file):
    client.remove_object(bucket_name,  object_name: f"{file.file_uuid}.{file.format}")

    return None
```

Рисунок 3.13 – Сетап MinIO та його залежностей

Останній важливий компонент системи файлів є автоматичне видалення через 24 години після перенесення файлу до корзини (рисунок 3.14).

```
2 usages  vitosssss *
async def schedule_file_deletion(db, file_obj, filename):
    vitosssss
    def delete_file():
        try:
            db.delete(file_obj)
            db.commit()
            # Delete the file from storage minIO
            delete(file_obj)

            print(f"Deleted file record from database: {filename}")
        except Exception as e:
            print(f"Error occurred during file deletion: {e}")

    date = datetime.now() + timedelta(hours=24)
    job = scheduler.add_job(delete_file, trigger: "date", run_date=date)

    print("Job has been scheduled successfully")

    return job.id
```

Рисунок 3.14 – Автоматичне завдання видалення файлу через 24 години

3.2 Опис модулів та компонентів клієнтської частини

На рисунку 3.15 зображена структура клієнтської частини вебсайту, яка ілюструє логіку переходів між основними модулями. Початкова точка – головна сторінка, з якої користувач може перейти до входу або реєстрації. Після авторизації відкривається доступ до панелі керування (dashboard), яка є центральним вузлом навігації.

Повний програмний код для компоненти dashboard наведено у лістингу Б.1 додатку Б.

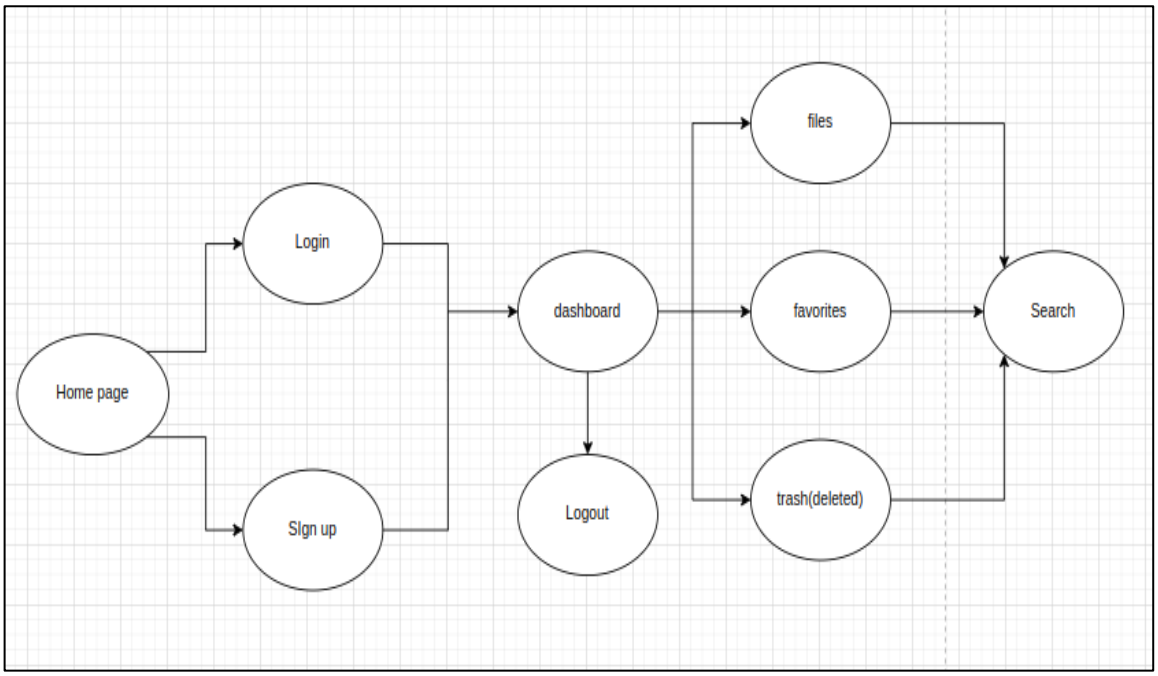

Рисунок 3.15 – Структура вебсайту

З панелі користувач переходить до розділів – файли, улюблені, корзина або пошук. Пошук охоплює всі ці розділи, що забезпечує гнучкість у роботі з файлами. Також реалізовано можливість виходу з облікового запису.

На рисунку 3.16 зображено домашню сторінку сайту, а також на рисунку 3.17 частково зображена компонента цієї сторінки в коді.

Рисунок 3.16 – Домашня сторінка

```tsx
export default function Home() : React.JSX.Element  {  no usages  vitossss
  const { user : User | null  } = useAuth();
  const router : AppRouterInstance  = useRouter();

  if (user) {
    router.push( href: "/dashboard");
    return (
      <div className="w-full h-[calc(100vh-150px)] flex justify-center items-center">
        <div className="loader"></div>
      </div>
    );
  }

  return (
    <div className="bg-slate-50 grainy-light">
      <section>
        <MaxWidthWrapper className="pb-24 pt-10 lg:grid lg:grid-cols-3 sm:pb-32 lg:gap-x-0 xl:gap-x-8 lg:pt-24 xl:pt-32 lg:pb-52">
          <div className="col-span-2 px-6 lg:px-0 lg:pt-4">
            <div className="relative mx-auto text-center lg:text-left flex flex-col items-center lg:items-start">
              <div className="absolute w-28 left-0 -top-20 hidden lg:block">
                <div className="absolute inset-x-0 bottom-0 bg-gradient-to-t via-slate-50/50 from-slate-50 h-28" />
                <img src="/file_storage_1.jpg" className="w-full rounded-md" />
              </div>
              <h1 className="relative w-fit tracking-tight text-balance mt-16 font-bold !leading-tight text-gray-900 text-5xl md:text-6xl lg:text-7xl">
                {" "}
                <span className="bg-orange-600 px-2 text-white">
                  Empower
                </span>{" "}
                Your Files,{" "}
                <span className="bg-orange-600 px-2 text-white">Unleash</span>{" "}
                Your Potential!
              </h1>
              <p className="mt-8 text-lg lg:pr-10 max-w-prose text-center lg:text-left text-balance md:text-wrap">
                Welcome to our innovative file storage platform, where
                efficiency meets security and convenience reigns supreme. Say
```

Рисунок 3.17 – Компонента домашньої сторінки

Наступні два рисунки (рисунок 3.18 та рисунок 3.19) форми логіну та реєстрації.

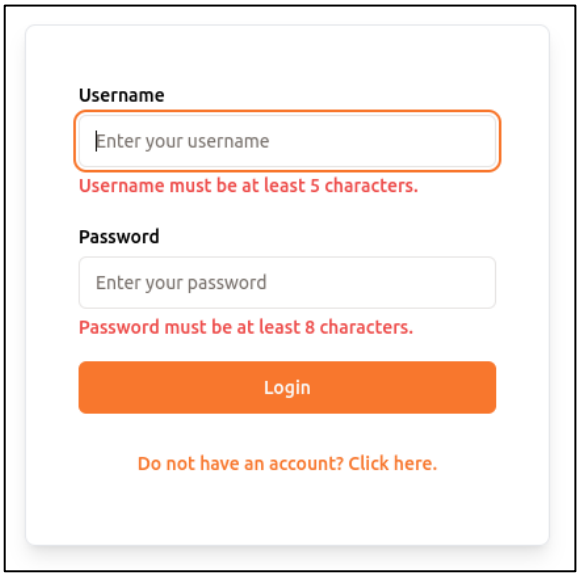

Рисунок – 3.18 Форма логіну

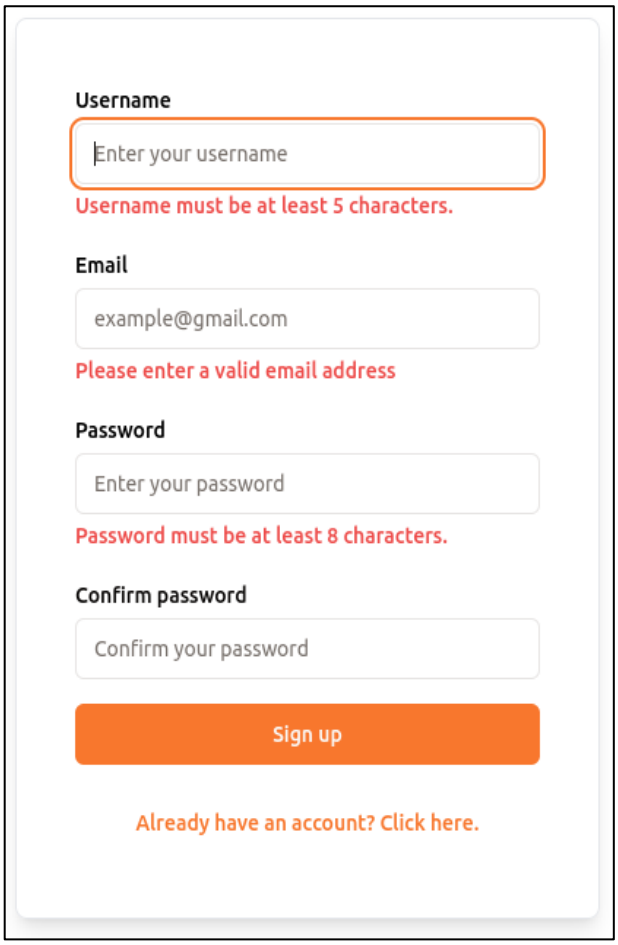

Рисунок 3.19 – Форма реєстрації

Валідація форми є важливою частиною вебдодатків, оскільки допомагає забезпечити правильність та безпеку введених користувачем даних. Перевірка даних на валідність перед їхнім використанням дозволяє запобігти можливим помилкам та атакам на додаток. Валідація форм також забезпечує зручність для користувачів, бо вони отримують сповіщення про помилки або некоректно введені дані ще до надсилання форми. Це сприяє покращенню користувацького досвіду та збільшенню надійності додатку, що в кінцевому підсумку може позитивно позначитися на успіху проєкту та задоволенні його користувачів.

У формі реєстрації важливо, щоб поле для підтвердження пароля містило такий самий пароль, який був введений у відповідному полі для введення пароля. Це дозволяє переконатися, що користувач ввів пароль

правильно і не допустив помилки. Це стандартна практика для багатьох вебдодатків і допомагає підвищити безпеку та надійність авторизації користувачів.

Повний програмний код провайдера для автентифікації наведено у лістингу В.1 додатку В.

На наступному рисунку 3.20 зображено дошку з усіма файлами після того як користувач залогінився.

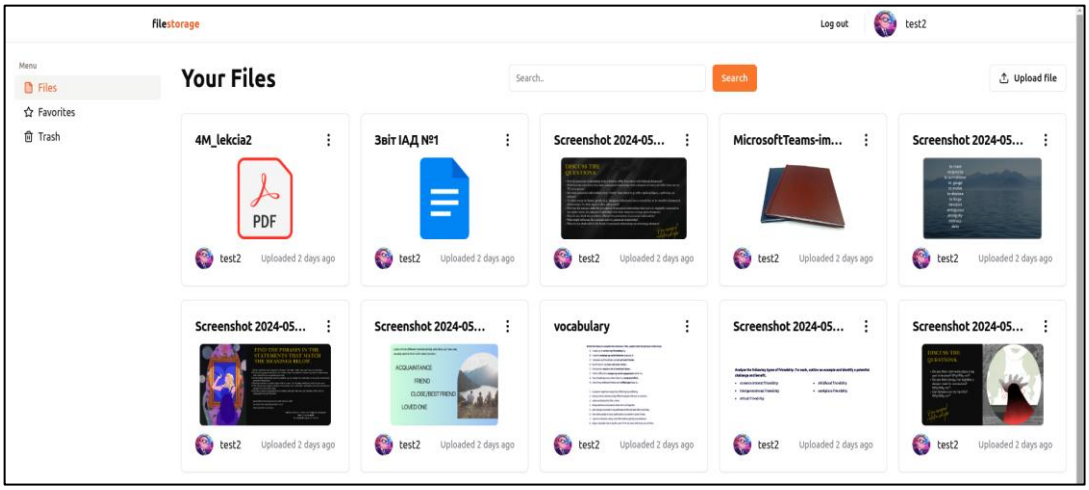

Рисунок 3.20 – Дошка з усіма файлами користувача

Наступний рисунок 3.21 зображує форму для завантаження файлу, а також зображує цю компоненту у коді (рисунок 3.22).

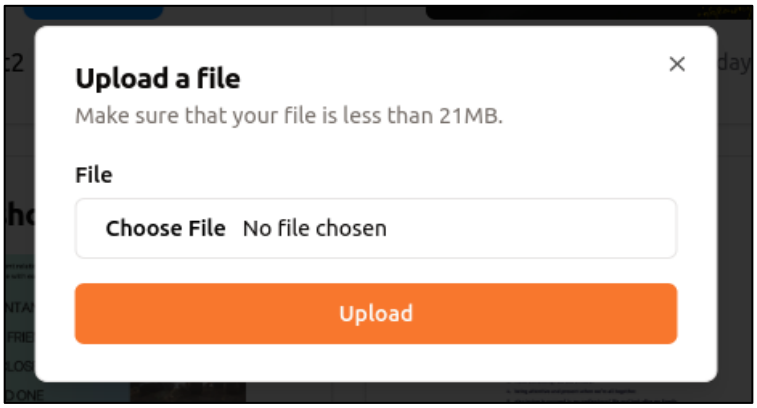

Рисунок 3.21 – Форма для завантаження файлу

```
const onSubmit = async (values: z.infer<typeof formSchema>) : Promise<void>  => {  Show usages  ▲ vitossss
  const formData : FormData  = new FormData();
  formData.append( name: "file", values.file[0]);
  const config : {headers: {"Content-Type": str...  = {
    headers: {
      "Content-Type": "multipart/form-data",
    },
  };

  try {
    const response : AxiosResponse<any, any>  = await axiosInstance.post(
      url: "/api/v1/file/upload",
      formData,
      config,
    );
    if (response.status === 200) {
      setFiles( value: (prevState : [] | FilesFavorite[] ) => [
        ...prevState,
        { data: response.data, fav: null },
      ]);
      toast( {...props}: {
        title: "File Uploaded",
      });
      setOpen( value: false);
    }
  } catch (e) {
    toast( {...props}: {
      title: "Something went wrong",
      description: "Your file could not be uploaded, try again later",
    });
  }
};
```

Рисунок 3.22 – Реалізація компоненти для завантаження файлу

3.3 Інструкція до розробленого вебсайту

Щоб розпочати роботу з розробленим вебсайтом, користувачеві необхідно виконати базову послідовність дій, спрямовану на реєстрацію, авторизацію та подальше використання функціональних можливостей системи. Для цього спочатку потрібно відкрити сайт у браузері, ввівши відповідну URL-адресу. На головній сторінці відображається інтерфейс з кнопками «Вхід» та «Реєстрація». Нові користувачі можуть створити обліковий запис, заповнивши просту форму з введенням логіну, електронної пошти та паролю. Після підтвердження даних користувача буде збережено в базі даних, і система автоматично надасть доступ до особистого кабінету.

Після входу відкривається клієнтська частина сайту, де користувач має змогу взаємодіяти з файлами. Усі елементи інтерфейсу реалізовані з урахуванням принципів зручності, логічної ієрархії та простоти

використання. У верхній частині екрана розташована панель навігації, де можна перейти до таких розділів, як «Мої файли», «Улюблені», «Кошик» або використати функцію пошуку. У центральній частині інтерфейсу відображаються файли, які можна завантажувати, переглядати, видаляти або додавати до вибраного списку. Видалені файли зберігаються протягом 24 годин, після чого система їх очищає автоматично, забезпечуючи баланс між безпекою та ефективним використанням сховища.

Ключовою інновацією вебсайту є інтегрований пошуковий механізм, побудований на основі NLP-моделей. Завдяки обробці природної мови користувачі можуть вводити запити не лише у форматі точного збігу імені файлу, а й у вигляді синонімів, контекстних ключових слів або навіть нечітких формулювань. Пошукова система аналізує запит, використовує попередньо навчену модель та повертає релевантні результати з урахуванням семантики та структури даних. Це дозволяє значно скоротити час на пошук потрібної інформації, особливо у великих сховищах.

Крім того, система оптимізована під різні типи користувачів. Наприклад, гість має обмежений доступ і може лише зареєструватися або увійти до системи. У свою чергу, авторизований користувач має повноцінний доступ до всіх функцій – від завантаження файлів до керування улюбленими та використання пошуку. Така архітектура забезпечує як безпеку, так і гнучкість, дозволяючи ефективно масштабувати систему.

Після завершення сесії користувач може натиснути кнопку «Вихід», що призведе до очищення токена з локального сховища та завершення авторизації. Цей підхід підвищує безпеку та захищає дані користувача. У підсумку, розроблений вебсайт є зручним, сучасним і технічно складним інструментом для зберігання та пошуку файлів, побудованим із використанням передових технологій веброзробки та обробки природної мови.

ВИСНОВКИ

В рамках кваліфікаційної роботи було розроблено хмарний сервіс для збереження файлів.

Розробка сучасних вебзастосунків і хмарних сервісів вимагає глибокого аналізу та використання найкращих доступних технологій. Зокрема, для створення вебсайту для прослуховування музики та хмарного середовища для зберігання файлів було обрано ряд популярних технологій і бібліотек. Кожна з цих технологій забезпечує високий рівень функціональності, продуктивності та зручності використання, що є ключовими факторами для успішної розробки [20].

Було розглянуто процес проєктування та розробки хмарного середовища для зберігання файлів. Перш за все, було визначено специфікацію вимог до системи, що слугувала основою для подальшої розробки. Далі було створено UML діаграми, які допомогли візуалізувати структуру та взаємодію компонентів системи.

Після цього був проведений детальний аналіз і вибір технологій для розробки як графічного інтерфейсу користувача, так і бекенду.

Також було детально описано розробку графічного інтерфейсу вебсервісу, включаючи ключові компоненти та принципи дизайну. Останнім етапом стало проєктування бази даних, де були визначені необхідні таблиці та поля для забезпечення зберігання та управління даними користувачів і файлами.

Розглянули модулі та компоненти як серверної, так і клієнтської частини вебсайту. Описали їх функціональні можливості та взаємодію між собою для забезпечення належної роботи вебсайту.

Також було підготовлено інструкцію до розробленого вебсайту, яка дозволить користувачам ефективно використовувати всі його функціональність. Ця інструкція охоплює кроки реєстрації, входу в систему, навігації по сайту, взаємодії з контентом та виходу з системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Natural Language Processing. *NNLM*. URL: https://www.nnlm.gov/guides/data-glossary/natural-language-processing (дата звернення: 11.04.2025).

2. JavaScript | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 15.04.2025).

3. Get Started. *React Hook Form - performant, flexible and extensible form library*. URL: https://react-hook-form.com/get-started (дата звернення: 20.04.2025).

4. Vercel. Next.js Docs | Next.js. *Next.js by Vercel - The React Framework*. URL: https://nextjs.org/docs (дата звернення: 26.04.2025).

5. W3Schools.com. *W3Schools Online Web Tutorials*. URL: https://www.w3schools.com/css/ (дата звернення: 30.04.2025).

6. Welcome to Python.org. *Python.org*. URL: https://www.python.org/ (дата звернення: 03.05.2025).

7. FastAPI. *FastAPI*. URL: https://fastapi.tiangolo.com/ (дата звернення: 15.04.2025).

8. Introduction. *Build your Component Library - shadcn/ui*. URL: https://ui.shadcn.com/docs (дата звернення: 07.05.2025).

9. Installing with Vite - Installation. *Tailwind CSS - Rapidly build modern websites without ever leaving your HTML*. URL: https://tailwindcss.com/docs/installation/using-vite (дата звернення: 11.05.2025).

10. Quick start – react. *React*. URL: https://react.dev/learn (дата звернення: 20.05.2025).

11. PostgreSQL 17.5 documentation. *PostgreSQL Documentation*. URL: https://www.postgresql.org/docs/current/index.html (дата звернення: 27.05.2025).

12. Ukrainian W. Уроки від W3Schools українською онлайн. W3Schools українською для школярів та студентів. URL: https://w3schoolsua.github.io/sql/index.html#gsc.tab=0 (дата звернення: 01.06.2025).

13. Extensible Markup Language (XML) 1.0 (Fifth Edition). *W3C*. URL: https://www.w3.org/TR/xml/ (дата звернення: 04.06.2025).

14. MinIO object storage for macos – minio object storage for macos. *MinIO | S3 Compatible Storage for AI*. URL: https://min.io/docs/minio/macos/index.html (дата звернення: 07.06.2025).

15. Docker Hub Container Image Library | App Containerization. *Docker Hub Container Image Library | App Containerization*. URL: https://hub.docker.com/ (дата звернення: 09.06.2025).

16. Як будувати UML-діаграми. Url : https://dou.ua/forums/topic/40575/ дата звернення: 10.06.2025).

17. Working with JSON - Learn web development | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/JSON (дата звернення: 11.06.2025).

18. Advanced Python Scheduler – APScheduler. *Advanced Python Scheduler*. URL: https://apscheduler.readthedocs.io/en/3.x/ (дата звернення: 12.06.2025).

19. HTTP: Hypertext Transfer Protocol | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (дата звернення: 12.06.2025).

20. Коваль, Г. К., Гриньова, О. Є. Застосування моделей NLP для оптимізації пошуку в файлових сховищах. *Радіоелектроніка та молодь у XXI столітті*, м. Харків. 2025. С. 34-36. URL: https://openarchive.nure.ua/handle/document/30714 (дата звернення: 13.06.2025).