НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ
кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

“До захисту допущено”
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

“___”_____2024 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
“Цифрові технології в енергетиці”
зі спеціальності *122 “Комп’ютерні науки”*

на тему: **Сервіс зберігання файлів з авторизованим доступом**

Виконала:
студентка IV курсу, групи ТР-03

Нерух Анжела Сергіївна _____
(прізвище, ім’я, по батькові) (підпис)

Керівник: *доцент каф. цифрових технологій в енергетиці*
доцент, к.військ.н Андрій ОНИСЬКО _____
(посада, вчене звання ,науковий ступінь, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент: *доцент, к.т.н. Олександр ГАГАРІН*
_____ _____
(посада, вчене звання, науковий ступінь, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль: *асистент Микита ГОЛОВАКІН* _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка ___
(підпис)

Київ – 2024

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА_

«__»_____2024 р.

ЗАВДАННЯ
на дипломну роботу студенту

Нерух Анжелі Сергіївні
(прізвище, ім’я, по батькові)

1. Тема роботи **Сервіс зберігання файлів з авторизованим доступом**

керівник роботи ***Онисько Андрій Ілліч, доцент к.військ.н***
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 202__ р. №_____

2. Термін подання роботи студентом **07.06.2024 року**

3. Вихідні дані до роботи мова програмування Typescript, фреймворки Next.js та Nest.js, СКБД MySql, Typeorm, середовище розробки Visual Studio Code, інструмент для тестування Swagger

4. Перелік питань, які потрібно розробити
1) провести аналіз серед наявних аналогів
2) веб-систем для для зберігання файлів

3) аргументувати вибрані інструменти, технології та алгоритми для реалізації програмного забезпечення

4) побудувати архітектури програмного забезпечення, баз даних та сховища файлів

5) створити прикладний програмний веб-інтерфейс

6) представити процес застосування веб-інтерфейсу

5. Орієнтований перелік ілюстративного матеріалу діаграми, що демонструють роботу алгоритмів веб-інтерфейсу, архітектуру системи, бази даних та сховища файлів, взаємодію користувачів із системою, класи програмного забезпечення; приклади роботи з програмним продуктом.

6. Дата видачі завдання «15» вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі	17-20.04.24	
3.	Розробка архітектури та загальної структури системи	21-25.04.24	
4.	Розробка частин окремих підсистем	25.04-02.05.24	
5.	Програмна реалізація системи	03-07.05.24	
6.	Оформлення пояснювальної записки	08-12.05.24	
7.	Захист програмного продукту	13-17.05.24	
8.	Передзахист	03-06.06.24	
9.	Подання готової роботи на кафедру	07.06.2024	
10.	Захист	17-21.06.2024	

Студент _____ **Анжела НЕРУХ**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ **Андрій ОНИСЬКО**
(підпис) (ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 75 сторінках, містить 22 ілюстрацій, 11 таблиць, 1 додаток, 62 джерел в переліку посилань.

Мета роботи – створення веб-застосунку з контрольованим доступом для зберігання та керування файлами.

Методи та засоби Методи та засоби: Фронтенд розроблений з використанням Next.js для забезпечення серверного рендерингу і оптимізації продуктивності. Бекенд реалізовано на основі Nest.js для створення масштабованої і модульної архітектури. База даних MySQL з використанням ORM TypeORM для взаємодії з даними. Хмарні технології AWS S3 для зберігання файлів з високою доступністю і надійністю. Авторизація та аутентифікація користувачів, реалізовані за допомогою JWT (JSON Web Token). Оптимізація запитів за допомогою useSWR для покращення продуктивності фронтенду**.**

Результат – розроблений веб-застосунок з контрольованим доступом, який дозволяє користувачам зберігати, керувати та спільно використовувати файли. Застосунок забезпечує безпечний доступ до файлів, завантажених до хмарного сховища AWS S3, і має гнучку систему ролей та прав доступу.

Ключові слова: ВЕБ-ЗАСТОСУНОК, КОНТРОЛЬ ДОСТУПУ, NEXT.JS, NEST.JS, AWS S3, ФАЙЛОВЕ СХОВИЩЕ, ХМАРНІ ТЕХНОЛОГІЇ.

ABSTRACT

The qualification work includes an explanatory note (72 p., 22 fig., 11 tables).

Objective – to develop a web application with controlled access for file storage and management.

Methods and tools: the frontend is developed using Next.js to ensure server-side rendering and optimize performance, the backend is implemented based on Nest.js to create a scalable and modular architecture, the database is MySQL with the use of ORM TypeORM for data interaction, cloud technologies AWS S3 are used for file storage with high availability and reliability, user authorization and authentication are implemented using JWT (JSON Web Token), query optimization is achieved using useSWR to enhance frontend performance.

Result – a web application with controlled access that allows users to store, manage, and share files. The application provides secure access to files uploaded to the AWS S3 cloud storage and has a flexible system of roles and access rights.

Keywords: WEB APPLICATION, ACCESS CONTROL, NEXT.JS, NEST.JS, AWS S3, FILE STORAGE, CLOUD TECHNOLOGIES.

ЗМІСТ

ВСТУП

У сучасному світі цифрових технологій та глобальної комунікації зберігання та обмін файлами є невід'ємною частиною повсякденного життя та роботи. Все більше людей та організацій потребують надійних та безпечних рішень для зберігання, доступу та управління своїми файлами у будь-який час та з будь-якого місця. Традиційні методи зберігання файлів на локальних пристроях або переносних носіях вже не відповідають вимогам сучасності, оскільки вони обмежені фізичним доступом та ризикують втратою даних через апаратні збої або людські помилки.

Веб-сервіси зберігання файлів стали популярним рішенням для задоволення цих потреб, надаючи користувачам можливість зберігати свої файли у хмарному сховищі та отримувати до них доступ через мережу Інтернет. Однак, зі зростанням популярності таких сервісів, питання безпеки та конфіденційності даних стають все більш актуальними. Користувачі хочуть бути впевненими, що їхні файли захищені від несанкціонованого доступу та що тільки авторизовані особи можуть переглядати або змінювати їх вміст.

Саме тому розробка веб-сервісу зберігання файлів з авторизованим доступом є актуальною темою дослідження. Такий сервіс повинен забезпечувати не тільки зручність використання та доступність файлів, але й надійний захист конфіденційних даних користувачів. Реалізація механізмів автентифікації, авторизації та розмежування доступу є ключовими аспектами безпеки веб-сервісу, що гарантують, що лише авторизовані користувачі можуть отримати доступ до файлів та виконувати дозволені операції.

Окрім безпеки, важливими факторами при розробці веб-сервісу зберігання файлів є продуктивність, масштабованість та зручність використання. Сервіс повинен забезпечувати швидке та надійне завантаження, зберігання та скачування файлів, навіть при високому навантаженні та великій кількості користувачів. Він

також повинен мати інтуїтивно зрозумілий та привабливий інтерфейс користувача, що дозволяє легко керувати файлами, папками та правами доступу.

Дослідження та розробка веб-сервісу зберігання файлів з авторизованим доступом є актуальними не лише з точки зору технічної реалізації, але й з точки зору вивчення та застосування кращих практик та стандартів у галузі веб-розробки та безпеки. Це дослідження може стати основою для подальшого розвитку та вдосконалення подібних сервісів, а також для навчання та обміну досвідом серед розробників та дослідників у цій сфері.

Метою дослідження є розробка веб-сервісу зберігання файлів з авторизованим доступом, який забезпечує безпечне та зручне зберігання, доступ та управління файлами для користувачів.

Для досягнення поставленої мети були визначені наступні завдання:

1. Проаналізувати існуючі рішення для зберігання файлів з авторизованим доступом та визначити їх переваги та недоліки.
2. Розробити архітектуру веб-сервісу, яка забезпечує масштабованість, продуктивність та безпеку.
3. Спроектувати та реалізувати базу даних для зберігання інформації про файли та користувачів.
4. Розробити та реалізувати алгоритми автентифікації, авторизації та розмежування доступу до файлів.
5. Розробити зручний та привабливий інтерфейс користувача для взаємодії з веб-сервісом.
6. Реалізувати функціонал завантаження, зберігання та скачування файлів з авторизованим доступом.
7. Провести тестування веб-сервісу та виправити виявлені помилки та недоліки.
8. Розгорнути веб-сервіс на сервері та налаштувати його роботу.
9. Створити документацію та інструкції для користувачів веб-сервісу.

Розроблений веб-сервіс зберігання файлів з авторизованим доступом може бути використаний як готове рішення для зберігання та обміну файлами у різних сферах діяльності, таких як освіта, бізнес, державне управління тощо. Він надає

користувачам безпечне та зручне середовище для роботи з файлами, забезпечуючи конфіденційність та цілісність даних. Веб-сервіс може бути інтегрований з іншими системами та додатками, розширюючи його функціональність та можливості застосування.

Припускається, що розробка веб-сервісу зберігання файлів з авторизованим доступом, який базується на сучасних технологіях та стандартах безпеки, дозволить забезпечити надійне та зручне зберігання та обмін файлами для користувачів, зберігаючи конфіденційність та цілісність даних.

Новизна роботи полягає у створенні оригінальної архітектури веб-сервісу зберігання файлів з авторизованим доступом, яка поєднує у собі передові технології та підходи до розробки веб-сервісів та систем безпеки. Запропоновані рішення та алгоритми автентифікації, авторизації та розмежування доступу до файлів є удосконаленими та адаптованими до специфічних потреб веб-сервісу зберігання файлів. Розроблений інтерфейс користувача відрізняється високою зручністю використання та привабливим дизайном, що підвищує якість користувацького досвіду.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

Основна задача дипломної роботи – розробити веб-застосунок, який забезпечує надійне та безпечне зберігання файлів з авторизованим доступом. Це включає в себе можливість завантаження файлів користувачами, організацію файлів у структуру папок, налаштування доступу до файлів та папок для різних користувачів, а також можливість спільного використання файлів з контролем доступу.

Застосунок призначений для користувачів, які потребують централізованого зберігання та управління документами. Це можуть бути компанії будь-якого масштабу, які працюють з великою кількістю документів та потребують забезпечення їх конфіденційності та безпеки. Застосунок також буде корисним для організацій, які ведуть спільну роботу над проектами і потребують зручного інструменту для обміну та спільного редагування файлів.

1.1 Огляд існуючих рішень для зберігання файлів з авторизованим доступом

У сучасному світі інформаційних технологій зберігання та обмін файлами є невід'ємною частиною роботи багатьох організацій та індивідуальних користувачів. Для забезпечення безпеки та конфіденційності даних необхідно використовувати рішення, які дозволяють здійснювати авторизований доступ до файлів. Одним із таких рішень є використання хмарних сховищ, які надають можливість зберігати файли на віддалених серверах та отримувати до них доступ через Інтернет.

Хмарні сховища, такі як Google Drive, Dropbox та OneDrive, пропонують користувачам зручний інтерфейс для завантаження, зберігання та обміну файлами.

Ці сервіси забезпечують високий рівень безпеки завдяки використанню шифрування даних та можливості налаштування прав доступу до файлів для різних користувачів. Крім того, хмарні сховища дозволяють синхронізувати файли між різними пристроями, що значно спрощує роботу з документами та забезпечує їх доступність з будь-якого місця.

Іншим рішенням для зберігання файлів з авторизованим доступом є використання корпоративних систем управління контентом (Enterprise Content Management, ECM) [6, с. 56]. ECM-системи призначені для централізованого зберігання, управління та обміну документами в рамках організації. Вони забезпечують розмежування доступу до файлів на основі ролей користувачів та дозволяють здійснювати контроль версій документів.

Прикладами ECM-систем є Microsoft SharePoint, Alfresco та OpenText. Ці системи надають можливість створювати репозиторії документів, налаштовувати робочі процеси (workflows) для автоматизації бізнес-процесів та забезпечувати колективну роботу над документами. ECM-системи також підтримують інтеграцію з іншими корпоративними додатками, такими як системи управління взаємовідносинами з клієнтами (CRM) та системи планування ресурсів підприємства (ERP).

Ще одним рішенням для зберігання файлів з авторизованим доступом є використання систем контролю версій, таких як Git та Subversion. Ці системи дозволяють зберігати та відстежувати зміни в файлах, забезпечуючи можливість співпраці над проектами для команд розробників. Системи контролю версій надають механізми розмежування доступу до репозиторіїв та гілок коду, що дозволяє контролювати, хто має право вносити зміни та переглядати файли [2, с. 19].

Для забезпечення додаткового рівня безпеки при зберіганні файлів з авторизованим доступом можуть використовуватися технології шифрування, такі як AES (Advanced Encryption Standard) та RSA (Rivest-Shamir-Adleman). Шифрування дозволяє захистити конфіденційність даних навіть у випадку несанкціонованого доступу до файлів. Крім того, для підвищення безпеки можуть

застосовуватися механізми двофакторної автентифікації, які вимагають від користувачів надання додаткового підтвердження їх особистості, наприклад, введення одноразового коду з SMS або використання біометричних даних.

При виборі рішення для зберігання файлів з авторизованим доступом важливо враховувати специфічні потреби організації або індивідуального користувача. Слід звертати увагу на такі фактори, як обсяг даних, що зберігаються, необхідність інтеграції з існуючими системами, вимоги до безпеки та конфіденційності, а також зручність використання та доступність сервісу.

Отже, існує декілька рішень для зберігання файлів з авторизованим доступом, кожне з яких має свої переваги та недоліки. Хмарні сховища забезпечують зручність доступу та синхронізації файлів, але можуть мати обмеження щодо конфіденційності даних. ECM-системи надають потужні можливості для управління документами в корпоративному середовищі, але вимагають значних ресурсів для впровадження та підтримки. Системи контролю версій є ефективними для розробки програмного забезпечення, але можуть бути менш зручними для зберігання інших типів файлів. Тому при виборі рішення необхідно ретельно проаналізувати вимоги та врахувати специфіку конкретного проекту або організації.

1.2 Аналіз вимог до веб-сервісу зберігання файлів

Для створення ефективного веб-сервісу зберігання файлів з авторизованим доступом необхідно провести ґрунтовний аналіз вимог до такого сервісу. Цей аналіз дозволить визначити ключові функціональні та нефункціональні характеристики, які повинен мати веб-сервіс для задоволення потреб користувачів та забезпечення надійності й безпеки зберігання даних.

Одним з основних функціональних вимог до веб-сервісу зберігання файлів є можливість завантаження та скачування файлів користувачами. Сервіс повинен надавати зручний та інтуїтивно зрозумілий інтерфейс для завантаження файлів на

сервер та їх подальшого скачування. Крім того, важливо забезпечити підтримку різних форматів файлів, таких як документи, зображення, аудіо та відео [8, с. 39].

Іншою ключовою вимогою є реалізація системи авторизації та аутентифікації користувачів. Веб-сервіс повинен забезпечувати можливість реєстрації нових користувачів та їх автентифікацію при вході в систему. Це дозволить контролювати доступ до файлів та забезпечити конфіденційність даних користувачів. Автентифікація може здійснюватися за допомогою логіна та пароля, а також з використанням додаткових факторів, таких як одноразові коди або біометричні дані.

Для забезпечення безпеки зберігання файлів веб-сервіс повинен використовувати надійні методи шифрування даних. Шифрування повинно здійснюватися як при передачі файлів через мережу, так і при їх зберіганні на сервері. Це допоможе захистити конфіденційність інформації навіть у випадку несанкціонованого доступу до серверу або перехоплення даних під час передачі.

Важливою вимогою до веб-сервісу є можливість розмежування доступу до файлів на основі ролей та прав користувачів [3, с. 61]. Сервіс повинен дозволяти власнику файлу визначати, хто має право переглядати, редагувати або видаляти файл. Це може бути реалізовано за допомогою системи розподілу прав доступу, де кожному користувачу або групі користувачів призначаються певні привілеї щодо конкретних файлів або папок.

Для зручності користувачів веб-сервіс повинен надавати можливість організації файлів у директорії. Це дозволить структурувати інформацію та полегшити навігацію по файловій системі. Крім того, сервіс може надавати функції пошуку файлів за назвою, типом або вмістом, що значно спростить роботу з великою кількістю документів.

Важливо також врахувати вимоги до зручності використання веб-сервісу. Інтерфейс користувача повинен бути інтуїтивно зрозумілим та легким у навігації. Сервіс повинен надавати чіткі інструкції та підказки для виконання основних операцій, таких як завантаження файлів, створення папок та надання доступу іншим користувачам. Крім того, веб-сервіс повинен бути адаптивним та коректно

відображатися на різних пристроях, включаючи настільні комп'ютери, ноутбуки, планшети та смартфони [2, с. 23].

Для забезпечення конфіденційності даних користувачів веб-сервіс повинен відповідати вимогам законодавства щодо захисту персональних даних, таким як GDPR (General Data Protection Regulation) в Європейському Союзі. Це передбачає забезпечення можливості доступу та видалення даних на вимогу користувача, а також дотримання принципів мінімізації та цільового використання даних.

Для забезпечення безпеки веб-сервісу необхідно регулярно проводити аудит безпеки та тестування на проникнення. Це допоможе виявити потенційні вразливості та слабкі місця в системі, які можуть бути використані зловмисниками. Крім того, важливо своєчасно встановлювати оновлення безпеки та виправляти виявлені недоліки.

Отже, аналіз вимог до веб-сервісу зберігання файлів з авторизованим доступом є важливим етапом у процесі розробки такого сервісу. Врахування функціональних та нефункціональних вимог, таких як можливість завантаження та скачування файлів, авторизація користувачів, шифрування даних, розмежування доступу, організація файлів у папки, продуктивність, масштабованість, зручність використання, конфіденційність даних, безпека та документація, дозволить створити надійний та ефективний веб-сервіс, який відповідатиме потребам користувачів та забезпечить безпечне зберігання та обмін файлами.

1.3 Формулювання мети та завдань розробки веб-сервісу

Формулювання мети та завдань розробки веб-сервісу зберігання файлів з авторизованим доступом є ключовим етапом у процесі планування та реалізації проекту. Чітке визначення мети дозволяє зосередити зусилля на досягненні конкретних результатів та забезпечує розуміння кінцевого продукту, який повинен бути створений. Мета розробки веб-сервісу зберігання файлів полягає у створенні

надійної та безпечної платформи для зберігання, обміну та управління файлами з можливістю авторизованого доступу користувачів.

Для досягнення поставленої мети необхідно визначити конкретні завдання, які потрібно виконати в процесі розробки веб-сервісу. Одним з основних завдань є проектування та реалізація системи авторизації та автентифікації користувачів. Це завдання передбачає створення механізмів реєстрації користувачів, їх ідентифікації при вході в систему та забезпечення безпечного доступу до файлів на основі призначених прав та ролей.

Іншим важливим завданням є розробка зручного та інтуїтивно зрозумілого інтерфейсу користувача для взаємодії з веб-сервісом [15, с. 48]. Інтерфейс повинен надавати користувачам можливість легко завантажувати, скачувати та управляти файлами, створювати папки та надавати доступ іншим користувачам. Крім того, інтерфейс має бути адаптивним та коректно відображатися на різних пристроях, таких як настільні комп'ютери, ноутбуки, планшети та смартфони.

Забезпечення безпеки зберігання та передачі файлів є ще одним ключовим завданням при розробці веб-сервісу. Це завдання включає в себе реалізацію надійних методів шифрування даних, як при зберіганні файлів на сервері, так і при їх передачі через мережу. Шифрування гарантує конфіденційність інформації та захищає її від несанкціонованого доступу.

Для забезпечення ефективного управління файлами та зручності користувачів необхідно реалізувати функціональність організації файлів у папки та підпапки [19, с. 105]. Це завдання передбачає створення ієрархічної структури зберігання файлів, яка дозволить користувачам логічно групувати та впорядковувати свої документи. Також важливо реалізувати можливість пошуку файлів за різними критеріями, такими як назва, тип або вміст, що значно спростить роботу з великою кількістю файлів.

Забезпечення відповідності веб-сервісу законодавчим вимогам щодо захисту персональних даних є важливим завданням для дотримання конфіденційності інформації користувачів. Це завдання включає реалізацію механізмів отримання згоди користувачів на обробку їх персональних даних, забезпечення можливості

доступу та видалення даних на вимогу користувача, а також дотримання принципів мінімізації та цільового використання даних відповідно до законодавства, такого як GDPR [20, с. 57].

Проведення регулярних аудитів безпеки та тестування на проникнення є важливим завданням для підтримки високого рівня захищеності веб-сервісу. Це завдання передбачає проведення комплексного аналізу безпеки системи, виявлення потенційних вразливостей та слабких місць, а також своєчасне встановлення оновлень безпеки та виправлення виявлених недоліків. Регулярний аудит безпеки дозволяє мінімізувати ризики несанкціонованого доступу та захистити дані користувачів.

Створення детальної документації та інструкцій для користувачів є важливим завданням для забезпечення ефективного використання веб-сервісу . Це завдання включає розробку користувацької документації, яка містить опис функціональності сервісу, інструкції щодо виконання основних операцій, інформацію про політику конфіденційності та умови використання. Наявність чіткої та зрозумілої документації допомагає користувачам швидко освоїти роботу з веб-сервісом та отримати відповіді на поширені запитання [14, с. 103].

Отже, формулювання мети та завдань розробки веб-сервісу зберігання файлів з авторизованим доступом є фундаментальним етапом у процесі створення ефективної та надійної платформи. Чітке визначення мети, яка полягає у розробці безпечного та зручного сервісу для зберігання та обміну файлами, а також встановлення конкретних завдань, таких як реалізація системи авторизації, розробка інтуїтивно зрозумілого інтерфейсу, забезпечення безпеки даних, організація файлів у папки, колективна робота, продуктивність та масштабованість, резервне копіювання, інтеграція з іншими системами, відповідність законодавчим вимогам, аудит безпеки, створення документації та підтримка користувачів, а також планування маркетингової стратегії, дозволяє структурувати процес розробки та забезпечити успішну реалізацію веб-сервісу, який відповідатиме потребам та очікуванням користувачів.

1.4 Вибір інструментів та технологій для розробки веб-сервісу

Вибір інструментів та технологій для розробки веб-сервісу зберігання файлів з авторизованим доступом є важливим етапом, який впливає на ефективність, продуктивність та надійність кінцевого продукту. Правильний вибір технологічного стеку дозволяє оптимізувати процес розробки, забезпечити необхідну функціональність та масштабованість веб-сервісу. При виборі інструментів та технологій необхідно враховувати специфіку проекту, вимоги до продуктивності, безпеки та зручності використання, а також досвід та навички команди розробників.

Для клієнтської частини веб-сервісу, яка відповідає за взаємодію з користувачем через веб-браузер, зазвичай використовуються технології HTML, CSS та JavaScript. HTML (HyperText Markup Language) використовується для структурування та розмітки веб-сторінок, CSS (Cascading Style Sheets) - для стилізації та візуального оформлення, а JavaScript - для додавання інтерактивності та динамічної поведінки веб-сторінок. Також популярними є фреймворки та бібліотеки, такі як React, Next.js, Angular та Vue.js, які спрощують розробку клієнтської частини та забезпечують створення інтерактивних та респонсивних інтерфейсів користувача [14, с. 98].

Таблиця 1.4.1 - Порівняння UI-фреймворків для розробки веб-інтерфейсів

Фреймворк	**Популярність**	**Розмір**	**Продуктивність**	**Крива навчанн**
React	Висока	Середній	Висока	Середня
Angular	Середня	Великий	Середня	Висока
Vue.js	Висока	Малий	Висока	Низька
Bootstrap	Висока	Середній	Середня	Низька
Material UI	Середня	Середній	Середня	Середня

Для зберігання та управління даними веб-сервісу необхідно обрати відповідну базу даних. Популярними рішеннями є реляційні бази даних, такі як MySQL, PostgreSQL та Oracle, а також нереляційні бази даних, такі як MongoDB,

Cassandra та Redis. Вибір бази даних залежить від структури даних, які потрібно зберігати, масштабованості та продуктивності, яка необхідна для веб-сервісу. Для зберігання структурованих даних та забезпечення цілісності даних часто використовуються найкраще підходять реляційні бази даних.

Таблиця 1.4.2 - Порівняння реляційних та нереляційних баз даних

Критерій	**Реляційні бази даних (SQL)**	**Нереляційні бази даних (NoSQL)**
Модель даних	Структурована, на основі таблиць та зв'язків	Гнучка, на основі колекцій, документів, ключ-значення
Схема даних	Фіксована, визначається заздалегідь	Динамічна, може змінюватися під час роботи
Масштабованість	Вертикальна (збільшення потужності сервера)	Горизонтальна (додавання нових серверів)
Продуктивність	Оптимізована для складних запитів та транзакцій	Висока для простих операцій читання/запису
Транзакції	Підтримуються ACID-транзакції	Обмежена підтримка транзакцій
Гнучкість	Менша гнучкість через фіксовану схему	Більша гнучкість через динамічну схему

Для забезпечення безпеки веб-сервісу необхідно використовувати надійні інструменти та технології. Одним з важливих аспектів безпеки є автентифікація та авторизація користувачів. Для цього можуть використовуватися протоколи та стандарти, такі як OAuth 2.0 та JWT (JSON Web Token). OAuth 2.0 - це протокол авторизації, який дозволяє користувачам надавати доступ до своїх ресурсів стороннім додаткам без необхідності передавати свої облікові дані. JWT - це стандарт для створення токенів доступу, які містять інформацію про автентифікацію та авторизацію користувача.

Для захисту даних під час передачі через мережу необхідно використовувати протокол HTTPS (HTTP Secure), який забезпечує шифрування даних за допомогою SSL/TLS [19, с. 87]. Це гарантує конфіденційність та цілісність даних при обміні

між клієнтом та сервером. Для зберігання конфіденційної інформації, такої як паролі користувачів, рекомендується використовувати надійні алгоритми хешування, такі як bcrypt або Argon2, які захищають дані від несанкціонованого доступу навіть у випадку компрометації бази даних.

Таблиця 1.4.3 - Порівняння алгоритмів хешування паролів

Алгоритм	**Швидкість**	**Безпека**	**Використання пам'яті**
MD5	Висока	Низька	Низьке
SHA-256	Середня	Середня	Середнє
BCrypt	Низька	Висока	Високе
Argon2	Низька	Висока	Високе

Для розгортання веб-сервісу необхідно обрати відповідну інфраструктуру та платформу. Хмарні платформи, такі як Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP), надають широкі можливості для розгортання та масштабування веб-застосунків [11, с. 46]. Ці платформи пропонують послуги, такі як віртуальні машини, контейнери, бази даних, системи зберігання даних та інструменти для моніторингу та управління інфраструктурою. Використання хмарних платформ дозволяє зменшити витрати на обладнання та забезпечити гнучкість та масштабованість веб-сервісу.

Для автоматизації процесу розгортання та управління інфраструктурою веб-сервісу можуть використовуватися інструменти інфраструктури як коду (Infrastructure as Code - IaC), такі як Terraform, AWS CloudFormation та Ansible . Ці інструменти дозволяють описувати інфраструктуру у вигляді коду, що забезпечує відтворюваність, версіонування та автоматизацію процесу розгортання. Використання IaC дозволяє зменшити ризики помилок при налаштуванні інфраструктури та спростити управління великомасштабними системами [14, с. 94].

При виборі інструментів та технологій для розробки веб-сервісу також важливо враховувати наявність та активність спільноти розробників, документацію

та підтримку. Наявність великої та активної спільноти розробників означає доступність ресурсів, бібліотек, плагінів та можливість отримати допомогу та поради від інших розробників. Якісна документація та підтримка від розробників інструментів та технологій спрощує процес навчання та вирішення проблем під час розробки.

Окрім вибору основних інструментів та технологій, важливо також розглянути допоміжні бібліотеки та фреймворки, які можуть спростити та прискорити розробку веб-сервісу. Наприклад, для роботи з базами даних можуть використовуватися ORM (Object-Relational Mapping) бібліотеки, такі як Typeorm або Prizma, які дозволяють абстрагуватися від низькорівневих операцій з базою даних та працювати з об'єктами.

Не менш важливим фактором є безпека обраних інструментів та технологій. Веб-сервіс зберігання файлів може містити конфіденційну інформацію користувачів, тому необхідно забезпечити захист від потенційних вразливостей та атак. Обрані інструменти та технології повинні мати вбудовані механізми безпеки, регулярні оновлення безпеки та можливість налаштування додаткових заходів безпеки, таких як двофакторна аутентифікація та шифрування даних.

Отже, вибір інструментів та технологій для розробки веб-сервісу зберігання файлів з авторизованим доступом є комплексним процесом, який вимагає врахування багатьох факторів. Необхідно враховувати специфіку проекту, вимоги до функціональності, продуктивності та безпеки, досвід та навички команди розробників, наявність спільноти та підтримки, ліцензування та вартість, можливості інтеграції та масштабування. Правильний вибір інструментів та технологій дозволяє оптимізувати процес розробки, забезпечити необхідну функціональність та надійність веб-сервісу, а також закласти основу для подальшого розвитку та масштабування системи.

2 ПРОЕКТУВАННЯ ВЕБ-СЕРВІСУ ЗБЕРІГАННЯ ФАЙЛІВ

З розвитком інформаційних технологій та збільшенням обсягів цифрових даних питання безпечного зберігання файлів та спільного доступу до них стає все більш актуальним. Сучасні підприємства стикаються з низкою викликів, пов'язаних з управлінням конфіденційною інформацією, забезпеченням її доступності для співробітників та захистом від несанкціонованого доступу.

Однією з ключових проблем є забезпечення надійного зберігання файлів. Зберігання даних у локальних системах не завжди є ефективним рішенням через ризики втрати даних у разі технічних несправностей або фізичних пошкоджень обладнання. Використання хмарних сховищ, таких як AWS S3, дозволяє значно підвищити надійність та доступність даних, але при цьому виникають питання безпеки та конфіденційності інформації.

Таким чином, розробка веб-застосунку для безпечного зберігання файлів та організації спільного доступу є актуальною задачею, яка потребує комплексного підходу та використання сучасних технологій для забезпечення надійності, безпеки та зручності використання. У наступних підпунктах буде розглянуто основні аспекти цієї проблеми та запропоновані рішення для їх подолання.

2.1 Розробка архітектури веб-сервісу

Розробка архітектури веб-сервісу є ключовим етапом у процесі створення веб-сервісу зберігання файлів з авторизованим доступом. Архітектура визначає структуру та організацію компонентів веб-сервісу, їх взаємодію та принципи роботи. Правильно спроектована архітектура забезпечує масштабованість, продуктивність, безпеку та можливість подальшого розвитку веб-сервісу.

Для веб-сервісу зберігання файлів з авторизованим доступом доцільно використовувати архітектуру, засновану на принципах REST (Representational State

Transfer). REST - це архітектурний стиль, який визначає набір обмежень та властивостей для побудови розподілених систем. RESTful веб-сервіси використовують протокол HTTP для комунікації та маніпулювання ресурсами через стандартні методи, такі як GET, POST, PUT та DELETE [23, с. 112].

Архітектура веб-сервісу складається з таких основних компонентів:

1. Клієнтська частина (Front-end): це інтерфейс користувача, з яким взаємодіє клієнт через веб-браузер. Клієнтська частина відповідає за відображення даних, отриманих від серверної частини, та надсилання запитів на сервер.
2. Серверна частина (Back-end): це основна логіка веб-сервісу, яка обробляє запити від клієнтської частини, виконує операції з базою даних та надає відповіді у форматі JSON. Серверна частина складається з таких компонентів:
   - Веб-сервер: обробляє HTTP-запити та маршрутизує їх до відповідних контролерів.
   - Контролери: містять бізнес-логіку обробки запитів та формування відповідей.
   - Сервіси: інкапсулюють окремі функціональні можливості та операції з даними.
   - Репозиторії: забезпечують доступ до бази даних та виконують операції з даними.
1. База даних: зберігає інформацію про файли, користувачів та інші дані, необхідні для роботи веб-сервісу. Використовується реляційна база даних MySQL [29, с. 57].

Для автоматизації розгортання та управління інфраструктурою веб-сервісу використовується, Docker. Docker дозволяє створювати контейнери з додатком та його залежностями, що спрощує розгортання та масштабування веб-сервісу.[29, с. 83]. Docker Compose використовує YAML-файл для визначення багатоконтейнерної програми. У цьому файлі вказуються служби, мережі та томи, необхідні для програми.

Рисунок 2.1.1 – Docker compose файл

Наданий файл Docker Compose визначає налаштування для запуску контейнера бази даних MariaDB. Файл починається із зазначення версії Docker Compose, яка використовується, а саме версії 3.5. У розділі services визначено єдиний сервіс з назвою «db». Ця служба використовує образ MariaDB, зокрема версію 10.6.15, а контейнер має назву «filesync-mariadb». У розділі ports порт 3308 на хост-машині зіставляється з портом 3306 у контейнері, що дозволяє зовнішній доступ до служби MariaDB.

Змінні оточення використовуються для налаштування бази даних: MYSQL_ROOT_PASSWORD, надаючи користувачеві root пароль; MYSQL_DATABASE, що визначає ім'я бази даних за замовчуванням, яку буде створено; MYSQL_USER, а MYSQL_PASSWORD, створюючи нового користувача з цими обліковими даними, який має доступ до вказаної бази даних.

Для збереження даних секція volumes пов'язує іменований том .data з каталогом даних контейнера за адресою /var/lib/mysql, гарантуючи, що дані бази даних зберігатимуться поза контейнером, що зробить їх незмінними після перезапуску контейнера. І останнє, розділ volumes визначає том .data, який Docker Compose створить і яким буде керувати.

Таке налаштування дозволяє легко розгортати базу даних MariaDB з попередньо визначеними обліковими даними користувачів та базою даних, що робить її придатною для середовищ розробки та тестування.

Для забезпечення безпеки зберігання файлів використовується шифрування даних на стороні клієнта перед їх надсиланням на сервер. Це гарантує, що навіть у разі компрометації сервера, зловмисники не зможуть отримати доступ до конфіденційних даних користувачів. Для шифрування була використана бібліотека cryptoJs та алгоритм AES (Advanced Encryption Standard).

```
export function encryptFile(file: Blob, secretKey: string) {
  const reader = new FileReader()

  return new Promise((resolve, reject) => {
    reader.onload = () => {
      if (reader.result instanceof ArrayBuffer) {
        const wordArray = CryptoJS.lib.WordArray.create(reader.result)
        const encrypted = CryptoJS.AES.encrypt(wordArray, secretKey).toString()
        resolve(encrypted)
      } else {
        reject(new Error('File reading failed'))
      }
    }
    reader.onerror = error => {
      reject(error)
    }
    reader.readAsArrayBuffer(file)
  })
}
```

Рисунок 2.1.2 – Шифрування файлу

Для документування архітектури та функціональності веб-сервісу використовується Swagger. Цей інструменти дозволяє створювати інтерактивну документацію API, яка описує доступні ендпоінти, параметри запитів та формати відповідей. Використана бібліотека reflect-metadata для автоматичного генерування документації.

Рисунок 2.1.3 – Swagger налаштування

У цьому налаштуванні спочатку створюється екземпляр DocumentBuilder для налаштування документації Swagger. Цей екземпляр включає аутентифікацію на основі файлів cookie за допомогою addCookieAuth і аутентифікацію за допомогою токенів пред'явника за допомогою addBearerAuth, що дозволяє користувачам автентифікуватися за допомогою файлів cookie і JWT (JSON Web Tokens). Назва документації API встановлюється як «FileSync» за допомогою setTitle, а опис надається як “The FileSync API description” за допомогою функції setDescription. Після цього конфігурація фіналізується і збирається за допомогою методу build. Наявність чіткої та актуальної документації полегшує інтеграцію з веб-сервісом для розробників та користувачів.

Отже, розробка архітектури веб-сервісу зберігання файлів з авторизованим доступом вимагає ретельного планування та врахування різних аспектів, таких як вибір архітектурного стилю, розподіл компонентів, забезпечення безпеки, масштабованості, продуктивності та зручності використання. Правильно спроектована архітектура закладає міцний фундамент для успішної реалізації та подальшого розвитку веб-сервісу.

2.2 Проектування бази даних для зберігання інформації про файли та користувачів

Проектування бази даних є важливим етапом у розробці веб-сервісу зберігання файлів з авторизованим доступом. База даних забезпечує надійне та ефективне зберігання інформації про файли, користувачів та інші дані, необхідні для роботи веб-сервісу [30, с. 162]. Правильно спроектована база даних дозволяє оптимізувати продуктивність, забезпечити цілісність даних та спростити розробку та супровід веб-сервісу.

Для зберігання інформації про файли використовується реляційна база даних MySQL. Реляційні бази даних забезпечують структуроване зберігання даних у таблицях, зв'язаних між собою за допомогою ключів. Вони підтримують транзакції, забезпечують цілісність даних за допомогою обмежень та дозволяють виконувати складні запити за допомогою мови SQL (Structured Query Language).

Для забезпечення цілісності даних використовується зовнішні ключі (foreign keys) між таблицями [23, с. 174]. Зовнішні ключі дозволяють встановлювати зв'язки між записами в різних таблицях та гарантують, що дані в залежних таблицях відповідають даним у головних таблицях. Наприклад, зовнішній ключ user_id у таблиці "Файли" гарантує, що файл належить існуючому користувачу в таблиці "Користувачі".

Таблиця "Користувачі" (users)

- **id**: INT, AUTO_INCREMENT, PRIMARY KEY – Унікальний ідентифікатор користувача.
- **firstname**: VARCHAR(50), NOT NULL – Ім'я користувача.
- **lastname**: VARCHAR(50), NOT NULL – Прізвище користувача.
- **fullname**: VARCHAR(50), NOT NULL – Повне ім'я користувача.
- **email**: VARCHAR(100), NOT NULL, UNIQUE – Унікальна електронна пошта.
- **password**: VARCHAR(255), NOT NULL – Хешований пароль.
- **refresh_token**: VARCHAR(255), DEFAULT NULL – Токен для оновлення сесії.
- **space**: BIGINT, DEFAULT 15000000 – Простір у байтах.

- **created_at**: TIMESTAMP, DEFAULT CURRENT_TIMESTAMP – Час створення.
- **updated_at**: TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP – Час останнього оновлення.

Таблиця "files" є важливою складовою системи зберігання файлів, яка містить метадані про кожний файл, що зберігається у системі. Ось короткий опис колонок таблиці "files":

- **id**: ця колонка містить унікальний ідентифікатор кожного файлу. Тип даних - INT з обмеженнями AUTO_INCREMENT і PRIMARY KEY, що гарантує унікальність кожного запису.
- **user_id**: ідентифікатор користувача, якому належить файл. Це зовнішній ключ, що посилається на колонку "id" в таблиці "Users". Тип даних - INT з обмеженням NOT NULL.
- **folder_id**: ідентифікатор папки, що містить файл. Це зовнішній ключ, що посилається на колонку "id" в таблиці "Folders". Тип даних - INT.
- **name**: назва файлу. Тип даних - VARCHAR(255) з обмеженням NOT NULL, що забезпечує обов'язкову наявність назви файлу.
- **url**: унікальний ключ S3 для зберігання файлу. Тип даних - VARCHAR(255) з обмеженнями NOT NULL і UNIQUE, що гарантує унікальність кожного файлу.
- **path**: розташування файлу. Тип даних - JSON з дефолтним значенням NULL.
- **summary**: короткий опис файлу. Тип даних - VARCHAR(500) з дефолтним значенням NULL.
- **keywords**: мета дані файлу. Тип даних - JSON з дефолтним значенням NULL.
- **is_file**: вказує, чи це файл. Тип даних - BOOLEAN з дефолтним значенням TRUE.
- **mime_type**: MIME тип файлу. Тип даних - VARCHAR(100) з обмеженням NOT NULL.
- **size**: розмір файлу у байтах. Тип даних - BIGINT з обмеженням NOT NULL.
- **is_public**: вказує, чи є файл публічним. Тип даних - BOOLEAN з дефолтним значенням FALSE.

- **created_at**: час створення файлу. Тип даних - TIMESTAMP з дефолтним значенням CURRENT_TIMESTAMP.
- **updated_at**: час останнього оновлення файлу. Тип даних - TIMESTAMP з дефолтним значенням CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP.

Ця структура забезпечує ефективне зберігання та керування файлами, підтримуючи належний рівень доступу та безпеки для кожного користувача і файлу в системі.

Таблиця "Папки" (folders)

- **id**: INT, AUTO_INCREMENT, PRIMARY KEY – унікальний ідентифікатор кожної папки.
- **user_id**: INT, NOT NULL, FOREIGN KEY (Users.id) – ідентифікатор користувача, якому належить папка.
- **path**: JSON, DEFAULT NULL – розташування файлу.
- **is_file**: BOOLEAN, DEFAULT FALSE – вказує, чи це файл.
- **parent_id**: INT, FOREIGN KEY (Folders.id) – ідентифікатор батьківської папки.
- **name**: VARCHAR(255), NOT NULL – назва папки.
- **created_at**: TIMESTAMP, DEFAULT CURRENT_TIMESTAMP – час створення папки.
- **updated_at**: TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP – час останнього оновлення папки.

Для моніторингу продуктивності та оптимізації роботи бази даних використовується інструмент профілювання та аналізу запитів MySQL Workbench. Цей інструмент дозволяє відстежувати час виконання запитів, виявляти повільні запити та оптимізувати їх за допомогою індексів або зміни структури бази даних[24, с. 111].

Для забезпечення можливості пошуку файлів за різними критеріями використовується повнотекстовий пошук. Повнотекстовий пошук дозволяє шукати файли за їх вмістом, використовуючи ключові слова або фрази.

Для забезпечення можливості обміну файлами між користувачами реалізована функціональність спільного доступу до файлів. Це дозволить користувачам надавати доступ до своїх файлів іншим користувачам з різними рівнями прав, такі як перегляд, видалення. Для реалізації спільного доступу використана таблиця «Права доступу».

Таблиця "Права доступу" (permissions) забезпечує управління доступом до файлів у системі. Вона дозволяє надавати різні рівні доступу до файлів для користувачів, тим самим підвищуючи гнучкість і безпеку системи. Це важливо для підтримки конфіденційності та контролю доступу до чутливих даних.

Опис структури таблиці "Права доступу" (permissions)

- **id**: INT, AUTO_INCREMENT, PRIMARY KEY – унікальний ідентифікатор кожного спільного доступу.
- **file_id**: INT, NOT NULL, FOREIGN KEY (Files.id) – ідентифікатор файлу, до якого надано спільний доступ.
- **user_id**: INT, NOT NULL, FOREIGN KEY (Users.id) – ідентифікатор користувача, якому надано дозвіл.
- **email**: VARCHAR(100), NOT NULL – електронна пошта особи, з якою поділилися файлом.
- **permission_type**: ENUM ('read', 'write') NOT NULL – тип наданого дозволу.
- **token**: VARCHAR(255), NOT NULL – унікальний токен для доступу.
- **token_expiry**: TIMESTAMP, NOT NULL – час закінчення дії токена.
- **created_at**: TIMESTAMP, DEFAULT CURRENT_TIMESTAMP – час створення спільного доступу.
- **updated_at**: TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP – час останнього оновлення спільного доступу.

Така структура дозволяє ефективно керувати правами доступу до файлів, забезпечуючи гнучкість у налаштуванні рівнів доступу і тимчасових обмежень. Це є важливим компонентом системи для підтримки безпеки і конфіденційності даних.

Отже, проектування бази даних для зберігання інформації про файли та користувачів є важливим етапом у розробці веб-сервісу зберігання файлів з

авторизованим доступом. Правильно спроектована база даних забезпечує ефективне зберігання, швидкий доступ та цілісність даних. Використання реляційної моделі бази даних з такими таблицями, як "Користувачі", "Файли", "Папки" та "Права доступу", дозволяє структурувати інформацію та встановлювати зв'язки між сутностям.

Для забезпечення цілісності даних у базі даних використовуються обмеження (constraints), такі як первинні та зовнішні ключі, унікальні індекси та перевірки. Це допомагає підтримувати узгодженість та правильність даних у таблицях.

При проектуванні бази даних для веб-сервісу зберігання файлів враховано можливість розширення функціональності в майбутньому. Використання гнучких схем даних та продуманої структури таблиць дозволяє легко додавати нові функції та атрибути без значних змін у базі даних.

2.3 Розробка алгоритму авторизації та аутентифікації користувачів

Розробка алгоритму авторизації та аутентифікації користувачів є критично важливим аспектом при створенні веб-сервісу зберігання файлів. Авторизація - це процес надання користувачу доступу до певних ресурсів або функціональних можливостей системи на основі його ідентифікації та прав доступу. Аутентифікація, в свою чергу, є процесом перевірки достовірності наданих користувачем облікових даних, щоб підтвердити його особу.

Для розробки ефективного та безпечного алгоритму авторизації та аутентифікації користувачів необхідно враховувати різні фактори, такі як конфіденційність даних, захист від несанкціонованого доступу, зручність використання та масштабованість системи [33, с. 112]. Одним із найпоширеніших підходів до реалізації авторизації та аутентифікації є використання токенів доступу, зокрема JWT (JSON Web Token) [34, с. 48].

JWT - це відкритий стандарт (RFC 7519) для створення токенів доступу, які містять інформацію про автентифікацію користувача у вигляді JSON-об'єкта .

Токен складається з трьох частин: заголовка (header), корисного навантаження (payload) та підпису (signature). Заголовок містить метадані про токен, такі як тип токена та алгоритм підпису. Корисне навантаження містить твердження (claims) про користувача, такі як його ідентифікатор, ім'я та ролі. Підпис використовується для перевірки цілісності та автентичності токена.

Процес авторизації та аутентифікації з використанням JWT відбувається наступним чином:

1. Користувач надсилає запит на аутентифікацію, надаючи свої облікові дані, а саме пошта та пароль.
2. Сервер перевіряє надані облікові дані та, у разі успішної аутентифікації, генерує JWT-токен.
3. Сервер надсилає згенерований JWT-токен клієнту у відповіді.
4. Клієнт зберігає отриманий JWT-токен у локальному сховищі браузера та додає його до кожного наступного запиту до сервера в заголовку Authorization.
5. Сервер отримує запит з JWT-токеном, перевіряє його підпис та, у разі успішної верифікації, надає доступ до запитуваних ресурсів або функціональних можливостей.

Окрім використання JWT-токенів, для підвищення безпеки авторизації та аутентифікації застосовувались додаткові заходи, такі як:

- Двофакторна автентифікація (2FA): вимагає від користувача надати фактори підтвердження особи, наприклад, пароль та одноразовий код, надісланий на мобільний телефон.
- Хешування паролів: зберігання паролів користувачів у вигляді хешів замість відкритого тексту, що ускладнює їх викрадення та використання зловмисниками.
- Використання SSL/TLS: забезпечення шифрування даних при передачі між клієнтом і сервером для запобігання перехопленню та несанкціонованому доступу [43, с. 211].

Для забезпечення безпеки при автентифікації користувачів використано алгоритм хешування паролів за допомогою бібліотеки bcrypt. Цей алгоритм додає

сіль (salt) до паролів перед хешуванням, що ускладнює їх зворотне відновлення у разі компрометації бази даних.

Для підвищення безпеки облікових записів користувачі використовується двофакторна аунтифікація (2FA). Окрім введення пароля, користувач повинен надати додатковий фактор, такий як одноразовий код, який надсилається на email для підтвердження своєї особистості [55, с. 203].

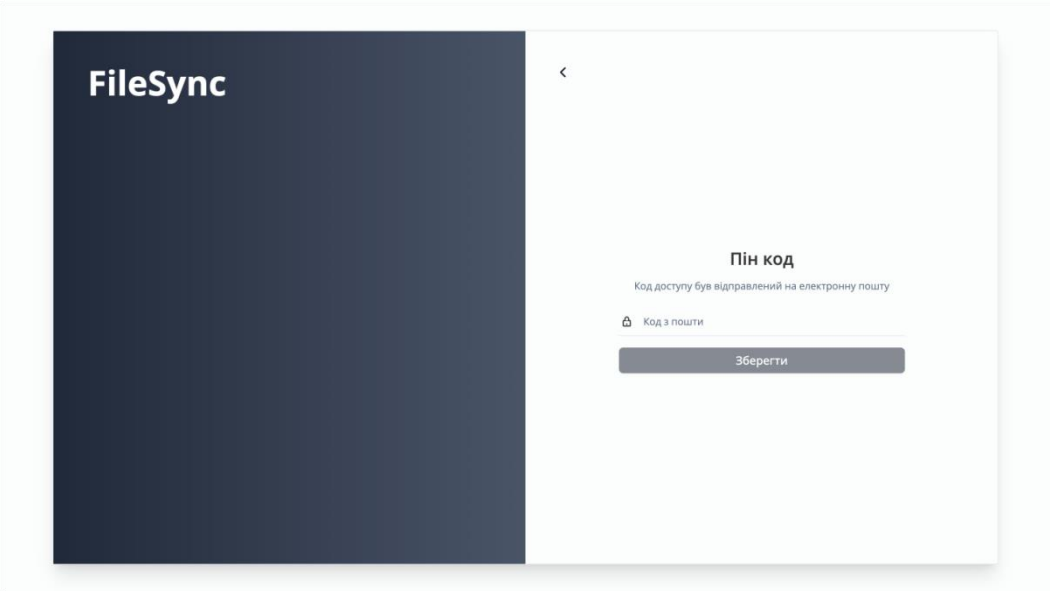

Рисунок 2.3.1 – Двофакторна аутентифікація

Для авторизації доступу до захищених ресурсів веб-сервісу використовується рольова модель контролю доступу (RBAC). RBAC дозволяє призначати користувачам ролі, які визначають їх права доступу до певних функцій або даних веб-сервісу. Користувачам надаються лише ті права доступу, які необхідні для виконання їх завдань. Це зменшує ризики, пов'язані з неавторизованим доступом або зловживанням привілеями.

Для запобігання атакам перебором паролів (brute-force attacks) використовується механізми обмеження кількості невдалих спроб входу або реєстрації. Після певної кількості невдалих спроб система може тимчасово заблокувати обліковий запис або вимагати додаткової верифікації, введення CAPTCHA [58, с. 112].

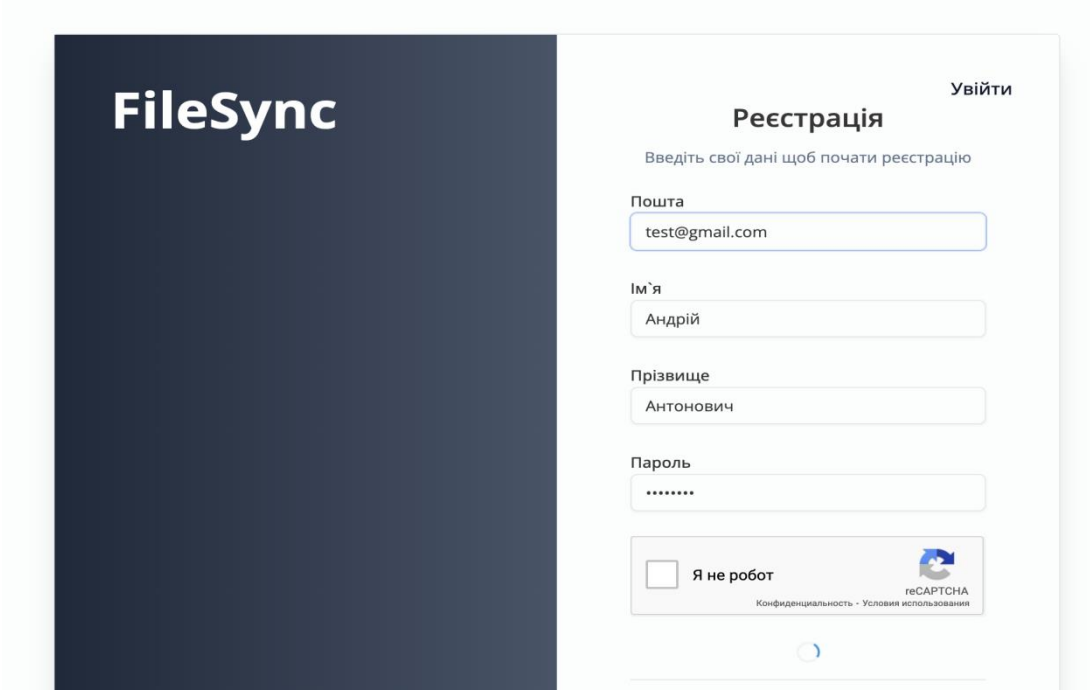

Рисунок 2.3.2 – Введення CAPTCHA

Для забезпечення безпечної передачі даних аутентифікації та авторизації між клієнтом і сервером використовується шифрування. Протокол HTTPS з використанням SSL/TLS дозволяє шифрувати весь трафік між клієнтом і сервером, захищаючи конфіденційні дані від перехоплення [61, с. 181].

Таким чином, розробка надійного та безпечного алгоритму авторизації та аутентифікації користувачів є ключовим аспектом при створенні веб-сервісу зберігання файлів. Використання JWT-токенів, двофакторної автентифікації, хешування паролів та шифрування даних дозволяє забезпечити високий рівень безпеки та захисту конфіденційних даних користувачів. Дотримання вимог законодавства щодо захисту персональних даних, моніторинг підозрілої активності та забезпечення масштабованості системи є важливими факторами при розробці алгоритму авторизації та аутентифікації користувачів у веб-сервісі зберігання файлів.

2.4 Проектування інтерфейсу користувача веб-сервісу

Проектування інтерфейсу користувача є важливим етапом у розробці веб-сервісу зберігання файлів з авторизованим доступом. Інтерфейс користувача (UI - User Interface) - це візуальна частина веб-сервісу, з якою безпосередньо взаємодіє користувач. Він повинен бути інтуїтивно зрозумілим, зручним у використанні та естетично привабливим, щоб забезпечити позитивний досвід користувача (UX - User Experience).

При проектуванні інтерфейсу користувача веб-сервісу зберігання файлів необхідно враховувати цільову аудиторію та її потреби. Інтерфейс повинен бути адаптований до різних пристроїв, таких як настільні комп'ютери, ноутбуки, планшети та смартфони, щоб забезпечити зручність використання на різних екранах. Основними принципами проектування інтерфейсу користувача є:

- Простота та інтуїтивність: інтерфейс повинен бути простим і зрозумілим для користувача, з чіткою навігацією та мінімальною кількістю кроків для виконання основних завдань [51, с. 123].
- Консистентність: елементи інтерфейсу повинні бути послідовними та узгодженими у всіх частинах веб-сервісу, використовуючи єдиний стиль, кольорову гаму та розташування елементів.
- Візуальна ієрархія: важливі елементи інтерфейсу повинні бути виділені за допомогою розміру, кольору та розташування, щоб привернути увагу користувача та полегшити навігацію.
- Доступність: інтерфейс повинен бути доступним для користувачів з обмеженими можливостями, дотримуючись стандартів веб-доступності, таких як WCAG (Web Content Accessibility Guidelines).

При проектуванні інтерфейсу користувача веб-сервісу зберігання файлів можна виділити такі основні компоненти:

- Головна сторінка: містить інформацію про веб-сервіс, переваги використання, заклик до дії (наприклад, "Зареєструватися" або "Увійти") та посилання на основні розділи веб-сервісу.

- Сторінка реєстрації та входу: дозволяє користувачам створити новий обліковий запис або увійти в існуючий, використовуючи логін та пароль або інші методи автентифікації (наприклад, OAuth) [57, с. 261].
- Особистий кабінет користувача: надає доступ до збережених файлів користувача, можливість завантаження нових файлів, управління папками та доступом до файлів, а також налаштування облікового запису.
- Сторінка перегляду файлу: відображає деталі обраного файлу, такі як назва, розмір, дата завантаження, власник та можливості для скачування або надання доступу іншим користувачам.
- Сторінка налаштувань: дозволяє користувачу змінювати особисті дані, пароль та безпеку облікового запису [60, с. 88].

Для розробки інтерфейсу користувача веб-сервісу зберігання файлів можна використовувати різні технології та інструменти, такі як:

- HTML (HyperText Markup Language) та Tailwind CSS для верстки та стилізації сторінок.
- Typescript та фреймворк Next.js для створення інтерактивних елементів та динамічного оновлення вмісту сторінок.
- UI-бібліотека та фреймворки Radix, ShadCn для використання готових компонентів та стилів.

Для забезпечення зручності використання та швидкого завантаження сторінок веб-сервісу можна застосовувати техніки оптимізації продуктивності, такі як:

- Мінімізація та стиснення файлів HTML, CSS та JavaScript для зменшення розміру завантажуваних ресурсів.
- Використання техніки ледачого завантаження (lazy loading) для відкладеного завантаження зображень та інших ресурсів, які не є видимими на екрані.
- Кешування статичних ресурсів на стороні клієнта та сервера для зменшення кількості запитів до сервера та прискорення завантаження сторінок.

Для забезпечення адаптивності інтерфейсу користувача до різних пристроїв можна використовувати техніки адаптивного дизайну (responsive design) та медіа-

запити CSS [39, с. 179]. Адаптивний дизайн дозволяє створювати єдиний інтерфейс, який автоматично підлаштовується під розмір екрану пристрою, змінюючи розташування та розмір елементів. Медіа-запити CSS дозволяють застосовувати різні стилі для різних роздільних здатностей екрану та орієнтації пристрою (портретна або ландшафтна).

Для покращення доступності інтерфейсу користувача для людей з обмеженими можливостями можна дотримуватися рекомендацій WCAG (Web Content Accessibility Guidelines). Це включає використання альтернативного тексту для зображень, забезпечення достатнього контрасту кольорів, можливість навігації за допомогою клавіатури, використання семантичної розмітки HTML та надання субтитрів для відео- та аудіо контенту .

Для забезпечення зручності користування інтерфейсом веб-сервісу зберігання файлів можна реалізувати такі функції:

- Drag-and-drop завантаження файлів: дозволяє користувачам перетягувати файли безпосередньо з їхнього комп'ютера у веб-інтерфейс для завантаження.
- Попередній перегляд файлів: надає можливість переглядати вміст файлів (наприклад, зображення або документи) безпосередньо у веб-інтерфейсі без необхідності їх завантаження.
- Пошук файлів: дозволяє користувачам шукати файли за назвою, типом, датою завантаження, транскриптом та коротким вмістом.

Отже, проектування інтерфейсу користувача веб-сервісу зберігання файлів з авторизованим доступом вимагає врахування потреб цільової аудиторії, забезпечення зручності використання, адаптивності до різних пристроїв та доступності для користувачів з обмеженими можливостями. Використання сучасних технологій та інструментів, таких як HTML, CSS, TypeScript та UI-фреймворки, дозволяє створити привабливий та функціональний інтерфейс. Застосування технік оптимізації продуктивності, адаптивного дизайну та дотримання рекомендацій щодо доступності забезпечує високу якість користувацького досвіду.

3 РЕАЛІЗАЦІЯ ВЕБ-СЕРВІСУ ЗБЕРІГАННЯ ФАЙЛІВ

Впровадження захищеного веб-сервісу для зберігання файлів передбачає кілька ключових кроків, щоб забезпечити відповідність системи функціональним вимогам і вимогам безпеки. У цьому розділі описані основні етапи цього процесу.

Першим кроком є визначення структури системи, вибір відповідного технологічного стеку та встановлення робочих процесів і потоків даних. Архітектура повинна бути масштабованою, підтримуваною та безпечною.

Налаштування середовища розробки передбачає налаштування необхідних інструментів, бібліотек та фреймворків, щоб забезпечити узгоджене налаштування для кодування, тестування та розгортання. Технології можуть включати фронтенд-фреймворки, такі як React, бекенд-фреймворки, такі як Nest.js, та бази даних, такі як MySql.

Фактичне кодування веб-сервісу включає в себе реалізацію фронтенд та бекенд компонентів та їх інтеграцію. Фронтенд-розробка зосереджена на створенні зручного інтерфейсу, в той час як бекенд забезпечує надійну логіку на стороні сервера, взаємодію з базами даних та протоколи безпеки.

3.1 Реалізація серверної частини веб-сервісу

Реалізація серверної частини веб-сервісу є ключовим етапом у розробці веб-сервісу зберігання файлів з авторизованим доступом. Серверна частина відповідає за обробку запитів від клієнтської частини, виконання бізнес-логіки, взаємодію з базою даних та надання відповідей клієнту. Правильно спроектована та реалізована серверна частина забезпечує надійність, продуктивність та безпеку веб-сервісу.

Для реалізації серверної частини веб-сервісу зберігання файлів використано фреймворк Nest.js з Typescript та серверна платформа Node.js. Node.js дозволяє

створювати високопродуктивні та масштабовані веб-сервери, використовуючи подвійно-орієнтовану архітектуру та неблокуючий ввід/вивід [3, с. 48].

Основними компонентами серверної частини веб-сервісу зберігання файлів є:

1. Веб-сервер: обробляє HTTP-запити від клієнтів та надсилає відповіді. Nest.js надає зручний інтерфейс для маршрутизації запитів, обробки параметрів та генерації відповідей.
2. Контролери: містять бізнес-логіку обробки запитів та формування відповідей. Контролери отримують дані від клієнта, виконують необхідні операції (наприклад, збереження файлу, оновлення метаданих), взаємодіють з сервісами та репозиторіями, та повертають результат клієнту.
3. Сервіси: інкапсулюють окремі функціональні можливості та операції з даними. Сервіси містять логіку для роботи з файлами (наприклад, збереження, зчитування, видалення), автентифікації користувачів, генерації унікальних ідентифікаторів тощо.
4. Репозиторії: забезпечують доступ до бази даних та виконують операції з даними. Репозиторії використовують об'єктно-реляційне відображення (ORM) або драйвери бази даних для взаємодії з базою даних, виконання запитів та отримання результатів.

Для забезпечення безпеки веб-сервісу на серверній частині реалізований механізми аутентифікації та авторизації користувачів. Автентифікація передбачає перевірку особи користувача, за допомогою логіна та пароля та токену доступу JWT [8, с. 57]. У системі Nest.js та JWT інтегрується за допомогою пакета nestjs/jwt, який добре працює з nestjs/passport для стратегій автентифікації. Авторизація визначає, які дії та ресурси доступні користувачу на основі його ролі та прав доступу.

Для збереження файлів на серверній частині веб-сервісу використано підхід збереження файлів з використання хмарного сховища Amazon S3. Цей сервіси надає можливість зберігання великих обсягів даних, забезпечує високу доступність та надійність. Використання хмарного сховища дозволяє зменшити навантаження на сервер веб-сервісу та забезпечити масштабованість зберігання файлів.

Для забезпечення можливості обміну файлами між користувачами та надання доступу до файлів на серверній частині веб-сервісу реалізовано механізми управління правами доступу. Це включає в себе створення системи ролей та дозволів, де кожен користувач має певну роль власник, читач та відповідні дозволи на виконання дій з файлами перегляд, редагування, видалення.

Отже, реалізація серверної частини веб-сервісу зберігання файлів з авторизованим доступом вимагає ретельного проектування архітектури, вибору відповідних технологій та врахування аспектів безпеки, продуктивності та масштабованості. Використання фреймворків, бібліотек та хмарних сервісів може значно спростити та прискорити розробку серверної частини, забезпечуючи надійність, безпеку та ефективність веб-сервісу. Правильно реалізована серверна частина є фундаментом для успішного функціонування веб-сервісу зберігання файлів з авторизованим доступом.

3.2 Реалізація клієнтської частини веб-сервісу

Реалізація клієнтської частини веб-сервісу зберігання файлів з авторизованим доступом є не менш важливою, ніж реалізація серверної частини. Клієнтська частина відповідає за взаємодію з користувачем, відображення інтерфейсу, надсилання запитів до серверної частини та обробку отриманих відповідей. Правильно реалізована клієнтська частина забезпечує зручність використання, швидкість роботи та привабливий дизайн веб-сервісу.

Для реалізації клієнтської частини веб-сервісу зберігання файлів використано фреймворк Next.js в поєднанні з Typescript. Для стилізації веб-сторінок використано утилітарний CSS-фреймворк, Tailwind, який можна комбінувати для створення будь-якого дизайну безпосередньо у розмітці.

Основними компонентами клієнтської частини веб-сервісу зберігання файлів є:

1. Інтерфейс користувача: веб-сторінки, які відображають елементи керування, форми для введення даних, списки файлів, кнопки та інші візуальні компоненти.

2. Логіка взаємодії з сервером: функції та методи, які відповідають за надсилання запитів до серверної частини, обробку відповідей та оновлення інтерфейсу користувача відповідно до отриманих даних. Для реалізації цієї логіки застосовуються бібліотеки, такі як Fetch API та useSwr для спрощення роботи з HTTP-запитами та реалізації повторного отримання даних при їх оновлені в базі даних[36, с. 87].
3. Управління станом: механізми для зберігання та оновлення стану клієнтської частини, такі як дані про автентифікацію користувача, список завантажених файлів, обрані файли тощо. Для управління станом використано бібліотеку яка є швидким та масштабованим рішенням для керування станом Zustand: що має зручний API на основі хуків, які дозволяють централізовано зберігати та оновлювати стан додатку.

Для забезпечення зручності використання та швидкої роботи клієнтської частини веб-сервісу застосовані такі підходи:

- Розробка адаптивного дизайну (responsive design), який забезпечує коректне відображення інтерфейсу на різних пристроях та розмірах екрану. Адаптивний дизайн досягається за допомогою використання медіа-запитів CSS та гнучких макетів.
- Оптимізація продуктивності шляхом мінімізації та стиснення файлів JavaScript та CSS, використання техніки ледачого завантаження (lazy loading) для відкладеного завантаження ресурсів, кешування статичних файлів та інших технік оптимізації швидкості завантаження сторінок.
- Забезпечення доступності (accessibility) інтерфейсу для користувачів з обмеженими можливостями шляхом дотримання стандартів доступності, таких як WCAG (Web Content Accessibility Guidelines), використання семантичної розмітки HTML, надання альтернативного тексту для зображень та забезпечення можливості керування інтерфейсом за допомогою клавіатури.

Для реалізації функціональності завантаження та відображення файлів у клієнтській частині веб-сервісу використані такі підходи:

- Відображення прогресу завантаження файлів за допомогою індикаторів прогресу або анімацій для інформування користувача про стан завантаження.
- Попередній перегляд завантажених файлів, таких як зображення або документи, безпосередньо у веб-інтерфейсі за допомогою вбудованих засобів перегляду браузера або сторонніх бібліотек.
- Організація файлів у папки та надання можливості навігації по файловій структурі за допомогою панелі навігації.
- Реалізація функцій сортування, фільтрації та пошуку файлів за різними критеріями, такими як назва файлу, вміст, розмір, дата завантаження тощо.
- Drag-and-drop завантаження файлів, що дозволяє користувачам перетягувати файли безпосередньо з їх локальної файлової системи у веб-інтерфейс для завантаження.

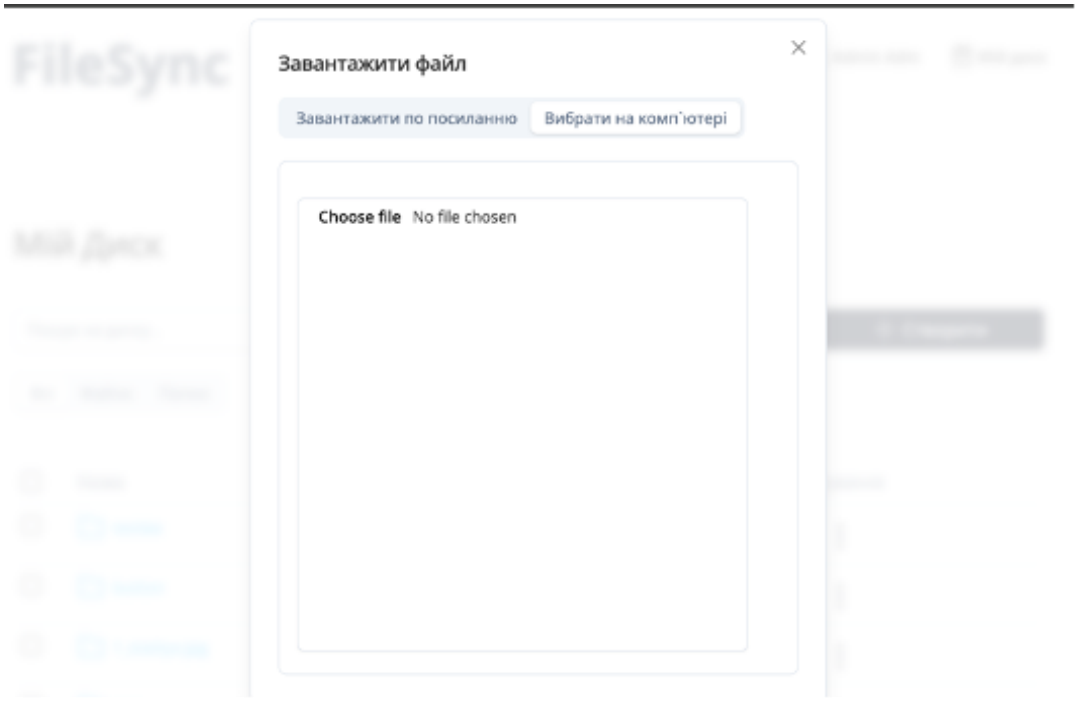

Рисунок 3.2.1 – Завантаження файлу

- Надання сповіщень та оповіщень користувачам про важливі події, такі як завершення завантаження файлу, отримання нового доступу до файлу або наближення до ліміту зберігання (рисунок - 3.2.2-3.2.3).

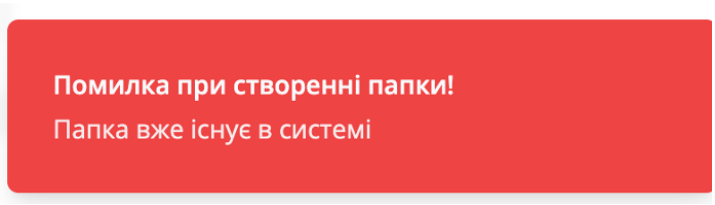

Рисунок 3.2.2 – Повідомлення про помилку

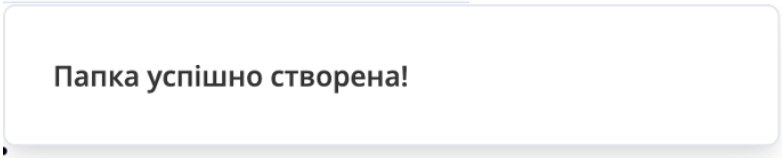

Рисунок 3.2.3 – Повідомлення про успішне виконання

Для забезпечення безпеки та конфіденційності даних користувачів у клієнтській частині веб-сервісу дотримані такі практики:

- Використано HTTPS для шифрування даних під час передачі між клієнтом та сервером, що запобігає перехопленню та несанкціонованому доступу до конфіденційної інформації [46, с. 157].
- Валідація введених користувачем даних на клієнтській стороні перед відправкою на сервер для запобігання атакам, таким як міжсайтовий скриптинг (XSS) або ін'єкції SQL.
- Застосувано аутентифікацію та авторизацію користувачів для обмеження доступу до захищених ресурсів та функціональності веб-сервісу. Використання токенів автентифікації, таких як JWT (JSON Web Tokens), для безпечної передачі інформації про аутентифікацію між клієнтом і сервером.
- Обмеження доступу до конфіденційних даних, таких як паролі користувачів, шляхом зберігання їх у захищеному вигляді (хешування) та уникнення їх зберігання у відкритому вигляді на клієнтській стороні [49, с. 217].

Для забезпечення адаптивності та кросбраузерності клієнтської частини веб-сервісу враховано відображення застосунку в системах пристроїв та браузерів, які використовують користувачі. Це включає:

- Тестування веб-сервісу на різних браузерах (Chrome, Firefox, Safari, Edge) та їх версіях для забезпечення коректного відображення та функціонування.
- Використання адаптивного дизайну та медіа-запитів CSS для забезпечення коректного відображення інтерфейсу на різних розмірах екрану та пристроях (десктопи, ноутбуки, планшети, смартфони) (рисунок - 3.2.4) [56, с. 155].

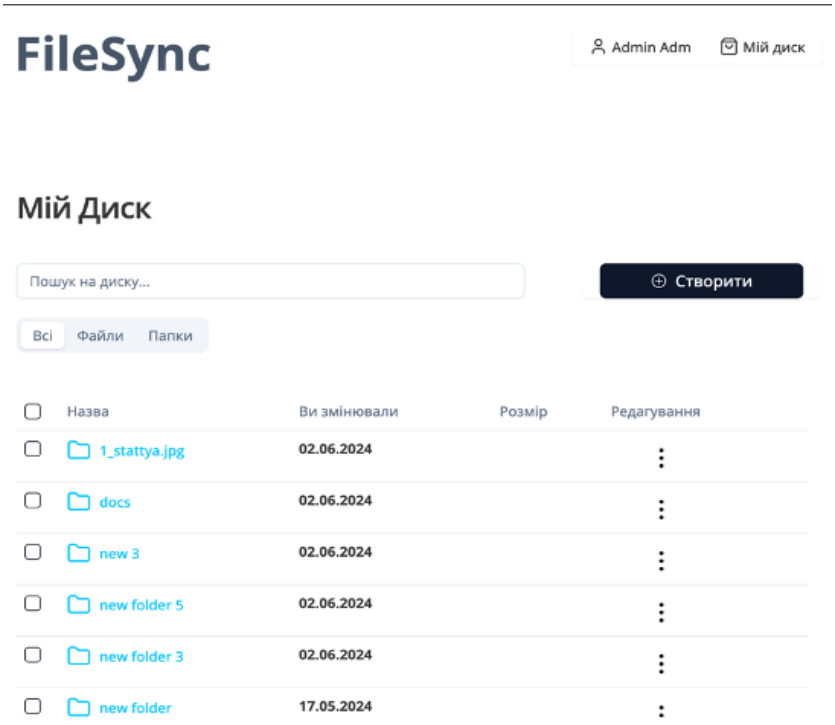

Рисунок 3.2.4 – Приклад адаптиву для планшету

Таким чином, реалізація клієнтської частини веб-сервісу зберігання файлів з авторизованим доступом вимагає використання сучасних веб-технологій, дотримання принципів зручності використання, адаптивності та безпеки. Застосування HTML, CSS та JavaScript у поєднанні з фреймворками та бібліотеками дозволяє створити привабливий та функціональний інтерфейс користувача. Оптимізація продуктивності, забезпечення доступності та кросбраузерності є важливими факторами для успішної реалізації клієнтської частини веб-сервісу. Увага до деталей, таких як зручність завантаження файлів, організація файлової структури, безпека даних та взаємодія з користувачем,

дозволяє створити високоякісний та затребуваний веб-сервіс зберігання файлів з авторизованим доступом.

3.3 Реалізація функціоналу завантаження, зберігання та скачування файлів

Реалізація функціоналу завантаження, зберігання та скачування файлів є ключовим аспектом веб-сервісу зберігання файлів з авторизованим доступом. Цей функціонал забезпечує можливість користувачам завантажувати свої файли на сервер, зберігати їх у структурованому вигляді та отримувати доступ до них для подальшого скачування. Правильно реалізований функціонал завантаження, зберігання та скачування файлів гарантує надійність, безпеку та зручність використання веб-сервісу.

Однією з найважливіших особливостей цієї системи є можливість завантажувати файли до AWS S3 за допомогою попередньо визначених URL-адрес. Цей метод забезпечує безпечний та ефективний спосіб завантаження файлів. Попередньо визначені URL-адреси дозволяють користувачам завантажувати файли безпосередньо на S3 без необхідності проходити через внутрішній сервер, що зменшує навантаження на сервер і підвищує продуктивність. Сервер генерує попередньо визначену URL-адресу для кожного запиту на завантаження, яку клієнт використовує для завантаження файлу безпосередньо на S3. Цей процес гарантує, що файли безпечно передаються і зберігаються в масштабованому хмарному сховищі.

Для забезпечення безпеки при завантаженні файлів здійснюється перевірка та валіація файлів. Це включає перевірку типу файлу та розширення. Також встановлено обмеження на максимальний розмір файлу, який може бути завантажений, щоб запобігти переповненню дискового простору та атакам типу "відмова в обслуговуванні" (DoS).

База даних MySQL зберігає метадані про файли і папки, зокрема інформацію про їхню структуру, права доступу та власників. Ця база даних необхідна для

підтримки ієрархічної організації файлів і папок, а також для забезпечення контролю доступу та дозволів.

Після успішного завантаження файлу, база даних MySQL зберігає метадані про файли і папки, зокрема інформацію про їхню структуру, права доступу та власників. При збереженні файлів у файловій системі, створюється спеціальна директорія для кожного користувача (рисунок - 3.3.1).

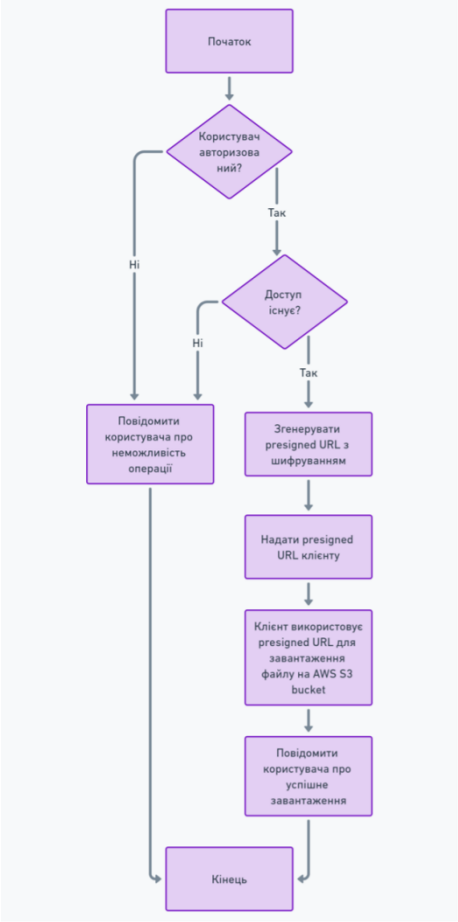

Рисунок 3.3.1 – Блок-схема завантаження файлу

Отримання файлів з сервера та AWS S3 є важливим аспектом сервісу зберігання файлів. Він включає в себе отримання метаданих файлів з сервера і власне вмісту файлів з AWS S3, забезпечуючи безперебійну та ефективну роботу користувачів.

Процес отримання файлів складається з двох основних завдань: отримання метаданих з сервера і доступ до вмісту файлу з AWS S3. Метадані включають таку

інформацію, як ім'я файлу, тип, розмір, права доступу та його розташування в ієрархії папок. Вміст файлу зберігається в AWS S3, і його потрібно отримати безпечно та ефективно.

Коли користувач хоче переглянути файл або керувати ним, першим кроком є отримання метаданих файлу з сервера. Метадані містять важливу інформацію про файл і зберігаються в базі даних MySQL. Ця база даних містить детальну інформацію про структуру файлу, права власності та дозволи.

Фронтенд-додаток, надсилає запит на внутрішній сервер, щоб отримати метадані. Цей запит зазвичай робиться за допомогою кінцевої точки RESTful API. Сервер обробляє його шляхом запиту до бази даних MySQL і повертає відповідні метадані. Ця відповідь містить такі дані, як ім'я файлу, тип, розмір, публічний чи приватний статус, а також URL-адресу AWS S3, де зберігається файл. Потім інтерфейсний додаток використовує отримані метадані для відображення інформації користувачеві і підготовки до отримання вмісту файлу. Після того, як метадані отримані, можна отримати доступ до вмісту файлу з сервісу AWS S3.

Для покращення користувацького досвіду при скачуванні файлів, реалізовано функцію відображення прогресу скачування. Також надано інформацію про швидкість скачування, час, що залишився, та розмір файлу.

Файли в S3 можуть бути як загальнодоступними, так і приватними, а доступ до них контролюється за допомогою попередньо підписаних URL-адрес з метою безпеки. Для загальнодоступних файлів метадані містять пряму URL-адресу до S3, що дозволяє користувачеві отримати доступ до файлу напряму. Для приватних файлів сервер генерує попередньо підписану URL-адресу, яка надає тимчасовий доступ до файлу. Ця попередньо підписана URL-адреса включається у відповідь з метаданими і дозволяє користувачеві безпечно завантажити файл, не розкриваючи облікові дані S3.

При скачуванні файлу, є реалізована функції попереднього перегляду файлів, конвертації форматів та створення архівів. Попередній перегляд файлів дозволяє користувачам переглядати вміст файлу безпосередньо у веб-інтерфейсі, без необхідності скачувати його.

Таким чином, реалізація функціоналу завантаження, зберігання та скачування файлів у веб-сервісі зберігання файлів з авторизованим доступом ретельно спроектована з використанням відповідних технологій і підходів. Безпека, продуктивність та масштабованість є ключовими факторами, які враховані при реалізації цього функціоналу. Реалізація додаткових функцій, таких як попередній перегляд файлів, відображення завантаження, покращує користувацький досвід та розширює можливості веб-сервісу зберігання файлів з авторизованим доступом.

3.4 Реалізація системи авторизації та розмежування доступу до файлів

Реалізація системи авторизації та розмежування доступу до файлів є критично важливим аспектом веб-сервісу зберігання файлів з авторизованим доступом. Ця система забезпечує безпеку та конфіденційність даних користувачів, а також дозволяє контролювати доступ до файлів на основі прав та ролей користувачів. Правильно реалізована система авторизації та розмежування доступу гарантує, що лише авторизовані користувачі можуть отримати доступ до відповідних файлів та виконувати дозволені операції.

Авторизація - це процес надання користувачу дозволу на виконання певних дій або доступ до певних ресурсів на основі його прав та привілеїв. У контексті веб-сервісу зберігання файлів з авторизованим доступом, авторизація визначає, які файли та операції (перегляд, редагування, видалення) доступні для конкретного користувача. Авторизація зазвичай відбувається після успішної автентифікації користувача, тобто підтвердження його особи [31, с. 92].

Щоб мати можливість взаємодіяти з файлами користувач повинен авторизуватись. Користувач заповнює форму з поштою та паролем. Після відправки форми, перевіряється наявність даного користувача в системі та повідомляється про результат (рисунок - 3.4.1).

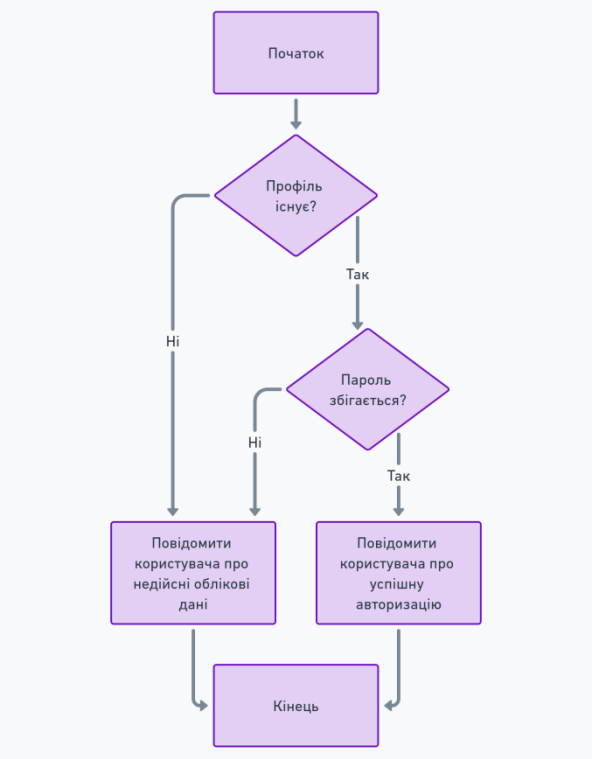

Рисунок 3.4.1 – Процес авторизації

Розмежування доступу - це процес обмеження доступу користувачів до файлів та ресурсів на основі їх ролей, груп або індивідуальних прав. Розмежування доступу дозволяє визначити, хто може переглядати, редагувати або видаляти файли, а також забезпечує безпеку та конфіденційність даних. Розмежування доступу може бути засноване на різних моделях, таких як дискреційна модель (DAC), мандатна модель (MAC) або рольова модель (RBAC).

Для реалізації системи авторизації та розмежування доступу до файлів у веб-сервісі зберігання файлів з авторизованим доступом можна використовувати різні підходи та технології. Одним з найпоширеніших підходів є використання токенів автентифікації, таких як JWT (JSON Web Token) [34, с. 201]. JWT - це стандарт для створення токенів доступу, які містять інформацію про автентифікацію та авторизацію користувача в компактному та захищеному форматі.

Процес авторизації з використанням JWT складається з наступних кроків:

1. Користувач надсилає запит на аутентифікацію, надаючи свої облікові дані (наприклад, ім'я користувача та пароль).

2. Сервер перевіряє надані облікові дані та, у разі успішної аутентифікації, генерує JWT-токен, який містить інформацію про користувача та його права доступу.
3. Сервер надсилає згенерований JWT-токен клієнту у відповіді.
4. Клієнт зберігає отриманий JWT-токен у локальному сховищі браузера та додає його до кожного наступного запиту до сервера в заголовку Authorization.
5. Сервер отримує запит з JWT-токеном, перевіряє його цілісність та достовірність, а також извлекает з нього інформацію про користувача та його права доступу.
6. На основі інформації з JWT-токена, сервер приймає рішення про надання доступу до запитуваних файлів або операцій [36, с. 189].

Для зберігання інформації про права доступу користувачів до файлів ми використовуємо базу даних. У базі даних створюється таблиця, яка містить зв'язки між користувачами та файлами і визначає рівень доступу (наприклад, читання, запис, видалення) для кожного зв'язку. У файловій системі було використано списки контролю доступу (ACL), які визначають права доступу для кожного файлу або каталогу.

Для зберігання інформації про права доступу користувачів до файлів була використана база даних. У ній створено таблицю, яка містить зв'язки між користувачами та файлами, а також визначає рівень доступу (наприклад, читання, запис, видалення) для кожного зв'язку. У файловій системі використано списки контролю доступу (ACL - Access Control List), які визначають права доступу для кожного файлу або директорії.

Для забезпечення гнучкості та масштабованості системи авторизації та розмежування доступу, використано рольову модель (RBAC - Role -Based Access Control) [39, с. 236]. RBAC дозволяє групувати користувачів за ролями, кожна з яких має певний набір прав та дозволів. Створено ролі "адміністратор", "редактор" та "користувач" з різними рівнями доступу до файлів та операцій. Користувачам може бути призначено одну або декілька ролей, що спрощує управління правами доступу та зменшує ризик помилок налаштування.

Для реалізації розмежування доступу на рівні файлів та директорій, використано ієрархічну модель доступу. Кожен файл або директорія може мати

власника (користувача, який створив або завантажив файл) та список прав доступу для інших користувачів або груп. Права доступу включає читання, запис, видалення та інші операції. Ієрархічна модель дозволяє успадковувати права доступу від батьківських директорій до дочірніх файлів та підпапок [42, с. 127].

Для забезпечення безпеки системи авторизації та розмежування доступу, було реалізовано механізми захисту від несанкціонованого доступу та атак. Це включає:

- Шифрування JWT-токенів за допомогою надійних алгоритмів, для запобігання підробки або розшифровки токенів зловмисниками.
- Перевірку цілісності та терміну дії JWT-токенів на сервері при кожному запиті, щоб гарантувати, що токен не був змінений або скомпрометований [44, с. 109].
- Використання захищених протоколів передачі даних, таких як HTTPS, для шифрування комунікації між клієнтом та сервером та запобігання перехопленню JWT-токенів або інших конфіденційних даних.
- Реалізацію механізмів автентифікації користувачів, таких як двофакторна автентифікація (2FA) або автентифікація на основі сертифікатів, для підвищення безпеки доступу до веб-сервісу.

Для надання користувачам можливості керування доступом до своїх файлів, реалізовано функціонал розподілу прав доступу. Користувачі можуть надавати або відкликати доступ до своїх файлів іншим користувачам або групам, встановлювати різні рівні доступу (наприклад, перегляд, редагування) та встановлювати терміни дії доступу [49, с. 191]. Це дозволяє користувачам гнучко контролювати, хто може отримувати доступ до їх файлів та як довго.

Для забезпечення відповідності законодавчим вимогам щодо захисту персональних даних, таким як GDPR (General Data Protection Regulation), реалізовано механізми отримання згоди користувачів на обробку їх даних та надання можливості керування своїми даними [53, с. 198]. Користувачі мають можливість переглядати, які дані про них зберігаються в системі та видаляти свої дані у разі потреби (рисунок - 3.4.2).

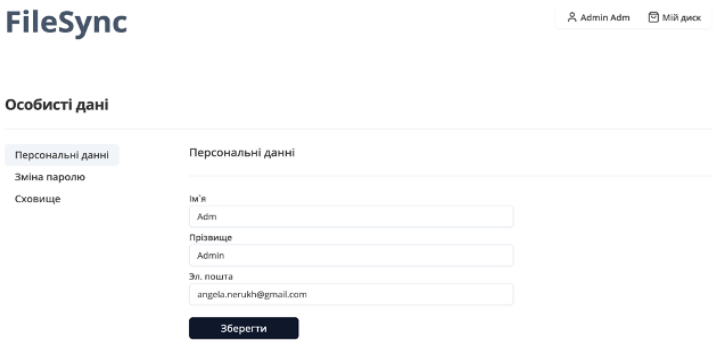

Рисунок 3.4.2 – Персональний кабінет

Для спрощення процесу надання доступу до файлів, був реалізований функціонал запрошень та публічних посилань. Користувачі можуть створювати спеціальні запрошення або посилання, які дозволяють іншим користувачам отримати доступ до певних файлів або папок без необхідності створення облікового запису в системі. Ці запрошення надсилаються на пошту та мають обмежений термін дії (рисунок - 3.4.3).

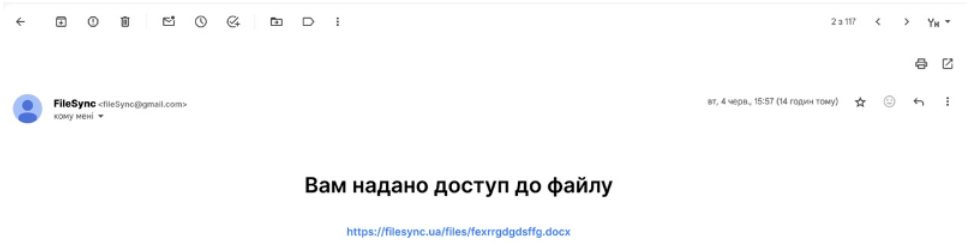

Рисунок 3.4.3 – Повідомлення з посиланням на тимчасовий доступу до файлу

Для забезпечення відповідності законодавчим вимогам щодо захисту персональних даних, таким як GDPR (General Data Protection Regulation), реалізовано механізми отримання згоди користувачів на обробку їх даних та надання можливості керування своїми даними [53, с. 198]. Користувачі мають

можливість переглядати, які дані про них зберігаються в системі та видаляти свої дані у разі потреби.

Таким чином, реалізація системи авторизації та розмежування доступу до файлів є невід'ємною частиною веб-сервісу зберігання файлів з авторизованим доступом. Використання токенів автентифікації, таких як JWT, дозволяє ефективно та безпечно передавати інформацію про автентифікацію та авторизацію між клієнтом і сервером. Застосування різних моделей розмежування доступу, таких як RBAC та ієрархічна модель, забезпечує гнучкість та масштабованість управління правами доступу. Реалізація механізмів, керування доступом та відповідності законодавчим вимогам гарантує захист конфіденційних даних користувачів та дозволяє створити надійну та безпечну систему авторизації та розмежування доступу до файлів у веб-сервісі зберігання файлів з авторизованим доступом.

4 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ВЕБ-СЕРВІСУ

Тестування та впровадження - найважливіші етапи розробки веб-сервісу для безпечного зберігання файлів. Ці етапи забезпечують функціональність, надійність, продуктивність і безпеку системи, виявляючи та вирішуючи потенційні проблеми ще до випуску продукту.

Тестування починається з розробки тестових кейсів на основі функціональних і нефункціональних вимог до веб-сервісу з урахуванням різних сценаріїв використання. Воно включає модульне тестування для перевірки окремих компонентів, інтеграційне тестування для забезпечення безперебійної роботи компонентів, а також тестування безпеки для виявлення вразливостей. Тестування продуктивності оцінює швидкість реакції системи та обробку даних під різними навантаженнями, а тестування прийнятності для користувачів (UAT) збирає відгуки реальних користувачів для вдосконалення системи.

Під час впровадження готується середовище розгортання шляхом встановлення необхідного програмного забезпечення та налаштування сервера. Автоматизовані інструменти та конвеєри безперервної інтеграції/поставки (CI/CD) використовуються для оптимізації процесу розгортання. Заходи безпеки, такі як конфігурація HTTPS, автентифікація користувачів та контроль доступу, застосовуються для захисту веб-сервісу. Оптимізація продуктивності включає в себе балансування навантаження, кешування та оптимізацію сервера.

4.1 Розробка тест-кейсів та проведення тестування веб-сервісу

Розробка тест-кейсів та проведення тестування веб-сервісу є невід'ємною частиною процесу розробки програмного забезпечення. Тестування дозволяє перевірити функціональність, надійність, продуктивність та безпеку веб-сервісу, а також виявити потенційні помилки та недоліки перед випуском продукту.

Правильно розроблені тест-кейси та ретельне тестування гарантують якість веб-сервісу та задоволеність користувачів.

Тест-кейс - це набір умов або змінних, за якими тестувальник визначатиме, чи відповідає система, що тестується, вимогам чи працює вона коректно. Тест-кейси розробляються на основі функціональних та нефункціональних вимог до веб-сервісу, а також з урахуванням можливих сценаріїв використання системи.

При розробці тест-кейсів для веб-сервісу зберігання файлів з авторизованим доступом враховувались різні аспекти функціональності системи, такі як:

- Реєстрація та автентифікація користувачів
- Завантаження та скачування файлів
- Створення та управління папками
- Надання та обмеження доступу до файлів
- Пошук та фільтрація файлів
- Обробка помилок та виняткових ситуацій

Тест-кейси охоплювали як позитивні, так і негативні сценарії використання веб-сервісу. Позитивні тест-кейси перевіряють коректність роботи системи при валідних вхідних даних та очікуваних умовах [34, с. 163]. Наприклад, тест-кейс для перевірки успішного завантаження файлу може містити такі кроки:

1. Користувач авторизується в системі
2. Користувач переходить на сторінку завантаження файлу
3. Користувач вибирає файл з локальної файлової системи
4. Користувач натискає кнопку "Завантажити"
5. Система успішно завантажує файл та відображає повідомлення про успішне завантаження [50, с. 235]

Приклади результатів тестування наведені у таблицях 4.1 - 4.9, що містять інформацію про виконані тести, початковий стан системи, вхідні дані необхідні, щоб відтворити знайдені дефекти, очікуваний результат та опис проблеми. Звіти про тестування дозволяють відстежувати прогрес тестування, оцінювати та підтримувати якість продукту та приймати обґрунтовані рішення щодо випуску веб-сервісу.

Таблиця 4.1 – Тест 1.1

Назва тесту	Перевірка реєстрації користувача
Модуль	Аутентифікація
Початковий стан системи	Сторінка входу в систему
Вхідні дані	Дійсна електронна пошта та пароль
Опис тесту	Користувач реєструється з дійсними обліковими даними
Очікуваний результат	Користувач успішно зареєстрований, відбулось перенаправлення на сторінку логіну.
Фактичний результат	Користувач успішно зареєстрований, відбулось перенаправлення на сторінку логіну.

Таблиця 4.2 – Тест 1.2

Назва тесту	Перевірка входу користувача
Модуль	Аутентифікація
Початковий стан системи	Сторінка входу в систему
Вхідні дані	Дійсна електронна пошта та пароль
Опис тесту	Користувач входить з дійсними обліковими даними
Очікуваний результат	Користувач успішно ввійшов
Фактичний результат	Користувач успішно ввійшов

Таблиця 4.3 – Тест 1.3

Назва тесту	Перевірка помилкового входу користувача
Модуль	Аутентифікація
Початковий стан системи	Сторінка входу в систему
Вхідні дані	Неправильна електронна пошта та пароль
Опис тесту	Користувач намагається увійти з невірними даними
Очікуваний результат	Повідомлення про помилку, що дані невірні
Фактичний результат	Повідомлення про помилку, що дані невірні

Таблиця 4.4 – Тест 1.4

Назва тесту	Перевірка скачування файлу
Модуль	Управління файлами
Початковий стан системи	Панель користувача з завантаженими файлами
Вхідні дані	Файл для скачування
Опис тесту	Користувач завантажує файл
Очікуваний результат	Файл успішно завантажений, відображення повідомлення про успішне завантаження файлу
Фактичний результат	Файл успішно завантажений, відображення повідомлення про успішне завантаження файлу

Таблиця 4.5 – Тест 1.5

Назва тесту	Перевірка видалення файлу
Модуль	Управління файлами
Початковий стан системи	Панель користувача з завантаженими файлами
Вхідні дані	Існуючий файл
Опис тесту	Користувач видаляє файл
Очікуваний результат	Файл успішно видалений
Фактичний результат	Файл успішно видалений

Таблиця 4.6 – Тест 1.6

Назва тесту	Перевірка налаштування доступу файлу
Модуль	Управління доступом
Початковий стан системи	Панель користувача з завантаженими файлами
Вхідні дані	Існуючий файл, пошта іншого користувача
Опис тесту	Користувач встановлює дозволи до файлу
Очікуваний результат	Дозволи успішно встановлені і поштою надіслане іншому користувачеві посилання на файл
Фактичний результат	Дозволи успішно встановлені і поштою надіслане іншому користувачеві посилання на файл

Таблиця 4.8 – Тест 1.8

Назва тесту	Перевірка 404 та 500 сторінок

Модуль	Обробка помилок
Початковий стан системи	Будь-яка сторінка
Вхідні дані	Перехід на неіснуючу сторінку
Опис тесту	Користувач переходить на 404 та 500 сторінку помилки дозволи до файлу
Очікуваний результат	Користувачу відображається 404 та 500 сторінка
Фактичний результат	Користувачу відображається 404 та 500 сторінка

Таблиця 4.9 – Тест 1.9

Назва тесту	Перевірка пошуку файлів за метаданими
Модуль	Пошук та фільтрація
Початковий стан системи	Панель користувача з завантаженими файлами
Вхідні дані	Ключові слова файлу
Опис тесту	Користувач шукає файл за змістом
Очікуваний результат	Файл знайдено
Фактичний результат	Файл знайдено

Також було проведено тестування безпеки веб-сервісу, що включає перевірку на наявність вразливостей, таких як ін'єкції SQL, міжсайтовий скриптинг (XSS), міжсайтова підробка запитів (CSRF). Для перевірки на наявність SQL ін'єкцій використано мануальне тестування. Додано корисне навантаження SQL-ін'єкції до параметрів URL-адреси і перевірено, чи немає незвичної поведінки або помилок. Вручну було введено типову SQL-ін'єкцію “DROP TABLE users;” в різні поля і було проаналізовано поведінку програми. У висновку програма не зреагувала на ін'єкцію. Ці підходи дозволяють пріоритизувати тестові випадки, зосередитись на найбільш критичних областях системи та оптимізувати використання ресурсів [41, с. 96].

Таким чином, розробка тест-кейсів та проведення тестування веб-сервісу зберігання файлів з авторизованим доступом є критично важливими етапами в процесі розробки програмного забезпечення. Ретельне планування та реалізація тестування, використання автоматизованих інструментів та методологій, а також документування результатів тестування дозволяють забезпечити високу якість,

надійність та безпеку веб-сервісу. Регулярне та ефективне тестування є запорукою успішного розгортання та функціонування веб-сервісу зберігання файлів з авторизованим доступом.

4.2 Аналіз результатів тестування та виправлення помилок

Аналіз результатів тестування та виправлення помилок є невід'ємною частиною процесу забезпечення якості веб-сервісу зберігання файлів з авторизованим доступом. Цей етап дозволяє виявити та усунути недоліки, помилки та вразливості системи перед її випуском або розгортанням. Ефективний аналіз результатів тестування та своєчасне виправлення помилок гарантують стабільність, надійність та безпеку веб-сервісу.

При аналізі результатів тестування важливо враховувати не тільки функціональні помилки, але й проблеми з продуктивністю, зручністю використання, безпекою та сумісністю веб-сервісу. Ці аспекти можуть мати значний вплив на користувацький досвід та загальне сприйняття якості продукту [44, с. 203]. Для аналізу ефективності та продуктивністю веб-сервісу було використано розширення Lighthouse.

Lighthouse - це автоматизований інструмент з відкритим вихідним кодом для покращення якості веб-сторінок. Його можна запустити в Chrome DevTools, з командного рядка або як модуль Node. Він забезпечує аудит продуктивності, доступності, прогресивних веб-додатків, SEO тощо.

Звіт був сформований у Chrome DevTools. Налаштовані категорії, такі як продуктивність, доступність, кращі практики, SEO. Після першої генерації звіту (рисунок - 4.2.1) було розглянуто та проаналізовано оцінки та детальні рекомендації щодо покращення веб-сторінки.

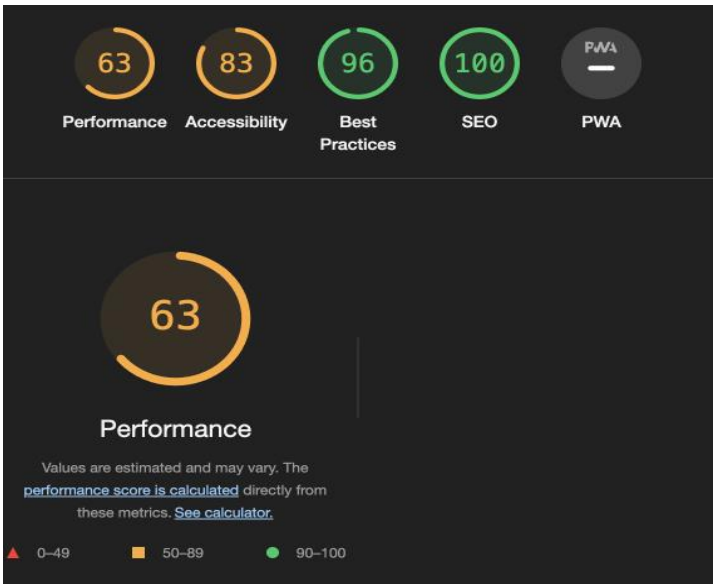

Рисунок 4.2.1 - Звіт з завантаженості сторінки Lighthouse

Після ідентифікації та класифікації помилок, розроблено план їх виправлення та проведені роботи над усуненням цих помилок. В результаті покращено продуктивність завантаження з 63% до 93% (рисунок - 4.2.1)

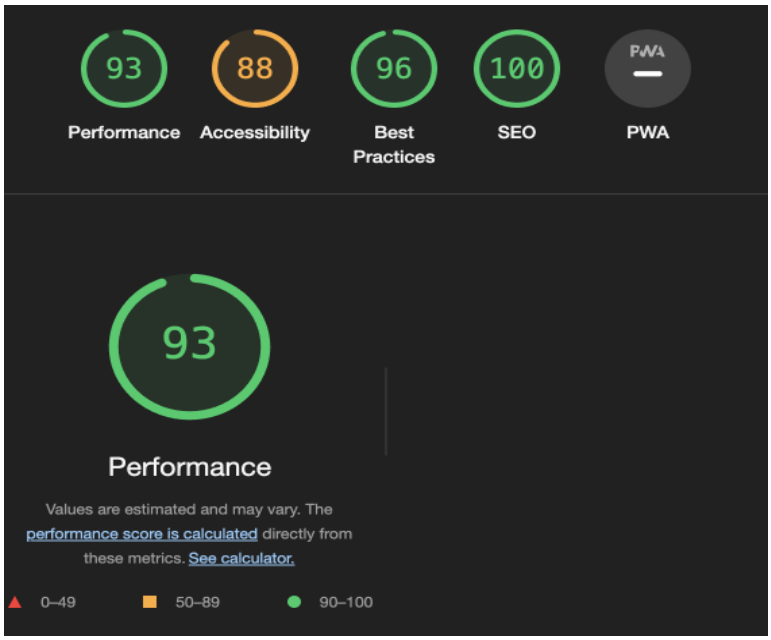

Рисунок 4.2.2 - Покращений звіт завантаженості сторінки Lighthouse

Після внесення виправлень, необхідно проведене повторне тестування відповідних компонентів та функціональності веб-сервісу, щоб переконатися у коректності виправлень та відсутності нових помилок.

Таким чином, аналіз результатів тестування та виправлення помилок є критично важливими етапами в життєвому циклі розробки веб-сервісу зберігання файлів з авторизованим доступом. Ефективний аналіз результатів тестування дозволяє виявити та класифікувати помилки, визначити їх пріоритетність та розробити план їх виправлення. Процес виправлення помилок повинен бути ретельно спланований, задокументований та перевірений за допомогою повторного тестування. Використання автоматизованих інструментів та систем відстеження помилок може значно спростити та пришвидшити цей процес. Регулярний моніторинг та аналіз метрик якості дозволяє оцінювати ефективність процесу тестування та виправлення помилок, а також приймати обґрунтовані рішення щодо вдосконалення процесів розробки. Своєчасне та якісне виправлення помилок забезпечує надійність, безпеку та високу якість веб-сервісу зберігання файлів з авторизованим доступом.

4.3 Розгортання веб-сервісу на сервері та налаштування його роботи

Розгортання веб-сервісу на сервері та налаштування його роботи є важливим етапом у процесі впровадження веб-сервісу зберігання файлів з авторизованим доступом. Цей етап включає в себе встановлення необхідного програмного забезпечення, конфігурацію серверного середовища, розгортання веб-сервісу та його компонентів, а також налаштування параметрів безпеки та продуктивності.

Ефективне та надійне розгортання веб-сервісів є критично важливим аспектом сучасної розробки програмного забезпечення. Інтегруючи Docker для контейнеризації, конвеєри CI/CD для автоматизації та AWS для масштабованої інфраструктури, можна досягти надійних робочих процесів розгортання.

Для контейнеризації веб-додатку використано Docker, забезпечуючи узгодженість середовища на різних етапах розробки та розгортання. Створений Docker-файл (рисунок - 4.3.1) для визначення процесу збірки образу Docker.

```
Dockerfile > ...
25  ##################
26  # BUILD FOR PRODUCTION
27  ##################
28
29  FROM node:18-alpine As build
30
31  WORKDIR /usr/src/app
32
33  COPY --chown=node:node package*.json ./
34  COPY --chown=node:node yarn.lock ./
35
36  COPY --chown=node:node --from=development /usr/src/app/node_modules ./node_modules
37
38  COPY --chown=node:node . .
39
40  # Run the build command which creates the production bundle
41  RUN yarn build
42
43  RUN yarn install --production --immutable --immutable-cache --check-cache
44
45
46  USER node
47
48  ##################
49  # PRODUCTION
50  ##################
51
52  FROM node:18-alpine As production
53
54  # Copy the bundled code from the build stage to the production image
55  COPY --chown=node:node --from=build /usr/src/app/node_modules ./node_modules
56  COPY --chown=node:node --from=build /usr/src/app/dist ./dist
57  COPY --chown=node:node --from=build /usr/src/app/package.json ./
58
59  # Start the server using the production build
60  CMD yarn start:prod
```

Рисунок 4.3.1 – Конфігураційний докер-файл, у якому описано інструкції, що будуть застосовані під час складання докер-образу

Для автоматизації процесу розгортання веб-сервісу використано інструменти неперервної інтеграції та доставки (CI/CD), Github CI/CD [7, с. 157]. Цей інструмент дозволяє створювати конвеєри (pipelines) розгортання, які автоматично будують, тестують та розгортають веб-сервіс при внесенні змін до коду.

Для забезпечення безпеки веб-сервісу під час розгортання налаштований протокол HTTPS для шифрування трафіку між клієнтом та сервером. Це вимагає отримання SSL-сертифікату від довіреного центру сертифікації та налаштування

веб-сервера для використання цього сертифікату. А також обмежено мережевий доступ до серверу лише з авторизованих IP-адрес або діапазонів.

Після розгортання веб-сервісу проведено додаткове тестування для перевірки коректності роботи всіх компонентів та функціональності в умовах реального середовища. Це включає тестування продуктивності, безпеки, сумісності та зручності використання веб-сервісу [52, с. 83].

Отже, розгортання веб-сервісу на сервері та налаштування його роботи є критично важливим етапом у життєвому циклі веб-сервісу зберігання файлів з авторизованим доступом. Цей етап вимагає ретельного планування, автоматизації та дотримання кращих практик безпеки, продуктивності та масштабованості. Використання сучасних інструментів та підходів, таких як CI/CD, IaC, моніторинг та комунікація, дозволяє забезпечити надійне та ефективне розгортання веб-сервісу та його безперебійну роботу.

4.4 Взаємодія користувача з програмною системою

Написання документації та інструкцій для користувачів веб-сервісу зберігання файлів з авторизованим доступом є важливим етапом у процесі впровадження та підтримки веб-сервісу. Документація та інструкції допомагають користувачам зрозуміти функціональність, можливості та правила використання веб-сервісу, а також полегшують його освоєння та ефективну роботу з ним. Далі розглянемо сценарій взаємодії користувача з інтерфейсом веб-застосунку.

Під час першого запуску або після виходу з облікового запису, користувачу буде запропоновано авторизуватися або зареєструватися. Для цього потрібно ввести свої облікові дані (пошту та пароль) або створити новий обліковий запис, заповнивши реєстраційну форму (рисунок - 4.4.1).

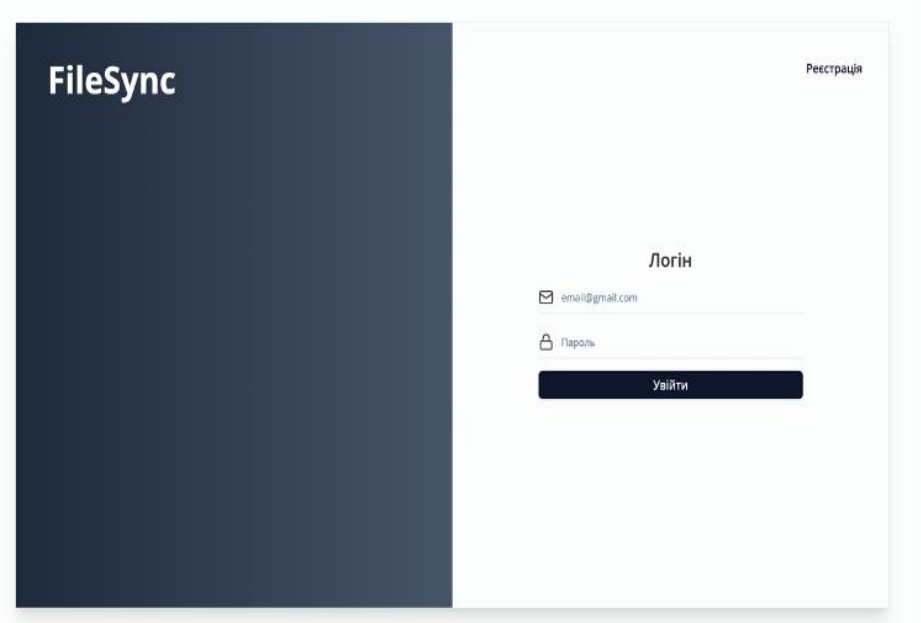

Рисунок 4.4.1 – Сторінка входу

Після успішної авторизації користувач потрапляє на основний екран веб-застосунку (рисунок - 4.4.2), де відображаються основні функціональні блоки, такі як список файлів, папок та інші елементи управління документами.

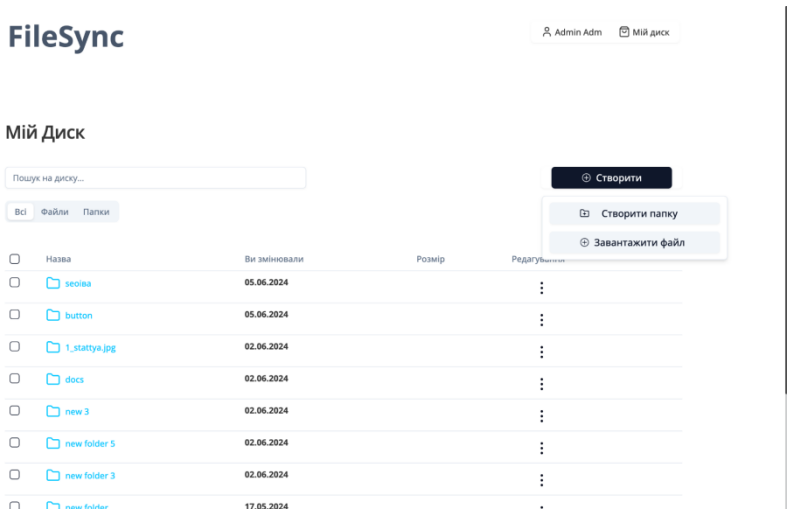

Рисунок 4.4.2 – Основний екран

Користувач може завантажувати файли на сервер за допомогою зручного інтерфейсу завантаження(рисунок - 4.4.3). Це включає вибір файлу з локального

пристрою та його завантаження у хмарне сховище (AWS S3). Після завантаження файлу, користувач може переглядати його у списку доступних документів.

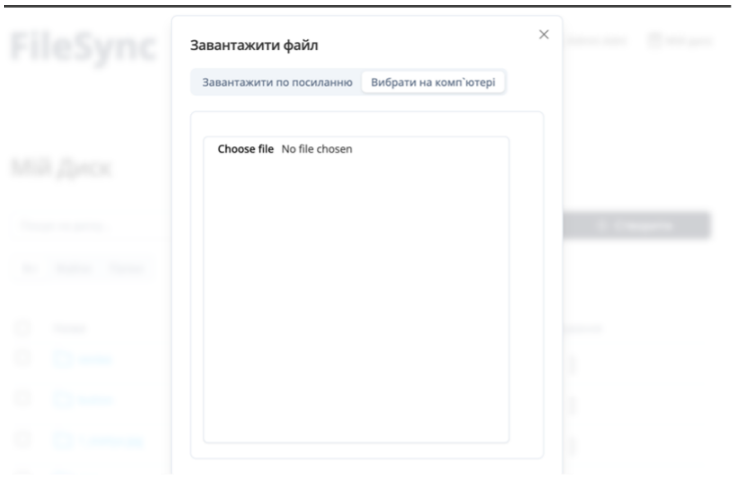

Рисунок 4.4.3 – Завантаження файлу

Користувачі можуть організовувати файли у папки для зручного управління. Це включає створення нових папок, перейменування, переміщення та видалення файлів та папок (рисунок - 4.4.4). Також користувачі можуть переглядати метадані файлів, такі як дата завантаження, розмір файлу та інші атрибути.

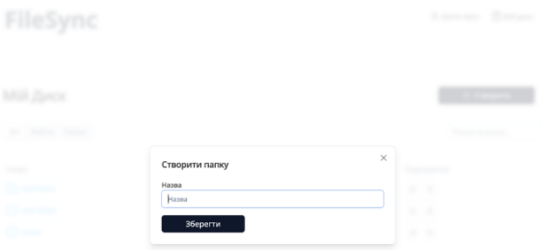

Рисунок 4.4.4 – Створення папки

Користувачі можуть налаштовувати доступ до файлів та папок для інших користувачів (рисунок - 4.4.5). Це включає встановлення різних рівнів доступу

(перегляд, редагування, видалення) та запрошення інших користувачів до спільного використання документів. Налаштування доступу дозволяють контролювати, хто може бачити або редагувати файли.

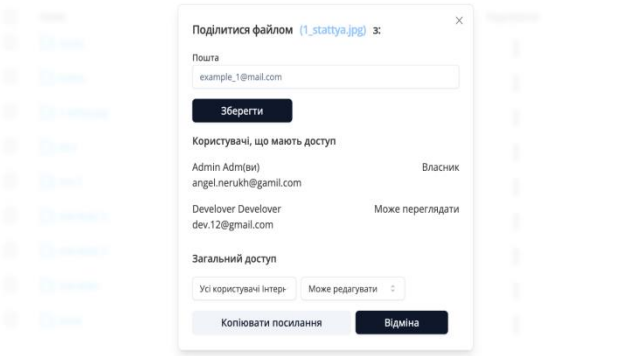

Рисунок 4.4.5 – Налаштування доступу до папки

Користувачі можуть ділитися файлами з іншими користувачами, надаючи їм тимчасовий і постійний доступ (рисунок - 4.4.6). Це може бути зроблено шляхом відправки запрошення на електронну пошту або наданням спеціального посилання для доступу до файлу.

Рисунок 4.4.6 – Спільний доступ до файлів

Для швидкого доступу до необхідних документів користувачі можуть використовувати функцію пошуку. Пошук може здійснюватися за назвою файлу, метаданими або тегами, що дозволяє швидко знаходити потрібні файли серед великої кількості документів.

Ці етапи забезпечують зручний та інтуїтивно зрозумілий сценарій взаємодії користувача з веб-застосунком, що сприяє підвищенню продуктивності та ефективності роботи з документами**.**

Таким чином, написання документації та інструкцій для користувачів веб-сервісу зберігання файлів з авторизованим доступом є невід'ємною частиною процесу розробки та підтримки веб-сервісу. Якісна та зрозуміла документація допомагає користувачам ефективно використовувати веб-сервіс, знаходити відповіді на питання та вирішувати проблеми самостійно. Використання сучасних інструментів, форматів та підходів до створення документації дозволяє забезпечити високу якість обслуговування та задоволеність користувачів веб-сервісом.

ВИСНОВКИ

У результаті виконання дипломної роботи було створено веб-сервіс зберігання файлів з авторизованим доступом, який повністю відповідає поставленим цілям та сучасним вимогам у сфері веб-розробки та безпеки. Цей веб-сервіс гарантує безпечне, зручне та ефективне зберігання, доступ та управління файлами для користувачів.

Перед початком розробки проведено аналіз існуючих рішень для зберігання файлів з авторизованим доступом, що дозволило визначити їхні сильні та слабкі сторони. На основі цього сформульовано основні вимоги та функціональні характеристики для нового веб-сервісу. Архітектура веб-сервісу розроблена з урахуванням масштабованості, продуктивності та безпеки системи.

Значну увагу приділено проектуванню та реалізації бази даних для зберігання інформації про файли та користувачів. Продумана структура бази даних забезпечує ефективну організацію та управління даними, гарантуючи їх цілісність та узгодженість.

Особливий акцент зроблено на розробці алгоритмів автентифікації, авторизації та розмежування доступу до файлів. Реалізовані механізми безпеки гарантують, що лише авторизовані користувачі можуть отримати доступ до файлів та виконувати дозволені операції, забезпечуючи конфіденційність та цілісність даних.

Інтерфейс користувача веб-сервісу відрізняється високою зручністю використання та привабливим дизайном. Він пропонує інтуїтивно зрозумілі інструменти для завантаження, зберігання, пошук та скачування файлів, а також для керування правами доступу та організації файлової структури.

Функціональність завантаження, зберігання та скачування файлів з авторизованим доступом успішно реалізована та протестована. Веб-сервіс забезпечує надійне та ефективне виконання цих операцій, гарантуючи збереження та доступність файлів для авторизованих користувачів.

Тестування веб-сервісу дозволило виявити та усунути помилки та недоліки, що покращило якість та надійність роботи системи. Розгортання веб-сервісу на сервері та налаштування його роботи успішно завершено, забезпечуючи доступність та продуктивність сервісу для користувачів.

Розроблена документація та інструкції для користувачів веб-сервісу спрощують процес ознайомлення з сервісом та ефективного використання його функціональності. Вони надають детальні пояснення та приклади використання веб-сервісу, що підвищує зручність та задоволеність користувачів.

Створений веб-сервіс зберігання файлів з авторизованим доступом має практичне застосування у різних сферах діяльності для безпечного та зручного зберігання та обміну файлами. Він також робить внесок у теоретичні основи розробки веб-сервісів та систем безпеки, пропонуючи вдосконалені рішення та підходи.

Результати дослідження підтверджують гіпотезу про те, що розробка веб-сервісу зберігання файлів з авторизованим доступом, заснованого на сучасних технологіях та стандартах безпеки, дозволяє забезпечити надійне та зручне зберігання та обмін файлами для користувачів, зберігаючи конфіденційність та цілісність даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дудзяний І.М., Пасічник В.В., Гарасим Ю.Р. Бази даних та знань: навч. посіб. Львів: Магнолія 2016, 2019. 456 с.
2. Гладун А.Я., Рогушина Ю.В. Онтологічний аналіз у Web: навч. посіб. Київ: НАН України, 2019. 302 с.
3. Гриценко В.Г., Власенко О.В., Стеценко О.О. Проектування інформаційних систем: навч. посіб. Черкаси: ЧНУ ім. Б. Хмельницького, 2019. 434 с.
4. Гушко С.В., Шубенкова І.А., Білик В.М. Проектування та розробка веб-додатків: навч. посіб. К.: Видавництво Ліра-К, 2021. 280 с.
5. Гушко С.В., Шубенкова І.А., Білик В.М. Управління ІТ-проектами: навч. посіб. Київ: Видавництво Ліра-К, 2020. 224 с.
6. Дичка І.А., Новосад А.О., Онай М.В. Тестування програмного забезпечення: навч. посіб. Київ: КПІ ім. Ігоря Сікорського, 2021. 196 с.
7. Зубик Л.В., Зубик Я.Я., Карпович І.М. Веб-технології та веб-дизайн: навч. посіб. Рівне: НУВГП, 2018. 264 с.
8. Кузьменко Б.В., Чайковська О.А., Складанний П.М. Сучасні методи та засоби розробки інформаційних систем: навч. посіб. К.: Видавництво Ліра-К, 2021. 448 с.
9. Кузьменко О.В., Чаплінський Ю.П., Ніколаєв І.В. Web-технології та web-дизайн: навч. посіб. Київ: Університет «Україна», 2021. 300 с.
10. Левикін В.М., Чалий О.В., Щербак І.О. Проектування та розробка інформаційних систем: навч. посіб. Харків: ХНЕУ ім. С. Кузнеця, 2019. 288 с.
11. Мельник О.Г., Адамів О.П., Кічор В.П. Інформаційні системи та технології в менеджменті: навч. посіб. Львів: Видавництво Львівської політехніки, 2020. 240 с.
12. Пасічник В.В., Резніченко В.А. Веб-технології: підручник. Львів: Магнолія 2006, 2018. 336 с.

13. Шаховська Н.Б., Болюбаш Ю.Я. Організація великих даних: навч. посіб. Львів: Видавництво Львівської політехніки, 2018. 200 с.
14. Дудзяний І.М., Пасічник В.В., Гарасим Ю.Р. Інтелектуальний аналіз даних: навч. посіб. Львів: Магнолія 2019, 2020. 288 с.
15. Субботін С.О. Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень: навч. посіб. Запоріжжя: ЗНТУ, 2018. 324 с.
16. Литвин В.В., Пасічник В.В., Яцишин Ю.В. Інтелектуальні системи: підручник. Львів: Новий Світ-2000, 2020. 406 с.
17. Берко А.Ю., Висоцька В.А., Рішняк І.В. Системи електронної контент-комерції: навч. посіб. Львів: Видавництво Львівської політехніки, 2018. 440 с.
18. Висоцька В.А., Чирун Л.Б., Лесько Н.Б. Інформаційні технології в управлінні туристичним бізнесом: навч. посіб. Львів: Видавництво Львівської політехніки, 2019. 256 с.
19. Харченко О.Г., Яцишин А.Ю., Ковальчук В.П. Розроблення клієнт-серверних застосувань з використанням Node.js: навч. посіб. Київ: НТУУ «КПІ ім. Ігоря Сікорського», 2021. 200 с.
20. Кунанець Н.Е., Кунанець О.О., Пасічник В.В. Консолідована інформація: підходи до створення та використання: навч. посіб. Львів: Видавництво Львівської політехніки, 2020. 276 с.
21. Андрейчиков О.О., Комар М.П., Лупенко С.А. Методи та системи штучного інтелекту: навч. посіб. Львів: Видавництво Львівської політехніки, 2019. 268 с.
22. Казаріна О.Ю., Коротенко Г.М., Коротенко Л.М. Системи штучного інтелекту: навч. посіб. Кам'янське: ДДТУ, 2021. 160 с.
23. Табунщик Г.В., Каплієнко Т.І., Петрова О.А. Проектування та моделювання програмного забезпечення сучасних інформаційних систем: навч. посіб. Запоріжжя: Дике Поле, 2018. 250 с.
24. Сидоров М.О., Костенко О.Б., Гавриленко С.Ю. Проектування програмного забезпечення інформаційно-управляючих систем: навч. посіб. Харків: НТУ «ХПІ», 2019. 272 с.

25. Мельник О.Г., Адамів О.П., Кічор В.П. Інформаційні системи та технології в менеджменті: навч. посіб. Львів: Видавництво Львівської політехніки, 2020. 240 с.
26. Шаховська Н.Б., Болюбаш Ю.Я. Організація великих даних: навч. посіб. Львів: Видавництво Львівської політехніки, 2018. 200 с.
27. Субботін С.О. Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень: навч. посіб. Запоріжжя: ЗНТУ, 2018. 324 с.
28. Литвин В.В., Пасічник В.В., Яцишин Ю.В. Інтелектуальні системи: підручник. Львів: Новий Світ-2000, 2020. 406 с.
29. Гладун А.Я., Рогушина Ю.В. Онтологічний аналіз у Web: навч. посіб. Київ: НАН України, 2019. 302 с.
30. Кузьменко О.В., Чаплінський Ю.П., Ніколаєв І.В. Web-технології та web-дизайн: навч. посіб. Київ: Університет «Україна», 2021. 300 с.
31. Пасічник В.В., Резніченко В.А. Веб-технології: підручник. Львів: Магнолія 2006, 2018. 336 с.
32. Гриценко В.Г., Власенко О.В., Стеценко О.О. Проектування інформаційних систем: навч. посіб. Черкаси: ЧНУ ім. Б. Хмельницького, 2019. 434 с.
33. Гушко С.В., Шубенкова І.А., Білик В.М. Проектування та розробка веб-додатків: навч. посіб. К.: Видавництво Ліра-К, 2021. 280 с.
34. Дудзяний І.М., Пасічник В.В., Гарасим Ю.Р. Бази даних та знань: навч. посіб. Львів: Магнолія 2019. 456 с.
35. Бондаренко С.В., Сєров Ю.О., Шабранський В.В. Проектування та розробка баз даних: навч. посіб. К.: КНУБА, 2018. 252 с.
36. Гайдаржи В.І., Кухаренко Д.В., Шаров С.В. Бази даних та інформаційні системи: навч. посіб. Мелітополь: ФОП Однорог Т.В., 2019. 220 с.
37. Глибовець А.М., Глибовець М.М., Поляков В.В. Методи та засоби розробки програмного забезпечення: навч. посіб. К.: Київський університет, 2020. 304 с.
38. Овсяк О.В., Кириченко Г.І. Бази даних та інформаційні системи. Полтава: Національний університет "Полтавська політехніка імені Юрія Кондратюка", 2018. 160 с.

39. Гладун А.Я., Рогушина Ю.В. Розробка веб-сервісів з використанням технології JWT. Київ: Інститут програмних систем НАН України, 2020. 120 с.
40. Петренко О.О., Петренко А.І. Безпека веб-додатків та захист даних. Київ: Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського", 2021. 200 с.
41. Кузьменко Б.В., Чайковська О.А. Хмарні технології та розгортання веб-додатків. Одеса: Одеський національний політехнічний університет, 2019. 160 с.
42. Волошин В.С., Жежнич П.І. Інфраструктура та інструменти розробки веб-сервісів. Львів: Національний університет "Львівська політехніка", 2020. 240 с.
43. Мельник В.А. Розробка веб-додатків з використанням фреймворку Spring Boot. Львів: Видавництво Львівської політехніки, 2019. 192 с.
44. Глибовець А.М., Любченко К.О. Методи та засоби розробки веб-застосунків. Київ: Наукова думка, 2018. 240 с.
45. Кузьменко О.В., Чаплінський Ю.П. Сучасні технології розробки веб-сервісів. Черкаси: Черкаський державний технологічний університет, 2019. 180 с.
46. Яцишин В.В., Яцишин В.О. Розробка та використання RESTful веб-сервісів. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2020. 220 с.
47. Бойко І.В., Хрущ В.К. Проектування та розробка баз даних. Львів: Видавництво Львівської політехніки, 2019. 280 с.
48. Казарін О.В., Куратченко А.Г., Бондаренко С.В. Безпека баз даних: навч. посіб. К.: НАУ, 2020. 180 с.
49. Пасічник В.В., Шестакевич Т.В., Кунанець Н.Е. Технології баз даних та знань: навч. посіб. Львів: Видавництво Львівської політехніки, 2018. 404 с.
50. Гладун А.Я., Рогушина Ю.В. Розробка веб-сервісів з використанням технології JWT. Київ: Інститут програмних систем НАН України, 2020. 120 с.

51. Петренко О.О., Петренко А.І. Безпека веб-додатків та захист даних. Київ: Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського", 2021. 200 с.
52. Кузьменко Б.В., Чайковська О.А. Хмарні технології та розгортання веб-додатків. Одеса: Одеський національний політехнічний університет, 2019. 160 с.
53. Волошин В.С., Жежнич П.І. Інфраструктура та інструменти розробки веб-сервісів. Львів: Національний університет "Львівська політехніка", 2020. 240 с.
54. Мельник В. А. Розробка веб-додатків з використанням фреймворку Spring Boot. Львів: Видавництво Львівської політехніки, 2019. 192 с.
55. Глибовець А. М., Любченко К. О. Методи та засоби розробки веб-застосунків. Київ: Наукова думка, 2018. 240 с.
56. Кузьменко О. В., Чаплінський Ю. П. Сучасні технології розробки веб-сервісів. Черкаси: Черкаський державний технологічний університет, 2019. 180 с.
57. Яцишин В. В., Яцишин В. О. Розробка та використання RESTful веб-сервісів. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2020. 220 с.
58. Бойко І. В., Хрущ В. К. Проектування та розробка баз даних. Львів: Видавництво Львівської політехніки, 2019. 280 с.
59. Казарін О. В., Куратченко А. Г., Бондаренко С. В. Безпека баз даних: навч. посіб. К.: НАУ, 2020. 180 с.
60. Пасічник В. В., Шестакевич Т. В., Кунанець Н. Е. Технології баз даних та знань: навч. посіб. Львів: Видавництво Львівської політехніки, 2018. 404 с.
61. Субботін С. О. Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень: навч. посіб. Запоріжжя: ЗНТУ, 2018. 324 с.
62. Литвин В. В., Пасічник В. В., Яцишин Ю. В. Інтелектуальні системи: підручник. Львів: Новий Світ-2000, 2020. 406 с.

ДОДАТОК А
ТЕКСТ ПРОГРАМИ

Бекенд

Folders Service:

```
@Injectable()
export class FoldesService extends EntityService<FolderEntity> {
  constructor(
    @InjectRepository(FolderEntity)
    readonly _repository: Repository<FolderEntity>
  ) {
    super();
  }
  async updateFolderPath(id: number) {
    const folder = await this.findOne({
      where: { id },
      relations: ['parent'],
      select: { id: true, name: true, parent: { path: true, name: true } }
    });
    if (!folder) throw new NotFoundException('File not found');
    const path = generatePath(folder.parent, folder);
    await this.updateOne({ where: { id } }, { path });}
  async createFolder(folder: CreateFolderDto, userId: number): Promise<FolderEntity>{
    const exist = await this.findOne({ where: { name: folder.name, parentId: folder.parentId } });
    if (exist) throw new BadRequestException('Folder already exist');
    const savedFile = await this._repository.save(cleanFilters({ ...folder, userId }));
    if (!savedFile) {
      throw new BadRequestException('Couldn`t save folder');
    }

    return savedFile;
  }
  async getFolderById(id: number): Promise<any> {
    return await this.findOne({ where: { id }, relations: ['files'] });
  }
  async getFolderPath(id: number): Promise<any> {
```

```
  console.log('getFolderPath', id)
  if(!id) return [{ path: '/', name: '' }]
  let folder = await this.findOne({ where: { id }, relations: ['parent'] });
  if(!folder?.parentId) return [{ path: '/', name: folder?.name }]
  const path: { path: number; name: string }[] = [];
  while (folder.id !== 1) {
     path.unshift({ path: folder.id, name: folder.name });
    folder = await this.findOne({
      where: { id: folder.parent?.id },
      relations: ['parent'],
      select: { id: true, path: true, name: true, parent: { id: true, path: true, name: true } }
    });
  }
  return path;
}
async getFolders(userId: number, body: GetFilesDto) {
  const { dateFrom, dateTo, search, page = 0, parentId } = body;
  const query = this.queryBuilder.where(
    cleanFilters({ userId, ...(search ? {} : { parentId: parentId ?  parentId : IsNull() }) })
  );
   if (dateFrom && dateTo) {
    query.andWhere('FolderEntity.dateModified BETWEEN :dateFrom AND :dateTo', {
      dateFrom: new Date(dateFrom),
      dateTo: new Date(dateTo)
    });
  }
  if (search) {
    const searchFilters = ['FolderEntity.name'];
    const searchQuery = generateSearchFilterQueryStrong(search, searchFilters);
    query.andWhere(searchQuery);
  }
  query.orderBy({ 'FolderEntity.dateModified': 'DESC' });
  query.skip(page * 100).take(100);
  const folders = await query.getMany();
   if (!folders) {
    throw new NotFoundException('File not found');
  }
  return folders;
}
```

```
}
```

Сервіс для взаємодії з aws s3 для реалізації логіки збереження файлів, видалення та оновлення:

```
@Injectable()
export class StorageService {
  private readonly client = new S3Client({
    credentials: {
      accessKeyId: this.configService.get('aws.accessKeyId'),
      secretAccessKey: this.configService.get('aws.secretAccessKey')
    },
    region: this.configService.get('aws.region')
  });
  private readonly logger = new Logger(StorageService.name);
  constructor(private readonly configService: ConfigService) { }

  private getBucket(bucketKey: IWithConfiguredBucket) {
    return bucketKey.Bucket || this.configService.get(`s3.buckets.${bucketKey.BucketKey}`);
  }

  generateGetPresignedUrl(input: Omit<GetObjectCommandInput, 'Bucket'> & IWithConfiguredBucket) {
    const Bucket = this.getBucket(input);
    const command = new GetObjectCommand({ Bucket, ...input });

    this.logger.log(`Generating Getting Presigned URL for "${Bucket}/${input.Key}"`);

    return getSignedUrl(this.client, command);
  }

  updateObject(input: Omit<PutObjectCommandInput, 'Bucket'> & IWithConfiguredBucket) {
    const Bucket = this.getBucket(input);
    const command = new PutObjectCommand({ Bucket, ...input });

    this.logger.log(`Generating Updating Presigned URL for "${Bucket}/${input.Key}"`);

    return getSignedUrl(this.client, command);
  }
```

```
  deleteObject(input: Omit<DeleteObjectCommandInput, 'Bucket'> & IWithConfiguredBucket) {
    const Bucket = this.getBucket(input);
    const command = new DeleteObjectCommand({ Bucket, ...input });

    this.logger.log(`Generating Deleting Presigned URL for "${Bucket}/${input.Key}"`);
    return getSignedUrl(this.client, command);
  }

  deleteObjects(input: Omit<DeleteObjectsCommandInput, 'Bucket'> & IWithConfiguredBucket) {
    const Bucket = this.getBucket(input);
    const command = new DeleteObjectsCommand({ Bucket, ...input });
    this.logger.log(`Generating Deleting Presigned URL for "${Bucket}/${input.Delete.Objects}"`);

    return getSignedUrl(this.client, command);
  }
  generateUploadPresignedUrl(input: Omit<PutObjectCommandInput, 'Bucket'> & IWithConfiguredBucket) {
    const Bucket = this.getBucket(input);
    const command = new PutObjectCommand({ Bucket, ...input });
    this.logger.log(`Generating Uploading Presigned URL for "${Bucket}/${input.Key}"`);
    return getSignedUrl(this.client, command);
  }
}
```

Auth Service:

```
@Injectable()
export class AuthService {
  constructor(
    private jwtService: JwtService,
    private userService: UsersService,
  ) { }
  async login(body: LoginUserDto) {
    const { password, email } = body;
    const user = await this.userService.findOne({
      where: { email },
      select: ['password', 'email', 'id', 'category']
    });
    if (password !== user.password) {
      throw new BadRequestException('Invalid credentials');
```

```
  }
  const tokensPair = await this.getTokensPair({
    sub: +user.id,
    email: user.email,
    role: user.category as Role
  });
  await this.userService.updateRefreshToken(user.id, tokensPair.refreshToken);
  return tokensPair;
}
async register(body: RegisterUserDto) {
  const { email, captchaToken, firstName, lastName, password } = body;
  const userExists = await this.userService.findOne({
    where: { email },
    select: ['email']
  });
  if (userExists) {
    throw new BadRequestException('Check credentials');
  }
  await this.validateCaptcha(captchaToken);
  const newUser = await this.userService.create({
    email,
    firstName,
    lastName,
    password
  });
  return newUser;
}
async getTokensPair(payload: ITokenPayload) {
  const [accessToken, refreshToken] = await Promise.all([
    this.jwtService.signAsync(payload, {
      secret: config.jwt.accessSecret,
      expiresIn: config.jwt.accessExpiresIn
    }),
    this.jwtService.signAsync(payload, {
      secret: config.jwt.refreshSecret,
      expiresIn: config.jwt.refreshExpiresIn
    })
  ]);
```

```
    return { accessToken, refreshToken };
  }

  async refreshTokens(userId: number, refreshToken: string) {
    const user = await this.userService.getUserIfRefreshMatches(userId, refreshToken);

    const tokenPayload: ITokenPayload = {
      sub: user.id,
      email: user.email,
      role: user.category as Role
    };
    const tokensPair = await this.getTokensPair(tokenPayload);
    await this.userService.updateRefreshToken(user.id, tokensPair.refreshToken);
    return tokensPair;
  }

  async validateCaptcha(recaptcha: string): Promise<void> {
    const recapchaKey = config.captcha.secretKey;

    const verificationURL =
`https://www.google.com/recaptcha/api/siteverify?secret=${recapchaKey}&response=${recaptcha}`;
    const verificationResponse = await axios.get(verificationURL);
    if (!verificationResponse.data.success) {
      throw new BadRequestException(ExceptionMessage.BAD_REQUEST.INVALID_RECAPTCHA);
    }
  }
}
```

Фронтенд

Форма для завантаження файлу:

```
const FileUploadForm: React.FC<AccountFormProps> = ({ isLoading, userId, onSuccess, folderId }) => {
  const [fileUploading, setFileUploading] = useState(false)
  const handleFileUpload = async ({ target: { files } }: ChangeEvent<HTMLInputElement>) => {
    console.log('userfolderId', folderId)

    const file = files?.[0]
    if (!file) return
```

```
  try {
    setFileUploading(true)
    await filesService.uploadFile(file, folderId)
    toast({ title: 'Файл успішно завантажений' })
    onSuccess && onSuccess()
  } catch {
    toast({
      variant: 'destructive',
      title: 'Не вийшло завантажити файл',
      description: 'Спробуйте пізніше'
    })
  } finally {
    setFileUploading(false)
  }
  mutate([FetchFilesKey.FILES])
}
return (
  <Alert className='mt-5 py-5'>
    <SkeletonWrapper
      className='h-7 max-w-sm mt-3'
      wrapperClassName='space-y-5'
      quantity={1}
      active={!userId || isLoading}
    >
      <div className='grid w-full max-w-sm items-center gap-1.5 mt-3 h-[400px]'>
        {fileUploading ? (
          <div className='flex mt-3 w-[100px] h-7'>
            <Spinner />
          </div>
        ) : (
          <Input
            id='ID'
            type='file'
            onChange={handleFileUpload}
            placeholder='Вибрати файл на комп`ютері'
            className='min-h-[400px] min-w-[400px] flex items-center justify-center'
          />
        )}
      </div>
```

```
      </SkeletonWrapper>
    </Alert>
  )
}

export default FileUploadForm
```

FileService для взаємодії з бекендом:

```
export enum FetchFilesKey {
  FILES = 'FILES'
}

const endpoint = `files`

class FilesService {
  async fetchFiles(filters?: FileStore['filters'], headers?: RequestInit['headers']) {
    if (filters?.category === DirectoryType.Folder) return
    const query = filters ? qs.stringify(filters) : ''
    const pathname = query ? `${endpoint}?${query}` : endpoint
    return await API.get<any[]>(pathname, headers && { headers })
  }

  async fetchFile(id: number, headers?: RequestInit['headers']) {
    const res = await API.get<any>(`${endpoint}/${id}`, headers && { headers })
    return res.url
  }

  async createFile(data: any, headers?: RequestInit['headers']) {
    return await API.post<File>(endpoint, data, headers && { headers })
  }

  async uploadFile(file: Blob, folderId?: number, headers?: RequestInit['headers']) {
    const { url, key } = await API.post<any>(`${endpoint}/upload`, headers && { headers })
    const data = await uploadFileWithPresignedUrL(url, file)
    console.log('Uploaded File Data', data)

    const body = {
      name: file.name,
```

```
      size: file.size,
      prototype: file.type,
      url: key,
      ...(folderId ? { folderId } : {})
    }
    await filesService.createFile(body)
  }

  async shareFile(id: string, data: { email: string }, headers?: RequestInit['headers']) {
    return await API.put<any>(`${endpoint}/${id}/share`, data, headers && { headers })
  }

  async getShareFiles(headers?: RequestInit['headers']) {
    return await API.get<any>(`${endpoint}/shared`, headers && { headers })
  }

  async updateFile(id: string, data: Partial<any>, headers?: RequestInit['headers']) {
    return await API.patch<any>(`${endpoint}/${id}`, data, headers && { headers })
  }

  async deleteFile(id: string, headers?: RequestInit['headers']) {
    return await API.delete<any>(`${endpoint}/${id}`, undefined, headers && { headers })
  }
  async deleteFiles(id: string, headers?: RequestInit['headers']) {
    return await API.delete<any>(`${endpoint}/${id}`, undefined, headers && { headers })
  }
}
export const filesService = new FilesService()
```

Сторінка з файлами та папками:

```
const FilesPage = ({ slug }: { slug: string }) => {
  const router = useRouter()
  const patchFilters = useFileStore(state => state.patchFilters)
  const session = useSession()
  const userId = get(session, 'data.user.id', null)
  const { data: user, isLoading: userLoading } = useSWR(userId && FetchUsersKey.PROFILE, {
    fetcher: () => userService.fetchProfile()
  })
```

```
const openConfirmDialog = useAppDialogsStore(state => state.setConfirmDialog)

const [details, setDetails] = useState<any>()
const [file, setFile] = useState<any>()

const { filters, orderId, setOrderId } = useFileStore(state => state)

const { data: folders, mutate: mutateFolders, ...foldersParams } = useSWR(
  [FetchFoldersKey.FOLDERS, filters, slug],
  () => foldersService.fetchFolders(filters),
  {
    onError: err => console.error('Error fetching folders:', err)
  }
)
const { data: files, mutate: mutateFiles, ...filesParams } = useSWR(
  [FetchFilesKey.FILES, filters, slug],
  () => filesService.fetchFiles(filters)
)

const mutateData = () => {
  mutateFiles()
  mutateFolders()
}

const changeDetails = (details?: any) => () => {
  orderId && setOrderId(null)
  setDetails(details)
  mutateData()
}

const changeFile = (details?: any) => () => setFile(details)

const foldersAndFiles = useMemo(
  () => [
    ...(Array.isArray(folders) ? folders : []),
    ...(Array.isArray(files) ? files : []),
    ...(filters?.search ? vides : [])
  ],
  [folders, files]
```

```
)

const deleteFile = async (file: any) => {
  if (file.isFile) {
    await filesService.deleteFile(file.id)
    mutateFiles()
  } else {
    await foldersService.deleteFolder(file?.id)
    mutateFolders()
  }
}

const handleDelete = (file: any) => () => {
  openConfirmDialog({
    description: `Ви впевнені, що бажаєте видалити ${file.isFile ? 'файл' : 'папку'}?`,
    confirmText: 'Видалити',
    onConfirm: () => deleteFile(file),
    title: 'Видалити'
  })
}

const fetchFileUrl = (id: number) => async () => {
  const url = await filesService.fetchFile(id)
  window.open(url, '_blank')
}

useEffect(() => {
  if (!isNumeric(slug)) router.replace('/account/dashboard')
  patchFilters({ folderId: isNumeric(slug) ? +slug : undefined })
}, [patchFilters, router, slug])

if (!userId || !user || userLoading) return <RandomLoader />

return (
  <main className='pt-5 w-full'>
    <DetailsForm folder={details} onClose={changeDetails()} />
    <ShareForm file={file} onClose={changeFile()} />
    <Header user={user} mutate={mutateData} length={foldersAndFiles.length} />
    {filesParams.isLoading || foldersParams.isLoading ? (
```

```
  <DropLoader />
) : foldersAndFiles?.length ? (
  <div className='w-full'>
    <Table className='mt-10'>
      <TableHeader>
        <TableRow>
          <TableHead>
            <Checkbox
              checked={
                table.getIsAllPageRowsSelected() ||
                (table.getIsSomePageRowsSelected() && "indeterminate")
              }
              onCheckedChange={(value) => table.toggleAllPageRowsSelected(!!value)}
              aria-label='Select all'
            />
          </TableHead>
          <TableHead>Назва</TableHead>
          <TableHead>Ви змінювали</TableHead>
          <TableHead>Розмір</TableHead>
          <TableHead>Редагування</TableHead>
        </TableRow>
      </TableHeader>
      <TableBody>
        {foldersAndFiles.map(item => (
          <FolderItem
            key={item.id}
            item={item}
            onEdit={changeDetails(item)}
            onDelete={handleDelete(item)}
            openFile={fetchFileUrl(item.id)}
            onShare={changeFile(item)}
          />
        ))}
      </TableBody>
    </Table>
  </div>
) : (
  <FilesEmptyPlaceholder />
)}
```

```
    </main>
  )
}

export default FilesPage
```