

Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

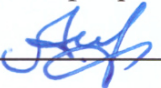
ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра
на тему: «Проект безпечного файлового сховища
організації»

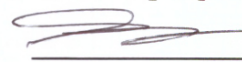
Виконав: студент 4-го курсу групи КІ-21-1
спеціальності 123 «Комп'ютерна інженерія»

 Ілля УШАКОВ

Керівник: Доцент кафедри КІ та КБ, к.т.н.,

 Олексій ШЕЛУХА

Рецензент: Доцент кафедри КІ та КБ, к.т.н., доцент

 Юрій РОССІНСЬКИЙ

Житомир – 2025

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної
інженерії та кібербезпеки

 Андрій ЄФІМЕНКО

3 березня 2025 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу бакалавра

Студент: Ушаков Ілля Вікторович

Тема роботи: «Проект безпечного файлового сховища організації»

затверджена Наказом університету від 03 березня 2025 р. № 79/с

Термін здачі студентом закінченої роботи 9 червня 2025 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

- засоби реалізації безпечного вебзастосунку для зберігання, обміну та резервного копіювання файлів.
- хмарна інфраструктура вебзастосунку з підтримкою доступу по ролях і під-розділах.

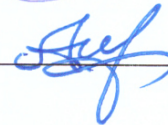
Обладнання та програмне забезпечення:

1. Сервер на базі Amazon EC2
2. Next.js 14
3. GitHub
4. Amazon S3

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

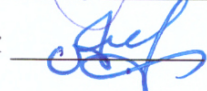
Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1-3	Фальковський І.Г.	05.03.2025	05.03.2025 

Студент  Ілля УШАКОВ

Керівник  Олексій ШЕЛУХА

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	03.03.2025	Виконано
2	Аналіз аналогічних рішень та вибір інструментів реалізації	10.03.2025	Виконано
3	Проектування структури бази даних та сховища файлів	21.03.2025	Виконано
4	Реалізація механізму автентифікації та авторизації	03.04.2025	Виконано
5	Створення інтерфейсу користувача та реалізація UI-функціоналу	10.04.2025	Виконано
6	Реалізація API-ендпоінтів для роботи з файлами та папками	20.04.2025	Виконано
7	Налаштування сховища Amazon S3, реалізація підписаних URL	25.04.2025	Виконано
8	Розгортання проєкту на EC2-сервері, налаштування PM2	25.04.2025	Виконано
9	Організація CI/CD через GitHub Actions	15.05.2025	Виконано
10	Налаштування моніторингу CloudWatch та резервного копіювання	20.05.2025	Виконано
11	Тестування системи, підготовка висновків і пояснювальної записки	25.05.2025	Виконано
12	Оформлення пояснювальної записки	09.06.2025	Виконано

Студент  Ілля УШАКОВ
Керівник  Олексій ШЕЛУХА

АНОТАЦІЯ

У кваліфікаційній роботі бакалавра представлено розробку приватного хмарного файлового сховища організації з можливістю розмежування окремих підрозділів компанії, завантаження файлів, керування доступом та резервного копіювання. Пояснювальна записка містить 70 сторінок, у тому числі 45 рисунків, 5 таблиці та 3 додатки, розміщених на 6 сторінках. Список використаних джерел складається з 16 найменувань.

Метою кваліфікаційної роботи є розробка проєкту безпечного хмарного файлового сховища організації із підтримкою автентифікації, авторизації, ролей користувачів та інтеграцією із хмарною інфраструктурою Amazon Web Services із організацією доступу у вигляді вебзастосунку..

У ході виконання роботи:

- проаналізовано базові рішення для автентифікації, зберігання даних та безпеки в хмарі;
- розглянуто особливості роботи з Amazon S3, MongoDB та EC2;
- показано розробку вебінтерфейсу на базі Next.js з авторизацією через JWT та підтвердженням сесії;
- наведено результати впровадження CI/CD-процесу, моніторингу CloudWatch та механізму резервного копіювання через S3-версіонування.

КЛЮЧОВІ СЛОВА: ХМАРНЕ СХОВИЩЕ, ХМАРНА МЕРЕЖЕВА ІНФРАСТРУКТУРА, АВТОРИЗАЦІЯ, СИСТЕМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ, АВТОМАТИЗАЦІЯ, ПІДПИСАНІ URL, ФАЙЛОВИЙ МЕНЕДЖЕР

					ККІК.КР.Б – 123 – 15 – ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата					
Розроб.		Ушаков І.В.		23.06.2025	ПРОЄКТ БЕЗПЕЧНОГО ФАЙЛОВОГО СХОВИЩА ОРГАНІЗАЦІЇ				
Керівник		Шелуха О.О.		25.06.25					
Реценз.		Росінський Ю.М.		25.06.25	ФІКТ, Гр. КІ-21-1				
Н. контр.		Шелуха О.О.		25.06.25					
Затверд.		Сфіменко А.А.		25.06.25					
					Літ.	Арк.	Аркушів		
						4	69		

ANNOTATION

The bachelor's qualification work presents the development of a private cloud-based file storage system for an organization, featuring the separation of company departments, file uploading, access control, and data backup capabilities. The explanatory report contains 70 pages, including 45 figures, 5 tables, and 3 appendices placed over 6 pages. The list of references includes 16 sources.

The purpose of the qualification work is to develop a project for a secure cloud file storage for an organization with support for authentication, authorization, user roles, and integration with the Amazon Web Services cloud infrastructure with access organized as a web application.

During the course of the work:

- basic solutions for authentication, data storage, and cloud security were analyzed;
- the specifics of working with Amazon S3, MongoDB, and EC2 were considered;
- the development of a web interface based on Next.js with JWT-based authorization and session validation was demonstrated;
- the results of implementing a CI/CD pipeline, CloudWatch monitoring, and a backup mechanism using S3 versioning were presented.

KEYWORDS: CLOUD STORAGE, CLOUD NETWORK INFRASTRUCTURE, AUTHORIZATION, DATABASE MANAGEMENT SYSTEMS, AUTOMATION, SIGNED URL, FILE MANAGER

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	7
ВСТУП.....	8
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ТА ОБҐРУНТУВАННЯ РОЗРОБКИ	
ХМАРНОГО ФАЙЛОВОГО СХОВИЩА.....	10
1.1 Аналіз існуючих аналогів систем хмарного зберігання файлів.....	11
1.2 Аналіз засобів розробки хмарного сховища	14
1.3 Аналіз принципів організації сховища файлів	17
1.4 Аналіз інструментів для автоматизації обміну та збереження даних	20
1.5 Аналіз засобів моніторингу та збору метрик.....	25
Висновки до розділу 1	27
РОЗДІЛ 2 ПРОЄКТУВАННЯ ХМАРНОГО ФАЙЛОВОГО СХОВИЩА	28
2.1 Архітектура та структура майбутнього програмного комплексу.....	28
2.2 Розробка структури бази даних.....	32
2.3 Моделювання системи доступу користувачів	34
2.4 Інфраструктура доступу, моніторингу та резервного копіювання.....	37
2.5 Автоматизація розгортання та моніторинг інфраструктури	40
Висновки до розділу 2	42
РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ	44
3.1 Створення проєкту.....	44
3.2 Розгортання застосунку.....	45
3.3 Налаштування збереження файлів.....	46
3.4 Тестування працездатності розробленого додатку	47
Висновки до розділу 3	60
ВИСНОВКИ	61
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	63
ДОДАТКИ.....	64

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface, інтерфейс прикладного програмування

AWS – Amazon Web Services, хмарна платформа для розгортання та обслуговування застосунків

Bcrypt – бібліотека для хешування паролів

Bucket – логічне сховище файлів в Amazon S3

CI/CD – Continuous Integration / Continuous Deployment, безперервна інтеграція та деплой

CloudWatch – сервіс AWS для моніторингу ресурсів

CPU – Central Processing Unit, центральний процесор

DB – Database, база даних

Deploy – процес розгортання застосунку

EC2 – Elastic Compute Cloud, сервіс AWS для віртуальних серверів

JSON – JavaScript Object Notation, формат зберігання даних

JWT – JSON Web Token, формат передачі авторизаційної інформації

Lifecycle Rule – правило життєвого циклу файлів у S3

Next.js – фреймворк для React з підтримкою SSR і маршрутизації

S3 – Simple Storage Service, сервіс для зберігання об'єктів в AWS

Session – сесія користувача, авторизований сеанс роботи

Signed URL – підписане посилання, тимчасовий захищений доступ до ресурсу

SSH – Secure Shell, протокол віддаленого доступу

Telegram Bot – автоматизований бот для Telegram, використаний для нотифікацій

UI – User Interface, інтерфейс користувача

URL – Uniform Resource Locator, уніфікований ідентифікатор ресурсу (посилання)

UX – User Experience, досвід користувача

YAML – Yet Another Markup Language, формат конфігураційних файлів

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. У сучасних умовах цифровізації підприємств питання безпечного зберігання та доступу до даних є надзвичайно актуальним. Компанії та організації щодня оперують великою кількістю внутрішньої документації, яка повинна бути не лише доступною співробітникам, але й надійно захищеною від несанкціонованого доступу, втрати або компрометації. Вибір теми обумовлений потребою впровадження власного хмарного файлового сховища, що забезпечує контроль за доступом, зберіганням, а також безпекою даних без використання сторонніх комерційних сервісів, таких як Google Drive чи Dropbox. В межах роботи вирішується проблема організації приватного хмарного середовища з підтримкою автентифікації та авторизації, керування правами доступу та базового функціоналу файлового менеджера.

Метою кваліфікаційної роботи є розробка проєкту безпечного хмарного файлового сховища організації із підтримкою автентифікації, авторизації, ролей користувачів та інтеграцією із хмарною інфраструктурою Amazon Web Services із організацією доступу у вигляді вебзастосунку. Визначена мета обумовлює наступні завдання:

- провести аналіз існуючих рішень для організації хмарного зберігання файлів;
- розробити проєкт безпечного хмарного файлового сховища, зокрема реалізувати серверну та клієнтську частину вебзастосунку, налаштувати автентифікацію та авторизацію користувачів;
- реалізувати функціонал керування файлами та папками через зручний вебінтерфейс, налаштувати систему для автоматичного розгортання додатку та забезпечити інтеграцію з хмарним сховищем;
- провести тестування працездатності системи та її стійкості до типових загроз.

Об'єктом дослідження є процес організації безпечного хмарного середовища для зберігання та керування файлами в межах внутрішніх потреб організації.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

Предметом дослідження є технології побудови вебзастосунків (Next.js, Node.js, MongoDB), засоби забезпечення безпеки автентифікації (JWT, bcrypt), та сервіси хмарної інфраструктури (AWS S3, EC2, CI/CD через GitHub Actions), що використовуються для реалізації безпечного файлового сховища

У процесі виконання кваліфікаційної роботи було розроблено проект безпечного хмарного вебсховища, що дозволяє користувачам авторизуватись, створювати папки, завантажувати, переглядати та видаляти файли через зручний вебінтерфейс.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ТА ОБҐРУНТУВАННЯ РОЗРОБКИ ХМАРНОГО ФАЙЛОВОГО СХОВИЩА

У сучасних умовах стрімкого розвитку цифрових технологій усе більше організацій переходять до використання хмарних рішень для зберігання, обробки та обміну даними. Такий підхід дозволяє забезпечити високу доступність інформації, масштабованість інфраструктури та економію ресурсів. Водночас широке впровадження хмарних технологій створює нові виклики у сфері захисту даних, що зумовлює підвищений інтерес до розробки безпечних систем зберігання інформації.

Особливої актуальності ця тема набуває у контексті побудови внутрішніх корпоративних рішень для зберігання файлів, які мають підвищені вимоги до безпеки, розмежування доступу, контролю дій користувачів та відповідності внутрішнім політикам організацій. Існуючі комерційні сервіси, як-от Google Drive або Dropbox, часто не враховують специфіку внутрішніх потреб компаній, пов'язаних з контролем над інфраструктурою, вимогами до локального хостингу або інтеграції з внутрішніми системами управління.

В рамках переддипломної практики було досліджено функціонування кількох існуючих рішень у середовищі організації, зокрема Nextcloud, Seafile та внутрішніх самописних сховищ. Аналіз виявив, що попри широкий функціонал, більшість із них мають обмежену гнучкість налаштування прав доступу, складність в інтеграції з кастомними системами автентифікації, або ж потребують високих технічних ресурсів для розгортання.

У зв'язку з цим, розробка кастомізованого хмарного файлового сховища, побудованого на сучасному стеку технологій (Next.js, MongoDB, AWS S3), з можливістю гнучкого управління ролями, створення підрозділів, персональних та спільних сховищ, є актуальним та обґрунтованим завданням. Такий підхід дозволяє не лише забезпечити збереження конфіденційної інформації, а й адаптувати систему під конкретні потреби організації, не покладаючись на сторонні сервіси.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

Таким чином, тема кваліфікаційної роботи є актуальною з точки зору потреб сучасного ринку, вимог до інформаційної безпеки та необхідності створення масштабованих, ефективних і безпечних хмарних рішень.

1.1 Аналіз існуючих аналогів систем хмарного зберігання файлів

На сучасному ринку представлено значну кількість рішень для організації хмарного зберігання файлів. Ці системи відрізняються за функціональністю, архітектурою, модельністю розгортання та рівнем захисту даних. Для обґрунтування потреби у створенні нового рішення доцільно провести аналіз найпоширеніших аналогів, таких як Google Drive, Dropbox, Nextcloud та Seafile (табл.1.1).

Google Drive є одним із найпопулярніших SaaS-рішень для зберігання та обміну файлами. Його переваги полягають у високій доступності, глибокій інтеграції з іншими сервісами Google (Docs, Sheets, Gmail) та простоті використання. Проте цей сервіс є хмарним виключно у сторонній інфраструктурі (GCP), що унеможливає повний контроль над даними. Також, він не призначений для внутрішніх корпоративних систем із розмежованими зонами доступу, кастомними ролями чи інтеграцією з LDAP.

Dropbox також є комерційним сервісом із широкими можливостями для зберігання та синхронізації файлів. Він підтримує базове керування правами доступу, історію версій та спільну роботу з документами. Як і у випадку з Google Drive, дані користувачів зберігаються на сторонніх серверах, а кастомізація поведінки системи або логіки автентифікації суттєво обмежена.

Nextcloud – це відкрите самохостингове рішення, яке забезпечує високу гнучкість налаштувань, включаючи інтеграцію з LDAP, підтримку ролей, розширення через плагіни, а також функції відеоконференцій, перегляду документів тощо. Основна перевага – можливість повного контролю над даними, розміщення в приватній інфраструктурі. Проте Nextcloud вимагає значних ресурсів для розгортання, а складна архітектура ускладнює підтримку та розробку нових функцій.

Seafile – ще одне потужне самохостингове рішення, що робить акцент на синхронізації даних, високу продуктивність та захищене шифрування. Інтерфейс у

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

ньому більш мінімалістичний порівняно з Nextcloud, проте реалізація кастомних логік (наприклад, складних моделей підрозділів і ролей) вимагає глибокого доопрацювання.

Таблиця 1.1

Порівняльна таблиця аналогів

Характеристика	Google Drive	Dropbox	Nextcloud	Seafile
Тип	SaaS	SaaS	Self-hosted	Self-hosted
Контроль над даними	Низький	Низький	Повний	Повний
Підтримка ролей	Базова	Базова	Розширена	Обмежена
Підтримка підрозділів	Відсутня	Відсутня	Через додатки	Вручну
Інтеграція з LDAP	Ні	Ні	Так	Так
Гнучкість кастомізації	Низька	Низька	Висока	Середня
Простота розгортання	-	-	Складна	Помірна

Таким чином, аналіз існуючих аналогів показує, що жодне з рішень повною мірою не задовольняє вимоги до розробки гнучкої, кастомізованої системи для організації захищеного внутрішнього сховища файлів з підтримкою підрозділів, ролей, гнучкого доступу та простоти розгортання. Це створює підґрунтя для розробки власного рішення, орієнтованого на конкретні задачі.

1.1.1 Особливості та очікувані переваги розроблюваної системи

Запропоноване рішення покликане подолати обмеження типових хмарних файлових сховищ і забезпечити потреби організацій зі складною структурою прав доступу (рис. 1.1).

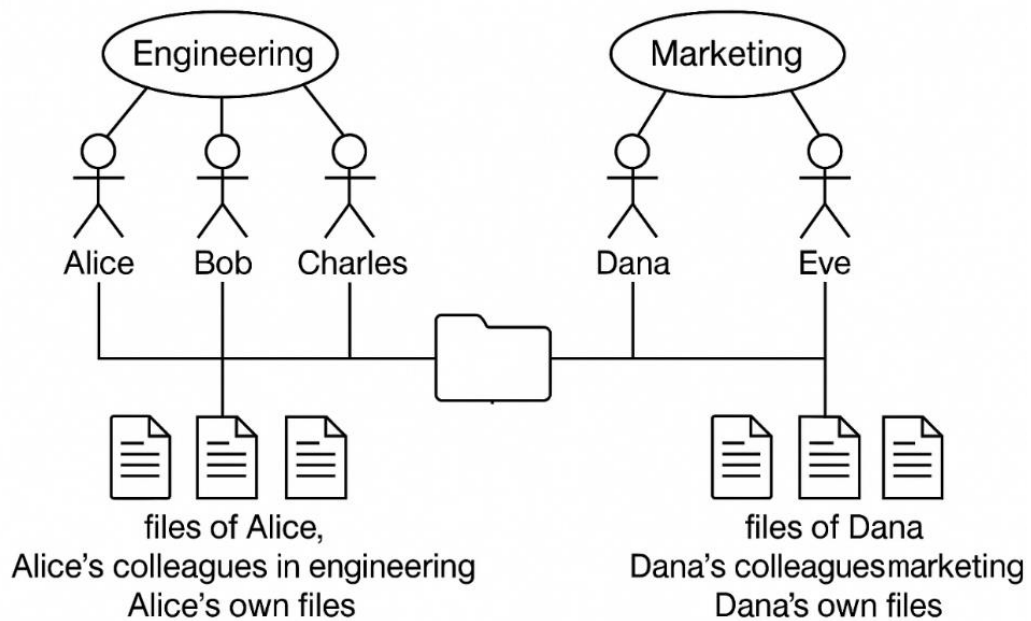


Рисунок 1.1 – Схема видимості окремих підрозділів

Ключовою перевагою є підтримка підрозділів: кожен користувач закріплюється за певним відділом, а файли можуть бути як особистими, так і доступними всім членам підрозділу.

Система ролей реалізована гнучко — адміністратор має змогу створювати підрозділи, призначати права, видаляти файли та керувати структурою папок. Побудований інтерфейс забезпечує інтуїтивність, адаптивність і привабливий вигляд інтерфейсу на будь-якому пристрої. З точки зору безпеки, застосунок використовує JWT-токени для автентифікації сесій та bcrypt для хешування паролів, що дозволяє реалізувати захищену авторизацію без залучення зовнішніх сервісів.

Уся система розроблена як єдиний Next.js-проект, у якому поєднано frontend і backend, що значно спрощує як CI/CD-процеси, так і розгортання на таких платформах, як AWS EC2 [\[13\]](#) або Amplify. Завдяки цій архітектурі рішення також легко масштабується — наприклад, шляхом переносу на Kubernetes або поділу на мікросервіси в майбутньому. Користувачам доступна зручна структура папок: вони можуть створювати власні каталоги, додавати файли, а також бачити файли своїх колег у межах одного підрозділу, що робить співпрацю в команді ефективнішою.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

Таким чином, створення власної системи дозволить досягти балансу між безпекою, гнучкістю та простотою використання, що рідко зустрічається у готових SaaS-рішеннях або комерційних пропозиціях.

1.2 Аналіз засобів розробки хмарного сховища

У процесі розробки хмарного файлового сховища було використано низку сучасних бібліотек, які забезпечили безпечну, ефективну та масштабовану архітектуру. Для створення інтерфейсу було застосовано Tailwind CSS — утилітарний CSS-фреймворк, який дозволив будувати адаптивний UI без потреби в написанні власного CSS-коду. Це значно пришвидшило розробку та забезпечило чисту й зручну структуру стилів. Взаємодія з базою даних MongoDB реалізована за допомогою бібліотеки Mongoose, яка є об'єктно-документним мапером (ODM). Вона дала змогу описати схеми даних, налаштувати валідацію, зв'язки між моделями й реалізувати CRUD-операції з колекціями користувачів, файлів і папок у зручному форматі.

З метою забезпечення безпечної автентифікації було інтегровано бібліотеку bcrypt, яка відповідає за хешування паролів: замість зберігання пароля у відкритому вигляді, система зберігає лише хеш, що значно підвищує рівень безпеки користувачьких даних. У парі з нею працює jsonwebtoken — бібліотека, яка дозволяє створювати та перевіряти JSON Web Token (JWT). Саме JWT використовується як основний механізм автентифікації: після входу користувач отримує токен, що зберігається на клієнті й передається з кожним запитом у заголовках. Сервер перевіряє цей токен, щоб визначити, чи має користувач доступ до ресурсу.

Для роботи з хмарною інфраструктурою, зокрема зі сховищем Amazon S3, у проєкті використано AWS SDK — набір бібліотек, що забезпечують інтеграцію з сервісами Amazon Web Services. Саме завдяки цій бібліотеці реалізовано функціонал завантаження, збереження та видалення файлів у S3-бакеті.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

1.2.1 Вибір фреймворку для розробки: чому Next.js

Для розробки сучасного вебзастосунку, особливо коли йдеться про інтеграцію frontend та backend частин, важливо обрати фреймворк, який забезпечує баланс між продуктивністю, простотою розгортання та масштабованістю. У рамках даного проєкту було розглянуто кілька варіантів (табл.1.2), зокрема React, Next.js, Nuxt, Remix та Angular. Найближчим конкурентом виступав React у зв'язці з окремим backend-сервером.

Таблиця 1.2

Порівняльний аналіз фреймворків

Критерій	React + Express	Next.js	Angular
Розділення frontend та backend	Так	Ні (все в одному проєкті)	Так
Server-side rendering (SSR)	Потрібна ручна реалізація	Вбудовано	Вбудовано
API-роути	Потрібно налаштовувати	Прямо в структурі app/api	Через окремий сервер
Простота деплою	Складніше, дві частини	Легше: один застосунок	Складна конфігурація
SEO та продуктивність	Низька без SSR	Висока	Висока
Крива навчання	Середня	Низька	Висока

Під час вибору технології для розробки вебзастосунку основним критерієм стала ефективність, швидкість розгортання, простота підтримки та можливість майбутнього масштабування. Саме тому Next.js [\[1\]](#) був обраний як основний фреймворк, адже він поєднує в собі клієнтську і серверну логіку, дозволяючи створити повноцінний монолітний застосунок у межах єдиного репозиторію. Уніфікована архітектура, що поєднує фронтенд і бекенд у межах одного проєкту, значно спрощує не лише розгортання, а й налагодження, обслуговування та тестування функціоналу. Оскільки Next.js має вбудовану підтримку API-роутів, зникає потреба

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

піднімати окремий сервер на базі Express або NestJS — усе необхідне для побудови повноцінної логіки взаємодії з клієнтом уже реалізовано у фреймворку.

Також Next.js «із коробки» підтримує такі технології як SSR (серверний рендеринг), ISR (інкрементальна генерація) та SSG (статична генерація), що дозволяє гнучко підходити до реалізації продуктивного і SEO-оптимізованого сайту. У межах нашого проєкту наразі використовується клієнтська отрисовка, проте за потреби є можливість швидко інтегрувати SSR без радикальних змін у коді. Крім того, Next.js надає зручну підтримку Tailwind CSS, TypeScript та ESLint, що дозволило уникнути витрат часу на конфігурування цих інструментів і зосередитися на розробці бізнес-логіки. Структура проєкту в Next.js є інтуїтивно зрозумілою та масштабованою, що робить його ідеальним варіантом не лише для навчальних або внутрішніх проєктів, але й для комерційного продакшену.

Зважаючи на це, було прийнято рішення не розділяти фронтенд і бекенд, як це часто робиться у великих продуктах. Такий інтегрований підхід став доцільним саме для цього проєкту: уся логіка знаходиться в одному репозиторії, і це суттєво полегшило як деплой, так і відлагодження. У результаті — одна збірка, один контейнер і єдиний сервіс на EC2 або Vercel. Це дозволило швидше розгортати застосунок, уникнути зайвого оверхеду, а також полегшити роботу розробникам, які не зобов'язані розбиратися у двох окремих кодових базах або сервісах.

У перспективі, коли з'явиться потреба в масштабуванні, API-частину можна буде винести в окремий мікросервіс, зберігаючи водночас зручну архітектуру. Таким чином, обраний інтегрований підхід на базі Next.js став найбільш ефективним рішенням з погляду продуктивності, швидкості розробки, простоти обслуговування та готовності до продакшену. Усі ці переваги дозволили реалізувати безпечну, зручну та повнофункціональну систему для управління файлами в хмарному середовищі.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

1.3 Аналіз принципів організації сховища файлів

Оскільки зберігання великих бінарних об'єктів (зображень, PDF, архівів тощо) у базі даних є неефективним, для зберігання самих файлів було обрано сервіс AWS S3 (Simple Storage Service) (рис. 1.2).

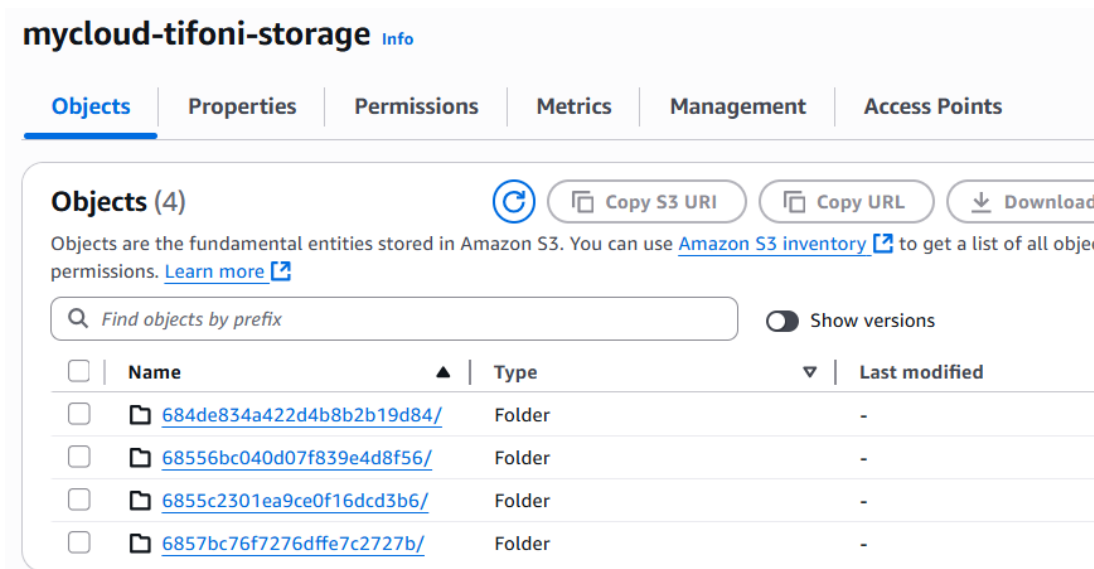


Рисунок 1.2 – Вигляд AWS S3 бакету.

Папки користувачів, де зберігаються їх файли

Фізичне зберігання файлів буде реалізовано через Amazon S3: кожен файл після завантаження потрапляє у відповідний бакет у хмарному сховищі. У самій базі даних зберігається лише метадані — назва файлу, розмір, MIME-тип, унікальний ключ (s3Key), а також посилання на користувача (userId), папку (folderId) та підрозділ (departmentId). Це дозволяє ефективно керувати файлами, не навантажуючи базу великими об'єктами. Доступ до файлів організовано через механізм підписаних URL — бекенд формує тимчасове посилання з обмеженим терміном дії, за яким користувач може безпечно завантажити файл напряму з S3, без потреби пересилання через сервер.

Основні принципи організації:

Кожен файл має прив'язку до папки (через folderId) та конкретного користувача (userId). Для забезпечення спільного доступу всередині департаменту використовується поле departmentId, що дозволяє користувачам бачити файли колег,

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

які належать до того самого підрозділу. Структура папок також зберігається у базі даних, ієрархія реалізується через поле `parentId`, що дозволяє створювати вкладені директорії — подібно до традиційних файлових систем (рис. 1.3).

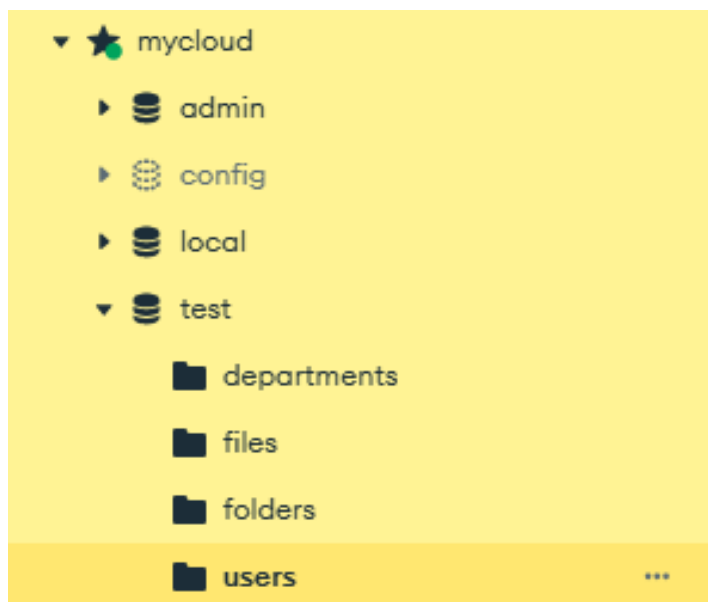


Рисунок 1.3 – Структура бази даних (Вигляд у MongoDB Compass)

Такий підхід має низку переваг. Масштабованість забезпечується завдяки використанню S3 — сервісу, здатного зберігати великі обсяги даних без зниження продуктивності. Безпека гарантується тим, що файли зберігаються приватно і доступні лише через тимчасові підписані посилання. А гнучкість системи досягається тим, що логіка керування доступом реалізована у MongoDB, що дозволяє ефективно управляти структурою, ролями та взаємозв'язками, не обмежуючи масштабування обробкою самих файлів.

Під час проєктування внутрішнього хмарного файлового сховища було прийнято рішення використовувати документо-орієнтовану базу даних MongoDB.

На відміну від реляційних СУБД (таких як PostgreSQL або MySQL), MongoDB дозволяє зберігати об'єкти зі складною структурою (JSON-документи) без попереднього визначення чіткої схеми, що пришвидшує розробку MVP, зменшує кількість JOIN-операцій і добре підходить для застосунків з часто змінюваною структурою даних (табл. 1.3).

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

Порівняння баз даних

Характеристика	MongoDB	PostgreSQL	MySQL
Тип БД	NoSQL (доку- ментоорієнт.)	Реляційна (SQL)	Реляційна (SQL)
Гнучкість структури	Висока	Середня (через JSONB)	Низька
Підтримка вкладених об'єктів	Так	Частково (через JSONB)	Частково (через JSON)
Продуктивність для чи- тання	Висока	Висока	Висока
Масштабованість (гори- зонтальна)	Проста	Складніша	Складніша
Ідеально підходить для	Вкладених структур, MVP	Структурованих даних, BI	Прості веб-за- стосунки
Підтримка транзакцій	Обмежена (але є)	Повна	Повна

Основними причинами цього вибору стали:

Гнучка структура даних, яка дозволяє легко змінювати схему під час розробки без необхідності складної міграції;

Відсутність складних JOIN-операцій: замість цього використовуються вкладені документи або посилання через ID;

Природна відповідність між документами MongoDB та об'єктами JavaScript (JSON);

Добра масштабованість у випадку великої кількості файлів та користувачів (табл. 1.4).

Таблиця 1.4

Ключові сутності

Сутність	Опис
User	Містить інформацію про користувача: email, пароль, роль, підрозділ
File	Представляє окремий файл: назва, шлях, власник, до якого folder належить
Folder	Каталог користувача, може бути вкладеним. Має ім'я, власника, parentId
Department	Визначає підрозділ (відділ), до якого належать користувачі та спільні файли

Таблиця 1.5

Зв'язки між цими сутностями

Від кого	До кого	Тип зв'язку
User	Department	N:1 (один підрозділ, багато користувачів)
File	User	N:1 (власник файлу)
File	Folder	N:1 (файл належить папці або root)
Folder	User	N:1 (власник)
Folder	Folder	1:N (вкладеність)

Цей підхід дозволяє легко реалізувати функціонал особистих і спільних файлів, підтримку вкладених папок та контроль доступу на рівні підрозділу.

1.4 Аналіз інструментів для автоматизації обміну та збереження даних

Для забезпечення зручності розробки, автоматизації процесів розгортання та підтримки сталої інфраструктури, у межах проєкту було використано низку сучасних інструментів.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

Одним із ключових рішень стало використання GitHub Actions для реалізації CI/CD-процесу [16]. Кожна зміна в кодовій базі, що потрапляє до відповідної гілки (develop, staging або main), автоматично запускає сценарій GitHub Actions, який:

- виконує перевірку коду (lint, build);
- проганяє тести (у разі наявності);
- деплоїть проєкт на відповідне середовище (наприклад, EC2 або Amplify).

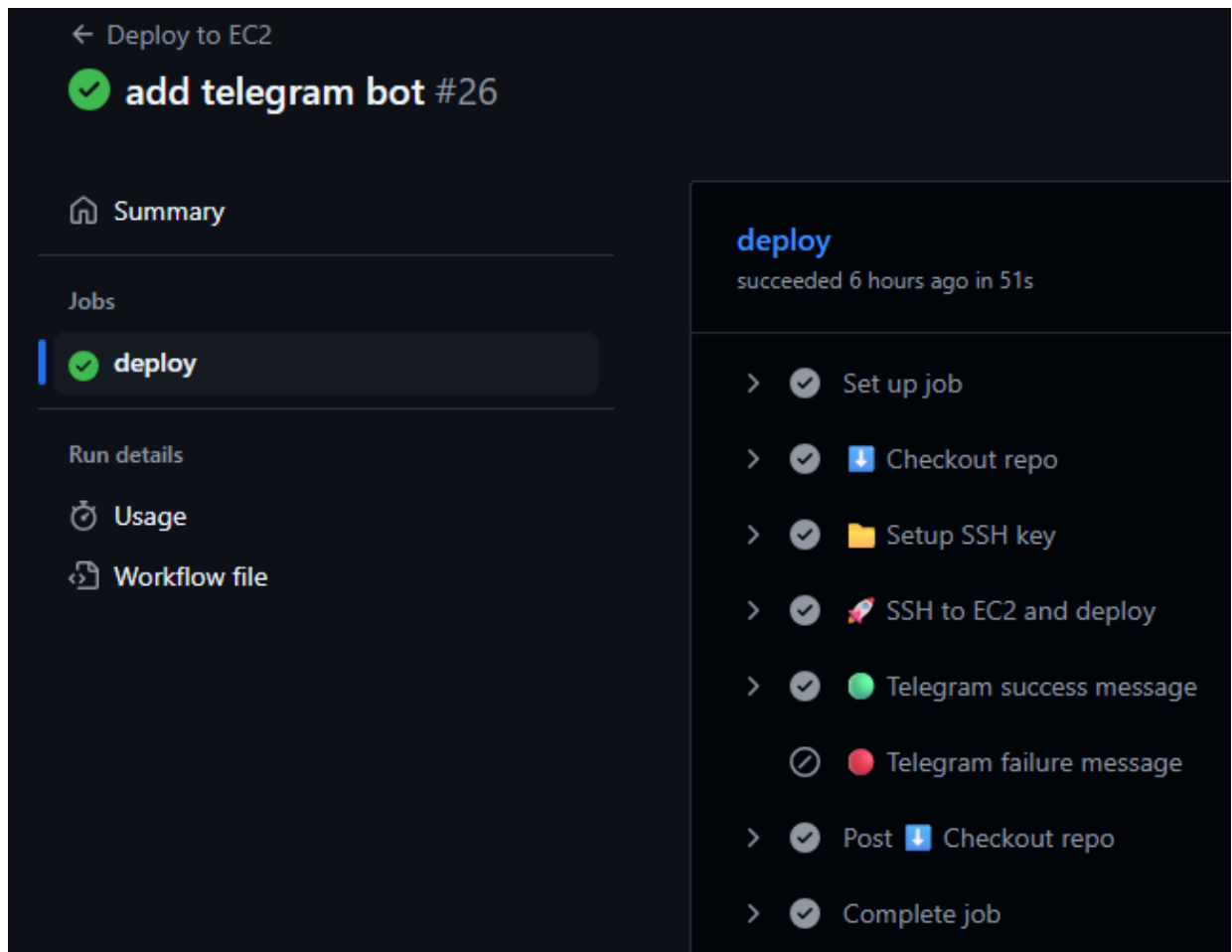


Рисунок 1.4 – CI CD Jobs, описані в пайплайні, які виконуються при пуші у гілку

Таким чином, розробникам не потрібно вручну запускати сценарії чи логінітись на сервер – усе відбувається автоматично після коміту (Додаток Б).

Для безпечної роботи з конфіденційними даними застосовуються .env файли, у яких зберігаються змінні середовища:

- MONGODB_URI – для підключення до бази даних MongoDB;
- JWT_SECRET – ключ для підпису токенів автентифікації;

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

- `AWS_ACCESS_KEY_ID` та `AWS_SECRET_ACCESS_KEY` – для взаємодії з AWS S3;
- інші ключі, потрібні для налаштування API та клієнтської частини.

Ці файли не потрапляють до репозиторію, що гарантує безпеку. У GitHub Actions значення змінних конфігуровано через GitHub Secrets

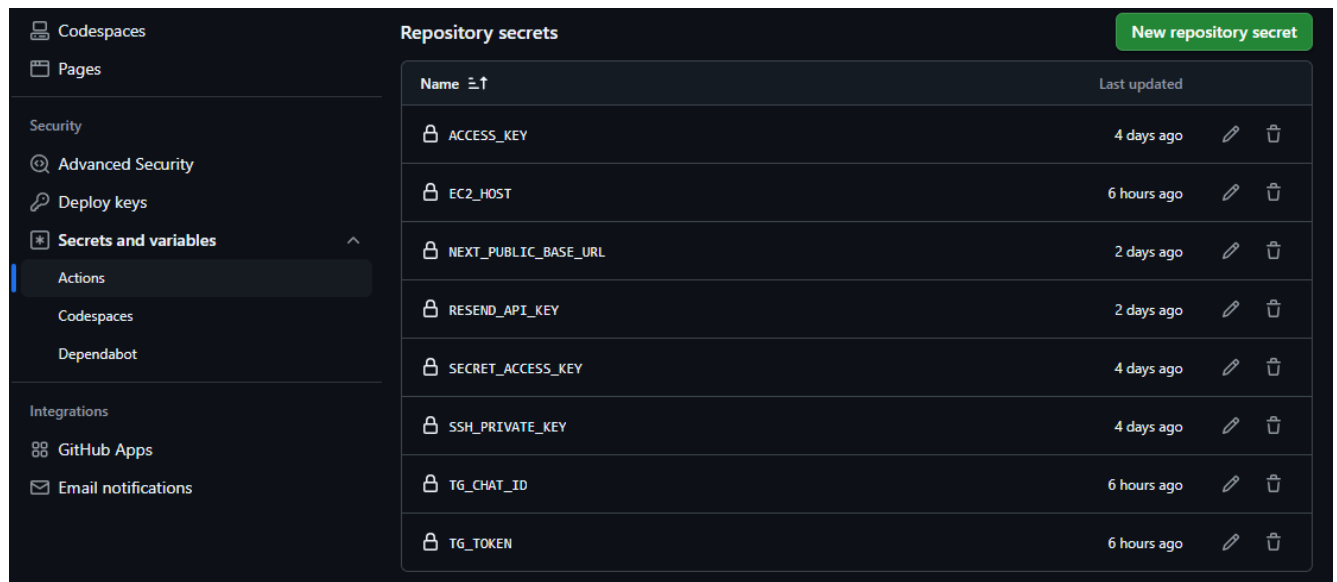


Рисунок 1.5 – GitHub Secrets, які потрібні для використання пайплайном

Завдяки такому підходу команда отримує:

- швидке оновлення застосунку;
- менше ручної роботи;
- менше шансів на помилки;
- зручність масштабування та переходу між середовищами (dev, staging, prod).

1.4.1. Використання підписаних URL для захищеного доступу до файлів у S3

У межах реалізації хмарного файлового сховища важливо забезпечити безпечний доступ до файлів користувачів без необхідності надавати публічний доступ до AWS S3-бакету. Для цього в проєкті реалізовано механізм підписаних URL (Signed URLs), які дозволяють надавати тимчасовий контрольований доступ до об'єктів, що зберігаються у сховищі Amazon S3.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

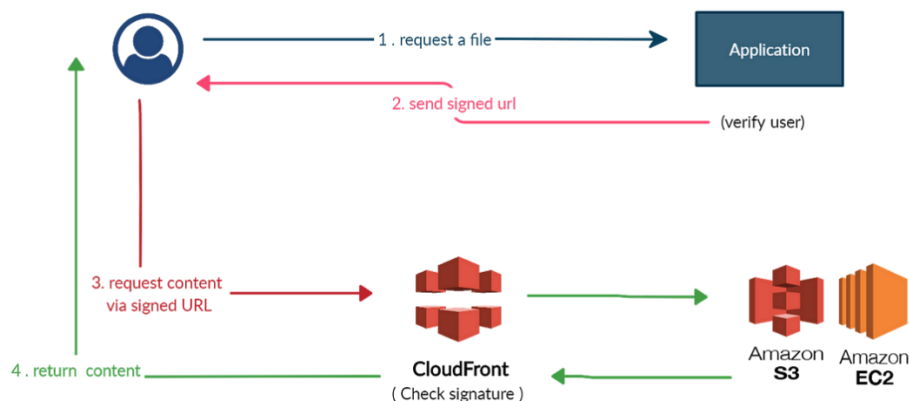


Рисунок 1.6 – Схема принципу роботи підписних URLs

Підписане посилання – це спеціальний URL, який містить цифровий підпис та параметри безпеки, що підтверджують авторизований доступ до конкретного файлу. Такий URL створюється на сервері за допомогою AWS SDK і включає інформацію про:

- Назву бакету та ключ файлу (шлях до об'єкта),
- Термін дії посилання (наприклад, 60 секунд),
- Підпис, згенерований на основі секретного ключа AWS.

Таким чином, навіть якщо S3-бакет не має публічного доступу, користувач може отримати доступ до власного файлу лише за умови, що:

1. Він пройшов автентифікацію у вебзастосунку;
2. Отримав від бекенду унікальне підписане посилання на свій файл;
3. Використав його до завершення терміну дії.

Цей підхід має низку переваг:

- **Безпека:** жоден файл не може бути отриманий без авторизації;
- **Оптимізація:** файли передаються напряму з S3 до браузера користувача, минаючи серверну частину;
- **Контроль доступу:** можна точно регулювати, коли і хто отримає доступ до ресурсу.

Розвантаження сервера за допомогою signed URLs

При спробі завантаження файлу користувач отримує тимчасове підписане посилання, що веде безпосередньо до S3. Завдяки цьому:

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

- EC2-сервер не обробляє запит на завантаження – файл передається напряму з AWS S3 до браузера;
- Це дозволяє обробляти сотні одночасних завантажень без зниження продуктивності;
- Навіть у випадку пікового навантаження (наприклад, 100+ запитів), інстанс EC2 залишається стабільним.

Наприклад, під час тестування система витримувала навантаження в кілька одночасних завантажень, при цьому середнє використання CPU залишалося нижчим за 0.5%.

У проєкті генерація підписаного URL відбувається на основі AWS SDK команд типу `GetObjectCommand`, що в комбінації з `getSignedUrl()` забезпечує гнучкий, безпечний механізм видачі файлів. Таким чином, підписані URL є ключовим елементом безпечної та ефективної організації файлового обміну між сервером та користувачем.

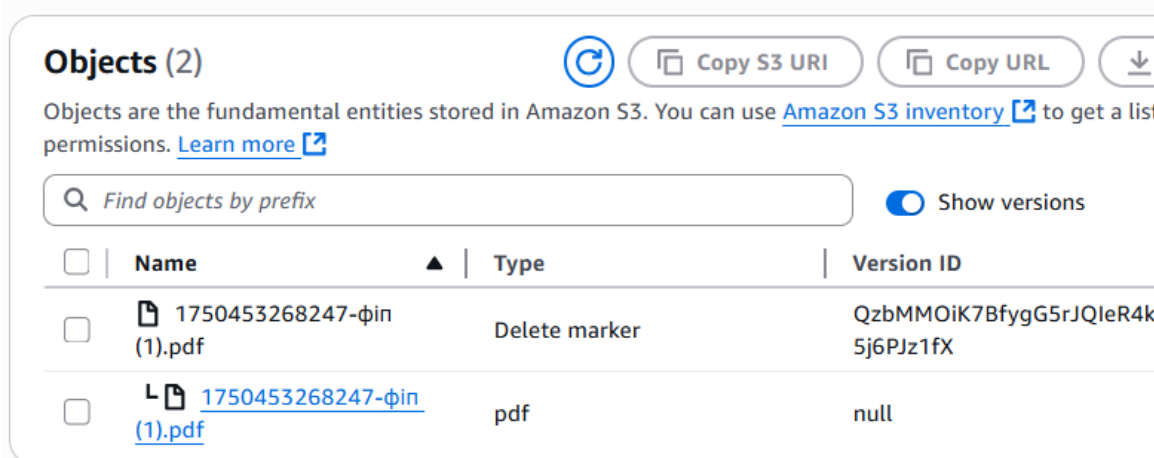


Рисунок 1.7 – Вигляд видаленого файлу під міткою

Наприклад, після видалення файлу через інтерфейс AWS S3, система не знищує об’єкт, а лише позначає його як видалений, зберігаючи сам файл у попередній версії.

Автоматичне очищення старих версій

Щоб уникнути надмірного накопичення застарілих версій і не перевищувати квоту зберігання, було налаштовано Lifecycle Rule з назвою DeleteOldVersionsAfter30Days, яка:

- Автоматично видаляє застарілі версії файлів через 30 днів після їх створення;
- Поширюється на всі об'єкти бакету;
- Не впливає на поточні версії файлів.

Таким чином, реалізоване рішення забезпечує:

- Можливість відновлення видалених файлів;
- Автоматичне управління версіями об'єктів;
- Мінімальні витрати на зберігання через політику очищення;
- Повну прозорість резервного копіювання без потреби ручних дій.

1.5 Аналіз засобів моніторингу та збору метрик

Для забезпечення стабільної роботи вебзастосунку та вчасного виявлення можливих проблем було впроваджено базову систему моніторингу на базі AWS CloudWatch. Цей сервіс дозволяє збирати та візуалізувати метрики продуктивності EC2-інстансу, зокрема завантаження CPU, використання оперативної пам'яті, обсяг доступного місця на диску та мережеву активність.

Переваги використання AWS CloudWatch полягають у тому, що він забезпечує централізоване логування та збір метрик — усі дані про стан EC2-інстансу збираються в одному зручному інтерфейсі. Завдяки підтримці графіків і дашбордів у реальному часі, розробник може оперативно оцінювати навантаження на сервер і виявляти тренди використання ресурсів.

Крім цього, CloudWatch дозволяє налаштовувати нотифікації за допомогою сервісу Amazon SNS — таким чином можна отримувати повідомлення про критичні події у Telegram, на електронну пошту або інші канали. Це значно спрощує моніторинг без постійного ручного контролю.

Сервіс також дає змогу здійснювати аналіз продуктивності — наприклад, можна відслідковувати, як змінюється навантаження після нових релізів, деплоїв

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

або у випадку критичних помилок у коді. Все це в комплексі підвищує стабільність системи, адже дозволяє швидко реагувати на проблеми, як-от вичерпання оперативної пам'яті, надмірне завантаження CPU або зависання процесів.

Крім стандартного моніторингу, CloudWatch дозволяє задавати автоматичні правила реагування (наприклад, запуск скриптів або масштабування інстансу при перевищенні певного порогу навантаження), а також інтегрується з іншими сервісами AWS, такими як Auto Scaling, Lambda чи S3, що робить систему легко масштабованою та придатною до розширення.

Окрім збору метрик, AWS CloudWatch також надає можливість централізованого логування через CloudWatch Logs. Це дає змогу зберігати журнали (логи) з EC2-інстансів, контейнерів, Lambda-функцій або будь-яких інших сервісів AWS у єдиному сховищі. У контексті розробленого вебзастосунку це дозволяє отримувати докладну інформацію про роботу бекенда, результати API-запитів, повідомлення про помилки, а також будь-які системні події.

Завдяки логам можна:

1. шукати помилки або винятки, які не завжди видно з UI, і швидко їх усувати;
2. відстежувати поведінку користувачів (наприклад, частоту запитів до конкретного API або повторні невдалі спроби логіну);
3. аудитити події безпеки, включаючи спроби доступу без авторизації або підозрілі дії;
4. зберігати історію подій для аналізу в майбутньому.

CloudWatch Logs підтримує фільтрацію за ключовими словами, створення метрик на основі логів (наприклад, кількість помилок 500), а також налаштування алермів на основі певних патернів у логах. Наприклад, якщо у логах за останні 5 хвилин з'явилося понад 10 помилок `MongoNetworkError`, CloudWatch може автоматично надіслати сповіщення адміністратору або активувати автоматичне відновлення.

Централізація логування в CloudWatch Logs також спрощує віддалену діагностику: адміністратор не мусить заходити на сервер через SSH, щоб перевірити журнали — вся потрібна інформація доступна через вебінтерфейс AWS. У поєднанні з

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

моніторингом метрик це створює повноцінну картину стану системи у реальному часі.

У результаті, використання AWS CloudWatch стає ключовим компонентом інфраструктурної надійності, забезпечуючи проактивний моніторинг, сповіщення та засоби аналізу, необхідні для підтримки високого рівня доступності вебза-стосунку.

Висновки до розділу 1

За результатами Розділу 1, було проаналізовано існуючі рішення для хмарного зберігання (Google Drive, Dropbox, Nextcloud тощо) та сформульовано ключові вимоги до майбутньої системи. Розроблено архітектуру програмного комплексу, що базується на AWS-сервісах і Next.js. Обґрунтовано вибір бази даних MongoDB, технологій фронтенду, хмарної інфраструктури, системи CI/CD та моніторингу. Визначено принципи взаємодії між клієнтом, сервером і хмарним сховищем. Зібрано перелік функціональних та нефункціональних вимог до системи.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

РОЗДІЛ 2 ПРОЄКТУВАННЯ ХМАРНОГО ФАЙЛОВОГО СХОВИЩА

2.1 Архітектура та структура майбутнього програмного комплексу

Проектоване хмарне файлове сховище має модульну архітектуру, що базується на сучасному стеку вебтехнологій та забезпечує гнучкість, масштабованість і безпеку. Система реалізується за моделлю клієнт–сервер і орієнтована на використання як у внутрішньоорганізаційному середовищі, так і в приватному хмарному середовищі (наприклад, AWS або локальний сервер).

Основні компоненти системи складаються з клієнтської частини, серверної частини, бази даних і об'єктного сховища файлів. Клієнтська частина (frontend) реалізована з використанням фреймворку Next.js. Вона відповідає за взаємодію з користувачем, зокрема за завантаження та перегляд файлів, навігацію по папках, управління доступом і автентифікацію.

Серверна частина (backend) розробляється на основі Next.js. Вона обробляє запити, взаємодіє з базою даних, забезпечує автентифікацію користувачів, перевіряє права доступу та генерує підписані посилання для завантаження файлів.

У якості бази даних буде використовуватись MongoDB. У ній буде зберігатися інформація про користувачів, підрозділи, файли, ролі та структуру папок. Завдяки документно-орієнтованій природі ця база дозволяє ефективно працювати зі складною вкладеною структурою даних.

Фізичне зберігання файлів повинно бути реалізовано через об'єктне сховище Amazon S3 або інше S3-сумісне рішення. Файли зберігаються в приватному бакеті, а доступ до них надається лише за допомогою підписаних URL, які мають обмежений термін дії.

Серверна частина складається з кількох основних модулів. Модуль автентифікації відповідає за логін користувачів, генерацію та перевірку JWT-токенів, а також хешування паролів. Модуль користувачів забезпечує керування профілями, ролями та приєднанням до підрозділів. Модуль файлів реалізує завантаження, видалення, перегляд і формування списків файлів, а також розподіл доступу до них.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

Модуль папок відповідає за створення ієрархічної структури з підтримкою вкладених директорій. Окремо виділений модуль доступу (ACL), який забезпечує перевірку прав, дозволів і доступності файлів або папок для конкретного користувача.

У системі підтримуються ролі користувача та адміністратора підрозділу. Користувач має базовий доступ до власних файлів і папок. Адміністратор підрозділу має додаткові можливості — зокрема, створення підрозділів і керування користувачами в межах цих підрозділів.

2.1.1. Інструментальні засоби реалізації програмного продукту та вимоги до апаратного забезпечення

Розробка системи безпечного хмарного файлового сховища ґрунтується на використанні сучасного технологічного стеку, що забезпечує масштабованість, продуктивність, безпеку та гнучкість при розробці.

Клієнтська частина реалізована за допомогою фреймворку Next.js, який базується на React і підтримує серверний рендеринг, App Router, а також оптимізацію продуктивності. Для розробки адаптивного, уніфікованого й візуально привабливого інтерфейсу застосовуються Tailwind CSS. Завдяки використанню TypeScript розробники отримують гарантії типобезпеки, що допомагає уникнути численних помилок ще на етапі написання коду. Анімації в інтерфейсі реалізовані за допомогою бібліотеки Framer Motion, що дозволяє досягти плавних і приємних для користувача візуальних ефектів.

Серверна частина базується на MongoDB — документоорієнтованій базі даних, яка ідеально підходить для зберігання структурованої й вкладеної інформації, зокрема про користувачів, підрозділи та файли. Для взаємодії з цією базою застосовується бібліотека Mongoose, яка забезпечує зручну роботу зі схемами й валідацією даних. Механізм автентифікації побудований на основі JWT (JSON Web Token), що дозволяє ефективно управляти сесіями користувачів, а для безпечного зберігання паролів використовується бібліотека bcrypt, яка виконує їх хешування.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		29

Файли зберігаються у хмарному об'єктному сховищі Amazon S3. Доступ до них реалізовано через механізм підписаних посилань (Signed URLs), які мають обмежений термін дії. Це дозволяє безпечно передавати доступ до файлів без ризику відкриття повного доступу до бакету.

Інфраструктура для розгортання проєкту побудована на базі Amazon EC2, що забезпечує як розміщення фронтенду, так і бекенду. Автоматизація розгортання реалізована через GitHub Actions: залежно від гілки, відбувається автоматичний деплой із виконанням всіх необхідних команд для оновлення застосунку.

Вимоги до апаратного забезпечення:

Оскільки система може бути як хостована у хмарі, так і розгорнута локально, апаратні вимоги залежать від кількості користувачів та обсягів файлів.

Було здійснено аналіз теоретичних засад побудови систем хмарного зберігання файлів з акцентом на безпеку, гнучкість та масштабованість. Обґрунтовано актуальність теми кваліфікаційної роботи, яка зумовлена зростанням потреб організацій у безпечному, контрольованому та кастомізованому файловому сховищі.

Проведене порівняння популярних рішень – Google Drive, Dropbox, Nextcloud та Seafile – дало змогу виокремити їхні переваги й недоліки. Зокрема, комерційні сервіси мають простоту у використанні, але не забезпечують повного контролю над даними. Водночас Nextcloud і Seafile дозволяють локальне розгортання та високу гнучкість, однак мають складну архітектуру, потребують доопрацювання для реалізації специфічної логіки доступу та управління ролями.

У результаті обґрунтовано доцільність створення власного рішення, орієнтованого на потреби організації: з підтримкою ролей, підрозділів, контролем прав доступу, простим інтерфейсом і можливістю інтеграції з внутрішніми системами. Описана архітектура програмного комплексу дозволяє забезпечити модульність, ізольованість компонентів, масштабованість, а також відповідність вимогам інформаційної безпеки.

Визначені інструментальні засоби реалізації забезпечують баланс між продуктивністю, зручністю розробки та безпечним управлінням даними. Також

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

окреслено апаратні вимоги, що дозволяють використовувати систему як у хмарному середовищі, так і на локальних серверах

2.1.2. Вибір хмарної інфраструктури та її конфігурація в AWS

Для реалізації надійного та масштабованого хмарного файлового сховища було обрано хмарну платформу Amazon Web Services (AWS). Цей вибір обумовлений високим рівнем безпеки, доступністю широкого спектру сервісів, зручністю масштабування та підтримкою автоматизованого розгортання інфраструктури.

Використання сервісів AWS

Для забезпечення роботи файлового сховища були використані наступні сервіси AWS:

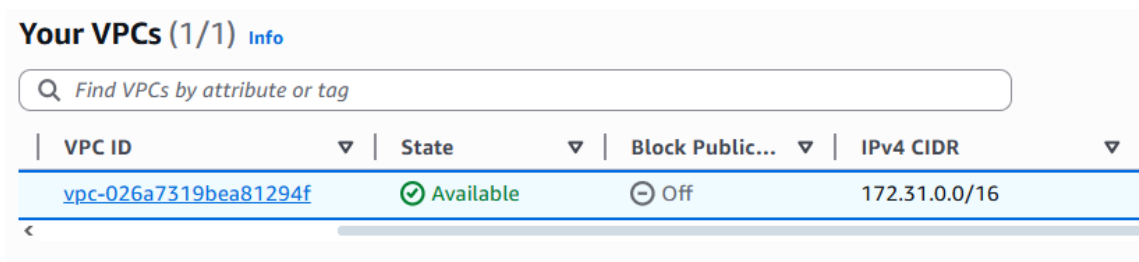
Amazon EC2 (Elastic Compute Cloud) використовується як хостингова платформа для повноцінного монолітного вебзастосунку, реалізованого на Next.js. У межах цього проєкту EC2-екземпляр виконує роль одночасно клієнтської та серверної частини – через API-роути Next.js реалізовано бекенд-логіку без потреби в окремому сервері. Таким чином, EC2 виступає основним середовищем виконання усього застосунку, забезпечуючи його доступність в мережі та можливість масштабування при необхідності.

Amazon S3 (Simple Storage Service) використовується для зберігання файлів, завантажених користувачами. Кожен файл потрапляє в певний бакет (bucket), має унікальний ключ і зберігається у хмарному сховищі з високою доступністю. Налаштовані політики доступу до S3 забезпечують захист даних, завдяки ролям (IAM) та обмеженням на рівні бакетів.

VPC (Virtual Private Cloud) забезпечує ізольоване віртуальне середовище в хмарі AWS, у якому розміщується EC2-екземпляр із вебзастосунком. У межах цього проєкту використовується стандартна (дефолтна) VPC, створена автоматично під час запуску екземпляра. Цього достатньо для забезпечення базової мережевої інфраструктури, зокрема прив'язки публічної IP-адреси до інстансу та надання доступу до нього через інтернет. VPC дозволяє створити ізольовану мережу для сервісів проєкту. У проєкті використано стандартну конфігурацію з

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

публічними та приватними підмережами, що дозволяє контролювати мережевий трафік і безпеку.



VPC ID	State	Block Public...	IPv4 CIDR
vpc-026a7319bea81294f	Available	Off	172.31.0.0/16

Рисунок 2.1 – Створений VPC, де розміщується EC2 з додатком

Також налаштовано ролі та політики доступу для обмеження прав користувачів та сервісів до ресурсів AWS, тобто IAM (Identity and Access Management)

2.2 Розробка структури бази даних

Для зберігання даних у системі було обрано MongoDB — документоорієнтовану базу даних, яка ідеально підходить для роботи з вкладеними структурами. Завдяки своїй гнучкості, вона дозволяє ефективно зберігати як дані користувачів, так і структуру папок, підрозділи та метадані про файли без складних JOIN-запитів, як у реляційних базах.

У межах даного проєкту було створено кілька ключових колекцій, кожна з яких відповідає за окрему частину логіки системи:

```
_id: ObjectId('6855c2301ea9ce0f16dcd3b6')
email: "ki211_uiv@student.ztu.edu.ua"
password: "$2b$10$i8oeI48ShFbId0n0Rwv9M04V4gsOQP0y1R5aYsyTlH00H..m8eXym"
nickname: "NoName"
role: "admin"
department: ObjectId('6855c66e1ea9ce0f16dcd40e')
verifiedSessions: Array (empty)
__v: 0
```

users - зберігає інформацію про користувачів, зокрема їхні облікові дані, ролі, приналежність до департаменту, а також технічну інформацію (наприклад, дату створення облікового запису, статус підтвердження авторизації тощо). Структура

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

дозволяє легко перевірити, до якого підрозділу належить користувач і які права має.

```
_id: ObjectId('68556c3840d07f839e4d8f64')
name : "1"
parentId : null
userId : ObjectId('68556bc040d07f839e4d8f56')
__v : 0
```

folders - забезпечує ієрархічну структуру з підтримкою вкладених директорій. Кожна папка має поле `parentId`, яке дозволяє реалізувати дерево вкладеності. Також зберігається інформація про власника папки (`userId`) та її належність до департаменту (`departmentId`), якщо це спільна директорія.

```
_id: ObjectId('6851211c243fb9e958faa637')
userId : "684de834a422d4b8b2b19d84"
filename : "Без_імені-removebg-preview.png"
s3Key : "684de834a422d4b8b2b19d84/68511d32243fb9e958faa62b/1750147356556-Без_ім..."
url : "https://mycloud-tifoni-storage.s3.eu-north-1.amazonaws.com/684de834a42..."
folderId : ObjectId('68511d32243fb9e958faa62b')
uploadedAt : 2025-06-17T08:02:36.915+00:00
__v : 0
```

files - ця колекція містить метайнформацію про кожен файл: його назву, MIME-тип, розмір, дату завантаження, користувача-власника (`userId`), шлях до файлу у S3 (ключ `s3Key`), папку, у якій файл зберігається (`folderId`), та, за потреби, департамент (`departmentId`) — якщо файл призначено для колективного користування. Завдяки цьому користувачі можуть бачити свої файли, а також файли своїх колег з одного департаменту.

```
_id: ObjectId('6855c4501ea9ce0f16dcd3d5')
name : "First"
createdBy : ObjectId('684de834a422d4b8b2b19d84')
createdAt : 2025-06-20T20:28:00.760+00:00
__v : 0
```

departments - зберігає інформацію про існуючі департаменти (назву, опис) та список учасників. Це дозволяє формувати логіку обмеження доступу до файлів і папок у межах певної групи користувачів.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

Фізично файли не зберігаються в базі даних, а розміщуються у приватному S3-бакеті, що дозволяє уникнути зберігання великих бінарних об'єктів у MongoDB. Натомість у базу записується лише супровідна інформація, необхідна для контролю доступу та навігації.

2.3 Моделювання системи доступу користувачів

Для забезпечення безпеки та гнучкого контролю прав у вебзастосунку буде реалізовано **JWT-аутентифікацію**. Користувачі при реєстрації створюють обліковий запис, пароль хешується за допомогою бібліотеки bcrypt, а при вході в систему – перевіряється на відповідність. Після успішної автентифікації створюється JWT-токен за допомогою jsonwebtoken, який надсилається клієнту та зберігається в локальному сховищі.

Кожен API-роут Next.js перевіряє наявність та дійсність токена в заголовках Authorization. Для цього використовуються утиліти з lib/utlils/verifyToken.ts. Якщо токен недійсний або відсутній – повертається помилка 401.

Таким чином, кожен запит до захищених маршрутів забезпечений перевіркою авторизації. Додатково в об'єкті користувача ведеться облік verifiedSessions, де зберігаються активні сесії з мітками часу, що дозволяє реалізувати логіку підтвердження нових входів або захисту від повторного використання токенів.

Додатково буде механізм підтвердження сесії: після логіну користувачеві на пошту надсилається спеціальне посилання для підтвердження нової сесії (рис.2.2).

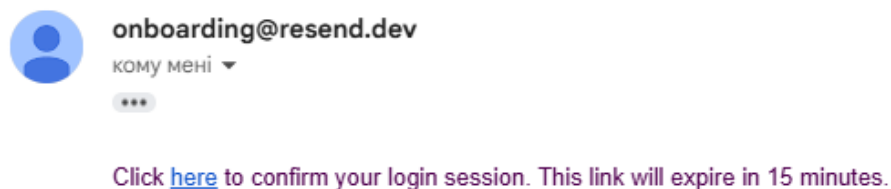


Рисунок 2.2 – Лист на пошту для підтвердження сесії

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

Лише після переходу за цим посиланням сесія вважається перевіреною, і користувач отримує доступ до системи. Це дозволить підвищити безпеку облікового запису та запобігти несанкціонованому доступу.

Уся серверна логіка реалізована у вигляді **API-роутів** Next.js у директорії `app/api`. Кожен підкаталог відповідає окремому ресурсу (користувачі, файли, підрозділи тощо), і всередині містить обробники HTTP-запитів (GET, POST, DELETE).

Основні групи ендпоінтів (Додаток А):

- `api/auth/` – логін і реєстрація користувача. Після успішної автентифікації видається JWT-токен.
- `api/user/` – отримання профілю користувача, включаючи його роль та підрозділ.
- `api/files/` – завантаження файлів, видалення, отримання списку файлів. Усі операції захищені перевіркою токена.
- `api/folders/` – створення нових папок і отримання дерева для відображення у файловому менеджері.
- `api/departments/` – створення, оновлення та отримання списку підрозділів. Доступ обмежено для адміністратора.
- `api/sessions/` – перевірка активної сесії користувача через список `verifiedSessions`.

Логіка доступу до файлів:

Для контролю прав доступу використовується наступна перевірка:

- Користувач бачить тільки свої власні файли (`userId === currentUser.id`).
- Якщо користувач належить до підрозділу, він також бачить файли інших учасників цього підрозділу.
- Адміністратор (роль `admin`) має змогу створювати підрозділи, до яких в подальшому можуть приєднуватись користувачі.

Усі запити до `files` і `folders` проходять через захищений бекенд, де перед операцією перевіряється (Додаток А):

- авторизація користувача через токен;

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

- відповідність folder.userId або file.userId поточному користувачу;
- або user.department === fileOwner.department.

Таким чином реалізовано гнучку і безпечну модель доступу, яка дозволяє співпрацювати в межах підрозділів, зберігаючи ізоляцію файлів між ними.

Графічний інтерфейс користувача (UI) буде реалізовано на базі React із використанням TailwindCSS. Увесь інтерфейс, зв'язаний з файлами та папками побудовано як односторінковий застосунок (SPA), який дозволяє повністю керувати ними через зручну вебпанель.

Основні можливості інтерфейсу:

Створення папок: користувач може створювати власні папки у файловому менеджері. Також підтримується вкладена структура: можна створювати підпапки будь-якої глибини (папки в папках, підпапки в підпапках тощо).

Видалення папок: дозволено тільки для власника цієї папки. Права перевіряються на сервері. Інші користувачі, навіть з того ж підрозділу, не можуть видалити чужі папки.

Завантаження файлів: по натисканню на кнопку можна викликати форму для вибору файлу. Кожен файл прив'язаний до власника й до папки (або зберігається у корені, якщо папку не обрано).

Скачування файлів: користувач може завантажити свої файли, а також файли з папок, які належать учасникам його підрозділу. Завантаження реалізується через механізм підписаних URL, тому не створює навантаження на сервер.

Видалення файлів: дозволено лише власнику файлу. Якщо інший користувач із підрозділу переглядає ці файли, функція видалення для нього недоступна.

Створення нових підрозділів: для користувачів, які мають роль admin в профілі є можливість створити додатковий підрозділ, до якого в подальшому зможуть приєднуватись користувачі (рис.2.3)

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

Рисунок 2.3 – Можливість створення підрозділу (відділу)

2.4 Інфраструктура доступу, моніторингу та резервного копіювання

Щоб уникнути перевантаження сервера та не відкривати публічний доступ до Amazon S3-бакету, треба реалізувати механізм підписаних URL (Signed URLs). Для цього використано бібліотеку `@aws-sdk/s3-request-presigner`, яка дозволяє згенерувати тимчасове посилання для прямого завантаження файлу з бакету S3.

Механізм працює наступним чином:

1. Користувач клікає на кнопку “Download”.
2. У браузері формується запит до API-роуту `/api/files/download`, в якому передається:
 - ключ (s3Key) файлу в бакеті;
 - JWT-токен користувача.
3. На сервері:
 - перевіряється токен користувача;
 - створюється інстанс команди `GetObjectCommand`;
 - генерується підписаний URL з обмеженням часом дії (наприклад, 60 секунд) через функцію `getSignedUrl()`.
4. Користувачу повертається редірект на цей URL, і браузер починає завантаження файлу без участі серверної частини.

Методи та засоби забезпечення стабільної роботи системи

З метою попередження збоїв у роботі вебзастосунку буде реалізовано механізми моніторингу навантаження на серверну інфраструктуру. Для цього треба

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

використати вбудовані інструменти Amazon Web Services (AWS), зокрема CloudWatch Agent, що дозволяє в режимі реального часу отримувати статистику щодо споживання ресурсів віртуального сервера (EC2).

AWS CloudWatch Agent було встановлено та налаштовано з параметрами збору наступних метрик:

- Завантаження процесора (CPU): `cpu_usage_user`, `cpu_usage_system`, `cpu_usage_idle`;
- Споживання оперативної пам'яті (RAM): `mem_used_percent`;
- Використання дискового простору: `disk_used_percent`.

Конфігурація агента зберігається у файлі `config.json` (Додаток В) у форматі JSON та передається системі за допомогою утиліти `amazon-cloudwatch-agent-ctl`. Важливою частиною налаштування є прив'язка до конкретного екземпляра EC2 через параметр `InstanceId`.

Отримані метрики виводяться в AWS CloudWatch → Metrics, що дає змогу аналізувати навантаження в часовому розрізі, виявляти піки активності та потенційні «вузькі місця».

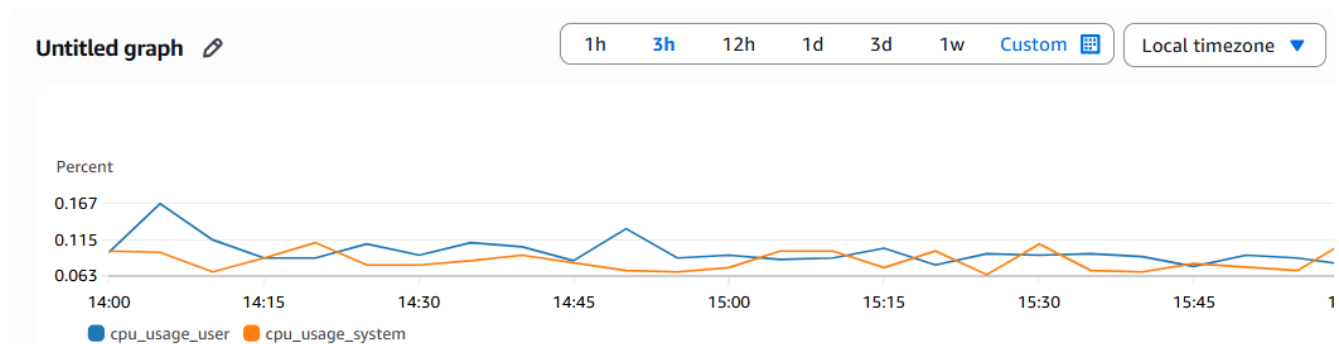


Рисунок 2.4 – Графік навантаження cpu в AWS CloudWatch → Metrics

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				38
Змн.	Арк.	№ докум.	Підпис	Дата		

Browse	Multi source query	Graphed metrics (2)	Options	Source	=
Metrics (3) Info					
Stockholm ▼ InstanceId, cpu Search for any metric, dimension, resource id or account id					
Instance name 3/3 InstanceId cpu Metric name Alarms					
☐	tifoni-server	i-00950b8e6bd53...	cpu...	cpu_usage_idle ⓘ	No alarms
☒	tifoni-server	i-00950b8e6bd53...	cpu...	cpu_usage_user ⓘ	1 alarm(s)
☒	tifoni-server	i-00950b8e6bd53...	cpu...	cpu_usage_system ⓘ	No alarms

Рисунок 2.5 – Вибір метрик для відображення на графіку

Автоматизація реагування на перевантаження

Окремо було реалізовано CloudWatch Alarm, яка спрацює при перевищенні порогу використання CPU понад 80% протягом заданого інтервалу. У разі спрацювання, система надсилає повідомлення на email, що дозволяє оперативно реагувати на інциденти

Alarms (1)					
Hide Auto Scaling alarms Clear selection Create composite alarm Actions					
Search Alarm state: Any Alarm type: Any Actions status: Any					
☐	Name	State	Last state update (Local)	Conditions	Actions
☐	CPUAlarmTopic	OK	2025-06-22 11:14:46	cpu_usage_user > 80 for 1 datapoints within 5 minutes	Actions enabled

Рисунок 2.6 – Alarm, який буде надсилати сповіщення про перевищення навантаження на сервері більше за 80%

Також налаштовано GitHub Actions пайплайн, який забезпечує автоматичний деплой застосунку на сервер при кожному пуші до гілки master. Це дозволяє гарантувати актуальність коду та мінімізувати ризик людських помилок при розгортанні.

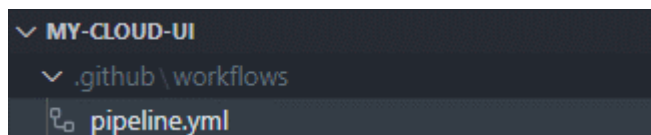


Рисунок 2.7 – yml файл пайплайну в папці проєкту

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

Перед кожним запуском додатку пайплайн виконує перевірку, що перед стартом старий процес зупинено, навіть якщо він завис чи був некоректно завершений. Це дозволяє перезапускати застосунок без зависання процесів, навіть якщо попередня сесія не була завершена коректно.

Завдяки впровадженню AWS CloudWatch Agent та системи автоматичних деплоїв через GitHub Actions, було досягнуто високого рівня стабільності та контролюваності роботи серверної частини застосунку. Це знижує ймовірність збоїв і забезпечує безперервний моніторинг системних ресурсів.

Резервне копіювання та захист даних

У рамках забезпечення надійності та відновлення даних у вебсховищі було прийнято рішення не видаляти об'єкти остаточно, а зберігати їхні попередні версії. Для цього у головному бакеті mycloud-tifoni-storage було увімкнено версіонування об'єктів. Такий підхід дозволяє в будь-який момент відновити файл, навіть якщо його було випадково або навмисно видалено. Видалення в такому разі лише додає спеціальну мітку (delete marker), не стираючи сам файл зі сховища.

2.5 Автоматизація розгортання та моніторинг інфраструктури

Для автоматичного розгортання вебзастосунку після кожного оновлення репозиторію має бути налаштовано CI/CD-процес за допомогою GitHub Actions. У репозиторії повинен бути файл .github/workflows/pipeline.yml, який запускається при пуші в гілку.

У пайплайні має бути налаштована автоматична процедура розгортання вебзастосунку на EC2-сервері. Першим кроком має відбуватись клонування репозиторію за допомогою actions/checkout@v3, щоб отримати актуальний код із віддаленої гілки.

Далі слід реалізувати налаштування SSH-з'єднання: необхідно зчитувати приватний ключ із GitHub Secrets (SSH_PRIVATE_KEY) та зберігати його у файл ~/.ssh/id_rsa, а також виконувати команду ssh-keyscan для додавання хоста до known_hosts, що забезпечить безпечне підключення.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				40
Змн.	Арк.	№ докум.	Підпис	Дата		

Після встановлення з'єднання, пайплайн повинен заходити на EC2-сервер по SSH та виконувати такі дії:

- перейти у директорію з проєктом;
- зробити `git pull` для оновлення коду;
- встановити всі залежності через `npm install`;
- побудувати застосунок (`npm run build`);
- зупинити попередній процес (через `pm2 delete` або іншу команду);
- запустити оновлену версію застосунку з `pm2`, вказавши унікальну назву процесу (наприклад, `my-cloud-ui`);
- зберегти конфігурацію `pm2`, щоб забезпечити автозапуск сервісу після перезавантаження сервера.

Усе це має відбуватись автоматично після пушу у відповідну гілку, забезпечуючи безперервну інтеграцію та доставку нової версії застосунку.

Telegram-сповіщення

Після завершення деплою відправляється повідомлення в Telegram через HTTP-запит до бота [\[12\]](#). Якщо деплой успішний – надсилається повідомлення “Deploy to EC2 succeeded!”, у випадку помилки – “Deploy to EC2 failed!”. Для цього використовуються змінні середовища `TG_TOKEN` та `TG_CHAT_ID`.

Це дозволяє забезпечити безперервну інтеграцію та швидке оновлення сервісу після змін у репозиторії без ручного втручання, що суттєво покращує стабільність та зручність розгортання.

Для контролю навантаження на сервер вебзастосунку було налаштовано моніторинг через сервіс **Amazon CloudWatch** [\[6\]](#). Це дозволяє в реальному часі відстежувати стан віртуальної машини, на якій запущено застосунок, та реагувати на перевищення порогів навантаження.

Встановлення CloudWatch Agent

На інстанс EC2 вручну встановлено CloudWatch Agent за допомогою офіційної інструкції від AWS. Агент працює як служба, яка збирає системні метрики та передає їх у CloudWatch.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

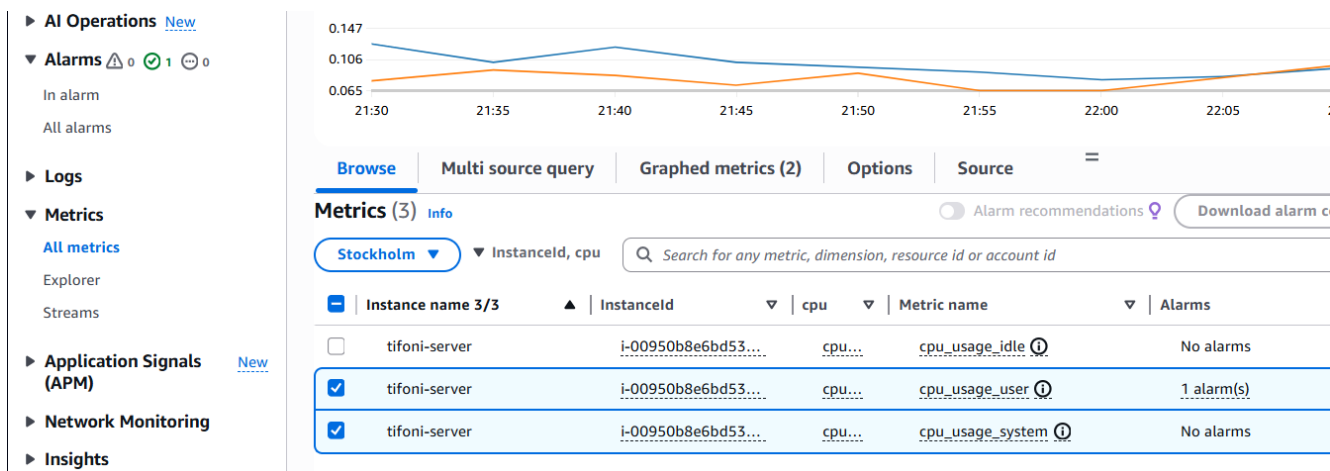


Рисунок 2.8– CloudWatch інтерфейс для спостереження за метриками

Метрики, які відстежуються

Після налаштування агент передає в CloudWatch наступні дані:

- `cpu_usage_user` – використання CPU користувачами;
- `mem_used_percent` – використання оперативної пам'яті;
- `disk_used_percent` – заповненість диска;
- інші системні показники (наприклад, навантаження на ядра, swap, IO).

Переваги

- Дає змогу аналізувати продуктивність сервера;
- Дозволяє вчасно виявити пікові навантаження;
- Можна використовувати для автоматичних дій (наприклад, масштабування, сповіщення в Telegram чи Email).

Таким чином, моніторинг через CloudWatch Agent є важливою частиною підтримки надійної роботи серверної інфраструктури.

Висновки до розділу 2

У межах розділу сформовано структуру бази даних, яка включає колекції користувачів, папок, файлів та департаментів — з підтримкою вкладеності та гнучкого доступу. Проект передбачає реалізацію авторизації через JWT, а також безпечну роботу з паролями через хешування. Зберігання файлів спроектовано через Amazon S3, де об'єкти зберігатимуться у приватному бакеті, а доступ до них —

відбуватиметься через підписані URL з обмеженим часом дії. Для зниження навантаження на сервер передбачено пряму передачу файлів між клієнтом і S3. Також у плані резервного копіювання передбачено використання механізму версіонування файлів у S3, що дозволить відновлювати видалені або змінені об'єкти. Пайплайн автоматичного деплою спроектовано через GitHub Actions, де кожне оновлення репозиторію буде запускати збірку проєкту, оновлення коду на EC2-сервері, перезапуск через pm2 та збереження конфігурації.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Створення проєкту

Розробка почалася зі створення репозиторію my-cloud-ui у GitHub. Було ініціалізовано проєкт за допомогою create-next-app. Обрано Next.js як основний фреймворк. Додано підтримку TypeScript, ESLint, TailwindCSS для інтерфейсу.

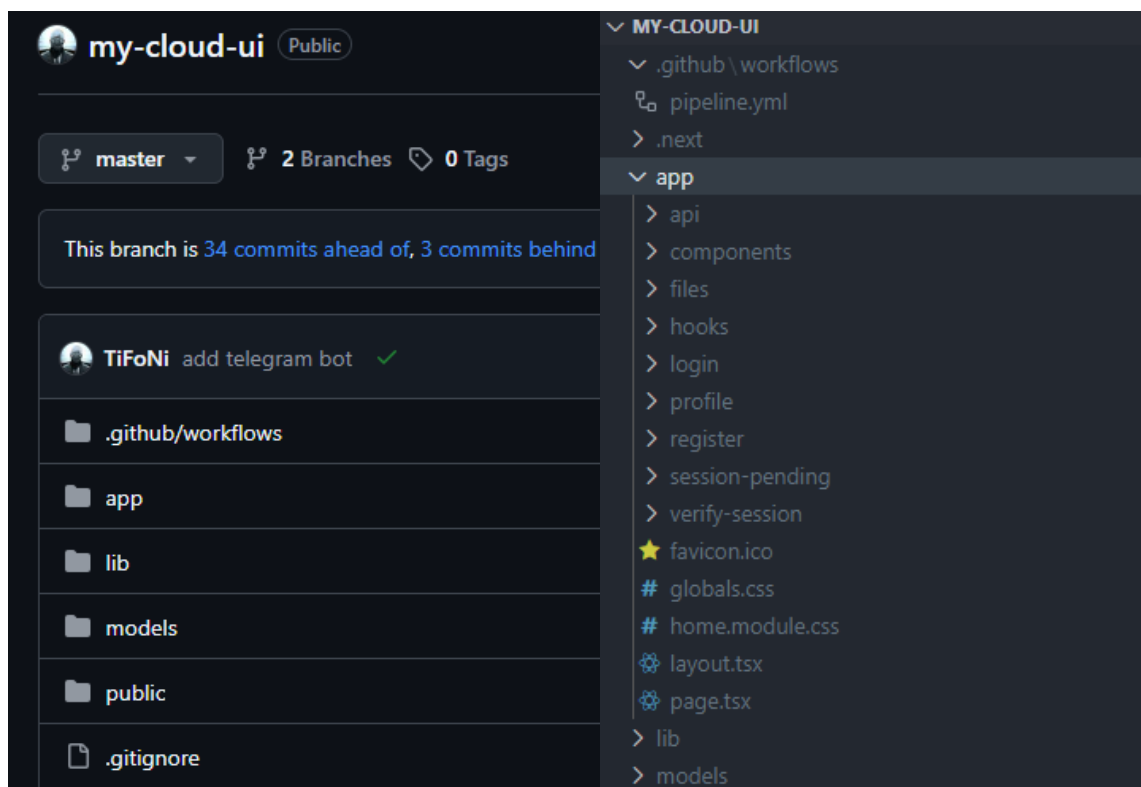


Рисунок 3.1 – Репозиторій в GitHub та структура проєкту

3.1.1 Структура проєкту

Проєкт має чітко організовану структуру директорій, яка відповідає сучасним підходам розробки з використанням Next.js 14+ (app router):

- **app/** – основна директорія для маршрутів і API:
 - **api/** – містить серверні ендпоінти:
 - **auth/** – логін, реєстрація, перевірка токена
 - **departments/** – робота з департаментами
 - **files/** – завантаження, отримання, видалення файлів
 - **folders/** – створення/отримання папок

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

- sessions/ – перевірка сесії користувача
- user/ – отримання профілю користувача
- **components/** – окремі візуальні компоненти (модалки, тулбари тощо)
- **files/, hooks/** – логіка відображення файлів і кастомні React-хуки
- **login/, register/, profile/, verify-session/, session-pending/** – сторінки інтерфейсу
- **models/** – Mongoose-схеми для MongoDB (Users, Files, Folders, Departments)
- **lib/** – утиліти: підключення до БД, перевірка токена, завантаження в S3 тощо
- **public/** – статичні ресурси
- **workflows/pipeline.yml** – GitHub Actions pipeline для CI/CD

3.2 Розгортання застосунку

Вебзастосунок було розгорнуто на віртуальній машині Amazon EC2 у регіоні eu-north-1. Машина була створена на базі Amazon Linux 2, після чого на неї було встановлено Node.js, PM2 та Git.

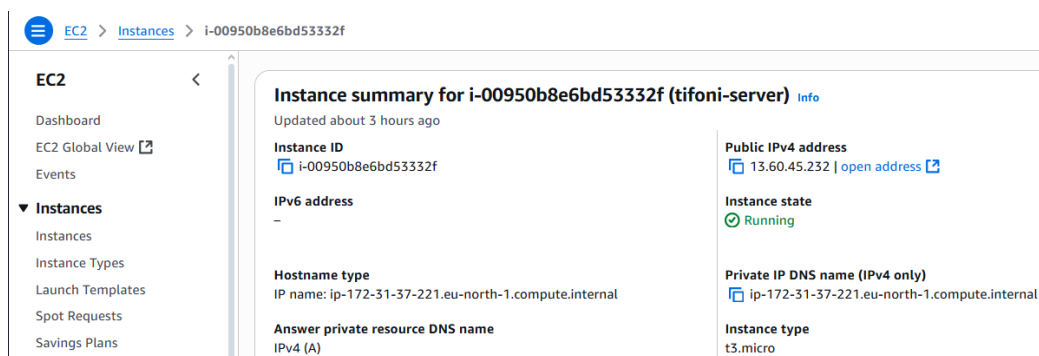


Рисунок 3.2 – Інстанс у AWS EC2

Процес деплою включає такі етапи:

1. Підключення до EC2.
2. Клонування репозиторію проєкту з GitHub
3. Встановлення залежностей за допомогою `npm install`.
4. Компіляція проєкту через `npm run build`.
5. Запуск Next.js застосунку в production-режимі за допомогою `pm2`:

Для забезпечення стабільності сервера:

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

- Застосунок працює у фоновому режимі завдяки pm2, що автоматично перезапускає сервер у випадку падіння.
- Налаштовано збереження конфігурації pm2, щоб застосунок стартував автоматично після перезавантаження машини.

Це дозволило забезпечити безперервну роботу сервісу, а також можливість легкого оновлення коду через git pull без ручного втручання.

3.3 Налаштування збереження файлів

У бакеті Amazon S3 активовано Versioning – функція зберігання кількох версій одного об’єкта. Завдяки цьому кожна зміна або видалення файлу не призводить до остаточної втрати даних. Навіть після видалення, попередня версія все ще доступна й може бути відновлена.

Політика життєвого циклу (Lifecycle Rule)

Для економії місця створено правило (Lifecycle Rule), яке:

- автоматично видаляє старі версії файлів через 30 днів після створення;
- зменшує кількість накопиченого непотрібного об’єму в S3;
- забезпечує баланс між надійністю зберігання і витратами.

У графічному інтерфейсі AWS це налаштовується в розділі Management → Lifecycle rules для бакету.

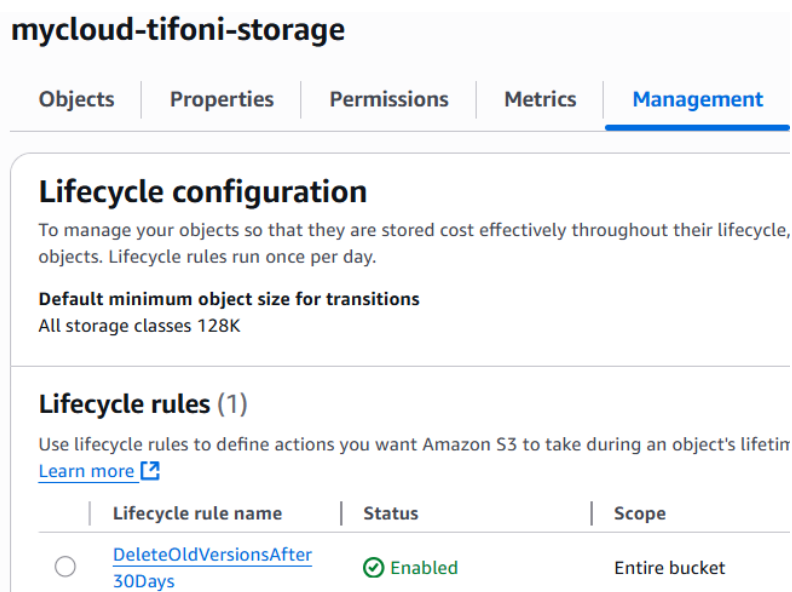


Рисунок 3.3 – Правило життєвого циклу файлу, видаленого з версіонуванням

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

Коли версіонування увімкнено й користувач видаляє файл, файл фізично не зникає. Замість цього:

- AWS додає спеціальний об'єкт – Delete Marker;
- Попередні версії файлу залишаються в бакеті;
- Коли відкриваєш файл у UI, здається, що його нема – але насправді є попередні версії.

Це дає низку можливостей в подальшому:

1. Відновити видалений файл

- У AWS S3 можна зайти в об'єкт → подивитись версії → знайти останню не Delete Marker версію → зробити її актуальною.

2. Очищати автоматично

Через Lifecycle Rule можна:

- автоматично видаляти Delete Markers (через N днів);
- або очищати старі версії файлів через 30 днів;
- можна комбінувати: залишити тільки 1 останню версію.

3. Використати як кошик

Це буквально аналог кошика:

- Користувач видалив → файл сховався → є шанс відновити;
- Можна зробити UI-кошик у застосунку, який покаже такі версії (через `listObjectVersions`);
- І дати кнопку “Відновити файл” або “Очистити остаточно”.

3.4 Тестування працездатності розробленого додатку

На цьому етапі виконається перевірка функціональності вебзастосунку відповідно до поставлених вимог. Основна мета тестування — переконатися, що всі ключові функції (реєстрація, авторизація, завантаження й перегляд файлів, розмежування доступу, створення підрозділів тощо) працюють коректно, а також виявити можливі помилки до впровадження застосунку в реальне середовище. Тестування проводиться вручну у вебінтерфейсі користувача, а також через перевірку API-запитів.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

3.4.1 Початок роботи зі сторони користувача

Для початку, зареєструємо нового користувача:

Рисунок 3.4 – Форма реєстрації

Після реєстрації дані про користувача вже з'являться в MongoDB

```
_id: ObjectId('68585f31f7276dffe7c2740c')
email: "iitbgtifoni@gmail.com"
password: "$2b$10$4YLLBKT1nAWRq5Jwa8TsOupx1I/SpXE3M2G6rnJMilbtJair0rmgC"
nickname: "NoName"
role: "user"
department: null
verifiedSessions: Array (empty)
__v: 0
```

Рисунок 3.5 – Дані користувача, внесені в базу після реєстрації

Для успішного входу, користувач повинен перейти за посиланням, що надійшло на пошту, а поки він це робить, його тримає на сторінці Session Not Verified

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

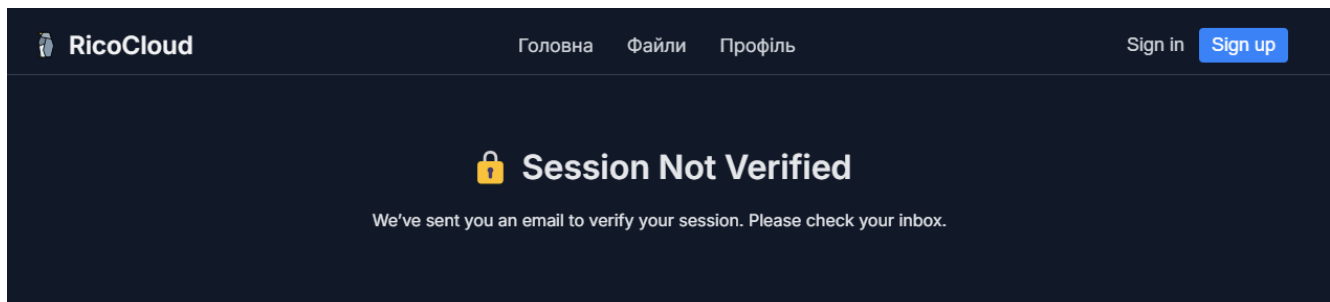


Рисунок 3.6 – Сторінка, поки сесія не підтвердиться користувачем

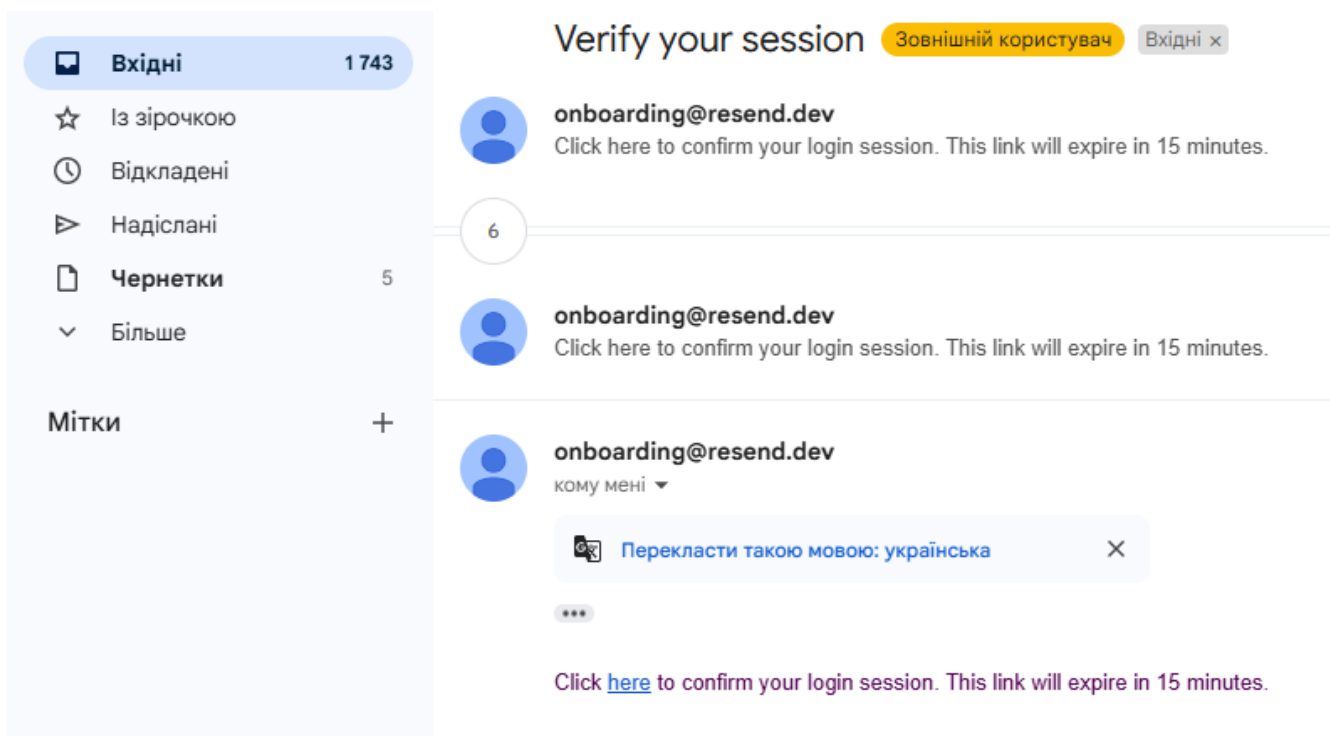


Рисунок 3.7 – Вигляд листа для підтвердження сесії з посиланням на сайт

Після успішного входу, буде дано доступ до функціоналу управління файлами в нашому хмарному сховищі

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

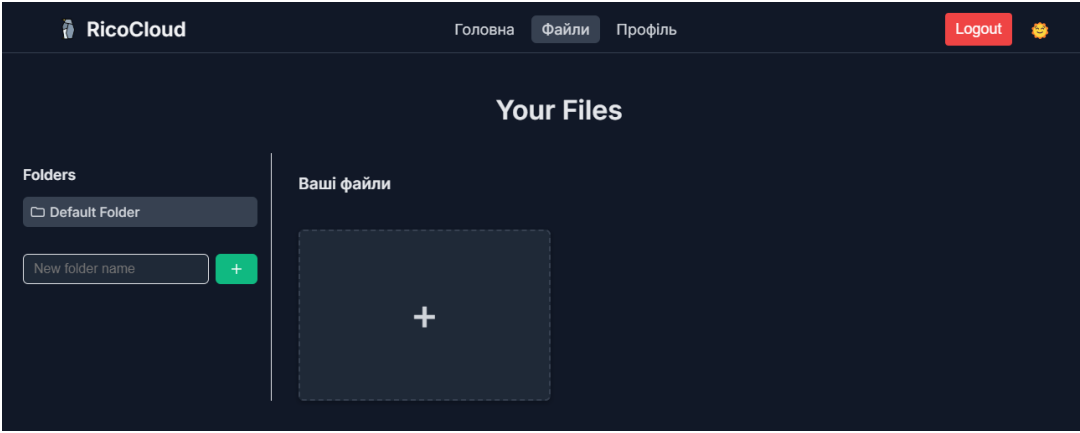


Рисунок 3.8 – Початковий вигляд додатку

Користувач може вже додавати файли, видаляти їх, або завантажувати

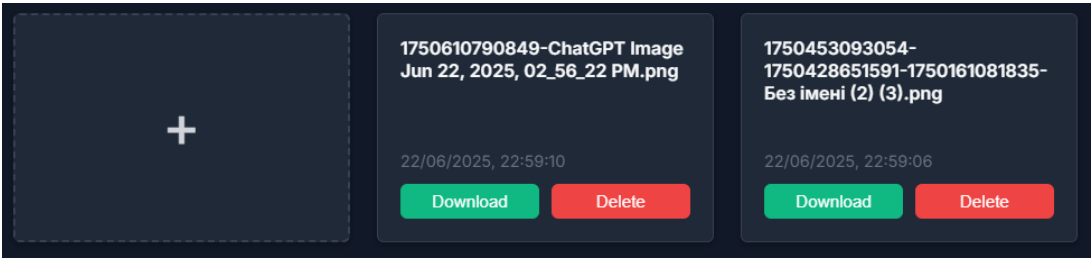


Рисунок 3.9 – Додані користувачем файли, що зберігаються в s3 бакеті

Через signed URLs запити на скачування файлу відправляються напряму до s3 бакета, а не проходять через сервер:

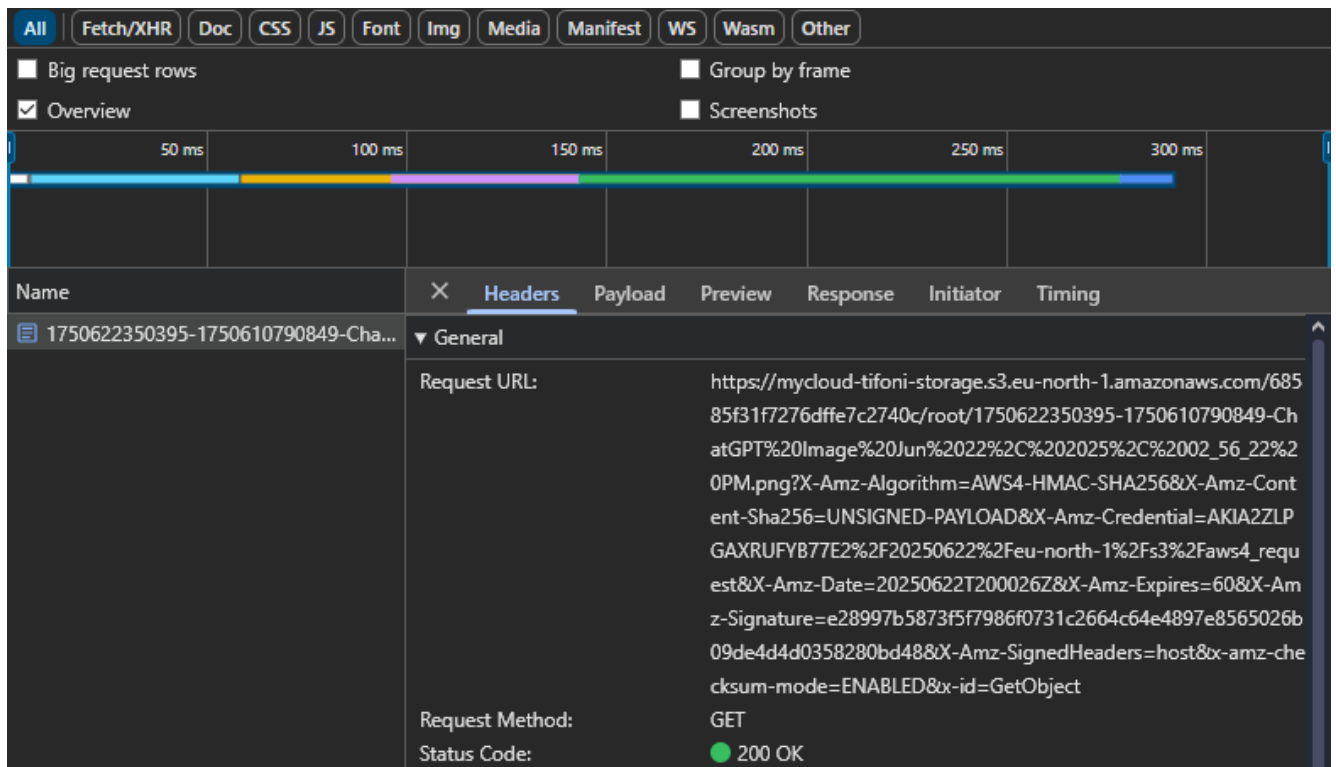


Рисунок 3.10 – Запит напряму до s3 бакету на отримання файлу

Видалення файлу покличе за собою спрацювання декількох запитів, що в сумі видалять файл з бакету, видалять його метадані з бази даних та запросять оновлення папок та файлів, для того, щоб не потрібно було перезавантажувати сторінку, щоб побачити результат, так є візуальне узгодження нашим діям

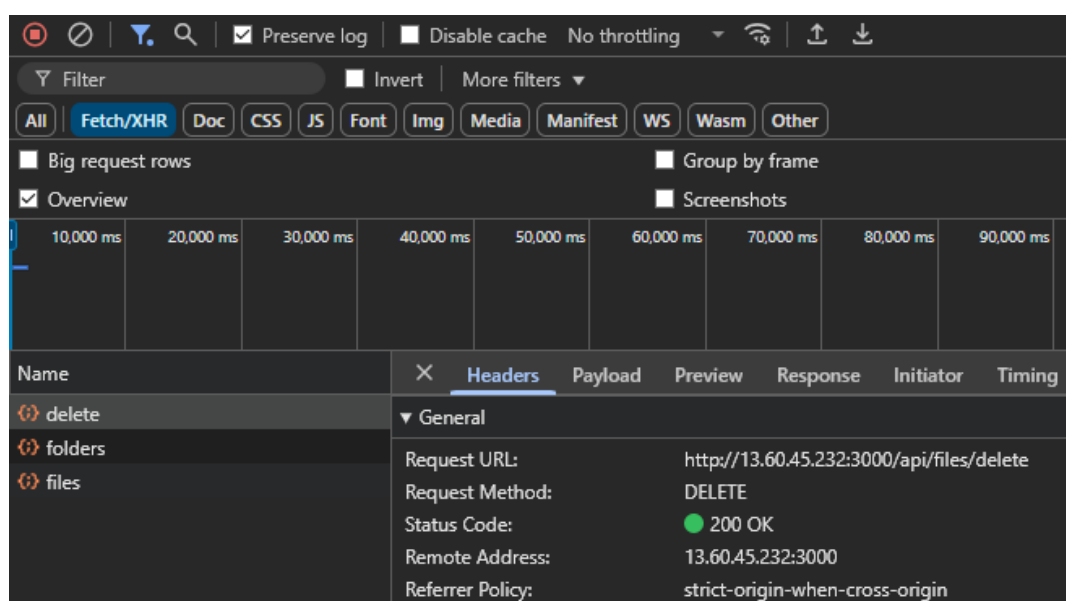


Рисунок 3.11 – Запит на видалення та 2 запити для оновлення

Користувач також може створювати папки, а також підпапки (на даний момент без ліміту на кількість вкладених папок)

Folders

📁 Default Folder

📁 1

×

↳ 📁 2

×

📁 3

×

New folder name

+

Рисунок 3.12 – Структура папок-підпапок

Є можливість видалення папки, що покличе за собою видалення всіх підпапок та файлів, що в них знаходяться (як з бази даних так і з s3 бакету)

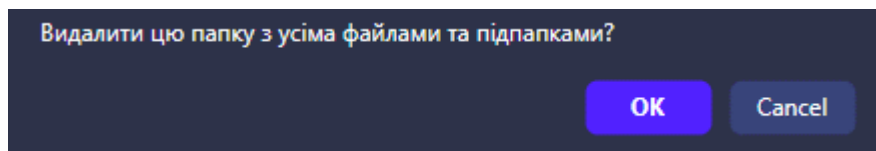


Рисунок 3.13 – Алерт про видалення папки

Користувач (працівник певного відділу) може обрати відділ, до якого він входить, після чого буде бачити файли інших користувачів цього ж відділу

Виберіть відділ:

First

Second

Рисунок 3.14 – Селектор вибору відділу

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				52
Змн.	Арк.	№ докум.	Підпис	Дата		

Перевіримо:

Користувач А створює 3 папки, в першу кладе файл, має можливість видалення як папки, так і файлу

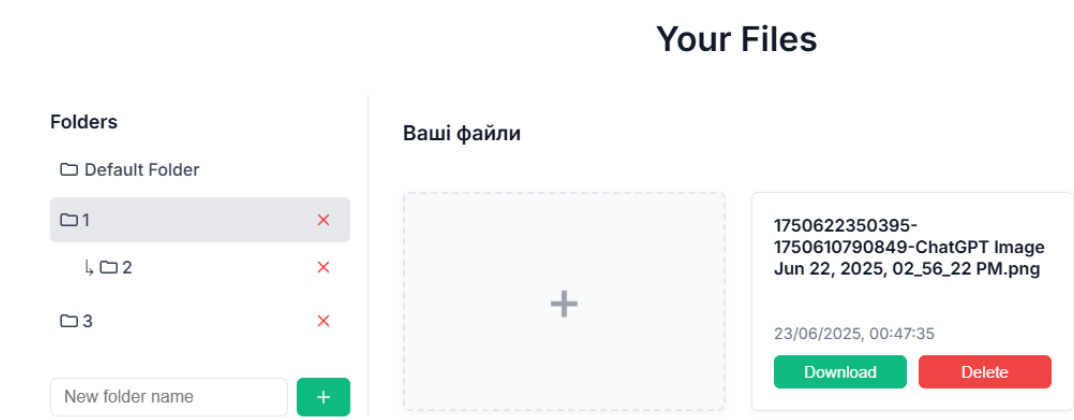


Рисунок 3.15 – Створені 3 папки та завантажений файл Користувача А

Користувач Б знаходячись в відділі Second бачить папки і файли, створені першим користувачем, але не має змоги їх видаляти, лише переглядати та завантажувати

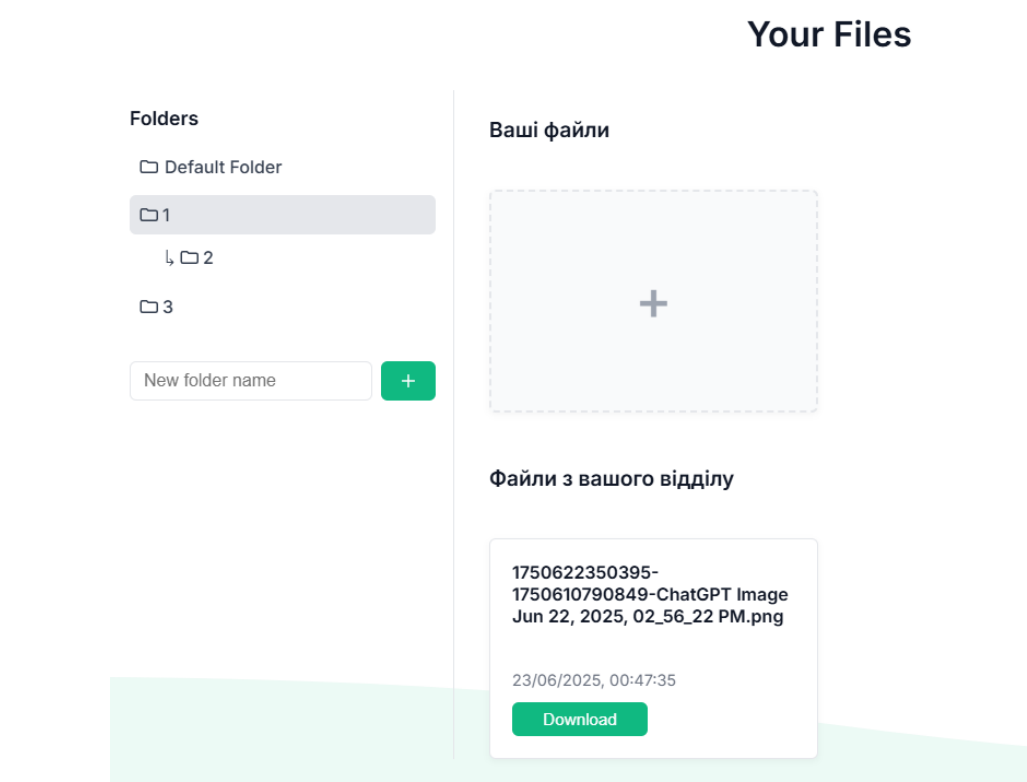


Рисунок 3.16 – Створені 3 папки та завантажений файл Користувача Б

Якщо видалити папку 1, видалиться і дочірня папка 2

До видалення папки 1:



Рисунок 3.17 – Вигляд сайту та вміст бази до видалення

Після видалення папки 1:



Рисунок 3.18 – Вигляд сайту та вміст бази після видалення

Можливість зміни теми зі світлої на темну:

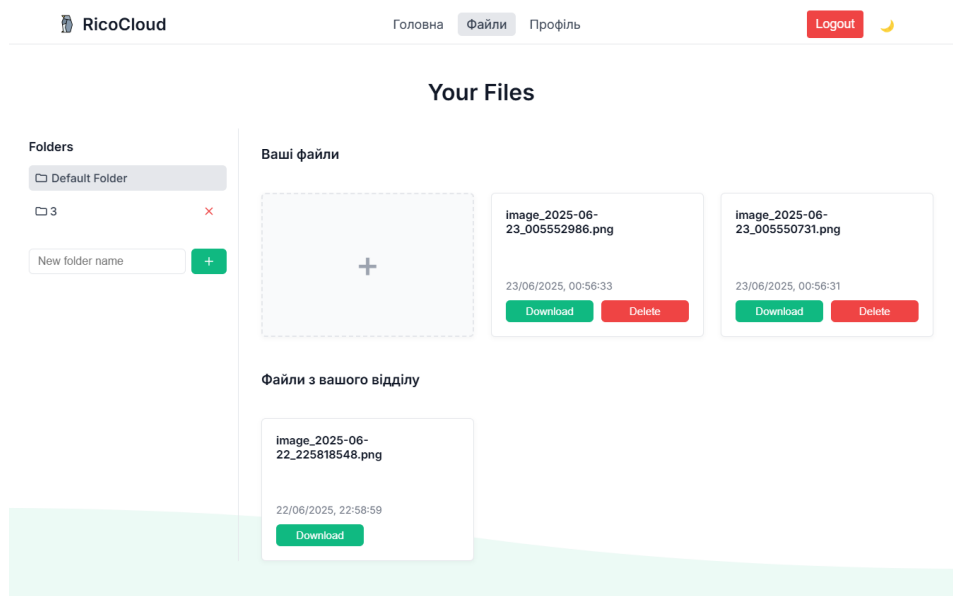


Рисунок 3.19 – Світла тема

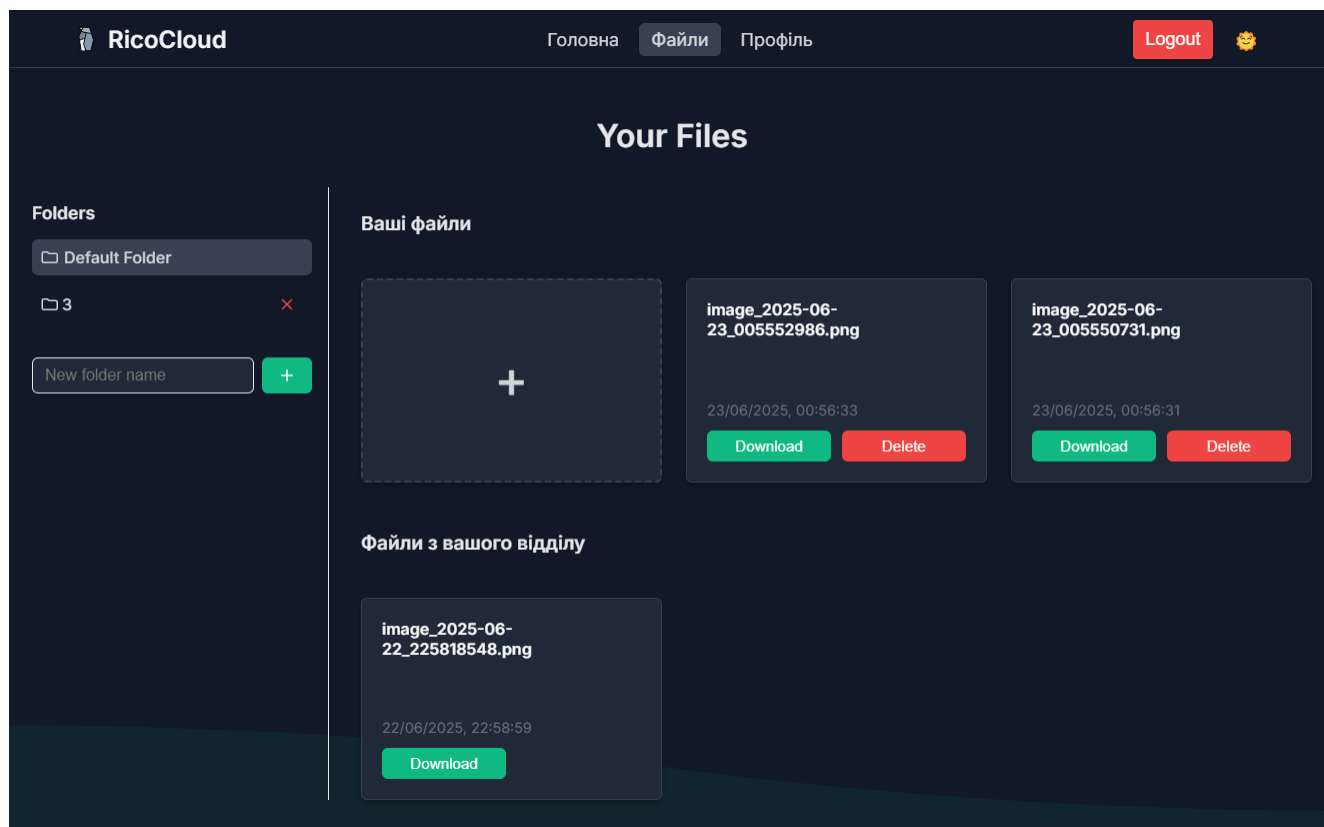


Рисунок 3.20 – Нічна тема

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				55
Змн.	Арк.	№ докум.	Підпис	Дата		

3.4.2 Перевірка можливостей зі сторони розробника

Внесемо деякі зміни з наш проєкт, створимо коміт, та запустим його відразу в основну вітку (перед цим можна локально прогнати білди для впевненості в працездатності)

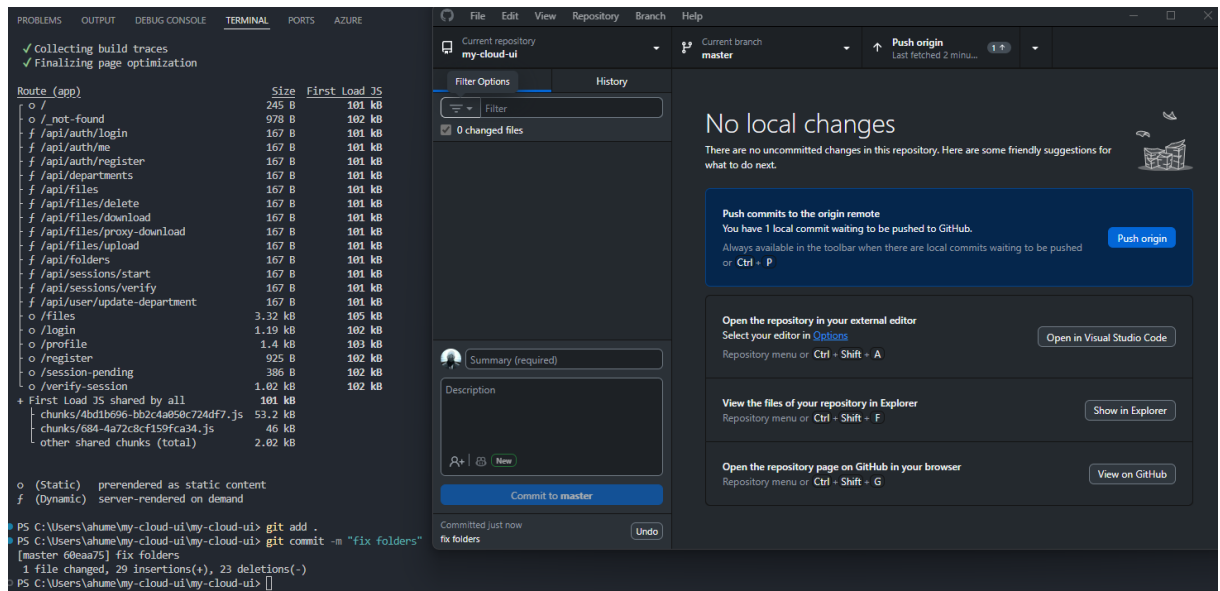


Рисунок 3.21 – Пуш коміту після локального білду

В GitHub Actions можна побачити, що запустились процеси, описані в пайп-лайнні:

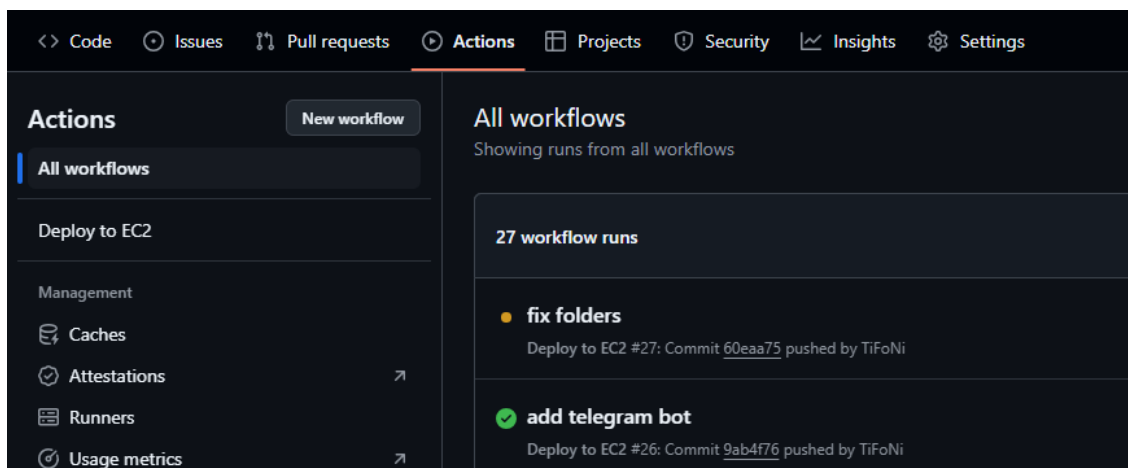


Рисунок 3.22 – Відображення процесу білда на GitHub Actions

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				56
Змн.	Арк.	№ докум.	Підпис	Дата		

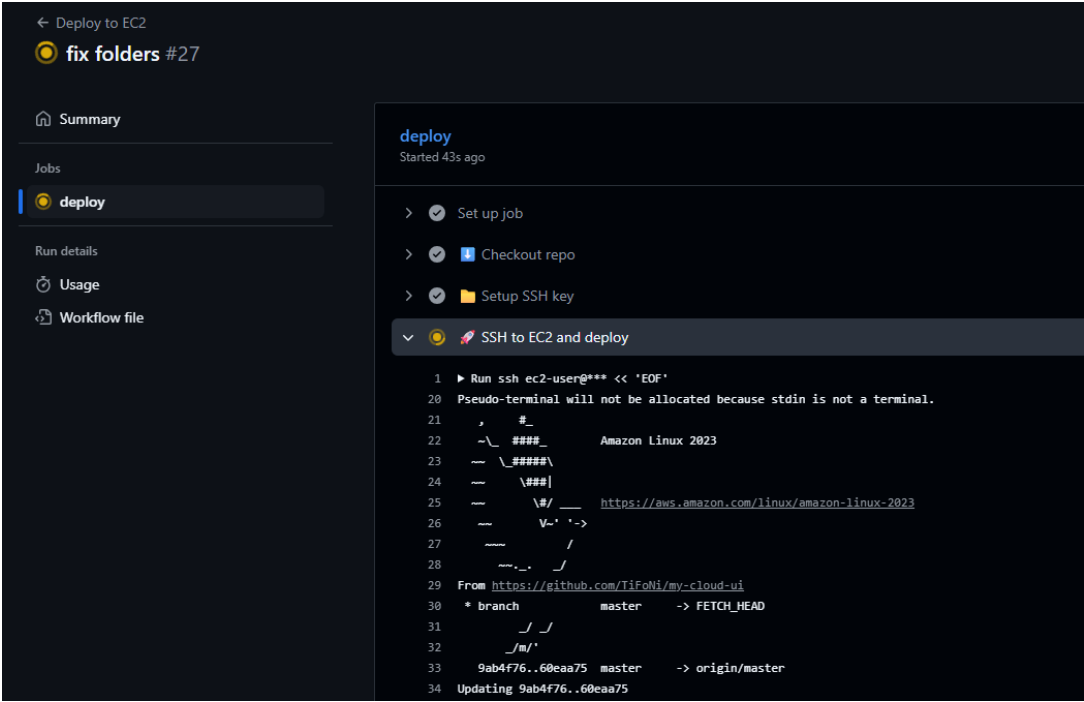


Рисунок 3.23 – Зручна перевірка, на якому етапі білд проєкту

А по закінченню білда, при успішному завершенні, буде оновлено процес на новий, з оновленим кодом



Рисунок 3.24 – Виконання успішного білду, видно що сервер online



Рисунок 3.25 – Повідомлення по закінченню виконання пайплайну на телеграм

А також прийде повідомлення з результатом виконання білда в телеграм, для зручності

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				57
Змн.	Арк.	№ докум.	Підпис	Дата		

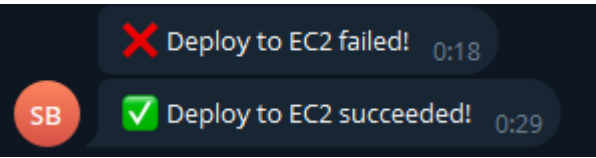


Рисунок 3.26 – Види повідомлень бота, в залежності від результату збірки

Якщо спробувати збілдити щось, результат чого буде фатальним, то сервер не впаде, а продовжить роботу зі старим, працюючим кодом

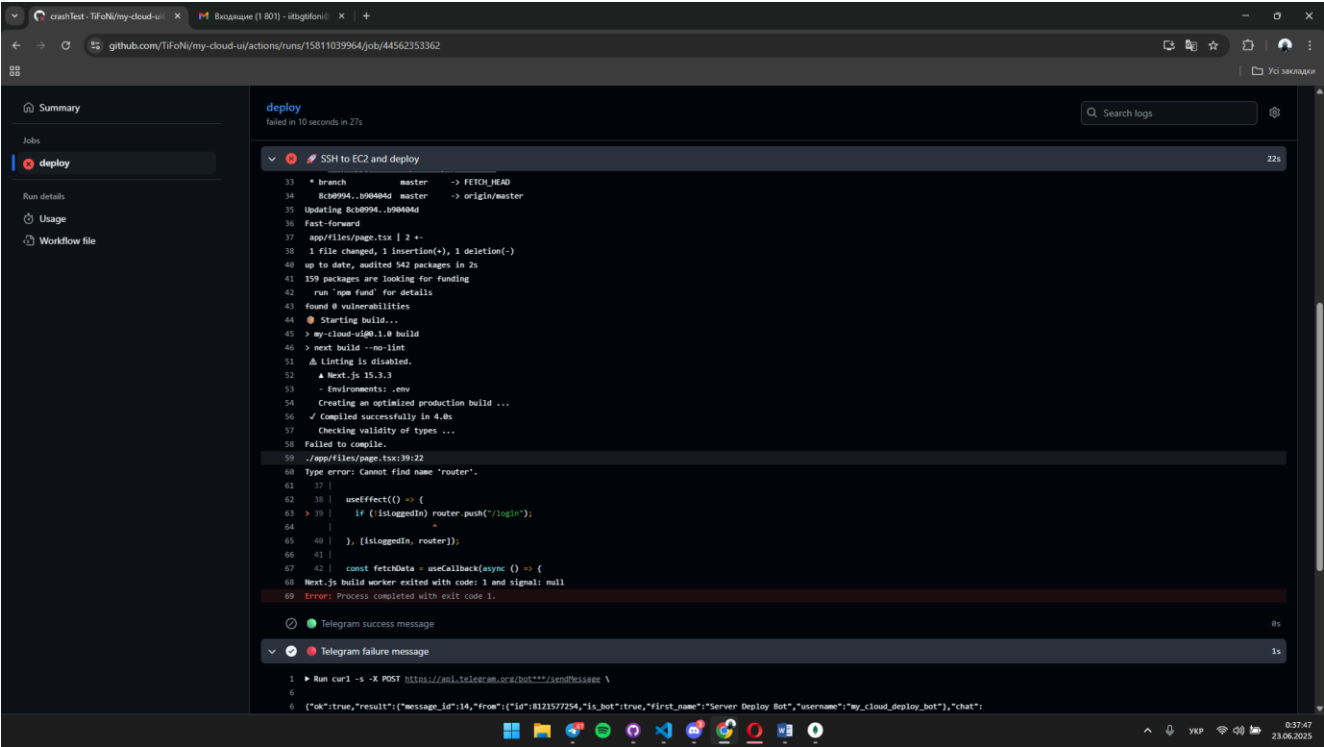


Рисунок 3.27 – Результат невдалого деплою

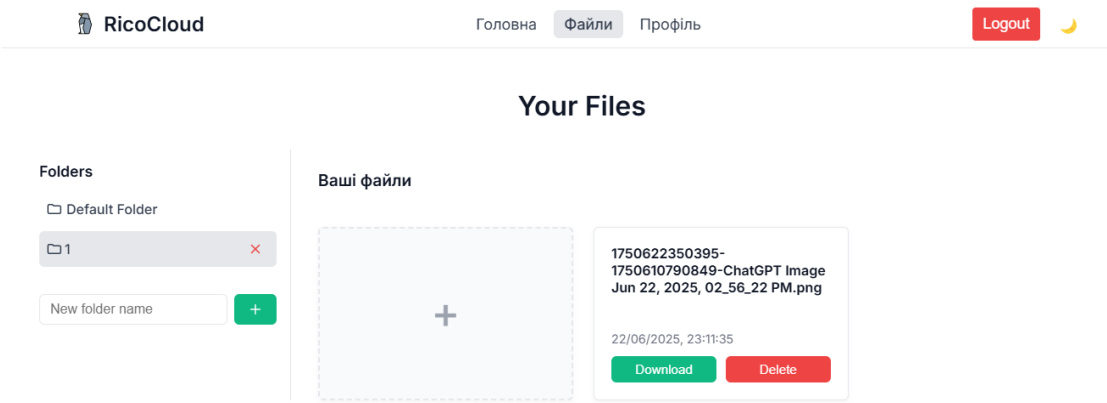


Рисунок 3.28 – Повноцінно робочий сайт, навіть якщо білд нової версії впав

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				58
Змн.	Арк.	№ докум.	Підпис	Дата		

Можна переглядати файли з маркером **видалені** (Delete marker), прямо в s3 бакеті:

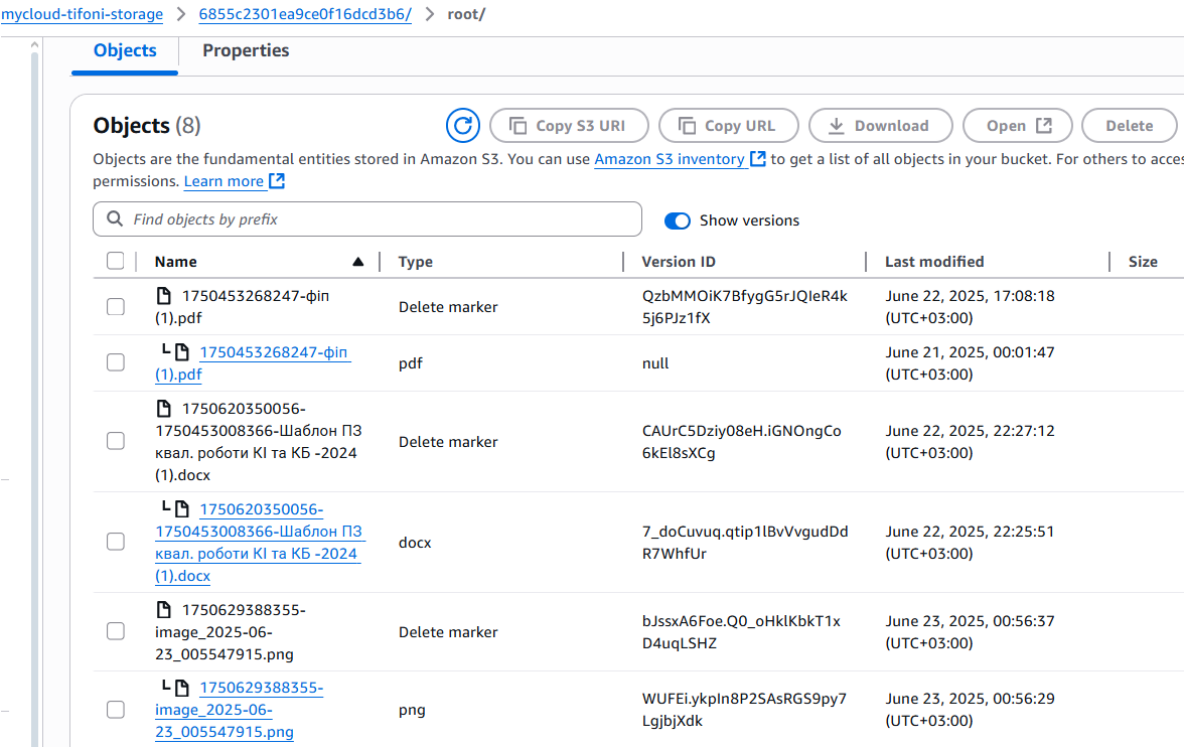


Рисунок 3.29 – Видалені файли лежать в бакеті, але приховані маркером „delete„

Також є можливість слідкувати за метриками, зібраними агентом, для подальшого покращення інстансу, при потребі:

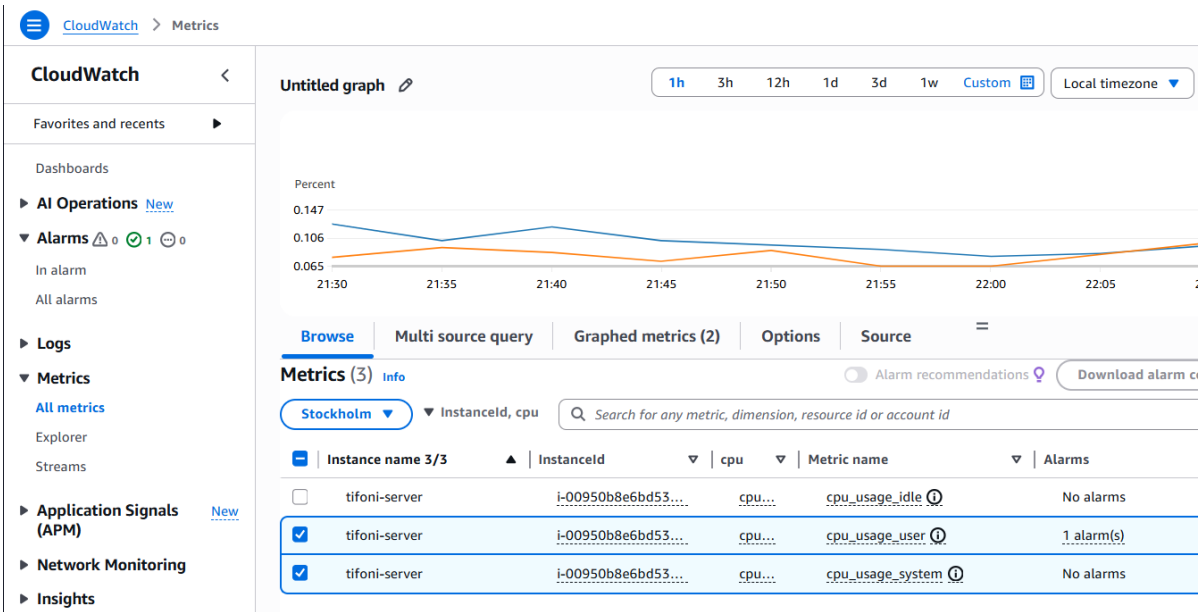


Рисунок 3.30 – Моніторинг сру сервера за допомогою CloudWatch

Висновки до розділу 3

У третьому розділі було розроблено, протестовано та проаналізовано працездатність приватного хмарного файлового сховища. Здійснено налаштування інфраструктури для автоматичного розгортання застосунку за допомогою CI/CD через GitHub Actions, що забезпечує швидке оновлення сервісу після змін у коді. Для моніторингу використано сервіс Amazon CloudWatch, який дозволяє відстежувати основні метрики стану EC2-сервера та своєчасно реагувати на перевантаження. Реалізовано механізм резервного копіювання через версіонування в Amazon S3, що забезпечує збереження попередніх версій файлів навіть після їх видалення.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		60

ВИСНОВКИ

У межах кваліфікаційної роботи було реалізовано повноцінний вебзастосунок для безпечного зберігання та обміну файлами всередині організації з підтримкою багаторівневого доступу, резервного копіювання, моніторингу та автоматичного розгортання. Систему спроектовано з урахуванням вимог до безпеки, масштабованості та зручності користування. Завдяки використанню сучасних технологій (Next.js, MongoDB, AWS S3, EC2, GitHub Actions, CloudWatch) вдалося створити інфраструктуру, яка забезпечує стабільну роботу в умовах підвищеного навантаження. Усі поставлені завдання були успішно виконані.

За результатами Розділу 1, було проаналізовано існуючі рішення для хмарного зберігання (Google Drive, Dropbox, Nextcloud тощо) та сформульовано ключові вимоги до майбутньої системи. Розроблено архітектуру програмного комплексу, що базується на AWS-сервісах і Next.js. Обґрунтовано вибір бази даних MongoDB, технологій фронтенду, хмарної інфраструктури, системи CI/CD та моніторингу. Визначено принципи взаємодії між клієнтом, сервером і хмарним сховищем. Зібрано перелік функціональних та нефункціональних вимог до системи.

За результатами Розділу 2, реалізовано повний стек застосунку: створено інтерфейс, логіку авторизації з підтвердженням сесій, систему керування папками та файлами з підтримкою підрозділів. Налаштовано підписані URL для безпечного скачування файлів, щоб мінімізувати навантаження на сервер. Запроваджено моніторинг через CloudWatch Agent для збору метрик із EC2. Виконано зберігання файлів з поміткою через версіонування. Налаштовано GitHub Actions для CI/CD з повідомленнями про успішні та помилкові деплої. Проєкт має чисту структуру з логічним поділом на компоненти, сторінки та API, що робить код зручним для масштабування та підтримки.

За результатами Розділу 3, проведено перевірку всіх функціональних можливостей системи: створення облікового запису, підтвердження сесії, робота з файлами, видалення, скачування, перевірка прав доступу на основі підрозділів. Демон-

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				61
Змн.	Арк.	№ докум.	Підпис	Дата		

стровано роботу пайплайна, який автоматично розгортає застосунок на EC2. Показано роботу моніторингу, поведінку системи під навантаженням, автоматичну генерацію повідомлень у Telegram при оновленнях. Підтверджено, що версіонування дозволяє відновлювати видалені файли, а база даних фіксує всі зміни. Уся система успішно виконує свої задачі в реальному середовищі.

У результаті виконання кваліфікаційної роботи було повністю реалізовано проєкт хмарного вебсховища для організації, що поєднує зручний інтерфейс, безпечну авторизацію, логіку доступу за підрозділами, ефективне зберігання файлів і сучасну хмарну інфраструктуру. Усі поставлені завдання – від вибору архітектури до CI/CD, моніторингу та резервного копіювання – реалізовано та перевірено на практиці. Система є готовим продуктом, який може бути розгорнутий в організації для повсякденного використання з можливістю подальшого масштабування.

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Офіційна документація Next.js. – <https://nextjs.org/docs>
2. MongoDB Manual. – <https://www.mongodb.com/docs/manual>
3. Mongoose Documentation. – <https://mongoosejs.com/docs/guide.html>
4. AWS S3 Documentation. – <https://docs.aws.amazon.com/s3/index.html>
5. AWS SDK for JavaScript v3. – <https://docs.aws.amazon.com/AWSJavaScriptSDK/v3>
6. CloudWatch Agent User Guide. – <https://docs.aws.amazon.com/Amazon-CloudWatch/latest/monitoring/Install-CloudWatch-Agent.html>
7. GitHub Actions Documentation. – <https://docs.github.com/en/actions>
8. Tailwind CSS Documentation. – <https://tailwindcss.com/docs>
9. bcrypt Documentation. – <https://github.com/kelektiv/node.bcrypt.js>
10. jsonwebtoken Documentation. – <https://github.com/auth0/node-jwebtoken>
11. shadcn/ui – <https://ui.shadcn.com>
12. Telegram Bot API. – <https://core.telegram.org/bots/api>
13. EC2 Documentation. – <https://docs.aws.amazon.com/ec2>
14. The Best Way To Learn AWS – <https://dev.to/kylegalbraith/the-best-way-to-learning-aws-is-to-start-using-it>
15. AWS Well-Architected Framework – <https://aws.amazon.com/architecture/well-architected/?wa-lens-whitepapers.sort-by=item.additionalFields.sortDate&wa-lens-whitepapers.sort-order=desc&wa-guidance-whitepapers.sort-by=item.additionalFields.sortDate&wa-guidance-whitepapers.sort-order=desc>
16. Що таке CI/CD – https://itedu.center/ua/blog/review/what-is-ci-cd/?srsId=AfmBOoqestTZNLWYkIysk6fDPkJTvbrIeaFQ9paMutR9TdtzyyZ4H_S-

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

ДОДАТКИ

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

Основні API запити:

Отримання файлів:

```

import { NextRequest, NextResponse } from "next/server";
import { connectDB } from "@lib/db";
import { verifyToken } from "@lib/utills/verifyToken";
import File from "@models/File";
import { User } from "@models/User";

export async function GET(req: NextRequest) {
  await connectDB();

  const token = req.headers.get("authorization")?.split(" ")[1];
  const userId = verifyToken(token);
  if (!userId)
    return NextResponse.json({ error: "Unauthorized" }, { status: 401 });

  const { searchParams } = new URL(req.url);
  const folderIdRaw = searchParams.get("folderId");
  const folderId = folderIdRaw === "null" ? null : folderIdRaw;

  const currentUser = await User.findById(userId);
  if (!currentUser?.department) {
    const query: { userId: string; folderId?: string | null } = { userId };
    if (folderIdRaw !== null) {
      query.folderId = folderId;
    }

    const ownFiles = await File.find(query).sort({ uploadedAt: -1 });
    return NextResponse.json({ ownFiles, deptFiles: [] });
  }

  const usersInDept = await User.find({ department: currentUser.department });
  const userIds: string[] = usersInDept.map((u) => u._id.toString());

  const queryAll: {
    userId: { $in: string[] };
    folderId?: string | null;
  } = {
    userId: { $in: userIds },
  };

  if (folderIdRaw !== null) {
    queryAll.folderId = folderId;
  }

  const allFiles = await File.find(queryAll).sort({ uploadedAt: -1 });

  const ownFiles = allFiles.filter((file) => file.userId.toString() === userId);
  const deptFiles = allFiles.filter(
    (file) => file.userId.toString() !== userId
  );

```

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

```
);

return NextResponse.json({ ownFiles, deptFiles });
}
```

Видалення файлів:

```
import { NextRequest, NextResponse } from "next/server";
import { connectDB } from "@lib/db";
import { verifyToken } from "@lib/Utils/verifyToken";
import File from "@models/File";
import { S3Client, DeleteObjectCommand } from "@aws-sdk/client-s3";

const s3 = new S3Client({
  region: process.env.AWS_REGION!,
  credentials: {
    accessKeyId: process.env.AWS_ACCESS_KEY_ID!,
    secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY!,
  },
});

export async function DELETE(req: NextRequest) {
  await connectDB();

  const token = req.headers.get("authorization")?.split(" ")[1];
  const userId = verifyToken(token);
  if (!userId)
    return NextResponse.json({ error: "Unauthorized" }, { status: 401 });

  const { fileId } = await req.json();
  const file = await File.findOne({ _id: fileId, userId });
  if (!file)
    return NextResponse.json({ error: "File not found" }, { status: 404 });

  await s3.send(
    new DeleteObjectCommand({
      Bucket: process.env.AWS_BUCKET_NAME!,
      Key: file.s3Key,
    })
  );

  await File.deleteOne({ _id: fileId });

  return NextResponse.json({ message: "File deleted" });
}
```

Скачування файлів:

```
import { NextRequest, NextResponse } from "next/server";
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";
```

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

```

import { getSignedUrl } from "@aws-sdk/s3-request-presigner";
import { verifyToken } from "@lib/Utils/verifyToken";

const s3 = new S3Client({
  region: process.env.AWS_REGION!,
  credentials: {
    accessKeyId: process.env.AWS_ACCESS_KEY_ID!,
    secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY!,
  },
});

export async function GET(req: NextRequest) {
  const { searchParams } = req.nextUrl;
  const s3Key = searchParams.get("s3Key");
  const token = searchParams.get("token");

  if (!token || !verifyToken(token)) {
    return NextResponse.json({ error: "Unauthorized" }, { status: 401 });
  }

  if (!s3Key) {
    return NextResponse.json({ error: "Missing s3Key" }, { status: 400 });
  }

  try {
    const command = new GetObjectCommand({
      Bucket: process.env.AWS_BUCKET_NAME!,
      Key: s3Key,
    });

    const signedUrl = await getSignedUrl(s3, command, { expiresIn: 60 });

    return NextResponse.redirect(signedUrl);
  } catch (err) {
    console.error("Download error:", err);
    return NextResponse.json(
      { error: "Failed to generate signed URL" },
      { status: 500 }
    );
  }
}

```

Завантаження файлів:

```

import { NextRequest, NextResponse } from "next/server";
import { S3Client, PutObjectCommand } from "@aws-sdk/client-s3";
import { connectDB } from "@lib/db";
import { verifyToken } from "@lib/Utils/verifyToken";
import File from "@models/File";

const s3 = new S3Client({

```

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		67

```

region: process.env.AWS_REGION!,
credentials: {
  accessKeyId: process.env.AWS_ACCESS_KEY_ID!,
  secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY!,
},
});

export async function POST(req: NextRequest) {
  await connectDB();

  const token = req.headers.get("authorization")?.split(" ")[1];
  const userId = verifyToken(token);
  if (!userId)
    return NextResponse.json({ error: "Unauthorized" }, { status: 401 });

  const formData = await req.formData();
  const file = formData.get("file") as File;
  const folderId = formData.get("folderId")?.toString() || null;

  if (!file)
    return NextResponse.json({ error: "No file uploaded" }, { status: 400 });

  const buffer = Buffer.from(await file.arrayBuffer());
  const s3Key = `${userId}/${folderId || "root"}/${Date.now()}-${file.name}`;

  try {
    await s3.send(
      new PutObjectCommand({
        Bucket: process.env.AWS_BUCKET_NAME!,
        Key: s3Key,
        Body: buffer,
      })
    );
  }

  const url = `https://${process.env.AWS_BUCKET_NAME}.s3.${process.env.AWS_REGION}.amazonaws.com/${s3Key}`;

  await File.create({ userId, filename: file.name, s3Key, url, folderId });

  return NextResponse.json({ message: "File uploaded", url });
} catch (err) {
  console.error("Upload error:", err);
  return NextResponse.json({ error: "Upload failed" }, { status: 500 });
}
}

```

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

Код пайплайну (.github\workflows\pipeline.yml)

```

name: Deploy to EC2
on:
  push:
    branches: [master]
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: 📄 Checkout repo
        uses: actions/checkout@v3
      - name: 🗑 Setup SSH key
        run: |
          mkdir -p ~/.ssh
          echo "${{ secrets.SSH_PRIVATE_KEY }}" > ~/.ssh/id_rsa
          chmod 600 ~/.ssh/id_rsa
          ssh-keyscan ${{ secrets.EC2_HOST }} >> ~/.ssh/known_hosts
      - name: 🔌 SSH to EC2 and deploy
        run: |
          ssh ec2-user@${{ secrets.EC2_HOST }} << 'EOF'
            set -e
            cd my-cloud-ui
            git pull origin master
            npm install
            echo "🚧 Starting build..."
            NODE_ENV=production NEXT_IGNORE_TYPECHECK=1 npm run build
            echo "🗑 Deleting old process if exists..."
            pm2 delete my-cloud-ui || true
            echo "🔌 Starting new process..."
            pm2 start "npm run start" --name my-cloud-ui
            pm2 save
          EOF
      - name: ☑ Telegram success message
        if: success()
        run: |
          curl -s -X POST https://api.telegram.org/bot${{ secrets.TG_TOKEN }}/sendMessage \
            -d chat_id=${{ secrets.TG_CHAT_ID }} \
            -d text="✔ Deploy to EC2 succeeded!"
      - name: ● Telegram failure message
        if: failure()
        run: |
          curl -s -X POST https://api.telegram.org/bot${{ secrets.TG_TOKEN }}/sendMessage \
            -d chat_id=${{ secrets.TG_CHAT_ID }} \
            -d text="✖ Deploy to EC2 failed!"

```

		Ушаков І.В.			ККІКБ.КР.Б – 123 – 15 – ПЗ	Арк.
		Шелуха О.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

Конфігураційний файл для агента AWS моніторингу
(/opt/aws/amazon-cloudwatch-agent/bin/config.json)

```
{
  "metrics": {
    "metrics_collected": {
      "cpu": {
        "measurement": ["cpu_usage_idle", "cpu_usage_user", "cpu_usage_system"],
        "totalcpu": true
      },
      "mem": {
        "measurement": ["mem_used_percent"]
      },
      "disk": {
        "measurement": ["used_percent"],
        "resources": ["*"]
      }
    },
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    }
  }
}
```