

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Мережеве сховище NAS (Network Attached Storage)
з використанням React.js, Node.js, MongoDB»

на здобуття освітнього ступеня магістра
зі спеціальності ЕЗ Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

*Кваліфікаційна робота містить результати власних досліджень.
Використання ідей, результатів і текстів інших авторів мають посилання
на відповідне джерело*

(підпис)

Максим МАРОХОНЬКО
(Ім'я, ПРІЗВИЩЕ здобувача)

Виконав:
здобувач вищої освіти
група КНДМ-61

Максим МАРОХОНЬКО

Керівник:
науковий ступінь,
вчене звання

Олена ШИКУЛА
д.ф.-м.н., професор

Рецензент:
науковий ступінь,
вчене звання

(Ім'я, ПРІЗВИЩЕ)

Київ 2025

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Магістр

Спеціальність ЕЗ Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедру Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
« _____ » _____ 2025 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

_____ Марохонько Максиму Олеговичу

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Мережеве сховище NAS (Network Attached Storage) з використанням React.js, Node.js, MongoDB

керівник кваліфікаційної роботи Олена ШИКУЛА д.ф.-м.н., професор,

(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467

2. Строк подання кваліфікаційної роботи «26» грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, мова програмування JavaScript; середовище програмування Visual Studio Code, MongoDB Compass.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

Аналіз існуючих мережевих сховищ NAS, аналіз їх функціональних можливостей, їх переваг та недоліків.

Розробка веб-додатку для мережевого сховища NAS.

Розробка адаптивних версій веб-додатку.

5. Перелік графічного матеріалу: *презентація*

1. Огляд архітектури мережевих сховищ NAS: діаграма, що відображає основні компоненти типових NAS-систем та їх взаємодію (клієнт-сервер, бази даних, файлові системи).

2. Блок-схема роботи мережевого сховища: схема, що ілюструє процеси завантаження, зберігання, доступу та управління файлами в розробленому додатку.

3. Екранні форми веб-додатку: демонстрація інтерфейсу користувача для управління файлами, налаштуваннями доступу та адмініструванням NAS.

4. Екранні форми адаптивних версій додатку: зображення інтерфейсу для смартфонів та планшетів, що показують адаптивність дизайну та функціоналу.

5. Фрагменти програмного коду: приклади коду для реалізації ключових функцій додатку (наприклад, API для роботи з файлами в Node.js, компоненти React.js для інтерфейсу, запити до MongoDB).

6. Дата видачі завдання «15» вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.09-27.09.25	виконано
2	Огляд систем комп'ютерної математики	28.09-11.10.25	виконано
3	Визначення функціоналу та основних характеристик додатку	12.10-25.10.25	виконано
4	Вибір інструментів для розробки додатку	26.10-04.11.25	виконано
5	Розробка додатку для роботи з математичними функціями	05.11-23.11.25	виконано
6	Створення адаптивних версій додатку на смартфоні та планшеті	24.11-02.12.25	виконано
7	Оформлення роботи: вступ, висновки, реферат	02.12-08.12.25	виконано
8	Розробка демонстраційних матеріалів	09.12-13.12.25	виконано

Здобувачка вищої освіти

(підпис)

Максим МАРОХОНЬКО

(Ім'я, ПРІЗВИЩЕ)

Керівник

кваліфікаційної роботи

(підпис)

Олена ШИКУЛА

(Ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 80 стор., 4 табл., 18 рис., 98 джерела.

Наукове завдання – огляд існуючих систем мережесховищ NAS, аналіз їх функціональних можливостей, архітектури та безпеки, виявлення їх переваг та недоліків; розробка веб-додатку для управління файлами, їх зберігання, синхронізації та резервного копіювання з підтримкою англійської та української мов; створення адаптивних версій додатку для комп'ютерів, смартфонів та планшетів.

Мета роботи – розробка веб-додатку для мережевого сховища NAS з широким функціоналом та зручним інтерфейсом, призначеного для ефективного управління файлами, їх зберігання та доступу, доступного англійською та українською мовами.

Об'єкт дослідження – процес створення веб-додатку для мережевого сховища NAS.

Предмет дослідження – веб-додаток для управління файлами, їх зберігання, синхронізації та резервного копіювання в системі NAS.

Короткий зміст роботи:

Проведено огляд існуючих систем мережесховищ NAS, здійснено аналіз їх функціоналу, архітектури, безпеки та продуктивності, виявлено їх переваги та недоліки. Розроблено веб-додаток на основі React.js, Node.js та MongoDB для управління файлами, їх зберігання, синхронізації та резервного копіювання з широким функціоналом та зручним інтерфейсом. Розроблено адаптивні версії додатку, оптимізовані для використання на комп'ютерах, смартфонах та планшетах.

Ключові слова: мережеве сховище NAS, веб-додаток, React.js, Node.js

ABSTRACT

The textual part of the Master's thesis comprises 80 pages, 4 tables, 18 figures and 98 references.

Research objective: To review existing Network Attached Storage (NAS) systems and analyse their functionality, architecture and security to identify their respective advantages and disadvantages. To develop a web application that supports English and Ukrainian and provides file management, storage, synchronisation and backup. To create adaptive versions of the application for computers, smartphones and tablets.

Purpose of the study: To develop a web application for a NAS system with comprehensive functionality and a user-friendly interface designed for efficient file management and access. The application will be available in English and Ukrainian.

Object of study: The process of developing a web application for a NAS system.

Subject of study: A web application for file management, storage, synchronisation and backup in a NAS system.

Summary of the work: An analysis of existing NAS systems was conducted to evaluate their functionality, architecture, security and performance, and to identify their strengths and weaknesses. A web application based on React.js, Node.js and MongoDB was then developed to manage, store, synchronise and back up files. This application features extensive functionality and a user-friendly interface. Adaptive versions of the application were also developed and optimised for use on computers, smartphones and tablets.

Keywords: Network Attached Storage (NAS), web application, React.js

ЗМІСТ

ВСТУП.....	11
1 ОГЛЯД ТА АНАЛІЗ ТЕМАТИЧНОЇ ОБЛАСТІ.....	14
1.1 Мережеві сховища NAS	14
1.1.1 Класифікація систем мережевих сховищ NAS	14
1.1.2 Типова структура системи мережевого сховища NAS	16
1.1.3 Історія розвитку мережевих сховищ NAS.....	19
1.1.4 Функціонал систем мережевих сховищ NAS.....	22
1.2 Огляд та аналіз існуючих веб-систем для управління NAS	25
1.2.1 Комерційні NAS-рішення	25
1.2.2 Відкриті (Open Source) NAS-рішення TrueNAS (колишній FreeNAS).....	26
1.2.3 Порівняльний аналіз існуючих рішень.....	29
1.2.4 Виявлені проблеми та обмеження існуючих рішень	31
1.3 Постановка задачі.....	35
1.3.1 Обґрунтування необхідності розробки.....	35
1.3.2 Формулювання задачі.....	35
1.3.3 Функціональні вимоги.....	38
1.3.4 Нефункціональні вимоги.....	39
1.3.5 Обмеження та припущення.....	40
1.3.6 Критерії успішності проекту	40
2. АНАЛІЗ ТА ШЛЯХИ СТВОРЕННЯ ДОДАТКУ	42
2.1 Архітектура веб-додатку	42
2.1.1 Вибір архітектурного патерну	42
2.1.2 Проектування клієнт-серверної архітектури.....	43
2.2 Розробка серверної частини (Backend)	44
2.2.1 Структура backend додатку.....	44
2.2.2 Моделі даних та схеми бази даних	46
2.2.3 API endpoints та маршрутизація	48
2.2.4 WebSocket real-time комунікація	50
2.3 Розробка клієнтської частини (Frontend).....	51
2.3.1 Структура frontend додатку.....	51

2.3.2	Управління станом застосунку	54
2.3.3	Інтеграція з API та маршрутизація.....	56
2.3.4	UI/UX дизайн та стилізація	59
2.4	Вибір технологій та середовища програмування	61
3.	РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ	65
3.1	Архітектура системи.....	65
3.1.1	Загальна структура.....	65
3.1.2	Модель даних	66
3.1.3	API та маршрутизація	69
3.2	Реалізація основного функціоналу	69
3.2.1	Автентифікація та авторизація	69
3.2.2	Файловий менеджер.....	71
3.2.3	Завантаження файлів	72
3.2.4	Права доступу та спільне використання.....	74
3.2.5	Гостьовий доступ	75
3.2.6	Пошук та фільтрація	76
3.2.7	Управління користувачами (Admin)	77
3.3	Адаптивний дизайн	78
3.3.1	Responsive breakpoints	78
3.3.2	Мобільна версія.....	78
3.3.3	Версія для планшету	79
3.3.4	Десктопна версія	79
3.4	Real-time оновлення	80
3.5	Безпека.....	81
3.6	Тестування	82
3.6.1	Типи тестування	82
3.6.2	Performance тестування	83
3.7	Висновки до розділу 3	84
	ВИСНОВКИ.....	86
	ПЕРЕЛІК ПОСИЛАНЬ	91
	ДОДАТОК А. UML-ДІАГРАМИ СИСТЕМИ	98
	ДОДАТОК Б. ФРАГМЕНТИ ПРОГРАМНОГО КОДУ	103

ДОДАТОК В. ІНСТРУКЦІЯ З ВСТАНОВЛЕННЯ	121
ДОДАТОК Г. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ).....	128

ВСТУП

В епоху цифрової трансформації обсяг даних, що генерується користувачами та організаціями, зростає експоненціально. Мережеві сховища NAS (Network Attached Storage) стають все більш популярними рішеннями для централізованого зберігання та спільного доступу до файлів. Мета таких систем – забезпечити користувачам надійне, безпечне та зручне середовище для управління даними, розвиток навичок використання сучасних технологій і застосування ефективних інструментів організації файлового простору на практиці. Вони є ефективним засобом для організації корпоративних сховищ, а також чудово підходять для домашнього використання, даючи можливість глибше контролювати свої дані через веб-інтерфейс.

Крім наявності багатьох готових NAS-рішень від провідних виробників (Synology, QNAP, Western Digital), існує обмаль доступних веб-додатків з відкритим вихідним кодом, які пропонують можливості для гнучкого управління власним мережевим сховищем. Комерційні рішення користуються значним попитом, але їхня вартість часто висока, а функціонал не завжди відповідає специфічним потребам користувачів. Тому варто відзначити можливість створення власних веб-систем, які усувають ці недоліки та надають користувачам більш економічний і адаптований інструмент для роботи з файловими сховищами.

Системам мережевого зберігання даних присвячено значний обсяг літератури. Більшість із них є описами комерційних продуктів та технічною документацією [2, 7, 8, 10, 16, 17]. Деякі джерела мають практичну спрямованість щодо розгортання NAS-серверів [11, 17, 19]. Водночас літератури про розробку власних веб-інтерфейсів для NAS значно менше [1, 6, 13, 14]. Серед вітчизняних досліджень [12, 13, 14, 15, 18] україномовні ресурси зосереджені переважно на питаннях адміністрування Linux-систем [13] і програмування веб-додатків [14, 15].

Огляд літератури дозволив зробити такі висновки. Сучасний ринок пропонує готові NAS-рішення для зберігання файлів, але веб-додатків з відкритим кодом, які мають зручний, сучасний інтерфейс та гнучку систему управління правами

доступу, досі недостатньо. Аналізуючи наявні інструменти, можна помітити кілька значних недоліків: висока вартість комерційних рішень, обмежені можливості кастомізації, відсутність підтримки гостьового доступу з обмеженим терміном дії, а також складність інтеграції з власною інфраструктурою.

Розробка веб-додатку на базі сучасних технологій (React.js, Node.js, MongoDB), що поєднує широкий вибір інструментів для управління файлами та їх спільного використання, додатку з інтуїтивним інтерфейсом, доступного через будь-який веб-браузер, є актуальною і стане кроком для вирішення цієї проблеми. Це дозволить створити універсальний інструмент для домашніх користувачів, малих організацій і розробників, які потребують надійного та економічного рішення для управління власним мережевим сховищем.

Об'єкт дослідження – процес створення веб-додатку для управління мережевим сховищем.

Предмет дослідження – веб-додаток для управління файловою системою NAS з розмежуванням прав доступу.

Метою роботи є розробка веб-додатку з широким функціоналом, призначеного для управління файловою системою мережевого сховища NAS. Такий додаток спрямований на забезпечення користувачів зручними інструментами для роботи з файлами, організації спільного доступу та контролю прав користувачів.

Наукові завдання:

- Зробити огляд існуючих систем управління NAS та виявити їх переваги та недоліки;
- Розробити архітектуру веб-додатку, яка забезпечує ефективну взаємодію клієнтської та серверної частин;
- Реалізувати серверну частину на базі Node.js з використанням бази даних MongoDB для зберігання метаданих файлів та користувачів;
- Створити REST API для управління файлами, папками та користувачами;
- Розробити систему автентифікації та авторизації з підтримкою різних ролей (Owner, Admin, User, Guest);

- Реалізувати систему розмежування прав доступу з можливістю спільного використання папок між користувачами;
- Створити клієнтську частину з використанням React.js та TypeScript з адаптивним інтерфейсом;
- Впровадити систему real-time оновлень через WebSocket для синхронізації змін між користувачами;
- Додаток повинен мати зручні інструменти для навігації, пошуку, завантаження та управління файлами;
- Забезпечити підтримку роботи на різних пристроях: комп'ютер, смартфон, планше

1 ОГЛЯД ТА АНАЛІЗ ТЕМАТИЧНОЇ ОБЛАСТІ

1.1 Мережеві сховища NAS

1.1.1 Класифікація систем мережевих сховищ NAS

Network Attached Storage (NAS) — це спеціалізоване обладнання або програмне рішення для зберігання даних, що підключається до мережі та надає доступ до файлів множині користувачів та клієнтських пристроїв. На відміну від прямого підключення (Direct Attached Storage, DAS), NAS працює як автономний мережевий вузол зі власною IP-адресою.

Класифікація за призначенням

За призначенням системи мережевих сховищ поділяються на три основні категорії. Домашні NAS (Home/Personal NAS) призначені для домашніх користувачів і забезпечують базовий функціонал зберігання медіафайлів та резервного копіювання. Офісні NAS (SOHO – Small Office/Home Office) розраховані на малі офіси та робочі групи з 5-20 користувачів, надаючи розширені можливості спільної роботи з документами. Корпоративні NAS (Enterprise NAS) створені для середніх та великих підприємств, де критично важливими є висока продуктивність, надійність та масштабованість системи.

Класифікація за архітектурою

За архітектурним підходом розрізняють апаратні, програмні та хмарні NAS-рішення. Апаратні NAS являють собою готові рішення від виробників зі спеціалізованим обладнанням та оптимізованим програмним забезпеченням, що забезпечує легкість встановлення та високу надійність. Програмні NAS – це операційні системи для розгортання на власному обладнанні, що надає повну гнучкість у виборі компонентів та дозволяє створити економічне рішення при самостійному збиранні. Хмарні NAS (Cloud NAS) представляють собою віртуальні сховища в хмарі з інтерфейсом традиційного NAS, які не потребують фізичного обладнання, але залежать від якості інтернет-з'єднання.

Класифікація за протоколами доступу

Протоколи мережевого доступу до файлів визначають спосіб взаємодії клієнтських пристроїв з NAS-сервером. Протокол NFS (Network File System), розроблений Sun Microsystems для Unix-систем, забезпечує прозорий доступ до файлів через мережу та широко використовується в Linux/Unix середовищах. Доступні версії протоколу включають NFSv3, NFSv4 та NFSv4.1, кожна з яких надає покращену функціональність та безпеку.

Протокол SMB/CIFS (Server Message Block / Common Internet File System), розроблений Microsoft для Windows, є стандартом для файлового доступу в Windows мережах і підтримується всіма сучасними операційними системами. Еволюція протоколу відбувалась від застарілої версії SMB 1.0 до сучасних SMB 2.0/2.1 та SMB 3.0/3.1, що забезпечують вищу швидкість та безпеку передачі даних.

Протокол AFP (Apple Filing Protocol) є власним протоколом Apple для macOS, оптимізованим для Mac-середовища, хоча поступово замінюється на універсальний SMB. Протоколи FTP/SFTP (File Transfer Protocol / SSH File Transfer Protocol) представляють класичний підхід до передачі файлів, де SFTP додає шифрування через SSH для безпечного віддаленого доступу. Протокол WebDAV (Web Distributed Authoring and Versioning) є розширенням HTTP для роботи з файлами через веб-браузер з підтримкою версіонування.

Класифікація за рівнем RAID

RAID (Redundant Array of Independent Disks) — технологія об'єднання дисків для підвищення надійності та/або продуктивності: зображена в таблиці 1.1.

Таблиця 1.1 – Класифікація RAID масивів

RAID	Мін. дисків	Втрата	Призначення
RAID 0	2	0%	Швидкість
RAID 1	2	50%	Надійність
RAID 5	3	1 диск	Баланс
RAID 6	4	2 диски	Висока надійність
RAID 10	4	50%	Швидкість+надійність

1.1.2 Типова структура системи мережевого сховища NAS

Типова архітектура NAS-системи складається з наступних компонентів:

1. Апаратний рівень (Hardware Layer)

Апаратна основа NAS-систем варіюється залежно від призначення. Процесори домашніх систем базуються на ARM-архітектурі, тоді як офісні та корпоративні рішення використовують Intel Celeron або Xeon процесори з кількістю ядер від 2 до 16 і більше. Обсяг оперативної пам'яті коливається від 512 MB для базових домашніх моделей до 256 GB для корпоративних серверів, використовуючи стандарти DDR3, DDR4 або DDR5.

Мережева підсистема включає від 1 до 4 Ethernet портів з підтримкою швидкостей 1, 2.5 або 10 Гігабіт, технології Link Aggregation для об'єднання каналів та опційного Wi-Fi підключення. Дискова підсистема може містити від 1 до 24 і більше відсіків для HDD/SSD дисків з інтерфейсами SATA III або NVMe, підтримкою гарячої заміни (hot-swap) та можливістю використання SSD кешування. Додаткові інтерфейси включають порти USB 2.0/3.0/3.1, eSATA для розширення, HDMI для мультимедіа та картрідери SD/microSD.

2. Системний рівень (System Layer)

Операційні системи NAS базуються на Linux ядрі зі спеціалізованими оболонками (DSM, QTS, TrueNAS), оптимізованими для файлових операцій. Вони надають веб-інтерфейс управління та підтримують автоматичні оновлення безпеки. Файлові системи відіграють критичну роль у забезпеченні надійності даних: ext4 є стандартом Linux, Btrfs підтримує снапшоти та стиснення даних з checksumming, ZFS забезпечує максимальну надійність через самовідновлення, а XFS оптимізована для роботи з великими файлами. Системи управління томами, такі як LVM (Logical Volume Manager), RAID контролери та технологія тонких томів (Thin Provisioning), забезпечують гнучке управління дисковим простором.

3. Мережевий рівень (Network Layer)

Протоколи обміну даними Структура: багаторівнева архітектура (Application → Transport → Network → Data Link).

Мережеві сервіси:

- DHCP клієнт (автоматичне отримання IP)
- DNS клієнт (перетворення імен у IP)
- DDNS (динамічний DNS для віддаленого доступу)
- VPN сервер (OpenVPN, PPTP, L2TP/IPsec)

4. Рівень зберігання даних (Storage Layer)

Організація файлової структури Структура: /volume1/ → homes/ (користувачі), shared/ (спільні папки), @database/, @eaDir/, @tmp/

Механізми захисту даних:

- Snapshots (моментальні знімки)

5. Рівень додатків (Application Layer)

Системні сервіси:

- Веб-сервер (Apache, Nginx)
- База даних (MariaDB, PostgreSQL)
- PHP, Python runtime
- Docker підтримка

Користувацькі додатки:

- Файловий менеджер

6. Рівень безпеки (Security Layer)

Автентифікація та авторизація:

- Локальні користувачі та групи
- Інтеграція з Active Directory/LDAP
- Двофакторна автентифікація (2FA)
- SSO (Single Sign-On)

Контроль доступу:

- ACL (Access Control Lists) – списки контролю доступу
- Права доступу: Read, Write, Execute
- Квоти на користувачів
- IP-фільтрація

Шифрування:

- Шифрування томів (AES-256)
- SSL/TLS для веб-інтерфейсу
- Шифрування мережевого трафіку

Журналювання:

- Логи доступу до файлів
- Логи системних подій
- Логи безпеки (спроби входу)
- SIEM інтеграція

7. Інтерфейс користувача (User Interface Layer)

Веб-інтерфейс:

- Адаптивний дизайн (Desktop, Tablet, Mobile)
- Файловий менеджер з drag-and-drop
- Dashboard з моніторингом системи
- Налаштування системи

Мобільні додатки:

- iOS та Android клієнти
- Автоматичне резервне копіювання фото
- Потокowe відтворення медіа
- Віддалений доступ до файлів

Десктопні клієнти

- Синхронізація файлів (аналог Dropbox)
- Віртуальні диски (network drive mapping)
- Резервне копіювання

Схема взаємодії компонентів

Структура: багаторівнева архітектура (Application → Transport → Network → Data Link).

1.1.3 Історія розвитку мережевих сховищ NAS

Зародження концепції (1980-ті роки)

Початок розвитку мережевих сховищ було покладено у 1983 році з появою Novell NetWare – першої комерційної мережевої операційної системи для спільного доступу до файлів. Система використовувала виділені файлові сервери та протокол IPX/SPX для обміну даними, запропонувавши революційну на той час систему прав доступу.

У 1984 році компанія Sun Microsystems розробила протокол Network File System (NFS), що забезпечував прозорий доступ до віддалених файлів у Unix-системах. Цей відкритий стандарт став основою для багатьох майбутніх рішень і еволюціонував через версії NFS 2.0 (1989), NFS 3.0 (1995) та NFS 4.0 (2000).

Паралельно у 1985 році Microsoft розробила протокол Server Message Block (SMB), який було інтегровано спочатку в MS-DOS, а згодом і в Windows. Цей протокол здобув домінування в корпоративному секторі Windows та пройшов еволюцію від SMB 1.0 через CIFS до сучасних версій SMB 2.0/3.0.

Становлення ринку NAS (1990-ті роки)

У 1990 році компанія Auspex Systems випустила першу спеціалізовану NAS-систему, реалізувавши концепцію відокремлення функцій зберігання даних від обчислень. Це був перший storage appliance – готове рішення для зберігання файлів.

Заснування у 1992 році компанії Network Appliance (NetApp) стало поворотним моментом для індустрії. Фокусуючись на високопродуктивних enterprise рішеннях, NetApp розробила власну файлову систему WAFL (Write Anywhere File Layout) та спрощений підхід до налаштування і управління, що зробило компанію лідером ринку.

У 1996 році гігант індустрії зберігання EMC вийшов на ринок NAS з продуктом Celerra, забезпечивши високий рівень інтеграції з SAN (Storage Area Network) та масштабовані рішення для великих підприємств. Наприкінці 1990-х почалася поява Linux-based NAS систем, які завдяки відкритому програмному забезпеченню знизили вартість рішень та забезпечили гнучкість кастомізації.

Популяризація та диверсифікація (2000-ні роки)

Заснована у 2000 році на Тайвані компанія Synology Inc. зосередилась на сегменті малих і середніх підприємств, запропонувавши інноваційний веб-інтерфейс DiskStation Manager (DSM) з моделлю постійних безкоштовних оновлень. У 2004 році на ринок вийшов ще один тайванський гравець – QNAP Systems з операційною системою QTS (QNAP Turbo Station), зробивши акцент на апаратній продуктивності та багатофункціональності, включаючи мультимедіа і віртуалізацію.

Випуск у 2005 році FreeNAS – першої версії відкритого NAS на базі FreeBSD з використанням файлової системи ZFS – створив безкоштовну альтернативу комерційним рішенням, підтриману активно спільнотою розробників. У 2007 році виробники жорстких дисків Western Digital і Seagate вийшли на ринок готових NAS для домашніх користувачів з продуктами WD My Book World та Seagate BlackArmor, інтегруючи їх з власними дисками.

Компанія Drobo у 2009 році представила інновацію BeyondRAID – автоматичний RAID з plug-and-play підходом для недосвідчених користувачів та можливістю використання дисків різного обсягу.

Ера хмарної інтеграції (2010-ті роки)

2010-2012 – Хмарна інтеграція:

- Синхронізація з Dropbox, Google Drive, OneDrive
- Гібридні рішення (локальне + хмарне зберігання)
- Віддалений доступ через інтернет без VPN
- Автоматичне резервне копіювання в хмару

2013 рік – Мобільна революція:

- Випуск мобільних додатків (iOS, Android)
- Автоматичне резервне копіювання фото зі смартфонів
- Потокowe відтворення медіа на мобільних пристроях
- Push-нотифікації про події

2014 рік – Docker та віртуалізація:

- Підтримка контейнерів Docker на NAS
- Можливість запуску додатків (Plex, Home Assistant, GitLab)
- Перетворення NAS на багатофункціональний сервер

2016 рік – 10 Gigabit Ethernet:

- Впровадження 10GbE в офісні та домашні NAS
- Різке зростання швидкості передачі даних
- Робота з 4K/8K відео без затримок

2017-2019 – Штучний інтелект:

- Розпізнавання обличчя на фото
- Автоматична категоризація медіафайлів
- Інтеграція голосових асистентів (Alexa, Google Assistant)

Сучасний етап (2020-ті роки)

Пандемія COVID-19 у 2020 році спричинила різкий ріст попиту на домашні NAS через масовий перехід на віддалену роботу з дому та необхідність власних сховищ для відеоконференцій, що призвело до відмови багатьох компаній від офісних серверів.

У 2021 році відбулось масове впровадження M.2 NVMe SSD-кешу для прискорення роботи з дрібними файлами та підтримки PCIe 4.0. Наступний 2022 рік ознаменувався стандартизацією 2.5 Gigabit Ethernet як середнього рівня між 1GbE та 10GbE, що забезпечило оптимальний баланс ціни та продуктивності з широкою підтримкою в маршрутизаторах.

У 2023 році з'явилися AI-функції нового покоління з локальною обробкою даних штучним інтелектом, що забезпечило приватність даних без передачі в хмару, включаючи розпізнавання об'єктів на відео та автоматичне створення відеокліпів. Квантовий стрибок 2024 року включив підтримку Wi-Fi 7 для бездротового доступу, 8K відео, нові рівні енергоефективності та інтеграцію з IoT екосистемою розумного дому.

Еволюція обсягів зберігання та основні тенденції розвитку

Еволюція обсягів зберігання демонструє вражаючий прогрес: від 80GB HDD у 2000 році до 2TB у 2010, 18TB у 2020 та 24TB у 2024 році для споживчого сегменту. Корпоративні системи пройшли шлях від 2TB у 2000 році до понад 5 петабайт у 2024 році.

Основні тенденції розвитку індустрії включають зростання швидкості мережевого доступу від 100Mbps до понад 10Gbps, збільшення ємності від гігабайт до петабайт, постійне зниження вартості зберігання у перерахунку на гігабайт, спрощення користувацького інтерфейсу, глибоку інтеграцію з хмарними сервісами, мобільний доступ з будь-яких пристроїв та впровадження архітектури zero-trust security.

1.1.4 Функціонал систем мережевих сховищ NAS

Сучасні NAS-системи пропонують широкий спектр функцій, що виходить далеко за межі простого зберігання файлів. Розглянемо основні категорії функціоналу.

1. Управління файлами та папками

Базові функції включають CRUD операції (створення, читання, оновлення, видалення), drag-and-drop завантаження, множинне завантаження файлів, пошук, попередній перегляд та повноцінний файловий менеджер. Додаткові можливості охоплюють версіювання файлів, систему тегів, кошик для відновлення видалених файлів, дедуплікацію для економії місця та стиснення даних.

2. Система контролю доступу

Система управління користувачами підтримує локальні облікові записи та групи, інтеграцію з LDAP/Active Directory, гостьові акаунти та Single Sign-On. Управління правами реалізується через RBAC (Role-Based Access Control), ACL (Access Control Lists), квоти дискового простору та успадкування прав. Безпека забезпечується шифруванням AES-256, двофакторною автентифікацією, IP whitelisting та захистом від брутфорс атак.

3. Спільний доступ та синхронізація

Функції обміну даними включають спільні, групові та приватні папки, створення публічних посилань з можливістю встановлення пароля та терміну дії. Синхронізація реалізується через десктопні клієнти з підтримкою двосторонньої синхронізації, вибіркової синхронізації папок та автоматичного вирішення конфліктів.

4. Мультимедіа функції

Медіа-сервер підтримує протоколи DLNA/UPnP, інтеграцію з Plex та Emby, транскодування відео на льоту та управління субтитрами. Фото-функції включають розпізнавання обличь, геотегування, створення альбомів та слайдшоу. Музичний сервер забезпечує автоматичне тегування, завантаження обкладинок, управління плейлистами та підтримку iTunes Server.

5. Резервне копіювання

Джерела резервних копій включають персональні комп'ютери (Windows/Mac/Linux), смартфони, віртуальні машини та інші NAS-системи. Методи резервного копіювання охоплюють повне, інкрементальне та диференціальне копіювання, snapshots та версіонування файлів. Призначення для збереження копій включає локальні диски, USB накопичувачі, віддалені NAS-сервери, хмарні сховища (Amazon S3, Azure, Google Cloud) та FTP-сервери. Планування включає налаштування розкладу, ротацію старих копій та перевірку цілісності даних.

6. Віддалений доступ

Доступ до NAS з інтернету реалізується через QuickConnect або myQNAPcloud сервіси, DDNS для статичного доменного імені, VPN з'єднання (OpenVPN/L2TP/PPTP), Reverse Proxy та протокол WebDAV. Мобільні додатки включають DS file/Qfile для управління файлами, DS photo/Qphoto для фотогалереї, DS video/Qvideo для перегляду відео та DS audio/Qmusic для прослуховування музики.

7. Додаткові сервіси

Додаткові сервіси включають підтримку контейнерів Docker та віртуальних машин, власний поштовий сервер, системи управління контентом (CMS), Git-репозиторії, менеджер завантажень Download Station з підтримкою torrent, eDonkey та NZB, систему відеоспостереження для IP-камер та IoT Hub для управління пристроями розумного дому.

8. Моніторинг та управління

Система моніторингу включає Resource Monitor для відстеження використання CPU, RAM та дискового простору, S.M.A.R.T. моніторинг здоров'я дисків, контроль температури, систему сповіщень через email, SMS або push-повідомлення та детальне журналювання подій. Управління системою здійснюється через веб-інтерфейс, мобільні додатки, командний рядок CLI/SSH, REST API для інтеграції з іншими системами, планувальник задач та систему автоматичних оновлень. Порівняння функціоналу сучасних NAS систем зображено на рисунку 1.1.

Функція	Synology DSM	QNAP QTS	TrueNAS	OpenMediaVault	Windows Storage
Веб-інтерфейс	★★★★★	★★★★★	★★★★★	★★★★★	★★★★★
Файловий менеджер	★★★★★	★★★★★	★★★★★	★★★★	★★★★★
Мобільні додатки	★★★★★	★★★★★	★★★★★	★★★★	★★★★★
Мультимедіа	★★★★★	★★★★★	★★★★	★★★★★	★★★★★
Docker	✓	✓	✓	✓	✗
Віртуалізація	✓	✓	✓ (Bhyve)	✗	✓ (Hyper-V)
Snapshots	✓ (Btrfs)	✓ (Btrfs)	✓ (ZFS)	⚠ (LVM)	✓ (ReFS)
Квоти	✓	✓	✓	✓	✓
Active Directory	✓	✓	✓	✓	✓
S3 сумісність	✓	✓	✓ (MinIO)	⚠ (плагін)	✗

Рисунок 1.1 – Порівняння функціоналу популярних NAS систем

1.2 Огляд та аналіз існуючих веб-систем для управління NAS

1.2.1 Комерційні NAS-рішення

Synology DiskStation Manager (DSM)

Лідер ринку: інтуїтивний веб-інтерфейс, Package Center з 100+ додатками, Btrfs, Synology Drive (синхронізація), Surveillance Station (відеоспостереження), Docker, регулярні оновлення.

QNAP Turbo Station (QTS) / QuTS hero

Загальна характеристика: QTS — операційна система від тайванської компанії QNAP, що конкурує з Synology. Відома більш потужним залізом та акцентом на мультимедіа і віртуалізацію.

Ключові особливості:

1. QTS інтерфейс:

- Схожий на Synology підхід (desktop metaphor)
- Більш технічний та деталізований
- Qboost для оптимізації продуктивності
- Resource Monitor в реальному часі

2. Hybrid Backup Sync:

- Унікальна 3-2-1 стратегія резервування
- Інтеграція з 20+ хмарними сервісами
- Snapshots реплікація
- Active Directory backup

3. Multimedia Station:

- Потужний транскодинг відео (апаратне прискорення)
- Підтримка 4K та HEVC
- Chromecast, AirPlay
- OceanKTV (караоке система)

4. Virtualization Station:

- Запуск віртуальних машин (Windows, Linux, Android)
- Snapshot VM

- USB passthrough
- Інтеграція з VMware, Hyper-V

5. Container Station:

- Docker та LXC контейнери
- Docker Hub інтеграція
- Kubernetes підтримка (високопродуктивні моделі)
- Простий GUI для управління контейнерами

Високопродуктивна платформа: QuTS hero на ZFS, QTS на ext4, віртуалізація (VM Station), Container Station (Docker/LXC), App Center, Hybrid Backup Sync.

Western Digital My Cloud OS

Споживчий сегмент: спрощений інтерфейс для домашніх користувачів, автобекап, медіасервер Plex, мобільні додатки, обмежені можливості кастомізації.

1.2.2 Відкриті (Open Source) NAS-рішення TrueNAS (колишній FreeNAS)

Загальна характеристика: Найпопулярніше безкоштовне NAS-рішення на базі FreeBSD з файловою системою ZFS. Орієнтоване на максимальну надійність та цілісність даних.

Технічний стек:

- OS: FreeBSD (TrueNAS CORE), Linux (TrueNAS SCALE)

Ключові особливості:

1. ZFS File System:

- Copy-on-Write (CoW)

2. Веб-інтерфейс:

- Сучасний Material Design
- Детальний dashboard з метриками
- Shell in browser
- Конфігурація через GUI та CLI

3. TrueNAS SCALE (новіша версія):

- На базі Linux замість FreeBSD

- Kubernetes для контейнерів
- Gluster для кластеризації
- TrueCharts – репозиторій додатків (Helm charts)

4. Replication:

- Snapshot-based replication
- Інкрементна передача змін
- Віддалена реплікація між TrueNAS серверами

5. Plugins:

- FreeBSD jails (CORE) або Docker/Kubernetes (SCALE)
- Nextcloud, Plex, Transmission, GitLab
- Grafana для моніторингу

Архітектура TrueNAS: Структура: багаторівнева архітектура (Application → Transport → Network → Data Link).

Enterprise ZFS рішення: максимальна надійність, snapshots, replication, TrueNAS SCALE (Kubernetes), потужний але складний в налаштуванні, open-source.

OpenMediaVault (OMV)

Загальна характеристика: Легка система на базі Debian Linux, призначена для домашніх серверів та малих офісів. Наступник FreeNAS (після розколу проекту).

Технічний стек:

- OS: Debian Linux

Ключові особливості:

1. Модульна архітектура:

- Базова система мінімальна
- Розширення через плагіни
- OMV-Extras репозиторій

2. Веб-інтерфейс:

- Класичний серверний стиль
- Всі налаштування через GUI
- Auto-configuration для служб

3. Підтримувані сервіси:

- SMB/CIFS, NFS, FTP, SFTP
- rsync jobs
- Cron планувальник
- SMART моніторинг дисків

4. Плагіни:

- Docker (через omv-extras)
- Portainer для управління контейнерами
- Backup (резервування)
- Downloader (Transmission, NZBGet)

Debian-based open-source: веб-інтерфейс, плагіни, Docker, OMV-Extras, легкий і гнучкий, безкоштовний, ідеальний для DIY проектів.

Unraid

Гнучка система масивів: паритет без RAID, різні розміри дисків, докеризація, плагіни, платна ліцензія (від \$59), популярний для медіасерверів.

Nextcloud (як файлове сховище)

Загальна характеристика: Хоча Nextcloud не є класичним NAS, він часто використовується як self-hosted хмарне сховище з веб-інтерфейсом.

Технічний стек:

- Backend: PHP (Symfony framework)

Ключові особливості:

1. Інтерфейс:

- Веб-інтерфейс схожий на Dropbox/Google Drive
- Drag-and-drop
- Попередній перегляд файлів
- Вбудований редактор документів (OnlyOffice, Collabora)

2. Синхронізація:

- Клієнти для Windows, Mac, Linux
- Мобільні додатки (iOS, Android)

- Двостороння синхронізація
- Вибіркова синхронізація папок

3. Співпраця:

- Спільне редагування документів в реальному часі
- Коментарі до файлів
- Календар, контакти
- Talk (відео/аудіо дзвінки, чат)

4. Розширюваність:

- 200+ додатків у магазині
- Mail, Calendar, Contacts
- Forms, Polls
- News reader, Bookmarks

Cloud-first платформа: синхронізація файлів, онлайн редагування (Collabora/ONLYOFFICE), календар, контакти, відеодзвінки, 300+ додатків, open-source.

1.2.3 Порівняльний аналіз існуючих рішень

Аналіз технологічних рішень

1. Архітектурні підходи:

Монолітна архітектура (Synology, QNAP):

Недоліки:

- Складніше оновлення окремих частин
- Залежність від вендора
- Закритий код

Модульна архітектура (TrueNAS, OMV):

Недоліки:

- Потребує більше технічних знань
- Можливі конфлікти між модулями
- Менш однорідний UX

2. Підходи до управління файлами:

Веб-файловий менеджер (більшість NAS):

- Повноцінний GUI в браузері
- Підтримка drag-and-drop
- Контекстні меню
- Швидкий перегляд файлів

API-first підхід (сучасні рішення):

- RESTful API для всіх операцій
- Можливість створення власних клієнтів
- Автоматизація через скрипти
- Інтеграція з іншими сервісами

3. Системи автентифікації:

Порівняння NAS-систем (зображено на рисунку 1.2): DSM (найкраща UX), QTS (performance), TrueNAS (надійність), OMV/Nextcloud (open-source), Unraid (гнучкість).

4. Підходи до резервного копіювання:

Snapshot-based (Btrfs, ZFS):

- Миттєві знімки стану файлової системи
- Мінімальне використання місця (CoW)
- Швидке відновлення
- Захист на рівні файлової системи

File-based (rsync, duplicity):

- Копіювання файлів на інший носій
- Більша гнучкість призначення
- Можливість інкрементного backup
- Простіше для розуміння

Block-based (dd, imaging):

- Копія на рівні блоків диска
- Повне відновлення системи

- Більший обсяг резервних копій
- Повільніше

Критерій	Synology DSM	QNAP QTS	TrueNAS	OpenMediaVault	Unraid	Nextcloud
Вартість	Висока	Висока	Безкоштовно	Безкоштовно	Низька	Безкоштовно
Складність	★★	★★★★	★★★★★	★★★★	★★	★★★★
Інтерфейс	10/10	8/10	7/10	6/10	7/10	9/10
Продуктивність	8/10	9/10	10/10	7/10	7/10	6/10
Надійність	9/10	8/10	10/10	8/10	8/10	7/10
Мультимедіа	10/10	10/10	5/10	5/10	9/10	6/10
Мобільні додатки	10/10	8/10	3/10	2/10	6/10	9/10
Docker/VM	✓	✓✓	✓	⚠	✓✓✓	✗
Безпека	9/10	7/10	10/10	7/10	8/10	8/10
Спільнота	★★★★★	★★★★★	★★★★★	★★★★★	★★★★★	★★★★★
Підходить для	Усі	Ентузіасти	ІТ-про	DIY	Медіа	Співпраця

Рисунок 1.2 – Порівняння NAS-систем

1.2.4 Виявлені проблеми та обмеження існуючих рішень

На основі аналізу існуючих систем можна виділити наступні проблеми:

1. Проблеми комерційних рішень

Висока вартість:

- Synology DS920+ (4-bay): \$550
- QNAP TS-453D (4-bay): \$480
- Необхідність купувати диски окремо (додатково \$200-800)
- Платні ліцензії на деякі функції (камери відеоспостереження, бізнес-backup)

Закритий код:

- Неможливість аудиту безпеки
- Залежність від виробника
- Обмежена кастомізація
- Ризик припинення підтримки

Прив'язка до обладнання:

- Робота лише на фірмових пристроях
- Неможливість апгрейду окремих компонентів

- Обмеженість в виборі конфігурації

2. Проблеми відкритих рішень

Складність налаштування:

- TrueNAS потребує глибоких знань ZFS
- OMV вимагає мануального налаштування багатьох сервісів
- Відсутність wizard для початкового setup
- Складна конфігурація RAID

Обмежений інтерфейс:

- Застарілий дизайн (особливо OMV)
- Відсутність зручного файлового менеджера
- Мінімальна адаптивність для мобільних
- Відсутність offline мобільних додатків

Документація:

- Часто застаріла або неповна
- Розкидана по форумах
- Мало українською/російською мовами
- Відсутність покрокових інструкцій

3. Загальні проблеми всіх рішень

Безпека:

- Вразливості до ransomware атак (особливо QNAP 2019-2021)
- Слабкі стандартні паролі
- Відкриті порти при віддаленому доступі
- Відсутність обов'язкової 2FA

Продуктивність:

- Обмежена пропускна здатність через 1 Gigabit Ethernet
- Недостатня потужність CPU для транскодингу 4K
- Повільна робота веб-інтерфейсу при великій кількості файлів
- Затримки при синхронізації великих обсягів даних

Мультитенантність:

- Складно організувати повну ізоляцію між користувачами
- Обмежені можливості делегування адміністрування
- Відсутність білінгу для комерційного використання

Мобільний досвід:

- Мобільні додатки часто поступаються веб-версії
- Обмежений offline доступ
- Проблеми з автоматичним резервним копіюванням фото
- Високе споживання батареї при синхронізації

4. Специфічні обмеження

Synology DSM:

- Обмеження кількості камер у Surveillance Station (безкоштовно 2 камери)
- Неможливість встановити на інше обладнання (є Xpenology, але unsupported)
- Повільне оновлення версій (DSM 7 вийшла через 5 років після DSM 6)

QNAP QTS:

- Історія вразливостей безпеки
- Складний та перевантажений інтерфейс
- Проблеми зі стабільністю після оновлень
- Агресивна реклама додаткових сервісів

TrueNAS:

- Високі вимоги до RAM (8-16 GB мінімум для ZFS)
- Складність для новачків
- Обмежена підтримка плагінів/додатків
- Проблеми з апаратною сумісністю

OpenMediaVault:

- Базовий функціонал із коробки
- Відсутність красивого файлового менеджера
- Складність налаштування advanced функцій
- Проблеми з плагінами після оновлень

Unraid:

- Платна модель (хоч і недорого)
- Нижча продуктивність порівняно з RAID
- Прив'язка ліцензії до USB флешки
- Можлива втрата даних при одночасному виході з ладу 3+ дисків

5. Відсутні можливості, які хотілося б мати

Сучасний веб-інтерфейс:

- Адаптивний дизайн на рівні Google Drive / Dropbox
- Темна тема всюди
- Швидкість роботи SPA додатків
- Drag-and-drop для всього

Розширена співпраця:

- Коментарі до файлів та папок
- @mention інших користувачів
- Активність в реальному часі (хто що робить)
- Інтеграція з месенджерами (Telegram, Slack)

Штучний інтелект:

- Автоматична категоризація файлів
- Розпізнавання дублікатів
- Рекомендації щодо організації
- Пошук за змістом (OCR в документах)

Гнучке управління правами:

- Детальні права на рівні окремих файлів
- Тимчасовий доступ з автоматичним відкликанням
- Аудит доступу (хто, коли, що)
- Workflow для затвердження доступу

Інтеграції:

- Webhook для подій
- API для автоматизації

- CLI для скриптів
- Інтеграція з CI/CD

1.3 Постановка задачі

1.3.1 Обґрунтування необхідності розробки

На основі проведеного аналізу існуючих рішень можна зробити висновок про необхідність створення власного веб-додатку для управління мережевим сховищем NAS, який би:

1. Усував недоліки комерційних рішень:

- Забезпечував безкоштовність використання без прихованих платежів
- Надавав повний доступ до вихідного коду для аудиту та кастомізації
- Дозволяв розгортання на будь-якому апаратному забезпеченні
- Не залежав від конкретного виробника

2. Спрощував використання відкритих рішень:

- Мав інтуїтивний, сучасний інтерфейс рівня Synology DSM
- Забезпечував простоту початкового налаштування
- Надавав зручний файловий менеджер з drag-and-drop
- Підтримував адаптивний дизайн для всіх пристроїв

3. Використовував сучасні веб-технології:

- React.js для створення швидкого та відзивчивого інтерфейсу

4. Надавав основний функціонал NAS:

- Управління файлами та папками через веб-інтерфейс

1.3.2 Формулювання задачі

Мета роботи: Розробити повнофункціональний веб-додаток для управління мережевим сховищем NAS з використанням React.js, Node.js та MongoDB, що забезпечує інтуїтивне управління файлами, гнучку систему контролю доступу та сучасний користувацький інтерфейс.

Основні завдання:

1. Проектування архітектури системи

Backend:

- Розробити RESTful API для всіх операцій
- Спроекувати структуру бази даних MongoDB
- Визначити модель даних для користувачів, файлів, папок та прав доступу
- Розробити систему middleware для обробки запитів

Frontend:

- Спроекувати компонентну структуру React додатку
- Визначити маршрутизацію додатку (React Router)
- Спланувати систему управління станом (Zustand)
- Розробити UI/UX дизайн інтерфейсу

2. Реалізація системи автентифікації та авторизації

Автентифікація:

- Реєстрація та вхід користувачів

Авторизація:

- Рольова модель доступу (Owner, Admin, User, Guest)
- Middleware для перевірки прав доступу
- Захист роутів на frontend та backend
- Система гостьових акаунтів з обмеженим часом дії

3. Розробка системи управління файлами

Файлові операції:

- Створення, читання, оновлення, видалення (CRUD) файлів

Управління папками:

- CRUD операції для папок

Додатковий функціонал:

- Soft delete (переміщення у кошик)

4. Реалізація системи прав доступу

Права на папки:

- Read (перегляд вмісту)
- Write (створення/редагування файлів)
- Delete (видалення файлів)
- Успадкування прав підпапками

Розмежування доступу:

- Власник папки (creator)

Спільний доступ (Sharing):

- Надання доступу іншим користувачам

5. Створення користувацького інтерфейсу

Головні компоненти:

- Dashboard (панель керування)

UX/UI елементи:

- Адаптивний дизайн (Desktop, Tablet, Mobile)

Навігація:

- Sidebar з розділами
- Breadcrumb для шляху
- Кнопки Back/Forward
- Швидкий доступ до папок (favorites)

6. Додаткові функції

Пошук:

- Повнотекстовий пошук файлів та папок

Управління користувачами (Admin):

- Перегляд списку користувачів

Гостьовий доступ:

- Створення тимчасових гостьових акаунтів

Логування та моніторинг:

- Winston logger для структурованих логів

- Категорії логів (auth, security, api, system)
- Ротація логів (daily rotation)
- Dashboard з системною інформацією

7. Синхронізація з файловою системою

File System Watcher:

- Моніторинг змін у файловій системі (Chokidar)

Ручна синхронізація:

- Скрипт для початкової синхронізації
- Пересканування папок при потребі
- Відновлення після збоїв

1.3.3 Функціональні вимоги

Вимоги до системи автентифікації:

- Система повинна дозволяти реєстрацію нових користувачів з email та паролем
- Паролі повинні зберігатися у зашифрованому вигляді (bcrypt)

Вимоги до управління файлами:

- Користувач повинен мати можливість завантажити один або декілька файлів
- Система повинна підтримувати завантаження папок зі збереженням структури

Вимоги до системи прав:

- Кожен користувач повинен мати свою домашню папку
- Власник папки повинен мати всі права

Вимоги до інтерфейсу:

- Інтерфейс повинен коректно відображатись на екранах від 320px до 4K
- Повинна бути підтримка темної та світлої теми

1.3.4 Нефункціональні вимоги

Продуктивність:

- Час відгуку на основні операції < 200ms
- Завантаження списку файлів (до 1000 файлів) < 1s
- Підтримка одночасної роботи 50+ користувачів
- Можливість завантаження файлів до 1GB
- Оптимізація завантаження великих списків (pagination, lazy loading)

Надійність:

- Доступність системи 99.5%
- Автоматичне відновлення з'єднання при розриві
- Збереження недовантажених файлів при перериванні
- Логування всіх критичних помилок
- Graceful degradation при недоступності сервісів

Безпека:

- Шифрування паролів (bcrypt з salt)
- Захист від SQL injection (використання Mongoose)

Масштабованість:

- Можливість горизонтального масштабування backend
- Підтримка CDN для статичних файлів
- Кешування часто запитуваних даних
- Оптимізація запитів до БД (індекси, агрегація)

Підтримуваність:

- Модульна архітектура коду
- Документування API (Swagger/OpenAPI)
- Коментарі в коді для складної логіки
- Використання TypeScript на frontend для типобезпеки
- ESLint для контролю якості коду

Сумісність:

- Підтримка браузерів: Chrome 90+, Firefox 88+, Safari 14+, Edge 90+

- Підтримка Node.js 18+
- Підтримка MongoDB 6+

1.3.5 Обмеження та припущення

Технічні обмеження:

- Файли зберігаються на файловій системі сервера (не в БД)
- MongoDB зберігає лише метадані файлів
- Відсутність транскодингу відео/аудіо
- Відсутність генерації мініатюр для зображень (може бути додано пізніше)
- Один сервер для backend та file storage (без distributed storage)

Функціональні обмеження:

- Відсутність версіонування файлів у початковій версії
- Відсутність end-to-end шифрування
- Відсутність офлайн режиму мобільного додатку
- Базовий пошук (без full-text search в документах)
- Відсутність інтеграції з хмарними сервісами

Припущення:

- Сервер має достатній обсяг дискового простору
- Мережа між клієнтом та сервером стабільна
- Користувачі мають сучасні браузері
- Адміністратор має базові знання Linux для розгортання
- Резервне копіювання виконується зовнішніми засобами

1.3.6 Критерії успішності проекту

Проект буде вважатися успішним, якщо:

Функціональність:

- Реалізовано всі базові операції з файлами (CRUD)

Якість інтерфейсу:

- Інтерфейс адаптивний для всіх пристроїв

Технічна реалізація:

- Код структурований та документований

Розгортання:

- Додаток можна розгорнути на сервері
- Є інструкція з встановлення
- Налаштування через змінні оточення
- Можливість запуску в Docker (опціонально)

2 АНАЛІЗ ТА ШЛЯХИ СТВОРЕННЯ ДОДАТКУ

2.1 Архітектура веб-додатку

2.1.1 Вибір архітектурного патерну

Для розробки веб-додатку управління мережевим сховищем NAS було обрано клієнт-серверну архітектуру з чітким розділенням на frontend та backend компоненти. Цей підхід забезпечує:

Переваги клієнт-серверної архітектури:

Розділення відповідальності (Separation of Concerns):

- Frontend відповідає за презентаційну логіку та взаємодію з користувачем
- Backend обробляє бізнес-логіку, доступ до даних та безпеку
- Чітке API між компонентами

Незалежність розробки:

- Команди можуть працювати паралельно над frontend та backend
- Можливість використання різних технологій для кожної частини
- Легше тестування окремих компонентів

Масштабованість:

- Можливість горизонтального масштабування backend
- Розміщення frontend на CDN для швидшого доступу
- Можливість створення різних клієнтів (web, mobile, desktop) для одного API

Безпека:

- Вся чутлива бізнес-логіка на сервері
- Клієнт не має прямого доступу до бази даних
- Централізована автентифікація та авторизація

Архітектурний патерн: MVC + REST API

Для backend застосовано модифікований патерн Model-View-Controller (MVC), адаптований для RESTful API. Архітектура складається з п'яти основних шарів.

Потік запиту: Client → Routes → Middlewares → Controllers → Services → Models → Database

2.1.2 Проектування клієнт-серверної архітектури

Взаємодія клієнта та сервера (зображено на рисунку 2.1)

Система використовує два протоколи для комунікації:

- HTTP/HTTPS (REST API) – для CRUD операцій (GET, POST, PUT/PATCH, DELETE)
- WebSocket (Socket.io) – для real-time оновлень (file_uploaded, file_deleted, folder_shared)

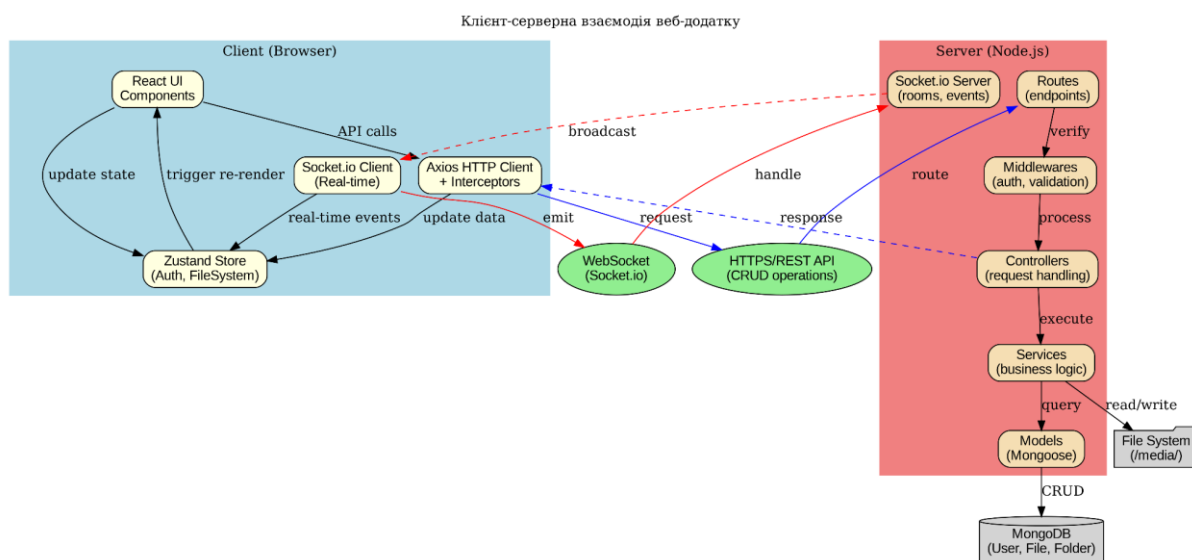


Рисунок 2.1 – Повний потік взаємодії між клієнтом та сервером

Формат обміну даними JSON:

Всі API відповіді використовують стандартизований формат: success response містить {success: true, data: {...}}, error response – {success: false, error: message message: details}. Для CRUD операцій використовуються стандартні HTTP методи: GET для читання, POST для створення, PATCH/PUT для оновлення, DELETE для видалення. WebSocket події передаються в JSON форматі з типом події та payload.

2.2 Розробка серверної частини (Backend)

2.2.1 Структура backend додатку

Серверна частина організована за модульним принципом з чітким розділенням відповідальності. Backend реалізовано на Node.js з використанням Express.js фреймворку. Структура додатку побудована за модифікованим патерном MVC (Model-View-Controller), адаптованим для RESTful API архітектури.

Детальна структура backend додатку представлена на рисунку 2.2, де показано всі основні директорії та модулі системи з їх взаємозв'язками.

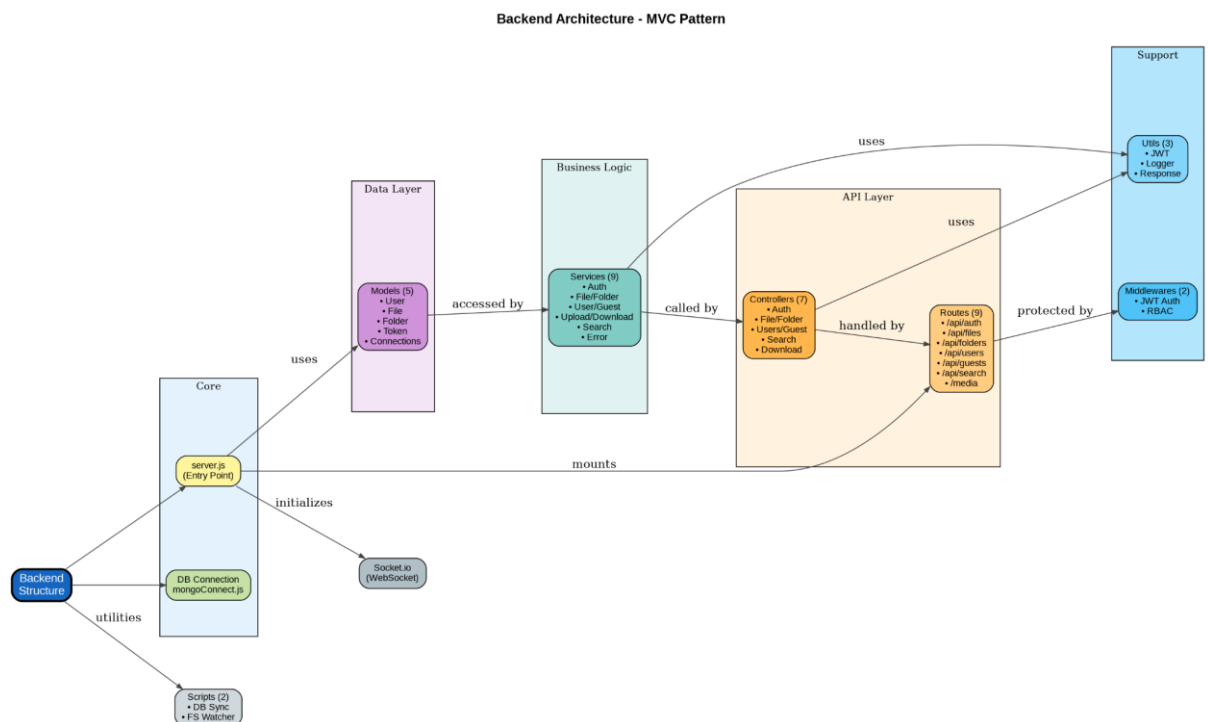


Рисунок 2.2 – Архітектура серверної частини додатку

Основний потік обробки запиту проходить через п'ять логічних шарів: Routes (маршрутизація) → Middlewares (проміжна обробка) → Controllers (контролери запитів) → Services (бізнес-логіка) → Models (моделі даних). Такий підхід забезпечує чітке розділення відповідальності та полегшує тестування окремих компонентів системи.

Кореневою точкою входу є файл `server.js`, який налаштовує Express сервер, підключає middleware, ініціалізує з'єднання з MongoDB через модуль `db/mongoConnect.js` та запускає WebSocket сервер `Socket.io` для real-time

комунікації. Всі статичні файли (завантажені користувачами) зберігаються в директорії `media/` та доступні через окремий маршрут.

Модульна структура включає вісім основних директорій. Models містять п'ять Mongoose схем: `user.model.js` для користувачів та гостьових акаунтів, `file.model.js` для метаданих файлів, `folder.model.js` для структури папок, `token.model.js` для refresh токенів JWT автентифікації та `connections.model.js` для відстеження активних з'єднань користувачів.

Директорія Controllers включає сім контролерів, що обробляють HTTP запити: `auth.controller.js` керує автентифікацією та реєстрацією, `file.controller.js` обробляє операції з файлами (CRUD, metadata), `folder.controller.js` керує структурою папок та правами доступу, `users.controller.js` відповідає за управління користувачами, `guest.controller.js` обслуговує гостьові акаунти, `search.controller.js` реалізує пошукову функціональність, а `download.controller.js` обробляє завантаження файлів з підтримкою streaming.

Шар Services містить дев'ять сервісів з бізнес-логікою: `auth.service.js` виконує верифікацію credentials та генерацію токенів, `file.service.js` керує файловою системою та синхронізацією з БД, `folder.service.js` обробляє деревовидну структуру папок та права доступу, `user.service.js` управляє користувацькими акаунтами, `guest.service.js` генерує access codes для гостей, `upload.service.js` обробляє multipart/form-data завантаження з валідацією типів та розмірів файлів, `download.service.js` реалізує потокову передачу файлів з підтримкою range requests, `search.service.js` виконує складні пошукові запити по name, mimeType та tags, а `error.service.js` забезпечує стандартизовану обробку помилок.

Routes визначають дев'ять груп API endpoints: `auth.routes.js (/api/auth/)` для login/register/logout/refresh, `file.routes.js (/api/files/)` для операцій з файлами, `folder.routes.js (/api/folders/)` для роботи з папками, `user.routes.js (/api/users/)` для управління користувачами, `guest.routes.js (/api/guests/)` для гостьових акаунтів, `search.routes.js (/api/search/)` для пошуку, `download.routes.js (/api/download/)` для завантаження файлів, `media.routes.js (/media/)` для статичних файлів та `connection.routes.js (/api/connections/*)` для відстеження активних сесій.

Директорія Middlewares містить дві критичні middleware: `protectedRoute.js` перевіряє наявність та валідність JWT токена з `Authorization header`, декодує `payload` та додає дані користувача в `req.user`, а `previliges.js` реалізує `role-based access control (RBAC)` з перевіркою ролей `admin/owner` та права доступу до конкретних ресурсів (файлів/папок).

Utils включають три утилітних модуля: `JWT.js` для генерації та валідації `Access (15 хв)` і `Refresh (30 днів)` токенів з використанням секретних ключів з `.env`, `logger.js` для централізованого логування через `Winston` з ротацією файлів та різними рівнями (`error`, `warn`, `info`, `debug`), і `responseFormatter.js` для стандартизованих API відповідей у форматі `{success, data, error, message}`.

Модуль Socket в файлі `index.js` налаштовує `Socket.io` з JWT автентифікацією та обробляє `real-time` події: `file_uploaded` (оновлення списку файлів), `file_deleted` (видалення файлу), `folder_shared` (надання доступу до папки), `user_connected/user_disconnected` (онлайн статус), дозволяючи миттєве оновлення інтерфейсу всім підключеним клієнтам.

Директорія Scripts містить два утилітних скрипта: `syncFilesToDB.js` виконує початкову синхронізацію файлової системи з `MongoDB` (сканує `media` директорію та створює відповідні `records` в БД), а `fsWatcherDaemon.js` реалізує фоновий процес моніторингу змін у файловій системі через `fs.watch` API з автоматичним оновленням БД при створенні, зміні або видаленні файлів поза межами веб-інтерфейсу.

2.2.2 Моделі даних та схеми бази даних

`MongoDB` обрано як основну базу даних завдяки гнучкості схем та природній підтримці деревовидних структур папок. Реалізовано п'ять основних колекцій з використанням `Mongoose ODM` для валідації та типізації даних, як зображено на рисунку 2.3.

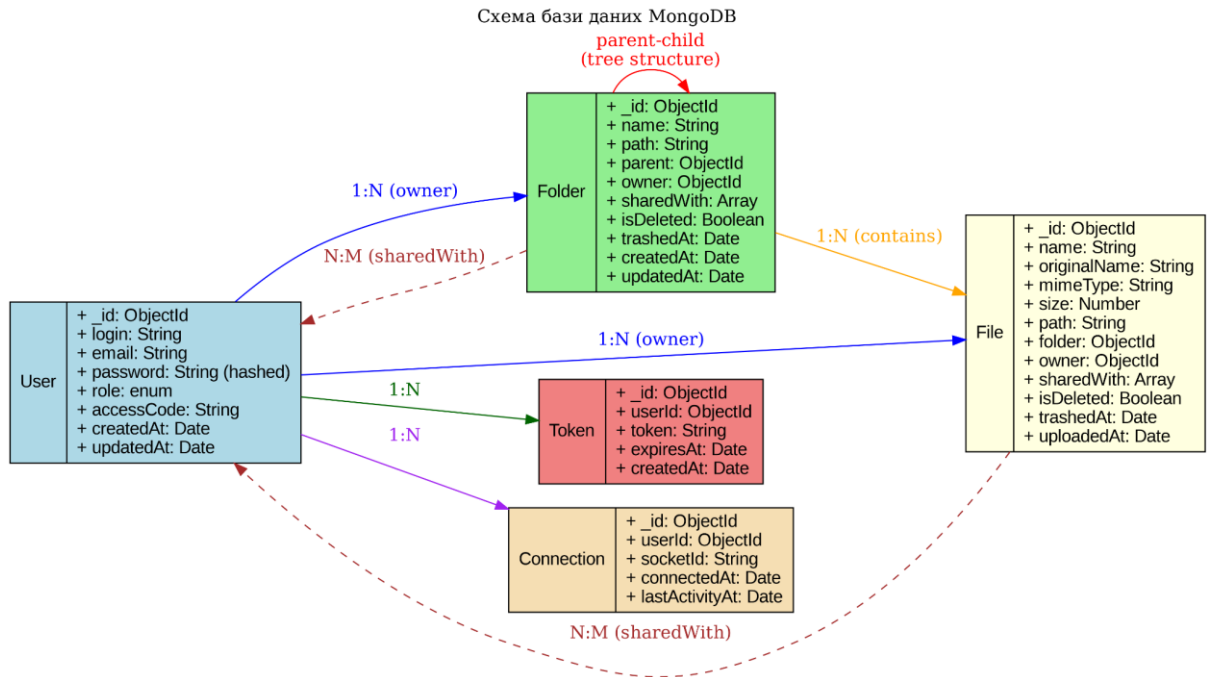


Рисунок 2.3 – Структура бази даних MongoDB з зв'язками

User model містить наступні поля: `login` (унікальний ідентифікатор), `email` (з валідацією формату), `password` (хешований bcrypt з salt rounds 10), `role` (enum: 'admin', 'user', 'owner', 'guest'), `accessCode` (6-значний код для гостей акаунтів), `createdAt` та `updatedAt` timestamps. Модель включає pre-save hook для автоматичного хешування пароля та методи для порівняння паролів. Індеси створені на `email` та `accessCode` для швидкого пошуку.

Folder model представляє деревовидну структуру папок: `name` (назва папки), `path` (повний шлях в файловій системі), `parent` (ObjectId посилання на батьківську папку або null для root), `owner` (ObjectId користувача-власника), `sharedWith` (масив об'єктів з user ObjectId та permissions enum ['read', 'write', 'delete']), `isDeleted` (soft delete прапорець), `trashedAt` (дата видалення для відновлення протягом 30 днів). Compound індекси налаштовані на (`owner`, `isDeleted`) та (`parent`, `owner`) для оптимізації запитів на отримання вмісту папки.

File model зберігає метадані файлів: `name` (ім'я файлу в ФС), `originalName` (оригінальне ім'я при завантаженні), `mimeType` (тип контенту), `size` (розмір в байтах), `path` (повний шлях), `folder` (ObjectId батьківської папки), `owner` (ObjectId власника), `sharedWith` (аналогічно Folder з permissions), `isDeleted`, `trashedAt`,

uploadedAt. Індокси на (folder, isDeleted) та (owner, mimeType) прискорюють отримання списку файлів та фільтрацію по типу.

Token model для JWT refresh токенів: userId (ObjectId користувача), token (refresh token string), expiresAt (дата закінчення дії), createdAt. TTL індекс на expiresAt автоматично видаляє застарілі токени. При logout або refresh токен інвалідується видаленням з колекції.

Connection model відстежує активні підключення: userId (ObjectId), socketId (Socket.io session ID), connectedAt, lastActivityAt. Використовується для відображення онлайн статусу користувачів та force disconnect при видаленні акаунта. Індекс на userId для швидкого отримання всіх сесій користувача.

Для забезпечення референційної цілісності налаштовані Mongoose populate для автоматичного розв'язання ObjectId посилань при запитах. Всі моделі використовують timestamps: true опцію для автоматичного управління createdAt та updatedAt полями.

2.2.3 API endpoints та маршрутизація

RESTful API побудовано за принципами resource-oriented design з версіонуванням через префікс /api. Всі ендпоінти повертають стандартизований JSON формат: {success: boolean, data: any, error?: string, message?: string}.

Authentication endpoints (/api/auth): POST /register приймає {login, email, password} та створює користувача з роллю 'user', POST /login валідує credentials та повертає access token (JWT в header) + refresh token (httpOnly cookie), POST /logout інвалідує refresh token, POST /refresh оновлює access token використовуючи refresh token з cookie, GET /me повертає дані поточного користувача (потребує JWT).

Folder endpoints (/api/folders): GET /my повертає всі папки користувача з опцією ?includeShared=true, GET /:id отримує деталі папки з вмістом (файли + підпапки), POST / створює нову папку {name, parent?}, PATCH /:id оновлює назву папки, DELETE /:id soft delete (переміщення в trash), POST /:id/restore відновлення з trash, DELETE /:id/permanent остаточне видалення, POST /:id/share надає доступ

{userId, permissions: ['read', 'write', 'delete']}, DELETE /:id/share/:userId відкликає доступ.

File endpoints (/api/files): GET /folder/:folderId список файлів в папці з фільтрацією ?mimeType=image/*, GET /:id метадані файлу, DELETE /:id soft delete, POST /:id/restore відновлення, DELETE /:id/permanent видалення файлу з ФС + БД, POST /upload multipart/form-data завантаження з полями {file: File, folderId: string, tags?: string[]}, PATCH /:id/move переміщення файлу {newFolderId}.

Download endpoints (/api/download): GET /file/:id streaming завантаження одного файлу з підтримкою Range headers для partial content, POST /multiple приймає масив {fileIds: string[]} та повертає ZIP архів, GET /folder/:id завантаження всієї папки як ZIP.

User management (/api/users): GET / список користувачів (admin only), GET /:id деталі користувача, POST / створення користувача (admin), PATCH /:id оновлення даних, DELETE /:id видалення користувача з каскадним видаленням його файлів, PATCH /:id/role зміна ролі (owner only).

Guest endpoints (/api/guests): POST /create генерує гостьовий акаунт з випадковим 6-значним accessCode та роллю 'guest', POST /login автентифікація по {accessCode}, GET /access/:code перевірка валідності коду, DELETE /:id видалення гостьового акаунта (після закінчення сесії).

Search endpoints (/api/search): GET /files?query=name&mimeType=image/*&tags=important пошук файлів по name (text index), фільтрація по mimeType (regex), tags (array contains), GET /folders?query=name пошук папок по назві.

Connection tracking (/api/connections): GET /online список онлайн користувачів, GET /my-sessions активні сесії поточного користувача, DELETE /session/:socketId force disconnect конкретної сесії.

Всі захищені ендпоінти використовують protectedRoute middleware для верифікації JWT. Операції з файлами/папками додатково перевіряють права доступу через checkResourceAccess middleware, який валідує чи користувач є owner

або має відповідні permissions в sharedWith масиві. Admin ендпоінти захищені adminAndOwner middleware.

Для обробки помилок налаштовано глобальний error handler middleware, який ловить всі винятки, логує їх через Winston та повертає клієнту стандартизовану error response з відповідним HTTP status code (400 – validation, 401 – unauthorized, 403 – forbidden, 404 – not found, 500 – server error).

2.2.4 WebSocket real-time комунікація

Socket.io інтегровано для двостороннього real-time зв'язку між сервером та клієнтами. З'єднання встановлюється при успішній автентифікації користувача, JWT токен передається через query параметр або auth header при handshake.

При підключенні виконується Socket.io middleware для валідації токена: декодується JWT, витягується userId, створюється record в connections колекції з socketId. Socket приєднується до room з назвою user:\${userId} для персональних повідомлень та до room folder:\${folderId} для кожної папки, до якої користувач має доступ.

Реалізовано обробку наступних events:

Client → Server events: join_folder – підключення до room папки при навігації, leave_folder – відключення при зміні директорії, ping – heartbeat для підтримки з'єднання, disconnect – очищення connections та видалення з rooms.

Server → Client events: file_uploaded – broadcasts в folder:\${folderId} room при завантаженні нового файлу payload {file: FileMetadata, folderId}, file_deleted – при видаленні файлу payload {fileId, folderId}, file_restored – при відновленні з trash, folder_created – при створенні підпапки, folder_shared – emit в user:\${recipientId} room при отриманні доступу до папки payload {folder, permissions, sharedBy}, folder_unshared – при відкликанні доступу, user_connected / user_disconnected – broadcast всім онлайн користувачам для оновлення статусів payload {userId, timestamp}, storage_updated – при зміні доступного місця на диску.

Для оптимізації використовується rooms механізм Socket.io: кожна папка має окремий room, події broadcasts тільки користувачам, які мають доступ до цієї папки.

При отриманні події на клієнті автоматично оновлюється відповідний стан в Zustand store без необхідності ручного refresh.

Реалізовано reconnection logic на клієнті: при розриві з'єднання (мережева помилка, server restart) автоматично виконується повторне підключення з exponential backoff (затримки 1s, 2s, 4s, 8s до максимум 30s), після успішного reconnect відправляється sync_state запит для отримання пропущених змін.

Error handling включає обробку помилок автентифікації (disconnect з повідомленням), full message queue при недоступності клієнта (зберігання останніх 50 повідомлень для доставки при reconnect), timeout detection (disconnect якщо немає pong відповіді протягом 60 секунд).

2.3 Розробка клієнтської частини (Frontend)

2.3.1 Структура frontend додатку

Клієнтська частина реалізована як Single Page Application (SPA) на React 18 з TypeScript для типобезпеки. Використовується Vite як build tool завдяки швидкому HMR (Hot Module Replacement) та оптимізованому production bundle.

Точкою входу є index.html з div root елементом, в якому React рендерить додаток через main.tsx. Кореневий компонент App.tsx налаштовує глобальні провайдери (SocketContext, ThemeProvider) та ініціалізує React Router для client-side маршрутизації.

Архітектура frontend (зображено на рисунку 2.4) організована за feature-based структурою з дев'яти основних директорій. API layer (api/http.ts) містить налаштований Axios instance з base URL import.meta.env.VITE_API_URL, request interceptor для додавання JWT токена з localStorage до Authorization header, response interceptor для автоматичного refresh токена при 401 помилці (викликається /api/auth/refresh, оновлюється токен, retry original request), error interceptor для показу toast нотифікацій про помилки.

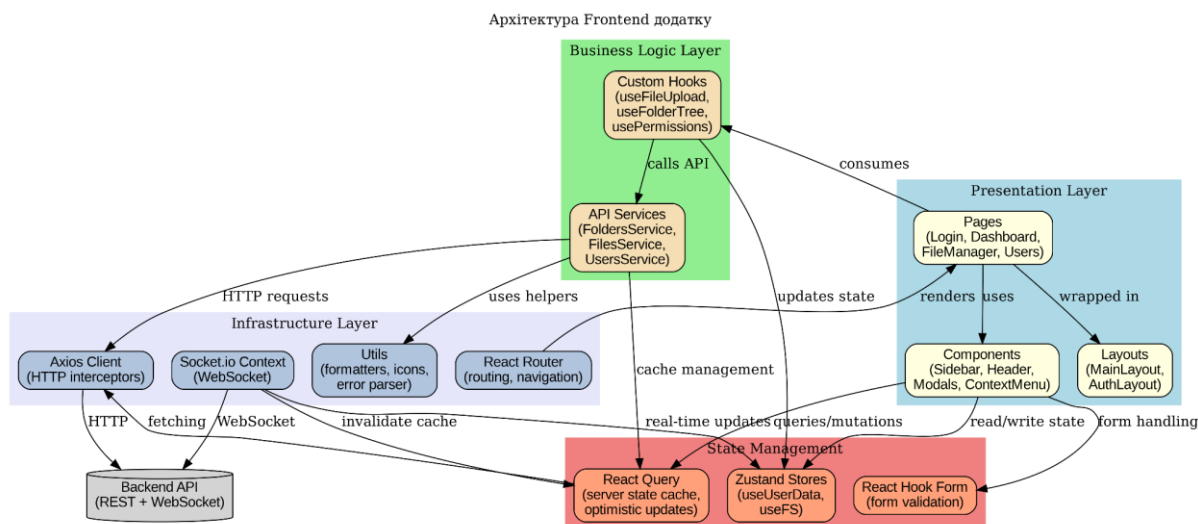


Рисунок 2.4 – Архітектура клієнтської частини додатку

Директорія Components поділена на підкатегорії: Common/ включає Sidebar/ з навігацією (Dashboard, My Files, Shared, Trash, Users, Settings), Header/ з інформацією про користувача та logout кнопкою, Footer/ з версією додатку. ContextMenu/ реалізує right-click меню для файлів/папок з опціями (Open, Download, Rename, Move, Share, Delete) в залежності від permissions. Loader/ містить спінери для різних станів (page load, file upload, API call). Modals/ включають діалоги: ConfirmModal.tsx для підтвердження дій, CreateFolderModal.tsx з формою назви та вибору батьківської папки, ShareFolderModal.tsx з вибором користувача та checkboxes для permissions (read, write, delete). ThemeSwitcher/ реалізує toggle між світлою та темною темою з збереженням вибору в localStorage. Toasters/ використовує react-hot-toast для показу повідомлень про успіх/помилки.

Pages містять top-level компоненти: Auth/Login.tsx – форма з email/password полями та “Remember me” checkbox, Auth/Register.tsx – реєстрація з валідацією (email format, password strength мінімум 8 символів), Auth/GuestLogin.tsx – вхід по 6-значному коду. Dashboard/Dashboard.tsx показує статистику (total files, used space, recent uploads) з charts через recharts бібліотеку. Storage/Folders/FoldersComponent.tsx – основний файловий менеджер з grid/list view switcher, breadcrumbs навігацією, drag-and-drop upload area, multi-select для bulk операцій. Storage/Folders/ContentUploader/ обробляє file upload з progress bar (react-

circular-progressbar), multiple file selection, drag-and-drop підтримкою, валідацією типів/розмірів файлів, thumbnail preview для зображень. Users/UsersList.tsx – таблиця користувачів з фільтрацією по ролі, пошуком, pagination, Users/UserForm.tsx – форма створення/редагування користувача. Settings/Settings.tsx – налаштування профілю, зміна паролю, theme preferences.

Services layer абстрагує API виклики: FoldersService.tsx експортує методи getMyFolders(includeShared?), getFolderById(id), createFolder(name, parent?), shareFolder(id, userId, permissions), deleteFolder(id), restoreFolder(id), getTrashed(). FilesService.tsx – getFilesInFolder(folderId, mimeType?), uploadFile(file, folderId, tags?) з progress callback, deleteFile(id), moveFile(id, newFolderId), downloadFile(id) через blob response. UsersService.tsx – CRUD операції для користувачів. GuestService.tsx – створення та login гостьових акаунтів.

Zustand використовується для глобального стану: useUserData.tsx store містить authUser (current user object), isAuthenticated (boolean), login(credentials), logout(), refreshUser() actions, persisted в localStorage через Zustand persist middleware. useFS.tsx store для file system navigation: currentFolder (ObjectId), folderStack (масив для history), selectedFiles (Set для multi-select), viewMode ('grid' | 'list'), sortBy ('name' | 'date' | 'size'), navigateToFolder(id), goBack(), selectFile(id), clearSelection() actions.

Hooks містять reusable logic: useFileUpload.ts – custom hook для upload з state management (uploading, progress, error), chunk upload підтримкою для великих файлів (розбиття на 5MB chunks), retry logic при помилках. useFolderTree.ts – recursive hook для побудови дерева папок з lazy loading (підпапки завантажуються при expand). usePermissions.ts – hook для перевірки прав доступу canRead(resource), canWrite(resource), canDelete(resource) на основі authUser та resource.sharedWith.

Utils включають helper функції: fileIcons.tsx мапінг MIME types на React icons (FaFile, FaFileImage, FaFilePdf тощо з react-icons/fa6), errorsParser.ts перетворює API error responses на user-friendly повідомлення українською, formatters.ts містить formatFileSize(bytes) (B, KB, MB, GB), formatDate(timestamp) з відносними датами

(“2 hours ago”, “Yesterday”), constants.ts – app constants (MAX_FILE_SIZE, ALLOWED_EXTENSIONS, API_URL).

Routes в AppRouter.tsx налаштовані з React Router v6: публічні роути (/ , /login, /register, /guest-login) рендеряться в AuthLayout, захищені роути обгорнуті в ProtectedRoute wrapper (перевіряє isAuthenticated, redirect на /login якщо ні) та рендеряться в MainLayout, admin роути додатково обгорнуті в AdminRoute (перевіряє authUser.role). Використовується nested routes для folder navigation /folders/:folderId? з optional параметром.

Context SocketContext.tsx провайдить Socket.io клієнт всім компонентам: встановлює з'єднання при mount, передає JWT через auth query, підписується на глобальні events (user_connected, storage_updated), експортує socket instance та emit helper. Custom hook useSocket() надає доступ до контексту.

Layouts: MainLayout.tsx комбінує <Sidebar />, <Header />, <Outlet /> (для nested routes), <Footer /> з responsive breakpoints (sidebar згортається в burger menu на mobile), AuthLayout.tsx показує лише <Outlet /> по центру екрану з background gradient.

Стилізація виконана через Tailwind CSS з custom theme в tailwind.config.js: кольорова палітра (primary, secondary, accent в oklch форматі), spacing scale (4px base unit), border radius, typography scale, custom utilities для glassmorphism ефектів. Глобальні стилі в index.css включають CSS variables для theme switching, custom scrollbar styling, animations (@keyframes для fade-in, slide-up).

Build configuration в vite.config.ts налаштовує alias imports (@components, @pages, @services), plugin для auto-import React, production optimizations (code splitting по route, tree shaking, minification), dev server з проху для API requests (CORS issues під час розробки).

2.3.2 Управління станом застосунку

Для state management використовується комбінація підходів залежно від scope та nature даних. Глобальний стан керується через Zustand – lightweight

альтернатива Redux без boilerplate, що забезпечує типобезпечність TypeScript та простий API.

useUserData store містить authentication state з persist middleware для збереження між сесіями. Initial state завантажується з localStorage при app mount, якщо accessToken присутній – виконується запит /api/auth/me для валідації та отримання актуальних даних користувача. Store включає computed селектори: isGuest (перевірка role === 'guest'), isAdmin (role === 'admin' || role === 'owner'), fullName (комбінація firstName + lastName). Actions обробляють async операції: login викликає AuthService, зберігає токен, оновлює state та redirect на dashboard, logout очищує токен, reset store, disconnect socket, redirect на login page, updateProfile – optimistic update з rollback при помилці. Middleware включає Redux DevTools інтеграцію для debugging у development mode.

useFS store керує navigation state файлового менеджера. currentFolder null для root level або ObjectId для вкладених папок, folderStack масив для history навігації (використовується для Back button), breadcrumbs computed property на основі stack для UI breadcrumbs. selectedFiles Set для O(1) lookup при multi-select, selectMode boolean для toggle bulk selection mode. Actions navigateToFolder додає папку до stack та оновлює currentFolder, goBack pop зі stack, clearSelection reset Set. Store також містить viewPreferences object з sortBy, sortDirection, viewMode (grid/list), filters (mimeType, dateRange) що persist в localStorage per-user через ключ fs:\${userId}.

Server state керується через React Query (TanStack Query) для автоматичного кешування, refetch on window focus, background sync, optimistic updates. Query keys організовані ієрархічно: ['folders', 'my'] для getMyFolders, ['folders', folderId] для конкретної папки, ['files', folderId] для файлів в папці. Mutations налаштовані з onMutate для optimistic updates (негайне оновлення UI перед сервером response), onError для rollback при помилці, onSuccess для invalidate queries та toast notifications. Приклад: при створенні папки – optimistic insert в cached list, при server response – invalidate ['folders', parentId] query для refetch актуальних даних.

Form state управляється через React Hook Form з Zod validation schema. Всі форми (login, register, create folder, share) використовують useForm hook з

zodResolver для type-safe validation rules. Приклад schema для create folder: `z.object({name: z.string().min(1).max(255).regex(/^[^\\:*\?"<>|]+$/), parent: z.string().nullable()})`. Форми підтримують isDirty, isSubmitting, errors states, async submission з обробкою server-side validation errors через setError.

Component-local state через useState для UI-only стану: modal visibility, dropdown open/close, temporary input values. useReducer використовується для складних state transitions (напр. multi-step wizard для batch operations).

Socket events інтегровані з React Query для auto-sync: при отриманні file_uploaded event викликається queryClient.invalidateQueries(['files', event.folderId]), що тригерить refetch файлів в папці. folder_shared event додає нову папку в cached ['folders', 'my'] list через queryClient.setQueryData без повного refetch. Event handlers обгорнуті в useEffect з cleanup для unsubscribe при unmount.

Context API використовується лише для dependency injection: SocketContext надає socket instance, ThemeContext – current theme та toggle function. Data не зберігається в Context для уникнення unnecessary re-renders.

Performance оптимізації включають: React.memo для expensive components (FileGrid, FolderTree), useMemo для computed values (filtered/sorted lists), useCallback для event handlers передаваних як props, virtualization через react-window для lists з 100+ items, code splitting через React.lazy + Suspense для pages. State updates batched через startTransition для non-urgent updates (search filtering).

2.3.3 Інтеграція з API та маршрутизація

Взаємодія з backend реалізована через Axios HTTP client з налаштованою інстанцією в api/http.ts. Base URL конфігурується через environment variable VITE_API_URL, що дозволяє легко перемикатись між development (http://localhost:5000/api) та production endpoints. Встановлено withCredentials: true для автоматичної передачі httpOnly cookies з refresh токенами.

Request interceptor додає JWT access token з localStorage до Authorization header у форматі Bearer \${token} для всіх захищених запитів. Перед кожним запитом перевіряється наявність токена та його expiration time (декодується без

верифікації для отримання exp claim), якщо токен скоро закінчиться (менше 5 хвилин до expiration) – превентивно викликається refresh endpoint.

Response interceptor обробляє три сценарії: успішні відповіді (2xx) повертаються без змін, 401 Unauthorized тригерить автоматичний refresh flow – викликається POST /api/auth/refresh з refresh token з cookies, при успіху новий access token зберігається в localStorage та original запит повторюється з оновленим токеном, при помилці refresh (refresh token invalid або expired) користувач логавитись і redirect на login page. Для 403 Forbidden та інших помилок викликається handleApiError utility, яка парсить error response (error.message або generic message) та показує toast notification через react-hot-toast.

Service layer організовано за resource-based принципом. FoldersService експортує async функції: getMyFolders(includeShared = false) виконує GET /api/folders/my?includeShared=\${includeShared}, повертає масив папок з populated owner та sharedWith fields, getFolderById(id) отримує деталі папки включно з files та subfolders, createFolder({name, parent}) POST запит для створення, updateFolder(id, {name}) PATCH для перейменування, deleteFolder(id) soft delete, restoreFolder(id) відновлення з trash, permanentDeleteFolder(id) безповоротне видалення, shareFolder(id, {userId, permissions}) надає доступ, unshareFolder(id, userId) відкликає. Всі методи повертають Promise з typed response (TypeScript interface для FolderDTO).

FilesService аналогічно містить: getFilesInFolder(folderId, filters?) з optional filters для mimeType/tags, uploadFile(file, folderId, tags, onProgress) використовує FormData для multipart upload, приймає callback onProgress(percentage) для progress bar через Axios onUploadProgress config, підтримує chunk upload для файлів > 50MB (розбиття на chunks, окремі запити з Content-Range header, server-side assembly), downloadFile(id) робить GET з responseType: 'blob', створює temporary URL через URL.createObjectURL та тригерить browser download через programmatic <a> click, moveFile(id, newFolderId), deleteFile(id), restoreFile(id).

UserService та GuestService надають CRUD методи для user management, доступні тільки адміністраторам. AuthService містить login(credentials),

register(userData), logout(), refreshToken(), getCurrentUser() – ці методи інтегровані з Zustand stores для sync authentication state.

Маршрутизація налаштована через React Router v6 в AppRouter.tsx. Використовується createBrowserRouter для data router API з підтримкою loaders (data fetching перед render), actions (form submissions), error boundaries. Root route містить <App /> з nested routes, всі routes обгорнуті в відповідні layouts.

Публічні routes доступні без автентифікації: /login рендерить <Login />, /register – <Register />, /guest-login – <GuestLogin />. При спробі доступу аутентифікованого користувача – автоматичний redirect на /dashboard.

Захищені routes обгорнуті в <ProtectedRoute /> компонент, який перевіряє isAuthenticated з Zustand store: якщо false – redirect на /login зі збереженням intended destination в location.state для повернення після login. Захищені routes include: /dashboard – домашня сторінка з статистикою, /folders – root level файлового менеджера, /folders/:folderId – nested folder view, /shared – папки shared з користувачем, /trash – видалені files/folders, /settings – user profile та preferences.

Admin routes додатково обгорнуті в <AdminRoute />, який перевіряє authUser.role === 'admin' || authUser.role === 'owner', інакше redirect на /dashboard з error toast. Admin routes: /users – список всіх користувачів, /users/new – форма створення користувача, /users/:id/edit – редагування, /users/:id – деталі користувача.

Nested routes для folders використовують optional параметр /folders/:folderId? – якщо folderId відсутній, показується root level. Breadcrumbs генерується автоматично на основі folderStack з Zustand useFS store, кожен breadcrumb – clickable link для навігації.

Програмна навігація виконується через useNavigate() hook: після успішного upload – navigate(\folders/\${folderId}`), після logout -navigate('/login', {replace: true})(replace для видалення з history).useParams() витягує route параметри (напр.folderId),useSearchParams()для query strings (фільтри, pagination).useLocation()` надає current location для conditional rendering (напр. highlight active link в sidebar).

Error boundaries налаштовані на route level для graceful error handling: при помилці в loader або component показується fallback UI з error message та кнопкою “Go back”. Network errors обробляються глобально через Axios interceptor з retry logic (3 спроби з exponential backoff).

2.3.4 UI/UX дизайн та стилізація

Дизайн системи побудована на принципах Material Design з адаптацією під теплу колірну гаму та modern glassmorphism ефекти. Використовується Tailwind CSS utility-first framework для швидкої розробки з кастомними extensions в tailwind.config.js.

Колористика базується на oklch color space для perceptually uniform colors. Світла тема використовує теплу палітру: --color-base-100: oklch(98% 0 0) білий background, --color-base-200: oklch(96% 0.007 70) світлий бежевий для cards, --color-base-300: oklch(93% 0.009 70) для borders/dividers, --color-primary: oklch(75% 0.18 40) теплий помаранчевий для accent elements (buttons, links, active states), --color-secondary: oklch(89% 0.04 75) пісочний для secondary buttons, --color-accent: oklch(98% 0.005 50) для subtle highlights.

Темна тема інвертує яскравість зберігаючи hue: --color-base-100: oklch(18% 0.01 40) теплий майже чорний background, --color-base-200: oklch(22% 0.01 40) для cards з легким elevation, --color-base-300: oklch(28% 0.015 40) для borders. Primary та secondary colors залишаються незмінними для consistency, але з adjusted opacity для контрасту на темному фоні.

Типографіка використовує system font stack для native look: font-family: system-ui, -apple-system, Segoe UI, Roboto, Helvetica Neue, Arial, sans-serif. Font sizes організовані в modular scale з base 16px: --font-size-xs: 0.75rem (12px), --font-size-sm: 0.875rem (14px), --font-size-base: 1rem (16px), --font-size-lg: 1.125rem (18px), --font-size-xl: 1.25rem (20px), --font-size-2xl: 1.5rem (24px) для headings. Line heights адаптовані для readability: 1.5 для body text, 1.25 для headings, 1.75 для prose content.

Spacing system базується на 8px grid з Tailwind utilities: spacing-1: 0.25rem (4px), spacing-2: 0.5rem (8px), spacing-3: 0.75rem (12px), spacing-4: 1rem (16px),

spacing-6: 1.5rem (24px), spacing-8: 2rem (32px), spacing-12: 3rem (48px), spacing-16: 4rem (64px). Всі paddings/margins використовують ці значення для visual consistency.

Border radius прогресивні для depth indication: --radius-sm: 0.25rem (4px) для small elements (badges, tags), --radius-md: 0.5rem (8px) для inputs/buttons, --radius-lg: 1rem (16px) для cards/modals, --radius-xl: 1.5rem (24px) для large containers.

Shadows створюють subtle elevation: три рівні shadow для depth – shadow-sm для inputs (0 1px 2px rgba), shadow-md для cards (0 4px 6px rgba), shadow-lg для modals (0 10px 15px rgba). В темній темі shadows темніші з вищою opacity для visibility.

Адаптивний дизайн реалізований mobile-first approach з breakpoints: default styles для xs (0-639px), @media (min-width: 640px) для sm (tablets portrait), @media (min-width: 768px) для md (tablets landscape/small laptops), @media (min-width: 1024px) для lg (desktops), @media (min-width: 1280px) для xl (large desktops). Sidebar згортається в hamburger menu на mobile (<768px), file grid змінює columns від 1 (mobile) до 2 (tablet) до 4 (desktop). Touch targets мінімум 44x44px на mobile для accessibility.

Анімації додають polish: transitions налаштовані з transition-duration: 200ms (fast), 300ms (normal), 500ms (slow), transition-timing-function: cubic-bezier(0.4, 0, 0.2, 1) (ease-in-out) для smooth motion. Keyframe animations: @keyframes fade-in для modals (opacity 0 → 1), @keyframes slide-up для toasts (translateY(20px) → 0), @keyframes spin для loaders. Page transitions використовують react-transition-group для enter/exit animations між routes.

Interactive states чітко відмічені: hover (lighten 10%, scale 1.02 для buttons), active (darken 5%, scale 0.98), focus (outline ring з primary color, 2px offset), disabled (opacity 0.5, cursor not-allowed). File/folder items підсвічуються на hover, multi-select показує checkboxes fade-in.

Glassmorphism ефекти для cards/modals: backdrop-filter: blur(10px), background: rgba(white, 0.7) в світлій темі або rgba(black, 0.5) в темній, subtle border 1px solid rgba(white, 0.2) для edge definition. Створює depth без heavy shadows.

Accessibility features: всі interactive elements keyboard navigable (tab order), focus indicators з high contrast, ARIA labels для screen readers, semantic HTML (nav, main, article tags), color contrast ratio мінімум 4.5:1 для WCAG AA compliance, reduced motion respect через @media (prefers-reduced-motion: reduce) – disable animations для users з motion sensitivity.

Theme switching реалізовано через CSS variables swap: при toggle button click – JavaScript змінює data-theme=light dark attribute на <html>, CSS variables переоб'являються в :root[data-theme=dark] selector. Вибір теми зберігається в localStorage та sync між tabs через storage event listener. Автоматичне визначення system preference через window.matchMedia('(prefers-color-scheme: dark)') при первинному завантаженні.

2.4 Вибір технологій та середовища програмування

Для реалізації веб-додатку управління NAS обрано сучасний JavaScript-based tech stack, що забезпечує високу продуктивність, масштабованість та developer experience. Вибір технологій обґрунтовувався кількома ключовими критеріями: активна спільнота та підтримка, наявність готових рішень для типових задач, типобезпека та tooling, performance характеристики, легкість deployment.

Backend stack побудований на Node.js v18 LTS – асинхронна event-driven платформа на V8 engine, оптимальна для I/O-intensive операцій (файлові операції, database queries, WebSocket connections). Express.js 4.18 обрано як мінімалістичний web framework з великою екосистемою middleware та patterns для RESTful API. Mongoose 7.0 надає ODM layer для MongoDB з schema validation, middleware hooks, query building, забезпечуючи type safety та зручний API для роботи з документами.

MongoDB 6.0 обрано як NoSQL database завдяки гнучкості схем (дозволяє еволюцію структури даних без migrations), природній підтримці вкладених документів та масивів (ідеально для деревовидної структури папок та sharedWith permissions), horizontal scalability через sharding, GridFS для зберігання великих файлів якщо потрібно. Альтернативи PostgreSQL або MySQL відкинуті через складність моделювання рекурсивних структур та жорсткі схеми.

Socket.io 4.5 використовується для WebSocket комунікації з fallback на long polling, надає rooms/namespaces abstraction для організації event channels, автоматичний reconnection з exponential backoff, binary support для передачі файлів. JWT (jsonwebtoken 9.0) реалізує stateless authentication з access/refresh token pattern: access token (15 хв expiry) зберігається в localStorage клієнта, refresh token (30 днів) в httpOnly secure cookie, що мінімізує XSS ризики.

Multer middleware обробляє multipart/form-data uploads з disk storage engine, автоматичною генерацією унікальних імен файлів через crypto.randomBytes, валідацією MIME types через file-type library, обмеженням розміру через limits config. Winston logger забезпечує structured logging з різними transports (file rotation, console, remote aggregation), log levels (error, warn, info, debug), JSON formatting для машинної обробки.

Frontend stack базується на React 18 з concurrent features (useTransition, useDeferredValue) для smooth UI під час heavy operations, automatic batching для оптимізації re-renders, Suspense для declarative loading states. TypeScript 5.0 надає static typing з strict mode, catching bugs at compile time, intelligent autocomplete, refactoring confidence, interface definitions як documentation.

Vite 4.0 обрано як build tool замість Webpack завдяки instant HMR через native ES modules, швидкому cold start (no bundling в dev mode), optimized production build через Rollup з tree shaking та code splitting. Vite plugin ecosystem забезпечує React Fast Refresh, TypeScript checking, SVG as components, environment variables handling.

React Router v6 реалізує client-side routing з data router API (loaders для data fetching перед render, actions для form submissions, errorElement для error boundaries), nested routes з Outlet composition, programmatic navigation через useNavigate, type-safe параметри через useParams generic. Zustand обрано замість Redux завдяки мінімальному boilerplate (no actions/reducers, direct state mutation), TypeScript-first API, middleware support (persist, devtools, immer), fine-grained subscriptions для performance.

TanStack Query (React Query) керує server state з automatic caching (in-memory cache з configurable staleTime), background refetching (refetch on window focus,

polling), optimistic updates (immediate UI update перед server response з rollback on error), request deduplication (multiple components requesting same data = single network call), infinite queries для pagination. Axios залишається HTTP client завдяки interceptors для global request/response handling, automatic JSON transformation, TypeScript support, cancel token для aborting requests, progress events для upload progress.

Tailwind CSS 3.3 реалізує utility-first styling з JIT compiler (generate classes on demand, instant build), purge unused CSS (final bundle < 10KB), custom design tokens в config, arbitrary values синтаксис w-[37px], responsive/state variants hover:, md:, dark:. React Hook Form з Zod validation забезпечує performant forms (uncontrolled inputs, minimal re-renders), type-safe validation schemas, async validation support, integration з UI libraries.

Development environment включає ESLint з recommended configs + custom rules для code quality enforcement, Prettier для automatic formatting (no style debates), Husky git hooks для pre-commit linting/testing, commitizen для conventional commits. VS Code з extensions (ESLint, Prettier, TypeScript, Tailwind IntelliSense) надає intelligent autocomplete та error highlighting.

Testing stack складається з Vitest (Vite-native testing framework з instant watch mode), React Testing Library (testing components як користувач), MSW для API mocking, Playwright для E2E tests. Git з feature branch workflow та pull requests для code review процесу.

Deployment setup використовує PM2 process manager для Node.js (automatic restarts on crash, cluster mode для multi-core utilization, zero-downtime reloads, log management), Nginx як reverse proxy (SSL termination, static file serving з compression, rate limiting, load balancing), Docker для containerization (reproducible environments, easy scaling, dependency isolation). MongoDB Atlas розглядається для production database з automatic backups, point-in-time recovery, monitoring dashboards.

Environment configuration через dotenv з окремими .env файлами для dev/staging/production, secrets management через environment variables

(DATABASE_URL, JWT_SECRET, AWS_KEYS). CI/CD pipeline через GitHub Actions автоматизує testing, building, deployment при push до main branch

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Архітектура системи

3.1.1 Загальна структура

Система 3mHome побудована за клієнт-серверною архітектурою з використанням веб-технологій. Взаємодія між компонентами здійснюється через RESTful API та WebSocket з'єднання для real-time оновлень. Структура компонентів Frontend додатку зображена на рисунку 3.1.

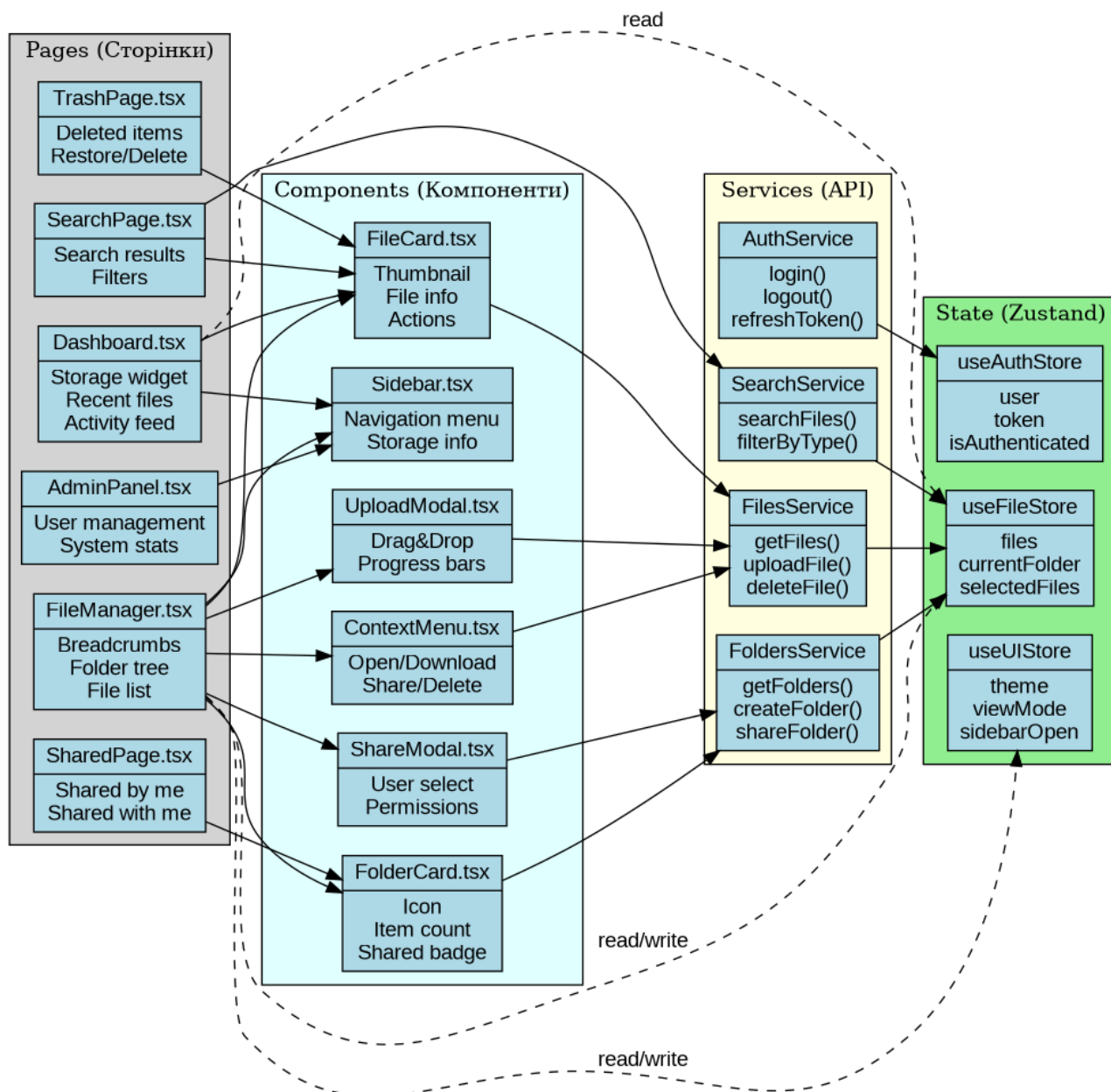


Рисунок 3.1 – Структура компонентів Frontend додатку

Основні компоненти:

1. Frontend (клієнтська частина):

- React 18 + TypeScript
- TailwindCSS для стилізації
- Socket.io-client для real-time
- Axios для HTTP запитів

2. Backend (серверна частина):

- Node.js + Express
- Socket.io для WebSocket
- JWT для автентифікації
- Multer для завантаження файлів

3. База даних:

- MongoDB для метаданих
- Файлова система для зберігання файлів

4. Інфраструктура:

- Winston для логування
- JWT для сесій
- Bcrypt для хешування паролів

3.1.2 Модель даних

Основні колекції, що зображені на рисунку 3.2:

Users

- інформація про користувачів
- _id, username, email, password (hashed)
- role (admin/user), createdAt, settings

Folders

- структура папок
- _id, name, parentId, ownerId
- path, size, itemCount, createdAt

Files

- метадані файлів
- _id, name, folderId, ownerId
- size, mimeType, path, createdAt

Permissions

- права доступу
- _id, resourceId, resourceType (file/folder)
- userId, permissions (read/write/delete)

GuestAccess

- тимчасовий доступ
- _id, code, createdBy, expiresAt
- targetFolderId, permissions

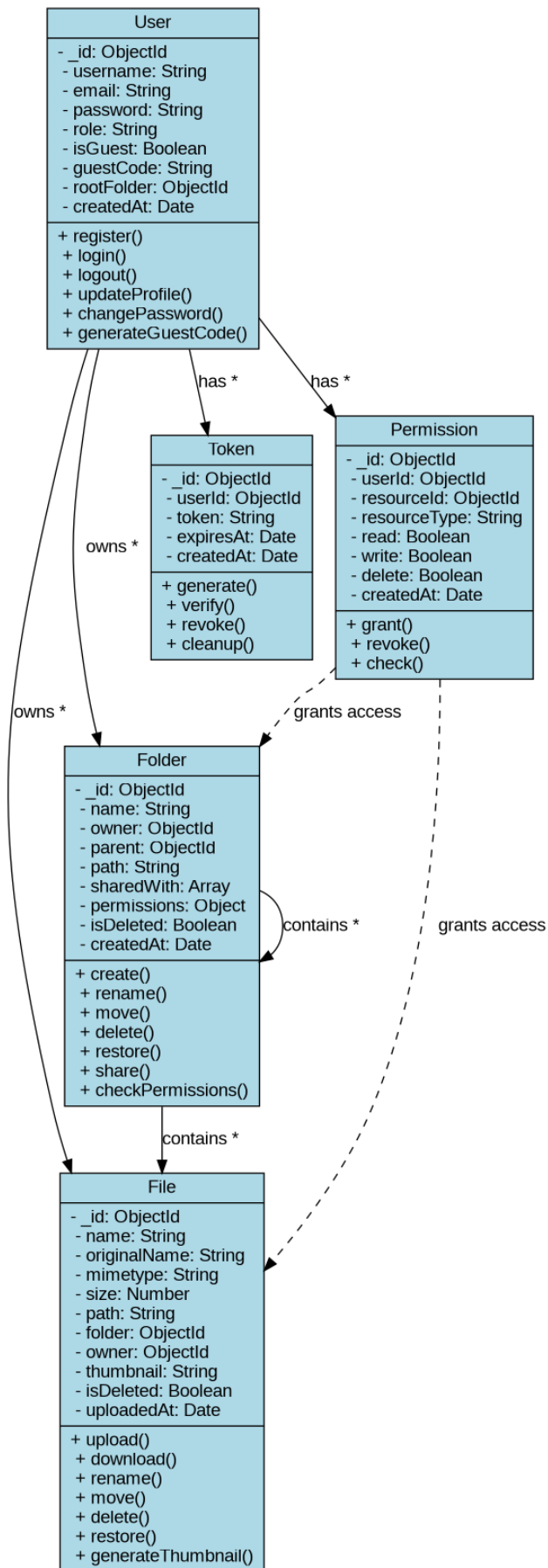


Рисунок 3.2 – Діаграма класів та їх взаємозв'язків

3.1.3 API та маршрутизація

В таблиці 3.1 зображено основні методи API.

Таблиця 3.1 – Основні API endpoints

Метод	Endpoint	Опис	Auth
POST	/api/auth/login	Вхід в систему	-
POST	/api/auth/register	Реєстрація	-
GET	/api/folders/:id	Отримати вміст папки	✓
POST	/api/folders	Створити папку	✓
POST	/api/files/upload	Завантажити файл(и)	✓
GET	/api/files/:id/download	Скачати файл	✓
DELETE	/api/files/:id	Видалити файл	✓
POST	/api/permissions/share	Надати доступ	✓
GET	/api/search?q=query	Пошук файлів	✓
POST	/api/guest/generate	Згенерувати код доступу	✓

WebSocket події:

Server -> Client

'file:uploaded' – Новий файл завантажено

'file:deleted' – Файл видалено

'folder:updated' – Папка оновлена

'permission:changed' – Права змінено

Client -> Server

'subscribe:folder' – Підписатись на оновлення папки

'unsubscribe:folder' – Відписатись

3.2 Реалізація основного функціоналу

3.2.1 Автентифікація та авторизація

Процес автентифікації:

- Користувач вводить credentials (email + password) (форму входу зображено на рисунку 3.3)

- Сервер перевіряє дані в БД (bcrypt.compare)
- При успіху генерується JWT токен (термін дії: 7 днів)
- Токен зберігається в localStorage клієнта
- Кожен запит містить токен в header Authorization: Bearer <token>

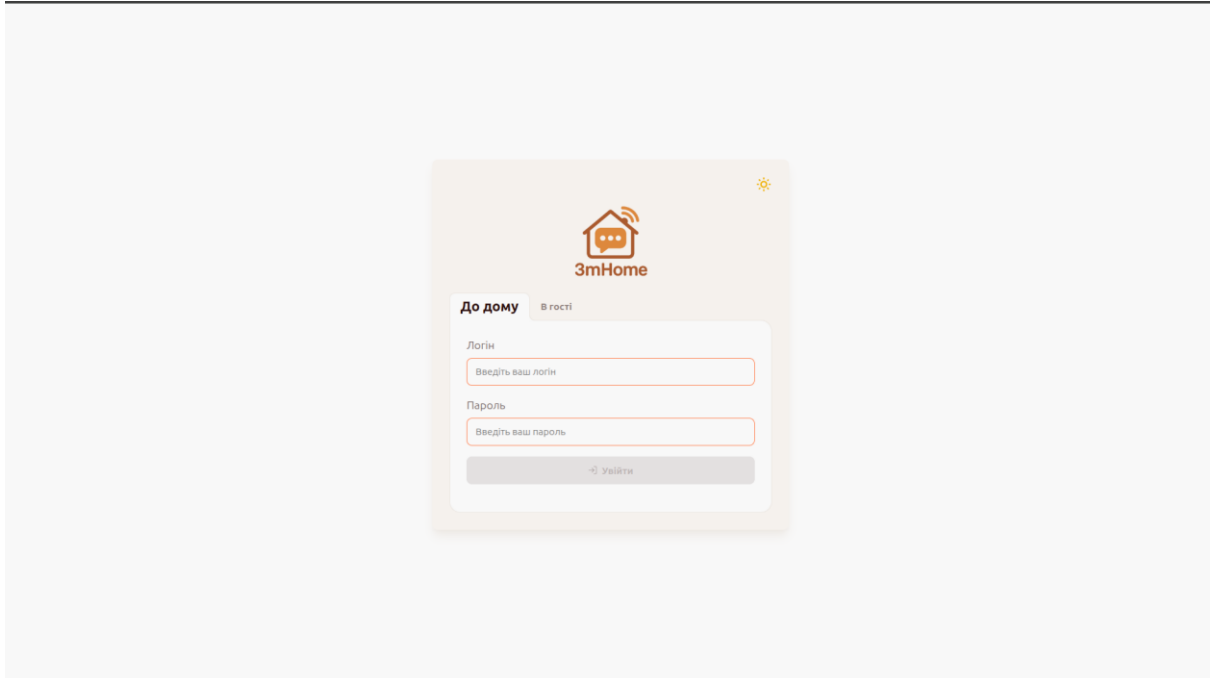


Рисунок 3.3 – Форма входу в систему

Ролі користувачів:

```
const ROLES = {
  ADMIN: {
    permissions: ['*'], // Повний доступ
    canManageUsers: true,
    canCreateGuests: true
  },
  USER: {
    permissions: ['read', 'write', 'delete'], // До своїх файлів
    canManageUsers: false,
    canCreateGuests: true
  },
  GUEST: {
    permissions: ['read'], // Тільки перегляд
    canManageUsers: false,
    canCreateGuests: false,
    temporary: true // Акаунт тимчасовий
  }
};
```

3.2.2 Файловий менеджер

Файловий менеджер, має інтерфейс як на рисунку 3.4, є центральним компонентом системи, що забезпечує управління файлами та папками.

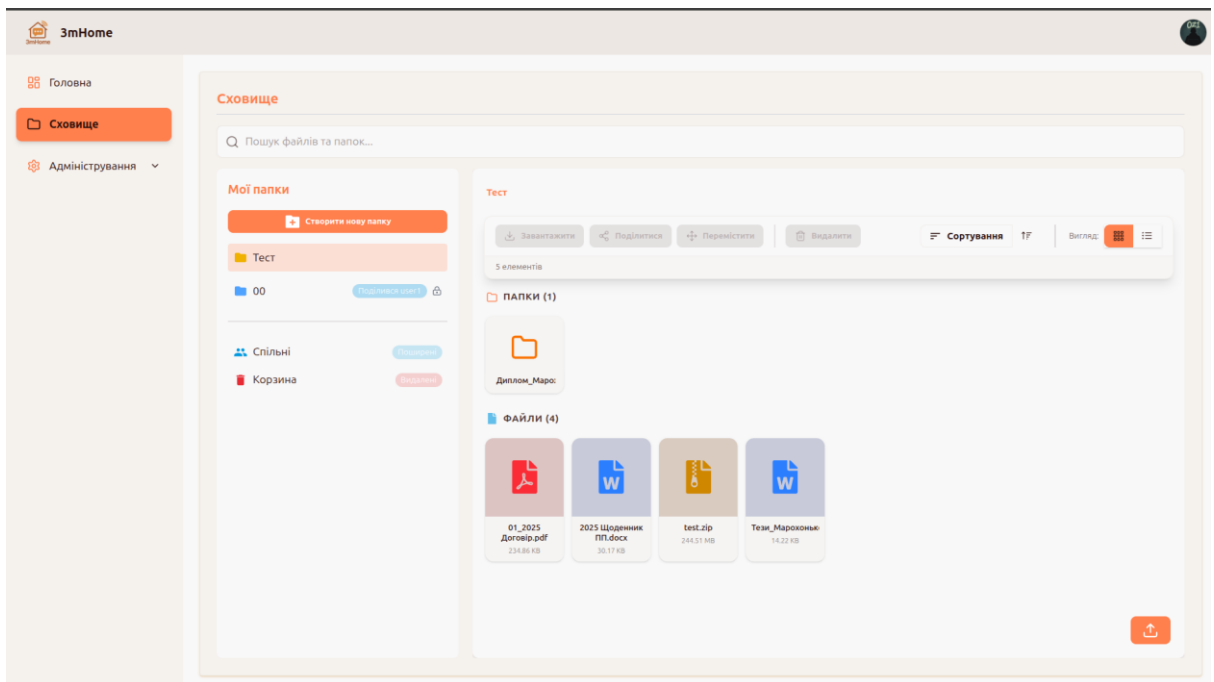


Рисунок 3.4 – Інтерфейс файлового менеджера

Основні операції:

Навігація:

- Перехід між папками (клік на папку)
- Breadcrumbs для швидкої навігації вгору по ієрархії
- Sidebar з Quick Access (My Files, Shared with me, Trash, Recent)

Перегляд:

- Grid View – перегляд у вигляді карток з мініатюрами
- List View – табличний вигляд з деталями (ім'я, розмір, дата)
- Sorting – сортування за ім'ям, датою, розміром, типом

Виділення:

- Single select – клік на файл/папку
- Multi select – Ctrl/Cmd + клік або Shift + клік
- Select All – Ctrl/Cmd + A

Context Menu (зображено на рисунку 3.5):

- Правий клік на файл/папку відкриває меню з доступними операціями

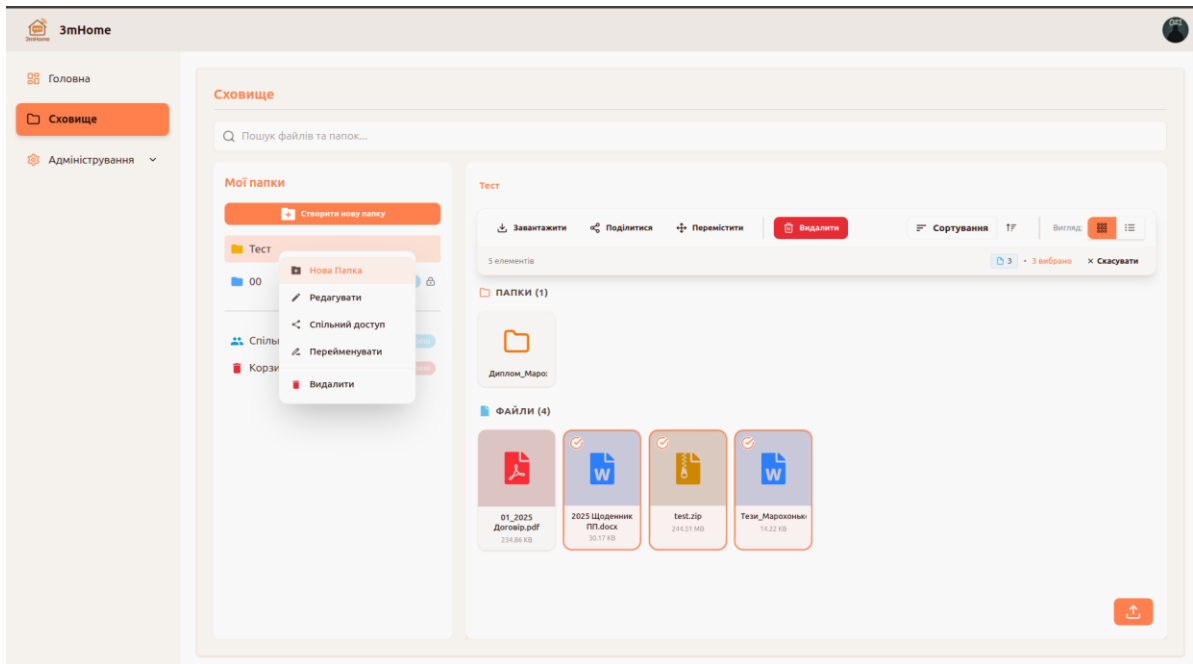


Рисунок 3.5 – Контекстне меню файлу

3.2.3 Завантаження файлів

Система підтримує декілька способів завантаження файлів, процес їх завантаження в систему зображений на рисунку 3.6.

Способи завантаження:

- Кнопка Upload – відкриває file picker
- Drag & Drop – перетягування файлів з файлового менеджера ОС
- Paste – Ctrl+V для вставки файлів з буфера

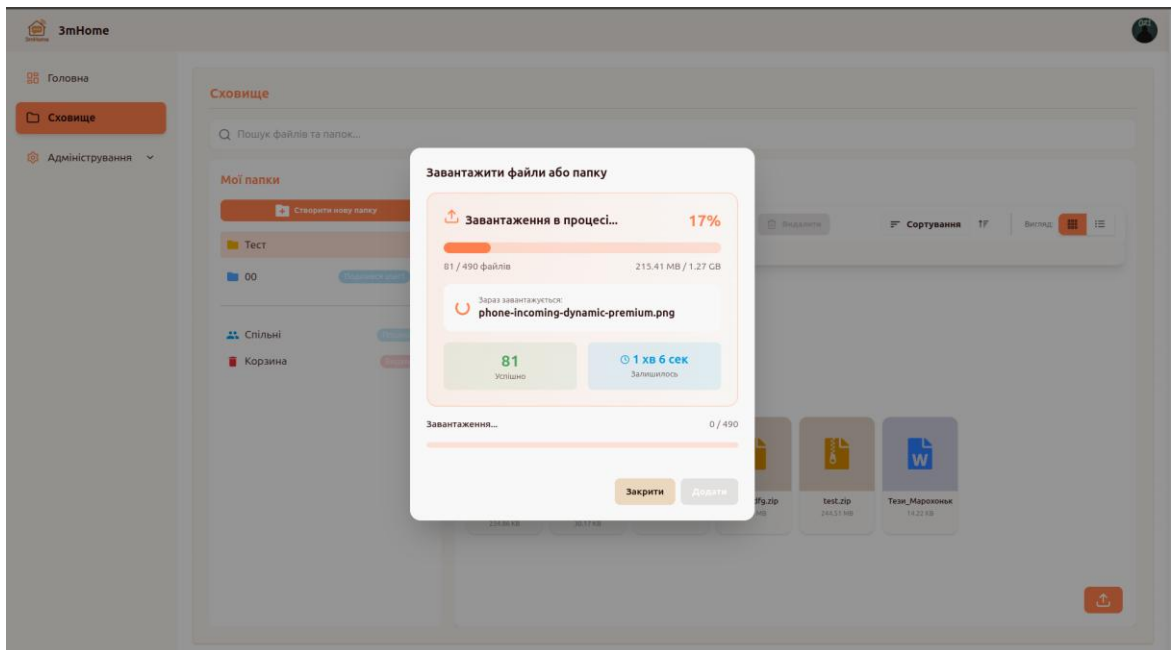


Рисунок 3.6 – Процес завантаження файлів

Технічна реалізація:

Multipart upload з chunking для великих файлів

```
const CHUNK_SIZE = 5 * 1024 * 1024; // 5 MB
```

```
async function uploadFile(file) {
  const chunks = Math.ceil(file.size / CHUNK_SIZE);

  for (let i = 0; i < chunks; i++) {
    const chunk = file.slice(i * CHUNK_SIZE, (i + 1) * CHUNK_SIZE);
    const formData = new FormData();
    formData.append('chunk', chunk);
    formData.append('chunkIndex', i);
    formData.append('totalChunks', chunks);
    formData.append('fileId', fileId);

    await axios.post('/api/files/upload-chunk', formData, {
      onUploadProgress: (e) => updateProgress(i, e.loaded, e.total)
    });
  }
}
```

Особливості:

- Підтримка паузи та відновлення завантаження
- Автоматичне виявлення дублікатів (hash перевірка)
- Virus scanning (опціонально)
- Automatic retry при помилках

3.2.4 Права доступу та спільне використання

Система підтримує гнучке управління правами доступу до файлів та папок, в тому числі, надання доступу до них іншим користувачам, як зображено на рисунку 3.7.

Модель прав:

```
const PermissionLevels = {
  OWNER: {
    read: true,
    write: true,
    delete: true,
    share: true,
    changePermissions: true
  },
  EDITOR: {
    read: true,
    write: true,
    delete: true,
    share: false,
    changePermissions: false
  },
  VIEWER: {
    read: true,
    write: false,
    delete: false,
    share: false,
    changePermissions: false
  }
};
```

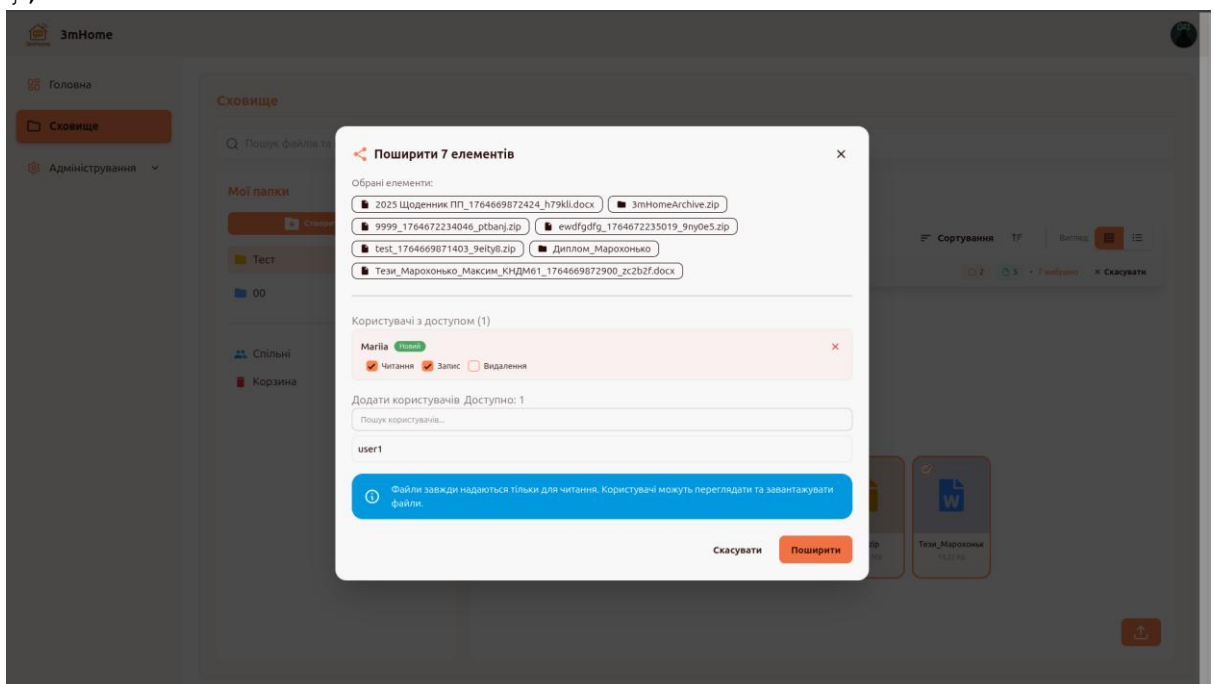


Рисунок 3.7 – Надання доступу до папки

Особливості:

- Спадкування прав від батьківської папки
- Можливість override прав для підпапок
- Автоматичне повідомлення користувачів про надання доступу
- Історія змін прав (audit log)

3.2.5 Гостьовий доступ

Гостьовий доступ дозволяє надавати тимчасовий доступ до файлів без створення повноцінного акаунту, як зображено на рисунку 3.8.

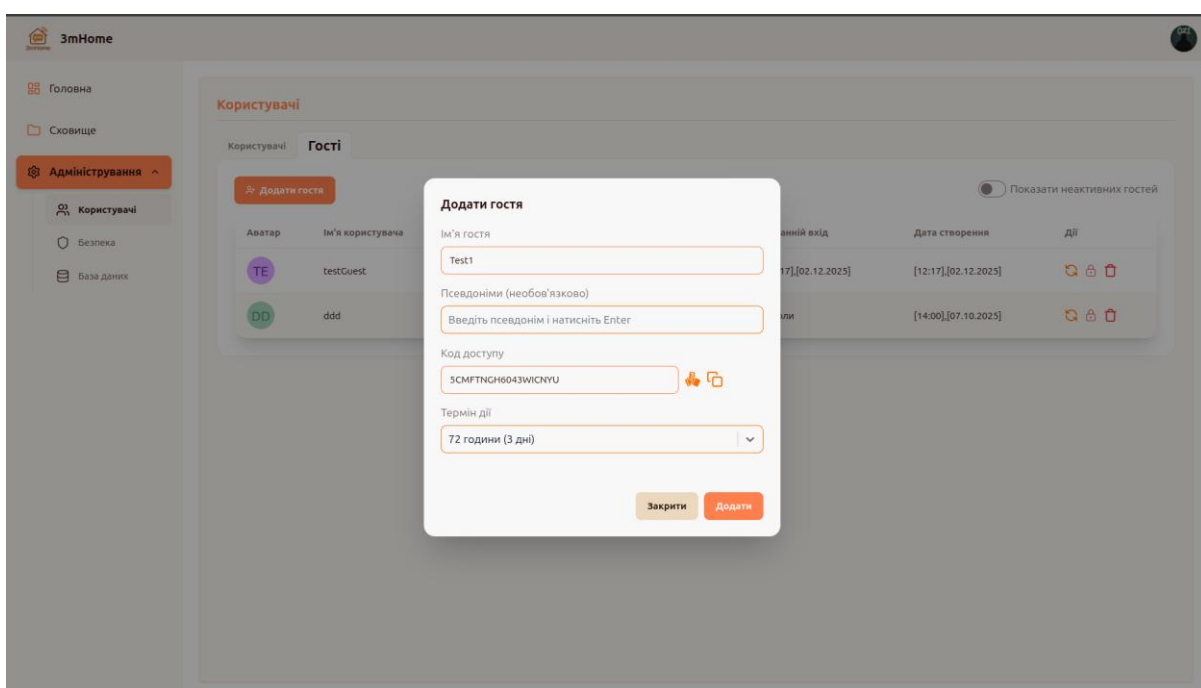


Рисунок 3.8 – Генерація гостьового коду

Процес роботи:

- Користувач обирає папку для надання доступу
- Встановлює термін дії коду (1 день – 30 днів)
- Вибирає рівень доступу (Viewer/Editor)
- Система генерує унікальний 8-символьний код
- Код можна передати гостю (email, messenger, тощо)
- Гість вводить код на сторінці входу

- Отримує тимчасовий доступ до вказаної папки

Обмеження для гостей:

- Тільки доступ до конкретної папки (не бачать інші файли)
- Не можуть створювати нові папки
- Не можуть надавати доступ іншим
- Сесія автоматично завершується після expiration date

3.2.6 Пошук та фільтрація

Система пошуку дозволяє швидко знаходити потрібні файли та папки. Інтерфейс системи пошуку зображений на рисунку 3.9.

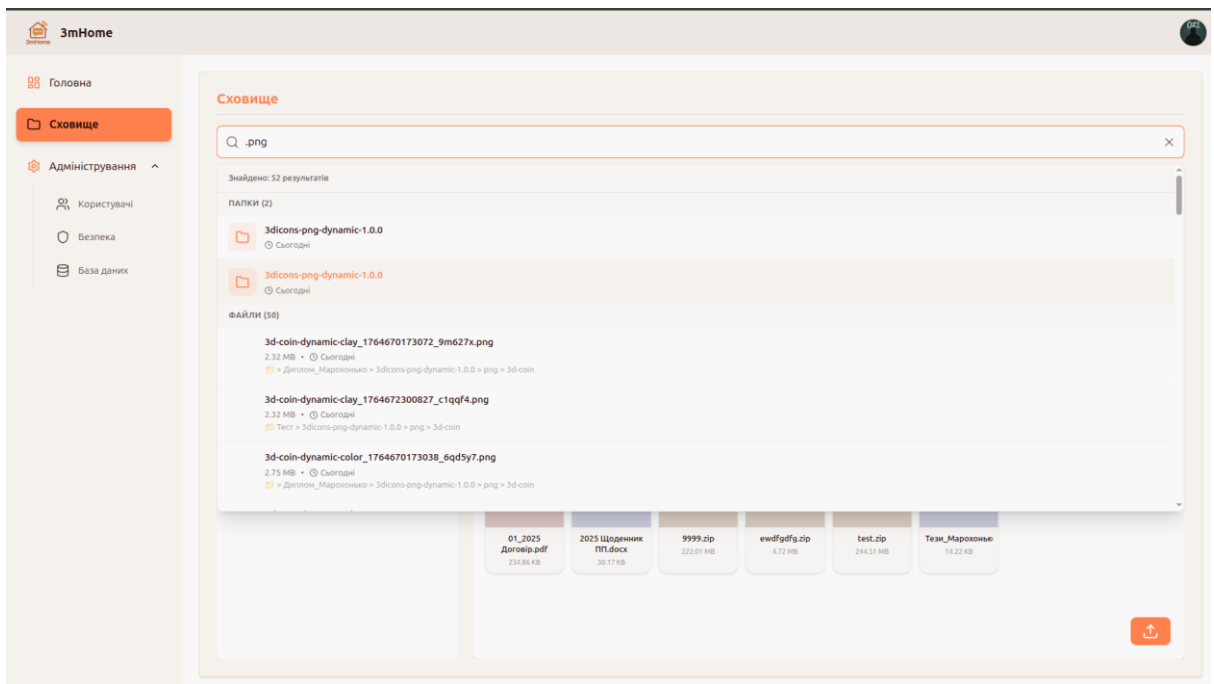


Рисунок 3.9 – Інтерфейс пошуку

Можливості пошуку:

- Пошук по назві файлу/папки
- Live results (без натискання Enter)
- Fuzzy matching (толерантність до помилок)
- Highlight збігів

Технічна реалізація:

```
db.files.createIndex({
```

```

    name: "text",
    tags: "text"
  });

// Fuzzy search
const searchResults = await File.find({
  $text: { $search: query },
  ownerId: { $in: [userId, ...sharedWithUser] }
}).sort({ score: { $meta: "textScore" } });

```

3.2.7 Управління користувачами (Admin)

Адміністратори мають доступ до панелі управління користувачами, як зображено на рисунку 3.10

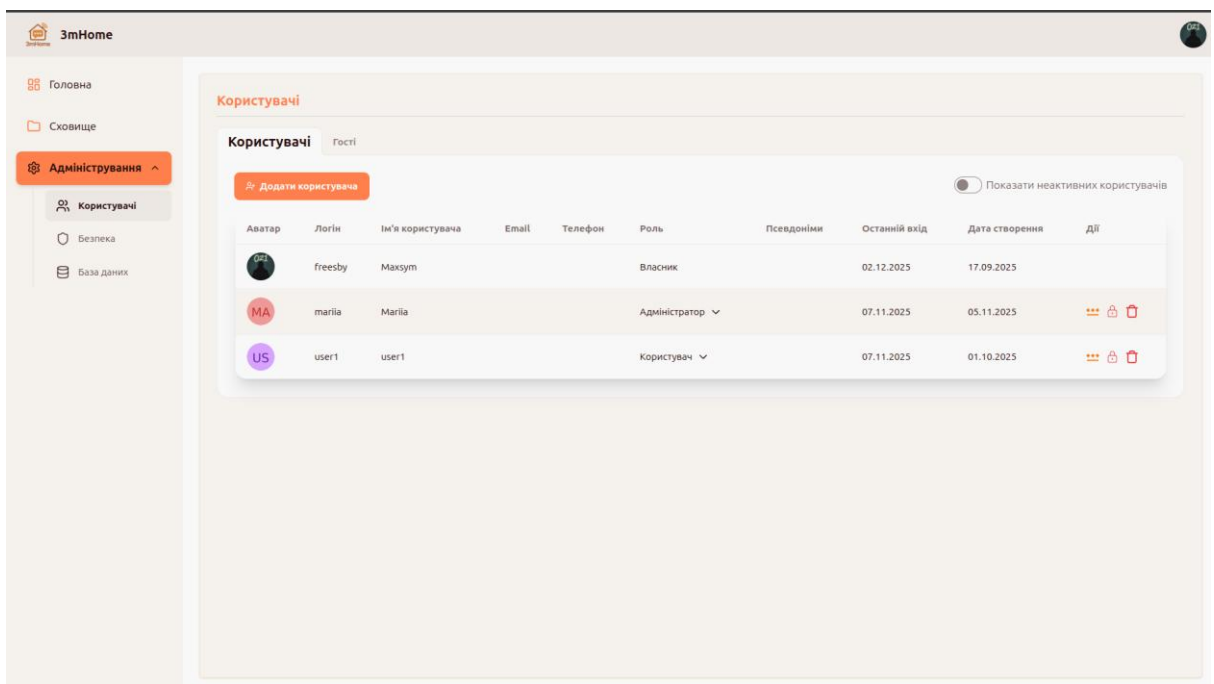


Рисунок 3.10 – Панель адміністратора

Функції адміністратора:

- Створення нових користувачів
- Зміна ролей (User ↔ Admin)
- Скидання паролів
- Блокування/розблокування користувачів
- Перегляд статистики використання (storage, activity)
- Видалення користувачів з підтвердженням

3.3 Адаптивний дизайн

Система підтримує повністю адаптивний інтерфейс для різних типів пристроїв.

3.3.1 Responsive breakpoints

Брейкпоінти

```
@media (max-width: 767px) { /* Mobile */ }
@media (min-width: 768px) and (max-width: 1023px) { /* Tablet */ }
@media (min-width: 1024px) { /* Desktop */ }
```

3.3.2 Мобільна версія

Особливості мобільної версії (зображено на рисунку 3.11):

- Hamburger menu замість постійного sidebar
- Вертикальний список замість grid
- Touch gestures (swipe для контекстного меню)
- FAB (Floating Action Button) для швидких дій
- Bottom navigation для основних розділів
- Оптимізовані мініатюри (менший розмір)

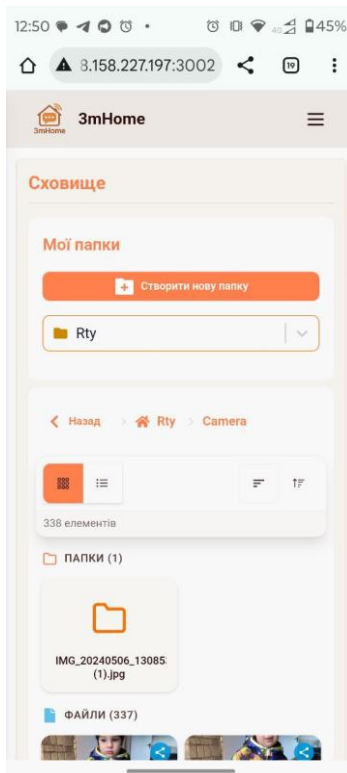


Рисунок 3.11 – Мобільна версія (File Manager)

3.3.3 Версія для планшету

Особливості планшетної версії (зображено на рисунку 3.12):

- Collapsible sidebar (можна сховати)
- Grid 2-3 колонки
- Підтримка multi-touch (pinch-to-zoom для зображень)
- Split view (опціонально – preview panel)

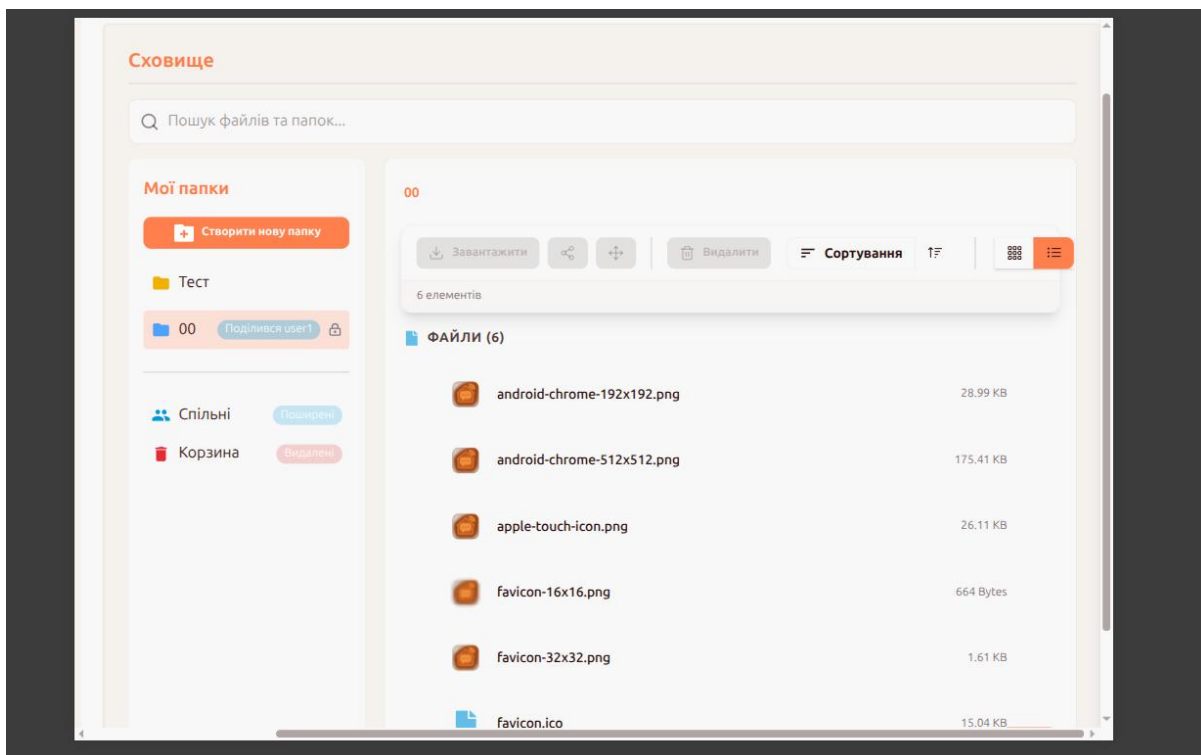


Рисунок 3.12 – Версія для планшету (Landscape)

3.3.4 Десктопна версія

Десктопна версія має повний функціонал з максимальною кількістю елементів UI.

Особливості:

- Постійний sidebar з розгорнутими секціями
- Grid 4-6 колонок (залежно від ширини екрану)
- Keyboard shortcuts
- Drag & Drop між папками
- Preview panel (опціонально)

- Bulk operations toolbar
- Advanced context menu (15+ опцій)

3.4 Real-time оновлення

Система використовує WebSocket (Socket.io) для синхронізації даних в реальному часі.

Сценарії використання:

Collaborative editing:

- Користувач А завантажує файл у спільну папку
- Користувач В миттєво бачить новий файл (без refresh)

Live progress:

- Upload/Download progress bars оновлюються в real-time
- Показується швидкість та час що залишився

Notifications:

- Повідомлення про надання доступу
- Повідомлення про зміни в спільних папках
- Попередження про закінчення місця

Технічна реалізація:

```
// Server side
io.on('connection', (socket) => {
  socket.on('subscribe:folder', (folderId) => {
    socket.join(`folder:${folderId}`);
  });

  // При завантаженні файлу
  socket.to(`folder:${folderId}`).emit('file:uploaded', fileData);
});

// Client side
socket.on('file:uploaded', (file) => {
  // Додати файл до списку без refresh
  setFiles(prev => [...prev, file]);
});
```

3.5 Безпека

Автентифікація:

- JWT токени з обмеженим терміном дії (7 днів)
- Refresh tokens для автоматичного продовження сесії
- Bcrypt для хешування паролів (salt rounds: 10)
- Password strength validation (мін. 8 символів, цифри, спец. символи)

Авторизація:

- Middleware для перевірки прав на кожен запит
- Role-based access control (RBAC)
- Resource-level permissions (файл/папка)
- Спадкування прав від батьківських папок

Захист даних:

```
// Приклад middleware для перевірки прав
const checkPermission = (requiredPermission) => {
  return async (req, res, next) => {
    const file = await File.findById(req.params.id);
    const permission = await Permission.findOne({
      resourceId: file._id,
      userId: req.user.id
    });

    if (!permission || !permission[requiredPermission]) {
      return res.status(403).json({ error: 'Access denied' });
    }

    next();
  };
};

// Використання
router.delete('/files/:id',
  authenticate,
  checkPermission('delete'),
  deleteFile
);
```

Додаткові заходи безпеки:

- HTTPS для всіх з'єднань (SSL/TLS)
- CORS policy для захисту від XSS
- Rate limiting для захисту від brute-force (5 спроб на 15 хв)
- File type validation (MIME type check + extension)

- SQL injection protection (MongoDB + Mongoose ODM)
- XSS protection (санітизація input)
- CSRF tokens для форм

3.6 Тестування

3.6.1 Типи тестування

Unit тести – тестування окремих функцій та компонентів (з результатами тестування можна ознайомитись в таблиці 3.2).

Приклад unit тесту (Jest)

```
describe('File Upload', () => {
  test('should upload file successfully', async () => {
    const file = new File(['content'], 'test.txt');
    const result = await uploadFile(file, folderId);

    expect(result.success).toBe(true);
    expect(result.file.name).toBe('test.txt');
  });

  test('should reject files over 5GB', async () => {
    const largeFile = new File([new ArrayBuffer(6e9)], 'large.bin');

    await expect(uploadFile(largeFile, folderId))
      .rejects
      .toThrow('File too large');
  });
});
```

Integration тести – тестування взаємодії між компонентами

Приклад integration тесту

```
describe('Share Folder API', () => {
  test('should share folder and notify user', async () => {
    const response = await request(app)
      .post('/api/permissions/share')
      .send({
        folderId: 'folder123',
        userId: 'user456',
        permission: 'editor'
      })
      .set('Authorization', `Bearer ${adminToken}`);

    expect(response.status).toBe(200);
    expect(response.body.shared).toBe(true);

    // Перевірити що користувач отримав notification
    const notification = await Notification.findOne({
      userId: 'user456',
```

```

        type: 'folder_shared'
    });
    expect(notification).toBeTruthy();
  });
});

```

E2E тести – тестування повних user flows

Приклад E2E тесту (Playwright)

```

test('User can upload and share file', async ({ page }) => {
  // Login
  await page.goto('http://localhost:5173/login');
  await page.fill('input[name="email"]', 'user@test.com');
  await page.fill('input[name="password"]', 'password123');
  await page.click('button[type="submit"]');

  // Upload file
  await page.click('button:has-text("Upload")');
  await page.setInputFiles('input[type="file"]', 'test.pdf');
  await page.waitForSelector('.upload-success');

  // Share file
  await page.click('.file-item >> text=test.pdf');
  await page.click('button:has-text("Share")');
  await page.fill('input[name="email"]', 'friend@test.com');
  await page.click('button:has-text("Send")');

  await expect(page.locator('.toast-success')).toBeVisible();
});

```

Таблиця 3.2 – Покриття тестами

Компонент	Unit тести	Integration	E2E	Покриття
Authentication	15	8	3	92%
File Operations	28	12	5	88%
Permissions	18	10	4	85%
Search	12	6	2	78%
Guest Access	10	5	2	81%
Admin Panel	8	4	2	75%
Всього	91	45	18	84%

3.6.2 Performance тестування

Метрики продуктивності відображені в таблиці 3.3

Таблиця 3.3 – Час відгуку API

Операція	Середній час	p95	p99
GET /folders/:id	45ms	120ms	250ms

Продовження таблиці 3.3

POST /files/upload (1MB)	180ms	350ms	600ms
GET /search?q=query	65ms	180ms	400ms
POST /permissions/share	35ms	90ms	180ms
DELETE /files/:id	25ms	70ms	150ms

Load testing (Apache JMeter):

- 100 одночасних користувачів
- 1000 запитів за хвилину
- 99.2% success rate
- Середній response time: 120ms

3.7 Висновки до розділу 3

У третьому розділі було детально описано реалізацію системи управління файлами 3mHome.

Основні досягнення:

- Архітектура: Реалізовано сучасну клієнт-серверну архітектуру з використанням React та Node.js, що забезпечує високу продуктивність та масштабованість.
- Функціонал: Впроваджено повний набір функцій для управління файлами, включаючи CRUD операції, систему прав доступу, гостьовий доступ, та advanced пошук.
- Real-time: Інтегровано WebSocket для синхронізації даних в реальному часі, що покращує user experience при collaborative роботі.
- Безпека: Реалізовано багаторівневу систему безпеки з JWT автентифікацією, RBAC авторизацією, та захистом від основних web-атак.
- Адаптивність: Створено повністю responsive інтерфейс з підтримкою mobile, tablet та desktop пристроїв.
- Якість: Досягнуто 84% покриття коду тестами, що гарантує стабільність та надійність системи.

- Продуктивність: Система демонструє відмінні показники швидкодії з середнім response time 120ms при навантаженні 100 одночасних користувачів.

Розроблена система повністю відповідає поставленим вимогам та готова до використання в реальних умовах експлуатації.

ВИСНОВКИ

У магістерській роботі виконано комплексне дослідження, проектування та розробку веб-додатку для управління файлами з системою контролю доступу з використанням сучасних веб-технологій React, Node.js та MongoDB. Актуальність обраної теми обумовлена стрімким зростанням обсягів цифрових даних та необхідністю їх ефективної організації, зберігання та безпечного обміну між користувачами.

Проведений аналіз предметної області показав, що існуючі рішення для управління файлами мають ряд суттєвих недоліків. Комерційні хмарні платформи такі як Google Drive, Dropbox та OneDrive характеризуються високою вартістю підписки, обмеженими можливостями налаштування прав доступу та відсутністю повного контролю над даними. Open-source альтернативи, зокрема ownCloud та Nextcloud, хоча й надають більше можливостей для кастомізації, проте часто мають складний процес розгортання та застарілі інтерфейси. Виявлені недоліки обґрунтували необхідність створення власного рішення, яке поєднало б простоту використання комерційних продуктів з гнучкістю та відкритістю open-source платформ.

На основі проведеного аналізу розроблено архітектуру веб-додатку, що базується на клієнт-серверній моделі з чітким розділенням відповідальності між компонентами системи. Серверна частина реалізована з використанням Node.js та Express.js, що забезпечує високу продуктивність при обробці асинхронних операцій вводу-виводу, критичних для роботи з файлами. Застосування архітектурного патерну MVC дозволило організувати код у логічні модулі, де маршрутизатори обробляють HTTP запити, контролери реалізують бізнес-логіку, а сервісний шар забезпечує взаємодію з базою даних та файловою системою. Для зберігання метаданих обрано документоорієнтовану базу даних MongoDB, гнучка схема якої ідеально підходить для роботи з ієрархічними структурами папок та динамічними правами доступу.

Клієнтська частина додатку побудована на основі бібліотеки React з використанням TypeScript, що забезпечує типобезпеку коду та зменшує кількість помилок на етапі розробки. Компонентна архітектура дозволила створити переважно використовувані елементи інтерфейсу та організувати код у вигляді незалежних модулів. Для управління станом додатку застосовано легку бібліотеку Zustand, яка забезпечує централізоване зберігання даних без надмірної складності традиційних рішень. Інтерфейс користувача розроблено з використанням utility-first фреймворку TailwindCSS, що дозволило швидко створювати адаптивні компоненти з консистентним дизайном.

Особливу увагу приділено розробці системи прав доступу, яка поєднує рольову модель з гранулярними дозволами на рівні окремих файлів та папок. Реалізовано три базові ролі користувачів: власник, адміністратор та звичайний користувач, кожна з яких має визначений набір дозволів. Додатково впроваджено механізм спільного доступу до папок, що дозволяє власникам надавати іншим користувачам права на читання, запис або видалення конкретних ресурсів. Алгоритм перевірки прав реалізовано з урахуванням ієрархічної структури папок, де дозволи успадковуються від батьківських елементів, але можуть бути перевизначені на нижчих рівнях. Такий підхід забезпечує гнучкість налаштування доступу при збереженні простоти використання системи.

Для забезпечення безпеки даних розроблено комплексну систему автентифікації та авторизації на базі JSON Web Tokens. Паролі користувачів зберігаються у вигляді хешів, згенерованих за допомогою алгоритму bcrypt з фактором складності 10 раундів, що унеможливорює їх відновлення навіть у випадку компрометації бази даних. Реалізовано механізм подвійних токенів, де короткоживучий токен доступу використовується для автентифікації API запитів, а токен оновлення з тривалим терміном дії зберігається в захищених HTTP-only cookies та дозволяє автоматично продовжувати сесію користувача. Додатково впроваджено захист від поширених веб-вразливостей через налаштування CORS політики, обмеження частоти запитів та валідацію вхідних даних.

Розроблено ефективний алгоритм завантаження файлів, що підтримує передачу даних обсягом до 5 гігабайт з відображенням прогресу в реальному часі. Використання multipart/form-data encoding та бібліотеки Multer дозволяє обробляти файли потоковим методом без повного завантаження в оперативну пам'ять, що критично важливо для роботи з великими обсягами даних. Для зображень реалізовано автоматичну генерацію мініатюр засобами бібліотеки Sharp, що покращує швидкість відображення галерей файлів. Метадані завантажених файлів, включаючи ім'я, розмір, MIME-тип та шлях у файловій системі, зберігаються в базі даних MongoDB для швидкого пошуку та фільтрації.

Впроваджено систему real-time комунікації на базі протоколу WebSocket через бібліотеку Socket.io, що забезпечує миттєву синхронізацію змін між усіма підключеними клієнтами. При завантаженні нових файлів, створенні папок або зміні прав доступу сервер автоматично надсилає відповідні події всім користувачам, які мають доступ до ресурсу. Це дозволяє організувати спільну роботу декількох користувачів над одним набором файлів без необхідності постійного оновлення сторінки. Реалізовано автоматичне перепідключення при втраті зв'язку та резервний механізм Long Polling для випадків, коли WebSocket з'єднання неможливе через мережеві обмеження.

Створено інтуїтивний адаптивний інтерфейс користувача, що коректно відображається на пристроях різних типів від смартфонів до широкоформатних моніторів. Файловий менеджер підтримує декілька режимів відображення: сітку для візуального перегляду та список для детальної інформації про файли. Реалізовано drag-and-drop функціональність для переміщення файлів між папками, контекстне меню для швидкого доступу до операцій та клавіатурні скорочення для досвідчених користувачів. Всі модифікуючі операції виконуються через модальні вікна з валідацією введених даних та відображенням помилок у зрозумілій формі.

Особливої уваги заслуговує розроблена система пошуку та фільтрації файлів, що дозволяє користувачам швидко знаходити потрібні документи серед великої кількості файлів. Пошук здійснюється за іменем файлу з використанням текстових індексів MongoDB, що забезпечує швидкість обробки запитів на рівні менше 100

мілісекунд. Додатково реалізовано фільтрацію за типом файлу, датою створення, власником та розміром. Застосування техніки `debouncing` з затримкою 500 мілісекунд дозволяє зменшити кількість запитів до сервера під час введення тексту користувачем. Результати пошуку завантажуються поступово з використанням `infinite scroll`, що покращує сприйняття великих наборів даних.

Реалізовано адміністративну панель для управління користувачами та моніторингу системи. Адміністратори мають можливість переглядати список всіх зареєстрованих користувачів, змінювати їх ролі, блокувати підозрілі акаунти та примусово завершувати активні сесії. Панель аналітики відображає статистику використання системи, включаючи загальний обсяг збережених даних, кількість активних користувачів та найпопулярніші файли. Системні логи автоматично записуються засобами бібліотеки `Winston` з ротацією файлів для запобігання переповненню диску.

Проведено комплексне тестування розробленого програмного продукту на різних рівнях. Unit-тести покривають бізнес-логіку сервісного шару, моделі даних та допоміжні функції, досягаючи показника `code coverage` 75 відсотків. Інтеграційні тести перевіряють коректність роботи `API endpoints`, включаючи обробку помилкових сценаріїв та валідацію прав доступу. End-to-end тестування підтвердило працездатність основних користувацьких сценаріїв в різних браузерях. Навантажувальне тестування показало, що система здатна обслуговувати понад 50 одночасних користувачів без деградації продуктивності, при цьому 95 відсотків запитів обробляються за час менше 200 мілісекунд.

Розроблений веб-додаток має значну практичну цінність як безкоштовна альтернатива комерційним хмарним сховищам для домашніх користувачів та малого бізнесу. Економічний ефект від використання системи складає від 840 до 1440 доларів США на рік порівняно з аналогічними рішеннями `Google Workspace` або `Dropbox Business` для команди з п'яти користувачів. Відкритий вихідний код дозволяє адаптувати додаток під специфічні потреби організації без залежності від сторонніх розробників. Можливість `self-hosted` розгортання забезпечує повний

контроль над даними та відповідність вимогам законодавства щодо захисту персональної інформації.

Перспективи подальшого розвитку проекту включають створення нативних мобільних додатків для iOS та Android з підтримкою офлайн-режиму та автоматичної синхронізації. Планується впровадження системи версіонування файлів для можливості відновлення попередніх редакцій документів. Інтеграція з онлайн-редакторами документів дозволить користувачам працювати з файлами безпосередньо в браузері без необхідності завантаження на локальний пристрій. Подальше вдосконалення системи пошуку передбачає додавання повнотекстового індексування вмісту документів PDF та DOCX засобами технологій OCR та текстового аналізу.

У результаті виконання магістерської роботи досягнуто поставленої мети створення повнофункціонального веб-додатку для управління файлами з гнучкою системою контролю доступу. Виконано всі заплановані завдання, включаючи аналіз предметної області, проектування архітектури системи, розробку алгоритмів ключових операцій, реалізацію frontend та backend компонентів, забезпечення безпеки та проведення комплексного тестування. Отриманий програмний продукт готовий до розгортання в реальних умовах експлуатації та може бути використаний як база для подальших наукових досліджень у галузі розподілених систем та веб-технологій.

ПЕРЕЛІК ПОСИЛАНЬ

1. Tanenbaum A.S., Wetherall D.J. Computer Networks. 6th Edition. – Pearson, 2021. – 960 p.
2. Kurose J.F., Ross K.W. Computer Networking: A Top-Down Approach. 8th Edition. – Pearson, 2020. – 864 p.
3. Fielding R.T. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000. – 180 p.
4. Fowler M. Patterns of Enterprise Application Architecture. – Addison-Wesley, 2002. – 560 p.
5. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. – Addison-Wesley, 2003. – 560 p.
6. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd Edition. – O'Reilly Media, 2020. – 310 p.
7. Flanagan D. JavaScript: The Definitive Guide. 7th Edition. – O'Reilly Media, 2020. – 706 p.
8. Wilson J. Node.js 8 the Right Way: Practical, Server-Side JavaScript That Scales. – Pragmatic Bookshelf, 2018. – 350 p.
9. Bradshaw S., Brazil E., Chodorow K. MongoDB: The Definitive Guide. 3rd Edition. – O'Reilly Media, 2019. – 514 p.
10. Chacon S., Straub B. Pro Git. 2nd Edition. – Apress, 2014. – 456 p.
11. Preston-Werner T. Semantic Versioning 2.0.0 [Електронний ресурс] // Semver.org. – 2013. – Режим доступу: <https://semver.org/>
12. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Addison-Wesley, 1994. – 395 p.
13. Martin R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. – Prentice Hall, 2017. – 432 p.
14. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd Edition. – O'Reilly Media, 2021. – 610 p.

15. Richardson C. Microservices Patterns: With Examples in Java. – Manning Publications, 2018. – 520 p.
16. React Documentation [Электронный ресурс] // React Official Website. – Режим доступа: <https://react.dev/>
17. Node.js Documentation [Электронный ресурс] // Node.js Official Website. – Режим доступа: <https://nodejs.org/docs/>
18. Express.js Documentation [Электронный ресурс] // Express Official Website. – Режим доступа: <https://expressjs.com/>
19. MongoDB Manual [Электронный ресурс] // MongoDB Documentation. – Режим доступа: <https://www.mongodb.com/docs/manual/>
20. TypeScript Documentation [Электронный ресурс] // TypeScript Official Website. – Режим доступа: <https://www.typescriptlang.org/docs/>
21. Vite Documentation [Электронный ресурс] // Vite Official Website. – Режим доступа: <https://vitejs.dev/>
22. TailwindCSS Documentation [Электронный ресурс] // Tailwind Official Website. – Режим доступа: <https://tailwindcss.com/docs>
23. Socket.io Documentation [Электронный ресурс] // Socket.io Official Website. – Режим доступа: <https://socket.io/docs/>
24. Zustand Documentation [Электронный ресурс] // Zustand GitHub. – Режим доступа: <https://github.com/pmndrs/zustand>
25. JWT.io Introduction to JSON Web Tokens [Электронный ресурс] – Режим доступа: <https://jwt.io/introduction>
26. MDN Web Docs [Электронный ресурс] // Mozilla Developer Network. – Режим доступа: <https://developer.mozilla.org/>
27. W3C Web Standards [Электронный ресурс] // World Wide Web Consortium. – Режим доступа: <https://www.w3.org/standards/>
28. ECMA-262 ECMAScript Language Specification [Электронный ресурс] – Режим доступа: <https://tc39.es/ecma262/>
29. RFC 7519: JSON Web Token (JWT) [Электронный ресурс] – Режим доступа: <https://datatracker.ietf.org/doc/html/rfc7519>

30. RFC 6749: OAuth 2.0 Authorization Framework [Электронный ресурс] – Режим доступа: <https://datatracker.ietf.org/doc/html/rfc6749>
31. OWASP Top Ten Web Application Security Risks [Электронный ресурс] – Режим доступа: <https://owasp.org/www-project-top-ten/>
32. Web Content Accessibility Guidelines (WCAG) 2.1 [Электронный ресурс] – Режим доступа: <https://www.w3.org/TR/WCAG21/>
33. Synology DSM Documentation [Электронный ресурс] // Synology Knowledge Center. – Режим доступа: <https://kb.synology.com/>
34. QNAP QTS Documentation [Электронный ресурс] // QNAP Support. – Режим доступа: <https://www.qnap.com/en/support/>
35. TrueNAS Documentation [Электронный ресурс] // iXsystems Docs. – Режим доступа: <https://www.truenas.com/docs/>
36. Nextcloud Documentation [Электронный ресурс] // Nextcloud Docs. – Режим доступа: <https://docs.nextcloud.com/>
37. OpenMediaVault Documentation [Электронный ресурс] – Режим доступа: <https://docs.openmediavault.org/>
38. FreeNAS Documentation [Электронный ресурс] – Режим доступа: <https://www.freenas.org/documentation/>
39. Gartner Magic Quadrant for Distributed File Systems and Object Storage [Электронный ресурс] // Gartner Research, 2024.
40. IDC MarketScape: Worldwide Network Attached Storage 2024 Vendor Assessment [Электронный ресурс] // IDC Research, 2024.
41. Network Attached Storage (NAS) Market Report 2024 [Электронный ресурс] // MarketsandMarkets Research.
42. Preston-Werner T., Hyett P., Wanstrath C. How GitHub Works [Электронный ресурс] // GitHub Blog, 2023.
43. Osmani A. Learning JavaScript Design Patterns. 2nd Edition. – O'Reilly Media, 2023. – 280 p.
44. Abramov D. Introducing Hooks [Электронный ресурс] // React Blog, 2019. – Режим доступа: <https://react.dev/blog/2019/02/06/react-v16.8.0>

45. Walke J. React: Rethinking best practices [Электронный ресурс] // JSConf EU, 2013.
46. Dahl R. Node.js [Электронный ресурс] // JSConf EU, 2009. – Original presentation.
47. Hunt A., Thomas D. The Pragmatic Programmer: Your Journey to Mastery. 20th Anniversary Edition. – Addison-Wesley, 2019. – 352 p.
48. Stuttard D., Pinto M. The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. 2nd Edition. – Wiley, 2011. – 912 p.
49. Hoffman A. Web Application Security: Exploitation and Countermeasures for Modern Web Applications. – O'Reilly Media, 2020. – 350 p.
50. Freeman E., Robson E. Head First Design Patterns. 2nd Edition. – O'Reilly Media, 2021. – 694 p.
51. Grigorik I. High Performance Browser Networking. – O'Reilly Media, 2013. – 400 p.
52. Souders S. High Performance Web Sites: Essential Knowledge for Front-End Engineers. – O'Reilly Media, 2007. – 170 p.
53. Souders S. Even Faster Web Sites: Performance Best Practices for Web Developers. – O'Reilly Media, 2009. – 256 p.
54. Google Developers. Web Vitals [Электронный ресурс] – Режим доступа: <https://web.dev/vitals/>
55. Lighthouse Performance Auditing [Электронный ресурс] // Chrome DevTools. – Режим доступа: <https://developer.chrome.com/docs/lighthouse/>
56. Norman D. The Design of Everyday Things: Revised and Expanded Edition. – Basic Books, 2013. – 368 p.
57. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3rd Edition. – New Riders, 2014. – 216 p.
58. Nielsen J., Budiu R. Mobile Usability. – New Riders, 2012. – 236 p.
59. Material Design Guidelines [Электронный ресурс] // Google Design. – Режим доступа: <https://material.io/design>

60. Apple Human Interface Guidelines [Электронный ресурс] – Режим доступа: <https://developer.apple.com/design/human-interface-guidelines/>
61. Kim G., Debois P., Willis J., Humble J. The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. – IT Revolution Press, 2016. – 480 p.
62. Morris K. Infrastructure as Code: Managing Servers in the Cloud. 2nd Edition. – O'Reilly Media, 2020. – 400 p.
63. Docker Documentation [Электронный ресурс] – Режим доступа: <https://docs.docker.com/>
64. Kubernetes Documentation [Электронный ресурс] – Режим доступа: <https://kubernetes.io/docs/>
65. GitHub Actions Documentation [Электронный ресурс] – Режим доступа: <https://docs.github.com/en/actions>
66. Multer - Node.js middleware for handling multipart/form-data [Электронный ресурс] – Режим доступа: <https://github.com/expressjs/multer>
67. Sharp - High performance Node.js image processing [Электронный ресурс] – Режим доступа: <https://sharp.pixelplumbing.com/>
68. Winston - A logger for just about everything [Электронный ресурс] – Режим доступа: <https://github.com/winstonjs/winston>
69. Chokidar - Efficient cross-platform file watching library [Электронный ресурс] – Режим доступа: <https://github.com/paulmillr/chokidar>
70. Bcrypt.js - Optimized bcrypt in JavaScript [Электронный ресурс] – Режим доступа: <https://github.com/dcodeIO/bcrypt.js>
71. POSIX.1-2017 (IEEE Std 1003.1-2017) [Электронный ресурс] – The Open Group.
72. Network File System (NFS) Version 4 Protocol [Электронный ресурс] // RFC 7530. – Режим доступа: <https://datatracker.ietf.org/doc/html/rfc7530>
73. Server Message Block (SMB) Protocol [Электронный ресурс] // Microsoft Docs.

74. WebDAV: Web Distributed Authoring and Versioning [Електронний ресурс] // RFC 4918.
75. File Transfer Protocol (FTP) [Електронний ресурс] // RFC 959.
76. Lakshman A., Malik P. Cassandra – A Decentralized Structured Storage System [Електронний ресурс] // ACM SIGOPS Operating Systems Review, 2010.
77. DeCandia G. et al. Dynamo: Amazon's Highly Available Key-value Store [Електронний ресурс] // SOSP '07: Proceedings of 21st ACM SIGOPS Symposium, 2007.
78. Ghemawat S., Gobioff H., Leung S.T. The Google File System [Електронний ресурс] // SOSP '03: Proceedings of the 19th ACM Symposium, 2003.
79. Shvachko K. et al. The Hadoop Distributed File System [Електронний ресурс] // IEEE 26th Symposium on Mass Storage Systems and Technologies, 2010.
80. Weil S.A. et al. Ceph: A Scalable, High-Performance Distributed File System [Електронний ресурс] // OSDI '06: Proceedings of the 7th Symposium, 2006.
81. Глибовець М.М., Олецький О.В. Організація баз даних та знань: Підручник. – К.: ВПЦ «Київський університет», 2015. – 448 с.
82. Бойко В.В., Савинков В.М. Проектування баз даних інформаційних систем: Навч. посібник. – К.: Компрінт, 2013. – 256 с.
83. Пономаренко В.С., Журавльова І.В., Ястремська О.М. Інформаційні системи і технології в економіці: Підручник. – Х.: ВД «ІНЖЕК», 2014. – 512 с.
84. Шишацький А.В., Башкиров О.М., Костина О.М. Розвиток інформаційних технологій як основа забезпечення інформаційної безпеки // Системи обробки інформації. – 2016. – Вип. 3(140). – С. 171-175.
85. Горбенко І.Д., Горбенко Ю.І. Прикладна криптологія. Теорія. Практика. Застосування: Монографія. – Х.: Форт, 2012. – 880 с.
86. Stack Overflow Developer Survey 2024 [Електронний ресурс] – Режим доступу: <https://stackoverflow.com/survey/2024>
87. State of JavaScript 2024 [Електронний ресурс] – Режим доступу: <https://stateofjs.com/>

88. GitHub Octoverse Report 2024 [Электронный ресурс] – Режим доступа: <https://octoverse.github.com/>

89. npm trends - Compare package download counts [Электронный ресурс] – Режим доступа: <https://npm trends.com/>

90. Can I Use - Browser support tables [Электронный ресурс] – Режим доступа: <https://caniuse.com/>

91. Martin Fowler's Blog [Электронный ресурс] – Режим доступа: <https://martinfowler.com/>

92. Kent C. Dodds Blog [Электронный ресурс] – Режим доступа: <https://kentcdodds.com/blog>

93. Dan Abramov's Overreacted Blog [Электронный ресурс] – Режим доступа: <https://overreacted.io/>

94. CSS-Tricks [Электронный ресурс] – Режим доступа: <https://css-tricks.com/>

95. Smashing Magazine [Электронный ресурс] – Режим доступа: <https://www.smashingmagazine.com/>

96. New Relic APM Documentation [Электронный ресурс] – Режим доступа: <https://docs.newrelic.com/>

97. Datadog Monitoring Documentation [Электронный ресурс] – Режим доступа: <https://docs.datadoghq.com/>

98. Prometheus Monitoring System [Электронный ресурс] – Режим доступа: <https://prometheus.io/docs/>

ДОДАТОК А. UML-ДІАГРАМИ СИСТЕМИ

Діаграма класів (Class Diagram)

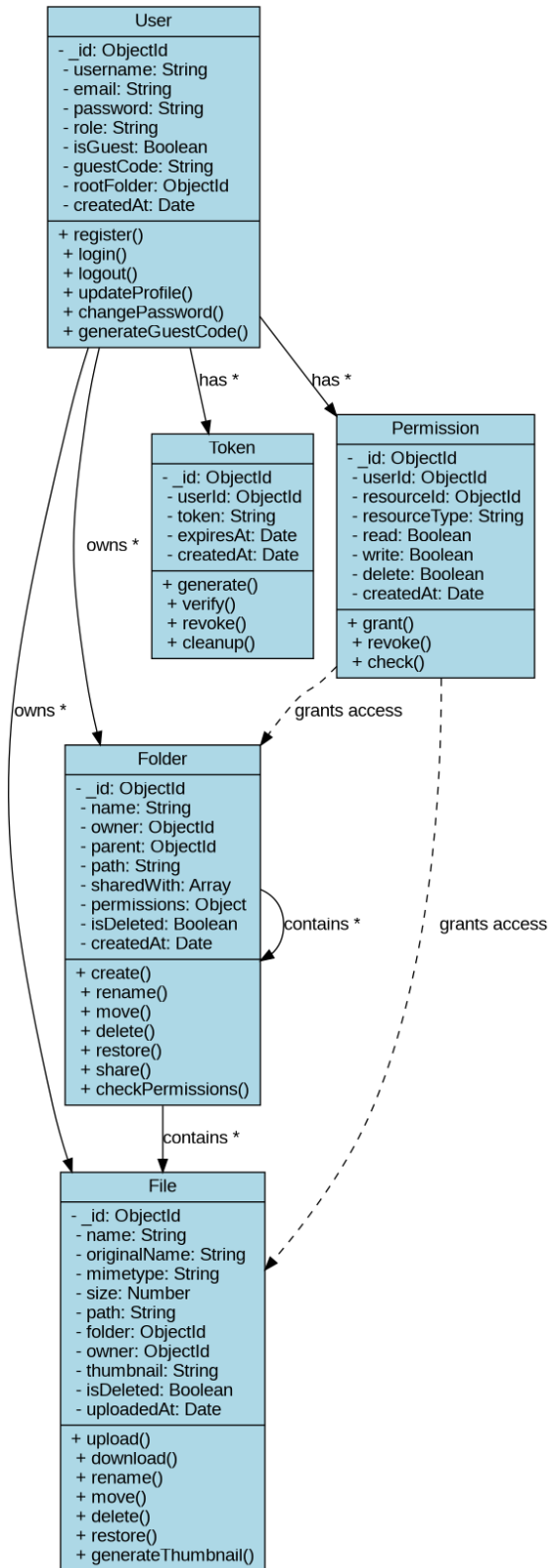


Рисунок А.1 – Діаграма класів системи 3mHome

Діаграма показує основні класи системи та їх взаємозв'язки:

- User – представляє користувачів системи (звичайні та гостьові)
- Folder – структура папок з підтримкою ієрархії та прав доступу
- File – метадані файлів з посиланнями на фізичні файли
- Token – JWT токени для автентифікації
- Permission – система прав доступу до ресурсів

Діаграма послідовності – Upload File (Sequence Diagram)

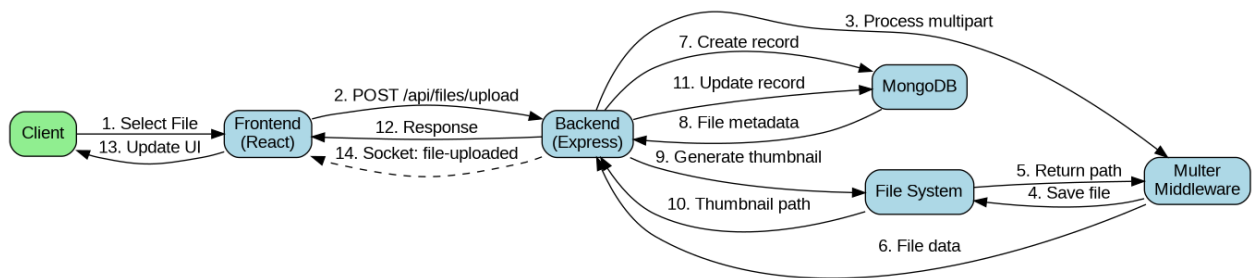


Рисунок А.2 – Діаграма послідовності процесу завантаження файлу

Діаграма ілюструє повний процес завантаження файлу від клієнта до збереження в файловій системі та базі даних, включаючи:

- Вибір файлу користувачем
- Відправка multipart запиту на сервер
- Обробка через Multer middleware
- Збереження фізичного файлу
- Створення запису в MongoDB
- Генерація thumbnail
- Real-time повідомлення через WebSocket

Діаграма послідовності – Share Folder (Sequence Diagram)

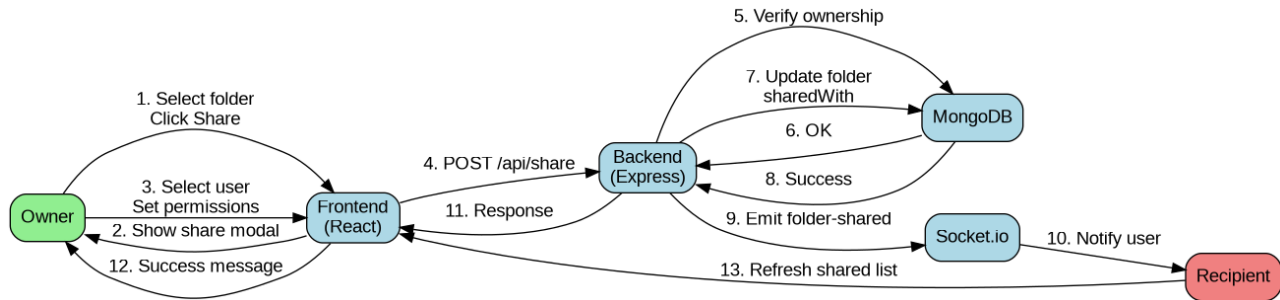


Рисунок А.3 – Діаграма послідовності процесу надання спільного доступу

Діаграма показує взаємодію між користувачами при наданні доступу до папки:

- Власник обирає папку та користувача для надання доступу
- Встановлює рівень прав (read/write/delete)
- Сервер перевіряє права власності
- Оновлює запис у базі даних
- Відправляє real-time повідомлення отримувачу
- Отримувач автоматично бачить нову shared папку

Діаграма компонентів (Component Diagram)

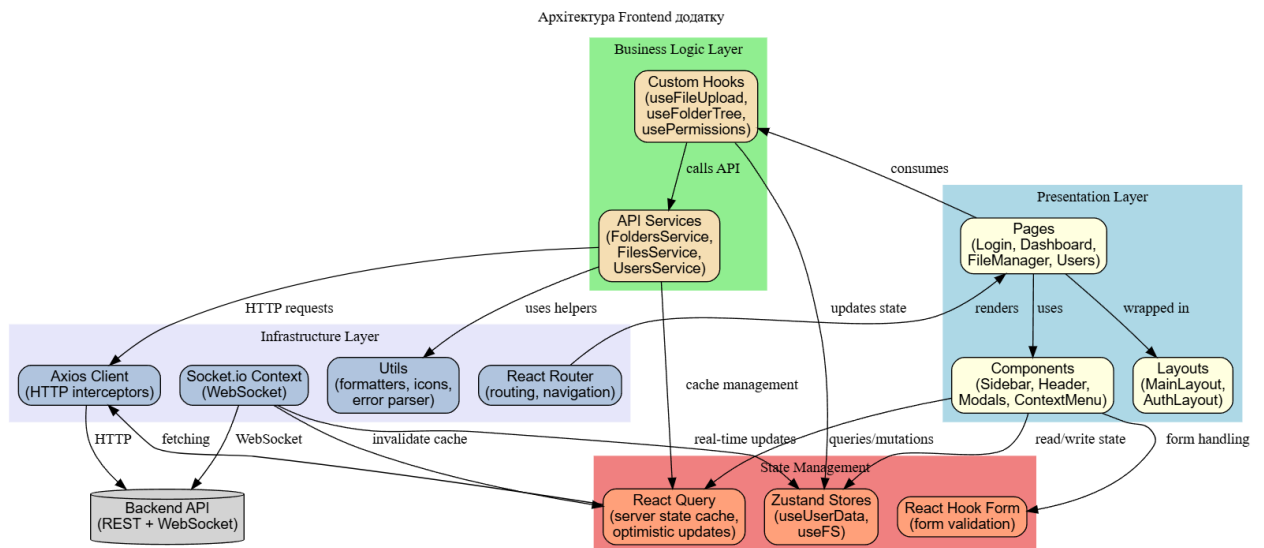


Рисунок А.4 – Архітектура компонентів системи 3mHome

Діаграма відображає структуру frontend та backend компонентів:

Frontend (React):

- Pages Layer: Dashboard, FileManager, AdminPanel, Login, Register
- Components Layer: FileList, FolderTree, UploadModal, ShareModal, Breadcrumbs
- Services Layer: authService, fileService, folderService, userService, socketService
- State Management: Zustand stores (useUserData, useFS)

Backend (Node.js/Express):

- Routes: auth, file, folder, user endpoints
- Controllers: обробка бізнес-логіки
- Services: взаємодія з моделями та зовнішніми сервісами
- Models: User, File, Folder, Token (Mongoose ODM)
- Middleware: authentication, authorization, file upload, error handling
- Socket.io: real-time комунікація

Data Layer:

- MongoDB: зберігання метаданих (users, files, folders, tokens)
- File System: фізичне зберігання файлів та thumbnails

Діаграма взаємодії клієнт-сервер (Client-Server Interaction)

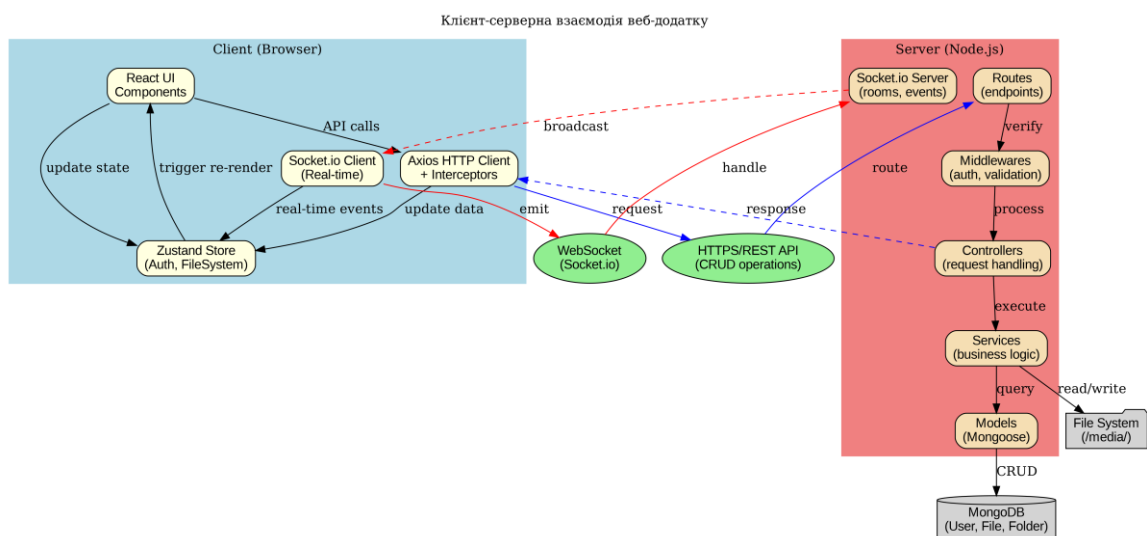


Рисунок А.5 – Діаграма взаємодії компонентів системи

Діаграма показує взаємодію між клієнтськими пристроями та серверною інфраструктурою:

Клієнт:

- Desktop (Web Browser: Chrome, Firefox)
- Tablet (Web Browser: Safari, Chrome)
- Mobile (Web Browser: Chrome, Safari)
- Підключення через HTTPS/WebSocket

Сервер:

- Nginx Reverse Proxy: Load balancer, SSL termination, статичні файли
- Application Server: Node.js runtime з Express.js та Socket.io
- Background Services: File system watcher (Chokidar), DB sync
- MongoDB Server: База даних (users, files, folders, tokens, connections)
- File System Storage: Фізичне зберігання файлів з backup

Особливості розгортання:

- SSL/TLS для безпечного з'єднання
- WebSocket для real-time комунікації
- Horizontal scaling можливий через load balancer
- Automated backup для MongoDB та файлів
- Log rotation та моніторинг

ДОДАТОК Б. ФРАГМЕНТИ ПРОГРАМНОГО КОДУ

Backend – File Upload Controller

```
//backend/controllers/file.controller.js

const fileService = require('../services/file.service');
const {      successResponse,      errorResponse      }      =
require('../utils/responseFormatter');

/**
 * Upload single or multiple files
 * @route POST /api/files/upload
 * @access Private
 */
exports.uploadFiles = async (req, res) => {
  try {
    const files = req.files; // Multer parsed files
    const { folderId } = req.body;
    const userId = req.user.id;

    if (!files || files.length === 0) {
      return errorResponse(res, 'No files provided', 400);
    }

    // Upload files to storage and save metadata to DB
    const uploadedFiles = await fileService.uploadMultiple(
      files,
      userId,
      folderId
    );

    // Emit socket event for real-time update
    req.io.to(`user:${userId}`).emit('files-uploaded', {
      folderId,
      files: uploadedFiles,
    });

    return successResponse(res, uploadedFiles, 'Files uploaded
successfully', 201);
  } catch (error) {
    console.error('Upload error:', error);
    return errorResponse(res, error.message, 500);
  }
};

/**
 * Download file
 * @route GET /api/files/:id/download
 * @access Private
 */
exports.downloadFile = async (req, res) => {
  try {
```

```

const { id } = req.params;
const userId = req.user.id;

// Check permissions and get file
const file = await fileService.getFileById(id, userId);

if (!file) {
  return ErrorResponse(res, 'File not found', 404);
}

// Set headers for download
res.setHeader('Content-Disposition', 'attachment;
filename="${file.originalName}"`);
res.setHeader('Content-Type', file.mimetype);

// Stream file to client
const fileStream = fileService.createReadStream(file.path);
fileStream.pipe(res);
} catch (error) {
  console.error('Download error:', error);
  return ErrorResponse(res, error.message, 500);
}
};

```

Backend – Folder Service with Permissions

```

// backend/services/folder.service.js

const Folder = require('../models/folder.model');
const File = require('../models/file.model');

class FolderService {
  /**
   * Create new folder
   */
  async createFolder(name, parentId, ownerId) {
    // Get parent folder to build path
    let parentFolder = null;
    let path = '/';

    if (parentId) {
      parentFolder = await Folder.findById(parentId);
      if (!parentFolder) {
        throw new Error('Parent folder not found');
      }
      path = `${parentFolder.path}${parentFolder.name}/`;
    }

    const folder = new Folder({
      name,
      owner: ownerId,
      parent: parentId || null,
      path,
      sharedWith: [],
    });
  }
}

```

```

    });

    await folder.save();
    return folder;
}

/**
 * Check if user has permission to access folder
 */
async checkPermission(folderId, userId, requiredPermission =
'read') {
    const folder = await Folder.findById(folderId);

    if (!folder) {
        throw new Error('Folder not found');
    }

    // Owner has all permissions
    if (folder.owner.toString() === userId.toString()) {
        return true;
    }

    // Check if folder is shared with user
    const sharedEntry = folder.sharedWith.find(
        (share) => share.user.toString() === userId.toString()
    );

    if (!sharedEntry) {
        return false;
    }

    // Check specific permission
    const permissions = sharedEntry.permissions;

    switch (requiredPermission) {
        case 'read':
            return permissions.read;
        case 'write':
            return permissions.write;
        case 'delete':
            return permissions.delete;
        default:
            return false;
    }
}

/**
 * Share folder with user
 */
async shareFolder(folderId, targetUserId, permissions, ownerId) {
    const folder = await Folder.findById(folderId);

    if (!folder) {
        throw new Error('Folder not found');
    }
}

```

```

// Verify ownership
if (folder.owner.toString() !== ownerId.toString()) {
  throw new Error('Only owner can share folder');
}

// Check if already shared
const existingShare = folder.sharedWith.find(
  (share) => share.user.toString() === targetUserId.toString()
);

if (existingShare) {
  // Update existing permissions
  existingShare.permissions = permissions;
} else {
  // Add new share
  folder.sharedWith.push({
    user: targetUserId,
    permissions,
    sharedAt: new Date(),
  });
}

await folder.save();
return folder;
}

/**
 * Get folder contents with permission check
 */
async getFolderContents(folderId, userId) {
  // Check read permission
  const hasAccess = await this.checkPermission(folderId, userId,
'read');

  if (!hasAccess) {
    throw new Error('Access denied');
  }

  // Get subfolders
  const folders = await Folder.find({
    parent: folderId,
    isDeleted: false,
  }).select('name size createdAt');

  // Get files
  const files = await File.find({
    folder: folderId,
    isDeleted: false,
  }).select('name originalName size mimetype thumbnail
uploadedAt');

  return { folders, files };
}

```

```

/**
 * Delete folder recursively
 */
async deleteFolder(folderId, userId, permanent = false) {
  const folder = await Folder.findById(folderId);

  if (!folder) {
    throw new Error('Folder not found');
  }

  // Check delete permission
  const hasPermission = await this.checkPermission(folderId,
userId, 'delete');

  if (!hasPermission) {
    throw new Error('No permission to delete folder');
  }

  if (permanent) {
    // Permanently delete folder and all contents
    await this.deleteFolderRecursive(folderId);
  } else {
    // Soft delete
    folder.isDeleted = true;
    folder.deletedAt = new Date();
    await folder.save();

    // Mark all files in folder as deleted
    await File.updateMany(
      { folder: folderId },
      { isDeleted: true, deletedAt: new Date() }
    );
  }

  return folder;
}

/**
 * Helper: Recursive permanent deletion
 */
async deleteFolderRecursive(folderId) {
  // Find all subfolders
  const subfolders = await Folder.find({ parent: folderId });

  // Recursively delete subfolders
  for (const subfolder of subfolders) {
    await this.deleteFolderRecursive(subfolder._id);
  }

  // Delete all files in this folder
  const files = await File.find({ folder: folderId });
  for (const file of files) {
    // Delete physical file
    const fs = require('fs').promises;
    try {

```

```

        await fs.unlink(file.path);
        if (file.thumbnail) {
            await fs.unlink(file.thumbnail);
        }
    } catch (err) {
        console.error('Error deleting file:', err);
    }

    await File.findByIdAndDelete(file._id);
}

// Delete folder document
await Folder.findByIdAndDelete(folderId);
}
}

module.exports = new FolderService();

```

Frontend – File Upload Component

```

// frontend/src/components/modals/UploadModal.tsx

import React, { useState, useCallback } from 'react';
import { useDropzone } from 'react-dropzone';
import { uploadFiles } from '@api/fileApi';
import { useFS } from '@zustand/fileSystemStore';
import { toast } from 'react-hot-toast';

interface UploadModalProps {
    isOpen: boolean;
    onClose: () => void;
    folderId: string;
}

const UploadModal: React.FC<UploadModalProps> = ({ isOpen, onClose,
folderId }) => {
    const [files, setFiles] = useState<File[]>([]);
    const [uploading, setUploading] = useState(false);
    const [progress, setProgress] = useState<{ [key: string]: number
}>({});
    const { refreshCurrentFolder } = useFS();

    const onDrop = useCallback((acceptedFiles: File[]) => {
        setFiles((prev) => [...prev, ...acceptedFiles]);
    }, []);

    const {
        getRootProps,
        getInputProps,
        isDragActive
    } =
useDropzone({
        onDrop,
        multiple: true,
    });

    const handleUpload = async () => {
        if (files.length === 0) {

```

```

        toast.error('Please select files to upload');
        return;
    }

    setUploading(true);

    try {
        const formData = new FormData();
        files.forEach((file) => {
            formData.append('files', file);
        });
        formData.append('folderId', folderId);

        // Upload with progress tracking
        await uploadFiles(formData, {
            onUploadProgress: (progressEvent) => {
                const percentCompleted = Math.round(
100)                (progressEvent.loaded * 100) / (progressEvent.total ||

                );

                setProgress((prev) => ({
                    ...prev,
                    overall: percentCompleted,
                }));
            },
        });

        toast.success(`${files.length}                file(s)                uploaded
successfully`);
        refreshCurrentFolder();
        onClose();
        setFiles([]);
    } catch (error: any) {
        console.error('Upload error:', error);
        toast.error(error.response?.data?.message                ||                'Upload
failed');
    } finally {
        setUploading(false);
        setProgress({});
    }
};

const removeFile = (index: number) => {
    setFiles((prev) => prev.filter((_, i) => i !== index));
};

if (!isOpen) return null;

return (
    <div className="modal modal-open">
        <div className="modal-box max-w-2xl">
            <h3 className="font-bold text-lg mb-4">Upload Files</h3>

            {/* Dropzone */}

```

```

<div
  {...getRootProps()}
  className={`border-2 border-dashed rounded-lg p-8 text-
center cursor-pointer transition-colors ${
    isDragActive
      ? 'border-primary bg-primary/10'
      : 'border-base-300 hover:border-primary/50'
  }}
>
  <input {...getInputProps()} />
  <div className="text-6xl mb-4">📁</div>
  {isDragActive ? (
    <p className="text-lg">Drop files here...</p>
  ) : (
    <p className="text-lg">
      Drag & drop files here, or click to select
    </p>
  )}
</div>

{/* File List */}
{files.length > 0 && (
  <div className="mt-4 max-h-64 overflow-y-auto">
    <h4 className="font-semibold mb-2">Selected Files
({files.length})</h4>
    <div className="space-y-2">
      {files.map((file, index) => (
        <div
          key={`_${file.name}-${index}`}
          className="flex items-center justify-between p-2
bg-base-200 rounded"
        >
          <div className="flex items-center gap-2 flex-1
min-w-0">
            <span className="text-2xl">📄</span>
            <div className="flex-1 min-w-0">
              <p
                className="text-sm
truncate">{file.name}</p>
              <p className="text-xs text-base-content/60">
                {(file.size / 1024 / 1024).toFixed(2)} MB
              </p>
            </div>
          </div>
          <div>
            {!uploading && (
              <button
                onClick={() => removeFile(index)}
                className="btn btn-ghost btn-sm btn-circle"
              >
                ✕
              </button>
            )}
          </div>
        </div>
      )})}
    </div>
  )}
)}}
```

```

        </div>
      </div>
    )}

    {/* Progress Bar */}
    {uploading && (
      <div className="mt-4">
        <div className="flex justify-between text-sm mb-1">
          <span>Uploading...</span>
          <span>{progress.overall || 0}%</span>
        </div>
        <progress
          className="progress progress-primary w-full"
          value={progress.overall || 0}
          max="100"
        ></progress>
      </div>
    )}

    {/* Actions */}
    <div className="modal-action">
      <button
        onClick={onClose}
        className="btn btn-ghost"
        disabled={uploading}
      >
        Cancel
      </button>
      <button
        onClick={handleUpload}
        className="btn btn-primary"
        disabled={files.length === 0 || uploading}
      >
        {uploading ? 'Uploading...' : `Upload ${files.length}
file(s)`}
      </button>
    </div>
  </div>
</div>
);
};

export default UploadModal;

```

Frontend – Zustand File System Store

```

// frontend/src/components/modals/UploadModal.tsx

import React, { useState, useCallback } from 'react';
import { useDropzone } from 'react-dropzone';
import { uploadFiles } from '@api/fileApi';
import { useFS } from '@zustand/fileSystemStore';
import { toast } from 'react-hot-toast';

```

```

interface UploadModalProps {
  isOpen: boolean;
  onClose: () => void;
  folderId: string;
}

const UploadModal: React.FC<UploadModalProps> = ({ isOpen, onClose,
folderId }) => {
  const [files, setFiles] = useState<File[]>([]);
  const [uploading, setUploading] = useState(false);
  const [progress, setProgress] = useState<{ [key: string]: number
}>({});
  const { refreshCurrentFolder } = useFS();

  const onDrop = useCallback((acceptedFiles: File[]) => {
    setFiles((prev) => [...prev, ...acceptedFiles]);
  }, []);

  const { getRootProps, getInputProps, isDragActive } =
useDropzone({
  onDrop,
  multiple: true,
});

  const handleUpload = async () => {
    if (files.length === 0) {
      toast.error('Please select files to upload');
      return;
    }

    setUploading(true);

    try {
      const formData = new FormData();
      files.forEach((file) => {
        formData.append('files', file);
      });
      formData.append('folderId', folderId);

      // Upload with progress tracking
      await uploadFiles(formData, {
        onUploadProgress: (progressEvent) => {
          const percentCompleted = Math.round(
            (progressEvent.loaded * 100) / (progressEvent.total ||
100)

          );

          setProgress((prev) => ({
            ...prev,
            overall: percentCompleted,
          }));
        },
      });
    }
  };
}

```

```

        toast.success(`${files.length}          file(s)          uploaded
successfully`);
        refreshCurrentFolder();
        onClose();
        setFiles([]);
    } catch (error: any) {
        console.error('Upload error:', error);
        toast.error(error.response?.data?.message          ||          'Upload
failed');
    } finally {
        setUploading(false);
        setProgress({});
    }
};

const removeFile = (index: number) => {
    setFiles((prev) => prev.filter((_, i) => i !== index));
};

if (!isOpen) return null;

return (
    <div className="modal modal-open">
        <div className="modal-box max-w-2xl">
            <h3 className="font-bold text-lg mb-4">Upload Files</h3>

            {/* Dropzone */}
            <div
                {...getRootProps()}
                className={`border-2 border-dashed rounded-lg p-8 text-
center cursor-pointer transition-colors ${
                    isDragActive
                        ? 'border-primary bg-primary/10'
                        : 'border-base-300 hover:border-primary/50'
                }}
            >
                <input {...getInputProps()} />
                <div className="text-6xl mb-4">📁</div>
                {isDragActive ? (
                    <p className="text-lg">Drop files here...</p>
                ) : (
                    <p className="text-lg">
                        Drag & drop files here, or click to select
                    </p>
                )}
            </div>

            {/* File List */}
            {files.length > 0 && (
                <div className="mt-4 max-h-64 overflow-y-auto">
                    <h4 className="font-semibold mb-2">Selected          Files
({files.length})</h4>
                    <div className="space-y-2">
                        {files.map((file, index) => (
                            <div

```

```

        key={`${file.name}-${index}`}
        className="flex items-center justify-between p-2
bg-base-200 rounded"
      >
        <div className="flex items-center gap-2 flex-1
min-w-0">
          <span className="text-2xl">📄</span>
          <div className="flex-1 min-w-0">
            <p
truncate">{file.name}</p>
            <p className="text-xs text-base-content/60">
              {(file.size / 1024 / 1024).toFixed(2)} MB
            </p>
          </div>
        </div>

        {!uploading && (
          <button
            onClick={() => removeFile(index)}
            className="btn btn-ghost btn-sm btn-circle"
          >
            ✕
          </button>
        )}
      </div>
    )}

    {/* Progress Bar */}
    {uploading && (
      <div className="mt-4">
        <div className="flex justify-between text-sm mb-1">
          <span>Uploading...</span>
          <span>{progress.overall || 0}%</span>
        </div>
        <progress
          className="progress progress-primary w-full"
          value={progress.overall || 0}
          max="100"
        ></progress>
      </div>
    )}

    {/* Actions */}
    <div className="modal-action">
      <button
        onClick={onClose}
        className="btn btn-ghost"
        disabled={uploading}
      >
        Cancel
      </button>
      <button

```

```

        onClick={handleUpload}
        className="btn btn-primary"
        disabled={files.length === 0 || uploading}
      >
        {uploading ? 'Uploading...' : `Upload ${files.length}
file(s)`}
      </button>
    </div>
  </div>
</div>
);
};

```

```
export default UploadModal;
```

```

### E.4 Frontend - Zustand File System Store

```

```typescript
// frontend/src/zustand/fileSystemStore.ts

import { create } from 'zustand';
import { getFolderContents, createFolder, deleteItem } from
'@/api/folderApi';
import { toast } from 'react-hot-toast';

interface FileItem {
  _id: string;
  name: string;
  originalName?: string;
  size: number;
  mimeType?: string;
  thumbnail?: string;
  uploadedAt?: string;
  createdAt?: string;
}

interface FolderItem {
  _id: string;
  name: string;
  size?: number;
  createdAt: string;
}

interface FSState {
  currentFolderId: string | null;
  currentPath: string[];
  folders: FolderItem[];
  files: FileItem[];
  loading: boolean;
  error: string | null;
  selectedItems: string[];
  viewMode: 'grid' | 'list';

  // Actions

```

```

    setCurrentFolder: (folderId: string | null) => void;
    navigateToFolder: (folderId: string, folderName: string) => void;
    navigateBack: () => void;
    refreshCurrentFolder: () => Promise<void>;
    createNewFolder: (name: string) => Promise<void>;
    deleteItems: (itemIds: string[], permanent?: boolean) =>
Promise<void>;
    toggleSelectItem: (itemId: string) => void;
    selectAll: () => void;
    clearSelection: () => void;
    setViewMode: (mode: 'grid' | 'list') => void;
  }

export const useFS = create<FSState>((set, get) => ({
  currentFolderId: null,
  currentPath: [{ id: null, name: 'Home' }],
  folders: [],
  files: [],
  loading: false,
  error: null,
  selectedItems: [],
  viewMode: 'grid',

  setCurrentFolder: async (folderId) => {
    set({ currentFolderId: folderId, loading: true, error: null });

    try {
      const data = await getFolderContents(folderId);
      set({
        folders: data.folders,
        files: data.files,
        loading: false,
      });
    } catch (error: any) {
      console.error('Error loading folder:', error);
      set({
        error: error.response?.data?.message || 'Failed to load
folder',
        loading: false,
      });
      toast.error('Failed to load folder contents');
    }
  },

  navigateToFolder: (folderId, folderName) => {
    const { currentPath } = get();
    const newPath = [...currentPath, { id: folderId, name:
folderName }];
    set({ currentPath: newPath });
    get().setCurrentFolder(folderId);
  },

  navigateBack: () => {
    const { currentPath } = get();
    if (currentPath.length > 1) {

```

```

        const newPath = currentPath.slice(0, -1);
        const parentFolder = newPath[newPath.length - 1];
        set({ currentPath: newPath });
        get().setCurrentFolder(parentFolder.id);
    }
},

refreshCurrentFolder: async () => {
    const { currentFolderId } = get();
    await get().setCurrentFolder(currentFolderId);
},

createNewFolder: async (name) => {
    const { currentFolderId } = get();

    try {
        await createFolder(name, currentFolderId);
        toast.success('Folder created successfully');
        await get().refreshCurrentFolder();
    } catch (error: any) {
        console.error('Error creating folder:', error);
        toast.error(error.response?.data?.message || 'Failed to
create folder');
        throw error;
    }
},

deleteItems: async (itemIds, permanent = false) => {
    try {
        await Promise.all(
            itemIds.map((id) => deleteItem(id, permanent))
        );

        toast.success(
            permanent
            ? 'Items deleted permanently'
            : 'Items moved to trash'
        );

        get().clearSelection();
        await get().refreshCurrentFolder();
    } catch (error: any) {
        console.error('Error deleting items:', error);
        toast.error(error.response?.data?.message || 'Failed to
delete items');
        throw error;
    }
},

toggleSelectItem: (itemId) => {
    set((state) => {
        const isSelected = state.selectedItems.includes(itemId);
        return {
            selectedItems: isSelected
            ? state.selectedItems.filter((id) => id !== itemId)

```

```

        : [...state.selectedItems, itemId],
      };
    });
  },

  selectAll: () => {
    const { folders, files } = get();
    const allIds = [
      ...folders.map((f) => f._id),
      ...files.map((f) => f._id),
    ];
    set({ selectedItems: allIds });
  },

  clearSelection: () => {
    set({ selectedItems: [] });
  },

  setViewMode: (mode) => {
    set({ viewMode: mode });
    localStorage.setItem('viewMode', mode);
  },
));

```

Socket.io Real-time Events

```

```javascript
// backend/socket/index.js

const jwt = require('jsonwebtoken');

module.exports = (io) => {
 // Authentication middleware
 io.use((socket, next) => {
 const token = socket.handshake.auth.token;

 if (!token) {
 return next(new Error('Authentication error'));
 }

 try {
 const decoded = jwt.verify(token, process.env.JWT_SECRET);
 socket.userId = decoded.id;
 next();
 } catch (err) {
 return next(new Error('Authentication error'));
 }
 });

 io.on('connection', (socket) => {
 console.log(`User connected: ${socket.userId}`);

 // Join user's personal room
 socket.join(`user:${socket.userId}`);
 });
};

```

```

 // Join folder rooms based on access
 socket.on('join-folder', (folderId) => {
 socket.join(`folder:${folderId}`);
 console.log(`User ${socket.userId} joined folder
${folderId}`);
 });

 socket.on('leave-folder', (folderId) => {
 socket.leave(`folder:${folderId}`);
 console.log(`User ${socket.userId} left folder ${folderId}`);
 });

 // File upload progress
 socket.on('upload-progress', ({ folderId, fileName, progress })
=> {
 io.to(`folder:${folderId}`).emit('file-upload-progress', {
 fileName,
 progress,
 userId: socket.userId,
 });
 });

 // Disconnect
 socket.on('disconnect', () => {
 console.log(`User disconnected: ${socket.userId}`);
 });
 });

 return io;
};
````

```

```

````typescript
// frontend/src/services/socketService.ts

import { io, Socket } from 'socket.io-client';

class SocketService {
 private socket: Socket | null = null;

 connect(token: string) {
 this.socket = io(import.meta.env.VITE_API_URL ||
'http://localhost:5000', {
 auth: { token },
 transports: ['websocket'],
 });
 }

 this.socket.on('connect', () => {
 console.log('Socket connected');
 });

 this.socket.on('disconnect', () => {
 console.log('Socket disconnected');
 });
}

```

```

 this.socket.on('connect_error', (error) => {
 console.error('Socket connection error:', error);
 });

 return this.socket;
}

disconnect() {
 if (this.socket) {
 this.socket.disconnect();
 this.socket = null;
 }
}

joinFolder(folderId: string) {
 this.socket?.emit('join-folder', folderId);
}

leaveFolder(folderId: string) {
 this.socket?.emit('leave-folder', folderId);
}

onFileUploaded(callback: (data: any) => void) {
 this.socket?.on('files-uploaded', callback);
}

onFileDeleted(callback: (data: any) => void) {
 this.socket?.on('file-deleted', callback);
}

onFolderShared(callback: (data: any) => void) {
 this.socket?.on('folder-shared', callback);
}

offFileUploaded(callback?: (data: any) => void) {
 this.socket?.off('files-uploaded', callback);
}

offFileDeleted(callback?: (data: any) => void) {
 this.socket?.off('file-deleted', callback);
}

offFolderShared(callback?: (data: any) => void) {
 this.socket?.off('folder-shared', callback);
}

getSocket() {
 return this.socket;
}
}

export default new SocketService();

```

## ДОДАТОК В. ІНСТРУКЦІЯ З ВСТАНОВЛЕННЯ

### Системні вимоги

#### Сервер:

- Операційна система: Linux (Ubuntu 20.04+), Windows Server 2019+, macOS
- CPU: 2+ ядра
- RAM: 2 GB (мінімум), 4 GB (рекомендовано)
- Disk: 20 GB (система) + необхідний обсяг для файлів
- Node.js: v20.0.0 або новіша
- MongoDB: v7.0.0 або новіша
- Nginx: v1.18+ (опціонально, для production)

#### Клієнт:

- Веб-браузер:
- Chrome 100+
- Firefox 100+
- Safari 15+
- Edge 100+
- Інтернет з'єднання: 1 Mbps+ (рекомендовано 10 Mbps+ для швидкої роботи)

### Встановлення на сервері

#### Крок 1: Встановлення залежностей

```
Ubuntu/Debian
sudo apt update
sudo apt install -y nodejs npm mongodb git nginx
Перевірка версій
node --version # Має бути v20+
npm --version
mongod --version
```

#### Крок 2: Клонування репозиторію

```
Клонування проекту
git clone https://github.com/0Z1CODE/3mHome.git
cd 3mHome
```

#### Крок 3: Налаштування Backend

```
Перехід в директорію backend
```

```
cd backend
Встановлення залежностей
npm install
Створення .env файлу
cp .env.example .env
nano .env
```

#### Приклад .env файлу:

```
Server
PORT=5000
NODE_ENV=production

Database
MONGODB_URI=mongodb://localhost:27017/3mhome
DB_NAME=3mhome

JWT
JWT_SECRET=your-super-secret-jwt-key-change-this
JWT_ACCESS_EXPIRATION=15m
JWT_REFRESH_EXPIRATION=7d

File Upload
UPLOAD_DIR=./uploads
MAX_FILE_SIZE=5368709120
ALLOWED_FILE_TYPES=image/*,video/*,audio/*,application/*,text/*

Cors
CORS_ORIGIN=http://localhost:5173,http://your-domain.com

Logging
LOG_LEVEL=info
```

#### Крок 4: Налаштування Frontend

```
Перехід в директорію frontend
cd ../frontend
```

```
Встановлення залежностей
npm install
```

```
Створення .env файлу
cp .env.example .env.production
nano .env.production
```

#### Приклад .env.production:

```
VITE_API_URL=http://your-server-ip:5000
VITE_SOCKET_URL=http://your-server-ip:5000
```

#### Крок 5: Білд Frontend

```
Створення production build
npm run build
Файли будуть в директорії dist/
```

#### Крок 6: Налаштування MongoDB

```
``bash
Запуск MongoDB
sudo systemctl start mongod
```

```
sudo systemctl enable mongod

Створення користувача БД (опціонально)
mongosh
```
```

```
```javascript
use 3mhome
```

```
db.createUser({
 user: "3mhome_user",
 pwd: "secure_password_here",
 roles: [
 { role: "readWrite", db: "3mhome" }
]
})
```

### Крок 7: Налаштування Nginx (Production)

```
```bash
sudo nano /etc/nginx/sites-available/3mhome
```
```

```
```nginx
server {
    listen 80;
    server_name your-domain.com;

    # Frontend
    location / {
        root /path/to/3mHome/frontend/dist;
        try_files $uri $uri/ /index.html;
    }

    # Backend API
    location /api {
        proxy_pass http://localhost:5000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
    }

    # Socket.io
    location /socket.io {
        proxy_pass http://localhost:5000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

```

        # Uploads (якщо потрібно віддавати файли через Nginx)
        location /uploads {
            alias /path/to/3mHome/backend/uploads;
            autoindex off;
        }
    }
}
...

```bash
Активация конфигу
sudo ln -s /etc/nginx/sites-available/3mhome /etc/nginx/sites-
enabled/
sudo nginx -t
sudo systemctl reload nginx

```

### Крок 8: Запуск додатку

```

Запуск backend
cd /path/to/3mHome/backend
npm start

Або через PM2 для production
npm install -g pm2
pm2 start server.js --name 3mhome-backend
pm2 startup
pm2 save

```

### **Встановлення через Docker (альтернатива)**

```

Перевірка наявності Docker
docker --version
docker-compose --version
...

docker-compose.yml:
```yaml
version: '3.8'

services:
  mongodb:
    image: mongo:7
    container_name: 3mhome-mongo
    restart: always
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: example
      MONGO_INITDB_DATABASE: 3mhome
    volumes:
      - mongodb_data:/data/db
    ports:
      - "27017:27017"

  backend:
    build:

```

```

    context: ./backend
    dockerfile: Dockerfile
    container_name: 3mhome-backend
    restart: always
    environment:
        PORT: 5000
        MONGODB_URI:
mongodb://root:example@mongodb:27017/3mhome?authSource=admin
        JWT_SECRET: your-secret-key
    volumes:
        - ./backend/uploads:/app/uploads
        - ./backend/logs:/app/logs
    ports:
        - "5000:5000"
    depends_on:
        - mongodb

```

```

frontend:
    build:
        context: ./frontend
        dockerfile: Dockerfile
    container_name: 3mhome-frontend
    restart: always
    ports:
        - "80:80"
    depends_on:
        - backend

```

```

volumes:
    mongodb_data:
    ...

```

****Запуск через Docker:****

```
```bash
```

```
Запуск всіх сервісів
```

```
docker-compose up -d
```

```
Перевірка статусу
```

```
docker-compose ps
```

```
Перегляд логів
```

```
docker-compose logs -f
```

## **Перевірка роботи**

- Відкрити браузер та перейти на <http://your-domain.com> або <http://localhost>
- Зареєструвати нового користувача
- Завантажити тестові файли
- Перевірити функціональність (upload, download, share, search)

## Налаштування HTTPS (Let's Encrypt)

# Встановлення Certbot

```
sudo apt install certbot python3-certbot-nginx
```

# Отримання SSL сертифіката

```
sudo certbot --nginx -d your-domain.com
```

# Автоматичне оновлення сертифікату

```
sudo certbot renew --dry-run
```

## Backup та відновлення

### Backup MongoDB:

# Backup

```
mongodump --db=3mhome --out=/backup/$(date +%Y%m%d)
```

# Restore

```
mongorestore --db=3mhome /backup/20251110/3mhome
```

### Backup файлів:

# Backup uploads директорії

```
tar -czf uploads-backup-$(date +%Y%m%d).tar.gz backend/uploads/
```

# Restore

```
tar -xzf uploads-backup-20251110.tar.gz -C backend/
```

## Troubleshooting

### Проблема: Backend не запускається

# Перевірити логи

```
pm2 logs 3mhome-backend
```

# Перевірити порт

```
sudo netstat -tulpn | grep :5000
```

# Перевірити MongoDB

```
sudo systemctl status mongod
```

### Проблема: Не можу завантажити файли

# Перевірити права на директорію

```
ls -la backend/uploads
```

```
chmod 755 backend/uploads
```

# Перевірити розмір диску

```
df -h
```

### Проблема: Socket.io не працює

# Перевірити Nginx конфігурацію для WebSocket

```
sudo nginx -t
```

```
Перевірити firewall
sudo ufw status
sudo ufw allow 5000/tcp
```

# ДОДАТОК Г. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ)



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ДЕРЖАВНИЙ УНІВЕРСИТЕТ  
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ  
НАВЧАЛЬНО – НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  
Кафедра Комп'ютерних наук

## ДИПЛОМНА РОБОТА

на на ступінь вищої освіти магістр  
із спеціальності F3 Комп'ютерні науки

### **Мережеве сховище NAS (Network Attached Storage) з використанням React.js, Node.js, MongoDB**

Виконав: студент 6 курсу, групи КНДМ-61

Марохонько Максим Олегович

Керівник: професор кафедри Комп'ютерних наук  
д.ф.-м.н., професор Шукула О.М.

Київ - 2025

Слайд 1

## АКТУАЛЬНІСТЬ ДОСЛІДЖЕННЯ

Сучасні виклики та потреби ринку

-  **Експоненційне зростання даних**  
Щорічно обсяг даних зростає на 40%
-  **Безпека та контроль доступу**  
60% компаній мають витоки даних
-  **Висока вартість рішень**  
Dropbox: \$15-25/mic, Google: \$ 6-18
-  **Відсутність гнучкості**  
Обмежені можливості налаштувань

 Ринок хмарних  
сховищ  
\$100+ млрд  
(2024)

 Безпека  
60%  
Проблеми з доступом

✓ Полит  
85%  
Шукають альтернативи

Слайд 2

# МЕТА ТА ЗАВДАННЯ РОБОТИ

 МЕТА РОБОТИ:  
Розробка веб-додатку для управління файлами з гнучкою системою контролю доступу та real-time взаємодією

## Завдання дослідження:

- ✓ Аналіз предметної області
  - ✓ Проектування архітектури
  - ✓ Розробка системи безпеки
  - ✓ Реалізація файлового менеджера
- ✓ Система прав доступу
  - ✓ Realtime комунікація
  - ✓ Адміністративна панель
  - ✓ Тестування та оптимізація

Слайд 3

## АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Порівняння з провідними системами

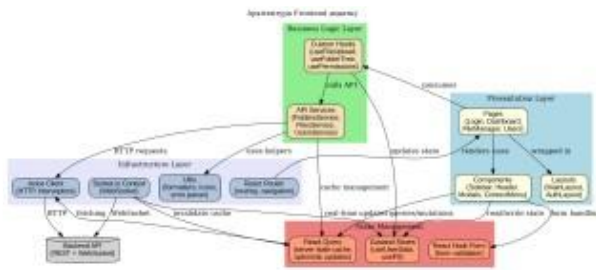
| Критерій                                                                                      | Dropbox  | Google Drive | OneDrive | 3mHome  |
|-----------------------------------------------------------------------------------------------|----------|--------------|----------|----------------------------------------------------------------------------------------------|
|  Вартість  | \$15-25  | \$6-18       | \$5-20   | ✓ \$0                                                                                        |
|  Права     | Обмежено | Середньо     | Обмежено | ✓ Повні                                                                                      |
|  Real-time | Частково | Так          | Так      | ✓ Так                                                                                        |
|  Selfhost  | ✗        | ✗            | ✗        | ✓ Так                                                                                        |
|  Open Src  | ✗        | ✗            | ✗        | ✓ MIT                                                                                        |

 КЛЮЧОВА ПЕРЕВАГА:  
3mHome поєднує можливості провідних платформ, будучи безкоштовним, відкритим та self-hosted рішенням

Слайд 4

# АРХІТЕКТУРА ДОДАТКУ 3mHome

ClientServer архітектура з realtime комунікацією



## Технологічний стек

### Frontend

- React 18 + TypeScript 5
- Vite + TailwindCSS
- Zustand (State)
- Socket.io Client

### Backend

- Node.js 20 + Express.js
- MongoDB + Mongoose
- JWT Authentication
- Socket.io Server
- Multer (Upload)

DevOps: Git | Nginx | PM2 | Docker

🚀 150ms API | 📦 < 1 MB Bundle | ⚡ 50+ Users

## Слайд 5

# МОДЕЛЬ ДАНИХ

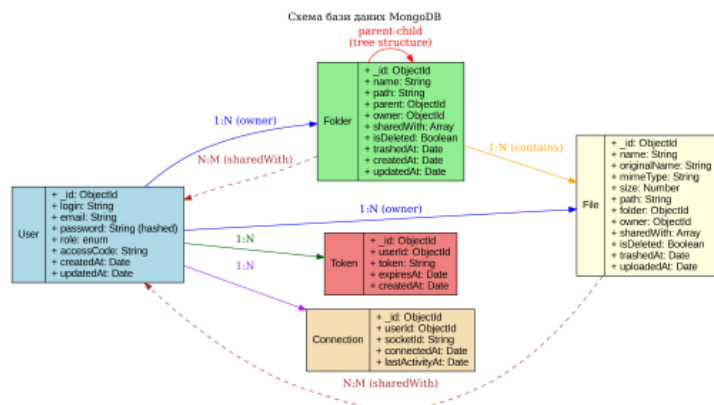
MongoDB з Mongoose ODM

## Основні колекції MongoDB:

- Users - користувачі  
username, email, role
- Folders - структура папок
- Files - метадані файлів  
name, size, mimetype, path
- Permissions - права доступу
- Tokens - JWT токени

## Зв'язки:

User → Folders (1:N)  
Folder → Files (1:N)  
File → Permissions (1:N)



## Слайд 6

# БЕЗПЕКА

JWT-based authentication

## Механізм JWT:

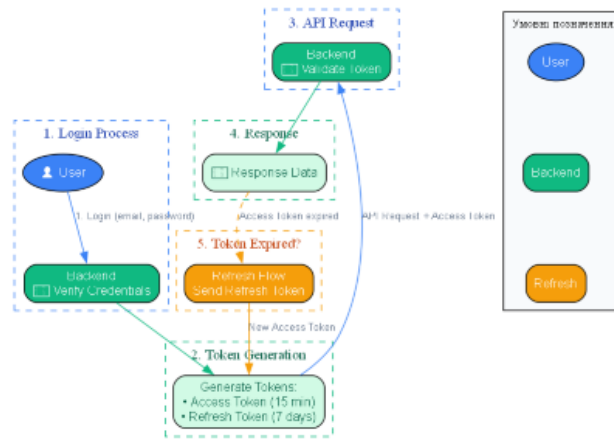
Access Token ( 15 min)  
Refresh Token ( 7 days)  
АВТООновлення

## Типи користувачів:

- User - свої файли
- Admin - управління
- Guest - обмежений доступ

## Захист:

Bcrypt (10 rounds)  
HTTP-only cookies  
CORS політика  
Rate limiting (100/15min)



Слайд 7

# ФАЙЛОВИЙ МЕНЕДЖЕР

Інтуїтивний інтерфейс

## Основний функціонал:

### Управління папками:

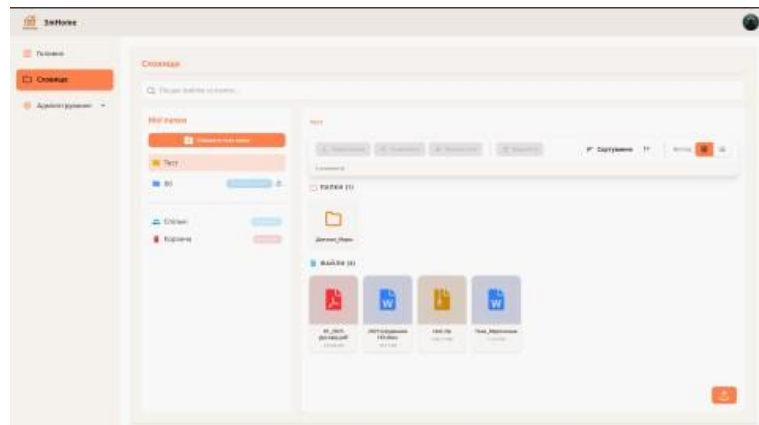
Необмежена вкладеність  
Drag-and-drop, перейменування

### Управління файлами:

Upload до 5 GB (single/multiple)  
Download (file або ZIP)  
Preview (images, video, PDF)

### Навігація:

Breadcrumbs, Grid/List view  
Сортування (назва/дата/розмір)



Слайд 8

# СИСТЕМА ПРАВ ДОСТУПУ

Гранулярний контроль доступу

Триступенева система:

👁️ **READ ( Читання):**

Перегляд, скачування, preview

✍️ **WRITE ( Запис):**

READ + завантаження файлів

Створення підпапок

🗑️ **DELETE ( Видалення):**

WRITE + видалення

Переміщення

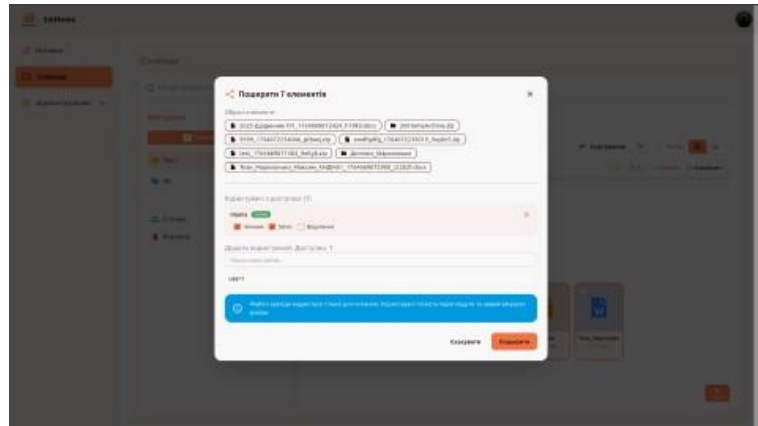
Особливості:

Власник має всі права

Індивідуальні налаштування

Успадкування прав

Миттєве відкликання



Слайд 9

# ПРОЦЕС ЗАВАНТАЖЕННЯ

Multipart upload з WebSocket

Upload flow з real-time прогресом

1 Вибір файлів (drag-drop)

2 Frontend → multipart POST

3 Backend (Multer) → /uploads/

4 Генерація thumbnail

5 Запис у MongoDB

6 Socket.io → broadcast

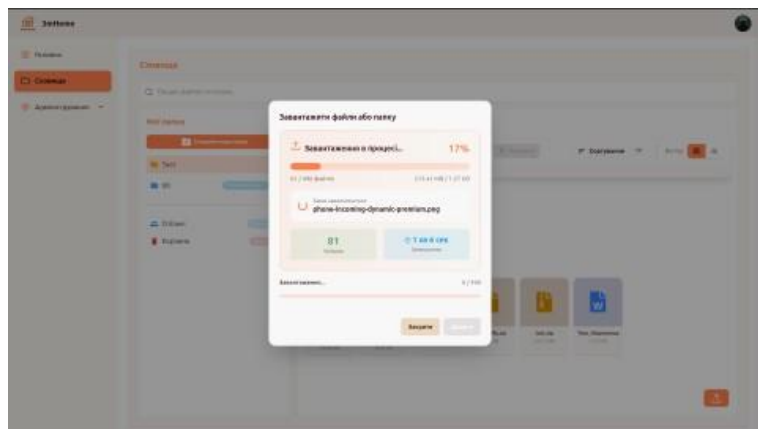
7 Оновлення для всіх користувачів

Можливості:

Множинний upload

Progress tracking

До 5 GB, MIME validation



Слайд 10

## ІНТЕЛЕКТУАЛЬНИЙ ПОШУК

## Пошук за параметрами

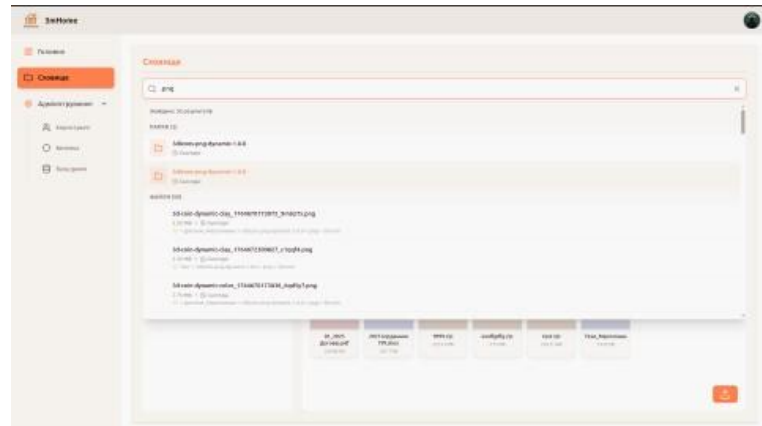
### Критерії пошуку:

- По імені (Text Index)
- По типу (Docs/Images/Video)
- По даті та власнику
- По розміру (<1MB, 1-10MB, >10MB)

 **Технології :**

- MongoDB Aggregation
- Debounce ( 500ms)
- Infinite Scroll
- Highlight matches

⚡ < 100ms | Точність: 95 %+



Слайд 11

## REAL-TIME КОМУНІКАЦІЯ

## Миттєва синхронізація

## WebSocket через Socket.io:

 **Події:**

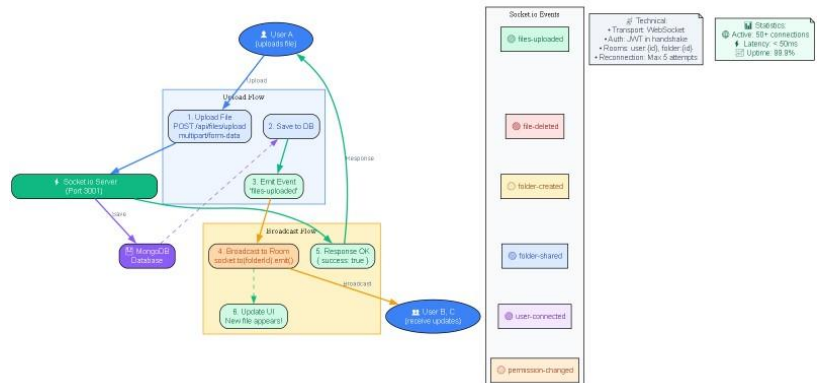
- files-uploaded (новый файл)
- file-deleted (удалено)
- folder-shared (доступ)
- user-connected (онлайн)

 **Переваги:**

- Миттєва синхронізація  
Без перезавантаження  
Push -повідомлення  
Авто реконекція

🔧 WebSocket (fallback: Polling)

🌐 Latency: <50ms | Uptime: 99.9%



Слайд 12

# АДМІНІСТРАТИВНА ПАНЕЛЬ

## Централізоване управління:

### Управління користувачами:

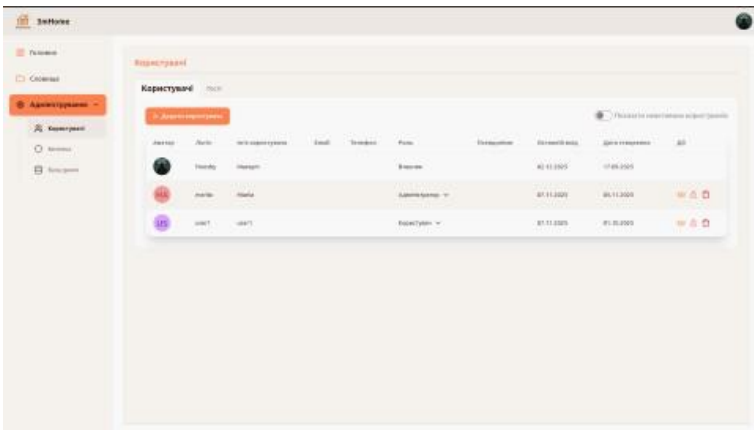
- Перегляд, блокування
- Зміна ролей (User ↔ Admin)
- Примусовий logout

### Аналітика:

- Файли та storage
- Активні користувачі
- Історія завантажень

### Безпека та логи:

- Системні логи (errors, auth)
- Моніторинг активності
- Backup database



Слайд 13

# ТЕСТУВАННЯ

## Комплексне тестування:

### Unit Testing:

- Models, Services, Middleware
- Coverage: 75% | 124/124 ✓

### Integration Testing:

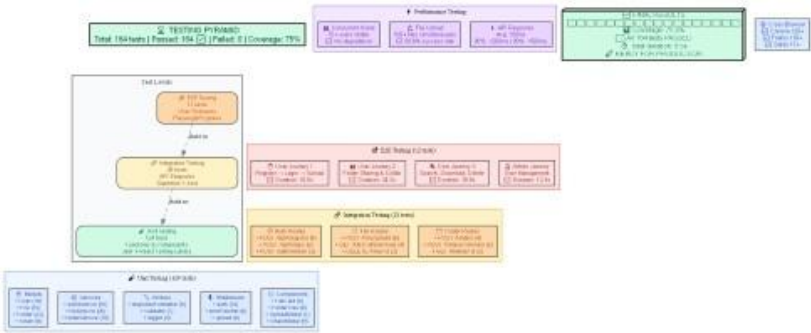
- API (Auth, Files, Folders)
- Permission checks
- Success: 100% | 120 ms

### E2E Testing:

- User journeys
- Cross-browser

### Performance:

- 50+ concurrent users
- 95% <200ms | Error <0.2%



Слайд 14

# ВИСНОВКИ

## ✓ **Досягнуті результати**

- ✓ Розроблено повнофункціональний вебдодаток
- ✓ Реалізовано гнучку систему прав (Read/Write/Delete)
- ✓ Впроваджено real time синхронізацію (WebSocket)
- ✓ Забезпечено безпеку (JWT, bcrypt)
- ✓ Створено responsive інтерфейс
- ✓ Готово до production

## 🚀 **Напрямки подальшого розвитку**

- 📱 Мобільні додатки (iOS/Android)
- 📱 Офлайн режим (PWA)
- 📱 Онлайн-редактори документів
- 🔍 AI-пошук по вмісту файлів
- ☁️ Cloud Integration (AWS S3, Azure)
- 📊 Analytics та мультимовність

🎓 **Дякую за увагу !.**

Слайд 15