Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: *Розподілена комп'ютерна система зберігання цифрових ресурсів з розмежуванням прав доступу*

Виконав: студент 4 курсу, групи СІ-42
спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

(підпис) *Ковальський А. Ю.* (прізвище та ініціали)

Керівник (підпис) *Тиш Є. В.* (прізвище та ініціали)

Нормоконтроль (підпис) *Луцик. Н. С.* (прізвище та ініціали)

Завідувач кафедри (підпис) *Осухівська Г. М.* (прізвище та ініціали)

Рецензент (підпис) *Гром'як Р.С.* (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет *комп'ютерно-інформаційних систем і програмної інженерії*
(повна назва факультету)

Кафедра *комп'ютерних систем та мереж*
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Осухівська Г.М.*
(підпис) (прізвище та ініціали)
«25» квітня 2026 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *бакалавр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студента *Ковальського Антона Юрійовича*
(прізвище, ім'я, по батькові)

1. Тема роботи *Розподілена комп'ютерна система зберігання цифрових ресурсів з розмежуванням прав доступу*

Керівник роботи *кандидат технічних наук, доцент кафедри КС Тиш Євгенія Володимирівна*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від *« 25 » квітня 2026 року № 4/9-188*
2. Термін подання студентом завершеної роботи 16.06.2026
3. Вихідні дані до роботи *Технічне завдання*

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз технічного завдання
2. Проєктна частина
3. Практична частина
4. Безпека життєдіяльності, основи охорони праці.
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Електрична структурна схема
2. Структурна схема програмного забезпечення
3. Електрична принципова схема
4. Блок-схема алгоритму

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	*Сенчишин В.С., к.т.н., доц. каф. МТ*		

7. Дата видачі завдання *25.04.2026 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка технічного завдання*	*26.01 –01.02*	*Викон.*
2.	*Робота над першим розділом: Аналіз технічного завдання, огляд існуючих рішень для хмарного зберігання та формування вимог до системи.*	*02.02 – 14.02*	
3.	*Робота над другим розділом: Проєктна частина, обґрунтування вибору апаратної бази (Orange Pi) та проєктування архітектури реляційної бази даних.*	*18.04 – 26.04*	
4.	*Робота над третім розділом: Практична частина, розробка серверної логіки (Flask), верстка адаптивного інтерфейсу та тестування модуля ефемерних посилань.*	*27.04 – 08.05*	
5.	*Безпека життєдіяльності, основи охорони праці*	*09.05 – 20.05*	
6.	*Оформлення пояснювальної записки і графічного матеріалу*	*21.05 – 07.06*	
7.	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*08.06 – 14.06*	
8.	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*	
9.	*Захист кваліфікаційної роботи бакалавра*	*23.06.2026*	

Студент _____ *Ковальський А. Ю.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Тиш Є. В.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Ковальський А. Ю. Розподілена комп’ютерна система зберігання цифрових ресурсів з розмежуванням прав доступу: робота на здобуття кваліфікаційного ступеня бакалавра: спец. 123 — комп’ютерна інженерія. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2026.

Ключові слова: хмарне сховище, веб-додаток, обмін даними, ефемерні посилання, клієнт-серверна архітектура, Flask, SQLite.

У кваліфікаційній роботі розроблено вебсистему розподіленого хмарного зберігання та обміну даними. Проведено аналіз існуючих рішень для хмарного зберігання (Google Drive, Dropbox тощо), що дозволило сформувати чіткі вимоги до функціоналу, безпеки та інтерфейсу розроблюваного програмного продукту.

Проєктна частина містить розробку логічної архітектури реляційної бази даних, алгоритмів безпечної автентифікації користувачів, обробки файлових потоків та механізму генерації тимчасових (ефемерних) посилань з автоматичним знищенням даних для безпечного публічного обміну. Програмна частина (серверна логіка) реалізована мовою Python з використанням мікрофреймворку Flask та СУБД SQLite. Клієнтська частина розроблена з використанням сучасних вебтехнологій (HTML5, CSS3, JavaScript) та забезпечує адаптивний користувацький інтерфейс. Отримані результати тестування показують високу швидкість обробки запитів, стабільність роботи системи управління файлами та надійність механізмів розмежування прав доступу.

ANNOTATION

Kovalskyi A. Y. Distributed Computer System for Digital Resource Storage with Access Rights Management: Bachelor's Graduation Thesis: speciality 123 — computer engineering. Ternopil: Ternopil Ivan Puluj National Technical University, 2026.

Keywords: cloud storage, web application, data exchange, ephemeral links, client-server architecture, Flask, SQLite.

The qualification thesis covers the development of a web system for distributed cloud storage and data exchange. An analysis of existing cloud storage solutions (Google Drive, Dropbox, etc.) was conducted, which allowed for the formulation of clear requirements regarding the functionality, security, and interface of the developed software product.

The design stage includes the development of the logical architecture of a relational database, algorithms for secure user authentication, file stream processing, and a mechanism for generating temporary (ephemeral) links with automatic data destruction for secure public sharing. The software component (server-side logic) is implemented in Python using the Flask microframework and SQLite DBMS. The client side is developed using modern web technologies (HTML5, CSS3, JavaScript), ensuring a responsive user interface. The testing results demonstrate high query processing speed, stability of the file management system, and robustness of the access control mechanisms.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

ACID – Atomicity, Consistency, Isolation, Durability (Атомарність, Узгодженість, Ізольованість, Довговічність)

ARM – Advanced RISC Machine

CPU – Central Processing Unit (Центральний процесор)

CSS – Cascading Style Sheets (Каскадні таблиці стилів)

HTML – HyperText Markup Language (Мова розмітки гіпертексту)

HTTP – HyperText Transfer Protocol (Протокол передавання гіпертексту)

I/O – Input/Output (Введення/Виведення)

IP – Internet Protocol (Міжмережевий протокол) J

S – JavaScript

JSON – JavaScript Object Notation (Текстовий формат обміну даними)

NAS – Network Attached Storage (Мережеве сховище даних)

RAM – Random Access Memory (Оперативна пам'ять)

SoC – System-on-a-chip (Система на кристалі)

QL – Structured Query Language (Мова структурованих запитів)

SSH – Secure Shell (Протокол безпечного віддаленого доступу)

TTL – Time-to-Live (Час життя / Термін дії ефемерних даних)

USB – Universal Serial Bus (Універсальна послідовна шина)

UUID – Universally Unique Identifier (Універсальний унікальний ідентифікатор)

VPN – Virtual Private Network (Віртуальна приватна мережа)

ВСТУП

У сучасному світі стрімкий розвиток інформаційних технологій та глобальна цифровізація кардинально змінюють підходи до управління даними та підвищують вимоги до безпеки зберігання інформації у найрізноманітніших сферах — від корпоративного сектору до освітніх та наукових установ. Системи зберігання цифрових ресурсів залишаються одними з ключових складових сучасної IT-інфраструктури: здатність надійно ізолювати, структурувати та контролювати доступ до файлових потоків відкриває нові можливості для автоматизації процесів, командної роботи та захисту конфіденційних даних. Особливо актуальним є застосування гнучких веб-платформ із чітким розмежуванням прав користувачів, адже сучасні цифрові активи потребують захисту від несанкціонованого доступу, витоків інформації та потребують впровадження механізмів безпечного тимчасового обміну даними в умовах постійних кіберзагроз.

Ця робота присвячена розробці розподіленої комп'ютерної системи зберігання цифрових ресурсів з розмежуванням прав доступу. Головною метою виступає створення інтегрованого клієнт-серверного програмного комплексу, що забезпечує оперативне та безпечне завантаження файлів, динамічний моніторинг і аналіз використання дискового простору сховища, гнучке налаштування прав доступу для внутрішніх користувачів системи та інтуїтивне відображення результатів і журналів аудиту оператору в режимі реального часу через вебінтерфейс.

Особлива увага приділялася проєктуванню реляційної архітектури бази даних, розробці алгоритмів безпечної автентифікації користувачів із застосуванням сучасних методів криптографічного хешування паролів, а також реалізації механізму ефемерних (тимчасових) посилань із обмеженим часом життя (TTL). Такий підхід дозволяє мінімізувати ризики довгострокового компрометування посилань, усунути «сліпі зони» у контролі

за публічними файлами та забезпечити стабільне автоматичне очищення серверного простору від застарілих пакетів даних.

У роботі проведено аналіз існуючих аналогів та готових рішень у сфері хмарного зберігання і хмарних сервісів, що дозволило сформувати обґрунтовані технічні вимоги до майбутнього програмного продукту. Особливу увагу приділено ергономіці й адаптивності користувацького інтерфейсу, відмовостійкості серверної логіки під час паралельної обробки файлових потоків, а також інтеграції засобів моніторингу безпеки (зокрема, фіксації використання засобів анонімізації трафіку). У фінальній частині наведено результати функціонального тестування системи, обговорено практичні аспекти розгортання і впровадження веб-додатка та окреслено перспективи його подальшого масштабування й удосконалення.

РОЗДІЛ 1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Аналіз предметної області та постановка задачі

Будь-яка комп’ютеризована система розробляється за схожим алгоритмом: перший етап розробки включає аналіз готових рішень та дослідження принципів їх роботи. Даний етап має певну складність через необхідність забезпечення високого рівня захисту даних та розмежування прав доступу в сучасних вебдодатках [13]. Ця проблема тісно пов’язана з постійним зростанням обсягів цифрової інформації та ризиками несанкціонованого доступу до неї [17].

Система хмарного зберігання необхідна для безпечного розміщення файлів та керування ними. Надійна архітектура потрібна для корпоративних, освітніх та особистих цілей [3]. Особливу увагу під час проєктування варто приділити підбору методів, які забезпечують створення ефемерних (тимчасових) посилань з автоматичним знищенням даних. Зазначені аспекти дозволяють сформулювати ключові вимоги до системи. Насамперед вона повинна гарантувати надійне розмежування прав доступу та безпечну автентифікацію, пропонуючи при цьому простий, інтуїтивно зрозумілий і адаптивний інтерфейс користувача [10]. Додатково від архітектури вимагається висока швидкість обробки файлових потоків, зокрема під час завантаження чи генерації ZIP-архівів, а також загальна висока відмовостійкість і стабільність роботи серверної частини.

На основі аналізу формується концепція системи, яка включає визначення необхідних програмних компонентів та їх взаємодії (клієнт-серверна архітектура). Особливу увагу необхідно зосередити на виборі методів для хешування паролів [20], генерації унікальних токенів (UUID) та

автоматичного очищення дискового простору від застарілих файлів [14]. Теоретичне обґрунтування вибору технологій (Flask [11], SQLite [18]) базується на аналізі їхніх технічних характеристик, легкості розгортання та можливостей масштабування.

Розробка алгоритмів включає створення методів обробки HTTP-запитів, безпечного збереження файлів у файловій системі сервера та взаємодії з реляційною базою даних. Передбачається реалізація захисту від несанкціонованого доступу та системного аудиту дій користувачів. Логіка роботи системи забезпечує її точність і стабільність. Теоретична розробка програмного забезпечення базується на розроблених алгоритмах. Програма реалізує алгоритми завантаження даних, генерації посилань та виведення статистики. Визначаються методи взаємодії з користувачем через графічний вебінтерфейс.

Додатково, дослідження охоплює проєктування способів відображення інформації. Теоретичний макет інтерфейсу розроблено для зручного отримання інформації про стан сховища, наявність вільного місця та статус спільних файлів.

Завдання дослідження включають створення ефективного інструменту для зберігання та обміну даними з підтримкою функції самознищення пакетів файлів (TTL).

Ця розробка спрямована на впровадження технології, яка об'єднає точність управління даними, простоту використання та реальну користь для організацій або окремих користувачів. Основна мета системи полягає у створенні рішення, що дозволить не лише зберігати ресурси, а й інтегрувати ці процеси у безпечне цифрове середовище.

Особлива увага приділяється реалізації інтуїтивного управління, що зробить систему доступною для широкого кола користувачів без додаткового навчання. Вона зможе забезпечити оперативний моніторинг використання дискового простору у реальному часі, зберігаючи стабільність і надійність

роботи навіть при великих навантаженнях [5].

Результатом реалізації стане вебдодаток, здатний забезпечити ефективний обмін файлами і підтримувати широкий спектр завдань у різних сферах, таких як освіта, розробка програмного забезпечення чи корпоративний сектор. Завдяки такій системі користувачі зможуть оперативно ділитися файлами без ризику їх постійного перебування у відкритому доступі, що зробить проєкт важливим для захисту конфіденційної інформації.

У кваліфікаційній роботі буде дотримано всіх методичних вказівок до виконання роботи.

1.2 Розгляд готових рішень

У сучасному цифровому просторі домінування на ринку та ефективність робочих процесів все частіше залежать не лише від обчислювальних потужностей, а й від здатності компаній захищати свої дані за рахунок інформаційної безпеки. Одним із важливих інструментів у досягненні цієї переваги є системи контролю доступу та моніторингу цифрових активів [6]. Йдеться про технології, які виявляють несанкціоновані спроби доступу — зокрема використання анонімізаторів (VPN/Proxy) [5] або спроби масового завантаження — і в режимі реального часу інформують адміністратора про потенційну загрозу витоку. Готові рішення для хмарного зберігання уже давно не є експериментальними системами — вони активно застосовуються у корпоративному секторі XXI століття та довели свою ефективність.

В основі таких систем лежить принцип проактивного захисту та контролю життєвого циклу даних. Чим швидше система розпізнає закінчення терміну актуальності інформації або підозрілу активність, тим більше шансів уникнути її витоку. У випадку з публічним обміном файлами однією з найпоширеніших загроз є неконтрольоване поширення постійних посилань. У відповідь на такі дії система має або сповістити власника, або автоматично

запустити механізми захисту — фізично видалити файл та деактивувати токен доступу (ефемерні посилання), що створює бар'єр для подальшого завантаження сторонніми особами. Принципово важливо, що подібна дія має відбуватися автоматично, без необхідності ручного втручання адміністратора.

Класичним прикладом реалізованої системи такого типу є Google Drive (рис. 1.1), яка масово використовується як у приватному, так і в корпоративному секторах. Вона здатна зберігати дані різноманітних форматів і точно визначати рівні доступу користувачів, надаючи відповідні права (перегляд, коментування, редагування). Цікаво, що при налаштуванні корпоративних політик Workspace, система автоматично може ініціювати заходи безпеки — наприклад, заблокувати можливість завантаження або скасувати доступ для зовнішніх користувачів.

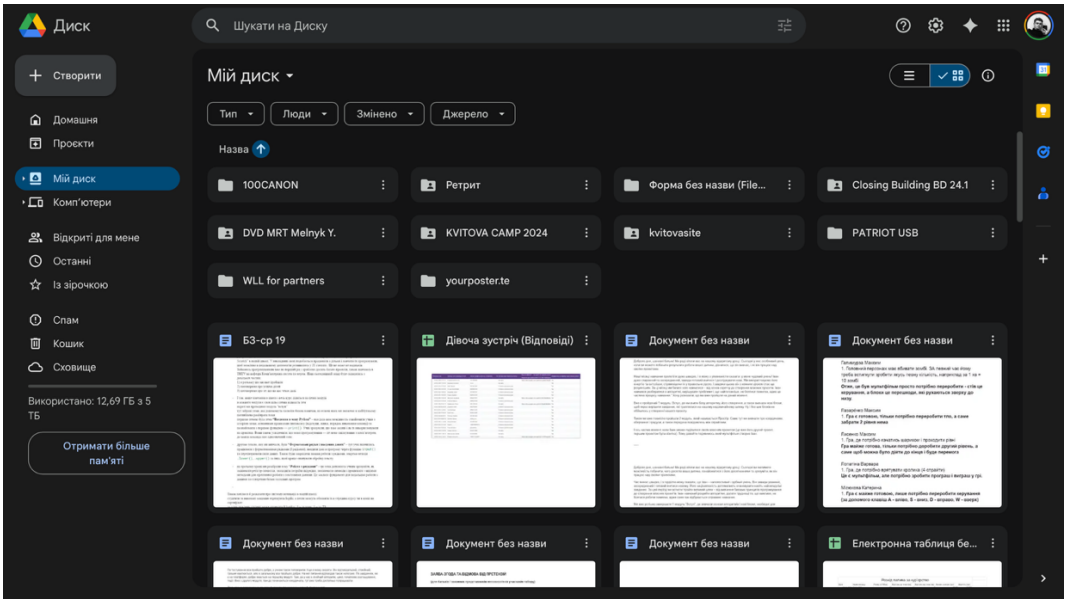

Рисунок 1.1 – Інтерфейс хмарного сховища Google Drive

Схожі функціональні можливості реалізовані і в хмарному сервісі Dropbox. Цей сервіс, який був одним із піонерів зручної синхронізації файлів у кінці 2000-х років, згодом отримав глибоку модернізацію, в ході якої було інтегровано розширені системи командної роботи. Ця платформа має

алгоритми, які фіксують історію версій та сигналізують власнику про зміни у файлах. У платних версіях система дозволяє встановлювати паролі та терміни дії для публічних посилань, що створює перешкоди для небажаного доступу. На відміну від широкої екосистеми Google, рішення Dropbox спочатку було більш обмеженим функціонально, проте повністю відповідало доктрині компанії щодо максимальної простоти і швидкості синхронізації файлів.

Якщо ж звернутися до історичних прикладів, то ще на етапі становлення мережі Інтернет було закладено ідею віддаленого зберігання та виявлення доступу за допомогою протоколу FTP (File Transfer Protocol). Одним із перших прикладів такої реалізації стали класичні FTP-сервери, принципово важливі в історії розвитку таких технологій. Ця система встановлювалася на широкому спектрі серверних ОС (UNIX, Windows). Основною функцією FTP була не лише можливість передати файл, а й базова автентифікація за логіном і паролем, а також класифікація прав доступу (запис чи читання). У клієнтських програмах відображався деревоподібний інтерфейс (рис. 1.2), який показував структуру каталогів на сервері. Хоча система не мала автоматичного зв'язку із сучасними засобами криптографії, її наявність значно підвищувала можливості віддаленої роботи з інформацією.

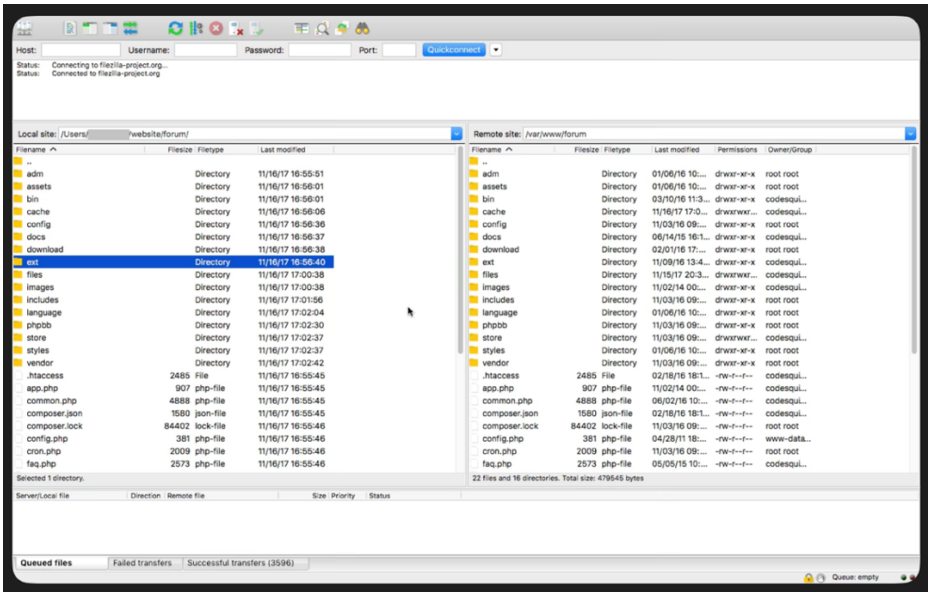

Рисунок 1.2 – Інтерфейс класичного FTP-клієнта (на прикладі FileZilla)

Проте розвиток технологій не зупиняється лише на базовому зберіганні. У корпоративних мережах, де швидкість реагування на інциденти ще критичніша, використовуються більш складні системи — зокрема комплекси управління ідентифікацією та аудитом. Вони базуються на аналізі цифрового сліду, який залишається за кожним користувачем. Це особливо актуально для організацій, які є вразливими до викрадення конфіденційних даних через компрометацію облікових записів.

У цьому контексті платформи корпоративного рівня (наприклад, AWS S3 або Microsoft OneDrive for Business) відіграють незамінну роль. Вони оснащені системами логування (рис. 1.3), які покривають усі запити до сховища простору і можуть виявити аномальну активність ще до того, як зловмисник завантажить критичний об'єм даних. У такому випадку система миттєво подає сигнал адміністратору або автоматично блокує токени доступу, змушуючи зловмисника втратити зв'язок із сервером.

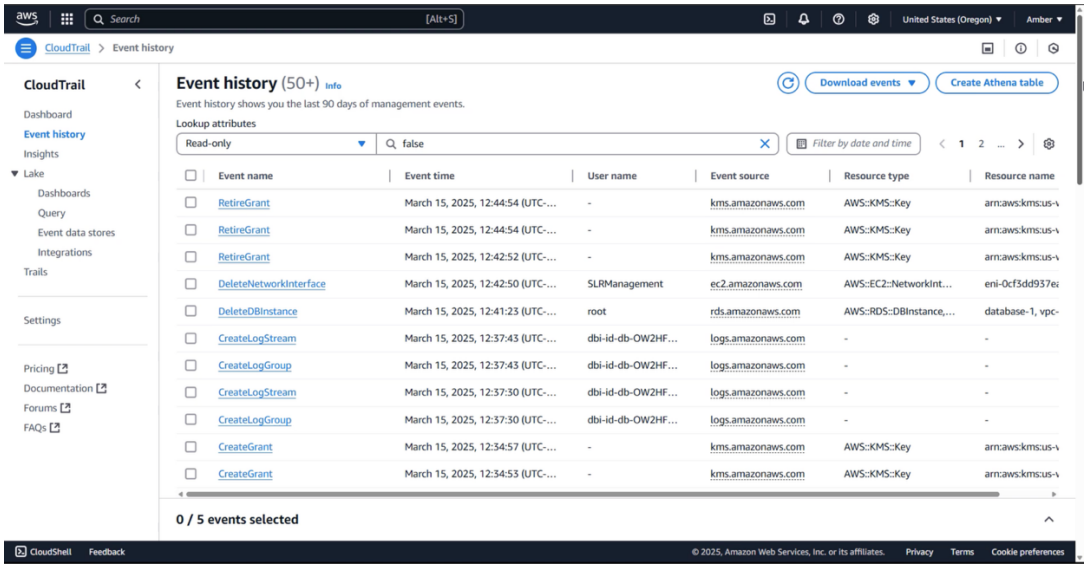

Рисунок 1.3 – Блок аудиту та логування доступу в панелі адміністратора (AWS CloudTrail)

Не менш важливу роль відіграють системи моніторингу стану сервера. Вони є невід'ємною частиною комплексу обслуговування більшості вебдодатків. Такі системи здатні фіксувати не тільки обсяги використаного простору, а й типи файлів, що зберігаються (документи, медіа, архіви). Сучасні варіанти дашбордів відображають ситуацію в реальному часі на дисплеї адміністратора та користувача, що дозволяє раціонально управляти дисковими квотами (рис. 1.4).

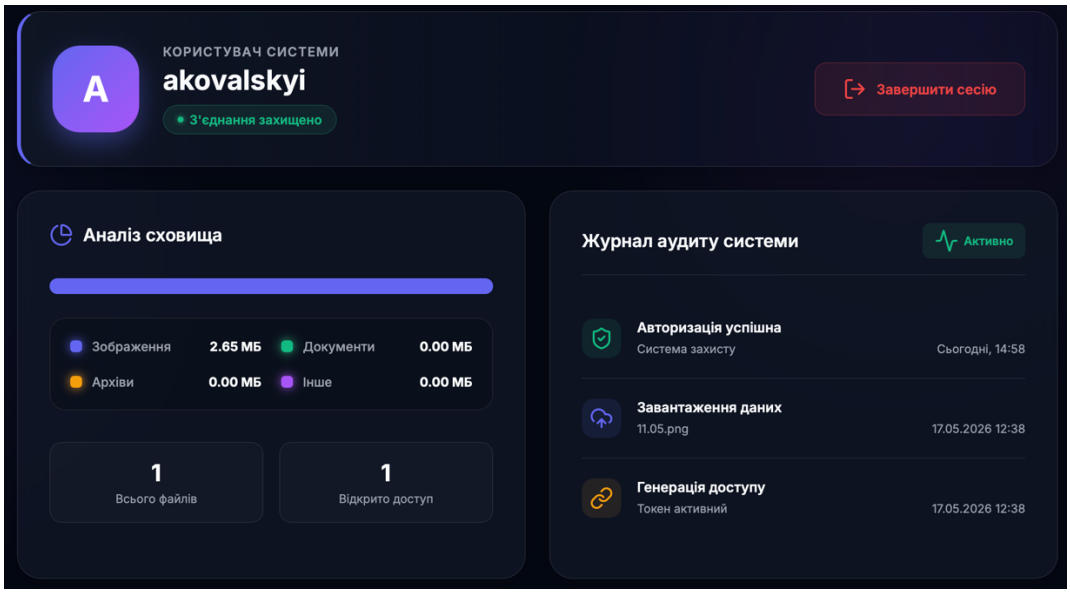

Рисунок 1.4 – Інтерфейс аналітичної панелі використання сховища

Готові рішення систем хмарного зберігання та обміну даними демонструють високий рівень технічного виконання та можливостей. Вони вже давно стали необхідною умовою для збереження ефективності компаній в умовах сучасної цифровізації. У сьогоденні, де інформація є головним ресурсом, надійне розмежування доступу є першим рівнем безпеки, який дозволяє уникнути витоків та продовжити виконання робочих завдань. Системи, обладнані інструментами аудиту та ефемерними посиланнями, мають можливість успішно протидіяти загрозам втрати контролю над даними.

1.2.1 Аналіз платформ довготривалого зберігання та корпоративної синхронізації

Окрім глобальних комерційних гігантів, на ринку існують рішення класу «self-hosted» (з можливістю розгортання на власних серверах), які дозволяють організаціям зберігати повний контроль над своїми даними. Найвідомішими представниками цієї категорії є системи Nextcloud та ownCloud. Вони пропонують широкі можливості: від синхронізації файлів між пристроями до інтеграції з офісними пакетами для спільного редагування документів.

Однак, незважаючи на відкритий вихідний код та високу гнучкість, такі платформи мають суттєві недоліки в контексті оптимізації ресурсів. Архітектура Nextcloud, побудована на базі стеку PHP та важких реляційних баз даних (MySQL/MariaDB або PostgreSQL), вимагає значних обчислювальних потужностей сервера та складного адміністрування. Для малого бізнесу, освітніх ініціатив або використання у складі IoT-пристроїв (наприклад, на базі одноплатних комп'ютерів) розгортання таких систем є надлишковим. Це створює попит на легковагові (lightweight) альтернативи, побудовані на сучасних мікрофреймворках [12], які можуть працювати з портативними базами даних (наприклад, SQLite) без втрати базового функціоналу розмежування доступу.

1.2.2 Аналіз сервісів експрес-обміну та тимчасового доступу (ефемерні сховища)

Другим важливим напрямком еволюції систем обміну даними є спеціалізовані сервіси для безпечної передачі великих файлів без необхідності реєстрації отримувача. Класичними представниками цієї ніші є WeTransfer, SwissTransfer та DropMeFiles. Основний принцип їхньої роботи полягає у завантаженні пакета файлів на транзитний сервер, після чого генерується унікальне посилання із заздалегідь визначеним часом життя (Time-to-Live, TTL) [1] — від кількох годин до кількох тижнів. Після завершення цього

терміну файли безповоротно видаляються з серверів.

Головною проблемою використання сторонніх публічних сервісів експрес-обміну є питання конфіденційності та цифрового суверенітету. Відправляючи корпоративні чи особисті чутливі дані через зовнішні вузли, користувач не може гарантувати їхній захист від перехоплення або аналізу на стороні провайдера послуги. Крім того, у таких сервісах відсутня єдина панель управління (дашборд), де авторизований користувач міг би одночасно керувати як своїми постійними файлами, так і тимчасовими посиланнями, переглядати статистику завантажень та за потреби достроково анулювати доступ.

Отже, аналіз існуючих рішень демонструє наявність прогалини на стику двох концепцій: ринку не вистачає легкої у розгортанні, автономної системи, яка б поєднувала надійне довготривале зберігання з вбудованим модулем генерації ефемерних посилань (з автоматичним знищенням файлів) під контролем єдиного адміністратора на власному сервері.

1.3 Обґрунтування технічних вимог до розроблюваної системи

Проведений огляд предметної області та існуючих варіантів реалізації показав, що для вирішення поставленого завдання з розробки розподіленої комп'ютерної системи зберігання цифрових ресурсів [7], наш програмно-апаратний комплекс матиме наступні особливості:

— надійне зберігання файлів на сервері та можливість безпечно їх передавати за допомогою тимчасових посилань, щоб уникнути будь-якого ризику витоку інформації.

— зручна робота з файлами, можливість додавати будь-які формати (фото, документи, архіви) та за потреби завантажувати їх назад усі разом одним автоматично створеним ZIP-архівом.

— зручний і зрозумілий вебінтерфейс, який підлаштовується під

будь-які екрани і працює виключно в темній темі (Dark Mode), щоб очі не втомлювалися під час довгої роботи.

Технічні вимоги до апаратної частини сервера та мережевої інфраструктури формуються з огляду на необхідність забезпечення безперебійного доступу та швидкої обробки I/O операцій. Основні апаратні та системні вимоги є наступними:

— У ролі фізичного сервера використовується компактна плата Orange Pi One. Вибір зумовлений оптимальним співвідношенням енергоефективності та продуктивності для завдань хостингу легковагових вебдодатків на базі Flask.

— Orange Pi One [16] снащений чотириядерним процесором Allwinner H3 (архітектура ARM Cortex-A7). Це дозволяє стабільно обробляти паралельні HTTP-запити, виконувати криптографічне хешування паролів та керувати базою даних SQLite без критичних затримок [20].

— Для забезпечення високої пропускної здатності та мінімального пінгу при передачі великих обсягів даних (завантаження та вивантаження файлів) сервер підключається до високошвидкісного інтернет-каналу за допомогою оптоволоконного з'єднання (FTTH/PON) [1].

— Для локального контролю за станом системи передбачено використання інформаційного мінідисплея, підключеного до сервера. На нього виводитиметься критично важлива телеметрія у реальному часі (IP-адреса, навантаження на процесор, температура та залишок вільного дискового простору).

— Створення серверної логіки, написання скриптів на Python, конфігурація бази даних та компіляція фронтенд-компонентів виконується у локальному середовищі на базі 13-дюймового ноутбука з процесором Apple M1, що забезпечує високу швидкість обробки коду та тестування мікросервісної архітектури перед розгортанням на робочому сервері.

РОЗДІЛ 2 ПРОЄКТНА ЧАСТИНА

2.1 Апаратна конфігурація та периферійне забезпечення вузла комп'ютерної системи

Фізична архітектура локального вузла розподіленої системи зберігання даних побудована на базі одноплатного мікрокомп'ютера Orange Pi One, який виконує роль центрального контролера. Апаратний комплекс складається з кількох взаємопов'язаних підсистем: обчислювального ядра, підсистеми зберігання даних, модуля візуалізації телеметрії та підсистеми енергозабезпечення.

Основою обчислювальної підсистеми є SoC (System-on-a-Chip) Allwinner H3. Дана мікросхема інтегрує в собі чотириядерний процесор архітектури ARM Cortex-A7, апаратний криптографічний співпроцесор та контролери периферійних шин. Оперативна пам'ять обсягом 512 МБ стандарту DDR3 розпаяна безпосередньо на платі та підключена до процесора загальною шиною даних, що мінімізує затримки при обміні інформацією. Для комунікації із зовнішніми пристроями використовується 40-контактна гребінка GPIO (General-Purpose Input/Output), яка забезпечує апаратну підтримку протоколів низького рівня: I2C, SPI, UART та ШІМ (PWM).

Архітектура пам'яті вузла розділена на два фізичні рівні для оптимізації швидкодії та підвищення надійності.

Для завантаження операційної системи (Boot ROM) і зберігання системних файлів ядра Linux використовується карта пам'яті microSD, підключена через інтерфейс SDIO. Такий підхід дозволяє швидко відновлювати працездатність вузла шляхом заміни системного образу без втрати користувацьких даних.

Для фізичного зберігання цифрових ресурсів користувачів використовується зовнішній твердотільний або магнітний накопичувач, підключений через інтерфейс USB 2.0 (Host). Використання виділеного апаратного USB-контролера дозволяє ізолювати операції інтенсивного вводу-виводу (I/O) користувацьких файлів від системних процесів ОС, що працюють з microSD, тим самим подовжуючи життєвий цикл флеш-пам'яті (wear leveling).

Для забезпечення апаратного моніторингу стану вузла до системи інтегровано OLED-дисплей (на базі контролера SSD1306 з діагоналлю 0.96 дюйма). Інтеграція здійснена за допомогою двопровідної послідовної шини I2C. Апаратне підключення реалізовано через контакти роз'єму GPIO:

— sda (serial data) – лінія передачі даних (фізичний пін 3 на платі);

— scl (serial clock) – лінія тактування (фізичний пін 5 на платі);

— лінії живлення 3.3V та заземлення.

Використання шини I2C дозволяє обмінюватися даними з дисплеєм на частоті до 400 кГц (Fast mode), що не створює значного переривання для центрального процесора. Це інженерне рішення перетворює вузол на повністю автономний пристрій, здатний виводити критичні параметри (температуру кристала процесора, поточну IP-адресу, статус монтування USB-диска) безпосередньо на фізичний екран.

Стабільність роботи мікрокомп'ютерної системи, особливо при операціях запису на зовнішні USB-накопичувачі, критично залежить від якості живлення. Orange Pi One використовує спеціалізований коаксіальний роз'єм живлення (Barrel Jack 4.0x1.7 мм) замість стандартного microUSB, що є важливою інженерною перевагою. Це дозволяє пропускати струм до 2А при напрузі 5В без значних втрат та просадок напруги, які притаманні конекторам microUSB.

На платі інтегровано контролер живлення (PMIC – Power Management IC), який динамічно регулює напругу на ядрах процесора залежно від навантаження (технологія DVFS – Dynamic Voltage and Frequency Scaling).

Оскільки при інтенсивній мережевій передачі файлів чіп Allwinner H3 схильний до тепловиділення, апаратна конфігурація передбачає встановлення пасивного алюмінієвого радіатора на корпус SoC для відведення тепла та запобігання тепловому тролінгу (thermal throttling), що гарантує безперервну роботу системи в режимі 24/7.

2.2 Розробка структури комп'ютеризованої системи зберігання цифрових ресурсів

Для побудови надійної розподіленої комп'ютеризованої системи необхідно визначити комплексні вимоги до її апаратного та мікропрограмного забезпечення. Проєктований вузол зберігання та обміну даними потребує:

— енергоефективної та стабільної мікропроцесорної платформи для безперервної обробки переривань та мережевих запитів;

— оптимізованого мікропрограмного середовища для прямої маршрутизації вводу-виводу (I/O) та управління потоками даних;

— локальної системи управління базами даних для транзакційного контролю цілісності файлової системи;

— розробки повного комплекту конструкторської документації для опису фізичної та логічної архітектури пристрою.

Під час проєктування було підібрано необхідну елементну базу та розгорнуто середовище для написання низькорівневого програмного коду. Зважаючи на жорсткі вимоги до паралельної обробки даних без перевантаження шини пам'яті, обчислювальним ядром системи було обрано мікрокомп'ютер Orange Pi One. Такий вибір насамперед зумовлений оптимальним співвідношенням продуктивності чипа SoC Allwinner H3 та низького енергоспоживання, що є вкрай критичним параметром для автономних хмарних вузлів.

Для організації апаратного моніторингу та локального інформування використано OLED-дисплей, підключений по шині I2C. Це інженерне рішення

дозволяє адміністратору фізично відстежувати критичну телеметрію, таку як IP-адреса, поточне навантаження на процесор і пам'ять, а також статус накопичувача, уникаючи необхідності щоразу ініціалізувати віддалене SSH-з'єднання.

Мікропрограмне керування серверною частиною реалізовано за допомогою фреймворку Flask мовою Python. Він був обраний завдяки відсутності надлишкових модулів, що дає змогу використовувати його як максимально ефективний маршрутизатор системних викликів в умовах обмеженої оперативної пам'яті обсягом 512 МБ. Роль системи управління метаданими виконує реляційна СУБД SQLite, яка реалізує концепцію безсерверної архітектури. Зберігання бази даних у вигляді єдиного локального файлу суттєво мінімізує використання оперативної пам'яті та гарантує цілісність даних у випадку раптових збоїв електроживлення.

Розробка та крос-компіляція мікропрограмного забезпечення здійснювалася на хост-машині з ARM-сумісною архітектурою Apple M1, що дозволило значно оптимізувати процес налагодження перед остаточним розгортанням коду на цільовій платформі Orange Pi. На додаток, для повноцінного інженерного опису розроблюваної системи було створено комплекс схем, які детально відображають проєкт на різних рівнях абстракції — від загальної концепції до фізичних електричних з'єднань.

2.2.1 Структурна організація системи

Визначає основні функціональні частини системи, їх призначення та інформаційні зв'язки між ними на найвищому рівні. На структурній схемі відображено ієрархію та підключення апаратних модулів обчислювального вузла. Центральним елементом виступає мікрокомп'ютер Orange Pi One, який виконує роль головного контролера та здійснює маршрутизацію через порти вводу-виводу.

Наведена схема наочно демонструє розподіл периферійних пристроїв та мережевих вузлів за типами апаратних інтерфейсів (див. рис. 2.1).

Безперебійна робота вузла підтримується підсистемою живлення, яка забезпечується стабілізованим джерелом постійного струму з параметрами 5В/2А через лінію DC. Лівий блок схеми відповідає за підсистему вводу та зберігання і включає фізичну кнопку керування, підключену через цифровий інтерфейс Digital, HDD-накопичувач для збереження масивів користувацьких даних за допомогою інтерфейсу USB, а також допоміжний модуль бездротового зв'язку Wi-Fi ESP8266, який комунікує з контролером по послідовній шині UART. Своєю чергою правий блок формує підсистему виводу та мережевої взаємодії, де критична телеметрія транслюється на локальний OLED-дисплей через двопровідну шину GPIO (I2C), тоді як зовнішня маршрутизація даних і надання доступу до вебсайту здійснюються за захищеним протоколом HTTPS через бездротове Wi-Fi з'єднання.

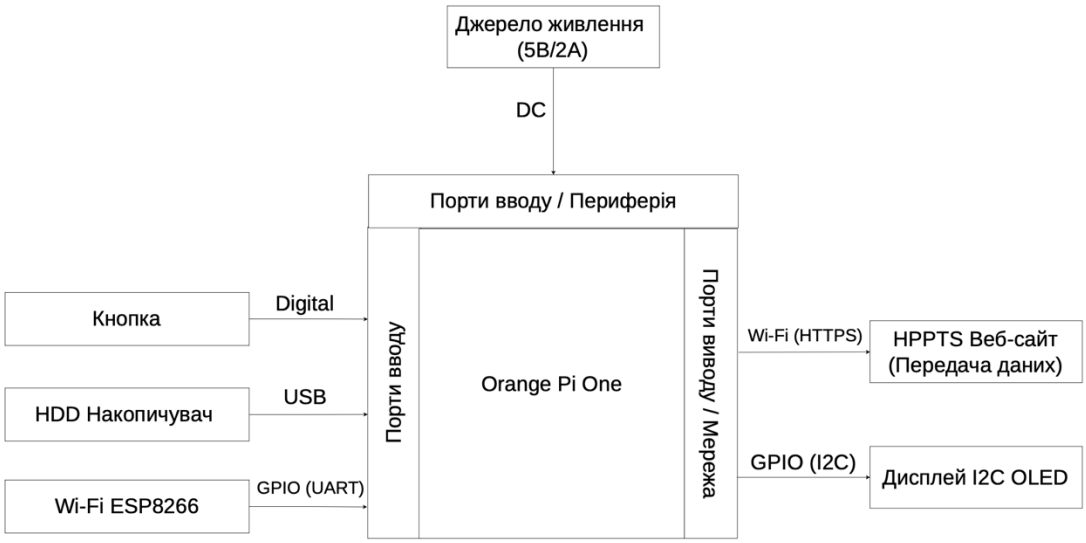

Рисунок 2.1 – Структурна схема системи

Ця схема дозволяє оцінити загальну архітектуру розподіленої системи та логіку взаємодії модулів без заглиблення у низькорівневу схемотехніку.

2.2.2 Взаємодія програмних компонентів

Відображає архітектуру мікропрограмного забезпечення та ієрархію його базових функціональних модулів. Схема деталізує процеси обробки інформації на логічному рівні: маршрутизацію HTTP-запитів через мікрофреймворк Flask, алгоритми керування життєвим циклом ефемерних файлів, безпечну взаємодію з реляційною базою даних SQLite та механізми контролю доступу (див. рис. 2.2).

Рисунок 2.2 – Структурна схема програмного забезпечення

Вона пояснює, яким чином серверні скрипти розподіляють обчислювальні задачі між підсистемами та ініціюють системні виклики до апаратної частини (взаємодія з файловою системою зовнішнього накопичувача та виведення телеметрії на I2C-дисплей) для забезпечення стабільної автономної роботи вузла.

2.2.3 Принципи електричного підключення

Цей підрозділ надає вичерпну інформацію про фізичну конфігурацію пристрою безпосередньо на рівні радіоелектронних компонентів та ліній зв'язку. Зокрема, на принциповій схемі детально відображено трасування ліній живлення від понижуючого DC-DC перетворювача на базі модуля LM2596 до контактів живлення мікрокомп'ютера та супутньої периферії (див. рис. 2.3). Крім того, графічно проілюстровано фізичне підключення ліній синхронізації

та даних цифрової шини I2C до відповідних пінів інтерфейсу GPIO. Окрему увагу приділено правильному розведенню диференційних пар стандарту USB, що є критично необхідним для стабільної та коректної роботи підсистеми зберігання даних.

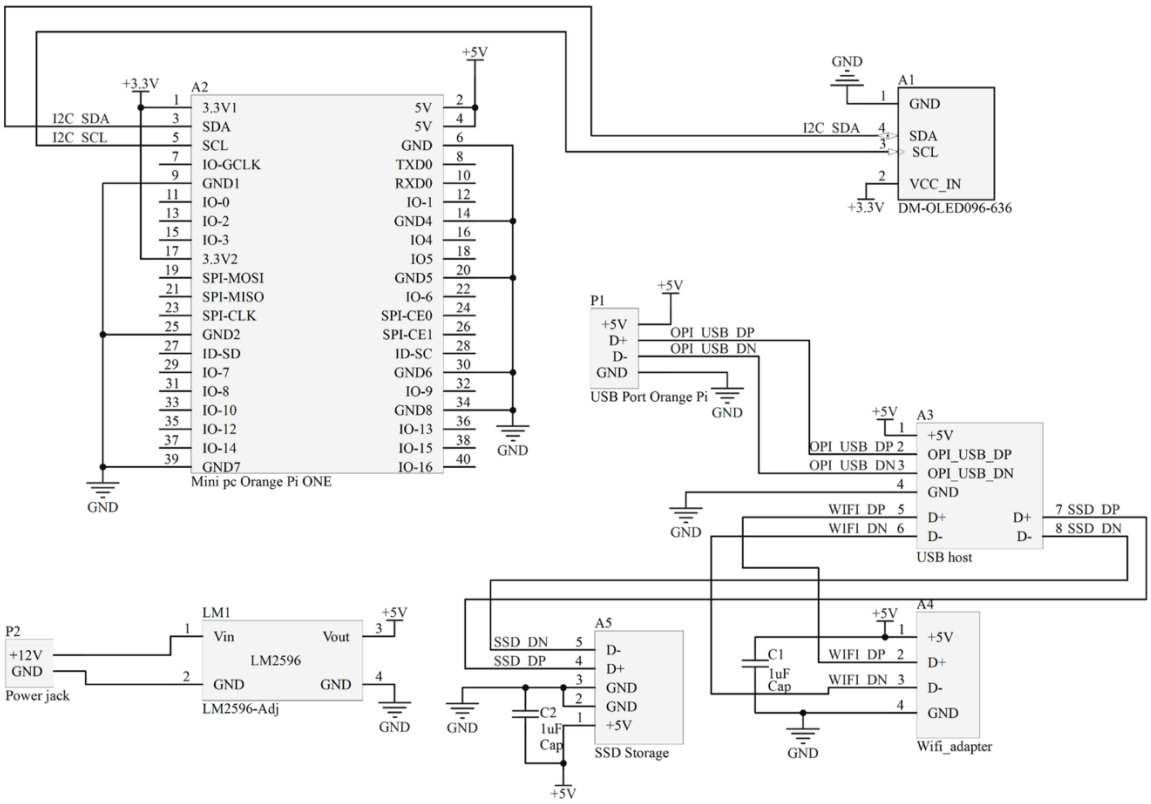

Рисунок 2.3 – Електрична принципова схема системи

Електрична принципова схема є базовим документом для фізичного збирання, діагностики та ремонту апаратної частини системи.

2.2.4 Розробка блок-схеми алгоритму мікропрограмного забезпечення

Наведена блок-схема графічно описує логіку роботи серверної частини комплексу та покроково ілюструє повний життєвий цикл обробки інформації (див. рис. 2.4). Цей процес охоплює всі ключові етапи функціонування системи, починаючи від стартової ініціалізації апаратних інтерфейсів і підключення бази даних SQLite. Далі алгоритм детально розкриває послідовність виконання складніших операцій, серед яких реалізація

паралельного запису файлів на зовнішній USB-накопичувач рівними частинами (чанками), а також запуск системної процедури автоматичного очищення та звільнення дискового простору.

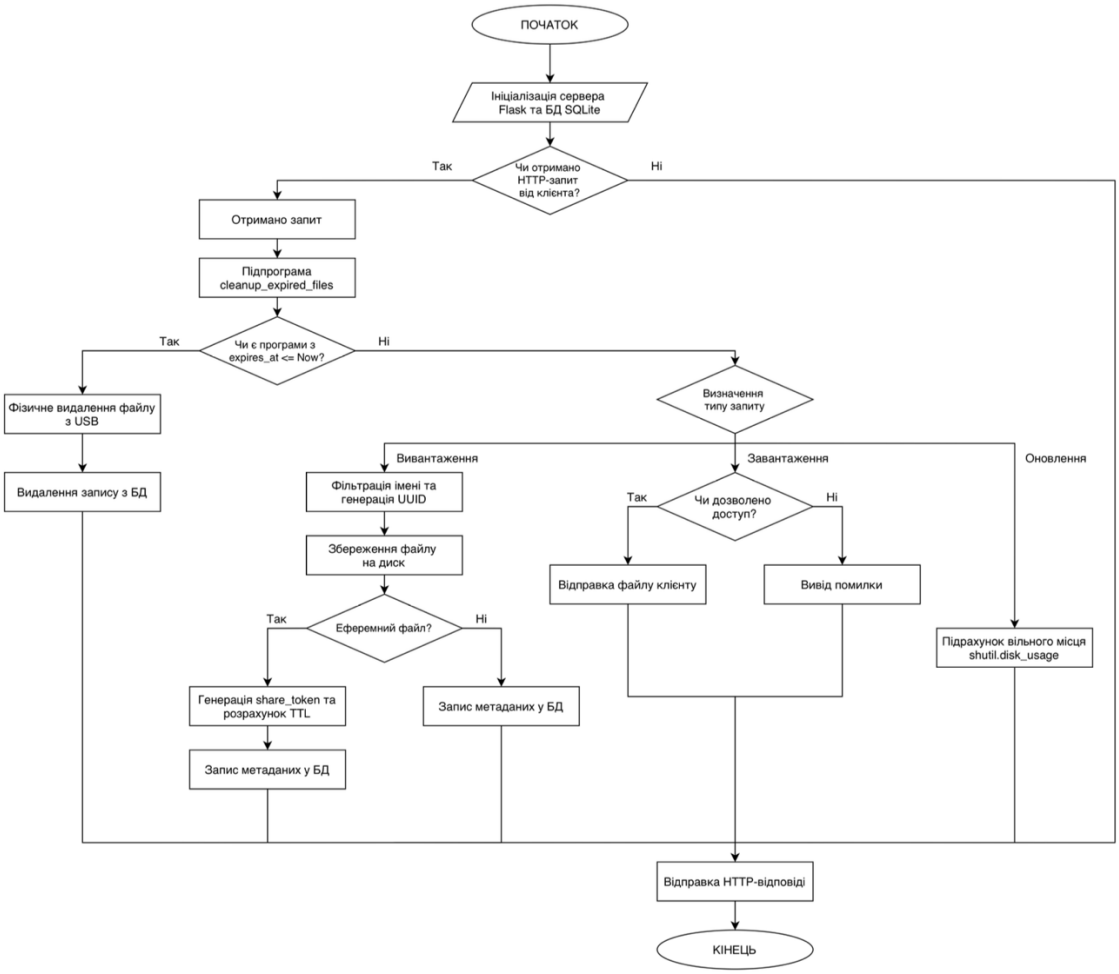

Рисунок 2.4 – Блок-схема алгоритму роботи серверної частини вузла

Вона демонструє механізми обробки виняткових ситуацій (наприклад, втрати зв'язку з накопичувачем) та забезпечення стабільності вузла.

2.3 Обґрунтування обраного апаратного та програмного забезпечення для проєктування комп’ютеризованої системи

Ефективність функціонування розподіленої комп'ютерної системи зберігання цифрових ресурсів безпосередньо залежить від раціонального

підбору обчислювальних модулів, конфігурації апаратних інтерфейсів та оптимізації низькорівневого програмного стеку. Оскільки до проєкту висуваються жорсткі вимоги щодо автономності, апаратної стабільності мережевого з'єднання, швидкодії обробки потоків даних та енергоефективності вузла за умов обмеженого бюджету, було проведено детальний архітектурно-технічний аналіз компонентів розроблюваного комплексу.

З метою технічного обґрунтування вибору фізичної обчислювальної платформи для розгортання локального сервера було виконано порівняльний аналіз найпопулярніших на ринку одноплатних мікрокомп'ютерів (Single Board Computers). Оскільки розроблюваний вузол системи має стабільно функціонувати в умовах постійного мережевого навантаження, вибір апаратного забезпечення вимагав ретельного підходу до оцінки апаратних можливостей кожного кандидата.

Під час проведення аналізу ключовими інженерними критеріями виступали архітектура мікропроцесорного ядра (SoC), яка відповідає за швидкість обробки програмних запитів та шифрування, а також обов'язкова наявність інтегрованого контролера мережі для гарантування безперебійного фізичного зв'язку без втрат пакетів. Важливу роль під час відбору також відігравали конфігурація цифрових ліній вводу-виводу (GPIO) для взаємодії з периферією, рівень енергоспоживання (TDP) разом із вимогами до систем охолодження, та економічна доцільність придбання плати у контексті потенційного масштабування проєкту. Підсумкові результати проведеного порівняння зведені у таблиці 2.1.

Таблиця 2.1 – Аналіз характеристик мікрокомп'ютерів

Характеристика / Критерій	Orange Pi One (Обрана платформа)	Raspberry Pi 4 Model B	Raspberry Pi Zero 2 W	Orange Pi Zero
Процесор (CPU)	Allwinner H3 (4 ядра Cortex-A7)	Broadcom BCM2711 (4 ядра Cortex-A72)	Broadcom BCM2710A1 (4 ядра Cortex-A53)	Allwinner H2+ (4 ядра Cortex-A7)
Тактова частота	1.2 ГГц	1.5 ГГц	1.0 ГГц	1.2 ГГц
Оперативна пам'ять (RAM)	512 МБ DDR3	Від 2 до 8 ГБ LPDDR4	512 МБ LPDDR2	256 / 512 МБ DDR3
Мережеві інтерфейси	10/100M Ethernet	Gigabit Ethernet, Wi-Fi 5	Тільки Wi-Fi 4 та Bluetooth	10/100M Ethernet, Wi-Fi
Енергоспоживання	Низьке (~2-3 Вт)	Високе (~3.5-7.5 Вт), потребує охолодження	Дуже низьке (~1.5 Вт)	Низьке (~2 Вт)
Орієнтовна вартість	Низька (Бюджетний сегмент)	Висока (Преміум сегмент)	Середня	Низька
Оцінка для проєкту	Оптимальний вибір. Баланс ціни, наявності Ethernet та потужності для Flask/SQLite.	Надлишкова потужність. Висока ціна та нагрів невиправдані для легковагового бекенду.	Не підходить. Відсутність порту Ethernet знижує стабільність передачі файлів.	Застаріла платформа. Гірший тепловідвід порівняно з версією One.

Як видно з таблиці 2.1, мікрокомп'ютер Raspberry Pi 4 володіє надлишковими апаратними ресурсами для локального вузла зберігання, що суттєво підвищує енергоспоживання системи та вимагає додаткових схем активного тепловідому. З іншого боку, архітектура Raspberry Pi Zero 2 W позбавлена вбудованого проводового інтерфейсу Ethernet, через що передача великих масивів даних створювала б значне навантаження на програмний стек бездротового зв'язку та центральний процесор.

У підсумку, базовим обчислювальним компонентом обрано плату Orange Pi One. Обчислювального потенціалу чипа Allwinner H3 (архітектура Cortex-A7) цілком достатньо для паралельної обробки переривань від мережевого контролера, конвеєризації HTTP-запитів легковагового бекенду, а також виконання криптографічних алгоритмів хешування та динамічного

стиснення даних без перевантаження шини пам'яті DDR3.

Для забезпечення автономності функціонування та візуального контролю стану апаратної частини без використання мережевого стеку, до системи інтегровано локальний інформаційний мінідисплей (на базі контролера SSD1306 або аналогічного). Підключення дисплея реалізовано на фізичному рівні за допомогою послідовної синхронної шини I2C (Inter-Integrated Circuit), що використовує лінії GPIO мікрокомп'ютера (сигнали SDA та SCL). Таке інженерне рішення дозволяє безпосередньо зчитувати дані з регістрів ОС та виводити критичну телеметрію (поточну конфігурацію IP-адреси, апаратну температуру SoC, ступінь завантаження ядер процесора та обсяг вільного дискового простору на змонтованому USB-накопичувачі) у реальному часі. Це гарантує можливість фізичного моніторингу пристрою навіть у разі повної відсутності доступу по віддалених протоколах керування типу SSH.

Особлива увага приділена мережевій підсистемі вузла. Для виключення затримок при комутації пакетів даних і забезпечення максимальної пропускної здатності, пристрій інтегрується у локальну мережу через вбудований інтерфейс RJ-45, який взаємодіє з інтегрованим медіа-контролером (EMAC) та мікросхемою фізичного рівня (PHY) трансивера. Підключення базової інфраструктури до зовнішньої мережі за технологією оптоволоконного зв'язку (FTTH/PON) мінімізує апаратний джиттер (втрату пакетів) і забезпечує стабільну симетричну швидкість приймання-передачі файлових потоків, що є критично важливим для надійної роботи розподілених комп'ютерних систем.

2.3.1 Програмне забезпечення та середовище розробки

Реалізація програмної логіки за допомогою Python та Flask пояснюється вкрай низьким рівнем накладних витрат цієї зв'язки. Відмова від масивних монолітних фреймворків дозволила уникнути завантаження непотрібних вбудованих модулів, які б невиправдано споживали жорстко обмежений обсяг оперативної пам'яті пристрою. Водночас базова екосистема Flask надає всі

необхідні механізми для безпечної та швидкої роботи системи. Зокрема, вона пропонує гнучкі інструменти для мережевої маршрутизації, обробки користувацьких сесій, надійного криптографічного захисту паролів та маніпуляцій із файловими архівами в оперативній пам'яті завдяки відповідним бібліотекам.

В якості системи управління базами даних (СУБД) обрано реляційну базу даних SQLite. Оскільки архітектура системи передбачає зберігання структурованих метаданих файлів (оригінальна назва, UUID токен, розмір, дата завантаження, статус ефемерності), SQLite є ідеальним рішенням завдяки концепції "serverless". Вся база даних зберігається в одному локальному файлі users.db, не потребує запуску окремого фонового процесу та мінімізує використання оперативної пам'яті сервера, забезпечуючи при цьому повну підтримку транзакцій (ACID) та високу швидкість виконання SQL-запитів.

Інтерфейс користувача (Frontend) побудовано за допомогою сучасних вебстандартів HTML5, CSS3 та JavaScript з використанням векторних іконок Lucide. Дизайн реалізовано у темних відтінках (Dark Mode) для зниження зорового навантаження оператора та забезпечення адаптивності на різних пристроях.

Розробка та первинне налагодження всього програмного комплексу (написання серверних скриптів, проєктування структури таблиць БД, верстка адаптивного інтерфейсу та тестування логіки автоматичного видалення файлів за TTL) здійснювалися у локальному середовищі розробника на базі 13-дюймового ноутбука з процесором Apple M1. Висока продуктивність архітектури Apple Silicon дозволила оперативно проводити налагодження програмного коду за допомогою вбудованих інструментів дебагу Flask та аналізувати terminal-логи, що значно прискорило процес підготовки стабільної версії програмного продукту перед його розгортанням на сервері Orange Pi One.

2.4 Проєктування бази даних системи

Основою будь-якої системи керування цифровими ресурсами є надійна архітектура зберігання метаданих. Враховуючи вимоги до легкості розгортання та автономності, для проєкту було обрано реляційну систему управління базами даних SQLite. Уся структура зберігається у єдиному локальному файлі users.db, що забезпечує швидкий доступ та відсутність мережевих затримок при виконанні SQL-запитів.

Логічна структура бази даних спроєктована з урахуванням нормалізації та складається з двох основних пов'язаних сутностей (таблиць): users (користувачі) та files (файли).

2.4.1 Структура таблиці користувачів (users)

Таблиця users відіграє ключову роль у підсистемі автентифікації та призначена для надійного зберігання облікових даних адміністраторів і користувачів, а також керування їхніми статусами доступу. Кожен обліковий запис у системі ідентифікується за допомогою поля id, яке слугує первинним ключем та унікальним ідентифікатором із параметром автоінкременту. Для безпосередньої автентифікації та входу в систему використовується обов'язковий атрибут username, що містить унікальний логін користувача.

Особливу увагу під час проєктування цієї таблиці було приділено питанням кібербезпеки. З цих міркувань система принципово не зберігає паролі у відкритому вигляді — натомість у полі password міститься лише криптографічний хеш пароля. Для його генерації та перевірки використовується надійний алгоритм хешування з вбудованої бібліотеки werkzeug.security. Додатково для гнучкого адміністрування передбачено поле is_blocked, яке виконує роль логічного прапорця стану акаунта. Цей атрибут дозволяє адміністратору системи миттєво блокувати доступ для підозрілих облікових записів, ізолюючи їх від системи без необхідності фізичного видалення самого профілю та пов'язаної з ним історії дій.

2.4.2 Структура таблиці файлів (files)

Таблиця files виступає ядром системи моніторингу та розмежування доступу. Вона не містить самих бінарних файлів, які фізично зберігаються у файловій системі за адресою /mnt/usb_drive, проте акумулює всі критично важливі метадані про них. Кожен запис у базі ідентифікується за допомогою унікального поля id з властивістю автоінкременту. Для забезпечення суворого розмежування прав використовується зовнішній ключ user_id, який вказує на конкретного власника ресурсу та пов'язує запис із таблицею users. Завдяки цьому гарантується фундаментальне правило безпеки: користувач має змогу бачити та керувати виключно власними даними.

Інформація про файли структурно розділена для зручності як користувача, так і системи. Атрибут original_name зберігає початкову назву документа під час завантаження (наприклад, document.pdf) для коректного та звичного відображення в інтерфейсі. Натомість поле physical_name містить унікальне ім'я на фізичному диску, згенероване системою на базі алгоритмів UUID, що дозволяє повністю уникнути конфліктів у разі випадкового збігу назв завантажених файлів. Для ведення аналітики та виведення статистики використання дискового простору на дашборді фіксується розмір кожного файлу у байтах (поле size), а також точний час і дата його потрапляння на сервер (upload_date).

Окремий функціональний блок таблиці відповідає за гнучке керування публічним доступом. Базовий статус файлу визначається прапорцем is_public, який вказує, чи є ресурс приватним, чи доступним для інших користувачів. Під час генерації публічного посилання створюється унікальний криптографічний ключ share_token, який забезпечує безпечний доступ до ресурсу. Крім того, архітектура передбачає можливість обмеження часу дії посилань (Time-to-Live) через атрибут expires_at. Саме на це поле орієнтується системний модуль cleanup_expired_files() для ідентифікації застарілих даних. Паралельно з цим параметр auto_delete задає системі подальший алгоритм дій після вичерпання

часового ліміту: він вказує, чи необхідно безповоротно та фізично знищити файл із накопичувача, чи достатньо лише відкликати публічний доступ, залишивши файл у приватному сховищі власника.

Загалом зв'язок між таблицями users та files спроєктовано за типом «один-до-багатьох» (1:N), що є логічним відображенням життєвого циклу даних: один користувач може завантажувати необмежену кількість файлів, але кожен конкретний файл завжди належить суворо одному власнику. Такий підхід до проєктування бази даних гарантує високу цілісність інформації, оптимізує швидкість пошуку за токенами доступу та створює надійний захист від несанкціонованої взаємодії з чужими ресурсами.

РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА

3.1 Реалізація апаратно-програмної взаємодії та архітектури вузла

Базовим елементом запропонованої інфраструктури виступає серверний вузол на основі мікрокомп'ютера Orange Pi One. Для забезпечення стабільної роботи в умовах дефіциту оперативної пам’яті та зниження навантаження на обчислювальне ядро SoC Allwinner H3, програмну складову реалізовано за допомогою легковагового стеку Python та Flask. Архітектура застосунку використовує суворе розділення відповідальності: підсистеми маршрутизації трафіку, доступу до накопичувачів та управління периферійними екранами функціонують як ізольовані модулі [7].

Процес ініціалізації вузла починається з підключення потрібних системних бібліотек для безпосередньої взаємодії з апаратними інтерфейсами вводу-виводу та операційною системою. Важливим етапом є конфігурація точки монтування зовнішнього фізичного накопичувача, підключеного через шину USB 2.0 плати Orange Pi [16]. Оскільки вбудована пам'ять мікрокомп'ютера використовується переважно для ОС, система автоматично перевіряє стан монтування USB-диска і створює цільові директорії. Це забезпечує відмовостійкість вузла при апаратних збоях живлення або «гарячому» перепідключенні накопичувача (див. рис. 3.1).

```python
from flask import Flask, request, render_template, jsonify, send_from_directory, session, send_file
from werkzeug.utils import secure_filename
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime, timedelta
import os
import shutil
import sqlite3
import uuid
import zipfile
import io

app = Flask(__name__)
app.secret_key = 'super_secret_kvitova_key'

UPLOAD_FOLDER = '/mnt/usb_drive'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
```

Рисунок 3.1 – Лістинг ініціалізації середовища Flask та підключення системних бібліотек

Невід'ємною складовою апаратної взаємодії є виведення системної інформації на під'єднаний зовнішній дисплей. Під час ініціалізації запускається фоновий процес, який опитує апаратні сенсори плати (температуру процесора, завантаженість RAM) та мережеві інтерфейси [7] (поточну IP-адресу вузла), а також відстежує обсяг вільної пам'яті на зовнішньому накопичувачі. Зібрані дані динамічно транслюються на екран, що дозволяє здійснювати фізичний моніторинг стану пристрою в режимі реального часу.

Управління метаданими реалізовано через СУБД SQLite (users.db). Враховуючи специфіку роботи з Flash-пам'яттю (microSD-карткою, з якої завантажується ОС) [14], оптимізація транзакцій до бази даних є критичною для зменшення зносу (wear leveling) комірок пам'яті. Для оновлення схеми бази даних застосовується механізм міграції на основі програмного перехоплення операційних помилок. Такий підхід дає змогу динамічно адаптувати архітектуру таблиць під нові вимоги без ресурсоємного повного перезапису файлу, що, у свою чергу, суттєво подовжує експлуатаційний ресурс фізичного накопичувача.

Під час завантаження нових даних вузол обробляє файлові потоки з урахуванням апаратних обмежень оперативної пам'яті. Файли приймаються

через HTTP POST-запит і записуються на зовнішній USB-накопичувач чанками (частинами), оминаючи тривале буферизування в RAM. Кожен файл проходить санітизацію імені (secure_filename) та отримує унікальний ідентифікатор (uuid.uuid4()), що унеможливлює колізії у файловій системі FAT32/ext4 підключеного диска.

Управління дисковим простором на фізичному рівні реалізовано через функцію cleanup_expired_files(). Оскільки ємність підключеного носія є жорстко обмеженим ресурсом, алгоритм регулярно вивільняє блоки пам'яті. Після вичерпання часу доступу (expires_at) та за умови активного прапорця auto_delete, система виконує системний виклик os.remove. Це призводить до фізичного стирання даних з сектора накопичувача, зменшує навантаження на дискову підсистему і одночасно оновлює статистику вільного місця на під'єднаному дисплеї.

3.2 Розробка інтерфейсу користувача та клієнтської логіки

Клієнтська частина (Frontend) спроєктована з фокусом на ергономіку, швидкодію та мінімізацію зорового навантаження [2]. Інтерфейс побудовано з використанням сучасних стандартів HTML5, CSS3 та JavaScript, без залучення важких фронтенд-фреймворків, що забезпечує миттєве завантаження сторінок [10].

Дизайн реалізовано виключно у темній темі (Dark Mode) з використанням системи CSS-змінних. Для візуалізації елементів керування інтегровано бібліотеку векторних іконок Lucide, яка рендериться на стороні клієнта за допомогою JavaScript (див. рис. 3.2).

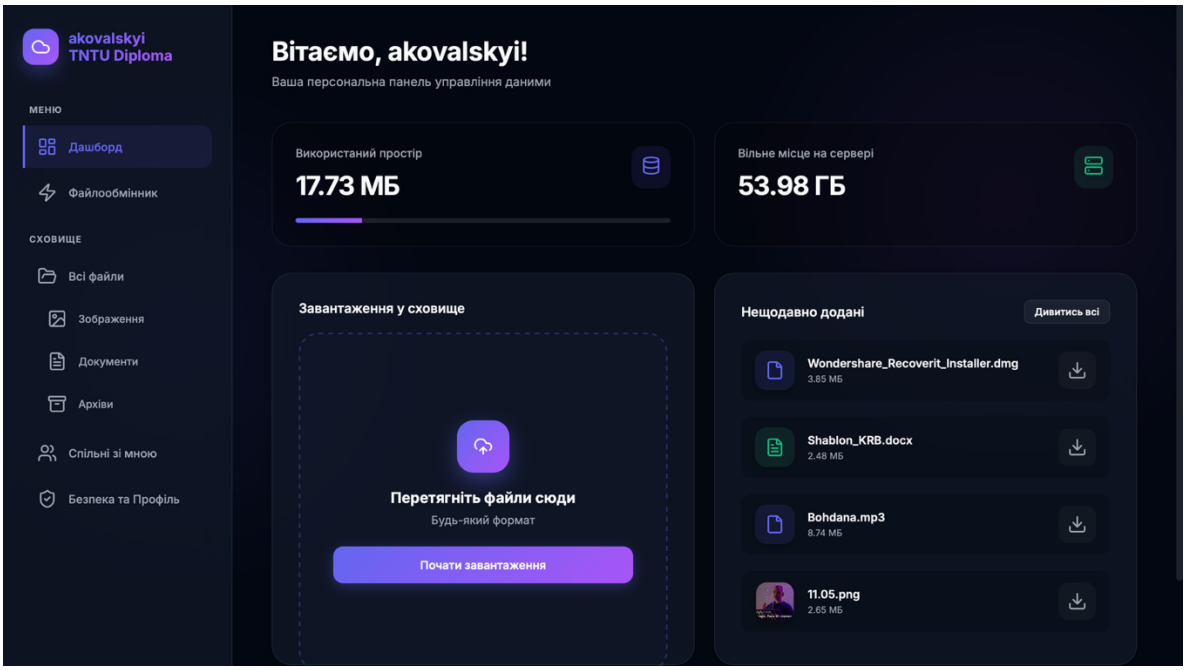

Рисунок 3.2 – Інтерфейс головної панелі керування (Dashboard)

Динамічна взаємодія користувача з системою забезпечується набором JavaScript-функцій. Зокрема, у файлі share.html реалізовано клієнтський таймер зворотного відліку, який вираховує різницю між поточним часом і значенням expires_at. Коли час вичерпується, скрипт автоматично блокує кнопки завантаження та виводить повідомлення «Файли знищено системою безпеки», що синхронізується з серверною логікою видалення [9].

Крім того, у файлі index.html реалізовано обробник подій, який здійснює безперервний моніторинг стану підключення. У разі втрати зв'язку з Інтернетом інтерфейс блокується спеціальним екраном, що запобігає спробам завантаження файлів "у порожнечу" та виникненню мережевих помилок [1] (див. рис. 3.3).

```
function updateTimer() {
    const diff = expireDate - new Date();
    const textEl = document.getElementById('timeLeftText');

    if (diff <= 0) {
        textEl.innerText = "Файли знищено системою безпеки";
        document.querySelector('.btn-primary').style.display = 'none';
        const secBtn = document.querySelector('.btn-secondary');
        if(secBtn) secBtn.style.display = 'none';
        return;
    }

    const days = Math.floor(diff / (1000 * 60 * 60 * 24));
    const hours = Math.floor((diff / (1000 * 60 * 60)) % 24);
    const minutes = Math.floor((diff / 1000 / 60) % 60);

    let timeStr = "";
    if (days > 0) timeStr = `${days} дн. ${hours} год.`;
    else if (hours > 0) timeStr = `${hours} год. ${minutes} хв.`;
    else timeStr = `${minutes} хв.`;

    if (isAutoDelete) textEl.innerHTML = `Автознищення через: <b>${timeStr}</b>`;
    else textEl.innerHTML = `Доступ активно ще: <b>${timeStr}</b>`;
}

updateTimer();
setInterval(updateTimer, 60000);
```

Рисунок 3.3 – Лістинг JavaScript-коду таймера знищення та виявлення статусу мережі

Після відновлення мережевого з'єднання система перехоплює зворотну подію online та автоматично деактивує захисний екран, дозволяючи оператору продовжити роботу без необхідності перезавантаження сторінки.

3.3 Інженерне розгортання системи на мікрокомп'ютері Orange Pi One

Після завершення розробки та успішного тестування у локальному середовищі на архітектурі Apple Silicon (ноутбук з процесором M1), програмний комплекс було перенесено на цільову апаратну платформу — мікрокомп'ютер Orange Pi One [16].

Для забезпечення максимальної продуктивності сервера базовою операційною системою було обрано дистрибутив Armbian (Server Linux) [14], а сам процес інженерного розгортання відбувався у кілька послідовних етапів. Насамперед виникла необхідність підготовки дискового простору. Оскільки внутрішня пам'ять MicroSD має малий об'єм, для збереження файлів користувачів довелося підключити зовнішній накопичувач, який за

допомогою утиліт відформатували у журнальовану файлову систему. Після цього накопичувач змонтували у директорію /mnt/usb_drive. Щоб гарантувати його автоматичне підключення після кожного перезавантаження сервера, відповідний запис додали до конфігураційного файлу за унікальним ідентифікатором пристрою (UUID) [14]. На завершальному етапі було проведено налаштування прав доступу, де для уникнення помилок при записі даних (Permission Denied) власнику процесу Flask [11] надали відповідні права на цільову директорію за допомогою системних команд chown та chmod 755.

Для того, щоб вебсистема працювала в автономному режимі 24/7, було створено системну службу tntu_storage.service під управлінням демона Systemd [14]. Конфігурація служби передбачає автоматичний запуск додатка app.py [12] після ініціалізації мережевих інтерфейсів та успішного монтування зовнішнього накопичувача. У разі критичної помилки або падіння процесу, Systemd автоматично перезапускає вебсервер параметром Restart=always.

3.4 Відладка, тестування та обробка помилок

На фінальному етапі було проведено комплексне функціональне та навантажувальне тестування системи у реальних умовах експлуатації через оптоволоконне з'єднання [9].

Під час стрес-тестування завантаження великих обсягів даних (відеофайли понад 5 ГБ) було виявлено відмову сервера через перевищення ліміту розміру тіла запиту. Проблему було локалізовано та усунуто шляхом збільшення системного параметра у конфігурації Flask [11].

Окрема увага приділялася тестуванню глобальних обробників помилок (див. рис. 3.4), що дозволило перевірити систему на стійкість до невалідних HTTP-запитів та спроб сканування директорій [13]. Важливою особливістю застосунку є те, що при виникненні позаштатних ситуацій він продовжує стабільно функціонувати, просто перенаправляючи користувача на єдину стилізовану сторінку error.html. Такий підхід забезпечує коректну обробку

цілого ряду проблем, серед яких помилка 404 у разі звернення до відсутніх сторінок чи файлів, помилка 500 при внутрішніх серверних збоях, а також помилка 413, коли користувач намагається завантажити файл, розмір якого перевищує встановлені ліміти [12].

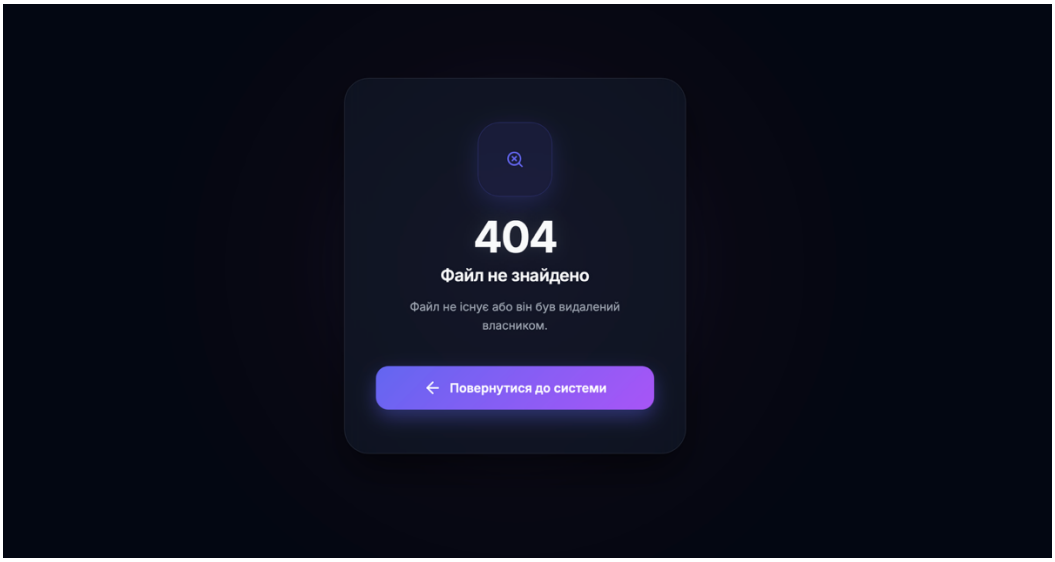

Рисунок 3.4 – Відображення стилізованої сторінки обробки помилок (error.html)

Тестування підтвердило, що реалізовані алгоритми забезпечують стабільну та безпечну роботу системи. Модуль створення ефемерних посилань коректно обмежує доступ до ресурсів за часом, а фоновий процес очищення своєчасно звільняє дисковий простір Orange Pi One від прострочених даних, що гарантує автономність сервера без необхідності ручного адміністрування.

3.4.1 Моніторинг роботи сервера та аналіз термінальних логів

Для забезпечення стабільної роботи вебдодатка та оперативного реагування на позаштатні ситуації було налаштовано систему безперервного моніторингу подій. Оскільки серверна логіка працює як фонова служба на базі Orange Pi One, усі стандартні потоки виведення та помилок мікрофреймворку Flask автоматично перехоплюються та записуються до системного журналу операційної системи Linux [14].

Щоб здійснювати моніторинг системних журналів у режимі реального часу, адміністратор використовує захищене SSH-з'єднання [21], яке дозволяє виводити на екран термінала безперервний «живий» потік логів роботи серверної служби.

Як показано на рисунку 3.5, у терміналі в реальному часі відображається перебіг усіх критичних процесів системи. Зокрема, безперервно фіксується мережева маршрутизація, тому кожен вхідний запит від клієнта супроводжується даними про цільовий URL, IP-адресу ініціатора та кінцевий статус-код відповіді [1]. Паралельно з цим система логує файлові операції, відзначаючи успішні транзакції збереження нових документів та етапи генерації ZIP-архівів під час масового завантаження. Окремо в консоль виводяться сповіщення модуля безпеки, які одразу сигналізують про спрацювання захисних механізмів, як-от виявлення VPN-трафіку [5] чи спроби авторизації заблокованого користувача. Також у терміналі можна спостерігати за фоновим аудитом ефемерних посилань: система фіксує роботу функції, відображаючи кількість перевірених токенів доступу та підтверджуючи фізичне знищення прострочених файлів із накопичувача.

Такий підхід до моніторингу дозволяє не лише ефективно відлагоджувати програмний код під час розробки, відслідковуючи трасування стека помилок (Stack Trace) у разі виникнення виключень (Exception) [12], але й виступає потужним інструментом для розслідування інцидентів інформаційної безпеки [6] під час робочої експлуатації (Production) сервера.

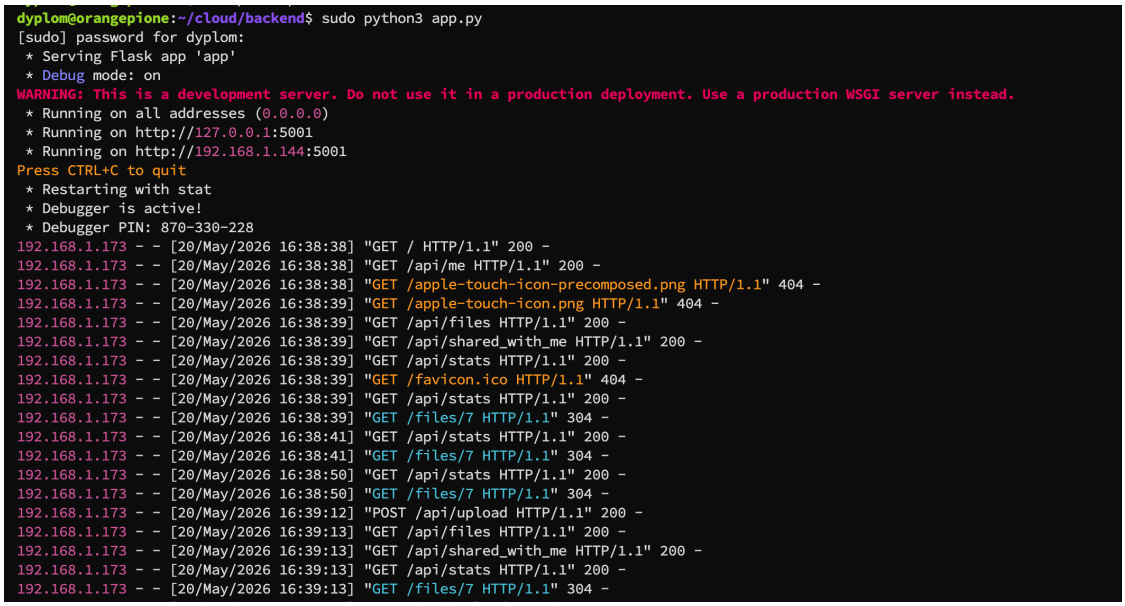

Рисунок 3.5 – Динамічний моніторинг HTTP-запитів та системних подій у терміналі сервера

Завдяки наявності повної хронології подій, адміністратор має змогу проводити ретроспективний аудит будь-яких нетипових ситуацій, що дозволяє своєчасно усувати вразливості [17] та гарантувати стабільну роботу вузла.

3.5 Забезпечення інформаційної безпеки, аудит доступу та протидія аномальній активності

Ураховуючи специфіку вебсистеми як платформи для зберігання та публічного обміну цифровими ресурсами, критично важливим етапом розробки стала реалізація багаторівневої системи інформаційної безпеки [13]. Захист від несанкціонованого доступу, моніторинг підозрілих дій та запобігання махінаціям реалізовані безпосередньо на рівні серверної логіки Flask [11].

3.5.1 Виявлення засобів анонімізації (VPN / Proxy)

Одним із ключових векторів захисту сховища є контроль походження мережевого трафіку. Для запобігання використанню системи зловмисниками

(наприклад, для розповсюдження шкідливого програмного забезпечення з прихованою ідентичністю), впроваджено модуль аналізу вхідних з'єднань. Система перевіряє HTTP-заголовки клієнта, щоб виявити реальну IP-адресу, навіть якщо запит проходить через проміжні вузли [1].

Для виявлення VPN-сервісів та проксі-вузлів алгоритм зіставляє IP-адресу клієнта з базами даних відомих хостинг-провайдерів та дата-центрів (ASN). Якщо трафік ідентифікується як анонімний або такий, що належить до публічних VPN-мереж, система маркує таку сесію як потенційно небезпечну, фіксуючи подію в системному журналі аудиту сервера [5].

3.5.2 Моніторинг та виявлення махінацій

Для захисту системи від автоматизованих кібератак (наприклад, перебору паролів типу Brute-Force, DDoS-атак на рівні додатку або масового парсингу чужих публічних посилань) реалізовано алгоритм виявлення аномальної поведінки [6]. Серверна логіка моніторить інтенсивність HTTP-запитів від кожного авторизованого користувача та гостя.

Система захисту класифікує як махінації три основні загрози. До них належать серійні невдалі спроби авторизації, навмисний спам бази даних через масову генерацію ефемерних посилань та безпосереднє втручання у структуру вебзапитів, коли зловмисник намагається змінити системні ідентифікатори для проникнення в директорію іншого користувача [17].

3.5.3 Механізм автоматичного блокування

Ефективною відповіддю на виявлені загрози або махінації виступає механізм миттєвого ізолювання зловмисника, в основі якого лежить використання параметра is_blocked у таблиці users реляційної бази даних SQLite [18]. У разі фіксації критичного порушення правил безпеки вебсервер автоматично виконує відповідну SQL-транзакцію для зміни статусу облікового запису. Відразу після оновлення цього прапорця в базі даних архітектура додатка ініціює комплекс невідкладних захисних дій.

Насамперед активний сеанс користувача негайно переривається за допомогою системних функцій скидання авторизації [12]. Паралельно з цим спрацьовує програмний перехоплювач запитів, реалізований на рівні серверного ядра. Цей механізм перевіряє статус акаунта при кожному новому зверненні, і якщо індикатор блокування активовано, будь-які подальші запити повністю відхиляються, що забезпечує надійне обмеження взаємодії з інтерфейсом. Додатково система заморожує всі цифрові активи порушника, автоматично деактивуючи раніше згенеровані ним публічні посилання та перекриваючи доступ до файлів у його особистому кабінеті з обов'язковим виведенням системного повідомлення про заборону доступу [13].

Розблокування облікового запису та відновлення доступу до файлової системи можливе виключно після ручного аудиту інциденту головним адміністратором сервера. Такий комплексний підхід гарантує, що вебдодаток залишається стійким до компрометації, а конфіденційні дані надійно захищені від витоку та несанкціонованих маніпуляцій.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Фізіологічний вплив факторів існування на життєдіяльність людини

Людина, працюючи із сучасними електронними та обчислювальними системами під час розробки програмного забезпечення, піддається численним фізичним та психофізіологічним факторам. Для інженера-програміста, який проєктує систему хмарного зберігання даних, найважливішим є вплив електромагнітного випромінювання, зорове навантаження та гіподинамія [2]. Коли розробник тривалий час взаємодіє з екраном монітора (наприклад, під час написання коду на MacBook Pro або моніторингу термінальних логів сервера), його зоровий апарат зазнає значної перевтоми. Проблема ускладнюється тим, що під час концентрації на програмному коді частота кліпання очима знижується у 2–3 рази, що призводить до пересихання рогівки та розвитку «синдрому сухого ока».

Крім зорового навантаження, робочу обстановку формують рівень освітленості, мікроклімат та електромагнітні поля. У приміщеннях, де встановлено серверну техніку (зокрема мікрокомп'ютер Orange Pi One [16] та мережеве обладнання для оптоволоконного зв'язку [1]), нерідко порушується оптимальний температурний режим через тепловіддачу процесорів. Недостатня вентиляція призводить до зниження концентрації кисню, що погіршує когнітивні функції людини: вона втрачає здатність довго зосереджуватися на складних алгоритмах архітектури бази даних, а ризик допущення критичних помилок у програмному коді зростає. Дослідження показують, що за таких умов продуктивність програміста може знижуватися на 15–20 % через неодноразові переривання роботи для відпочинку [8].

Статична робоча поза, яка є характерною для процесу програмування, також суттєво впливає на здоров’я. Тривале сидіння перед комп'ютером сприяє порушенню кровообігу в органах малого тазу, викривленню хребта та розвитку тунельного синдрому зап'ястя (через постійну роботу з клавіатурою і мишею). Це особливо критично при тонких налаштуваннях серверної конфігурації. М'язова напруга швидко призводить до фізичного стомлення.

Електромагнітні поля (ЕМП) у низькочастотному діапазоні в приміщенні, де встановлено кілька обчислювальних систем та Wi-Fi маршрутизаторів, можуть впливати на нервову систему. Хоча сучасна техніка (зокрема ARM-процесори) працює на низьких струмах і не створює критичного ЕМП, наведені поля від розміщеного поруч силового обладнання підсилюють ризик. За нормативами ДСН 3.3.6.096-2002 встановлено граничні допустимі рівні неіонізуючого випромінювання, перевищення яких вимагає перегрупування техніки або ізоляції робочої зони.

Нормативно-правовою основою для оцінки фізіологічного впливу факторів під час роботи з ЕОМ є низка документів:

— ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [2];

— ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень» [8];

— ДБН В.2.5-28:2018 «Природне і штучне освітлення» [3];

— НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин».

Враховуючи ці чинники, у кваліфікаційній роботі обґрунтовано принципи та методи захисту розробника [4].

Організаційні заходи включають раціональне чергування праці та відпочинку з регламентованими перервами по 15 хвилин кожні 2 години інтенсивної роботи з кодом і виконання комплексу вправ для очей та опорно-рухового апарату.

Програмні методи захисту (ергономіка інтерфейсу) полягають у тому, що з метою зниження зорового навантаження клієнтський вебінтерфейс розробленої системи спроєктовано виключно у темних відтінках (Dark Mode) [10], а використання CSS-змінних із темним фоном (--bg-base: #030712) та контрастним м'яким текстом знижує випромінювання синього спектра світла екраном, що доведено зменшує втому очей при тривалій роботі в умовах штучного освітлення.

4.2 Заходи з техніки безпеки при експлуатації обладнання

Експлуатація, налагодження та тестування апаратно-програмного комплексу на базі мікрокомп'ютера Orange Pi One [16] супроводжується безпосередньою роботою з електронними компонентами, блоками живлення та мережевим обладнанням. Комп'ютерна та серверна техніка належить до електроустановок напругою до 1000 В. Щоб гарантувати безпеку оператора та уникнути пошкодження обладнання під час розгортання системи, було дотримано комплексу заходів з техніки безпеки за чотирма основними напрямками.

Головну небезпеку під час експлуатації становить можливість ураження електричним струмом через пошкодження ізоляції кабелів живлення або несправність блоків живлення. Згідно з Правилами технічної експлуатації електроустановок споживачів (ПТЕЕС), було вжито таких заходів:

— підключення сервера до електромережі здійснювалося виключно через мережевий фільтр із вбудованим захистом від перепадів напруги та струмів короткого замикання;

— обладнання було підключено до контуру захисного заземлення (занулення) будівлі для безпечного відведення небезпечного струму пробою;

— категорично заборонялося використання пошкоджених розеток або кабелів із порушеною ізоляцією.

Електронні компоненти мікрокомп'ютера (зокрема відкриті контакти

GPIO), OLED-дисплей та підключені USB-накопичувачі є вкрай чутливими до електростатичних імпульсів [16]. Для попередження виходу з ладу портів передачі даних або пошкодження файлової системи, підключення периферії та будь-які маніпуляції з платою здійснювалися тільки після зняття статичної напруги з рук оператора (шляхом дотику до заземленого металевого предмета). Це запобігає пробою чутливих мікросхем.

З погляду пожежної безпеки, приміщення з обчислювальною технікою належить до категорії «В» (пожежонебезпечне). У разі виникнення загоряння електрообладнання під напругою категорично забороняється використовувати воду або пінні вогнегасники. Згідно з чинними нормами, робоче місце повинно бути оснащене вуглекислотним вогнегасником (типу ВВК-2 або ВВК-3). Вони ефективно збивають полум'я шляхом охолодження зони горіння і не залишають слідів, що дозволяє уникнути пошкодження сусідніх електронних компонентів.

Безперервна робота сервера (хостинг хмарного сховища в режимі 24/7) вимагає суворого дотримання мікроклімату для уникнення перегріву процесора Allwinner H3 та ризику теплового пробою [16]. Температура повітря у приміщенні повинна підтримуватися в межах 19–24 °C із відносною вологістю на рівні 40–60 % [8]. Це запобігає накопиченню статичної електрики (у разі надмірної сухості) або утворенню конденсату на платі (у разі надмірної вологості). Робоче місце було додатково обладнане системою кондиціонування повітря.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи над розподіленою комп'ютерною системою зберігання цифрових ресурсів було проведено ґрунтовне дослідження існуючих рішень і технічних вимог до сучасних хмарних сховищ з урахуванням специфіки інформаційної безпеки та розмежування прав доступу. Вивчення глобальних та корпоративних розробок дозволило окреслити головні критерії: швидкодію обробки файлових потоків, надійність зберігання метаданих та можливість безпечного публічного обміну інформацією. На цій основі було сформульовано набір вимог до серверної частини, швидкодії реляційної бази даних і ергономіки інтерфейсу користувача, що лягли в основу подальшої розробки.

Першим практичним кроком стало проєктування логічної архітектури клієнт-серверного додатка та структури бази даних. Було розроблено таблиці SQLite (users та files), які забезпечили жорстке розмежування доступу та зв'язок між обліковими записами і фізичними файлами. Було обґрунтовано вибір апаратної платформи — мікрокомп'ютера Orange Pi One з підключенням до оптоволоконної мережі (FTTH/PON), що дозволило забезпечити високу швидкість операцій введення-виведення без критичних енергозатрат.

Паралельно з підготовкою серверної інфраструктури велася розробка програмного забезпечення. У локальному середовищі на базі Apple M1 створено серверну логіку мовою Python із використанням мікрофреймворку Flask. Реалізовано алгоритми безпечної автентифікації (з криптографічним хешуванням паролів), обробки файлових потоків (з генерацією унікальних UUID) та створення ефемерних (тимчасових) посилань (TTL). Особливу увагу було приділено розробці фонового модуля автоматичного знищення прострочених даних cleanup_expired_files(), що гарантує автономність очищення дискового простору сервера.

Клієнтська частина (Frontend) була спроєктована з використанням сучасних вебстандартів (HTML5, CSS3, JavaScript) у темній стилістиці (Dark

Mode). Це рішення, спільно з імплементацією системи глобальної обробки помилок (404, 500, 413) та моніторингу статусу інтернет-з'єднання, забезпечило інтуїтивно зрозумілий та безперебійний досвід взаємодії користувача із системою.

Тестування розробленого програмно-апаратного комплексу проводилося в умовах реальної експлуатації: під навантаженням великими файлами (до 5 ГБ), в умовах втрати мережевого з'єднання клієнтом та під час симуляції аномальної поведінки (перевірка системи блокування is_blocked). Отримані результати підтвердили відповідність розробленої вебсистеми поставленим технічним вимогам з погляду швидкодії, відмовостійкості та інформаційної безпеки.

У процесі реалізації проєкту здобуто цінні навички проєктування реляційних баз даних, розробки бекенд-логіки на фреймворку Flask, адміністрування Linux-серверів на базі мікрокомп'ютерів (Orange Pi) та методологій тестування вебінтерфейсів. Розроблена система є повноцінним продуктом, перспективним для розгортання в освітніх закладах (наприклад, для студентських груп), корпоративних мережах малого бізнесу або для особистого використання, де повний контроль над даними є критично важливим.

Отже, виконана кваліфікаційна робота підтверджує ефективність обраного стеку технологій для побудови систем хмарного зберігання, демонструє практичну придатність розробленого вебдодатка та створює міцну архітектурну основу для його подальшого масштабування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Буров Є.В., Митник М.М. Комп’ютерні мережі. Підручник. Том другий. Львів: «Магнолія 2006», 2024. 204 с.
2. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПіН 3.3.2.007-98. Київ: МОЗ України, 1998. 18 с.
3. ДБН В.2.5-28:2018. Природне і штучне освітлення. Київ: Мінрегіон України, 2018. 128 с.
4. Жаровський Р.О., Луцик Н.С., Осухівська Г.М., Паламар А.М., Тиш Є.В. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальністю 123 «Комп’ютерна інженерія» усіх форм навчання. Тернопіль: ТНТУ, 2024. 39 с.
5. Карабан Д., Жаровський Р. Аналіз проблем забезпечення анонімності користувачів при використанні мережі Інтернет. Матеріали XII Міжнародної науково-технічної конференції молодих учених та студентів «Актуальні задачі сучасних технологій» (6-7 грудня 2023 року). Тернопіль: ТНТУ. 2023. С. 456.
6. Ковтун Н., Жаровський Р. Алгоритмічне забезпечення систем виявлення вторгнень. Матеріали XI науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі системи та технології» (13-14 грудня 2023 року). Тернопіль: ТНТУ. 2023. С. 156.
7. Лупенко С.А., Пасічник В.В., Тиш Є.В. Комп’ютерна логіка. Навчальний посібник. Львів: Видавництво «Магнолія 2006», 2024. 354 с.
8. Луцків А., Лупенко С., Пасічник В. Паралельні та розподілені обчислення. Навчальний посібник. Львів: Видавництво «Магнолія 2006», 2024. 566 с.

9. Рій І.І., Тиш Є.В. Методи побудови та порівняльний аналіз хаотичних алгоритмів шифрування для сучасних комп'ютеризованих систем. XIII науково-технічна конференція «Інформаційні моделі, системи та технології», Тернопіль: ТНТУ, 2025. 144 с.

10. Санітарні норми мікроклімату виробничих приміщень: ДСН 3.3.6.042-99. Київ: МОЗ України, 1999. 11 с.

11. Свергун С., Жаровський Р. Тестування програмного забезпечення побудованого на мікросервісній архітектурі. Матеріали X науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі системи та технології» (7-8 грудня 2022 року). Тернопіль: ТНТУ. 2022. С. 92.

12. Харченко О., Яцишин В. Розробка та керування вимогами до програмного забезпечення на основі моделі якості. Вісник ТДТУ. Тернопіль, 2009. Т. 14. №1. С. 201-207.

13. Flask Documentation (3.0.x). URL: https://flask.palletsprojects.com/ (дата звернення: 10.02.2026)

14. Grinberg M. Flask Web Development: Developing Web Applications with Python. 2nd ed. O'Reilly Media, 2018. 316 p. (дата звернення: 12.02.2026)

15. Hoffman A. Web Security for Developers: Real Threats, Practical Defense. No Starch Press, 2020. 328 p. (дата звернення: 15.02.2026)

16. Nemeth E., Snyder G., Hein T. R., Whaley B. UNIX and Linux System Administration Handbook. 5th ed. Addison-Wesley Professional, 2017. 1232 p. (дата звернення: 18.02.2026)

17. OpenVPN Documentation: Secure and Extend Your Network. URL: https://openvpn.net/community-resources/ (дата звернення: 22.02.2026)

18. Orange Pi One User Manual. Shenzhen Xunlong Software Co., Ltd. URL: http://www.orangepi.org/ (дата звернення: 25.02.2026)

19. OWASP Top 10:2021 – The Ten Most Critical Web Application Security Risks. URL: https://owasp.org/Top10/ (дата звернення: 28.02.2026)

20. Owens M., Allen G. SQLite. Apress, 2010. 344 p. (дата звернення:

05.03.2026)

21. SQLite Official Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 10.03.2026)

22. Stallings W. Cryptography and Network Security: Principles and Practice. 8th ed. Pearson, 2020. 768 p. (дата звернення: 18.03.2026)

23. Termius Documentation: The SSH client that works on Desktop and Mobile. URL: https://docs.termius.com/ (дата звернення: 25.03.2026)

24. Yatsyshyn V., Pastukh O., Palamar A., Zharovskyy R. Technology of relational database management systems performance evaluation during computer systems design. Scientific Journal of TNTU, Ternopil, Ukraine, 2023. Vol. 109, No 1. P. 54–65. (дата звернення: 01.04.2026)

Додаток А

Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії

Кафедра комп'ютерних систем та мереж

"Затверджую"

Завідувач кафедри КС

_____ Осухівська Г.М.

" 2 " лютого 2026 р.

Розподілена комп'ютерна система зберігання цифрових ресурсів

з розмежуванням прав доступу

ТЕХНІЧНЕ ЗАВДАННЯ

на _11_ листках

Вид робіт: Кваліфікаційна робота

На здобуття освітнього ступеня «Бакалавр»

Спеціальність 123 «Комп'ютерна інженерія»

«УЗГОДЖЕНО»	«ВИКОНАВЕЦЬ»
Керівник кваліфікаційної роботи	Студент групи СІ-42
_____ к.т.н., доц. Тиш Є. В.	_____ Ковальський А. Ю.
« 2 » лютого 2026 р.	« 2 » лютого 2026 р.

Тернопіль 2026

1 Загальні відомості

1.1 Повна назва та її умовне позначення

Повна назва теми кваліфікаційної роботи: «Розподілена комп’ютерна система зберігання цифрових ресурсів з розмежуванням прав доступу».

Умовне позначення кваліфікаційної роботи: КС КРБ 123.176.00.00

1.2 Виконавець

Студент групи СІ-42, факультету комп’ютерно-інформаційних систем і програмної інженерії, кафедри комп’ютерних систем та мереж, Тернопільського національного технічного університету імені Івана Пулюя, Ковальський А. Ю.

1.3 Підстава для виконання роботи

Підставою для виконання кваліфікаційної роботи є наказ по університету (№4/9-188 від 24.04.2026 р.)

1.4 Планові терміни початку та завершення роботи

Плановий термін початку виконання кваліфікаційної роботи – 26.01.2026 р.

Плановий термін завершення виконання кваліфікаційної роботи – 21.06.2026 р.

1.5 Порядок оформлення та пред’явлення результатів роботи

Порядок оформлення пояснювальної записки та графічного матеріалу здійснюється у відповідності до чинних норм та правил ISO, ЕСКД, ЕСПД та ДСТУ.

Пред’явлення проміжних результатів роботи з виконання кваліфікаційної роботи здійснюється у відповідності до графіку, затвердженого керівником роботи. Попередній захист кваліфікаційної роботи відбувається при готовності роботи – наявності пояснювальної записки та графічного матеріалу.

Пред’явлення результатів кваліфікаційної роботи відбувається шляхом захисту на відповідному засіданні ЕК, ілюстрацією основних досягнень за допомогою графічного матеріалу.

2 Призначення і цілі створення системи

2.1 Призначення системи

Вебсистема хмарного зберігання, що розробляється, призначена для:

– Надійного зберігання цифрових ресурсів користувачів.

– Безпечного публічного обміну файлами за допомогою ефемерних посилань (з автоматичним знищенням).

– Адміністрування прав доступу та моніторингу вільного дискового простору сервера.

2.2 Мета створення системи

Метою кваліфікаційної роботи є розробка та налаштування безпечного клієнт-серверного вебдодатка на базі мікрофреймворку Flask (Python) та бази даних SQLite з розгортанням на одноплатному мікрокомп'ютері Orange Pi One.

2.3 Характеристика об'єкту

Розроблена комп'ютерна система призначена для використання як автономне хмарне сховище в освітніх установах, малому бізнесі або для приватного (домашнього) використання з доступом через мережу Інтернет (оптоволоконне з'єднання).

3 Вимоги до системи

3.1 Вимоги до системи в цілому

3.1.1 Вимоги до структури та функціонування системи

Система повинна складатись з:

Апаратної частини (мікрокомп'ютер Orange Pi One, зовнішній USB-накопичувач, мінідисплей для моніторингу).

Серверної логіки (Бекенд на Flask, система маршрутизації, модуль обробки файлів).

Бази даних (SQLite для збереження облікових записів та токенів доступу).

Інтерфейсу користувача (Фронтенд з використанням HTML/CSS/JS).

3.1.2 Вимоги до способів та засобів зв’язку між компонентами системи

Основна вимога — обмін даними між клієнтом та сервером повинен відбуватися за протоколом HTTP. Взаємодія сервера з базою даних повинна здійснюватися через локальні SQL-транзакції без залучення зовнішніх мережевих вузлів.

3.1.3 Вимоги до режимів функціонування системи

Для системи визначено два режими функціонування: нормальний та аварійний.

Основним режимом є нормальний (безперебійна обробка запитів 24/7 у вигляді фонової служби Systemd).

Аварійний режим функціонування характеризується відсутністю з'єднання (спрацювання офлайн-екрана на фронтенді) або помилками виконання запитів, при яких система повинна перенаправляти користувача на сторінки обробки помилок (404, 500, 413) без зупинки основного процесу сервера.

3.1.4 Вимоги по діагностуванню системи

Для діагностування системи використовується вбудоване логування подій (stdout/stderr) з виведенням інформації про HTTP-запити та помилки у системний журнал операційної системи (Journalctl).

3.1.5 Перспективи розвитку, проектування системи

Система може бути масштабована за рахунок підключення накопичувачів більшого об'єму, впровадження SSL-сертифікатів (HTTPS) та розширення функціоналу (наприклад, додавання редагування текстових файлів безпосередньо у браузері).

3.2 Показники призначення

Система повинна передбачати можливість одночасної роботи кількох авторизованих користувачів без блокування потоків бази даних.

3.2.1 Вимоги до надійності

Система повинна забезпечувати працездатність та відновлення функцій при:

– Збоях в електропостачанні (автоматичне завантаження вебсервера після включення Orange Pi One завдяки демону Systemd).

– Автоматичному створенні директорій для завантаження (os.makedirs) у разі їх випадкового видалення.

– Захисті від переповнення диска (відслідковування об'єму накопичувача та автоматичне видалення файлів за TTL).

3.3 Вимоги до безпеки

Зовнішні елементи технічних засобів серверної частини системи (одноплатний мікрокомп'ютер Orange Pi One, зовнішній USB-накопичувач, блоки живлення та мережеве комутаційне обладнання), що перебувають під електричною напругою, повинні мати надійний конструктивний захист від випадкового дотику оператора. Самі технічні засоби повинні мати занулення або захисне заземлення відповідно до чинних Правил улаштування електроустановок (ПУЕ).

Система електроживлення апаратного вузла повинна забезпечувати автоматичне захисне вимикання при виникненні перевантажень, стрибків напруги та коротких замикань в колах навантаження. Для цього мають застосовуватися мережеві фільтри з елементами захисту від комутаційних завад. Обов'язково повинна бути передбачена можливість оперативного аварійного вимкнення всього комплексу шляхом ручного знеструмлення.

Загальні вимоги пожежної безпеки повинні відповідати чинним нормам на побутове та офісне електрообладнання. У разі виникнення пожежі ізоляційні матеріали провідників та корпуси пристроїв не повинні виділяти високотоксичних отруйних газів і густого диму. Після зняття електроживлення з пошкодженої ділянки мережі має бути доступне застосування сертифікованих засобів пожежогасіння (зокрема вуглекислотних вогнегасників типу ВВК, що запобігають пошкодженню електронних компонентів).

3.3.1 Вимоги до експлуатації, технічного обслуговування, ремонту і зберігання компонентів системи

Мікроклімат в приміщеннях, де розгорнуто фізичний сервер хмарного сховища, повинен відповідати нормам виробничого мікроклімату згідно з ДСН 3.3.6.042-99 і забезпечувати такі параметри:

– температуру повітря в межах від +15 °С до +30 °С (для запобігання перегріву процесора під час виконання інтенсивних операцій архівації даних);

– відносну вологість повітря при 25 °С в межах від 30 % до 70 % (без утворення конденсату на платі);

– атмосферний тиск 760±25 мм рт. ст.

Періодичне технічне обслуговування використовуваних технічних засобів має проводитися відповідно до вимог супровідної технічної документації, але не рідше ніж один раз на рік.

Періодичне технічне обслуговування і тестування повинні включати: візуальний контроль цілісності кабелів живлення та оптоволоконних ліній, очищення плати від пилу, програмну діагностику цілісності Ext4 файлової системи зовнішнього накопичувача та перевірку стабільності інтернет-каналу.

На підставі результатів тестування апаратних засобів повинен проводитися детальний аналіз причин виникнення виявлених дефектів і негайно прийматися заходи щодо їх повної ліквідації.

3.4 Вимоги до захисту інформації від несанкціонованого доступу

Система повинна забезпечувати надійний захист конфіденційних даних користувачів від несанкціонованого доступу (НСД) на рівні не нижче встановленого вимогами, що пред'являються до категорії 1Д за класифікацією діючого нормативного документа НД ТЗІ 2.5-004-99 «Автоматизовані системи. Захист від несанкціонованого доступу до інформації. Класифікація автоматизованих систем».

Компоненти підсистеми захисту від НСД (реалізовані на рівні серверної логіки) повинні безперебійно забезпечувати:

– ідентифікацію та автентифікацію користувачів за допомогою захищеного введення облікових даних;

– криптографічний захист паролів за допомогою алгоритмів надійного гешування (із використанням солі), що виключає зберігання паролів у відкритому вигляді;

– перевірку повноважень користувача при кожному HTTP-запиті до системи;

– жорстке розмежування доступу користувачів (користувач має доступ виключно до власних файлів та метаданих);

– захист від автоматизованих махінацій (модуль виявлення VPN/Proxy з можливістю миттєвого блокування акаунта is_blocked).

Рівень захищеності від несанкціонованого доступу засобів обчислювальної техніки, що здійснюють обробку конфіденційної інформації, повинен відповідати вимогам класу захищеності згідно з нормативним документом “Засоби обчислювальної техніки. Захист від несанкціонованого доступу до інформації. Показники захищеності від несанкціонованого доступу до інформації”.

3.4.1 Вимоги по збереженню інформації при аваріях

Цілісність та збереження інформації (як метаданих у базі даних, так і бінарних файлів на диску) при виникненні аварійних ситуацій (раптове відключення електроживлення або апаратні збої) повинні бути забезпечені. Це досягається підтримкою повної транзакційності (ACID) СУБД SQLite, використанням журнальованої файлової системи на накопичувачі та періодичним збереженням інформації на резервних носіях.

3.4.2 Вимоги по стандартизації і уніфікації

Система повинна розроблятися із використанням загальноприйнятих стандартизованих інструментів (Python, Flask, SQL) та відповідати вимогам ергономіки і зручності користування. Клієнтська частина має використовувати технології HTML5/CSS3/JavaScript, володіти адаптивною верстkою та бути реалізованою у темній стилістиці (Dark Mode) для зниження зорового втомлення оператора за умови комплектування високоякісним обладнанням (ЕОМ, монітор).

3.4.3 Вимоги до функцій (завдань), що виконуються системою:

– забезпечення зручного, інтуїтивного графічного веб-інтерфейсу з динамічним оновленням даних;

– надійне довготривале зберігання медіафайлів та документів із їх структуруванням за власниками;

– генерація унікальних ефемерних (тимчасових) посилань для безпечного зовнішнього поширення файлів із функцією автоматичного безповоротного видалення об'єктів з диска після закінчення встановленого ліміту часу (TTL);

– швидкий пошук необхідної інформації про стан дискового простору та ведення системного аудиту;

– забезпечення високої швидкодії сервера та низьких накладних витрат на обробку запитів.

4 Вимоги до документації

Документація повинна відповідати вимогам ЄСКД та ДСТУ

Комплект документації повинен складатись з:

– пояснювальної записки;

– графічного матеріалу:

а) Структурна схема.

б) Структурна схема програмного забезпечення

в) Схема електрична принципова.

г) Блок-схема алгоритму

*Примітка: У комплект документації можуть вноситися зміни та доповнення в процесі розробки.

5 Стадії та етапи проектування

Таблиця 1 – Стадії та етапи виконання кваліфікаційної роботи бакалавра

№ етапу	Назва етапу виконання кваліфікаційної роботи бакалавра	Термін виконання
1	*Розробка технічного завдання*	*26.01 – 01.02*
2	*Робота над першим розділом: «Аналіз технічного завдання, огляд існуючих рішень для хмарного зберігання та формування вимог до системи.»*	*02.02 – 12.02*
3	*Робота над другим розділом: Проєктна частина, обґрунтування вибору апаратної бази (Orange Pi) та проєктування архітектури реляційної бази даних.*	*18.04 – 26.04*
4	*Робота над третім розділом: Практична частина, розробка серверної логіки (Flask), верстка адаптивного інтерфейсу та тестування модуля ефемерних посилань.*	*27.04 – 08.05*
5	*Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»*	*09.05 – 20.05*
6	*Оформлення пояснювальної записки і графічного матеріалу*	*21.05 – 7.06*
7	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*8.06 – 14.06*
8	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*
9	*Захист кваліфікаційної роботи бакалавра*	*23.06.2026*

6 Додаткові умови виконання кваліфікаційної роботи

Під час виконання кваліфікаційної роботи у дане технічне завдання можуть вноситися зміни та доповнення.

Додаток Б

Перелік елементів

Поз. позначення	*Найменування*	*Кіл.*	*Примітка*
	Обчислювальний модуль		
A2	Мікрокомп'ютер Orange Pi One (SoC Allwinner H3, 512MB RAM)	1	
	Модулі розширення та інтерфейси		
A3	USB 2.0 Концентратор (Hub)	1	
	Накопичувачі даних		
A5	Твердотільний накопичувач SSD (від 120 ГБ) з адаптером USB-SATA	1	
—	Карта пам'яті MicroSD (Class 10, від 16 ГБ)	1	
	Мережеві пристрої		
A4	USB Wi-Fi адаптер (на базі чипа RTL8188 або MT7601)	1	
	Пристрої відображення		
A1	Модуль міні-дисплея OLED 0.96" (128x64, I2C, контролер SSD1306)	1	
	Джерела живлення		
LM1	Стабілізований блок живлення 5В / 3А (штекер DC 4.0x1.7 мм)	1	
	Допоміжні елементи		
—	Комплект пасивних радіаторів (охолодження SoC Allwinner H3)	1	

					КС КРБ 123.176.00.00 ПЕ	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		*Ковальський А.Ю.*			*Розподілена комп'ютерна система зберігання цифрових ресурсів з розмежуванням прав доступу Перелік елементів*	*Літ* / *Арк.* / *Аркушів*: *68* / *1*
Перевірив		*Тиш Є.В.*				
Рецензент		*Гром'як Р.С.*				
Н. Контр.		*Луцик Н.С.*				*ТНТУ, каф. КС, гр. СІ-42*
Зав. каф.		*Осухівська Г.М.*				

Додаток В

Лістинг коду серверної частини

```
from flask import Flask, request, render_template, jsonify,
     send_from_directory, session, send_file
from werkzeug.utils import secure_filename
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime, timedelta
import os
import shutil
import sqlite3
import uuid
import zipfile
import io

app = Flask(__name__)
app.secret_key = 'super_secret_kvitova_key'

UPLOAD_FOLDER = '/mnt/usb_drive'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

def init_db():
    conn = sqlite3.connect('users.db')
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY
     AUTOINCREMENT, username TEXT UNIQUE NOT NULL, password TEXT NOT
     NULL)''')
    try: c.execute('ALTER TABLE users ADD COLUMN is_blocked INTEGER DEFAULT
     0')
    except sqlite3.OperationalError: pass
    try: c.execute('ALTER TABLE files ADD COLUMN expires_at TIMESTAMP')
    except sqlite3.OperationalError: pass
    try: c.execute('ALTER TABLE files ADD COLUMN auto_delete INTEGER
     DEFAULT 0')
    except sqlite3.OperationalError: pass
```

```
c.execute('''CREATE TABLE IF NOT EXISTS files (id INTEGER PRIMARY KEY
     AUTOINCREMENT, user_id INTEGER NOT NULL, original_name TEXT NOT NULL,
     physical_name TEXT NOT NULL, size INTEGER NOT NULL, upload_date
     TIMESTAMP DEFAULT CURRENT_TIMESTAMP, is_public INTEGER DEFAULT 0,
     share_token TEXT, expires_at TIMESTAMP, auto_delete INTEGER DEFAULT 0,
     FOREIGN KEY(user_id) REFERENCES users(id))''')
    c.execute('''CREATE TABLE IF NOT EXISTS shared_access (id INTEGER
     PRIMARY KEY AUTOINCREMENT, file_id INTEGER NOT NULL,
     shared_with_user_id INTEGER NOT NULL, UNIQUE(file_id,
     shared_with_user_id), FOREIGN KEY(file_id) REFERENCES files(id),
     FOREIGN KEY(shared_with_user_id) REFERENCES users(id))''')
    conn.commit()
    conn.close()

init_db()

def get_user_folder():
    if 'username' not in session: return None
    user_folder = os.path.join(app.config['UPLOAD_FOLDER'],
     secure_filename(session['username']))
    os.makedirs(user_folder, exist_ok=True)
    return user_folder

def format_bytes(bytes_size):
    if bytes_size == 0: return "0 Байт"
    k = 1024
    sizes = ["Байт", "КБ", "МБ", "ГБ"]
    import math
    i = int(math.floor(math.log(bytes_size, k)))
    return f"{round(bytes_size / (k ** i), 2)} {sizes[i]}"

def cleanup_expired_files():
    conn = sqlite3.connect('users.db')
    now_str = datetime.now().isoformat()
    expired_files = conn.execute('SELECT f.id, f.physical_name, u.username
     FROM files f JOIN users u ON f.user_id = u.id WHERE f.auto_delete = 1
     AND f.expires_at IS NOT NULL AND f.expires_at < ?',
     (now_str,)).fetchall()
```

```
for f in expired_files:
        file_path = os.path.join(app.config['UPLOAD_FOLDER'],
     secure_filename(f[2]), f[1])
        if os.path.exists(file_path): os.remove(file_path)
        conn.execute('DELETE FROM shared_access WHERE file_id = ?',
     (f[0],))
        conn.execute('DELETE FROM files WHERE id = ?', (f[0],))
    conn.commit()
    conn.close()

@app.route('/api/register', methods=['POST'])
def register():
    data = request.json
    hashed_pw = generate_password_hash(data.get('password'))
    try:
        conn = sqlite3.connect('users.db')
        conn.execute('INSERT INTO users (username, password) VALUES (?,
     ?)', (data.get('username'), hashed_pw))
        conn.commit()
        conn.close()
        os.makedirs(os.path.join(app.config['UPLOAD_FOLDER'],
     secure_filename(data.get('username'))), exist_ok=True)
        return jsonify({'message': 'Реєстрація успішна'}), 200
    except sqlite3.IntegrityError: return jsonify({'error': 'Користувач вже
     існує'}), 400

@app.route('/api/login', methods=['POST'])
def login():
    data = request.json
    conn = sqlite3.connect('users.db')
    user = conn.execute('SELECT id, password, is_blocked FROM users WHERE
     username = ?', (data.get('username'),)).fetchone()
    conn.close()
    if user:
        if user[2] == 1: return jsonify({'error': 'Ваш акаунт заблоковано
     Адміністратором'}), 403
```

```
        if check_password_hash(user[1], data.get('password')):
            session['user_id'] = user[0]
```

```
            session['username'] = data.get('username')
            return jsonify({'message': 'Вхід успішний'}), 200
    return jsonify({'error': 'Невірний логін або пароль'}), 401

@app.route('/api/logout', methods=['POST'])
def logout():
    session.clear()
    return jsonify({'message': 'Вийшли'}), 200

@app.route('/api/me', methods=['GET'])
def get_me():
    if 'username' in session:
        conn = sqlite3.connect('users.db')
        is_blocked = conn.execute('SELECT is_blocked FROM users WHERE
     username = ?', (session['username'],)).fetchone()
        conn.close()
        if is_blocked and is_blocked[0] == 1:
            session.clear()
            return jsonify({'error': 'Акаунт заблоковано'}), 403
        return jsonify({'username': session['username']}), 200
    return jsonify({'error': 'Не авторизовано'}), 401

@app.route('/')
def index(): return render_template('index.html')

@app.route('/api/upload', methods=['POST'])
def upload_files():
    user_id = session.get('user_id')
    user_folder = get_user_folder()
    if not user_id: return jsonify({'error': 'Не авторизовано'}), 401

    files = request.files.getlist('file')
    exchange_hours = request.form.get('exchange_hours', type=int,
     default=0)

          if not files or files[0].filename == '': return
     jsonify({'error': 'Файли не знайдено'}), 400

    is_public = 1 if exchange_hours > 0 else 0
    share_token = uuid.uuid4().hex if is_public else None
    expires_at = (datetime.now() +
     timedelta(hours=exchange_hours)).isoformat() if is_public else None
    auto_delete = 1 if is_public else 0
```

```
    conn = sqlite3.connect('users.db')
    for file in files:
        original_name = secure_filename(file.filename)
        physical_name = uuid.uuid4().hex +
     os.path.splitext(original_name)[1]
        file.seek(0, os.SEEK_END)
        file_size = file.tell()
        file.seek(0)
        file.save(os.path.join(user_folder, physical_name))
        conn.execute('INSERT INTO files (user_id, original_name,
     physical_name, size, is_public, share_token, expires_at, auto_delete)
     VALUES (?, ?, ?, ?, ?, ?, ?, ?)',
                     (user_id, original_name, physical_name, file_size,
     is_public, share_token, expires_at, auto_delete))
    conn.commit()
    conn.close()
    cleanup_expired_files()
    return jsonify({'message': 'Завантажено', 'share_token': share_token}),
     200

@app.route('/api/files', methods=['GET'])
def list_files():
    user_id = session.get('user_id')
    if not user_id: return jsonify({'error': 'Не авторизовано'}), 401
    cleanup_expired_files()
    conn = sqlite3.connect('users.db')
    conn.row_factory = sqlite3.Row
    rows = conn.execute('SELECT id, original_name, size, upload_date,
     is_public, share_token, expires_at, auto_delete FROM files WHERE
     user_id = ? ORDER BY upload_date DESC', (user_id,)).fetchall()


          conn.close()
    files = [{'id': r['id'], 'name': r['original_name'], 'size': r['size'],
     'date': r['upload_date'], 'is_public': bool(r['is_public']),
     'share_token': r['share_token'], 'expires_at': r['expires_at'],
     'auto_delete': bool(r['auto_delete'])} for r in rows]
    return jsonify(files)

@app.route('/api/delete/batch', methods=['POST'])
def delete_batch():
    user_id = session.get('user_id')
```

```
    if not user_id: return jsonify({'error': 'Не авторизовано'}), 401
    file_ids = request.json.get('file_ids', [])
    conn = sqlite3.connect('users.db')
    for file_id in file_ids:
        file_record = conn.execute('SELECT physical_name FROM files WHERE
     id = ? AND user_id = ?', (file_id, user_id)).fetchone()
        if file_record:
            file_path = os.path.join(get_user_folder(), file_record[0])
            if os.path.exists(file_path): os.remove(file_path)
            conn.execute('DELETE FROM shared_access WHERE file_id = ?',
     (file_id,))
            conn.execute('DELETE FROM files WHERE id = ?', (file_id,))
    conn.commit()
    conn.close()
    return jsonify({'message': 'Файли видалено'}), 200

@app.route('/api/share/public/batch', methods=['POST'])
def toggle_share_batch():
    user_id = session.get('user_id')
    data = request.json or {}
    file_ids = data.get('file_ids', [])
    action = data.get('action', 'close')

    conn = sqlite3.connect('users.db')
    if action == 'close':
        for file_id in file_ids:
            conn.execute('UPDATE files SET is_public = 0, expires_at =
     NULL, auto_delete = 0 WHERE id = ? AND user_id = ?', (file_id,
     user_id))

 conn.commit()
        conn.close()
        return jsonify({'message': 'Доступ закрито'}), 200
    else:
        hours = int(data.get('hours', 0))
        expires_at = (datetime.now() + timedelta(hours=hours)).isoformat()
     if hours > 0 else None
        share_token = uuid.uuid4().hex
        for file_id in file_ids:
            conn.execute('UPDATE files SET is_public = 1, share_token = ?,
     expires_at = ? WHERE id = ? AND user_id = ?', (share_token,
     expires_at, file_id, user_id))
        conn.commit()
```

```
        conn.close()
        return jsonify({'message': 'Доступ відкрито', 'share_token':
    share_token}), 200

@app.route('/api/share/user/<int:file_id>', methods=['POST'])
def share_with_user(file_id):
    user_id = session.get('user_id')
    target_username = request.json.get('username')
    conn = sqlite3.connect('users.db')
    file_owner = conn.execute('SELECT user_id FROM files WHERE id = ?',
     (file_id,)).fetchone()
    if not file_owner or file_owner[0] != user_id: return jsonify({'error':
     'Немає доступу'}), 403
    target_user = conn.execute('SELECT id FROM users WHERE username = ?',
     (target_username,)).fetchone()
    if not target_user: return jsonify({'error': 'Користувача не
     знайдено'}), 404
    if target_user[0] == user_id: return jsonify({'error': 'Це ваш власний
     файл'}), 400
    try:
        conn.execute('INSERT INTO shared_access (file_id,
     shared_with_user_id) VALUES (?, ?)', (file_id, target_user[0]))
        conn.commit()
        msg = f'Доступ надано: {target_username}'
    except sqlite3.IntegrityError: msg = f'Користувач {target_username} вже
     має доступ'


          conn.close()
    return jsonify({'message': msg}), 200

@app.route('/api/shared_with_me', methods=['GET'])
def shared_with_me():
    user_id = session.get('user_id')
    if not user_id: return jsonify({'error': 'Не авторизовано'}), 401
    cleanup_expired_files()
    conn = sqlite3.connect('users.db')
    conn.row_factory = sqlite3.Row
    rows = conn.execute('''SELECT f.id, f.original_name, f.size,
     f.upload_date, u.username as owner_name
                           FROM files f JOIN shared_access sa ON f.id =
     sa.file_id JOIN users u ON f.user_id = u.id
```

```
                            WHERE sa.shared_with_user_id = ?''',
     (user_id,)).fetchall()
    conn.close()
    files = [{'id': r['id'], 'name': r['original_name'], 'size': r['size'],
     'date': r['upload_date'], 'owner': r['owner_name']} for r in rows]
    return jsonify(files)

@app.route('/files/<int:file_id>')
def get_file(file_id):
    user_id = session.get('user_id')
    if not user_id: return render_template('error.html', code="401",
     title="Необхідна авторизація", message="Увійдіть в систему, щоб
     переглянути цей файл."), 401
    conn = sqlite3.connect('users.db')
    file_record = conn.execute('SELECT f.original_name, f.physical_name,
     f.user_id, u.username FROM files f JOIN users u ON f.user_id = u.id
     WHERE f.id = ?', (file_id,)).fetchone()
    if not file_record:
        conn.close()
        return render_template('error.html', code="404", title="Файл не
     знайдено", message="Файл не існує або він був видалений власником."),
     404
    owner_id = file_record[2]
    owner_username = file_record[3]


          if owner_id != user_id:
        has_access = conn.execute('SELECT 1 FROM shared_access WHERE
     file_id = ? AND shared_with_user_id = ?', (file_id,
     user_id)).fetchone()
        if not has_access:
            conn.close()
            return render_template('error.html', code="403", title="Доступ
     заборонено", message="У вас немає доступу до цього файлу. Зверніться
     до власника."), 403
    conn.close()
    target_folder = os.path.join(app.config['UPLOAD_FOLDER'],
     secure_filename(owner_username))
    return send_from_directory(target_folder, file_record[1],
     download_name=file_record[0], as_attachment=False)

@app.route('/s/<share_token>')
def shared_file_landing(share_token):
```

```
    cleanup_expired_files()
    conn = sqlite3.connect('users.db')
    recs = conn.execute('SELECT f.original_name, u.username, f.expires_at,
     f.size, f.auto_delete FROM files f JOIN users u ON f.user_id = u.id
     WHERE f.share_token = ? AND f.is_public = 1',
     (share_token,)).fetchall()
    conn.close()

    if not recs: return render_template('error.html', code="404",
     title="Посилання недійсне", message="Файли не знайдено. Можливо,
     термін обміну закінчився і вони були знищені."), 404

    expires_at_str = recs[0][2]
    if expires_at_str and datetime.now() >
     datetime.fromisoformat(expires_at_str):
        return render_template('error.html', code="403", title="Термін дії
     минув", message="Час життя цього посилання закінчився."), 403

    total_size = sum(r[3] for r in recs)
    batch_info = {
        'is_multiple': len(recs) > 1, 'count': len(recs), 'owner':
     recs[0][1], 'expires_at': recs[0][2],
        'total_size': format_bytes(total_size), 'auto_delete': recs[0][4],



'token': share_token,
        'files': [{'name': r[0], 'size': format_bytes(r[3])} for r in
     recs][:5]
    }
    return render_template('share.html', batch=batch_info)

@app.route('/s/<share_token>/download')
def shared_file_download(share_token):
    cleanup_expired_files()
    conn = sqlite3.connect('users.db')
    recs = conn.execute('SELECT f.original_name, f.physical_name,
     u.username, f.expires_at FROM files f JOIN users u ON f.user_id = u.id
     WHERE f.share_token = ? AND f.is_public = 1',
     (share_token,)).fetchall()
    conn.close()
```

```
    if not recs: return render_template('error.html', code="404", title="Не
     знайдено", message="Файли були видалені або посилання недійсне."), 404
    if recs[0][3] and datetime.now() > datetime.fromisoformat(recs[0][3]):
     return render_template('error.html', code="403", title="Термін дії
     минув", message="Час життя цього посилання закінчився."), 403

    if len(recs) == 1:
        return
     send_from_directory(os.path.join(app.config['UPLOAD_FOLDER'],
     secure_filename(recs[0][2])), recs[0][1], download_name=recs[0][0],
     as_attachment=True)

    memory_file = io.BytesIO()
    with zipfile.ZipFile(memory_file, 'w') as zf:
        for rec in recs:
            file_path = os.path.join(app.config['UPLOAD_FOLDER'],
     secure_filename(rec[2]), rec[1])
            if os.path.exists(file_path): zf.write(file_path,
     arcname=rec[0])
    memory_file.seek(0)
    return send_file(memory_file, download_name="MyCloud_Files.zip",
     as_attachment=True)




@app.route('/api/stats', methods=['GET'])
def get_stats():
    try:
        cleanup_expired_files()
        user_used = sum(os.path.getsize(os.path.join(get_user_folder(), f))
     for f in os.listdir(get_user_folder()) if
     os.path.isfile(os.path.join(get_user_folder(), f)))
        total, _, free = shutil.disk_usage(app.config['UPLOAD_FOLDER'])
        return jsonify({'total': total, 'used': user_used, 'free': free})
    except: return jsonify({'total': 0, 'used': 0, 'free': 0})

# ========================================
# ГЛОБАЛЬНІ ОБРОБНИКИ ПОМИЛОК
# ========================================
@app.errorhandler(404)
def page_not_found(e):
```

```
    return render_template('error.html', code="404", title="Сторінку не
     знайдено", message="Можливо, ви ввели неправильну адресу, або сторінка
     була видалена."), 404

@app.errorhandler(500)
def internal_server_error(e):
    return render_template('error.html', code="500", title="Внутрішня
     помилка сервера", message="Наші алгоритми вже фіксують проблему.
     Спробуйте оновити сторінку пізніше."), 500

@app.errorhandler(413)
def request_entity_too_large(e):
    return render_template('error.html', code="413", title="Файл занадто
     великий", message="Сервер не може обробити такий великий об'єм даних
     за один раз."), 413

@app.errorhandler(405)
def method_not_allowed(e):
    return render_template('error.html', code="405", title="Метод не
     дозволено", message="Система безпеки заблокувала цей запит."), 405

if __name__ == '__main__': app.run(host='0.0.0.0', port=5001, debug=True)
```