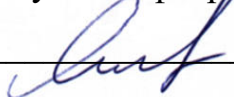


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**Національний університет кораблебудування імені адмірала Макарова****Навчально-науковий інститут комп'ютерних наук та управління проєктами
Кафедра інформаційних управляючих систем та технологій**

"Допущений до захисту"

Завідувач кафедри ІУСТ



к.т.н, доц. Михелєв І.Л.

" ____ " ____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА**на здобуття ступеня вищої освіти "магістр"****на тему:****“Розробка інформаційної системи хмарного сховища на основі системного
аналізу методів підвищення якості зберігання даних”**

Виконав: студент групи 6145 м

Небиков Є.
(підпис)

Керівник роботи:

доцент кафедри ІУСТ, к.т.н.,
(посада, науковий ступінь, вчене звання)Ажищев В.Ф.
(підпис)

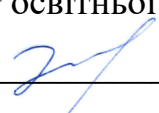
м. Миколаїв – 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова
 Навчально-науковий інститут комп'ютерних наук та управління проектами
 Кафедра інформаційних управляючих систем та технологій
 Спеціальність 124 "Системний аналіз"
 Освітня програма "Системний аналіз"

"ЗАТВЕРДЖУЮ"

Гарант освітньої програми


 д.ф-м.н, проф. Ушкац М.В.

" ____ " _____ 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ на здобуття ступеня вищої освіти "магістр"

Студенту Небикову Євгену Володимировичу

(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка інформаційної системи хмарного сховища на основі системного аналізу методів підвищення якості зберігання даних

Керівник роботи: Ажищев Віктор Федорович

Затверджені наказом ректора № 1355-уч. від "14" листопада 2025 року.

2. Термін подання роботи: 15 грудня 2025 р.

3. Вихідні дані до проекту (роботи) ДСТУ щодо обробки інформації, літературні джерела, технічна документація на існуючі аналоги систем,

4. Перелік питань, що належать до розробки (найменування розділів) Вступ. 1.Розробка концепції інформаційної системи. 2.Розробка технічного проекту інформаційної системи. 3.Реалізація проекту інформаційної системи. 4.Охорона праці. Висновки.

5. Перелік презентаційних матеріалів: Слайди: Розробка інформаційної системи хмарного сховища; Актуальність, ціль, завдання; Об'єкт дослідження; Концепція системи; Технічний проект; Моделі бази даних; Програмне забезпечення; Технічне забезпечення; Використання системи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Ажищев В.Ф.	05.09.2024	22.12.2024
2	Ажищев В.Ф.	23.12.2024	15.06.2025
3	Ажищев В.Ф.	15.06.2025	25.09.2025
4	Ажищев В.Ф.	25.09.2025	14.11.2025
5	Ажищев В.Ф.	25.09.2025	14.11.2025
6	Ажищев В.Ф.	25.09.2025	14.11.2025

7. Дата видачі завдання 05 вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів роботи	Примітка
1	Дослідження та аналіз предметної галузі	22.12.2024	
2	Розробка концепції	15.03.2025	
3	Розділ з розробки проєктних рішень по системі (та її частинах)	15.06.2025	
4	Реалізація проєкту та розробка робочої документації	25.09.2025	
5	Обчислення економічної ефективності	14.11.2025	
6	Вирішення питань охорони праці та навколишнього середовища	14.11.2025	
7	Оформлення пояснювальної записки	14.11.2025	
8	Подання магістерської роботи на перевірку на плагіат	16.11.2025	
9	Подання магістерської роботи рецензенту	20.11.2025	
10	Подання магістерської роботи на державний захист	15.12.2025	

Студент

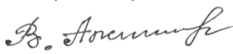


(підпис)

Небиков Є.В.

(ПІБ)

Керівник роботи



(підпис)

Ажищев В.Ф.

(ПІБ)

АНОТАЦІЯ

Кваліфікаційна магістерська робота **“Розробка інформаційної системи хмарного сховища на основі системного аналізу методів підвищення якості зберігання даних”** за освітньою програмою "Системний аналіз" спеціальності 124 "Системний аналіз".

Робота присвячена розробці інформаційної системи хмарного сховища на основі системного аналізу методів підвищення якості зберігання даних.

Актуальність теми зумовлена стрімким зростанням обсягів даних та переходом бізнес-процесів у хмарне середовище, що висуває підвищені вимоги до надійності, цілісності та безпеки зберігання інформації. Необхідність поєднання сучасних архітектурних рішень із ефективними методами захисту даних (реплікація, шифрування, резервне копіювання) визначає значущість даного дослідження.

Метою роботи є розробка інформаційної системи хмарного сховища, яка забезпечує надійне, безпечне та зручне керування файлами користувачів у хмарному середовищі на основі системного аналізу методів підвищення якості зберігання даних.

Об'єктом дослідження є процес зберігання, обробки та доступу до даних у хмарних інформаційних системах. У ході роботи було проведено аналіз конкурентних хмарних сховищ (Google Drive, Dropbox, MEGA, OneDrive, iCloud), досліджено ефективність методів забезпечення якості зберігання даних, розроблено концепцію та технічний проект системи, включаючи моделі даних, програмне та технічне забезпечення.

Практичне значення роботи полягає у створенні концепції та технічного проекту інформаційної системи, яка може слугувати основою для реального сервісу надійного та безпечного зберігання даних.

Робота виконана українською мовою, складається з 105 сторінок машинописного тексту, містить 22 ілюстрації, 3 таблиці і список використаних джерел з 50 найменувань.

Ключові слова: хмарне сховище, якість зберігання даних, системний аналіз, реплікація, шифрування, технічний проект, інформаційна система, захист даних.

ABSTRACT

Master's qualification thesis "Development of a Cloud Storage Information System Based on System Analysis of Methods for Improving Data Storage Quality" in the educational program "System Analysis", specialty 124 "System Analysis".

The thesis is devoted to the development of a cloud storage information system based on system analysis of methods for improving data storage quality.

The relevance of the topic is determined by the rapid growth of data volumes and the transition of business processes to the cloud environment, which imposes increased requirements for the reliability, integrity and security of information storage. The need to combine modern architectural solutions with effective data protection methods (replication, encryption, backup) defines the significance of this research.

The aim of the thesis is to develop a cloud storage information system that provides reliable, secure and convenient management of user files in a cloud environment based on system analysis of methods for improving data storage quality.

The object of the research is the process of storing, processing and accessing data in cloud information systems. In the course of the work, an analysis of competing cloud storage services (Google Drive, Dropbox, MEGA, OneDrive, iCloud) was carried out, the effectiveness of methods for ensuring data storage quality was investigated, and the concept and technical design of the system were developed, including data models, software and hardware components.

The practical significance of the thesis lies in the creation of a concept and technical design of an information system that can serve as a basis for a real service for reliable and secure data storage.

The thesis is written in Ukrainian, consists of 105 pages of typescript, contains 22 illustrations, 3 tables and a list of references with 50 sources.

Keywords: cloud storage, data storage quality, system analysis, replication, encryption, technical design, information system, data protection.

ЗМІСТ

Вступ.....	8
Розділ 1 Дослідження методів забезпечення якості зберігання даних у хмарних інформаційних системах.....	10
1.1 Опис об'єкту дослідження.....	12
1.2 Аналіз предметної області.....	22
1.3 Дослідження ефективності методів забезпечення якості зберігання даних у хмарних інформаційних системах.....	27
1.4 Формування вимог до інформаційної системи.....	32
1.5 Розробка концепції інформаційної системи.....	36
Розділ 2 Розробка технічного проекту інформаційної системи.....	42
2.1 Загальносистемні рішення.....	42
2.2 Рішення по інформаційному забезпеченню.....	49
2.2.1 Розробка концептуальної моделі даних.....	50
2.2.2 Розробка логічної моделі даних.....	51
2.2.3 Розробка фізичної моделі даних.....	53
2.3 Рішення з математичного забезпечення.....	56
2.4 Рішення з програмного забезпечення.....	57
2.5 Рішення з технічного забезпечення.....	66
2.6 Рішення щодо зберігання файлів.....	67
Розділ 3 Реалізація проекту інформаційної системи хмарного сховища.....	70
Розділ 4 Техніко-економічне обґрунтування розробки інформаційної системи хмарного сховища	74
Розділ 5 Охорона праці.....	81
Розділ 6 Охорона навколишнього середовища	86
Висновки.....	92
Список використаної літератури.....	94
Додаток А - Інструкція користувачу.....	98
Додаток Б - Текст програми.....	101

ВСТУП

Кваліфікаційна робота на тему **«Розробка інформаційної системи хмарного сховища на основі системного аналізу методів підвищення якості зберігання даних»** зумовлена актуальною потребою у підвищенні якості зберігання файлів у хмарних інформаційних системах. Зростання обсягів цифрових даних та активне використання хмарних сервісів вимагають застосування ефективних методів забезпечення надійності, цілісності, доступності та безпеки збереженої інформації.

Актуальність роботи полягає у необхідності вдосконалення підходів до зберігання даних у хмарному середовищі шляхом використання сучасних методів підвищення якості зберігання файлів, що дозволяє зменшити ризики втрати інформації, підвищити стабільність роботи системи та зручність її використання.

Метою кваліфікаційної роботи є розробка інформаційної системи хмарного сховища на основі системного аналізу методів підвищення якості зберігання даних, яка забезпечує надійне, безпечне та ефективне керування файлами користувачів.

Наукова новизна роботи полягає у застосуванні результатів системного аналізу методів підвищення якості зберігання даних при розробці інформаційної системи хмарного сховища, що дозволяє підвищити надійність і ефективність процесів зберігання файлів.

Робота складається зі вступу, п'яти розділів, висновків, списку використаних джерел та додатків.

Розділ 1 «Дослідження методів забезпечення якості зберігання даних у хмарних інформаційних системах» виконано аналіз предметної області, розглянуто існуючі хмарні сховища, досліджено методи підвищення якості зберігання даних та сформовано вимоги до інформаційної системи.

Розділ 2 «Розробка технічного проекту інформаційної системи хмарного сховища» розроблено загальносистемні рішення, моделі даних, а також обґрунтовано програмне, математичне та технічне забезпечення системи.

Розділ 3 «Реалізація проекту інформаційної системи хмарного сховища» описано процес реалізації розробленої системи та основні функціональні можливості програмного продукту.

Розділ 4 наведено техніко-економічне обґрунтування доцільності розробки інформаційної системи.

Розділ 5 розглянуто питання охорони праці та охорони навколишнього середовища.

На основі проведеного дослідження, було створено інформаційну систему хмарного сховища, впровадження якої забезпечує підвищення якості зберігання даних, надійності та безпеки роботи з інформаційними ресурсами, що у підсумку сприяє підвищенню конкурентоспроможності компанії.

Розділ 1 Дослідження методів забезпечення якості зберігання даних у хмарних інформаційних системах.

1.1 Опис об'єкту дослідження.

Зростаючі потреби бізнесу спонукають підприємства впроваджувати додаткову інфраструктуру зберігання даних. Багато підприємств стають дуже залежними від постачальників послуг хмарних сховищ у своїх бізнес-операціях. Об'єктом дослідження є компанія, спеціалізована на розробці інформаційної системи хмарного сховища. Ця система призначена для бізнесів, які мають високу потребу в безпечному та надійному зберіганні своїх даних, робочих команд, які потребують спільний доступ до файлів для вирішення робочих задач і, також, звичайних користувачів, яким потрібен зручний інструмент для збереження файлів з подальшим отриманням до них доступу з будь-якого пристрою. Схема взаємодії компанії з бізнесами, командами та користувачами зображена на рисунку 1.1.

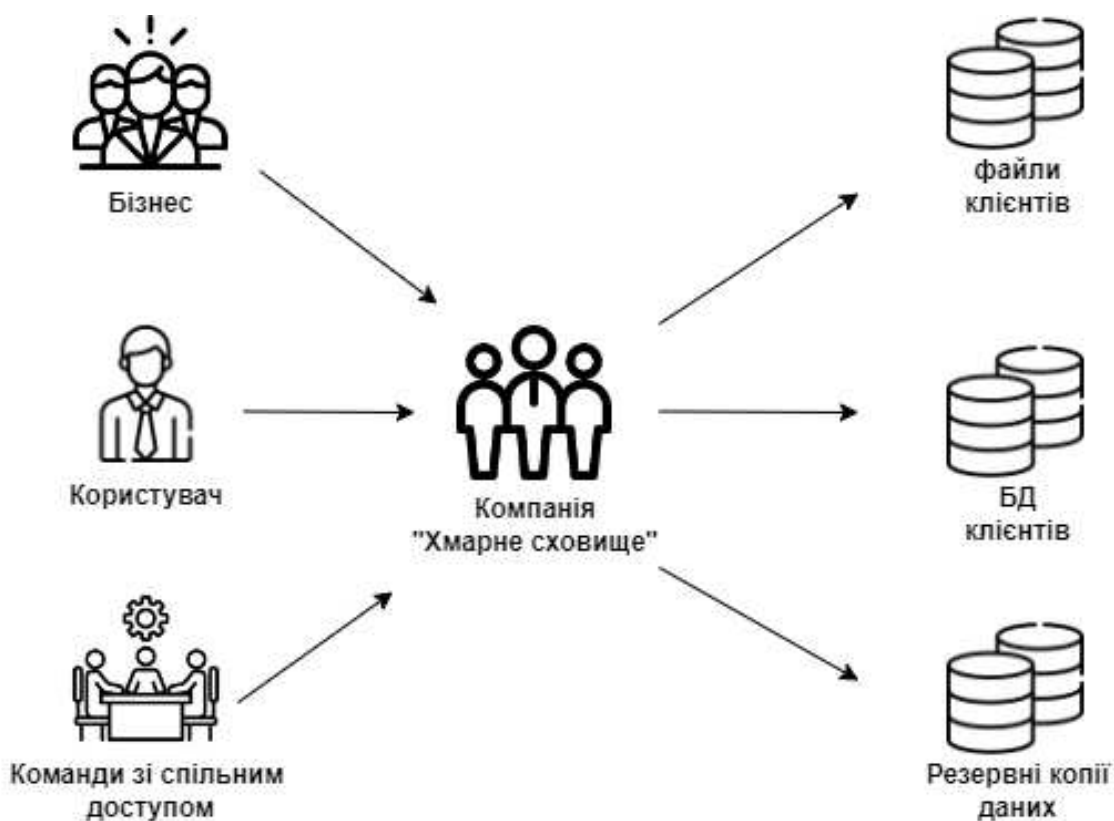


Рисунок 1.1 - Схема взаємодії компанії-стартапу з користувачами

На схемі зображено три групи користувачів:

- Бізнеси: представлені як організації, які потребують безпечного та надійного зберігання даних, таких як файли, бази даних та резервні копії;
- Робочі команди: представлені як групи людей, які співпрацюють над проектами та потребують спільного доступу до даних;
- Звичайні користувачі: представлені як особи, які використовують хмарні сховища для зберігання особистих даних, таких як фото, відео та документи;

Компанія, що надає послуги зі збереження даних є об'єктом дослідження і розташована в центрі схеми і пропонує широкий спектр послуг, включаючи:

- Зберігання даних: компанія надає безпечне та надійне місце для зберігання даних користувачів;
- Доступ до даних: користувачі можуть отримувати доступ до своїх даних з будь-якого пристрою з підключенням до Інтернету;
- Спільний доступ до даних: користувачі можуть ділитися своїми даними з іншими людьми або групами людей;
- Захист даних: компанія використовує передові технології для захисту даних користувачів від несанкціонованого доступу, використання або розкриття.

Бізнеси: можуть використовувати хмарні сховища для зберігання своїх даних, таких як фінансова інформація, кадрові записи та дані про клієнтів. Це може допомогти їм покращити безпеку даних, знизити витрати на зберігання та підвищити продуктивність.

Робочі команди: можуть використовувати хмарні сховища для спільного доступу до файлів, документів та інших даних. Це може допомогти їм покращити співпрацю, підвищити продуктивність та економити час.

Звичайні користувачі: можуть використовувати хмарні сховища для зберігання своїх особистих даних, таких як фото, відео та документи. Це може

допомогти їм звільнити місце на своїх пристроях, отримати доступ до своїх даних з будь-якого місця та захистити свої дані від втрати.

Переваги, які отримують клієнти компанії «Хмарне сховище»:

- **Безпека:** компанія використовує передові технології для захисту даних користувачів від несанкціонованого доступу, використання або розкриття;
- **Надійність:** компанія гарантує доступність даних користувачів 99,9% часу;
- **Масштабованість:** компанія може легко масштабувати свої послуги, щоб задовольнити потреби будь-якого бізнесу або користувача;
- **Доступність:** користувачі можуть отримувати доступ до своїх даних з будь-якого пристрою з підключенням до Інтернету;
- **Економічність:** хмарні сховища можуть допомогти користувачам економити гроші на витратах на зберігання даних.

Взаємодія бізнесів і компанії, що надає послуги хмарних сховищ, є ключовою для успішного функціонування обох сторін. Схоже на симбіоз, це партнерство забезпечує взаємну вигоду та підтримує сприятливе середовище для обміну даними та послугами.

Для бізнесів використання хмарних сховищ є важливою стратегічною складовою їхньої інформаційної інфраструктури. Вони залежать від компаній хмарних сховищ для зберігання та захисту своїх даних, забезпечуючи доступ до них з будь-якої точки світу, з будь-якого пристрою. Це надає їм гнучкість та мобільність в управлінні даними та сприяє підвищенню продуктивності працівників.

З іншого боку, компанії, які надають послуги хмарних сховищ, отримують можливість пропонувати свої послуги бізнесам різних галузей. Шляхом забезпечення безпеки, надійності та ефективності зберігання даних, вони приваблюють клієнтів і створюють стабільні джерела доходу. Партнерство з

бізнесами дозволяє цим компаніям розширювати свої можливості та розвиватися на ринку інформаційних технологій.

Навіть звичайні користувачі знаходять у хмарних сховищах велике застосування. Вони можуть зберігати свої особисті файли та документи, отримуючи доступ до них з будь-якого пристрою та з будь-якої точки світу. Це забезпечує їм зручність, безпеку та надійність у зберіганні їхніх цінних даних.

Робочі команди також користуються хмарними сховищами для ефективного обміну робочими файлами та спільної роботи над проектами. Завдяки цьому, вони можуть швидко та ефективно співпрацювати, незалежно від свого місця перебування, що збільшує продуктивність та прискорює процес роботи.

Взаємодія бізнесів, звичайних користувачів та робочих команд з компанією, що надає послуги хмарних сховищ, забезпечує різноманітність та всебічне задоволення потреб різних категорій користувачів.

Основні завдання компанії, що займається розробкою хмарного сховища.

Розробка та вдосконалення хмарної платформи:

- Проектування та масштабування архітектури хмарного сховища;
- Вибір та впровадження відповідних технологій віртуалізації та контейнеризації;
- Забезпечення високої доступності та стійкості до відмов;
- Забезпечення безпеки даних та захисту від кібератак;
- Розробка та вдосконалення функціоналу хмарного сховища;
- Забезпечення зберігання та доступу до файлів, баз даних та інших типів даних;
- Створення та надання користувачам інструментів для роботи з даними;
- Інтеграція з іншими хмарними сервісами та сторонніми додатками;
- Підтримка та оновлення хмарної платформи;
- виправлення помилок та усунення проблем;
- Впровадження нових функцій та оновлення існуючих;

- Моніторинг продуктивності та оптимізація ресурсів.

Забезпечення безпеки та конфіденційності даних:

- Впровадження та дотримання жорстких політик безпеки;
- Шифрування даних на льоту та в стані спокою;
- Контроль доступу на основі ролей та багатфакторна аутентифікація;
- Регулярне резервне копіювання та відновлення даних;
- Моніторинг та аудит безпеки;
- Відповідність нормативним вимогам;
- GDPR, CCPA, HIPAA та інші нормативні акти;
- Захист персональних даних та конфіденційної інформації;
- Прозорість та звітність щодо практики обробки даних.

Підтримка користувачів:

- Надання якісної служби підтримки;
- Цілодобова багатоканальна підтримка (телефон, чат, email);
- Кваліфіковані фахівці з досвідом роботи з хмарними сховищами;
- База знань та навчальні матеріали для самостійного вирішення проблем;
- Навчання користувачів;
- Документація та інструкції з використання хмарного сховища;
- Вебінари та онлайн-курси для підвищення обізнаності користувачів;
- Персональні консультації та допомога в налаштуванні хмарного сховища.

Маркетинг та просування хмарного сховища:

- Розробка та реалізація маркетингової стратегії;
- Визначення цільового ринку та позиціонування хмарного сховища;
- Створення рекламних кампаній та просування в ЗМІ;
- Участь у виставках та конференціях;
- Створення та підтримка позитивного іміджу;
- Наголошення на перевагах хмарного сховища;

- Демонстрація успішних кейсів та відгуків клієнтів;
- Забезпечення відкритої та прозорої комунікації.

Фінансове планування та управління:

- Розробка та контроль бюджету;
- Визначення витрат на розробку, експлуатацію та маркетинг хмарного сховища;
- Встановлення та контроль цін на послуги хмарного сховища;
- Аналіз та прогнозування фінансових показників;
- Залучення інвестицій;
- Підготовка та презентація інвестиційного проекту;
- Переговори з інвесторами та залучення коштів;
- Ефективне використання залучених інвестицій.

Управління командою та розробка кадрової політики:

- Підбір та найм кваліфікованих фахівців;
- Розробка та ведення кадрового резерву;
- Проведення співбесід та оцінка кандидатів [1].

Структура компанії, що надає послуги хмарного сховища зображена на рисунку 1.2.

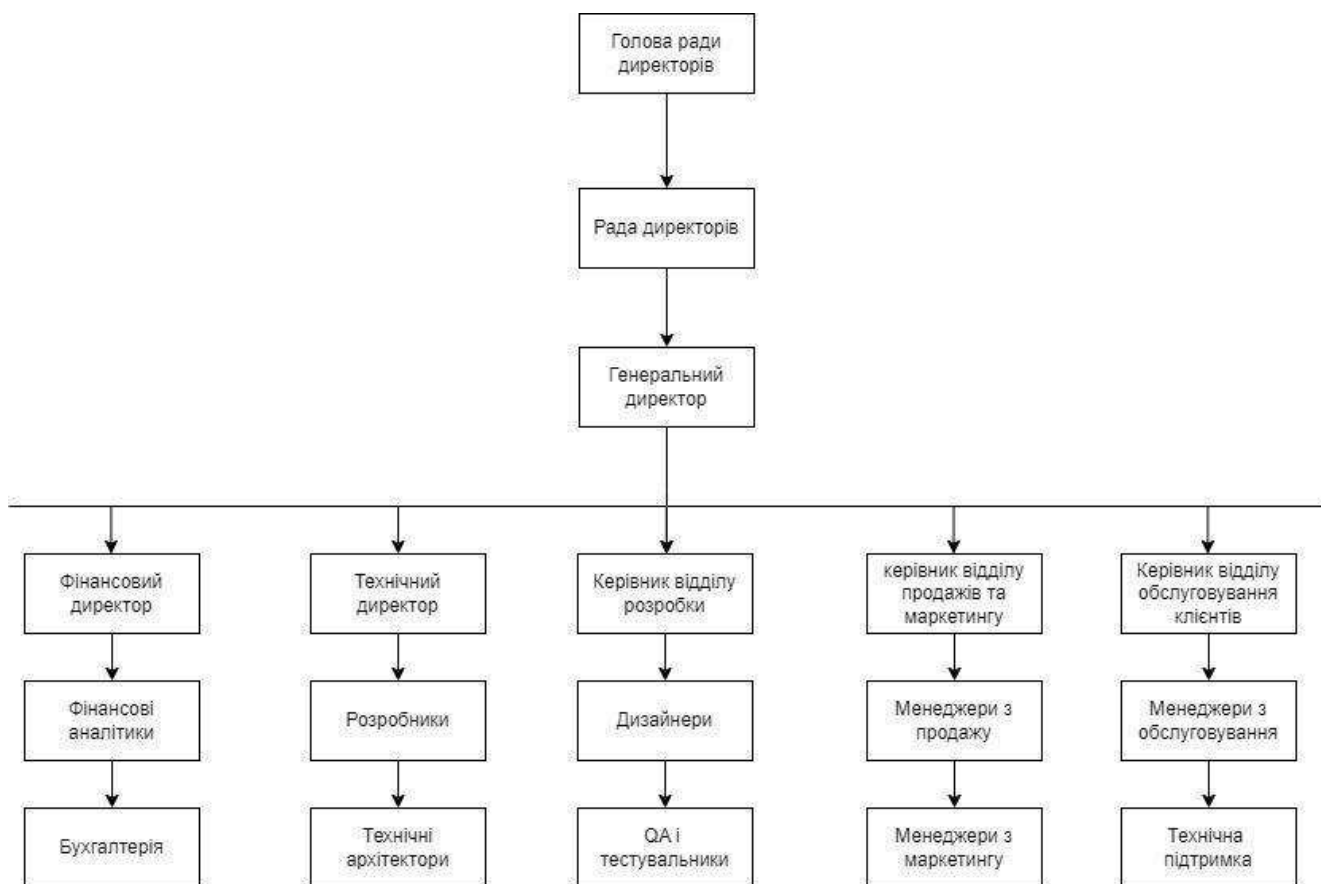


Рисунок 1.2 – Структура компанії

Виконавче керівництво є ключовим органом управління компанії, відповідальним за формулювання стратегії, прийняття стратегічних рішень та керівництво загальними діями компанії. На чолі цього відділу стоїть генеральний директор, який має на меті забезпечення ефективного функціонування всієї організації. Головна мета виконавчого керівництва полягає у досягненні стратегічних цілей компанії та забезпеченні її стабільного розвитку.

Крім того, виконавче керівництво підпорядковується раді директорів, яка виступає як орган управління, відповідальний за розробку стратегії компанії, здійснення загального керівництва та контроль над діяльністю виконавчого

органу. Рада директорів здійснює свої функції відповідно до статуту, кодексу корпоративного управління та інших внутрішніх документів компанії. Вона приймає стратегічні рішення, керуючись інтересами всіх учасників ради. Рішення, які приймає рада директорів, мають вирішальне значення для подальшого розвитку компанії та її успішної діяльності на ринку.

Технічне відділення відіграє велику роль у розробці та підтримці хмарного сховища. Основним завданням цього відділу є забезпечення технічної підтримки, розробка нових функцій та забезпечення безпеки та надійності системи. Відділ має таку ієрархічну структуру:

Керівництво технічного відділу відповідає за стратегічне управління та визначення загальної стратегії розвитку хмарного сховища. Його склад включає технічного директора або головного технічного офіцера (СТО), який відповідає за напрямок розвитку технологічних рішень та інновацій.

Розробники програмного забезпечення: цей відділ відповідає за розробку та підтримку програмного забезпечення для хмарного сховища. Включає в себе програмістів, інженерів програмного забезпечення та архітекторів систем.

Інфраструктурний відділ забезпечує функціонування інфраструктури, необхідної для роботи хмарного сховища. Він включає системних адміністраторів, інженерів мережі та адміністраторів баз даних.

Безпека і захист інформації: цей відділ відповідає за забезпечення безпеки та захисту інформації в хмарному сховищі. Включає в себе спеціалістів з кібербезпеки, аналітиків безпеки та інженерів з захисту даних.

Технічна підтримка: Відділ технічної підтримки надає допомогу користувачам у вирішенні технічних проблем та запитань щодо використання хмарного сховища. Це включає в себе технічних спеціалістів, які працюють на лінії підтримки та надають консультації через телефон, чат або електронну пошту.

Відділ розробки продукту є ключовим в складі компанії, що займається хмарним сховищем. Він відповідає за концептуалізацію, розробку та

впровадження нових функцій та продуктів у хмарному середовищі. Основними функціями цього відділу є:

- Проектування продукту: Розробка концепцій та визначення вимог до нових продуктів та функціоналу на основі аналізу ринку, потреб користувачів та технологічних можливостей;
- Розробка програмного забезпечення: Створення програмних рішень для забезпечення надійності, безпеки та продуктивності хмарного сервісу;
- Тестування та впровадження: Проведення випробувань нових продуктів та функціоналу перед їхнім запуском на ринок, а також впровадження у виробниче середовище;
- Підтримка продукту: Надання технічної підтримки та вирішення технічних питань щодо продуктів хмарного сховища;
- Стратегічне планування: Визначення стратегії розвитку продукту на майбутнє та впровадження інноваційних технологій для підвищення конкурентоспроможності компанії.

Відділ продажів та маркетингу відповідає за розвиток та збільшення клієнтської бази компанії, а також за просування її продуктів на ринку. Основними функціями цього відділу є:

- Маркетингові дослідження: Проведення аналізу ринку, визначення цільової аудиторії та конкурентного середовища для розробки стратегій маркетингу;
- Продажі та обслуговування клієнтів: Залучення нових клієнтів та підтримка вже існуючих, надання їм інформації про продукти та послуги компанії;
- Створення маркетингових кампаній: Розробка та впровадження маркетингових стратегій, рекламних кампаній та просування бренду компанії;
- Аналіз результатів: Моніторинг ефективності маркетингових заходів та аналіз їхнього впливу на продажі та сприйняття бренду;
- Партнерські відносини: Розвиток та підтримка партнерських відносин з іншими компаніями та організаціями для спільного розвитку ринку.

Відділ обслуговування клієнтів відіграє ключову роль у забезпеченні задоволеності та ретенції клієнтів. Основними завданнями цього відділу є:

- Клієнтська підтримка: Надання оперативної та професійної підтримки клієнтам щодо запитань, проблем або запитів, які виникають під час використання продуктів чи послуг компанії;

- Розв'язання конфліктів: Вирішення будь-яких конфліктних ситуацій з клієнтами з метою забезпечення високої якості обслуговування та підтримки позитивних відносин;

- Зворотний зв'язок: Збір, аналіз та зворотний зв'язок з клієнтами для отримання їхніх відгуків, пропозицій та скарг, що можуть бути використані для покращення продуктів та послуг;

- Тренінг та навчання: Надання користувачам навчальних матеріалів, інструкцій та тренінгів для ефективного використання продуктів чи послуг компанії;

- Клієнтська база даних: Управління та оновлення бази даних клієнтів для забезпечення актуальності та точності інформації.

Фінансовий відділ відповідає за фінансове управління та забезпечення фінансової стабільності компанії. Основні функції цього відділу включають:

- Бухгалтерський облік: Ведення обліку фінансової діяльності компанії, включаючи виписку рахунків, облік витрат та доходів, оподаткування тощо;

- Фінансове планування та аналіз: Розробка фінансових стратегій та планів, а також аналіз фінансових показників для прийняття ефективних управлінських рішень;

- Керування кредитами та фінансовими ресурсами: Контроль за кредитними лініями, керуванням ліквідністю та оптимізація використання фінансових ресурсів компанії;

- Фінансова звітність: Підготовка фінансової звітності та звітів для внутрішніх та зовнішніх зацікавлених сторін, включаючи аудиторів, партнерів та інвесторів;

- Управління ризиками: Виявлення, оцінка та керування фінансовими ризиками компанії з метою мінімізації можливих негативних наслідків.

Для подальшої розробки інформаційної системи хмарного сховища необхідно провести детальний аналіз предметної області на основі опису об'єкту дослідження, що дозволить визначити історичний контекст, сучасні тенденції та ключові переваги і недоліки існуючих рішень.

1.2 Аналіз предметної області.

Історія зберігання даних свідчить про постійний пошук ефективних та безпечних методів зберігання важливої інформації. Спочатку цей процес обмежувався зберіганням на папері, що, хоча і було стандартом свого часу, зазнало обмежень у відношенні об'єму та доступності даних. Жорсткі диски, які з'явилися в середині 20-го століття, принесли значне полегшення в зберіганні даних, але їх обмежені можливості масштабування та вразливість до втрати даних внаслідок непередбачених обставин змусило фахівців шукати альтернативні методи зберігання даних, внаслідок чого і було винайдено хмарні обчислення.

“Хмарні обчислення винайшов Джозеф Ліклайдер у 1960-х роках під час його роботи над ARPANET, щоб дати людям вільний доступ до даних у будь-якій точці планети та будь-коли. Однак, Курт Воннегут каже про «хмару», що «це занадто складно для всіх» у своїй книжці «Сирени Титана».

CompuServe запропонував його споживачам для використання 128 КБ. дискового простору, що підходить для зберігання будь-яких файлів.

AT&T запустила PersonaLink — інтернет-платформу для особистого та ділового спілкування на підприємстві. Даний сервіс був одним із перших веборієнтованих повністю, у рекламних роликах якого сказано «Ти можеш вважати наше віртуальне місце зустрічі хмариною!» (you can think of our electronic meeting place as the cloud).

Вебсервіси Amazon представили службу зберігання хмари AWS S3 у 2006 році й отримали широке визнання та прийняття як постачальник сховищ для таких популярних сервісів, як SmugMug, Dropbox та Pinterest.

У 2005 році компанія Vox оголосила про онлайн-обмін файлами та службу управління персональним хмарним вмістом для підприємств”[7].

За останні десятиліття інтенсивно розвивалася та набувала популярності технологія хмарних сховищ. Ця технологія відкрила нові перспективи в зберіганні та обробці даних, забезпечуючи їх безпеку, доступність та масштабованість на недосяжному раніше рівні.

Зберігання даних у хмарі виявилось ефективним рішенням для підприємств у всьому світі. Комерційні дані вимагають великого обсягу місця для зберігання, який може різнитися від малих компаній до великих корпорацій.

Зі зростанням обсягу даних, що генеруються за допомогою Інтернету речей (IoT), зберігання інформації на місці стає витратним. Чимало підприємств вирішили перемістити свої дані в хмарні сховища, звернувшись за допомогою до компаній, що спеціалізуються на цих технологіях.

Впровадження хмарних сервісів та аналітики великих обсягів даних зростає швидкими темпами. Разом з цим, збільшення цифровізації спричинило значне збільшення кількості даних, що генеруються та обробляються підприємствами.

Зростаючий попит на доступ до даних призводить до використання кількох пристроїв для отримання інформації, що зберігається в центрах обробки даних. Це створює потребу в потужних мережах, щоб оператори могли надавати якісніші послуги.

Виробники процесорів, жорстких і твердотільних дисків, такі як Intel, Seagate і Western Digital, прагнуть підвищити ефективність систем збереження даних за допомогою інноваційних пропозицій.

Значно зросла потреба в доступі до даних і кількості розгорнутих додатків. Що стосується інфраструктури зберігання даних, то споживачі (B2C) зростають разом зі збільшенням кількості користувачів Інтернету в усьому світі.

Використання ІТ-інфраструктури також зросло з появою віртуалізації, що призвело до конкуренції між компаніями. Щоб отримати кращий контроль над ІТ-інфраструктурою, організації впроваджують гіперконвергентну інфраструктуру.

1.2.1 Аналіз популярних хмарних сховищ.

Для порівняльного аналізу було обрано п'ять найбільш розповсюджених хмарних сховищ: Google Drive, Dropbox, OneDrive, iCloud та MEGA. Критеріями оцінювання виступають обсяг безкоштовного сховища, наявність шифрування за замовчуванням, зручність інтеграції в екосистему постачальника, підтримка кросплатформеності, а також додаткові механізми захисту й керування версіями файлів. Такий підхід дає змогу формалізувати порівняння та надалі використати отримані результати для обґрунтування вимог до власної інформаційної системи хмарного сховища.

З огляду на результати аналізу літератури та публічної документації сервісів, для кожного сховища введемо інтегральний показник якості QQ , що обчислюється як зважена сума нормованих оцінок за окремими критеріями: обсяг безкоштовного простору, наявність шифрування за замовчуванням, підтримка версіонування, інтеграція з офісним ПЗ та кросплатформенність.

Для порівняння варіантів хмарних сховищ використовуються такі вихідні дані: обсяг безкоштовного сховища (Free, ГБ) з фактичними значеннями для кожного сервісу, нормований показник обсягу за формулою $S = \frac{50}{Free}$, де 50 ГБ є максимальним безкоштовним обсягом, а також бальна оцінка функцій (шифрування, версіонування, інтеграція, кросплатформеність) від 0 до 1, де 1 означає повну реалізацію функції, 0,5 - часткову реалізацію з обмеженнями, а 0 - повну відсутність.

Таблиця 1.1 – Порівняльна характеристика популярних хмарних сховищ

Сервіс	Без.обсяг, ГБ	Показ. обсягу S	Шифр. за замовчуванням (E)	Верс. файлів (V)	Глибоінтеграція з офісним ПЗ (I)	Кросплатформеність (C)	Інтегральний показник якості Q
Google Drive	15	0,30	0,5	1	1	1	$0,30 \cdot 0,3 + 0,5 \cdot 0,2 + 1 \cdot 0,2 + 1 \cdot 0,15 + 1 \cdot 0,15 = 0,79$
Dropbox	2	0,04	1	1	0,5	1	$0,04 \cdot 0,3 + 1 \cdot 0,2 + 1 \cdot 0,2 + 0,5 \cdot 0,15 + 1 \cdot 0,15 = 0,69$
OneDrive	5	0,10	1	0,5	1	1	$0,10 \cdot 0,3 + 1 \cdot 0,2 + 0,5 \cdot 0,2 + 1 \cdot 0,15 + 1 \cdot 0,15 = 0,74$
iCloud	5	0,10	1	0,5	1	0,5	$0,10 \cdot 0,3 + 1 \cdot 0,2 + 0,5 \cdot 0,2 + 1 \cdot 0,15 + 0,5 \cdot 0,15 = 0,66$
MEGA	50	1,00	1	0	0,5	0,5	$1 \cdot 0,3 + 1 \cdot 0,2 + 0 \cdot 0,2 + 0,5 \cdot 0,15 + 0,5 \cdot 0,15 = 0,70$

Результати розрахунку інтегрального показника якості $Q = 0,3S + 0,2E + 0,2V + 0,15I + 0,15CQ = 0,3S + 0,2E + 0,2V + 0,15I$, де $S = \frac{Free}{50}$ —

нормований обсяг безкоштовного сховища, E, V, I, C — бальні оцінки шифрування, версіонування, інтеграції та кросплатформенності (0–1), ваги критеріїв обґрунтовані експертно з акцентом на обсяг та безпеку.

Google Drive лідирує за інтегральним показником завдяки балансу обсягу та функціоналу, що слугує орієнтиром для вимог до власної системи.

1.2.2 Інтерпретація результатів.

Розрахунки показують, що за інтегральним показником якості лідирує Google Drive, який поєднує відносно великий безкоштовний обсяг (15 ГБ), тісну інтеграцію з офісним пакетом Google Workspace, розвинуті можливості спільної роботи та високу кросплатформенність. Dropbox, попри суттєво менший обсяг безкоштовного сховища, демонструє високі значення показника за рахунок простого інтерфейсу, розвиненого версіонування та надійних механізмів шифрування за замовчуванням.

MEGA забезпечує максимальний безкоштовний обсяг (50 ГБ) та повноцінне наскрізне шифрування, однак відсутність версіонування файлів та менш розвинена інтеграція й кросплатформенність знижують його інтегральний показник у порівнянні з конкурентами. Для OneDrive та iCloud ключовою перевагою є глибока інтеграція відповідно з екосистемами Microsoft та Apple, що робить їх привабливими у випадку орієнтації на Windows або пристрої Apple, але обмежує універсальність використання поза цими екосистемами [6].

На основі представленої інформації, можна визначити, що система хмарного сховища є потужним інструментом для зберігання, обміну та управління даними. Завдяки хмарним технологіям користувачі отримують можливість доступу до своїх даних з будь-якого пристрою, що підключений до Інтернету, що сприяє підвищенню продуктивності та зручності роботи.

Проте, наявність обмежень у доступі до інформації та необхідність проведення деяких операцій через спеціалізовані інтерфейси може призвести до незручностей для користувачів. Також, питання безпеки та конфіденційності даних залишається однією з ключових проблем у використанні хмарних сховищ і було приділено увагу конкурентам, враховано їх сильні та слабкі сторони, щоб знайти золоту середину для розробки хмарного сховища, в якому є необхідність.

На підставі проведеного аналізу проведемо дослідження методів забезпечення якості зберігання даних у хмарних інформаційних системах, зокрема враховуючи потреби користувачів у зручному та безпечному доступі до даних, можливості ефективного управління та обміну інформацією, а також забезпечення високого рівня захисту та конфіденційності.

1.3 Дослідження ефективності методів забезпечення якості зберігання даних у хмарних інформаційних системах.

Розробка надійної та конкурентоспроможної інформаційної системи хмарного сховища, концепція якої була окреслена у попередніх підрозділах, вимагає глибокого аналізу технологій, що лежать в її основі. Одним із критично важливих аспектів, що визначає успіх такої системи та задоволеність користувачів, є якість і надійність зберігання даних. Забезпечення цілісності, доступності та конфіденційності інформації в розподіленому хмарному середовищі є складним завданням, яке вимагає застосування ефективних методів.

Для порівняння обрано чотири ключові підходи: реплікація, резервне копіювання, шифрування, контроль цілісності. Тестування здійснено на власнорозробленому прототипі сховища з типовим навантаженням (1000–10000 файлів середнього й великого розміру; до 50 паралельних користувачів).

Використовуються такі критерії: надійність у відсотках , час відновлення даних у секундах, вплив методу на продуктивність у відсотках, складність реалізації в балах, а також захищеність та відповідність стандартам.

Захист даних у хмарних середовищах передбачає конфіденційність, цілісність та доступності інформації, що зберігається й обробляється на віддалених серверах. Оскільки користувацькі дані розміщуються на хмарних платформах, обов'язок забезпечення їхньої безпеки поділяється між клієнтом і постачальником послуг. Як правило, механізми захисту охоплюють три ключові напрями: мережеву безпеку, шифрування даних, контроль доступу та резервне копіювання. Основним засобом гарантування безпеки виступає використання криптографічних методів під час передавання інформації (TLS/SSL) і при її зберіганні (AES, RSA). Це дозволяє запобігти несанкціонованому доступу до даних, навіть якщо злоумисник отримає фізичний чи логічний доступ до системи.

Виділяють два основні підходи до реалізації шифрування:

1. Шифрування на боці клієнта (client-side), коли дані шифруються до їх завантаження у хмару, і тільки клієнт має ключ для дешифрування;
2. Шифрування на боці сервера (server-side), коли за шифрування відповідає сам хмарний провайдер.

На рисунку 1.3 зображено спрощений приклад взаємодії клієнта з сервером[38].

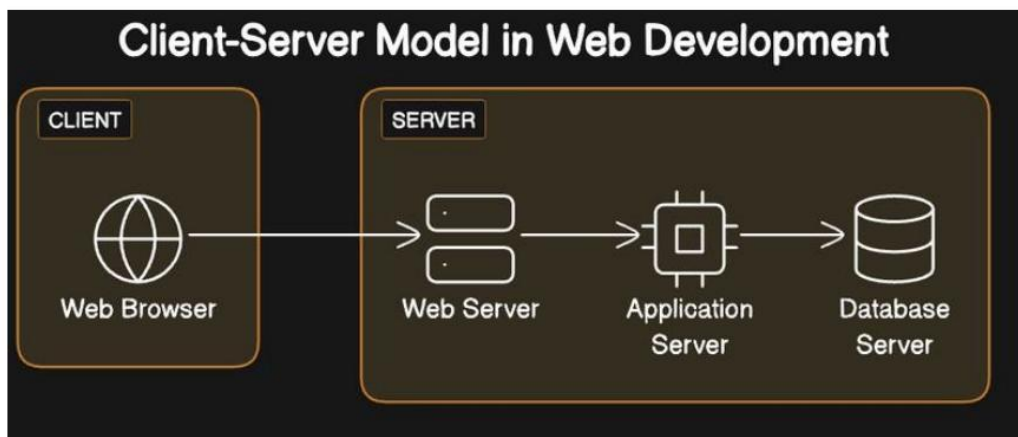


Рисунок 1.3 - Спрощений приклад взаємодії клієнта з сервером

Зашифровані дані при передачі повинні бути доступні тільки після автентифікації. Дані не вийде прочитати або зробити зміни в них, навіть у випадку доступу через ненадійні вузли. Такі технології досить відомі, алгоритми і надійні протоколи AES, TLS, IPsec давно використовуються провайдерами.

На рисунку 1.4 зображено схему взаємодії користувача, серверів управління ключами і хмарного сервера [38].

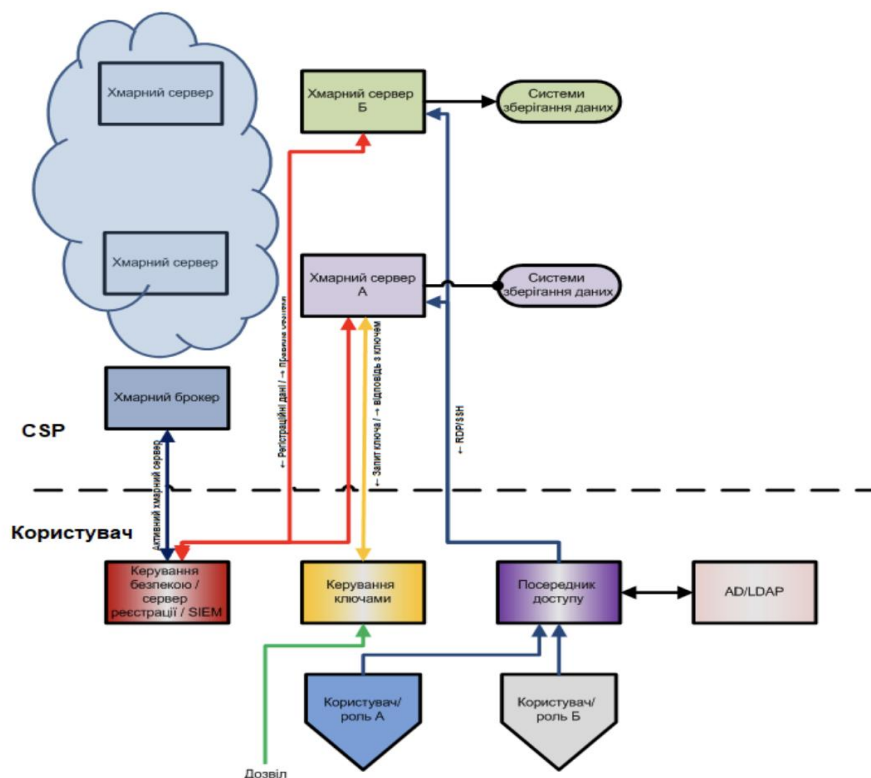


Рисунок 1.4 - Схема взаємодії користувача, серверів управління ключами і хмарного сервера

Важливою складовою є управління доступом, що охоплює механізми перевірки автентичності користувачів, такі як багатофакторна автентифікація (MFA) та токени доступу, а також контроль над правами доступу. Наприклад, рольова модель контролю доступу (RBAC) визначає, які користувачі можуть отримати дозвіл на взаємодію з певними ресурсами. Ще один ключовий елемент захисту даних у хмарі — це постійний моніторинг і ведення журналів дій користувачів і системи. Ці заходи дозволяють своєчасно виявляти й відстежувати можливі порушення безпеки, адже всі операції в хмарі фіксуються й можуть бути проаналізовані відповідними службами безпеки[36].

Таблиця 1.2 - Порівняльний аналіз методів

Метод	Надійність R, %	Час віднов. Tr	Вплив на продуктивн. P, %	Складн. (1–3)	Рівень захис.	Станд.
Реплікація	99.9	2	-3	2	середній	—
Васкуп	98	15	-5	1	високий	ISO/IEC 27017
Шифрування	—	5 (доступ)	-10	2	дуже високий	ISO/IEC 27017
Контроль цілісності	100	1	-2	1	базовий	—

Результати оцінки ключових параметрів: надійність (R), час відновлення (Tr), вплив на продуктивність (P), складність реалізації (1–3 бали), рівень захисту та відповідність стандартам. Дані отримані на основі імітаційних тестів (1000 циклів) та вимірювань часу доступу до файлу 100 МБ.

Розрахунок показників

Вплив на продуктивність (P) обчислюється за формулою:

$$P = \frac{T_{\text{захист}} - T_{\text{баз}}}{T_{\text{баз}}} \times 100\%$$

де $T_{\text{баз}} = 1.2$ с — час зчитування без захисту, $T_{\text{захист}} = 1.32$ с — з AES-шифруванням. Результат: $P \approx 10\%$.

Надійність (R) визначається як:

$$R = \frac{N_{\text{успішних}}}{N_{\text{заг}}} \times 100\%$$

Реплікація: $999/1000 = 99.9\%$; Backup: $980/1000 = 98\%$.

Таблиця 1.3 - Вплив різних методів на якість і доступність даних

Метод	Втрата даних (%)	Відновлення (%)	Час втрати/відновлення (с)	Вплив на масштабованість
Реплікація	0.1	99.9	2–3	добре масштабується
Backup	2	98	10–20	потребує додаткових ресурсів
Шифрування	0	–	+0.1–0.5 (доступ)	мінімальний вплив
Контроль цілісності	0	100	<1	легко інтегрується

Проведені розрахунки та експерименти показують, що для хмарного сервісу оптимальним є поєднання реплікації (оперативний захист), резервного копіювання (гарантія збереження у довгостроковій перспективі), шифрування

(захист від сторонніх загроз) і контролю цілісності (для виявлення та уникнення помилок при передачі).

Використання цих методів підвищує надійність та доступність даних до рівня 99,9%, знижує ризики втрат до <1%, а продуктивність системи залишається на високому рівні навіть при активованому захисті.

На підставі проведеного дослідження сформулюємо вимоги до інформаційної системи хмарного сховища.

1.4 Формування вимог до інформаційної системи.

Інформаційна система хмарного сховища для зберігання та обміну файлами розроблена з метою підвищення зручності та ефективності роботи з даними для користувачів. Основні функціональні можливості системи включають:

- Зберігання та завантаження файлів: Користувачі можуть зберігати свої файли в хмарному сховищі та завантажувати їх з будь-якого пристрою, підключеного до Інтернету;
- Доступ до файлів з будь-якого пристрою: Система надає можливість звертатися до файлів з будь-якого пристрою, що дозволяє користувачам працювати зі своїми даними навіть під час пересувань;
- Можливість ділитися файлами: Користувачі можуть легко ділитися своїми файлами з іншими користувачами, надаючи їм доступ до визначених документів або папок;
- Структурування файлів по директоріям: Система надає можливість організовувати файли у папки та підпапки для зручного управління даними;

- Авторизація за допомогою JWT токенів: Для забезпечення безпеки вхід до системи здійснюється за допомогою JWT токенів, що забезпечує високий рівень захисту даних;
- Односторінковий додаток: Додаток реалізований у вигляді односторінкового застосунку, що забезпечує зручну навігацію та швидкий доступ до всіх функцій системи;
- Пошук та сортування файлів: Користувачі можуть шукати файли за назвою або ключовими словами, а також сортувати їх за датою, розміром або форматом;
- Різноманітні варіанти відображення файлів: Система пропонує кілька варіантів відображення файлів та папок, включаючи вертикальний список з детальною інформацією про файл та горизонтальний список з іконками та назвами файлів.

Ще однією важливою вимогою є розділення додатку на дві складові частини: клієнтська сторона та серверна, та зв'язування їх за допомогою REST API.

Головні переваги такого підходу:

- Розділення відповідності: Цей підхід дозволяє чітко розділити відповідальності між клієнтом та сервером. Клієнт відповідає за відображення інтерфейсу користувача та взаємодію з ним, тоді як сервер відповідає за обробку даних та бізнес-логіку;
- Незалежність розвитку: Розділення на дві частини дозволяє розвивати клієнт та сервер незалежно один від одного. Це спрощує розробку, тестування та внесення змін у код, оскільки зміни в одній частині не впливають на іншу;
- Ефективність роботи: Використання REST API для зв'язку клієнтом та сервером дозволяє забезпечити ефективну комунікацію між цими двома

частинами. REST API є стандартом веб-сервісів, який дозволяє легко взаємодіяти з різними системами та мовами програмування;

- Масштабованість та гнучкість: REST API дозволяє розділити функціональність додатку на невеликі, самодостатні сервіси, що можуть бути масштабовані та змінювані незалежно один від одного. Це забезпечує гнучкість та адаптивність системи до змін у вимогах та обсязі даних;
- Безпека: REST API дозволяє контролювати доступ до даних та функціоналу серверу, що забезпечує високий рівень безпеки додатку.

Отже, використання REST API для зв'язку клієнта та сервера є оптимальним вибором для реалізації додатку хмарного сховища, оскільки цей підхід забезпечує чітке розділення відповідальностей, незалежність розвитку, ефективну комунікацію, масштабованість, гнучкість та безпеку системи.

Для написання клієнтської сторони додатку було обрано архітектуру “feature sliced design” а для серверної обрано архітектуру “Clean Architecture” додатку хмарного сховища, реалізованого на “Nest.js”

За допомогою такої інформаційної системи користувачі отримують зручний та надійний інструмент для зберігання, обміну та управління своїми файлами, що сприяє підвищенню продуктивності та ефективності роботи.

Вимоги до складу та параметрів технічних та програмних засобів для інформаційної системи хмарного сховища будуть наступними:

Оскільки інформаційна система реалізована у вигляді веб-додатку, вона працюватиме за принципом сесій, де клієнтський браузер відправляє запити на сервер хмарного сховища та отримує відповіді для подальшого відображення користувачу у зручному та зрозумілому форматі.

Для взаємодії з інформаційною системою хмарного сховища користувачу потрібен веб-браузер. Таким чином, до пристроїв, що можуть використовуватися

для доступу до системи, відносяться всі пристрої, на яких можливо запустити сучасний веб-браузер.

Враховуючи наведену інформацію, визначимо технічні вимоги до пристроїв користувача та адміністратора системи хмарного сховища.

Оскільки інформаційна система реалізована у формі веб-додатку, вона операційно працюватиме за принципом сесій, в яких клієнтський браузер відправляє запити на сервер хмарного сховища та отримує відповіді для подальшого відображення користувачеві у зручному форматі.

Технічні характеристики пристрою користувача будуть визначені відповідно до платформи, на якій він працює

Система повинна працювати у формі веб-додатку для зручності користувачів. Веб-додаток повинен гарантувати правильність передачі даних на сервер і бути розробленим з урахуванням всіх вимог щодо безпеки в Інтернеті. База даних має автоматично оновлюватися після внесення нової інформації. Інформаційна система повинна відповідати потребам компанії.

Враховуючи наведені технічні характеристики та вимоги до системи, розробляється концепція інформаційної системи хмарного сховища, що відповідає потребам компанії.

1.5 Розробка концепції інформаційної системи.

Концептуальна модель інформаційної системи представлена на рисунку 1.5

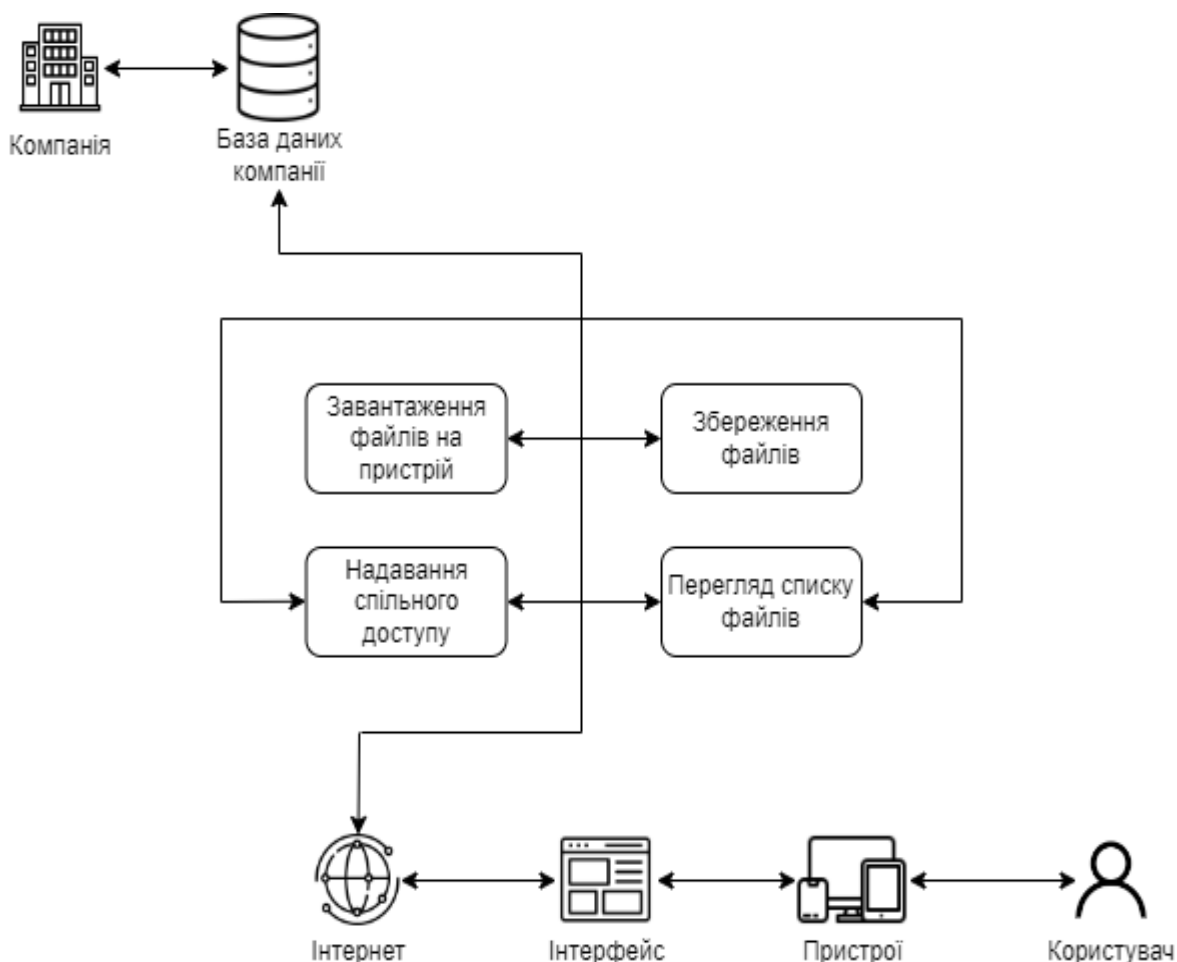


Рисунок 1.5 - Концептуальна модель інформаційної системи

Інформаційна система хмарного сховища, розроблена для зручного та безпечного зберігання та обміну файлами. При переході на веб-сайт користувач має можливість зареєструватися або авторизуватися. Якщо користувач вже має обліковий запис, він може відразу перейти до авторизації, використовуючи відповідну кнопку. Однак, якщо він увійшов у систему раніше на даному пристрої, він може користуватися додатком протягом доби без необхідності вводити логін та пароль, завдяки використанню jwt токenu.

Після успішної авторизації або реєстрації, користувач потрапляє до особистого робочого простору, де він бачить папку "Home", створену автоматично для нього. У верхній частині інтерфейсу є панель з кнопками, що надають можливість створювати нові папки, завантажувати файли з пристрою, шукати файли, сортувати їх за різними критеріями та змінювати відображення файлів.

Функція пошуку в веб-додатку хмарного сховища реалізована для забезпечення зручного та швидкого доступу користувачів до їх файлів та папок. У верхній частині інтерфейсу розташована кнопка пошуку з зображенням лупи. При натисканні на цю кнопку з'являється поле для введення запиту пошуку.

Користувач може вводити ім'я файлу або папки у це поле. Пошук автоматично здійснюється через 300 мс після того, як користувач завершить введення тексту. Це зроблено для зменшення навантаження на сервер та оптимізації роботи додатку. Такий підхід дозволяє уникнути надмірних запитів до сервера при кожному введенні символу.

Після завершення пошукового запиту система аналізує наявні файли та папки і виводить список елементів, ім'я яких співпадає з введеним текстом у полі пошуку. Якщо жодних відповідних файлів або папок не знайдено, користувач отримує повідомлення про відсутність результатів пошуку.

Важливо зазначити, що весь процес пошуку відбувається без необхідності натискання додаткових кнопок або перезавантаження сторінки. Це дозволяє зберегти час користувача та зробити процес пошуку максимально простим та зручним.

Для зручності користувача реалізовано функцію сортування файлів у веб-додатку хмарного сховища. У верхній частині інтерфейсу розташовано меню зі списком доступних варіантів сортування. Користувач має можливість обрати один з варіантів сортування файлів:

- За розміром: Файли будуть впорядковані від найменшого до найбільшого за розміром. Це дозволяє швидко знаходити найлегші або найважчі файли у списку;
- По даті створення: Файли будуть впорядковані від найновіших до найдавніших за датою створення. Цей спосіб сортування допомагає користувачу швидко зорієнтуватись у своєму архіві файлів та знаходити найсвіжіші матеріали;
- За форматом файлу: Файли будуть впорядковані за їх типом або форматом. Наприклад, всі текстові файли можуть бути окремо відображені від відео або зображення.

-

Після вибору одного з варіантів сортування файлів, система автоматично перегрупує файли відповідно до обраного критерію. Це дозволяє користувачу швидко організувати свої файли та знаходити необхідні матеріали. Реалізація цієї функції забезпечує зручність та ефективність використання веб-додатку.

Додаток підтримує функцію перетягування файлів з папки пристрою на вікно додатку для їх завантаження. Крім того, файли можна видаляти, завантажувати з додатку на пристрій, змінювати їх імена та надавати доступ іншим користувачам за допомогою посилання доступу.

Технічні вимоги пристрою користувача для використання веб додатку:

Для Windows:

- Процесор: Intel Pentium 4 або новіше з підтримкою SSE3;
- Обсяг оперативної пам'яті: не менше 512 МБ;
- Відео пам'ять: не менше 64 МБ;
- Вільне місце на жорсткому диску: не менше 350 МБ;
- Роздільна здатність екрану: не менше 800 × 600;
- Клавіатура та миша.

Для Linux:

Технічні характеристики аналогічні вимогам для Windows.

Для MacOS:

- Протестовано на OS X El Capitan (10.11) та вище.

Для Android:

- Підтримка Android 5.0 (Lollipop) та вище.

Для iOS:

- Підтримка iOS 12.2 (iPadOS 12.2) та вище.

Додаток розділено на дві складові частини: клієнт та backend(сервер), і зв'язані вони за допомогою REST API. Цей підхід дозволяє нам забезпечити ефективну комунікацію між частинами додатку, а також незалежність розвитку та гнучкість системи.

Веб-додаток використовує архітектурний підхід “feature sliced design” на стороні клієнта, що дозволяє розділити функціонал на невеликі, самодостатні модулі. Це забезпечує зручне керування та розвиток додатку.

“Feature sliced design” - це архітектурний підхід, що дозволяє розбити веб-додаток на невеликі функціональні модулі або фічі, які можуть бути розроблені, тестовані та розгорнуті окремо. Основні переваги цієї архітектури для веб-додатку хмарного сховища такі:

- Модульність: “Feature sliced design” дозволяє розділити додаток на невеликі, самодостатні модулі, що спрощує розробку та підтримку коду. Кожен модуль може бути розроблений незалежно, що дозволяє розробнику працювати над окремими функціями без необхідності втручання в інші частини додатку;
- Масштабованість: Ця архітектура дозволяє легко масштабувати додаток за рахунок додавання нових функцій або модулів. Кожен новий функціонал

може бути доданий як окремий модуль, що спрощує розширення додатку без необхідності переписувати або модифікувати існуючий код;

- Тестування: Розділення додатку на невеликі модулі спрощує процес тестування. Кожен модуль може бути протестований окремо, що дозволяє виявляти та виправляти помилки швидше та ефективніше;
- Швидкість розробки: “Feature sliced design” сприяє прискоренню розробки, оскільки розробники можуть працювати паралельно над різними функціями. Це дозволяє скоротити час розробки нового функціоналу та випуску оновлень;
- Гнучкість: Кожен модуль може бути розгорнутий та масштабований незалежно, що дозволяє реагувати на зміни вимог без значного впливу на інші частини системи. Це забезпечує гнучкість та адаптивність системи до змін у потребах користувачів та бізнесу.

Для серверної частини було обрано архітектуру: “Clean Architecture”. Ця архітектура вважається оптимальним вибором з наступних причин:

- Розділення відповідальностей: “Clean Architecture” дозволяє чітко розділити бізнес-логіку, презентаційні та інфраструктурні шари. Це допомагає забезпечити чистоту коду, легку зміну та тестування окремих компонентів системи;
- Незалежність від фреймворків та бібліотек: “Clean Architecture” створює ізольовані шари, які не залежать від конкретних фреймворків чи бібліотек. Це робить код більш переносним та можливим для майбутнього рефакторингу чи заміни технологій;
- Тестовість: Чиста архітектура дозволяє легко писати модульні та інтеграційні тести для кожного окремого шару. Це сприяє покращенню якості коду та швидкому виявленню помилок;
- Масштабованість та гнучкість: “Clean Architecture” дозволяє легко масштабувати додаток, додаючи нові функції чи модулі без необхідності

змінювати вже існуючий код. Це забезпечує гнучкість та адаптивність системи до змін у вимогах.

Отже, веб-додаток хмарного сховища не лише забезпечує зручне та безпечне зберігання файлів, але й використовує сучасні архітектурні підходи та сервіси для оптимізації його роботи та зручності користування.

Для реалізації інформаційної системи хмарного сховища використано наступні інструменти:

- Мова гіпертекстової розмітки – HTML;
- Мова каскадних таблиць стилів – CSS;
- Мова програмування – TypeScript;
- Бібліотека JavaScript – React;
- Середовище виконання JavaScript – Node.js
- Бібліотек JavaScript для шифрування даних – bcrypt.js
- Фреймворк JavaScript – Next.js;
- Фреймворк JavaScript – Nest.js;
- HTTP-клієнт – Axios;
- ORM (Об'єктно-реляційне відображення) – Sequelize;
- Система управління базами даних – MariaDB;
- Декларативна мова програмування – SQL;
- Система контролю версій - Git;
- Git хостинг – GitHub;

Для створення диплому використані такі інструменти:

- Програмне забезпечення для створення діаграм – draw.io;
- Програмне забезпечення для роботи з текстом – Word;
- Програмне забезпечення для роботи з растровою графікою – Krita.

Технічні засоби використані відповідно до вимог до інформаційної системи.

На основі дослідження сформована концепція інформаційної системи хмарного сховища у подальшій частині роботи здійснюється розробка технічного проекту системи.

Розділ 2 Розробка технічного проекту інформаційної системи

2.1 Загальносистемні рішення.

Існує різноманіття архітектурних підходів для розробки клієнтської частини програмного забезпечення з використанням сучасних фреймворків, таких як React. Одним із таких підходів є "feature sliced design" (FSD), який дозволяє розбити програму на логічні модулі або функціональні фічі. Додатковою архітектурою, яка також варта уваги, є "flux architecture" та "component-based architecture". Обрання FSD для фронтенд додатку на React обґрунтоване його здатністю забезпечити чітку структуру та організацію коду. Він дозволяє легко масштабувати програму, зберігаючи при цьому чистоту та гнучкість розробки. Крім того, FSD сприяє зручній співпраці між командами розробників, оскільки розділення програми на невеликі функціональні блоки дозволяє розробникам працювати над різними частинами незалежно один від одного.

FSD - це методологія проєктування фронтенд-додатків, яка базується на правилах та угодах щодо організації коду. Основна мета цієї методології полягає в тому, щоб зробити код зрозумілим та структурованим, особливо в умовах постійних змін вимог бізнесу. Однією з переваг FSD є чітка структура коду, яка базується на шарах, слайсах та сегментах.

Шари - це перший рівень організаційної ієрархії в FSD, які розділяють код в залежності від його відповідальності та взаємозв'язків з іншими модулями.

Кожен шар має свої власні семантичні значення, які допомагають визначити, яку відповідальність нести модулю у коді. Назви та значення шарів є стандартизованими для всіх проєктів, побудованих з використанням FSD.

Слайси - це другий рівень в організаційній ієрархії FSD, які групують код за його значенням для продукту, бізнесу або застосунку.

Структура проєкту з використанням "Feature sliced design" зображена на рисунку 2.1, який взято з матеріалу [14].

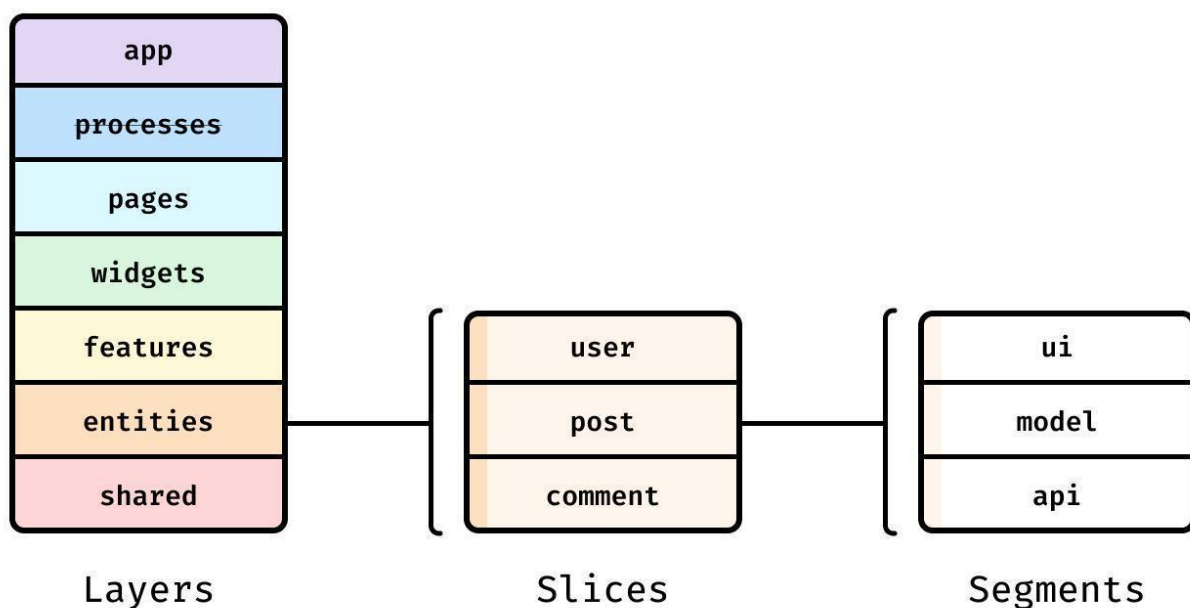


Рисунок 2.1 – Структура проекту з використанням “Feature sliced design”

Опис кожного шару:

Шари стандартизовані для всіх проектів і розташовані вертикально. Модулі на одному шарі можуть взаємодіяти лише з модулями з шарів строго нижче. Зараз існують сім шарів (від нижнього до верхнього):

shared — повторно використовувана функціональність, відокремлена від конкретики проекту/бізнесу. (наприклад, UIKit, бібліотеки, API)

entities — бізнес-сутності. (наприклад, Користувач, Товар, Замовлення)

features — взаємодії з користувачем, дії, що надають бізнес-значення користувачеві.

widgets — композиційний шар для поєднання сутностей та функцій в значущі блоки.

pages — композиційний шар для створення повних сторінок з сутностей, функцій та віджетів.

processes (застарілий) — складні сценарії між сторінками.

app — налаштування програми, стилі та постачальники на рівні програми.

Далі є слайси, які розділяють код за бізнес-доменами. Це дозволяє легко орієнтуватися в коді, тримаючи логічно пов'язані модулі поруч.

Назви слайсів не мають стандартизованого формату, оскільки вони прямо відображають потреби додатка в області бізнесу. Наприклад, для фотогалереї можуть бути використані слайси `photo`, `create-album`, `gallery-page`. У випадку соціальної мережі необхідні інші сегменти, такі як `post`, `add-user-to-friends`, `news-feed`.

Слайси, які мають схожий зміст, можуть бути організовані структурно в одну папку, але всі вони повинні дотримуватися тих самих правил ізоляції, що й інші сегменти. У цій директорії не повинно бути жодного коду, який може використовуватися декількома слайсами одночасно. Слайси не можуть використовувати інші сегменти на тому ж рівні, що допомагає з високою співпрацею та низькою зв'язаністю.

Кожен слайс, з свого боку, складається з сегментів. Це невеликі модулі, призначені для розділення коду в межах сегмента за його технічним призначенням.

Отже, “feature sliced design” є оптимальним вибором для веб-додатку хмарного сховища, оскільки вона сприяє швидкій розробці, масштабованості, тестуванню та забезпечує гнучкість системи. Інформацію про “feature sliced design” підготовано на основі матеріалу [14].

Серед архітектурних підходів для розробки серверної частини програмного забезпечення, вирішено використовувати “Clean Architecture”. Крім того, варто розглянути такі архітектурні підходи, як “Microservices Architecture” та “Model-View-Controller (MVC) Architecture”. Вибір “Clean Architecture” для бекенду на Nest.js обумовлений його здатністю забезпечити чистоту та модульність коду, а також легкість розширення і тестування системи. Цей підхід дозволяє чітко визначити рівні абстракції та розділити бізнес-логіку від технічних деталей реалізації. Крім того, “Clean Architecture” сприяє збереженню високого рівня незалежності між компонентами системи, що дозволяє змінювати

один шар без впливу на інші, тим самим полегшуючи підтримку та розвиток додатку.

Clean Architecture це архітектурний підхід, запропонований Робертом К. Мартіном (дядьком Бобом), що ґрунтується на багатьох архітектурних настановах, таких як “Hexagonal Architecture”, “Onion Architecture”.

Структура проекту з використанням “Clean Architecture” зображена на рисунку 2.2, який було взято з матеріалу [16, 17].

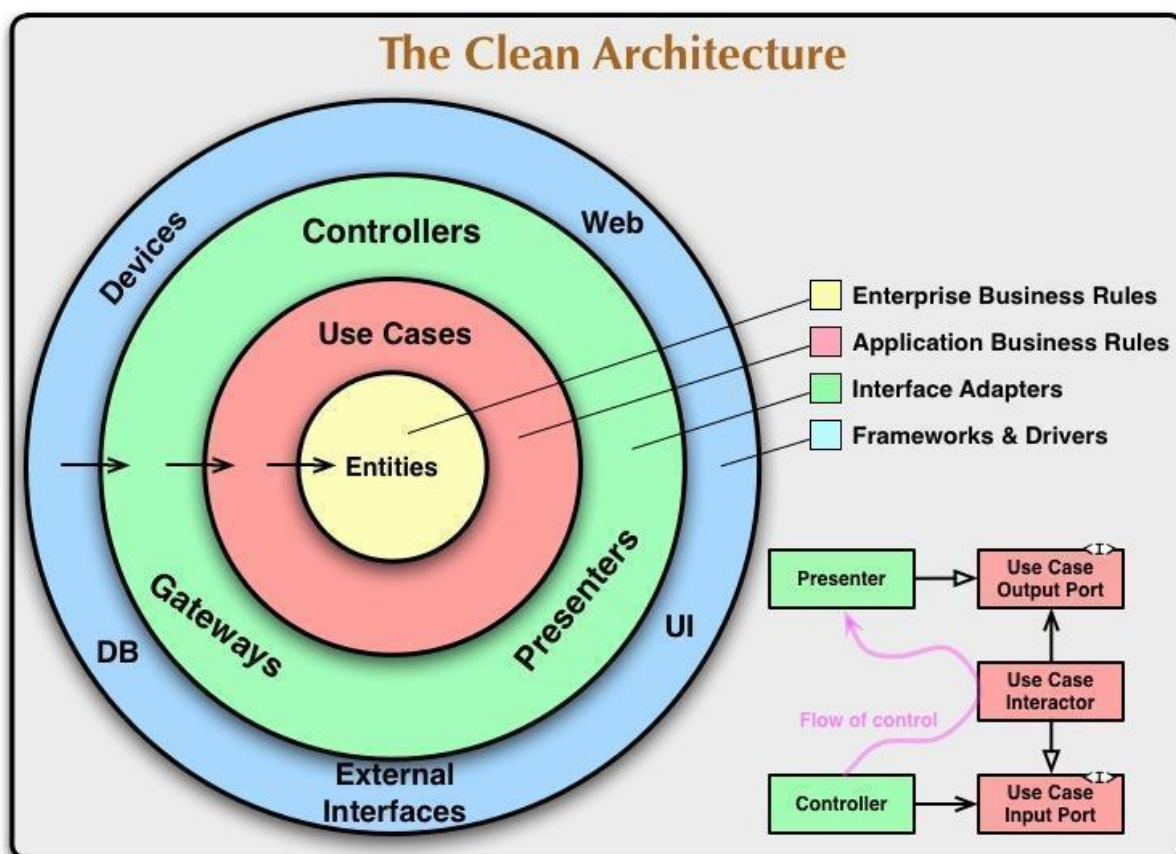


Рисунок 2.2 - Структура проекту з використанням Clean Architecture

Ми бачимо, що на діаграмі є чотири шари. Синій шар, зелений шар, червоний шар і жовтий шар.

Кожне коло представляє різні області програмного забезпечення. Зовнішній шар - це найнижчий рівень програмного забезпечення, і чим глибше

ми просуваємось, тим рівень буде вищим. Загалом, чим глибше ми просуваємось, тим менш схильний до змін шар.

Опишемо кожен шар архітектури. Entities (Сутності) - Сутності інкапсулюють бізнес-правила для всього підприємства. Сутність може бути об'єктом з методами або набором структур даних і функцій. Це не має значення, якщо сутності можуть використовуватися багатьма різними додатками на підприємстві.

Якщо у вас немає підприємства, і ви просто пишете один додаток, то ці сутності є бізнес-об'єктами додатку. Вони інкапсулюють найбільш загальні та високорівневі правила. Вони найменш схильні до змін, коли щось змінюється ззовні. Наприклад, ви не очікуєте, що на ці об'єкти вплине зміна навігації по сторінці або безпеки. Жодні операційні зміни в конкретному додатку не повинні впливати на рівень сутностей.

Use Cases (Варіанти використання) - Програмне забезпечення цього рівня містить бізнес-правила для конкретних додатків. Воно інкапсулює та реалізує всі варіанти використання системи. Ці варіанти використання організовують потік даних до та з підрозділів і спрямовують ці підрозділи на використання їхніх бізнес-правил для досягнення цілей варіанту використання.

Не очікується, що зміни на цьому рівні вплинуть на сутності. Також не очікується, що на цей рівень вплинуть зміни у зовнішніх середовищах, таких як база даних, інтерфейс користувача або будь-який із загальних фреймворків. Цей рівень ізолюваний від таких проблем.

Однак, очікується, що зміни в роботі програми вплинуть на варіанти використання, а значить, і на програмне забезпечення цього рівня. Якщо змінюються деталі варіанту використання, то це неодмінно вплине на деякий код на цьому рівні.

Interface Adapters (Інтерфейсні адаптери) - Програмне забезпечення цього рівня являє собою набір адаптерів, які перетворюють дані з формату, найбільш зручного для використання в конкретних випадках та сутностях, у формат,

найбільш зручний для зовнішніх агентів, таких як база даних або веб. Саме цей рівень, наприклад, буде повністю містити архітектуру MVC графічного інтерфейсу. Презентатори, представлення та контролери належать до цього шару. Моделі, швидше за все, є просто структурами даних, які передаються від контролерів до варіантів використання, а потім назад від варіантів використання до презентерів і представлень.

Аналогічно, на цьому рівні дані перетворюються з форми, найбільш зручної для сутностей і варіантів використання, у форму, найбільш зручну для будь-якого фреймворку персистентності, що використовується, тобто бази даних. Жоден код всередині цього кола не повинен знати нічого про базу даних. Якщо база даних є базою даних SQL, то весь SQL повинен бути обмежений цим шаром, і зокрема тими частинами цього шару, які мають відношення до бази даних.

Також на цьому рівні знаходиться будь-який інший адаптер, необхідний для перетворення даних з деякої зовнішньої форми, наприклад, зовнішнього сервісу, у внутрішню форму, що використовується варіантами використання та сутностями.

Frameworks and Drivers (Фреймворки та рушійні сили) - Зовнішній шар зазвичай складається з фреймворків та інструментів, таких як база даних, веб-фреймворк тощо. Зазвичай ви не пишете багато коду в цьому шарі, окрім коду-зв'язки, який зв'язує наступне коло всередину.

На цьому рівні знаходяться всі деталі. Веб - це деталь. База даних - це деталь. Ми тримаємо ці речі зовні, де вони не можуть завдати великої шкоди.

Інформацію про кожен шар архітектури взято з матеріалу [17].

Інші варіанти архітектур були відкинуті з таких причин:

“Monolithic Architecture”: хоча цей підхід є простим в розробці, вона не забезпечує достатньої гнучкості та масштабованості для великих проектів, таких як додаток хмарного сховища.

“Service-Oriented Architecture” (SOA): ця архітектура може бути складною для розгортання та управління, і не завжди є оптимальним вибором для невеликих проектів. Крім того, вона може створювати зайвий наклад у випадку простого додатку, як хмарне сховище.

Обрана “Clean Architecture” є найкращим вибором для додатку хмарного сховища з урахуванням її простоти, тестовості, масштабованості та гнучкості.

Згідно з написаним вище, додаток розроблюється з використанням архітектур “feature sliced design” та “clean architecture”.

Інформацію про “Clean Architecture” підготовано на основі матеріалів [15, 16, 17].

Для аутентифікації використовується JWT токен. Аутентифікація з використанням JWT

JWT (JSON Web Token) - це стандарт відкритих токенів на основі JSON для передачі даних аутентифікації між двома сторонами. Використання JWT для аутентифікації надає безпечний і ефективний спосіб перевірки ідентичності користувача.

Структуру JWT токена зображено на рисунку 2.3, рисунок взято з [18].

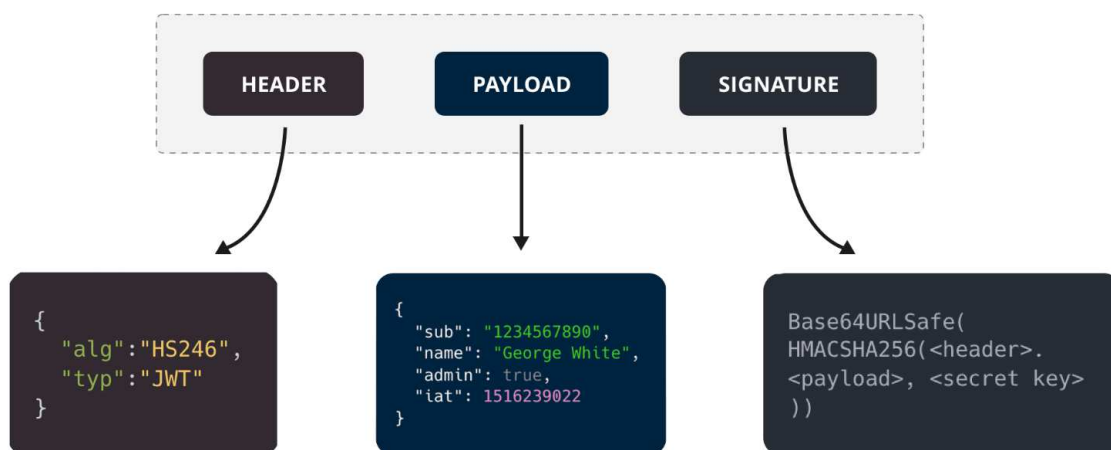


Рисунок 2.3 – структура JWT токена

JWT складається з трьох частин: заголовка(header), корисного навантаження(payload) та підпису(signature).

Заголовок JWT містить два параметри: тип токена і алгоритм підпису. Зазвичай це виглядає як JSON-об'єкт, який вказує на те, що це JWT і який алгоритм шифрування використовується.

У нашому веб-додатку для хмарного сховища ми використовуємо JWT для забезпечення безпеки та автентифікації користувачів. Після успішної автентифікації користувача, сервер створює JWT, який містить у собі інформацію про користувача та строк його дії. Клієнтська сторона зберігає цей токен, і використовує його при кожному запиті на сервер для підтвердження ідентичності.

JWT - це потужний механізм для забезпечення автентифікації та безпеки у веб-додатках. Використання JWT у нашому проєкті дозволяє нам забезпечити захист даних користувачів та ефективну систему автентифікації.

Базуючись на прийнятих вище рішеннях, розроблюється додаток хмарного сховища.

2.2 Рішення по інформаційному забезпеченню.

Для забезпечення роботи інформаційної системи необхідна база даних. У даному проєкті для цих цілей використовується СУБД MariaDB.

MariaDB — це вільна система керування реляційними базами даних, яка є розвиненою версією MySQL. MariaDB виникла як відповідь на невизначену перспективу MySQL після його придбання Oracle, що спричинило невпевненість серед спільноти розробників і користувачів.

Враховуючи обрану СУБД розроблюється концептуальна модель даних.

Інформацію про MariaDB підготовано на основі матеріалу [22].

2.2.1 Розробка концептуальної моделі даних.

Концептуальна модель даних - це абстрактна модель, яка визначає структуру модельованої системи, властивості її елементів та причинно-наслідкові зв'язки, що є суттєвими для досягнення мети моделювання[25]. Вона включає в себе:

- Формулювання змістовного і внутрішнього представлення, яке поєднує концепцію користувача та розробника моделі. Це включає в себе логіку, алгоритми, припущення та обмеження;
- Абстрактний опис об'єкта моделювання, який відображає концепцію дослідника щодо проблеми. Ця модель виявляє причинно-наслідкові зв'язки, властиві досліджуваному об'єкту в межах, визначених цілями дослідження.

Концептуальна модель даних виступає важливим засобом під час створення баз даних та інформаційних систем. Вона сприяє уточненню вимог, ідентифікації сутностей, їх властивостей та взаємозв'язків. Цей етап є першим у процесі розробки бази даних і створює фундамент для подальшого розроблення логічних та фізичних моделей даних.

Концептуальна модель даних зображена на рисунку 2.4

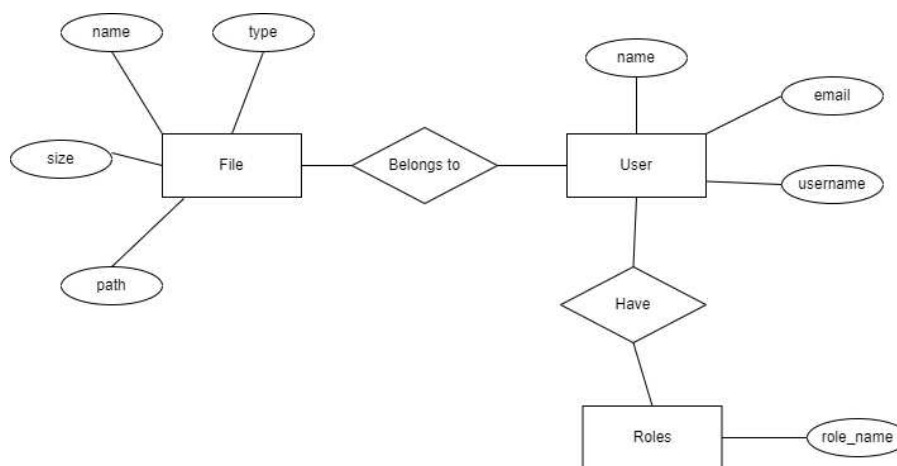


Рисунок 2.4 – Концептуальна модель даних

З метою реалізації потреб та функціоналу інформаційної системи хмарного сховища, була створена концептуальна модель бази даних. Ця модель враховує особливості зберігання, доступу та обробки даних в хмарному середовищі, відповідає вимогам безпеки та ефективності, та служить основою для подальшого проектування та реалізації логічної та фізичної моделей даних. Розроблену концептуальну модель використана для розробки логічної моделі даних.

2.2.2 Розробка логічної моделі даних.

Логічна (або даталогічна) модель бази даних є абстрактною представою про структуру даних і взаємозв'язки між ними, незалежну від конкретної системи управління базами даних. Вона визначає основні сутності та їх взаємозв'язки, а також може включати в себе визначення типів даних для окремих сутностей. Побудована така модель зазвичай за допомогою методу Сутність-зв'язок (Entity Relationship).

Логічну модель бази даних представлено на рисунку 2.5.

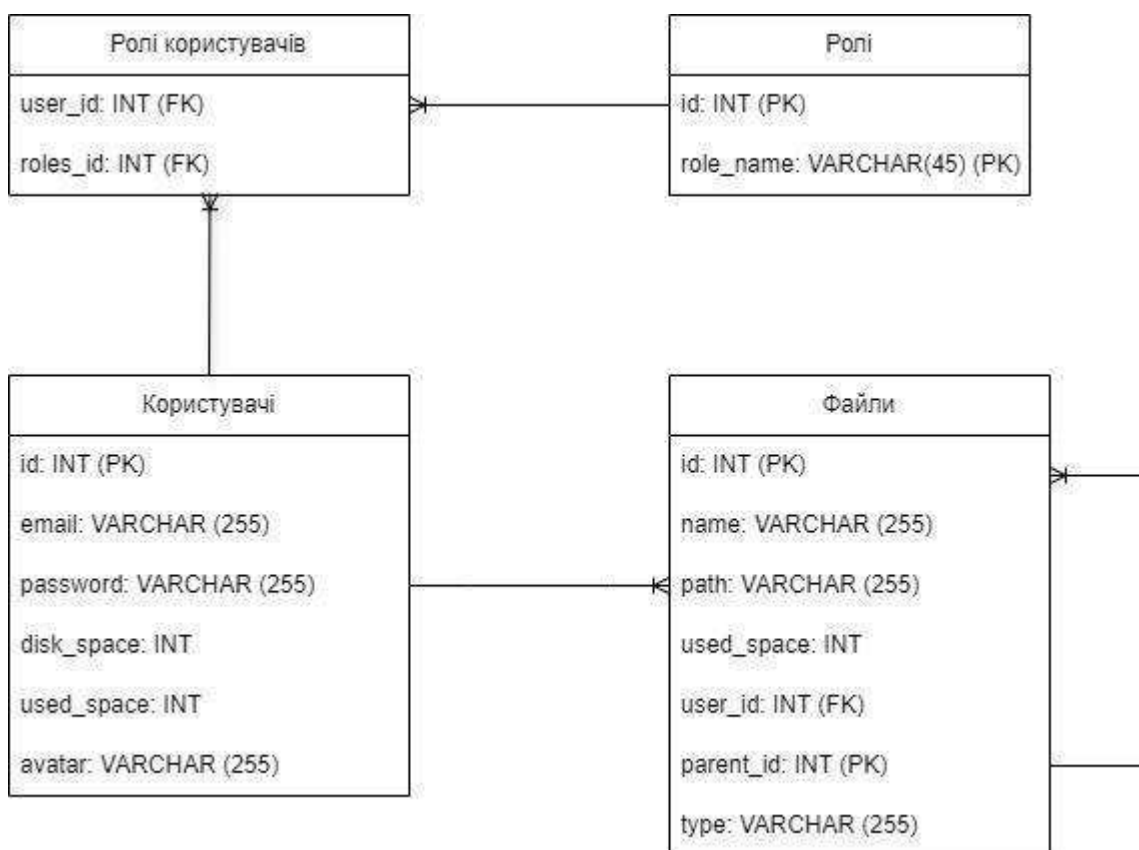


Рисунок 2.5 – Логічна модель бази даних

Аналіз на нормальні форми (NF) допомагає перевірити ефективність та коректність структури бази даних. Нормальні форми визначають, наскільки дані в БД організовані відповідно до стандартів нормалізації. Вище вказана структура бази даних складається з таких таблиць: ‘Користувачі’, ‘files’, ‘roles’, і ‘user_roles’.

Таблиця “Користувачі”:

Перша нормальна форма (1NF): усі атрибути є атомарними, тобто не розкладаються на більш прості елементи. Таблиця ‘Користувачі’ у 1NF, оскільки кожен стовпчик містить атомарні значення.

Друга нормальна форма (2NF): у 2NF кожний стовпчик пов'язаний із цілою первинною ключовою колонкою. Таблиця “Користувачі” також у 2NF, оскільки вона має однозначний первинний ключ (“id”).

Третя нормальна форма (3NF): у 3NF немає транзитивних залежностей між неключовими стовпчиками. Таблиця “Користувачі” у 3НФ, оскільки немає транзитивних залежностей.

Таблиця “files”:

1NF: кожен стовпчик містить атомарні значення.

2NF: у таблиці є складний первинний ключ (‘id’ і ‘User_id’), проте кожен стовпчик пов'язаний з цілою первинною ключовою колонкою.

3NF: немає транзитивних залежностей.

Таблиця ‘roles’:

1NF: відповідає, оскільки кожен стовпчик містить атомарне значення.

2NF: єдиний потенційний кандидат на первинний ключ (‘id’).

3NF: таблиця ‘roles’ в 3НФ, оскільки немає транзитивних залежностей.

Таблиця ‘user_roles’:

1NF: атомарні значення в кожному стовпчику.

2NF: є складний первинний ключ, але кожен стовпчик пов'язаний з цілою первинною ключовою колонкою.

3NF: немає транзитивних залежностей.

Загалом структура бази даних відповідає нормальним формам, що говорить про її ефективність та коректність з точки зору нормалізації даних. Логічну модель використано для розробки фізичної моделі.

2.2.3 Розробка фізичної моделі даних.

Фізична модель даних, або проектування бази даних, представляє собою опис дизайну даних у контексті конкретної системи управління базами даних або для її реалізації. Цей етап у процесі проектування бази даних зазвичай базується на логічній моделі даних, хоча може бути зворотно розроблений з наявної реалізації бази даних. У завершеній фізичній моделі даних будуть враховані всі аспекти бази даних, необхідні для створення зв'язків між таблицями або досягнення високої продуктивності, таких як індекси, обмеження, зв'язані або секціоновані таблиці, або кластери.

Фізичну модель представлено на рисунку 2.6.

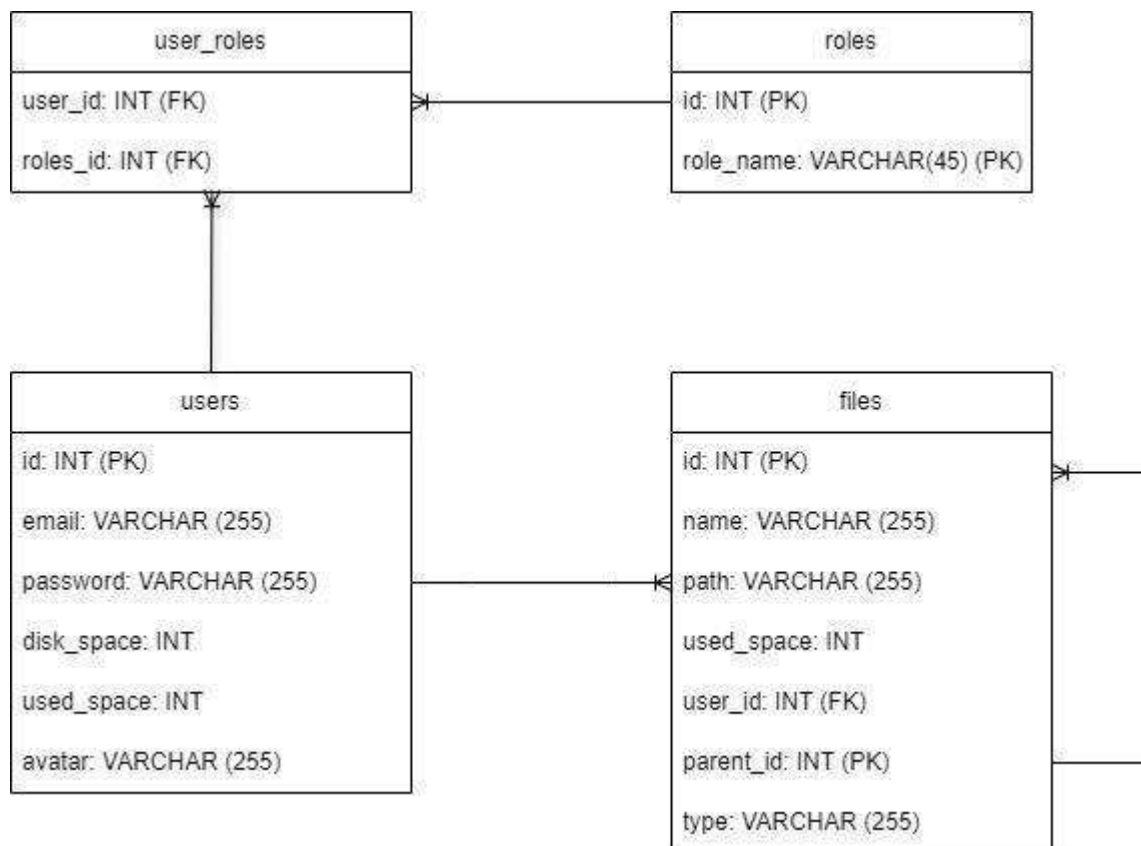


Рисунок 2.6 – Фізична модель даних

Таблиця «users»:

- id: Унікальний ідентифікатор користувача.
- email: Електронна пошта користувача, яка є унікальною та використовується для ідентифікації;
- password: Поле для зберігання хешованого паролю користувача.
- diskSpace: Кількість дискового простору, який доступний користувачеві.
- usedSpace: Кількість використаного дискового простору користувачем.
- avatar: Шлях до аватару користувача.

Таблиця «files»:

- id: Унікальний ідентифікатор файлу.
- name: Назва файлу.
- path: Шлях до файлу в системі.
- user_id: Зовнішній ключ, який пов'язує файл з конкретним користувачем (з таблиці `users`).

- size: Розмір файлу в байтах.
- parent_id: Зовнішній ключ, який вказує на батьківську папку цього файлу (на файл в таблиці «files»).
- type: Тип файлу (“dir” або “file”).

Таблиця «roles»:

- id: Унікальний ідентифікатор ролі.
- role_name: Назва ролі.

Таблиця «user_roles»:

- User_id: Зовнішній ключ, який пов'язує користувача з його ролями (з таблиці «users»).
- roles_id: Зовнішній ключ, який вказує на конкретну роль користувача (з таблиці «roles»).
- Таблиця «users» містить основну інформацію про користувачів, таку як електронна пошта, пароль, обсяг дискового простору та аватар.
- Таблиця «files» відображає інформацію про файли та папки, зокрема їхні розміри, шляхи та типи.
- Таблиця «roles» містить назви ролей, які можуть бути присвоєні користувачам.
- Таблиця «user_roles» встановлює зв'язок між користувачами та їхніми ролями.

Ця логічна модель дозволяє ефективно взаємодіяти з інформацією про користувачів, їхні файли та ролі у системі.

Таким чином, розроблено інформаційне забезпечення системи, включаючи концептуальну, логічну та фізичну моделі даних. Для повного завершення технічного проекту, наступний підрозділ присвячений опису ключових алгоритмів та методів, що забезпечують функціональність системи хмарного сховища.

2.3 Рішення з математичного забезпечення

Блок-схему загального алгоритму роботи додатку зображено на рисунку 2.7.

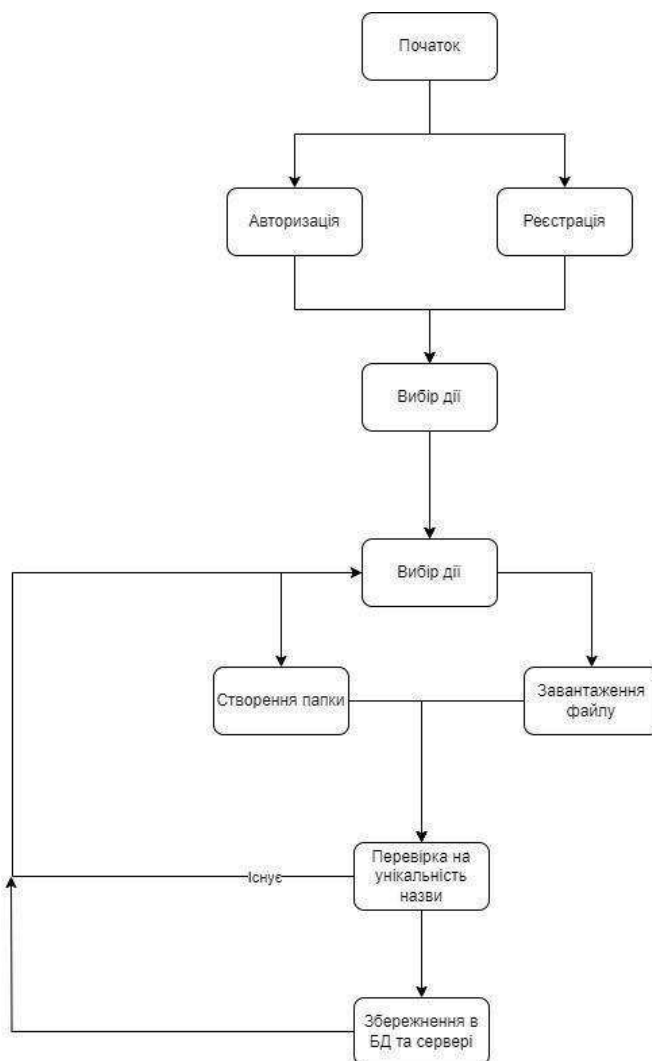


Рисунок 2.7 - Блок-схема загального алгоритму роботи додатку

Під час роботи з веб-додатком спершу відбувається процес авторизації або аутентифікації користувача за адресою додатку. Після успішної ідентифікації користувача відкривається можливість вибору дії, а саме: створення файлу чи папки. Після вибору дії, відповідний запит надсилається на сервер для подальшої обробки. На сервері здійснюється перевірка унікальності імені файлу чи папки в конкретній директорії. У випадку, якщо ім'я не є унікальним, сервер повертає відповідне повідомлення на клієнт. У разі ж унікальності імені, відбувається

процес збереження файлу в базі даних та на сервері, що підтверджує успішне виконання операції.

2.4 Рішення з програмного забезпечення.

Частина системи, що пов'язана з інформаційним забезпеченням, тобто база даних, розроблена на основі СУБД MariaDB та з використанням декларативної мови програмування SQL .

MariaDB - це система керування базами даних (СКБД), що походить від MySQL [22]. Вона зберегла сумісність з MySQL, але має ряд додаткових функцій та покращень. MariaDB використовується в широкому спектрі застосувань, від веб-сайтів та веб-додатків до великих корпоративних систем.

MariaDB Server - один з найпопулярніших серверів баз даних у світі. Він створений оригінальними розробниками MySQL і гарантовано залишається з відкритим вихідним кодом. Серед відомих користувачів - Wikipedia, WordPress.com та Google.

MariaDB Server перетворює дані на структуровану інформацію в широкому спектрі додатків, від банківської справи до веб-сайтів. Спочатку розроблений як вдосконалена заміна MySQL, MariaDB Server використовується тому, що він швидкий, масштабований і надійний, з багатою екосистемою механізмів зберігання даних, плагінів і багатьох інших інструментів, що робить його дуже універсальним для широкого спектру випадків використання.

MariaDB Server розробляється як програмне забезпечення з відкритим вихідним кодом і, як реляційна база даних, надає SQL-інтерфейс для доступу до даних. Останні версії MariaDB Server також включають функції ГІС та JSON.

Проект MariaDB ставив перед собою завдання високої сумісності з MySQL, зокрема, забезпечивши можливість заміщення функцій бінарним еквівалентом бібліотек і зберігаючи сумісність з API і командами MySQL. Навіть перша стабільна версія MariaDB була названа 5.2, що продовжувало послідовність версій MySQL.

MariaDB включає два нові рушії баз даних: Aria (раніше відомий як Maria), що заміщує MyISAM, та XtraDB, який стає альтернативою InnoDB (включно до версії 10.1). Серія MariaDB 5.2.x була повністю сумісна з MySQL 5.1.x і базувалася на тій же кодовій базі, що спрощувало міграцію з MySQL на MariaDB.

Незважаючи на збереження зворотної сумісності, MariaDB внесла певні зміни, включаючи підтримку декількох нових сховищ (FederatedX, PBXT, XtraDB, Aria, OQGRAPH), реалізацію додаткових оптимізацій, додавання нових можливостей (наприклад, віртуальні стовпці та Plug-in аутентифікації) і інтеграцію патчів, розроблених компанією Percona і спільнотою незалежних розробників. Вона була розроблена для забезпечення високої швидкодії обробки великих обсягів даних та має відкритий вихідний код. MariaDB поступово стала однією з найпопулярніших СКБД, зокрема завдяки своїй сумісності з MySQL та розширеним можливостям.

Вона активно використовується для розробки динамічних веб-сторінок, підтримуючи різноманітні мови програмування та забезпечуючи надійність та продуктивність роботи з базами даних. MariaDB є надійним рішенням для проєктів будь-якої складності, що потребують ефективного управління та збереження даних.

MariaDB має ряд функцій, які роблять її потужною та гнучкою СКБД. До них відносяться:

- Сумісність з MySQL: MariaDB повністю сумісна з MySQL, що робить її легкою заміною для MySQL;
- Підвищена масштабованість: MariaDB може обробляти більше даних та більші навантаження, ніж MySQL;
- Покращена продуктивність: MariaDB пропонує кращу продуктивність, ніж MySQL, за рахунок ряду оптимізацій;
- Розширена функціональність: MariaDB включає ряд додаткових функцій, які не доступні в MySQL, таких як реплікація Galera та віртуальні стовпці.

Інформацію про MariaDB підготовлено на основі матеріалу [22].

У проєкті використовувалась реляційна база даних MariaDB для зберігання та організації даних. Вона використовується для зберігання інформації про користувачів, їх файли, ролі та відносини між ними. MariaDB дозволяє ефективно оптимізувати запити до бази даних та забезпечує надійну та швидку роботу системи в цілому.

Для контролю версій та спільної роботи над кодом використовується система Git - це розподілена система керування версіями, яка дозволяє розробникам ефективно працювати з кодом та відстежувати його зміни.

Принцип роботи Git:

Git зберігає повний репозиторій проєкту на кожному комп'ютері, що дозволяє розробникам працювати незалежно, навіть офлайн. Кожне змінене становище файлів фіксується в коміті, що дозволяє повернутися до будь-якого попереднього стану проєкту. Зміни в коді можуть бути відстежені за допомогою гілок, що дозволяє розробникам працювати паралельно та об'єднувати зміни [23].

GIT використовувався для керування версіями програмного коду проєкту. Кожен розробник міг клонувати репозиторій проєкту, створювати та переходити між гілками для внесення змін, а також об'єднувати різні гілки для збереження та розподілу коду. GIT дозволив ефективно керувати розробкою проєкту, забезпечуючи можливість відслідковувати зміни, вносити правки та забезпечувати спільну роботу розробників над кодом.

Для створення клієнтської сторони додатку було використано мову гіпертекстової розмітки HTML5, мову каскадних таблиць стилів CSS3, мову програмування TypeScript, бібліотеку для HTTP запитів Axios, збірник проєкту Vite та JavaScript бібліотеку React.

HTML5 (Hypertext Markup Language 5) - це мова розмітки, яка використовується для структурування та відображення гіпертекстових документів у всесвітній мережі Інтернет. Це була п'ята і остання основна версія HTML, яка тепер є застарілою рекомендацією Консорціуму Всесвітньої павутини (W3C). Поточна специфікація відома як HTML Living Standard і підтримується Робочою групою з веб-технологій гіпертекстових застосунків

(WHATWG), консорціумом провідних виробників браузерів (Apple, Google, Mozilla та Microsoft).

HTML5 включає детальні моделі обробки, щоб сприяти більш сумісним імплементаціям; він розширює, поліпшує та раціоналізує розмітку, доступну для документів, і вводить розмітку та програмні інтерфейси застосунків (API) для складних веб-застосунків. З тих же причин HTML5 є також кандидатом для мобільних застосунків платформи, оскільки він включає функції, призначені для пристроїв з низькою потужністю.

Інформацію про HTML5 підготовано на основі матеріалу [27].

HTML використовувався для створення структури та маркування контенту веб-додатку. За допомогою HTML було створено різноманітні веб-сторінки, включаючи сторінку авторизації, головну сторінку з можливістю завантаження файлів та інші. HTML дозволив визначити структуру документів та їхній зовнішній вигляд, що забезпечило коректне відображення вмісту на різних пристроях та браузерах.

Каскадні таблиці стилів (CSS) - це мова таблиць стилів, яка використовується для визначення оформлення і стилізації документа, написаного на мові розмітки, такі як HTML або XML (включаючи діалекти XML, такі як SVG, MathML або XHTML). CSS є ключовою технологією всесвітньої павутини, разом з HTML і JavaScript.

CSS використовувався для стилізації і візуального оформлення веб-сторінок в проекті хмарного сховища. За допомогою CSS було визначено кольори, шрифти, розміри, відступи та інші властивості елементів HTML, що дозволило забезпечити єдність стилю та зовнішній вигляд всієї системи. Також CSS було використано для розробки адаптивного дизайну, щоб забезпечити коректне відображення сторінок на різних пристроях та роздільних здатностях екрану.

Інформацію про CSS підготовлено на основі матеріалу [28].

TypeScript - це вільна та відкрита високорівнева мова програмування, розроблена компанією Microsoft, яка додає статичне типізування з необов'язковими анотаціями типів до JavaScript. Вона призначена для розробки великих програм та транспілюється у JavaScript. Оскільки TypeScript є надмножиною JavaScript, всі програми JavaScript синтаксично вірні для TypeScript, але вони можуть не пройти перевірку типів з міркувань безпеки.

TypeScript можна використовувати для розробки JavaScript-застосунків як на стороні клієнта, так і на стороні сервера (як у Node.js, Deno або Bun). Існують кілька варіантів транспіляції. Можна використовувати стандартний компілятор TypeScript, або викликати компілятор Babel для перетворення TypeScript у JavaScript.

TypeScript використовувався для розробки клієнтської та серверної частини мого проекту. Використання TypeScript допомагло уникнути багатьох типових помилок на етапі розробки, полегшило рефакторинг коду та зробило його більш читабельним та розуміним. Також використання TypeScript надало можливість використовувати сучасні функціональні можливості мови, такі як стрілкові функції, розширені типи даних, інтерфейси, а також властивості і методи класів, що посприяло покращенню продуктивності та якості коду.

Інформацію про TypeScript підготовлено на основі матеріалу [30].

JavaScript, часто скорочується як JS, є мовою програмування та основною технологією Вебу, разом з HTML та CSS. 99% веб-сайтів використовують JavaScript на клієнтському боці для поведінки веб-сторінок.

Веб-браузери мають спеціальний двигун JavaScript, який виконує клієнтський код. Ці двигуни також використовуються на деяких серверах та в різноманітних додатках. Найпопулярніша система виконання для використання поза браузером - це Node.js.

JavaScript є високорівневою мовою, часто компілюється майже в реальному часі, що відповідає стандарту ECMAScript. Вона має динамічне типізування, об'єктно-орієнтоване програмування на основі прототипів та

функції першого класу. Вона багатопарадигмальна, підтримує подійно-орієнтований, функціональний та імперативний стилі програмування. JavaScript має програмні інтерфейси (API) для роботи з текстом, датами, регулярними виразами, стандартними структурами даних та Об'єктно-орієнтованою моделлю документу (DOM).

Інформацію про JavaScript підготовлено на основі матеріалу [29].

React - це популярна JavaScript бібліотека з відкритим кодом для створення динамічних інтерфейсів користувача (UI) веб-додатків. Розроблена компанією Facebook, вона використовується для побудови складних односторінкових додатків (SPA) та масштабних веб-застосунків.

React - це потужна та гнучка бібліотека, яка стала стандартом для розробки сучасних веб-інтерфейсів. Її компонентний підхід, JSX синтаксис та віртуальний DOM дозволяють створювати динамічні, масштабовані та керовані інтерфейси. Велика спільнота та екосистема React роблять її привабливим вибором для розробників фронт-енду різного рівня.

За допомогою бібліотеки React розроблено клієнтську частину проекту. React використано для створення інтерфейсу користувача, компонентів та їх взаємодії на веб-сторінках. Завдяки React реалізовано компонентну структуру додатку, що дозволило ефективно організувати код та робити розробку більш простою та швидкою. Компоненти React використовувалися для створення різноманітних елементів інтерфейсу, таких як форми, кнопки, списки файлів, та для управління їх станом та взаємодією з користувачем. Реактивність та швидкодія React сприяли поліпшенню користувацького досвіду та забезпечили більшу динамічність та інтерактивність додатку.

Інформацію про React підготовлено на основі матеріалу [31].

Vite - це сучасний збірник JavaScript, який швидко набирає популярності. Він використовує революційний підхід до збирання, що робить його значно швидшим за традиційні збирачі, такі як Webpack.

Vite можна використовувати для збирання будь-якого JavaScript-додатку. Він добре підходить для SPA (односторінкових додатків), веб-компонентів та бібліотек.

Vite - це сучасний, швидкий та зручний збирач JavaScript, який може стати чудовою альтернативою Webpack. Він добре підходить для SPA, веб-компонентів та бібліотек.

Збірник Vite був використаний для швидкої розробки та побудови клієнтської частини додатку. Vite надав швидкий процес розгортання, швидкість перезавантаження та оптимізований шлях до розробки за допомогою сучасних інструментів, таких як ESM (EcmaScript Modules). Його використання спростило процес налаштування середовища розробки та дозволило швидше створювати та тестувати компоненти інтерфейсу користувача. Збірник Vite допоміг забезпечити швидку розробку та покращений досвід розробників завдяки своїй продуктивності та простоті використання.

Інформацію про Vite підготовлено на основі матеріалу [32].

Для розробки серверної частини було використано сучасний фреймворк Nest.js, платформу Node.js для виконання програми та ORM sequelize для роботи з базою даних.

Nest (NestJS) - це фреймворк для побудови ефективних та масштабованих серверних додатків на Node.js. Він використовує прогресивний JavaScript, побудований з використанням та повністю підтримує TypeScript (але все ще дає розробникам можливість програмувати на чистому JavaScript) та поєднує елементи ООП (Об'єктно-орієнтованого програмування), ФП (Функціонального програмування) та FRP (Функціонально-реактивного програмування).

Nest надає готову архітектуру додатків, яка дозволяє розробникам та командам створювати високотестирувальні, масштабовані, слабкозв'язані та легкі в обслуговуванні додатки. Архітектура сильно інспірована Angular [33].

Nest.js використовувався для створення серверної частини додатку. Його використання дозволило побудувати потужну та масштабовану серверну частину за допомогою TypeScript. Nest.js надає різноманітні функціональні можливості, такі як модульність, вбудовану підтримку для обробки HTTP-запитів, систему middleware, сервіси, провайдери та інші. Використання Nest.js спростило розробку серверної логіки, забезпечуючи чистий та організований код, а також дозволило легко керувати залежностями та робити код більш підтримуваним і розширюваним. В цілому, Nest.js допоміг забезпечити ефективну та надійну роботу серверної частини додатку.

ORM Sequelize – це потужний інструмент для роботи з базами даних у середовищі Node.js, який дозволяє зручно та ефективно взаємодіяти з реляційними базами даних, такими як MySQL, PostgreSQL, SQLite та іншими. У цій статті ми розглянемо основні можливості та переваги використання ORM Sequelize, а також надамо приклади його застосування.

ORM Sequelize є потужним інструментом для роботи з базами даних у середовищі Node.js. Він надає зручний та ефективний інтерфейс для взаємодії з реляційними базами даних, що дозволяє розробникам швидко та легко реалізовувати складні операції з даними. З врахуванням його основних переваг, ORM Sequelize стає прекрасним вибором для будь-якого проекту, який вимагає роботи з базами даних в Node.js [34].

ORM Sequelize використовувався для забезпечення зручного та продуктивного доступу до бази даних MariaDB з використанням Node.js. Використання ORM Sequelize дозволило забезпечити консистентність та безпеку взаємодії з базою даних, а також зробило процес розробки більш простим та ефективним.

Node.js - це середовище виконання JavaScript, яке дозволяє розробникам створювати швидкі та ефективні додатки на серверному боці. Завдяки своїй асинхронності та подійно-орієнтованому підходу, Node.js став незамінним

інструментом для створення веб-додатків, API та інших серверних застосунків. У цій статті ми розглянемо основні переваги використання Node.js та його важливу роль у сучасній веб-розробці.

Node.js - це потужне та інноваційне середовище для розробки серверних додатків. Його висока швидкість, асинхронність, розширюваність та активна ком'юнітет роблять його одним з найпопулярніших інструментів у сфері веб-розробки. Використання Node.js дозволяє розробникам створювати швидкі, ефективні та масштабовані додатки, які задовольняють потреби сучасного інтернет-світу [35].

Node.js був використаний для розробки серверної частини додатку. За допомогою Node.js було створено сервер, який обробляє запити від клієнтів та здійснює взаємодію з базою даних. Цей сервер дозволяє виконувати операції з обробки даних, автентифікації та авторизації користувачів, а також забезпечує взаємодію з іншими компонентами системи. Використання Node.js дозволило створити швидкий та ефективний сервер, який забезпечує стабільну та надійну роботу додатку, а також він використовувався для взаємодії з файловою системою в додатку.

Для розробки додатку використовувалися зручні та популярні інструменти, зокрема, браузер Google Chrome, а також доступ до Інтернету. Використання цих інструментів дозволило забезпечити швидкий доступ до ресурсів, необхідних для розробки, та забезпечило зручність і ефективність процесу створення програмного забезпечення.

На основі зазначених програмних рішень розроблено технічні рішення для інформаційної системи, що відповідають її вимогам та дозволяють реалізувати необхідні функції.

2.5 Рішення з технічного забезпечення.

Для користування інформаційною системою необхідний веб-браузер. Це означає, що будь-який пристрій з достатніми технічними характеристиками для запуску браузера може використовуватися для взаємодії з інформаційною системою.

Основоючись на вищезазначеній інформації, сформульовано технічні характеристики пристрою користувача, відповідно до системних вимог браузера Google Chrome. Для Windows:

- Процесор Intel Pentium 4 або пізнішої версії із підтримкою SSE3;
- Об'єм оперативної пам'яті не менше 512 мб;
- Відео пам'ять не менше 64 мб;
- Вільне місце на жорсткому диску не менше 350 мб;
- Розширення екрану не менше 800 × 600;
- Клавіатура;
- Миша.

Для Linux:

- Технічні характеристики аналогічні Windows.

Далі технічні характеристики залежать формуватимуться на основі версії ОС, на базі якої працює пристрій, так як мінімальні вимоги до операційної системи, що підтримується, завжди перевершують технічні характеристики, які необхідні для коректної роботи браузера.

Для MacOS:

- Пристрій працює на базі OS X El Capitan (10.11) і версіях вище.

Для Android:

- Пристрій працює на базі Android 5.0 (Lollipop) і версіях вище.

Для IOS:

- Пристрій працює на базі iOS 12.2 (iPadOS 12.2) і версіях вище.

Технічні характеристики та налаштування сервера відповідають вимогам компанії. Адміністрування та технічне обслуговування сервера здійснюються співробітниками компанії «Хмарне Сховище».

На основі розробленого технічного проекту інформаційної системи реалізується проект інформаційної системи.

2.6 Рішення щодо зберігання файлів.

У розроблюваній системі хмарного сховища метадані користувачів, файлів та прав доступу зберігаються в реляційній СУБД MariaDB, тоді як самі файли зберігати в тій же БД або у файловій системі сервера є нераціонально через обмеження масштабованості, продуктивності та надійності. Для розв'язання цієї задачі обрано використання S3-сумісного об'єктного сховища, яке забезпечує практично необмежене масштабування, високу відмовостійкість та гнучкий керований доступ через HTTP(S)-API.

Об'єктне сховище організовано у вигляді «бакетів», кожен з яких містить об'єкти (файли) з унікальними ключами та набором метаданих. У системі передбачається створення окремого бакета для користувацьких файлів, де ключ формуватиметься на основі ідентифікатора користувача, внутрішнього UUID файлу та, за потреби, структури папок, що дозволяє уникнути колізій і спростити логічне групування даних. Метадані про файл (оригінальна назва, MIME-тип, розмір, хеш-сума, власник, права доступу, ключ у S3) зберігатимуться в таблицях MariaDB і зв'язуватимуться з об'єктом у сховищі.

Процес завантаження файлу реалізується на боці бекенду (Nest.js) з використанням механізму приймання файлів та SDK/клієнта S3-сервісу. Після проходження валідації (тип файлу, розмір) сервер або завантажує файл напряму до S3-сховища, або застосовує попередню обробку (наприклад, генерацію ескізів зображень) та записує об'єкт шматками (chunking) для великих файлів, використовуючи multi-part upload, що підвищує надійність передавання та дозволяє відновлювати перервані сесії. Для завантаження й завантаження файлів застосовуватимуться тимчасові підписані URL-адреси або проксі-ендпоїнти бекенду, що дає змогу поєднати безпеку доступу з високою пропускнуою здатністю S3.

S3-сумісне сховище також забезпечує вбудовані механізми версіонування, реплікації та класів зберігання, що дає можливість реалізувати політики життєвого циклу даних: активні файли зберігати в основному класі, архівні — переводити до більш дешевого, а критично важливі дані дублювати між регіонами або дата-центрами. На відміну від локального зберігання, такий підхід спрощує горизонтальне масштабування системи, оскільки бекенд-сервери стають «безстанними» щодо файлів і взаємодіють з єдиним централізованим об'єктним сховищем, що добре узгоджується з вибраною архітектурою та вимогами до відмовостійкості й якості зберігання даних.

У другому розділі було розроблено технічний проект інформаційної системи хмарного сховища, що стало логічним продовженням сформованої в першому розділі концепції. В результаті роботи були виконані такі ключові етапи:

- Розроблено загальносистемні рішення, визначено архітектуру системи на основі принципу горизонтального масштабування. Це включає перехід до «безстанних» (stateless) бекенд-серверів та використання централізованого об'єктного сховища для забезпечення високої доступності, відмовостійкості та ефективного управління життєвим циклом даних.
- Деталізовано рішення по інформаційному забезпеченню, де послідовно розроблено концептуальну, логічну та фізичну моделі даних. Це забезпечило структуроване зберігання метаданих (інформація про файли, користувачів, права доступу) та слугує основою для роботи системи управління базами даних.
- Обґрунтовано рішення з програмного забезпечення з вибором технологічного стеку, що відповідає вимогам до безпеки, продуктивності та масштабованості. Визначено основні програмні компоненти, їхні функції та взаємодію, що є необхідним для подальшої реалізації проекту.
- Сформовано рішення з технічного та математичного забезпечення, що включають опис необхідного обладнання та системних ресурсів, а також

ключові алгоритми для реалізації функціоналу (наприклад, для забезпечення цілісності даних чи оптимізації доступу до них).

Розроблений технічний проект інформаційної системи хмарного сховища визначає повний набір архітектурних, інформаційних та програмно-апаратних рішень, на основі якого проводиться практична реалізація системи. Наступним етапом є реалізація проекту інформаційної системи хмарного сховища.

Розділ 3. Реалізація проекту інформаційної системи хмарного сховища

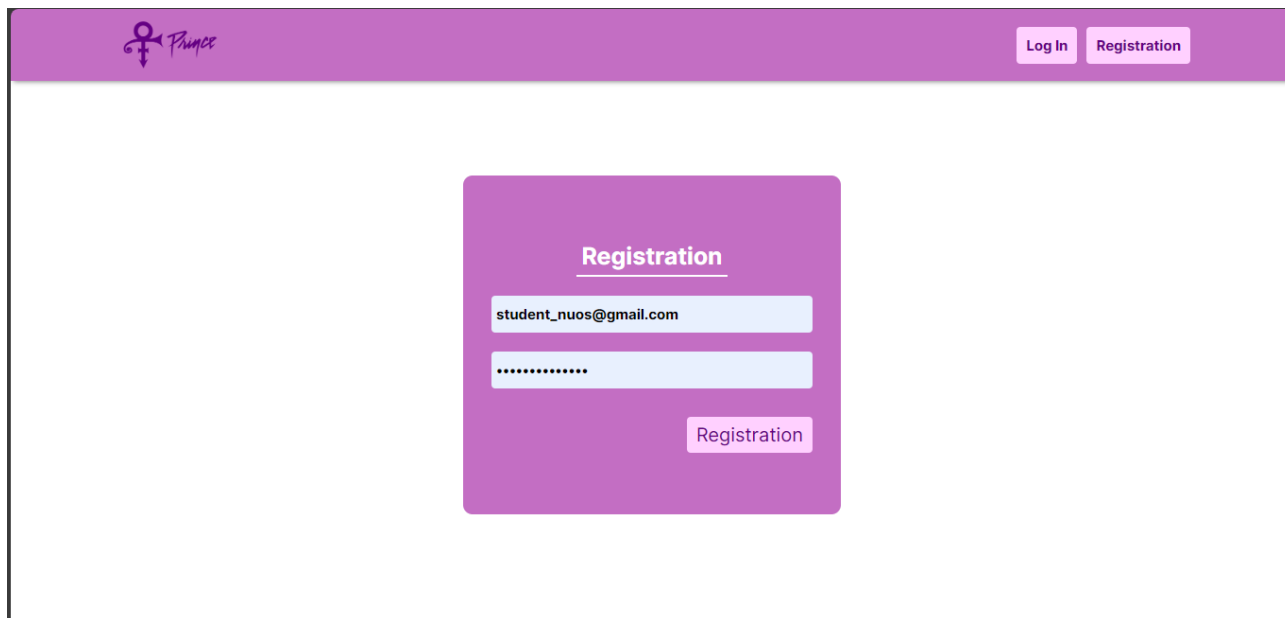
Інформаційна система хмарного сховища надає широкий функціонал для зручного та ефективного керування даними. Зокрема, система пропонує користувачам сторінку авторизації та аутентифікації, де вони можуть ввести свій логін та пароль для отримання доступу до сервісу. Після успішної реєстрації або аутентифікації, користувач потрапляє на головну сторінку системи.

На головній сторінці користувач має можливість виконати ряд дій з файлами та папками. Він може створити новий файл або завантажити існуючий, а також сортувати файли за різними критеріями, такими як дата, час додавання або розмір. Щоб полегшити сприйняття інформації, кожен файл та папка мають відповідну іконку, яка дозволяє користувачеві швидко розрізнити їх.

Система також пропонує функціонал пошуку файлів та папок за їхнім іменем, що дозволяє користувачеві швидко знаходити необхідні елементи. Кожна кнопка в інтерфейсі системи супроводжується відповідною іконкою, що полегшує розуміння її функціоналу.

У системи передбачено можливість зміни аватара користувача або видалення його. При видаленні аватара буде встановлена стандартна іконка користувача. Крім того, користувач може завантажувати файли на свій пристрій за допомогою відповідної кнопки або перетягуючи файли у вікно додатку у веб-браузері. Файли та папки можна видаляти вільно, а також завантажувати на пристрій користувача зручним для нього способом.

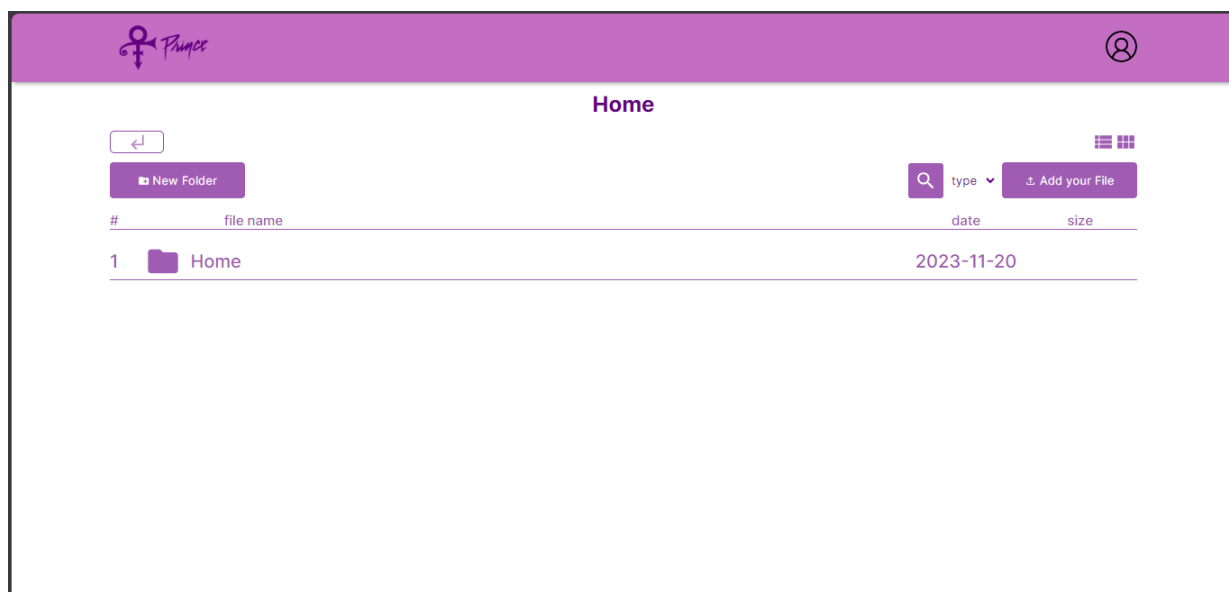
На рисунку 3.1 зображено сторінку реєстрації, де у верхньому куті можна перейти на сторінку авторизації.



The screenshot shows a web application interface with a purple header. On the left of the header is a logo consisting of a stylized female symbol and the word "Prince". On the right are two buttons: "Log In" and "Registration". The main content area is white and contains a purple rounded rectangle centered on the page. Inside this rectangle, the word "Registration" is underlined. Below it are two input fields: the first contains the email "student_nuos@gmail.com" and the second contains a series of dots for a password. At the bottom right of the purple rectangle is a "Registration" button.

Рисунок 3.1 – сторінка реєстрації

На рисунку 3.2 зображено головну сторінку з початковою папкою Home, яка створюється автоматично для кожного користувача.



The screenshot shows the "Home" page of the web application. The purple header contains the "Prince" logo on the left and a user profile icon on the right. The main content area has a white background with the word "Home" centered at the top. Below "Home" is a navigation bar with a back arrow button, a "New Folder" button, a search icon, a "type" dropdown menu, and an "Add your File" button. Below the navigation bar is a table with columns for "#", "file name", "date", and "size". The table contains one row with the number "1", a folder icon, the name "Home", the date "2023-11-20", and an empty "size" column.

Рисунок 3.2 – домашня сторінка користувача

На рисунку 3.3 зображена головна сторінка з відкритим полем для створення нової папки, в якій потрібно ввести ім'я папки, яке ви хочете їй дати та натиснути «».

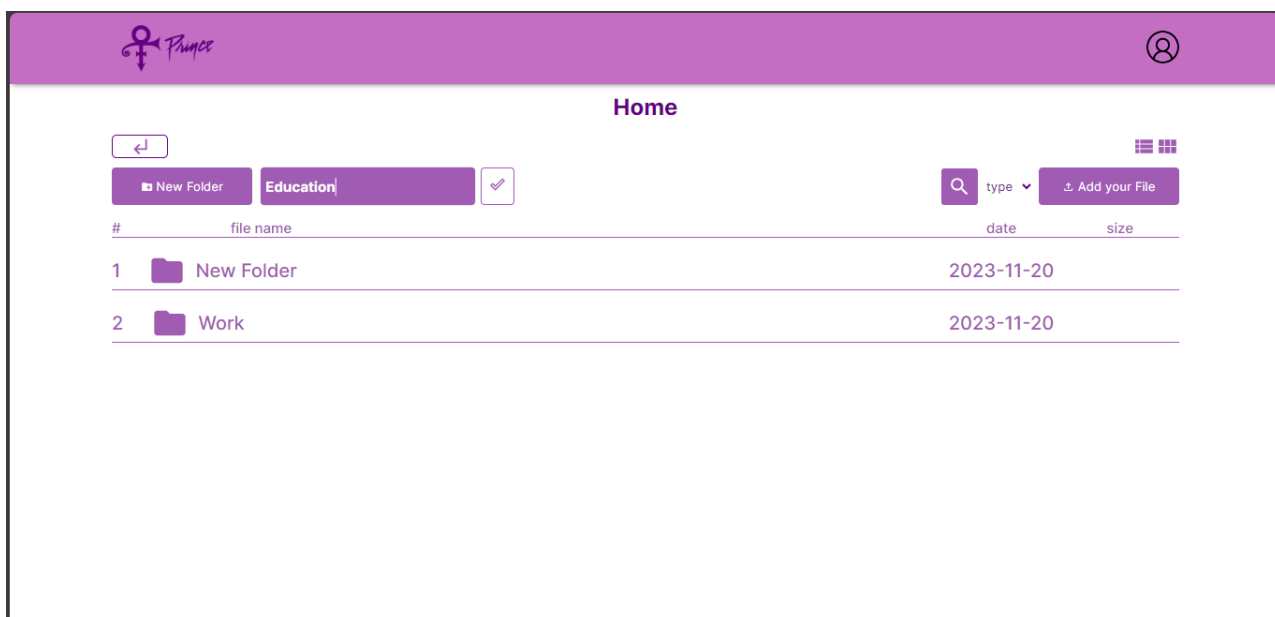


Рисунок 3.3 – створення папок

На рисунку 3.4 зображена головна сторінка зі зміненим відображенням папок з вертикального списку на горизонтальні іконки.

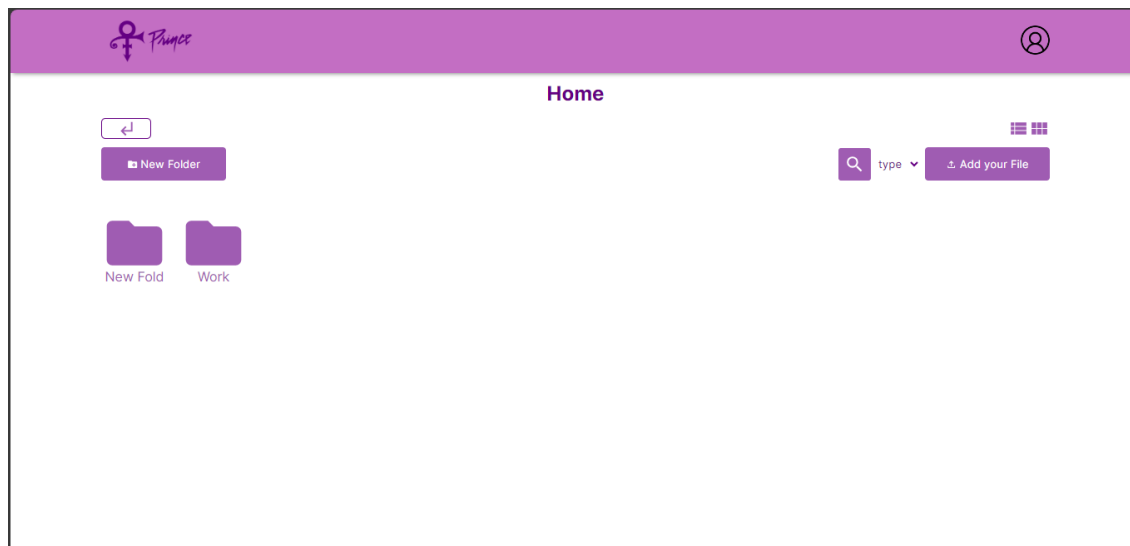


Рисунок 3.4 – змінений вид на горизонтальні іконки

Апробація та експлуатація хмарного сховища показали позитивні результати.

Проект досягнув поставленої цілі, розроблена інформаційна система значно полегшує життя користувачів та компаній, які використовують її сервіси.

Використання хмарного сховища дозволяє економити час та ресурси, оскільки вона надає зручний та швидкий доступ до даних з будь-якого пристрою, підключеного до Інтернету. Це зменшує трудовитрати на накопичувачі та сприяє підвищенню продуктивності користувачів.

Зручний та надійний інструмент для зберігання та обміну даними стає важливою перевагою у сучасному бізнес-середовищі, що привертає нових клієнтів та сприяє збереженню існуючих.

Технічна інструкція використання описана в додатку А

Розділ 4 Техніко-економічне обґрунтування розробки інформаційної системи хмарного сховища

Метою економічного обґрунтування є порівняння варіантів побудови інфраструктури хмарного сховища для компанії-стартапу та вибір рішення, яке забезпечує необхідний рівень якості послуг при мінімально можливих витратах протягом життєвого циклу системи. Для оцінювання використовуються основні параметри: функціональність, ефективність (продуктивність та витрати ресурсів), надійність, масштабованість та супроводжуваність, що відповідає типовому підходу функціонально-вартісного аналізу.

4.1 Вибір варіантів концепції інфраструктури.

Для розміщення серверної частини та зберігання файлів інформаційної системи розглядаються такі варіанти:

- Варіант 1 – повністю локальна інфраструктура: власні сервери додатку, бази даних і файлового сховища в орендованому стійковому просторі дата-центру.
- Варіант 2 – оренда віртуальних серверів у хмарного провайдера (IaaS) із локальним зберіганням файлів на дисках VM.
- Варіант 3 – гібридне рішення: бекенд (Nest.js) і MariaDB на керованих віртуальних серверах, а файли користувачів – в S3-сумісному об'єктному сховищі провайдера.
- Варіант 4 – використання керованої PaaS-інфраструктури та S3-подібного сховища з автоматичним масштабуванням обчислювальних ресурсів.

У таблиці 4.1 подано орієнтовні технічні характеристики та склад ресурсів для кожного варіанта (тип серверів, обсяг оперативної пам'яті, дискового простору, пропускна здатність мережі тощо) [46].

Таблиця 4.1: Технічні характеристики, оцінки параметрів та TCO варіантів інфраструктури

Параметр / Варіант	Варіант 1 (Локальна)	Варіант 2 (IaaS VPS)	Варіант 3 (Гібрид S3)	Варіант 4 (PaaS)
Технічні характеристики	- Сервер: 4C/8GB/160GB SSD	- VPS: 2vCPU/4GB/80GB	- VPS: 2vCPU/4GB/80GB (AWS Lightsail €10)	- PaaS: Railway/Render 1GB RAM 500GB
Функціональність (0-10)	8	9	10	9
Ефективність (0-10)	6	8	9	10
Надійність (0-10)	7	8	10	9
Масштабованість (0-10)	5	7	9	10
Супроводжуваність (0-10)	4	7	9	10
Інтегральний $K_{тр}$ (ф: 0.25/0.20/0.25/0.20/0.10)	6.45 (0.258 + 0.206 + 0.257 + 0.205 + 0.10*4)	7.90	9.65	9.70
Річні витрати (TCO, €)	3600 (€300 + €250/mic)	150 (€12.5/mic)	200 (€15/mic VPS + €3 storage)	180 (€15/mic)
$K_{тр} / C_j$ (нормалізовано)	0.0018	0.0527	0.0483	0.0539

Результати техніко-економічного аналізу, представлені в підрозділі 4.1, дозволили визначити найбільш вигідні з точки зору витрат варіанти інфраструктури. Проте, для повноцінного вибору доцільно провести функціональний аналіз, який детально оцінить здатність кожного варіанту задовольнити ключові вимоги до розроблюваної системи.

4.2 Функціональний аналіз варіантів.

Для оцінки варіантів визначено параметри системи: функціональність, ефективність, надійність, масштабованість, супроводжуваність. Кожен параметр оцінюється експертно за 10-бальною шкалою для кожного варіанта інфраструктури. Усереднені оцінки наведено в таблиці 4.2 (за аналогією з прикладом: рядки – параметри, стовпці – варіанти 1–4).[47]

Після цього визначаються вагові коефіцієнти φ_i для кожного параметра, що відображають їх відносну важливість для стартапу, орієнтованого на якість зберігання даних:

Таблиця 4.2: Технічні характеристики та оцінки варіантів інфраструктури.

Параметр	Коефіцієнт, φ_i
Функціональність	0.25
Ефективність	0.20
Надійність	0.25
Масштабованість	0.20
Супроводжуваність	0.10

(Сума коефіцієнтів дорівнює 1.0.) Значення φ_i встановлюються експертно з урахуванням того, що для хмарного сховища критичними є саме надійність, функціональність та здатність до масштабування.[48]

Для кожного варіанта j обчислюється показник технічного рівня за формулою

$$K_{тр,j} = \sum_{i=1}^n \varphi_i \cdot B_{ij}, K_{тр,j} = i = 1 \sum \varphi_i \cdot B_{ij},$$

де B_{ij} – бальна оцінка i -го параметра для j -го варіанта. Результати розрахунку зводяться у таблицю 4.2, а зведені значення $K_{тр,j}$ - у таблицю 4.3, де для кожного варіанта подається інтегральний показник технічного рівня.

Таблиця 4.3: Зведений показник технічного рівня варіантів

Варіант	$K_{тр,j}$	Розрахунок
Варіант 1 (Локальна)	6.45	$0.25 \cdot 8 + 0.20 \cdot 6 + 0.25 \cdot 7 + 0.20 \cdot 5 + 0.10 \cdot 40.25 \cdot 8 + 0.20 \cdot 6 + 0.25 \cdot 7 + 0.20 \cdot 5 + 0.10 \cdot 4$
Варіант 2 (IaaS VPS)	7.90	$0.25 \cdot 9 + 0.20 \cdot 8 + 0.25 \cdot 8 + 0.20 \cdot 7 + 0.10 \cdot 70.25 \cdot 9 + 0.20 \cdot 8 + 0.25 \cdot 8 + 0.20 \cdot 7 + 0.10 \cdot 7$

Варіант 3 (Гібрид S3)	9.65	$0.25 \cdot 10 + 0.20 \cdot 9 + 0.25 \cdot 10 + 0.20 \cdot 9 + 0.10 \cdot 90.25W_{c \cdot 10} + 0.20W_{c \cdot 9} + 0.25W_{c \cdot 10} + 0.20W_{c \cdot 9} + 0.10W_{c \cdot 90.25 \cdot 10 + 0.20 \cdot 9 + 0.25 \cdot 10 + 0.20 \cdot 9 + 0.10 \cdot 9}$
Варіант 4 (PaaS)	9.70	$0.25 \cdot 9 + 0.20 \cdot 10 + 0.25 \cdot 9 + 0.20 \cdot 10 + 0.10 \cdot 100.25W_{c \cdot 9} + 0.20W_{c \cdot 10} + 0.25W_{c \cdot 9} + 0.20W_{c \cdot 10} + 0.10W_{c \cdot 100.25 \cdot 9 + 0.20 \cdot 10 + 0.25 \cdot 9 + 0.20 \cdot 10 + 0.10 \cdot 10}$

Отримані кількісні показники фінансових витрат є ключовою складовою для фіналізації вибору. На основі цих розрахунків і з урахуванням якісних критеріїв, що були визначені раніше, у наступному підрозділі буде проведено комплексне порівняння, економічна оцінка та обґрунтування вибору найбільш ефективного варіанта реалізації інформаційної системи.

4.3 Економічна оцінка та вибір варіанта.

Паралельно проводиться розрахунок сумарних витрат на володіння (ТСО) для кожного варіанта: одноразові капітальні витрати (придбання або налаштування серверів, ліцензій, початкове розгортання) та щорічні експлуатаційні витрати (оренда обладнання чи ресурсів, оплата хмарних сервісів, електроенергія, адміністрування). На підставі комерційних пропозицій хмарних провайдерів і типових тарифів для малого бізнесу формується таблиця порівняння, де для кожного варіанта вказується орієнтовна річна вартість інфраструктури [31].

Для прийняття рішення використовується співвідношення «технічний рівень / витрати». Перевага надається варіанту, що має найбільше значення $K_{тр,j}/K_{тр,i}$

при прийнятному рівні витрат (або найвище значення відношення $K_{тр,j}/C_j K_{тр,j}/C_j$, де C_j – річні витрати для варіанта j). У більшості практичних сценаріїв для стартапів найкращим виявляється гібридний варіант з використанням віртуальних серверів і S3-сумісного сховища, оскільки він забезпечує високу надійність і масштабованість при помірних початкових витратах та можливості гнучко змінювати ресурси залежно від навантаження [49].

Таким чином, проведений функціонально-вартісний аналіз та порівняння вартості володіння показують доцільність впровадження обраної архітектури інформаційної системи хмарного сховища з використанням хмарної інфраструктури та об'єктного S3-подібного сховища. Обраний варіант забезпечує баланс між технічними показниками (надійність, масштабованість, продуктивність) та економічною ефективністю, що відповідає стратегічним цілям стартап-компанії.

4.4 Очікувані результати й ефективність реалізації обраного варіанта концепції системи.

Оскільки обраний варіант інфраструктури хмарного сховища (віртуальні сервери з використанням S3-сумісного об'єктного сховища) задовольняє основним вимогам до функціональності, надійності та масштабованості, його впровадження дозволяє очікувати досягнення таких результатів:

- забезпечення безпечного та безперервного зберігання файлів користувачів із гарантованою доступністю сервісу та можливістю відновлення даних у разі збоїв завдяки використанню реплікації й резервного копіювання;
- підвищення ефективності роботи користувачів і робочих команд за рахунок швидкого доступу до даних з будь-яких пристроїв, спільного

доступу до файлів і зручних засобів пошуку та організації інформації;

- зменшення капітальних витрат стартап-компанії на власну апаратну інфраструктуру за рахунок оренди хмарних ресурсів і поетапного масштабування потужностей відповідно до фактичного навантаження;
- спрощення супроводу та розвитку системи завдяки використанню сучасних технологій (Nest.js, Next.js, S3-подібне сховище) і модульних архітектурних підходів, що знижує експлуатаційні витрати та скорочує час впровадження нових функцій [31].

У сукупності це забезпечує досягнення цілей проєкту – створення конкурентоспроможного хмарного сховища з високою якістю зберігання даних, гнучкою масштабованою інфраструктурою та економічно обґрунтованими витратами на його розробку й експлуатацію [49].

4.5 Умови приймання системи.

Приймання розробленої інформаційної системи хмарного сховища здійснюється спільною комісією представників компанії-розробника та замовника (або внутрішнього замовника у межах стартапу) на підставі технічного завдання, вимог до якості зберігання даних і показників, визначених у технічному проєкті. До складу приймальних випробувань входять: перевірка реалізації функціональних вимог (реєстрація користувачів, завантаження та завантаження файлів, пошук, спільний доступ), тестування механізмів захисту та резервування даних, оцінка продуктивності під навантаженням, а також перевірка відповідності архітектурних і технічних рішень затвердженій концепції [47].

За результатами успішного проходження випробувань складається двосторонній акт приймання-передавання системи в промислову експлуатацію, у якому фіксуються основні техніко-економічні показники, перелік реалізованих функцій, умови подальшого супроводу та відповідальність сторін. Зазначений акт є підставою для переходу до етапу впровадження системи у виробниче середовище та подальшої її експлуатації [50].

Розділ 5. Охорона праці.

Вимоги безпеки та захисту здоров'я працівників при роботі з екранними пристроями затверджено наказом Міністерства соціальної політики від 14.02.2018 № 207. Ці вимоги поширюються на всіх суб'єктів господарювання незалежно від форм власності, організаційно-правової форми та видів діяльності та встановлюють мінімальні вимоги безпеки та захисту здоров'я при здійсненні роботи, пов'язаної з використанням екранних пристроїв незалежно від їх типу та моделі.

Вимоги не поширюються на:

- Робочі місця здобувачів освіти у комп'ютерних класах (кабінетах, аудиторіях тощо) навчальних закладів;
- Робочі місця пілотів, водіїв або операторів транспортних засобів, обладнані екранними пристроями у системах обробки даних на борту засобів сполучення, екранні пристрої у складі машин та обладнання, що переміщуються у процесі роботи;
- Робочі місця працівників, які займаються обслуговуванням, ремонтом та налагодженням екранних пристроїв;
- Портативні системи обробки даних, якщо вони не завжди використовуються на робочому місці;
- Обчислювальні машини (калькулятори), касові апарати та пристрої з невеликими пристроями індикації даних або результатів вимірювання;
- Цифрові друкарські машини, які обладнані візуальними дисплейними терміналами (дисплейні друкарські машини);
- Планшети, смартфони, мобільні телефони.

Роботодавець повинен проінформувати працівників під розписку про умови праці та наявність на їх робочих місцях небезпечних та шкідливих виробничих факторів (фізичних, хімічних, біологічних, психофізіологічних), які виникають при роботі з екранними пристроями та ще не усунуті, а також про можливі наслідки їх впливу на здоров'я працівників.

Роботодавець повинен забезпечити навчання та перевірку знань працівників з питань охорони праці та безпечного використання екранних пристроїв до початку роботи з ними, а також у випадках модифікації та організації роботи обладнання.

Роботодавець повинен вжити відповідних заходів, щоб забезпечити відповідність робочого місця працівника цим вимогам.

Під час облаштування робочого місця працівника з екранними пристроями необхідно вибирати таке обладнання, яке не створює надмірного шуму і не виділяє надмірного тепла. Рівні шуму на робочих місцях осіб, які працюють з екранними пристроями, повинні відповідати вимогам Санітарних норм виробничого шуму, ультразвуку та інфразвуку ДСН 3.3.6.037-99, затверджених ухвалою Головного державного санітарного лікаря України від 01.12.1999 № 37.

Роботодавець повинен за рахунок тривалості робочої зміни організувати внутрішні регламентовані перерви для відпочинку відповідно до Державних санітарних правил та норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПН 3.3.2.007-98, затверджених постановою Головного державного санітарного лікаря від 10.12.1998 № 7.

Роботодавець повинен забезпечити власним коштом проведення медичних оглядів працівників відповідно до вимог Порядку проведення медичних оглядів працівників певних категорій, затвердженим наказом Міністерства охорони здоров'я України від 21.05.2007 № 246.

За результатами цих оглядів роботодавець у разі потреби повинен забезпечити виконання відповідних оздоровчих заходів.

Роботодавець зобов'язаний за необхідності проводити лабораторні дослідження умов праці працівників з метою виявлення шкідливих та небезпечних факторів виробничого середовища, тяжкості та напруженості трудового процесу (зокрема щодо виявлення ризиків, пов'язаних із погіршенням зору, порушенням фізичного стану, стресом) та здійснювати заходи щодо усунення виявлених ризиків.

Робочі місця працівників з екранними пристроями мають бути спроектовані і мати такі розміри, щоб працівники мали простір зміни робочого становища і рухів.

Для забезпечення безпеки та захисту здоров'я працівників все випромінювання від екранних пристроїв має бути зведене до гранично допустимого рівня (вплив на людину факторів середовища - шуму, вібрації, забруднювачів, температури тощо, що не викликає соматичних або психічних розладів, а також змін стану здоров'я, працездатності, поведінки, що виходять за межі пристосувальних реакцій з точки зору безпеки та охорони здоров'я працівників.

Організація робочого місця працівника з екранними пристроями повинна забезпечувати відповідність всіх елементів робочого місця та їх розташування ергономічним, антропологічним, психофізіологічним вимогам, а також характером виконуваних робіт.

Висвітлення робочого місця працівника з екранними пристроями має створювати відповідний контраст між екраном та навколишнім середовищем (з урахуванням виду роботи) та відповідати вимогам ДСанПІН 3.3.2.007-98.

Мікроклімат виробничих приміщень з робочими місцями працівників з екранними пристроями повинен підтримуватися на постійному рівні та відповідати вимогам Санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042-99, затверджених постановою Головного державного санітарного лікаря України від 0.12.1999 №42 (далі - ДСП 3.3.6.042-99).

Робочий стіл або робоча поверхня повинні бути достатнього розміру та мати поверхню з низькою відбивною здатністю, допускати гнучкість при розміщенні екрана, клавіатури, документів та відповідного обладнання.

Робоче крісло має бути стійким і дозволяти працівникові з екранними пристроями легко рухатися та займати зручне положення.

Сидіння має регулюватися по висоті, спинка сидіння як по висоті, так і по нахилу. Слід передбачати підніжку для тих, кому це потрібно для зручності.

Мінімальні вимоги безпеки під час роботи з екранними пристроями:

- Щодня перед початком роботи необхідно очищати екранні пристрої від пилу та інших забруднень;
- Після завершення роботи екранні пристрої слід відключати від електричної мережі;
- У разі виникнення аварійної ситуації необхідно негайно відключити пристрій від електричної мережі.

Не допускається:

- Виконувати технічне обслуговування, ремонт та налагодження екранних пристроїв безпосередньо на робочому місці працівника під час роботи з екранними пристроями;
- Відключати захисні пристрої, самовільно проводити зміни у конструкції та складі екранних пристроїв або їх технічне налагодження;
- Працювати з екранними пристроями, де під час роботи виникають нехарактерні сигнали, нестабільне зображення на екрані та інші несправності.

При виконанні робіт операторського типу, пов'язаних з нервово-емоційною напругою, у приміщеннях при роботі з екранними пристроями, на пультах та постах управління технологічними процесами та в інших приміщеннях повинні дотримуватися оптимальних умов мікроклімату відповідно до вимог ГСН 3.3.6.042 -99.

Мінімальні вимоги безпеки до екранних пристроїв:

- Екранні пристрої не повинні бути джерелом ризику для працівників;
- Все випромінювання, крім видимої частини електромагнітного спектра, має бути зведено до незначного рівня з погляду безпеки та охорони здоров'я працівників;
- Символи на екранних пристроях повинні бути чіткими, відповідного розміру. Між символами та рядками символів має бути належна відстань;
- Зображення на екрані має бути стабільним, без миготінь чи інших видів нестабільності;

- Яскравість та/або контрастність символів повинні легко регулюватися працівником під час роботи з екранними пристроями, а також швидко адаптуватися до навколишніх умов;

- Вибираючи екрани, слід віддавати перевагу таким екранам, які легко та вільно обертаються та нахиляються відповідно до потреби працівника;

- При необхідності може використовуватися окрема підставка або регульований стіл для розміщення екрана;

- Екран не повинен відблискувати або відображати світло, щоб не викликати дискомфорту у працівника під час роботи з екранними пристроями;

- Вибираючи клавіатуру, слід віддавати перевагу такій клавіатурі, яка відкидається і є автономною (відокремленою від екрану), щоб працівник міг вибрати зручну робочу позу та уникнути втоми рук (кисті та верхньої частини руки);

- Поверхня клавіатури має бути матовою, щоб уникнути відображення. Розташування клавіш і клавіші повинні полегшувати роботу з клавіатурою. Позначення клавіш має бути досить контрастним та розбірливим;

- Обладнання, що входить до робочої станції, не повинно виділяти надлишкового тепла, що може заподіяти незручності працівникам при роботі з екранними пристроями;

- При розробці, виборі, замовленні та модифікації програмного забезпечення, а також при розробці задач, що передбачають використання обладнання з екранними пристроями, роботодавець повинен керуватися таким програмним забезпеченням, яке відповідає розв'язуванню завдань і просто у використанні, а де необхідно адаптованим до рівня знань та досвіду працівника.

Відповідно до статті 43 Закону України «Про охорону праці» за порушення законодавства про охорону праці та невиконання приписів (розпоряджень) посадових осіб органів виконавчої влади з нагляду за охороною праці юридичні та фізичні особи, які згідно із законодавством використовують найману працю, залучаються органами виконавчої влади з нагляду за охороною праці до сплати штрафу у порядку, встановленому законом.

Сплата штрафу не звільняє юридичну чи фізичну особу, яка відповідно до законодавства використовує найману працю, від усунення виявлених порушень у визначені терміни.

Максимальний розмір штрафу не може перевищувати п'яти відсотків середньомісячного фонду заробітної плати за попередній рік юридичної або фізичної особи, яка відповідно до законодавства використовує найману працю.

Несплата або неповна сплата юридичними або фізичними особами, які відповідно до законодавства використовують найману працю, штрафу тягне за собою нарахування пені на несплачену суму штрафу (його частини) з розрахунку 120% річних облікової ставки Національного банку України, що діяла в період такої несплати, за кожен день прострочення [36].

Розділ 6. Охорона навколишнього середовища.

Стрімкий розвиток інформаційних технологій та перенесення більшості бізнес-процесів у цифрове середовище супроводжується зростанням енергоспоживання, збільшенням кількості серверного обладнання та обсягів електронних відходів, що безпосередньо впливає на стан навколишнього природного середовища. Центри обробки даних, телекомунікаційна інфраструктура та користувацьке обладнання, необхідні для функціонування хмарних сховищ, опосередковано сприяють викидам парникових газів енергетичним сектором, споживанню природних ресурсів та формуванню відходів електроніки [40].

В Україні забезпечення екологічної безпеки та раціонального природокористування регламентується Законом «Про охорону навколишнього природного середовища», а також низкою спеціальних законів і державних програм у сфері енергозбереження, поводження з відходами та захисту атмосфери, водних і земельних ресурсів. Навіть програмні ІТ-проєкти, які на перший погляд не пов'язані з фізичним виробництвом, мають урахувати екологічні наслідки, оскільки їх експлуатація передбачає роботу дата-центрів,

мережевого обладнання та офісної інфраструктури, що створює сукупне навантаження на довкілля [42].

Розроблена інформаційна система хмарного сховища належить до сервісів, що потенційно обслуговують значну кількість користувачів і компаній, а отже, потребують стійкої, масштабованої й енергоємної інфраструктури. У контексті сталого розвитку важливо не лише забезпечити надійність і безпеку зберігання даних, а й мінімізувати екологічний слід проєкту шляхом упровадження енергоефективних рішень та екологічно відповідальної організації життєвого циклу ІТ-обладнання [40].

Метою даного розділу є аналіз можливих видів і джерел впливу розроблюваної системи хмарного сховища на навколишнє природне середовище, визначення нормативно-правових вимог у сфері охорони довкілля та обґрунтування комплексу організаційних і технічних заходів, спрямованих на зниження негативного впливу ІТ-інфраструктури на екосистеми. Досягнення поставленої мети передбачає розгляд особливостей функціонування дата-центрів, офісних приміщень і користувацького обладнання, які підтримують роботу хмарного сервісу, та пошук можливостей оптимізації їх енергоспоживання й екологічної безпеки [43].

Актуальність екологічного аспекту роботи хмарних сховищ зумовлена тим, що інформаційно-комунікаційні технології нині розглядаються як один із значущих споживачів енергії у світі, а дата-центри входять до переліку об'єктів, що демонструють стабільну тенденцію до зростання енергоспоживання. За оцінками досліджень, частка ІКТ-сектору у глобальних викидах парникових газів порівнювана з авіаційною галуззю, а масштабування хмарних сервісів без упровадження «зелених» підходів може посилювати кліматичні ризики та навантаження на енергетичні системи[41].

Для України питання екологічної безпеки ІТ-інфраструктури набуває особливого значення у зв'язку з інтеграцією до європейського правового

простору, адаптацією екологічних директив ЄС та необхідністю виконання міжнародних зобов'язань у сфері скорочення викидів і підвищення енергоефективності. У таких умовах впровадження хмарних технологій повинно супроводжуватися заходами з оптимізації енергоспоживання серверів, використанням сучасних систем охолодження, організацією належного поводження з відпрацьованим обладнанням та переходом на відновлювані джерела енергії там, де це можливо [46].

Отже, врахування екологічних аспектів у проєкті хмарного сховища дає змогу не лише зменшити негативний вплив на довкілля, а й підвищити репутаційну привабливість сервісу, відповідність міжнародним і національним стандартам, а також створити підґрунтя для подальшого впровадження принципів «зеленої» ІТ-інфраструктури в діяльність компанії-розробника.[44]

5.1 Забруднення навколишнього середовища відходами та енергоспоживанням ІТ-інфраструктури.

У даній магістерській роботі розглядається діяльність компанії-стартапу, що займається розробкою та супроводженням інформаційної системи хмарного сховища. У процесі функціонування такої компанії утворюються офісні тверді побутові відходи (папір, пакувальні матеріали, пластик, одноразова електроніка), а також відпрацьоване комп'ютерне та мережеве обладнання, яке належить до категорії небезпечних електронних відходів. Додатковим фактором впливу є значне енергоспоживання серверів, мережевої інфраструктури та робочих місць, що опосередковано сприяє викидам парникових газів енергетичним сектором [37].

Проблема поводження з твердими побутовими та електронними відходами є актуальною для більшості підприємств України, включно з організаціями ІТ-сектору. Неналежне сортування та захоронення відходів призводить до забруднення ґрунтів, поверхневих і підземних вод токсичними компонентами (важкі метали, пластик, компоненти акумуляторів), а також до забруднення атмосферного повітря при спалюванні сміття. Особливо небезпечними є

електронні відходи, що містять цінні, але й токсичні матеріали, утилізація яких без дотримання екологічних вимог створює загрозу для довкілля та здоров'я населення [39].

У контексті проєкту хмарного сховища додатковим джерелом навантаження на навколишнє середовище є робота серверного обладнання в дата-центрах, де розміщується програмно-апаратна платформа системи. Для забезпечення безперервного доступу до даних сервери працюють у цілодобовому режимі, споживаючи електроенергію та потребуючи систем охолодження, що збільшує загальний вуглецевий слід сервісу. З огляду на це важливим завданням є впровадження енергоефективних рішень, використання обладнання з низьким енергоспоживанням, віртуалізація ресурсів і, за можливості, перехід на постачальників «зеленої» електроенергії або дата-центрів, що мають сертифікати енергоефективності [38].

Таким чином, для компанії-розробника хмарного сховища екологічні ризики пов'язані насамперед із утворенням твердих побутових і електронних відходів, а також із сукупним енергоспоживанням ІТ-інфраструктури. Їх мінімізація потребує впровадження системи роздільного збирання та передачі відходів на переробку, укладання договорів зі спеціалізованими підприємствами з утилізації електроніки та реалізації комплексу енергозберігальних заходів у дата-центрах і офісних приміщеннях [40].

5.2 Розробка заходів щодо зменшення забруднення.

Екологічно відповідне функціонування компанії, що розробляє та експлуатує хмарне сховище, передбачає перехід від стихійного поводження з відходами та енергоресурсами до системного управління їх утворенням, використанням і утилізацією. Для цього необхідно поєднати організаційні, технологічні та фінансові заходи, які дозволяють одночасно скорочувати обсяги

відходів, зменшувати споживання енергії та мінімізувати негативний вплив на довкілля.[37]

Традиційний підхід, за якого відходи просто передаються на захоронення, у сучасних умовах є екологічно та економічно неефективним, особливо для електронних компонентів, що містять цінні метали й небезпечні речовини. Для ІТ-підприємств доцільно впроваджувати концепцію комплексного управління відходами, яка передбачає їх роздільне збирання, повторне використання, передачу на переробку та лише в крайньому разі – захоронення на спеціалізованих полігонах [39].

Основні напрями зменшення екологічного навантаження для розроблюваної системи хмарного сховища такі:

- запровадження системи сортування офісних відходів (папір, пластик, скло, органічні рештки) та укладення договорів з ліцензованими операторами з їх вивезення й переробки;
- організація окремого збору та передачі на спеціалізовані підприємства відпрацьованого електронного обладнання (сервери, маршрутизатори, комп'ютери, акумулятори), що дає змогу вилучити вторинну сировину та запобігти потраплянню токсичних компонентів у довкілля;
- поступове оновлення парку техніки на енергоефективні моделі з сертифікатами енергозбереження, використання режимів енергозбереження та віртуалізації серверних ресурсів для зменшення споживання електроенергії [38].

Для дата-центрів, у яких планується розміщення хмарної інфраструктури, важливо враховувати енергоефективність систем живлення та охолодження, показник PUE (Power Usage Effectiveness), можливість використання

відновлюваних джерел енергії або «зеленої» електрики за договорами постачання. Такі заходи не лише знижують вуглецевий слід сервісу, але й забезпечують економію експлуатаційних витрат протягом життєвого циклу системи [44].

Фінансово-економічна складова природоохоронних заходів передбачає формування окремого бюджету на екологічні програми (утилізація електроніки, модернізація обладнання, впровадження енергоощадних рішень) та врахування можливих податкових пільг або грантових програм у сфері «зеленої» енергетики та цифрової трансформації. У результаті реалізації зазначених заходів компанія-розробник хмарного сховища здатна істотно скоротити свій негативний вплив на навколишнє середовище, підвищити рівень екологічної безпеки та відповідати сучасним вимогам сталого розвитку [40].

Висновок

У роботі виконано дослідження методів забезпечення якості зберігання даних у хмарних інформаційних системах, на основі якого здійснено розробку інформаційної системи хмарного сховища, орієнтованої на забезпечення надійного, безпечного та зручного зберігання даних для бізнесу, робочих команд та індивідуальних користувачів у хмарному середовищі. Запропонований підхід поєднує системний аналіз методів підвищення якості зберігання даних із практичною реалізацією веб-сервісу, що відповідає сучасним вимогам до масштабованості, доступності та інформаційної безпеки.

1. Проведено огляд фахової та наукової літератури, проаналізовано еволюцію технологій зберігання даних та сучасний стан ринку хмарних сховищ. Визначено ключові проблеми традиційних локальних рішень і обґрунтовано доцільність переходу до хмарних інформаційних систем як основи для забезпечення гнучкого, масштабованого та відмовостійкого зберігання даних.
2. Виконано порівняльний аналіз популярних хмарних сховищ (Google Drive, Dropbox, OneDrive, iCloud, MEGA) із побудовою інтегрального показника якості на основі системи критеріїв (обсяг безкоштовного сховища, шифрування, версіонування, інтеграція, кросплатформенність). На основі результатів аналізу сформовано функціональні та нефункціональні вимоги до розроблюваної інформаційної системи хмарного сховища та розроблено її концепцію.
3. Досліджено та порівняно методи забезпечення якості зберігання даних у хмарних системах, зокрема реплікацію, резервне копіювання, шифрування та контроль цілісності. Виконані розрахунки показників надійності, часу відновлення та впливу на продуктивність засвідчили доцільність комбінованого використання зазначених методів для досягнення високого рівня доступності та захищеності даних.

4. Розроблено технічний проєкт інформаційної системи хмарного сховища. Побудовано концептуальну, логічну та фізичну моделі даних, визначено архітектуру програмної системи на основі стеку Nest.js, Next.js, MariaDB та S3-подібного сховища, обґрунтовано рішення щодо програмного й технічного забезпечення, а також організації зберігання файлів.
5. Реалізовано проєкт хмарного сховища, що забезпечує аутентифікацію користувачів, завантаження та завантаження файлів, організацію структури каталогів, пошук, спільний доступ і базові механізми захисту даних. Проведене тестування підтвердило працездатність обраних рішень та відповідність системи сформульованим вимогам. Виконано техніко-економічне обґрунтування варіантів інфраструктури (локальна, IaaS, гібридна, PaaS) із розрахунком технічного рівня та загальних витрат на володіння. Показано, що для стартап-компанії найбільш доцільним є використання хмарної інфраструктури з S3-подібним об'єктним сховищем, яка забезпечує баланс між вартістю, масштабованістю та надійністю.
6. Розглянуто питання охорони праці та захисту навколишнього середовища при розробці та експлуатації інформаційної системи, окреслено вимоги до безпечної організації робочих місць, інформаційної інфраструктури та мінімізації негативного впливу на довкілля.
7. Наукова новизна роботи полягає у застосуванні системного аналізу методів підвищення якості зберігання даних для обґрунтування архітектурних та функціональних рішень інформаційної системи хмарного сховища, а також у комплексному поєднанні методів реплікації, резервного копіювання та шифрування з урахуванням показників надійності, доступності та продуктивності.

Отримані результати підтверджують, що розроблена інформаційна система хмарного сховища є перспективним рішенням для забезпечення високої якості зберігання даних і може бути використана як основа для подальшого

розвитку сервісів хмарної інфраструктури та підвищення конкурентоспроможності компанії.

Список джерел

1. Пасічник В.В., Шаховська Н.Б. – «Сховища даних: Навчальний посібник» - Львів: "Магнолія 2006", 2008.-492 с.
2. Марченко А.В. Проектування інформаційних систем 2016 90с..
3. Пальмов С.В. Проектування інформаційних систем. М: ИСТ- 72, 2011.
4. Ременяк Л.В. Проектування інформаційних систем: конспект лекцій. Одеса, Одеський державний екологічний університет, 2016. 152.
5. [High Adoption of Cloud-based Storage to Boost the Growth of Hyperscale Data Center Market: Technavio | Business Wire](#)
6. [Переваги та недоліки хмарних сервісів .: Ресурсний центр ГУРТ \(gurt.org.ua\)](#)
7. [Хмарні сховища — Вікіпедія \(wikipedia.org\)](#)
8. ["Основи баз даних" \(навчальний посібник для вишів, 2011\), структурована сукупність даних, моделі, зв'язки.](#)
9. [Навчальний посібник з проектування інформаційних систем В.А. Яланецького \(2020\) з комп'ютерним практикумом](#)
10. [ПРОЕКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ.](#)
11. [UML.](#)
12. [Overview | Feature-Sliced Design](#)
13. [Clean Coder Blog](#)
14. [Introduction to Domain-centric Architectures \(clean, hexagonal, ...\) | by Luís Soares | CodeX | Medium](#)
15. [The Clean Architecture—Beginner's Guide | by Bharath | Better Programming](#)
16. [What is a JWT? Understanding JSON Web Tokens \(supertokens.com\)](#)
17. [Git - Book \(git-scm.com\)](#)
18. [GitHub — Вікіпедія \(wikipedia.org\)](#)
19. [Git — Вікіпедія \(wikipedia.org\)](#)
20. [MariaDB — Вікіпедія \(wikipedia.org\)](#)
21. [bcrypt - Wikipedia](#)
22. [bcryptjs - npm \(npmjs.com\)](#)

23. [Концептуальна модель — Вікіпедія \(wikipedia.org\)](https://uk.wikipedia.org)
24. [SQL - Wikipedia](https://uk.wikipedia.org)
25. [HTML5 - Wikipedia](https://uk.wikipedia.org)
26. [CSS - Wikipedia](https://uk.wikipedia.org)
27. [JavaScript - Wikipedia](https://uk.wikipedia.org)
28. [TypeScript - Wikipedia](https://uk.wikipedia.org)
29. [React](https://react.dev)
30. [Vite Docs](https://vitejs.dev)
31. [Documentation | NestJS - A progressive Node.js framework](https://docs.nestjs.com)
32. [Sequelize | Feature-rich ORM for modern TypeScript & JavaScript](https://sequelize.org)
33. [Node.js — About Node.js® \(nodejs.org\)](https://nodejs.org)
34. [gigacloud](https://gigacloud.io)
35. [КІБЕРБЕЗПЕКА В ХМАРНИХ СЕРВІСАХ ЗА ДОПОМОГОЮ ЗАСТОСУВАННЯ КРИПТОГРАФІЧНИХ МЕТОДІВ](#)
36. [АНАЛІЗ МЕХАНІЗМІВ ЗАХИСТУ ДАНИХ В ХМАРНИХ СЕРЕДОВИЩАХ](#)
37. [ЕКОЛОГІЯ ТА ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА](#)
[Методичні вказівки](#)
38. [ЕКОЛОГІЧНА БЕЗПЕКА В СИСТЕМІ МЕНЕДЖМЕНТУ ПІДПРИЄМСТВА](#)
39. [Екологічна безпека](#)
40. [Екологічна безпека та охорона довкілля](#)
41. [Методичні вказівки до виконання розділу «Охорона праці та захист навколишнього середовища»](#)
42. [ЕКОЛОГІЧНА БЕЗПЕКА](#)
43. [МЕТОДИЧНІ ВКАЗІВКИ до виконання розділу «Охорона праці та захист навколишнього середовища»](#)
44. [ЕКОЛОГІЧНА БЕЗПЕКА ТА ПРИРОДОКОРИСТУВАННЯ](#)
45. [Core - Docs](#)
46. [S3 Docs](#)

- 47.[Core](#)
- 48.[Системи зберігання даних: Як обрати оптимальне рішення для вашого бізнесу](#)
- 49.[Хмарне сховище даних De Novo](#)
- 50.[вимоги до виконання магістерської дисертації](#)

Додаток А – Інструкція користувачу

Необхідно перейти на сторінку веб-додатку, де користувача буде зустрічати вікно реєстрації, в якому потрібно ввести свої дані, або, якщо користувач вже має власний аккаунт, перейти у вікно “Log In”.

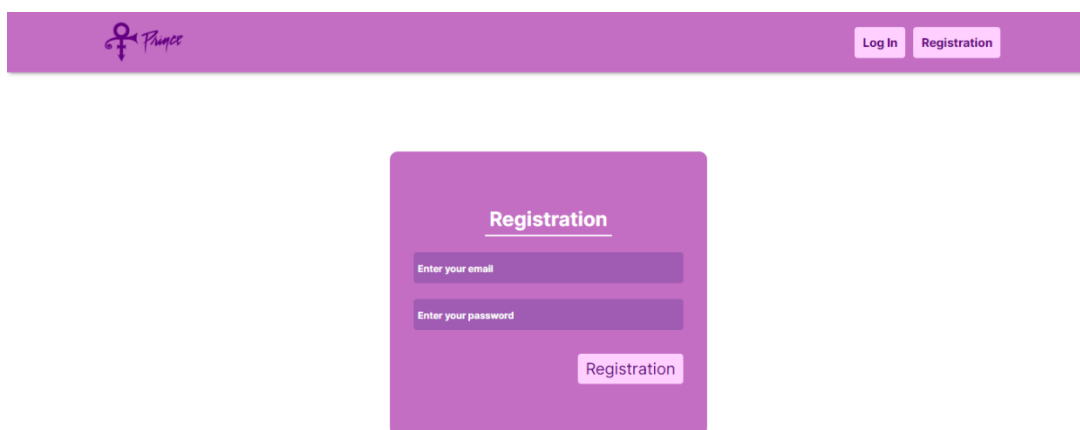
The image shows a web application interface with a purple header bar. On the left of the header is a logo consisting of a stylized female symbol and the word "Purple". On the right of the header are two buttons: "Log In" and "Registration". Below the header, centered on the page, is a white registration form with a purple border. The form has a title "Registration" at the top. It contains two input fields: "Enter your email" and "Enter your password". Below these fields is a purple button labeled "Registration".

Рисунок А1 – вікно реєстрації

1. Після авторизації користувач переадресовується на головну сторінку, де його зустрічає автоматично створена папка «Home», щоб створити нову папку, потрібно в лівому куті натиснути на «New Folder», потім увести назву і натиснути «✓»

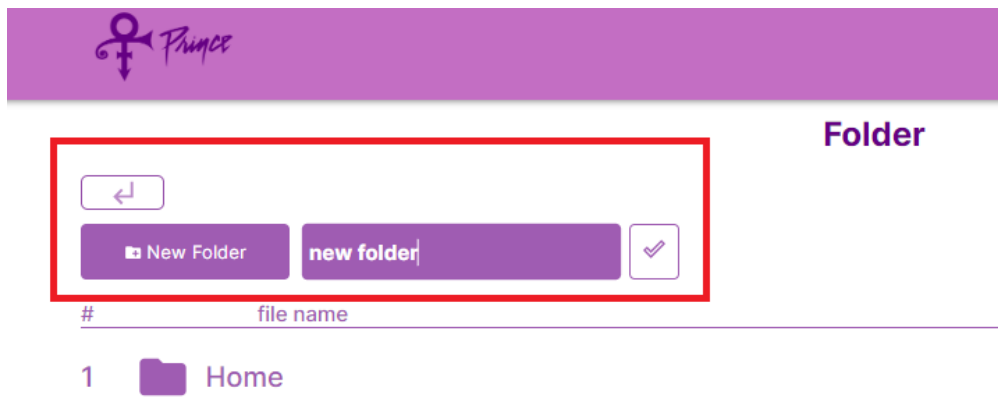


Рисунок А2 – створення папок

Для завантаження файлу з пристрою користувача існує два способи:

1. Натиснути на «Add your File» в правій частині додатку та обрати необхідний файл;
2. Перетягнути файл у вікно додатку.

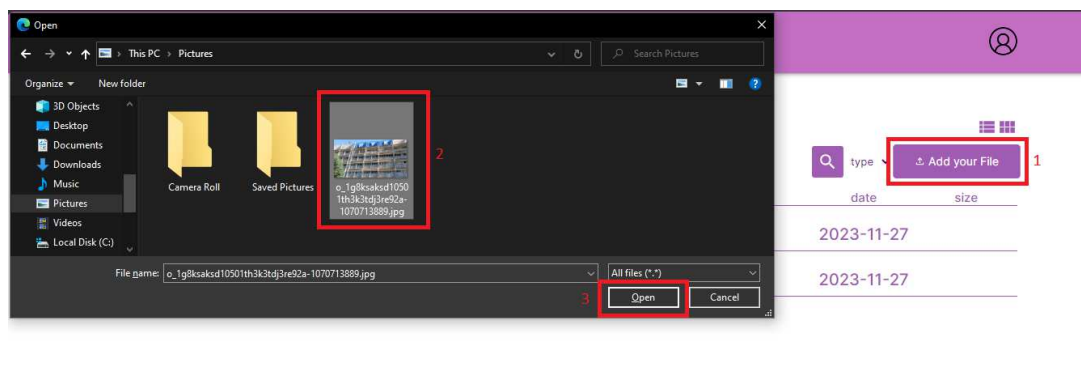


Рисунок А3 – завантаження файлу першим способом

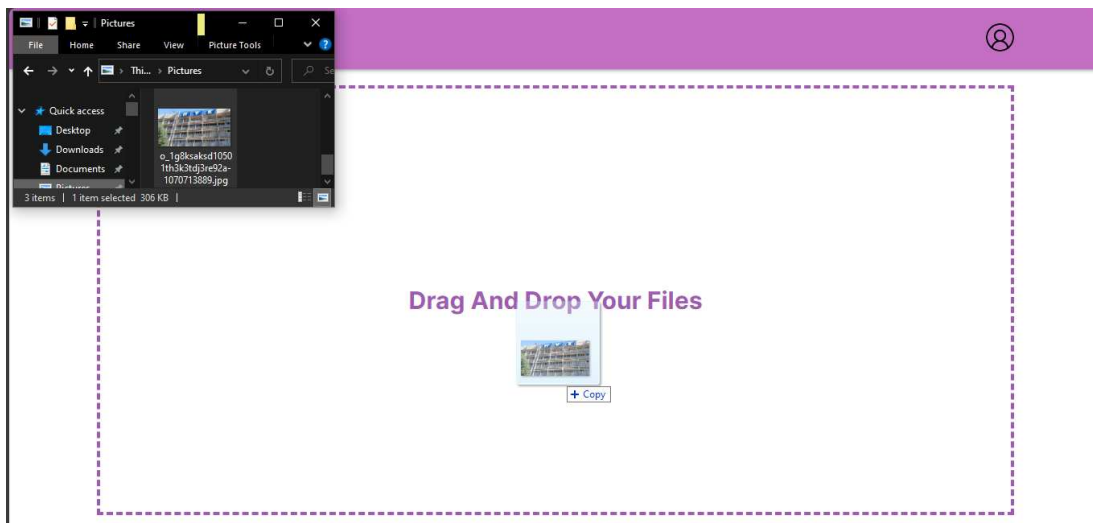


Рисунок А4 – завантаження файлу другим способом

Щоб змінити відображення списку файлів з вертикального списку на горизонтальні іконки або навпаки потрібно натиснути на одну з іконок, що знаходяться над кнопкою «Add your Files»

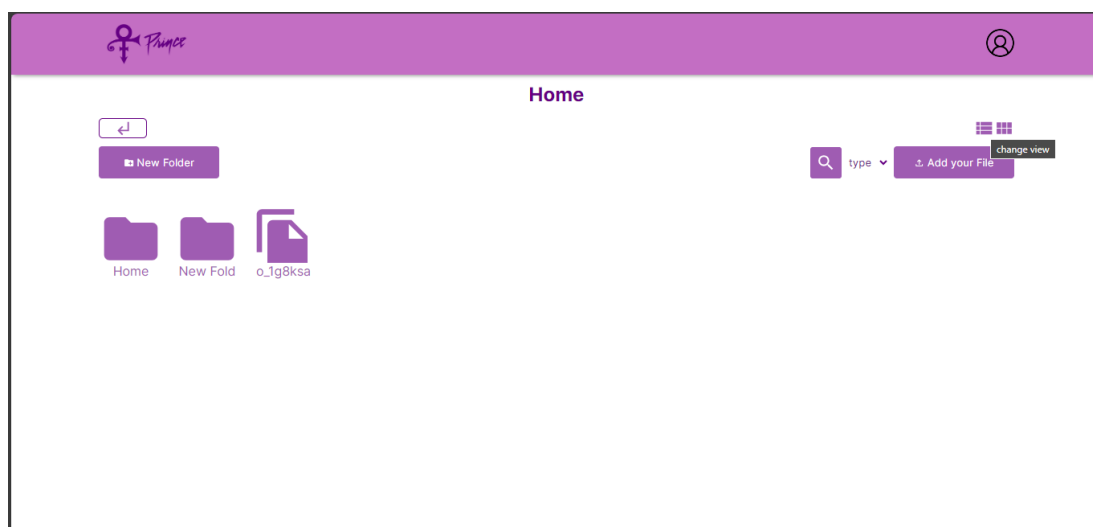


Рисунок А5 – зміна виду папок

Додаток Б – Текст програми

DiskPage.tsx:

```
import React from 'react';
import FileList from './FileList/ListView/FileList';
import DiskControl from './DiskControl/DiskControl';
import { getFiles, uploadFile } from '../../utils/api/fileApi';
import { useAppDispatch, useAppSelector } from '../../hooks/useRedux';
import './disk.scss';
import Uploader from './Uploader/Uploader';
import { hideCreateFolder } from '../../store/reducers/appSlice';
import { VIEW_MODE } from '../../types/types';
import FileModule from './FileList/ModuleView/FileModuleView';
import { setFolderName } from '../../store/reducers/fileSlice';

const Disk = () => {
  const [dragEnter, setDragEnter] = React.useState(false)
  const { currentDir, folderName, files } = useAppSelector(state => state.file)
  const { sortBy } = useAppSelector(state => state.app)
  const { isCreateFolder, viewMode } = useAppSelector(state => state.app)
  const dispatch = useAppDispatch()

  React.useEffect(() => {
    dispatch(getFiles(currentDir, sortBy))
    dispatch(setFolderName(folderName || 'Folder'))
  }, [currentDir, sortBy])

  const onDragEnter = (e: React.DragEvent<HTMLInputElement>) => {
    e.preventDefault()
    e.stopPropagation()
    setDragEnter(true)
  }

  const onDragLeave = (e: React.DragEvent<HTMLInputElement>) => {
    e.preventDefault()
    e.stopPropagation()
    setDragEnter(false)
  }
}
```

```

const onDrop = (e: React.DragEvent<HTMLInputElement>) => {
  e.stopPropagation()
  e.preventDefault()
  const uploadedFiles = Array.from(e.dataTransfer.files)
  uploadedFiles.forEach(file => dispatch(uploadFile(file, currentDir))
    .then(() => setDragEnter(false))
    .catch(e => alert(e.response.data.message))
  )
}

const onHideCreating = () => {
  if (isCreateFolder) {
    dispatch(hideCreateFolder())
  }
}

return (
  <>
    {!dragEnter ?
      <section
        className="diskPage"
        onDragEnter={onDragEnter}
        onDragLeave={onDragLeave}
        onDragOver={onDragEnter}
        onClick={onHideCreating}
      >
        <h1 className="diskPage__title" key={folderName}>{folderName ||
'Folder'}</h1>
        <DiskControl />
        {files?.length ?
          <>
            {viewMode === VIEW_MODE.LIST ?
              <FileList />
              :
              <FileModule />
            }
          </>
        }
      </>
    }
  )

```

```

        <span>There is no files</span>
      }
      <Uploader />
    </section>
    :
    <div className='dragSection' onDragEnter={onDragEnter}
onDragLeave={onDragLeave} onDrop={onDrop} onDragOver={onDragEnter}>Drag And Drop
Your Files</div>
  }
</>
)
}

export default Disk

```

auth.service.ts:

```

import { JwtService } from "@nestjs/jwt"
import { Injectable, HttpException, HttpStatus } from "@nestjs/common";
import { UsersService } from 'src/users/users.service';
import { InjectModel } from "@nestjs/sequelize";
import { File } from "src/files/models/files.model";
import { User } from "src/users/models/users.model";
import { CreateUserDto } from "src/users/dto/create-user.dto";
import * as bcrypt from 'bcryptjs';
import { FilesService } from "src/files/files.service";

@Injectable()
export class AuthService {
  constructor(
    private userService: UsersService,
    private jwtService: JwtService,
    private fileService: FilesService,
    @InjectModel(File) private fileModel: typeof File,
    @InjectModel(User) private userModel: typeof User
  ) {}

```

```

    async registration(dto: CreateUserDto) {
      try {
        const candidate = await this.userModel.findOne({where: {email:
dto.email}}})
        if (candidate) {
          throw new HttpException('User already exists',
HttpStatus.BAD_REQUEST)
        }
        const hashedPassword = await bcrypt.hash(dto.password, 8)
        const user = await this.userService.createUser(dto.email,
hashedPassword )

        if (user) {
          await this.fileService.createDir({name: "Home"}, user.id)
        }
        const token = await this.generateToken(user)
        return {
          token,
          user
        }
      } catch (e) {
        console.log(e)
        throw new HttpException('Registration error',
HttpStatus.BAD_REQUEST)
      }
    }

    async login(dto: CreateUserDto) {
      try {
        const user = await this.userModel.findOne({ where: { email:
dto.email } });
        if (!user) {
          throw new HttpException('User Not Found', HttpStatus.NOT_FOUND)
        }
        const isValidPassword = await bcrypt.compare(dto.password,
user.password)
        if (!isValidPassword) {
          throw new HttpException('Password is not Valid',
HttpStatus.BAD_REQUEST)

```



```

    }

    const token = await this.generateToken(user)

    return {
        token,
        user
    }
} catch (e) {
    throw new HttpException('Internal Server Error',
HttpStatus.INTERNAL_SERVER_ERROR);
}
}

async auth(userId: number) {
    const user = await this.userService.getById(userId)
    const token = await this.generateToken(user)

    return {
        token,
        user
    }
}

private generateToken(user: User) {
    const payload = { id: user.id }
    const token = this.jwtService.signAsync(payload)
    return token
}
}

```

Повний код програми можна переглянути за посиланням:

[nebykov/web_cw_pcd: Course work in the discipline: web technologies \(github.com\)](https://github.com/nebykov/web_cw_pcd)