МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА/ПРОЄКТ
на здобуття освітнього ступеня бакалавра

на тему: **«РОЗРОБКА ХМАРНОГО ФАЙЛОВОГО ХОСТИНГУ ДЛЯ ОРГАНІЗОВАНОГО ЗБЕРІГАННЯ ДАНИХ»**

Виконав: студент 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Костюшко Олександр Анатолійович

Керівник: старший викладач кафедри ЕММІТ,
Клебан Юрій Вікторович

Рецензент: Front-end Developer "DOODLE", LLC
Місай Володимир Віталійович,

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ
Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)
Протокол № 11 від 18 травня 2023 р.

Острог, 2023

Міністерство освіти і науки України
Національний університет «Острозька академія»

Факультет: *економічний*
Кафедра: *економіко-математичного моделювання та інформаційних технологій*
Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри економіко-математичного моделювання
та інформаційних технологій

_____ Ольга КРИВИЦЬКА
«____» _____20__ р.

З А В Д А Н Н Я
на кваліфікаційну роботу/проєкт студента

Костюшка Олександра Анатолійовича

1. Тема роботи "Розробка хмарного файлового хостингу для організованого зберігання даних"

керівник проєкту Клебан Юрій Вікторович, старший викладач кафедри ЕММІТ.

Затверджено наказом ректора НаУОА від 31 жовтня 2022 року №77.

2. Термін здачі студентом закінченої роботи/проєкту: 31 травня 2023 року.

3. Вихідні дані до роботи/проєкту: у своїй роботі я використовував такий стек технологій: React, Express, MongoDB та Node.js.

4. Перелік завдань, які належить виконати: розробити головну сторінку хмарного сховища, яка містить форму для авторизації та реєстрації, реалізувати функцію зміни пароля, можливість створювати папки на диску та завантажувати файли з персонального ПК, вивести інформацію про використане місце на диску, відокремити файли та папки один від одного, реалізувати перегляд відео та фото на диску, маніпуляції над файлами мають містити наступні функції: видалення, завантаження, перейменування, поширення файла шляхом генерації посилання як на одноразове завантаження так і на багаторазове, вивести всю інформацію про файл: тип, дата створення, назва, місце розміщення на диску, реалізувати пошук, сортування, зміна вигляду структури папок, файлів робочої області.

5. Перелік графічного матеріалу: рисунки, таблиці.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Клебан Ю. В.	01.12.2022	01.12.2022
2	Клебан Ю. В.	01.12.2022	01.12.2022
3	Клебан Ю. В.	01.12.2022	01.12.2022

7. Дата видачі завдання: 01.12.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1.	Затвердження теми роботи/проєкту	до 31.10.22.	
2.	Постановка технічного завдання	до 01.12.22	
3.	Верстка головної сторінки продукту, розбиття контенту на блоки	до 25.02.23	
4.	Робота над серверною частиною проекта, додавання та розробка нового функціоналу	до 14.04.23	
5.	Верстка клієнтської частини відповідно до розробленого функціоналу на серверній, налаштування screen media під мобільну версію	до 01.05.23	
6.	Реалізація функції доступу до файлів шляхом створення одноразового посилання або постійного	до 08.05.23	
7.	Вичитування останньої версії продукту, тестування, внесення правок	до 12.05.23	
8.	Тестування системи	до 13.05.23	
9.	Попередній захист кваліфікаційної роботи/проєкту	до 18.05.2023	
10.	Здача кваліфікаційної роботи/проєкту на кафедру	до 31.05.2023	

Студент: _____ Олександр КОСТЮШКО

Керівник кваліфікаційної роботи: _____ Юрій КЛЕБАН

АНОТАЦІЯ
кваліфікаційної роботи/проєкту
на здобуття освітнього ступеня бакалавра

Тема: *Розробка хмарного файлового хостингу для організованого зберігання даних*

Автор: *Костюшко Олександр Анатолійович*

Науковий керівник: *Клебан Юрій Вікторович, старший викладач кафедри ЕММІТ.*

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: *71 с., 25 рис., 9 табл., 25 джерел.*

Ключові слова: *хмарний файловий хостинг, організоване зберігання даних, клієнт-серверна архітектура, шифрування даних, безпека даних.*

Короткий зміст праці:

Метою моєї кваліфікаційної роботи/проєкту було створення хмарного файлового хостингу для організованого зберігання даних з урахуванням проблем безпеки. З метою забезпечення зручного доступу до даних користувачів, я використовував клієнт-серверну архітектуру, що дозволяє зберігати та отримувати доступ до файлів з будь-якого пристрою у будь-який час.

Для забезпечення безпеки даних реалізовано механізм шифрування, що запобігає несанкціонованому доступу. Також була використана система контролю версій Git та платформа розміщення коду Github, що дозволяють ефективно відстежувати та керувати змінами в проекті.

Для розробки моєї системи хмарного зберігання даних я використовував середовище розробки WebStorm, яке забезпечує зручний та продуктивний процес написання коду.

Зазначений стек технологій, що був використаний у проекті, включає React, Express, Node.js та MongoDB. Цей стек дозволив реалізувати функціональність хмарного файлового хостингу, забезпечуючи швидкість, масштабованість та надійність системи.

У результаті моєї роботи була розроблена ефективна та безпечна система зберігання та обміну даними в хмарі, що задовольняє вимоги щодо зручності використання та безпеки інформації.

The goal of my thesis/project was to create cloud file hosting for organized data storage with security issues in mind. In order to provide convenient access to user data, I used a client-server architecture that allows storing and accessing files from any device at any time.

To ensure data security, an encryption mechanism is implemented to prevent unauthorized access. The Git version control system and the Github code hosting platform were also used, allowing for efficient tracking and management of changes in the project.

To develop my cloud storage system, I used the WebStorm development environment, which provides a convenient and productive process for writing code.

The specified technology stack used in the project includes React, Express, Node.js and MongoDB. This stack made it possible to implement the functionality of cloud file hosting, ensuring the speed, scalability and reliability of the system.

As a result of my work, an efficient and secure system of data storage and exchange in the cloud was developed, which meets the requirements for ease of use and information security.

ЗМІСТ

ВСТУП

Розвиток будь-якої країни залежить не лише від природних, земельних, біологічних, а й насамперед від інформаційних ресурсів. Особливий інтерес представляють ресурси, пов'язані із застосуванням хмарних технологій (відомих як хмарні сервіси) в освіті, адже теперішніх студентів важко зацікавити звичайними методами навчання та задовольнити індивідуальні запити студентів стає все важче. У порівнянні з традиційним підходом, хмарні сервіси надають можливість управляти більшими інфраструктурами, обслуговувати різні групи користувачів в межах однієї хмари. Користувачу хмарних сервісів немає необхідності турбуватися про інфраструктуру, яка забезпечує працездатність сервісів, що надаються всі задачі по налаштуванню, усуненню несправностей бере на себе сервіс-провайдер [22].

Тенденції розвитку високих технологій зумовлюють зростання їх ролі у розвитку людства. Тому необхідно модернізувати інформаційно-освітнє та наукове середовище навчального закладу та привести його у відповідність до сучасного рівня розвитку науки, технологій і виробництва. Саме хмарні технології, які є нині передовими технологіями інформаційного суспільства, можуть відіграти роль провідного інструменту інформатизації освіти [20].

Хмарні технології – це зручне середовище для зберігання і обробки інформації. Робота в "хмарах" спрямована на зниження витрат і підвищення ефективності роботи закладів. Особливістю хмарних технологій є не прихильність до апаратної платформи і географічної території. Користувач може працювати з хмарними сервісами з будь-якої точки планети і з будь-якого пристрою, що має доступ в інтернет.

Хмарні сервіси є на даний момент повноцінним навчальним інструментом, що дозволяє навчальному закладу створити власний онлайн простір та формувати особисте освітнє середовище учнів та вчителів максимально ефективно.

Постійне використання нових засобів для навчання надає можливість не стояти на одному місці, привчає до нового стилю поведінки, легкому вирішенню будь-яких

ситуацій. Таке навчання допомагає зробити сам процес навчання відкритим і доступним для всіх [20].

Сьогодні ми спостерігаємо, як росте покоління візуалів, для яких головним джерелом інформації є зоровий ряд. Все це пояснює необхідність використання нових світових інформаційних розробок в освітній діяльності. Однією з інновацій в освітньому процесі, використовуваному в сучасному світі, є хмарні сервіси. Як показує досвід розвинених зарубіжних країн, впровадження в навчальний процес “хмарних технологій”є відмінним рішенням проблем комп'ютеризації освіти [20].

Крім того, хмарні технології дозволяють створювати спільні навчальні ресурси, які можуть бути використані вчителями та учнями з будь-якої точки світу. Наприклад, учитель може створити електронний курс, який буде доступний учням в будь-який час та з будь-якого місця. Крім того, хмарні технології дозволяють зберігати та обробляти великі обсяги даних, що є важливим у сучасному світі, де обробка даних відіграє ключову роль в бізнесі та науці.

Українські навчальні заклади також поступово переходять до використання хмарних технологій. Наприклад, в Україні було створено освітню хмару, яка дозволяє зберігати та обробляти навчальні матеріали та надавати до них доступ учням та вчителям з будь-якої точки країни. Крім того, у навчальних закладах використовуються хмарні сервіси для зберігання та обробки даних, створення електронних курсів та інтерактивних підручників, а також для взаємодії між вчителями та учнями.

У підсумку, хмарні технології відіграють важливу роль в інформатизації освіти та є необхідним інструментом для підвищення ефективності навчального процесу. Використання хмарних технологій дозволяє створювати спільні навчальні ресурси, зберігати та обробляти великі обсяги даних, а також знижувати витрати та підвищувати ефективність роботи навчальних закладів. Тому важливо продовжувати розвивати та вдосконалювати їх.

Метою кваліфікаційної роботи є аналіз теоретичних та практичних матеріалів для створення веб-сервісу для організованого зберігання даних, який в подальшому буде використовуватися для застосування користувачами відповідно до їх потреб.

Завдання:

1. Розкрити суть наступних понять: “хмара”, “хмарні обчислення” та “хмарні технології”.
2. Визначити ряд онлайн-сервісних середовищ, за допомогою яких можна удосконалити методику викладання дисциплін та охарактеризувати можливості використання кожної з них.
3. Розробити власний продукт для організованого збереження даних в інтернеті.

Об’єктом дослідження є структура веб-сервісу для організованого управління даними, яка включає в себе множину обробників подій, компоненти дизайну та структура візуалізації.

Предмет дослідження – розробка “хмарних технологій” з використання сучасних технологій та розробка власного програмного продукту.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ХМАРНИХ ТЕХНОЛОГІЙ ТА ЇХ ОСНОВНІ ПОНЯТТЯ

1.1. Постановка проблеми

Актуальність використання хмарних сервісів пояснюється широким застосуванням в освіті сучасних програм, які характеризуються своєю різноманітністю і простотою використання. Хмарні сервіси надають можливість автоматизувати подачу навчального матеріалу. По-перше, це автоматизація, як самого процесу створення, так і зберігання даних в будь-якому форматі. По-друге, це робота з практично необмеженим обсягом даних. По-третє, студенти набираються досвіду використання подібних технологій [20].

Популярність “хмарних сервісів” пов’язана з бурхливим розвитком Інтернету і супутніх технологій. На багатьох підприємствах люди працюють у віддаленому режимі, передаючи всю необхідну інформацію через інтернет. Хмарні технології надають споживачам рішення, повністю готові до роботи. Достатньо володіти будь-яким пристроєм, здатним з’єднатися з інтернетом, і можна отримати доступ до віддаленої бази, яка розташовується на віддаленому сервері. Хмарні технології відкривають нові можливості для підключення віддалених і сезонних працівників. Збільшуючи кількість персоналу, керівник може як підключати співробітників до хмарного сервісу так і відключати неактивних користувачів.

Розглянемо основні визначення:

Хмарні технології – це технології обробки даних, в яких комп’ютерні ресурси надаються Інтернет користувачеві як онлайн сервіс, одна велика концепція, що включає в себе багато різних понять, що надають послуги.

Хмарний сервіс – послуга надання хмарних ресурсів за допомогою технологій “хмарних обчислень”.

Хмарні обчислення (від англ. cloud computing) – це програмно-апаратне забезпечення, доступне користувачеві через Інтернет або локальну мережу у вигляді сервісу, що дозволяє використовувати зручний інтерфейс для віддаленого доступу до виділених ресурсів (обчислювальних ресурсів, програм і даних). Комп'ютер користувача виступає при цьому рядовим терміналом, підключеним до Мережі. Комп'ютери, які здійснюють cloud computing, називаються "обчислювальною хмарою". При цьому навантаження між комп'ютерами, що входять в "обчислювальну хмару", розподіляється автоматично.

Хмара – це розташування в Інтернеті, де можна зберігати різну інформацію, а потім легко отримувати до нього доступ на комп'ютері, телефоні чи іншому пристрої з підключенням до Інтернету.

В сучасному світі і в усіх сферах людської діяльності застосовуються новітні комп'ютерні програми, що ґрунтуються на використанні найновіших комп'ютеризованих інформаційно-комунікаційних технологій. Саме тому, серед людей навчальних закладів й інших сфер діяльності постає завдання забезпечити освітній процес якісними електронним засобами навчання, з використанням не лише комп'ютерів, а й інших сучасних пристроїв, які можна було б використовувати як під час занять, так і будучи поза межами навчального закладу. Використання такого навчального середовища, до якого забезпечується вільний доступ на основі інтернет ресурсів, значно підвищує інтерес студентів до навчання в цілому.

Реалізація всього вище переліченого можлива за умови використання сучасних хмарних сервісів. Актуальність впровадження новітніх технологій в освіту, зокрема використання хмарних сервісів, які надають нові способи навчання, формують уміння та навички до самостійного навчання, спільна взаємодія студентів та викладачів, отримання якісних знань, незалежно від місця знаходження студента.

Хмарні сервіси – це кардинально новий сервіс, використання якого дозволяє дистанційно використовувати засоби опрацювання і зберігання даних.

1.2. Історія хмарних технологій

Хмарні сервіси, що дозволяють перенести обчислювальні ресурси й дані на віддалені інтернет-сервери, в останні роки стали одним з основних трендів розвитку IT-технологій [23].

Концепція хмарних обчислень з'явилася ще в 1960 році, коли американський учений, фахівець з теорії ЕОМ Джон Маккарті (John McCarthy) висловив припущення, що коли-небудь комп'ютерні обчислення стануть надаватися подібно комунальним послугам (public utility). Розповсюдження мереж з високою потужністю, низька вартість комп'ютерів і пристроїв зберігання даних, а також широке впровадження віртуалізації, сервіс-орієнтованої архітектури привели до величезного зростання хмарних обчислень. Кінцеві користувачі можуть не перейматися роботою обладнання технологічної інфраструктури "в хмарі", яка їх підтримує. Аналогією обчислювальних "хмар" зі звичного життя можуть служити електростанції. Хоча домовласник може купити електрогенератор і піклуватися про його справність самостійно, більшість людей вважає за краще отримувати енергію від централізованих постачальників [23].

Майже всі сучасні характеристики хмарних обчислень, порівняння їх з електроенергетикою та використання приватних, публічних та громадських моделей були представлені Дугласом Паркхілом (Douglas Parkhill) в книзі "The Challenge of the Computer Utility", в 1966 році. Згідно інших джерел, хмарні обчислення беруть початок з 1950-х років, коли вчений Херб Грош (Herb Grosch) стверджував, що весь світ буде працювати на терміналах, якими керують близько 15 великих центрів обробки даних [23].

Сам термін "хмара" походить з телефонії, тому що телекомунікаційні компанії, які до 1990-х років пропонували в основному виділені схеми передачі "крапка-крапка", почали пропонувати віртуальні приватні мережі (VPN), з порівняною якістю обслуговування, але при набагато менших витратах. Перемикаючи трафік для оптимального використання каналів вони мали змогу ефективніше використовувати

мережу. Символ хмари був використаний для позначення розмежування між користувачем і постачальником [23].

Ключову роль в розвитку хмарних обчислень зіграв Amazon, модернізувавши свої центри обробки даних, які, як і більшість комп'ютерних мереж в один момент часу використовують лише 10% своєї потужності, заради забезпечення надійності при стрибку навантаження. Дізнавшись, що нова хмарна архітектура забезпечує значне внутрішнє підвищення ефективності, Amazon почав нові дослідження в галузі розвитку продуктів для забезпечення хмарних обчислень для зовнішніх клієнтів, і запустив Amazon Web Service (AWS) на основі розподілених обчислень в 2006 році [23].

На початку 2008 року Eucalyptus став першою API-сумісною платформою з відкритим кодом для розгортання приватної хмари. На початку 2008 року OpenNebula став першим проектом з відкритим кодом для розгортання приватних і гібридних хмар [23].

Але все таки датою відліку сучасної історії cloud computing став 2006 рік, коли компанія Amazon, яка вже на той момент була однією з найбільших, презентувала свою інфраструктуру веб-сервісів, яка була здатна забезпечити користувачеві не лише хостинг, а й надати віддалені обчислювальні потужності клієнтові. Новинку сприйняли і ухвалили такі гіганти, як Google, Sun і IBM, а в 2008 році про свій інтерес у цій галузі заявила корпорація Microsoft [23].

Хмарні технології пропонують масштабовану інфраструктуру і програмні засоби без прямої прив’язки до фізичних машин, при цьому економлячи витрати, серверні потужності і енергоспоживання під час простоювання. Хмарні технології – це можливість безлічі фізичних серверів бути єдиним обчислювальним середовищем. В цілому, сервіси хмарних обчислень є додатками, доступ до яких забезпечується через Інтернет за допомогою браузера або інших мережевих застосувань, наприклад, FTP-клієнта. Головна відмінність від звичного методу роботи з ПЗ полягає в тому, що користувач використовує не ресурси свого комп’ютера, або сервера своєї локальної мережі, а потужності, які надаються йому як Інтернет-послуга. При цьому користувач має повний доступ до власних даних і можливість роботи з ними з будь-якої точки

світу і з будь-якого пристрою, але не обтяжений управлінням операційною системою, програмною базою, обчислювальними потужностями, за допомогою яких ця робота відбувається. Зберігання в хмарі не лише даних, а й додатків змінює обчислювальну парадигму в бік традиційної клієнт-серверної моделі, при якій на стороні користувача зберігається мінімально необхідна функціональність.

Таким чином, необхідність встановлювати необхідні оновлення програмного забезпечення, проводити перевірку на віруси й інше обслуговування покладається на провайдера хмарного сервісу. Це також означає, що загальний доступ, управління версіями, спільне редагування стають набагато простішими, ніж коли додатки і дані розміщені на призначених для користувача комп'ютерах.

1.3. Суть хмарних технологій

Суть хмарних технологій, таким чином, полягає в перенесенні обробки даних з персональних комп'ютерів і робочих станцій на сервери всесвітньої мережі. В області комп'ютерного моделювання це означає розгортання програмних комплексів на ресурсах Інтернет. Користувач стає не покупцем обчислювальних програм і комплексів, а їх орендарем, якому надаються різноманітні послуги. Форма купівлі–продажу товару з відчуженням прав власності від продавця до покупця змінюється на форму оренди, в даному випадку – продажу не продукту, а послуг з його використання клієнтом без зміни власника продукту. При цьому забезпечена повна відповідність виробничих потужностей інфраструктури фактичним потребам користувача [24].

Хоча термін "хмарні технології" є сталим, в українській мові він має інше значення, ніж оригінал. "Cloud" окрім хмари має й інше значення – розсіяний; власне значення "розсіяний" і мається на увазі в англомовній термінології.

Хмарна обробка даних як концепція включає поняття:

1) інфраструктура як послуга;
2) платформа як послуга;
3) програмне забезпечення як послуга;

4) дані як послуга;
5) робоче місце як послуга;
6) інші технологічні тенденції, загальною рисою яких є впевненість, що мережа Інтернет у змозі задовольнити потреби користувачів в обробці даних.

1.4. Аналіз продуктів конкурентів на ринку

Суть хмарних технологій полягає в наданні користувачам віддаленого доступу до послуг, обчислювальних ресурсів і додатків (включаючи операційні системи і інфраструктуру) через Internet.

Всі три сервіси доступні у вигляді веб-додатку, яке може працювати офлайн, а також у мобільних додатках.

1. Google диск - Зручний варіант для зберігання всіляких файлів, якими можна ділитися з іншими користувачами. Гугл Диск – один з найбільш захищених і зручних сервісів [21].

Таблиця 1.1

SWOT аналіз сервісу для збереження даних від компанії Google

Сильні сторони	Слабкі сторони
1) Можливість поділитися файлами та папками. 2) Історія зміни файлу. 3) Має рівні доступу, перегляд, коментування, редагування.	1) Не можна синхронізувати папки за межами каталогу Google Drive.
Можливості	Загрози
1) Можливість доступу до документів із будь-яких пристроїв.	1) Кіберзлочинність. 2) Проекти з відкритим кодом. 3) Піратство.

Джерело: розроблено автором

2. Microsoft OneDrive - віртуальне сховище і файлообмінник, запущений ще в 2007 році, підходить тим, хто користується сервісами Microsoft [16].

Таблиця 1.2

SWOT аналіз сервісу для збереження даних від компанії Microsoft

Сильні сторони	Слабкі сторони
1) Доступ звідусіль. 2) Резервне копіювання та захист. 3) Спільний доступ. 4) Провідна компанія-розробник програмного забезпечення. 5) Найбільша в світі компанія з домінуючою часткою ринку.	1) Надмірний вплив на ринок. 2) Відсутність інновацій. 3) Безкоштовного місця лише на 5 гб.
Можливості	Загрози
1) Зростання хмарного бізнесу. 2) Інновації та штучний інтелект.	1) Кіберзлочинність. 2) Проекти з відкритим кодом. 3) Піратство.

Джерело: розроблено автором

3. pCloud - створений швейцарською компанією хмарний сервіс популярний вже за рахунок країни-творця: вона відома своїм трепетним ставленням до конфіденційності. Також сховище потрапило в 10-ку кращих [15].

Таблиця 1.3

SWOT аналіз сервісу для збереження даних від компанії pCloud

Сильні сторони	Слабкі сторони
1) Немає обмежень на розмір файлу в межах доступного простору. 2) 2-рівнева захист: через TLS/SSL протокол інформація йде на сервери і копіюється мінімум на 3 з них.	1) Лімітована швидкість завантаження файлу. 2) Лише для користувачів IOS.
Можливості	Загрози
1) Зростання хмарного бізнесу.	1) Кіберзлочинність. 2) Проекти з відкритим кодом.

	3) Піратство.

Джерело: розроблено автором

1.5. SWOT аналіз власного продукту

Cloud Storage - проєкт, який розроблений з використанням сучасних технологій, який дозволяє користувачам зберігати свої дані на серверах у “хмарі” та в подальшому передавати їх іншим користувачам в Інтернеті. Хмарне середовище, включає в себе: збереження файлів на сервері, створення папок, скачування файлів з сервера, видалення, пошук та сортування, а саме головне сервіс повністю безкоштовний.

Таблиця 1.4

SWOT аналіз власного продукту

Сильні сторони	Слабкі сторони
1) Сервіс повністю безкоштовний. 2) Розроблений з використанням сучасних технологій. 3) Простота використання.	1) Велика конкурентність на ринку. 2) Низька впізнаваність сервісу. 3) Невеликий обсяг пам’яті для зберігання. 4) Відсутня можливість перегляду вмісту файла.
Можливості	Загрози
1) Можливість доступу до документів із будь-яких пристроїв. 2) Розширення можливостей для користувача. 3) Реклама продукту.	1) Кіберзлочинність.

Джерело: розроблено автором

Впровадження хмарних технологій дозволяють вирішити ряд проблем та дають можливість створити віртуальні управлінські та навчальні структури, які забезпечать не тільки необмежений доступ до електронних освітніх ресурсів, а створять нові технології організації навчальної діяльності, комунікації тим закладам, де немає відповідних матеріально-технічних ресурсів.

РОЗДІЛ 2

ПРИКЛАДНА ЧАСТИНА, РОЗРОБКА ХМАРНОГО ФАЙЛОВОГО ХОСТИНГУ ДЛЯ ОРГАНІЗОВАНОГО ЗБЕРІГАННЯ ДАНИХ

2.1. Опис технологічного стеку

Технологічний стек — це набір мов програмування, фреймворків та програмного забезпечення, необхідних для розробки програми.

Оскільки веб-програми складаються з клієнтської та серверної частин, вимоги до їх функціональності змінюють мови програмування, фреймворки та програмне забезпечення, за допомогою яких буде вестися розробка. Іншими словами, вимоги до функціональності клієнтської та серверної елементів впливають на технологічний стек.

Клієнтська сторона – це видима частина веб-програми, з якою взаємодіють користувачі. Є 3 основні елементи розробки клієнтської частини будь-якого веб-додатку:

1) JavaScript — мова програмування, яка відповідає за інтерактивну частину веб-програми.
2) HTML — мова розмітки документів, яка потрібна для правильного відображення веб-програми в браузері.
3) CSS — формальна мова, яка потрібна для правильної стилізації веб-програми.

Якщо говорити про фреймворки, найчастіше для розробки клієнтської частини будь-якої веб-програми використовуються Bootstrap і React.js.

Серверна частина веб-застосунку — це те, що не бачить користувач, тому що вона знаходиться під клієнтською частиною. Для розробки серверної частини використовуються мови бекенд-програмування, бази даних, серверне середовище.

Стек технологій, використаний для розробки курсового проєкту:

1. ReactJS
2. GitHub
3. HTML
4. CSS
5. Figma
6. NPM
7. NodeJS
8. MongoDB
9. Express
10. WebStorm
11. Git

HTML

HTML — це мова тегів, засобами якої здійснюється розмічання веб-сторінок для мережі Інтернет. Браузери отримують HTML-документи з веб-сервера або з локальної пам'яті й передають документи в мультимедійні веб-сторінки. HTML описує структуру веб-сторінки семантично і початково підказки для відображення документа [10].

CSS

CSS — це спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду. Самі ж сторінки написані мовами розмітки даних. CSS є основною технологією всесвітньої павутини, поряд із HTML та JavaScript [9].

GitHub

GitHub — один з найбільших веб-сервісів для спільної розробки програмного забезпечення. Існують безкоштовні та платні тарифні плани користування сайтом. Базується на системі керування версіями Git і розроблений на Ruby on Rails і Erlang компанією GitHub [14].

Figma

Figma — векторний онлайн-сервіс розробки інтерфейсів та прототипування з можливістю організації спільної роботи, що розробляється однойменною компанією. Працює у двох форматах: у браузері та як клієнтський додаток на десктопі користувача. Зберігає онлайн-версії файлів, з якими працював користувач [17].

Npm

Npm - це менеджер пакунків для мови програмування JavaScript. Для середовища виконання Node.js це менеджер пакунків за замовчуванням. Включає в себе клієнт командного рядка, який також називається npm, а також онлайн-базу даних публічних та приватних пакунків, яка називається реєстром npm [19].

React.js

React.js – це відкрита JavaScript-бібліотека для створення інтерфейсів користувача, яка написана для вирішення проблеми часткового оновлення вмісту веб-сторінки, з якими програмісти зустрічаються при написанні та розробці односторінкових додатків. За допомогою React були написані всім відомі соціальні мережі як: Facebook та Instagram, також зараз активно використовують React: Netflix, Yahoo, Airbnb, Sony, Atlassian та інші. Також односторінкові застосунки, а саме SPA (Single Page Application) - це односторінковий веб-додаток, який завантажується на одну HTML-сторінку. Завдяки динамічному оновленню JavaScript, під час використання не потрібно довантажувати або перезавантажувати додаткові сторінки, оскільки це відбувається автоматично [8].

WebStorm

WebStorm - це інтегроване середовище для розробки на JavaScript та пов'язаних з ним технологіях. Як і інші IDE JetBrains, WebStorm дозволяє автоматизувати рутинну роботу та легко справлятися зі складними завданнями, роблячи розробку більш цікавою [20].

ExpressJS

Express - це мінімалістичний та гнучкий веб-фреймворк для програм Node.js, що надає широкий набір функцій для мобільних та веб-додатків.

NodeJS

NodeJS - це міжплатформне середовище виконання JavaScript з відкритим вихідним кодом , яке працює на JavaScript Engine і виконує код JavaScript поза веб-браузером , розробленим для створення масштабованих мережевих програм. Node.js дозволяє розробникам використовувати JavaScript для написання інструментів командного рядка та для сценаріїв на стороні сервера - виконання сценаріїв на стороні сервера для створення динамічного вмісту веб-сторінки перед тим, як сторінка надсилається у веб-браузер користувача.

Технологічний стек дуже сильно впливає на майбутнє будь-якої веб-програми. Він впливає на вартість його розробки, час розробки, масштабованість та багато інших нюансів.

Ваш вибір технологій залежить від призначення готового проекту. Тому спочатку необхідно ознайомитися з перевагами та недоліками всіх доступних технологій, а потім порадитись із досвідченими розробниками.

2.2. Опис архітектури рішення

Стек MERN дозволяє легко побудувати трирівневу архітектуру (інтерфейс, бек-енд, база даних), повністю використовуючи JavaScript і JSON (рис. 1.1).

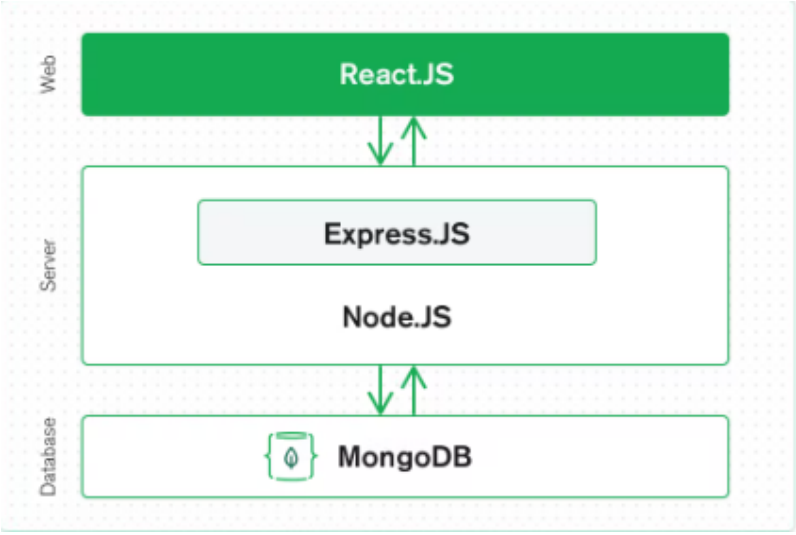

Рис. 1.1. 3-рівнева архітектура

Джерело: [25]

Інтерфейс React.js

Найвищим рівнем стеку MERN є React.js, декларативна структура JavaScript для створення динамічних клієнтських програм у HTML. React дозволяє створювати складні інтерфейси за допомогою простих компонентів, підключати їх до даних на сервері та відтворювати їх як HTML.

Сильна сторона React — це обробка інтерфейсів із збереженням стану, керованих даними, з мінімальною кількістю коду та мінімальними труднощами, а також має всі переваги, які ви очікуєте від сучасної веб-платформи: чудова підтримка форм, обробки помилок, подій, списків тощо [25].

Сервер Express.js і Node.js

Наступним рівнем є серверна структура Express.js, яка працює на сервері Node.js. Express.js називає себе «швидким, непереборним, мінімалістичним веб-фреймворком для Node.js», і це справді саме те, що він є. Express.js має потужні моделі для маршрутизації URL-адрес (зіставлення вхідної URL-адреси з функцією сервера) і обробки HTTP-запитів і відповідей [25].

Роблячи XML HTTP-запити (XHR) або GET або POST з інтерфейсу React.js, ви можете підключитися до функцій Express.js, які забезпечують роботу вашої програми. Ці функції, у свою чергу, використовують драйвери MongoDB Node.js, або через зворотні виклики, або за допомогою обіцянок, для доступу та оновлення даних у вашій базі даних MongoDB [25].

Рівень бази даних MongoDB

Якщо ваша програма зберігає будь-які дані (профілі користувачів, вміст, коментарі, завантаження, події тощо), тоді вам потрібна база даних, з якою так само легко працювати, як з React, Express та Node [25].

Тут на допомогу приходить MongoDB: документи JSON, створені у вашому інтерфейсі React.js, можна надіслати на сервер Express.js, де їх можна обробити та (якщо вони дійсні) зберегти безпосередньо в MongoDB для подальшого отримання. Знову ж таки, якщо ви будуєте в хмарі, вам захочеться подивитися на Atlas [25].

Чи є MERN комплексним рішенням?

Так, MERN — це повний стек, який дотримується традиційної трирівневої архітектурної моделі, включаючи рівень зовнішнього відображення (React.js), рівень додатків (Express.js та Node.js) і рівень бази даних (MongoDB) [25].

Чому було обрано саме цей стек?

Почнемо з MongoDB, бази даних документів у корені стеку MERN. MongoDB було розроблено для зберігання даних JSON (технічно вона використовує двійкову версію JSON під назвою BSON), і все, від інтерфейсу командного рядка до мови запитів (MQL або MongoDB Query Language) побудовано на JSON та JavaScript [25].

MongoDB надзвичайно добре працює з Node.js і неймовірно спрощує зберігання, обробку та представлення даних JSON на кожному рівні вашої програми. Для хмарних додатків MongoDB Atlas робить це ще простіше, надаючи вам автоматичне масштабування кластера MongoDB у хмарному провайдері за вашим вибором, за допомогою декількох натискань кнопки [25].

Express.js (працює на Node.js) і React.js роблять програму JavaScript/JSON MERN повним стеком. Express.js — це серверний фреймворк програми, який огортає HTTP-запити та відповіді та дозволяє легко зіставляти URL-адреси з функціями на стороні сервера. React.js — це зовнішній фреймворк JavaScript для побудови інтерактивних інтерфейсів користувача в HTML і зв’язку з віддаленим сервером [25].

Ця комбінація означає, що дані JSON природним чином переміщуються спереду назад, що робить їх швидким для створення та досить простим для налагодження. Крім того, вам потрібно знати лише одну мову програмування та структуру документа JSON, щоб зрозуміти всю систему [25].

MERN — це вибір для сучасних веб-розробників, які хочуть швидко рухатися, особливо для тих, хто має досвід React.js.

2.3. Опис реалізації веб-сервісу

Діаграма варіантів використання — це поведінкова діаграма UML, яка часто використовується для аналізу різних систем. Вони дозволяють візуалізувати різні типи ролей у системі та те, як ці ролі взаємодіють із системою.

На рисунку (рис. 1.2) зображена діаграма переходу станів для авторизації та реєстрації, проектування діаграм розроблено за допомогою сервісу [12].

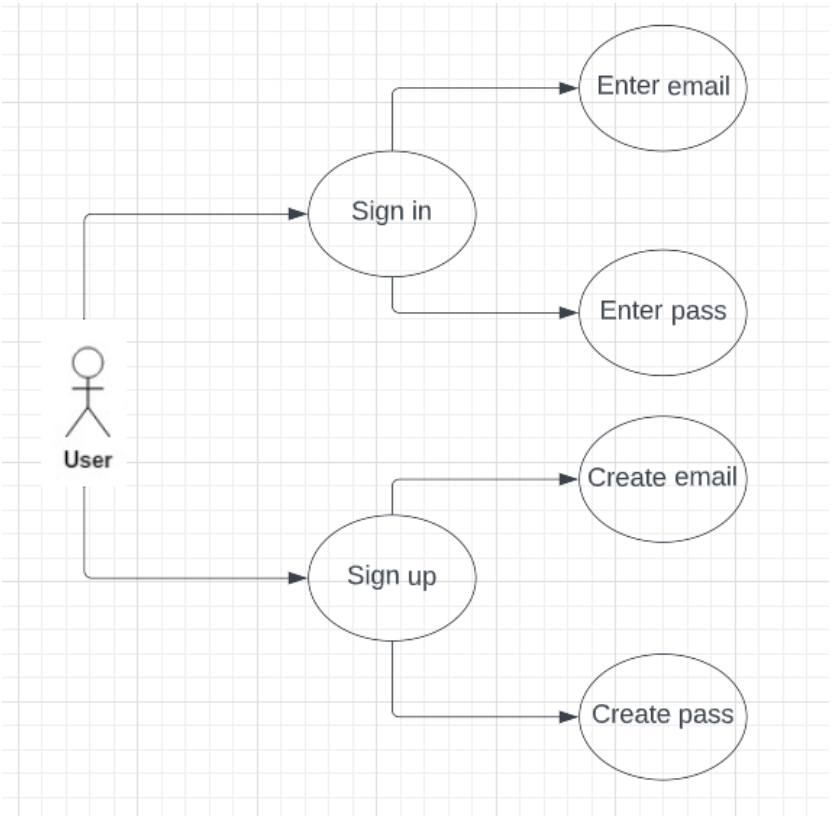

Рис. 1.2. Компонент авторизації та реєстрації

Джерело: розроблено автором

На рисунку (рис. 1.3) зображена діаграма переходу станів для організованого зберігання даних та їхнє відображення.

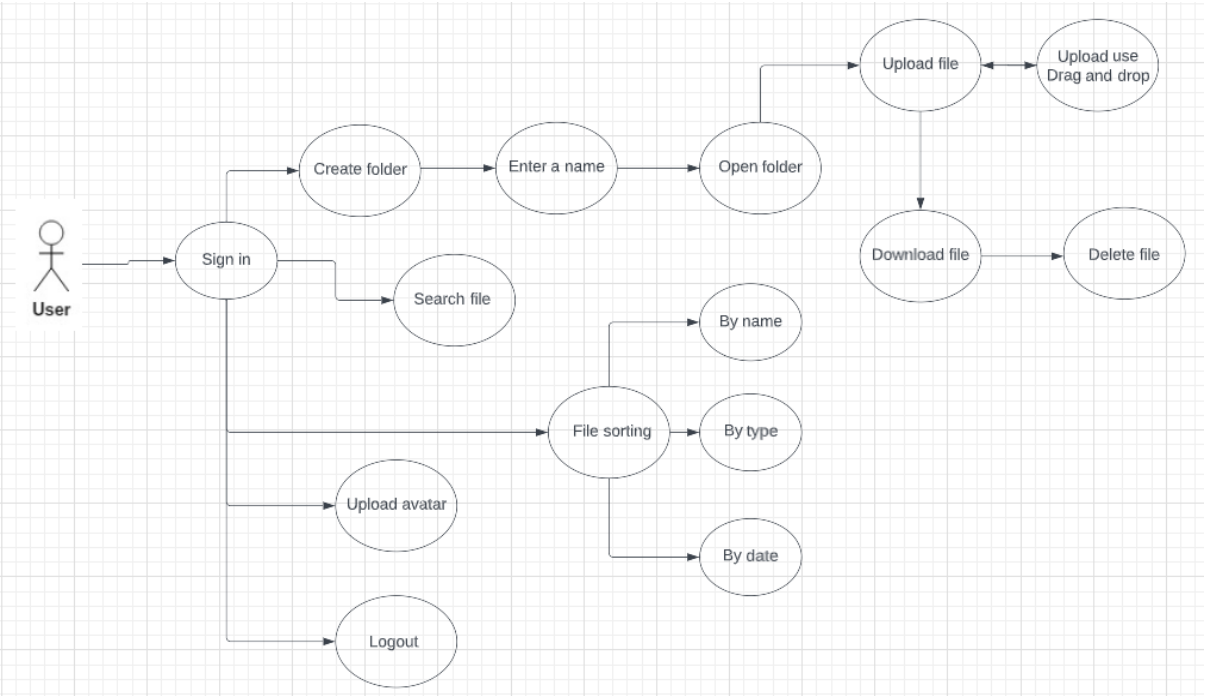

Рис. 1.3. Зберігання даних та їхнє відображення

Джерело: розроблено автором

Діаграма варіантів використання UML є основною формою вимог до системи/програмного забезпечення для нової недостатньо розробленої програми. Варіанти використання вказують на очікувану поведінку, а не на точний спосіб її здійснення.

Таблиця 2.1

Відправлення API запитів з Postman.

Метод	Ресурс	Опис
Post	http://localhost:5000/api/auth/registration	Перевірка реєстрації на сервісі.
Post	http://localhost:5000/api/	Перевірка авторизації за вже

	auth/login	зареєстрованим користувачем.
Post	http://localhost:5000/api/auth/logout	Вихід з системи.
Post	http://localhost:5000/api/files	Створення папки в авторизованого користувача.
Post	http://localhost:5000/api/files	Створення вкладеної папки через id наявної.
Get	http://localhost:5000/api/files	Перевірка отримання вже наявних папок.
Get	http://localhost:5000/api/files	Отримання вкладеної папки через id батьківської.
Post	http://localhost:5000/api/files/upload	Перевірка завантаження файла на сервер.
Post	http://localhost:5000/api/files/upload	Перевірка завантаження файла у вкладену папку.
Post	http://localhost:5000/api/files/avatar	Завантаження аватара.
Delete	http://localhost:5000/api/files/avatar	Видалення аватара.

Джерело: розроблено автором

Розробка форми для авторизації та реєстрації на сайті “Cloud Storage” (рис. 1.4).

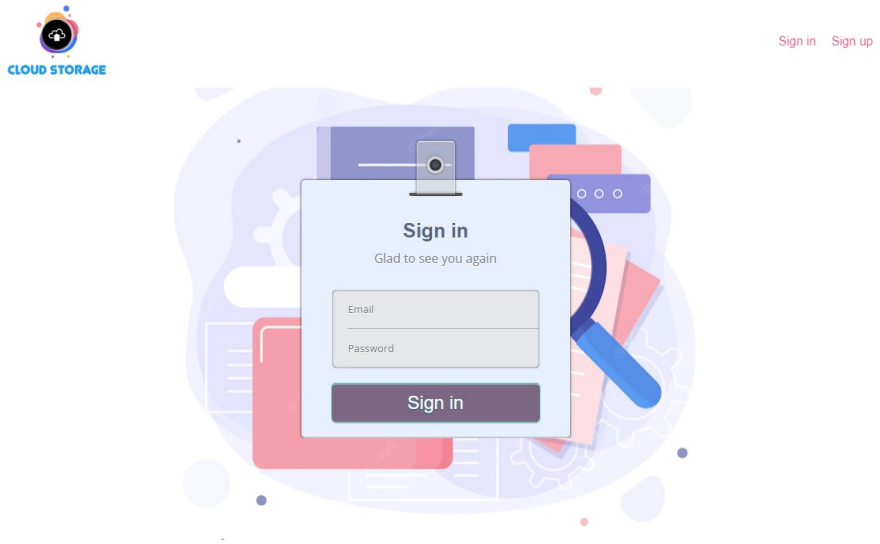

Рис. 1.4. Головне вікно сайту з авторизацією "Cloud Storage"

Джерело: розроблено автором

Розробка футера, це візуальний елемент, розташований в нижній частині сторінок. У його полі зазвичай розміщують посилання інші матеріали ресурсу, дублюють меню, розташовують іконки соцмереж (рис. 1.5).

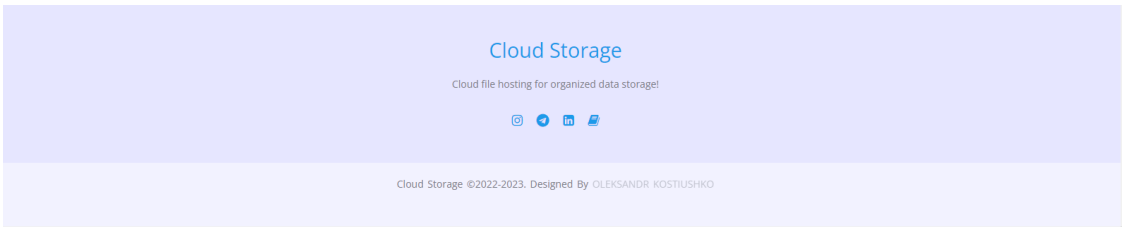

Рис. 1.5. Футер сайту "Cloud Storage"

Джерело: розроблено автором

Розробка сторінки для зберігання файлів, в якій розміщені три кнопки: назад, створити папку та кнопка завантаження файлів з персонального комп’ютера чи смартфона на хмарний сервіс. Кнопка “*Back*” повертає вас на один крок назад. Кнопка “*Create a folder*” при її натисканні відкривається модальне вікно в якому задається користувачем назва папки та після натискання створити вона з'являється в робочій області, після чого в неї вже можна завантажувати файли. Остання кнопка, це завантаження файлів з персонального комп’ютера, після її натискання відкривається вікно в якому обирається файли, додати їх можна як по одиноко так і декілька, також була реалізована функція додавання файлів шляхом перетягування на робочу область сервісу.

З правої частини розміщений компонент “випадаюче меню” в якій можна обрати тип сортування файлів.

Останні дві кнопки, вони відповідають за розположення файлів в форматі “List”, та “Plate”.

Самі файли після завантаження можна скачати та видалити з сервера, шляхом наведення на файл з'являються дві відповідні кнопки (кнопки для видалення, завантаження та зміни стилю відображення файлів взято з сервісу [13].

Також присутня інформація про файли, це дата завантаження, назва файла або папки і розмір. За підрахунок розміру в проекті відповідає файл “sizeFormat”, в якому розміщенні формули для підрахунку (рис. 1.6).

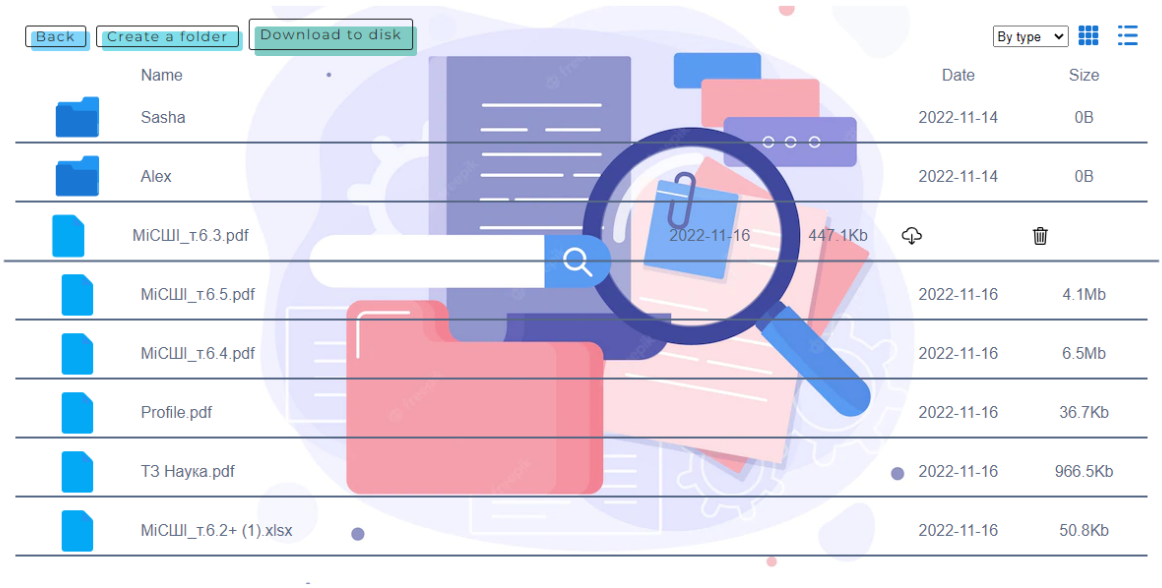

Рис. 1.6. Вигляд особистого кабінету зі списком файлів та папок на сайті "Cloud Storage"

Джерело: розроблено автором

Верстка хедера (або ж сайт бару) в якому розміщений створений логотип продукту, пошук по файлам, папкам, кнопка виходу на головну сторінку з авторизацією та аватар користувача який можна завантажити з пристрою (рис. 1.7).

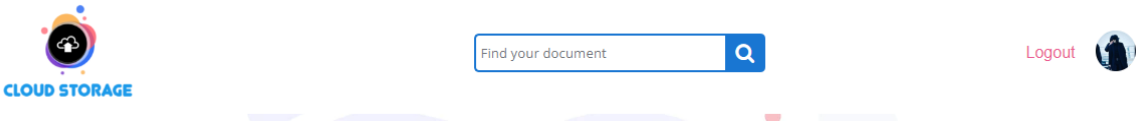

Рис. 1.7. Розробка шапки сайту "Cloud Storage"

Джерело: розроблено автором

Також було реалізовано спливаюче вікно при завантаженні файлу(ів) на диск, з інформацією які файли обрані та відсоток їх завантаження (рис. 1.8).

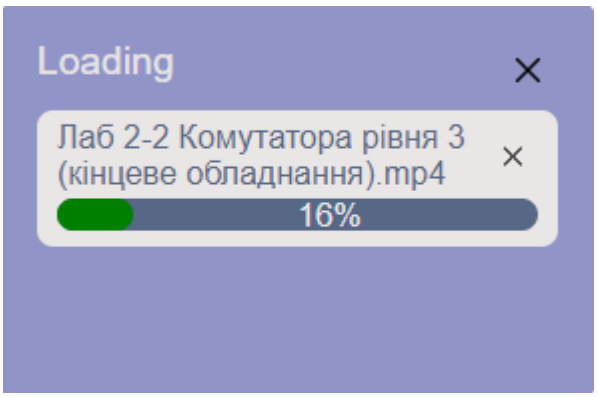

Рис. 1.8. Відсоток завантаження файлу на сайт

Джерело: розроблено автором

На рисунку зображений вигляд сайту по закінченню реалізації всіх запланованих пунктів в технічному завданні (рис. 1.9).

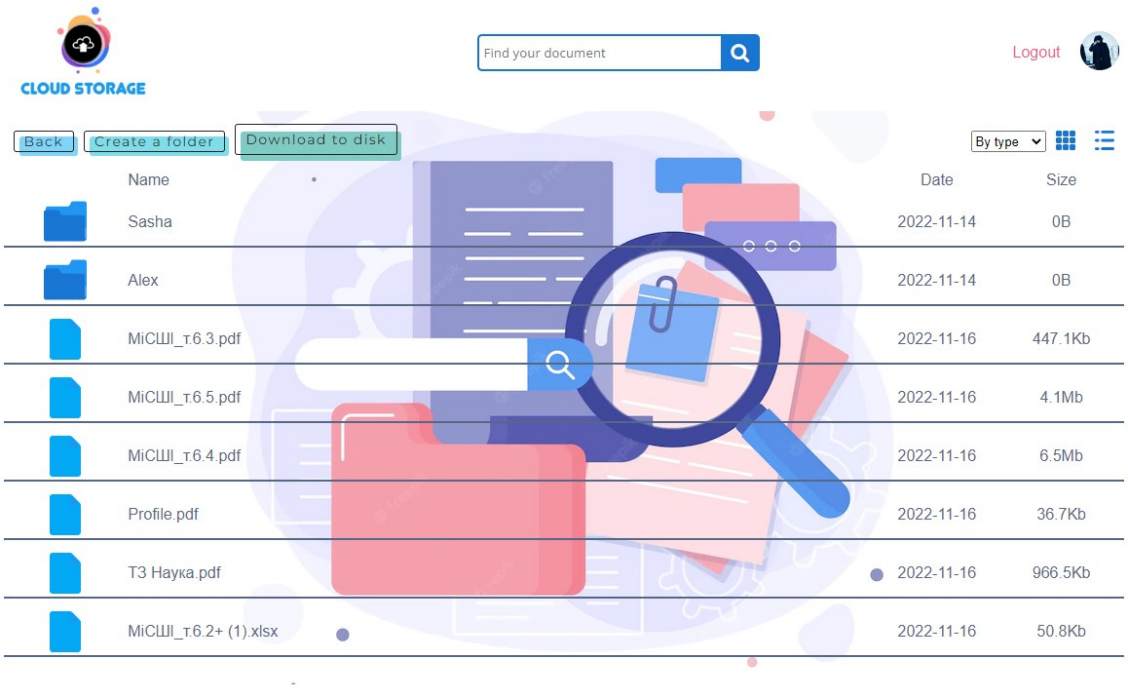

Рис. 1.9. Вигляд сайту "Cloud Storage"

Джерело: розроблено автором

В результаті виконання всіх пунктів, запланованих в технічному завданні, ми отримали сервіс з готовою серверною частиною, а саме головне реалізували інтерфейс з яким може взаємодіяти користувач.

2.4. Оновлення дизайну та розширення функціоналу продукту

Для подальшого розвитку проєкта була розпочата робота на оновленням дизайну виправлення недоліків та розробка додаткового функціоналу.

Щоб побачити наглядно зміни (рис. 2.2), було оновлено діаграму переходу станів для авторизації та реєстрації, проектування діаграм розроблено за допомогою сервісу [12].

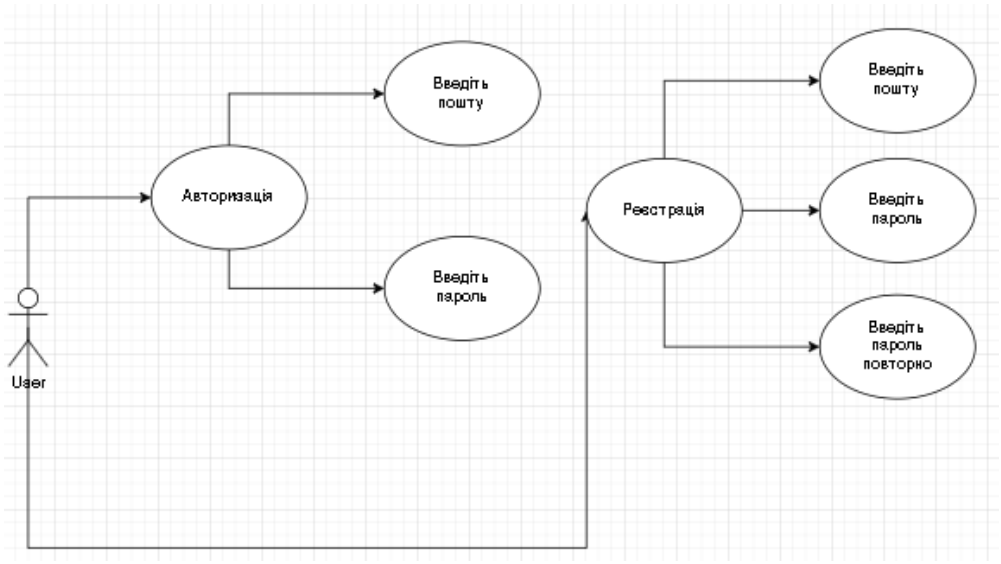

Рис. 2.2. Компонент авторизації та реєстрації

Джерело: розроблено автором

Наступне зображення (рис. 2.3) містить оновлену діаграму переходу станів для організованого зберігання даних та їхнє відображення відповідно новому функціоналу.

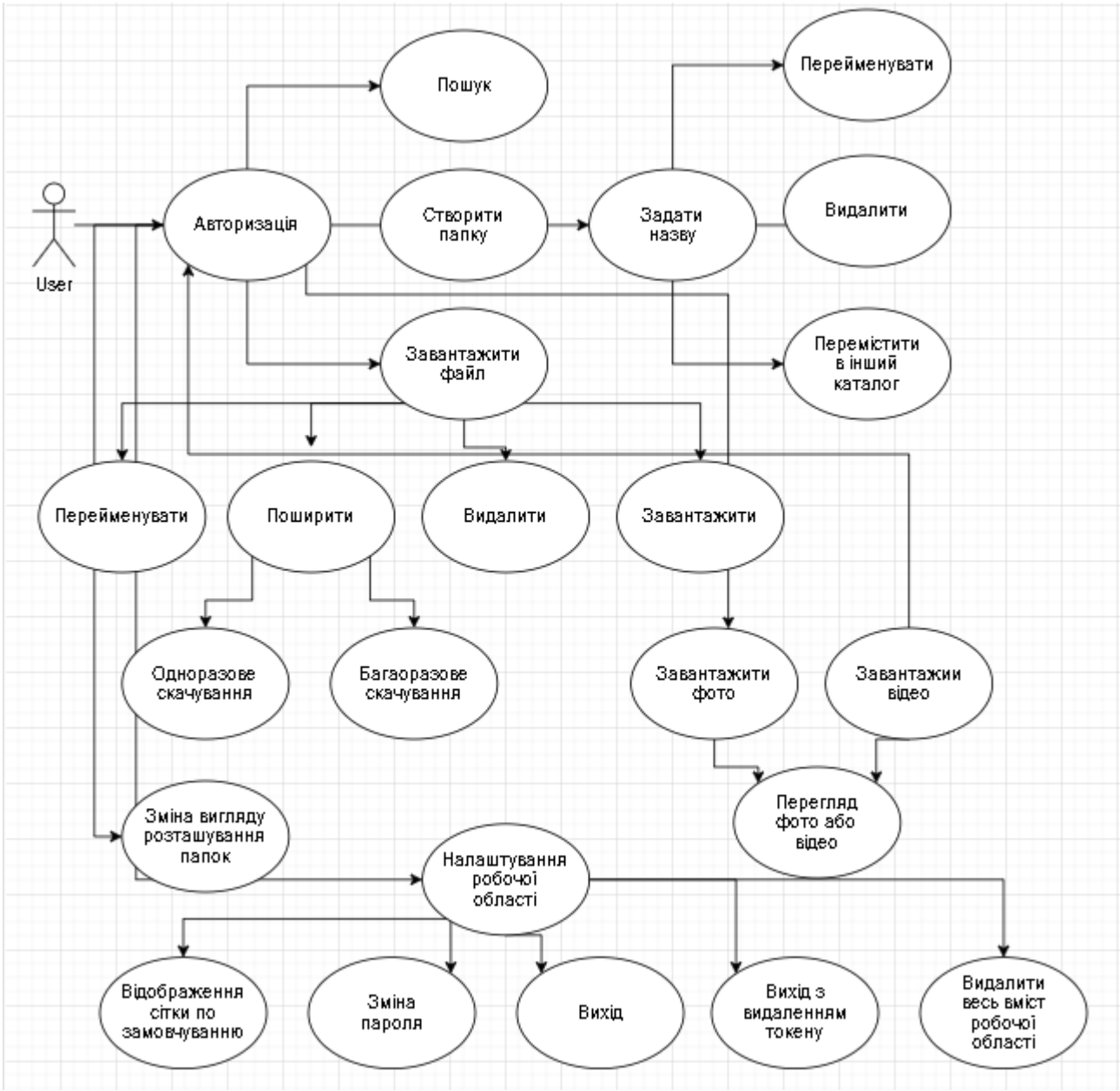

Рис. 2.3. Керування робочою областю

Джерело: розроблено автором

Початкова версія головної сторінки продукту містила лише форму для авторизації та реєстрації, тому було прийнято рішення розшири її. Нова сторінка містить в собі п’ять блоків:

1) хедер;
2) форму з реєстрацією та авторизацією;

3) опис основних можливостей продукту;
4) основні переваги веб-сервісу;
5) футер.

Хедер на сайті - це верхня частина веб-сторінки, яка може в собі містити заголовок сайту, логотип, меню навігації, контактну інформацію та інші елементи дизайну. Він є одним з найважливіших елементів веб-сайту, оскільки він першим знаходиться на сторінці і може впливати на враження від відвідування сайту (рис. 2.4).

Основна функція хедера полягає в тому, щоб дати відвідувачам швидкий доступ до ключової інформації про сайт та його навігації. Хедер також може включати пошукову стрічку, що полегшує користувачам пошук необхідної інформації на сайті.

Крім того, хедер може допомогти збільшити впізнаваність бренду, оскільки логотип та інші елементи дизайну, що зазвичай розташовані в ньому, можуть стати частинами корпоративного стилю бренду.

Таким чином, хедер на сайті є важливою частиною його дизайну та навігації, оскільки він допомагає користувачам знайти потрібну інформацію та впізнати бренд.

Рис. 2.4. Вигляд хедера на головній сторінці проєкту

Джерело: розроблено автором

Наступний рисунок містить в собі оновлену версію для реєстрації та авторизації, по ліву сторону якої розміщене зображення (рис. 2.5).

Форма реєстрації та авторизації на сайті необхідні для забезпечення безпеки та конфіденційності даних користувачів, а також для забезпечення зручності їх взаємодії з сайтом.

Форма реєстрації дозволяє користувачам створити власний обліковий запис на сайті, який може містити їх особисті дані, такі як ім'я, прізвище, електронна пошта,

пароль тощо. Ці дані можуть бути використані для ідентифікації користувача при вході на сайт, а також для забезпечення безпеки його даних та зручного доступу до персоналізованого контенту.

Форма авторизації, у свою чергу, дозволяє користувачам увійти на сайт, використовуючи свій обліковий запис та введені при реєстрації дані. Це дозволяє зберігати користувацьку інформацію в безпеці, а також забезпечує зручну та швидку авторизацію при кожному вході на сайт.

Крім того, форма реєстрації та авторизації може використовуватися для збору інформації про користувачів, яка може бути використана для поліпшення продукту або послуг, що надаються на сайті.

Отже, форма реєстрації та авторизації на сайті необхідні для забезпечення безпеки та конфіденційності даних користувачів, а також для забезпечення зручності їх взаємодії з сайтом та збору інформації про них.

Рис. 2.5. Форма реєстрації та авторизації

Джерело: розроблено автором

Наступний блок на сайті з описом можливостей продукту дозволяє користувачам детальніше ознайомитись з функціями та особливостями продукту (рис. 2.6).

Такий блок допомагає залучити увагу потенційних клієнтів до продукту та переконати їх у його корисності та вигодах. У блоку можуть бути наведені переваги продукту в порівнянні з аналогами на ринку, опис основних функцій, можливості налаштування та інша корисна інформація.

Блок з описом можливостей продукту також допомагає зменшити кількість запитань від потенційних клієнтів, адже вони вже можуть знайти відповіді на свої запитання в описі продукту. Це зменшує навантаження на підтримку клієнтів та збільшує шанси на успішну продажу продукту.

Отже, блок на сайті з описом можливостей продукту є важливим елементом веб-сторінки, який дозволяє залучити увагу потенційних клієнтів та забезпечити їм корисну інформацію про продукт.

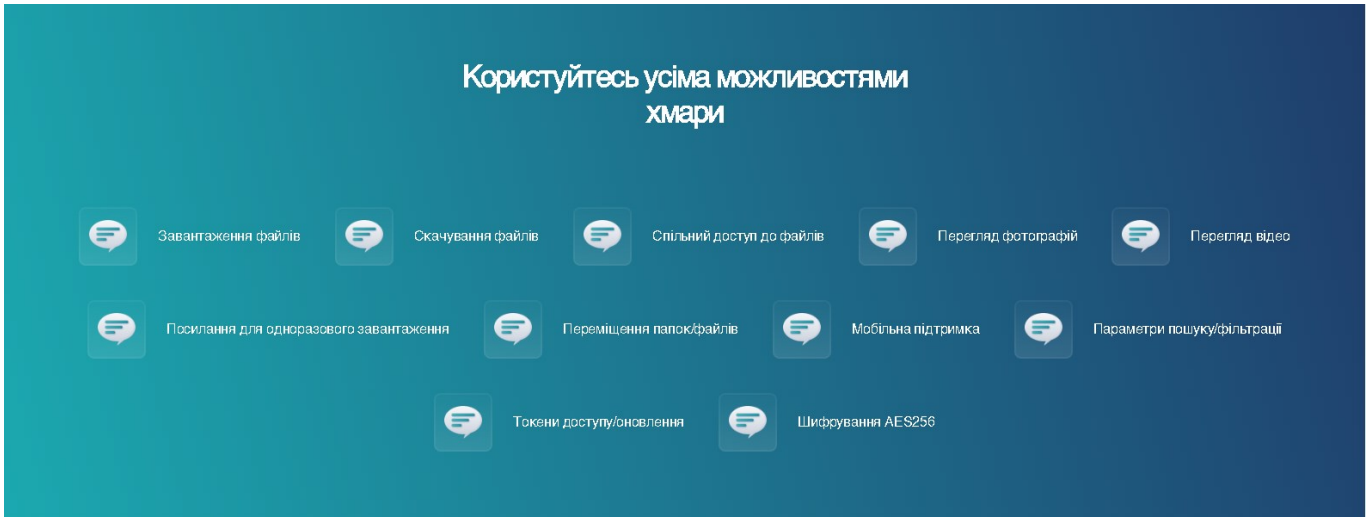

Рис. 2.6. Опис основних можливостей хмари

Джерело: розроблено автором

Наступний блок на сайті з описом основних переваг веб-сервісу є важливим елементом веб-сторінки, який дозволяє залучити увагу потенційних користувачів та переконати їх у корисності та вигодах використання веб-сервісу (рис. 2.7).

Такий блок може містити інформацію про основні функції та можливості веб-сервісу, а також наводити конкретні приклади ситуацій, коли використання сервісу допомагає ефективніше вирішувати певні завдання.

Блок з описом переваг також може включати відгуки та рекомендації від інших користувачів, що дозволяє підтвердити якість та корисність веб-сервісу.

Крім того, такий блок допомагає зменшити кількість запитань від потенційних користувачів, адже вони можуть знайти відповіді на свої запитання в описі переваг веб-сервісу. Це зменшує навантаження на підтримку користувачів та збільшує шанси на успішне використання веб-сервісу.

Отже, блок на сайті з описом основних переваг веб-сервісу є важливим елементом веб-сторінки, який допомагає залучити увагу та переконати потенційних користувачів у корисності та вигодах використання веб-сервісу.

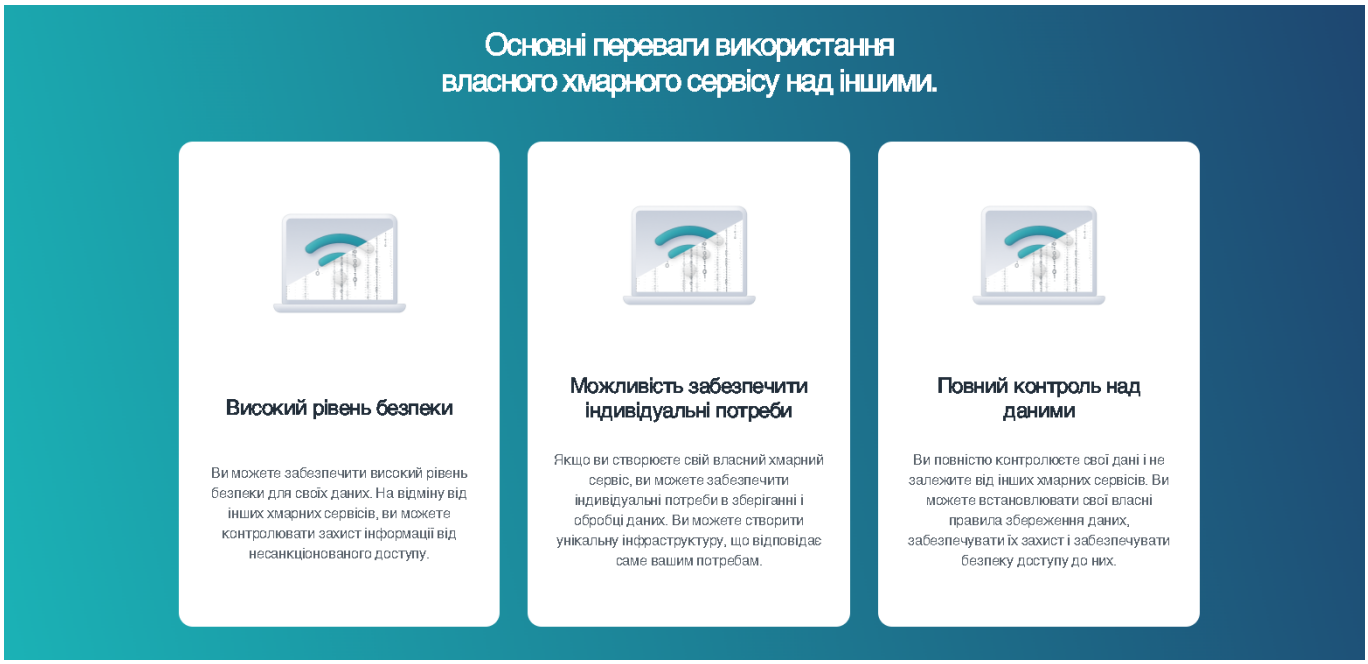

Рис. 2.7. Опис основних переваг хмари

Джерело: розроблено автором

Останній блок на сайті це - футер який є важливим елементом веб-сторінки, який розташований внизу інтерфейсу та містить додаткову інформацію про веб-сайт або компанію (рис. 2.8).

Основні функції футера:

1) Навігація. Футер може містити додаткову навігацію, яка допомагає користувачам знайти необхідну інформацію на сайті. Наприклад, посилання на сторінки "Про нас", "Контакти", "Політика конфіденційності" та ін.
2) Контактна інформація. Футер може містити контактну інформацію компанії, таку як адреса, телефон, електронна пошта, форма зворотнього зв'язку та ін.
3) Соціальні мережі. Футер може містити посилання на профілі компанії в соціальних мережах.
4) Копірайт. Футер може містити копірайт або авторські права на веб-сайт або його контент.
5) Додаткова інформація. Футер може містити додаткову інформацію про веб-сайт або компанію, таку як опис послуг або товарів, партнерів, сертифікати та інше.

Футер на сайті допомагає створити повнісний інтерфейс для користувачів, забезпечує зручний доступ до додаткової інформації та навігації, дозволяє підтримувати зв'язок з відвідувачами та підвищує довіру до веб-сайту та компанії в цілому.

Рис. 2.8. Вигляд футера на головній сторінці

Джерело: розроблено автором

Після початкової реєстрації та подальшої авторизації ми потрапляємо в робочу область хмари, де вже можемо проводити маніпуляції над файлами та папками (рис. 2.9).

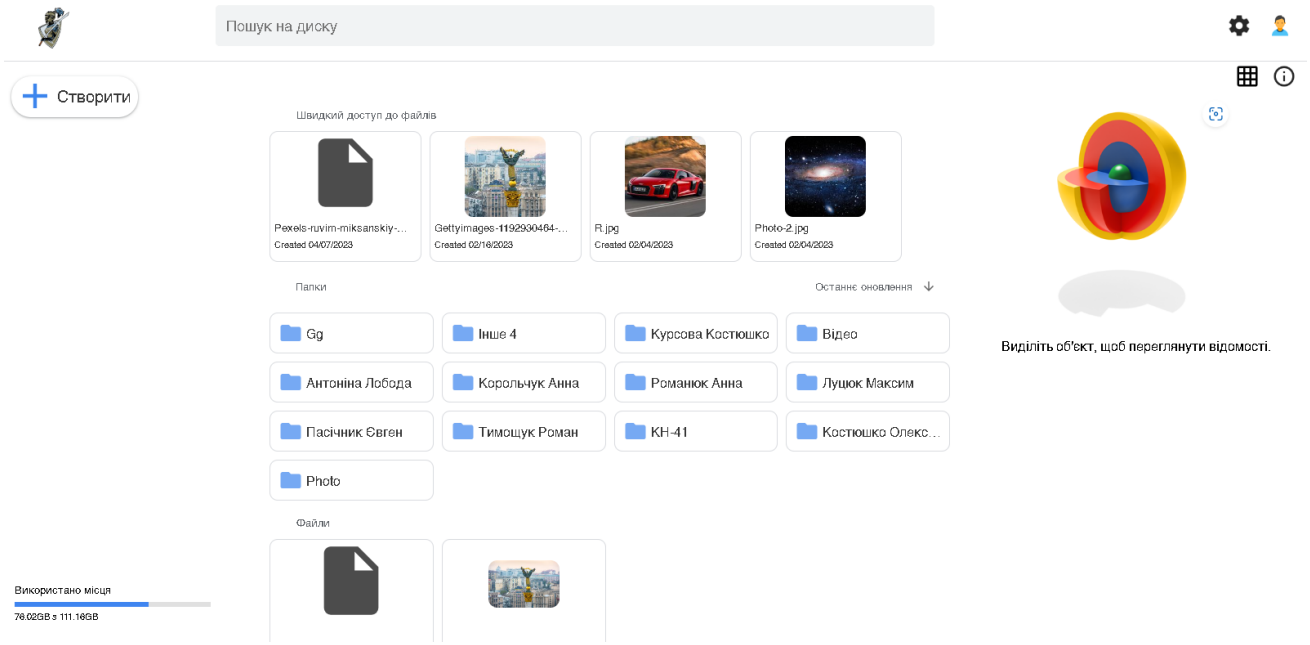

Рис. 2.8. Вигляд робочої області хмарного хостингу

Джерело: розроблено автором

В горі розміщений хедер робочої області, який містить голотип при натискання якого ми повертаємося на сторінку яку бачимо при авторизації. Правіше розміщений пошук по всім файлам та папках, пошук працює і на дочірні файли.

У правій частині розміщені налаштування при натисканні з'являється модальне вікно в якому можна налаштувати стиль розміщення вмісту, міститься інформація про використане місце на диску, реалізована функція зміни пароля, вихід з системи, повний вихід з системи зі знищенням токена доступа та кнопка видалення всього вмісту робочої області (рис. 3.0).

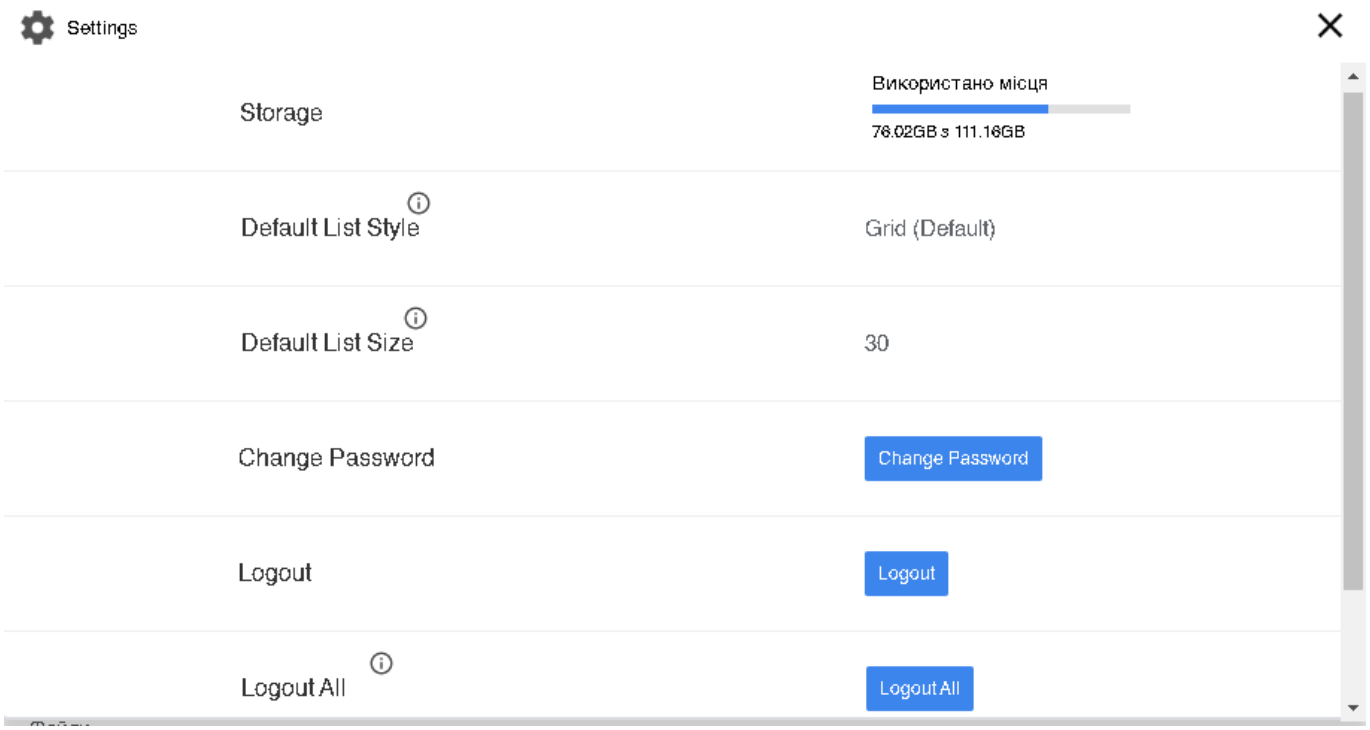

Рис. 3.0. Розділ налаштувань сайту

Джерело: розроблено автором

Робоча область візуально розбита на три частини, для наглядності провів вертикальні лінії (рис. 3.1).

Ліва частина містить кнопку «Створити» при натисканні на неї з’являється спливаюче меню в якій є дві кнопки «Завантажити файл» та «Створити папку».

В кінці першого блоку розміщена інформація про використане місце.

Центральна частина має розділ «Швидкий доступ до файлів», це ті файли які були завантажені останні. Розділи «файли» та «папки» розміщені окремо для зручності використання веб-сервісу.

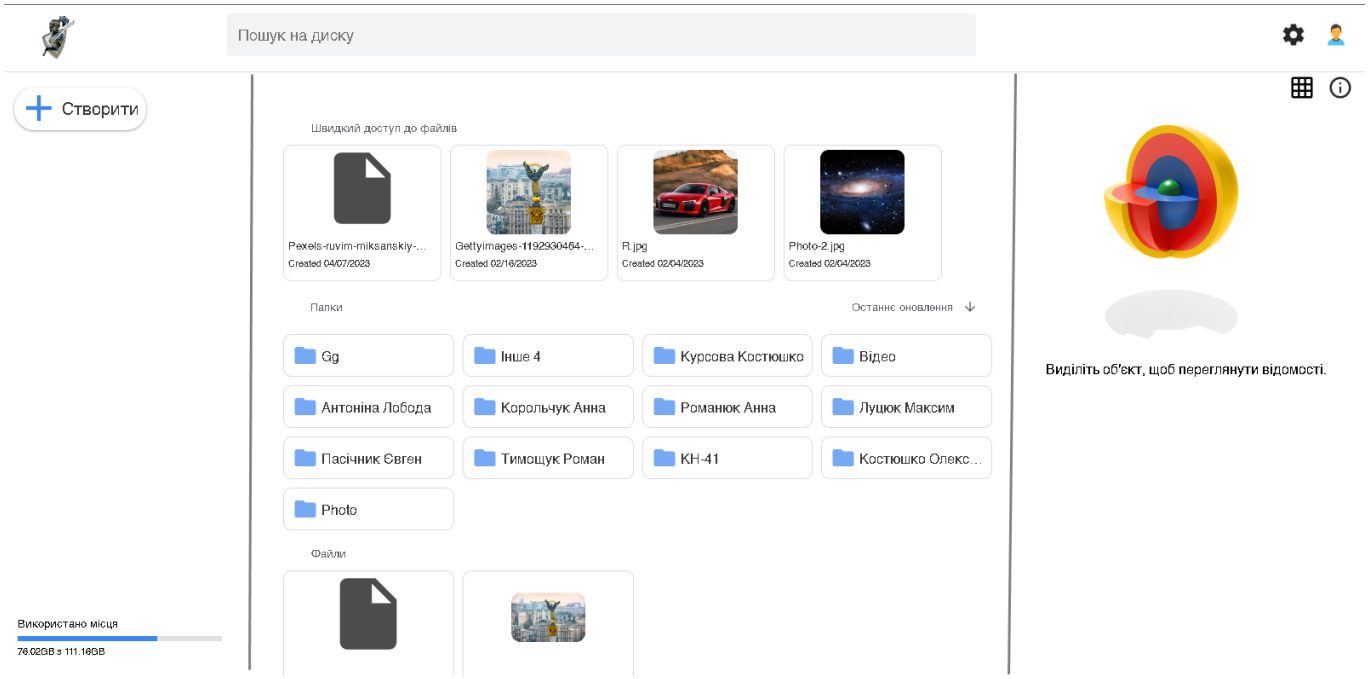

Рис. 3.1. Реалізація розміщення файлів та папок

Джерело: розроблено автором

Остання частина, це блок з інформацією про файл або папку при натискання на неї (рис. 3.2).

Рис. 3.2. Інформація про файл, який обраний

Джерело: розроблено автором

При кожній маніпуляції чи то з файлами і папками є відповідні вікна з підтвердженням (рис. 3.3).

Alerts для підтвердження дії, такі як вікна попереджень (alert) і вікна підтверджень (confirm), є корисними для забезпечення безпеки та запобігання непередбачуваним результатам дій користувачів.

Коли користувач намагається виконати дію, яка може мати серйозні наслідки, alert може бути викликаний, щоб попередити користувача про можливі наслідки цієї дії. Наприклад, якщо користувач намагається видалити важливі дані, може відобразитися вікно попередження, яке запитає користувача, чи він дійсно хоче видалити ці дані.

Це може запобігти непередбачуваним результатам та допомогти користувачеві уникнути помилок. Загалом, alert може збільшити рівень безпеки та зручності взаємодії користувача з програмним забезпеченням.

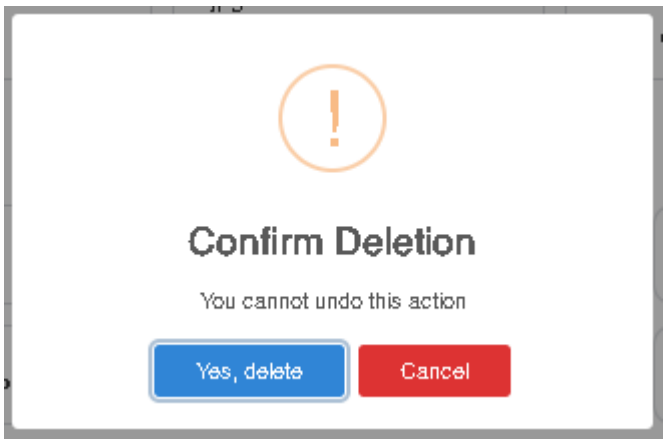

Рис. 3.3. Вікно з підтвердженням дії

Джерело: розроблено автором

Також реалізована функція перейменування папок, файлів завантажених з операційної системи (рис. 3.4).

Функція перейменування папок на хмарі є корисною для організації файлів і папок та полегшення пошуку і управління ними.

Коли користувачі зберігають багато файлів на диску, вони можуть створювати папки, щоб організувати ці файли в зручний спосіб. Проте іноді може знадобитися

змінити назву папки, наприклад, якщо користувач змінює свої потреби щодо організації файлів.

Функція перейменування папок дозволяє користувачеві змінювати назву папки швидко і легко, не втрачаючи доступу до збережених в ній файлів. Це допомагає підтримувати порядок в структурі збереження файлів та полегшує їхнє знаходження та відновлення.

Крім того, функція перейменування є важливою для забезпечення безпеки даних, оскільки дозволяє користувачам змінювати назви папок з конфіденційними даними, якщо їм потрібно приховати або захистити ці дані від несанкціонованого доступу.

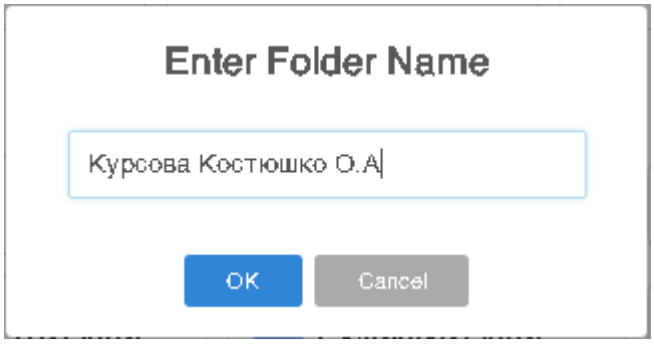

Рис. 3.4. Модальне вікно з перейменуванням папки

Джерело: розроблено автором

Наступна функція на хмарі, це переміщення вмісту в інші папки (рис. 3.5).

Функція переміщення файлів та папок на диску дозволяє користувачам легко і швидко організовувати свої файли та папки та забезпечує більш ефективне управління даними.

Коли користувачі зберігають багато файлів на диску, їм може знадобитися перенести їх в іншу папку, щоб полегшити їх пошук і організацію. Функція переміщення дозволяє користувачеві переносити файли і папки з однієї папки в іншу, не втрачаючи при цьому доступу до цих файлів.

Це допомагає користувачам зберігати свої файли та папки організованіше, забезпечуючи більш ефективне управління даними. Крім того, переміщення файлів та папок може допомогти користувачам зменшити ризик втрати файлів, оскільки вони можуть перенести їх у безпечне місце з більш надійним зберіганням.

Загалом, функція переміщення файлів та папок на хмарі є важливим інструментом для управління даними та забезпечення безпеки даних. Вона дозволяє користувачам організовувати свої файли та папки зручним способом і забезпечувати ефективне управління даними.

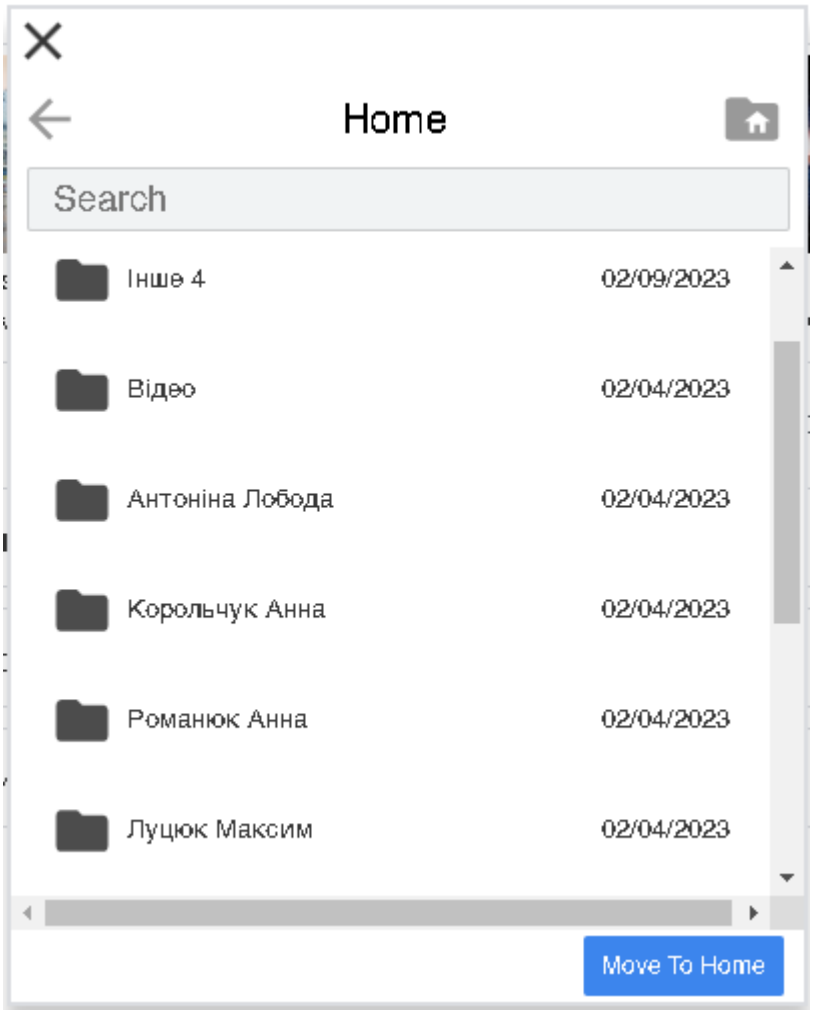

Рис. 3.5. Переміщення файлів, папок в інші каталоги

Джерело: розроблено автором

На диску реалізована функція перегляду відео не скачуючи його на пристрій (рис. 3.6).

Функція перегляду відео на диску дозволяє користувачам зручно переглядати відео файли, які зберігаються на їхньому обліковому записі.

Хмара підтримує різноманітні формати відеофайлів, такі як MP4, MOV, AVI, MKV та багато інших. Функція перегляду відео дозволяє користувачам забезпечити миттєвий доступ до своїх відео без необхідності завантажувати їх на свій комп'ютер або мобільний пристрій.

Функція перегляду відео дозволяє користувачам зберігати відео в одному місці разом з іншими даними, що забезпечує зручний доступ до файлів з будь-якого пристрою з доступом до Інтернету. Користувачі можуть також ділитися своїми відео з іншими користувачами, надаючи їм доступ до відеофайлів шляхом генерації лінка на його скачування, що дозволяє легко та швидко обмінюватися відео контентом.

Загалом, функція перегляду відео є важливою для забезпечення доступу та управління відеофайлами з будь-якого пристрою з Інтернетом. Вона дозволяє користувачам зберігати, ділитися та переглядати відео зручним і безпечним способом.

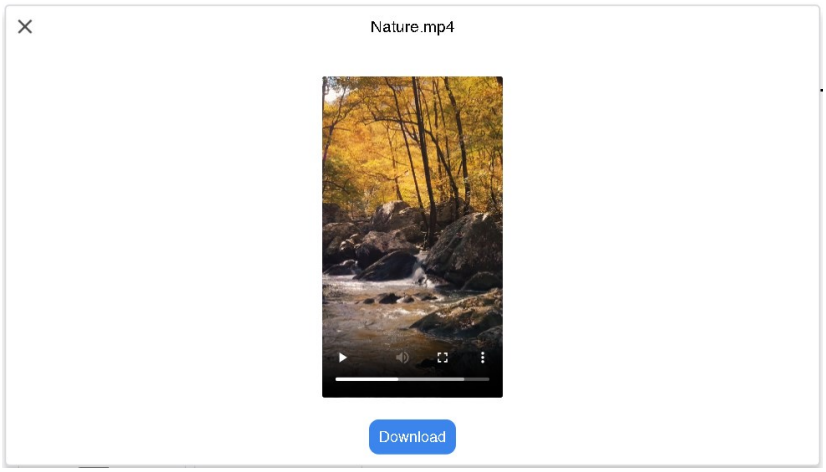

Рис. 3.6. Функція перегляду відео на диску

Джерело: розроблено автором

Крім відео на диску також можна переглядати зображення не скачуючи на свій пристрій (рис. 3.7).

Функція перегляду фото на диску дозволяє користувачам зручно переглядати та керувати своїми зображеннями, які зберігаються на їхньому обліковому записі.

Хмара підтримує різноманітні формати зображень, такі як JPEG, PNG, GIF, BMP та інші. Функція перегляду фото дозволяє користувачам забезпечити миттєвий доступ до своїх зображень без необхідності завантажувати їх на свій комп'ютер або мобільний пристрій.

Функція перегляду фото дозволяє користувачам зберігати зображення в одному місці разом з іншими даними, що забезпечує зручний доступ до файлів з будь-якого пристрою з доступом до Інтернету. Користувачі можуть також ділитися своїми зображеннями з іншими користувачами, надаючи їм доступ.

Загалом, функція перегляду фото є важливою для забезпечення доступу та управління зображеннями з будь-якого пристрою з Інтернетом. Вона дозволяє користувачам зберігати, ділитися та переглядати свої зображення зручним і безпечним способом.

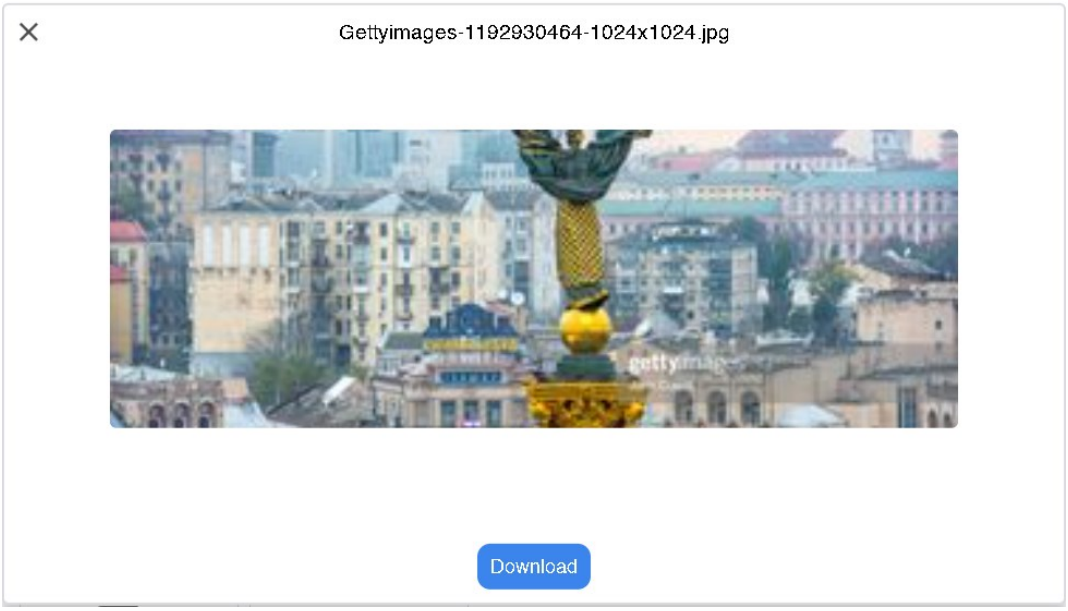

Рис. 3.6. Функція перегляду фото на диску

Джерело: розроблено автором

2.5. Використання JWT-токена в проєкті.

Для безпеки облікових записів користувачів, при розробці хмари був використаний JWT (або "JSON Web Token") - це міцний та безпечний механізм аутентифікації та авторизації, який використовується в багатьох сучасних веб-додатках та API.

JWT-токен відповідає за збереження інформації про користувача, що забезпечує безпеку при передачі даних між сервером та клієнтом. JWT складається з трьох частин: заголовку, клейма та підпису.

JWT-токени використовуються для аутентифікації та авторизації користувачів на веб-сайтах, мобільних додатках та API. Коли користувач успішно автентифікується на сервері, сервер створює JWT-токен і передає його клієнту, який зберігає токен в локальному сховищі, такому як cookies або local storage.

При кожному запиті клієнта на сервер, JWT-токен додається до заголовку запиту. Сервер перевіряє підпис токена, щоб переконатися, що токен був підписаний сервером і не був змінений з моменту створення. Якщо токен валідний, то сервер авторизує запит та повертає користувачеві очікувані дані.

JWT-токени дозволяють створювати безпечні та розширювані системи авторизації та автентифікації, що є важливим для захисту конфіденційної інформації та забезпечення безпеки користувачів.

2.6. Використання шифрування AES 256, для підвищення безпеки.

AES (Advanced Encryption Standard) - це симетричний алгоритм шифрування, що використовується для захисту конфіденційних даних в багатьох різних сферах, включаючи фінансові транзакції, збереження даних в хмарних сервісах, обмін повідомленнями, інформацію про кредитні картки та багато іншого.

AES забезпечує міцне шифрування, яке забезпечує захист даних від несанкціонованого доступу. AES 256 - це версія AES, яка використовується з 256-бітним ключем, що робить його ще більш міцним і надійним в порівнянні з AES 128, який використовує ключ довжиною 128 бітів.

Шифрування AES 256 використовується для захисту даних в різних системах, включаючи зберігання даних в хмарних сервісах, передачу даних по мережі Інтернет, а також для захисту даних на рівні зберігання, наприклад, на жорстких дисках, флеш-накопичувачах та інших носіях інформації.

AES 256 шифрування забезпечує високий рівень безпеки для захисту даних від зловмисників та несанкціонованого доступу, тому воно використовується у багатьох системах зберігання та передачі конфіденційної інформації.

Приклади: див. Додатки Г, Д, Е.

Опис додатку Г. ../serverUtils/changeEncryptionPassword

В додатку Г, код є набором функцій, які виконують різні завдання пов'язані з шифруванням файлів, та роботою з базою даних.

1) Функція “waitForDatabase” повертає проміс, яка очікує, поки стан з'єднання з базою даних не стане готовим до використання. Це робиться шляхом створення нового промісу, який перевіряє стан з'єднання. Якщо з'єднання готове, проміс розривається і повертається результат.

2) Функція “reencryptFile” приймає файл, новий ключ шифрування та користувача, та повертає проміс. Функція розшифровує заданий файл з допомогою старого ключа шифрування, шифрує його з новим ключем та завантажує до бази даних.

3) Функція “findFiles” витягує список користувачів з бази даних та проходиться по кожному з них. Для кожного користувача функція витягує список файлів, що належать цьому користувачеві, та виконує функцію “reencryptFile” для кожного з цих файлів.

4) Функція “generateEncryptionKeys” генерує ключі шифрування для користувача. Вона приймає користувача як параметр та генерує випадковий ключ, який шифрується з ключем користувача, який потім шифрується з майстер-ключем. Результат зберігається у відповідних полях користувача.

5) Функція “getOldEncryptionKey” отримує об'єкт користувача та повертає розшифрований ключ шифрування для даного користувача. Спочатку функція отримує пароль користувача, зашифрований приватний ключ, новий пароль та вектор ініціалізації (iv) з публічного ключа користувача.

Після чого функція використовує SHA256 хеш пароля користувача та нового пароля для отримання ключів шифрування USER_CIPHER_KEY та MASTER_CIPHER_KEY.

Потім вона дешифрує зашифрований приватний ключ з використанням MASTER_CIPHER_KEY, та результат цієї дешифрації дешифрує ще раз, використовуючи USER_CIPHER_KEY, для отримання остаточного розшифрованого ключа шифрування, який і повертається функцією.

Обіцянка (англ. Promise) - це об'єкт, який використовується в асинхронному програмуванні JavaScript для представлення результату асинхронної операції, що ще не виконана. Promise дає можливість працювати з асинхронним кодом, який потребує часу для виконання, і робити його схожим на синхронний. Використання Promise важливо для ефективної роботи з асинхронними операціями, наприклад, з запитами до сервера або операціями з файлами, що можуть займати багато часу. Promise має три стани: pending (очікування), fulfilled (виконання) і rejected (відхилений). При виконанні асинхронної операції Promise переходить в один зі станів - виконання або відхилення, після чого викликається відповідний обробник (then або catch).

Опис додатку Д. ../key/getKey

В додатку Д код експортує функцію “getKey”, яка використовується для отримання ключа шифрування. Ця функція перевіряє, чи є змінна середовища `KEY`, якщо ні, то запитує введення пароля для шифрування на сервері в режимі прихованого вводу. Якщо змінна середовища `NODE_ENV` встановлена, то вона використовується для перевірки, чи код працює на сервері, а якщо ні, то встановлюється значення "1234" в якості пароля за замовчуванням. Після цього

використовується хеш-функція MD5 для отримання хешу пароля, який зберігається в змінній “env.key”.

Опис додатку Е. ../serverUtils/createThumbnailBuffer

В додатку Е код експортує функцію “createThumbnail”, яка призначена для створення мініатюр зображень та зберігання їх у базі даних. Функція отримує на вхід параметри “file” - об'єкт файлу з бази даних, “filename” - ім'я файлу, “user” - об'єкт користувача, та “newKey” - ключ шифрування.

У функції спочатку створюється новий “GridFSBucket” для поточного з'єднання з базою даних MongoDB, який використовується для зберігання файлів.

Потім створюється новий потік для завантаження даних з файлу. Потік пропускається через розшифрування з використанням ключа “newKey”, що передається у функцію. Після чого застосовується обробка зображення, де використовується бібліотека “sharp”.

Зображення зменшується до розміру 300 пікселів. Отримане зменшене зображення шифрується з використанням ключа “newKey” та випадкового вектору ініціалізації (“thumbnailIV”), що створюється за допомогою “crypto.randomBytes(16)”. Шифрований текст зберігається у моделі “Thumbnail”, яка містить інформацію про мініатюру файлу та її власника.

У функції `crypto.randomBytes(16)` параметр `16` визначає кількість байтів (bytes), які будуть згенеровані випадковим чином за допомогою криптографічного генератора псевдовипадкових чисел в Node.js. У цьому конкретному випадку, генерується масив байтів довжиною 16, що може бути використано, наприклад, як ініціалізаційний вектор для шифрування або як унікальний ідентифікатор для об'єктів.

У збереженій моделі “Thumbnail” встановлюється параметр “name”, який дорівнює “filename”, та параметр “owner”, який дорівнює “_id” власника зображення. Після змінюються метадані файлу в колекції “fs.files”, де встановлюється параметр

“metadata.hasThumbnail” у “true”, а також параметр “metadata.thumbnailID” у “_id” моделі “Thumbnail”. Оновлений об'єкт файлу повертається з функції. У випадку помилки під час виконання будь-якої з цих операцій, функція поверне початковий об'єкт файлу.

Створення мініатюр зображень - це процес зменшення розміру зображення з метою зниження його об'єму, щоб воно займало менше місця на диску та завантажувалося швидше при відображенні на веб-сторінках або в мобільних додатках. Зазвичай мініатюри зображень використовуються для попереднього перегляду великих зображень, або як іконки в списку файлів.

Збереження мініатюр зображень в базі даних дозволяє зберегти їх з найкращою якістю та відновити у випадку втрати або пошкодження оригінальних файлів. Крім того, це забезпечує легкий доступ до мініатюр та дозволяє швидко відтворювати їх на сторінках веб-сайту або в додатках, що допомагає покращити швидкість завантаження сторінок.

2.7. Визначені шляхи (routes).

../src/express-routes/file.js

В нижче наведеній таблиці визначені шляхи (routes), які можуть бути запитані клієнтом через HTTP запити. При отриманні запиту, сервер відповість на запит відповідним контролером, що міститься в іншому файлі. Таким чином, “*file.js*”, забезпечує зв'язок між клієнтською та серверною частинами додатку.

Таблиця 2.2

Зв'язок між клієнтською та серверною частинами.

```
const fileController = new FileController()

router.post("/file-service/upload", auth,
fileController.uploadFile);

router.post("/file-service/transcode-video", auth,
fileController.transcodeVideo);
```

```
router.get("/file-service/thumbnail/:id", auth,
fileController.getThumbnail);

router.get("/file-service/full-thumbnail/:id", auth,
fileController.getFullThumbnail);

router.get("/file-service/public/download/:id/:tempToken",
fileController.getPublicDownload);

router.get("/file-service/public/info/:id/:tempToken",
fileController.getPublicInfo);

router.get("/file-service/info/:id", auth,
fileController.getFileInfo);

router.get("/file-service/quick-list", auth,
fileController.getQuickList);

router.get("/file-service/list", auth,
fileController.getList);

router.get("/file-service/download/get-token", auth,
fileController.getDownloadToken);

router.get("/file-service/download/get-token-video", auth,
fileController.getDownloadTokenVideo);

router.get("/file-service/stream-video-
transcoded/:id/:tempToken/:uuid", tempAuthVideo,
fileController.streamTranscodedVideo);

router.get("/file-service/stream-
video/:id/:tempToken/:uuid", tempAuthVideo,
fileController.streamVideo);

router.get("/file-service/download/:id/:tempToken",
tempAuth, fileController.downloadFile);

router.get("/file-service/suggested-list", auth,
fileController.getSuggestedList);

router.patch("/file-service/make-public/:id", auth,
fileController.makePublic);

router.patch("/file-service/make-one/:id", auth,
```

```
fileController.makeOneTimePublic);

router.patch("/file-service/rename", auth,
fileController.renameFile);

router.patch("/file-service/move", auth,
fileController.moveFile);

router.delete("/file-service/remove-link/:id", auth,
fileController.removeLink);

router.delete("/file-service/remove/token-
video/:tempToken", auth, fileController.removeTempToken);

router.delete("/file-service/transcode-video/remove", auth,
fileController.removeTranscodeVideo);

router.delete("/file-service/remove", auth,
fileController.deleteFile);

module.exports = router;
```

Джерело: розроблено автором

Доповнюючи вище сказане, це файл, що експортує об'єкт маршрутизатора (“router”), який містить шляхи та відповідні контролери для файлового сервісу, який працює з обліковим сховищем даних.

Тут визначено різні шляхи для завантаження файлів, отримання інформації про файли, їх перейменування, переміщення, видалення і т.д. Також визначено шляхи для отримання тимчасових токенів для завантаження та завантаження транскодованих відеофайлів, отримання та видалення тимчасових токенів для завантаження файлів, отримання списку файлів та багато іншого.

Контролери, які використовуються в шляхах, визначені в файлі “../controllers/file.js”. Там містяться методи, які обробляють запити до файлового сервісу.

../src/express-routes/folder.js

Таблиця 2.3

Зв'язок між клієнтською та серверною частинами.

```
router.post("/folder-service/upload", auth,
folderController.uploadFolder);

router.delete("/folder-service/remove", auth,
folderController.deleteFolder);

router.delete("/folder-service/remove-all", auth,
folderController.deleteAll);

router.get("/folder-service/info/:id", auth,
folderController.getInfo);

router.get("/folder-service/subfolder-list", auth,
folderController.getSubfolderList);

router.get("/folder-service/list", auth,
folderController.getFolderList);

router.patch("/folder-service/rename", auth,
folderController.renameFolder);

router.patch("/folder-service/move", auth,
folderController.moveFolder);

module.exports = router;
```

Джерело: розроблено автором

Файл містить операції для завантаження нової папки на сервер, видалення папки, видалення всіх папок, отримання інформації про конкретну папку, отримання списку підпапок, отримання списку всіх папок, перейменування папки та переміщення.

../src/express-routes/storage.js

Таблиця 2.4

Зв'язок між клієнтською та серверною частинами.

```
router.get("/storage-service/info", auth,
storageController.getStorageInfo);

module.exports = router;
```

Джерело: розроблено автором

Файл містить операцію для отримання інформації про простір зберігання, який повертає інформацію про загальний обсяг простору, який доступний для користувача, і обсяг використаного простору.

../src/express-routes/user.js

Таблиця 2.5

Зв'язок між клієнтською та серверною частинами.

```
router.get("/user-service/user", auth,
userController.getUser);

router.post("/user-service/login", userController.login);

router.post("/user-service/logout", auth,
userController.logout);

router.post("/user-service/logout-all", auth,
userController.logoutAll);

router.post("/user-service/create",
userController.createUser);

router.post("/user-service/change-password", auth,
userController.changePassword);

module.exports = router;
```

Джерело: розроблено автором

Файл містить маршрути для обробки запитів, що стосуються користувачів, такі як вхід у систему, вихід із системи, створення нового користувача, зміна пароля та отримання даних користувача. Захист від несанкціонованого доступу до деяких маршрутів здійснюється за допомогою middleware “auth”.

Middleware “auth” вказує на те, що деякі маршрути потребують авторизації перед доступом до них. Це захист від несанкціонованого доступу до захищених маршрутів.

Коли користувач намагається отримати доступ до захищеного маршруту, сервер перевіряє, чи має користувач правильний токен доступу. Якщо токен є дійсним, сервер дозволяє користувачеві отримати доступ до маршруту. Якщо токен недійсний або його не існує, сервер відхиляє запит і повертає помилку авторизації.

2.8. Вибір теми на кваліфікаційну роботу

Вибором даної теми для створення власного хмарного сервісу є декілька причин. Ось одні з них:

1) Забезпечення безпеки даних: Я зможу забезпечити безпеку своїх даних самостійно і здійснити контроль над ними.

2) Економія коштів: Створення власного хмарного сервісу може бути економічно вигідним варіантом в порівнянні з оплатою послуг інших хмарних сервісів.

3) Вільний доступ до власних даних: Коли ви маєте свій власний хмарний сервіс, ви маєте повний доступ і контроль над своїми даними, а не залежите від послуг інших компаній.

4) Контроль над функціональністю: Створюючи свій власний хмарний сервіс, ви можете додавати функціональність та змінювати її відповідно до своїх потреб і бізнес-вимог.

Зважаючи на ці фактори, я вважаю створення власного хмарного сервісу може бути корисним і доцільним для мене.

2.9. Перспективи розвитку проєкта

Проєкт хмарного сервісу може мати багато перспектив розвитку, особливо з урахуванням того, як швидко розвивається сфера технологій та інформаційних систем. Ось декілька можливих напрямків розвитку проєкта хмарного сервісу:

1) Розширення функціональності.

Розвиток нових можливостей та функцій для підвищення ефективності та продуктивності користувачів.

2) Поліпшення безпеки.

Забезпечення додаткових заходів безпеки, таких як шифрування даних та використання біометричних технологій.

3) Розширення мобільності.

Розвиток можливостей доступу до хмарних сервісів з різних пристроїв та з усіх куточків світу.

4) Використання інтелектуальних технологій.

Використання штучного інтелекту та машинного навчання для покращення якості та ефективності хмарних сервісів.

5) Розширення міжнародного присутності.

Розширення можливостей доступу до хмарних сервісів з різних країн світу та забезпечення підтримки мовних версій.

6) Використання блокчейн технологій.

Використання технологій блокчейн для забезпечення безпеки, захисту приватності та підвищення довіри користувачів до хмарних сервісів.

7) Розширення співпраці з іншими компаніями.

Розширення можливостей для співпраці з іншими компаніями та інтеграції хмарних сервісів з іншими послугами та додатками.

Враховуючи ці можливі напрямки розвитку, проєкт хмарного сервісу може мати багато перспектив для зростання.

2.10. Переваги використання власного хмарного сервісу.

Хмарний сервіс створювався для власного користування і ось основні переваги:

1) Повний контроль над даними.

Ви повністю контролюєте свої дані і не залежите від інших хмарних сервісів. Ви можете встановлювати свої власні правила збереження даних, забезпечувати їх захист і забезпечувати безпеку доступу до них.

2) Високий рівень безпеки.

Ви можете забезпечити високий рівень безпеки для своїх даних. На відміну від інших хмарних сервісів, ви можете контролювати захист інформації від несанкціонованого доступу.

3) Можливість зберігання великої кількості даних.

Ви можете зберігати велику кількість даних на своєму власному хмарному сервісі, без необхідності платити за додатковий простір.

4) Можливість забезпечити індивідуальні потреби.

Якщо ви створюєте свій власний хмарний сервіс, ви можете забезпечити індивідуальні потреби в зберіганні і обробці даних. Ви можете створити унікальну інфраструктуру, що відповідає саме вашим потребам.

5) Економія коштів.

Створення власного хмарного сервісу може бути економічним рішенням, особливо якщо у вас велика кількість даних, які необхідно зберігати і обробляти.

Крім того, з власним хмарним сервісом ви будете міцніше контролювати свої дані та безпеку. Інші сервіси можуть мати слабкі місця в безпеці, або можуть збирати та використовувати ваші дані для реклами або інших цілей. Якщо ви самі керуєте своїми даними, то зможете бути впевнені в їхньому захисті та конфіденційності.

Також, з власним хмарним сервісом ви можете налаштувати його так, як вам потрібно, та використовувати його для своїх власних цілей. Наприклад, ви можете налаштувати спеціальні права доступу до своїх файлів для конкретних користувачів або груп користувачів, які мають доступ до вашого хмарного сервісу. Це може бути особливо корисно для бізнесу або колективних проектів.

Власний хмарний сервіс може бути більш ефективним з точки зору витрат на зберігання даних, особливо якщо ви маєте великі обсяги даних. Інші сервіси можуть збільшувати свої ціни з часом, або можуть мати обмеження щодо обсягу даних, які ви можете зберігати. З власним хмарним сервісом ви можете налаштувати його так, щоб він відповідав вашим потребам та був ефективним з точки зору витрат.

ВИСНОВКИ

Підсумовуючи написане в кваліфікаційній роботі, можна сказати, що була поставлена проблема даної області дослідження, в ході якої ми дослідили, що актуальність використання хмарних сервісів пояснюється широким застосуванням в освіті сучасних програм, які характеризуються своєю різноманітністю і простотою використання. Хмарні сервіси надають можливість автоматизувати подачу навчального матеріалу. По-перше, це автоматизація, як самого процесу створення, так і зберігання даних в будь-якому форматі. По-друге, це робота з практично необмеженим обсягом даних. По-третє, студенти набираються досвіду використання подібних технологій.

Також прийшли до висновку, що популярність “хмарних сервісів” пов’язана з бурхливим розвитком Інтернету і супутніх технологій. На багатьох підприємствах люди працюють у віддаленому режимі, передаючи всю необхідну інформацію через інтернет. Хмарні технології надають споживачам рішення, повністю готові до роботи. Достатньо володіти будь-яким пристроєм, здатним з’єднатися з інтернетом, і можна отримати доступ до віддаленої бази, яка розташовується на віддаленому сервері.

Було розглянуто основні визначення, такі як: хмарні технології, хмарний сервіс, хмарні обчислення та сама хмара.

Затронули історію хмарних технологій, в якій дізналися, що концепція хмарних обчислень з'явилася ще в 1960 році, коли американський учений, фахівець з теорії ЕОМ Джон Маккарті (John McCarthy) висловив припущення, що коли-небудь комп'ютерні обчислення стануть надаватися подібно комунальним послугам (public utility). Розповсюдження мереж з високою потужністю, низька вартість комп'ютерів і пристроїв зберігання даних, а також широке впровадження віртуалізації, сервіс-орієнтованої архітектури привели до величезного зростання хмарних обчислень.

В кінці першого розділу провели SWOT аналіз основних конкурентів на ринку, таких як: Google Drive, Microsoft OneDrive, pCloud та дослідили власних продукт, де описали сильні сторони, слабкі, можливості та загрози для сервісу.

На початку другого розділу проводиться опис технологічного стеку, який використовується для створення продукту та детально описано, яка технологія за що відповідає.

Побудували діаграми варіантів використання, які часто використовується для аналізу різних систем. Вони дозволяють візуалізувати різні типи ролей у системі та те, як ці ролі взаємодіють із системою.

Також з'ясували, що створення власного хмарного сервісу може бути вигідним та корисним з різних причин, включаючи забезпечення безпеки даних, економію коштів, вільний доступ до власних даних та контроль над функціональністю. Використання шифрування AES 256 може підвищити безпеку зберігання та передачі даних у хмарному сервісі. Таким чином, розробка власного хмарного сервісу є цікавою і перспективною темою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fullstack React: The Complete Guide to ReactJS and Friends URL: https://www.goodreads.com/book/show/32705383-fullstack-react. (дата звернення: 12.11.2022 р).
2. Learning React: A Hands-On Guide to Building Web Applications Using React and Redux URL: https://www.oreilly.com/library/view/learning-react-a/9780134843582/. (дата звернення: 12.11.2022 р).
3. React: Up & Running: Building Web Applications URL: https://www.goodreads.com/book/show/26177715-react. (дата звернення: 12.11.2022 р).
4. Введення в ReactJS JavaScript Framework URL: https://www.telerik.com/blogs/introduction-to-the-react-javascript-framework. (дата звернення: 12.11.2022 р).
5. React: сучасні шаблони для розробки додатків 2-ге видання URL: https://booksit.com.ua/3353-react-sovremennye-shablony-dlya-razrabotki-prilojeniy-2-e-izdanie. (дата звернення: 12.11.2022 р).
6. Система управління навчанням Moodle URL: www.moodle.org. (дата звернення: 12.11.2022 р).
7. Хмарне середовище зберігання даних OneDrive URL: https://onedrive.live.com/. (дата звернення: 30.11.2022 р).
8. React Tutorial URL: https://www.w3schools.com/REACT/DEFAULT.ASP. (дата звернення: 30.11.2022 р).
9. CSS Tutorial URL: https://www.w3schools.com/css/. (дата звернення: 30.11.2022 р).
10. HTML Tutorial URL: https://www.w3schools.com/html/. (дата звернення: 30.11.2022 р).
11. JavaScript Tutorial URL: https://www.w3schools.com/js/. (дата звернення: 30.11.2022 р).

12. Lucid Visual Collaboration Suite URL: https://lucid.app/users/login#/login. (дата звернення: 8.12.2022 р).
13. Heroicons URL: https://heroicons.dev/. (дата звернення: 8.12.2022 р).
14. GitHub // Том Престон-Вернер, P. J. Hyett, Кріс Ванстрет, Скотт Чакон URL: https://github.com/. (дата звернення: 8.12.2022 р).
15. pCloud - Store, share and access all your files using one simple and highly secure platform, anytime and anywhere you go URL: https://www.pcloud.com/eu. (дата звернення: 8.12.2022 р).
16. OneDrive URL: https://www.microsoft.com/uk-ua/microsoft-365/onedrive/online-cloud-storage. (дата звернення: 8.12.2022 р).
17. Figma URL: https://www.figma.com/. (дата звернення: 11.01.2023 р).
18. npm // Ребекка Тернер, Кат Марчан, інші. URL: https://www.npmjs.com/. (дата звернення: 11.01.2023 р).
19. WebStorm // JetBrains URL: https://www.jetbrains.com/ru-ru/webstorm/. (дата звернення: 11.01.2023 р).
20. What Is Cloud Computing and the Top Cloud Technologies to Look Out for in 2023 URL: https://www.simplilearn.com/cloud-technologies-article. (дата звернення: 11.01.2023 р).
21. Google Drive URL: https://www.google.com/intl/uk_UA/drive/. (дата звернення: 11.01.2023 р).
22. Комплексне використання хмарних сервісів в електронному навчальному курсі URL: https://lib.iitta.gov.ua/11119/1/Gerasimenko_I_article.pdf. (дата звернення: 11.01.2023 р).
23. Хмарні технології навчання. Історія хмарних сервісів. Євтушок Олена Олегівна URL: https://sites.google.com/site/olwarr339/ikt-u-navcanni/hmarni-tehnologiie-navcanna. (дата звернення: 11.01.2023 р).
24. Хмарні технології для контролю знань у ВНЗ URL: https://sites.google.com/site/hmarnitehnologiie97/zmist-lekciie. (дата звернення: 11.01.2023 р).

25. MERN Stack Explained URL: https://www.mongodb.com/mern-stack. (дата звернення: 11.01.2023 р).

ДОДАТКИ

ДОДАТОК А

Посилання на вихідний код програмного продукту “Cloud Storage”

Розробка програмного продукту для організованого зберігання даних [Електронний ресурс] // Oleksandr Kostiushko. – 2022. – Режим доступу до ресурсу: KostiushkoOleksandr/cloud-Storage-Updated (github.com).

ДОДАТОК Б

Приклад використання модуля “crypto”

```
const waitForDatabase = () => {

   return new Promise((resolve, reject) => {

       if (conn.readyState !== 1) {

           conn.once("open", () => {

               resolve();

           })

       } else {

           resolve();
       }

   })
}

const reencryptFile = (file, newKey, user) => {

   return new Promise(async(resolve, reject) => {

       const fileID = file._id;
       const filename = file.filename;

       let decryptBucket = new mongoose.mongo.GridFSBucket(conn.db, {
           chunkSizeBytes: 1024 * 255,
           bucketName: "temp-fs"
       });

       let bucket = new mongoose.mongo.GridFSBucket(conn.db, {
           chunkSizeBytes: 1024 * 255
       });

       const metadata = file.metadata;

       const readStream =
decryptBucket.openDownloadStream(ObjectID(fileID));

       const writeStream = bucket.openUploadStream(filename, {metadata});

       const foundOldUser =  await conn.db.collection("temp-
users").findOne({_id: user._id});

       const password = getOldEncryptionKey(foundOldUser);

       const IV = file.metadata.IV.buffer

       const CIPHER_KEY =
```

```
crypto.createHash('sha256').update(password).digest()

        const decipher = crypto.createDecipheriv('aes256', CIPHER_KEY, IV);

        const NEW_CIPHER_KEY =
crypto.createHash('sha256').update(newKey).digest()

        const cipher = crypto.createCipheriv('aes256', NEW_CIPHER_KEY, IV);

        cipher.on("error", (e) => {
            console.log("de", e);
        })

        readStream.pipe(decipher).pipe(cipher).pipe(writeStream);

        writeStream.on("finish", async(newFile) => {

            const imageCheck = imageChecker(filename);

            if (file.length < 15728640 && imageCheck) {

               try {
                    await createThumbnail(newFile, filename, user, newKey);
               } catch (e) {
                   console.log("Cannot create thumbnail", e);
               }

                resolve();

            } else {

                resolve();
            }

        })

    })
}


const findFiles = async() => {

    const userListCursor = await conn.db.collection("users").find({});
    const userListCount = await
conn.db.collection("users").find({}).count();
    const progressBar = new cliProgress.SingleBar({},
cliProgress.Presets.shades_classic);

    progressBar.start(userListCount, 0);

    for await (const currentUser of userListCursor) {

        const currentUserID = currentUser._id;

        const newEncrpytionKey = getEncryptionKey(currentUser);

        const listCursor = await conn.db.collection("temp-
```

```
fs.files").find({"metadata.owner": ObjectID(currentUserID)});

        for await (const currentFile of listCursor) {

            await reencryptFile(currentFile, newEncrpytionKey, currentUser);
        }

        progressBar.increment()
    }

    progressBar.stop();
}

const generateEncryptionKeys = async(user) => {

    const userPassword = user.password;
    const masterPassword = env.newKey;

    const randomKey = crypto.randomBytes(32);

    const iv = crypto.randomBytes(16);
    const USER_CIPHER_KEY =
crypto.createHash('sha256').update(userPassword).digest();
    const cipher = crypto.createCipheriv('aes-256-cbc', USER_CIPHER_KEY,
iv);
    let encryptedText = cipher.update(randomKey);
    encryptedText = Buffer.concat([encryptedText, cipher.final()]);

    const MASTER_CIPHER_KEY =
crypto.createHash('sha256').update(masterPassword).digest();
    const masterCipher = crypto.createCipheriv('aes-256-cbc',
MASTER_CIPHER_KEY, iv);
    let masterEncryptedText = masterCipher.update(encryptedText);
    masterEncryptedText = Buffer.concat([masterEncryptedText,
masterCipher.final()]).toString("hex");

    user.privateKey = masterEncryptedText;
    user.publicKey = iv.toString("hex");

    return user;
}

const getOldEncryptionKey = (user) => {

    const userPassword = user.password;
    const masterEncryptedText = user.privateKey;
    const masterPassword = env.key;
    const iv = Buffer.from(user.publicKey, "hex");

    const USER_CIPHER_KEY =
crypto.createHash('sha256').update(userPassword).digest();
    const MASTER_CIPHER_KEY =
crypto.createHash('sha256').update(masterPassword).digest();

    const unhexMasterText = Buffer.from(masterEncryptedText, "hex");
    const masterDecipher = crypto.createDecipheriv('aes-256-cbc',
MASTER_CIPHER_KEY, iv)
```

```
    let masterDecrypted = masterDecipher.update(unhexMasterText);
    masterDecrypted = Buffer.concat([masterDecrypted,
masterDecipher.final()])

    let decipher = crypto.createDecipheriv('aes-256-cbc', USER_CIPHER_KEY,
iv);
    let decrypted = decipher.update(masterDecrypted);
    decrypted = Buffer.concat([decrypted, decipher.final()]);

    return decrypted;
}

const getEncryptionKey = (user) => {

    const userPassword = user.password;
    const masterEncryptedText = user.privateKey;
    const masterPassword = env.newKey;
    const iv = Buffer.from(user.publicKey, "hex");

    const USER_CIPHER_KEY =
crypto.createHash('sha256').update(userPassword).digest();
    const MASTER_CIPHER_KEY =
crypto.createHash('sha256').update(masterPassword).digest();

    const unhexMasterText = Buffer.from(masterEncryptedText, "hex");
    const masterDecipher = crypto.createDecipheriv('aes-256-cbc',
MASTER_CIPHER_KEY, iv)
    let masterDecrypted = masterDecipher.update(unhexMasterText);
    masterDecrypted = Buffer.concat([masterDecrypted,
masterDecipher.final()])

    let decipher = crypto.createDecipheriv('aes-256-cbc', USER_CIPHER_KEY,
iv);
    let decrypted = decipher.update(masterDecrypted);
    decrypted = Buffer.concat([decrypted, decipher.final()]);

    return decrypted;
}

const findUsers = async() => {

    const listCursor = await conn.db.collection("temp-users").find({});
    const listCount = await conn.db.collection("temp-
users").find({}).count();
    const progressBar = new cliProgress.SingleBar({},
cliProgress.Presets.shades_classic);

    progressBar.start(listCount, 0);

    for await (const currentUser of listCursor) {

       try {

        const newUser = await generateEncryptionKeys(currentUser);

        await conn.db.collection("users").insertOne(newUser);
```

```
        progressBar.increment()

        } catch (e) {
            console.log("e", e);
        }
    }

    progressBar.stop();

}
```

ДОДАТОК В

Приклад використання модуля “crypto”

```
const getKey = async() => {

   if (process.env.KEY) {

        env.key = process.env.KEY

   } else if (process.env.NODE_ENV) {

        let password = await prompt("Enter Server Encryption Password:
", {method: "hide"});

        password =
crypto.createHash("md5").update(password).digest("hex");

        env.key = password;

   } else {

        let password = "1234";

        password =
crypto.createHash("md5").update(password).digest("hex");

        env.key = password;
   }
}

module.exports = getKey;
```

ДОДАТОК Г

Приклад використання модуля “crypto”

```
const createThumbnail = async(file, filename, user, newKey) => {

   return new Promise((resolve) => {

        try {

        const password = newKey;

        let CIPHER_KEY =
crypto.createHash('sha256').update(password).digest()

        let bucket = new mongoose.mongo.GridFSBucket(conn.db, {
            chunkSizeBytes: 1024 * 255,
        })

        const readStream = bucket.openDownloadStream(ObjectID(file._id))

        readStream.on("error", (e) => {
            console.log("File service upload thumbnail error", e);
            resolve(file);
        })

        const decipher = crypto.createDecipheriv('aes256', CIPHER_KEY,
file.metadata.IV.buffer);

        decipher.on("error", (e) => {
            console.log("File service upload thumbnail decipher error", e);
            resolve(file)
         })

            const concatStream = concat(async(bufferData) => {

                    const thumbnailIV = crypto.randomBytes(16);

                    const thumbnailCipher = crypto.createCipheriv("aes256",
CIPHER_KEY, thumbnailIV);

                    bufferData = Buffer.concat([thumbnailIV,
thumbnailCipher.update(bufferData), thumbnailCipher.final()]);

                    const thumbnailModel = new Thumbnail({name: filename,
owner: user._id, data: bufferData});

                    await thumbnailModel.save();

                    let updatedFile = await conn.db.collection("fs.files")
                        .findOneAndUpdate({"_id": file._id}, {"$set":
{"metadata.hasThumbnail": true, "metadata.thumbnailID":
thumbnailModel._id}})

                    updatedFile = updatedFile.value;
```

```
                    updatedFile = {...updatedFile, metadata:
{...updatedFile.metadata, hasThumbnail: true, thumbnailID:
thumbnailModel._id}}

                    resolve(updatedFile);

                    }).on("error", (e) => {
                        console.log("File service upload concat stream
error", e);
                        resolve(file);
                    })

                    const imageResize = sharp().resize(300).on("error", (e)
=> {

                        console.log("resize error", e);
                        resolve(file);
                    })

            readStream.pipe(decipher).pipe(imageResize).pipe(concatStream);


        } catch (e) {

            console.log(e);
            resolve(file);
        }

    })
}

module.exports = createThumbnail;
```