Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Бакалаврська дипломна робота на тему:

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОКРАЩЕННЯ РОБОТИ З ХМАРНИМИ СХОВИЩАМИ

Виконав: студент 4 курсу, групи 2КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)

Пронь В. В.
(прізвище та ініціали)

Керівник: к. т. н., доцент каф. КН
Арсенюк І. Р.
(прізвище та ініціали)

« 07 » червня 2024 р

Рецензент: к. т. н., доцент каф. САІТ
Козачко О. М.
(прізвище та ініціали)

« 07 » червня 2024 р

Допущено до захисту
Зав. кафедри Яровий А. А.
« 07 » червня 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра Комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д. т. н. Яровий А. А.
"_07_" _червня_ 2024 року

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Проню Владиславу Валентиновичу

1. Тема роботи: Розробка програмного забезпечення для покращення роботи з хмарними сховищами.
Керівник роботи: Арсенюк Ігор Ростиславович, доцент.
затверджені наказом вищого навчального закладу від "11" березня 2024 року № 80
2. Термін подання студентом роботи 27.05.2024 р.
3. Вихідні дані до роботи
Можливість зберігання не менше ніж 1000 файлів.
Максимальний обсяг завантажених файлів 10 ГБ.
Мова програмування – об'єктно орієнтована.
Наявність графічного інтерфейсу.
Операційна система Windows.
4. Зміст текстової частини (перелік питань, які потрібно розробити):
Аналіз можливостей сучасних хмарних сховищ; проектування програмного забезпечення для покращення роботи з хмарними сховищами; реалізація програмного забезпечення для покращення роботи з хмарними сховищами; висновки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
– загальний алгоритм функціонування програмного забезпечення для покращення роботи з хмарними сховищами;
– діаграма варіантів використання розробленого програмного забезпечення;
– діаграма компонентів розробленого програмного забезпечення;
– діаграма розгортання розробленого програмного забезпечення;
– приклад роботи розробленого програмного забезпечення;

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Арсенюк І. Р., к. т. н., доцент		

7. Дата видачі завдання 07.03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Приміт
		початок	закінчення	
1	Аналіз можливостей сучасних хмарних сховищ	06.05.24	10.05.24	
2	Проектування програмного забезпечення для покращення роботи з хмарними сховищами	11.05.24	15.05.24	
3	Реалізація програмного забезпечення для покращення роботи з хмарними сховищами	16.05.24	26.05.24	
4	Тестування роботи розробленого програмного забезпечення	27.05.24	29.05.24	
5	Розробка інструкції користувача	30.05.24	03.06.24	
6	Оформлення матеріалів до захисту БДР	04.06.24	06.06.24	

Студент _____ Пронь В. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Арсенюк І. Р.
(підпис) (прізвище та ініціали

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 74 сторінок формату А4, на яких є 15 рисунків, 2 таблиці, список використаних джерел містить 21 найменування.

Метою дослідження є розширення функціональних можливостей роботи з хмарними сховищами.

Під час дослідження предметної області розглянуто особливості сучасних хмарних сховищ, а також обґрунтовано доцільність розробки відповідного програмного забезпечення. Здійснено проектування програмного забезпечення для покращення роботи з хмарними сховищами, наведено відповідний алгоритм роботи, а також структуру та взаємодію складових програмного забезпечення в цілому. Розроблено програмне забезпечення, обґрунтовано вибір мови програмування, фреймворку та бібліотека для розробки, наведено фрагменти лістингу окремих модулів а також представлено результати тестування розробленого програмного забезпечення.

Ключові слова: покращення роботи з хмарними сховищами, хмарне сховище, веб-розробки, хмарні технології, JavaScript, React.js.

ABSTRACT

The bachelor's thesis consists of 74 A4 pages, which have 15 figures, 2 tables, a list of sources used contains 21 titles.

The aim of the study is to expand the functionality of interactions with cloud storages.

During the study of the subject area, the features of the functionality of modern cloud storages were studied, as well as the feasibility of developing an appropriate software. Design of software for improving interactions with cloud storages was done, algorithm was given and also structure and interactions of parts of the software in the whole. Software was developed, choice of programming language, framework and library was justified, fragments of listing of some modules was provided and results of developed software testing were presented.

Keywords: Improving interactions with cloud storage, cloud storage, web-development, cloud technologies, JavaScript, React.js.

ЗМІСТ

ВСТУП

Актуальність досліджень. Хмарні сховища стають невід'ємною частиною сучасного цифрового життя, проте вони вимагають оптимізованих інструментів для ефективного управління та використання. Розробка програмного забезпечення, спрямованого на підвищення продуктивності та безпеки роботи з хмарними сховищами, набуває стратегічного значення. Забезпечуючи високий рівень функціональності, безпеки даних та оптимізації процесів, такі програми відповідають сучасним вимогам користувачів, а також допомагають уникнути можливих ризиків, пов'язаних з недбалою обробкою чутливої інформації в хмарному середовищі. Таким чином, розробка програмного забезпечення для хмарних сховищ є актуальною задачею у сучасній індустрії інформаційних технологій.

Метою дослідження є розширення функціональних можливостей роботи з хмарними сховищами. Це передбачає створення програмного продукту, який забезпечить ефективне управління, зберігання та обмін даними в хмарних сервісах, з метою підвищення продуктивності та зручності користувачів.

Об'єктом дослідження є процес взаємодії з хмарними сховищами.

Предметом дослідження є програмне забезпечення для покращення роботи з хмарними сховищами.

Задачі дослідження

1. Розглянути поточний стан ринку програмного забезпечення для хмарних сховищ та виокремити основні рішення, доступні користувачам.
2. Виконати аналіз функціональних можливостей, переваг, недоліків а також характеристик обраних програмних засобів.
3. Розробити загальний алгоритм роботи програмного забезпечення для покращення роботи з хмарними сховищами.
4. Розробити UML-діаграми роботи програмного забезпечення.

5. Обґрунтувати вибір мови програмування, фреймворків та бібліотек для розроблюваного програмного засобу
6. Виконати тестування роботи розробленого програмного забезпечення.

Апробація роботи.

Робота апробувалась на наступній конференції:

XIII International Scientific and Practical Conference INTERNATIONAL FORUM: PROBLEMS AND SCIENTIFIC SOLUTIONS held on June 6-8, 2024 in Melbourne, Australia. Доповідь: «Обґрунтування вибору інструментів для роботи з хмарними сховищами даних» [1].

Публікації: електронні тези конференції XIII International Scientific and Practical Conference INTERNATIONAL FORUM: PROBLEMS AND SCIENTIFIC SOLUTIONS held on June 6-8, 2024 in Melbourne, Australia [1].

1 АНАЛІЗ МОЖЛИВОСТЕЙ СУЧАСНИХ ХМАРНИХ СХОВИЩ

1.1 Обґрунтування актуальності розробки програмного забезпечення для покращення роботи з хмарними сховищами

Хмарні сховища (cloud storage) [2] стали невід’ємною складовою сучасного інформаційного простору, втілюючи в собі потужні можливості зберігання та обробки даних. За останні кілька років ці технології пережили бурхливий розвиток, ставши не лише надійними альтернативами традиційним локальним сховищам, але й важливими каталізаторами цифрової трансформації в різних сферах життєдіяльності. Їхня популярність пояснюється не лише спрощенням доступу до даних, але й ефективністю їхньої організації, забезпеченням безпеки та масштабованістю. У цьому контексті аналіз сучасних хмарних сховищ стає надзвичайно важливим завданням, спрямованим на розуміння їхнього впливу на сучасне суспільство та перспектив їхнього розвитку.

У сучасному цифровому світі, де обсяги даних стрімко зростають, а потреба в їх зберіганні та обробці постійно збільшується, хмарні сховища стають ключовим елементом інфраструктури для багатьох користувачів, від приватних осіб до бізнесу. Хмарні сховища стали незамінним інструментом для роботи та особистого життя, забезпечуючи зручний доступ до даних з будь-якого пристрою, з будь-якого місця, за умови наявності інтернет-з'єднання.

Однак з поширенням хмарних сховищ з'являються нові виклики та загрози, пов'язані з безпекою, конфіденційністю та доступністю даних. Одним з ключових викликів є забезпечення відповідності нормативним вимогам, таким як Загальний регламент про захист даних (GDPR) Європейського Союзу, який посилює вимоги до обробки та зберігання персональних даних.

Таким чином, аналіз сучасних хмарних сховищ та їхніх можливостей, виявлення потенційних ризиків та шляхів їх усунення є важливим завданням для того, щоб

відповідати сучасним вимогам та допомогти користувачам покращити якість зберігання та обробки даних у всіх сферах.

Актуальності розробки:

1. Обсяг даних що генеруються і зберігаються, за останні роки надзвичайно збільшився. Це стосується як персональних даних користувачів, так і даних компаній, організацій та асоціацій. Ефективне зберігання, доступ і захист цих даних стали ключовим викликом.

Масштаби та обсяги даних, що генеруються та зберігаються, досягли безпрецедентного рівня. Глобальна інформаційна платформа, що постійно розширюється, містить величезні обсяги цифрових даних, які надходять з різних джерел, включаючи мобільні пристрої, датчики Інтернету речей (IoT), веб-додатки, соціальні мережі, онлайн-транзакції, наукові дослідження та медичні записи.

Наприклад, за даними IDC, до 2025 року очікується, що обсяг цифрових даних збільшиться до понад 175 зеттабайт [6]. Зокрема, значна частина цього обсягу припадає на дані, що генеруються та зберігаються в хмарних сховищах. Це пояснюється тим, що більшість користувачів та організацій обирають хмарні платформи для зберігання своїх даних через їхню зручність, доступність та масштабованість. Крім того, використання Інтернету речей та розвиток сенсорних технологій означає, що обсяг даних продовжує стрімко зростати. Наприклад, сучасні мобільні пристрої здатні зберігати та обробляти величезні обсяги фотографій, відео та інших мультимедійних даних, збільшуючи загальний обсяг цифрової інформації.

2. Мобільність і віддалена робота: зростаюча кількість співробітників, які працюють віддалено, і потреба в легкому доступі до даних з будь-якого місця роблять хмарні сховища важливими як для бізнесу, так і для приватних осіб.

Мобільність та віддалена робота - два ключові аспекти сучасної роботи та приватного життя, які визначають сучасні тенденції в організації праці та використанні технологій.

Сучасні технології дозволяють користувачам бути мобільними і працювати та отримувати доступ до даних будь-де і будь-коли. Це означає, що вони можуть працювати не лише на робочому місці, але й в інших місцях, таких як кафе, громадські місця, транспорт або навіть під час подорожей. Така мобільність підвищує продуктивність та ефективність, оскільки користувачі можуть працювати в зручному та комфортному для них середовищі.

Дистанційна робота - це широкий термін, який описує ситуації, коли працівники виконують свою роботу у віддаленому місці, а не в традиційному офісі. Працівники можуть працювати з обраного ними місця, наприклад, з домашнього офісу або коворкінгу. Дистанційна робота стає все більш поширеною завдяки технологічному прогресу, який дозволяє віддаленим командам спілкуватися і співпрацювати, контролювати і організовувати свій робочий процес. Це дозволяє людям працювати з комфортом, підвищує задоволеність працівників і знижує витрати на офісні приміщення та інфраструктуру.

3. Кібербезпека: оскільки загрози кібербезпеки постійно зростають, забезпечення безпеки даних є ключовим викликом. Аналіз сучасних хмарних сховищ з точки зору кібербезпеки є особливо важливим.

Кібербезпека стала однією з найважливіших тем у сучасному цифровому суспільстві. Загрози кібербезпеці з кожним роком стають все більш складними та витонченими, а захист від них вимагає постійного оновлення та вдосконалення стратегій, технологій та практик.

Одним з найважливіших аспектів кібербезпеки є захист від кібератак. Кіберзлочинці завжди шукають нові способи проникнення в системи та мережі організації, щоб викрасти дані, вимагати викуп або завдати іншої шкоди. Важливо розробити та впровадити ефективні заходи захисту, такі як мережева безпека, моніторинг активності, шифрування даних та використання сучасного антивірусного та антишпигунського програмного забезпечення.

Іншим важливим аспектом кібербезпеки є захист персональних даних. Загальний регламент Європейського Союзу про захист даних (GDPR) та інші подібні

законодавчі акти посилили вимоги до зберігання та обробки персональних даних. Тому організаціям необхідно ретельно контролювати доступ до даних, забезпечувати їх безпечне зберігання та передачу, а також вживати заходів для запобігання витоку та несанкціонованому використанню даних.

Крім того, зростаюче використання хмарних технологій також створює виклики для кібербезпеки. Важливо переконатися, що хмарні платформи, які використовуються для зберігання та обробки даних, відповідають вимогам безпеки та конфіденційності, а також забезпечити захист від потенційних кібератак на ці платформи.

Таким чином, кібербезпека в сучасному цифровому середовищі вимагає комплексного підходу та постійного оновлення стратегій і захисних заходів для ефективного запобігання кіберзагрозам і забезпечення безпеки та конфіденційності даних.

4. Штучний інтелект та аналітика: розробки в галузі штучного інтелекту та аналізу даних створюють нові можливості для використання даних хмарних сховищ у різних сферах, зокрема в бізнесі, медицині та науці.

Штучний інтелект (ШІ) та аналітика даних - це дві ключові тенденції, які спонукають сучасні технології та промисловість оптимізувати процеси, приймати рішення та досягати вищих рівнів продуктивності.

Штучний інтелект використовується для розробки систем і алгоритмів, які можуть виконувати завдання, що традиційно вважаються сферою людського інтелекту. Сюди входить машинне навчання, глибоке навчання, обробка природної мови, комп'ютерний зір і багато інших. ШІ використовується для автоматизації процесів, аналізу даних, прогнозування тенденцій і вирішення складних завдань у різних сферах, включаючи охорону здоров'я, фінанси, технології та транспорт.

Аналіз даних використовується для виявлення закономірностей, тенденцій і корисної інформації з великих обсягів даних. Він включає в себе ряд методів, таких як статистичний аналіз, машинне навчання та візуалізація даних. Аналітика даних

дозволяє організаціям отримувати цінну інформацію з даних для прийняття кращих рішень та оптимізації своєї діяльності.

Як ШІ, так і аналітика даних відіграють важливу роль у вдосконаленні бізнес-процесів, виявленні нових можливостей і підвищенні ефективності в різних сферах діяльності ШІ та аналітика даних надають інструменти для прийняття обґрунтованих рішень на основі даних і прогнозування майбутніх тенденцій, щоб і тим самим сприяють підвищенню конкурентоспроможності організацій.

1.2 Огляд та аналіз сучасних хмарних сховищ

Сучасний аналіз хмарних сховищ – це комплексне дослідження та оцінка технологій, сервісів і функцій, які пропонують постачальники хмарних послуг. Ключові аспекти аналізу включають оцінювання основних можливостей, кібербезпеки, доступності та масштабованості, вартості, підтримки та сервісу хмарних сховищ.

Першим кроком в аналізі є вивчення ринку хмарних послуг та визначення постачальників хмарних сховищ. Їх є надзвичайно велика кількість, багато з них є досить різними і мають різні можливості, однак найтиповішими та найпопулярнішими є такі 3: Amazon Web Services (AWS) [3], Microsoft Azure [4] та Google Cloud Platform (GCP) [5].

На на ступному кроці доцільно оцінити функціональність хмарних сховищ. Це включає аналіз різних параметрів, таких як ємність сховища, можливості резервного копіювання та відновлення даних, доступ до даних з різних платформ і пристроїв, спільна робота з документами та інтеграція з іншими сервісами.

Також особливу увагу варто приділити важливому питанню кібербезпеки. Це містить оцінку заходів захисту даних, шифрування, аутентифікацію користувачів, захист від кібератак та відновлення після інцидентів безпеки.

Ще одним важливим аспектом є оцінка вартості та моделей ціноутворення провайдерів хмарних сховищ. Сюди входить аналіз вартості зберігання даних, передачі даних, обчислювальних ресурсів та інших послуг.

Зрештою, проаналізувавши найновіші хмарні сховища, користувачі та організації можуть вибрати постачальника хмарних послуг, який найкраще відповідає їхнім потребам, вимогам та бюджету.

AWS [3]:

AWS — це комплексна хмарна платформа, яка надає широкий спектр інфраструктурних послуг, таких як обчислювальні потужності, зберігання даних, бази даних, машинне навчання, аналітика та багато іншого. AWS була створена компанією Amazon і вперше представлена публіці у березні 2006 року. Платформа стала популярною завдяки своїй надійності, масштабованості та широкому спектру послуг, що дозволяють компаніям та розробникам швидко та ефективно створювати і керувати різноманітними додатками та сервісами.

Основні можливості:

AWS пропонує понад 200 послуг у своєму портфоліо.

1. Обчислювальні послуги: Amazon EC2 (Elastic Compute Cloud), Amazon Lambda, Amazon Elastic Container Service (ECS).
2. Зберігання даних: Amazon S3 (Simple Storage Service), Amazon EBS (Elastic Block Store), Amazon Glacier.
3. Бази даних: Amazon RDS (Relational Database Service), Amazon DynamoDB (NoSQL Database Service), Amazon Redshift (Data Warehouse Service).
4. Штучний інтелект та машинне навчання: Amazon SageMaker, Amazon Rekognition, Amazon Comprehend.

Інші послуги включають аналітичні, мережеві, розробницькі та інші.

Кібербезпека:

1. AWS витрачає мільярди доларів на рік на кібербезпеку.
2. AWS забезпечує рівень доступності для більшості своїх сервісів у 99.99% (4 дев'ятки) [7].

3. Шифрування даних: 256-бітне шифрування для даних у спокої (data-at-rest) та під час передачі (data-in-transit).
4. AWS Identity and Access Management (IAM) надає гнучкий контроль доступу до ресурсів.

Доступність та масштабованість:

1. AWS має 80 центрів обробки даних (регіонів) у всьому світі.
2. Спеціальні зони доступності (Availability Zones) в кожному регіоні забезпечують високу доступність та стійкість до відмов.
3. AWS масштабується від індивідуальних користувачів до великих корпорацій та урядових установ.

Вартість:

1. Плата за використання: Amazon EC2 може коштувати від $0.0058 за годину.
2. Ціни на зберігання: Amazon S3 може коштувати від $0.023 за гігабайт на місяць.
3. Існує безкоштовний рівень обслуговування для нових користувачів, який дозволяє безкоштовно використовувати деякі послуги протягом перших 12 місяців.

Підтримка та сервіс:

1. AWS пропонує різні рівні підтримки, включаючи безкоштовну та платну підтримку.
2. Базова безкоштовна підтримка включає доступ до документації та форумів.
3. Рівні платної підтримки надають пріоритетну технічну підтримку, консультації експертів та інші переваги.

Отже, AWS є повноцінною та розширеною платформою, яка надає широкий спектр послуг з високим рівнем надійності, безпеки та масштабованості за конкурентоспроможні ціни.

Основними недоліками є:

1. Складні для розуміння цінові моделі через велику кількість різноманітних варіантів і тарифів. Це значно ускладнює прогнозування витрат.

2. Велика вартість при масштабуванні. Зазвичай AWS є економічно вигідним на початкових етапах розвитку програмного забезпечення, однак при збільшені кількості даних вартість може зростати диспропорційно.

Microsoft Azure [4]:

Microsoft Azure — це комплексна хмарна платформа, яка надає широкий спектр послуг, включаючи обчислювальні потужності, зберігання даних, бази даних, аналітику, штучний інтелект, і багато іншого. Azure була розроблена компанією Microsoft і офіційно запущена у лютому 2010 року. Платформа стала популярною завдяки своїй інтеграції з іншими продуктами Microsoft, високій надійності, масштабованості та широкому спектру інструментів, що дозволяють компаніям ефективно розгортати, керувати та розвивати свої додатки та сервіси в хмарному середовищі.

Основні можливості:

Microsoft Azure пропонує понад 200 послуг та продуктів у своєму портфоліо.

1. Обчислювальні послуги: віртуальні машини, контейнери, функції Azure Functions.
2. Зберігання даних: Azure Blob Storage, Azure Files, Azure Disk Storage.
3. Бази даних: Azure SQL Database, Azure Cosmos DB, Azure Database for MySQL, PostgreSQL та інші.
4. Штучний інтелект та машинне навчання: Azure Machine Learning, Cognitive Services (Computer Vision, Speech Recognition, Natural Language Processing) та інші.

Кібербезпека:

1. Microsoft Azure забезпечує високий рівень кібербезпеки та відповідає різним сертифікатам та стандартам безпеки, таким як ISO 27001, SOC 1, SOC 2, HIPAA, GDPR тощо.
2. Шифрування даних: AES 256-бітне шифрування для даних у спокої та під час передачі.

3. Azure Active Directory (Azure AD) для керування доступом та автентифікацією користувачів.

Доступність та масштабованість:

1. Microsoft Azure має 60+ регіональних центрів обробки даних у всьому світі.
2. Кожний регіон Azure має певну кількість зон доступності, що забезпечують високу доступність та стійкість до відмов.
3. Azure Autoscale дозволяє автоматично масштабувати ресурси в залежності від навантаження.

Вартість:

1. Плата за використання: Вартість віртуальних машин починається від $0.008 за годину.
2. Ціни на зберігання: Вартість Azure Blob Storage починається від $0.0184 за гігабайт на місяць.
3. Існує безкоштовний тарифний план для нових користувачів, який надає обмежений обсяг послуг протягом перших 12 місяців.

Підтримка та сервіс:

1. Microsoft Azure пропонує різні рівні підтримки, включаючи безкоштовну та платну підтримку.
2. Платна підтримка включає технічну підтримку 24/7, консультації експертів, пріоритетний доступ до служб та інші переваги.

Основними недоліками є:

1. Часті оновлення і зміни в сервісах Azure можуть вимагати постійного моніторингу та адаптації з боку користувачів, що додає додаткових зусиль з адміністрування.
2. Деякі регіони можуть мати обмежену підтримку або меншу кількість доступних сервісів, що може вплинути на продуктивність і доступність.

Google Cloud Platform [6]:

Google Cloud Platform — це комплексна хмарна платформа, яка надає широкий спектр послуг, включаючи обчислювальні потужності, зберігання даних, бази даних,

аналітику, машинне навчання, і багато іншого. Платформа була розроблена компанією Google і офіційно запущена у квітні 2008 року. Google Cloud Services відзначається високою продуктивністю, надійністю, глобальною мережею дата-центрів та інтеграцією з іншими сервісами Google. Вона дозволяє компаніям та розробникам ефективно створювати, розгортати та керувати різноманітними додатками та сервісами в хмарному середовищі.

Основні можливості:

Google Cloud Platform пропонує більше 100 послуг та продуктів у своєму портфоліо.

1. Обчислювальні послуги: Google Compute Engine, Google Kubernetes Engine (GKE), Google App Engine.
2. Зберігання даних: Google Cloud Storage, Google Cloud SQL, Google Cloud Bigtable.
3. Бази даних: Google Cloud SQL, Google Cloud Bigtable, Google Cloud Spanner (глобальна розподілена база даних).
4. Штучний інтелект та машинне навчання: Google Cloud AI, Google Cloud Machine Learning Engine, Google Cloud Vision API, Google Cloud Natural Language API.

Кібербезпека:

1. Google Cloud Platform відповідає найвищим стандартам безпеки, таким як ISO 27001, SOC 2, HIPAA, PCI DSS.
2. Шифрування даних: Google Cloud Platform використовує шифрування AES 256-біт для даних у спокої та під час передачі.
3. Google Cloud Identity & Access Management (IAM) для керування доступом та автентифікацією користувачів.

Доступність та масштабованість:

1. Google Cloud Platform має 24 регіональних центри обробки даних та понад 100 зон доступності у всьому світі.
2. Це забезпечує високу доступність та стійкість до відмов, а також низьку затримку.

3. Google Cloud Autoscaling дозволяє автоматично масштабувати ресурси в залежності від навантаження.

Вартість:

1. Плата за використання: Вартість віртуальних машин починається від $0.0104 за годину.
2. Ціни на зберігання: Вартість Google Cloud Storage починається від $0.020 за гігабайт на місяць.
3. Існує безкоштовний тарифний план для нових користувачів, який дозволяє безкоштовно використовувати деякі послуги протягом перших 12 місяців.

Підтримка та сервіс:

1. Google Cloud Platform пропонує різні рівні підтримки, передбачаючи безкоштовну та платну підтримку.
2. Платна підтримка передбачає технічну підтримку 24/7, доступ до фахівців, а також гарантований час відклику.

Основними недоліками є:

1. Менш розвинена екосистема інструментів і бібліотек в порівнянні з AWS або Azure
2. Деякі регіони можуть мати обмежену підтримку або меншу кількість доступних сервісів, що може вплинути на продуктивність і доступність.

Порівняння основних проаналізованих хмарних сховищ показано в таблиці 1.1, переваг та недоліків – у таблиці 1.2.

Таблиця 1.1 – Співставлення існуючих хмарних сховищ

Характеристика	**AWS**	**Microsoft Azure**	**Google Cloud Platform**
Кількість послуг	Понад 200 послуг	Понад 200 послуг	Понад 100 послуг

Таблиця 1.1 – Продовження

Характеристика	**AWS**	**Microsoft Azure**	**Google Cloud Platform (GCP)**
Обчислювальні послуги	EC2 Lambda, ECS	Compute Engine, Kubernetes	Compute Engine, Kubernetes
Зберігання даних	S3, EBS, Glacier	Blob Storage, Disk Storage, Files	Cloud Storage, Cloud SQL, Cloud Bigtable
Бази даних	RDS, DynamoDB, Redshift	SQL Database, Cosmos DB, Spanner	SQL Database, Bigtable, Spanner
Штучний інтелект	SageMaker, Rekognition, Comprehend	Azure AI, Machine Learning, Vision API	AI, Machine Learning, Vision API
Кібербезпека	ISO 27001, SOC 1/2, HIPAA, GDPR	ISO 27001, SOC 1/2, HIPAA, PCI DSS	ISO 27001, SOC 1/2, HIPAA, PCI DSS
Центри обробки даних	80 регіонів, 245 зон доступності	60+ регіонів, 160 зон доступності	24 регіони, 100+ зон доступності
Шифрування даних	AES 256-бітне	AES 256-бітне	AES 256-бітне
Плата за використання	Починається від $0.0058 за годину	Починається від $0.008 за годину	Починається від $0.0104 за годину

Таблиця 1.1 – Продовження

Характеристика	**AWS**	**Microsoft Azure**	**Google Cloud Platform (GCP)**
Підтримка	Різні рівні підтримки, включаючи безкоштовну	Різні рівні підтримки, включаючи безкоштовну	Різні рівні підтримки, включаючи безкоштовну
Технічна підтримка	24/7	24/7	24/7

Таблиця 1.2. – Переваги та недоліки сервісів

Характерис-тика	**AWS**	**Microsoft Azure**	**Google Cloud Platform (GCP)**
Переваги	Найбільший вибір послуг та продуктів	Широка інтеграція з існуючими корпоративними рішеннями	Передові технології у сфері штучного інтелекту та машинного навчання
	Найбільша глобальна інфраструктура з 80+ регіонами та 245+ зонами доступності	Широкий спектр послуг для розробників та аналітики	Велика географічна присутність з 24 регіонами і 100+ зонами доступності

Таблиця 1.2 – Продовження

Характеристика	**Amazon Web Services (AWS)**	**Microsoft Azure**	**Google Cloud Platform (GCP)**
Недоліки	Важка перебачуваність витрат через складну систему ціноутворення	Необхідність додаткових зусиль з адміністрування через часті оновлення	Менш розвинена екосистема в порівнянні з AWS або Azure
	Велика вартість при масштабуванні	Обмежені послуги та підтримка для деяких регіонів	Обмежені послуги та підтримка для деяких регіонів

Порівнявши Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP), можна визначити деякі ключові відмінності та переваги кожного сервісу.

1. Функціональність:

- AWS має найбільший вибір послуг та продуктів у своєму портфоліо, включаючи широкий спектр інструментів для розробки, аналітики та машинного навчання.
- Microsoft Azure має сильний фокус на інтеграцію з існуючими корпоративними рішеннями, а також широкий спектр послуг для розробників та аналітики.
- Google Cloud Platform відомий своїми передовими технологіями штучного інтелекту та машинного навчання, а також потужними інструментами для аналізу даних.

2. Кібербезпека:

- У всіх трьох платформ високий рівень кібербезпеки забезпечується за рахунок шифрування даних, контролю доступу та відповідності стандартам безпеки.
- AWS та Microsoft Azure мають вражаючий набір сертифікатів безпеки та відповідають вимогам різних галузевих стандартів.
- Google Cloud Platform також надає високий рівень безпеки, але відомий своєю передовою розробкою алгоритмів машинного навчання для виявлення загроз та інцидентів.

3. Доступність та масштабованість:

- AWS має найбільшу глобальну інфраструктуру з більш ніж 80 регіональними центрами обробки даних та 245 зонами доступності.
- Microsoft Azure та Google Cloud Platform також мають велику географічну присутність, але меншу кількість регіонів порівняно з AWS.
- Всі три платформи можуть ефективно масштабуватися вгору та вниз в залежності від потреб користувачів.

4. Вартість:

- Ціни на послуги можуть змінюватися в залежності від регіону та обсягу використання.
- AWS, Microsoft Azure та Google Cloud Platform пропонують схожі моделі ціноутворення, включаючи плату за використання та різні пакетні пропозиції.

5. Підтримка та сервіс:

- Усі три платформи надають різні рівні підтримки, включаючи безкоштовну та платну підтримку з можливістю доступу до експертів та інженерів 24/7.

Отже, вибір найкращої платформи залежить від конкретних потреб, галузі діяльності та бюджету організації. Великі компанії можуть віддавати перевагу AWS або Azure через їхню широку функціональність та підтримку корпоративних потреб, тоді як компанії, які активно використовують технології штучного інтелекту та машинного навчання, можуть віддати перевагу GCP.

1.3 Постановка задачі дослідження

У процесі дослідження було проаналізовано та визначено інформацію щодо існуючих хмарних сховищ, їх роботи та функціональних можливостей.

У ході дослідження виявлено переваги та недоліки сучасних хмарних сховищ, та було прийнято рішення зосередитися на розробці власного програмного забезпечення для роботи з хмарними сховищами що зможе використовувати вже існуючі хмарні сховища, а також інтегруватись в інше програмне забезпечення.

Така система буде досить універсальною, може бути використана як розширення до програми, що може використовувати будь-який сервіс хмарного сховища або повноцінне сховище, що сьогодні дуже ціниться в світі.

Відповідно розробка що буде розроблена в ході дослідження повинна повноцінно задовольняти розроблене технічне завдання та виконувати попередньо весь функціонал що зазначено, мати можливість забезпечення безпеки та використання інтерфейсу користувача.

Технічне завдання на розробку:

Основні функції:

1. Автентифікація та авторизація:
   - Створення системи реєстрації та входу для користувачів, забезпечення безпеки даних.
2. Зберігання даних:
   - Розроблення механізму завантаження та зберігання файлів будь-якого типу в хмарному сховищі.
3. Управління даними:
   - Реалізація функцій керування файлами та папками: копіювання, переміщення, видалення, перейменування.
   - Підтримка версіонування файлів та можливість відновлення попередніх версій.
4. Доступ до даних:

- Надання користувачам можливості спільного доступу до файлів з різними рівнями привілеїв (читання, запис, редагування).

5. Безпека та конфіденційність:

- Захист даних шляхом застосування механізмів шифрування та автентифікації для забезпечення конфіденційності та безпеки.

6. Моніторинг та аналітика:

- Розроблення функцій моніторингу використання сховища та надання аналітичних звітів.

7. Інтеграція з іншими сервісами:

- Підтримка інтеграції з іншими хмарними сервісами та додатками для розширення функціональності.

Технічні вимоги:

Операційна система: Windows.

Мова програмування: JavaScript.

Використання сучасних технологій для створення веб-додатків та роботи з API хмарних сервісів.

План роботи:

Проектування архітектури: Визначення структури додатку та вибір технологій.

Розробка функціональності: Реалізація основних функцій зберігання, управління та доступу до даних.

Імплементація безпеки: Впровадження механізмів шифрування та автентифікації користувачів.

Розробка інтерфейсу: Створення зручного та інтуїтивно зрозумілого інтерфейсу користувача.

Тестування та оптимізація: Проведення тестів для виявлення та виправлення помилок, а також оптимізація продукту для підвищення продуктивності та безпеки.

Реліз та підтримка: Випуск готового продукту та надання підтримки користувачам.

1.4 Висновок до розділу 1

Здійснено обґрунтування актуальності розробки програмного забезпечення для покращення роботи з хмарними сховищами. Визначено, що основними актуальностями є: великий обсяг даних, що створюється в сучасному світі, необхідність мобільності і можливості віддаленої робота, покращення кібербезпеки, розширення можливостей штучного інтелекту та аналітики.

Виконано огляд та аналіз існуючих сервісів хмарного зберігання даних, який показав, що ринок цих послуг наразі представлений великою кільістю продуктів, багато з них є досить різними і мають різні можливості, однак найтиповішими та найпопулярнішими є такі 3: Amazon Web Services (AWS) [3], Microsoft Azure [4] та Google Cloud Platform (GCP) [5], які і було розглянуто. Кожен з цих провайдерів пропонує широкий спектр послуг зі зберігання та обробки даних у хмарному середовищі; AWS відомий кількістю доступних сервісів та гнучкістю конфігурації, однак має складну систему ціноутворення. Azure, з іншого боку, пропонує глибоку інтеграцію з іншими продуктами Microsoft та широкий спектр інструментів для аналізу даних, але створює необхідність додаткових зусиль з адміністрування через часті оновлення, а GCP славиться своєю швидкістю та високою масштабованістю, але має менш розвинену екосистема в порівнянні з AWS або Azure.

Здійснено постановку задачі дослідження, де було визначено ключові функції, які розроблюване програмне забезпечення повинно реалізовувати, а саме: автентифікація та авторизація користувачів, зберігання та управління даними, безпека та конфіденційність, моніторинг та аналіз, а також інтеграція з іншими сервісами. Також програмне забезпечення повинне мати можливість перемикатись між режимами збереження файлів: за допомогою локального серверу або AWS.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОКРАЩЕННЯ РОБОТИ З ХМАРНИМИ СХОВИЩАМИ

2.1 Розробка алгоритму роботи програмного забезпечення для покращення роботи з хмарними сховищами

Алгоритм – це точна послідовність інструкцій, що виконуються для вирішення конкретного завдання або розв'язання конкретної проблеми [8].

Ефективність алгоритму – це його здатність вирішувати проблему за обмежений проміжок часу, використовуючи обмежені ресурси, такі як пам'ять і обчислювальна потужність.

Алгоритми можуть використовуватися для різних цілей, включаючи сортування даних, пошук найкоротшого шляху, оптимізацію завдань і обробку тексту.

При розробці алгоритмів важливо враховувати такі параметри, як час виконання (складність), використання пам'яті та стійкість даних.

Аналіз алгоритмів допомагає оцінити і порівняти ефективність алгоритмів за різними критеріями і вибрати найкращий алгоритм для конкретного завдання.

Для створення алгоирмту потрібно розібрати загальні частини додатку, тому проаналізувавши можна виділити наступні елементи:

1. Автентифікація користувача:
   - Першим кроком є автентифікація користувача. Додаток повинен надати можливість ввійти або створити обліковий запис, який буде використовуватися для доступу до хмарного сховища.
2. Вибір хмарного провайдера:
   - Користувач повинен мати можливість вибрати хмарного провайдера, з яким буде працювати додаток. Це може бути AWS, Microsoft Azure, Google Cloud або інший провайдер.
3. Авторизація з хмарним провайдером:

- Після вибору провайдера, додаток повинен отримати авторизаційні дані для взаємодії зі службами цього провайдера. Це може включати отримання токенів доступу або ключів API.

4. Взаємодія з API хмарного сховища:
   - Після успішної авторизації, додаток може використовувати API хмарного сховища для виконання операцій, таких як завантаження файлів, зберігання даних, видалення файлів тощо.
5. Керування файлами та даними:
   - Додаток повинен надати можливість користувачеві завантажувати, переглядати, редагувати та видаляти файли в хмарному сховищі. Це може включати створення нових папок, переміщення файлів та інші операції керування.
6. Обробка помилок та винятків:
   - Додаток повинен мати механізми обробки помилок та винятків для випадків невдалих операцій або проблем з підключенням до хмарного сховища.
7. Опції конфігурації та налаштувань:
   - Користувач повинен мати можливість налаштувати додаток, включаючи параметри, такі як мова інтерфейсу, дозволи доступу до файлів та інші опції.

Загальний алгоритм має такі кроки:

1. Початок
2. Запуск серверу: на цьому етапі відбувається запуск серверу, підключення його до бази даних та хмарних сховищ, після чого сервер відкривається для запитів
3. Отримання запиту: цей етап відбувається коли з графічного інтерфейсу, що є веб-сайтом, відправляється запит. На цьому етапі всі дані про запит серіалізуються, щоб їх можна було обробити на сервері.
4. Користувачу дозволена ця дія: на цьому етапі перевіряються права доступу користувача. Якщо він зовсім не аутентифікований, або немає достатніх прав

доступу до ресурсу за запитом, відбувається перехід до кроку 5, якщо йому надано доступ – перехід до кроку 6

5. Для користувача створюється повідомлення про помилку авторизації, після чого відбувається перехід до кроку 7
6. Запит користувача виконується відповідно до URL на який він був надісланий, та даних, що були отримані. Сервер звертається до своєї бази даних і, за потреби, до обраного хмарного сховища.
7. Сформована відповідь відправляється клієнту, тобто на веб-сторінку з інтерфейсом користувача.
8. Якщо подана команда на завершення роботи, то сервер зупиняє свою роботу, розриваючи існуючі підключення. Якщо команда не подана, повернення до кроку 3.

Відповідно до цього побудуємо загальний алгоритм функціонування програмного забезпечення, що зображений на рисунку 2.1:

Рисунок 2.1 – Загальний алгоритм функціонування програмного забезпечення для покращення роботи з хмарними сховищами

Далі створено детальну функціональну діаграму, що зображена на рисунку 2.2:

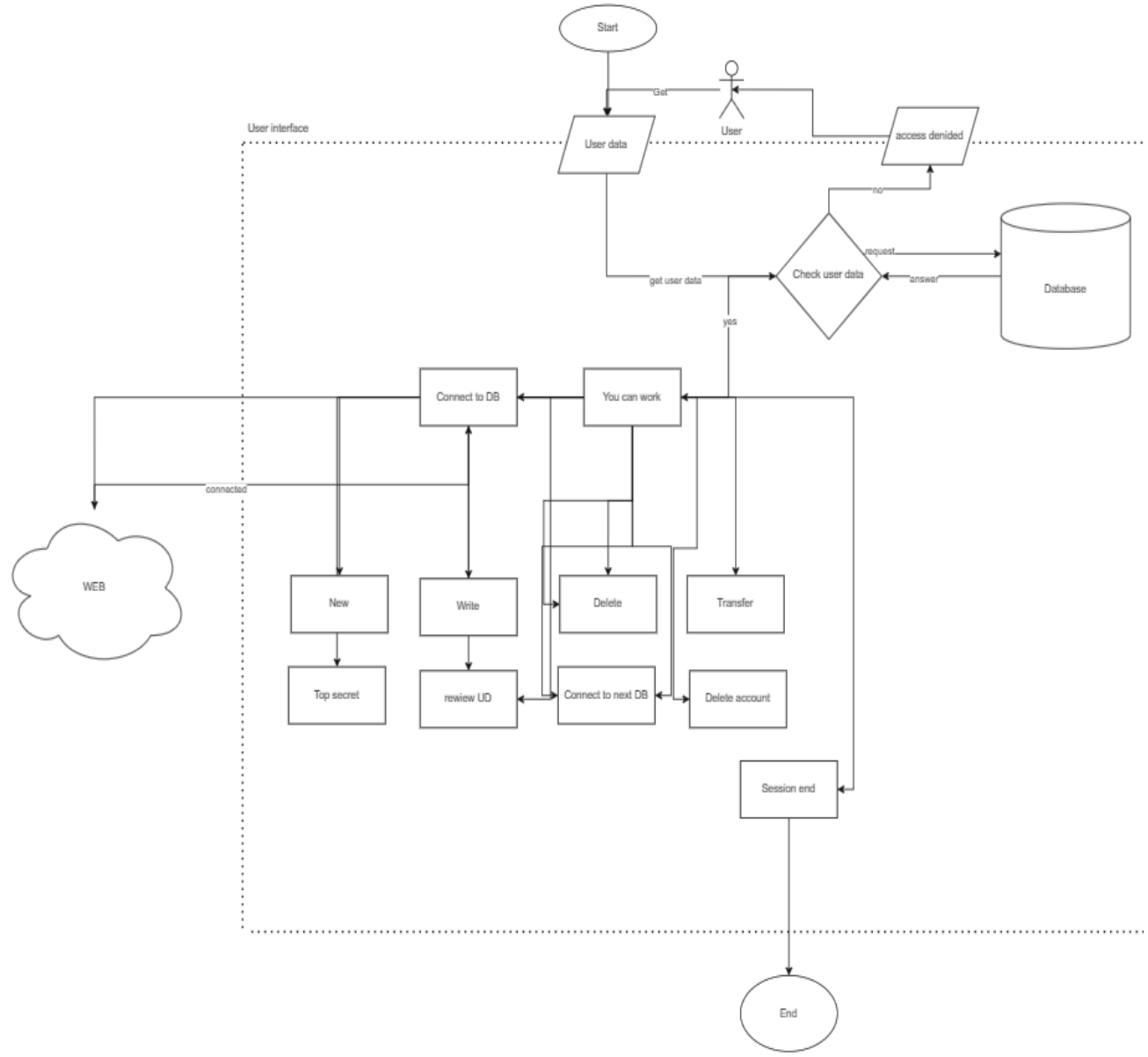

Рисунок 2.2 – Повна функціональна діаграма роботи програмного забезпечення

Відповідно до даної діаграми можна наступним чином пояснити функціонал:

1. Автентифікація та авторизація:
   - Додаток використовує протокол OAuth для взаємодії з платформою хмарного сховища. Після введення облікових даних користувача, запити авторизації передаються через захищений канал HTTPS.
2. Взаємодія з API платформи хмарного сховища:

- Додаток використовує HTTP-запити до API платформи для виконання операцій з файлами (наприклад, завантаження, оновлення, видалення). Всі запити передаються за допомогою стандартних HTTP-методів, таких як GET, POST, PUT та DELETE.

3. Синхронізація даних:

- Для автоматичної синхронізації даних між різними пристроями, додаток використовує техніку polling або можливості, що надаються самою платформою хмарного сховища для цієї цілі.

4. Захист інформації:

- Всі дані, що передаються між додатком і платформою хмарного сховища, шифруються за допомогою протоколу TLS (Transport Layer Security) для забезпечення конфіденційності та цілісності.

5. Кешування даних:

- Додаток може використовувати локальне сховище для кешування певних даних та зменшення кількості запитів до сервера. Це дозволяє зберігати копії недавно отриманих даних на пристрої користувача для швидкого доступу.

6. Моніторинг та журналювання:

- Додаток може вести журнал подій та використовувати моніторинг для відстеження проблем та оптимізації продуктивності.

2.2 Розробка структурних UML-діаграм для створення програмного забезпечення

Діаграма класів (Class Diagram): Ця діаграма відображає структуру програмного коду, включаючи класи, їх атрибути та методи, і зв'язки між ними [9]. Вона допоможе краще зрозуміти організацію програми та взаємозв'язки між її складовими частинами.

Дана діаграма висвітлює відповідно всі класи що потрібні для реалізації та розроблення відповідного додатку

Компоненти:

1. Клас "Користувач": Містить атрибути, такі як ім'я користувача, електронна пошта та пароль, методи автентифікації, авторизації та реєстрації.
2. Клас "Хмарне сховище": Представляє хмарне сховище, з яким взаємодіє користувач. Має методи для завантаження, збереження та керування файлами, а також для авторизації користувачів. Містить дані для підключення до хмарного сховища
3. Клас "Файл": Представляє окремий файл у хмарному сховищі. Має атрибути, такі як ім'я файлу, розмір та дата створення, а також методи для збереження та видалення файлів.
4. Клас "Дозвіл": Представляє права доступу до файлів у хмарному сховищі. Містить атрибути, які визначають рівень доступу, а також методи для надання та зміни дозволів.
5. Клас "Синхронізація": Містить методи для синхронізації даних між хмарним сховищем та пристроями користувача.

Зв'язки між цими класами:

1. Залежність між "Користувач" та "Хмарне сховище": Користувач використовує хмарне сховище для доступу до своїх файлів.
2. Композиція між "Хмарне сховище" та "Файл": Файли є частиною хмарного сховища і залежать від нього для збереження та управління.
3. Агрегація між "Хмарне сховище" та "Дозвіл": Дозволи на доступ до файлів пов'язані з хмарним сховищем і використовуються для керування доступом користувачів.
4. Залежність між "Користувач" та "Синхронізація": Користувач може ініціювати процес синхронізації даних між своїми пристроями та хмарним сховищем.

На рисунку 2.3 зображена розроблена діаграма класів.

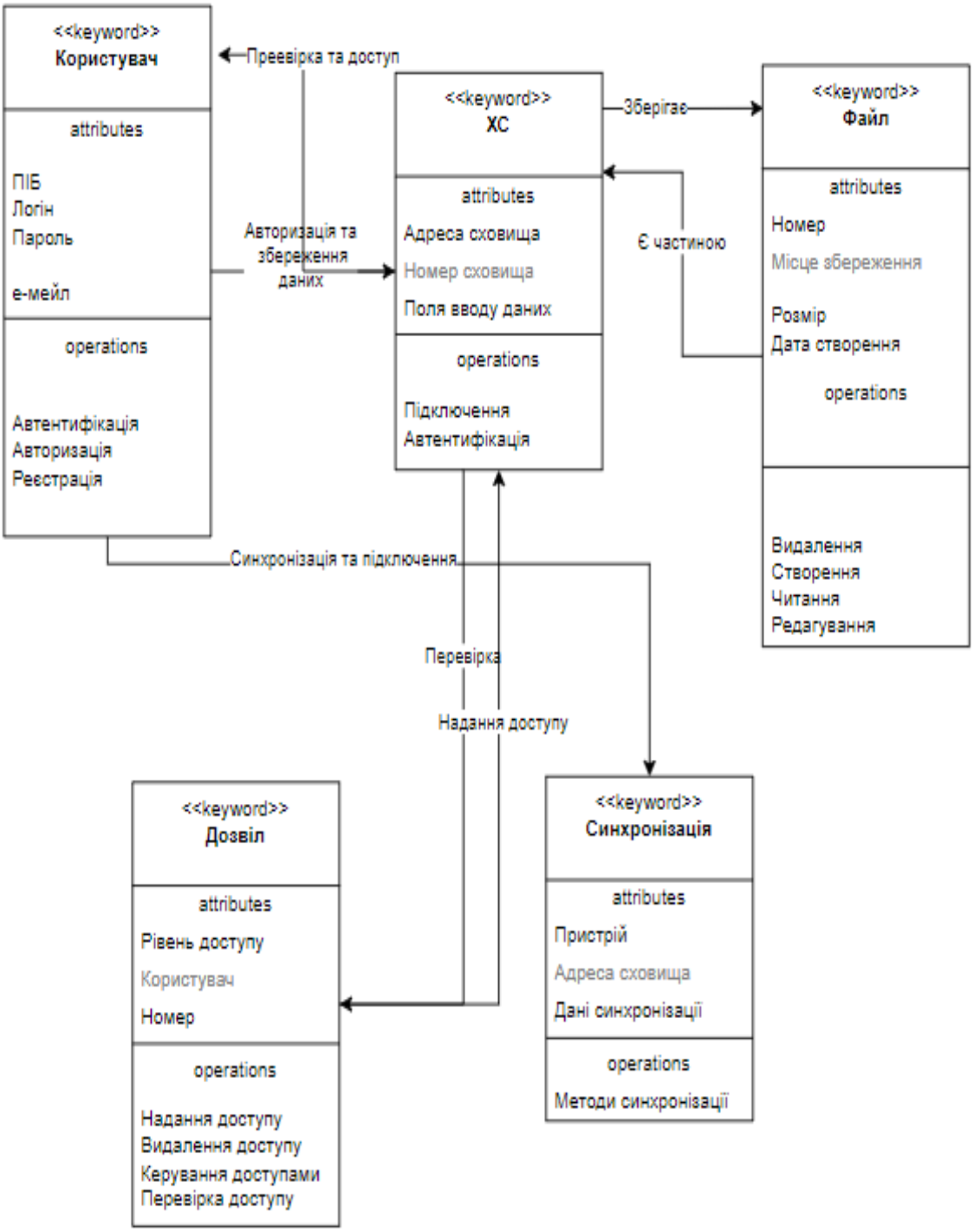

Рисунок 2.3 – Діаграма класів програмного забезпечення для роботи з хмарними сховищами

Діаграма компонентів (Component Diagram): Ця діаграма допомагає візуалізувати структуру програмної системи, показуючи компоненти, з яких вона складається, та зв'язки між ними [10]. Вона вказує, які частини системи відповідають за виконання різних функцій.

1. Користувацький інтерфейс (UI):
   - Цей компонент відповідає за візуальне взаємодію з користувачем.
   - Включає у себе елементи, такі як екрани, кнопки, поля введення тощо.
   - Забезпечує зручний та інтуїтивно зрозумілий інтерфейс для користувача.

2. Додаток (Application):
   - Це основний компонент, який містить бізнес-логіку системи.
   - Він координує взаємодію між іншими компонентами і обробляє запити користувача.
   - Може включати у себе різні модулі та сервіси, такі як сервіс авторизації, сервіс роботи з файлами тощо.

3. Хмарне сховище (Cloud Storage):
   - Цей компонент представляє собою зовнішнє хмарне сховище, з яким взаємодіє додаток.
   - Він надає можливість зберігання та керування файлами в хмарному середовищі.
   - Може мати власні сервіси авторизації та безпеки.

4. База даних (Database):
   - Це компонент, який використовується для зберігання даних про користувачів, файли, дозволи тощо.
   - Забезпечує доступ до даних для додатку та забезпечує їхню цілісність та безпеку.

5. Мережевий стек (Network Stack):
   - Цей компонент відповідає за мережеву взаємодію додатку з хмарним сховищем.

- Включає у себе протоколи та сервіси для забезпечення надійного та безпечного обміну даними.

6. Бібліотеки (Libraries):

- Це компоненти, які використовуються для реалізації певних функцій в додатку.
- Можуть включати у себе сторонні бібліотеки для роботи з шифруванням, авторизацією, роботою з мережею тощо.

Взаємодії*:*

1. Інтерфейс користувача ←→ Сервер для обробки запитів:

- Користувацький інтерфейс надсилає запити користувача до серверу, наприклад, запит на завантаження або синхронізацію файлів.
- Сервер обробляє ці запити та відповідає на них. Інтерфейс користувача відображає отримані відповіді.

2. Сервер для обробки запитів ←→ Хмарне сховище (Cloud Storage):

- Додаток взаємодіє з хмарним сховищем для здійснення операцій з файлами, такими як завантаження, збереження, видалення тощо.
- Він надсилає запити до хмарного сховища через мережу Інтернет та отримує відповіді з результатами операцій.

3. Сервер для обробки запитів ←→ База даних (Database):

- Додаток може взаємодіяти з базою даних для зберігання та отримання інформації про користувачів, файли, дозволи тощо.
- Він виконує запити до бази даних для отримання необхідної інформації та збереження змін.

Розроблена діаграма компонентів наведена на рисунку 2.4.

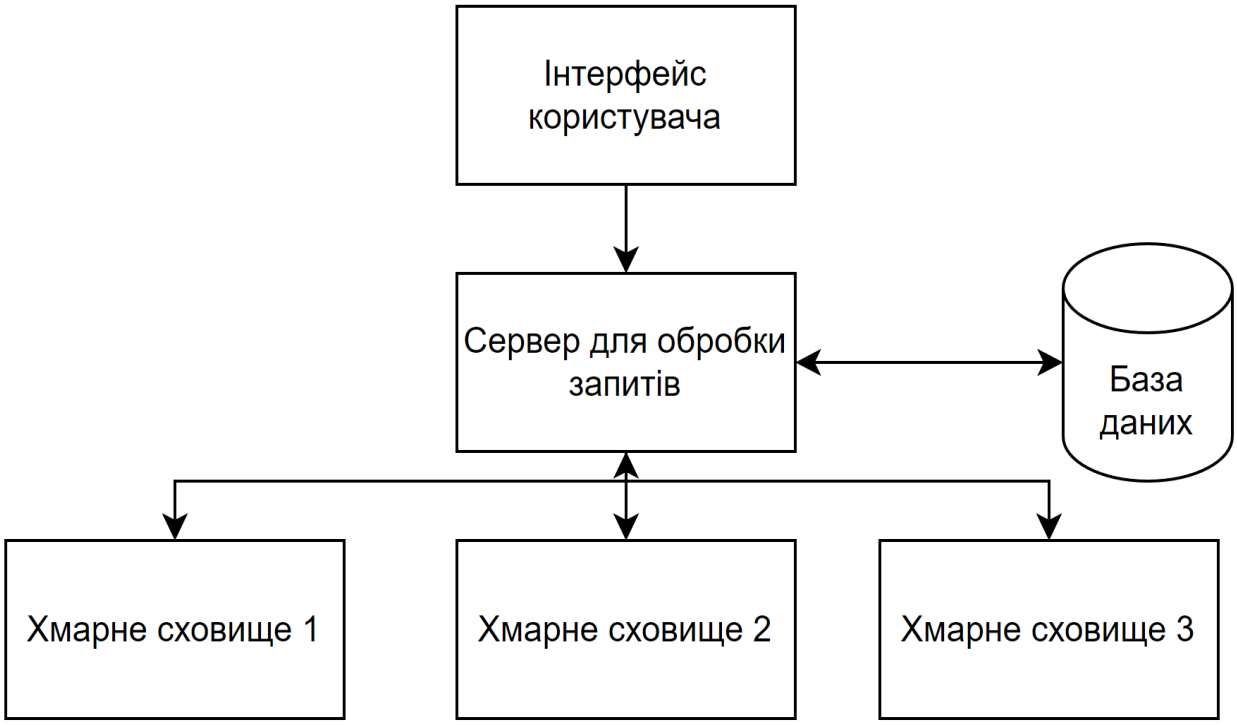

Рисунок 2.4 – Діаграма компонентів програмного забезпечення

Діаграма зв'язків між компонентами системи допомагає зрозуміти, як кожен елемент системи взаємодіє з іншими для досягнення загальної функціональності. Вона відображає поток даних та керування між компонентами, визначаючи, які компоненти виконують певні завдання та як вони обмінюються інформацією.

Ця діаграма є важливим інструментом для розробників програмного забезпечення, оскільки допомагає їм краще зрозуміти структуру системи та взаємозв'язки між її складовими частинами. Вона також допомагає командам розробників спілкуватися та узгоджувати свою роботу, розуміючи, як їхні зміни впливають на інші частини системи.

Загалом, ця діаграма є інструментом для моделювання та аналізу системи перед її реалізацією, дозволяючи розробникам уникнути помилок та оптимізувати роботу системи.

Діаграма розгортання (Deployment Diagram): Ця діаграма відображає фізичну архітектуру системи, показуючи, як компоненти програмної системи розгортані на

апаратному забезпеченні [11]. Вона вказує, як програмна система взаємодіє з апаратним середовищем.

Компоненти

1. Користувацькі пристрої (User Devices):
   - Це компонент, який представляє собою апаратні пристрої, такі як комп'ютери, смартфони, планшети, на яких запускається додаток підключення до хмарних сховищ.
   - Користувачі взаємодіють з системою через ці пристрої, використовуючи їх для завантаження, перегляду та редагування файлів у хмарних сховищах.
2. Сервери хмарних сховищ (Cloud Storage Servers):
   - Це сервери, які надають послуги зберігання та обробки даних у хмарних сховищах.
   - Вони розташовані у дата-центрах провайдерів хмарних послуг та забезпечують доступність та надійність зберігання даних.
3. База даних (Database Server):
   - Це сервер, на якому розміщується база даних, що використовується для зберігання метаданих про файли, прав доступу та іншої інформації.
   - Він може бути розміщений окремо від серверів хмарних сховищ або використовувати ту ж інфраструктуру.
4. Мережеві комунікації (Network Communication):
   - Цей компонент представляє мережеву інфраструктуру, яка забезпечує зв'язок між користувацькими пристроями та серверами хмарних сховищ.
   - Включає в себе мережеве обладнання, таке як маршрутизатори, комутатори, а також програмне забезпечення для забезпечення безпеки та надійності мережі.
5. Інтернет (Internet):

- Це компонент, що представляє собою глобальну мережу, через яку здійснюється зв'язок між користувацькими пристроями та серверами хмарних сховищ.
- Забезпечує доступ до хмарних сховищ через будь-яке мережеве з'єднання, таке як Wi-Fi, мобільна мережа, Ethernet тощо.

Звязки:

1. Зв'язок між користувацькими пристроями та серверами хмарних сховищ: користувацькі пристрої встановлюють мережеве з'єднання з серверами хмарних сховищ через Інтернет. Це може бути здійснено за допомогою різних протоколів та інтерфейсів, таких як HTTP, HTTPS, FTP, SFTP тощо.
2. Зв'язок між серверами хмарних сховищ та базою даних: **с**ервери хмарних сховищ взаємодіють з базою даних для зберігання метаданих про файли, дозволів доступу, інформації про користувачів тощо. Цей зв'язок може бути здійснений за допомогою мережевих запитів до сервера баз даних.
3. Зв'язок між користувацькими пристроями та мережевою інфраструктурою: користувацькі пристрої підключаються до мережевої інфраструктури, яка забезпечує зв'язок з серверами хмарних сховищ. Цей зв'язок може бути бездротовим або провідним, залежно від типу підключення користувача.
4. Зв'язок між серверами хмарних сховищ та мережевою інфраструктурою: сервери хмарних сховищ також підключені до мережевої інфраструктури, що забезпечує їхню доступність через Інтернет. Цей зв'язок може бути здійснений через різні мережеві пристрої та протоколи.
5. Зв'язок між серверами хмарних сховищ та інтернетом: сервери хмарних сховищ повинні мати доступ до Інтернету для забезпечення з'єднання з користувацькими пристроями. Цей зв'язок може бути здійснений через маршрутизатори та інші мережеві пристрої.

   Розроблена діаграма розгортання зображена на рисунку 2.5.

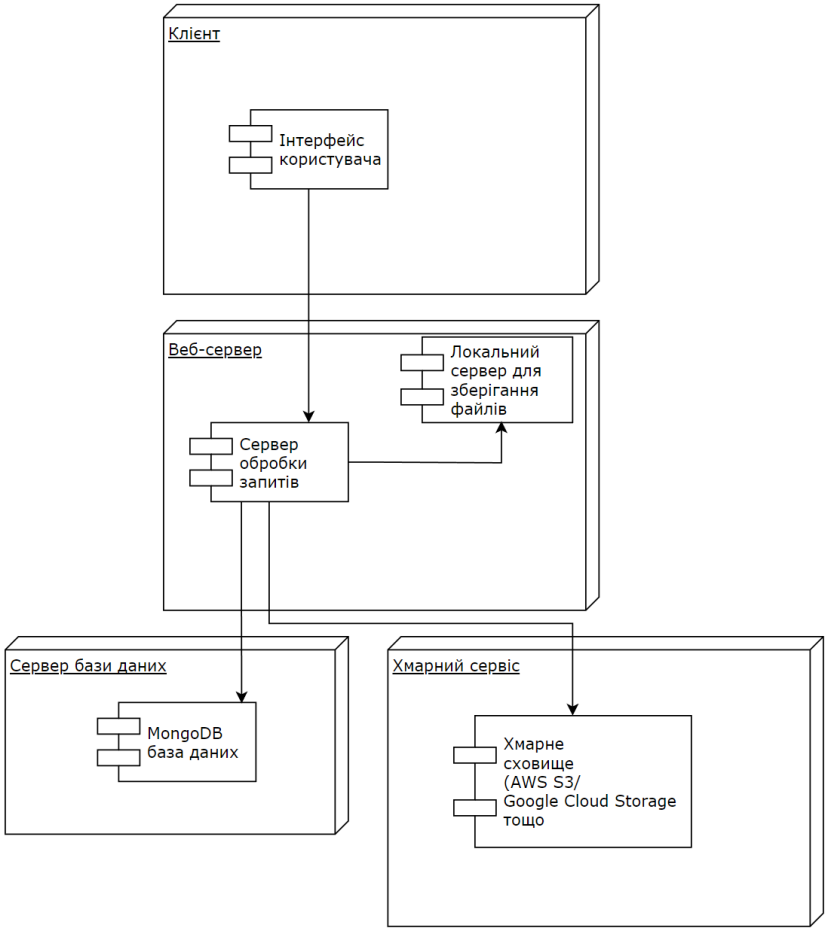

Рисунок 2.5 – Діаграма розгортання розроблюваного програмного забезпечення

Діаграма розгортання - це інструмент для візуалізації фізичної архітектури системи підключення до хмарного сховища. Вона показує, як розташовані компоненти системи та як вони взаємодіють на апаратному рівні. Розглянемо основні компоненти та їх зв'язки:

Клієнт: компонент, показаний у верхній частині діаграми. Інтерфейс користувача – веб-додаток за допомогою якого користувачі отримують доступ до роботи з хмарним сховищем.

Веб-сервер – компонент який містить в собі локальний сервер для зберігання файлів та сервер обробки запитів, з яким комунікує інтерфейс користувача, і який комунікує з всіма іншими компонентами.

Сервер бази даних – компонент який комунікує з сервером обробки запитів і містить в собі розгорнуту базу даних MongoDB.

Хмарний сервіс – в цьому компоненті розташовані будь-які підключені сторонні хмарні сховища, у випадку розробленого програмного забезпечення таким сховищем є AWS.

2.3 Розробка поведінкових UML-діаграм для створення програмного забезпечення

Діаграма варіантів використання (Use Case Diagram): Ця діаграма допоможе проілюструвати взаємодію між користувачами та системою, визначивши основні функціональні можливості додатку, такі як підключення до хмарних сховищ, завантаження та збереження файлів, керування доступом тощо [12].

Актори:

1. Користувач: це актор системи, який може лише завантажувати публічні файли, реєструватись або входити в систему.
2. Авторизований користувач: це основний актор системи, який взаємодіє з додатком для підключення до хмарних сховищ.

Прецеденти:

1. Зареєструватись: даний варіант використання потрібен для додавання користувача в систему.
2. Увійти в систему: це варіант використання, що показує, як користувач може підключитися до хмарного сховища для доступу до своїх файлів та даних. Це може включати аутентифікацію користувача, авторизацію та отримання доступу до хмарного сховища.
3. Завантаження файлів: цей варіант використання показує процес завантаження файлів користувачем до хмарного сховища. Користувач може вибрати файли зі свого пристрою та завантажити їх до хмарного сховища для зберігання та подальшого використання.
4. Збереження файлів: цей варіант використання описує процес збереження файлів користувачем у хмарному сховищі після завантаження. Користувач

може вибрати місце збереження файлів у хмарному сховищі та зберегти їх для подальшого доступу.

5. Поділитись файлом: цей варіант використання показує, як користувач може керувати доступом до своїх файлів у хмарному сховищі. Це може включати налаштування прав доступу, обмеження доступу до певних файлів або папок, а також надання спільного доступу до файлів іншим користувачам.
6. Здійснити пошук по файлам: цей варіант показує, що користувач може здійснити пошук по файлам які він раніше завантажив у хмарне сховище.
7. Видалити файл: варіант використання, що дозволяє користувачу видалити збережений раніше файл з хмарного сховища.

На рисунку 2.6 зображена розроблена діаграма варіантів використання розробленого ПЗ.

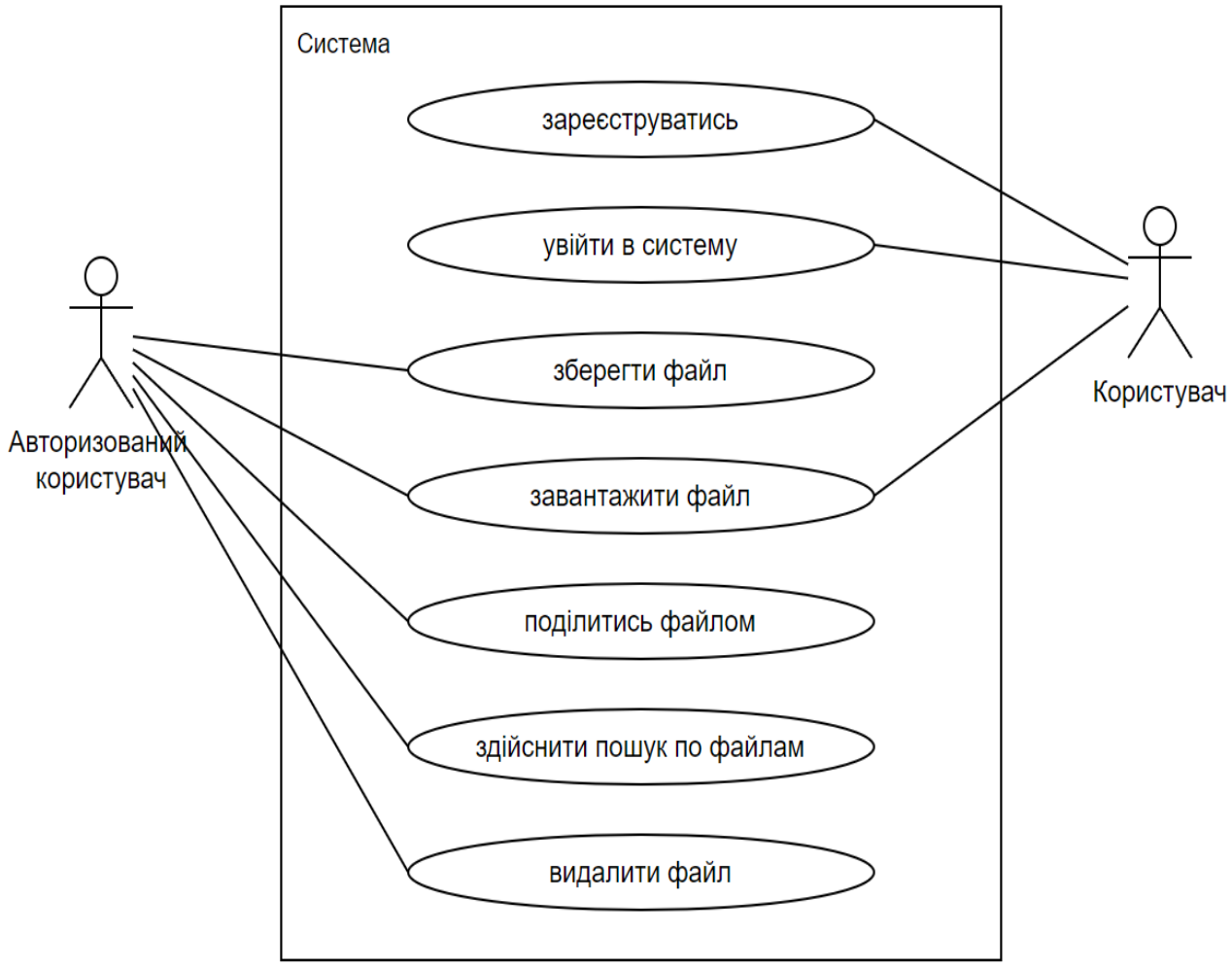

Рисунок 2.6 – Діаграма варіантів використання розробленого ПЗ

На розробленій діаграмі можна простежити як користувач авторизується через інтерфейс, та далі може виконувати функції в системі.

Діаграма станів (State Diagram): Ця діаграма показує стани, в яких може перебувати система або її окремі компоненти, та переходи між цими станами в результаті подій. Вона допомагає візуалізувати поведінку системи в різних умовах.

Діаграма станів — це інструмент в моделюванні системи, що відображає всі можливі стани, в яких може перебувати об'єкт або система, а також переходи між цими станами [13]. Давайте розглянемо компоненти та зв'язки діаграми станів для нашої системи підключення до хмарних сховищ:

1. Користувацька сесія (User Session): це початковий стан, в якому перебуває система, коли користувач увійшов у свою особисту обліковий запис. З цього стану користувач може переходити до інших станів, залежно від виконуваних дій [14].
2. Завантаження файлу (File Upload): цей стан відображається, коли користувач розпочинає завантаження файлу до хмарного сховища. З цього стану можливий перехід до стану "Завершено" або "Помилка", залежно від результату завантаження.
3. Завантаження завершено (Upload Complete): цей стан вказує на успішне завершення завантаження файлу до хмарного сховища. З цього стану користувач може перейти до інших станів, наприклад, до стану "Перегляд файлів" або "Вихід".
4. Помилка завантаження (Upload Error): якщо під час завантаження файлу виникає помилка, система переходить у цей стан. Звідси користувач може спробувати завантажити файл знову або перейти до інших опцій.
5. Перегляд файлів (View Files): у цьому стані користувач може переглядати список своїх файлів, які зберігаються в хмарному сховищі. Звідси користувач може переходити до різних станів в залежності від дій, які він хоче виконати з файлами.

6. Вихід (Logout): це кінцевий стан сесії користувача, в якому він виходить зі свого облікового запису. Звідси можливий перехід до стану "Користувацька сесія", якщо користувач вирішить увійти знову, або завершення роботи системи.

Розроблена діаграма станів наведена на рисунку 2.7.

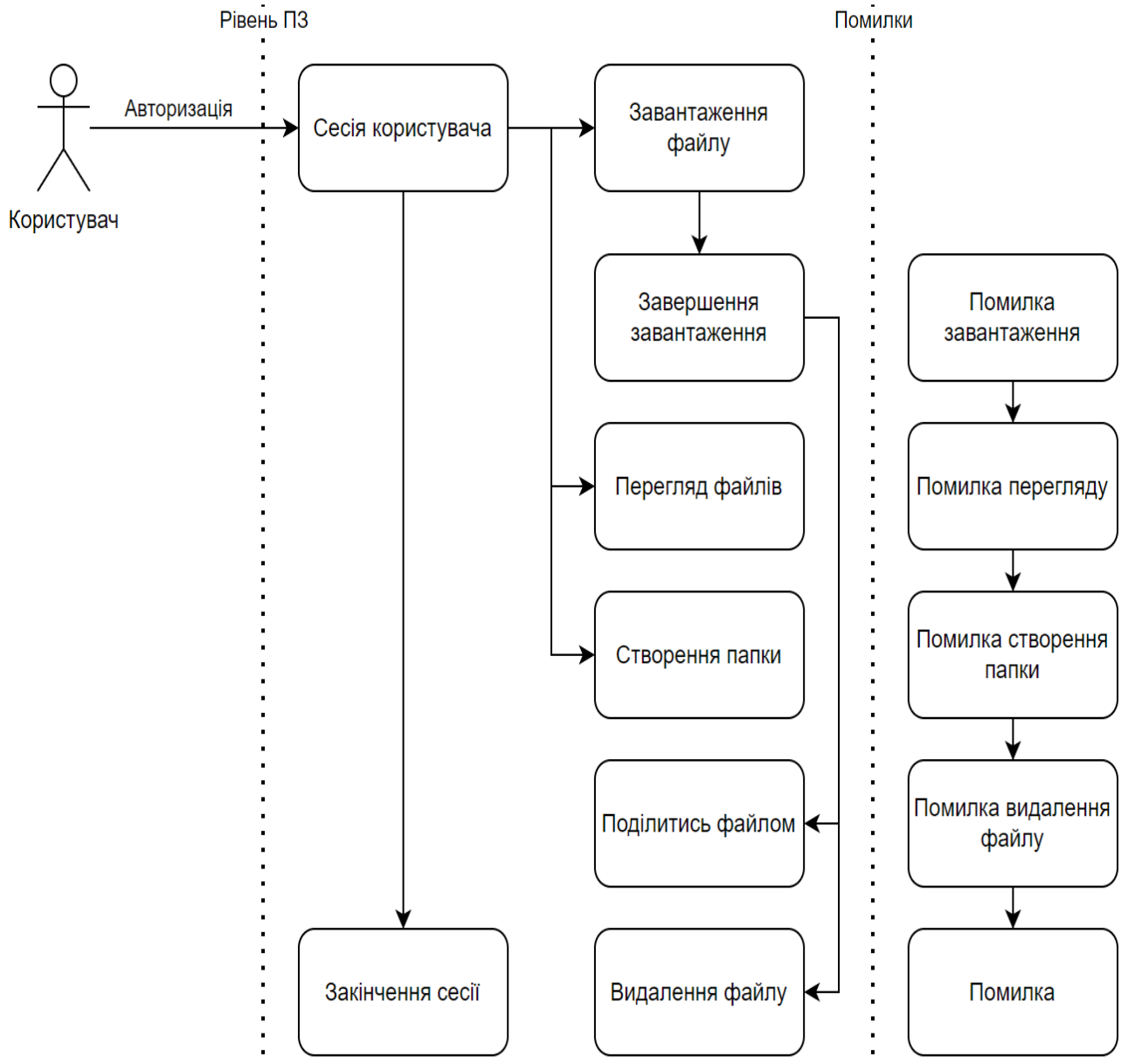

Рисунок 2.7 – Діаграма станів розроблюваного програмного забезпечення

2.4 Висновок до розділу 2

Даний розділ присвячений проектуванню програмного забезпечення для покращення роботи з хмарними сховищами.

Розроблено загальний алгоритм роботи програмного забезпечення, що перебачає наявність реалізації додаткових функцій, які полягають в можливості використання додаткових хмарних сховищ, чого немає у існуючих системах. Це дало можливість покращити роботу користувача з хмарними сховищами та підвищити надійність.

Розроблено структурну схему функціонування програмного забезпечення, що описує основні етапи роботи додатку і включає в себе додаткові функції, що дозволяють використовувати одразу декілька хмарних сховищ, чого немає у існуючих системах. Це дало можливість підвищити надійність роботи з хмарними сховищами.

Було розроблено UML-діаграми класів, компонентів, розгортання, варіантів використання, станів, взаємодій для візуального представлення функціональності та взаємодії компонентів системи. Вони деталізують основи функціонування додатку і пояснюють взаємодію його частин.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОКРАЩЕННЯ РОБОТИ З ХМАРНИМИ СХОВИЩАМИ

3.1 Обґрунтування вибору мови програмування та бібліотек для реалізації програмного засобу

Для вибору мови програмування необхідно проаналізувати існуючі на ринку. Їх є надзвичайно велика кількість, які ще й можна комбінувати між собою, однак найдоцільніше розглянути JavaScript [15], C# [16] та Python[17], оскільки вони є найпоширенішими.

Python:

1. Python має багато бібліотек для роботи з графікою та інтерфейсами, такі як PyQt, Tkinter, Kivy, PyGTK і багато інших.
2. Легко вивчити та розуміти, особливо для новачків в програмуванні.
3. Python дозволяє швидко створювати інтерфейси, особливо для прототипування та демонстрації концепцій.

JavaScript:

1. JavaScript є стандартною мовою для розробки веб–інтерфейсів, що робить його дуже потужним для розробки користувацьких інтерфейсів веб–додатків.
2. Легко інтегрується з HTML та CSS, що дозволяє створювати багатофункціональні інтерфейси.
3. Є багато бібліотек та фреймворків, таких як React, Vue і Angular, які допомагають значно полегшити розробку.

C#:

1. C# використовується для розробки додатків для операційних систем Windows, що робить його ідеальним вибором для створення інтерфейсів Windows–додатків.
2. .NET Framework надає багато готових елементів інтерфейсу користувача, які можна використовувати для швидкої розробки.

3. Можливості C# для розробки інтерфейсів можуть бути розширені за допомогою бібліотек, таких як Windows Presentation Foundation (WPF).

У кожної мови є свої переваги та недоліки. Python чудово підходить для прототипування та демонстрації концепцій. JavaScript відмінно підходить для розробки веб–інтерфейсів та має багато фреймворків та бібліотек, що полегшують розробку. C# є ідеальним вибором для розробки інтерфейсів для Windows–додатків та має готові елементи інтерфейсу користувача, що дозволяють швидко розробляти додатки.

Після проведення аналізу мов програмування користувача було обрано JavaScript через те, що він найкраще підходить для веб-розробки.

Також необхідно проаналізувати існуючі фреймворки та бібліотеки, щоб обрати ті, які буде найдоцільніше використовувати у розробці.

1. Платформи:

Node.js — це серверна платформа на базі JavaScript, яка використовує движок V8 від Google [18]. Вона забезпечує асинхронну, подієво-орієнтовану архітектуру, що робить її відмінним вибором для масштабованих додатків.

2. Фреймворки:

Express.js: Легкий і гнучкий фреймворк для Node.js, який дозволяє швидко створювати веб-додатки та API [19]. Він надає потужний набір функцій для веб- і мобільних додатків.

NestJS: Прогресивний фреймворк для створення ефективних, надійних і масштабованих серверних додатків [20]. Використовує TypeScript і дозволяє структурувати код у вигляді модулів.

3. Бібліотеки для реалізації інтерфейсу користувача:

React.js — це популярна бібліотека JavaScript для створення користувацьких інтерфейсів, розроблена компанією Facebook і вперше випущена у 2013 році [21]. Вона використовує компонентний підхід, що дозволяє розробникам створювати багаторазові компоненти з власним станом. React.js відзначається високою

продуктивністю завдяки використанню віртуального DOM і широкою екосистемою інструментів та бібліотек, які полегшують розробку складних додатків.

Vue.js — це прогресивний фреймворк JavaScript для створення користувацьких інтерфейсів, розроблений Еваном Ю і вперше випущений у 2014 році. Vue.js забезпечує простоту та гнучкість у розробці, завдяки інтуїтивно зрозумілій структурі та легкої інтеграції з існуючими проектами. Він також використовує компонентний підхід і має реактивну систему для управління даними, що робить його привабливим для розробників.

Після проведення аналізу було обрано наступні фреймворки та бібліотеки:

Node.js. Основні причини вибору:

1. Асинхронність і подієво-орієнтована архітектура: Node.js обробляє велику кількість одночасних запитів без блокування, що ідеально підходить для веб-додатків, які мають взаємодіяти з базами даних у реальному часі.
2. Велика екосистема: Node.js має велику кількість доступних бібліотек і модулів, що спрощує розробку та інтеграцію з різними сервісами.
3. Швидкість і ефективність: Завдяки використанню движка V8 від Google, Node.js забезпечує високу продуктивність і швидкість виконання.

Express.js. Основні причини вибору:

1. Express.js — це мінімалістичний веб-фреймворк для Node.js. Основні причини вибору Express.js:
2. Простота і гнучкість: Express.js надає легкий і зрозумілий інтерфейс для створення веб-серверів і маршрутизації запитів.
3. Широке використання: Express.js є одним з найпопулярніших фреймворків для Node.js, що означає велику кількість документації, прикладів та підтримки спільноти.
4. Розширюваність: Express.js дозволяє легко інтегрувати додаткові модулі та середовища, що розширюють його функціональність.

React.js. Основні причини вибору:

1. React.js має велику спільноту розробників, що забезпечує значну кількість ресурсів, навчальних матеріалів та бібліотек.
2. Використання віртуального DOM у React.js дозволяє оптимізувати оновлення інтерфейсу, що забезпечує високу продуктивність додатків.
3. React.js має багату екосистему з багатьма інструментами, такими як React Router і Redux, що полегшують розробку великих та складних додатків.
4. React.js активно підтримується Facebook та використовується багатьма великими компаніями, що додає йому стабільності та довговічності.

Також обрано TypeScript розширення можливостей мови JavaScript, що дозволяє статичну типізацію, яка дозволяє дізнаватись про помилки під часу розробки чи компіляції, у свою чергу пришвидшуючи розробку.

3.2 Розробка програмного забезпечення для роботи з хмарними сховищами

Так як передбачається, що програмне забезпечення зможе використовувати декілька хмарних сховищ, а також розширювати цей список використовуваних хмарних сховищ, доцільно використати модульний підхід.

Для визначення конкретної сутності хмарного сховища необхідно створити інтерфейс:

```
interface CloudStorage {
  uploadFile: (file: File) => FileUploadResult;
  uploadFileBatch: (files: File[]) => FileBatchUploadResult;
  downloadFile: (path: string) => string;
  downloadFileBatch: (paths: string[]) => string[];
  deleteFile: (path: string) => FileDeleteResult;
  deleteFileBatch: (paths: string[]) => FileBatchDeleteResult;
}
```

Які методи передбачає цей інтерфейс:

1. uploadFile: метод для додавання файлу у хмарне сховище. Приймає параметр типу File та повертає FileUploadResult.
2. uploadFileBatch: метод для додавання одразу певної кількості файлів у хмарне сховище. Приймає параметр масиву типу File та повертає FileBatchUploadResult.
3. downloadFile: метод для завантаження файлу з хмарного сховища. Приймає параметр типу string, що є шляхом до місцезнаходження файлу у хмарному сховищі та повертає результат типу string, що є посиланням на завантаження файлу.
4. downloadFileBatch: метод для завантаження одразу певної кількості файлів з хмарного сховища. Приймає параметр масиву типу string, що є шляхом до місцезнаходження файлів у хмарному сховищі та повертає результат масиву типу string, що є посиланнями на завантаження файлів.
5. deleteFile: метод для видалення файлу з хмарного сховища. Приймає параметр типу string, що є шляхом до місцезнаходження файлу у хмарному сховищі та повертає результат типу FileDeleteResult.
6. deleteFileBatch: метод для додавання одразу певної кількості файлів у хмарне сховище. Приймає параметр масиву типу string, що є шляхом до місцезнаходження файлів у хмарному сховищі та повертає результат типу FileBatchDeleteResult.

За допомогою цього інтерфейсу можна додавати будь-яку кількість будь-яких хмарних сховищ. І поки вони задовольняють вимоги цього інтерфейсу – їх можна буде використати у розробленому програмому забезпеченні.

Такий підхід дозволяє іншим сутностям не хвилюватись про деталі реалізації конкретного хмарного сховища, які будуть різнитись, адже у кожному класі хмарного сховища будуть всі необхідні методи для роботи з файлами. І за побажанням користувача буде використовуватись обране хмарне сховище.

Також важливо розглянути REST API серверу для обробки запитів. Express.js дозволяє створювати API для будь-яких HTTP запитів з чітко визначеними шляхами.

У розробленому програмному забезпечені мають бути щонайменше API для роботи з файлами, їх пошуком та авторизацією. Express.js дозволяє групувати API за допомогою об’єкта Router. Їх можна поєднувати між собою великою кількістю способів. Варто зауважити, що якщо на шлях запиту клієнта підпадає декілька обробників, вони відпрацьовуватимуть по порядку реєстрації. Це дозволяє гнучке комбінування обробників запитів, які ще називаються Middleware.

Реєстрація API має наступний вигляд:

```
router.post('', authMiddleware, fileController.createDir)
router.post('/upload', authMiddleware, fileController.uploadFile)
router.post('/avatar', authMiddleware, fileController.uploadAvatar)
router.get('', authMiddleware, fileController.getFiles)
router.get('/download', authMiddleware, fileController.downloadFile)
router.get('/search', authMiddleware, fileController.searchFile)
router.delete('/', authMiddleware, fileController.deleteFile)
router.delete('/avatar', authMiddleware, fileController.deleteAvatar)
```

В цьому Router є 8 API, 3 з яких для GET HTTP запитів, 3 – для POST, 2 для DELETE.

Однак необхідний ще один Router для авторизації, він має такий вигляд:

```
router.post('/registration', authController.register)
router.post('/login', authController.login)
router.get('/auth', authController.auth)
```

В цьому Router є 3 API, 1 з яких для GET HTTP запитів, 2 – для POST. Ці API покривають необхідності системи авторизації, а саме: реєстрацію, аутентифікацію та авторизацію.

3.3 Тестування розробленого програмного забезпечення

Розпочати варто з тестування реєстрації користувача, на рисунку 3.9 зображено успішну реєстрацію користувача в системі.

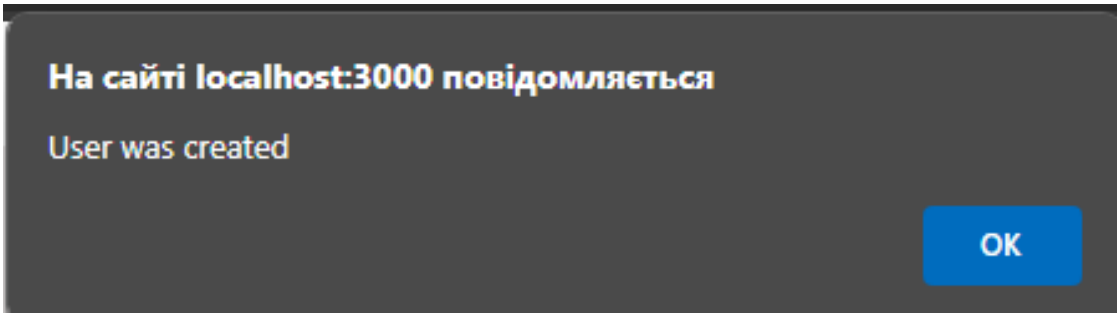

Рисунок 3.9 – Тестування реєстрації користувача

Далі розглянемо авторизацію створеного користувача, вікно авторизації зображено на рисунку 3.10.

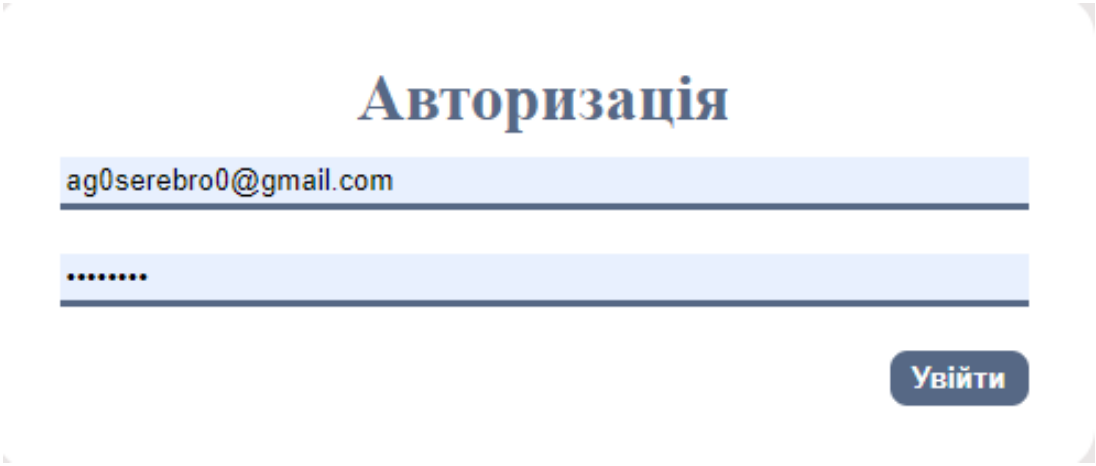

Рисунок 3.10 – Вікно авторизації користувача

Після введення правильних даних користувача і натискання кнопки «Увійти» отримуємо, перехід на головну сторінку додатку, що зображена на рисунку 3.11.

Рисунок 3.11 – Успішна авторизація

Тестуємо завантаження файлу. Для цього перетягуємо файл у вікно веб-додатку, одразу отримуємо повідомлення про завантаження. Ця дія зображена на рисунку 3.12

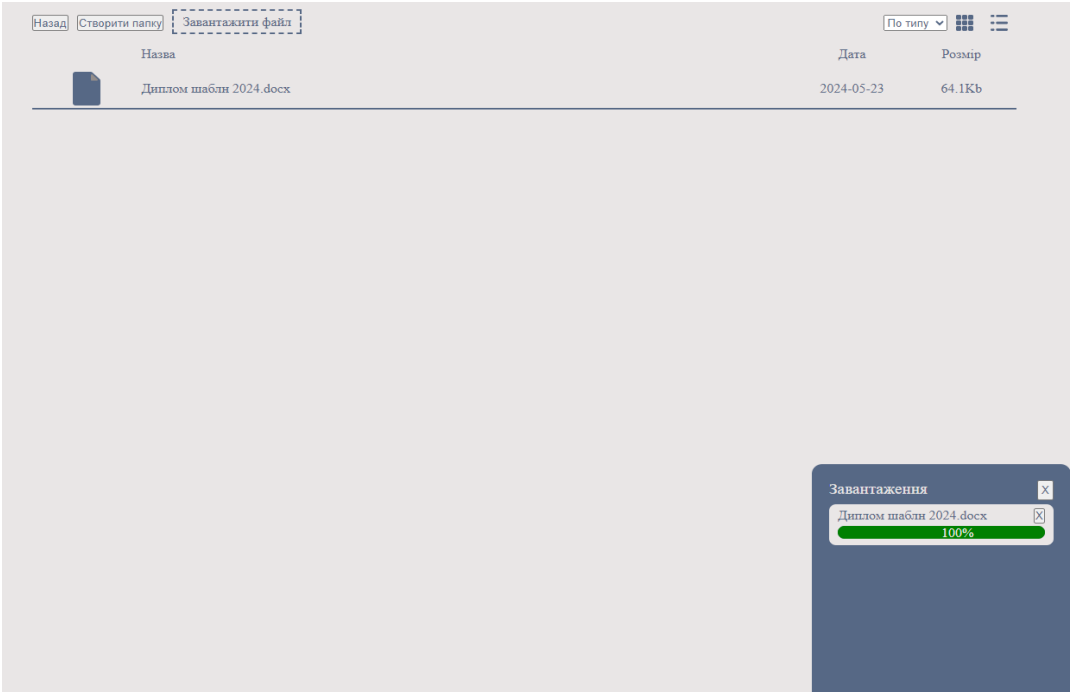

Рисунок 3.12 – Результат початку завантаження файлу

Через деякий час файл успішно завантажено, що зображено на рисунку 3.13.

Рисунок 3.13 – Результат завантаження файлу

Також необхідно протестувати створення папки для файлів, при натисканні кнопки «Створити папку» створюється нове вікно, після заповнення назви папки у якому та натиску на кнопку успішно створюється нова папка. Вікно створення папки зображено на рисунку 3.14.

Рисунок 3.14 – Вікно створення папки

Наступним кроком в тестуванні переглянемо пошук файлів по назві файлу. Для цього введемо у поле пошуку назву папки, яка була попередньо створена, і отримаємо очікуваний результат, що показано на рисунку 3.15.

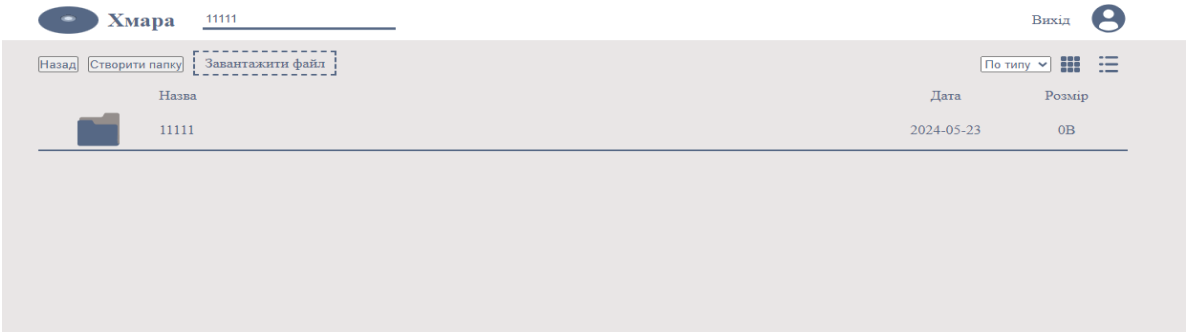

Рисунок 3.15 – Тестування пошуку по назві файлу

Необхідно також протестувати перемикач між AWS та локальним сервером для збереження фалів. При заході на сторінку очікувано автоматично обирається локальний сервер, однак при натисканні на перемикач значення очікувано змінюється на AWS, що зображено на рисунку 3.16.

Рисунок 3.16 – Тестування перемикання між локальним сервером для збереження файлів та AWS

На основі отриманих результатів тестування можна вважати успішним.

3.4 Висновок до розділу 3

Для реалізації програмного забзепечення для покращення роботи з хмарними сховищами було обрано мову програмування JavaScript та середовище Node.js. JavaScript є однією з найпопулярніших мов програмування, яка дозволяє створювати як клієнтські, так і серверні додатки. Node.js, як середовище виконання JavaScript на серверній стороні, забезпечує високу продуктивність завдяки асинхронній, подієво-орієнтованій архітектурі.

Для розробки клієнтської частини обрано React.js, який відзначається високою продуктивністю завдяки використанню віртуального DOM і широкою екосистемою інструментів та бібліотек, які полегшують розробку складних додатків.

Також для розробки програмного забзезпеченя обрано модульний підхід, що дозволяє інкапсулювати та перевикористовувати логіку, що дозволить використовувати будь-яку кількість хмарних сховищ, які задовольнятимуть вимоги інтерфейсу.

Було виконано тестування розробленого програмного забезпечення, а саме: створення користувача, авторизації, завантаження файлів, створення папки та пошуку файлів, що завершилось успішно. Під час тестування було доведено, що програмне забезпечення має підвищену надійність завдяки можливості використання додаткових хмарних сховищ.

ВИСНОВКИ

У досліджені розглянуто питання ефективної роботи з хмарними сховищами даних, які на сьогоднішній день відіграють ключову роль у збереженні та управлінні великими обсягами інформації. Проаналізовано існуючі рішення та виявлено, що незважаючи на їхні переваги, існує потреба у покращенні надійності та зручності використання.

У роботі було розглянуто поточний стан ринку програмного забезпечення для хмарних сховищ та проаналізовано основні рішення, доступні користувачам. Відзначено, що їх є досить багато, але всі вони мають типові проблеми і недоліки, зокрема складні системи ціноутворення та обмеженість функціоналу в окремих регіонах.

Виконано аналіз функціональних можливостей, переваг, недоліків, а також характеристик обраних програмних засобів. Визначено, що вони є досить схожими по функціональним можливостях, перевагах та недоліках.

Розроблено загальний алгоритм роботи програмного забезпечення, який відображає процес взаємодії основних складових розробки. Цей алгоритм відрізняється можливістю використання додаткових хмарних сховищ, що дає можливість підвищити надійність роботи з хмарними сховищами.

Розроблено UML-діаграми роботи програмного забезпечення: діаграму класів, діаграму компонентів, діаграму розгортання, діаграму варіантів використання та діаграму станів. Ці діаграми дали змогу побачити в деталях основи функціонування додатку і пояснили взаємодію його частин.

Обґрунтувано вибір мови програмування для реалізації програмного засобу. Обґрунтовано доцільність використання JavaScript, C# та Python, оскільки вони повністю задовольняють вимогам розробок веб-додатків та дозволяють створювати високоефективне програмне забезпечення. Для мови JavaScript обрано фреймворк Express.js для розробки серверу і React.js для розробки клієнтської частини.

На основі обраних вище інструментів розроблено відповідне програмне забезпечення для покращення роботи з хмарними сховищами.

Виконано успішне тестування роботи розробленого програмного забезпечення. Тестування показало що програмне забезпечення функціонує коректно, ефективно, дозволяє реалізувати додаткові функції, а саме використовувати додаткові хмарні сховища. За допомогою тестування доведено, що програмне забезпечення працює справно та дозволяє користувачам ефективно працювати з хмарними сховищами даних, забезпечуючи високу продуктивність і надійність.

Результати виконаної роботи свідчать про те, що обране середовище розробки, інструменти та підходи дозволили створити високоефективне програмне забезпечення для роботи з хмарними сховищами даних, отже, мету роботи досягнуто та поставлені задачі виконано в повному обсязі.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Пронь В. В., Арсенюк І. Р. Обґрунтування вибору інструментів для роботи з хмарними сховищами даних. Матеріали конференції «XIII International Scientific and Practical Conference International Forum: Problems and Scientific Solutions Held on June 6-8, 2024 in Melbourne, Australia», 2024 – P. 375-378. URL: https://archive.interconf.center/index.php/conference-proceeding/issue/view/6-8.06.2024/213 (дата звернення 08.06.2024)
2. Що таке хмарні сховища та як воно працює. URL: https://online.novaposhta.education/blog/shho-take-hmarni-shovishha-i-yak-ih-vikoristovuvati-use-shho-treba-znati (дата звернення: 06.05.2024)
3. Cloud Computing Services – Amazon Web Service. URL: https://aws.amazon.com/ (дата звернення: 06.05.2024)
4. Google Cloud: Cloud Computing Services. URL: https://cloud.google.com/?hl=en (дата звернення: 06.05.2024).
5. Microsoft Azure: Cloud Computing Services. URL: https://azure.microsoft.com/en-us (дата звернення 06.05.2024)
6. IDC: Worldwide IDC Global DataSphere Forecast, 2023-2027: It's a Distributed, Diverse, and Dynamic (3D) DataSphere. URL: https://www.idc.com/getdoc.jsp?containerId=US50554523 (дата звернення 08.05.2024)
7. AWS Service level agreement. URL: https://aws.amazon.com/legal/service-level-agreements/?aws-sla-cards.sort-by=item.additionalFields.serviceNameLower&aws-sla-cards.sort-order=asc&awsf.tech-category-filter=*all (дата звернення: 08.05.2024).
8. Побудова та розуміння алгоритмів. URL: https://cloud.itstep.org/blog_3/building-and-understanding-algorithms-a-step-by-step-guide-for-beginners (дата звернення: 11.05.2024)

9. Діаграми класів. URL: https://ua5.org/oop/392-diagrami-klasiv.html (дата звернення: 11.05.2024)
10. Що таке діаграма компонентів UML: символи та посібник. https://www.mindonmap.com/uk/blog/uml-component-diagram/ (дата звернення: 11.05.2024)
11. Діаграма розгортання: Підручник з UML із ПРИКЛАДОМ. URL: https://www.guru99.com/uk/deployment-diagram-uml-example.html (дата звернення 12.05.2024)
12. Варіанти використання та сценарії (Use Cases and Scenarios). URL: https://www.maxzosim.com/use-cases-and-scenarios/ (дата звернення 12.05.2024)
13. Лекція 12. Нотація UML. Моделювання поведінки. URL: https://elib.lntu.edu.ua/sites/default/files/elib_upload/%D0%9A%D0%BE%D0%BD%D0%B4%D1%96%D1%83%D1%81%202%20%D0%B3%D0%BE%D1%82%D0%BE%D0%B2%D0%B2%D0%B0/page15.html (дата звернення 14.05.2024)
14. ПОНЯТТЯ КУКИ, КЕШ, СЕСІЯ В БРАУЗЕРІ. URL: https://training.qatestlab.com/blog/technical-articles/cookies-cache-session-in-browser/ (дата звернення 14.05.2024)
15. JavaScript. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення 16.05.2024)
16. A tour of the C# language. URL: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview (дата звернення 16.05.2024)
17. Python. URL: https://www.python.org/doc/ (дата звернення 16.05.2024)
18. Node.js Introduction. URL: https://www.w3schools.com/nodejs/nodejs_intro.asp (дата звернення 18.05.2024)
19. Express web framework (Node.js/JavaScript). URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs (дата звернення 18.05.2024)
20. Nest.js. URL: https://nestjs.com/ (дата звернення 20.05.2024)

21. Що таке React JS? Як почати вивчати Реакт? Навички для react developer. URL: https://cases.media/en/article/sho-take-react-js-yak-pochati-vivchati-reakt-navichki-dlya-react-developer (дата звернення 21.05.2024)

ДОДАТКИ

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка програмного забезпечення для покращення роботи з хмарними сховищами

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 97,75% Схожість 2,25%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Пронь В.В.

Керівник роботи Арсенюк І.Р.

ДОДАТОК Б
ІНСТРУКЦІЯ КОРИСТУВАЧА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОКРАЩЕННЯ РОБОТИ З ХМАРНИМИ СХОВИЩАМИ

Для початку роботи необхідно зареєструватись. Щоб це зробити необхідно натиснути на кнопку «Реєстрація» в меню навігації і ввести свої дані у вікно реєстрації. Це вікно зображене на рисунку Б.1. Розпочати варто з тестування реєстрації користувача, на рисунку 3.9 зображено успішну реєстрацію користувача в системі.

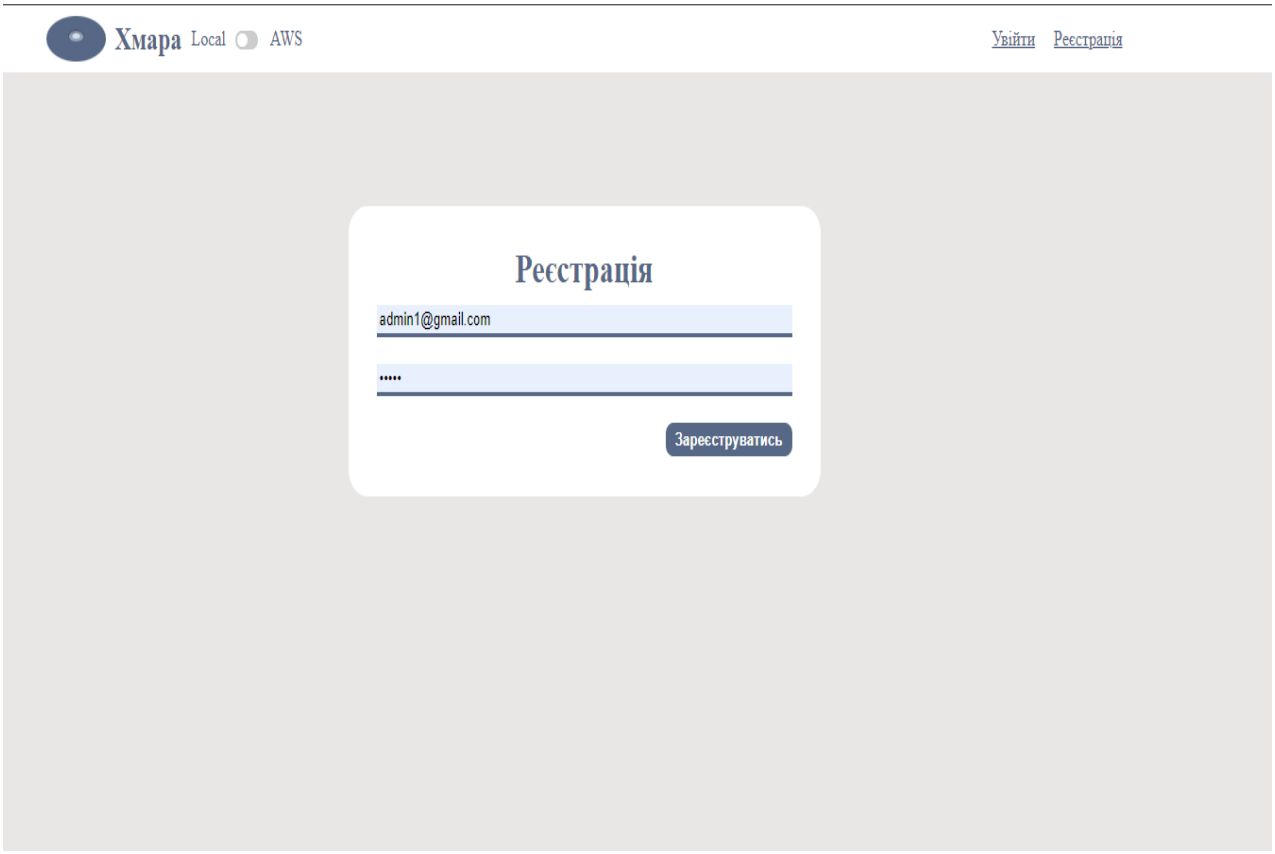

Рисунок Б.1 – Вікно реєстрації користувача

При успішній реєстрації буде відображене повідомлення, що зображене на рисунку Б.2.

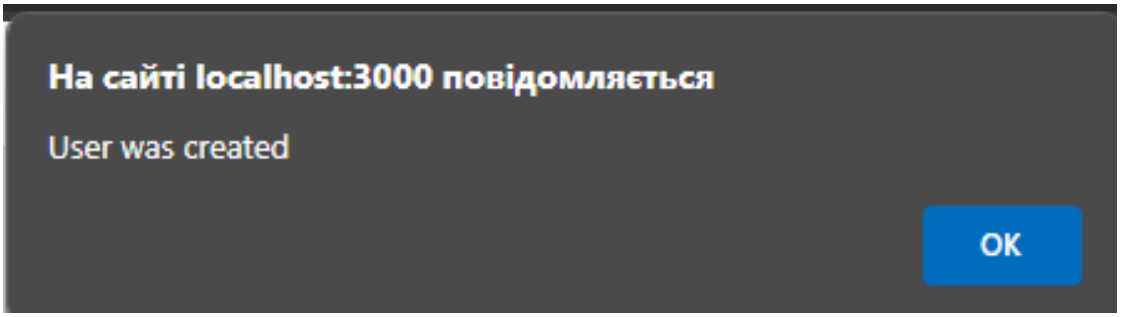

Рисунок Б.2 – Успішна реєстрація користувача

Далі необхідно перейти в меню «Увійти» і ввести свої дані. За умови успішного входу відбудеться перехід на головну сторінку програми, що зображена на рисунку Б.3.

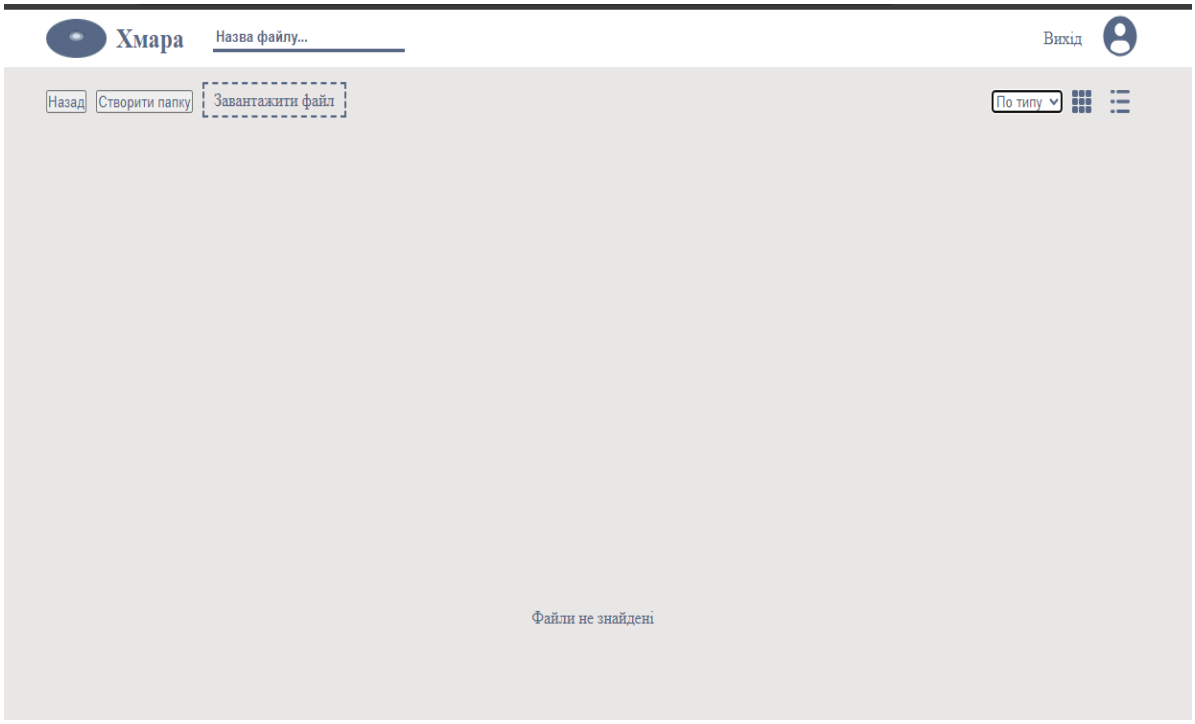

Рисунок Б.3 – Успішний вхід у програму

Для завантаження файлу необхідно його перетягнути у вікно веб-додатку, одразу отримуємо повідомлення про завантаження, як зображено на рисунку Б.4.

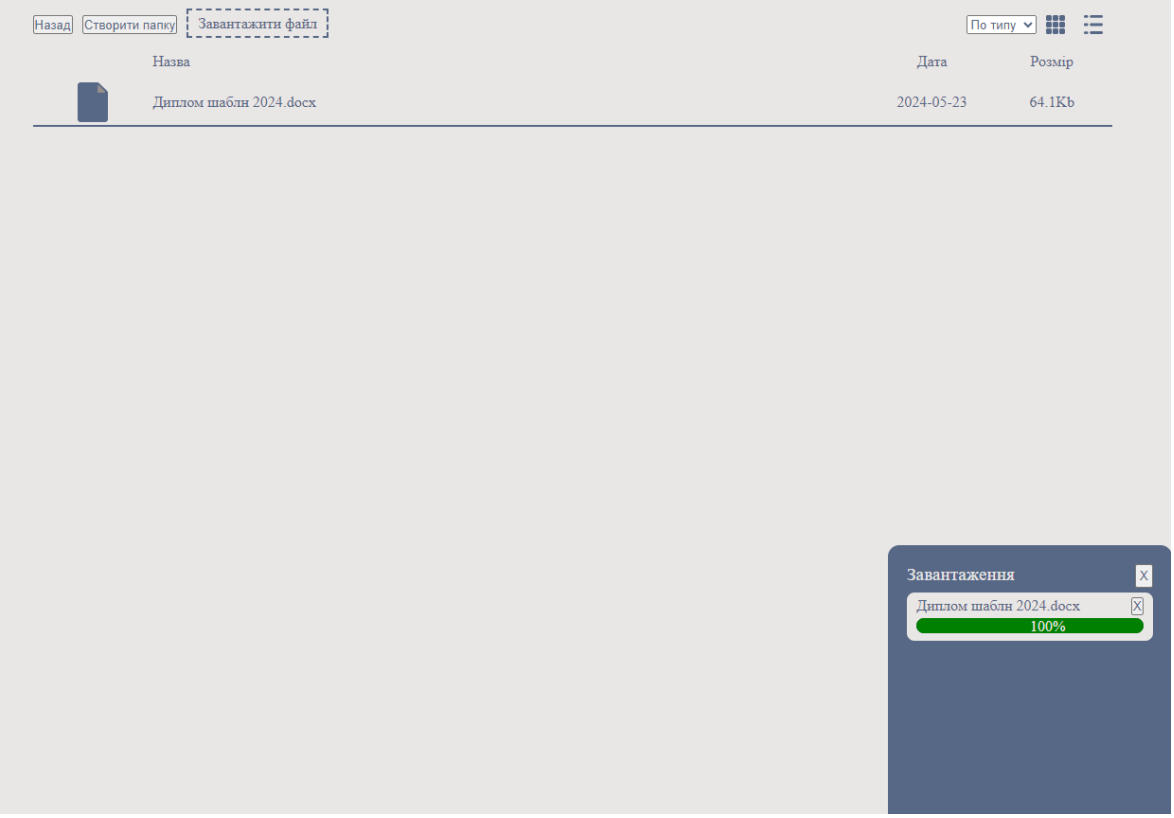

Рисунок Б.4 – Завантаження файлу

Коли файл буде успішно завантажено, він відобразиться у списку файлів, як зображено на рисунку Б.5.

Рисунок Б.5 – Результат завантаження файлу

Для створення папки необхідно натиснути кнопку «Створити папку», після чого відкриється нове вікно, яке зображене на рисунку Б.6, після заповнення назви папки у якому та натиску на кнопку створиться нова папка з вказаною назвою.

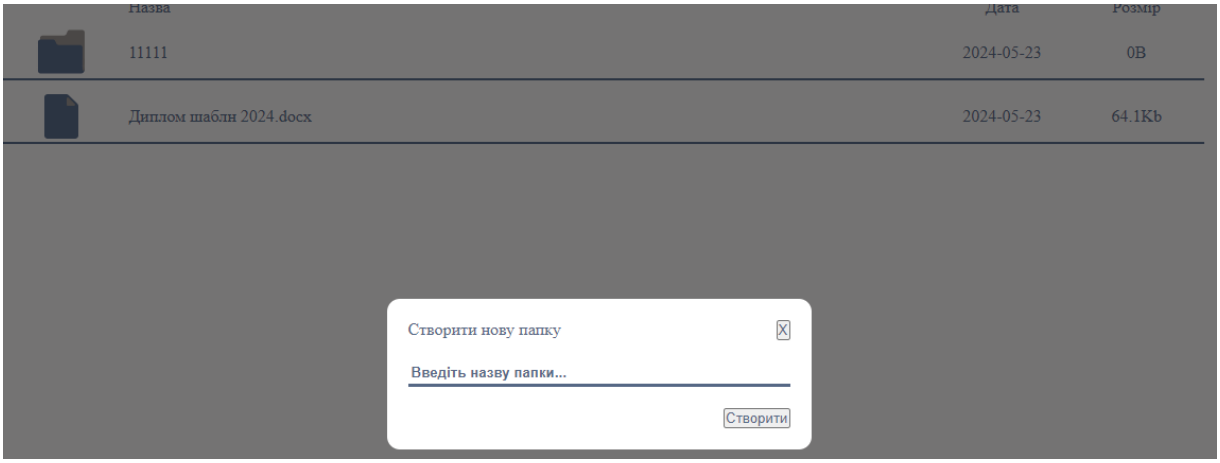

Рисунок Б.6 – Вікно створення папки

Для виконання пошуку файлів або папок за назвою необхідно ввести у поле пошуку назву папки або файлу, які була попередньо створені, після чого список файлів буде відфільтрований за назвою, як зображено на рисунку Б.7.

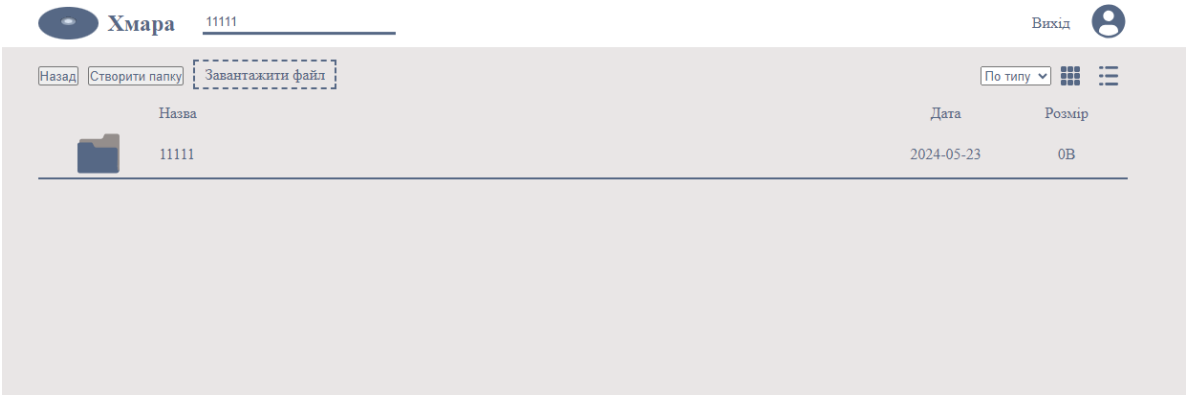

Рисунок Б.7 – Пошук по назві файлу

Для перемикання між AWS та локальним сервером для збереження фалів необхідно натиснути на перемикач у меню навігації. При заході на сторінку очікувано автоматично обирається локальний сервер, при перемиканні змінюється на AWS, як зображено на рисунку Б.8.

Рисунок Б.8 – Перемикання між локальним сервером для збереження файлів та AWS

ДОДАТОК В
ЛІСТИНГ ПРОГРАМИ

Обробка запитів для файлів

```
const UserEntity = require("../models/User");
const Uuid = require("uuid");
const FileEntity = require("../models/File");
const fileService = require("../services/fileService");
const config = require("config");
const fs = require("fs");

class FileController {
 async createDir(request, response) {
  try {
   const { name, type } = request.body;
   const { parent } = request.body;
   const fileRecord = new FileEntity({
    name,
    type,
    parent,
    user: request.user.id,
   });
   const parentFile = await FileEntity.findOne({ _id: parent });
   if (!parentFile) {
    fileRecord.path = name;
    await fileService.createDir(fileRecord);
   } else {
    fileRecord.path = `${parentFile.path}\\${fileRecord.name}`;
    await fileService.createDir(fileRecord);
    parentFile.childs.push(fileRecord._id);
    await parentFile.save();
   }
   await fileRecord.save();
   return response.json(fileRecord);
  } catch (e) {
```

```
    console.log(e);
    return response.status(400).json(e);
  }
}

async getFiles(request, response) {
  try {
    const { sort: parmeter } = request.query;
    let allFiles;
    switch (parmeter) {
      case "NAME":
        allFiles = await FileEntity.find({
          user: request.user.id,
          parent: request.query.parent,
        }).sort({ name: 1 });
        break;
      case "TYPE":
        allFiles = await FileEntity.find({
          user: request.user.id,
          parent: request.query.parent,
        }).sort({ type: 1 });
        break;
      case "DATE":
        allFiles = await FileEntity.find({
          user: request.user.id,
          parent: request.query.parent,
        }).sort({ date: 1 });
        break;
      default:
        allFiles = await FileEntity.find({
          user: request.user.id,
          parent: request.query.parent,
        });
        break;
    }
```

```
    return response.json(allFiles);
  } catch (e) {
    console.log(e);
    return response.status(500).json({ message: "Can not get files" });
  }
}

async uploadFile(request, response) {
  try {
    const requestFile = request.files.file;

    const parentFile = await FileEntity.findOne({
      user: request.user.id,
      _id: request.body.parent,
    });
    const userRecord = await UserEntity.findOne({ _id: request.user.id });

    if (userRecord.usedSpace + requestFile.size > userRecord.diskSpace) {
      return response
        .status(400)
        .json({ message: "There no space on the disk" });
    }

    userRecord.usedSpace = userRecord.usedSpace + requestFile.size;

    let pathToFile;
    if (parentFile) {
      pathToFile = `${config.get("filePath")}\\${userRecord._id}\\${
        parentFile.path
      }\\${requestFile.name}`;
    } else {
      pathToFile = `${config.get("filePath")}\\${userRecord._id}\\${
        requestFile.name
      }`;
    }
```

```
    if (fs.existsSync(pathToFile)) {
      return response.status(400).json({ message: "File already exist" });
    }
    requestFile.mv(pathToFile);

    const type = requestFile.name.split(".").pop();
    let filePath = requestFile.name;
    if (parentFile) {
      filePath = parentFile.path + "\\" + requestFile.name;
    }
    const fileInDB = new FileEntity({
      name: requestFile.name,
      type,
      size: requestFile.size,
      path: filePath,
      parent: parentFile?._id,
      user: userRecord._id,
    });

    await fileInDB.save();
    await userRecord.save();

    response.json(fileInDB);
  } catch (e) {
    console.log(e);
    return response.status(500).json({ message: "Upload error" });
  }
}

async downloadFile(request, response) {
  try {
    const file = await FileEntity.findOne({
      _id: request.query.id,
      user: request.user.id,
```

```
    });
    const pathToFile = fileService.getPath(file);
    if (fs.existsSync(pathToFile)) {
      return response.download(pathToFile, file.name);
    }
    return response.status(400).json({ message: "Download error" });
  } catch (e) {
    console.log(e);
    response.status(500).json({ message: "Download error" });
  }
}

async deleteFile(request, response) {
  try {
    const fileRecord = await FileEntity.findOne({
      _id: request.query.id,
      user: request.user.id,
    });
    if (!fileRecord) {
      return response.status(400).json({ message: "file not found" });
    }
    fileService.deleteFile(fileRecord);
    await fileRecord.remove();
    return response.json({ message: "File was deleted" });
  } catch (e) {
    console.log(e);
    return response.status(400).json({ message: "Dir is not empty" });
  }
}

async searchFile(request, response) {
  try {
    const searchName = request.query.search;
    let fileRecords = await FileEntity.find({ user: request.user.id });
    fileRecords = fileRecords.filter((file) =>
```

```
      file.name.includes(searchName)
    );
    return response.json(fileRecords);
   } catch (e) {
    console.log(e);
    return response.status(400).json({ message: "Search error" });
   }
  }
}

module.exports = new FileController();
```

Допоміжні методи для роботи з файлами

```
const fs = require("fs");
const config = require("config");

class FileService {
  createDirectory(file) {
   const pathToFile = `${config.get("filePath")}\\${file.user}\\${file.path}`;
   return new Promise((resolve, reject) => {
    try {
     if (!fs.existsSync(pathToFile)) {
      fs.mkdirSync(pathToFile);
      return resolve({ message: "success" });
     } else {
      return reject({ message: "File exists" });
     }
    } catch (e) {
     return reject({ message: "Error" });
    }
   });
  }

  deleteFile(file) {
```

```
    const pathToFile = this.getPathToFile(file);
    if (file.type === "directory") {
      fs.rmdirSync(pathToFile);
    } else {
      fs.unlinkSync(pathToFile);
    }
  }

  getPathToFile(relativeFilePath) {
    return config.get("filePath") + "\\" + relativeFilePath.user + "\\" + relativeFilePath.path;
  }
}

module.exports = new FileService();
```

API шляхи для роботи з файлами

```
const Router = require("express");
const files = require("../controllers/fileController");
const auth = require("../middleware/auth.middleware");

const router = new Router();
router.post("/upload", auth, files.uploadFile);
router.get("/download", auth, files.downloadFile);
router.get("/search", auth, files.searchFile);
router.get("", auth, files.getFiles);
router.post("", auth, files.createDirectory);
router.delete("/", auth, files.deleteFile);

module.exports = router;
```

ДОДАТОК Г

ГРАФІЧНА ЧАСТИНА

«РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОКРАЩЕННЯ РОБОТИ З ХМАРНИМИ СХОВИЩАМИ»

Виконав: студент 4 курсу, групи 2КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)

Пронь В. В.
(прізвище та ініціали)

Керівник: к. т. н., доцент каф. КН
Арсенюк І. Р.
(прізвище та ініціали)

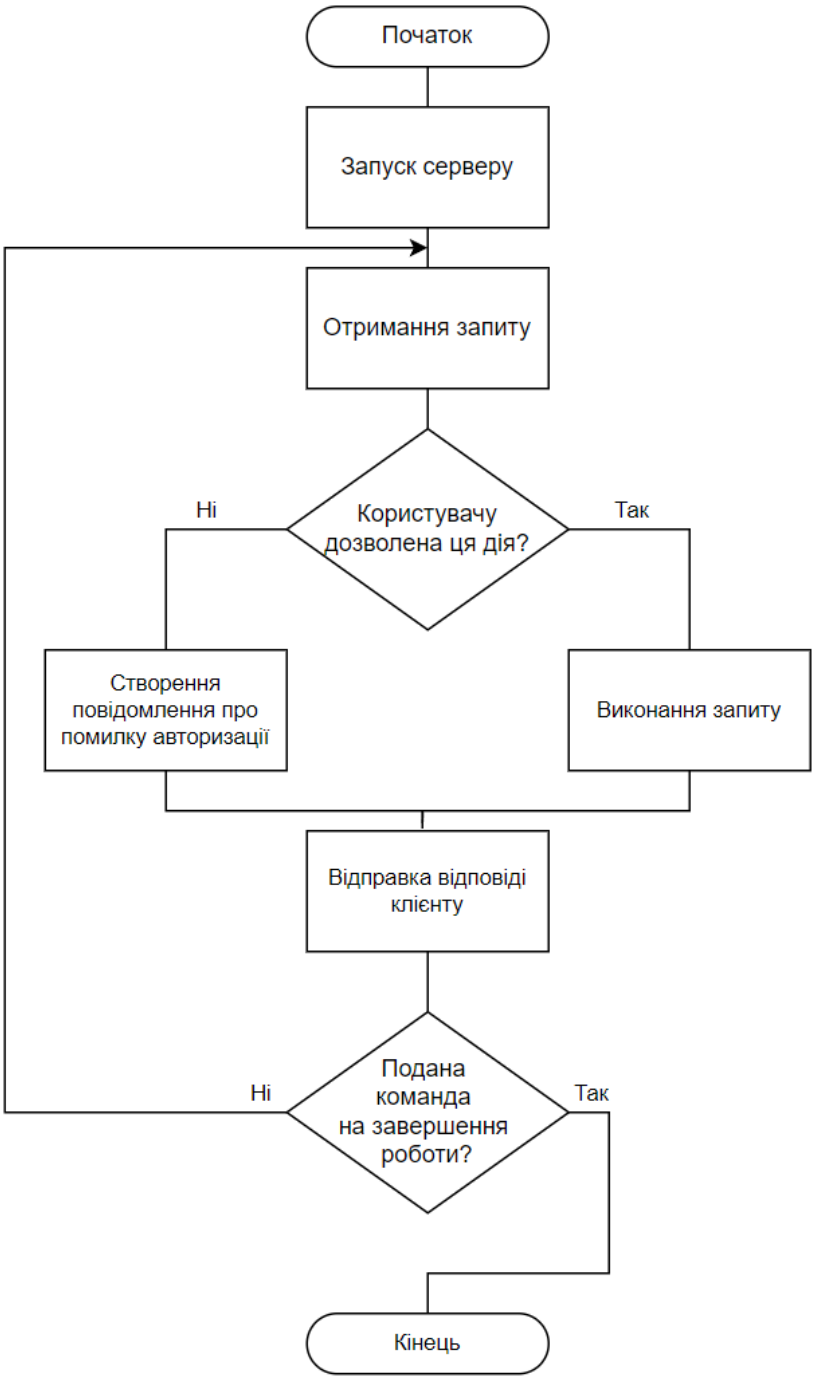

Рисунок Г.1 – Загальний алгоритм функціонування програмного забезпечення для покращення роботи з хмарними сховищами

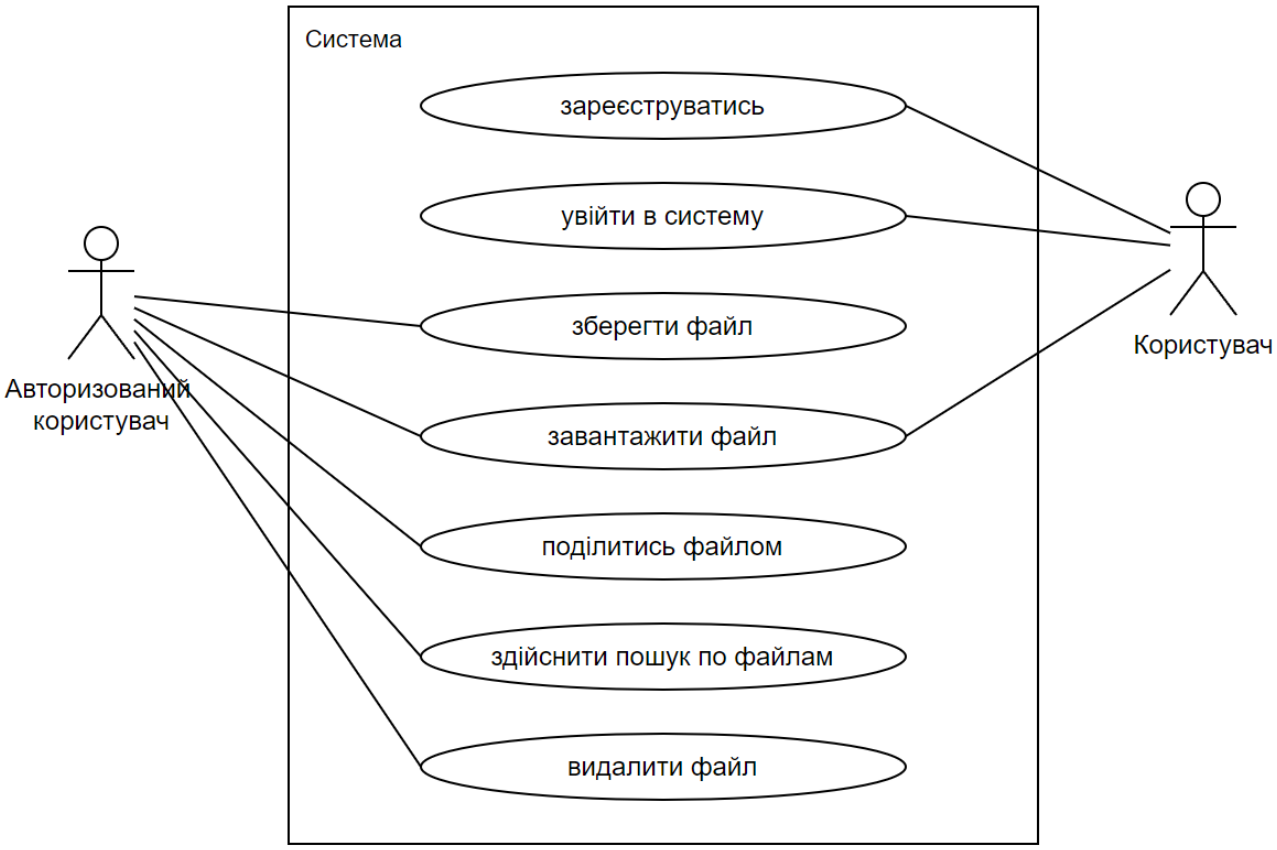

Рисунок Г.3 – Діаграма варіантів використання розробленого програмного забезпечення

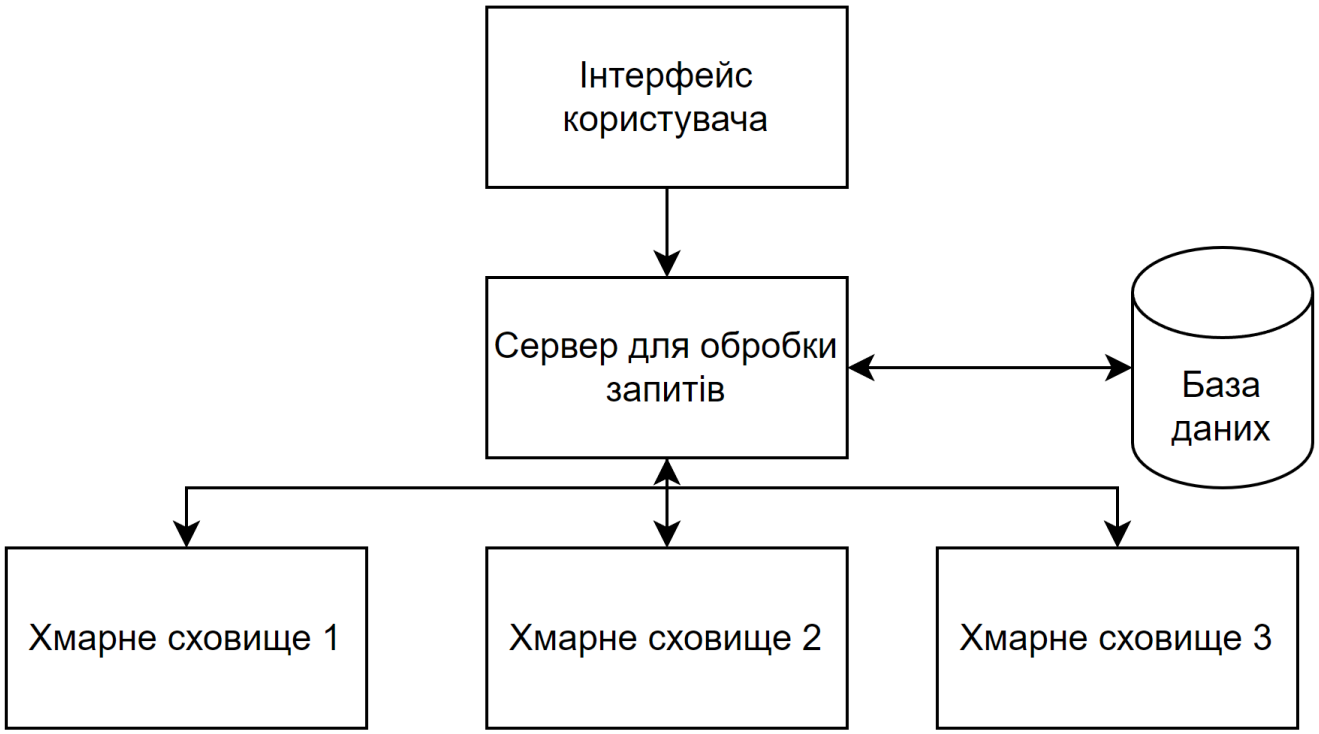

Рисунок Г.3 – Діаграма компонентів розробленого програмного забезпечення

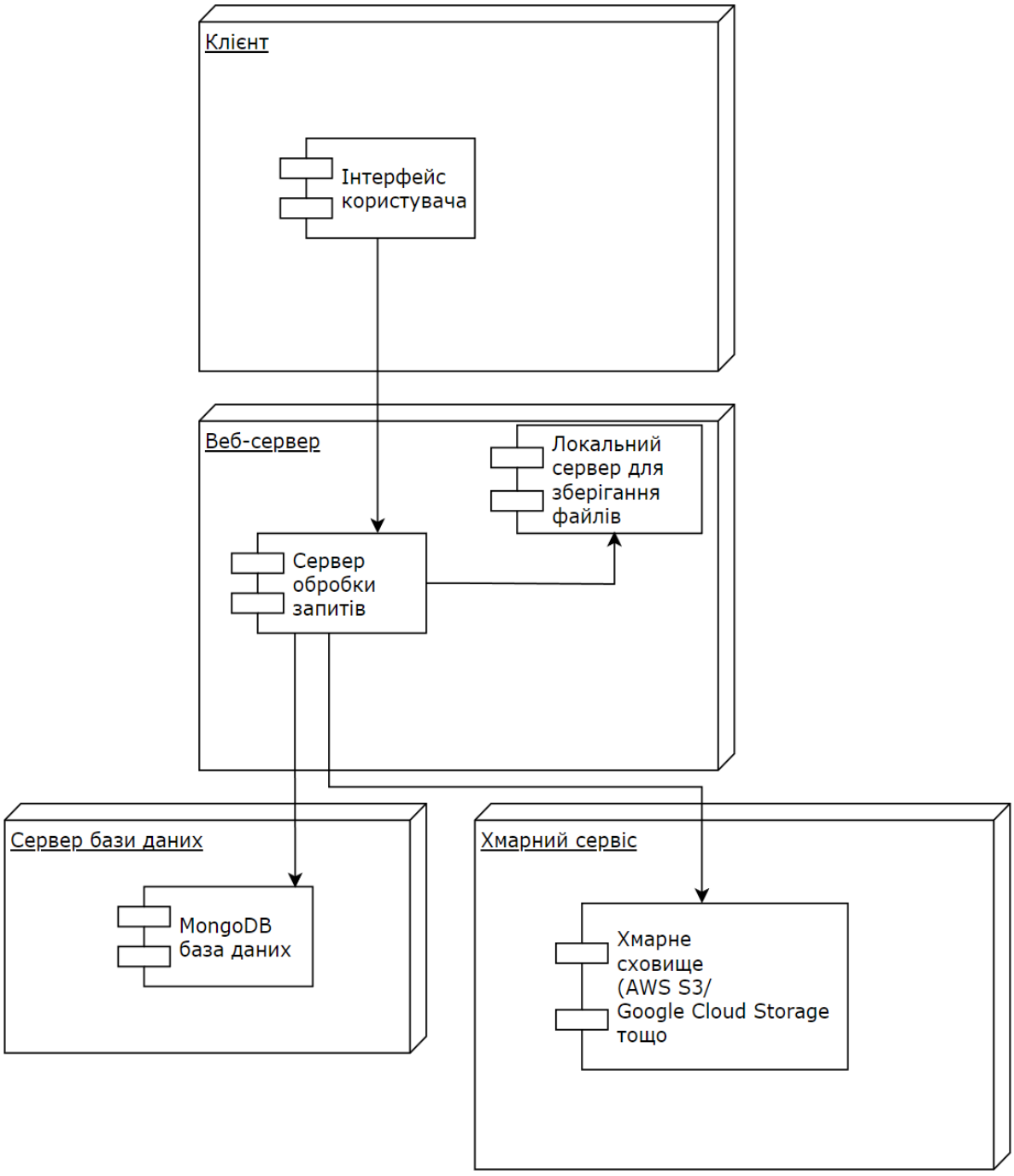

Рисунок Г.4 – Діаграма розгортання розробленого програмного забезпечення

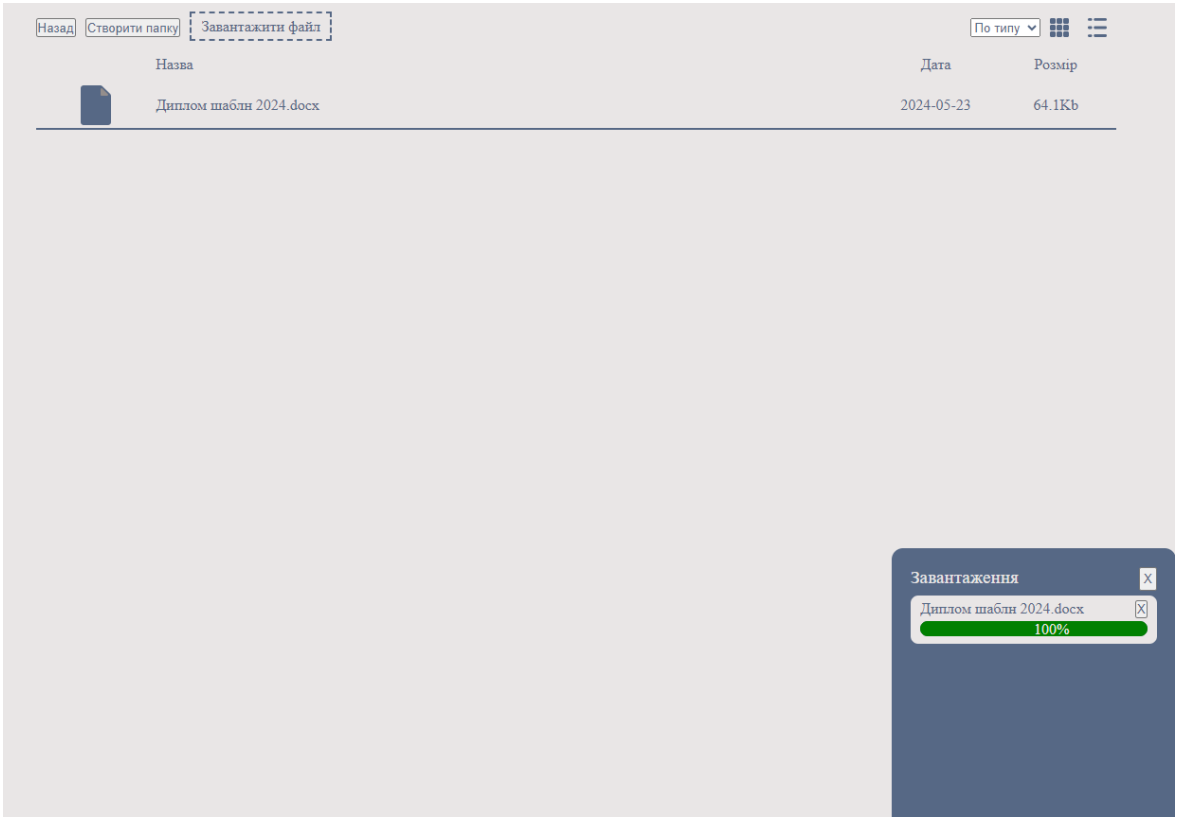

Рисунок Г.5 – Приклад роботи розробленого програмного забезпечення