Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Методи і програмні засоби для розподілення даних між хмарними сховищами»

Виконав: студент 2-го курсу групи 2ПІ-20м
спеціальності 121 «Інженерія програмного забезпечення»
Бондарчук В. К.
(прізвище та ініціали)
Керівник: д.т.н., професор каф. ПЗ
Ліщинська Л.Б.
(прізвище та ініціали)
«___» _____ 2021 р.
Опонент: к.т.н., доцент кафедри КН
Арсенюк І.Р.
(прізвище та ініціали)
«___» _____ 2021 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н. проф. Романюк О.Н.
(призвіще та ініціали)
« » _____2021 р.

Вінниця ВНТУ - 2021 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
«13» вересня 2021 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Бондарчуку Вячеславу Костянтиновичу

1. Тема роботи – Методи і програмні засоби для розподілення даних між хмарними сховищами.

Керівник роботи: Ліщинська Людмила Броніславівна, д.т.н., професор каф. ПЗ, затверджені наказом вищого навчального закладу від «24» вересня 2021 р. № 277.

2. Строк подання студентом роботи 1 грудня 2021 р.

3. Вихідні дані до роботи: методи розподілу даних між хмарними сховищами такими як Google Drive і One Drive; моделі розподілу даних між хмарними сховищами; вихідні дані для розподілу – типи файлів які є критичними; дані про кількість та тип сховищ користувача; вихідні дані – додаток, що розподіляє дані між декількома хмарними сховищами користувача.

4. Зміст розрахунково-пояснювальної записки: аналіз актуальності даного додатка; аналіз методів розподілу даних; аналіз моделей розподілу даних; класифікація алгоритмів розподілу даних; використання хмарних сховищ для зберігання даних користувачів; розробка структури додатку; розробка інтерфейсу додатка; розробка загального алгоритму роботи додатка; економічна частина.

5. Перелік ілюстративного матеріалу: галузі застосування методів та засобів розподілу даних; алгоритм роботи додатку; структурна схема; інтерфейс додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Ліщинська Л.Б., д.т.н., проф. каф. ПЗ	14.09.2021	01.12.2021
5	Бурєннікова Н. В., д.е.н., проф. кафедри ЕПВМ	14.11.2021	01.12.2021

7. Дата видачі завдання ___14 вересня 2021 р._____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної галузі і засобів розподілу даних	15.09.2021 – 31.10.2021	Вик.
2	Методи і моделі розподілу даних	31.10.2021 – 07.11.2021	Вик.
3	Розробка програмних засобів розподілу даних між хмарними сховищами	07.11.2021 – 10.11.2021	Вик.
4	Тестування програмного продукту	10.11.2021 – 14.11.2021	Вик.
5.	Економічна частина	14.11.2021 – 21.11.2021	Вик.

Студент _____ **Бондарчук В. К.**

(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Ліщинська Л. Б.**

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 621.374.415

Бондарчук В. К. Методи і програмні засоби для розподілення даних між хмарними сховищами. Магістерська кваліфікаційна робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – програмна інженерія. Вінниця: ВНТУ, 2021 99 с.

На укр. мові. Бібліогр.: 22 назв; рис.: 19; табл. 17.

Результатом виконання магістерської кваліфікаційної роботи є веб-додаток для розподілу даних між хмарними сховищами. Вдосконалено метод розподілу даних. Було описано класи, структури, графічний інтерфейс веб-додатку та саму його архітектуру.

При створенні веб-додатка та його серверної частини використано мову програмування Java та Scala та фреймворк Spring Boot для серверної частини, для клієнтської частини було використано скриптову мову Java Script та Bootstrap. Розроблений додаток відповідає вимогам, має сучасний зручний та зрозумілий інтерфейс.

Розробка продукту була виконана у інтегрованому середовищі Intellij IDEA.

Ключові слова: хмарне сховище, Mirror Mode, розподіл даних, One Drive, Google Drive.

ABSTRACT

UDC 621.374.415

Bondarchuk VK Methods and software for data distribution between cloud storage. Master's thesis in specialty 121 - Software Engineering, educational program - software engineering. Vinnytsia: VNTU, 2021 99 p.

In Ukrainian. Bibliogr .: 22 titles; fig .: 19; table 17.

The result of the master's qualification work is a web application for the distribution of data between cloud storage. The method of data distribution has been improved. The classes, structures, graphical interface of the web application and its architecture were described.

The Java and Scala programming languages and the Spring Boot framework for the server part were used to create the web application and its server part, and the Java Script and Bootstrap scripting languages were used for the client part. The developed application meets the requirements, has a modern user-friendly and understandable interface

Product development is performed in the integrated environment Intellij IDEA.

Key words: cloud storage, Mirror Mode, data distribution, One Drive, Google Drive.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному світі дані займають основну нішу і є невід'ємною частиною у теперішньому житті людини. У даних зберігається все – від фото до важливих документів, які б дуже не хотілось втратити. Також дані є критичною секцією у будь-якій IT компанії.

Ми живемо у добу розвитку, де панує хаотичний потік даних, який безперервно протікає між нами, а отже ці дані потрібно десь зберігати, адже втрата їх може призвести до великих втрат.

Актуальність розробки полягає у тому, що вона дозволяє розподіляти дані між декількома хмарними сховищами зі зниженням ймовірності їх втрати а також збільшити відсоток доступності у момент недоступного одного з хмарних сервісів.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалась згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою магістерської кваліфікаційної роботи є підвищення ефективності розподілу даних зі зниженням ймовірності їх втрати.

Основними задачами дослідження є:

- аналіз існуючих методів та засобів розподілу даних;
- удосконалення методу розподілу даних;
- аналіз архітектурного підходу;
- розробка алгоритмів функціонування модулів;
- розробити модулі та сервіси для розподілу даних;
- провести тестування програмного продукту.

Об'єктом дослідження є процес розподілення даних.

Предметом дослідження є методи і засоби розподілу даних між хмарними сховищами.

Методи дослідження. У магістерській кваліфікаційній роботі використано метод проектування програмного сервісу для розподілу даних;

методи теорії алгоритмів для вдосконалення існуючого алгоритму розподілу даних.

Наукова новизна отриманих результатів. Подальшого розвитку отримав метод Mirror Mode розподілу даних між сховищами, який відрізняється від відомого спрощенням обчислювального процесу та можливістю адаптації до прикладних задач ї та поширення на хмарні середовища, що дозволяє підвищити ефективність розподілу даних зі зниженням ймовірності їх втрати.

Практична цінність отриманих результатів полягає у тому, що на основі отриманих у магістерській кваліфікаційній роботі теоретичних положень запропоновано програмні засоби для зменшення ймовірності втрати критичних даних користувача. Програмний продукт реалізований у вигляді web-орієнтованої платформи, що забезпечує мобільність при використанні.

Особистий внесок здобувача. Усі наукові результати отримано автором самостійно. У праці, опублікованій у співтоваристві, здобувачу належить запропонований аналіз методів і засобів розподілу даних [3,4].

Апробація результатів. Основні положення й результати досліджень обговорювались на:

1. XIV Міжнародній науково-практичній конференції «Інформаційні технології і автоматизація - 2021», 15 жовтня 2021 р.
2. Міжнародній науково-практичній Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ», 9-10 листопада 2021 р.

Публікації. Результати досліджень опубліковано у двох збірниках матеріалів тез [3,4].

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ЗАСОБІВ РОЗПОДІЛУ ДАНИХ

1.1 Аналіз предметної галузі та особливості розподілу даних

На сьогодні, дані є основою цифрового світу. Дані підтримують усе, від відеороликів, до мільярдів фінансових операцій, які відбуваються щодня. В основі всього цього лежить розподілене зберігання.

На практиці існує багато видів розподілених сховищ даних. Зазвичай вони поставляються як послуги, якими керують хмарні провайдери, або продукти, які розгортаються самостійно. Також є можливість створити власне сховище.

Щоб по-справжньому зрозуміти, спочатку потрібно усвідомити масштаби даних на сьогодні. Розглянемо деякі конкретні цифри:

– Steam мав пік у 18,5 мільйонів одночасних користувачів, розгорнув сервери з 2,7 петабайтами твердотільного накопичувача та поставив 15 екзабайт користувачам у 2018 році;

– Nasdaq у 2020 році за один день зібрав максимум 113 мільярдів записів, збільшившись у середньому з 30 мільярдів лише двома роками раніше [1].

Кожен фрагмент даних ретельно зберігається і десь обробляється.

Одномашинні сховища даних просто не можуть задовольнити такі вимоги. Тому, замість цього використовуються розподілені сховища даних, які пропонують ключові переваги в продуктивності, масштабованості та надійності.

Продуктивність є критичною. Існує незліченна кількість досліджень, які кількісно визначають та показують вплив затримок на бізнес на 100 мс.

Повільний час реагування не просто засмучує людей - це коштує трафіку, продажів і, зрештою, доходу [1].

У випадку одномобільних сховищ даних часто досить просто перейти на більш швидку машину. Якщо цього недостатньо, то в справу вступають інші форми масштабування.

Горизонтальне масштабування означає додавання або видалення комп'ютерів (також відомих як машини або вузли).

Вертикальне масштабування означає зміну процесора машини, оперативної пам’яті, ємності пам’яті чи іншого обладнання [1].

Горизонтальне масштабування - це те, чому розподілені сховища даних можуть випереджати сховища даних для однієї машини. Поширюючи роботу на сотні комп’ютерів, сукупна система має більш високу продуктивність та надійність. Хоча розподілені сховища даних покладаються переважно на горизонтальне масштабування, вертикальне масштабування використовується разом для оптимізації загальної продуктивності та вартості.

Масштабування існує в широкому спектрі від ручного до повністю керованого [1].

Нарешті, деякі служби обробляють усі масштабування, навіть не думаючи про це розробнику, наприклад, S3 Amazon Web Service.

Незалежно від підходу, усі послуги мають деякі обмеження, які неможливо збільшити, наприклад максимальний розмір об’єкта [1].

Деякі програми настільки важливі для нашого життя, що навіть секунди збою є неприйнятними. Ці програми не можуть використовувати одномашинні сховища даних через неминучі збої обладнання та мережі, які можуть поставити під загрозу всю службу. Натомість використовуються розподілені сховища даних, тому що вони можуть пристосуватися для окремих комп’ютерів або помилкових мережевих шляхів.

Щоб мати високу надійність, система повинна бути доступною та надійною.

Доступність - це відсоток часу, протягом якого служба доступна і нормально відповідає на запити [1].

Відмовостійкість - це здатність переносити апаратні та програмні збої. Повна відмовостійкість неможлива.

Хоча спочатку доступність та відмовостійкість можуть виглядати подібними, але насправді вони дуже різні.

Доступний, але не витримує помилок: система, яка виходить з ладу щохвилини, але відновлюється протягом мілісекунд. Користувачі можуть отримати доступ до послуги, але довготривалі завдання ніколи не мають достатньо часу на завершення [1].

Відмовостійкий, але недоступний: система, де половина вузлів постійно перезавантажуються, а інші стабільні. Якщо ємність стабільних вузлів недостатня, то деякі запити доведеться відхилити.

Для розробника додатків ключовим моментом є те, що розподілені сховища даних можуть масштабувати продуктивність та надійність далеко за межі окремих машин. Загадка в тому, що вони мають застереження щодо того, як вони працюють, що може обмежити їх потенціал [1].

Розбиття на розділи. Набори даних часто занадто великі, щоб їх можна було зберігати на одній машині. Щоб подолати це, потрібно розділили дані на менші підмножини, які окремі машини можуть зберігати та обробляти.

Існує багато способів розподілу даних, кожен з яких має свої компроміси. Два основні підходи - вертикальне та горизонтальне розділення.

Вертикальний розподіл - означає розділення даних за відповідними полями. Вони можуть бути властивостями якогось загального об’єкта. Точний спосіб вертикального розподілу даних між машинами в кінцевому підсумку залежить від властивостей сховища даних та шаблонів використання [1].

Горизонтальний розподіл(також відомий як шардинг) - це коли розподіляються дані на підмножини, усі з однією схемою. Наприклад, горизонтальне розділення таблиці реляційних баз даних, згрупувавши рядки у фрагменти, які будуть зберігатися на окремих машинах. Стратегії розподілу поділяються на дві категорії, алгоритмічну та динамічну, але існують гібриди [1].

Алгоритмічний розподіл визначає, якому фрагменту розподілити дані на основі функції ключа даних. Наприклад, зберігаючи URL-адреси зіставлення даних "ключ-значення" до HTML, ми можемо розподіляти дані за допомогою розподілу ключів-значень відповідно до першої літери URL-адреси.

Наприклад, усі URL -адреси, що починаються на “A”, надходитимуть на першому комп’ютері, “B” на другому, тощо.

Динамічне шардування явно вибирає розташування даних і зберігає це місце в таблиці пошуку. Щоб отримати доступ до даних, потрібно звернутись до служби за допомогою таблиці пошуку або перевірити локальний кеш.

Таблиці пошуку можуть бути досить великими, і, отже, вони можуть мати таблиці пошуку, що вказують на таблиці додаткового пошуку, наприклад, B+-дерево. Динамічне фрагментування є більш гнучким, ніж алгоритмічне шардінг [1].

На практиці розподіл розділів є досить складним і може створити багато проблем, про які потрібно знати:

– частинки можуть мати нерівномірний розмір даних. Це поширене явище в алгоритмічному шардінгу, де важко отримати правильну функцію. Ми пом'якшуємо це, адаптуючи стратегію розподілу даних;

– осколки можуть мати гарячі точки, де певні дані запитуються за величиною частіше, ніж інші. Перерозподіл даних для обробки додавання або видалення вузлів із системи утруднений при збереженні високої бно розділити. Індекси можуть індексувати фрагмент, на якому він зберігається (локальний індекс), або він може індексувати весь набір даних і бути розділений (глобальний індекс). Кожен з них має компроміси;

– транзакції між розділами можуть працювати, або вони можуть бути відключені, повільні або непослідовні. Це особливо важко під час створення власного розподіленого сховища даних з одномобільних сховищ даних.

Розбиття даних - це лише частина. Маршрутизація запитів може відбуватися на різних рівнях стека програмного забезпечення.

– розбиття на стороні клієнта - це коли клієнт тримає логіку прийняття рішення, до якого вузла запитувати. Перевагою є концептуальна простота, а недоліком - кожен клієнт повинен реалізувати логіку маршрутизації запитів;

– розбиття на основі проксі-це коли клієнт надсилає всі запити проксі. Потім цей проксі-сервер визначає, до якого вузла серверної частини слід

звертатись. Це може допомогти зменшити кількість одночасних з'єднань на ваших серверних серверах та відокремити логіку програми від логіки маршрутизації;

– розбиття на основі сервера - це коли клієнт підключається до будь-якого серверного вузла, і вузол або обробляє, або перенаправляє, або пересилає запит;

Остання концепція - це реплікація. Реплікація означає зберігання кількох копій одних і тих же даних;

– надмірність даних - коли апаратне забезпечення неминуче виходить з ладу, дані не втрачаються, оскільки є інша копія;

– доступність даних - клієнти можуть отримати доступ до даних з будь -якої копії. Це підвищує стійкість до перебоїв у роботі центру обробки даних та мережевих розділів;

– підвищена пропускна здатність читання - є більше машин, які можуть обслуговувати дані, і тому загальна ємність вища;

– зменшення затримки мережі - клієнти можуть отримати доступ до найближчої до них репліки, зменшивши затримку мережі.

Реалізація реплікації вимагає неперевершених протоколів консенсусу та вичерпного аналізу сценаріїв збоїв [1].

Реплікуючи дані близько один до одного, мінімізовується затримка мережі при оновленні даних між машинами. Однак, копіюючи дані далі один від одного, створюється захист від збоїв центру обробки даних, розділів мережі та потенційно зменшується мережева затримка читання.

При реплікації дані можуть бути синхронними або асинхронними.

– синхронна реплікація означає, що дані копіюються у всі репліки, перш ніж відповісти на запит. Це має перевагу у забезпеченні однакових даних у репліках за рахунок більшої затримки запису;

– асинхронна реплікація означає, що дані зберігаються лише в одній репліці, перш ніж відповісти на запит. Це має перевагу більш швидкого запису, а недоліки - слабку послідовність даних та можливу втрату даних.

Розбиття на частини, маршрутизація запитів та реплікація є складовими блоками розподіленого сховища даних. Різні реалізації постають як різні функції та властивості, між якими ви робите компроміс [1].

Модель даних - це тип даних і спосіб їх запиту. Поширені типи включають:

– документ - вкладені колекції документів JSON. Запит з ключами або фільтрами.

– ключ-значення - пари ключ-значення. Запит з ключем.

– реляційний - таблиці рядків з явною схемою. Запит з SQL;

– двійковий об'єкт - довільні двійкові краплі. Запит з ключем;

– файлова система - каталоги файлів. Запит із шляхом до файлу;

– графік - вузли з ребрами. Запит з мовою запиту графіку;

– повідомлення - Групи пар ключ-значення, наприклад JSON або python dict. Запит із черги, теми чи відправника;

– часові ряди - дані впорядковані за міткою часу. Запит з SQL або іншою мовою запиту;

– текст - текст або журнали довільної форми. Запит з мовою запитів;

– Різні сховища даних дають різні «гарантії» поведінки. Хоча технічно не потрібні гарантії для розробки надійних додатків. Загальні гарантії такі:

– консистентність - полягає в тому, чи дані виглядають однаковими для всіх читачів і є актуальними;

– доступність - це можливість доступу до даних;

– довговічність - це те, чи зберігаються дані безпечними та непошкодженими.

Моделі хмарних обчислень бувають трьох типів:

– інфраструктура як сервіс;
– платформа як сервіс;
– програмне забезпечення як сервіс.

Інфраструктура як служба, іноді скорочена як IaaS, містить основні будівельні блоки для хмарних IT і, як правило, надає доступ до мережевих функцій, комп'ютерів (віртуальних або на спеціальному обладнанні) та простору для зберігання даних. Інфраструктура як послуга надає вам найвищий рівень гнучкості та управлінського контролю над вашими IT -ресурсами і найбільш схожа на існуючі IT -ресурси, з якими сьогодні знайомі багато IT - відділів та розробників [1].

Платформи як сервіс усувають необхідність організації керувати базовою інфраструктурою (зазвичай це апаратне забезпечення та операційні системи) і дозволяють зосередитися на розгортанні та управлінні своїми програмами. Це допомагає вам бути більш ефективними, оскільки вам не потрібно турбуватися про закупівлю ресурсів, планування потужностей, обслуговування програмного забезпечення, виправлення або будь -які інші недиференційовані важкі роботи, які беруть участь у запуску вашої програми [1].

Програмне забезпечення як послуга надає готовий продукт, яким керує постачальник послуг. У більшості випадків люди, які називають Програмне забезпечення як послугу, мають на увазі програми кінцевого користувача. З пропозицією SaaS вам не доведеться думати про те, як підтримується служба або як управляється базовою інфраструктурою; вам потрібно лише подумати про те, як ви будете використовувати саме цей програмний продукт[1].

Поширеним прикладом програми SaaS є електронна пошта в Інтернеті, де ви можете надсилати та отримувати електронну пошту без необхідності керувати доповненнями до продукту електронної пошти або обслуговуванням серверів та операційних систем, на яких працює програма електронної пошти.[1]

Отже, після аналізу предметної області можна з впевненістю сказати, що розподілене сховище потрібно дотримуватись таким трьох характеристикам:

- відмовостійкість;
- консистентність;
- доступність.

Тільки дотримування вище перелічених характеристик може забезпечити надійне хмарне сховище.

1.2 Порівняльний аналіз програмних засобів для розподілу даних

На поточний момент існує безліч методів та засобів для розподілу даних, але всі вони потребують використання однорідних за типом сховищ.

Amazon S3 - це система зберігання розподілених об’єктів. У S3 об’єкти складаються з даних та метаданих. Метадані-це набір пар імен і значень, які надають інформацію про об’єкт, наприклад дату останньої зміни. S3 підтримує стандартні поля метаданих та спеціальні метадані, визначені користувачем[4].

Об’єкти об’єднані у chunk(відерце). Користувачам Amazon S3 потрібно створити сегменти та вказати, у якому відрі зберігати об’єкти чи отримувати об’єкти. Відра - це логічні структури, які дозволяють користувачам упорядковувати свої дані. Поза кулісами фактичні дані можуть бути розподілені по великій кількості вузлів зберігання в декількох зонах доступності Amazon (AZ) в межах одного регіону. Відро Amazon S3 завжди прив’язане до певного географічного регіону-наприклад, США-Схід 1 (Північна Вірджинія)-і об’єкти не можуть виїхати з регіону[4].

Кожен об’єкт у S3 ідентифікується відром, ключем та ідентифікатором версії. Ключ - це унікальний ідентифікатор кожного об’єкта в його сегменті. S3 відстежує декілька версій кожного об'єкта, що позначається ідентифікатором версії[4].

Завдяки теоремі CAP, Amazon S3 забезпечує високу доступність і толерантність до розділів, але не може гарантувати послідовність. Натомість він пропонує можливу модель послідовності:

Коли ви вставляєте або видаляєте дані в S3, дані зберігаються безпечно, але може знадобитися деякий час для того, щоб зміни повторилися на Amazon S3 [4].

Коли відбувається зміна, клієнти, які негайно читають дані, все одно бачитимуть стару версію даних, доки зміна не розповсюдиться.

S3 гарантує атомність-коли клієнт читає об’єкт, він може переглянути стару версію об’єкта або нову версію, але ніколи не пошкоджену або часткову версію.

SeaweedFS - це проста та масштабована розподілена файлова система. Є дві цілі: зберігати мільярди файлів для швидкого обслуговування, замість того, щоб підтримувати повну семантику файлової системи POSIX, SeaweedFS вирішила реалізувати лише відображення файлу ключ–файл [4].

Huststore - це високопродуктивна розподілена система баз даних з відкритим вихідним кодом, не тільки надає послуги зберігання ключ-значення з надзвичайно високою продуктивністю, до 100 тисяч QPS, але також підтримує такі структури даних, як хеш, набір, відсортований набір тощо. Також може зберігати двійкові дані, як значення з пари ключ-значення , і, отже, може бути використаний як альтернатива Redis. Крім того, huststore реалізує розподілену чергу повідомлень шляхом інтеграції спеціального модуля HA, функцій, включаючи повідомлення Push Stream та повідомлення Publish-SubScribe, ці функції можна використовувати як заміну відповідним особливості в rabbitmq та gearman [4].

Braft - Реалізація алгоритму консенсусу RAFT промислового рівня C ++ та реплікації автомата стану на основі brpc. Braft розроблений і реалізований для сценаріїв, що вимагають великого навантаження та низьких накладних затримок, з урахуванням простих для розуміння концепцій, щоб інженери всередині Baidu могли будувати власні розподілені системи індивідуально та правильно. Побудуйте brpc, що є основною залежністю бренду [4].

Dragonboat - високопродуктивна багатогрупова бібліотека консенсусу Raft у Go з підтримкою зв'язування C ++ 11. Консенсусні алгоритми, такі як Raft, забезпечують відмовостійкість, оскільки всі системи продовжують працювати, доки доступні більшість вузлів. Наприклад, кластер Raft з 5 вузлів може прогресувати, навіть якщо 2 вузла виходять з ладу. Він також видається клієнтам

як єдиний вузол з постійною узгодженістю даних, що завжди надається. Усі запущені вузли можна використовувати для ініціювання запитів на читання для сукупної пропускної здатності читання.

Alluxio - це віртуальна розподілена система зберігання. Усуває розрив між обчислювальними рамками та системами зберігання даних, дозволяючи обчислювальним програмам підключатися до численних систем зберігання даних через спільний інтерфейс [4].

Sheepdog - це система зберігання розподілених об'єктів для служб томів та контейнерів, яка розумно керує дисками та вузлами. Sheepdog відрізняється простотою використання, простотою коду і може масштабуватися до тисяч вузлів. Абстракція томів на рівні блоків може бути приєднана до віртуальних машин QEMU та Linux SCSI Target і підтримує розширені функції управління томами, такі як знімок, клонування та тонке забезпечення [4].

Ceph - забезпечує безперебійний доступ до об'єктів, використовуючи ввімкнену рідну мову або radosgw, інтерфейс REST, сумісний із програмами, написаними для S3 та Swift. Блокований пристрій RADOS від Ceph (RBD) надає доступ до зображення блокових пристроїв, смугастів та тиражованих у всіх кластерів пам’яті. Ceph пропонує вимірювання файлової системи, сумісну з POSIX, яка прагне до високої продуктивності, велику кількість накопичення даних та максимальну сумісність із застарілими програмами.

XtreemFS - це розподілена, тиражована та відмовостійка файлова система для об’єднаних IT-інфраструктур. XtreemFS відтворює файлові дані на декількох вузлах зберігання, які можна розповсюджувати по всьому світу. Він масштабується відповідно до потреб протягом декількох хвилин, просто додаючи нове стандартне обладнання. Починаючи з одного вузла, перетворюючись на кластер і в центри обробки даних [4].

OrangeFS - це масштабована мережева файлова система, розроблена для використання у високотехнологічних обчислювальних системах (HEC), яка паралельно забезпечує високопродуктивний доступ до багатосерверного дискового сховища. Сервер та клієнт OrangeFS - це код на рівні користувача, що

робить їх дуже простими у встановленні та управлінні. OrangeFS оптимізувала підтримку MPI-IO для паралельних та розподілених додатків, вона використовується у виробничих установках і використовується як дослідницька платформа для розподіленого та паралельного зберігання [4].

BeeGFS (раніше FhGFS) - це провідна паралельна кластерна файлова система, розроблена з високою орієнтацією на продуктивність і розроблена для дуже легкої установки та управління. Він прозоро поширює дані користувачів на декілька серверів. Збільшивши кількість серверів та дисків у системі, ви можете просто масштабувати продуктивність та ємність файлової системи до потрібного вам рівня, безперебійно від невеликих кластерів до систем корпоративного класу з тисячами вузлів [4].

DAOS - Розподілене сховище асинхронних об'єктів, що визначається програмним забезпеченням з відкритим кодом, розроблене з нуля для масово розподіленої енергонезалежної пам'яті (NVM). DAOS користується перевагами технологій NVM наступного покоління, таких як пам’ять класу зберігання (SCM) та NVM експрес (NVMe), одночасно представляючи інтерфейс зберігання ключ-значення та надаючи такі функції, як транзакційний неблокуючий ввід-вивід, розширений захист даних із самовідновленням найвищий рівень обладнання, цілісність даних наскрізь, точний контроль даних та еластичне зберігання для оптимізації продуктивності та вартості [4].

Отже, щоб провести більш детальний аналіз створимо таблицю. Таблиця порівняння наведена нижче.

Таблиця 1.1 – Порівняльна характеристика існуючих аналогів

Назва	Доступність	Відмовостійкість	Консистентність	Асинхронність
Amazon S3	+	+	+	+
Braft	+	+	-	-
Dragonboat	+	-	+	+
Alluxio	+	-	-	-
Sheepdog	+	+	-	
Ceph	+	+	+	-

Продовження таблиці 1.1

Назва	Доступність	Відмовостійкість	Консистентність	Асинхронність
XtreemFS	+	+	-	+
OrangeFS	+	-	+	-
BeeGFS	+	-	-	-

Отже, після аналізу аналогів, враховуючи їх недоліки, зроблено висновок, що розробка власного програмного продукту є доцільною. Буде розроблено веб-додаток на англійській мові для розподілу даних між хмарними сховищами.

1.3 Постановка задачі дослідження

Після аналізу стану систем розподілу даних, порівняння існуючих аналогів та аналізу методів і засобів реалізації програмного продукту було визначено наступні завдання, які необхідно виконати для розробки програмного продукту:

– вдосконалити метод розподілу даних;
– інтегрувати такі хмарні сховища як Google Drive і One Drive;
– розробити модулі сервісу розподілу даних у хмарних сховищах;
– провести тестування програмного продукту.

1.4 Висновки

В цьому розділі розглянуто аналіз стану програмних засобів для розподілу даних, що показав, що створення власного ПЗ є актуальним. Було розглянуто такі аналоги як: Amazon S3, Braft, Dragonboat, Alluxio, Sheepdog, Ceph, XtreemFS, OrangeFS, BeeGFS. Порівняння існуючих аналогів на даний момент часу показало, що є необхідність розробити власний веб-додаток. Також було проведено аналіз методів реалізації програмного продукту. Було обрано метод розподілу даних Mirror Mode. В результаті було розроблено основні завдання, які необхідно виконати при розробці програмного продукту.

2 МЕТОДИ ТА МОДЕЛІ РОЗПОДІЛУ ДАНИХ

2.1 Аналіз методів розподілу даних

На даний момент часу існують такі методи розповсюдження даних: Mirror Mode, Simple Distribution Mode, DistributeSendMinimal, Batch Mode[5].

Mirror Mode - у дзеркальному режимі DIH копіює всі дії індексу на всі дочірні сервери, щоб створити декілька ідентичних копій індексу.

DIH не аналізує дані документа, а це означає, що можливо використовувати параметр PreserveDREADD, щоб надіслати вихідну дію індексу на дочірні сервери, зменшуючи навантаження на мережу. Часто можна виключити режим дзеркала з багаторівневої архітектури, використовуючи групи серверів[5].

Переваги:

– цей метод корисний для відмов і балансування навантаження;

– dih не аналізує дані документа, а це означає, що ви можете використовувати параметр preservedreadd, щоб надіслати вихідну дію індексу на дочірні сервери, зменшуючи навантаження на мережу.

Недоліки:

– цей метод не розподіляє дані між кількома серверами;

– часто можна виключити режим дзеркала з багаторівневої архітектури, використовуючи групи серверів.

Simple Distribution Mode - у цьому методі режим дзеркала вимкнено, але не вмикаються інші параметри розповсюдження[5].

DIH копіює всі дії індексування на всі дочірні сервери, але змінює дії додавання, щоб кожен дочірній сервер індексував лише частину документів.

Дочірній сервер не індексує інші документи, але використовує їх для дедуплікації з наявним вмістом відповідно до параметрів KillDuplicates[5].

За замовчуванням DIH розміщує весь файл даних на кожному дочірньому сервері, який може бути потенційною мережею або вузьким місцем введення-виводу[5].

Переваги:

– можливість видалити дублікати відповідно до вмісту будь-якого поля;

– можливість додати дочірні сервера у будь-який час;

– dih не аналізує дані документа, а це означає, що можна використовувати параметр preservedreadd, щоб надіслати вихідну дію індексу на дочірні сервери, зменшуючи навантаження на мережу[5];

– можливість зважити дочірні сервери або вказати, що дочірній сервер повинен оновлювати лише наявні документи;

– можливість автоматично обробляти сервери, які досягли налаштованої кількості документів, як updateonly, і за бажанням вибрати заповнення фіксованої кількості серверів за раз;

– можливість використати параметри перерозподілу, щоб індексувати лише активні дочірні сервери[5].

Недоліки:

– відсутня можливість використовувати методи дедуплікації, які зберігають наявний документ.

– за замовчуванням dih розміщує весь файл даних на кожному дочірньому сервері, який може бути потенційною мережею або вузьким місцем введення-виводу[5].

DistributedSendMinimal – у цьому режимі DIH поширює документи так саме, як і в режимі Simple Distribution. Кожен сервер отримує змінену дію додавання індексу, яка вказує йому індексувати лише частину отриманих документів [5].

Однак, на відміну від простого режиму, DIH аналізує та змінює дані, які він надсилає дочірнім серверам. Для документів, які дочірній сервер повинен індексувати, він надсилає повний текст документа. Для всіх інших документів DIH надсилає лише мінімальне представлення, яке містить лише поля посилання, які сервер повинен використовувати для видалення будь-яких існуючих документів, які тепер є дублікатами [5].

Переваги:

– існує велика потенційна економія обсягу даних, які dih надсилає своїм дочірнім серверам, порівняно із простим режимом розповсюдження;

– більшість функцій простого режиму розповсюдження все ще доступні, наприклад зважування, повага дочірнього наповнення та режим дочірнього сервера updateonly [5].

Недоліки:

– dih висуває підвищені вимоги до процесора та тимчасового дискового простору, коли він готує дочірні копії вхідних даних;

– опція preservedreadd недоступна.

Batch Mode – у цьому режимі дії додавання індексу чергуються між дочірніми серверами, так що кожна дія надходить на один дочірній сервер.

Переваги [5]:

– цей режим є найлегшим режимом розповсюдження. dih не аналізує жодних даних, і всі дані передаються на один дочірній сервер;

– dih не аналізує дані документа, а це означає, що ви можете використовувати параметр preservedreadd, щоб надіслати вихідну дію індексу на дочірні сервери, зменшуючи навантаження на мережу;

– ви можете розподілити документи відповідно до того, наскільки заповнені дочірні сервери, або вказати, що дочірній сервер повинен оновлювати лише наявні документи;

– ви можете використовувати параметри перерозподілу, щоб індексувати лише активні дочірні сервери.

Недоліки [5]:

– ви не можете видалити дублікати документів на всіх ваших серверах;

– цей метод може бути непридатним, якщо ви індексуєте документи нечасто або особливо великими партіями різного розміру. у цих випадках дані можуть бути розподілені нерівномірно.

Взаємне виключення є фундаментальною проблемою в розподілених обчислювальних системах [5].

Також, окрім звичайних методів розподілу, є ще алгоритми взаємного виключення, які дають можливість розподілу даних в асинхронному режимі.

Взаємне виключення гарантує, що одночасний доступ процесів до спільного ресурсу або серіалізації даних виконується взаємовиключним способом. Взаємне виключення в розподіленій системі стверджує, що тільки одному процесу дозволяється виконувати критичну секцію в будь-який момент часу. У розподіленій системі спільні змінні або локальне ядро не можуть використовуватися для реалізації взаємного виключення [6]. Передача повідомлень є єдиним засобом для реалізації розподіленого взаємного виключення. Рішення про те, якому процесу дозволено наступний доступ до критичної секції, приймається шляхом передачі повідомлень, при якій кожен процес дізнається про стан усіх інших процесів певним послідовним способом. Розробка алгоритмів розподіленого взаємного виключення є складною, оскільки цим алгоритмам доводиться мати справу з непередбачуваними затримками повідомлень і неповним знанням стану системи [6]. Існує три основних підходи для реалізації розподіленого взаємного виключення:

- підхід на основі токенів;
- підхід, не заснований на токенах;
- підхід на основі кворуму.

У підході на основі токенів унікальний токен розподіляється між процесами. Процесові дозволено входити в свою критичну секцію, якщо володіє маркером, і/та продовжує утримувати маркер, доки виконання критичної секції не закінчиться. Взаємне виключення забезпечується, оскільки токен унікальний [6]. Алгоритми, засновані на основі токенів, істотно відрізняються тим, як процес здійснює пошук токена. У підході без маркерів, два або більше послідовних раундів повідомлень обмінюються процесами, щоб визначити, який процес наступним увійде до критичної секції. Процес переходить у

критичну секцію, коли твердження, визначене для його локальних змінних, стає істинним [6].

Взаємне виключення застосовується, оскільки твердження стає істинним лише на одному процесові в будь-який момент часу.

У підході на основі кворуму кожен процес запитує дозвіл на виконання критичної секції з підмножини процесів(так званий кворум) [6].

Кворуми формуються таким чином, що коли два процеси одночасно запитують доступ до критичної секції, принаймні один з них отримує обидва запити, і цей процес відповідає за те, щоб лише один запит виконувався в критичній секції в будь-який момент час [6].

Система складається з N вузлів, S1, S2, ..., SN . Без втрати загальності припустимо, що на кожному вузлі виконується один процес. Процес на місці Si позначається Pi. Усі ці процеси спілкуються асинхронно через базову комунікаційну мережу. Процес, який бажає увійти в критичну секцію, запитує всі інші або підмножину процесів, надсилаючи повідомлення REQUEST, і чекає відповідних відповідей, перш ніж увійти до критичної секції [6]. Під час очікування процесу не дозволяється робити подальші запити на вхід до критичної секції.

Процес може перебувати в одному з наступних трьох станів: запитує критичну секцію(request of critical section), виконує критичну секцію(performing of critical section) або не запитує і не виконує критичну секцію(does not request/performing of critical section), тобто неактивний. У стані «Request of critical section» процес заблокований і не може робити подальші запити для критичної секції. У стані очікування процес виконується за межами критичної секції. В алгоритмах на основі токенів процес також може перебувати в стані, що містить маркер, виконується за межами критичної секції. Такий стан іменується станом неактивного маркера [6]. У будь-який момент на процесі може бути декілька незавершених запитів на критичну секцію. Процес ставить ці запити в чергу і обслуговує їх по одному. Це специфічний алгоритм.

Деякі алгоритми взаємного виключення розроблені для вирішення таких ситуацій. Багато алгоритмів використовують логічні годинники в стилі Лампорта для призначення позначки часу запитам критичного розділу [6].

Позначки часу використовуються для визначення пріоритету запитів у разі конфлікту. Загальне правило, яке дотримується, полягає в тому, що чим менше мітка часу запиту, тим вище його пріоритет для виконання у критичній секції.

Алгоритм взаємного виключення повинен задовольняти наступним властивостям [6]:

– властивість безпеки стверджує, що в будь-який момент часу лише один процес може виконати критичну секцію. Це суттєва властивість алгоритму взаємного виключення [6];

– властивість «життя» свідчить про відсутність тупиків(deadlock) і голодування(starvation of thread). Два або більше процеси не повинні нескінченно чекати повідомлень, які ніколи не прийдуть. Крім того, процес не повинен чекати нескінченно довго, щоб виконати критичну секцію, поки інші процеси неодноразово виконують її. Тобто, кожен процес-запитувач повинен отримати можливість виконати критичну секцію за скінченний час [6];

– справедливість у контексті взаємного виключення означає, що кожен процес отримує справедливий шанс виконати критичну секцію. В алгоритмах взаємного виключення властивість справедливості зазвичай означає, що запити на виконання критичної секції виконуються в порядку їх надходження в систему (час визначається логічним годинником).

Перша властивість є абсолютно необхідною, а дві інші властивості вважаються важливими в алгоритмах взаємного виключення [6].

Показники продуктивності. Ефективність алгоритмів взаємного виключення зазвичай вимірюється за допомогою наступних чотирьох показників:

– складність повідомлень – це кількість повідомлень, необхідних для виконання критичної секції процесом;

– затримка синхронізації - це час, необхідний до того, як наступний процес увійде в критичну секцію. Зазвичай, один або кілька послідовних обмінів повідомленнями можуть знадобитися після того, як процес виходить із критичної секції і до того, як наступний процес може увійти до критичної секції;

– час відповіді - це інтервал часу, протягом якого запит очікує завершення виконання критичної секції після того, як його повідомлення запиту були відправлені. Таким чином, час відповіді не включає час очікування запиту на сайті до того, як повідомлення запиту будуть надіслані [6];

– системна пропускна здатність - це швидкість, з якою виконуються запити на критичну секцію. Якщо SD – це затримка синхронізації, а E – середній час виконання критичної секції, то пропускна здатність визначається наступним рівнянням [6]:

$$ST = \frac{1}{(SD + E)}$$

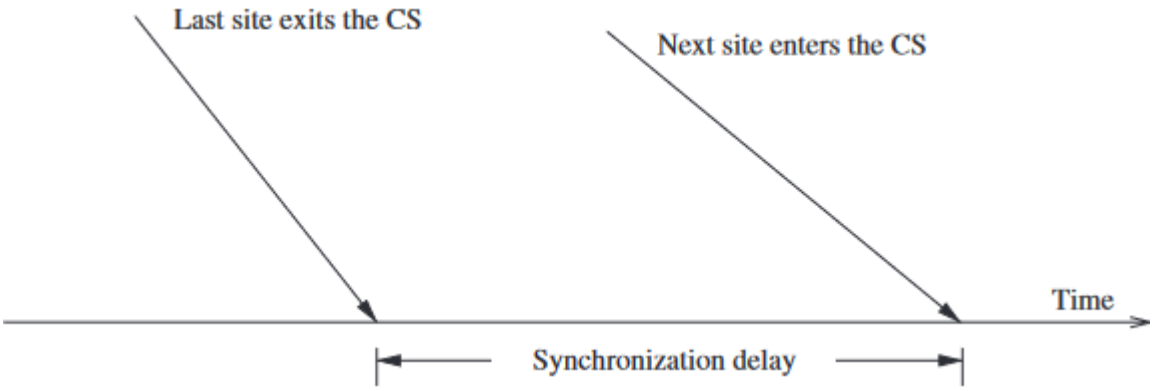

Рисунок 2.1 – Схема затримки синхронізації

Як правило, значення показника ефективності статистично коливається від запиту до запиту, і зазвичай розглядається середнє значення такого показника [6].

Навантаження визначається швидкістю надходження запитів на виконання критичної секції.

Ефективність алгоритму взаємного виключення залежить від навантаження [6]. В умовах низького навантаження рідко існує більше одного

запиту на критичну секцію, присутній в системі одночасно. За умов великого навантаження завжди є запит на критичний розділ на об’єкті. Таким чином, в умовах великого навантаження після виконання запиту процес негайно починає дії для виконання наступного запиту критичної секції. Процес рідко перебуває в стані простою в умовах великого навантаження [6].

Як правило, алгоритми взаємного виключення мають найкращі та найгірші випадки для показників продуктивності. У кращому випадку переважають умови такі, що показник продуктивності досягає найкращого можливого значення. Наприклад, у більшості алгоритмів взаємного виключення найкращим значенням часу відповіді є затримка повідомлення в обидві сторони плюс час виконання критичної секції, 2T + E. Часто для алгоритмів взаємного виключення найкращий і найгірший випадки збігаються з низьким і високим навантаженням відповідно. Наприклад, найкраще та найгірше значення часу відгуку досягаються, коли навантаження, відповідно, низьке та високе; в деяких алгоритмах взаємного виключення найкращий і найгірший трафік повідомлень генерується при низькому і великому навантаженні відповідно [6].

Алгоритм Лемпорта - розподілений алгоритм взаємного виключення є справедливим у тому сенсі, що запит на критичну секцію виконується в порядку їх часових позначок, а час визначається логічними годинниками. Коли процес обробляє запит на критичну секцію, він оновлює свій локальний годинник і призначає запиту мітку часу [6].

Алгоритм виконує запити на критичну секцію в порядку зростання часових позначок. Кожен процес Si зберігає чергу, яка містить запити на взаємне виключення, упорядковані за їх мітками часу. Ця черга відрізняється від черги, яка містить локальні запити на виконання критичної секції, які очікують своєї черги. Цей алгоритм вимагає, щоб канали зв'язку доставляли повідомлення в порядку FIFO [6].

Коли процес Si хоче увійти в критичну секцію, він передає запит повідомлення на всі інші процеси та поміщає запит в чергу позначає мітку часу запиту [6].

Коли процес Sj отримує повідомлення запиту від процесу Si, він розміщує запит процесу Si в чергу і повертає Si повідомлення відповіді з міткою часу.

Процес Si входить до критичної секції, якщо виконуються дві наступні умови L1 та L2.

L1. Процес Si отримав повідомлення з міткою часу більше, ніж з усіх інших процесів [6].

Процес Si, виходячи з критичної секції, видаляє свій запит з початку черги запитів і передає всім іншим процесам повідомлення вивільнення з міткою часу.

Коли процес Sj отримує повідомлення вивільнення від процесу Si, він видаляє запит Si зі своєї черги запитів.

Коли процес видаляє запит зі своєї черги запитів, його власний запит може опинитися у верхній частині черги, дозволяючи йому увійти до критичної секції. Очевидно, що коли процес отримує повідомлення REQUEST, REPLY або RELEASE, він оновлює свій годинник, використовуючи позначку часу в повідомленні [6].

Алгоритм Рікарта – Агравали передбачає, що канали зв'язку є FIFO. Алгоритм використовує два типи повідомлень: REQUEST і RESPONSE. Процес надсилає повідомлення REQUEST всім іншим процесам, щоб запросити їх дозвіл увійти до критичного розділу. Процес надсилає повідомлення RESPONSE процесу, щоб надати йому дозвіл [6]. Процеси використовують логічні годинники в стилі Лампорта для призначення позначки часу запитам критичного розділу [6]. Позначки часу використовуються для визначення пріоритету запитів у разі конфлікту – якщо процес pi, який очікує на виконання критичного розділу отримує повідомлення REQUEST від процесу pj, тоді, якщо пріоритет запиту pj нижчий, pi відкладає відповідб до pj і надсилає повідомлення RESPONSE до pj лише після виконання CS для свого запиту, що очікує на розгляд. В іншому випадку pi негайно надсилає повідомлення RESPONSE до pj, за умови, що на даний момент він не виконується у критичній секції. Таким чином, якщо кілька процесів запитують виконання критичної секції, запит з

найвищим пріоритетом успішно збирає всі необхідні повідомлення REPLY і отримує можливість виконати критичну секцію [6].

Кожен процес pi підтримує масив відкладеного запиту, RDi, розмір якого дорівнює кількості процесів у системі. Спочатку ∀i ∀j: RDi j =0. Щоразу, коли pi відкладає запит, надісланий pj, він встановлює RDi j =1, і після того, як він надіслав повідомлення RESPONSE до pj, він встановлює RDi j =0.

Коли процес Si хоче увійти в критичну секцію, він передає мітку часу на всі інші процеси [6].

Коли процес Sj отримує повідомлення REQUEST від процесу Si, він надсилає RESPONSE повідомлення процесу Si, якщо процес Sj не запитує і не виконує критичну секцію, або якщо процес Sj запитує, а мітка часу запиту Si є менше, ніж мітка часу власного запиту сайту Sj. Інакше відповідь така відкладається і Sj встановлює RDj i =1.

Процес Si входить до критичної секції після того, як він отримав повідомлення RESPONSE від кожного процесу, на який він надіслав повідомлення запиту [6].

Коли процес Si виходить із критичної секції, він надсилає всі відкладені повідомлення відповіді: ∀j, якщо RDi j = 1, потім надсилає повідомлення відповіді на Sj і встановлює RDi j =0.

Коли процес отримує повідомлення, він оновлює свій годинник, використовуючи позначку часу в повідомленні. Крім того, коли сайт приймає запит на критичну секцію для обробки, він оновлює свій локальний годинник і призначає запиту мітку часу. У цьому алгоритмі повідомлення відповіді процесу блокуються лише процесами, які запитують критичну секцію з вищим пріоритетом. Таким чином, коли процес надсилає відкладені повідомлення відповіді, процес із наступним найвищим пріоритетом запиту отримує останнє необхідне повідомлення відповіді та входить до критичної секції. Виконання запитів критичної секції в цьому алгоритмі завжди відбувається в порядку їх часових позначок [6].

Алгоритм динамічної інформаційної структури Сінгхала. Більшість алгоритмів взаємного виключення використовують статичний підхід для виклику взаємного виключення, тобто вони завжди вживають однакових дій, щоб викликати взаємне виключення, незалежно від стану системи. Проблема цих алгоритмів полягає в недостатній ефективності, оскільки ці алгоритми не можуть використовувати умови, що змінюються в системі. Алгоритм може використовувати динамічні умови системи для оптимізації продуктивності. Наприклад, якщо кілька процесів дуже часто звертаються до взаємного виключення, а інші процеси використовують взаємне виключення набагато рідше, то процесу, який часто викликає, не потрібно запитувати дозвіл процесу, який рідше викликає, щоразу, коли він запитує доступ до критичної секції. Потрібно лише отримати дозвіл з усіх інших процесів, які часто викликаються. На основі цього спостереження Сінгхал розробив адаптивний алгоритм взаємного виключення [6].

Інформаційна структура алгоритму розвивається з часом, оскільки процеси дізнаються про стан системи через повідомлення. Алгоритми взаємного виключення з динамічною інформаційною структурою є привабливими, оскільки вони можуть адаптуватися до коливань системних умов для оптимізації продуктивності. Розробка таких адаптивних алгоритмів взаємного виключення є складною, через такі проблеми проектування [6]:

– як процес знає, які інші процеси зараз активно використовують взаємне виключення;

– як робиться взаємне виключення, коли процесу, який рідше звертається, потрібно задіяти взаємне виключення;

– як гарантується взаємне виключення, якщо процес не отримує дозвіл від усіх інших процесів;

– як гарантується, що алгоритм динамічного взаємного виключення працює не витрачаючи ресурси та час на збір стану системи.

Розглядається розподілена система, що складається з n автономних процесів, скажімо, S1, S2, , Sn, які з'єднані мережею зв'язку. Припускається, що процеси взаємодіють повністю за допомогою передачі повідомлень.

Затримка поширення повідомлень є кінцевою, але непередбачуваною, і між будь-якою парою процесів повідомлення доставляються в порядку їх надсилання. Для зручності презентації припустимо, що базова комунікаційна мережа є надійною і процеси не виходять з ладу. Проте були запропоновані методи відновлення після втрат повідомлень і збоїв процесу.

2.2 Удосконалення методу Mirror Mode

Оскільки, даний метод не розподіляє дані між декількома серверами ми можемо його удосконалити, а саме, додати розподілення даних у сховищах, які є не прив’язаними локально або до місцеположення системи і які можна горизонтально масштабувати у будь-який проміжок часу, а саме хмарні сховища.

Отже, щоб краще уявити як працює метод розподілу Mirror Mode з хмарними сховищами, давайте поглянемо на структурну схему наведену на рисунку на рисунок 2.2.

Як наведено на рисунку 2.2 при вдосконаленні методу розподілу даних Mirror Mode з'являється додатковий блок, який передає та зберігає дані у хмарних сховищах. Також, щоб мати повну уяву з хмарними сховищами поглянемо на рисунок 2.3.

Представимо користувача, який хоче зберегти свою дані на декількох хмарних сховищах. Отже, користувач обирає файл, який для нього є критичним, під словом критичним мається на увазі важливий або з важливою інформацією, який потрібно зберегти на декількох хмарних сховищам, що надав користувач, щоб зменшити імовірність його втрати.

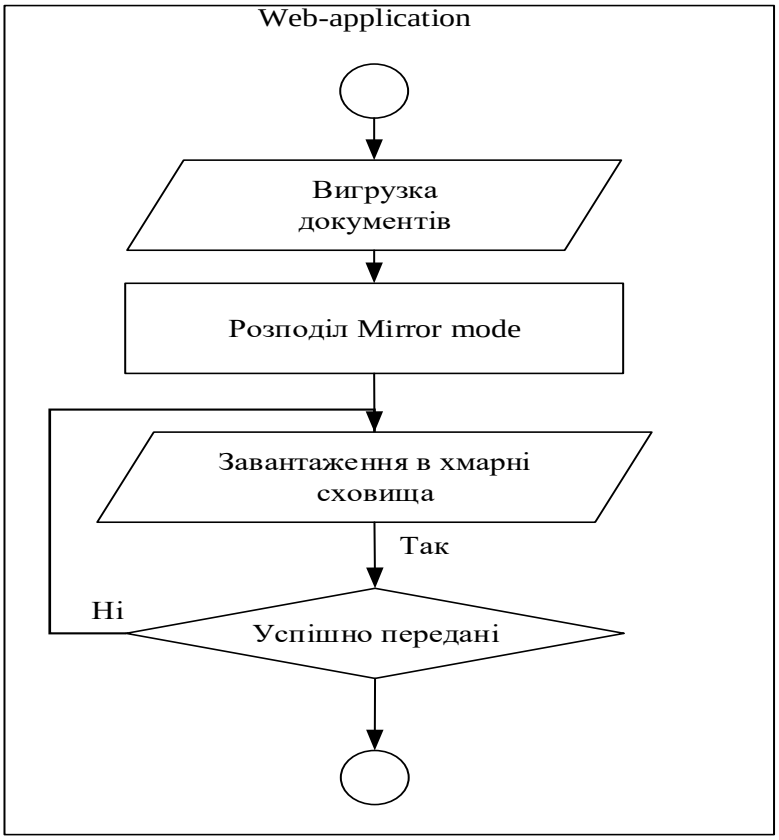

Рисунок 2.2 - Загальна схема Mirror Mode з використанням хмарних сховищ

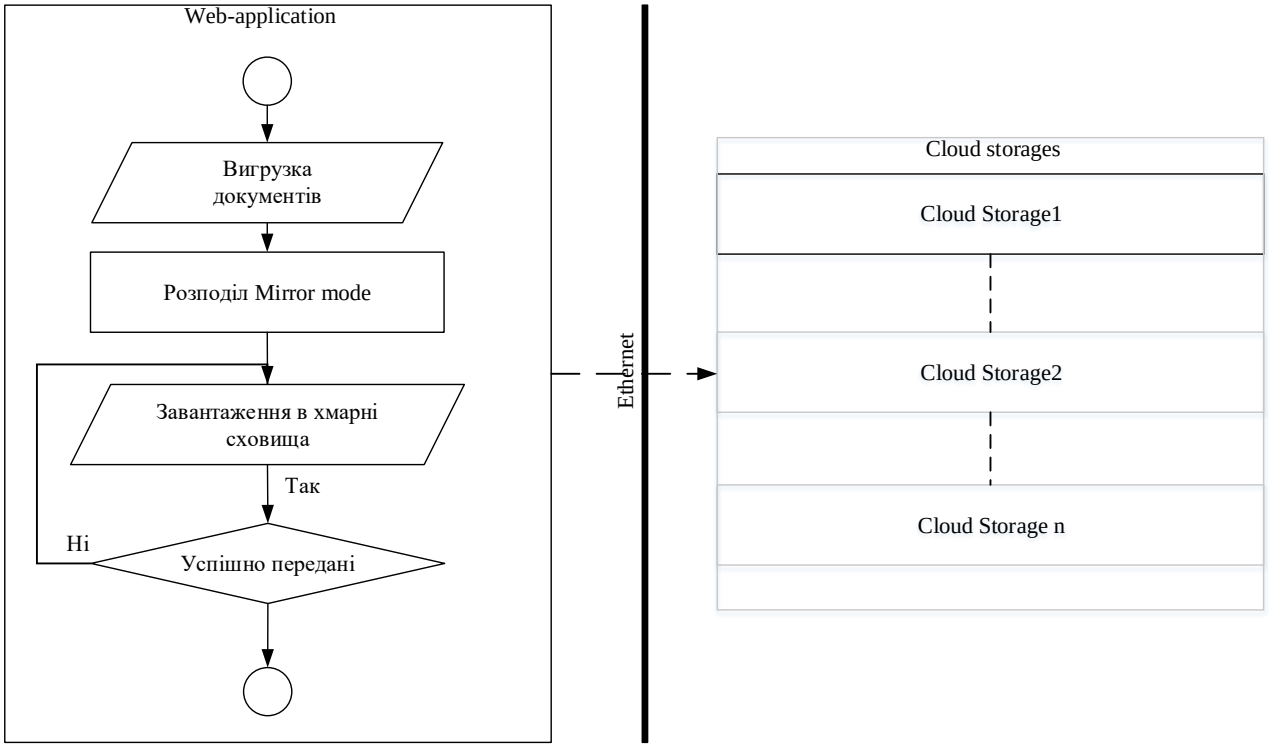

Рисунок 2.3 – Загальна схема вдосконаленого методу Mirror Mode

Тоді, кроки будуть такими:

– користувач заходить до облікового запису;
– переходить на сторінку вивантаження файлів;
– обирає критичні файли;
– надає доступ до хмарних сховищ;
– вивантаження обраних файлів до хмарних сховищ.

Отже, як видно з кроків описаних вище, користувачу потрібно лише надати доступ до свої хмарних сховищ щоб вивантажити файли.

2.3 Вибір технологій реалізації компонентів системи

Щоб створити web-додаток для розподілу даних потрібно підготовити середовище розробки та додати необхідні бібліотеки та фреймворки. Для цього переходимо на офіційний сайт Java бібліотек Maven Repository та обираємо необхідні бібліотеки для реалізації додатку, а саме:

– spring boot;
– spring boot data;
– jdbc postgresql;
– google-oauth-client-jetty
– flyway-core;
– Lombok;
– r2dbc-postgresql;
– google-api-client
– spring boot configuration;

Щоб максимально спростити роботу з залежностями, які необхідні проектові, використаємо фреймворк для збірки Gradle.

Gradle – це автоматизоване середовище збірки проекта в залежності від конфігурації.

Щоб додати бібліотеки до проекту, потрібно відкрити gradle.build файл та додати необхідні бібліотеки для підвантаження. На рисунку 2.4 наведено додавання необхідних бібліотек.

Після імпорту необхідних залежностей потрібно обов'язково оновити проект, щоб зміни відбулись та застосувались зміни після додавання бібліотек.

```
34
35  dependencies {
36      implementation group: 'org.springframework.boot', name: 'spring-boot-starter-data-r2dbc', version: springVersion
37      implementation group: 'org.springframework.boot', name: 'spring-boot-starter-data-jdbc', version: springVersion
38      implementation group: 'org.springframework.boot', name: 'spring-boot-starter-actuator', version: springVersion
39      implementation group: 'org.springframework.boot', name: 'spring-boot-starter-security', version: springVersion
40      implementation group: 'org.springframework.boot', name: 'spring-boot-starter-webflux', version: springVersion
41
42      implementation group: 'com.google.api-client', name: 'google-api-client', version: googleApiClientVersion
43      implementation group: 'com.google.oauth-client', name: 'google-oauth-client-jetty', version: googleOauthClientVersion
44      implementation group: 'com.google.apis', name: 'google-api-services-drive', version: googleApiDriveServicesVErsion
45      implementation group: 'org.flywaydb', name: 'flyway-core', version: flywayVersion
46      implementation group: 'org.postgresql', name: 'postgresql', version: postgresJdbcDriverVersion
47
48
49      compileOnly group: 'org.projectlombok', name: 'lombok', version: lombockVersion
50      annotationProcessor group: 'org.projectlombok', name: 'lombok', version: lombockVersion
51
52      developmentOnly group: 'org.springframework.boot', name: 'spring-boot-devtools', version: springVersion
53      annotationProcessor group: 'org.springframework.boot', name: 'spring-boot-configuration-processor', version: springVersion
54
55      implementation group: 'io.r2dbc', name: 'r2dbc-postgresql', version: postgresR2dbcVersion
56
57      testImplementation group: 'org.springframework.boot', name: 'spring-boot-starter-test', version: springVersion
58      testImplementation group: 'io.projectreactor', name: 'reactor-test', version: projectreactorVersion
59  }
```

Рисунок 2.4 – Додавання необхідних бібліотек

Отже, після того, як ми виконали необхідні кроки, наше середовище налаштоване та готове до подальшої розробки програмного продукту.

2.4. Висновки

У даному розділі було зроблено варіантний аналіз методів розподілу даних. Розглянуто специфіку розподілу даних, запропоновано удосконалення методу розподілу даних Mirror Mode з використанням хмарних сховищ.

Проведено їх аналіз та опис основних положень, та їх особливостей. алгоритмічну схему розподілу даних з використанням хмарних сховищ, також наведено загальну структурну схему комунікації додатку розподілу даних та хмарних сховищ.

3 АНАЛІЗ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ДОДАТКА РОЗПОДІЛУ ДАНИХ МІЖ ХМАРНИМИ СХОВИЩАМИ

3.1 Аналіз архітектурного підходу для розробки додатку

Користування сучасними додатки передбачає як так вимогу комфорт,

тому, щоб виконати цю вимогу потрібно розробити графічний інтерфейс, а саме веб-інтерфейс.

Для побудови ефективної системи розподілу даних, потрібно визначити архітектуру системи. Архітектура системи впливає не тільки на початковий етап розробки та проектування але і на подальші зміни та вдосконалення, також від обрання правильного архітектурного підходу можливо зменшити витрати на саму розробку. Отже, що таке веб-додаток.

Веб-додаток — це прикладна програма, яка зберігається на віддаленому сервері. Веб-сервіси за визначенням є веб-додатками, і багато веб-сайтів, хоча й не всі, містять веб-програми. Загальна схема вед-додатка наведена на рисунку 3.1.

Веб-додатки можуть бути розроблені для широкого спектру використання і можуть використовуватися будь-ким; від організації до окремої особи з багатьох причин. Зазвичай, використовувані веб-додатки можуть включати веб-пошту, онлайн-калькулятори або магазини електронної комерції [7].

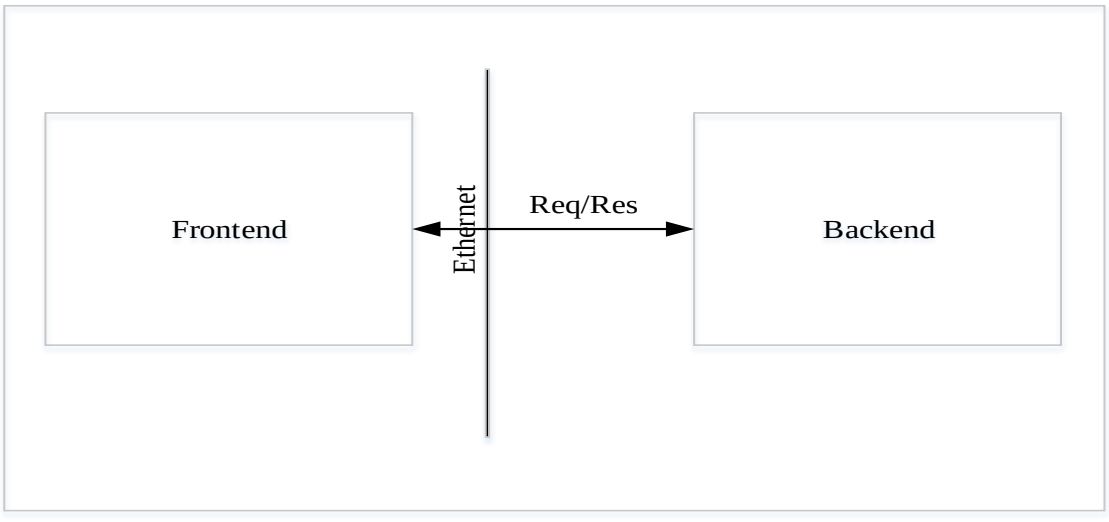

Рисунок 3.1 – Загальна структура веб-додатка

До деяких веб-додатків можна отримати доступ лише через певний браузер; однак більшість із них доступні незалежно від браузера[7].

В основному структуру веб-додатків можна розділити на два рівня, таких як frontend та backend. Кожний з цих рівнів має свою інфраструктуру та є незалежним від іншого, тобто, якщо сервер backend-у вийде з ладу, то fronted буде працювати, але без корисного навантаження, так саме, якщо і сервер frontend-у вийде з ладу то backend сервер буде спокійно працювати[7].

Якщо говорити про загальні web-принципи, то під цим передбачається використання клієнт-серверної архітектури. Отже, клієнт це веб-додаток, який розміщений на окремому веб-сервері та здійснює запити на backend для отримання даних, щоб відобразити їх користувачеві.

Веб-додатки є декількох типів:

– статичний веб-додаток;
– динамічний веб-додаток;
– e-commerce;
– веб-додаток порталу;
– система управління контентом.

Статичний веб-додаток зазвичай розробляються лише з використанням таких технологій як HTML і CSS. Однак анімовані об’єкти, такі як банери, GIF-файли, відео тощо, також можуть бути включені та показані в них. Їх також можна розробляти за допомогою jQuery та Ajax [7].

Прикладами розробки статичних веб-додатків є професійні портфоліо або цифрові навчальні програми [7].

Подібним чином веб-сторінка, що представляє компанію, також може використовувати такий тип веб-програми для відображення контактної інформації тощо.

Крім того, змінювати вміст статичних веб-програм непросто. Для цього спочатку потрібно завантажити HTML-код, потім змінити його і, нарешті, знову завантажити на сервер. Ці зміни можуть бути внесені лише веб-майстром або компанією-розробником, яка спочатку спланувала та розробила програму[7].

Динамічні веб-додатки набагато складніші на технічному рівні. Вони використовують бази даних для завантаження даних, і їх вміст оновлюється щоразу, коли користувач звертається до них. Зазвичай вони мають панель адміністрування (так звану CMS), з якої адміністратори можуть виправляти або змінювати вміст програми, включаючи текст та зображення [7].

Для динамічної розробки веб-додатків можна використовувати багато різних мов програмування.

Динамічні веб-додатки зазвичай мають панель адміністратора (CMS) для внесення змін.

– e-commerce, процес розробки такого додатка є складнішим, оскільки він повинен включати електронні платежі за допомогою кредитних карток, PayPal або інших способів оплати.

– Портал, під порталом мається на увазі своєрідний додаток, в якому можливо отримати доступ до кількох його розділів або категорій через домашню сторінку.

– Система управління контентом, контент необхідно постійно оновлювати, коли мова йде про розробку веб-додатків, тому встановлення системи керування вмістом (CMS) є серйозним варіантом, який варто розглянути. Адміністратор може використовувати цю CMS для внесення змін та оновлень.

3.2 Розробка загальної структурної схеми додатку для розподілу даних між хмарними сховищами

Логічний потік веб-додатку передбачає такі етапи:

– реєстрація користувача;

– надання доступу до хмарних сховищ;

– вивантаження даних у хмарні сховища;

– перегляд вивантажених даних;

– пошук у хмарних сховищах;

– завантаження вивантажених даних.

Для відображення поведінки системи та її взаємозв’язків можна використати діаграму лояльності. Діаграма лояльності відображає логічні потоки за якими відбуваються основні процеси у додаткові. За допомогою мови моделювання UML(Unified Modeling Language) здійснимо візуалізацію логічного потоку веб-додатка. Модель діяльності системи зображена на рисунку

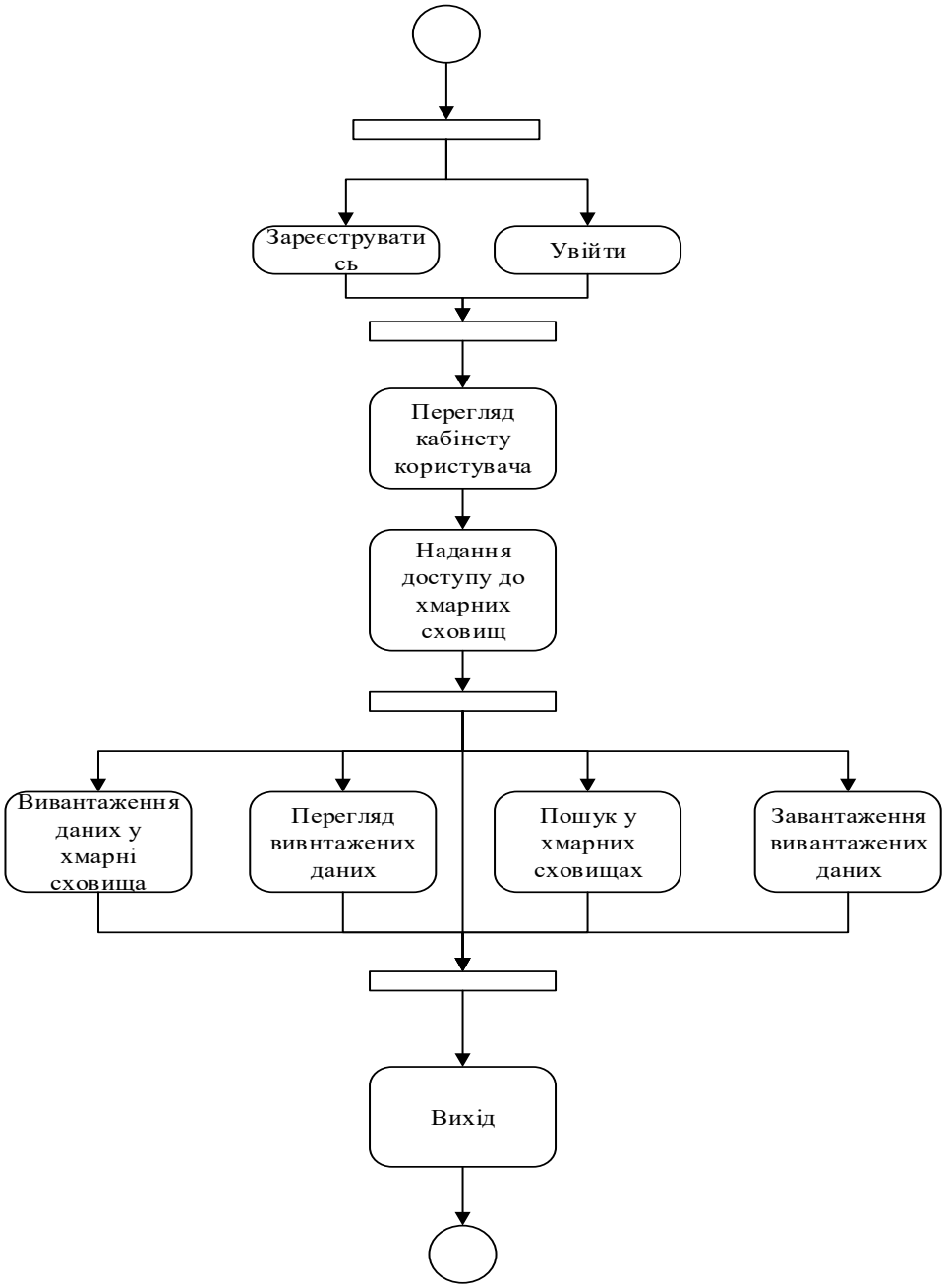

Рисунок 3.2 – Модель діяльності системи

Використання системи, буде простим, оскільки немає зайвого функціоналу та навантаженого користувацького інтерфейсу.

Система розподілу даних між хмарними сховищами буде складатись з таких модулів:

– модуль авторизації та автентифікації;
– модуль взаємодії з хмарними сховищами;
– модуль операцій над даними.

Схема зв’язків між модулями системи для розподілу даних між хмарними сховищами наведена на рисунку 3.3.

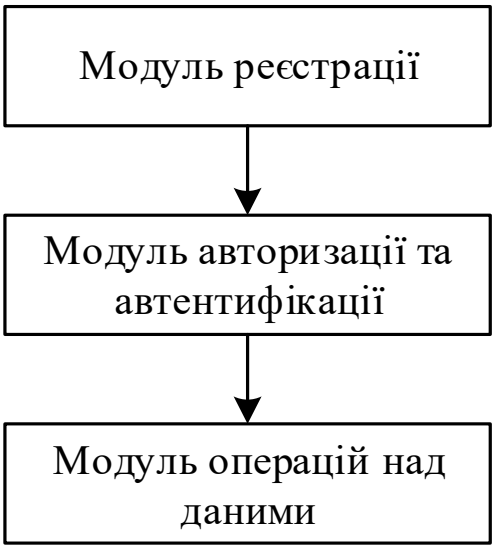

Рисунок 3.3 – Схема зв’язків між модулями

Робота системи починається із взаємодії користувача з модулем реєстрації. Для того, щоб використовувати систему для розподілу даних між хмарними сховищами новим користувачам потрібно створити обліковий запис та пройти автентифікацію та авторизацію. Для того щоб створити обліковий запис користувачеві потрібно заповнити такі поля, як: username, firstname, lastname, email, password. Після реєстрації/ автентифікації та авторизації користувач потрапить у свій особистий кабінет, у якому він зможе виконувати такі дії:

– завантаження даних у хмарні сховища;
– вивантаження даних;
– пошук у хмарних сховищах.

Також, усі користувачі мають однакову роль «USER», яка дозволяє їм редагувати дані свого облікового запису.

3.3 Проектування компонентів веб-додатку розподілу даних між хмарними сховищами

Першим етапом розробки будь-якого додатка або системи є обрання методу проектування та структурного аналізу. Цей етап є важливим оскільки потрібно розробити структурну та функціональну схеми. Є 2 загальних моделі: IDEF3 та Об’єктно орієнтований підхід.

IDEF3 – метод моделювання, який являється частиною сімейства стандартів IDEF, був розроблений в кінці 1980-х років. Цей метод призначений для таких моделей процесів, в яких важливо знати послідовність виконання дій та взаємозв’язок між ними.

UOW(Unity of Work) являються центральними компонентами моделі. В IDEF3 роботи візуалізовуються прямокутниками та мають ім’я, яке позначає процес дії та її ідентифікатор.

Зв’язки IDEF3 відображають взаємовідносини робіт. Всі зв’язки IDEF3 є одно направленими і мають наступний тип:

- тимчасове попередження;
- об’єктний потік;
- нечіткий потік.

Перехрестя – використовуються для відображення логіки взаємодії стрілок при зливанні та розгалуженні або для відображення декількох подій, які можуть або повинні бути завершеними пере виконання наступного завдання.

Об’єктно-орієнтований підхід – слідує ітеративному процесу з нарощуванням можливостей. Єдина модель конкретизується на етапах аналізу, проєктування і реалізації – в результаті успішних ітерацій додаються нові деталі, при необхідності вносяться зміни та покращення.

О’бєктно орієнтований підхід має такі переваги:

– використання об’єктного підходу значно збільшує рівень уніфікації розробки та придатності для повторного використання, що призводить до створення середовища розробки і переходу до складального створення моделей;

– об’єктна декомпозиція дозволяє уникнути створення складних моделей, так як воно припускає еволюційний шлях розвитку моделі на базі відносно невеликих систем;

– об’єктна модель є природною, оскільки орієнтована на людське сприйняття світу.

– До недоліків об’єктно орієнтованого підходу можна віднести:

– високі початкові затрати. цей підхід не дає моментальної віддачі. ефект від його застосування позначається після розробки двох-трьох проєктів і накоплення повторно використаних компонентів;

– складно реалізувати керування проєктом;

– зростає складність рішення.

Отже, найліпшим варіантом для розробки даного додатка буде об’єднання цих двох методологій.

Проектування це важливий етап у розробці будь-якого додатка та/або системи. В програмуванні з використанням UML(Unified Modeling Language) можна розробити структурну, поведінкову моделі системи. Модель мабутнього додатка або системи будується для того щоб краще зрозуміти деталі на тонкощі які потрібно врахувати під час розробки.

Отже, для початку необхідно побудувати поведінкову модель системи. Поведінкова модель відображає компоненти системи, які залежать від часу і передають динамічні концепції системи та їх співвідношення один з одним.

Головної перевагою використання Unified Modeling Language полягає в тому, що всі члени проєкту після короткого введення можуть розуміти як система працює у загальному.

Також, для розробників має велике значення діаграми класів. Діаграма класів - це тип діаграми статичної структури, яка описує структуру системи,

показуючи класи системи, їхні атрибути, операції (або методи) та зв’язки між об’єктами.

Діаграми класів використовуються для:

– статичний опис вигляду системи/додатку;

– відображення звя’зків між об’єктами;

– опису функціоналу,що виконує система/додаток;

Отже, щоб мати повне представлення про систему потрібно розробити такі базові сутності як: User, Drive, DriveType, File.

Для сутності User основними полями/аттрибутами є:

– id – унікальний ідентифікатор;

– firstname – ім’я користувача;

– lastname – фамілія користувача;

– email – адрес електронної скриньки;

– files - список файлів даного користувача;

– drives – доступні хмарні сховища для поточного користувача;

– authorities – ролі користувача;

– isEnabled – обліковий запис увімкнений;

– isAccountNonLocked – блокування облікового запису;

– isAccountNonExpired – дійсність облікового запису;

– isCredentialsNonExpired – креншели дійсні.

Діаграма класу User наведена нижче на рисунку 3.4.

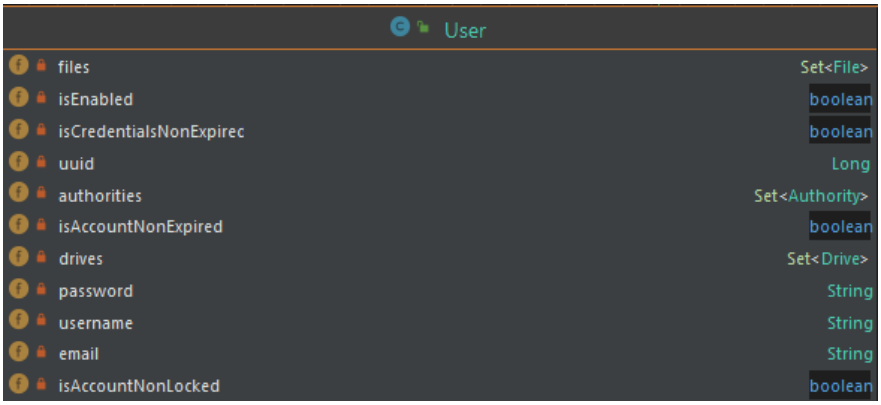

Рисунок 3.4 – Діаграма класу User

Також, не менш важливою сутністю є File, який містить такі поля:

– id – унікальний ідентифікатор;

– userUuid – унікальний ідентифікатор користувача, що є власником;

– name – ім’я файлу;

– fileType – тип файлу(FILE/FOLDER)

Діаграма класу File наведена нижче на рисунку 3.5.

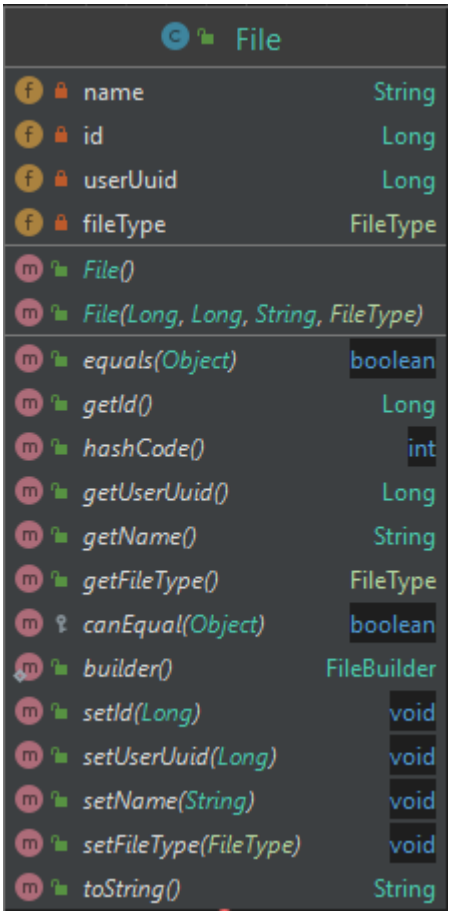

Рисунок 3.5 – Діаграма класу File

Отже, як видно з діаграм класів User та File мають тісний зв’язок один між одним, тобто якщо простими словами, то користувач має певний список файлів, які він завантажив у хмарні сховища.

3.4 Розробка алгоритмів функціонування модулів інформаційної технології організації процесу тестування знань

Алгоритм – це поетапний опис певного процесу, який обов’язково має

початок та кінець. Якщо алгоритм не має закінчення, тоді при розробці щось пішло не так. З допомогою алгоритму роботи системи/веб-додатка полегшується сам етап розробки а саме розробка логіки у певному модулі.

Алгоритм можна описати такими способами:

– лінгвістичний;

– схематичний.

Лінгвістичний спосіб опису передбачає опис алгоритму словами. Великим мінусом такого підходу є те, що все було сказано має властивість забуватись, а отже програмісту потрібно знову уточнювати все.

Схематичний спосіб опису передбачає опис алгоритму за допомогою структурних блоків. Великим плюсом, є те, що потрібно всього лише один раз розробити схему та видати її програмісту.

Розробимо алгоритм розподілу даних між хмарними сховищами:

1. початок;
2. завантаження файлу;
3. перевірка доступних файлових сховищ;
4. відправка файлів на доступні файлові сховища.
5. отримання результату відправки.
6. кінець.

Щоб зрозуміти алгоритм більш детальніше потрібно показати його графічно. Схема вивантаження файлу у хмарні сховища наведена на рис. 3.6.

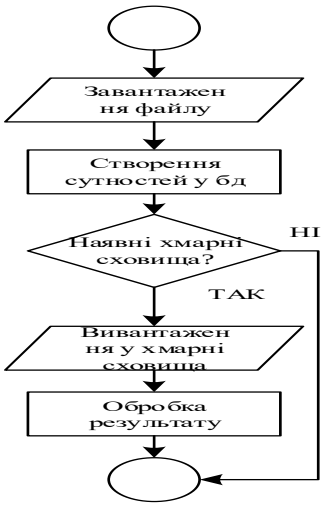

Рисунок 3.6 – Схема алгоритму вивантаження файлів у хмарні сховища

Схема алгоритму пошуку у хмарних сховищах наведена на рисунку 3.7.

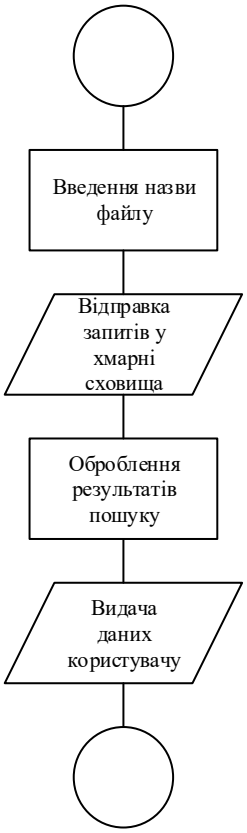

Рисунок 3.7 – Схема алгоритму пошуку у хмарних сховищах.

Отже, таким чином були розроблені алгоритми та схеми для які є необхідними по вимогам функціоналу системи/додатка.

2.5 Розробка архітектури бази даних

Важливим етапом у розробці системи/веб-додатка є розробка архітектури бази даних, оскільки при неправильній архітектурі будуть зайві дані у таблиці, оскільки вони не будуть мати кореляції з зберігаємими записами.

Щоб зменшити імовірність побудови некоректної архітектури бази даних потрібно дотримуватись хоча б трьох правил нормалізації.

Отже, розробимо архітектуру бази даних системи для таких сутностей як:

- user;
- file;

- drive

Ці три сутності є основними складовими логіки та процесів системи, які необхідно описати.

Інформація про користувачів:

- унікальний ідентифікатор користувача(Id);
- ім’я користувача(username);
- адреса електронної скриньки;
- прапорець чи ввімкнений обліковий запис(is_enabled);
- прапорець заблокованого облікового запису;
- прапорець дійсності облікового запису;
- прапорець дійсності повноважень.

Інформація про файли:

- унікальний ідентифікатор(id);
- унікальний ідентифікатор користувача(uuid);
- ім’я файлу(name);
- тип файлу(file_type);

Інформація про ролі користувачів:

- унікальний ідентифікатор(id);
- унікальний ідентифікатор користувача(uuid);
- типі ролі(authority_type).

Інформація про хмарні сховища:

- унікальний ідентифікатор(id);
- унікальний ідентифікатор користувача(uuid);
- прапорець обраного хмарного сховища(is_selected);
- тип хмарного сховища(drive_type).

Усі атрибути сутностей наведені у таблиці 2.1.

Таблиця **2.1** – Перелік атрибутів сутностей

№	Назва атрибуту	Ідентифікатор атрибуту	Опис атрибуту
1	id – користувача	id	Унікальний ідентифікатор користувача

Продовження таблиці 2.1

№	Назва атрибуту	Ідентифікатор атрибуту	Опис атрибуту
2	username – користувача	username	Унікальне ім’я користувача
3	email – користувача	email	Унікальна адреса електронної скриньки користувача
4	is_enabled – користувача	is_enabled	Прапорець активності облікового запису
5	is_account_non_expired – користувача	is_account_non_expired	Прапорець дійсності облікового запису
6	is_account_non_locked-користувача	is_account_non_locked	Прапорець заблокованого облікового запису
7	is_creadentials_non_expired – користувача	is_creadentials_non_expired	Прпорець дійсності повноважень
8	id – файлу	id	Унікальний ідентифікатор файлу
9	uuid – файлу	uuid	Зовнішній ключ, який зв’язує файл з користувачем
10	name – файлу	name	Ім’я файлу
11	file_type – файлу	file_type	Тип файлу
12	id – ролі користувача	id	Унікальний ідентифікатор
№	назва атрибуту	ідентифікатор атрибуту	Опис атрибуту
13	uuid – ролі користувача	uuid	Завнішній ключ, який звя’зує користувача з роллю

Продовження таблиці 2.1

№	Назва атрибуту	Ідентифікатор атрибуту	Опис атрибуту
14	authority_type – ролі користувача	authority_type	Тип полі
15	id – сховища	id	Унікальний ідентифікатор
16	uuid - сховища	uuid	Зовнішній ключ, який зв’язує сховище з користувачем
17	is_selected – сховища	is_selected	Прапорець що відображає обране сховище
18	drive_type – сховища	drive_type	Тип сховища

R: (id-user, username-user, email-user, is_enabled-user, is_account_non_expired, is_account_non_locked, is_credentials_non_expired, id-file, uuid-file, name-file, filte_type-file, id-authority, uuid-authority, uuid-authority, authority_type-authority, id-drive, uuid-drive, is_selected-drive, drive_type – drive).

Отже, потужність універсального сета – 19. Для більш детального відображення зробимо ER-діаграму, яка зображена на рисунку 3.8.

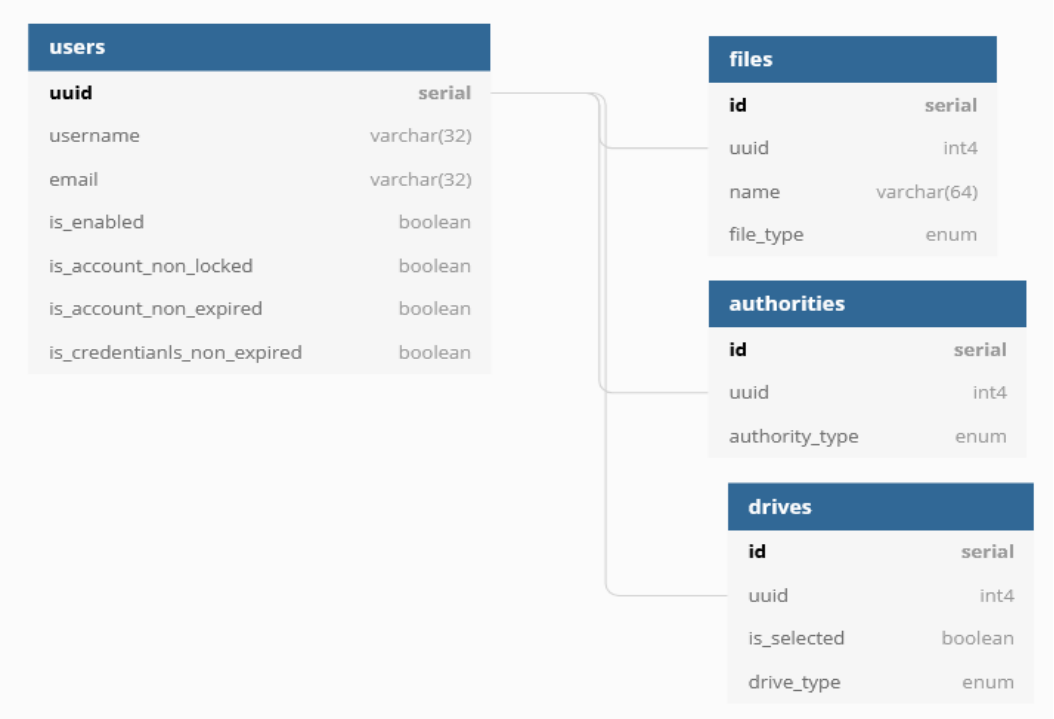

Рисунок 3.8 - ER-діаграма зв’язків

ER-діаграма, це різновид блок-схем, де показано, як різні «сутності» пов'язані між собою всередині системи. ER-схеми частіше всього застосовуються для проектування та підтримки реляційних баз даних у сфері освіти, дослідження та розробки програмного забезпечення та інформаційних систем для бізнесу. ER-діаграми (або ER-моделі) покладаються на стандартний набір символів, включаючи прямокутники, ромби, овали та сполучні лінії, для відображення сутностей, їх атрибутів і зв'язків. Ці діаграми устроені по тому же принципу, що і граматичні структури: сутності виконують роль істотних, а зв'язок — голову.

ER-діаграми - "родичі" схем структури даних (DSD), де замість зв'язків між самими сутностями відображаються відносини між елементами всередині них. ER-діаграми часто використовуються у поєднанні з діаграмами DFD, що схематично показують рух потоків інформації в рамках процесу або системи.

Визначаючи сутності та їх атрибути а також показуючи зв’язки між ними, ЕР-діаграма ілюструє логічну структуру баз даних. ER-діаграми надзвичайно корисні при проектуванні та при створенні архітектури баз даних, оскільки за допомогою них можна продумати до найменших деталей сутність, яка буде відображати описуваний об’єкт без реального створення таблиць. Також, допоможе визначити поля для яких потрібно створити індекси, щоб пришвидшити пошук та вибірку з таблиць.

3.5 Висновки

У цьому розділі було зроблено аналіз архітектурного підходу до побудови веб-додатку, розроблено загальну структурну схему веб-додатку, спроектовано компоненти веб-додатку, а також розроблено алгоритми функціонування веб-додатку розподілу даних між хмарними сховищами.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ РОЗПОДІЛУ ДАНИХ МІЖ ХМАРНИМИ СХОВИЩАМИ

4.1 Обґрунтування вибору мови програмування для серверної частини

Перед початком самої розробки веб-додатка потрібно обрати оптимальну мову програмування. Мова програмування - це інструмент який призначений для вирішення тих чи інших проблем. Оскільки система буде складатись з двох частин, а саме клієнтської та серверної необхідно обрати дві мови які будуть найбільше підходити для вирішення цих задач.

Серед найпопулярніших мов програмування для розробки серверної частини є Java, C#, Node JS, GO Lang, Scala.

Мова програмування Java була розроблена компанією Sun Microsystems на початку 1990-х років. Хоча вона в основному використовується для Інтернет-додатків, Java є простою, ефективною мовою загального призначення[8]. Спочатку Java була розроблена для вбудованих мережевих програм, що працюють на кількох платформах. Це портативна, об'єктно-орієнтована, інтерпретована мова[8]..

Java надзвичайно портативна. Одна й та сама програма Java працюватиме однаково на будь-якому комп’ютері, незалежно від апаратних можливостей чи операційної системи, якщо в ній є інтерпретатор Java. Крім портативності, ще однією з ключових переваг Java є набір функцій безпеки, які захищають ПК, на якому запущена програма Java, не тільки від проблем, викликаних помилковим кодом, але й від шкідливих програм (наприклад, вірусів) [8]. Ви можете безпечно запускати аплет Java, завантажений з Інтернету, оскільки функції безпеки Java перешкоджають цим типам аплетів отримати доступ до жорсткого диска ПК або мережевих з’єднань. Аплет, як правило, є невеликою програмою на Java, яка вбудована в сторінку HTML[8].

Java можна вважати як компільованою, так і інтерпретованою мовою, оскільки її вихідний код спочатку компілюється у двійковий байт-код. Цей байт-

код працює на віртуальній машині Java (JVM), яка зазвичай є програмним інтерпретатором[8]. Використання скомпільованого байт-коду дозволяє інтерпретатору (віртуальній машині) бути невеликим і ефективним (і майже таким же швидким, як і процесор, який виконує рідний скомпільований код).

Крім того, цей байт-код надає Java портативності: він працюватиме на будь-якій JVM, яка правильно реалізована, незалежно від комп'ютерної апаратної чи програмної конфігурації. Більшість веб-браузерів (наприклад, Microsoft Internet Explorer або Netscape Communicator) містять JVM для запуску Java-аплетів[8].

У порівнянні з C++ (іншою об'єктно-орієнтованою мовою), код Java працює трохи повільніше (через JVM), але він більш переносимий і має набагато кращі функції безпеки. Віртуальна машина забезпечує ізоляцію між ненадійною програмою Java та ПК, на якому запущено програмне забезпечення. Синтаксис Java подібний до C++, але мови зовсім інші[8].

Наприклад, Java не дозволяє програмістам реалізувати перевантаження операторів, тоді як C++ це робить. Крім того, Java є динамічною мовою, де ви можете безпечно змінювати програму під час її виконання, тоді як C++ не дозволяє цього. Це особливо важливо для мережевих програм, які не можуть дозволити собі простої. Крім того, всі основні типи даних Java є попередньо визначеними і не залежать від платформи, тоді як деякі типи даних можуть змінюватися в залежності від платформи, що використовується в C або C++ (наприклад, тип int) [8].

Програми на Java є більш високоструктурованими, ніж еквіваленти C++. Усі функції (або методи Java) і виконувані оператори в Java повинні знаходитися в межах класу, тоді як C++ дозволяє визначенням функцій і рядкам коду існувати за межами класів (як у програмах у стилі C) [8]. Глобальні дані та методи не можуть перебувати за межами класу в Java, тоді як C++ дозволяє це. Ці обмеження, хоча часом і громіздкі, допомагають підтримувати цілісність і безпеку програм Java і змушують їх бути повністю об'єктно-орієнтованими[8].

Ще однією ключовою особливістю Java є те, що це відкритий стандарт із загальнодоступним вихідним кодом. Sun Microsystems контролює мову Java та пов'язані з нею продукти, але ліберальна політика ліцензування Sun сприяла тому, що Інтернет-спільнота прийняла Java як стандарт.

C# — сучасна, об'єктно-орієнтована та безпечна для типів мова програмування. C# дозволяє розробникам створювати багато типів безпечних і надійних програм, які працюють у .NET. C# має свої коріння в сімействі мов C і буде відразу знайомий програмістам C, C++, Java та JavaScript[9].

C# - це об'єктно-орієнтована, компонентно-орієнтована мова програмування. C# надає мовні конструкції для безпосередньої підтримки цих концепцій, роблячи C# природною мовою для створення та використання програмних компонентів[9]. З моменту свого виникнення в C# були додані функції для підтримки нових робочих навантажень і нових методів проектування програмного забезпечення. За своєю суттю C# є об'єктно-орієнтованою мовою [9].

Кілька функцій C# допомагають створювати надійні та довговічні програми. Збір сміття автоматично відновлює пам’ять, зайняту недоступними невикористаними об’єктами[9]. Типи, які допускають значення NULL, захищають від змінних, які не посилаються на виділені об’єкти. Обробка винятків забезпечує структурований і розширений підхід до виявлення та відновлення помилок [9]. Лямбда-вирази підтримують методи функціонального програмування.

Синтаксис мовного інтегрованого запиту (LINQ) створює загальний шаблон для роботи з даними з будь-якого джерела. Підтримка мови для асинхронних операцій забезпечує синтаксис для побудови розподілених систем. C# має уніфіковану систему типів [9]. Усі типи C#, включаючи примітивні типи, такі як int і double, успадковуються від одного кореневого типу об’єкта. Усі типи мають набір загальних операцій. Цінності будь-якого типу можна зберігати, транспортувати та використовувати узгоджено [9]. Крім того, C# підтримує як визначені користувачем типи посилань, так і типи значень. C# дозволяє

динамічно розподіляти об'єкти та вбудовано зберігати полегшені структури. C# підтримує загальні методи та типи, які забезпечують підвищену безпеку та продуктивність типів[9]. C# надає ітератори, які дають змогу реалізаторам класів колекції визначати користувацьку поведінку для клієнтського коду.

Node.js — це надзвичайно потужна платформа на основі JavaScript, яка використовується для розробки онлайн-додатків для чату, сайтів для потокового відео, односторінкових програм та багатьох інших додатків [10].

Побудований на движку JavaScript V8 Google Chrome, він використовується як великими, відомими компаніями, так і новоспеченими стартапами (Netflix, Paypal, NASA і Walmart, щоб назвати лише деякі).

Node.js є відкритим вихідним кодом і повністю безкоштовним, яким користуються тисячі розробників по всьому світу[10].

Node.js - це кросплатформне середовище виконання JavaScript з відкритим вихідним кодом і бібліотека для запуску веб-додатків поза браузером клієнта. Райан Даль розробив його у 2009 році, а його остання версія, версія 15.14, була випущена в квітні 2021 року. Розробники використовують Node.js для створення веб-додатків на стороні сервера, і він ідеально підходить для додатків із великим обсягом даних, оскільки використовує асинхронну подію [10].

Go — це мова загального призначення, розроблена з урахуванням системного програмування. Спочатку його розробили в Google у 2007 році Роберт Гріземер, Роб Пайк та Кен Томпсон. Він строго і статично типізований, забезпечує вбудовану підтримку збирання сміття та підтримує одночасне програмування [11].

Програми створюються за допомогою пакетів для ефективного управління залежностями. Реалізації програмування Go використовують традиційну модель компіляції та посилання для створення виконуваних двійкових файлів. Мова програмування Go була анонсована в листопаді 2009 року і використовується в деяких виробничих системах Google [11].

Scala, скорочення від Scalable Language, є гібридною функціональною мовою програмування. Його створив Мартін Одерський. Scala плавно інтегрує

особливості об'єктно-орієнтованих і функціональних мов[12]. Scala скомпільовано для роботи на віртуальній машині Java. Багато існуючих компаній, які залежать від Java для критичних бізнес-додатків, звертаються до Scala, щоб підвищити продуктивність розробки, масштабованість додатків і загальну надійність[12].

Scala - це чиста об'єктно-орієнтована мова в тому сенсі, що кожне значення є об'єктом. Типи та поведінка об’єктів описуються класами та ознаками [12].

Класи розширюються за допомогою підкласів і гнучкого механізму композиції на основі міксину як чиста заміна множинного успадкування.

Scala також є функціональною мовою в тому сенсі, що кожна функція є значенням, а кожне значення є об’єктом, тому в кінцевому підсумку кожна функція є об’єктом [12].

Scala забезпечує легкий синтаксис для визначення анонімних функцій, підтримує функції вищого порядку, дозволяє вкладати функції та підтримує карріювання [12].

На відміну від деяких інших мов зі статичним типом (C, Pascal, Rust тощо), не очікує, що ви надасте зайву інформацію про типи. У більшості випадків не потрібно вказувати тип і, звичайно, не потрібно його повторювати [12].

Отже після аналізу мов програмування для серверної частини було обрано Scala та Java, оскільки вони працюють на одній і ті ж самі віртуальні машині, їх можна використовувати разом [12].

4.2 Обґрунтування вибору мови програмування для клієнтської частини

Користувацький інтерфейс – це важлива частина кожного додатку, адже від цього залежить зручність користування.

Отже, щоб розробити графічний інтерфейс для веб додатка потрібно вибрати відповідну мову програмування.

JavaScript - це динамічна мова комп'ютерного програмування. Він легкий і найчастіше використовується як частина веб-сторінок, чиї реалізації дозволяють клієнтському сценарію взаємодіяти з користувачем і створювати динамічні сторінки[13]. Це інтерпретована мова програмування з об'єктно-орієнтованими можливостями.

Спочатку JavaScript був відомий як LiveScript, але Netscape змінив назву на JavaScript, можливо, через хвилювання, викликане Java[13]. Вперше JavaScript з’явився в Netscape 2.0 у 1995 році під назвою LiveScript. Основне ядро мови загального призначення було вбудовано в Netscape, Internet Explorer та інші веб-браузери[13].

Клієнтський JavaScript є найпоширенішою формою мови. Сценарій повинен бути включений в HTML-документ або посилання на нього, щоб код інтерпретував браузер[13].

Це означає, що веб-сторінка не обов’язково має бути статичним HTML, але може включати програми, які взаємодіють з користувачем, керують браузером і динамічно створюють вміст HTML[13].

Механізм на стороні клієнта JavaScript надає багато переваг перед традиційними серверними сценаріями CGI. Наприклад, ви можете використовувати JavaScript, щоб перевірити, чи ввів користувач дійсну адресу електронної пошти в поле форми[13].

Код JavaScript виконується, коли користувач надсилає форму, і лише якщо всі записи дійсні, вони будуть передані на веб-сервер.

Отже, для реалізації графічного веб-інтерфейсу було обрано JavaScript.

4.3 Обґрунтування вибору середовища розробки

Для комфортної та швидкої розробки додатку потрібно обрати середовище розробки, або другими словами IDEA.

IDE - Інтегроване середовище розробки, це програма, яка надає програмістам комплексні засоби для розробки програмного забезпечення. IDE

зазвичай складається принаймні з редактора вихідного коду, засобів автоматизації збірки та налагоджувача. Деякі IDE, такі як NetBeans і Eclipse, містять необхідний компілятор, інтерпретатор або обидва; інші, такі як SharpDevelop і Lazarus, цього не роблять. Інтегровані середовища розробки розроблені для максимальної продуктивності програміста, забезпечуючи тісно пов’язані компоненти з подібними інтерфейсами користувача. IDE представляють єдину програму, в якій здійснюється вся розробка. Ця програма зазвичай надає багато функцій для створення, модифікації, компіляції, розгортання та налагодження програмного забезпечення. Це контрастує з розробкою програмного забезпечення з використанням непов’язаних інструментів, таких як vi, GDB, GCC або make.

Однією з цілей IDE є скорочення конфігурації, необхідної для об’єднання кількох утиліт розробки, замість цього вона надає той самий набір можливостей, що й один цілісний блок. Зменшення часу встановлення може підвищити продуктивність розробника, особливо в тих випадках, коли навчання використання IDE відбувається швидше, ніж ручна інтеграція та вивчення всіх окремих інструментів. Тісніша інтеграція всіх завдань розробки може підвищити загальну продуктивність, а не лише допомогу з налаштуванням. Наприклад, код можна безперервно аналізувати під час його редагування, забезпечуючи миттєвий зворотний зв’язок при появі синтаксичних помилок, що дозволяє розробникам набагато швидше та легше налагоджувати код за допомогою IDE.

Деякі IDE призначені для певної мови програмування, що дозволяє створити набір функцій, який найбільше відповідає парадигмам програмування цієї мови. Однак існує багато багатомовних IDE.

Хоча більшість сучасних IDE є графічними, текстові IDE, такі як Turbo Pascal, були популярними до появи віконних систем, таких як Microsoft Windows і X Window System (X11). Вони зазвичай використовують функціональні клавіші або гарячі клавіші для виконання часто використовуваних команд або макросів.

Межа між IDE та іншими частинами більш широкого середовища розробки програмного забезпечення не чітко визначена; іноді інтегрована система контролю версій або різні інструменти для спрощення побудови графічного інтерфейсу користувача (GUI). Багато сучасних IDE також мають браузер класів, браузер об’єктів та діаграму ієрархії класів для використання при розробці об’єктно-орієнтованого програмного забезпечення.

Отже, IDEA які можна використати для розробки:

– Eclipse;

– NetBeans;

– Intellij IDEA.

Eclipse - у контексті обчислювальної техніки є інтегрованим середовищем розробки (IDE) для розробки додатків із використанням мови програмування Java та інших мов програмування, таких як C/C++, Python, PERL, Ruby тощо[14].

Платформа Eclipse, яка є основою для Eclipse IDE, складається з плагінів і призначена для розширення за допомогою додаткових плагінів. Розроблена з використанням Java, платформа Eclipse може використовуватися для розробки багатих клієнтських програм, інтегрованих середовищ розробки та інших інструментів[14]. Eclipse можна використовувати як IDE для будь-якої мови програмування, для якої доступний плагін.

Проект Java Development Tools (JDT) надає плагін, який дозволяє використовувати Eclipse як IDE Java, PyDev — це плагін, який дозволяє використовувати Eclipse як IDE Python, C/C++ Development Tools (CDT) — це плагін -in, який дозволяє використовувати Eclipse для розробки додатків на C/C++, плагін Eclipse Scala дозволяє використовувати Eclipse в IDE для розробки додатків Scala, а PHPeclipse – це плагін для eclipse, який надає повний інструмент розробки для PHP[14].

NetBeans IDE — це безкоштовне інтегроване середовище розробки з відкритим кодом (IDE), яке дає змогу розробляти настільні, мобільні та веб-додатки. IDE підтримує розробку додатків різними мовами, включаючи Java, HTML5, PHP і C++ [15]. IDE забезпечує інтегровану підтримку для повного

циклу розробки, від створення проекту до налагодження, профілювання та розгортання. IDE працює на Windows, Linux, Mac OS X та інших системах на базі UNIX [15].

IDE забезпечує повну підтримку технологій JDK 7 та останніх удосконалень Java. Це перша IDE, яка забезпечує підтримку JDK 7, Java EE 7 і JavaFX 2. IDE повністю підтримує Java EE з використанням останніх стандартів для Java, XML, веб-сервісів і SQL і повністю підтримує GlassFish Server, довідкову реалізацію Java EE [15].

IntelliJ IDEA — це інтегроване середовище розробки (IDE) для мов JVM, розроблене для максимальної продуктивності розробників[16]. Він виконує рутинні й повторювані завдання за вас, забезпечуючи розумне завершення коду, статичний аналіз коду та рефакторинг, а також дозволяє зосередитися на яскравій стороні розробки програмного забезпечення, роблячи це не тільки продуктивним, але й приємним.

IntelliJ IDEA випускається в трьох виданнях[16]:

– IntelliJ IDEA Ultimate: комерційне видання для JVM, веб- та корпоративного розвитку. Він включає в себе всі функції видання Community, а також додає підтримку мов, на яких зосереджені інші IDE на платформі IntelliJ, а також підтримку різноманітних серверних і інтерфейсних фреймворків, серверів додатків, інтеграцію з базою даних і профілювання. інструменти та інше;

– IntelliJ IDEA Community Edition: безкоштовна версія на основі відкритого коду для розробки JVM та Android.

– IntelliJ IDEA Edu: безкоштовне видання з вбудованими уроками для вивчення Java, Kotlin і Scala, інтерактивними завданнями програмування та спеціальними можливостями для вчителів для створення власних курсів і керування навчальним процесом (див. IntelliJ IDEA Edu) [16].

Отже, після аналізу IDE враховуючи усі ті чи інші переваги та недоліки вибір упав на Intellij IDEA, оскільки вона зручна, швидка, має безліч функцій, які допомагають пришвидшити розробку.

4.4 Обґрунтування вибору бази даних

На теперішній час є багато варіантів баз даних, які можна використовувати. Бази даних повинні відповідати чотирьом принципам, таким як [17]:

- атомарність;
- консистентність;
- ізольованість;
- надійність.

Атомарність - властивість безперервності операції. Якщо іншими словами, то операція, яка має вкладені операції і вони повинні виконатись успішно щоб зміни були збережені [17].

Консистентність – це узгодженість даних між собою, цілісність даних даних, а також внутріння есуперечність. Тобто, представимо банківську транзакцію на списання певної суми n з рахунку. Якщо в однаковий момент часу транзакцій на списання коштів буде більше чим 1, то вони повинні виконуватись послідовно після [17].

Ізольованість – це рівень на якому відбуваються транзакції. Тобто, кожна транзакція не повинна мати доступ до іншої транзакції[17].

Надійність – це характеристика того, що при будь яких непередбачуваних ситуаціях дані будуть збережені.

Отже, розглянемо дві реляційних бази даних як SQL Server та PostgreSQL.

SQL Server — це механізм даних, представлений Microsoft[18]. Він забезпечує середовище, яке використовується для створення та керування базами даних. Це забезпечує надійне та ефективне зберігання. Він надає інші компоненти та послуги, які підтримують платформу бізнес-аналітики для створення звітів і аналізу даних [18].

Ключові компоненти та послуги SQL Server

Нижче наведено основні компоненти та служби SQL-сервера:

– database engine: цей компонент обробляє зберігання, швидку обробку транзакцій і захист даних[18];

– sql server, ця служба запускає, зупиняє, призупиняє та продовжує роботу екземпляра Microsoft SQL Server. Ім’я виконуваного файлу – sqlservr.exe;

– агент SQL Server: виконує роль планувальника завдань. Він може бути викликаний будь-якою подією або за бажанням. Ім'я виконуваного файлу sqlagent.exe[18];

– браузер SQL Server: це прослуховує вхідний запит і підключається до потрібного екземпляра сервера SQL. Ім’я виконуваного файлу – sqlbrowser.exe[18].

– Повнотекстовий пошук SQL Server: це дозволяє користувачеві виконувати повнотекстові запити щодо даних символів у таблицях SQL. Ім’я виконуваного файлу – fdlauncher.exe[18];

– sql server VSS Writer: дозволяє резервне копіювання та відновлення файлів даних, коли сервер SQL не запущено. Ім’я виконуваного файлу – sqlwriter.exe[18];

PostgreSQL — це потужна об'єктно-реляційна база даних з відкритим вихідним кодом, яка використовує та розширює мову SQL у поєднанні з багатьма функціями, які безпечно зберігають і масштабують найскладніші робочі навантаження даних [19]. Витоки PostgreSQL сягають 1986 року в рамках проекту POSTGRES в Каліфорнійському університеті в Берклі і має понад 30 років активної розробки на базовій платформі [19].

PostgreSQL заслужив міцну репутацію завдяки своїй перевіреній архітектурі, надійності, цілісності даних, надійному набору функцій, розширюваності та відданості спільноти відкритих вихідних кодів, які стоять за програмним забезпеченням, щоб постійно надавати продуктивні та інноваційні рішення[19]. PostgreSQL працює на всіх основних операційних системах, має ACID-сумісність з 2001 року і має потужні надбудови, такі як популярний розширювач геопросторової бази даних PostGIS. Не дивно, що PostgreSQL став

реляційною базою даних з відкритим кодом, яку вибирають багато людей та організацій [19].

Створимо таблицю 4.1 порівняння характеристик баз даних.

Таблиця 4.1 – Порівняння характеристик баз даних.

№	Критерій	PostgreSQL	SQL Server
1	Контроль доступу	+	-
2	Резервне копіювання та відновлення	+	+
3	Міграція даних	+	+
4	Реплікація даних	+	-
5	Перетворення бази даних	+	-
6	Мобільний доступ	-	-
7	Підтримка кількох мов програмування	+	-
8	Аналіз продуктивності	+	+
9	Реляційний	+	+
10	Віртуалізація	+	-
	Висновки	9/10	4/10

Після аналізу баз даних PostgrSQL та SQL Server за десятьма ознаками, які є надзвичайно важливими для розробки додатку було обрано PostgreSQL. Тому що, як видно з таблиці 4.1 PostgreSQL має значні переваги за SQL Server, а саме:

- контроль доступу;
- реплікація даних;
- перетворення бази даних;
- підтримка кількох мов програмування;
- віртуалізація.

4.5 Висновки

У даному розділі було проведено аналіз мови програмування для серверної частини, для клієнтської частини та вибрано найкращу яка підходить для вирішення проблеми постановленої задачі. Обґрунтовано вибір середовища розробки та базу даних.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

5.1 Методи тестування

Тестування неважливо якого додатку є загальним — загальне поняття, яке включає у себе планування, проектування та проходження юніт тестів, які були написані заздалегідь на етапі розробки.

Тестування веб-сайту – це етап тестування під час якого визначається зручність та комфортність користування самим веб-сайтом для користувача, швидкість завантаження веб-сторінок. Тестування веб-сайтів – це трудомісткий процес, який відбувається уже на фінальному етапі, тобто коли веб-сайт уже розроблений повністю [20].

WEB-тестування – це перевірка веб-додатку на наявність помилок, які були допущені на етапі розробки. WEB-тестування перевіряє функціонал, зручність використання, безпеку, сумісність, продуктивність веб-програми або веб-сайту[20].

У інженерії програмного забезпечення можна виділити настпні методи тестування, в залежності від вимог до веб-тестування [20]:

- тестування функціональності веб-сайту;
- тестування на зручність використання;
- тестування інтерфейсу;
- тестування бази даних;
- тестування на сумісність;
- тестування продуктивності;
- тестування безпеки.

Тестування функціональності веб-сайту – це поетапний процес, який включає в себе кілька параметрів тестування, таких як користувацький інтерфейс, API тестування, тестування бази даних, звісно, якщо вона є, тестування захищеності веб-додатка, клієнт/серверне тестування[20].

Функціональне тестування дуже зручне, оскільки, воно дозволяє користувачам виконувати manual, так і automationтестування [20].

Веб-тестування включає в себе чи всі посилання на веб-сторінках, перевіряє їх на коректність [20].

Посилання, які необхідно перевірити:

– вихідні посилання;
– внутрішні посилання;
– якірні посилання.

Перевірки валідації на формах повинні працювати відповідним чином, як те вимагає бізнес логіка. Наприклад, якщо якийсь користувач не заповнює усі обов’язкові поля, то з’явиться повідомлення з помилкою валідації[20].

Тестові файли cookie працюють належним чином. Файли cookie – це невеликі файли, які використовуються веб-сайтами для запам’ятовування активних сеансів користувача, тому вам не потрібно входити в систему щоразу, коли ви відвідуєте веб-сайт [20]. Тестування файлів cookie включатиме

Тестові файли cookie (сеанси) видаляються або після очищення кешу, або після закінчення терміну їх дії.

Видаліть файли cookie (сесії) і перевірте, чи запитуються облікові дані для входу, коли ви наступного разу відвідуєте сайт.

Тестування робочого процесу включатиме в себе тестування вашого робочого процесу/бізнес-сценарії від кінця до кінця, яке веде користувача через серію веб-сторінок [20].

Також тестуйте негативні сценарії, щоб, коли користувач виконує несподіваний крок, у вашій веб-програмі відображалося відповідне повідомлення про помилку або довідка.

Тестування юзабіліті тепер стало важливою частиною будь-якого веб-проекту. Перевірка навігації по сайту: меню, кнопки або посилання на різні сторінки вашого сайту мають бути легко видимими й узгодженими на всіх веб-сторінках.

Застосування, тестові запити правильно надсилаються в базу даних, а вихідні дані на стороні клієнта відображаються правильно. Помилки, якщо

такі є, повинні бути перехоплені програмою та повинні бути показані лише адміністратору, а не кінцевому користувачеві[20].

Тестовий веб-сервер обробляє всі запити додатків без відмови в обслуговуванні [20].

Переконайтеся, що запити, надіслані до бази даних, дають очікувані результати.

Перевірте відповідь системи, коли не вдається встановити з'єднання між трьома рівнями (додаток, веб і база даних), і кінцевому користувачеві відображається відповідне повідомлення.

База даних є одним із найважливіших компонентів вашого веб-додатка, і необхідно приділити особливу увагу, щоб ретельно перевірити його. Перевірте, чи не відображаються помилки під час виконання запитів. Цілісність даних підтримується під час створення, оновлення або видалення даних у базі даних. Перевірте час відповіді на запити та налаштуйте його, якщо необхідно [20]. Тестові дані, отримані з вашої бази даних, точно відображаються у вашому веб-додатку.

Тест на сумісність гарантує, що ваш веб-додаток правильно відображається на різних пристроях. Це включатиме - тест на сумісність браузера: один і той самий веб-сайт у різних браузерах відображатиметься по-різному. Вам потрібно перевірити, чи правильно відображається ваш веб-додаток у браузерах, JavaScript, AJAX та аутентифікація працюють нормально. Ви також можете перевірити сумісність мобільного браузера.

Відображення веб-елементів, таких як кнопки, текстові поля тощо, змінюється зі зміною операційної системи. Переконайтеся, що ваш веб-сайт добре працює для різних комбінацій операційних систем, таких як Windows, Linux, Mac і браузерів, таких як Firefox, Internet Explorer, Safari тощо [20].

Це забезпечить роботу вашого сайту при будь-яких навантаженнях. Діяльність з тестування програмного забезпечення включатиме, але не обмежуючись. Час відповіді веб-сайту на різних швидкостях підключення.

Перевірте навантаження веб-програми, щоб визначити її поведінку при звичайних і пікових навантаженнях. Стрес-тестування вашого веб-сайту, щоб визначити його точку зупинки, коли він перевищив нормальне навантаження в час пік. Перевірте, чи відбувається збій через пікове навантаження, як сайт відновлюється після такої події. Переконайтеся, що такі методи оптимізації, як стиснення zip, кеш браузера та сервера, увімкнено, щоб скоротити час завантаження [20].

Тестування безпеки є життєво важливим для веб-сайтів електронної комерції, які зберігають конфіденційну інформацію про клієнтів. Не слід дозволяти тестовий несанкціонований доступ до захищених сторінок. Обмежені файли не можна завантажувати без відповідного доступу. Сеанси перевірки автоматично припиняються після тривалої бездіяльності користувача. При використанні сертифікатів SSL веб-сайт має перенаправляти на зашифровані сторінки SSL [20].

5.2 Тестування веб-додатка

Тестування було проведено на двох хмарних сховищах, таких як Google Drive і One Drive. Оскільки це веб додаток, то і використання буде через браузер. На рисунку 5.1 наведено графічний інтерфейс додатку.

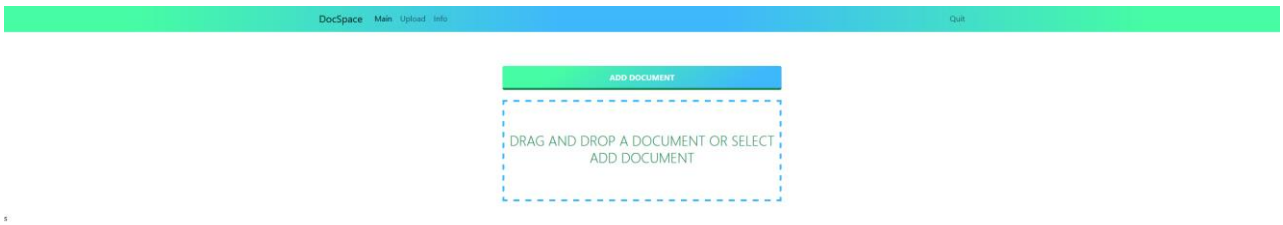

Рисунок 5.1 – Вигляд графічного графічного інтерфейсу

Щоб протестувати веб-додаток, потрібно завантажити файл у компонент file uploder, це можливо зробити двома шляхами, перший, це клікнути на поле

“Drag and Drop” і вибрати файл, а другий перенести необхідний файл у область file uploader який потрібно розподілити між двома сховищами.

На рисунку 5.2 наведено вікно вибору файлу.

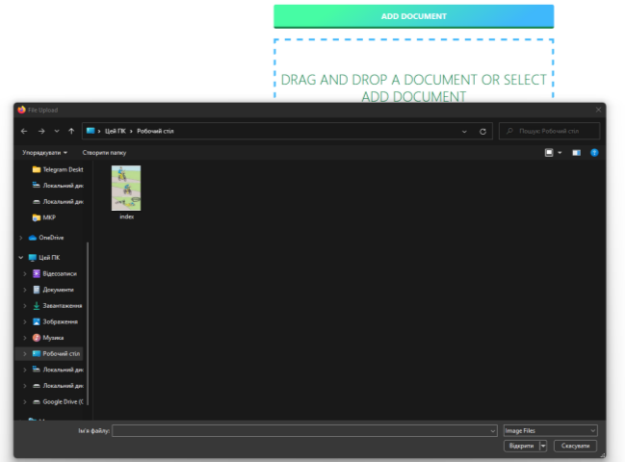

Рисунок 5.2 – Вигляд вікна вибору файлу

Отже, щоб протестувати, обираємо файл, та чекаємо відповідний час, поки файл завантажиться у хмарні сховища.

Для перевірки того, чи файл був завантажений, потрібно зайти у сховища та переконатись що файл присутній як і в Google Drive так і в One Drive хмарному сховищі. На рисунку 5.3 наведено результат завантаження файлу у One Drive.

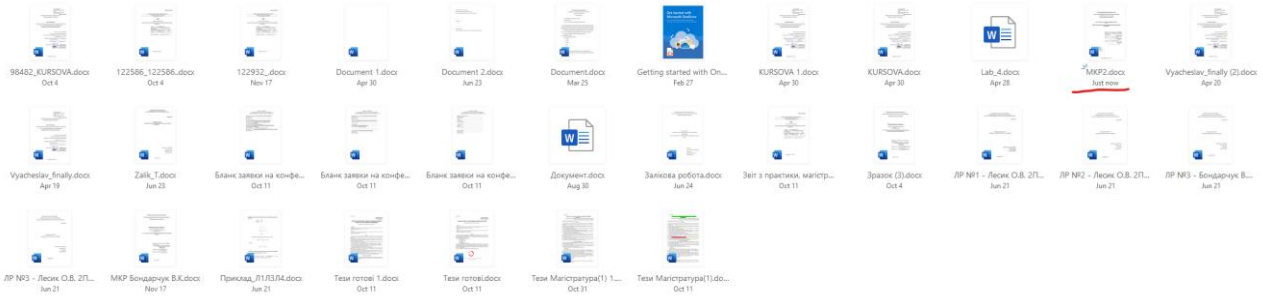

Рисунок 5.3 – Результат завантаження файлу у хмарне сховище One Drive

Після того як завантажили файл у file uploader, файл повинен передатись на бекенд, а після цього, уже розприділитись між двома файловими сховищами

На рисунку 5.4 наведено результат завантаження у хмарне сховище Google Drive.

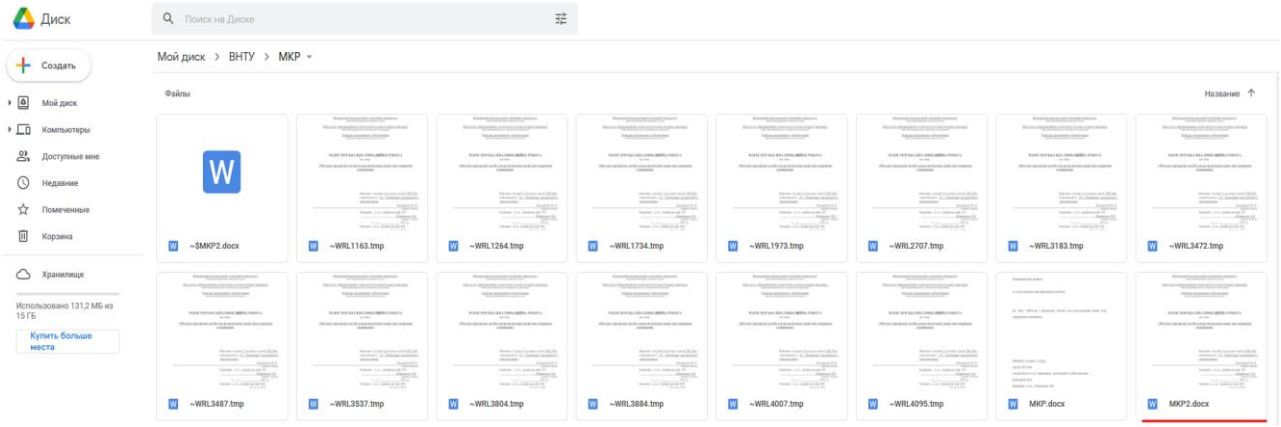

Рисунок 5.4 – Результат завантаження файлу у хмарне сховище Google Drive

Отже, по результатам тестування можна сказати, що додаток працює коректно, згідно до поставлених вимог.

5.3 Висновки

Було розглянуто, що таке тестування, як воно відбувається та від яких речей залежить. Також, необхідними умовами для тестування є наявність: об’єкту дослідження та наглядача.

Виконано тестування веб-додатку. Таким чином мета магістерської роботи досягнена.

6 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота за темою «Методи та засоби розподілення даних між хмарними сховищами» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1) проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

6.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Методи та засоби розподілення даних між хмарними сховищами» є

оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 6.1 **[**21**]**.

Таблиця 6.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження табл. 6.1- Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуютьсяу військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці.

Таблиця 6.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	5	5	5
2. Ринкові переваги (наявність аналогів)	4	4	4
3. Ринкові переваги (ціна продукту)	3	3	3
4. Ринкові переваги (технічні властивості)	4	4	4
5. Ринкові переваги (експлуатаційні витрати)	3	3	3
6. Ринкові перспективи (розмір ринку)	4	4	3
7. Ринкові перспективи (конкуренція)	3	3	4
8. Практична здійсненність (наявність фахівців)	3	3	3
9. Практична здійсненність (наявність фінансів)	3	4	4
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	4	4	3
Сума балів	44	45	44
Середньоарифметична сума балів $СБ_с$	44,3		

За результатами розрахунків, наведених в таблиці 4.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 6.3 [21].

Таблиця 6.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Методи та засоби розподілення даних між хмарними сховищами» становить 44,3 бала, що, відповідно до таблиці 6.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

6.2 Оцінювання рівня новизни розробки

Виводячи на ринок новинку виробник вважає, що тієї новизни, якою наділена нова розробка є достатньо для того, щоб вона була сприйнята споживачем як нова. Але це не завжди так, в силу того, що споживач і виробник неоднозначно визначать її рівень новизни. Тому доцільним є визначення рівня новизни розробки отриманої в результаті досліджень за темою «Методи та засоби розподілення даних між хмарними сховищами».

Саме визначення рівня і ступеня інтегральної новизни є найбільш актуальним, оскільки її рівень визначає ступінь однакового позитивного сприйняття новизни розробки як виробником, так і споживачем, а отже і ринком в цілому, а це, у свою чергу, є гарантією того, що новинка знайде своє місце на ринку, користуватиметься попитом у споживачів і забезпечить відшкодування витрат, зазнаних товаровиробником під час розроблення та виробництва технічної розробки **[**21**].**

Рівень новизни нової продукції розраховуємо експертним методом шляхом протиставлення нової продукції та її аналогів, що існують в даний час на ринку, за чинниками що визначають її значення, в системі «краще-гірше». Рівень новизни встановлюємо відносно рівня аналога (або продукту, що досить близький до аналога).

Для визначення *i*-го виду новизни, застосуємо чинники, які впливають на її рівень. Кожен чинник *i*-го виду новизни розраховуємо в балах. Більша кількість набраних балів свідчить про більший рівень новизни. Для оцінювання рівня новизни використаємо думки експертів, які встановлюють визначені бали відповідним чинникам. Бал відповідності проставляється в діапазоні від (-5 – значно гірше аналога до +5 – значно краще аналога). Результати попереднього оцінювання зведемо до відповідного листа оцінювання (таблиця 6.4).

Таблиця 6.4 – Лист оцінювання рівня новизни експертами

Види та чинники		Бали та експерти		
		Експерт 1	Експерт 2	Експерт 3
1		*2*	*3*	*4*
Споживча новизна	Питома вага 0,225	Максимальний бал $Б_{i\ MAX}$		25
1. Зміна поведінкових звичок споживача		4	5	4
2. Ступінь задоволення потреб і запитів		5	4	4
3. Спосіб задоволення потреби		3	3	4
4. Формування нової потреби		1	1	2
5. Формування нового споживача		0	0	0
Середній бал експертів $Б_{i\ отр}$		13		
Товарна новизна	Питома вага 0,217	Максимальний бал $Б_{i\ MAX}$		30
1. Параметричні зміни показників продукції				
1.1. Якісні		3	4	3
1.2. Технічні		4	4	3
1.3. Економічні		3	3	3
1.4. Сервісні		4	4	4
2. Якість продукції по відношенню до конкурентів		3	3	3
3. Функціональні зміни		3	3	3
Середній бал експертів $Б_{i\ отр}$		20		
Виробнича новизна	Питома вага 0,042	Максимальний бал $Б_{i\ MAX}$		25
1. Рівень унікальності товару для підприємства		5	5	5
2. Рівень унікальності для галузі		3	4	3
3. Рівень унікальності товару для країни		1	1	1
4. Зміна виробничої системи		4	4	4
5. Відносно існуючого асортименту		2	2	2
Середній бал експертів $Б_{i\ отр}$		15		
Прогресивна новизна	Питома вага 0,179	Максимальний бал $Б_{i\ MAX}$		25
1. Зміна технології виготовлення		4	4	4
2. Рівень застосування нових компонентів і матеріалів		1	2	1
3. Зміна технологічного принципу дії виробу		1	2	1
4. Зміна конструктивного виконання		3	2	3
5. Рівень застосування інновацій		2	2	2
Середній бал експертів $Б_{i\ отр}$		11		
Ринкова новизна	Питома вага 0,12	Максимальний бал $Б_{i\ MAX}$		20
1. Новий виріб на новому ринку		0	0	0

Продовження таблиці 6.4 – Лист оцінювання рівня новизни експертами

2. Новий виріб на відомому ринку		2	2	2
3. Модернізований виріб		2	2	2
4. Нова модель		1	2	2
Середній бал експертів $Б_{i\ отр}$		6		
Екологічна новизна	Питома вага 0,035	Максимальний бал $Б_{i\ MAX}$		20
1. Рівень екологічної чистоти технології виробництва		5	5	5
2. Рівень впровадження мало- та безвідходних технологій		5	5	5
3. Рівень екологічно небезпечних режимів експлуатації продукції		5	5	5
4. Рівень забруднення навколишнього середовища		5	5	5
Середній бал експертів $Б_{i\ отр}$		20		
Соціальна новизна	Питома вага 0,036	Максимальний бал $Б_{i\ MAX}$		20
1. Використання нового товару приводить до покращення стану здоров'я нації		0	0	0
2. Використання нового товару приводить до зростання доходів населення		0	0	0
3. Виробництво нового товару приводить до збільшення (зменшення) кількості робочих місць на підприємстві		4	5	4
4. Виробництво нового товару приводить до підвищення кваліфікації персоналу		3	3	3
Середній бал експертів $Б_{i\ отр}$		7		
Маркетингова новизна	Питома вага 0,146	Максимальний бал $Б_{i\ MAX}$		20
1. Нові методи маркетингових досліджень		0	0	0
2. Вживання нових стратегій сегментації ринку		3	3	3
3. Вибір нової маркетингової стратегії обхвату і розвитку цільового сегмента		2	3	2
4. Побудова нових каналів збуту		2	2	2
Середній бал експертів $Б_{i\ отр}$		7		

Значення *i*-го виду новизни розрахуємо за формулою **[**22**]**:

$$I_i = \frac{Б_{i\,отр}}{Б_{i\,MAX}}, \qquad (6.1)$$

де $Б_{i\ отр}$ – отримана кількість балів за шкалою оцінок чинників, що визначають *i*-й вид новизни;

$Б_{i\ MAX}$ – максимальна кількість балів, що може бути отримана за *i*-м видом новизни.

Загальний рівень інтегральної новизни розраховуємо шляхом перемноження отриманого значення *i*-го виду новизни на її вагомість, причому вагомість *i*-го виду новизни визначаємо експертним методом, за формулою **[**22**]**:

$$N_{інт} = \sum_{i}^{n} W_i \cdot I_i \quad , \tag{6.2}$$

де $N_{інт}$ – рівень інтегральної (сукупної) новизни;

W_i – вагомість (питома вага) *i*-го виду новизни;

n – загальна кількість видів новизни.

$$N_{інт} = (0{,}225 \cdot 13/25)+(0{,}217 \cdot 20/30)+(0{,}042 \cdot 15/25)+(0{,}179 \cdot 11/25)+(0{,}12 \cdot 6/20)+$$
$$(0{,}035 \cdot 20/20)+(0{,}036 \cdot 7/20)+(0{,}146 \cdot 7/20) = 0{,}507.$$

Отримане значення інтегрального рівня новизни зіставляємо зі шкалою, що наведена в табл. 6.5 [21]

Таблиця 6.5 – Рівні новизни нового товару та їхня характеристика

Рівні новизни товару	Значення інтегральної новизни	Характеристика товару	Вид нового товару
Найвища	1,00	Абсолютно новий товар	Новий товар, що наділений ознаками інноваційності (інноваційний товар)
Висока	0,8…0,99	Товар, який не має аналогів	
Значуща	0,6…0,79	Принципова зміна споживчих властивостей товару	
Достатня	0,4…0,59	Принципова технологічна модифікація товару	
Незначна	0,2…0,39	Кардинальна зміна параметрів	Новий товар
Помилкова	0,00…0,19	Малоістотна модифікація	

Згідно таблиці 6.5 розробка відповідає рівню при значенні інтегральної новизни 0,507 - достатня новизна; за характеристикою: принципова технологічна модифікація товару; вид розробки - новий товар, що наділений ознаками інноваційності (інноваційний товар).

6.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Методи та засоби розподілення даних між хмарними сховищами», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

6.3.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою **[**21**]**:

$$З_о = \sum_{i=1}^{k} \frac{М_{пі} \cdot t_і}{Т_р}, \qquad (6.3)$$

де k – кількість посад дослідників залучених до процесу досліджень;

$М_{пі}$ – місячний посадовий оклад конкретного дослідника, грн;

$t_і$ – число днів роботи конкретного дослідника, дн.;

$Т_р$ – середнє число робочих днів в місяці, $Т_р$=24 дні.

$$З_о = 13500{,}00 \cdot 32 / 24 = 18000{,}00 \ грн.$$

Проведені розрахунки зведемо до таблиці 6.6.

Таблиця 6.6 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	13500,00	562,50	32	18000,00
Інженер-розробник програмного забезпечення	12200,00	508,33	18	9150,00
Науковий співробітник дослідження проблем програмного забезпечення	12500,00	520,83	12	6250,00
Технік	7100,00	295,83	5	1479,17
Всього				34879,17

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Методи та засоби розподілення даних між хмарними сховищами» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i \quad , \tag{6.4}$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} \quad , \tag{6.5}$$

де $М_М$ – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=2379,00 грн;

$К_і$ – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [21];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 24 дн;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 2379{,}00 \cdot 1{,}10 \cdot 1{,}65 / (24 \cdot 8) = 22{,}49$ *грн.*

$З_{р1} = 22{,}49 \cdot 6{,}00 = 134{,}93$ *грн.*

Таблиця 6.7 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Установка електронно-обчислювального обладнання	6,00	2	1,10	22,49	134,93
Підготовка робочого місця дослідника	2,40	2	1,10	22,49	53,97
Інсталяція програмного забезпечення	2,20	5	1,70	34,76	76,46
Формування дослідної бази даних інформації хмарного сховища	12,00	2	1,10	22,49	269,87
Налагодження програмних блоків	5,60	5	1,70	34,76	194,63
Монтаж серверного обладнання	10,00	5	1,70	34,76	347,56
Монтаж блоків пам'яті	3,00	4	1,50	30,67	92,00
Тестування системи	8,00	2	1,10	22,49	179,91
Всього					1349,34

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \qquad (6.6)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 12%.

$З_{дод}$ = (34879,17 + 1349,34) · 12 / 100% = 4347,42 грн.

6.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (6.7)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (34879,17 + 1349,34 + 4347,42) · 22 / 100% = 8926,70 грн.

6.3.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою «Методи та засоби розподілення даних між хмарними сховищами».

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (6.8)$$

де $Н_j$ – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 ... 1,15*);

$В_j$ – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$M_1 = 3,00 \cdot 111,00 \cdot 1,1 - 0,000 \cdot 0,00 = 366,30$ грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 6.8 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір Calipso Plus A4-500-80	111,00	3,00	0,000	0,00	366,30
Папір для записів Calipso Parers Light A5	73,00	1,00	0,000	0,00	80,30
Органайзер офісний Calipso Office	210,00	3,00	0,000	0,00	693,00
Канцелярське приладдя (набір офісного працівника)	175,00	5,00	0,000	0,00	962,50
Картридж для принтера Canon LBP6500	875,00	1,00	0,000	0,00	962,50
Диск оптичний NewLine CD-RW	12,10	3,00	0,000	0,00	39,93
Flesh-пам'ять Kingston 16 GB	105,00	1,00	0,000	0,00	115,50
Тека для паперів CALIPSO BOX	82,00	3,00	0,000	0,00	270,60
Всього					3490,63

6.3.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі $(К_в)$, які використовують при проведенні НДР на тему «Методи та засоби розподілення даних між хмарними сховищами» відсутні.

6.3.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного

для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \quad , \tag{6.9}$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань устаткування.

$B_{спец} = 29870,00 \cdot 1 \cdot 1,1 = 32857,00$ *грн.*

Отримані результати зведемо до таблиці:

Таблиця 6.9 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Серверне обладнання на основі ПК ZEVS PC 13430U i5 9400F + GTX 1060 3GB	1	29870,00	32857,00
Засоби передачі даних	1	6230,00	6853,00
Пам'ять (SSD диск) Samsung 870 QVO 1TB 2.5" V-NAND 4bit MLC (QLC) SATA III (MZ-77Q1T0BW)	2	3000,00	6600,00
Всього			46310,00

6.3.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \quad , \qquad (6.10)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

$B_{прг} = 10250,00 \cdot 1 \cdot 1,1 = 11275,00$ грн.

Отримані результати зведемо до таблиці:

Таблиця 6.10 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Пакет імітаційного моделювання	1	10250,00	11275,00
Пакет прикладного інженерного математичного моделювання MatchLab 14 pro	1	8700,00	9570,00
Всього			20845,00

6.3.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12} \quad , \qquad (6.11)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (24370{,}00 \cdot 2) / (2 \cdot 12) = 2030{,}83$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 6.11 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний комп'ютер	24370,00	2	2	2030,83
Робоче місце дослідника	7880,00	5	2	262,67
Пристрої виводу інформації	6875,00	4	2	286,46
Оргтехніка	8675,00	4	2	361,46
Приміщення лабораторії	212000,00	20	2	1766,67
ОС Windows 10	5450,00	2	2	454,17
Прикладний пакет Microsoft Office 2016	3795,00	2	2	316,25
Всього				5478,50

6.3.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (6.12)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e = 4{,}50$ грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0{,}45 \cdot 240{,}0 \cdot 4{,}50 \cdot 0{,}95 / 0{,}97 = 486{,}00$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 6.12 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп'ютер	0,45	240,0	486,00
Робоче місце дослідника	0,15	240,0	162,00
Пристрої виводу інформації	0,03	25,0	3,38
Оргтехніка	0,65	6,0	17,55
Серверне обладнання на основі ПК ZEVS PC 13430U i5 9400F + GTX 1060 3GB	0,60	5,0	13,50
Засоби передачі даних	0,05	200,0	45,00
Пам'ять Samsung 870 QVO 1TB 2.5" V-NAND 4bit MLC (QLC) SATA III (MZ-77Q1T0BW)	0,01	200,0	9,00
Всього			736,43

6.3.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему «Методи та засоби розподілення даних між хмарними сховищами» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{H_{св}}{100\%}, \quad (6.13)$$

де $H_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $H_{св}$ = 20%.

$B_{св}$ = (34879,17 + 1349,34) · 20 / 100% = 7245,70 грн.

6.3.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (6.14)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 30%.

$B_{сп}$ = (34879,17 + 1349,34) · 30 / 100% = 10868,55 грн.

6.3.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (6.15)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (34879,17 + 1349,34) · 50 / 100% = 18114,25 грн.

4.3.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (6.16)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 120%.

$B_{нзв}$ = (34879,17 + 1349,34) · 120 / 100% = 43474,21 грн.

Витрати на проведення науково-дослідної роботи на тему «Методи та засоби розподілення даних між хмарними сховищами» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + B_{спец} + B_{прг} + A_{обл} + B_е + B_{св} + B_{сп} + I_в + B_{нзв}. \quad (6.17)$$

$B_{заг}$= 34879,17 +1349,34 +4347,42 +8926,703812 +3490,63 +0,00 +46310,00 + 20845,00 + 5478,50 +736,43 +7245,70 +10868,55 +18114,25 +43474,21 = 206065,90 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (6.18)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,95.

ЗВ = 206065,90 / 0,95 = 216911,47 грн.

6.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів

тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою «Методи та засоби розподілення даних між хмарними сховищами» передбачають комерціалізацію протягом 4-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

Показник	1-й рік	2-й рік	3-й рік	4-й рік
Збільшення кількості споживачів, осіб	1000	2500	3500	1000

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 9500 осіб;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 2100,00 грн;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 500,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 4-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [21]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (6.19)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2021 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо $\rho = 38\%$;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2021 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 =$ (500,00·9500,00+2600,00·1000)·0,83·0,38·(1-0,18/100%)=1900915,80 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 =$ (500,00·9500,00+2600,00·3500)·0,83·0,38·(1-0,18/100%)=3581997,80 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 =$ (500,00·9500,00+2600,00·7000)·0,83·0,38·(1-0,18/100%)=5935512,60 грн.

Збільшення чистого прибутку 4-го року:

$\Delta П_4 =$ (500,00·9500,00+2600,00·8000)·0,83·0,38·(1-0,18/100%)=6607945,40 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (6.20)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ=0,14;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = 1900915,80/(1+0,14)^1+3581997,80/(1+0,14)^2+5935512,60/(1+0,14)^3+$$
$$+6607945,40/(1+0,14)^4=1667470,00+2756230,99+4006301,94+3912434,15=$$
$$=12342437,08 \text{ грн.}$$

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \tag{6.21}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=1,5;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 216911,47 грн.

$$PV = k_{інв} \cdot ЗВ = 1,5 \cdot 216911,47 = 325367,21 \text{ грн.}$$

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{6.22}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 12342437,08 грн;

PV – теперішня вартість початкових інвестицій, 325367,21 грн.

$E_{абс} = ПП - PV$ = 12342437,08 - 325367,21 = 12017069,87 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Тж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (6.23)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 12017069,87 грн;

PV – теперішня вартість початкових інвестицій, 325367,21 грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки.

$E_в = \sqrt[Тж]{1 + \frac{E_{абс}}{PV}} - 1$ = (1+12017069,87/325367,21)$^{1/4}$= 1,48.

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (6.24)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2021 році в Україні d =0,11;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,16.

$\tau_{мін}$= 0,11+0,16 = 0,27 < 1,48 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Методи та засоби розподілення даних між хмарними сховищами» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{6.25}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок} = 1 / 1{,}48 = 0{,}67$ р.

$T_{ок} <$ 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

6.6 Висновки

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Методи та засоби розподілення даних між хмарними сховищами» становить 44,3 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

Також термін окупності становить 0,67 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Методи та засоби розподілення даних між хмарними сховищами».

ВИСНОВКИ

У магістерській кваліфікаційній роботі був вдосконалений метод розподілу даних Mirror Mode, з використанням мови програмування Java.

Було проаналізовано сучасний стан проблеми розподілу даних. Розглянуто основні аналоги, визначено їх особливості та недоліки і проведено порівняння з власним програмним продуктом. В результаті порівняння було відображено доцільність розробки магістерської кваліфікаційної роботи.

Було розроблено графічний інтерфейс програмного додатку. Розроблено структурні схеми інтерфейсу робочої сторінки та інтерфейсу програмного додатку.

В результаті проведеного варіантного аналізу було обрано мову програмування Java та програмну платформу Spring для серверної, Bootstrap та HTML для клієнтської частини та середовище розробки Intellij IDE.

Було вирішено наступні задачі:

– вдосконалення методу для розподілу даних;
– розроблено модулі та сервіс для розподілу даних;
– проведено тестування програмного продукту.

Було проведено оцінку комерційного потенціалу методів та засобів розподілення даних між хмарними сховищами, які зменшують імовірність втрати критично важливого файлу.

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому технічному завданню.

Магістерську кваліфікаційну роботу було оформлено відповідно до встановлених вимог.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Quentin Truong. Introduction to Distributed Data Storage [Електронний ресурс] / Quentin Truong – Режим доступу до ресурсу: https://towardsdatascience.com/introduction-to-distributed-data-storage-2ee03e02a11d

2. Бондарчук В.К., Ліщинська Л. Б. Методи і засоби розподілення даних між хмарними сховищами. Інформаційні технології і автоматизація – 2021: матеріали XIV міжнародної науково-практичної конференції (м. Одеса, 21-22 жовтня 2021 р.). Одеса : ОНАХТ, 2021. С. 191-194.

3. Бондарчук В.К., Ліщинська Л. Б. Побудова масштабованої системи розподіленого збереження даних на основі алгоритму RAFT. Інформаційні технології та комп'ютерна інженерія. Вінниця : ВНТУ, 2021. №1. С. 31-33.

4. OPEN SOURCE PROJECTS [Електронний ресурс] – Режим доступу до ресурсу: https://www.findbestopensource.com/tagged/distributed-storage.

5. Basic Distribution Methods [Електронний ресурс] – Режим доступу до ресурсу: https://www.microfocus.com/documentation/idol/IDOL/Servers/IDOLServer/11.3/Guides/html/English/expert/Content/IDOLExpert/Distribution/DIH_Basic_Distribution_Metho.htm

6. Distributed computing. Principles, Algorithms, and Systems [Електронний ресурс] – Режим доступу до ресурсу: https://eclass.uoa.gr/modules/document/file.php/D245/2015/DistrComp.pdf

7. Luchaninov Y. Web Application Architecture in 2021: Moving in the Right Direction [Електронний ресурс] – Режим доступу до ресурсу: https://mobidev.biz/blog/web-application-architecture-types

8. Introduction to Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/introduction-to-java/

9. Overview of C# [Електронний ресурс] – Режим доступу до ресурсу: https://www.wideskills.com/csharp/overview-csharp

10. Node.js – Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

11. Go – Overview[Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/go/go_overview.htm

12. Scala – Overview[Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/scala/scala_overview.html

13. A re-introduction to JavaScript. [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript

14. Eclipse [Електронний ресурс] – Режим доступу до ресурсу: https://www.eclipse.org/

15. Introduction to NetBeans IDE [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/cd/E40938_01/doc.74/e40142/gs_nbeans.htm

16. IntelliJ IDEA [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/ru-ru/idea/

17. Требования ACID на простом языке [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/555920/

18. Overview of SQL Server [Електронний ресурс] – Режим доступу до ресурсу: https://www.wideskills.com/sql-server/overview-sql-server

19. About [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/about/

20. Different Types of Testing in Software [Електронний ресурс] – Режим доступу до ресурсу: https://www.perfecto.io/resources/types-of-testing

21. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

22. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум /В. В. Кавецький, В. О. Козловський, І. В. Причепа – Вінниця : ВНТУ, 2016. – 113с.

ДОДАТКИ

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"13" вересня 2021 р.

Технічне завдання
на магістерську кваліфікаційну роботу
«Методи і програмні засоби для розподілення даних між хмарними сховищами»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
д.т.н., профессор каф ПЗ, Ліщинська Л.Б.
«13» вересня 2021 р.
Виконав:
студент гр. 2ПІ-20м, Бондарчук В.К.
«13» вересня 2021 р.

ВНТУ – 2021

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Методи і програмні засоби для розподілення даних між хмарними сховищами».

Галузь застосування – Big Data.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ ректора по ВНТУ № 277 від « 24 » вересня 2021 р. про закріплення тем МКР.

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є підвищення ефективності розподілу даних зі зниженням ймовірності їх втрати.

Призначення роботи – розробка методів та засобів розподілу даних між хмарними сховищами.

3 Вихідні дані для проведення МКР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР:

1. Quentin Truong. Introduction to Distributed Data Storage [Електронний ресурс] / Quentin Truong – Режим доступу до ресурсу: https://towardsdatascience.com/introduction-to-distributed-data-storage-2ee03e02a11d

2. Бондарчук В.К., Ліщинська Л. Б. Методи і засоби розподілення даних між хмарними сховищами. Інформаційні технології і автоматизація – 2021: матеріали XIV міжнародної науково-практичної конференції (м. Одеса, 21-22 жовтня 2021 р.). Одеса : ОНАХТ, 2021. С. 191-194.

3 Бондарчук В.К., Ліщинська Л. Б. Побудова масштабованої системи розподіленого збереження даних на основі алгоритму RAFT. Інформаційні технології та комп’ютерна інженерія. Вінниця : ВНТУ, 2021. №1. С. 31-33.

4 OPEN SOURCE PROJECTS [Електронний ресурс] – Режим доступу до ресурсу: https://www.findbestopensource.com/tagged/distributed-storage

5 Basic Distribution Methods [Електронний ресурс] – Режим доступу до ресурсу: https://www.microfocus.com/documentation/idol/IDOL/Servers/IDOLServer/11.3/Guides/html/English/expert/Content/IDOLExpert/Distribution/DIH_Basic_Distribution_Metho.htm

4. Технічні вимоги

Методи розподілу даних між хмарними сховищами такими як Google Drive і One Drive; моделі розподілу даних між хмарними сховищами; вихідні дані для розподілу – типи файлів які є критичними; дані про кількість та тип сховищ користувача; вихідні дані – додаток, що розподіляє дані між декількома хмарними сховищами користувача.

5. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та використанні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

6. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;
– технічне завдання;
– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації та ДСТУ.

9. Стадії та етапи розробки:

№з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз методів розподілу даних	15.09.2021-26.09.2021
2	Аналіз моделей розподілу даних	27.092021-15.10.2021
3	Класифікація алгоритмів розподілу даних	16.10.2021-7.11.2021
4	Програмна реалізація додатку для розподілу даних	8.11.2021-21.11.2021
5	Економічна частина	22.11.2021-30.11.2021

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б. Протокол перевірки навчальної (кваліфікаційної) роботи

Назва роботи: **Методи і програмні засоби для розподілення даних між хмарними сховищами.**

Тип роботи: кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 2ПІ – 20м

Науковий керівник: д.т.н. проф. Ліщинська Л.Б..

Unicheck	
Оригінальність	**96,4%**
Схожість	3,6%

Аналіз звіту подібності

■ 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 автора. Роботу направити на доопрацювання.

 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.

Заявляю, що ознайомлена з повним звітом подібності, який був згенерований Системою щодо роботи «Методи і програмні засоби для розподілення даних між хмарними сховищами».

Автор _____ Бондарчук Вячеслав Костянтинович

Опис прийнятого рішення: **допустити до захисту**

Особа, відповідальна за перевірку _____ Черноволик Г. О.
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)

Додаток В. Лістинг програмного додатку

```
@Service
@AllArgsConstructor
public class UserServiceImpl implements UserService {
    private final ReactiveAuthorityRepository reactiveAuthorityRepository;
    private final ReactiveDriveRepository reactiveDriveRepository;
    private final ReactiveUserRepository reactiveUserRepository;
    private final PasswordEncoder passwordEncoder;

    @Override
    public Mono<User> create(UserDTO userDTO) {
        userDTO.setPassword(passwordEncoder.encode(userDTO.getPassword()));
        return reactiveUserRepository.save(new User(userDTO))
                .doOnSuccess((user -> Optional.ofNullable(userDTO.getDrives())
                        .ifPresent(drives -> drives.stream()
                                .peek(drive -> drive.setUserUuid(user.getUuid()))
                                .forEach(reactiveDriveRepository::save))))
                .doOnSuccess(user -> Optional.ofNullable(userDTO.getAuthorities())
                        .ifPresent(authorities -> authorities.stream()
                                .peek(authority ->
authority.setUserUuid(user.getUuid()))
                                .forEach(reactiveAuthorityRepository::save)
                        ));
    }

    @Override
    public Mono<User> update(UserDTO userDTO) {
        return reactiveUserRepository.save(null);
    }

    @Override
    public Mono<User> read(Long id) {
        return reactiveUserRepository.findById(id);
    }

    @Override
    public Mono<Void> delete(Long id) {
        return reactiveUserRepository.deleteById(id);
    }

    @Override
    public Mono<UserDetails> findByUsername(String username) {
        return reactiveUserRepository.findByUsername(username);
    }
```

```
@Configuration
public class GoogleOauth2Configuration {
    private final String applicationName;

    public GoogleOauth2Configuration(@Value("${application.name}") String
applicationName) {
        this.applicationName = applicationName;
    }

    @Bean
    public NetHttpTransport netHttpTransport() throws GeneralSecurityException,
```

```
IOException {
        return GoogleNetHttpTransport.newTrustedTransport();
    }

    @Bean
    public GoogleAuthorizationCodeFlow
googleAuthorizationCodeFlow(NetHttpTransport netHttpTransport) throws
IOException {
        return new GoogleAuthorizationCodeFlow.Builder(netHttpTransport,
                JacksonFactory.getDefaultInstance(),
GoogleDriveOauth2CredentialsLoader.getGoogleClientSecrets(), DriveScopes.all())

.setDataStoreFactory(MemoryDataStoreFactory.getDefaultInstance())
                .setAccessType("offline")
                .build();
    }

    @Bean
    public LocalServerReceiver localServerReceiver() {
        return new LocalServerReceiver.Builder()
                .setPort(8888)
                .setCallbackPath("/oauth")
                .build();
    }

    @Bean
    public AuthorizationCodeInstalledApp
authorizationCodeInstalledApp(AuthorizationCodeFlow authorizationCodeFlow,
LocalServerReceiver localServerReceiver) {
        return new AuthorizationCodeInstalledApp(authorizationCodeFlow,
localServerReceiver);
    }


}

package io.github.viacheslavbondarchuk.distributedstorage.authorization.impl;

import com.google.api.client.auth.oauth2.Credential;
import
com.google.api.client.extensions.java6.auth.oauth2.AuthorizationCodeInstalledAp
p;
import
io.github.viacheslavbondarchuk.distributedstorage.authorization.DriveAuthorizat
ionService;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;

import java.io.IOException;

@Service
@AllArgsConstructor
public class GoogleDriveAuthorizationServiceImpl implements
DriveAuthorizationService<Credential> {
    private final AuthorizationCodeInstalledApp authorizationCodeInstalledApp;

    @Override
    public Credential authorize(String username) throws IOException {
        return authorizationCodeInstalledApp.authorize(username);
    }
}

package
io.github.viacheslavbondarchuk.distributedstorage.configuration.database;

import io.r2dbc.postgresql.PostgresqlConnectionConfiguration;
import io.r2dbc.postgresql.PostgresqlConnectionFactory;
```

```
import io.r2dbc.spi.ConnectionFactory;
import lombok.AllArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.r2dbc.config.AbstractR2dbcConfiguration;

@Configuration
@AllArgsConstructor
public class R2dbcConfiguration extends AbstractR2dbcConfiguration {
    private final PostgresqlConnectionConfiguration
postgresqlConnectionConfiguration;

    @Override
    public ConnectionFactory connectionFactory() {
        return new
PostgresqlConnectionFactory(postgresqlConnectionConfiguration);
    }

    @Bean
    public PostgresqlConnectionConfiguration
postgresqlConnectionConfiguration() {

    }
}

package io.github.viacheslavbondarchuk.distributedstorage.configuration.google;

import com.google.api.client.auth.oauth2.AuthorizationCodeFlow;
import
com.google.api.client.extensions.java6.auth.oauth2.AuthorizationCodeInstalledAp
p;
import com.google.api.client.extensions.jetty.auth.oauth2.LocalServerReceiver;
import
com.google.api.client.googleapis.auth.oauth2.GoogleAuthorizationCodeFlow;
import com.google.api.client.googleapis.javanet.GoogleNetHttpTransport;
import com.google.api.client.http.javanet.NetHttpTransport;
import com.google.api.client.json.jackson2.JacksonFactory;
import com.google.api.client.util.store.MemoryDataStoreFactory;
import com.google.api.services.drive.Drive;
import com.google.api.services.drive.DriveScopes;
import
io.github.viacheslavbondarchuk.distributedstorage.util.GoogleDriveOauth2Credent
ialsLoader;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.io.IOException;
import java.security.GeneralSecurityException;

@Configuration
public class GoogleOauth2Configuration {
    private final String applicationName;

    public GoogleOauth2Configuration(@Value("${application.name}") String
applicationName) {
        this.applicationName = applicationName;
    }

    @Bean
    public NetHttpTransport netHttpTransport() throws GeneralSecurityException,
IOException {
        return GoogleNetHttpTransport.newTrustedTransport();
    }

    @Bean
```

```
    public GoogleAuthorizationCodeFlow
googleAuthorizationCodeFlow(NetHttpTransport netHttpTransport) throws
IOException {
        return new GoogleAuthorizationCodeFlow.Builder(netHttpTransport,
                JacksonFactory.getDefaultInstance(),
GoogleDriveOauth2CredentialsLoader.getGoogleClientSecrets(), DriveScopes.all())

.setDataStoreFactory(MemoryDataStoreFactory.getDefaultInstance())
                .setAccessType("offline")
                .build();
    }

    @Bean
    public LocalServerReceiver localServerReceiver() {
        return new LocalServerReceiver.Builder()
                .setPort(8888)
                .setCallbackPath("/oauth")
                .build();
    }

    @Bean
    public AuthorizationCodeInstalledApp
authorizationCodeInstalledApp(AuthorizationCodeFlow authorizationCodeFlow,
LocalServerReceiver localServerReceiver) {
        return new AuthorizationCodeInstalledApp(authorizationCodeFlow,
localServerReceiver);
    }


}

package
io.github.viacheslavbondarchuk.distributedstorage.configuration.security;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Scope;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.web.server.ServerHttpSecurity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.server.SecurityWebFilterChain;

@Configuration
public class SecurityConfiguration {

    @Bean
    public SecurityWebFilterChain securityWebFilterChain(ServerHttpSecurity
http) {
        return http
                .csrf().disable()
                .cors().disable()
                .authorizeExchange()
                .pathMatchers(HttpMethod.POST,
"/registration/users").permitAll()
                .anyExchange().authenticated()
                .and().httpBasic()
                .and().build();
    }

    @Bean
    @Scope("prototype")
    public BCryptPasswordEncoder bCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }

}
```

```
package io.github.viacheslavbondarchuk.distributedstorage.controller;

import io.github.viacheslavbondarchuk.distributedstorage.model.User;
import
io.github.viacheslavbondarchuk.distributedstorage.service.DatabaseService;
import lombok.AllArgsConstructor;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import reactor.core.publisher.Mono;

@RestController
@AllArgsConstructor
@RequestMapping("/registration/users")
public class RegistrationController {
    private final DatabaseService<User, Long> databaseService;

    @PostMapping(consumes = "application/json")
    public Mono<User> register(@RequestBody User user) {
        return databaseService.create(user);
    }

}

package io.github.viacheslavbondarchuk.distributedstorage.drive.google;

import com.google.api.client.auth.oauth2.Credential;
import com.google.api.client.http.javanet.NetHttpTransport;
import com.google.api.services.drive.Drive;
import
io.github.viacheslavbondarchuk.distributedstorage.authorization.DriveAuthorizat
ionService;
import io.github.viacheslavbondarchuk.distributedstorage.drive.DriveFactory;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Component;

@Component
@AllArgsConstructor
public class GoogleDriveFactoryImpl implements DriveFactory<Drive> {
    private final DriveAuthorizationService<Credential>
driveAuthorizationService;
    private final NetHttpTransport netHttpTransport;


    @Override
    public Drive create() {
//        return new Drive.Builder(netHttpTransport,
JacksonFactory.getDefaultInstance(), driveAuthorizationService.authorize("us"))
//                .setApplicationName("Distributed Storage")
//                .build();
        return null;
    }
}
package io.github.viacheslavbondarchuk.distributedstorage.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.relational.core.mapping.Table;
import org.springframework.security.core.GrantedAuthority;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Table("authorities")
public class Authority implements GrantedAuthority {
```

```
    private Long id;
    private Long uuid;
    private AuthorityType authorityType;

    @Override
    public String getAuthority() {
        return authorityType.name();
    }
}
```

```
package io.github.viacheslavbondarchuk.distributedstorage.model;

public enum AuthorityType {
    USER, ADMIN;
}
```

```
package io.github.viacheslavbondarchuk.distributedstorage.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.annotation.Id;
import org.springframework.data.relational.core.mapping.Table;

@Data
@Table("drives")
@NoArgsConstructor
@AllArgsConstructor
public class Drive {
    @Id
    private Long id;
    private Long uuid;
    private boolean isSelected;
    private DriveType driveType;
}
```

```
package io.github.viacheslavbondarchuk.distributedstorage.model;

public enum DriveType {
    GOOGLE_DRIVE, ONE_DRIVE, DROP_BOX;
}
```

```
package io.github.viacheslavbondarchuk.distributedstorage.model;

import lombok.*;
import org.springframework.data.annotation.Id;
import org.springframework.data.relational.core.mapping.Table;




@Data
@Builder
@Table("files")
@EqualsAndHashCode
@NoArgsConstructor
@AllArgsConstructor
public class File {
    @Id
    private Long id;
    private Long uuid;
    private String name;
    private FileType fileType;
}
```

```
package io.github.viacheslavbondarchuk.distributedstorage.model;
```

```
public enum FileType {
    FOLDER, FILE;
}
```

```
package io.github.viacheslavbondarchuk.distributedstorage.model;

import lombok.*;
import org.springframework.data.annotation.Id;
import org.springframework.data.relational.core.mapping.Column;
import org.springframework.data.relational.core.mapping.Table;
import org.springframework.security.core.userdetails.UserDetails;

import java.util.List;

@Data
@Builder
@Table("users")
@EqualsAndHashCode
@NoArgsConstructor
@AllArgsConstructor
public class User implements UserDetails {
    @Id
    @Column
    private Long id;
    @Column
    private String username;
    @Column
    private String email;
    @Column
    private String password;

    private transient List<File> files;
    private transient List<Drive> drives;
    private transient List<Authority> authorities;

    private boolean isEnabled = true;
    private boolean isAccountNonLocked = true;
    private boolean isAccountNonExpired = true;
    private boolean isCredentialsNonExpired = true;


}
```

```
package io.github.viacheslavbondarchuk.distributedstorage.repository;

import io.github.viacheslavbondarchuk.distributedstorage.model.Authority;
import org.springframework.data.r2dbc.repository.R2dbcRepository;


public interface ReactiveAuthorityRepository extends R2dbcRepository<Authority,
Long> {
}
```

```
package io.github.viacheslavbondarchuk.distributedstorage.repository;

import io.github.viacheslavbondarchuk.distributedstorage.model.Drive;
import org.springframework.data.r2dbc.repository.R2dbcRepository;

public interface ReactiveDriveRepository extends R2dbcRepository<Drive, Long> {

}
```

```
package io.github.viacheslavbondarchuk.distributedstorage.repository;

import io.github.viacheslavbondarchuk.distributedstorage.model.User;
import org.springframework.data.r2dbc.repository.Query;
```

```
import org.springframework.data.r2dbc.repository.R2dbcRepository;
import
org.springframework.security.core.userdetails.ReactiveUserDetailsService;
import reactor.core.publisher.Mono;

public interface ReactiveUserRepository extends R2dbcRepository<User, Long>,
ReactiveUserDetailsService {
    @Override
    @Query("insert into users(id, username, email, password, is_enabled,
is_account_non_locked, is_account_non_expired, is_credentials_non_expired) " +
            "values ($1.id, $1.username, $1.email, $1.password, $1.isEnabled,
$1.isAccountNonLocked, $1.isAccountNonExpired, $1.isCredentialsNonExpired)")
    <S extends User> Mono<S> save(S entity);

}
```

```
package io.github.viacheslavbondarchuk.distributedstorage.service.user.impl;

import io.github.viacheslavbondarchuk.distributedstorage.model.Authority;
import
io.github.viacheslavbondarchuk.distributedstorage.repository.ReactiveAuthorityR
epository;
import
io.github.viacheslavbondarchuk.distributedstorage.service.AbstractDatabaseServi
ce;
import org.springframework.stereotype.Service;

@Service
public class AuthorityDatabaseServiceImpl extends
AbstractDatabaseService<Authority, Long, ReactiveAuthorityRepository> {
    public AuthorityDatabaseServiceImpl(ReactiveAuthorityRepository repository)
{
        super(repository);
    }
}
```

```
package io.github.viacheslavbondarchuk.distributedstorage.service.user.impl;

import io.github.viacheslavbondarchuk.distributedstorage.model.Drive;
import
io.github.viacheslavbondarchuk.distributedstorage.repository.ReactiveDriveRepos
itory;
import
io.github.viacheslavbondarchuk.distributedstorage.service.AbstractDatabaseServi
ce;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;

@Service
public class DriveDatabaseServiceImpl extends AbstractDatabaseService<Drive,
Long, ReactiveDriveRepository> {
    public DriveDatabaseServiceImpl(ReactiveDriveRepository repository) {
        super(repository);
    }
}
```

```
package io.github.viacheslavbondarchuk.distributedstorage.service.user.impl;

import io.github.viacheslavbondarchuk.distributedstorage.model.User;
import
io.github.viacheslavbondarchuk.distributedstorage.repository.ReactiveAuthorityR
epository;
import
io.github.viacheslavbondarchuk.distributedstorage.repository.ReactiveDriveRepos
itory;
```

```
import
io.github.viacheslavbondarchuk.distributedstorage.repository.ReactiveUserReposi
tory;
import
io.github.viacheslavbondarchuk.distributedstorage.service.AbstractDatabaseServi
ce;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import reactor.core.publisher.Mono;
import reactor.core.scheduler.Schedulers;

import java.util.Optional;

@Service
public class UserDatabaseServiceImpl extends AbstractDatabaseService<User,
Long, ReactiveUserRepository> {
    private final ReactiveAuthorityRepository reactiveAuthorityRepository;
    private final ReactiveDriveRepository reactiveDriveRepository;
    private final PasswordEncoder passwordEncoder;

    public UserDatabaseServiceImpl(ReactiveAuthorityRepository
reactiveAuthorityRepository,
                                   ReactiveDriveRepository
reactiveDriveRepository, PasswordEncoder passwordEncoder,
ReactiveUserRepository reactiveUserRepository) {
        super(reactiveUserRepository);
        this.reactiveAuthorityRepository = reactiveAuthorityRepository;
        this.reactiveDriveRepository = reactiveDriveRepository;
        this.passwordEncoder = passwordEncoder;
    }


    @Override
    @Transactional
    public Mono<User> create(User user) {
        return repository.save(encodePassword(user))
                .doOnSuccess(savedUser ->
Optional.ofNullable(savedUser.getDrives())
                        .ifPresent(drives -> drives.stream()
                                .peek(drive -> drive.setUuid(user.getId()))
                                .forEach(reactiveDriveRepository::save)))
                .doOnSuccess(savedUser ->
Optional.ofNullable(user.getAuthorities())
                        .ifPresent(authorities -> authorities.stream()
                                .peek(authority ->
authority.setUuid(user.getId()))
                                .forEach(reactiveAuthorityRepository::save)
                        ))
                .subscribeOn(Schedulers.parallel());
    }

    @Override
    public Mono<User> update(User user) {
        return repository.save(user);
    }

    @Override
    public Mono<User> read(Long id) {
        return repository.findById(id);
    }

    @Override
    public Mono<Void> delete(Long id) {
        return repository.deleteById(id);
    }
```

```
    public Mono<UserDetails> findByUsername(String username) {
        return repository.findByUsername(username);
    }



    private User encodePassword(User user) {

        Optional.of(user)
                .map(User::getPassword)
                .map(passwordEncoder::encode)
                .ifPresentOrElse(user::setPassword, () -> {throw new
IllegalArgumentException("");});
        return user;
    }
}
```

```
package io.github.viacheslavbondarchuk.distributedstorage.service;

import org.springframework.data.r2dbc.repository.R2dbcRepository;
import reactor.core.publisher.Mono;
import reactor.core.scheduler.Schedulers;

;

public class AbstractDatabaseService<T, ID, R extends R2dbcRepository<T, ID>>
implements DatabaseService<T, ID> {
    protected final R repository;

    public AbstractDatabaseService(R repository) {
        this.repository = repository;
    }


    @Override
    public Mono<T> create(T t) {
        return repository.save(t)
                .subscribeOn(Schedulers.parallel());
    }

    @Override
    public Mono<T> update(T t) {
        return repository.save(t)
                .subscribeOn(Schedulers.parallel());
    }

    @Override
    public Mono<T> read(ID id) {
        return repository.findById(id)
                .subscribeOn(Schedulers.parallel());
    }

    @Override
    public Mono<Void> delete(ID id) {
        return repository.deleteById(id)
                .subscribeOn(Schedulers.parallel());
    }
}
```

```
package io.github.viacheslavbondarchuk.distributedstorage.storage.impl;

import io.github.viacheslavbondarchuk.distributedstorage.model.File;
import io.github.viacheslavbondarchuk.distributedstorage.storage.Storage;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;
```

```
public final class GoogleDriveStorageImpl implements Storage<File> {

    @Override
    public Mono<File> save(File entity) {
        return null;
    }

    @Override
    public Mono<File> findByName(String name) {
        return null;
    }

    @Override
    public Flux<File> findAll() {
        return null;
    }

    @Override
    public Mono<Void> deleteByName(String name) {
        return null;
    }
}
```

```
package io.github.viacheslavbondarchuk.distributedstorage.util;

import com.google.api.client.googleapis.auth.oauth2.GoogleClientSecrets;
import com.google.api.client.json.jackson2.JacksonFactory;

import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Objects;

public final class GoogleDriveOauth2CredentialsLoader {
    private static final String GOOGLE_DRIVE_CREDENTIALS_FILE_PATH =
"credentials/google-drive-oauth.json";

    private static GoogleClientSecrets googleClientSecrets;

    private GoogleDriveOauth2CredentialsLoader() {
    }

    public static GoogleClientSecrets getGoogleClientSecrets() throws
IOException {
        if (Objects.isNull(googleClientSecrets)) {
            synchronized (GoogleDriveOauth2CredentialsLoader.class) {
                if (Objects.isNull(googleClientSecrets)) {
                    try (InputStream inputStream =
GoogleDriveOauth2CredentialsLoader.class.getClassLoader().getResourceAsStream(G
OOGLE_DRIVE_CREDENTIALS_FILE_PATH);
                         InputStreamReader inputStreamReader = new
InputStreamReader(Objects.requireNonNull(inputStream))) {
                        googleClientSecrets =
GoogleClientSecrets.load(JacksonFactory.getDefaultInstance(),
inputStreamReader);
                    }
                }
            }
        }
        return googleClientSecrets;
    }
}
```

```
package io.github.viacheslavbondarchuk.distributedstorage;

import org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.data.r2dbc.repository.config.EnableR2dbcRepositories;
import
org.springframework.security.config.annotation.method.configuration.EnableReact
iveMethodSecurity;
import
org.springframework.security.config.annotation.web.reactive.EnableWebFluxSecuri
ty;
import org.springframework.web.reactive.config.EnableWebFlux;

@EnableWebFlux
@EnableWebFluxSecurity
@SpringBootApplication
@EnableR2dbcRepositories
@EnableReactiveMethodSecurity
public class DistributedStorageApplication {

    public static void main(String[] args) {
        SpringApplication.run(DistributedStorageApplication.class, args);
    }

}
```

Додаток Г. Ілюстративна частина

МЕТОДИ І ПРОГРАМНІ ЗАСОБИ ДЛЯ РОЗПОДІЛЕННЯ ДАНИХ МІЖ ХМАРНИМИ СХОВИЩАМИ

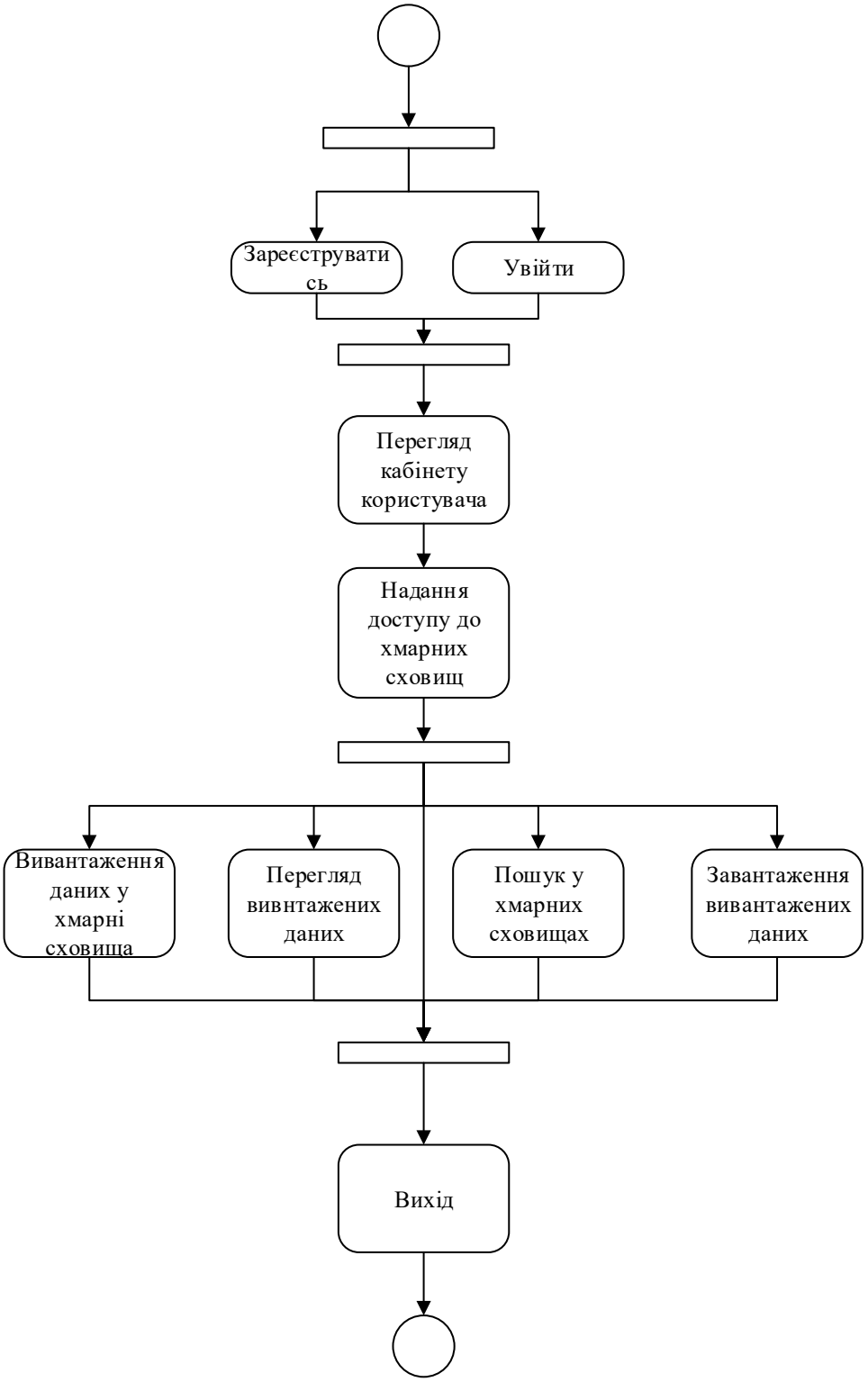

Рисунок Г.1 - Модель діяльності системи

Рисунок Г.2 - ER-діаграма зв’язків

```java
package io.github.viacheslavbondarchuk.distributedstorage.service.user.impl;

import io.github.viacheslavbondarchuk.distributedstorage.dto.UserDTO;
import io.github.viacheslavbondarchuk.distributedstorage.model.User;
import io.github.viacheslavbondarchuk.distributedstorage.repository.ReactiveAuthorityRepository;
import io.github.viacheslavbondarchuk.distributedstorage.repository.ReactiveDriveRepository;
import io.github.viacheslavbondarchuk.distributedstorage.repository.ReactiveUserRepository;
import io.github.viacheslavbondarchuk.distributedstorage.service.user.UserService;
import lombok.AllArgsConstructor;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import reactor.core.publisher.Mono;

import java.util.Optional;

@Service
@AllArgsConstructor
public class UserServiceImpl implements UserService {
    private final ReactiveAuthorityRepository reactiveAuthorityRepository;
    private final ReactiveDriveRepository reactiveDriveRepository;
    private final ReactiveUserRepository reactiveUserRepository;
    private final PasswordEncoder passwordEncoder;

    @Override
    public Mono<User> create(UserDTO userDTO) {
        userDTO.setPassword(passwordEncoder.encode(userDTO.getPassword()));
        return reactiveUserRepository.save(new User(userDTO))
                .doOnSuccess((user -> Optional.ofNullable(userDTO.getDrives())
                        .ifPresent(drives -> drives.stream()
                                .peek(drive -> drive.setUserUuid(user.getUuid()))
                                .forEach(reactiveDriveRepository::save))))
                .doOnSuccess(user -> Optional.ofNullable(userDTO.getAuthorities())
                        .ifPresent(authorities -> authorities.stream()
                                .peek(authority -> authority.setUserUuid(user.getUuid()))
                                .forEach(reactiveAuthorityRepository::save)
                        ));
    }

    @Override
    public Mono<User> update(UserDTO userDTO) { return reactiveUserRepository.save( entity: null); }

    @Override
    public Mono<User> read(Long id) { return reactiveUserRepository.findById(id); }

    @Override
    public Mono<Void> delete(Long id) { return reactiveUserRepository.deleteById(id); }

    @Override
    public Mono<UserDetails> findByUsername(String username) { return reactiveUserRepository.findByUsername(username); }
}
```

Рисунок Г.3 – Користувацький сервіс

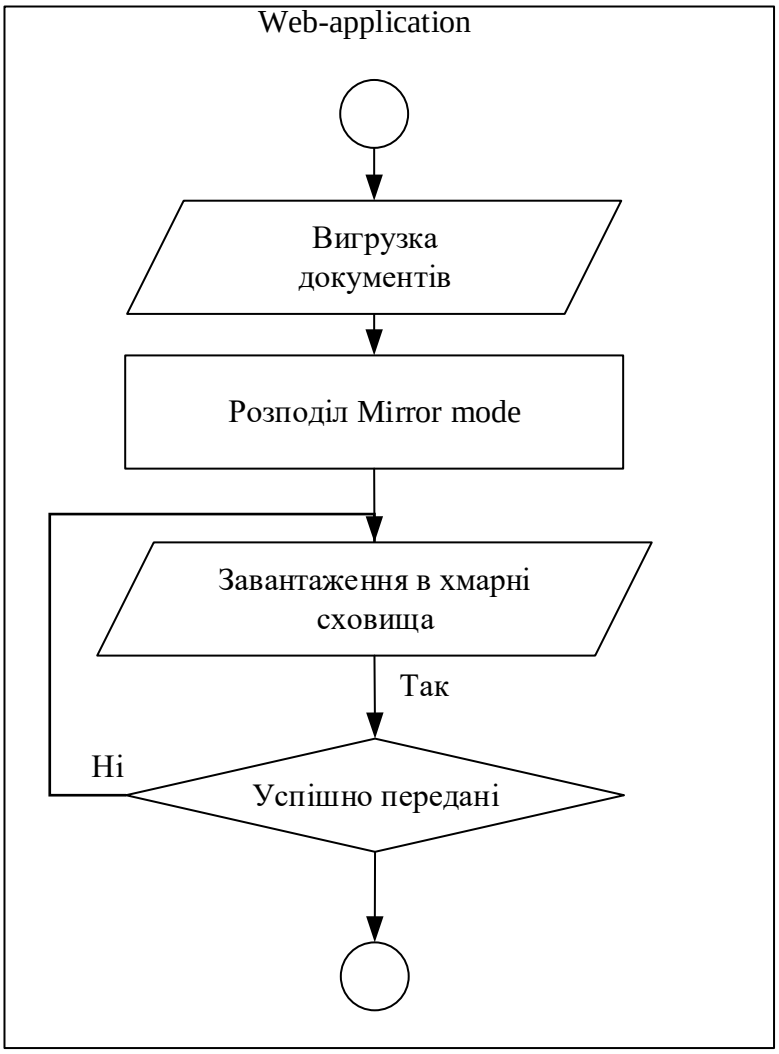

Рисунок Г.4 - Загальна схема Mirror Mode з використанням хмарних сховищ

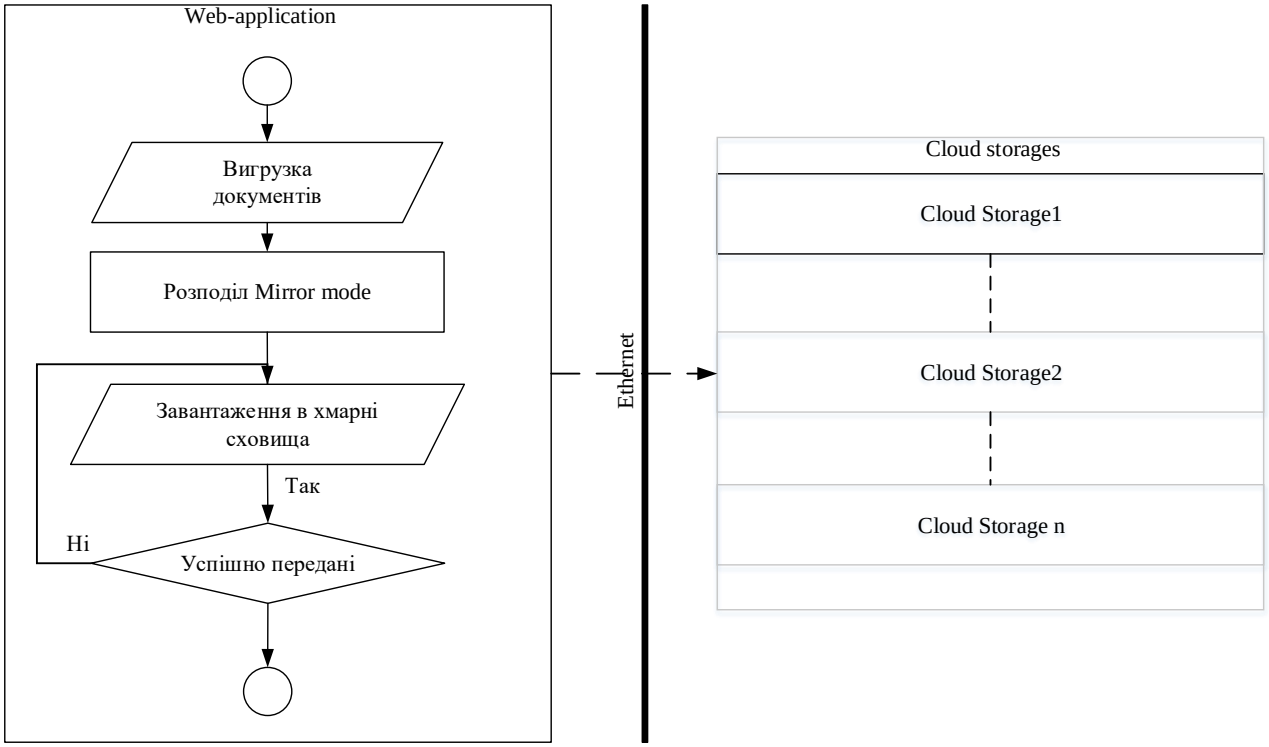

Рисунок Г.5 - Загальна схема вдосконаленого методу Mirror Mode

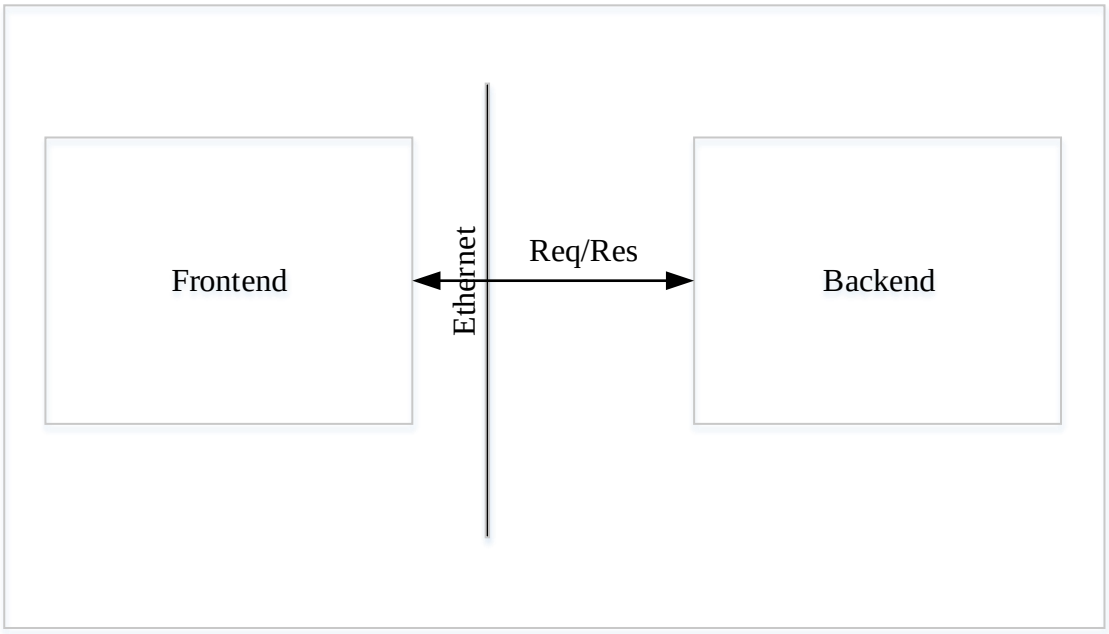

Рисунок Г.6 - Загальна структура веб-додатка