НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

"___"_____2016 р.

Дипломна робота

першого (бакалаврського) **рівня вищої освіти**
(першого (бакалаврського), другого (магістерського))

зі спеціальності7.05010102, 8.05010102 Інформаційні технології проектування
7.05010103, 8.05010103 Системне проектування
(код та назва спеціальності)

на тему: Серверна_частина_системи_зберігання корпоративного контенту

Виконав: студент 4 курсу, групи ДА-21
(шифр групи)

_____ Журавльов Павло Володимирович_____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник _____ к.т.н., доц. Безносик О.Ю._ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант економічний проф. док. ек. н. Семенченко Н. В.__ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент_____д.т.н., проф. Бідюк П.І._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Нормоконтроль _____ст. викладач Бритов О.А.

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2016 року

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти Перший(Бакалаврський).
(перший (бакалаврський), другий (магістерський) або спеціаліста)

Спеціальність 7.05010102, 8.05010102 Інформаційні технології проектування
7.05010103, 8.05010103 Системне проектування
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____2016 р.

ЗАВДАННЯ
на дипломний проект (роботу) студенту
Журавльову Павлу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)) Серверна_частина_системи_зберігання корпоративного контенту
керівник проекту (роботи)) Безносик О.Ю., к.т.н., доц.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 12 травня 2016 р. № 50-ст

2. Строк подання студентом проекту (роботи) 08.06.2016

3. Вихідні дані до проекту (роботи)

Дослідити способи організації процесу зберігання корпоративних даних.

Дослідити існуючі програмні продукти та сформувати вимоги до власного програмного продукту.

Дослідити можливості використання метаданих для оптимізації робочого процесу.

Для реалізації використовувати об’єктне сховище Openstack Swift, оптимізувати роботу з ним використовуючи індексацію та пошук по метаданим.

Результати досліджень мають включати наступні пункти: архітектура програмного продукту, яка відповідає заявленим вимогам, результати тестування прототипа програмної системи.

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)

1. Постановка задачі та аналіз предметної області
2. Дослідження технологій створення файлових сховищ
3. Проектування архітектури програмної системи
4. Вибір інструментальної бази програмної системи
5. Результати досліджень
6. Функціонально-вартісний аналіз програмного продукту

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслеників, плакатів тощо)

1. Архітектура системи – плакат.
2. Архітектура сховища – плакат.
3. Архітектура пошукової системи – плакат.

6. Консультанти розділів проекту (роботи)[1*]

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	проф. док. ек. н. Семенченко Н. В.		

7. Дата видачі завдання 01.02.2016

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2016	
2	Збір інформації	15.02.2016	

[1*] Консультантом не може бути зазначено керівника дипломного проекту (роботи).

3	Вивчення алгоритмів оптимізації та вибір варіанту для розробки	28.02.2016	
4	Розробка алгоритму та структури системи	10.03.2016	
5	Розробка програмної моделі	15.03.2016	
6	Тестування додатку та отримання результатів	25.03.2016	
7	Оформлення дипломної роботи	31.05.2016	
8	Отримання допуску до захисту та подача роботи в ДЕК		

Студент _____ П.В. Журавльов
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ О.Ю. Безносик
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

бакалаврської дипломної роботи Журавльова Павла Володимировича
на тему «Серверна частина системи зберігання корпоративного контенту»

Дана дипломна робота присвячена дослідженню проблеми зберігання корпоративних даних. Метою роботи є аналіз існуючих способів організації безпечного зберігання великих обсягів даних, і реалізація програмної системи, що дозволяє розгорнути файлове сховище на апаратному забезпеченні підприємства.

Розроблений програмний продукт призначений для невеликих підприємств, які працюють з великими обсягами даних і потребують маловитратних, легких в розгортанні і масштабуванні рішень.

Були досліджені особливості роботи з даними для малих підприємств. На їх основі сформульовані вимоги до програмного продукту, який оптимізує роботу з даними.

Результатом роботи є спроектована архітектура спрямована на задоволення цих вимог і розроблений реалізуючий її прототип програмної системи зберігання даних.

Загальний обсяг роботи: 71 с., 11 рис., 6 табл., 19 посилань.

Ключові слова: файлове сховище, об’єктне сховище, метадані, Openstack Swift, Elasticsearch, Ruby on Rails

АНОТАЦІЯ

бакалаврской дипломной работы Журавлёва Павла Владимировича
на тему: « Серверная часть системы хранения корпоративного контента»

Данная дипломная работа посвящена исследованию проблемы хранения корпоративных данных. Целью работы является анализ существующих способов организации безопасного хранения больших объемов данных, и реализация программной системы, позволяющего развернуть файловое хранилище на аппаратном обеспечении предприятия.

Разработанный программный продукт предназначен для небольших предприятий, которые работают с большими объемами данных и нуждаются в малозатратном, легко развертываемом и масштабируемом решении.

Были исследованы особенности работы с данными для малых предприятий. На их основе сформулированы требования к программному продукту, оптимизирующему работу с данными.

Результатом работы является спроектированная архитектура направленная на удовлетворение этих требований и разработан реализующий ее прототип программной системы хранения данных.

Результат работы: Общий объем работы 71 с., 11 рис., 6 табл., 19 источников.

Ключевые слова: файловое хранилище, объектное хранилище, метаданные, Openstack Swift, Elasticsearch, Ruby on Rails

ABSTRACT

on Pavel Zhuravlov bachelor’s degree

thesis: “Back-end of enterprise content storage system”

This thesis is devoted to the study of corporate data storage problems. The aim is to analyze the existing methods of the safe storage of large amounts of data and implementation of a software system that allows file storage to deploy on hardware company.

The developed software application is designed for small businesses that work with large amounts of data and need cost-effective, easily deployable and scalable solution.

In this work the features of data for small businesses were investigated. On this basis the requirements to software product that optimizes data storage and usage developed.

The result of the research is architecture designed to meet these requirements and software prototype that implements it.

Bachelor's work size: 71 p., 11 pic., 6 tables, 19 sources.

Keywords: file storage, object storage, metadata, Openstack Swift, Elasticsearch, Ruby on Rails

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – програмне забезпечення.

API – прикладний програмний інтерфейс (англ. application programming interface).

REST – Передача стану подання (англ. Representation State Transfer).

SQL (мова структурованих запитів) — декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних і її модифікації, системи контролю за доступом до бази даних (англ. Structured query language).

MVC — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення (англ. Model-view-controller).

БД – база даних (англ. database).

ВСТУП

Практика зберігання документів у сховищах широко поширилася за останні п'ять років. Підприємства переходять від традиційного персонального комп'ютера с кількома вбудованими жорсткими дисками до ноутбуків з невеликими SSD накопичувачами, вважаючи віддалене сховище пріоритетним. Збільшення використання мобільних пристроїв змінює вимоги способу організації доступу до корпоративних даних. Працівники прагнуть отримати доступ до файлів в будь-який час і з різних пристроїв. Вони не хочуть бути залежними від Інтернет з’єднання, об’єму доступної пам’яті чи навіть від працездатності зовнішньої системи.

Більшість співробітників підприємств зберігають корпоративні дані на своїх пристроях, або використовують сторонні сервіси. У першому випадку подібна практика може привести до втрати даних при апаратному збої, в другому дані можуть передаватися по незахищених каналах зв'язку, що може привести до витоку важливих даних. Крім того дані зберігаються в неструктурованому вигляді і розподілені по різних пристроїв що ускладнює доступ до них.

Вирішення цих проблем дозволить підприємствам збільшити контроль над даними і надати працівникам зручний інструмент для зберігання і спільного доступу до даних. Всі наявні програмні продукти є дорогими або не володіють достатньою функціональністю для усунення всіх цих проблем.

Метою даної роботи є дослідження проблеми зберігання даних малими підприємствами, виявлення вимог до корпоративного файлового сховища, проектування архітектури, яка відповідає заявленим вимогам та створення створення прототипу програмної системи.

До основних завдань, що виносяться на дипломну роботу відносяться:

1. З’ясування специфічних для сфери зберігання даних на малих підприємствах вимог.
2. Дослідження існуючих рішень і підходів до зберігання корпоративних даних.
3. Вивчення підходів до проектування файлових сховищ з використанням пошуку за метаданими.
4. Постановка прикладної задачі, яка б надала можливість для детального дослідження.
5. Безпосереднє використання засобів дослідження до поставленої задачі.
6. Аналіз результатів, оцінка використаних та досліджених підходів, визначення переваг і недоліків створеного рішення.

1. ПОСТАНОВКА ЗАДАЧІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Особливості роботи з даними на підприємствах

Для підприємств будь-якого типу і розміру, еволюція робочих процесів висуває вимоги, які кидають виклик традиційним методам обміну файлами. Якщо співробітники не мають доступу до системи зберігання файлів з можливістю інтенсивного обміну інформацією, використовуючи будь-які пристрої, інформація буде переміщатися по неконтрольованим і потенційно небезпечним каналам зв'язку. Співробітники вимагають ефективного і зручного способу організації спільної роботи с файлами, і організації, які не реалізують цей функціонал внутрішньо будуть все частіше виявляти що конфіденційні дані передаються по каналах поза їхнім контролем - в той час, коли методи викрадення даних досягли безпрецедентного рівня складності.

Реалізуючи мобільні системи зберігання та спільного використання файлів, які використовують моделі розгортання на місці, підприємства можуть зберегти конфіденційну інформацію під наглядом без шкоди для продуктивності праці співробітників, ефективності роботи і конкурентної переваги.

З розповсюдженням публічних хмарних рішень, поширення інформації стало простіше, ніж коли-небудь. І працівники почали користуватись такими сервісами для поширення корпоративних даних.

Збільшення використання мобільних пристроїв змінює вимоги способу організації доступу до корпоративних даних. Працівники прагнуть отримати доступ до файлів в будь-який час і з різних пристроїв. Вони не хочуть бути залежними від Інтернет з’єднання, об’єму доступної пам’яті чи навіть від працездатності зовнішньої системи.

Синхронізація та поширення файлів на основі хмарних сервісів стають все популярнішими. Ці послуги є простими для використання, але рівень контролю та безпеки даних компанії є неоднозначним.

Експоненціальне зростання даних з Інтернету, мобільних пристроїв, цифрових бізнес-процесів, а також Інтернету речей надає компаніям безцінний потенціал активів, які можуть бути використані для досягнення конкурентної вигоди.

Без правильних інструментів і стратегій, можливості цих активів, як правило, залишаються нереалізованими. Більш того, неконтрольоване збільшення кількості даних може збільшувати вартість управління процесами зберігання і резервного копіювання даних.

Крім того, зростаюча популярність хмарних технологій означає, що дані підприємства можуть бути розподілені географічно і бути розміщеними в різноманітних місцях, включаючи фізичні сервера, віртуальні машини, персональні комп'ютери, мобільні пристрої тощо.

Всі ці дані повинні надійно зберігатися і мати можливість бути відновленими з резервних копій в будь-який час, щоб запобігти втратам від перебоїв в подачі електроенергії, стихійних лих, неправомірних дій і людських помилок. Нездатність надійно масштабувати сховища даних може створити перешкоди для орієнтованого на зростання підприємства.

Замість того, щоб ефективно використовувати дані в якості конкурентного активу, підприємства, які не в змозі управляти і захищати їх дані ризикують:

1. дозволити корпоративній та клієнтській інформації бути загубленою або вкраденою.
2. витрачати надмірну кількість ресурсів на зберігання даних.
3. втратити конкурентоспроможність підприємства в порівнянні з тими, хто використовує більш гнучкі і масштабовані рішення.

працівники мають потребу у засобі зберігання і обміну файлами, який буде:

1. зберігати великі обсяги даних
2. мінімізувати втрати даних через апаратні сбої
3. давати можливість доступу з пристроїв різного типу через зручний універсальний інтерфейс
4. забезпечувати доступність даних у будь який час без затримки
5. надавати механізми контролю над потоками даних в межах підприємства

1.2 Способи зберігання даних

Підприємства малого бізнесу мають широкий вибір способів зберігання і обміну даними. Вони варіюються від портативних накопичувачах флеш-пам'яті до підключаються до мережі систем зберігання даних, які можуть бути розташовані фізично в будь-якому місці мережі.

1.2.1 Флеш накопичувачі.

Цей пристрою є корисними в тому випадку коли співробітник нуждається в зручному засобі зберігання невеликого обсягу персональних даних, тому що вони споживають мало енергії, мають невеликі розміри і не мають рухомих частин.

1.2.2 Зовнішні жорсткі диски.

Підключення зовнішнього жорсткого диска до комп'ютера є простим і відносно недорогим способом додати більше дискового простору. Однак зовнішні жорсткі диски безпосередньо підключені до ПК мають ряд недоліків. Серед них:

- Дані доступні лише в місці підключення диска
- Низька завадостійкість жорстких дисків
- Дані не захищені в разі в разі пожежі чи іншої катастрофи на робочому місці

1.2.2 Публічне хмарне сховище.

Сервіси, які забезпечують віддалене зберігання і резервне копіювання через Інтернет пропонують підприємствам ряд безперечних переваг. Шляхом резервного копіювання файлів на віддаленому сервері, вони дозволяють захищатись від втрат даних, що зберігаються на робочому місці.

Такі сервіси надають можливість обмінюватися великими файлами з клієнтами і партнерами, надаючи їм захищений паролем доступ до хмарних сховищ, тим самим усуваючи необхідність відправки цих великих файлів по електронній пошті. І в більшості випадків, вони дають можливість можете увійти в обліковий запис користувача з будь-якого пристрою через веб-браузер - це відмінний спосіб для отримання файлів, коли користувач знаходиться далеко від робочого місця.

1.2.3 Мережеве сховище.

Являє собою комп'ютер з деяким дисковим масивом, підключений до мережі (зазвичай локальної) і підтримує роботу з прийнятим в ній протоколам. Диски можуть бути об'єднані в RAID-масив. Кілька таких комп'ютерів можуть бути об'єднані в одну систему. Забезпечує надійність зберігання даних, легкість доступу для багатьох користувачів, легкість адміністрування, масштабованість.

Інформація, що міститься у вигляді повідомлень електронної пошти, документів, презентацій, баз даних, графіки, аудіо файлів і електронних таблиць є важливою для більшості компаній. Зростаюча потреба зберігати великі мультимедійні файли, такі як відео, і зробити їх доступними для користувачів мережі породжує попит на більш складні рішення для зберігання даних.

1.3 Огляд існуючих програмних рішень

1.3.1 OwnCloud

ownCloud — система для організації зберігання, синхронізації й обміну даними, розміщеними на зовнішніх серверах. Спочатку проект розвивався спільнотою KDE, але згодом засновники проекту створили комерційну компанію ownCloud Inc, яка взяла в свої руки розробку ownCloud і розпочала надання платних сервісів та Enterprise-версії платформи. Для доступу до даних, збережених в ownCloud, можна використовувати веб-інтерфейс або протокол WebDAV. Додатково до зберігання даних можна відзначити функції підтримки засобів для забезпечення спільного доступу і можливість синхронізації між різними машинами таких даних, як адресна книга, календар-планувальник і закладки, з можливістю їхнього перегляду і редагування з будь-якого пристрою в будь-якій точці мережі.Сервер ownCloud можна розгорнути на будь-якому хостингу, який підтримує виконання PHP-скриптів і надає доступ до SQLite, MySQL або PostgreSQL.

1.3.2 Git-annex

Git-annex - сервіс синхронизації файлів направлений на розв’язання проблем загального доступу до файлів і синхронізації, але не залежить від будь-якої комерційної служби або центрального сервера. Він написаний на Haskell і доступний для Linux, Android, OS X і Windows. Git-annex зберігає лише посилання на файли в git репозиторії і управляє ними в окремому місці. Він також створює репліки файлу для відновлення втраченої інформації. Слід зазначити, що git-annex доступний на різних Linux дистрибутивах, включаючи: Fedora, Ubuntu, Debian і т.д.

1.3.3 Seafile

Seafile — відкрита платформа для створення сервісу хмарного зберігання даних. Крім базових функцій зберігання на віддаленому сервері і забезпечення

синхронізації даних між комп'ютерами, Seafile надає гнучкі можливості з організації спільної роботи з контентом. Для зручності спільної роботи підтримується створення робочих областей, в яких члени групи можуть розміщувати довільну інформацію, цікаву для учасників групи. Набори файлів можуть об'єднуватися в бібліотеки, до яких може відкриватися доступ для окремих користувачів або груп, а також публічний доступ.

Кожна бібліотека у сховищі представлена у формі, що нагадує Git-репозиторій. Ця особливість дає можливість використання версійного контролю, в тому числі підтримку доступу до минулих редакцій збереженого контенту, можливість відстежити всі внесені зміни (хто, коли і що міняв), повернути колишній стан файлу або відновити випадково вилучений файл. В основі Seafile лежать технології, застосовувані в системі управління сирцевими текстами Git. При цьому Seafile не залежить від Git і самостійно реалізує потрібні методи, які спрощені і перероблені для виконання завдань автоматичної синхронізації даних, забезпечення поновлення передачі даних у випадку розриву з'єднання і підтримки різних сервісів зберігання на стороні сервера. Дані зберігаються з розбиттям на блоки, що підвищує ефективність зберігання і дає можливість прискорення передачі файлів за рахунок паралельного завантаження блоків з різних серверів зберігання.

1.4 Висновки

Розглянута проблема організації роботи з даними на підприємствах. Сучасні підприємства працюють з великими обсягами різнорідних даних. За відсутності засобу організації процесів зберігання та обміну інформацією працівники змушені користуватись неоптимальними відносно їх потреб інструментами. Сформульовані характеристики якими повинен володіти інструмент зберігання та обміну файлами для сучасних малих підприємств.

Розглянуті способи зберігання файлів. Можна зробити висновок, що персональні засоби зберігання даних такі як жорсткі диски та флеш

накопичувачі не відповідають вимогам працівників. Ця проблема потребує більш комплексного рішення яке інтегрує дані підприємства в єдину систему, яка повинна мати функціонал структурування та захисту великих обсягів даних.

Досліджені відкриті програмні рішення не відповідають сформульованим вимогам. Компанії постачальники пропонують платні програмні системи, спеціалізовані на рішення даної проблеми які не підходять малим підприємствам які не мають достатньої кількості ресурсів.

Отже, було вирішено спроектувати та розробити власну програмну систему.

2. ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ СТВОРЕННЯ ФАЙЛОВИХ СХОВИЩ

2.1 Технології організації сховищ даних

Основою всіх сховищ є деякий набір протоколів фізичного зберігання даних. Сьогодні використовуються три основні класи фізичних моделей збереження даних.

2.1.1 Direct Attached Storage

Технологія DAS - пряме (безпосереднє) підключення накопичувачів до сервера або до ПК. При цьому накопичувачі (жорсткі диски, стрічкові накопичувачі) можуть бути як внутрішніми, так і зовнішніми. Випадок найпростіший DAS-системи - це один диск всередині сервера або ПК. Крім того, до DAS-системі можна віднести і організацію внутрішнього RAID-масиву дисків з використанням RAID-Контролера.

Варто відзначити, що, незважаючи на формальну можливість використання терміна «DAS-системи» по відношенню до одиночного диску або до внутрішнього масиву дисків, під DAS-системою прийнято розуміти зовнішню стійку з дисками, яку можна розглядати як автономну СЗД. Крім незалежного живлення, автономні DAS-системи мають спеціалізований контролер (процесор) для управління масивом накопичувачів. Наприклад, в якості такого контролера може виступати RAID-контролер з можливістю організації RAID-масивів різних рівнів.

Слід зазначити, що автономні DAS-системи можуть мати кілька зовнішніх каналів введення-виведення, що забезпечує можливість підключення до DAS-системі декількох комп'ютерів одночасно.

Як інтерфейсів для підключення накопичувачів (внутрішніх або зовнішніх) в технології DAS можуть виступати інтерфейси SCSI (Small Computer Systems Interface), SATA, PATA і Fibre Channel. Якщо SCSI інтерфейси, SATA і PATA застосовуються переважно для підключення внутрішніх накопичувачів, то інтерфейс Fibre Channel служить виключно для підключення зовнішніх накопичувачів і автономних СЗД. Перевага Інтерфейсу Fibre Channel тут полягає в тому, що він не має жорсткого обмеження по довжині і може використовуватися в тому випадку, коли сервер або ПК, що підключається до DAS-системі, знаходиться від неї на значній відстані. Інтерфейси SCSI та SATA також можуть застосовуватися для підключення зовнішніх СЗД (в цьому випадку інтерфейс SATA ESATA називають), однак вони мають суворе обмеження по максимальній довжині кабелю, що з'єднує DAS-систему і підключається сервер.

До основних переваг DAS-систем можна віднести їх низьку вартість (у порівнянні з іншими рішеннями СЗД), простоту розгортання і адміністрування, а також високу швидкість обміну даними між системою зберігання і сервером. Власне, саме з цієї причини вони стали дуже популярні в сегменті малих офісів і невеликих корпоративних мереж. У той же час DAS-системи мають і свої недоліки - в першу чергу це висока вартість зберігання і управління даними внаслідок їх розкиданості по організації, а також вимушений простій мережі в момент додавання нових дисків і необхідність нарощування пам'яті або процесорної потужності сервера при перевищенні певного розміру дискового простору. Перевантаженість мережевого трафіку з додаванням нових серверів ускладнює проблему захисту даних, перешкоджає ефективному використанню ресурсів і т.д. Витрати і нові проблеми ростуть як сніжний ком.

В даний час DAS-системи займають лідируюче положення, однак частка цих систем постійно скорочується, і на зміну їм приходять або універсальні рішення з можливістю плавної міграції до NAS-систем, або системи, що

передбачають можливість їх використання як в якості DAS-, і NAS так - і навіть SAN-систем.

2.1.2 Network Attached Storage

NAS-системи - це мережеві системи зберігання даних, що підключаються безпосередньо до мережі точно так же, як і мережевий принт-сервер, маршрутизатор або будь-яке інше мережеве пристрій. Фактично NAS-системи являють собою еволюцію файл-серверів. Для того щоб зрозуміти різницю між традиційним файл-сервером і NAS-пристроєм, згадаємо, що традиційний файл-сервер являє собою виділений комп'ютер (сервер), на якому зберігається інформація, доступна користувачам мережі. Для зберігання інформації можуть використовуватися жорсткі диски, що встановлюються в сервер (як правило, вони розташовуються в спеціальних кошиках), або підключення до сервера DAS-пристрої. Адміністрування файл-сервера реалізується за допомогою серверної операційної системи. Такий підхід до організації систем зберігання даних в даний час є найбільш популярним в сегменті невеликих локальних мереж, але має один істотний недолік. Справа в тому, що універсальний сервер (та ще в поєднанні з серверної операційної системою) - аж ніяк не дешеве рішення. У той же час більшість функціональних можливостей, властивих універсального сервера, в файл-сервері просто не використовується. Ідея полягає в тому, щоб створити оптимізований файл-сервер з оптимізованої операційною системою і збалансованої конфігурацією. Саме цю концепцію і втілюють в собі NAS-пристрої, які в цьому сенсі можна розглядати як тонкі файл-сервери, або, як їх ще називають, файлери (файлери).

Крім оптимізованої ОС, звільненої від всіх функцій, не пов'язаних з обслуговуванням файлової системи і реалізацією введення-виведення даних, NAS-системи мають оптимізовану по швидкості доступу файлову систему. NAS-системи проектуються таким чином, що вся їх обчислювальна потужність фокусується виключно на операціях обслуговування і зберігання файлів. Сама

операційна система розташовується у флеш-пам'яті та встановлюється фірмою-виробником. Підключення NAS-пристроїв до мережі і їх конфігурація є досить просту задачу і під силу будь-якому досвідченому користувачеві, не кажучи вже про системний адміністратора.

У порівнянні з традиційними файловими серверами, NAS-пристрої є більш продуктивними і менш дорогими. В даний час практично всі NAS-пристрої орієнтовані на використання в мережах Ethernet (Fast Ethernet, Gigabit Ethernet) на основі протоколів TCP / IP. Доступ до пристроїв NAS здійснюється за допомогою спеціальних протоколів доступу до файлів. Найбільш поширеними протоколами файлового доступу є протоколи CIFS, NFS і ДАФСА.

2.1.3 Storage Area Network

SAN - це спеціалізована мережева інфраструктура для зберігання даних (мережа зберігання даних). Ці мережі інтегруються у вигляді окремих спеціалізованих підмереж до складу локальної (LAN) або глобальною (WAN) мережі.

По суті, SAN-мережі пов'язують один або кілька серверів (SAN-серверів) з одним або декількома пристроями зберігання даних. SAN-мережі дозволяють будь-якому SAN-сервера отримувати доступ до будь-якого пристрою зберігання даних, не завантажуючи при цьому ні інші сервери, ні локальну мережу. Крім того, можливий обмін даними між пристроями зберігання даних без участі серверів. SAN-мережі дозволяють дуже великому числу користувачів зберігати інформацію в одному місці (з швидким централізованим доступом) і спільно використовувати її. Як пристрої зберігання даних можуть застосовуватися RAID-масиви, різні бібліотеки (стрічкові, магнітооптичні і ін.), А також JBOD-системи (масиви дисків, не об'єднані в RAID).

Для побудови мереж SAN використовується або стандарт Fibre Channel (FC), або стандарт ISCSI.

2.1.4 Віртуалізація

Віртуалізація змінила вигляд сучасних сховищ даних. Подібно до того, як фізичні машини абстрагувалися в віртуальні машини, фізичні сховища абстрагуються в віртуальні диски. Використовуючи приципи віртуалізації монітор віртуальних машин забезпечує емуляцію апаратної середовища для кожної віртуальної машини, включаючи комп'ютер, пам'ять і зберігання. Основні сучасні рішення, використовують емуляцію локальних фізичних дисків як спосіб надати сховище для кожної віртуальної машини. Іншими словами, використовують–DAS модель як спосіб реалізувати сховище в віртуальних машинах. Подібно до того, як основною одиницею зберігання в DAS є фізична машина, основною одиницею в пам'яті віртуальному диску є віртуальна машина. Віртуальні диски розглядаються не як незалежні об'єкти, але як частина конкретної віртуальної машини, точно так, як локальні диски є частиною фізичного комп'ютера. Як і DAS, віртуальний диск існує і знищується з самої віртуальної машиною. Більшість традиційних платформ віртуалізації використовують модель віртуального дискового сховища

Реалізація віртуальних дисків, у вигляді моделі зберігання блочного сховища DAS-стилі, на вершині NAS або SAN, ілюструє одну з важливих характеристик сучасного зберігання центрів обробки даних. Так як потік даних з віртуальної машини передається від програмного забезпечення в гипервизор, а не до апаратних засобів на шини пристрої, протокол, який використовується в віртуальній машині для обміну даними з гіпервізором не повинен відповідати протоколу використовуваному для зв'язку з пристроєм зберігання. Це призводить до поділу між моделлю зберігання, яка переміщається вгору до віртуальної машини, і протоколом, який використовується гіпервізором щоб зберігати дані, що дає гнучкість змішувати і поєднувати моделі зберігання даних і протоколи зберігання, і навіть динамічно змінювати протокол зберігання без шкоди для віртуальних машин.

Вибір реалізації повністю прозорий для додатка, тому що в кінцевому підсумку всі ці протоколи будуть виглядати однаково для віртуальної машини і адміністратора; вони будуть виглядати як локальні налаштовані фізичні диски. Таким чином, розробник програми в більшості інфраструктур публічних хмарних не може знати, який протокол зберігання використовується; Справді, протокол може навіть змінюватися динамічно.

Через поділу між моделлю зберігання і протоколом зберігання, протокол для зберігання стає інфраструктурним завданням, в першу чергу важливим для вартості і продуктивності. Функціональні можливості надають програмно визначаються рішення.

2.1.5 Програмно визначене сховище

Програмно визначені сховища (Software-defined storage, SDS) - це наступний крок у розвитку програмного забезпечення по організації зберігання даних для управління послугами зберігання на основі бізнес-орієнтованих політик (правил) незалежно від апаратного забезпечення.

Ідея використання систем зберігання даних в яких програмне забезпечення відділене від апаратних засобів є привабливою для малих підприємств оскільки дає їм можливість конкурувати з великими підприємствами.

Проте програмно визначені сховища відрізняються від віртуальних сховищ, які дозволяють безлічі пристроїв зберігання бути об'єднаними в єдиний віртуальний пристрій. Програмно визначені сховища абстрагують від апаратного забезпечення функціонал замість ємності. Програмно визначене сховище може бути представлено як програмна система яка надає послуги зберігання даних включаючи такий функціонал як реплікації, знімки сховища даних і так далі.

Програмно визначені сховища дозволяють інтегрувати наявні апаратні ресурси підприємства в єдину інфраструктуру яка надає можливість змінювати свої параметри в відповідність до потреб підприємства.

Відокремлюючи апаратне забезпечення сховища від програмного забезпечення яке управляє ним, програмно визначені сховища дозволяють підприємствам використовувати доступне різнорідне апаратне забезпечення.

2.2 Типи моделей даних сховищ

Існують три основні категорії сховищ даних: файлове, блочне і об’єктне сховище.

2.2.1 Файлове сховище

Файлове сховище забезпечує доступ до файлової системи. Це самий знайомий вигляд зберігання-це те, що ми взаємодіємо з більшістю на щоденній основі. Користувачі зберігання файлів мають доступ до файлів і може читати і писати або весь файл або його частину. Файлові системи, що операційні системи надають на всіх наших персональних комп'ютерів. У загальному середовищі, для зберігання файлів часто розглядається як мережевий диск. Хоча файлове сховище забезпечує зручну модель даних, існують проблеми з масштабуванням. Файлове сховище потребує суворої узгодженості, що створює труднощі коли система зростає і кількість запитів збільшується.

2.2.2 Блочне сховище

Блочне сховище зберігає структуровані дані, які представлені у вигляді блоків однакового розміру, не змінюючи інтерпретацію збережених бітів. Більшість пристроїв зберігання має вбудований блоковий інтерфейс на рівні драйверів. Наприклад, драйвер жорсткого диска записує і зчитує блоки на їхню адресу на отформатованому диску.

Блокові системи зберігання використовуються багатьма додатками для постійних операцій введення-виведення. Показовий приклад - більшість додатків реляційних СУБД (Oracle, DB2 і т. П.). Програми, що використовують інші засоби зберігання (наприклад, файлові системи), делегують вихідні операції введення-виведення блокових систем зберігання базової файлової системи. Ця система відіграє роль проміжного елемента між вихідним інтерфейсом блокової системи зберігання і файловими операціями введення-виведення, які генеруються додатками (наприклад, HDFS, NFS і ін.). Мережі зберігання даних завжди надають клієнтським застосуванням інтерфейс блокової системи зберігання. У багатьох випадках в залежності від підтримки обладнання можна налаштувати розмір блоків і різні параметри (наприклад, розміщення блоків на фактичному носії). Клієнтську програму виконує зіставлення власних форматів зберігання з базової блоковою системою зберігання.

У багатьох випадках в залежності від підтримки обладнання можна налаштувати розмір блоків і різні параметри (наприклад, розміщення блоків на фактичному носії). Клієнтську програму виконує зіставлення власних форматів зберігання з базової блоковою системою зберігання.

Блокова система зберігання пропонує більш високу продуктивність і швидкодію в порівнянні з файловими системами зберігання. Кожен блоковий тому можна розглядати як незалежний диск, керований за допомогою зовнішньої ОС сервера.

2.2.3 Об’єктне сховище

Особливістю об’єктного сховища є те, що кожен файл має унікальний глобальний ідентифікатор крім користувацького. Якщо фізичне розташування об'єкта змінюється, зміни обробляються внутрішнім механізмом об'єктного сховища, а ідентифікатор, призначений користувачем або додатком

залишається незмінним, що дозволяє легко отримати доступ до нього в будь-який інший час.

Об’єктне сховище може зберігати велику кількість даних та досту до них є більш простим, оскільки воно має декілька незалежних вузлів, які зберігають дані і центральний вузол не має знати місцезнаходження об’єкта для того, щоб отримати його.

Ідентифікатори можна використовувати для того, щоб легко порівняти два файла без необхідності завантаження. Об'єктне сховище також дає можливість використання великої кількості, простого, більш дешевого апаратного забезпечення різних типів, пов'язуючи все разом в єдину систему.

Кожен об'єкт, як правило, має кілька реплік, можливо в географічно розподілених кластерах. Кожен об'єкт також містить контрольну суму, яка дозволяє легко виявити пошкодження даних, в цьому випадку можлива генерація нової копії об'єкта для заміни пошкодженої.

Для досягнення економічності хмарної системи зберігання слід почати з доступної СЗД. Складнощі і обмеження неефективних файлових систем, що лежать в основі традиційних масивів зберігання NAS і SAN, можуть легко перемістити на другий план потенційне скорочення витрат, яке забезпечує хмарна СЗД. У таких системах ускладнюється адміністрування СЗД, масштабованість обмежується штучними межами ємності і відбувається вимушена прив'язка до одного постачальника, який пропонує дороге Власницьке обладнання. Об'єктна система зберігання набагато краще підходить для хмарних інфраструктур. Замість використання складної і застарілої файлової системи, якій до того ж важко керувати, об'єктні системи зберігання застосовують єдиний однорівневий адресний простір. Воно дозволяє автоматично направляти дані в правильні системи зберігання, визначає життєвий цикл змісту і зберігає активні і архівні дані на одному рівні, забезпечуючи їх належний захист. Завдяки цьому об'єктна система зберігання показує кращі результати. Вони обумовлені узгодженням важливості даних і

вартості їх зберігання, а також відсутністю великих витрат на управління для переміщення даних вручну на необхідний рівень. При цьому забезпечується необмежена масштабованість для підтримки можливості хмарної СЗД надавати ємність на вимогу.

Інші переваги об'єктної системи зберігання

• Об'єктна система зберігання може працювати максимально ефективно на стандартному серверному обладнанні.

• Об'єктна система зберігання забезпечує простий доступ до сховища з будь-якого розташування, в будь-який час і для будь-якого пристрою по протоколу HTTP.

• Хмарна СЗД зазвичай надається у вигляді додатку моделі «зберігання даних як послуга» через Інтернет. Тому використання HTTP в якості основного протоколу для доступу до об'єктним пулів зберігання даних значно спрощує постачальникам хмарних систем зберігання процес інтеграції систем зберігання в їх пропоновані послуги.

Об'єктні сховища вже знайшли застосування в галузі охорони здоров'я, фінансів та індустрії розваг, на додаток до хмарних сервісів і зараз стають привабливими для ще більш широкої групи підприємств, в той час як технічні, управлінські, а також архівні вимоги ввели попит на масивні сховища даних в межах IT відділів.

Facebook, Instagram і Twitter використовують дану технологію для зберігання мільйонів користувальницьких фотографій кожен день. Spotify використовує її для зберігання мільйонів музичних треків. Dropbox і інші сервіси зберігають мільйони завантажених документів в об'єктному сховищі, прихованому за звичним, що містить файли і папки, інтерфейсом.

Використовуючи об’єктне локальне файлове сховище, організації зберігають повний контроль над безпекою та фізичним місцезнаходженням своїх даних, і в той же час проявляються ключові переваги хмари і об'єктного сховища такі, як масштабованість, ефективність, завадостійкість і простота.

Також локальне об’єктне сховище дозволяє значно знизити вартість зберігання об'єкта, в порівнянні з публічною хмарою.

Іншою характерною особливістю зберігання об'єкта є використання метаданих, або даних про дані. Кожен об'єкт при зберіганні маркований не тільки своїм унікальним ідентифікатором, але також інформацією про дані в об'єкті. Наприклад, оцифрована пісня поміщена в об'єктне сховище може зберігатися з додатковими метаданими про назву пісні, виконавця, альбомом, тривалістю, роком і так далі. Важливо відзначити, що метадані кожного об'єкта фіксуються і зберігаються з ним. Немає необхідності тримати їх в окремій централізованій базі даних, яка може некеровано збільшитись.

Роль метаданих стає все більш суттєвою в той час, як їх використання дозволяє поглянути на дані в якості стратегічного активу різним людям починаючи від фахівців з аналізу даних і бізнес-аналітиків до посадових осіб і фінансових аудиторів. Метадані це безцінний елемент в стратегії організацій, який дозволяє оцінити статус, місце розташування і володіння корпоративними даними. Метадані також грають важливу роль в трансформаційних робочих навантаженнях, таких, як інтелектуальний аналіз даних і складна аналітика, які перетворюють вихідні дані в матеріальні цінності, які можуть бути кількісно оцінені в таких показниках, як дохід, прибуток, частка ринку і задоволеність клієнтів. Відсутність структури управління метаданими часто призводить до таких проблем, як більш тривалий пошук в сховищах даних або завданнях електронного виявлення. Прикріплення метаданих дозволяє здійснювати набагато складніший пошук даних в об'єктному сховищі, ніж у файловому, де ім'я файлу часто є єдиним ключем до вмісту файлу. Використання метаданих в якості частини платформи управління інформацією в масштабах підприємства підвищує видимість і допомагає знизити ризик, виявляти нові можливості для бізнесу та краще зрозуміти економічну цінність даних.

Важливою характеристикою об'єктного сховища є те, що схема метаданих може бути спроектована відповідно до предметної області, з урахуванням потреб користувачів файлових сховищ і додатків. Також якщо не

всі файли мають однакову здатність для пошуку, але можливість додати теги позволяє зберегти інформацію про зміст та призначення файлу. Метадані також можуть бути проаналізовані, щоб отримати інформацію про переваги користувачів і клієнтів.

Оскільки дані в об’єктному сховищі не ідентифікуються місцезнаходженням, немає необхідності перебудовувати індекс якщо дані були перерозподілені по апаратним носіям.

Оптимізований процес додавання нових файлів в об’єктне сховище має наступний вигляд. Спочатку дані, які зберігаються через інтерфейс користувача поступають на проміжний сервер, який додає до об’єкта основні метадані системи і іншу інформацію, корисну для емуляції файлової системи. Отже, дані перехоплюються під час процесу зберігання, їх вміст сканується та аналізується, і вся створена інформація додається до об'єкта у вигляді метаданих.

2.3 Використання метаданих

Метадані - це дані про використання даних, які часто не записуються а лише запам’ятовуються працівниками підприємства.

З тих пір, як дані мають велику цінність, метадані, що описують їх, стали одним з ключових активів сучасних підприємств, що грає велику роль у виявленні дійсного значення даних. Організований процес управління корпоративними метаданими може допомогти виділити цінну інформацію з неявних джерел. Бізнес метадані додають контекст до даних, зберігаючи інформацію про їх ролі в бізнес процесах з урахуванням предметної області і службові дані про стан файлів, такі як формат, місце розташування, інформацію про те, хто, коли і яким чином використовував файл.

Сховище метаданих дозволяє зберігати всю інформацію про структуру контейнерів з цими організації в одному місці. Це надає велику кількість

матеріалу для обробки і аналізу з подальшою оптимізацією бізнес процесів. Зазвичай підприємства витрачають велику кількість часу і коштів на дослідження і прийняття рішень щодо впровадження та управління новими структурами даних. Сховища метаданих грають велику роль в організації великих обсягів неструктурованих даних, надаючи інфраструктуру для обробки метаданих та складання бізнес глосарію.

Добре складений бізнес глосарій є незамінним ресурсом для надання інформації про зв'язки та відносини між даними в базах даних, графах, моделях і т. д. Управління метаданими спрощує процес побудови бізнес словника в разі, якщо кількість визначень виростає до сотень і тисяч, спрощуючи взаємодію між різними організаційними структурами підприємства.

Фіксація і освоєння цих метаданих в легко доступному каталозі може відкрити великі можливості для організації. Зокрема, каталог метаданих може підвищити доступність даних підприємства. Працівники можуть швидко і впевнено зібрати необхідні дані для аналізу, зрозуміти, як взаємодіють дані. Користувацькі інтерфейси дозволяють легко для фахівцям по роботі з даними, власникам і користувачам мати зручний доступ до каталогу, що може допомогти в організації спільної роботи і отримання загального розуміння джерел даних по всьому підприємству.

Метадані можуть зберігатися або всередині, в тому ж файлі , або зовні, в окремому файлі або полі з описом даних. Сховище даних зазвичай зберігає метадані окремо від даних, але воно може бути спроектовано для підтримки мішаних підходів до зберігання метаданих. Кожен варіант має свої переваги і недоліки.

Внутрішнє зберігання означає, що метадані завжди переміщуються як частина описуваних ними даних. Таким чином, метадані завжди доступні з даними, і ними можна маніпулювати локально. Цей метод створює надлишковість і не дозволяє управляти всіма метадані системи в одному місці.

Це, можливо, збільшує цілісність, так як метадані легко змінюються щоразу, коли змінюються дані.

Відокремлене сховище метаданих дозволяє локалізувати метадані для всього сховища, наприклад, в базі даних, для більш ефективного пошуку і управління. При такому підході, метадані можуть бути об'єднані з контентом, коли інформація передається, або можна вставляти веб посилання.

Метадані можуть бути збережені в бінарному вигляді або зручному для читання форматі. У другому випадку, наприклад, використовуючи формати XML або JSON, користувачі можуть зрозуміти і редагувати його без спеціальних інструментів. З іншого боку, ці формати рідко оптимізовані для ємності, часу передачі, і швидкості обробки даних. Бінарні метадані позбавлені цих недоліків, але вимагають спеціальних бібліотек для перетворення двійковій інформації в формат прийнятний для людини.

2.4 Висновки

В даному розділі були досліджені технології яка використовується для побудови сучасних систем зберігання і обміну даними. Розглянуто основні типи фізичних моделей які використовуються для побудови апаратних сховища даних. Розглянуто використання прийомів віртуалізації для поділу програмних засобів управління розподіленим сховищем і апаратних засобів зберігання даних.

В якості основи для побудови програмного продукту вибрані програмно-обумовлені сховища оскільки вони реалізують більшу частину необхідного функціоналу з управління розподілом сховищем і добре підходять для сформульованої задачі.

Розглянуто три основних типи моделей даних розподілених сховищ. Для реалізації поставленого завдання обрана об'єктна модель даних оскільки вона є

оптимальною для вирішення завдання зберігання великих обсягів неструктурованих даних.

Розглянуто можливості використання метаданих для оптимізації роботи з сховищем. Можливість об'єктне сховище додавати метадані до файлів можна використовувати для структурування даних. Метадані файлів можна зберігати разом у вигляді пошукових індексів що дає можливість обробляти складні пошукові запити.

3. ПРОЕКТУВАННЯ АРХІТЕКТУРИ ПРОГРАМНОЇ СИСТЕМИ

3.1 Проектування архітектури

Під архітектурою програмної системи розуміється уявлення, яке т інформацію про компоненти складають систему, про взаємозв'язки між цими компонентами і правилах, що регламентують ці взаємозв'язки. Програмний продукт являє собою систему з трьох основних елементів, яка зображена на рисунку 3.1:

1. проміжний сервер
2. файлове сховище
3. Пошукова система

Взаємодія між компонентами системи здійснюється з використанням протоколів TCP / IP на транспортно-мережевому рівні і HTTP на прикладному рівні.

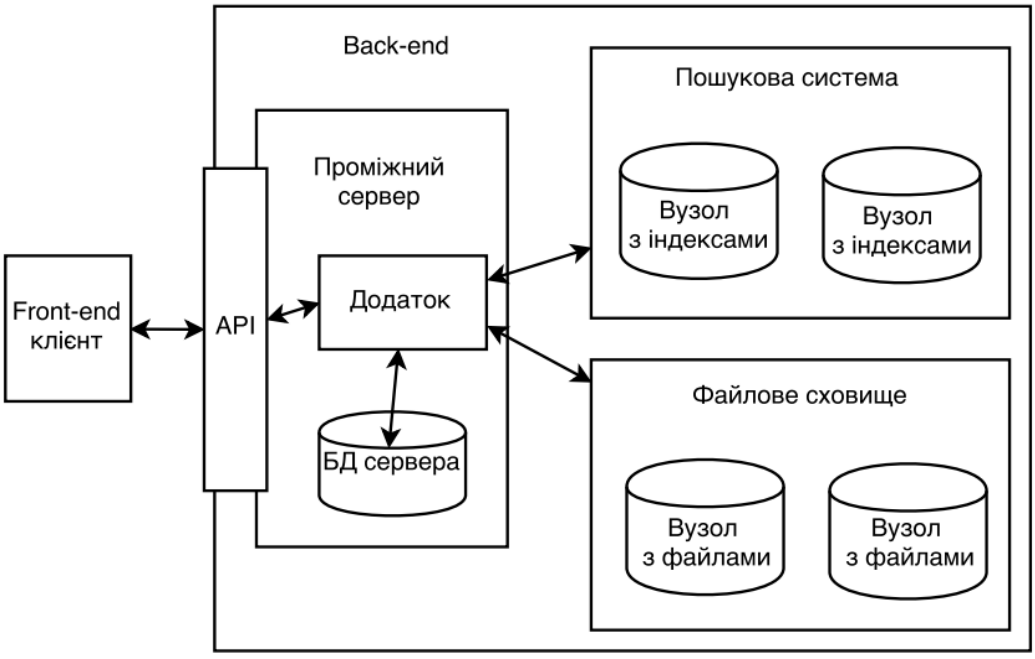

Рисунок 3.1 – Загальна архітектура системи

Проміжний сервер містить серверний додаток, яке пов'язує клієнтську програму і інші компоненти архітектури. Для зберігання його сутностей сервер містить базу даних. Додаток надає API для клієнтського додатка і містить модулі-клієнти для відправлення запитів до API сховища і пошукової системи. Також воно надає функціонал адміністрування системи, реєстрації та авторизації користувачів, створення і зберігання схем метаданих.

Файлове сховище являє собою кластер, на якому розгорнуто об'єктне сховище. Надає API для додавання, зміни і видалення об'єктів і адміністрування системи.

Пошукова система являє собою кластер, на якому розгорнута пошукова система, що зберігає індекси файлів сховища і їх метаданих і надає API для пошуку і індексації об'єктів і адміністрування системи.

3.2 Опис API

API (скор. Від англ. Application Programming Interface - «прикладний програмний інтерфейс») - набір функцій, який визначає спосіб взаємодії користувача з програмною системою і надає таку функціональність:

1. Робота з користувачами: реєстрація, авторизація.
2. Робота з файлами: створення, видалення, перегляд папок, додавання, видалення і зміна файлів.
3. Робота з метаданими: додавання, редагування, видалення тегів.
4. Обробка пошукових запитів
5. Адміністрування: перегляд і зміна статусів користувачів.

3.3 Функціональні аспекти системи. Діаграми прецедентів.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутних), асоціації між акторами та прецедентами, відношення серед

прецедентів та відношень узагальнення між акторами.

Рисунок 3.2 – Діаграма прецедентів

Ролі в системі:

- Користувач.
- Адміністратор.

Можливості кожного з акторів наведені на рис 3.2.

Кожний користувач повинен пройти етап автентифікації та авторизації перед використанням системи задля підтвердження особистості та отримання відповідних прав доступу. Адміністратор може переглядати статуси акаунтів користувачів та змінювати їх.

На рисунку 3.3 наведені можливості користувача щодо роботи з файлами. Він може переглядати файлову структуру у вигляді ієрархічної папочної структури, додавати та видаляти папки, додавати файли різних типів з додаванням метаданих у вигляді тегів, редагувати метадані файла, видаляти та завантажувати файли.

Рисунок 3.3 – Діаграма прецедентів: робота з файлами

Користувач також має можливість пошуку файлів за кількома мітками. Також він має можливості роботи з метаданими, зображені на рис 3.4, які визначаються такими операціями:

- Переглядання тегів
- Додавання тегів
- Видалення тегів

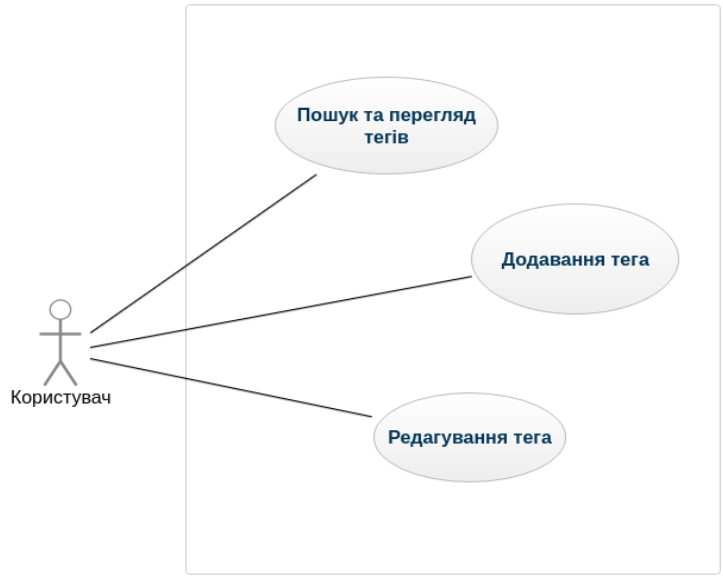

Рисунок 3.4 – Діаграма прецедентів: робота з метаданими

3.4 Структурні особливості взаємодії об’єктів. Діаграми послідовностей.

Діаграма послідовності – в UML, відображає взаємодії об’єктів, підпорядкованих за часом. Зокрема, такі діаграми відображають задіяні сутності та послідовність відправлених ними повідомлень.

На рисунку 3.5 зображена діаграма послідовності додавання користувачем файла: спочатку дані, які зберігаються через інтерфейс користувача поступають на проміжний сервер, який додає до об’єкта основні метадані системи і іншу інформацію, корисну для емуляції файлової системи. Отже, дані перехоплюються під час процесу зберігання, їх вміст сканується та аналізується, і вся згенерована інформація додається до об'єкта у вигляді метаданих.

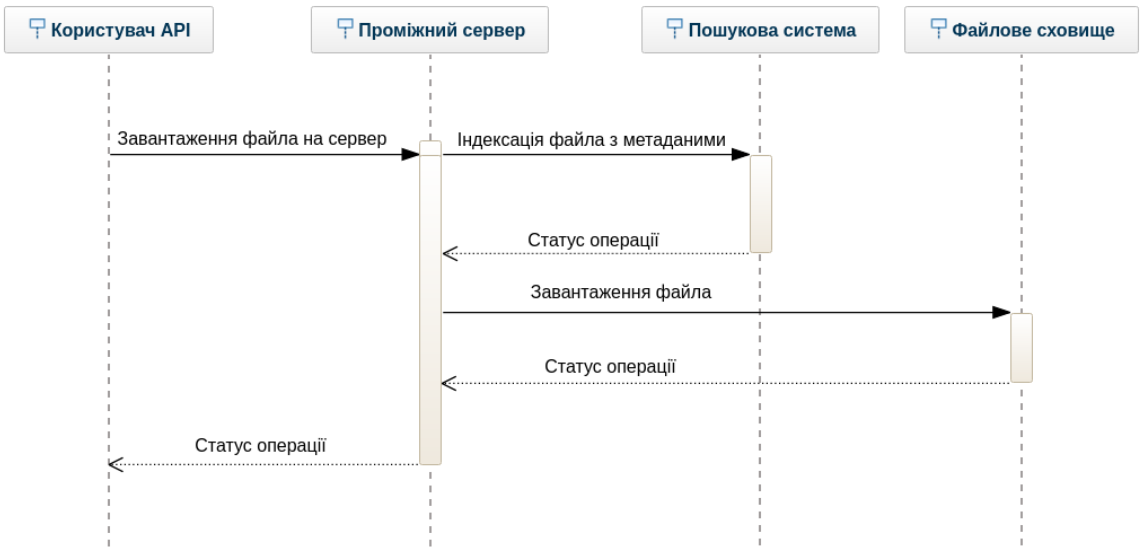

Рисунок 3.5 – Процес завантаження файлу на сервер

На рисунку 3.6 зображена діаграма послідовності пошуку та завантаження файлу: спочатку користувач, відправляє пошуковий запит який поступає на проміжний сервер, який робить валідацію запита та перенаправляє запит до пошукової системи. Потім відповідь повертається клієнту, який робить запит на завантаження файла. Проміжний сервер направляє запит до файлового

сховища, який повертає об’єкт який складається з файлу з метаданими. Отриманий результат повертається користувачу.

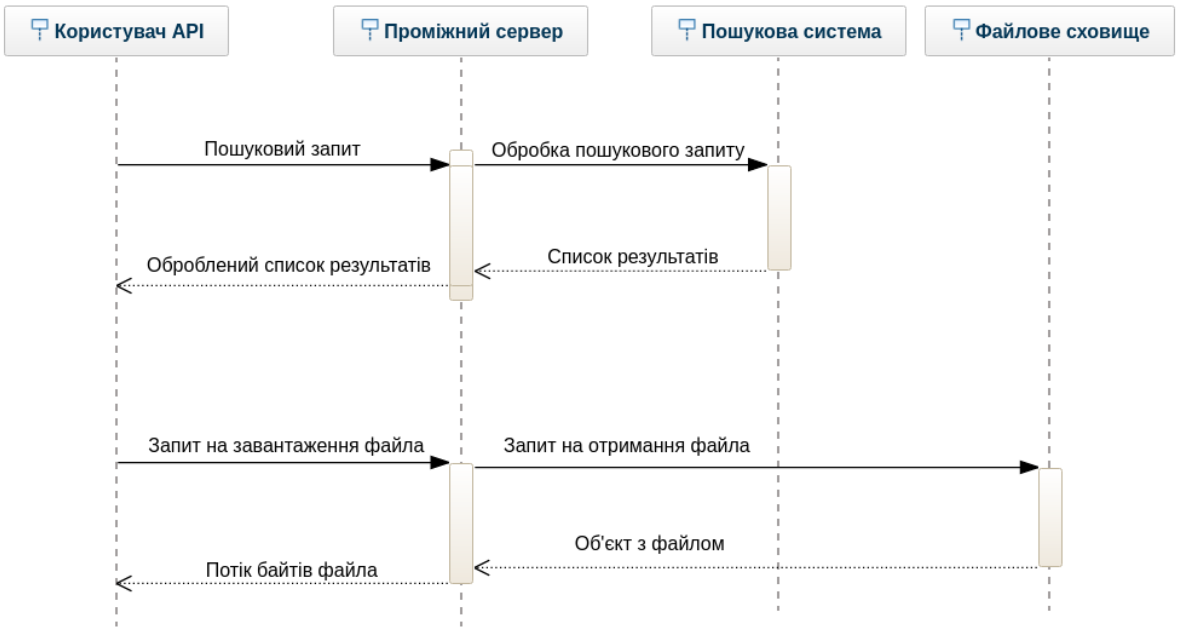

Рисунок 3.6 – Пошук та завантаження файлу

4.4 Висновки

В даному розділі було спроектовано архітектуру програмної системи. Була проведена об'єктна декомпозиція та представлення взаємозв’язків між підсистемами. Створені діаграми прецедентів, що відображають основні функціональні аспекти системи та відносини розширення між акторами системи. Діаграми послідовностей представляють порядок взаємодії сутностей системи та їх обмін повідомленнями.

4. ВИБІР ІНСТРУМЕНТАЛЬНОЇ БАЗИ ПРОГРАМНОЇ СИСТЕМИ

4.1 Опис файлового сховища.

Для реалізації файлового сховища було обрано об’єктне сховище OpenStack Object Swift, яке являє собою розподілене, стійке до поломок і високонадійне сховище об'єктів.

Swift функціонує як розподілена, доступна через API-інтерфейс платформа зберігання, яку можна інтегрувати безпосередньо в додатки або використовувати для зберігання образів віртуальних машин, резервних копій і архівів, а також менш великих файлів, таких як фотографії та електронні листи.

В основі об'єктного сховища лежать дві основні концепції - об'єкти і контейнери.

Об'єкт - це основна сутність зберігання. Він містить контент і всі можливі додаткові метадані, асоційовані з файлами, що зберігаються в системі OpenStack Object Storage. Дані зберігаються в стислому і в незашифрованому вигляді і складаються з імені об'єкта, його контейнера і, можливо, метаданих, представлених у формі пар "ключ / значення". Об'єкти розподілені між кількома дисками в масштабі всього центру обробки даних, ніж Swift гарантує реплікацію даних і цілісність даних. Розподілена організація дозволяє застосовувати недорогі масові апаратні засоби, а також покращує такі характеристики, як масштабованість, надмірність і довговічність.

Контейнер подібний папці Windows® в тому сенсі, що він є відсіком сховища для розміщення набору файлів. Контейнери не можуть бути вкладені одна в одну, проте у орендаря є можливість створити необмежену кількість контейнерів. Об'єкти повинні зберігатися в контейнері, тому для того, щоб

використовувати сховище об'єктів Object Storage, необхідно не менше одного контейнера.

На відміну від традиційного файлового сервера, сховище Swift розподілено між кількома системами. Swift автоматично зберігає резервні копії кожного об'єкта, щоб забезпечити максимальну ступінь доступності і масштабованості. Управління версіями об'єктів забезпечує додатковий захист від ненавмисного втрати даних і перезапису.

Архітектура Swift складається з трьох компонентів - сервери, процеси і кільця.

4.1.1 Сервери

Архітектура Swift є розподіленою, що запобігає виникненню єдиної точки відмови, а також забезпечує горизонтальне масштабування. До її складу входять такі чотири сервера:

1. Проксі-сервер
2. Сервери об'єктів
3. Сервери контейнерів
4. Сервери облікових записів

Проксі-сервер являє уніфікований інтерфейс для інших елементів архітектури OpenStack Object Storage. Він приймає запити на створення контейнерів, на завантаження файлів або на зміну метаданих, а також може надавати списки контейнерів або збережені файли. При отриманні запиту цей сервер визначає місце розташування облікового запису, контейнера або об'єкта в кільці, а потім перенаправляє запит на відповідний сервер.

Сервер об'єктів - це простий сервер, здатний завантажувати, змінювати і вилучати об'єкти (зазвичай це файли), що зберігаються на пристроях, якими він управляє. Об'єкти зберігаються в локальній файловій системі з використанням

розширених атрибутів, які можуть містити додаткові метадані. Маршрут об'єкта базується на хеш-функції імені об'єкта і на мітки часу.

Сервер контейнерів по суті є каталогом об'єктів. Він здійснює прив'язку об'єктів до певного контейнера і за запитом надає списки контейнерів. З метою резервування ці списки реплицируются в масштабі всього кластера.

Сервер облікових записів управляє обліковими записами за допомогою сервісів сховища об'єктів. Він функціонує подібно сервера контейнерів - в тому сенсі, що він надає списки, в яких в цьому випадку перераховані контейнери, прив'язані до заданої облікового запису.

4.1.2 Процеси

Сховищем даних управляють кілька виконуваних за розкладом службових процесів, до яких відносяться сервіси реплікації, а також процеси аудиту і процеси відновлення.

Сервіси реплікації мають величезне значення: вони гарантують узгодженість і доступність в масштабі всього кластера. Однією з основних цілей сховища об'єктів є реалізація з його допомогою розподіленого зберігання, тому платформа OpenStack повинна гарантувати узгоджене стан в умовах випадкових помилок, таких як збої живлення або відмови компонентів. Вона вирішує цю задачу за допомогою регулярного порівняння локальних даних з їх дистанційно розташованими копіями і підтвердження того, що всі репліки містять останню версію.

Щоб звести до мінімуму обсяг мережевого трафіку, необхідного для цього порівняння, сервіси створюють хеш для кожної підсекції розділу і порівнюють ці списки. При реплікації контейнерів і облікових записів також використовуються хеш-кодування, але вони доповнюються загальними найвищими оцінками (high-water mark). Реальні поновлення здійснюються за

принципом проштовхування, в загальному випадку - за допомогою використання rsync для копіювання об'єктів, контейнерів та облікових записів.

Крім того, реплікатор виконує складання сміття, забезпечуючи примусове узгоджене видалення супутніх даних в разі видалення будь-яких об'єктів, контейнерів або облікових записів. Після видалення будь-якого об'єкта і т. Д. Система позначає його останню версію "надгробним каменем" (tombstone), що є сигналом реплікатора про необхідність видалення відповідного елемента на всіх реплікованих вузлах.

Однак навіть найкраща схема реплікації ефективна лише настільки, наскільки ефективні реалізують її компоненти. Виробничі середовища повинні бути в змозі справлятися зі збоями, незалежно від того, чи є конкретний збій результатом відмови апаратних засобів або програмних помилок або просто наслідком недостатньої ємності. У середовищі Swift це завдання вирішується за допомогою процесів оновлення і аудиту.

Процеси відновлення відповідають за забезпечення цілісності системи при виникненні збою. Коли сервіси реплікації стикаються з проблемою і не в змозі оновити контейнер або обліковий запис, настає період неузгодженості, під час якого об'єкт існує в сховище, але не перераховано на всіх серверах контейнерів або серверах облікових записів. В цьому випадку система поміщає відповідне оновлення в чергу в локальній файловій системі, після чого процес оновлення регулярно робить спробу поновлення.

Процеси аудиту забезпечують додатковий рівень захисту від неузгодженості. Вони регулярно сканують локальний репозитарій на предмет верифікації цілісності облікових записів, контейнерів та об'єктів. У разі виявлення пошкодження вони ізолюють уражену елемент і замінюють його копією з іншого репліки. У разі виявлення неузгодженості, яку вони не в змозі усунути (наприклад, якщо об'єкт не належать до жодного з контейнерів), вони записують помилку в журнальний файл.

4.1.3 Кільця

Користувачі та інші проекти OpenStack звертаються до збережених сутностей по їх логічним іменам, проте в кінцевому підсумку все запити (як з читання, так і по запису) повинні відображатися на конкретну фізичну розташування. Для цього проксі-сервер і забезпечують процеси, включаючи сервіси реплікації, повинні вміти відображати логічні імена на фізичні місцеположення. Це відображення зветься кільце (ring). Облікові записи, контейнери і об'єкти зіставляються з окремими кільцями. Кільце описує це відображення в термінах пристроїв, розділів, реплік і зон.

У цьому контексті термін розділ (partition) відноситься до логічних підмножини контенту, що зберігається в кільці. Рекомендується виділяти по 100 розділів для кожного бере участь пристрою. Ці розділи розподіляються рівномірно між усіма пристроями, приписаними до компоненту OpenStack Object Storage. Якщо в кластері використовуються диски різних розмірів, також можна призначати вагові коефіцієнти для вирівнювання розподілу розділів між пристроями.

За замовчуванням кожен розділ реплицируется три рази. Можна використовувати більшу кількість реплік (replica) для оптимізації доступності, проте очевидно, що при цьому також збільшиться споживання ресурсів зберігання. Кільце також вказує, які пристрої слід використовувати при передачі управління в сценаріях збою і як перерозподіляти розділи при додаванні пристроїв в кластер або при видаленні їх з кластера.

Останній елемент відображення за допомогою кільця - це зона (zone), яка використовується для включення / відключення т. Н. "Близькості даних" (data affinity). Зона може задавати пристрій зберігання даних, фізичний сервер або місце розташування (стійка, прохід або центр обробки даних). Зона - це логічна концепція, яку користувачі можуть застосовувати у відповідності зі своїми

потребами, проте зазвичай вона відображає такі фізичні елементи, як місце розташування, джерело живлення і підключення до мережі.

4.2 Опис пошукової системи.

Для розгортання пошукової системи було обрано Elasticsearch - пошуковий сервер, розроблений на базі Lucene. Надає розподілений, мультиарендний повнотекстовий пошуковий рушій з HTTP веб-інтерфейсом і підтримкою безсхемних JSON документів.

Elasticsearch може використовуватись для індексування та пошуку будь-яких типів документів. Він надає масштабовний пошук, має пошук близький до реального часу і підтримку мультиарендності.

Elasticsearch має можливість розподілення, індекси можуть бути розділені по шардах, при чому кожен шард може мати нуль чи більше реплік. Кожен вузол містить один чи більше шардів і діє як координатор делегування операцій на потрібний шард. Балансування та маршрутизація виконується автоматично.

Elasticsearch, підходить таким підприємствам, які ще не мають пошукову систему і не мають можливостей вкладати багато ресурсів на її інтеграцію, або які мають справу з великими обсягами даних і потребують легких засобів масштабування і захисту пошукових індексів.

4.3 Опис фреймворка для створення додатку проміжного сервера

Ruby on Rails – фреймворк, написаний на мові програмування Ruby, тобто програмне забезпечення, що полегшує розробку та об’єднання декількох окремих складових проекту (наприклад, аутентифікація та авторизація користувачів або каталог статей в блозі).

Фреймворк, на відміну від CMS (система керування вмістом), яку може розгорнути та налаштувати навіть не-програміст, потребує проектування та розробки кваліфікованими спеціалістами. Але на ньому зручніше та швидше створювати проекти, які зовсім відрізняються функціоналом від типового сайту. А до веб-студій та агентств нечасто приходять за повністю типовими сайтами, оскільки замовники часто змінюють поведінку “на льоту”.

Основною перевагою мови програмування Ruby та фреймворку Ruby on Rails є швидкість розробки. На практиці швідкість розробки проектів на RoR вище на 30-40% по відношенню до інших мов програмування або фреймворків. Такий приріст швидкості розробки пояснюється широким набором готових до роботи «з коробки» інструментів RoR, можливістю використовувати готові бібліотеки інших розробників та, звичайно, зручністю програмування на Ruby.

Крім цього, на відміну від інших фреймворків, до складу RoR входять ефективні засоби для автоматизованого тестування, що прискорює перехід проекту від стадії “програму написано” до стадії “програма працює без помилок”. Цей перехід майже завжди займає найбільшу кількість часу під час реалізації майже будь-яких проектів. Також варто відмітити, що Ruby on Rails забезпечує кращу безпеку для додатків. Під час використання інструментів RoR виключені SQL-ін’єкції та XSS-атаки, всі вхідні параметри екрануються за замовчанням, вихідні змінні в шаблонах також екрануються. У розробника майже немає шансів допустити помилку безпеки.

Деякі розробники, що недостатньо добре знайомі з цією технологією, чомусь вважають, що інтернет-проекти на RoR погано масштабуються. Як приклад майже всі розробники наводять Twitter, який в свій час відмовився від Rails через якісь внутрішні причини. Але треба звернути увагу на більш відомі проекти, як Kickstarter, Groupon або Basecamp – всі ці проекти написані з використанням Rails без проблем з масштабуванням. В будь-якому випадку, проблеми продуктивності будь-якого проекту, - це не проблеми помилкового вибору платформи чи мови програмування. Найчастіше ці проблеми були

викликані помилками під час проектування архітектури проекту, кешуванням даних або неоптимальним вибором СУБД.

4.4 Висновки

В даному розділі розглянуті основні переваги обраних технологій для реалізації прототипу системи та їх порівняльна характеристика з іншими альтернативами.

Розгортання файлового сховища буде відбуватися з використанням об’єктного розподіленого сховища Openstack Swift. Для розгортання пошукової системи обрано Elasticsearch. Додаток на проміжному сервері буде написаний за допомогою фреймворку Ruby on Rails.

5. РЕЗУЛЬТАТИ ДОСЛІДЖЕНЬ

В ході вивчення предметної області були сформульовані вимоги до інструменту зберігання корпоративних даних, що дозволило спроектувати архітектуру програмної системи. Рішення, що реалізують даний архітектурний підхід підходять невеликим підприємствам які:

• не хочуть порушувати конфіденційність даннях.

• хочуть мати механізми захисту даних від апаратних збоїв.

• хочуть просте в розгортанні і підтримки рішення.

• не мають ресурсів для покупки дорогих рішень від спеціалізуються на цій області постачальників програмного забезпечення.

• які працюють з потенційно збільшуються обсягами даних і потребують легко масштабованому горизонтально вирішенні.

Такий програмний продукт дозволяє розгорнути сховище без великих вимог до апаратного забезпечення і не використовує зовнішні сервіси. Це призводить до того, що при розгортанні системи локально в межах підприємства безпеку даних підвищується. Компоненти вибрані для програмного прототипу дозволяють захищати дані механізмами надмірності, які не залежать від апаратного забезпечення і прості в розгортанні.

Також дана програмна система має функціонал додавання метаданих до файлів, що дозволяє:

• описувати роль файлів в будь-якому контексті

• зібрати метадані окремо і про індексувати, що дозволяє робити складні пошукові запити

• зберігає інформацію про розподілене географічно сховище даних в межах підприємства.

• пошукові індекси складені з метаданих зберігаються в одному місці

Практика виділення метаданих в окреме сховище надає такі можливості:

• оптимізація доступу до даних

• можливість отримання даних з використанням складних пошукових запитів

За допомогою пошуку по метаданих можна структурувати великі обсяги неструктурованих даних.

6. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

Програмний продукт призначено для використання на персональних комп’ютерах під управлінням будь-якої операційної системи.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

- визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

- для кожної функції визначаються повні річні витрати й кількість робочих часів.
- для кожної функції наоснові оцінок попереднього пункту визначається кількісна характеристика джерел витрат.
- після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

6.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки. Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

- програмний продукт повинен функціонувати на персональних комп’ютерах із стандартним набором компонент;
- забезпечувати високу швидкість обробки великих об’ємів даних у реальному часі;
- забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;
- передбачати мінімальні витрати на впровадження програмного продукту.

6.2 Обґрунтування функцій програмного продукту

Головною метою програмного продукту є створення сервісу зюерігання корпоративних файлів. Ґрунтуючись на цьому, виділимо основні функції:

F1 - вибір фреймворку для реалізації сервера;

F2 - вибір сховища;

F3 - вибір середовища розробки;

Кожна з основних функцій може мати кілька варіантів рішення:

для F1:

а) Ruby on Rails;

б) Django;

в) Spring framework;

для F2:

а) об’єктне сховище Openstack Swift;

б) блочне сховище Openstack Cinder;

для F3:

а) середовище розробки JetBrains RubyMine;

б) текстовий редактор Atom;

За розглянутими варіантами будуємо морфологічну карту (рис.4.1). На основі цієї карти побудуємо позитивно-негативну матрицю (табл.4.1).

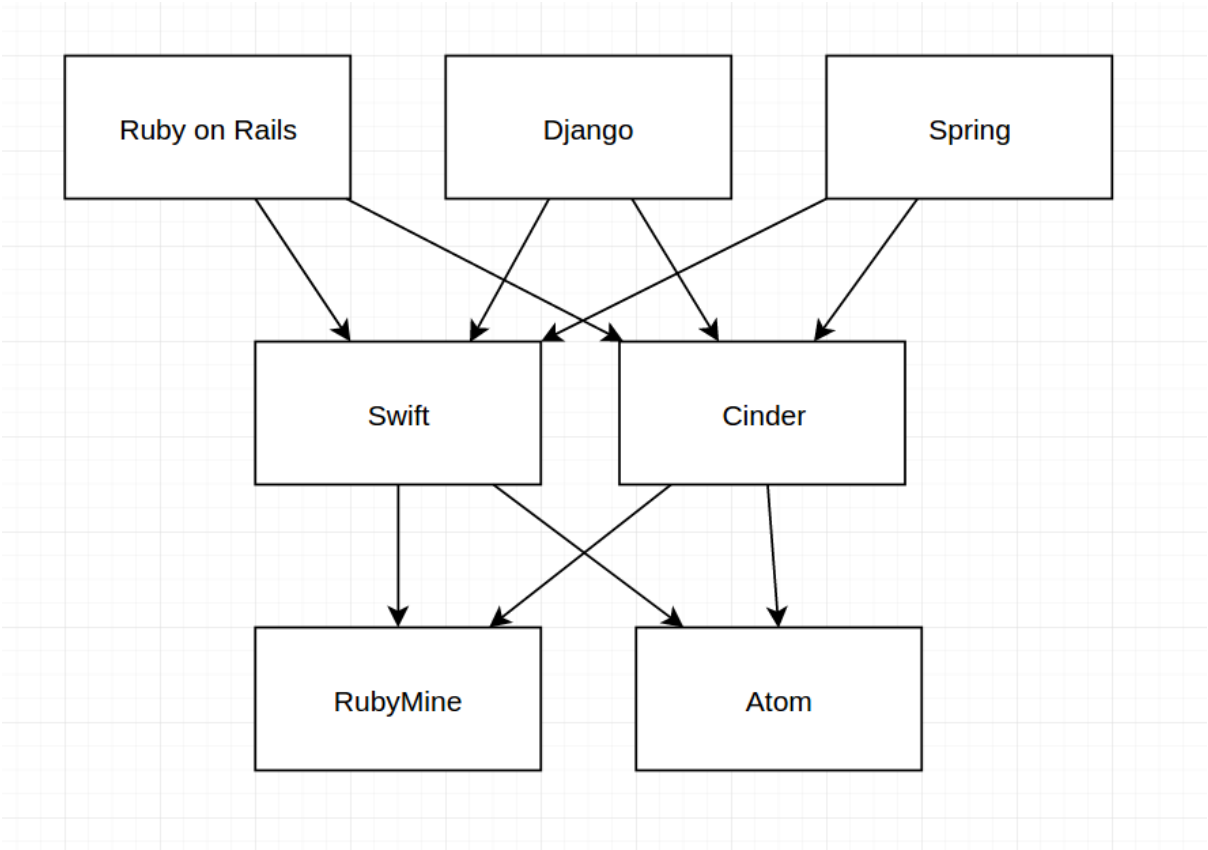

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F1	а)	Легкий у розробці Більш автоматизований	Складність налагодження
	б)	Легкий у розробці	Вимагає більш часу
	в)	Орієнтованість на корпоративну розробку	Висока складність розробки
F2	а)	Можливість масштабування, висока доступність	Цілісність в кінцевому рахунку
	б)	Висока цілісність	Погана масштабованість
F3	а)	Орієнтованість на язик програмування	Платна корпоративна версія
	б)	Легкий, є можливість розширення, швидкий	Відсутність деяких можливостей

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам.

Ці варіанти відзначені у морфологічній карті.

Функція F1:

Оскільки розрахунки проводяться з великими об’ємами вхідних даних, то час виконання програмного коду є дуже необхідним, тому варіант а) має бути відкинутий.

Функція F2:

Вибір СКБД не відіграє велику роль у даному програмному продукту, тому вважаємо варіанти а) та б) гідними розгляду.

Функція F3:

Оскільки, програмний продукт реалізується на мові Java, використовуємо варіант Б як єдиний можливий.

У зв’язку із оглядом основних функцій ПП наведеним вище, будемо розглядати наступні варіанти реалізації:

А: F1а) – F2а) – F3б)

Б: F1а) – F2б) – F3б)

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

6.3 Обґрунтування системи параметрів програмного продукту

6.3.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

X1 – швидкодія мови програмування;

X2 – об’єм пам’яті для збереження даних;

X3 – час обробки даних;

X4 – потенційний об’єм програмного коду.

X1: Відображає швидкодію операцій залежно від обраної мови програмування.

X2: Відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми.

X3: Відображає час, який витрачається на дії.

X4: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

6.3.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 4.2.

Таблиця 4.2 – Основні параметри ПП

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			Гірші	Середні	Кращі
Швидкодія мови програмування	X1	Оп/мс	19000	11000	2000
Об’єм пам’яті для збереження даних	X2	Мб	32	16	8
Час обробки запитів користувача	X3	мс	1000	420	60
Потенційний об’єм програмного коду	X4	кількість строк коду	2000	1500	1000

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

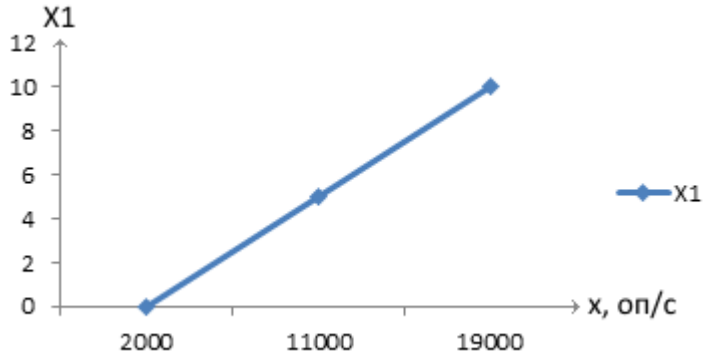

Рисунок 4.2 – X1, швидкодія мови програмування

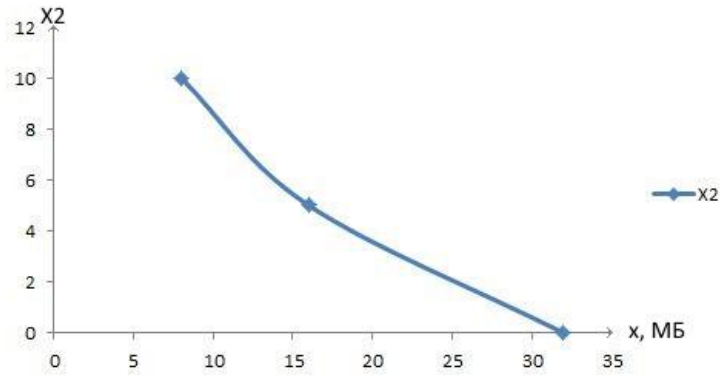

Рисунок 4.3 – X2, об’єм пам’яті для збереження даних

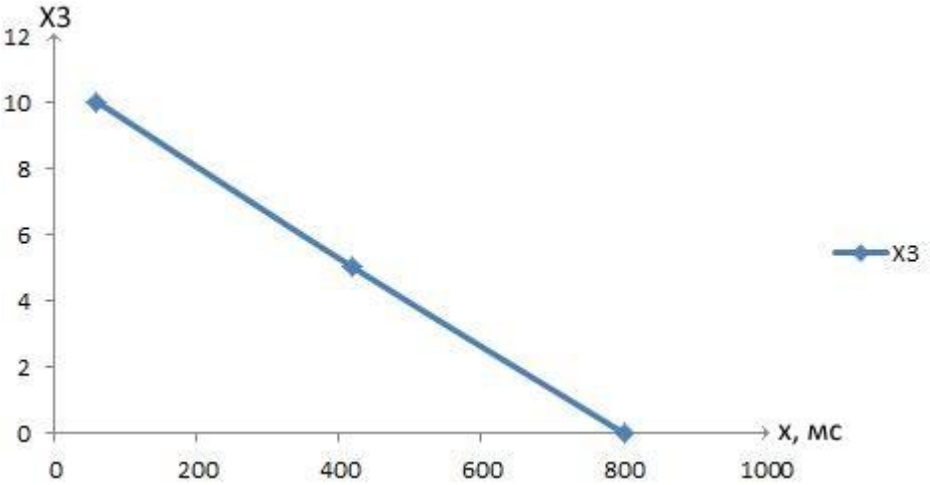

Рисунок 4.4 – X3, час виконання запитів користувача

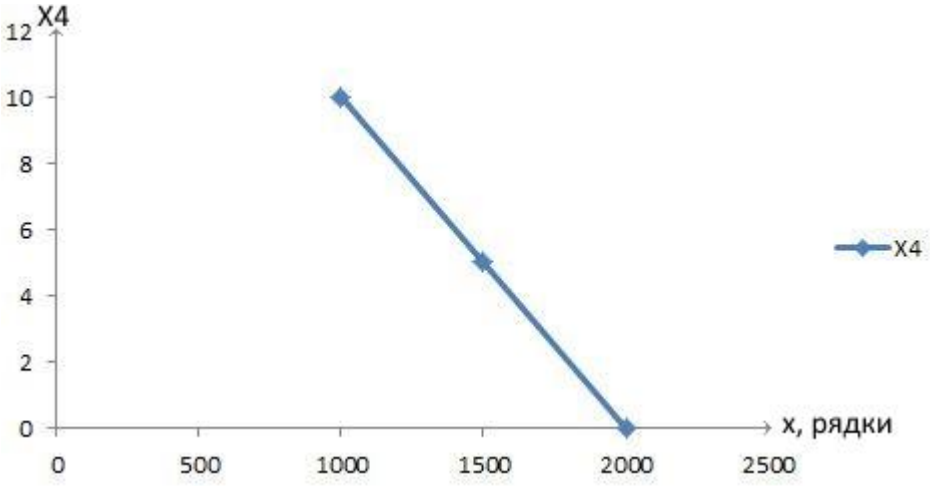

Рисунок 4.5 – X4, потенційний об’єм програмного коду

6.3.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 105 \qquad (4.1)$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 26{,}25 \qquad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T \qquad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 420{,}75 \qquad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 420{,}75}{7^2(5^3 - 5)} = 1{,}03 > W_k = 0{,}67$$

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	4	3	4	4	4	4	4	27	0,75	0,56
X2	Об’єм пам’яті для збереження даних	Мб	4	4	4	3	4	3	3	25	-1,25	1,56
X3	Час обробки запитів користувача	Мс	2	2	1	2	1	2	2	12	-14,25	203,06
X4	Потенційний об’єм програмного коду	кількість строк коду	5	6	6	6	6	6	6	41	14,75	217,56
	Разом		15	15	15	15	15	15	15	105	0	420,75

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Числове значення, що визначає ступінь переваги і–го параметра над j–тим, аіj визначається по формулі:

$$a_{ij} = \begin{cases} 1{,}5 \text{ при } Xi > Xj \\ 1.0 \text{ при } Xi = Xj \\ 0.5 \text{ при } Xi < Xj \end{cases}$$

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	=	>	=	<	=	<	<	<	0,5
X1 і X3	<	<	<	<	<	<	<	<	0,5
X1 і X4	>	>	>	>	>	>	>	>	1,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	>	>	>	>	>	>	>	>	1,5
X3 і X4	>	>	>	>	>	>	>	>	1,5

З отриманих числових оцінок переваги складемо матрицю А=‖ аіj ‖.

Для кожного параметра зробимо розрахунок вагомості Кві за наступними формулами:

$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}$,де $b_i = \sum_{i=1}^{N} a_{ij}$.

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).

На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}$,де $b_i' = \sum_{i=1}^{N} a_{ij} b_j$.

Як видно з таблиці 5.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметриx_i	Параметриx_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{вi}$	b_i^1	$K_{вi}^1$	b_i^2	$K_{вi}^2$
X1	1,0	0,5	0,5	1,5	3,5	0,219	22,25	0,216	100	0,215
X2	1,5	1,0	0,5	1,5	4,5	0,281	27,25	0,282	124,25	0,283
X3	1,5	1,5	1,0	1,5	5,5	0,344	34,25	0,347	156	0,348
X4	0,5	0,5	0,5	1,0	2,5	0,156	14,25	0,155	64,75	0,154
Всього:					16	1	98	1	445	1

6.4 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X1 (швидкість виконання) та X4 (кількість строк) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X3 (ресурсоємність) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 20 % або варіанту б) 80%.

Абсолютне значення параметра X2 (час виконання) також обрано на основі експертних знань: а) 2 с; Б) 5с.

6.5 Розрахунок показників якості варіантів реалізації

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так:

KK(j)=i=1nKвi,jBi,j,

де n – кількість параметрів; К$в$і– коефіцієнт вагомості i–го параметра; B_i – оцінка i–го параметра в балах.

Розрахунок показників рівня якості варіантів реалізації основних функцій ПП:

За даними з таблиці за формулою

КК=КТУ[F1k]+КТУ[F2k]+...+КТУ[Fzk],

визначаємо рівень якості кожного з варіантів:

1. $K_1 = 1.344 + 0.729 + 2.589 + 1.001 = 5.663$
2. $K_2 = 1.344 + 2.264 + 0.661 + 1.001 = 5.27$

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Параметри	Реалізації функцій	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
X1(F1)	А, Б	400	6.25	0,215	1.344
X2(F3)	А	2	2.8	0,283	0.729
	Б	5	8	0,283	2.264
X3(F3)	А	20%	7.44	0,348	2.589
	Б	80%	1.90	0,348	0.661
X4(F2)	А, Б	2000	6.5	0,154	1.001

Як видно з розрахунків, кращим є варіант А, для якого коефіцієнт технічного рівня має найбільше значення.

6.6 Економічний аналіз варіантів розробки програмного продукту

Розрахуємо трудомісткість розробки нашого ПП за різних умов реалізації.

Норми часу беремо відповідно до мов програмування, тому коефіцієнт $К_м$ = 1. Якщо для розробки ПП використовують стандартні модулі чи пакети прикладних програм, стандартні програми, норми часу коригуються за допомогою коефіцієнта $К_{ст}$ = 0,6–0,8. У нашому випадку $К_{ст}$ = 0,7. Якщо розробляють стандартний ПП, норму часу потрібно коригувати за допомогою коефіцієнта $К_{ст.м}$ = 1,2–1,6. У нашому випадку $К_{ст.м}$ = 1,6. У загальному випадку трудомісткість ПП розраховуємо за формулою:

$$Т_О = Т_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М},$$

де $Т_Р$ – трудомісткість розробки ПП;$К_П$ – поправочний коефіцієнт;$К_{СК}$ – коефіцієнт на складність вхідної інформації; $К_М$ – коефіцієнт рівня мови програмування; $К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм; $К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Визначаємо трудомісткість кожного з варіантів реалізації ПП:

$Т_I$ = (59,46 + 65,35 + 127,06) · 8 = 2014,96 людино-годин;

$Т_{II}$ = (59,46 + 65,35 + 178,63) · 8 = 2427,55 людино-годин;

Більш високу трудомісткість має варіант II.

В розробці бере участь один програміст з окладом 15 000 грн. Визначимо зарплату за годину за формулою:

$$С_ч = МТm \cdot tгрн.,$$

де М – місячний оклад працівників;Tm – кількість робочих днів на місяць;t – кількість робочих годин в день.

$С_ч$=1500021·8=89,28 грн.

Тоді, розрахуємо заробітну плату за формулою

$$С_{ЗП} = Сч \cdot Ті \cdot К_Д,$$

де $С_ч$– величина погодинної оплати праці програміста;Ті – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 89.28 \cdot 2014,96 \cdot 1.2 = 215\ 874,75$ грн.

II. $С_{ЗП} = 89.28 \cdot 2427,55 \cdot 1.2 = 260\ 077,99$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0,22 = 215\ 874,75 \cdot 0,22 = 47\ 492,44$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0,22 = 260\ 077,99 \cdot 0,22 = 57\ 217,15$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 15000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 15000 \cdot 0,2 = 36\ 000$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 36\ 000 \cdot (1 + 0.2) = 43\ 200$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0,22 = 43\ 200 \cdot 0,22 = 9\ 504$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 25000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1,15 \cdot 0,25 \cdot 25000 = 7187,5$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1,15 \cdot 25000 \cdot 0,05 = 1437,5$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0,9 = 1706,4$ годин,

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706,4 \cdot 0,156 \cdot 0,4 \cdot 2,0218 = 215,28$ грн,

де $N_С$ – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} \cdot 0.67 = 25000 \cdot 0,67 = 16\,750$ грн.

Тоді, річні експлуатаційні витрати будуть:

$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$

$С_{ЕКС} = 43\,200 + 9\,504 + 7187,5 + 1437,5 + 215,28м + 16\,750 = 85790,33$грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 85790,33 / 1706,4 = 50,27$ грн/час.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$С_М = С_{М\text{-}Г} \cdot Т$

I. $С_М = 50,27 \cdot 2014,96 = 101\ 292$ грн.;

II. $С_М = 50,27 \cdot 2427,55 = 122\ 032,9$ грн.;

Накладні витрати складають 67% від заробітної плати:

$С_Н = С_{ЗП} \cdot 0,67$

I. $С_Н = 215\ 874,75 \cdot 0,67 = 144\ 636,08$ грн.;

II. $С_Н = 260\ 077,99 \cdot 0,67 = 174\ 252,25$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н$

I. $С_{ПП} = 215\ 874,75 + 47\ 492,44 + 101\ 292 + 144\ 636,08 = 509295$ грн.;

II. $С_{ПП} = 260\ 077,99 + 57217,15 + 122032,9 + 174252,25 = 613580,29$ грн.;

6.7 Вибір кращого варіанта програмного продукту техніко-економічного рівня

Коефіцієнт техніко-економічного рівня розраховують за формулою:

- $К_{тер1} = 5,663 / 509295 = 1.1 \cdot 10^{-5}$;
- $К_{тер2} = 5.27 / 613580,29 = 0.85 \cdot 10^{-5}$;

Отже, найбільш ефективним є перший варіант реалізації програми.

6.8 Висновки

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх

важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 1.1 • 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- Фреймворк – Ruby on Rails;
- Створення стилів – OpenStack Swift;
- Програмне середовище - Atom.

ВИСНОВКИ

Метою роботи було проектування архітектури системи зберігання корпоративного контенту та реалізація прототипу програмної системи.

В першу чергу була розглянута проблема організації роботи з даними на підприємствах. Сформульовані характеристики якими повинен володіти інструмент зберігання та обміну файлами для сучасних малих підприємств.

Розглянуті способи зберігання файлів. Можна зробити висновок, що персональні засоби зберігання даних такі як жорсткі диски та флеш накопичувачі не відповідають вимогам працівників. Ця проблема потребує більш комплексного рішення яке інтегрує дані підприємства в єдину систему, яка повинна мати функціонал структурування та захисту великих обсягів даних.

Досліджені відкриті програмні рішення не відповідають сформульованим вимогам. Компанії постачальники пропонують платні програмні системи, спеціалізовані на рішення даної проблеми які не підходять малим підприємствам які не мають достатньої кількості ресурсів.

Також були досліджені технології які використовується для побудови сучасних систем зберігання і обміну даними. Розглянуто основні типи фізичних моделей які використовуються для побудови апаратних сховища даних. Розглянуто використання прийомів віртуалізації для поділу програмних засобів управління розподіленим сховищем і апаратних засобів зберігання даних.

В якості основи для побудови програмного продукту вибрані програмно-визначені сховища оскільки вони реалізують більшу частину необхідного функціоналу з управління розподілом сховищем і добре підходять для заявленої задачі.

Розглянуто три основних типи моделей даних розподілених сховищ. Для реалізації поставленого завдання обрана об'єктна модель даних оскільки вона є

оптимальною для вирішення завдання зберігання великих обсягів неструктурованих даних.

Розглянуто можливості використання метаданих для оптимізації роботи з сховищем. Можливість об'єктне сховище додавати метадані до файлів можна використовувати для структурування даних. Метадані файлів можна зберігати разом у вигляді пошукових індексів що дає можливість обробляти складні пошукові запити.

Як результат було спроектовано архітектуру програмної системи. Була проведена об'єктна декомпозиція та представлення взаємозв’язків між підсистемами. Створені діаграми прецедентів, що відображають основні функціональні аспекти системи та відносини розширення між акторами системи.

Також було проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

В результаті роботи був реалізований прототип програмної системи. Реалізовану програмну систему можна використовувати для вирішення задач зберігання та обміну файлами на малих підприємствах.

ПЕРЕЛІК ПОСИЛАНЬ

1. Kapadia, A. Implementing cloud storage with Openstack swift: Design, implement, and successfully manage your own cloud storage cluster using the popular OpenStack swift software. / Kapadia, A., Varma, S., Rajana, K. // United Kingdom: Packt Publishing, 2014 – pp. 134 – 136.

2. Arnold, J. OpenStack swift: Using, administering, and developing for swift object storage. / Arnold, J. // O'Reilly Media, 2014 – pp. 254 – 256.

3. Gormley, Elasticsearch: The definitive guide. / Gormley, C., Tong, Z. // O'Reilly Media, 2015 – pp. 134 – 138.

4. Toigo, J.W. The holy Grail of enterprise data storage. / Toigo, J.W. // Prentice-Hall, 1999 – pp. 64 – 69.

5. Ali Babar. Guidelines for Building a Private Cloud Infrastructure. / Ali Babar. // IT-Universitetet i København, 2012 – pp. 274 – 276.

6. Girish L S. Building Private Cloud using OpenStack. / Girish L S., Dr. H. S. Guruprasad. // International Journal of Emerging Trends & Technology in Computer Science, 2014 – pp. 134 – 138.

7. Marc Farley. Rethinking Enterprise Storage: A Hybrid Cloud Model. / Marc Farley. // Microsoft Press, 2013 – pp. 24 – 28.

8. Kapadia, Amar, Kris Rajana, Sreedhar Varma. OpenStack Object Storage (Swift) Essentials. / Kapadia, Amar, Kris Rajana, Sreedhar Varma // Packt Publishing, 2015 – pp. 184 – 187.

9. Beach Daniel. Building a Search Server with Elasticsearch. / Beach Daniel. // Packt Publishing, 2015 – pp. 267 – 269.

10. Metz, Sandi. Practical Object-oriented Design in Ruby: An Agile Primer. / Metz, Sandi. // Addison-Wesley, 2016 – pp. 150 – 154.

11. Hartl, Michael. Ruby on Rails Tutorial: Learn Web Development with Rails. / Hartl, Michael. //Addison-Wesley, 2016 – pp. 194 – 196.

12. Richardson, Leonard. RESTful Web APIs. / Richardson, Leonard. // O'Reilly Media, 2013 – pp. 74 – 79.

13. Openstack Swift developer documentation. – Режим доступу: http://docs.openstack.org/developer/swift/. – Дата доступу: 14.04.2016.

14. Elastic search developer documentation. – Режим доступу: https://www.elastic.co/guide/index.html. – Дата доступу: 16.05.2016.

15. Ruby on rails developer documentation. – Режим доступу: http://api.rubyonrails.org/. - Дата доступу: 12.05.2016.

16. Hypertext Transfer Protocol overview. – Режим доступу: https://www.w3.org/Protocols/. – Дата доступу: 18.04.2016.

17. REST - Semantic Web Standards - W3C. – Режим доступу: https://www.w3.org/2001/sw/wiki/REST. – Дата доступу: 6.04.2016.

18. Elastic search developer documentation. – Режим доступу: https://www.elastic.co/guide/index.html. – Дата доступу: 27.05.2016.

19. Принцип MVC в веб программировании. – Режим доступу: http://folkprog.net/printsip-mvc-u-web-programmirovanii/. – Дата доступу: 23.05.2016.