НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислюваної техніки
(повна назва інституту/факультету)

Кафедра автоматики та управління в технічних системах
(повна назва кафедри)

«На правах рукопису»
УДК 004.75

«До захисту допущено»
Завідувач кафедри
_____ О. І. Ролік
(підпис) (ініціали, прізвище)
“___”_____2018р.

Магістерська дисертація

зі спеціальності 126 Інформаційні системи та технології
(код і назва спеціальності)

на тему: Розподілена система збереження даних на основі технології блокчейн
Виконав: студент II курсу, групи ІА-з72мп
(шифр групи)

Озеракін Микита Дмитрович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник доцент, к.т.н., доцент Амонс О. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент доц. каф. АПЕПС, ТЕФ, к.т.н., доцент Крамар Ю. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ – 2018 року

ЗАВДАННЯ

РЕФЕРАТ

Дисертація освітньо-кваліфікаційного рівня “магістр” на тему «Розподілена система збереження даних на основі технології блокчейн»: 134 с., 27 рис., 29 табл., 10 додатків, 38 джерел.

Об'єкт дослідження – система для підтримки процесу віддаленого та розподіленого збереження даних із використанням технології блокчейн за запропонованим алгоритмом консенсусу даних.

Мета роботи – розробка системи збереження даних із використанням технології блокчейн. Проаналізувати існуючі системи збереження даних, запропонувати алгоритм консенсусу даних в розподілених системах, спроектувати систему котра підтримує запропонований алгоритм та реалізація підсистеми розподіленого збереження даних.

Наукова новизна дослідження полягає в тому, що реалізовано проект системи, який надає можливість підтримки розподіленого збереження даних згідно до запропонованого алгоритму та реалізація підсистеми збереження даних.

При розробці системи та її компонентів використовувалися сучасні технології програмування такі як ASP .NET Core MVC та Service Fabric, зокрема реалізація моделі акторів.

Прогнозні припущення про розвиток дослідження – застосування алгоритму консенсусу розподіленого збереження даних у системах які займаються питанням цілісності та віддаленого збереження даних.

АЛГОРИТМ КОНСЕНСУСУ, ЗБЕРЕЖЕННЯ ДАНИХ, PROOF-OF-AGREEMENT, ХМАРНІ СИСТЕМИ, УЗГОДЖЕННЯ РОЗПОДІЛЕНИХ ДАНИХ.

REPORT

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 8
ВСТУП 9
1 ПОНЯТТЯ СИСТЕМИ ЗБЕРЕЖЕННЯ ДАНИХ 11
1.1 Загальні поняття про системи збереження даних 11
1.1.1 Централізована СЗД 15
1.1.2 Розподілена СЗД 20
1.1.3 Децентралізована СЗД 22
1.2 Загальні поняття про технологію блокчейн 24
1.2.1 Proof-of-Work 25
1.2.2 Смарт-контракти 27
1.3 Огляд існуючих рішень 29
1.3.1 Dropbox 29
1.3.2 Storj 30
1.3.3 Sia 32
1.4 Алгоритм консенсусу для систем збереження на основі технології блокчейн 34
2 ПРОЕКТУВАННЯ СИСТЕМИ ЗБЕРЕЖЕННЯ ДАНИХ 38
2.1 Загальна архітектура системи 38
2.2 Архітектура клієнтської частини 39
2.3 Архітектура серверної частини 42
2.4 Комунікація між мікросервісами 45
2.5 Типова архітектура сервісу 46
2.6 Основні функції та ролі в системі 49
2.6.1 Підсистема управління даними 50
2.6.2 Підсистема управління контрактами про рівень доступності даних 51
2.6.3 Підсистема тендера для провайдерів сховищ 52
2.6.4 Підсистема рейтингу для провайдерів сховищ 53
2.6.5 Підсистема забезпечення оплати та виплат 54
2.6.6 Підсистема обміну валют 54

2.7 Безпека даних 54
2.8 Масштабованість 55
2.9 Розгортання та експлуатація 57
2.10 Необхідні характеристики підтримуючого технічного обладнання 59
3 РЕАЛІЗАЦІЯ СИСТЕМИ 61
3.1 Вибір стеку технологій та інструментів розробки 61
3.2 Розробка концептуальної моделі 62
3.3 Проектування схеми бази даних 64
3.4 Реалізація серверної частини 66
3.5 Реалізація API фасаду 67
3.6 Реалізація типової архітектури сервісу 70
3.7 Реалізація клієнтської частини 72
3.8 Організація типової структури модулів для сервісу 74
3.9 Принципи розробки системи 75
4 ТЕХНОЛОГІЧНИЙ РОЗДІЛ 78
4.1 Керівництво користувача 78
4.2 Мануальне тестування 81
4.3 Модульне тестування 85
5 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ 88
5.1 Опис ідеї проекту 88
5.2 Технологічний аудит ідеї проекту 91
5.3 Аналіз ринкових можливостей запуску стартап-проекту 92
5.4 Розроблення ринкової стратегії проекту 99
5.5 Розроблення маркетингової програми стартап-проекту 102
ВИСНОВКИ 107
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 108
Додаток А. Акт про впровадження 111
Додаток Б. Алгоритм консенсусу 112
Додаток В. Способи використання розподілених комп’ютерних ресурсів 117
Додаток Г. Діаграма активності для завантаження файлу 123

Додаток Ґ. Діаграма активності для отримання файлу 124
Додаток Д. Діаграма компонентів 125
Додаток Е. Діаграма прецедентів 126
Додаток Є. Діаграма розгортання 127
Додаток Ж. Концептуальна модель 128
Додаток З. Специфікація сервісу управління файлами 129
Додаток И. Вихідний код реалізації шару доступу до даних 130
Додаток І. Специфікація клієнтської частини 132
Додаток Ї. Вихідний код реєстрації на клієнтській частині 133

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

СЗД – Система збереження даних

ПЗ – програмне забезпечення

DAS – Direct Attached Storage

NAS – Network Attached Storage

SAN – Storage Area Network

PoW – Proof-of-Work

PoA – Proof-of-Agreement

UI – користувацький графічний інтерфейс

RPC – Remote Procedure Call

AMQP – Advanced Message Queuing Protocol

НФ – нормальна форма

API – Application Programming Interface

ВСТУП

Пошуки ефективного застосування технології блокчейн поширюються не тільки на фінансовий сектор і все, що з ним пов'язано, але і на такі сфери, як розподілене зберігання даних. Незважаючи на досить високу ефективність і популярність, проблеми з традиційними, тобто централізованими, сховищами як і раніше існують.

Неодноразово був скомпрометований і такий популярний сервіс як Dropbox. Примітно, що на відміну від багатьох своїх централізованих конкурентів Dropbox не використовує шифрування даних, в той час як кожне розподілене сховище за замовчуванням зберігає зашифровані дані користувача.

На жаль, більшість хмарних сервісів зберігання даних схильна ризикам, що випливають з централізації, і розподілені системи з споконвічно закладеним потенціалом до постійного розширення дійсно можуть бути набагато ефективніше як з точки зору стійкості до цензури, так і з точки зору безпеки в цілому. І саме в сфері розподіленого зберігання даних блокчейн може змінити ситуацію кардинально.

Зв’язок роботи з науковими програмами, темами кафедри.

Результати даної роботи впроваджено в навчальний процес кафедри автоматики та управління в технічних процесах в навчальну дисципліну «Технології розробки програмного забезпечення-2. Технології безперервної імплементації та розгортання програмного забезпечення» (додаток А).

Метою роботи є розробка системи збереження даних із використанням технології блокчейн. Проаналізувати існуючі системи збереження даних, запропонувати алгоритм консенсусу даних в розподілених системах, спроектувати систему котра підтримує запропонований алгоритм та реалізація підсистеми розподіленого збереження даних.

Відповідно до мети роботи необхідно вирішити такі завдання:

- розглянути поняття про системи збереження даних;
- описати устрій спроектованої системи для розподіленого збереження даних;
- описати реалізацію системи;
- навести результати випробовування розробленого рішення;
- розробити стартап-проект.

Об'єкт дослідження – система для підтримки процесу віддаленого та розподіленого збереження даних із використанням технології блокчейн за запропонованим алгоритмом консенсусу даних.

Наукова новизна дослідження полягає в тому, що реалізовано проект системи, який надає можливість підтримки розподіленого збереження даних згідно до запропонованого алгоритму та реалізація підсистеми збереження даних.

Магістерська дисертація складається зі вступу, п’яти розділів основної частини, висновків, списку використаних джерел та додатків.

Апробація результатів роботи.

Основні результати роботи обговорювались на VII Міжнародній науково–практичній конференції з інформаційних систем та технологій – Winter InfoCom 2018, м. Київ, 1 грудня 2018 р. (додаток Б).

Публікації.

За результатами роботи було опубліковано дві наукові праці:

- тези статті у журналі «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення» (випуск 33), 2018 р. (додаток В);
- тези доповіді на VII Міжнародній науково-практичній конференції з інформаційних систем та технологій – Winter InfoCom 2018, м. Київ, 1 грудня 2018 р. (додаток Б).

1 ПОНЯТТЯ СИСТЕМИ ЗБЕРЕЖЕННЯ ДАНИХ

1.1 Загальні поняття про системи збереження даних

Системи збереження даних (СЗД) – це програмно(-апаратне) рішення для організації надійного зберігання інформаційних ресурсів та надання гарантованого доступу до них [1].

Інформаційні ресурси як одиницю даних за рівнем використанням можна поділити на три групи: блоки, файли та об’єкти. Кожні з них мають свої переваги та способи використання [2].

Рівень блоків надає безпосередньо доступ до роботи з самими байтами інформації (eng. «bare metal»). На цьому рівні відсутній концепт «файлів». Існують лише упорядковані блоки даних. Як правило, використання блочного рівня зберігання даних забезпечує найкращу продуктивність за часом, але це досить низький рівень. Наприклад, сервери бази даних часто можуть використовувати системи блочного зберігання. Прикладом використання такого рівня є Storage Area Network (SAN) [3].

На рівні файлів вже маємо доступ до файлової системи. Це найпопулярніший вид зберігання даних – це те, що з чим ми працюємо у повсякденному житті. Працюючи на цьому рівні, ми маємо доступ до файлів та можливість читати та писати в цілому файл або його частину. Файлові системи – це те, що операційні системи використовують на всіх наших персональних комп'ютерах. У середовищах із спільним використанням та зберіганням файлів часто використовується мережевий диск.

Рівень об’єктів – це, мабуть, найменш знайомий тип зберігання для більшості людей. Об’єкт не забезпечує доступ до фізичних блоків даних. Він не надає файловий доступ. Цей рівень забезпечує доступ до цілих об'єктів або блоків даних, і робить це за допомогою API (Application Programming Interface), специфічного для цієї системи [4]. На відміну від рівня файлів, об'єкт взагалі не дозволяє писати в одну частину файлу. Об'єкти можуть оновлюватися лише як одна цілісна одиниця даних. Три найбільш поширених комерційних систем зберігання об'єктів – Amazon S3, EMC Atmos та Rackspace Cloud Files [5]. Використання цього рівня передбачає зберігання інформації

без обмежень за розміром. Найяскравіші способи використання такого рівня є резервні копіїї, архіви та статичний веб-вміст. Однією з основних переваг систем зберігання об'єктів є їх здатність надійно зберігати велику кількість даних за відносно низькою ціною.

Зберігання документів є цікавим випадком для систем зберігання об'єктів. На перший погляд, це, здається, є тим самим випадком використання, як файлів, але системи управління документами вийдуть за рамки рівня файлів, якщо надати політику для кожного документа, яка описує дозволи та час зберігання.

На рівні об'єктів також зручно зберігати великі набори даних. Наукове дослідження збирає величезну кількість даних, оскільки вчені спостерігають за найбільшими та найменшими речами у нашому всесвіті. Ці дані часто неможливо замінити і вони стануть основою для подальших досліджень на наступні десятиліття. Системи зберігання об'єктів дозволяють надійно зберігати цю інформацію економічно-ефективним способом.

Системи зберігання об'єктів також можуть бути використані як основа для інших систем зберігання даних. Наприклад, система зберігання об'єктів може бути використана як основа для системи зберігання блоків. Кожен "блок" в системі зберігання блоків буде представлений як об'єкт в системі зберігання об'єктів. Такі абстракції, що складаються, подібно до цього, дозволять системам блоків виростати в дуже великому масштабі, але вони матимуть певну деградацію за ефективністю. Система блоків, побудована на основі об'єктної системи, матиме набагато більшу затримку при операціях запису або читання, ніж традиційна система зберігання блоків, але при цьому їх кількість необмежена.

Умовно можна виділити декілька класів систем збереження даних такі як вбудовані (eng. built-in), знімні (eng. removable), мережеві та хмарні (eng. cloud).

Перша система збереження ваших файлів буде збережена на вбудованому жорсткому диску комп'ютера. Можна встановити кілька жорстких дисків на одному комп'ютері. Ви можете, наприклад, призначити один конкретний жорсткий диск для цілей архівації. Пам'ятайте, що ваша операційна система та програмне забезпечення також займають місце на жорсткому диску, тому вам потрібно обчислити його, якщо ви хочете оцінити, яка потужність вам знадобиться. Станом на жовтень 2018 р. жорсткі диски до 14 ТБ доступні на комерційній основі.

Найпоширеніші знімні системи зберігання для комп'ютерів включають зовнішні жорсткі диски, USB флеш пам’ять та DVD-диски. Зовнішній жорсткий диск має найбільшу потужність і може бути підключений до комп'ютера через порт USB. Вони доступні в тих самих розмірах, що й звичайні жорсткі диски, а також можна перетворити звичайний жорсткий диск на зовнішній, використовуючи спеціальний корпус. Перевага над іншими знімними системами полягає в тому, що ви можете переміщати великі файли досить швидко, і це в цілому вийде дешевше на кожен гігабайт, ніж менший USB-накопичувач. USB флеш пам’ять є ідеальними для тимчасового зберігання файлів, оскільки вони є невеликими та портативними. Карти пам'яті USB тепер доступні в ємностях до 1 ТБ. DVD-диски є гарним варіантом для архівації, якщо у вас немає надто багато файлів для резервного копіювання або якщо ви хочете надати файли своїм колегам або клієнтам назавжди. DVD-диски досить недорогі, і ви отримуєте їх у записаних та перезаписуваних версіях.

У офісному середовищі з локальною мережею вигідно мати один центральний сервер, який містить усі файли для доступу всіх користувачів. Найпростішим варіантом зберігання мережі є доступ до одного комп'ютера з під однієї мережі та створення загальної папки на ньому. Якщо ви не хочете мати завжди один ввімкнений ПК, щоб мати доступ до збережених даних, маршрутизатор з порту USB є іншим рішенням: підключіть USB-накопичувач або зовнішній жорсткий диск до порту, а файли на ній будуть доступні у мережі. Деякі зовнішні жорсткі диски також мають мережеву підтримку, що дозволяє поділитися ними безпосередньо з мережею, а ви отримуєте накопичувачі на жорсткому диску, які перетворять ваш внутрішній привід на мережеву картку пам'яті. Оптимальним варіантом зберігання мереж є мережеві пристрої зберігання даних (NAS) [6]. Деякі з цих серверів NAS призначені для обміну та архівації файлів у вашій мережі, але інші також можуть працювати як принтер, мультимедіа-стример або навіть система спостереження.

Сервіси онлайн-сховищ є хорошим варіантом зберігання файлів, якщо ваші співробітники працюють на віддалених робочих столах. Є багато сервісів хмарного зберігання в інтернеті, серед яких досить багато, які ви можете безкоштовно використовувати з певними обмеженнями. Більшість цих СЗД беруть платню за простір, який ви використовуєте, і надає можливість автоматичної синхронізації даних на різних

пристроїв. Деякі з них призначені для конкретних цілей, таких як музична бібліотека або система резервного копіювання. Багато програмних пакетів також використовують хмарне обчислення, тому програма залишається на онлайновому сервері, залишаючи більше місця на власному ПК для ваших власних файлів.

За побудовою архітектури системи збереження даних можна поділити на такі як централізовані, децентралізовані та розподілені [7]. На рисунку 1.1 зображено схематично відмінності між цими архітектурами.

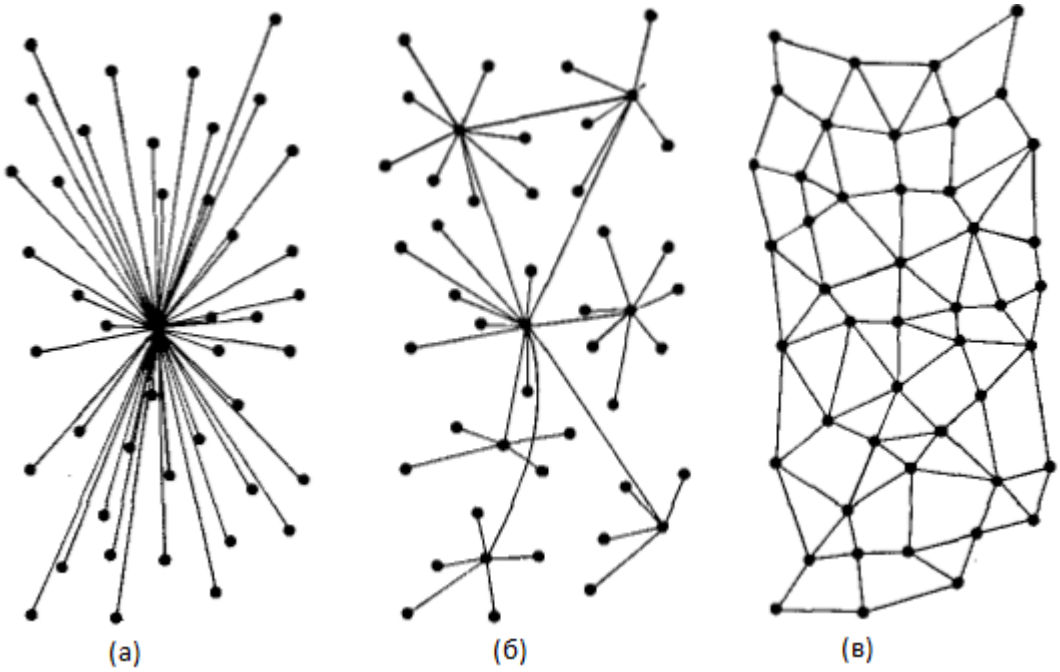


Рисунок 1.1 – Схематичне зображення розташування вузлів при (а)централізованій, (б)децентралізованій та (в)розподіленій архітектурах

Централізована СЗД – це сховище, яке знаходиться, підтримується та зберігається в одному місці. Щоб отримати доступ до даних, вам потрібно підключитися до сервера, який є основним комп'ютером цієї системи. Дані в цій архітектурі зазвичай обробляються основним комп'ютером.

Децентралізована СЗД – це сховище набагато складніше, ніж попередня. Децентралізована система не має єдиного розташування, і, навпаки, фрагменти інформації зберігаються в різних місцях, які всі пов'язані один з одним. Обробка даних у цьому сховищі розподіляється між різними вузлами.

Розподілена СЗД – у цьому сховищі пристрої зберігання, що містять дані, не підключаються до одного головного сервера, і замість цього ці дані можуть бути територіально розташовані на різних пристроях в одному і тому ж місці або поширюватися між мережами взаємопов'язаних серверів. Кожен вузол, що містить інформацію, має свій набір даних та рівні права.

1.1.1 Централізована СЗД

Централізоване зберігання дозволяє консолідацію збережених ресурсів на рівні файлу або блоку. Це дозволяє ефективно використовувати вільний простір, оскільки всі сервери можуть отримати доступ до одного і того ж пулу пам'яті. Управління та резервне копіювання даних також простіше, оскільки вони централізовані та розглядаються як єдине місце для зберігання.

Безпосередньо під’єднане сховище (eng. Direct Attached Storage, DAS) – це комп'ютерне сховище, яке підключено до одного комп'ютера та недоступне іншим комп'ютерам [8]. Жорсткий диск або SSD є звичайною формою сховища з прямим підключенням. На підприємстві індивідуальні дискові накопичувачі на сервері називаються сховищем з прямим підключенням, а також групами дисків, які є зовнішніми для сервера, але безпосередньо приєднані через SCSI, SATA, SAS, FC чи iSCSI.

DAS пристрій не може бути підключений через мережу. Немає з'єднання через Ethernet або FC-комутатори, які підключаються до NAS чи SAN.

Інші типи зберігання, такі як оптичні пристрої та стрічки, технічно є DAS, оскільки вони безпосередньо прикріплені до системи. Проте, коли мова йде про DAS, це, як правило, стосується внутрішнього або зовнішнього первинного або вторинного зберігання у вигляді жорстких дисків та SSD.

DAS може забезпечити користувачів кращою продуктивністю, ніж мережеве сховище, оскільки сервер не робить мережеві виклики. Саме тому підприємства часто обирають DAS для певних типів додатків, які вимагають високої продуктивності. Наприклад, Microsoft рекомендує, для Exchange Server використовувати DAS.

У минулому DAS часто критикували як неефективний спосіб керування корпоративним сховищем, оскільки його неможливо легко розділити, і це не надає відмово стійкості серверу. Оскільки віртуалізація набирає велику популярність, переваги, які пропонує DAS, знову стали більш вагомими.

На відміну від централізованого та мережевого спільного сховища, такого як NAS або SAN – де загальна ємність є пулом і розподіляється між серверами за допомогою швидкого мережного підключення, безпосередньо підключена пам'ять належить одному серверу. Це означає, що підключення (connectivity) та масштабованість обмежуються кількістю слотів розширення на сервері. Розмір корпусу DAS також обмежує об'єм зберігання. Ці недоліки продовжують обмежувати цей вид зберігання даних. Наприклад, для обміну даними з DAS зазвичай обмежуються невеликою кількістю портів або хостів.

Однак DAS є менш дорогим, ніж SAN або NAS, і простіше розгортати його під час безпосереднього підключення до сервера. Просто придбайте нові диски, підключіть їх до сервера, і ви потенційно збільшили обсяг пам'яті на кілька терабайт. Це зробило DAS практичним вибором для багатьох малих та середніх підприємств, де витрати на зберігання є основною статтею витрат.

Більшість фізичних серверів продовжують завантажуватися з DAS. Швидкість SSD, зокрема, робить завантаження локально за допомогою безпосередньо підключеного сховища більш швидшим, ніж SAN. Завдяки локальним SSD, для завантаження або перезавантаження фізичного сервера може знадобитися лише кілька секунд. Це корисна функція, коли фізичний сервер хостить віртуальні машини (VM), які потрібно швидко розгорнути у разі аварії або після запланованого простою або технічного обслуговування.

Крім того, DAS часто не вистачає багатьох функцій керування сховищем, які є для пристроїв NAS та SAN, наприклад віддаленої реплікації та резервних копій.

Мережеве сховище (eng. Network-attached storage, NAS) – це спеціальний файловий накопичувач, який дозволяє декільком користувачам отримувати дані з централізованого сховища даних. Користувачі в локальній мережі (LAN) отримують доступ до спільного сховища через стандартне з'єднання Ethernet. Пристрої NAS, як правило, не мають клавіатури або дисплея, а налаштовуються та управляються за допомогою браузера.

Кожен NAS розташований в локальній мережі як незалежний мережевий вузол, визначений власною унікальною IP-адресою.

Найбільш характерним властивостями для NAS є легкість доступу, висока продуктивність і досить низька вартість. Пристрої NAS забезпечують інфраструктуру для консолідації даних та для виконання завдань адміністрування, таких як архівація та резервне копіювання.

NAS і SAN є двома основними типами мережевого сховища. NAS обробляє неструктуровані дані, такі як аудіо, відео, веб-сайти, текстові файли, документи Microsoft Office і т.д. SAN призначені в першу чергу для зберігання даних на рівні блоків, наприклад, для баз даних, також відомих як структуровані дані.

NAS надає користувачам можливість більш ефективно співпрацювати та обмінюватися даними, особливо робочими групами, які знаходяться віддалено. NAS підключається до бездротового маршрутизатора, що облегшує для розподілених робочих середовищ для доступу до файлів і папок з будь-якого пристрою, підключеного до мережі. Організації зазвичай використовують середовище NAS як основу для особистої або приватної хмари.

Продукти NAS призначені для використання на великих підприємствах, а також для домашніх офісів або малого бізнесу. Пристрої зазвичай містять принаймні два відсіки для приводу, хоча для некритичних даних доступні одиночні системи. Приладдя Enterprise NAS розроблено з більш високотехнологічними функціями даних, що допомагають керуванню зберіганням даних, і зазвичай постачається принаймні з чотирма дисковими відділеннями.

До NAS, підприємствам доводилося налаштовувати та управляти сотнями або навіть тисячами дискретними файловими серверами. Щоб розширити об'єм зберігання, пристрої NAS оснащені більшою кількістю дисків, відомими як scale-up NAS, або об'єднані у кластер для масштабування сховища.

Крім того, більшість постачальників NAS співпрацюють із постачальниками хмарних систем зберігання, щоб надати клієнтам гнучкість для резервного копіювання.

Мережеве сховище залежить від жорстких дисків (жорстких дисків) для надання даних. При значному навантаженні коли багато користувачів одночасно запитують дані може виникати боротьба за один і той самий ресурс (eng. I/O contention).

Тип HDD, вибраного для NAS, продиктований програмами, які потрібно підтримувати. Спільне використання електронних таблиць Microsoft Excel або документів Word з колегами є звичайним завданням (routine task), таке ж як і періодичне резервне копіювання даних. І навпаки, використання NAS для обробки великих об'ємів мультимедійних файлів потокового вмісту потребує більших дисків потужності, більше пам'яті та більш потужної обробки даних в мережі.

У домашніх умовах люди часто використовують систему NAS для зберігання та обслуговування мультимедійних файлів або для автоматизації процесу резервного копіювання. Домовласники можуть покладатися на NAS, щоб керувати сховищем для інтелектуальних телевізорів, систем безпеки та інших IoT-компонентів.

В умовах підприємства, NAS-масив може бути використаний в резервних цілях таких як архівація та аварійне відновлення. Якщо пристрій NAS має режим сервера, він також може служити електронною поштою, мультимедійними файлами, базами даних і т.д.

Деякі продукти NAS містять достатньо дисків для підтримки RAID (резервний набір незалежних дисків). Ця конфігурація зберігання перетворює декілька жорстких дисків в один логічний блок для підвищення продуктивності та високої доступності.

Мережа зберігання (eng. Storage area network, SAN) – це виділена високошвидкісна мережа або підмережа, яка зв’язує і надає спільні пули пристроїв зберігання даних на декількох серверах.

SAN переміщує ресурси зберігання з загальної мережі яка призначена для користувачів та реорганізує їх у незалежну високопродуктивну мережу. Це дає змогу кожному серверу отримати доступ до спільного сховища, так само якщо б це був диск, безпосередньо приєднаний до сервера. Коли хост хоче отримати доступ до пристрою зберігання даних у SAN, він надсилає запит на доступ до пристрою на рівні блоків.

Мережа зберігання даних зазвичай збирається за допомогою трьох основних компонентів: кабелів, адаптерів вузлів шини та комутаторів, прикріплених до масивів

зберігання та серверів. Кожна система комутації та зберігання в SAN повинна бути взаємопов'язаною, а фізичні взаємозв'язки повинні підтримувати рівні пропускної здатності, які дозволяють адекватно реагувати на пікові активності. IT-адміністратори централізовано управляють мережами зберігання.

SAN Fibre Channel (FC) мають репутацію дорогих та складних для керування. Інтерфейс iSCSI на основі Ethernet зменшив ці проблеми, інкапсулюючи команди SCSI в IP-пакети, які не потребують з'єднання FC.

Поява iSCSI означає, що замість навчання, побудови та управління двома мережами – локальною мережею Ethernet (LAN) для комунікації з користувачами та FC SAN для зберігання – організація може використовувати існуючі знання та інфраструктуру для LAN та SAN. Це особливо корисний підхід у малих та середніх підприємствах, які можуть не мати коштів або досвіду для підтримки Fibre Channel SAN.

Організації використовують SAN для розподілених програм, які потребують високої потужності локальної мережі. SAN удосконалюють доступність додатків за допомогою декількох шляхів передачі даних. Вони також можуть покращити продуктивність додатків, оскільки вони дозволяють IT-адміністраторам відключити функції зберігання та відокремити мережі.

Крім того, SAN допомагає підвищити ефективність та використання сховища, оскільки вони дозволяють адміністраторам консолідувати ресурси та забезпечувати багаторівневий спосіб зберігання. SAN також покращують захист даних та безпеку.

Головним достоїнством використання SAN є те, що дані зберігаються як сукупність ресурсів, які адміністратор може централізовано керувати та розподіляти за необхідністю. SAN також мають високу масштабованість, оскільки за потреби може бути додана ємність без простою.

Основними недоліками SAN є вартість та складність. Апаратне забезпечення SAN має тенденцію бути дорогим, і для побудови та управління SAN потребує висококваліфікованих фахівців.

1.1.2 Розподілена СЗД

Розподілена система зберігання даних може відноситись до будь-якого з трьох рівнів зберігання: блок, файл та об'єкт. У випадку систем зберігання на рівні блоків, як правило, відноситься до однієї системи зберігання в тісній географічній зоні, зазвичай розташованої в одному центрі обробки даних, оскільки вимоги до продуктивності є дуже високими. Неможливо зробити розподілену систему зберігання даних, що забезпечує високу продуктивність на великих відстанях, просто тому, що закони фізики не дозволяють це робити – це займає багато часу для синхронізації системи, яка поширюється на 3 континентах.

У випадку систем зберігання об'єктів – вони можуть бути як в одному місці, так і в інших місцях, і тут географічно може працювати розподілена система зберігання, оскільки вимоги до продуктивності не такі ж високі, як і для блочного рівня зберігання.

Системи розподіленого зберігання використовують стандартні сервери, які наразі досить потужні (у процесорі, оперативній пам'яті, а також мережевих з'єднань/інтерфейсах), таким чином, вони дозволяють зберігати дані як програмне забезпечення, подібно до баз даних, операційних систем, віртуалізації та всіх інших програм. Для цього більше не потрібна спеціалізована коробка для обробки функції зберігання. Використання серверу для зберігання даних, окрім додатків, є головним проривом – це означає спрощення IT-стеку та створення єдиного блоку серверів для центру обробки даних – серверів, підключених до однорангових мереж. Це дозволяє масштабувати, додавши більше серверів і таким чином підвищуючи продуктивність і продуктивність лінійно. Це також означає, що ви можете мати сервери, які подвоюються як накопичувачі та обчислювальні вузли, але також дозволяє обчислення чи збереження окремо на різних вузлах.

Поглянувши на спеціалізований масив пам’яті, можна побачити, що це по суті сервер – він має процесор, оперативну пам'ять, мережеві інтерфейси та диски. Однак це "заблокований" сервер, який може використовуватися лише для збереження даних. Для того, щоб мати швидку систему зберігання, потрібна висококласна коробка для зберігання, яка має високу вартість. Також навіть у більшості систем, коли ви додаєте

більше цих «ящиків» для зберігання до СЗД, навіть у наші дні це не збільшує продуктивність всієї системи, оскільки весь трафік проходить через "головний вузол" або головний сервер, який виступає як вузол керування. Це стає вузьким місцем.

Розподілена система дозволяє робити горизонтальне масштабування. Розглянемо на прикладі єдиного сервера баз даних. Єдиним способом обробки більшого трафіку буде оновлення апаратного забезпечення, на якому запущена база даних. Це називається масштабування вертикально.

Масштабування вертикально добре працює до певного часу, але після певного моменту можна помітити, що навіть найкращого обладнання недостатньо для достатнього трафіку, не кажучи вже про неприпустимість хостингу.

Горизонтальне масштабування просто означає збільшення кількості комп'ютерів, а не оновлення апаратного забезпечення одного. Воно стає набагато дешевше після певного порогу. Це значно дешевше, ніж вертикальне масштабування після певного порогу, але це не єдина перевага.

Вертикальне масштабування може лише покращити продуктивність до найновіших можливостей апаратного забезпечення. Ці можливості виявилися недостатніми для технологічних компаній із середніми та великими навантаженнями. Найкраще, що стосується горизонтального масштабування, є те, що немає обмежень на те, скільки разів масштабуватися – коли починається деградація за потужністю, ви просто додаєте іншу машину – це потенційно можна робити до нескінченності.

Просте масштабування не є єдиною перевагою, яку можна отримати від розподілених систем. Відмовостійкість та низькі затримки також не менш важливі.

Відмовостійкість – кластер з десяти машин у двох центрах обробки даних по своїй суті є більш стійким до відмови, ніж для однієї машини. Навіть якщо один центр обробки даних потрапить у вогонь, ваші дані не будуть повністю втрачені.

Низька затримка – час, протягом якого мережевий пакет подорожує світом, фізично обмежений швидкістю світла. Наприклад, найкоротший час для round-trip time (тобто туди і назад) у волоконно-оптичному кабелі між Нью-Йорком та Сіднею становить 160 мс. Розподілені системи дозволяють мати вузол в обох містах, що дозволяє перенаправити трафік до найближчого вузла.

1.1.3 Децентралізована СЗД

Децентралізована мережа складається з багатьох процесів, вузлів кожен з яких має локальний стан, і пропонує цей локальний стан як ресурс для використання іншими пірами в мережі. Стан всієї мережі являє собою поєднання всіх цих локальних станів, а також дії багатьох рівних рівнів, що розділяють частину своїх ресурсів, це те, що дозволяє загальному сервісу утворюватися з безлічі дрібних взаємодіючих частин (наприклад, багато пірів пропонують невелику частину їх обчислювальної потужності, що може створити щось на кшталт децентралізованого суперкомп'ютера). Кожен вузол у мережі має індивідуальний стан і є постачальником цього стану. Децентралізований сервіс – це мережа, яка побудована із незалежних провайдерів сервісу.

Перевага децентралізованих систем полягає в тому, що жоден провайдер не контролює весь стан у мережі. Якщо провайдер послуг у централізованої системи вирішує скомпрометувати дані користувача, то цей сервіс становиться скомпрометованим. Поганий провайдер тягне за собою поганий сервіс. У децентралізованій системі, що складається з багатьох постачальників послуг (пірів), які контролюють лише частину ресурсів всієї служби, недобросовісний провайдер не є еквівалентом недобросовісного сервісу: існують інші провайдери на різних кінцях системи. Децентралізований сервіс може бути реалізований через мережу незалежних провайдерів, які конкурують з метою надання якісних послуг.

Децентралізоване зберігання відбувається шляхом надання спільного доступу до файлу через мережу однорангової мережі.

По-перше, завантажувач шифрує файл, і ця особа зберігає ключ, щоб дешифрувати. Потім файл розбивається на багато менших фрагментів. Кожен з цих невеликих зашифрованих частин дублюється, щоб забезпечити надмірність. Нарешті, ці частини надсилаються на різні індивідуальні комп'ютери в одноранговій мережі.

Ті особи, які зберігають файли, мають лише невелику частину вмісту файлу, і вона зашифрована. Це означає, що хости не можуть отримати інформацію з файлу. Це також означає, що запуск атаки на хостинг вузол буде безглуздим.

Щоб відновити файл, оригінальний користувач використовує хеш таблицю розміщену у блокчейні, щоб знайти всі частини вихідного файлу та надіслати запит до мережу щоб відновити файл. Після того, як вузли відправлять різні частини, файл буде перебудовано. Тоді користувач, який завантажував файл, використовує оригінальний ключ шифрування, щоб дешифрувати.

Таблиця 1.1 – Порівняння архітектур

	Централізована	Децентралізована	Розподілена
Підтримка	Легка	Середня	Висока
Відмовостійкість	Низька	Середня	Висока
Масштабованість	Низька, зазвичай вертикально	Висока	Потенційно нескінченна
Простота розробки	Вище середнього	Досить важка	Досить важка
Оркестратор	+	-	+
Безпека даних	Низька	Висока	Висока

1.2 Загальні поняття про технологію блокчейн

Блокчейн – загальнодоступний, розподілений обліковий запис, який полегшує процес запису транзакцій і відстеження активів в мережі [9]. Актив може бути будь-чим – будинок, машина, готівка, земля – або нематеріальні як інтелектуальна власність, така як патенти, авторські права чи брендинг. Практично будь-яке значення може відстежуватися та продаватися в мережі блокчейн, зменшуючи ризик і скорочення витрат для всіх учасників.

Блокчейн має свою назву так, як у ньому зберігаються дані транзакцій – в блоках, які з'єднані разом для утворення ланцюга який зображений на рисунку 2. Блоки записують і підтверджують час і послідовність транзакцій, які потім входять в блокчейн, в межах децентралізованої мережі регулюються правилами, узгодженими учасниками мережі.

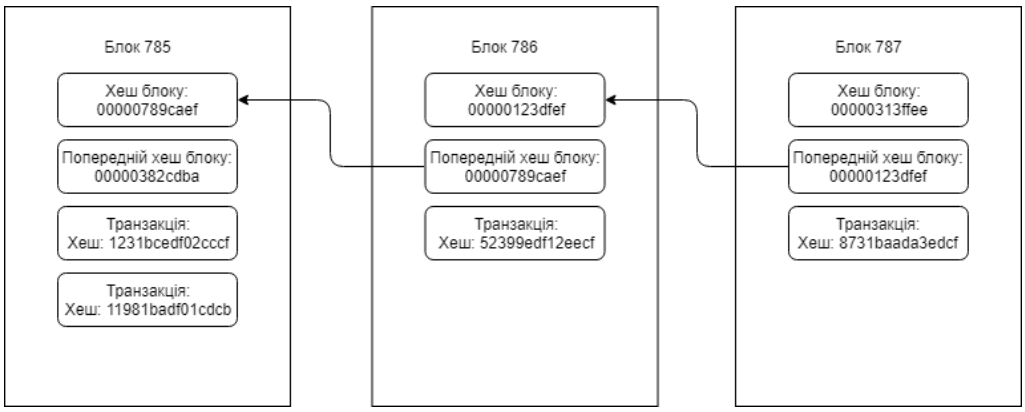


Рисунок 1.2 – Приклад блокчейну

Кожен блок містить хеш (цифровий відбиток або унікальний ідентифікатор) з датою партії нещодавніх дійсних транзакцій, а також хеш попереднього блоку. Попередній блок хешу пов'язує поточний блок разом і запобігає зміні будь-якого блоку або вставку блоку між двома існуючими блоками. Таким чином, кожен наступний блок посилює перевірку попереднього блок і, отже, весь блокчейн.

В той час, як блокчейн містить дані транзакцій, це не заміна для баз даних, технології обміну повідомленнями, обробки транзакцій або бізнес-процесів. Він містить

перевірене підтвердження транзакцій. Проте, поки блокчейн по суті служить базою даних для реєстрації транзакцій, його переваги виходять далеко за межі традиційних баз даних.

1.2.1 Proof-of-Work

Proof-of-Work, або PoW, (доказ виконання роботи) – це алгоритм досягнення консенсусу в блокчейні; він використовується для підтвердження транзакцій і створення нових блоків [10]. За допомогою PoW майнери конкурують один з одним за завершення транзакцій в мережі і за винагороду. Користувачі мережі надсилають один одному цифрові маркери, після чого всі транзакції збираються в блоки і записуються в розподілений реєстр, тобто в блокчейн. Але при підтвердженні транзакцій і організації блоків потрібно дотримуватися обережності. Робота мережі заснована на вирішенні складних математичних задач і можливості легко довести, що рішення отримано.

Це одна з проблем, що вимагають значної обчислювальної потужності. Таких проблем багато: хеш-функція, або спроба знайти вхідні дані, знаючи вихідні; розкладання цілого числа на множники; «головоломка для екскурсанта»: якщо сервер підозрює DoS-атаку, він вимагає від клієнта обчислення хеш-функцій, іноді в певному порядку, тоді це проблема обчислення значень ланцюжка хеш-функцій. У випадку з PoW використовується хешування. У міру зростання мережі проблеми стають все серйозніше, і алгоритми хешування вимагають все більшої обчислювальної потужності, так що складність завдання – актуальна проблема.

Від цього механізму залежить точність і швидкість блокчейна. При цьому проблема не повинна бути занадто складною – в цьому випадку генерація блоку займе багато часу, а значить, в мережі «зависне» багато незавершених транзакцій. Якщо проблема не може бути вирішена за передбачуване час, створення блоків стане щасливою випадковістю. Якщо ж проблема вирішується дуже просто, це робить систему вразливою для зловживань, спаму і DoS-атак. Рішення повинно бути легко платника, в іншому випадку не всі вузли зможуть зрозуміти, чи правильно був проведений розрахунок, а

значить, їм доведеться довіряти іншим вузлам, що не узгоджується з одним із найважливіших принципів блокчейна – повною прозорістю.

Майнери вирішують задачу, формують новий блок і підтверджують транзакції. Складність завдання залежить від кількості користувачів, поточної потужності і навантаження на мережу. Крім того, хеш кожного блоку містить в тому числі хеш попереднього блоку, що підвищує безпеку і унеможливлює порушення порядку створених блоків.

Основні його переваги – захист від DoS-атак і низький вплив частки криптовалюта у власності у Майнера на можливості видобутку. PoW накладає певні обмеження на дії учасників, оскільки для вирішення завдання потрібні значні зусилля. Ефективна атака також вимагає великих обчислювальних потужностей і тривалих обчислень, тому вона можлива, але невигідна на тлі високих витрат. Неважливо, скільки грошей у вас в гаманці – важливо мати великі обчислювальні можливості для вирішення завдань і формування нових блоків, а значить, власники великих капіталів не можуть приймати рішення за всю мережу.

Основні проблеми: величезні витрати, «марність» обчислень і «атака 51%». Для складних розрахунків потрібне спеціалізоване і дороге комп'ютерне обладнання. Витрати некеровано ростуть, і майнінг стає можливий тільки для великих груп майнерів. Крім того, спеціалізовані комп'ютери споживають масу енергії, що збільшує витрати. Наслідком з цього стає поступове підвищення централізації системи, оскільки це вигідно. І саме це відбувається у випадку з біткоїнами.

Майнери виконують роботу зі створення блоків, попутно споживаючи величезну кількість енергії, але обчислення, які вони роблять, абсолютно марні самі по собі. Так, вони гарантують безпеку в мережі, але їх результати не можна використовувати в бізнесі або в науці.

Атака 51% або атака більшості можлива в ситуації, коли користувач або група користувачів контролюють більшу частину потужностей мережі – це дає їм можливість контролювати які події відбуваються в мережі. Так, вони можуть монополізувати створення нових блоків і отримувати всю винагороду [11].

1.2.2 Смарт-контракти

Смарт-контракт – це електронний протокол, написаний за допомогою комп'ютерного коду [12]. Його призначення – передача інформації та забезпечення виконання умов контракту обома сторонами. Його ще називають розумний контракт. Розумні контракти засновані на технології блокчейн. Це розподілений реєстр, який являє собою децентралізовану систему, яка існує завдяки безлічі комп'ютерів, об'єднаних в одну мережу. Блокчейн дозволяє користувачам здійснювати транзакції, передавати інформацію та матеріальну цінність без банків і посередників.

Смарт-контракти – це по суті програми, які створюються на основі комп'ютерної логіки і передаються у вигляді коду. Саме тому учасники угоди або договору можуть бути впевнені, що всі умови контракту будуть дотримані, і ніхто з учасників не зможе змінити умови або інтерпретувати під себе. Код – це закон розумних контрактів.

Найпростішим прикладом використання смарт-контрактів є мультипідпис. За допомогою такого підпису, учасники договору можуть заморозити певну суму монет на блокчейні так, що в разі необхідності її витратити потрібні підписи більше половини учасників. Така умова контракту забезпечує безпеку коштів, вкладених в проект. У разі провалу, кошти будуть повернуті інвесторам автоматично. Якщо збір заявленої суми пройшов успішно, тоді учасники мультипідпису активують свої ключі, підтверджуючи сумлінність проекту, в який інвестують.

Смарт-контракти можна використовувати для будь-яких фінансових дій в сфері страхування, реєстрації або передачі власності, кредитуванні. Найбільш широке поширення розумних контрактів спостерігається в бізнес – сфері, де передбачаються виплати і дії, обумовлені платіжками. У таблиці 2 приведено порівняння смарт-контракту зі звичайним паперовим контрактом.

Таблиця 1.2 – Порівняння контраків

Смарт-контракт	Паперовий контракт
Це програма чи протокол котрий використовується на основі блокчейну	Паперовий документ
В основі закладено програмний код	Базується праві та законодавчих актах
Друкується на мові програмування	Друкується на юридичній мові
Умови неможливо змінити	Умови можна змінити, переписати чи по-іншому інтерпретувати
Умови автоматично виконуються усіма сторонами контракту	Умови можуть бути не виконані чи виконані неналежним чином
При порушенні умов контракту автоматично вводяться штрафні санкції які передбачені контрактом	При порушенні умов необхідно звертатися до суду
Усі операції здійснюються без третіх осіб і посередників	Усі операції здійснюються через багатьох посередників, тому необхідна допомога юристів, державних органів та інших
Використовуються криптовалюти для здійснення транзакцій	Транзакції здійснюються через банки
При дотриманні умов контракт, сторони отримають цінності моментально	Обмін цінностями між сторонами здійснюється зазвичай з затримками
У блокчейні зберігаються всі дані про контрагентів й можна обрати яка саме інформація буде публічною	Інформацію про контрагентів можна дізнатися лише за умови, що він наддасть виписки і довідки з державних органів

Продовження таблиці 1.2

Смарт-контракт	Паперовий контракт
Контракт можна укласти з людина з будь-якої точки світу без особистої присутності	Контракт підписується лише при особистій зустрічі двох сторін або їх довірених осіб
Гарантується безпека операції	Немає ніяких гарантій. Будь-який закон можна обійти
Шахрайство та хабарництво неможливі	Є вірогідність шахрайства та хабарництва

1.3 Огляд існуючих рішень

1.3.1 Dropbox

Dropbox – це сервіс файлового хостингу, який пропонує хмарне збереження даних, синхронізацію файлів, приватну хмару та клієнтські додатки [13]. Програмне забезпечення Dropbox дозволяє користувачам зберігати будь-який файл у виділеній папці. На рисунку 3 зображено загальний вигляд веб-додатку Dropbox. Файл потім автоматично завантажується до сервісу хмарного збереження і стає доступним для будь-якого іншого комп'ютера та пристроїв користувача, які також мають встановлене програмне забезпечення Dropbox, зберігаючи файл на всіх системах. Коли файл у каталозі Dropbox користувача змінюється, Dropbox завантажує тільки ті фрагменти файлу, які були змінені, коли це можливо. Коли файл або папка видаляються, користувачі можуть відновити його протягом 30 днів. Для користувачів Dropbox Plus цей час відновлення можна подовжити до одного року, придбавши окрему послугу "Розширені історії версій". Dropbox також пропонує функцію синхронізації у LAN мережі, де замість отримання інформації та даних з серверів Dropbox, комп'ютери в локальній мережі можуть обмінюватися файлами безпосередньо між собою, що потенційно може суттєво покращити швидкість синхронізації.

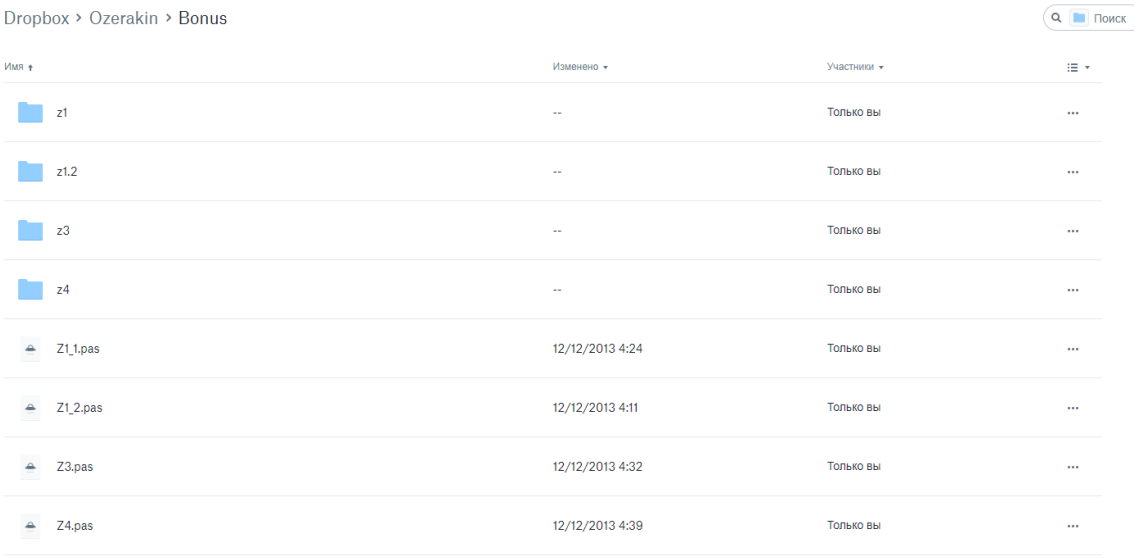


Рисунок 1.3 – Загальний вигляд веб-додатку Dropbox

Переваги:

– надає до 2ГБ безкоштовного користування хмарним сховищем;
– постійна синхронізація завантажених файлів між різними додатками;
– надає можливість надати доступ до файлів зовнішнім користувачам.

Недоліки:

– доступен лише рівень роботи з об’єктами;
– повністю централізована;
– дані на безкоштовному обліковому запису при довгій неактивності можуть бути видалені;
– мінімальний обсяг виділеного сховища котрий можна придбати 1Тб.

1.3.2 Storj

Storj – децентралізована система зберігання файлів, яка спрямована на використання всіх надлишкових можливостей у всьому світі у вигляді невикористаних жорстких дисків та пропускної здатності [14]. Ядро платформи – це спільнота користувачів або так званих "фермерів", які служать вузлами, що забезпечують зберігання даних і пропускну здатність системи. Використовуючи децентралізовану систему зберігання даних, Storj вирішує основні проблеми наявних в даний час хмарних

серверів, а саме: ризику втрати даних в результаті хакерства, простоїв в мережі або відключення живлення централізованої системи. На рисунку 4 зображено загальний вигляд Storj додатку.

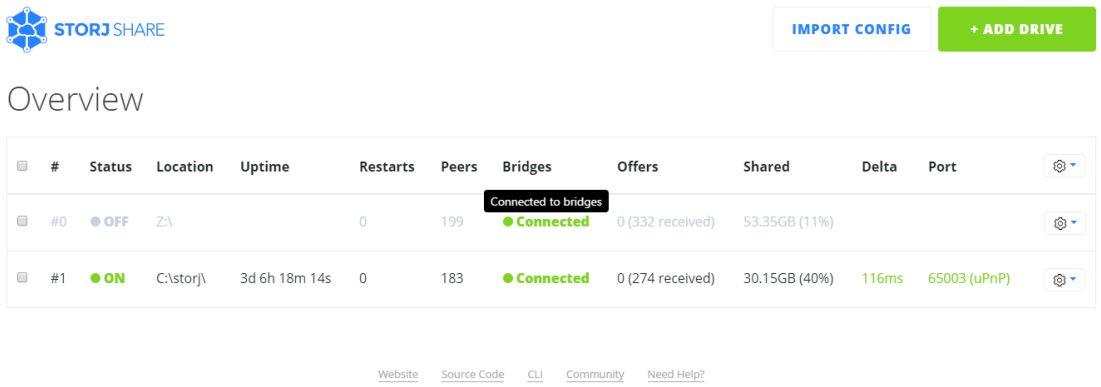


Рисунок 1.4 – Загальний вигляд додатку Storj

Перш ніж завантажувати свої дані в блочний блок, їх необхідно зашифрувати на своєму пристрої. Дані шифруються за допомогою алгоритму AES256-CTR. Це вже досить безпечно, але існує сфера застосування для інших типів шифрування, таких як конвергентне шифрування.

Однією з головних переваг для користувачів шифрування на стороні клієнта є те, що вся інформація, необхідна для дешифрування файлів, зберігається далеко від вузлів (фермерів). Є лише один ключ дешифрування, і він зберігається на клієнтській машині. Звичайно, може бути екземпляр, де ви хотіли б розшифрувати свої файли на машині, яка не використовувалась для локального шифрування. Саме тут приходить на допомогу Storj міст.

Міст Storj – це найновіша ініціатива, яка дозволить користувачам зберігати свої ключі на так званих "мостових" серверах.

Переваги:

– фіксована ціна;
– платити необхідно лише за використаний ресурс;
– завдяки розподіленій архітектурі це простіше підтримувати та оновлювати, можна масштабувати, щоб відповідати вимогам мережі;

– швидкість передачі вище, ніж у конкурентів;
– найбільша мережа.

Недоліки:

– єдина точка відказу Storj Labs сервіс (storj мости);
– більше піддається вразливостям (наприклад, DDOS) через фрагментовану систему;
– немає пропозицій з динамічними цінами

1.3.3 Sia

Sia – це розподілена система на основі блокчейн, учасники якої зберігають інформацію на жорстких дисках за платню [15]. На рисунку 5 зображено загальний вигляд Sia додатку. Участник системи може вирішити бути тільки користувачем, завантажуючи дані (в термінології Sia – renter), або же сервером, що приймає дані (в термінології Sia – host), або поєднати ці дві ролі. Хости заохочені у збереженні даних, так як вони лишаються винагороди, у разі втрати дані.

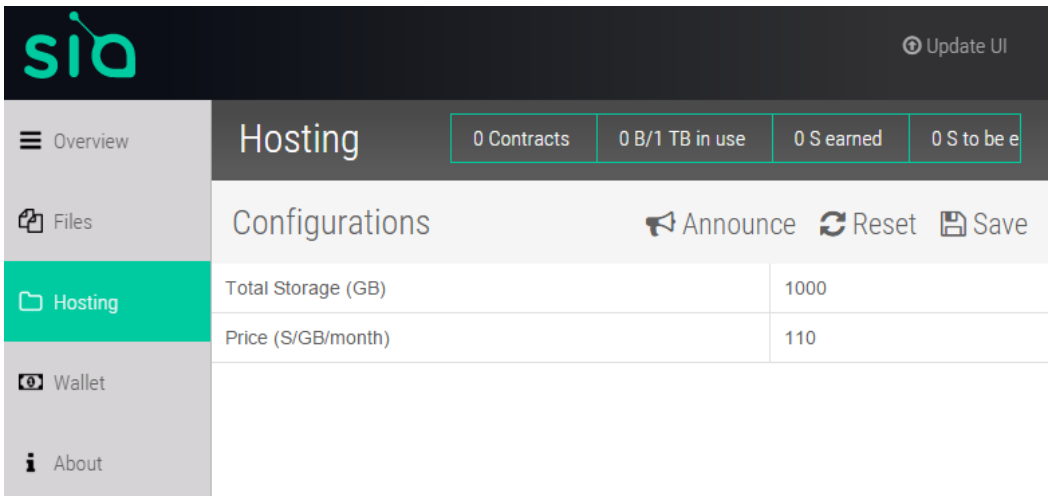


Рисунок 1.5 – Загальний вигляд Sia додатку

Коли орендар створює контракт на зберігання та починає завантажувати свої дані, Sia шифрує та розбиває дані на частини і розподіляє їх на кілька хостів (наразі 30). Дані

розбиваються, використовуючи алгоритм, який називається кодами Ріда-Соломона, до рівня x3 надмірності. Це означає, що для реконструкції даних потрібні лише 10 штук оригіналу із 30. Це дозволяє до 20 з 30 хостів бути недоступними або втратити дані, і дані все одно будуть доступні. Якщо ваші хости в мережі 90% часу, це призводить до 99,99% загального часу роботи даних, що зберігаються в мережі Sia.

Оскільки дані зашифровані і розділені на частини, хости не мають доступу до даних й навіть не можуть визначати, які дані вони зберігають. Орендар може відчувати себе в безпеці, знаючи, що тільки він має доступ до своїх даних, і хост захищений від відповідальності за будь-які дані, які вони зберігають, оскільки хост не може нічого знати про дані. Хости також мають стимул залишатись в Інтернеті та зберігати дані за допомогою кількох показників які підвищують їх рейтинг.

Переваги:

- повністю децентралізований, не має єдиної точки відмови;
- має пропозиції із динамічними цінами;
- дуже надійний завдяки консенсусу блокчейна;
- мінімально можливий ризик компрометації.

Недоліки:

- необхідно внести депозит у вигляді монет SIA якщо бажаєте орендувати сховище в мережі;
- допускається лише оплата за допомогою рідної криптовалюти SIA;
- потрібно спочатку завантажити весь блокчейн, щоб стати фермером, що вимагає виділення простору та годин синхронізації;
- важко масштабувати та обробляти великі обсяги даних через обмеження блокчейну;
- sia блокує ваші кошти на зберігання якщо ви виступаєте як фермер для застави.

1.4 Алгоритм консенсусу для систем збереження на основі технології блокчейн

На сьогоднішній день хмарні системи збереження повністю інтегрувалися з повсякденним життям. З найбільш яскравих представників можна відмітити такі як Dropbox, Google Drive, OneDrive та інші [16]. Вони надають у хмарі потенційно нескінчений простір. Але здебільшого вартість цієї «нескінченності» занадто висока. Також із-за характеру централізованості дані можуть бути скомпрометовані, втрачені чи тимчасово недоступні. Щоб усунити ці недоліки слід задуматися над використанням розподілених чи децентралізованих рішень. У статті пропонується алгоритм досягнення обраних налаштувань доступності даних та їх захист у разі спроби викрадення.

Визначимо умови котрих необхідно досягнути при розробці алгоритму консенсусу для збереження даних:

- мінімізація можливості скомпрометувати дані;
- виконання обраного часу доступності даних;
- прозорість процесу оплати за зберігання;
- автономна перевірка доступності даних;
- розподілене збереження даних.

Алгоритм отримав референсу назву Proof-of-Agreement. Визначимо взаємодіючі сторони та їх ролі при використанні алгоритму:

А) Клієнт:

Як клієнт я укладаю договір з провайдером сервісу про збереження даних. У договорі визначається скільки годин в добу дані повинні доступні, розмір збережувальних даних, протягом якого часу необхідно зберігати. Перед початком завантаження даних, вноситься депозит на суму котру розрахував провайдер сервісу та встановлюється тимчасове блокування, після того як дані завантажені, у дію вступає заключений договір;

Б) Провайдер сервісу:

Як провайдер сервісу я надаю можливість завантаження даних у систему. Виступаю у ролі регулятора в питаннях виконання умов доступності даних. Надаю

прозорість щодо проведеного аудиту доступності даних. Обчислюю рейтинг кожного провайдера сервісу, котрий буде надалі використано у тендері за збереження даних;

В) Провайдер сховища:

Як провайдер сховища я надаю вільний простір у своєму сховищі яким може бути наприклад жорсткий диск вбудований у комп’ютер. Зберігаю завантажені дані. Проходжу час від часу аудит щодо доступності даних. Варто зазначити, що провайдер сховища – це одиниця робочої сили для провайдера сервісу. Тобто провайдер сервісу виступає агрегатом провайдерів сховищ.

Перейдемо до опису алгоритму з технічної сторони. Для досягнення прозорості процесу оплати та виконання умов договору застосуємо технологію блокчейн та смарт-контракт – це комп'ютерний алгоритм, призначений для укладання і підтримки комерційних контрактів в технології блокчейн. Найбільш популярна мережа котра використовує ідею смарт-контрактів є Ethereum [17]. Але його використання у даному алгоритмі має певні складнощі. Оскільки умови й алгоритм контракту повинні бути повністю детермінованими – це необхідно задля того, щоб майнер міг завантажити увесь блокчейн і прорахувати хеші блоків за алгоритмом дерева Меркла [18], щоб впевнитися, що блокчейн не містить підробок. Саме тому даний алгоритм потребує розробку власної мережі, оскільки процес аудиту побудовано на зовнішніх викликах провайдерів сховищ і не може бути відтвореним із 100% вірогідністю.

Смарт-контракт повинен контролювати процес виплат винагородження провайдеру сховища (Storage Reward) та податку провайдеру сервісу (Service Fee). Опишемо послідовність дій при укладанні договору:

1) користувач надсилає запит на збереження даних;
2) провайдер сервісу розраховує необхідний залишок на адресі клієнта (під «адресою» мається на увазі його фізичний гаманець у блокчейні);
3) створюється смарт-контракт;
4) здійснюється блокування коштів клієнта;
5) завантажуються дані;
6) починається процес аудиту доступності даних.

Розглянемо процеси які проходять у системі такі як завантаження та отримання даних й аудит доступності даних, а у додатках Г та Ґ графічно зображено кроки алгоритму за допомогою нотації UML діаграми активності.

Для мінімізації можливості скомпрометувати дані, перед тим як надіслати дані до провайдеру сервісу вони шифруються особистим ключем клієнта, після цього сервер розбиває дані на блоки з фіксованим розміром й надсилає їх до провайдерів сховищ, обчислює їх хеш та створює блок у блокчейні що містить перелік усіх хешів блоків та відповідних їм провайдерів сховищ що були завантажені.

Для вибору набору сховищ використовуються такі параметри як доступність та завантаженість провайдера сховищ, його рейтинг та вимоги клієнта щодо доступності даних. В-першу чергу беруться у тендер ті сховища які на даний момент часу присутні у мережі, далі порівнюється відносна завантаженість кожного з них котра розраховується як відношення використаного місця до виділеного місця. Після того як отримано такий набор сховищ необхідно співставити їх рейтинг, вимоги клієнта щодо до доступності даних та необхідну кількість реплік блоків. Якщо клієнт бажає високий рівень доступності даних, то більше шансів виграти тендер буде у тих провайдерів котрі мають високий рейтинг й меншу завантаженість. Кількість реплік блоків залежить від того які провайдери будуть обрані у результаті проведеного тендеру, якщо тендер виграють сховища з середнім, то кількість реплік буде більшою задля досягнення вимог клієнта.

Після того як дані були завантаженні провайдер сервісу згідно до вимог клієнта щодо доступності даних розраховується коли необхідно перевірити провайдерів сховищ щодо наявності завантажених блоків даних.

Розраховується відрізок часу протягом якого буде здійснено ряд перевірок. Припустимо, що клієнт забажав доступність даних протягом 20 годин у добу. Отже, оберемо тих провайдерів які мають рейтинг, який свідчить про те, що вони доступні не менше ніж 20 годин та створимо репліки на інших сховищах з доступністю не менше ніж 10 годин. Провайдер сервісу знає хеш блоку, його зсув у файлі та довжину, він обирає певний відсоток блоків від загальної кількості які приймають участь у аудиті. Далі знаходяться відповідні їм master та slave сховища [19]. Другі зберігають репліки блоків даних. Спочатку провайдер сервісу надсилає запит до головного сховища та вимагає

обрахувати хеш блоку з певним зсувом та довжиною, якщо провайдер не відповідає протягом певного короткого проміжку часу, здійснюється кілька разів спроба повторити запит, далі запити поширюються на slave сховища.

У разі успішної перевірки створюється транзакція на блокчейні яка перераховує частину грошей з адреси клієнта на адреси провайдерів сховищ та сервісу. У разі невдачі перевірка відкладається на більш довгий час й за тим повторюється знову процес аудиту. Якщо після декількох таких довгих спроб аудит невдалий, то клієнту повертається його залишок котрий розраховуються як депозит з урахуванням усіх виплат по аудиту.

Після кожного аудиту перераховується рейтинг провайдера сховища. При вдалому проходженні перевірки рейтинг збільшується, й навпаки, при невдачі рейтинг зменшується. Рейтинг відіграє невідмінну роль при виборі провайдера сховища. Якщо клієнт бажає дуже високої доступності, то необхідно мати список «довірених» провайдерів сховищ, й навпаки, необхідно слідкувати за «слабкими» провайдерами й менше їм довіряти блоків з більшою кількістю реплік.

Для продовження терміну збереження даних є можливість внеску додаткового депозиту розрахованого відповідно до визначеного терміну. Варто зазначити, що на цьому етапі вже не можна змінити умови доступності даних, оскільки це може призвести до процесу перерозподілення блоків задля задоволення умов.

Клієнт надсилає запит до провайдер сервісу для отримання даних. Сервер знаходить відповідність даних до їх блоків. Після чого формує набор master та slave сховищ які зберігають необхідні блоки. Спочатку сервіс робить запит до головних сховищ якщо ті не відповідають протягом певного часу робиться запит до slave сховищ.

У разі невдалої спроби отримання даних запит клієнта ставить у чергу й тепер клієнт виступає як аудитор й бере участь в одному циклі перевірки даних, якщо ж після проходу одного циклу перевірки дані не вдалося отримати у повному обсязі, такий аудит вважається невдалим й клієнт отримує залишок. Після того як клієнт отримав дані, він їх дешифрує за допомогою особистого ключа й може далі продовжувати користуватися ними.

2 ПРОЕКТУВАННЯ СИСТЕМИ ЗБЕРЕЖЕННЯ ДАНИХ

2.1 Загальна архітектура системи

В основу архітектури системи було вкладено одразу декілька типових підходів таких як клієнт-серверна, модель акторів, багатошарова та мікросервісна. На рисунку 2.1 зображено взаємодію усіх компонентів системи. У додатку Д зображено архітектуру системи за допомогою діаграми компонентів.

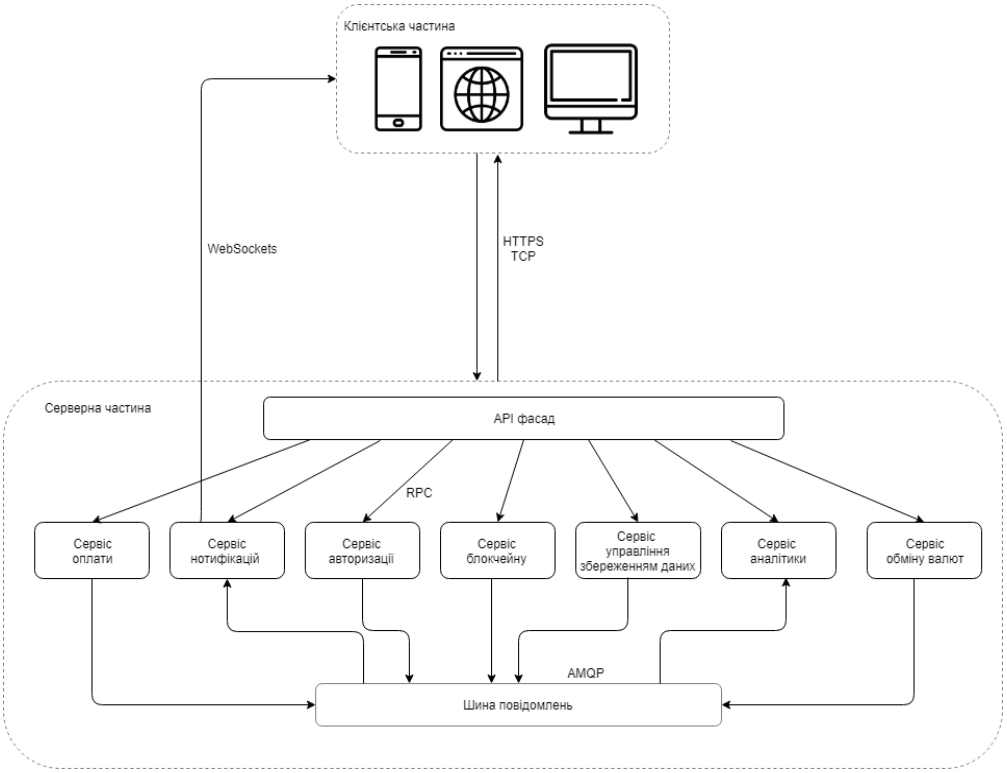


Рисунок 2.1 – Загальна архітектура системи

Розглянемо дві основні взаємодіючі сторони – клієнт та сервер. Клієнтська частина забезпечує взаємодію з користувачем і може бути виконана за допомогою мобільного, настільного чи веб додатку. Вона робить запити до серверу, обробляє отримані дані задля того, щоб відобразити їх користувачу.

Серверна частина обробляє запити від клієнта, надсилає відповіді, закриває собою комунікацію з мікросервісами, зберігає отримані дані , надсилає пуш повідомлення клієнтам та інше.

Між клієнтом та сервером комунікація може бути здійснена за двома протоколами HTTPS та TCP. Це обґрунтовується тим, що при взаємодії з публічним API об'єм даних є досить невеликим й можна знехтувати надлишковістю HTTPS протоколу. При завантаженні чи отриманні попередньо завантажених даних використовується чистий TCP протокол тому, що кількість даних може бути занадто висока, але для веб-додатку використовується HTTPS оскільки не має можливості для комунікації напряму з сокетом.

Використання HTTPS для публічного фасаду серверу надає можливість легко сформувати контракти взаємодії з використанням Web API з дотримання рекомендацій REST-стилю й перспективу для впровадження Open API [20]. Далі розглянемо кожну частину більш детально.

2.2 Архітектура клієнтської частини

В основу архітектури клієнтської частини закладено використання багатошарової, а саме тришарової [21]. На рисунку 2.2 зображено ієрархію шарів. Можна виділити три основні шари такі як робота з даними, бізнес-логіка й презентації/представлення. Розглянемо кожен шар і його відповідні обов’язки.

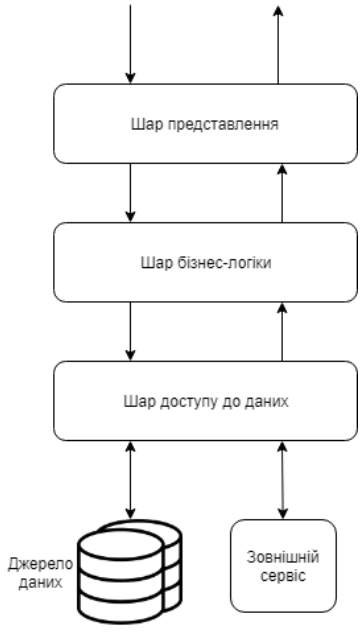


Рисунок 2.2 – Тришарова архітектура

Шар презентації/представлення (Presentation Layer) містить функціональність, орієнтовану на користувача, відповідальна за керування взаємодією користувача з системою, і, як правило, складається з компонентів, які забезпечують загальний міст у основній бізнес-логікі, інкапсульованої в бізнес-шарі.

Компоненти користувацького інтерфейсу – це візуальні елементи програми, які використовуються для відображення інформації користувачеві та прийняття введення користувача.

Компоненти логічного представлення – це код програми, який визначає логічну поведінку та структуру програми таким чином, що не залежить від будь-якої конкретної реалізації користувальницького інтерфейсу. При виконанні шаблону "Відокремлена презентація" (Separated Presentation pattern) компоненти логіки презентації можуть включати Presenter, Presentation Model та компоненти ViewModel. Шар презентації може також включати компоненти "Модуля представлення шару моделей" (Presentation Layer Model components), які інкапсулюють дані з вашого бізнес-рівня або компонентів "Представницького об'єкту" (Presentation Entity), які інкапсулюють бізнес-логіку та дані у формі, які можна легко використати за допомогою шаблону презентації.

Шар бізнес-логіки (Business Logic Layer) реалізує основну функціональність системи та інкапсулює відповідну бізнес-логіку. Вона, як правило, складається з компонентів, деякі з яких можуть виявляти сервісні інтерфейси, які можуть використовуватися іншими викликаючими сторонами. Цей шар зазвичай включає в себе наступне:

Фасад застосування (Application facade) – це необов'язковий компонент, як правило, забезпечує спрощений інтерфейс компонентів бізнес-логіки, часто шляхом об'єднання декількох бізнес-операцій в одну операцію, що полегшує використання бізнес-логіки. Це зменшує залежності, тому що зовнішнім викликаючим сторонам не потрібно знати деталі бізнес-компонентів та взаємовідносини між ними.

Компоненти бізнес-логіки – бізнес-логіка визначається як будь-яка логіка додатків, яка стосується пошуку, обробки, перетворення та керування даними програми; застосування бізнес-правил та політики; і забезпечення узгодженості та обґрунтованості даних. Щоб максимізувати можливості повторного використання, компоненти бізнес-

логіки не повинні містити будь-якої поведінки або логіки додатків, що є специфічною для use case або user story.

Шар доступу до даних (Data Access Layer) забезпечує доступ до даних, розміщених у межах системи, а також даних, відкритих іншими мережевими системами; можливо, доступ через сервіси. Він виражає загальні інтерфейси, які можуть використовувати компоненти бізнес-рівня.

Компоненти доступу до даних – ці компоненти абстрагують логіку, необхідну для доступу до основних сховищ даних. Вони централізують загальну функціональність доступу до даних, щоб спростити налаштування та підтримку програми. Деякі схеми доступу до даних можуть вимагати, щоб розробник визначив і впроваджував загальну логіку доступу до даних в окремих компонентах які можуть бути повторно використані. Інші схеми доступу до даних, включаючи ORM (Object/Relational Mapping), автоматично реалізують такі компоненти, зменшуючи обсяг коду доступу до даних, який розробник повинен написати.

Сервісні агенти (Service agents). Коли бізнес-компонент повинен мати доступ до даних, наданих зовнішньою службою, вам може знадобитися реалізація коду для управління семантикою спілкування з цим конкретним сервісом. Сервісні агенти реалізують компоненти доступу до даних, які ізолюють різні вимоги до викликаючих сторін у вашій програмі, а також можуть надавати додаткові послуги, такі як кешування, підтримка в автономному режимі та основне відображення між форматом даних, що надаються сервісом, та формат, який вимагає ваша програма.

В архітектурі клієнтського додатка шар доступу до даних є HTTP-клієнт котрий серіалізує дані, виконує запити до конкретного ресурсу сервера та десеріалізує відповідь. Також при роботі з даними (завантаженні чи отриманні) використовується RPC-клієнт. Шар презентації може бути виконаний як мобільний, настільний чи веб-застосунок.

2.3 Архітектура серверної частини

Серверна частина виконана у стилі мікросервісної архітектури (див. рисунок 2.3) [22]. Мікросервісна архітектура або просто мікросервіс є відмінним способом розробки програмних систем, які намагаються зосередити увагу на побудові одно функціональних модулів з чітко визначеними інтерфейсами та операціями.

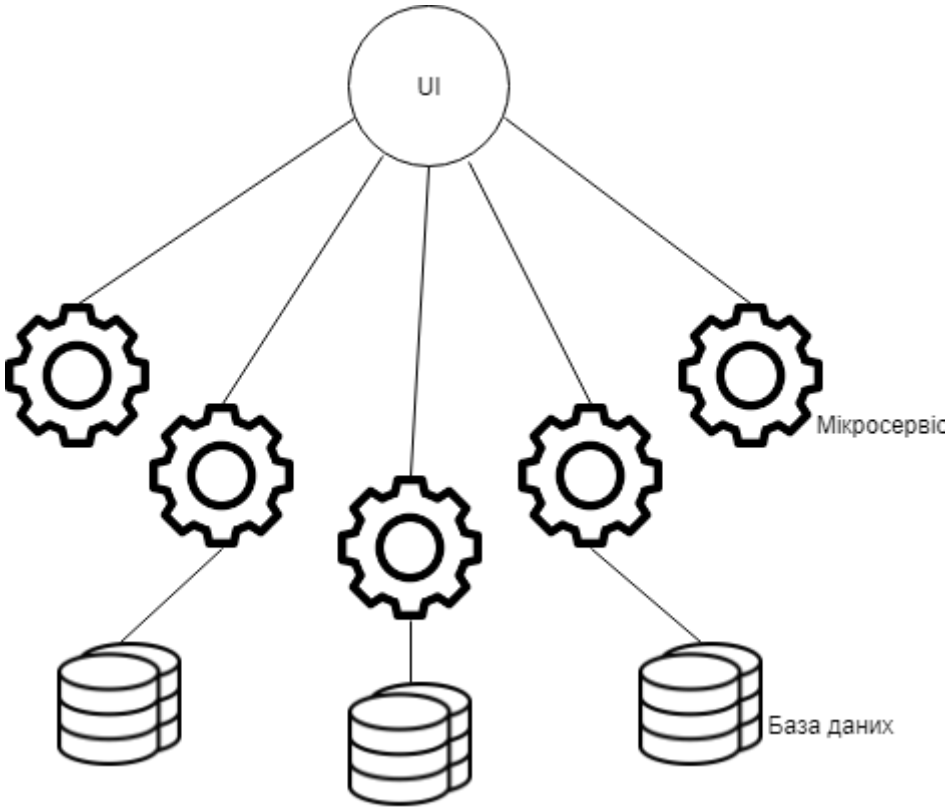


Рисунок 2.3 – Мікросервісна архітектура

Мікросевісна архітектура – це підхід до розробки єдиної програми як сукупності малих сервісів, кожна з яких працює у власному процесі та спілкується з легкими механізмами, часто – API ресурсів HTTP. Ці служби будуються на основі бізнес-можливостей і незалежно розгортаються за допомогою повністю автоматизованого механізму розгортання. Існує мінімальне централізоване управління цими службами, які можуть бути написані на різних мовах програмування та використовувати різні технології зберігання даних.

Корисно порівняти його з монолітним стилем (див. рисунок 2.4): монолітний додаток, побудоване як єдине ціле. Прикладні програми підприємства часто складаються з трьох основних частин: користувальницький інтерфейс на стороні клієнта (який складається з HTML-сторінок та JavaScript, який запускається в браузері на машині користувача); база даних (що складається з багатьох таблиць, вставлених в загальну та, як правило, реляційну систему керування базами даних система), а також серверний додаток. Прикладна програма на сервері буде обробляти HTTP-запити, виконувати доменну логіку, завантажувати та оновлювати дані з бази даних, а також вибирати і заповнювати HTML-представлення, що надсилаються браузеру. Цей серверний додаток є моноліт – єдиний логічний виконувальний файл [2]. Будь-які зміни в системі, передбачають створення та розгортання нової версії додатка на сервері.

Рисунок 2.4 – Монолітна архітектура

Такий монолітний сервер – це природний спосіб підійти до створення такої системи. Вся ваша логіка обробки запиту виконується в одному процесі, що дозволяє використовувати основні функції вашої мови для поділу програми на класи, функції та простори імен. Забезпечуючи певну обережність, ви можете запустити та протестувати програму на ноутбуку розробника, а також використовувати пайплайн розгортання, щоб

переконатися, що зміни правильно перевірені та розгорнуті у продакшені. Ви можете горизонтально масштабувати моноліт, запустивши багато екземплярів для балансування навантаження.

Монолітні програми можуть бути успішними, але все частіше люди відчувають розчарування з ними, особливо в міру того, як більше додатків розгортаються в хмарі. Зміни, зроблені невеликою частиною програми, вимагають, щоб весь моноліт був перебудований та розгорнутий. З часом часто важко зберегти гарну модульну структуру, що ускладнює збереження змін, які повинні впливати лише на один модуль у межах цього модуля. Масштабування вимагає масштабування всього додатку, а не його частини, що вимагає більшого ресурсу.

В нашій системі можна виділити такі обмежені контексти як авторизація, управління збереженням даних, аналітики, блокчейн й смарт-контракти, сповіщення, обмінник, оплата. Розглянемо головні функції кожного з них.

Сервіс авторизації його головними функціями є реєстрація користувачів, визначення ролей, валідація, випуск та перевипуск токенів, збереження активних сесій та пристроїв користувачів.

Сервіс управління збереженням даних його головними функціями є пошук активних провайдерів сховищ, збереження відповідності блоків до файлу й їх розташування у провайдерів, розсилка блоків, проведення аудиту.

Сервіс аналітики його головні функції такі як підраховування рейтингу провайдерів сховищ, регулярне опитування на предмет їх он-лайну у системі, кількість використаного місця й взаємодія з зовнішніми сервісами аналітики.

Сервіс блокчейну його головні функції такі як створення блокчейну, додавання блоку, створення транзакції, створення смарт-контракту, виконання смарт-контракту.

Сервіс обміну валют його головними функціями є віддача актуальних курсів валютних пар та їх обмін.

Сервіс оплати його функціями є приймання депозитів різними способами такими як через мережу Bitcoin, Ripple, Ethereum, VISA, MasterCard та навпаки надсилання грошей у ці мережі.

Задля забезпечення комунікації між клієнтом та сервіси закриємо мікросервісну частину публічним фасадом. Це паттерн проектування котрий призначен для приховання складностей реалізації системи та її підсистем й надання більш спрощеного інтерфейса для комунікації з нею. Даний фасад буде аутентифікувати запити та перенаправляти вхідні запити до відповідних мікросервісів.

2.4 Комунікація між мікросервісами

Мікросервіси можуть спілкуватися через TCP або AMQP протоколи. У випадку TCP використовується підхід RPC (Remote Procedure Call) [23].

Advanced Message Queuing Protocol (AMQP) – це відкритий протокол для асинхронних повідомлень за допомогою шини/хабу [24]. Він забезпечує шифрування та взаємодії обміну повідомленнями між сервісами та додатками. Протокол використовується в клієнт-серверній, мікросервісній архітектурах, IoT пристроях та інші.

AMPQ є ефективним, портативним, багатоканальним та безпечним. Бінарний протокол забезпечує аутентифікацію та шифрування за допомогою SASL або TLS, спираючись на транспортний протокол, такий як TCP. На рисунку 2.5 зображено основні компоненти протоколу. Протокол обміну повідомленнями швидкий і має гарантовану доставку з підтвердженням отриманих повідомлень. AMPQ добре працює в багатокористувацьких середовищах і забезпечує засоби для делегування завдань та збільшення швидкості обробки запитів від сервера. Оскільки AMPQ – це streamed система двійкових повідомлень з чітко визначеними поведінкою обміну повідомленнями, забезпечується взаємодія клієнтів різних постачальників.

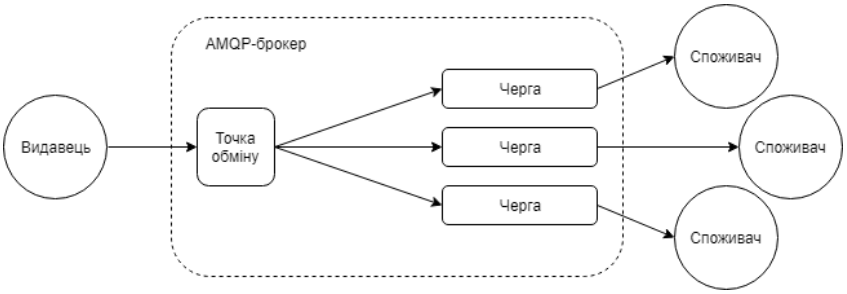


Рисунок 2.5 – Архітектура AMQP

AMQP дозволяє використовувати різні гарантовані режими обміну повідомленнями:

- at-most-once – відправлений один раз з можливістю втрати;
- at-least-once – гарантія доставки з можливістю дублювання повідомлень;
- exactly-once – гарантія одноразової доставки.

В протоколі AMQ фігурують три основні поняття:

- повідомлення – одиниця обміну інформацією, яка надсилається до точки обміну;
- точка обміну – приймає повідомлення та надсилає їх до відповідних черг (однієї чи більше) й не зберігає їх;
- черга — тут зберігаються повідомлення, доки вони не будь забрані споживачем.

З рисунку 2.5 видно що можна виділити дві сторони: постачальник (publisher) та споживач (consumer). Постачальник формує повідомлення та надсилає до точки обміну. Споживач слухає необхідну чергу(-и), отримує повідомлення та здійснює необхідну обробку.

В нашому випадку в ролі постачальників можуть виступати такі сервіси як оплати, управління збереженням даних, блокчейн, а споживачами можуть сервіси сповіщень та аналітики.

2.5 Типова архітектура сервісу

Кожен сервіс виконує операції читання та запису. Умовно можна виділити два компонента всередині кожного сервісу – управління (Management) та виконання (Operations).

Компонент управління займається тим, що обслуговує будь-які запити, у випадку читання – звертається до джерела даних, конвертує отримані дані у необхідний формат й повертає відповідь. Якщо розглянути запис, то компонент управління робить первинну

валідацію даних, звертається до джерела даних за необхідності, обробляє дані таким чином, щоб передати їх до компоненту виконання. Останній у свою чергу може здійснювати запис до джерела даних, викликати інші мікросервіси та надсилати повідомлення щодо виконаної операції. Задля забезпечення узгодження даних спроектуємо компонент виконання за моделлю акторів [25].

Модель актора має свої теоретичні коріння в моделюванні паралельних процесів та концепціях передачі повідомлень. Фундаментальною ідеєю моделі актора є використання акторів як одночасних примітивів, які можуть діяти при отриманні повідомлень різними способами:

– надіслати кінцеву кількість повідомлень іншим акторам.
– створюється кінцеве число нових акторів.
– зміна власної поведінки, набуває чинності, коли обробляється наступне вхідне повідомлення.

Для спілкування модель актора використовує асинхронне передавання повідомлень. На рисунку 2.6 зображено модель акторів та їх комунікацію. Зокрема, він не використовує будь-які проміжні об'єкти, такі як канали. Натомість кожен актор має mailbox і власну адресу. Ці адреси не можна плутати з ідентифікаціями, і кожен актор може не мати жодної чи однієї або декількох адрес. Коли актор надсилає повідомлення, він повинен знати адресу одержувача. Крім того, акторам дозволено надсилати повідомлення собі, які вони отримають і оброблятимуть пізніше на наступному етапі. Зауважте, що відображення адрес та акторів не є частиною концептуальної моделі (хоча це і є особливість реалізацій).

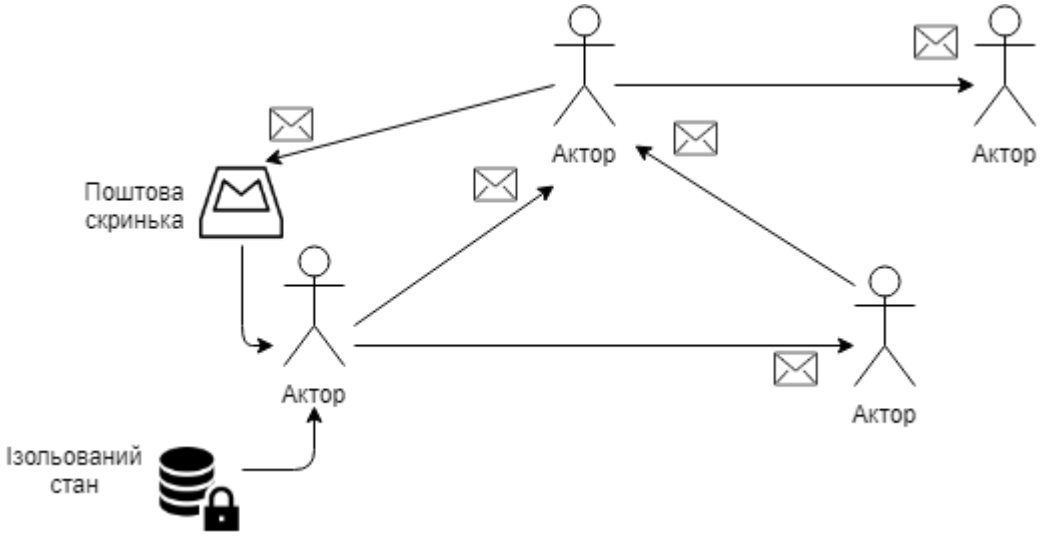


Рисунок 2.6 – Модель акторів

Повідомлення відправляються асинхронно і можуть оброблятися досить довго, щоб в кінцевому підсумку прийти до поштової скриньки приймача. Крім того, моделі акторів не гарантують доставки повідомлень. Очікування та видалення повідомлень у поштовій скриньці – це атомні операції, тому не може бути race condition. Актор обробляє вхідні повідомлення з його поштової скриньки послідовно, використовуючи вищезгадані можливості реагувати. Третя можливість, зміна власної внутрішньої поведінки, врешті-решт, дозволяє врегулювати змінну стану. Однак нова поведінка застосовується лише після обробки поточного повідомлення. Таким чином, кожне виконання обробки повідомлень, як і раніше, являє собою функцію вільного побічного ефекту від концептуальної перспективи. Модель актора може бути використана для моделювання розподілених систем, оскільки кожен актор цілком незалежний від будь-яких інших. Спільного стану немає, а взаємодія між акторами ґрунтується лише на асинхронних повідомленнях, як показано на рисунку 2.6.

Якщо розглянути архітектуру сервісу за тришаровим стилем (див. рисунок 2.7), то маємо наступне:

– шар доступу до даних представляє собою обслуговуючі сервіси для роботи з реляційними чи NoSQL базами даних, котрі використовуються для читання в компоненті управління та читання й запису в компоненті виконання;

– шар бізнес-логіки зосереджений в основному в компоненті виконання, в компоненті управління може бути частина бізнес-логіки задля виконання попередньої валідації вхідних даних й стану об’єкта;
– шар представлення/презентації являє собою фасад котрий виставляє компонент управління для забезпечення комунікації іншим сервісам.

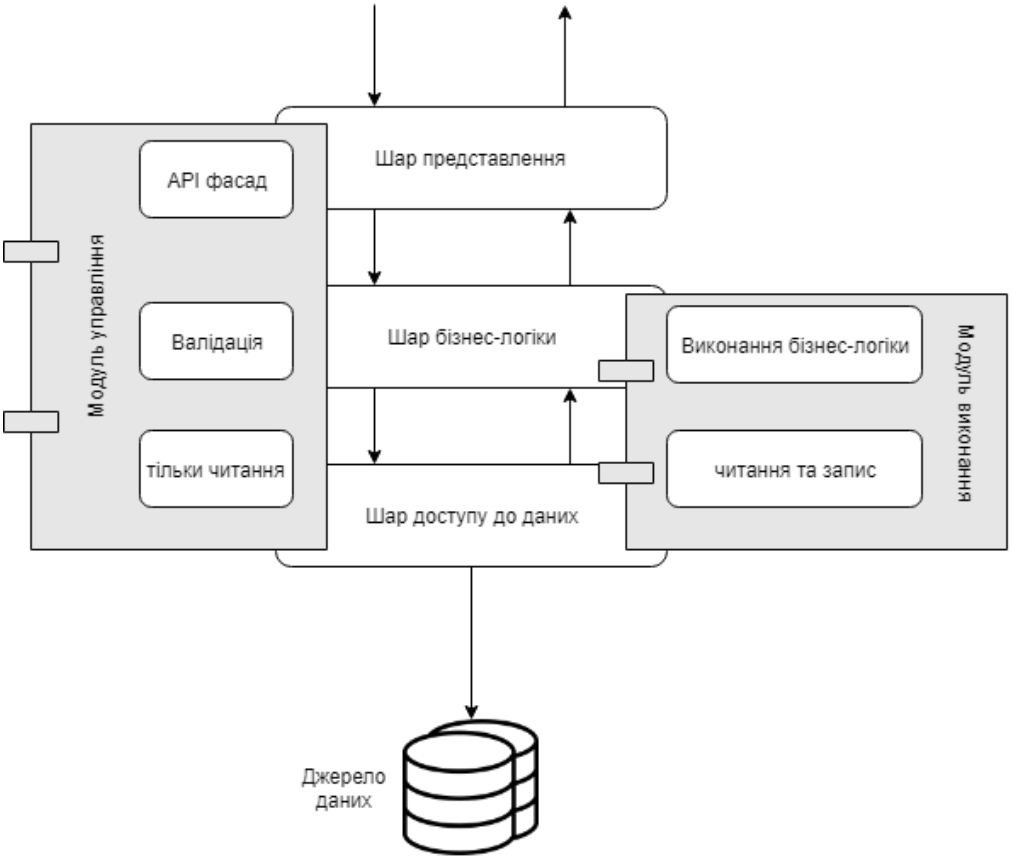


Рисунок 2.7 – Типова архітектура сервісу в розрізі тришарової архітектури

2.6 Основні функції та ролі в системі

У додатку Е зображено діаграму прецедентів для існуючих акторів у системі. Розглянемо ключові ролі та їх можливості. Можна виділити три основні сторони такі як клієнт, сервіс та провайдери сховищ.

Клієнт має можливість завантажувати, отримувати дані, створювати власні ключі шифрування та використовувати їх при завантаження.

Сервіс проводить тендери для провайдерів сховищ, розбиває на блоки дані від клієнта та виступає регулятором щодо умов виконання попередньо укладеного контракту.

Провайдер сховища може надавати вільне місце, виводити гроші, котрі отримав від зберігання блоків даних та приймати участь у тендері.

2.6.1 Підсистема управління даними

До головних функцій підсистеми управління даними можна віднести такі як завантаження, отримання, відображення списку завантажених даних, побудова віртуальної файлової системи (під цим слід розуміти логічну структуризацію завантажених даних).

Розглянемо процес завантаження даних від клієнта до провайдера сховища. Перш за все клієнт обирає файл, надалі необхідно обрати ключ яким буде зроблено шифрування завантажуваємих даних, надалі укладається контракт обирається рівень доступності даних. Після того як визначено рівень доступності даних, до серверу надсилається інформація про об’єм й який рівень необхідно забезпечити даним. Після обробки отриманої інформація, приймається рішення щодо кількості й рівню провайдерів сховищ, також яку надлишковість даних обрати та аналізується поточний рівень завантаженості мережі.

Усі складові котрі входять щодо аналізу вимог клієнту формуються підсистемами оцінювання й рейтингу та тендера. Після отримання відповіді від цих підсистем, обирається необхідний розмір розбиття файлу на блоки, кількість реплік й резервується місце на обраних провайдерах сховища. При високому рівні доступності буде обрано більший розмір блоку й більша кількість реплік, при низькому рівні буде обрано менший розмір блоку й помірна кількість реплік.

Далі відкривається сесія завантаження даних. Сервер відкриває сокет по якому необхідно завантажувати дані й повідомляє про клієнта. Останній підключається до нього й починає завантажувати дані попередньо шифруючи їх власним ключем. Дані потрапляють на фасад, розбиваються на блоки, с кожного блоку вираховується його хеш

й зберігається відповідність зміщення у файлі, довжини блоку та хеш. Після цього блок та його репліка завантажується до необхідних провайдерів сховищ.

Після того як дані повністю завантаженні, клієнт надсилає запит щодо закінчення сесії завантаження. Надалі застосовуються підтримуючі процеси задля виконання умов контракту котрі будуть описані у наступних підрозділах.

Розглянемо процес отримання даних від запиту до повного їх дешифрування. Клієнт надсилає запит щодо отримання даних, сервіс визначає на яких провайдерах сховищ він розташований. Починає з кожним з них відкривати сесію отримання блоків. У разі недоступності основного провайдера, протягом певного часу, сервіс починає використовувати резервні сховища. Якщо ж і з ними неможливо встановити сесію, то далі від обраного рівня доступності вираховується через скільки годин, хвилин, секунд необхідно повторити запит. У разі невдачі при повторній спробі, контракт вважається недійсним й клієнт отримує свою компенсацію.

Після того як встановлено сесії з провайдерами сховищ, сервер відкриває сокет й повідомляє про це клієнта. Сервер починає отримувати блоки від сховищ, звіряти хеші, й віддавати їх клієнту. Клієнт починає викачувати з сокету дані й дешифрувати. Після того як дані усі отримано сесія закривається.

2.6.2 Підсистема управління контрактами про рівень доступності даних

До головних функцій підсистеми укладання контракту можна віднести такі як формування контракту, перевірку підпису й створення тимчасового адреса для оплати послуг системи.

Контракт закладається між клієнтом та провайдером сервісу (нашою системою). В ньому вказується скільки годин у добу дані повинні бути доступні, протягом якого часу, об’єм даних й підписується клієнтом. Після підпису, створюється тимчасова адреса, потім у пулі непідтверджених транзакцій створюється транзакція котра перераховує з адреси користувача на тимчасову адресу. Передбачається створення власної криптовалюти й блокчейну, оскільки існуючі рішення не задовольняють потреби системи, але перші реалізації передбачають використання існуючих рішень.

Після того як хтось із майнерів успішно побудує блок та його підтвердить мережа, контракт вступає в силу. Надалі здійснюється аудит виконання умов контракту й проводяться виплати з тимчасової адреси провайдеру сервісу та провайдерам сховищ підсистемою оплати та виплат.

Аудит здійснюється враховуючи обраний рівень доступності даних. Суть полягає у тому щоб провайдер сховища довів те що він виконує поставлені перед ним умови. Перед ним ставиться невелика задачка, підсистема обирає довільним чином блок котрий повинен зберігати провайдер й надсилає запит щоб він обрахував хеш того блоку. Після отримання відповіді сервіс порівнює із власними хешем котрий він отримав якщо він такий сам аудит вважається пройденим. Такий процес робиться для значної більшості блоків даних. Також задля впевненості що дані дійсно зберігаються у сховищі, аудитор може запитати не їх хеш, а повний блок. Потім на своїй стороні обрахувати хеш і порівняти.

У разі якщо не вдалося отримати відповідь, аудитор робить декілька спроб повторити запит у короткий проміжок часу, щоб отримати відповідь. Якщо після й надалі не має відповіді провайдера, розраховується наступний час аудиту котрий необхідно зробити саме з цим провайдером й не порушує умов контракту. Після довгої затримки ще раз надсилається запит, якщо й надалі провайдер не відповідає та порушує умови доступності – тобто не має вже можливості відкласти на довгий час, контракт вважається порушеним.

Після того як контракт визнається недійсним клієнту повертаються його залишок котрий розраховується як різниця між внесеним депозитом та кількістю оплаченого часу зберігання даних.

2.6.3 Підсистема тендера для провайдерів сховищ

До головних функцій підсистеми тендера можна віднести такі як моніторинг поточної завантаженості мережі, прийняття рішення хто із провайдерів отримає можливість зберігати ті чи інші дані.

Моніторинг завантаженості мережі надає можливість визначити хто із провайдерів сховищ та з яким пріоритетом ввійде до тендеру. Найпріорітетніші це ті хто має найменшу відносну завантаженість – завантаженість розраховується як відношення вільного місця до виділеного, та найвищий рейтинг (у наступному підрозділі описано підсистему рейтинга). Формується список кандидатів з урахуванням необхідної надлишковості й перевіряється їх доступність у мережі. У разі якщо набрано необхідну кількість кандидатів, тендер вважається завершеним й починається завантаження блоків. Якщо недостатньо кандидатів, тендер бере ще необхідну кількість кандидатів й знову починає опитування. Якщо після декількох спроб список кандидатів так і не сформовано оскільки вже неможливо задовольняти необхідний рівень доступності даних, тендер вважається програним та користувачеві пропонується понизити бажаний рівень доступності даних.

2.6.4 Підсистема рейтингу для провайдерів сховищ

До головних функцій підсистеми рейтингу можна віднести такі як відстеження щодо кількості збережених блоків, пройдених аудитів за хешем та за повним блоком.

При реєстрації провайдера сховища у системі йому одразу нараховується 100 у.о. рейтингу. За кожен виділений 1 ГБ простору він отримує 10 у.о. За кожен збережений 1 ГБ та 1 годину доступу до нього він отримує 30 у.о. Після пройденого аудиту за хешом провайдер отримує 50 у.о., за повним блоком – 100 у.о. Штрафні санкції у разі не дотримання умов контракту знімається 100 у.о. за кожен 1 ГБ який користувач хотів отримати. Розглянемо розрахунок рейтингу на прикладі.

При реєстрації я як провайдер сховища отримую 100 у.о. та виділяю 100 ГБ вільного місця отже на моєму рахунку вже 100 + 10 * 100 ГБ = 1100 у.о. Далі мені надсилають 32 ГБ даних, в середньому моє сховище працює 15 годин у добу, отже за місяць (28 днів) я отримаю за зберігання 32 ГБ * 15 * 28 * 30 = 403 200 у.о. рейтингу, що значно вплине на мої шанси у тендері за дані. Також варто не забувати за аудити у середньому я проходжу в день 10 аудитів за хешом та 3 аудити за блоком даним, отже за 1 день я отримую 10 * 50 + 100 * 3 = 800 у.о.

2.6.5 Підсистема забезпечення оплати та виплат

До головних функцій підсистеми оплати та виплат можна віднести такі як можливість поповнення або виведення кошт власного аккаунту – це доступно як для клієнта так і для провайдера сховища.

Задля забезпечення зручності оплати передбачається інтеграція як з фіатними так і криптовалютними сервісами котрі підтримую наступні мережі: з таких найпопулярніших фіатних мереж за допомогою банківської карточки як VISA, MasterCard, Maestro, банківських переказів таких як SWIFT, FasterPayments, SEPA та підтримка найпопулярніших мереж у світі криптовалют таких як Bitcoin, Ethereum, Ripple [26]. Задля введення грошей до системи необхідно ввести дані котрі потребує той чи інший сервіс відповідно. Для виведення достатньо вказати реквізити чи адресу отримувача.

2.6.6 Підсистема обміну валют

До головних функцій підсистеми обміну валют можна віднести такі як отримання поточних курсів валютних пар та здійснення обміну грошей на біржах.

Варто відзначати, що надається можливість обміну між фіат-фіат, крипто-крипто, крипто-фіат, фіат-крипто валютами – це надає зручність клієнту та провайдеру сховища, оскільки вирішує проблему виводу грошей із системи та їх вводу. Планується також розробка власної криптовалюти та підтримка усіх можливих пар.

2.7 Безпека даних

У сучасному інформаційному світі кожен здійснюються напади на інформаційні системи. Ціллю цих нападів зазвичай є чи грошова нажива чи викрадення цінних даних. Оскільки наша система призначена для збереження даних, то було введено ряд мір при проектування задля забезпечення безпеки даних.

Для комунікації між клієнтом та сервером використано протокол HTTPS. HTTPS (від англ. HyperText Transfer Protocol Secure – безпечний протокол передачі гіпертексту)

– це розширення протоколу HTTP, що підтримує шифрування за допомогою криптографічних протоколів SSL та TLS [27]. Відмінності у порівнянні з HTTP:

- HTTPS не є окремим протоколом передачі даних, а представляє собою розширення протоколу HTTP з надстройкою шифрування;
- передані по протоколу HTTP дані не захищені, HTTPS забезпечує конфіденційність інформації шляхом її шифрування;
- HTTP використовує порт 80, HTTPS – порт 443.

Розглянемо безпеку даних у розрізі їх зберігання. Перед відправкою клієнт шифрує інформації власним ключем із використання симетричного алгоритму шифрування – не має можливості розшифрувати їх без ключа. Далі файли розбиваються на блоки котрі розподіляються між різними провайдерами сервісами, що робить неможливим зрозуміти, що це за дані. Також це захищає дані від несанкціонованого втручання правових органів, оскільки вони можуть бути розподілені між різними країнами чи навіть континентами.

2.8 Масштабованість

Масштабованість – здатність пристрою збільшувати свої можливості шляхом нарощування числа функціональних блоків чи апаратно їх вдосконалювати, що виконують одні й ті ж завдання. Зазвичай про масштабування починають думати тоді, коли один сервер не справляється з покладеною на нього роботою. Робота будь-якого web-сервера за великим рахунком зводиться до основного заняття комп'ютерів – обробці даних. Відповідь на HTTP (або будь-який інший) запит мається на увазі проведення деяких операцій над якимись даними. Відповідно, у нас є дві основні сутності – це дані (що характеризуються своїм обсягом) і обчислення (що характеризуються складністю). Сервер може не справлятися зі своєю роботою через великий обсяг даних (вони можуть фізично не поміщатися на сервері), або через велику обчислювального навантаження. Мова тут йде, звичайно, про сумарне навантаження – складність обробки одного запиту може бути невелика, але велика їх кількість може «завалити» сервер.

Існують два способи масштабування: горизонтальне та вертикальне [28]. При вертикальному масштабування здійснюється посилення потужностей самих вузлів чи окремих компонентів системи, наприклад, збільшується об’єм оперативної пам’яті, кількість процесорі чи замінюється HDD на SSD.

При горизонтальному масштабуванні збільшується кількісна характеристика системи. Наприклад, збільшується кількість вузлів серверного обладнання, деякі компоненти системи переїзжають на окремі сервера.

Якщо порівняти вертикальне та горизонтальне підходи до масштабування систем, то очевидно, що перше – це найпростіше рішення й не потребує ніяких змін в архітектурі додатку чи заздалегідь закластися при проектуванні системи. При горизонтальному ж масштабуванні необхідно заздалегідь закладувати таку можливість при проектуванні й писати код з урахуванням того, що він може виконуватися на різних копіях вузлів й стан може бути розподілений між ними. Незважаючи на складність другого підходу, він надає потенційно необмежені можливості щодо розширення системи, на відміну від першого тому, що рано чи пізно досягається максимум можливих обчислюваних потужностей чи це становиться занадто дорого для компанії.

Розподілити по декількох серверах можна не тільки роботу коду, а й сервіси збереження даних (мова йде про бази даних). Якщо СУБД виконує багато складних запитів, займаючи процесорний час сервера, можна створити кілька копій бази даних на різних серверах. При цьому виникає питання синхронізації даних при змінах, і тут інсує кілька підходів.

Синхронізація на рівні додатку – у цьому випадку наш код самостійно записує зміни на всі копії бази даних. Це не кращий варіант, оскільки він вимагає обережності при реалізації і вельми нестійкий до помилок.

Реплікація – тобто автоматичне копіювання змін, зроблених на одному сервері, на всі інші сервера. зазвичай при використанні реплікації зміни записуються завжди на один і той же сервер – його називають master, а решта копії – slave. У більшості СУБД є вбудовані або зовнішні інструменти для досягнення реплікації. Розрізняють синхронну реплікацію – в цьому випадку запит на зміну даних чекатиме, поки дані будуть скопійовані на всі сервери, і лише потім завершиться успішно та асинхронну – в цьому

випадку зміни копіюються на slave-сервера з затримкою, зате запит на запис завершується швидше.

Multi-master реплікація – цей підхід аналогічний попереднього, проте тут ми можемо здійснювати зміну даних, звертаючись ні до конкретного серверу, а до будь-якої копії серверу бази даних. При цьому зміни синхронно або асинхронно потраплять на інші копії.

2.9 Розгортання та експлуатація

Задля спрощення процесу розробки, розгортання та підтримки системи планується використання хмарних обчислень. У додатку Є зображено діаграму розгортання на п’яти вузловому кластері. Розглянемо декілька популярних хмарних провайдерів.

Amazon Web Services – це хмарний провайдер для побудови бізнес-рішень із використанням інтегрованих веб-сервісів [29]. AWS пропонує широкий спектр послуг IaaS (Infrastructure-as-a-Service) та PaaS (Platform-as-a-Service). До них відносяться Elastic Cloud Compute (EC2), Elastic Beanstalk, Simple Storage Service (S3) та Relational Database Service (RDS). AWS пропонує широкі адміністративні елементи керування, доступні через їх веб-клієнт. Користувачі можуть отримати доступ до низки функцій, включаючи створення ключів шифрування та аудиту.

AWS має три різні моделі ціноутворення: «Pay-as-you-Go», «Save when you reserve» та «Pay less using more». Щоб отримати додаткову інформацію про них, користувачі повинні звернутися безпосередньо до відділу продажу. AWS також пропонує безкоштовний 12-місячний доступ до їх послуг. Після завершення пробного періоду ви повинні вибрати платний план або скасувати підписку.

Microsoft Azure надає широкий вибір рішень, придатних для всіх видів систем [30]. Усі потреби бізнесу будуть враховані. Azure означає відсутність необхідності мати фізичні сервери. Це зменшує звичайні витрати, такі як команда підтримки для обслуговування серверної частини.

Microsoft Azure пропонує безкоштовний 12-місячний рівень, який включає доступ до всіх популярних послуг, кредит у розмірі 200 доларів США та понад 25 послуг "Always

Free". Всі ціни та плани Microsoft Azure детально викладені на їх сайті. Сторінка містить калькулятор вартості та послугу "Pay as you go". Кожен план може бути адаптований до необхідних потреб.

Google Cloud Platform є постачальником хмарних послуг Google [31]. Ця платформа дозволяє користувачам створювати бізнес-рішення за допомогою наданих Google модульних веб-сервісів. Він пропонує широкий спектр послуг, включаючи рішення IaaS та PaaS.

Завдяки багатошаровій інфраструктурі Google Cloud користувачі можуть бути впевнені, що все, що ви будуєте, створюєте, розробляєте чи зберігаєте, буде захищено. Це досягається завдяки прихильності прозорості та висококваліфікованій команді інженерів. Google Cloud має різні інструменти для забезпечення продуктивності та керування. До них відносяться Compute Engine, App Engine, Container Engine, Cloud Storage Big Query. Google також пропонує плавне переміщення на віртуальні машини з гнучкою ціною.

Існує безкоштовна 12-місячна пробна версія, яка включає в себе 300 доларів США на всі послуги та продукти, запропоновані Google Cloud Platform.

Оскільки планується використовувати стек технологій Microsoft тож й оберемо Microsoft Azure тому, що він пропонує максимальну інтеграцію зі своїм стеком технологій. Microsoft Azure надає наступні можливості, що необхідні для нашої системи: Azure Service Fabric, Azure Event Hubs, Azure Notification Hubs.

Azure Service Fabric – це платформа розподілених систем, що полегшує розробку, розгортання та керування масштабованими та надійними мікросервісами та контейнерами. Він також вирішує важливі проблеми, пов'язані з розробкою та керуванням хмарними додатками. Розробники та адміністратори можуть уникати складних проблем в галузі інфраструктури та зосередити увагу на реалізації критично важливих завдань, що потребують масштабованості, надійності та керованості.

Azure Event Hubs – це повнофункціональна, надійна та легка у масштабуванні послуга для прийому даних у реальному часі. Надає можливість здійснювати потік мільйонів подій у секунду з будь-якого джерела для побудови динамічних pipelines даних та мінімальними затримками виконувати бізнес-логіку. Надає можливість обробки даних

під час надзвичайних ситуацій, використовуючи функції відновлення у разі катастрофи (geo-disaster recovery) та гео-реплікації (geo-replication).

Azure Notification Hubs – широко масштабуєма підсистема push-сповіщень, котра здатна швидко відправляти мільйоні повідомлень на прилади, що працюють на iOS, Android, Windows або Kindle [19]. Оскільки, у подальшому планується написання мобільних клієнтів, то цей сервіс значно поліпшує процес відправки сповіщень до користувачів.

Окрім цього, використання хмарних обчислень надає легкість у розгортанні, а саме завдяки підходу PaaS (Platform-as-a-Service) у декілька кліків можна розгорнути всю систему. Також, в часи високого попиту на нашу систему, ми можемо за декілька хвилин підняти декілька серверів, щоб збалансувати навантаження, що неодмінно є однієї з головною перевагою використань хмарних сервісів.

2.10 Необхідні характеристики підтримуючого технічного обладнання

Рекомендується використати обладнання яке має наступні технічні характеристики:

А) Для мобільного клієнта

- процесор з тактовою частотою не менше 1.8 ГГц;
- об’єм оперативної пам’яті від 128 МБ;
- наявність 64 МБ вільного місця на зберігаючому пристрої;
- операційна система Android 8 чи iOS 11 та старше;

Б) Для веб-клієнту

- мінімум процесор AMD Ryzen 5 чи Intel Core i5 з тактовою частотою не менше 2 ГГц;
- об’єм оперативної пам’яті від 1 ГБ;
- наявність браузера з підтримкою WebSocket та ECMAScript 6.

В) Для настільного клієнта

- наявність інтегрованої чи дискретної відеокарт;
- процесор з тактовою частотою не менше 2 ГГц починаючи з AMD Ryzen 5 чи Intel Core i5 та старше;
- наявність 300МБ вільного місця на зберігаючому пристрої;
- операційна система Windows 10;
- встановлений .NET Framework 4.7.1.

Г) Для провайдера сховища

- мінімум процесор AMD Ryzen 5 чи Intel Core i5 з тактовою частотою не менше 2 ГГц;
- наявність 5 ГБ вільного місця на жорсткому диску;
- об’єм оперативної пам’ять від 4 ГБ;
- наявність надійного інтернет з’єднання зі швидкістю 100 Mbps та вище;
- операційна система Linux, Windows 10, Windows Server 2016, OS X;
- встановлений .NET Core 2.1;

Д) Для серверної частини

Кластер мінімум із п’яти вузлів, с наступними характеристиками для кожного:

- об’єм оперативної пам’ять від 32 ГБ;
- інтернет-з'єднання зі швидкістю 1 Gbps та вище;
- наявність 30 ГБ вільного місця на SSD диску;
- наявність відеокарти;
- операційна система Windows Server 2016;

Е) Для серверу баз даних

- об’єм оперативної пам’ять від 64 ГБ;
- інтернет-з'єднання зі швидкістю 1 Gbps та вище;
- наявність 100 ГБ вільного місця на SSD диску;
- наявність відеокарти;
- операційна система Windows Server 2016;
- встановлений MS SQL Server 2016

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Вибір стеку технологій та інструментів розробки

Для розробки системи використовувалися наступні технології та інструменти такі як C#, ASP .NET Core, Enitity Framework Core, WPF.

C# – це мова програмування призначена для розробки найрізноманітніших додатків, призначених для виконання в середі .NET Framework. Мова C# проста, типобезпечна та об’єктно-орієнтовна Завдяки багатьох нововведень C# забезпечує можливість швидкої розробки додатків і при цьому зберігає елегантність C-подібних мов. З останніми тенденціями та нововведеннями такими як .NET Core та .NET Standard надає можливість кросплатформенної розробки як і при розробці під ОС Windows.

ASP .NET Core – це новий кросплатформений фреймворк з відкритим вихідними кодом («open-source») для розробки сучасних хмаро орієнтованих додатків, таких як веб-застосування, IoT (Internet of Things) додатків та серверних частин для мобільних додатків [32].

ASP .NET Core надає наступні основні можливості:

- розробка графічного веб інтерфейсу та Web Api як єдиного цілого;
- інтеграцію з сучасними клієнтськими фреймворками;
- систему конфігурацій, що забезпечує підтримку хмарних обчислень;
- вбудований IoC-контейнер;
- нову модульну обробку HTTP запитів;
- новий набір інструментів, що полегшують веб розробку;
- можливість запуску ASP .NET додатків на таких ОС як Windows, Mac, Linux.

Entity Framework (EF) Core – це легковісна та кросплатформена версія популярного EF, що надає доступ до даних [33]. Це об’єктно-реляційне відображення, що надається розробникам .NET для роботи з БД. Він відображує реляційні таблиці на об’єкти та зменшує обсяг коду для взаємодії з БД. Надає можливість вибірки та додавання/оновлення/видалення даних з БД.

Windows Presentation Foundation (WPF) – система для побудови клієнтських додатків Windows з візуальними можливостями комунікації з користувачем [34]. В основі WPF закладено векторну систему візуалізації, що не залежна від приладу виводу та створена з урахуванням можливостей сучасного графічного обладнання. WPF надає інструментарій для створення візуального інтерфейсу, що містить в основі мову XAML (eXtensible Application Markup Language) зокрема, елементи управління, прив'язку даних, макети, двомірну та трьох мірну графіку, анімацію, стилі, шаблони і т. д.

3.2 Розробка концептуальної моделі

У додатку Ж зображено повну концептуальну модель системи виконаної за допомогою нотації UML діаграми класів. Умовно її можна поділити на три частини такі як персональні дані користувача, дані об завантажених файлових даних та фінансова частина.

Розглянемо відношення котрі стосуються файлових даних. На рисунку 3.1 зображено відношення котрі необхідні для завантаження файлу – приймання заявки й проведення тендеру та місцезнаходження блоків файлу.

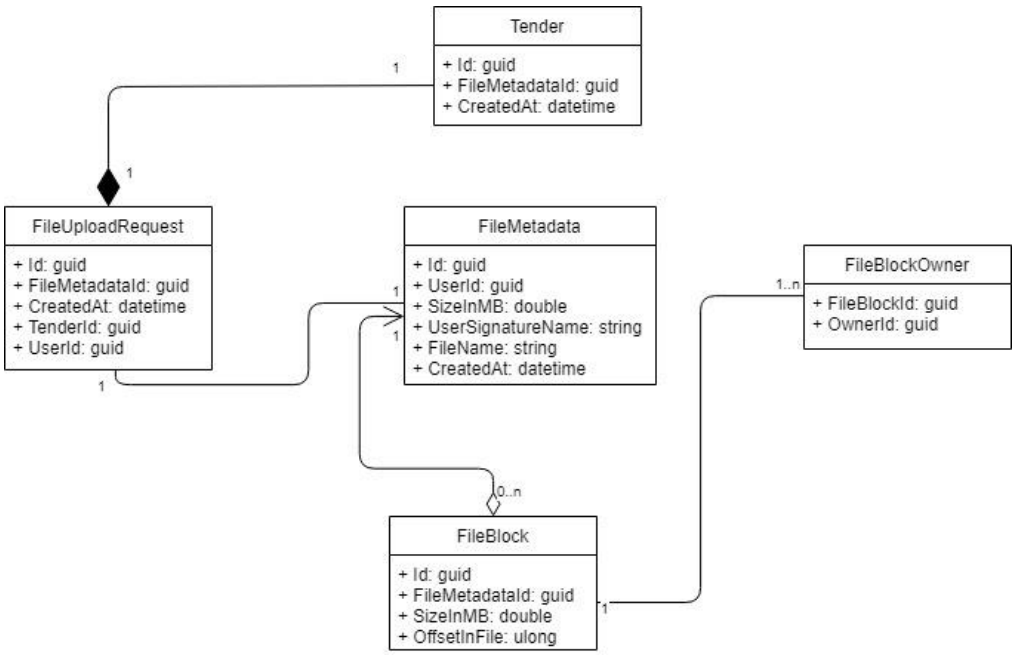


Рисунок 3.1 – Відношення сутностей для завантажених файлів

Перед завантаженням файлу створюється відповідний запит на завантаження «FileUploadRequest» та записуються метадані файлу «FileMetadata». Після того як проведено «Tender», файл розбивається на блоки «FileBlock», які надсилаються до відповідних провайдерів сховищ «FileBlockOwner»

Розглянемо завантаження та зберігання файлу з фінансової сторони. На рисунку 3.2 зображено відношення та сутності котрі необхідні для підтримки процесу оплати за зберігання – сплата користувачем, розподіл грошей між сервісом та провайдерами сховищ.

Після того як заявка на завантаження була прийнята та було проведено тендер з рахунку користувача списується необхідний об’єм кошті «HoldTransaction» за попередньо укладеним контрактом «StorageContract» та переводяться ці кошти до транзитного рахунку «TransitBalance». Надалі за умовами контракту через певні періоди часу здійснюється часткова сплата послуг, а саме нарахування винагороди провайдеру сховища «RewardTransaction» та комісія сервісу «ServiceFeeTransaction».

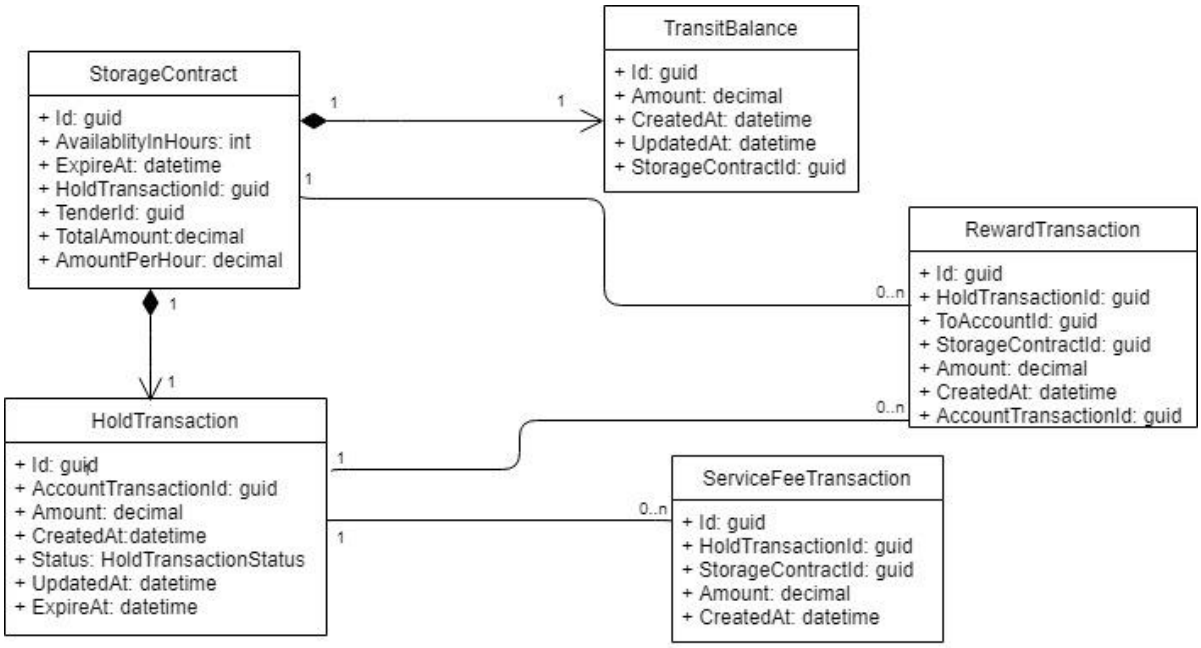


Рисунок 3.2 – Відношення сутностей для фінансової частини процесу завантаження файлу

3.3 Проектування схеми бази даних

Було обрано як сервер бази даних Microsoft SQL Server, а саме його хмарну версію Azure SQL Server, оскільки планується розгортати всю інфраструктуру в Azure.

База даних SQL Azure – це повністю керована інтелектуальна служба реляційної хмарної бази даних, яка надає розширену сумісність з ядром SQL Server. Microsoft пропонує кілька варіантів розгортання:

- розгорнути єдину базу даних на логічний сервер;
- розгорнути на еластичний пул (elastic pool) на логічному сервері, щоб розподілити ресурси та зменшити витрати;
- розгорнути на Managed Azure SQL Database Instance.

Хмарна версія SQL Server спрощує процес адміністрування, надає можливість у короткий час підвищити потужність серверу – достатньо лише «потягнути повзунок» й має вбудовані інструменти аналізу перфомансу та його автоматичного підвищення.

Зазвичай при проектуванні схеми бази даних оперують поняттями нормальних форм. Нормалізація передбачає застосування нормальних форм до структури даних. Існують 7 нормальних форм. Кожна нормальна форма (за винятком першої) має на увазі, що до даних уже була застосована попередня нормальна форма. Наприклад, перш ніж застосувати третю нормальну форму до даних повинна бути застосована друга нормальна форма. Слід сказати, база даних вважається нормалізованої, якщо до неї застосовується третя нормальна форма і вище. Розглянемо перші три форми:

- перша нормальна форма (1NF) передбачає, що зберігаються дані на перетині рядків і стовпців повинні представляти скалярний значення, а таблиці не повинні містити повторюваних рядків;
- друга нормальна форма (2NF) передбачає, що кожен стовпець, який не є ключем, повинен залежати від первинного ключа;
- третя нормальна форма (3NF) передбачає, що кожен стовпець, який не є ключем, повинен залежати тільки від первинного ключа.

Розглянемо ключові відношення на рівні баз даних котрі описувалися у розділі 3.1. Оскільки обрано мікросервісну архітектуру, то кожен сервіс має власну базу даних. Усі сутності котрі належать процесу завантаження файла зберігаються у «File Management Database Server», фінансова частина – «Payment Database Server». За замовчуванням обрано схему «dbo».

На рисунку 3.3 зображено відношення між «FileUploadRequests», «Tenders», «FileMetadatas», «FileBlocks» та «FileBlockOwners». Між тендером та зявкою на завантаження відношення один до одного оскільки на кожне завантаження один тендер, між метаданими та блоками відношення один до багатьох тому, що файл розбивається на блоки.

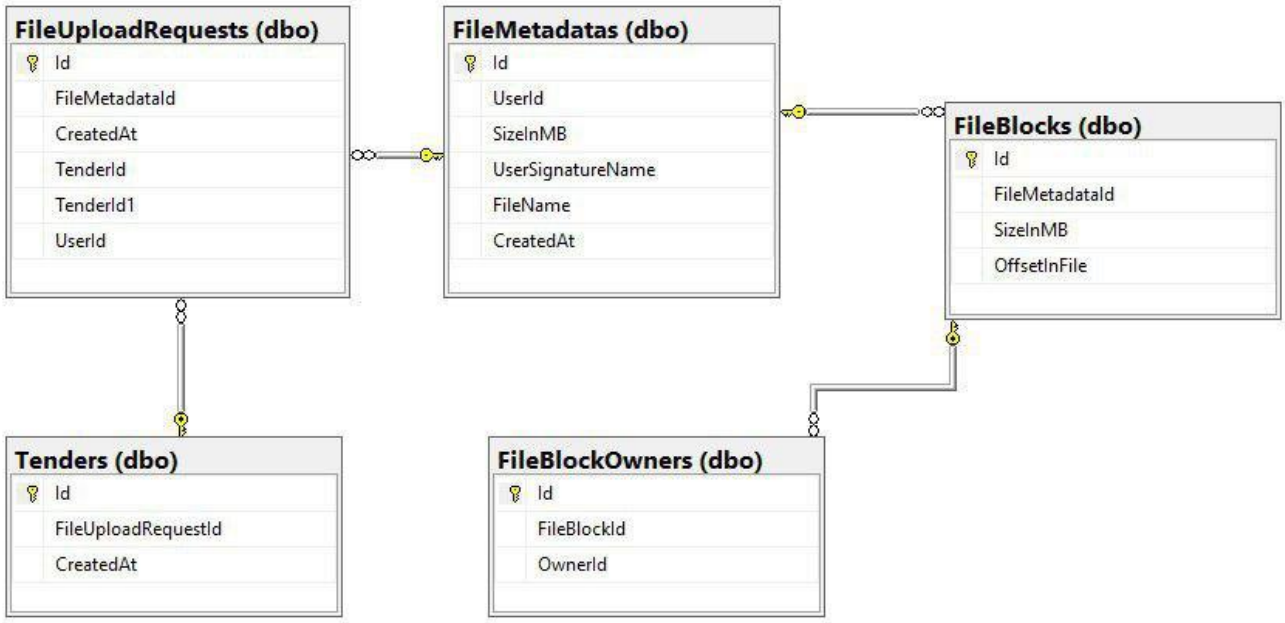


Рисунок 3.3 – Відношення між сутностями для завантаження файлу

На рисунку 3.4 зображено відношення між «StorageContracts», «RewardContracts», «HoldTransactions», «TransitBalances» та «ServiceFeeTransactions». На кожний контракт створюється один транзитний баланс «TransitBalance» котрий утримує кошти клієнта й транзакція списання «HoldTransaction», щоб наперед зняти платню з клієнта. За умовами контракту списується винагорода провайдеру сховища «RewardTransaction» та комісія

сервісу «ServiceFeeTransaction» через певні проміжки часу та вдалого проходження аудиту першими.

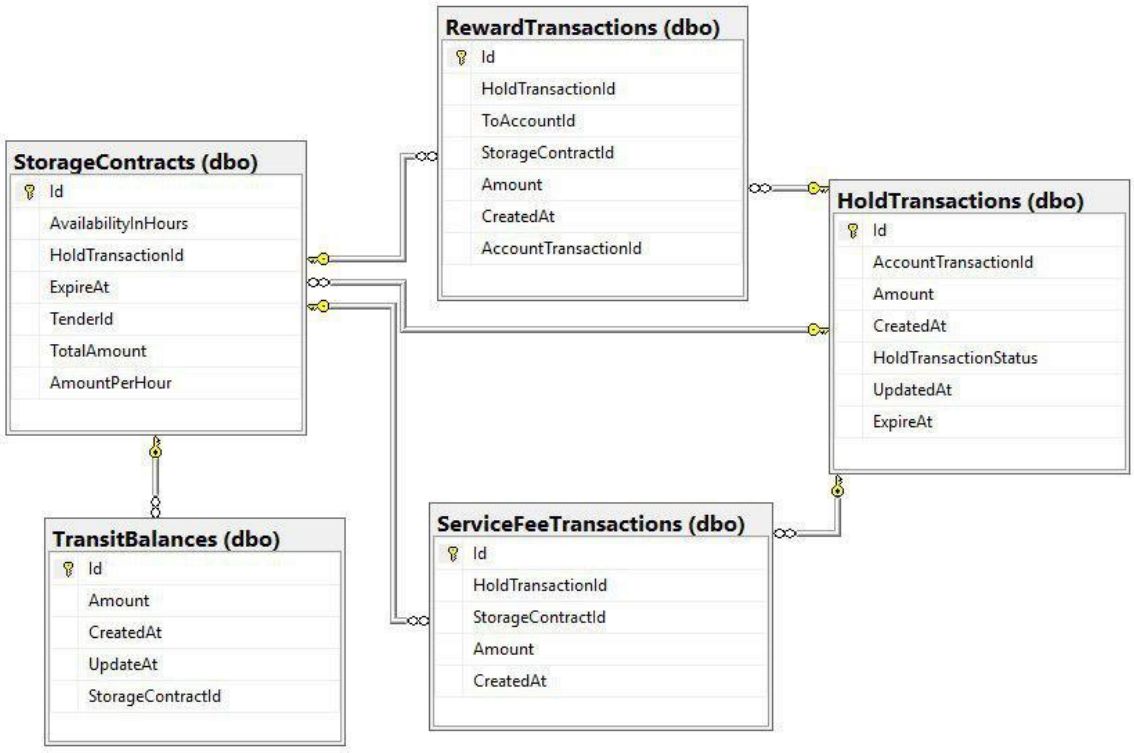


Рисунок 3.4 – Відношення між сутностями для списання грошей для оплати послуг

3.4 Реалізація серверної частини

Для реалізації серверної частини було обрано Azure Service Fabric [35]. Розглянемо деякі ключові поняття цієї технології. Azure Service Fabric спрощує написання та управління stateful та stateless Reliable Services.

Stateless сервіс – це сервіс котрий не має постійного стану всередині себе. Будь-який стан є повністю одноразовим та не потребує синхронізації, реплікації, постійності та високої доступності. Загальний приклад того, як stateless сервіс використовуються в Service Fabric, – це інтерфейс, який надає API для веб-додатку, який доступний для зовнішнього світу. Фасад приймає запит від користувача та перенаправляє до необхідної

партиції stateful сервісу, після того як отримає відповідь, перенаправляє її до оригінального користувача.

Stateful сервіс – це сервіс котрому необхідно для коректного функціонування підтримувати частини стану в узгодженому вигляді. Більшість сервісів сьогодні зберігає свій стан у зовнішніх сховищах, що забезпечує надійність, доступність, масштабованість та узгодженість стану. У Service Fabric необов'язково, щоб сервіси зберігали стан у зовнішніх сховищах. Він відповідає таким вимогам до коду та стану сервісу.

Reliable актори – це платформа сервісів Service Fabric, заснована на шаблоні віртуальних акторів. Їх API надає однопоточну модель програмування, яка базується на надійності та масштабованості, які гарантує Service Fabric.

3.5 Реалізація API фасаду

API фасад виконано з використанням stateless сервісу та технології ASP .NET Core MVC з використанням Web Api для комунікації через HTTPS з користувачами. Раніше Microsoft диференціювало Web Api та MVC на два фреймворки: ASP .NET WebApi та ASP .NET MVC, відповідно, зараз же, маємо один фреймворк ASP .NET Core MVC, який поєднав два попередні фреймворки.

Для побудови Web Api використовувалася REST архітектура. Ми використовуємо чотири головні операції (ще їх називають «action verbs» або «HTTP verbs») GET, POST, PUT, DELETE. Розглянемо їх призначення:

- GET – використовується для операцій отримання даних;
- POST – використовується для операцій створення даних;
- PUT – використовується для операцій зміни даних;
- DELETE – використовується для операцій видалення даних.

За замовчуванням, URL складається із IP, далі номер порту, що відокремлена двокрапкою, потім сегмент з назвою «api», далі ім’я контролера, для визначення який метод викликати використовуються «action verbs», за необхідності створюється ще один

сегмент після назви контроллера. Пізніше, після випуску першої версії системи буде куплено DNS ім’я, що замінить IP адресу та номер порту.

Усі контроллери, що створюються унаслідуються від BaseController (додаток Б), який в свою чергу унаслідується від Controller, який поставляється у фреймворці ASP .NET Core MVC. Ми створили базовий контролер, щоб винести туди всю загальну логіку для всіх дочірніх контролерів.

За замовчуванням, усі користувачі повинні бути автентифікованими у системі – для цього використовується атрибут «AuthorizeAttribute», який застосовано до базового контролера, для доступу до деяких методів не автентифікованим користувачам використовується атрибут «AllowAnonymousAttribute».

До базового контроллеру застосовано ще два основних атрибути «RouteAttribute» та «HandleExceptionFilterAttribute». Перший використовується для задання шаблону маршруту, в нашому випадку, за замовчуванням, це «api/[controller]». Другий використовується для обробки усіх помилок, що можуть статися при обробці запиту. Якщо не використовувати його і виникла помилка при запиті, буде повернена 500 Internal Server Error, що не є достатньо інформативним для користувача, тому завдяки цьому атрибуту, ми повертаємо 400 Bad Request, що містить у тілі відповіді («Response Body») опис помилки.

JSON Web Token (JWT) – це безпечний URL-засіб репрезентації інформації користувача («claims»), яку необхідно передати між двома сторонами [26]. Інформація в JWT кодується як JSON об’єкт, який є являє собою JSON Web Signature (JWS) структуру чи як відкритий текст («plaintext») з шифруванням JSON Web Encryption (JWE) структури, що здійснює цифровий підпис інформації користувача.

Для роботи з JWT токенами використовується бібліотека Microsoft.AspNetCore.Authentication.JwtBearer, яка надає можливість створення токенів з вибором шифрування, назначення видавника тощо.

При збільшення кількості копій сервісу для горизонтального масштабування постає питання синхронізації сесій. Одна з головних переваг використання токенів це те, що вони не зберігають стан (stateless). Цим самим токени полегшують процес горизонтального масштабування.

Для впровадження таких функцій як реєстрація, додавання до ролі та інше використовується бібліотека ASP .NET Core Identity, що має реалізацію цих та інших функцій. ASP .NET Core Identity надає вбудовану в ASP .NET систему автентифікації та авторизації. Дана система надає можливість користувачам створювати облікові записи, аутентифікуватися, керувати обліковими записами або використовувати для входу у систему облікові записи зовнішніх провайдерів таких як Facebook, Google, Microsoft, Twitter та інших.

ASP .NET Core має вбудований IoC-контейнер, що надає можливість ставити у відповідність абстракцію та реалізацію з вибором стратегії життєвого циклу об’єкта. Виділяють три основні стратегії:

- transient – на кожен запит до контейнеру створюється об’єкту;
- scoped – на кожен запит до контейнеру в рамках однієї сесії створюється один об’єкт;
- singleton – об’єкт створюється лише один раз.

Веб-додатки ASP .NET розгорталися на веб-серверах IIS. Але, оскільки, ASP .NET Core має кросплатформену природу, виникла необхідність відв’язати ASP .NET Core від IIS та від Windows в цілому. На даний момент ASP .NET Core підтримує розгортання додатку на стандартних веб-серверах IIS та IIS Express, також надає можливість запускати додатки без IIS в рамках власного процесу за допомогою двох додаткових HTTP-серверів, котрі йдуть в комплекті з ASP .NET Core:

- Microsoft.AspNetCore.Server.WebListener (чи просто WebListener);
- Microsoft.AspNetCore.Server.Kestrel (чи просто Kestrel).

WebListener працює лише на платформі Windows, а Kestrel є кросплатформеним.

3.6 Реалізація типової архітектури сервісу

Типова архітектура сервісу складається з двох компонентів таких як управління та виконання. У додатку З наведено специфікацію класів для сервісу управління файлами. Для першого обрано stateful сервіс, для другого – Reliable актор відповідно. Сервіс із станом надають нам можливість кешувати агрегати задля підвищення перфомансу системи та зниження кількості запитів до бази даних. Актори наддадуть нам гарантію узгодженості стану й однопоточної обробки запитів на зміну стану й даних у БД.

Розглянемо реалізацію типової архітектури за тришаровим стилем та з урахуванням компонентів.

Шар доступу до даних виконано з використанням таких шаблонів як UnitOfWork та Repository [36] (додаток И). Шаблон Repository (рисунок 3.5) забезпечує роботу з даними БД, наприклад, вибірка, додавання, оновлення, видалення. Для перетворення реляційних даних (дані із БД) у об’єкти використовується Entity Framework Core (див. розділ 3.1). Він надає можливість перетворення C# лямбда-виражень у SQL запити, що полегшує процес розробки.

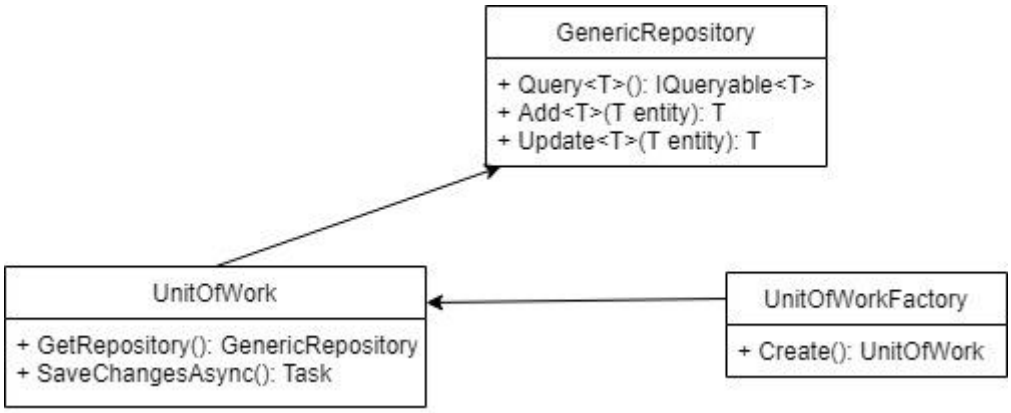


Рисунок 3.5 – Діаграма класів UnitOfWork та Repository

Шаблон UnitOfWork виконує роль контексту взаємодії з БД. В нашій серверній частині, під контекстом слід розуміти проміжок часу від часу надходження запиту до його обробки (час відправлення відповіді). Тобто, контекст існує в цьому часовому проміжку часу та є індивідуальним для кожної сесії запит-відповідь. Він надає

можливість виконати операції вибірки, створення, редагування та видалення даних для будь-якої сутності, а потім після виклику методу SaveChanges, що також є членом даного класу, при необхідності одним чи декількома запитами зафіксувати зміни.

Варто відзначити, що метод SaveChanges підтримує асинхронну модель взаємодії з використанням CancellationToken, призначений для відміни дії, якщо це можливо, користувача в контексті виконання.

Вище використана абстракція StateRepository який підтримує можливість працювати зі станом та доступне лише компоненту управління. Його задача – це наповнювати стан, у разі відсутності необхідного об’єкта, він звертається до БД через Repository. Repository в компоненті управління використовується лише для читання даних, а в компоненті виконання як для читання так і для запису.

Шар бізнес-логіки відповідає за такі основні операції як перевірка прав доступу, перевірку надісланих даних у відповідності з бізнес-правил, виконання безпосередньо бізнес-логіки та взаємодії між шарами презентації та доступу до даних.

Під перевіркою прав слід розуміти процес, який визначає чи може користувач здійснювати ті чи інші дії. Наприклад, користувач має роль «Клієнт», якщо він здійснить редагування ціни за 1 ГБ даних, то отримає помилку, оскільки він не має на це прав.

Під перевіркою даних у відповідності з бізнес-правилами слід розуміти процес, який визначає чи не порушують отримані дані із шару презентації бізнес-правилам. Наприклад, користувач хочу зареєструватися в системі, у поле «Username» ввів більше 20-ти символів, то він отримає помилку, про те що допускається до 20 символів для даного поля.

Розглянемо організацію класів шару бізнес-логіки для компоненту управління. Кожен клас, який призначено для комунікації з шаром презентації має закінчення «Manager». Дані сутності інкапсулюють роботу з шаром доступу до даних, та виконують вищезгадані операції. Також, за необхідністю, виконує конвертацію сутностей. Наприклад, є сутність «User», що має поле «PasswordHash», користувач не повинен його отримати, тому для цього вводиться сутність «UserInfo», що не містить дане поле, а Manager конвертує із «User» у «UserInfo» та віддає його шару презентації.

У розрізі компоненту виконання кожен клас має закінчення «Service». Його головними задачами є виконання бізнес-процесів й взаємодія із шаром доступу до даних для внесення змін до джерела даних.

Шар презентації виконано за допомогою класів котрі надаються для організації комунікації для Stateful сервісів у Service Fabric й являє собою хостом для RPC звернень. До нього зазвичай звертається публічний фасад для обробки запиту від клієнта, також можуть звертатися інші сервіси у кластері.

3.7 Реалізація клієнтської частини

Було реалізовано клієнтський додаток під ОС Windows із використанням технології WPF. У додатку І наведено специфікацію класів для клієнтської частини. Розглянемо реалізацію тришаровою архітектури у розрізі клієнта.

Шар доступу до даних виконує роль комунікації з публічним API фасадом та серіалізація й десеріалізація запитів й відповідей відповідно. Виділимо три головних класах за допомогою яких реалізовано цей шар: HttpClient, ApiClient, Resource.

HttpClient – це набір методів обгорток для формування HTTP запитів для спрощення написання коду й форматування запитів й відповідей. Має підтримку основних action verbs таких як GET, POST, PUT, DELETE.

ApiClient – це агрегатор ресурсів (Resource) та управляє створенням їх реалізацій.

Ресурси призначені для відображення у програмному коді викликів до відповідних методі на публічному API фасаді. Для здійснення викликів використовують HttpClient. Найменування методів та ресурсів є відображенням контролерів та екшенів на серверній частині.

Шар бізнес-логіки для валідації вхідних даних, перевірки на коректність даних для поточного бізнес процесу та слугує для передачі даних від шару презентації до шару доступу до даних й у зворотному направленні.

Шар презентації займається візуалізацією користувацького інтерфейсу з використанням шаблону MVVM (рисунок 3.4) [37]. Паттерн MVVM (Model-View-

ViewModel) дозволяє відокремити логіку застосування від візуальної частини (представлення).

Модель описує використовувані в додатку дані. Моделі можуть містити логіку, безпосередньо пов'язану цими даними, наприклад, логіку валідації властивостей моделі. У той же час модель не повинна містити ніякої логіки, пов'язаної з відображенням даних і взаємодією з візуальними елементами управління.

View або представлення визначає візуальний інтерфейс, через який користувач взаємодіє з додатком. Стосовно до WPF представлення – це код в xaml, який визначає інтерфейс у вигляді кнопок, текстових полів та інших візуальних елементів.

ViewModel або модель представлення пов'язує модель і представлення через механізм прив'язки даних. Якщо в моделі змінюються значення властивостей, при реалізації моделлю інтерфейсу INotifyPropertyChanged автоматично йде зміна відображуваних даних в представленні, хоча безпосередньо модель та представлення не пов'язані. У додатку Ї представлено вихідний код ViewModel для реєстрації користувача.

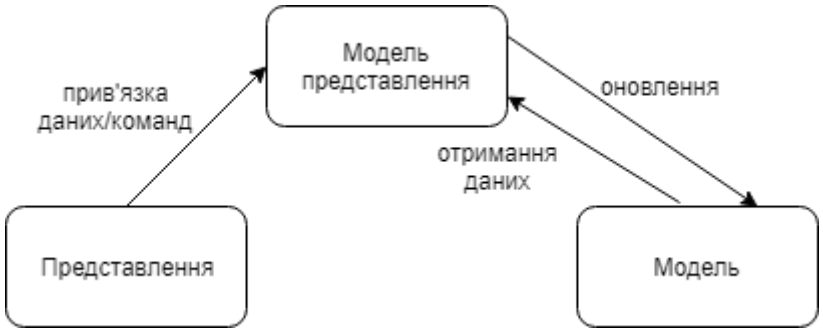


Рисунок 3.6 – Шаблон MVVM

Для забезпечення слабо зв’язності між моделями представленнями реалізовано шаблон Publisher & Subcriber (рисунок 3.5) Він складається з таких взаємодіючих сторін як видавець, шина та споживач. Видавець формує дані, шина містить інформацію про споживачів котрим необхідні ці типи даних, споживачі мають власну логіку щодо обробки цих даних. Видавець має метод Publish котрий надсилає деяку подію (event) до шини, далі шина знаходить усіх споживачів й викликає на них метод Handle.

Рисунок 3.7 – Схема роботи Publisher & Subsriver

При візуальному дизайні клієнтської частини використовувався «матеріальний дизайн» (Material Design). Матеріальний дизайн є дизайнерською мовою, розробленою компанією Google у 2014 році. Розширюючи мотиви «card», які дебютували в Google Now, Material Design використовує більше макетів на основі сіток, м'якої анімації та проміжних переходів, та глибинні ефекти, такі як освітлення та тіні. Його використовують такі відомі компанії як Google, WhatsApp, Apk Mirror, YouTube та інші.

3.8 Організація типової структури модулів для сервісу

Для розробки системи використовувалося середовище Microsoft Visual Studio.

Microsoft Visual Studio – це інтегроване середовище розробки додатків, що рощроблено компанією Microsoft та інших засобів, які надаються у вигляді інструментів [29]. Цей продукт надає можливість розробляти програми різних видів таких як консольних, графічних (WPF, Windows Forms), веб-сайти, веб-додатки (ASP .NET, ASP .NET Core) з використанням різних мов програмування. З останніми тенденціями у хмарних обчисленнях, це середовище надає інтеграцію з Microsoft Azure.

На рисунку 3.6 наведено типову структуру модулів у Visual Studio. Логічно розбито усі проекти на Core (загальні модулі), Management (модуль управління), Operations (модуль виконання – актор), Tests (модульні та інтеграційні тести).

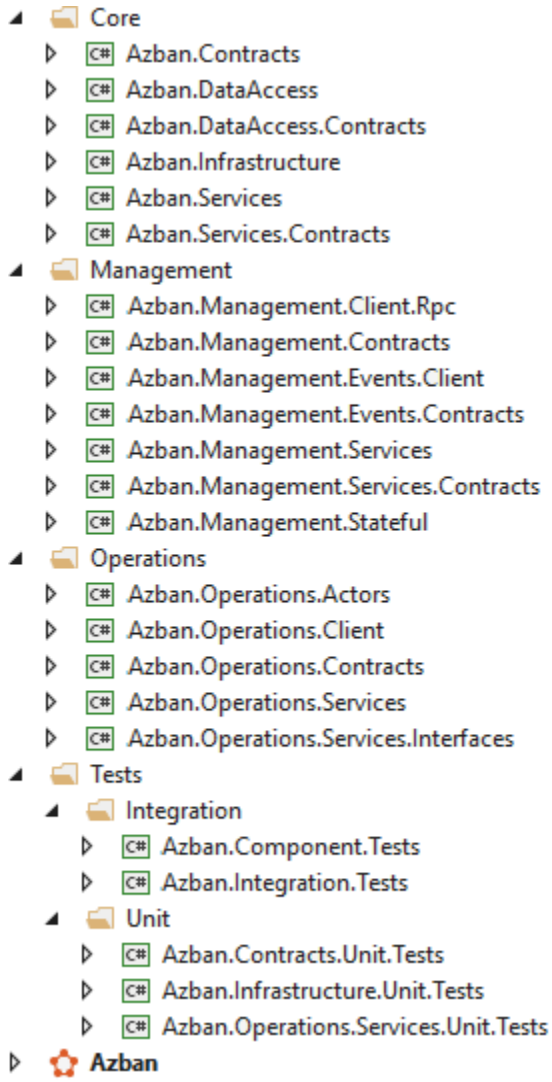


Рисунок 3.8 – Типова структура модулів

3.9 Принципи розробки системи

При розробці системи використовувалися такі підходи та принципи: ООП, SOLID, DRY, KISS, YAGNI та coding guidelines, які рекомендує Microsoft [38].

ООП (об’єктно-орієнтоване програмування) – методологія програмування, що заснована на представленні програми у вигляді сукупності об’єктів, кожен з яких являє собою екземпляр деякого класу, а класи утворюють ієрархії наслідування. Можна виділити три основних парадигми ООП:

– інкапсуляція – властивість системи, яка надає можливість об’єднувати дані та методи, що працюють з ними, в класі та приховати деталі реалізації від користувача;

- наслідування – властивість системи, яка надає можливість описати новий клас, що базується на існуючому з частковою або повною функціональністю;
- поліморфізм – властивість системи, яка надає можливість використовувати об’єкти з однаковими інтерфейсами без інформації про тип та внутрішню структуру об’єкта.

SOLID – являє собою акронім для набору практик проектування програмного коду, побудові гнучкої та адаптивної програми. Сам акронім складається з перших літер назв SOLID-принципів:

- Single Resposibility Principle (Принцим єдиного обов’язку);
- Open/Closed Principle (Принцип відкритості/закритості);
- Liskov Substitution Principle (Принцип підстановки Лісков);
- Interface Segregation Principle (Принцип поділу інтерфейсів);
- Dependency Inversion Principle (Принцип інверсій залежностей).

Принцип єдиного обов’язку – у кожного класу повинна бути тільки одна причина для зміни. Під обов’язком тут слід розуміти набір функцій, які виконують єдину задачу. Суть цього принципу полягає в тому, що клас повинен виконувати одну єдину задачу. Весь функціонал класу повинен бути цілісним та мати високу зв’язність (high cohension).

Принцип відкритості/закритості – сутності програми повинні бути відкритими для розширення, але закритими для зміни. Суть цього принципу полягає в тому, за всі її наступні зміни повинні бути реалізовані за допомогою додавання нового коду, а не зміною вже існуючого.

Принцип підстановки Лісков – повинна бути можливість замість базового типу підставити будь-який його підтип. Наприклад, клас S може вважатися підкласом T, якщо заміна T на об’єкти S не призведе до зміни роботи програми.

Принцип поділу інтерфейсів – клієнти не повинні примусово залежати від методів, якими не користуються. При порушенні цього принципу клієнт, який використовує

деякий інтерфейс з усіма його методами, залежить від методів, якими не користується, тому він буде сприйнятливим до змін в цих методах.

Принцип інверсій залежностей – модулі верхнього рівня не повинні залежить від модулей нижнього рівня. Обидва повинні залежити від абстракцій.

DRY (не повторюйся, Don’t Repeat Yourself) – цей принцип полягає в тому, что потрібно уникати повторень одного й тож самого коду.

KISS (не ускладнюй, Keep It Simple, Stupid) – цей принцип полягає в тому, що потрібно робити максимально просту та зрозумілу архітектуру, застосовувати шаблони проектування та не винаходити велосипед.

YAGNI (це вам не знадобиться, You Ain’t Gonna Need It) – цей принцип полягає в тому, що необхідно реалізовувати тільки поставлені задачі та відмовитися від надлишкового функціоналу.

4 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

4.1 Керівництво користувача

Керівництво користувача передбачає опис головних функцій клієнтського додатка та системи в цілому таких як автентифікація, реєстрація, завантаження файлу, їх перегляд та отримання, створення власних ключів шифрування.

При запуску настільного клієнта першим відкривається форма автентифікація користувача (рисунок 4.1) у якій необхідно ввести «Email» та «Password».

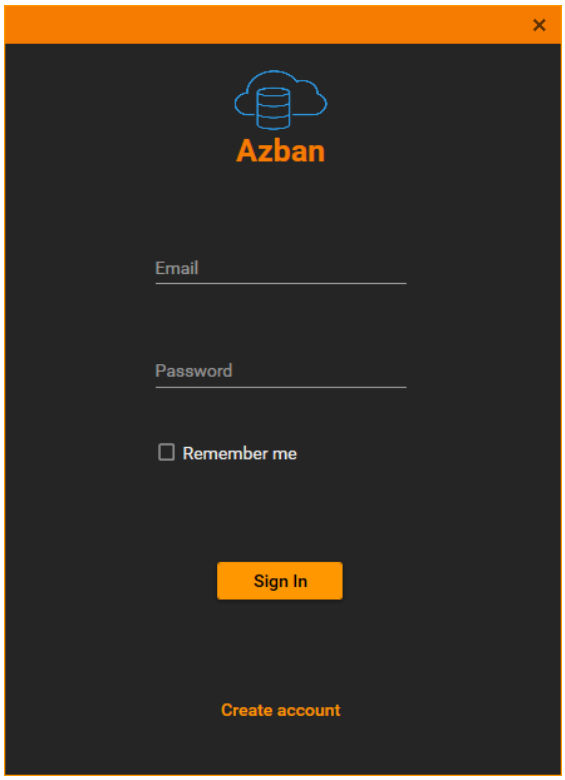


Рисунок 4.1 – Форма автентифікації користувача

Якщо користувач не має облікового запису у системі – він може його створити натиснувши кнопку «Create account» й заповнивши відповідні поля (рисунок 4.2).

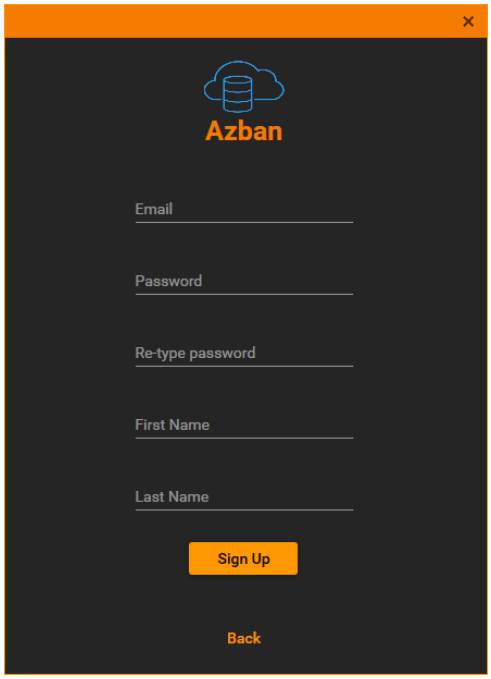


Рисунок 4.2 – Форма реєстрація нового користувача

Після успішної автентифікації користувача, відкривається головна форма з відображення всіх завантажених файлів у систему (рисунок 4.3).

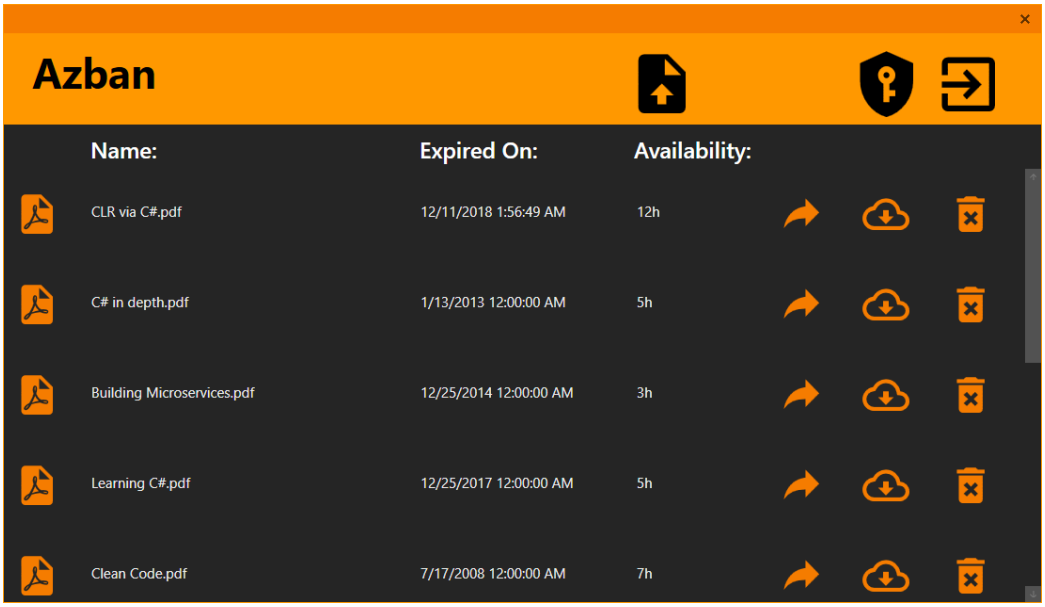


Рисунок 4.3 – Головна форма

Для завантаження файлу необхідно натиснути кнопку «Upload file», обрати необхідний файл, вибрати ключ шифрування, термін дії контракту та рівень доступності даних (рисунок 4.4).

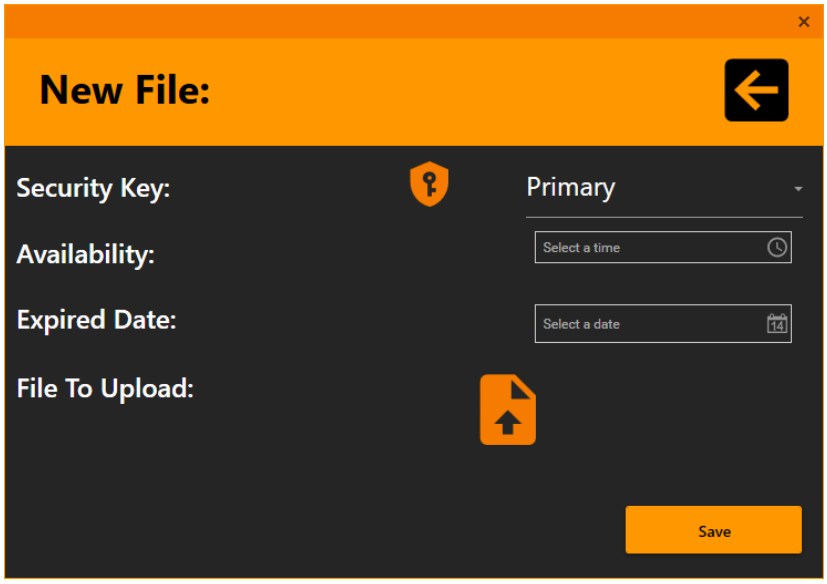
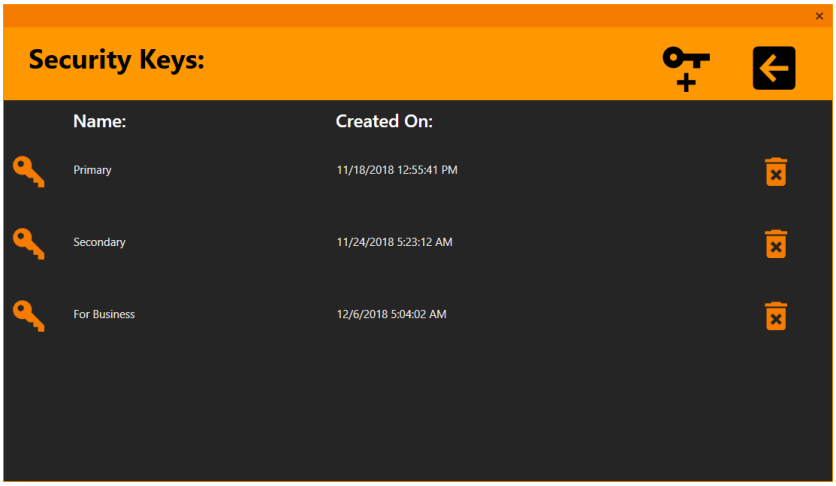


Рисунок 4.4 – Форма завантаження файлу

На рисунку 4.5 зображено графічну форму управління власними ключами шифрування.



Рисунку 4.5 – Форма управління ключами шифрування

Для додавання нового ключа необхідно натиснути кнопку «Add key», відкриється форма створення ключа у якій необхідно вказати назву ключа (рисунок 4.6).

Рисунок 4.6 – Форма створення ключа шифрування

4.2 Мануальне тестування

Під час мануального тестування було перевірено основні функції системи. Нижче наведено перелік випробувань та отриманий результат.

Таблиця 4.1 – Тестовий сценарій «Автентифікація у системі»

Назва	Тест для перевірки графічної форми автентифікації у системі	
Use case	Автентифікація користувача у системі	
Дія	Очікуваний результат	Результат тесту
Передумова		
Запустити головний виконавчий файл	Форма автентифікації відкрита	Пройдений
Заповнити поля такими значеннями: «Username» : CustomerUsr «Password» : Qwerty123	Немає помилок про невірні дані. Кнопка «Sign In» доступна.	Пройдений
Натиснути кнопку «Sign In»	Користувач автентифікований. Відкрилася головна графічна форма	Пройдений

Таблиця 4.2 – Тестовий сценарій «Реєстрація користувача у системі»

Назва	Тест для перевірки графічної форми автентифікації у системі	
Use case	Реєстрація нового користувача у системі	
Дія	Очікуваний результат	Результат тесту
Передумова		
Запустити головний виконавчий файл	Форма автентифікації відкрита	Пройдений
Натиснути кнопку «Create Account»	Форма реєстрації відкрита. Усі текстові поля видимі.	Пройдений
Заповнити поля наступними значеннями: «Username» : CustomerUsr «Password» : Qwerty123 «Re-type password» : Qwerty123, «First name» : John «Last name» : Smith «Email» : john.sth@e.com	Немає помилок про невірні дані. Кнопка «Sign Up» доступна.	Пройдений
Натиснути кнопку «Sign Up»	Користувач зареєстрований. Відкрилася форма про успішну реєстрацію користувача	Пройдений

Таблиця 4.3 – Тестовий сценарій «Завантаження нового файлу»

Назва	Тест для перевірки графічної форми завантаження файлу	
Use case	Завантаження файлу	
Дія	Очікуваний результат	Результат тесту
Передумова		
Запустити головний виконавчий файл	Форма автентифікації відкрита	Пройдений

Продовження таблиці 4.3

Назва	Тест для перевірки графічної форми завантаження файлу	
Use case	Завантаження файлу	
Дія	Очікуваний результат	Результат тесту
Заповнити поля такими значеннями: «Username» : CustomerUsr «Password» : Qwerty123	Немає помилок про невірні дані. Кнопка «Sign In» доступна.	Пройдений
Натиснути кнопку «Sign In»	Користувач автентифікований. Відкрилася головна графічна форма	Пройдений
Натиснути кнопку завантаження нового файлу	Відкрилася форма завантаження нового файлу. Усі текстові поля доступні	Пройдений
Заповнити поля такими значеннями: «Path» : C:\file.txt	Немає помилок про невірні дані. Кнопка «Upload» доступна.	Пройдений
Натиснути кнопку «Upload»	Створено новий проект. Закрилася графічна форма створення проекту. На головній формі, зліва, оновився список файлів	Пройдений

Таблиця 4.4 – Тестовий сценарій «Отримання файлу»

Назва	Тест для перевірки графічної форми отримання файлу	
Use case	Отримання файлу	
Дія	Очікуваний результат	Результат тесту
Передумова		
Запустити головний виконавчий файл	Форма автентифікації відкрита	Пройдений

Продовження таблиці 4.4

Назва	Тест для перевірки графічної форми отримання файлу	
Use case	Отримання файлу	
Дія	Очікуваний результат	Результат тесту
Заповнити поля такими значеннями: «Username» : CustomerUsr «Password» : Qwerty123	Немає помилок про невірні дані. Кнопка «Sign In» доступна.	Пройдений
Натиснути кнопку «Sign In»	Користувач автентифікований. Відкрилася головна графічна форма	Пройдений
Натиснути кнопку створення запрошення	Відкрилася форма створення запрошення. Усі текстові поля доступні	Пройдений
Обрати файл «file.txt»	Файл обрано. Кнопка «Download» активна	Пройдений
Натиснути кнопку «Download»	Почалася сесія завантаження файлу.	Пройдений

Таблиця 4.5 – Тестовий сценарій «Створення власного ключа шифрування»

Назва	Тест для перевірки графічної форми створення ключа шифрування	
Use case	Створення власного ключа шифрування	
Дія	Очікуваний результат	Результат тесту
Передумова		
Запустити головний виконавчий файл	Форма автентифікації відкрита	Пройдений

Продовження таблиці 4.5

Назва	Тест для перевірки графічної форми створення ключа шифрування	
Use case	Створення власного ключа шифрування	
Дія	Очікуваний результат	Результат тесту
«Username» : CustomerUsr «Password» : Qwerty123		
Натиснути кнопку «Sign In»	Користувач автентифікований. Відкрилася головна графічна форма	Пройдений
Натиснути кнопку створення нового ключа шифрування	Відкрилася форма створення нового ключа шифрування. Усі текстові поля доступні	Пройдений
Заповнити поля такими значеннями: «Name» : MyKey «Path» : C:\keys	Немає помилок про невірні дані. Кнопка «Create» доступна.	Пройдений
Натиснути кнопку «Create»	Створено новий ключ. Закрилася графічна форма створення проекту. На формі оновився список ключів	Пройдений

4.3 Модульне тестування

Модульне тестування – це рівень тестування програмного забезпечення, де тестуються окремі одиниці/компоненти програмного забезпечення. Мета полягає в перевірці того, що кожна одиниця програмного забезпечення виконує, як задумано. Модуль – це найменша тестована частина будь-якого програмного забезпечення. Вона зазвичай має один або декілька входів, і зазвичай один вихід. У процедурному

програмуванні одиницею може бути індивідуальна програма, функція, процедура тощо. У об'єктно-орієнтованому програмуванні найменша одиниця – це метод, який може належати до базового/супер класу, абстрактного класу або похідного/дочірнього класу. (Деякі розглядають модуль програми як одиницю. Це не рекомендується, оскільки в межах цього модуля, ймовірно, буде багато окремих одиниць). Фреймворки тестування, драйвери, stub/mock-об'єкти використовуються в модульному тестування.

Тестування модулів підвищує в безпеці зміні коду. Якщо добре написані тести на модуль, і якщо вони виконуються щоразу, коли будь-який код змінюється, ми зможемо негайно спіймати будь-які дефекти, створені внаслідок змін.

Код є більш багаторазовими. Для того, щоб зробити тестування на модуль, код повинен бути модульними. Це означає, що код легше повторно використовувати.

Розробка відбувається швидше. Вартість виправлення дефекту, виявленого під час тестування, менша порівняно з дефектами, виявленими на більш високих рівнях. Виправлення помилок набагато легше. Коли тест не вдається, потрібно виправити лише останні зміни.

Модульний тест повинен мати такі властивості:

- це має бути автоматизованим та повторюваним;
- це повинно бути простим у здійсненні;
- це має бути актуальним завтра;
- будь-хто повинен мати можливість запускати його одним натисканням кнопки;
- це має працювати швидко;
- він повинен бути послідовним у своїх результатах (він завжди повертає той же результат, якщо ви не змінюйте нічого між пробіжками);
- він повинен мати повний контроль над випробуваним об’єктом;
- це повинно бути повністю ізольованим (працює незалежно від інших тестів).

При написанні тестів зазвичай використовують модель «arrange-act-assert». Ідея полягає в тому, щоб створити mock та налаштовувати їх в arrange частині тесту, здійснити act над випробуваним об’єктом та assert отриманих даних.

Було обрано бібліотеку NUnit для написання юніт тестів. Розглянемо ключові можливості. Якщо метод позначено OneTimeSetUp атрибутом, то він викликається перед початок сесії юніт тестів, SetUp – викликає метод перед кожним юніт тестом, TearDown – після кожного юніт тесту, OneTimeTearDown – після сесії юніт тестів, для позначення класу з юніт-тестами використовують TestFixture, а для самого методу – Test. На рисунку 4.1 зображено результат модульних тестів для фасаду з роботою завантаженням файлів – FileController.

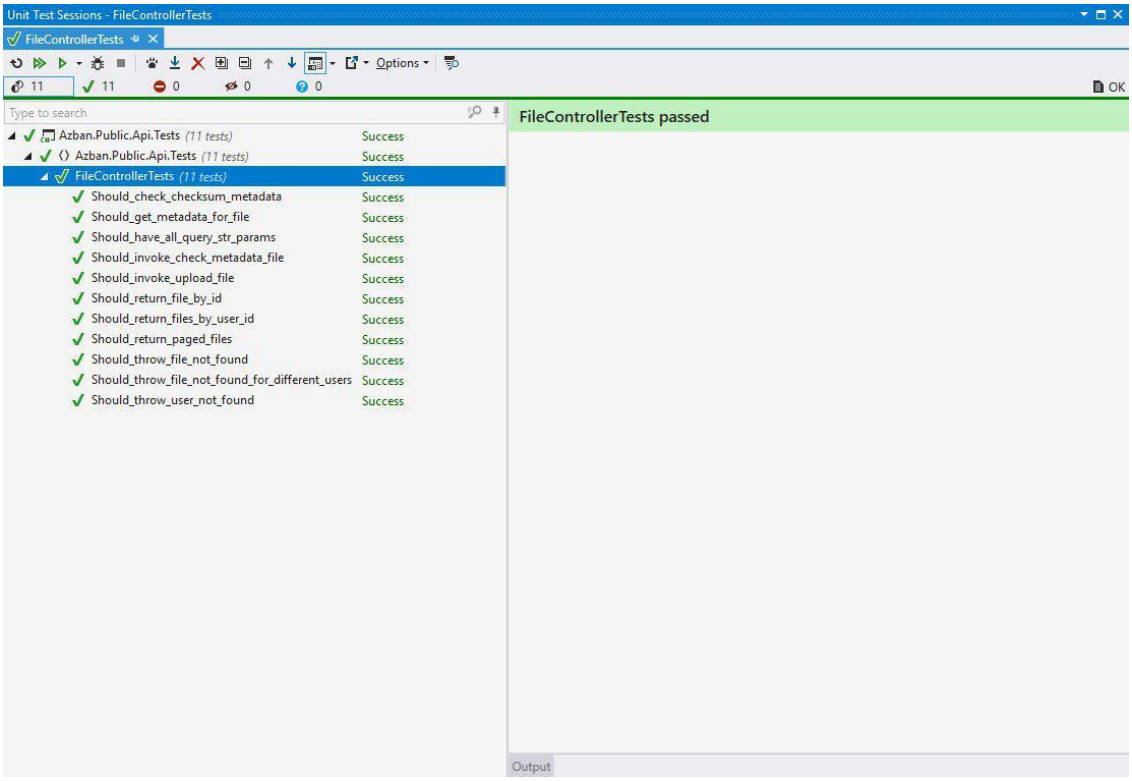


Рисунок 4.1 – Результат виконання модульних тестів

5 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

5.1 Опис ідеї проекту

Метою дисертації є розробка розподіленої системи збереження даних на основі технології блокчейн. У таблиці 5.1 розглянуто зміст ідеї, потенційно можливі сфери й способи використання та основні переваги котрі користувач може отримати користуючись нашої системою.

Таблиця 5.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Розподілене та хмарне збереження файлів, цінних даних з необмеженим простором	1. Для збереження власних даних	Економія власного місця. Потенційна можливість зберігати необмежений обсяг даних
	2. Збереження конфіденційних даних	Не потрібно піклуватися про безпеку даних
	3. Передача даних	Дані можна передавати по будь-якому каналу зв’язку й не піклуватися про їх безпеку
	4. Для архівації даних	Можна зберігати великі архіви даних за достатньо низьку платню

Запропонований спосіб збереження даних відрізняються від звичних хмарних збережень тим, що надає можливість обрати з якою доступністю зберігати дані, усі транзакції фіксуються у блокчейні – що унеможливлює підробку та дані шифруються й зберігаються розподіллено – що надає максимальний рівень їх безпеки.

Після опису ідеї було зроблено аналіз потенційно можливих техніко-економічних переваг алгоритму та системи в цілому. Зроблено порівняльний аналіз з існуючими

аналогами щодо збереження даних за наступними характеристиками: технічна, економічна та надійність.

Проведено порівняльний аналіз показників: для власної ідеї визначаються показники, що мають а) гірші значення (W, слабкі); б) аналогічні (N, нейтральні) значення; в) кращі значення (S, сильні) (таблиця 5.2).

Таблиця 5.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів		W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
	Мій проект	Dropbox			
Технічна	Розподілене та хмарне збереження даних, шифрування даних, можливість вибору терміну зберігання	Хмарне збереження даних, необмежений обсяг, щомісячна оплата, наявність безкоштовних гігабайтів простору		Хмарне збереження	Наявність шифрування власним ключем, вибір терміну зберігання

Продовження таблиці 5.2

Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів		W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
	Мій проект	Dropbox			
Економічна	Необхідні витрати на хмарні обчислення (Microsoft Azure), розробка системи, маркетингова компанія	Не потрібно розробляти систему, необхідні кошти на підтримку	Необхідні витрати на хмарні обчислення		
Надійність	Дані розподілені, мають декілька реплік, можливість налаштувати рівень доступності	Дані централізовані			Наявність розподіленості та реплік

Визначений перелік слабких, сильних та нейтральних характеристик та властивостей ідеї потенційного товару є підґрунтям для формування його конкурентоспроможності.

5.2 Технологічний аудит ідеї проекту

В межах даного підрозділу необхідно провести аудит технології, за допомогою якої можна реалізувати ідею проекту (технології створення товару).

Визначення технологічної здійсненності ідеї проекту передбачає аналіз таких складових (таблиця 5.3):

Таблиця 5.3 – Технологічна здійсненність ідеї проекту

Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
Система збереження даних розподіллено та хмарно на основі технології блокчейн, з можливістю шифрування та вибором рівня доступності	Розподілене збереження даних	Вже наявні аналогічні рішення	Технологія доступна
	Розробка системи з аудитом умов договору збереження на основі технології блокчейн	Середовище розробки IDEA, мова Java	Технологія доступна
		Середовище розробки Microsoft Visual Studio, мова C#	Технологія доступна
	Реалізація CI (Continuous Integration)/CD (Continuous Delivery)	Використання фізичних машин	Технологія доступна
		Використання хмарних потужностей	Технологія доступна
Обрана технологія реалізації ідеї проекту: для розробки системи було обрано мову C# та середовище розробки Visual Studio оскільки вже наявні готовий інструментарій розробки для розподілених систем, для підтримки CI/CD процесів обрано хмарні потужності тому, що вони надають можливість легко розгортати будь-яку кількість потужностей			

За результатами аналізу таблиці робиться висновок щодо можливості технологічної реалізації проекту: так чи ні, а також технологічного шляху, яким це

доцільно зробити (з поміж названих технологій обираються такі, що доступні авторам проекту та є наявними на ринку).

5.3 Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

1) Спочатку проводиться аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку (таблиця 5.4).

Таблиця 5.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	5
2	Загальний обсяг продаж, грн/ум.од	45 000 за рік
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Відсутні
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі (або по ринку), %	50

Ринок є привабливий для входу та потребує новітніх алгоритмів пошуку найкращих, оптимальних методів вирішення задач.

2) Надалі визначаються потенційні групи клієнтів, їх характеристики, та формується орієнтовний перелік вимог до товару для кожної групи (таблиці 5.5).

Таблиця 5.5 – Характеристика потенційних клієнтів стартап-проекту

Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
Хмарне збереження даних із можливістю шифрування та вибору рівня доступності даних	Як для персонального використання так і для корпоративних клієнтів	Важлива безпека даних; дотримання умов доступності	Легкість використання; низька ціна за послуги; відсутність початкових ручних налаштувань; підтримка

3) Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (таблиці 5.6 та 5.7). Фактори в таблиці подаються в порядку зменшення значущості.

Таблиця 5.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Підвищення цін на хмарні обчислення	Провайдер хмарної платформи підвищує ціни на послуги	Розгортання тестових середовищ на власних потужностях
2	Недостатня кількість провайдерів сховищ	Недостатньо провайдерів сховищ у системі задля забезпечення сумарного вільного місця	Підвищення винагороди за збереження даних

Продовження таблиці 5.6

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
3	Зменшення попиту	Кількість користувачів системи зменшується	Можливе зниження цін на збереження даних та проведення маркетингової кампанії

Таблиця 5.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Збільшення кількості інвесторів	Зростає інтерес до системи та капіталовкладень	Використання більш потужних планів хмарного провайдера; збільшення кількості працівників
2	Збільшення кількості провайдерів сховищ	Збільшення кількості провайдерів сховищ пропорційне збільшенню сумарного вільного місця	Зниження цін на збереження даних; надання безкоштовного місця з обмеженим об’ємом

4) Надалі проводиться аналіз пропозиції: визначаються загальні риси конкуренції на ринку (таблиця 5.8).

Таблиця 5.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1.Вказати тип конкуренції – чиста	Досить багато рішень для хмарного збереження даних	Інноваційність рішення, надання послуг вищого класу

Продовження таблиці 5.8

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
2. За рівнем конкурентної боротьби – міжнародна	Багато конкурентів працюють на міжнародній арені	Зменшення вартості послуг
3. За галузевою ознакою - внутрішньогалузева	Боротьба між провайдерами хмарного збереження	Надання більш якісного сервісу; постійні акції
4. Конкуренція за видами товарів: - товарно-видова	Різновиди однієї категорії товару, які здатні задовольнити конкретне бажання покупця	Більше маркетингової кампанії націленої на низьку вартість зберігання
5. За характером конкурентних переваг - неціновa	Ключової відмінністю є шифрування та розподілене зберігання даних з гарантією дотримання умов про рівень доступності	Заохочення більшої кількості провайдерів сховищ
6. За інтенсивністю - не марочна	Не має конкретної прив’язки до марки, наявна можливість використовувати будь-де	Співпраця з потенційно вигідними партнерами

5) Після аналізу конкуренції проводиться більш детальний аналіз умов конкуренції в галузі (таблиця 5.9).

Таблиця 5.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Dropbox, Google Drive, Sia	Можуть мати більш інноваційний спосіб збереження даних	Microsoft	Клієнти потребують рішення для забезпечення повної безпеки даних	Централізовані рішення
Висновки:	Конкуренти не мають інноваційних рішень	Є вірогідність виходу у міжнародну арену	У розробці Permacoin	Можна забезпечити безпечний обмін приватних ключах	Вже присутні на ринку

Вихід на міжнародний ринок можливий завдяки інноваційності ідеї та відсутності її часткової реалізації у конкурентів. Запропонований алгоритм володіє наступними сильними сторонами: гарантоване дотримання договору, шифрування даних та розподілене зберігання.

6) На основі аналізу конкуренції, проведеного в таблиці 5.9, а також із урахуванням характеристик ідеї проекту (таблиця 5.2), вимог споживачів до товару (таблиці 5.5) та факторів маркетингового середовища (таблиці № 5.6-5.7) визначається та обґрунтовується перелік факторів конкурентоспроможності. Аналіз оформлюється за таблицею 5.10.

Таблиця 5.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Можливість вибору рівня доступності	Надає можливість обрати скільки годин у добу дані будуть доступні
2	Шифрування та розподілене зберігання даних	Система розподіляє блоки даних між різними провайдерами сховищ у зашифрованому вигляді, що максимізує безпеку даних
3	Аудит умов контракту	Система проводить регулярний аудит провайдерів сховищ задля забезпечення виконання умов укладеного контракту

7) За визначеними факторами конкурентоспроможності (табл. 5.10) проводиться аналіз сильних та слабких сторін стартап-проекту (табл. 5.11). (С.П. – стартап проект, К.1 – Конкурент 1, К.2 – Конкурент 2).

Таблиця 5.11 – Порівняльний аналіз сильних та слабких сторін

№ п/п	Фактор конкурентоспроможності	Бали (1-20)	Рейтинг товарів-конкурентів у порівнянні з ...(Конкурент 1,2,3)						
			-3	-2	-1	0	+1	+2	+3
1	Можливість вибору рівня доступності	20				К.1, К.2			С.П.
2	Шифрування та розподілене зберігання даних	19				К.1,	К.2		С.П.
3	Аудит умов контракту	20			К.2	К.1			С.П.

8) Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) (табл. 5.12) на основі виділених ринкових загроз та можливостей, та сильних і слабких сторін (табл. 5.10).

Таблиця 5.12 – SWOT-аналіз стартап-проекту

Сильні сторони: можливість вибору рівня доступності даних	Слабкі сторони: недотримання умов контракту щодо рівня доступності
Можливості: збільшення попиту	Загрози: зменшення попиту, конкуренція

На основі SWOT-аналізу розробляються альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок (таблиця. 5.9, аналіз потенційних конкурентів).

Визначені альтернативи аналізуються з точки зору строків та ймовірності отримання ресурсів (таблиця 5.13).

Таблиця 5.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Консультація з приводу роботи алгоритму консенсесу	висока	До трьох місяців
2	Маркетингова кампанія продукту	Низька	До двох місяців

3	Співпраця з більш сильнішими конкурентами	висока	Від одного року

Після аналізу необхідно зазначити обрану альтернативу.

Найбільш вигідною альтернативою є надання консультації з приводу роботи алгоритму консенсусу збереження даних у розподілених системах із використанням технології блокчейн та пояснення основних засад.

5.4 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (табл. 5.14).

Таблиця 5.14 – Вибір цільових груп потенційних споживачів

Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
Корпоративний сектор	Споживачі мають деякі сумління щодо безпеки своїх даних	Помірний попит є в цільовому сегменті	Конкуренція мінімальна, але й попит також	Вхід в сегмент потребує підтримки зі сторони інвесторів

Продовження таблиці 5.14

Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
Які цільові групи обрано: звичайні користувачі, котрі бажають використовувати переваги хмарного збереження даних				

За результатами аналізу потенційних груп споживачів (сегментів) обираються цільові групи, для яких пропонується товар, та визначається стратегія охоплення ринку:

- якщо компанія зосереджується на одному сегменті – вона обирає стратегію концентрованого маркетингу;
- якщо працює із кількома сегментами, розробляючи для них окремо програми ринкового впливу – вона використовує стратегію диференційованого маркетингу;
- якщо компанія працює із всім ринком, пропонуючи стандартизовану програму (включно із характеристиками товару/послуги) – вона використовує масовий маркетинг.

Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку (табл. 5.15).

Таблиця 5.15 – Визначення базової стратегії розвитку

Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Стратегія наслідування лідера	Стратегія диференціації	Надання спеціальних пропозицій; більш професіональна розробка системи	Стратегія обрана для надання товару більш відмітних властивостей, які роблять його унікальним

Наступним кроком є вибір стратегії конкурентної поведінки (табл. 5.16).

Таблиця 5.16 – Визначення базової стратегії конкурентної поведінки

Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
Частково	Активно забирати існуючих та захоплювати нових	Так, можливість розповсюджувати файл за допомогою веб-лінки	Стратегія заняття конкурентної ніші

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та до продукту (див. табл. 5.14), а також в залежності від обраної базової стратегії розвитку (табл. 5.15) та стратегії конкурентної поведінки (табл. 5.16) розробляється стратегія позиціонування (табл. 5.17). що полягає у формуванні ринкової

позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 5.17 – Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
Можливість вибору доступності даних	Стратегія спеціалізації	Запропоновано інший підхід до вибору плану зберігання	Розподілене зберігання даних; використання технології блокчейн; Низька вартість збереження даних
Шифрування та розподілене зберігання даних	Стратегія диференціації	Максимальне безпека даних	

Результатом виконання підрозділу є узгоджена система рішень щодо ринкової поведінки стартап-компанії, яка визначає напрями роботи стартап-компанії на ринку.

5.5 Розроблення маркетингової програми стартап-проекту

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у таблиці 5.18 потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару.

Таблиця 5.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
1	Гнучкість щодо вибору плану оплати за збереження даних	Готове рішення з можливістю вибору кількості годин у добу доступності даних	Інноваційність
2	Впевненість щодо безпеки збереження даних	Власні ключі шифрування та розподілене зберігання даних	Шифрування даних власним ключем

Надалі розробляється трирівнева маркетингова модель товару: уточняється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання (табл. 5.19).

Таблиця 5.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Розподілена система збереження даних на основі технології блокчейн котра надає можливість зберігати файли хмарно та у зашифрованому вигляді з вибором рівня доступності		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Вибір рівня доступності 2. Шифрування власним ключем		

Продовження таблиці 5.19

Рівні товару	Сутність та складові
	Якість: Інтуїтивно-зрозумілий графічний інтерфейс; постійна підтримка користувача
	Серверна частина розгорнута у хмарі; клієнтська частина доступна як веб-, мобільний чи настільний додаток
	Марка: Azban
III. Товар із підкріпленням	Програмне забезпечення
Закритий репозиторій із вихідним кодом; захищена інфраструктура	

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар (остаточне визначення ціни відбувається під час фінансово-економічного аналізу проекту), яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів (табл. 5.20). Аналіз проводиться експертним методом.

Таблиця 5.20 – Визначення меж встановлення ціни

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
10 тис. грн	15 тис. грн	100 тис. грн	5 – 12 тис. грн

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (табл. 5.21):

– проводити збут власними силами або залучати сторонніх посередників (власна або залучена система збуту);
– вибір та обґрунтування оптимальної глибини каналу збуту;
– вибір та обґрунтування виду посередників.

Таблиця 5.21 – Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Вибір плану зберігання, безпосередня оплата у додатку,	Відсутні	Виробник–споживач	Веб- чи мобільний додаток

Останньою складової маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (табл. 5.22).

Таблиця 5.22 – Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
Знають гігантів ринку та конкретні їх послуги	Телефонія, Веб-додатки	Інноваційність рішення, надання гнучкого плана оплати	Висвітлення найсильніших сторін рішення	Низька ціна зберігання даних й гнучкий план оплати

Результатом пункту 5.5 є ринкова (маркетингова) програма, що включає в себе концепції послуг, збуту, просування та попередній аналіз можливостей ціноутворення, спирається на цінності та потреби потенційних клієнтів, конкурентні переваги ідеї, стан та динаміку ринкового середовища, в межах якого буде впроваджено проект, та відповідну обрану альтернативу ринкової поведінки.

ВИСНОВКИ

В магістерській дисертації було спроектовано систему розподілену систему збереження даних на основі технології блокчейн та реалізовано підсистему для завантаження та отримання даних.

Було розглянуто типи архітектур, котрі використовуються при побудові системи збереження даних такі як централізована, розподілена та децентралізована. Було наведено переваги та недоліки кожної з них.

Було спроектовано систему збереження даних із використанням запропонованого алгоритму збереження даних в розподілених системах на основі технології блокчейн. Розглянуто основні функції системи. При побудові архітектури за основу було взято мікросервісну архітектуру. Було виділено обмежені контексти. Комунікацію з сервісами було закрито за фасадом. Завдяки даній архітектурі система має можливість горизонтального масштабування.

Було реалізовано підсистему для завантаження та отримання даних. Серверна частина розроблялась із використанням найновітніших технологій від Microsoft таких як Service Fabric та .NET Core. Перша надає великий інструментарій для створення мікросервісів та підтримує модель акторів. Друга – це новий кросплатформений фреймворк, що у майбутньому надає можливість розгортати систему на Linux так само легко як і для Windows. Публічний фасад було виконано за допомогою ASP .NET Core. Для підтримки асинхронного спілкування між сервісами було використано Azure Event Hubs. Було реалізовано клієнтську частину як настільний додаток для ОС Windows із використанням технології WPF.

В цілому, було розроблену підсистему для завантаження й отримання даних та їх розподіленого збереження. Систем цілком відповідає та є реалізацією запропонованого алгоритму консенсусу збереження даних в розподілених системах.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. File storage systems [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/support/knowledgecenter/en/SSQRB8/com.ibm.spectrum.si.doc/tpch_r_storagesystem_file.html.
2. Nuncic M. The Evolution of Storage: File Storage vs. Block Storage vs. Object Storage – Part 1 [Електронний ресурс] / Michael Nuncic. – 2018. – Режим доступу до ресурсу: https://www.ontrack.com/blog/2018/02/22/the-evolution-of-storage-file-storage-vs-block-storage-vs-object-storage-part-1/.
3. Introduction to Storage Area Networks / J.Tate, P. Beck, H. Ibarra, L. Miklas., 2017. – 300 с. – (1-ше). – (9780738442884).
4. What is an API? (Application Programming Interface) [Електронний ресурс] – Режим доступу до ресурсу: https://www.mulesoft.com/resources/api/what-is-an-api.
5. Gartner: Top 10 cloud storage providers [Електронний ресурс] // Network World. – 2013. – Режим доступу до ресурсу: https://www.networkworld.com/article/2162466/cloud-computing/cloud-computing-gartner-top-10-cloud-storage-providers.html.
6. Gan C. How to build a Network Attached Storage (NAS) / Chin Gan., 2016. – 84 с. – (B01BU2NTO0).
7. Pessach D. Distributed Storage: Concepts, Algorithms, and Implementations / Distributed Storage: Concepts, Algorithms, and Implementations Pessach., 2013. – 106 с. – (1-ше). – (978-1482561043).
8. Shrivastava A. Information Storage and Management: Storing, Managing, and Protecting Digital Information / Alok Shrivastava., 2009. – 106 с. – (1-ше). – (9780470294215).
9. Swan M. Blockchain: Blueprint for a New Economy / Melanie Swan., 2015. – 152 с. – (1).
10. Martinez J. Understanding Proof-of-Work [Електронний ресурс] / Julian Martinez. – 2018. – Режим доступу до ресурсу: https://medium.com/@julianrmartinez43/understanding-proof-of-work-part-1-586d7ee6b014.
11. Morris K. How Much Does A 51% Attack Cost? [Електронний ресурс] / Kai Morris. – 2018. – Режим доступу до ресурсу: https://cryptodisrupt.com/how-much-does-a-51-attack-cost/.

12. Reed J. Smart Contracts: The Essential Guide to Using Blockchain Smart Contracts for Cryptocurrency Exchange / Jeff Reed., 2016. – 54 с. – (1).
13. Что такое Dropbox? [Електронний ресурс] – Режим доступу до ресурсу: https://www.dropbox.com/features.
14. Storj: A Decentralized Cloud Storage Network Framework [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://storj.io/storj.pdf.
15. Vorick D. Sia: Simple Decentralized Storage [Електронний ресурс] / D. Vorick, L. Champine. – 2014. – Режим доступу до ресурсу: https://sia.tech/sia.pdf.
16. Drake N. Best cloud storage of 2018 : Free, paid and business options [Електронний ресурс] / Nate Drake. – 2018. – Режим доступу до ресурсу: https://www.techradar.com/news/the-best-cloud-storage.
17. Buterin V. Ethereum White Paper [Електронний ресурс] / Vitalik Buterin. – 2013. – Режим доступу до ресурсу: http://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf.
18. Antonopoulos A. Mastering Bitcoin: Unlocking Digital Cryptocurrencies / Andreas Antonopoulos., 2014. – 298 с. – (1).
19. Hanmer R. Pattern-Oriented Software Architecture For Dummies / Robert Hanmer. – USA: For Dummies, 2013. – 384 с. – (1).
20. Richardson L. RESTful Web Services / Leonard Richardson., 2007. – 454 с. – (1-ше). – (978-0596529260).
21. Wolfe D. 3-Tier Architecture in ASP.NET with C# tutorial / Donald Wolfe., 2013. – 62 с.
22. Microservice Architecture: Aligning Principles, Practices, and Culture / M.Amundsen, I. Nadareishvili, R. Mitra, M. McLarty., 2016. – 146 с. – (1-ше). – (978-1491956250)
23. Bloomer J. Power Programming with RPC / John Bloomer., 1992. – 522 с.
24. Videla A. RabbitMQ in Action: Distributed Messaging for Everyone / A. Videla, J. Williams., 2012. – 312 с. – (1-ше). – (978-1935182979).
25. Vernon V. Reactive Messaging Patterns with the Actor Model: Applications and Integration in Scala and Akka / Vaughn Vernon., 2015. – 480 с. – (1-ше). – (978-0133846836).

26. All Cryptocurrencies [Електронний ресурс] – Режим доступу до ресурсу: https://coinmarketcap.com/all/views/all/.
27. HTTP Over TLS [Електронний ресурс]. – 2000. – Режим доступу до ресурсу: https://www.rfc-editor.org/info/rfc2818.
28. SCALABILITY: SCALE-UP OR SCALE-OUT, WHAT IT IS AND WHY YOU SHOULD CARE [Електронний ресурс]. – 2013. – Режим доступу до ресурсу: https://www.brianjgraf.com/2013/05/17/scalability-scale-up-scale-out-care/.
29. Amazon S3 [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/s3/.
30. Microsoft Azure [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/en-us/.
31. Google Cloud Platform [Електронний ресурс] – Режим доступу до ресурсу: cloud.google.com.
32. Lock A. ASP.NET Core in Action / Andrew Lock., 2018. – 712 с. – (9781617294617).
33. Smith J. Entity Framework Core in Action / Jon Smith., 2018. – 520 с. – (1-ше). – (978-1617294563).
34. Nathan A. WPF 4.5 Unleashed / Adam Nathan., 2013. – 864 с. – (1-ше). – (978-0672336973).
35. Bai H. Programming Microsoft Azure Service Fabric / Haishi Bai., 2018. – 528 с. – (2-ге). – (978-1-5093-0709-8).
36. Karlsson K. C#—UnitOfWork And Repository Pattern [Електронний ресурс] / Kristoffer Karlsson. – 2017. – Режим доступу до ресурсу: https://medium.com/@utterbbq/c-unitofwork-and-repository-pattern-305cd8ecfa7a.
37. Weil A. Learn WPF MVVM - XAML, C# and the MVVM pattern: Be ready for coding away next week using WPF and MVVM / Arnaud Weil., 2016. – 176 с. – (B01M365NCZ).
38. C# Coding Conventions [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions.

Додаток А. Акт про впровадження

«ЗАТВЕРДЖУЮ»
Декан ФІОТ
Національного технічного
університету України «КПІ ім.
І. Сікорського»
д.т.н., професор
_____ Теленик С. Ф.
«12» грудня 2018р.

АКТ

впровадження в навчальний процес результатів дисертаційної роботи студента кафедри автоматики та управління в технічних системах Озеракіна М. Д. «Розподілена система збереження даних на основі технології блокчейн»

Комісія у складі: голова – зав. кафедри АУТС, професор Ролік О. І., члени комісії – доцент Амонс О. А., асистент Хмелюк В. С. підтвердила, що на кафедрі АУТС НТУУ «КПІ ім. І. Сікорського» впроваджені результати дисертаційної роботи студента кафедри АУТС Озеракіна М. Д. «Розподілена система збереження даних на основі технології блокчейн» в навчальну дисципліну «Технології розробки програмного забезпечення-2. Технології безперервної імплементації та розгортання програмного забезпечення».

Впровадження результатів дисертаційної роботи Озеракіна М. Д. дає можливість ознайомити студентів з новими підходами до побудови процесів безперервної імплементації та розгортання програмного забезпечення, розробки розподілених систем та алгоритмів консенсусу в системах такого типу.

Зав. кафедри АУТС, проф.	Ролік О. І.
Доц. каф. АУТС	Амонс О. А.
Асистент каф. АУТС	Хмелюк В. С.

Додаток Б. Алгоритм консенсусу

CONFERENCE PROCEEDINGS

МАТЕРІАЛИ КОНФЕРЕНЦІЇ

winter infoCom 2018

VII МІЖНАРОДНА НАУКОВО-ПРАКТИЧНА КОНФЕРЕНЦІЯ
З ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Winter InfoCom Advanced Solutions 2018

7th INTERNATIONAL SCIENTIFIC AND PRACTICAL CONFERENCE
ON INFORMATION SYSTEMS AND TECHNOLOGIES

1-2 грудня 2018 року
Україна, Київ

1-2 december 2018
Ukraine, Kyiv

ISBN 978-966-2344-66-0

УДК 004

Winter InfoCom 2018: Матеріали VII Міжнародної науково-практичної конференції з інформаційних систем та технологій, м. Київ, 1-2 грудня 2018 р. – К.: Вид-во ТОВ "Інжиніринг", 2018. – 66с. – Мови укр., рос., англ.

Конференція входить до Переліку міжнародних та всеукраїнських науково-практичних конференцій здобувачів вищої освіти та молодих учених у 2018 році.

Проведення конференції регламентоване наказом ректора КПІ ім. Ігоря Сікорського № 3/598 від 21 листопада 2018 р.

Редакційна колегія дотримується прийнятих міжнародною спільнотою принципів публікаційної етики, відображених, зокрема, в рекомендаціях Комітету з етики наукових публікацій (Committee on Publication Ethics, COPE), а також враховує досвід авторитетних міжнародних видавництв. Щоб уникнути недобросовісної практики в публікаційній діяльності (плагіат, виклад недостовірних відомостей та ін.), з метою забезпечення високої якості наукових публікацій, визнання громадськістю отриманих автором наукових результатів, кожен член редакційної колегії, автор, рецензент, видавець, а також установи, які беруть участь в видавничому процесі, зобов'язані дотримуватися етичних стандартів, норм і правил та вживати всіх можливих заходів для запобігання їх порушень. Дотримання правил етики наукових публікацій усіма учасниками цього процесу сприяє забезпеченню прав авторів на інтелектуальну власність, підвищенню якості видання і виключення можливості неправомірного використання авторських матеріалів в інтересах окремих осіб.

Озеракін М., Амонс О.
Алгоритм консенсусу для систем збереження на основі технології блокчейн..... 45
Дорошенко А., Туліка Є.
Еквівалентні перетворення послідовних програм до моделі акторів... 47
Шатов О., Вовк Є., Амонс О., Хмелюк В.
Масштабована система розпізнавання осіб для спостереження за об'єктами..... 49
***Розподілені та паралельні обчислення / Distributed and Parallel Computing*..... 53**
Дорошенко А., Томишин Ю.В., Яценко О.А.
Проектування програми метеорологічного прогнозування для багатоядерної платформи..... 55
***Abstracts*..... 57**

Алгоритм консенсусу для систем збереження на основі технології блокчейн

Озеракін Микита
КПІ ім. Ігоря Сікорського
Київ, Україна

Амонс Олександр
КПІ ім. Ігоря Сікорського
Київ, Україна

Анотація. **Запропоновано алгоритм для досягнення умов укладеного договору щодо збереження даних між користувачем та провайдером сервісу збереження даних із використанням технології блокчейн та смарт-контрактів.**

Ключові слова: блокчейн, смарт-контракт, збереження даних, Proof-of-Agreement, аудит.

ВСТУП

На сьогоднішній день хмарні системи збереження повністю інтегрувалися з повсякденним життям. З найбільш яскравих представників можна відмітити такі як Dropbox, Google Drive, OneDrive та інші [1]. Вони надають у хмарі потенційно нескінчений простір. Але здебільшого вартість цієї «нескінченності» занадто висока. Також із-за характеру централізованості дані можуть бути скомпрометовані, втрачені чи тимчасово недоступні. Щоб усунути ці недоліки слід задуматися над використанням розподілених чи децентралізованих рішень. У роботі пропонується алгоритм досягнення обраних налаштувань доступності даних та їх захист у разі спроби викрадення.

РОЗРОБКА АЛГОРИТМУ КОНСЕНСУСУ ДЛЯ СИСТЕМ ЗБЕРЕЖЕННЯ ДАНИХ НА ОСНОВІ ТЕХНОЛОГІЇ БЛОКЧЕЙН

Визначимо умови котрих необхідно досягнути при розробці алгоритму консенсусу для збереження даних:

- мінімізація можливості скомпрометувати дані;
- виконання обраного часу доступності даних;
- прозорість процесу оплати за зберігання;
- автономна перевірка доступності даних;
- розподілене збереження даних.

Алгоритм отримав назву Proof-of-Agreement. Визначимо взаємодіючі сторони та їх ролі при використанні алгоритму.

А) Клієнт. Як клієнт я укладаю договір з провайдером сервісу про збереження даних. У договорі визначається скільки годин в добу дані повинні бути доступні, розмір даних, протягом якого часу необхідно зберігати. Перед початком завантаження даних, вноситься депозит на суму котру розрахував провайдер сервісу та встановлюється тимчасове блокування, після того як дані завантажені, у дію вступає заключений договір.

Б) Провайдер сервісу. Як провайдер сервісу я надаю можливість завантаження даних у систему. Виступаю у ролі регулятора в питаннях виконання умов доступності даних. Надаю прозорість щодо проведеного аудиту доступності даних. Обчислюю рейтинг кожного провайдера сервісу, котрий буде надалі використаний у тендері за збереження даних.

В) Провайдер сховища. Як провайдер сховища я надаю вільний простір у своєму сховищі яким може бути наприклад жорсткий диск вбудований у комп'ютер. Зберігаю завантажені дані. Проходжу час від часу аудит щодо доступності даних. Варто зазначити, що провайдер сховища – це одиниця робочої сили для провайдера сервісу. Тобто провайдер сервісу виступає агрегатором провайдерів сховищ.

Перейдемо до опису алгоритму з технічної сторони. Для досягнення прозорості процесу оплати та виконання умов договору застосуємо технологію блокчейн [2] та смарт-контракт [3] – це комп'ютерний алгоритм, призначений для укладання і підтримки комерційних контрактів в технології блокчейн. Найбільш популярна мережа котра використовує ідею смарт-контрактів є Ethereum [4]. Але його використання у даному алгоритмі має певні складноші. Оскільки умови й алгоритм контракту повинні бути повністю детермінованими – це необхідно задля того, щоб майнер міг завантажити увесь блокчейн і прорахувати хеші блоків за алгоритмом дерева Меркла [5], щоб впевнитися, що блокчейн не містить підробок. Саме тому даний алгоритм потребує розробку власної мережі, оскільки процес аудиту побудований на зовнішніх викликах провайдерів сховищ і не може бути відтвореним із 100% вірогідністю.

Смарт-контракт повинен контролювати процес виплат винагородження провайдеру сховища (Storage Reward) та податку провайдеру сервісу (Service Fee). Опишемо послідовність дій при укладанні договору:

1) користувач надсилає запит на збереження даних;
2) провайдер сервісу розраховує необхідний залишок на адресі клієнта (під «адресою» мається на увазі його фізичний гаманець у блокчейні);
3) створюється смарт-контракт;
4) здійснюється блокування коштів клієнта;
5) завантажуються дані;
6) починається процес аудиту доступності даних.

Розглянемо процеси завантаження та отримання даних, аудит доступності даних.

Для мінімізації можливості скомпрометувати дані, перед тим як надіслати дані до провайдеру сервісу вони шифруються особистим ключем клієнта, після цього сервер розбиває дані на блоки з фіксованим

розміром й надсилає їх до провайдерів сховищ, обчислює їх хеш та створює блок у блокчейні що містить перелік усіх хешів блоків та відповідних їм провайдерів сховищ що були завантажені.

Для вибору набору сховищ використовуються такі параметри як доступність та завантаженість провайдера сховищ, його рейтинг та вимоги клієнта щодо доступності даних. В першу чергу беруться у тендер ті сховища які на даний момент часу присутні у мережі, далі порівнюється відносна завантаженість кожного з них котра розраховується як відношення використаного місця до виділеного місця. Після того як отримано такий набор сховищ необхідно співставити їх рейтинг, вимоги клієнта щодо до доступності даних та необхідну кількість реплік блоків. Якщо клієнт бажає високий рівень доступності даних, то більше шансів виграти тендер буде у тих провайдерів котрі мають високий рейтинг й меншу завантаженість. Кількість реплік блоків залежить від того які провайдери будуть обрані у результаті проведеного тендеру, якщо тендер виграють сховища з середнім, то кількість реплік буде більшою задля досягнення вимог клієнта.

Після того як дані були завантаженні провайдер сервісу згідно до вимог клієнта щодо доступності даних розраховується коли необхідно перевірити провайдерів сховищ щодо наявності завантажених блоків даних.

Розраховується відрізок часу протягом якого буде здійснено ряд перевірок. Припустимо, що клієнт забажав доступність даних протягом 20 годин у добу. Отже, оберемо тих провайдерів які мають рейтинг, який свідчить про те, що вони доступні не менше ніж 20 годин та створимо репліки на інших сховищах з доступністю не менше ніж 10 годин. Провайдер сервісу знає хеш блоку, його зсув у файлі та довжину, він обирає певний відсоток блоків від загальної кількості які приймають участь у аудиті. Далі знаходяться відповідні їм master та slave сховища [6]. Другі зберігають репліки блоків даних. Спочатку провайдер сервісу надсилає запит до головного сховища та вимагає обрахувати хеш блоку з певним зсувом та довжиною, якщо провайдер не відповідає протягом певного короткого проміжку часу, здійснюється кілька разів спроба повторити запит, далі запити поширюються на slave сховища.

У разі успішної перевірки створюється транзакція на блокчейні яка перераховує частину грошей з адреси клієнта на адреси провайдерів сховищ та сервісу. У разі невдачі перевірка відкладається на більш довгий час й за тим повторюється знову процес аудиту. Якщо після декількох таких довгих спроб аудит невдалий, то клієнту повертається його залишок котрий розраховуються як депозит з урахуванням усіх виплат по аудиту.

Після кожного аудиту перераховується рейтинг провайдера сховища. При вдалому проходженні перевірки рейтинг збільшується, й навпаки, при невдачі рейтинг зменшується. Рейтинг відіграє невідмінну роль при виборі провайдера сховища. Якщо клієнт бажає дуже високої доступності, то необхідно мати список «довірених» провайдерів сховищ, й навпаки, необхідно слідкувати за «слабкими» провайдерами й менше їм довіряти блоків з більшою кількістю реплік.

Для продовження терміну збереження даних є можливість внеску додаткового депозиту розрахованого відповідно до визначеного терміну. Варто зазначити, що на цьому етапі вже не можна змінити умови доступності даних, оскільки це може призвести до процесу перерозподілення блоків задля задоволення умов.

Клієнт надсилає запит до провайдер сервісу для отримання даних. Сервер знаходить відповідність даних до їх блоків. Після чого формує набор master та slave сховищ які зберігають необхідні блоки. Спочатку сервіс робить запит до головних сховищ якщо ті не відповідають протягом певного часу робиться запит до slave сховищ.

У разі невдалої спроби отримання даних запит клієнта ставить у чергу й тепер клієнт виступає як аудитор й бере участь в одному циклі перевірки даних, якщо ж після проходу одного циклу перевірки дані не вдалося отримати у повному обсязі, такий аудит вважається невдалим й клієнт отримує залишок. Після того як клієнт отримав дані, він їх дешифрує за допомогою особистого ключа й може далі продовжувати користуватися ними.

Висновки

Було запропоновано алгоритм консенсусу для систем збереження даних з використанням технології блокчейн. Задля виконання умов договору обрано смарт-контракти, оскільки вони можуть гарантувати їх виконання. Розглянуто мережу Ethereum, але із-за особливостей її побудови краще розробити власну. Оскільки дані шифруються, розбиваються на блоки й зберігаються на декількох провайдерах сховищ одночасно – це гарантує мінімально можливість їх компрометації. Запропоновано прозорий процес оплати, оскільки кожна виплата по факту аудиту створює нову транзакцію у блокчейні та її може побачити кожен користувач. У разі втрати даних залишок буде гарантовано повернено клієнту завдяки попередньо укладеному смарт-контракту.

Література

1. Drake N. Best cloud storage of 2018 :Free, paid and business options [Електронний ресурс] – 2018. – Режим доступу: https://www.techradar.com/news/the-best-cloud-storage.

2. Swan M. Blockchain: Blueprint for a New Economy / Melanie Swan., 2015. – 152 с. – (1).

3. Reed J. Smart Contracts: The Essential Guide to Using Blockchain Smart Contracts for Cryptocurrency Exchange / Jeff Reed., 2016. – 54 с. – (1).

4. Buterin V. Ethereum White Paper [Електронний ресурс] – 2013. – Режим доступу: http://blockchainlab.com/pdf/Ethereum_white_paper-next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf.

5. Antonopoulos A. Mastering Bitcoin: Unlocking Digital Cryptocurrencies / Andreas Antonopoulos., 2014.

6. Hanmer R. Pattern-Oriented Software Architecture For Dummies r. – USA: For Dummies, 2013.

Додаток В. Способи використання розподілених комп’ютерних ресурсів

www.konferenciaonline.org.ua

Міжнародна наукова інтернет-конференція

"Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення" (випуск 33)

13 листопада 2018 р.

Частина 1

Тернопіль – 2018

Міжнародна наукова інтернет-конференція "Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення (випуск 33)" / Збірник тез доповідей: випуск 33 (м. Тернопіль, 13 листопада 2018 р.). – Частина 1. – Тернопіль. – 2018. – 126 с.

УДК 001 (063)
ББК 72я431

ISSN 2522-932X

Збірник тез доповідей підготовлено за матеріалами Міжнародної наукової інтернет-конференції (випуск 33) від 13 листопада 2018 р.

Збірник матеріалів науково-практичної інтернет-конференції включаються до наукометричної бази даних "РІНЦ/RSCI".

Тексти матеріалів конференції подаються в авторській редакції. Відповідальність за точність, достовірність і зміст поданих матеріалів несуть автори.

Наша адреса: Оргкомітет МНІК "Конференція онлайн"
а/с 1079, м. Тернопіль 46010
тел. моб. 068 366 0 525
e-mail: inetkonf@gmail.com

URL Інтернет-конференції: http://www.konferenciaonline.org.ua/

Ляшук А.М.
Алгоритм напівказуальної фільтрації цифрових зображень……..…….....55

Макарова Д.О.
Підвищення прозорості систем електронних голосувань, які використовують технологію Blockchain……...58

Малітчук А.Д.
Технічні та програмні засоби розробки і використання AR-технологій….....60

Масечко І.О.
Проблема обробки неструктурованих даних……..……..…....61

Маханець Б.О.
Мінімізація ризиків податкових надходжень……..…..…....62

Михайлюк А.М.
Розробка програмного додатку "Reference creator"……...…65

Мішин В.А.
Програмний додаток для пошуку користувачів в соціальній мережі Instagram……66

Музичин Ю.І., Шабатура Ю.В.
Проектування інтелектуальних модулів захисту для системи «Розумний будинок»……68

Назаров О.С., Лягуша О.А.
Система навігації в приміщенні за допомогою GPS та біконів……73

Ніколаєва А.В.
Моделювання поведінки учасників біржових торгів……..…….....75

Озеракін М.Д.
Способи використання розподілених комп'ютерних ресурсів на основі технології блокчейн……79

Олійник Я.М.
Can smartphones replace PCs……....…81

Онукевич А.В.
Система захисту даних персонального кабінету, шифрування…….……....84

4. Для використання маржинальної торгівлі агенту потрібно мати як мінімум $p \geq 0.7$ для того, щоб не збанкрутувати з ймовірністю 0.99, і $p \geq 0.85$, щоб не збанкрутувати з ймовірністю 0.001, що можливо лише для інсайдерів. Для звичайного дрібного трейдера така частка успішних прогнозів протягом 10 років виглядає недосяжною.

Таким чином, запропонована поведінкова модель описує і математично моделює поведінку учасників біржі на основі статистичних даних в залежності від їх здібностей, наявної у них інформації і принципів прийняття ними рішень з купівлі та продажу фінансових інструментів в умовах економічної кризи.

Результати такого моделювання у вигляді теоретичної моделі, а також програмного комплексу, можуть бути використані для наукових досліджень і при вирішенні прикладних задач (для управління інвестиційним портфелем з боку індивідуального трейдера). Дана модель дозволяє аналізувати і пояснювати інформацію про поведінку учасників біржі, формувати і тестувати наукові гіпотези і вивчати досвід успішних біржових гравців.

Література:

1. Вітлінський В.В. Аналіз, моделювання та управління економічним ризиком: навч.метод. посіб. для самост. вивч. дисц. / В.В.Вітлінський, П.І. Верченко.- К.: КНЕУ, 2000. – 292 с.
2. Вітлінський В.В. Моделювання економіки: Навч. посібник / В.В. Вітлінський. – К.: КНЕУ, 2003. – 408 с.
3. Кремер Н.Ш., Путко Б.А. Эконометрика. – М.: ЮНИТИ-ДАНА, 2010.
4. Електронний ресурс http://www.investing.com/

Озеракін М.Д.
НТУУ «Київський політехнічний інститут імені Ігоря Сікорського», м. Київ
Кафедра автоматизації управління в технічних системах, студент

СПОСОБИ ВИКОРИСТАННЯ РОЗПОДІЛЕНИХ КОМП'ЮТЕРНИХ РЕСУРСІВ НА ОСНОВІ ТЕХНОЛОГІЇ БЛОКЧЕЙН

Комп'ютерні ресурси можно розподілити на дві основні категорії: обчислюванні та збережувальні. Обчислюваними ресурсами є центральний та графічний процесори, збережувальні – жорсткий диск, оптичні диски, флеш-пам'ять та інші.

Найяскравішим прикладом використання розподілених й децентралізованих обчислюваних ресурсів в останій час є мережа Bitcoin побудована на технології блокчейн (Blockchain).

Блокчейн – це публічний колективний регістр на якому заснована вся мережа Bitcoin. Всі підтверджені транзакції включаються в ланцюжок блоків. На основі цієї інформації, біткойн-гаманці можуть розрахувати залишок вашого балансу і перевірити, що в нових транзакціях біткойни дійсно витрачаються їх

власником. Цілісність і хронологічний порядок ланцюжка блоків заснований на надійній криптографії.

Транзакція – це передача коштів між біткойн-гаманцями, інформація про яку включається в ланцюжок блоків. Біткойн-гаманці містять конфіденційну інформацію, яка називається секретним ключем, яка використовується, щоб підписувати транзакції, забезпечуючи математичне доказ того, що транзакція дійсно схвалена власником гаманця. Цей підпис так само запобігає зміні транзакції після того, як вона була передана в мережу. Усі транзакції транслюються між користувачами і починають підтверджуватися мережею за допомогою процесу під назвою Майнінг (Mining).

Майнінг – це розподілена обчислювана система, яка використовується для підтвердження транзакцій за допомогою включення їх до блоку. Майнінг забезпечує хронологічний порядок транзакцій у ланцюзі, нейтральність мережі, а також дозволяє різним комп'ютерам "домовитися" про єдиний стан у системі. Для того, щоб транзакції стали підтвердженими, вони повинні потрапити до блоку, який задовольняє суворим криптографічним вимогам і має бути перевірений іншими учасниками мережі. Ці правила не дозволяють змінювати попередній блок, так як в такому випадку всі наступні блоки виявилися б недійсними.

Головною вимогою при побудові блоку є задовільнити поточну складність хешу блоку. Зазвичай складність визначається кількістю передуючих нулів, що містяться у хеші. Сам хеш блоку обчислюється враховуючи хеші транзацій які присутні у ньому та деякого випадково-згенерованого числа яке називається «nonce». Саме це число майнери намагаються вгадати використовуючи обчислювані ресурси.

Таким чином саме за допомогою використання розподілених (оскільки кількість майнерів у мережі досягає до мільйонів) обчислюваних комп'ютерних ресурсів з'являються нові блоки, в середньому на одним блок потрібно виконати до трильйону операцій з пошуку «nonce» числа, а за часом в середньому витрачається 10-15 хвилин.

Розглянемо використання розподілених збережувальних ресурсів. Одним з перших проектів що поєднав розподілені сховища та блокчейн є Storj.io.

Storj.io - розподілена децентралізована платформа з відкрит кодом для хмарного зберігання даних, робота якої заснована на використанні транзакцій Bitcoin (blockchain) транзакцій і власних peer-to-peer протоколів. Платформа також виконує функції платіжної системи, використовуючи для цього власну криптовалюту - StorjcoinX (SJCX), що працює на протоколі Counterparty. Крім цього, за задумом розробників, платформа буде підтримувати розміщення власних повноцінних веб-додатків.

Кожен бажаючий учасник системи може за певну плату, виражену в SJCX, скористатися сховищем і помістити туди свої дані. Доступ до цього компоненту платформи здійснюється через Metadisk - веб-клієнт створений спеціально для взаємодії користувача зі сховищем.

Безпека і анонімність сервісу гарантуються усуненням посередників (децентралізація), а також використанням шифрування для захисту даних під

час їх передачі по каналах зв'язку. При цьому автори проекту виступають категорично проти можливості вилучення або копіювання даних користувачів на вимогу правоохоронних органів або ручного втручання в роботу системи, в разі порушення користувачем правил користувацької угоди. Концепція Storj передбачає повну автономність і невтручання будь-яких людських посередників у вирішення конфліктних ситуацій.

Розробники Storj стверджують, що крім використання публічного реєстра транзакцій Bitcoin, система успадкує його механізм шифрування відкритими і закритими ключами, а також використання криптографічних хеш функцій.

Отже, було розглянуто способи використання розподілених й в одночас децентралізованих ресурсів таких як обчислюваних та збережувальних. У перших яскравим лідером є мережа Bitcoin котра принесла можливість створення високозахищених від підробки транзакцій завдяки використанню децентралізованих обчислюваних ресурсів. У других першопочатківцем стала система збереження даних Storj яка забезпечує анонімність та захист від компромації даних завдяки тому, що кожен файл розбивається на частини й зберігається у різних учасниках мережі.

Література:

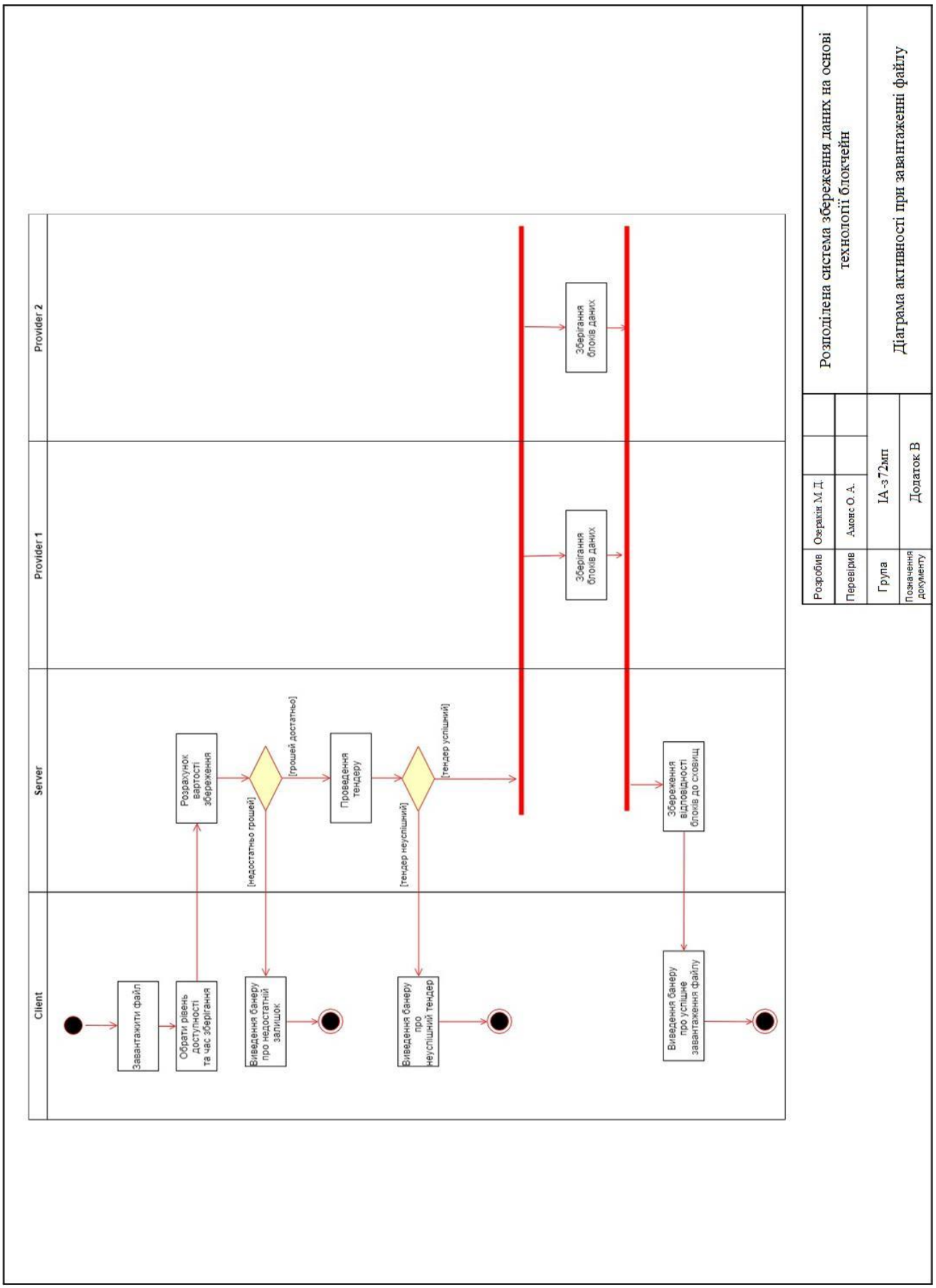
1. Swan M. Blockchain: Blueprint for a New Economy / Melanie Swan., 2015. – 152 с.
2. Tanenbaum A. Distributed Systems / A. Tanenbaum, M. Steen., 2017. – 596 с.
3. Mahmoud M. Decentralized Systems with Design Constraints / Magdi Mahmoud., 2011. – 549 с.

Олійник Я.М., *студент*
Національний технічний університет України ім. Ігоря Сікорського
"Київський політехнічний інститут", м. Київ
Факультет інформатики та обчислювальної техніки
Кафедра автоматизованих систем обробки інформації та управління,
студент 4-го курсу

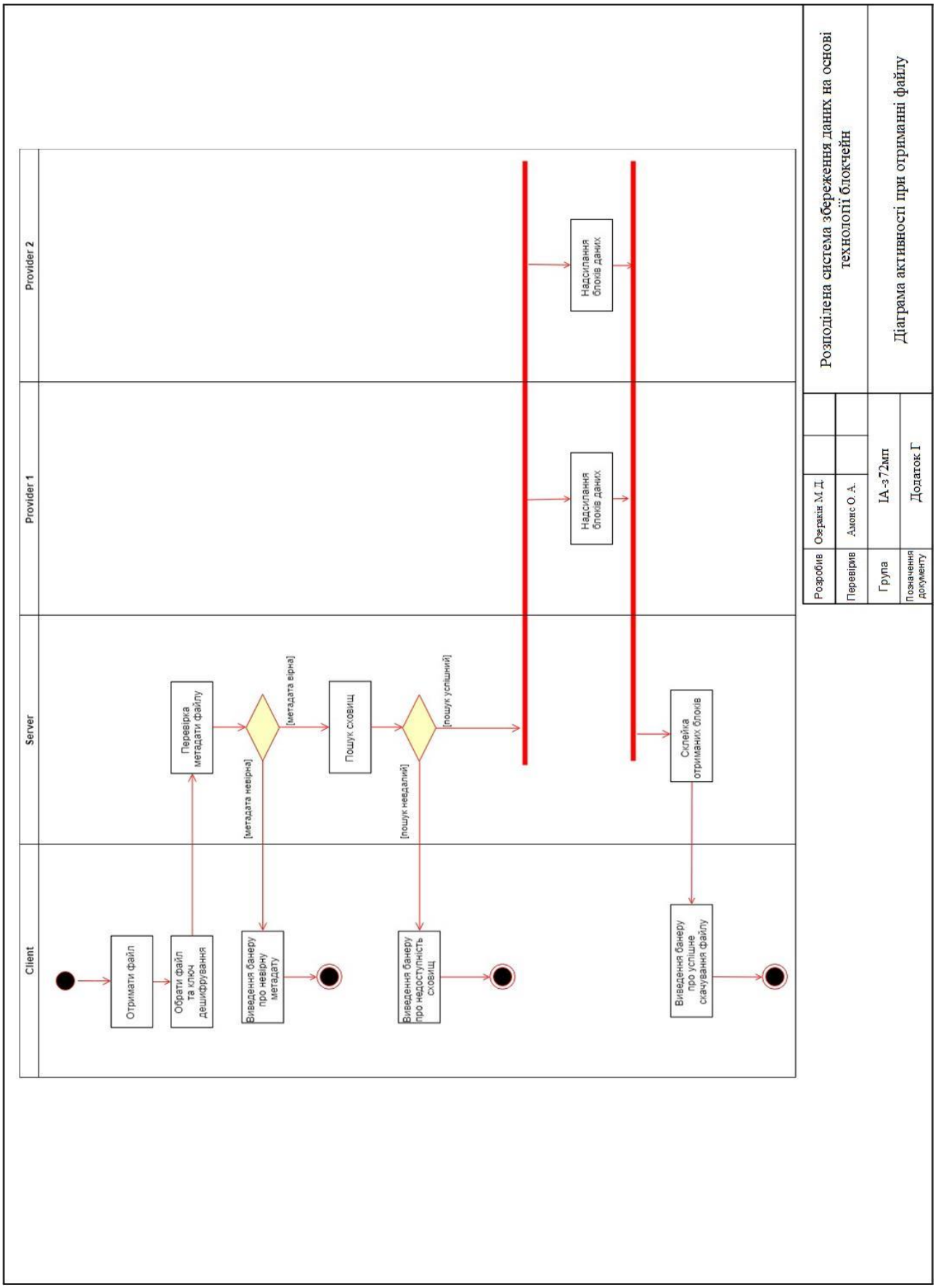
CAN SMARTPHONES REPLACE PCS?

Nowadays technologies are developing with a stupendous speed. It happens because 80% of people on Earth are ready to pay good money to ease their lives. Thus, companies are challenging to gain as many customers as they can. And the most efficient way to do it is to make their own product better than a competitor does. Especially it refers to smartphones. On the graph below you can see linear increasement of smartphone usage (billions of people along the vertical axis and year along the horizontal axis). As you can see, it tends to grow.

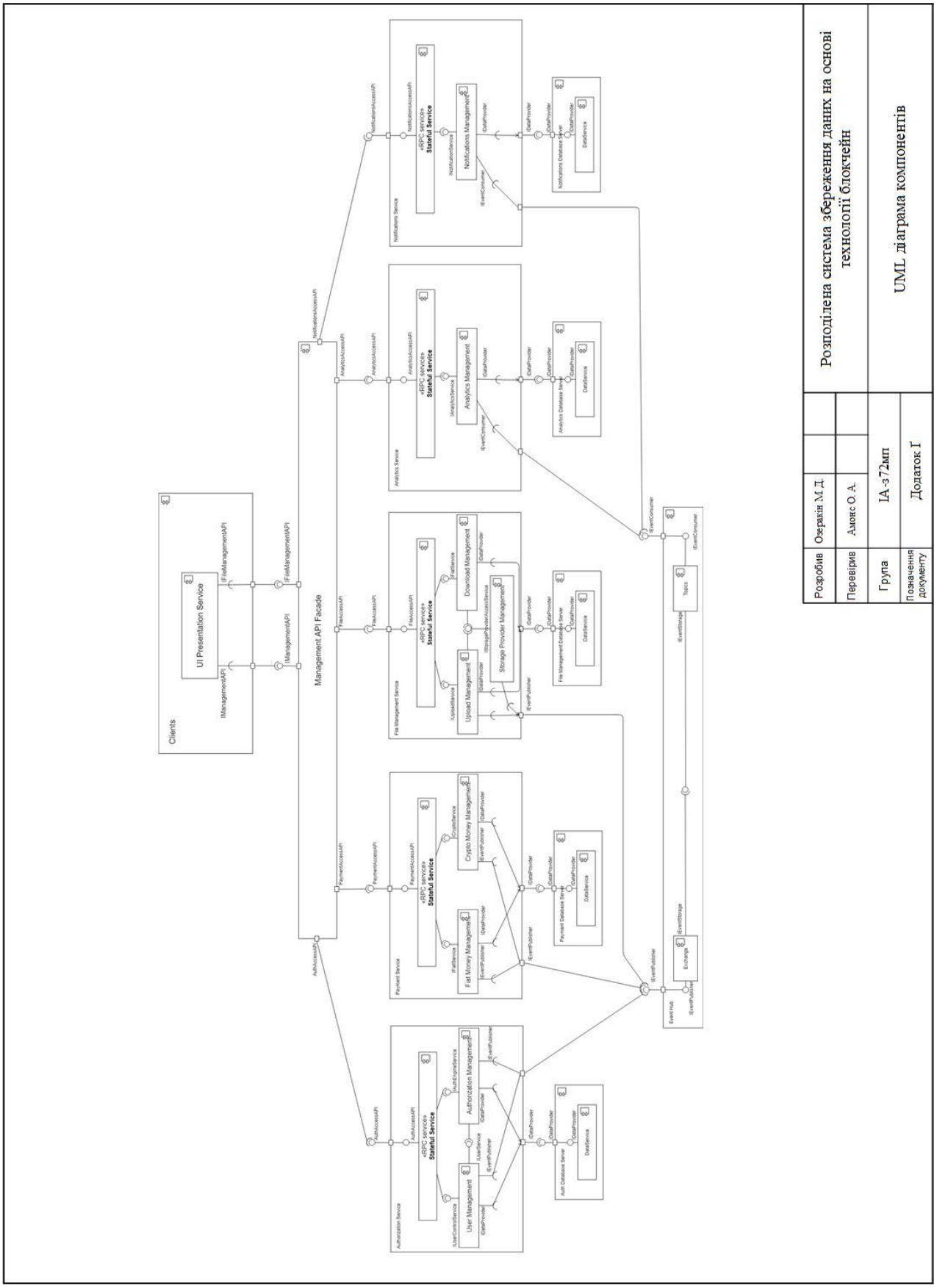
Додаток Г. Діаграма активності для завантаження файлу



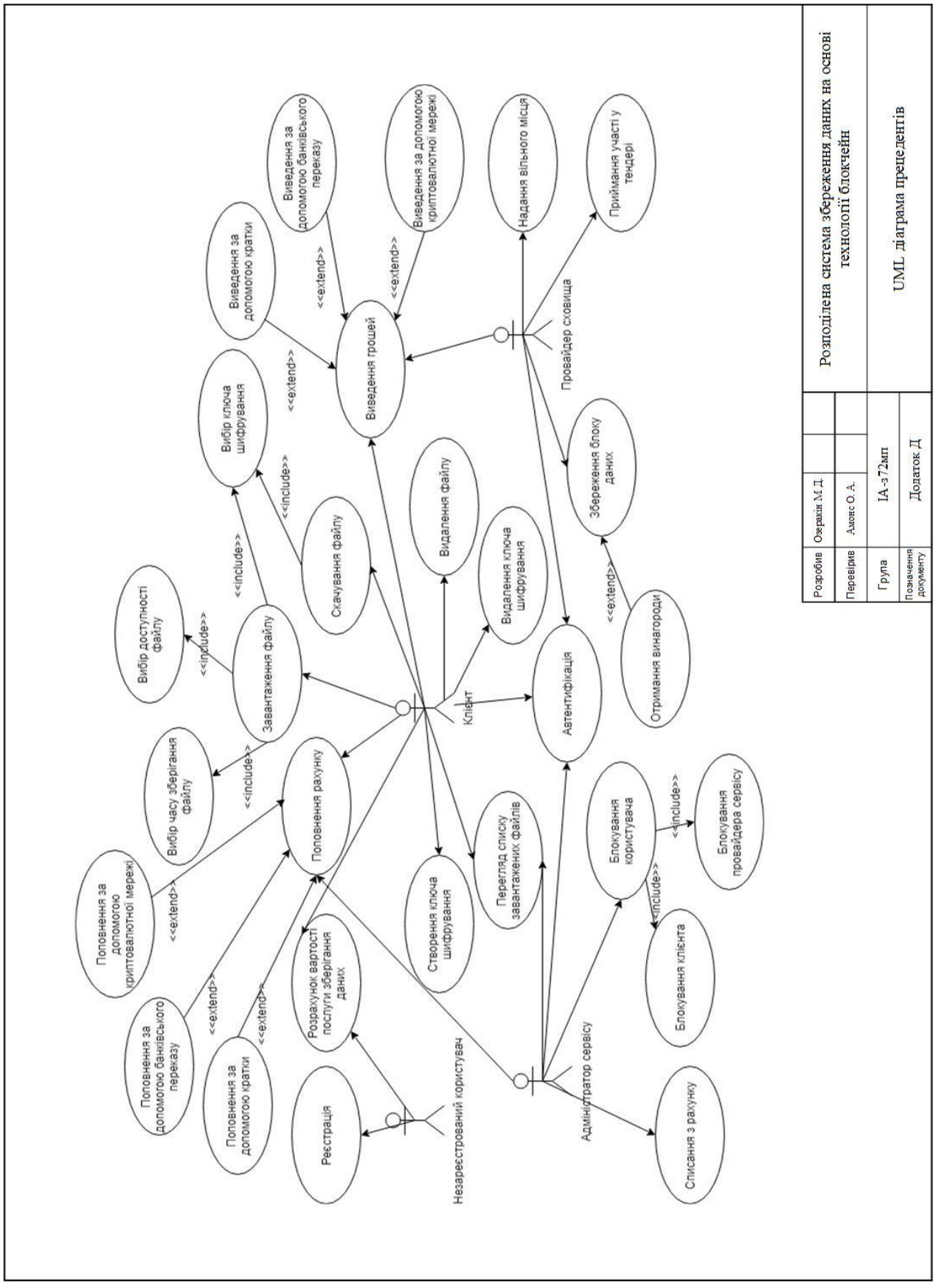
Додаток Ґ. Діаграма активності для отримання файлу



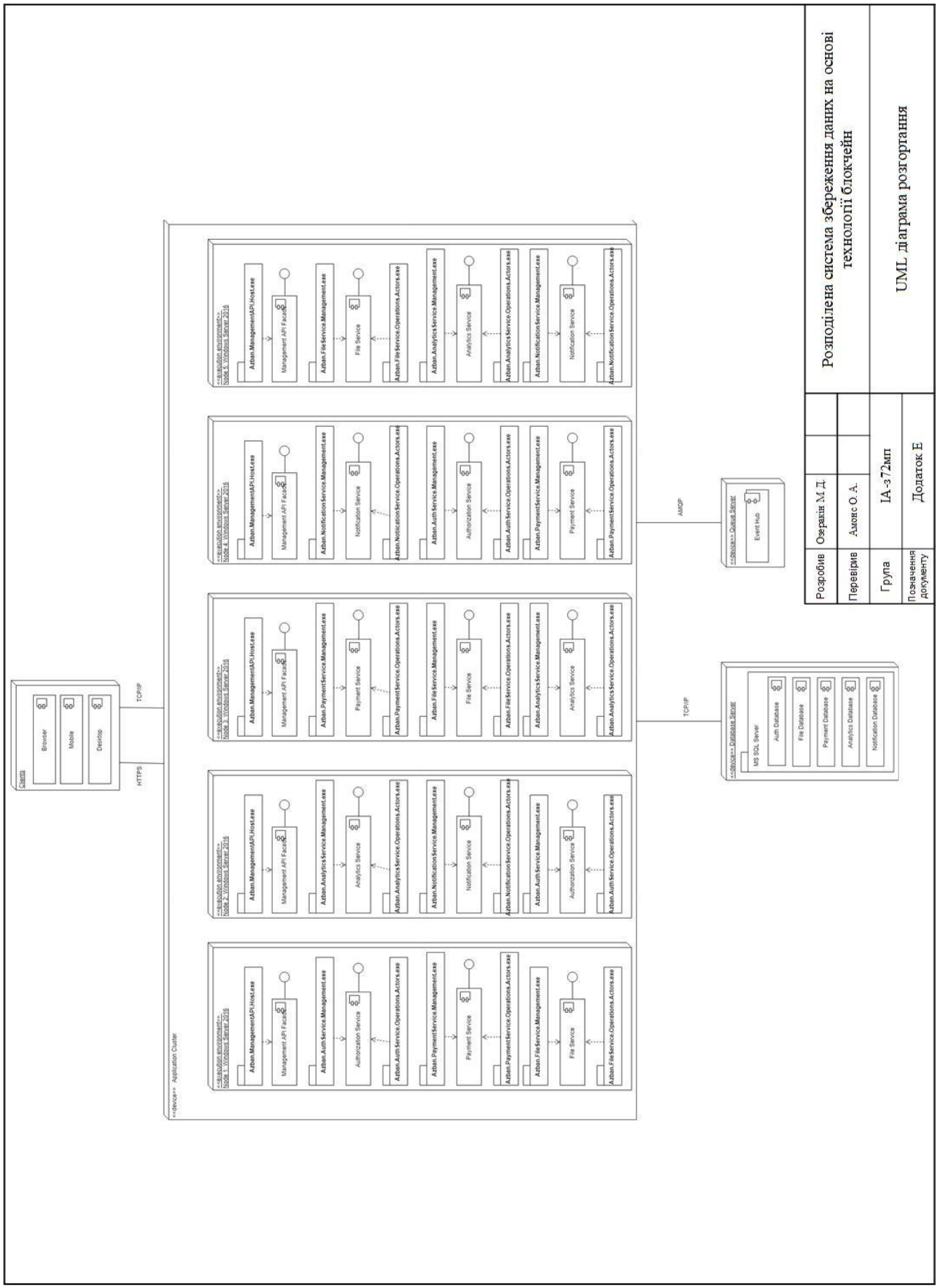
Додаток Д. Діаграма компонентів



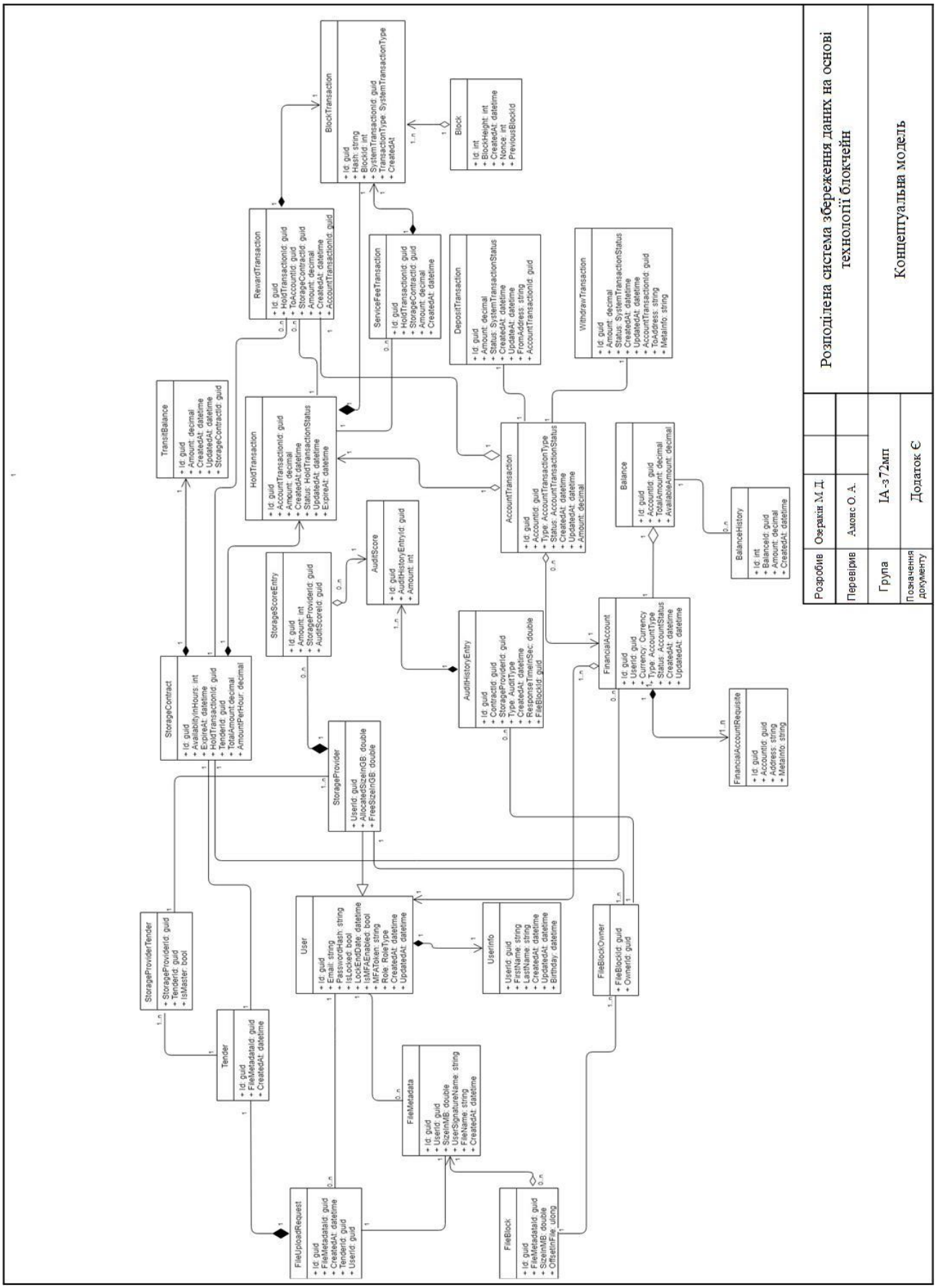
Додаток Е. Діаграма прецедентів



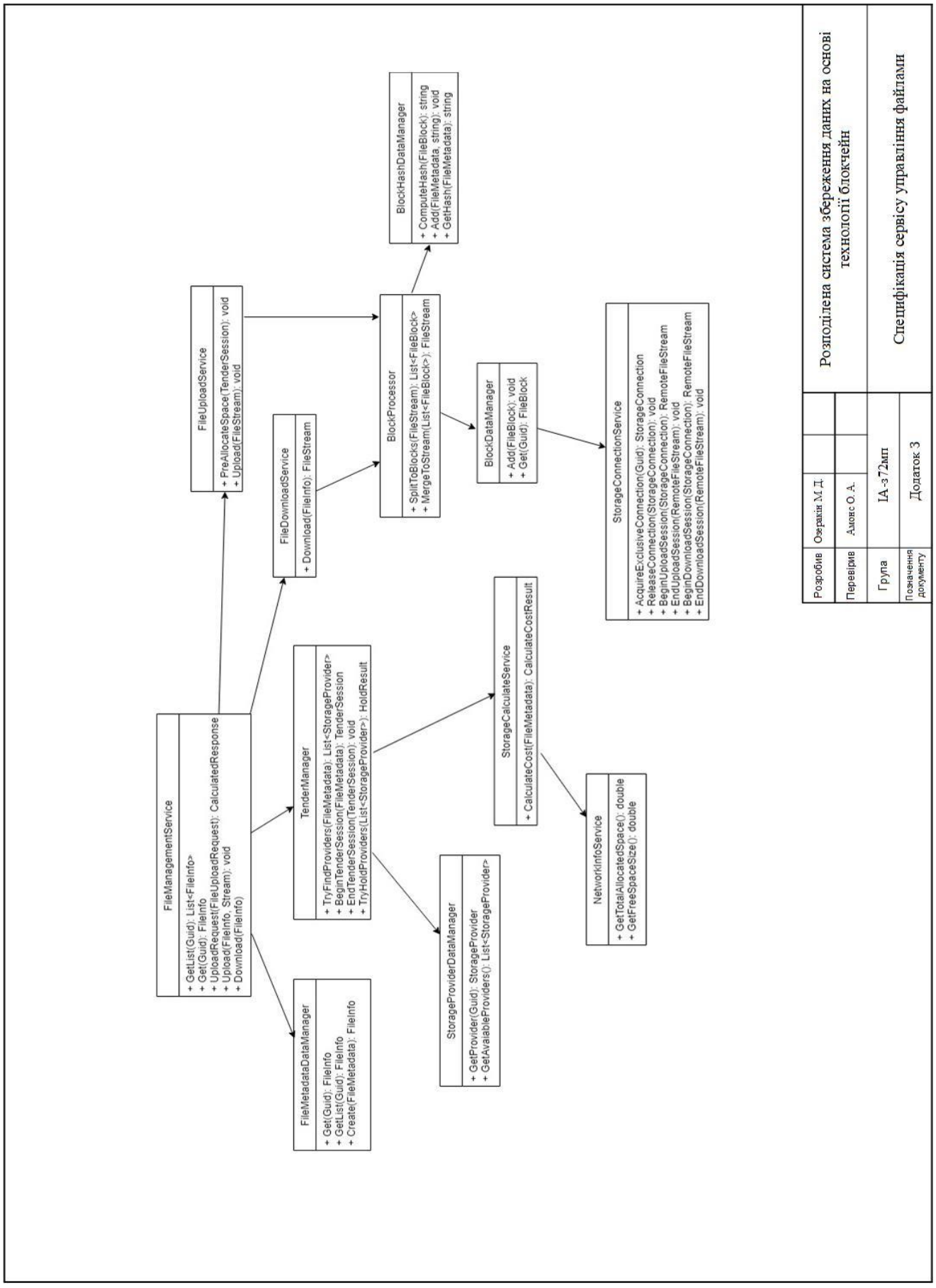
Додаток Є. Діаграма розгортання



Додаток Ж. Концептуальна модель



Додаток 3. Специфікація сервісу управління файлами

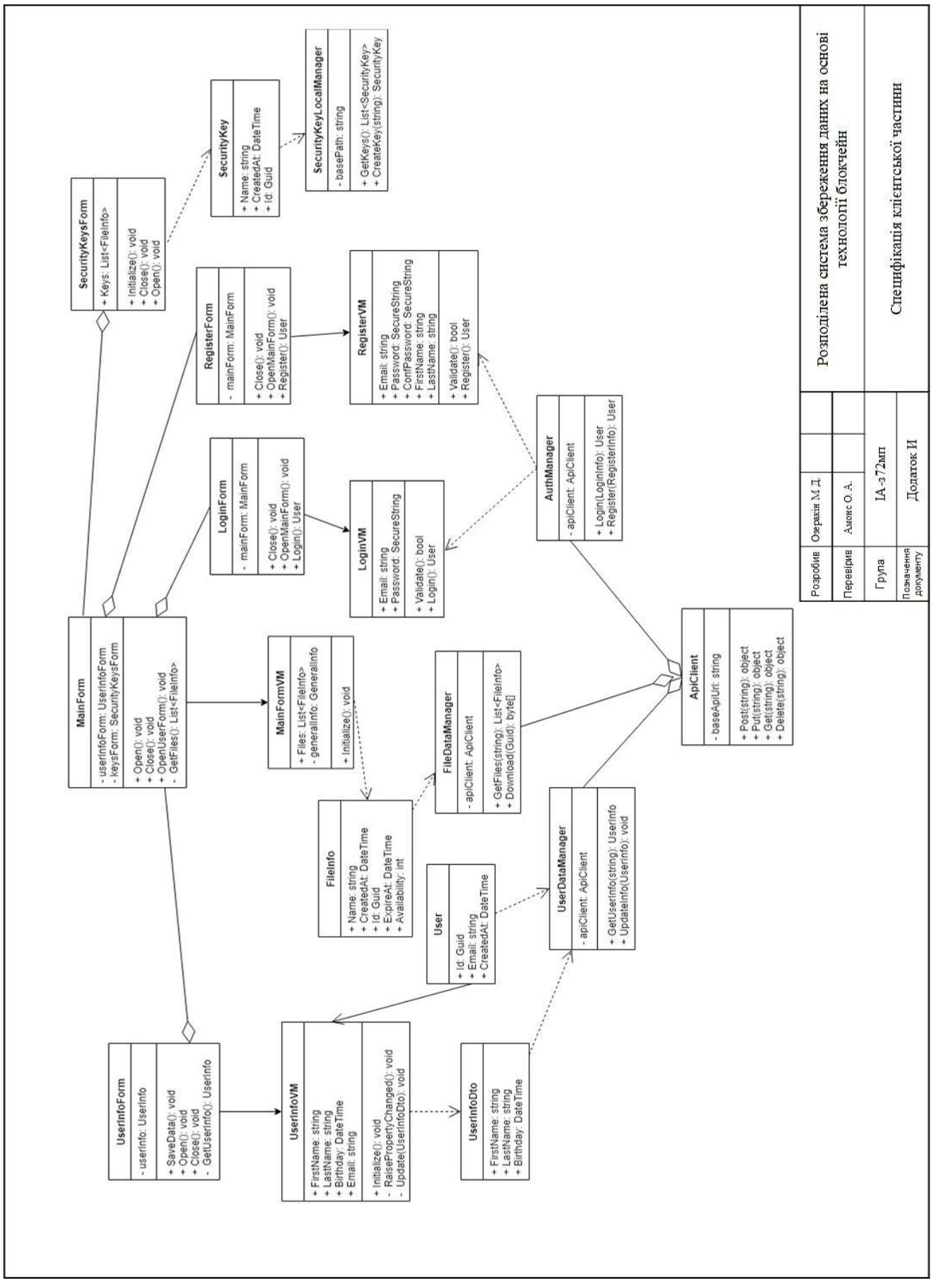


Додаток И. Вихідний код реалізації шару доступу до даних

```
public class UnitOfWork : IUnitOfWork
{
    private readonly DbContext _dbContext;

    public UnitOfWork(string connectionString)
    {
        var options = new
DbContextOptionsBuilder().UseSqlServer(connectionString).Options;

        _dbContext = new DbContext(options);
        Repository = new GenericRepository(_dbContext);
    }

    public void Dispose()
    {
        DisposeInternal(true);
    }

    private void DisposeInternal(bool disposing)
    {
        if (disposing)
        {
            _dbContext?.Dispose();
        }
    }

    public IRepository Repository { get; }

    public Task SaveChangesAsync() => _dbContext.SaveChangesAsync();
}
```

```
public class GenericRepository : IRepository
{
   private readonly DbContext _dbContext;

   public GenericRepository(DbContext dbContext)
   {
      _dbContext = dbContext;
   }

   public IQueryable<T> Query<T>() where T : class => _dbContext.Set<T>();

   public T Add<T>(T entity) where T : class
   {
      _dbContext.Set<T>().Add(entity);

      return entity;
   }

   public T Update<T>(T entity) where T : class
   {
      _dbContext.Set<T>().Attach(entity);
      _dbContext.Entry(entity).State = EntityState.Modified;

      return entity;
   }
}


public class UnitOfWorkFactory : IUnitOfWorkFactory
{
   private readonly SecureString _connectionString;

   public UnitOfWorkFactory(IAppConfig config)
   {
      _connectionString = config. DbConnectionString;
   }

   public IUnitOfWork Create() => new UnitOfWork(_connectionString.ToString());
}
```

Додаток І. Специфікація клієнтської частини



Додаток Ї. Вихідний код реєстрації на клієнтській частині

```
public class SignUpVM : ViewModelBase
{
   #region Private
   private SecureString _password;
   private SecureString _passwordConfirm;
   private string _firstName;
   private string _lastName;
   private string _email;
   private readonly IAccountManager _accountManager;

   #endregion

   public SignUpVM(IAccountManager accountManager)
   {
      _accountManager = accountManager;
      SignUpCommand = new RelayCommand(SignUp);
      BackCommand = new RelayCommand(Back);
   }

   #region Properties

         public SecureString Password
   {
      get { return _password; }
      set { Set(ref _password, value); }
   }

   public SecureString PasswordConfirm
   {
      get { return _passwordConfirm; }
      set { Set(ref _passwordConfirm, value); }
   }

   public string Email
   {
      get { return _email; }
      set { Set(ref _email, value); }
   }

   public string FirstName
   {
      get { return _firstName; }
```

```
            set { Set(ref _firstName, value); }
        }

        public string LastName
        {
            get { return _lastName; }
            set { Set(ref _lastName, value); }
        }

        #endregion

        #region Commands

        public RelayCommand SignUpCommand { get; }

        public RelayCommand BackCommand { get; }

        #endregion

        #region Command Actions

        private void SignUp()
        {
            _accountManager.CreateAccountPassword, PasswordConfirm, Email, FirstName,
LastName);
        }



        #endregion

    }
```