КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра радіотехніки та радіоелектронних систем

До захисту допущено: «На правах рукопису»
Завідувач кафедри _____ Ігор АНІСІМОВ
18 травня 2023 р.

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Хмарна інфраструктура для опрацювання документів»

Виконав:
студент 2-го курсу магістратури
денної форми навчання
спеціальності 172 Телекомунікації та радіотехніка
ОНП «Інформаційна безпека телекомунікаційних систем і мереж»
Лубін Владислав Ігорович _____

Науковий керівник:
к.ф.-м. н., доц. Кельник Олександр Ігорович _____

Рецензент:
д.т.н., проф. Шелестов Андрій Юрійович _____

Засвідчую, що у цій магістерській роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____

Робота допущена до захисту в ЕК рішенням кафедри радіотехніки та радіоелектронних систем від 18 травня 2023 р., протокол № 18.

Завідувач кафедри радіотехніки та радіоелектронних систем,
доктор фіз.-мат. наук, професор
Анісімов Ігор Олексійович _____

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ВМ – віртуальна машина.

VPC - virtual private cloud.

API – application programming interface.

OCR – optical character recognition.

ОС – операційна система.

Демон – програма, працююча у фоновому режимі без взаємодії з користувачем.

NLP – natural language processing.

ZK – zero-knowledge.

ВСТУП

Хмарні обчислення є дуже актуальними у сфері документообігу. Оскільки документообіг включає обмін великою кількістю даних між різними сторонами, забезпечення безпеки і конфіденційності даних є важливим завданням. Хмарні обчислення можуть допомогти забезпечити цільову безпеку і конфіденційність даних у документообігу.

Хмарні рішення дозволяють зберігати дані в хмарі замість локальних пристроїв. Це забезпечує безпеку даних, оскільки хмарні провайдери зазвичай мають більше ресурсів для захисту даних від кібератак.

Вони також сприяють співпраці між різними сторонами в документообігу. Кілька користувачів можуть отримувати доступ до тих самих даних одночасно, що дозволяє вести більш ефективну співпрацю. Хмарні рішення можуть включати в себе автоматизовані процеси обробки даних, що зменшує ризик людських помилок.

Хмарні рішення можуть допомогти зменшити витрати на обладнання та програмне забезпечення, оскільки вони не вимагають фізичного зберігання даних на локальних серверах. Також можуть бути доступні з будь-якого місця, що дозволяє користувачам отримувати доступ до даних зі своїх мобільних пристроїв.

Отже, хмарні обчислення дозволяють забезпечити безпеку, ефективність та зручність в обробці даних.

РОЗДІЛ 1

ANDROID РОЗРОБКА

1.1. Розробка Android застосунків

Розробка Android — це процес створення програмних додатків, які працюють на пристроях Android, таких як смартфони, планшети та переносні пристрої. Android — це операційна система з відкритим кодом, розроблена Google і використовується багатьма виробниками.

Розробку Android можна здійснювати за допомогою різних мов програмування, включаючи Java, Kotlin і C++. Android SDK (Software Development Kit) надає інструменти та бібліотеки, необхідні для розробки програм Android.

Процес розробки програми для Android зазвичай включає такі кроки:

- Розробка інтерфейсу користувача: це передбачає створення візуальних елементів програми, таких як кнопки, меню та екрани.
- Написання коду: це передбачає використання мови програмування та Android SDK для реалізації функцій програми.
- Тестування програми: це передбачає запуск програми на різних пристроях і емуляторах, щоб переконатися, що вона працює правильно.
- Публікація програми: це передбачає надсилання програми до Google Play Store або інших магазинів програм для розповсюдження.

Програми для Android можна розробляти для різних цілей, наприклад для розваг, продуктивності, спілкування та навчання. Їх також можна інтегрувати з іншими службами, такими як платформи соціальних мереж і хмарне сховище.

Спільнота розробників Android є великою та активною, з багатьма ресурсами, доступними для розробників, такими як онлайн-форуми, навчальні посібники та документація. Розробка Android може бути корисною та складною сферою для розробників, які хочуть створювати інноваційні програми для широкого кола користувачів.

1.2. Android SDK

Android SDK (Software Development Kit) — це набір інструментів, бібліотек і API (інтерфейсів прикладного програмування), які використовуються для розробки програм Android. Це важливий компонент у процесі розробки Android, який забезпечує розробників усім необхідним для створення програм Android. Android SDK містить такі компоненти:

- Android Studio: офіційне IDE (інтегроване середовище розробки) для розробки Android. Він надає графічний інтерфейс користувача для розробки інтерфейсів програми та інструментів для тестування та налагодження коду.
- Емулятор Android: віртуальний пристрій Android, який можна використовувати для тестування програм без потреби у фізичному обладнанні.
- Інструменти Android SDK: набір інструментів для керування компонентами Android SDK, створення програм і тестування коду.

- Платформа Android: набір API і системних образів, які забезпечують платформу для розробки програм Android.
- Бібліотеки підтримки: набір бібліотек, які забезпечують зворотну сумісність і додаткові функції для розробки програм.
- Служби Google Play: набір API, які надають доступ до служб Google, таких як Карти, Google Sign-In і Google Drive.
- Приклади: колекція зразків коду та програм, які демонструють, як використовувати різні компоненти Android SDK.

Розробники можуть використовувати Android SDK для створення програм Android за допомогою Java, Kotlin або C++. Вони також можуть використовувати сторонні інструменти та бібліотеки для розширення функціональності SDK.

Android SDK регулярно оновлюється, щоб підтримувати нові версії Android і надавати розробникам нові функції та інструменти. Він доступний для завантаження з веб-сайту розробників Android і може використовуватися в операційних системах Windows, macOS і Linux.

РОЗДІЛ 2

АРХІТЕКТУРА РЕДАКТОРУ ДОКУМЕНТІВ

2.1. Ключові компоненти редагування

Розробка текстового редактора в мобільному додатку передбачає реалізацію кількох функцій і функцій. Деякі з ключових етапів розробки текстового редактора в мобільній програмі включають в себе такі компоненти.

Інтерфейс користувача створюється інтуїтивно зрозумілий, який дозволяє користувачам створювати, редагувати та зберігати текстові файли. Інтерфейс користувача має включати панель інструментів із основними інструментами редагування тексту, такими як тип і розмір шрифту, жирний шрифт, курсив, підкреслення та колір. Інтерфейс користувача також повинен містити параметри для створення нових файлів, відкриття існуючих файлів і збереження файлів.

Введення та редагування тексту мають бути реалізовані у функціях введення та редагування тексту, наприклад введення, виділення та видалення тексту. Текстовий редактор також має підтримувати функції скасування та повторення, дозволяючи користувачам скасовувати та повторювати дії.

Реалізовуються функції форматування тексту, такі як тип шрифту, розмір, стиль, колір і вирівнювання. Користувачі повинні мати можливість форматувати виділений текст або весь документ.

Збереження та завантаження мають бути доступними користовачу, дозволяючи їм зберігати файли на своєму пристрої або в хмарному сховищі

та завантажувати їх назад у редактор. Редактор також має мати можливість працювати з різними форматами файлів, такими як .txt, .doc і .docx.

Спільний доступ включає в себе реалізацію функції спільного доступу, дозволяючи користувачам ділитися своїми файлами з іншими за допомогою електронної пошти, програм для обміну повідомленнями або платформ соціальних мереж.

Налаштування зовнішнього вигляду редактора, наприклад тип і розмір шрифту, тему та колір фону. Інтегрується текстовий редактор з іншими функціями, такими як перевірка орфографії, автовиправлення та автопропозиція.

Для розробки текстового редактора в мобільному додатку розробники можуть використовувати різні інструменти та технології, такі як Java або Kotlin для Android, Swift для iOS, а також кросплатформні фреймворки, такі як React Native або Flutter. Вибір технології залежить від конкретних вимог програми та цільової платформи.

2.2. MVVM архітектура застосунку

У додатку Android, який відповідає шаблону архітектури MVVM (Model-View-ViewModel), дані перетікають між трьома основними компонентами: моделлю, представленням і ViewModel.

Модель: Компонент Model представляє дані та бізнес-логіку програми. Він може включати джерела даних, такі як локальна база даних або віддалений API, і керує операціями пошуку, зберігання та маніпулювання даними. Компонент Model надає дані на рівень ViewModel через інтерфейс.

Перегляд: компонент View представляє інтерфейс користувача програми. Він відображає дані для користувача та фіксує введені користувачем дані. Представлення відповідає за оновлення інтерфейсу користувача на основі змін даних і введення користувача.

ViewModel: Компонент ViewModel діє як посередник між моделлю та компонентами View. Він отримує дані з рівня моделі, обробляє їх за потреби та надає їх на рівень перегляду за допомогою спостережуваних або інших конструкцій реактивного програмування. ViewModel також слухає дані користувача з View і оновлює модель за потреби.

У типовому сценарії потоку даних рівень моделі отримує дані з джерела даних, наприклад віддаленого API, і передає їх на рівень ViewModel. ViewModel маніпулює даними за потреби та надає їх шару View. Рівень View відображає дані для користувача та фіксує введені користувачем дані, які передаються назад у ViewModel. ViewModel оновлює модель будь-якими змінами, зробленими користувачем, і запускає будь-які необхідні операції пошуку даних або маніпуляції.

Потік даних у додатку MVVM Android можна реалізувати за допомогою різних шаблонів і технологій, таких як LiveData, RxJava або Kotlin Coroutines. Конкретна реалізація залежить від вимог програми та переваг розробника. На рис. 2.1 зображена структура MVVM архітектури, яка застосовується в мобільній розробці.

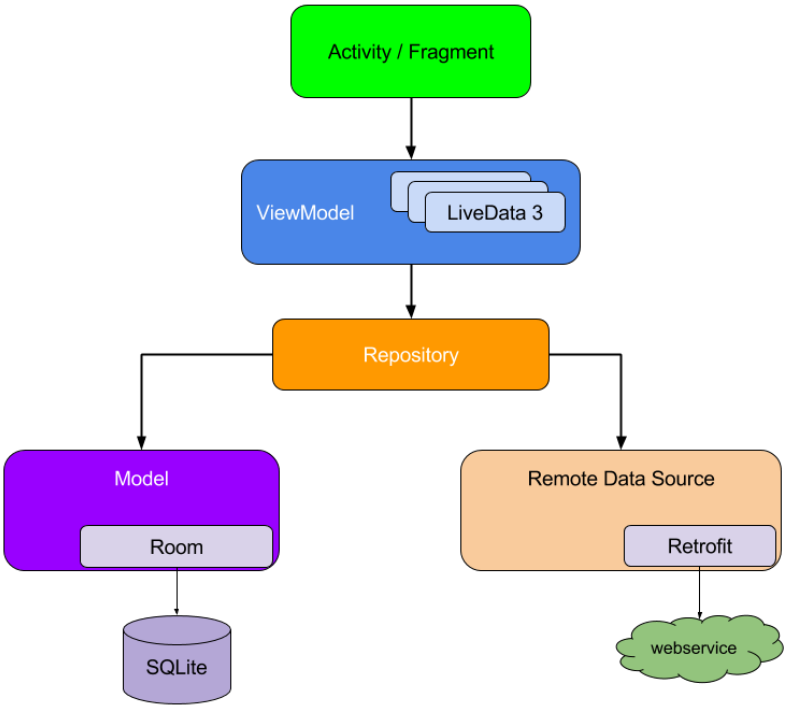

Рис. 2.1. Структура MVVM

На рис. 2.1 зображено: Activity/Fragment – це елементи інтерфейсу з якими взаємодіє користувач. ViewModel – класс, який керує потоком даних, що потрапляють на інтерфейс та обробляє результат взаємодії. Repository – це класс, що керує та зберігає данні, які отримуються з серверу, а також віддає їх у разі запиту, на інтерфейс. Model – це об’єкт, що описує одиницю данних, яку додаток отримує з серверу. Romote Data Source – класс, що використовує бібліотеку Retrofit для надсилання запитів на сервер та обробки відповідей від нього.

РОЗДІЛ 3

GOOGLE CLOUD PLATFORM

3.1. Ознайомлення з GCP

Google Cloud Platform (GCP) — це платформа хмарних обчислень, запропонована Google, яка дозволяє користувачам створювати та запускати програми та служби в інфраструктурі Google. Він надає широкий спектр послуг і інструментів для розробки, розгортання та керування хмарними програмами та рішеннями. Деякі з ключових функцій і послуг, які пропонує Google Cloud Platform:

Обчислювальні послуги: Google Cloud Platform пропонує різноманітні обчислювальні послуги, зокрема віртуальні машини, контейнери та параметри безсерверного обчислення, такі як Cloud Functions і App Engine.

Служби зберігання: Google Cloud Platform надає кілька варіантів зберігання для різних типів даних, включаючи сховище об’єктів (Google Cloud Storage), сховище файлів (Google Cloud Filestore) і блокове сховище (Google Compute Engine Persistent Disks).

Послуги баз даних: Google Cloud Platform пропонує низку служб баз даних, зокрема бази даних NoSQL (Cloud Firestore, Cloud Bigtable), реляційні бази даних (Cloud SQL) і служби керованих баз даних (Cloud Spanner).

Мережеві послуги: Google Cloud Platform надає низку мережевих служб, зокрема віртуальну приватну хмару (VPC), балансування навантаження в хмарі, хмарну мережу CDN і хмарне з’єднання.

Сервіси машинного навчання: Google Cloud Platform пропонує різноманітні сервіси машинного навчання, включаючи попередньо підготовлені моделі для розпізнавання зображень і мовлення (Cloud Vision

API, Cloud Speech-to-Text API) та інструменти для створення власних моделей (TensorFlow, Cloud AutoML).

Сервіси великих даних: Google Cloud Platform надає інструменти та служби для керування великими даними, включаючи обробку потокових даних у реальному часі (Cloud Dataflow), аналітику великих даних (BigQuery) і сховище даних (Cloud Bigtable).

Безпека та відповідність: Google Cloud Platform надає різноманітні функції безпеки та відповідності, зокрема шифрування в стані спокою та під час передачі, керування ідентифікацією та доступом, а також відповідність різноманітним галузевим стандартам.

Google Cloud Platform пропонує гнучкі варіанти ціноутворення, включно з оплатою за використання та моделі ціноутворення на основі підписки. Він також надає різні інструменти розробки та API, такі як Cloud SDK і Google Cloud API, щоб розробники створювали та керували своїми програмами та службами.

3.2 Compute engine

Google Compute Engine (GCE) — це служба хмарної інфраструктури, яку пропонує Google Cloud Platform і дозволяє користувачам створювати віртуальні машини (VM) у хмарі та керувати ними. Він забезпечує гнучкий і масштабований спосіб запуску програм і служб в інфраструктурі Google.

Екземпляри ВМ: GCE дозволяє користувачам створювати екземпляри ВМ у хмарі та керувати ними. Користувачі можуть вибирати з безлічі попередньо налаштованих екземплярів віртуальної машини або створювати власні екземпляри з бажаною операційною системою, розміром диска та іншими параметрами конфігурації. GCE дозволяє користувачам легко

збільшувати або зменшувати свої екземпляри ВМ відповідно до мінливого попиту. Користувачі можуть додавати або видаляти екземпляри, збільшувати або зменшувати кількість процесорів і пам’яті, а також змінювати інші параметри конфігурації за потреби.

Мережа: GCE надає низку параметрів мережі, включаючи віртуальні приватні хмари (VPC), балансування навантаження та брандмауери, щоб забезпечити безпечне та надійне підключення до мережі для екземплярів віртуальних машин.

Зберігання: GCE надає кілька варіантів зберігання, включаючи постійні диски, локальні SSD і Google Cloud Storage, для зберігання даних, пов’язаних з екземплярами віртуальних машин.

Інтеграція з іншими службами GCP: GCE інтегрується з іншими службами Google Cloud Platform, такими як Google Kubernetes Engine (GKE) і Cloud SQL, щоб забезпечити комплексне хмарне інфраструктурне рішення. GCE надає гнучкі варіанти виставлення рахунків, у тому числі знижки за розрахунки за використання та безперервне використання, щоб допомогти користувачам оптимізувати свої витрати.

GCE підтримує різноманітні операційні системи, включаючи Linux і Windows, і пропонує ряд інструментів і API для керування екземплярами віртуальних машин, як-от Google Cloud Console, Cloud SDK і REST API.

3.3 Document AI

API Document AI Google Cloud – це API на основі машинного навчання, який допомагає отримувати структуровані дані зі сканованих документів. Він використовує вдосконалену технологію OCR (оптичне розпізнавання символів) і алгоритми машинного навчання для аналізу

неструктурованих даних і перетворення їх у структуровані дані, які можна аналізувати, шукати та обробляти. Ключовими особливостями цієї технології є:

- Обробка документів: API Document AI може аналізувати та отримувати інформацію з широкого діапазону типів документів, включаючи відсканові документи, PDF-файли та зображення.
- Advanced OCR: API використовує вдосконалену технологію OCR для розпізнавання тексту у відсканованих документах, навіть якщо текст не має стандартного формату чи компонування.
- Вилучення даних: API може витягувати структуровані дані з документів, включаючи таблиці, форми та інші типи даних.
- Вилучення сутностей: API може ідентифікувати та витягувати ключові сутності та зв’язки з документів, наприклад імена, адреси та номери телефонів.
- Налаштування: API можна налаштувати відповідно до конкретних потреб бізнесу, навчивши його за допомогою спеціальних моделей.
- Інтеграція: API Document AI можна легко інтегрувати з іншими сервісами Google Cloud Platform, такими як BigQuery, Cloud Storage та Cloud Functions.

Серед поширених випадків використання Document AI API — обробка рахунків-фактур, вилучення інформації з контрактів і угод, а також аналіз фінансових звітів.

3.4. Document AI processors

Процесори Document AI – це набір інструментів і API, наданий Google Cloud Platform, який можна використовувати для автоматичного вилучення структурованих даних із неструктурованих документів. Ціль цих процесорів полягає в тому, щоб дозволити підприємствам більш ефективно обробляти великі обсяги документів, таких як рахунки-фактури, контракти та квитанції, шляхом автоматизації ручного введення даних.

Процесори штучного інтелекту документів використовують комбінацію алгоритмів машинного навчання та методів обробки природної мови (NLP) для розуміння вмісту документів і вилучення відповідних даних. Процесори можна налаштувати для обробки певних типів документів і полів даних, і їх можна інтегрувати з іншими службами GCP, такими як Google Cloud Storage і BigQuery.

AI для розуміння документів, цей процесор можна використовувати для вилучення структурованих даних із рахунків-фактур, квитанцій та інших фінансових документів.

Синтаксичний аналізатор форм, цей процесор можна використовувати для отримання даних зі структурованих форм, таких як опитування та анкети.

Парсер закупівельних документів, цей процесор можна використовувати для отримання даних із закупівельних документів, таких як замовлення на купівлю та рахунки-фактури.

Аналіз контрактів, цей процесор можна використовувати для автоматичного вилучення ключових пунктів і умов із контрактів.

Процесори штучного інтелекту для документів можуть допомогти компаніям заощадити час і зменшити кількість помилок, пов’язаних із

ручним введенням даних. Автоматизуючи процес вилучення структурованих даних із неструктурованих документів, підприємства можуть підвищити ефективність і точність обробки великих обсягів документів.

РОЗДІЛ 4

ДЕЦЕНТРАЛІЗОВАНЕ СХОВИЩЕ

4.1. Ceph

Хмарні файлові сховища продовжують набирати популярності, і вимоги до них продовжують зростати. Сучасні системи вже не в змозі повністю задовольнити всі ці вимоги без значних витрат ресурсів на підтримку та масштабування цих систем. Під системою я маю на увазі кластер з тим чи іншим рівнем доступу до даних. Для користувача важлива надійність зберігання та висока доступність, щоб файли можна було завжди легко та швидко отримати, а ризик втрати даних прагнув нуля. У свою чергу для постачальників та адміністраторів таких сховищ важлива простота підтримки, масштабованість та низька вартість апаратних та програмних компонентів.

Ceph — це програмно обумовлена розподілена файлова система з відкритим вихідним кодом, позбавлена вузьких місць і єдиних точок відмови, яка представляє з себе кластер вузлів, що легко масштабується до петабайтних розмірів, виконують різні функції, забезпечуючи зберігання і реплікацію даних, а також розподіл навантаження, що гарантує високу доступність та надійність. Система безкоштовна, хоча розробники можуть надати платну підтримку. Жодного спеціального обладнання не потрібно.

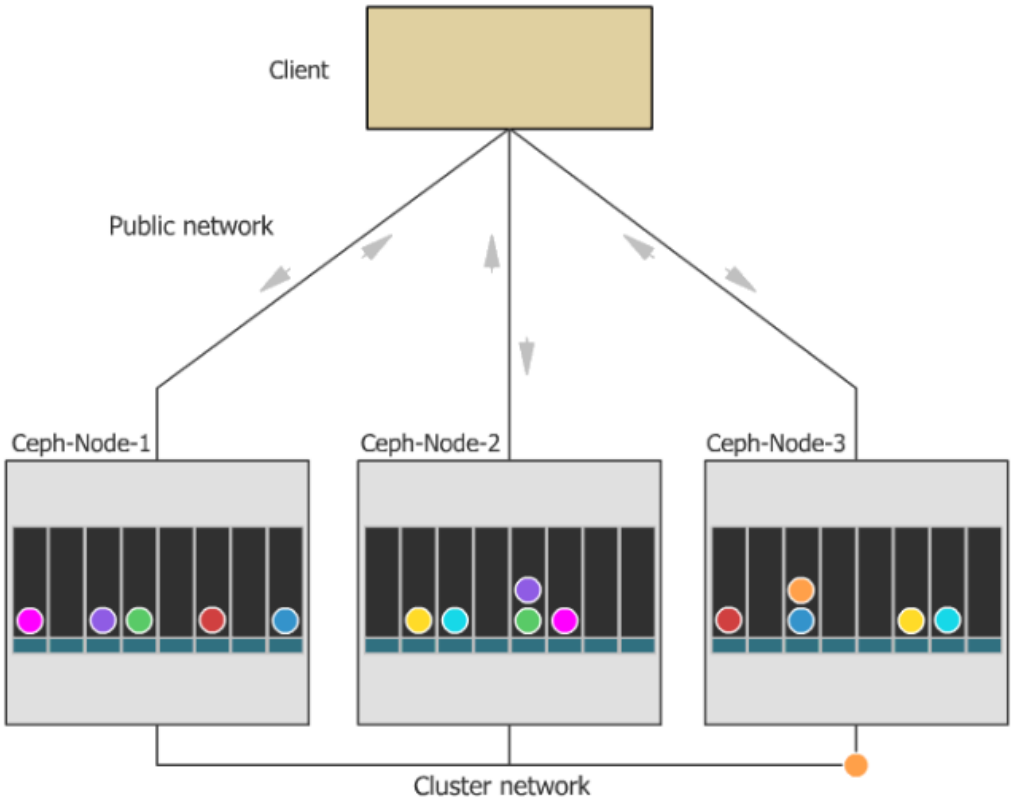

Рис. 4.1. Візуалізація функціонування файлової системи

При виході будь-якого диска, вузла або групи вузлів з ладу Ceph не тільки забезпечить збереження даних, але і сам відновить втрачені копії на інших вузлах доти, поки вузли, що вийшли з ладу, або диски не замінять на робітники. При цьому ребілд відбувається без секунди простою та прозоро для клієнтів.

4.1.1 Ролі вузлів та демони. Оскільки система програмно визначається і працює поверх стандартних файлових систем та мережевих рівнів, можна взяти пачку різних серверів, набити їх різними дисками різного розміру, з'єднати все це щастя якоюсь мережею (краще швидкою) і підняти кластер. Можна вставити у ці сервери по другій мережній карті, і з'єднати їх другою мережею для прискорення міжсерверного обміну даними. А експерименти з налаштуваннями та схемами можна легко проводити навіть у віртуальному середовищі. Мій досвід експериментів показує, що найдовше у цьому процесі – це встановлення ОС. Якщо у нас є три сервери з дисками та

налаштованою мережею, то підняття працездатного кластера з дефолтними налаштуваннями займе 5-10 хвилин (якщо все робити правильно).

Поверх операційної системи працюють демони Ceph, які виконують різні ролі кластера. Таким чином, один сервер може виступати, наприклад, і в ролі монітора (MON), і в ролі сховища даних (OSD). А інший сервер тим часом може виступати в ролі сховища даних та в ролі сервера метаданих (MDS). У великих кластерах демони запускаються на окремих машинах, але в малих кластерах, де кількість серверів дуже обмежена, деякі сервери можуть виконувати відразу дві чи три ролі. Залежить від потужності сервера та самих ролей. Зрозуміло, все буде працювати швидше на окремих серверах, але не завжди це можливо реалізувати. Кластер можна зібрати навіть з однієї машини та всього одного диска, і він працюватиме. Інша розмова, що це не матиме сенсу. Слід зазначити і те, що завдяки програмній визначеності, сховище можна підняти навіть поверх RAID або iSCSI-пристрою, проте в більшості випадків це не матиме сенсу.

У документації перераховано 3 види демонів:

- Mon - демон монітора
- OSD - демон сховища
- MDS - сервер метаданих (необхідний лише у разі використання CephFS)

Початковий кластер можна створити з кількох машин, поєднуючи ними ролі кластера. Потім, зі зростанням кластера та додаванням нових серверів, якісь ролі можна дублювати на інших машинах або повністю виносити на окремі сервери.

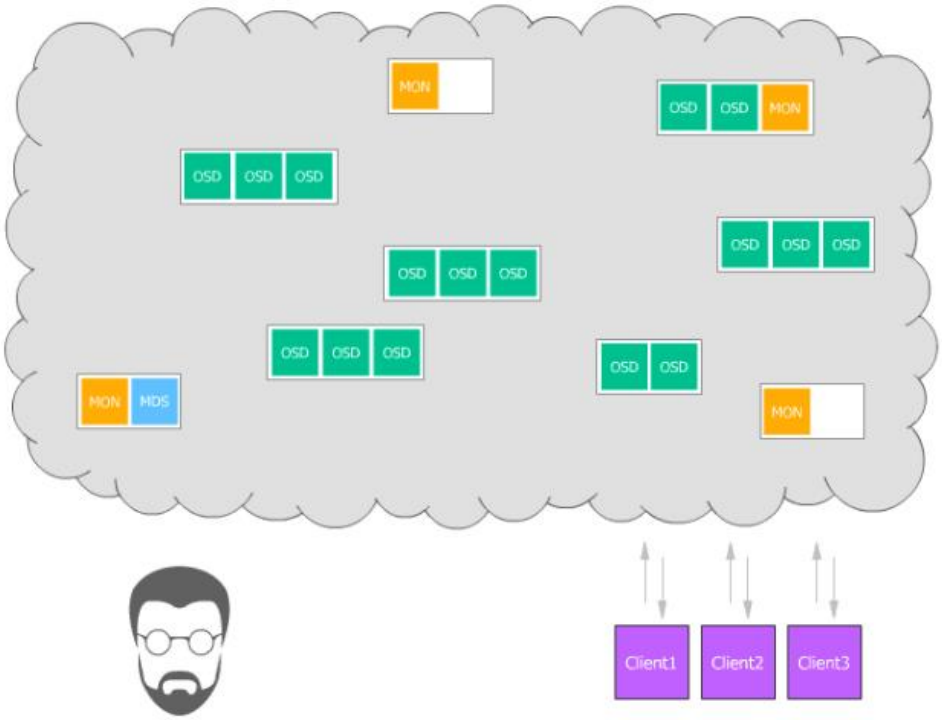

Рис. 4.2. Взаємодія вузлів та демонів

4.1.2 Структура зберігання. Кластер може мати один або багато пулів даних різного призначення та з різними налаштуваннями. Пули поділяються на плейсмент-групи. У плейсмент-групах зберігаються об'єкти, до яких звертаються клієнти. На цьому логічний рівень закінчується, і починається фізичний, тому що за кожною плейсмент-групою закріплений один головний диск та кілька дисків-реплік (скільки залежить від фактора реплікації пула). Іншими словами, логічно об'єкт зберігається в конкретній плейсмент-групі, а на фізичному — на дисках, які за нею закріплені. При цьому диски фізично можуть бути на різних вузлах або навіть у різних датацентрах.

Чинник реплікації — це надмірності даних. Кількість копій даних, що зберігатиметься на різних дисках. Цей параметр відповідає змінна size. Чинник реплікації може бути різним для кожного пулу, і його можна міняти на льоту. Взагалі, у Ceph практично всі параметри можна міняти на льоту, миттєво отримуючи реакцію кластера. Спочатку у нас може бути 2, і в цьому

випадку пул зберігатиме по дві копії одного шматка даних на різних дисках. Цей параметр пула можна поміняти на 3, і в цей же момент кластер почне перерозподіляти дані, розкладаючи ще одну копію даних по дисках, не зупиняючи роботу клієнтів.

Пул це логічний абстрактний контейнер для організації зберігання даних користувача. Будь-які дані зберігаються у пулі як об'єктів. Декілька пулів можуть бути розмазані по одним і тим же дискам (а може і по різних, як налаштувати) за допомогою різних наборів плейсмент-груп. Кожен пул має ряд параметрів, що настроюються: фактор реплікації, кількість плейсмент-груп, мінімальна кількість живих реплік об'єкта, необхідна для роботи і т. д. Кожному пулу можна налаштувати свою політику реплікації (по містах, датацентрах, стійках або навіть дисках). Наприклад, пул під хостинг може мати фактор реплікації 3, а зоною відмови будуть датацентри. І тоді Ceph гарантуватиме, що кожен шматочок даних має по одній копії у трьох датацентрах. Тим часом пул для віртуальних машин може мати фактор реплікації 2, а рівнем відмови вже буде серверна стійка. І в цьому випадку, кластер зберігатиме лише дві копії. При цьому, якщо у нас дві стійки зі сховищем віртуальних образів в одному датацентрі, і дві стійки в іншому, система не буде звертати увагу на датацентри, і обидві копії даних можуть відлетіти в один датацентр, проте гарантовано різні стійки, як ми і хотіли .

4.1.3 Плейсмент-група. Плейсмент-групи - це така сполучна ланка між фізичним рівнем зберігання (диски) та логічною організацією даних (пули).

Кожен об'єкт на логічному рівні зберігається у конкретній плейсмент-групі. На фізичному ж рівні він лежить у потрібній кількості копій на різних фізичних дисках, які в цю плейсмент-групу включені (насправді не диски, а OSD, але зазвичай один OSD це і є один диск, і для простоти я називатиму це диском, хоча нагадаю, за ним може бути і RAID-масив або iSCSI-пристрій). При факторі реплікації size=3, кожна плейсмент група включає три диски. Але при цьому кожен диск знаходиться в безлічі плейсмент-груп,

і для якихось груп він буде первинним, для інших — реплікою. Якщо OSD входить, наприклад, до складу трьох плейсмент-груп, то при падінні такого OSD плейсмент-групи виключать його з роботи, і на його місце кожна плейсмент-група вибере робочий OSD і розмаже по ньому дані. За допомогою даного механізму досягається досить рівномірний розподіл даних і навантаження. Це дуже просте і водночас гнучке рішення.

4.1.4 Монітори. Монітор - це демон, що виконує роль координатора, з якого починається кластер. Як тільки у нас з'являється хоч один робочий монітор, у нас з'являється Ceph-кластер. Монітор зберігає інформацію про здоров'я та стан кластера, обмінюючись різними картами з іншими моніторами. Клієнти звертаються до моніторів, щоб дізнатися, які OSD писати/читати дані. При розгортанні нового сховища насамперед створюється монітор (або кілька). Кластер може прожити на одному моніторі, але рекомендується робити 3 або 5 моніторів, щоб уникнути падіння всієї системи через падіння єдиного монітора. Головне, щоб кількість їх була непарною, щоб уникнути ситуацій роздвоєння свідомості (split-brain). Монітори працюють у кворумі, тому якщо впаде більше половини моніторів, кластер заблокується для запобігання неузгодженості даних.

OSD - це юніт сховища, який зберігає самі дані та обробляє запити клієнтів, обмінюючись даними з іншими OSD. Зазвичай це диск. І зазвичай за кожен OSD відповідає окремий OSD-демон, який може запускатись на будь-якій машині, на якій встановлений цей диск. Це друге, що потрібно додавати в кластер при розгортанні. Один монітор і один OSD - мінімальний набір для того, щоб підняти кластер і почати користуватися ним. Якщо на сервері крутиться 12 дисків під сховище, на ньому буде запущено стільки ж OSD-демонів. Клієнти працюють безпосередньо з самими OSD, минаючи вузькі місця, і досягаючи тим самим розподілу навантаження. Клієнт завжди записує об'єкт на первинний OSD для якогось плейсменту групи, а вже далі

даний OSD синхронізує дані з іншими (вторинними) OSD із цієї ж плейсмент-групи. Підтвердження успішного запису може надсилатися клієнту відразу після запису на первинний OSD, а може після досягнення мінімальної кількості записів (параметр пула min_size). Наприклад, якщо фактор реплікації 3, а мінімальний 2, то підтвердження про успішний запис відправиться клієнту, коли об'єкт запишеться хоча б на два OSD з трьох (включаючи первинний).

При різних варіантах налаштування цих параметрів ми спостерігатимемо і різну поведінку.

Якщо 3 і 2: все буде добре, поки 2 із 3 OSD плейсмент-групи живі. Коли залишиться лише один живий OSD, кластер заморозить операції цієї плейсмент-групи, поки не оживе хоча б ще один OSD.

Якщо size=min_size, то плейсмент-група блокуватиметься при падінні будь-якого OSD, що входить до її складу. А через високий рівень розмазаності даних більшість падінь хоча б одного OSD буде закінчуватися заморозкою всього або майже всього кластера. Тому параметр size завжди повинен бути хоча б на один пункт більшим за параметр min_size.

Якщо size=1, кластер працюватиме, але смерть будь-якої OSD означатиме безповоротну втрату даних. Ceph дозволяє виставити цей параметр в одиницю, але навіть якщо адміністратор робить це з певною метою на короткий час, ризик бере на себе.

Диск OSD і двох частин: журнал і самі дані. Відповідно, дані спочатку пишуться в журнал, потім уже розділ даних. З одного боку, це дає додаткову надійність і деяку оптимізацію, а з іншого боку — додаткову операцію, яка позначається на продуктивності. Питання продуктивності журналів розглянемо нижче.

В основі механізму децентралізації та розподілу лежить так званий CRUSH-алгоритм (Controlled Replicated Under Scalable Hashing), що відіграє

важливу роль в архітектурі системи. Цей алгоритм дозволяє однозначно визначити місце розташування об'єкта на основі хешу імені об'єкта та певної карти, яка формується виходячи з фізичної та логічної структур кластера (датацентри, зали, ряди, стійки, вузли, диски). Карта не включає інформацію про місцезнаходження даних. Шлях до даних кожен клієнт визначає сам, за допомогою CRUSH-алгоритму та актуальної карти, яку він попередньо запитує у монітора. Під час додавання диска або падіння сервера картка оновлюється.

Завдяки детермінованості, два різних клієнти знайдуть один і той же однозначний шлях до одного об'єкта самостійно, позбавляючи систему необхідності тримати всі ці шляхи на якихось серверах, синхронізуючи їх між собою, даючи величезне надлишкове навантаження на сховище в цілому.

Клієнт хоче записати об'єкт object1 в пул Pool1. Для цього він дивиться в карту плейсмент-груп, яку раніше люб'язно надав монітор, і бачить, що Pool1 розділений на 10 плейсмент-груп. Далі за допомогою CRUSH-алгоритму, який на вхід приймає ім'я об'єкта та загальну кількість плейсмент-груп у пулі Pool1, обчислюється ID плейсмент-групи. Наслідуючи карту, клієнт розуміє, що за цією плейсмент-групою закріплено три OSD (припустимо, їх номери: 17, 9 і 22), перший з яких є первинним, а значить клієнт робитиме запис саме на нього. До речі, їх три, тому що в цьому пулі встановлено фактор реплікації size=3. Після успішного запису об'єкта на OSD_17 робота клієнта закінчена (це якщо параметр пула min_size=1), а OSD_17 реплікує цей об'єкт на OSD_9 і OSD_22, закріплені за цією плейсмент-групою. Важливо розуміти, що це спрощене пояснення алгоритму.

За замовчуванням, наша CRUSH-карта плоска, всі ноди знаходяться в одному просторі. Однак, можна цю площину легко перетворити на дерево, розподіливши сервери по стійках, стійки по рядах, ряди залів, зали по

датацентрам, а датацентри по різних містах і планетах, вказавши який рівень вважати зоною відмови. Оперуючи такою новою картою, Ceph грамотніше розподілятиме дані, враховуючи індивідуальні особливості організації, запобігаючи сумним наслідкам пожежі в датацентрі або падінню метеориту на ціле місто. Більше того, завдяки цьому гнучкому механізму можна створювати додаткові шари, як на верхніх рівнях (датацентри та міста), так і на нижніх (наприклад, додатковий поділ на групи дисків у межах одного сервера).

4.1.5 Кешування. Ceph передбачає кілька способів збільшення продуктивності кластеру методами кешування.

У кожного OSD є кілька ваг, і одна з них відповідає за те, який OSD у плейсмент-групі буде первинним. А як ми з'ясували раніше, клієнт пише дані саме на первинний OSD. Так от можна додати в кластер пачку SSD дисків, зробивши їх завжди первинними, знизивши вагу primary-affinity HDD дисків до нуля. І тоді запис здійснюватиметься завжди спочатку на швидкий диск, а потім уже не поспішаючи реплікуватися на повільні. Цей метод найнеправильніший, проте найпростіший у реалізації. Головний недолік у тому, що одна копія даних завжди лежатиме на SSD і потрібно дуже багато таких дисків, щоб повністю покрити реплікацію. Хоча цей спосіб хтось і застосовував на практиці, але його я скоріше згадав, щоб розповісти про можливість управління пріоритетом запису.

Взагалі левова частка продуктивності залежить від журналів OSD. Здійснюючи запис, демон спочатку пише дані до журналу, а потім до самого сховища. Це вірно завжди, крім випадків використання BTRFS як файлову систему на OSD, яка може робити це паралельно завдяки техніці copy-on-write, але я так і не зрозумів, наскільки вона готова до промислового застосування. На кожен OSD йде власний журнал, і за умовчанням він знаходиться на тому ж диску, що й дані. Однак журнали з чотирьох або п'яти дисків можна винести на один SSD, непогано прискоривши операції запису.

Метод не дуже гнучкий та зручний, але досить простий. Недолік методу в тому, що при вильоті SSD з журналом ми втратимо відразу кілька OSD, що не дуже приємно і вносить додаткові труднощі на всю подальшу підтримку, яка скельюється зі зростанням кластера.

Ортодоксальність даного методу в його гнучкості та масштабованості. Схема така, що у нас є пул із холодними даними та пул із гарячими. При частому зверненні до об'єкта, який ніби нагрівається і потрапляє в гарячий пул, який складається зі швидких SSD. Потім, якщо об'єкт остигає, він потрапляє в холодний пул із повільними HDD. Дана схема дозволяє легко змінювати SSD у гарячому пулі, який у свою чергу може бути будь-якого розміру, бо параметри нагрівання та охолодження регулюються.

4.2 Hyperledger

Hyperledger Foundation — це спільний проект із відкритим вихідним кодом, створений для просування технології блокчейну в різних галузях. Він був заснований у 2015 році Linux Foundation і з тих пір став однією з найвідоміших і найактивніших спільнот розробників блокчейнів, дослідників і технологічних компаній у всьому світі.

Основною метою Hyperledger Foundation є сприяння розвитку технологій блокчейну корпоративного рівня, інструментів і фреймворків, які можна використовувати для створення безпечних, сумісних і масштабованих рішень на основі блокчейну для різних випадків використання. Для досягнення цієї мети Фонд об’єднує галузевих експертів, науковців і розробників з різних галузей для співпраці над проектами з відкритим кодом, які вирішують найнагальніші проблеми, що постають перед екосистемою блокчейну.

Hyperledger Foundation наразі розміщує кілька блокчейн-проектів з відкритим кодом, кожен зі своїми унікальними функціями та варіантами використання. Ось деякі з найбільш помітних проектів Фонду:

Hyperledger Fabric: Hyperledger Fabric — це модульна блокчейн-платформа корпоративного рівня, призначена для побудови приватних дозволених блокчейн-мереж. Fabric забезпечує високу гнучкість і масштабованість, дозволяючи розробникам проектувати та розгортати власні блокчейн-рішення, які відповідають конкретним вимогам бізнесу. Він також пропонує розширені функції конфіденційності та конфіденційності, що робить його ідеальним для випадків використання, які потребують суворого контролю доступу до даних.

Hyperledger Sawtooth: Hyperledger Sawtooth — це ще одна блокчейн-платформа корпоративного рівня, призначена для створення рішень розподіленої книги. Sawtooth використовує унікальний алгоритм консенсусу під назвою «Доказ часу, що минув» (PoET), який забезпечує кращу масштабованість і пропускну здатність порівняно з іншими алгоритмами консенсусу. Sawtooth також дуже модульний, що дозволяє легко налаштовувати та розширювати його для конкретних випадків використання

Hyperledger Indy: Hyperledger Indy — це блокчейн-платформа, розроблена спеціально для керування ідентифікацією та перевірки. Він пропонує децентралізовану систему керування ідентифікацією, яка дозволяє особам контролювати свої особисті дані та безпечно ділитися ними з довіреними сторонами. Indy створено з використанням передової криптографії та технологій підвищення конфіденційності, що робить його ідеальним для випадків використання, які потребують надійної перевірки особи та захисту конфіденційності.

Hyperledger Besu: Hyperledger Besu — це клієнт Ethereum корпоративного рівня, розроблений як для публічних, так і для приватних блокчейн-мереж. Besu надає широкий спектр функцій та інструментів для створення та

розгортання смарт-контрактів, що робить його ідеальним для випадків використання, які потребують складної логіки та функціональності смарт-контрактів.

Hyperledger Foundation пропонує кілька переваг своїм членам і ширшій блокчейн-спільноті. Деякі з ключових переваг включають:

1. Співпраця: Hyperledger Foundation надає платформу для співпраці між розробниками, дослідниками та галузевими експертами з різних галузей. Ця співпраця дозволяє розробляти інноваційні блокчейн-рішення, які відповідають конкретним вимогам бізнесу та потребам галузі.
2. Відкритий вихідний код: усі проекти Hyperledger є відкритими, тобто вихідний код є у вільному доступі для перегляду, використання та зміни. Цей підхід із відкритим кодом заохочує прозорість, співпрацю та інновації в екосистемі блокчейну.
3. На основі стандартів: проекти Hyperledger розробляються з використанням галузевих стандартів і найкращих практик, що гарантує, що отримані рішення є сумісними, безпечними та масштабованими.
4. Підтримка: Hyperledger Foundation надає підтримку та ресурси своїм членам, включаючи технічну допомогу, документацію та навчання. Ця підтримка допомагає членам створювати та розгортати успішні блокчейн-рішення.

4.3 Zk-rollups

ZK-зведення (Zero-Knowledge Rollups) — це тип рішення для масштабування рівня 2 для блокчейнів, який спрямований на збільшення

пропускної здатності транзакцій при одночасному зниженні витрат і збереженні високого рівня безпеки.

У ZK-зведенні транзакції обробляються поза мережею та об’єднуються в одну транзакцію, яка потім надсилається в основний блокчейн як доказ. Доказ перевіряється смарт-контрактом у головному ланцюзі, який гарантує, що транзакції дійсні та що не було шахрайських дій.

Ключовим нововведенням ZK-rollups є використання доказів з нульовим знанням, які дозволяють користувачам доводити дійсність своїх транзакцій, не розкриваючи жодних базових даних. Це забезпечує високий рівень конфіденційності та безпеки, а також дозволяє значно покращити масштабованість.

Останніми роками ZK-rollups стають все більш популярними, особливо в екосистемі Ethereum. Вони використовувалися для підтримки широкого спектру децентралізованих програм, включаючи децентралізовані обміни, протоколи кредитування та ігрові платформи.

4.4 Zk-SNARKs

Стислий неінтерактивний аргумент знань із нульовим знанням (ZK-SNARKs) — це криптографічна технологія, яка дозволяє перевіряти твердження без розкриття будь-якої додаткової інформації про нього.

ZK-SNARK стають все більш популярними в технології блокчейн, оскільки вони надають спосіб перевірити правильність обчислень без необхідності розкривати будь-які вхідні чи вихідні дані. Це особливо корисно в блокчейн-додатках, орієнтованих на конфіденційність, де важлива конфіденційність даних.

Ось як працюють ZK-SNARK:

- Фаза налаштування: довірена сторона генерує набір загальнодоступних параметрів, які використовуються в процесі створення доказів і перевірки. Ці параметри є загальнодоступними для будь-кого.
- Етап доведення: довідник створює доказ того, що твердження є істинним, не розкриваючи жодної додаткової інформації про твердження. Проверник робить це шляхом створення набору зашифрованих обчислень, які відповідають вхідним даним оператора. Потім ці обчислення перетворюються на низку обмежень, які можна перевірити на загальнодоступних параметрах, створених на етапі налаштування.
- Етап перевірки: верифікатор перевіряє докази на загальнодоступні параметри, щоб переконатися, що твердження правдиве, не розкриваючи жодної додаткової інформації. Верифікатору не потрібно знати вхідні чи вихідні дані обчислень, лише те, що вони правильні.

ZK-SNARK мають ряд важливих застосувань у світі блокчейнів. Наприклад, їх можна використовувати для реалізації технологій підвищення конфіденційності, таких як конфіденційні транзакції, коли суми транзакцій приховані від громадськості. Їх також можна використовувати для створення масштабованих і безпечних децентралізованих програм, як у випадку з ZK-зведеннями.

Однак важливо зазначити, що ZK-SNARK потребують інтенсивних обчислень і потребують значної потужності обробки для створення та перевірки доказів. Крім того, процес довіреного налаштування може бути потенційною слабкою стороною, якщо параметри не генеруються належним чином або скомпрометовані.

РОЗДІЛ 5

СТРУКТУРА ПРОЕКТУ

5.1 Архітектура проекту

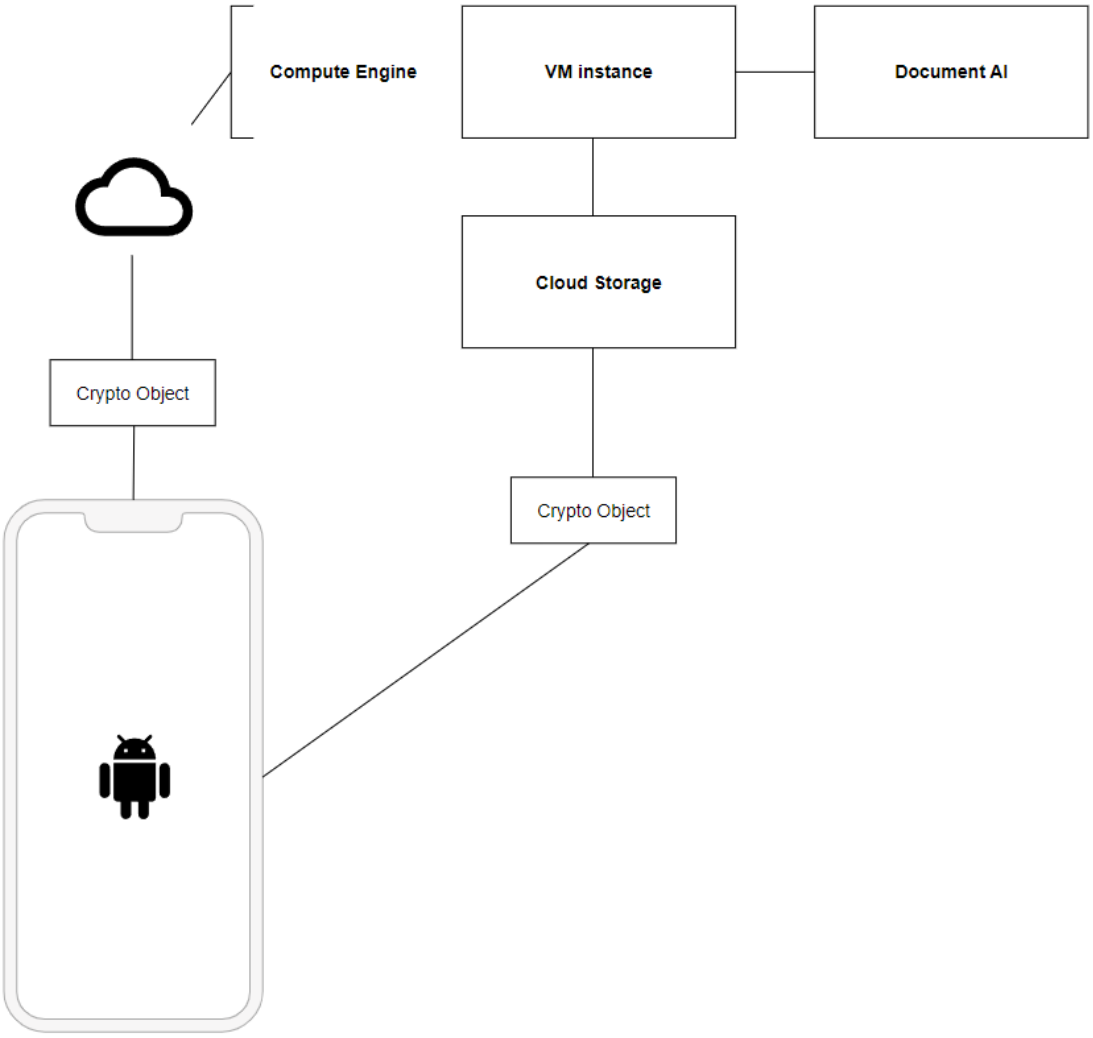

Рис. 5.1. Процедура взаємодії компонентів

На рис. 5.1 зображено: Crypto Object – зашифрований об’єкт, який містить у собі файл на обробку. Compute Engine – інструмент GCP, який дозволяє розгорнути інфраструктуру, необхідну для потрібної нам обробки

інформації. VM instances – віртуальні машини, на яких буде розгортатися інфраструктура. Document AI – апі GCP, який дозволяє використовувати нейронну мережу для аналізу документів. Cloud Storage – сервіс GCP, який дозволяє нам зберігати копії документів, які в подальшому можна буде використати для дублювання потрібного документу.

5.2. Архітектура мобільного застосунку

У проекті я використовував архітектурний паттерн MVVM, про який я розповідав у першому розділі. Із загальним виглядом цього паттерна все зрозуміло, тепер я уточню більш детально на прикладі свого конкретного проекту.

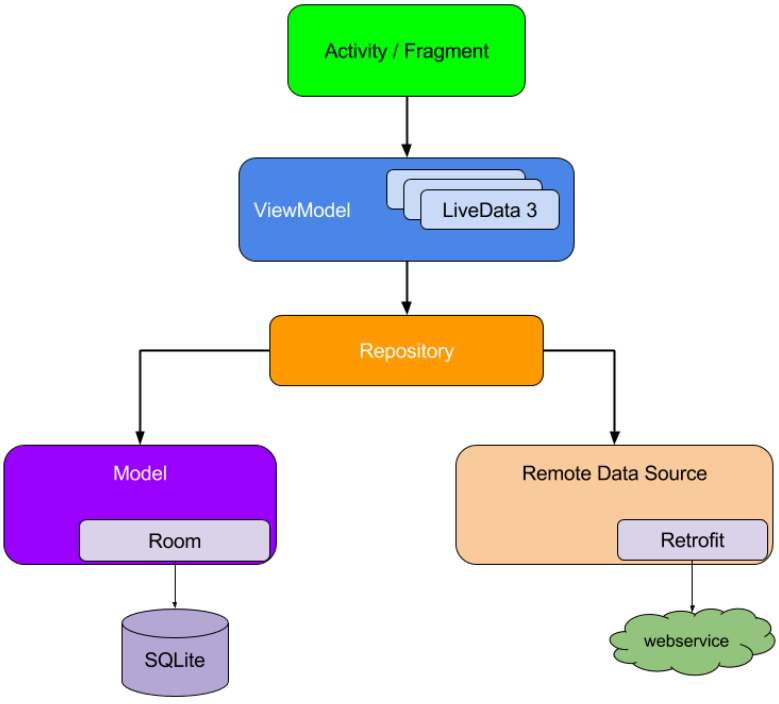

Рис. 5.2. MVVM архітектура

У розділі 1 я вже пояснював які компоненти зображені на рисунку.

Щодо реалізації цієї архітектури в моєму проекті, основні компоненти за їх застосування такі:

Activity – клас, який слугує входом у наш застосунок, саме з нього починається запуск застосунку мобільним пристроєм. Також у ньому ініціалізуються об’єкти, які в собі тримають посилання на ключові компоненти програми. Фрагменти також утворюються та зберігаються у контейнері, який ініціалізований у класі Activity.

Fragment – наслідувач класу Activity, який був упроваджений компанією Google для оптимізації ресурсів при виконанні програми на мобільному пристрої. Не може існувати без Activity, до якого прив’язаний. Це означає, що при знищенні об’єкта, фрагменти також знищуються і програма перестає виконання. У моєму проекті така реалізація, що у програмі 1 класс Activity і до нього прив’язанно посиланнями 24 класи Fragment.

ViewModel – класс, що містить у собі функціїї, які відповідають за відображення данних та взаємодію користувача з ними. Також цей класс дозволяє нам виконувати обробку зміни положення пристрою, так як данні зберігаються у об’єкти, з яких швидко дістати данні для відображення.

Repository – клас, що включає в себе функції для взаємодії із локальним сховищем даних, та віддаленим. В нашому випадку, цей клас є доволі великим, так як маємо справу з відправкою файлів на віддалений сервер та обробку отримання від нього зміненого зашифрованого об’єкту. Подальша обробка, розпакування та взаємодія теж буде керуватися з цього классу.

Model - клас, суть якого полягає у зображенні даних, їх формату, типу даних, для отримання або відправки. Він не несе якусь логіку в собі, в моїй реалізації, лиже зображення даних, їх модифікація проводиться в інших областях проекту.

Room – бібліотека для реалізації локального сховища у мобільних застосунках у вигляді таблиці SQLite. В моєму застосунку не використовується.

Remote data source – клас, у якому викликаються методи бібліотеки Retrofit для роботи з мережевими запитами. У цій області також реалізовуються класи і методи для налаштування з'єднання по веб-сокетам задля безшовного, безпечного з'єднання.

5.3. Архітектура хмарної інфраструктури

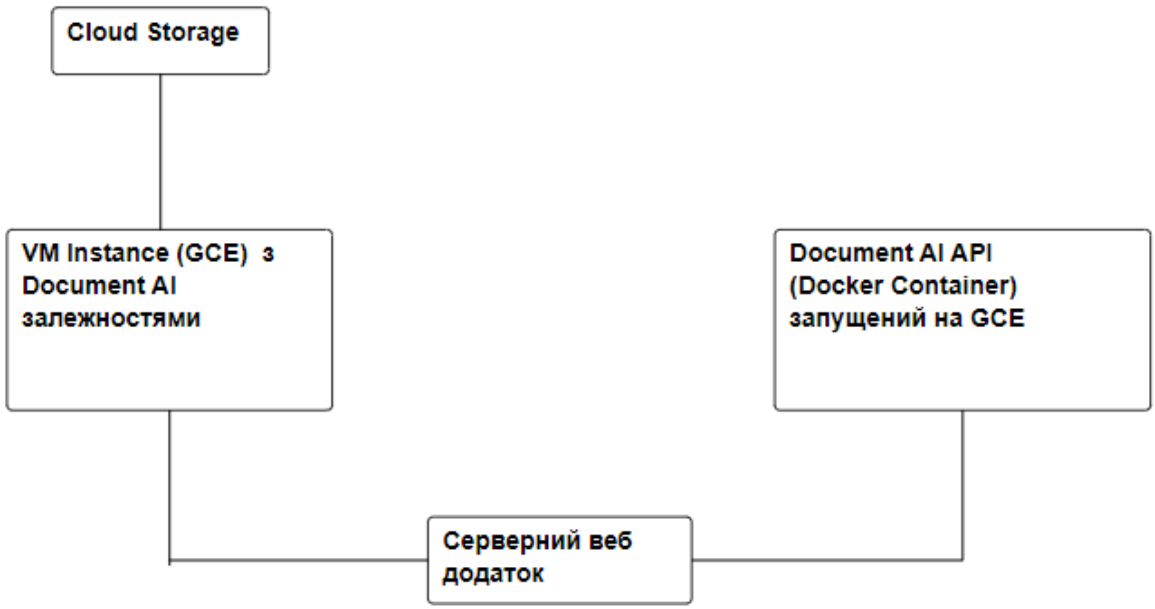

Рис. 5.3. Структура компонентів хмарної інфраструктури

На рис. 5.3 зображено діаграму залежності компонентів хмарної інфраструктури при обробці документів отриманих з клієнта. У цій архітектурі екземпляр віртуальної машини, що працює на Google Cloud Platform, налаштовано з необхідними залежностями для запуску API Document AI у контейнері Docker. Сервер додатків, який може бути

написаний у фреймворку, наприклад Flask або Django, взаємодіє з API через HTTP-запити.

API Document AI можна використовувати для обробки документів, що зберігаються в Cloud Storage. Сервер додатків може отримувати документи з хмарного сховища, передавати їх в API Document AI для обробки, а потім отримувати витягнуті дані назад з API.

Ця архітектура дозволяє масштабовано та гнучко обробляти документи за допомогою API Document AI, а також забезпечує безпечне та надійне середовище для сервера додатків і даних, які він обробляє.

5.4. Функціональний веб-застосунок

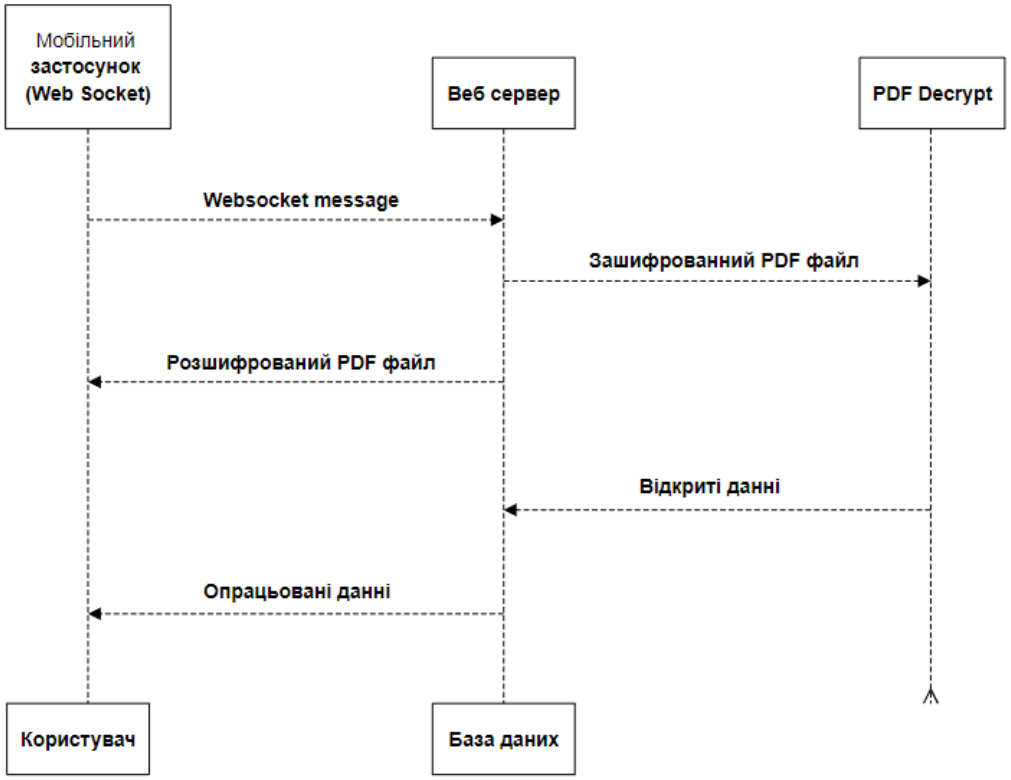

Рис. 5.4. Структура ngnx веб-застосунку

На рис. 5.4 зображено принцип функціонування веб-застосунку для приймання зашифрованого пакету даних, відправку їх на сервер, який

поєднаний зі сховищем з якого бере данні Document AI апі. Після обробки, розпакований набір даних відправляється на сервер від якого жде відповіді мобільний застосунок. Оброблені дані відображаються у застосунку.

Сам застосунок є стандартною реалізацією необхідного функціоналу з відкритого проекту на GitHub. Загалом, впровадження зв'язку через веб-сокет між сервером Nginx і застосунком Android може забезпечити швидкий і ефективний спосіб надсилання й отримання даних у режимі реального часу. Однак це вимагає ретельного планування та реалізації, щоб забезпечити безпечний і надійний зв'язок.

5.5 Шифрування об'єктів за допомогою Spongy Castle

Spongy Castle — це версія криптографічної бібліотеки Bouncy Castle, сумісна з Android. Він надає широкий спектр алгоритмів шифрування, включно з тими, які не включені у вбудовані API шифрування платформи Android. Загальний вигляд використання шифрування Spongy Castle у програмі Android наступний.

Додається Spongy Castle до свого проекту включивши його файли JAR як залежності. Крім того, було використано інструмент автоматизації збірки, Gradle, щоб керувати залежностями проекту.

Spongy Castle підтримує широкий спектр алгоритмів шифрування, включаючи AES, Blowfish, RSA тощо. Було обрано алгоритм RSA оскільки це безпечний і добре налагоджений алгоритм шифрування. Він використовується в багатьох програмах, включаючи безпечний веб-перегляд ,шифрування електронної пошти та цифрові підписи.

Щоб зашифрувати та розшифрувати дані, знадобиться секретний ключ. У програмі створено ключ за допомогою класу KeyPairGenerator для асиметричних алгоритмів шифрування, таких як RSA.

Шифрування даних проходить наступним чином: коли є ключ, можна використовувати клас Cipher для шифрування своїх даних. Спочатку створюється об'єкт Cipher за допомогою вибраного алгоритму шифрування та методу init() із параметром Cipher.ENCRYPT_MODE. Потім передаються дані відкритого документу в метод doFinal(), щоб зашифрувати їх.

Розшифрування проходить у іншому місці, а саме у веб-застосунку: щоб розшифрувати зашифровані дані, створено інший об'єкт Cipher, використовуючи той самий алгоритм шифрування та метод init() із параметром Cipher.DECRYPT_MODE. Потім передаються зашифровані дані методу doFinal(), щоб розшифрувати їх.

Я використав Android KeyStore API для зберігання ключа в зашифрованому вигляді на пристрої.

Слід пам'ятати про безпеку. Застосовуючи шифрування у своїй програмі Android, важливо пам'ятати про потенційні вразливості безпеки, як-от проблеми з керуванням ключами або атаки побічних каналів. Обов'язково дотримуватись найкращих практик і звертатися до відповідних ресурсів безпеки таких як OWASP Mobile Application Security.

Загалом Spongy Castle може бути корисним інструментом для реалізації шифрування в програмах Android, надаючи доступ до широкого спектру алгоритмів шифрування. Однак, як і будь-яке впровадження шифрування, воно вимагає ретельного розгляду питань безпеки та найкращих практик.

5.6 Аналіз документу процесором

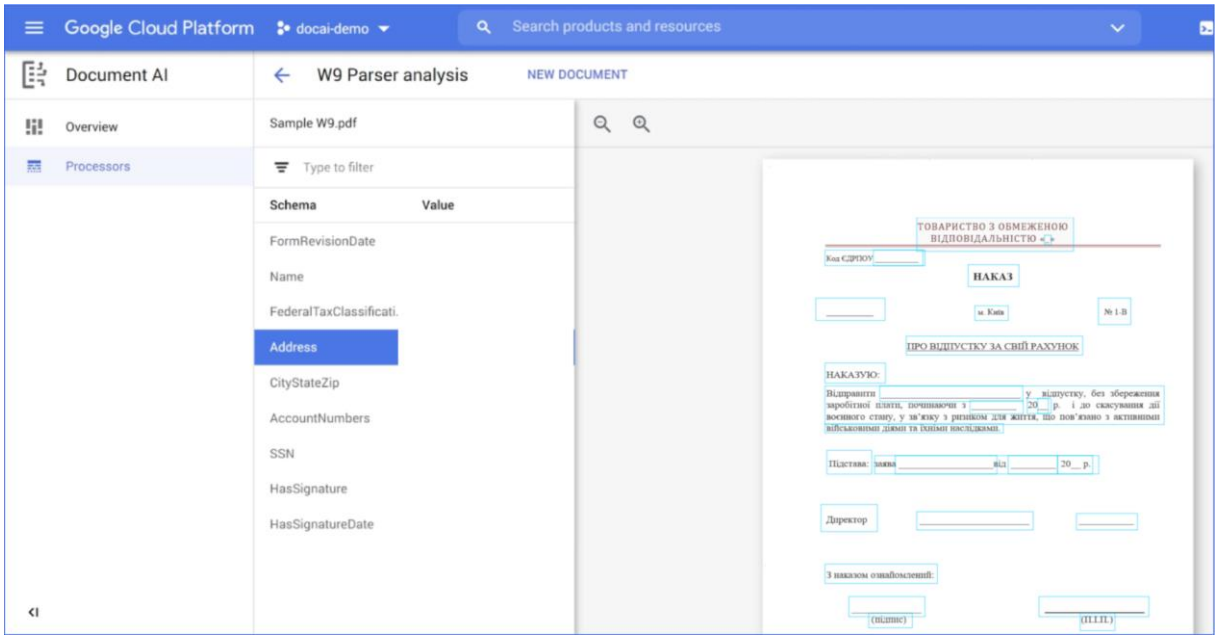

Рис. 5.5. Консоль аналітики документів

На рис. 5.5 зображена консоль аналітики Document AI що встановлена на віртуальній машині, яка розгорнута у хмарному сервісі GCP. Справа можемо побачити стовпчик, де можна обрати різні процесори на огляд роботи. Посередині стовпчик у якому таблиця, що відповідає полю, яке відмітив процесор, та значення яке стоїть у цьому полі, в нашому випадку не заповнене.


ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ «_»

Код ЄДРПОУ _____

НАКАЗ

_____ м. Київ № 1-В

ПРО ВІДПУСТКУ ЗА СВІЙ РАХУНОК

НАКАЗУЮ:

Відправити _____ у відпустку, без збереження заробітної плати, починаючи з _____ 20__ р. і до скасування дії воєнного стану, у зв'язку з ризиком для життя, що пов'язано з активними військовими діями та їхніми наслідками.

Підстава: заява _____ від _____ 20__ р.

Директор _____ _____

З наказом ознайомлений:

_____ _____

(підпис) (П.І.П.)

Рис. 5.6. Приклад обробленого файлу з документом

На рис. 5.6 бачимо, що процесор розпізнає поля з текстом, а також поля для заповнення. Саме розпізнавання полів для заповнення я використав у своїй роботі. Принцип такий, що веб-додаток поміщає у хмарне сховище розшифрований файл у якому знаходиться документ, після визначення місць у яких знаходяться поля для заповнення, ці параметри передаються разом із документом на мобільний застосунок, де триває заповнення необхідних полів документу.

РОЗДІЛ 6

МОБІЛЬНИЙ ЗАСТОСУНОК CLOUDSIGN

Рис. 6.1. Головний екран застосунку

На рис. 6.1 зображено: Add file – меню вибору з рис. 6.2. Scan Document – екран камери пристрою з рис. 6.3. Next – кнопка переходу на екран підписання, або редагування документу, не активна до моменту додавання або сканування хоча б однієї сторінки.

Рис. 6.2. Меню вибору джерела вкладень

На рис. 6.2 зображено: Scan – перехід на екран камери пристрою для сканування документу. Gallery – перехід на екран перегляду галереї пристрою с можливістю підтягнути раніше зроблені фото документів. Files – перехід на екран перегляду файлової системи пристрою з можливістю підтягнути завантажені вкладення.

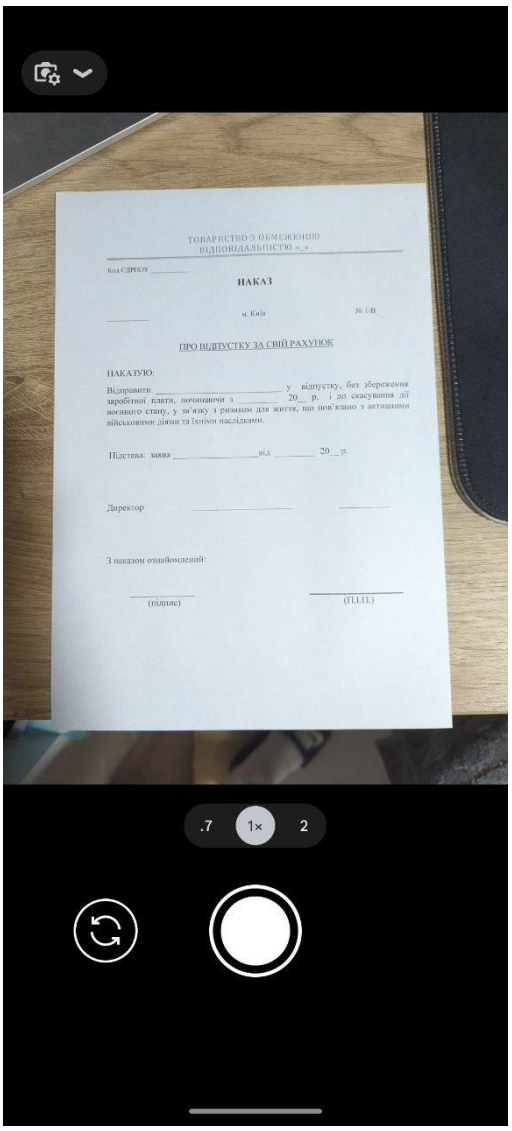

Рис. 6.3. Екран сканування документів за допомогою камери пристрою

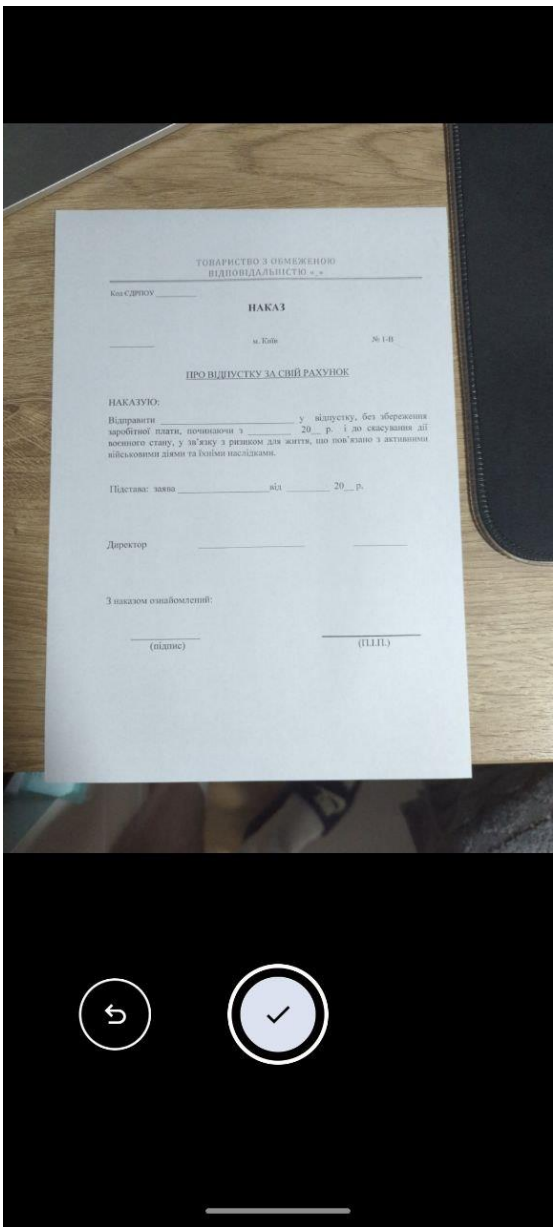

Рис. 6.4. Екран підтвердження успішного сканування документів за допомогою камери пристрою з можливістю повторити ще раз.

На рис. 6.4 показана реалізація сканування документів за допомогою камери мобільного пристрою використовуючи стандартну реалізацію виклику камери та роботи зі збереженням там використання зображення.

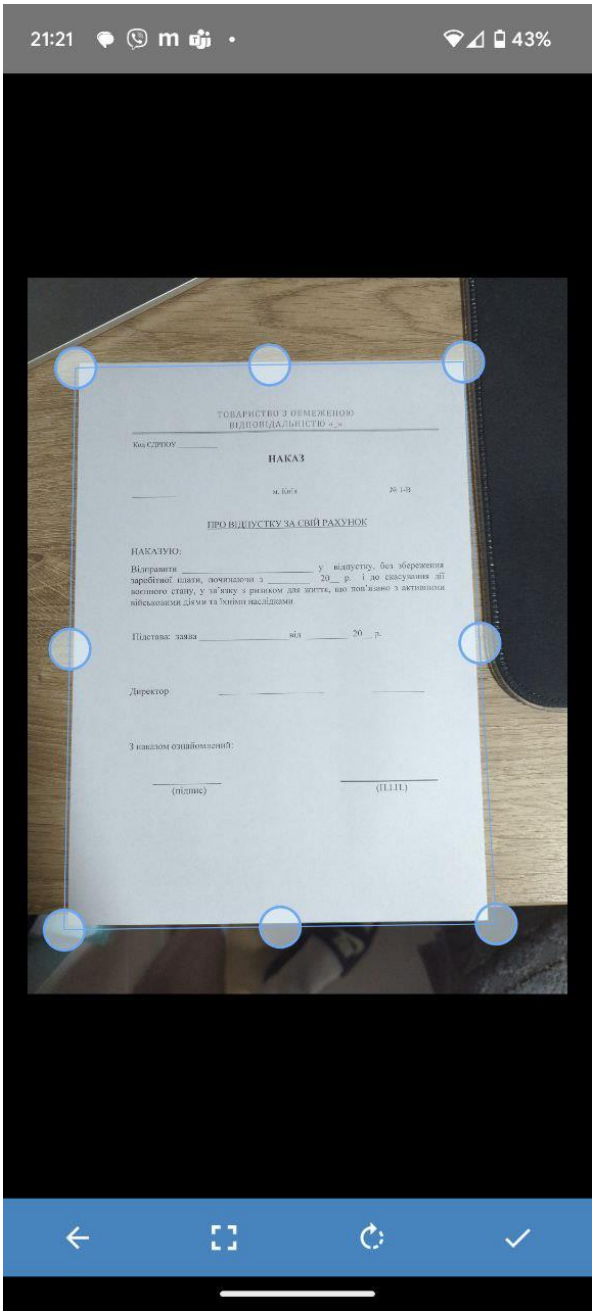

Рис. 6.5. Екран виділення меж документу із загального зображення

На рис. 6.5 зображено екран виділення по контуру меж необхідної частини зображення для подальшої роботи з документом. Функціонал реалізовано за допомогою загальнодоступної бібліотеки під назвою SimpleCropView. Стрілка назад повертає на попередній екран для повторного сканування, наступна кнопка скидає виділення документу до початкового стану, третя кнопка змінює орієнтацію зображення, остання кнопка відправляє зображення на хмарний сервіс на обробку.

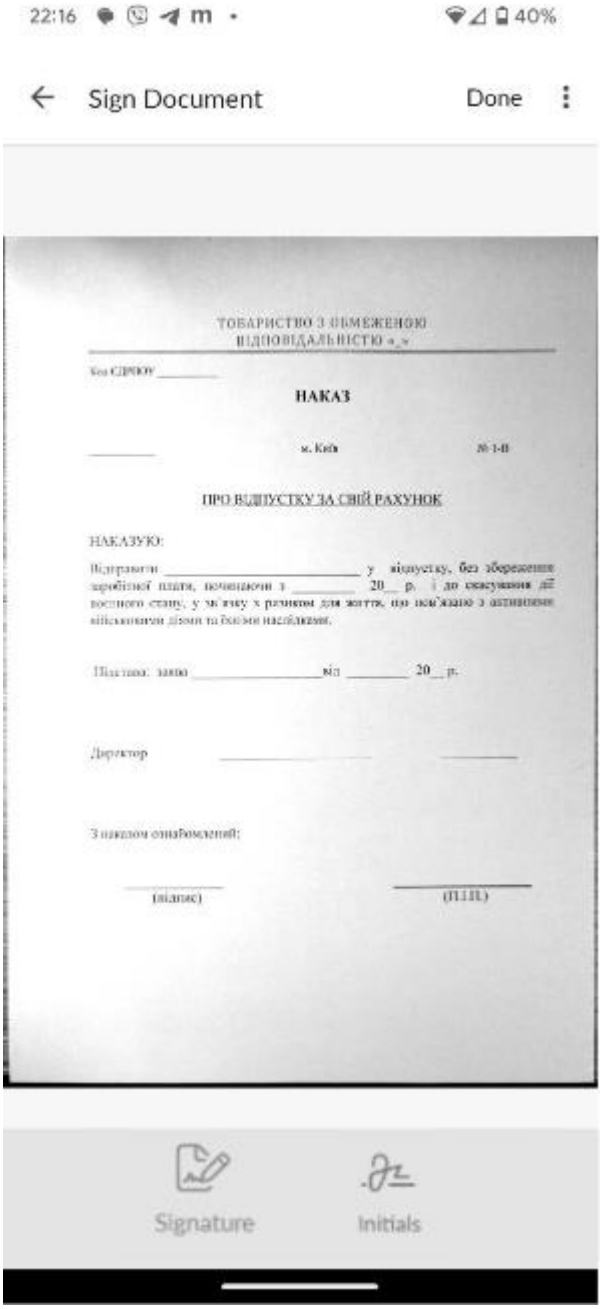

Рис. 6.6. Екран огляду документу для подальшого підписання

На рис. 6.6 зображено: Signature – перехід на екран створення підпису (рис. 6.7), Initials – екран створення підпису у вигляді ініціалів. Стрілка назад повертає на попередній екран виділення документу. Done – підтвердження закінчення роботи з документом, відправляє на екран поширення файлу (рис. 6.8).

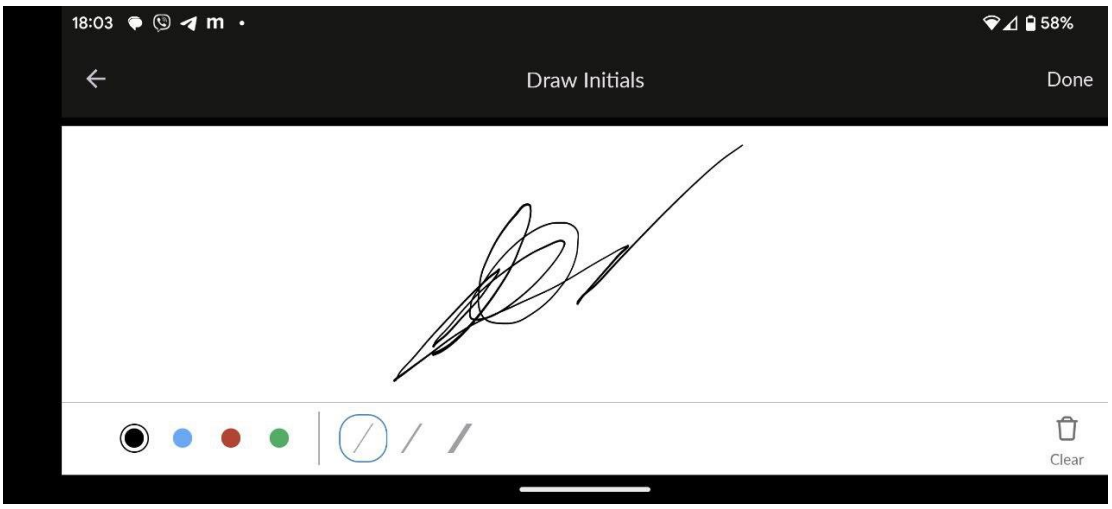

Рис. 6.7. Екран створення підпису

На рисунку 6.7 зображено: Стрілка назад – повертає на екран огляду та редагування документу. Done – зберігає підпис як векторний малюнок у локальному сховищі, що дає змогу накладати його поверх полів для підпису без додаткової обробки так як малюнок має прозорий фон. Вибір кольору підпису, товщини лінії, а також кнопка для очистки поля для створення підпису.

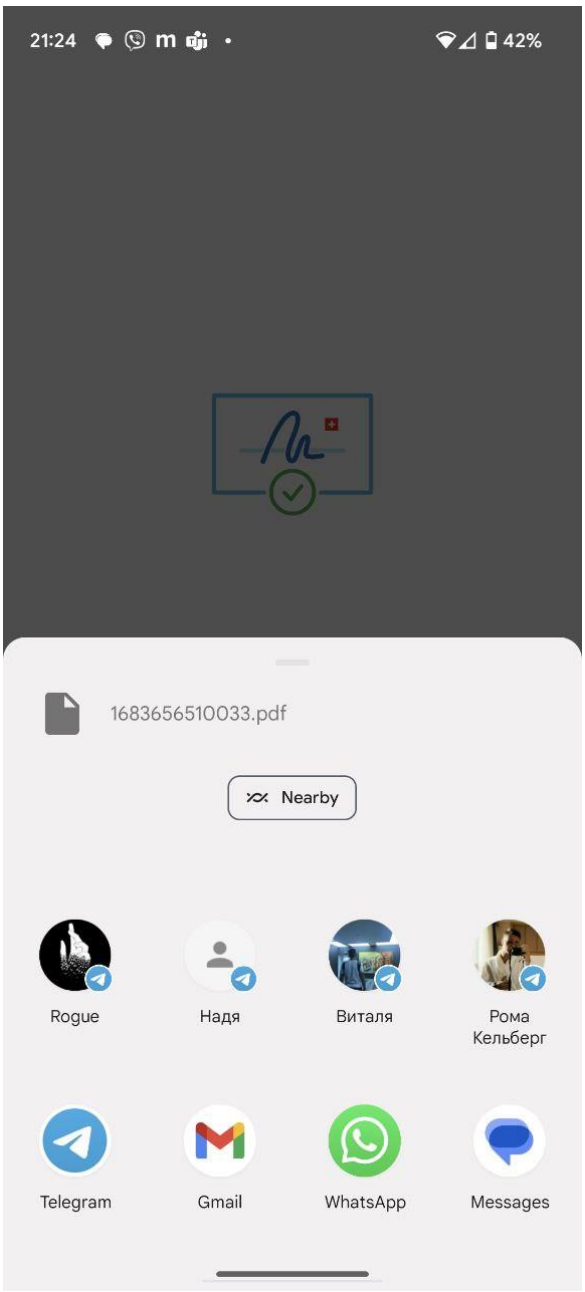

Рис. 6.8. Екран поширення файлу

На рис. 6.8 зображено екран, на якому викликається стандартне меню вибору способу поширення документу. Серед запропонованих варіантів – встановленні месседжери, електронная пошта та смс, а також у верхньому рядку розміщено останні використанні методи поширення.

ВИСНОВКИ

Мені вдалося побудувати рішення для оптимізації обчислювального процесу при документообігу. Ідея полягала в тому, щоб отримати максимально зрозумілий і зручний застосунок, який виконує свої задачі безвідмовно, щоби не відбувалось у країні. Існуючі рішення спирались на обчислювальні потужності пристрою, на якому проводяться маніпуляції, у моєму ж випадку всі ресурсномісткі обчислення проводяться у хмарному середовищі, захищеному компанією з хорошою репутацією та головне, операції можуть виконуватись на будь якому необхідному пристрої.

Швидкість обчислень буде залежати від потреби замовника такого рішення, так як потужність і кількість віртуальних середовищ можна розгортати інфраструктуру доволі швидко і з гнучким налаштуванням. Мобільний додаток побудовано з використанням останніх технологій, враховані більшість випадків зловмисного втручання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stefano Ferilli. Automatic Digital Document Processing and Management: Problems, Algorithms and Techniques (Advances in Computer Vision and Pattern Recognition) 2011th Edition. URL: https://www.amazon.com/Automatic-Digital-Document-Processing-Management/dp/0857291971
2. Straive. Intelligent Document Processing - Playbook to the Executives URL:https://www.straive.com/e-books/intelligent-document-processing-playbook-to-the-executives
3. J.C van Vliet and J.B Warmer. An Annotated Bibliography on Document Processing, 2010. URL: https://www.cambridge.org/core/books/abs/text-processing-and-document-manipulation/an-annotated-bibliography-on-document-processing/B316E4683F0D33852156D5B87F3A17FD
4. GCP. Document AI overview. URL: https://cloud.google.com/document ai/docs/overview
5. GCP. Full processor and detail list. URL: https://cloud.google.com/document-ai/docs/processors-list
6. Android Developers. Documentation. URL: https://developer.android.com/docs
7. Ceph. Ceph Documentation. URL: https://docs.ceph.com/en/quincy/
8. Ethereum. Zero-Knowledge Rollups. URL: https://ethereum.org/en/developers/docs/scaling/zk-rollups/