КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТІМЕНІ ТАРАСА ШЕВЧЕНКА
ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ
Кафедра радіотехніки та радіоелектронних систем

«На правах рукопису»

Робота допущена до захисту в ЕК
рішенням кафедри радіотехніки та радіоелектронних систем
від ___ травня 2024 року, протокол № _____.
Завідувач кафедри доктор фіз.-мат. наук, професор
_____ Ігор АНІСІМОВ

ДИПЛОМНА РОБОТА МАГІСТРА
на тему:

«РОЗРОБКА ТЕЛЕКОМУНІКАЦІЙНОГО ЗАСТОСУНКУ ДЛЯ ЗБОРУ, НАКОПИЧЕННЯ ТА ПРОВЕДЕННЯ ПЕРВИННОГО АНАЛІЗУ СТАНУ ДОРОЖНЬО-ТРАНСПОРТНОЇ МЕРЕЖІ»

Виконав:
студент 2-го курсу магістратури
денної форми навчання
спеціальності 172 - Телекомунікації та радіотехніка
ОНП «Інформаційна безпека телекомунікаційних систем і мереж»
Веренич Данило Дмитрович_____

Науковий керівник:
канд. фіз.-мат. наук, доцент
Кононов Михайло Володимирович _____

Рецензент:
канд.тех. наук, доцент
зав. каф. Інформаційних технологій
Київського Національного Університету Будівництва і Архітектури
Гончаренко Тетяна Андріївна_____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент Данило ВЕРЕНИЧ

Київ – 2024

РЕФЕРАТ

Дипломна робота: 46 с., 1 рис., 10 дод. (27с.), 15 джерел.

ТЕЛЕКОМУНІКАЦІЙНИЙ ЗАСТОСУНОК, GIS, ДОРОЖНЬО-ТРАНСПОРТНА МЕРЕЖА, ВЕБ-КАРТОГРАФІЯ

Об’єкт розроблення – телекомунікаційний застосунок для збору, накопичення та проведення первинного аналізу стану дорожньо-транспортної мережі.

Мета роботи – збільшення зручності моніторингу та ефективності керування дорожньо-транспортною мережею на основі розподіленої системи отримання даних про її стан.

Розроблено систему для збору, накопичення та аналізу стану дорожньо-транспортної мережу, забезпечену повним набором інструментів для завантаження та обробки відповідної інформації. Для відображення накопичених даних, розроблено веб сервіс візуалізації мап у форматі растрових, або векторних зображень, відповідно до класифікації набору даних. Цей сервіс оптимізовано для роботи з растровими даними великого об’єму. Інформаційна безпека забезпечується автентифікацією на основі електронного цифрового підпису та журналюванням дій користувача та стану системи.

Компоненти даної системи наразі впроваджені і використовуються у додатках Єдиної державної системи у сфері будівництва, а також у регіональних додатках містобудівного кадастру. Деякі з сервісів проходять етап тестування для їх використання у додатках Єдиного державного реєстру адрес та додатку Містобудівного кадастру державного рівня.

У майбутньому планується масштабування системи в напрямку розширення варіацій об’єктів дорожньо-транспортної мережі та забезпечення балансування навантаження для збільшення швидкодії.

ЗМІСТ

ВСТУП 5
1. ТЕЛЕКОМУНІКАЦІЙНИЙ ЗАСТОСУНОК ДЛЯ ЗБОРУ, НАКОПИЧЕННЯ ТА ПРОВЕДЕННЯ ПЕРВИННОГО АНАЛІЗУ СТАНУ ДОРОЖНЬО-ТРАНСПОРТНОЇ МЕРЕЖІ 7
1.1. Принципи побудови телекомунікаційних застосунків 9
1.2. Геоінформаційні системи (GIS) 10
1.3. Віддалене сховище даних 13
1.3.1. Основні принципи побудови віддаленого файлового сховища 13
2. АРХІТЕКТУРА РОЗРОБЛЕНОГО ЗАСТОСУНКУ 16
2.1. База даних 18
2.1.1. PostgreSQL та PostGIS 19
2.2. Веб-сервери 20
2.3. Додаткові сервіси 21
2.3.1. Картографічний сервіс 21
2.3.2. Сервіс роботи з ЕЦП 22
2.4. Кешуючий сервіс 22
2.4.1. Redis. Переваги та недоліки 23
2.4.2. Архітектура кешуючого сервісу 24
2.5. Сховище даних на сонові технології AmazonS3 25
2.5.1. Переваги та недоліки 25
2.5.2. Архітектура сховища 26
2.6. Система резервного копіювання 27
2.6.1. Переваги та недоліки 27
2.6.2. Архітектура сервісу резервного копіювання 28
3. ОПИС РОЗРОБЛЕНОГО ЗАСТОСУНКУ 30
3.1. Сервіс веб-мап 30
3.1.1. Опис операцій, які реалізовані 31
3.2. Сервіс аутентифікації користувача на основі ЕЦП 33
3.3. База даних 35
3.4. Веб-додаток 36
3.5. Сховище даних 38
3.6. Сервіс кешування 40
3.7. Розгортання застосунку 41

ВИСНОВКИ 43
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 45
ДОДАТКИ 47

ВСТУП

В сучасному світі, де транспортна інфраструктура відіграє ключову роль у функціонуванні економіки та забезпеченні комфортного життя громадян, ефективне управління дорожньо-транспортною мережею набуває особливої актуальності. Швидке зростання міст, збільшення кількості транспортних засобів та необхідність забезпечення безпеки дорожнього руху ставлять перед фахівцями складні завдання з моніторингу, аналізу та управління дорожньо-транспортною інфраструктурою.

Телекомунікаційні технології, зокрема веб-застосунки, відіграють важливу роль у вирішенні цих завдань, надаючи інструменти для збору, зберігання та аналізу даних у реальному часі. Використання сучасних технологій, таких як геоінформаційні системи (GIS), хмарні сервіси для зберігання даних, системи кешування та масштабовані бази даних, дозволяє створювати потужні та ефективні системи управління дорожньо-транспортною мережею.

Метою даної дипломної роботи є розробка телекомунікаційного застосунку для збору, накопичення та проведення первинного аналізу стану дорожньо-транспортної мережі. У роботі розглядаються загальні принципи побудови таких застосунків, аналізуються сучасні технології та інструменти, що використовуються для реалізації подібних рішень, а також надаються практичні рекомендації щодо впровадження розробленої системи.

В рамках дослідження були розглянуті наступні аспекти:

- Принципи побудови телекомунікаційних (веб) застосунків для збору, накопичення та аналізу даних.
- Використання віддалених сховищ даних на базі хмарних сервісів, зокрема Amazon S3.
- Реалізація системи кешування за допомогою Redis для підвищення продуктивності та зниження навантаження на основну базу даних.
- Інтеграція геоінформаційних технологій (PostgreSQL з PostGIS) для обробки та аналізу просторових даних.

- Використання Docker для контейнеризації та розгортання веб-застосунків, забезпечуючи їх ізольоване та консистентне середовище на всіх етапах життєвого циклу.

Запропонована система дозволяє ефективно вирішувати завдання моніторингу та управління дорожньо-транспортною мережею, забезпечуючи високу продуктивність, надійність та безпеку даних. Результати роботи можуть бути використані для подальшого розвитку та впровадження подібних систем у різних галузях, що вимагають ефективного управління великою кількістю даних та прийняття оперативних рішень на основі аналізу цих даних.

1. ТЕЛЕКОМУНІКАЦІЙНИЙ ЗАСТОСУНОК ДЛЯ ЗБОРУ, НАКОПИЧЕННЯ ТА ПРОВЕДЕННЯ ПЕРВИННОГО АНАЛІЗУ СТАНУ ДОРОЖНЬО-ТРАНСПОРТНОЇ МЕРЕЖІ

У сучасному світі, де обсяги даних стрімко зростають, телекомунікаційні застосунки відіграють ключову роль у зборі, передачі та аналізі інформації. Це стає фундаментом для прийняття обґрунтованих рішень у різних галузях. Зокрема, у сфері управління дорожньо-транспортними мережами, телекомунікаційні застосунки забезпечують можливість ефективного моніторингу та аналізу стану доріг, що є критично важливим для забезпечення безпеки та оптимізації руху транспорту.[1]

З кожним роком обсяг даних зростає в геометричній прогресії, що ставить нові виклики перед існуючими телекомунікаційними системами. Телекомунікаційні застосунки, такі як мобільні додатки, датчики IoT та системи моніторингу, є критично важливими для збору та аналізу цих даних. Вони дозволяють збирати дані в реальному часі, передавати їх до центрів обробки і аналізувати для прийняття оперативних рішень.

Розвиток технологій телекомунікацій, таких як 5G, Інтернет речей (IoT), хмарні обчислення та великі дані, значно змінив підхід до управління дорожньо-транспортними мережами. Впровадження цих технологій дозволяє створювати більш інтегровані та ефективні системи, що забезпечують високу точність і швидкість обробки даних, необхідних для прийняття рішень у режимі реального часу.

У сфері дорожньо-транспортної інфраструктури важливо мати надійні дані про стан доріг, потоки транспорту, погодні умови та інші фактори, що впливають на безпеку і ефективність дорожнього руху. Застосування телекомунікаційних технологій дозволяє автоматизувати процес збору цих даних, знижуючи витрати та підвищуючи точність і своєчасність інформації.

Одним із головних завдань телекомунікаційних застосунків є забезпечення безперервного збору та аналізу даних. Це досягається за рахунок використання розподілених систем збору даних, які можуть працювати в реальному часі та

забезпечувати високу надійність і доступність інформації. Інтеграція цих систем з іншими компонентами ITS дозволяє створити єдину інформаційну екосистему, що забезпечує всебічний моніторинг і управління дорожньо-транспортною мережею.

Важливу роль у забезпеченні ефективного збору та аналізу даних відіграють сучасні методи аналітики та обробки інформації. Використання машинного навчання, штучного інтелекту та великих даних дозволяє здійснювати глибокий аналіз зібраної інформації, виявляти приховані закономірності та тренди, а також передбачати можливі проблеми та ризики. Це забезпечує більш обґрунтовані та ефективні рішення, спрямовані на покращення безпеки та ефективності дорожньо-транспортної мережі.

Зростання обсягів даних та розвиток технологій створюють нові можливості для підвищення ефективності та безпеки дорожньо-транспортної мережі. Однак, це також вимагає постійного вдосконалення існуючих систем і методів обробки інформації, а також впровадження нових технологій і підходів до управління даними. У цьому контексті, телекомунікаційні застосунки виступають ключовим інструментом для забезпечення надійного і ефективного управління дорожньо-транспортною інфраструктурою.

На завершення, варто зазначити, що роль телекомунікаційних застосунків у забезпеченні ефективного управління дорожньо-транспортними мережами буде тільки зростати. З впровадженням нових технологій та методів обробки даних, ці системи ставатимуть ще більш потужними та ефективними, забезпечуючи високу точність і швидкість обробки інформації, необхідної для прийняття рішень у режимі реального часу.

Таким чином, розвиток телекомунікаційних застосунків є критично важливим для забезпечення безпеки та ефективності дорожньо-транспортної інфраструктури, а також для підвищення якості життя громадян. Подальші дослідження та інновації в цій галузі сприятимуть створенню більш інтегрованих та ефективних систем, здатних вирішувати найскладніші аналітичні завдання і забезпечувати стабільне та безпечне функціонування транспортної мережі.

1.1. Принципи побудови телекомунікаційних застосунків

Побудова телекомунікаційних застосунків для збору, накопичення та аналізу даних дорожньо-транспортної мережі вимагає дотримання певних принципів, що забезпечують їх ефективність, масштабованість та безпеку. Ці принципи охоплюють модульність, забезпечення високої продуктивності, безпеку та конфіденційність даних, а також використання сучасних технологій і методів.

1.1.1. Модульність та масштабованість

Модульність є одним з основних принципів, що дозволяє створювати гнучкі системи, які легко масштабуються. Це означає, що систему можна розділити на окремі модулі або компоненти, кожен з яких відповідає за певну функцію. Такий підхід дозволяє зменшити складність системи, полегшити її розробку та підтримку, а також забезпечити можливість додавання нових функцій без суттєвих змін в існуючій архітектурі. [2]

Для досягнення модульності і масштабованості, телекомунікаційні застосунки часто використовують мікросервісну архітектуру. Кожен мікросервіс відповідає за окрему частину функціональності системи і може бути незалежно розроблений, розгорнутий та масштабований. Це дозволяє забезпечити високу гнучкість і адаптивність системи до змінних умов та вимог.

1.1.2. Забезпечення високої продуктивності

Для забезпечення ефективної роботи телекомунікаційних застосунків необхідно забезпечити високу швидкість обробки даних та мінімальні затримки. Це досягається за рахунок використання сучасних технологій обробки даних, таких як розподілені обчислення та технології 5G. [3]

Розподілені обчислення дозволяють розподілити завдання обробки даних між багатьма вузлами або серверами, що значно підвищує продуктивність і надійність системи. Використання технологій 5G забезпечує високу швидкість передачі даних та низькі затримки, що є критично важливим для систем, що працюють в режимі реального часу.

1.1.3. Безпека та конфіденційність даних

Забезпечення безпеки та конфіденційності даних є одним з ключових аспектів при розробці телекомунікаційних застосунків. Це включає використання шифрування, автентифікації користувачів та регулярного моніторингу безпеки системи. [4]

Одним із підходів до забезпечення безпеки даних є використання протоколів SSL/TLS для захищеної передачі даних. Шифрування даних на всіх етапах їх обробки і зберігання дозволяє захистити їх від несанкціонованого доступу. Крім того, використання багатофакторної автентифікації (MFA) забезпечує додатковий рівень захисту, знижуючи ризик компрометації облікових записів користувачів.

1.1.4. Використання сучасних технологій і методів

Сучасні телекомунікаційні застосунки повинні використовувати передові технології та методи, що забезпечують їх високу ефективність і надійність. До таких технологій належать Інтернет речей (IoT), хмарні обчислення, великі дані та штучний інтелект (AI).

Інтернет речей (IoT) забезпечує можливість збору даних з численних датчиків, встановлених на транспортних засобах та інфраструктурі. Ці дані використовуються для моніторингу стану доріг, аналізу трафіку та виявлення аномалій у реальному часі.[1] Хмарні обчислення дозволяють здійснювати масштабовану і гнучку обробку даних, забезпечуючи високу продуктивність і доступність системи.

Великі дані і методи машинного навчання дозволяють здійснювати глибокий аналіз зібраної інформації, виявляти приховані закономірності та тренди, а також передбачати можливі проблеми та ризики. [5] Це забезпечує більш обґрунтовані та ефективні рішення, спрямовані на покращення безпеки та ефективності дорожньо-транспортної мережі.

1.2. Геоінформаційні системи (GIS)

Геоінформаційні системи (GIS) є потужним інструментом для збору, зберігання, аналізу та візуалізації просторових даних, що дозволяє ефективно

управляти дорожньо-транспортною мережею. GIS-технології використовуються для створення картографічних представлень даних, моделювання транспортних потоків та аналізу географічних інформацій. Завдяки цьому, можна забезпечити комплексний підхід до моніторингу та управління станом дорожньо-транспортної мережі.[6]

1.2.1. Основи gis-технологій

GIS об'єднують дані з різних джерел, таких як GPS, IoT, бездротові сенсорні мережі та інші джерела просторової інформації, для створення інтерактивних карт та візуалізацій. Це дозволяє відображати реальний стан транспортної мережі у режимі реального часу, що є важливим для прийняття оперативних рішень.

GIS-системи складаються з наступних основних компонентів:

- Просторова база даних**:** місце зберігання географічних даних, яке дозволяє здійснювати їх швидкий пошук та обробку.
- Геоінформаційне програмне забезпечення**:** інструменти для аналізу, моделювання та візуалізації просторових даних.
- Користувацький інтерфейс**:** забезпечує взаємодію користувачів із системою, дозволяючи здійснювати аналіз даних та створювати звіти.

1.2.2. Візуалізація та аналіз даних у GIS

Візуалізація даних є одним з найважливіших аспектів використання GIS. Вона дозволяє представити складні дані у вигляді зрозумілих карт та діаграм, що значно полегшує їх аналіз та інтерпретацію. GIS дозволяє накладати різні шари даних, такі як транспортні потоки, стан дорожнього покриття, погодні умови та інші фактори, що впливають на дорожню ситуацію.

За допомогою GIS можна проводити наступні типи аналізу:

- Аналіз трафіку**:** оцінка інтенсивності та напрямків транспортних потоків, визначення вузьких місць та зон заторів.
- Просторовий аналіз: визначення географічних зон з підвищеною аварійністю, аналіз впливу інфраструктурних змін на транспортні потоки.

- Моделювання сценаріїв**:** прогнозування впливу різних сценаріїв, таких як закриття дороги або введення нових маршрутів громадського транспорту, на загальну ситуацію на дорогах.[7]

1.2.3. Інтеграція GIS з іншими технологіями

GIS-технології можуть бути інтегровані з іншими системами для забезпечення більш повного та точного аналізу даних. Наприклад, інтеграція з системами IoT дозволяє отримувати дані з численних датчиків у реальному часі, що підвищує точність і актуальність інформації.

Інтеграція з методами машинного навчання та великими даними дозволяє автоматично аналізувати великі обсяги даних та виявляти приховані закономірності та тренди. Це дозволяє більш точно прогнозувати можливі проблеми та приймати превентивні заходи для їх вирішення.[5]

1.2.4. Реальні приклади використання GIS у дорожньо-транспортних системах

Одним із прикладів використання GIS у дорожньо-транспортних системах є проект "Waze" від Google. Waze використовує дані GPS від користувачів для створення карт у реальному часі, що відображають поточні умови на дорогах, включаючи затори, аварії та інші події. Це дозволяє водіям отримувати актуальну інформацію та оптимізувати свої маршрути. [6]

Інший приклад – система моніторингу дорожньо-транспортної ситуації у Лондоні. Використовуючи дані з камер спостереження, датчиків трафіку та інших джерел, GIS-система дозволяє в режимі реального часу контролювати ситуацію на дорогах, планувати ремонтні роботи та оперативно реагувати на аварійні ситуації.[7]

1.2.5. Переваги використання GIS у дорожньо-транспортних мережах

Використання GIS у дорожньо-транспортних мережах надає ряд переваг:

- Покращення прийняття рішень**:** GIS дозволяє приймати більш обґрунтовані рішення, базуючись на актуальних та точних даних.
- Зниження витрат**:** автоматизація процесів збору та аналізу даних дозволяє знизити витрати на обслуговування транспортної інфраструктури.

- Підвищення безпеки**:** аналіз даних про аварії та дорожні умови дозволяє визначити небезпечні зони та прийняти превентивні заходи для підвищення безпеки.
- Покращення планування**:** GIS дозволяє більш ефективно планувати будівництво та ремонт доріг, оптимізуючи використання ресурсів.[8]

1.3. Віддалене сховище даних

Віддалене файлове сховище є критично важливою компонентою телекомунікаційних застосунків, які збирають, накопичують та аналізують дані про стан дорожньо-транспортної мережі. Забезпечення безпечного та ефективного зберігання великого обсягу даних, доступ до них з будь-якої точки світу, а також їх обробка та аналіз у режимі реального часу є важливими завданнями для таких систем.

1.3.1. Основні принципи побудови віддаленого файлового сховища

Масштабованість є одним з ключових принципів побудови віддаленого файлового сховища. Система повинна мати можливість ефективно обробляти збільшення обсягів даних без втрати продуктивності. Це досягається за рахунок використання розподілених обчислень, де дані зберігаються та обробляються на декількох серверах або в дата-центрах.

- Розподілені файлові системи**:** Використання розподілених файлових систем, таких як HadoopDistributedFileSystem (HDFS) або GoogleFileSystem (GFS), дозволяє розподіляти дані між багатьма вузлами. Це забезпечує високу масштабованість та продуктивність, дозволяючи системі обробляти великі обсяги даних паралельно.[7]
- Динамічне розширення ресурсів**:** Завдяки хмарним технологіям, ресурси можуть бути динамічно розширені в залежності від поточних потреб системи. Це дозволяє оптимально використовувати обчислювальні та зберігаючі ресурси, забезпечуючи стабільну роботу системи навіть при різких збільшеннях обсягів даних.[9]

Для забезпечення безперебійної роботи телекомунікаційних застосунків важливо забезпечити високу доступність та надійність файлового сховища. Це

досягається за рахунок реплікації даних на декілька вузлів, що дозволяє забезпечити безперервний доступ до даних навіть у випадку виходу з ладу одного з вузлів.

- Реплікація даних**:** Реплікація даних на кілька вузлів або дата-центрів забезпечує їхню високу доступність. У разі виходу з ладу одного вузла, дані залишаються доступними на інших вузлах, що забезпечує безперервність роботи системи.[4]
- Географічно розподілені кластери**:** Використання географічно розподілених кластерів дозволяє забезпечити додатковий рівень надійності. Дані зберігаються в різних фізичних локаціях, що захищає їх від регіональних катастроф та збоїв у роботі інфраструктури. [9]

Забезпечення безпеки даних є критичним аспектом при побудові віддаленого файлового сховища. Це включає кілька ключових аспектів, які забезпечують захист даних на всіх етапах їх зберігання та передачі.

Шифрування даних забезпечує захист від несанкціонованого доступу як під час передачі, так і під час зберігання. Під час передачі дані шифруються за допомогою протоколу SSL/TLS, що забезпечує захист від перехоплення та несанкціонованого доступу. Під час зберігання дані можуть бути зашифровані за допомогою алгоритму AES-256 або інших методів шифрування, що забезпечує високий рівень захисту даних.[9]

Для управління доступом до даних використовуються системи, які дозволяють створювати користувачів, групи та ролі, а також призначати їм політики доступу. Це забезпечує гнучке управління доступом та дозволяє забезпечити принцип мінімального привілею, коли користувачі мають доступ лише до тих ресурсів, які їм необхідні для виконання своїх обов'язків.

- Політики доступу**:** Використання політик доступу дозволяє точно налаштувати права доступу до даних на рівні об'єктів або баків. Це забезпечує високу гнучкість у налаштуванні безпеки та дозволяє обмежити доступ до чутливих даних.[9]

- Багатофакторна автентифікація**:** Використання багатофакторної автентифікації додає додатковий рівень безпеки, вимагаючи від користувачів підтвердження своєї особи за допомогою двох або більше методів автентифікації.[9]

Для забезпечення прозорості та безпеки, системи віддаленого зберігання даних підтримують інтеграцію з інструментами аудиту та моніторингу. Це дозволяє відстежувати та фіксувати всі запити до даних, включаючи створення, зміну та видалення об'єктів.

- AWS CloudTrail**:** Інструмент, який забезпечує аудит всіх дій, що відбуваються з даними. Це дозволяє відстежувати всі запити до даних та фіксувати їх, забезпечуючи додатковий рівень безпеки та прозорості.[9]
- AWS CloudWatch**:** Інструмент для моніторингу стану системи в режимі реального часу. Він дозволяє налаштовувати оповіщення у випадку виявлення підозрілої активності або перевищення заданих порогів, що дозволяє оперативно реагувати на потенційні загрози.[9]

2. АРХІТЕКТУРА РОЗРОБЛЕНОГО ЗАСТОСУНКУ

Архітектура застосунку є ключовим фактором для забезпечення його ефективної роботи, масштабованості, надійності та безпеки. У сучасному світі телекомунікаційні застосунки повинні обробляти великі обсяги даних у режимі реального часу, забезпечувати швидкий відгук на запити користувачів та інтеграцію з різноманітними сервісами та платформами. Це особливо важливо для застосунків, які здійснюють моніторинг та управління дорожньо-транспортною мережею, де кожна секунда має значення. На Рисунку 1 наведено схему архітектури даного застосунку.

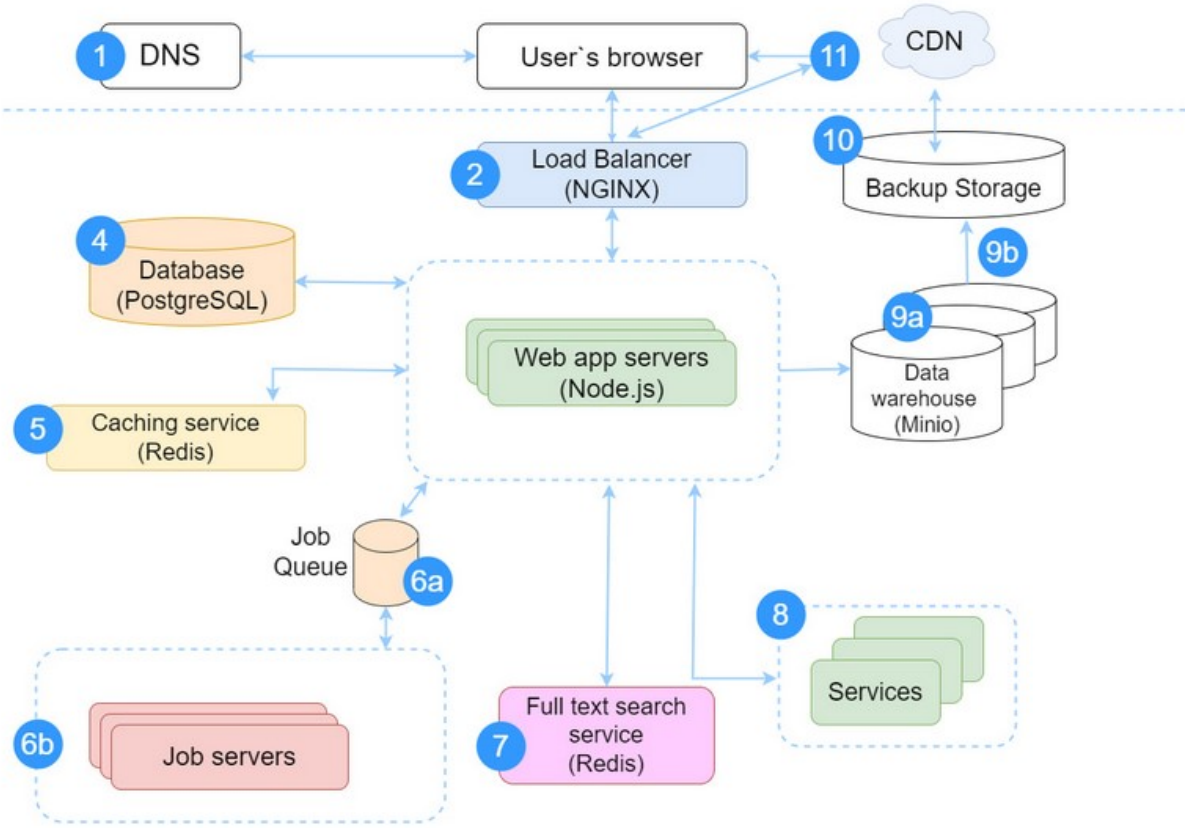


Рисунок 2.1 – схема архітектури застосунку

Метою цього розділу є представлення архітектури телекомунікаційного застосунку, яка забезпечує стабільність, ефективність та безпеку системи. В основу архітектури покладені такі принципи, як розподіленість, горизонтальне масштабування, висока доступність та безперервність обслуговування. Також

значну увагу приділено питанням безпеки даних, що включає шифрування, управління доступом та аудит.

На основі наданої схеми, архітектура застосунку складається з наступних ключових компонентів:

1. DNS (DomainNameSystem)**:** Забезпечує перетворення доменних імен у IP-адреси, що дозволяє користувачам знаходити та підключатися до серверів застосунку.

2. Балансувальник навантаження (LoadBalancer)**:** Використовується для розподілу вхідних запитів між кількома веб-серверами для забезпечення рівномірного навантаження та підвищення продуктивності.

3. Веб-сервери (WebAppServers): Відповідають за обробку запитів користувачів, виконання бізнес-логіки та взаємодію з іншими компонентами системи.

4. База даних (Database): Зберігає основні дані застосунку, забезпечуючи надійне та масштабоване зберігання інформації.

5. Кешуючий сервіс (CachingService): Прискорює доступ до часто використовуваних даних, знижуючи навантаження на базу даних.

6. Черга завдань (JobQueue) та сервери завдань (JobServers): Використовуються для асинхронної обробки завдань у фоновому режимі, що дозволяє основним серверам залишатися доступними для обробки нових запитів.

7. Сервіс повнотекстового пошуку (FullTextSearchService): Забезпечує швидкий пошук по великій кількості текстових даних.

8. Додаткові сервіси (Services): Виконують різні допоміжні функції, необхідні для роботи застосунку.

9. Сховище даних (DataWarehouse) та резервне копіювання (BackupStorage): Використовується для зберігання великих обсягів даних та резервних копій.

10. CDN (ContentDeliveryNetwork): Забезпечує швидку доставку статичних ресурсів до користувачів по всьому світу.

Ця архітектура побудована з використанням сучасних технологій та методів, що дозволяє забезпечити високий рівень продуктивності та надійності. Далі у розділі буде детально розглянуто кожен з компонентів архітектури, їх функції, взаємодію та важливість для загальної роботи системи.

2.1. База даних

У сучасних телекомунікаційних застосунках не обійтись без бази даних. Вони використовуються для зберігання великого об’єму структурованої інформації. Наразі існує два підходи до побудови БД:

- Реляційний.
- Не реляційни.

Так як, більшість компонентів дорожньо-транспортної мережі пов’язані між собою, вибір реляційної БД буде кращим ніж не реляційний, бо надає можливість будувати зв’язки.

Серед реляційних БД, вибір рішення, яке забезпечить можливість зберігання просторових даних, є також не таким тривіальним. Наразі існує декілька рішень для зберігання просторових даних всередині БД. Найпростішим з них є зберігання координат XYZу окремих колонках таблиці БД. Такий спосіб є найбільш простим, але у нього є кілька недоліків:

- Для зберігання використовується надлишкова кількість колонок
- Нема прив’язки до системи координат, що зменшує рівень доступності інформації
- Усі функції з просторового аналізу потрібно писати самостійно
- При збільшення кількості об’єктів, швидкодія алгоритмів просторового аналізу зменшується
- Нема індексації для пришвидшення роботи запитів

Для вирішення даних проблем, потрібно використовувати надбудови, які допомагають зберігати просторові дані з прив’язкою до систем координат та з можливістю побудови просторових індексів, для забезпечення швидкодії.

2.1.1. PostgreSQL та PostGIS

PostgreSQL є однією з найпопулярніших реляційних баз даних з відкритим кодом, яка забезпечує високу продуктивність, надійність та гнучкість. PostGIS розширює можливості PostgreSQL, додаючи підтримку географічних об'єктів та функцій для обробки просторових даних.

Неодмінною перевагою використання даного набору інструментів є те, що у них є підтримка геопросторової індексації об’єктів БД. Індексація є важливим механізмом для підвищення продуктивності запитів до бази даних. PostgreSQL та PostGIS підтримують різні типи індексів, що дозволяють швидко виконувати складні просторові запити. Загалом виділяють два основні типи індексів:

- GiST (GeneralizedSearchTree):є узагальненим індексним типом, який дозволяє створювати індекси для широкого спектру даних, включаючи просторові дані. GiST індекси дозволяють ефективно виконувати запити на основі просторових відношень, таких як перетин, включення та відстань.
- SP-GiST (Space-PartitionedGeneralizedSearchTree):є спеціалізованим типом індексів, оптимізованих для просторових даних. Вони використовують розділення простору для організації даних, що дозволяє ще ефективніше виконувати просторові запити.

Використання GiST індексів значно прискорює виконання запитів до просторових даних. Це досягається за рахунок зменшення кількості перевірок даних, які необхідно виконати для отримання результату.[10] SP-GiST індекси використовують алгоритми розділення простору, такі як квадродерева або октодерева, для організації даних. Це дозволяє швидко знаходити географічні об'єкти у визначених просторових інтервалах.[10] Побудова індексації об’єктів БД, використовуючи ці два типи індексації, надає модливість прискорення виконання аналітичних та просторових запитів до БД, що пришвидшує швидкодію та доступність.

Також, для забезпечення захисту даних від втрат, потрібно мати механізм реплікації даних. Реплікація – це створення копій самого себе. Реплікація даних є важливим механізмом для забезпечення високої доступності та надійності бази

даних. Вона дозволяє створювати копії бази даних на кількох серверах, що забезпечує безперервний доступ до даних навіть у разі виходу з ладу одного з серверів. PostgreSQL підтримує різні методи реплікації, включаючи синхронну та асинхронну реплікацію.[11]

Синхронна реплікація забезпечує високу консистентність даних між основним сервером (primary) і репліками (standbys). При синхронній реплікації транзакції на основному сервері завершуються лише після того, як зміни будуть записані на одну або більше реплік. Асинхронна реплікація дозволяє основному серверу завершувати транзакції без очікування підтвердження від реплік. Зміни передаються на репліки асинхронно, що забезпечує менші затримки у виконанні транзакцій. Для забезпечення швидкодії додатку. Також PostgreSQL надає можливість каскадної реплікації. Каскадна реплікація дозволяє реплікам отримувати дані не тільки від основного сервера, але й від інших реплік. Це знижує навантаження на основний сервер та забезпечує більш гнучке масштабування системи.

2.2. Веб-сервери

Для синхронізації та поєднання роботи усіх сервісів, потрібно мати певний обробчик команд, який буде виконувати роль проміжного клієнта між сервісами, та веб-застосунком. Веб-застосунок, який виконує роль такого обробчика, написаний на Node.js, що є ефективним середовищем виконання JavaScript на серверній стороні. Node.js забезпечує асинхронну обробку запитів, що дозволяє обробляти велику кількість одночасних запитів з мінімальними затримками.

Основні функції веб-застосунку на Node.js:

- Обробка запитів: Веб-застосунок отримує запити від користувачів через балансувальник навантаження (NGINX) і обробляє їх, взаємодіючи з іншими компонентами системи.
- Взаємодія з базою даних: Node.js сервери виконують запити до бази даних PostgreSQL для отримання та збереження даних.

- Кешування: Для підвищення продуктивності, веб-застосунок використовує кешуючий сервіс Redis для зберігання часто використовуваних даних.
- Обробка завдань: Завдання, що потребують значних ресурсів або часу на виконання, додаються до черги завдань (JobQueue) і обробляються окремими серверами завдань (JobServers).
- Пошук: Використання повнотекстового пошуку на основі Redis дозволяє швидко знаходити необхідну інформацію.

2.3. Додаткові сервіси

У веб-застосунку, який працює з просторовими даними, повинна бути можливість відображення цих даних. Тим більше, застосунок повинен мати можливість відображати данні на картографічній підкладці, бо данні мають також і географічну прив’язку на місцевості. Також сюди відноситься сервіс роботи з електронним цифровим підписом, так як до застосунку є вимога до отримання користувачем доступу з прив’язкою до реальних ідентифікаційних даних (РНОКПП).

2.3.1. Картографічний сервіс

Сервіс для відображення геопросторових даних на картографічній підкладці, повинен виконувати наступні функції:

- Відображення стандартної картографічної підкладки у форматі растрових зображень формату GeoTIFF;
- Відображення картографічної підкладки з зовнішніх джерел інформації;
- Відображення динамічних векторних даних;
- Відображення даних з бази даних у форматі векторних зображень.

Також, потрібно забезпечити можливість завантаження власних картографічних підкладок користувачем. Для цього, сервіс повинен мати власне сховище даних, та мати функціонал файлового провідника, для завантаження, вивантаження та модифікації файлів, для коректного відображення.

Також сервіс повинен бути гнучким у налаштуванні, мати структурований набір параметрів запиту. Сервіс повинен мати власну систему логування стану системи з можливістю її перегляду адміністратором. Також, для налаштування перегляду логів, бажано використовувати набір інструментів ELK, що забезпечує зручний перегляд логів та ведення статистики.

2.3.2. Сервіс роботи з ЕЦП

Сервіс роботи з ЕЦП відповідає за ідентифікацію користувача, для налаштування доступу до адміністративної частини застосунку тільки користувачам з відповідним рівнем доступу до системи. Рівні доступу повинні налаштовуватись через адмін частину, але доступ до цих налаштувань повинен бути обмежений невеликим набором головних адміністраторів.

Цей сервіс є складовою системи інформаційної безпеки і відповідає за роботу з криптографічною бібліотекою, яка була розроблена АТ «Інститут Інформаційних Технологій» на замовлення ДП «Дія». Цей сервіс відповідає за ідентифікацію користувача у системі на основі його унікального ключа цифрового підпису. Даний сервіс повинен відповідати наступним умовам:

- бути повністю незалежним;
- працювати віддалено;
- виконувати наступні операції:
  - розшифровка даних користувача при вході у систему через електронний підпис;
  - підпис документів електронним підписом;
  - отримувати з файлу інформацію про усіх підписантів;
- сервіс повинен відповідати усім сучасним стандартам безпеки;
- сервіс повинен бути стійким до зовнішнього втручання у його роботу.

2.4. Кешуючий сервіс

Кешування є важливою складовою частиною сучасних веб-застосунків, що дозволяє підвищити продуктивність та знизити навантаження на основні бази даних. Кешуючий сервіс зберігає часто використовувані дані в пам'яті, що дозволяє значно прискорити доступ до них. Одним із найпопулярніших

інструментів для реалізації кешування є Redis — високопродуктивна, відкрита база даних типу ключ-значення, яка зберігає дані в пам'яті.

2.4.1. Redis. Переваги та недоліки

Redis (RemoteDictionary Server) є високопродуктивною системою управління базами даних типу ключ-значення з відкритим кодом, яка підтримує широкий спектр структур даних, таких як рядки, списки, множини, хеші та інші. Основні переваги Redis включають:

- Висока продуктивність: Зберігання даних у оперативній пам'яті забезпечує дуже швидкий доступ до них. Redis може обробляти мільйони операцій читання та запису за секунду, що робить його ідеальним для використання у якості кешуючого сервісу. Операції в Redis виконуються з мінімальною затримкою, зазвичай менше однієї мілісекунди, що забезпечує швидкий відгук системи.[12]
- Масштабованість: Redis підтримує горизонтальне масштабування за допомогою кластерів, що дозволяє розподіляти дані між кількома вузлами. Реплікація даних між основним сервером та репліками забезпечує високу доступність та надійність даних.[13]
- Гнучкість та розширюваність: Redis має клієнтські бібліотеки для більшості популярних мов програмування, таких як Python, Java, JavaScript, PHP, Ruby та інші, що полегшує інтеграцію з веб-застосунком. Підтримка різних структур даних дозволяє використовувати Redis для реалізації складних сценаріїв кешування.[12]

Недоліки Redis:

- Обмеження пам'яті: Оскільки Redis зберігає всі дані в оперативній пам'яті, обсяг даних, які можна зберігати, обмежений обсягом доступної пам'яті на сервері. Це може бути проблемою для додатків, які потребують зберігання великих обсягів даних.
- Втрата даних: У разі виходу з ладу сервера Redis, який не налаштований на регулярне збереження знімків даних на диск або реплікацію,

можлива втрата даних. Для запобігання цьому потрібно налаштовувати механізми резервного копіювання та реплікаці.[13]

2.4.2. Архітектура кешуючого сервісу

Архітектура кешуючого сервісу на основі Redis складається з кількох ключових компонентів, які забезпечують ефективне зберігання та доступ до даних у пам'яті.

Інтеграція з веб-застосунком, написаним на Node.js, є центральним елементом архітектури. Веб-застосунок взаємодіє з Redis для кешування часто використовуваних даних. Коли веб-застосунок отримує запит від користувача, він спочатку перевіряє наявність необхідних даних у кеші Redis. Якщо дані знайдені, вони негайно повертаються користувачеві, що забезпечує швидкий доступ. Якщо дані не знайдені у кеші, веб-застосунок запитує їх з основної бази даних PostgreSQL, а потім зберігає отримані дані у Redis для подальшого швидкого доступу.

Управління даними в кеші забезпечується різними механізмами, які дозволяють ефективно керувати обсягом збережених даних та їх актуальністю. Redis підтримує кілька політик витіснення даних, таких як LRU (LeastRecentlyUsed) та LFU (LeastFrequentlyUsed), що дозволяє автоматично видаляти найменш використовувані дані при досягненні ліміту пам'яті. Крім того, Redis надає можливість встановлення терміну життя (TTL) для кожного ключа, після закінчення якого дані автоматично видаляються з кешу.

Реплікація та кластеризація є важливими аспектами архітектури Redis, які забезпечують високу доступність та надійність даних. Основний сервер Redis може реплікувати дані на один або більше реплік, що забезпечує їх доступність навіть у випадку виходу з ладу основного сервера. Кластеризація дозволяє розподіляти дані між кількома вузлами, забезпечуючи масштабованість та розподіл навантаження. Це дозволяє ефективно обробляти великі обсяги даних та забезпечує відмовостійкість системи.

Архітектура Redis інтегрована з іншими компонентами веб-застосунку, такими як база даних PostgreSQL та сервери завдань. Веб-застосунок

використовує Redis для кешування результатів складних запитів до бази даних та обробки завдань у реальному часі. Це дозволяє значно підвищити продуктивність системи та знизити затримки у відповідях на запити користувачів. Завдяки асинхронній обробці запитів та підтримці різних структур даних, Redis стає потужним інструментом для створення високопродуктивних та масштабованих веб-застосунків.

2.5. Сховище даних на сонові технології AmazonS3

AmazonSimpleStorageService (S3) є одним із найпопулярніших рішень для зберігання великих обсягів даних у хмарі. S3 забезпечує надійне, масштабоване і доступне сховище для даних різного типу, включаючи документи, зображення, відео, журнали та резервні копії. У телекомунікаційних застосунках, де важливо зберігати великі обсяги даних та забезпечувати до них швидкий і надійний доступ, використання Amazon S3 є особливо ефективним.

S3 надає об'єктно-орієнтовану модель зберігання даних, де кожен файл (об'єкт) зберігається у бакеті та ідентифікується унікальним ключем. Це дозволяє легко керувати великими обсягами даних, забезпечувати їх захист і доступність, а також інтегрувати з іншими сервісами AmazonWebServices (AWS) для реалізації складних рішень.

2.5.1. Переваги та недоліки

Amazon S3 надає численні переваги, які роблять його привабливим рішенням для зберігання даних у телекомунікаційних застосунках. Однією з ключових переваг є масштабованість. S3 дозволяє зберігати необмежену кількість даних, автоматично масштабуючи ресурси для задоволення потреб користувачів. Це означає, що можна додавати нові об'єкти без будь-яких обмежень на обсяг сховища або кількість запитів.

Висока доступність є ще однією суттєвою перевагою S3. Дані зберігаються в декількох дата-центрах, що забезпечує надійність і захист від втрат даних у разі виходу з ладу окремих компонентів інфраструктури. Безпека даних також є сильною стороною S3. Підтримка шифрування даних як під час зберігання, так і

під час передачі, разом з політиками IAM (Identityand Access Management) для управління доступом, дозволяє забезпечити високий рівень захисту даних.

Інтеграція з іншими сервісами AWS є важливою перевагою S3. Це забезпечує можливість легко поєднувати S3 з іншими інструментами, такими як AmazonCloudFront (CDN), AWS Lambda, AmazonRedshift та інші, що дозволяє створювати складні та масштабовані рішення. Відновлення даних у S3 також є дуже зручним завдяки підтримці версійності об'єктів. Це дозволяє зберігати кілька версій одного файлу та відновлювати попередні версії у разі необхідності. Крім того, S3 пропонує різні класи зберігання, такі як Glacier для архівного зберігання даних за низькою ціною.

Проте, S3 має і деякі недоліки. Одним з них є затримки доступу. Незважаючи на високу швидкість доступу до даних, S3 може мати затримки у порівнянні з локальними сховищами, особливо для малих файлів або при великій кількості запитів на читання. Вартість зберігання та передачі даних у S3 також може бути високою, особливо при зберіганні великих обсягів даних або при частих запитах на доступ, що вимагає ретельного планування та оптимізації витрат.

Крім того, управління великими обсягами даних у S3 може бути складним завданням. Незважаючи на зручність використання, ефективне управління вимагає додаткових інструментів та автоматизації для забезпечення оптимальної роботи та безпеки даних.

2.5.2. Архітектура сховища

Архітектура сховища даних на основі Amazon S3 складається з кількох ключових компонентів, які забезпечують ефективне зберігання, управління та доступ до даних. Веб-застосунок, написаний на Node.js, взаємодіє з S3 для зберігання та отримання даних. Веб-застосунок використовує SDK AWS для Node.js для виконання операцій з S3, таких як завантаження файлів, отримання файлів та управління метаданими. При завантаженні файлу користувачем веб-застосунок приймає файл, зберігає його тимчасово на сервері, а потім завантажує

його у S3. Кожен об'єкт у S3 ідентифікується унікальним ключем, що дозволяє легко організувати доступ до даних.

Для забезпечення безпеки даних у S3 використовуються політики IAM для управління доступом, а також шифрування даних як під час зберігання, так і під час передачі. Кожен об'єкт може бути зашифрований за допомогою ключів, керованих AWS (SSE-S3) або користувацьких ключів (SSE-C). Для забезпечення відповідності даних корпоративним політикам та нормативним вимогам використовуються інструменти аудиту та моніторингу, такі як AWS CloudTrail, які записують всі дії, пов'язані з доступом до об'єктів у S3. Це дозволяє відстежувати та аналізувати всі операції, виконані над даними.

Amazon S3 автоматично реплікує дані між кількома дата-центрами в межах одного регіону, забезпечуючи високу доступність та захист від втрат даних. У разі виходу з ладу одного з дата-центрів дані залишаються доступними з інших дата-центрів. Крім того, S3 підтримує різні класи зберігання, які дозволяють оптимізувати витрати та забезпечити необхідний рівень доступності. Наприклад, S3 Standard забезпечує високу доступність для часто доступних даних, тоді як S3 Glacier використовується для архівного зберігання даних, до яких рідко звертаються, за низькою ціною.

2.6. Система резервного копіювання

Система резервного копіювання є критично важливою складовою будь-якого телекомунікаційного застосунку, оскільки вона забезпечує збереження даних у випадку апаратних збоїв, помилок програмного забезпечення або інших непередбачуваних подій. AmazonSimpleStorageService (S3) є ідеальним рішенням для реалізації системи резервного копіювання завдяки своїм характеристикам надійності, масштабованості та безпеки.

2.6.1. Переваги та недоліки

Amazon S3 пропонує безліч переваг для реалізації системи резервного копіювання. Однією з головних переваг є висока надійність зберігання даних. S3 забезпечує автоматичну реплікацію даних між кількома дата-центрами в межах одного регіону, що мінімізує ризик втрати даних у випадку збоїв. Це дозволяє

користувачам бути впевненими у безпеці своїх даних навіть у разі виникнення критичних ситуацій.

Іншою важливою перевагою є масштабованість. S3 дозволяє зберігати необмежену кількість даних, що особливо важливо для телекомунікаційних застосунків, де обсяги даних можуть швидко зростати. Автоматичне масштабування S3 забезпечує безперервне розширення сховища відповідно до потреб користувачів, без необхідності втручання адміністратора.

Безпека даних також є однією з основних переваг S3. Підтримка шифрування даних як під час зберігання, так і під час передачі, разом з політиками IAM (Identityand Access Management) для управління доступом, дозволяє забезпечити високий рівень захисту даних. Ці інструменти забезпечують надійний захист від несанкціонованого доступу та забезпечують відповідність нормативним вимогам.

Однак, S3 має і деякі недоліки. Одним з них є вартість. Хоча S3 пропонує різні класи зберігання з різними ціновими категоріями, вартість зберігання та передачі даних може бути значною, особливо при великих обсягах даних. Це вимагає ретельного планування та оптимізації витрат. Крім того, затримки доступу до даних можуть бути проблемою у порівнянні з локальними сховищами, особливо для малих файлів або при великій кількості запитів на читання.

2.6.2. Архітектура сервісу резервного копіювання

Архітектура сервісу резервного копіювання на основі Amazon S3 включає кілька ключових компонентів, які забезпечують ефективне зберігання, управління та відновлення даних.

Першим компонентом є веб-застосунок, який відповідає за створення резервних копій даних та їх зберігання у S3. Веб-застосунок, написаний на Node.js, використовує SDK AWS для Node.js для виконання операцій з S3, таких як завантаження резервних копій, отримання файлів та управління метаданими. Коли веб-застосунок створює резервну копію, він зберігає її тимчасово на сервері, а потім завантажує у S3. Кожен об'єкт у S3 ідентифікується унікальним ключем, що дозволяє легко організувати доступ до резервних копій.

Для забезпечення безпеки даних у S3 використовуються політики IAM для управління доступом, а також шифрування даних як під час зберігання, так і під час передачі. Кожен об'єкт може бути зашифрований за допомогою ключів, керованих AWS (SSE-S3) або користувацьких ключів (SSE-C). Це забезпечує захист даних від несанкціонованого доступу та відповідає вимогам безпеки.

Однією з ключових особливостей S3 є можливість налаштування різних класів зберігання, що дозволяє оптимізувати витрати на зберігання резервних копій. Наприклад, для часто використовуваних даних можна використовувати клас S3 Standard, тоді як для архівного зберігання даних, до яких рідко звертаються, можна використовувати S3 Glacier, що дозволяє знизити витрати. Це особливо важливо для телекомунікаційних застосунків, де обсяги резервних копій можуть бути дуже великими.

Процес відновлення даних з резервних копій у S3 є простим і ефективним. Веб-застосунок може отримати резервну копію з S3, використовуючи унікальний ключ об'єкта, і відновити дані на сервері. Це забезпечує швидке відновлення даних у випадку збоїв або втрат, мінімізуючи простої в роботі системи.

3. ОПИС РОЗРОБЛЕНОГО ЗАСТОСУНКУ

Розроблений застосунок базується на вищеописаній архітектурі. Для розв’язання поставлених задач, був використаний наступний інструментарій:

- NodeJSy якості серверної частини застосунку (веб-застосунок);
- Python + GRPCдля розробки сервісу мап та сервісу роботи з ЕЦП;
- PostgreSQL + PostGISдля реалізації бази даних з можливістю зберігання великої кількості геопросторових даних;
- AmazonS3 для реалізації сховища даних та резервного копіювання системи;
- Redisдля реалізації системи кешування.

Також система була повністю забезпечена автоматичними тестами для перевірки працездатності роботи систем. Тести були поділені на юніт-тести та інтеграційні тести, бо система включає у себе інтеграції із зовнішніми системами. Дана процедура була виконана з міркувань забезпечення стабільності роботи додатку.

Для забезпечення легкого відслідковування кіберінциндентів, біло забезпечено систему логування, яка використовує у якості структури логу структуру логівELK, для можливості подальшого впровадження даного стеку технологій.

3.1. Сервіс веб-мап

Сервіс веб мап розроблений з метою забезпечення системи можливістю відображення картографічної інформації. Даний сервіс включає у себе наступні компоненти:

- Сервіс тайлінгу веб мап.
- Сервіс відображення векторних даних у форматі WFS, WMS, TMS.
- Систему налаштування веб мап.
- Компонент завантаження георесурсів.

Даний сервіс розроблений мовою програмування Pythonта використовує систему виклику віддалених процедурGRPC, а також використовується мова програмування NodeJS.

Сервіс виконує наступні операції:

- Тайлінг веб мап.
- Створення файлів конфігурації веб мап.
- Виконання операцій над об’єктами файлового сховища.
- Створення копій мап для друку.
- Перетворення геоінформаційних даних між собою.

3.1.1. Опис операцій, які реалізовані

Основною операцією, яку повинен виконувати даний сервіс – це надання можливості відображення геопросторових даних на картографічній підкладці. Дану задачу було вирішено за допомогою використання бібліотеки Mapnik, яка дозволяє, за наперед заданою конфігурацією, формувати зображення фрагменту мапи, спираючись на географічну позицію та наближення мапи. У якості конфігураційного файлу, бібліотека використовує файл xmlпевної структури, яка наведена у Додатку А. Дана структура включає у себе наступні параметри:

- Map**:** Кореневий елемент, який містить всі інші елементи. Може мати атрибути, такі як bgcolor (колір фону) і srs (система координат).
- Style: Визначає стиль для шарів. В кожному стилі можна мати кілька правил (Rules), які визначають, як рендерити дані (полігони, лінії, текст тощо).
- Rule: Містить символізатори, які визначають, як рендерити дані в межах правил.
- PolygonSymbolizer: Визначає символізацію полігонів. Наприклад, колір заливки (fill).
- LineSymbolizer: Визначає символізацію ліній. Наприклад, колір лінії (stroke) і товщина лінії (stroke-width).
- Layer: Визначає шар даних, який буде рендеритися. Містить атрибути, такі як name (ім'я шару) і srs (система координат шару).

- Datasource: Визначає джерело даних для шару. Містить параметри, такі як type (тип джерела, наприклад, shape) і file (шлях до файлу даних). Також підтримуються запити до баз даних, для відображення динамічних даних.

Для формування таких конфігураційних файлів був розроблений предпроцесор на NodeJS. Його функція полягає у тому, щоб приймати набір параметрів, які задають набір даних і перетворювати його у конфігураційний файл. На вхід нього передаються наступні параметри:

- Опис стилі для шарів
- Опис даних:

1. Підключення до БД
2. Фізичне розташування файлу
3. Підключення да зовнішнього ресурсу

На основі цих даних, предпроцесор формує текст конфігураційного файлу. Для цього використовується npmпакет @mapbox/carto-generator, який формує тест файлу CartoCSS. Далі цей файл потрібно завантажити до файлового сховища сервісу формування зображення мап. Для цього використовуються фукції, які реалізують операції над об’єктами файлового сховища. Приклад цих функцій наведено у Додатку Б.

Завантаження файлу на сервіс відбувається за рахунок функції uploadFile. Дана функція приймає файл шматками від клієнта, який створений мовою програмування NodeJS. Також, у я кості параметру, передається шлях, куди треба зберегти файл. Також можна передати відносний шлях до файлового сховищаортофотознімків. Під цей вид растрових зображень мап виділено окреме сховище, через великий розмір таких файлів і необхідність створення стиснутих копій, для відображення растрової інформації на малих наближеннях мапи.

Також реалізовані функції перегляду файлів (checkFiles, readDir), функція видалення (deleteFile) та функція вивантаження файлу (downloadFile).

Через те, що сервіс тайлінгу, повинен підтримувати відображення веб мап на усіх можливих наближеннях мап, для спрощення даного процесу, було розроблено можливість стиснення растрових зображень. Векторні зображення

стискати немає сенсу, бо вони зберігаються у форматі svgy одному екземплярі. Для роботи з растровими даними, було використано пакет gdal-cli, який надає набір консольних команд для роботи з растровою інформацією. Даний пакет розроблений мовою програмування C++ та є відкритим. Для виконання операції стиснення, використовується команда gdal_translate. У сервісі розроблено функцію-предпроцесор, яка формує консольну команду і виконує її. Дана функція наведена у Додатку В. Вона приймає у себе назву команди та параметри виклику. Окремо передаються параметри вхідного шляху та вихідного. Після виконання опереації стиснення, якщо ми маємо справу з набором растрових зображень, які покривають велику територію, використовується команда gdalvrt. ЇЇпризначення полягає у формуванні конфігурації мапи з описом положення кожного растрового зображення на мапі. Для цього, обов'язковою вимого є використання растрових зображень, які використовуються у геоінформатиці, що гарантує їм геопросторовуприв'язку. У результаті отримаємо файл формату VRT, який виступає у якості джерела даних для конфігураційного файлу мапи.

Для реалізації операції тайлінгу, реалізовано функцію render, яка формує зображення мапи для заданих параметрів наближення та географічних координат, які обмежують зображення. Дана функція наведена у Додатку Г.

Сервіс використовує систему віддаленого виклику процедур GRPC. Дана система забезпечує можливість структурувати запити та визначати типи даних для кожного параметру запиту. Це досягається за рахунок використання прототипування сервісу. Для задання структури запиту та типів даних параметрів використовується мова опису інтерфейсів Protobuf. Вона дозволяє задавати не тільки типи даних, а й окремі структури, які можуть виступати у якості параметрів запиту. Приклад конфігурації для сервісу наведено у Додатку Ґ.

3.2. Сервіс аутентифікації користувача на основі ЕЦП

Для аутентифікації у адмін частині застосунку, було використаносервіс роботи з ЕЦП. Даний сервіст також реалізовано на основі мови програмування Pythonта технології GRPC. Також було реалізовано клієнт мовою програмування NodeJS. З міркувань безпеки та комерційної таємниці, код реалізації даного

функціоналу у розділі не наводиться, але надалі буде наведено загальний опис підходів та методів реалізації даного сервісу.

Для роботи з ЕЦП було використано бібліотеку, розроблену для мови програмування Python, Інститутом Інформаційних Технологій. Дана бібліотека реалізовує функціонал зчитування інформації з електронного підпису, дешифрувати дані отриманні від віджету роботи з електронним підписом, який був розроблений ДП «ДІЯ», та підпису файлів за допомогою ЕЦП.

Для отримання доступу до адміністративної частини застосунку, користувачу потрібно спочатку пройти процедуру надання даних про свою особу за допомогою віджета розробленого ДП «ДІЯ». Даний віджет наразі використовується на державних сайтах для аунтентифікації особи. Після надання цих даних, віджет надсилає дані на веб-додаток, який, через клієнт, передає ці дані у зашифрованому вигляді на сервіс. Сервіс, у свою чергу, розшифровує ці дані та передає їх назад на веб-додаток, для проходження перевірки на наявність у користувача прав доступу.

Для реалізації усіх вищеописаних операцій потрібно мати також ключ шифрування. Дані ключі видаються на скінченний термін часу ліцензованими закладами. Дані ключі можуть зберігатися як локально, та і за допомогою спеціалізованого обладнання. Ключі, які зберігаються локально, повинні бути класифіковані у системі. Ця класифікація виконується за рахунок заведення шляхів до ключів у конфігураційному файлі веб-додатку. Потім, відповідно до налаштувань додатку, на сервіс передається шлях до цього ключа, як параметр запиту.

Ключі також можуть зберігатись на спеціальном пристрої під назвою «ГРЯДА-301». Даний пристрій надає можливість звертатись до нього за допомогою бібліотеки, та за допомогою задання позиції ключа, надавати його бітовий код для сервісу. Даний код потім дешифрується і використовується для роботи з ЕЦП.

3.3. База даних

Додаток повинен мати можливість зберігати інформацію про наступний набір об’єктів дорожньо-транспортної мережі:

- Дорожні знаки.
- Розмітка.
- Розтяжки.
- Стовпи для дорожніх знаків
- Світлофори
- Світлофорне обладнання
- Камери спостереження
- Дороги

Кожен з наведених класів містить у собі ще деяку кількість підкласів. Наприклад: дорожні знаки – це сам знак та опора на якій він розміщується. Кожен з таких композитних об’єктів містить свій інвентарний номер. Також, такі номери мають і об’єкти з яких він складається. Це зроблено для того, щоб мати можливість окремо класифікувати стан кожної складової об’єкту. Приклад реалізації такого класу наведено у Додатку Д. З метою забезпечення комерційної таємниці, повну схему БД не буде наведено у даній роботі.

Також до об’єктів дорожньо-транспортної мережі є ДТП. Ці об’єкти зберігаються в окремій схемі у БД. Також дані об’єкти класифікуються за станом ДТП та станом постраждалих. На основі цих даних формуються теплові мапи та мапи місць концентрації. Для побудови місць концентрації використовується функція просторової агрегації об’єктів, яка надається модулем PostGIS. Робота з даними об’єктами була висвітлена у публікації 2019 року на конференції 2019 IEEE InternationalConferenceonAdvancedTrendsinInformationTheory.

Кожен з наведених вище об’єктів має власну геопросторовуприв’язку. Це потрібно для подальшого їх відображення у системі на картографічній основі, а також для проведення аналізу стану дорожньо-транспортної мережі у певному районі. Також, під час експлуатації системи, було виявлено, що кількість об’єктів

на територію міста Київ починає перевищувати 100 тисяч. Це призводить до того, що геопросторовий аналіз уповільнюється, а також загалом уповільнюється система. Для вирішення цієї проблеми, усі геопросторові об’єкти були проіндексовані, що підвищило продуктивність системи на 60%, у порівняні з попереднім станом.

Для забезпечення доступності даних, була налаштована реплікація. На клієнті веб-додатку було розроблено конфігураційний файл, який містить у собі підключення до БД. Дані реплікації знаходяться на різних серверах, тому для кожної параметри прописуються окремо. При втраті зв’язку з одним сервером БД, клієнт автоматично починає використовувати наступний. Реплікація відбувається асинхронним методом і відпрацьовує кожні годину, для забезпечення достовірності інформації у реплікованих копіях. Також реалізовано процес перестворення конфігураційних файлів для веб-мап, при втраті зв’язку.

3.4. Веб-додаток

Веб-додаток виконує роль клієнту, для інших сервісів. Він розроблений мовою програмування NodeJS. Усі розроблені АПІ для забезпечення роботи додатку використовують структуру REST. Даний сервіс призначений для виконання наступних операцій:

- Виконання запитів до БД
- Виконання запитів до сервісу веб-мап
- Отримання інформації від клієнта та обробка його запитів
- Кешування складних аналітичних запитів
- Робота зі сховищем S3

Також додаток виконує функцію відображення інформації на веб сторінці динамічно. Для цього використовується технологіє handlebars та предпроцесор. Задача предпроцесора отримати на вхід параметри відображення інформації та параметри фільтрації інформації і на основі них сформувати відповідну сторінку користувачу. У свою чергу, handlebars — це шаблонізатор для JavaScript, який дозволяє ефективно розділяти логіку програми від її представлення (інтерфейсу користувача). Ось основні випадки використання Handlebars:

- Шаблонізація HTML: Handlebars дозволяє створювати HTML-шаблони, які можуть бути заповнені динамічними даними. Це робить створення HTML-коду більш організованим і легко підтримуваним.
- Відокремлення логіки від представлення: Замість того, щоб мати логіку і HTML змішаними в одному файлі, ви можете зберігати шаблони окремо і використовувати їх для рендерингу інтерфейсу користувача.
- Легке оновлення інтерфейсу: Оновлення інтерфейсу користувача стає простішим і швидшим, оскільки ви можете змінювати дані, що передаються в шаблон, і Handlebars автоматично оновить відповідний HTML.
- Управління складними структурами даних: Handlebars підтримує вкладені шаблони, що дозволяє легко працювати зі складними структурами даних, такими як списки або об'єкти.
- Розширення можливостей шаблонів: Handlebars підтримує користувацькі хелпери (функції), які можуть бути використані для виконання складних операцій в межах шаблону.

Так як у системі використовується достатньо складне представлення об’єктів дорожньо-транспортної мережі, дана технологія надає переваги у швидкодії. Також, це надає можливість для створення більш складного предпроцесору. Його задачею є формування сторінок для відображення інформації на основі певного конфігураційного файлу. Приклад такого файлу наведено у Додатку Е. Предпроцесор приймає у себе даний файл і формує відповідну сторінку з наповненням. Це дозволяє забезпечити гнучкість додатку та можливість легкого масштабування. З метою забезпечення комерційної таємниці, код даного предпроцесору не буде наведений у дані роботі.

Також, потрібно згадати, що даний сервіс повинен бути захищенний від кіберінциндентів. Одним з інструментів попередеження їх є система логування, яка запровадженна на кожному з окремих компонентів додатка. Додатково, у системі є можливість повнотекстового пошуку та фільтрації інформації на основі певних параметрів. Такі можливості одразу призводять до виникнення XSSта SQLін’єкцій. Для запобігання їх, було розроблено мідлвейр, який перевіряє усі

параметри, які приходять від форм заповнення даних, на можливість наявності сигнатури ін’єкції. Також, для запобігання виконання шкідливих скриптів, було обмежено джерела звідки скрипти можуть бути виконані за допомогою налаштування CSP.

Реалізація клієнта для сервісу веб мап базується на основі такого самого файлу protobuf, як і сам сервіс. Це потрібно для забезпечення правильності передачі та розпізнавання даних. Для побудови клієнту було використано npmпакет @grpc. Запропонований пакет надає можливість легкого налаштування клієнту, а також надає інструментарій для побудови запитів та відслідковування помилок. Параметри доступу до сервіса записується окремо у конфігураційному файлі і потім на основі них будується зв’язок. Приклад реалізації клієнта наведено у Додатку Є.

Також, деякі з АПІ повинні бути надані у публічний доступ. Це було однією з вимог до додатку, під час початку його розробки. Але, через характер даних, які містяться у системі, було вирішено забезпечити це за допомогою реалізації схеми авторизації через OAuth 2.0. За даною схемо, користувачу надаються параметри генереації ключа входу у систему, а він, у свою чергу, додає отриманий ідентифікаційний код у якості параметру запиту. У якості технології створення ключа було обрано JWT. Даний тип токенів (ключів) надає можливість однозначно визначити тип користувача, його права, а також запроваджує механізм перевірки дійсності ключа на основі підпису.

3.5. Сховище даних

Розробка сховища даних є важливою частиною будь-якого сучасного веб-застосунку, що дозволяє зберігати та керувати великими обсягами даних. AmazonSimpleStorageService (S3) разом з Node.js забезпечують потужну та гнучку платформу для створення масштабованих, надійних та безпечних рішень для зберігання даних. У цьому розділі буде розглянуто процес створення сховища даних за допомогою Amazon S3 та Node.js, включаючи налаштування середовища, інтеграцію з S3 та реалізацію основних операцій з даними.

Першочергово, при виборі сховища було взято до уваги можливість структуювання збереженої інформації. S3 надає таку можливість, бо є по-факту віддаленим файловим сховищем з власним провідником, який реалізований шляхом надання доступу до АПІ. Також, s3 надає можливість створення бакетів, що є аналогом окремого жорсткого диски для зберігання інформації.

Для реалізації сховища даних для даного проекту, було створено окремий бакет. Кожен файл, який потрапляє у систему, класифікується нею. На основі проведеної класифікації, файлу задається шлях. Після отримання шляху, файл передається на клієнт s3. Також, до цього файлового сховища потрапляють необхідні резервні копії системи, для забезпечення швидкого розгортання аткуальної версії системи, якщо вона дала збій.

Для реалізації зв’язку між веб-додатком та файловим сховищем, було розроблено клієнт мовою програмування NodeJS. Код даного клієнта наведено у Додатку Ж. Клієнт реалізує набір стандартних функцій роботи з файловою системою. Параметри підключення до сховища зберігаються у окремому конфігураційному файлі і передаються на сервіс під час старту системи. Також, кожна з цих функцій додатково захищенна політиками безпеки AWS IAM для забезпечення стабільності за безпеки роботи сховища даних. Налаштування даних політик відбувається шляхом передачі додаткових параметрів у запит до s3. Вони задаються за допомогою об’єкта, який має наступні параметри:

- «Sid» або ідентифікатор декларації є необов'язковим полем, яке служить для унікальної ідентифікації декларації у межах політики. Це корисно для зручного управління та посилання на конкретні декларації у політиці.
- «Effect» визначає дію, яка буде застосована при виконанні декларації. Можливі значення:

1. «Allow»: дозволяє виконання зазначених дій.
2. «Deny»: забороняє виконання зазначених дій.

- «Principal» вказує на суб'єкт (користувач, група, роль або сервіс), до якого застосовується політика. Може бути конкретний ARN

(AmazonResourceName) або символічне значення «*», що означає всіх користувачів.

- «Action» визначає одну або більше дій, які дозволені або заборонені для зазначених ресурсів. Наприклад, s3:GetObject дозволяє отримувати об'єкти з бакету, s3:PutObject дозволяє завантажувати об'єкти у бакет.
- «Resource» вказує на ресурс або ресурси, до яких застосовується декларація. Це може бути конкретний об'єкт або весь бакет. Вказується у форматі ARN. Для бакетів S3 це може виглядати так: arn:aws:s3:::your-bucket-name/* для всіх об'єктів у бакеті або arn:aws:s3:::your-bucket-name/your-object-key для конкретного об'єкта.
- «Condition» є додатковим елементом, який дозволяє встановлювати умови, за яких декларація буде застосовуватись. Наприклад, можна обмежити доступ за часом, IP-адресою або використовувати інші атрибути.

3.6. Сервіс кешування

Кешування є важливою складовою частиною сучасних веб-застосунків, що дозволяє підвищити продуктивність та знизити навантаження на основні бази даних. Redis, як високопродуктивна база даних типу ключ-значення, є ідеальним інструментом для реалізації кешування даних. Використання Redis разом з Node.js забезпечує ефективне зберігання та доступ до часто використовуваних даних у пам'яті, що значно прискорює обробку запитів користувачів. У цьому розділі буде розглянуто процес створення кешування даних за допомогою Redis та Node.js, включаючи налаштування середовища, інтеграцію з Redis та реалізацію основних операцій з кешем. Приклад реалізації клієнта для redisнаведено у Додатку З.

У додатку реалізовано декілька різних баз redisдля кешування різних типів інформації. Наприклад: системна інформація кешується у базі 0, а інформація зі складних аналітичних запитів кешується у баз 5. Під ці бази виділено свою частину вільного місця, на фізичному носії. Так як redisбудує кеш на сонові схеми «ключ-дані», то у системі повинен бути спосіб створення однозначного ключа визначення даних. Для цього, було обрано набір параметрів, на основі яких і буде будуватись ключ. До них відносяться:

- Дата та час надходження запиту.
- Унікальний ідентифікатор користувача від якого надійшов запит.

На основі цих двох параметрів можна однозначно визначити набір даних, які потрібно передати користувачу з кеша, як результат його запиту.

3.7. Розгортання застосунку

Для впровадження даного застосунку, його потрібно розгорнути на сервері. Для того, щоб кожна компонента застосунку працювала коректно, було обрано використання технології Dockerдля розгортання застосунку.Docker є популярною технологією контейнеризації, яка дозволяє розробникам упаковувати застосунки та їх залежності в ізольовані середовища, відомі як контейнери. Використання Docker спрощує процес розгортання та управління веб-застосунками, забезпечуючи консистентність середовища на всіх етапах розробки, тестування та виробництва. У цьому розділі буде детально розглянуто використання Docker для розгортання веб-застосунків на сервері, включаючи налаштування середовища, створення Dockerfile, побудову та запуск контейнерів, а також управління контейнерами у виробничому середовищі.

Для кожного компоненту було сформовано Dockerfileдля налаштування створення контейнеру. У кожному з цих фалів окремо прописуються команди для запуску сервісу, встановлення усього необхідного, а також вбудовано механізм перевірки версії на основі геш суми, яка потім порівнюєтьс з геш сумою, яка зберігається у директорії .git. Ця можливість з’явилась за рахунок використання GITтехнологій під час етапу розробки.

У якості операційної системи для кожного контейнера, було обрано останній стабільний реліз операційної системи Ubuntu. Виключенням є сервіс роботи з ЕЦП та сервіс веб-мап. Ці сервіси працюють на специфічних збірках контейнерів. Для сервісу веб мап було обрано контейнер gdal, який вже містить повністю налаштовано необхідну для роботи сервісу бібліотеку. Для сервісу роботи з цифровим підписом, було обрано контейнер python3.9, який базується на основі операційної системи Debian. Цей вибір пов’язаний зі зменшенням розміру контейнера шляхом прибирання зайвого функціоналу.

Викачка необхідних пакетів для роботи сервісів виконується або за наперед заданим списком, або набір пакетів береться з конфігураційного файлу, або і те і інше одночасно. Підхід з конфігураційними файлами використовується під час створення контейнерів для веб-застосунку, який викачує npmпакети, які потрібні для роботи NodeJS, а також у контейнерах сервісів, які написані на Python. У сервісах розроблених на Pythonвикористовується спеціальний файл requirements.txt, який містить опис пакетів та їх версій. У NodeJSцей файл має назву package.jsoni формується автоматично, під час будь-якого нового завантаження пакету.

Дані контейнери збираються на окремому сервері і далі пакуються у архів. Даний процес повністю автоматизований за рахунок використання технології pipeline. Кожен контейнер автоматично збирається при створенні нової версії продукту у гіті. Перед створенням, pipelineперевіряє наявність версії у сховищі і, якщо не знайшов таку – створює нову. Якщо ж версія була вже знайдена, скрипт автоматично видасть помилку наявності такої версії. Після створення контейнерів, вони архівуються і складаються у файловому сховищі CDN. Для запуску даних контейнерів використовується консольна команда Docker, яка надає можливість завантажувати, розпаковувати та запускати контейнери з віддаленого сховища.

ВИСНОВКИ

Результати проведених досліджень та розробок у рамках дипломної роботи підтвердили ефективність використання сучасних телекомунікаційних технологій для вирішення складних аналітичних задач у сфері дорожньо-транспортних мереж. У роботі були розглянуті принципи побудови телекомунікаційних застосунків, що дозволяють ефективно збирати, накопичувати та аналізувати дані про стан дорожньої інфраструктури.

Однією з головних переваг запропонованої архітектури є її модульність та масштабованість, що дозволяє легко адаптувати систему до змінних умов та вимог. Використання мікросервісної архітектури забезпечує високу гнучкість і надійність системи, дозволяючи незалежно розробляти, розгортати та масштабувати кожен сервіс. Це значно знижує складність управління системою та підвищує її стійкість до збоїв.

Запропонована система використовує геоінформаційні технології (GIS) для збору, зберігання, аналізу та візуалізації просторових даних. Інтеграція GIS з іншими компонентами системи дозволяє забезпечити всебічний моніторинг та управління дорожньо-транспортною мережею. Це сприяє підвищенню точності та оперативності прийняття рішень щодо управління інфраструктурою та оптимізації транспортних потоків.

Таким чином, проведені дослідження та розробки підтверджують доцільність використання сучасних телекомунікаційних технологій для управління дорожньо-транспортними мережами. Запропонована архітектура забезпечує високу ефективність, надійність та безпеку системи, що дозволяє ефективно вирішувати складні аналітичні задачі та забезпечувати стабільне функціонування дорожньо-транспортної інфраструктури. Подальший розвиток та впровадження таких технологій сприятимуть підвищенню безпеки та якості транспортних послуг, а також оптимізації управління транспортними потоками.

У даній роботів використані результати моїх попередніх розробок. [14, 15]

Компоненти даного застосунку розроблені та впровадженні на базі державних додатків Єдиного Державного Реєстру у сфері будувництва, Єдиного Державного Реєстру Адрес та додатку містобудівного-кадастру державного рівня.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. MishraCloudComputingandIoTforSmartTransportation/ Mishra V., Manish K. Y. / InternationalJournalofVehicleStructuresandSystems. –December 2022. – Vol 21. № 10. - P. 4097–4115

2. Gammaetal.DesignPatterns: ElementsofReusableObject-OrientedSoftware / Gamma, E., Helm, R., Johnson, R., &Vlissides, J. / Addison-Wesley - 1995

3. ZhangReal-timeTrafficPredictionwithMachineLearning / Zhang, H. / TransportationResearchPart C: Emerging Technologies – 2017 – P 86, 64-85

4. SommervilleSoftwareEngineering (10th ed.) / Sommerville, I. / Pearson - 2015

5. Martínez-Ramón&MartinMachineLearningTechniquesforTransportationNetworkAnalysis / Martínez-Ramón, M., &Martin, D. / JournalofBigData. - 2018 – Vol 5. № 1. P 14-28

6. Yang&GidlundGIS Technologies forTrafficFlowMonitoringandAnalysis / Yang, H., &Gidlund, M. / InternationalJournalofGeo-Information. – 2021 – Vol 10. № 2. P 89-104

7. Chen&XiangIntroductiontoDistributedFileSystems: HDFS and GFS. / Chen, J., &Xiang, Z. / JournalofCloudComputing – Vol 5, №1 – P 25-35

8. AntoniouDemandForecastinginTransportation / Antoniou, C. / Springer - 2016

9. AmazonWebServices. URL:https://aws.amazon.com/s3/ (дата звернення 09.03.2024)

10. Obe&HsuPostGISinAction (3rd ed.) / Obe, R., &Hsu, L. / ManningPublications. - 2021

11. Hwang, Fox, &Dongarra / DistributedandCloudComputing: FromParallelProcessingtotheInternetofThings / Hwang, K., Fox, G., &Dongarra, J. / MorganKaufmann - 2012

12. SanfilippoRedis: A HighPerformanceIn-MemoryDataStructureStore. URL: https://redis.io/ *(дата звернення 08.03.2024)*

13. Carlson / RedisinAction / Carlson, T. / ManningPublications– 2013

14. D. Verenych, M. Kononov The system for the accumulation and spatial-temporal analysis of accident / D. Verenych, M. Kononov / The 22th Annual International Young Scientists Conference on Applied Physics – May 2022

15. D. Verenych and etc. GIS-TechnologiesUsingforSpatialDataAnalyseofthe RoadTrafficAccidencesontheExampleofKyiv/ D. Verenych, O. Verenych, I. Vasiliev, S. Bezshapkin / IEEE Xplore – December 2019

ДОДАТКИ

ДОДАТОК А. ПРИКЛАД ФАЙЛУ КОНФІГУРАЦІЇ МАПИ

```
<?xmlversion="1.0" encoding="utf-8"?>
<!DOCTYPE Map [
<!ENTITY entities SYSTEM "entities.xml">
]>
<Mapbgcolor="#ffffff" srs="+proj=merc +lon_0=0 +k=1 +x_0=0 +y_0=0
+datum=WGS84 +units=m +no_defs">

<!-- Визначення стилів -->
<Stylename="example_style">
<Rule>
<PolygonSymbolizerfill="#f2eff9" />
<LineSymbolizerstroke="#ccc" stroke-width="0.5" />
</Rule>
</Style>

<!-- Визначення шарів -->
<Layername="example_layer" srs="+proj=longlat +datum=WGS84 +no_defs">
<StyleName>example_style</StyleName>
<Datasource>
<Parametername="type">shape</Parameter>
<Parametername="file">data/example.shp</Parameter>
</Datasource>
</Layer>

</Map>
```

ДОДАТОК Б. КОД ФУНКЦІЙ ФАЙЛОВОГО СХОВИЩА СЕРВІСУ ВЕБ-МАП

```
defuploadFile(self, request, context):
start_time = time.time()

output_filepath = request.relativeFilepath
data = request.fileBytes
ifnotos.path.exists(os.path.dirname(output_filepath)):
os.makedirs(os.path.dirname(output_filepath))
ifnotos.path.exists(f'{os.path.dirname(output_filepath)}/preview'):
os.makedirs(f'{os.path.dirname(output_filepath)}/preview')
try:
ifoutput_filepath:
withopen(output_filepath, 'ab') as f:
f.write(data)
message = {
                'funcs': 'uploadFile',
                'info': {'status': 'OK', 'time': time.time() - start_time},
            }
            logging.info(json.dumps(message))
return map_pb2.fileManagerOut(status=True, operation='upload')
exceptExceptionas e:
message = {
                'funcs': 'uploadFile',
                'error': repr(e).replace('\n', '; ')
            }
logging.error(json.dumps(message))
raise


defdeleteFiles(self, request, context):

importshutil

success = []
error = []
try:
forobjectinrequest.objectList:
rel_path = f'{request.dir}/{object}'
rel_preview_path = f'{request.dir}/preview/{object}'
ifos.path.isfile(rel_path) oros.path.islink(rel_path):
try:
```

```
      os.remove(rel_path)
      success.append(map_pb2.deleteObjSuccess(object=object, type='file',
status='delete'))
      exceptExceptionas e:
      success.append(map_pb2.deleteObjError(object=object, fullPath=rel_path,
      result=repr(e).replace('\n', '; ')))
      try:
      os.remove(rel_preview_path)
      except:
      pass
      elifos.path.isdir(rel_path):
      try:
      shutil.rmtree(rel_path)
      success.append(map_pb2.deleteObjSuccess(object=object, type='dir',
status='delete'))
      exceptExceptionas e:
      success.append(map_pb2.deleteObjError(object=object, fullPath=rel_path,
      result=repr(e).replace('\n', '; ')))
      if '/data/ortho/files/manager' inrel_path:
      if os.path.isfile(f'/data/ortho/files/manager/{request.dir}/mosaic.vrt')
andnotobject == 'map.xml':
      os.remove(f'/data/ortho/files/manager/{request.dir}/mosaic.vrt')
      if os.path.isfile(f'/data/ortho/files/manager/{request.dir}/preview.vrt')
andnotobject == 'map.xml':

os.remove(f'/data/ortho/files/manager/{request.dir}/preview.vrt')
      if os.path.isfile(f'/data/ortho/files/manager/{request.dir}/map.xml')
andnotobject == 'map.xml':
      os.remove(f'/data/ortho/files/manager/{request.dir}/map.xml')
      return map_pb2.deleteOut(success=success, error=error)
      exceptExceptionas e:
      message = {
                      'funcs': 'checkFiles',
                      'error': repr(e).replace('\n', '; ')
                  }
      logging.error(json.dumps(message))
      raise

      defdownloadFile(self, request, context):

      chunk_size = 1024
```

```
        ifos.path.exists(request.filePath):
        withopen(request.filePath, mode="rb") as f:
        whileTrue:
        chunk = f.read(chunk_size)
        ifchunk:
        yield map_pb2.downloadFileOut(chunk_data=chunk)
        else:
        return


        defcheckFiles(self, request, context):
        try:
        dir = request.dir
        fileSearch = request.filesSearch
        ifnotos.path.exists(dir):
        return map_pb2.filesStatus(fileLoadingStatus=0)
        file_names = [fileforfileinos.listdir(dir)
ifos.path.isfile(os.path.join(dir, file))]
        ifset(fileSearch).issubset(set(file_names)):
        return map_pb2.filesStatus(fileLoadingStatus=1)
        else:
        return map_pb2.filesStatus(fileLoadingStatus=0)
        exceptExceptionas e:
        message = {
                        'funcs': 'checkFiles',
                        'error': repr(e).replace('\n', '; ')
                    }
        logging.error(json.dumps(message))
        raise

        deffileList(self, request, context):
        info_list = []
        try:
        forfileinos.listdir(f'{request.dir}'):
        file_path = os.path.join(f'{request.dir}', file)
        ifos.path.isfile(file_path):
        info_list.append(map_pb2.fileInfoL(file=map_pb2.fileInfoObj(
        type='file',
        birthtime=str(datetime.fromtimestamp(
        os.path.getctime(file_path))),
        name=os.path.basename(file_path),
        size=os.path.getsize(file_path),
```

```
      ext=os.path.splitext(file_path)[1],
      server='grpc'
                        )))
      exceptExceptionas e:
      pass
      try:
      fordirinos.listdir(f'{request.dir}'):
      subdir_path = os.path.join(f'{request.dir}', dir)
      ifos.path.isdir(subdir_path):
      info_list.append(map_pb2.fileInfoL(dir=map_pb2.dirInfoObj(
      type='dir',
      birthtime=str(datetime.fromtimestamp(
      os.path.getctime(subdir_path))),
      name=os.path.basename(subdir_path),
      size=int(subprocess.check_output(f'du -sb "{subdir_path}" | cut -f1',
shell=True)),
      count=len(os.listdir(subdir_path)),
      server='grpc'
                        )))
      exceptExceptionas e:
      pass

      return map_pb2.fileInfo(fileInfoList=info_list)

      defreadDir(self, request, context):
      res = []
      dir_root = '/data/ortho/files/manager'
ifnotrequest.fullPathelserequest.fullPath
      filter_list = request.filterifrequest.filterelse ['.tif', '.tiff']
      try:
      forfileinos.listdir(f'{dir_root}/{request.dir}'):
      file_path = os.path.join(f'{dir_root}/{request.dir}', file)
      ifos.path.isfile(file_path) and (os.path.splitext(file_path)[1].lower()
infilter_list):
      res.append(os.path.basename(file_path))
      elifos.path.splitext(file_path)[1].lower() infilter_list:
      res.append(os.path.basename(file_path))
      except:
      pass

      return map_pb2.readDirOut(res=res)
```

ДОДАТОК В. КОД ФУНКЦІЇ ПРЕДПРОЦЕСОРА КОМАНД GDAL-CLI

```
defGdal(self,request,context):
start_time=time.time()
try:
pathIn=f"{request.path}"ifrequest.pathelse''
pathOut=f"{request.out}"ifrequest.outelse''
params=request.paramsifrequest.paramselse''
cm_name=request.name
ifnotcm_name:
raiseException('GDAL commandnameneeded!!!')
cmd=f"""{cm_name}{params}{pathIn}{pathOut}"""
print(cmd)
ifcm_name=='gdalbuildvrt':
cwd=os.path.dirname(pathOut).split(' -o ')[1]
res=subprocess.run(cmd.replace(cwd+'/',''),cwd=cwd,shell=True,stdout=subproc
ess.PIPE,stderr=subprocess.PIPE,universal_newlines=True)
else:
res=subprocess.run(cmd,shell=True,stdout=subprocess.PIPE,stderr=subprocess.P
IPE,universal_newlines=True)
ifres.stderrand'warning'notinres.stderr.lower()andnotres.stdout:
raiseException(res.stderr)
ifres.stderrand'warning'notinres.stderr.lower()andcm_name=='gdalwarp':
raiseException(res.stderr)
message={
'funcs':'Gdal',
'info':{'status':'OK','return':{'result':str(res.stdout)},'time':time.time()
-start_time},
'request':str(request)
}
logging.info(json.dumps(message))
returnmap_pb2.gdal_out(result=res.stdout,err='')
exceptExceptionase:
message={
'funcs':'Gdal',
'error':repr(e).replace('\n','; '),
'request':str(request)
}
logging.error(json.dumps(message))
returnmap_pb2.gdal_out(result='',err=f'{e}')
```

ДОДАТОК Г. КОД ФУНКЦІЙ СТВОРЕННЯ ТАЙЛІВ

```
defRender(self,request,context):
m=False
try:
if""'inrequest.path:
request.path=request.path.replace('"','')
ifnotrequest.pathandnotrequest.name:
raiseTypeError('Requirepathornamebutnothinggiven')
m=self.LoadXMLFile(request.path,request.name)
ifnotm:
returnmap_pb2.render_out(err='allrendererbusy')
# m = mapList[mapName];
m.ready=False

# m = mapList[md5]
start_time=time.time()
bbox=mapnik.Box2d(request.bbox[0],request.bbox[1],request.bbox[2],request.bb
ox[3])
m.zoom_to_box(bbox)

ifrequest.tile:
dir=os.path.dirname(request.tile)
ifnotos.path.exists(dir):
os.makedirs(dir,0o777,True)
mapnik.render_to_file(m,request.tile)
# print('tile',request.tile)
m.ready=True
message={
'funcs':'Render',
'info':{'status':'OK','time':time.time()-start_time},
'request':str(request)
}
logging.info(json.dumps(message))
returnmap_pb2.render_out(tile=request.tile,base64='',err='')
# imgrender
ifTrue:
im=mapnik.Image(256,256)
mapnik.render(m,im)
image_data=base64.b64encode(im.tostring('png'))
m.ready=True
message={
'funcs':'Render',
```

```
        'info':{'status':'OK','time':time.time()-start_time},
        'request':str(request)
        }
        logging.info(json.dumps(message))
        returnmap_pb2.render_out(base64=image_data,err='')

        # filerender

        filename='tmp/'+str(uuid4())+'.png'
        mapnik.render_to_file(m,filename)
        image_file=open(filename,"rb")
        img=image_file.read()
        image_data=base64.b64encode(img)
        os.remove(filename)

        message={
        'funcs':'Render',
        'info':{'status':'OK','return':{'base64':str(image_data)},'time':time.time()
-start_time},
        'request':str(request)
        }
        logging.info(json.dumps(message))
        returnmap_pb2.render_out(base64=image_data,err='')
        exceptExceptionase:
        ifm:
        m.ready=True
        message={
        'funcs':'Render',
        'error':repr(e).replace('\n','; '),
        'request':str(request)
        }
        logging.error(json.dumps(message))
        returnmap_pb2.render_out(base64='',err=str(e))

        defLoadXMLFile(self,path,name):
        md5=(hashlib.md5(path.encode())).hexdigest()ifpathelsename

        foriinrange(1,MAX_RENDERER_NUMBER):
        mapName=md5+"-"+str(i)

        # ifmapList.get(mapName) andmapList[mapName].time:
        #   print(mapName, time.time() - mapList[mapName].time)
```

```
      ifmapList.get(mapName)andnotmapList[mapName].readyand(time.time()-
mapList[mapName].time)>15:
      print(mapName,time.time()-mapList[mapName].time)
      mapList[mapName].ready=True

      ifmapList.get(mapName)andmapList[mapName].ready:
      # print('ready',mapName)
      mapList[mapName].time=time.time()
      returnmapList[mapName]
      ifmapList.get(mapName)andnotmapList[mapName].ready:
      continue

      name="tmp/"+md5+".xml"
      ifos.path.exists(path):
      name=path
      # print('create',mapName)
      m=mapnik.Map(256,256)
      mapnik.load_map(m,name)
      # return m

      mapList[mapName]=m
      mapList[mapName].ready=True
      mapList[mapName].time=time.time()
      # print(mapReady)
      returnmapList[mapName]
```

ДОДАТОК Ґ.ПРИКЛАД ФАЙЛУ КОНФІГУРАЦІЇ ДЛЯ СЕРВІСУ

```
syntax = "proto3";


messagerender_in {
  stringpath = 1;
  stringname = 2;
  repeateddoublebbox = 3 [packed=true];
  stringtile = 4;
  stringxml = 5;
  int32width = 6;
  int32height = 7;
}


messagerender_out {
  stringerr = 1;
  string base64 = 2;
  stringtile = 3;
}

messagexml_in {
  stringpath = 1;
  stringname = 2;
  stringxml = 3;
  boolreload = 4;
}

messagexml_out {
  stringerr = 1;
  boolis_ok = 2;
}

messagecmd_in {
  stringcmd = 1;
  stringsvg = 2;
  stringfilename = 3;
}

messagecmd_out {
  stringerr = 1;
```

```
  string base64 = 2;
}

messagegdal_in {
  stringname = 1;
  stringpath = 2;
  stringout = 3;
  stringparams = 4;
}
messagegdal_out {
  stringresult = 1;
  stringerr = 2;
}

messageclear_tiles_in {
  stringpath = 1;
}

messageclear_tiles_out {
  oneofclear_message {
          boolok = 1;
          stringerr = 2;
  }
}

messageempty {}

messagestatusRenderInfo {
  stringname = 1;
  boolisActive = 2;
  floatworkTime = 3;
}

messagestatusInfo {
  repeatedstatusRenderInfoinfo = 1;
}

messageprintMapOut {
  stringmap = 1;
  stringerr = 2;
}
```

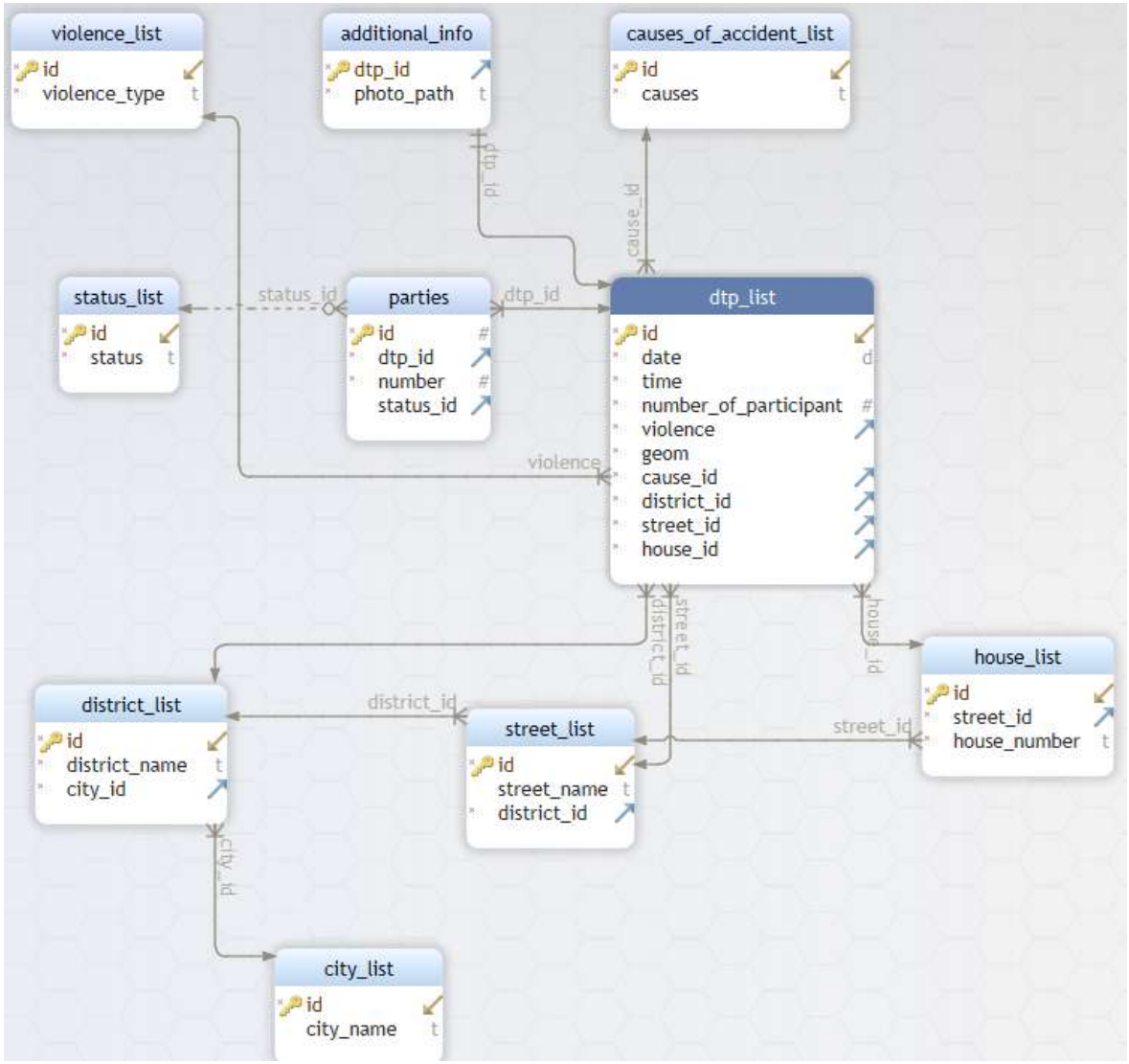
```
messagexyz {
  float x = 1;
  float y = 2;
  float z = 3;
}

messageprintMapIn {
  stringbaseurl = 1;
  stringbaseImg = 2;
  repeatedstringxml = 3;
  stringoverlayXml = 4;
  stringoverlay = 5;
  int32width = 6;
  int32height = 7;
  repeateddoublebbox = 8;
  xyzpos = 9;
  stringgeojsonSettings = 10;
}

messagelog_in {
  uint32rows = 1;
  stringlevel = 2;
}

messagelog_out {
  repeatedstringlogs = 1;
}

messagesqlToShpIn {
stringdb = 1;
stringhost = 2;
stringport = 3;
stringsql = 4;
stringsavePath = 5;
  stringshpName = 6;
}

messageshapeOut {
stringzipPath = 1;
}

messagefileIn {
```

```
  bytesfileBytes = 1;
  stringrelativeFilepath = 2;
}

messagedeleteFileIn {
  stringdir = 1;
  repeatedstringobjectList = 2;
}

messagefileManagerOut {
  boolstatus = 1;
  stringoperation = 2;
}

messagecheckFilesDir {
  stringdir = 1;
  repeatedstringfilesSearch = 2;
}

enumstatus {
  LOADING = 0;
  OK = 1;
}

messagefilesStatus {
  statusfileLoadingStatus = 1;
}

messagefileListIn {
  stringdir = 1;
}

messagefileInfo {
  repeatedfileInfoLfileInfoList = 1;
}

messagefileInfoL {
  oneoffileInfo {
          fileInfoObjfile = 1;
          dirInfoObjdir = 2;
  }
}
```

```
messagefileInfoObj {
  stringtype = 1;
  stringbirthtime = 2;
  stringname = 3;
  uint64size = 4;
  stringext = 5;
  stringserver = 6;
}

messagedirInfoObj {
  stringtype = 1;
  stringbirthtime = 2;
  stringname = 3;
  uint64size = 4;
  uint64count = 5;
  stringserver = 6;
}

messagedeleteOut {
  repeateddeleteObjSuccesssuccess = 1;
  repeateddeleteObjErrorerror = 2;

}

messagedeleteObjSuccess {
  stringobject = 1;
  stringtype = 2;
  stringstatus = 3;
}

messagedeleteObjError {
  stringobject = 1;
  stringfullPath = 2;
  stringresult = 3;
}

messagereadDirIn {
  stringdir = 1;
  repeatedstringfilter = 2;
  stringfullPath = 3;
}
```

```
messagereadDirOut {
  repeatedstringres = 1;

}

messagedownloadFileIn {
  stringfilePath = 1;
}

messagedownloadFileOut {
  byteschunk_data = 1;
}


serviceMap {
  rpcRender(render_in) returns (render_out) {}
  rpcRenderXML(render_in) returns (render_out) {}
  rpcLoadXML(xml_in) returns (xml_out) {}
  rpcMarkerIcon(cmd_in) returns (cmd_out) {}
  rpcGdal(gdal_in) returns (gdal_out) {}
  rpcClearTiles(clear_tiles_in) returns (clear_tiles_out) {}
  rpcRenderStatus(empty) returns (statusInfo) {}
  rpcPrintMap(printMapIn) returns (printMapOut) {}
  rpcLog(log_in) returns (log_out) {}
  rpcSqlToShp(sqlToShpIn) returns (shapeOut) {}
  rpcuploadFile(fileIn) returns (fileManagerOut) {}
  rpcdeleteFiles(deleteFileIn) returns (deleteOut) {}
  rpccheckFiles(checkFilesDir) returns (filesStatus) {}
  rpcfileList(fileListIn) returns (fileInfo) {}
  rpcreadDir(readDirIn) returns (readDirOut) {}
  rpcdownloadFile(downloadFileIn) returns (streamdownloadFileOut) {}
}
```

ДОДАТОК Д. ПРИКЛАД РЕАЛІЗАЦІЇ КЛАСУ У БДНА ПРИКЛАДІ ДТП



ДОДАТОК Е. ПРИКЛАД ФАЛУ КОНФІГУРАЦІЇ ДЛЯ ГЕНЕРУВАННЯ СТОРІНКИ


```
{
"label_style":"vertical",
"schema":{
"name":{
"type":"Text",
"ua":"Назва",
"ru":"Название",
"en":"Name",
"placeholder":{
"ua":"Назва",
"ru":"Название",
"en":"Name"
},
"validators":[
"required"
]
},
"type":{
"type":"Select2",
"ua":"Тип",
"default":"json",
"data":"user_cls.type",
"validators":[
"required"
],
"disable":true
}
}
}
```

ДОДАТОК Є. КОД РЕАЛІЗАЦІЇ КЛІЄНТА ДЛЯ СЕРВІСУ ВЕБ МАП

```
      constpath = require('path');
      constgrpc = require('@grpc/grpc-js');
      constprotoLoader = require('@grpc/proto-loader');

      constconfig = require('../../../../config/configServer.json');

      const { mapServerAddress } = config;

      constfsFuncsList = ['uploadFile', 'deleteFiles', 'downloadFile',
'checkFiles', 'fileList', 'readDir'];

      constproto = grpc.loadPackageDefinition(
      protoLoader.loadSync(path.join(__dirname, 'map.proto'), {
      keepCase: true,
      longs: String,
      enums: String,
      defaults: true,
      oneofs: true,
        }),
      );

      constmapClient = mapServerAddress ? newproto.Map(
      mapServerAddress,
      grpc.credentials.createInsecure(),
      ) : null;

      // eslint-disable-next-linemax-len
      constfuncs = ['render', 'renderXML', 'loadXML', 'markerIcon', 'gdal',
'printMap', 'log', 'sqlToShp', 'uploadFile', 'deleteFiles', 'downloadFile',
'checkFiles', 'fileList', 'readDir', 'renderStatus'].reduce((p, el) => {
      // eslint-disable-next-lineno-param-reassign
        p[el] = config.mapServerAddress ? async (data) =>newPromise((res, rej) =>
{
      constcapitalized = fsFuncsList.includes(el) ? el :
el.charAt(0).toUpperCase() + el.slice(1);
      // constmethod = mapClient[capitalized].bind(mapClient);
      consttime = Date.now();
      // eslint-disable-next-lineno-param-reassign
      if (data) {
      if (data?.path&&data.path?.[0] !== '"') {
      // eslint-disable-next-lineno-param-reassign
```

```
      data.path = `"${data.path}"`;
            }
          }

      if (el === 'downloadFile') {
      constdownloadBuffer = [];
      constdownload = mapClient[capitalized].call(mapClient, data, (err, data1) =>
{
      //      console.log(`grpc ${capitalized}: ${Date.now() - time}
${data.path}`);
      /* logger.file('grpc/map', {
      time: Date.now() - time, name: el, err: err || data1.err, status: err ? 500
: 200, param: { path: data.path, bbox: data.bbox },
              }); */
      if (err || data1?.result?.startsWith?.('Usage:')) {
      returnrej(err || data1?.result);
              }
      res(data1);
            });
      download.on('data', (resp) => {
      downloadBuffer.push(...resp.chunk_data);
            });
      download.on('end', () =>res({ data: Buffer.from(downloadBuffer) }));
          } else {
      mapClient[capitalized].call(mapClient, data, (err, data1) => {
      //      console.log(`grpc ${capitalized}: ${Date.now() - time}
${data.path}`);
      /* logger.file('grpc/map', {
      time: Date.now() - time, name: el, err: err || data1.err, status: err ? 500
: 200, param: { path: data.path, bbox: data.bbox },
              }); */
      if (err || data1?.result?.startsWith?.('Usage:')) {
      returnrej(err || data1?.result);
              }
      res(data1);
            });
          }
        }) : null;
      return p;
      }, {});

      module.exports = {
        ...funcs,
      mapServerAddress,
      };
```

ДОДАТОК Ж. КОД КЛІЄНТА ВІДДАЛЕНОГО СХОВИЩА ДАНИХ

```
      const { Readable } = require('stream');
      const { getS3Client } = require('./clientv2');
      const { getValidData, getDataSize } = require('../../helpers/helper');
      constdataTypes = require('../../helpers/handlers/dataTypes');

      constcopyFile = (s3Settings) =>async ({ filepath, destFilepath }) => {
      const s3Client = getS3Client(s3Settings);
      constbucketName = s3Settings.containerName;
      constcopyBucketParams = {
      Bucket: bucketName,
      CopySource: `${bucketName}/${filepath?.slice(1)}`, // pathshouldn'tstartwith
/
      Key: destFilepath?.slice(1),
        };
      constlogObj = {
      name: 'copyFile', filepath, destFilepath,
        };
      try {
      constcopyData = await s3Client.copyObject(copyBucketParams).promise();
      logObj.level = 'INFO';
      logObj.statusCode = 200;
      return { copyData };
        } catch (err) {
      logObj.level = 'ERROR';
      logObj.error = err;
      logObj.statusCode = err?.$metadata?.httpStatusCode || 404;
      returnrequire('../index')().copyFile({ filepath, destFilepath });
      // thrownewError('Файл не знайдено');
        } finally {
      logger.file('s3', logObj);
        }
      };

      constmoveFile = (s3Settings) =>async (filepath, destFilepath) => {
      const s3Client = getS3Client(s3Settings);
      constbucketName = s3Settings.containerName;
      // constcopyBucketParams = {
      //   Bucket: bucketName,
      //   CopySource: `${bucketName}/${filepath?.slice(1)}`, //
pathshouldn'tstartwith /
      //   Key: destFilepath?.slice(1),
```

```
// };
constbucketParamsDownload = {
Bucket: bucketName,
Key: filepath?.slice(1),
  };
constdeleteBucketParams = {
Bucket: bucketName,
Key: filepath?.slice(1),
  };

constlogObj = {
name: 'moveFile', filepath, destFilepath,
  };
try {
// constcopyData = await s3Client.copyObject(copyBucketParams).promise();
const { Body } = await s3Client.getObject(bucketParamsDownload).promise();
constbucketParamsUpload = {
Bucket: bucketName,
Key: destFilepath?.slice(1),
Body,
    };
constres = await s3Client.upload(bucketParamsUpload).promise();

constdeleteData = await s3Client.deleteObject(deleteBucketParams).promise();
logObj.level = 'INFO';
logObj.statusCode = 200;
return { res, deleteData };
  } catch (err) {
logObj.level = 'ERROR';
logObj.error = err;
logObj.statusCode = err?.$metadata?.httpStatusCode || 404;
returnrequire('../index')().moveFile(filepath, destFilepath);
// thrownewError('Файл не знайдено');
  } finally {
logger.file('s3', logObj);
  }
};

constgetFileStream = (s3Settings) =>async (filepath, options = {}) => {
const s3Client = getS3Client(s3Settings);
constbucketName = s3Settings.containerName;
constbucketParams = {
```

```
      Bucket: bucketName,
      Key: filepath?.slice(1),
        };
      constlogObj = {
      name: 'downloadFile', filepath,
        };
      try {
      constdata = await s3Client.getObject(bucketParams).promise();
      logObj.level = 'INFO';
      logObj.statusCode = 200;
      if (options.buffer) {
      returndata.Body;
          }
      returnReadable.from(data.Body);
        } catch (err) {
      logObj.level = 'ERROR';
      logObj.error = err;
      logObj.statusCode = err?.$metadata?.httpStatusCode || 404;
      returnrequire('../index')().downloadFile(filepath);
      // thrownewError('Файл не знайдено');
        } finally {
      logger.file('s3', logObj);
        }
      };

      constuploadFile = (s3Settings) =>async ({ filepath, data }) => {
      constvalidData = awaitgetValidData({ data, types: [dataTypes.buffer] });
      constsize = awaitgetDataSize({ data });
      constlogObj = {
      name: 'uploadFile', filepath, size, data: data.constructor.name, validData:
validData.constructor.name,
        };
      try {
      const s3Client = getS3Client(s3Settings);
      constbucketName = s3Settings.containerName;
      constbucketParams = {
      Bucket: bucketName,
      Key: filepath?.slice(1),
      Body: validData,
          };
      constres = await s3Client.upload(bucketParams).promise();
      logObj.level = 'INFO';
```

```
      logObj.statusCode = 200;
      returnres; // Forunittests.
        } catch (err) {
      logObj.level = 'ERROR';
      logObj.error = err;
      logObj.statusCode = err?.$metadata?.httpStatusCode || 404;
      if (err.retryable) {
      awaituploadFile(s3Settings)({ filepath, data });
      return;
          }
      thrownewError(`Uploaderror s3: ${err}. Message: ${err.message}. Keys:
${Object.keys(err)}. Filepath: ${filepath}. Size:${size}Datatype:
${typeofvalidData}.`);
        } finally {
      logger.file('s3', logObj);
        }
      };

      constgetFileMetadata = (s3Settings) =>async (filepath) => {
      const s3Client = getS3Client(s3Settings);
      constbucketName = s3Settings.containerName;
      constbucketParams = {
      Bucket: bucketName,
      Key: filepath?.slice(1),
        };
      constlogObj = {
      name: 'headFile', filepath,
        };
      try {
      constdata = await s3Client.headObject(bucketParams).promise();
      logObj.level = 'INFO';
      logObj.statusCode = 200;
      returndata;
        } catch (err) {
      logObj.level = 'ERROR';
      logObj.error = err;
      logObj.statusCode = err?.$metadata?.httpStatusCode || 404;
      returnrequire('../index')().fileExists(filepath);
      // returnfalse;
        } finally {
      logger.file('s3', logObj);
        }
```

```
};

constdeleteFile = (s3Settings) =>async (filepath) => {
const s3Client = getS3Client(s3Settings);
constbucketName = s3Settings.containerName;

constdeleteBucketParams = {
Bucket: bucketName,
Key: filepath?.slice(1),
  };
constlogObj = {
name: 'deleteFile', filepath,
  };
try {
constresp = await s3Client.deleteObject(deleteBucketParams).promise();
logObj.level = 'INFO';
logObj.statusCode = 200;
returnresp;
  } catch (err) {
logObj.level = 'ERROR';
logObj.error = err;
logObj.statusCode = err?.$metadata?.httpStatusCode || 404;
returnrequire('../index')().deleteFile(filepath);
// thrownewError('Файл не знайдено');
  } finally {
logger.file('s3', logObj);
  }
};

/*
* exampleprefix 'data/softpro/work/geo/admin_dabi/files/report_pgbadger/'
* shouldbenoslashatthebeginning!!!
*
*/
constlistObjects = (s3Settings) =>async (prefix) => {
const s3Client = getS3Client(s3Settings);
constbucketName = s3Settings.containerName;
constbucketParams = {
Bucket: bucketName,
Prefix: prefix,
  };
constlogObj = {
```

```
prefix,
  };
try {
constdata = await s3Client.listObjectsV2(bucketParams).promise();
logObj.level = 'INFO';
logObj.statusCode = 200;
returndata.Contents.map((obj) =>obj.Key);
  } catch (err) {
// console.log(err);
logObj.level = 'ERROR';
logObj.error = err;
logObj.statusCode = err?.$metadata?.httpStatusCode || 404;
return [];
  } finally {
logger.file('s3', logObj);
  }
};

constgetFileSize = (s3Settings) =>async ({ filepath }) => {
constmetadata = awaitgetFileMetadata(s3Settings)(filepath);
returnmetadata.ContentLength;
};

constgetMDate = (s3Settings) =>async ({ filepath }) => {
constmetadata = awaitgetFileMetadata(s3Settings)(filepath);
returnnewDate(metadata.LastModified);
};

module.exports = (s3Settings) => ({
name: 's3v2',
hardDeleteFile: deleteFile(s3Settings),
deleteFile: moveFile(s3Settings),
downloadFile: getFileStream(s3Settings),
uploadFile: uploadFile(s3Settings),
copyFile: copyFile(s3Settings),
moveFile: moveFile(s3Settings),
fileExists: getFileMetadata(s3Settings),
listObjects: listObjects(s3Settings),
getFileSize: getFileSize(s3Settings),
getMDate: getMDate(s3Settings),
});
```

ДОДАТОК З. КОД РЕАЛІЩАЦІЇ КЛІЄНТА REDIS

```
constredis = require('redis');
const { randomUUID: v4 } = require('crypto');
constconfig = require('../../config/configServer.json');

functionpromisifyRedisCommand(rclient) {
constcacheProps = {};
returnnewProxy(rclient, {
get(target, prop) {
if (cacheProps[prop]) {
returncacheProps[prop]();
      }
constisJSON = prop.includes('JSON');
constnameMethod = prop.replace(/JSON$/, '').replace(/Async$/, '');
if (typeoftarget[nameMethod] !== 'function') {
cacheProps[prop] = () =>target[nameMethod];
      } else {
cacheProps[prop] = () =>(...args) => (typeofargs[args.length - 1] ===
'function' ? target[nameMethod](...args) : newPromise((resolve, reject) => {
try {
target[nameMethod](...args, (err, result) => {
if (err) {
//                console.error(target._uuid);
reject(err);
               } else {
resolve(isJSON ? JSON.parse(result) : result);
               }
             });
           } catch (err) {
reject(err);
           }
         }));
      }
returncacheProps[prop]();
    },
  });
}

const { cache } = require('./cache');

functionCreate(setting) {
constrclient = redis.createClient({
```

```
enable_offline_queue: true,
host: setting.host,
port: setting.port,
no_ready_check: true,
db: setting.db,
auth_pass: setting.password || null,
offline_queue_length: 100,
// retry_strategy: () =>setting.reconnection_interval || 1000000000,
  });
rclient.on('error', (err) => {
console.log('rediserror', rclient._uuid, err);
  });

// require('./close').then(() =>rclient.quit());
rclient._uuid = v4();
returnrclient;
}

functionGetCacheOrCreate(param) {
constchangeSetting = {
host: config.redis?.host || '127.0.0.1',
port: config.redis?.port || 6379,
  };

constsetting = { ...changeSetting, ...param };

constkey = `${setting.host}-${setting.db || 0}`;
if (cache[key] && !setting.new) {
returncache[key];
  }
constrclient = Create(setting);
constpromiseRclient = promisifyRedisCommand(rclient);
cache[key] ||= promiseRclient;
returnpromiseRclient;
}

module.exports = GetCacheOrCreate;
```