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

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 126 «Інформаційні системи та технології»
на тему: «Платформа управління сховищем даних»

Виконала:
студентка IV курсу, групи ІС-93
Кравченко Олена Олексіївна _____

Керівник:
доцент каф. ІСТ, к.т.н.
Галушко Дмитро Олександрович _____

Рецензент:
ст. викладач каф. ОТ
Алещенко Олексій Вадимович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Кравченко Олені Олексіївні

1. Тема проєкту «Платформа управління сховищем даних», керівник проєкту Галушко Дмитро Олександрович, к.т.н, доцент каф. ІСТ, затверджені наказом по університету від «31» травня 2023 р. №2101-с

2. Термін подання студентом проєкту: 12 червня 2023 року

3. Вихідні дані до проєкту: HTTP-відповідь, зміна стану сховища даних

4. Зміст пояснювальної записки:
1. Аналіз наявних аналогів: типи сховищ даних, огляд наявних сховищ даних.
2. Опис системи: опис процесу діяльності, опис функціональної моделі, засоби розробки, інформаційне забезпечення.
3. Реалізація системи: змістовна постановка задачі, математична постановка задачі, обґрунтування методу розв'язання, опис методу розв'язання, архітектура програмного забезпечення.
4. Технологічний розділ: керівництво користувача, випробування програмного продукту.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)
1. Діаграма варіантів використання

2. Entity Relationship діаграма бази даних
3. Діаграма послідовності створення сховища даних
4. Діаграма послідовності запису даних

6. Дата видачі завдання 27 лютого 2023 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Аналіз наявних аналогів	17.04.2023-23.04.2023	
2	Створення опису системи	24.04.2023-30.04.2023	
3	Реалізація системи	01.05.2023-07.05.2023	
4	Створення технологічного розділу	08.05.2023-14.05.2023	
5	Створення висновків	15.05.2023-21.05.2023	
6	Оформлення пояснювальної записки	22.05.2023-31.05.2023	

Студент Олена КРАВЧЕНКО

Керівник Дмитро ГАЛУШКО

АНОТАЦІЯ

Кравченко О.О. Платформа управління сховищем даних. КПІ ім. Ігоря Сікорського, Київ 2023.

Пояснювальна записка містить 4 розділи, 16 ілюстрацій, 11 таблиць, 4 кресленики, 16 інформаційних посилань.

Ключові слова: великий обсяг даних, прикладний програмний інтерфейс для управління сховищем даних, розподілена система, сховище даних, централізоване джерело даних

Об’єктом розробки є сховище даних.

Предметом розробки є інструмент для управління сховищем даних.

Метою роботи є спрощення процесу обробки та аналізу даних з декількох джерел для прогнозування та підтримки прийняття рішень.

У дипломному проєкті було розроблено платформу управління сховищем даних, яку можна використовувати з власним налаштуванням кластерів для обробки та збереження даних, які можуть бути локальними або ж розгорнуті за допомогою хмарних сервісів.

Платформа управління сховищем даних була реалізована у вигляді прикладного програмного інтерфейсу на мові програмування Python, основними бібліотеками є FastAPI та PySpark. В якості бази даних було вибрано Postgres.

Отримані результати можуть бути корисними при розробці розподіленої системи для роботи з великим обсягом даних.

SUMMARY

Kravchenko O.O. Data warehouse control platform. Igor Sikorsky KPI, Kyiv, 2023.

The project contains 4 chapters, 16 illustrations, 11 tables, 4 drawings, 16 information links.

Keywords: application programming interface for data warehouse management, big data, centralized data source, data storage, distributed system

Its objective is data warehouse.

The subject of development is a tool designed for data warehouse management.

The project aims to simplify the processing and analysis of data from multiple sources, enabling forecasting and decision-making support.

As part of the project, a data storage control platform was developed. It can be used with your own configuration of clusters for data processing and storage, whether they are local or deployed using cloud services.

The data storage control platform was implemented as an application programming interface using the Python programming language, with FastAPI and PySpark serving as the main libraries. Postgres was selected as the database.

The achieved results can be applied to the development of a distributed system for handling large volumes of data.

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІС93.130БАК.004 ПЗ	Пояснювальна записка	61		
6	А3	ІС93.130БАК.004 Д1	Платформа управління	1		
7			сховищем даних. Діаграма			
8			варіантів використання			
9	А3	ІС93.130БАК.004 Д2	Платформа управління	1		
10			сховищем даних. Entity			
11			Relationship діаграма бази даних			
12	А3	ІС93.130БАК.004 Д3	Платформа управління	1		
13			сховищем даних. Діаграма			
14			послідовності створення			
15			сховища даних			
16	А3	ІС93.130БАК.004 Д4	Платформа управління	1		
17			сховищем даних. Діаграма			
18			послідовності запису даних			
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						

Зм.	Аркуш	№ докум.	Підпис	Дата	ІС93.130БАК.004 ТП		
Розроб.		Кравченко О.О.			Платформа управління сховищем даних. Відомість проєкту	Літ.	Аркуш 1 / Аркушів 1
Керівн.		Галушко Д.О.				Т	
Затв.						КПІ ім. Ігоря Сікорського Група ІС-93	

Пояснювальна записка
до дипломного проєкту
на тему: «Платформа управління сховищем даних»

Київ – 2023 року

ЗМІСТ

ВСТУП

Перші теоретичні напрацювання, пов’язані із сховищем даних, з’явились у 1960-х, а фінальна концепція – у кінці 1980-х. Вона полягала у створенні архітектурної моделі для потоку бізнес-даних із різних джерел у певне середовище підтримки прийняття рішень. Головною проблемою було те, що у великих організаціях ці середовища існували незалежно один від одного та, як правило, часто потребували однакових процесів обробки даних та схожих наборів даних. Також джерела постійно оновлювались та вимоги до них часто переглядались, що створювало додаткову складність обслуговування декількох незалежних середовищ. Для вирішення цих проблем була запропонована концепція універсального та централізованого сховища даних.

Потреба у розподілених системах зберігання інформації випливає із швидкого зростання обсягів даних завдяки поширенню використання онлайн-сервісів, соціальних мереж, веб сайтів, пристроїв Інтернету речей та при цьому постійному запиту на аналіз цієї інформації з метою отримання цінних даних та для підтримки прийняття рішень. Традиційні рішення не здатні забезпечувати роботу в цих умовах.

Таким чином, сховище даних стало невід’ємною частиною інформаційних систем багатьох організацій. Завдяки постійному розвитку технологій, сховища даних стають більш ефективними, масштабованими та гнучкими до потреб організацій. За допомогою них є можливість аналізувати великі масиви даних та створювати процеси для очищення та отримання цінних висновків, як-от визначення тенденцій на ринку цінних паперів, прогнозування та стратегічне планування бізнесу тощо.

На сьогоднішній день більшість наявних сервісів для управління сховищем даних тісно інтегровані з певною хмарною платформою або декількома найпопулярнішими. У дипломному проєкті було розроблено платформу управління сховищем даних, яку можна використовувати з власним налаштуванням кластерів для обробки та збереження даних.

Об’єктом розробки є сховище даних.

Предметом розробки є інструмент для управління сховищем даних.

Метою роботи є спрощення процесу обробки та аналізу даних з декількох джерел для прогнозування та підтримки прийняття рішень.

Для досягнення мети можна виділити наступні завдання:

– огляд та аналіз наявних сервісів для управління сховищем даних;

– проектування, розробка та перевірка програмного продукту на відповідність вимогам;

– виокремлення основних типів джерел даних;

– інтеграція джерел даних у програмний продукт.

Дипломний проєкт складається з наступних розділів: вступ, основні розділи, висновки, список використаних джерел з 16 найменувань. Графічна частина включає 4 креслеників формату А3. Загальний обсяг 61 сторінка.

1 АНАЛІЗ НАЯВНИХ АНАЛОГІВ

Перші реалізації сховища даних з’явились на початку 1990-х. У 2008 році вийшла книга Біла Інмона, Дерека Страуса та Genia Neushloss «DW 2.0: The Architecture for the Next Generation of Data Warehousing» і сховища даних набули сучасного вигляду. Згодом виникли такі програмні продукти як Snowflake (2012 р.), Amazon RedShift (2012 р.), Google BigQuery (2012 р.), Azure Synapse Analytics (2016 р.) та інші, здебільшого як частина хмарних платформ.

1.1 Типи сховищ даних

В даному розділі розглядається класифікація сховищ даних за такими критеріями: за типом джерела даних, за типом архітектури системи.

1.1.1 За типом джерела даних

За типом джерела даних виділяють корпоративне сховище даних (англ. «Enterprise Data Warehouse»), операційне сховище даних (англ. «Operational Data Store») та вітрину даних.

Корпоративне сховище даних – це централізоване сховище, яке зберігає всі дані певної організації з усіх її джерел у стандартизованому форматі. Цей тип сховища даних зазвичай складається з наступних компонентів:

– джерела даних з операційних та транзакційних систем;

– проміжне середовище для збирання, очищення та стандартизації даних;

– середовище загального використання для проведення аналітики (створення запитів та звітів);

– набір інструментів для інтеграції даних (застосунки для бізнес-аналітики (англ. «Business Intelligence»), для здійснення Extract, Transform, Load процесу);

– прикладний програмний інтерфейс (англ. «Application Programming Interface»).

Основними функціями є:

– доступ до будь-яких даних організації в реальному часі;

– підтримка прийняття рішень та надання даних для предикативного аналізу і цілісного розуміння актуальних потреб бізнесу;

– надання можливості проводити аудит щодо відповідності даних певним стандартам та вимогам (наприклад, General Data Protection Regulation).

Головними недоліками є висока вартість обслуговування та сильна зв’язність системи.

Операційне сховище даних – це джерело даних для корпоративного сховища даних, описаного вище. Воно зберігає дані з операційних систем та баз даних у реальному або наближеному до реального часу та використовується для оперативної звітності. Негативним наслідком цього є відсутність історичних даних за довгий період часу. Операційне сховище є гнучким та може бути швидше зміненим відповідно до бізнес вимог та потреб в даних, ніж корпоративне сховище.

Вітрина даних – це частина сховища даних, яка зазвичай використовується певним відділом організації та сфокусована на конкретному бізнес напрямку, наприклад, кадровому обліку. Наслідком цього є відносно невелика вартість розробки та можливість швидко модифікувати та розширювати вітрину даних. Недоліком є обмежена кількість даних, які можуть бути не стандартизовані між різними вітринами, викликаючи через це ускладнення запитів для аналізу та прийняття рішень.

1.1.2 За типом архітектури системи

За типом архітектури системи можна виокремити модель Кімбалла (авторства Ральфа Кімбалла), модель Інмона (авторства Вільяма Інмона) та Data Vault.

Модель Кімбалла полягає у використанні вимірної моделі даних, яка має схему зірки (рисунок 1.1) – одну центральну таблицю (таблицю фактів) та декілька таблиць розмірностей. Таблиця фактів складається із зовнішніх ключів до

розмірних даних, метаданих (дата, час) та сурогатних ключів для ідентифікації кожного запису. Таблиця розмірностей містить велику кількість атрибутів для детального опису даних.

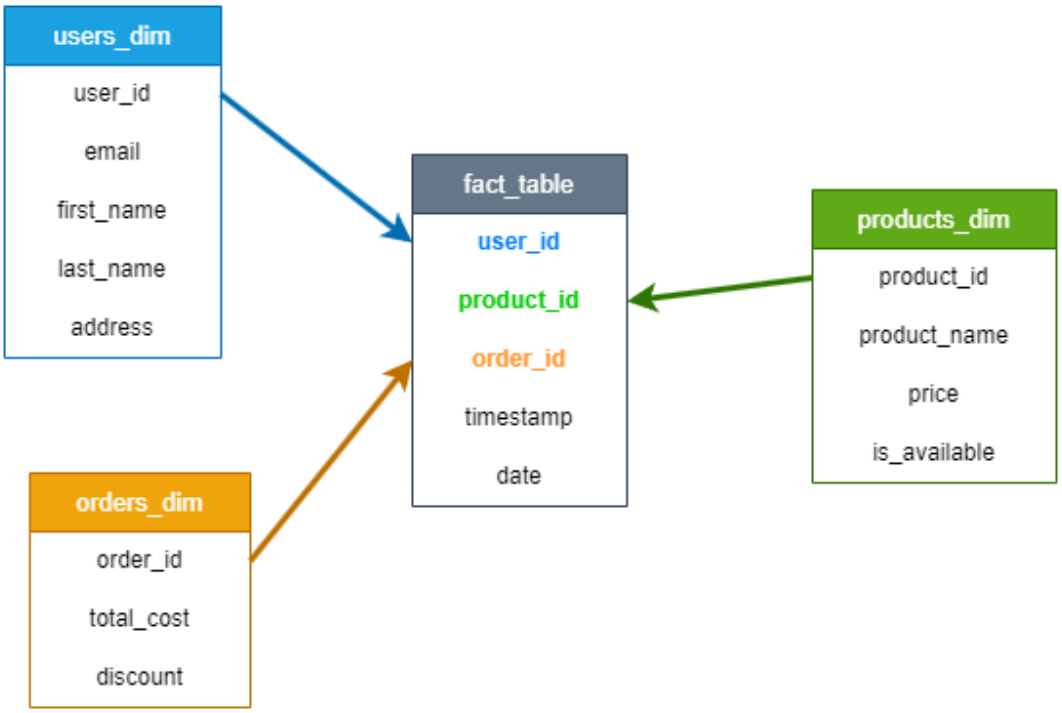

Рисунок 1.1 – Приклад схеми зірки

Схема зірки є ненормалізованою, тобто такою, що може містити надмірні дані та як наслідок може мати логічно помилкові результати (аномалії). Перевагами використання цієї схеми є проста зрозуміла логіка запитів та оптимізація зчитування даних за допомогою використання індексів, зовнішніх та сурогатних ключів. Іншою особливістю моделі є висхідний підхід, який передбачає ітеративне створення вітрин даних для певних бізнес вимог, які потім інтегруються у сховище даних. Цей підхід надає можливість змінювати сховище даних в залежності від бізнес вимог, оскільки будь-які зміни в них призводять до змін на рівні певних вітрин даних та не впливають на все сховище даних. Недоліками є відсутність узгодженості та стандартизації даних і необхідність обслуговувати кожну вітрину даних окремо. Приклад моделі Кімбалла зображено на рисунку 1.2.

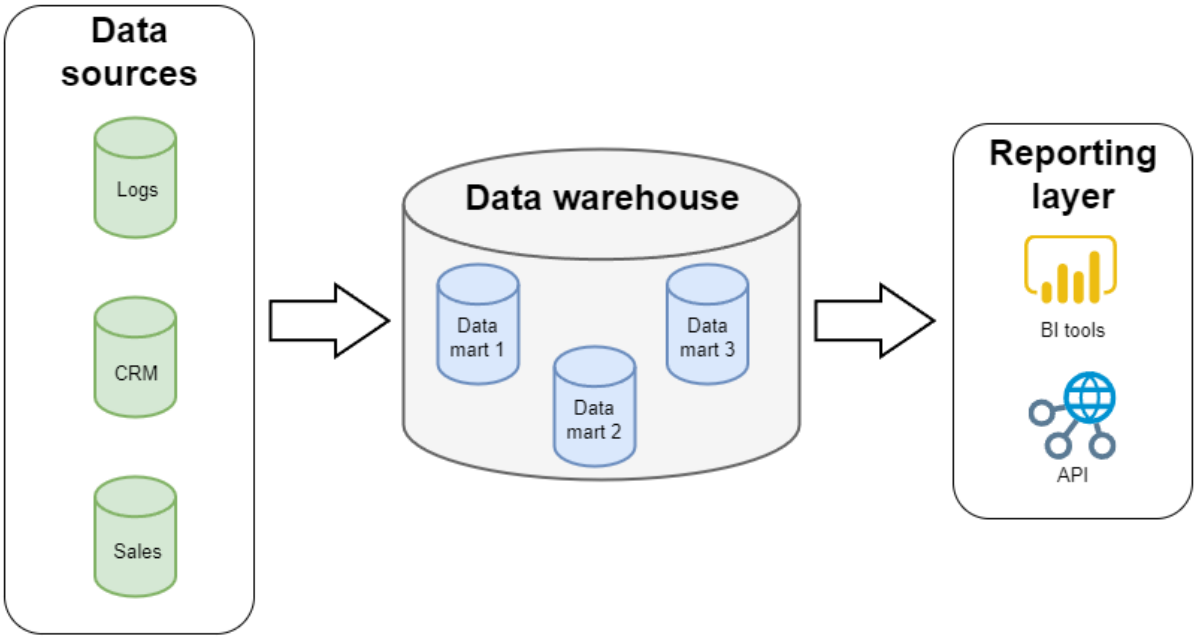

Рисунок 1.2 – Приклад моделі Кімбалла

Модель Інмона полягає у створенні єдиного джерела даних, яке використовуються для аналізу та прийняття рішень в певній організації. Але для цього необхідні етапи очистки, обробки та нормалізації даних, які потребують багато часу під час розробки. У цій моделі використовується низхідний підхід, який забезпечує точність та стандартизацію даних відповідно до бізнес вимог та полягає у структурованому підході до створення сховища даних. Недоліком є відносно велика складність розробки через нормалізацію даних та внесення змін у систему через сильну зв'язність. Приклад моделі Інмона зображено на рисунку 1.3.

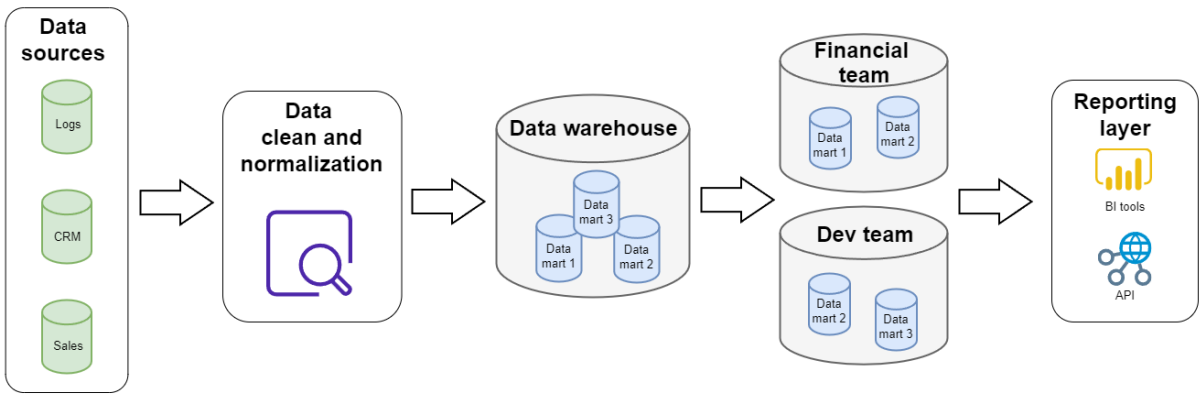

Рисунок 1.3 – Приклад моделі Інмона

Data Vault – це гібридний підхід, який поєднує властивості схеми зірки та 3-ої нормальної форми. Він має три типи сутностей (рисунок 1.4) – хаби (англ. «hubs»), посилання (англ. «links») та супутники (англ. «satellites»).

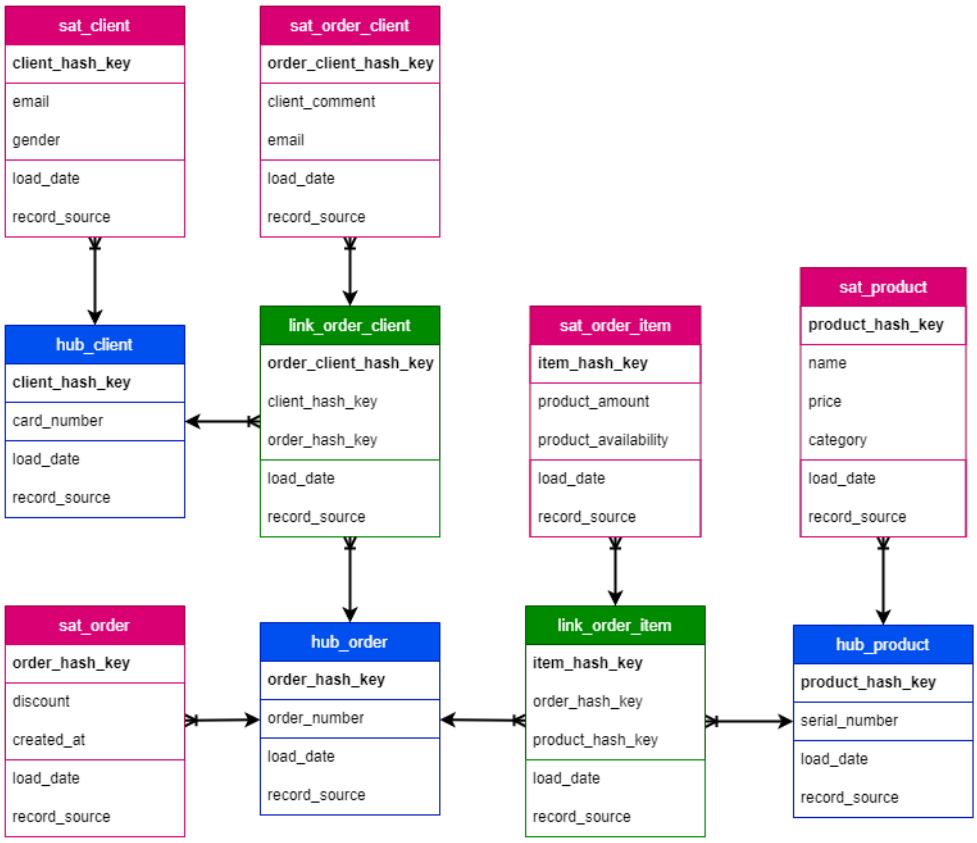

Рисунок 1.4 – Приклад Data Vault

Хаби містять в собі ключові бізнес дані (індивідуальний податковий номер клієнта, серійний номер продукту, адреса доставки), які мають бути унікальними та незмінними, та метадані (дата створення запису, звідки прийшов цей запис). Посилання необхідні для того, щоб показати зв'язки багато-до-багатьох між хабами, тому вони містять дані з цих сутностей. Між хабами або посиланнями та супутниками будується зв'язок один-до-багатьох. В супутнику знаходяться додаткові дані для опису батьківської сутності (хабу або посилання) та для історичного аналізу на основі версій. Важливо зазначити, що супутник може бути пов'язаний лише з однією сутністю та нею не може бути інший супутник. Також варто зазначити, що Data Vault надає можливість мати декілька версій даних для

подальшого аналізу. Перевагами такого підходу є висока гнучкість, масштабованість, в тому числі завдяки версіонуванню, та структурованість даних, яка забезпечена використанням трьох сутностей, які наведені вище. Однак, недоліком такої архітектури є надлишковість даних та ускладнення запитів через велику кількість додаткових зв'язків між сутностями.

1.2 Огляд наявних сховищ даних

1.2.1 Snowflake

Хмарне середовище даних Snowflake є повністю автономним сервісом. Тобто, користувачу не потрібно вибирати, налаштовувати і обслуговувати технічне обладнання (сервери, центри даних) і програмне забезпечення. Всі можливі налаштування, обслуговування, оптимізації та оновлення проводяться на стороні хмарного середовища Snowflake. В нього входить сховище, обробка даних і сервіси аналітики, які швидші, простіші у використанні та набагато гнучкіші, ніж традиційні пропозиції [1].

Snowflake не використовує жодну існуючу технологію баз даних або програмні продукти для великих обсягів даних, такі як Hadoop, а має власний підхід запитів SQL із власною архітектурою, спеціально розробленою для хмарних платформ [1].

Snowflake використовує віртуальні обчислювальні одиниці для потреб в обчисленні та окремі сервіси для зберігання даних. Цей хмарний продукт не підтримує приватні хмарні інфраструктури (локальні або виділені хмарною платформою). Також Snowflake – це не пакетне програмне забезпечення, яке може встановити користувач, тому всіма етапами встановлення та оновлення керує сам хмарний продукт.

Архітектура Snowflake – це гібридний підхід, де поєднано традиційну архітектуру вузлів із спільним диском та архітектуру бази даних без спільної роботи. Подібно до першого типу архітектури, Snowflake використовує центральне сховище даних для постійних даних, які доступні з усіх обчислювальних вузлів

платформи. З іншої сторони, обробка запитів відбувається за допомогою обчислювальних кластерів масово-паралельної архітектури, характерними особливостями якої є розподілені блоки обчислень з власною пам'яттю, за рахунок чого не виникає проблем з масштабуванням. Загалом, такий підхід забезпечує простоту керування даними на спільних дисках одночасно з високою продуктивністю та масштабованістю.

Snowflake складається з 3 основних шарів – шару сховища даних, шару обробки запитів та шару хмарних сервісів. Шар сховища даних відповідає за зберігання даних у оптимізованому, скомпресованому форматі по окремим стовпцям у хмарному середовищі. Snowflake керує всіма етапами того як ці дані зберігаються – організацією, розміром файлів, структурою, стисненням, метаданими, статистикою та багатьма іншими. У користувача немає прямого доступу до записів у даному хмарному середовищі, вони доступні лише через SQL запити, які виконуються за допомогою Snowflake. На рисунку 1.5 зображена схема архітектури Snowflake.

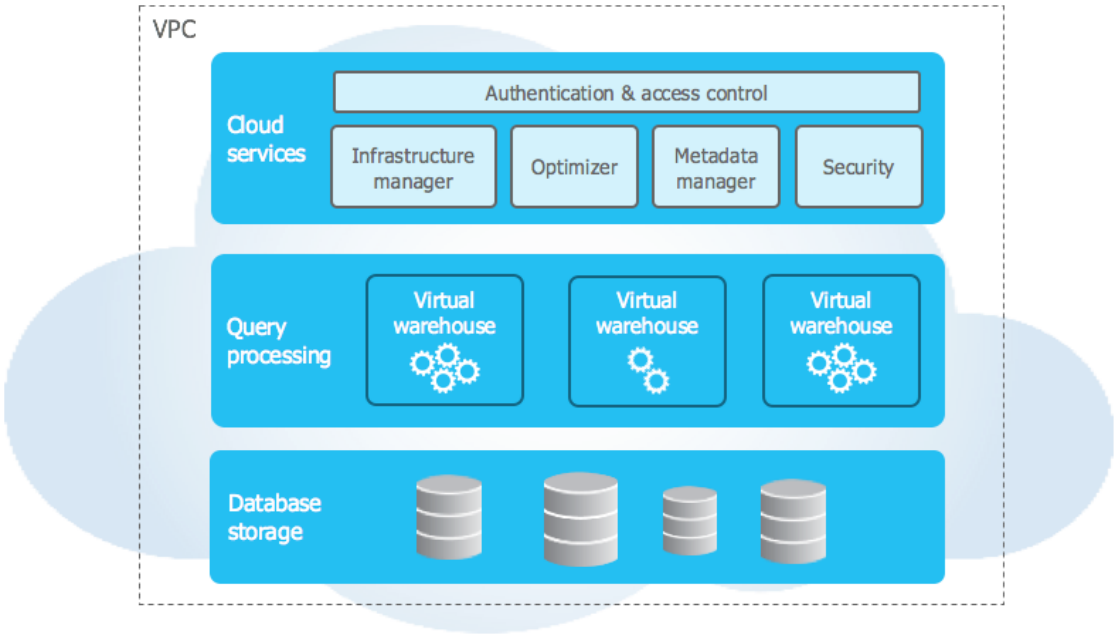

Рисунок 1.5 – Схематичне зображення архітектури Snowflake [1]

Виконання запитів виконується на рівні шару обробки запитів за допомогою так званих «віртуальних сховищ». Кожне віртуальне сховище – це обчислювальний

кластер масово-паралельної архітектури, що складається з кількох вузлів, виділених системою у хмарній платформі. Ці кластери є незалежними та не мають спільних ресурсів з іншими віртуальними сховищами. Таким чином, жоден з них не впливає на продуктивність інших.

Шар хмарних сервісів – це набір сервісів, які координують діяльність системи, серед них: аутентифікація, управління інфраструктурою, управління метаданими, первинна обробка та оптимізація запитів, управління доступом користувача. Ці сервіси об’єднують усі компоненти Snowflake від входу користувача в систему до відправлення та обробки запитів. Цей шар також працює на обчислювальних одиницях хмарної платформи, на потужностях якої працює Snowflake.

Snowflake підтримує декілька способів підключення та використання:

– веб-інтерфейс користувача, з якого можна отримати доступ до всіх компонентів;

– застосунки командного рядка (наприклад, SnowSQL), за допомогою яких можна отримати доступ до всіх компонентів;

– драйвери ODBC і JDBC, які можуть бути використані в інших програмних продуктах для підключення до Snowflake;

– власні програмні продукти (наприклад, веб-додаток з використанням Python, Apache Spark), які можна використовувати для подальшої розробки для підключення до Snowflake;

– інші наявні програмні продукти, які можна використовувати для таких функцій, як інструменти ETL (наприклад, Informatica) і інструменти BI (наприклад, ThoughtSpot).

1.2.2 Amazon Redshift

Завдяки Amazon Redshift користувач отримує доступ до даних та можливість їх проаналізувати без попереднього налаштування спеціального сховища даних. Всі ресурси автоматично виділяються сервісом, а обсяг сховища масштабується в

залежності від навантаження для забезпечення високої продуктивності. З користувача не стягується плата за період простою сховища, тобто користувач платить лише за ті ресурси, які він використовує. Для отримання даних та надсилання запитів достатньо або скористатись вбудованим інструментом для запитів, або налаштувати будь-який інший інструмент для бізнес-аналітики (англ. «Business Intelligence») [2].

Amazon Redshift можна використовувати як безсерверний сервіс, так і налаштувати власні кластери на хмарній платформі Amazon.

Безсерверний варіант має два основні концепти – простір імен (англ. «namespace») та робоча група (англ. «workgroup»). Простір імен – це набір об’єктів бази даних та користувачів, який об’єднує всі ресурси, які використовує кластер, як-от схеми, таблиці, користувачі, об’єкти спільних даних та фіксовані стани системи (англ. «snapshots»). Робоча група – це набір обчислювальних ресурсів, які використовуються для виконання обчислювальних завдань. До даних ресурсів належать модулі обробки Redshift (RPU), безпекові групи, ліміти використання. Параметри мережі та безпеки робочих груп можуть бути задані за допомогою вбудованого терміналу Amazon Redshift Serverless, інтерфейсу командного рядка AWS або прикладного програмного інтерфейсу (англ. «Application Programming Interface»).

Amazon Redshift, який використовує кластери, виділені хмарною платформою, включає в себе 2 типи кластерів – звичайний та базу даних. Звичайний кластер є основним компонентом інфраструктури сховища даних. Він складається з одного або кількох обчислювальних вузлів, які запускають скомпільований код. Якщо кластер має два або більше вузли, то створюється головний вузол, який координує роботу інших. Він обробляє зовнішні запити від програмних продуктів, наприклад від інструментів бізнес-аналітики та редакторів запитів. Кластер бази даних містить одну або декілька баз даних, в яких зберігаються дані користувача. Цей тип кластеру також створює головний вузол, який координує виконання запитів, відправлених певним SQL клієнтом. Організація даних відбувається за допомогою схем.

Amazon Redshift – це система керування реляційною базою даних (англ. «RDBMS»), яка сумісна з іншими аналогічними інструментами. Вона має стандартний для такого типу систем функціонал, в який входять обробка онлайн транзакцій - запис та видалення даних. Також сервіс можна використовувати для створення звітів на основі наборів даних і для аналізу даних за визначений період.

Сховище даних Amazon Redshift – це корпоративна система управління реляційною базою даних та запитами. Воно підтримує декілька розповсюджених способів підключення, зокрема з інструментами бізнес-аналітики, створення звітів, аналізу даних. На рисунку 1.6 зображено 3 основні етапи потоку обробки даних: отримання, обробка та використання даних.

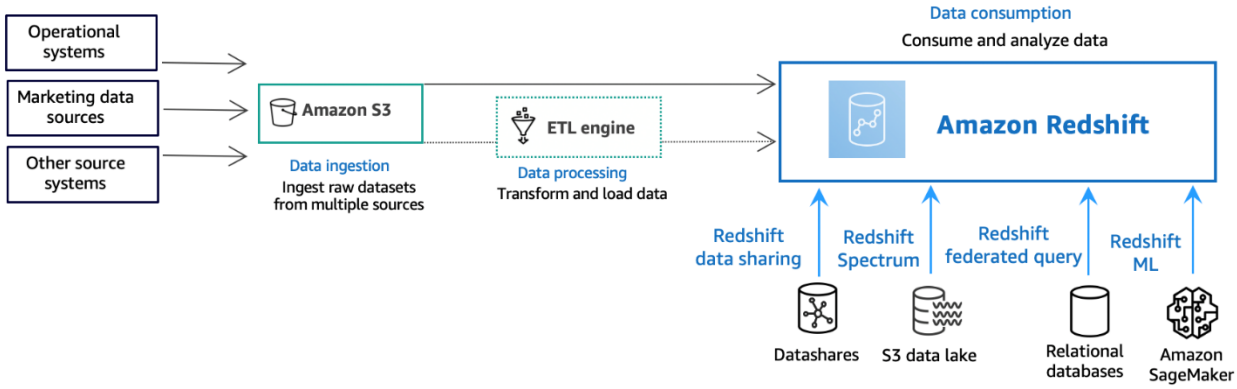

Рисунок 1.6 – Приклад обробки даних за допомогою Amazon Redshift [2]

На етапі отримання дані постійно надходять з різних типів джерел у структурованому, напівструктурованому або неструктурованому форматі. Цей етап виконує функцію зберігання даних у різних станах готовності до використання. Прикладом сховище може бути об’єкт Amazon Simple Storage Service.

Етап обробки даних є необов’язковим. На ньому вхідні дані проходять попередню обробку, перевірки та зазнають перетворень відповідно до створених користувачем ETL (extract, transform, load) або ж ELT (extract, load, transform) процесів. Потім ці набори даних доповнюються за допомогою ще одного ETL процесу. Прикладом ETL сервісу є AWS Glue.

На етапі використання дані завантажуються в кластер Amazon Redshift, де вже виконуються певні аналітичні задачі.

1.2.3 Azure Synapse Analytics

Azure Synapse Analytics – це сервіс для аналітики, який пришвидшує час для аналізу даних у сховищах даних та систем з великим обсягом даних. Цей програмний продукт має власну систему SQL запитів – Synapse SQL, що використовується у корпоративних сховищах даних, рушій Apache Spark, який використовується для роботи із великими обсягами даних, інструмент Data Explorer для аналізу подій та часових рядів, інструменти для інтеграції даних і ETL/ELT, а також глибоку інтеграцію з іншими сервісами хмарної платформи Azure, такими як Power BI, CosmosDB та AzureML [3].

Synapse SQL – це розподілена система запитів для T-SQL, яка має функціонал запитів до сховища даних і візуалізації даних та яка розширює можливості T-SQL для потокової передачі та машинного навчання. Вона підтримує безсерверний режим і режим з виділеними ресурсами хмарної платформи. Безсерверний режим дозволяє масштабуватись при незапланованих або різких навантаженнях. Для прогнозованої вартості та продуктивності краще підходить резервування ресурсів. Потокова передача даних забезпечує надходження даних з різних хмарних джерел у SQL таблиці сховища даних. Інтеграція штучного інтелекту реалізована за допомогою розширення T-SQL.

Synapse SQL використовує масштабовану архітектуру для розподілу навантаження між кількома вузлами. Вузли для обчислення відокремлені від зберігання даних, що дає змогу масштабувати їх незалежно від даних у системі. Synapse SQL має два типи інфраструктури – виділений та безсерверний набори SQL вузлів. У виділеному наборі одиницею масштабування є абстракція обчислювальної потужності, яка має назву «одиниця сховища даних». Для безсерверного набору масштабування виконується автоматично в залежності від необхідних ресурсів для запитів. Через постійне додавання, видалення вузлів або аварійні ситуації топологія адаптується до цих змін та гарантує виділення необхідних ресурсів для успішного завершення запиту.

Synapse SQL використовує Azure Storage сервіси для зберігання та захисту даних користувача. Безсерверний набір вузлів дозволяє лише зчитувати файлі зі сховища даних, тоді як виділений набір – додавати та зчитувати. Коли дані надходять до виділеного набору, вони діляться на одиниці розподілу (англ. «distribution») для оптимізації продуктивності системи. Користувач може вибрати тип розподілу даних при створенні таблиці, які включають в себе – хешування, циклічний розподіл даних, реплікація.

Також варто зазначити, що Synapse SQL має два типи вузлів – вузол керування та вузол обчислення. Єдиною точкою входу є вузол керування, до якого підключаються програмні продукти та надходять T-SQL команди. Для обміну даними між вузлами Synapse SQL має сервіс переміщення даних (англ. «Data Movement Service»), який потрібен для виконання запитів паралельно та повернення точних результатів.

Вузол керування – це інтерфейс, який взаємодіє з усіма програмними продуктами та підключеннями. На цьому типі вузлів працює система розподілених запитів для оптимізації та координації паралельних запитів. Будь-яка T-SQL команда до виділеного набору SQL вузлів перетворюється на паралельні запити. У безсерверному наборі механізм розподіленої обробки запитів (англ. «Distributed Query Processing») оптимізує запити за допомогою поділу на менші частинки, які виконуються на обчислювальних вузлах. Кожна така частинка називається завданням та представляє одиницю розподіленого виконання.

Вузли обчислення зберігають усі дані користувача в Azure Storage та виконують паралельні запити. У виділеному наборі вузлів одиниці розподілу відповідають певним вузлам обчислення для обробки даних. У безсерверному наборі кожному вузлу призначається завдання та набір файлів для його виконання.

Azure Synapse забезпечує інтеграцію даних за допомогою ETL процесів, які можна створити в даному сервісі. Підтримується отримання даних з понад 90 типів джерел даних, серед яких є бази та сховища даних найбільших хмарних платформ (Azure, Google Cloud Platform, Amazon Web Services). Також є підтримка ODBC драйверу, HTTP з’єднання, прикладного програмного інтерфейсу з використанням REST технології. Підтримуються наступні формати даних:

– Avro – має компактний формат, складається з заголовку файлу і одного або більше блоків даних, є незалежним від мови програмування;

– Apache Parquet – формат зберігання даних по стовпцях, який використовується для великих обсягів даних, спеціалізується на ефективному зберігання та обробці вкладених типів даних;

– Apache ORC (Optimized Row Columnar) – формат зберігання даних по стовпцях, який використовується для великих обсягів даних, використовується для читання, запису й обробки даних у Hadoop Hive;

– дельта-формат – використовується для зберігання даних у транзакційний спосіб, дозволяє ефективно оновлювати та видаляти дані, підтримує транзакції та відкати даних, побудований на основі Apache Parquet;

– JSON – легкий популярний формат обміну даних, який зазвичай використовується для передачі даних між сервером та клієнтом;

– XML – мова розмітки, яка використовується для опису та обміну структурованими даними;

– двійковий формат – зберігає дані у двійковому коді (1 і 0), який зазвичай використовується в комп'ютерних системах і є більш ефективним для зберігання та передачі даних;

– формат загальної моделі даних (англ. «Common Data Model») – стандартний формат для організації й опису даних, забезпечує послідовний опис даних у різних системах;

– формат тексту з роздільними знаками – кожне поле в записі відокремлено символом-роздільником (комою або табуляцією), CSV є найпопулярнішим прикладом даного формату.

Висновок до розділу

У розділі було розглянуто різні типи сховищ даних за типом джерела даних та за типом архітектури системи, їхні переваги та недоліки. Вітрини даних є найбільш гнучкими та сфокусовані на певному бізнес напрямку. Корпоративне

сховище даних навпаки має широкий набір даних, зібраних з усіх джерел організації, що дає перевагу при предикативному аналізі та визначенні актуальних потреб бізнесу, але з іншої сторони вартість обслуговування є порівняно високою. Головною особливістю операційного сховища даних, яка відрізняє її від попередніх типів, є доступ до даних у реальному часі, що може бути використано для моніторингу певних показників та швидкого прийняття рішень у критичних ситуаціях.

Якщо порівнювати за типом архітектури системи, то Data Vault є гібридним типом, що поєднує в собі переваги та деякі недоліки моделей Кімбалла та Інмона та який підходить для бізнесу, який швидко збільшує обсяги даних або потреби якого змінюються. Модель Кімбалла може бути ефективно використана в організаціях, де вже визначені бізнес потреби і яким потрібно розробити сховище для великих обсягів даних якнайшвидше. Модель Інмона є хорошим вибором для великих організацій, яким потрібно мати централізоване джерело точних структурованих даних для комплексних запитів. Отже, вибір на користь кожного з типів залежить від ресурсів, які бізнес готовий витратити на розробку та обслуговування сховища даних, та від способу використання даних.

Також у розділі розглянуті наявні аналоги сховищ даних, особливості архітектури їх систем, ключові концепти та інтеграції з іншими сервісами. Визначено, що сховища даних або є частиною хмарних платформ, або використовують їх в якості інфраструктури, але вони не надають користувачу можливості використовувати приватні сервери (локальні або надані хмарною платформою). Можна зробити припущення, що існують сервіси, які надають таку можливість. Іншим виходом з даної ситуації є наявність бібліотек, за допомогою яких користувач може побудувати свою розподілену систему сховища даних та надсилати запити до неї за допомогою власного прикладного програмного інтерфейсу.

2 ОПИС СИСТЕМИ

2.1 Опис процесу діяльності

Для використання платформи управління сховищем даних користувач має зареєструватися та авторизуватися в системі. Для подальшого використання системи користувач має вказувати токен, який він отримав після авторизації.

Спершу користувач має налаштувати власну інфраструктуру, в якості якої можна використати як сервіси від хмарних платформ, так і власні або виділені сервери. Обов’язковою умовою є те, що вона має використовувати технологію Apache Spark для управління запитами. Необхідні дані для доступу до інфраструктури мають бути додані на етапі створення проєкту.

Після цього користувач отримує доступ до управління проєктом, в якому може знаходитись одне і лише одне сховище даних та декілька джерел даних. Платформа надає можливість управляти джерелами даних всередині системи (тобто, видалення джерел даних призводить лише до видалення доступу до певного джерела в системі) та працювати з даними, які вони містять.

Існує декілька можливих сценаріїв використання платформи, які залежать від потреб користувача. Наприклад, інженер даних (англ. «data engineer») буде здебільшого керувати джерелами даних та займатись моделюванням сховища даних для оптимізації роботи з даними. Для аналітика головною функціональністю буде зчитування даних за допомогою комплексних запитів та перегляд наявних таблиць і сховищ даних.

2.2 Опис функціональної моделі

В опис функціональної моделі входить визначення акторів та їх функцій. Акторами системи є користувач, сервер та джерело даних.

Основні функції користувача:

- реєстрація та авторизація;
- управління проєктами;

– управління сховищами даних;

– управління джерелами даних;

– запис та зчитування даних.

Основними функціями серверу є надання доступу до нього та виконання запитів для управління сховищем даних.

Основними функціями джерела даних є надання доступу, та передача даних.

Діаграму варіантів використання для користувача зображено у графічному матеріалі ІС93.130БАК.004 Д1.

Функціональні вимоги до програмного забезпечення:

– система повинна надати користувачу можливість зареєструватись та авторизуватись;

– система повинна надати користувачу можливість створити сховище даних на основі існуючої інфраструктури;

– система повинна надати користувачу можливість створити таблиці сховища даних;

– система повинна надати користувачу можливість наповнити таблиці сховища даних із зовнішніх джерел даних;

– система повинна надати користувачу можливість виконувати SQL запити для отримання даних зі сховища даних;

– система не повинна видаляти дані зі сховища даних.

2.3 Засоби розробки

Для реалізації платформи управління сховищем даних потрібно створити прикладний програмний інтерфейс, бізнес-логіку та вибрати тип бази даних та саму базу даних.

2.3.1 Вибір мови програмування

Найрозповсюдженішими мовами програмування для реалізації прикладного програмного інтерфейсу є JavaScript, Java та Python. Вибір JavaScript часто

пов’язаний зі зручністю використання однієї мови для всього програмного продукту, адже графічний інтерфейс користувача зазвичай створюють саме цією мовою. Цей аргумент на користь JavaScript не є релевантним для даної реалізації.

Стосовно бібліотек для створення розподіленого сховища даних, то основними мовами програмування є лише такі мови як Java, Python та Scala.

Python - це високорівнева мова програмування з динамічною типізацією, яка зазвичай є інтерпретованою (однак, існують компілятори для перетворення програмного коду на байт-код або двійкові виконувані файли). Перевагами цієї мови є відносна простота, популярність (а отже й наявність великої кількості бібліотек) та універсальність, адже Python широко використовується для створення веб-додатків, графічних інтерфейсів, для машинного навчання, аналізу та обробки даних. Недоліком є глобальне блокування інтерпретатора (англ. «Global interpreter lock»), що може призвести до погіршення продуктивності багатопотокових операцій, перешкоджати масштабуванню та в роботі з розподіленою архітектурою системи [4].

Java – розповсюджена об’єктно-орієнтовна мова програмування, яка зазвичай використовується в корпоративних проєктах. Має усталену екосистему, багато бібліотек для розробки та зазвичай використовується для побудови масштабованих надійний систем, в тому числі є гарним вибором для обробки великих обсягів даних. Також варто зазначити, що більша частина технологій та бібліотек з самого початку створено цією мовою, а вже потім адаптовано до інших мов.

Scala – це мова зі статичною типізацією, яка компілюється за допомогою Java Virtual Machine. Має схожий до Java синтаксис, тому іноді використовується як замінник, хоча автори мови визначають її і як функціональну мову програмування (всі функції є значеннями), і як об’єктно-орієнтовну (всі значення є об’єктами), через що часто розробники не використовують всі можливості та переваги мови. Перевагами є підтримка екосистеми Java та простий знайомий синтаксис, за допомогою якого можна побудувати складні масштабовані системи з використанням лише функціонального підходу. Недоліками є висока вартість

розробки та обслуговування через непопулярність мови та невелику кількість авторів бібліотек, які створені виключно цією мовою [5].

При порівнянні Java, Scala та Python, можна зробити висновок, що третя мова має велику кількість бібліотек, в тому числі і для створення прикладного програмного інтерфейсу та управління сховищем даних та потребує істотно менше часу на розробку програмного продукту, що є важливим аргументом при створенні прототипів системи (англ. «Proof of concept»).

2.3.2 Вибір бібліотек

В даному підрозділі наведено перелік та опис бібліотек, які були використані у платформі управління сховищем даних.

FastAPI – сучасна, високопродуктивна бібліотека, яка використовує специфікацію OpenAPI для створення веб-застосунків за допомогою мови програмування Python версії 3.7 та вище з використанням вбудованих анотацій типів. Має вбудовані модулі, серед яких інтерактивна документація Swagger UI, перевірка типів, авторизації OAuth 2.0 з підтримкою JWT токенів, впровадження залежностей (англ. «Dependency injection») та можливість інтегрувати свої модулі. Варто зазначити, що FastAPI побудовано на бібліотеці Starlette [6].

Pydantic – бібліотека для перевірки даних за допомогою анотації типів, для керування налаштуваннями користувача та для автоматичного перетворення даних (наприклад, JSON у певний клас). Надає можливість користуватись вже готовими типами даних та перевірками і можливість створити власні функції для перевірки даних. Також має вбудовану підтримку SQLAlchemy, що полегшує роботу із базою даних [7].

SQLAlchemy – комплексний набір інструментів для роботи з базами даних і Python. Має декілька різних модулів під різні сценарії застосування. Основними компонентами є:

– SQLAlchemy ORM (Object Relational Mapper) – абстрактний шар, за допомогою якого можна створювати таблиці баз даних та виконувати операції з ними;

– SQLAlchemy Core, до якого входять рушій, мова SQL запитів, схеми та типи, діалекти під різні бази даних, інструмент для підключення;

– DBAPI – інтерфейс певної бази даних.

Також SQLAlchemy надає можливість мігрувати між базами даних та має інтеграцію з багатьма бібліотеками для створення прикладного програмного інтерфейсу [8].

Structlog – гнучка бібліотека логування подій для Python. Надає можливість додавати контекст до повідомлень та метадані до записів, використовуючи уніфікований інтерфейс. Має інтеграцію з іншими бібліотеками та підтримку плагінів [9].

PySpark – це API для Python для використання Apache Spark. Головною задачею цієї технології є розподілена обробка даних у реальному часі та аналіз даних в інтерактивному режимі. Основними характеристиками Apache Spark є:

– швидкість, яка є наслідком використання лінивих обчислень та обчислень в оперативній пам’яті;

– масштабованість обчислень та зберігання даних за рахунок збільшення кількості вузлів;

– відмовостійкість забезпечує збереження джерел даних та переобчислення втрачених даних в разі збою;

– гнучкість при виборі мови програмування, адже є підтримка Scala, Java, Python і R;

– розширена аналітика, що включає в себе виконання комплексний SQL-запитів, машинне навчання, обробка потокових даних, графіків тощо.

PySpark має підтримку всіх функцій та компонентів Spark, серед них Spark SQL, DataFrames, Structured Streaming, MLlib, Spark Core, RDD, Spark Streaming. Також є інтеграція з багатьма типами джерела даних, включаючи розподілену файлову систему Hadoop, Apache Hive, сервіси хмарних платформ [10].

Delta Spark – це бібліотека для роботи з Delta Lake, шаром зберігання даних з відкритим вихідним кодом [11]. Ця технологія забезпечує ACID транзакції та масштабованість даних, має спільний інтерфейс для потокової (англ. «streaming processing») та пакетної (англ. «batch processing») передачі даних на базі існуючих джерел даних, серед яких AWS S3, Google Cloud Storage, Azure Data Lake Storage, HDFS. Також Delta Lake має функцію подорожі в часі за допомогою версіонування даних [12].

2.3.3 Вибір бази даних

Основними типами є реляційна та нереляційна (англ. «NoSQL») база даних.

Особливостями реляційної бази даних є структурованість даних у таблиці та наявність відношень між ними. Таблиці складаються з рядків (або ж записів) та стовпців та обов'язково мають первинний ключ для ідентифікації запису. Між таблицями можуть бути такі типи відношень:

– один-до-одного – певний запис таблиці А може бути пов'язаний з 1 та лише 1 записом таблиці Б та навпаки. Прикладом може бути відношення між сутністю «країна» та сутністю «столиця» - кожна країна має лише одну столицю та кожна столиця є такою лише для однієї країни;

– один-до-багатьох – запис таблиці А може бути пов'язаним з декількома записами таблиці Б, але не навпаки. Наприклад, один клієнт може мати декілька замовлень, але замовлення не може мати декілька клієнтів;

– багато-до-багатьох – декілька записів таблиці А можуть бути пов'язані з декількома записами таблиці Б і навпаки. Прикладом може слугувати відношення між сутністю «студент» та сутністю «навчальний предмет». Студент може записатись на декілька навчальних предметів, а кожен навчальний предмет може мати декілька студентів. Щоб створити відношення цього типу потрібно мати проміжну таблицю, в якій буде вказано унікальні ідентифікатори обох сутностей.

Також важливою особливістю реляційних баз даних є наявність ACID транзакцій, що включає в себе:

– атомарність (англ. «Atomicity») – транзакція розглядається як єдина неподільна одиниця. Це гарантує, що всі операції в рамках транзакції будуть виконані повністю або повністю відкотяться назад при помилці;

– узгодженість (англ. «Consistency») – до виконання та після виконання транзакції база даних перебуває в узгодженому стані, завдяки чому забезпечується виконання правил та обмежені і зберігається цілісність даних;

– ізольованість (англ. «Isolation») – при одночасному виконанні транзакцій, вони не впливають одна на одну, бо кожна є ізольованою від всіх інших. Це запобігає пошкодженню даних або порушенням правил;

– довговічність (англ. «Durability») – після виконання транзакції її зміни обов'язково є постійними та невразливими до можливих майбутніх збоїв у системи.

Особливостями нереляційної бази даних є зберігання частково структурованих (із схемою, яка може змінюватись) або неструктурованих (без визначеної схеми) даних, відсутність повної підтримки ACID транзакцій та відношень між таблицями. Перевагами такого підходу є швидкість роботи з даними, можливість горизонтального масштабування системи для збереження великих обсягів даних. Також варто зазначити, що в разі роботи з великими обсягами даних у реляційній базі, синхронізація всіх даних є умовою успішного завершення транзакції. Також відсутність відношень між таблицями пришвидшує додавання, видалення та оновлення записів.

Для платформи управління сховищем даних підходить реляційна база даних, адже немає необхідності зберігати великі обсяги даних для роботи системи та є потреба у наявності відношень між таблицями. Найпоширенішою реляційною базою даних є PostgreSQL. Важливою перевагою є те, що вона підтримує JSON тип даних, який потрібен для збереження налаштувань підключення до джерел даних та вузлів сховища, які мають різний набір параметрів.

2.3.4 Вибір середовища розробки

Для розробки програмного забезпечення було вибрано Visual Studio Code як основне середовище розробки, DBeaver для роботи з базою даних та Postman для тестування.

Visual Studio Code – це середовище розробки, яке доступне для операційних систем Windows, macOS та Linux. Має багато вбудованих плагінів та екосистему з розширеннями для багатьох мов програмування [13].

DBeaver – це безкоштовний інструмент з відкритим кодом для роботи з базами даних. Надає можливість працювати з різними базами даних паралельно за допомогою уніфікованого інтерфейсу. Також має багато функцій для адміністрування баз даних, створення Entity Relationship діаграм та імпорту, експорту даних [14].

Postman – інструмент для розробки та тестування прикладного програмного інтерфейсу, який спрощує процес створення, документування та тестування. Надає інтерфейс для створення та відправки HTTP запитів, набір інструментів для автоматичного тестування і моніторингу та середовище для взаємодії між декількома користувачами [15].

2.4 Інформаційне забезпечення

2.4.1 Вхідні дані

Платформа управління сховищем даних має прикладний програмний інтерфейс з кінцевими точками та відповідними вхідними даними. Вхідні дані надсилаються у вигляді HTTP запиту. Він складається з:

- назви метод;
- шляху запиту;
- версії протоколу HTTP;
- заголовку, в якому вказано додаткову інформацію для серверу;

– тіла запиту, яке є необов’язковим;

– параметрів запиту, які є необов’язковими.

Вхідними даними для авторизація та реєстрації є електронна пошта та пароль. Обов’язковими вхідними даними для всіх інших кінцевих точок є токен з даними про користувача.

Управління проєкту включає в себе створення, зчитування всіх проєктів користувача або одного конкретного проєкту, оновлення даних та видалення проєкту. Для створення користувачу необхідно надати ім’я проєкту та URL адресу для підключення до інфраструктури сховища даних. Для зчитування та видалення конкретного проєкту потрібен його унікальний ідентифікатор. Для оновлення даних потрібно надати унікальний ідентифікатор та нове ім’я проєкту або нову URL адресу інфраструктури. Також варто зазначити, що проєкт може включати в себе лише одне сховище даних та декілька джерел даних, для отримання відомостей про них потрібно надіслати HTTP запит з унікальним ідентифікатором проєкту.

Управління сховищем даних включає в себе створення, отримання списку всіх сховищ даних користувача, отримання конкретного сховища, оновлення даних та видалення. Для створення необхідно надати унікальний ідентифікатор проєкту, до якого належатиме сховище, назву сховища та список унікальних ідентифікаторів таблиць з джерел даних. Для отримання списку всіх сховищ даних користувача, необхідно надати токен, який містить інформацію про користувача. Для отримання даних про певне сховище даних або його видалення, потрібно вказати його унікальний ідентифікатор. У сховищі даних доступно для оновлення ім’я та перелік таблиць.

Управління джерелом даних включає в себе створення, отримання списку всіх джерел даних користувача, отримання доступних типів джерела даних, отримання конкретного джерела, оновлення та видалення. Для створення необхідно надати унікальний ідентифікатор проєкту, назву джерела, його тип та відомості для підключення до джерела даних. Для всіх інших дій потрібно надати токен, який містить інформацію про користувача та унікальних

ідентифікатор джерела даних. Користувач може оновлювати назву, тип та налаштування джерела даних.

Для створення запиту до сховища даних, користувач має надати SQL запит та унікальний ідентифікатор проєкту.

2.4.2 Вихідні дані

Результатом роботи платформи є розподілене сховище даних, наповнене даними з різних джерел. Вихідними даними системи є HTTP відповідь та певна трансформація сховища даних – додавання нових даних через нові джерела, зчитування даних, створення нових таблиць.

Кожен запит користувача повертає певний HTTP статус та повідомлення, в якому знаходиться набір даних, пояснення помилки або інформація про певні зміни в системі. Існує обмежена кількість HTTP статусів, котрі мають короткий опис. Їх можна поділити на 5 наступних категорій:

– інформаційні (100-199), які повідомляють про отримання запиту та про проміжні результати процесу обробки запиту;

– успішні (200-299), які вказують на успішне виконання запиту сервером та можуть містити інформацію із результатом запиту;

– перенаправлення (300-399), які вказують на виконання додаткових запитів або звернення до переміщених ресурсів;

– клієнтські помилки (400-499), які містять інформацію про те, що запит не було виконано через помилки клієнта;

– серверні помилки (500-599), які повідомляють про проблему з сервером.

Також внаслідок запитів може змінюватись база даних або сховище даних. Користувач може створювати, оновлювати, видаляти сутності в базі даних та записувати або зчитувати дані зі сховища даних.

2.4.3 Структура бази даних

Платформа управління сховищем даних складається з наступних сутностей: Користувач, Проєкт, Сховище даних, Джерело даних, Таблиця, Запит. Кожен з них, окрім Запиту, має відповідну таблицю в базі даних, між якими існують наступні відношення:

– один-до-одного: Проєкт – Сховище даних;

– один-до-багатьох: Користувач – Проєкт, Проєкт – Джерело даних, Джерело даних – Таблиця, Сховище даних – Таблиця.

Entity Relationship діаграму бази даних зображено на рисунку 2.1 та у графічному матеріалі ІС93.130БАК.004 Д2.

Рисунок 2.1 – Діаграма бази даних

На діаграмі можна побачити, що такі сутності як Джерело даних (datasources) та Таблиця сховища даних (warehouseDatatable) мають певний тип. Ці типи визначено за допомогою перелічуваного типу даних (англ. «Enumerated type») – типу даних, що складається з множини іменованих елементів та відповідних значень. Тип Джерела даних потрібен для перевірки підключення до джерела та є одним з наступних значень: «mysql», «postgresql», «mongodb», та «datatable». Типи

Таблиці сховища даних обмежуються моделлю Кімбалла і можуть бути або «fact», або «dimension».

Висновок до розділу

В даному розділі описано процес діяльності та функціональна модель, визначено акторів та варіанти використання. Основними варіантами використання є управління певними сутностями (додавання, модифікація та видалення) та керування розподіленим сховищем даних. Також обґрунтовано вибір мови програмування Python та інших засобів для реалізації. Python часто використовується для створення прикладного програмного інтерфейсу та для роботи з даними, тому має багато бібліотек для цих задач, які постійно оновлюються.

Окрім цього, визначено інформаційне забезпечення – вхідні та вихідні дані та структура бази даних. Вхідними даними є HTTP запити від користувача, які модифікують стан системи або надають певні дані зі сховища даних. Вихідними даними є HTTP відповідь та/або зміна в базі даних і сховищі даних. Структура бази даних представлена у вигляді Entity Relationship діаграми.

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Змістовна постановка задачі

Коли компанія починає зберігати великі обсяги даних, вона стикається з необхідністю працювати з Big Data технологіями, для яких притаманні наступні проблеми:

– обсяг даних, з яким не можуть ефективно працювати традиційні бази даних;

– різноманітність форматів даних, які поділяються на 3 категорії – структуровані (наявні в реляційних базах даних), частково структуровані (існує формат даних, який змінюється) та неструктуровані (текст, зображення, відео);

– швидкість оновлення даних, яка може бути непостійною. Також іноді виникає потреба в отриманні даних в реальному часі;

– якість та цілісність даних. Складність виникає внаслідок великого обсягу та різноманітності форматів даних.

Таким чином, виникає потреба в сховищі даних, яке є централізованим джерелом всіх даних компанії. Платформа управління сховищем даних полегшує роботу з ним шляхом автоматизації збору даних із декількох джерел у сховище даних та автоматизації створення сховища даних.

3.2 Математична постановка задачі

Платформа управління сховищем даних складається з множини джерел даних, множини сховищ даних, множини таблиць та множини запитів. На цих множинах визначені такі операції як створення, проведення запитів та додавання нових елементів.

3.3 Обґрунтування методу розв’язання

Кінцевими користувачами платформи управління сховищем даних є інженери даних та розробники програмного забезпечення. Тому в якості

інтерфейсу був вибраний прикладний програмний інтерфейс, адже на основі нього кінцевий користувач може створити декілька сервісів в залежності від варіантів використання.

REST API було вибрано як архітектуру прикладного програмного інтерфейсу тому, що вона є стандартизованою (HTTP методи та коди стану) та підтримується всіма браузерами за замовчуванням. Також вона є простішою та більш розповсюдженою в порівнянні з RPC або GraphQL підходами, що впливає на зручність використання інтерфейсу кінцевими користувачами.

Також варто зазначити, що в якості моделі для сховища даних було вибрано модель Кімбалла. Архітектура цієї моделі є більш простою в порівнянні з Data Vault або моделлю Інмона та одночасно надає користувачу можливість ітеративно змінювати сховище даних.

Окрім цього, було визначено, що користувач не може повністю видаляти дані зі сховища даних, а лише створювати нові версії всіх наборів даних. Це зроблено для можливості аналізу історичних даних. Тобто, після видалення всі дані, які мають залишитись, отримують оновлення версії, за рахунок чого користувач не може отримати видалені дані без явного вказання старішої версії. Аналогічний процес відбувається із оновленням даних. Таким чином користувач може переглядати зміни, які відбулись у таблиці, та використовувати це для аналізу або у ETL/ELT процесах.

3.4 Опис методу розв’язання

Спочатку потрібно визначити сутності системи – це сховище даних, джерело даних, користувач і таблиця. Оскільки користувач може мати декілька сховищ даних, кожне з яких буде мати одне та більше джерел даних та таблиць, то виникає потреба виділити ще одну сутність для структуризації навколо сховища даних – проєкт.

Іншою важливою складовою є типи джерел даних, які мають бути інтегровані з системою. Було вибрано такі розповсюджені джерела даних, як-от MySQL, PostgreSQL, MongoDB, та файли формату CSV або JSON.

В якості бази даних для роботи системи було вибрано PostgreSQL. Для роботи із сховищем даних було вибрано бібліотеку Apache Spark, за допомогою якої можна керувати таблицями, надсилати запити та завантажувати або записувати дані з різних джерел напряму.

3.5 Архітектура програмного забезпечення

Платформа управління сховищем даних складається з 3 рівнів:

– рівню маршрутизаторів або кінцевих точок прикладного програмного інтерфейсу, який приймає запити від користувача за допомогою HTTP протоколу та містить виклики сервісів;

– рівню сервісів, які інкапсулюють бізнес логіку, взаємодію з базою даних та зовнішніми сервісами (джерелами даних, сховищами даних);

– рівню бази даних, який зберігає та оброблює внутрішні дані системи за допомогою об’єктно-реляційного відображення (англ. «Object-relational mapping», «ORM»).

На рисунку 3.1 зображена структурна схема платформи управління сховищем даних.

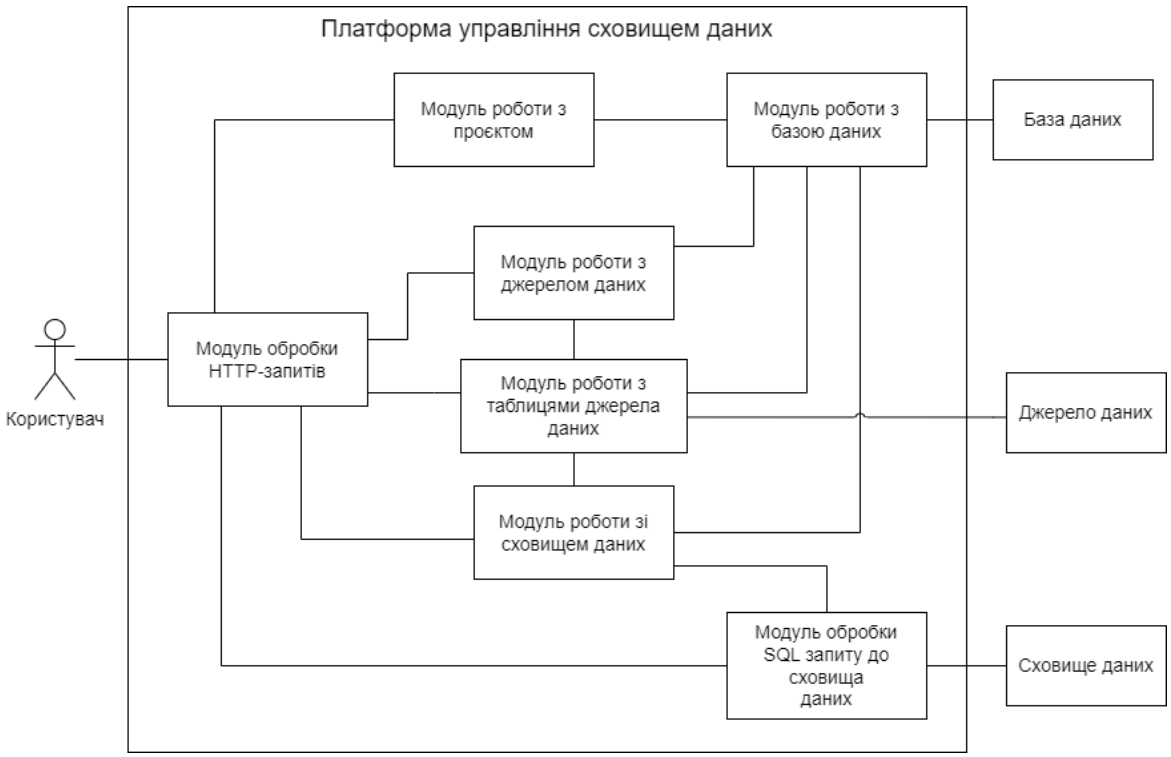

Рисунок 3.1 – Структурна схема

На схемі зображено наступні модулі:

– модуль обробки HTTP-запитів, який містить прикладний програмний інтерфейс та ізолює бізнес-логіку;

– модуль роботи з базою даних, який містить моделі таблиць баз даних та операції з ними;

– модуль роботи з проєктом, який об’єднує одне сховище даних та декілька джерел даних;

– модуль роботи з джерелом даних, який містить стандартні операції для управління джерелом даних та взаємодіє з модулем роботи з таблицями джерела даних;

– модуль роботи зі сховищем даних, який містить стандартні операції для управління сховищем даних, для чого взаємодіє з модулем роботи з таблицями джерела даних та модулем обробки SQL-запитів;

– модуль роботи з таблицями джерела даних збирає відомості про необхідні таблиці джерела та зчитує дані з цих таблиць;

– модуль обробки SQL-запиту до сховища даних обробляє та надсилає запити до сховища даних та обробляє отримані дані.

3.5.1 Діаграма компонентів

Платформа управління сховищем даних складається з наступних компонент:

– Token – необхідний для автентифікації та авторизації, для отримання користувач має вказати певні дані про себе (електронну пошту та пароль);

– Query service – компонент, який надає користувачу можливість виконувати запити та внаслідок цього отримувати певні дані;

– Database – зберігає інформацію для коректної роботи системи;

– Project – містить сховище даних та джерела даних;

– Data warehouse – сховище даних, з якого можна отримувати дані за допомогою запитів;

– Data source – джерело даних, з якого дані копіюються у сховище даних;

– Table – складова джерела даних, яка копіюється до сховища даних.

Діаграму компонентів зображено на рисунку 3.2.

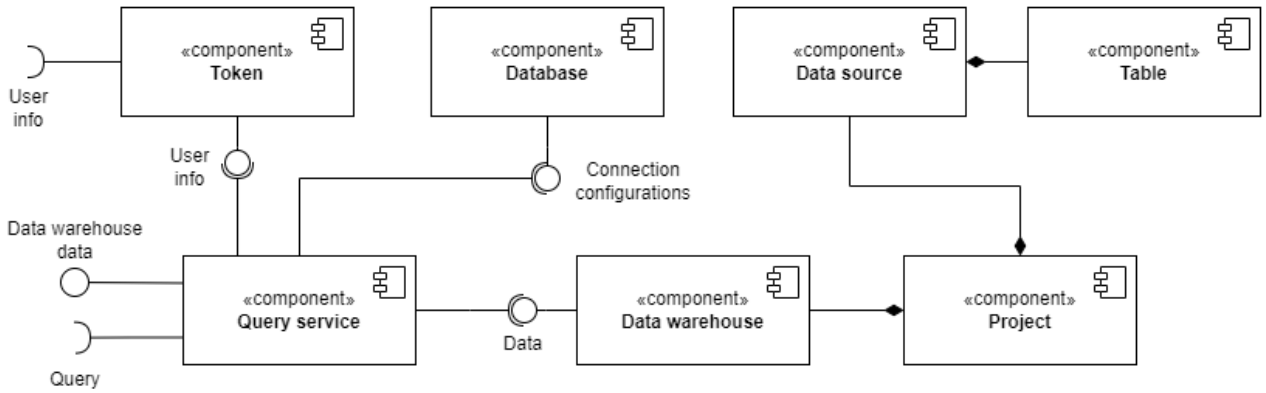

Рисунок 3.2 – Діаграма компонентів

На діаграмі компонентів зображено необхідні зовнішні дані для роботи платформи управління сховищем даних – інформація про користувача та запит.

3.5.2 Діаграми послідовності

Платформа управління сховищем даних має багато ідентичних процесів для всіх типів об’єктів, як-от створення, перегляд всіх або конкретного об’єкту, оновлення та видалення. Тому в даному розділі буде описано процеси лише для одного типу.

Спершу користувач має зареєструватись та авторизуватись у системі. Ці процеси є ідентичними, тому на рисунку 3.3 зображено лише один з них – авторизацію.

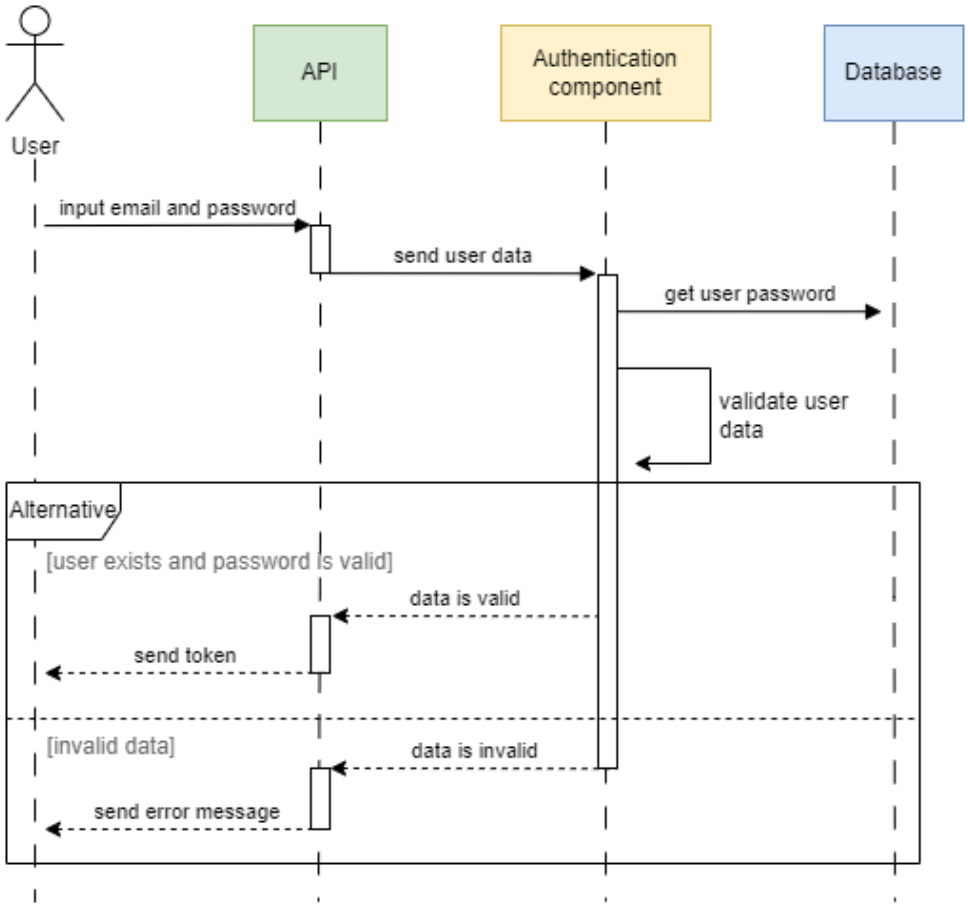

Рисунок. 3.3 – Діаграма послідовності для процесу авторизації

Для створення певного об’єкту, користувачу потрібно мати доступ до пов’язаних об’єктів та вказати необхідні дані у тілі HTTP запиту. У графічному матеріалі ІС93.130БАК.004 ДЗ зображено діаграму послідовностей для процесу створення сховища даних.

Для отримання переліку всіх об’єктів певного типу, до яких користувач має доступ, йому необхідно надати токен, який можна отримати після авторизації. Процес отримання переліку всіх проєктів користувача зображено на рисунку 3.4.

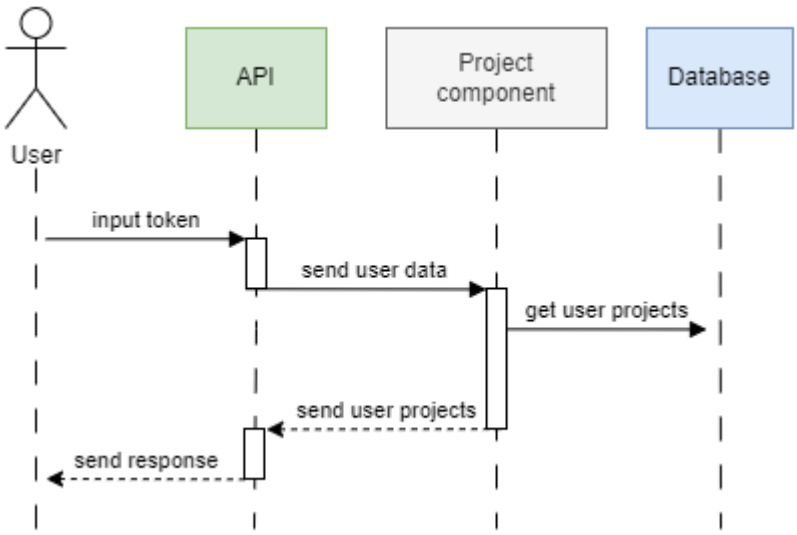

Рисунок 3.4 – Діаграма послідовності для процесу отримання всіх проєктів користувача

Для отримання даних про певний об’єкт користувач має мати доступ до нього, вказати унікальний ідентифікатор об’єкту та токен. Діаграму послідовності для процесу отримання даних про певне джерело даних зображено на рисунку 3.5.

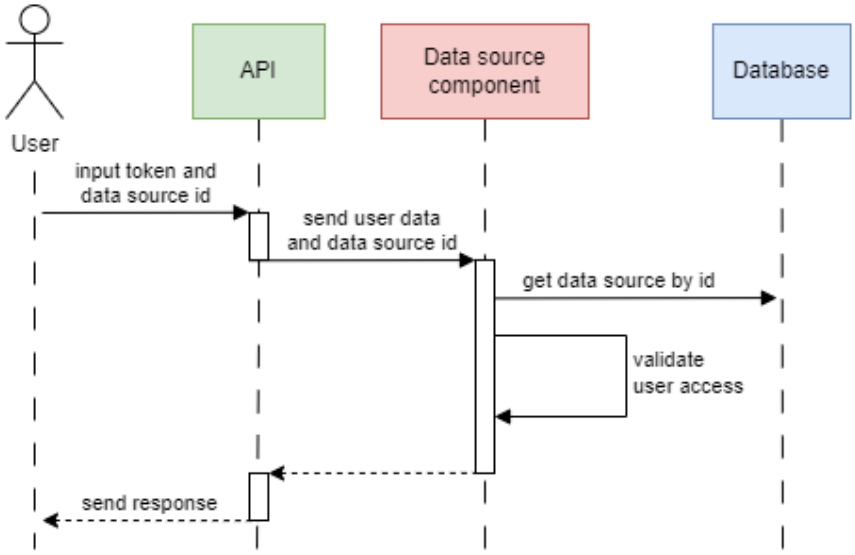

Рисунок 3.5 – Діаграма послідовності для процесу отримання даних про певне джерело даних

Для оновлення даних про об’єкт потрібно мати доступ до нього, вказати унікальний ідентифікатор, дані, які мають бути оновлені та токен. Діаграму послідовності для процесу оновлення даних про сховище даних зображено на рисунку 3.6.

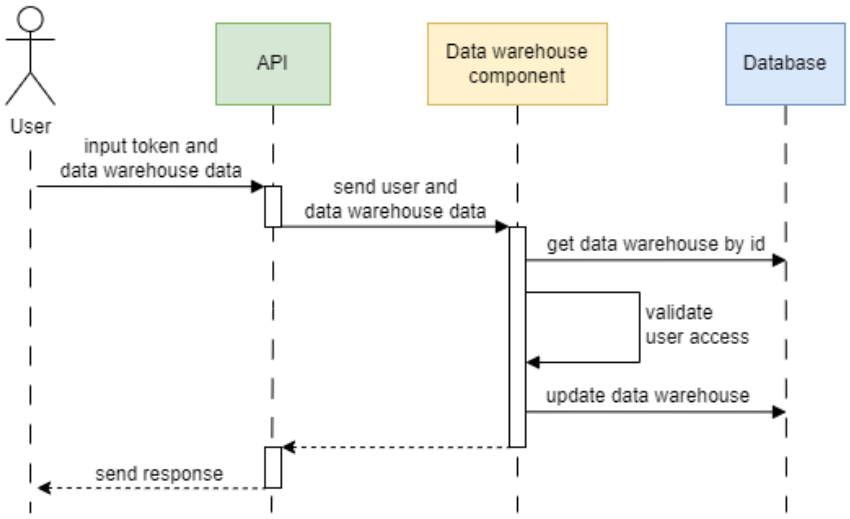

Рисунок 3.6 – Діаграма послідовності для процесу оновлення даних про сховище даних

Для видалення об’єкту користувач має мати доступ до нього, надати токен та унікальний ідентифікатор об’єкту. На рисунку 3.7 зображено діаграму послідовностей для видалення проєкту.

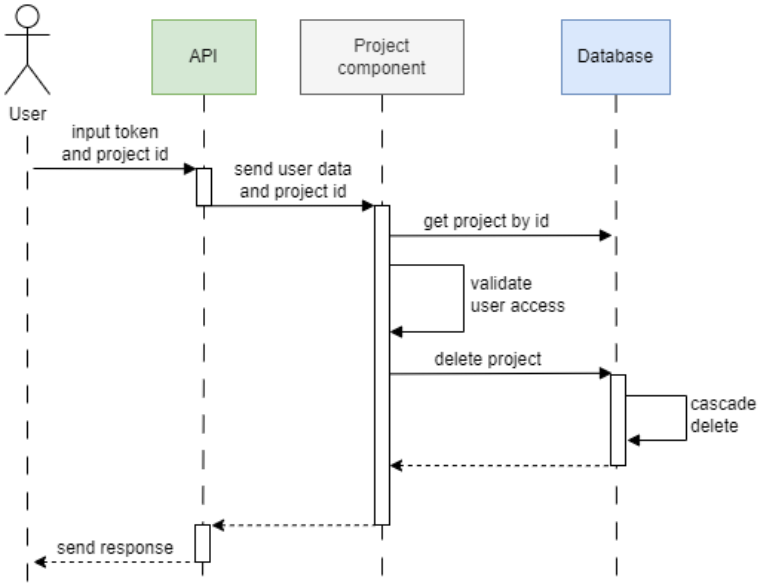

Рисунок 3.7 – Діаграма послідовності для процесу видалення проєкту

Для зчитування даних користувачу потрібно вказати унікальний ідентифікатор проєкту, мати доступ до нього, надати токен і SQL запит. Діаграму послідовності для процесу зчитування даних зображено на рисунку 3.8.

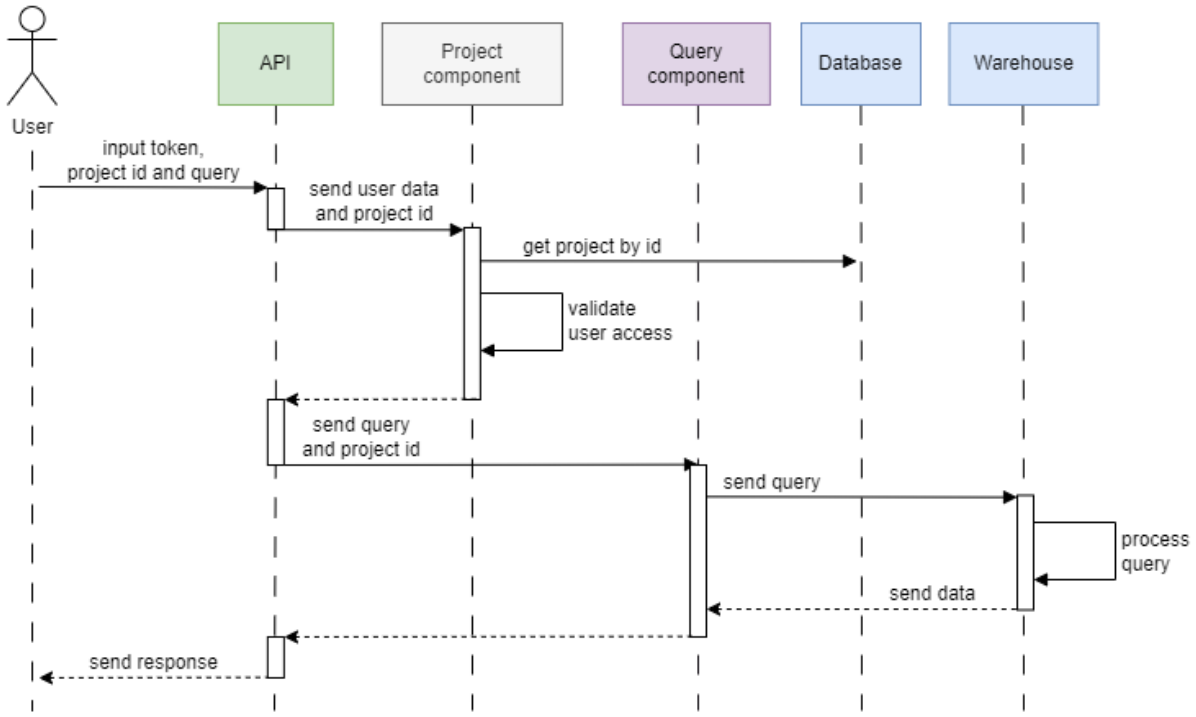

Рисунок 3.8 – Діаграма послідовності для зчитування даних

Для запису даних користувачу потрібно вказати унікальний ідентифікатор проєкту, мати доступ до нього, надати токен і файл із даними. Діаграму послідовності для процесу запису даних зображено в графічному матеріалі ІС93.130БАК.004 Д4.

3.5.3 Опис класів

В таблиці 3.1 розглянуті основні класи, які потрібні для роботи платформи управління сховищем даних.

Таблиця 3.1 – Опис класів

Назва	Опис
ProjectService	Містить запити до бази даних та бізнес логіку створення, оновлення, отримання переліку та видалення об’єктів проєкту. Також містить логіку отримання пов’язаних об’єктів - сховища даних та джерел даних.
WarehouseService	Містить запити до бази даних та бізнес логіку створення, оновлення, отримання переліку та видалення об’єктів сховища даних. Також містить логіку додавання таблиць з існуючих джерел даних.
DatasourceService	Містить запити до бази даних та бізнес логіку створення, оновлення, отримання переліку та видалення об’єктів джерела даних. Також містить логіку оновлення таблиць, які пов’язані з джерелом даних.
QueryService	Містить перевірку запиту, виконання запиту, логіку зчитування та запису даних та обробку результатів.
DatasourceCreate	Містить модель джерела даних для створення та перевірку налаштувань для кожного типу джерела даних.
DatasourceUpdate	Містить модель джерела даних для оновлення таких полів як назва, тип та налаштування.

Висновок до розділу

В даному розділі було проведено дослідження і опис платформи управління сховищем даних. У змістовній постановці задачі сформульовано проблематику, у математичній постановці задачі – опис множин та операцій на них. Обґрунтування

методу розв’язання та опис методу розв’язання містять детальні пояснення вибору певного підходу для реалізації платформи управління сховищем даних. Для реалізації програмного продукту було вибрано прикладний програмний інтерфейс.

Архітектура програмного забезпечення була розроблена з урахуванням функціональної моделі й функціональних вимог та описана за допомогою структурної схеми, що пояснює взаємодію між різними модулями, та діаграми компонентів, яка складається з об’єктів та зв’язків між ними в системі. Також наведені діаграми послідовностей, які демонструють взаємодію між об’єктами, з поділом на різні сценарії та впорядковані за часом.

Розділ завершується описом основних класів, які потрібні для роботи програмного продукту.

Загалом, цей розділ містить огляд та детальний опис ключових етапів підготовки для реалізації платформи управління сховищем даних, в тому числі формулювання задачі, обґрунтування методу розв’язання, розгляд архітектури програмного продукту і опис класів. Це забезпечує чітке розуміння задачі програмного продукту та його структури, що є важливим під час створення прикладного програмного інтерфейсу.

4 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

4.1 Керівництво користувача

4.1.1 Початок роботи

Для використання платформи управління сховищем даних користувачу необхідно встановити застосунок для контейнеризації Docker Desktop та запустити docker-compose.yaml, який створить збірку контейнерів з прикладним програмним інтерфейсом та базою даних.

Іншим варіантом є використання хмарних сервісів для бази даних Postgres і локальне розгортання прикладного програмного інтерфейсу за допомогою інструменту uvicorn. Для цього користувачу необхідно мати встановлену версію Python 3.10 або вище, віртуальне середовище з встановленими пакетами та задати змінні середовища, які також вказані у файлі docker-compose.yaml.

Користувач має 2 варіанти використання платформи управління сховищем даних:

– використання сторінки інтерактивної документації Swagger;

– використання сторонніх додатків, таких як Postman, для взаємодії з прикладним програмним інтерфейсом.

Для першого варіанту користувачу необхідно отримати доступ до прикладного програмного інтерфейсу та зайти на сторінку інтерактивної документації, яка зазвичай знаходиться за URL-адресою «hostname/docs». На цій сторінці користувач може взаємодіяти з усіма кінцевими точками.

Для другого варіанту користувачу необхідно отримати доступ до прикладного програмного інтерфейсу та ознайомитись з переліком доступних кінцевих точок. Після цього користувач може взаємодіяти з ними за допомогою стороннього додатку.

Також користувач має створити власну інфраструктуру, яка буде мати Spark-кластер для управління та забезпечити можливість виконання зовнішніх запитів. Наприклад, можна використати сервіс Azure Databricks для створення кластеру. Користувачу необхідно зареєструватися на порталі Azure та знайти Azure

Databricks серед доступних сервісів в категорії “Analytics”. Далі потрібно ввести ім’я середовища для кластеру, вибрати доступну Azure підписку, вибрати або створити ресурсну групу, зазначити регіон та цінову категорію кластеру. Рекомендований регіон – West Europe, адже він дає можливість користуватись всіма сервісами Azure. Рекомендована цінова категорія – Premium, адже вона надає можливість виконувати зовнішні запити до сховища даних. Окрім цього, потрібно створити ще декілька ресурсів для отримання можливості виконувати зовнішні запити до кластеру. Детальніше з цим можна ознайомитись в офіційній документації до Azure Databricks [16].

4.1.2 Опис основних сценаріїв роботи

Подальші інструкції будуть мати детальний опис кінцевої точки та HTTP-запит та приклад відповіді.

Для початку користувачу потрібно зареєструватись у системі та отримати токен у відповіді. Для цього потрібно надати пошту та пароль. Пошта має бути унікальна, тобто кожен користувач може мати лише один акаунт. Пароль має відповідати наступним вимогам:

- складатись з мінімум 8 символів;
- не мати пробілів;
- мати хоча б 1 велику літеру;
- мати хоча б 1 цифру;
- не мати спеціальних символів (!@#$%^&*()-+?_=,<>/).

HTTP-запит:

POST /auth/sign_up HTTP/1.1

Host: {{base-url}}

Content-Length: 67

{ "email": "test@somedomain.com", "password": "testPwds3" }

Відповідь для правильного запиту:

201 Created

{

"details": "New user created. Password is valid.",

"token": "{\"access_token\": \"mytoken \", \"token_type\": \"bearer\"}"

}

Відповідь для неправильного запиту:

400 Bad Request

{ "details": "User with this email has already exist. Password must: contain an uppercase letter, contain a number." }

Для продовження роботи з платформою користувачу потрібно авторизуватись, щоб отримати токен. Необхідно надати пошту та пароль, які були використані для реєстрації. HTTP-запит:

POST /auth/login HTTP/1.1

Host: {{base-url}}

Content-Length: 250

Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

------WebKitFormBoundary7MA4YWxkTrZu0gW

Content-Disposition: form-data; name="username"

test@somedomain.com

------WebKitFormBoundary7MA4YWxkTrZu0gW

Content-Disposition: form-data; name="password"

testPwds3

------WebKitFormBoundary7MA4YWxkTrZu0gW—

Відповідь для правильного запиту:

200 OK

{

"details": "User verified",

"token": "{\"access_token\": \"mytoken\", \"token_type\": \"bearer\"}"

}

Відповідь для неправильного запиту:

401 Unauthorized

{ "details": "Value is not a valid password" }

Для початку роботи зі сховищем даних користувачу потрібно створити проєкт. Необхідно надати ім’я проєкту та посилання для підключення до сховища даних і вказати токен у заголовку. HTTP-запит:

POST /projects HTTP/1.1

Host: {{base-url}}

Content-Type: application/json

Authorization: Bearer mytoken

Content-Length: 79

{ "name": "user_project_1", "node_url": "datawarehouse.domain.com" }

Відповідь для правильного запиту:

200 OK

{"details": "Project(id=147c3075-2a38-4cc9-b368-37e605d02354) created"}

Наступним кроком є створення двох джерел даних. Щоб створити джерело даних необхідно надати ім’я, унікальний ідентифікатор проєкту, тип джерела даних та налаштування для підключення. Унікальний ідентифікатор проєкту можна отримати з відповіді на запит про створення або надіслати запит для отримання всіх проєктів, які доступні користувачу. HTTP-запит для отримання переліку проєктів:

GET /projects HTTP/1.1

Host: {{base-url}}

Authorization: Bearer mytoken

Для отримання доступних типів джерела даних можна надіслати наступний запит:

GET /datasources/types HTTP/1.1

Host: localhost:8000

Authorization: Bearer mytoken

Отримавши необхідні дані, можна надіслати запит для створення джерела даних. Також варто зазначити, що поле з налаштуваннями підключення даних має відповідати певним вимогам.

Для поля з налаштуванням підключення бази даних є наступні вимоги:

- має містити поле “host”;
- поле “host” має мати формат “ <host_url>:<port>/<databaseName>”;
- має містити поле “username”;
- має містити поле “password”;
- має містити назви таблиць, перелічені через кому, у полі “tables”.

```
POST /datasources HTTP/1.1
Host: {{base-url}}
Content-Length: 311
{
   "name": "datasource_1",
   "project_id": "147c3075-2a38-4cc9-b368-37e605d02354",
   "ds_type": "postgresql",
   "config": {
      "host": "localhost:80/test_db",
      "username": "postgres",
      "password": "mypassword",
      "tables": "p_table2"
   }
}
```

Відповідь для правильного запиту:

```
200 OK
{“details": "Datasource(id=d10ef85f-3d20-48cc-b16a-bfffc01a46f7) created, tables added"}
```

Для створення сховища даних необхідно надати ім’я, перелік унікальних ідентифікаторів таблиць із зазначенням типу таблиці та унікальний ідентифікатор проєкту. Унікальний ідентифікатор проєкту можна отримати з переліку проєктів, HTTP-запит було наведено вище. Унікальні ідентифікатори таблиць можна отримати з HTTP-запиту для отримання таблиць конкретного джерела даних. HTTP-запит для отримання переліку таблиць:

GET /datatables/f2b7f3bb-af9f-46af-8059-dda3f11cbf0e HTTP/1.1

Host: {{base-url}}

Authorization: Bearer mytoken

Отримавши необхідні дані, можна надіслати запит для створення сховища даних:

POST /warehouses HTTP/1.1

Host: localhost:8000

Content-Type: application/json

Authorization: Bearer mytoken

Content-Length: 282

```
{
   "name": "datasource_1",
   "datatables": {
      "fact": ["4c6ab50b-80ed-405b-850d-e706a7732430"],
      "dimension": ["32fbdabd-9dfb-44b7-9634-cd083da1b43b", "a97b538a-a3d3-4b77-b7b7-baecc72815d7"]
   },
   "project_id": "147c3075-2a38-4cc9-b368-37e605d02354"
}
```

Відповідь для правильного запиту:

200 OK

```
{"details": "Warehouse(id=8dc28701-01bb-4afb-929f-d61cea1065f1) created"}
```

Приклад створених таблиць у Spark кластері наведено на рисунку 4.1.

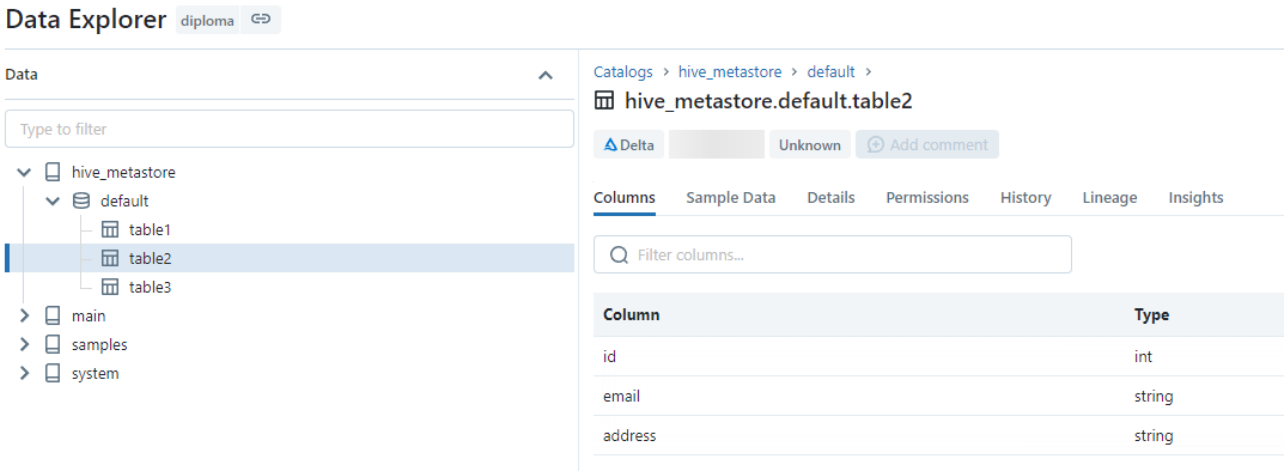

Рисунок 4.1 – Перелік створених таблиць у Spark кластері

Тепер можна переглянути відомості про проєкт та побачити всі його складові. HTTP-запит для отримання відомостей про проєкт:

GET /projects/95c7863d-c211-4d81-adf1-c7d2862c0fe5 HTTP/1.1

Host: {{base-url}}

Відповідь на запит:

200 OK

{

"details": {

"id": "95c7863d-c211-4d81-adf1-c7d2862c0fe5",

"name": "project_1",

"node_url": " datawarehouse.domain.com ",

"warehouse": {

"name": "warehouse_1",

"datatables": {"fact": [], "dimension": ["0ca72be6-133c-4872-b64d-bf64a7cf536d"]},

"project_id": "95c7863d-c211-4d81-adf1-c7d2862c0fe5",

"id": "ef3863a1-a521-465f-8230-0f39a6d9909c"

},

"datasources": [

{"name": "datasource_3",

"project_id": "95c7863d-c211-4d81-adf1-c7d2862c0fe5",
"ds_type": "mongodb",
"config": {"host": "localhost:27017/test_db", "port": "27017", "tables": "mg_table2,mg_table3", "password": "password", "username": "admin"},
"id": "5f9bbeb4-4b42-4c26-87b5-da0f5e6ae730"},
{"name": "datasource_2",
"project_id": "95c7863d-c211-4d81-adf1-c7d2862c0fe5",
"ds_type": "mysql",
"config": {"host": "localhost:3306/test_db", "port": "3306", "tables": "m_table1", "password": "mypassword", "username": "test_user"},
"id": "b1f789dd-c093-4e47-b4e0-3501659f6de1"},
{"name": "datasource_1",
"project_id": "95c7863d-c211-4d81-adf1-c7d2862c0fe5",
"ds_type": "postgresql",
"config": {"host": "localhost:80/test_db", "port": "80", "tables": "p_table2", "password": "mypassword", "username": "postgres"},
"id": "269538c3-03c7-44b3-a5c6-9c77f1508249"}
]
}
}

Далі користувач може надіслати будь-який SQL-запит в кластер. Для цього необхідно надати унікальний ідентифікатор проєкту та сам запит. HTTP-запит для виконання будь-якого SQL-запиту:

POST /query HTTP/1.1

Host: {{base-url}}

Content-Length: 94

{ "project_id": "95c7863d-c211-4d81-adf1-c7d2862c0fe5", "query": "SELECT 'test';"}

Відповідь на запит:

200 OK

{ "details": ["{'test': 'test'}"] }

Для зчитування даних користувачу також необхідно вказати унікальний ідентифікатор проєкту та SQL-запит. HTTP-запит для зчитування даних:

{ "project_id": "95c7863d-c211-4d81-adf1-c7d2862c0fe5", "query": "SELECT * from mg_table3;"}

Відповідь на запит:

200 OK

{

"details": [

"{'id': 1, 'name': 'GTY Technology Holdings, Inc.', 'amount': 55.400001525878906}",

"{'id': 2, 'name': 'Varonis Systems, Inc.', 'amount': 34.79999923706055}",

"{'id': 3, 'name': 'Teekay Offshore Partners L.P.', 'amount': 45.099998474121094}",

]

}

Для запису даних користувачу необхідно надати унікальні ідентифікатори проєкту, таблиці з даними та опціонально файл формату CSV або JSON. HTTP-запит для запису даних:

POST /query/write HTTP/1.1

Host: {{base-url}}

Content-Length: 300

Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

------WebKitFormBoundary7MA4YWxkTrZu0gW

Content-Disposition: form-data; name="project_id"

95c7863d-c211-4d81-adf1-c7d2862c0fe5

------WebKitFormBoundary7MA4YWxkTrZu0gW

Content-Disposition: form-data; name="datatable_id"

3248ac39-9139-4c33-b122-895dbc0871ad

------WebKitFormBoundary7MA4YWxkTrZu0gW—

Відповідь на запит:

200 OK

{"details": "Data is written"}

Також користувачу доступно оновлення та видалення проєктів, джерел даних та сховищ даних. Всі вони при видаленні видаляють пов’язані з ними дочірні об’єкти.

4.2 Випробування програмного продукту

Для випробування програмного продукту було проведено функціональне тестування прикладного програмного інтерфейсу. Нижче наведено опис кожного тест-кейсу та результати.

4.2.1 Мета випробувань

Метою випробувань програмного продукту є перевірка коректності роботи системи та виконання всіх функціональних вимог.

4.2.2 Результати випробувань

У таблиці 5.1 наведено результат перевірки процесу реєстрації користувача.

Таблиця 5.1 – Тестування реєстрації користувача

Передумова	Користувач не зареєстрований у системі. Вхідні дані не були використані іншим користувачем
Вхідні дані	Електронна пошта та пароль користувача
Послідовність кроків	– створити тіло запиту з вхідними даними – надіслати POST запит

Продовження таблиці 5.1

Очікувана відповідь	– повідомлення про успішну реєстрацію – токен із даними користувача
Зміни в системі	В базі даних створено запис з даними користувача

У таблиці 5.2 наведено результат перевірки створення сховища даних в системі та додавання таблиць у кластер сховища даних.

Таблиця 5.2 – Тестування створення сховища даних

Передумова	– користувач зареєстрований – користувач створив проєкт з актуальним URL для підключення до сховища даних – користувач створив одне джерело даних з актуальним URL для підключення
Вхідні дані	– ім’я сховища даних – перелік унікальних ідентифікаторів таблиць із групуванням за типом таблиці – унікальний ідентифікатор проєкту
Послідовність кроків	– створити тіло запиту з вхідними даними – надіслати POST запит
Очікувана відповідь	– повідомлення про створення із унікальним ідентифікатором сховища даних
Зміни в системі	– в базі даних створено запис із новим сховищем даних – в кластері користувача створені порожні таблиці

У таблиці 5.3 наведено результат перевірки обробки комплексного запиту та отримання пов’язаних об’єктів.

Таблиця 5.3 – Тестування отримання відомостей про проєкт

Передумова	– користувач зареєстрований – користувач створив проєкт – користувач створив джерело даних – користувач створив сховище даних
Вхідні дані	– унікальний ідентифікатор проєкту
Послідовність кроків	– створити запит з вхідними даними в якості параметру – надіслати GET запит
Очікувана відповідь	– відомості про проєкт – відомості про пов’язані джерела даних – відомості про пов’язане сховище даних – відомості про таблиці сховища даних
Зміни в системі	Змін не відбулося

У таблиці 5.4 наведено результат перевірки отримання відомостей про конкретний об’єкт.

Таблиця 5.4 – Тестування отримання відомостей про певне джерело даних

Передумова	– користувач зареєстрований – користувач створив проєкт – користувач створив джерело даних
Вхідні дані	– унікальний ідентифікатор джерела даних
Послідовність кроків	– створити запит з вхідними даними в якості параметру – надіслати GET запит
Очікувана відповідь	Відомості про джерело даних
Зміни в системі	Змін не відбулося

У таблиці 5.5 наведено результат перевірки отримання переліку таблиць конкретного джерела даних.

Таблиця 5.5 – Тестування отримання переліку таблиць з джерела даних

Передумова	– користувач зареєстрований – користувач створив проєкт – користувач створив джерело даних
Вхідні дані	– унікальний ідентифікатор джерела даних
Послідовність кроків	– створити запит з вхідними даними в якості параметру – надіслати GET запит
Очікувана відповідь	– перелік таблиць – відомості про назви та типи колонок таблиці
Зміни в системі	Змін не відбулося

У таблиці 5.6 наведено результат перевірки оновлення відомостей про конкретний об’єкт.

Таблиця 5.6 – Тестування оновлення відомостей про джерело даних

Передумова	– користувач зареєстрований – користувач створив проєкт – користувач створив джерело даних
Вхідні дані	– унікальний ідентифікатор джерела даних – нове ім’я джерела даних
Послідовність кроків	– створити запит з вхідними даними в якості параметру та тіла запиту відповідно – надіслати PUT запит
Очікувана відповідь	– повідомлення про зміну джерела даних з вказанням унікального ідентифікатора
Зміни в системі	Запис про джерело даних оновлено в базі даних

У таблиці 5.7 наведено результат перевірки видалення об’єкту та пов’язаних об’єктів.

Таблиця 5.7 – Тестування видалення проєкту

Передумова	– користувач зареєстрований – користувач створив проєкт – користувач створив джерело даних – користувач створив сховище даних
Вхідні дані	– унікальний ідентифікатор проєкту
Послідовність кроків	– створити запит з вхідними даними в якості параметру – надіслати DELETE запит
Очікувана відповідь	– повідомлення про видалення проєкту з вказанням унікального ідентифікатора
Зміни в системі	– видалення запису з проєктом з бази даних – видалення запису з пов’язаними джерелами даних з бази даних – видалення запису з пов’язаними таблицями з бази даних – видалення запису з пов’язаним сховищем даних з бази даних – видалення запису з пов’язаними таблицями сховища даних з бази даних

У таблиці 5.8 наведено результат перевірки запису даних у кластер із зовнішнього джерела.

Таблиця 5.8 – Тестування запису даних

Передумова	– користувач зареєстрований – користувач створив проєкт – користувач створив джерело даних – користувач створив сховище даних
Вхідні дані	– унікальний ідентифікатор проєкту – унікальний ідентифікатор таблиці

Продовження таблиці 5.8

Послідовність кроків	– створити запит з вхідними даними у форматі форми – надіслати POST запит
Очікувана відповідь	– підтвердження запису даних
Зміни в системі	– в кластері користувача створені таблиці – в таблиці додано дані з джерела даних

У таблиці 5.9 наведено результат перевірки виконання SQL-запитів для отримання даних з кластеру користувача.

Таблиця 5.9 – Тестування зчитування даних

Передумова	– користувач зареєстрований – користувач створив проєкт – користувач створив джерело даних – користувач створив сховище даних
Вхідні дані	– унікальний ідентифікатор проєкту – SQL-запит для зчитування даних
Послідовність кроків	– створити тіло запиту з вхідними даними – надіслати POST запит
Очікувана відповідь	Дані з кластеру користувача
Зміни в системі	Змін не відбулося

У таблиці 5.10 наведено результат перевірки обробки некоректних даних.

Таблиця 5.10 – Тестування надання неправильних даних для створення джерела даних

Передумова	– користувач зареєстрований

Продовження таблиці 5.10

Вхідні дані	– назва джерела даних – унікальний ідентифікатор неіснуючого проєкту – тип джерела даних – налаштування підключення до джерела даних
Послідовність кроків	– створити тіло запиту з вхідними даними – надіслати POST запит
Очікувана відповідь	Повідомлення про помилку та неіснуючий проєкт
Зміни в системі	Змін не відбулося

Висновок до розділу

В даному розділі надано покрокове пояснення доступних функцій платформи управління сховищем даних. Описано початок роботи з платформою управління сховищем даних і вказано рекомендації щодо розгортання системи. Надано приклади запитів та відповідей для правильних та неправильних вхідних даних.

Також проведено тестування системи, перевірка відповідності функціональним вимогам. Визначено тест-кейси, в яких описана ціль, послідовність кроків та очікуваний результат.

ВИСНОВКИ

Метою даного дипломного проєкту є спрощення процесу обробки та аналізу даних з декількох джерел для прогнозування та підтримки прийняття рішень.

У розділі з аналізом існуючих аналогів було розглянуто різні типи сховищ даних, їх переваги та недоліки. Вітрини даних фокусуються на конкретних бізнес-напрямках, корпоративні сховища даних мають широкий набір даних, а операційні сховища даних надають доступ до даних у реальному часі. За типом архітектури, Data Vault поєднує переваги та недоліки моделей Кімбалла та Інмона, модель Кімбалла підходить для швидкого розроблення сховища даних, а модель Інмона – для нормалізованих та очищених даних. Вибір типу сховища даних залежить від ресурсів та способу використання даних організацією. Також розглянуті аналоги сховищ даних, їх особливості та інтеграція з іншими сервісами. Підсумовуючи, вибір типу залежить від ресурсів, які доступні для розробки та обслуговування сховища даних, та від способу використання даних.

В другому розділі описано функціональну модель, акторів, варіанти використання системи та функціональні вимоги. Основними варіантами використання є управління сутностями та керування розподіленим сховищем даних. Обґрунтовано вибір мови програмування Python та інших засобів розробки. Також визначено інформаційне забезпечення системи, включаючи вхідні та вихідні дані та описано структуру бази даних.

В третьому розділі було проведено детальне дослідження та опис платформи управління сховищем даних. Він включає в себе змістовну постановку задачі, математичну постановку задачі, обґрунтування та опис методу розв'язання. Архітектура програмного забезпечення була розроблена з урахуванням функціональних вимог та представлена у вигляді структурної схеми, діаграми компонентів та діаграм послідовностей. Також наведено опис основних класів, необхідних для роботи програмного продукту. В цілому, розділ надає чітке розуміння задачі та структури платформи управління сховищем даних, що є важливим для подальшого розвитку та реалізації продукту.

В технологічному розділі представлено керівництво користувача та опис випробувань програмного продукту. В першому підрозділі описано початок роботи з платформою управління сховищем даних та основні операції, які користувач може здійснювати. Також наведені рекомендації щодо розгортання та побудови інфраструктури кластерів. В другому підрозділі було визначено мету випробувань та проведено тестування програмного продукту для перевірки виконання функціональних вимог.

Прикладний програмний інтерфейс в подальшому може бути використаний як компонент веб-сервісу для управління сховищем даних. Створений програмний продукт може бути доповнений користувацьким інтерфейсом та розширений інтеграцією з інструментами для аналітики даних. Також може бути розглянута можливість роботи з потоковими даними.

Підсумовуючи, внаслідок розробки платформи управління сховищем даних робота з даними з декількох джерел, в тому числі збір та аналіз даних, була спрощена, що і було основною метою дипломного проєкту.

ПЕРЕЛІК ПОСИЛАНЬ

1. Snowflake key concepts. URL: https://docs.snowflake.com/en/user-guide/intro-key-concepts (дата звернення 02.05.2023).

2. Amazon Redshift conceptual overview. URL: https://docs.aws.amazon.com/redshift/latest/gsg/getting-started.html (дата звернення 02.05.2023).

3. What is Azure Synapse Analytics? URL: https://learn.microsoft.com/en-us/azure/synapse-analytics/overview-what-is (дата звернення 02.05.2023).

4. What is Python? Executive Summary. URL: https://www.python.org/doc/essays/blurb/ (дата звернення 08.05.2023).

5. Tour of Scala. Introduction. URL: https://docs.scala-lang.org/tour/tour-of-scala.html (дата звернення 08.05.2023).

6. FastAPI documentation. URL: https://fastapi.tiangolo.com/ (дата звернення 08.05.2023).

7. Pydantic Overview. URL: https://docs.pydantic.dev/latest/ (дата звернення 08.05.2023).

8. SQLAlchemy Overview. URL: https://docs.sqlalchemy.org/en/20/intro.html (дата звернення 08.05.2023).

9. Structlog documentation. URL: https://www.structlog.org/en/stable/ (дата звернення 08.05.2023).

10. PySpark Overview. URL: https://spark.apache.org/docs/latest/api/python/ (дата звернення 10.05.2023).

11. Delta Spark documentation. URL: https://docs.delta.io/latest/api/python/index.html (дата звернення 10.05.2023).

12. Delta Lake Introduction. URL: https://docs.delta.io/latest/delta-intro.html (дата звернення 10.05.2023).

13. Visual Studio Code documentation. URL: https://code.visualstudio.com/docs (дата звернення 10.05.2023).

14. DBeaver wiki. URL: https://dbeaver.com/docs/wiki/ (дата звернення 10.05.2023).

15. Postman Overview. URL: https://learning.postman.com/docs/introduction/overview/ (дата звернення 10.05.2023).

16. Azure Databricks documentation. URL: https://learn.microsoft.com/en-us/azure/databricks/getting-started/ (дата звернення 11.05.2023).