КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота

на здобуття ступеня магістра
за освітньо-науковою програмою «Інформатика»
за спеціальністю 122 Комп'ютерні науки
на тему:

Розробка системи управління засобами завантаження даних у Snowflake

Виконав студент 2-го курсу
Олександр ГУТАРЕВИЧ

(підпис)

Науковий керівник:
доцент, кандидат технічних наук
Олексій ТКАЧЕНКО

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань

Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
« 08 » травня 2023 р.,
протокол № 16
Завідувач кафедри
М. С. Нікітченко

(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 82 сторінки, 29 ілюстрацій, 6 таблиць, 2 додатки, 43 джерела посилань.

ХМАРНЕ СХОВИЩЕ ДАНИХ, MPP-АРХІТЕКТУРА, SNOWFLAKE, ЗАСОБИ ЗАВАНТАЖЕННЯ ДАНИХ, SNOWPIPE, УПРАВЛІННЯ ОБ'ЄКТАМИ SNOWPIPE.

Об'єктом дослідження є хмарні сховища даних. Предметом роботи є управління засобами завантаження даних в Snowflake. Метою роботи є створення системи управління засобами завантаження даних в Snowflake.

Методи розроблення: веб-розробка, архітектурні рішення.

Інструменти розроблення: операційна система – Windows 10, Visual Studio 2022, .NET 6, Microsoft SQL Server 2019.

Результати роботи: детально розглянуто концепцію сховища даних, її відмінності від інших концепцій збереження даних; розглянуто основні ідеї хмарних сховищ даних, виділено їх основні аспекти; проведений огляд найбільш популярних на ринку хмарних сховищ даних, на основі чого була створена порівняльна таблиця; детально розглянуто деталі реалізації хмарного сховища даних Snowflake; розглянуто засоби завантаження даних в Snowflake та виділено основні проблеми існуючого рішення для управління ними; розроблено систему зі зручним веб-інтерфейсом, яка вирішує визначені проблеми.

Зміст

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

IDE – Integrated Development Environment – інтегроване середовище розробки

OLTP – Online transaction processing – обробка транзакцій в реальному часі

OLAP – Online analytical processing – аналітична обробка у реальному часі

DML – Data Manipulation Language, мова маніпуляції даними

SQL – Structured Query Language, мова структурованих запитів

ПЗ – програмне забезпечення

БД – база даних

СУБД – система управління базами даних

REST – Representational state transfer – передача репрезентативного стану

API – Application programming interface – прикладний програмний інтерфейс

ETL – Extract-Transform-Load

MPP – Massively parallel processing

SPA – Single-page application – односторінковий застосунок

LTS – Long time support – довготривала підтримка

WASM – WebAssembly

ORM – Object-Relational Mapper

JWT – JSON Web Token

ВСТУП

Оцінка сучасного стану об’єкта розробки. На сьогодні однією з найбільш цінних речей є інформація. Вже давно пройшли часи коли компанії приймали стратегічні рішення на інтуїтивному рівні. В сучасному світі всі стратегічні рішення приймаються на основі аналізу великих обсягів даних. Для виконання такого аналізу виникла потреба в спеціальних інструментах, які б дозволяли зберігати та аналізувати інформацію ефективно. З цією ціллю все більше компаній додають в свою інфраструктуру сучасні хмарні сховища даних.

Наразі ринок хмарних сховищ даних росте з кожним роком, лідером на ринку є Snowflake [1]. Це сучасне хмарне сховище даних дозволяє ефективно зберігати та аналізувати гігантські об'єми даних, приносячи неоціненну користь бізнесу.

Актуальність роботи та підстави для її виконання. З кожним роком все більше і більше компаній упевнюються в необхідності мати власне сховище даних, щоб приймати обґрунтовані рішення та залишатися конкурентними на ринку. Крім того, використання саме хмарних технологій сприяє максимально ефективному використанню обчислювальних ресурсів та часто знижує витрати (як на персонал, так і на обладнання). Тому хмарні сховища даних користуються значним попитом та активно розвиваються.

Ситуація на ринку підтверджує, що все більше компаній інтегрують Snowflake в свої системи. Однією з ключових задач при використанні Snowflake є завантаження в нього даних, для чого рекомендується використовувати специфічні об’єкти завантаження даних сервісу Snowpipe. [2]

Проте, існуючі інструменти для управління цими об’єктами в Snowflake містять ряд недоліків, які погіршують досвід користувача та можуть призвести до проблем в процесі підтримки хмарного сховища даних: історія використання об’єктів завантаження даних зберігається лише

протягом 14 днів і видаляється разом з об’єктами; відсутній графічний інтерфейс користувача для керування, використання та перегляду історії об’єктів завантаження даних.

Мета й завдання роботи. Метою роботи є проектування та розробка системи для виправлення існуючих недоліків в управлінні засобами завантаження даних в Snowflake. Для досягнення цієї мети поставлено такі завдання:

- Розглянути концепцію сховища даних, порівняти з іншими концепціями збереження даних (база даних, озеро даних).
- Провести огляд найбільш популярних рішень на ринку хмарних сховищ даних.
- Розглянути структуру, основні елементи, засоби та інструменти Snowflake для завантаження даних.
- Визначити основні проблеми в існуючому рішенні для управління засобами завантаження даних в Snowflake.
- Розробити систему, яка б вирішувала визначені проблеми управління засобами завантаження даних в Snowflake.

Об’єкт, методи та засоби розроблення. Об’єктом розроблення системи є засоби Snowflake для завантаження даних. Предметом роботи є розробка системи управління засобами завантаження даних в Snowflake.

Розробка проходила за ітераційною моделлю. Кожна ітерація розробки передбачала імплементацію певного обсягу функціоналу системи.

Для розробки була обрана платформа .NET 6 [3] та мова C# [4]. В якості внутрішньої бази даних розроблюваної системи використовувалася СУБД Microsoft SQL Server 2019 [5].

В якості інструменту розробки програмного засобу було обрано Visual Studio 2022 [6] – безкоштовне, вільно поширюване інтегроване середовище розробки (IDE).

РОЗДІЛ 1. КОНЦЕПЦІЇ ЗБЕРЕЖЕННЯ ДАНИХ

1.1 Концепція сховища даних

Бізнес став активно цікавитись корпоративними сховищами ще наприкінці минулого століття. Їх впроваджували для збільшення швидкості реагування на зміни, моніторингу показників ефективності та автоматизації процесів. Різні програми відповідали за різні процеси: одні використовувалися для фінансових операцій, інші – для координації ланцюжків постачання, треті допомагали аналізувати показники продажів.

Однак, такий підхід призвів до того, що ключові дані бізнесу зберігалися розрізнено. Виникла потреба в рішенні, яке б дозволило аналізувати інформацію повністю, а не дані з різних систем окремо.

Для вирішення цієї проблеми було створено особливий інструмент – сховище даних, або Data Warehouse. Фактично сховище даних – це предметно-орієнтована база даних, яка консолідує важливу бізнес-інформацію та дозволяє в автоматичному режимі готувати консолідовані звіти.

Data Warehouse – це єдине корпоративне сховище архівних даних із різних джерел (систем, департаментів та інше). Мета Data Warehouse – забезпечити користувача (компанію та її ключових осіб) можливістю приймати вірні рішення в процесі управління бізнесом на основі цілісної інформаційної картини.

1.2 Порівняння концепцій сховища даних та бази даних

На перший погляд база даних і сховище даних є схожими суміжними поняттями, проте ці дві концепції мають ряд суттєвих відмінностей. Найбільш істотна відмінність між базами даних і сховищами даних полягає в

тому, що база даних як правило, працює з даними з одного джерела, в той час як сховище даних наповнюється з багатьох джерел даних.

Бази даних використовують технологію обробки даних OLTP (Online transaction processing), яка забезпечує швидке видалення, вставку та оновлення великої кількості коротких онлайн-транзакцій. Цей тип обробки негайно реагує на запити користувачів і тому використовується для обробки повсякденних операцій бізнесу в режимі реального часу. Наприклад, якщо користувач хоче забронювати готельний номер за допомогою форми онлайн-бронювання, процес виконується за допомогою OLTP.

Сховища даних використовують технологію обробки даних OLAP (Online analytical processing) для швидкого аналізу величезних обсягів даних. Цей процес дає аналітикам можливість дивитися на дані з різних точок зору. Наприклад, база даних записує дані про продажі за кожну хвилину кожного дня, проте аналітику необхідно просто знати загальну суму продажів за кожний день. Для цього потрібно зібрати та підсумувати дані про продажі за кожен день. OLAP спеціально розроблена для таких цілей і дозволяє виконувати подібні операції в тисячі разів швидше, ніж якби застосовувався OLTP.

Ще однією ключовою відмінністю бази та сховища даних є різні вектори оптимізації. Оптимізація баз даних, в першу чергу, спрямована на забезпечення високої швидкості та ефективності виконання DML операцій (отримання, вставки, видалення, оновлення). Для ефективної обробки транзакцій час відповіді бази даних має бути мінімальним. На противагу, сховища даних оптимізовані для швидкого виконання невеликої кількості складних запитів до великих багатовимірних наборів даних.

Не менш суттєвою відмінністю є структура даних при збереженні в базі та сховищі даних. Дані в базах даних зазвичай нормалізовані. Мета нормалізації полягає в тому, щоб зменшити і навіть усунути надмірність даних, тобто не зберігати один і той самий фрагмент даних більше одного

разу. Таке усунення дублювання інформації призводить до підвищення узгодженості даних, що в свою чергу сприяє покращенню точності даних.

Нормалізація даних розбиває їх на багато різних таблиць. Кожна таблиця представляє окрему сутність даних. Наприклад, база даних зображена на рисунку 1.1, яка зберігає інформацію про BOOK SALES (продажі книг), може мати три таблиці для позначення інформації BOOK (книга), SUBJECT (тема, що висвітлюється в книзі) і PUBLISHER (видавець).

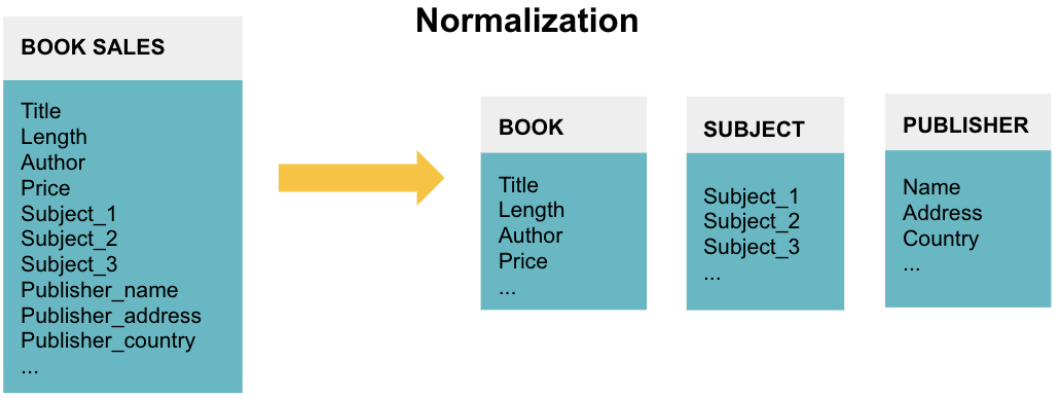


Рис. 1.1 – Приклад нормалізації даних [8]

Нормалізація даних гарантує, що база даних займає мінімальний простір на дисках, і тому вона сприяє ефективному використанню пам’яті. Проте, нормалізація негативно впливає на швидкість виконання запитів. Запити до нормалізованої бази даних можуть бути повільними та громіздкими, тому в сховищах даних частіше зберігають дані в денормалізованому або частково-денормалізованому вигляді. Таким чином значно полегшується складність запитів та пришвидшується їх виконання. На рисунку 1.2 наведено приклад денормалізації даних.

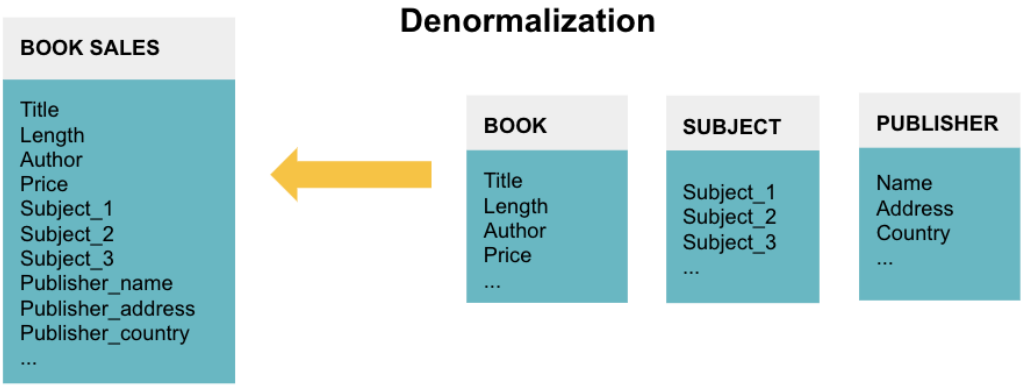


Рис. 1.2 – Приклад денормалізації даних [8]

Бази даних зазвичай просто обробляють транзакції, проте часто паралельно необхідно виконувати певний аналіз даних. Глибокий аналіз в базах даних є досить складним як для обчислювальних потужностей так і користувача (аналітика), що зумовлено використанням нормалізованої структури даних (необхідно виконувати чисельні з'єднання таблиць). Для створення складних запитів в СУБД (система управління базами даних) потрібен кваліфікований розробник або аналітик та значні обчислювальні потужності. Крім того, аналіз бази даних не є глибоким, адже база даних лише надає статичний «знімок» стану даних в певний момент часу.

Сховища даних дозволяють виконувати складні багатовимірні аналітичні запити набагато простіше як для користувача (аналітика) так і обчислювальних потужностей. Відпадає необхідність в навчанні персоналу користуватися складним програмним забезпеченням СУБД. Сховища даних не лише спрощують аналіз, а й значно покращують його якісь та корисність: можна глибоко зануритися в дані та побачити, як вони змінюються з часом, а не просто отримати знімок стану даних, який здана надати база даних.

Бази даних обробляють транзакції, які стосуються лише певної частини бізнесу. Тому вони як правило містять поточні, а не історичні дані про один бізнес-процес. Сховища даних використовуються для аналітичних цілей і бізнес-звітності, тому вони зазвичай зберігають історичні дані шляхом інтеграції копій даних транзакцій з різних джерел. Сховища даних також часто використовують потокові канали обробки даних для формування звітів, які використовують найновішу інтегровану інформацію, у режимі реального часу.

Ще однією відмінністю є те, що бази даних спроектовані для одночасного обслуговування великої кількості користувачів з мінімальними втратами продуктивності. Так як основною ціллю сховищ даних є аналітика, вони не підтримують велику кількість одночасних користувачів. Сховища даних зазвичай відокремлені від зовнішніх додатків та використовуються

обмеженим числом бізнес-аналітиків, які виконують масивні складні запити, які є досить дорогими з точки зору обчислення.

Ще однією відмінністю є дотримання вимог ACID (atomicity, consistency, isolation, durability). Бази даних, як правило, дотримуються цих вимог, адже вони зберігають дані про бізнес-операції, для яких дотримання цілісності є критичним аспектом. Оскільки сховища даних зосереджені, в першу чергу, на зчитуванні даних, а не їх зміні, допускається часткове дотримання вимог ACID. Проте, більшість хмарних реалізацій сховищ даних підтримують запити з виконанням вимог ACID.

Також варто зазначити, що SLA (Service Level Agreement) для баз даних зазвичай зазначають, що база має бути доступна 99.99% часу. До сховищ даних такі строгі вимоги не застосовуються, більш того, нормальною практикою є недоступність сховища даних при завантаженні нових даних.

Порівняння бази даних та сховища даних узагальнено в таблиці 1.1:

Табл. 1.1 - Порівняння бази і сховища даних. Частина 1

Аспект	База даних	Сховище даних
Метод обробки	OLTP (Online transaction processing)	OLAP (Online analytical processing)
Оптимізація	Пришвидшення видалення, вставки, оновлення коротких онлайн-транзакцій	Пришвидшення виконання складних масивних запитів до багатовимірних даних
Структура даних	Зазвичай нормалізовані дані, містить мінімум дублювання інформації	Денормалізовані або часково-денормалізовані дані
Природа даних	Поточні дані про певну частину бізнесу	Історичні дані про всі частини бізнесу
Одночасні користувачі	Підтримує велику кількість користувачів, які паралельно виконують запити	Призначене для невеликої кількості одночасних користувачів

Табл. 1.1 - Порівняння бази і сховища даних. Частина 2

Аспект	База даних	Сховище даних
Придатність до аналізу	Аналіз обмежений, виконується повільно та складно (необхідно з'єднувати купу таблиць)	Аналіз повний та глибокий, виконується досить швидко та неважко (не потребує великої кількості з'єднання таблиць)
Дотримання ACID	Транзакції проводяться з виконанням вимог ACID	Не завжди дотримується вимог ACID, проте більшість сучасних реалізацій надають таку можливість
SLA	99.99% часу працює	Може бути тимчасово недоступне в зв'язку з завантаженням нових даних
Джерела даних	Обмежена одним джерелом даних з певної бізнес-функції	Усі джерела даних з усіх бізнес-функцій
Типові запити	Прості транзакційні запити	Складні запити для глибинного аналізу

Отже, база даних і сховище даних є ефективними засобами збереження даних, які відрізняються, в першу чергу, ціллю, з якою дані зберігаються. Бази даних використовуються для підтримки працездатності одного певного бізнес-процесу, наприклад, продаж товарів в інтернет-магазині. Сховища даних необхідні для проведення глибинного сукупного аналізу всіх наявних в організації бізнес-процесів, наприклад, прогнозування попиту та прибутку для виявлення категорій товарів та послуг, на продажі яких варто сфокусуватися для максимізації прибутку.

1.3 Концепція озера даних та порівняння зі сховищем даних

В результаті автоматизації бізнес-процесів, організації стали продукувати гігантський об’єм інформації. Інформацію, користь якої є очевидною, сучасні корпорації зберігають в сховищах даних. Процес ETL (Extract, Transform, Load) необхідний для завантаження даних до сховища є відносно затратним, якщо проводити його для всіх даних, які продукує організація. Проте, інформація – це сучасне золото, і навіть якщо корпорація поки не знає з якою ціллю вона може використати якісь дані, втрачати їх вона не хоче, але й обробляти їх затратно. Це призвело до виникнення концепції озера даних.

Озеро даних (Data lake) – це репозиторій, який зберігає необроблені дані з різних джерел в їх оригінальному форматі до тих пір поки ці дані не знадобляться.

Порівняно зі сховищем даних, озеро даних зберігає дані в їх оригінальному вигляді, що призводить до необхідності використання набагато більшої кількості пам’яті для збереження даних. Необроблені дані добре піддаються аналізу та їх зручно використовувати для машинного навчання. Основний ризик збереження даних в необробленому вигляді полягає в тому, що озера даних можуть стати «болотами» даних, якщо дані поганої якості та не виконуються заходи для керування ними.

Сховища даних, натомість, виконують обробку даних, що дозволяє заощадити простір для збереження даних та забезпечити їх високу якість. Крім того, оброблені дані легше сприймаються людиною, що полегшує їх аналіз.

Збереження даних в необробленому вигляді також призводить до того, що для їх аналізу необхідно залучати спеціаліста в data science і використовувати спеціальні інструменти для їх використання в певних бізнес-цілях.

Архітектура озера даних не має структури, що сприяє полегшенню використання озера даних в цілому (а не даних в них). Так як архітектура не має структури, то її легко зміни і будь-які зміни, які вносяться в дані, можна зробити дуже швидко, адже озера даних мають досить мало обмежень. Сховища даних, натомість, є більш структурованими. Це призводить до того, що дані легше розшифрувати та зрозуміти їх сенс, проте структурованість робить сховища даних складними та дорогими для оновлення структури.

Порівняння сховища даних та озера даних узагальнено в таблиці 1.2:

Табл. 1.2 - Порівняння сховища і озера даних.

Аспект	Сховище даних	Озеро даних
Форма даних	Оброблені структуровані дані	Необроблені дані в оригінальній формі
Ціль даних	Чітко визначена, дані активно використовуються	Не визначена
Користувачі	Бізнес-аналітики	Дослідники даних (Data scientists)
Гнучкість до змін	Зміни в структурі можуть бути досить дорогими	Чіткої структури немає, тому схильне до змін

Отже озеро даних, як і сховище даних, є ефективним засобом збереження даних і відрізняється від сховища даних тим, що дані зберігаються в необробленому оригінальному вигляді. З цього випливає ряд описаних переваг і недоліків, порівняно зі сховищем даних.

РОЗДІЛ 2. ОГЛЯД СУЧАСНИХ ХМАРНИХ СХОВИЩ ДАНИХ

В минулому розділі було детально розглянуто концепцію сховища даних, проведено порівняння з іншими концепціями збереження даних. Виявилося, що сховища даних наразі необхідні бізнесу та користуються значним попитом. Прогнозується, що до 2028 року ринок сховищ даних зросте до 50 млрд. доларів США [11].

В цьому розділі буде розглянуто сучасні реалізації, які дозволяють компаніям ефективно створювати і підтримувати свої сховища даних.

2.1 Переваги сучасних хмарних сховищ даних

До приходу хмарних провайдерів сховищ даних на ринок компаніям доводилося будувати свої сховища даних на основі локальних серверів. Це призводило до того, що при бажанні збільшити розмір сховища або його обчислювальну потужність, компаніям доводилося тратити значні кошти для придбання додаткової пам’яті та більш потужних серверів.

Крім того, для управління сховищем даних необхідний був спеціальний персонал, який слідкував за станом сховища та усував неполадки як на програмному так і на фізичному рівнях. Така робота під силу лише висококваліфікованому персоналу, на який необхідно було виділяти додаткові кошти.

З приходом хмарних провайдерів на ринок стало можливим створення гнучких хмарних сховищ даних. Це дозволило компаніям значно скоротити витрати на придбання дорогого обладнення та зменшити витрати на персонал. Використання хмарних сховищ даних надає ряд суттєвих переваг.

Однією з переваг є використання технології розподілених обчислень, що дозволяє значно пришвидшити обробку даних шляхом використання багатьох обчислювальних вузлів. Більшість сучасних хмарних сховищ даних використовують архітектуру MPP (Massively parallel processing), яка

координує виконання певної задачі між багатьма процесорами, кожен з яких використовує власну операційну систему та оперативну пам’ять і комунікує з іншими процесорами завдяки певним інтерфейсам повідомлень. Використання цієї архітектури забезпечує чудову масштабованість при виконанні розподілених обчислень.

Ще однією ключовою перевагою хмарних сховищ даних є їх еластичність. Тобто сховище даних динамічно розподіляє обчислювальні ресурси залежно від поточних потреб. Це дозволяє значно знизити витрати і водночас отримати всі переваги високопродуктивних сховищ даних.

Не менш важливою перевагою є те, що використовуючи хмарні сховища даних компанія знімає з себе відповідальність за управління та обслуговування фізичних ресурсів (серверів). Також хмарні сховища значно полегшують створення резервних копій та реплік (в територіально інших дата-центрах), які можуть врятувати дані компанії у випадку виходу серверів з ладу (наприклад, в результаті стихійного лиха).

Отже, більшість сучасних реалізацій хмарних сховищ даних дозволяють легко розподіляти обчислення, забезпечують масштабованість, еластичність та простоту управління. Тому, насьогодні, все більше і більше організацій переходять на використання хмарних сховищ даних.

2.2 Ключові аспекти сучасних хмарних сховищ даних

Перед тим як розпочати огляд і порівняння сучасних хмарних сховищ даних, варто виділити ключові аспекти, відносно яких потрібно розглядати хмарне сховище даних.

Архітектура. Сучасні хмарні сховища даних використовують різні підходи до організації архітектурних компонентів, проте практично всі вони спираються на MPP (Massively parallel processing). Наразі є два основні варіанта MPP архітектури: SN (shared-nothing) MPP та SD (shared disk) MPP.

В SN архітектурі кожен вузол (процесор) працює незалежно і не має спільної з іншими вузлами (процесорами) пам’яті. Різні дані обчислюються паралельно на різних вузлах.

В SD архітектурі вузли (процесори) також виконують обробку даних паралельно, проте вони всі під’єднані до спільного диску, що означає, що всі вузли в кластері мають доступ до всіх даних (а не до їх частини).

Різницю між цими архітектурами наглядно демонструє рисунок 2.1, на якому 1-4 це обчислювальні вузли (процесори), а A-D – частини даних які оброблюються на цих вузлах. Як помітно, в архітектурі SD кожен вузол має доступ до всіх частин даних, а в архітектурі SN кожен вузол має доступ лише до тієї частини даних, яку йому необхідно обробити.

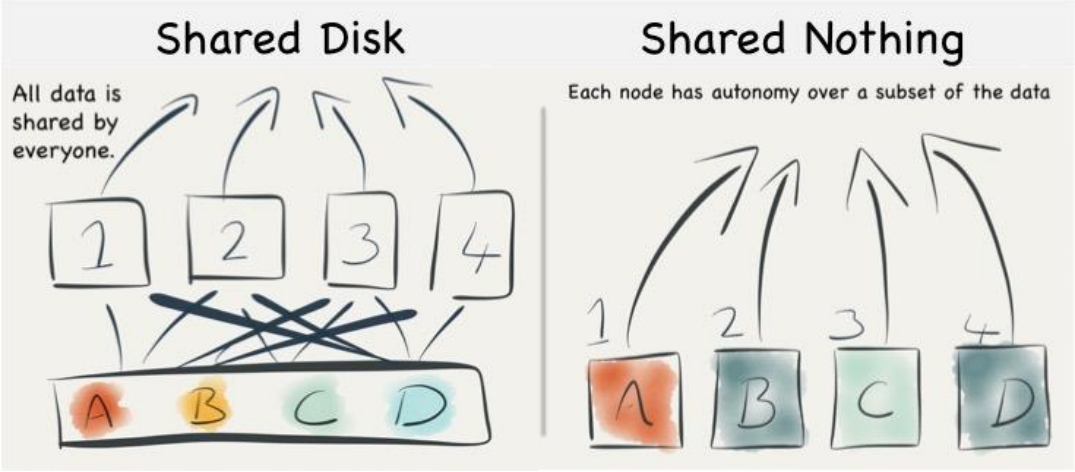


Рис. 2.1 – SD (зліва) та SN (справа) архітектури [13]

SN архітектура краще підходять для великих компаній, які обслуговують мільйони користувачів. Вона вимагає більш складної стратегії розподілу та, в ідеалі, уникнення операцій, які охоплюють декілька сховищ. В більшості випадків SD архітектури може бути цілком достатньо. Однак наразі існують сучасні хмарні сховища даних, такі як Snowflake, які дозволяють використовувати обидва MPP підходи одночасно.

Продуктивність і швидкість обробки даних. Висока продуктивність зменшує витрати, оскільки деякі хмарні провайдери стягують плату за час використання їх обчислювальних можливостей. Ще одним важливим

фактором є підтримка виконання одночасних запитів, що також значно прискорює виконання обчислень.

Масштабованість. Масштабування дозволяє компаніям контролювати продуктивність своєї системи, що оптимізує використання ресурсів і заощаджує кошти. Існує два види масштабованості: горизонтальна – зміна продуктивності за рахунок зміни кількості обчислювального обладнання; та вертикальна – зміна продуктивності системи шляхом заміни наявного обчислювального обладнання на більш потужне (наприклад, перехід на більш потужні процесори або збільшення обсягу оперативної пам’яті).

Інформаційна безпека. В сучасному світі інформаційній безпеці приділяють все більше уваги, тому провайдери хмарних сховищ даних мають слідкувати за безпекою конфіденційних даних своїх клієнтів та забезпечувати ефективні механізми захисту інформації.

Інтеграція зі стороннім ПЗ. Наразі наявно багато інструментів для аналітики даних і сумісність хмарних сховищ даних з цими інструментами є важливим фактором при міграції сховища даних організації в хмару.

Завантаження даних. Один з ключових процесів при використанні сховища даних це завантаження в нього даних. Сучасні хмарні сховища даних мають надавати широкий спектр можливостей стосовно завантаження даних: підтримувати завантаження потокових даних (даних в режимі реального часу) та різноманітні способи завантаження частинами (multipart uploading).

Резервне копіювання та відновлення даних. Інформація, яку компанії зберігають в сховищах даних безумовно є дуже цінною і її втрата вході якоїсь несправності або стихійного лиха є надзвичайно небажаною. Тому сучасні хмарні сховища даних мають забезпечувати клієнтам можливість створення резервних копій та відновлення даних при виникненні непередбачуваних ситуацій.

Процес імплементації. Кожна компанія має свої специфічні дані та потреби, тому сучасні хмарні сховища даних мають забезпечувати широкі

можливості налаштування відповідно до потреб клієнта. Також важливим фактором є наявність зручних якісно документованих інструментів для розробників, які дозволяють легко вибудувати процеси завантаження і обробки даних.

Коштовність. Можливість збереження коштів для організацій є одним з головних мотиваторів для переходу на використання хмарних сховищ даних. Сучасні провайдери пропонують різні моделі стягнення плати за надання послуг, що дозволяє клієнтам обрати найбільш вигідну залежно від їх потреб.

2.3 Огляд сучасних хмарних сховищ даних

2.3.1 Snowflake

Snowflake [1] – це перше хмарне сховище даних, розроблене з врахуванням всіх аспектів та особливостей хмарних обчислень. Він постачається як SaaS (Software-as-a-Service - програмне забезпечення як послуга), хоча самі творці Snowflake виділяють окрему концепцію для свого продукту – Datawarehouse-as-a-Service (сховище даних як послуга). Snowflake можна розгорнути на найпопулярніших хмарних провайдерах, таких як Amazon Web Services [14], Microsoft Azure [15] та Google Cloud Platform [16].

Snowflake швидкий, гнучкий та простий у використанні. Від клієнта очікується лише встановлення розміру та кількості обчислювальних кластерів, а всі інші аспекти підтримки Snowflake бере на себе, що робить це рішення практично безсерверним (serverless).

Архітектура. Архітектура Snowflake [1] спеціально розроблена з врахуванням всіх аспектів хмарних обчислень, та поєднана з інноваційним двигуном SQL запитів. Вона спирається на гібридну комбінацію SN та SD архітектур, що надає можливість користуватися перевагами обох архітектур при виконанні розподілених обчисленнях. Як показано на рисунку 2.2, архітектура Snowflake містить три основні шари: збереження даних, обробка запитів, хмарні сервіси.

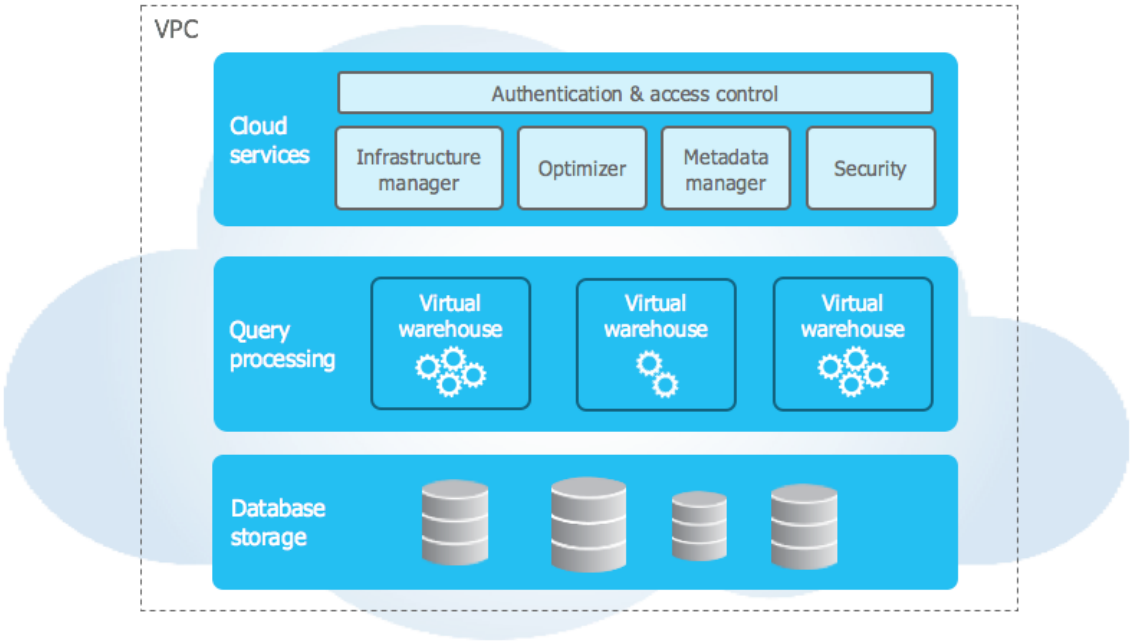


Рис. 2.2 – Архітектура Snowflake [17]

Продуктивність і швидкість обробки даних. Завдяки виокремленню окремих шарів збереження даних та виконання запитів, Snowflake дозволяє виконувати одночасні запити до даних не втрачаючи продуктивність. Різні запити виконуються незалежно не впливаючи один на одного, що призводить до підвищення продуктивності. Згідно тестів, проведених Fivetran, 99 запитів TPC-DS (кожна TPC-DS складається з 24 таблиць, найбільша з яких містить 4 мільйони рядків даних) Snowflake в середньому обробляє 8,21 секунд [22].

Масштабованість. Snowflake надає широкий спектр можливостей з точки зору масштабування, в тому числі і авто-масштабування (як горизонтальне, так вертикальне). Тобто для масштабування не потрібно залучення оператора бази даних або адміністратора, оскільки програмне забезпечення здатне автоматично змінювати розмір та потужність обчислювальних кластерів відповідно до нагальних потреб.

Інформаційна безпека. Snowflake забезпечує високий рівень безпеки, про що свідчить відповідність більшості стандартів захисту даних, включаючи SOC 1 Type 2, SOC 2 Type 2 для всіх версій Snowflake і PCI DSS,

HIPAA, HITRUST для Business Critical Edition або вище. Система також забезпечує засоби для контролю доступу і високорівневу безпеки даних (автоматичне шифрування всіх даних та файлів).

Інтеграція зі стороннім ПЗ. Snowflake забезпечує сумісність з різноманітними засобами інетеграції даних, BI (Business Intelligence) та аналітичним інструментами. Наприклад, такими як IBM Cognos, Azure Data Factory, Oracle Analytics Cloud, Fivetran, та Google Cloud.

Завантаження даних. Snowflake підтримує підходи до інтеграції даних ELT і ETL, тобто перетворення даних може відбуватися під час або після самого завантаження. Він надає зручні інструменти з широкими можливостями для автоматичного завантаження даних з сховища того провайдера, на якому він розгорнутий, наприклад для AWS – це S3. Snowflake також чудово працює з потоковими даними.

Резервне копіювання та відновлення даних. Snowflake використовує підхід fail-safe (захист від збоїв) замість резервного копіювання. Fail-safe підхід пропонує 7-денний період, протягом якого можна відновити дані, які могли бути пошкоджені або втрачені через збої системи.

Процес імплементації. Snowflake вважається одним з найбільш інтуїтивно зрозумілих та простих хмарних сховищ даних, що багато в чому завдячує відчуттю безсерверної архітектури. Snowflake успадкував багато функцій традиційних реляційних баз даних і поєднав їх з хмарними технологіями. Платформа дозволяє створювати та підтримувати декілька сховищ даних в межах одного облікового запису. Розмір обчислювального кластера для кожного сховища даних можна детально налаштувати. Проте, налагодження Snowflake та будування процесів завантаження та обробки даних, все одно, потребує знань та навичок SQL та хорошого розуміння архітектури сховища даних. Варто відмітити, що Snowflake має досить добре пророблену інформативну документацію, яка доступно розкриває більшість питань, з якими може стикнутися розробник.

Коштовність. Snowflake пропонує різноманітні тарифні плани, які варіюються починаючи з рівня надання послуг, закінчуючи хмарним провайдером та регіоном (дата центру) розгортання. Наявні як on-demand (на вимогу) так і pre-purchase (по передплаті) тарифні плани. Оскільки рівні збереження даних та обчислення розділені, останні оплачуються окремо на посекундній основі (мінімум 60 секунд).

2.3.2 Google BigQuery

Google BigQuery [20] – хмарне сховище даних, розроблене Google націлене на виконання аналітичних запитів з великими обсягами даних. З моменту випуску, BigQuery [20] активно розвивався: було додано та оновлено чисельні функції (в тому числі методи машинного навчання), які значно покращили продуктивність, безпеку та надійність сховища.

Архітектура. BigQuery [20] має безсерверну SN MPP архітектуру (продемонстрована на рисунку 2.3), де сховище та обчислення розділені. Основний компонент архітектури BigQuery, Dremel – це інноваційний потужний паралельний механізм запитів, який в комбінації з іншими низькорівневими архітектурними технологіями, такими як Colossus, Jupiter та Borg, дозволяє обробку тисяч записів за секунду. Дані зберігаються в розподілених сховищах, які підтримують реплікацію, і обробляються в обчислювальних кластерах. Така структура забезпечує величезну гнучкість і сприяє швидкій обробці великих об’ємів даних.

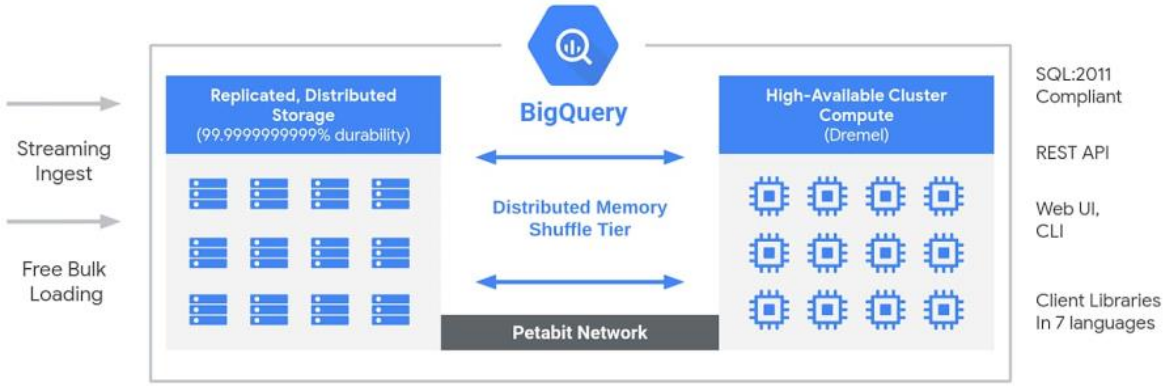


Рис. 2.3 – Архітектура BigQuery [21]

Продуктивність і швидкість обробки даних. BigQuery підтримує розділення даних, що сприяє високій продуктивність запитів. Запити до даних можна виконати за допомогою SQL або через Open Database Connectivity (ODBC). Згідно з тестами Fivetran, Google BigQuery показав хорошу, але не найвищу продуктивність – середня тривалість виконання 99 запитів TPC-DS (кожна TPC-DS складається з 24 таблиць, найбільша з яких містить 4 мільйони рядків даних) становить 11,18 секунд [22].

Масштабованість. Подібно до Snowflake, BigQuery окремо виділяє рівні обчислення та збереження, що дозволяє користувачам масштабувати (як вертикально так і горизонтально) ресурси відповідно до своїх потреб. BigQuery здатний на відносно швидко виконання запити в режимі реального часу до петабайт даних.

Інформаційна безпека. Для захисту конфіденційних даних, BigQuery пропонує безпеку навіть на рівні колонок, що дозволяє створювати політики та перевіряти статус доступності. BigQuery також пропонує Cloud DPL (запобігання втрати даних у хмарі) та систему керування ключами шифрування. Як частина середовища Google Cloud, BigQuery підтримує відповідність чисельним стандартам інформаційної безпеки, зокрема HIPAA, FedRAMP, PCI DSS, ISO/IEC, SOC 1,2,3.

Інтеграція зі стороннім ПЗ. Крім операційних баз даних, BigQuery сумісна з широким набором інструментів інтеграції даних, рішень BI та AI. Він також інтегрується з системами Google Cloud Platform [16], що робить його чудовим вибором, оскільки сьогодні чимало компаній використовують Google Workspace, раніше відомий як G Suite.

Завантаження даних. Разом з традиційним пакетним завантаженням даних ETL/ELT за допомогою стандартного діалекту SQL, BigQuery дозволяє здійснювати потокову передачу даних – завантажувати дані рядок за рядком у режимі реального часу за допомогою спеціального API для потокових даних.

Резервне копіювання та відновлення даних. BigQuery підтримує широкі можливості для резервного копіювання даних та їх аварійне відновлення. Користувачам також доступна можливість створення запитів до знімків стану даних за останні 7 днів.

Процес імплементації. Що стосується рівня зручності використання, BigQuery займає високі рейтинги завдяки повністю керованій природі сховищ даних, а це означає, що обслуговуванням та оновленнями займається команда інженерів BigQuery. Але, попри це вибудова процесів обробки даних вимагає поглиблених знань мови SQL та інструментів ETL. Для кваліфікованих розробників процес налаштування й конфігурації BigQuery є відносно швидким і не надто складним.

Коштовність. BigQuery пропонує як on-demand (на вимогу) так і flat-rate (за фіксованою ставкою) підписки. В будь-якому випадку стягується плата за зберігання даних (0,020 дол. США за ГБ на місяць) і виконання запитів (5 дол. США за ТБ), такі речі, як експорт, завантаження та копіювання даних є безкоштовними.

2.3.3 Amazon Redshift

Amazon Redshift [23] – це хмарна СУБД, орієнтована на побудову і управління сховищами даних. Amazon Redshift [23] чудово підходить для якісної аналітики даних та забезпечує швидке виконання паралельних запитів відповідно до нагальних потреб. Хоч Redshift [23] має ряд авто-керованих інструментів, він є більш самокерованим рішенням, тобто інженерам доведеться витрачати час на керування ресурсами та серверами.

Архітектура. Redshift [23] розроблено на основі SN MPP архітектури. Як показано на рисунку 2.4, він включає кластери сховища даних з обчислювальними вузлами, розділеними на фрагменти вузлів. Система взаємодіє з клієнтськими програмами за допомогою стандартних драйверів JDBC і ODBC. Технологію можна інтегрувати з більшістю існуючих

клієнтських програм на базі SQL, ETL, BI, аналітикою даних та data mining інструментами.

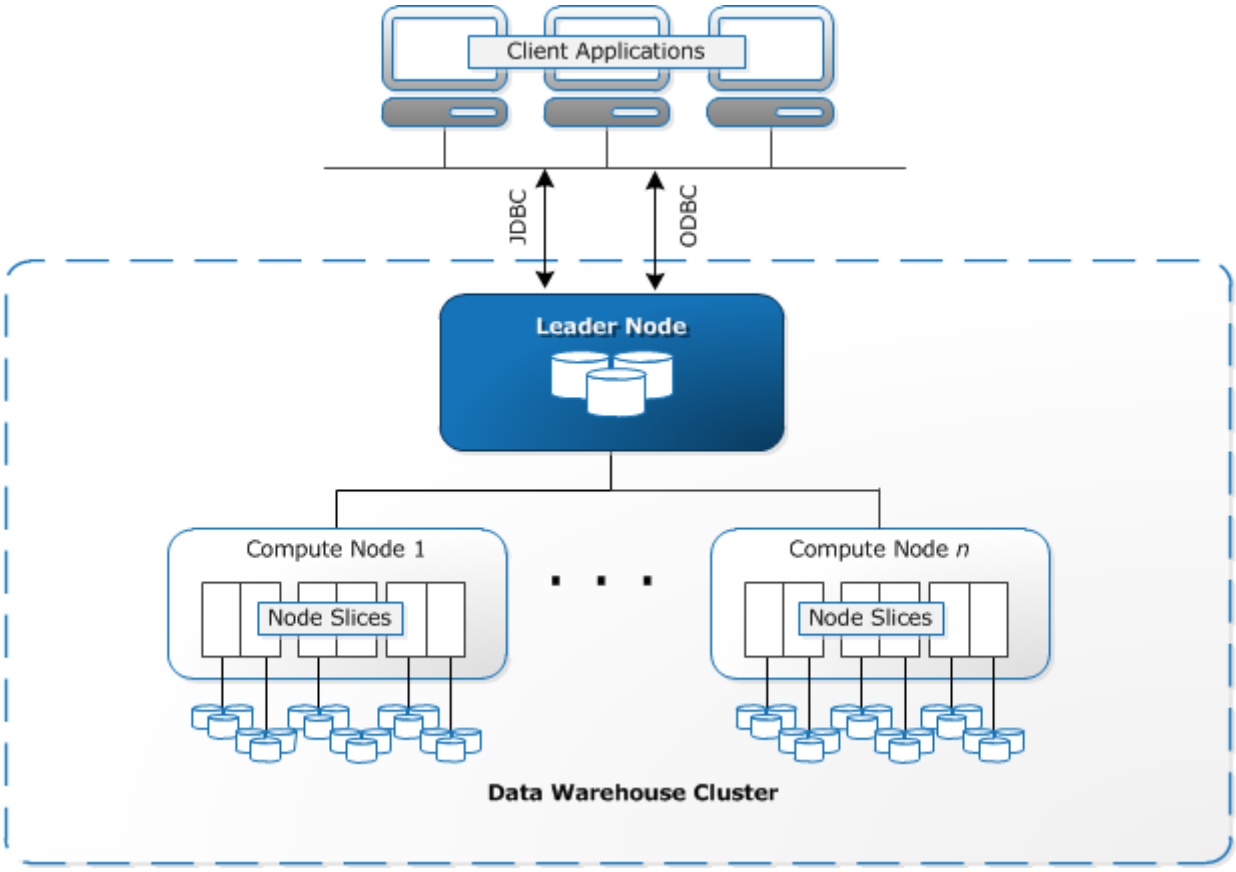


Рис. 2.4 – Архітектура Redshift [24]

Продуктивність і швидкість обробки даних. Загалом продуктивність для більшості типів даних є нормальною, проте вона досить низька при використанні напівструктурованих даних (наприклад, файлів у форматі JSON). Для оптимальної продуктивності користувачам рекомендується використовувати концепцію розподільних ключів. Згідно з тестами Fivetran, Amazon Redshift показав хорошу продуктивність – середня тривалість виконання 99 запитів TPC-DS (кожна TPC-DS складається з 24 таблиць, найбільша з яких містить 4 мільйони рядків даних) становить 8,24 секунд [22].

Масштабованість. Redshift підтримує горизонтальне і вертикальне масштабування. По замовчуванні він налаштований на підтримку 500

одночасних підключень і до 50 одночасних запитів, які виконуються одночасно в одному кластері. При виникненні необхідності в більшій кількості одночасних запитів, Redshift автоматично додає ємність іншого кластера до існуючого. Різні кластери можуть використовуватися для виконання різних задач з тими самими даними.

Інформаційна безпека. Redshift, як і інші сервіси AWS, використовує модель розподіленої відповідальності за безпеку: AWS піклується про безпеку хмари, а відповідальність за безпеку в хмарі – це відповідальність клієнта. Оскільки безпека є найвищим пріоритетом для AWS, доступ до ресурсів Redshift ретельно контролюється на всіх рівнях. Redshift забезпечує відповідність стандартам безпеки, включаючи ISO, PCI, HIPAA BAA і SOC 1,2,3.

Інтеграція зі стороннім ПЗ. Redshift підтримує можливість тісної інтеграції з всією екосистемою AWS, включаючи Amazon S3, Amazon RDS, Amazon DynamoDB, Amazon EMR, AWS Glue і AWS Data Pipeline. Redshift також сумісний з чисельними засобами та інструментами бізнес аналітики, BI та AI.

Завантаження даних. Підтримуючи як підходи ELT/ETL, так і стандартні команди DML, Redshift стверджує, що найефективніший спосіб завантаження даних в сховище - це їх команду COPY. За допомогою цієї команди можна одночасно працювати з кількома потоками даних або ж читати дані з кількох файлів даних одночасно. Платформа також підтримує широкі можливості при роботі з потоковими даними.

Резервне копіювання та відновлення даних. Amazon Redshift дозволяє створювати резервні копії як вручну так і автоматично. Вони зберігаються в Amazon S3 за допомогою зашифрованого з’єднання SSL.

Процес імплементації. Механізм запитів Redshift нагадує інтерфейс PostgreSQL. Тому, при наявності у розробника досвіду використання PostgreSQL або схожої SQL системи, створення та виконання запитів не буде важким завданням. Крім того, Redshift автоматизує деякі з найбільш

трудомістких завдань конфігурації кластерів, таких як резервне копіювання даних, оновлення, реплікація. Проте, для роботи з Redshift, все таки, необхідні кваліфіковані DevOps спеціалісти, які повинні правильно налаштувати кластери та розподілити пам’ять. Тому Redshift відносять до більш самокерованих сховищ даних.

Коштовність. Redshift пропонує різні тарифні плани. При використанні on-demand підходу, плата стягується погодинно за використання обчислювального кластеру (стартує від 0,25 дол. США за годину та залежить від розміру кластеру). Крім того, щомісячно стягується плата за збереження даних.

2.3.4 Azure Synapse (SQL DW)

Azure Synapse SQL DW [25] являє собою хмарну реляційну базу даних з можливістю завантаження та обробки петабайт даних, а також широкими можливостями для звітності а аналітики. Завдяки тісній інтеграції з Microsoft SQL Server рішення є надзвичайно привабливим для компаній які використовують Microsoft SQL Server та бажають перейти на хмарне сховище даних.

Архітектура. Azure Synapse SQL DW [25] побудоване на базі SD MPP архітектури, що надає користувачам можливість виконувати більше сотні паралельних запитів одночасно. Як показано на рисунку 2.5, керуючий вузол отримує команди T-SQL від програм, готує запити для одночасної обробки та надсилає операції дочірнім обчислювальним вузлам.

Продуктивність і швидкість обробки даних. Використання MPP архітектури дозволяє користувачам досить швидко витягувати та візуалізувати бізнес-інсайти. Тести проведені GigaOm, показали, що Azure Synapse демонструє найкращу продуктивність: для виконання 103 запитів знадобилося лише 2.996 секунд, в той час як конкуренти показали набагато гірші результати (Snowflake – 5.793, Redshift – 7.143, BigQuery – 35.283) [19].

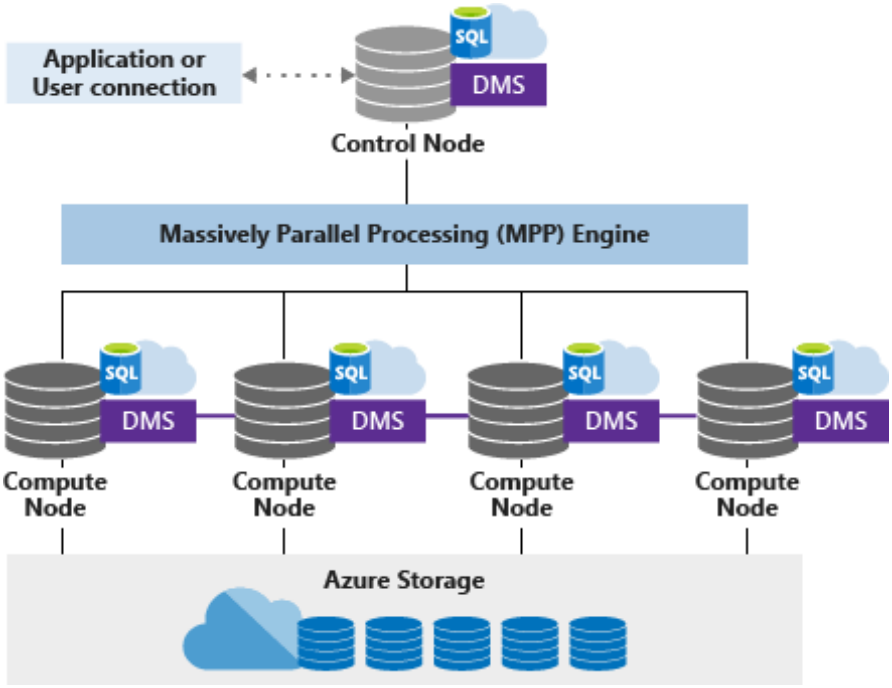


Рис. 2.5 – Архітектура Azure Synapse

Масштабованість. Azure Synapse [25] пропонує як вертикальне так і горизонтальне масштабування відповідно до вимог користувача. Взагалі сервіси Azure [15] є доволі еластичними і зручними у використанні.

Інформаційна безпека. Azure [15] пропонує різноманітні послуги, спрямовані на підвищення рівня інформаційної безпеки. Вони включають керування доступом, безпеку мережі, захист від загроз та захист даних. Сервіс Microsoft пропонує понад 90 сертифікатів відповідності безпеці, зокрема HITRUST, HIPAA, ISO, NIST CSF.

Інтеграція зі стороннім ПЗ. Azure Synapse [25] пропонує широкий набір інструментів інтеграції, що дозволяють підключитися до широкого спектру сторонніх служб. Система без проблем інтегрується з операційними базами даних, інструментами BI та ML.

Завантаження даних. Azure Synapse пропонує понад 95 конекторів, які дозволяють побудувати процеси ETL/ELT практично без необхідності написання коду.

Резервне копіювання та відновлення даних. Azure містить вбудоване рішення (Azure Backup) для створення резервних копій та відновлення даних.

Azure Backup підтримує досить широкий спектр налаштувань резервного копіювання.

Процес імплементації. Для розробників добре знайомих з продуктами Microsoft впровадження сховища даних не стане важкою задачею. Для впевненої роботи з Azure Synapse необхідний практичний досвід використання SQL та Spark. Хоч Azure Synapse дозволяє використовувати ряд безсерверних компоненів, він є доволі самокерованим: для налаштування сховища необхідні кваліфіковані інженери даних. Azure надає інтуїтивно зрозумілий інтерфейс та добре організовану документацію, що сприяє простоті розробки та використання хмарного сховища даних.

Коштовність. Azure Synapse стягує плату за обчислення та збереження даних окремо. Пропонується ряд різноманітних тарифних планів, які дозволяють зекономити кошти та отримати потрібні послуги.

2.4 Порівняння сучасних хмарних сховищ даних

Результати порівняння розглянутих рішень на ринку хмарних сховищ даних узагальнено в таблиці 2.1. Варто зазначити що всі розглянуті рішення чудово підходять для побудови хмарних сховищ даних та користуються значним попитом.

Табл. 2.1 - Порівняння хмарних сховищ даних. Частина 1

	Snowflake	Google BigQuery	Amazon Redshift	Azure Synapse
Архітектура	Гібрид SN та SD MPP	SN MPP	SN MPP	SD MPP
Продуктивність	Висока	Висока	Висока	Найкраща
Масштабованість	Вертикальна та горизонтальна			
Інформаційна безпека	Високий рівень, відповідність чисельним стандартам			

Табл. 2.1 - Порівняння хмарних сховищ даних. Частина 2

	Snowflake	Google BigQuery	Amazon Redshift	Azure Synapse
Інтеграція з ПЗ	Чисельні інструменти аналітики, BI та AI	Екосистема AWS, чисельні інструменти аналітики, BI та AI	Екосистема Google, чисельні інструменти аналітики, BI та AI	Екосистема Azure, чисельні інструменти аналітики, BI та AI
Завантаження даних	ETL/ELT, підтримка потокових даних			
Резервне копіювання та відновлення даних	Наявне (в різних реалізаціях)			
Складність імплементації	Потребує знань SQL та особливостей архітектури	Потребує знань SQL та інструментів ETL	Потребує знань PostgreSQL (або схожих систем)	Потребує знань SQL та Spark
Управління серверами	Більшою мірою безсерверний	Безсерверний	Самокерований	Самокерований
Стягнення плати	На вимогу, по передплаті	На вимогу, за фіксованою ставкою	На вимогу, плата за збереження	Плата за обчислення, плата за збереження

РОЗДІЛ 3. ЗАСОБИ ЗАВАНТАЖЕННЯ ДАНИХ В SNOWFLAKE ТА ПІДСТАВИ ДЛЯ РОЗРОБКИ СИСТЕМИ ДЛЯ УПРАВЛІННЯ НИМИ

3.1 Архітектура Snowflake

Як зазначалося раніше, архітектура Snowflake [1] являє собою комбінацію SN та SD архітектур. Snowflake [1] використовує центральне сховище даних, доступне з усіх обчислювальних вузлів платформи, як в SD архітектурі. При цьому Snowflake [1] використовує обчислювальні кластери MPP для виконання запитів, в яких кожен вузол зберігає частину всього набору даних локально, як в SN архітектурі. Такий гібрид архітектур дозволяє поєднати простоту керування даними SD архітектури з ефективністю та перевагами масштабування SN архітектури.

Як показано на рисунку 3.1, архітектура Snowflake містить три основні шари: збереження даних, обробка запитів, хмарні сервіси.

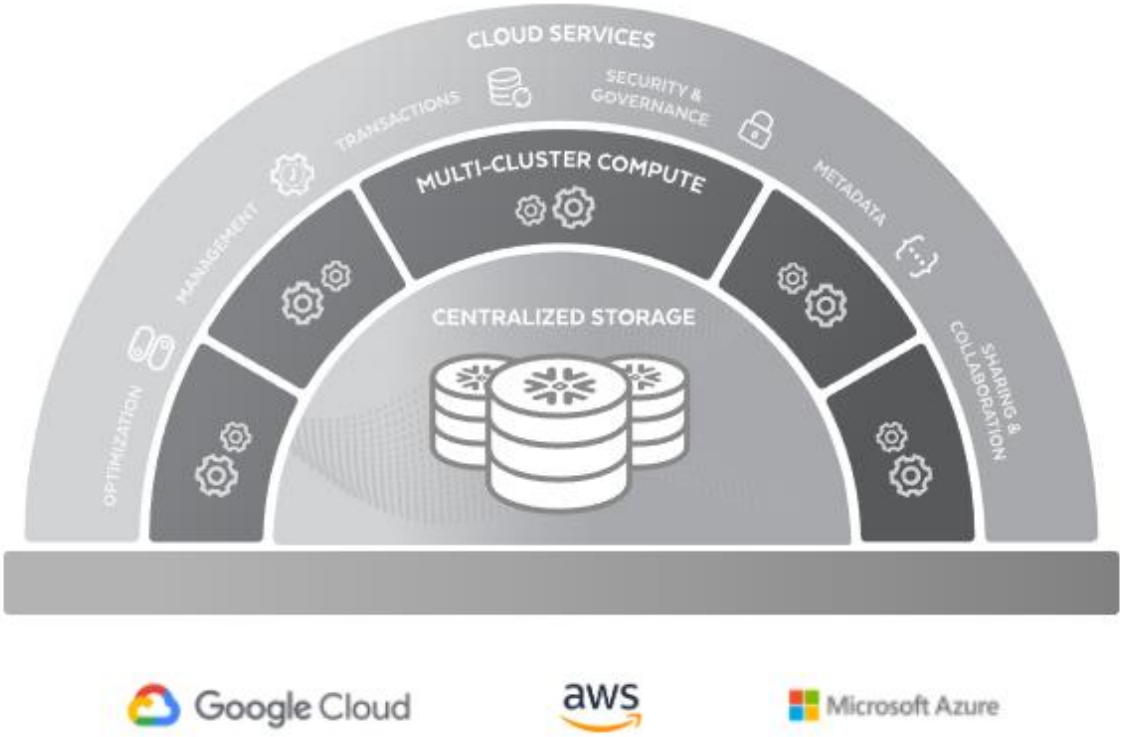


Рис. 3.1 – Архітектура Snowflake [27]

Шар збереження даних. Snowflake розділяє дані на численні крихітні оптимізовані стиснуті розділи. Snowflake містить масштабований тип хмарного BLOB сховища, яке слугує для зберігання структурованих і напівструктурованих даних (включаючи JSON, AVRO і Parquet). Snowflake зберігає та керує даними в хмарі за допомогою підходу властивого SD архітектурам, що робить керування даними простим. Елементи SN архітектури гарантують, що користувачу не доведеться турбуються про розподіл даних між кількома вузлами. Snowflake приховує об’єкти даних користувача і робить їх доступними лише через SQL-запити через шар обробки запитів, що дозволяє підвищити надійність системи та зменшити ймовірність виникнення помилок при неправильному керуванні ресурсами.

Шар обробки запитів (або обчислювальний шар). Для виконання запитів Snowflake використовує, так звані, віртуальні сховища (Virtual Warehouses) – обчислювальні кластери MPP, що складаються з багатьох вузлів з процесорами і пам’яттю, розміщеними в акаунті Snowflake на обраному хмарному провайдері. Snowflake дозволяє створити багато віртуальних сховищ для різних вимог залежно від робочого навантаження. Кожне віртуальне сховище може використовувати один рівень зберігання. Віртуальне сховище в більшості випадків має власний обчислювальний кластер і не взаємодіє з іншими віртуальними сховищами. Віртуальні сховища можуть бути автоматично відновлені та автоматично призупинені, легко розширюються та підтримують автоматичне масштабування.

Шар хмарних сервісів. Цей шар містить усі операції, які координуються в Snowflake, такі як аутентифікація, безпека, керування метаданими та оптимізація запитів. Хмарний сервіс в Snowflake – це обчислювальний ресурс (без збереження стану), який працює в різних зонах доступності та використовує високодоступну та корисну інформацію. Шар хмарних сервісів забезпечує клієнтський інтерфейс SQL для операцій з даними (DDL, DML). Перш ніж передавати запит шару обчислень, шар хмарних сервісів виконує його оптимізацію. Також цей рівень зберігає

важливі метадані необхідні для підвищення ефективності та оптимізації запитів.

Такий архітектурний розподіл Snowflake [1] на шари дозволяє масштабування будь-якого шару незалежно від інших. Можна, наприклад, еластично масштабувати шар збереження даних та оплачувати його використання незалежно від інших шарів. Коли для пришвидшення обробки запитів та підвищення продуктивності можна зручно можна розгорнути нові віртуальні сховища (виконати горизонтальне масштабування) або підвищити розмір існуючих віртуальних сховищ (виконати вертикальне масштабування).

Також така архітектура зробила можливим обирати хмарного провайдера для розгортання Snowflake [1], що для багатьох клієнтів виявилося суттєвою перевагою. Взагалі, Snowflake [1] своєю популярністю завдячує вражаючій гнучкості та еластичності.

3.2 Організація даних в Snowflake

Всі дані в Snowflake зберігаються в базах даних. Кожна база даних складається з однієї або кількох схем, які є логічними угрупованнями об’єктів бази даних, таких як таблиці та подання (views). Snowflake не накладає жорстких обмежень на кількість баз даних, схем (в межах бази даних) або об’єктів (в межах схеми), які можна створити.

Snowflake зберігає дані в таблицях бази даних, логічно структурованих як колекції стовпців і рядків. Для найбільш ефективного використання таблиць Snowflake (особливо великих таблиць), необхідно мати розуміння фізичної структури, що стоїть за логічною структурою даних.

Дані в таблицях Snowflake автоматично поділяються на мікророзділи. Кожен мікророзділ містить від 50 МБ до 500 МБ нестиснених даних (фактичний розмір у Snowflake менший, оскільки дані завжди зберігаються стиснутими). Групи рядків у таблицях виокремлюються в окемі

мікророзділи, організовані у вигляді стовпців. Такий розмір і структура дозволяють надзвичайно ефективно обрізати (data pruning) дуже великі таблиці, які можуть складатися з мільйонів або навіть сотень мільйонів мікро-розділів.

Snowflake зберігає метадані про всі рядки, що зберігаються в мікророзділі, включаючи:

- Діапазон значень для кожного зі стовпців мікророзділу.
- Кількість різних значень.
- Додаткові властивості, що використовуються як для оптимізації пам’яті, так і для ефективної обробки запитів.

Для максимально ефективної роботи з великими таблицями Snowflake забезпечує кластеризацію даних по мікророзділам. При створенні нових мікророзділів Snowflake записує спеціальні метадані про кластеризацію для кожного мікророзділу. Ці метадані дозволяють максимально ефективно виконувати аналітичні запити до даних.

3.3 Інструменти та засоби Snowflake для завантаження даних

Одним з ключових етапів використання хмарних сховищ даних є процес завантаження даних в сховище. Для цього Snowflake пропонує широкий вибір різноманітних засобів та інструментів, що дозволяє обрати найбільш підходящий та зручно користуватися всіма перевагами хмарного сховища даних.

3.3.1 Масове завантаження (Bulk Loading)

Так як звичайні INSERT INTO вирази мови SQL виконуються по-одному, що дуже негативно впливає на продуктивність, Snowflake пропонує використовувати масове завантаження даних (Bulk Loading). Bulk Loading дозволяє взяти велику кількість даних і вставити їх у базу даних одним

пакетом. Масове завантаження даних у Snowflake забезпечує пакетне завантаження даних із файлів у хмарному сховищі, наприклад з AWS S3.

Якщо файли даних наразі не знаходяться в хмарному сховищі, є можливість скопіювати файли даних з локального комп'ютера в проміжну область хмарного сховища, перш ніж завантажувати їх у Snowflake. Snowflake дозволяє як створювати внутрішнє хмарне сховище (яке буде знаходитися в акаунті Snowflake), так і інтегрувати власне хмарне сховище (воно має бути на тому ж хмарному провайдері, що і Snowflake, Наприклад AWS S3, Azure Blob Storage). Об'єкт Snowflake, який інкапсулює дані про сховище називається STAGE. Файли даних передаються з локальної машини у сховище, а потім завантажуються в таблиці за допомогою команди COPY.

Snowflake підтримує більшість поширених форматів файлів, які використовуються для збереження даних:

- Файли з роздільником (підтримується будь-який дійсний роздільник; за замовчуванням – кома (CSV); для файлів з роздільником наявні широкі можливості для конфігурації)
- JSON, JSONL
- XML
- Avro (включаючи автоматичне виявлення та обробку поетапних файлів Avro, які були стиснуті за допомогою Snappy)
- ORC (включаючи автоматичне виявлення та обробку поетапних файлів ORC, які були стиснуті за допомогою Snappy або zlib)
- Parquet (включаючи автоматичне виявлення та обробку поетапних файлів Parquet, які були стиснуті за допомогою Snappy)

Під час розміщення нестиснутих файлів на стейджі Snowflake файли автоматично стискаються за допомогою gzip, якщо стиснення не вимкнено явно. Snowflake може автоматично виявляти методи стиснення gzip, bzip2, deflate та raw_deflate. Автовизначення поки що не підтримується для brotli та

zstandard. Тому під час створення або завантаження файлів, стиснутих будь-яким із цих методів, необхідно явно вказати метод стиснення, який використовувався.

Для масивного завантаження даних, Snowflake рекомендує розбити дані на фали розміром від 10 МБ до 100 МБ в стиснутому вигляді. Коли файли потрапляють в стейдж (як внутрішнє так і інтегроване зовнішнє), для них генеруються метадані, які дозволяють виконувати SELECT запити до даних в стейджі. Це також дозволяє виконувати базові перетворення даних при завантаженні даних в таблиці за допомогою команди COPY. Наприклад, для даних в форматі JSON можна їх аналізувати (вказуючи шлях по структурі JSON рекорду) та копіювати потрібні значення в потрібні колонки таблиці. Операції які підтримуються для команди COPY:

- Зміна порядку колонок
- Пропуск колонки
- Приведення типів
- Обрізання текстових рядків, які перевищують довжину цільового стовпця

Масове завантаження даних використовує віртуальне сховище, створене користувачем, вказане в команді COPY. Від користувачів вимагається відповідний розмір віртуального сховища, щоб коректно опрацювати очікуване навантаження.

3.3.2 Сервіс Snowpipe

На практиці часто виникає потреба у безперервному завантаженні даних до сховища даних. Наприклад, якийсь сервіс генерує певний об’єм даних і вивантажує його щохвилини. Для постійного завантаження даних в базу довелося б створювати якусь окрему програмну сутність, яка б по розкладу переміщала файл в стейдж і ініціювала виконання COPY команди. Такий підхід працює, проте він містить ряд недоліків, так як для нього

використовуються ресурси віртуального сховища, якому б довелося встановлювати великий розмір для швидкої обробки.

Для рішення цієї задачі Snowflake пропонує спеціальний сервіс Snowpipe. Snowpipe – це хмарний сервіс Snowflake з автомасшабуванням, який забезпечує безперервне завантаження даних у сховище даних Snowflake з внутрішнього або зовнішнього стейджу.

Існує 2 варіанти роботи з Snowpipe. Перший варіант – використовувати Snowpipe з зовнішнім стейджом AWS S3, де визначається сповіщення про появу нових файлів на S3, які при появі нових файлів надсилаються в SQS (черга сповіщень) Snowflake акаунту. Завдяки цьому файли автоматично підбираються Snowpipe і завантажуються в сховище даних. Для використання цієї опції при створенні об’єкту Snowpipe необхідно вказати опцію auto_ingest=true. Рисунок 3.2 схематично демонструє використання Snowpipe з опцією auto_ingest=true.

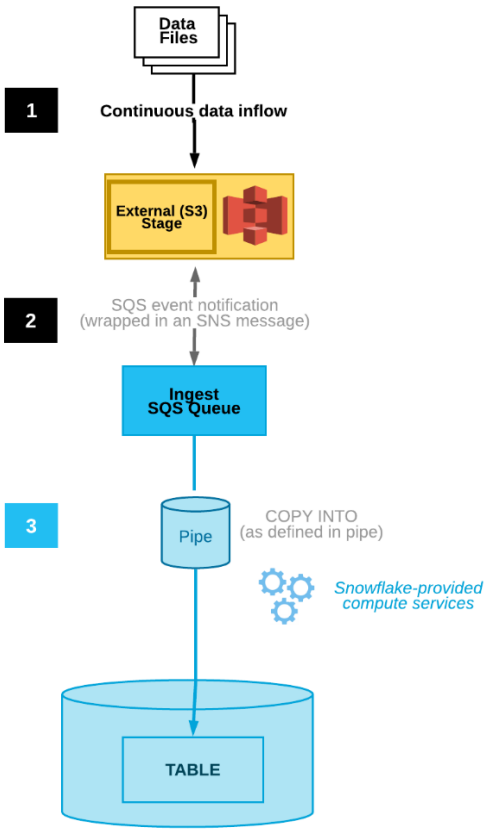


Рис. 3.2 – Схема безперервного завантаження даних за допомогою Snowpipe з опцією auto_ingest=true [28]

Другий варіант – створити власну інтеграцію з Snowpipe за допомогою REST API. Тобто створити програму, яка буде викликати відповідний ендпоїнт REST API для завантаження нового файлу.

Snowpipe використовує обчислювальні ресурси, надані Snowflake (тобто для користувача він є безсерверним). Ці ресурси, надані Snowflake, автоматично масштабуються відповідно до поточних потреб. Плата за використання цих ресурсів стягується посекундно і залежить лише від фактичного робочого навантаження.

Snowpipe використовує вираз COPY, який підтримує всі ті самі трансформації, що і для масового завантаження даних. Для Snowpipe рекомендований обсяг файлів – до 250 МБ в стиснутому вигляді. Практично у всіх випадках дані будуть завантажені протягом хвилини після ініціалізації роботи сервісу.

Нижче наведено порівняння масового завантаження (Bulk Loading) та завантаження за допомогою сервісу Snowpipe [32]:

Автентифікація:

- Bulk Loading: Покладається на параметри безпеки, які підтримуються клієнтом для автентифікації та ініціювання сеансу користувача.
- Snowpipe: Під час виклику ендпоїнтів REST потрібна автентифікація пари ключів за допомогою JSON Web Token (JWT). JWT підписуються за допомогою пари відкритий/приватний ключ із шифруванням RSA.

Історія завантаження:

- Bulk Loading: Зберігається в метаданих цільової таблиці протягом 64 днів.
- Snowpipe: Зберігається в метаданих об’єкту Snowpipe протягом 14 днів.

Транзакційність:

- Bulk Loading: Завантаження завжди виконуються в одній транзакції. Дані вставляються в таблицю так само як і іншими операторами SQL.
- Snowpipe: Завантаження об’єднуються або розбиваються на одну чи кілька транзакцій на основі кількості та розміру рядків у кожному файлі даних. Рядки частково завантажених файлів (на основі параметра копіювання ON_ERROR) також можуть бути об’єднані або розділені на одну або кілька транзакцій.

Обчислювальні ресурси:

- Bulk Loading: Використовує віртуальне сховище, яке створює та налаштовує користувач.
- Snowpipe: Використовує обчислювальні ресурси, надані Snowflake.

Ціна:

- Bulk Loading: Плата виставляється за час, протягом якого віртуальне сховище є активним.
- Snowpipe: Плата виставляється лише за час безпосередньої роботи ресурсів, наданих Snowflake.

Snowpipe дуже корисний і зручний для автоматизації процесу завантаження даних і активно використовується на практиці. Самі творці Snowflake виділяють Snowpipe як найкраще рішення (best practice) для завантаження даних в Snowflake. Тому, розроблювана система буде сфокусована саме на управлінні об’єктами Snowpipe.

3.3.3 Альтернативні способи завантаження даних

Крім вже зазначених інструментів для завантаження даних, Snowflake також пропонує спеціально розроблений Kafka конектор [29], який дозволяє приєднатися до Apache Kafka [30] серверу, зчитувати дані з одного або кількох топіків і завантажувати їх в Snowflake таблиці. Проте, насправді, під капотом Kafka конектор [29] використовує сервіс Snowpipe, тому часто значно дешевше і ефективніше використовувати сервіс Snowpipe напряму.

Також Snowflake надає можливість виконувати аналіз даних без їх завантаження в таблиці шляхом створення зовнішніх таблиць (External tables) (схоже на концепцію озера даних). Зовнішні таблиці дозволяють запитувати наявні дані, збережені у зовнішньому хмарному сховищі (наприклад на AWS S3), для аналізу без попереднього завантаження їх у Snowflake. Набори даних, матеріалізовані в Snowflake за допомогою матеріалізованих представлень, доступнмх лише для читання.

Таке рішення чудово підходить для організацій, які мають велику кількість даних, що зберігаються у зовнішньому хмарному сховищі, і хочуть виконувати аналіз лише частини даних; наприклад, аналізувати найбільш свіжі дані. Користувачі можуть створювати матеріалізовані представлення про підмножини цих даних для підвищення ефективності виконання запитів.

3.4 Проблеми управління та моніторингу об’єктів Snowpipe

Попри те, що Snowpipe є основним рекомендованим засобом завантаження даних в Snowflake, розробники не потурбувалися про розробку зручної системи управління та моніторингу об’єктів Snowpipe.

Об’єкт Snowpipe – це об’єкт в логічній структурі Snowflake, який зберігає логіку завантаження даних в Snowflake.

Стандартний безальтернативний веб-інтерфейс Snowflake – Snowsight, дозволяє лише переглянути означення об’єкту та видалити його (як показано на рисунку 3.3), при цьому не надаючи інтерфейс для перегляду історії

завантаження даних та перевірки поточного стану об’єкту Snowpipe. Всі дії над об’єктами Snowpipe, такі як перегляд історії, зупинка та активація, можливо виконати лише за допомогою консольного інтерфейсу, шляхом написання запитів, які не є інтуїтивно зрозумілими та потребують певного рівня компетенції користувача.

Рис. 3.3 – Фрагмент користувацького інтерфейсу Snowsight для роботи з об’єктами Snowpipe

Крім того, якщо об’єкт Snowpipe створений без опції auto-ingest, як консольний так і графічний веб-інтерфейс Snowsight не надають можливості ініціювати завантаження. Ініціація завантаження доступна лише за допомогою виклику певного ендпоїнту Snowpipe REST API, що не є простою задачею, адже Snowpipe REST API підтримує лише key-pair автентифікацію та вимагає генерувати JWT.

Ще одним суттєвим недоліком є те, що історія завантаження даних за допомогою Snowpipe в системних табличках Snowflake зберігається лише протягом 14 днів, після чого вона безповоротно видаляється. Крім того, відповідно до документації Snowflake, потрапляти в системну таблицю історія завантаження може з затримкою до 2 годин. А Snowpipe REST API, дозволяє переглянути інформацію лише про останні 10000 завантаження даних).

Також, так як історія завантаження зберігається в метаданих об’єкту Snowpipe, при його видалені або перестворенні вся його історія завантаження втрачається, як і історія про існування цього об’єкту. Тобто, в разі

виникнення якихось важливих помилок в завантаженні даних, які потребують дослідження, користувачу може не вистачити інформації для визначення джерела помилки, адже історія не буде збережена.

Таким чином, з точки зору збереження історії існування об’єктів Snowpipe та історії завантаження даних, в Snowflake наявні не лише проблеми відсутності зручного користувацького інтерфейсу, а й системна проблема збереження цих даних.

Отже, можна виділити наступі проблеми, які має вирішувати розроблювана система:

- Відсутність інтуїтивно-зрозумілого графічного інтерфейсу користувача для перегляду інформації про стан об’єктів Snowpipe.
- Відсутність інтуїтивно-зрозумілого графічного інтерфейсу користувача для зміни стану об’єктів Snowpipe (призупинення / активація).
- Відсутність інтуїтивно-зрозумілого графічного інтерфейсу користувача для ініціації завантаження даних за допомогою об’єктів Snowpipe.
- Збереження історії існування об’єктів Snowpipe.
- Збереження повної історії завантаження даних за допомогою об’єктів Snowpipe.

РОЗДІЛ 4. ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

Розроблювана система складається з двох частин:

- Серверний застосунок – ASP.NET Core Web API
- Клієнтський застосунок (SPA) – Blazor WASM

Обидва застосунки реалізовані на платформі .NET 6 [3], мова програмування – C# [4]. Для збереження даних використовується база даних Microsoft SQL Server 2019 [5], взаємодія з якою відбувається за допомогою Entity Framework Core. Для створення зручної та красивої розмітки клієнтського застосунку використовувався фреймворк Bootstrap [36].

4.1 Платформа .NET

Платформа .NET [3] - одна з найпопулярніших платформ розробки програмного забезпечення, розроблена компанією Microsoft. Вона включає ряд інструментів, технологій та середовищ для розробки, виконання та управління додатками на різних платформах, таких як Windows, Linux та macOS.

Актуальною версією платформи є .NET 7, проте розробка проводилася на версії .NET 6, адже вона є LTS (long time support) і буде підтримуватися довше, аніж .NET 7, який не є LTS.

Для реалізації була обрана платформа .NET, враховуючи наступні її переваги:

- Кросплатформність. Тобто, .NET надає розробникам можливість створювати програми, які можуть працювати на різних операційних системах, таких як Windows, Linux і macOS.
- .NET підтримує кілька мов програмування, таких як C#, F#, Visual Basic і інші. Це дає розробникам вибір мови, на якій їм зручніше розробляти свої додатки, і можливість використання різних мов у різних частинах одного додатку.

- Додатки, розроблені на платформі .NET, відомі своєю високою продуктивністю та швидкодією. Платформа має вбудовану оптимізацію коду, підтримку JIT (Just-In-Time) компіляції та можливість використання багатопотоковості для досягнення високої продуктивності програм.
- Платформа .NET має велику кількість класів та бібліотек, що допомагають розробникам ефективно виконувати різноманітні завдання, такі як робота з базами даних, мережевою комунікацією, графікою, безпекою, шифруванням, тестуванням та іншим.
- .NET забезпечує широкі можливості для розробки веб-застосунків, включаючи підтримку ASP.NET, ASP.NET Core і Blazor, що дозволяє розробникам ефективно створювати веб-застосунки різного рівня складності та архітектури.

4.2 Мова програмування C#

Мова програмування C# [4] – це основна мова програмування платформи .NET. Вона була представлена в 2000 році як частина розробки платформи .NET, і з тих пір стала однією з найпопулярніших мов програмування для розробки різноманітних типів програм, таких як десктопні застосунки, веб-застосунки, мобільні додатки та навіть SPA.

Своїй популярності вона завдячує наступним перевагам:

- C# є об'єктно-орієнтованою мовою, яка підтримує всі звичні концепції та принципи ООП.
- C# містить велику кількість «синтаксичного цукру» і постійно оновлюється та вдосконалюється (за останні 5 років вийшло аж 5 версій мов)

- C# має звичний C-подібний синтаксис, схожий на C++ та Java, з великою кількістю спільних елементів, таких як робота з класами, інтерфейсами, делегатами, подіями та іншими конструкціями.
- C# повністю інтегрована з платформою .NET, що дозволяє використовувати всі функції та бібліотеки .NET.
- C# має зручну вбудовану підтримку асинхронного програмування, що дозволяє створювати ефективні та швидкодіючі додатки, які можуть ефективно виконувати асинхронні операції, такі як мережеві виклики, робота з файлами та інші, без блокування основного потоку виконання.

4.3 Технологія ASP.NET Core

ASP.NET Core [34] — це кросплатфений високопродуктивний фреймворк з відкритим кодом призначений для веб-розробки. ASP.NET Core дозволяє створювати різноманітні веб-застосунки та сервіси за різними схемами: MVC, Web API, Web Pages.

Для розробки Web API був обраний фреймворк ASP.NET Core, враховуючи наступні переваги:

- ASP.NET Core дозволяє використовувати звичний підхід MVC для розробки Web API. Контролери визначають дії (actions), які можуть бути викликані за допомогою HTTP-запитів, а моделі та представлення використовуються для обробки даних та генерації відповідей.
- ASP.NET Core побудований на основі модульної архітектури, що дозволяє зручно конфігурувати та розширювати конвеєр обробки запитів, додаючи власні мідлвеари та фільтри.

- ASP.NET Core має вбудовану підтримку документування веб API за допомогою стандарту OpenAPI (раніше відомого як Swagger). Це дозволяє автоматично генерувати документацію API на основі коду, що спрощує розробку, тестування та використання API.
- ASP.NET Core надає широкі можливості для налаштування авторизації та аутентифікації Web API. Він забезпечує вбудовану підтримку різних механізмів авторизації, таких як JWT (JSON Web Tokens), OAuth, інтеграцію з соціальними мережами, тощо.
- ASP.NET Core має вбудовану систему ін'єкції залежностей (Dependency Injection), що дозволяє зручно управляти залежностями між різними компонентами Web API, таким чином сприяючи розширюваності та тестованості коду.

4.4 Технологія Blazor WASM

WebAssembly [33] (або скорочено WASM) – незалежний від браузера універсальний низькорівневий проміжний код для виконання в браузері застосунків, скомпільованих з різних мов програмування (в тому числі і C#). Він створений для того, щоб дозволити виконання важких обчислень в браузері з високою швидкістю та ефективністю, що дозволяє розширити можливості веб-додатків та забезпечити кращий досвід користувача. Наразі практично всі сучасні браузери підтримують технологію WebAssembly.

Blazor WASM [35] (WebAssembly) - це одна з технологій, розроблених компанією Microsoft, яка дозволяє розробникам створювати веб-додатки з використанням C# та .NET, що виконується безпосередньо в браузері. Ця технологія в основному використовується для розробки клієнтської частини веб-додатків SPA (Single Page Applications), без необхідності розгортання серверної частини.

Основні можливості технології:

- Компіляція C# коду до WebAssembly: Код C# компілюється в мову низького рівня WebAssembly, яка може виконуватися безпосередньо в браузері.
- Застосування Razor: Розмітка Razor, яка активно використовується в ASP.NET, може бути використана в Blazor WASM для створення візуальних елементів користувацького інтерфейсу, таких як компоненти, сторінки, форми та інші елементи.
- Компонентна модель: Blazor WASM заснований на компонентній моделі, яка дозволяє створювати компоненти, повторно їх використовувати і керувати їхнім життєвим циклом, подіями та станом.
- Двосторонній зв'язок: Blazor WASM надає можливість взаємодії між клієнтською та серверною частинами додатка, включаючи відправку запитів на сервер і отримання оновлень стану в режимі реального часу.
- Серверні опції: Blazor WASM також підтримує можливість розгортання серверної частини, де весь код C# виконується на сервері, а клієнтська частина взаємодіє з сервером за допомогою сокетів SignalR.
- Зручна розробка: Розробка додатків на Blazor WASM може вестися з використанням звичних інструментів розробки .NET, таких як Microsoft Visual Studio або Visual Studio Code. Це дозволяє розробникам використовувати знайомий стек технологій .NET для створення веб-додатків на Blazor WASM.

4.5 Фреймворк Bootstrap

Bootstrap [36] - відкрита веб-технологія, розроблена Twitter, яка дозволяє розробникам створювати ефективні та стильні веб-інтерфейси. Вона базується на HTML, CSS та JavaScript, і надає широкий набір готових компонентів та стилів, які можна використовувати для побудови сучасних веб-застосунків та сайтів.

РОЗДІЛ 5. РЕАЛІЗАЦІЯ СИСТЕМИ

5.1 Задачі розроблюваної системи та їх вирішення

Зважаючи на проблеми наявної системи управління об’єктами Snowpipe, розглянуті в розділі 3.4, розроблювана система має виконувати наступні завдання:

- Зберігати та постійно оновлювати історію існування об’єктів Snowpipe.
- Зберігати та постійно оновлювати історію завантаження даних за допомогою об’єктів Snowpipe.
- Надати інтуїтивно-зрозумілий графічний веб-інтерфейс для перегляду та зміни стану об’єктів Snowpipe.
- Надати інтуїтивно-зрозумілий графічного інтерфейсу користувача для ініціації завантаження даних за допомогою об’єктів Snowpipe з вимкненою опцією auto_ingest.

Для вирішення задачі збереження історії існування об’єктів Snowpipe та їх історії завантаження даних була створена реляційна база даних, структура якої буде розглянута в наступних підрозділах.

Для надання доступу до цієї інформації було створено серверний застосунок, реалізований як ASP.NET Core Web API, який забезпечує набір ендпоїнтів для отримання інформації та маніпуляції об’єктами Snowpipe.

Для вирішення задачі постійного оновлення історії існування об’єктів Snowpipe та їх історії завантаження даних, в серверний застосунок додано сервіс, який за заданим розпорядком актуалізує дані, збережені у базі.

Для надання зручного інтуїтивно-зрозумілого графічного інтерфейсу було розроблено клієнтський застосунок, який надає користувачам зручний веб-інтерфейс для взаємодії з серверним застосунком.

Розроблювану систему назвемо SnowpipeTracker.

5.2 Схема роботи системи

Система SnowpipeTracker розрахована на багатьох користувачів, які створюють в ній профілі – об’єкти які зберігають дані, необхідні для авторизації в Snowflake для витягнення даних про об’єкти Snowpipe. Користувач може створювати, оновлювати та видаляти доступні йому профілі. Також передбачена можливість поділитися доступом до профілю з іншим користвачем.

Кожний профіль автоматично синхронізує дані про об’єкти Snowpipe в зоні його видимості та історію завантаження файлів. Процес синхронізації відбувається наступним чином:

1. Виконується запит до Snowflake про поточний стан всіх наявних об’єктів Snowpipe;
2. Для кожного наявного об’єкту Snowpipe його стан зарівнюється зі станом, збереженим в базі даних SnowpipeTracker:
    a. Якщо об’єкт не був присутній в системі SnowpipeTracker, то він додається в систему як активний Snowpipe;
    b. Якщо об’єкт присутній в системі SnowpipeTracker, то його стан оновлюється;
    c. Якщо об’єкт присутній в системі SnowpipeTracker, проте він був перестворений в Snowflake, то минула версія об’єкту Snowpipe помічається як історична, а нова додається в систему;
3. Всі об’єкти Snowpipe, наявні в системі SnowpipeTracker, про які не було отримано інформації з Snowflake, помічаються як історичні;
4. Для кожного активного об’єкту Snowpipe в системі SnowpipeTracker запитується інформація про історію його завантаження та зберігається в системі.

Таким чином, виконуючи синхронізацію через певний конфігурований проміжок часу, система SnowpipeTracker зберігає історію існування об’єктів Snowpipe та історію завантаження даних.

Процес синхронізації всіх об’єктів Snowpipe, окремого об’єкту Snowpipe та історії завантаження окремого об’єкту Snowpipe може бути окремо ініційована користувачем під час активної взаємодії з системою.

5.3 Архітектура системи

Розроблювана система використовує клієнт-серверну архітектуру, тобто система поділяється на два рішення: серверний застосунок ASP.NET Core Web API та клієнтський застосунок на Blazor WASM.

Таке розділення зумовлене в першу чергу орієнтованістю розроблюваної системи. Цільова аудиторія системи – розробники та технічні фахівці, у яких може виникнути необхідність інтегрувати власну систему з розроблюваною. Завдяки тому, що серверний застосунок повністю незалежний від клієнтського та надає RESTful API, таку інтеграцію виконати буде неважко.

Крім того, клієнт-серверна архітектура дозволяє масштабувати систему, розподіляючи навантаження між клієнтами та серверами. Це дозволяє системі легко впоратися з більшим числом користувачів, забезпечуючи швидкодію та високу продуктивність.

Також клієнт-серверна архітектура дозволяє розширювати функціональність системи, додавати нові функції на серверній стороні, без необхідності змінювати клієнтську частину. Це дозволяє швидко розвивати систему та впроваджувати нові можливості, забезпечуючи гнучкість розробки та оновлення.

5.4 Структура бази даних

База даних містить 3 основні таблиці, які містять інформацію про профілі користувачів, об’єкти Snowpipe та історію завантаження.

Так, як система розрахована на багатьох користувачів, то база також містить набір таблиць бібліотеки ASP.NET Core Identity [37], яка надає зручну та безпечну інфраструктуру для збереження інформації про користувачів та значно полегшує впровадження авторизації та автентифікації.

Користувачі зареєстровані в системі можуть під’єднати свій аккаунт Snowflake шляхом створення профілю, детальна інструкція як це зробити буде надана в розділі 6.

Таблиця Profiles містить дані необхідні для доступу до аккаунту Snowflake для подальшої взаємодії з ним. Детальний опис всіх полів таблиці поданий в таблиці 5.1.

Табл. 5.1 – Опис полів таблиці Profiles

Стовпець	Тип	Опис
Id	int	Ідентифікатор профілю
Name	nvarchar(MAX)	Назва профілю
SfAccount	nvarchar(MAX)	Ідентифікатор Snowflake акаунту користувача
SfUser	nvarchar(MAX)	Юзер в Snowflake
SfRole	nvarchar(MAX)	Роль в Snowflake
SfWarehouse	nvarchar(MAX)	Віртуальне сховище для виконання дій в Snowflake
SfDatabase	nvarchar(MAX)	База даних в Snowflake
PrivateKey	nvarchar(MAX)	Приватний RSA ключ для авторизації в Snowflake
PublicKey	nvarchar(MAX)	Публічний RSA ключ для авторизації в Snowflake

Таблиця Snowpipes містить дані про об’єкти Snowpipe в системі. Більшість полів відображають стан об’єкту Snowpipe, який береться з двох джерел в Snowflake: з системного представлення INFORMATION_SCHEMA.PIPES та з функції SYSTEM$PIPE_STATUS. Також наявно декілька полів, необхідних для роботи системи SnowpipeTracker. В Snowflake ключем для об’єкту Snowpipe виступає комбінація бази, схеми і назви. Опис полів таблиці наведено в таблиці 5.2:

Табл. 5.2 – Опис полів таблиці Snowpipes. Частина 1

Стовпець	Тип	Опис
Id	int	Ідентифікатор об’єкту Snowpipe
Name	nvarchar(MAX)	Назва об’єкту Snowpipe
Schema	nvarchar(MAX)	Схема в Snowflake
Database	nvarchar(MAX)	База в Snowflake
Definition	nvarchar(MAX)	Опис об’єкту (інструкція створення)
IsAutoIngest	bit	Чи ввімкнена опція auto_ingest
IsDeleted	bit	Чи є об’єкт історичним чи акутальним
TrackingEnabled	bit	Чи треба автоматично оновлювати історію зантаження в системі SnowpipeTracker
Tracked	datetime2(7)	Час останньої синхронізації історії завантаження в системі SnowpipeTracker
Created	datetime2(7)	Час створення в Snowflake
Updated	datetime2(7)	Час оновлення в Snowflake
TargetTableName	nvarchar(MAX)	Таблиця, в яку налаштоване завантаження даних

Табл. 5.2 – Опис полів таблиці Snowpipes. Частина 2

Стовпець	Тип	Опис
StageName	nvarchar(MAX)	Stage, з якого об’єкт бере файли для завантаження
Error	nvarchar(MAX)	Помилка
LastIngestedTimestamp	datetime2(7)	Час останнього завантаження
LastPipeErrorTimestamp	datetime2(7)	Час останньої помилки при завантаженні
LastPipeFaultTimestamp	datetime2(7)	Час останньої критичної помилки під час завантаження
NotificationChannelName	nvarchar(MAX)	Посилання на внутрішню чергу об’єкту Snowpipe
ExecutionState	nvarchar(MAX)	Стан об’єкту Snowpipe
ProfileId	int	Зовнішній ключ для зв’язку з профілями
Synchronized	datetime2(7)	Час останньої синхронізації в системі SnowpipeTracker
PendingFileCount	int	Кількість файлів в черзі

Таблиця TrackingInfos зберігає інформацію про завантаження файлу в таблицю об’єктом Snowpipe. Практично всі поля таблиці беруть інформцію з ендпоїнту “loadHistoryScan” Snowpipe REST API. Опис стовпців подано в таблиці 5.3:

Табл. 5.3 – Опис полів таблиці TrackingInfos. Частина 1

Стовпець	Тип	Опис
Id	int	Ідентифікатор завантаження
SnowpipeId	int	Id об’єкта Snowpipe
Path	nvarchar(MAX)	Відносний шлях до завантаженого файлу з даними

Табл. 5.3 – Опис полів таблиці TrackingInfos. Частина 2

Стовпець	Тип	Опис
StageLocation	nvarchar(MAX)	Посилання (включаючи шлях) на Stage локацію, на якій знаходиться завантажуваний файл
FileSize	bigint	Розмір завантаженого файлу
TimeReceived	datetime2(7)	Час надходження запиту на завантаження файлу з даними
lastInsertTime	datetime2(7)	Час останньої транзакції завантаження даних
RowsInserted	bigint	Кількість доданих записів в таблицю
RowsParsed	bigint	Кількість проаналізовних рядків
ErrorsSeen	int	Кількість помилок
ErrorsLimit	int	Максимальна допустима кількість помилок
FirstError	nvarchar(MAX)	Перша перехоплена помилка
FirstErrorLineNum	bigint	Номер рядка файлу, на якому виникла помилка
FirstErrorCharacterPos	bigint	Номер позиції символу, де виникла помилка
FirstErrorColumnName	nvarchar(MAX)	Назва колонки, для якої виникла помилка
SystemError	nvarchar(MAX)	Системна помилка
Complete	bit	Чи завершене завантаження
Status	nvarchar(MAX)	Статус загрузки даних

Як вже було зауважено, база даних SnowpipeTracker також містить ряд таблиць, які необхідні для роботи ASP.NET Core Identity. Наразі використовується лише таблиця AspNetUsers, проте інші таблиці можуть бути корисними для подальшого розширення системи, для додавання декількох ролей користувачів, або імплементації Claim-based підходу для авторизації користувачів в системі.

Таблиця користувачів AspNetUsers зв'язана відношенням багато-до-багатьох з таблицею Profiles за допомогою таблиці AspNetUserProfile. Діаграма бази даних (без таблиць ASP.NET Core Identity, які не використовуються в системі) подано в Додатку А.

5.5 Реалізація серверного застосунку

Як вже було згадано, серверний представляє собою ASP.NET Core Web API. Розроблений API побудований за канонами REST архітектури, проте не відповідає їй повністю. Відповідно до моделі зрілості Річардсона (Richardson Maturity Model) розроблений API відповідає другому рівню. Для повної відповідності REST архітектурі необхідно додати архітектурні обмеження HATEOAS.

5.5.1 Архітектура рішення

Серверний застосунок побудований на основі Onion-архітектури. Onion-архітектура (цибулева архітектура) передбачає розділення застосунку на рівні, причому існує один незалежний рівень, який знаходиться в центрі архітектури і не залежить від інших. Від цього рівня залежить другий, від якого залежить третій, який залежить також від першого, і так далі, таким чином формуючи структуру схожу на цибулину. Схематичну таку архітектуру зображено на рисунку 5.1:

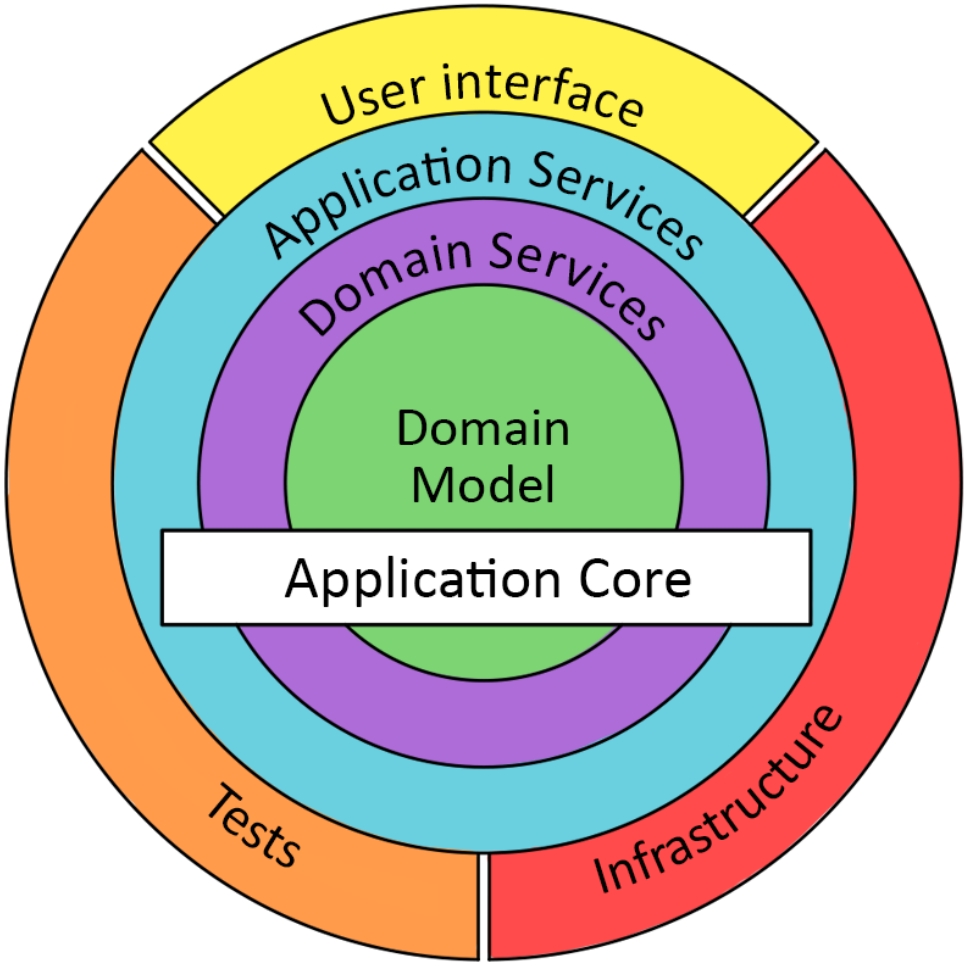


Рис. 5.1 – Схематичне зображення Onion-архітектури

В центрі «цибулини» знаходиться доменна модель (Domain Model) – ті об’єкти які використовує застосунок (зазвичай зберігаються в базі даних). Навколо доменної моделі розташований рівень доменних сервісів (Domain Services), зазвичай це набір інтерфейсів, які надають доступ до об’єктів доменної моделі. Навколо рівня доменних сервісів розташований рівень сервісів застосунку (Application Services), який типово містить набір інтерфейсів бізнес-логіки застосунку. Ці три рівні вважаються ядром застосунку (Application Core) і вважається, що вони не будуть змінюватися часто.

Зовнішній рівень складається зі всіх інших компонентів застосунку: інтерфейс користувача, юніт тести, допоміжні інфраструктурні клас, тощо. Також до цього рівня відносяться всі конкретні реалізації інтерфейсів ядра

застосунку (Application Core). Взагалі передбачається, що ядро застосунку містить лише класи для опису об’єктів домену та інтерфейси, реалізація яких розташовується на зовнішньому рівні архітектури.

Така архітектурний підхід дає ряд важливих переваг:

- Onion архітектура поділяє застосунок на рівні та модулі, кожен з яких виконує свою специфічну задачу. Це забезпечує прозору та добре організовану структуру застосунку, що дозволяє зрозуміти код, існуючі залежності в ньому та полегшує його підтримку.
- Кожен рівень може бути реалізований незалежно, причому з використанням різних технологій, що сприяє пришвидшенню розробки
- Onion архітектура дозволяє легко розуміти залежності, наявні в застосунку, що сприяє його розширюваності та гнучкості при змінах.
- Модульність та поділ на рівні полегшують написання як модульних так і інтеграційних тестів.

Розроблений серверний застосунок складається з 8 проектів-модулів. Структура рішення зображена на рисунку 5.2:

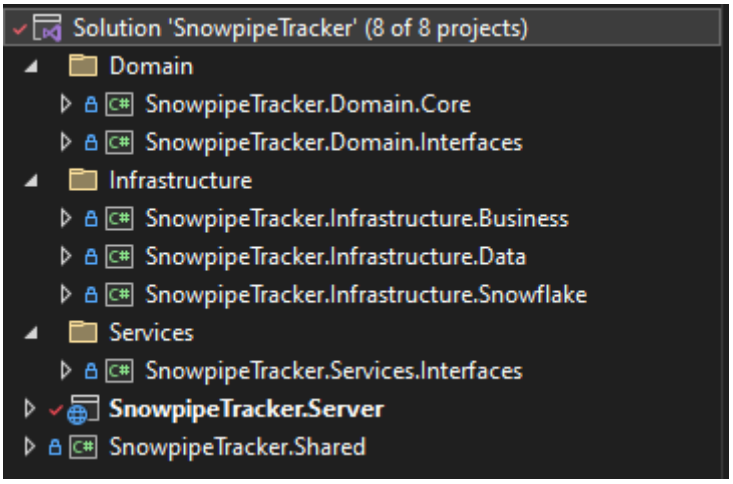


Рис. 5.2 – Схематичне зображення Onion-архітектури

Проект SnowpipeTracker.Domain.Core містить прості класи для опису об’єктів доменної моделі, такі як SfProfile, Snowpipe, TrackingInfo. Такі класи практично не містять ніякої логіки, крім атрибутів, необхідних для побудови бази даних на основі цих моделей. В контексті Onion-архітектури цей проект відповідає рівню доменної моделі (Domain Model).

Проект SnowpipeTracker.Domain.Data містить інтерфейси, які надають доступ до об’єктів доменної моделі з джерел даних (наразі таких джерела 2: внутрішня база системи SnowpipeTracker та власне саме сховище Snowflake). В контексті Onion-архітектури цей проект відповідає рівню доменних сервісів (Domain Services).

Проект SnowpipeTracker.Services.Interfaces містить інтерфейси сервісів, які реалізують бізнес-логіку системи. В контексті Onion-архітектури цей проект відповідає рівню сервісів застосунку (Application Services).

Всі інші проекти відносяться до зовнішнього рівня Onion-архітектури:

Проект SnowpipeTracker.Infrastructure.Data містить реалізацію частини інтерфейсів доменних сервісів, які відповідають за маніпуляції з об’єктами доменної моделі в внутрішній базі даних системи SnowpipeTracker.

Проект SnowpipeTracker.Infrastructure.Snowflake містить реалізацію частини інтерфейсів доменних сервісів, які відповідають за маніпуляції з об’єктами доменної моделі в системі Snowflake.

Проект SnowpipeTracker.Infrastructure.Business містить реалізацію інтерфейсів сервісів застосунку (Application Services), тобто по суті містить бізнес-логіку системи.

Проект SnowpipeTracker.Server – це ASP.NET Web API з чисельними ендпоїнтами для взаємодії користувача з системою.

Проект SnowpipeTracker.Shared містить класи-контракти для уніфікації взаємодії клієнта з сервером. Цей проект спільний для клієнта та сервера.

5.5.2 Взаємодія з базою даних

При розробці системи для взаємодії з внутрішньою базою була використана технологія Entity Framework Core. Entity Framework Core – це ORM (Object-Relational Mapper) технологія, яка дозволяє розробникам працювати з базою даних за допомогою об’єктів .NET.

Сама база даних розроблена за допомогою підходу Code-first, який дозволяє генерувати та оновлювати базу даних на основі класів в .NET. Таким чином було спочатку розроблено ряд доменних моделей, а потім на їх основі згенерована база даних.

Було розроблено ряд репозиторіїв для роботи з контекстом бази даних. Для об’єднання бізнес-транзакцій було реалізовано паттерн UnitOfWork, який об’єднує доступ до розроблених репозиторіїв.

Як вже було зазначено, в якості серверу бази даних був обраний Microsoft SQL Server.

5.5.3 Взаємодія з Snowflake

Взаємодія системи з Snowflake умовно поділяється на 2 частини: взаємодія з Snowflake SQL та взаємодія з Snowpipe REST API.

Взаємодія з Snowflake SQL відбувається за допомогою бібліотеки Snowflake.Data. Для зручності мапингу результатів SQL запитів в об’єкти .NET було розроблено декоратор для Snowflake з’єднання за допомогою бібліотеки Dapper. Також було розроблено фабрику з’єднань, яка повертає з’єднання з аккаунтом в Snowflake на основі профілю користувача.

Для взаємодії з Snowpipe REST API було розроблено кастомний Http-клієнт, який дозволяє використовувати всі 3 наявних ендпоїнтів Snowpipe REST API, хоча система наразі використовує лише 2 з них. В системі розроблений клієнт реєструється та використовується за допомогою сучасного підходу HttpClientFactory.

5.5.4 Автентифікація та авторизація

В розроблюваному Web API автентифікація відбувається за допомогою JWT (JSON Web Token). Після успішної автентифікації сервер генерує JWT-токен для клієнта, який клієнт зберігає та додає до всіх подальших запитів до сервера. Для інтеграції такої системи автентифікації була використана бібліотека Microsoft.AspNetCore.Authentication.JwtBearer.

Маніпуляція з користувачами відбувається за допомогою сервісів, наданих бібліотекою AspNetCore.Identity, таким чином делегуючи відповідальність за безпечне збереження облікових даних користувача та їх верифікацію на творців цієї бібліотеки.

За рахунок сервісів AspNetCore.Identity можна легко додати підтримку різних ролей користувачів, або ж реалізувати Claim-based авторизацію. Наразі ж авторизація полягає лише в бізнес-валідації доступності ресурсу для користувача.

5.5.5 Специфікація розробленого Web API

Розроблений Web API містить 4 контролери, які реалізують чисельні ендпоїнти. Всі ендпоїнти доступні лише для авторизованих користувачів, за виключення ендпоїнтів login та register. Swagger-специфікація всіх ендпоїнтів застосунку наведена в додатку Б. Наведемо короткий опис кожного з контролерів та його ендпоїнтів (кінцевих точок).

Контролер Auth слугує для авторизації користувача та реєстрації нових користувачів. Містить ендпоїнт (POST) /api/login для авторизації користувача та (POST) /api/register для реєстрації нового коритсувача.

Контролер Profile містить ряд ендпоїнтів для управління профілями користувача:

- GET /api/Profile – повертає профілі, доступні користувачу;
- GET /api/Profile/{id} – повертає профіль з заданим id;

- DELETE /api/Profile/{id} – видаляє заданий профіль з системи;
- PUT /api/Profile/{id} – оновлює облікові дані користувача для доступу в Snowflake;
- POST /api/Profile – створює новий профіль користувача;
- PATCH /api/Profile/{id}/remove – забирає доступ до профілю у поточного користувача;
- PATCH /api/Profile/{id}/share – надає доступ до профілю заданому користувачу;

Контролер Snowpipe надає ендпоїнти для роботи з об’єктами Snowpipe:

- GET /api/Profile/{profileId}/Snowpipe – повертає всі об’єкти Snowpipe в системі для заданого профілю;
- GET /api/Profile/{profileId}/Snowpipe/actual – повертає дійсні (ті що не видалені на Snowflake) об’єкти Snowpipe в системі для заданого профілю;
- GET /api/Profile/{profileId}/Snowpipe/{pipeId} – повертає об’єкт Snowpipe з заданим ідентифікатором;
- DELETE /api/Profile/{profileId}/Snowpipe/{pipeId} – видаляє заданий об’єкт Snowpipe з Snowflake;
- GET /api/Profile/{profileId}/Snowpipe/{pipeId}/latest – повертає актуальну версію об’єкта Snowpipe для об’єкта Snowpipe з заданим ідентифікатором;
- POST /api/Profile/{profileId}/Snowpipe/sync – синхронізує об’єкти Snowpipe в заданому профілі;
- POST /api/Profile/{profileId}/Snowpipe/{pipeId}/sync – синхронізує заданий об’єкт Snowpipe;
- PATCH /api/Profile/{profileId}/Snowpipe/{pipeId}/resume – активує заданий об’єкт Snowpipe в Snowflake;

- PATCH /api/Profile/{profileId}/Snowpipe/{pipeId}/pause – деактивує заданий об’єкт Snowpipe в Snowflake;
- POST /api/Profile/{profileId}/Snowpipe/{pipeId}/trigger – ініціює завантаження заданих файлів за допомогою заданого об’єкту Snowpipe;

Контролер Tracking надає ендпоїнти для керування історією завантаження об’єкта Snowpipe:

- GET /api/Profile/{profileId}/Snowpipe/{pipeId}/Tracking – повертає історію завантаження для об’єкта Snowpipe з заданим ідентифікатором;
- POST /api/Profile/{profileId}/Snowpipe/{pipeId}/Tracking – оновлює історію завантаження для об’єкта Snowpipe з заданим ідентифікатором;
- PATCH /api/Profile/{profileId}/Snowpipe/{pipeId}/Tracking/enable – вмикає авто-оновлення історії завантаження для об’єкта Snowpipe з заданим ідентифікатором;
- PATCH /api/Profile/{profileId}/Snowpipe/{pipeId}/Tracking/disable – вимикає авто-оновлення історії завантаження для об’єкта Snowpipe з заданим ідентифікатором;

5.5.6 Сервіс синхронізації об’єктів Snowpipe

Для регулярної синхронізації об’єктів Snowpipe в серверному застосунку є окремий сервіс, який запускається через заданий конфігурований проміжок часу. В термінах ASP.NET Core такий сервіс є HostedService. Він для кожного існуючого профілю синхронізує об’єкти Snowpipe та їх історію завантаження (в разі якщо автоматичний моніторинг історії ввімкнений користувачем) по алгоритму описаному в підрозділі 5.2.

Таке рішення є простим та зрозумілим, проте воно не є оптимальним, адже при синхронізації сервер використовує ті ж обчислювальні ресурси, що й використовуються для обробки запитів користувачів, що може призвести для небажаних просадок продуктивності сервера.

Кращим варіантом вирішення цієї задачі – є винесення цього сервісу в якесь хмарне безсерверне рішення з автоматичним масштабуванням, наприклад в AWS Lambda або Azure Function, які б виконували синхронізацію кожного профілю окремо. На щастя, код спроектований таким чином, що зробити це буде відносно не важко, проте в контексті даної роботи на це немає часу і ресурсів.

5.6 Реалізація клієнтського застосунку

Як вже зазначалося, клієнтський застосунок розроблений за допомогою вже розглянутої технології Balzor WASM. Застосунок являє собою набір компонентів та сторінок, створених за допомогою синтаксису Razor, які взаємодіють з кінцевими точками розробленого серверного застосунку.

JWT-токен користувача зберігається в сховищі сесії (session storage) в закодованому вигляді за допомогою бібліотеки Blazored.SessionStorage та додається до кожного запиту до серверного застосунку.

При розробці розмітки для інтерфейсу користувача активно використовувалася технологія Bootstrap. Вона дозволила створити зручний та інтуїтивно-зрозумілий веб-інтерфейс.

Завдяки тому, що як серверний так і клієнтський застосунок використовують мову C#, всі POCO-класи для передачі даних між клієнтом та сервером винесені в окремий проект SnowpipeTracker.Shared. Це дозволяє уніфікувати всі контракти між сервером та клієнтом, що значно полегшило розробку та унеможливило помилки. Також, завдяки цьому, уніфікована вся валідація для моделей даних, які однаково валідуються як на сервері так і на клієнті (за допомогою валідації, заснованої на атрибутах).

При розробці активно використовувалися вбудовані тег-хелпери, які також полегшували та пришвидшували розробку. Наприклад, для кастомізації представлення для користувача залежно від того чи є він авторизованим, використовується тег-хелпери <AuthorizeView>, <Authorized> / <NotAuthorized>.

РОЗДІЛ 6. ІНСТРУКЦІЯ КОРИСТУВАЧА

6.1 Реєстрація та вхід в систему

Для роботи з системою користувачу необхідно ввійти в свій аккаунт запис, або ж створити новий аккаунт. Для реєстрації нового акаунту користувач має ввести свої дані на сторінці реєстрації та натиснути кнопку “Register”: див. рисунок 6.1:

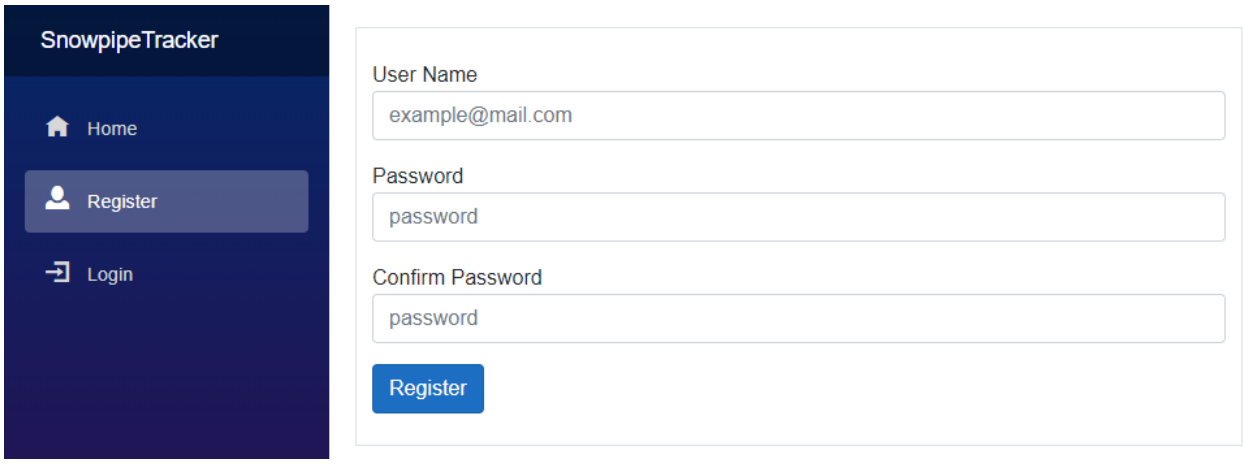


Рис. 6.1 – Сторінка реєстрації користувача

Для входу в систему необхідно зайти у вкладку “Login”, заповнити поля логіну та паролю та натиснути кнопку “Log in”, як показано на рисунку 6.2. В разі неуспішної авторизації, як і у разі будь-якого іншого неуспішного запиту до серверного застосунку, в правій стороні екрану користувачу буде показано сповіщення.

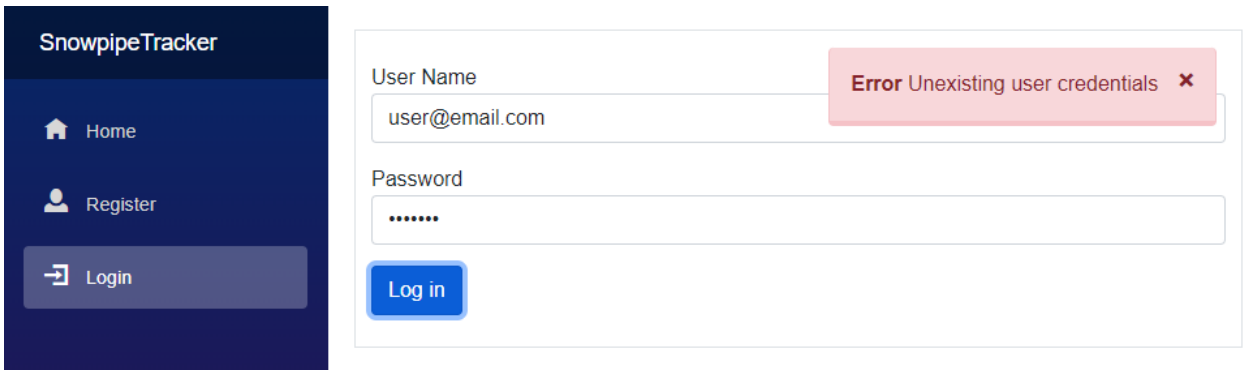


Рис. 6.2 – Сторінка авторизації користувача

Після успішної авторизації користувачу буде показано список доступних йому профілів. Для того щоб вийти з системи необхідно натиснути кнопку “Logout” в лівій частині екрану, див. рисунок 6.3:

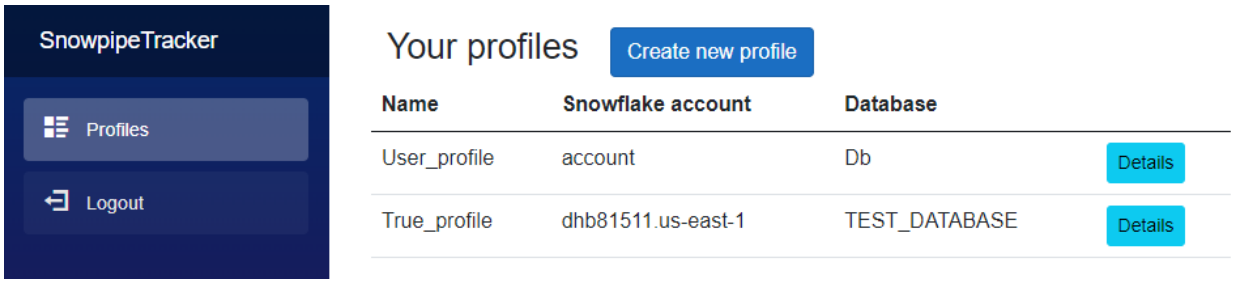


Рис. 6.3 – Екран користувача після авторизації

6.2 Управління профілями

Після успішної авторизації користувач потрапляє на сторінку доступних йому профілів, як показано на рис. 6.3. Для створення нового профілю користувач має натиснути кнопку “Create new profile”, після чого він буде перенаправлений на сторінку, показану на рисунку 6.4

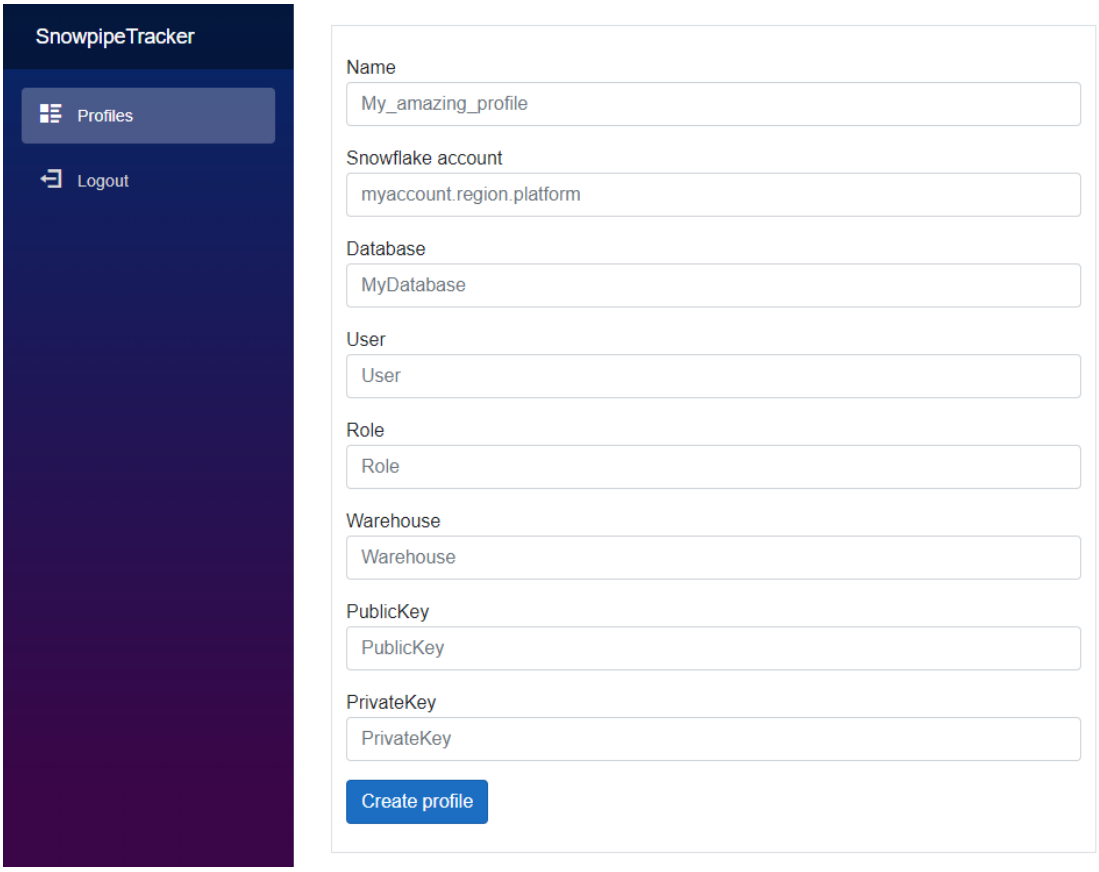


Рис. 6.4 – Сторінка створення профілю

Для створення профілю користувачу необхідно ввести бажану назву профілю в поле “Name” та облікові дані для доступу в систему Snowflake. В поле Snowflake_account необхідно записати ідентифікатор акаунту Snowflake. В поля Database, Role, Warehouse необхідно вписати базу даних в Snowflake, роль, яку буде використовувати система, та віртуальне сховище даних для використання системою. Поля Database, Account та Name змінювати після створення не можна, тому треба бути уважним під час їх заповнення.

Для підключення системи SnowpipeTracker до Snowflake рекомендується створити окрему роль з мінімальним набором прав, необхідних для роботи системи. Таку роль можна створити за допомогою скрипта наведеного нижче:

```
SET TRACKING_ROLE = '<TrackingRole>';
SET DATABASE_NAME = '<Database>';
SET TRACKING_USER = '<User>';
GRANT MONITOR, OPERATE ON DATABASE $DATABASE_NAME TO
$TRACKING_ROLE;
GRANT MONITOR, OPERATE ON ALL SCHEMAS IN DATABASE $DATABASE_NAME
TO ROLE $TRACKING_ROLE;
GRANT MONITOR, OPERATE ON FUTURE SCHEMAS IN DATABASE
$DATABASE_NAME TO ROLE $TRACKING_ROLE;
GRANT SELECT ON ALL VIEWS IN DATABASE $DATABASE_NAME TO ROLE
$TRACKING_ROLE;
GRANT SELECT ON FUTURE VIEWS IN DATABASE $DATABASE_NAME TO ROLE
$TRACKING_ROLE;
GRANT ALL ON ALL STAGES IN DATABASE $DATABASE_NAME TO ROLE
$TRACKING_ROLE;
GRANT ALL ON FUTURE STAGES IN DATABASE $DATABASE_NAME TO ROLE
$TRACKING_ROLE;
GRANT ALL ON ALL PIPES IN DATABASE $DATABASE_NAME TO ROLE
$TRACKING_ROLE;
GRANT ALL ON FUTURE PIPES IN DATABASE $DATABASE_NAME TO ROLE
$TRACKING_ROLE;
GRANT SELECT, INSERT ON ALL TABLES IN DATABASE $DATABASE_NAME TO
ROLE $TRACKING_ROLE;
GRANT SELECT, INSERT ON FUTURE TABLES IN DATABASE $DATABASE_NAME
TO ROLE $TRACKING_ROLE;
GRANT USAGE ON INTEGRATION $TRACKING_ROLE_TEST_INT TO ROLE
$TRACKING_ROLE;
GRANT ROLE $TRACKING_ROLE TO USER TRACKING_USER;
```

Де <TrackingRole> - назва ролі для доступу системи SnowpipeTracker, <Database> - назва бази в Snowflake, <User> - назва юзеру в акаунті Snowflake, який буде використовуватися системою SnowpipeTracker.

Для роботи системи також необхідно налаштувати для Snowflake юзера автентифікацію по парі ключів, які необхідно вписати в поля PrivateKey та PublicKey. Детальна інструкція як це зробити наведена в статті [42].

Після введення даних, користувач має натиснути кнопку “Create profile”, після чого система провалідує дані введені користувачем та спробує зайти в Snowflake за допомогою введених облікових даних. В разі успіху, профіль буде успішно створений в системі та користувач перейде до сторінки своїх профілів (рисунок 6.3), інакше користувачу буде виведене повідомлення в правому верхньому куті.

На сторінці профілів (рис. 6.3) користувачу відображено список його профілів в вигляді таблиці. При натисненні кнопки “Details” користувач перейде на сторінку відповідного профілю див. рисунок 6.5.

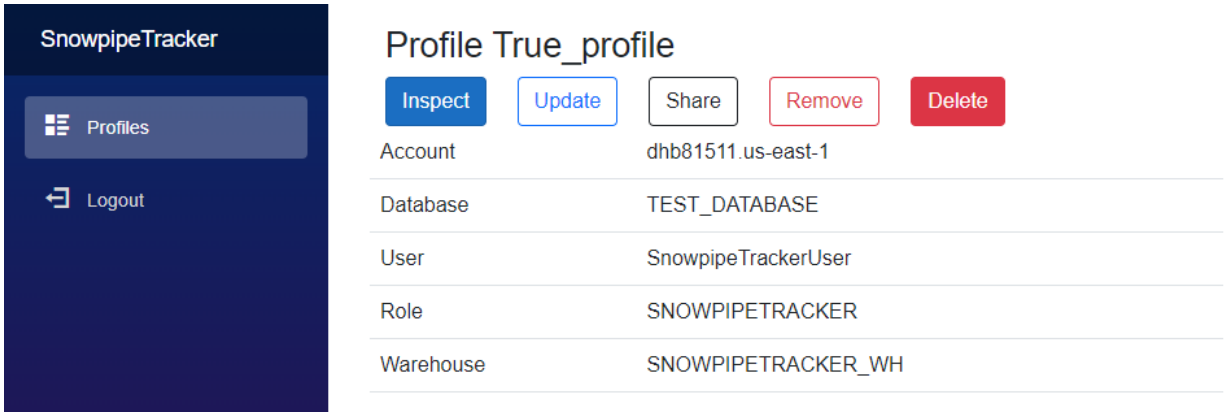


Рис. 6.5 – Сторінка профілю

В разі якщо профіль не є активним, тобто його облікові дані не дають під’єднатися до Snowflake, то користувачу буде показано відповідне сповіщення (див. рис. 6.6):

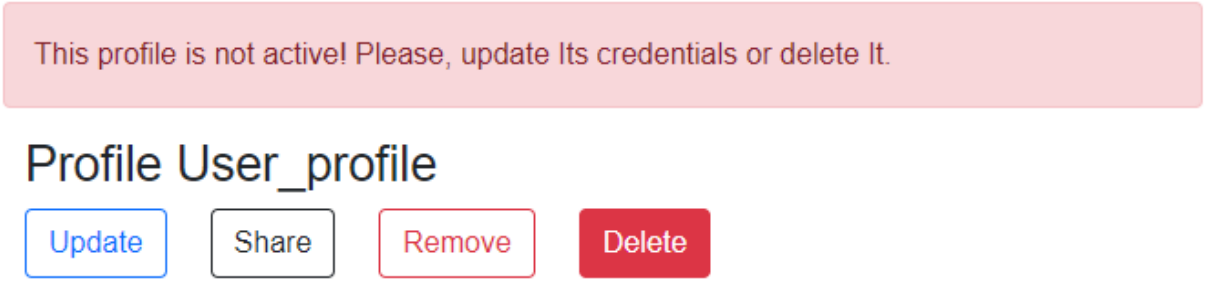


Рис. 6.6 – Сторінка неактивного профілю

На сторінці профілю користувач може оновити облікові дані профілю для доступу в Snowflake, видалити профіль лише для себе, поділитися профілем з іншими користувачами та видалити профіль з системи. Для активних профілів також доступний перегляд їх вмісту (об’єктів Snowpipe).

Для оновлення облікових даних профілю для доступу в Snowflake, користувач має натиснути кнопку “Update”, після чого відкриється форма як на рисунку 6.7:

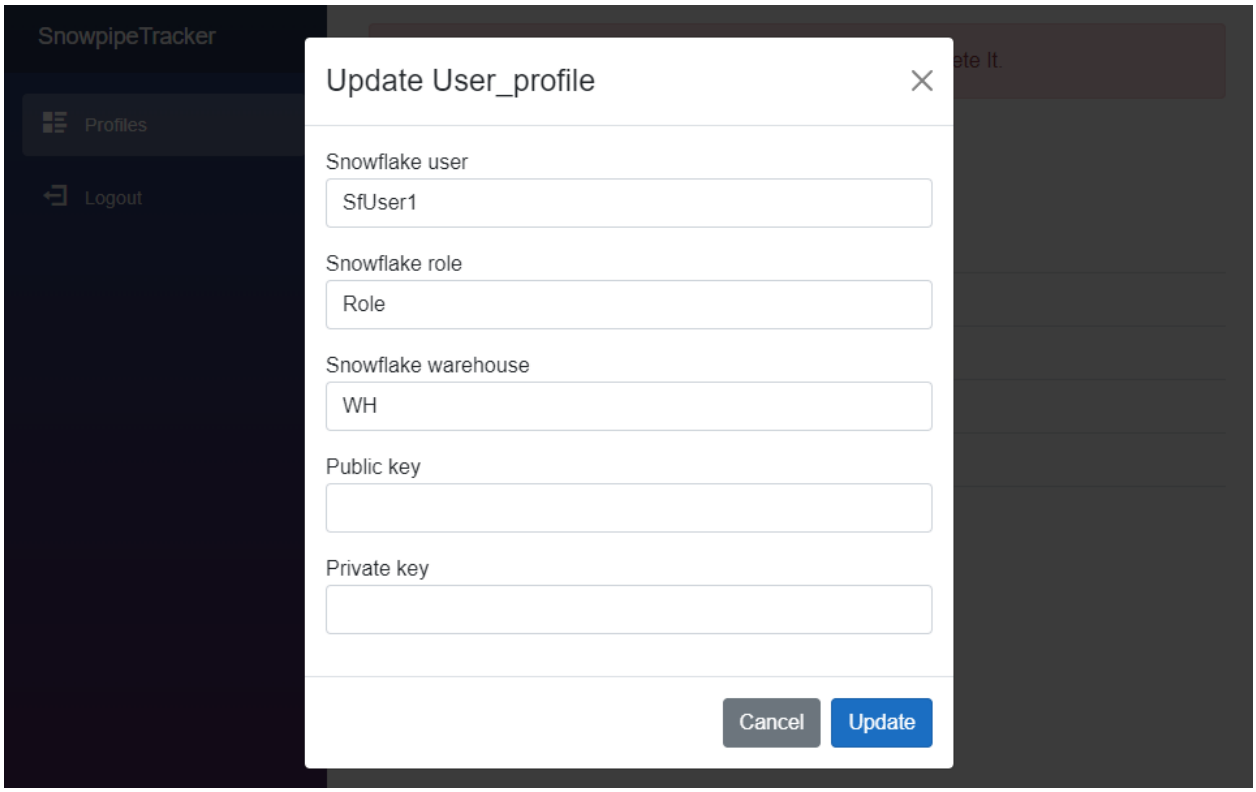


Рис. 6.7 – Форма оновлення облікових даних профілю

Після заповнення форми та натиснення кнопки “Update”, валідність облікових даних буде перевірено та в разі їх коректності, профіль буде оновлено, після чого користувач зможе зайти в нього.

Для повного видалення профілю з системи та видалення його лише для себе, користувачу необхідно натиснути кнопки “Delete” та “Remove” відповідно. Після чого користувач має підтвердити свої дії в діалоговому вікні, як на рисунку 6.8:

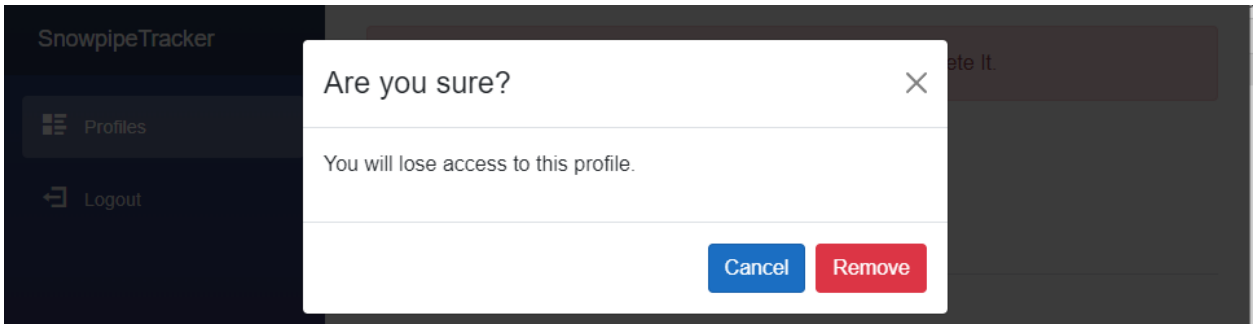


Рис. 6.8 – Діалогове вікно для підтвердження дії користувача

Для того щоб поділитися профілем з іншим користувачем, користувач має натиснути кнопку “Share” на сторінці профіля (див. рис. 6.5). Після цього користувачу буде показано діалогове вікно як на рисунку 6.9, де користувач має ввести email користувача, якому він хоче надати доступ до профілю та натиснути кнопку “Share”:

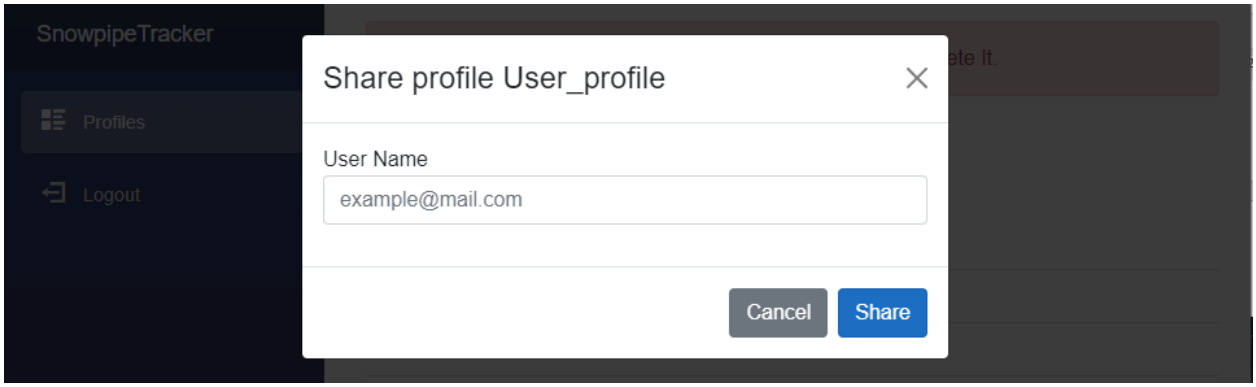


Рис. 6.9 – Діалогове вікно для надання доступу до профілю іншому користувачу

Для входу в профіль та перегляду об’єктів Snowpipe, які він містить, користувачу необхідно натиснути кнопку “Inspect” на сторінці профілю користувача (див. рис. 6.5). Ця кнопка доступна лише для активних профілів.

6.3 Управління об’єктами Snowpipe

Після того, як користувач зайшов у профіль, натиснувши кнопку “Inspect” на сторінці профілю користувача (див. рис. 6.5), користувач переходить на сторінку об’єктів Snowpipe, да відображається список

існуючих (тих які наразі присутні в Snowflake) об’єктів Snowpipe у вигляді таблиці, як показано на рисунку 6.10:

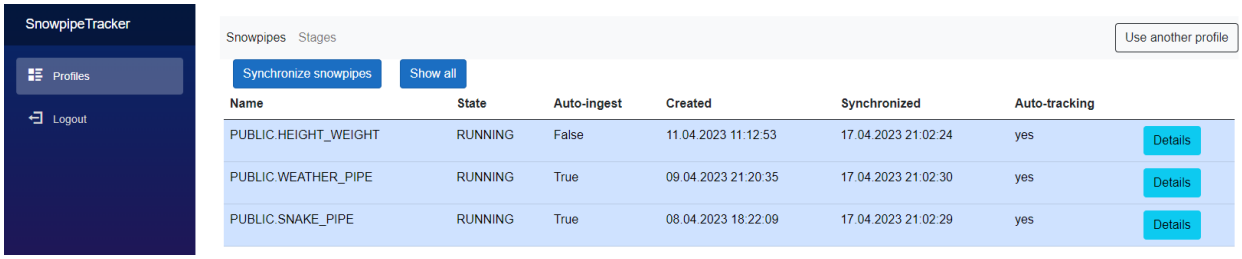


Рис. 6.10 – Сторінка об’єктів Snowpipe в профілі (існуючі об’єкти)

Для синхронізації всіх об’єктів Snowpipe поточного профілю з наявними у Snowflake, користувачу необхідно натиснути кнопку “Synchronize snowpipes” на сторінці об’єктів Snowpipe (див. рис. 6.10).

Для відображення всіх об’єктів Snowpipe, існуючих в системі SnowpipeTracker, тобто не лише активних, а й історичних об’єктів Snowpipe, користувачу необхідно натиснути кнопку “Show all” на сторінці об’єктів Snowpipe (див. рис. 6.10). Після цього список об’єктів на сторінці буде оновлено. Активні об’єкти Snowpipe мають синій фон, а історичні – сірий, див. рис. 6.11:

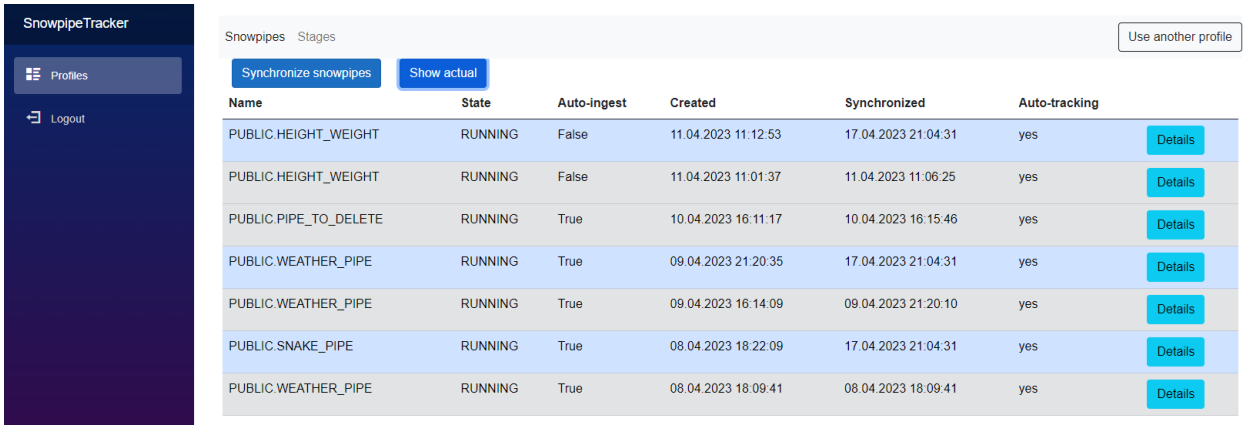


Рис. 6.11 – Сторінка об’єктів Snowpipe в профілі (історичні об’єкти)

Для відображення лише існуючих в Snowflake об’єктів, користувачу необхідно натиснути кнопку “Show actual”, після чого, в списку об’єктів Snowpipe будуть відображені лише існуючі в Snowflake об’єкти, як на рисунку 6.10.

Для перегляду детальної інформації про об’єкт Snowpipe, користувачу необхідно натиснути кнопку “Details” біля об’єкту, який його цікавить, на сторінці об’єктів Snowpipe (див. рис. 6.10). Після цього відкриється сторінка конкретного об’єкту Snowpipe як на рисунку 6.12:

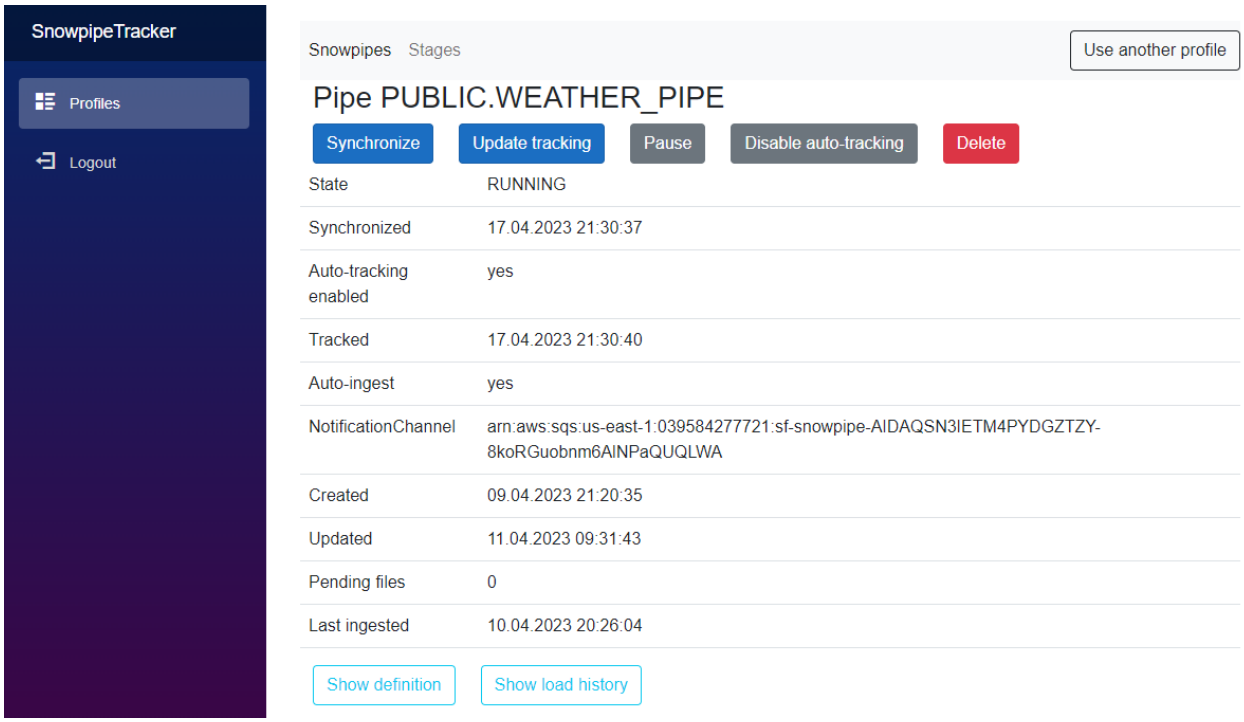


Рис. 6.12 – Сторінка об’єкту Snowpipe (існуючий, auto_ingest=true)

Якщо користувач перейшов на сторінку перегляду детальної інформації для історичного об’єкту Snowpipe, то в верхній частині екрану буде показане сповіщення, що цей об’єкт є історичним та кнопка “To latest version”, як на рисунку 6.13.

При натисканні на кнопку “To latest version” система перевірить чи є ця версія об’єкту останньою в системі. Якщо версія об’єкту дійсно є останньою, то користувачу буде виведено про це сповіщення в правому верхньому куті екрану. Якщо ж в системі наявна новіша версія об’єкту, то користувач буде перенаправлений на сторінку цього об’єкту Snowpipe.

Для історичних об’єктів Snowpipe доступний перегляд їх визначення та перегляд їх історії завантаження. Переглянути які, можна так само, як і для активних (існуючих) об’єктів Snowpipe.

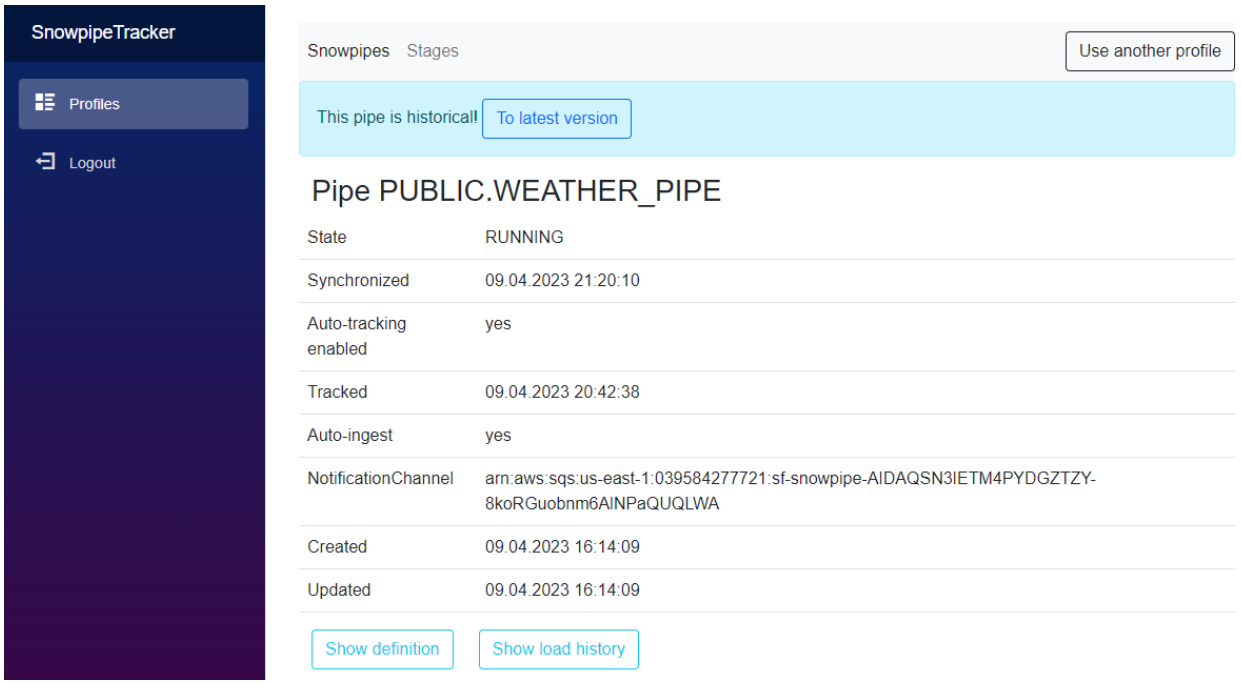


Рис. 6.13 – Сторінка об’єкту Snowpipe (історичний, auto_ingest)

Для перегляду історії завантаження для об’єкту Snowpipe, необхідно натиснути кнопку “Show load history” на сторінці об’єкту Snowpipe (див. рис. 6.12). Після цього, під кнопкою відобразиться історія завантаження об’єкту Snowpipe в вигляді таблиці, де подана інформація про завантаження кожного окремого файлу, як показано на рисунку 6.14:

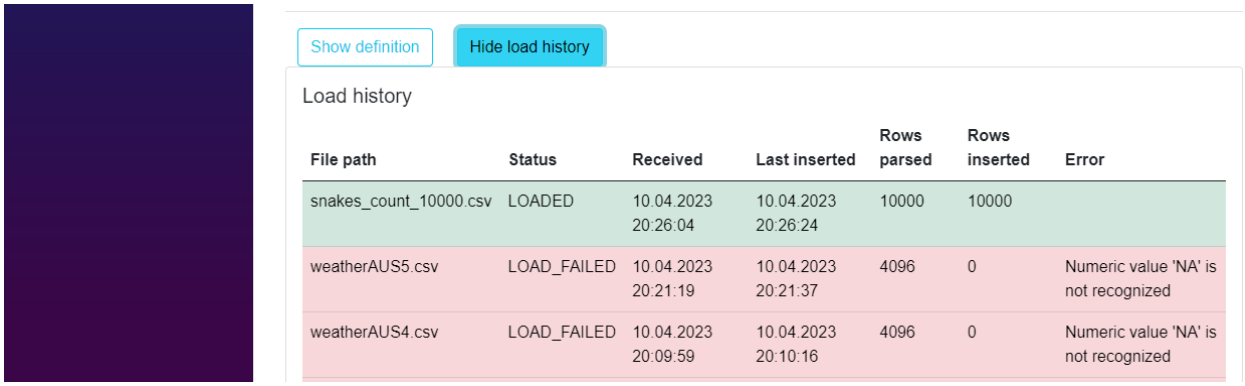


Рис. 6.14 – Історія завантаження для об’єкту Snowpipe

Для того щоб переглянути означення об’єкту Snowpipe, тобто його COPY INTO інструкція, користувачу необхідно натиснути кнопку “Snow definition” на сторінці об’єкту Snowpipe (див. рис. 6.12). Після цього його означення буде показано нижче самої кнопки, як показано на рисунку 6.15:

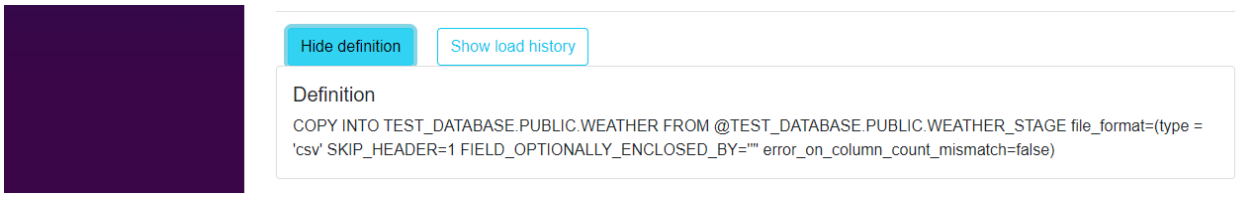


Рис. 6.15 – Означення об’єкту Snowpipe

Для синхронізації окремого об’єкту Snowpipe користувачу потрібно натиснути кнопку “Synchronize” на сторінці об’єкту Snowpipe (див. рис. 6.12), після чого дані на сторінці будуть оновлені.

Для оновлення історії завантаження для окремого об’єкту Snowpipe, користувачу необхідно натиснути кнопку “Update tracking” (див. рис. 6.12), після чого дані про історію завантаження об’єкту Snowpipe будуть оновлені на сторінці у разі якщо вони відображалися до цього.

Якщо об’єкт Snowpipe створений з вимкненою опцією auto_ingest, то для нього відображається кнопка “Trigger load” (як показано на рисунку 6.16), яка дозволяє ініціювати завантаження даних об’єктом Snowpipe.

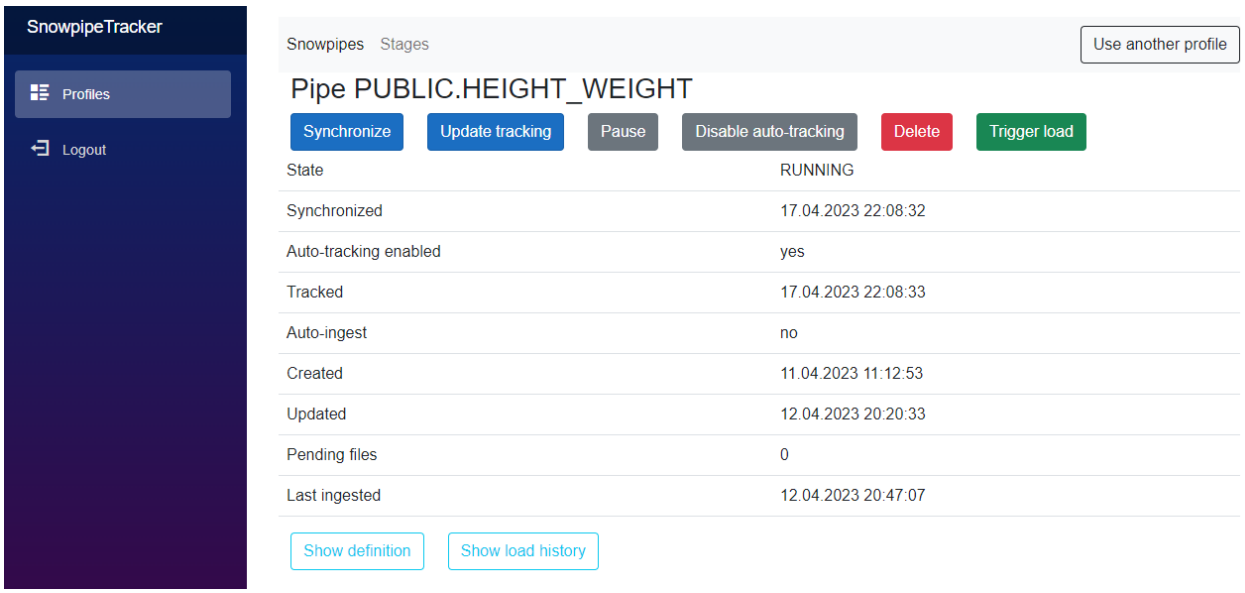


Рис. 6.16 – Сторінка об’єкту Snowpipe (існуючий, auto_ingest=false)

Для ініціації завантаження даних за допомогою об’єкту Snowpipe (лише з опцією auto_ingest=false) користувачу необхідно натиснути кнопку “Trigger load” на сторінці об’єкту Snowpipe (див. рис. 6.16). Після цього користувачу відкриється діалогове вікно, як на рисунку 6.17:

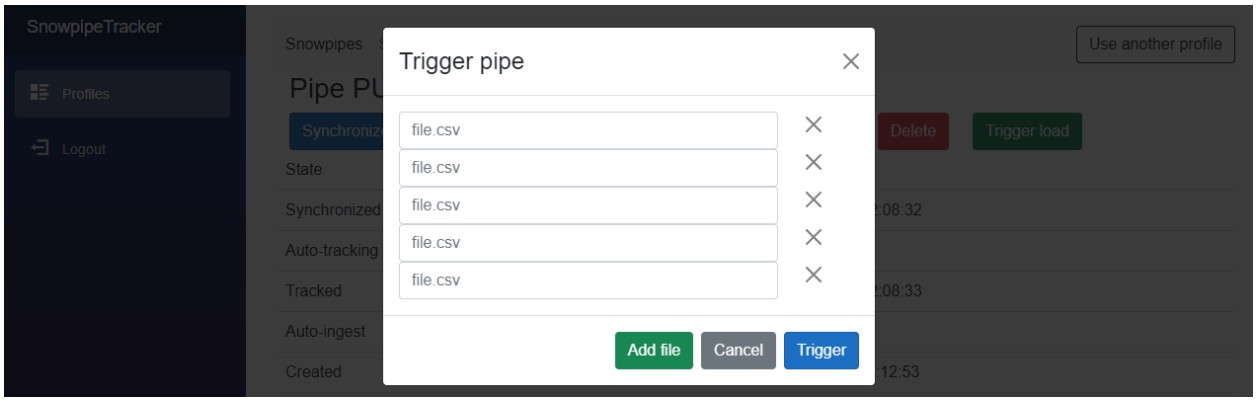


Рис. 6.17 – Діалогове вікно для ініціації завантаження даних за допомогою об’єкту Snowpipe

На діалоговому вікні ініціації завантаження даних об’єктом Snowpipe (див. рис. 6.17) користувач може додавати файли, натискаючи кнопку “Add file” та вводити їх відносний шлях в стейдж-локації, на яку налаштований об’єкт Snowpipe. При натисненні хрестика біля поля введення шляху файлу, воно зникне. Для ініціації завантаження введених файлів, користувачу необхідно натиснути кнопку “Trigger”, після чого, в разі успішної валідації введених користувачем даних процес завантаження буде ініційований в системі Snowflake.

Для призупинення/відновлення роботи об’єкту Snowpipe, користувачу необхідно натиснути кнопку “Pause”/”Resume” на сторінці об’єкту Snowpipe (див. рис. 6.12). Після цього статус об’єкта Snowpipe буде оновлений.

Для ввімкнення/вимкнення автоматичного оновлення історії завантаження для об’єкта Snowpipe, користувачу необхідно натиснути кнопку “Enable auto-tracking”/”Disable auto-tracking” на сторінці об’єкту Snowpipe (див. рис. 6.12).

Для видалення об’єкта Snowpipe, користувачу необхідно натиснути кнопку “Delete” на сторінці об’єкту Snowpipe (див. рис. 6.12), після чого об’єкт Snowpipe буде видалений зі Snowflake та промаркований в системі SnowpipeTracker як історичний.

ВИСНОВКИ

Було детально розглянуто концепцію сховища даних, проаналізовано її відмінності від інших концепцій збереження даних, виділено основні переваги та недоліки розглянутих концепцій збереження даних.

Було розглянуто основні ідеї хмарних сховищ даних, їх переваги над традиційними рішеннями сховищ даних, виділено основні аспекти хмарних сховищ даних. Був проведений огляд найбільш популярних на ринку хмарних сховищ даних: Snowflake [1], Google BigQuery [20], Amazon Redshift [23] та Azure Synapse (SQL DW) [25]. На основі огляду була створена порівняльна таблиця для цих реалізацій.

Було розглянуто деталі реалізації хмарного сховища даних Snowflake, детально розглянуто засоби завантаження даних в Snowflake. Виявилося, що основним та рекомендованим засобом завантаження даних в Snowflake є сервіс Snowpipe; існуюче рішення для управління об'єктами Snowpipe містить ряд проблем.

Для вирішення визначених проблем управління об’єктами Snowpipe, була спроектована та реалізована система, яка складається з двох частин: серверний застосунок ASP.NET Core [34] Web API та клієнтський SPA з використанням технології Blazor WASM [35].

Було розглянуто основні підходи, архітектурні рішення та технології, використані для розробки системи. Було складено детальну інструкцію користувача для управління засобами завантаження даних в Snowflake за допомогою розробленої системи.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Snowflake [Електронний ресурс] – Режим доступу до ресурсу: https://www.snowflake.com/
2. Jumpstart Snowflake. A Step-by-Step Guide to Modern Cloud Analytics – [Dmitry Anoshin, Dmitry Shirokov, Donna Strok]
3. .NET [Електронний ресурс] – https://dotnet.microsoft.com/
4. C# [Електронний ресурс] – https://learn.microsoft.com/uk-ua/dotnet/csharp/
5. Microsoft SQL Server 2019 [Електронний ресурс] – https://www.microsoft.com/uk-ua/sql-server/sql-server-2019
6. Visual Studio 2022 [Електронний ресурс] – https://visualstudio.microsoft.com/en/vs/
7. Data Warehouse vs Database: 9 Important Differences [Електронний ресурс] – Режим доступу до ресурсу: https://hevodata.com/learn/data-warehouse-vs-database/
8. The Difference Between a Data Warehouse and a Database [Електронний ресурс] – Режим доступу до ресурсу: https://panoply.io/data-warehouse-guide/the-difference-between-a-database-and-a-data-warehouse/
9. Data Lake vs Data Warehouse [Електронний ресурс] – Режим доступу до ресурсу: https://www.talend.com/resources/data-lake-vs-data-warehouse/.
10. Data Lakes vs. Data Warehouses [Електронний ресурс] – Режим доступу до ресурсу: https://www.datacamp.com/blog/data-lakes-vs-data-warehouses
11. Data Warehousing Market Insights [Електронний ресурс] – Режим доступу до ресурсу: https://www.alliedmarketresearch.com/data-warehousing-market
12. Snowflake, Redshift, BigQuery, and Others: Cloud Data Warehouse Tools Compared [Електронний ресурс] – Режим доступу до ресурсу: https://www.altexsoft.com/blog/snowflake-redshift-bigquery-data-warehouse-tools/
13. Shared Nothing v.s. Shared Disk Architectures: An Independent View [Електронний ресурс] – Режим доступу до ресурсу:

http://www.benstopford.com/2009/11/24/understanding-the-shared-nothing-architecture/

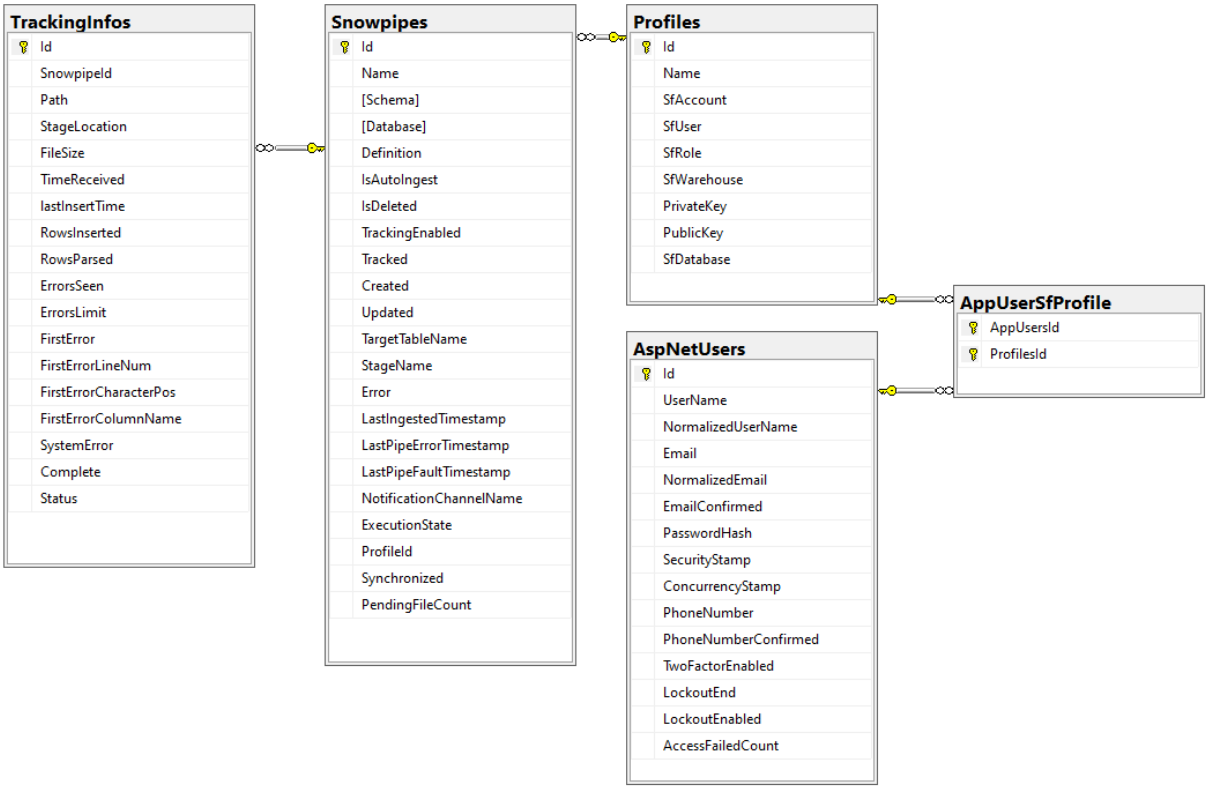
14. Amazon Web Services [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/
15. Microsoft Azure [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/en-us/
16. Google Cloud Platform [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/
17. Snowflake doc: Key Concepts & Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://docs.snowflake.com/en/user-guide/intro-key-concepts.html
18. BigQuery vs Snowflake Comparison: Choosing the Right Data Warehouse in 2021 Made Easy [Електронний ресурс] – Режим доступу до ресурсу: https://hevodata.com/learn/google-bigquery-vs-snowflake-comparison/
19. Data Warehouse in the Cloud Benchmark [Електронний ресурс] – Режим доступу до ресурсу: https://research.gigaom.com/report/data-warehouse-cloud-benchmark/
20. BigQuery [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/bigquery
21. BigQuery explained: An overview of BigQuery's architecture [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/blog/products/data-analytics/new-blog-series-bigquery-explained-overview
22. 2020 Cloud Data Warehouse Benchmark: Redshift, Snowflake, Presto and BigQuery [Електронний ресурс] – Режим доступу до ресурсу: https://www.fivetran.com/blog/warehouse-benchmark
23. AWS Redshift [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/ru/redshift/
24. AWS doc: Data warehouse system architecture [Електронний ресурс] – Режим доступу до ресурсу:

https://docs.aws.amazon.com/redshift/latest/dg/c_high_level_system_architecture.html

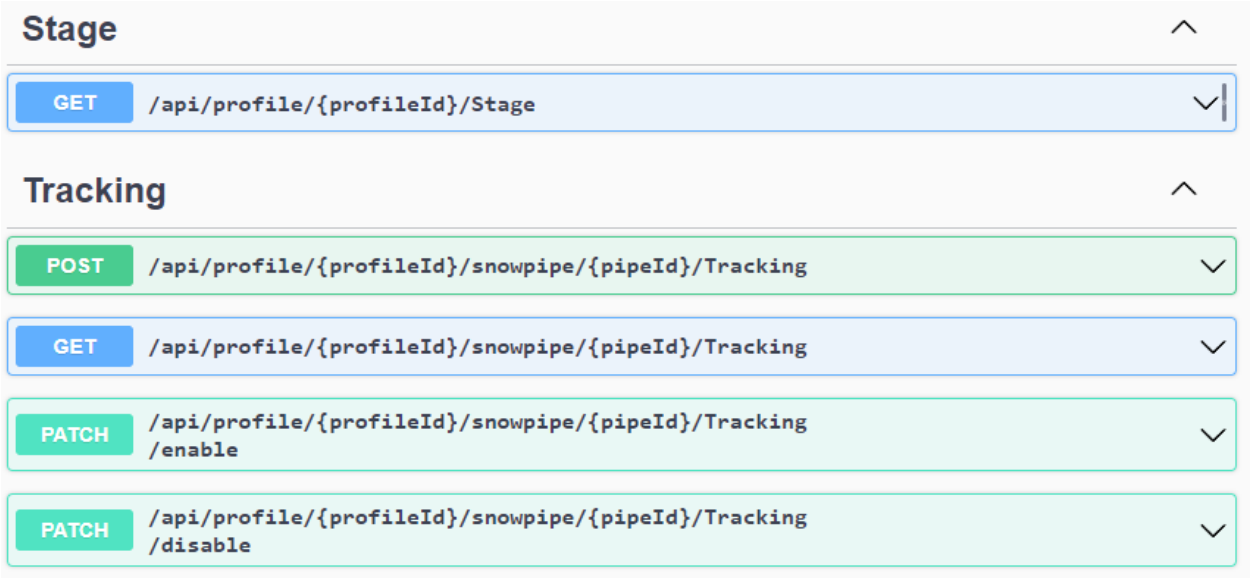
25. Azure Synapse [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/ru-ru/services/synapse-analytics/
26. Dedicated SQL pool (formerly SQL DW) architecture in Azure Synapse Analytics [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/massively-parallel-processing-mpp-architecture
27. Snowflake architecture [Електронний ресурс] – Режим доступу до ресурсу: https://www.snowflake.com/product/architecture/
28. Automating Snowpipe for Amazon S3 [Електронний ресурс] – Режим доступу до ресурсу: https://docs.snowflake.com/en/user-guide/data-load-snowpipe-auto-s3.html
29. Overview of the Kafka Connector [Електронний ресурс] – Режим доступу до ресурсу: https://docs.snowflake.com/en/user-guide/kafka-connector-overview.html
30. Apache Kafka [Електронний ресурс] – Режим доступу до ресурсу: https://kafka.apache.org/
31. O'REILLY REPORT: ARCHITECTING DATA-INTENSIVE SAAS APPLICATIONS [Електронний ресурс] – Режим доступу до ресурсу: https://www.snowflake.com/resource/oreilly-report-architecting-data-intensive-saas-applications/?utm_cta=website-oreilly-saas-report
32. Introduction to Snowpipe [Електронний ресурс] – Режим доступу до ресурсу:https://docs.snowflake.com/en/user-guide/data-load-snowpipe-intro
33. WebAssembly [Електронний ресурс] – Режим доступу до ресурсу: https://webassembly.org/
34. ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0

35. Blazor WASM [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-6.0
36. Bootstrap [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com/
37. Introduction to Identity on ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity
38. What is REST [Електронний ресурс] – Режим доступу до ресурсу: https://restfulapi.net/
39. Richardson Maturity Model [Електронний ресурс] – Режим доступу до ресурсу: https://martinfowler.com/articles/richardsonMaturityModel.html
40. Onion Architecture Is Interesting [Електронний ресурс] – Режим доступу до ресурсу: https://dzone.com/articles/onion-architecture-is-interesting
41. Entity Framework Core [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/ef/core/
42. Key Pair Authentication & Key Pair Rotation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.snowflake.com/en/user-guide/key-pair-auth
43. Гутаревич О. С. Розробка системи управління засобами завантаження даних у Snowflake: матеріали наук.-практ. конф. студентів та аспірантів «Теоретичні та прикладні аспекти розробки комп’ютерних систем 2023» [Електронний ресурс] – Режим доступу до ресурсу: http://econference.nubip.edu.ua/index.php/taacsd/2023/paper/view/2811

ДОДАТОК А. Діаграма бази даних



ДОДАТОК Б. Swagger-специфікація розробленого Web API


Stage
GET
/api/profile/{profileId}/Stage
Tracking
POST
/api/profile/{profileId}/snowpipe/{pipeId}/Tracking
GET
/api/profile/{profileId}/snowpipe/{pipeId}/Tracking
PATCH
/api/profile/{profileId}/snowpipe/{pipeId}/Tracking
/enable
PATCH
/api/profile/{profileId}/snowpipe/{pipeId}/Tracking
/disable