Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Налаштування процесів автоматизованого вивантаження даних з локальної бази даних до сервісів Google Cloud

Виконав: студент 2 курсу, групи
спеціальності 151 – Автоматизація
та комп'ютерно-інтегровані технології

12 грудня Владислав ПАНЧУК
Ім'я ПРІЗВИЩЕ

Керівник:
к. т. н., доцент кафедри КСУ
степінь, звання, посада
Марія ЮХИМЧУК
Ім'я ПРІЗВИЩЕ
« 12 » грудня 2022 р.

Опонент:
к.т.н., доцент кафедри АІІТ
степінь, звання, посада
Ярослав КУЛИК
Ім'я ПРІЗВИЩЕ
« 13 » грудня 2022 р.

Допущено до захисту
Зав. кафедри КСУ
В'ячеслав КОВТУН
« 19 » 12 2022

Вінниця ВНТУ – 2022 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління
Рівень вищої освіти другий (магістерський)
Галузь знань – 15 – Автоматизація та приладобудування
Спеціальність – 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітньо - професійна програма – Інтелектуальні комп'ютерні системи

ЗАТВЕРДЖУЮ
Завідувач кафедри КСУ

В'ячеслав КОВТУН

"03" жовтня 2022 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Панчуку Владиславу Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи. «Налаштування процесів автоматизованого вивантаження даних з локальної бази даних до сервісів Google Cloud»
керівник магістерської кваліфікаційної роботи доцент кафедри КСУ Юхимчук Марія Сергіївна к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ВНТУ від "14" вересня 2022 року №203
2. Строк подання студентом магістерської кваліфікаційної роботи 12.12. 2022 р.

3. Вихідні дані до роботи: Сервіси Google Cloud, розробити сервіс для вивантаження даних з локальної машини до хмарних сервісів,налаштувати вивантаження на BigQuery, Google Storage.
4. Зміст текстової частини: Вступ; 1 Аналіз проблеми та постановка задач, 2 Вибір програмних засобів для розробки сервісу вивантаження даних, 3 Розробка системи автоматизованого вивантаження даних з локальної бази, 4 Економічна частина, Висновки.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Модель тестової міграції даних на хмарний сервіс BigQuery, Модель взаємодії розробленої програми з локальною базою та сервісами Google Cloud -2 шт, вигляд екранів розробленого розробленого сховища даних – 2 шт

1. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1 -3	Юхимчук М. С., доцент кафедри КСУ		
4	Небава М. І., пофесор кафедри ЕПВМ		

2. Дата видачі завдання "03" жовтня 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів Роботи	Строк виконання етапів роботи	Примітка
1	Дослідження актуальності поставленої задачі	04.09.2022р.	Вик.
2	Вибір програмних засобів дл розробки сервісу вивантаження даних	22.09.2022р.	Вик.
3	Розробка системи автоматизованого вивантаження даних з локальної бази	13.10.2022р.	Вик.
4	Економічна частина	03.11.2022р.	Вик
5	Оформлення пояснювальної записки, графічного матеріалу і презентації	08.12.2022р.	Вик
6	Графічні матеріали: – Модель тестової міграції даних на хмарний сервіс BigQuery – Модель взаємодії розробленої програми з локальною базою та сервісами Google Cloud – вигляд екранів розробленого розробленого сховища даних	 01.12.2022 р 01.12. 2022 р. 03.12. 2022 р.	 Вик
7	Захист МКР	20.12. 2022 р.	Вик.

Студент Владислав ПАНЧУК

(підпис) (*Ім'я ПРІЗВИЩЕ*)

Керівник роботи Марія ЮХИМЧУК

(підпис) (*Ім'я ПРІЗВИЩЕ*)

АНОТАЦІЯ

УДК 621.374.415

Панчук В.В. Налаштування процесів автоматизованого вивантаження даних з локальної бази даних до сервісів Google Cloud. Магістерська кваліфікаційна робота зі спеціальності 151 – Автоматизація та комп'ютерно-інтегровані технології, освітня програма – Інтелектуальні комп'ютерні системи. Вінниця: ВНТУ, 2022. 94 с.

В ході виконання магістерської кваліфікаційної роботи було розроблено сервіс для автоматизованої вигрузки даних з локальної бази до хмарних сервісів, які предоставляє Google Cloud. В роботі проведено аналіз актуальності хмарних сервісів на сьогодення, проаналізовані технології, які використовуються у проекті. Також розроблена програма на базі Node.js, який керує вигрузкою даних, аналізує їх, вміє працювати з різними сервісами. За основу було взято BigQuery, CloudSql та Google Storage (Bucket) сервіси з якими взаємодіє розроблена програма, та вигружає до них дані, в кінці описана економічна частина з підрахунками.

Ключові слова: хмарний сервіс, вигрузка даних, Google Cloud, BigQuery, CloudSql.

ANNOTATION

Pachnuk V.V. Setting up automated data upload processes from the local database to Google Cloud services. Master's thesis on specialty 151 - Automation and computer-integrated technologies, educational program - Intelligent computer systems. Vinnytsia: VNTU, 2022. 94 p.

In the course of the master's qualification work, a service was developed for automated data upload from the local database to cloud services provided by Google Cloud. The paper analyzes the relevance of cloud services today, analyzes the technologies used in the project. A program based on Node.js was also developed, which manages data uploading, analyzes them, and is able to work with various services. BigQuery, CloudSql and Google Storage (Bucket) services with which the developed program interacts and uploads data to them were taken as a basis, the economic part with calculations is described at the end.

Keywords: cloud service, data upload, Google Cloud, BigQuery, CloudSql.

ВІДГУК
керівника магістерської кваліфікаційної роботи

студента (-ки) Панчука Владислава Валерійовича
(прізвище, ім'я, по батькові)
на тему: Налаштування процесів автоматизованого вивантаження даних з локальної бази даних до сервісів Google Cloud

Магістерська кваліфікаційна робота є актуальною, тому що останнім часом впровадження хмарних технологій стрімко зростає, завдяки хмарним технологіям освіта стає ще доступнішою, адже, вчитися можна скрізь. Завдяки зростанню популярності хмарних технологій, для навчальних закладів з'являються нові можливості управління навчальним процесом. Одним з основних питань управління є підвищення рівня методичної роботи навчального закладу на усіх його рівнях. Саме тому новітні віртуальні та хмарні технології допомагають зробити освіту більш доступною.

Метою магістерської кваліфікаційної роботи є впровадження системи, яка здатна, на відміну від аналогів, безкоштовно робити вигрузку даних з локальної бази даних на хмарні сервіси, та вигружати їх підлаштовуючись до користувача.

В першому розділі було проведено аналіз актуальності хмарних систем в цілому, їх впли та важливість у сьогоденні. Також був проведений огляд та аналіз хмарних сервісів, які предоставляє Google Cloud. В другому розділі було проведено дослідження та вибір необхідних інструментів для розробки сервісу, основою для реалізації було вибрано платформу Node.js, як посередника між локально базою, та Google Cloud сервісами. Було проведено аналіз хмарних сервісів Google Cloud, так як вони являються одними із самий популярних та часто використовуються компаніями в своїх проектах. В третьому розділі описано процес розробки сервісу: налаштування локальної бази та заповнення її тестовими даними, якими надалі буде оперувати сервіс, ініціалізація проекту node.js та підключення до нього локальної бази та робота з відповідними

підрахунками.

Магістерська кваліфікаційна робота повністю відповідає темі та завданню. Графічна частина МКР відповідає основному змісту пояснювальної записки та виконана у відповідності до встановлених вимог.

До недоліків можна віднести недостатній аналіз хмарних сервісів. Подана магістерська кваліфікаційна робота відповідає вимогам, які висуваються до магістерських кваліфікаційних робіт зі спеціальності 151 - «Автоматизація та комп'ютерно-інтегровані технології», виконана на достатньо високому науково-технічному рівні та заслуговує на оцінку «В» у шкалі ECTS, а Панчук Владислав Валерійович – присвоєння кваліфікації: ступінь вищої освіти магістр, спеціальність «Автоматизація та комп'ютерно-інтегровані технології», освітня програма «Інтелектуальні комп'ютерні системи».

Керівник магістерської кваліфікаційної роботи

к.т.н., доцент кафедри КСУ (посада, науковий ступінь, вчене звання) (підпис) Марія ЮХИМЧУК *Ім'я ПРІЗВИЩЕ*

ВІДГУК
опонента на магістерську кваліфікаційну роботу

студента (-ки) Панчука Владислава Валерійовича
(прізвище, ім'я, по батькові)

на тему: Налаштування процесів автоматизованого вивантаження даних з локальної бази даних до сервісів Google Cloud

В магістерській кваліфікаційній роботі було проведено аналіз сервісів Google Cloud, та можливості взаємодії з ними з сторонніх сервісів, також досконало було проведене дослідження та вибір необхідних інструментів для розробки сервісу. Оскільки метою роботи була розробка програмного забезпечення для автоматизованого вивантаження даних з локальної бази даних на хмарні сервіси Goooogle Cloud, створено сервіс на платформі Node.js, який зчитує дані з реляційної локальної бази типу SQl, та оновлює дані на сервісах, залежно від внутрішніх персональних налаштувань. Також в роботі опубліковані тести, які підтверджують роботоспроможність програмного забезпечення, та можливість його використання в сторонніх продуктах. В коді приведено багато коментарів, які роблять його читання зрозумілим, та наведені описи до кожної сутності. В розділах наведені аргументації вибору тої чи іншої технології та описані ролі кожного компоненту програми, наведеніі відповідні скріншоти, які ілюструються процес розробки програми.

Робота складається з чотирьох розділів, які логічно пов'язані між собою. Магістерська кваліфікаційна робота повністю відповідає темі та завданню.

Матеріал у роботі викладено грамотно, логічно, зрозуміло, що свідчить про вміння студента збирати, систематизувати та аналізувати необхідну інформацію. Висновки, зроблені у пояснювальній записці за отриманими результатами, обґрунтовано. Робота має практичне значення та може бути рекомендована для використання

Графічна частина МКР відповідає основному змісту пояснювальної записки та виконана у відповідності до встановлених вимог.

Подана магістерська кваліфікаційна робота відповідає вимогам, які висуваються до магістерських кваліфікаційних робіт зі спеціальності 151 -

«Автоматизація та комп'ютерно-інтегровані технології», виконана на достатньо високому науково-технічному рівні та заслуговує на оцінку «А» у шкалі ECTS, Панчук Владислав Валерійович – присвоєння кваліфікації: ступінь вищої освіти магістр, спеціальність «Автоматизація та комп'ютерно-інтегровані технології», освітня програма «Інтелектуальні комп'ютерні системи».

Опонент

Доцент кафедри АІІТ _____ Ярослав КУЛИК
(посада, науковий ступінь, вчене звання) (підпис) (*Ім'я ПРІЗВИЩЕ*)

Печатка установи, організації опонент

ЗМІСТ

ВСТУП

Актуальність. Останнім часом впровадження хмарних технологій стрімко зростає, завдяки хмарним технологіям освіта стає ще доступнішою, адже, вчитися можна скрізь. Завдяки зростанню популярності хмарних технологій, для навчальних закладів з’являються нові можливості управління навчальним процесом. Одним з основних питань управління є підвищення рівня методичної роботи навчального закладу на усіх його рівнях. Саме тому новітні віртуальні та хмарні технології допомагають зробити освіту більш доступною.

Окрім цього популярність хмарних технологій широко виросла за останні пару років. Цей факт можна прослідкувати за поведінкою бізнесу, який поступово переходить на сервіси Google.

Використання хмарних технологій може стати у великій нагоді якщо потрібно працювати з штучним інтелектом, аналітикою, масштабним центром обробки даних, або ж центром де ці дані потрібно зберігати у великих об'ємах.

Мета і завдання дослідження. Метою магістерської кваліфікаційної роботи є впровадження системи, яка здатна, на відміну від аналогів, безкоштовно робити вигрузку даних з локальної бази даних на хмарні сервіси, та вигружати їх підлаштовуючись до користувача.

Для досягнення наведеної мети були поставлені та вирішені наступні задачі:

1) проведено аналіз сервісів Cloud DB та проаналізовані методи вигрузки даних;
2) проведено дослідження Google Cloud API, яке надає хмарне середовище;
3) локально піднята реляційна база типу MySQL;
4) написаний сервіс на базі Node.js;
5) проведено дослідження та тестування отриманих результатів.

Об’єктом дослідження є процес вигрузки даних з локальної бази через сервіс написаний на node.js.

Предметом дослідження є методи та засоби програмування, за допомогою яких можна спроектувати сервіс, який здатних відслідковувати зміни в локальній базі, та автоматизовано робити вигрузки на хмарні сервери.

Методи дослідження. У процесі дослідження застосовувалися: платформа nodejs, API Google Cloud, WorkBench, теорії розробки чистого коду на javascript, методика Scrum,мова запитів SQL.

Наукова новизна одержаних результатів.

1. Проведено оцінку роботи існуючої локальної бази даних та запропоновано методи поліпшення зберігання та обробки даних з використаннях хмарних технологій.
2. Спроектовано та розроблено моделі сховища бази даних, що покривають декілька бізнес-потреб АСУ інтернет магазином.
3. Впроваджено пайплан, який реалізує доставку даних до хмарного сервісу.

Практичне значення одержаних результатів полягає в тому, що на основі отриманих теоретичних положень був написаний сервіс, який значно полегшить компаніям розробку та взаємодію з хмарними сервісами Google Cloud.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується строгістю постановки задач, коректним застосуванням усіх методів та засобів функціонування інформаційних систем, вживаючи Agile методику при проведенні реорганізації структури бази даних автоматизованої системи управління інтернет магазином для формування сховища даних у хмарі.

Особистий внесок здобувача. Усі результати отримано автором самостійно.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧ

1.1. Аналіз хмарних сервісів та їх актуальність

За одностайними прогнозами провідних консалтингових компаній світу, швидке вдосконалення та поширення хмарних технологій (cloud computing) зараз є одним з тих ключових трендів, що в найближчі 5-8 років помітно вплинуть на глобальний розвиток не лише IT-індустрії, але й бізнесу, фінансів, державного управління, медицини, освіти і багатьох інших сфер людського життя. А в умовах випереджаючого розвитку ІКТ і чергового спаду світової економіки технологія, яка, наприклад, дозволяє організаціям та іншим суб'єктам відмовитись від значних витрат на власну IT-інфраструктуру на користь отримання всіх необхідних IT-ресурсів онлайн, розглядається як перспективний та рентабельний модернізаційний вибір, оптимальна інвестиція в майбутнє. Достатньо сказати, що, за підрахунками авторитетної International Data Corporation (IDC), вже у 2025 році до 90 % всіх даних людства зберігатиметься у хмарах. У найрозвиненіших регіонах світу вже прийняті стратегічні рішення та плани дій щодо системного та комплексного розвитку хмарних сервісів, розгорнута відповідна робота.

У глобальному вимірі ринок хмарних обчислень стає полем чимдалі жорсткішої конкуренції між провідними світовими IT-корпораціями (Google, Yahoo, Amazon, Microsoft, Zoho, Cisco, Symantec, Fujitsu і ціла низка інших)[14]. Крупні бізнес-гравці, які ще не мають своєї «долі» на цьому ринку, готуються завойовувати її в найближчому майбутньому. Така ситуація додатково інтенсифікує техніко-технологічну гонку, тому нові апаратні рішення, стартапи, програмне забезпечення, розробляються і просуваються у хмарному секторі справді випереджаючими темпами[17].

Працюючи в сфері it технологій я неодноразово зіштовхувався з використанням хмарних сервісів: доводилося працювати саме з Google Cloud [1], та його API (Application Programming Interface). Хмарний сервіс надає низку

інструментів (більшість з них будуть використані у розробці, та будуть описані нижче) таких як (BigQuery, BigTable, Cloud SQL, Virtual Machines (VM's) тощо.

Головний сенс цих сервісів полягає у тому, що користувач, створюючи акаунт на платформі, має доступ до особистої інформації з будь-якого пристрою та з будь-якого місця знаходження. Це є не лише зручно, але й безпечно. Переваги використання хмарних технологій:

1) не потрібні великі обчислювальні потужності ПК - по суті будь-який смартфон, планшет і т.д., при відкритті вікна браузера отримує величезний потенціал.

2) відмовостійкість

3) певний рівень безпеки.

4) висока швидкість обробки даних.

5) ваш власний вінчестер не наповнюється - всі дані зберігаються в мереж

Хмарні обчислення здійснюються з використанням трьох моделей [2]:

1) програмне забезпечення як послуга (SaaS). За моделлю SaaS забезпечує доступ до апаратних компонентів, а саме до серверів, мереж, сховищ. За SaaS можуть надаватися різноманітні послуги – від веб-пошти до управління запасами, обробки даних Перевагою такої моделі є те, що кінцевий користувач може вільно користуватися послугою з будь-якої точки світу;

2) платформа як послуга (PaaS). PaaS у хмарі визначається як набір програмних продуктів та засобів розроблення, що розміщені на інфраструктурі провайдера. Це операційні системи, управління базами даних, засоби тестування та розробки. шлюзи або програмне забезпечення, встановлене на комп'ютері клієнта.

3) інфраструктура як послуга (IaaS). IaaS бізнес-модель поширення програм споживачам, при якій постачальник розробляє Web-додаток і самостійно управляє ним з метою використання через інтернет. Основною перевагою використання моделі SaaS є відсутність витрат, пов'язаних з придбанням ліцензії, її встановленням та обслуговуванням. За необхідності

клієнт звертається на сервер, працює з програмним забезпеченням і сплачує лише орендну плату за користування й зберігання.

Можна добавити пару слів про компанію robota.ua, та її процес становлення з Google Cloud[6]

“У 2017 році компанія вирішила розширити кількість команд, які працюють над розробкою продукту. Для того, щоб робота відбувалася якісно і злагоджено, фахівці шукали аналітичний інструмент, який буде оцінювати, наскільки ефективне для користувачів те чи інше доопрацювання. В той момент було прийнято рішення про використання сервісу BigQuery в Google Cloud Platform — хмарної бази даних з високою швидкістю обробки величезних масивів даних.

«Це рішення стало переломним моментом, поштовхом, щоб застрибнути в Google Cloud», — розповідає Олександр Марченко, Chief technical office в robota.ua.”

До появи хмарних обчислень компанії мали зберігати всі свої дані та програмне забезпечення на власних жорстких дисках і серверах. Що більша компанія, то більше сховища їй потрібно. Цей спосіб обробки даних не можна швидко масштабувати. Наприклад, якби про ваш бізнес поширювалася інформація, і у вас раптом з’явилося багато онлайн-замовлень, ваші сервери, ймовірно, вийшли б з ладу. Хороший бізнес означав важку роботу для IT-відділу.

Не лише підприємства отримують вигоду від хмарних обчислень. Хмара також змінила наше життя як окремих людей. Багато з нас щодня користуються хмарними сервісами. Коли ми оновлюємо свій статус у соціальних мережах, переглядаємо нові потокові серіали або перевіряємо свої банківські рахунки, ми, швидше за все, використовуємо програми, розміщені в хмарних службах. Доступ до цих програм здійснюється через підключення до Інтернету, а не встановлюється на наші жорсткі диски чи пристрої.

Сьогодні хмарні технології означають, що компанії можуть масштабувати та адаптуватися зі швидкістю та масштабом, прискорювати інновації, сприяти гнучкості бізнесу, оптимізувати операції та зменшувати витрати. Це може не

тільки допомогти компаніям подолати нинішню кризу, але й призвести до прискореного сталого зростання. Згідно з нашим дослідженням Future Systems, компанії, які мають більш стратегічний підхід до технологій, досягають кращих фінансових результатів. Вони досягають більш ніж удвічі більшого середнього зростання прибутку, ніж компанії, які повільно впроваджують і використовують свої технології. Насправді 95 відсотків лідерів запровадили складні хмарні сервіси[7].

Класифікація за типом середовища хмарного сервісу:

Загальнодоступні хмарні служби — це служби, в яких послуги, які надає постачальник кількох клієнтів через Інтернет, називаються загальнодоступними хмарними службами. Наведені вище приклади SaaS, IaaS та PaaS надають загальнодоступні хмарні послуги. Найбільшою перевагою використання загальнодоступних хмарних сервісів є можливість широко розподіляти ресурси, що дозволяє організаціям надавати більше можливостей для співробітників, ніж вони могли б самостійно.

Приватні хмарні служби — це служби, де служби, які постачальник не робить загальнодоступними для корпоративних користувачів або передплатників, називаються приватними хмарними службами. У моделі служб приватної хмари програми та дані доступні через внутрішню інфраструктуру організації. Платформа та програмне забезпечення обслуговують ту саму компанію і недоступні зовнішнім користувачам. Компанії, які працюють з особливо конфіденційними даними, наприклад у сфері охорони здоров'я та банківської справи, часто використовують приватні хмари, щоб скористатися перевагами передових протоколів безпеки та розширити ресурси у віртуалізованому середовищі за необхідності.

У гібридному хмарному середовищі приватне хмарне рішення поєднується із загальнодоступними хмарними службами. Ця схема часто використовується, коли організації необхідно зберігати конфіденційні дані в приватній хмарі, але вона хоче, щоб співробітники мали доступ до загальнодоступних хмарних програм і ресурсів для повсякденного спілкування і спільної роботи.

Запатентоване програмне забезпечення використовується для забезпечення зв'язку між службами хмар, часто через єдину консоль управління IT.[9]

На підставі короткого опису хмарних сервісів можна дійти невтішного висновку, що тема хмарних обчислень досить актуальна нині, оскільки попит і волатильність хмарних сервісів збільшується кожні 16 років. Хмарні обчислення тісно пов'язані з бізнесом та сучасними технологіями, охоплюючи більшість сфер нашого життя.

У рамках роботи хмарний сервіс Google Cloud буде вважатися сервісом, що швидко розвивається, що вимагатиме рішень для прискорення ефективності його роботи з організаційної точки зору, даний вид сервісу більше орієнтований на масовий сегмент ринку (по облич), але в той же час вона тісно пов'язана з бізнесом.

Спільної думки щодо визначення економічної моделі немає. Однак, навіть якщо елементи варіюються від одного визначення до іншого, більшість представників, що використовуються в літературі, включають чотири взаємопов'язані компоненти: опис ціннісної пропозиції компанії (послуги), технології (сервісної платформи), ланцюжка створення вартості та джерела доходу (модель доход). Ці компоненти слідують моделі STOF, дотримуючись визначення бізнес-моделі як «плану надання послуг, який описує визначення послуги та очікувану вартість для цільової групи, джерело доходу та забезпечує архітектуру для надання послуг, включаючи опис необхідних ресурсів, а також організаційних та фінансових домовленостей між учасниками господарюючими суб'єктами, включаючи опис їхньої ролі та розподілу витрат та доходів між господарюючими суб'єктами:

- сервіс: опис передбачуваної вартості, поставленої вартості, очікуваної вартості, сприйнятої вартості

- технологія: опис технічної архітектури, сервісних платформ, пристроїв, додатків

- організація: опис суб'єктів , ролі, взаємодії, стратегії та цілі, ціннісні заходи.

- фінанси: опис джерел інвестицій, джерел витрат, джерел доходів, джерел ризику та ціноутворення

Отже, впровадження хмарних технологій дозволяє розширити простір для роботи з інформацією, підвищити надійність, зручність, економічність і мобільність бізнес-процесів без додаткових витрат.

1.2 Актуальність автоматизованого вивантаження даних до хмарних сервісів з локальної бази даних

Одним з недоліків хмарних сервісів є те, що дані доступні тільки в режимі онлайн (детальніше про недоліки описано в розділі 1.3), перебуваючи в сучасних реаліях війни, коли країна потерпає від щоденних атак з боку сусідньої держави, в такому випадку дуже сильно страждає енергетична інфраструктура країни, та вся її енергосистема, це супроводжується регулярними відключеннями світла або ж повного блекауту. В такому випадку наявність локальної бази на локальних машинах розробників, аналітиків чи то будь-яких інших спеціалістів позволяє продовжувати працювати без пауз, та координувати дії локально.

Відновлення даних із пошкоджених фізичних серверів і жорстких дисків може бути досить проблематичним. Якщо фізичний пристрій сильно пошкоджено, відновлення може бути неможливим. Ці проблеми можуть призвести до втрати бізнесом важливих даних, особливо якщо для них не створено відповідну резервну копію. З хмарними обчисленнями ці проблеми не настільки актуальні.

Коли дані зберігаються в хмарі, вони зазвичай зберігаються постачальником у кількох місцях. Це означає, що ваші дані не просто зберігаються в одному фізичному місці. Навіть якщо у них виникли проблеми з одним із їхніх серверів або великих сховищ, ваші дані в безпеці, оскільки їх копія знаходиться в іншому місці.

Малі підприємства рідко мають інфраструктуру чи фінанси для встановлення складних і безпечних систем фізичного резервного копіювання.

Але завдяки хмарним обчисленням і сховищам вони можуть отримати відмінний сервіс за доступною ціною.

Розробка власного сервісу на базі Node.js, який дозволяє одноразово чи автоматично вигружати дані до хмарних сервісів за допомогою дампів (mysqldump), чи окремими запитами, за допомогою Google Cloud API. Хмарні сервіси Google позволяють робити авторизацію за допомогою їх власного CLI (Command Line Interface).

Важливим файтором, являється гнучкість у відправці даних на сервер, для цього є декілька варантів:

1) одноразова відправка (якщо команді потрібно оновити базу на сервері)

2) автоматизована відправка на базі крону, який заплановує графіки оновлення бази.

3) виконання окремих запитів на оновлення

Звісно такий сервіс не являється новітнім, та вже має низку платних схожих програм, які розроблялись професійними командами, але на відміну від аналогів – розроблене програмне забезпечення являється безкоштовним, та має ряд привабливих нововведень: варіанти налаштувань вигрузок даних, та сегментоване розподілення даних в залежності від направленої тематики під яку налаштована база.

1.3 Переваги та недоліки використання хмарних сервісів

Однією з найновіших форм зберігання електронних документів є хмарне зберігання. Зараз ця поширена технологія активно розвивається та модифікується на ринку систем електронного документообігу (ВЕД). В цей час цей метод зберігання популярний, особливо для малих та середніх організацій у сфері бізнесу. Віддалене зберігання документів по порівняно з локальним надає переваги як споживачам, так і підприємствам.

Розглянемо основні переваги, що дає хмарне зберігання електронних документів для користувачів та організацій.

Одні з найголовніших переваг – доступність і мобільність. Доступність означає можливість отримати доступ до всіх документів, файлів, папок у хмарі з будь-якої точки світу. Звичайно, за умови, що є необхідні облікові дані та доступ до Інтернету. Також, не важливий вигляд самого пристрою, доступ до інформації, що зберігається на хмарі, можна отримати з комп'ютера, ноутбука, планшетного ПК або мобільного пристрою, підключеного до інтернету. Персонал із щільним графіком роботи або ті, хто живе далеко від корпоративного офісу, можуть використовувати цю функцію, щоб завжди бути в курсі подій із клієнтами та колегами. Також усунено необхідність перенесення файлів між пристроями, що дратує користувачів, і навіть ускладнять процес роботи з документами, тобто. є можливість синхронізації даних. Крім того, файли залишаються однаковими на всіх пристроях, оскільки вони автоматично оновлюються при зміні. Користувач завжди має останню версію файлу, незалежно від того, коли і як він витягнутий.

З вищеперелічених переваг випливає таку перевагу – це розширення співробітництва. Якщо у бізнесі два або більше співробітників, то співпраця має стати пріоритетом для компанії. Хмарні обчислення спрощують процес співробітництва. Точно так, як дратує передача файлів між пристроями туди і назад, дуже складно відправляти десятки електронних листів тільки для того, щоб ділитися файлами. Із хмарним сховищем для цього є рішення. Члени групи можуть легко та безпечно переглядати, обмінюватися інформацією на хмарній платформі. Деякі хмарні послуги навіть надають спільні соціальні простори для зв'язку співробітників в організації, що підвищує зацікавленість. Співпраця може бути можливою і без використання хмарних обчислень, але вона буде менш спрощеною та ефективною.

Наступна перевага – зручність використання. Користувачі можуть легко перетягувати документи та файли до хмарного сховища. Самі дані легко зберегти у хмарі, для цього не потрібно жодних технічних знань. Хмарні системи зберігання мають можливість легко поділитися доступом у хмарному середовищі з іншим користувачем. Дані, що зберігаються у хмарі, легко та

безпечно передаються клієнтам та колегам. Також хмарний інтерфейс для роботи з документами, як правило, наочний та інтуїтивно зрозумілий, що сприяє більш швидкому навчанню та роботі з ним співробітників організації.

Далі можна виділити таку перевагу як економічність. Компанії та приватні особи, які використовують хмарні служби, з більшою ймовірністю скоротять свої експлуатаційні витрати, ніж ті, хто все ще використовує власні рішення чи зовнішні жорсткі диски. Організація може не купувати дорогі, потужні комп'ютери та програмне забезпечення, не створювати власний центр обробки даних, а платити тільки за оренду такого обладнання, використовуючи всі потужності провайдера. Оплата при цьому провадиться зазвичай раз на місяць. Також, варто зазначити, що при переміщенні даних у хмару вони зникають з пристроїв та обладнання. Тобто дані не займають цінний простір вдома або в офісі, тому що провайдер пропонує місце для віртуального зберігання. Також відпадає потреба у наймані IT-персоналу, який займається обслуговуванням СЕД.

Наступна перевага – висока технологічність та продуктивність. Як було зазначено вище, під час роботи з документами використовуються платформа провайдера. Це дозволяє використовувати її для зберігання та обробки своїх даних з використанням усіх потужностей комп'ютерів, оскільки хмарні постачальники зазвичай використовують сучасні високопродуктивні обладнання та технології, спеціально створені для обробки величезної кількості даних. Відповідно, хмарні СЕД мають можливість масштабованості.

Тепер розглянемо важливу перевагу – контроль якості. Збитки успіху бізнесу може завдати неякісна, непослідовна звітність. У хмарній системі всі документи повинні зберігатися в одному місці та в одному форматі. Завдяки тому, що всі отримують доступ до однієї інформації, є можливість підтримувати «несуперечність» своїх даних, уникати людських помилок і мати чіткий запис будь-яких змін або оновлень.

Ще одна перевага – автоматичне оновлення програмного забезпечення. Для працівників, повністю завантажених своєю роботою, дратівливим чинником

може стати очікування встановлення оновлення системи. Хмарні програми оновлюються автоматично, тому IT-відділу не потрібно виконувати ручне оновлення для всієї організації. Це заощаджує дорогоцінний час IT-персоналу та гроші, витрачені на консультації.

Далі розглянемо таку перевагу як надійність та безпека даних. Хмарні сервіси, як правило, забезпечують швидке відновлення даних для всіх видів надзвичайних ситуацій, від стихійних лих до перебоїв у подачі електроенергії та збоїв у системі. Для запобігання витоку даних у хмарній системі зберігання документів використовується резервне копіювання. Варто зазначити, що клієнт практично не бере участі в налаштуванні забезпечення безпеки даних, що по-багатьом знижує ризик помилок. Таким чином, поломка обладнання ніяк не позначиться на збереженні даних, тому що інформація зберігається на сервері в хмарі. Крім того, щоб уникнути викрадення даних хакерським шляхом у багатьох хмарних СЕД використовується кодування. Вид кодування зазвичай клієнт може вибрати. При цьому інформація кодується ще до того, як покине мережу організації. Резервне копіювання здійснюється після кодування. Таким чином, хмарна система роботи з документами пропонує досить надійні рішення для збереження даних, проте не варто забувати, що практично будь-яку систему захисту можна обійти.

Ідеального способу зберігання електронних документів на даний момент немає. Завжди є ймовірність інформаційного витоку та втрати даних, внаслідок злому, поломки техніки чи масштабного збою системи. Однак системи хмарного документообігу, як і самі хмарні технології, зараз перебуває на етапі розвитку і з кожним роком з'являються нові рішення для їх вдосконалення, а також усунення помилок.

При переході на хмарне зберігання документів необхідно враховувати, що далеко не всі інформаційні системи, що використовуються в сьогоднішніх компаніях, готові і можуть бути переведені в хмару, а в деяких випадках такий перехід просто недоцільний. Наприклад, сюди можна віднести системи, простий

яких може призвести до катастрофічних наслідків чи втрати бізнесу, наприклад, внутрішня кооперативна система банка1.

Тому не можна сказати, що зараз хмарна система зберігання є найоптимальнішою, але цілком можна припустити, що в майбутньому завдяки розвитку у сфері IT та система електронного документообігу хмарна технологія замінить усі традиційні методи зберігання документів.

У будь-чого в цьому світі є дві сторони медалі, тому дуже важливо висвітлити недоліки, які, нажаль, присутні в хмарних технологіях.

Повна залежність клієнта від Інтернету. Будь-який збій в інтернеті призводить до обмеження, а то й повної зупинки доступу користувача до даних, що може серйозно уповільнити роботу організації. Проте, цей недолік негаразд істотний, т.к. проблеми з інтернет-з'єднанням нині не часті. Що стосується втрати даних при відключенні інтернету, рішенням може стати резервне копіювання даних, яке надає більшість провайдерів.

Технічні неполадки та збої в системі. Будь-яка інформаційна технологія завжди схильна до збоїв та інших технічних проблем. Одна з можливих проблем у роботі з хмарним обладнанням є ймовірність, що хмарному сервісі може статися поломка, яка призведе до збою всієї системи. Незважаючи на те, що трапляється таке нечасто, у хмарних центрах є приклади помилок, які призводять до поломки серверів.

Багато сервісів хмарного зберігання мають вбудовані функції безпеки, завдяки чому навіть не мають збоїв в історії роботи своєї системи. Але навіть якщо після поломки дані можна буде відновити, наприклад, за допомогою резервного копіювання такий збій на тривалий час призупинить роботу офісу або організації, що може призвести до простоїв.

Відсутність контролю над даними. Перевага, завдяки якій клієнту не потрібно витрачати час та ресурси на обслуговування та контроль своєї хмари, є також серйозним недоліком. Замовник може контролювати та керувати лише додатками, даними та службами, що працюють поверх системи, але не керувати самою внутрішньою інфраструктурою.

Підприємства можуть зіткнутися з втратою контролю над конфіденційними даними. Проблема тут полягає в тому, що при використанні сторонніх служб обміну файлами дані зазвичай витягуються за межі IT-середовища компанії, а це означає, що конфіденційність даних знаходиться поза контролем підприємства. Оскільки більшість хмарних сервісів використовує резервне копіювання своїх даних у режимі реального часу, багато даних, які не призначені для спільного використання, можуть зрештою переглядатися неавторизованим персоналом. Найкращий спосіб уникнути такого ризику – переконатися, що провайдер зашифровує файли організації під час зберігання.

Довічні витрати. Зручність однієї низької щомісячної орендної плати спочатку виглядає привабливою, особливо для компаній на старті свого бізнесу. Однак, при використанні загальнодоступного хмарного сховища, ціна протягом багатьох років може помітно зрости.

Крім того, якщо підрахувати всі витрати протягом трьох, чотирьох або п'яти років за оренду сервера для хмари для роботи з документами, то залишається питанням - чи буде це дійсно дешевше використання свого обладнання та локальної мережі.

Обмежена пропускна спроможність. Не всі хмарні провайдери створено однаково. В ідеалі варто користуватися послугами постачальника, що пропонує необмежену пропускну здатність.

2 ВИБІР ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ СЕРВІСУ ВИВАНТАЖЕННЯ ДАНИХ

У попередньому розділі було основні поняття про хмарні сервіси. Тепер необхідно розглянути процес розробки програми для вигрузки даних до хмарних сервісів та інструменти, які використовуються для її створення.

2.1. Аналіз та вибір мови програмування для реалізації сервісу автоматизованого вивантаження даних

Хмарний сервіс Google Cloud надає нам можливості розробки на низці цікавих мов або фреймворках таких як: Java, Go, C#, Node.js, Ruby тощо. Але маючи достатній досвід програмування на Node.js, вибір пав саме на цей фреймворк.

Node.js — це серверне середовище з відкритим кодом. Node.js є кросплатформним і працює на Windows, Linux, Unix і macOS. Node.js — це серверне середовище виконання JavaScript. Node.js працює на V8 JavaScript Engine і виконує код JavaScript поза веб-браузером.

Node.js дозволяє розробникам використовувати JavaScript для написання інструментів командного рядка та сценаріїв на стороні сервера. Функціональність запуску сценаріїв на стороні сервера створює динамічний вміст веб-сторінки перед тим, як сторінка надсилається у веб-браузер користувача. Отже, Node.js представляє парадигму «JavaScript всюди»[6], об’єднуючи розробку веб-додатків навколо однієї мови програмування, а не різних мов для серверних і клієнтських сценаріїв.

Фреймворк має наступні властивості:

1) Асинхронна одно-нитева модель виконання запитів.
2) Неблокуючий ввід/вивід.
3) Рушій JavaScript Google V8.

У традиційному підході до створення веб-серверів, для кожного вхідного запиту або з'єднання сервер створює новий потік виконання або навіть розганяє

новий процес для обробки запиту і надсилає відповідь. Концептуально це має сенс, але на практиці це викликає великі накладні витрати [13].

Наявність великої кількості потоків може призвести до того, що сильно завантажена система витрачає дорогоцінні цикли на планування потоків і перемикання контекстів, що додає затримки і накладає обмеження на масштабованість і пропускну здатність.

Для середовища Node.js також за замовчуванням існує менеджер пакунків npm (Node Package Manager). Має власну CLI (клієнт командного рядка), а також онлайн-базу даних публічних та приватних пакунків, яка називається реєстром npm. Реєстр доступний через клієнт, а доступні пакунки можна переглядати та шукати через вебсайт npm.

Node.js – бере за основу мову javascript, яка в свою чергу має асинхронність та являється функціональною мовою програмування, звісно, є можливість створювати класи які являються парадигмою ООП, але це являється «синтаксичним сахаром» вбудованим у мову, та представляє собою надбудову на сутнісю об’єкта, чим являється абсолютно все в javascript.

Javascript мова, яка має асинхроннісь - асинхронне програмування як один із шляхів вирішення проблеми продуктивності. Це допоможе істотно підвищити пропускну здатність серверів і скоротити час відповіді на клієнтські запити - дві проблеми, які часто виникають при розробці високонавантажених систем.

Всі користувачі хоч раз стикалися з такою ситуацією, коли ви заходите на веб-ресурс, а все працює повільно і курсор замість стрілки показує процес завантаження, таким чином операційна система ніби говорить нам: «Якийсь затяжний процес завершується».

Подібна подія свідчить, в першу чергу, про некоректне використання процесорного часу, незважаючи на те, що більшість сучасних пристроїв мають кілька процесорів. Все, що потрібно зробити в такій ситуації, це перенести наступне завдання на інше вільне ядро процесора для виконання. За допомогою цього дистрибутива різні завдання можуть виконуватися одночасно, навіть в однопоточному JavaScript - це асинхронний підхід до програмування. Таким

чином, довгі мережеві запити не блокуватимуть виконання основного потоку, а працюватимуть лише незалежно у фоновому режимі. Це актуально через те, що багато сучасних браузерів мають таку можливість і можуть виконувати асинхронні події.

Асинхронний JavaScript може включати два стилі такого коду, які будуть зустрічатися під час розробки в JS: зворотні виклики. Зворотні виклики — це старий метод реалізації асинхронного коду. Обіцянки. Promises — це підхід до впровадження асинхронного коду.

Асинхронні зворотні виклики є дещо застарілою практикою, але вони все ще поширені, тому ви повинні знати про них. Його суть полягає в тому, що JS-код, який працює у фоновому режимі, після завершення роботи викликає функцію зворотного виклику, яка сповіщає про те, як була виконана робота і чи були якісь труднощі при її виконанні. Такі функції завжди передаються як аргумент іншим функціям, які мають виконуватися в потоці. Зворотні виклики не завжди повинні виконуватися миттєво, а лише за потреби. Їх універсальність полягає в тому, що вони дають можливість контролювати порядок виконання інших функцій. Крім того, ви можете контролювати всі передані дані за допомогою зворотних викликів.

Promises — це сучасний стиль асинхронної розробки JavaScript, який перевірено знову і знову. Обіцянки - це об'єкти успішної або неуспішної асинхронної операції. Ви також можете сказати, що обіцянка — це проміжний і «підвішений» стан коду. Простими словами, це обіцянка браузера повернутися до нас із відповіддю якомога швидше. Обіцянки мають такі переваги перед зворотними викликами:

- вони дозволяють об'єднати кілька асинхронних операцій в один блок;
- всі промісні виклики виконуються в найсуворішому порядку, який розміщується в черзі подій;
- помилки обробляються більш ефективно;
- обіцянки виключають інверсії в контролі.

Найпомітніші особливості функціонального програмування такі:

• мови функціонального програмування розроблені навколо концепції математичних функцій, які використовують умовні вирази та рекурсію для виконання обчислень.

• функціональне програмування підтримує функції вищого порядку та функції відкладеного оцінювання.

• мови функціонального програмування не підтримують керування потоком, наприклад оператори циклу та умовні оператори, такі як оператори If-Else та Switch. Вони безпосередньо використовують функції та виклики функцій.

• як ООП, функціональні мови програмування підтримують такі популярні концепції, як абстракція, інкапсуляція, успадкування та поліморфізм. Функціональне програмування має такі переваги:

• функціональне програмування не має стану, тому немає побічних ефектів і можна писати код без помилок.

• функціональні мови програмування не мають змінних стану, тому змінити стан не проблема. «Функції» можуть бути запрограмовані паралельно як «інструкції». Такий код підтримує легке повторне використання та перевірку.• Функціональна програма складається з незалежних блоків, які можуть працювати одночасно. Тому такі програми більш ефективні.

• функціональне програмування підтримує вкладені функції.

• функціональне програмування підтримує ліниві функціональні конструкції, такі як ліниві списки, ліниві карти тощо.

Як недолік, функціональне програмування вимагає багато пам'яті. Оскільки немає поняття стану, нові об’єкти повинні створюватися кожного разу, коли виконується дія. Функціональне програмування використовується в ситуаціях, коли над одним набором даних необхідно виконати багато різних операцій.

Для відмалювання інтерфейсу був вибраний шаблонізатор ejs, на відміну від монструозних фронтендових фреймворків таких як (Vue, Angular, React etc) він набаго простіший у використанні, не потребує додаткових знань та фішок

фреймворків, набагато легший для проету. З мінусів є відсутність вбудованих інструментів ререндерингу.

Приклад підключення до проекту

npm install ejs –save

app.set("view engine", "ejs");

Розглядаючи загрузку даних в Google BigQuery (рис. 1.1), зазвичай потрібно створити набір даних та таблицю користуючись інтерфейсом, але описаний надалі сервіс зможе керувати цим процесом самостійно, потребуючи мінімальних зусиль.

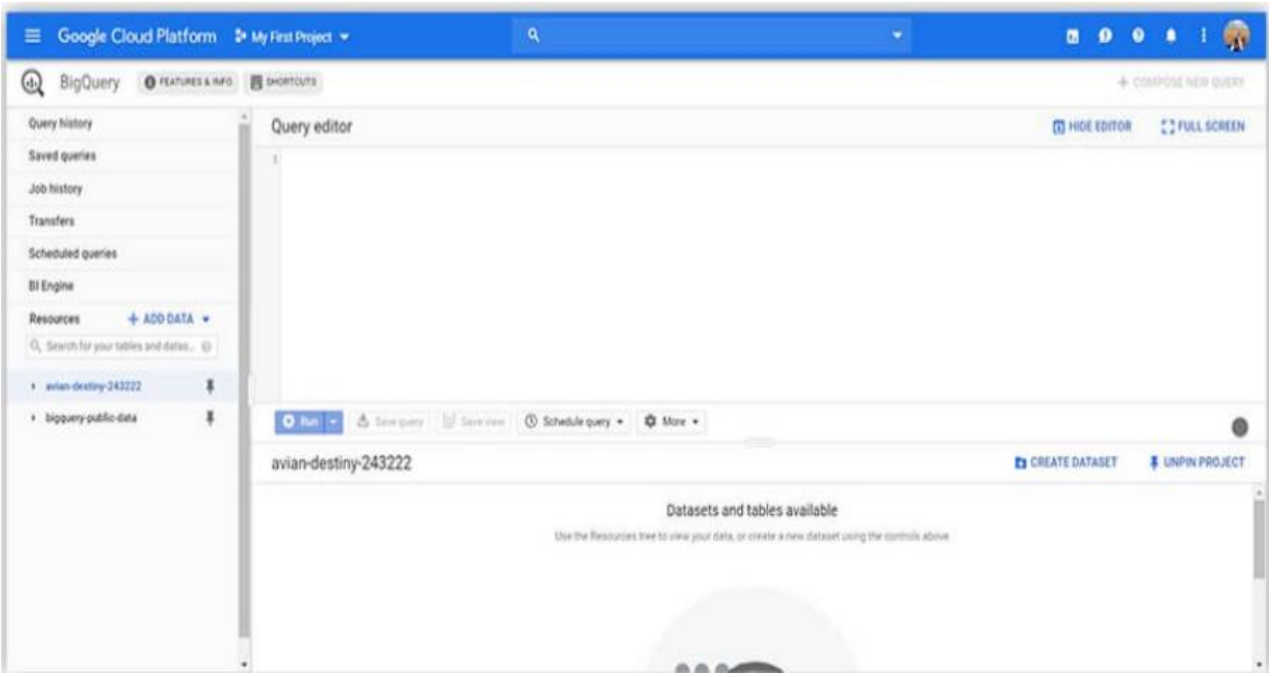


Рисунок 1.1 – інтерфейс створення Google Cloud Bucket

Далі у вікні «Create dataset» вказати ідентифікатор набору даних, потім вибрати місце обробки даних та встановити термін зберігання таблиці.

Локальна база типу sql – створюється з локальним ip 127.0.0.1 на порту 3306, та підключена до менеджеру MySql Workbench. Перша версія MySQL Workbench була випущена у вересні 2005 року[5].

Вибір менеджеру для локальної бази даних пав на MySQL Workbench (рис. 1.3). MySQL Workbench був першим сімейством продуктів, який був доступний

в двох варіантах. Щоб привернути розробників в основну команду розробки, комерційна стандартна версія програми (Standard Edition) пропонується поверх вільної версії (Community Edition), що розповсюджується під ліцензією GNU GPL.[6] «Community Edition» є повнофункціональним продуктом, що має всі основні можливості комерційного варіанту. Будучи основою для всіх майбутніх релізів, він буде отримувати користь від всіх майбутніх зусиль, прикладених для розвитку продукту. «Standart Edition» розширює «Community Edition» серією модулів і плагінів, що дозволяють оптимізувати робочий процес і, тим самим, заощадити час і уникнути помилок[7].

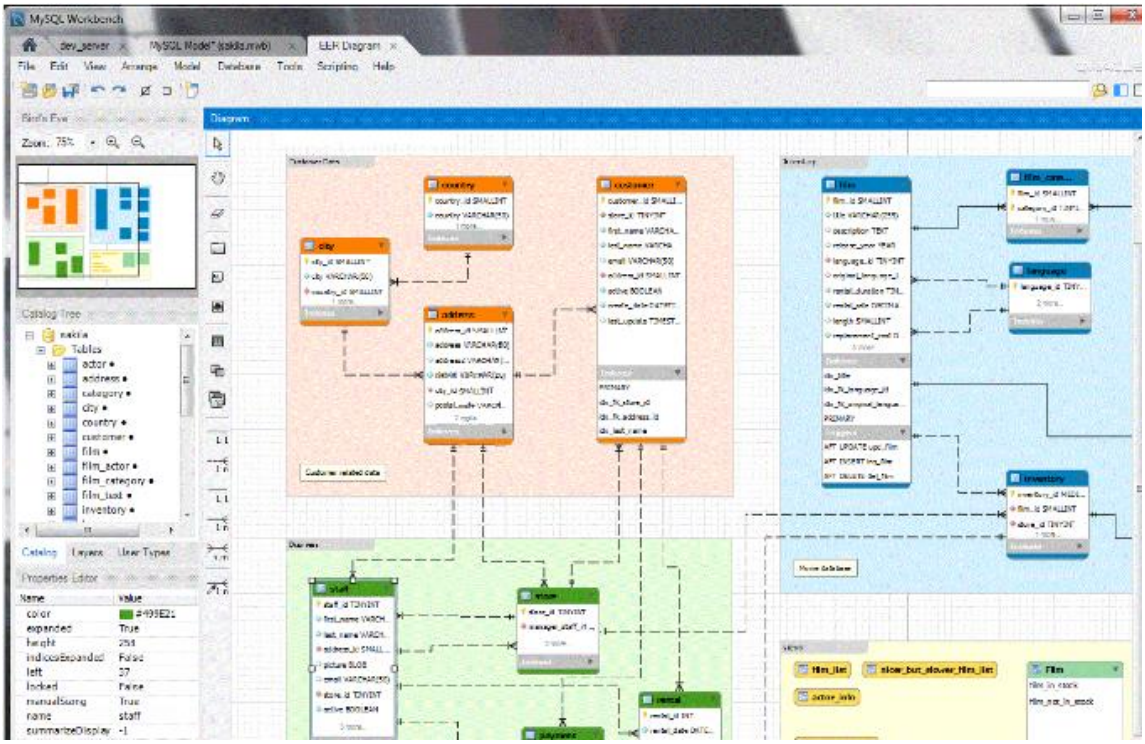


Рисунок 1.2 – інтерфейс MySql Workbench

2.2. Аналіз та вибір сервісу Google Cloud

Нижче проведено перелік переваг вибраного хмарного сервісу, та його можливості. Підвищення продуктивності завдяки швидкому доступу до інновацій. Системи Google можуть доставляти оновлення ефективно і щотижня.

Менше збоїв, коли користувачі впроваджують нову функціональність. Замість великих руйнівних пакетів змін, Google забезпечує керовані поліпшення в безперервному потоці.

Співробітники можуть працювати з будь-якої точки світу. Вони можуть отримати повний доступ до інформації на різних пристроях з будь-якої точки світу через веб-додатки на базі хмари Google.

Хмара Google забезпечує швидку спільну роботу. Багато користувачів можуть брати участь в проектах і отримувати до них доступ одночасно, оскільки дані зберігаються в хмарі, а не на їх комп'ютерах.

Інвестиції в Google забезпечують безпеку клієнтів. Клієнти отримують вигоду від інвестицій, заснованих на процесах і фізичної безпеки, зроблених Google. Google наймає провідних експертів з безпеки.

На вразливих пристроях зберігається менше даних. Мінімальні дані зберігаються на комп'ютерах, які можуть бути скомпрометовані після того, як користувач перестане використовувати веб-додатки в хмарі.

У клієнтів підвищується час безвідмовної роботи та надійність. Якщо центр обробки даних з будь-якої причини недоступний, система негайно перемикається на додатковий центр без будь-якого переривання обслуговування, видимого користувачам.

Контроль і гнучкість, доступні для користувачів. Ви контролюєте технології і володієте своїми даними в додатках Google. Якщо Ви вирішили більше не користуватися сервісом (рис. 1.3), то можете отримати свої дані з хмари Google.

Масштабна економія Google дозволяє клієнтам витрачати менше. Google мінімізує накладні витрати і об'єднує невелику кількість конфігурацій серверів. Google Cloud надає обширні послуги, під різні бізнес процеси

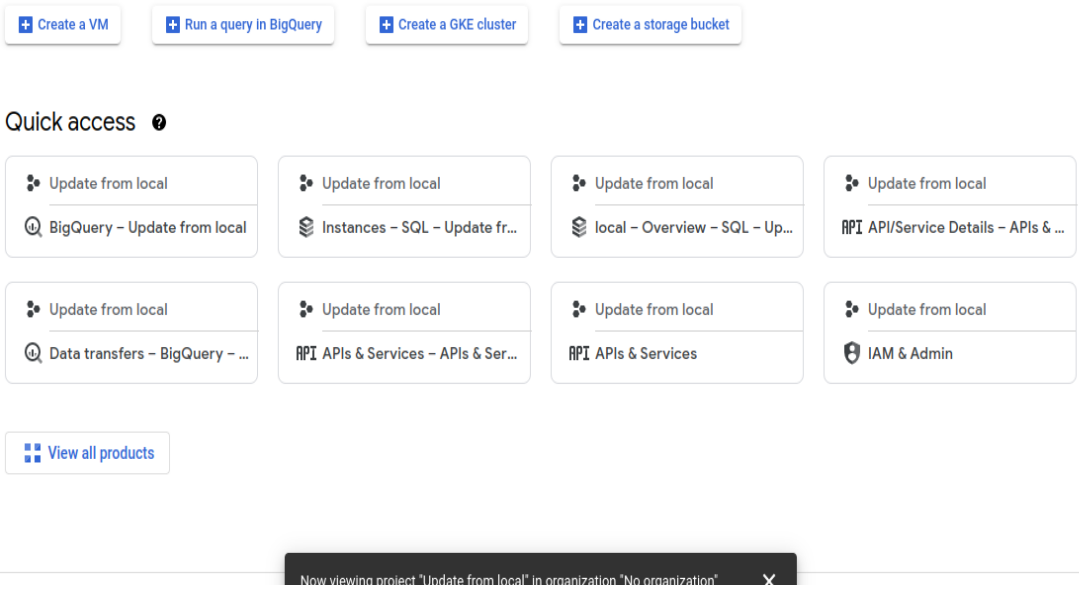


Рисунок 1.3 – список сервісів Google Cloud

Для баз на «хмарі» було вибрано BigQuery та CloudDB, які мають ряд характеристик, які підходять для розробки програми. До того ж вони Sql орієнтовні, що відповідає локальній базі.

BigQuery – це хмарний сервіс Google, призначений для роботи з Big Data, запущений у 2011 році. Він пропонує онлайн-сховище даних (рис. 1.4), дозволяючи надійно зберігати та швидко обробляти великі масиви інформації без необхідності задіяти для цього окремий сервер.

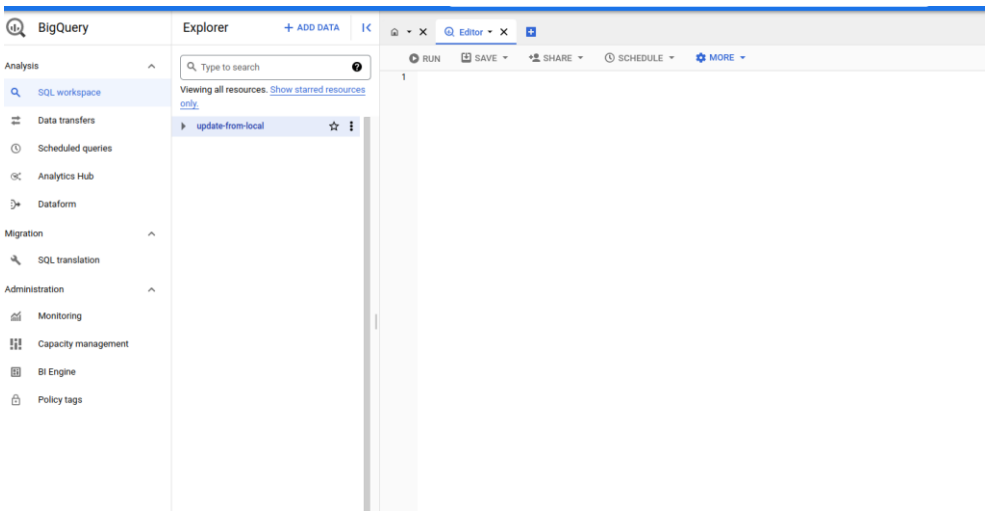


Рисунок 1.4 – інтерфейс Google Cloud BigQuery

Google BigQuery є PaaS-сервіс («платформа як послуга»), який підтримує більшість функцій СУБД. Він входить до складу Google Cloud Platform, де є ще кілька десятків програм для аналізу, зберігання та обчислення даних.

По суті, BigQuery є хмарною БД з необмеженим сховищем та високою швидкістю обробки великих масивів даних. Він має великий функціонал, його користувачі можуть оперативно завантажувати масштабний обсяг даних, зберігати їх у вигляді двовимірних таблиць, звертатися до них за допомогою SQL-запитів, а також зберігати та вивантажувати їх результати.

Управління даними – сервіс дозволяє створювати та видаляти таблиці та функції користувача, а також імпортувати дані у форматах JSON, Avro, Parquet або CSV. Щоб використовувати дані в Big Query, їх потрібно завантажити до сервісу Google Storage, а вже звідти провести імпорт даних через API. Також підтримується прямий імпорт та стримінг даних із Google Analytics.

Запити – Google BigQuery створюються через стандартний діалект SQL, а результат повертається до JSON-формату. Стандартний розмір відповіді становить 128 Мб, але також він може бути і більшим (межа необмежена) при виставленні відповідних налаштувань.

Контроль доступу – користувачі сервісу можуть надавати стороннім особам публічний або обмежений доступ до своїх даних.

Машинне навчання – сервіс дає можливість створювати та запускати ML-моделі за допомогою SQL-запитів.

Інтеграції – сервіс можна використовувати як скрипт Google Apps Scripts або ж створений будь-якою іншою мовою, сумісною з REST API.

3 РОЗРОБКА СИСТЕМИ АВТОМАТИЗОВАНОГО ВИВАНТАЖЕННЯ ДАНИХ З ЛОКАЛЬНОЇ БАЗИ

3.1 Ініціалізація та підключення локальної бази

Для початку потрібно ініціалізувати основу нашого сервісу, а саме, локальну базу даних. Як було описано вище, ми використовуємо реляційну базу даних, та реляційну систему управління базами даних – mysql.

Для наших вимог підійде локальна root база, створюється з терміналу – командою mysql –u root –p, за замовчуванням пароль – root, але бажано його змінити, тому потрібно прописати ще декілька команд для цієї реалізації

update user set password=PASSWORD('******************) where User='root';

Після цього оновлюємо привілегії наступною командою

flush priviliges;

та, нарешті команда quit;, коли всі минулі запити були успішно виконані.

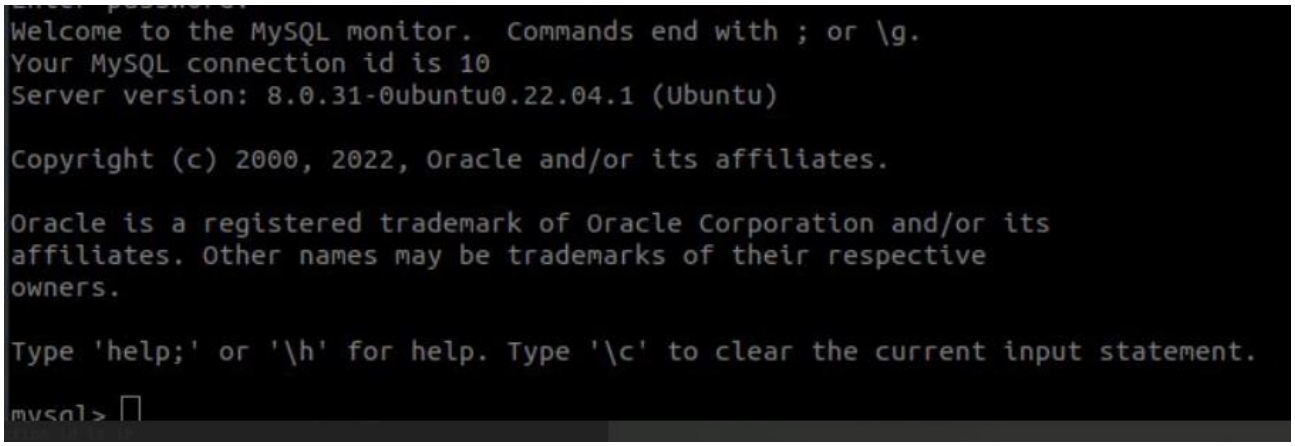


Рисунок 3.1 – термінал Mysql

Взаємодіяти з базою через термінал не дуже зручно та практично, тому підключаємось до нашої новоствореної бази даних з робочого середовища.

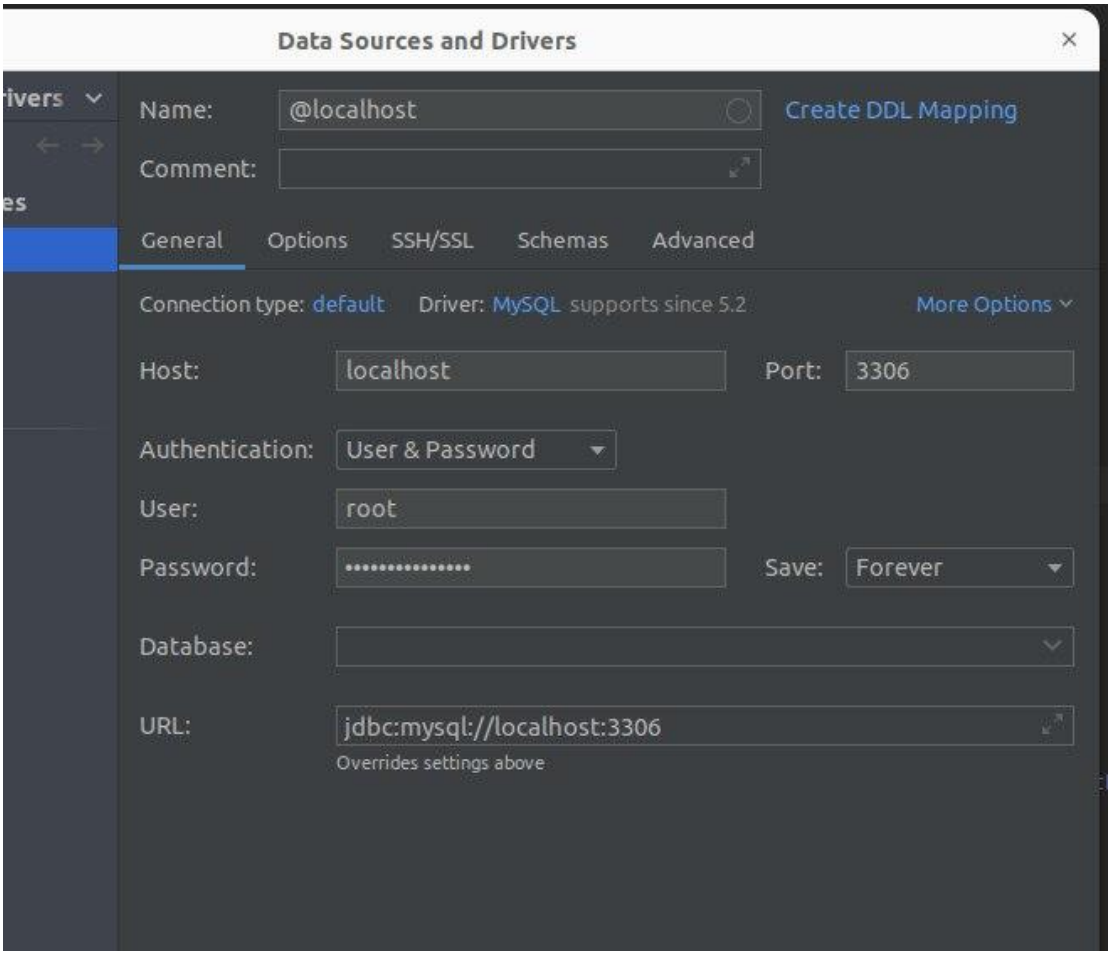


Рисунок 3.2 – підключення до бази в середовищі PhpStorm 5.6.

Після внесення всіх необхідних даних виконуємо тестове підключення до бази, де отримуємо інформацію про успішність такого підключення (рис. 3.3).

Рисунок 3.3 – результат підключення до бази через IDE UI

3.2 Розробка сервісу на платформі Node.js

Щоб розпочати створювати сервіс на node, потрібно для початку ініціалізувати директорію, це робиться з терміналу наступною командою:

cd Desktop && mkdir localDump && npm init

Саме npm init – y – ініціалізує проект Node, описує та ініціалізує саму базову інформацію про проект (рис. 3.4).

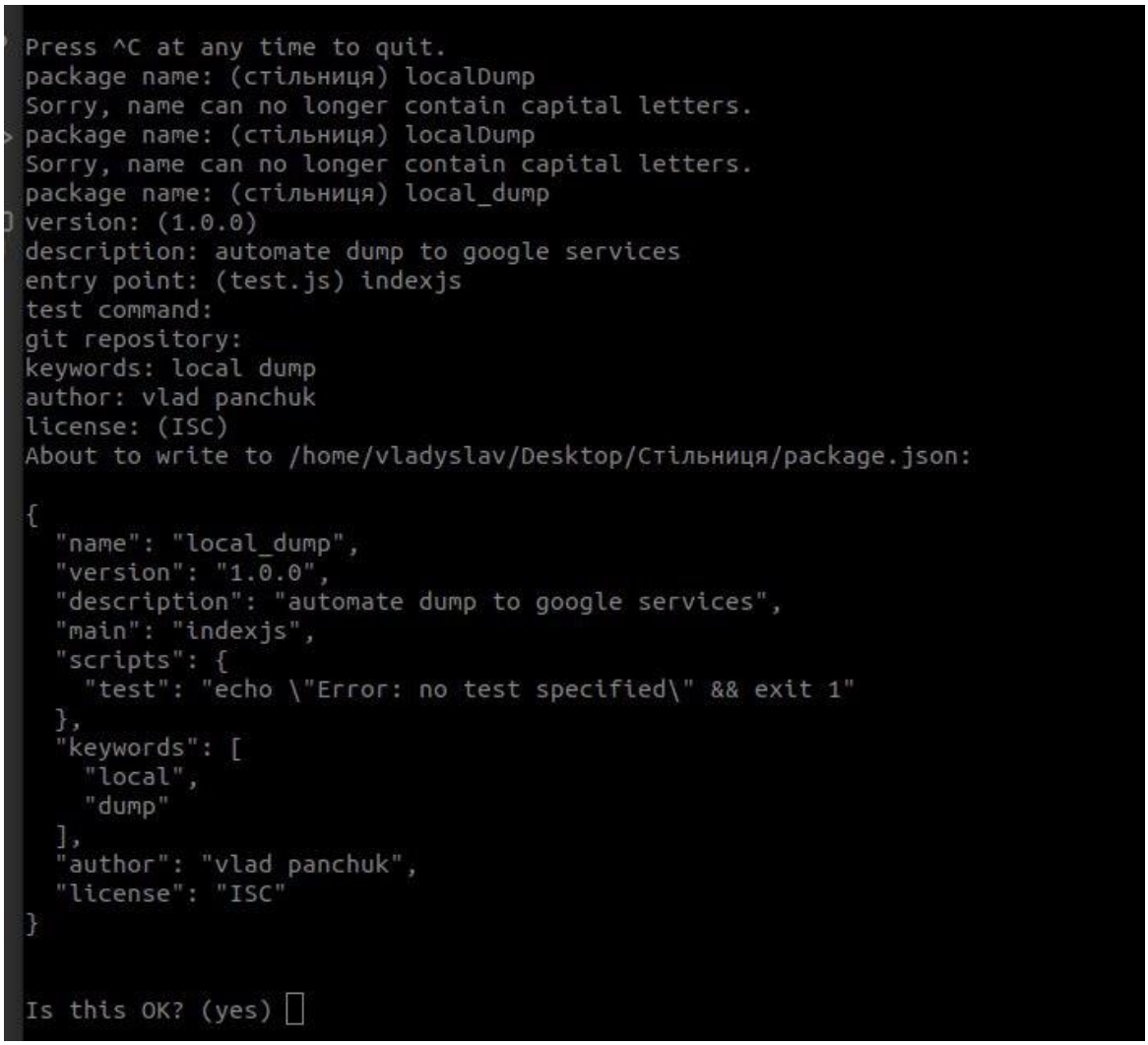


Рисунок 3.4 Результат роботи команди npm init

Після успішної операції в директорію проєкту добавляться файли package.json та package-lock.json – вони є серцем будь-якого проекту Node. Вони записують важливі метадані про проект, які потрібні перед публікацією в

репозиторій, а також визначають функціональні атрибути проекту, які npm використовує для встановлення залежностей, запуску сценаріїв і визначення точки входу в наш проєкт.

Також встановлюємо перші необхідні пакети для зручної та безпечної розробки

npm i nodemon mysql-query-promise dotenv

Nodemon — це інструмент, який допомагає розробляти програми на основі Node.js шляхом автоматичного перезапуску програми node, коли виявляються зміни файлів у каталозі.

Mysql-query-promise – Пакет призначений для зв’язку між проектом та самою базою даних

Dotenv – пакетний менеджер для зберігання змінних проекту в окремому файлі, який прописується в gitignore та недоступний нікому.

Тепер коли проєкт ініціалізовано, та встановлено базові пакети для початку роботи — в корневій директорії має бути створений файл .env (рис 3.5), в якому пропишемо потрібні змінні для підключення до бази.

```
apikey="AIzaS******************765WQQ"
DB_HOST="127.0.0.1"
DB_USER="root"
DB_PORT=3306
DB_PASSWORD="*****************"
DB_NAME="dom"
```

Рисунок 3.5 – Вигляд .env (dotenv) файлу

Зі зрозумілих причин apiKey та пароль до бази на скріншоті не вказані. Тепер є можливість використати ці дані конфігураційному файлі.

```
require('dotenv').config();
```

```
module.exports ={
   database: {
   'master': {
      host: process.env.DB_HOST,
      user: process.env.DB_USER,
      password: process.env.DB_PASSWORD,
      port: process.env.DB_PORT,
      database: process.env.DB_NAME,
      connectionLimit: 10
   },
  }
}
```

Тепер є можливість тестово зробити запит до локальної бази.

```
const query = require('mysql-query-promise');

function getActiveTables() {

  const qs = 'SHOW TABLES';

  return query(qs)
      .then((data) => {
         return data
      })
      .catch(err => {
console.log(`Occurred error in db connection ${err}`);
 });
}

module.exports = {
   getActiveTables,
};
```

Проєкт успішно підключений до локальної бази та може відтворювати повноцінний CRUD (Create, Read, Update, Delete).

3.3 Підключення сервісу до Google Cloud (BigQuery)

Для створення успішної взаємодії між локальним сервісом описаним вище, та середовищем Google Cloud, потрібно авторизувати локальне середовище.

Так як для розробки використовується операційна система Linux, офіційна документація[5] пропонує зробити це наступним чином:

- встановимо потрібні сертифікати **- sudo apt-get install apt-transport-https ca-certificates gnupg**
- Додамо URI розповсюдження CLI gcloud як джерело пакета. Ubuntu дистрибутив підтримує опцію підпису, тому виконуємо таку команду: **echo "deb [signed-by=/usr/share/keyrings/cloud.google.gpg] https://packages.cloud.google.com/apt cloud-sdk main" | sudo tee -a /etc/apt/sources.list.d/google-cloud-sdk.list**
- Додаємо Google Cloud Public Key curl **https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key --keyring /usr/share/keyrings/cloud.google.gpg add -**
- Обновлюємо систему та встановлюємо CLI **sudo apt-get update && sudo apt-get install google-cloud-cli**
- Ну і на сам кінець ініціалізуємо проєкт, вбиваючи наступну команду в його терміналі **gcloud init**

Після виконання вищеперечислених команд в терміналі дистрибутива Ubuntu (важливо нічого не наплутати, тому що під кожен дистрибутив існує свій набір репозиторіїв, ключів та команд) – можна констатувати той факт що на сервіс підключено до Google Cloud, та маємо хмогу користуватись його API.

- google-cloud-cli
- google-cloud-cli-anthos-auth
- google-cloud-cli-cloud-build-local
- google-cloud-cli-config-connector

3.4 Міграція даних локальної бази до сервісу BigQuery

Підключивши проект до Google Cloud через CLI – ми отримали змогу створювати таблиці. Сервіс проводитеме міграції з JSON файлів в яких додані наступні сутності: dataset, table, rows, seeds.

```
{
    "dataSetName": "dom",
    "tables": [
      {
        "name": "newbuildings",
        "schema": [
          {"name": "name",    "type": "STRING", "mode": "REQUIRED"},
          {"name": "estateType",     "type": "INTEGER", "mode":
"REQUIRED"},
          {"name": "isActive",  "type": "BOOLEAN"},
          {"name": "cityId", "type": "INTEGER"},
          {"name": "stateId", "type": "INTEGER", "mode": "REQUIRED"},
          {"name": "floorsCount", "type": "INTEGER"}
        ],
        "rows": [
          {"name": "super zhk", "estateType": 1, "isActive": true, "cityId": 1,
"stateId": 1, "floorsCount": 5 },
          {"name": "cool zhk", "estateType": 2, "isActive": true, "cityId": 9,
"stateId": 9, "floorsCount": 7 },
        ]
      },
      {
        "name": "buyers",
        "schema": [
          {"name": "name", "type": "STRING", "mode": "REQUIRED"},
          {"name": "surname",  "type": "STRING"},
          {"name": "age",  "type": "INTEGER", "mode": "REQUIRED"},
          {"name": "newbuildId", "type": "INTEGER"}
        ],
        "rows": [
          {"name": "vlad", "surname": "panchuk", "age": 22, "newbuildId": 10 },
          {"name": "katya", "surname": "liubinchak", "age": 22, "newbuildId": 10
},
```

```
            {"name": "kotlarevski", "surname": "just a kit", "age": 1, "newbuildId":
10 }
          ]
        }]}
```

Маючи файл міграції, ми можемо трансформувати його в запити до бази на Google Cloud, для цього поставимо ще один допоміжний пакет для node js – через npm, а саме[6]:

```
npm install @google-cloud/bigquery
```

Залежність автоматично записується до файлів package.json та package-lock.json

Імпортуємо сутність BigQuery з відповідного npm пакету, та відтворюємо з нього нову сутність.

```
// Imports the Google Cloud client library
  const { BigQuery } = require('@google-cloud/bigquery');
   const bigqueryClient = new BigQuery();
```

Змінна bigqueryClient – являється саме тією сутністю, яка відповідатиме за зв’язок між нашим сервісом, та хмарним сховищем даних. Далі спробуємо створити функцію для перевірки на наявність вже створеного датасету на сервері **BigQuery**

```
const bigqueryClient = new BigQuery();

/**
 *
 * @param {Int} id
 * @returns {Boolean}
 */
async function getDataset(id) {
   const [dataset] = await bigqueryClient.dataset(id).get();
   return !!dataset;
}
```

```
async function isDatasetAlreadyDeclared(dataSetName) {
      let isDatasetAlreadyDeclared = false;
      try {
         isDatasetAlreadyDeclared = await getDataset(dataSetName);
         return true;
      } catch (error) {
         return error.errors[0].reason !== 'notFound';
      }
   }
```

При розробці стараємось притримуватись мінімальних принципів та концепцій чистого коду, який можна писати на мові javascript, а саме оформлювати та описувати параметри функцій в jDoc[7], так як було вирішено не використовувати в проєкті Typescript, щоб лишній раз не ускладнювати логіку. Не потрібно також забувати про відслідковування помилок в коді, так як це явище глобальне та ніхто від цього не застрахований, не залежно скільки у Вас досвіду в програмуванні, Ви всеодно можете помилитись та допустити помилку, тому для кожної асинхронної функції, та не тільки, в проєкті прописаний блок try catch[8].

Коли протестовано з’єднання з сервером, ми можемо прочитати JSON файл, виконати його по-інструкційно: створити датасет (якщо його ще не існує) Добавити описані у файлі таблиці, колонки до таблиці, та відповідні дані. Тому створимо допоміжний файл helper.js, та винесемо до нього всі функціх, які напряму взаємодіють з Google Cloud[9], наприклад створення таблиці, або ж запис seed(початквих даних до таблиць).

```
   /
 *
 * @param {Int} datasetId
 * @param {Int} tableId
 * @param {Object} schema
 * @returns
 */
async function createTable(
   datasetId, // Existing dataset
```

```
  tableId, // Table to be created
  schema
 ) {

  const options = {
    schema: schema,
    location: 'US',
  };

  const [table] = await bigqueryClient
    .dataset(datasetId)
    .createTable(tableId, options);
  console.log(table ${table.id} has been created);
  return table;
 }

 /**
 *
 * @param {Int} datasetId
 * @param {Int} tableId
 * @param {Object} rows
 */
async function insertRowsAsStream(
    datasetId, // Existing dataset
    tableId, // Table to be created
    rows = []
   ) {

     // returning empty resolved if rows are empty
     if (!rows.length) return Promise.resolve();

     // Inserts the JSON objects into my_dataset:my_table.

     // Insert data into a table
     await bigqueryClient
       .dataset(datasetId)
       .table(tableId)
       .insert(rows);
```

```
}
```

Коли допоміжні функції створено, можна звести їх до одного цілого, та описати відповідну команду в файлі package.json. Реалізація міграції описується функцією init() у файлі migration.js, та виглядає наступним чином.

```
async function init() {

    const datasetJSON = fs.readFileSync(path.join('./assets', '/datasets', '/migration.json'));
    const datasetData = JSON.parse(datasetJSON);


    const { dataSetName, tables } = datasetData;

    const isDataSetCreated = await isDatasetAlreadyDeclared(dataSetName);
    if (isDataSetCreated) return;

    const dataSetId = await createDataset(dataSetName);

    const tablePromises = tables.map(({ name, schema }) => {
      return createTable(dataSetId, name, schema);
    });

    // creating tables
    const createdTables = await Promise.all(tablePromises);
    const tableIds = createdTables.map(({ id: tableId }) => tableId);


    // had to add small delay, because of issue when table has been added, but not available
    setTimeout(() => {
      try {
        const tableSeedPromises = tableIds.map((id, i) => insertRowsAsStream(dataSetId, id, tables[i].rows));
        // seeding the tables
        Promise.all(tableSeedPromises).then(() => console.log('seeding finished successfully'));
      } catch (error) {
        console.log('Occured an error while seeding ', error);
```

```
        }
      }, SEED_DELAY);

  }
```

Використовуючи можливості Node.js, а саме вміння створювати потоки для читання з файлів, ми можемо зчитати записи в json файли з допомогою модуля[10], який вміє працювати з файлами fs, та модуля path, який побудує посилання на файл правильно, незалежно від системи на якій запускається програма[16].

Функція міграції працює попрядку: створює датасет, після цього, відразу, створює таблиці, далі поля, та дані в ці поля, як можна побачити для створення даних добавлено невеличкий setTimeout, це тільки тому що, таблички створені у BigQuery не відразу стають активними, тому ми даємо їх 5 секунд, після чого починаємо записувати в них дані.

Також опишемо функцію для модифікації полів у Bigquery

```
const {BigQuery} = require('@google-cloud/bigquery');
const bigquery = new BigQuery();
async function addEmptyColumn() {
 // Adds an empty column to the schema.

 // const datasetId = 'my_dataset';
 // const tableId = 'my_table';
 const column = {name: 'size', type: 'STRING'};
 // Retrieve current table metadata
 const table = bigquery.dataset(datasetId).table(tableId);
 const [metadata] = await table.getMetadata();
 // Update table schema
 const schema = metadata.schema;
 const new_schema = schema;
 new_schema.fields.push(column);
 metadata.schema = new_schema;
 const [result] = await table.setMetadata(metadata);
 console.log(result.schema.fields);
}
```

Видалення dataset винесемо в інший файл, так як з видаленням ще будемо праюцвати та доповнювати (рис. 3.7).

```
// Import the Google Cloud client library
const {BigQuery} = require('@google-cloud/bigquery');
const bigquery = new BigQuery();
async function deleteDataset() {
 // Deletes a dataset named "my_dataset".
  Uncomment the following lines before running the sample.

 // const datasetId = 'my_dataset';
 // Create a reference to the existing dataset
 const dataset = bigquery.dataset(datasetId);
 // Delete the dataset and its contents
 await dataset.delete({force: true});
 console.log(`Dataset ${dataset.id} deleted.`);
}
```

Останнім штрихом додамо до package.json дкілька команд, для швидкого запуску з терміналу(рис. 3.6).

Рисунок 3.6 – вигляд package.json scripts

Прописавши команду можемо розпочинати виконувати функцію міграції (рис 3.7)

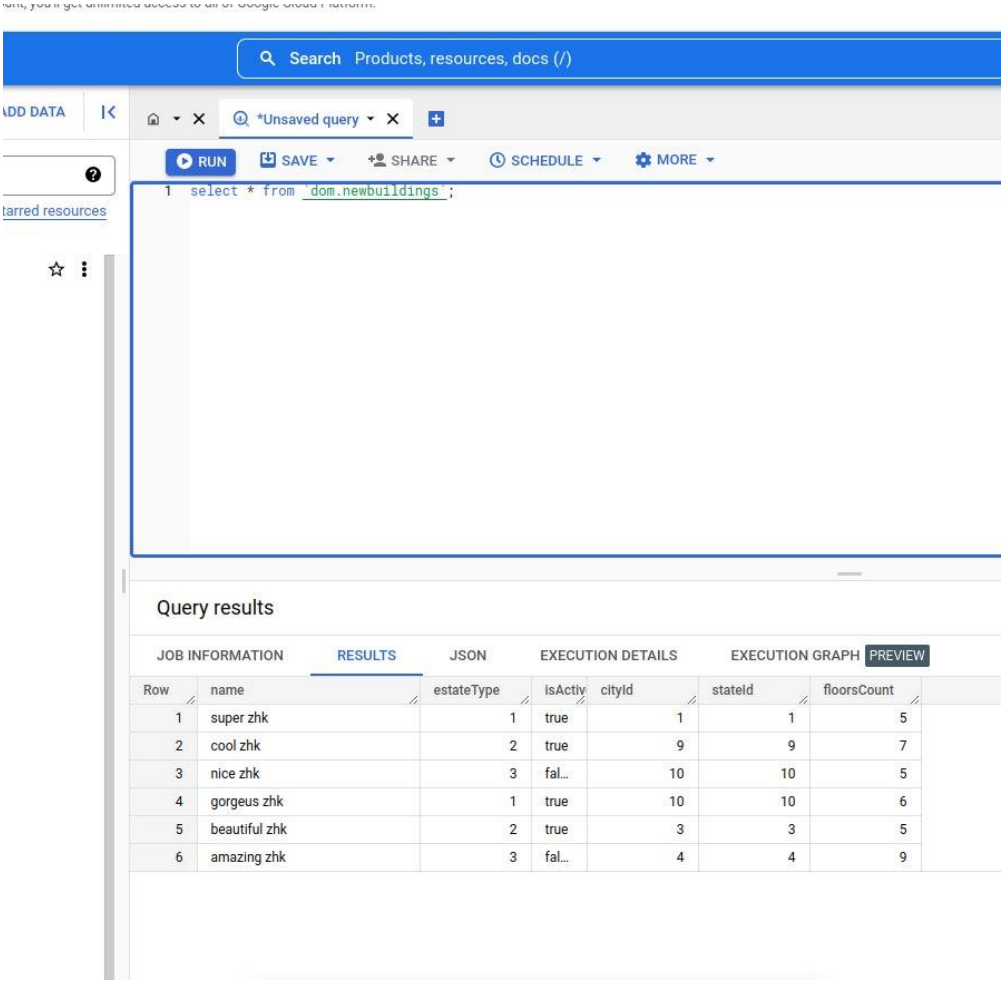


Рисунок 3.7 – Успішне виконання міграції

Орієнтуємось на вивід з консолі, так як попередньо добавили логи дл я відслідковування помилок (рис. 3.8).

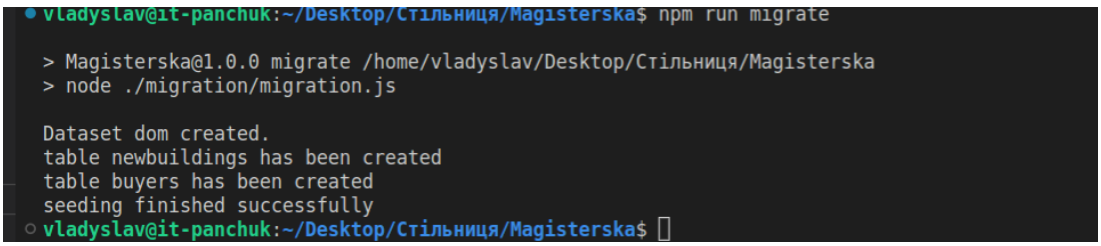


Рисунок 3.8 – Вивід консолі

Тепер коли налаштований зв’язок з BigQuery — можемо спроектувати автоматизацію накшталт реплікації master — slave.

Як випливає з назви, master-slave — це архітектура бази даних, яка розділена на головну базу даних і підлеглі бази даних. Підлегла база даних служить резервною копією для основної бази даних.

Головна база даних фактично є зберігачем ресурсів даних, а також місцем, де виконуються всі запити на запис. Операції читання розподіляються між кількома підлеглими базами даних відносно основної бази даних. Ця архітектура використовується для більшого підвищення надійності сайту.

Уявіть ситуацію, коли ваш сайт бомбардують кількома запитами на дані, що спричиняє сплеск веб-трафіку. Тепер, якщо є лише одна головна база даних, вона буде перевантажена, що зробить сайт повільнішим для всіх ваших користувачів.

Щоб масштабувати свою програму, вам потрібні два джерела даних: одне для обробки операцій запису, а інше для обробки операцій читання. І ось тут стає в нагоді архітектура “господар-підлеглий”. По суті, бази даних «головний-підлеглий» передбачають кешування даних з головної бази даних у підлеглі бази даних. Цей процес реплікації допомагає адміністраторам бази даних копіювати копії батьківської бази даних на кілька серверів одночасно. Кожного разу, коли оновлення надсилаються до головної бази даних, вони поширюються на підлеглі бази даних у контакті[17].

Тепер, якщо ви все ще скептично ставитеся до переваг бази даних головний-підлеглий, ось інший сценарій - уявіть, що ви підтримуєте великомасштабну програму, а ваш сервер не працює. Після кількох сеансів налагодження ви виявите, що це пов’язано із запитами на читання/запис до вашого сервера бази даних. У таких випадках рекомендується вибрати архітектуру бази даних головний-підлеглий. Розподіляючи навантаження між підлеглими вузлами, ця архітектура може допомогти вашій програмі масштабуватись відповідно до зростаючих користувачів.

Основним моментом, який нам потрібен — це файл з логами, який логує в себе всі зміни, які відбуваються з базою, на його основі ми зможемо ефективніше взаємодіяти з сервером, та економити ресурси. Також це називають двійковим

журналом. Двійковий журнал потрібно зчитувати кожного разу, коли дані копіюються. Кожен підлеглий пристрій додає навантаження на головний, оскільки двійковий журнал потрібно прочитати перед копіюванням даних на підлеглі вузли.

В директорію assets добавляємо файл db_logs.txt — куди будуть записуватись всі зміни в базі. Логуємо їх при кожній взаємодії з базою.

Далі пишемо функцію конвертер, яка має бути обладнана функціоналом, та вміти конвертувати логи з файлу в запити до Google Cloud BigQuery[24].

```
const fs = require('fs');
fs.readFile('./logs.txt', 'utf8', (err, data) => {
    if (err) { console.error(err); return;
    } console.log(data);
});
```

Експортування з локального сервера MySQL за допомогою mysqldump[18]

Для експортування локальної бази даних MySQL для імпорту в базу даних Cloud SQL, потрібно використовувати утиліту mysqldump із такими прапорцями:

--databases - використовувати параметр --databases, щоб вказати явний список баз даних для експорту, і цей список не повинен містити базу даних системи mysql.

--hex-blob Якщо база даних містить двійкові поля, потрібно використовувати цей прапорець, щоб переконатися, що двійкові поля імпортуються правильно.

--set-gtid-purged=ВИМК. Інформацію про GTID не можна включати у файл дампа SQL, а двійкове журналювання не повинно бути вимкнено файлом дампа SQL. (Не потрібно для MySQL 5.5 або зовнішньої реплікації.)[25]

--single-transaction Починає транзакцію перед запуском. Замість того, щоб блокувати всю базу даних, це дозволяє mysqldump читати базу даних у поточному стані, створюючи послідовний дамп даних[26].

З командного рядка потрібно запустити mysqldump:

```
mysqldump --databases DATABASE_NAME -h INSTANCE_IP -u USERNAME -p \
--hex-blob --single-transaction --set-gtid-purged=OFF \
--default-character-set=utf8mb4 > SQL_FILE.sql
```

Тепер зміни до бази записуються в окремий файл, та налоштоване їхх зчитування- потрібно автоматизувати процес зчитки та вигрузки даних на сервер[27].

Для Node.js в пакетному менеджері існує утиліта node-cron, ка допоможе автоматизувати локальний процес.

Cron — це допоміжна програма, яка дозволяє користувачам багаторазово вводити команди для планування завдань у певний час. Завдання, заплановані в cron, називаються завданнями cron[28]. Користувачі можуть визначити, яке завдання вони хочуть автоматизувати та коли його слід виконати.

Фоновий процес, який виконує неінтерактивні завдання. У Windows ви можете бути знайомі з фоновими процесами, такими як служби, які працюють подібно до демона cron.

Файл cron — це простий текстовий файл, який містить команди для періодичного запуску в певний час. Типовою системною таблицею cron або файлом конфігурації crontab є /etc/crontab[29].

Лише системні адміністратори можуть редагувати системний файл crontab. Однак Unix-подібні операційні системи підтримують кілька адміністраторів. Кожен може створити файл crontab і написати команди для виконання завдань у будь-який час (рис. 3.9).

За допомогою завдань cron користувачі можуть автоматизувати обслуговування системи, моніторинг дискового простору та планувати резервне копіювання. Через свою природу завдання cron чудово підходять для комп’ютерів, які працюють 24/7, наприклад для серверів[30].

Хоча завдання cron використовуються в основному системними адміністраторами, вони також можуть бути корисними для веб-розробників[33].

Наприклад, як адміністратор веб-сайту ви можете налаштувати одне завдання cron для автоматичного резервного копіювання вашого сайту щодня опівночі, інше для перевірки непрацюючих посилань щопонеділка опівночі, а третє для очищення кешу сайту щоп’ятниці опівдні[19].

Однак, як і будь-яка інша програма, cron має обмеження, які слід враховувати перед його використанням:

Найкоротший інтервал між завданнями становить 60 секунд. З cron ви не зможете повторювати завдання кожні 59 секунд або менше.

Централізовано на одному комп'ютері. Завдання Cron не можна розповсюджувати на кілька комп’ютерів у мережі. Отже, якщо комп’ютер, на якому запущено cron, виходить з ладу, заплановані завдання не виконуватимуться, а пропущені завдання можна буде запустити лише вручну.

Немає механізму автоматичної повторної спроби. Cron розроблений для запуску в строго визначений час. Якщо завдання не виконується, воно не буде запущено знову до наступного запланованого часу. Це робить cron непридатним для інкрементних завдань.

Встановлюємо node-cron до нашого ссервісу наступною командою з терміналу npm install --save node-cron.

З документції підключення до сервісу виглядає наступним чином

```
var cron = require('node-cron');

cron.schedule('* * * * *', () => {
  console.log('running a task every minute');
});
```

Налаштувати графік виконання роботи крону допомагають внутрішні інструменти самого пакету (бібліотеки).

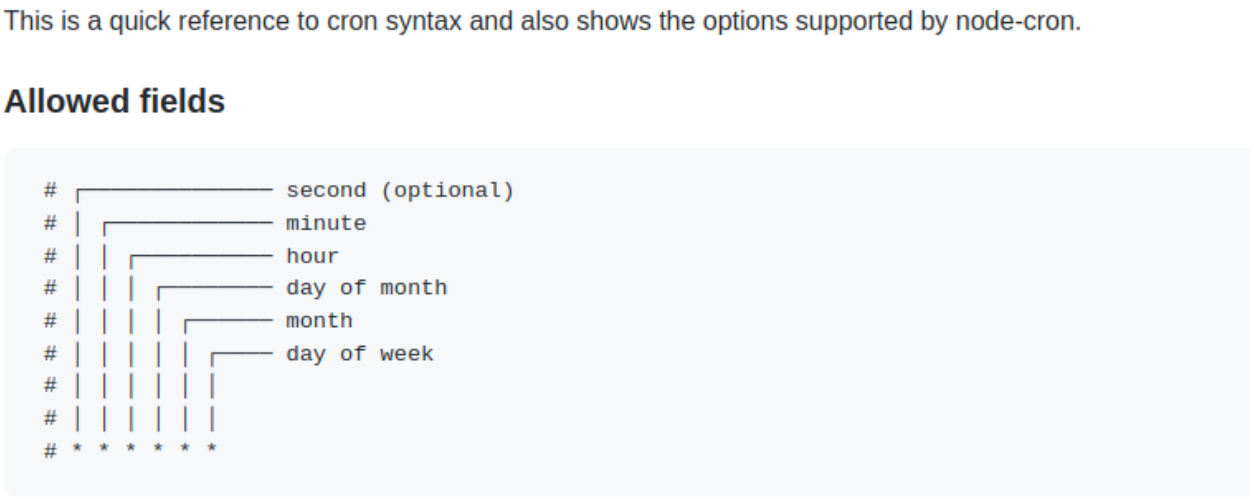


Рисунок 3.9 – Налаштування виконання cron-job

В основному файлі проєкту (index.js) — добавляємо бібліотеку cron-job

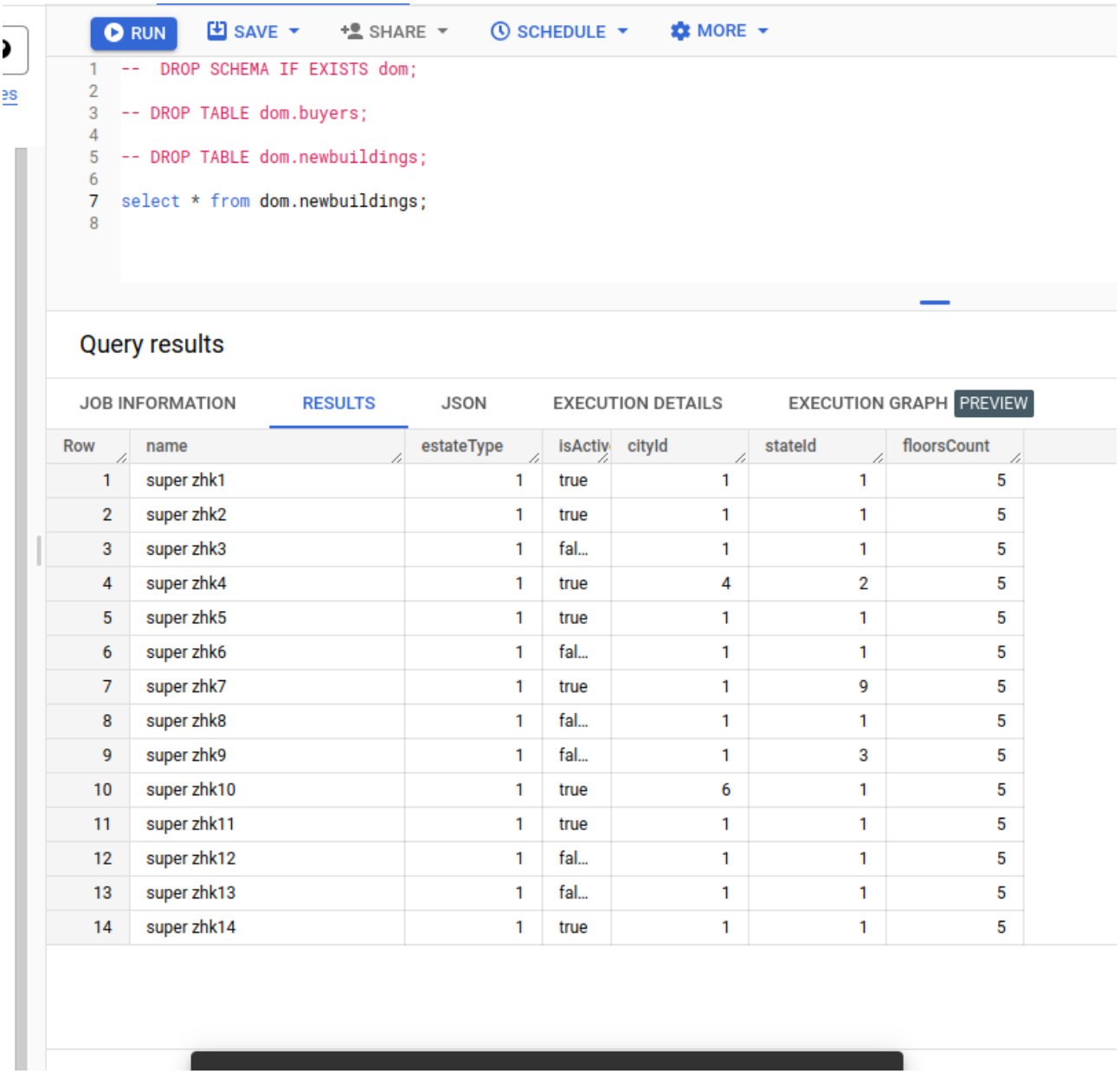
```
function runScheduleSync() {
    cron.schedule('* * * * *, async () => {
        try {
            await synchronizer();
            console.log('running a task every hour completed');
        } catch (error) {
            console.log(Error in schedule cron ---> ${error});
        }
    });
}
async function main() {
    if (Number(process.env.TYPE) === 1) {
        runScheduleSync()
    } else {
        const a = await synchronizer();
        console.log({ a });
    }

}
```

В залежності від типу виконання[20], який передається в команді яка стартує проєкт (start — описана у файлі package.json) — ми запускаємо процес вигрузки на Google Cloud, так, проєкт налаштований і на одноразову вигрузку даних(рис 3.10). Для тесту налаштуємо виконання на малий термін — кожну хвилину[32].

Спробуємо добавити в локальну базу декілька полів, та очікуємо синхронізацію з BigQuery.

```
INSERT INTO newbuildings
VALUES
("super zhk1", 1, true, 1, 1, 5),
("super zhk2", 1, true, 1, 1, 5),
("super zhk3", 1, false, 1, 1, 5),
("super zhk4", 1, true, 4, 2, 5),
("super zhk5", 1, true, 1, 1, 5),
("super zhk6", 1, false, 1, 1, 5),
("super zhk7", 1, true, 1, 9, 5),
("super zhk8", 1, false, 5, 1, 5),
("super zhk9", 1, false, 1, 3, 5),
("super zhk10", 1, true, 6, 1, 5),
("super zhk11", 1, false, 1, 1, 5),
("super zhk12", 1, false, 1, 1, 5),
("super zhk14", 1, true, 1, 1, 5);
```

Так як в налаштування крону ми проставили стандартне значення (всі зірочки), що означає кожну хвилину, це було зроблено для тесту, в робочому стані потрібність у вигрузці даних в залежності від масштабів проєкту та аналітики в ньому варіюється, але стандартом буде між годиною на трьома днями[33].



Row	name	estateType	isActiv	cityId	stateId	floorsCount
1	super zhk1	1	true	1	1	5
2	super zhk2	1	true	1	1	5
3	super zhk3	1	fal...	1	1	5
4	super zhk4	1	true	4	2	5
5	super zhk5	1	true	1	1	5
6	super zhk6	1	fal...	1	1	5
7	super zhk7	1	true	1	9	5
8	super zhk8	1	fal...	1	1	5
9	super zhk9	1	fal...	1	3	5
10	super zhk10	1	true	6	1	5
11	super zhk11	1	true	1	1	5
12	super zhk12	1	fal...	1	1	5
13	super zhk13	1	fal...	1	1	5
14	super zhk14	1	true	1	1	5

Рисунок 3.10 – Результат вигрузки даних

3.5 Обробка файлових даних для Google Storage Bucket

Також важливо звернути увагу нва хмарні сховища, - то му що вони являються невід’ємною частиною в роботі зі сервісами Google Cloud.

Хмарне сховище – це служба для зберігання ваших об’єктів у Google Cloud. Об'єкт - це незмінна частина даних, що складається з файлу будь-якого формату. Зберігаються об’єкти в контейнерах, які називаються “відрами”. Усі сегменти пов’язані з проектом, і ви можете згрупувати свої проекти за організацією. Кожен

проект, сегмент і об’єкт у Google Cloud є ресурсом у Google Cloud, як і такі речі, як екземпляри Compute Engine[21].

Можна створювати сегменти Cloud Storage, завантажувати об’єкти в сегменти та завантажувати об’єкти з сегментів. Ви також можете надати дозволи, щоб зробити ваші дані доступними для зазначених вами принципалів або — для певних випадків використання, наприклад розміщення веб-сайту — доступними для всіх у загальнодоступному Інтернеті[34].

Для роботи з цими сутностями ініціалізуємо новий контролер у проєкті з назвою bucketController.js

У створенні ініціалізуючої функції нам допоможе головна документація від Google.

```
const path = require('path');
const cwd = path.join(__dirname, '..');

function main(
  bucketName = 'my-bucket',
  fileName = 'test.txt',
  destFileName = path.join(cwd, 'downloaded.txt')
) {
  // [START storage_download_file]
  /**
   * TODO(developer): Uncomment the following lines before running the sample.
   */
  // The ID of your GCS bucket
  // const bucketName = 'your-unique-bucket-name';

  // The ID of your GCS file
  // const fileName = 'your-file-name';
```

```
// The path to which the file should be downloaded
// const destFileName = '/local/path/to/file.txt';

// Imports the Google Cloud client library
const {Storage} = require('@google-cloud/storage');

// Creates a client
const storage = new Storage();

async function downloadFile() {
  const options = {
    destination: destFileName,
  };

  // Downloads the file
  await storage.bucket(bucketName).file(fileName).download(options);

  console.log(
    gs://${bucketName}/${fileName} downloaded to ${destFileName}.
  );
}

downloadFile().catch(console.error);
```

Реалізація видалення відбувається наступним чином[22], а саме, створюється функція з назвою deleteFile, та отримує низку параметрів: ім’я файлу, ім’я google storage bucket, та додаткові опції, детальніше про які можна почитати в документації.

```
/ Imports the Google Cloud client library
```

```
const {Storage} = require('@google-cloud/storage');

// Creates a client
const storage = new Storage();

// Optional:
// Set a generation-match precondition to avoid potential race conditions
// and data corruptions. The request to upload is aborted if the object's
// generation number does not match your precondition. For a destination
// object that does not yet exist, set the ifGenerationMatch precondition to 0
// If the destination object already exists in your bucket, set instead a
// generation-match precondition using its generation number.
const deleteOptions = {
  ifGenerationMatch: generationMatchPrecondition,
};
async function deleteFile() {
  await storage.bucket(bucketName).file(fileName).delete(deleteOptions);

  console.log(gs://${bucketName}/${fileName} deleted);
}

deleteFile().catch(console.error);
```

Реалізувавши ці дві функції ми отримали можливість записувати та видаляти файли[23]. Файли з розширенням csv — стануть основними, якими буде оперувати bucketController, тому що це найпопулярніший варіант робити mysqldump (процес коли всі сутності бази компонуються у файл, у нашому випадку з розширенням sql). Саме сутність bucket буде давати нам можливість створювати міграцію даних бази між CloudSql та BigQuery.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку налаштування процесів автоматизованого вивантаження даних з локальної БД до сервісів Google Cloud. Особливістю програми є те, що дана технологія дозволяє вивантажувати дані до хмарних сервісів, не залежно від того чи проект задеплоєний, а також не витрачаються лишні кошти на деплой, та аналітику нового проєкту будується ще на стадії розробки.

Аналогічне ПЗ існує тільки як дочірні проєкти компаній. Ціна використання такого ПЗ близько 20000 грн.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 4.1.

Таблиця 4.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність продукту в реальних умовах

Продовження табл. 4.1

Ринкові переваги					
2	Багато аналогів на малому ринку	Ринкові п Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження табл. 4.1

Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так із комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію прод.	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних кошт.	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 4.2

Таблиця 4.2 – Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	3	4	3
Наявність аналогів на ринку	4	4	3
Цінова політика	4	4	4
Технічні та споживчі властивості виробу	4	3	3
Експлуатаційні витрати	3	3	4
Ринок збуту	4	4	3
Конкурентоспроможність	3	3	4
Фахівці з технічної і комерційної реалізації	3	4	3
Фінансування	4	3	4
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	3	3	3
Супровідна документація	3	4	3
Сума	41	42	40
Середньоарифметична сума балів	(41+42+40) / 3 = 41		

За даними таблиці 4.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 4.3.

Таблиця 4.3 - Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок того, що програмний продукт відрізняється від існуючих тим, що дана технологія дозволяє вивантажувати дані до хмарних сервісів, не залежно від того чи проект задеплоєний, а також не витрачаються лишні кошти на деплой, та аналітику нового проєкту будується ще на стадії розробки.

4.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

4.2.1 Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{М}{Т_р} \cdot t, \quad (4.1)$$

де М – місячний посадовий оклад конкретного розробника (дослідника), грн.;

$Т_р$ – число робочих днів в місяці, 23 днів;

t – число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 4.1.

Таблиця 4.1 – Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	25000	1086,96	46	50000,000
Програміст	22000	956,52	46	44000,000
Всього				94000,00

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

4.2.2 Додаткова заробітна плата розробників, які приймали участь в розробці обладнання.

Додаткова заробітна плата прийнято розраховувати як 10 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 10\ \% / 100\ \% \quad (4.2)$$

$З_д = (94000{,}00 \cdot 10\ \% / 100\ \%) = 9400{,}00$ (грн.)

4.2.3 Нарахування на заробітну плату розробників.

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \quad (4.3)$$

$Н_з = (94000{,}00 + 9400{,}00) \cdot 22\ \% / 100\ \% = 22748{,}00$ (грн.)

4.2.4. Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

4.2.5 Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді амортизація обладнання, що використовувалась для

розробки розраховується за формулою:

$$А = \frac{Ц}{Т_в} \cdot \frac{t_{вик}}{12} \text{ [грн.]}. \quad (4.4)$$

де Ц – балансова вартість обладнання, грн.;

Т – термін корисного використання обладнання згідно податкового законодавства, років

$t_{вик}$ – термін використання під час розробки, місяців

Розрахуємо, для прикладу, амортизаційні витрати на комп'ютер балансова вартість якого становить 21000 грн., термін його корисного використання згідно податкового законодавства – 2 роки, а термін його фактичного використання – 2,00 міс.

$$А_{обл} = \frac{21000}{2} \times \frac{2}{12} = 1750 \text{грн.}$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 4.2. Для розрахунку амортизації нематеріальних ресурсів використовується формула:

$$А_{н.р.} = \frac{Ц_{Н.р.} * Н_а * t_{вик}}{12} А_{н.р.} = \frac{Ц_{н.р.} * Н_а * t_{вик.}}{12}$$

(4.5)

Але, так як вартість ліцензійної ОС та спеціалізованих ліцензійних нематеріальних ресурсів менше 20000 грн, то даний нематеріальний актив не амортизується, а його вартість включається у вартість розробки повністю, $В_{нем.ак.}$ = 2920 грн.

Таблиця 4.2 – Амортизаційні відрахування матеріальних і нематеріальних ресурсів для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн.
Комп'ютер та комп'ютерна переферія	21000	2	2,00	1750,000
Офісне обладнення	24000	4	2,00	1000,000
Приміщення	1090000	20	2,00	9083,333
Всього				11833,33

5.2.6 Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$å = å \cdot ï \cdot Ô \cdot æ, \qquad (4.6)$$

де В – вартість 1 кВт-години електроенергії для 1 класу підприємства, В = 6,2 грн./кВт;

П – встановлена потужність обладнання, кВт. П = 0,4 кВт;

Ф – фактична кількість годин роботи обладнання, годин.

$К_п$ – коефіцієнт використання потужності, $К_п$ = 0,9.

$В_е = 0{,}9 \cdot 0{,}4 \cdot 8 \cdot 46 \cdot 6{,}2$ = 821,376 (грн.)

5.2.7 Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.7)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

$І_в$ = 94000,00 * 80% / 100% = 75200 (грн.)

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$Н_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (4.8)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$Н_{нзв}$ = 94000,00 * 130 % / 100 % = 122200 (грн.)

5.2.9 Витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$B_{заг}$ = 94000,00+9400,00+22748,00+11833,33+2920+821,38+75200+
+122200 = 339122,71 грн.

5.2.11 Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються *ЗВ*, визначається за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} \text{ (грн)}, \quad (5.9)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

ЗВ = 339122,71 / 0,5 = 678245 грн.

4.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

а) вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

б) зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

в) кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб'єктів (зацікавлених осіб) цього попиту;

г) визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов'язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

- абсолютного економічного ефекту (чистого дисконтованого доходу);

- внутрішньої економічної дохідності (внутрішньої норми дохідності);
- терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об'єднати, враховуючи визначені ситуації з відповідними умовами.

4.3.1 Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = \qquad (4.10)$$

де $\pm\Delta Ц_o$ – зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_o = Ц_б \pm \Delta Ц_o$;

$Цб$ – вартість програмного продукту у році до впровадження результатів розробки;

ΔN – збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт $\lambda = 0{,}8333$.

p – коефіцієнт, який враховує рентабельність продукту;

ϑ – ставка податку на прибуток, у 2022 році ϑ =18%.

Припустимо, що при прогнозованій ціні 3500 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на *500* грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року – на 4000 шт., протягом другого року – на 3000 шт., протягом третього року на 2000 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

*$\Delta П_1$ = (0*500 + (*3500 + *500)*4000)* 0,8333* 0,3) * (1 - 0,18) = 2869999,885 грн.*

*$\Delta П_2$ = (0*500 + (*3500 + *500)*(4000+3000)* 0,8333* 0,3) * (1 - 0,18) = 5739999,770 грн.*

*$\Delta П_3$ = (0*500 + (*3500 + *500)*(4000+3000+2000)* 0,8333* 0,3) * (1 - 0,18) = 7379999,705 грн.*

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 15989999,36 грн.

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності.

Розраховуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_1^T \frac{\Delta П_i}{}, \qquad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т – період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0,05...0,15$;

t – період часу (в роках).

Збільшення прибутку ми отримаємо починаючи з першого року:

ПП= (2869999,885/(1+0,1)[1])+(5739999,770/(1+0,1)[2])+(7379999,705/(1+0,1)[3]) = 2609090,80 + 4743801,463 + 5544703,009 = 12897595,28 грн.

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \quad (4.12)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

PV = 2 * 678245 = 1356490,84 *грн.*

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід *(NPV, Net Present Value)* для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV, \quad (4.13)$$

$E_{абс}$ = 12897595,28 – 1356490,84 = 11541104,44 грн.

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності *(IRR, Internal Rate of Return)* вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

(4.14)

$T_ж$ – життєвий цикл наукової розробки, роки.

$E_в$ = з (1 + 11541104,44/1356490,84 – 1 = 1,119

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

(4.15)

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d = (0,09…0,14);

f−показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\min} = 0,14 + 0,05 = 0,19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в}, \tag{4.16}$$

$T_{ок}$ = 1 / 1,119 = 0,89 р.

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,89 роки, то фінансування даної наукової розробки є доцільним.

Висновки до розділу: економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 678245 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 0,89 роки.

ВИСНОВКИ

В ході виконання магістерської кваліфікаційної роботи були проведені дослідження та аналіз хмарних сервісів Google Cloud. Метою роботи було розробити автоматизоване вивантаження даних на сервіси з локальної бази. Було проведено аналіз існуючих сервісів, та прийнято рішення працювати з базами реляційного типу, так як локальна база також типу SQl. До того ж було проведено досліження функціональності вибраних хмарних сервісів, а саме їх API та CLI, які сервіси пропонують для зручної роботи з ними. Також проведено дослідження та тестування отриманих результатів.

В першому розділі було проведено аналіз актуальності хмарних систем в цілому, їх впли та важливість у сьогоденні. Також був проведений огляд та аналіз хмарних сервісів, які предоставляє Google Cloud. Для розробки були вибрані BigQuery, CloudSql та Google Cloud Storage (Bucket).

В другому розділі було проведено дослідження та вибір необхідних інструментів для розробки сервісу, основою для реалізації було вибрано платформу Node.js, як посередника між локально базою, та Google Cloud сервісами. Також основою проєкту виступатиме реляційна база типу SQL.

Було проведено аналіз хмарних сервісів Google Cloud, так як вони являються одними із самий популярних та часто використовуються компаніями в своїх проектах.

В третьому розділі описано процес розробки сервісу: налаштування локальної бази та заповнення її тестовими даними, якими надалі буде оперувати сервіс, ініціалізація проекту node.js та підключення доо нього локальної бази та робота з API хмарних сервісів. В розділі наведені аргументації вибору тої чи іншої технології та описані ролі кожного компоненту програми, наведеііі відповідні скріншоти, які ілюструються процес розробки програми.

В четвертому розділі описана економічна частина проекту, з всіма відповідними підрахунками.

ПЕРЕЛІК ДЖЕРЕЛ

1. Офіційний сайт компанії Cloud Client Libraries [Електронний ресурс] Режим доступу: https://cloud.google.com/nodejs/docs/reference
2. Волокита А., Мухін В., Стешин В. Специфіка інформаційних систем на основі технології cloud computing. [Електронний ресурс] Режим доступу: http://archive.nbuv.gov.ua/portal/natural/vcndtu/2011_53/29.html
3. Хмарні технології переваги та недоліки [Електронний ресурс]. Режим доступу: https://valtek.com.ua/ua/system-integration/it-infrastructure/clouds/cloud-technologies
4. Документація Google Cloud по авторизації сервісу: [Електронний ресурс]. Режим доступу: https://cloud.google.com/sdk/docs/install
5. Google Cloud BigQuery documentation [Електронний ресурс]. Режим доступу: https://cloud.google.com/nodejs/docs/reference/bigquery/latest
6. Google Cloud Node.js API[Електронний ресурс]. Режим доступу: https://cloud.google.com/nodejs/docs/reference
7. Основи опису програмного забезпечення на мові javascript [Електронний ресурс]. Режим доступу: https://jsdoc.app/
8. Node.js Error Handling Tips and Tricks [Електронний ресурс]. Режим доступу: https://blog.appsignal.com/2022/11/16/nodejs-error-handling-tips-and-tricks.html
9. Cloud Locations. [Електронний ресурс]. Режим доступу: https://cloud.google.com/about/locations#network
10. Alibaba Cloud's Global Infrastructure. [Електронний ресурс]. Режим доступу : https://www.alibabacloud.com/global-locations#J_7563247410/
11. Google Cloud vs AWS: Difference Between AWS and GCP. [Електронний ресурс]. Режим доступу: https://www.guru99.com/google-cloud-vs-aws.html/

12. Google Cloud Big Data: Building Your Big Data Architecture on GCP. [Електронний ресурс]. Режим доступу : https://cloud.netapp.com/blog/gcp-cvo-blg-google-cloud-big-data-build-abig-data-architecture-on-gcp
13. Top programming languages: Most popular and fastest growing choices for developers. [Електронний ресурс]. Режим доступу : https://www.zdnet.com/article/top-programminglanguages-most-popular-and-fastest-growing-choices-for-developers/
14. Вакалюк Т.А. Можливості використання хмарних технологій в освіті / Т.А. Вакалюк // Актуальні питання сучасної педагогіки. Матеріали міжнародної науково-практичної конференції (м. Острог, 1-2 листопада 2019 року). – Херсон: Видавничий дім "Гельветика", 2019. – С. 97–99
15. Биков В.Ю. Хмарна комп'ютерно-технологічна платформа відкритої освіти та відповідний розвиток організаційно-технологічної будови іт-підрозділів навчальних закладів / В.Ю. Биков // Теорія і практика управління соціальними системами. – 2018. – № 1. – С. 81-98.
16. Нечаєва А.В., Рибакова Л.В. Шляхи використання хмарних технологій у самостійній роботі студентів при вивченні дисципліни ―алгоритмічні мови і програмування‖ [Електронний ресурс]. Режим доступу : http://www.kntu.kr.ua/doc/zbirnyki/2015/3.pd/
17. Хмарні технології [Електронний ресурс]. Режим доступу: http://wiki.kubg.edu.ua/
18. Лекція хмарні технології [Електронний ресурс]. Режим доступу: https://sites.google.com /site/navcalnapraktikakitvoin/lekciie/lekcia-hmarni-tehnologiie
19. Dr. Christian Wurll. ROBOGISTICS [Електронний ресурс] / Dr. Christian Wurll – Режим доступу до ресурсу: https://www.ostwestfalen.ihk.de/fileadmin/user_upload/20151022-PR-WuC-E-1- Robogistics_-_Robotics_in_Logistics.public_Vortrag_Dr._Wurll.pdf.

20. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A.Wallach, et al. Bigtable: A Distributed Storage System for Structured Data. Proceedings of OSDI, 2016.
21. Петров А. Big Data от А до Я. Hadoop, Hbase. [Електронний ресурс]. Режим доступу: https://habr.com/company/dca/blog/267361/
22. Петухов Д. Spanner. NewSQL хранилище от Google. [Електронний ресурс]. Режим доступу: https://www.codeinstinct.pro/2013/12/spanner.html
23. ERwin [Електронний ресурс] – Режим доступу до ресурсу: https://old.dataone.org/software-tools/erwin
24. What you'll love about SQL Server 2019 [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/en-us/sql-server/sql-server-2019
25. SQL Server Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.sqlservertutorial.net/
26. Connect Microsoft SQL Server to Metabase [Електронний ресурс] – Режим доступу до ресурсу: https://www.metabase.com/data_sources/microsoft-sql-server
27. The world’s leading analytics platform [Електронний ресурс] – Режим доступу до ресурсу: https://www.tableau.com/
28. Визуализация и аналитика данных Tableau [Електронний ресурс] – Режим доступу до ресурсу: https://corewin.ua/ru/analytics-and-data/tableau/
29. Описание системы Tableau Desktop [Електронний ресурс] – Режим доступу до ресурсу: https://soware.ru/products/tableau-desktop/
30. Аббакумов А. Базы данных (MS SQL Server). Учеб. пособие. Саранск: Изд-во СВМО, 2015. 66 с.
31. Берегер А. Microsoft SQL Server 2005Analysis Services. OLAP и многомерный анализ данных. – Спб.: ВХБ – Петербург, 2007. – 928с.

32. S. P. T. Krishnan, Ugia Gonzalez, The Google Cloud Platform Difference, Building Your Next Big Thing with Google Cloud Platform, Apress, Berkeley, CA, 2015, pp 3-12
33. Панчук В.В., Богач І.В., Гуральник Ф.Б. Процеси та види тестування програмного забезпечення. Науково-технічна конференція факультету комп'ютерних систем і автоматики Вінницького національного технічного університету : матеріали L науково-технічної конференції ВНТУ, Вінниця, 2021. - URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa- 2021/paper/view/12826/

ДОДАТКИ

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: « Налаштування процесів автоматизованого вивантаження даних з локальної бази даних до сервісів Google Cloud»

Тип роботи: Магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ КСУ, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 71,1% Схожість 28,9%

Аналіз звіту подібності (відмітити потрібне)

☐ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на розгляд експертної комісії кафедри.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Галушак А.В.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unichek щодо роботи.

Автор роботи _____ Панчук В.В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Юхимчук М.С.
(підпис) (прізвище, ініціали)

ДОДАТОК А
(обов'язковий)

ЗАТВЕРДЖЕНО
Зав. кафедри КСУ ВНТУ,
д.т.н., професор
_____ В'ячеслав КОВТУН

"03" жовтня 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

Налаштування процесів автоматизованого вивантаження даних з локальної бази даних до сервісів Google Cloud

08-33.МКР.009.00.000 ТЗ

Студент групи 2АКІТ-21м

Підпис Владислав ПАНЧУК
Ім'я ПРІЗВИЩЕ

Керівник Марія ЮХИМЧУК

Підпис _____
Ім'я ПРІЗВИЩЕ

Вінниця 2022

1. Назва та галузь застосування

1.1. Назва Налаштування процесів автоматизованого вивантаження даних з локальної бази даних до сервісів Google Cloud

1.2. Галузь застосування – хмарні сервіси Google Cloud.

2. Підстава для проведення розробки.

Тема магістерської кваліфікаційної роботи затверджена наказом по ВНТУ від “14” вересня 2022 року №203

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є налаштування процесу автоматизованої вигрузки даних з локальної бази на хмарні сервіси

4. Джерела розробки.

Магістерська кваліфікаційна робота виконується вперше. В ході проведення розробки повинні використовуватись такі документи:

1. Офіційний сайт компанії Google Cloud (Електронний ресурс) Режим доступу: URL: https://console.cloud.google.com/

2. Лекція хмарні технології [Електронний ресурс]. Режим доступу: https://sites.google.com /site/navcalnapraktikakitvoin/lekciie/lekcia-hmarni-tehnologiie

3. Google Cloud Node.js API[Електронний ресурс]. Режим доступу: https://jsdoc.app

4. Node.js Error Handling Tips and Tricks[Електронний ресурс]. Режим доступу:

5. Вимоги до розробки.

5.1. Перелік головних функцій:

- Можливість міграції даних через JSON файл;
- Можливість налаштування роботи крону для можливості гнучного налаштування оновлень бази;
- використані різнітипи сховищ;

6. Стадії та етапи розробки.

6.1 Пояснювальна записка:

1. Аналіз методів, принципів, підходів і засобів реалізації задачі автоматизації процесами в об’єкті управління відповідно до теми кваліфікаційної роботи. Постановка задач дослідження «03» 09 2022 р.
2. Удосконалення технології прийняття рішень при автоматизації об’єкту управління «15 » 09 2022 р.
3. Визначення технічних характеристик системи «09» 10 2022 р.
4. Розробка програмного забезпечення системи «24» 10 2022 р.

6.2 Графічні матеріали:

1. Розробка сервісу для автоматизованої вигрузки даних з локальної бази даних на сервіси Google Cloud «04» 11 2022 р.

7. Порядок контролю і приймання.

7.1. Хід виконання роботи контролюється керівником роботи. Рубіжний контроль провести до «28» 11 2022 р.

7.2. Атестація МКР здійснюється на попередньому захисті. Попередній захист магістерської кваліфікаційної роботи провести до «16» 12 2022 р.

7.3. Підсумкове рішення щодо оцінки якості виконання роботи приймається на засіданні ЕК. Захист магістерської кваліфікаційної роботи провести до «23» 12 2022 р.

ДОДАТОК Б
(довідковий)

Вигляд програми

```
const cron = require('node-cron');
const {
    getActiveTables,
} = require('./controllers/mysqlController');

require('dotenv').config();

async function synchronizer() {
    return await getActiveTables();
}



function runScheduleSync() {
    cron.schedule('*/0.5 * * * *', async () => {
        try {
            await synchronizer();
            console.log('running a task every hour completed');
        } catch (error) {
            console.log(Error in schedule cron ---> ${error});
        }
    });
}

async function main() {

    if (Number(process.env.TYPE) === 1) {
```

```
        runScheduleSync()
    } else {
        const a = await synchronizer();
        console.log({ a });
    }

}

main();

const fs = require('fs');
const path = require('path');
// const { PerformanceObserver } = require('perf_hooks');
// Imports the Google Cloud client library
const { BigQuery } = require('@google-cloud/bigquery');
const { getDataset, createDataset, createTable, insertRowsAsStream } =
        require('./helper');

const bigqueryClient = new BigQuery();


const SEED_DELAY = 5000;

async function isDatasetAlreadyDeclared(dataSetName) {
    let isDatasetAlreadyDeclared = false;
    try {
        isDatasetAlreadyDeclared = await getDataset(dataSetName);
        return true;
    } catch (error) {
```

```
        return error.errors[0].reason !== 'notFound';
    }
}

async function init() {


    const datasetJSON = fs.readFileSync(path.join('./assets', '/datasets',
        '/migration.json'));
    const datasetData = JSON.parse(datasetJSON);


    const { dataSetName, tables } = datasetData;

    const isDataSetCreated = await isDatasetAlreadyDeclared(dataSetName);
    if (isDataSetCreated) return;

    const dataSetId = await createDataset(dataSetName);

    const tablePromises = tables.map(({ name, schema }) => {
        return createTable(dataSetId, name, schema);
    });

    // creating tables
    const createdTables = await Promise.all(tablePromises);
    const tableIds = createdTables.map(({ id: tableId }) => tableId);
```

```
    // had to add small delay, because of issue when table has been added, but not
       available
    setTimeout(() => {
       try {
          const tableSeedPromises = tableIds.map((id, i) =>
       insertRowsAsStream(dataSetId, id, tables[i].rows));
          // seeding the tables
          Promise.all(tableSeedPromises).then(() => console.log('seeding finished
       successfully'));
       } catch (error) {
          console.log('Occured an error while seeding ', error);
       }
    }, SEED_DELAY);

}

init();

Vlad Panchuk, [13.12.2022 18:25]
require('dotenv').config();

module.exports ={
    database: {
    'master': {
       host: process.env.DB_HOST,
       user: process.env.DB_USER,
       password: process.env.DB_PASSWORD,
       port: process.env.DB_PORT,
       database: process.env.DB_NAME,
```

```
        connectionLimit: 10
    },
   }
}
```

Vlad Panchuk, [13.12.2022 18:25]

```
{
  "name": "Magisterska",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "nodemon index.js",
    "start-cron-sync": "TYPE=1 npm start",
    "migrate": "node ./migration/migration.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@google-cloud/bigquery": "^6.0.3",
    "dotenv": "^16.0.3",
    "mysql-query-promise": "^1.0.1",
    "mysql2": "^2.3.3",
    "node-cron": "^3.0.2",
    "nodemon": "^2.0.20"
  }
}
```

Vlad Panchuk, [13.12.2022 18:25]

```
const query = require('mysql-query-promise');

function getActiveTables() {

  const qs = 'SHOW TABLES';

  return query(qs)
      .then((data) => {
         return data
      })
      .catch(err => console.log(err));
}

module.exports = {
   getActiveTables,
};

// Imports the Google Cloud client library
const { BigQuery } = require('@google-cloud/bigquery');

const bigqueryClient = new BigQuery();

/
 *
 * @param {Int} id
 * @returns
 */
async function getDataset(id) {
```

```
    const [dataset] = await bigqueryClient.dataset(id).get();
    return dataset;
}



/
 *
 * @param {Int} dataSetId
 * @returns
 */
async function createDataset(dataSetId) {
    // Create the dataset
    const [dataset] = await bigqueryClient.createDataset(dataSetId);
    console.log(Dataset ${dataset.id} created.);
    return dataset.id;
}

/
 *
 * @param {Int} datasetId
 * @param {Int} tableId
 * @param {Object} schema
 * @returns
 */
async function createTable(
    datasetId, // Existing dataset
    tableId, // Table to be created
    schema
  ) {
```

```
    const options = {
        schema: schema,
        location: 'US',
    };

    const [table] = await bigqueryClient
        .dataset(datasetId)
        .createTable(tableId, options);
    console.log(`table ${table.id} has been created`);
    return table;
  }

/
 *
 * @param {Int} datasetId
 * @param {Int} tableId
 * @param {Object} rows
 */
async function insertRowsAsStream(
  datasetId, // Existing dataset
  tableId, // Table to be created
  rows = []
) {

    // returning empty resolved if rows are empty
    if (!rows.length) return Promise.resolve();

    // Inserts the JSON objects into my_dataset:my_table.
```

```
    // Insert data into a table
    await bigqueryClient
      .dataset(datasetId)
      .table(tableId)
      .insert(rows);

}

module.exports = {
    getDataset,
    createDataset,
    createTable,
    insertRowsAsStream,
}
```

Vlad Panchuk, [13.12.2022 18:25]

```
{
    "dataSetName": "dom",
    "tables": [
        {
            "name": "newbuildings",
            "schema": [
                {"name": "name",    "type": "STRING", "mode": "REQUIRED"},
                {"name": "estateType",     "type": "INTEGER", "mode":
"REQUIRED"},
                {"name": "isActive",  "type": "BOOLEAN"},
                {"name": "cityId", "type": "INTEGER"},
                {"name": "stateId", "type": "INTEGER", "mode": "REQUIRED"},
```

```
            {"name": "floorsCount", "type": "INTEGER"}
        ],
        "rows": [
            {"name": "super zhk1", "estateType": 1, "isActive": true, "cityId": 1,
"stateId": 1, "floorsCount": 5 },
            {"name": "super zhk2", "estateType": 1, "isActive": true, "cityId": 1,
"stateId": 1, "floorsCount": 5 },
            {"name": "super zhk3", "estateType": 1, "isActive": false, "cityId": 1,
"stateId": 1, "floorsCount": 5 },
            {"name": "super zhk4", "estateType": 1, "isActive": true, "cityId": 4,
"stateId": 2, "floorsCount": 5 },
            {"name": "super zhk5", "estateType": 1, "isActive": true, "cityId": 1,
"stateId": 1, "floorsCount": 5 },
            {"name": "super zhk6", "estateType": 1, "isActive": false, "cityId": 1,
"stateId": 1, "floorsCount": 5 },
            {"name": "super zhk7", "estateType": 1, "isActive": true, "cityId": 1,
"stateId": 9, "floorsCount": 5 },
            {"name": "super zhk8", "estateType": 1, "isActive": false, "cityId": 1,
"stateId": 1, "floorsCount": 5 },
            {"name": "super zhk9", "estateType": 1, "isActive": false, "cityId": 1,
"stateId": 3, "floorsCount": 5 },
            {"name": "super zhk10", "estateType": 1, "isActive": true, "cityId": 6,
"stateId": 1, "floorsCount": 5 },
            {"name": "super zhk11", "estateType": 1, "isActive": true, "cityId": 1,
"stateId": 1, "floorsCount": 5 },
            {"name": "super zhk12", "estateType": 1, "isActive": false, "cityId": 1,
"stateId": 1, "floorsCount": 5 },
            {"name": "super zhk13", "estateType": 1, "isActive": false, "cityId": 1,
"stateId": 1, "floorsCount": 5 },
```

{"name": "super zhk14", "estateType": 1, "isActive": true, "cityId": 1, "stateId": 1, "floorsCount": 5 }

]
},
{
"name": "buyers",
"schema": [
{"name": "name", "type": "STRING", "mode": "REQUIRED"},
{"name": "surname", "type": "STRING"},
{"name": "age", "type": "INTEGER", "mode": "REQUIRED"},
{"name": "newbuildId", "type": "INTEGER"}
],
"rows": [
{"name": "vlad", "surname": "panchuk", "age": 22, "newbuildId": 10 },
{"name": "katya", "surname": "liubinchak", "age": 22, "newbuildId": 10 },
{"name": "kotlarevski", "surname": "just a kit", "age": 1, "newbuildId": 10 }
]
}
]
}

ДОДАТОК В
(обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

Налаштування процесів автоматизованого вивантаження даних з локальної бази даних до сервісів Google Cloud

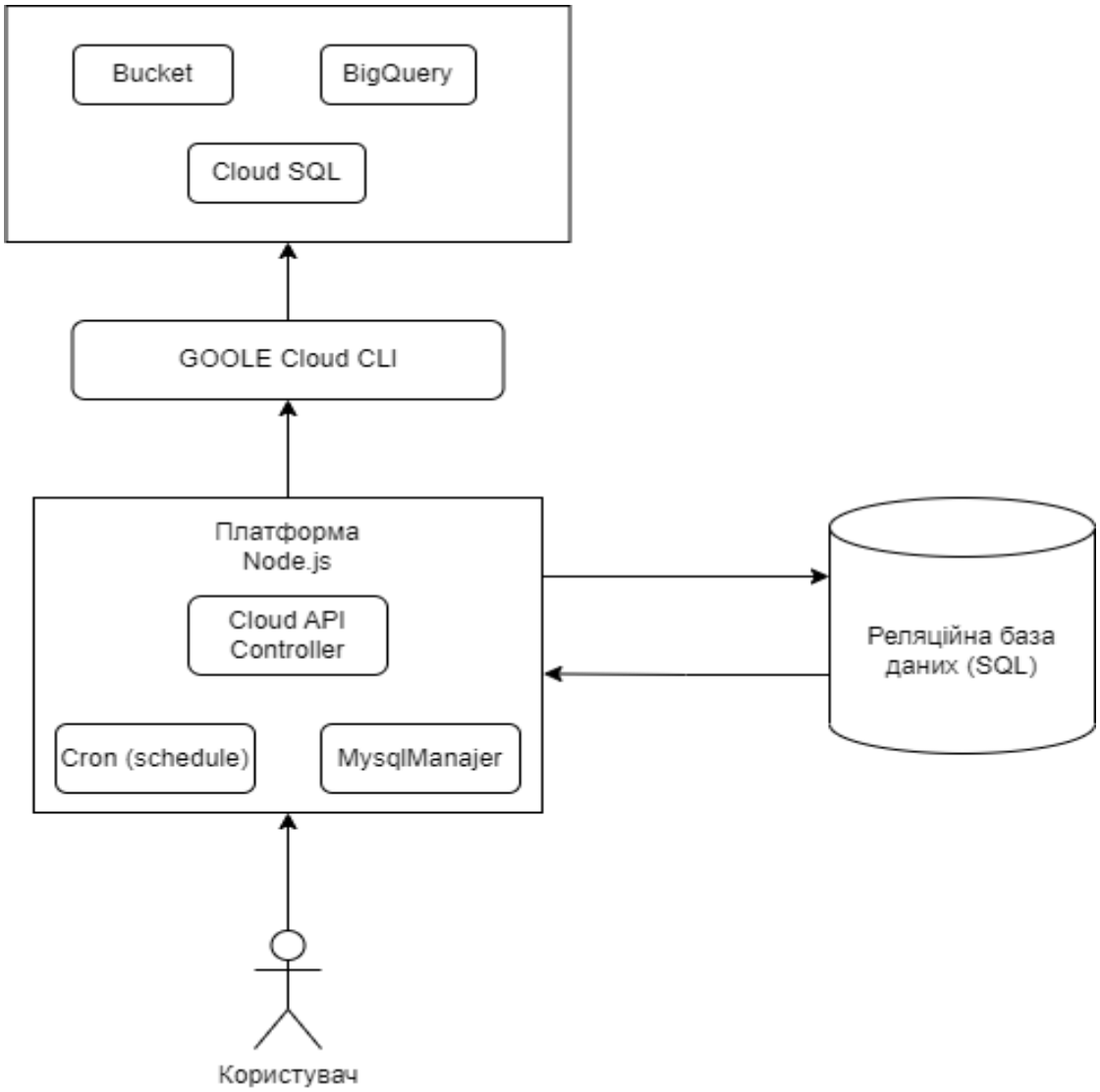
Студент групи 2 АКІТ-21м

Підпис  Владислав ПАНЧУК
Ім'я ПРІЗВИЩЕ

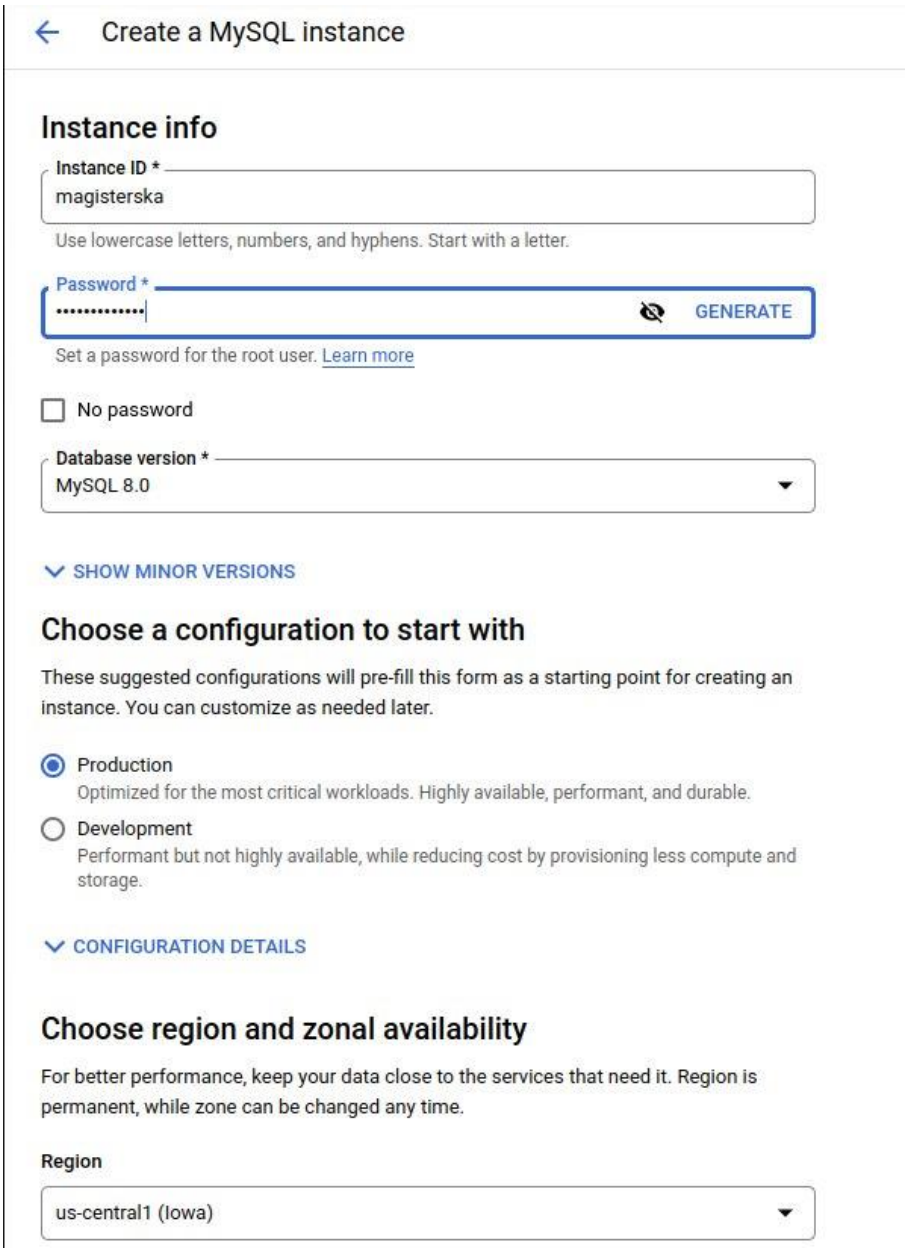
Керівник к. т. н., доцент кафедри КСУ

Підпис  Марія ЮХИМЧУК
Ім'я ПРІЗВИЩЕ

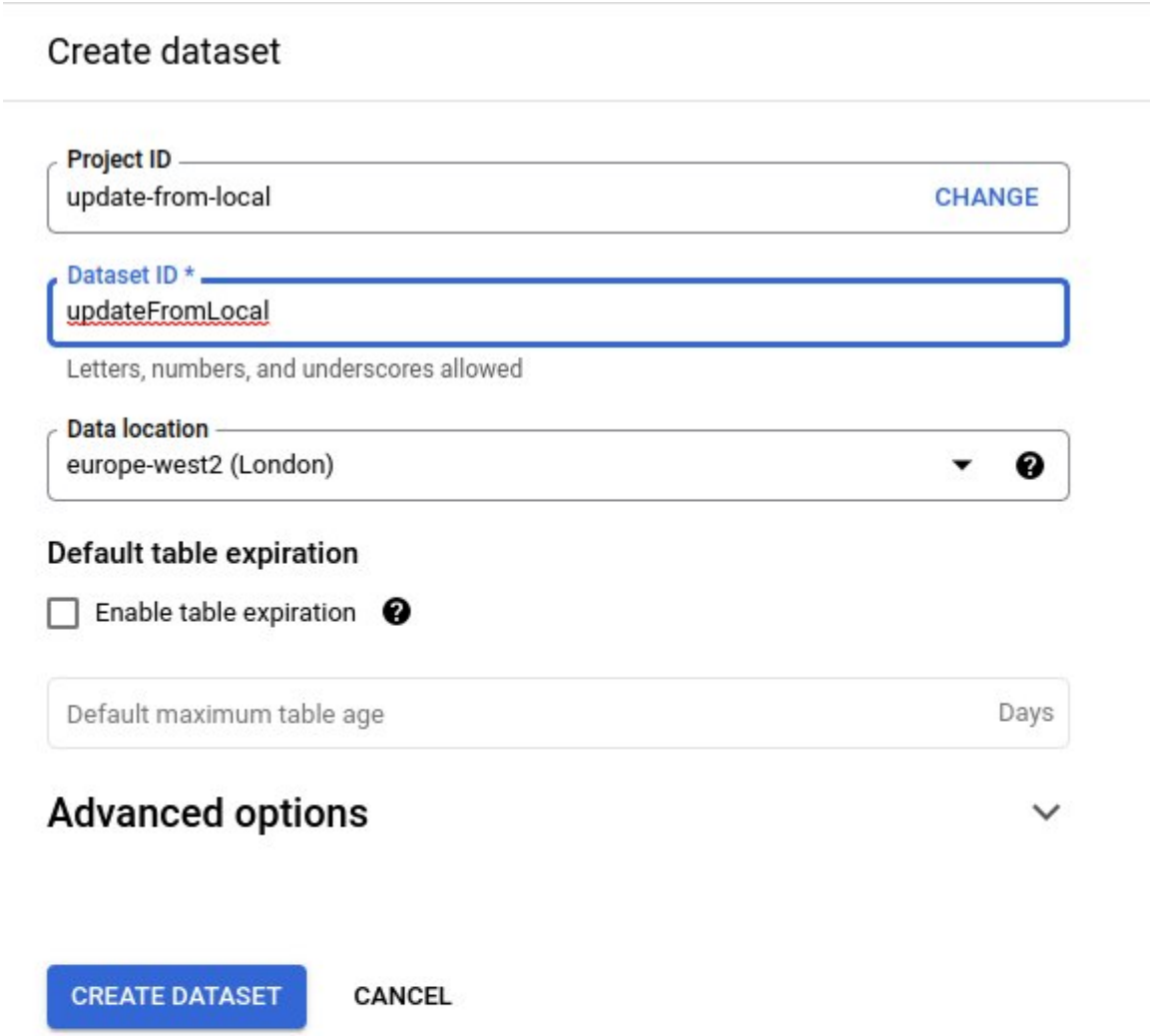
Схематичне зображення роботи розробленої програми з локальною базою даних та сервісами Google Cloud



Налаштування сутності CloudSql



Налаштування датасету в Google Cloud(BigQuery)



Схематичне зображення проведення тестової міграції даних на хмарний сервіс BigQuery