Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Магістерська робота

освітній ступінь – магістр

на тему: **«ПОБУДОВА БАГАТОРІВНЕВОГО ВЕБ-ЗАСТОСУВАННЯ З ВИСОКОЮ ДОСТУПНІСТЮ НА ХМАРНІЙ ПЛАТФОРМІ GCP ДЛЯ ДИСТАНЦІЙНОГО НАВЧАННЯ**

Виконав: студент 2-го року навчання,

Спеціальності
121 Інженерія програмного забезпечення

Накитняк Вадим Ігорович

Керівник Черкасов Д.І.,

кандидат технічних наук, старший

викладач

Рецензент_____
(прізвище та ініціали)

Магістерська робота захищена з

оцінкою_____

Секретар ЕК_____

“____” _____ 2024 р.

Київ 2024

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
зав. кафедри інформатики
к.ф-м.н., доц. Гороховський С.С.
_____
(підпис)
“_____” _____2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на магістерську роботу

студенту 2 р.н. магістерської програми Інженерія Програмного Забезпечення Накитняку Вадиму Ігоровичу
Розробити Багаторівневе веб-застосування з високою доступністю на хмарній платформі GCP для дистанційного навчання

Зміст текстової частини до магістерської роботи:

Зміст
Анотація
Вступ
1 Огляд наявних рішень
2 Структурна розробка власного рішення
3 Детальна розробка вебзастосування
Висновки
Використані джерела
Додатки

Дата видачі “_____” _____ 2023 р.

Керівник
Черкасов Д.І, ст. викладач, к.т.н _____
(підпис)

Завдання отримав
Накитняк В.І. _____
(підпис)

Тема: Побудова багаторівневого веб-застосування з високою доступністю на хмарній платформі GCP для дистанційного навчання

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проєкту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу	16.10.2023	
2.	Огляд технічної літератури за темою роботи	20.11.2023	
3.	Виконання аналізу сучасних рішень	16.01.2024	
4.	Огляд можливостей Google Cloud Platform та порівняння з аналогами	16.02.2024	
5.	Погодження функціонального призначення вебзастосування з керівником	05.03.2024	
6.	Написання вступу та першого розділу роботи	13.03.2024	
7.	Розробка архітектури системи та написання другого розділу	30.03.2024	
8.	Розробка веб-застосування для дистанційного навчання, інтеграція з GCP	11.04.2024	
9.	Написання третього розділу	10.05.2024	
10.	Попередній захист роботи	17.05.2024	
11.	Вдосконалення роботи за результатами попереднього захисту і відгуку керівника	28.05.2024	
12.	Остаточне оформлення роботи та створення презентації	31.05.2024	
13.	Захист магістерської роботи	11.06.2024	

Студент Накитняк Вадим Ігорович

Керівник Черкасов Дмитро Іванович “____”_____ 2024 р.

ЗМІСТ

Анотація

Ця робота присвячена розробці багаторівневого вебзастосування з високою доступністю на хмарній платформі GCP для дистанційного навчання. У роботі було проведено дослідження наявних рішень у сфері хмарних технологій, а також у сфері роботи з даними, проведено аналіз можливих компонентів для використання в розроблюваній системі та впроваджено власне рішення на основі розглянутих інструментів на хмарній платформі GCP. Результатом роботи є вебзастосунок для дистанційного навчання, який складається з декількох модулів. Розроблено модуль для роботи з даними, який може створювати пайплайни для вивантаження даних з локальних баз клієнтів в централізоване сховище даних Firestore.

Ключові слова: GCP, data ingestion, Apache Beam, Dataflow

Вступ

У світлі стрімкого розвитку технологій, бізнеси різних масштабів все частіше звертаються до хмарних провайдерів, Google, Amazon та Microsoft, для вирішення різноманітних завдань, пов’язаних із забезпеченням інфраструктури для вебзастосунків, зберіганням та резервним копіюванням даних, аналізом великих обсягів даних, налаштуванням мережевих сервісів та інших компонентів інформаційної інфраструктури. Хмарні рішення привертають увагу завдяки їх високій доступності, гнучкій масштабованості, відмовостійкості та іншим перевагам. Завдяки цьому, вони здатні спростити більшість проблем, які можуть виникати в сучасних програмних продуктах. Одною з таких проблем є збір та обробка даних.

Збір та обробка даних в умовах сучасного світу стають ключовими завданнями, які дозволяють підприємствам та організаціям приймати обґрунтовані рішення та вдосконалювати свої послуги та продукти. Ефективне використання хмарних інструментів для цих цілей може значно полегшити процес збору, зберігання та обробки великих обсягів даних. Типовими сценаріями для збору даних є: збір даних для подальшої інтеграції з інструментами машинного навчання, міграція даних з локальних баз даних на хмарну інфраструктуру, збір гетерогенних даних в сховище даних. Сфера застосування включає майже всі відомі напрямки, від IoT до освітніх застосунків.

В даній роботі, зокрема, планується зосередитись на проєктуванні вебзастосування системи для дистанційного навчання та детальній розробці пайплайну для збору та обробки освітніх даних. Основна мета запропонованої системи – надання зручного інтерфейсу для керування освітнім процесом онлайн, моніторинг освітнього процесу учнів, їх результатів та здібностей.

Система для дистанційної освіти може містити багато модулів для використання учнями, їх батьками та вчителями. З розвитком технологій машинного навчання та штучного інтелекту, також з’явилась можливість покращити досвід користувачів та зробити застосунок більш персоналізованим під окремого учня.

Для розгортання вебзастосування буде використано Google Cloud Platform. Хоча хмарні рішення і мають беззаперечні переваги у сфері високої доступності та масштабованості, важливо враховувати також і інші аспекти пов’язані з безпекою даних і захистом приватності користувачів. Планується уважно вивчити та врахувати ці та інші аспекти в процесі дизайну архітектури навчальної системи та під час детальної розробки її модулів.

1. Огляд наявних рішень

1.1. Сучасна архітектура вебзастосувань

Протягом останнього десятиліття архітектура вебзастосувань не зазнала кардинальних змін. Найбільш значним прогресом стало зміщення від монолітної архітектури до більш сервісно-орієнтованих підходів. Це, в першу чергу, пов’язано зі стрімким розвитком хмарних обчислень, які змогли частково усунути обмеження пов’язані із горизонтальним масштабуванням та обмеженими фізичними ресурсами.

На сьогодні існують наступні типи архітектур для побудови вебзастосувань: монолітна архітектура, мікросервісна архітектура, мультирівнева, або багаторівнева архітектура та подійно-орієнтована архітектура [1]. Розглянемо кожну з наявних архітектур окремо.

Монолітна архітектура

Найбільш класичний, стабільний та інтуїтивний підхід до розробки програмного забезпечення. В монолітній архітектурі весь додаток розглядається як одне ціле, зазвичай побудоване на єдиній технологічній платформі. Усі компоненти додатка, такі як інтерфейс користувача, бізнес-логіка та доступ до даних, реалізуються в межах одного кодового базису. Це означає, що при внесенні змін у функціонал додатка необхідно перекомпілювати та перезавантажувати весь додаток.

Рисунок 1 Схема монолітної архітектури

Мікросервісна архітектура

На відміну від монолітної архітектури, в мікросервісній кожен модуль програми, або мікросервіс, є окремим незалежним сервісом, який може функціонувати та масштабуватися незалежно від інших сервісів. Кожен сервіс відповідає за свою область функціонала та може бути розгорнутий, масштабований та оновлюваний без значного впливу на решту системи.

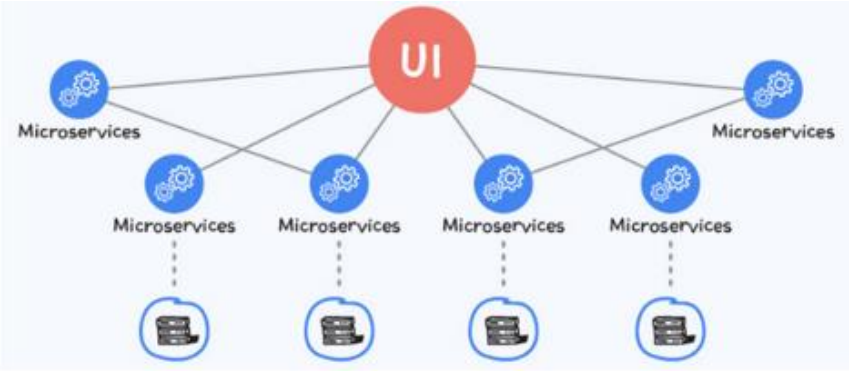


Рисунок 2 Схема мікросервісної архітектури

Багаторівнева архітектура

У багаторівневій архітектурі програмне забезпечення розділяється на логічні рівні, які можуть включати рівень презентації, або відображення, рівень бізнес-логіки та рівень доступу до даних. Ці рівні комунікують між собою за допомогою визначених інтерфейсів. Додаткові рівні, такі як сервіси або інфраструктурні компоненти, можуть бути додані залежно від потреб системи.

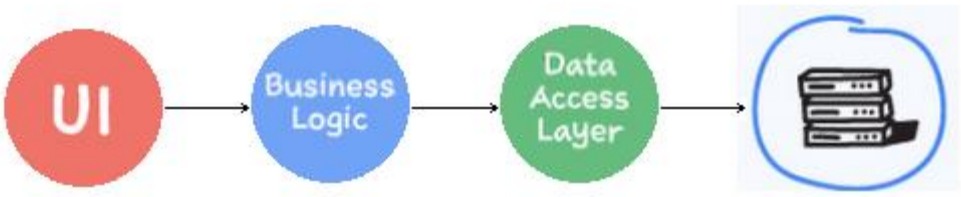


Рисунок 3 Схема багаторівневої архітектури

Подійно-орієнтована архітектура

У подійно-орієнтованій архітектурі компоненти програмного забезпечення спілкуються за допомогою асинхронного обміну повідомленнями або подіями. Кожен компонент може бути виробником подій або підписником на події інших компонентів, що дозволяє побудувати гнучку та розширювану систему.

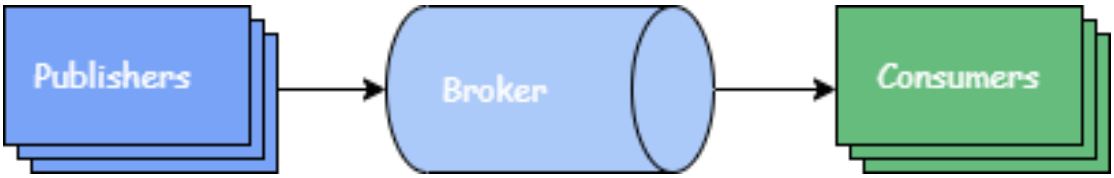


Рисунок 4 Схема подійно-орієнтованої архітектури

1.2. Хмарні платформи

Для ефективної реалізації будь-якої архітектури та її розгортання необхідно мати сервери, чи то фізичні, чи віртуальні. Раніше це можна було зробити тільки на власних потужностях. Це потребувало значних капіталовкладень, оскільки потрібно було забезпечити серверну з достатньою кількістю фізичних серверів. Проте повністю розгорнуту всю інфраструктуру застосунку на власному залізі недостатньо, також потрібно було слідкувати за серверами, і в разі потреби збільшувати обчислювальні потужності або обсяг доступної пам’яті, докуповуючи відповідне обладнання. Через це виникали проблеми з масштабованістю, доступністю, та ефективністю.

Для подолання цих та інших проблем на допомогу прийшли провайдери хмарних послуг. Сьогодні вже складно уявити сучасний застосунок, який тим чи іншим чином не покладався б на хмарні обчислення. Хмарні провайдери не лише надають інфраструктуру для розгортання сучасних застосунків, але й

пропонують широкий спектр послуг для автоматизації та полегшення процесу розробки. Однією з таких послуг є CI/CD (Continuous Integration/Continuous Deployment), що дозволяє розробникам автоматизувати процеси розгортання та впровадження змін у вебзастосунки. Це дозволяє швидко та ефективно додавати нові функції, виправляти помилки та забезпечувати стабільну роботу додатків.

Поміж інших керованих сервісів, які пропонують хмарні провайдери, можна відзначити такі, як керовані бази даних, сервіси для кешування, моніторинг та аналіз даних, автоматизовані системи безпеки та захисту, а також інструменти для моніторингу та аналізу ресурсів та використання даних. Ці сервіси дозволяють компаніям зосередитися на розробці застосунків, не витрачаючи час на управління та підтримку інфраструктури.

Хмарні рішення також забезпечують гнучкість та масштабованість інфраструктури. Компанії можуть легко збільшувати або зменшувати обсяги обчислювальних ресурсів та ресурсів для зберігання даних згідно з потребами проєкту, що дозволяє ефективно використовувати ці та інші ресурси та оптимізувати витрати на утримання інфраструктури. Такий підхід забезпечує надійність та ефективність у роботі з даними та додатками, дозволяючи компаніям швидко реагувати на зміни у вимогах та умовах ринку.

1.3. Великі дані в сучасних вебзастосуваннях

Іншу важливу роль у покращенні функціональності та ефективності сучасних вебзастосувань відіграють великі дані. Для того, щоб оптимально використовувати великі дані, цей процес поділено на рівні, починаючи з джерел даних, таких як вебсайти, мобільні додатки, сенсори IoT, соціальні мережі тощо. На першому рівні відбувається збір та потік даних, де дані збираються з різних джерел та трансформуються для подальшої обробки. На

другому рівні, дані зберігаються у великих сховищах даних, таких як data lakes (озера даних) або data warehouses (сховища даних). Третій рівень включає обробку та аналіз даних, де дані агрегуються, аналізуються та використовуються для прийняття рішень та покращення досвіду користувачів. І на останньому рівні, дані подаються для використання цільовим системам.

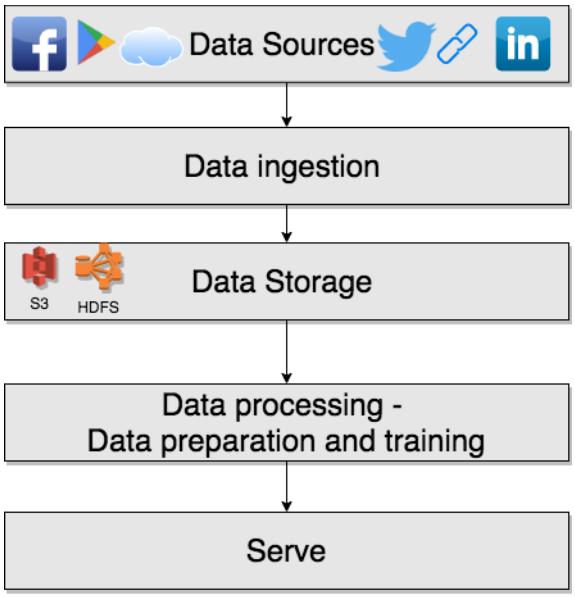


Рисунок 5 Рівні великих даних

1.4. Збір даних

Приймання, або збір, даних можна вважати першим рівнем великих даних [2]. Цей рівень відповідає за імпортування великої кількості неупорядкованих гетерогенних даних з одного чи різних джерел в єдине інформаційне середовище. Таким єдиним середовищем зазвичай являється база даних, для структурованих даних, або сховище даних для напівструктурованих даних. Збір даних також включає підготовку даних, а саме: очищення, трансформацію та інтеграцію даних.

Метою збору даних є забезпечення доступності та придатності великих обсягів інформації для подальшого аналізу, обробки, візуалізації та інтеграції в production-середовищі.

Розрізняють два основні типи збору даних: пакетне (batch ingestion) та потокове (real-time ingestion). Пакетний збір даних є найпоширенішим типом, де дані збираються партіями у великій кількості. Цей процес відбувається через певний проміжок часу як запланована активність. На відміну від пакетного, потоковий збір є безперервним процесом для збору даних в реальному часі. Він використовується, коли для потреб бізнесу важлива швидкість надходження актуальних даних, наприклад в соціальних мережах. Також існують й інші різновиди, такі як: гібридний збір даних, збір даних API, вебскрейпінг даних, збір даних з логів, збір даних IoT. Але ці типи можна так чи інакше віднести до категорій пакетного, чи потокового збирання.

Щодо проблем при зборі даних, основні виклики при зборі даних полягають у забезпеченні їхньої точності та якості, врахуванні різноманітних форматів і джерел даних, швидкості обробки, а також підтримці безперервного процесу.

Важливими складовими збору даних є процеси Extract-Transform-Load (ETL), Extract-Load-Transform (ELT), а також Extract Transform Load Transform (ETLT). Основна відмінність полягає в послідовності операцій: ETL обробляє дані перед їх завантаженням до сховища даних, тоді як ELT використовує потужність сховища даних для обробки даних після завантаження. Щодо ETLT, то цей тип комбінує обидва підходи, виконуючи базову підготовку даних перед завантаженням і додаткову обробку в сховищі.

1.5. Інструменти для збору даних

У світі великих даних існують готові рішення для збору даних: Apache Flume, NiFi, Kafka [3]. Вони розроблені для ефективного збору, обробки та передачі даних у реальному часі, незалежно від їхнього джерела та обсягу. Розгортання цих інструментів в хмарних середовищах дозволяє легко масштабувати їхні функції згідно з потребами проєкту, забезпечуючи високу доступність, еластичність та швидкість обробки даних. Крім того, вони можуть інтегруватись з іншими сервісами хмарних платформ для реалізації різних сценаріїв обробки та аналізу даних.

1.6. CDC

Ще одним важливим концептом у сфері збору даних є Change Data Capture (CDC). CDC - це патерн для безперервної ідентифікації та фіксації поступових змін даних в базі. Цей патерн виник 20 років назад, і основна його мета полягає в тому, щоб в режимі реального часу реплікувати дані з одного джерела в інше. CDC забезпечує ефективну передачу даних із низькою затримкою в робочу систему, а також для аналітики, з низьким впливом на production-середовище. Він фіксує зміни в базі, як тільки вони відбуваються, і гарантує, що дані в цільовій системі будуть актуальними та узгодженими з вихідною системою.

CDC чудово підходить для сучасних хмарних архітектур, оскільки це високоефективний спосіб переміщення даних через глобальну мережу інтернет. Можна сказати, що це своєрідний ETL процес, у якому дані періодично витягуються з вихідної системи, перетворюються відповідно до вимог цільової системи та завантажуються в цільову систему.

Існують 4 різні типи CDC [4]:

1. На основі міток часу

Найпростіший метод для CDC, який полягає у додаванні колонок з часовим типом до потрібних таблиць. Порівнюючи мітку останньої зміни запису в базі, можна прослідкувати нові зміни. Цей метод працює для всіх типів баз даних, проте має свої обмеження, оскільки враховує не всі зміни даних за проміжок часу від одного викачування даних до іншого.

2. Но основі журналу логів

Це найбільш надійний спосіб, який полягає у зчитуванні специфічних для баз даних логів транзакцій. В цих логах зберігаються всі зміни до бази даних, включаючи додавання, видалення та редагування записів.

3. На основі тригерів

Цей метод оснований на використанні механізму тригерів в реляційних базах даних. При оновленні основної таблиці, запускається код тригера, який оновлює пов’язану аудит-таблицю. Оскільки саме по собі використання тригерів є поганою практикою, то цей підхід не рекомендовано до використання в production-середовищі.

4. На основі запитів

Найбільш гнучким підходом до CDC є використання кастомних запитів до цільової бази даних. Такий запит має регулярно запускатись і оновлювати дані в вихідній табличці. Так само як і перший підхід з мітками часу, цей підхід підійде для всіх баз даних, бо є універсальним.

1.7. CDC в базах даних

В різних базах даних існують власні реалізовані механізми CDC. В більшості випадків цим механізмом є журнал транзакцій. Наприклад, в MySQL

це binlog, в PostgreSQL - write ahead log, в SQL Server - Sql Agent. В нереляційних базах також є схожі механізми. Наприклад, в ScyllaDB CDC увімкнено для кожної таблиці автоматично. Цей механізм працює за замовчуванням і для кожної таблиці створює іншу таблицю даних. Вона зберігається розподілено в кластері, маючи ті самі властивості, що й звичайні дані. Немає потреби в окремому спеціалізованому додатку для його використання.

1.8. Платформи CDC

Для реалізації CDC існують готові платформи, які автоматично ідентифікують та фіксують зміни у даних бази даних, з подальшим доставленням цих оновлень у режимі реального часу на потрібну платформу даних. Одними з таких інструментів є Debezium, Qlik та Dataphos Publisher. Важливо зазначити, що ці інструменти насамперед створені для систем реляційних баз даних, але є і винятки у використання їх з іншими джерелами даних.

1.9. Проблеми CDC

Отже, для того, щоб створити процес безперервної реплікації даних з бази даних до виділеного сховища даних, є два варіанти: або запускати самописний пайплайн для реплікації бази даних кожен день, чи навіть декілька разів на день; або використати досить дорогий інструмент для швидкої та надійної міграції даних. Що в першого, що в другого підходу є свої недоліки та переваги.

Щодо першого варіанту, то в результаті ми маємо не найнадійніший спосіб, який відпрацьовує з затримкою, тому дані доволі часто є не зовсім актуальними. Проте цей недолік не є критичним у випадку, якщо вихідна база

рідко оновлюється, або клієнту чи користувачу не обов'язково бачити миттєвий результат. Ще одним недоліком є те, що записуючий пайплайн створює додатково навантаження на базу даних, і в production-середовищі це може вплинути на продуктивність застосунку, чи навіть зупинити базу даних. Тепер розглянемо недоліки інструментів CDC.

Причина номер один, чому інструменти CDC не застосовуються більш широко, це вартість [5]. Постачальники інструментів CDC інвестують значні кошти в розуміння форматів журналів багатьох баз даних, які самі можуть продовжувати розвиватися. Ця вимога щодо безперервного інвестування обмежує справжні інструменти CDC декількома ключовими гравцями, дозволяючи їм стягувати немаленьку платню за свій продукт.

Друга причина, чому інструменти CDC не застосовуються ширше, — це небажання адміністраторів баз даних вихідної системи надавати привілейований доступ до рівня журналу логів бази даних інструментам CDC сторонніх виробників. Це обумовлено міркуваннями безпеки, адже такий доступ може розкрити конфіденційну інформацію та підвищити ризик несанкціонованого доступу або інших вразливостей у системі.

Приклад CDC системи:

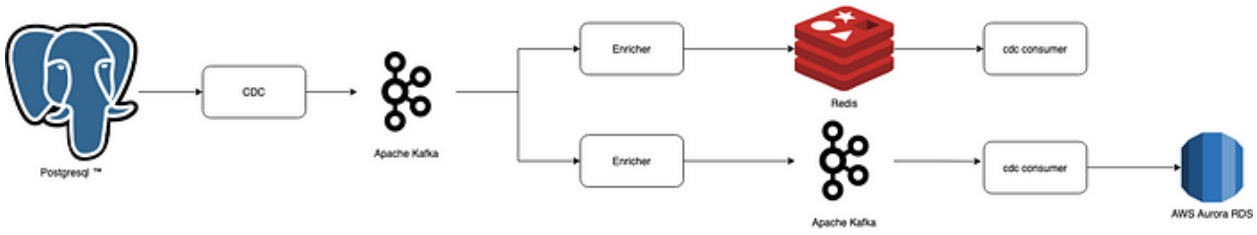


Рисунок 6 Приклад CDC системи [6]

1.10. Інструменти для збору даних на хмарній платформі GCP

При виборі певного інструменту для обробки даних на хмарній платформі, потрібно мати на увазі, що дані мають знаходитися близько від місця обробки чи аналізу. Саме тому відповідні сервіси зі збору даних бажано використовувати від того самого хмарного провайдера. Це дозволить скоротити витрати при переміщенні даних у внутрішній віртуальній приватній мережі, які надає провайдер. Попередньо кількість даних може бути необмеженою, тому одразу потрібно орієнтуватись на розробку легко масштабованого рішення, якщо в майбутньому доведеться підтримувати набагато більше користувачів, або даних.

Серед розглянутих раніше інструментів, всі з них можна розгорнути на хмарній інфраструктурі. Проте на хмарній платформі GCP є і власні, так звані нативні сервіси, які полегшують процес збору даних. Google розділяє сервіси для збору даних на 3 категорії: no-code(без коду), low-code(мало коду) та cloud native [7].

No-code

BigQuery Data Transfer Service є прикладом рішення, яке не потребує розробки кастомного пайплайну. Достатньо налаштувати передачу даних, і на регулярній основі дані будуть завантажуватись в базу BigQuery. Як джерело BigQuery Data Transfer Service дозволяє вказати понад 150 варіантів. Як зазначає Google: “Ваша аналітична команда може закласти основу для сховища даних BigQuery, не написавши жодного рядка коду [8].”

Low-Code (мало коду)

Data Fusion є сервісом для налаштування кастомізованих пайплайнів для збору та передачі даних з інтерактивним інтерфейсом. Він має 150 попередньо

доступних конекторів та трансформаторів, які можна поєднувати, та конфігурувати. Архітектура повністю керована Google Cloud, тому не потрібно думати про масштабування, оскільки Google робить це під капотом.

Cloud Native

Поєднання керованих сервісів GCP, таких як Pub/Sub та Dataflow, дозволяє розробляти кастомні пайплайни для збору та трансформації даних. У випадку Dataflow, Google також забезпечує автоматичне масштабування та оптимізацію обробки даних. Для запуску пайплайнів використовується уніфікована модель Apache Beam. Вона має SDK для мов програмування Python, Java та Go. Щодо Pub/Sub, то інтеграція з ним дозволяє писати на Dataflow стримінгові пайплайни. Всі події, які публікуються в топіку Pub/Sub направляються в пайплайн Dataflow, підписаний на оновлення цього топіку, для подальшої обробки та збереження.

Ще одним рішенням типу cloud native є Cloud Composer. Cloud Composer — це між платформний інструмент для оркестрування завдань з приймання даних з різних джерел, який підтримує AWS, Azure, GCP(і багато іншого) із можливостями керування, планування та обробки. Cloud Composer покладається на Apache Airflow, і керує всіма процесами, координуючи завдання, які можуть включати BigQuery, Dataflow, Dataproc, Cloud Storage тощо.

В порівнянні з Dataflow, Cloud Composer є більш комплексним рішенням. Тоді як основна мета Dataflow – просто виконувати завдання, то першочергова мета Composer – планувати виконання завдань. Це особливо корисно, присутні складні робочі процеси або потрібно координувати декілька десятків завдань.

В поєднанні з сервісом Cloud Scheduler, Dataflow може досягти того самого ефекту.

Отже, якщо потрібне просте рішення без написання коду для приймання необроблених даних, тоді Storage Transfer Service — надійний вибір. Для більшої гнучкості також можна використати Data Fusion. Але для максимальної кастомізованості й можливості обробляти дані з будь-яких джерел і виконувати різні завдання перетворення даних, краще скористатися Cloud Composer або Apache Beam у поєднанні з Cloud Dataflow.

1.11. Apache Beam

Apache Beam є уніфікованою моделлю для розробки пакетних і стрімінгових пайплайнів, розробленою компанією Google на основі низки внутрішніх технологій, зокрема MapReduce, FlumeJava та Millwheel. Основною метою Apache Beam є спрощення розробки масштабованих систем обробки даних за допомогою абстракції процесу обчислень, яка уніфікує різні типи обробки даних. Завдяки цьому, пайплайни на Apache Beam можуть виконуватися на різних розподілених обчислювальних рушіях.

Пайплайн(або конвеєр) — це граф перетворень, які застосовуються до колекцій даних. Цей граф являє собою DAG (Directed Acyclic Graph - Орієнтований Ациклічний Граф). DAG визначає порядок виконання обчислювальних операцій, де вузли графу представляють індивідуальні обчислення, а ребра — потік даних між ними [9].

Типовий Apache Beam пайплайн складається з послідовності перетворень. Перший крок включає вичитування даних з певного джерела за допомогою відповідного конектора. Джерелом може бути Pub/Sub, Cloud Storage, BigQuery або інше сховище даних. Після зчитування даних, вони

передаються в PTransform для виконання різних трансформацій. PTransform — це основний елемент обробки даних в Apache Beam. Трансформації можуть включати фільтрацію, агрегацію, об'єднання та інші операції. Apache Beam дозволяє розпаралелювати обробку даних для підвищення продуктивності. Завдяки цьому, трансформації можуть виконуватися одночасно на великій кількості обчислювальних вузлів. Після виконання всіх необхідних трансформацій, дані записуються в кінцеве сховище.

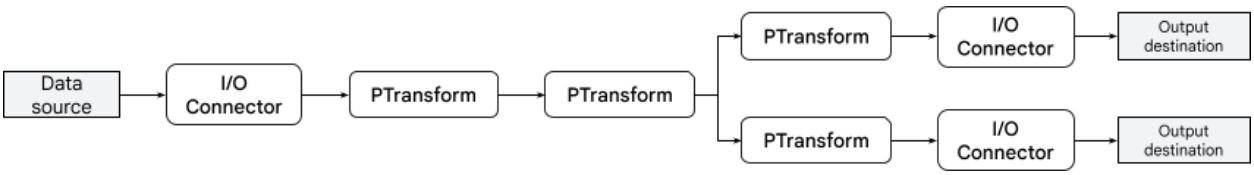


Рисунок 7 Схема типового Apache Beam пайплайну

1.12. Сховища даних на хмарній платформі GCP

Останнім кроком операції збору даних є, власне, збереження цих даних в сховище. Спершу потрібно розділити, які бувають сховища для даних. Існують реляційні бази даних, нереляційні бази даних, озера даних, а також інформаційні сховища. GCP надає готове рішення для кожного типу сховища.

Cloud SQL

Керований сервіс для зберігання даних в реляційній базі. На вибір представлено наступні типи РБДМС: MySQL, PostgreSQL, SQL Server. Частіше за все використовується для зберігання структурованих даних, які вимагають транзакційності. Такими даними можуть бути дані про користувачів, про товари чи замовлення, журнали активності тощо.

Bigtable

Це високопродуктивна, високомасштабна NoSQL база даних. Зазвичай використовується для зберігання великих обсягів даних, які вимагають швидкого доступу, таких як дані телеметрії, ігрові дані, дані IoT тощо.

Cloud Storage

Це об'єктне сховище для зберігання будь-яких типів даних у формі об'єктів. Підходить для зберігання різноманітних даних, включаючи неструктуровані файли, зображення, відео, резервні копії, журнали тощо.

Firestore

Гнучка та масштабована NoSQL база даних, що підтримує зберігання структурованих даних та автоматичну синхронізацію в реальному часі. Firestore підходить для зберігання даних користувачів, мобільних даних та даних вебзастосунків, а також для додатків, які потребують високої доступності та продуктивності.

BigQuery

Це повноцінне хранилище даних та сервіс аналітики в хмарі. BigQuery спеціалізується на аналізі великих обсягів даних, швидко виконує SQL-запити до даних петабайтного масштабу завдяки розподіленій архітектурі.

2. Структурна розробка власного рішення

2.1. Короткий опис загальної системи

Основна ідея запропонованої платформи полягає у наданні користувачам, в першу чергу адміністраторам навчальних закладів, а також учням і вчителям, освітньої платформи для дистанційного навчання зі зручним механізмом управління даними з навчального процесу. Завдяки розгортанню в хмарі, система зможе бути централізованою і розповсюджуватись у форматі PaaS(Platform-as-a-Service). Це означає, що кінцевим клієнтам не потрібно хостити систему і бази даних у себе, оскільки про все подбає інфраструктура GCP.

Дані надходитимуть в систему зі сторонніх джерел: локальних реляційних баз даних навчальних закладів, наприклад MySQL або PostgreSQL; CSV файлів, завантажених адміністраторами шкіл; або напряму додані адміністраторами через інтерфейс вебзастосунка. Початково сховищем даних буде виступати Firestore, проте в перспективі можна буде вивантажити їх в BigQuery для подальшого аналізу, а також покращення навчального процесу на рівні окремих закладів, або на рівні держави, завдяки великим даним.

Як попередньо було зазначено, система буде розміщена на хмарній платформі GCP. Більшість з використаних сервісів є сервісами Google. Єдиними сторонніми сервісами, використаними в архітектурі будуть сервіс аутентифікації Auth0, Git репозиторій GitHub для збереження коду проєктів і сервіс Hostinger для резервування доменного імені.

Щодо архітектури системи для дистанційного навчання на хмарній платформі, вона складатиметься з декількох модулів і поєднуватиме в собі

елементи не тільки багаторівневої, а також мікросервісної та подійно-орієнтованої архітектур.

Одна частина застосунку буде пов’язана з роботою над даними. Буде використано наступні сервіси платформи GCP: Dataflow, Firestore, Pub/Sub, Cloud SQL, Cloud Storage, Cloud Functions та Cloud Scheduler.

Частина застосунку для роботи із даними включатиме збір і обробку даних. Для збору даних буде використано автоматизовані пайплайни для одноразового або періодичного копіювання даних з клієнтських баз даних. Для створення цих автоматизованих пайплайнів буде окремий модуль для збору даних у вигляді API-сервісу, з яким будуть взаємодіяти клієнти через вебінтерфейс сторінки адміністратора школи. Цей модуль матиме додатковий механізм відновлення після відмови у вигляді черги повідомлень для можливості перезапуску хмарних функцій. Як основне сховище даних буде використано сервіс GCP Firestore, який ідеально підходить для рішень пов’язаних зі збереженням структурованих даних.

Щодо обробки даних, то дані з Firestore можна буде вивантажити в BigQuery завдяки сервісу для експорту даних. Зібрані таким чином дані, можна буде обробляти в окремих аналітичних пайплайнах, або навіть з використанням машинного навчання. Результати обробки даних будуть записуватись в BigQuery, а потім назад у Firestore для кінцевих користувачів. Також можливо напряму використовувати потужності BigQuery для аналізу, а ще BigQuery пропонує функцію машинного навчання з BigQueryML. Тому можливо створювати моделі та прогнозувати прямо з інтерфейсу користувача BigQuery з використанням SQL.

Друга, програмна частина системи, являє собою більш класичний MVC(Model-View-Controller) застосунок. Він складатиметься з App Engine для розгортання фронтенду, Cloud Run для розгортання бекенд сервісу API, Cloud Functions для виконання додаткових операцій над даними, Pub/Sub для черги створення черги завдань, CloudSQl з базою MySQL як допоміжна база даних, та Firestore як основна база даних для шкіл.

2.2. Multitenancy в GCP

Для вебзастосунків, які мають підтримувати більше ніж одного клієнта, в GCP присутня підтримка multitenancy(множинна оренда). В нашому випадку окремими клієнтами системи виступатимуть школи. Існує декілька способів для реалізації multitenancy в GCP: на рівні проєктів, на рівні інфраструктури та на сервісному рівні.

- На рівні проєктів

Кожен клієнт має власний проєкт, що забезпечує ізоляцію ресурсів та налаштувань безпеки. Цей підхід дозволяє повністю ізолювати інфраструктуру клієнтів, проте потребує найбільше ресурсів та коштів для підтримки реалізації.

- На рівні інфраструктури

Використання окремих інстанцій або контейнерів для кожного клієнта, забезпечуючи фізичну або віртуальну ізоляцію. Дані клієнтів все ще ізольовані, проте всі ресурси перебувають в межах одного проєкту на GCP.

- На рівні сервісів

Це реалізація логічної ізоляції в межах одного сервісу, наприклад, за допомогою окремих баз даних або схем для кожного клієнта. Є найпростішим

варіантом реалізації multitenancy, проте не завжди підходить під вимоги, оскільки дані клієнтів не є повністю ізольованими. Також, при виході зі строю одного компонента в системі, всі клієнти втрачають доступ до застосунку.

В запропонованій архітектурі multitenancy реалізовано на рівні сервісів. У ролі бази даних використано нереляційну базу даних Firestore. GCP надає можливість створювати окрему базу Firestore для кожного клієнта, з базовим обмеженням в 100 інстансів(можна збільшувати за запитом). Таким чином, при додаванні нової школи в систему, буде створено окремий інстанс бази даних Firestore з потрібними налаштуваннями доступності в тому регіоні, де перебуває клієнт.

Для підтримки різних шкіл, система має легко адаптовуватись та створювати потрібну інфраструктуру під нових клієнтів без затримок. Для того, щоб додати в систему нову школу, потрібно здійснити ряд кроків:

1) Адміністратор має створити заявку через вебінтерфейс, де має вказати мінімально необхідну інформацію про школу
2) В разі підтвердження заявки, на основі даних про школу, для адміністратора зі школи створюється особистий акаунт в Auth0 і надаються права адміністратора
3) Через Terraform в інфраструктуру додається нова база даних для школи, в яку встановлюються початкові дані
4) Після цих налаштувань, адміністратор має додати користувачів в систему за допомогою Data Ingestion Module API, через міграцію користувачів з локальної бази даних школи, якщо така база підтримується системою, або через завантаження CSV файлів в систему

2.3. Принципи високої доступності вебзастосувань

Висока доступність є ключовою властивістю інформаційних системи, яка забезпечує безперервну роботу та мінімальний час простою, навіть при виникненні збоїв або надзвичайних ситуацій. Для вебзастосувань, особливо тих, що надають критично важливі послуги, висока доступність є ключовим фактором, який забезпечує вдоволеність користувачів та збереження репутації.

Існує безліч шляхів та рішень, завдяки яким можна досягти високої доступності в системах. Легше всього скористатися вже готовими рішенням одного з провайдерів хмарних послуг. Наприклад, GCP гарантує час роботи 99.95% - 99.99 (близько 20 хв простою на місяць) на місяць для більшості зі своїх керованих сервісів, наприклад Cloud run, App Engine або Compute Engine.

Однак просто скористатися сервісами хмарних провайдерів не достатньо для досягнення високої доступності. Висока доступність завжди включає структурні рішення — резервування, балансування навантаження, відмовостійкість, надлишковість, резервне копіювання.

Ось кілька основних порад для забезпечення високої доступності в системах [10]:

- Мультирегіональність

Для того, щоб зробити вебзастосування невразливим до зональних неполадок, архітектуру застосунку потрібно розширювати та реплікувати на декілька регіонів одночасно і перемикати трафік з одного на інший у разі збою.

- Надлишковість

Системи з високими вимогами до надійності повинні уникати єдиних точок відмови та мати ресурси, репліковані через кілька доменів відмов. Домен

відмов — це група ресурсів, які можуть виходити з ладу незалежно, наприклад, віртуальні машини, зони або регіони. Реплікація через такі домени підвищує загальний рівень доступності.

- Усунення вузьких місць

Простими словами, потрібно прагнути горизонтального масштабування, оскільки вертикальне масштабування має певні обмеження. Цього можна досягти за допомогою шардингу, або розділення між різними віртуальними машинами або зонами. Також Google рекомендує використовувати власні керовані сервіси, які масштабуються горизонтально без втручання користувача.

- Запобігання стрибків трафіку

Не є бажаною синхронізація трафіку між клієнтами або сервісам в системі, оскільки забагато одночасних запитів можуть призвести до каскадних збоїв в системі. Для протидії рекомендовано використовувати стратегії для пом’якшення стрибків навантаження, такі як черги повідомлень, патерни мікросервісної архітектури, наприклад circuit breaker(автоматичний вимикач), та виваження критичних запитів.

- Автоматичне масштабування

Більшість керованих сервісів GCP мають автоматичне масштабування, тому можуть масштабуватись без втручання користувача. Для цього потрібно лише вказати мінімальну та максимальну кількість копій для того чи іншого сервісу.

- Мінімізація критичних залежностей

Наявність критичних залежностей в коді, може призводити до неочікуваних наслідків. Для того, щоб уникнути цього, варто використовувати статичні аналізатори коду, наприклад SonarQube.

2.4. Сервіси для високої доступності

Для досягнення високої доступності в GCP також існують окремі спеціалізовані рішення. Розглянемо деякі з них, які використано в архітектурі застосунку.

Google Cloud Load Balancer

Google Cloud Load Balancer розподіляє вхідні запити між кількома серверами в хмарних сервісах або на віртуальних машинах, забезпечуючи рівномірний розподіл навантаження. Cloud Balancer автоматично визначає здорові машини та перенаправляє трафік до них, що дозволяє мінімізувати час простою і забезпечити безперервну роботу застосунку. У ролі серверної частини для балансувальника навантаження в GCP можна використати групу віртуальних машин з підключеним Health Check, або безсерверне рішення, таке як CloudRun або App Engine, а також Cloud Storage, в який балансувальник перенаправлятиме трафік пов’язаний зі статичним контентом. Серед основних переваг Google Cloud Load Balancer: підтримка розподілення трафіку між регіонами, автоматичне масштабування, інтеграція з іншими сервісами GCP, такими як Cloud DNS, Cloud CDN та Cloud Armor

Google Cloud CDN

Google Cloud CDN (Content Delivery Network) ще один сервіс для забезпечення високої доступності, який використовує глобальну мережу точок присутності для кешування та доставлення контенту якомога ближче до кінцевих користувачів. Це знижує затримки та покращує продуктивність

застосунку, одночасно зменшуючи навантаження на сервери. Основними переваги Cloud CDN є: кешування контенту в глобальних точках присутності для швидшого доставлення, зменшення кількості запитів і в результаті навантаження до оригінальних серверів, проста інтеграція з Google Cloud Load Balancer.

Google Cloud Armor

Google Cloud Armor забезпечує захист вебзастосувань від DDoS-атак та інших вебзагроз. Використовуючи політики безпеки, Cloud Armor дозволяє створювати правила, що обмежують доступ до ресурсів на основі IP-адрес, географічного розташування або інших характеристик. Основні переваги Cloud Armor: автоматичне виявлення та блокування DDOS-атак великого масштабу, створення кастомізованих правил доступу для захисту ресурсів, захист застосунків, що використовують балансувальник навантаження.

Так, наприклад, в запропонованій системі створено правило, яке блокує незахищений трафік до вебзастосунка по публічній IP-адресі. Cloud Armor перевіряє заголовок ‘Host’ в HTTPS-запитах, і блокує ті, які не збігаються з доменним іменем застосунку. Дозволяється лише трафік направлений на доменне ім’я cloud-edu.online, або піддомен www.cloud-edu.online, з налаштованим SSL/TLS сертифікатом через сервіс Certificate Manager.

Cloud Endpoints

Cloud Endpoints, ще один сервіс GCP для забезпечення високої доступності, який виконує роль шлюзу API. Для того, щоб скористатись цим сервісом, потрібно додатково налаштувати та розгорнути проксі-сервер ESPv2. Cloud Endpoints виконує кілька важливих функцій для забезпечення надійної роботи систем, а саме, розподіляє запити між кількома бекенд-сервісами, та

автоматично масштабується. Серед основних його функцій є аутентифікація та авторизація, моніторинг і логування запитів, та кешування запитів. Таким чином, Cloud Endpoints сприяє підвищенню доступності, надійності та продуктивності.

Google Cloud Storage

Google Cloud Storage забезпечує надійне та високо доступне зберігання об'єктів з автоматичним резервуванням даних у кількох географічних регіонах та автоматичною реплікацією даних між декількома регіонами. Також існує функція версіювання файлів, для можливості відновлення файлів до попередніх версій.

2.5. Управління API в GCP

В створеній архітектурі, буде два основних сервіси, які надаватимуть програмний інтерфейс для доступу до системи, Data Ingestion Service та Cloud Edu Backend Service. Task Manager Service буде доступним лише для внутрішнього використання, і весь трафік буде курсувати в приватному VPC для більшої захищеності. Data Ingestion Service разом з Edu Backend Service будуть відкритими для інтернету і зовнішніх користувачів, тому потребуватимуть додаткових шарів безпеки. Для додаткового захисту та управління API GCP пропонує 3 сервіси: API Gateway, Cloud Endpoints та Apigee [11].

API Gateway

API Gateway це повністю керований сервіс, який дозволяє розробникам легко створювати, розгортати та керувати API. Він базується на технології Envoy, яка виступає як шлюз API на краю мережі і дозволяє ефективно і

безпечно обробляти трафік API. API Gateway підтримує аутентифікацію, ключі API, моніторинг, ліміти та квоти.

Cloud Endpoints

Cloud Endpoints також є керованим сервісом, який спрощує створення, розгортання та керування API. Використовує OpenAPI та gRPC для конфігурації API та інтегруються з Google Cloud сервісами для моніторингу, управління та аутентифікації.

Apigee

Apigee є найбільш гнучким і кастомізованим варіантом, але і коштує відповідно найбільше. Він найкраще підходить для великих підприємств, які потребують детального контролю над своїми API.

Через високу ціну Apigee, вибір сервісів відбувався між API Gateway та Cloud Endpoints. API Gateway краще підходить для легкого і швидкого розгортання API. Проте, Cloud Endpoints має більший контроль над конфігураціями, тому краще підходить, коли потрібна тісна інтеграція з іншими нативними сервісами Google Cloud. Також, Cloud Endpoints дозволяє інтегруватись з проксі ESPv2 (The Extensible Service Proxy V2).

ESPv2 це масштабований проксі-сервер, який стоїть перед бекендом API й забезпечує такі функції керування API, як аутентифікація, моніторинг і журналювання. ESPv2 підтримує специфікації OpenAPI V2, та gRPC [12]. Найбільшим недоліком є відсутність підтримки нової версії специфікації OpenAPI V3, проте ця підтримка відсутня і в інших сервісах Google Cloud.

2.6. Опис архітектури розроблюваної системи на GCP

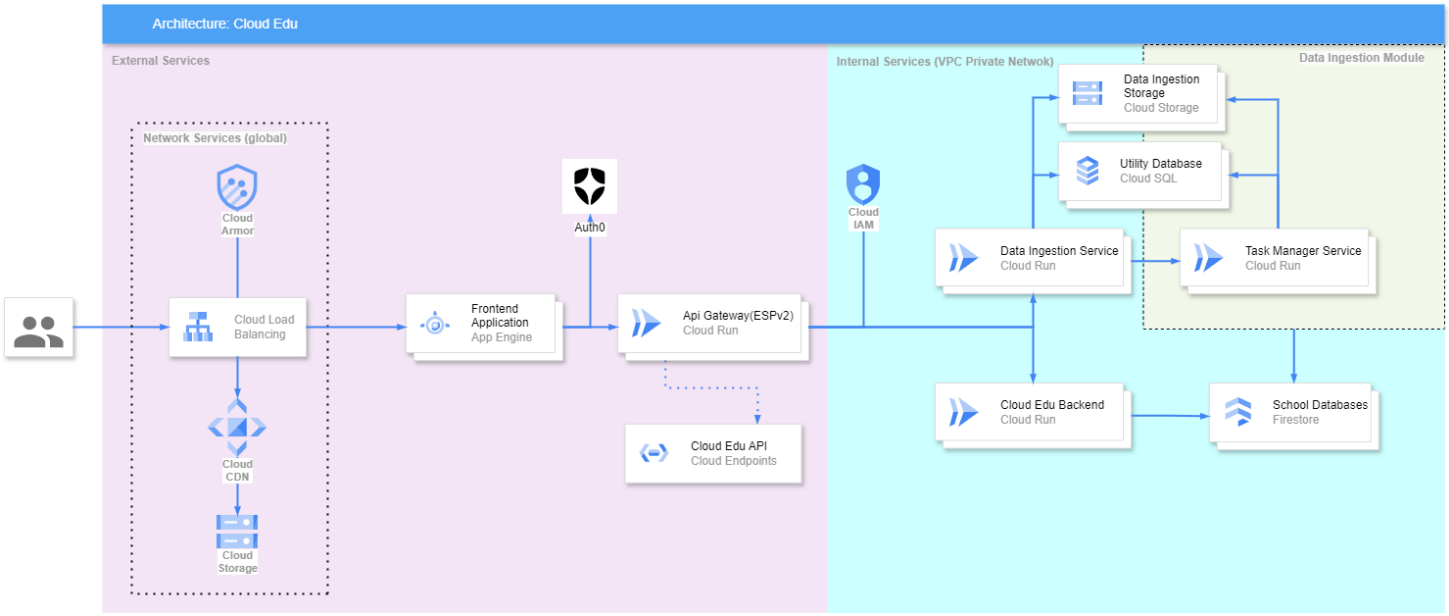


Рисунок 8 Архітектура системи для дистанційного навчання на GCP

Як і було зазначено, архітектура системи для дистанційного навчання складається з декількох основних модулів. Для кінцевих користувачів буде доступна лише передня частина застосунку, розгорнута на App Engine. Доступ до неї здійснюватиметься через балансувальник навантаження і буде дозволений виключно через протокол HTTPS, використовуючи доменне ім'я cloud-edu.online. Сертифікат SSL/TLS для безпечного зв'язку створено через GCP Certificate Manager та налаштовано на Cloud Load Balancer. Трафік додатково буде фільтруватись через Cloud Armor, а також деякий статичний контент буде автоматично кешуватись і надаватись користувачам через сервіс Cloud CDN.

Аутентифікація та авторизація користувачів здійснюватиметься через протокол Oauth2. Логін користувачів здійснюватиметься через Auth0, а нові акаунти учнів і вчителі зможуть створювати лише адміністратори через вебінтерфейс. Після успішного логіну, користувачу буде доступний інтерфейс

для школи, або шкіл, до яких він належить. Запити на бекенд здійснюватимуться з допомогою JWT токену, який буде верифікуватись та валідуватись через шлюз API ESPv2. Авторизовані через Auth0 запити шлюз надсилатиме приватну мережу VPC, з додатковою авторизацією для внутрішньої комунікації через Cloud IAM. Cloud IAM налаштовано так, що запити до сервісів Data Ingestion Module та Cloud Edu Backend будуть здійснюватись тільки з сервісного акаунту Api Gateway. Для цього буде використано identity token, який вони зможуть розпізнавати.

2.7. Детальний опис модулів вебзастосування

Надалі зосередимось на окремих модулях розроблюваної системи. Спочатку розглянемо частину застосунку пов’язану зі збором даних.

На етапі зі збором даних основна задача полягатиме в тому, щоб правильно синхронізувати дані, які надходять через Data Ingestion Service та через вебінтерфейс, оскільки дані можуть повторюватись. Також важливо забезпечити конфіденційність інформації кінцевих користувачів протягом всього конвеєру. А найголовніше в цьому — забезпечити безперервний потік даних від сторонніх джерел, до сховища даних і від сховища до нашого застосунку. Джерелами даних в нашому випадку будуть клієнти та їх локальні бази даних, а також CSV файли, попередньо завантажені адміністраторами шкіл в Cloud Storage. Клієнтські дані потраплятимуть в систему через представлені засоби для збору даних. Ці засоби додатково валідують і трансформують дані, приводячи їх до уніфікованого та зручного для використання вигляду і записуються в окрему базу школи в Firestore. Всі дані в Firestore є готовими для подальшої обробки, і подальшого експорту в BigQuery для аналізу та тренування в модулях машинного навчання.

Всі засоби використанні для роботи з даними є нативними сервісами GCP, тому з коробки включатимуть масштабування, логування, моніторинг та високу доступність(для мультирегіону). На більш детальній схемі архітектури модуля збору даних, можна побачити наявність черги повідомлень, яка слугуватиме допоміжною ланкою в підтримці функціонування системи. Pub/Sub допоможе розвантажити сервіс Cloud Run, та перенести частину задач на хмарні обчислення функцій, які можуть необмежено масштабуватись.

Детальна схема архітектури модуля збору даних:

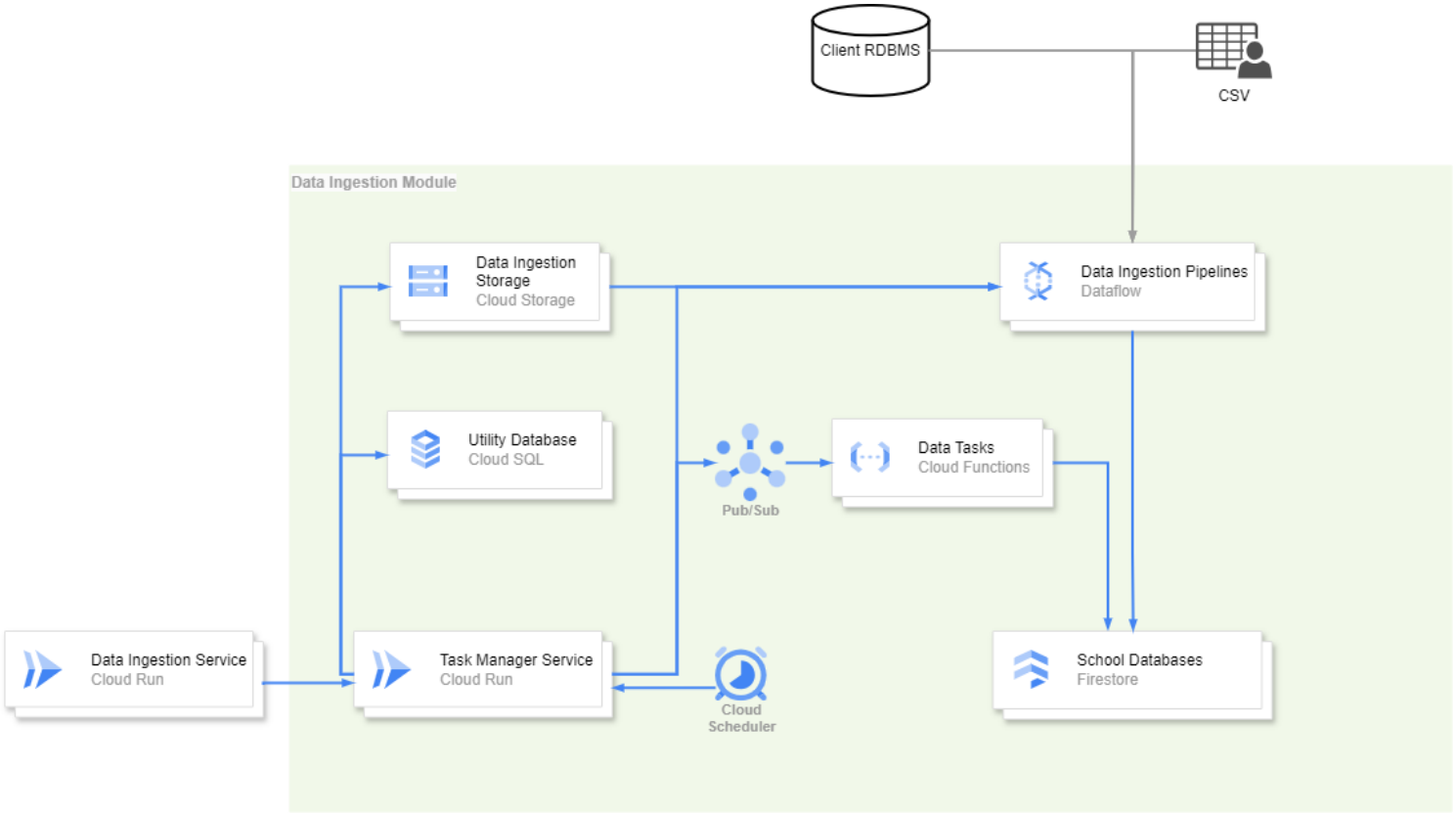


Рисунок 9 Детальна схема модуля Data Ingestion Module

Тепер детальніше пройдемося по основним модулям системи.

2.7.1.Data Ingestion Service

Основна мета Data Ingestion Service полягатиме в тому, щоб надати клієнтам, а саме школам, програмний інтерфейс для зручного завантаження даних. Прикладом такого API, може слугувати ДІЯ, яка надає інтерфейс для ЦНАПІв, для вивантаження замовлених послуг до репозиторію. Доступ до цього сервісу здійснюватиметься тільки через вебінтерфейс вебзастосування для користувачів з роллю адміністратора відповідної школи.

Ось неповний перелік ендпоінтів, які надаватиме сервіс:

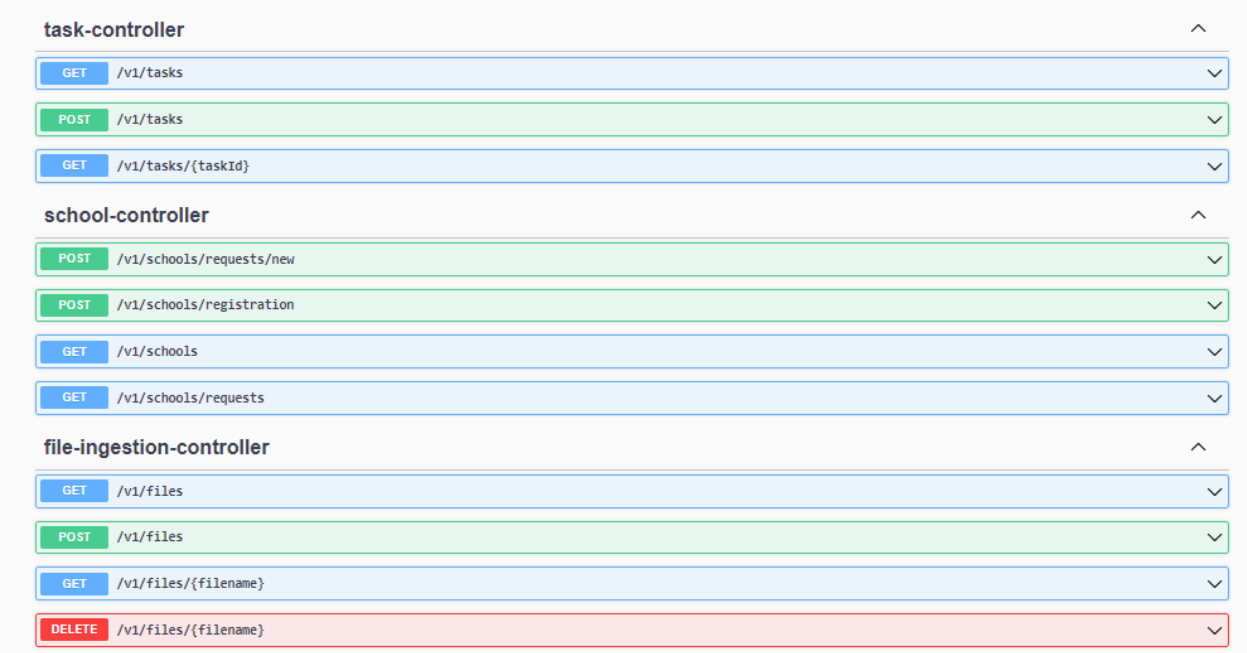


Рисунок 10 Перелік ендпоінтів сервісу Data Ingestion

2.7.2.Cloud Edu Backend Service

Це основний сервіс API, функціоналом якого будуть користуватись студенти та вчителі системи. API цього сервісу включатимуть запити пов’язані з отриманням списків учнів, вчителів, уроків, оцінок учнів, інформацію про відвідування уроків тощо.

2.7.3.Task Manager Service

Цей модуль буде відповідати за запуск конвеєрів для запису даних, за виконання запланованих задач та можливість їх повторного запуску. Самі задачі будуть запускатись в сервісі Cloud Function, одна задача = одна функція. Такими задачами можуть бути: імпортування даних, видалення даних, вивантаження даних з файлу в BQ, повторна валідація даних. Результати про виконані, невиконані, або задачі в процесі будуть зберігатись в CloudSQL базі, створеній під цей сервіс.

2.7.4.Data Ingestion Pipelines

Пайплайн для збору даних буде основною ланкою, яка поєднуватиме дані клієнтів, та нашу систему. Через пайплайни дані вивантажуватимуться з локальних баз клієнтів, а потрібні дані в процесі будуть у правильній формі записуватись у відповідний клієнтській базі Firestore в колекцію. Дані з цих колекцій вже будуть готовими для процесингу та використання в системі надалі.

2.7.5.Authentication/Authorization Service

Для аутентифікації зовнішніх запитів до в систему, буде використано сервіс Auth0, а для внутрішнього комунікування між сервісами базовий механізм GCP Cloud IAM. Клієнтські акаунти адміністраторів шкіл створюватимуться за запитом і матимуть права на доступ до модуля збору даних через вебінтерфейс. Для учнів та вчителів адміністратори створюватимуть акаунти через вебінтерфейс.

2.7.6.Frontend Application

Частина застосунку, яка відповідає за рівень відображення. Через інтерфейс вебзастосування користувачі отримуватимуть доступ до системи.

Запити до фронтенду надходитимуть з балансувальника навантаження, а фронтенд своєю чергою комунікуватиме зі шлюзом API, а також з сервісом для авторизації користувачів Auth0.

2.8. Моніторинг та логування в GCP

Основними сервісами GCP для моніторингу і логування є Cloud Monitoring та Cloud Logging. Всі логи створені на рівні додатку, в моєму випадку App Engine чи в Cloud Run, зберігаються в єдиному місці в Cloud Logging. Також там зберігаються логи з білдів Cloudbuild. Завдяки цьому можна прослідкувати всі помилки, які виникатимуть під час виконання вебзастосунка. Також це дозволяє зручно зневаджувати програмний код, додаючи до нього логи DEBUG рівня.

Ще одним сервісом, який допомагає слідкувати за роботою застосунків, а саме API, є Cloud Endpoints. Cloud Endpoints надає зручний інтерфейс для моніторингу всіх процесів пов’язаних з API. Він збирає статистику по всім доступним ендпоінтам, а саме кількість викликів, середній час виконання, наявність помилок тощо.

3. Детальна розробка вебзастосування

3.1. Використані технології

При виборі технологій надавалась перевагами тим, які є повністю, або частково сумісними з хмарними сервісами GCP.

Для розгортання інфраструктури в хмарі використано інструмент Terraform. Детальніше про інфраструктуру застосунку ідеться в наступному розділі. В якості Git репозиторію для проєктів використано GitHub, який теж є повністю сумісним з GCP, а саме з сервісом для CI/CD CloudBuild.

Для розробки рівня відображення, використано популярний фреймворк для фронтенд розробки React, мовою програмування JavaScript. Основна його перевага в популярності, тому доволі легко знаходити пов’язану інформацію в інтернеті, а також простоті, що робить його ідеальним кандидатом для швидкої побудови MVP(Minimum viable product) версії вебзастосування. Також JavaScript в поєднанні з React має нативну підтримку в сервісі App Engine, який дозволяє розгортати сервер фронтенду в хмарі всього однією командою - gcloud app deploy.

Розробка рівня бізнес-логіки велася мовою програмування Java версії 17. Як фреймворк для сервісів API обрано Spring Boot версії 3, як найбільш надійний та зрозумілий. Java ідеально підходить для розробки під GCP, оскільки має ряд бібліотек, розроблених Google, для легкої інтеграції під кожен сервіс GCP, від Cloud Storage до Firestore.

Розробка пайплайну для збору даних теж велася мовою програмування Java 17 з використанням SDK Apache Beam.

На рівні доступу до даних використано дві бази, нереляційна база Firestore, та реляційна база CloudSQL, а також додатково використано сховище для неструктурованих даних Cloud Storage.

3.2. Налаштування інфраструктури

Для налаштування інфраструктури було вирішено використати IaaC(Infrastructure-as-a-Code) рішення Terraform. Terraform є open source інструментом, який дозволяє створювати та керувати інфраструктурою на хмарних платформах і є сумісним з усіма хмарними провайдерами.

Додатково був розглянутий варіант із сервісом GCP Deployment Manager [13], проте він програє Teraformy у всіх аспектах, а в першу чергу в кастомізованості. Deployment Manager підтримує не всі ресурси наявні на цей момент в GCP. Єдиним явним недоліком Teraform в порівнянні з Deployment Manager є необхідність зберігати стан інфраструктури. Проте цей мінус легко перетворюється на перевагу, коли виникає необхідність відкотити систему до попереднього стану. Також це можливість прослідкувати всіх змін яких зазнала інфраструктура за весь час.

Основною перевагою використання IaaC рішення є можливість швидко розвернути інфраструктуру з нуля. Це дає змогу зробити репліку середовища розробки для різних потреб. Наприклад, в рамках одного проєкту GCP створити середовище **dev** для розробників, в рамках іншого проєкту **qa** середовище для тестування, та окремо середовища **uat** і **prod** для приймального тестування клієнтами й фінальне виробниче середовище відповідно.

Для симуляції клієнтських баз даних створено віртуальні машини та заповнено їх тестовими даними. Для генерації тестових даних використано

Python-скрипт з бібліотекою Faker, який генерує задану кількість даних про учнів і записує в csv файл. Ось код на Terraform створення тестових віртуальних машин клієнтів:

```
66      size_of_test_dataset = 100000
67      clients = {
68        client1-mysql-instance = {
69          machine_type = "n1-standard-1"
70          zone         = "us-central1-a"
71        }
72        client2-mysql-instance = {
73          machine_type = "n1-standard-1"
74          zone         = "us-central1-b"
75        }
76      }
77    }
78
79
80    resource "google_compute_instance" "mysql_instance" {
81      for_each     = local.clients
82      name         = each.key
83      machine_type = each.value.machine_type
84      zone         = each.value.zone
85      tags         = ["allow-ssh", "allow-mysql-access", "allow-ping", "allow-mysql"]
86
87      boot_disk {
88        initialize_params {
89          image = "ubuntu-2004-lts" # Ubuntu 20.04 LTS
90        }
91      }
92
93      metadata = {
94        ssh-keys = "${var.username}:${tls_private_key.ssh_key.public_key_openssh}"
95      }
96
97      allow_stopping_for_update = true
98
99      network_interface {
100       network = google_compute_network.vpc_network.name
101       access_config {
102         // Ephemeral IP will be assigned
103       }
104     }
105
106     service_account {
107       email  = google_service_account.instance_service_account.email
108       scopes = ["cloud-platform", "storage-full"]
109     }
110
111     metadata_startup_script = local.startup_script
112   }
```

Рисунок 11 Приклад коду інфраструктури на HCL

При створенні інстансу запускається startup_script(**додаток Г**), який встановлює всі потрібні пакети, такі як Git, python mysql, копіює потрібний

репозиторій з кодом скриптів, створює базу даних, налаштовує користувача і запускає скрипт для генерації тестових даних. Після налаштування бази MySQl вона заповнюється тестовими даними з csv файлів, згенерованими через python-скрипт з Faker.

3.3. CI/CD в GCP

Для процесів CI/CD на хмарній платформі GCP використано сервіси Cloud Build і GitHub в якості Git репозиторію, оскільки нативний сервіс GCP Google Cloud Repositories скоро втрачає підтримку. На Github всього створено 5 репозиторіїв: kma-ms-thesis-tf, cloud-edu-front, data-ingestion-pipelines, cloud-edu-back, task-manager-service. Детальний опис репозиторіїв з посиланнями на Github:

1) Репозиторій kma-ms-thesis-tf містить інфраструктурний код на HCL для Terraform(репозиторій приватний через наявність чутливої інформації).
2) Репозиторій cloud-edu-front(https://github.com/brutovsky/cloud-edu-front) містить код передньої частини застосунку.
3) Репозиторій data-ingestion-pipelines(https://github.com/brutovsky/data-ingestion-pipelines) містить код Apache Beam темплейтів пайплайнів для збору даних.
4) Репозиторій cloud-edu-back(https://github.com/brutovsky/cloud-edu-back-public) містить мультимодульний Maven проєкт, який складається з 4 Java модулів:
   4.1) Common – проєкт зі спільними класами, а також утилітами
   4.2) Cloud-edu-backend-module – основний модуль API, який надає доступ до системи користувачам

4.3) Data-ingestion-module – модуль API адміністраторів шкіл для управління освітніми даними

4.4) Deploy-module – модуль, створений суто для того, щоб поєднати код контролерів обох API через залежності і створення OpenAPI версії 2 специфікації у форматі Swagger yaml

5. Репозиторій task-manager-service(https://github.com/brutovsky/task-manager-service) містить код сервісу для створення та запуску задач по збору даних в Dataflow

Для використання GitHub в якості Git репозиторію, спочатку потрібно надати Google Cloud доступ до репозиторіїв, програмний код яких планується автоматизувати. В самих проєктах потрібно створити файли формату yaml, з переліком команд, або кроків виконання. Ці кроки можна задати як на стороні CloudBuild через Terraform, так і в самих репозиторіях. GCP зможе автоматично зчитати ці файли з Github, та виконає кожен крок в окремому вказаному Docker контейнері в середовищі Cloud Build.

Для того, щоб Cloudbuild реагував на зміни в Git-репозиторії, потрібно створити відповідні тригери, які будуть запускати файли з командами на виконання. Кожен тригер має бути прив'язаний до одного yaml файлу, проте в одному репозиторії їх може бути декілька. Наприклад, в проєкті cloud-edu-backend містяться файли cloudbuild-deploy.yaml та cloudbuild-push.yaml. Команди з cloudbuild-deploy.yaml активуються при злитті змін з develop гілки в main, і активують процеси CD - розгортання проєкту на CloudRun. Команди з файлу cloudbuild-push.yaml тригеряться при створенні коміту в гілку develop і запускають процеси CI - компіляції проєкту та тестування.

Приклад коду тригера в Terraform:

```
resource "google_cloudbuild_trigger" "data-ingestion-pipelines-deploy" {
  location = "us-central1"
  name     = "data-ingestion-pipelines-deploy"

  github {
    owner = "brutovsky"
    name  = "data-ingestion-pipelines"
    push {
      branch       = "main"
      invert_regex = false
    }
  }

  substitutions = {
    "_ENV"                        = var.env
    "_CLOUD_RUN_REGION"           = var.region
    "_DATAFLOW_ARTIFACTS_BUCKET" = google_storage_bucket.kma-ms-thesis-dataflow-artifacts.name
    "_DATAFLOW_TEMP_BUCKET"       = google_storage_bucket.kma-ms-thesis-dataflow-temp.name
  }

  filename = "cloudbuild.yaml"
}
```

Рисунок 12 Код на HCL, який визначає тригер в сервісі CloudBuild

Перелік виконаних та невдалих білдів в CloudBuild:

Status	Build	Source	Ref	Commit	Trigger Name	Created	Duration	Security Insights
✗	792a2664	brutovsky/data-ingestion-module	main	8c114e2	deploy-data-ingestion-app	5/8/24, 3:04 PM	1 min 45 sec	-
✓	77b1d167	brutovsky/data-ingestion-module	main	5906ddd	deploy-data-ingestion-app	5/8/24, 1:10 AM	2 min 1 sec	-
✓	a263a118	brutovsky/data-ingestion-module	main	118131f	deploy-data-ingestion-app	5/7/24, 9:29 PM	1 min 48 sec	-
✗	835f0f28	brutovsky/data-ingestion-module	main	74dcc77	deploy-data-ingestion-app	5/7/24, 9:29 PM	-	-
✓	b2c5dc12	brutovsky/data-ingestion-pipelines	main	53e905a	data-ingestion-pipelines-deploy	5/4/24, 1:10 AM	2 min 40 sec	-
✓	f6300367	brutovsky/data-ingestion-pipelines	main	50a7d83	data-ingestion-pipelines-deploy	5/4/24, 12:41 AM	2 min 31 sec	-
✗	757fa23c	brutovsky/data-ingestion-pipelines	main	db7ec33	data-ingestion-pipelines-deploy	5/3/24, 11:54 PM	1 min 51 sec	-
✗	6ebbc316	brutovsky/data-ingestion-pipelines	main	c49c3b0	data-ingestion-pipelines-deploy	5/3/24, 11:40 PM	2 min 4 sec	-
✗	6ccd11c5	brutovsky/data-ingestion-pipelines	main	2eaa12e	data-ingestion-pipelines-deploy	5/3/24, 9:32 PM	1 min 52 sec	-
✗	a447d56f	brutovsky/data-ingestion-pipelines	main	904eda3	data-ingestion-pipelines-deploy	5/3/24, 9:27 PM	1 min 54 sec	-
✗	074b001c	brutovsky/data-ingestion-pipelines	main	de189e4	data-ingestion-pipelines-deploy	5/3/24, 9:24 PM	2 min 14 sec	-
✗	e423cf19	brutovsky/data-ingestion-pipelines	main	353dc5e	data-ingestion-pipelines-deploy	5/3/24, 9:22 PM	27 sec	-
✓	0ab82c5e	brutovsky/data-ingestion-module	main	4ba0d4b	deploy-data-ingestion-app	4/17/24, 10:30 PM	1 min 35 sec	-
✓	dbdeb6b7	brutovsky/data-ingestion-module	main	2f4691c	deploy-data-ingestion-app	4/17/24, 10:26 PM	1 min 34 sec	-
✗	377ff38e	brutovsky/data-ingestion-module	main	2f4691c	deploy-data-ingestion-app	4/17/24, 10:23 PM	1 min 12 sec	-
✗	3f96a47e	brutovsky/data-ingestion-module	main	3cc29be	deploy-data-ingestion-app	4/17/24, 10:19 PM	29 sec	-

Рисунок 13 Список білдів в сервісі CloudBuild

3.3.1.Continious Integration

Метою безперервної інтеграції(CI) вебзастосування є постійна перевірка вихідного коду. CI дозволяє автоматизувати процеси перевірки правильності коду, тестування коду, а також забезпечує його інтеграцію між різними гілками(develop/release/main) на регулярній основі. Це зменшує ймовірність виникнення конфліктів між змінами, покращує якість коду та прискорює виявлення помилок завдяки швидкому зворотному зв'язку.

В рамках запропонованого вебзастосування реалізовано спрощену версію процесу CI, суть якого в основному полягає в запуску різних етапів збирача проєкту. У випадку модуля API у ролі збирача проєкту використано maven. Всі кроки CI процесу описані у yaml файлі:

```yaml
steps:
  - id: Fetch secret file
    name: 'gcr.io/cloud-builders/gcloud'
    entrypoint: 'sh'
    args:
      - '-c'
      - |
        # Fetch the secret and write it to a file
        gcloud secrets versions access latest --secret=security-definition-file > ${_SECURITY_DEFINITION_PATH}

  - id: Compile image
    name: "gcr.io/cloud-builders/mvn"
    env:
      - 'API_GATEWAY_HOST=${_API_GATEWAY_SERVICE_NAME}'
      - 'GCP_PROJECT_ID=${PROJECT_ID}'
    entrypoint: "bash"
    args:
      - "-cue"
      - |-
        cat /workspace/deploy-module/src/main/resources/securityDefinition.json
        mvn clean compile

  - id: Run unit tests
    name: "gcr.io/cloud-builders/mvn"
    env:
      - 'API_GATEWAY_HOST=${_API_GATEWAY_SERVICE_NAME}'
      - 'GCP_PROJECT_ID=${PROJECT_ID}'
    entrypoint: "bash"
    args:
      - "-cue"
      - |-
        mvn test

```

Рисунок 14 Вміст файлу cloudbuild-push.yaml

3.3.2.Continious Deployment

Безперервне розгортання служить для релізу нових версій вебзастосування і полягає у виконанні переліку кроків для збірки програмного коду, завантаження готової збірки в репозиторій артефактів і розгортання збірки в сервісі GCP. Розглянемо цей процес на прикладі проєкту cloud-edu-back з модуля API вебзастосування для дистанційного навчання. Ось приклад успішного виконання завдання з розгортання модуля API вебзастосування:

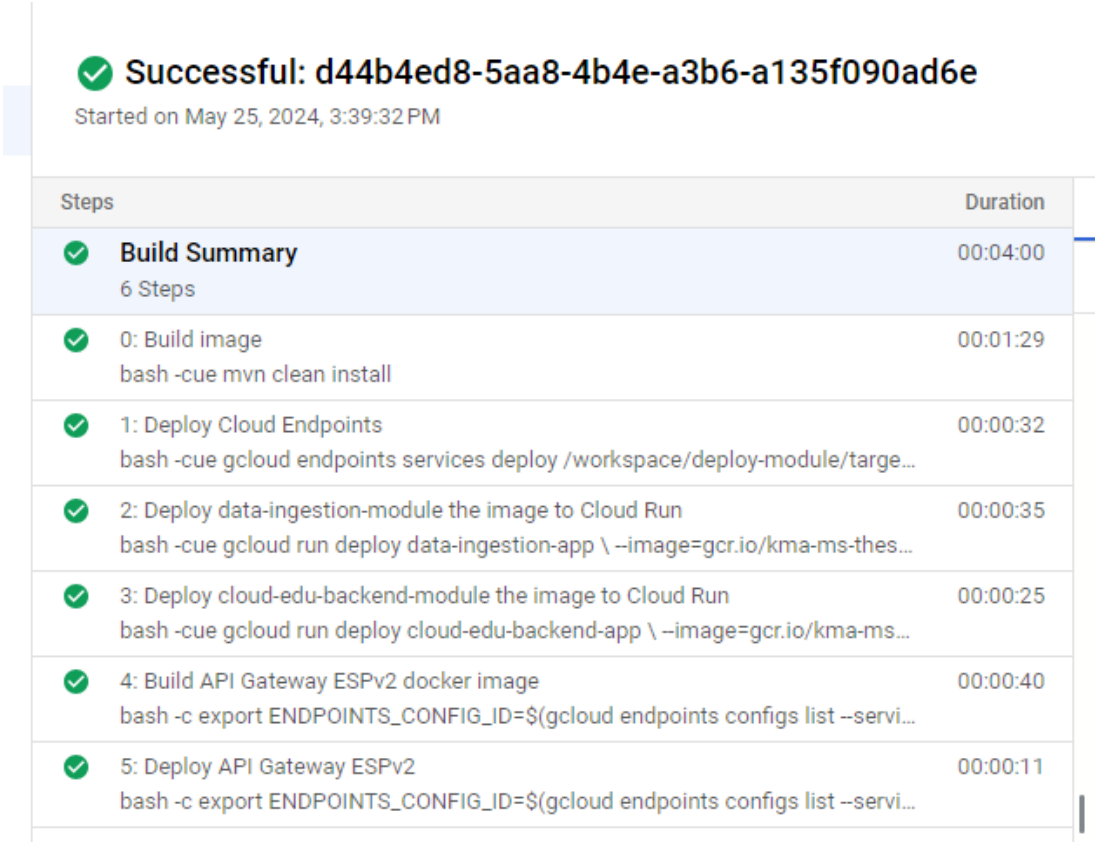


Рисунок 15 Успішний білд в CloudBuild

Загалом, процес безперервного розгортання для проєкту cloud-edu-back складається з 6 кроків:

1) На першому кроці збирач проєктів Maven компілює та інсталює код 4 модулів у визначеному порядку, а також запускає допоміжні плагіни, такі

як jib-maven-plugin(для створення образу докер контейнерів з jar файлів та завантаженню їх на сервіс Artifact Registry) та swagger-maven-plugin(для генерації swagger yaml файлу)

2) На другому кроці згенерований на попередньому кроці swagger yaml файл з OpenAPI специфікацією об'єднаного API обох модулів розгортається на сервіс Cloud Endpoints
3) На третьому кроці на Cloud Run розгортається побудований плагіном jib докер контейнер модуля data-ingestion-module з Artifact Registry
4) На четвертому кроці на Cloud Run розгортається побудований плагіном jib докер контейнер cloud-edu-backend-module з Artifact Registry
5) На п'ятому кроці запускається shell скрипт, який створює докер образ для контейнера шлюзу API ESPv2 і завантажує його на Artifact Registry
6) І на останньому кроці на Cloud Run розгортається сам шлюз API ESPv2 з образу, створеного в попередньому кроці

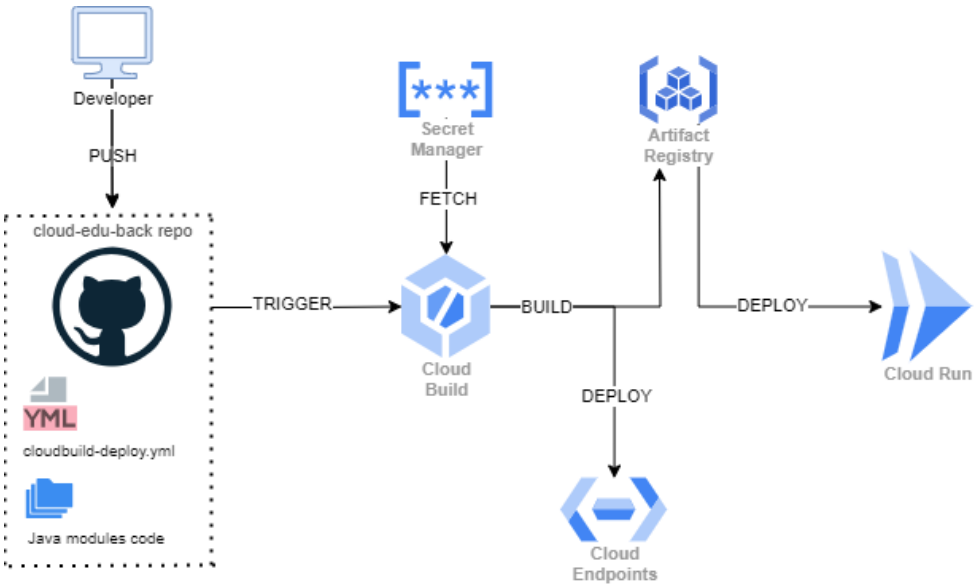
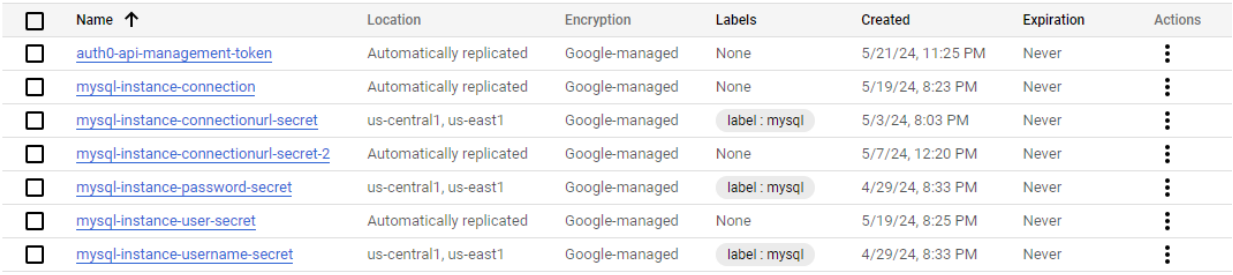


Рисунок 16 Схема розгортання нової версії API з проєкту cloud-edu-back

Для більшої захищеності конфігурацій у зв'язці з Cloud Build також було використано сервіс Secret Manager. В менеджері секретів, зокрема,

зберігаються такі чутливі дані як назва підключення до бази даних CloudSQL, логін та пароль до CloudSQL бази, токен для auth0 менеджмент API, а також різні конфігураційні файли, вилучені з публічного репозиторію GitHub.



Name	Location	Encryption	Labels	Created	Expiration	Actions
auth0-api-management-token	Automatically replicated	Google-managed	None	5/21/24, 11:25 PM	Never	
mysql-instance-connection	Automatically replicated	Google-managed	None	5/19/24, 8:23 PM	Never	
mysql-instance-connectionurl-secret	us-central1, us-east1	Google-managed	label : mysql	5/3/24, 8:03 PM	Never	
mysql-instance-connectionurl-secret-2	Automatically replicated	Google-managed	None	5/7/24, 12:20 PM	Never	
mysql-instance-password-secret	us-central1, us-east1	Google-managed	label : mysql	4/29/24, 8:33 PM	Never	
mysql-instance-user-secret	Automatically replicated	Google-managed	None	5/19/24, 8:25 PM	Never	
mysql-instance-username-secret	us-central1, us-east1	Google-managed	label : mysql	4/29/24, 8:33 PM	Never	

Рисунок 17 Сервіс Secret Manager

3.4. Рівень відображення

Код застосунку рівня відображення, або іншими словами фронтенду, написаний на фреймворку React з використанням мови програмування JavaScript. Для створення сучасного та зручного інтерфейсу використовуються компоненти з бібліотеки Material UI, що дозволяє швидко розробляти прості та функціональні інтерфейси. Для обміну даними з бекендом застосунку використовується бібліотека Axios, яка забезпечує зручний і ефективний спосіб роботи з API.

Передня частина системи для дистанційного навчання розгорнута на сервісі Google App Engine. Завдяки простоті та зручності цього сервісу, процес безперервного розгортання (CD) є дуже легким і не вимагає складних налаштувань. Пайплайн для CD фронтенду складається всього з трьох кроків:

1) npm install – команда для встановлення всіх необхідних залежностей
2) npm run build - команда для збирання застосунку в оптимізований для production-середовища формат
3) gcloud app deploy - команда для розгортання зібраного застосунку на App Engine

Для налаштування середовища виконання в App Engine, у теці проєкту має бути конфігураційний файл app.yaml, у якому визначаються тип інстансу, тип середовища та змінні оточення. Приклад конфігураційного файлу виглядає так:

```yaml
runtime: nodejs20
env: standard
instance_class: F1
env_variables:
  REACT_APP_API_ENDPOINT: "https://espv2-cloud-run-gqvd3vvqha-uc.a.run.app"
automatic_scaling:
  target_cpu_utilization: 0.65
  min_instances: 1
  max_instances: 1

handlers:
  - url: /static
    static_dir: build/static

  - url: /(.*\.(json|ico|js|txt))
    static_files: build/\1
    upload: build/.*\.(json|ico|js|txt)

  - url: /.*
    static_files: build/index.html
    upload: build/index.html
```

Рисунок 18 Конфігураційний файл для App Engine app.yaml

3.5. Рівень бізнес-логіки

Рівень бізнес-логіки є центральним компонентом системи, який відповідає за надання основного функціонала через програмний інтерфейс API. Цей рівень по суті являється головним, бо надає програмний інтерфейс з основним функціоналом системи й містить декілька незалежних сервісів: Cloud Edu Backend Service, Data Ingestion Service і Task Manager Service.

Всі сервіси рівня бізнес-логіки написані мовою програмування Java 17 з використанням вебфреймворку Spring Boot 3. Налаштування здійснюються через файли application.properties, причому більшість конфігураційних параметрів береться зі змінних оточення для підвищення безпеки. Ці змінні визначаються на рівні Cloud Build, витягуються з Google Secret Manager і

включаються в Docker-образи сервісів під час побудови maven плагіном jib. Це забезпечує конфіденційність та безпеку конфігураційних даних на всіх етапах розгортання.

Сервіси Cloud Edu Backend і Data Ingestion є повністю незалежними, в той час, як сервіс Task Manager в основному поєднується із сервісом Data Ingestion. Комунікація між ними відбувається за допомогою протоколу HTTPS, а запити авторизуються через identity token, який розпізнає Cloud IAM.

Оскільки сервіси виконують роль RESTful API, програмний інтерфейс автоматично документується за допомогою OpenAPI та SwaggerUI, що забезпечує зручний доступ до документації для розробників.

Статистику по API можна переглянути через сервіс Cloud Endpoints:

Select columns: Requests, 4xx, 5xx, Latency median, Avg resp size, and Avg req size

Method	Requests ↓	4xx	5xx	Latency median	Avg resp size	Avg req size	Links
GET /api/v1/files	250	8	38	59 ms	1447 B	2462 B	View logs
GET /api/v1/tasks	111	0	6	125 ms	1820 B	2473 B	View logs
GET /api/v1/schools/{school_id}/students/paginate	75	4	0	437 ms	2824 B	2584 B	View logs
POST /api/v1/schools/requests/new	65	12	27	105 ms	633 B	1082 B	View logs
GET /api/v1/tasks/{task_id}	30	3	0	415 ms	1061 B	2485 B	View logs
POST /api/v1/files	19	2	0	498 ms	841 B	187267 B	View logs
GET /api/v1/private	15	2	0	18 ms	651 B	1846 B	View logs
POST /api/v1/tasks	11	1	0	1835 ms	804 B	2583 B	View logs
DELETE /api/v1/files/{filename}	10	0	0	218 ms	611 B	2544 B	View logs
GET /api/v1/files/{filename}	9	0	0	426 ms	1427 B	2579 B	View logs

Rows per page: 10 1 – 10 of 32

Рисунок 19 Статистика API в сервісі Cloud Endpoints

3.6. Рівень доступу до даних

Рівень доступу до даних охоплює сервіси GCP Firestore, CloudSQL та Cloud Storage.

Доступ до Firestore здійснюється через модуль Cloud Edu Backend за допомогою бібліотеки від Google spring-cloud-gcp-data-firestore. Для доступу

достатньо мати налаштований сервісний акаунт з правами на запис та читання з Firestore, а також вказану конфігурацію в програмі, яка визначає, до якої бази потрібно підключатися. База визначається залежно від школи, що дозволяє кожній школі мати власну ізольовану базу даних, забезпечуючи високий рівень безпеки та конфіденційності даних.

CloudSQL використовується сервісами Data Ingestion та Task Manager. База даних знаходиться в приватному VPC, що забезпечує високий рівень захисту даних. Доступ до неї здійснюється лише через Cloud Run сервіси з налаштованими Serverless VPC Access конекторами. Це дозволяє обмежити доступ до бази даних тільки авторизованими сервісами, мінімізуючи ризики несанкціонованого доступу та забезпечуючи високу продуктивність і масштабованість системи.

Cloud Storage слугує як основне сховище для CSV файлів, або конфігураційних файлів. Також там зберігаються всілякі допоміжні Cloudbuild артефакти, статичний контент для фронтенду, та згенеровані темплейти Apache Beam пайплайнів.

3.7. Розробка пайплайну для збору даних

Пайплайн для збору даних теж можна визначати як частину рівня доступу до даних, оскільки напряму відповідає за наповненість даними системи. Для розробки пайплайну використано уніфіковану модель Apache Beam і SDK для мови програмування Java 17. Всього реалізовано два пайплайни: MySqlToFirestore та CsvToFirestorePipeline. Ось код методу пайплайну MySqlToFirestorePipeline, який реєструє основні кроки пайплайну для завантаження даних із клієнтської MySql бази в кінцеву базу на Firestore:

```java
@Override
public void registerSteps() {
    log.info("Starting pipeline MySqlToFirestorePipeline");
    final DataSourceProvider mySqlDataSourceProvider = DataSourceProvider.builder()
            .dbConnection(options.getMySqlConnectionUrl())
            .username(options.getMySqlUsername())
            .password(options.getMySqlPassword())
            .dataSourceVendor(DataSourceVendor.MYSQL)
            .build();
    final MySqlModelContainer modelContainer = new MySqlModelContainer();
    final SerializableSupplier<SqlEntityRowMapper<? extends BaseMySqlModel>> rowMapperSupplier =
            MySqlModelUtil.createSqlEntityRowMapperSupplier(options.getEntityName(), modelContainer);

    // Read JSON data from a file
    final PCollection<String> jsonLines = pipeline
            .apply( name: "Read JSON File", FileIO.match().filepattern(options.getMappingLocation())) PCollection<Metadata>
            .apply( name: "Read Whole File", FileIO.readMatches()) PCollection<ReadableFile>
            .apply( name: "Convert to String", ParDo.of(new WholeFileReader()));

    // Parse and deserialize JSON data into Mapping objects
    final PCollection<Mapping> mapping = jsonLines.apply(ParDo.of(new JsonToMappingFn()));

    final PCollection<? extends BaseMySqlModel> entities = mapping
            .apply( name: "Fetching data from MySql",
                    ParDo.of(new FetchSqlDataFunction<>(mySqlDataSourceProvider, rowMapperSupplier)));

    // Transform data and write to Firestore in batches
    entities.apply( name: "Map to KV", ParDo.of(new MakeBatchers())) PCollection<KV<...>>
            .apply( name: "Group into Batches", GroupIntoBatches.ofSize(BATCH_SIZE)) PCollection<KV<...>>
            .apply( name: "Write Batches to Firestore",
                    ParDo.of(new WriteToFirestoreFn(StaticValueProvider.of(options.getProject()),
                            options.getDatabaseId(),
                            options.getFirestoreCollectionName())));

}
```

Рисунок 20 Код пайплайну MySqlToFirestorePipeline

На вхід пайплайн отримує декілька параметрів, серед яких шлях до JSON файлу з конфігурацією доступу до клієнтської бази з мапінгом полів, засекречений логін та пароль до клієнтської бази, назва кінцевої бази даних школи. Дані викачані з MySql завантажуються на Firestore батчами по 1000 входжень, проте це значення також можливо кастомізувати.

```
{
  "sourceVendor": "MYSQL",
  "query": "SELECT * FROM edu_database.students",
  "sourceTable": "students",
  "connectionInfo": {
        "username" : "<username>",
        "password": "<password>",
        "host": "<host>",
        "port": 3306,
        "database": "edu_database"
  },
  "tableMappings": {
    "INTERNAL_ID": {
      "name": "id",
      "type": "INT"
    },
    "EMAIL": {
      "name": "email",
      "type": "VARCHAR"
    },
    "FIRST_NAME": {
      "name": "first_name",
      "type": "VARCHAR"
    },
    "LAST_NAME": {
      "name": "last_name",
      "type": "VARCHAR"
    },
    "CITY_OF_BIRTH": {
      "name": "city_of_birth",
      "type": "VARCHAR"
    },
    "GRADE": {
      "name": "grade",
      "type": "INT"
    }
  }
}
```

Рисунок 21 Приклад конфігураційного JSON файлу для MySqlToFirestorePipeline

Для того, щоб ці пайплайни можна було стартувати в Dataflow, потрібно згенерувати відповідні темплейти під цей сервіс. Після успішної збірки jar-файлу, його потрібно виконати з указанням базових параметрів темплейту, таких як: тип пайплайна, ранер, тип та кількість воркерів, локації для тимчасових файлів і файлу готового темплейту в Cloud Storage.

```
- id: "Create Dataflow Template for MySqlToFirestorePipeline"
  name: "gcr.io/cloud-builders/mvn"
  entrypoint: "bash"
  args:
    - -c
    - |
      java -cp ".:/workspace/target/${REPO_NAME}-$(cat /workspace/version.txt).jar" \
      com.nakytniak.PipelineStarter \
      --pipelineType=MYSQL_TO_FIRESTORE \
      --region=us-central1 \
      --runner=DataflowRunner \
      --project=$PROJECT_ID \
      --enableStreamingEngine=true \
      --diskSizeGb=40 \
      --workerMachineType=n1-standard-1 \
      --maxNumWorkers=2 \
      --stagingLocation=gs://${_DATAFLOW_ARTIFACTS_BUCKET}/$REPO_NAME/${BRANCH_NAME}${TAG_NAME}/staging/mysql-to-firestore-pipeline \
      --tempLocation=gs://${_DATAFLOW_TEMP_BUCKET}/$REPO_NAME/${BRANCH_NAME}${TAG_NAME}/temp \
      --templateLocation=gs://${_DATAFLOW_ARTIFACTS_BUCKET}/$REPO_NAME/${BRANCH_NAME}${TAG_NAME}/template/mysql-to-firestore-pipeline \
      --filesToStage=/workspace/target/${REPO_NAME}-$(cat /workspace/version.txt).jar
```

Рисунок 22 Команда для створення темплейту пайплайну

Після того, як Cloudbuild згенерує темплейти пайплайнів і збереже їх у Cloud Storage, вони стануть доступними для виконання в Dataflow. Їх можливо запускати як вручну через інтерфейс GCP, так і програмним шляхом. В системі дистанційного навчання передбачено створення пайплайну адміністратором через інтерфейс вебзастосунка Для цього спершу потрібно завантажити CSV файл, або файл у форматі JSON з конфігурацією.

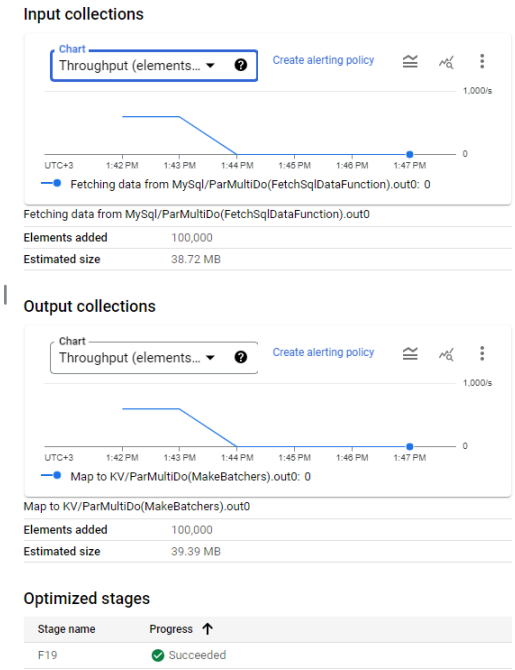
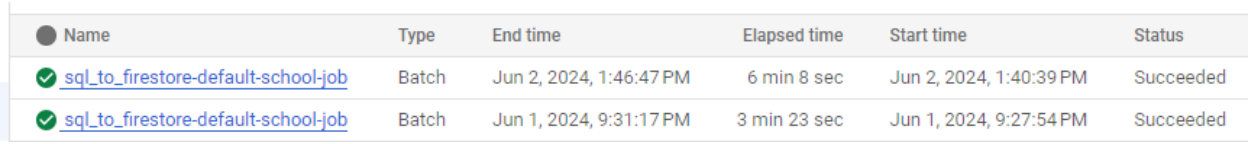


Рисунок 23 Тестування виконання пайплайну на 100000 даних

Ось як виглядає успішне виконання одного з таких пайплайнів на хмарному сервісі Dataflow:



Name	Type	End time	Elapsed time	Start time	Status
sql_to_firestore-default-school-job	Batch	Jun 2, 2024, 1:46:47 PM	6 min 8 sec	Jun 2, 2024, 1:40:39 PM	Succeeded
sql_to_firestore-default-school-job	Batch	Jun 1, 2024, 9:31:17 PM	3 min 23 sec	Jun 1, 2024, 9:27:54 PM	Succeeded

Рисунок 24 Приклад успішного виконання пайплайну в сервісі Dataflow

Додатково, на зображенні з **додатка Д** можна побачити ті самі кроки, які було визначеними в коді програми темплейту, але які вже розпізнав і успішно виконав сервіс Dataflow. Результатом роботи пайплайну є заповнена колекція даними студентів в базі даних Firestore.

Висновки

В ході роботи було розроблено багаторівневе вебзастосування з високою доступністю на хмарній платформі GCP для дистанційного навчання в школах. В першому розділі було розглянуто наявні підходи для побудови сучасних вебзастосувань з використанням інструментів для роботи з даними, а саме для збору та обробки даних.

В другому розділі було проаналізовано та обрано інструменти й сервіси GCP, які найкраще підходять для розв’язання поставленої задачі. З використанням обраних інструментів та сервісів розроблено запропоновану архітектуру вебзастосування для дистанційного навчання в школах з акцентом на високу доступність. Архітектуру вебзастосунка було додатково проаналізовано на наявність вузьких місць, і прийнято рішення для усунення недоліків. Зокрема, було додано балансувальник навантаження для контролю над трафіком, використано хмарний CDN сервіс для додаткового кешування статичного контенту, додано чергу повідомлень для зменшення одночасного навантаження на сервіси, а також використано проксі-сервер шлюзу API для підвищення ефективності бекенд сервісів.

У третьому розділі роботи було описано процес побудови інфраструктури на хмарній платформі GCP, необхідної для функціонування вебзастосування та розробки самого застосування. Кінцевим доробком стала розроблена мінімальна версія вебзастосунка з робочими модулями фронтенду, бекенду і модулем для роботи з даними. Налагоджено необхідні CI/CD процеси, а також налаштовано інфраструктуру на GCP з використанням Terraform.

Використані джерела

1. // https://www.redhat.com: [сайт]. [2020]. URL: https://www.redhat.com/en/topics/cloud-native-apps/what-is-an-application-architecture#layered-or-n-tier (дата звернення: 10.Березень.2024).

2. // Medium: [сайт]. [2021]. URL: https://towardsdatascience.com/what-is-data-ingestion-5220edf50677 (дата звернення: 10.Квітень.2024).

3. Šprem Š.&.T.N.&.M.J.&.H.M., "Building Advanced Web Applications Using Data Ingestion and Data Processing Tools," // Electronics, Vol. 13, No. 10.3390/electronics13040709, Лютий 2024. P. 709.

4. // Medium: [сайт]. [2023]. URL: https://amsayed.medium.com/the-various-methods-of-change-data-capture-cdc-with-examples-and-code-snippets-e5d6ea14dc88 (дата звернення: 10.Квітень.2024).

5. // Medium: [сайт]. [2021]. URL: https://towardsdatascience.com/change-data-capture-cdc-for-data-ingestion-ca81ff5934d2 (дата звернення: 10.Квітень.2024).

6. // Medium: [сайт]. [2021]. URL: https://medium.com/israeli-tech-radar/building-a-cdc-lessons-learned-part-3-18921b74e216 (дата звернення: 10.Квітень.2024).

7. // Google Cloud: [сайт]. [2024]. URL: https://cloud.google.com/data-movement (дата звернення: 20.Квітень.2024).

8. // Google Cloud: [сайт]. [2024]. URL: https://cloud.google.com/bigquery/docs/dts-introduction (дата звернення: 25.Квітень.2024).

9. // Google Cloud: [сайт]. [2024]. URL: https://cloud.google.com/dataflow/docs/concepts/beam-programming-model (дата звернення: 2.Травень.2024).

10. // Google Cloud: [сайт]. [2023]. URL: https://cloud.google.com/architecture/framework/reliability/design-scale-high-availability (дата звернення: 20.Травень.2024).

11. // Google Cloud: [сайт]. [2024]. URL: https://www.googlecloudcommunity.com/gc/Community-Blogs/Navigating-the-Cloud-API-Maze-Apigee-API-Gateway-or-Cloud/ba-p/701627 (дата звернення: 20.Травень.2024).

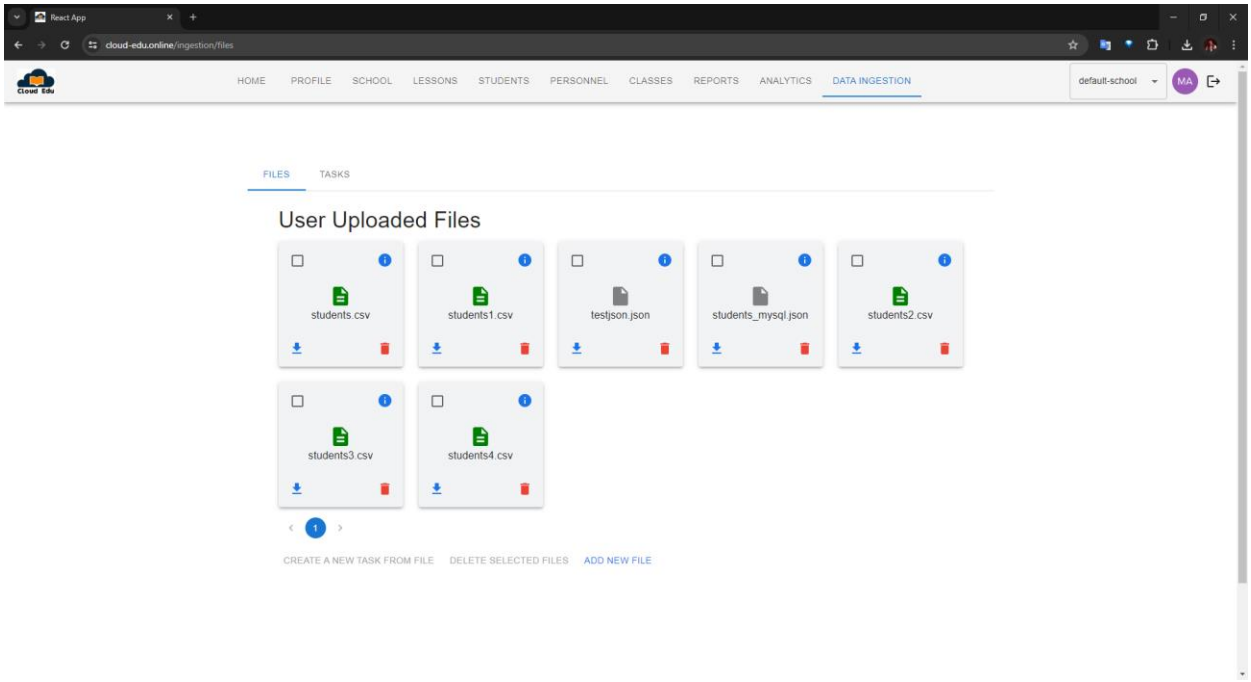
12. // Google Cloud: [сайт]. [2024]. URL: https://cloud.google.com/endpoints/docs/openapi/set-up-cloud-run-espv2 (дата звернення: 20.Квітень.2024).

13. // Medium: [сайт]. [2022]. URL: https://medium.com/@larry_nguyen/comparing-terraform-and-cloud-deployment-manager-3e7b56f9de7b (дата звернення: 25.Травень.2024).

Додаток А

(обов’язковий)

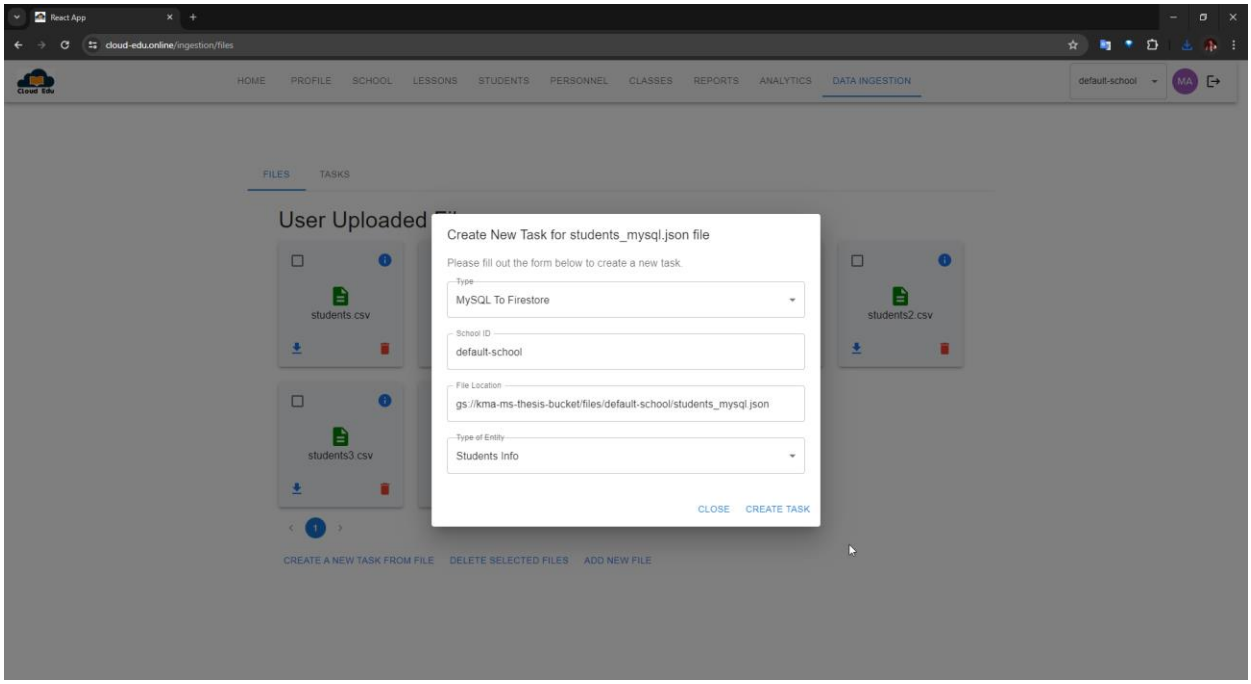
Сторінка інтерфейсу адміністратора для завантаження файлів конфігурацій



Додаток Б

(обов’язковий)

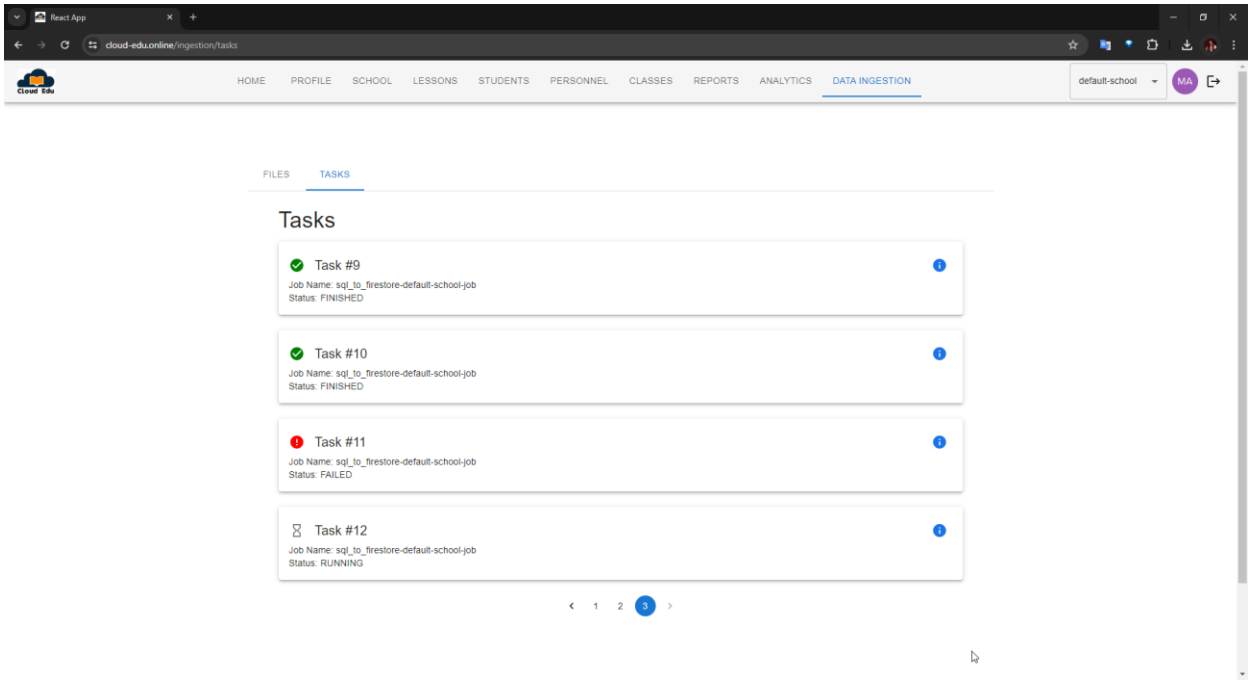
Форма для створення завдання з завантаження даних



Додаток В

(обов’язковий)

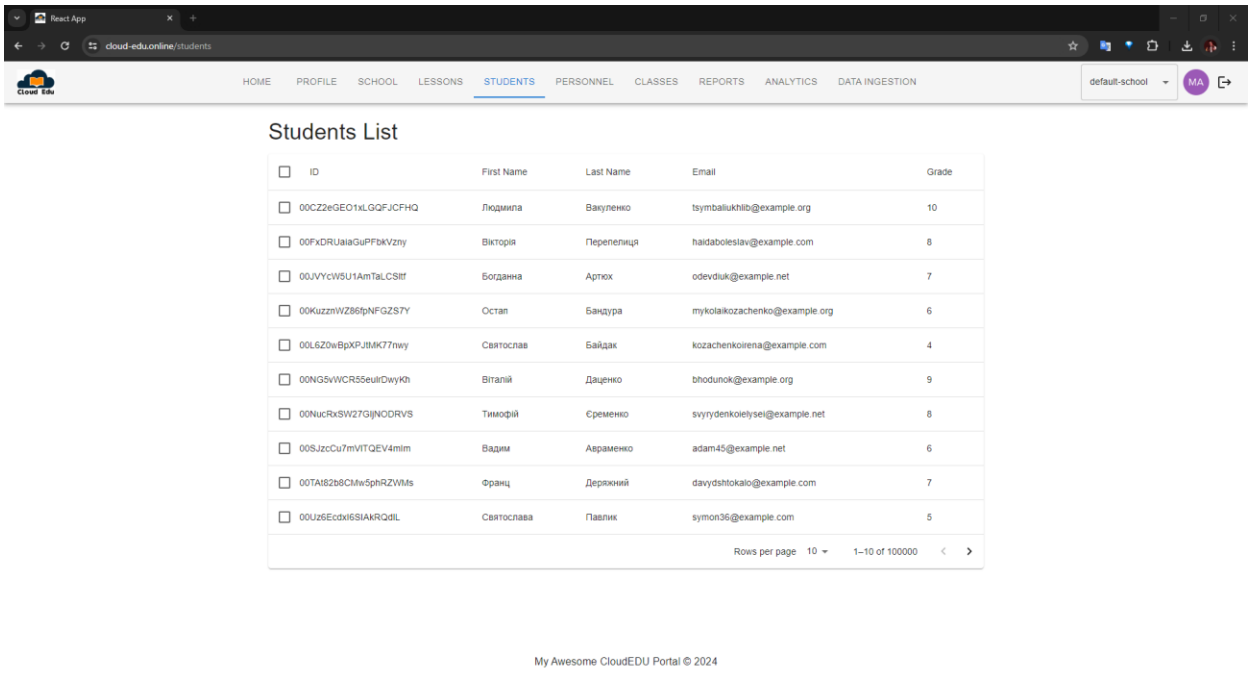
Сторінка інтерфейсу адміністратора з поточними завданнями



Додаток Г

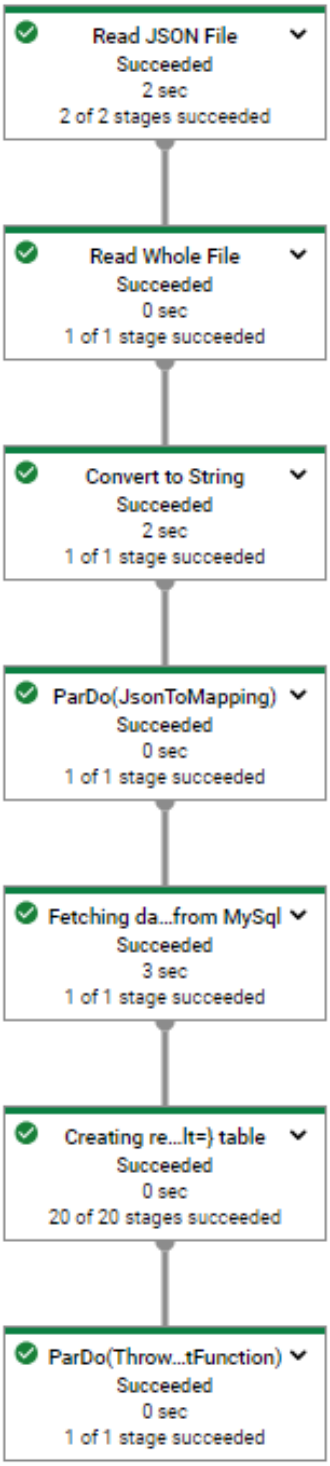
(обов’язковий)

Сторінка інтерфейсу адміністратора із завантаженими даними про студентів



Додаток Ґ

(обов'язковий)



Додаток Д

(обов’язковий)

```
#!/bin/bash

sudo apt-get update -y

sudo apt-get install -y mysql-server python3 python3-pip git

echo "Cloning repo for generating data"

git clone https://github.com/brutovsky/ideal-robot.git /home/${var.username}/data

echo "Starting mysql"

sudo systemctl start mysql

# Run python Faker script

echo "Running faker script"

pip install faker

python3 /home/${var.username}/data/generate_students.py
${local.size_of_test_dataset} /home/${var.username}/data/students_info.csv

# Retrieve secret values using `gcloud` CLI

password=$(gcloud secrets versions access latest --secret="mysql-instance-
password-secret" --format="value(payload.data)")

# Run MySQL commands to create user and grant privileges

echo "Running MySQL commands to create user and grant privileges"

sudo mysql -e "ALTER USER 'root'@'localhost' IDENTIFIED WITH
mysql_native_password BY '$password'; FLUSH PRIVILEGES;"
```

```
sudo mysql -u root -p"$password" -e "SET GLOBAL local_infile = 'ON';"

sudo mysql -u root -p"$password" -e "SHOW VARIABLES LIKE 'local_infile';"

# Run initial MySql script

echo "Running initial mysql script"

cat /home/${var.username}/data/init.sql

sudo mysql --local-infile=1 -u root -p"$password" <
/home/${var.username}/data/init.sql

echo "Creating new user in mysql"

sudo mysql -u root -p"$password" -e "CREATE USER 'vadym'@'%' IDENTIFIED
WITH mysql_native_password BY '$password';"

sudo mysql -u root -p"$password" -e "GRANT ALL PRIVILEGES ON *.* TO
'vadym'@'%' IDENTIFIED BY '$password' WITH GRANT OPTION;"

sudo mysql -u root -p"$password" -e "GRANT ALL PRIVILEGES ON
edu_database.* TO 'vadym'@'%' WITH GRANT OPTION;"

sudo mysql -u root -p"$password" -e "FLUSH PRIVILEGES;"

echo "Finished running MySQL commands to create user and grant privileges"

# Restart MySQL service to apply changes

sudo sed -i 's/^bind-address\s*=.*/bind-address = 0.0.0.0/'
/etc/mysql/mysql.conf.d/mysqld.cnf && sudo systemctl restart mysql
```