

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»**

Кафедра мережних технологій факультету інформатики

**Побудова багаторівневого веб-застосування
на платформі Google Cloud Platform**

Текстова частина до курсової роботи
за спеціальністю «Комп'ютерні науки та інформаційні технології» - 122

Керівник курсової роботи

к.т.н., ст. викладач Черкасов Д. І.

(підпис)

“ ____ ” _____ 2021 р.

Виконав студент 4 курсу

Куручкін І. Є.

“ ____ ” _____ 2021 р.

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ РОБОТИ

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання теми курсової роботи.	15.10.2020	
2.	Ознайомлення з техніками побудови безсерверних застосунків та використання хмарних сервісів	30.10.2020	
3.	Порівняння існуючих рішень для хмарних обчислень, підбір інструментарію для реалізації власного рішення	15.02.2021	
4.	Написання перших двох розділів	20.02.2021	
5.	Реалізація компоненту бек-енду	05.03.2021	
6.	Розгортання компоненту бек-енду та моделювання компоненту користувацького інтерфейсу	14.03.2021	
7.	Створення презентації та написання доповіді для захисту роботи.	22.03.2021	
9.	Корегування роботи згідно із зауваженнями керівника	31.03.2021	

Зміст

1. ВСТУП.....	4
2. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ.....	6
2.1. Традиційна ІТ інфраструктура.....	6
2.2. Хмарні обчислення.....	7
2.2.1. Amazon Web Services	8
2.2.2. Microsoft Azure	10
2.2.3. Google Cloud Platform	11
3. РОЗРОБКА ВЛАСНОГО ЗАСТОСУВАННЯ	13
3.1. Архітектура застосування	15
3.2. Компоненти застосування	17
3.2.1. База даних	17
3.2.2. Бекенд	20
3.2.3. GraphQL API	22
3.2.4. Автентифікація та авторизація	25
3.2.5. Сервіс продуктів	28
3.2.6. Сервіс категорій.....	30
3.2.7. Сервіс замовлень	31
3.2.8. Клієнтське застосування.....	32
3.3. Процес розробки застосування	36
4. ВИСНОВКИ.....	40
5. Список використаної літератури.....	42

1. ВСТУП

Сучасні мережеві застосування є складними програмними системами, які розробляються у співпраці групами розробників. Головними вимогами до застосувань є гнучкість, масштабованість, висока доступність та здатність витримувати навантаження. Для ефективного розподілу задач між розробниками і виконання зазначених вимог застосування проектуються згідно багаторівневої архітектури. Найпоширеніші такі базові рівні:

- інтерфейс користувача (frontend),
- бізнес логіка (middleware),
- сервер з базою даних (backend).

Крім цього існує багато проміжного програмного забезпечення для виконання підтримки існуючого застосунку, покращення ефективності за рахунок додаткових оптимізацій.

Окрім розробки коду багаторівневого застосування виникає ряд додаткових задач. До них належить:

- розгортання та налаштування зв'язків між компонентами,
- оновлення коду та даних,
- оновлення структури шляхом зміни складу використовуваних компонентів та зв'язків між ними,
- моніторинг показників ефективності функціонування.
- налаштування процесу розгортання та налаштування зв'язків між рівнями, здійснення обслуговування існуючого додатку .

Попри це потрібно підтримувати базу даних, сховище інформації про користувачів для авторизації, та інші сервіси. Ефективно перелічені задачі

допомагають хмарні платформи. Вони об'єднують значну кількість сервісів, які можна легко підключати до свого проекту, як окремі модулі.

Метою моєї курсової роботи є побудова багаторівневого мережевого застосування на прикладі онлайн-магазину комп'ютерних компонентів на хмарній платформі Google Cloud Platform (GCP). GCP — це сучасна хмарна платформа, до переваг якої належить швидкість роботи, простота налаштувань та зручність у масштабуванні.

В умовах сучасної міграції торгівельних послуг в Інтернет, а також завдяки перевагам онлайн-сервісів, особливо під-час карантинних обмежень, набувають популярності мережеві сайти електронної комерції. Тому робота з розробки застосувань на зразок онлайн-магазину комп'ютерних компонентів наразі є актуальною.

Основні вимоги до застосування, що розробляється:

- багаторівнева архітектура
- наявність системи рекомендацій комп'ютерних деталей на основі сумісності товарів
- динамічна масштабованість відповідно до рівня активності користувачів,
- оновлення даних та коду без переривання роботи сервісу
- високий ступінь захисту приватних даних.

2. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Існують різні підходи до побудови веб застосування. Найпопулярнішими зараз є побудова на хмарних сервісах та розробка з використанням традиційної ІТ інфраструктури. Далі будуть розглянуті переваги та недоліки різних підходів.

2.1. Традиційна ІТ інфраструктура

Традиційні дата центри складаються з різних апаратних засобів. Таких як настільний комп'ютер, що є підключеним до мережі через віддалений сервер. Як правило, його розміщують у приміщенні компанії. Визначеним працівникам надають доступ до даних та застосунків, що містяться на сервері. Ті підприємства, що використовують традиційний підхід до ІТ інфраструктури повинні придбати додаткове фізичне обладнання та оновлення, у випадку, коли потрібно збільшити сховище або налаштувати підтримку більшої кількості користувачів.

Іншою особливістю традиційного підходу є потреба у контролі оновлень програмного забезпечення. Це необхідно для забезпечення кращої надійності та відмово стійкості. Багато підприємств, що користуються таким підходом, також містять спеціальний ІТ-відділ для встановлення та обслуговування апаратного забезпечення.

З іншого боку, традиційні ІТ-інфраструктури вважаються одними з найбільш захищених рішень для розміщення даних. Таким способом, можна досягти повного контролю над програмами та даними компанії на локальному сервері [1].

2.2. Хмарні обчислення

На противагу традиційному підходу з дорогим фізичним обладнанням, хмарні рішення містять сервіси, програми та мережі, що надають віртуальне середовище, яке розміщене на багатьох серверах одночасно.

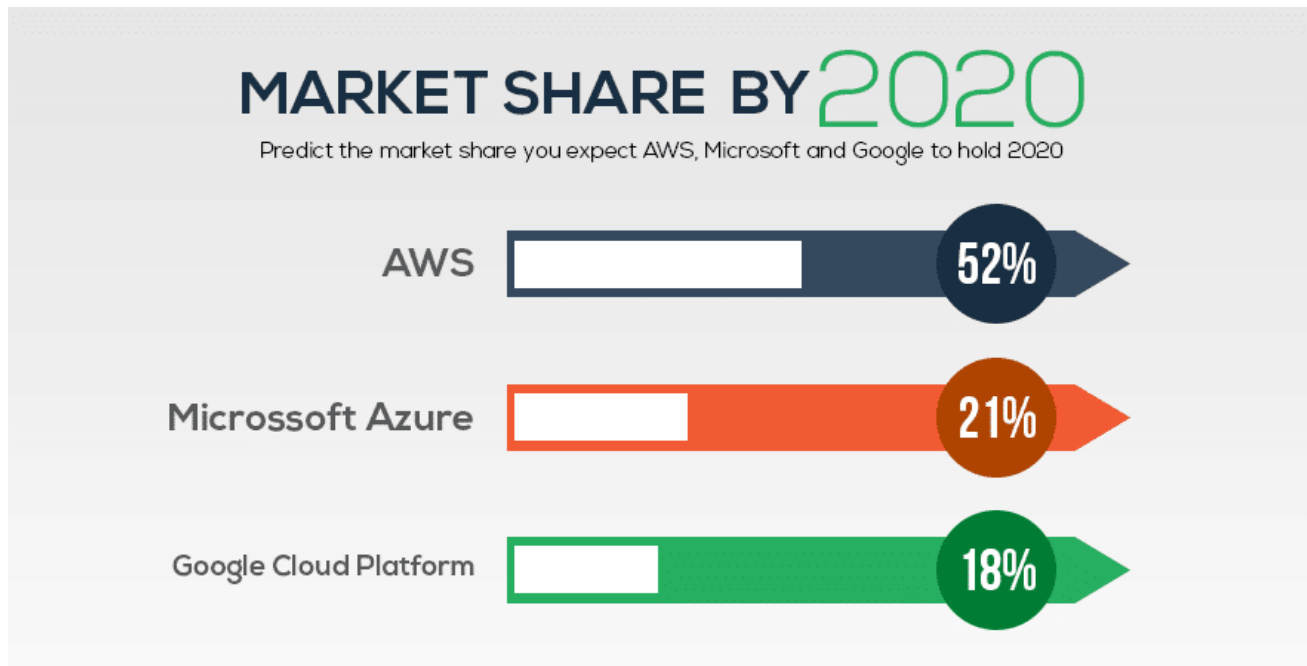
Це означає, що замість вкладення грошей в купівлю фізичного обладнання та його подальшого обслуговування, можна просто орендувати сховище та обчислювальні потужності. Оплата буде здійснюватися в залежності від використання.

Такий підхід є значно економніший з боку фінансів, через те, що відповідальність за надлишкове апаратне забезпечення, лежатиме не на підприємстві, а на хмарних провайдерах. Також значно зменшується кількість часу та коштів на обслуговування такої інфраструктури.

Ще однією зручністю хмарних рішень є можливість масштабування з незначними витратами по часу, та оптимальними витратами по коштах.

Загалом така простота дозволяє зменшити кількість працівників, необхідних для налаштування та обслуговування веб застосунків. Команди, які працюють з хмарними системами можуть залишатись краще сфокусованими на продукті, не витрачаючи багато часу на інфраструктуру.

Серед існуючих хмарних рішень найпопулярнішими є AWS (Amazon Web Services), Microsoft Azure та GCP (Google Cloud Platform). Загалом усі вони надають схожий базовий функціонал. Але чим же вони відрізняються?



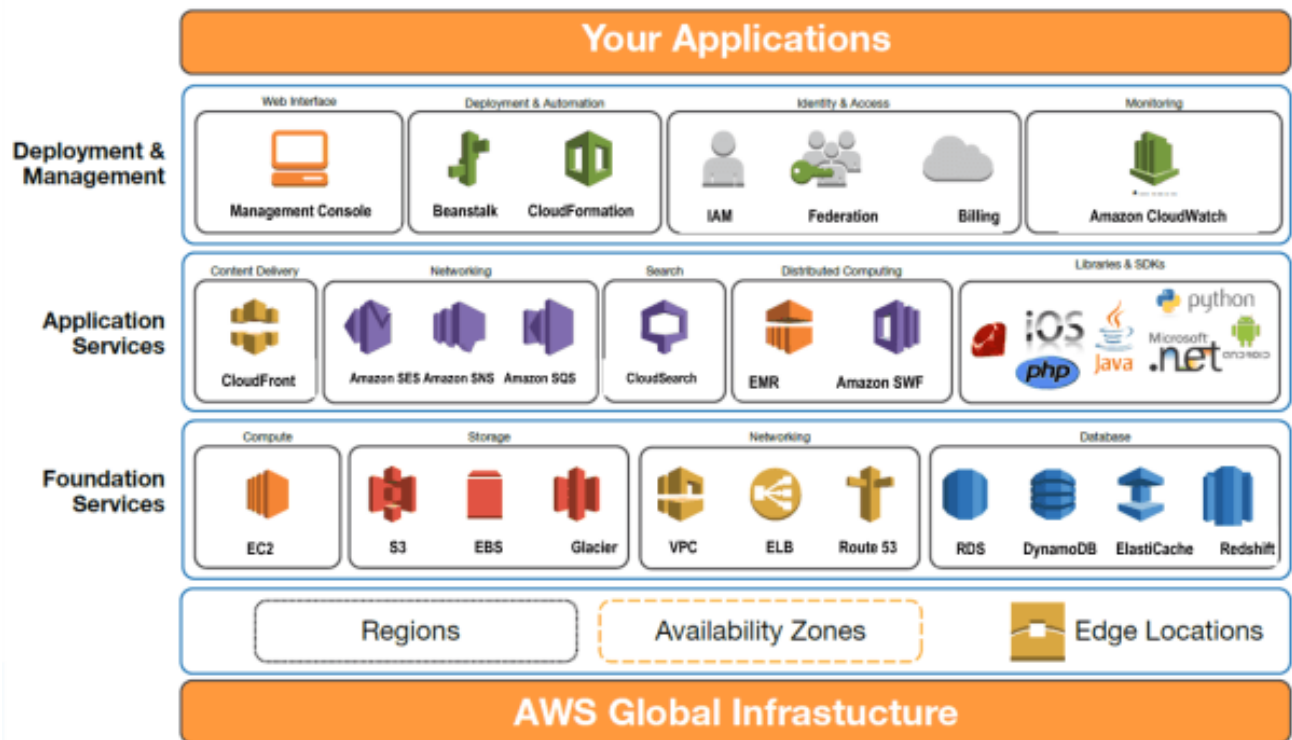
*Частки ринку хмарних сервісів станом на 2020 рік [2].
Рисунок 2.1*

2.2.1. Amazon Web Services

AWS є провідною платформою із 30% часткою на ринку хмарних провайдерів. Вона може похвастатись високою обчислювальною потужністю, послугами резервного копіювання, великим обсягом зберігання даних, та великою кількістю інших функцій для бізнес процесів та DevOps.

AWS є провідною платформою із 30% часткою на ринку хмарних провайдерів. Вона може похвастатись високою обчислювальною потужністю, послугами резервного копіювання, великим обсягом зберігання даних, та великою кількістю інших функцій для бізнес процесів та DevOps.

AWS володіє гібридною моделлю сховища через Storage Gateway, який зручно використовувати з функцією резервного копіювання Glacier. Платформа пропонує звичайне блочне сховище S3 або блочне сховище з E2B. Еластичне сховище файлів AWS розширюється зі швидкістю створення та додавання файлів.



Amazon Web Services Global Infrastructure [3].

Рисунок 2.2

Платформа AWS також пропонує сервіс для здійснення обчислень — Amazon Elastic Compute Cloud. Таке рішення є досить вигідним з економічного боку. Також можливе швидке масштабування залежно від бізнес потреб. Також існує сервіс для керування контейнерами — Amazon Elastic Container Service (Amazon ECS). AWS також має підтримку контейнерів Kubernetes.

Що стосується безпеки, AWS надає API для моніторингу активності, оцінки вразливості та налаштування firewall. Також можна з легкістю керувати захистом даних та правами доступу. Крім цього до AWS дозволяє фільтрувати трафік на основі визначених правил та проводити бенчмаркінг.

Продукти AWS:

- Elastic Compute Cloud
- Relational Database Service
- Amazon DynamoDB
- Amazon SimpleDB
- Simple Storage Service
- Amazon Elastic Block Store

- Amazon Elastic File System
- Elastic Load Balancer
- Direct Connect
- Amazon Route 53

2.2.2. Microsoft Azure

Microsoft Azure володіє до 16% ринку і є другою по популярності платформою. Вона пропонує повний набір рішень для вирішення щоденних задач при розробці та підтримці продукту. Надається хороша підтримка паралельних обчислень та масштабування за лічені хвилини. Більшість сервісів можуть бути легко інтегровані в існуючий проект.

Azure пропонує сховище, що називається Blob Storage. Його особливістю є те, що доступ до об'єктів базується на основі REST. Подібно до AWS, Azure має великий вибір баз даних з підтримкою SQL. Також існує підтримка попередніх хмарних функцій Microsoft SQL Server.

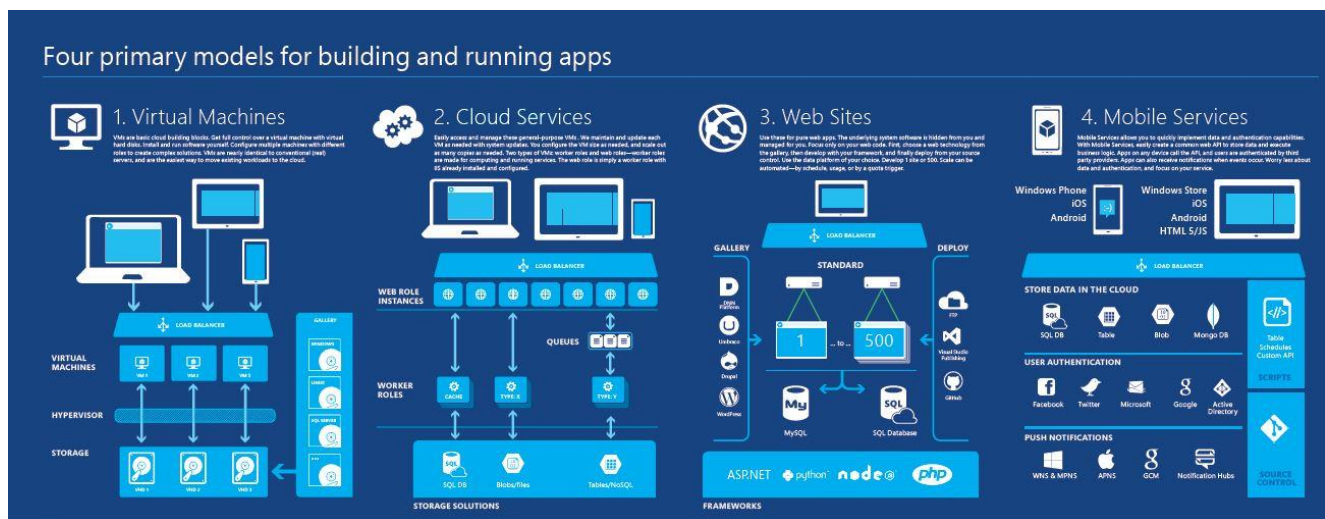


Схема сервісів та юзкейсів Microsoft Azure Platform [4].

Рисунок 2.3

Хмарні обчислення на MS Azure відбуваються на віртуальних машинах. На них виконуються процедури, від розгортання додатків, до розробки, тестування, та виконання обчислень для дата-центрів. Віртуальні машини сумісні з серверами Windows, Linux, SQL Servers та іншими. Також

можлива гібридна модель, яка поєднує обчислення на Azure з функціональністю інших хмарних сервісів. Існує рішення для швидкої контейнеризації, розгортання та управління програмами — Azure Kubernetes Service (AKS).

Що стосується безпеки, Azure Security Center відповідає за налаштування безпеки і надає доступ до моніторингу та логів. Контроль безпеки розділений на рівні та дозволяє налаштувати захист від перевищення навантажень, безпеку криптографічних ключів, електронних адрес та документів.

Продукти Amazon Web Services:

- Virtual Machines
- Azure Functions
- SQL Database
- App Service and Cloud Services
- Azure Kubernetes Service (AKS)
- Table Storage
- Azure Cosmos DB
- Disk Storage
- Blob Storage
- Azure File Storage
- Azure Archive Blob Storage
- Load Balancer
- ExpressRoute
- Azure DNS

2.2.3. Google Cloud Platform

Хмарна платформа від Google з'явилась дещо пізніше ніж AWS та MS Azure, і тому володіє не таким великим відсотком ринку. GCP може похвастатись можливостями сервісів для роботи з ML та AI. Серед інших особливостей цієї хмарної платформи - розгортання підводного сервера, зручна консоль.

GCP, як і попередньо розглянуті хмарні платформи надає хмарне сховище, реляційні та нереляційні бази даних. Google розробив стандарт

Kubernetes, який пізніше був реалізований Amazon Web Services та Microsoft Azure. Існує підтримка роботи з Docker контейнерами, надаючи функції для моніторингу продуктивності, автоматичної масштабованості, та запуску коду з Google Cloud, Assistant або Firebase.

GCP містить набір функцій для налаштування безпеки, які містяться в Cloud Security Control Center. Платформа побудована на захищеній архітектурі від апаратної інфраструктури до сховища та контейнерів Kubernetes. Відстежується навантаження, забезпечуючи цілодобовий моніторинг всіх елементів даних та каналів зв'язку.

Google Cloud містить велику кількість фізичних ресурсів серед яких сховища даних та віртуальні машини. Вони розподілені по дата центрах, що розміщені на усіх континентах. Також Google володіє великою кабельною системою, через що швидкість у користувачів Google Cloud та Google App може досягати 10 терабіт за секунду. Висока швидкість передачі та обробки даних дозволяє зменшити час, який потрібен для виконання задач та зберегти гроші. Похвилинна оплата, гнучка цінова політика та часті знижки роблять цей сервіс привабливим для побудови застосувань. Великою перевагою для розробників є якісна та детальна документація [5].

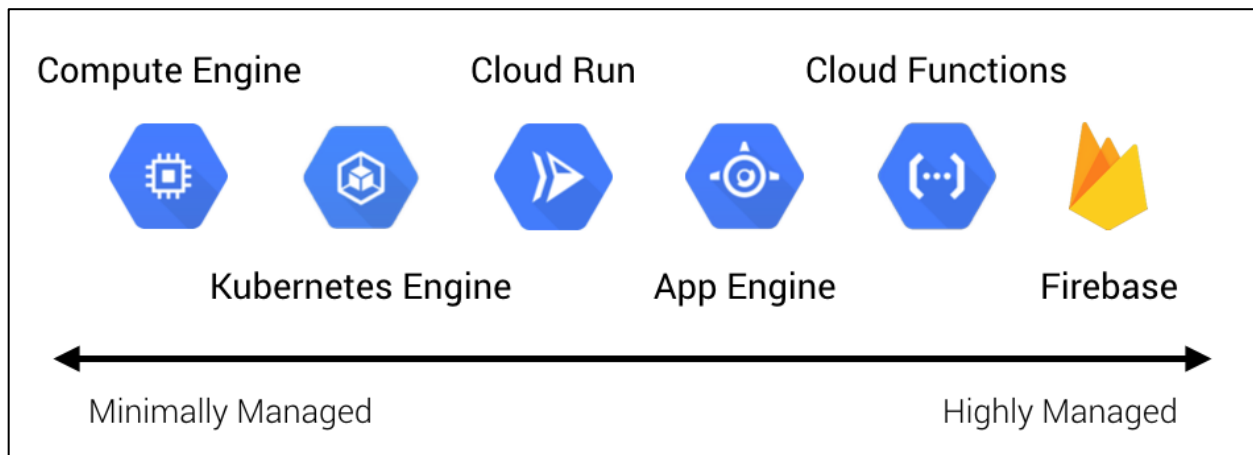
Сервіси GCP:

- Google Compute Engine
- Google App Engine
- Google Kubernetes Engine
- Google Cloud Functions
- Google Cloud SQL
- Google Cloud Datastore
- Google Cloud Bigtable
- Google Cloud Storage
- Virtual Private Cloud
- Google Cloud Load Balancing
- Google Cloud Interconnect
- Google Cloud DNS

3. РОЗРОБКА ВЛАСНОГО ЗАСТОСУВАННЯ

Для розробки власного застосування було вирішено використовувати мову програмування TypeScript. Цю мову можна застосувати, як для розробки бек-енду так і для написання клієнтської частини. Для взаємодії з сервером використовується мова запитів GraphQL. Дані зберігатимуться в нереляційній базі даних. Серед вимог до застосування є також розділення рівнів доступу.

Архітектуру застосування було вирішено побудувати на основі Serverless підходу. Безсерверні обчислення - це модель виконання хмарних обчислень, у якій хмарний провайдер відіграє роль сервера, динамічно керуючи розподілом машинних ресурсів. Ціна за використання сервісу базується на фактичній кількості ресурсів, задіяних в обчисленні. Це означає, що не потрібно оплачувати попередньо придбані обчислювальні ресурси [6].



Сервіси для хмарних обчислень Google [7]

Рисунок 3.1

Основними характеристиками безсерверних обчислень є:

- Абстракція,
- Еластичність,
- Ефективна вартість,
- Обмежений життєвий цикл.

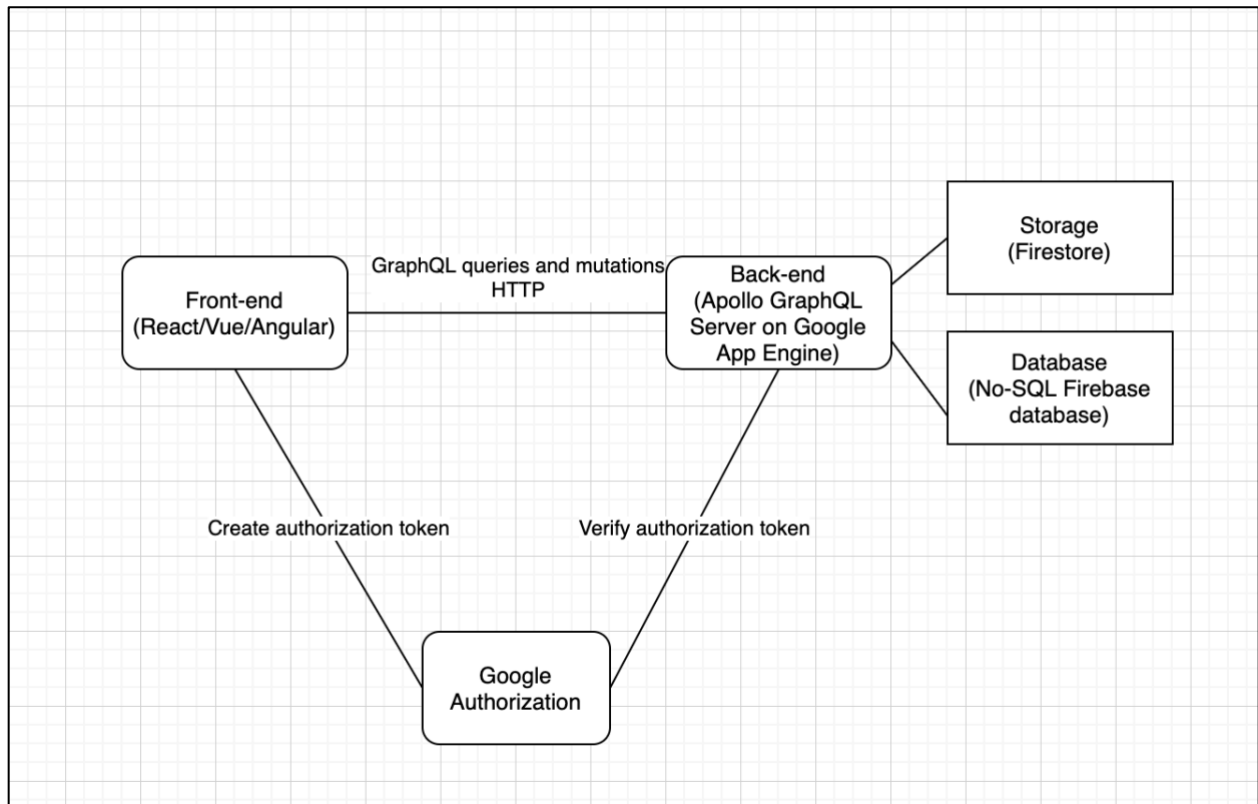
Безсерверні обчислення є абстрактними. Це означає що нема потреби керувати сервером, на якому запущена програма. Користувач взагалі нічого не знає про особливості операційної системи, оновлення, мережні налаштування та ряд інших деталей. Все, про що потрібно турбуватись - здатність платформи виконувати код написаний на вибраній мові програмування. Такий підхід дозволяє зосередитись на розробці функціоналу застосування, а не на адмініструванні серверів.

Еластичність означає, що хмарний провайдер бере на себе відповідальність за автоматичне збільшення або зменшення кількості ресурсів, задіяних для виконання обчислення, в залежності від навантаження.

Безсерверні обчислення дозволяють оптимально використовувати кошти. Якщо застосування знаходиться в простої — користувач сервісу не буде нічого платити, через те, що воно дійсно не використовує обчислювальні потужності в даний момент.

Ще одною характеристикою безсерверного підходу є обмежений життєвий цикл. Застосування запускається в межах контейнера та залишається запущеним від десяти хвилин до декількох годин. Після цього сервіс автоматично зупинить його. Для кожного виклику застосування відбуватиметься новий запуск контейнера.

3.1. Архітектура застосування



*Схема зв'язків між архітектурними компонентами застосування
Рисунок 3.2*

Як можна побачити на рисунку 3.2, основними архітектурними компонентами є: фронт-енд, бек-енд, база даних, сховище даних та сервіс для авторизації.

Серед сервісів, що пропонує Google Cloud Platform, було вирішено використовувати наступні:

- App engine
- Firestore
- Google Cloud Storage
- Google Authorization

На App Engine буде розміщено модулі, що стосуються АПІ застосування. Їх основні задачі — виконання обчислень пов'язаних з бізнес логікою застосування, доступ до бази даних та сховища даних, керування рівнями доступу.

Firestore — No-SQL база даних. Використовується для зберігання інформації про продукти, категорії та зв'язки між ними. Для ефективної роботи з базою даних було вирішено використовувати orm. Fireorm — зручний інструмент, який надає функції для роботи з базою даних Firebase. Дана бібліотека дозволяє використовуючи засоби TypeScript побудувати схему бази даних, що є великою зручністю, враховуючи те, що з коробки Firebase не володіє схемою та дозволяє зберігати дані довільними структурами.

Firestore - сховище даних, яке пропонує GCP, призначене для зберігання файлів. Його основна задача у моєму проекті - зберігати фотографії товарів. За потреби там можна розмістити і інші файли.

Google Authorization - зручне рішення для ідентифікації користувачів. Доступний широкий вибір способів автентифікації. Серед найпопулярніших - вхід через адресу електронної пошти та пароль, з використанням номеру телефону, через акаунти інших сервісів, Включаючи Google, Facebook, Apple, Outlook та інші. Здійснення автентифікації у моєму проекті відбувається через акаунт Google.

Користувачам сервісу Google Authorization не потрібно хвилюватись про шифрування та збереження паролів, генерацію та валідацію ключів доступу. Широка підтримка даного сервісу з боку розробників та велика кількість бібліотек для роботи з ним також прискорюють процес розробки застосування.

3.2. Компоненти застосування

3.2.1. База даних

Google Cloud Platform пропонує сервіси для роботи з різними базами даних. Серед No-SQL є такий вибір: Firebase Realtime Database або Cloud Firestore. Різниця між ними полягає в тому, що Realtime Database зберігає дані, як один великий JSON файл, а Cloud Firestore у іншому, схожому на JSON форматі, що надає можливість створення колекцій та підколекцій з документами. Через таку відмінність, перша більше підходить для зберігання невеликих даних, а друга для більш комплексних.

Data model	
Both Realtime Database and Cloud Firestore are NoSQL Databases.	
Realtime Database	Cloud Firestore
Stores data as one large JSON tree.	Stores data as collections of documents.
<ul style="list-style-type: none">• Simple data is very easy to store.• Complex, hierarchical data is harder to organize at scale.	<ul style="list-style-type: none">• Simple data is easy to store in documents, which are very similar to JSON.• Complex, hierarchical data is easier to organize at scale, using subcollections within documents.• Requires less denormalization and data flattening.
Learn more about the Realtime Database data model .	Learn more about the Cloud Firestore data model .

Порівняння Realtime Database та Cloud Firestore [8]

Рисунок 3.3

На Рисунку 3.3 зображена схема бази даних застосування. Вона містить такі об'єкти: Product, Category, CompatibilityGroup, Order. Product відповідає за зберігання інформації про різні види продуктів. Кожен об'єкт Product належить до певної категорії (або декількох категорій). Також об'єкти Product можуть належати до груп сумісності (CompatibilityGroup). Така структура дозволяє реалізувати функціонал рекомендацій сумісних товарів. При підтвердженні покупки клієнтом створюється об'єкт Order, який містить

інформацію про замовлення, таку, як кількість одиниць товару, контактні дані клієнта та дату створення.

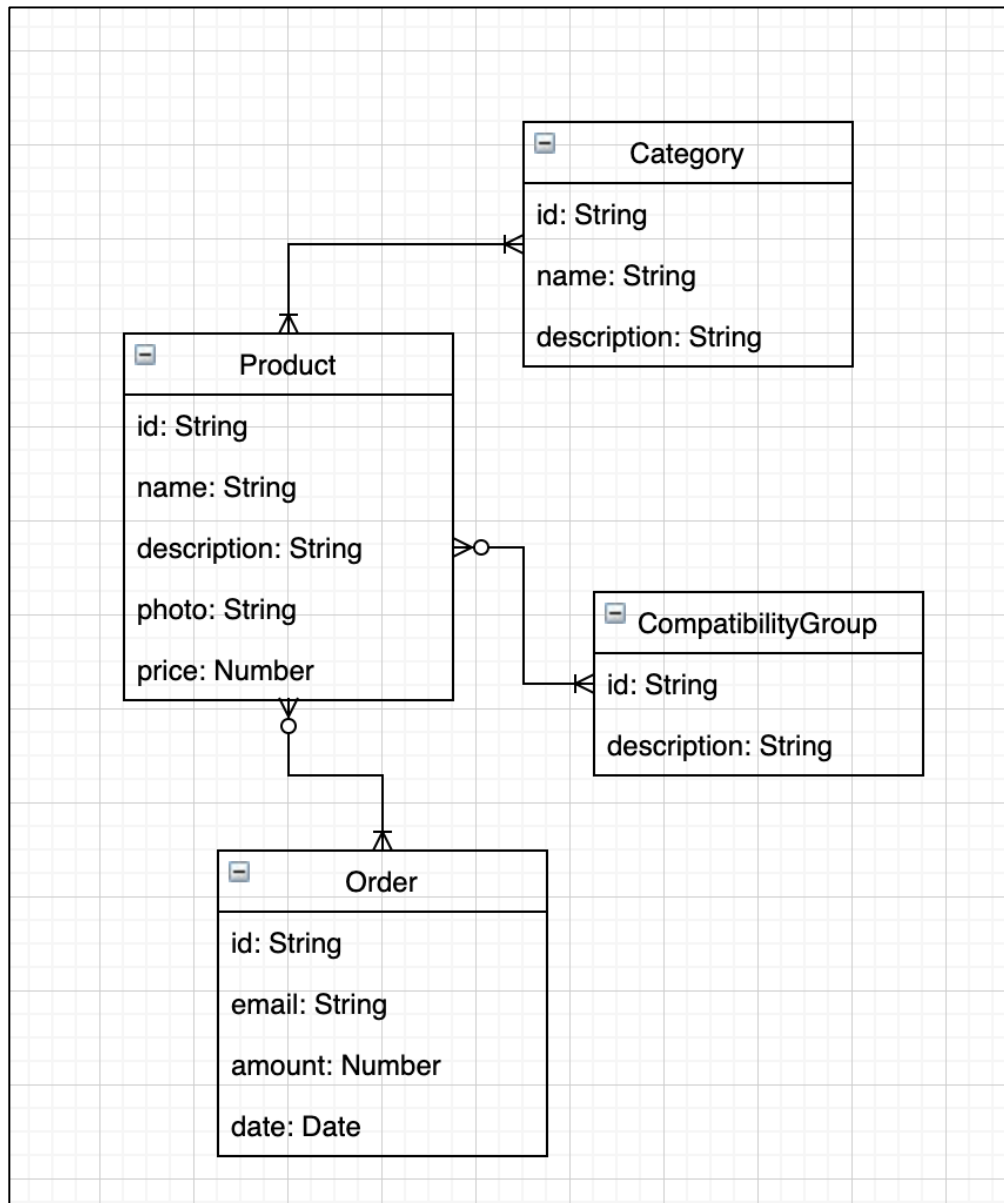


Схема бази даних застосування
Рисунок 3.4

Для зручної роботи з базою даних було вирішено використовувати ODM (Object Document Mapper). Такі інструменти, як ODM та ORM (Object Relation Mapper) виконують такі задачі:

- надають додатковий рівень абстракції над моделлю даних застосування,

- заховують та інкапсулюють запити за об'єктами вибраної мови програмування,
- виконують перетворення типів з тих, які підтримуються конкретною базою даних в ті, що підтримуються вибраною мовою програмування

Різниця між ORM та ODM полягає у тому, що ORM призначена для реляційних баз даних, а ODM – для нереляційних [9].

ORM/ODM – технологія у програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованого програмування. Використовуючи ORM розробник може читати та записувати дані в базу даних роблячи запити мовою програмування, якою написане саме застосування. Також збільшується рівень захисту застосування, через те, що унеможлиблюється виконання SQL Injection атаки.

Однією з найпопулярніших ODM для Google Firestore та TypeScript є fireorm. Перевагою даної ODM є те, що вона використовує функціонал декораторів TypeScript, що дозволяє проводити налаштування у декларативному стилі, що у свою чергу робить код більш компактним та зрозумілим.

Приклад налаштування моделі, наданий у документації. [10]:

```
import {Collection, getRepository} from 'fireorm';
@Collection()
class Todo {
  id: string;
  text: string;
  done: Boolean;
}
const todoRepository = getRepository(Todo);
```

3.2.2. Бекенд

Бекенд – компонент застосування, який об'єднує у собі декілька рівнів:

- GraphQL API-інтерфейс
- Бізнеслогіка
- Інфраструктурний рівень

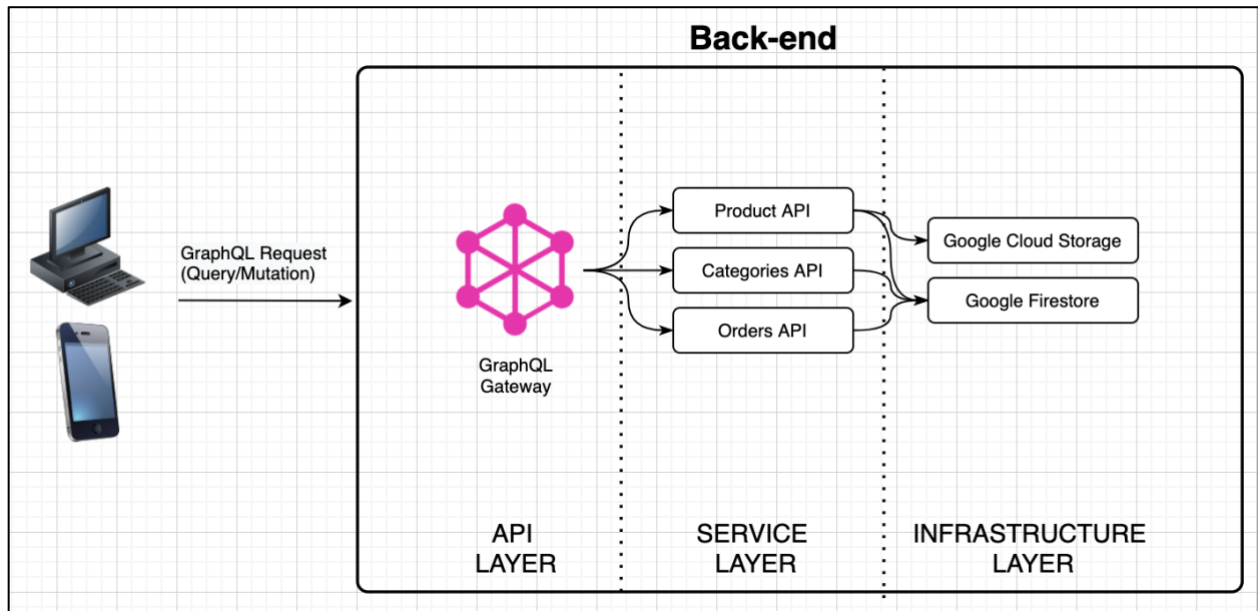


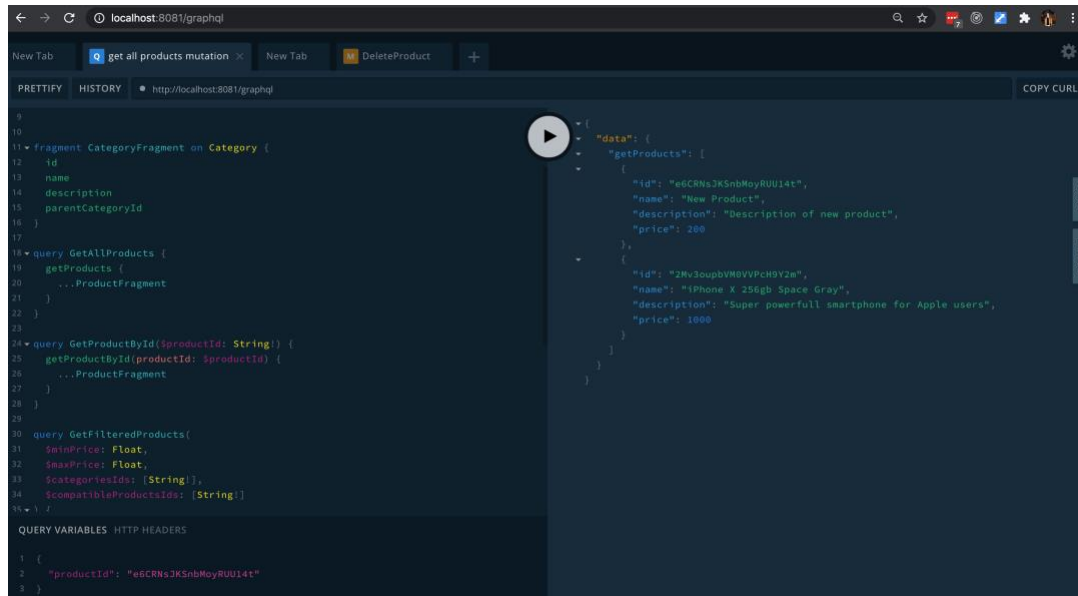
Схема компонентів бек-енду
Рисунок 3.5

GraphQL – мова запитів, що використовується для побудови програмних інтерфейсів застосунків. GraphQL виступає альтернативою класичному підходу з використанням RESTful API. Серед переваг цього підходу є:

- Система типів, що дозволяє детально описувати дані
- Запити, які дозволяють визначити, які саме поля даних отримувати
- Автоматичне створення документації
- Середовища для розробки та тестування GraphQL сервера [11]

Сервер побудовано на основі фреймворку Express, серед переваг якого є гнучкість та мінімалістичність. З використанням Express можна легко та зручно створити сервер, що буде отримувати та обробляти HTTP-запити. Архітектура застосунку дозволяє розділити процес обробки запиту на різні рівні, використовуючи Middlewares для обрахунків та Routes для маршрутизації.

Для того, щоб налаштувати підтримку GraphQL запитів було вирішено використовувати ApolloServer, який встановлюється, як окремий рівень express серверу. Усі GraphQL запити надходять за шляхом /graphql, тип http запиту – POST. При спробах доступитись до сервера з іншим шляхом, буде отримане повідомлення про помилку з кодом 404. У режимі розробки при GET-запиті до /graphql у браузері буде відкрито середовище для розробки та тестування GraphQL API.



Середовище для розробки та тестування GraphQL API

Рисунок 3.6

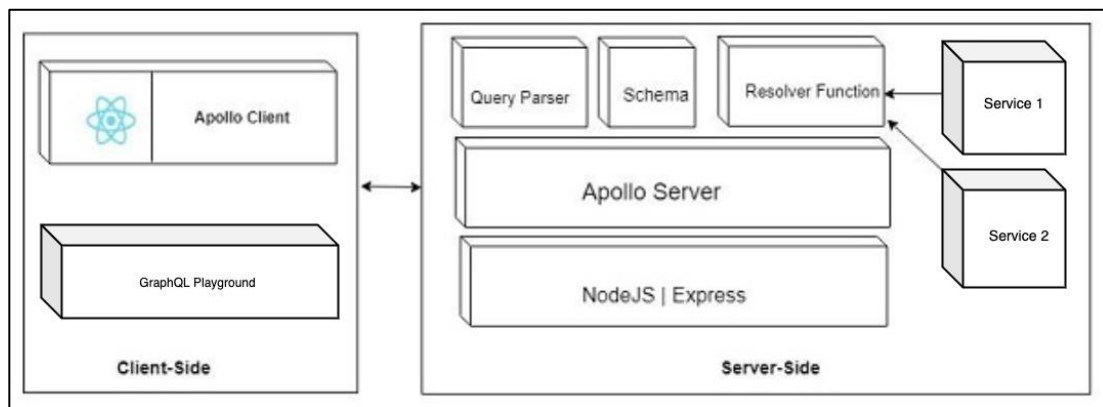
Основна бізнес логіка застосунку знаходиться на рівні сервісів: Products, Categories, Orders. Дані компоненти виконують обчислення пов'язані

зі створенням, пошуком та фільтрацією даних. Для здійснення доступу до бази даних використовуються об'єкти репозиторіїв, які надає `fireorm`. Також ці компоненти можуть викликати інші сторонні сервіси, наприклад, для виконання оплати або надсилання повідомлень на електронну пошту.

Код, який відповідає за налаштування до приєднання сторонніх компонентів належить до інфраструктурного рівню. Такий підхід дозволяє побудувати застосування на достатньому рівні абстракції, щоб не залежати від конкретних реалізацій сторонніх компонентів. Зміни платіжної системи або сервісу розсилки повідомлень будуть тривіальними і не потребуватимуть змін у компонентах бізнес логіки. Також в межах інфраструктурного рівня відбуваються налаштування середовища застосунку, що дозволяє налаштувати такі режими, як `development` та `production`.

3.2.3. GraphQL API

За підтримку запитів GraphQL відповідає `ApolloServer`, який функціонує на основі `Express Middleware`. При виклику сервера, відбувається розшифрування та валідація запиту. У випадку успішної обробки, використовуючи файл `schema.graphql`, відбувається виклик відповідної `Resolver`-функції.



Структура `ApolloServer`
Рисунок 3.7

Схема програмного інтерфейсу застосування міститься у файлі `schema.graphql`. До нього входить детальний опис усіх можливих запитів до API, разом з типами аргументів та результатів. Широка система типів дозволяє побудувати модель даних зі зв'язками між ними.

Запити GraphQL існують двох типів: `Query` та `Mutation`. Різниця між ними полягає у тому, що перший слід використовувати коли не відбувається зміни даних, а другий при здійсненні модифікацій.

Для зручнішої інтеграції GraphQL у TypeScript код було вирішено використовувати `type-graphql`. Ця бібліотека дозволяє згенерувати файл зі схемою GraphQL API використовуючи декоратори у TypeScript коді. Для цього потрібно позначити відповідні поля класів декоратором `@Field()`, за потреби вказавши додаткові конфігурації. Також потрібно позначити функції – які відповідають за виконання запитів використовуючи декоратори `@Query` та `@Mutation` відповідно.

Сигнатура Resolver-функції має наступний вигляд:

```
fieldname: (parent, args, context, info) => data;
```

- **fieldname** – назва запиту або підзапиту
- **parent** – результат обчислення батьківського поля (завжди обчислюється перед обрахунком дочірніх елементів)
- **args** – об'єкт, що містить аргументи
- **context** – об'єкт, що містить дані для усіх Resolver-функцій.

Зазвичай використовується для передачі певного початкового стану, наприклад токен авторизації [12].

При здійсненні запиту GraphQL дозволяє вказати, які саме поля користувач хоче отримати. Це дозволяє зменшити кількість даних, що передаються, що у свою чергу збільшує ефективність застосування. Також в

межах одного запиту можна об'єднувати різні виклики Resolver-функцій, що дозволяє зменшити кількість запитів, які виконуються.

Наприклад, для відображення списку товарів у клієнтському застосуванні, може бути потрібно лише найменування та фотографія кожного з них. Якщо об'єкти товарів містять інші поля, у класичному підході з використанням RESTful API доведеться завантажувати усі, тоді, як з GraphQL API запит буде більш оптимальним. Код запиту матиме такий вигляд:

```
{
  getProducts {
    id
    name
    photo
  }
}
```

Результат такого запиту матиме наступний вигляд:

```
{
  "getProducts": [
    {
      "id": 1,
      "name": "iPhone 12 pro",
      "photo": "https://photo.com/1"
    },
    ...
  ]
}
```

Модель даних, описана методами GraphQL може містити не тільки звичайні поля, а й обчислювані значення. Наприклад, для відображення сторінки певного продукту у клієнтському застосунку потрібно назва,

вартість, фото та список категорій, до яких належить даний товар. Такий GraphQL запит матиме наступний вигляд:

```
{
  getProductById (productid: 4) {
    id name photo
    categories { name }
  }
}
```

Нижче наведено приклад результату даного запиту:

```
{
  "getProductById": {
    "id": 4, "name": "iPhone 12",
    "photo": "https://photo.com/photo.png",
    "categories": [{"name": "smartphone"}, {"name": "apple"}, ...]
  }
}
```

Зв'язки між типами даних також дозволяють зменшити кількість запитів до сервера. Оскільки одним запитом можна отримати, наприклад, інформацію про продукт, про усі сумісні з ним товари та їх категорії.

3.2.4. Автентифікація та авторизація

Автентифікація – перевірка достовірності пред'явленого користувачем ідентифікатора. На практиці реалізовується за допомогою окремого сервісу, що відповідає за створення, валідацію та видалення токенів. Користувач додає свій токен до усіх подальших запитів до сервера, який у свою чергу виконує верифікацію токена.

Для реалізації автентифікації було вирішено використовувати Google Cloud Identity. Даний сервіс надає функціонал для додавання користувачів та налаштування прав доступу до застосунків, забезпечуючи захист даних та

масштабованість. При потребі можна виконати розширені налаштування з використанням багатфакторної автентифікації.

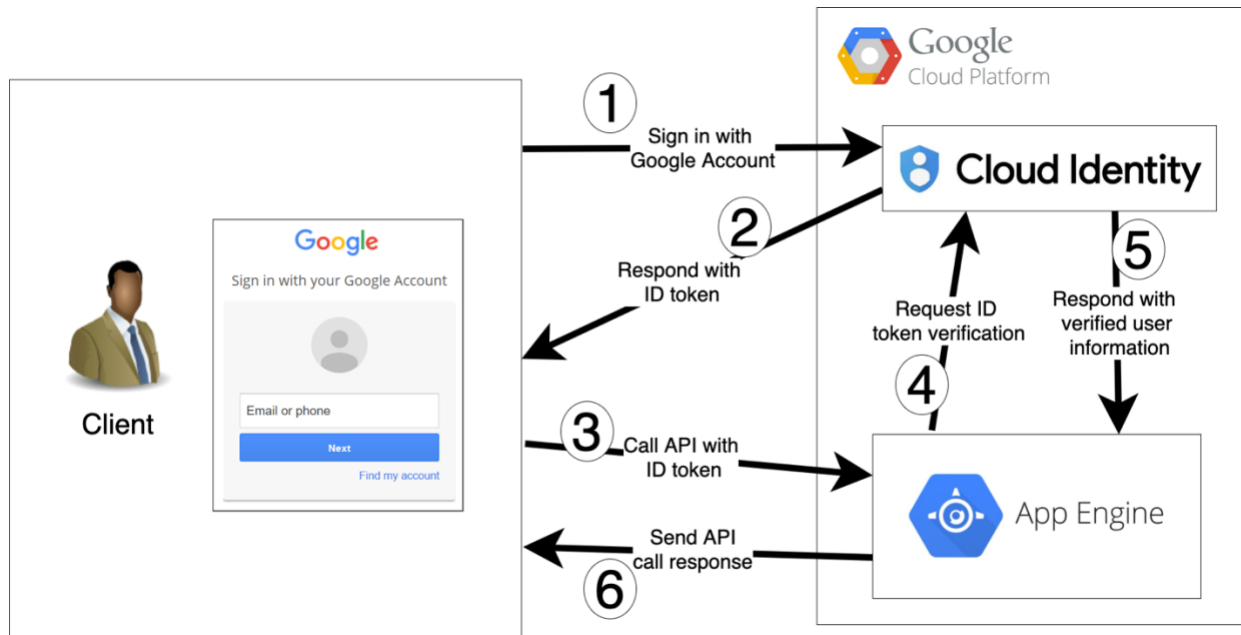


Схема функціонування автентифікації за допомогою сервісів Google Cloud.

Рисунок 3.8

На рисунку 3.8. зображено покроковий процес автентифікації та виконання запиту до сервера. Спочатку користувач, використовуючи клієнтське застосування вводить свої дані до облікового запису Google. При успішному вході сервіс Cloud Identity створює та видає ID токен. Варто зазначити, що можливі й інші способи ідентифікації користувача, включаючи вхід через логін та пароль або з допомогою акаунту стороннього сервісу. Для зміни або додавання нового способу ідентифікації достатньо виконати прості налаштування у Google Cloud Identity, не змінюючи нічого у самому коді застосування. Далі користувач зберігає отриманий ID токен та використовує його для виконання запитів до API.

Коли сервер отримує новий запит, спочатку відбувається перевірка наявності Authorization заголовку на рівні Express. Токен надсилається на Google Cloud Identity для перевірки. При успішній верифікації, запит

передається для обробки Apollo сервером, інформація про користувача зберігається в контексті.

Якщо отриманий токен невалідний, користувач отримає помилку з кодом “401 Unauthorized”. Це може означати, що токен хибний або застарілий. У випадку застарілого токenu, клієнт спробує поновити його через Google Cloud Identity та виконає запит заново.

Авторизація – процедура перевірки прав користувача на виконання певних дій на веб-ресурсі або в комп’ютерній системі, результатом якої є дозвіл або відмова у виконанні певних операцій, а також надання відповідно з правами користувачу можливостей, які гарантовані йому веб-ресурсом або комп’ютерною системою [13].

Після того, як сервер виконує процедуру ідентифікації та автентифікації користувача, виконується його авторизація. Кожен ресурс (GraphQL Query/Mutation) може мати обмеження на доступ залежно від ролі користувача. Наприклад, запити на додавання нових продуктів та категорій до системи можуть виконувати лише користувачі з правами адміністратора. Список ідентифікаторів адміністраторів налаштовується через змінні середовища застосування.

Налаштування прав доступу до кожного конкретного ресурсу виконується за допомогою бібліотеки type-graphql, яка містить TypeScript декоратор `@Authorized()`. Масив ролей користувачів, які можуть користуватись ресурсом передається як аргументи декоратора. Також при налаштуванні Apollo сервера вказується функція, яка виконує авторизацію. Для прийняття рішення, чи дозволити доступ користувачу до ресурсу, використовується інформація про користувача, отримана після його

автентифікацій. Якщо користувач не володіє достатніми правами, буде надіслано інформацію про помилку.

3.2.5. Сервіс продуктів

Сервіс продуктів надає функції для створення, редагування, пошуку та видалення товарів, представлених в інтернет-магазині. Усі функції з продуктами можна розділити на ті, що доступні усім клієнтам, та ті, що доступні лише адміністраторам магазину.

Пошук продуктів може відбуватись за різними критеріями. У випадку пошуку за назвою система вибере лише ті опції, що містять в назві певну ключову фразу незалежно від регістру символів. Також доступна фільтрація за мінімальною та максимальною вартістю. Крім того можна знайти продукт за категоріями або за іншими сумісними товарами.

Cloud Firestore надає потужний функціонал запитів для визначення, які саме елементи колекції або підколекції потрібно отримати. Це дозволяє створювати складені запити, що містять інформацію про фільтри, та отримувати лише ті дані, що потрібно. Також відпадає потреба організовувати логіку фільтрації у коді. Такий підхід у свою чергу дозволяє зменшити кількість даних, які передаються через мережу, та оптимізувати витрати за використання Google Cloud Platform. Нижче наведено приклад пошуку продуктів по ціні:

```
const queryBuilder = ProductRepository
  .whereGreaterOrEqualThan('price', minPrice)
  .whereLessOrEqualThan('price', maxPrice);
  .find();
```

Для реалізації пошуку за сумісними товарами, використовується колекція груп сумісності, яка налаштовується адміністратором. Система знаходить усі такі групи, у яких є заданий продукт. З кожної групи береться список, усі вони об'єднуються у результуючий масив, видаляючи дублікати та

продукт для якого відбувався пошук. Також сервіс продуктів надає функції для створення, оновлення та видалення товару за ідентифікатором. Нижче представлена структура сервісу продуктів.

```
ProductsService
  getProducts({query, minPrice, maxPrice, categoriesIds, compatibleProductsIds,}: GetProductsInput): Promise<Product[]>
  getProductById(productId: string): Promise<Product | null>
  createProduct(product: PartialBy<Product, "id">): Promise<Product>
  updateProduct(product: Product): Promise<Product>
  deleteProduct(productId: string): Promise<void>
```

На рівні програмного інтерфейсу GraphQL для клієнта застосування має надає схему:

```
type Product {
  id: String!
  name: String!
  photo: String!
  price: Float!
  description: String!
  categoriesIds: [String!]!
  categories: [Category!]!
  compatibleProducts: [Product!]!
}
type Query {
  getProductById(productId: String!): Product!
  getProducts(options: GetProductsInput = {}): [Product!]!
}
```

Така структура дозволяє отримати інформацію про продукти, а також про категорії, до яких належить кожен конкретний продукт, та власне до інших сумісних продуктів.

Адміністратору магазину на рівні GraphQL доступні наступні операції:

```
type CompatibleGroup {
  id: String!
  description: String!
  productsIds: [String!]!
  products: [Product!]!
}
```

```

type Query {
  getProductById(productId: String!): Product!
  getProducts(options: GetProductsInput = {}): [Product!]!
  getCompatibleGroups(productId: String!):
    [CompatibleGroup!]!
}

type Mutation {
  createProduct(options: CreateProductInput!): Product!
  deleteProduct(productId: String!): Void
  updateProduct(options: UpdateProductInput!): Product!
  updateCompatibleGroup(options: UpdateCompatibleGroupInput!):
    CompatibleGroup!
  createCompatibleGroup(options: CreateCompatibleGroupInput!):
    CompatibleGroup!
  deleteCompatibleGroup(compatibleGroupId: String!): Void
}

```

Адміністратор інтернет-магазину може створювати нові продукти, оновлювати та видаляти їх. Крім цього, у адміністратора є можливість отримати інформацію про групи суміжності (усі або для певного продукту). Також він може створювати нові групи, доповнювати або видаляти існуючі.

3.2.6. Сервіс категорій

Сервіс категорій надає функції для перегляду, пошуку, створення, оновлення та видалення категорій. Кожна з них може містити підкатегорії, тому можливий пошук за батьківською категорією. GraphQL схема для взаємодії з цим сервісом для клієнта має такий вигляд:

```

type Category {
  id: String!
  name: String!
  description: String!
  parentCategoryId: String
  products: [Product!]!
  subcategories: [Category!]!
}

type Query {
  getCategories(options: GetCategoriesInput = {}): [Category!]!
  getCategoryById(categoryId: String!): Category!
}

```

Така структура дозволяє для кожної категорії легко отримати список її продуктів, а також список підкатегорій разом з їх продуктами.

Адміністратору магазину доступні також функції додавання, оновлення та видалення категорій. Такий вигляд має GraphQL схема застосування для користувача-адміністратора:

```
type Mutation {  
  createCategory(options: CreateCategoryInput!): Category!  
  updateCategory(options: UpdateCategoryInput!): Category!  
  deleteCategory(categoryId: String!): Void  
}
```

3.2.7. Сервіс замовлень

Сервіс замовлень відповідає за створення та обробку нових замовлень від клієнтів. Адміністратору інтернет-магазину доступні функції для перегляду усіх замовлень, пошуку замовлення за електронною поштою клієнта. Також йому надається функція видалення неактуальних замовлень. Схема застосування для клієнта має такий вигляд:

```
type Order {  
  id: String!  
  date: DateTime!  
  email: String!  
  items: [OrderItem!]!  
}  
  
type OrderItem {  
  amount: Int!  
  productId: String!  
  product: Product!  
}  
  
type Mutation {  
  createOrder(options: CreateOrderInput!): Order!  
}
```

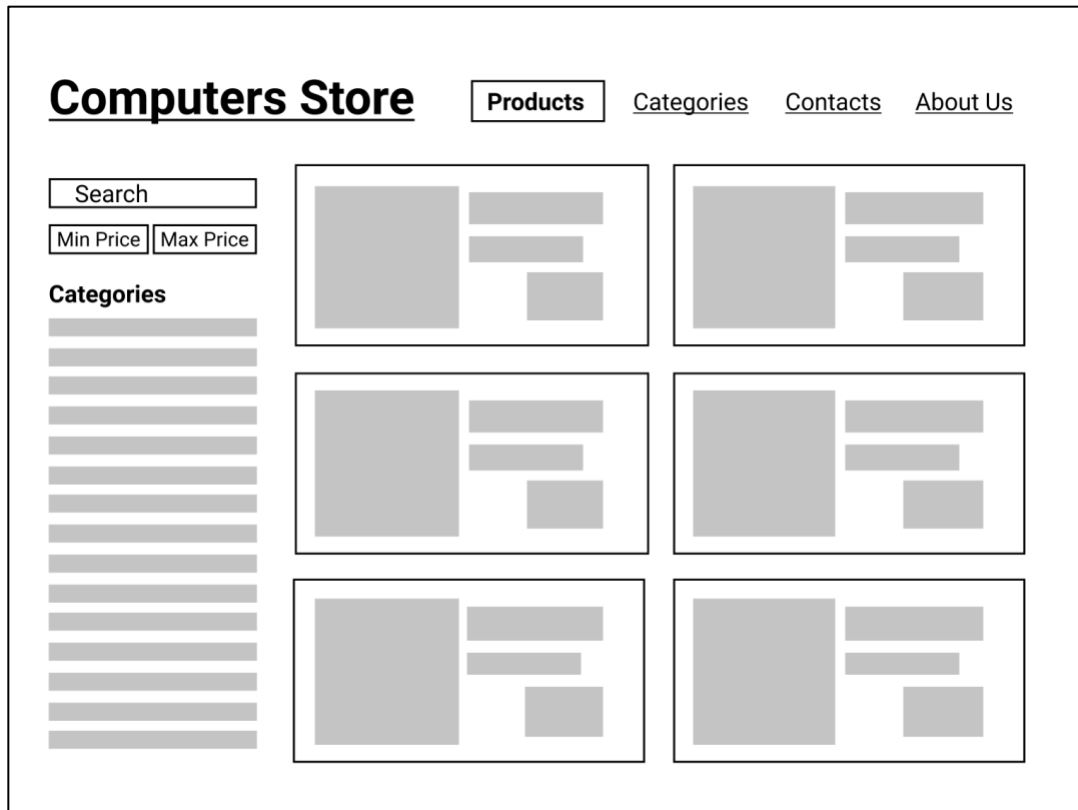
Схема застосування для адміністратора містить такі додаткові функції:

```
type Mutation {  
  deleteOrder(orderId: String!): Void!  
}  
  
type Query {  
  getOrders(email: String): [Order!]!  
  getOrderById(orderId: String!): Order!  
}
```

3.2.8. Клієнтське застосування

Клієнтське застосування, як і застосуванні API буде розгортатись за допомогою інструменту Google Cloud Platform – App Engine, як один із під-сервісів. Отже, клієнтське застосування буде функціонувати у межах безсерверного підходу, коли фізичний сервер буде працювати лише тоді коли відбуваються запити, а після цього вимикатиметься. Це також дозволить оптимізувати використання хмарних ресурсів.

Для розробки клієнтських застосувань зручно використовувати SPA (Single-page application) підхід. Однією із найпопулярніших JavaScript бібліотек його реалізації є React. Для виконання запитів до сервера відповідатиме React Apollo Client – бібліотека, яка дозволяє використовувати GraphQL API. Також здійснюється зчитування GraphQL-схеми та генерація типів мовою TypeScript, що дозволяє зменшити кількістю дублюючого коду.



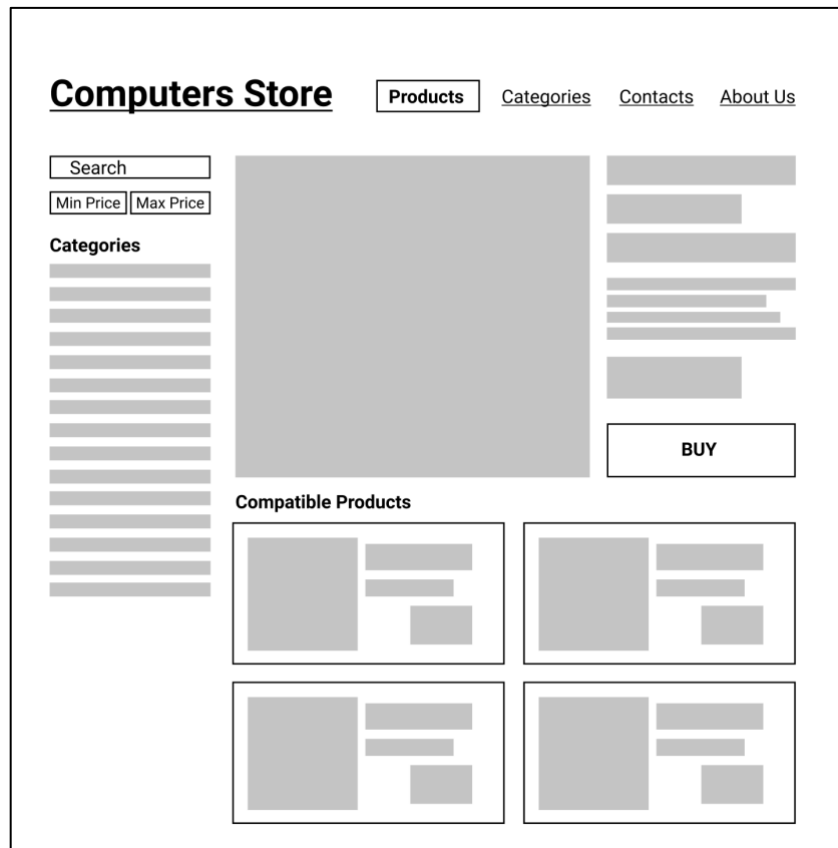
*Макет головної сторінки сайту інтернет-магазину
Рисунок 3.9*

На рисунку 3.9 можна зображено макет основної сторінки сайту. Вона містить список продуктів, список доступних категорій та інструменти для налаштування фільтрів. При завантаженні основної сторінки відбувається запит до сервера такого вигляду:

```
query GetHomePageData {  
  categoriesList: getCategories {  
    id  
    name  
  }  
  
  productsList: getProducts(options: {query: "iphone"}) {  
    id  
    name  
    photo  
    price  
  }  
}
```

Такий запит може поєднати в собі компоненти, що необхідно завантажити – список категорій та список продуктів з певними фільтрами. Відповідь сервера може мати такий вигляд:

```
{
  "categoriesList": [
    { "id": "7UycsY", "name": "Phones" },
    { "id": "Rokuqu", "name": "iPhone" },
    { "id": "gspBaSAr3rdtRVMGRv0l", "name": "Apple" }
  ],
  "productsList": [
    {
      "id": "2Mv3oupbVM0VVPch9Y2m",
      "name": "iPhone X 256gb Space Gray",
      "photo": "https://google.com/oupbVM0VVPch9Y2m.png",
      "price": 1000
    }
  ]
}
```



*Макет сторінки з детальною інформацією про продукт
Рисунок 3.10*

Користувач також може вибрати певний товар перейти на сторінку з детальнішою інформацією. Її макет зображено на рисунку 3.10. Основними компонентами сторінки є список категорій, детальна інформація про вибраний товар та список товарів, що сумісні з даним. GraphQL запит для отримання усіх цих даних має такий вигляд:

```
query GetProductPageData {
  categoriesList: getCategories {
    id name
  }
  product: getProductById(productId: "2Mv3oupbVM0VVPcH9Y2m") {
    id name photo price description
    compatibleProducts {
      id name photo price
    }
  }
}
```

Відповідь сервера буде містити список категорій та інформацію про продукт разом зі списком інших сумісних товарів.

```
{
  "categoriesList": [
    { "id": "7UycsY", "name": "Phones" },
    { "id": "Rokuqu", "name": "iPhone" },
    { "id": "gspBaSAr3rdtRVMGRv0l", "name": "Apple" }
  ],
  "product": {
    "id": "2Mv3oupbVM0VVPcH9Y2m",
    "name": "iPhone X 256gb Space Gray",
    "photo": "https://google.com/photo-2Mv3oupbVM0VVPcH9Y2m.png",
    "price": 1000,
    "description": "Super powerfull smartphone for Apple users",
    "compatibleProducts": [
      {
        "id": "e6CRNsJKSnbMoyRUU14t",
        "name": "Apple Lightning to USB Cable",
        "photo": "https://google.com/photo-e6CRNsJKSnbMoyRUU14t.png",
        "price": 1000
      }
    ]
  }
}
```

3.3. Процес розробки застосування

Процес розробки власного застосування на Google Cloud Platform можна розбити на такі складові:

- початкові налаштування GCP
- налаштування процесу для подальшого використання GCP

Спочатку я створив та налаштував обліковий запис у Google Cloud Platform. Наступним кроком слід створити новий проект. Для виконання подальших дій потрібно налаштувати спосіб оплати. У межах виконання курсової роботи вистачатиме безкоштовного тарифу, проте додавання оплати є обов'язковим кроком. GCP також надає функції для розрахунку витрат, налаштування бюджету та попередження про його перевищення.

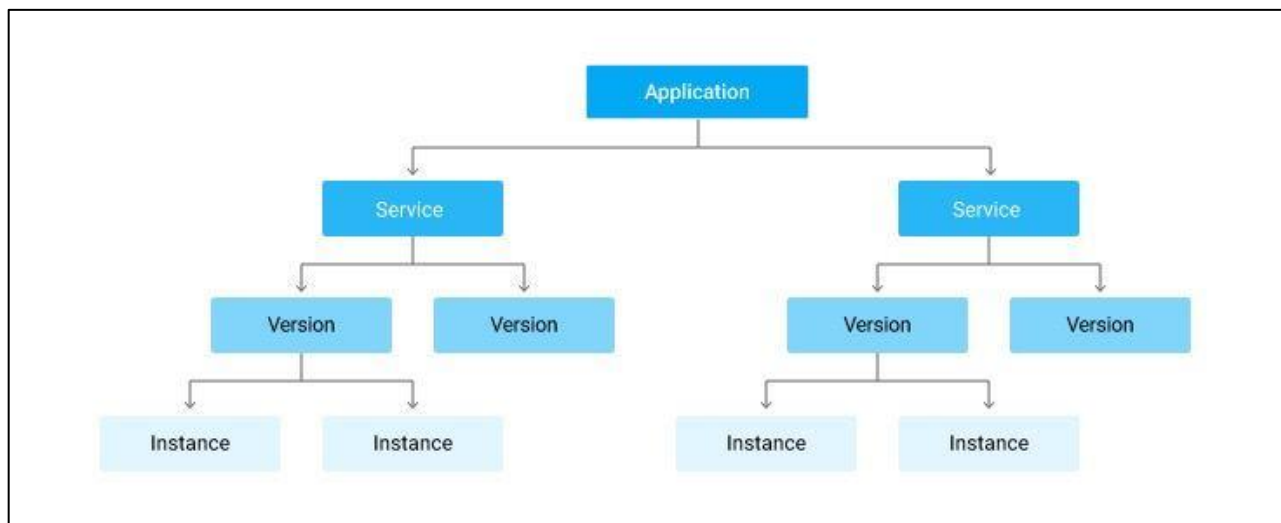
За посиланням **console.cloud.google.com/iam-admin** можна налаштувати окремі акаунти для сервісів або розробників, вказавши їх права на доступ до визначених ресурсів та виконання конкретних дій над продуктом. У цілях безпеки головний акаунт повинен використовуватись лише для регулювання рівнів доступу інших акаунтів. Усі решта операції виконуються через спеціальні акаунти.

Наступним кроком слід налаштувати сервіс, що відповідає за виконання безсерверних обчислень - App Engine. Серед його переваг є доступність для великої кількості мов програмування, підтримка контейнерів Docker, версіонування застосувань та розподіл трафіку, що робить можливим А/В тестування та поступове виконання оновлень [14].

Для завершення налаштування App Engine слід вказати регіон у якому будуть виконуватись обчислення.

У межах даного проекту App Engine буде використовуватись у двох цілях: розміщення GraphQL API та серверу, що буде містити клієнтське

застосування на React. Щоб налаштувати такий розподіл у App Engine слід налаштувати окремі сервіси для кожного. У кожному проєкті буде міститись файл `app.yaml`, у якому зазначена відповідна назва сервісу.



Ієрархія App Engine з багатьма сервісами [15]

Рисунок 3.11

Коли ви створюєте App Engine застосування усі ресурси будуть розміщені у вибраному регіоні, включаючи й сам код та усі ключі. App Engine містить як мінімум один сервіс – `default`. Кожен сервіс може містити багато версій, кожна з яких у деякий момент часу може виконуватись на різних віртуальних машинах.

Кожній версії конкретного сервісу присвоюється власне URL-посилання. Формується воно за наступним зразком:

https://VERSION_ID-dot-SERVICE_ID-dot-PROJECT_ID.REGION_ID.r.appspot.com

- **VERSION_ID** – номер версії конкретного сервісу, встановлюється автоматично
- **SERVICE_ID** – ідентифікатор конкретного сервісу, налаштовується через файл `app.yaml`

- **PROJECT_ID** – назва проекту, налаштовується при створенні
- **REGION_ID** – назва регіону у якому розміщено ресурси App Engine

Наступним компонентом Google Cloud Platform, що потрібно налаштувати є база даних. Серед варіантів, які пропонує GCP, було вирішено використовувати Google Firestore. Firestore – гнучка, масштабована NoSQL хмарна база даних для зберігання даних як клієнтом, так і сервером.

Для початку потрібно створити базу Firestore та вибрати локацію для її розміщення. Для доступу до бази даних зі сторони сервера потрібно налаштувати автентифікацію. Спочатку потрібно створити новий ключ сервіс-акаунта, вказавши його назву та налаштувавши відповідні права. У результаті буде отримано JSON об'єкт, який є ключем для доступу до бази даних. При запуску застосування, ключ буде отримано зі змінних середовища, та використано для ініціалізації Firestore в коді.

Після успішного налаштування Google Cloud Platform потрібно ініціалізувати локальний проект за допомогою проектного менеджера yarn. Також потрібно налаштувати скрипти для запуску застосування у режимі розробника, а також побудови та розгорнення на App Engine. Для запуску застосування потрібно передати такі змінні середовища:

- **ADMIN_EMAIL** – адреса електронної пошти адміністратора інтернет-магазину
- **CLIENT_ID** – ідентифікатор клієнтського застосування, що використовується для авторизації
- **GOOGLE_APPLICATION_CREDENTIALS** – json об'єкт або посилання на json-файл, що містить дані сервіс-акаунту, який використовується для використання сервісів Google Cloud Platform

Скрипт для запуску застосування у режимі розробника має наступний вигляд:

```
cross-env ADMIN_EMAIL=illya.kurochkin@gmail.com  
CLIENT_ID=407408718192.apps.googleusercontent.com  
GOOGLE_APPLICATION_CREDENTIALS= google-app-credentials.json  
nodemon --exec "npm run start" --ext "ts,json"
```

У такому режимі відбувається відстеження змін коду у файлах проекту, та автоматичний перезапуск сервера.

Щоб налаштувати розгортання проекту на Google Cloud Platform потрібно скачати інструмент командного рядка gcloud. Він надає можливості керування ресурсами різних сервісів GCP, включаючи App Engine, Cloud SQL, Kubernetes Engine та Cloud DNS. Для того, щоб вказати додаткові налаштування розгортання застосування потрібно додати файл app.yaml. Такий вигляд має app.yaml у проекті shop-api.

```
runtime: nodejs  
env: flex  
service: shop-api  
automatic_scaling:  
  target_cpu_utilization: 0.50  
  min_instances: 1  
  max_instances: 3
```

4. ВИСНОВКИ

У результаті виконання курсової роботи було спроектовано та описано структуру веб застосування для інтернет-магазину на основі багаторівневої архітектури та безсерверного підходу до обчислень.

Застосування побудоване на платформі Google Cloud Platform. Для розробки проекту було використано такі технології: Google App Engine для виконання обчислень, база даних Cloud Firestore та Google Identity Platform для авторизації користувача.

З огляду на цінову політику, часті акції, початковий пакет ресурсів для знайомства з платформою та похвилинну оплату, Google Cloud Platform є хорошим вибором для побудови проектів, що тільки починають свою роботу, а простота використання та налаштування дозволить команді краще зосередитись на вирішенні функціональних задач.

У межах роботи було продемонстровано сучасний стек технологій, що використовується для швидкої та зручної побудови багаторівневих застосувань. До нього входить мова програмування TypeScript та мова запитів до сервера GraphQL. Для кращої підтримки GraphQL було застосовано інструменти Apollo та type-graphql, що дозволяє зручно організовувати взаємодію між клієнтським застосуванням та бек-ендом.

У результаті роботи над практичною частиною було реалізовано компонент застосування - сервер GraphQL API, а також змодельовано та описано компонент клієнтського застосування. Розроблений компонент розгорнений на Google App Engine, що реалізовує безсерверну модель. Окрім послуг хостингу, App Engine володіє функціоналом для автоматичного масштабування та задіювання додаткових інстансів, у випадку збільшення навантаження. Для збереження даних використовується Cloud Firestore –

гнучка масштабована база даних, призначена для розробки серверів, мобільних та веб-застосунків, що легко інтегрується з іншими сервісами Google Cloud Platform.

У подальшому розроблену систему можна розширити додатковими функціями, потрібними для роботи онлайн магазину. Сервіс може бути розширено для підтримки інших доступних регіонів GCP. Розроблений код, структура проекту, та налаштування також можуть бути запозичені та застосовані для побудови інших багаторівневих застосунків, зокрема компоненту бек-енду.

5. Список використаної літератури

1. Traditional IT vs Cloud Computing (Amar Patel, Vice President & Chief Revenue Officer – JANUARY 2, 2020) – <https://www.connectria.com/blog/traditional-it-vs-cloud-computing/>
2. AWS vs. Azure vs. Google Cloud – <https://www.dincloud.com/blog/aws-vs-azure-vs-google-cloud>
3. Amazon Web Services: 6 Features That Can Make the Difference (Praveen Kumar Muppala – APRIL 17, 2014) – <https://cloudacademy.com/blog/common-features-improvements-for-amazon-web-services/>
4. Overview poster of Azure features, services, and common uses (James van den Berg – SEPTEMBER 10, 2014) – <https://mountainss.wordpress.com/2014/09/10/overview-poster-of-azure-features-services-and-common-uses-azure-cloud-sysctr-hyperv/>
5. 4 Reasons Why You Should Choose Google Cloud Over Others – (Evie Harrison on – JULY 17, 2020) – <https://geekflare.com/reasons-to-choose-google-cloud/>
6. Serverless computing – https://en.wikipedia.org/wiki/Serverless_computing
7. Serverless on GCP: A Comprehensive Guide (Tyler Treat – AUGUST 22, 2019) – <https://dzone.com/articles/serverless-on-gcp>
8. Google Firestore documentation. Choose a database: Cloud Firestore or Realtime Database – <https://firebase.google.com/docs/firestore/rtdb-vs-firestore>

9. What is ORM? Why to use it and Brief Introduction of ORM Frameworks (Vinayak Grover – NOVEMBER 8, 2019) – <https://medium.com/@grover.vinayak0611/what-is-orm-why-to-use-it-and-brief-introduction-of-orm-frameworks-b61b16d02a3c>
10. Fireorm documentation – <https://github.com/wovalle/fireorm>
11. GraphQL documentation – <https://graphql.org/>
12. ApolloServer documentation. Resolvers – <https://www.apollographql.com/docs/tutorial/resolvers/>
13. Авторизація – <https://promo.ingate.ru/seo-wikipedia/authorization/>
14. Google App Engine documentation – <https://cloud.google.com/appengine#all-features>
15. Google App Engine overview – <https://cloud.google.com/appengine/docs/standard/nodejs/an-overview-of-app-engine>

ДОДАТКИ

Додаток 1 (запит на отримання даних для початкової сторінки)

Запит:

```
query GetHomePageData {
  categoriesList: getCategories {
    id
    name
  }

  productList: getProducts(options:{query: "iphone"}) {
    id
    name
    photo
    price
  }
}
```

Відповідь сервера:

```
{
  "data": {
    "categoriesList": [
      {
        "id": "7UycsYg7ficg0zpknWLi",
        "name": "Phones"
      },
      {
        "id": "Rokuquye9g8ech3YdBD1",
        "name": "iPhone"
      },
      {
        "id": "S2HJYCjBmHTPElLbQPkR",
        "name": "Cables"
      },
      {
        "id": "gspBaSAr3rdtRVMGRvOl",
        "name": "Apple"
      }
    ],
    "productList": [
      {
        "id": "2Mv3oupbVM0VVPCh9Y2m",
        "name": "iPhone X 256gb Space Gray",
        "photo": "https://google.cloudstorage.com/photo-2Mv3oupbVM0VVPCh9Y2m.png",
        "price": 1000
      }
    ]
  }
}
```

Додаток 2 (запит на отримання даних для сторінки товару)

Запит:

```
query GetProductPageData {
  product: getProductById(productId: "2Mv3oupbVM0VVPcH9Y2m") {
    id
    name
    photo
    price
    description

    categories {
      id
      name
    }

    compatibleProducts {
      id
      name
      photo
      price
    }
  }
}
```

Відповідь сервера:

```
{
  "data": {
    "product": {
      "id": "2Mv3oupbVM0VVPcH9Y2m",
      "name": "iPhone X 256gb Space Gray",
      "photo": "https://google.cloudstorage.com/photo-2Mv3oupbVM0VVPcH9Y2m.png",
      "price": 1000,
      "description": "Super powerfull smartphone for Apple users",
      "categories": [
        {"id": "7UycsYg7ficg0zpknWLi", "name": "Phones"},
        {"id": "gspBaSAr3rdtRVMGRvOl", "name": "Apple"},
        {"id": "Rokuquye9g8ech3YdBD1", "name": "iPhone"}
      ],
      "compatibleProducts": [
        {
          "id": "e6CRNsJKSnbMoyRUU14t",
          "name": "Apple Lightning to USB Cable",
          "photo": "https://google.cloudstorage.com/photo-e6CRNsJKSnbMoyRUU14t.png",
          "price": 1000
        }
      ]
    }
  }
}
```

Додаток 3 (запит на створення замовлення)

Запит:

```
mutation CreateOrder {
  createOrder(options: {
    email: "i.kurochkin@ukma.edu.ua",
    items: [
      { amount: 1, productId: "2Mv3oupbVM0VVPcH9Y2m" },
      { amount: 2, productId: "e6CRNsJKSnbMoyRUU14t" }
    ]
  }) {
    id
    items {
      amount
      product {
        id, name, photo, price
      }
    }
  }
}
```

Відповідь сервера:

```
{
  "data": {
    "createOrder": {
      "id": "0drIKaaEAYCiAw6YxU6N",
      "items": [
        {
          "amount": 1,
          "product": {
            "id": "2Mv3oupbVM0VVPcH9Y2m",
            "name": "iPhone X 256gb Space Gray",
            "photo": "https://google.cloudstorage.com/photo-2Mv3oupbVM0VVPcH9Y2m.png",
            "price": 1000
          }
        },
        {
          "amount": 2,
          "product": {
            "id": "e6CRNsJKSnbMoyRUU14t",
            "name": "Apple Lightning to USB Cable",
            "photo": "https://google.cloudstorage.com/photo-e6CRNsJKSnbMoyRUU14t.png",
            "price": 1000
          }
        }
      ]
    }
  }
}
```

Додаток 4 (запит на додавання нового продукту адміністратором)

Запит:

```
mutation CreateProduct {
  createProduct(options: {
    name: "Intel Core i7",
    description:
      "Intel Core i7 is a line of Intel CPUs which span eight generations
of Intel chipsets.",
    price: 2400,
    photo: "https://intel.core/photos/i7.png",
  }) {
    id
    name
  }
}
```

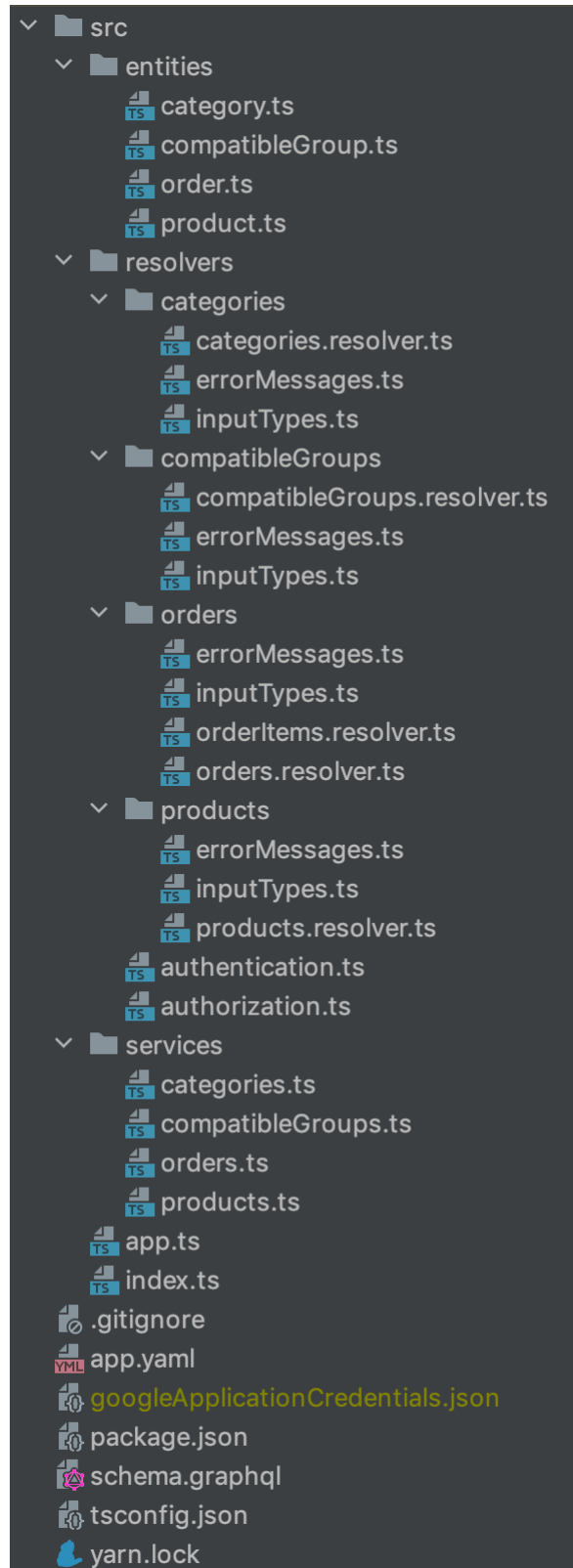
Заголовки:

```
{
  "Authorization":
"eyJhbGciOiJSUzI1NiIsImtpZCI6IjEzZzThkNDVhNDNjYjIyNDIxNTRjN2Y0ZGFmYWMyOTMzMmVhMjAzNzQiLCJ0eXAiOiJKV1QiLCJpc3MiOiJodHRwczovL2FjY291bnRzLmdvb2dsZS5jb20iLCJhenAiOiI0MDc0MDg3MTgxOTIuYXBwcy5nb29nbGVlc2VyY29udGVudC5jb20iLCJhdWQiOiI0MDc0MDg3MTgxOTIuYXBwcy5nb29nbGVlc2VyY29udGVudC5jb20iLCJzdWIiOiI0MTM2MDQwMjk1OTY1MTAxNjU3NTEiLCJlbWFPbCI6ImlsbHl1Lmt1cm9jaGtpbkBnbWFPbC5jb20iLCJlbWFPbF92ZXJpZmllZCI6dHJ1ZSwiYXRfaGFzaCI6InJ5SyTlCSk5JY0VEWXhoRWO0TnIza3ciLCJpYXQiOiJlMTcwMzY0OTU0ImV4cCI6MTYxNzA0MzU5NX0uJGozlaqJHFeLE_WIXGQ4I-frJ2pPmCvf9J5RzJnTlZnCcQqodqHXBkh5FvwVKugbIjod6aGl2zuITfTtpIpzcud-_4kg2H3020UMRemp7d_eOpvgDIYCwWn6kc7o95GD7r4HgL_BrJNE70K5YStVusrPmC10JxB7UJ4qs26IR3sxh6iVdPY4KSRyAE-DatIsvQa7VldIA-nSatgZzC1WOHfEYEO631NynCFSJF_a223NE7xwxTcQrhNdxjZCqOvpKnRdIO9-8RGh2vUo1lh9t_L3W5DjwpRWOMhfQ0bRGY6QlPS923HuJDB9hz7tnhn_sg7odH-kVcBahrXdzZIGelw"
}
```

Відповідь сервера:

```
{
  "data": {
    "createProduct": {
      "id": "dNTVSHZyzfMAz1AQb2I8",
      "name": "Intel Core i7"
    }
  }
}
```


Додаток 5 (структура проекту бек-енду)



Додаток 6 (код GraphQL API для сервісу замовлень)

/src/resolvers/orders/orders.resolver.ts

```
import {Arg, Authorized, Mutation, Query, Resolver} from "type-graphql";
import {VoidResolver} from 'graphql-scalars';
import {Role} from "../authorization";
import {Order} from "../../entities/order";
import OrdersService from "../../services/orders";
import {CreateOrderInput} from "../inputTypes";
import {ORDER_NOT_FOUND} from "../errorMessages";

@Resolver(() => Order)
export default class OrdersResolver {
  @Authorized(Role.ADMIN)
  @Query(() => Order)
  async getById(@Arg('orderId') orderId: string): Promise<Order> {
    const order = await OrdersService.getById(orderId);

    if(!order) {
      throw new Error(ORDER_NOT_FOUND);
    }

    return order;
  }

  @Authorized(Role.ADMIN)
  @Query(() => [Order])
  async getOrders(@Arg('email', {nullable: true}) email?: string):
  Promise<Order[]> {
    return OrdersService.getOrders(email);
  }

  @Authorized(Role.ADMIN)
  @Mutation(() => VoidResolver)
  async deleteOrder(@Arg('orderId') orderId: string): Promise<void> {
    await OrdersService.deleteOrder(orderId);
  }

  @Mutation(() => Order)
  async createOrder(@Arg('options') options: CreateOrderInput):
  Promise<Order> {
    return OrdersService.createOrder(options);
  }
}
```

Додаток 7 (код опису моделі продукту)

/src/entities/product.ts

```
import {Field, ObjectType} from "type-graphql";
import {Collection, getRepository} from "typeorm";

@ObjectType()
@Collection()
export class Product {
  @Field()
  id!: string;

  @Field()
  name!: string;

  @Field()
  description!: string;

  @Field()
  photo!: string;

  @Field()
  price!: number;

  @Field(() => [String])
  categoriesIds!: string[];
}

export const ProductRepository = getRepository(Product);
```

Додаток 8 (файл googleApplicationCredentials.json)

```
{
  "type": "service_account",
  "project_id": "graphic-tide-305412",
  "private_key_id": "1a063c4a37264ff97f84e9b6b7cc94d893eead6c",
  "private_key": "-----BEGIN PRIVATE KEY-----
\nMIIEvAIBADANBgkqhkiG9w0BAQEFAASCBywggSiAgEAAoIBAQC6vqmqMB6kKbh\n\nNoD6Bc1/a
+It+t/FXp3pUunxLA+hPrIDfggzdNdIAidnU72DG46Stzl0ytsaIe83\n\nw5cu+CIBi2DnkbDA+gnv
1fEGTr8FO56eXkJnBDmN86ZVLWGRN0xS2PQftPJE0q4M\n\nPShQLdL7O3dMC2IUYI3LaAFzq0p9wEA
ych5q+ask/jtkZviArWxu4etAFSGT7g50\n\n1SyULTJh8VDD8d2b1kRTd4Fnr6FXbWLPKYGrZHHI+
IvZRVQzz71cE0xarTsEO+4\n\nXc8Lj/UGweihH5wmhEQudVfjKdFVNKsLd2uebVsOqvg9vpsvbE6jy
IIOZCHKCyxv\n\nncpsvasonAgMBAAECggEASzmQOkzkS7ETwIRHWBExfSeMaLGq6fjDF1+oc5ENvN9w
\n44FDIbdNEHJ1Hdz7LuIc0lXqgroqQnMi0Af+G2pICgvGRTE9jieiQc5Cxb/+triTo\n\n9NpBdCM7h
m4ViyVUqBkaUmLwBRbONXh0flrWEy71IjFRBXqkSTLestE0pFCKMgms\n\nYnXQpvSITI7IjDiQZc5Z
mcDdMK4G05pflK2ywVXggvnRVPS6NkjYWpODbQvRUeu1\n\n61k6o3N2+hh1xK3ypIvBo9FZ1CPMbZz
iwhpO2AQhkiJ9Dm+ZXQf5SjyYjle0JvK\n\n7slIUB8PWWa5Uw1jWEfzD1a5NZp9sBYgUZiJ8Ky0QQ
KBgQDe8+huNCmm9jx+XUZe\n\njx9ldefYrwlSfybwQNTQgxtcE0frakue05hPCuBqIIobmswyJOVjC
CWNzYYftWD1\n\nny543bcyARDUuIYn5SRgRBCRpSYYP8u08HjqloLexd7kMana0Giew4NcwAI4cElFr
\nDe9eZZ7rfdTq29OzYWSHv677MQKBgQDB9K53LRngobRO/dHqxuJeh1spsRp77qb8\n\nnq3cjAAe74
WuyF7BAeZbJ3tycykjZoMxGuYpPs5N0GuHy/XvHpVZdclEbhazs3tv\n\nnERYUMraagf3GE1OTt15+
Bxn/ed2A8Aac4UBmA8e/lbME0WQvZZzEQIhZSQpnw6Rt\n\nnQEolY7YUlwKBgHZPJgJL86TFxLy67HnF
lL4NsOsiQMbk2QhpxwC+cMvOMQ7jX4iL5\n\nnE7S2B9opREBbx4nAbH09wHfcHRVMcYvfnii5ujBCgZ
nmMEo5kGzQC6b3Xht6iYC0\n\nn5dgoTfH9nxPIlw8AiVjCJ06QrNiWm9/cOC52bHt4fXjXiCKDRw6GG
DuxAoGAWW1A\n\nn19BisxC7uwQKqo5CxxT8CnAO/v9dXPiN+MRFzJm19yo26LNne/K+k0gtaLJCHWvk
\n2rS4kvrnHQxE6xtPg8x2mxL6WG7zxZUFMUtkqD6xTIEsRHQKjF8n3IRK3e1Njz3I\n\nnXID3s10Ja
npFEB1wu4fplgk4U/l/saPHzA96X4cCgYAYKmg/pycwwdg0hkQ31GM8\n\nnJR2PPSLJ0f66X604B7ir
FlBM/Q5Fe5Gpj3cS3X2y3Ca+GMuIsDWPzXiVeg7mJMav\n\nntQaFcZHURHxyjBXn9AGhrV7XKu7I7dn
/6F4ErLoQPaNI6i0oxlanvXZgknEwtEpc\n\nn+thViCjAh2p/xZPUUDhnGg==\n\n-----END PRIVATE
KEY-----\n",
  "client_email": "admin-442@graphic-tide-305412.iam.gserviceaccount.com",
  "client_id": "110310214490987894753",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url":
"https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url":
"https://www.googleapis.com/robot/v1/metadata/x509/admin-442%40graphic-tide-
305412.iam.gserviceaccount.com"
}
```

Додаток 9 (файл package.json)

```
{
  "name": "gcp-shop",
  "version": "1.0.0",
  "main": "src/index.ts",
  "license": "MIT",
  "scripts": {
    "start": "ts-node ./src/index.ts",
    "deploy": "gcloud app deploy",
    "generate-readme": "graphql-markdown ./schema.graphql > README.md"
  },
  "devDependencies": {
    "class-validator": "^0.13.1",
    "cross-env": "^7.0.3",
    "graphql-markdown": "^6.0.0",
    "nodemon": "^2.0.7",
    "ts-jest": "^26.5.3",
    "typescript": "^4.2.3"
  },
  "dependencies": {
    "@google-cloud/firestore": "^4.9.6",
    "apollo-server-cloud-functions": "^2.21.1",
    "apollo-server-express": "^2.21.1",
    "express": "^4.17.1",
    "fireorm": "^0.20.0",
    "google-auth-library": "^7.0.2",
    "graphql": "^15.5.0",
    "graphql-scalars": "^1.9.0",
    "reflect-metadata": "^0.1.13",
    "ts-node": "^9.1.1",
    "type-graphql": "^1.1.1"
  }
}
```