**FINAL PROJECT**

This Project Report Submitted in the fulfilment of the requirements for Final-Project Evaluation.



**CHAKKA SUDHEER KUMAR**

**EMP ID: 2320418**

**COHORT CODE: CSD24AIAGP003**

**TASKS:**

1.Create a firestore database

2.Create one collection

3.Add few documents

4.Create python script which will read data from a file to firestore.

**OBJECTIVE:**

To set up a Firestore database, create a collection, add documents to the collection, and develop a Python script to import data from a file into Firestore.

**INTRODUCTION:**

**NOSQL:**

* The term NoSQL, short for “not only SQL,” refers to non-relational databases that use a non-tabular format to store data, rather than in rule-based, relational tables like [relational databases](https://cloud.google.com/learn/what-is-a-relational-database).
* NoSQL databases use a flexible schema model that supports a wide variety of unstructured data, such as documents, key-value, wide columns, graphs, and more.
* Organizations choose NoSQL databases for their flexibility, horizontal scalability, and ease of development.
* Google Cloud’s NoSQL databases [Bigtable](https://cloud.google.com/bigtable), [Memorystore](https://cloud.google.com/memorystore), and [Firestore](https://cloud.google.com/firestore) can help to enhance applications and provide great customer experiences without worrying about scalability, reliability, or frequent data changes.

**Types of NOSQL Data:**

1. Document databases.
2. Column oriented databases.
3. Key-Value databases.
4. Graph databases.
5. In-memory databases.

**CAP Theorem:**

This theorem states that a distributed database system can only guarantee two out of the following three properties:

* **Consistency (C):** Every read operation returns the most recent data written.
* **Availability (A):** Every read/write operation succeeds in a fixed amount of time.
* **Partition Tolerance (P):** The system continues to function even when network partitions occur.

**Firestore's Choice (AP):** Firestore prioritizes Availability and Partition Tolerance over strict Consistency. This means:

* Firestore offers high availability, ensuring reads and writes succeed even during network disruptions.
* Data consistency is eventual, meaning it might take a short while for all replicas to reflect the latest update.

**Google Cloud Platform (GCP)**

Google Cloud Platform (GCP). offered by Google, is a suite of cloud computing services that Provides a series of modular cloud services including computing, data storage, data analytics, and machine learning, artificial intelligence alongside a set of management tools.

Gcp run on the same infrastructure that Google uses internally for its own products, such as Google Search and YouTube.

GCP is designed to be scalable, reliable, and secure. It is also cost-effective, with a pay-as-you-go pricing model.

There are many benefits to using GCP, including:

* Scalability: GCP can scale up or down to meet the demands of your application.
* Reliability: GCP is designed to be highly reliable, with a 99.9% uptime guarantee.
* Security: GCP is one of the most secure cloud platforms available, with a variety of security features to protect your data.
* Cost-effectiveness: GCP is a cost-effective cloud platform, with a pay-as-you-go pricing model.

GCP can be used for a wide variety of applications, including:

* Web and mobile applications: GCP can be used to host web and mobile applications.
* Data storage and analytics: GCP can be used to store and analyze data.
* Machine learning and artificial intelligence: GCP can be used to develop and deploy machine learning and artificial intelligence models.

**Firestore:**

Firestore is a NoSQL document database built for automatic scaling, high performance, and ease of application development. While the Firestore interface has many of the same features as traditional databases, as a NoSQL database it differs from them in the way it describes relationships between data objects.

It is used for mobile, web, and server development. It’s built on Google Cloud infrastructure and is part of the Firebase platform.

* Unlike traditional relational databases, Firestore doesn’t rely on fixed schemas. Instead, it uses a document-based model where data is stored in documents within collections.
* Firestore keeps your data in sync across client apps through realtime listeners and offers offline support, making it ideal for responsive applications.

**When to Use Firestore:**

A diagram of a diagram

Description automatically generated

Consider using Firestore in the following scenarios:

* Your application stores and manages **unstructured or semi-structured data**.
* You need a database with **flexible schema** that can accommodate evolving data structures.
* Your app requires **real-time data synchronization** across devices (mobile, web).
* **Fast reads and writes** are crucial for your application's performance.
* You need an **offline-capable** database for mobile and web apps.
* Your project's data model aligns well with **documents and collections**, the fundamental data structures in Firestore.

By following these guidelines, you can make an informed decision about whether Firestore is the best fit for your project's database requirements.

**Why Firestore?**

* **Simplicity**: Firestore simplifies data modeling and provides a flexible schema.
* **Realtime Sync**: It excels at keeping data in sync across devices.
* **Offline Capabilities**: Firestore works seamlessly offline, crucial for mobile apps.
* **Integration**: It integrates well with other Firebase services and Google Cloud products.
* **Scalability**: Firestore is designed to handle large-scale workloads.

**WAYS TO INTERACT WITH GCP:**

Google Cloud gives you three basic ways to interact with the services and resources.

**1.Google cloud console (GUI):**

* The **Google Cloud Console** is a web-based interface that allows you to manage GCP resources visually.
* You can create, configure, and monitor services, virtual machines, databases, and more.
* It’s beginner-friendly and provides an intuitive way to interact with GCP.
* When you use the Google Cloud console, you either create a new project or choose an existing project, and then use the resources that you create in the context of that project.

A screenshot of a computer

Description automatically generated

**2. Command-Line Interface (CLI)**:

* If you prefer to work at the command line, you can perform most Google Cloud tasks by using the google cloud cli. The gcloud CLI lets you manage development workflow and Google Cloud resources in a terminal window.
* The **gcloud CLI tool**, part of the **Google Cloud SDK**, lets you interact with GCP services via the command line.
* Use it for tasks like creating instances, managing storage, configuring networking, and deploying applications.
* The CLI is powerful for scripting and automation.

A screenshot of a computer

Description automatically generated

**3. Python Client Libraries**:

* GCP offers **Python client libraries** for various services.
* You can use these libraries in your Python code to interact with GCP programmatically.
* For example, you can create virtual machines, manage storage buckets, and access APIs directly from your Python scripts.

A screenshot of a computer

Description automatically generated

**IMPLEMENTATION:**

**Task 1**: Create a Firestore Database

There are two modes in firestore while creating database. They are:

* + - Native mode
    - Datastore mode

The main difference is:

1. **Firestore in Native mode**:
   * Next major version of Datastore.
   * Combines Datastore and Firebase Realtime Database features.
   * [Offers real-time updates, a new data model, and mobile/web client libraries](https://cloud.google.com/datastore/docs/firestore-or-datastore).
2. **Firestore in Datastore mode**:
   * Uses Datastore system behavior but accesses Firestore’s storage layer.
   * Removes Datastore limitations (e.g., eventual consistency, transaction limits).
   * [Disables Firestore features not compatible with Datastore](https://cloud.google.com/datastore/docs/firestore-or-datastore).
3. **Use Cases**:
   * **Native mode**: Ideal for real-time apps, scalability, and strong consistency.
   * [**Datastore mode**: Suitable for flexible data modeling and hierarchical data](https://snigdhasambit.medium.com/comparing-google-cloud-nosql-and-caching-databases-redis-firestore-native-and-datastore-mode-335ef5224560).

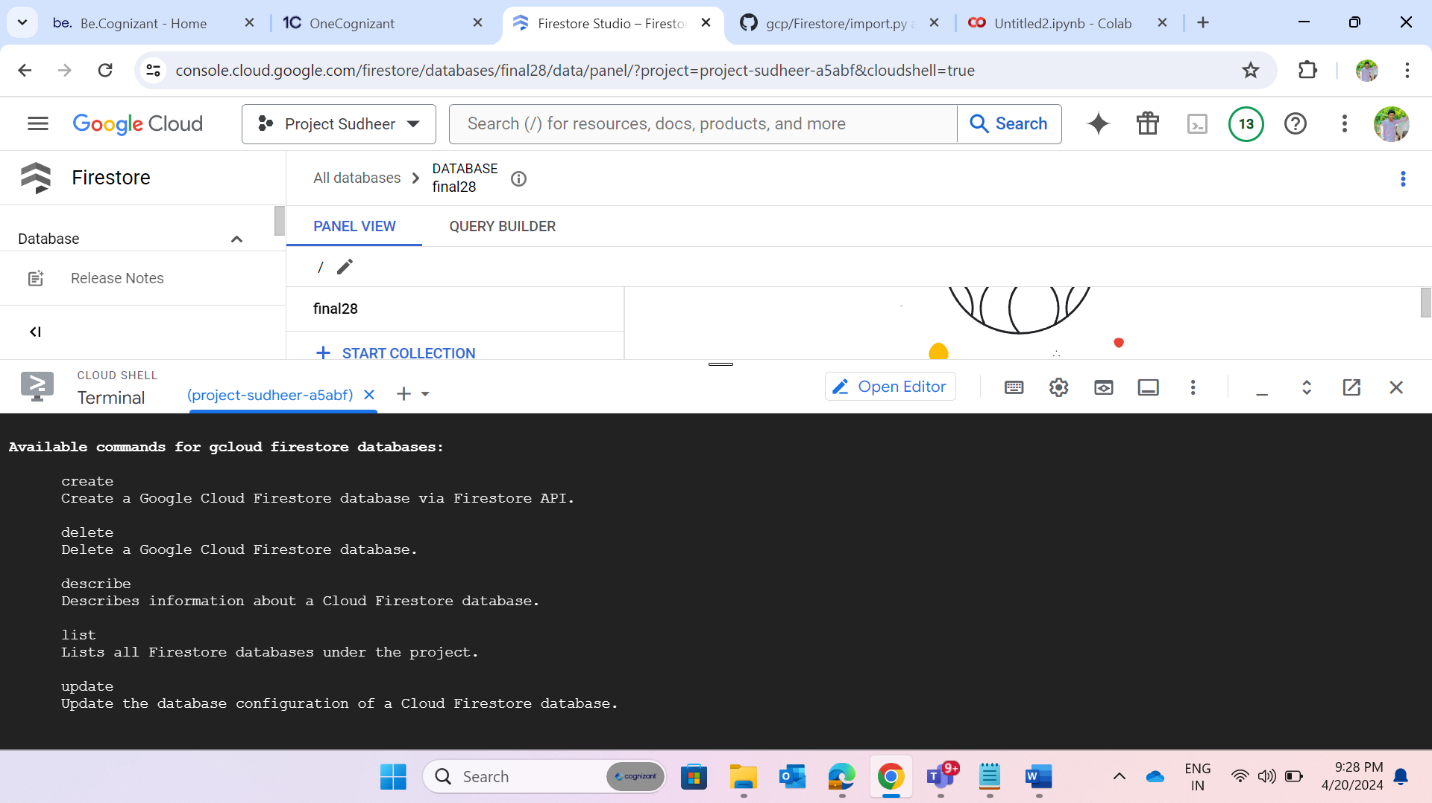
GUI Interactive way:

Select Native mode and create database



CLI way:

In command line interface way, it is limited to do certain database operations in firestore.



Command to create database in firestore is:

gcloud firestore databases create –database=DATABASE\_ID –location=LOCATION –type=DATABASE\_TYPE \  
[--delete-protection]

**A screenshot of a computer

Description automatically generated**

**Task 2:** Creation of collection

Collection creation is possible only with GUI and Python way but we can’t create collections and documents using CLI.

A screenshot of a computer

Description automatically generated

**Task 3:** Add few documents in the collection.

Add few documents to existing collection and documents has its own unique id.

A screenshot of a computer

Description automatically generated

**Task 4**: Create python script which will read data from a file to firestore.

* Create json file of student data by converting csv data to json ORcreate json file of student data from available public api url’s.
* Typecast the required fields to integers while creating json file.

A screenshot of a computer

Description automatically generated

Google collab notebook link for json file creation:

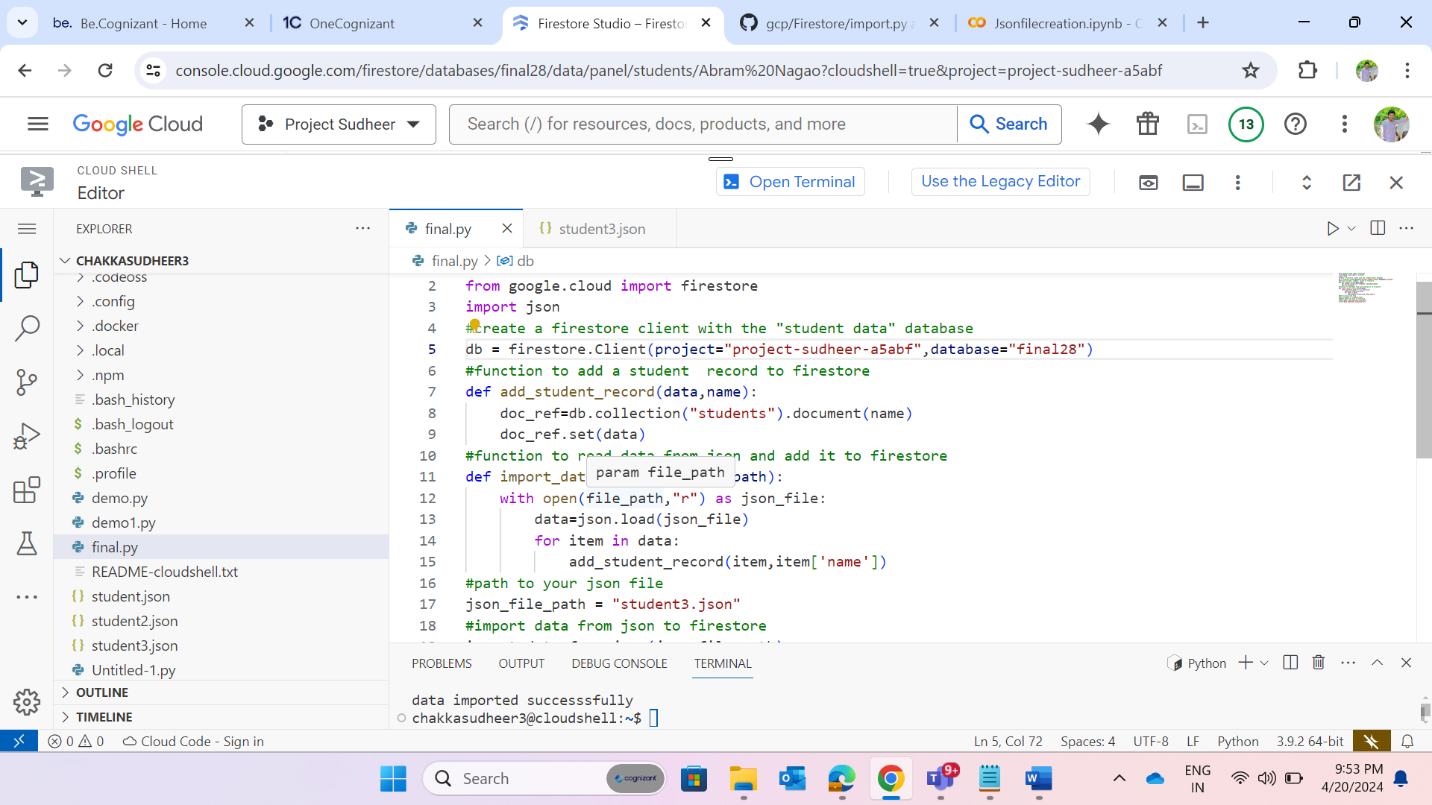
<https://colab.research.google.com/drive/1DFkVuNyKyBjskNxZ9lx5U9HmMHSNg0vi#scrollTo=v4kaK5aajSU7&line=1&uniqifier=1>

Github link for python codes used in this project:

<https://github.com/chsudheer3/gcp1/tree/main/import.py>

* Open cloud shell editor in gcp console,
* Below python code load the json data into existing collection.
* Here each row of data will be stored as individual document with document name as any field in data like stdname, rollno etc.,
* After loading the documents, we can do querying in console, but it is limited to some extent.

Here is the below python code that will load json data to collection as documents with document name as student name.



Here the documents looks like below, each document is named with the student name.

We can add collections inside documents and we can maintain multiple collections at a time.

**A screenshot of a computer

Description automatically generated**

**Additional Tasks:**

1. **Import/Export:**

Exporting the data in firestore to cloud storage bucket, bucket will store only metadata of original data in firestore.

A screenshot of a computer

Description automatically generated

Export the students collection data to bucket:

A screenshot of a computer

Description automatically generated

We can also export the data in firestore to buckets using CLI method also

Here is the following command to export :

$ gcloud firestore export ‘sourcefile:firestore’ ‘destfile:bucket’

A screenshot of a computer

Description automatically generated

Google cloud storage bucket gets metadata of firestore data after exporting.

A screenshot of a computer

Description automatically generated

We can import this metadata back to firestore from bucket by importing.

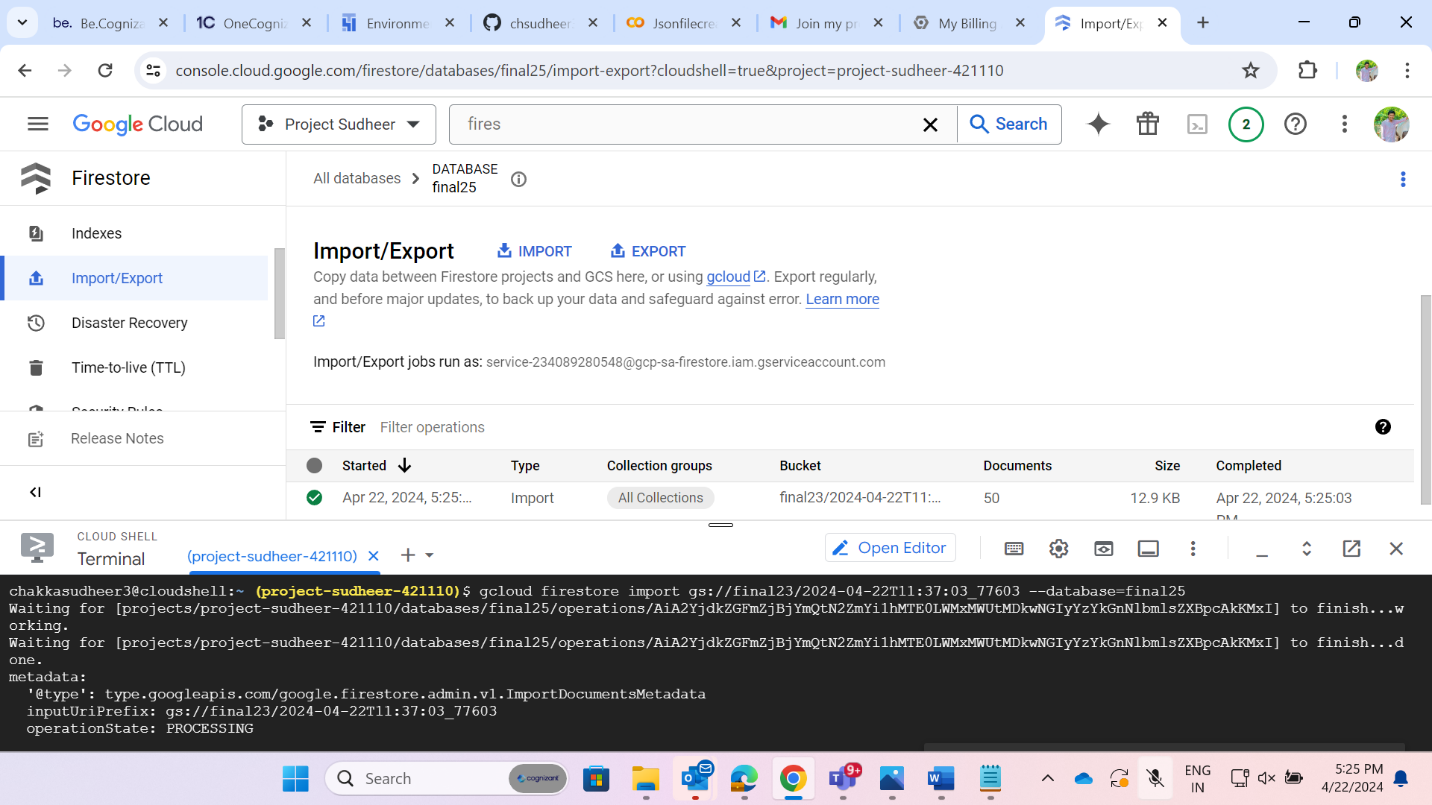
We can only import the data that is exported previously from firestore.

We can do import by gui console method as mentioned above.

Importing of metadata from bucket to firestore database through CLI way is:

Here is the following command to import:

$ gcloud firestore import ‘sourcefile:bucket’ destfile:firestore’



1. **Indexing:**

An index in Firestore is a way to organize and optimize data for faster queries. Indexes are created on specific fields in a collection, and they allow Firestore to quickly look up documents based on the values of those fields.

Indexes are especially useful for queries that filter or sort data based on a specific field. For example, if you have a collection of users and you want to query all users who live in a specific city, you can create an index on the city field. This will allow Firestore to quickly look up all users who live in that city, without having to scan the entire collection.

Single-field index:

A single-field index is an index that is created on a single field in a collection. Single-field indexes are the most basic type of index, and they can be used to improve the performance of queries that filter or sort data based on a specific field.

Composite index:

A composite index is an index that is created on multiple fields in a collection. Composite indexes are more complex than single-field indexes, but they can be used to improve the performance of queries that filter or sort data based on multiple fields.

Creation of composite index using gui console and cli command way:

While creation of composite index, we include specific fields of data that are going to be used for querying.

GUI way of composite index creation:

A screenshot of a computer

Description automatically generated

CLI command to create composite index is:

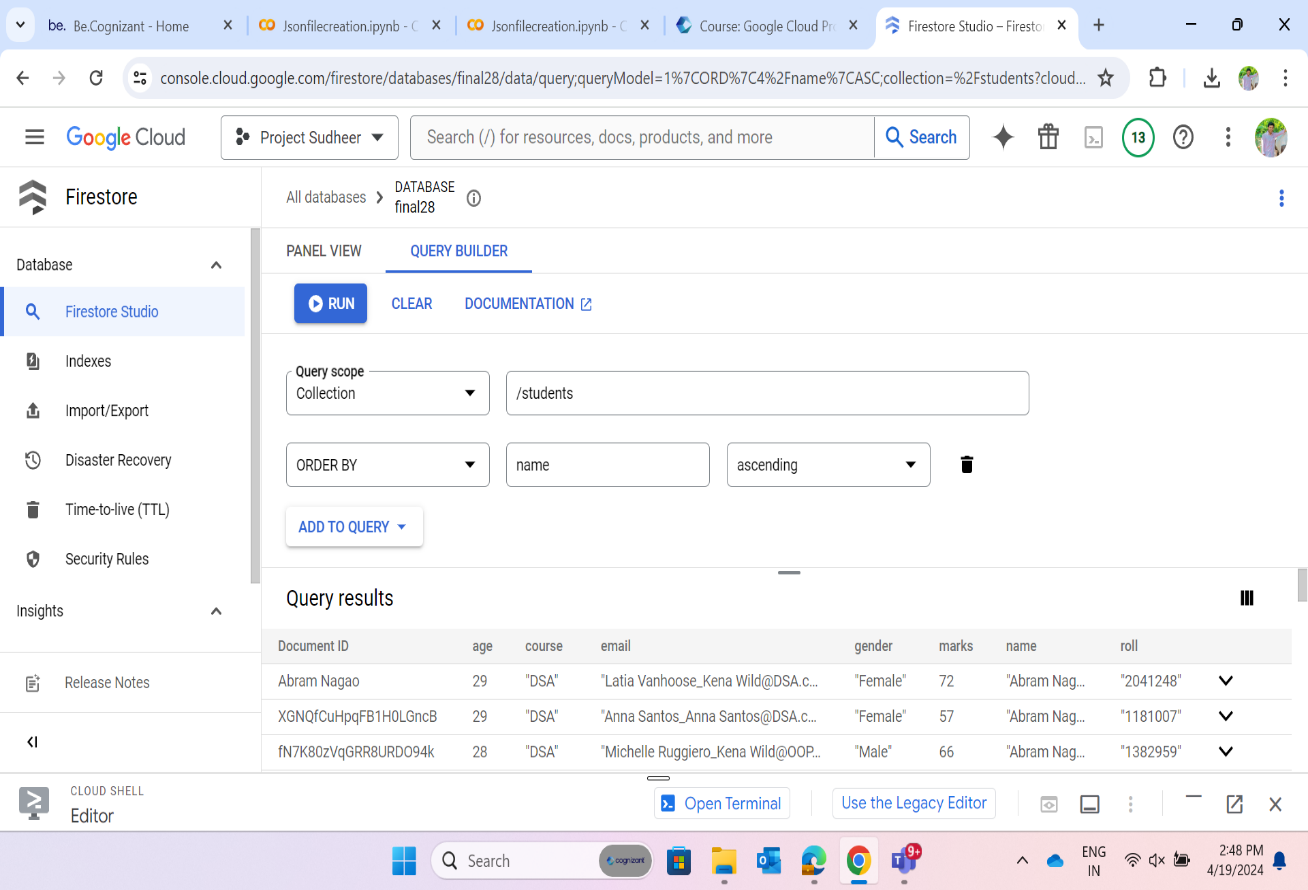
$ gcloud firestore indexes composite create --database=’databasename’ \  
     --collection-group=Events --field-config=field-path=user-id, order=ascending \  
     --field-config=field-path=timestamp, order=descending

A screenshot of a computer

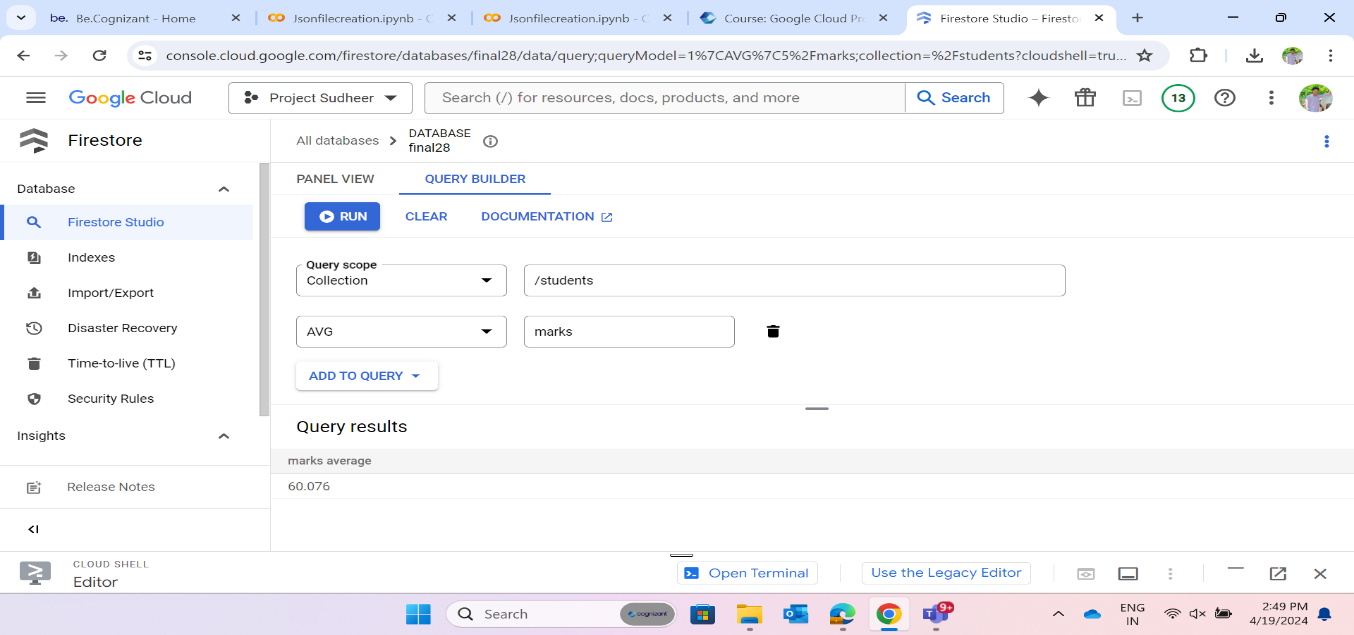
Description automatically generated

1. **Querying:**

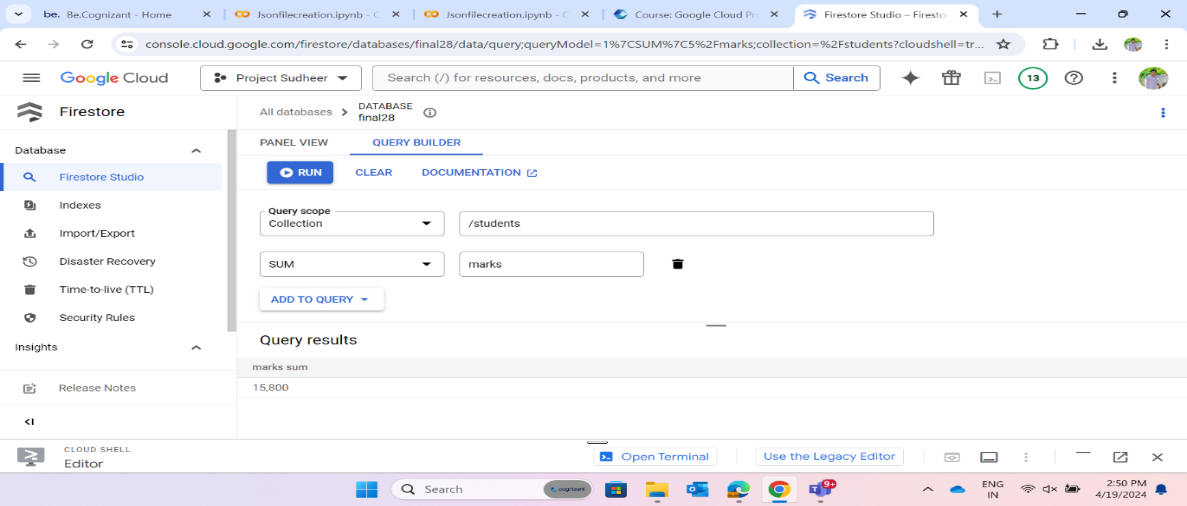
* Query that orders the data based on student name in ascending order.

****

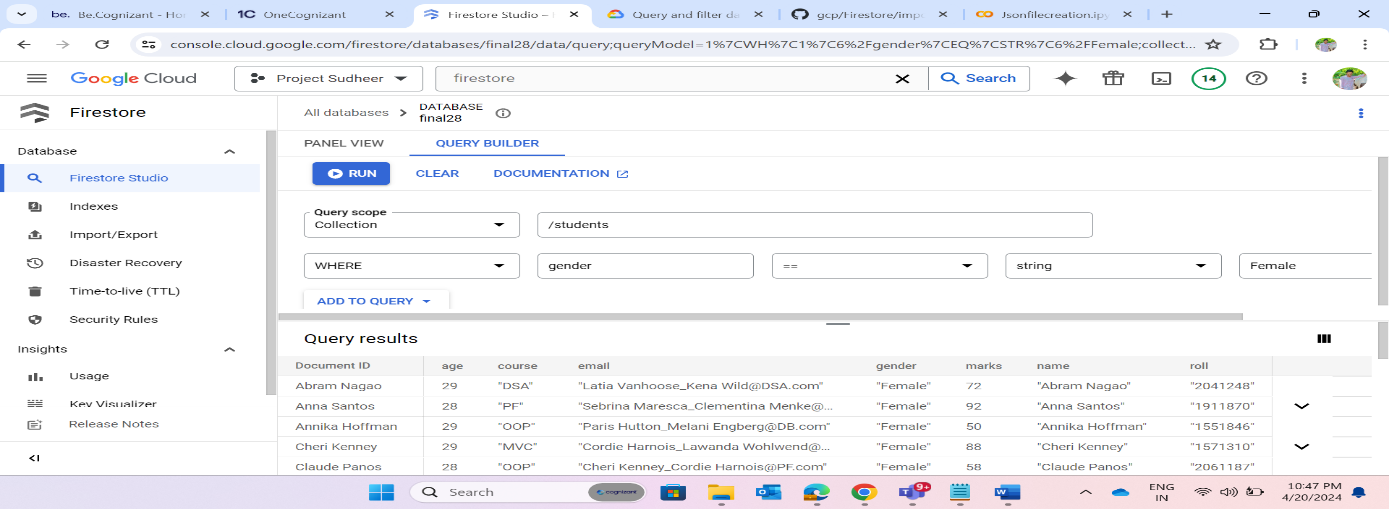
* Query that returns average marks of all students

****

* Query that returns the sum of marks of all students

****

* Query that returns records of all female students

****

* + Query that uses above created composite index for where condition as follows:

A screenshot of a computer

Description automatically generated

1. **Time-to-Live (TTL):**

* TTL policies are used to schedule expiration and bulk deletion of documents. Deletion jobs run continuously in the background.
* Documents are scheduled for deletion based on a timestamp field in the specified collection group.
* Data is typically deleted within 72 hours after its expiration date.

**Applications of Firestore:**

* Real-time applications: Firestore is ideal for building real-time applications because it provides real-time updates to data. This means that changes to data are reflected in the application immediately, without the need to refresh the page.
* Offline applications: Firestore can be used to build offline applications because it supports offline data storage. This means that users can access data even when they are not connected to the internet.
* Scalable applications: Firestore is a scalable database that can handle large amounts of data. It is designed to handle the demands of high-traffic applications.
* Flexible applications: Firestore is a flexible database that can be used to store a variety of data types. It supports documents, collections, and subcollections.
* Easy-to-use applications: Firestore is an easy-to-use database that is designed for developers of all levels. It provides a simple and intuitive API that makes it easy to get started with.

Here are some specific examples of how Firestore can be used:

* A social media application: Firestore can be used to store user profiles, posts, and comments. The real-time updates provided by Firestore ensure that users see the latest content immediately.
* A chat application: Firestore can be used to store chat messages and user information. The offline support provided by Firestore ensures that users can continue to chat even when they are not connected to the internet.
* An e-commerce application: Firestore can be used to store product information, orders, and customer information. The scalability of Firestore ensures that the application can handle the demands of a high-traffic e-commerce site.
* A gaming application: Firestore can be used to store game data, such as player profiles, scores, and levels. The flexibility of Firestore makes it easy to store a variety of data types.

**Advantages:**

1. Scalability: Firestore scales automatically with your application's usage, allowing it to handle large amounts of data and high traffic without manual intervention.

2. Real-time Updates: Firestore provides real-time synchronization, enabling instant updates to data across multiple clients and devices without the need for manual refreshing.

3. Serverless: As a serverless solution, Firestore abstracts away server management tasks, reducing operational overhead and allowing you to focus on application development.

4. Flexible Data Model: With its NoSQL document-based structure, Firestore offers flexibility in data modeling, allowing you to store nested data, arrays, and complex objects without the constraints of a traditional relational database.

5. Integration with Google Cloud: Firestore seamlessly integrates with other Google Cloud services such as Firebase Authentication, Cloud Functions, and Cloud Storage, enabling a comprehensive ecosystem for building and deploying applications.

**Disadvantages:**

1. Cost: While Firestore offers a generous free tier, costs can scale with usage, especially for high-traffic applications or large datasets. It's important to monitor usage and optimize queries to minimize costs.

2. Query Limitations: Firestore imposes some limitations on queries, such as the inability to perform complex joins or aggregations. This can require denormalizing data or using additional services for certain use cases.

3. Vendor Lock-in: Choosing Firestore ties your application to the Google Cloud Platform ecosystem, which may limit flexibility if you decide to migrate to a different provider in the future.

**Conclusion:**

Our project successfully explored Google Firestore services, covering database creation, collection management, and data import. By leveraging Firestore’s real-time capabilities, we can build responsive applications that handle both transactional and analytical workloads. Firestore’s scalability, security, and ease of use make it a powerful choice for modern applications.

Top of Form

Bottom of Form