**ASSIGNMENT 6 – CLOUD FREE FOR ALL**

**INTRODUCTION**

This assignment focuses on developing software applications on GCP. In previous versions we have mainly focused on developing applications that utilize GCP to be able to achieve a single task. For this assignment, the focus was to utilize the GCP to build a whole application.

I have built a personalized note taking app, usable for any students and professionals alike, on storing important notes, such as lectures, meeting and so on. The application utilizes many GCP API’s for its implementation and deployment. It mainly focuses on providing the most scalable and distributed method in deploying the application on the cloud. I have chosen such a basic application as to easily utilize the application development to properly demonstrate the various functionalities of GCP API’s that I have used. The application is complex enough to validate the use of microservices and thus a better way to demonstrate the workings of distributed systems architecture.

The **Notes App** is built as a distributed microservices model, where each service is essentially functioning independently and can scale independently with respect to the load the application will come across. GCP provides us with multiple ways to setup distributed architecture in the cloud. One of the ways is to utilize the Cloud Run, a popular variety of hosting stateless microservices, with the help of containerization.

For this assignment, I really wanted to understand and try a different variety of containerized application, and I opted to demonstrate my understanding with a simple application utilizing Cloud Run. Cloud Run requires a variety of other GCP API’s to enable CI/CD (Continuous Integration/ Continuous Deployment) which have also been utilized through this assignment.

The assignment mainly focuses on three microservices hosted in Cloud run, which renders the whole application frontend and backend layer to the user in an auto scaling manner, which will be discussed in detail in the upcoming sections.

The rest of the report will discuss on the design, implementation, performance, logs, and other weakness of such an application development.

**HOW TO RUN**

The application is auto set to run on GCP cloud run through and through without any manual intervention in the form of local commands. The application frontend has a cloud run initiated URL, which we can hit to interact with the application.

**Notes-App front end URL: https://notes-frontend-lb6yim52dq-uk.a.run.app/**

For editing and updating the application locally, the user must unzip the zipped folder into any local drive and follow the next steps.

Before starting to edit locally, the user needs to have the below requirements satisfied.

1. Python 3.7 or higher.
2. NodeJS/ npm.

**Instructions to run frontend**

* Navigate to the project folder: **cd Cloud\_Free\_for\_all**
* Navigate to the frontend folder: **cd notes-frontend**
* Do the prerequisite installation of libraries: **npm install**
* Run the frontend server locally: **npm start**

The backend microservices are also running on GCP Cloud run, but if the user wishes to edit the backend microservices, they need to follow the below instructions.

**Auth-service**

This microservice handles the authentication required for the application such as login, register for users. For editing, this microservice:

* Navigate to the auth-service folder: **cd Authentication**
* Install the prerequisites: **pip install -r requirements.txt**
* Run the python Flask application: **python3 app.py**

**Notes-service**

This microservice handles the user interaction with notes. It provides the basic CRUD operations on notes, such as to create a note, edit a note, delete a note, and view a note. For editing the microservice:

* Navigate to the notes-service folder: **cd Notes**
* Install the prerequisites: **pip install -r requirements.txt**
* Run the python Flask application: **python3 app.py**

Both the above services utilize GCP firestore for handling data, thus they cannot be run locally without setting up a Gcloud SDK for my individual project. Thus, it is not advisable to run these microservices locally.

But all the above microservices can be edited, updated, and run on Gcloud run with the below commands.

For updating the frontend service and deploying in your own GCloud run environment:

* Update the necessary pages in the frontend service.
* Run the command to build a new image: **gcloud builds submit --tag gcr.io/$PROJECT:ID/notes-frontend**
* Run the command to deploy the cloud run: **gcloud run deploy --image gcr.io/$PROJECT:ID/notes-frontend --platform managed**

Similar steps can be followed to edit and deploy each of the backend microservices as well to their respective cloud run instances.

**NOTE**: The backend cloud run instances URL must be updated in the frontend codebase, to have functional API calls.

**CLOUD API USED**

In this assignment I have utilized a variety of cloud API’s. These have been used for implementation as well as setting up the infrastructure to run the application and so on. The major cloud API’s used are as follows.

1. **GCP Cloud Run**

Cloud Run is basically like the Cloud Functions API that was utilized in the previous assignment. Cloud Run enables the user to **Develop and deploy highly scalable containerized applications on a fully managed serverless platform**. Cloud Run deploys an image stored in the Cloud Registry for GCP (like Docker Hub), in a container which can be auto scaled with respect to load. Cloud run allows users to simply write the application in whichever language they want and utilize the ability to ship containerized application.

1. **GCP Cloud Build**

Cloud Build is a tool which can be used to **perform CI/CD pipelines for the application with ease**. Cloud Build can be utilized to setup build triggers, define custom workflow for building, testing, and deploying applications in GCP. It can also be used to deploy the application across various environments. We have used Cloud Build in specifically two methods.

* Used the cloud build cli to automatically build a microservice into an image which can be stored in the Cloud registry to be accessed from anywhere.
* Setup cloud build triggers to check for source code update in GitHub and triggering a new build for the updated microservice automatically into the cloud registry.

The cloud build effectively allows us to package our application into containers, which can be deployed and distributed to any environment allowing containerized deployment, without changing the code base involved underneath.

I have setup my code base in my personal GitHub, to initiate the triggers. The link to my GitHub repo: <https://github.com/gouthamgopal/GCP_Notes_App>.

1. **GCP Firestore**

Cloud Firestore allows users to develop applications utilizing **fully managed, scalable, and serverless document database**. Firestore enables users to store collections and documents in the database, utilizing the power of not having to adhere to datatypes and format. It resembles much with the popular NoSQL database MongoDB, allowing easy interaction and looking up of data with queries. I used Firestore to enable myself to store data in the form of json, for easy lookup as well as editing the data through my microservice. GCP provides easy python libraries to enable the users to interact with Firestore, which I have also utilized.

My Firestore mainly has 2 collections, **users,** and **notes.** The user’s collection maintains the list of registered users, and the notes collection maintains the list of notes with each having their own user attached to the entry.

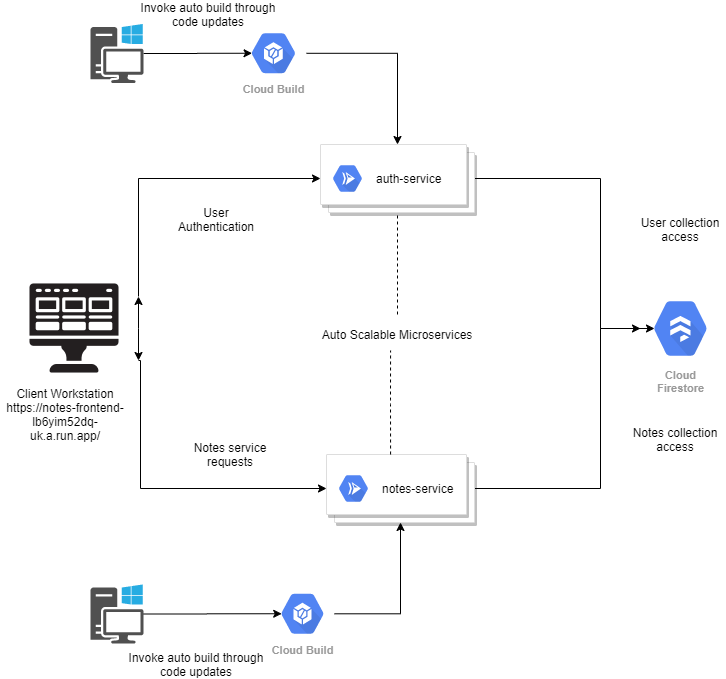
**DETAILED DESIGN**

The application design consists mainly of two sections, the frontend microservice and the two backend microservices along with the cloud infrastructure.

The backend design consists mainly of designing the python flask application which can interact with the cloud Firestore as well as be containerized using cloud build and shipped to cloud run for deployment.

The design basically focuses on connectivity and scalability of application with respect to user loads. For this we can imagine the application gaining multiple instances deployed through cloud run, and scaling in once the load has substantially decreased.

The application connectivity can be seen in the below diagram.



The application is mainly designed to interact with the backend microservices through HTTP requests raised by the frontend triggered with user interaction. The backend microservices utilizes the above mentioned Firestore to fetch the data and satisfy the user requirement raised by the HTTP calls.

The two major microservices were separated to follow the microservice architecture as defined by the development of distributed systems. The microservices are individual functionalities which are used to maintain only one aspect of the application. Thus, the two microservices, are deployed to maintain the separation of concerns amongst them, while assisting the frontend user interaction.

1. **auth-service cloud run microservice**

This cloud run deployed microservice, as the name suggests is utilized for ensuring authentication happens properly within the application. It has mainly two functions/ http triggers, login, and register, which are the basic requirements for any application authentication.

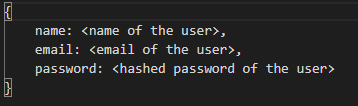
The register function interacts with the Firestore collection (users) to fetch or add the user details specified via the HTTP request.

1. **notes-service cloud run microservice**

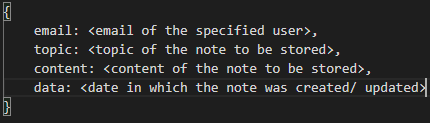
This cloud run deployed microservice, is utilized for ensuring the basic CRUD operations for notes, in the application. The cloud run service is invoked during the add, update, delete operations initiated by the client. The microservice utilizes Firestore similarly to the above function. It has several endpoints, targeting each individual functionality for the user. Various user requests may come to these different endpoints, which will satisfy the user requirements or raise an error with respect to the data specified by the user.

The **cloud Firestore** as mentioned above, is utilized for storing the user related data on the cloud. The microservices use a Firestore specific API to interact with the Firestore DB, which is a NoSQL database. As the application is designed to store data as a key value pair, we have created collections to maintain the requirement. Each collection can contain multiple sets of documents inside, which can be the unordered data in json format.

The format for storing user data in Firestore is as below:



The format for storing notes data for each user is as follows:



The Firestore client can access these documents based on the query specified in the microservice. An example of such a query to fetch a specific note for a user is:

docs = notesCollection.where(u'email', u'==', email).where(u'topic', u'==', topic).stream()

Such queries provide the reference object of the Firestore objects, which can then be utilized to aggregate the actual data.

**IMPLEMENTATION**

The implementation of the application can be easily explained by utilizing a live example from the beginning of a user creation to updation of a note added by the user. For this implementation demo, I shall utilize the already deployed frontend microservice url: [**https://notes-frontend-lb6yim52dq-uk.a.run.app/**](https://notes-frontend-lb6yim52dq-uk.a.run.app/)

The application is implemented using React as the front end and python Flask application for the backend microservices. As mentioned earlier, the application has 2 microservices, that are being used to provide request endpoints for both authentication and CRUD operations for notes.

The application frontend consists mainly of 5 pages, including a login, register, view notes, edit note as well as add note page. We shall cover each page as well as the backend functionality used for each of the page below.

1. The application starts off by routing to the home component. If the user is not logged in previously to the browser, the application will re route itself to the login page, as seen below (Fig 1).

The user can enter the email and password to login to the application or use the register button to go to the register page for new registration.

1. The register screen allows the user to register into the application, by entering few details such as email, name, and password. All necessary error handling has been done before registration.

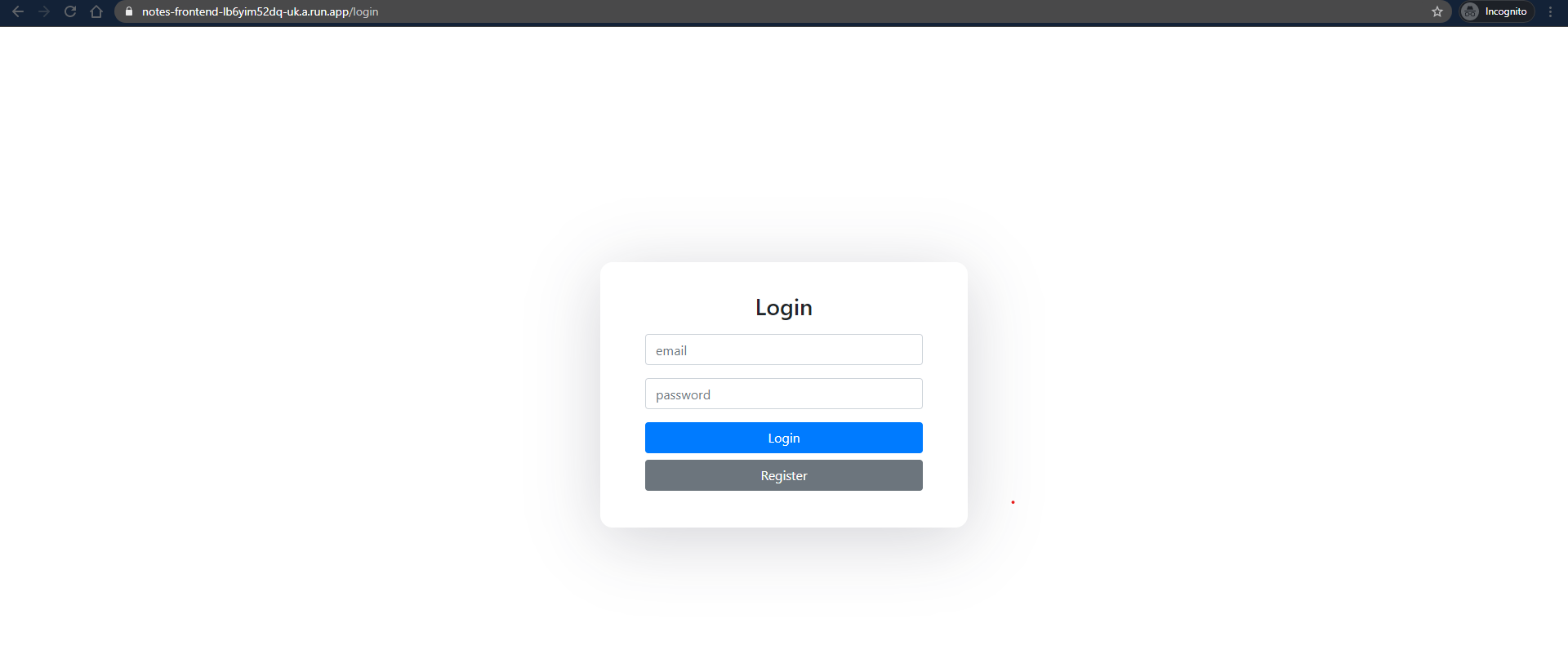


Figure 1: Login Screen

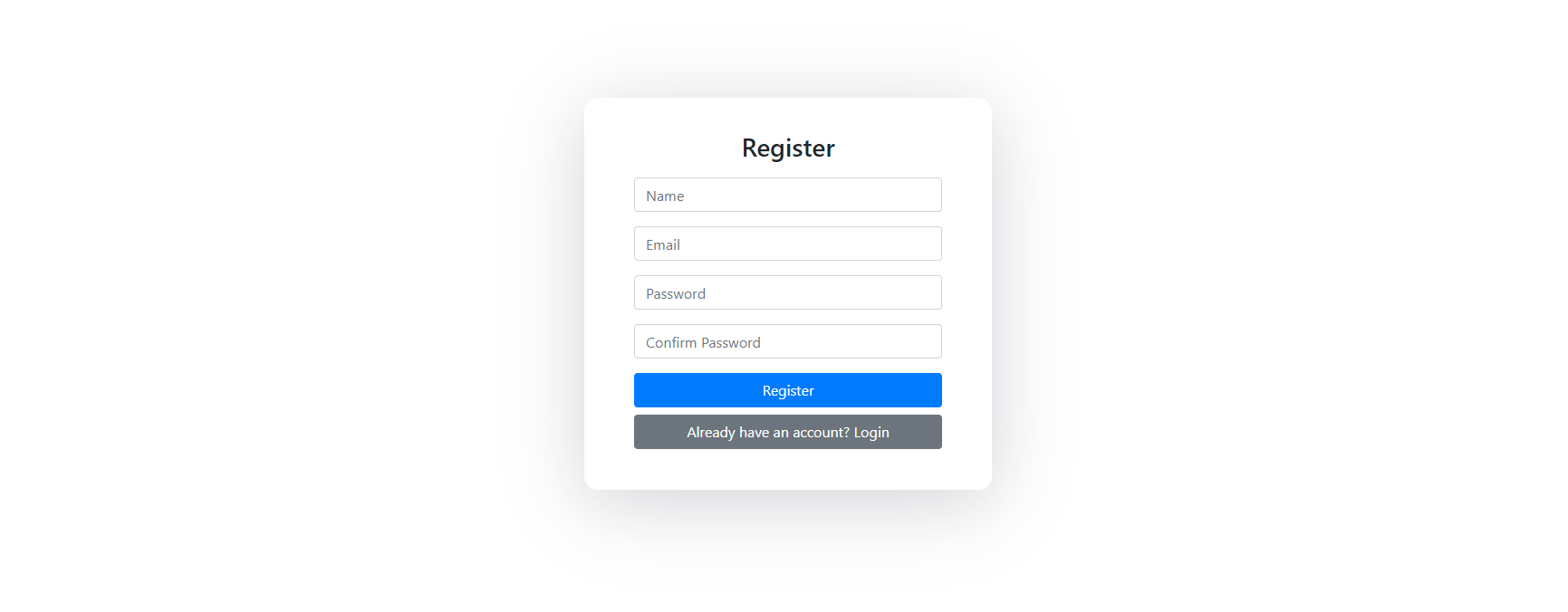
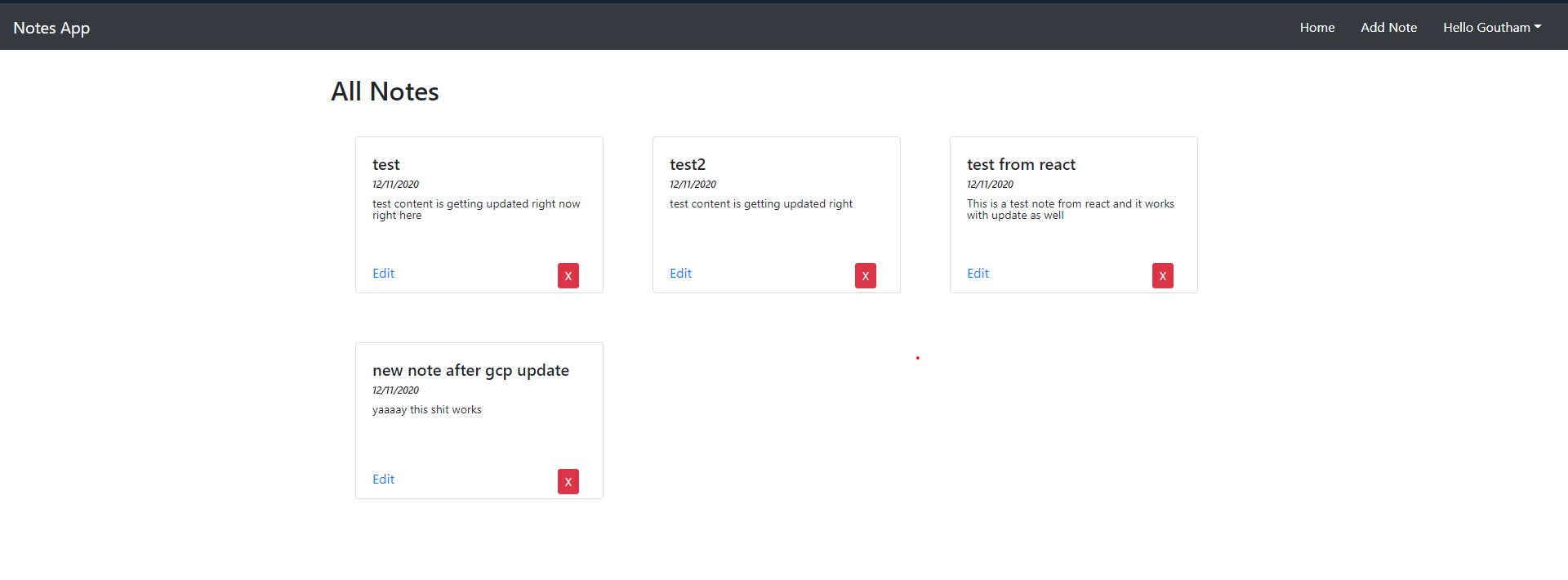


Figure 2: Register Screen

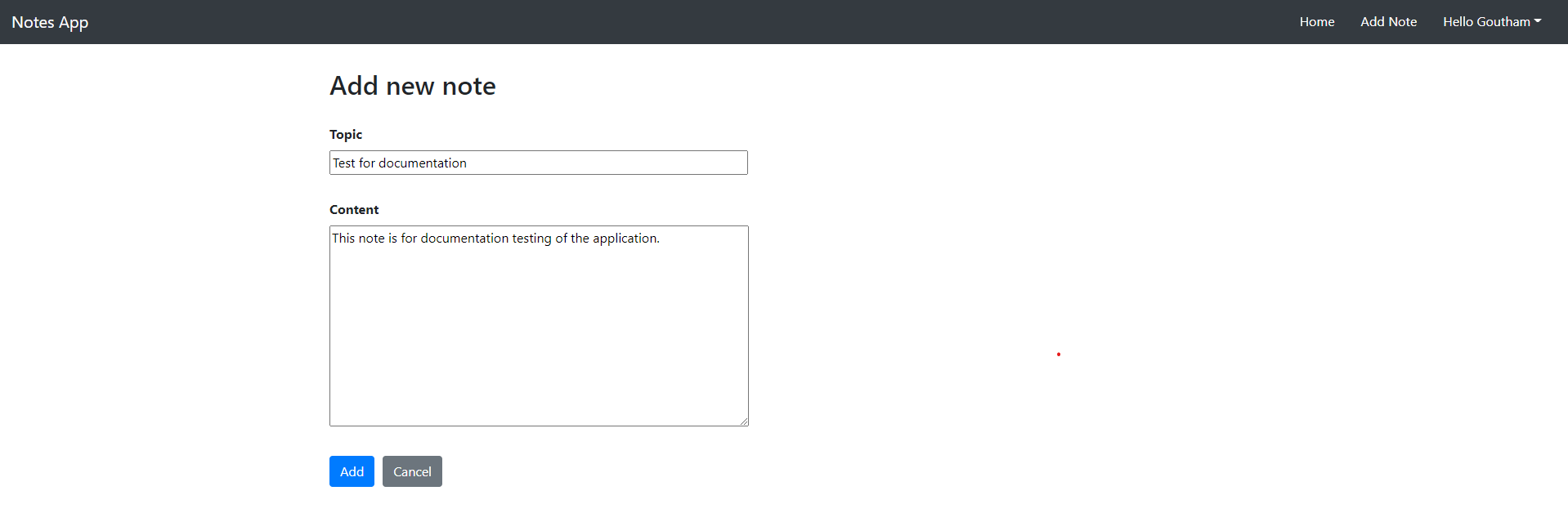
1. Once, the user is logged in, they will be taken to the home screen, where they can view all the notes that they have created till now.



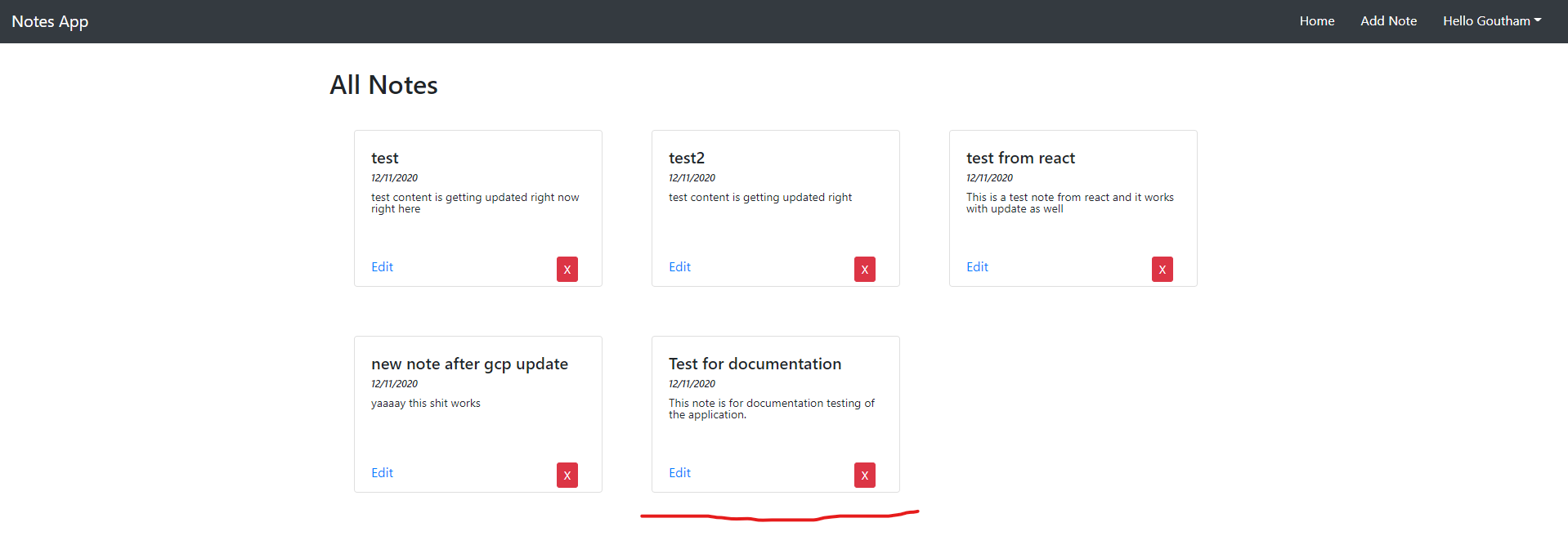
As you can see, all the notes are well placed on the screen, with the user having the ability to navigate through each note, as well as edit or delete the notes that they want.

Each screen will also have a top navigation bar, that they can use to navigate to different pages in the application. The user can click on their name on the top right which will show an option to Logout from the application.

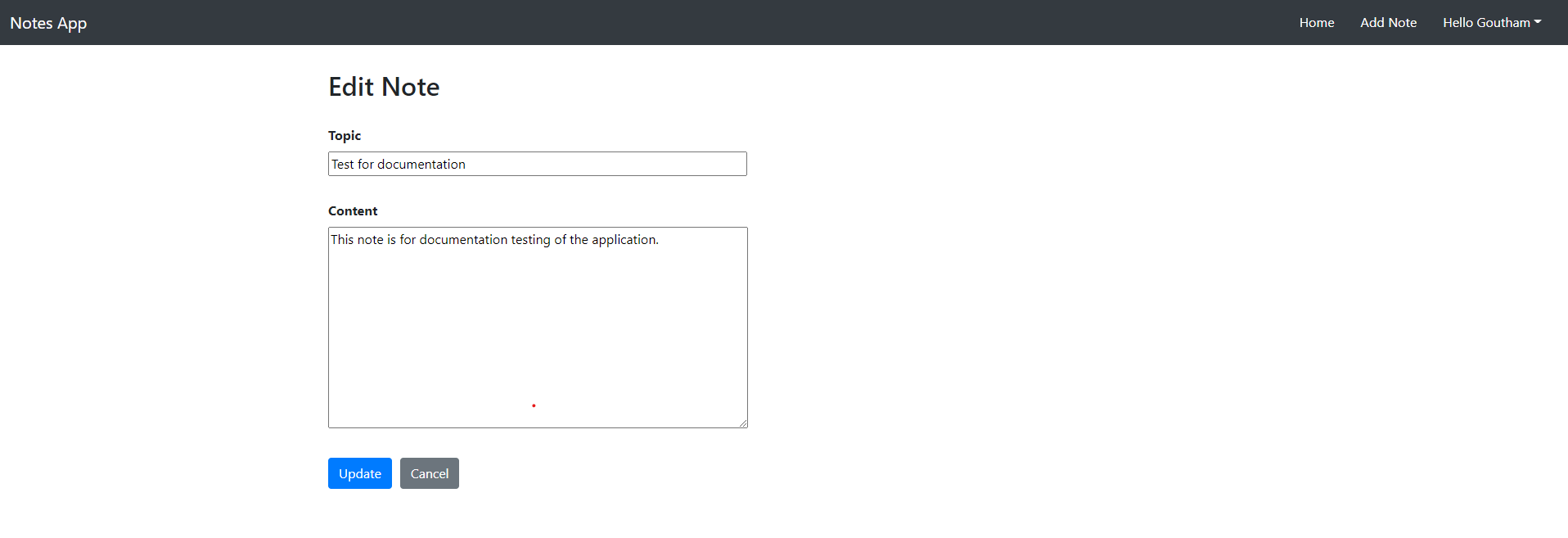
1. The add feature of a new note can be done by clicking the Add Note button on the top right, which will take the user to the add note page as shown below.



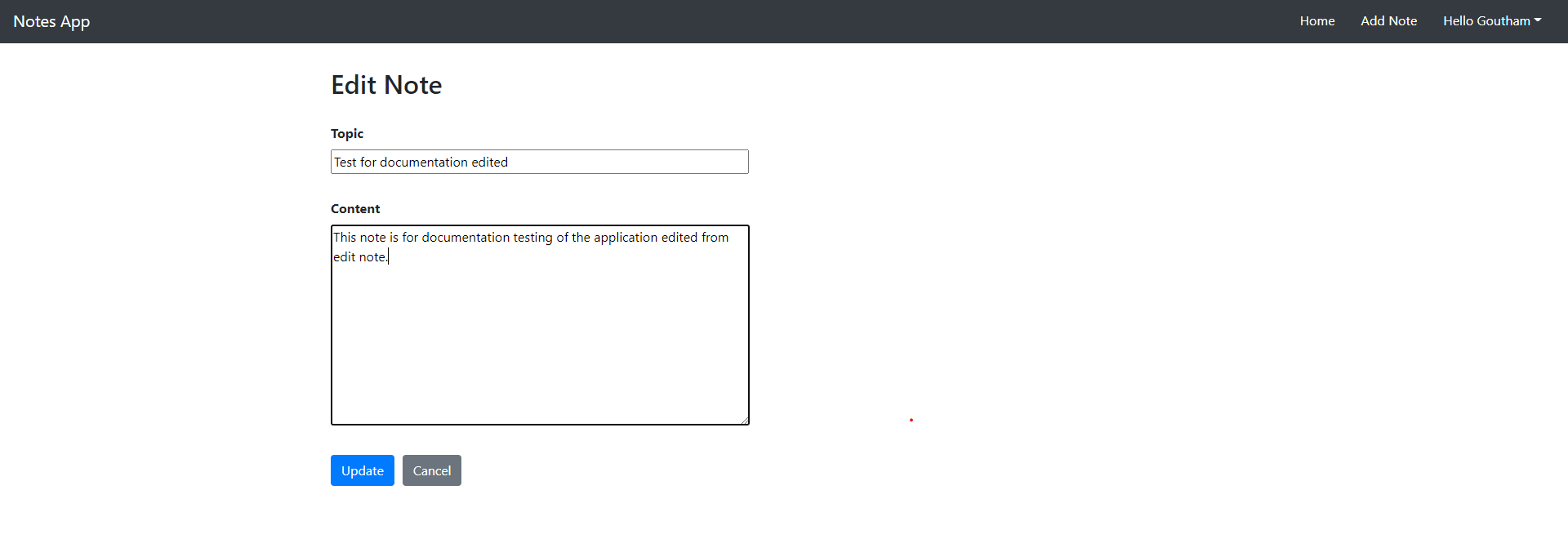
Once the user adds a new note, they will be re routed back to the main screen, where they can see the new note added to the list of notes. All notes are placed with the topic of the note, the date of creation/ updation of the note and the content of the note.

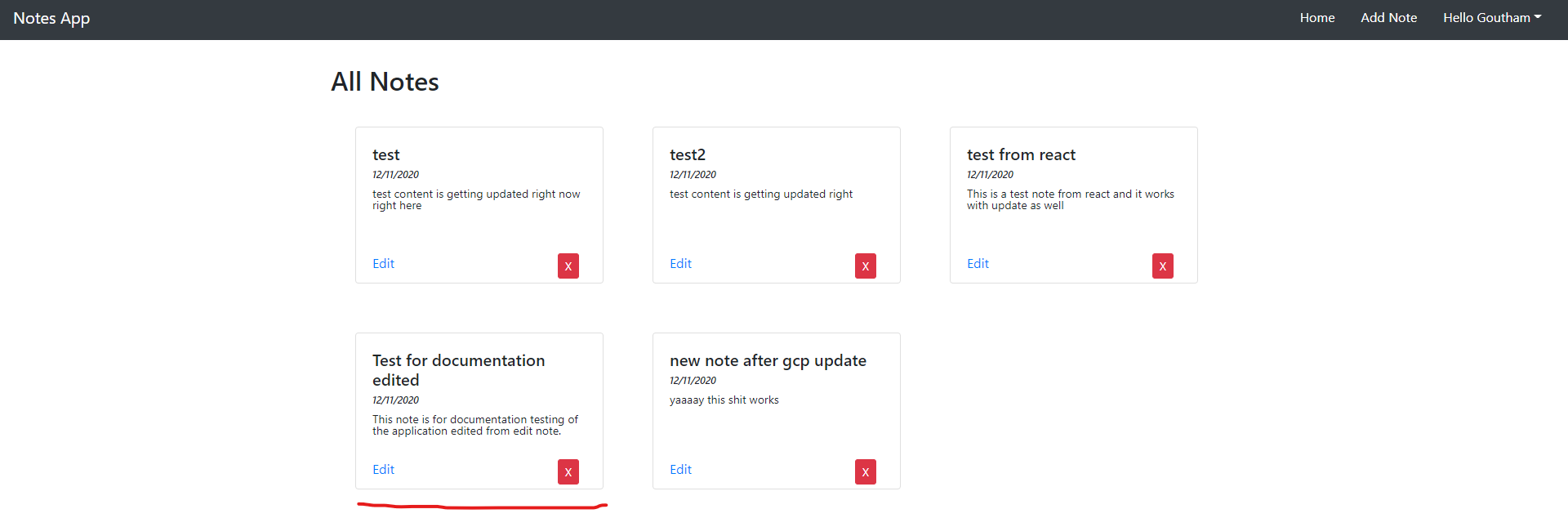


1. If the user wishes to edit any document, they can do so by clicking the edit button on the card, which will take them to the edit screen as shown below.



Here, the user can change the topic as well as the content and see that the notes reflect the updation as well on the home screen.





1. The user can also delete a note and they will just disappear from all notes view for the user.

**Microservices implementation overview**

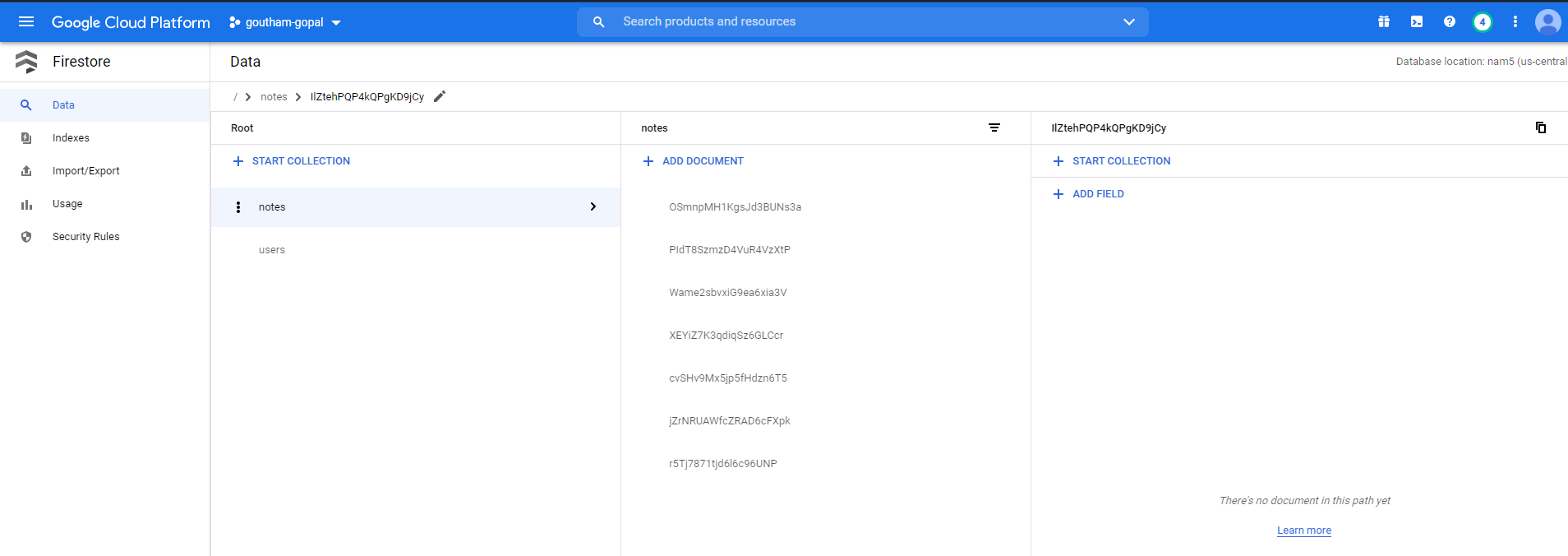
The authentication cloud run microservice, has two endpoints: /login and /register.

These endpoints as their name suggests work towards login or registration of any user to the system. The service utilizes the Firestore collection named ‘users’ to handle the database for the application, managed via a python API reference for the Firestore.

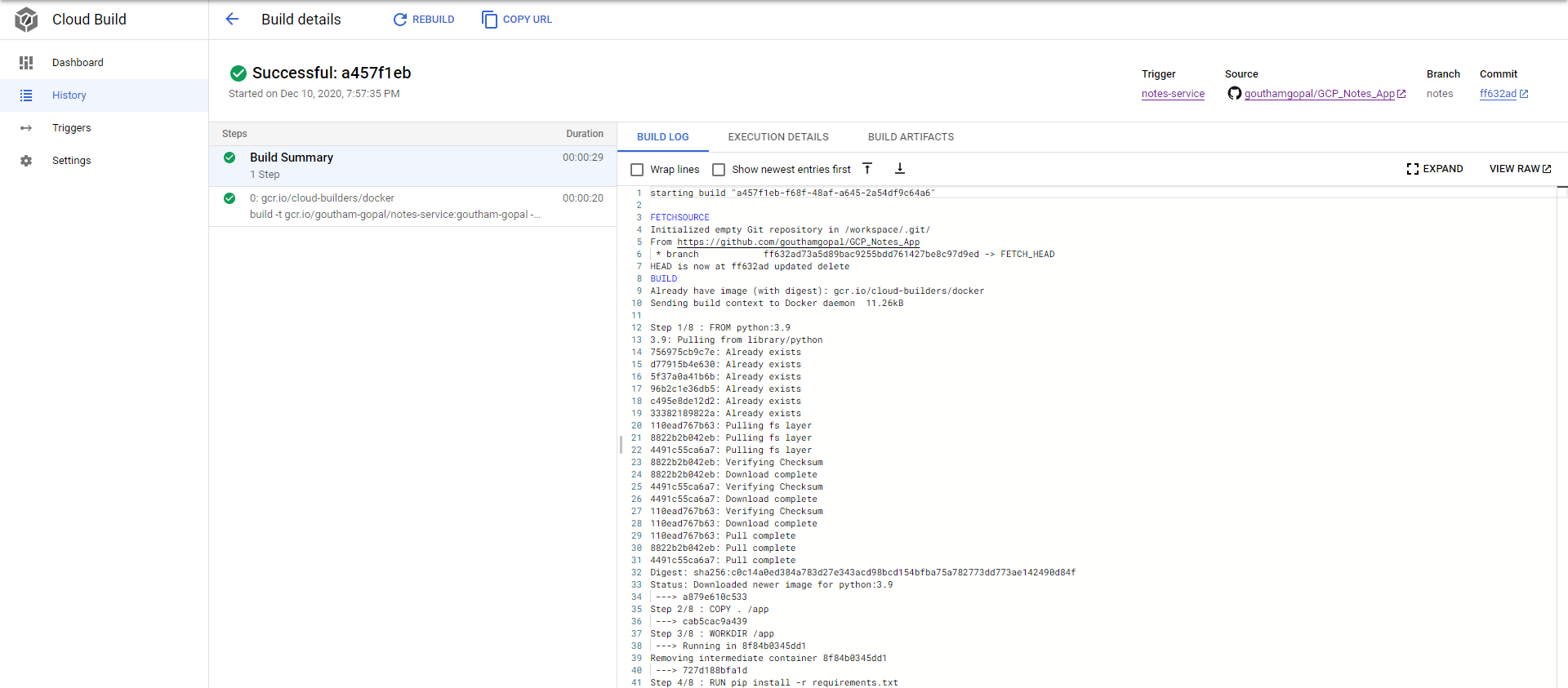
The notes cloud run microservice, has 4 endpoints: /fetch, /edit, /delete and /add.

These endpoints also work as their name suggests for each user, where the requests send in the data as well as the user jwt token created when the user was logged in to the system. This microservice also utilizes the Firestore to keep track of the database of notes for each user.

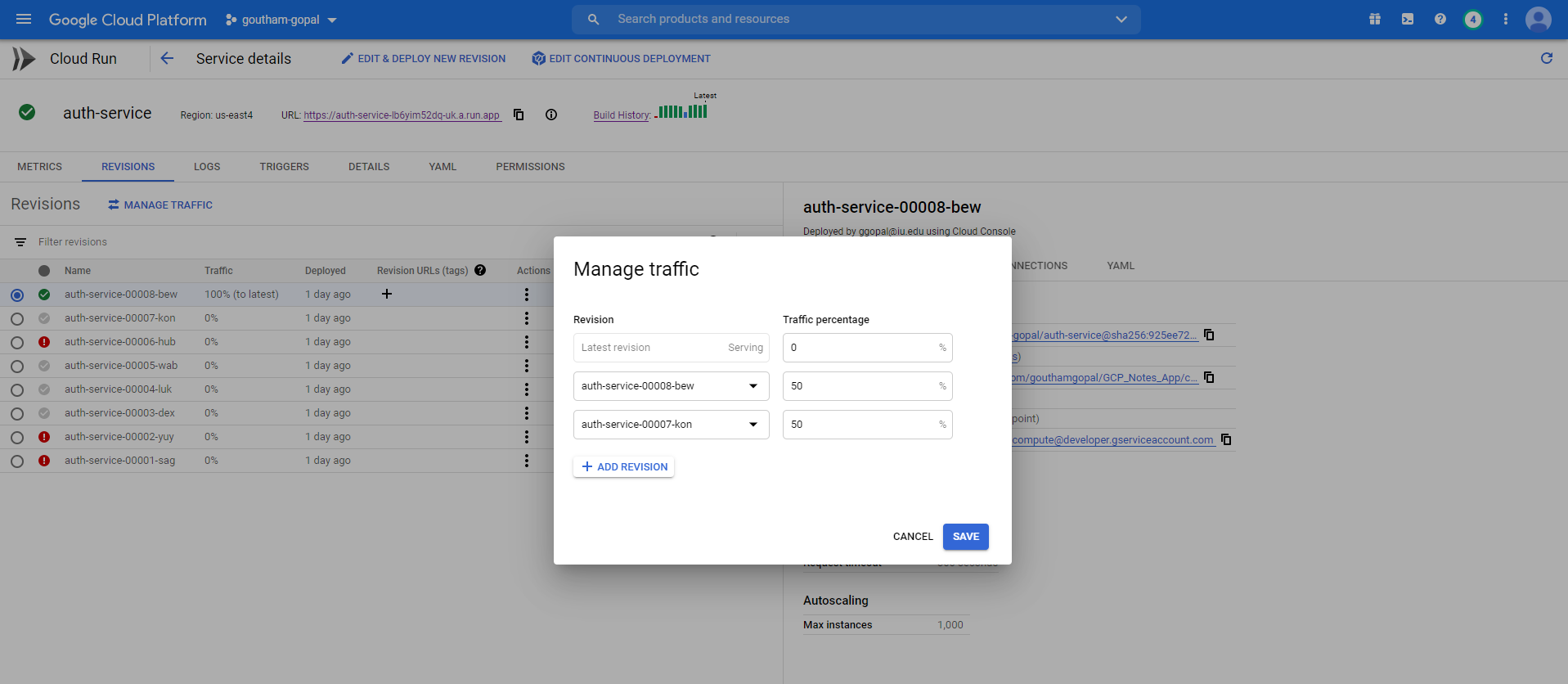
The Firestore collection is as below.



All the implementation focuses on containerized application deployment, which will help us maintain the updates as well as routing of the application in a simple manner. We can also see the build trigger logs once we update the application in GitHub as shown below.



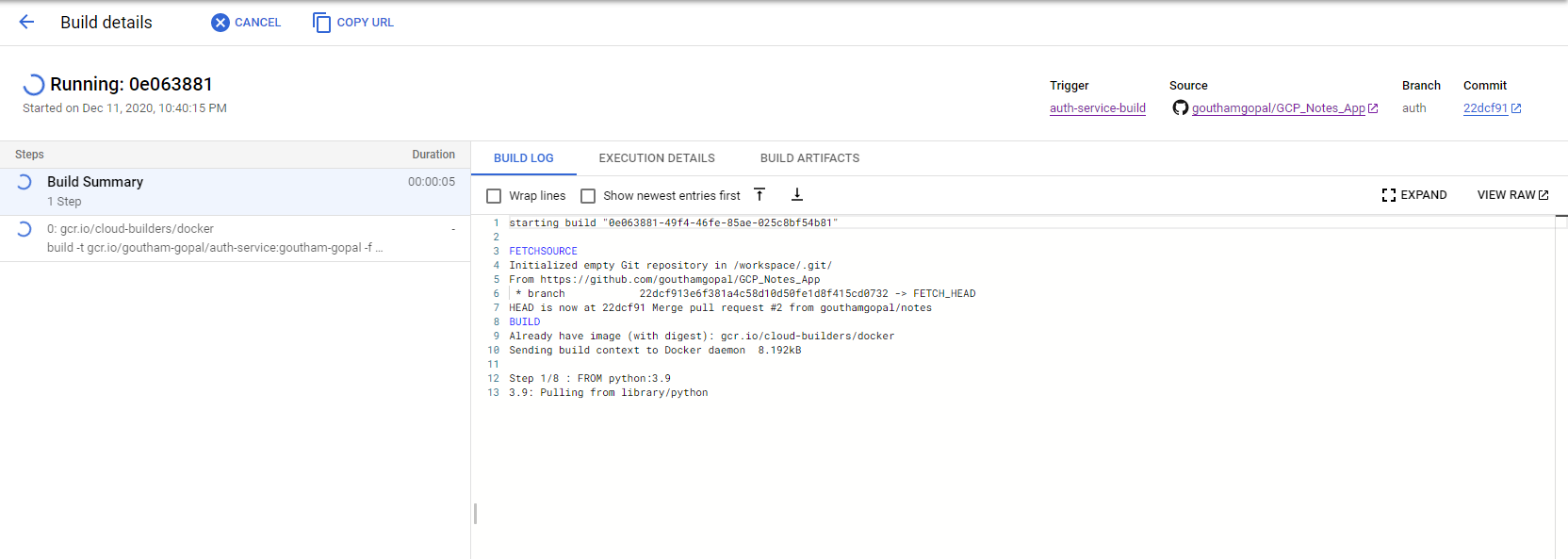
The cloud run also allows the microservices to contain multiple images active at the same time. This is a useful feature while implementing complex applications. We can take care of the routing and what percentage needs to be routed to which image from the section as shown below.

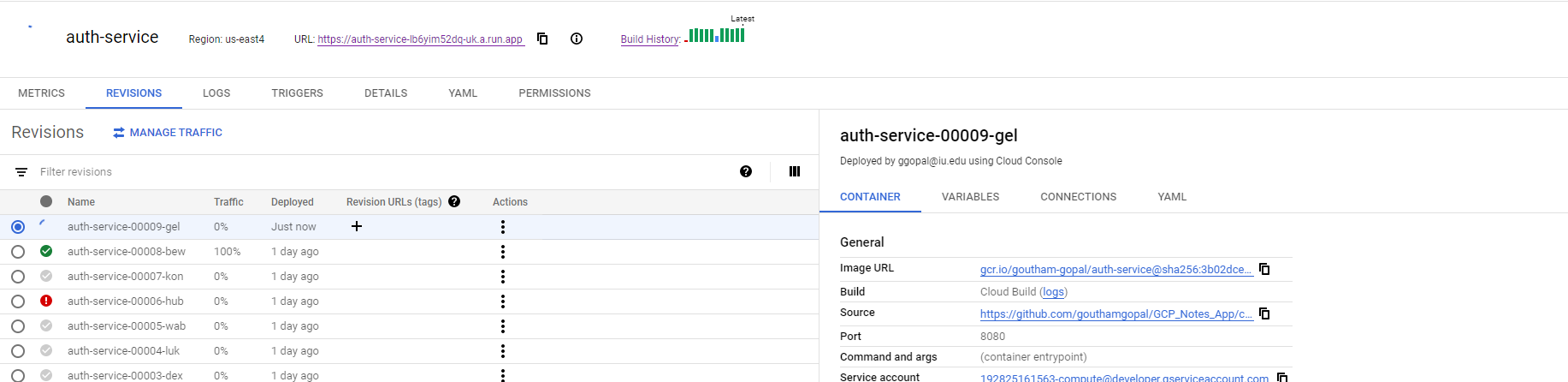


**PERFORMANCE**

The performance for the application can be measured in multiple ways. The performance for any containerized application basically is measured by the uptime the application can provide even during service failures and updation to the system. As we know that containerized images get pushed to the microservice automatically through CI/CD, the cloud run maintains the old service active for the users to send requests to until the new version is fully running without any error.

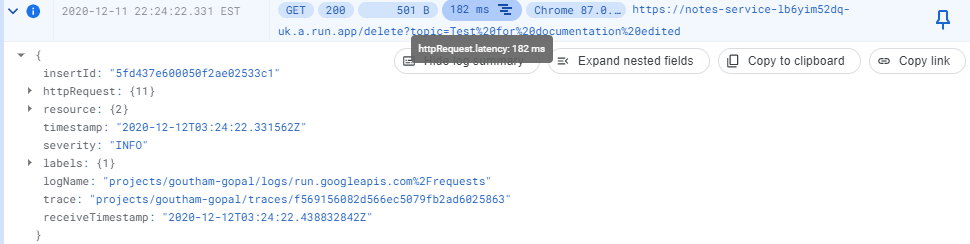
With the application, that I have developed, I could never notice any service failure during server crash or even during updating to a new revision that I have just updated. The services’ previous containerized image would still be active during updation as seen below.



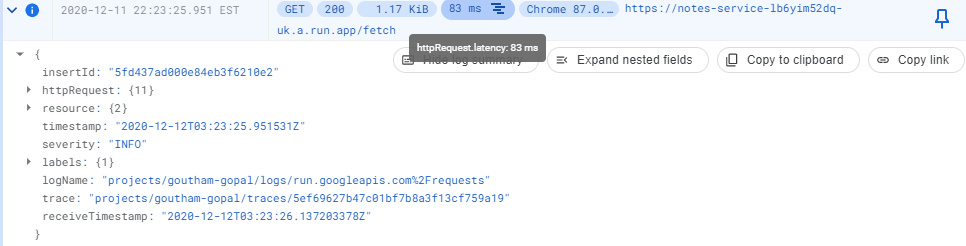


As we can see the old service is still active as the new one is being prepared to switch after an update.

We can also measure the performance of the application by checking the logs for the application to see how much time it took to satisfy each request, as shown below for a few requests we made earlier.

****

Images show the time taken by the cloud run microservice to respond to the request from the frontend. As we can see they have very low latency and can be scaled once the server requests/sec increase heavily.

****

**LEARNING FROM ASSIGNMENT**

This assignment was targeted for extensive learning about deploying cloud application to the GCP as well as creative learning through the process. For this intention, I made myself learn about the different ways in which we can deploy applications to the cloud and tried to learn a new way to upload and sustain distributed application through GCP. The major advantage of deploying microservices is the ability to distribute the application and re deploy anywhere without any issues with the base hardware specifications. The image will work with any containerization tool and will provide the same result.

I have utilized the remaining GCP credits to understand and demonstrate how we can achieve CI/CD for a basic microservice architecture-based application.

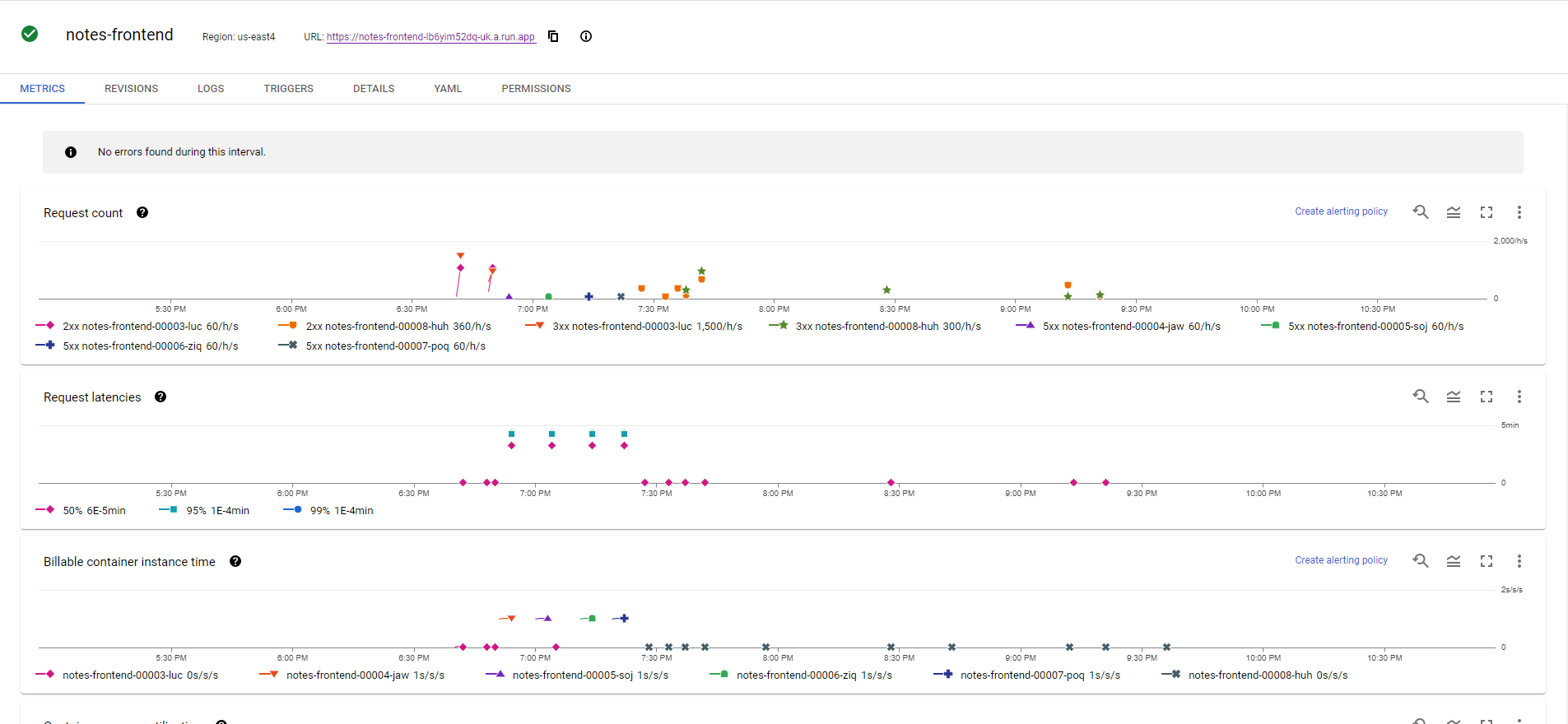
I have also understood how we can take any application, convert it into a microservice architecture by splitting the functionality and grouping similar services into a single microservice. This will allow us to containerize such microservices by utilizing various containerization methods like Docker, Cloud Build and so on. These images can then be deployed to any container application supported environment such as Cloud run or GKE for that matter.

The assignment has also taught me to have a better understanding on the concepts of image revisions as well as the ability to route the requests to multiple revisions as explained in the above section. This ability to enable us to provide with user’s different types of functionality from the same endpoint. These can be used for user-testing of a new functionality and other features as well.

It has also made me learn about how we can work with scaling and how we can control these scaling factors for each of the microservices. The cloud run in GCP, is like the Cloud functions API, where GCP will automatically handle the scale up and scale in of each microservice with respect to load. But Cloud run enables us to have a more detailed view on the requests and other parameters that relates to the scaling of the microservice as shown below.

These metrics enables us to understand how our app is performing better than how we could visualize while using Cloud functions. This better understanding also enables us to make scaling decisions on our own for our microservice, if we want to have manual scaling setup for the microservice.

I also got an opportunity to understand how we can utilize NoSQL databases like Cloud Firestore and be able to store data as collections and documents rather than a primitive RDBMS perspective. This enables us to easily store the json data that the request receives from the frontend as well as enables easy interaction with the data without much conversion compared to the tabular data formatting from RDBMS.



**CONCLUSION**

I have learnt a lot of useful techniques and gained a lot of knowledge developing a software application and using GCP to deploy it. The assignment has made me realize how and when to opt for serverless application as well as something like Cloud run, which is also serverless but with more control for users.

I have utilized my learnings to produce a simple application which talks via HTTP requests to microservices and produces output. Through the process I have also come across asynchronous requests that can be setup by using Google Pub Sub. This type of asynchronous system can be used for a data pipeline application, which is how we can reform data-oriented applications in the future.