# Assignment 2

## Part A

Meet Patel (B00899516)

Dalhousie University

## Subject

CSCI 5410 (Serverless Data Processing)

## Professor

Dr. Saurabh Dey

## Project Git Repository

**Gitlab Repository Link:** https://git.cs.dal.ca/patel13/csci5410_b00899516_meet_patel.git

**Website Link:** https://container-1-anmmfspxoa-ue.a.run.app/

## Firebase Firestore Setup

Figures from 1 to 6 are responsible for displaying the creation of **Firebase Firestore database without any data.**


*Figure 1: Firebase Firestore project creation with project name **CSCI5410***

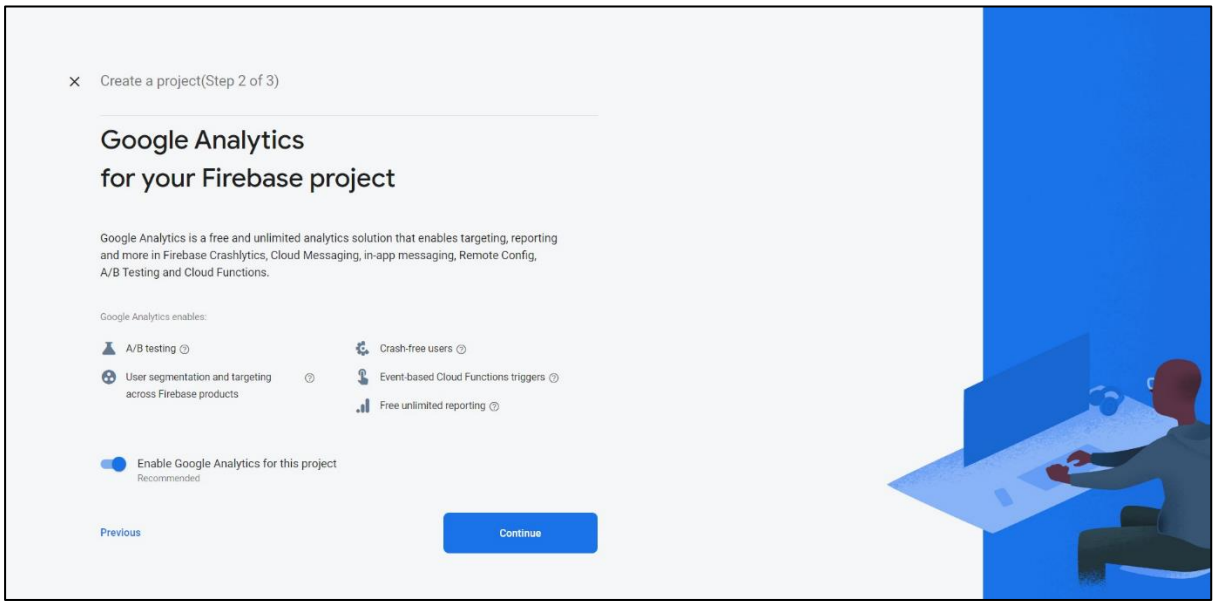



*Figure 2: Firebase Firestore project creation with project name **CSCI5410***

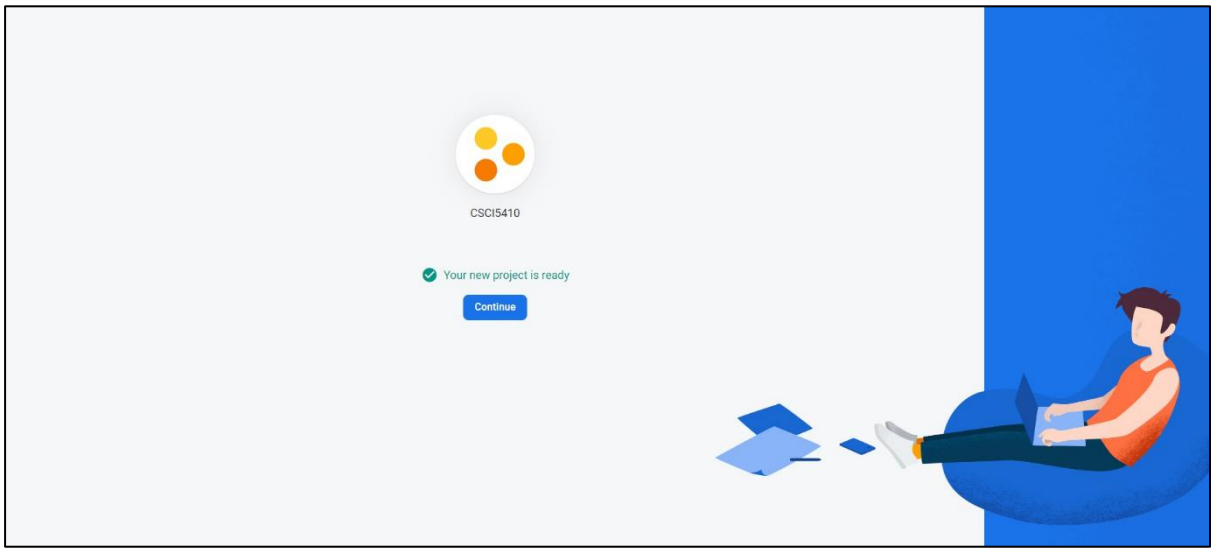



*Figure 3: Firebase Firestore project creation with project name **CSCI5410***

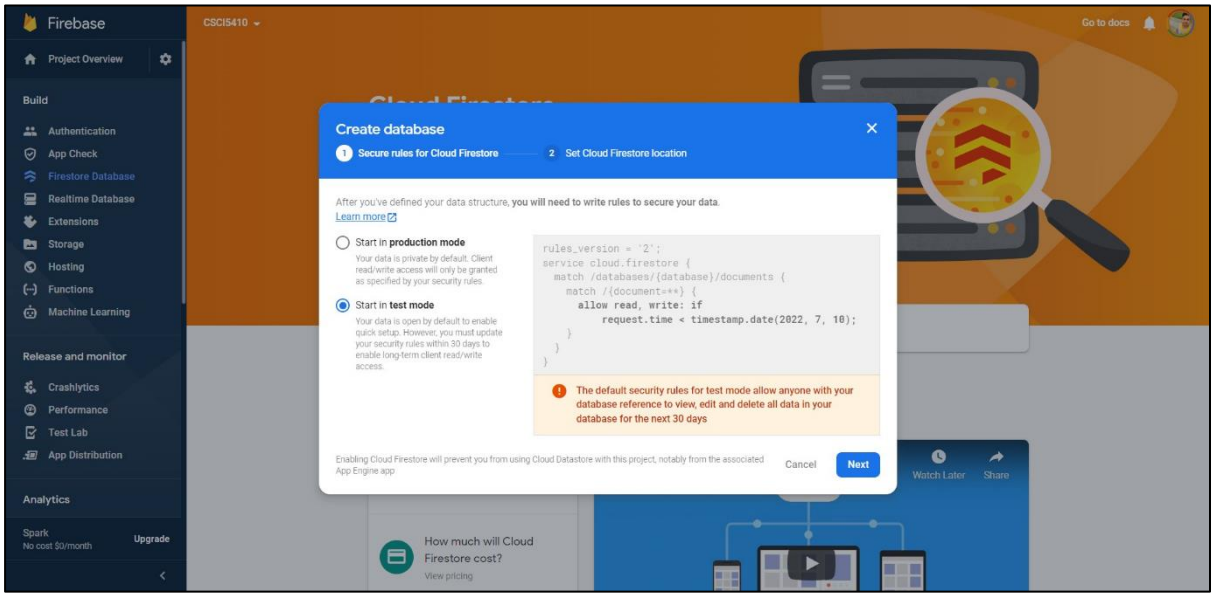



*Figure 4: Firebase Firestore database creation*

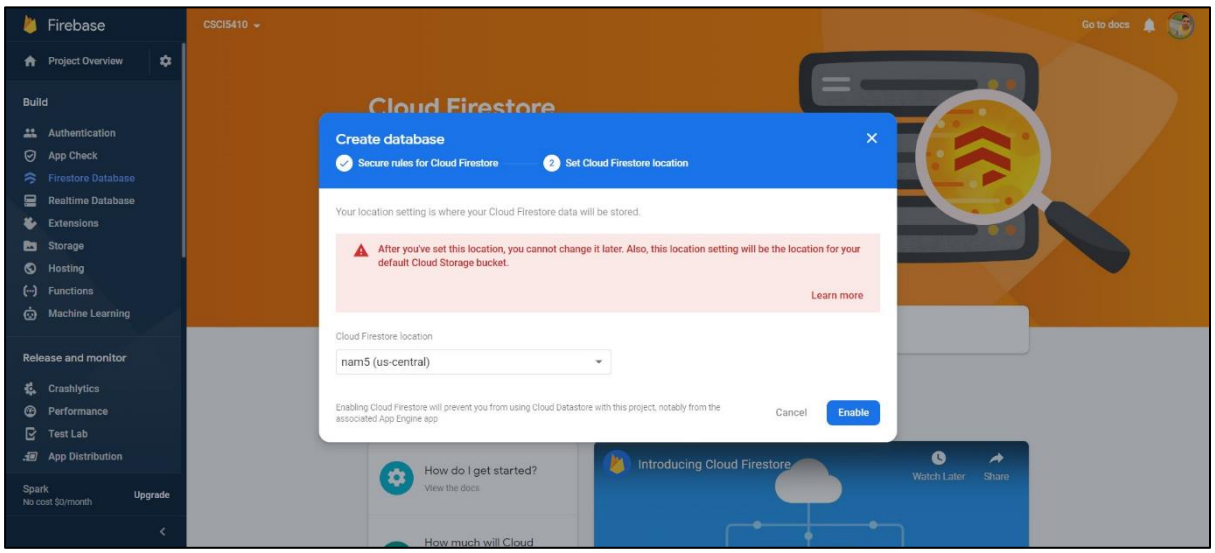



*Figure 5: Firebase Firestore database creation*

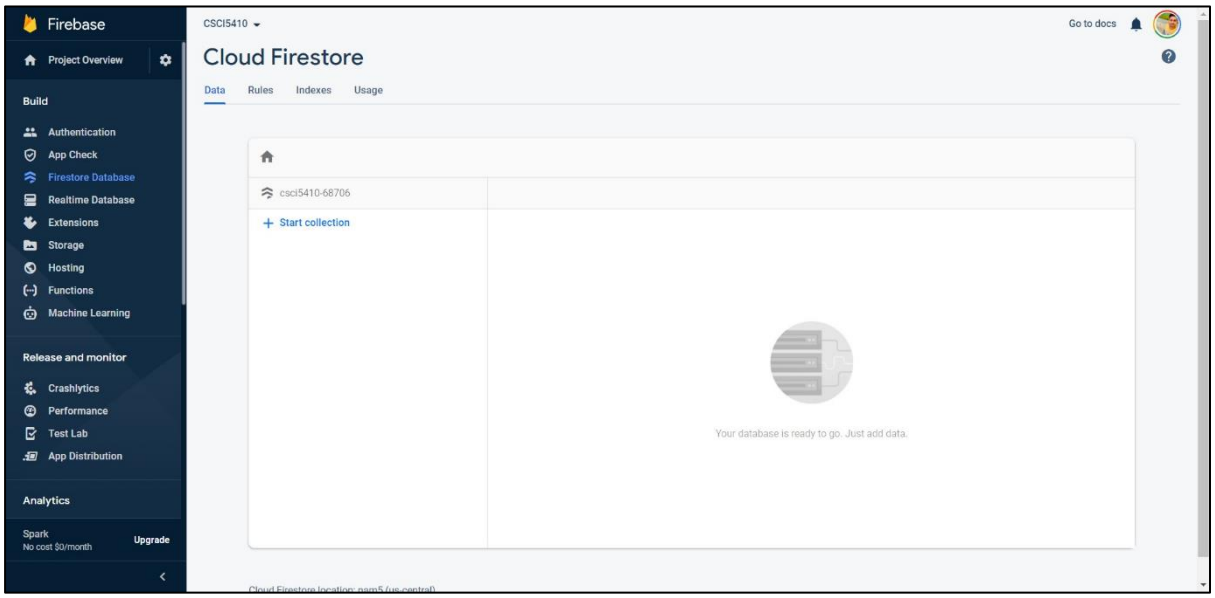



*Figure 6: Firestore with empty database (Without users)*

## Creation of docker file for react application

Figure 7 shows the docker file for Registration react application (**container_1**)

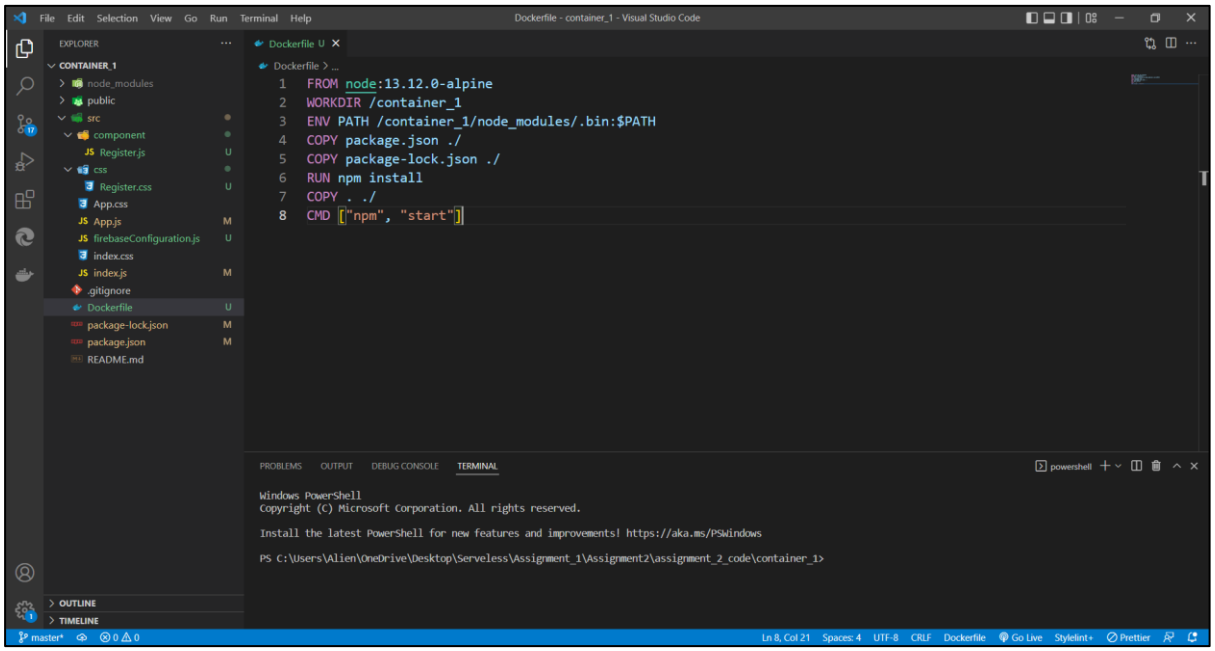



*Figure 7: Docker file for Registration react application (container_1)*

Figure 8 shows the docker file for login react application (**container_2**)

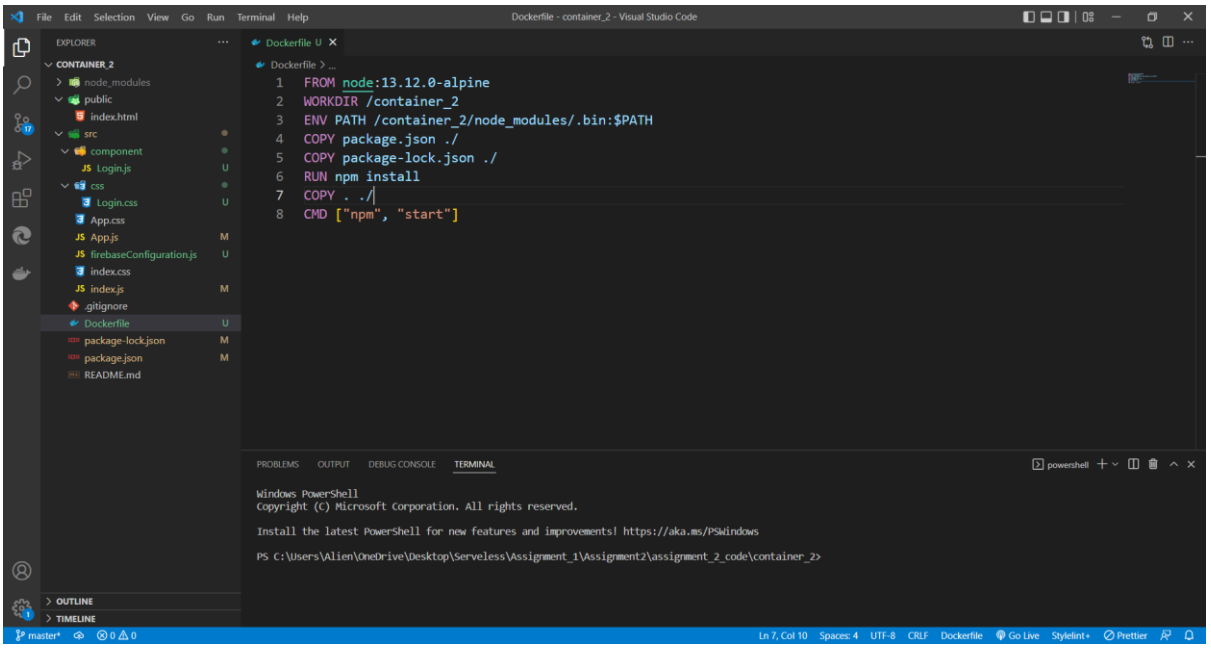



*Figure 8: Docker file for Registration react application (container_1)*

Figure 9 shows the docker file for UserProfile react application (**container_3**)

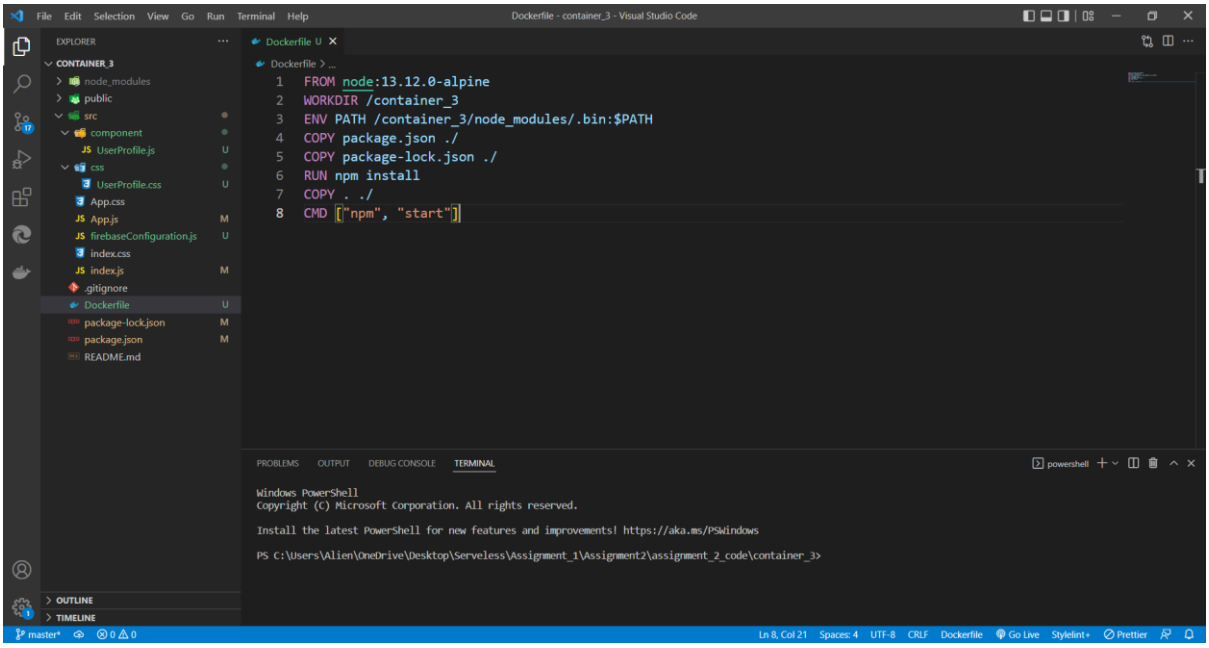



*Figure 9: Docker file for UserProfile react application (container_3)*

## Creation of Containers Images using docker

### Container 1 – Registration React Application

With the help of docker **container_1** registration react application has been created. This application accepts name, email, location, and password from user and store it in the Firebase database. After registration gets successful the user is redirected to Login react application which (Container_2). At this point as user is not logged in the status of the user is set to **offline.**

Figure 7 shows the successful creation of the docker image with name **container_1**

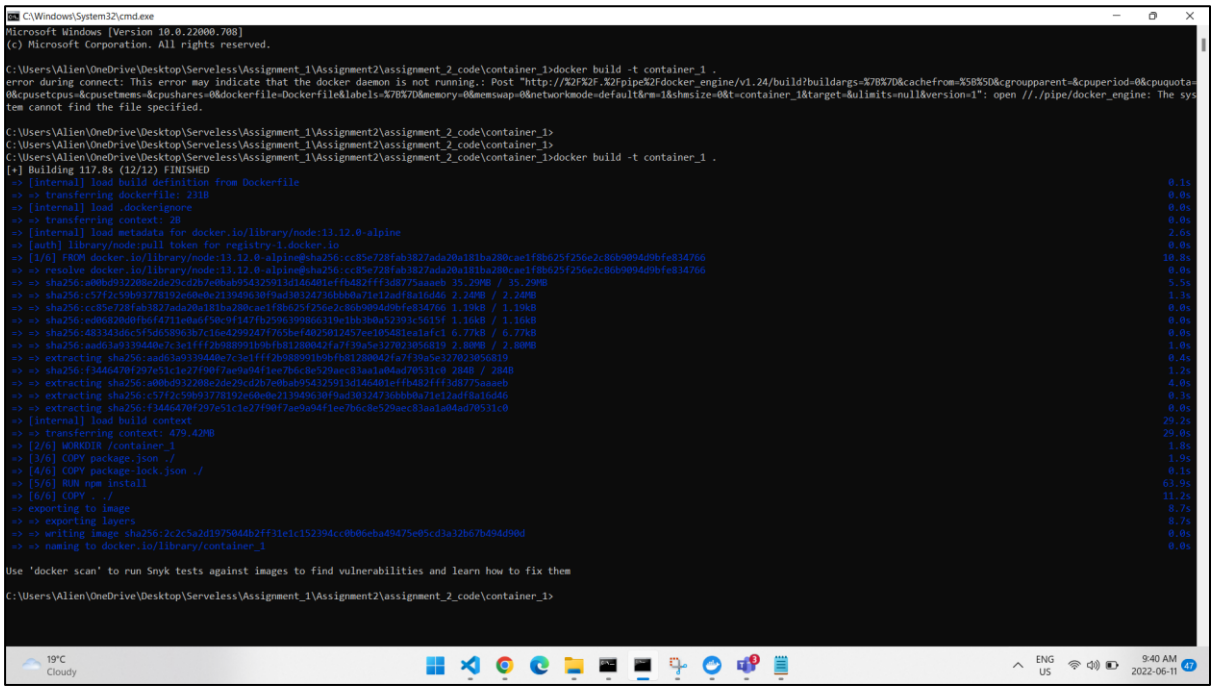



*Figure 10: Successful creation of the docker image with name **Container_1 (Registration React App)***

### Container 2 – Login React Application

With the help of docker **container_2** registration react application has been created. This application accepts email and password to let user logged in. This information is then verified using the data stored in the firebase database. After the information got verified, the user is redirected to the UserProfile react application (container_3). At this point the user is logged in and is online hence user status will be updated to **online**

Figure 8 shows the successful creation of the docker image with name **container_2**

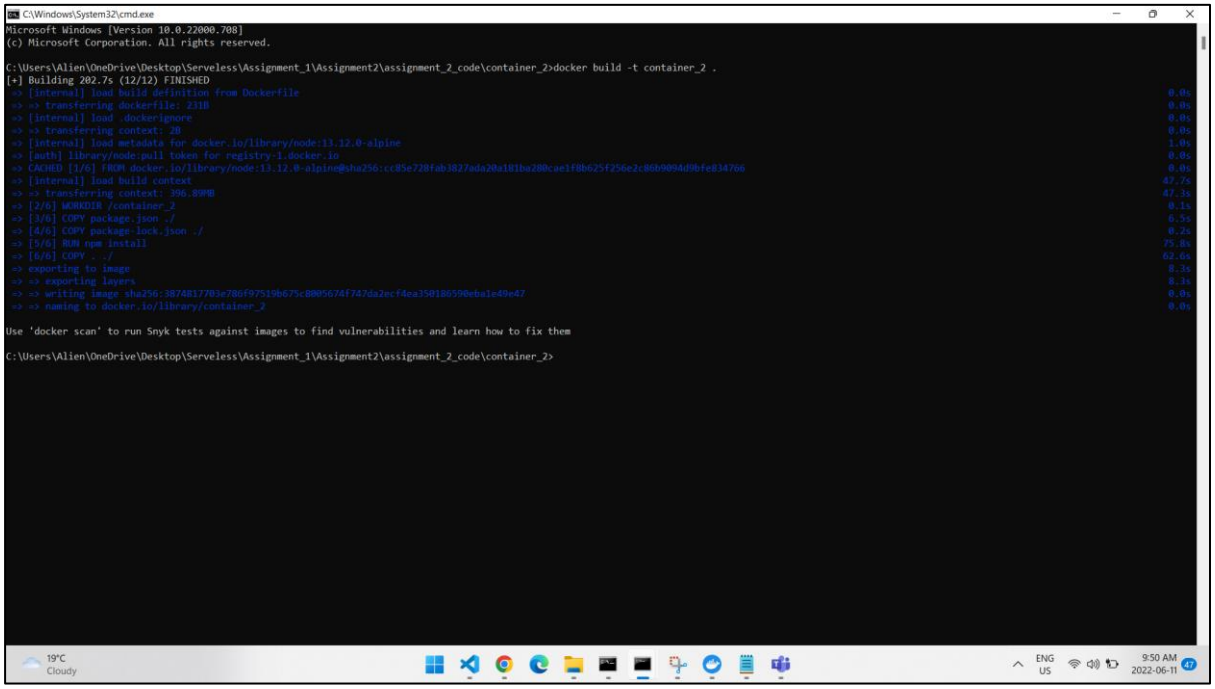



*Figure 11:Successful creation of the docker image with name **Container_2 (Login React App)***

## Container 3 – UserProfile React Application

With the help of docker **container_3** registration react application has been created. This page shows the email of the logged in user along all other users who are online. In addition, there is a logout button that allows users to logout. When user clicks on logout the user will gets logged out from his profile and the status of the user will be set to offline. Along with that the user is also redirected to back to the login react application (container_2)

Figure 9 shows the successful creation of the docker image with name **container_3**

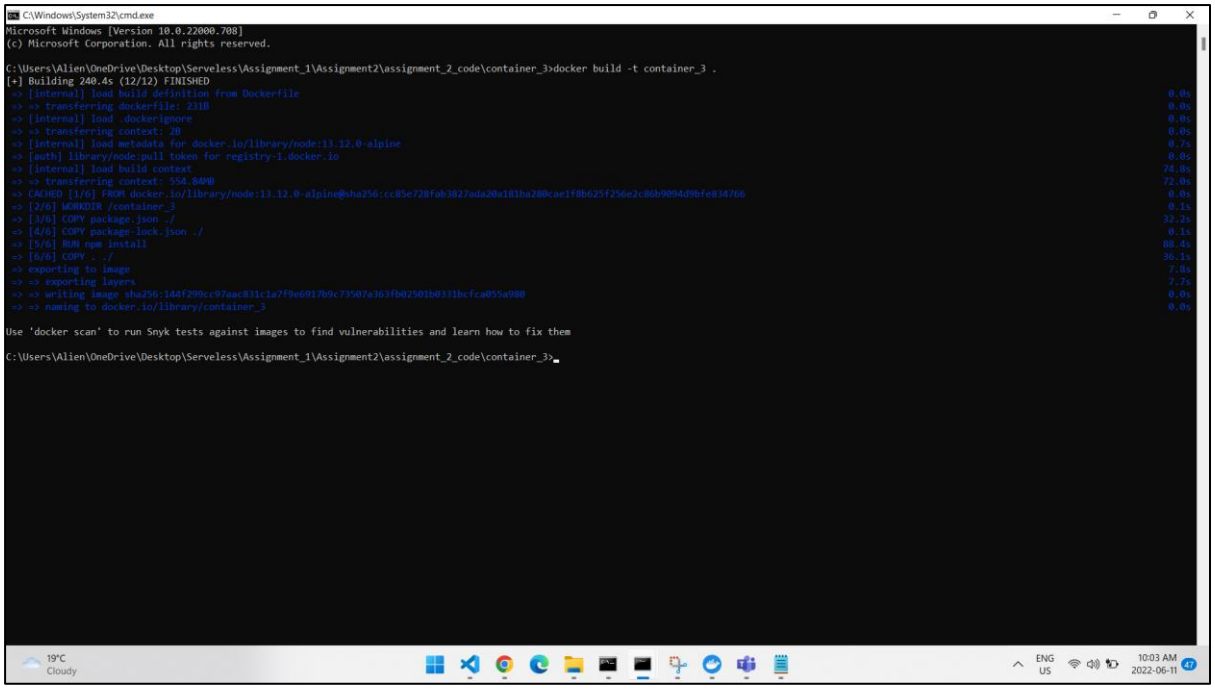
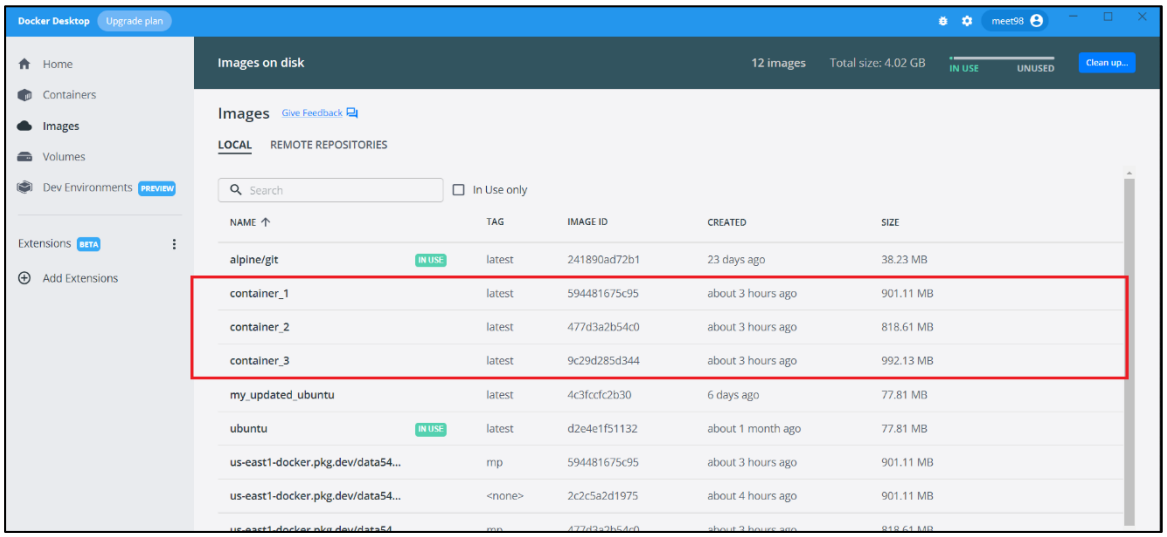



*Figure 12: Successful creation of the docker image with name Container_3 (UserProfile React App)*

## Successfully Creation of 3 Docker Images

Figure 10 shows successful creation of three docker images **(container_1, container_2 and container_3)**



# Website Working (Docker Image Localhost) with Firebase Images

## 1. Registration React Application

Figure 13 shows the **Registration Page (container_1)** using docker container on Localhost

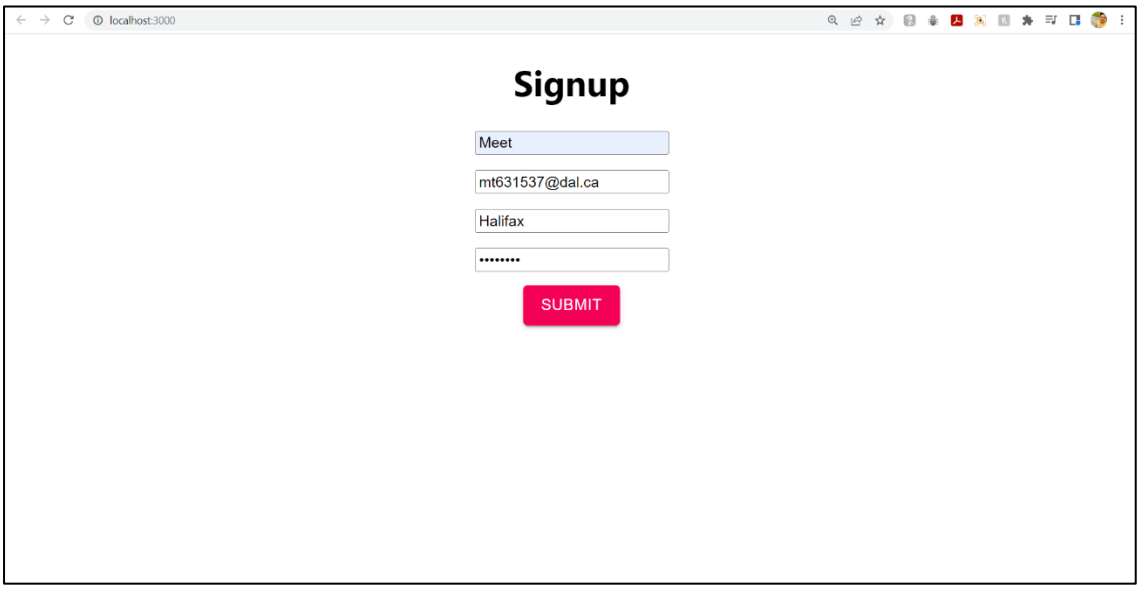



*Figure 13: Registration page showed using container_1*

Figure 14 and 15 shows the successful registration of the user with status set to **offline** and the data is stored in firebase database

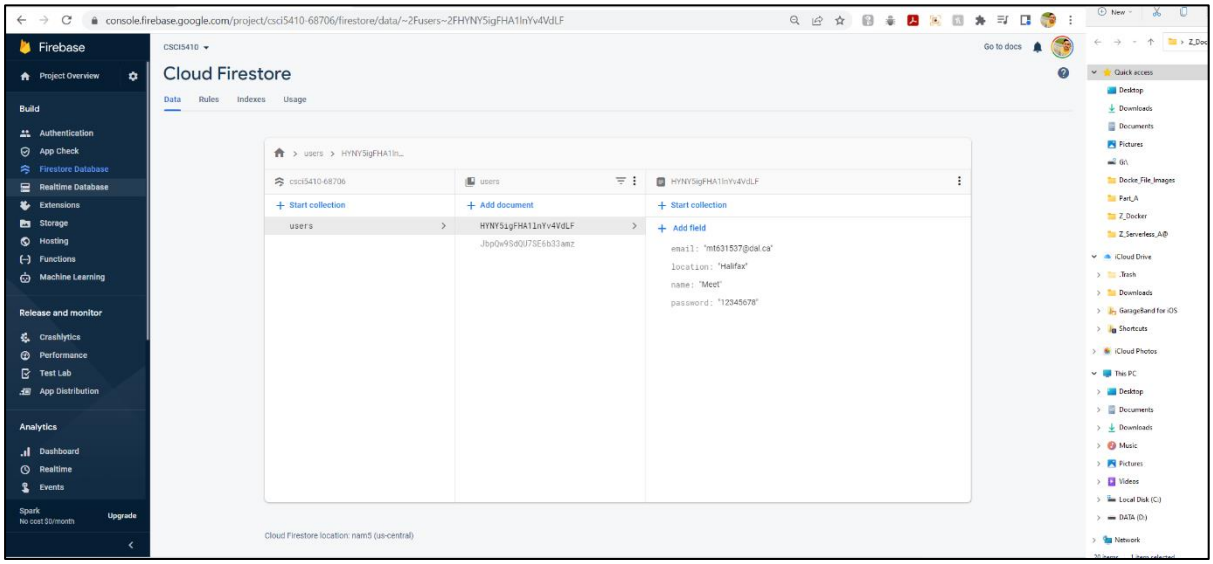



*Figure 14: Successful registration of user and data stored to firebase database*

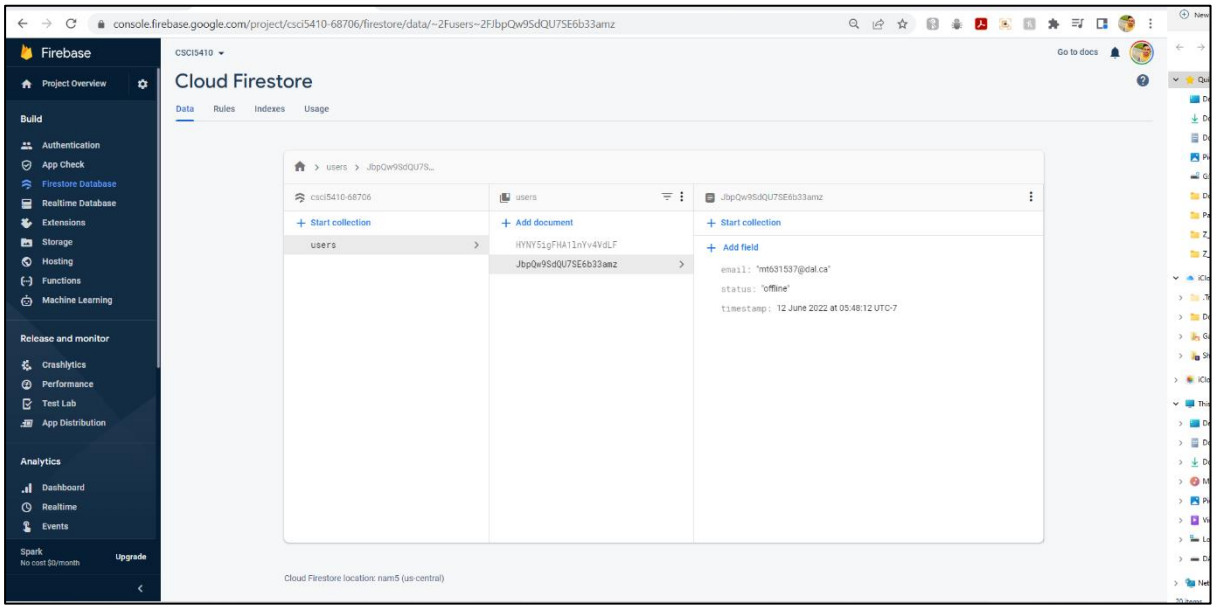



*Figure 15: After registration user's status set to **offline***

## 2. Login React Application

Figure 16 shows the **Login Page (container_2)** using docker container on Localhost

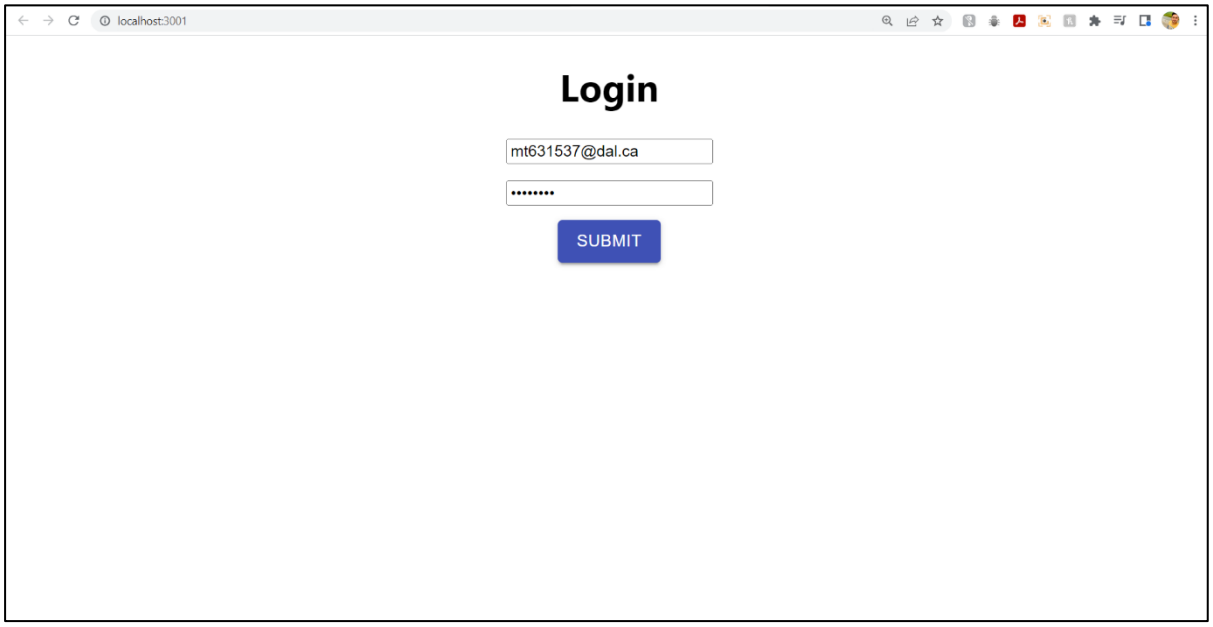



*Figure 16: Login page showed using container_2*

Figure 17 shows the status of the user updated to **online** after login

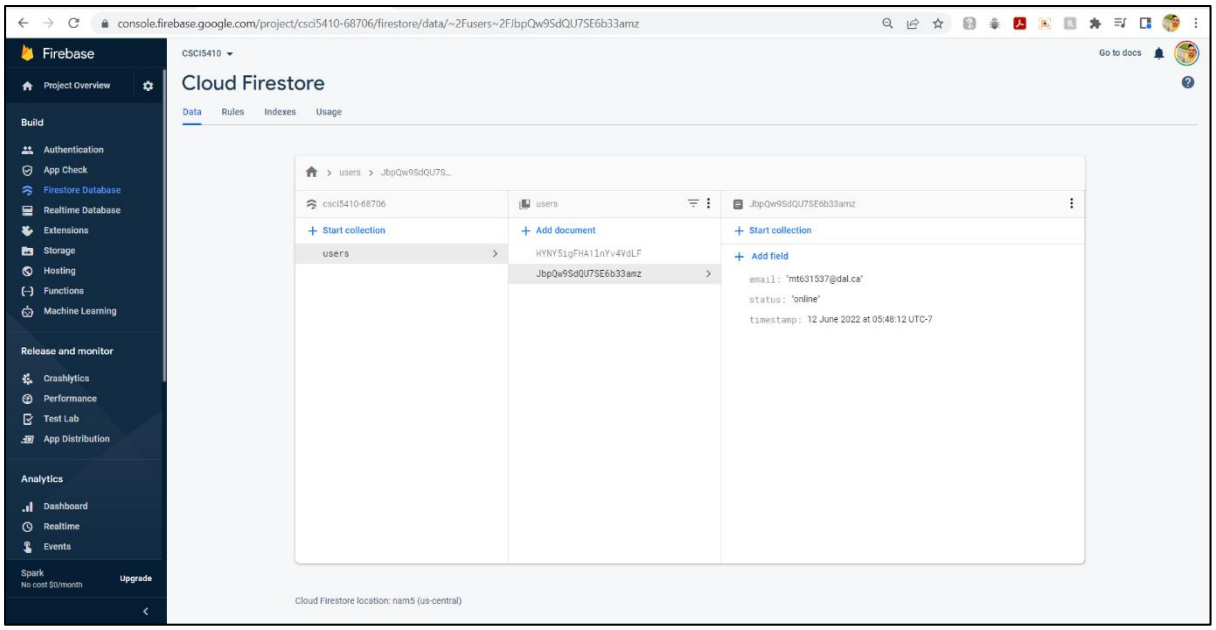



*Figure 17: Status set to **online** after login*

## 3. UserProfile React Application

Figure 18 shows the **UserProfile Page (container_3)** using docker container on Localhost. This page shows the logged in user email address and a logout button.

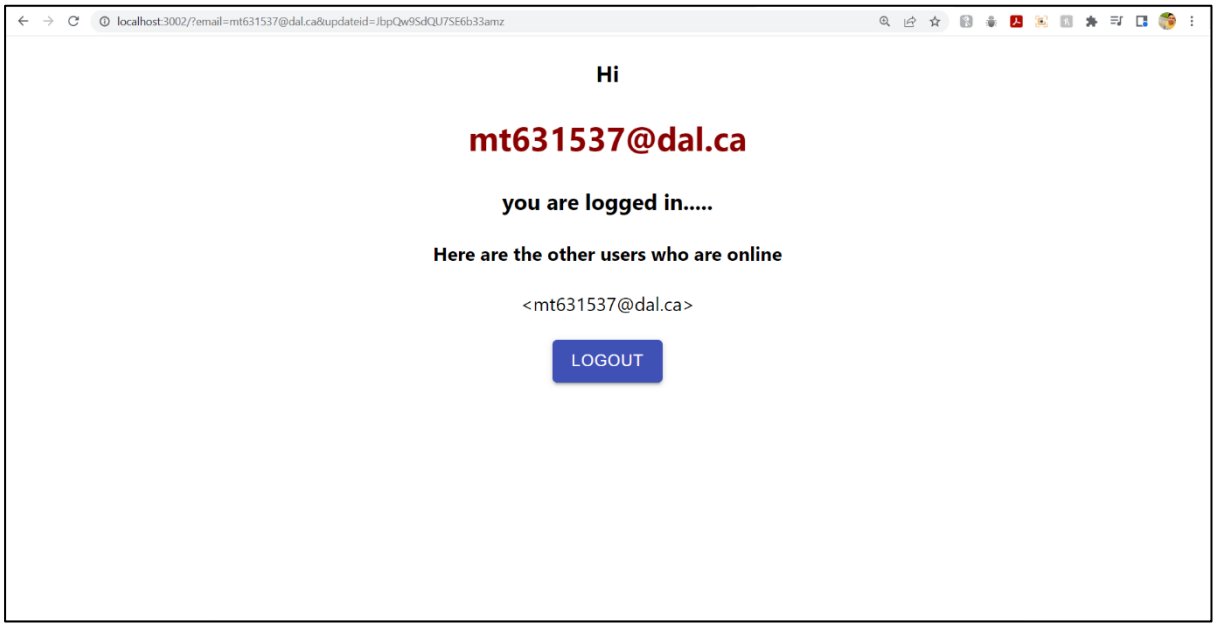



*Figure 18: UserProfile page showed using container_3*

Figure 19 shows the status of the user updated to **offline** after user logged out. The user is then redirected to login react application.

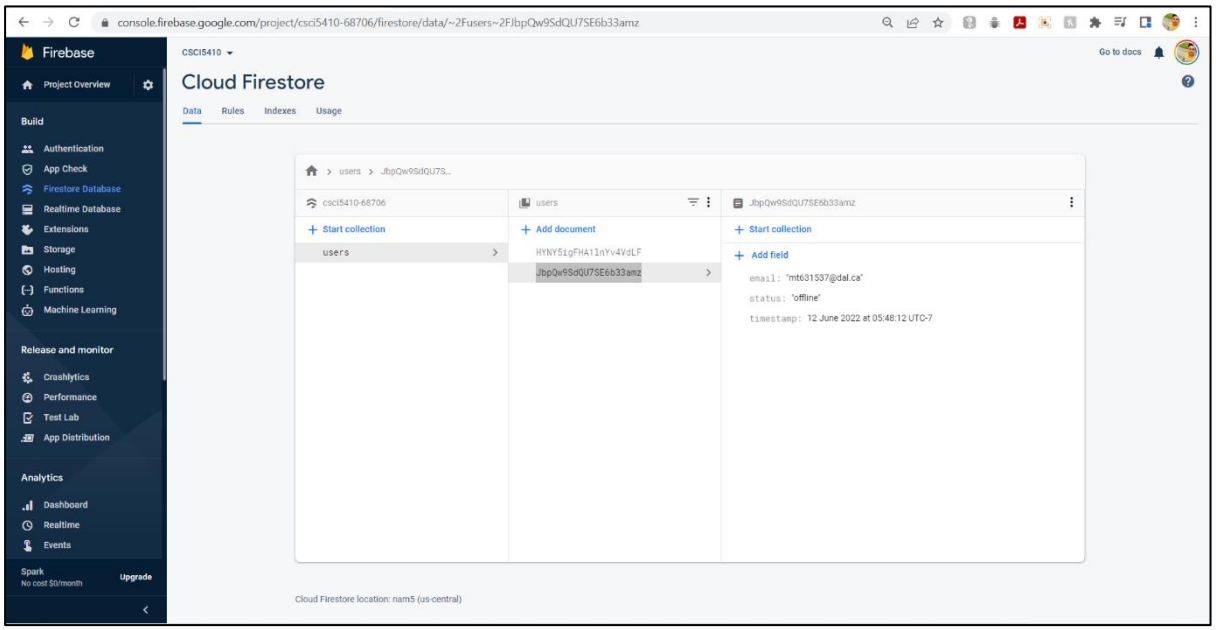



*Figure 19: User status set to **offline** after logout*

## Google Could CLI Setup (Google Cloud SDK)

**Figures from 20, 21, 22, 23, 24, and 25** shows the setup process of google cloud CLI setup


*Figure 20: Google CLI Setup Step-1*

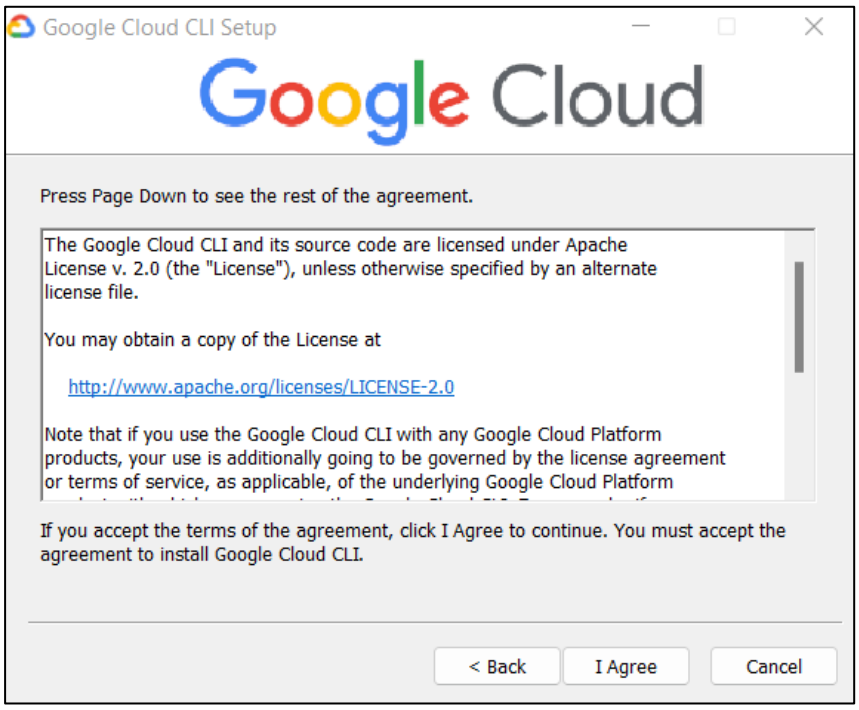



*Figure 21: Google CLI Setup Step-2*

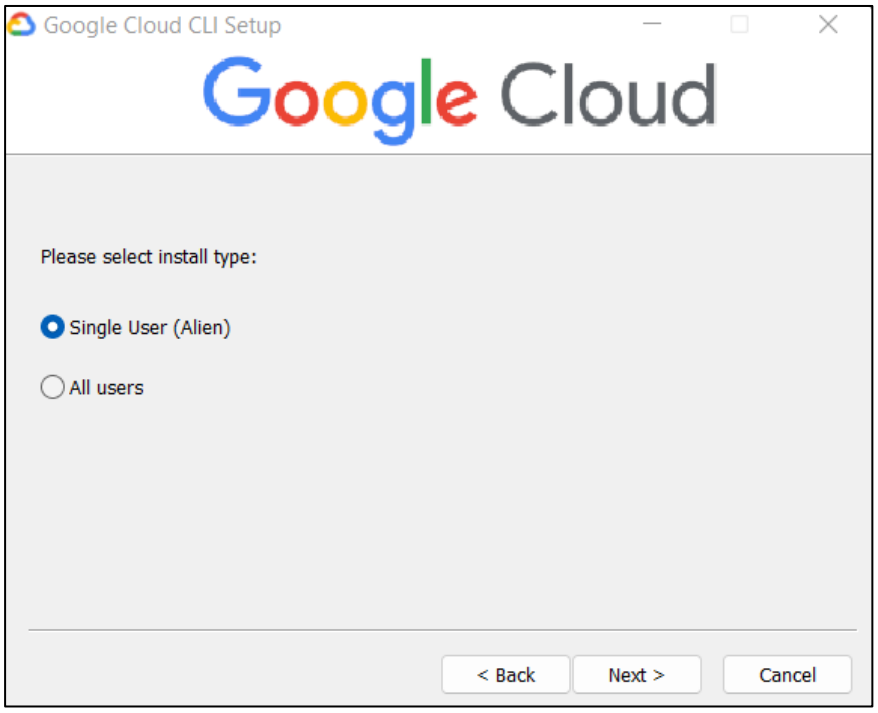



*Figure 22: Google CLI Setup Step-3*

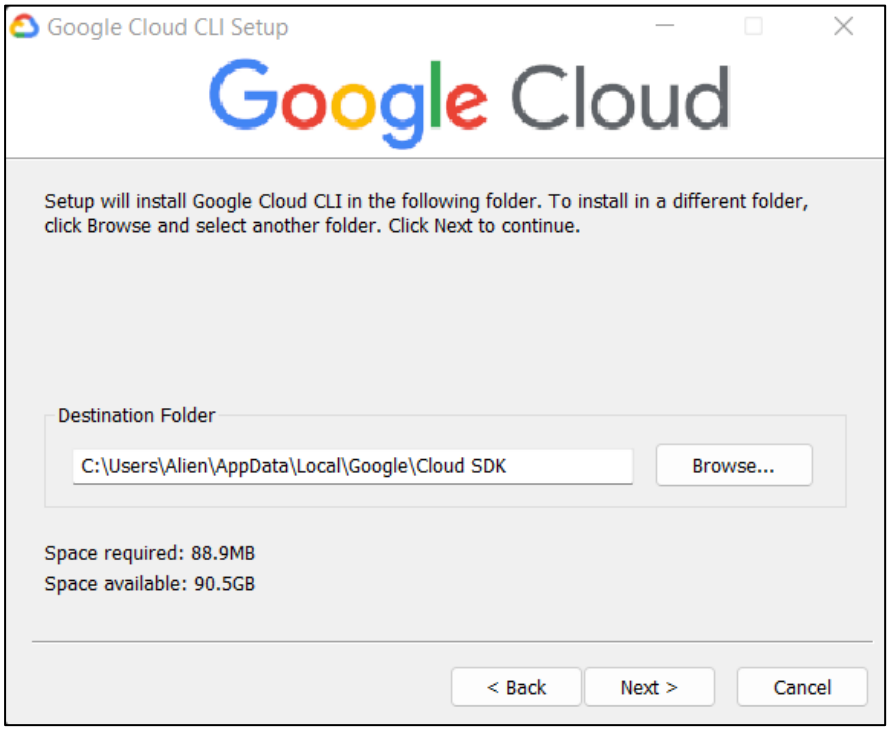



*Figure 23: Google CLI Setup Step-4*

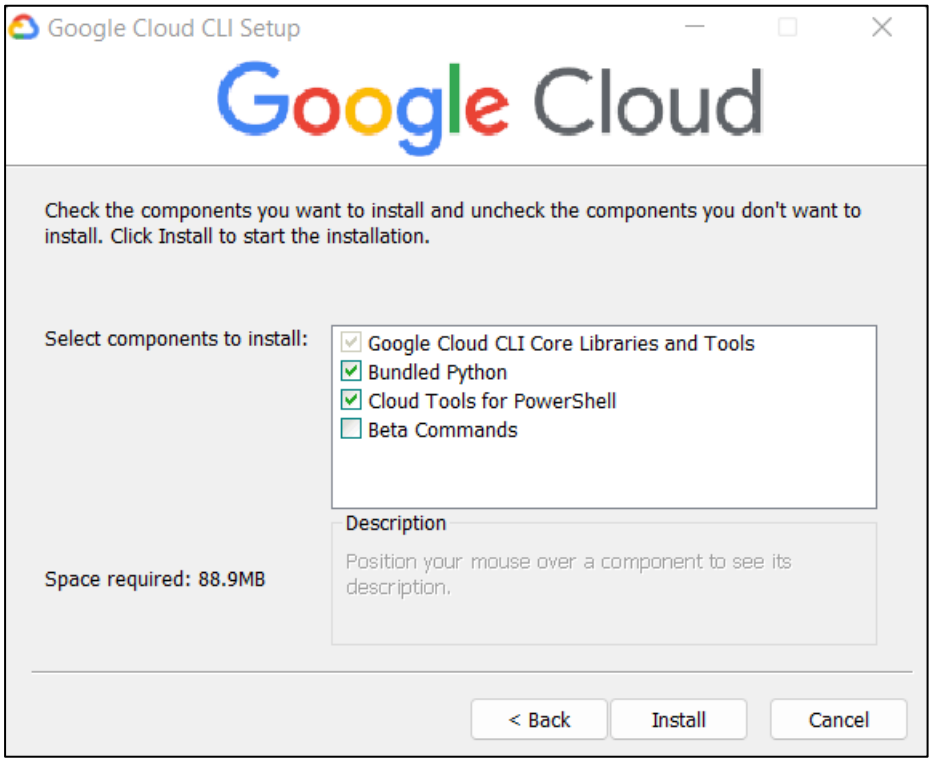



*Figure 24: Google CLI Setup Step-5*

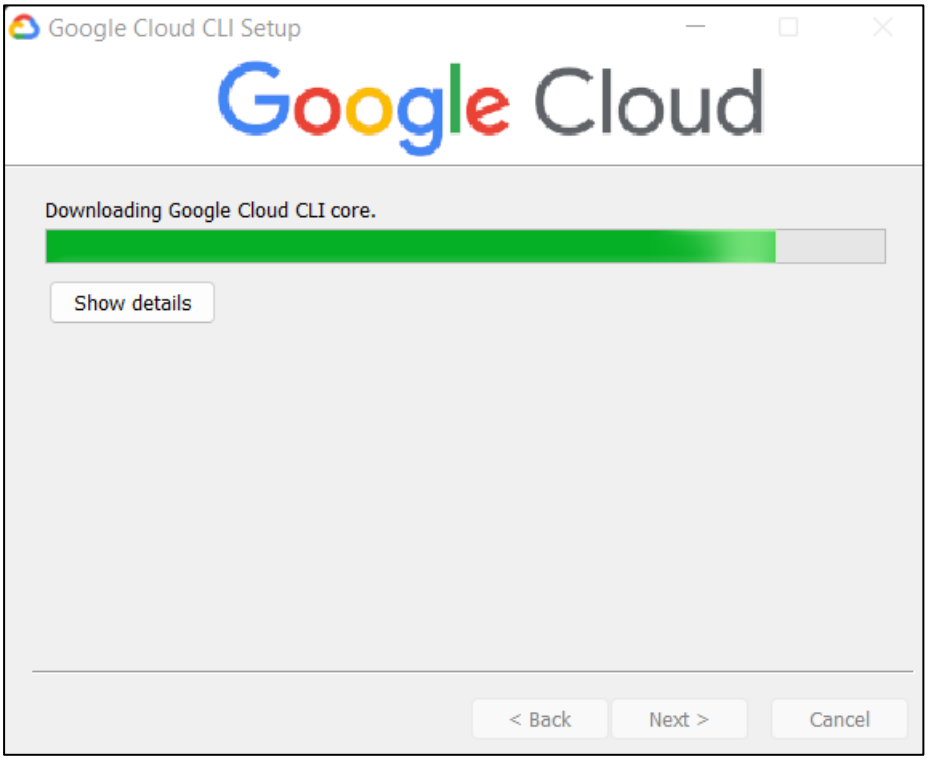



*Figure 25: Google CLI Setup Step-6*

## Google Container Registry

Steps:

1. Download and install Google Cloud SDK from Google's official website
2. In CLI login using google account
3. Then select the project in which you want to tag the docker image

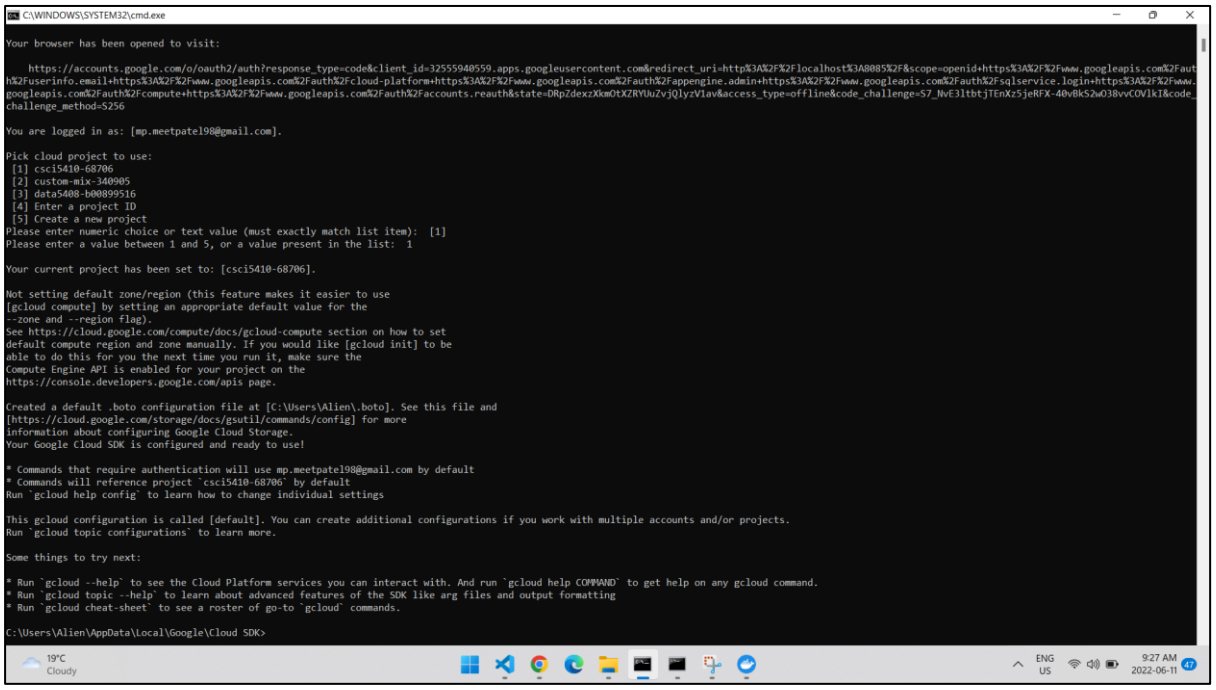



*Figure 26: Selection of the project on google cloud for tagging the docker images.*

# Google Cloud Auth Configuration, Tagging of Docker Images and Pushed docker images to Google Container Registry service

Figure 27, 28, 29 shows the tagging docker images (container_1, container_2 and container_3) and then pushed it to Google Container Registry service

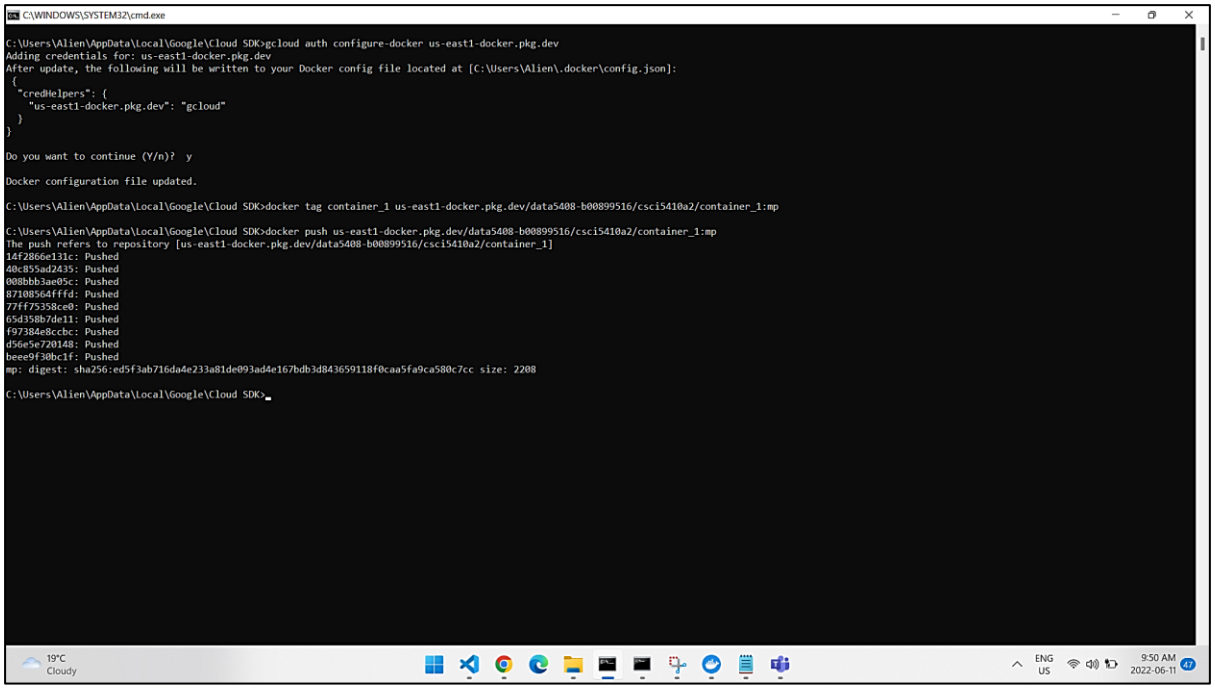



*Figure 27: Tagging of **container_1** docker and pushed it to Google Container Registry service*

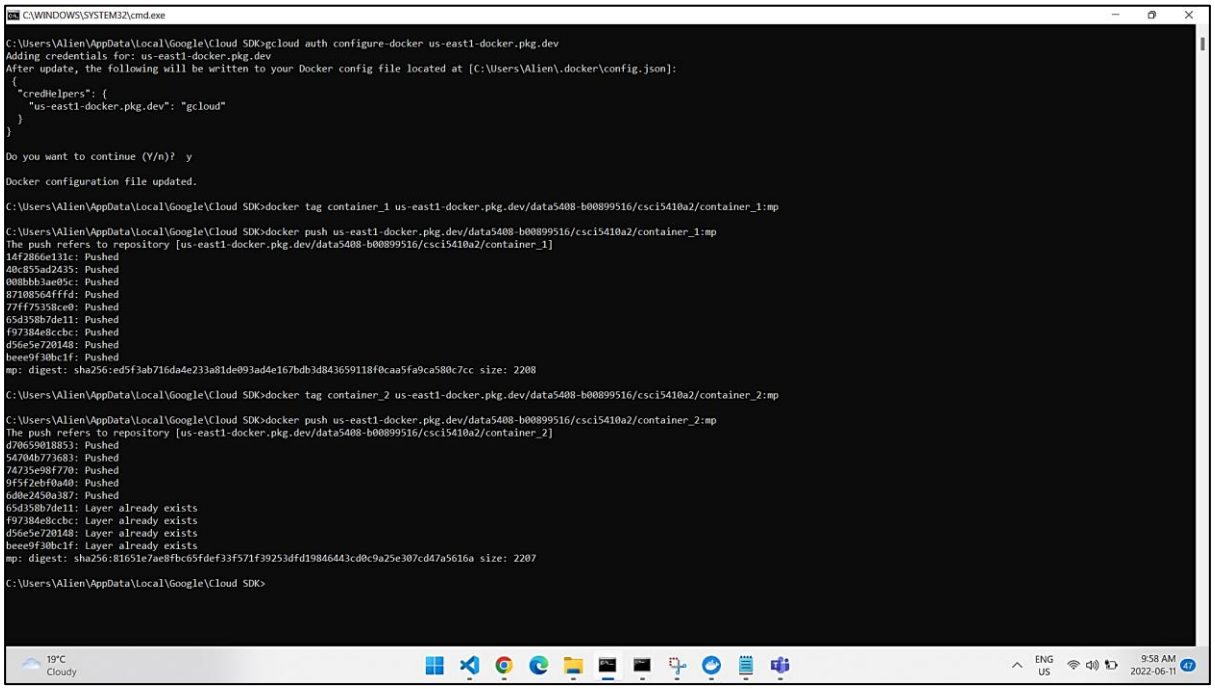



*Figure 28: Tagging of **container_2** docker and pushed it to Google Container Registry service*

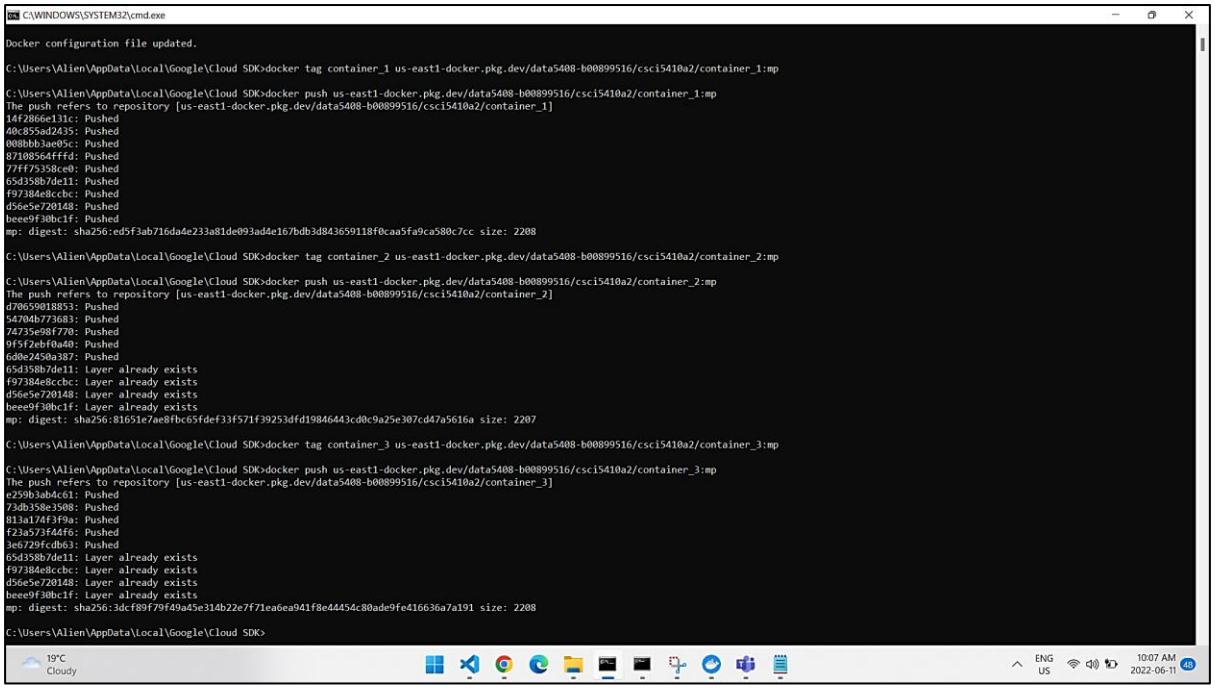



*Figure 29: Tagging of **container_3** docker and pushed it to Google Container Registry service*

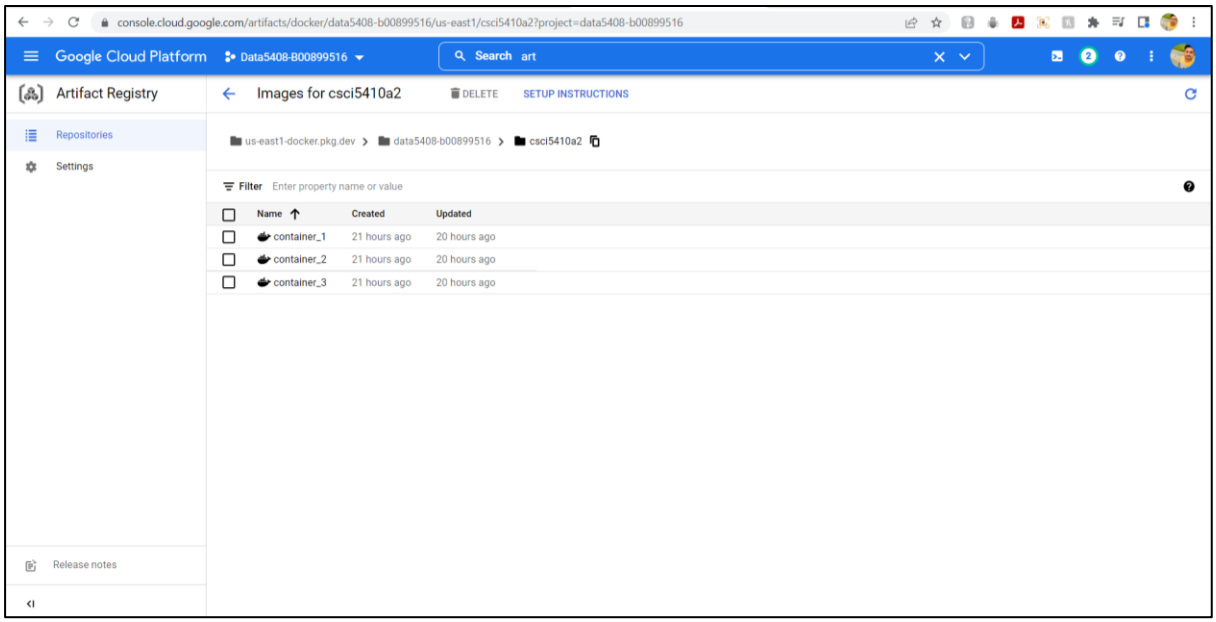



*Figure 30: All Three Docker images uploaded to Google Container Registry (**csci5410a2**)*

# Google Cloud Run and Website hosted on GCR

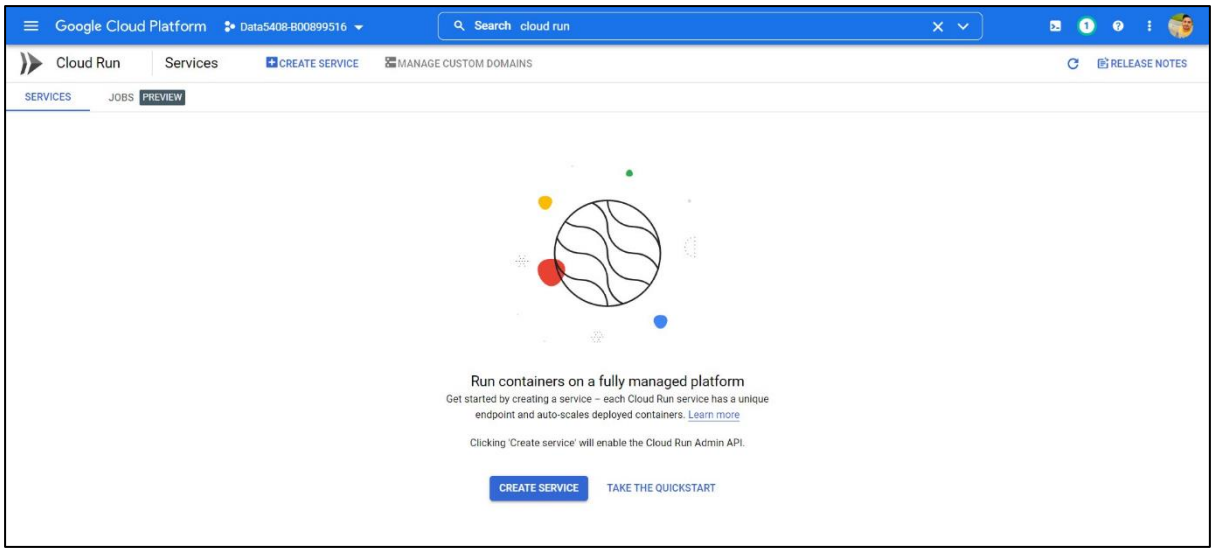



*Figure 31: Google cloud run empty dashboard*

Figures 32, 33, 34, 35, 36, and 37 shows the docker configuration named as container_1, container_2 and container_3 on Google cloud run.

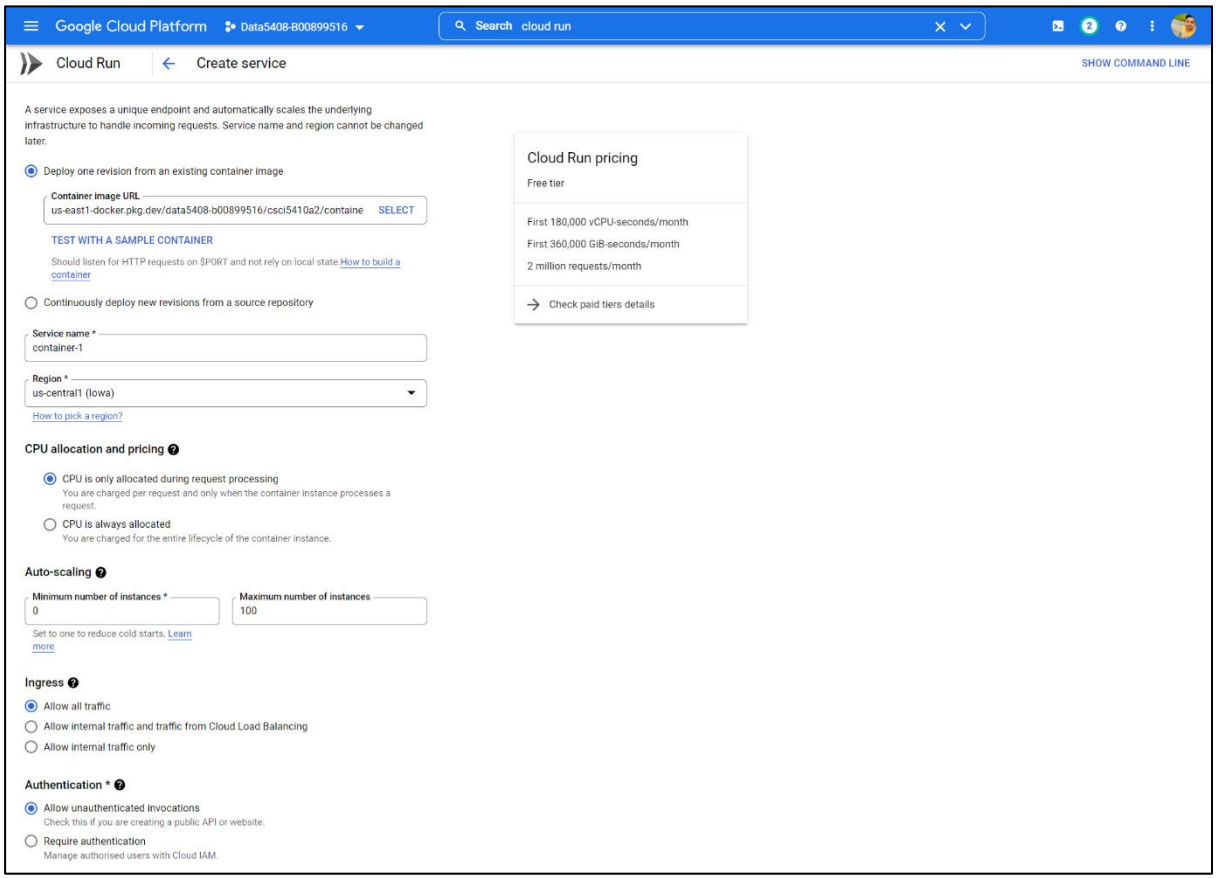



*Figure 32: **Container_1** Docker image Configuration on Google Cloud Run*

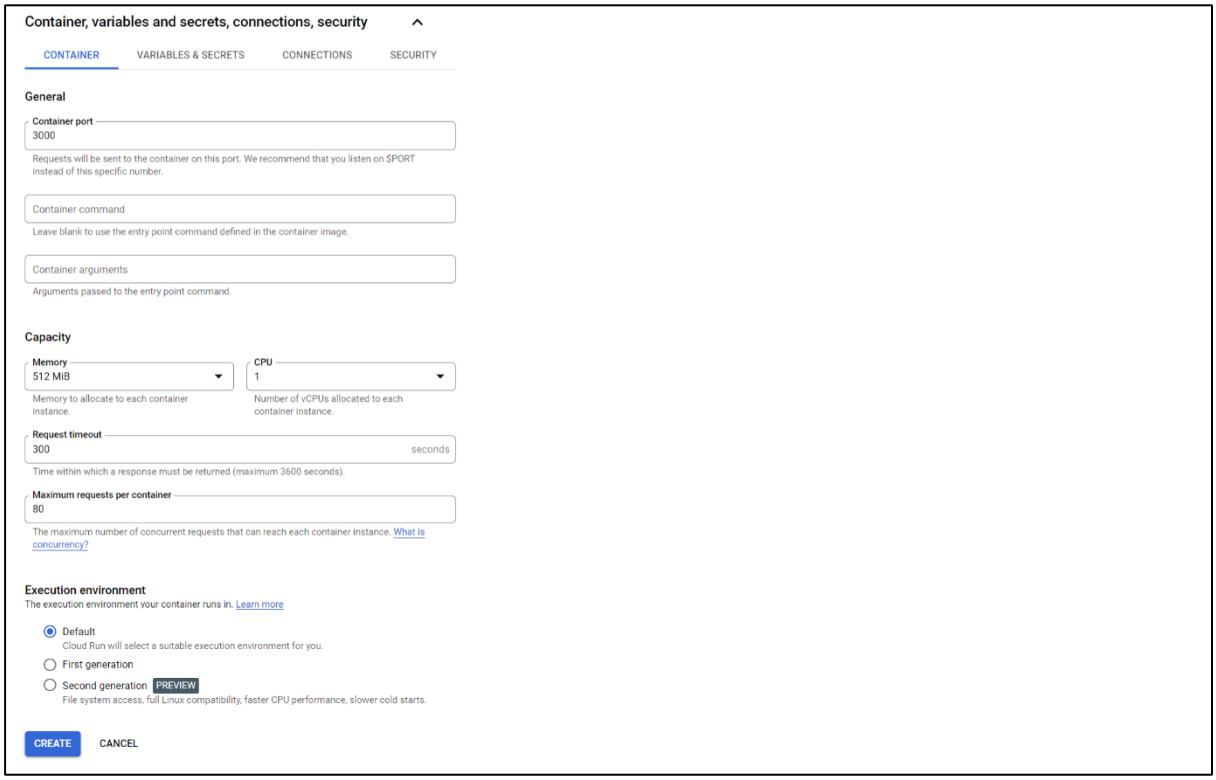



*Figure 33: Container_1 Docker image Configuration on Google Cloud Run*

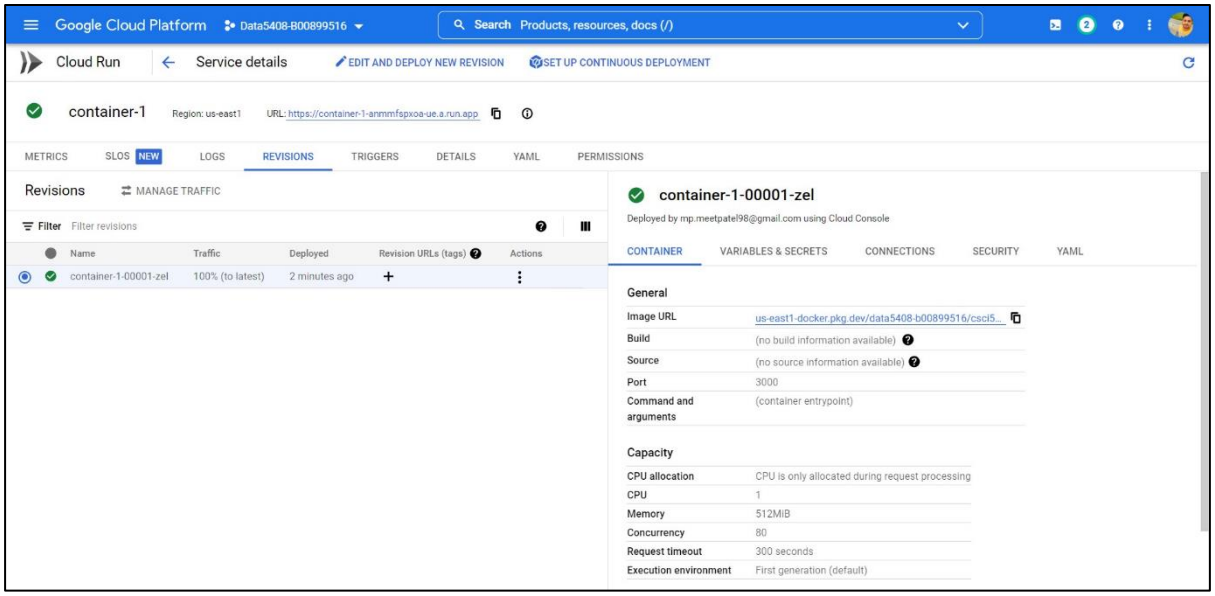



*Figure 34: Hosted continer_1 (Registration React Application) on GCR (Google Cloud Run)*

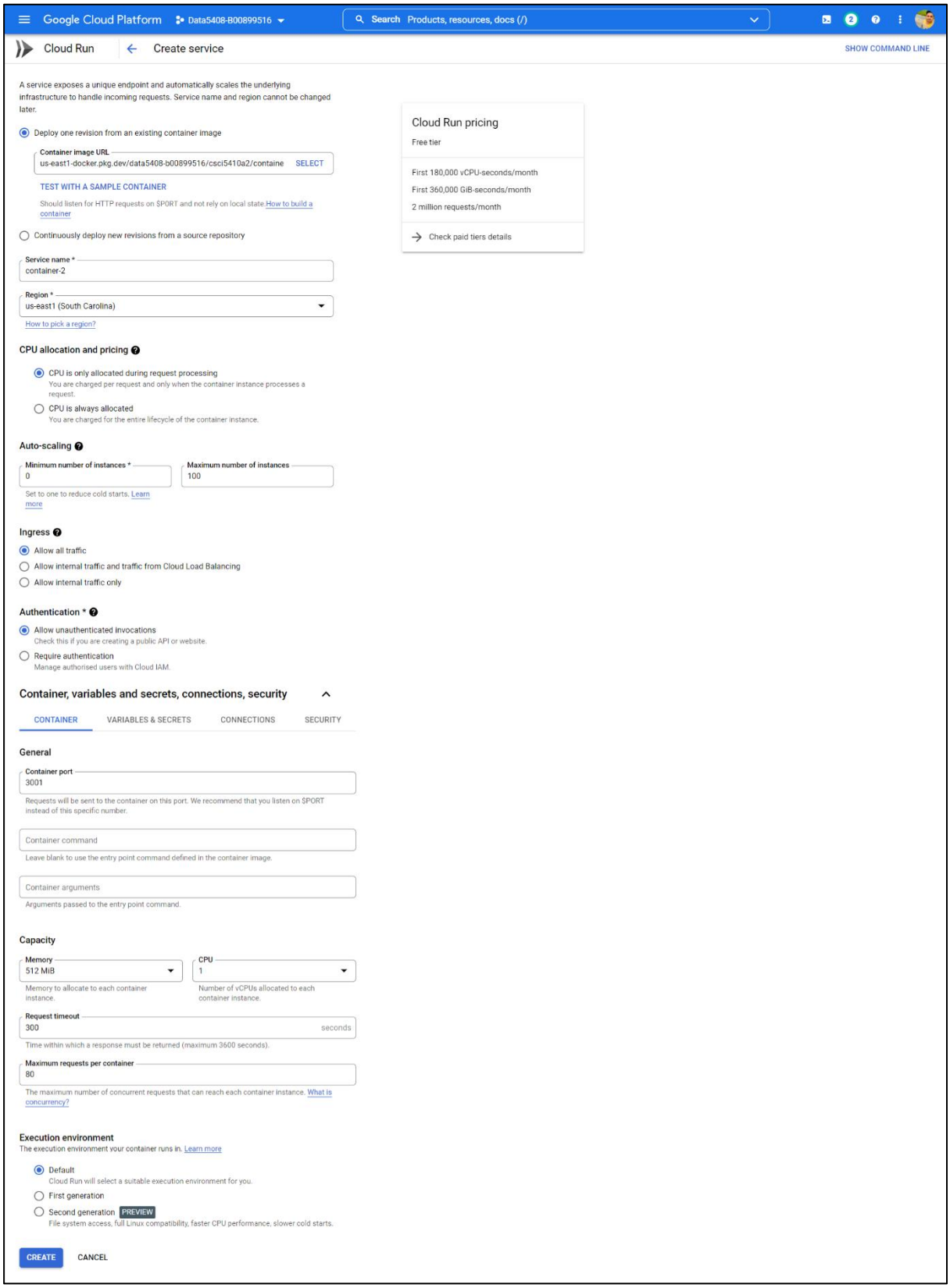



*Figure 35: Container_2 Docker image Configuration on Google Cloud Run*

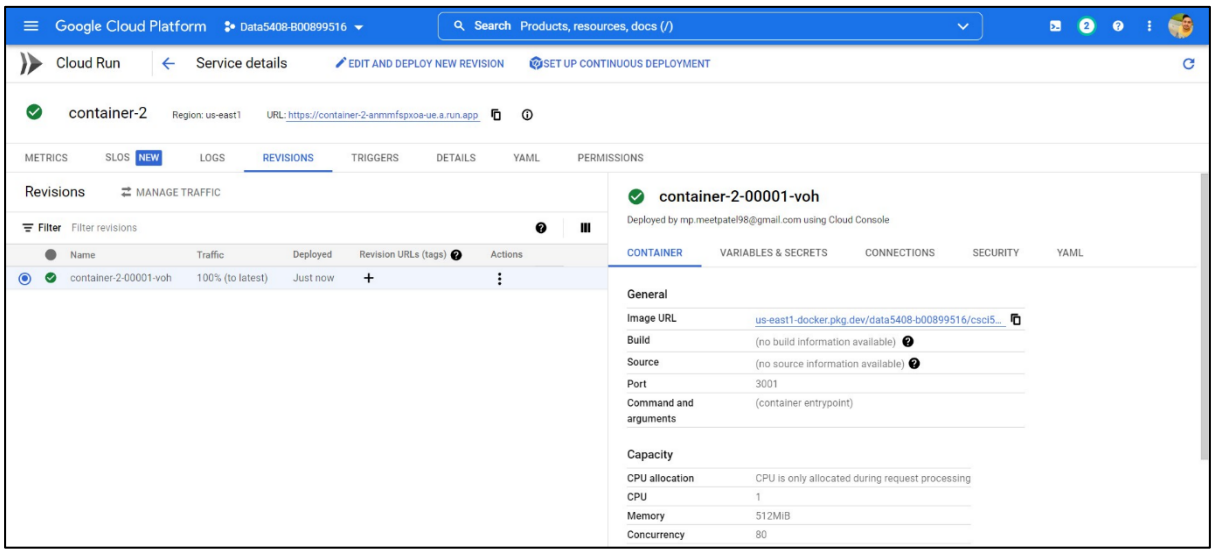



*Figure 36: Hosted continer_2 (Login React Application) on GCR (Google Cloud Run)*

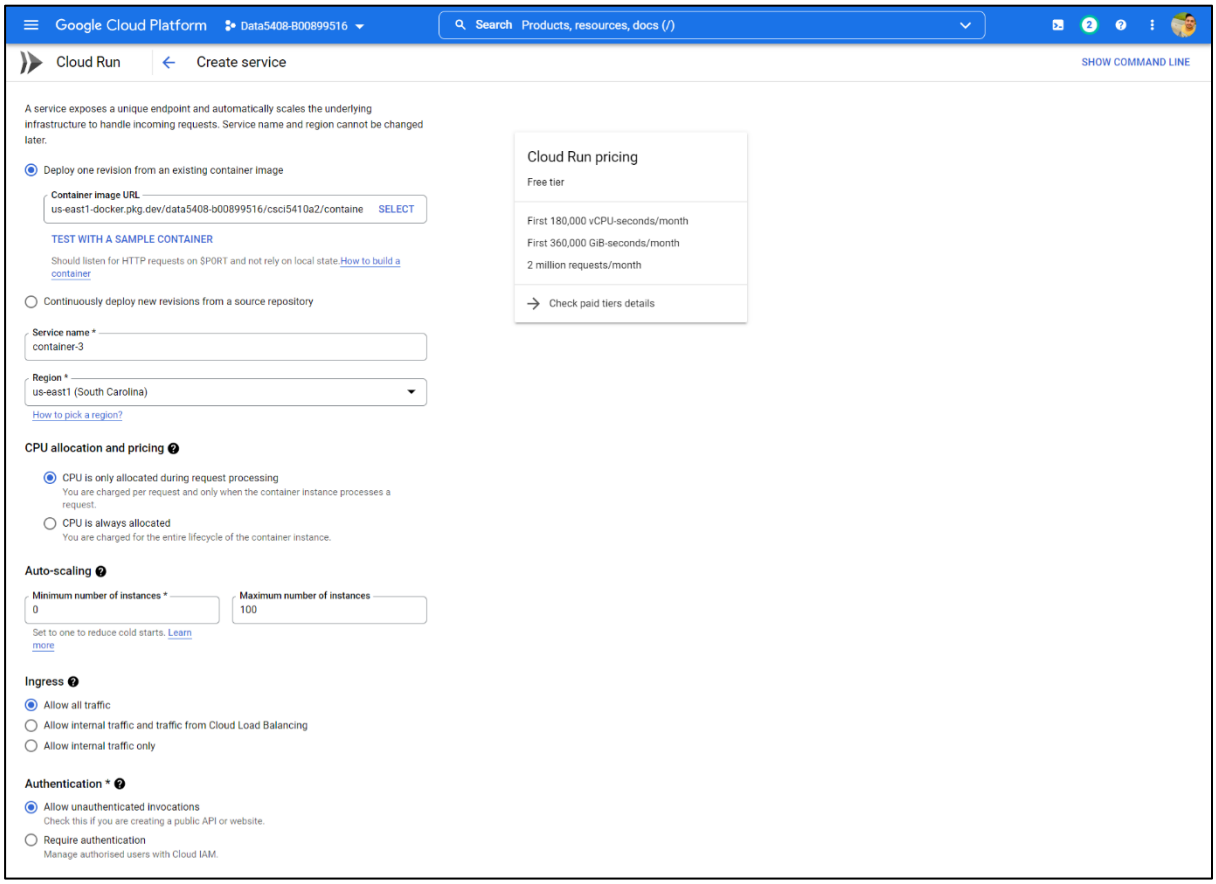



*Figure 37 : Container_3 Docker image Configuration on Google Cloud Run*

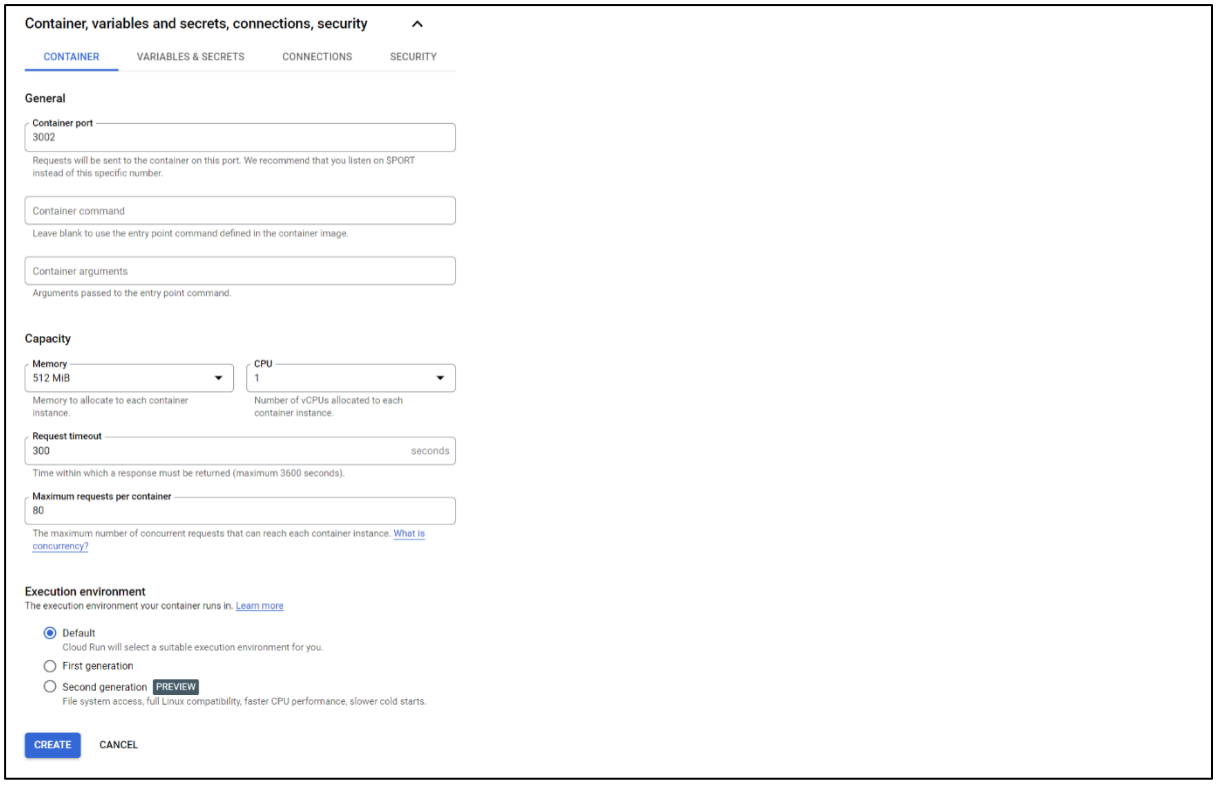



*Figure 38: **Container_3** Docker image Configuration on Google Cloud Run*

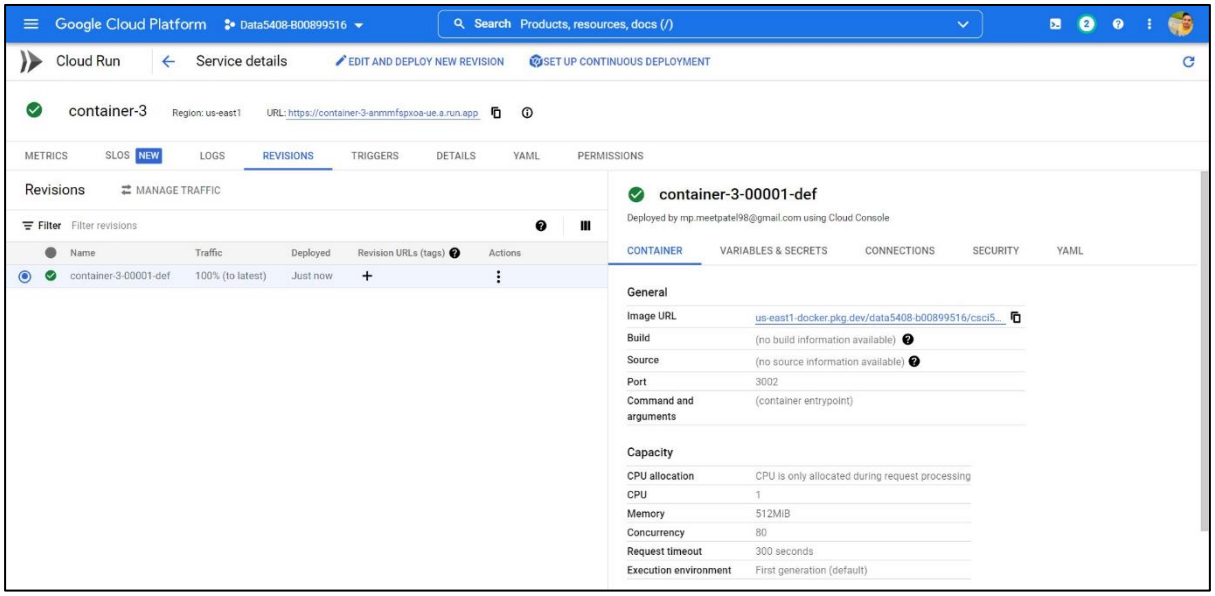



*Figure 39: Hosted **continer_3** (UserProfie React Application) on GCR (Google Cloud Run)*

# Website hosted on Google Cloud Run

## 1. Registration React Application

Figure 40 shows the **Registration Page (container_1)** hosted on Google Cloud Run

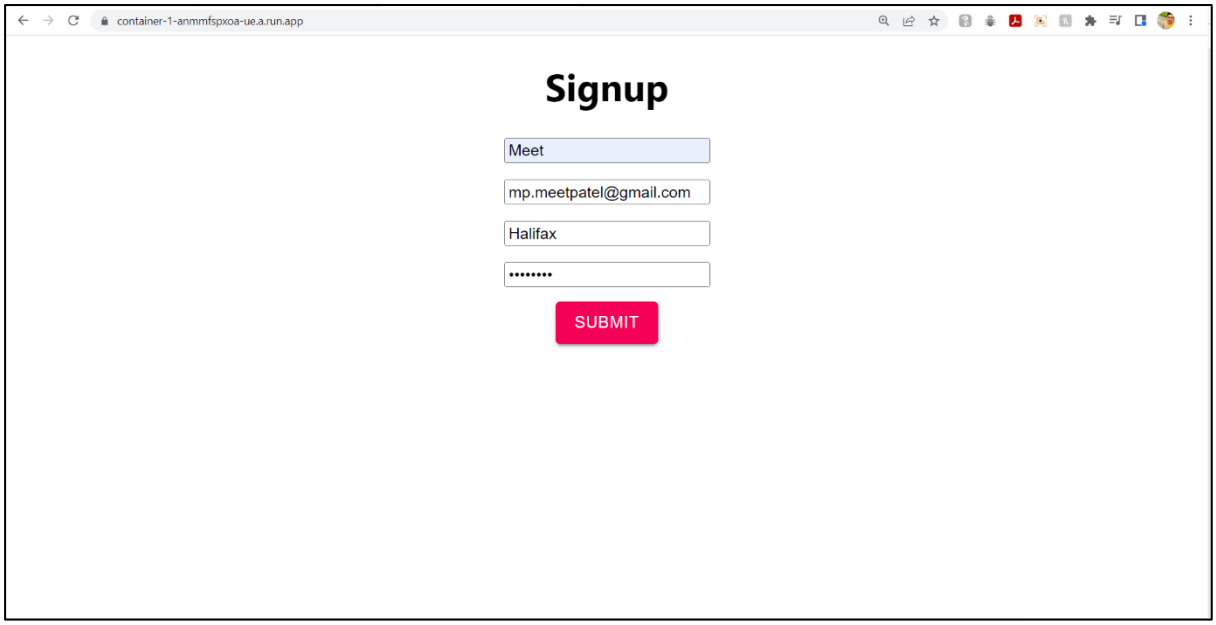



*Figure 40: **Registration Page (container_1)** hosted on Google Cloud Run*

Figure 41 and 42 shows the successful registration of the user with status set to **offline** and the data is stored in firebase database

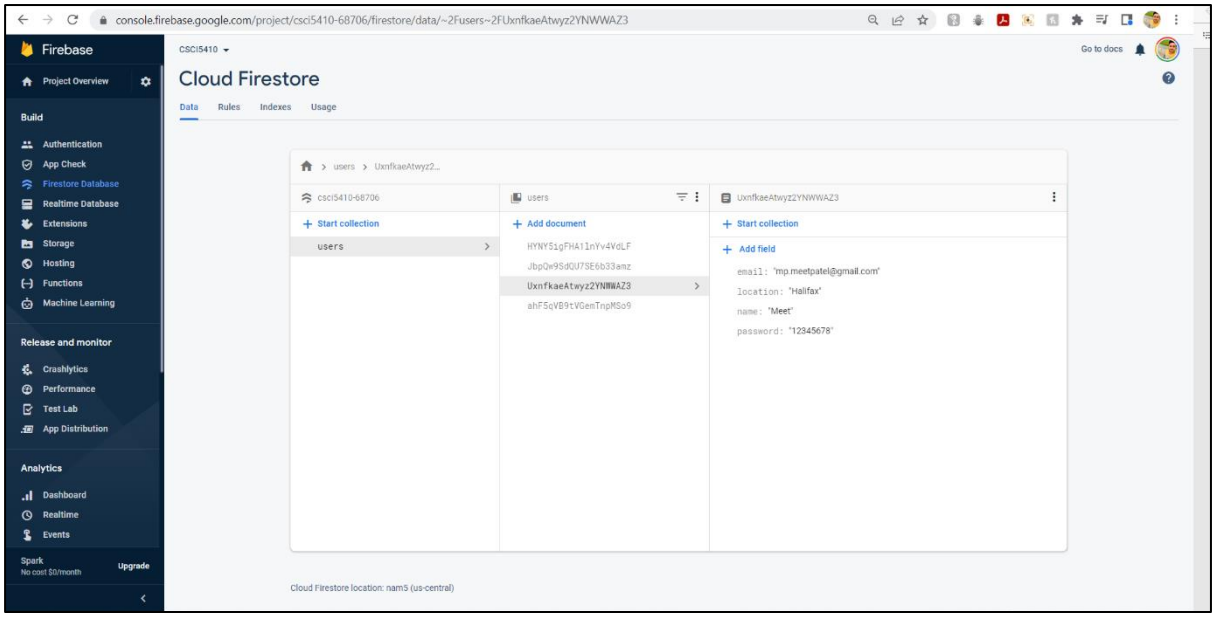



*Figure 41: Successful registration of user and data stored to firebase database*

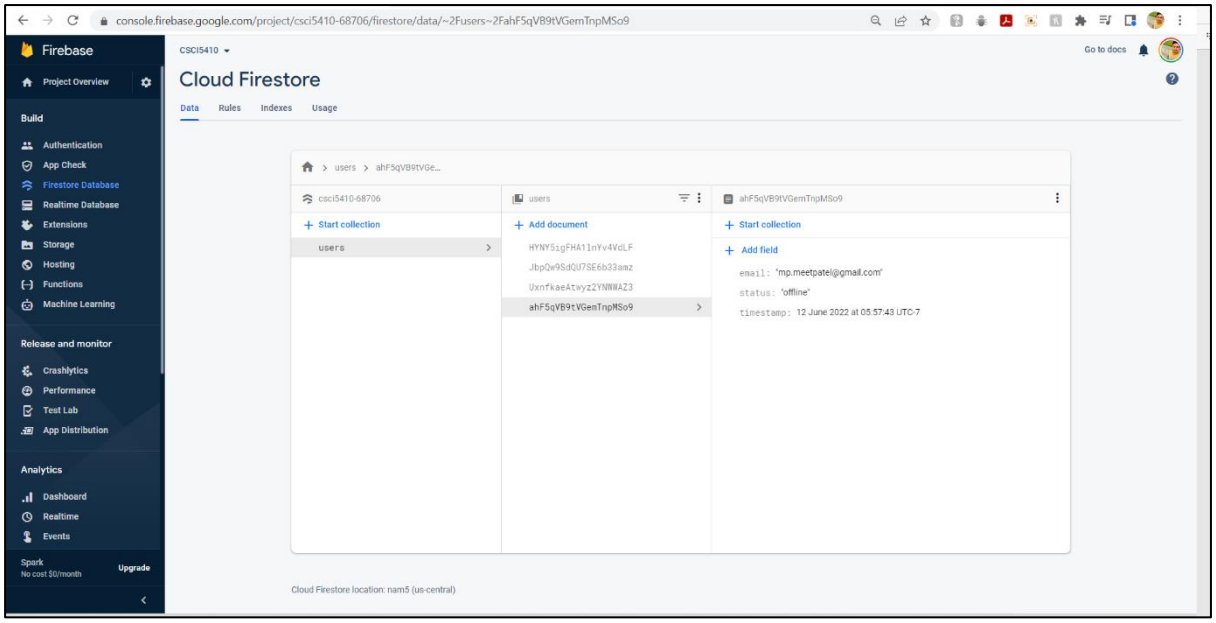



*Figure 42: After registration user's status set to **offline***

## 2. Login React Application

Figure 43 shows the **Login Page (container_2)** hosted on Google Cloud Run

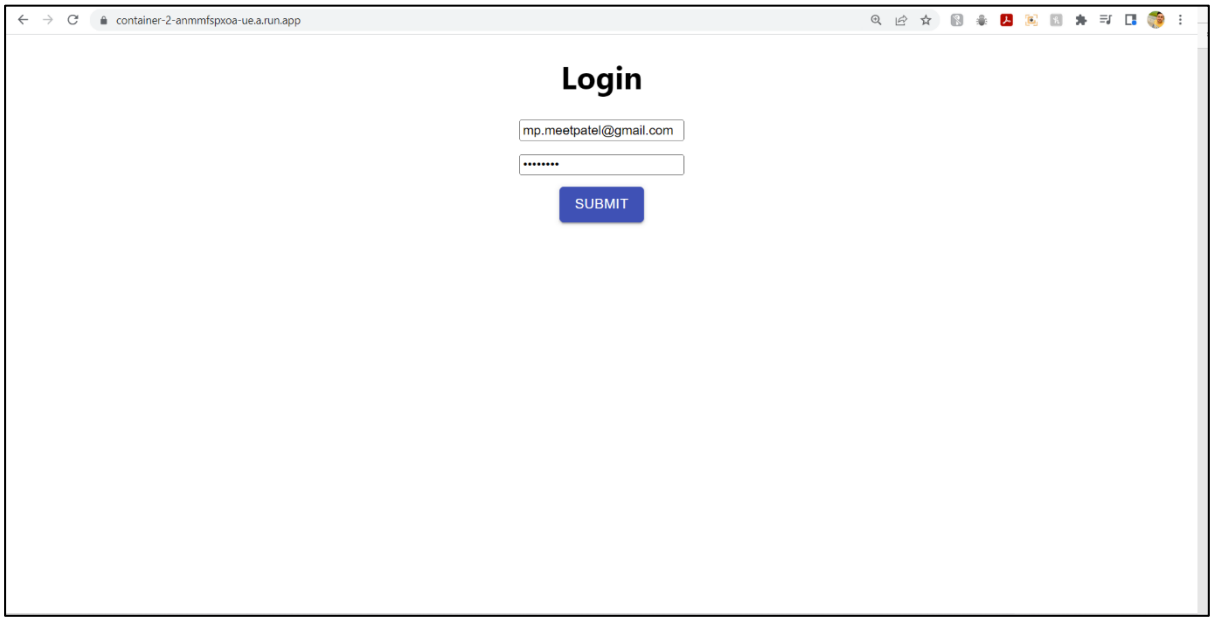



*Figure 43: **Login page (Container_2)** hosted on Google Cloud Run*

Figure 44 shows the status of the user updated to **online** after login

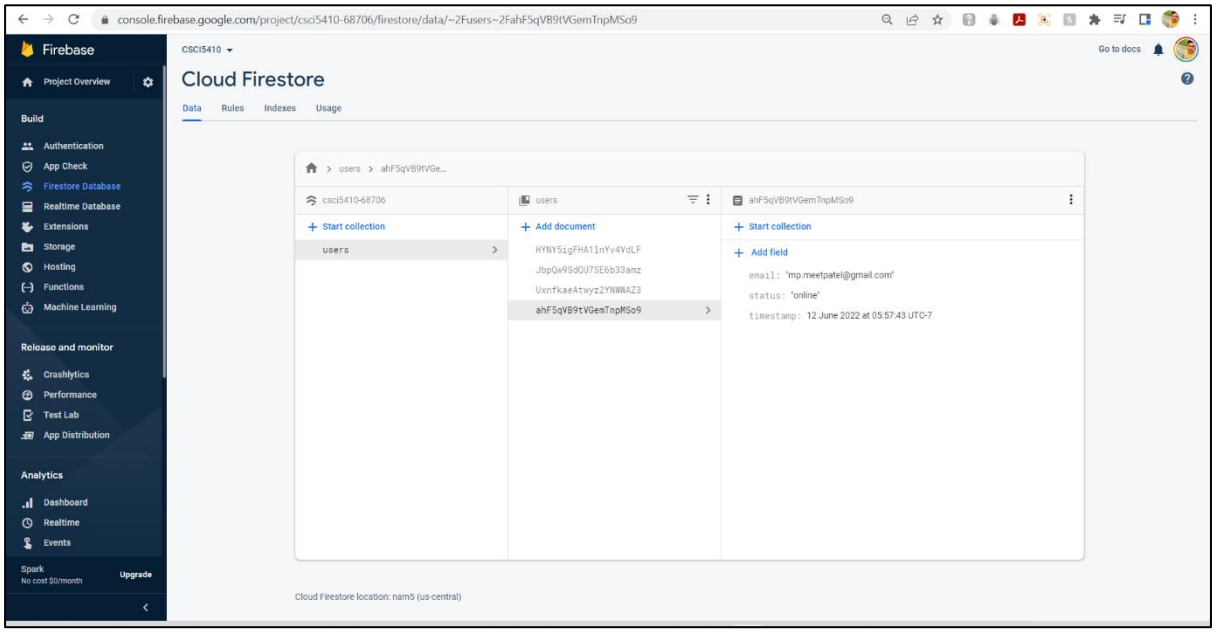



*Figure 44: Status set to **online** after login*

## 3. UserProfile React Application

Figure 45 shows the **UserProfile Page (container_3)** hosted on Google Cloud Run. This page shows the logged in user email address and a logout button.

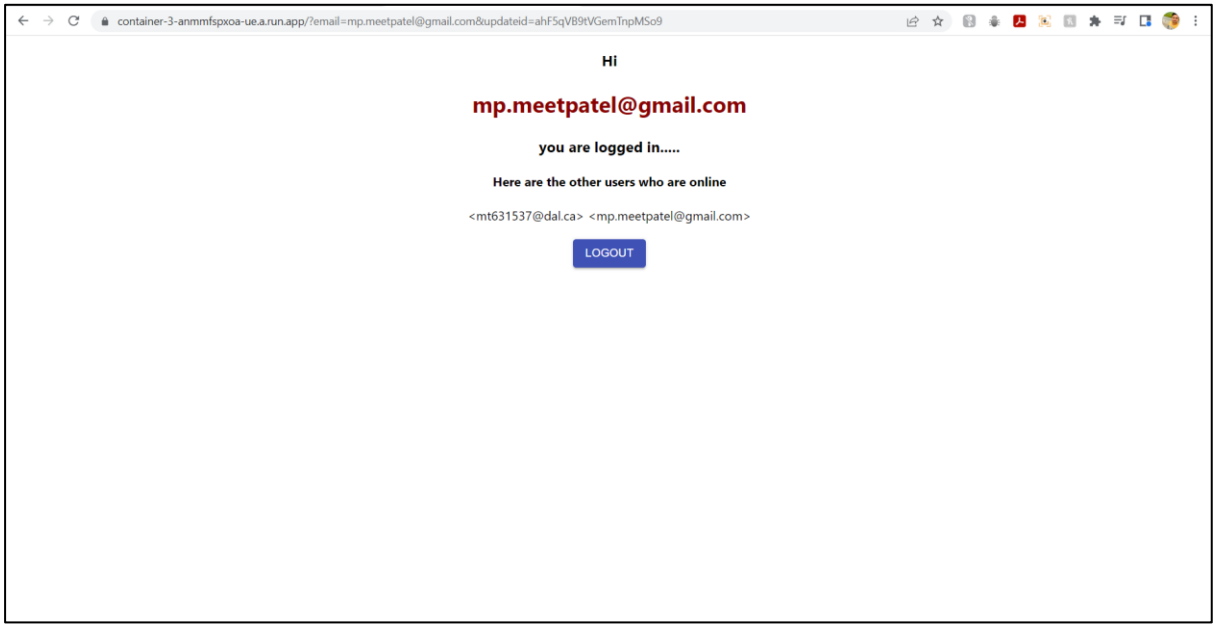



*Figure 45: **UserProfile page (container_3)** hosted on Google Cloud Run*

Figure 46 shows the status of the user updated to **offline** after user logged out. The user is then redirected to login react application.

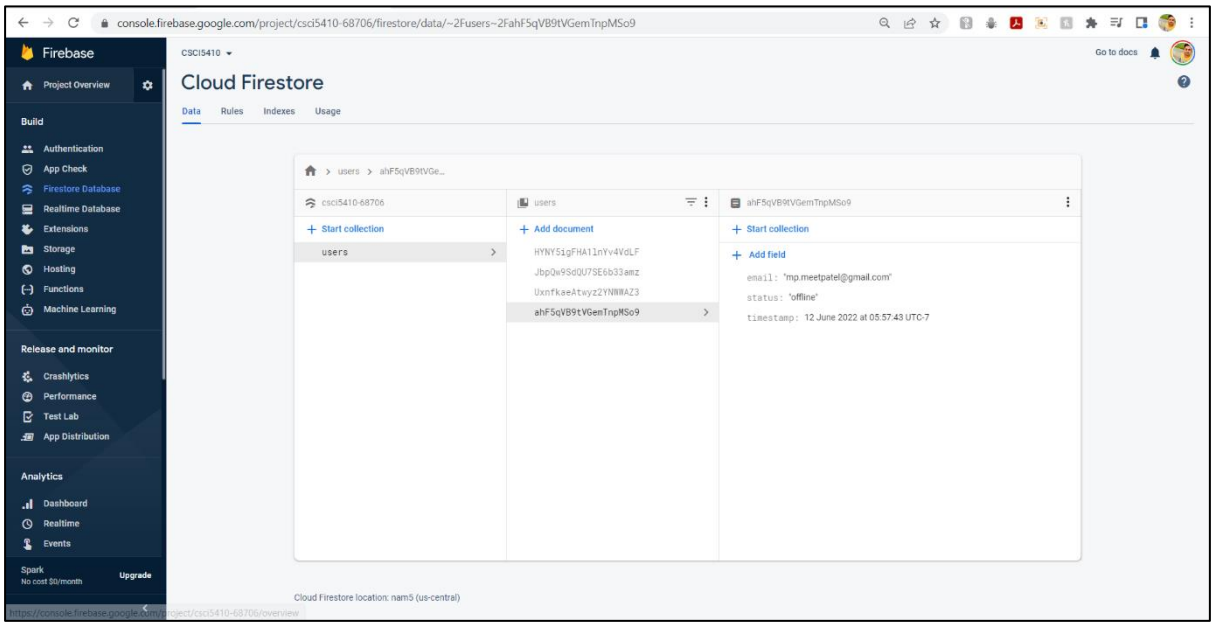



*Figure 46: User status set to **offline** after logout*

## Test Cases and Important Code

1. **Validate** method to validate name, email, location, and password - **Registration React Page**

```
function validate(data) {
    let errors = {};

    if (!data.name.trim()) {
      errors.name = "Name required";
    }

    if (!data.location.trim()) {
      errors.location = "Location required";
    }

    if (!data.email) {
      errors.email = "Email required";
      // Regex Source link: https://regexr.com/3e48o
    } else if (!/^[\w-.]+@([\w-]+\.)+[\w-]{2,4}$/.test(data.email)) {
      errors.email = "Email address is invalid";
    }

    if (!data.password) {
      errors.password = "Password required";
    } else if (data.password.length < 8) {
      errors.password = "Password needs to be 8 characters or more";
    }
    return errors;
  }
```

2. **Validate** method to validate email and password - **Login React Page (Login.js)**

```
function validate (data) {
    let errors = {};

    if (!data.email) {
      errors.email = 'Email required';
```

```
    }

    if (!data.password) {
      errors.password = 'Password required';
    }
    return errors;
  }
```

3. Methods to add data in the Firebase Firestore – **Register User Page (Register.js)**

```
const addUserInFireStore = async () => {
    try {
      const userState = await addDoc(collection(firestoreDatabase, "users"), {
        email: data.email,
        timestamp: serverTimestamp(),
        status: "offline",
      });
      console.log("Document written with ID: ", userState.id);
      updateData({
        name: "",
        email: "",
        location: "",
        password: "",
      });
      window.location.href = "http://localhost:3001/";
    } catch (e) {
      console.error("Error ", e);
    }
  };
```

4. Method to perform user login – **Login User page (Login.js)**

```
const loginUser = async () => {
    try {
      const queryObj = query(
        collection(firestoreDatabase, "users"),
        where("email", "==", data.email)
```

```
      );
      const querySnapshot = await getDocs(queryObj);
      querySnapshot.forEach(async (userDoc) => {
        if (data.password === userDoc.data().password) {
          querySnapshot.forEach(async (userDoc) => {
            if (userDoc.data().status) {
              console.log(userDoc.id, " => ", userDoc.data());
              const userState = doc(firestoreDatabase, "users", userDoc.id);
              await updateDoc(userState, {
                status: "online",
              });
              window.location.href = "http://localhost:3002/?email=" + userDoc.data().email
+"&updateid="+userDoc.id;
            }
          });
        }
        updateData({
          email: "",
          password: "",
        });
      });
    } catch (e) {
      console.error("Error while fetching data ", e);
    }
  };
```

5. Methods to fetch all the users who are online – **User Profile Page (UserProfile.js)**

```
useEffect(() => {
    async function fetchData() {
      var onlineUser = "";
      var x = 1;
      const queryObject = query(
        collection(firestoreDatabase, "users"),
        where("status", "==", "online")
      );
```

```
    const querySnapshot = await getDocs(queryObject);
    querySnapshot.forEach(async (userDoc) => {
      if (userDoc.data().status === "online") {
        if (x === 1) {
          onlineUser = " <" + userDoc.data().email + "> ";
          x = 0;
        } else {
          onlineUser = onlineUser + " <" + userDoc.data().email + "> ";
        }
      }
    });
    setUsers(onlineUser);
  }
  fetchData();
}, []);
```

## Screenshot of the Validation Performed on Website

1. **Signup page** validations

Figure 47 shows that all the fields are **mandatory**

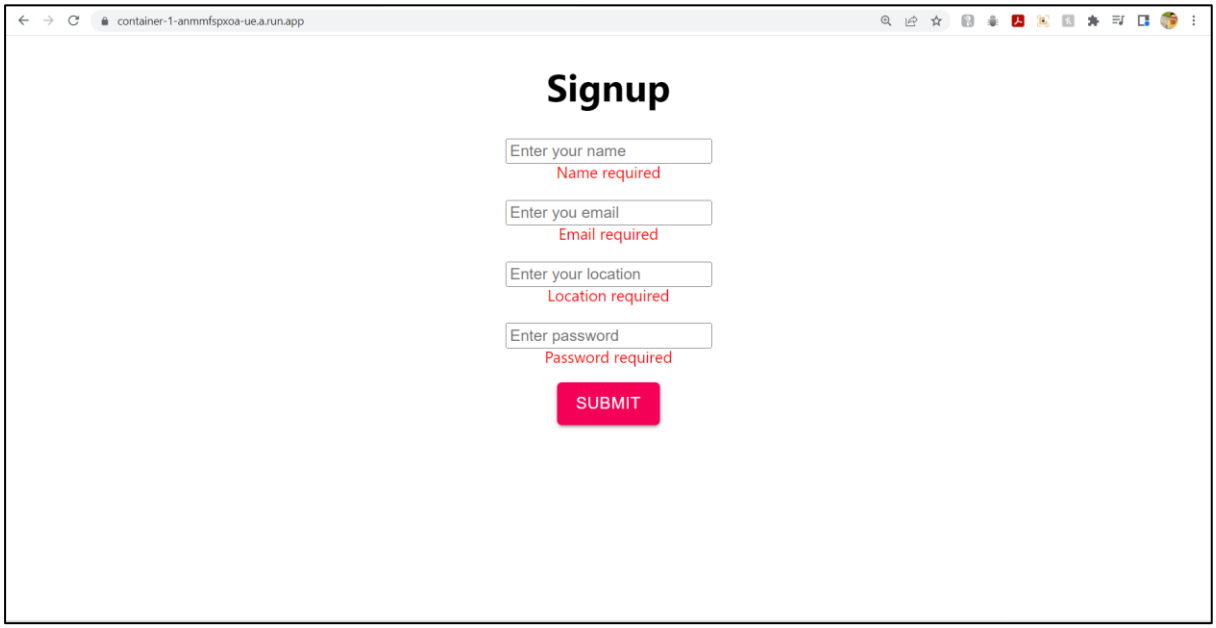



*Figure 47: All the fields are mandatory*

Figure 48 shows that **email address** must be **valid**

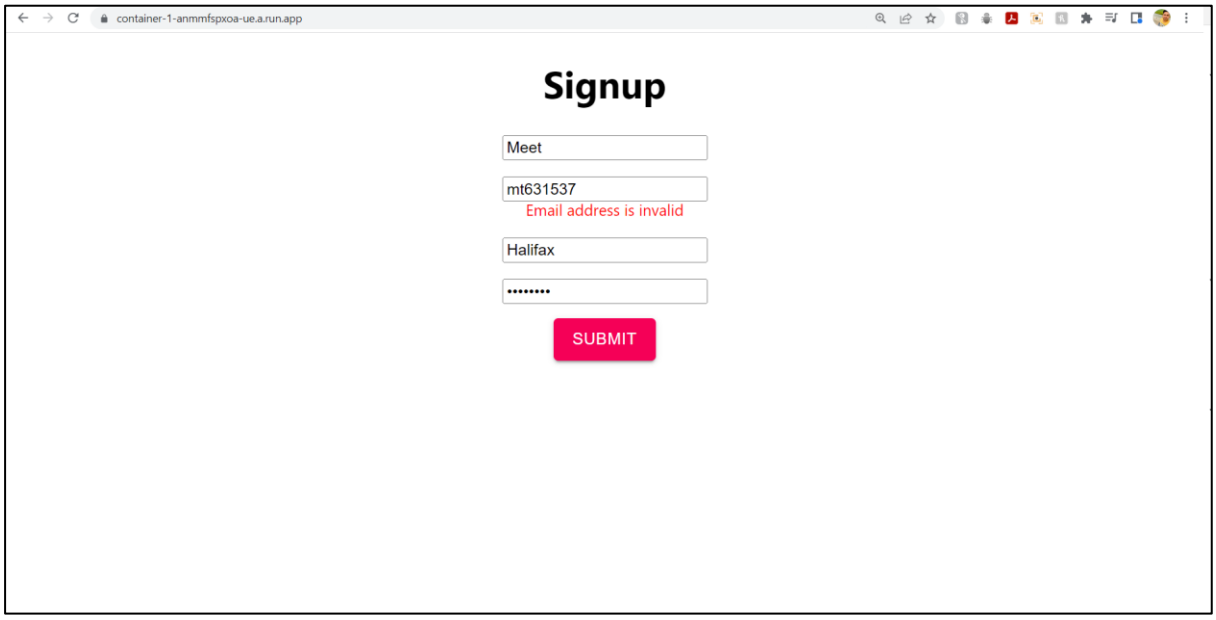



*Figure 48: Email address must be valid*

Figure 49 shows that length of password must 8 character or more

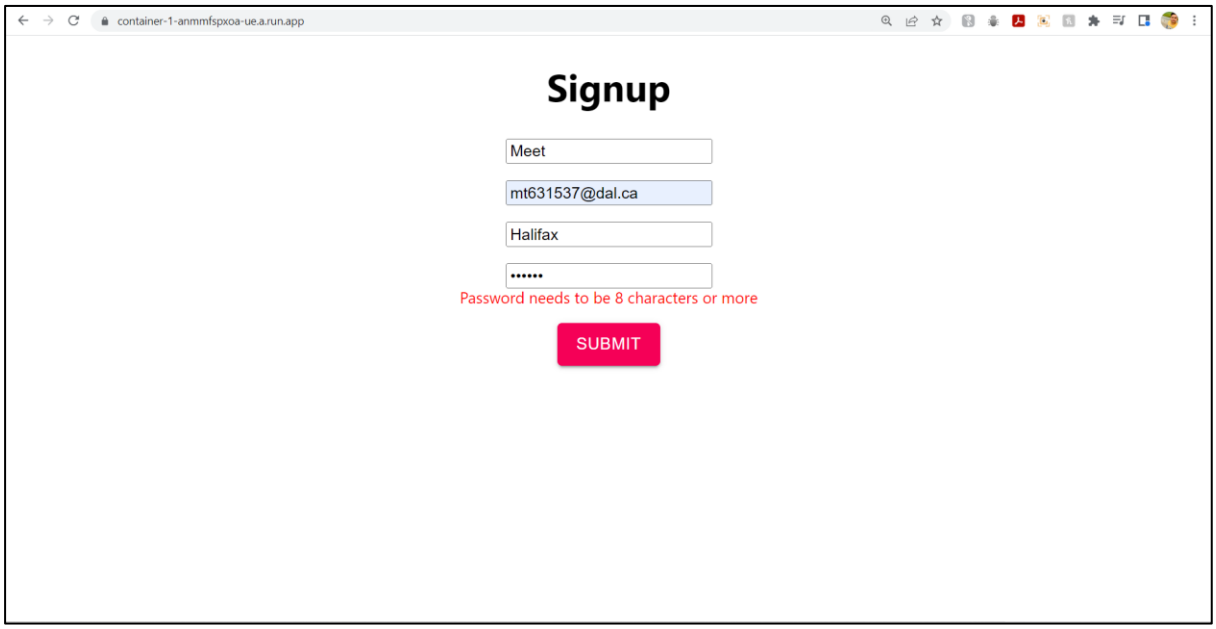



*Figure 49: Length of password must 8 character or more*

2. **Login page** validation

Figure 50 shows that **email address and password** are mandatory. In addition to that only **registered user is allowed to login**.

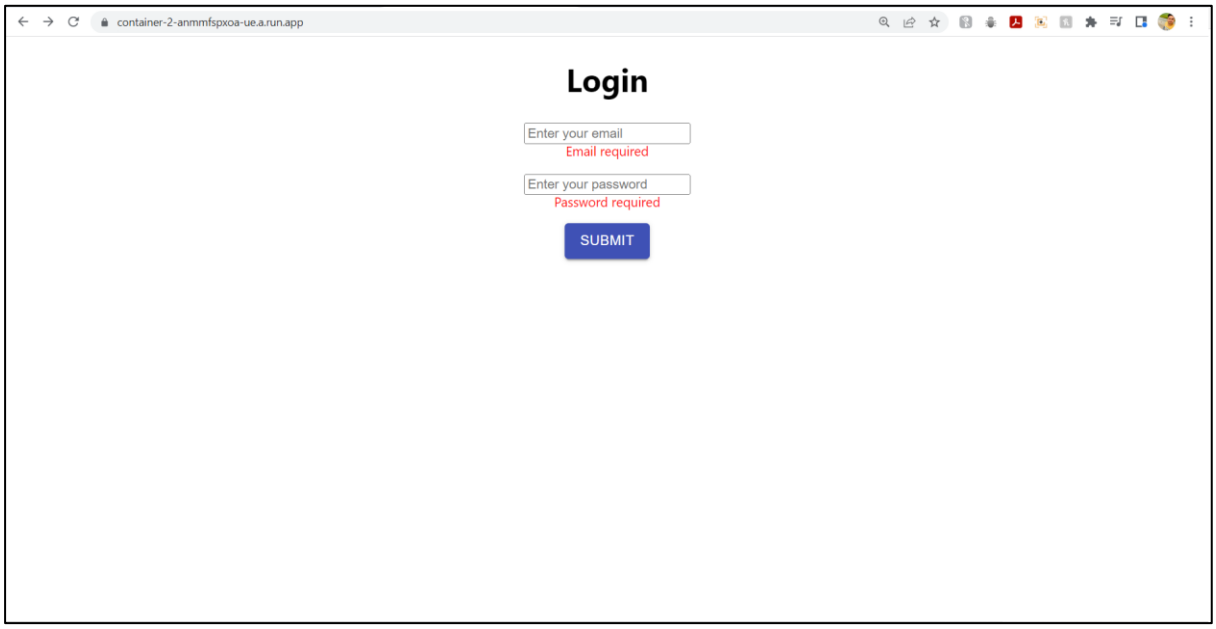



*Figure 50: Email address and password are mandatory*

## Summary

I have used following technologies to build and deploy this application.

- React JS
- Firestore Collection Database
- Google Cloud Platform
- Google Cloud Run
- Docker

For this task, I have made 3 webpages. First is the Signup/Register Page which accepts name, email, location, and password as an input and store it in the firebase database. Next is the login page which allows registered users to login to their profile using the email address and the password. Once they have logged in to their profile the user will be able to see other users who are online.

Docker containers, Google Container Registry, and Google Cloud Run are some of the modules which are used in this exercise.

**Google Cloud Run**
Google Cloud Run is a cloud-based service that lets you run an existing application. It's also a new solution for running serverless applications based on Docker containers. Hence, rather than worrying about managing infrastructure and infrastructure management, developers can focus on developing. In the developer's role, all one needs to do is specify container's location, what memory and CPU resources are required, and at last the developer needs to click on "Create "and then we can access the application with the help of http endpoints.

**Artifact Registry**
Artifact Registry is a fully managed platform that can manage container images, language packages, and operating system packages. Various features such as supporting regional repositories, Customer Managed Encryption Keys, VPC service controls, and others allow this Registry to have complete control over the software delivery process. The images can be stored in the registry and then deployed directly to Google Cloud Run using tools such as Compute Engine, Kubernetes Engine, and Google Cloud Run. We can build images which are secure then we can directly deploy on Google's provided engines so that we can run this container remotely.

**Docker Containers**
Docker Containers are Docker Virtual Machines which are runnable instance of an image. It allows you to connect a container to more than one network and by using the Docker API or the Docker CLI we can create, move, start, stop, or delete containers. Containers can be built multiple times using images, with the only difference being that the old image is deleted and a new one is used to construct the container.

## References

[1] Google, "Cloud Firestore," Google, [Online]. Available: https://firebase.google.com/products/firestore. [Accessed 11 June 2022].

[2] Docker, "Developers Love Docker," Docker, [Online]. Available: https://www.docker.com/. [Accessed 11 June 2022].

[3] Google, "Installing Cloud SDK," Google, [Online]. Available: https://cloud.google.com/sdk/docs/install. [Accessed 11 June 2022].

[4] Google, "Container Registry," Google, [Online]. Available: https://cloud.google.com/container-registry. [Accessed 11 June 2022].

[5] Google, "Cloud Run," Google, [Online]. Available: https://cloud.google.com/run. [Accessed 11 June October 2022]