CSCI 5410
Serverless Data Processing

Assignment 2
Part B
Build, deploy, and run a Containerized Application using GCP

Ritva Katrodiya
B00930131

Git Repository Link (A2 branch): [Files · A2 · Ritva Ashvinbhai Katrodiya / CSCI5410-SUMMER-23-B00930131 · GitLab (dal.ca)]

Table of Contents

Table of Figures 3
Firebase Firestore setup 6
Container images (Registration, Login and Profile React based applications) 13
Website design with firebase firestore 22
Project Creation on GCP 29
Artifact Registry Setup 30
Cloud Run Configuration 45
Deployed Web application 55
Test Cases 61
Unit tests 70
Summary 74
References 75

Table of Figures

Figure 1 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2] 6
Figure 2 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2] 7
Figure 3 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2] 7
Figure 4 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2] 8
Figure 5 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2] 8
Figure 6 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2] 9
Figure 7 Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2] 9
Figure 8 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2] 10
Figure 9 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2] 10
Figure 10 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2] 11
Figure 11 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2] 11
Figure 12 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2] 12
Figure 13 built docker image named **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container1/container1-image** for Registration react based application 13
Figure 14 image **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container1/container1-image** successfully created on docker [1] 14
Figure 15 image **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container1/container1-image** is successfully running in container**(serene_murdock)** [1] 15
Figure 16 built docker image named **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container2/container2-image** for Login react based application 16
Figure 17 Image **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container2/container2-image** successfully created on docker [1] 17
Figure 18 Image **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container1/container1-image** is successfully running in container**(sharp_swirles)** [1] 18
Figure 19 built docker image named **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container3/container3-image** for Profile react based application 19
Figure 20 Image **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container3/container3-image** successfully created on docker [1] 20
Figure 21 **Image us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container3/container3-image** is successfully running in container**(crazy_goldberg)** [1] 21
Figure 22 Registration react based application 22
Figure 23 User has successfully registered 23
Figure 24 user details successfully stored in **Reg** collection of firestore firebase database named **csci5410-A2-part2** [2] 24
Figure 25 **state** collection of firestore firebase database named **csci5410-A2-part2** conatins registered user details as well as timestamp and online/offline status [2] 25
Figure 26 Login react based application 26
Figure 27 Email and password validated successfully 26
Figure 28 On successful validation of email and password, user status changed to **online** [2] 27
Figure 29 Details of All registered users (Reg collection) [2] 27
Figure 30 State collection with registered users' status [2] 28
Figure 31 Profile page with all the online users 28
Figure 32 Project creation on GCP (CSCI5410-A2-PartA) [7] 29

Figure 33 Project created successfully on GCP (CSCI5419-A2-PartA) [7] 29
Figure 34 Artifact Registry of Google cloud platform [5] 30
Figure 35 Creation of repository named container1 on Artifact Registry for Dockerized image for registration react application [5] 31
Figure 36 Creation of repository named container1 on Artifact Registry for Dockerized image for registration react application [5] 31
Figure 37 Successfully created registry named conatiner1 in Artifact Registry [5] 32
Figure 38 Creation of repository named container2 on Artifact Registry for Dockerized image for login react application [5] 33
Figure 39 Creation of repository named container2 on Artifact Registry for Dockerized image for login react application [5] 34
Figure 40 Successfully created registry named conatiner2 in Artifact Registry [5] 34
Figure 41 Creation of repository named container3 on Artifact Registry for Dockerized image for profile react application [5] 35
Figure 42 Creation of repository named container3 on Artifact Registry for Dockerized image for profile react application [5] 36
Figure 43 Successfully created registry named conatiner3 in Artifact Registry [5] 36
Figure 44 Empty repository container1[5] 37
Figure 45 Empty repository container2 [5] 37
Figure 46 Empty repository container3 [5] 38
Figure 47 push Docker image namely **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container1/container1-image** to Google Artifact registry 39
Figure 48 Docker image namely **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container1/container1-image** successfully pushed to Google Artifact registry [5] 40
Figure 49 push Docker **image namely us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container2/container2-image** to Google Artifact registry 41
Figure 50 Docker image namely **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container2/container2-image** successfully pushed to Google Artifact registry [5] 42
Figure 51 push Docker image namely us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container3/container3-image to Google Artifact registry 43
Figure 52 Docker image namely **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container3/container3-image** successfully pushed to Google Artifact registry [5] 44
Figure 53 Configuration of conatiner1-image on Cloud run [4] 45
Figure 54 Configuration of conatiner1-image on Cloud run [4] 46
Figure 55 Configuration of conatiner1-image on Cloud run [4] 46
Figure 56 Configuration of conatiner1-image on Cloud run [4] 47
Figure 57 Configuration of conatiner1-image on Cloud run [4] 47
Figure 58 Configuration of conatiner1-image on Cloud run [4] 48
Figure 59 Conatiner1-image successfully deployed on cloud run [4] 48
Figure 60 Configuration of conatiner2-image on Cloud run [4] 49
Figure 61 Configuration of conatiner2-image on Cloud run [4] 49
Figure 62 Configuration of conatiner2-image on Cloud run [4] 50
Figure 63 Configuration of conatiner2-image on Cloud run [4] 50
Figure 64 Configuration of conatiner2-image on Cloud run [4] 51
Figure 65 Conatiner2-image successfully deployed on cloud run [4] 51

Figure 66 Configuration of conatiner3-image on Cloud run [4] 52
Figure 67 Configuration of conatiner3-image on Cloud run [4] 52
Figure 68 Configuration of conatiner3-image on Cloud run [4] 53
Figure 69 Configuration of conatiner3-image on Cloud run [4] 53
Figure 70 Conatiner3-image successfully deployed on cloud run [4] 54
Figure 71 successfully deployed three images on Google Cloud Run [4] 54
Figure 72 Deployed Registration react application on Google Cloud Run [4] 55
Figure 73 User Registered successfully [4] 56
Figure 74 User's details stored successfully in Reg collection [2] 56
Figure 75 State collection with user's details, timestamps, and status [2] 57
Figure 76 Deployed Login react application on Google Cloud Run [4] 58
Figure 77 User’s Credentials Validate Successfully [4] 58
Figure 78 On successful login, user’s status changed form offline to online [2] 59
Figure 79 Deployed Profile react application on Google Cloud Run [4] 60
Figure 80 Validation for Registration page (name) [4] 63
Figure 81 Validation for Registration page (location) [4] 63
Figure 82 Validation for Registration page (email) [4] 64
Figure 83 Validation for Registration page (password) [4] 64
Figure 84 Validation for login page (email) [4] 66
Figure 85 Validation for login page (password) [4] 66
Figure 86 Test case of invalid email and password [4] 67
Figure 87 Unit tests for Registration page 71
Figure 88 Unit tests for Login page 73

Firebase Firestore setup

Figures 1 to 10 illustrate the screenshots of the configuration setup for the Firebase Firestore database (Project name: csci5410-A2-part2).

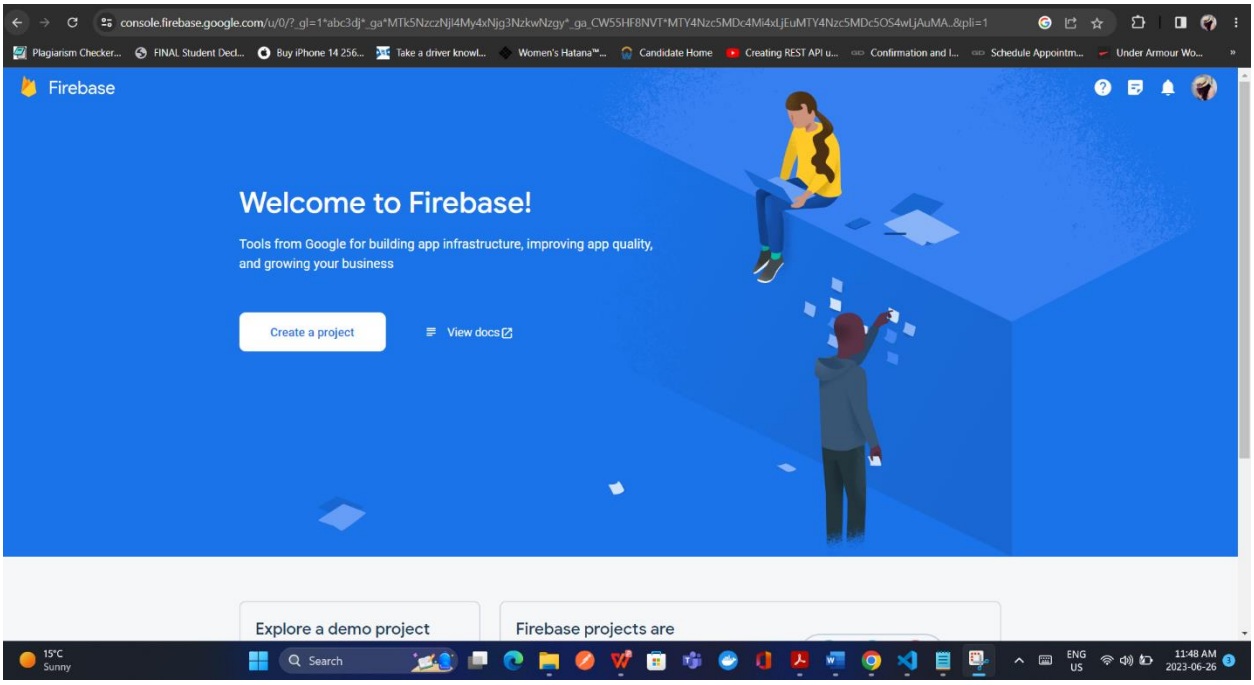

Figure 1 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2]

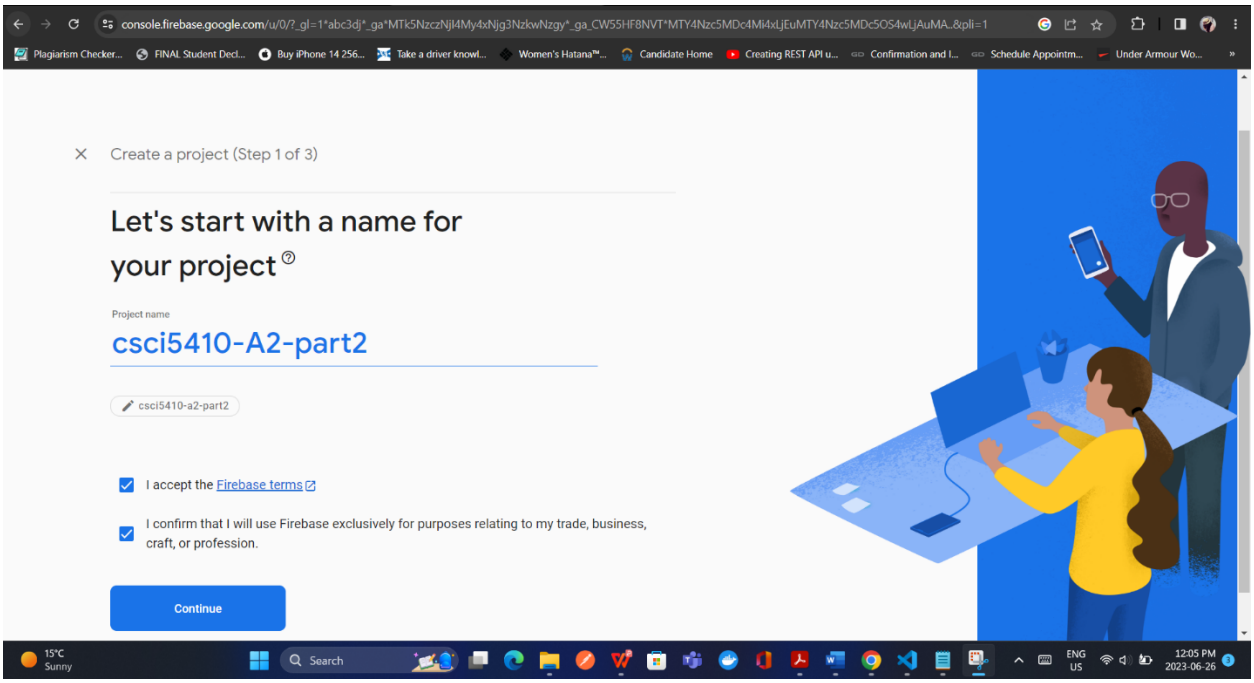

Figure 2 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2]

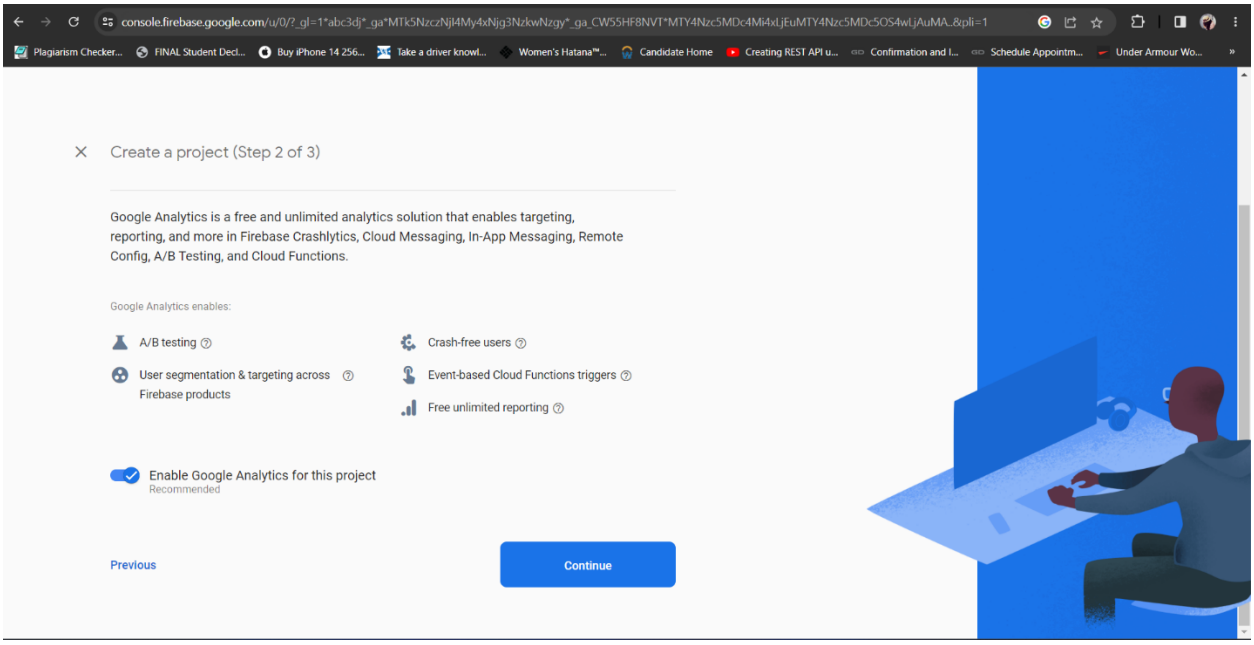

Figure 3 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2]

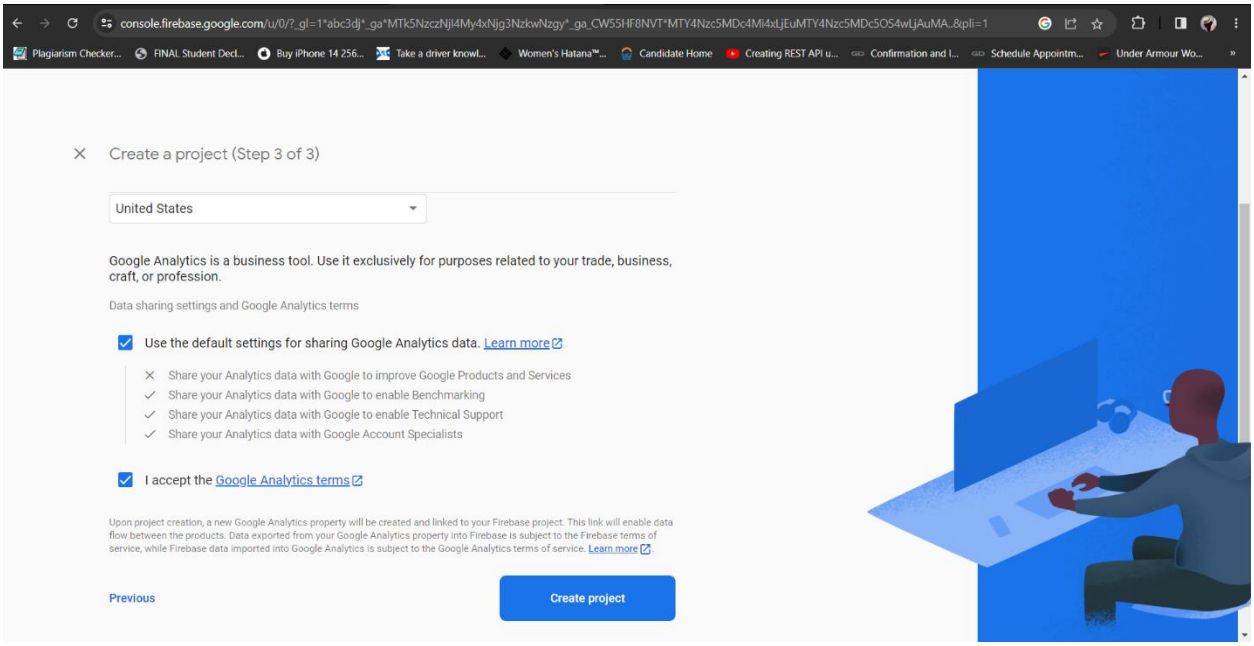

Figure 4 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2]

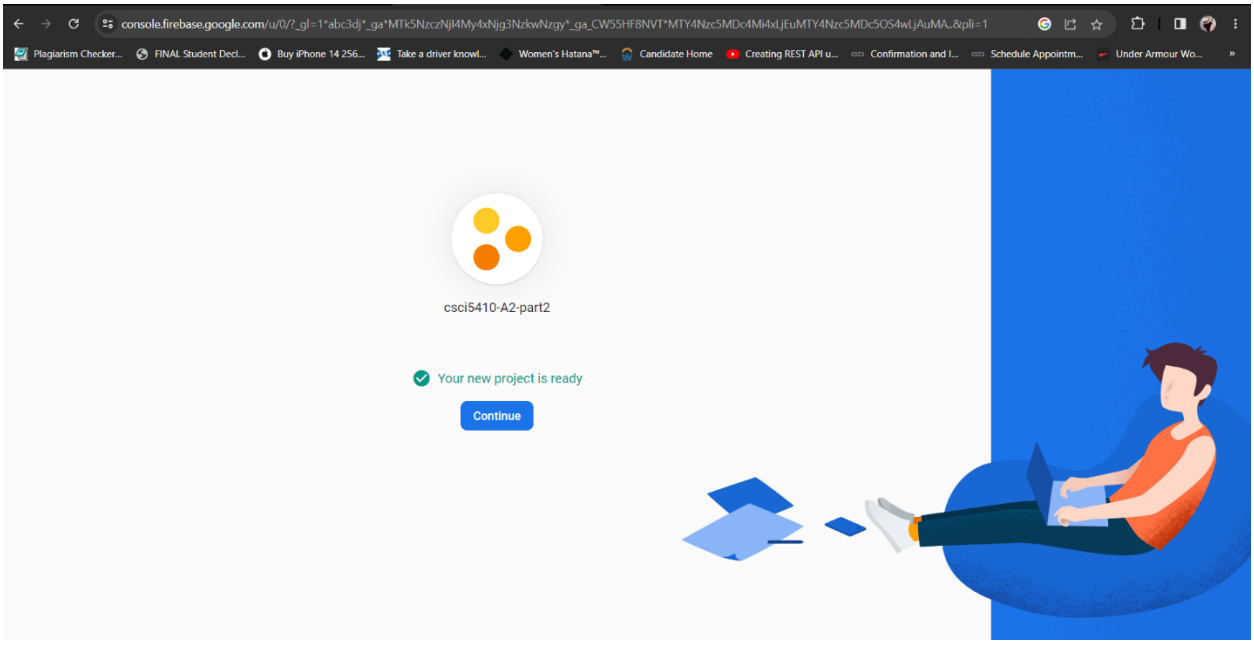

Figure 5 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2]

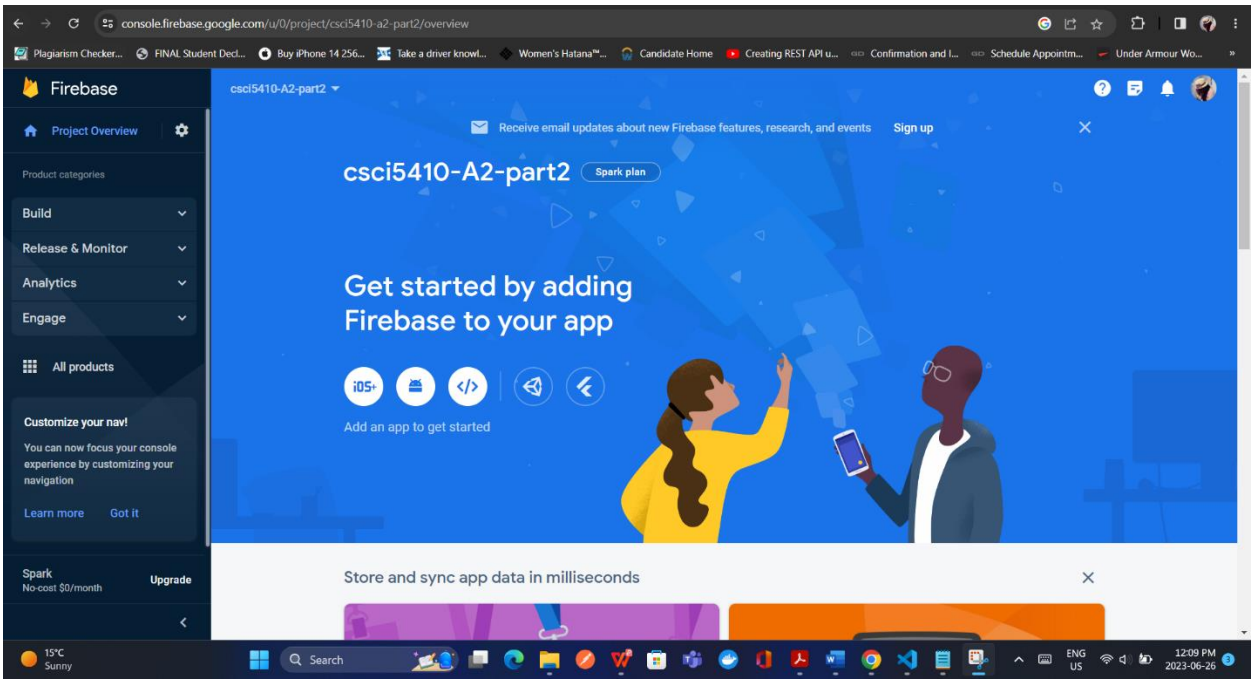

Figure 6 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2]

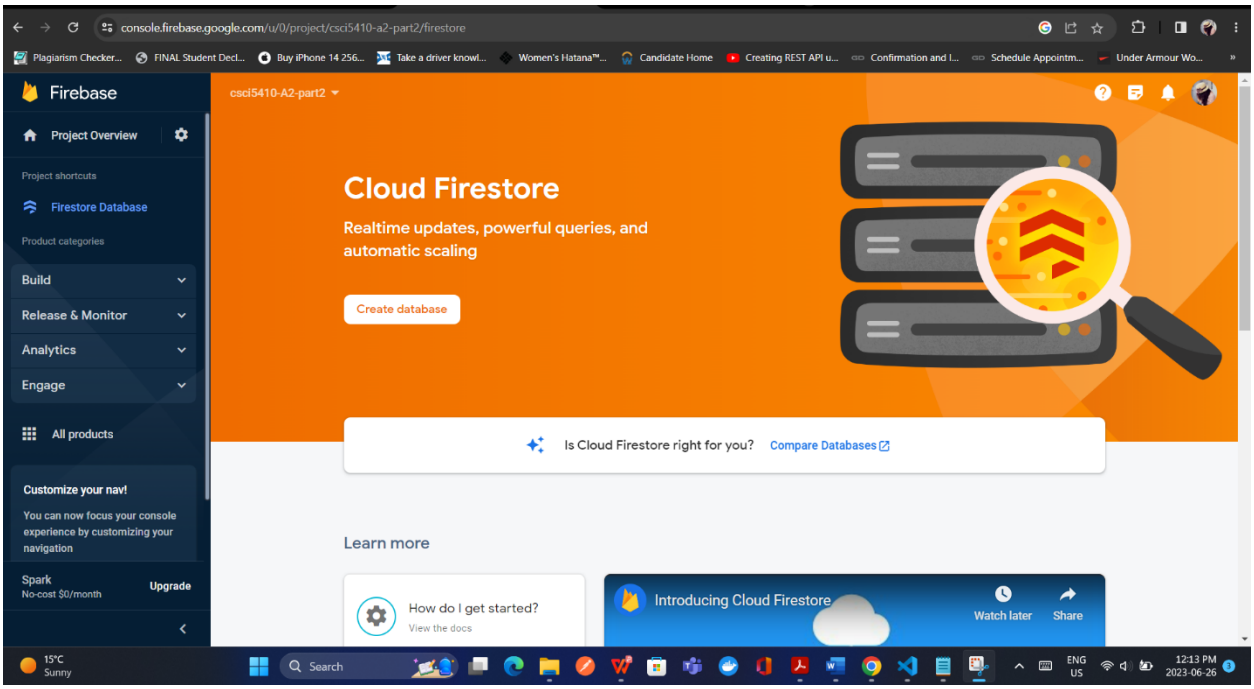

Figure 7 Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2]

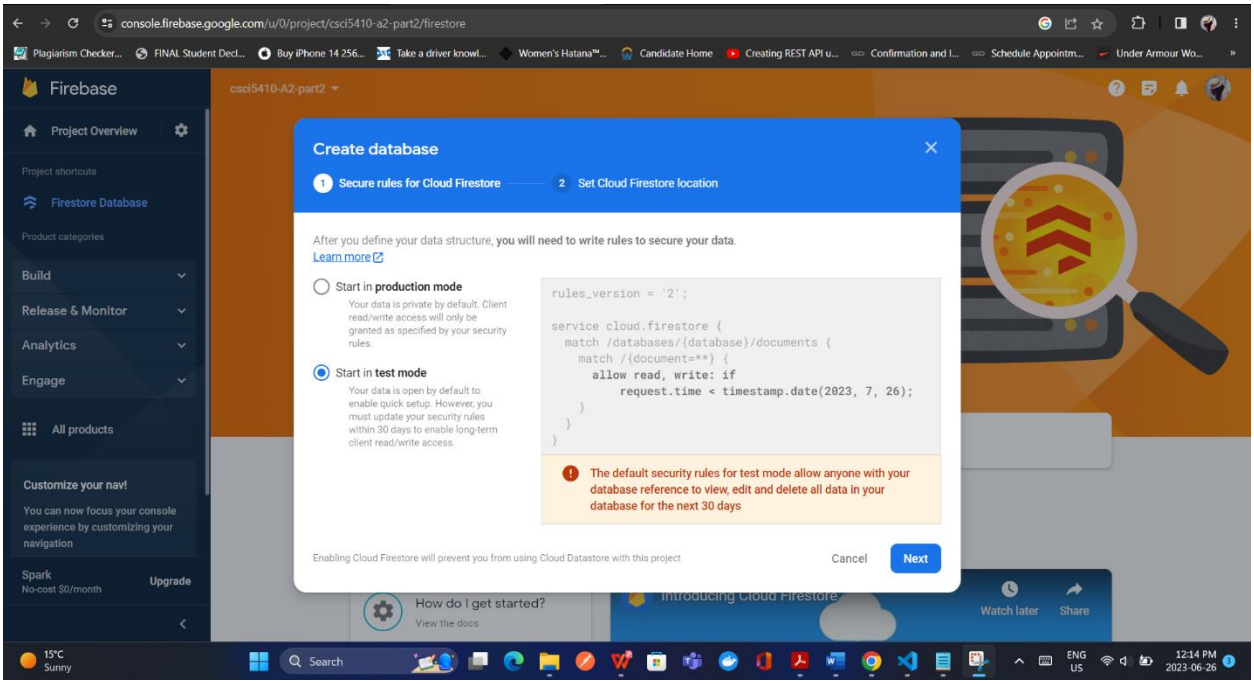

Figure 8 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2]

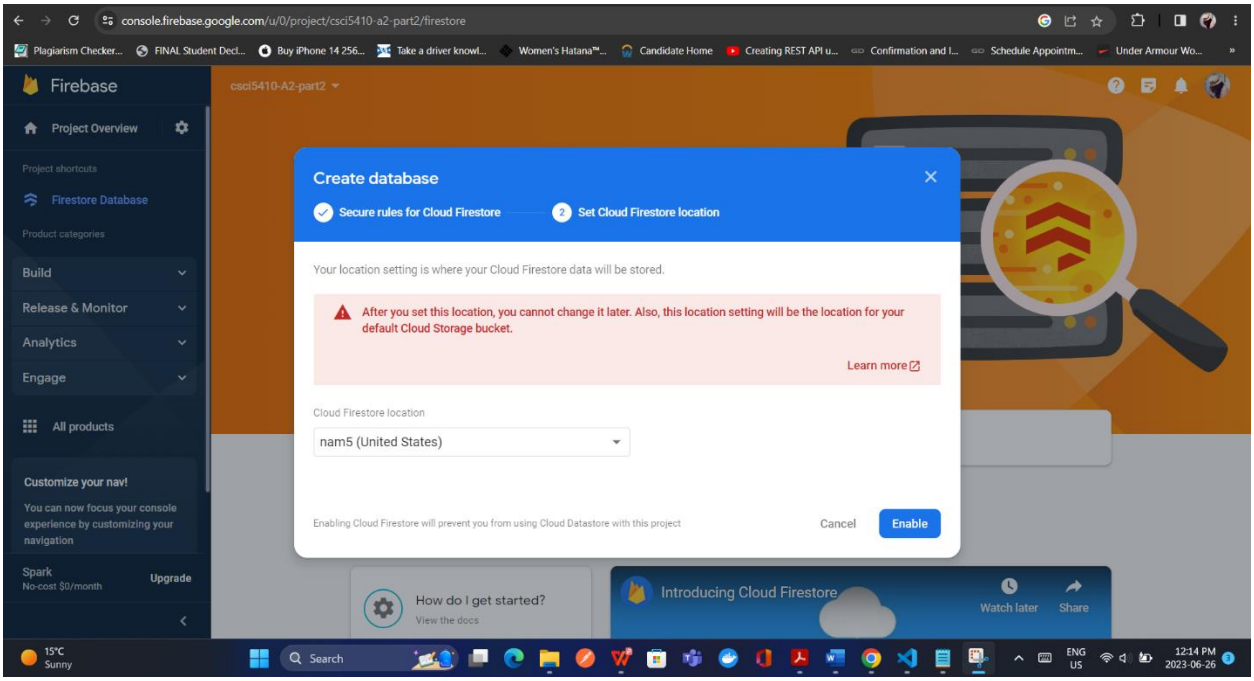

Figure 9 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2]

Figure 10 shows that the Firestore database is empty at the beginning.

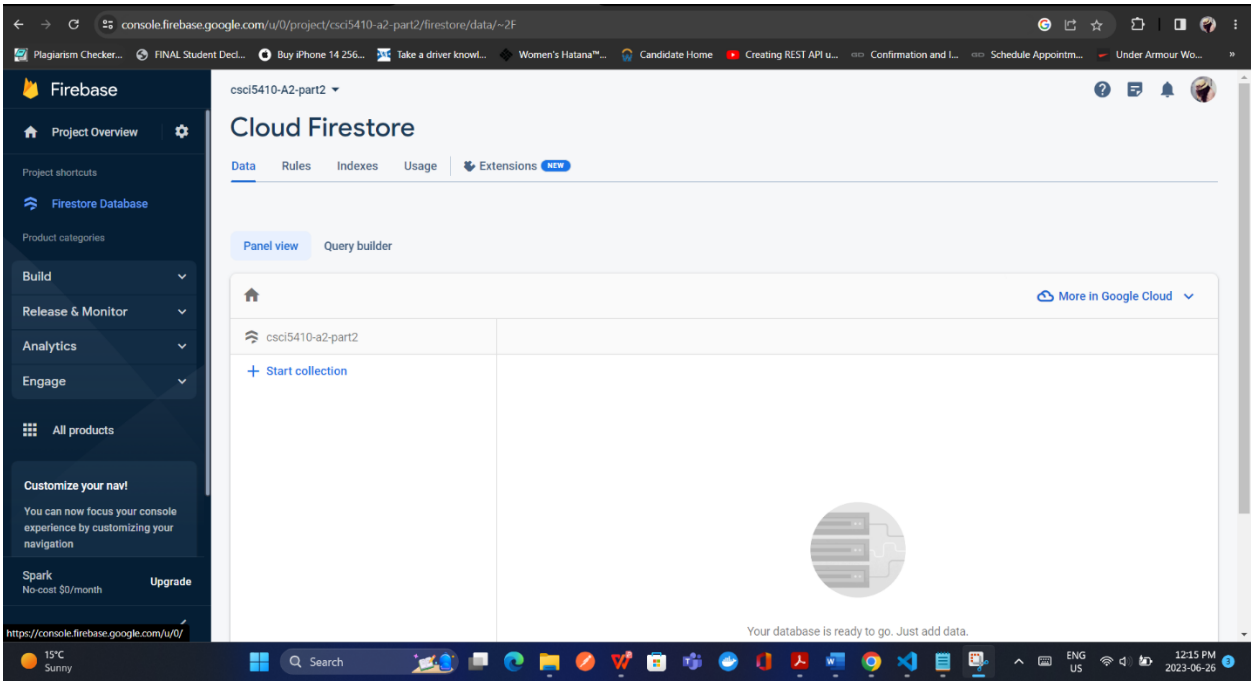

Figure 10 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2]

Figures 11 and 12 exhibit the process of adding the Firestore database to web applications.

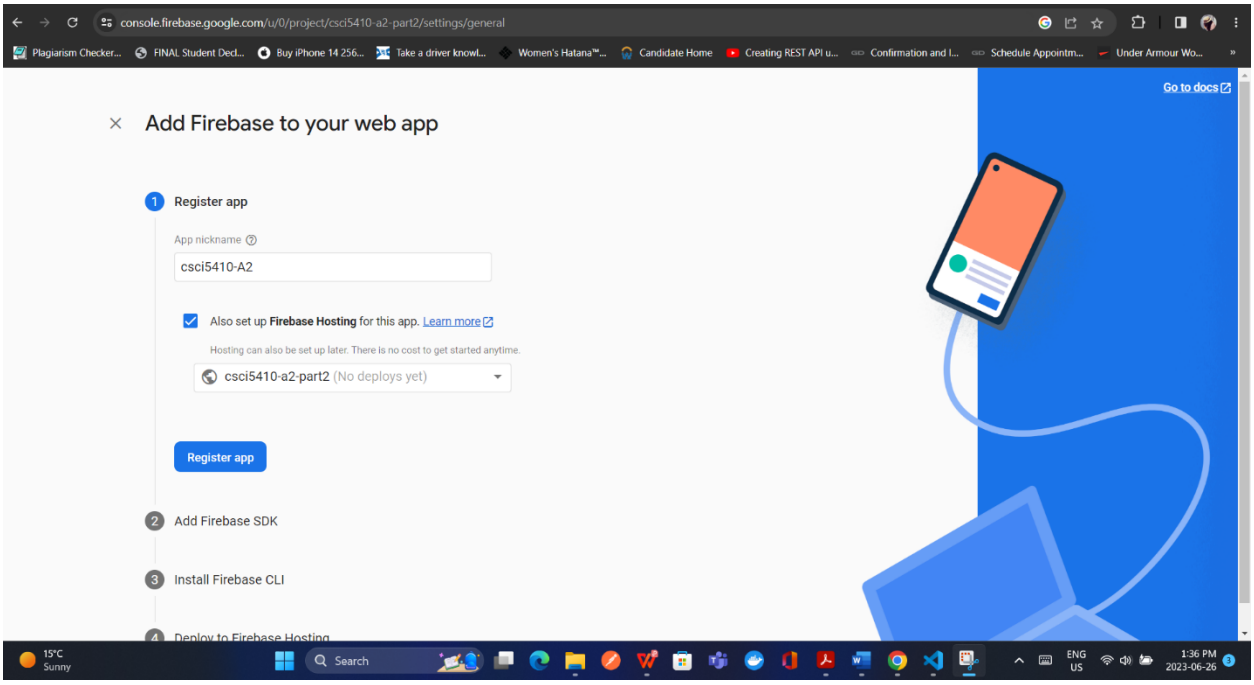

Figure 11 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2]

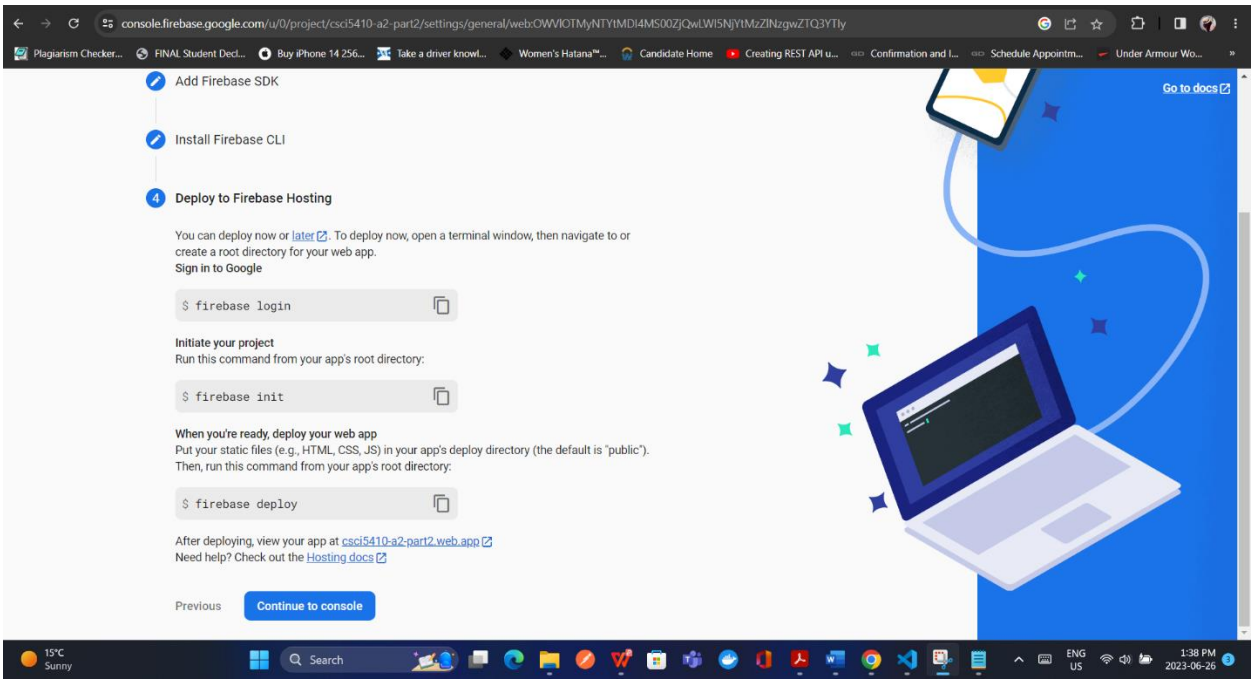

Figure 12 Project Creation in Firebase Firestore (Project name: csci5410-A2-part2) [2]

Container images (Registration, Login and Profile React based applications)

Container 1 (Registration react based application)

A Dockerized React-based application has been developed for the purpose of user registration. This application receives registration details (Name, location, email and password) from the frontend and stores them in a Firebase Firestore database named "csci5410-A2-part2." The database consists of two collections: "Reg," which stores the user registration data, and "State," which stores the user registration data along with a timestamp and online/offline status.

Figure 13 shows image of registration react based application is built successfully.

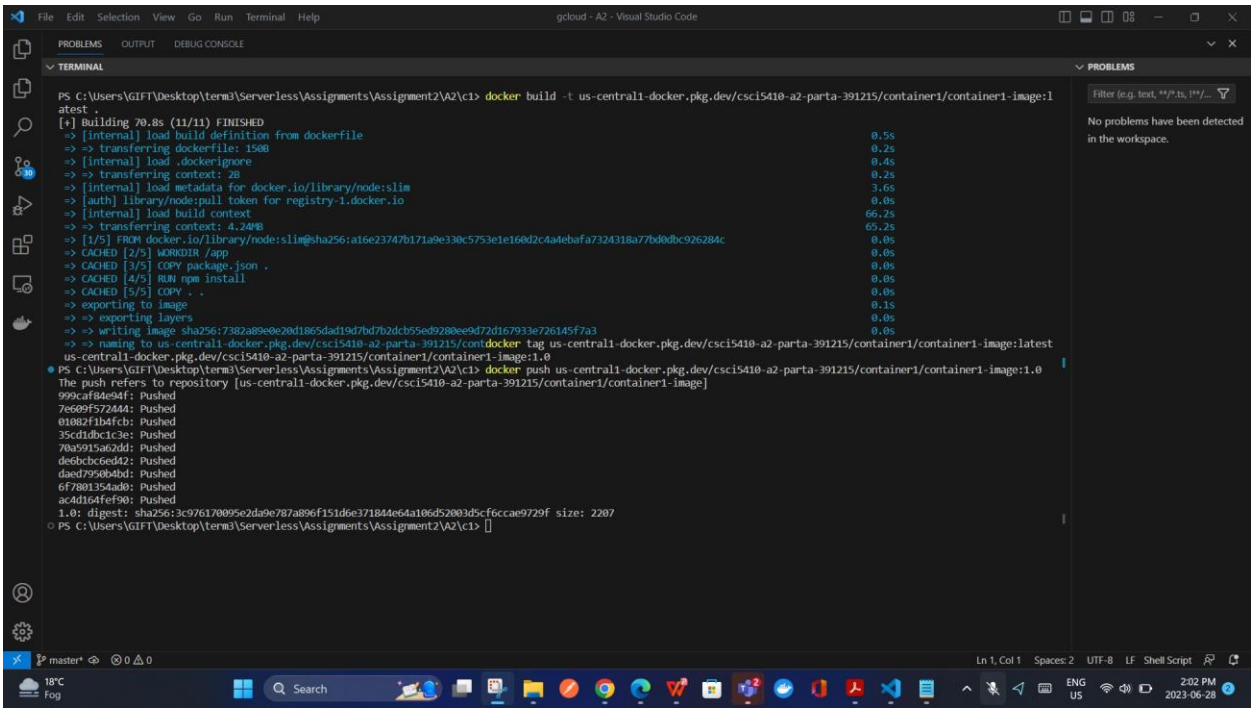

Figure 13 built docker image named **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container1/container1-image** for Registration react based application

Figure 14 defines the image (us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container1/container1-image) successfully created for registration react based application on docker dashboard.

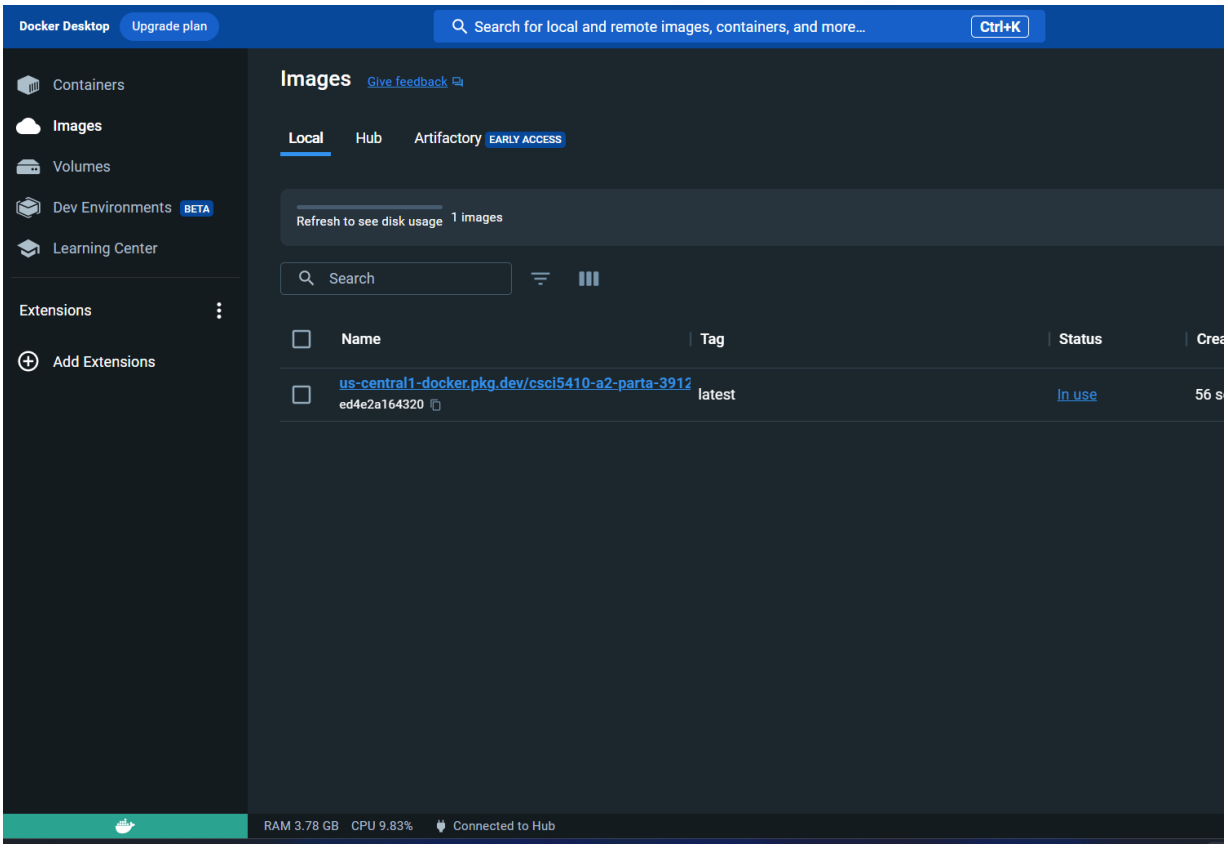

Figure 14 image **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container1/container1-image** successfully created on docker [1]

Figure 15 shows that image is running on container (serene_murdock).

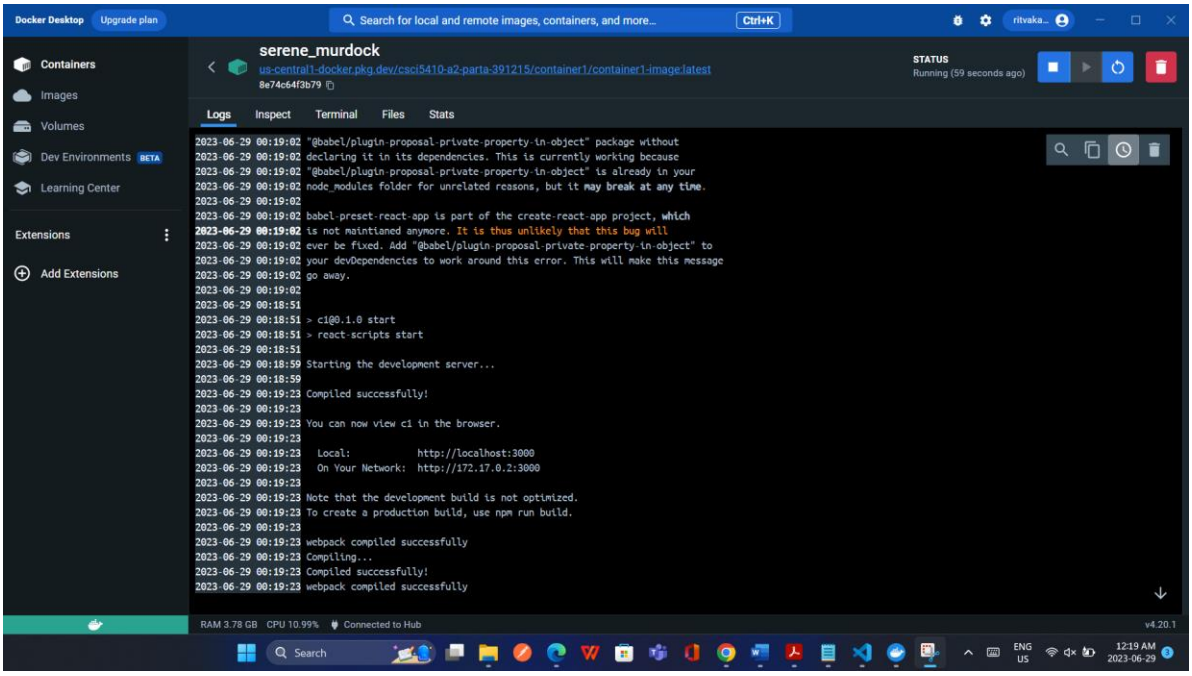

Figure 15 image **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container1/container1-image** is successfully running in container**(serene_murdock)** [1]

Container 2 (Login react based application)

An application built using Docker and React has been created for user login functionality. This application accepts login details (email and password) from the frontend and validates them against a Firebase Firestore database named "csci5410-A2-part2." If the login details are successfully validated, the user's status in the Firestore database's "State" collection will be updated from offline to online.

Figure 16 shows image of registration react based application is built successfully.

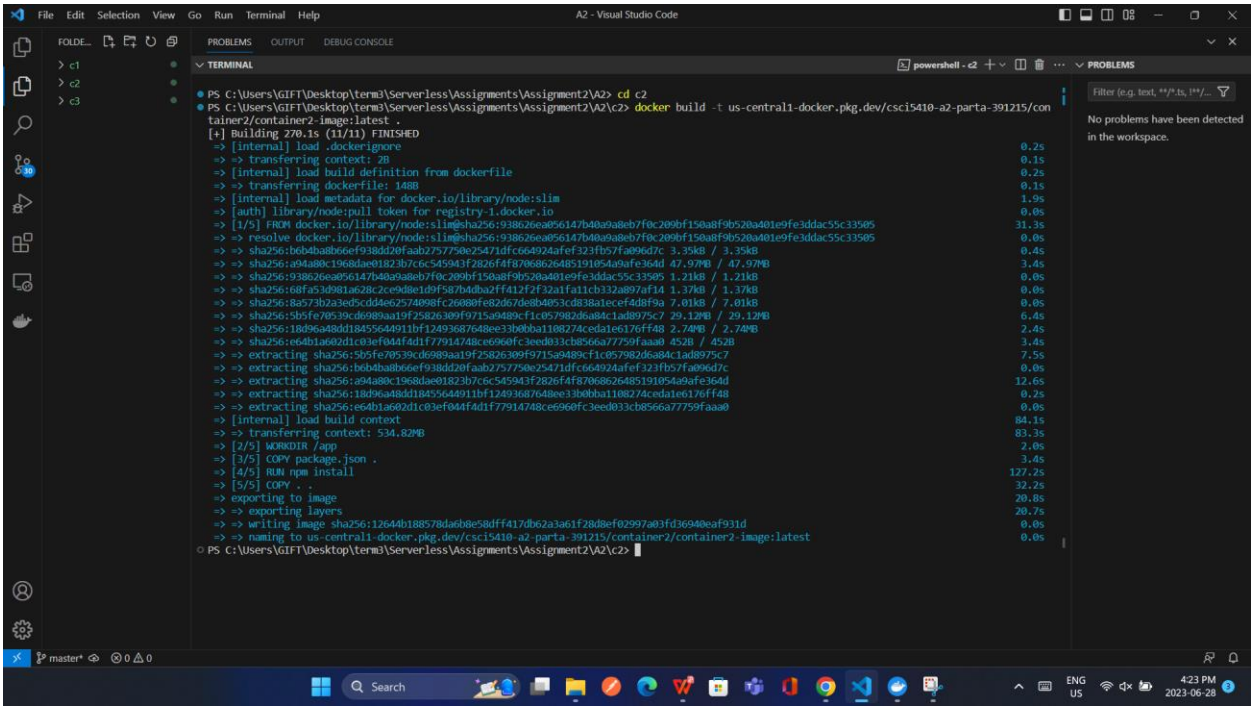

Figure 16 built docker image named **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container2/container2-image** for Login react based application

Figure 17 defines the image (us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container2/container2-image) created for registration react based application on docker dashboard.

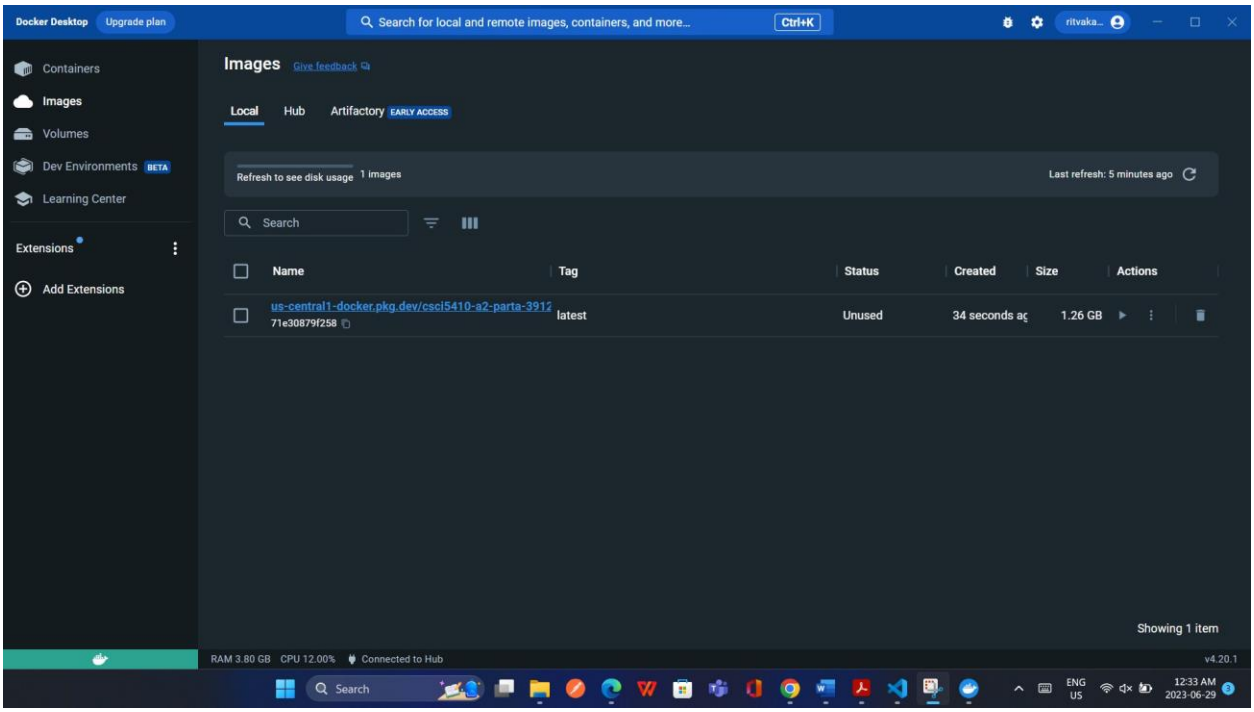

Figure 17 Image **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container2/container2-image** successfully created on docker [1]

Figure 18 shows that image is running on container (sharp_swirles).

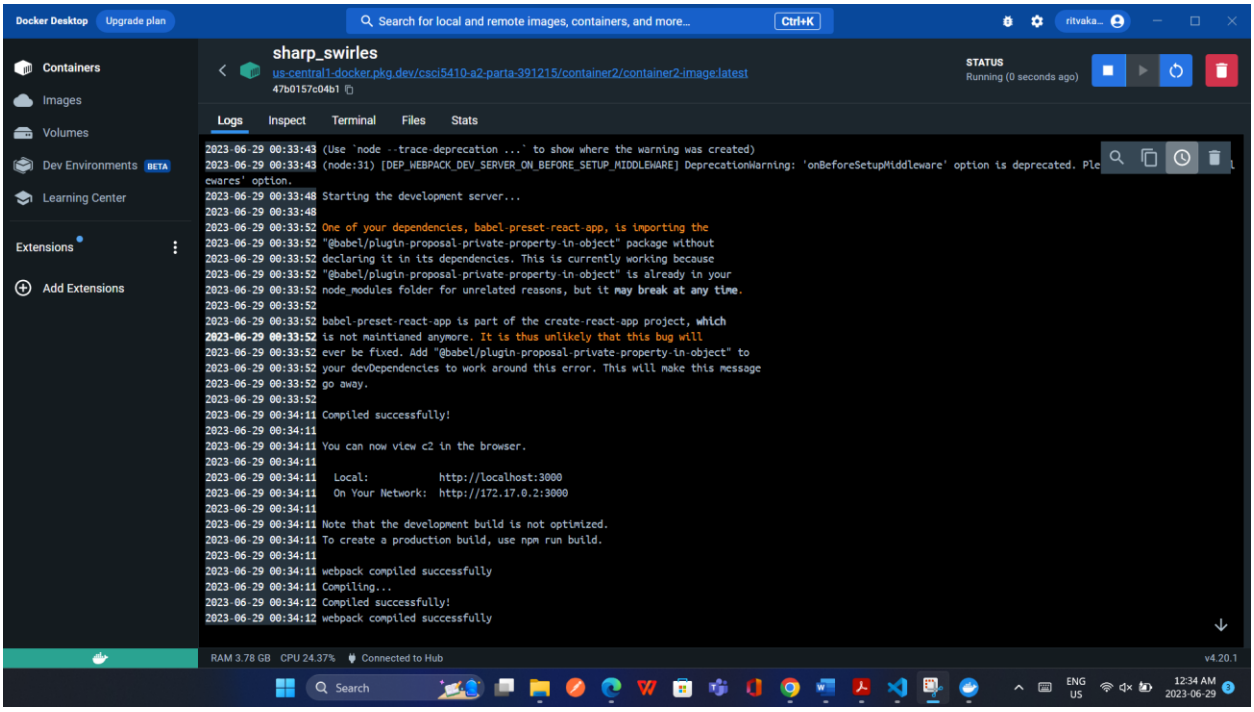

Figure 18 Image **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container1/container1-image** is successfully running in container**(sharp_swirles)** [1]

Container 3 (Profile react based application)

Developed an application using Docker and React that provides user profile functionality. The main reason behind developing of this application is to retrieve the state information from a database, specifically identifying which users are currently online. The application ensures that user sessions persist from login until logout. When a user clicks on the logout button, their session expires, and the application updates the corresponding state item in the Firestore database. Furthermore, the application also displays the names of other users who are currently logged in. This feature allows users to see a list of online users in addition to their own profile information.

Figure 19 shows image of profile react based application is built successfully.

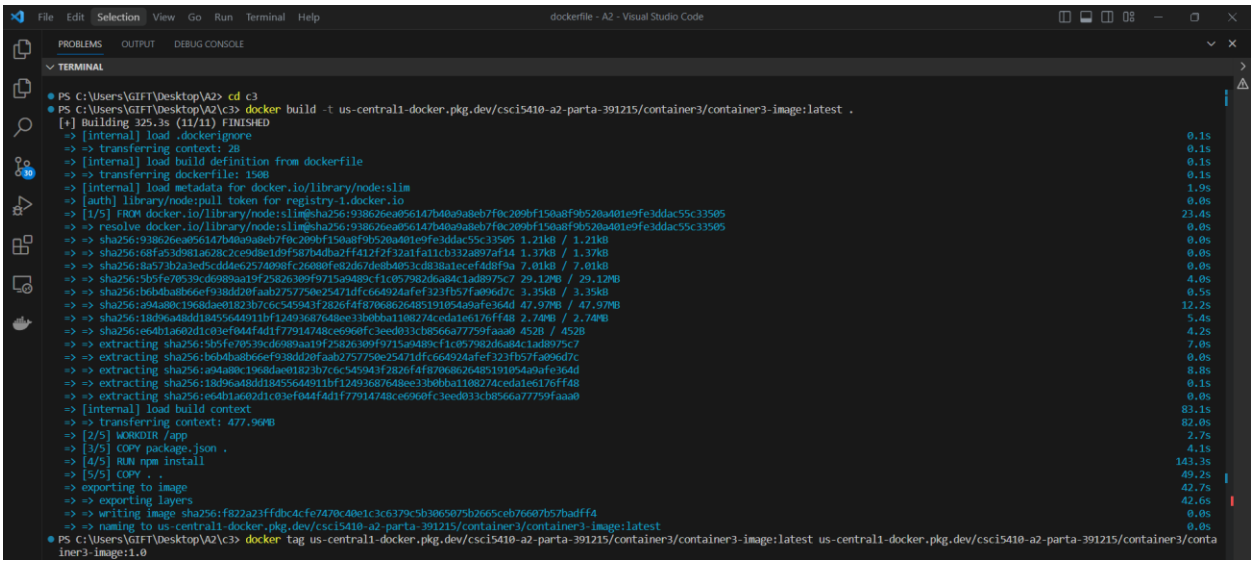

Figure 19 built docker image named **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container3/container3-image** for Profile react based application

Figure 20 defines the image (us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container3/container3-image) created for registration react based application on docker dashboard.

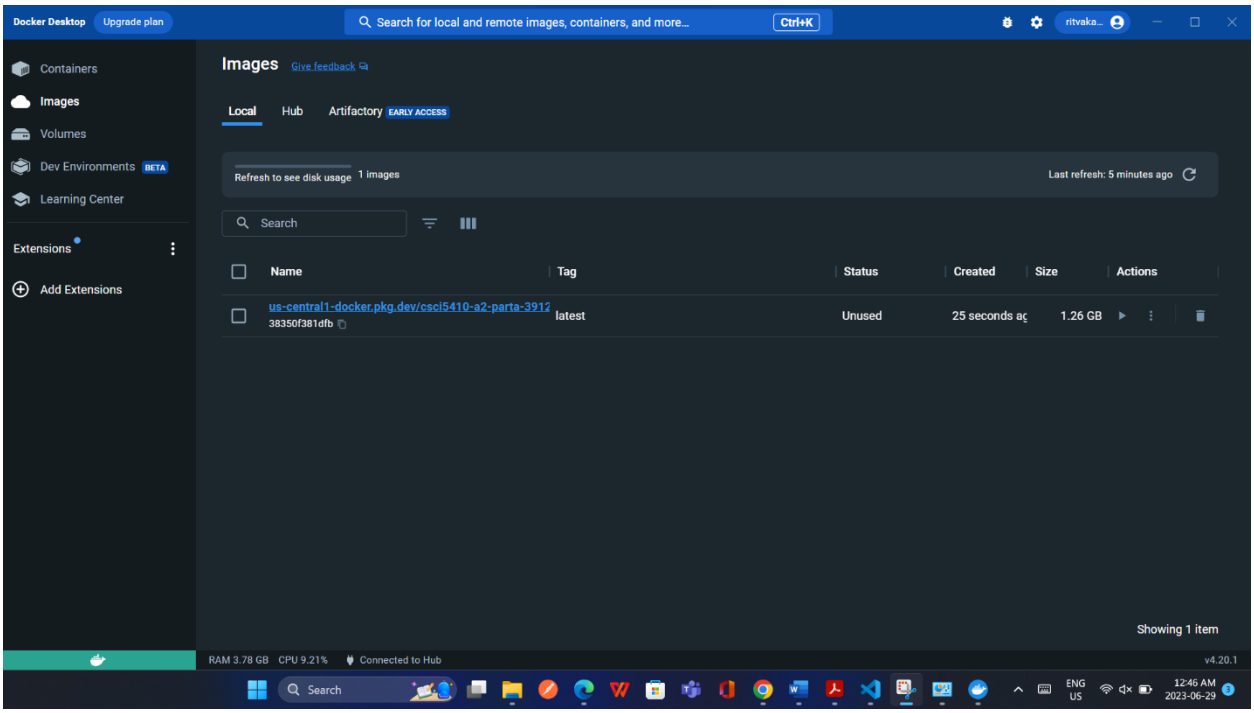

Figure 20 Image **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container3/container3-image** successfully created on docker [1]

Figure 21 shows that image is running on container (crazy_goldberg).

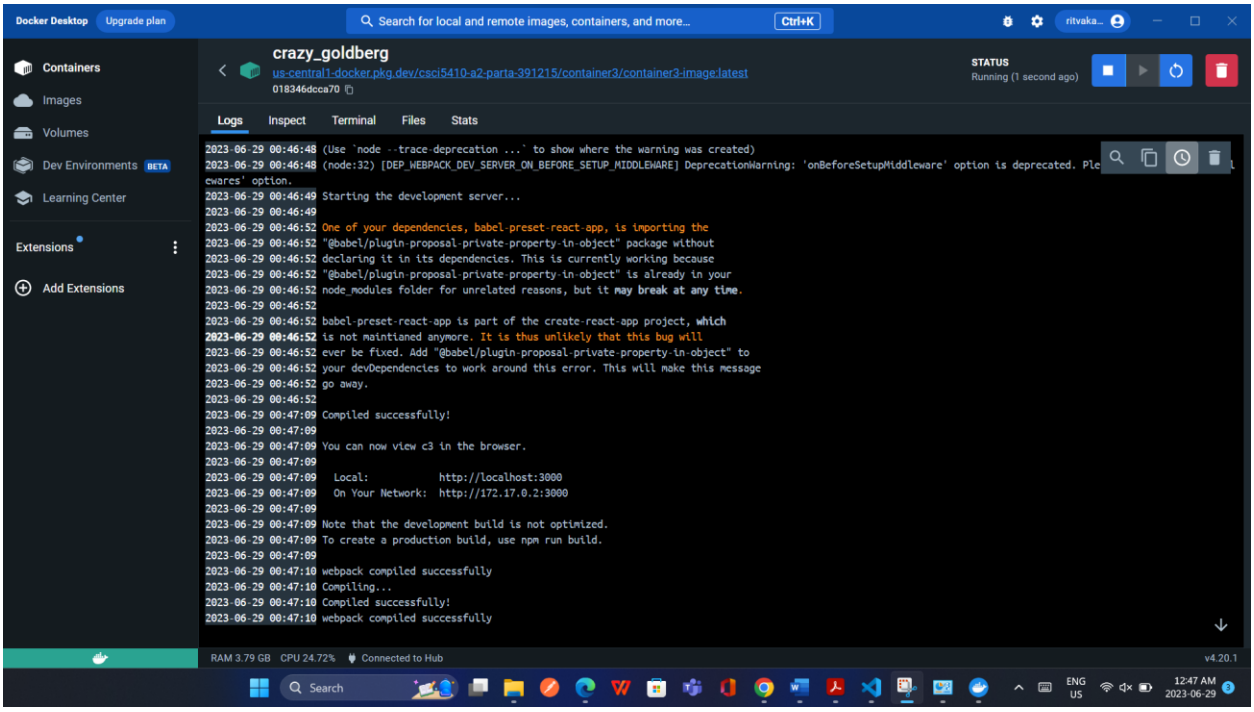

Figure 21 **Image us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container3/container3-image** is successfully running in container**(crazy_goldberg)** [1]

Website design with firebase firestore

Figure 22 defines the Registration page is developed using react technology.

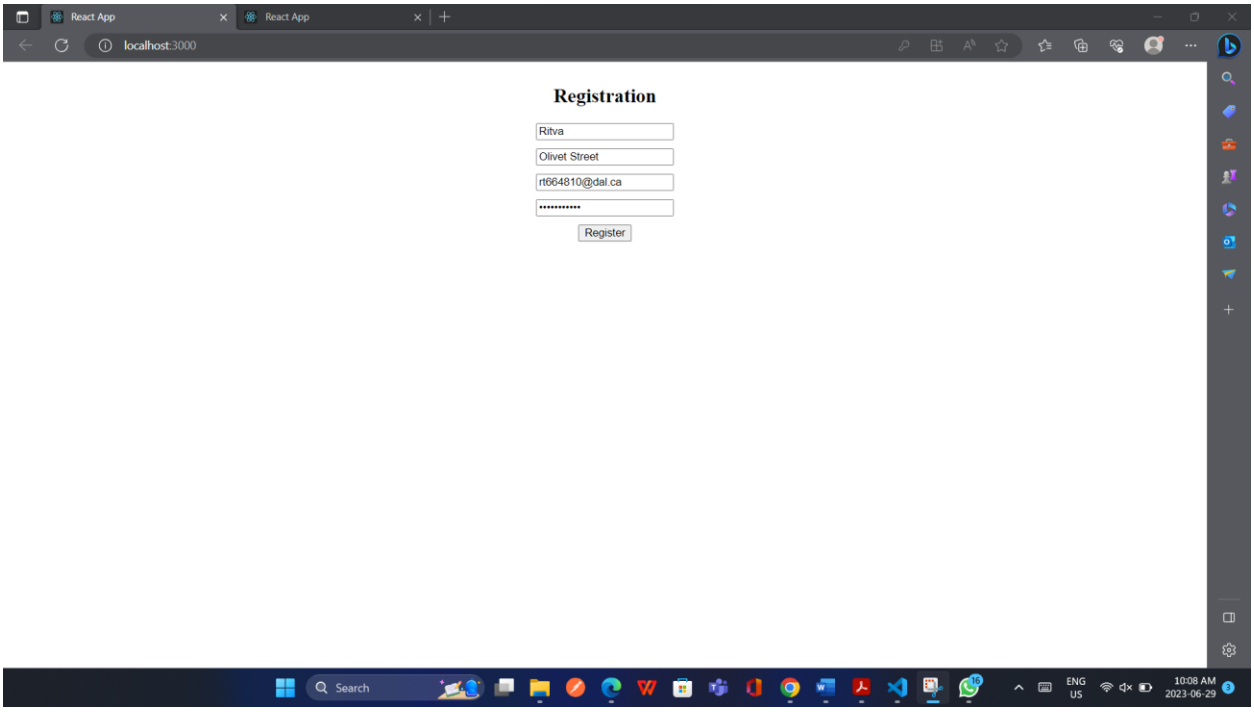

Figure 22 Registration react based application

Figure 23 depicts a successful user registration process where the user fills in their name, location, email, and password details.

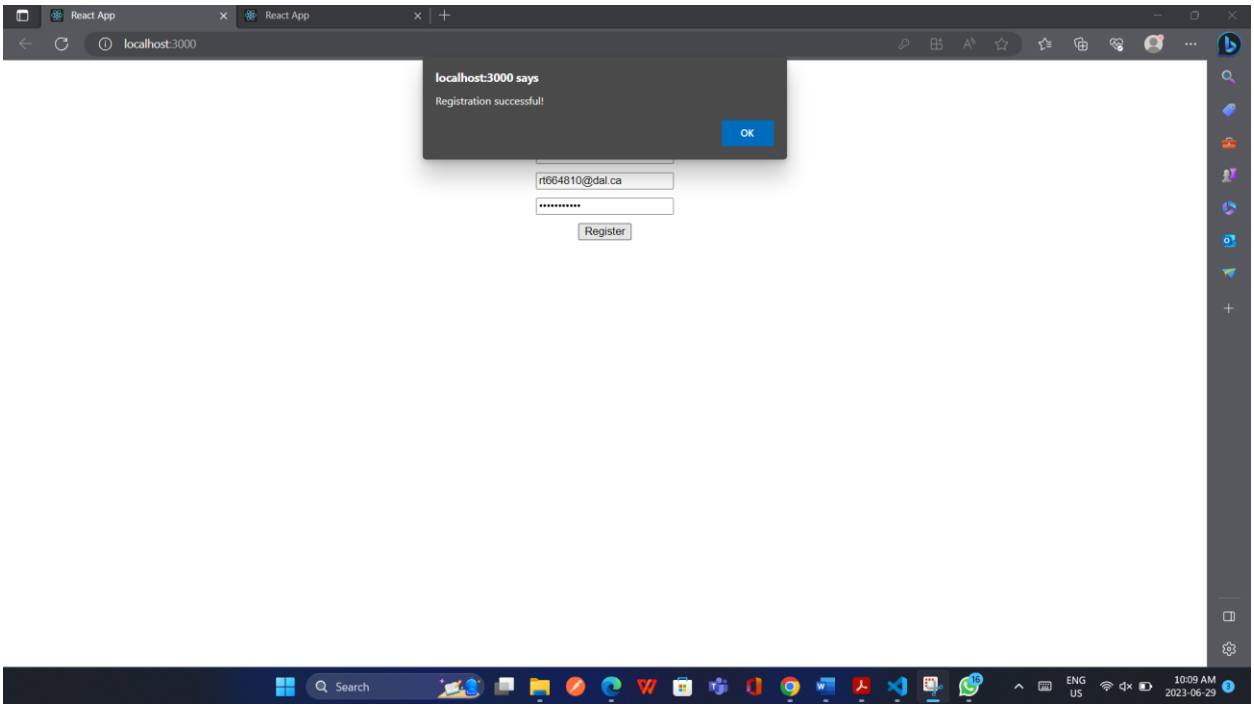

Figure 23 User has successfully registered

In Figure 24, the Firebase Firestore database "csci5410-A2-part2" consists of two collections: "Reg" and "state". The "Reg" collection contains all the user details, while the "state" collection stores user information along with additional fields such as timestamp and status, indicating whether the user is currently online or offline.

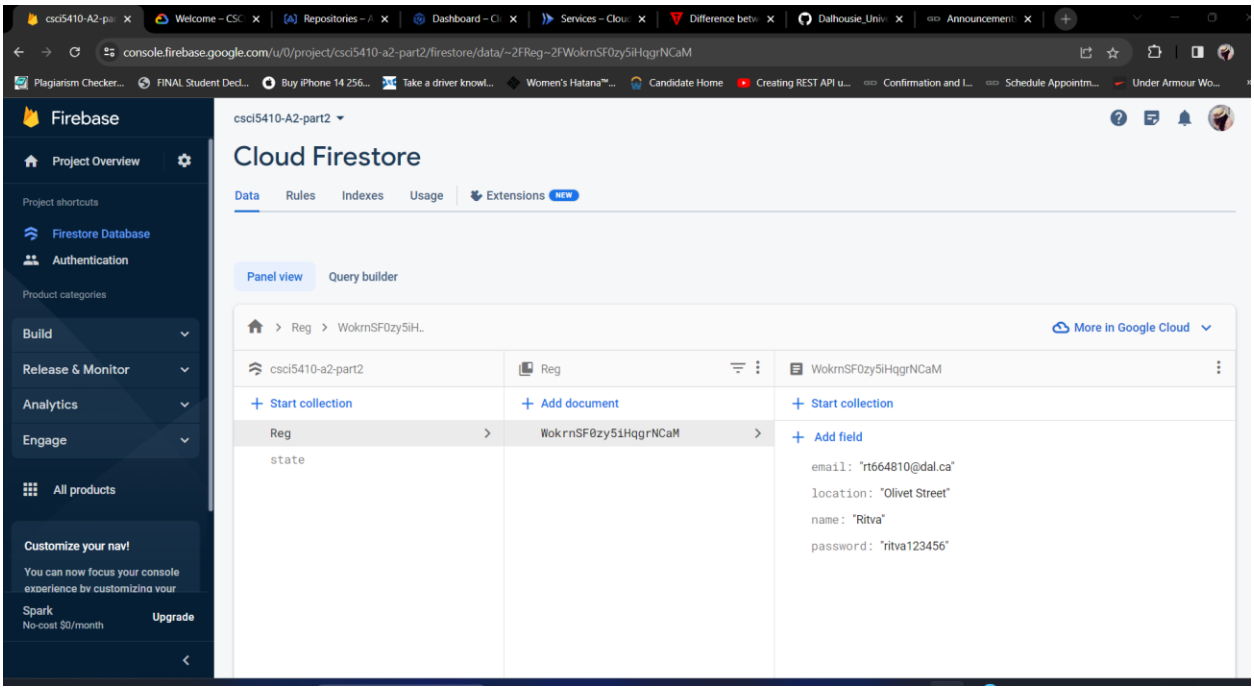

Figure 24 user details successfully stored in **Reg** collection of firestore firebase database named **csci5410-A2-part2** [2]

Figure 25 shows the state collection which stores the registered user details as well as time stamp and online or offline status. In below image, user Ritva has registered but user is not logged in right now. So, it shows status offline.

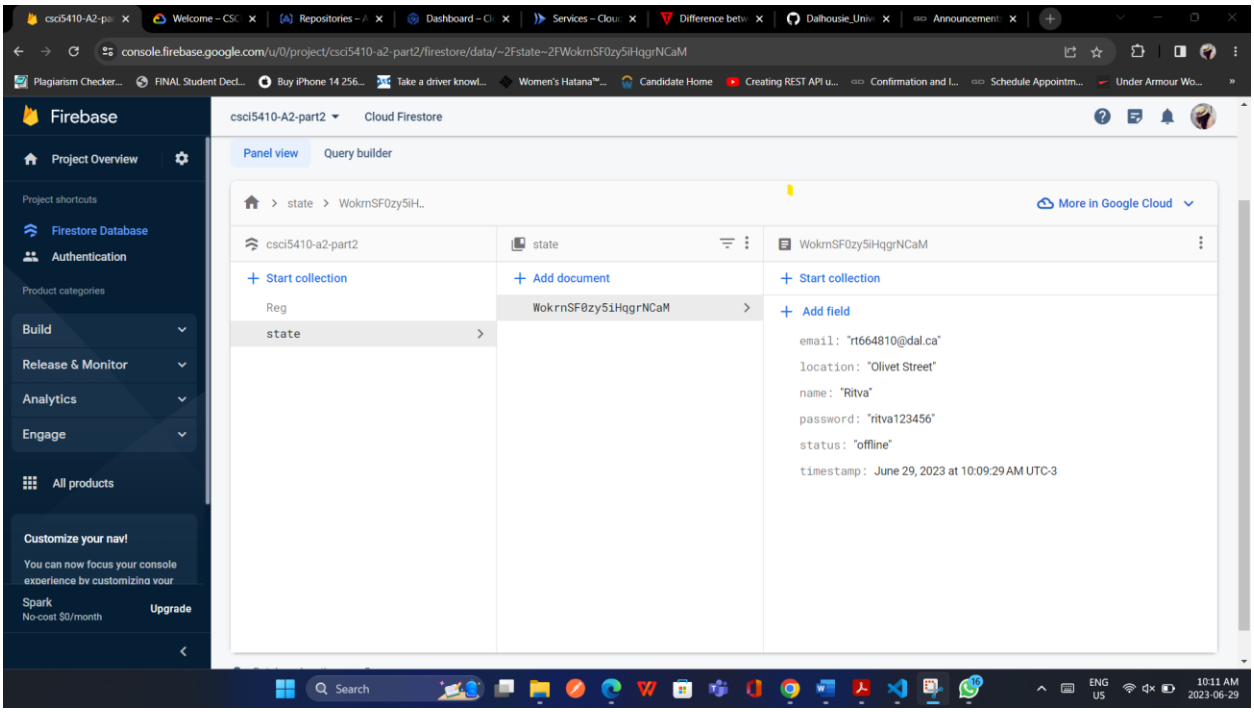

Figure 25 **state** collection of firestore firebase database named **csci5410-A2-part2** conatins registered user details as well as timestamp and online/offline status [2]

Figure 26 shows the Login page developed using react technology.

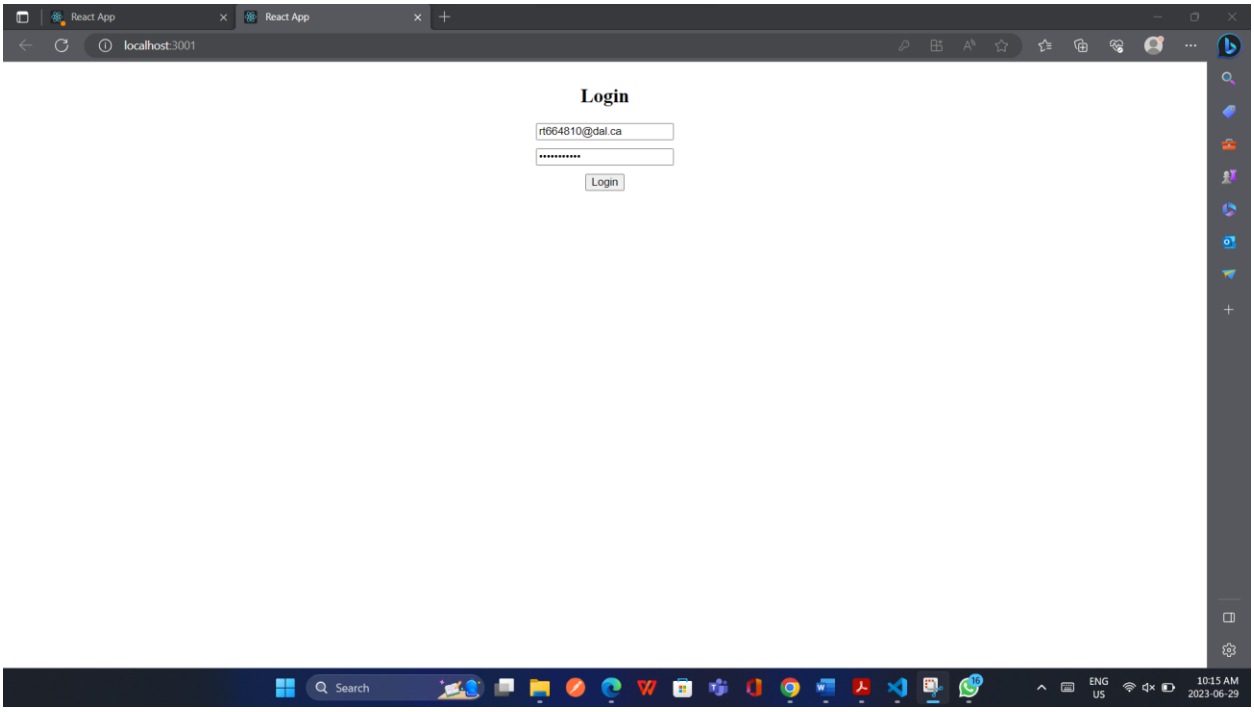

Figure 26 Login react based application

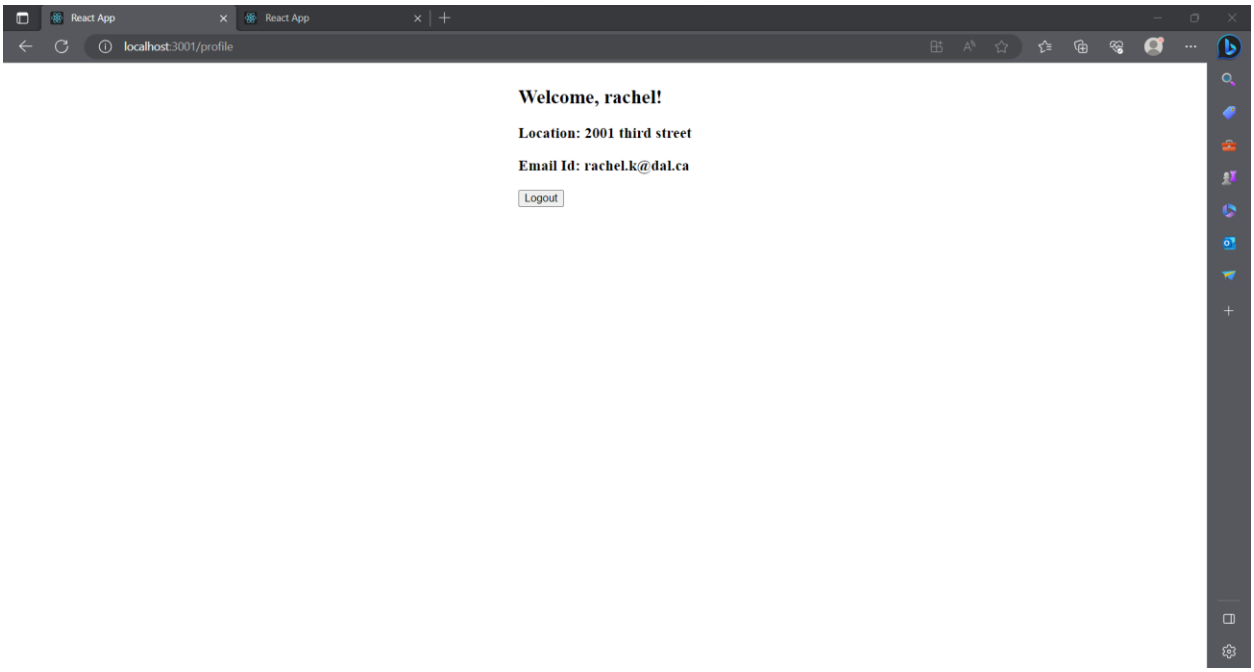

Figure 27 Email and password validated successfully

Figure 28 shows that user has logged in right now, his/her status of being offline is changed to **online** in state collection.

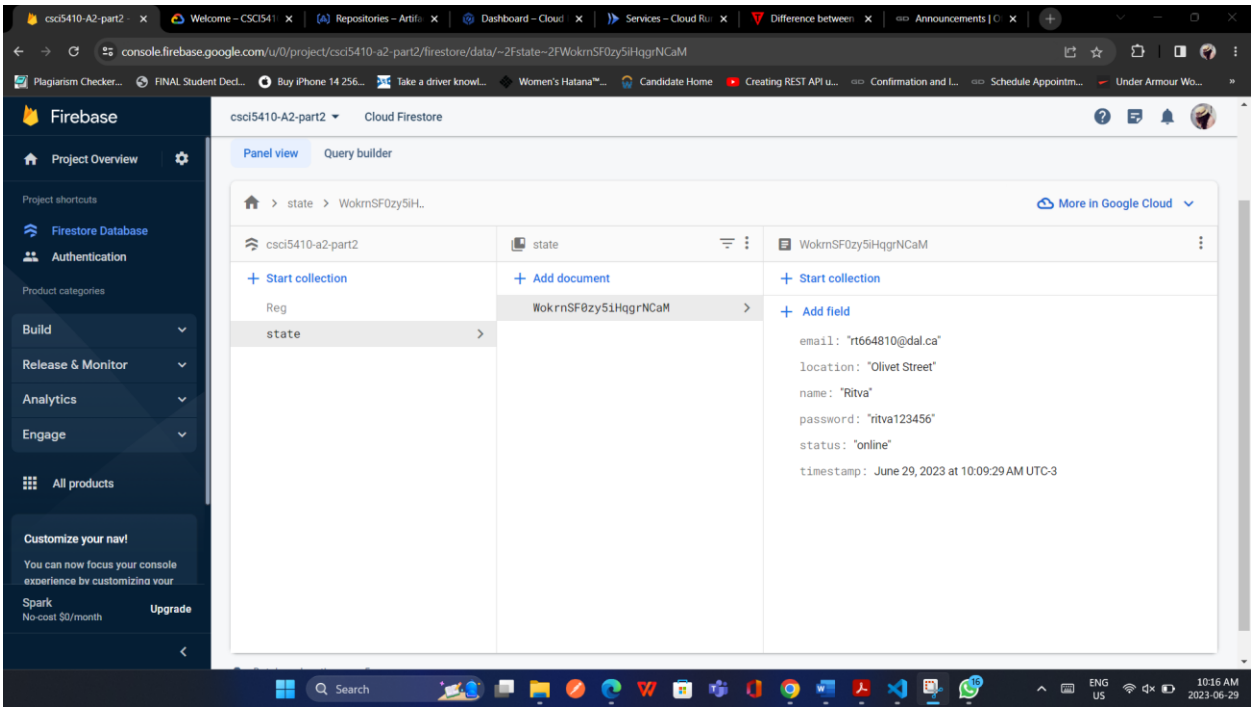

Figure 28 On successful validation of email and password, user status changed to **online** [2]

Figure 29 shows the Reg collection which has the details of all the registered users.

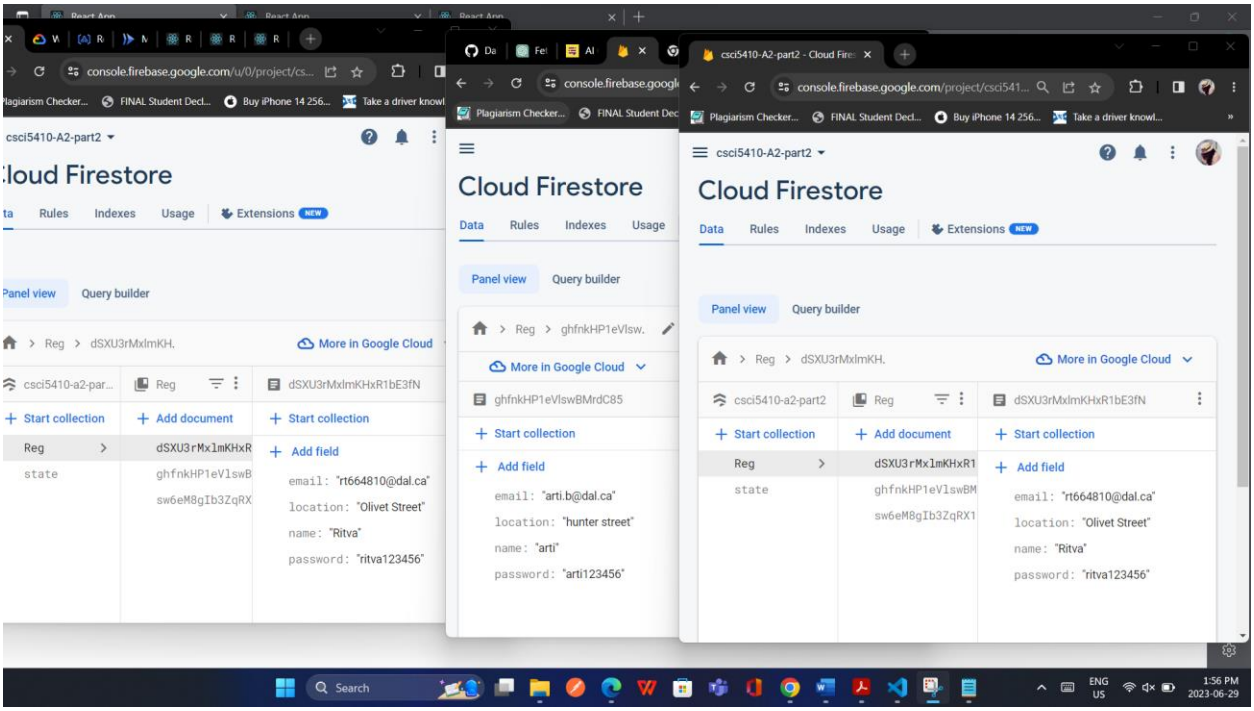

Figure 29 Details of All registered users (Reg collection) [2]

Figure 30 shows the state collection, and it shows that all the registered users have logged in right now (with online status).

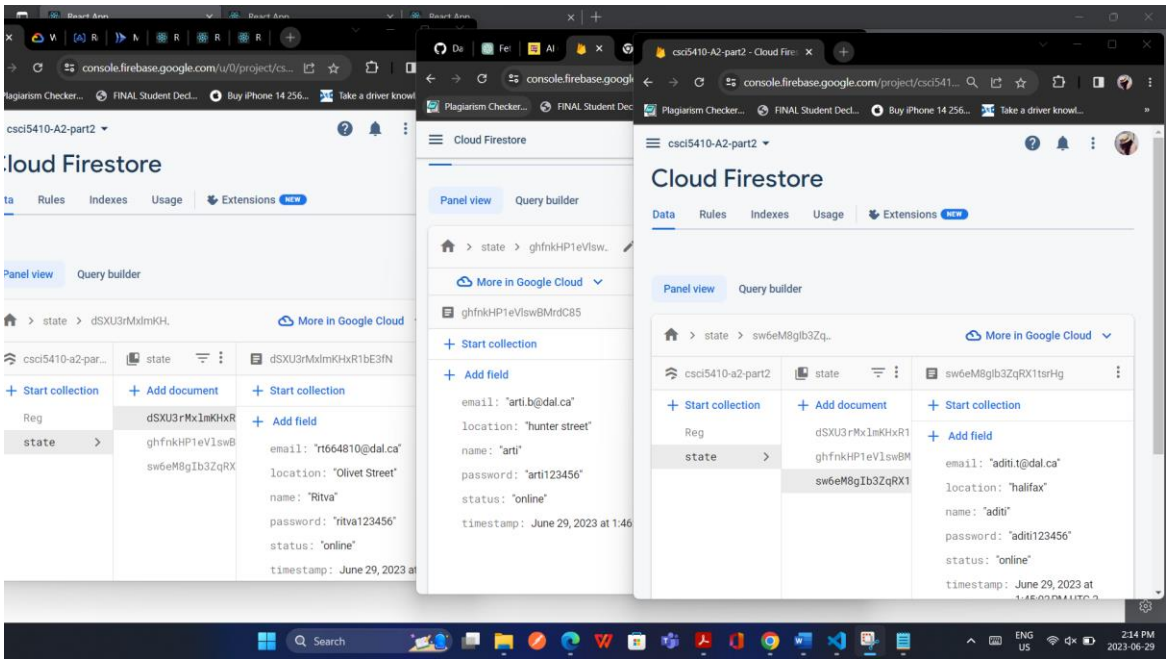

Figure 30 State collection with registered users' status [2]

Figure 31 shows the Profile page that has been developed using React technology. The below figure showcases that the user "Ritva" is currently logged in, and along with her own details, the page also displays the information of all the online users. This feature allows users to have visibility and access to the profiles of other users who are currently logged in.

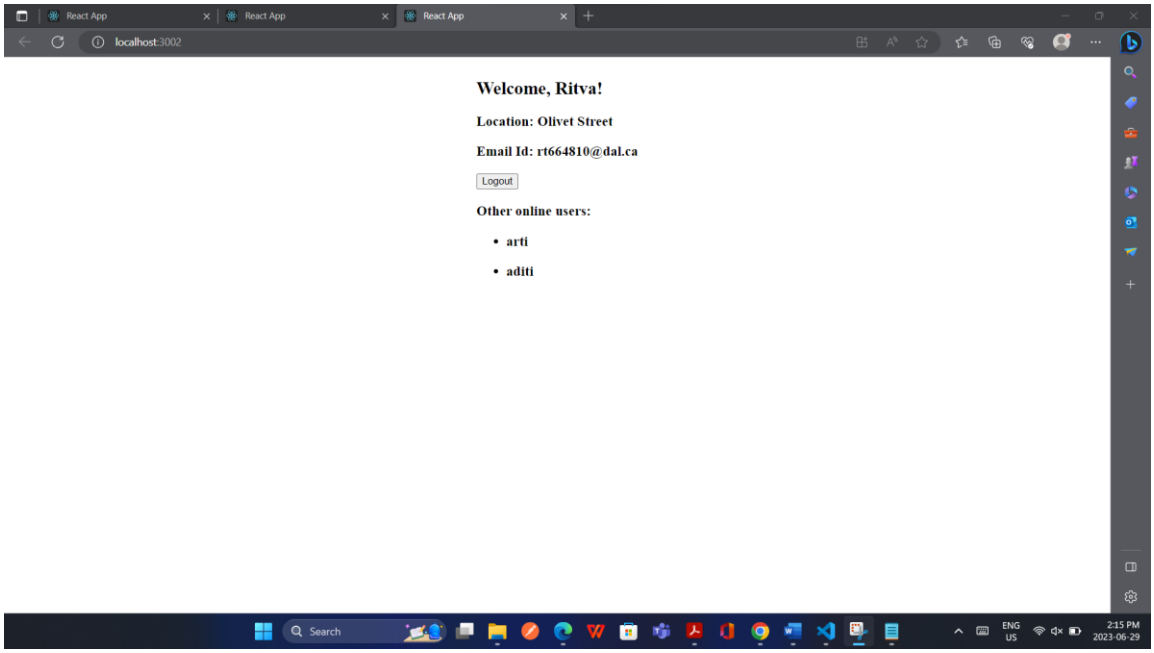

Figure 31 Profile page with all the online users

Project Creation on GCP

Figure 32 shows the creation of project named CSCI5410-A2-PartA on google cloud platform.

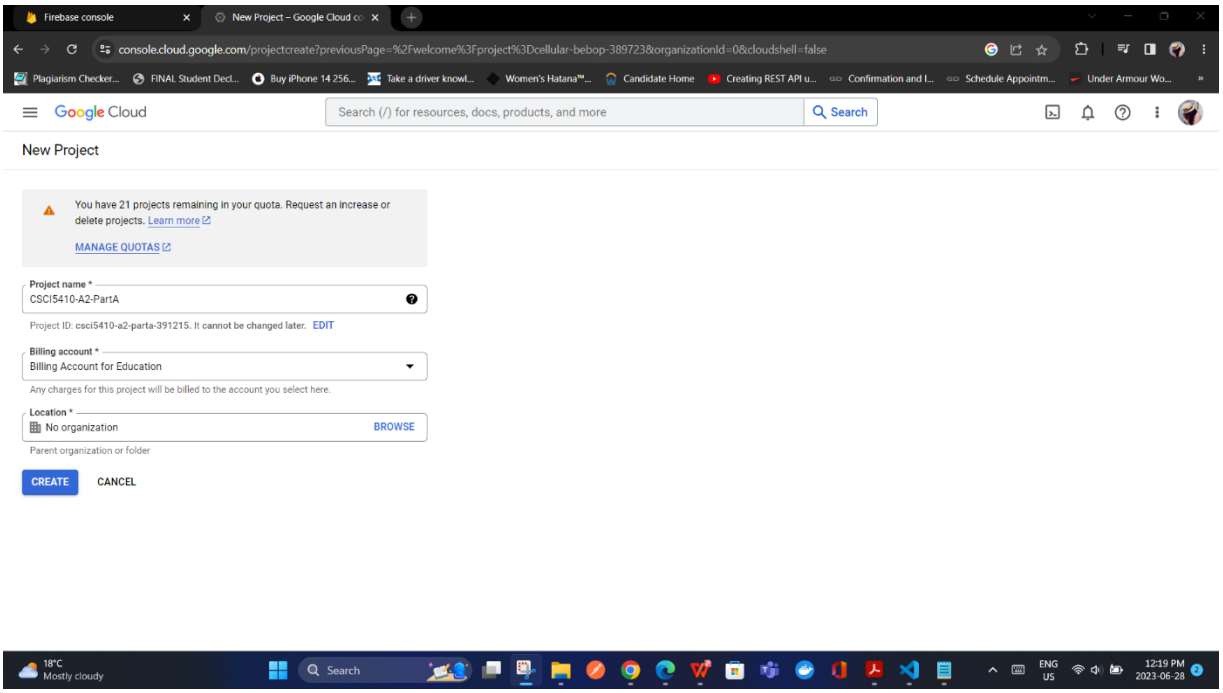

Figure 32 Project creation on GCP (CSCI5410-A2-PartA) [7]

Figure 33 shows the project named CSCI5410-A2-PartA successfully created on google cloud platform.

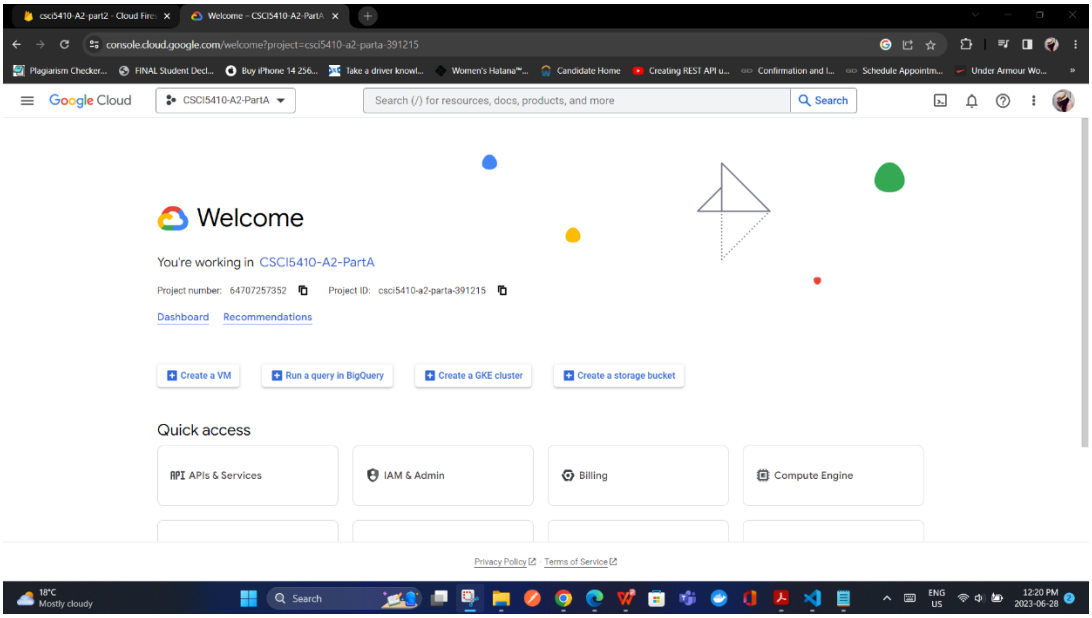

Figure 33 Project created successfully on GCP (CSCI5419-A2-PartA) [7]

Artifact Registry Setup

Figure 34 defines the Artifact registry on GCP and there is not any repositories.

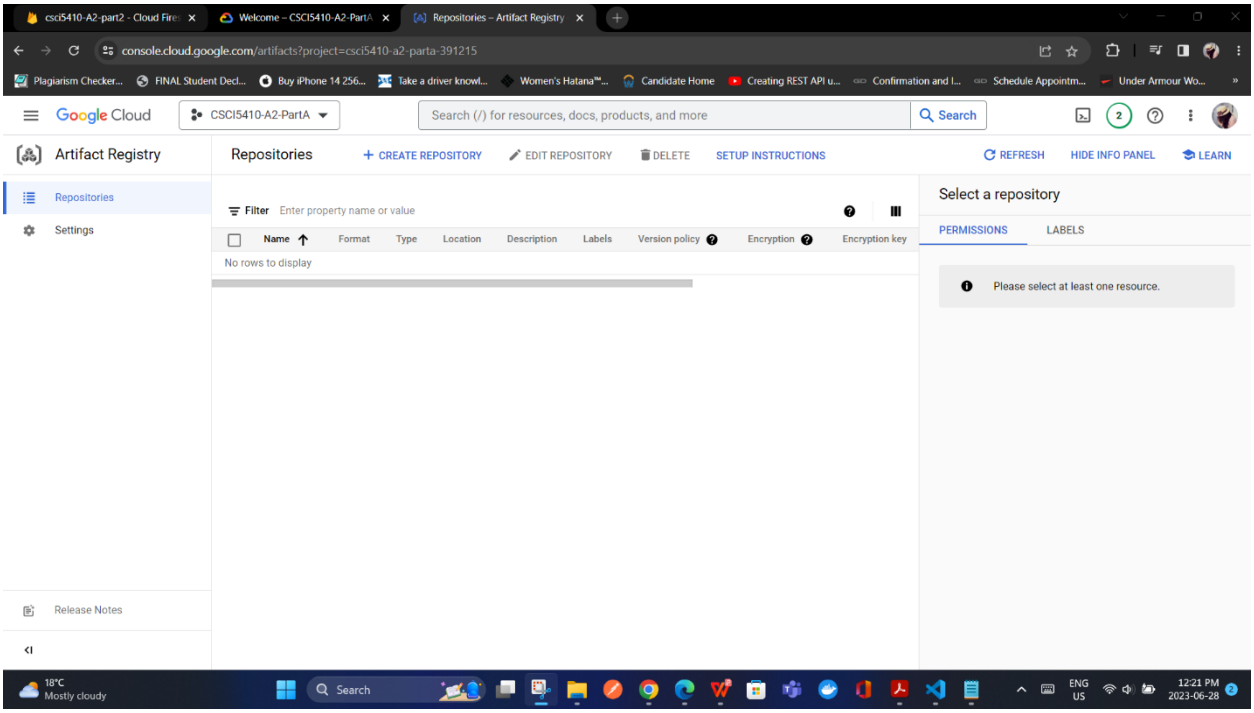

Figure 34 Artifact Registry of Google cloud platform [5]

Figures 35 and 36 show the steps required to create repository named conatiner1 in Artifact Registry.

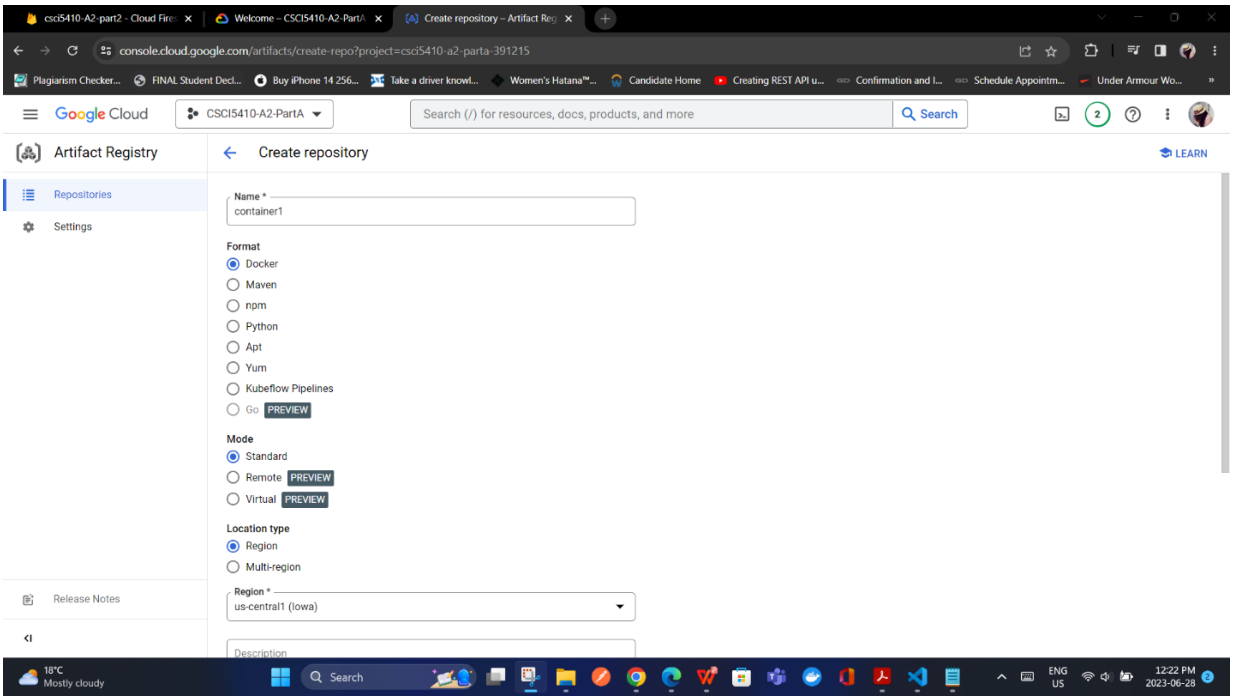

Figure 35 Creation of repository named container1 on Artifact Registry for *Dockerized* image for registration react application [5]

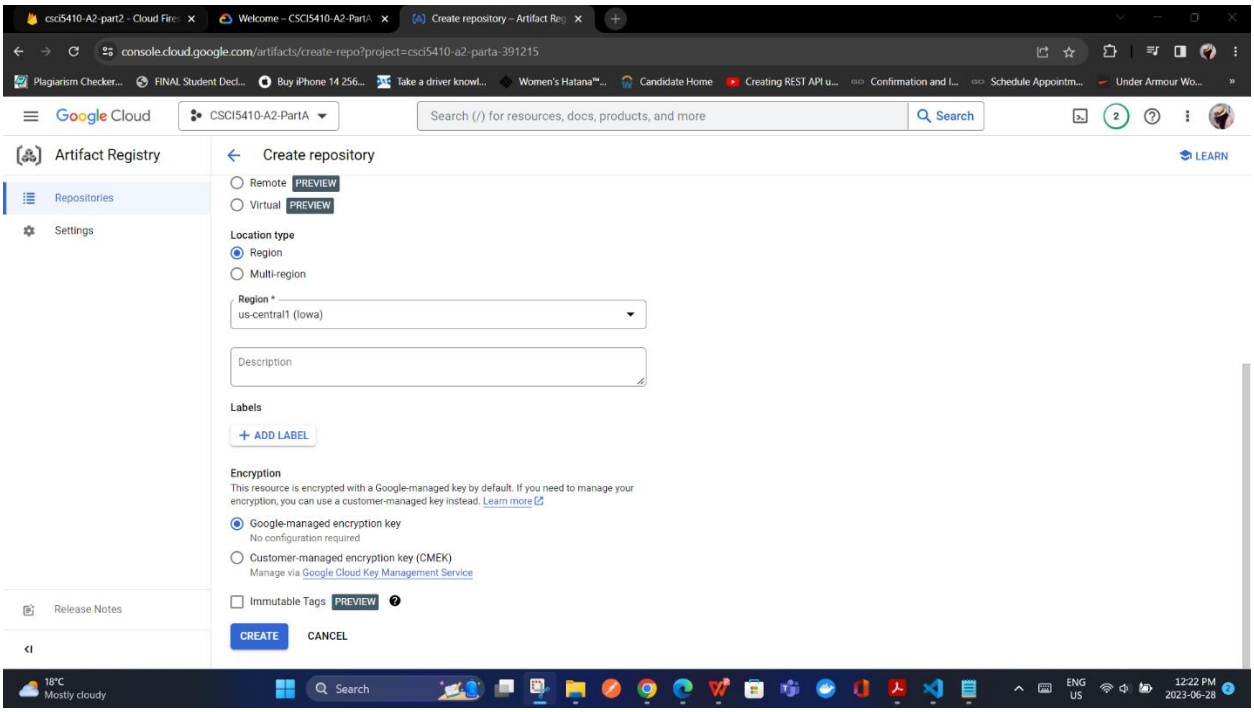

Figure 36 Creation of repository named container1 on Artifact Registry for Dockerized image for registration react application [5]

Figure 37 defines that conatiner1 repository created in Artifact Registry.

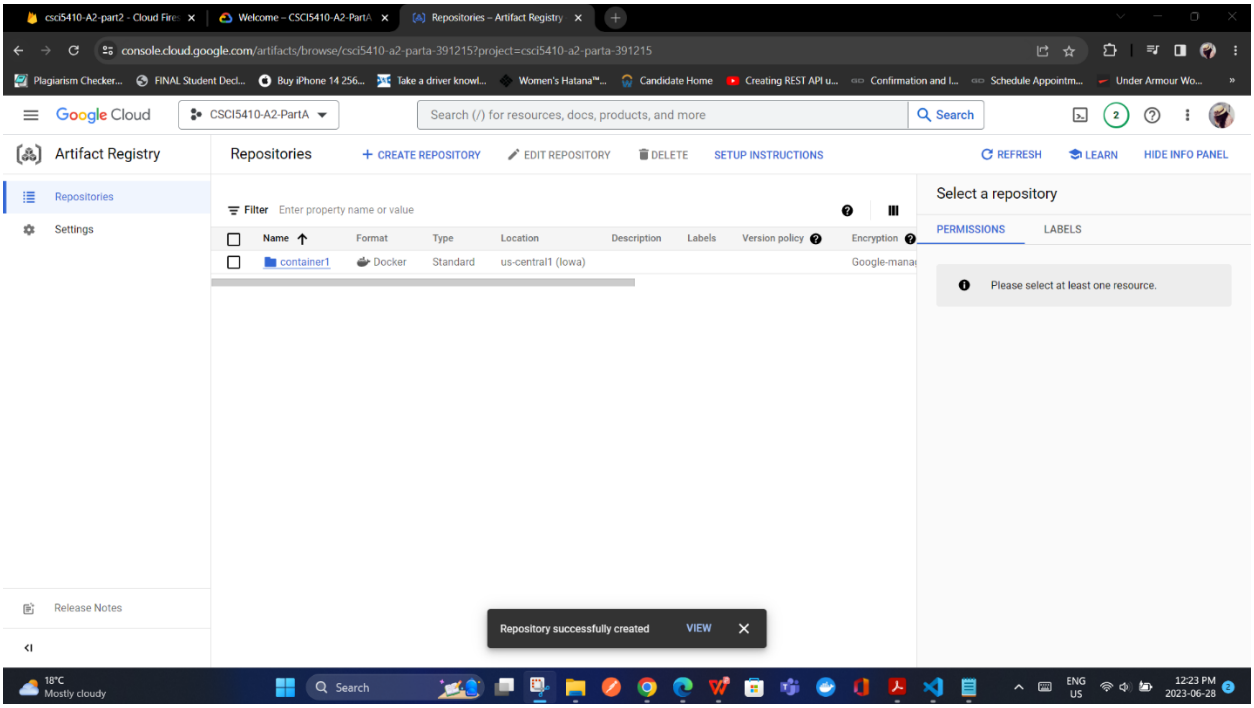

Figure 37 Successfully created registry named conatiner1 in Artifact Registry [5]

Figures 38 and 39 show the steps required to create repository named conatiner2 in Artifact Registry.

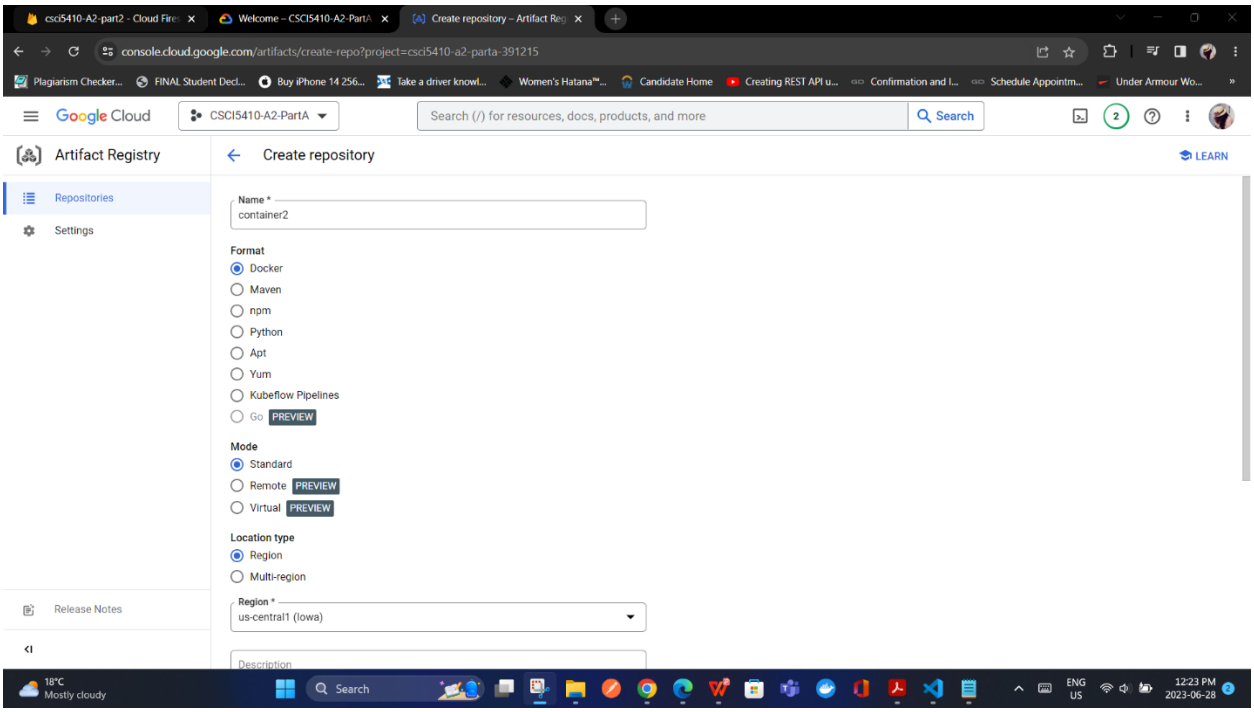

Figure 38 Creation of repository named container2 on Artifact Registry for Dockerized image for login react application [5]

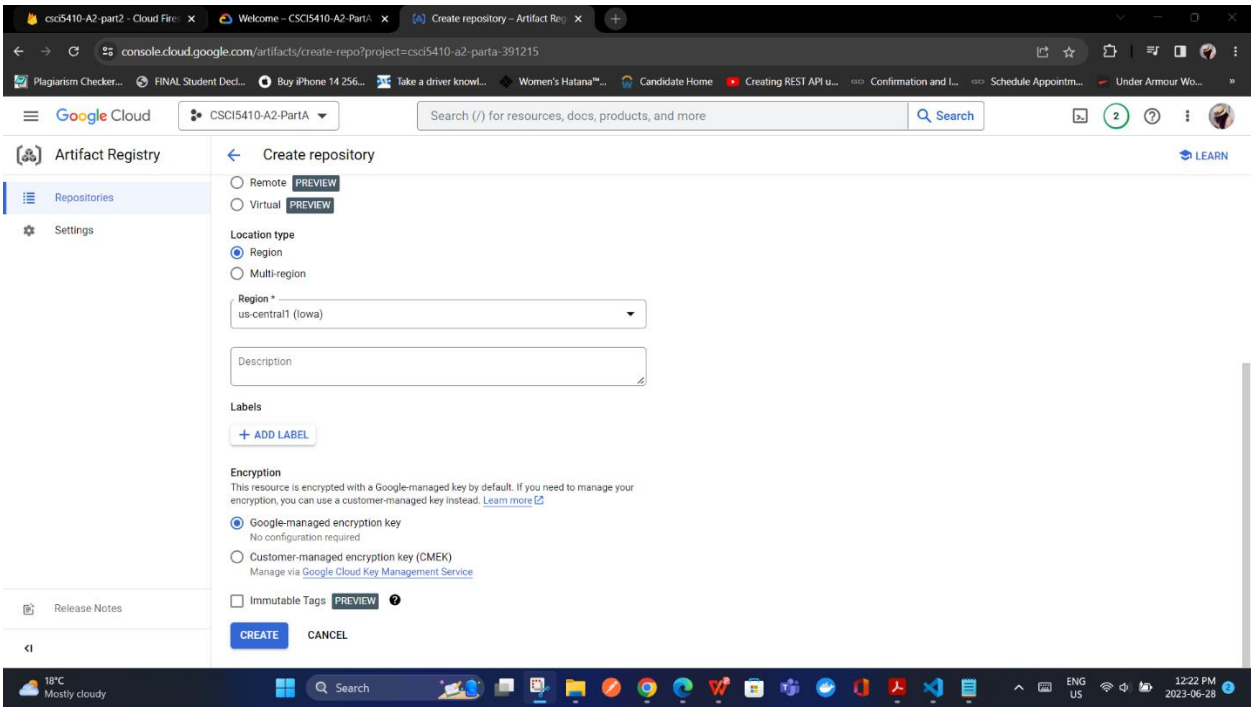

Figure 39 Creation of repository named container2 on Artifact Registry for Dockerized image for login react application [5]

Figure 40 defines that conatiner2 repository created in Artifact Registry.

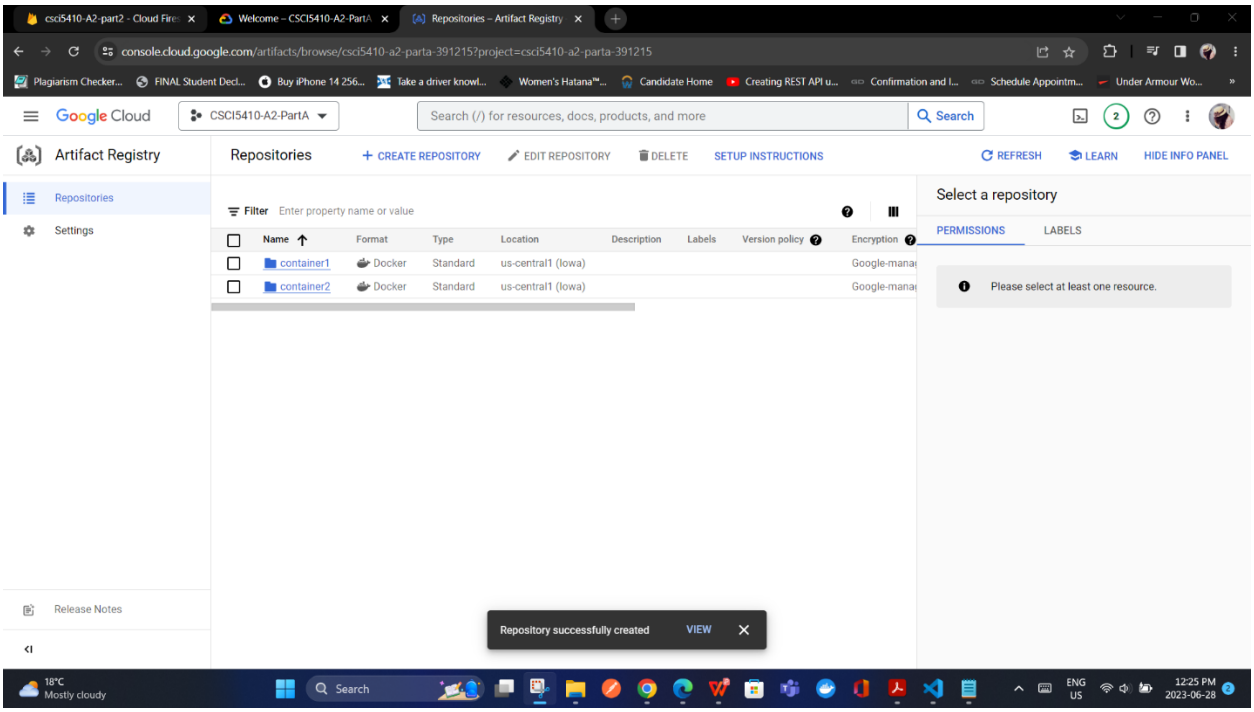

Figure 40 Successfully created registry named conatiner2 in Artifact Registry [5]

Figures 41 and 42 show the steps required to create repository named conatiner3 in Artifact Registry.

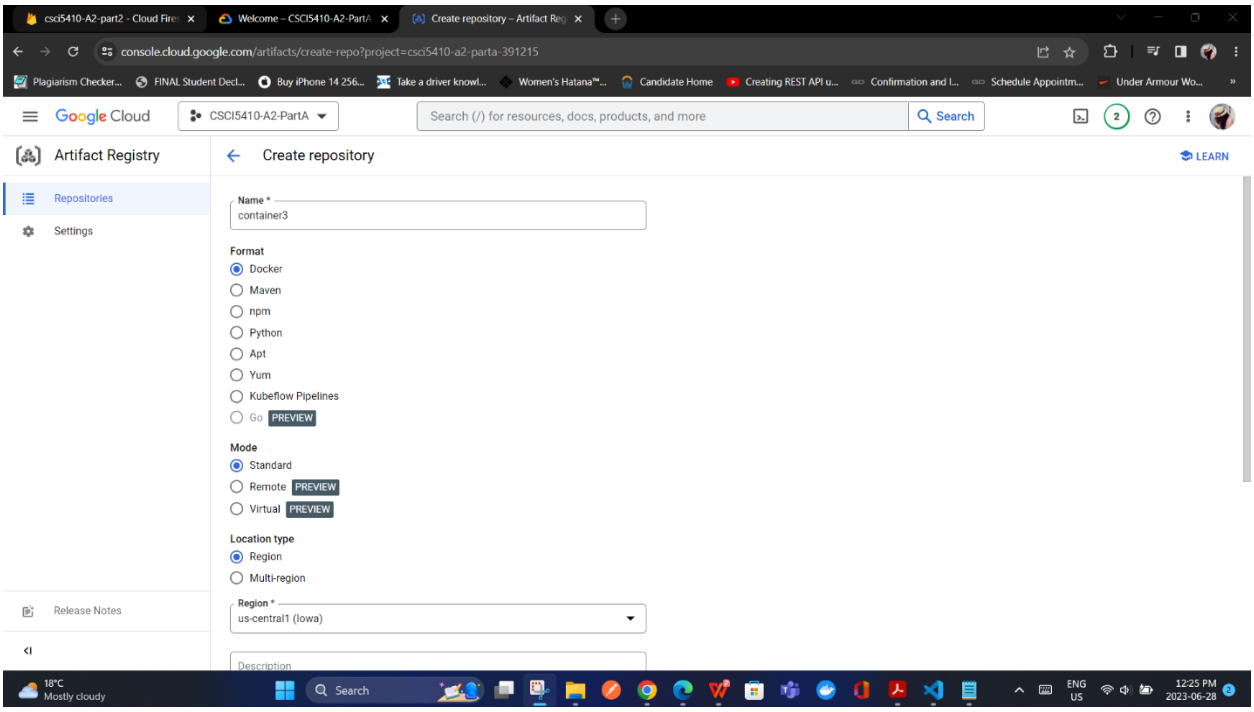

Figure 41 Creation of repository named container3 on Artifact Registry for Dockerized image for profile react application [5]

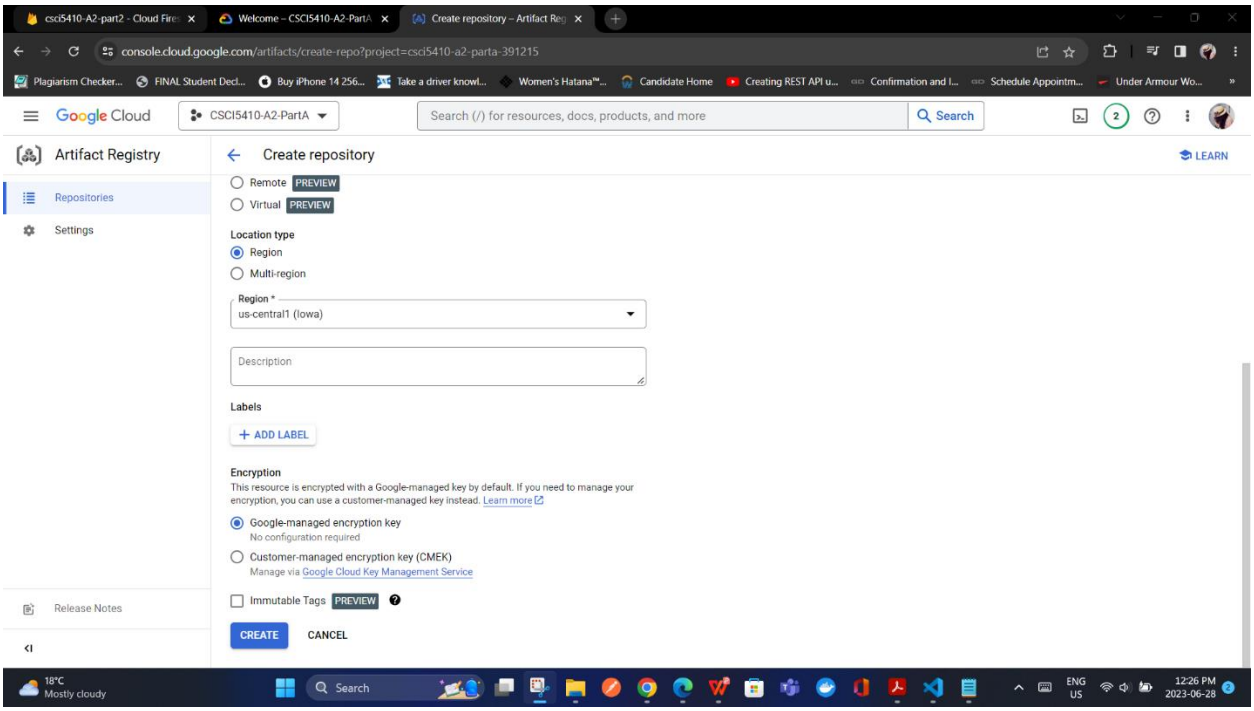

Figure 42 Creation of repository named container3 on Artifact Registry for Dockerized image for profile react application [5]

Figure 43 defines that conatiner3 repository created in Artifact Registry.

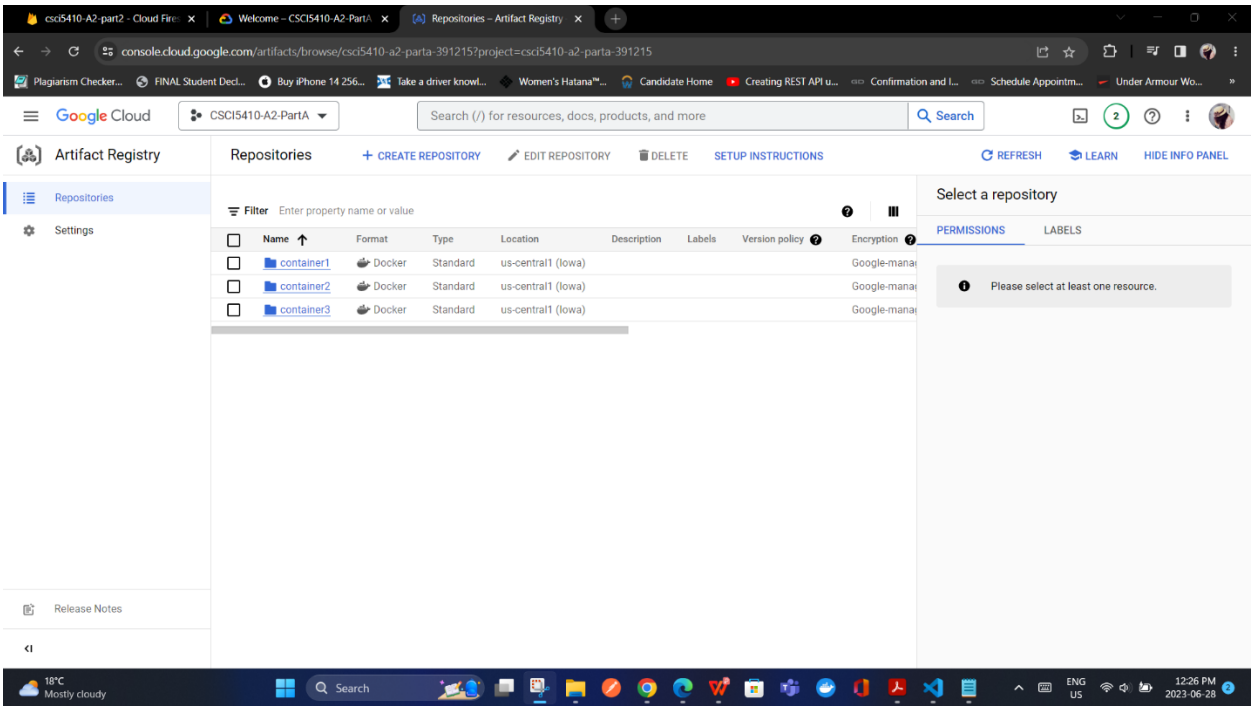

Figure 43 Successfully created registry named conatiner3 in Artifact Registry [5]

Figure 44 shows that there are not any pushed images in container1 repository.

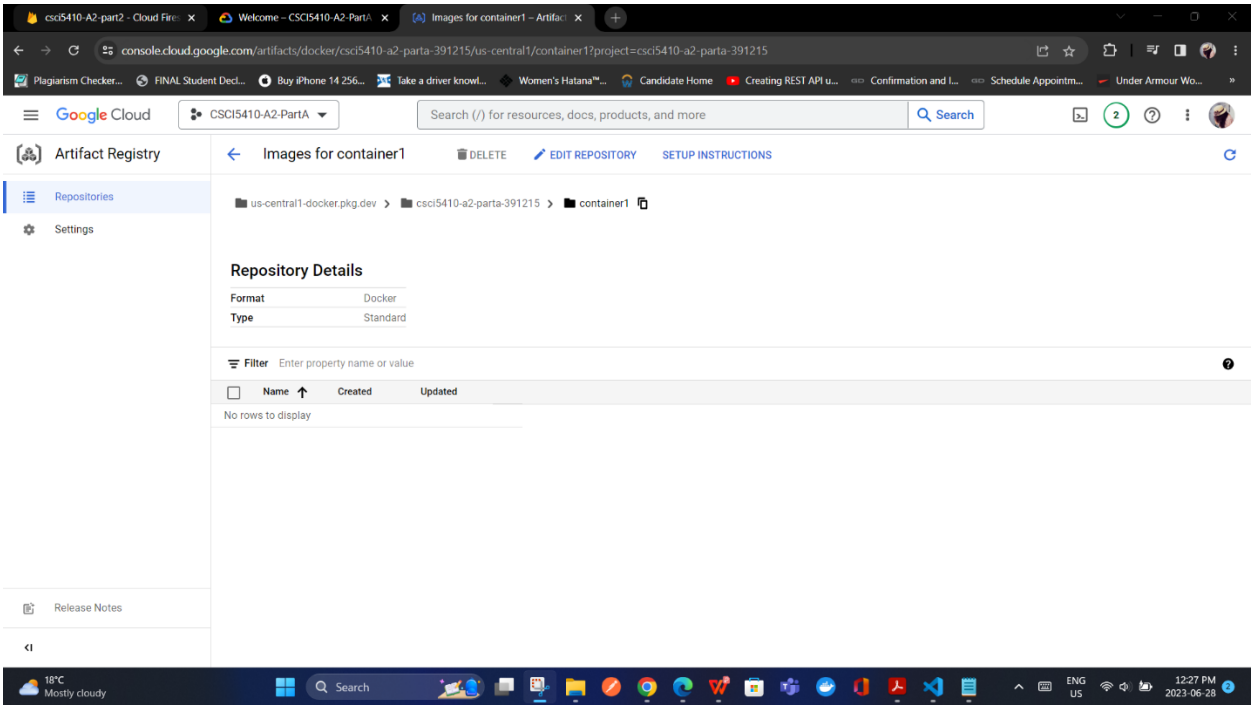

Figure 44 Empty repository container1[5]

Figure 45 shows that there are not any pushed images in container2 repository.

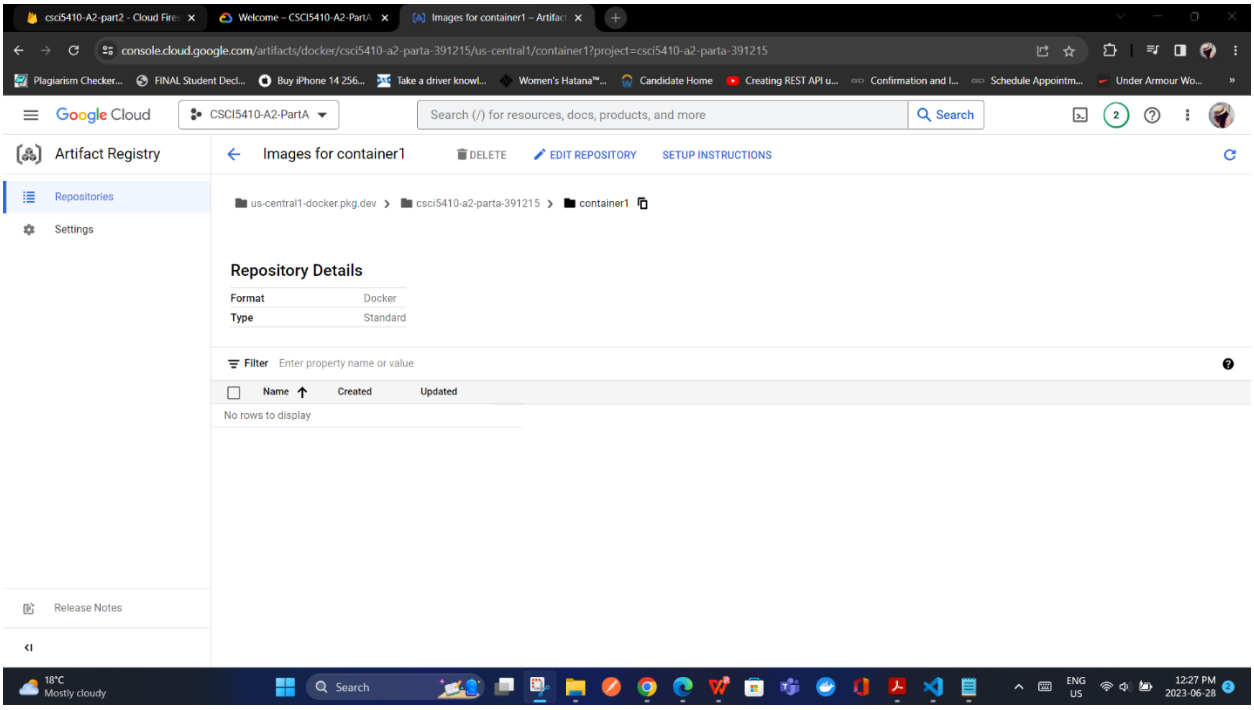

Figure 45 Empty repository container2 [5]

Figure 46 shows that there are not any pushed images in container2 repository.

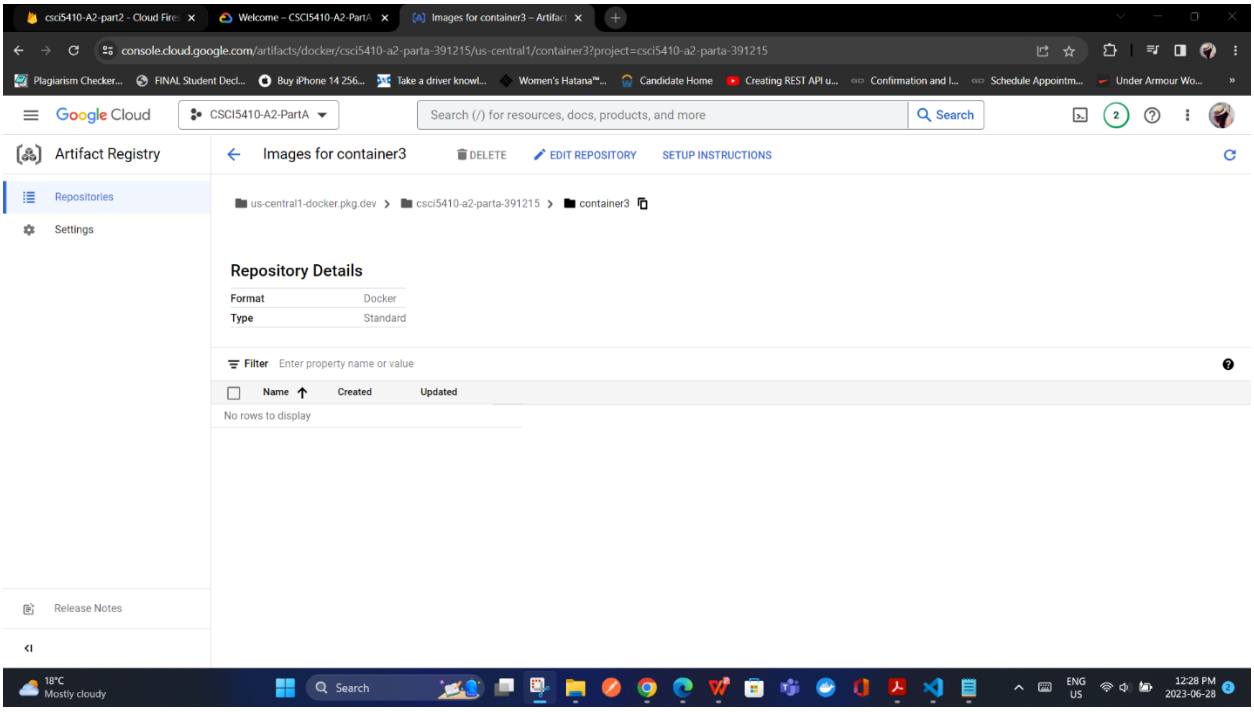

Figure 46 Empty repository container3 [5]

Figure 47 shows that docker image **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container1/container1-image** pushed to repository conatiner1 of google cloud artifact registry.

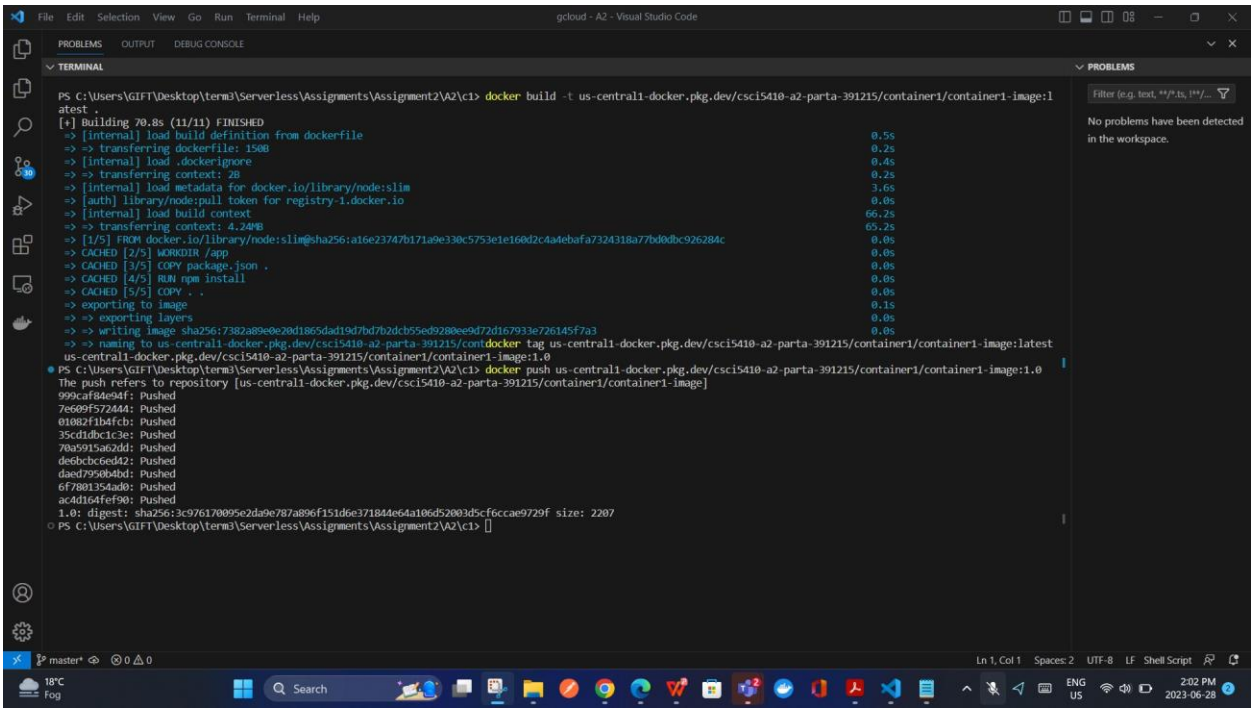

Figure 47 push Docker image namely **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container1/container1-image** to Google Artifact registry

Figure 48 shows that docker image **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container1/container1-image** successfully pushed to repository conatiner1 of google cloud artifact registry.

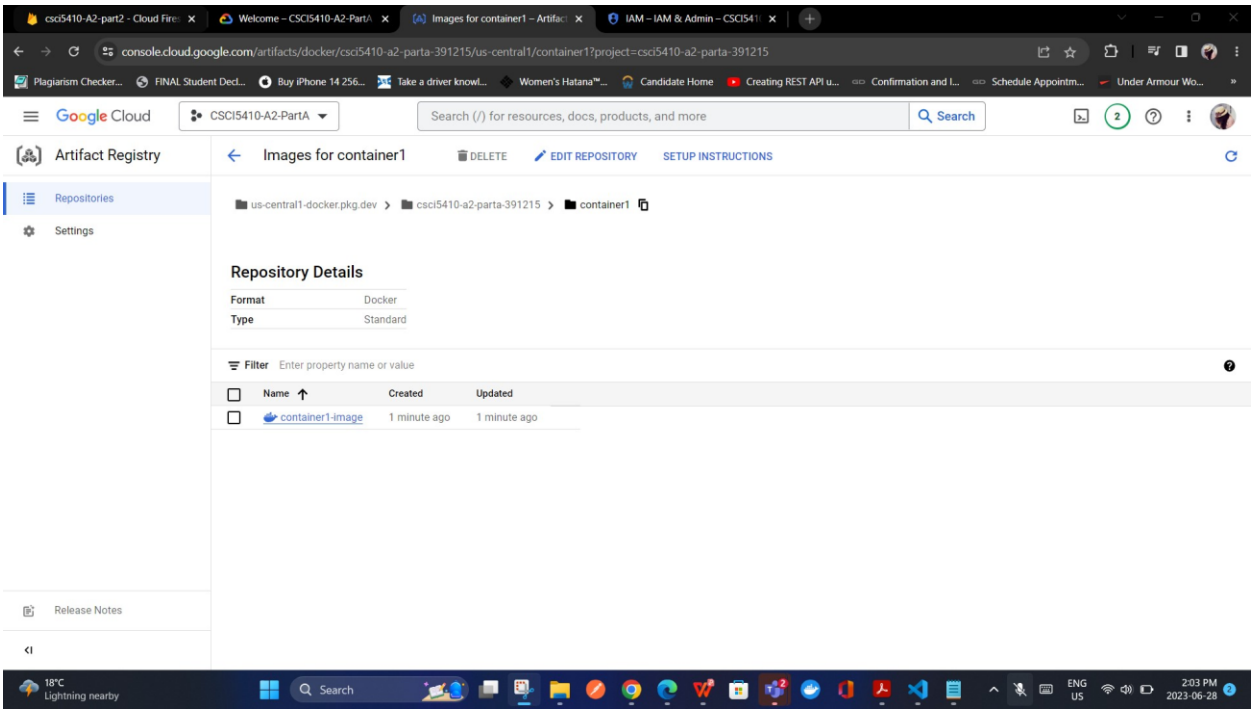

Figure 48 Docker image namely **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container1/container1-image** successfully pushed to Google Artifact registry [5]

Figure 49 shows that docker image **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container2/container2-image** pushed to repository conatiner2 of google cloud artifact registry.

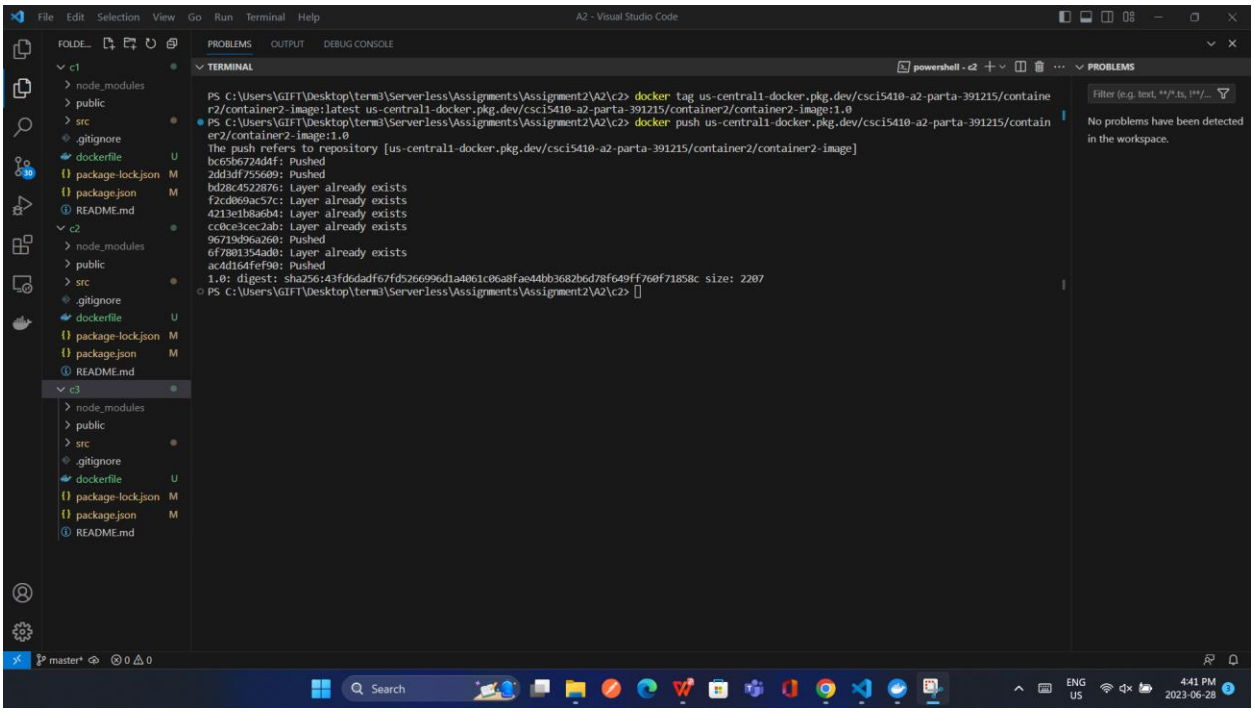

Figure 49 push Docker **image namely us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container2/container2-image** to Google Artifact registry

Figure 50 shows that docker image **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container2/container2-image** successfully pushed to repository conatiner2 of google cloud artifact registry.

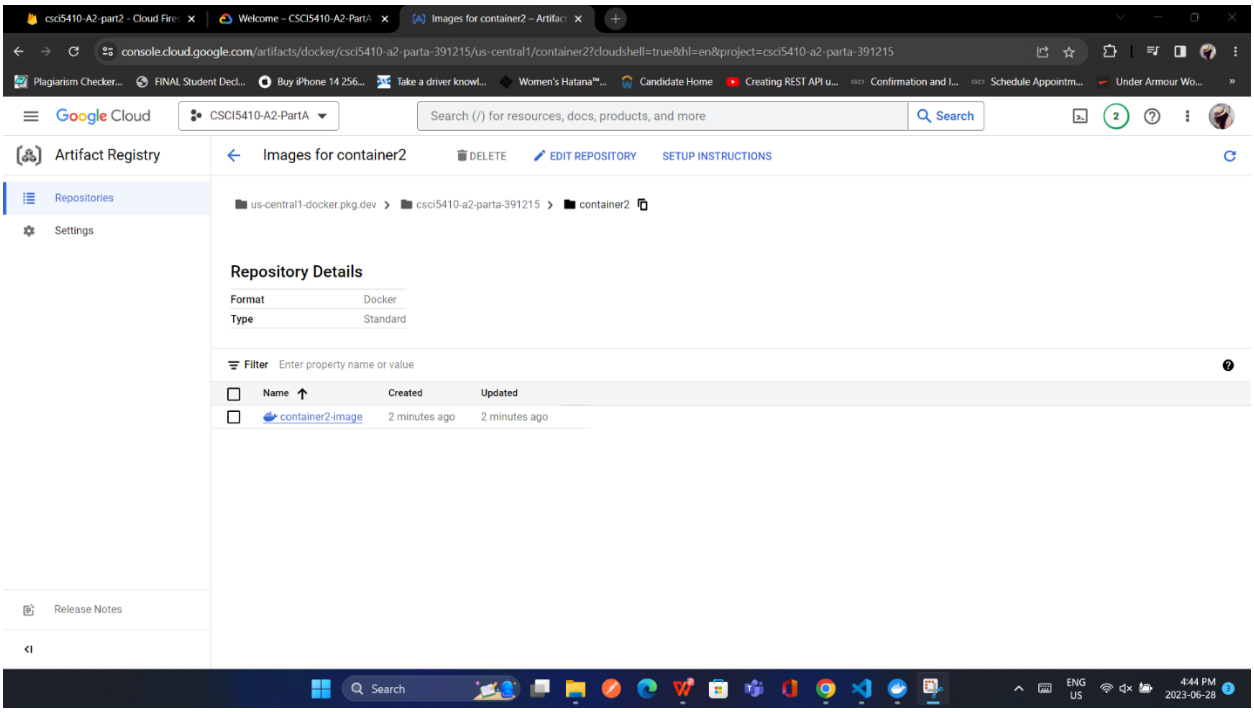

Figure 50 Docker image namely **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container2/container2-image** successfully pushed to Google Artifact registry [5]

Figure 51 shows that docker image **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container3/container3-image** pushed to repository conatiner3 of google cloud artifact registry.

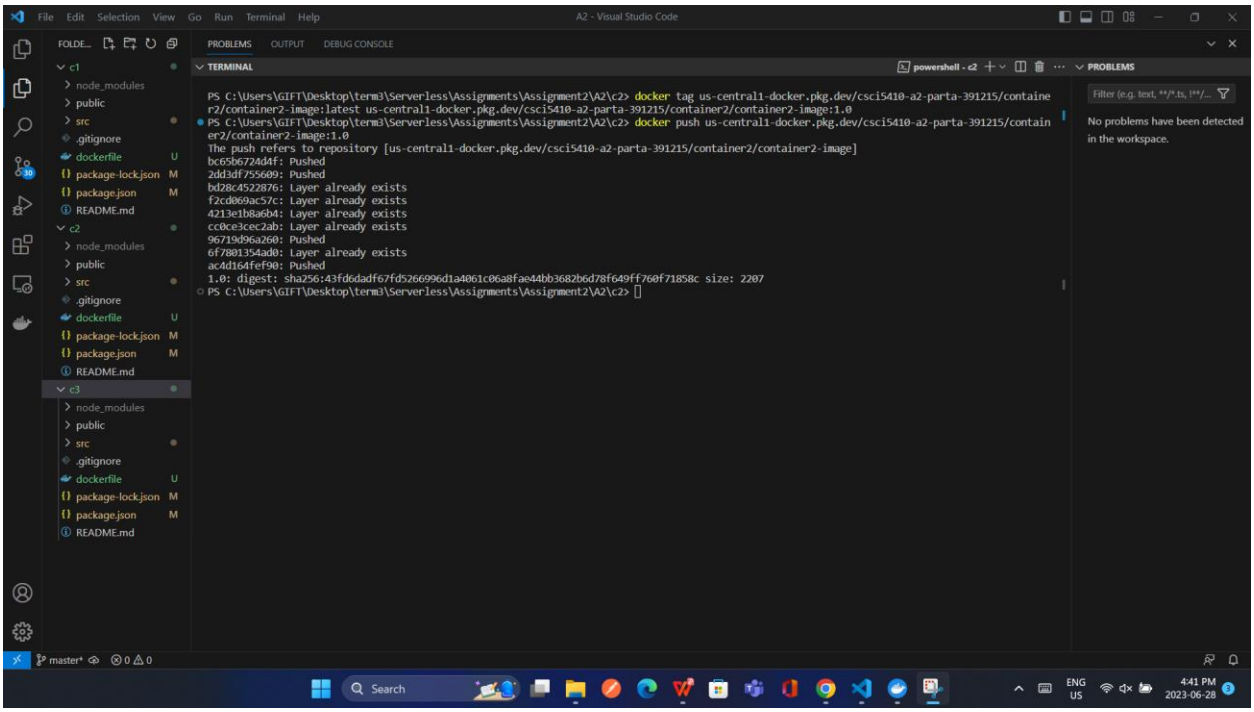

Figure 51 push Docker image namely us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container3/container3-image to Google Artifact registry

Figure 52 shows that docker image **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container3/container3-image** successfully pushed to repository conatiner2 of google cloud artifact registry.

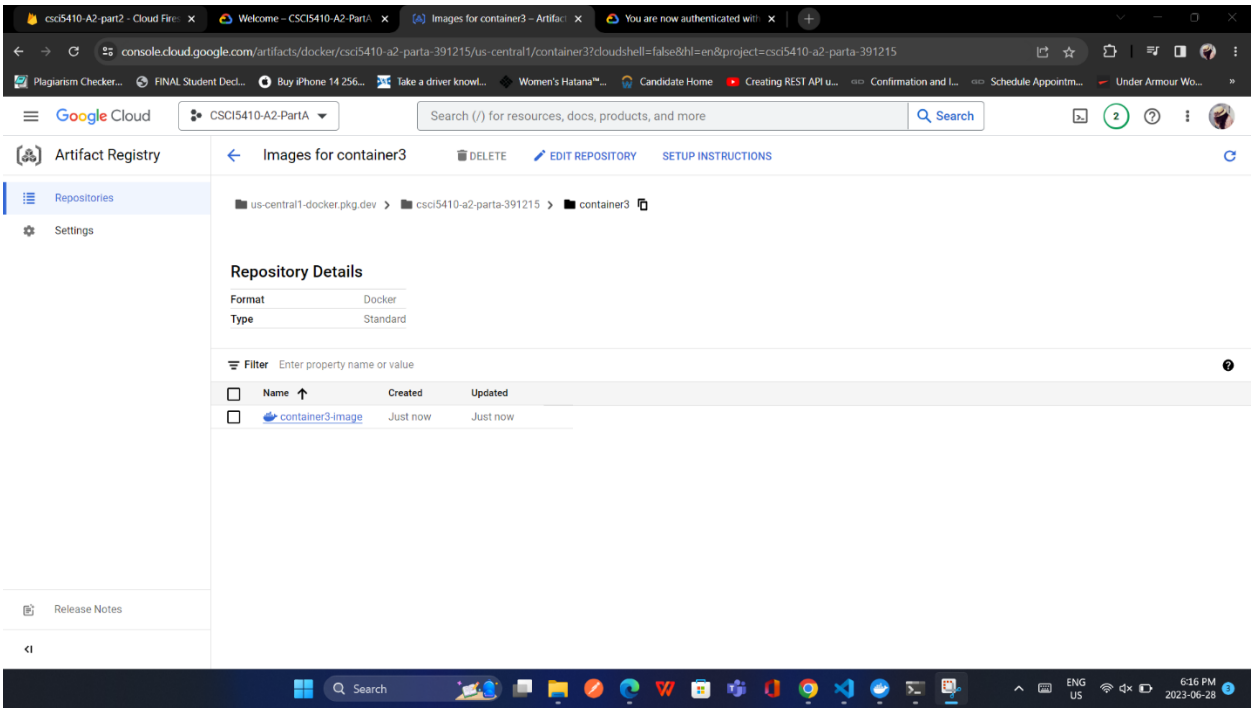

Figure 52 Docker image namely **us-central1-docker.pkg.dev/csci5410-a2-parta-391215/container3/container3-image** successfully pushed to Google Artifact registry [5]

Cloud Run Configuration

Figures 53,54,55,56,57 and 58 show steps required to configure docker image namely conatiner1-image to Google cloud tun.

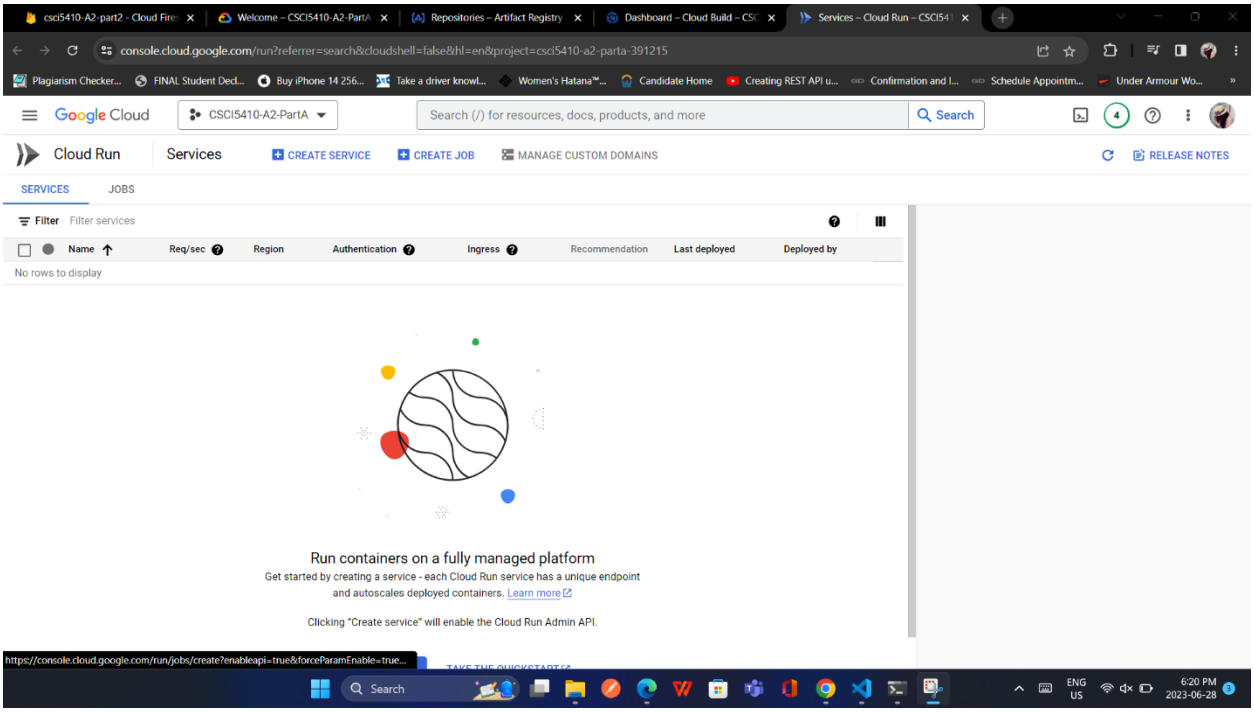

Figure 53 Configuration of conatiner1-image on Cloud run [4]

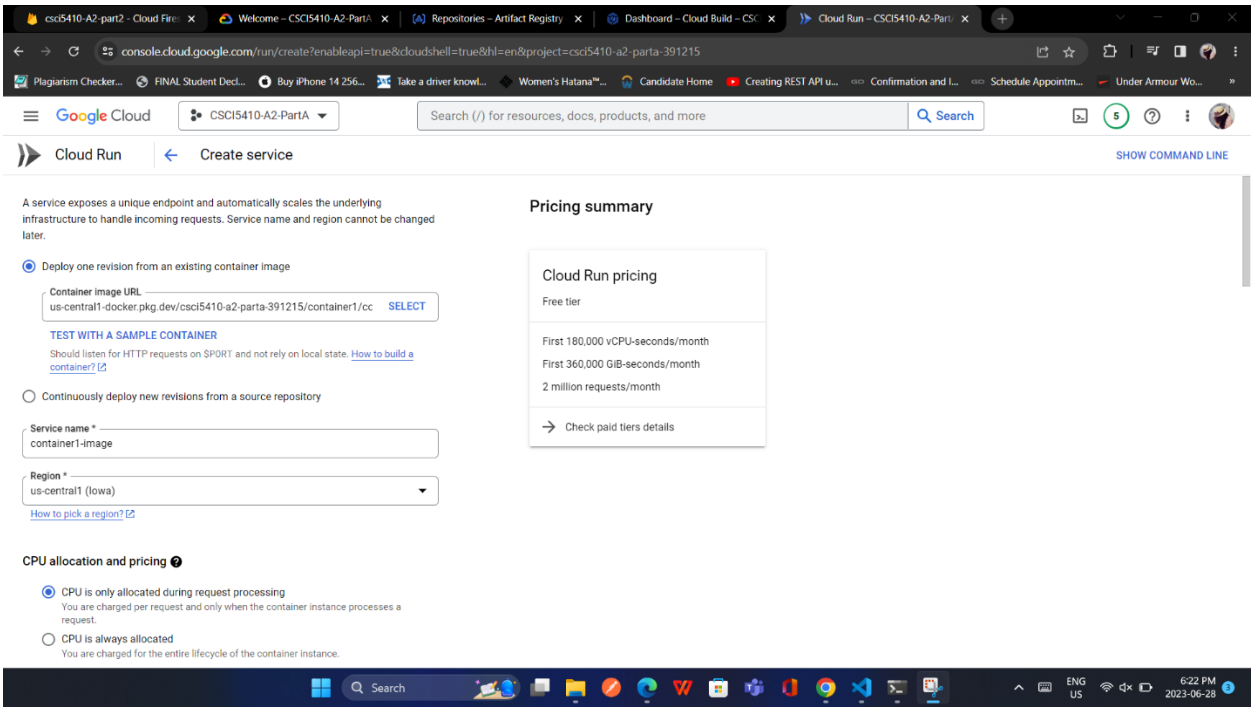

Figure 54 Configuration of conatiner1-image on Cloud run [4]

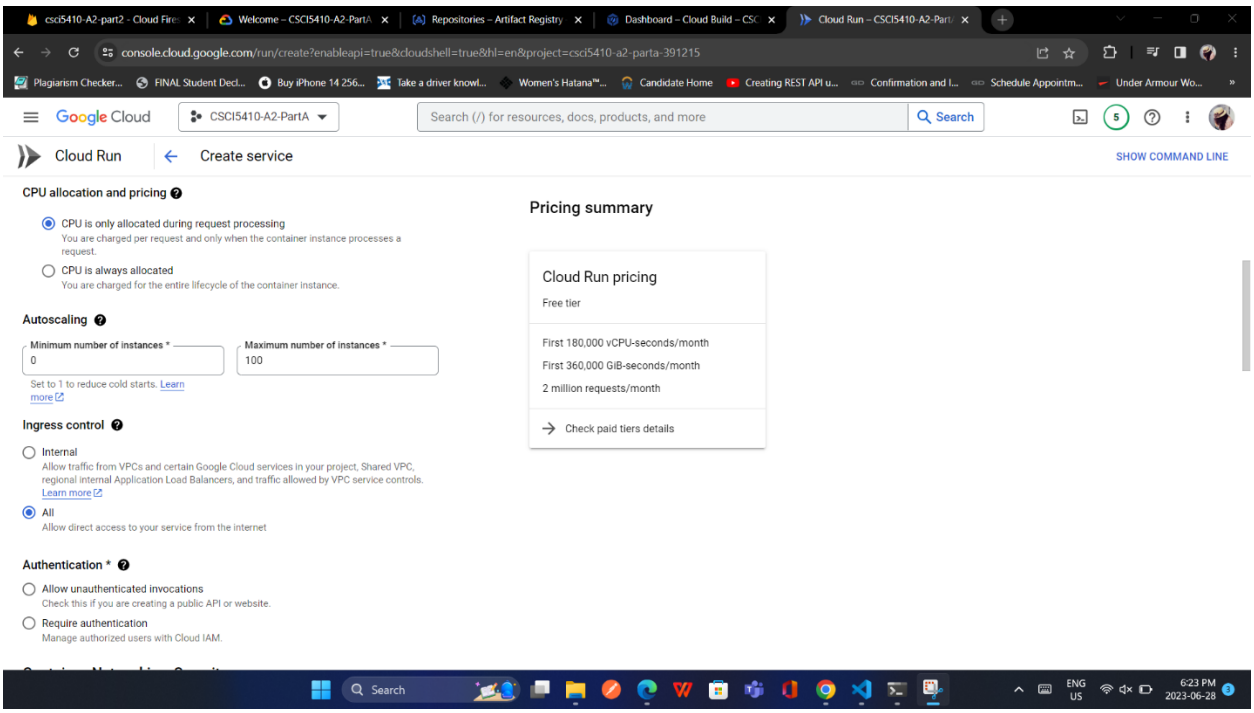

Figure 55 Configuration of conatiner1-image on Cloud run [4]

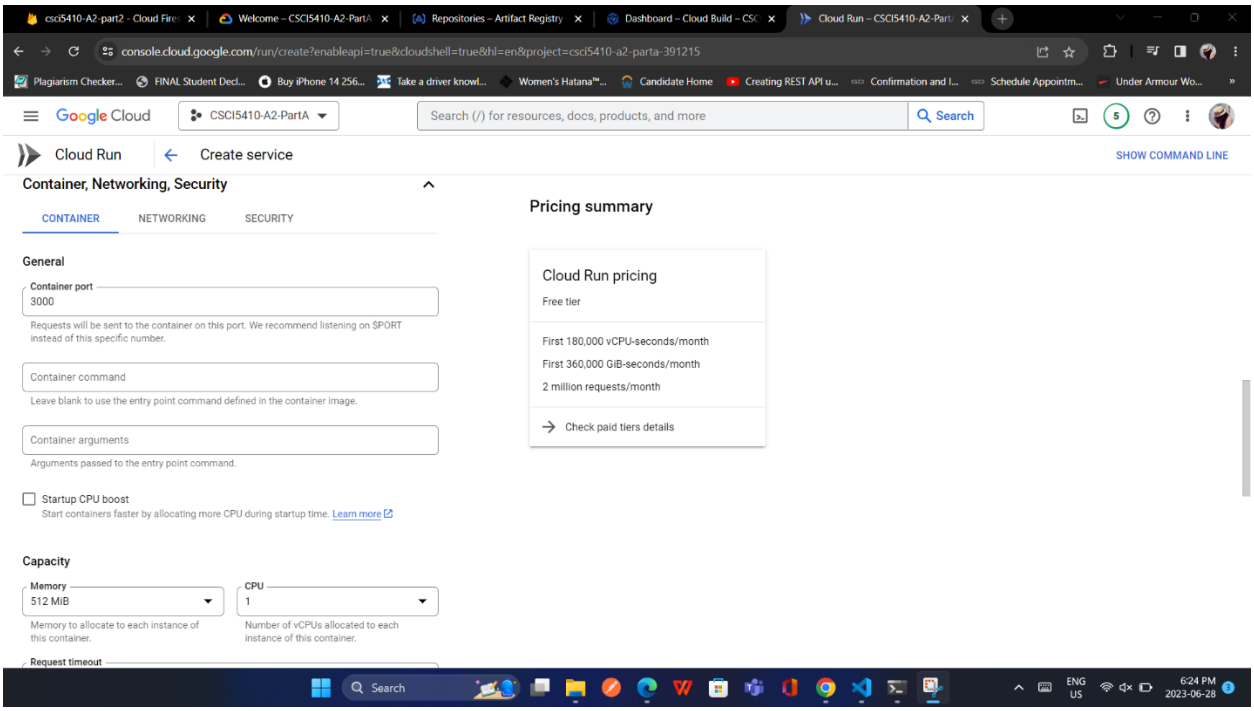

Figure 56 Configuration of conatiner1-image on Cloud run [4]

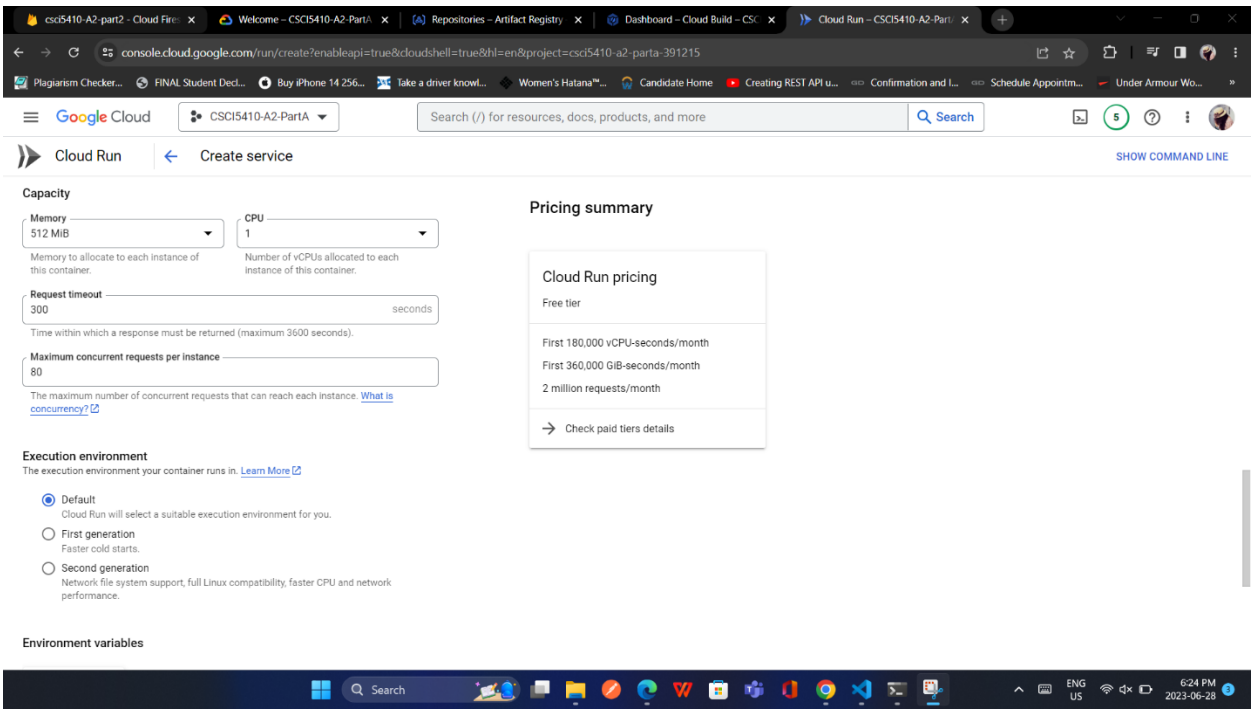

Figure 57 Configuration of conatiner1-image on Cloud run [4]

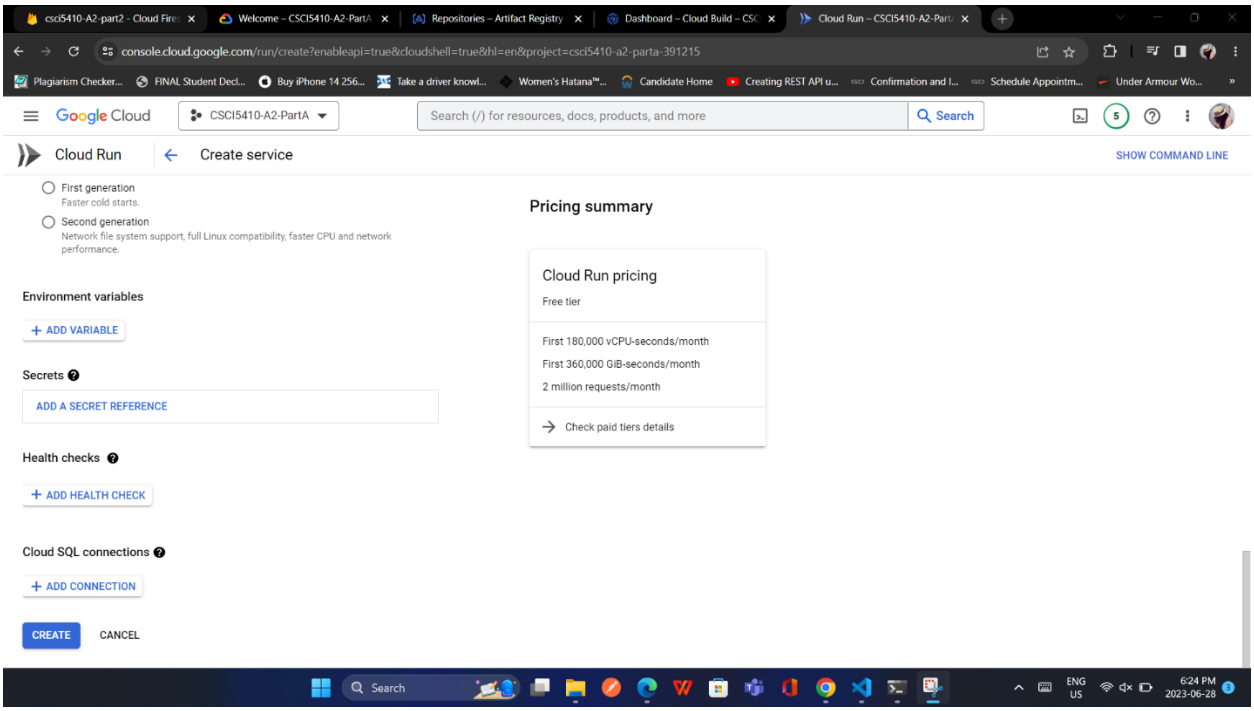

Figure 58 Configuration of conatiner1-image on Cloud run [4]

Figure 59 shows that conatiner1-image successfully build on Google Cloud Run.

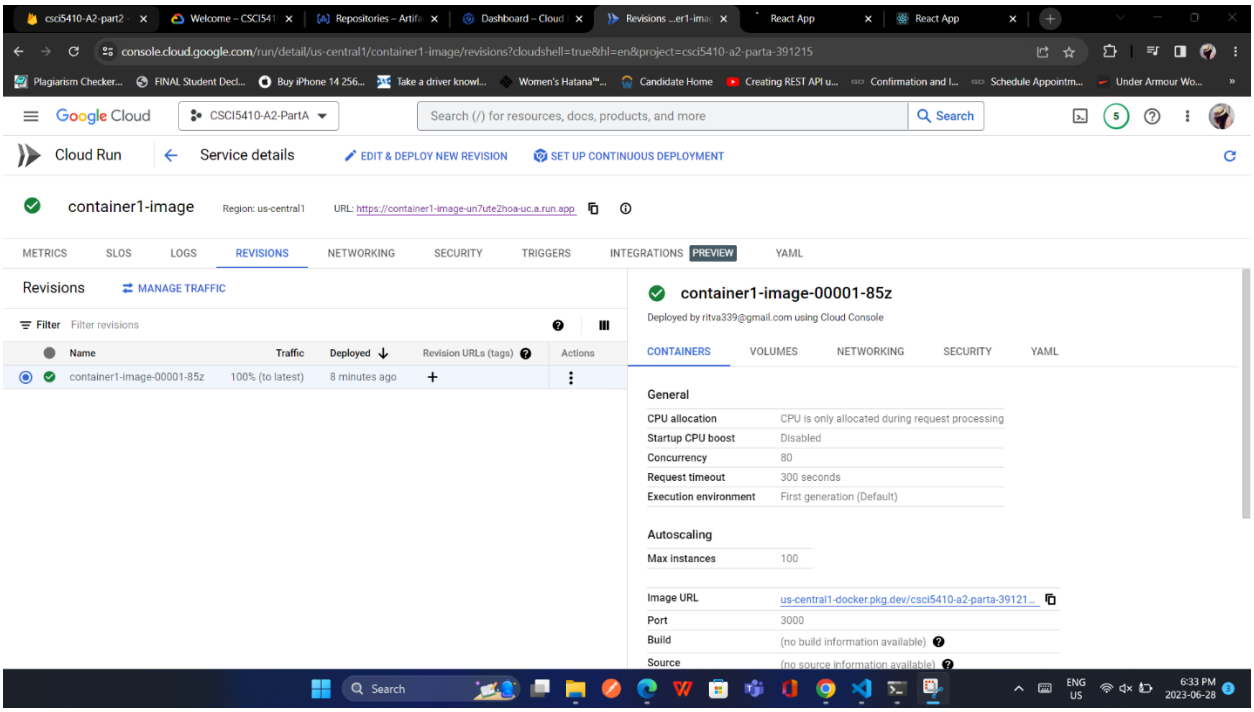

Figure 59 Conatiner1-image successfully deployed on cloud run [4]

Figures 60,61,62,63 and 64 show steps required to configure docker image namely conatiner2-image to Google cloud tun.

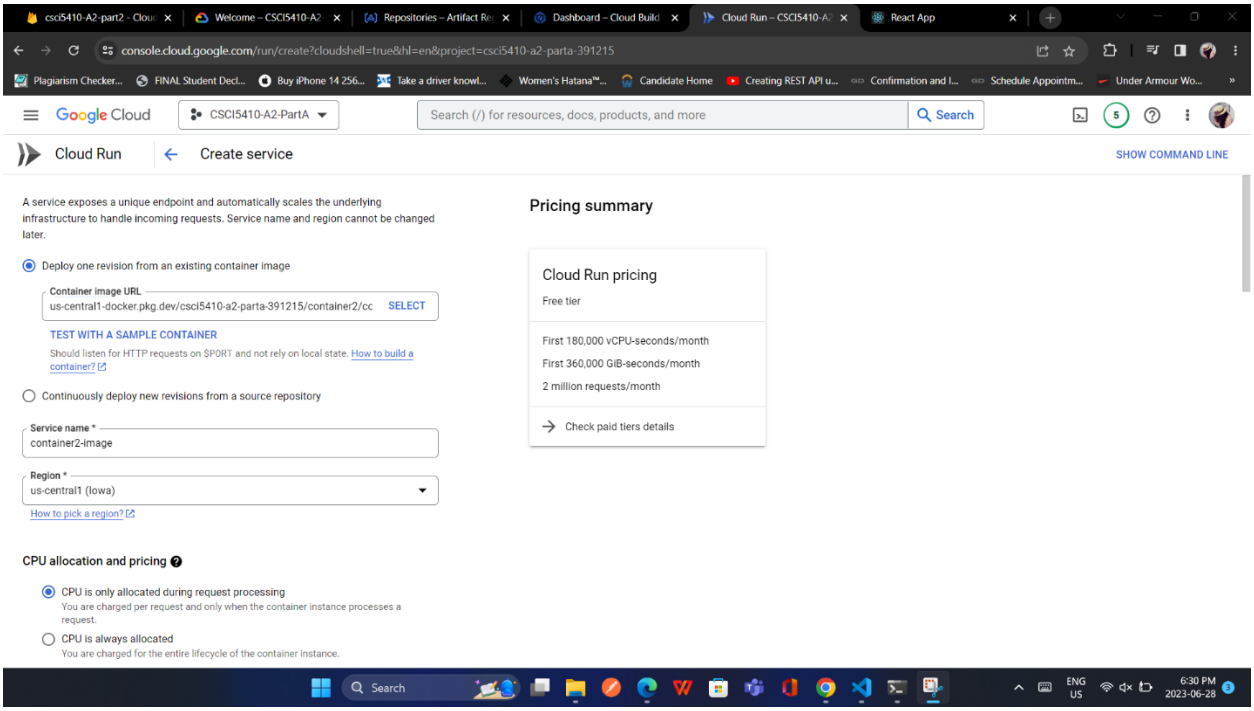

Figure 60 Configuration of conatiner2-image on Cloud run [4]

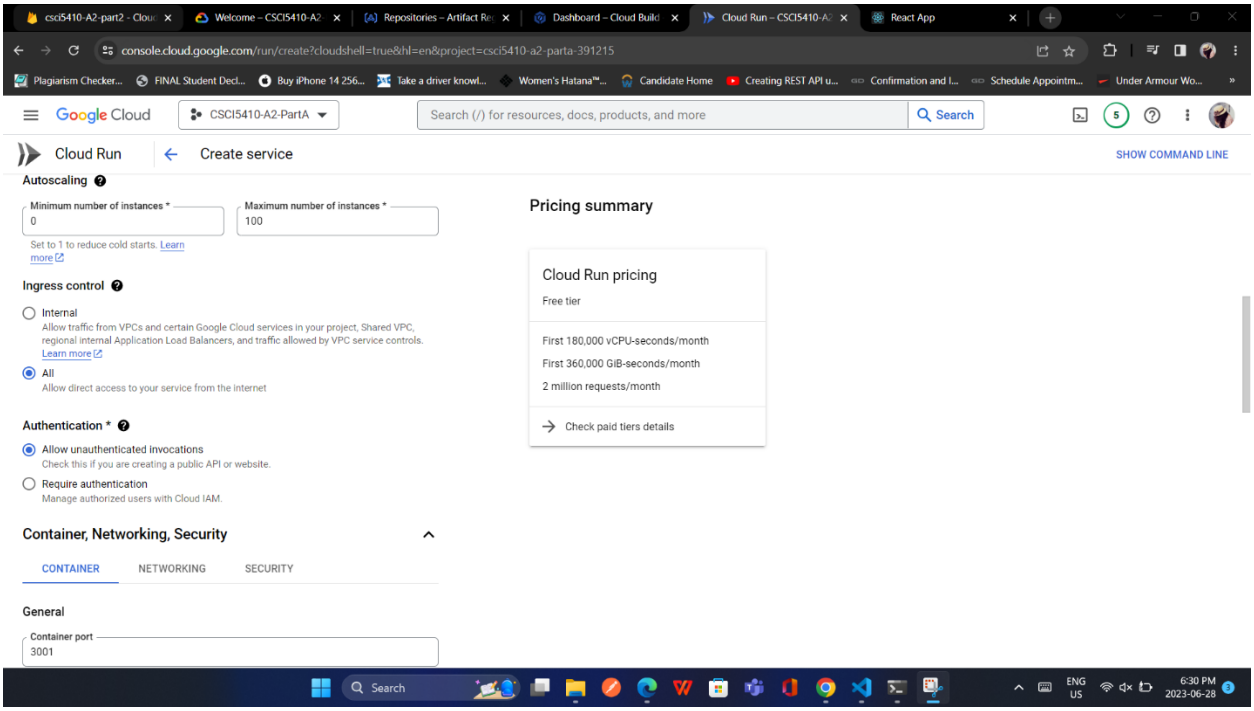

Figure 61 Configuration of conatiner2-image on Cloud run [4]

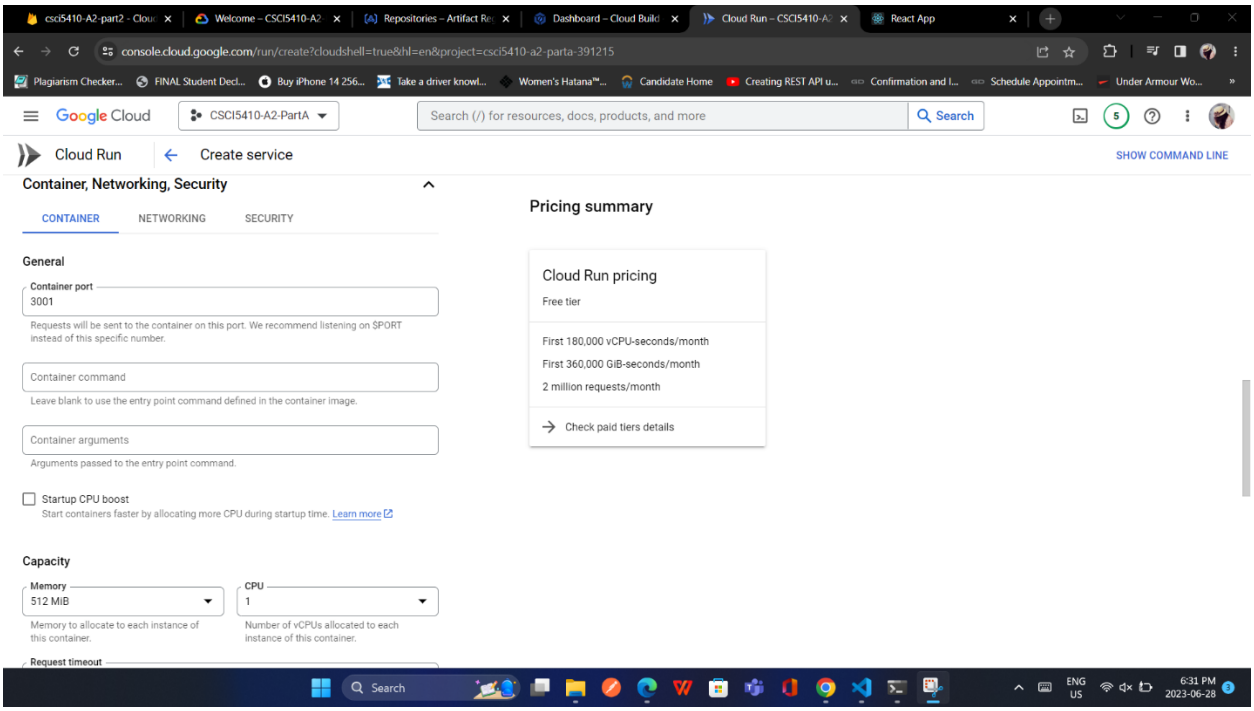

Figure 62 Configuration of conatiner2-image on Cloud run [4]

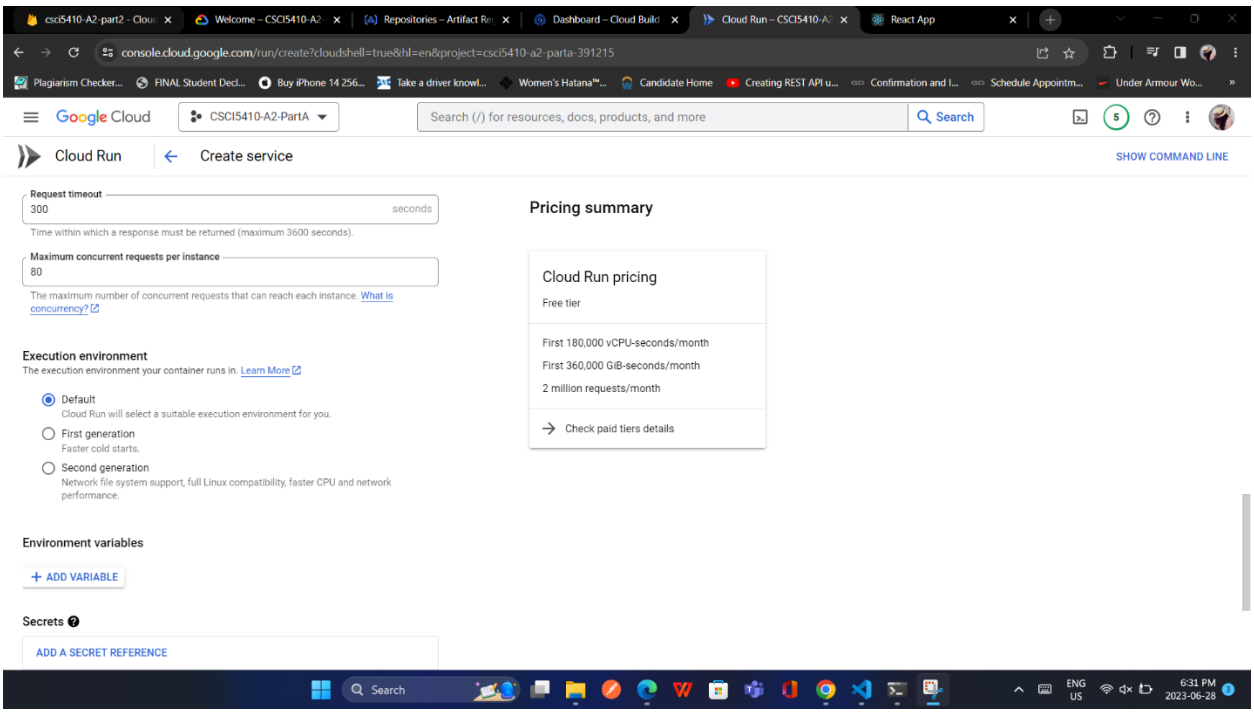

Figure 63 Configuration of conatiner2-image on Cloud run [4]

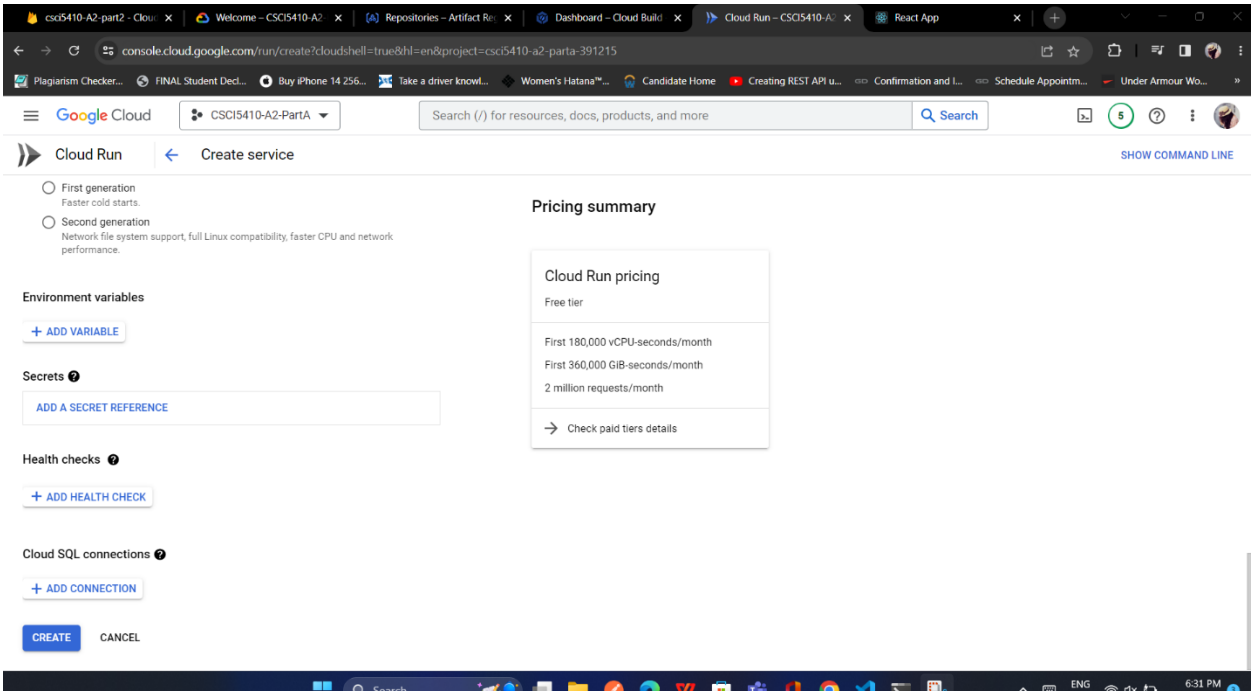

Figure 64 Configuration of conatiner2-image on Cloud run [4]

Figure 65 shows that conatiner2-image successfully build on Google Cloud Run.

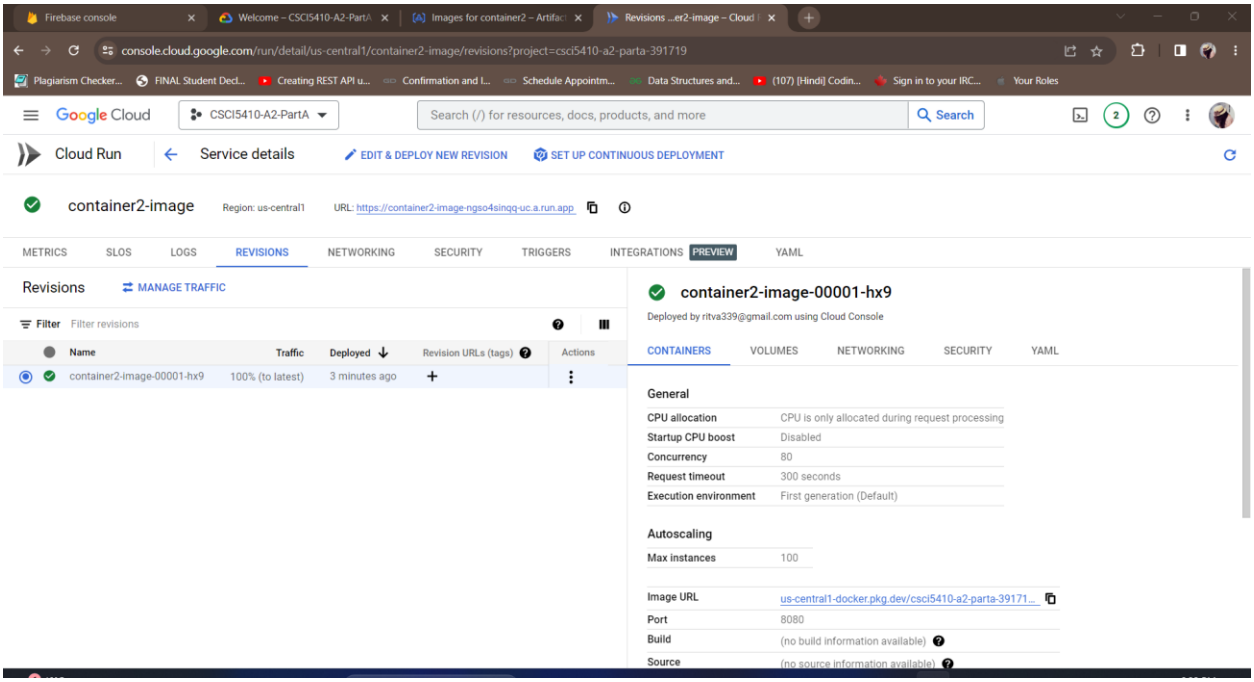

Figure 65 Conatiner2-image successfully deployed on cloud run [4]

Figures 66,67,68 and 69 show steps required to configure docker image namely conatiner3-image to Google cloud tun.

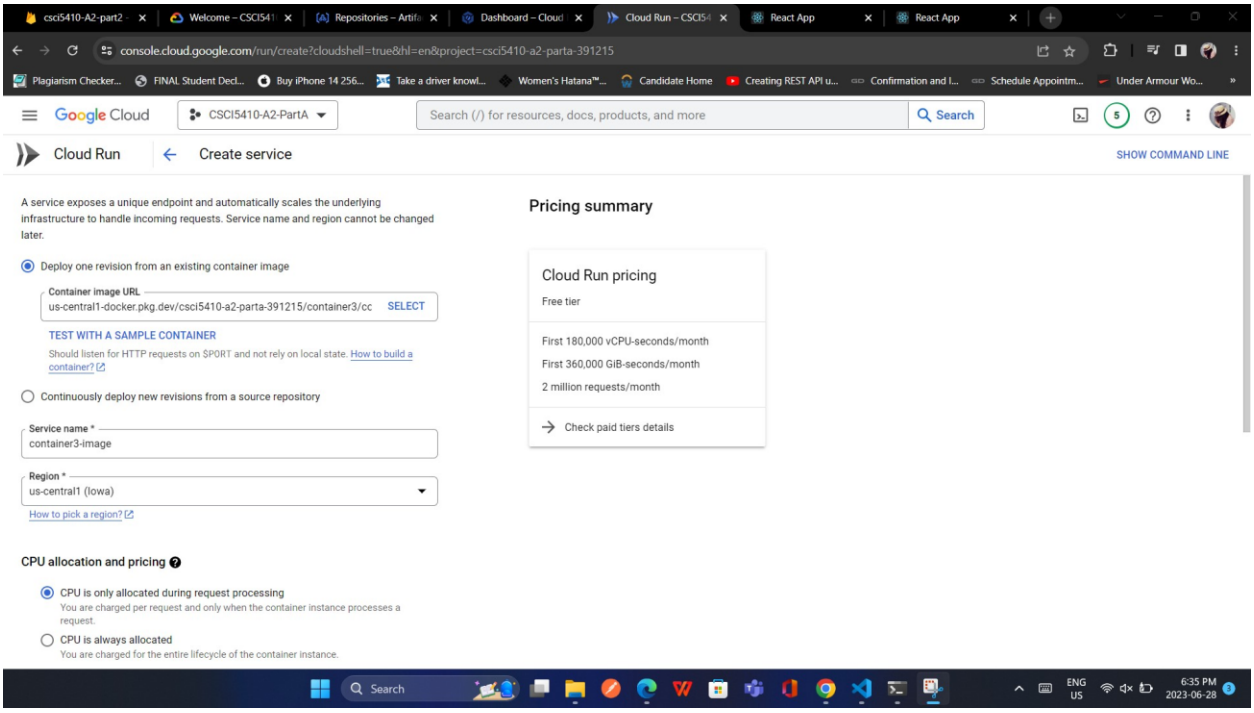

Figure 66 Configuration of conatiner3-image on Cloud run [4]

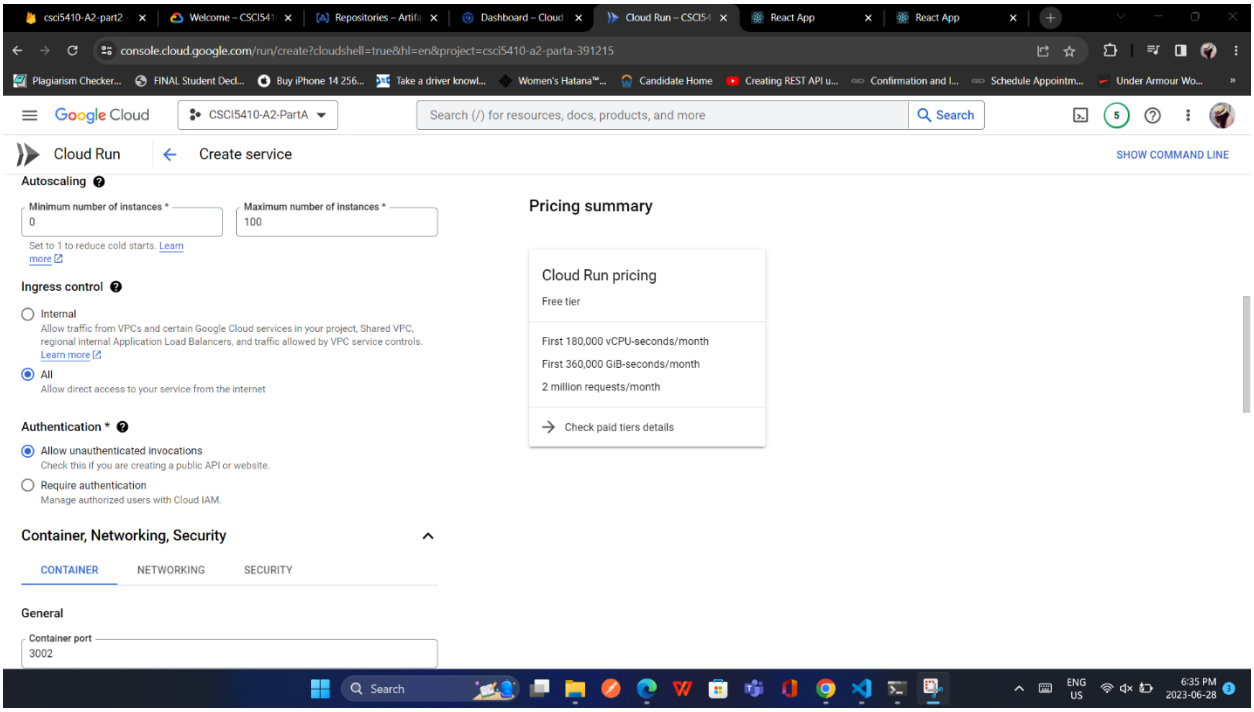

Figure 67 Configuration of conatiner3-image on Cloud run [4]

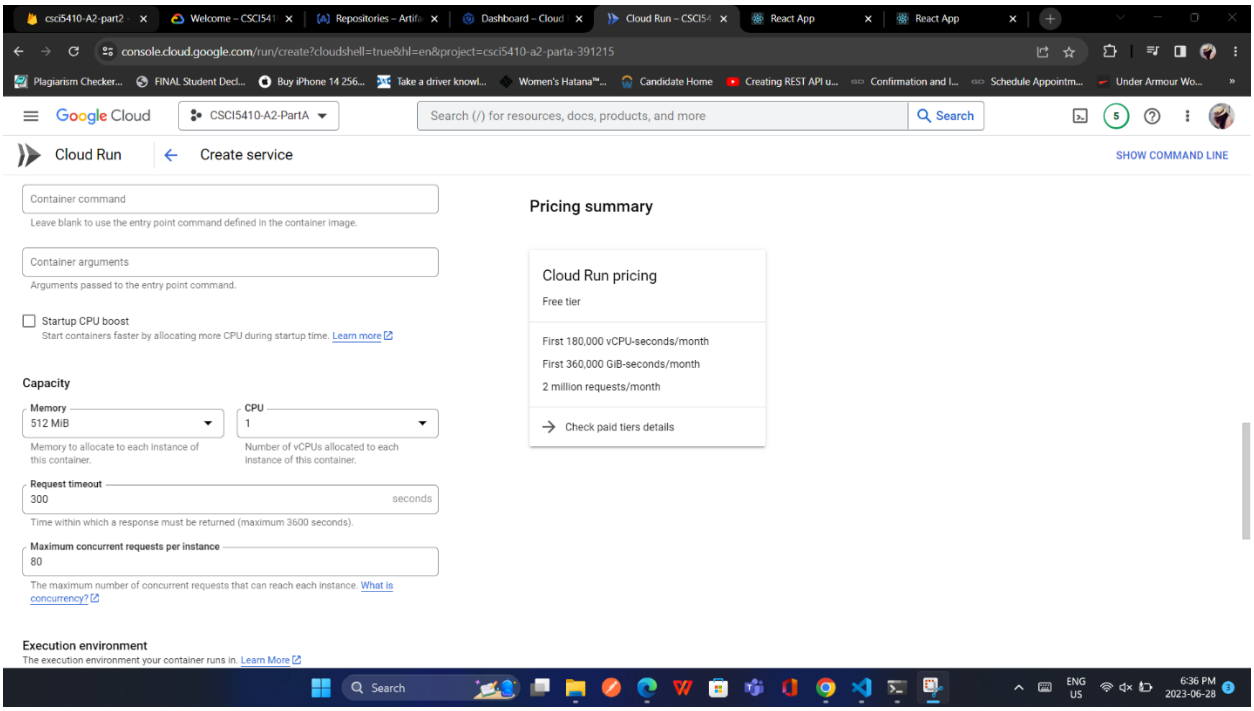

Figure 68 Configuration of conatiner3-image on Cloud run [4]

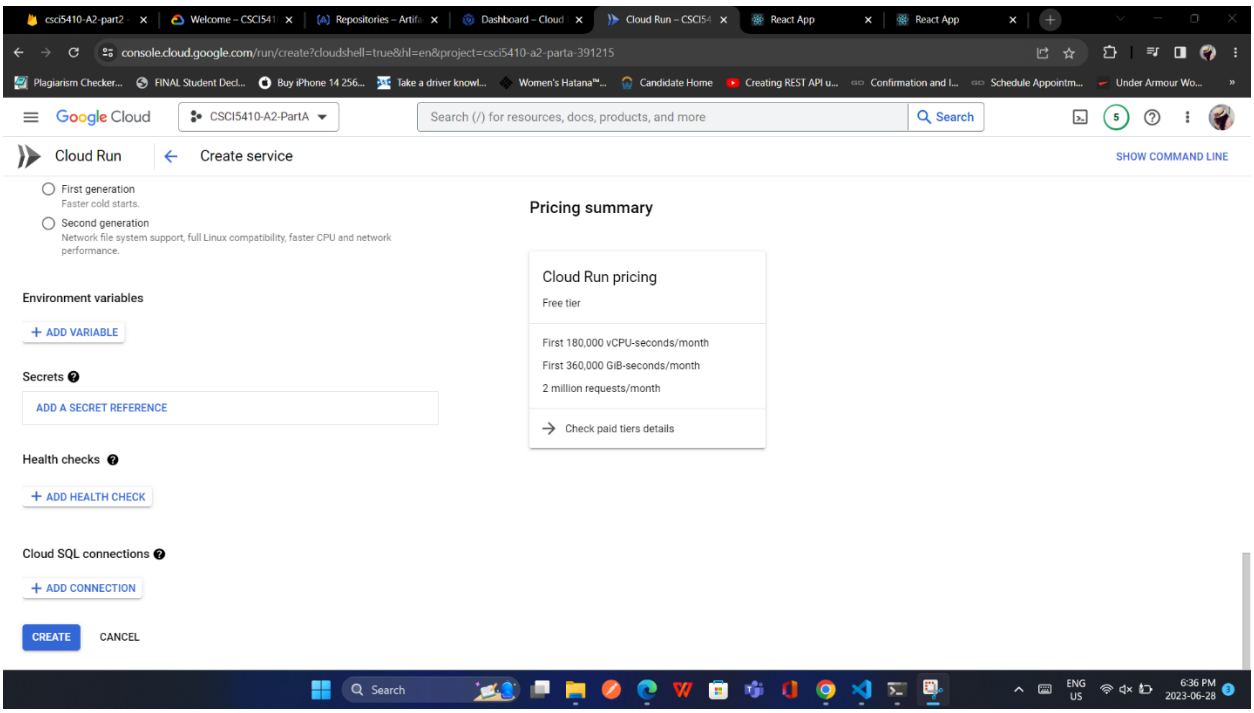

Figure 69 Configuration of conatiner3-image on Cloud run [4]

Figure 70 shows that conatiner3-image successfully build on Google Cloud Run.

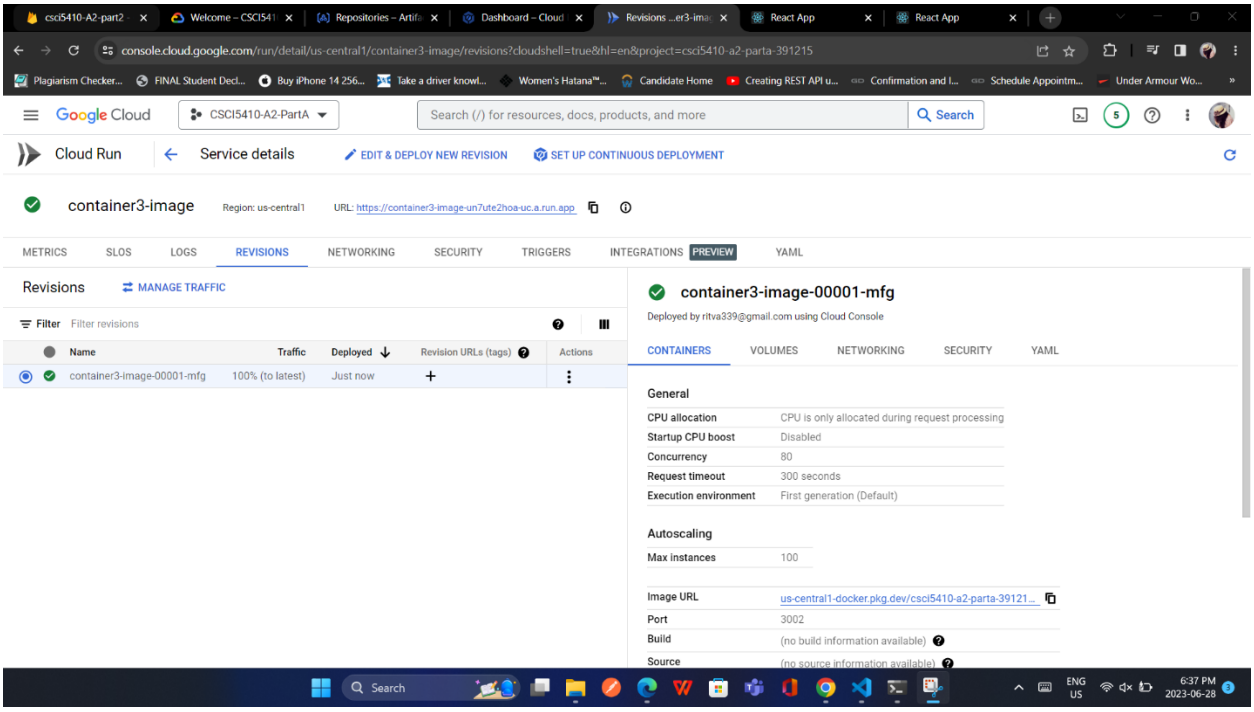

Figure 70 Conatiner3-image successfully deployed on cloud run [4]

Figure 71 displays all the images successfully build on Google Cloud Run.

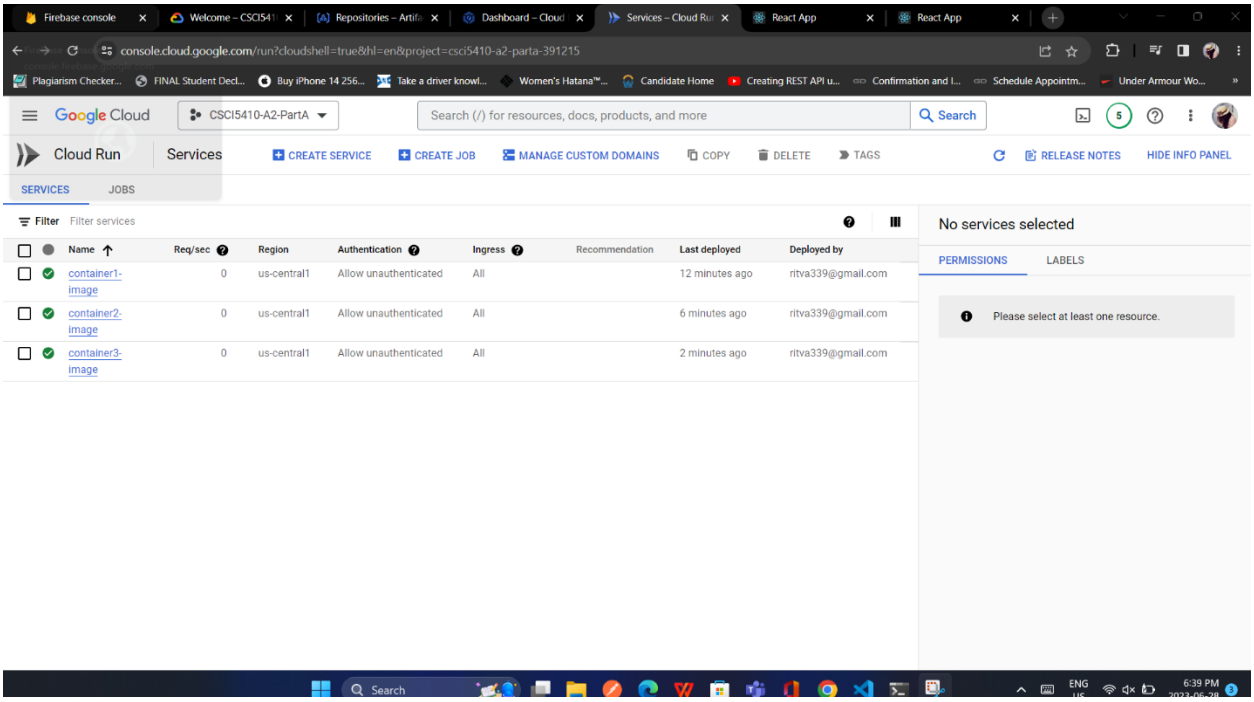

Figure 71 successfully deployed three images on Google Cloud Run [4]

Deployed Web application

Figure 72 shows the Registration react application deployed on cloud run and user has entered all the fields for registration. Figure 73 shows the popup about registration successful.

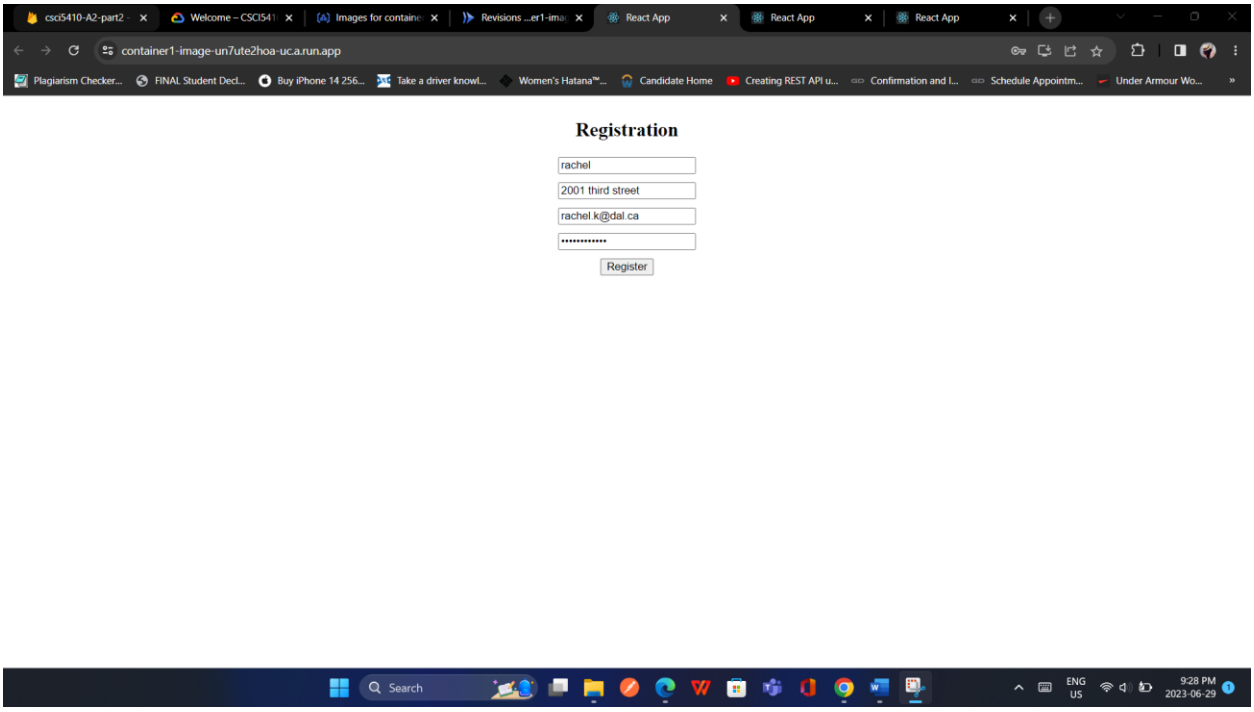

Figure 72 Deployed Registration react application on Google Cloud Run [4]

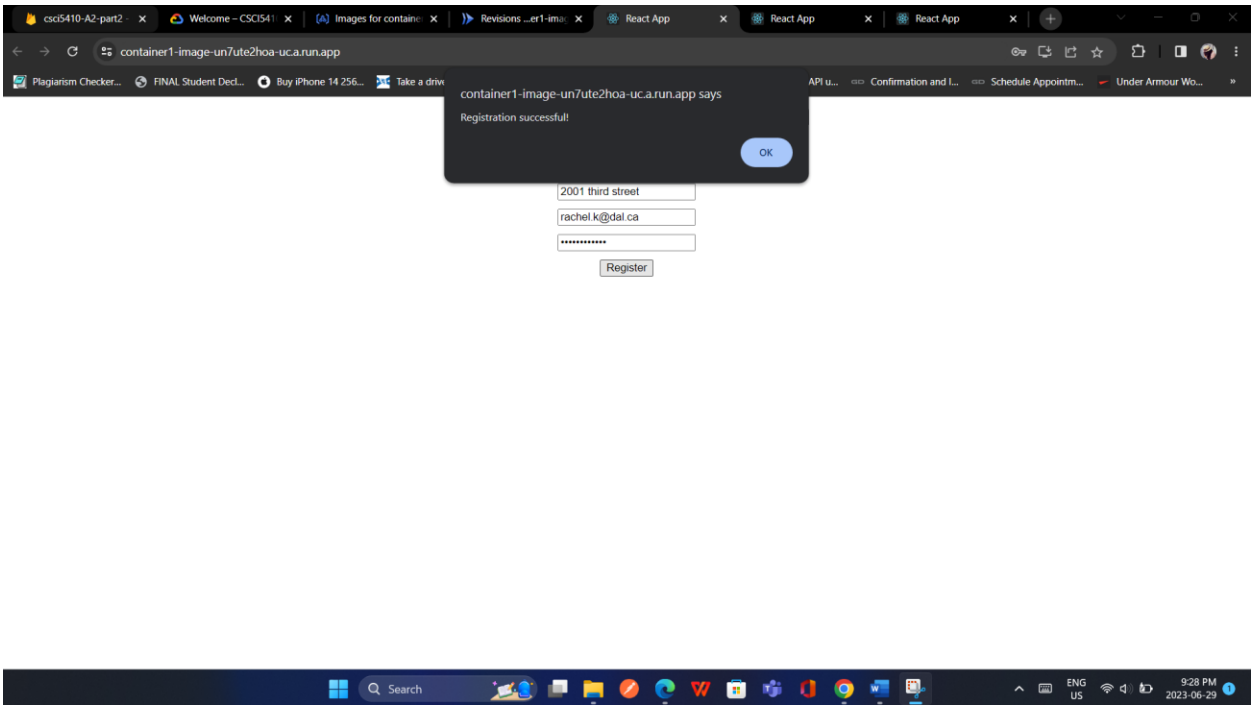

Figure 73 User Registered successfully [4]

Figure 74 defines that user's details successfully stored in Reg collection of csci5410-A2-part2.

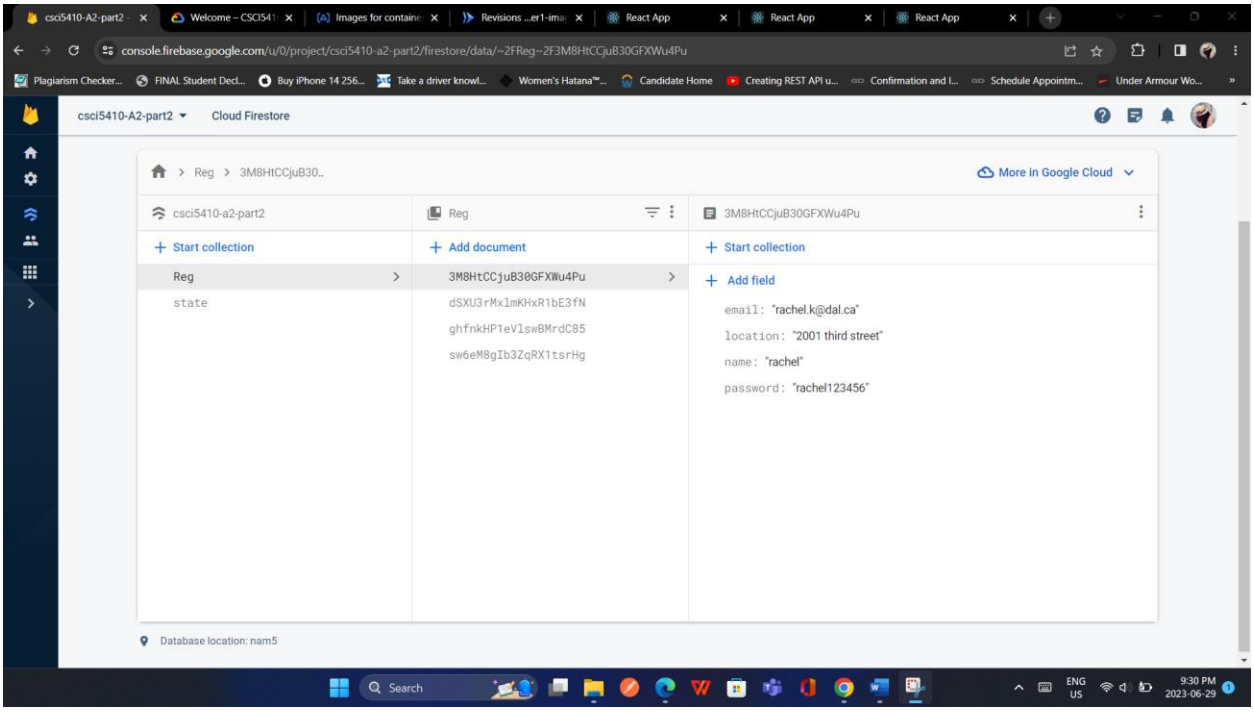

Figure 74 User's details stored successfully in Reg collection [2]

Figure 75 defines the state collection of csci5410-A2-part2 which stores registered user's details as well as timestamp and online/offline status. In below figure, user Rachel is not logged in yet, so her status is offline.

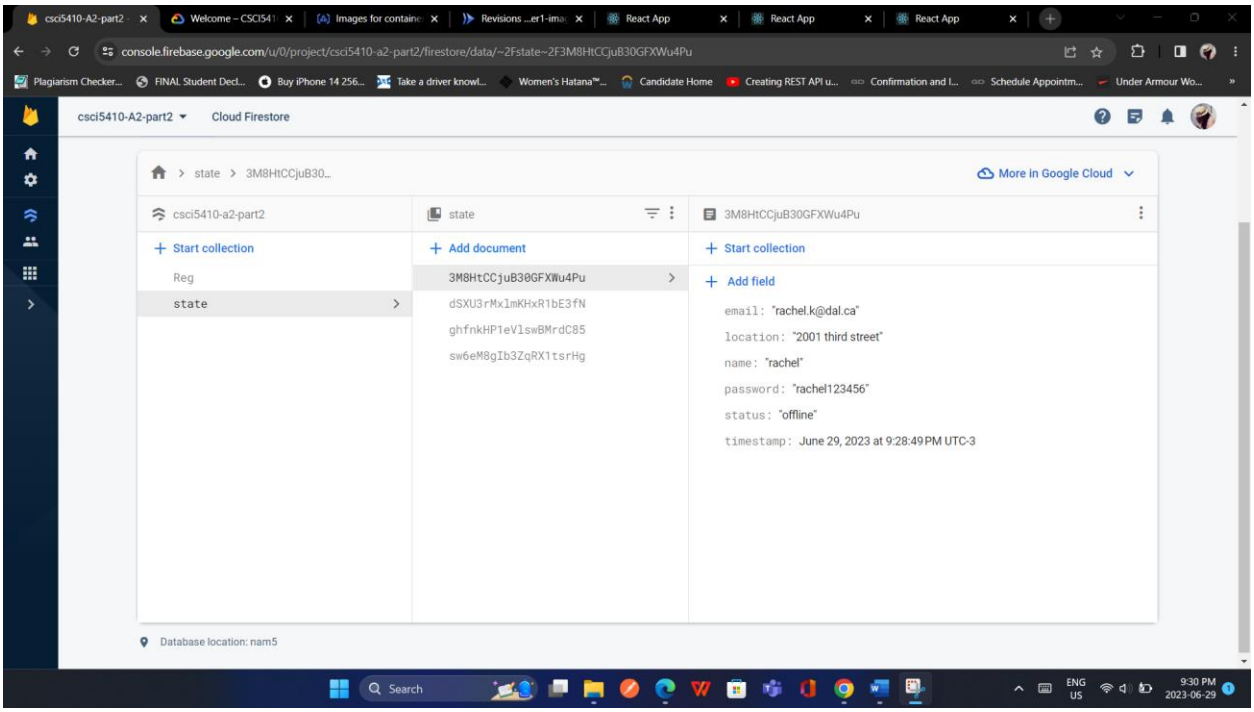

Figure 75 State collection with user's details, timestamps, and status [2]

Figure 76 shows the Login react application deployed on cloud run and it validates the login credentials with firebase firestore database. Figure 77 illustrates that on successful login, pop up will be shown up. In addition, Figure 78 shows that user status changed from offline to online on successful login.

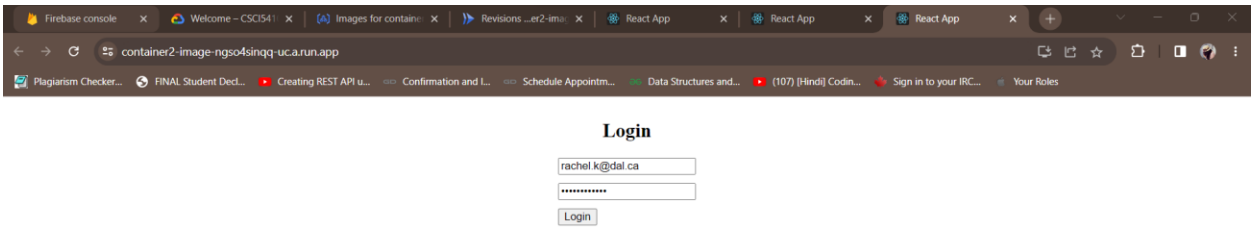

Figure 76 Deployed Login react application on Google Cloud Run [4]

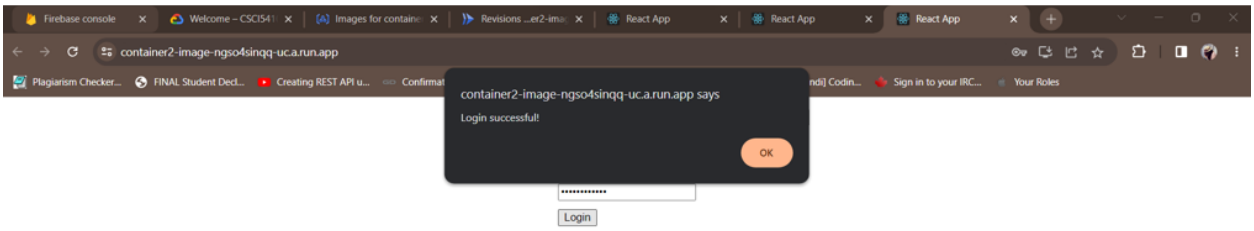

Figure 77 User's Credentials Validate Successfully [4]

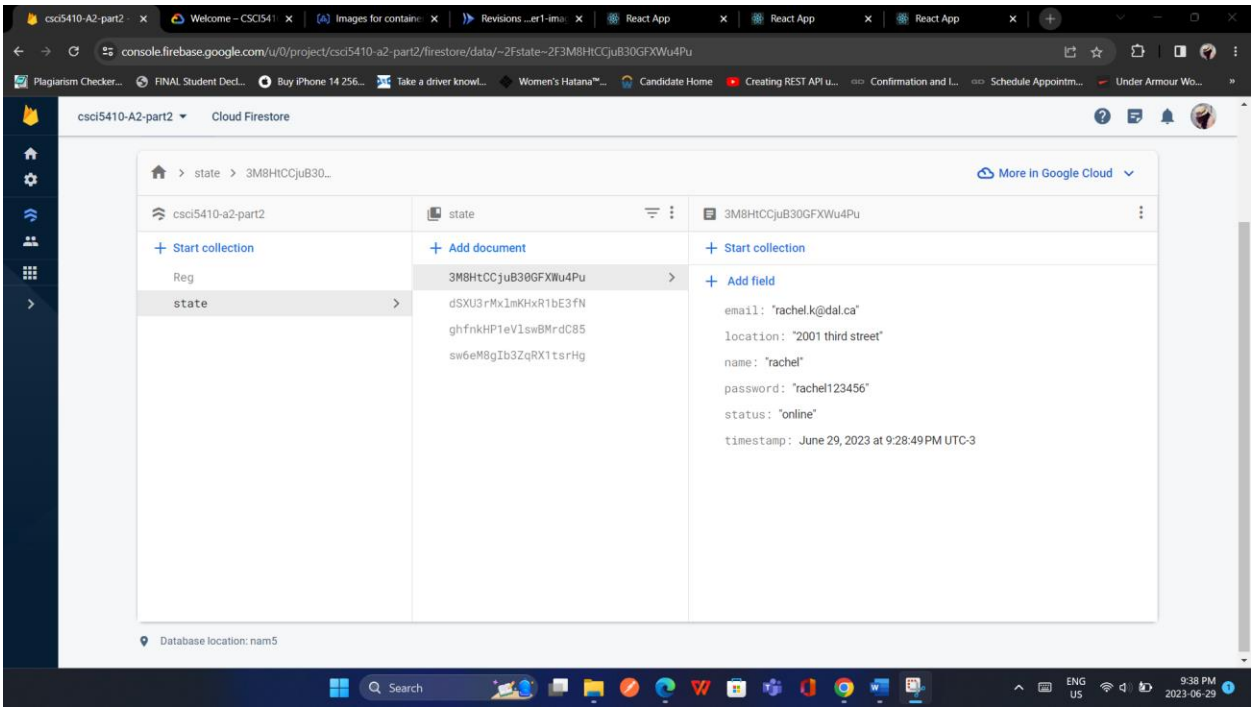

Figure 78 On successful login, user's status changed form offline to online [2]

Figure 77 shows the Profile react application deployed on Google Cloud Run. The below figure showcases that the user "rachel" is currently logged in, and along with her own details, the page also displays the information of all the online users. This feature allows users to have visibility and access to the profiles of other users who are currently logged in.

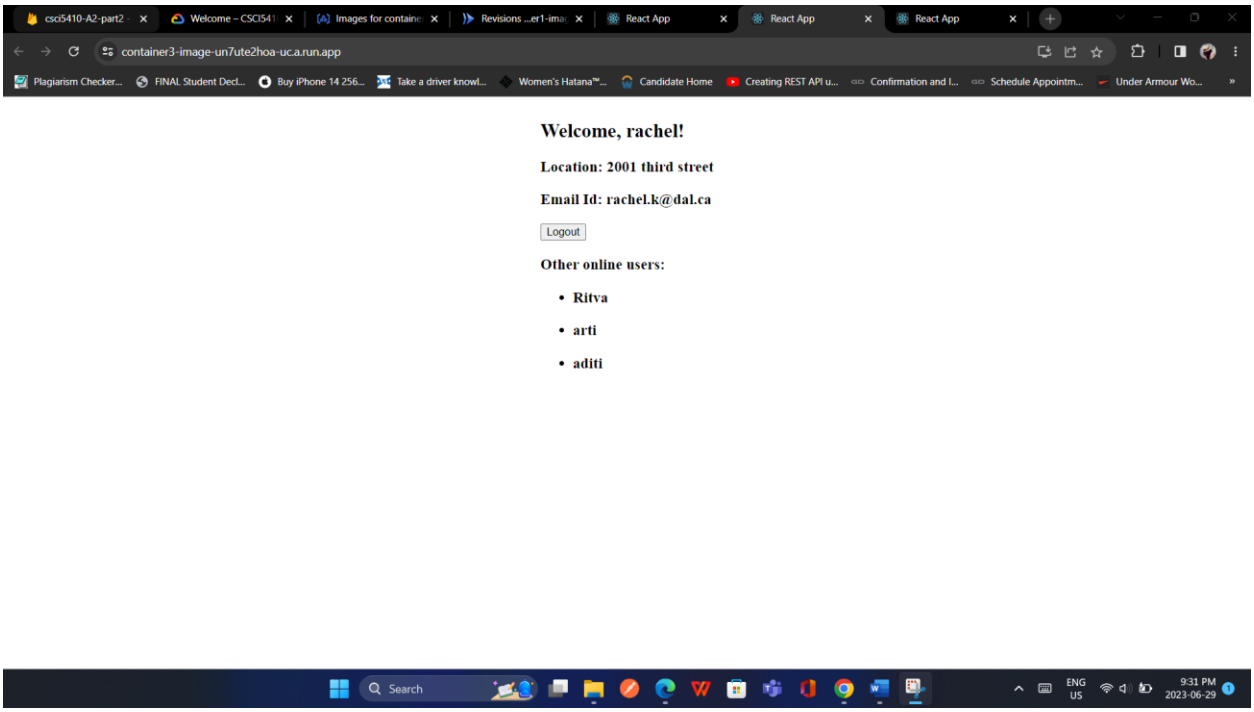

Figure 79 Deployed Profile react application on Google Cloud Run [4]

Test Cases

Registration react application:

Below method will validate all the fields such as name, email, location, and password.

```
const handleSubmit = async (e) => {
    e.preventDefault();
    const nameRegex = /^[A-Za-z]+$/;
    if (!nameRegex.test(name)) {
      setError('Name should contain only letters');
      return;
    }
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(email)) {
      setError('Invalid email format');
      return;
    }
    const passwordRegex = /^(?=.*[a-zA-Z])(?=.*\d).{8,}$/;
    if (!passwordRegex.test(password)) {
      setError('Password should contain at least 8 characters, including letters
and digits');
      return;
    }
```

For all the Fields, mentioned all the fields as required fields.

```
            <input
              type="text"
              placeholder="Name"
              value={name}
             onChange={(e) => setName(e.target.value)}
              required
             style={{ marginBottom: '10px' }}
            />

           <input
             type="text"
             placeholder="Location"
             value={location}
             onChange={(e) => setLocation(e.target.value)}
             required
             style={{ marginBottom: '10px' }}
```

```
        />

        <input
          type="email"
          placeholder="Email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          required
          style={{ marginBottom: '10px' }}
        />

         <input
          type="password"
          placeholder="Password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          required
          style={{ marginBottom: '10px' }}
        />
```

Below method will store all the details in database.

```
try {
      const db = firebase.firestore();
      const regRef = await db.collection('Reg').add(registrationData);
      await db.collection('state').doc(regRef.id).set(userStateData);
      setName('');
      setLocation('');
      setEmail('');
      setPassword('');
      setError('');
      alert('Registration successful!');
      window.location.href = 'http://localhost:3001/login';
    } catch (error) {
      console.error('Error registering user:', error);
    }
  };
```

Figures 80,81,82 and 83 show all the validations of registration page.

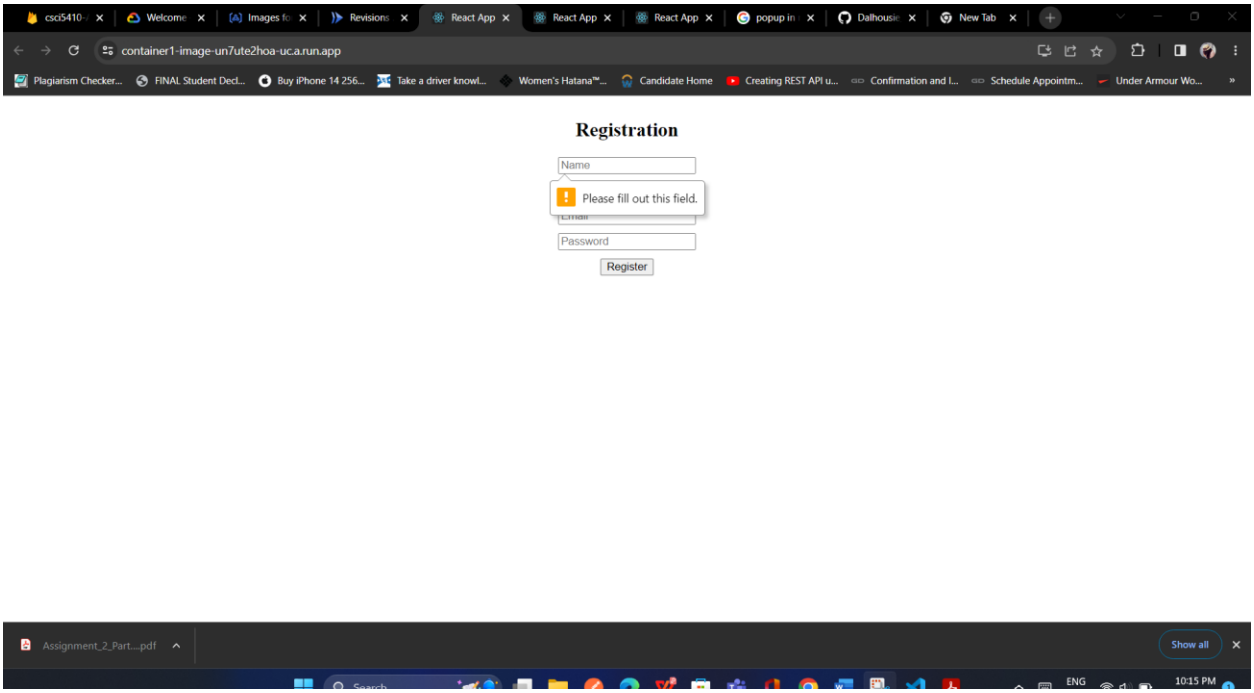

Figure 80 Validation for Registration page (name) [4]

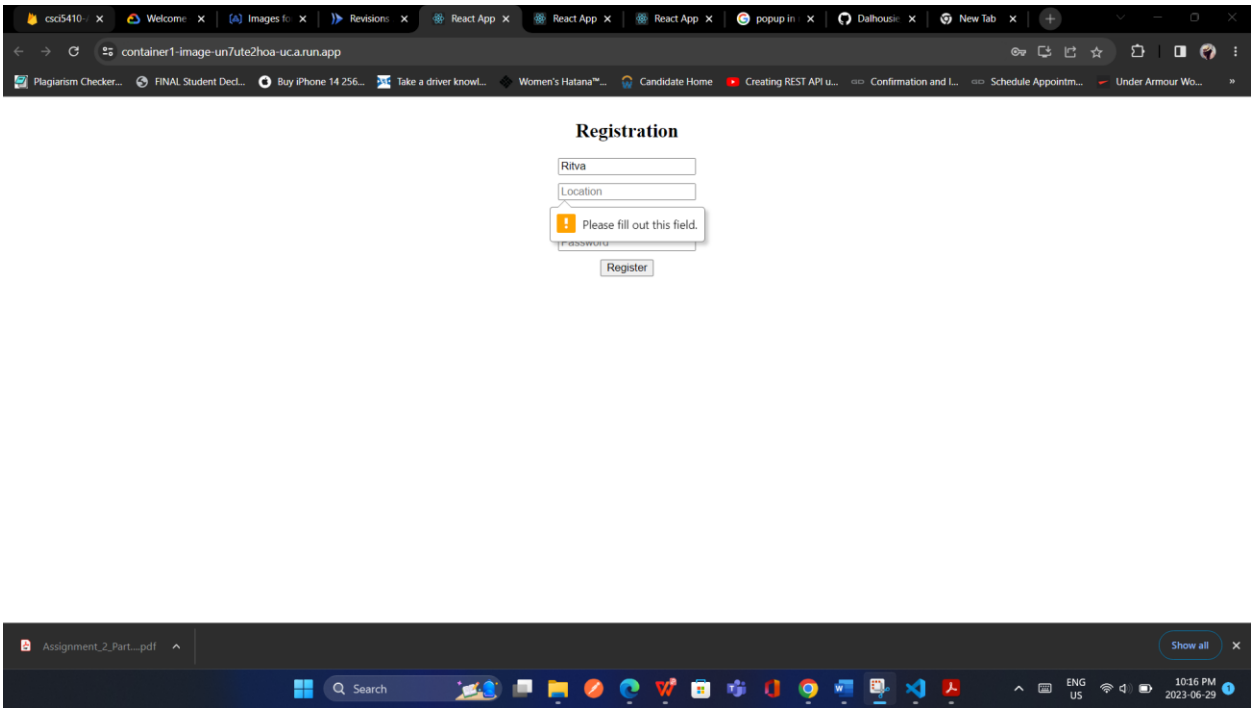

Figure 81 Validation for Registration page (location) [4]

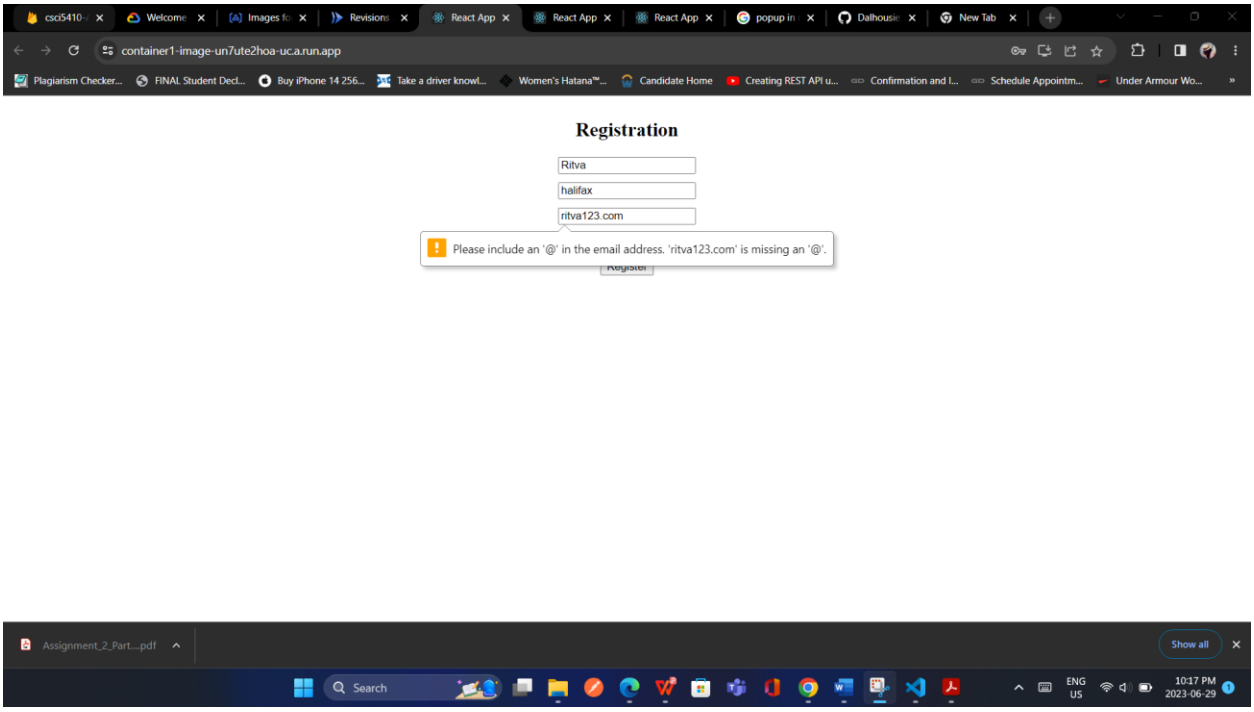

Figure 82 Validation for Registration page (email) [4]

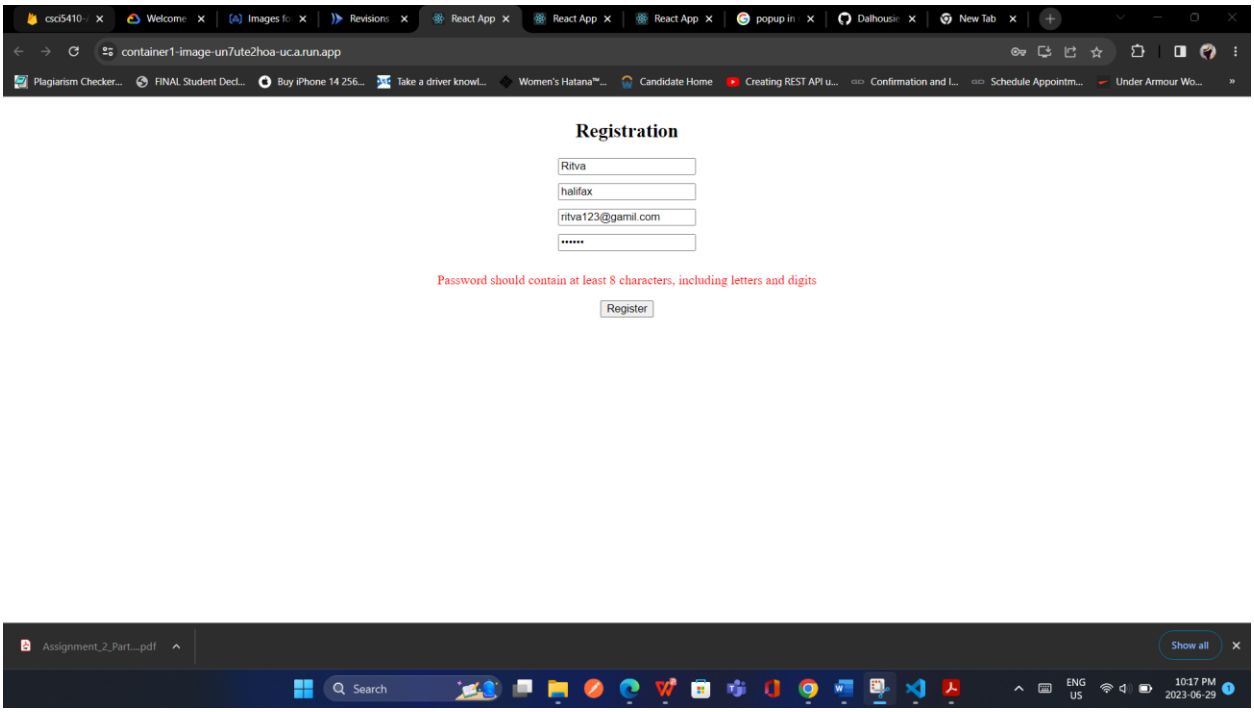

Figure 83 Validation for Registration page (password) [4]

Login react application:

Below method will validate email and password with database.

```
const LoginValidation = () => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [loggedIn, setLoggedIn] = useState(false);
  const [loginMessage, setLoginMessage] = useState('');

  const handleLogin = async (e) => {
    e.preventDefault();

    try {
      const db = firebase.firestore();
      const querySnapshot = await db
        .collection('Reg')
        .where('email', '==', email)
        .limit(1)
        .get();

      if (!querySnapshot.empty) {
        const user = querySnapshot.docs[0].data();
        if (user.password === password) {
          setLoggedIn(true);
          await updateUserStatus(querySnapshot.docs[0].id, 'online');
          setLoginMessage('Login successful!');
          window.location.href = 'https://localhost:3002/profile';
          return;
        }
      }

      setLoginMessage('Invalid email or password');
    } catch (error) {
      console.error('Error logging in:', error);
    }
  };
```

Figures 84,85 and 86 show all the validations of login page.

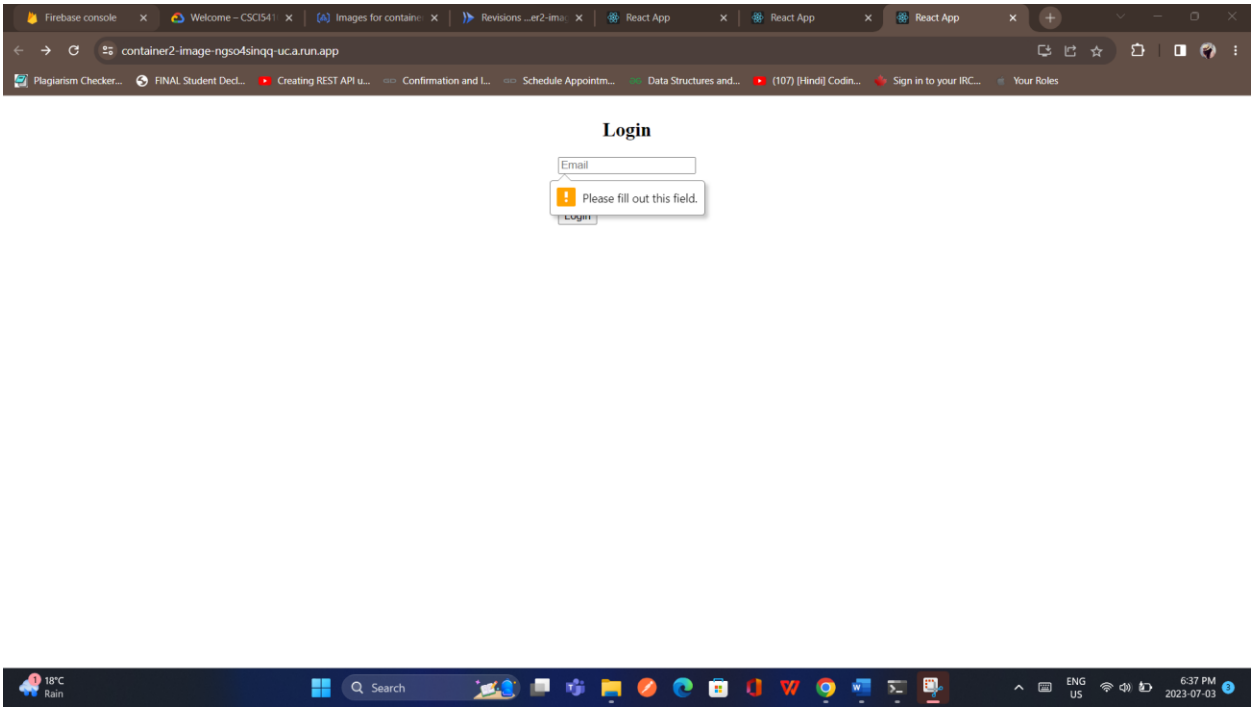

Figure 84 Validation for login page (email) [4]

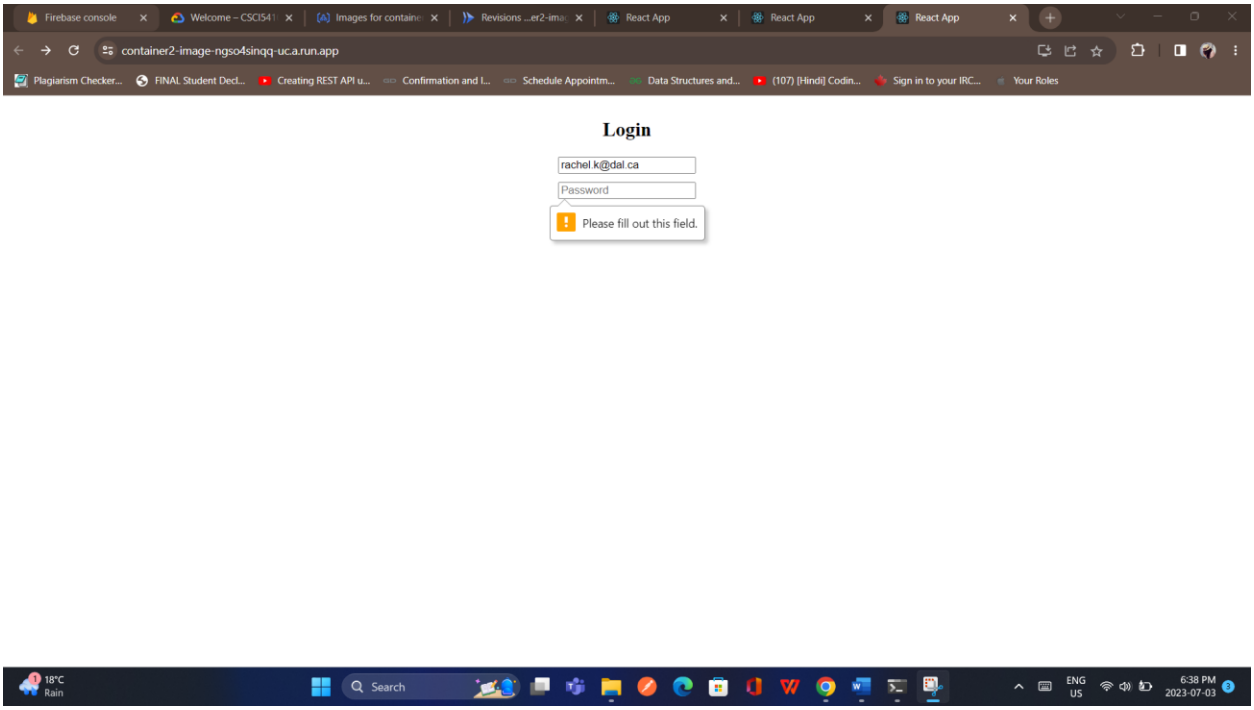

Figure 85 Validation for login page (password) [4]

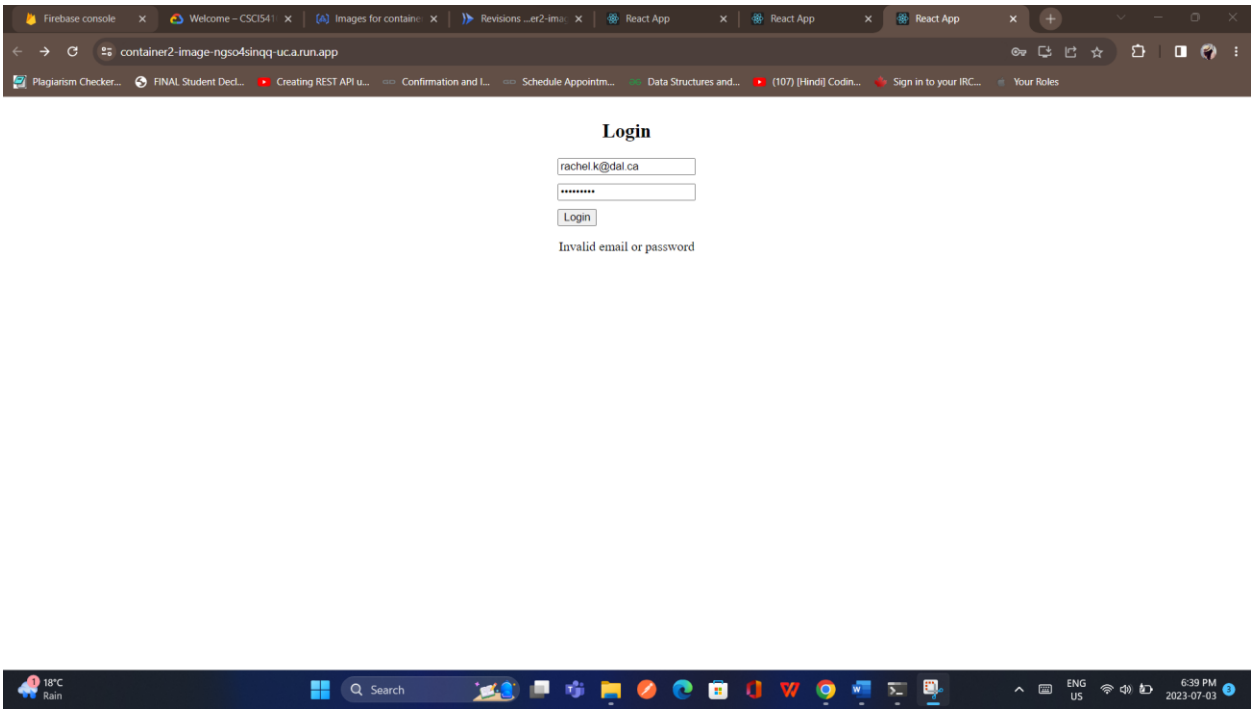

Figure 86 Test case of invalid email and password [4]

Below method will change the status of the logged in user from offline to online on successful login.

```
const updateUserStatus = async (docId, status) => {
   try {
     const db = firebase.firestore();
     await db.collection('state').doc(docId).update({ status });
   } catch (error) {
     console.error('Error updating user status:', error);
   }
 };
```

For login, email and password both fields are mentioned as required fields.

```
<input
            type="email"
            placeholder="Email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            required
            style={{ marginBottom: '10px' }}
```

```
            />
            <input
              type="password"
              placeholder="Password"
              value={password}
              onChange={(e) => setPassword(e.target.value)}
              required
              style={{ marginBottom: '10px' }}
            />
```

Profile react application:

Below method is written to fetch all the online users.

```
const LoginValidation = () => {
  const [loggedIn, setLoggedIn] = useState(false);
  const [userData, setUserData] = useState(null);

  useEffect(() => {
    const fetchUserData = async () => {
      const db = firebase.firestore();
      const onlineUsersSnapshot = await db.collection('state')
        .where('status', '==', 'online')
        .get();

      if (!onlineUsersSnapshot.empty) {
        const onlineUsersData = onlineUsersSnapshot.docs.map((doc) =>
doc.data());
        setUserData(onlineUsersData);
        setLoggedIn(true);
      }
    };

    fetchUserData();
  }, []);
```

Below method is used to handle lodout. On logout, below method will change the user's status from online to offline.

```
const handleLogout = async () => {
    try {
      await firebase.auth().signOut();
      await updateUserStatus('offline');

      setLoggedIn(false);
      setUserData(null);
    } catch (error) {
      console.error('Error logging out:', error);
    }
  };
```

Unit tests

Registration.test.js

```
import React from 'react';
import { render, screen, fireEvent } from '@testing-library/react';
import Registration from './Registration';

describe('Registration', () => {
  it('should show an error message if name contains invalid characters', () => {
    render(<Registration />);
    const nameInput = screen.getByPlaceholderText('Name');
    fireEvent.change(nameInput, { target: { value: '123' } });
    const errorMessage = screen.getByText('Name should contain only letters');
    expect(errorMessage).toBeInTheDocument();
  });

  it('should show an error message if email has an invalid format', () => {
    render(<Registration />);
    const emailInput = screen.getByPlaceholderText('Email');
    fireEvent.change(emailInput, { target: { value: 'invalidemail' } });
    const errorMessage = screen.getByText('Invalid email format');
    expect(errorMessage).toBeInTheDocument();
  });

  it('should show an error message if password is too short', () => {
    render(<Registration />);
    const passwordInput = screen.getByPlaceholderText('Password');
    fireEvent.change(passwordInput, { target: { value: 'pass' } });
    const errorMessage = screen.getByText('Password should contain at least 8
characters, including letters and digits');
    expect(errorMessage).toBeInTheDocument();
  });

  it('should submit the form if all inputs are valid', () => {
    render(<Registration />);
    const nameInput = screen.getByPlaceholderText('Name');
    const locationInput = screen.getByPlaceholderText('Location');
    const emailInput = screen.getByPlaceholderText('Email');
    const passwordInput = screen.getByPlaceholderText('Password');
    fireEvent.change(nameInput, { target: { value: 'rachel' } });
    fireEvent.change(locationInput, { target: { value: 'halifax' } });
    fireEvent.change(emailInput, { target: { value: 'rachel.p@dal.ca' } });
    fireEvent.change(passwordInput, { target: { value: 'rachel@123' } });
    const submitButton = screen.getByText('Register');
```

```
    fireEvent.click(submitButton);
  });
});
```

Figure 87 shows the all the tests cases of Registration.test.js and App.test.js passed successfully as which validates the name, password, location and email.

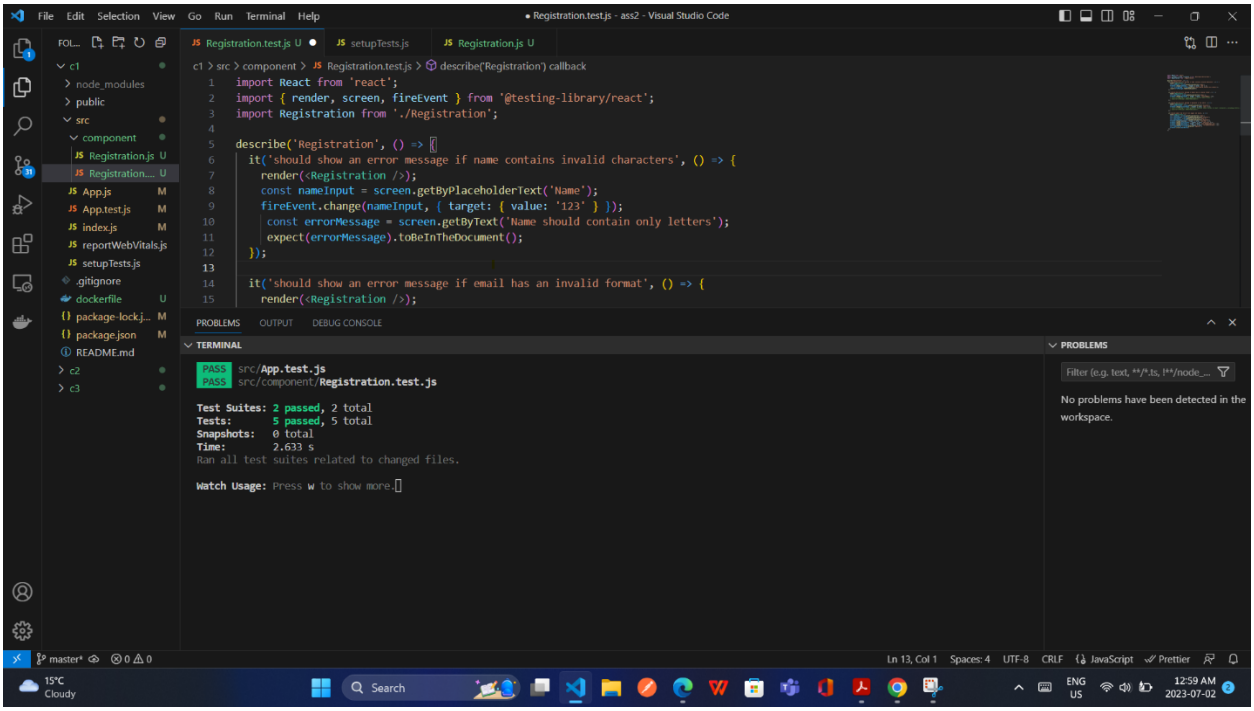

Figure 87 Unit tests for Registration page

Login.test.js

```
import React from 'react';
import { render, screen, fireEvent } from '@testing-library/react';
import LoginValidation from './Login';

describe('LoginValidation', () => {
  test('renders email and password input fields', () => {
    render(<LoginValidation />);

    const emailInput = screen.getByPlaceholderText('Email');
    const passwordInput = screen.getByPlaceholderText('Password');

    expect(emailInput).toBeInTheDocument();
    expect(passwordInput).toBeInTheDocument();
```

```
  });

  test('updates email and password state on input change', () => {
    render(<LoginValidation />);

    const emailInput = screen.getByPlaceholderText('Email');
    const passwordInput = screen.getByPlaceholderText('Password');

    fireEvent.change(emailInput, { target: { value: 'test@example.com' } });
    fireEvent.change(passwordInput, { target: { value: 'password123' } });

    expect(emailInput.value).toBe('test@example.com');
    expect(passwordInput.value).toBe('password123');
  });

  test('requires email and password fields to be filled', () => {
    render(<LoginValidation />);

    const loginButton = screen.getByRole('button', { name: 'Login' });

    fireEvent.click(loginButton);

    const emailErrorContainer = screen.getByTestId('email-error');
    const passwordErrorContainer = screen.getByTestId('password-error');

    expect(emailErrorContainer).toHaveClass('error-message');
    expect(passwordErrorContainer).toHaveClass('error-message');
  });



  test('submits the form when email and password fields are filled', () => {
    render(<LoginValidation />);

    const emailInput = screen.getByPlaceholderText('Email');
    const passwordInput = screen.getByPlaceholderText('Password');
    const loginButton = screen.getByRole('button', { name: 'Login' });

    fireEvent.change(emailInput, { target: { value: 'test@example.com' } });
    fireEvent.change(passwordInput, { target: { value: 'password123' } });
    fireEvent.click(loginButton);
  });
});
```

Figure 88 shows the all the tests cases of Login.test.js and App.test.js passed successfully as which validates the email and password for authentication.

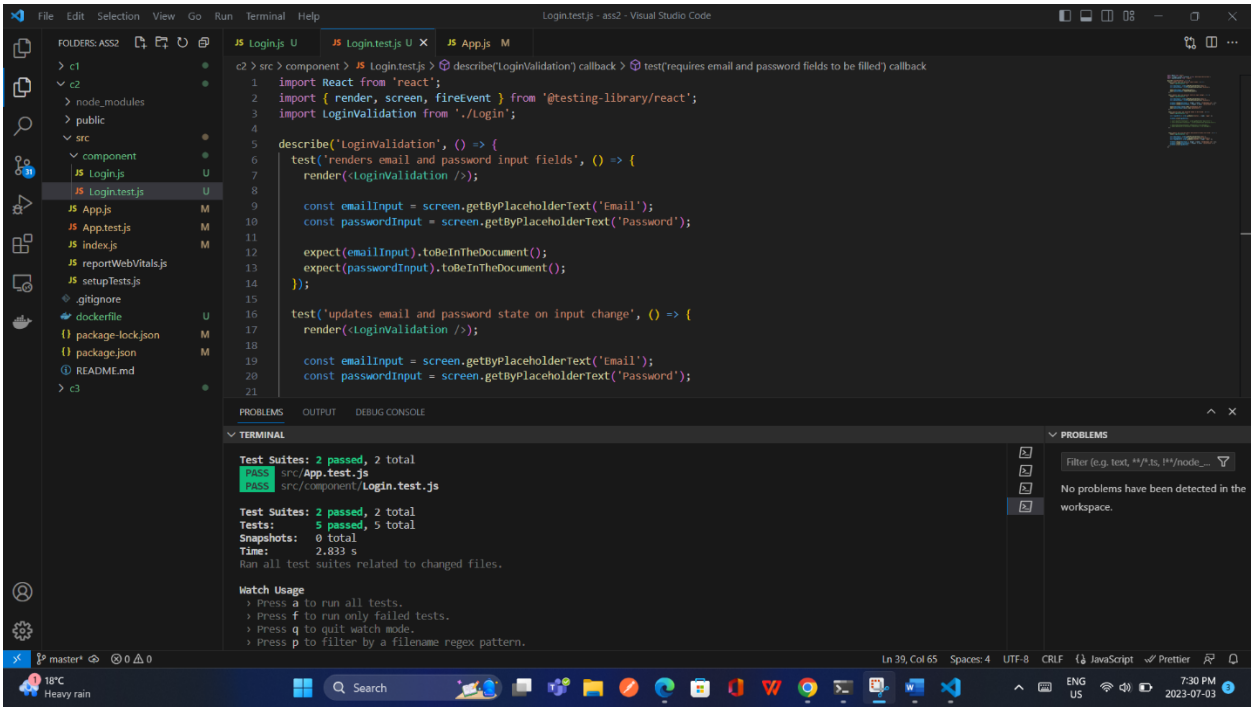

Figure 88 Unit tests for Login page

Summary

Serverless data processing allows the containerization of web applications, along with their required dependencies, utilizing the services offered by Google Cloud Platform, Amazon Web Services (AWS), and Microsoft Azure. In the context of this assignment, I delved into Google Cloud Run, Docker containerization, and the Artifact Registry to facilitate the building, deployment, and management of container images.

The utilization of Docker containers played a pivotal role in my application development process. Docker containers are standardized units of software that encapsulate code and all its dependencies, ensuring the application runs consistently and reliably across different computing environments. Docker container images, which are lightweight and standalone, include everything necessary to run an application, such as the code, runtime, system tools, libraries, and configurations. Thanks to Docker, I was able to build and push container images to the cloud's Artifact Registry securely, and effortlessly deploy them on the cloud platform.

Cloud Artifact Registry provides a secure and private repository to store and manage my Docker container images, ensuring that only authorized users and services have access. Secondly, Cloud Artifact Registry offers versioning capabilities, allowing me to maintain a history of changes and easily manage different versions of my container images. Additionally, it seamlessly integrates with Google Cloud Run, simplifying the deployment process and enabling me to easily pull the required container images for deployment. Lastly, Cloud Artifact Registry is highly scalable and performs well even with large-scale repositories, ensuring efficient storage and retrieval of container images.

Google Cloud Run provides a serverless execution environment for my containerized applications, allowing me to focus on application development without the burden of infrastructure management. Secondly, Cloud Run offers automatic scaling, ensuring my applications can handle varying levels of traffic efficiently. Additionally, Cloud Run seamlessly integrates with other Google Cloud services, enabling easy integration with services such as Artifact Registry for streamlined deployment workflows. Lastly, Cloud Run supports a wide range of programming languages and frameworks, providing flexibility and compatibility for my application development needs.

References

[1] Docker, “Enterprise Application Container Platform | Docker,” Docker, 2018. [Online]. Available: https://www.docker.com/. [Accessed 30 June 2023].

[2] “Cloud Firestore | Store and sync app data at global scale,” Firebase. [Online]. Available: https://firebase.google.com/products/firestore. [Accessed 30 June 2023].

[3] “Installing Google Cloud SDK | Cloud SDK Documentation,” Google Cloud. [Online]. Available: https://cloud.google.com/sdk/docs/install.. [Accessed 30 June 2023].

[4] “Cloud Run: Container to production in seconds,” Google Cloud. [Online]. Available: https://cloud.google.com/run. [Accessed 30 June 2023].

[5] “Artifact Registry,” Google Cloud. [Online]. Available: https://cloud.google.com/artifact-registry. [Accessed 30 June 2023].

[6] “Fragments – React,” legacy.reactjs.org. [Online]. Available: https://legacy.reactjs.org/docs/fragments.html. [Accessed 30 June 2023].

[7] Google, “Cloud Computing Services | Google Cloud,” Google Cloud, 2019. [Online]. Available: https://cloud.google.com/. [Accessed 30 June 2023].

[8] “Getting Started – React,” reactjs.org. [Online]. Available: https://legacy.reactjs.org/docs/getting-started.html. [Accessed 30 June 2023].