CSCI 5410

Serverless Data Processing (Summer 2023)

MASTER OF APPLIED COMPUTER SCIENCE

Assignment-2 (Part B): Containerized Application

Name: **Jainil Sevalia** | Banner Id : **B00925445** | Email:jn498899@dal.ca

GitLab Link : https://git.cs.dal.ca/sevalia/csci5410-summer-23-b00925445/-/tree/A2?ref_type=heads

Table of Figures:

B. Your database should contain only 2 collections. One collection is "Reg" to contain registration data (Name, Password, Email, Location), another collection is "state" to contain user state (online, offline, timestamp etc.) information.

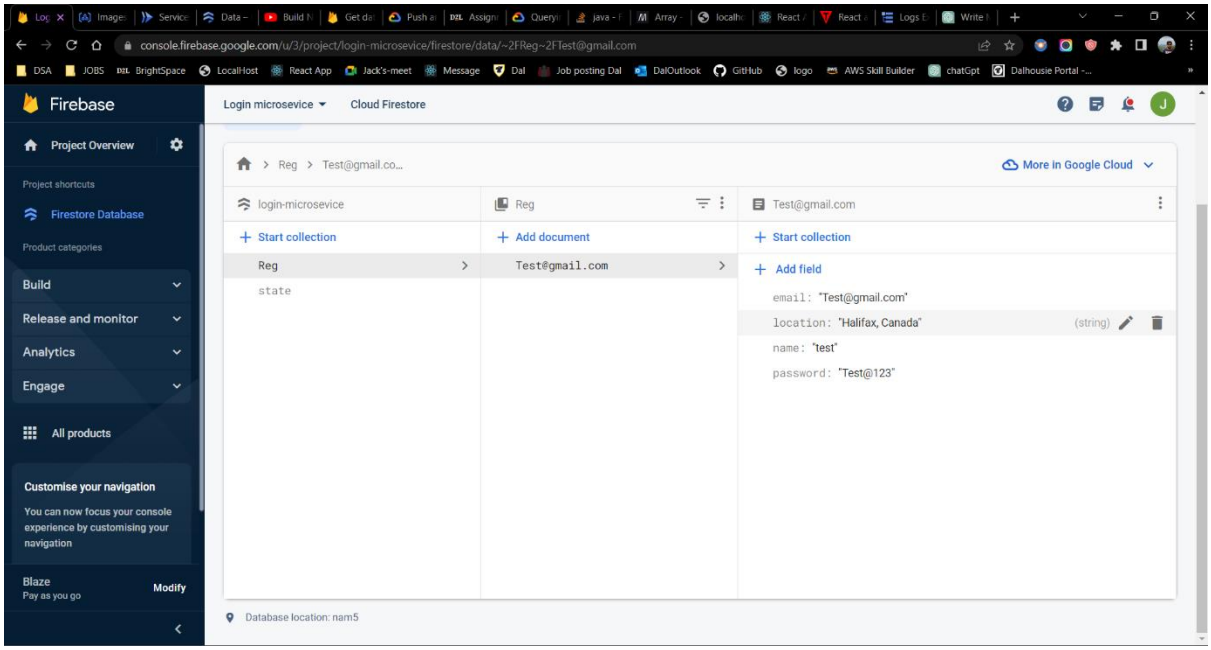

Figure 1 : Reg collection created on Fire store.

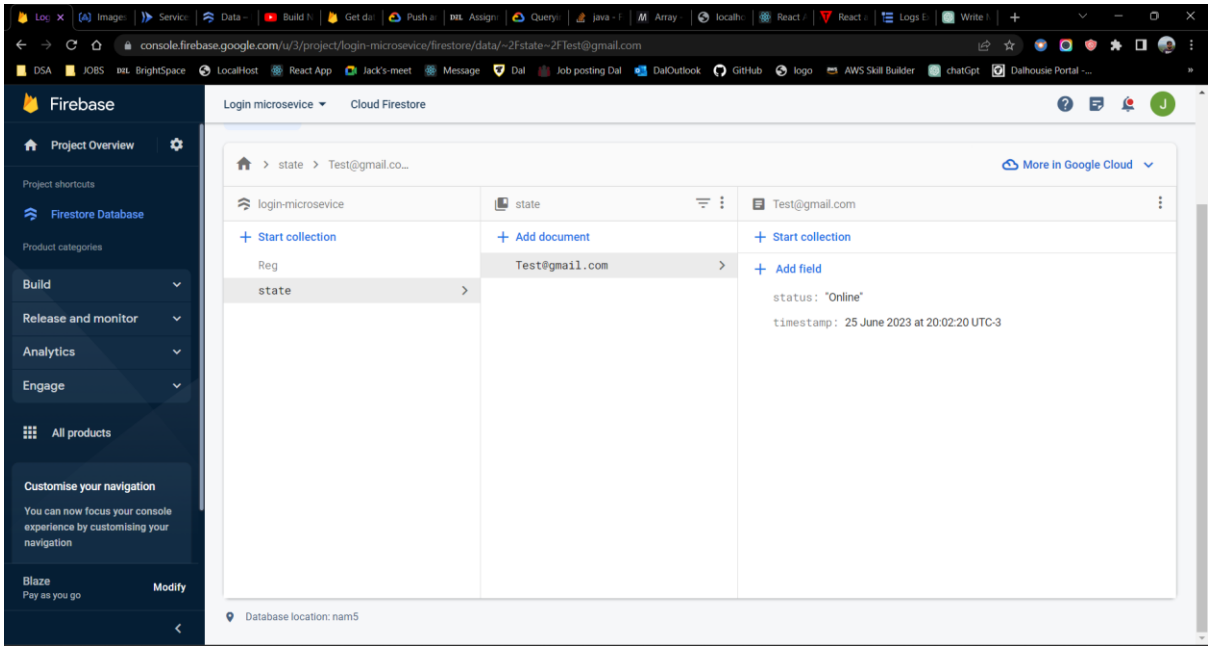

Figure 2: state collection created on fire store.

C. Code and the required dependencies of Container #1 are responsible for accepting registration details from frontend and store it in backend database.
Container – 1:

- Business logic/ Function for Registration:

```
app.post(`/Register`, async (req, res) => {
  try {
    const email = req.body.email;
    if (email == "") {
      return res.status(400).json("Email is required.");
    }

    if (!email.includes("@")) {
      return res.status(400).json("Invalid Email Address.");
    }
    const userInfo = {
      name: req.body.name,
      password: req.body.password,
      email: req.body.email,
      location: req.body.location,
    };

    const RegRef = firestore.collection("Reg").doc(`${email}`);

    const regSnapshot = await RegRef.get();

    if (regSnapshot.exists) {
      return res.status(409).json("Email already registered.");
    }
    const document = firestore.doc(`Reg/${email}`);
    const response = document.set(userInfo);
    return res.status(201).json(userInfo);
  } catch (e) {
    return res.send(e);
  }
});
```

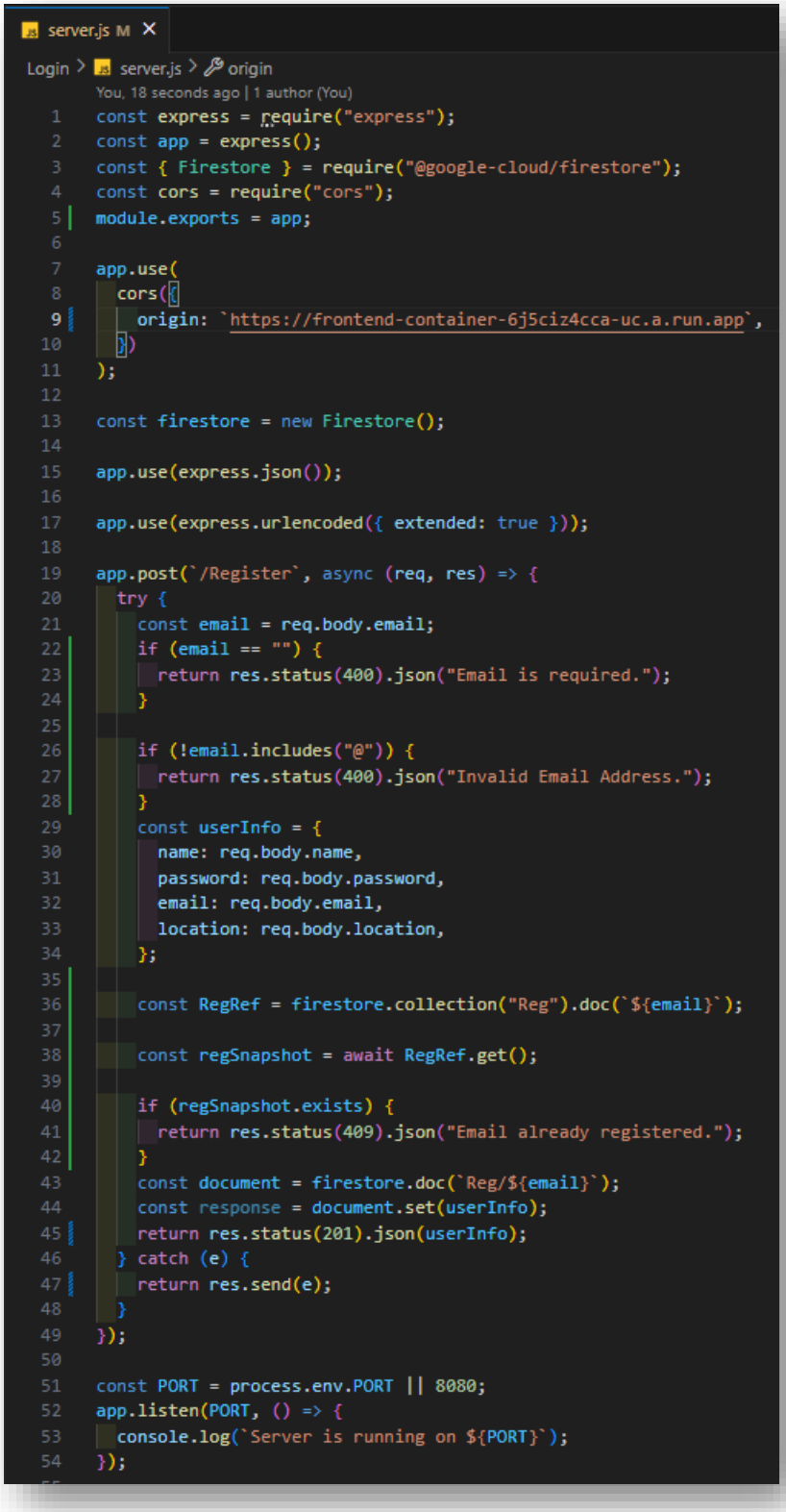

Figure 3 : Application business logic Code for container -1

- Build docker image and tagged it with specific name before pushing it on artifact registry.

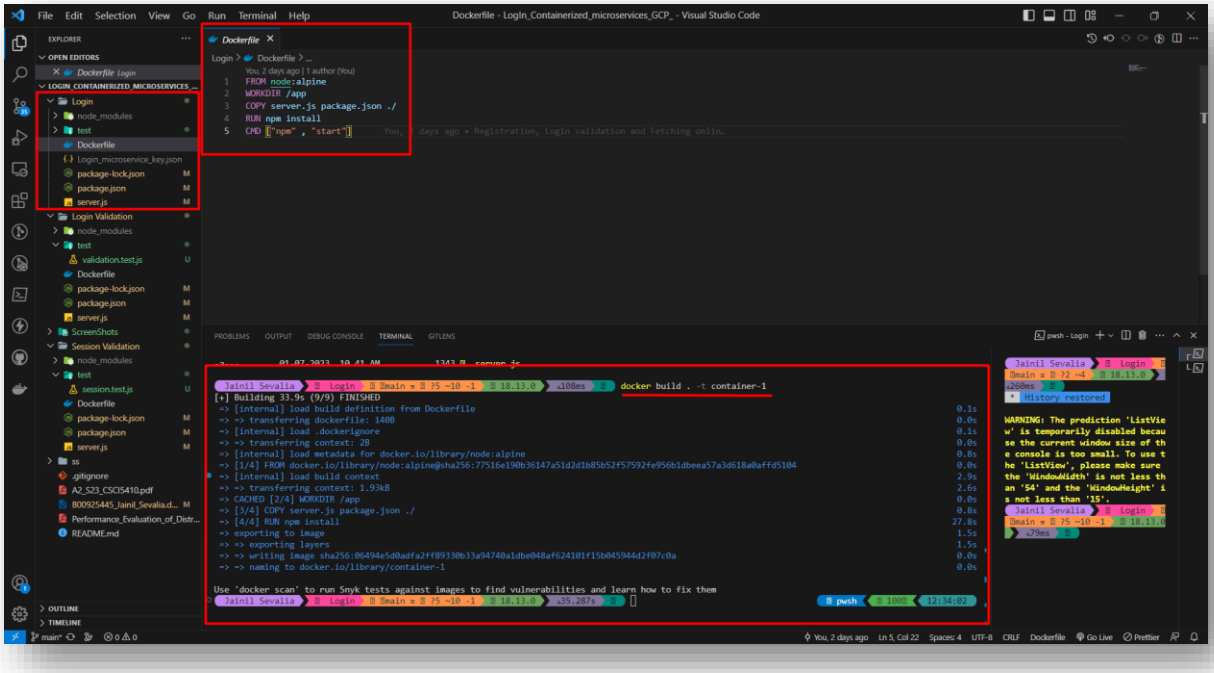

Figure 4 : built docker image and tagged it.

- Docker image pushed using command mentioned in below image.

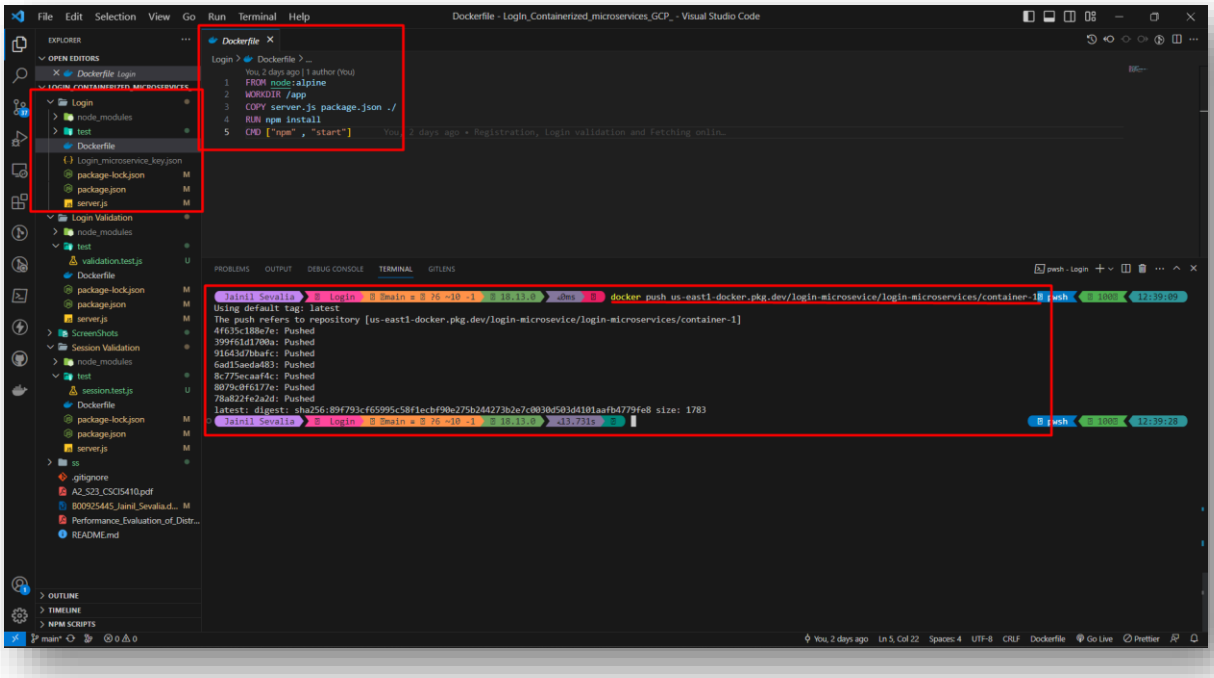

Figure 5 : Docker image pushed on GCP Artifact Registry.

- GCP Artifact Registry shows the pushed image successfully.

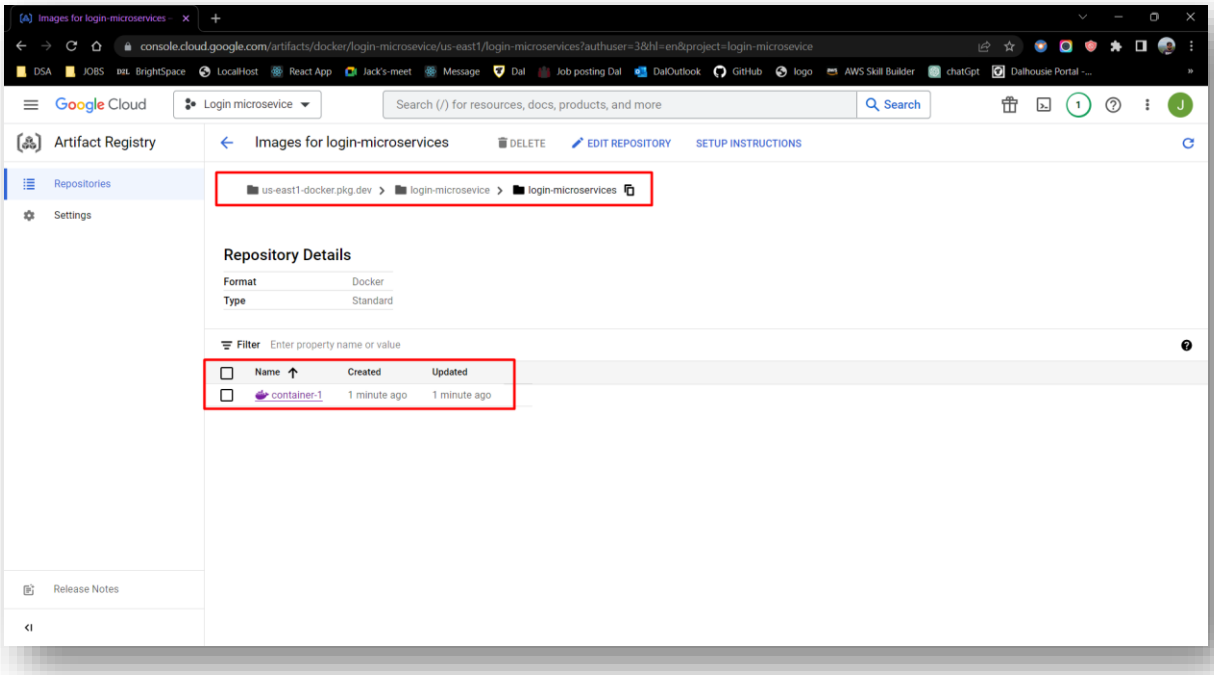

Figure 6 : Artifact Registry console GCP.

- Created New Service in Cloud Run to Run the container 1.

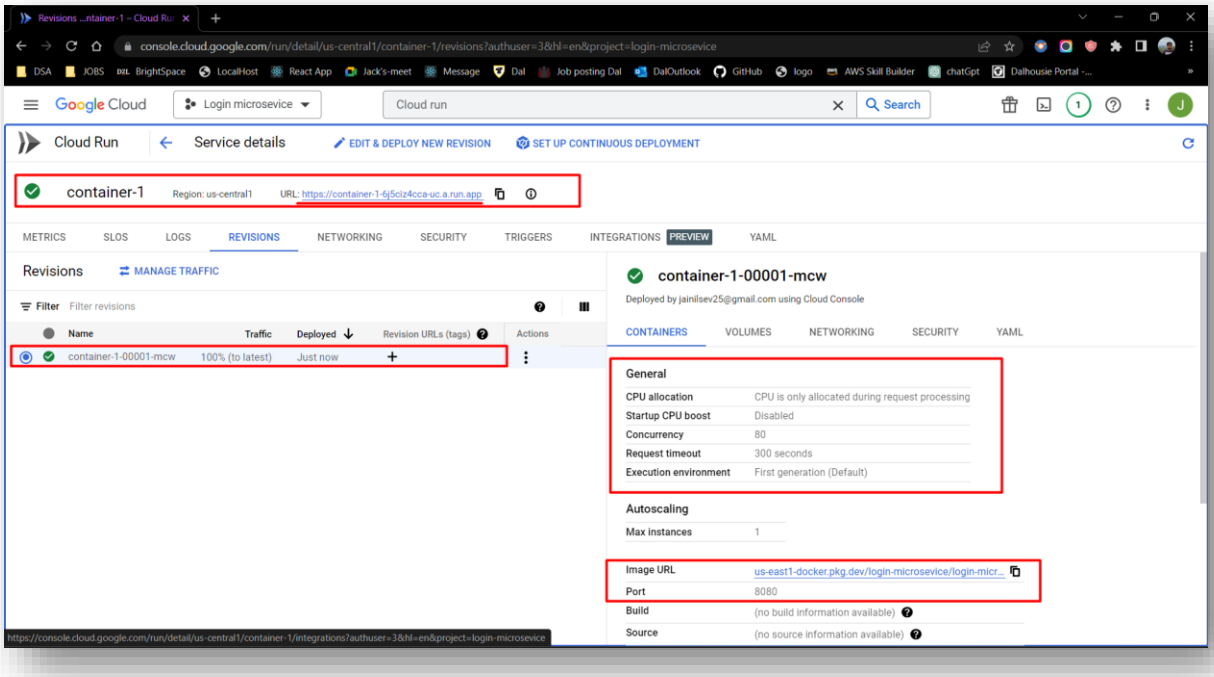

Figure 7 : Container-1 Running on Cloud Run service.

- Testing the running container by sending POST request form postman to that container.

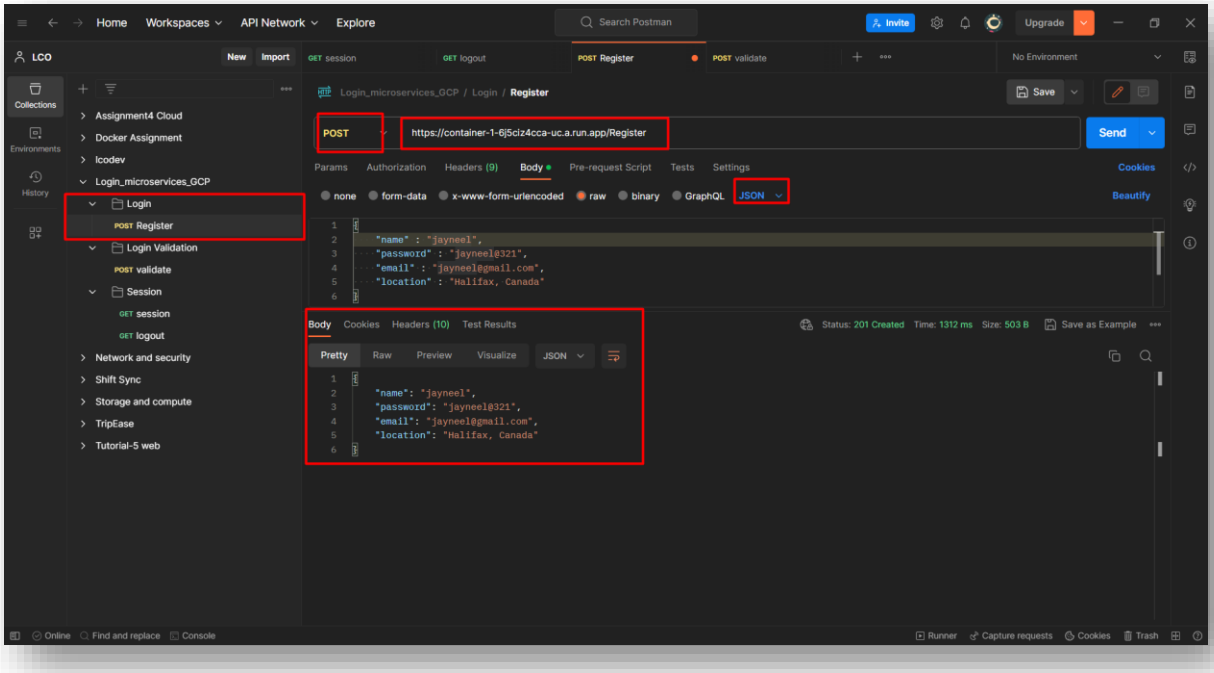

Figure 8 : Postman Sending request to container to store the User Data.

- Successfully got data in firestore.

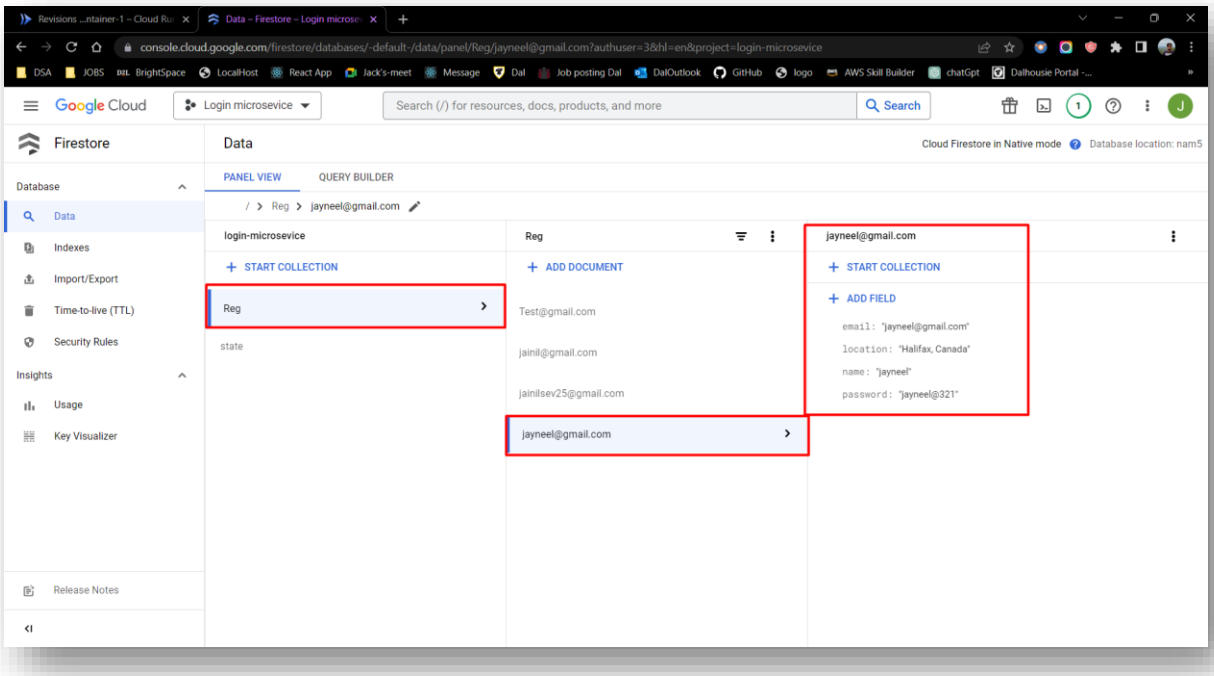

Figure 9 : Container-1 running successfully on GCP and getting all the data in the Firestore.

- Handled Edge cases for Registration microservices.
- Null email handling.

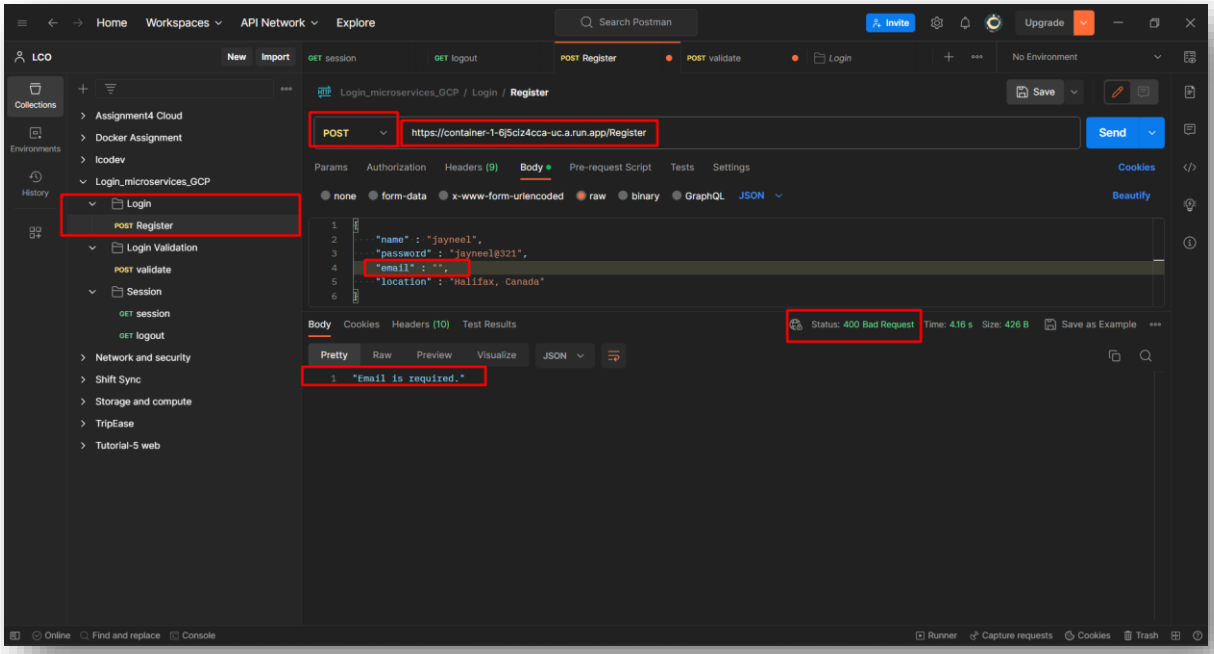

Figure 10 : Null email edge case for registration.

- Invalid form of email handling.

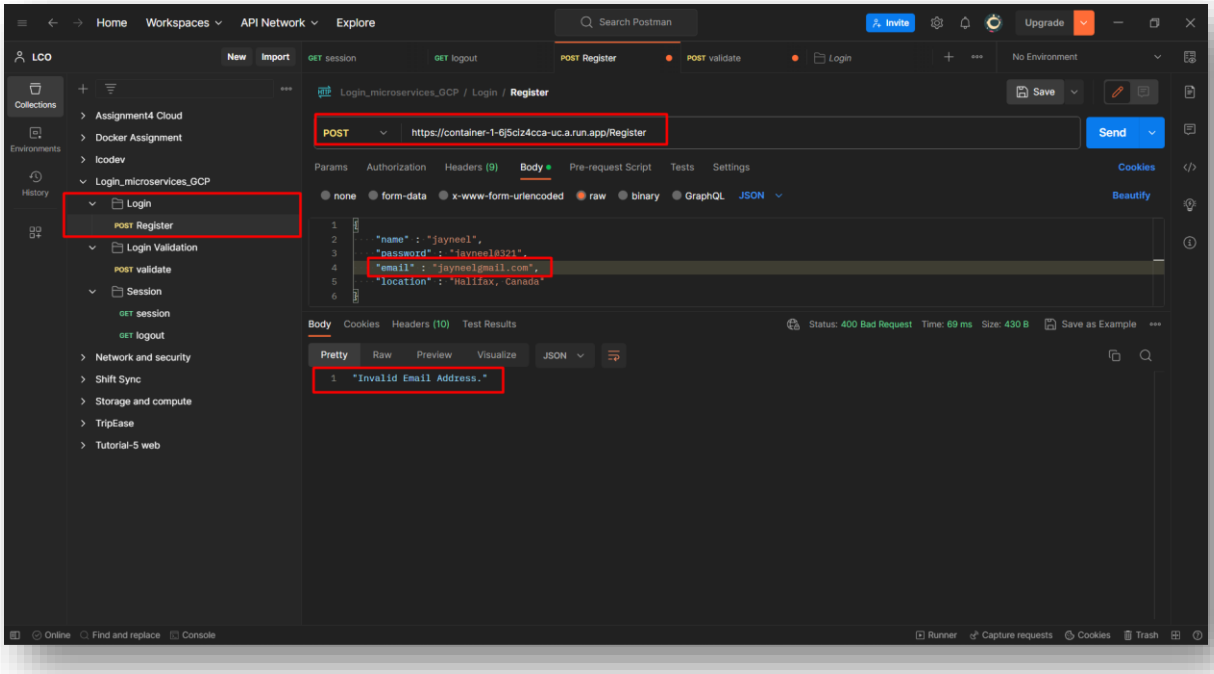

Figure 11 : Invalid form of email handling.

- Already registered email handling.

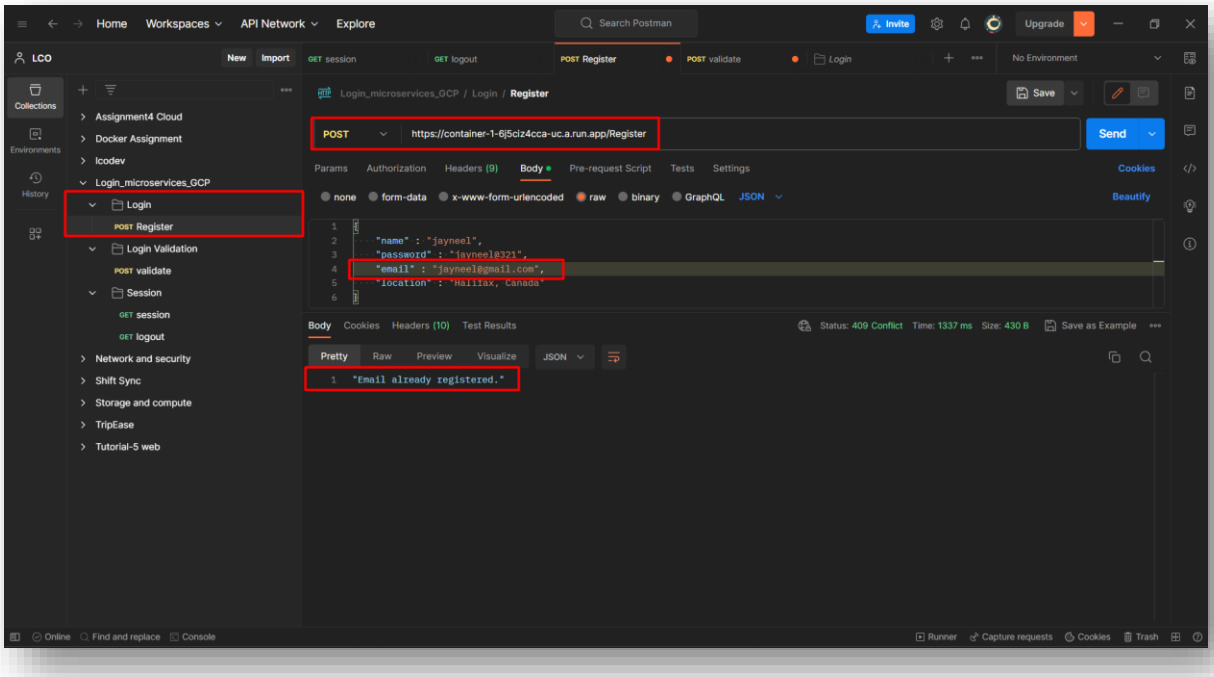

Figure 12: Already email registered email handling.

H (For container 1). Test Cases for /Registration endpoint.

- Successfully registered user.
- Empty email. (Email is required).
- Email validation (Invalid email).
- User already exists with email. (Email already registered).

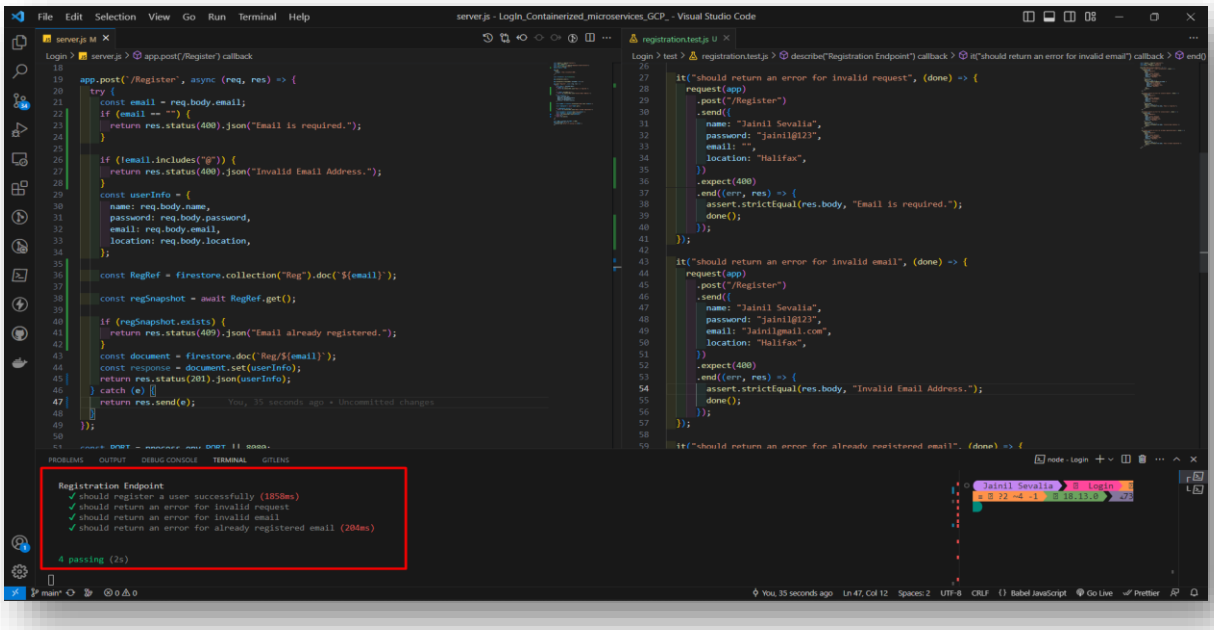

Figure 13 : Test cases for Container-1.

D. Container – 2:

- Business logic/ Function for Login:

```
app.post(`/validate`, async (req, res) => {
  const email = req.body.email;
  const password = req.body.password;
  try {
    if (!email || !password) {
      res.status(404).send({ Error: "User Null" });
    }
    const regRef = firestore.collection("Reg").doc(email);
    const doc = await regRef.get();
    let isLoggedIn = false;
    if (!doc.exists) {
      res.status(404).send({ Error: "User Not Found" });
    } else if (email === doc.data().email && password ===
doc.data().password) {
      isLoggedIn = true;
    } else {
      res.status(200).send({ Error: "Credential not matched" });
    }
    if (isLoggedIn) {
      const status = {
        status: "Online",
        timestamp: new Date(),
      };
      const document = firestore.doc(`state/${email}`);
      const response = await document.set(status);
      res.status(200).json({ ...doc.data(), ...status });
    }  } catch (e) {
}});
```

- Application business logic Code for container -2

server.js M

Login Validation > server.js > origin

You, 2 seconds ago | 1 author (You)

```
const express = require("express");
const app = express();
const { Firestore } = require("@google-cloud/firestore");
module.exports = app;

const cors = require("cors");

app.use(
  cors({
    origin: `https://frontend-container-6j5ciz4cca-uc.a.run.app`,
  })
);
const firestore = new Firestore();
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

app.post(`/validate`, async (req, res) => {
  const email = req.body.email;
  const password = req.body.password;
  try {
    if (!email || !password) {
      res.status(404).send({ Error: "User Null" });
    }

    const regRef = firestore.collection("Reg").doc(email);
    const doc = await regRef.get();
    let isLoggedIn = false;

    if (!doc.exists) {
      res.status(404).send({ Error: "User Not Found" });
    } else if (email === doc.data().email && password === doc.data().password) {
      isLoggedIn = true;
    } else {
      res.status(200).send({ Error: "Credential not matched" });
    }

    if (isLoggedIn) {
      const status = {
        status: "Online",
        timestamp: new Date(),
      };
      const document = firestore.doc(`state/${email}`);
      const response = await document.set(status);
      res.status(200).json({ ...doc.data(), ...status });
    }
  } catch (e) {
    console.log(e);
  }
});

const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on ${PORT}`);
});
```

Figure 14 : Application business logic Code for container -2.

- Build docker image and tagged it with specific name before pushing it on artifact registry.

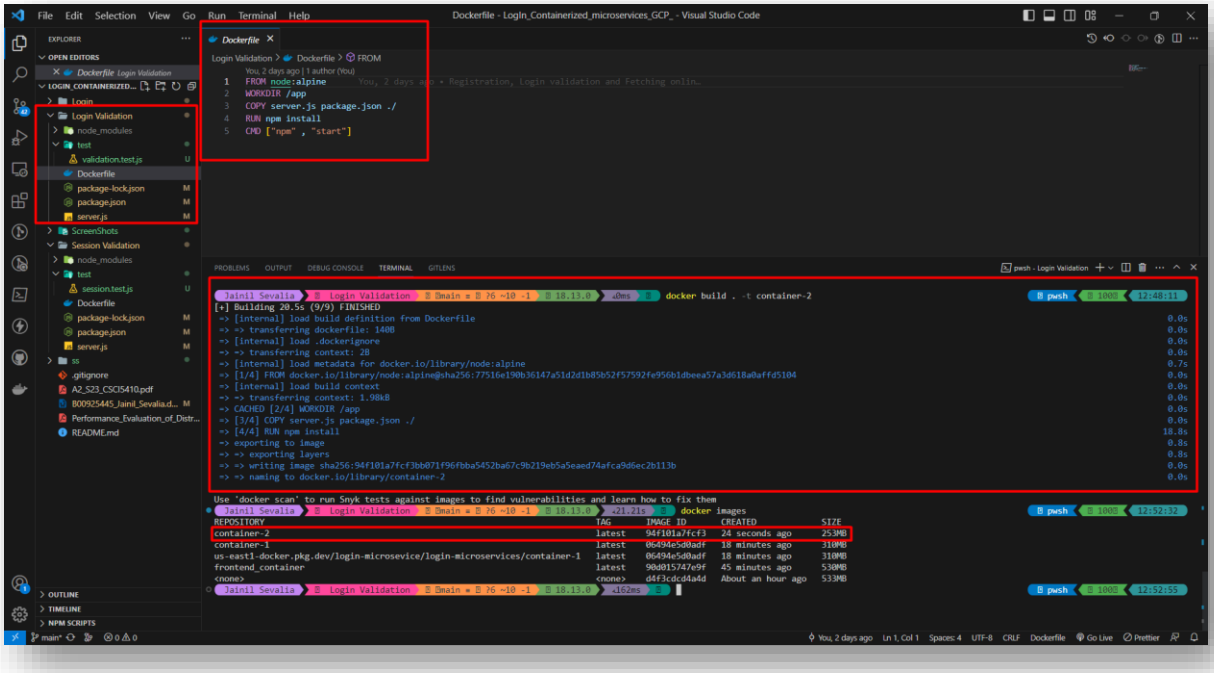

Figure 15 : Build docker image for container 2.

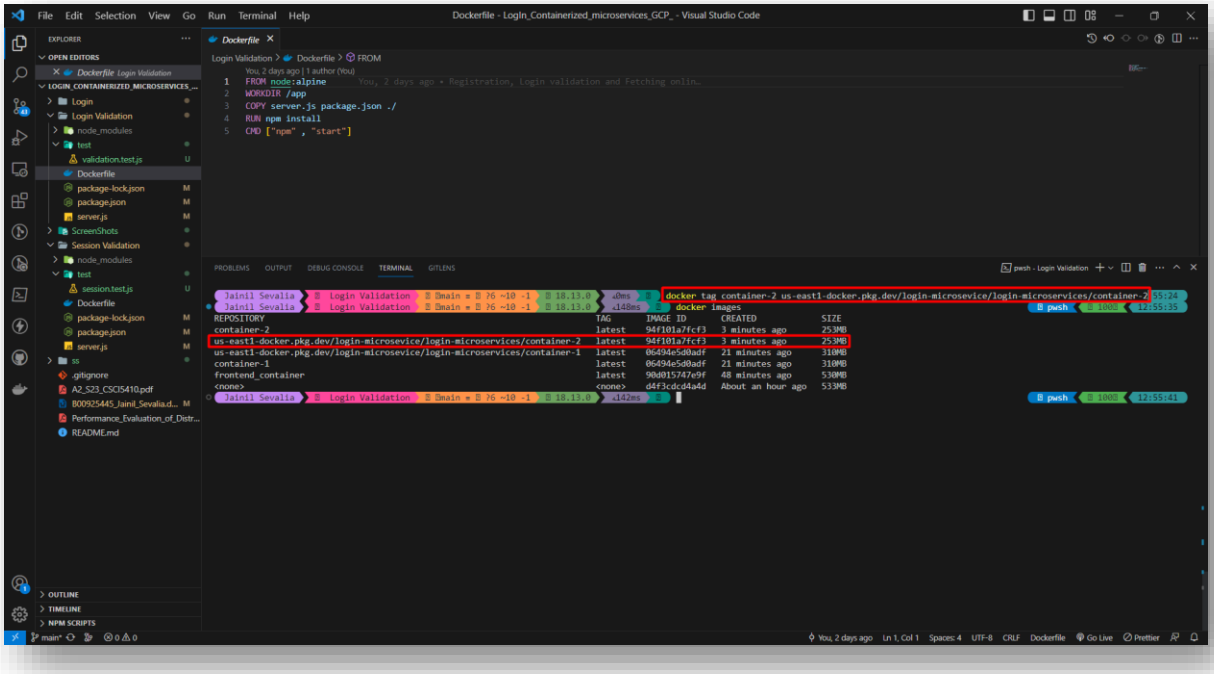

Figure 16 : Docker image tagged with project name to push on Artifact Registry.

- Docker image pushed using command mentioned in below image.

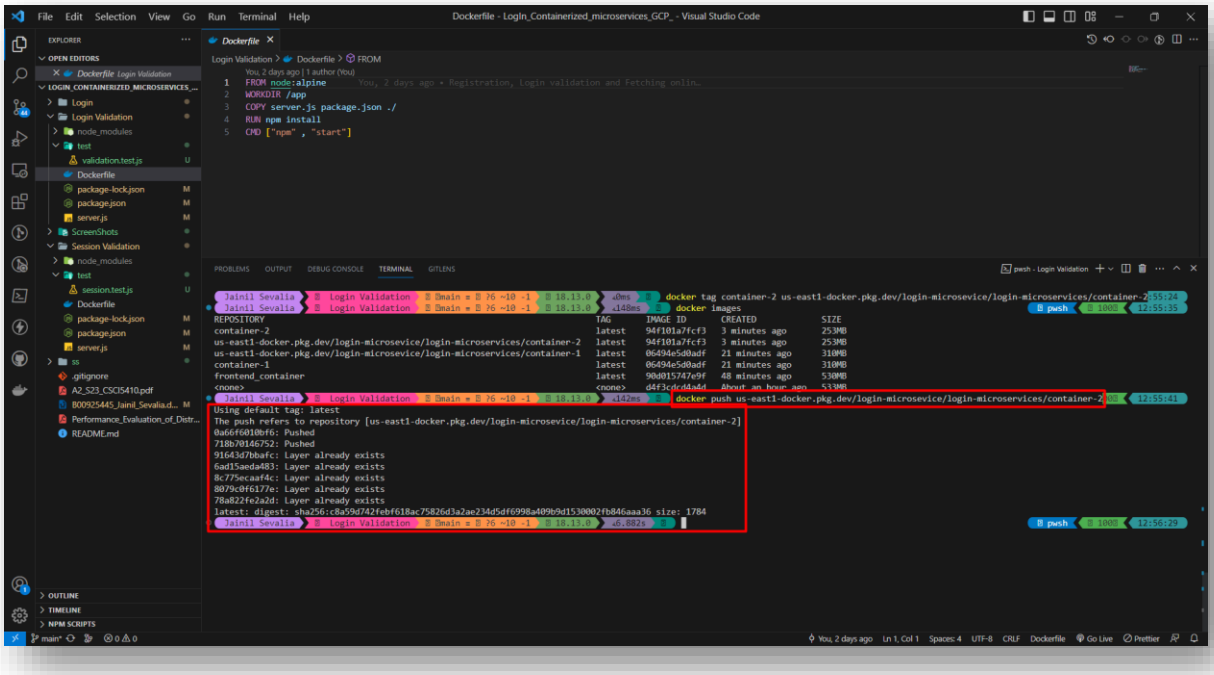

Figure 17 : Docker Image build and pushed for container-2

- GCP Artifact Registry shows the pushed image successfully.

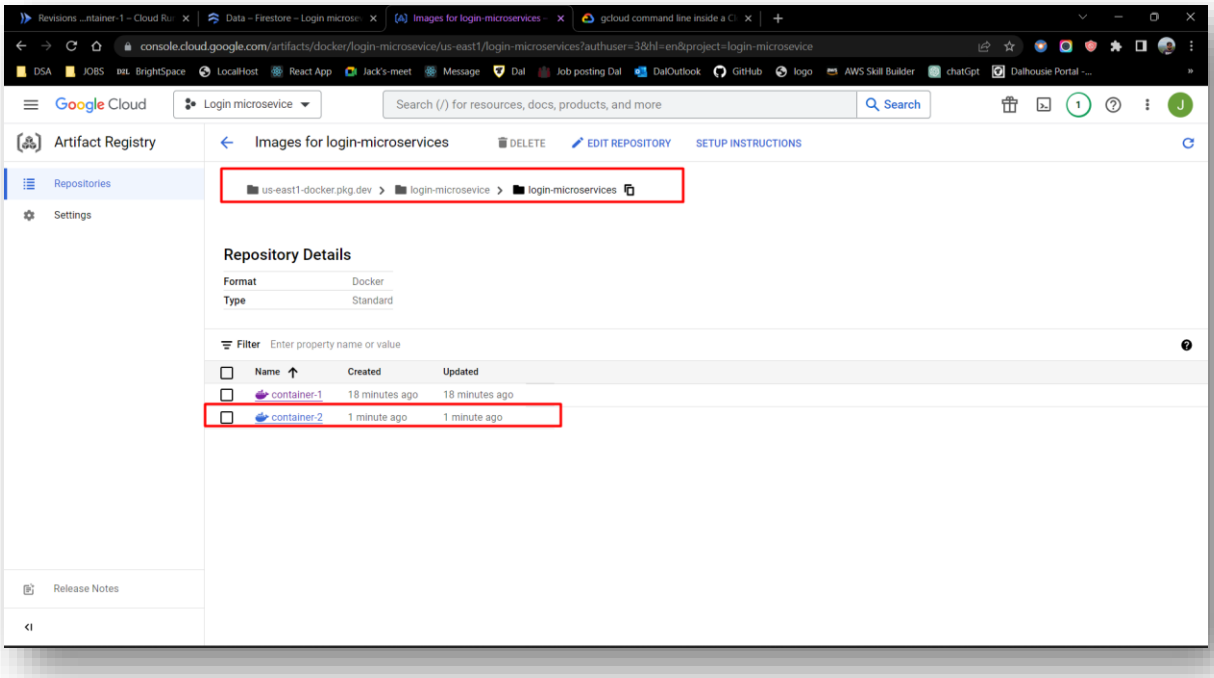

Figure 18: Artifact Registry GCP console.

- Creating service for container 2 on Cloud Run.

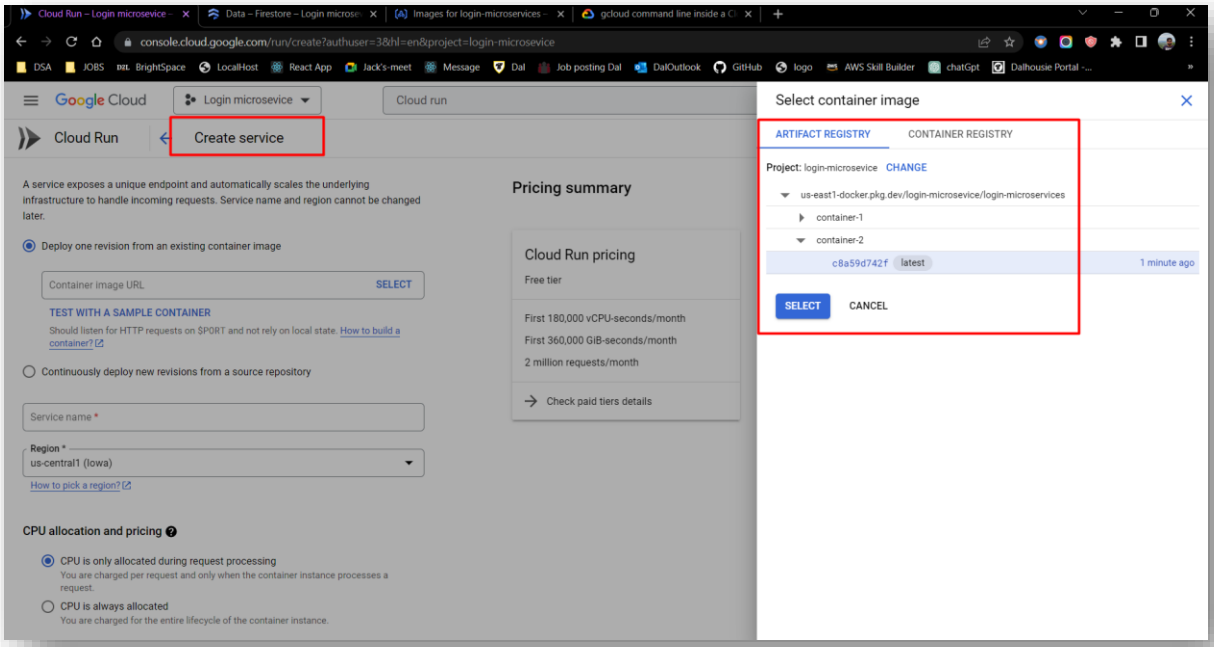

Figure 19 : Creating service for container 2.

- Container is running on the GCP cloud run service, which is shown below.

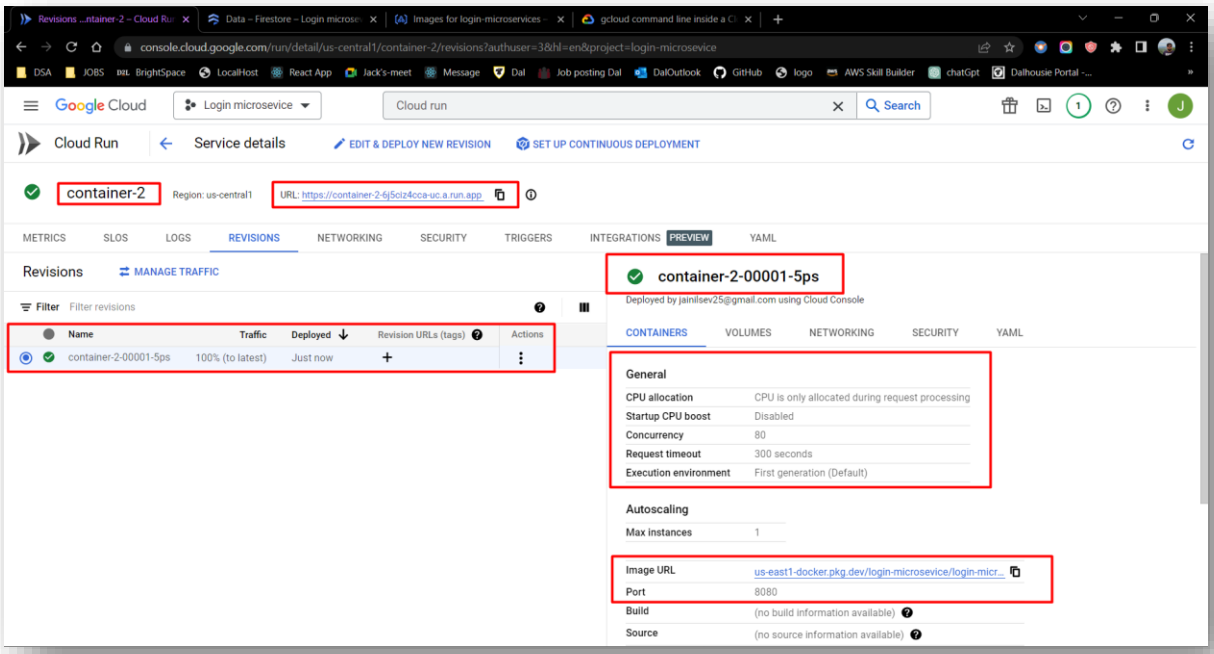

Figure 20 : Container 2 is running on Cloud Run service.

- Testing the container by sending request to container using postman.

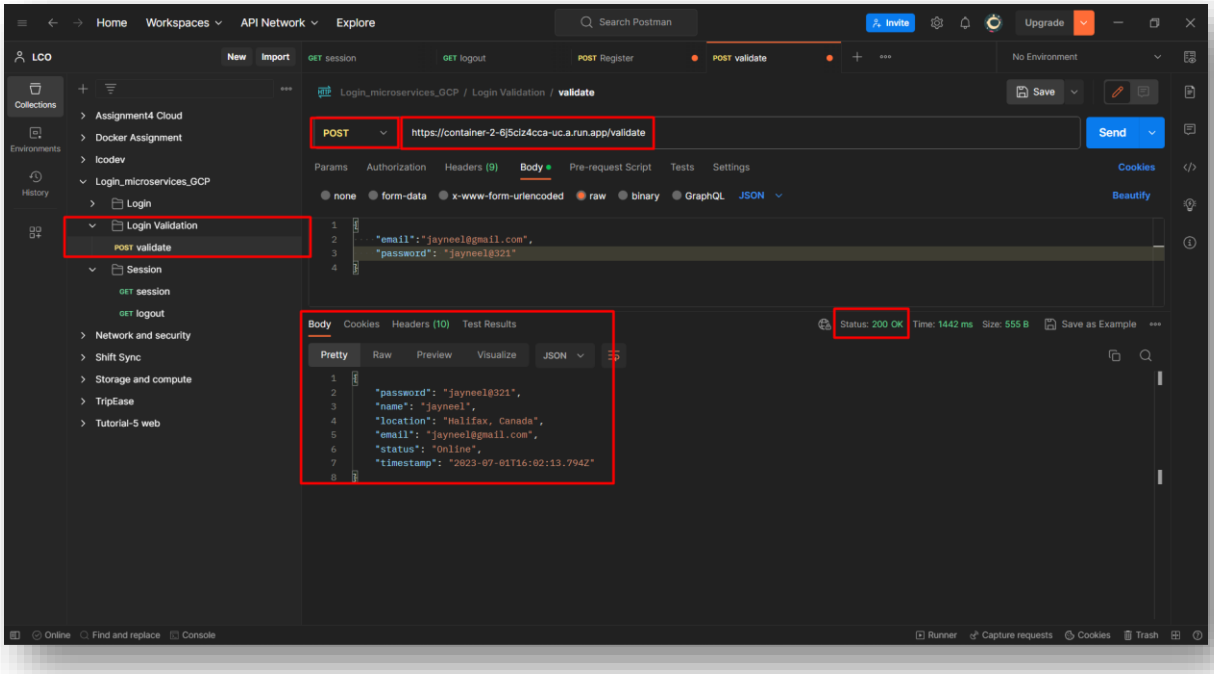

Figure 21: Postman Sending request to container to validate the User Data.

- User Status updated in firestore.

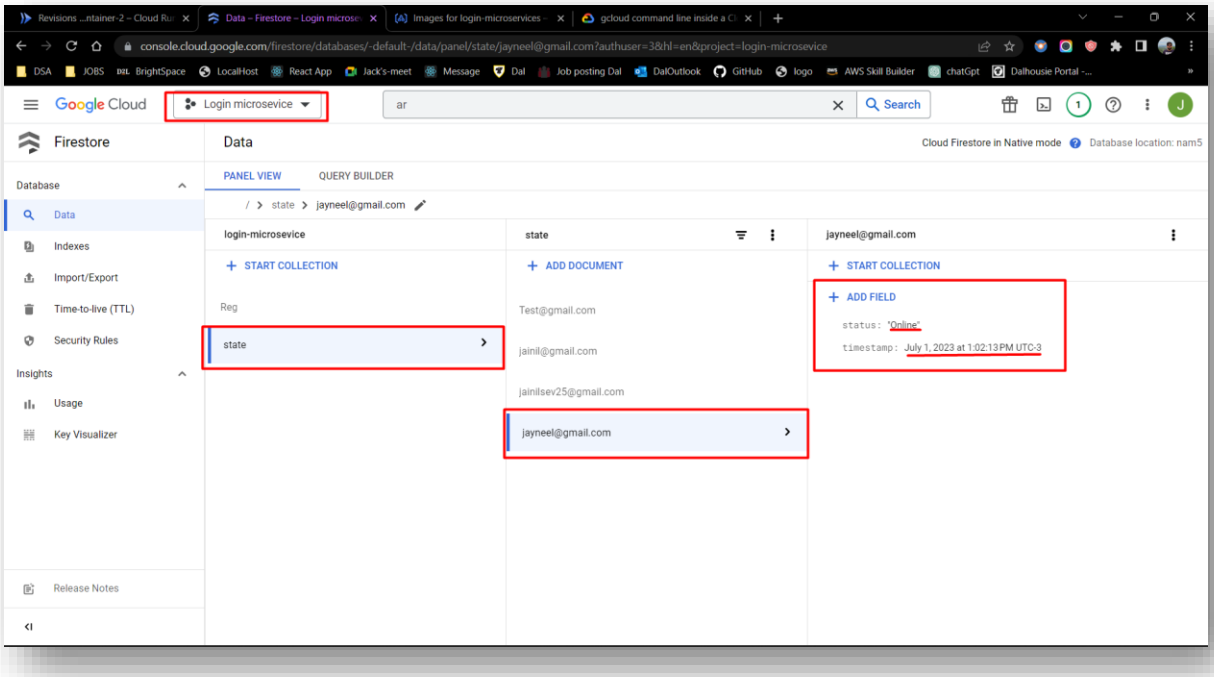

Figure 22 : User status updated in firestore.

- **Edge cases handling.**
- Null credential field passed in request payload.

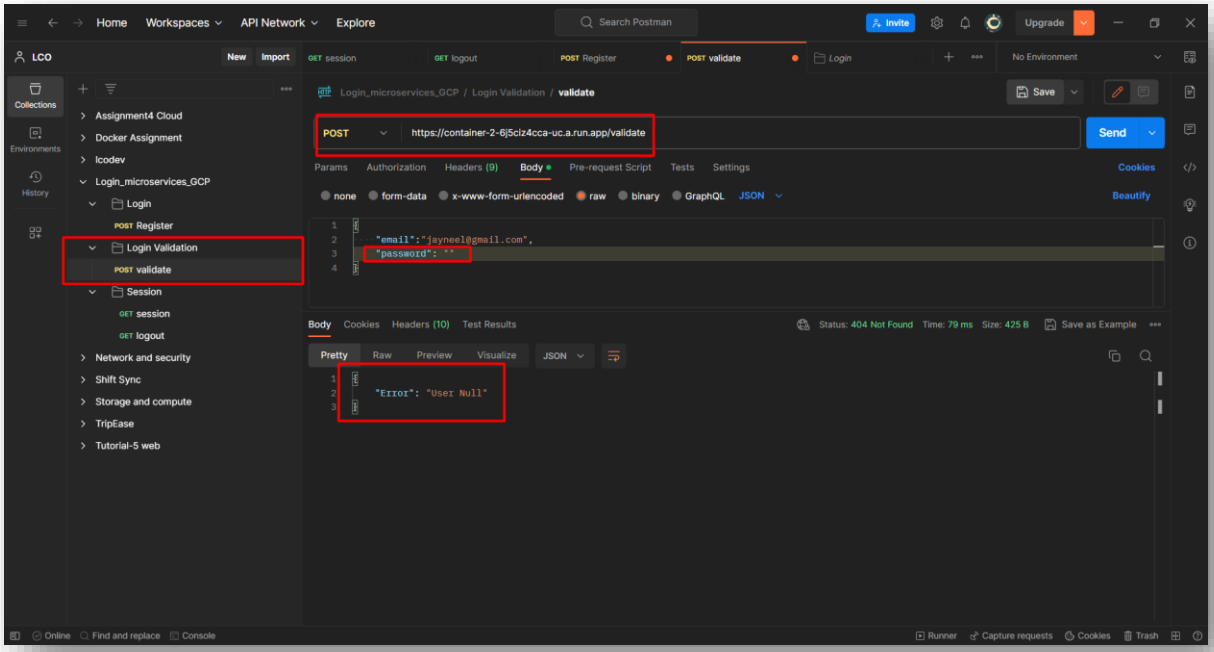

Figure 23 : Null credential field passed in request payload.

- User tried to login using not registered user.

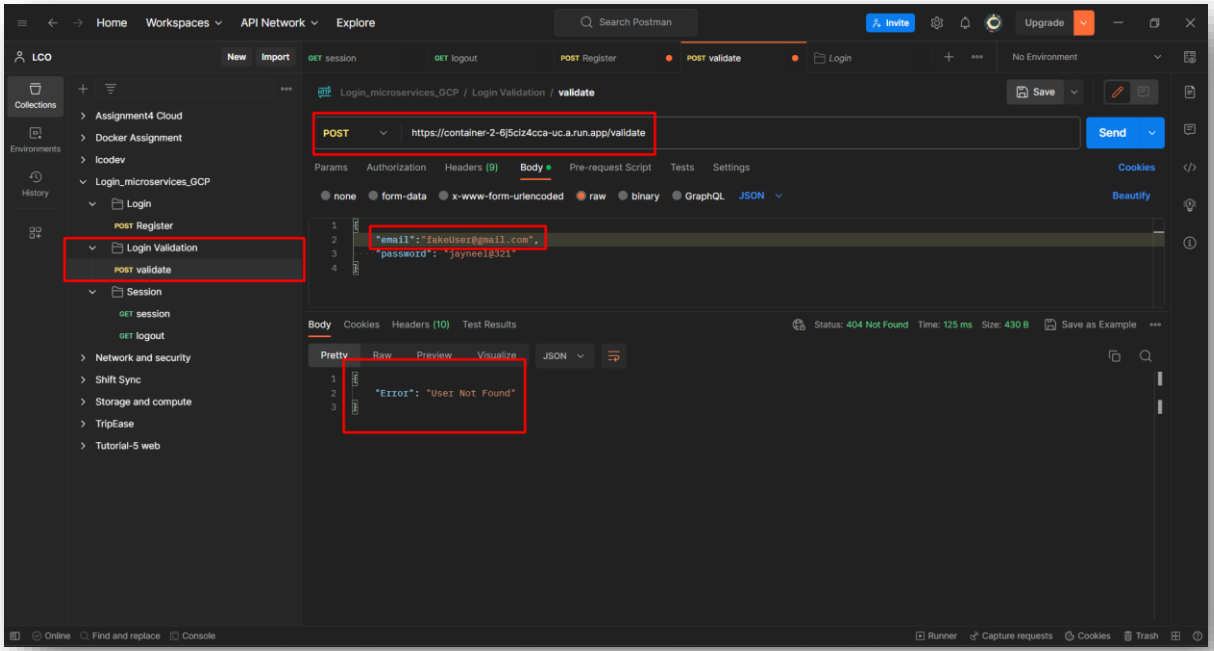

Figure 24 : non-Registered user tried to login.

H (For Container -2). Test Cases for /Login endpoint.

- Successfully Logged In user.
- Empty Credentials. (Email and Password is required).
- Valid Credentials (Incorrect Email or Password).
- Non-existing user. (Invalid Email).

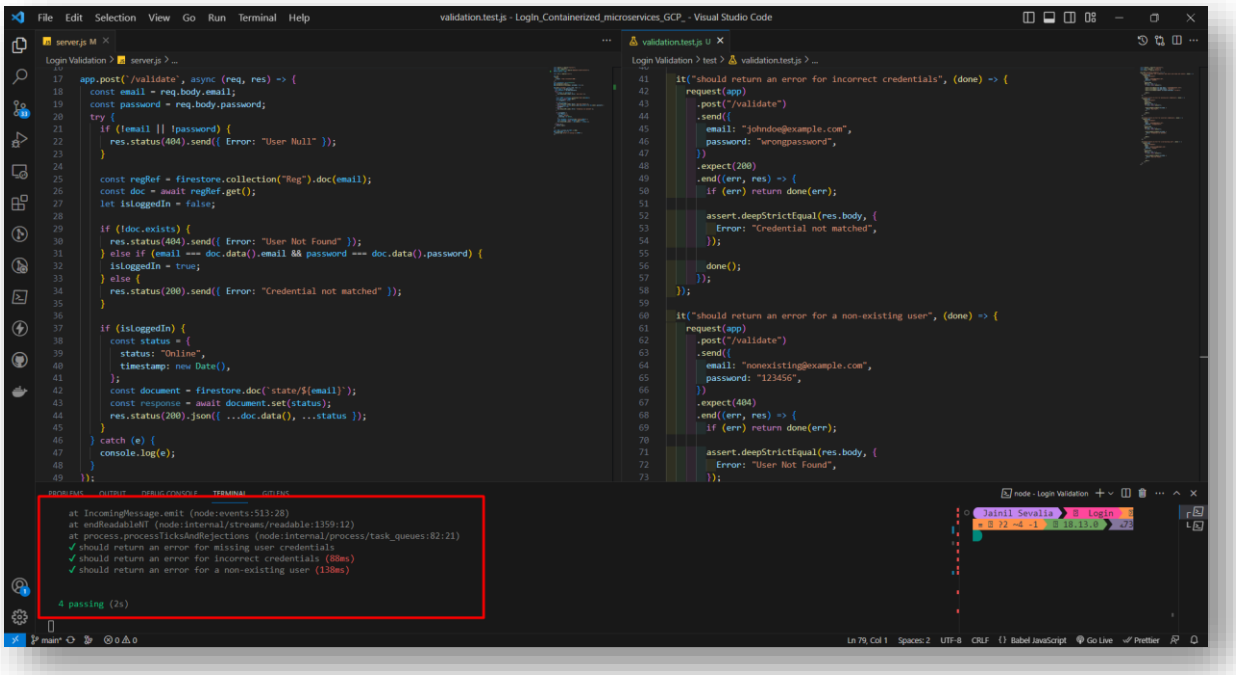

Figure 25 : Test Cases for container 2.

E. Container – 3:

- Business logic/ Function for Session and logout:

```
app.get(`/session`, async (req, res) => {
  const stateRef = firestore.collection("state");

  const allOnlineUsers = await stateRef.where("status", "==",
"Online").get();
  let userArray = [];
  let response = [];

  allOnlineUsers.forEach(async (doc) => {
    const userObj = {
      email: doc.id,
      status: doc.data().status,
      timestamp: doc.data().timestamp,
    };
    userArray.push(userObj);
  });

  for (i = 0; i < userArray.length; i++) {
    const userInfoDoc =
firestore.collection("Reg").doc(userArray[i].email);
    const document = await userInfoDoc.get();
    const userName = document.data().name;
    const userLocation = document.data().location;
    response.push({
      name: userName,
      location: userLocation,
      email: userArray[i].email,
      status: userArray[i].status,
      timestamp: userArray[i].timestamp,
    });
  }

  res.status(200).json(response);
});

app.get(`/logout`, async (req, res) => {
  try {
    const status = {
      status: "Offline",
      timestamp: new Date(),
    };
    const email = req.query.email;
    const stateRef = firestore.collection("state").doc(email);
    const doc = await stateRef.set(status);
    res.sendStatus(200);
  } catch (e) {}});
```

- Application business logic Code for container -3

server.js M

Session Validation > server.js > origin

You, 1 second ago | 1 author (You)

```
const express = require("express");
const app = express();
const { Firestore } = require("@google-cloud/firestore");
module.exports = app;

const cors = require("cors");

app.use(
  cors({
    origin: `https://frontend-container-6j5ciz4cca-uc.a.run.app`,
  })
);

const firestore = new Firestore();

app.use(express.json());

app.use(express.urlencoded({ extended: true }));

app.get(`/session`, async (req, res) => {
  const stateRef = firestore.collection("state");

  const allOnlineUsers = await stateRef.where("status", "==", "Online").get();
  let userArray = [];
  let response = [];

  allOnlineUsers.forEach(async (doc) => {
    const userObj = {
      email: doc.id,
      status: doc.data().status,
      timestamp: doc.data().timestamp,
    };
    userArray.push(userObj);
  });

  for (i = 0; i < userArray.length; i++) {
    const userInfoDoc = firestore.collection("Reg").doc(userArray[i].email);
    const document = await userInfoDoc.get();
    const userName = document.data().name;
    const userLocation = document.data().location;
    response.push({
      name: userName,
      location: userLocation,
      email: userArray[i].email,
      status: userArray[i].status,
      timestamp: userArray[i].timestamp,
    });
  }

  res.status(200).json(response);
});

app.get(`/logout`, async (req, res) => {
  try {
```

Figure 26: Application business logic Code for container -3 (Session API).

- Logout API endpoint in container 3.

```
server.js M
Session Validation > server.js > origin
50    res.status(200).json(response);
51  });
52
53  app.get(`/logout`, async (req, res) => {
54    try {
55      const status = {
56        status: "Offline",
57        timestamp: new Date(),
58      };
59      const email = req.query.email;
60      const stateRef = firestore.collection("state").doc(email);
61      const doc = await stateRef.set(status);
62      res.sendStatus(200);
63    } catch (e) {
64      console.log(e);
65    }
66  });
67
68  const PORT = process.env.PORT || 8080;
69  app.listen(PORT, () => {
70    console.log(`Server is running on ${PORT}`);
71  });
72
```

Figure 27: Logout API endpoint in container 3.

- Build docker image of container 3.

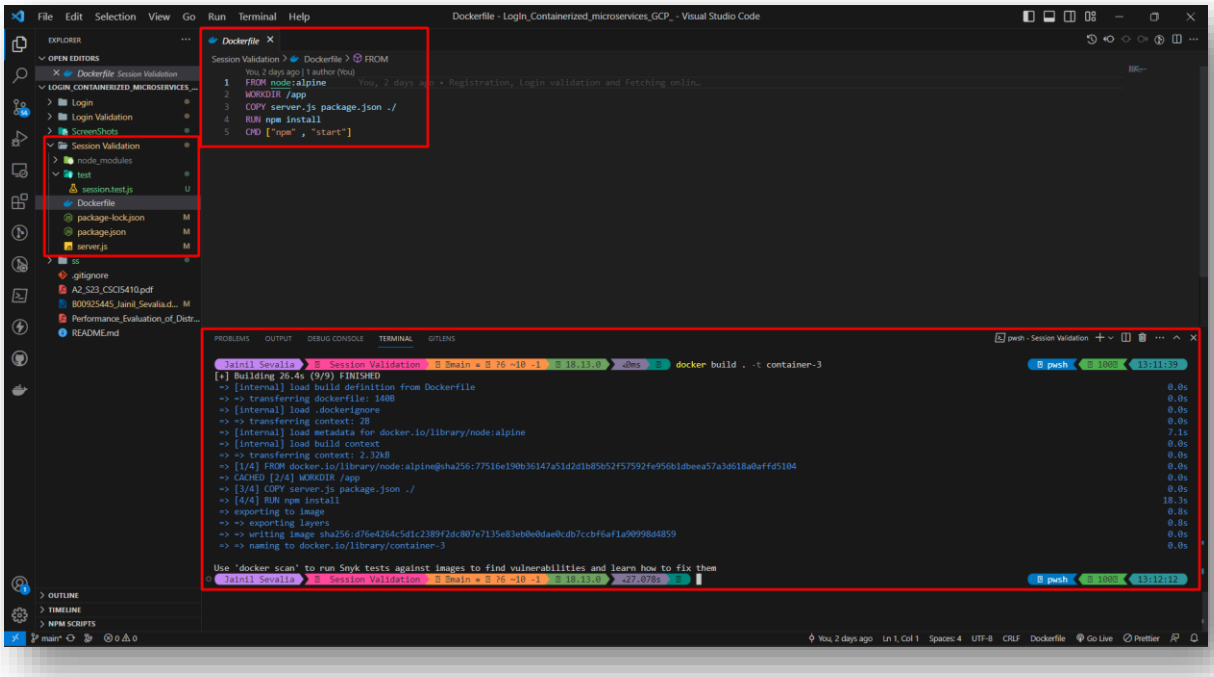

Figure 28 : Build image for container 3.

- Docker image tagged to push it on Artifact Registry.

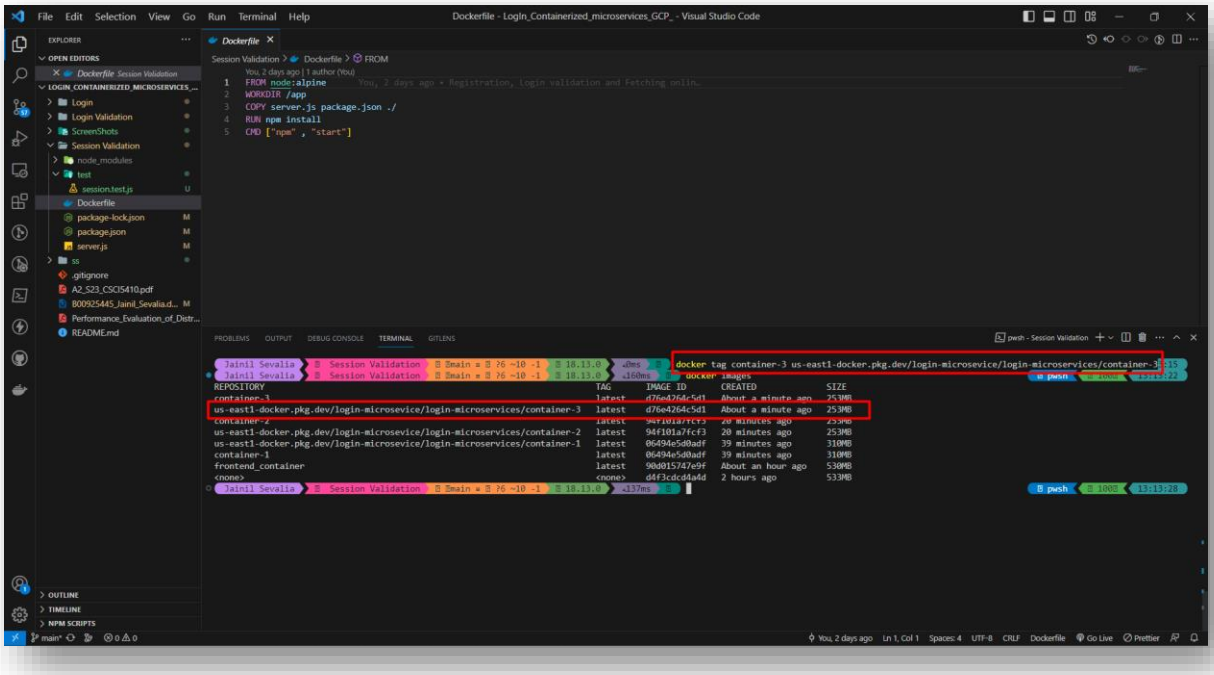

Figure 29: Docker image tagged for container 3.

- Docker image pushed using command mentioned in below image.

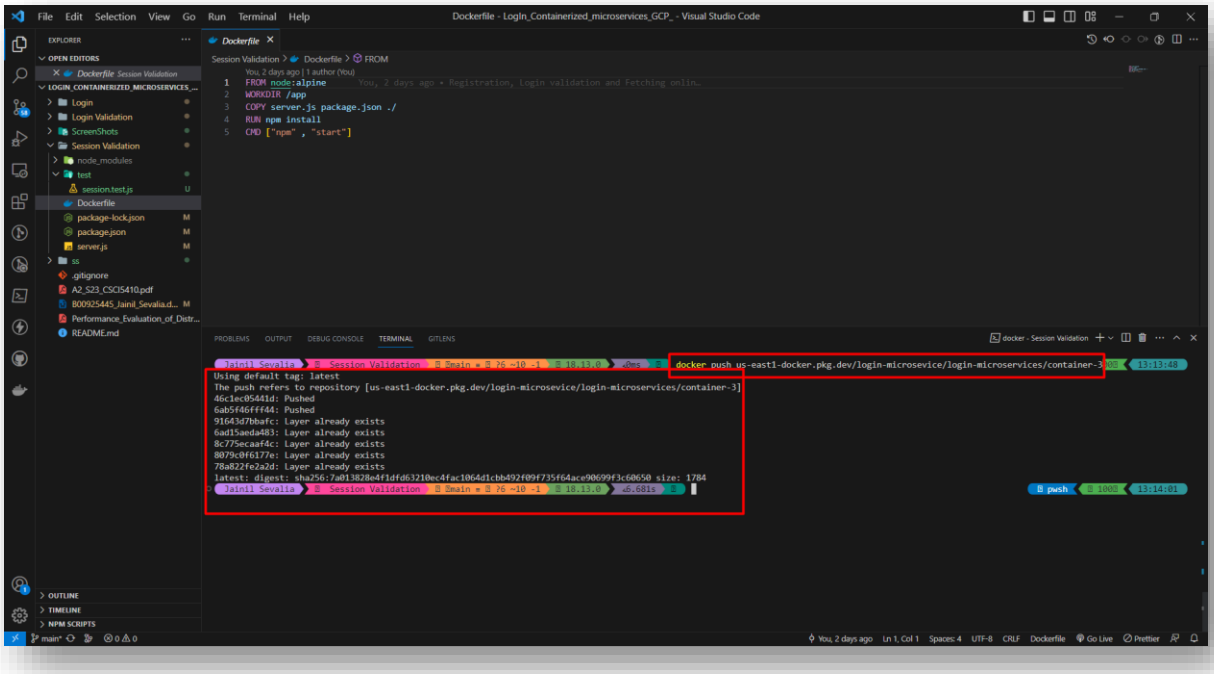

Figure 30 : Docker image build and pushed on the GCP artifact registry.

- GCP Artifact Registry shows the pushed image successfully.

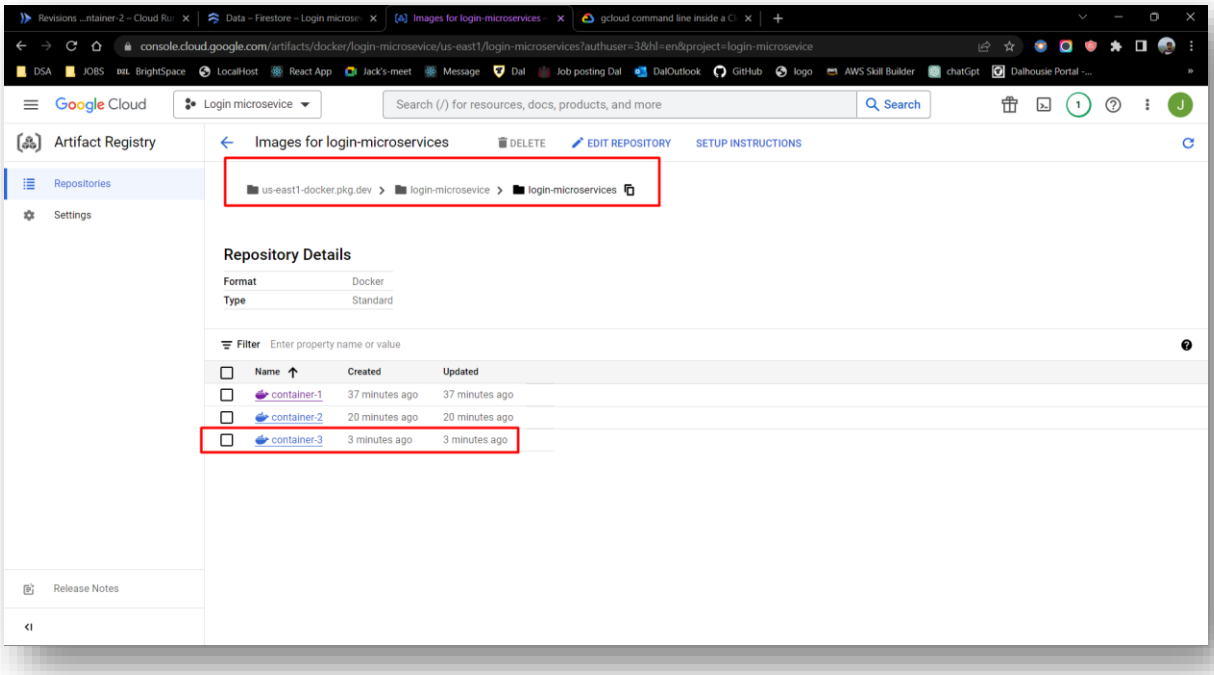

Figure 31: Artifact Registry GCP console.

- Created cloud Run service using pushed image for container 3.

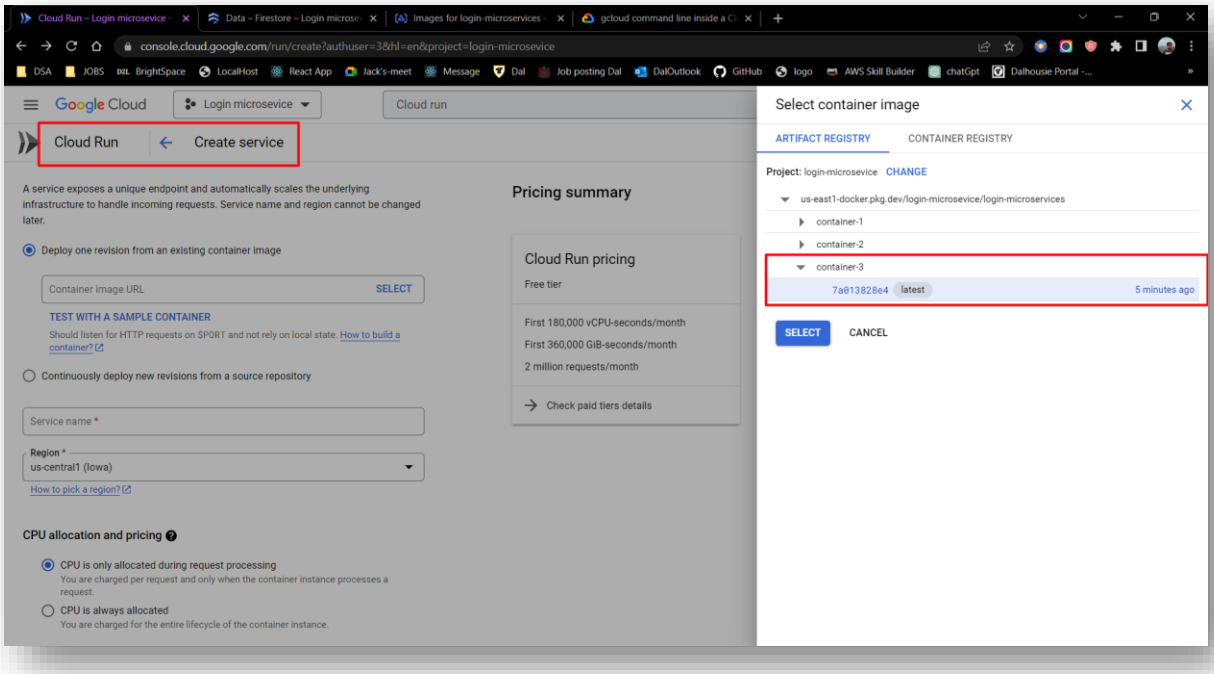

Figure 32 : Creating Cloud Run Service for Container-3.

- Container is running on the GCP cloud run service, which is shown below.

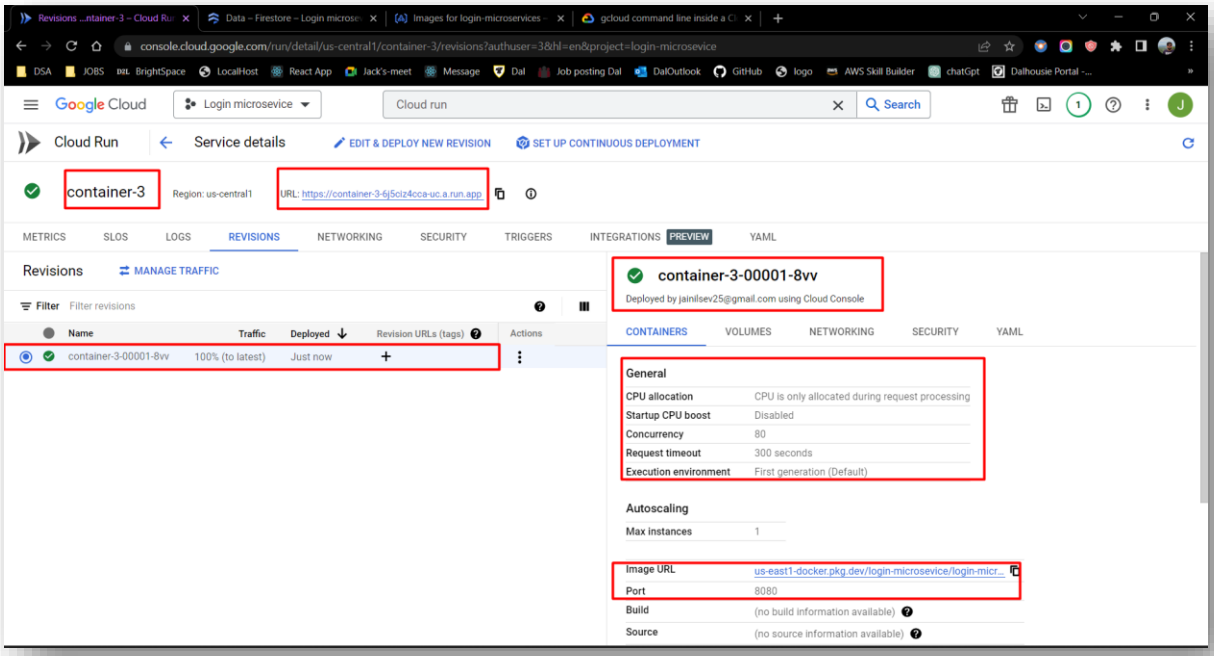

Figure 33: Container-3 Running on Cloud Run service.

- Testing the container by sending request to container using postman.
  - GET request to get all online users.

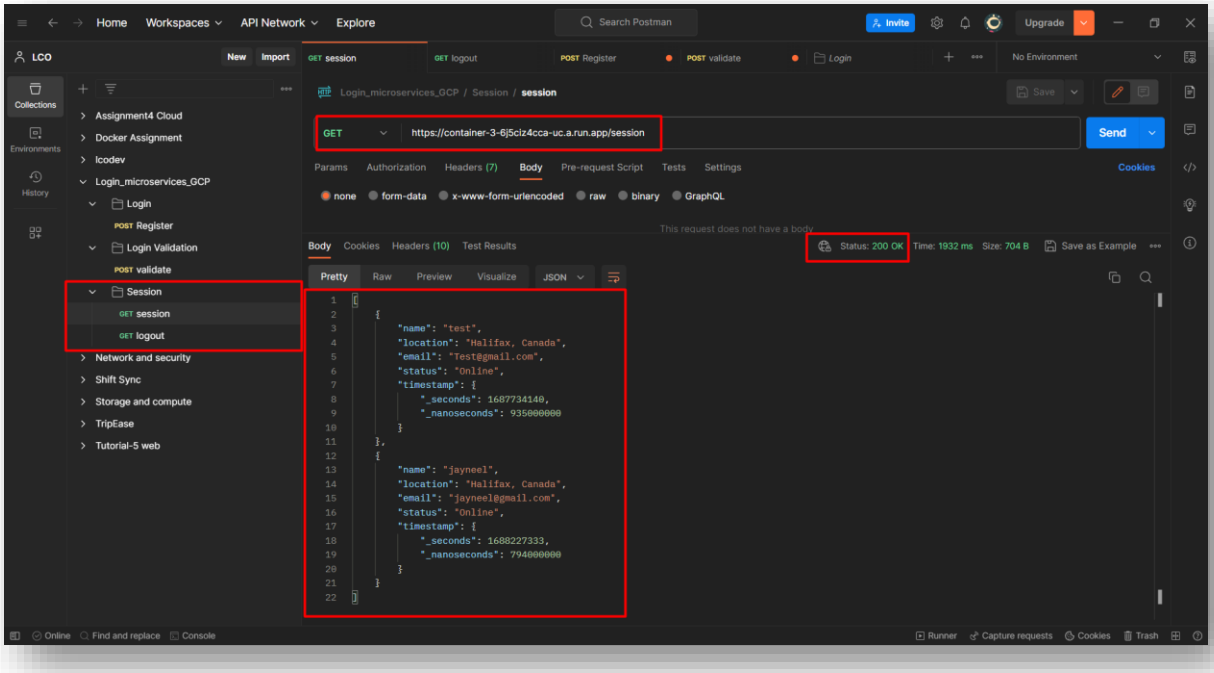

Figure 34 : GET request to get all online users.

- GET request to Logout(end session) of current user.

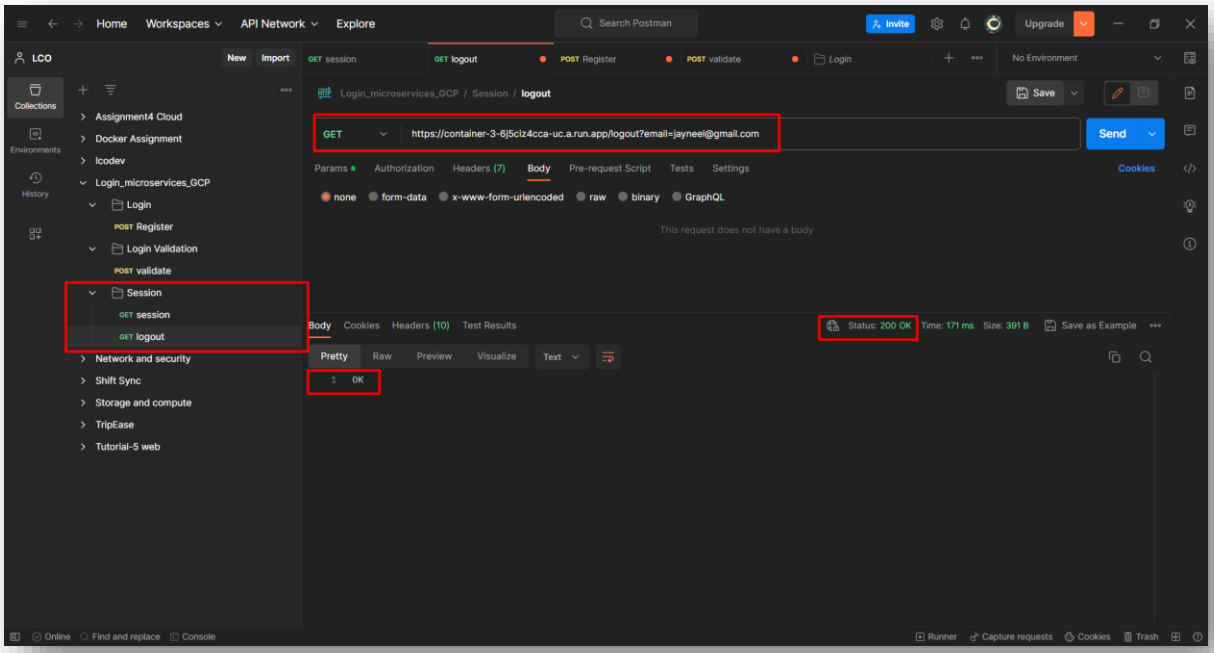

Figure 35 : GET request to Logout user.

- Logout will update the status of user in state collection to offline.

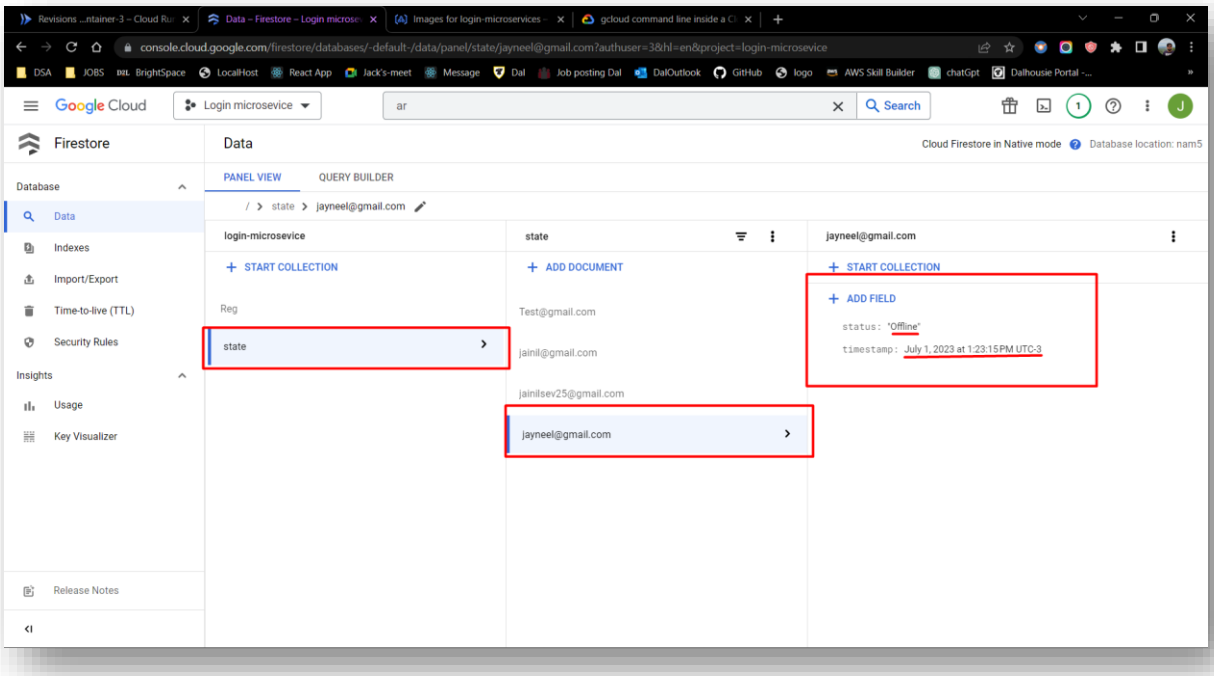

Figure 36: Logout will update the user status to offline in state collection.

H(For Container3). Test Cases for /Logout and /Session endpoint.

- /Session endpoint Should return online users with their information.
- /Logout Should update the user status to offline and return the updated status.

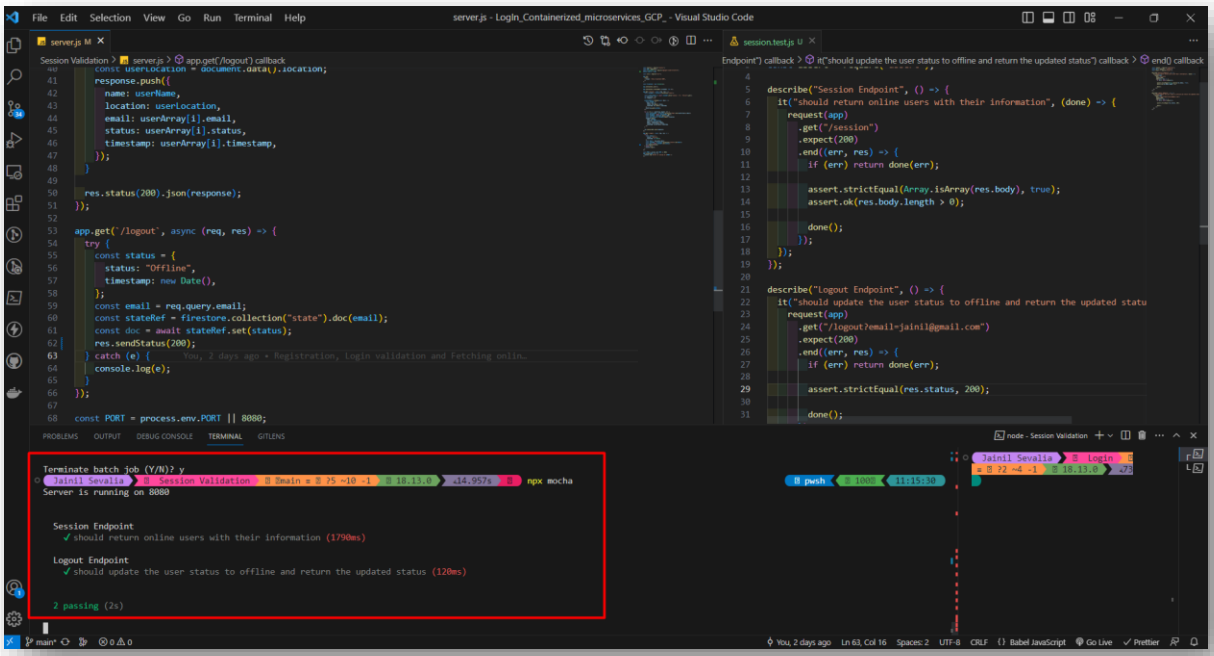

Figure 37 : Test Cases for container 3

G. Frontend and Final testing using UI.

- Docker image building of frontend project.

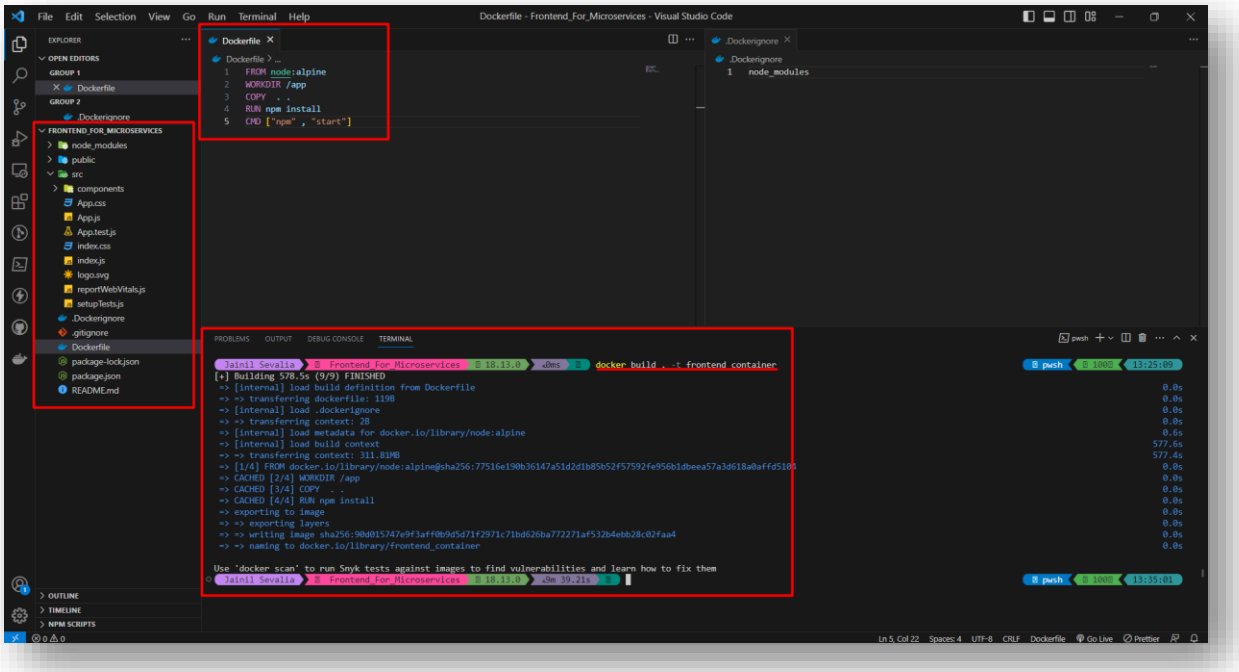

Figure 38 : Docker Image building of Frontend.

- Docker image tagged.

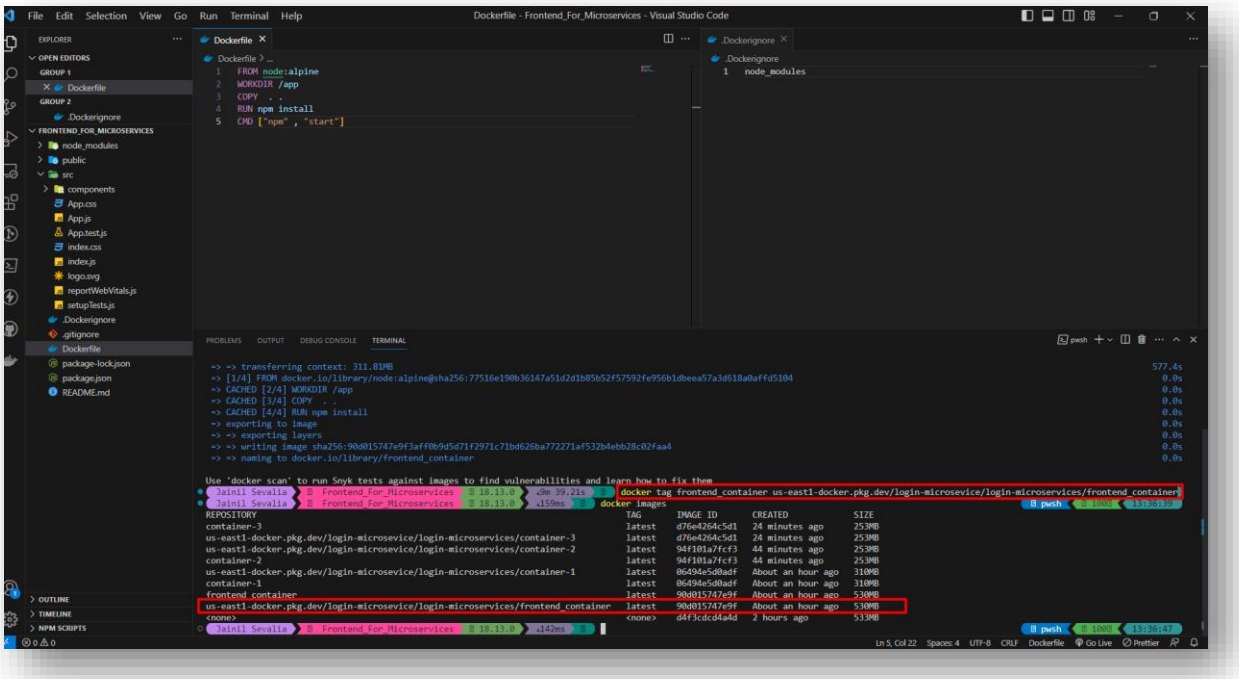

Figure 39 : Docker image tagged for frontend container.

- Push Docker image to Artifact Registry.

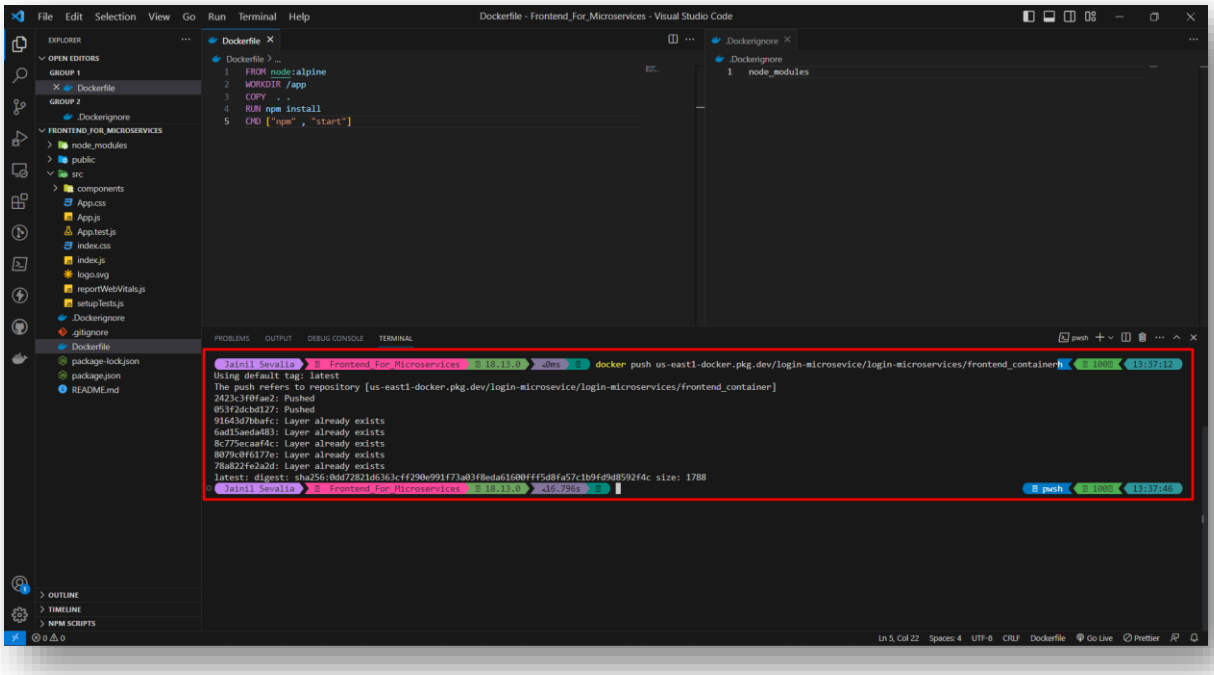

Figure 40 : Frontend Image Pushed to Artifact Registry.

- Artifact Registry Console.

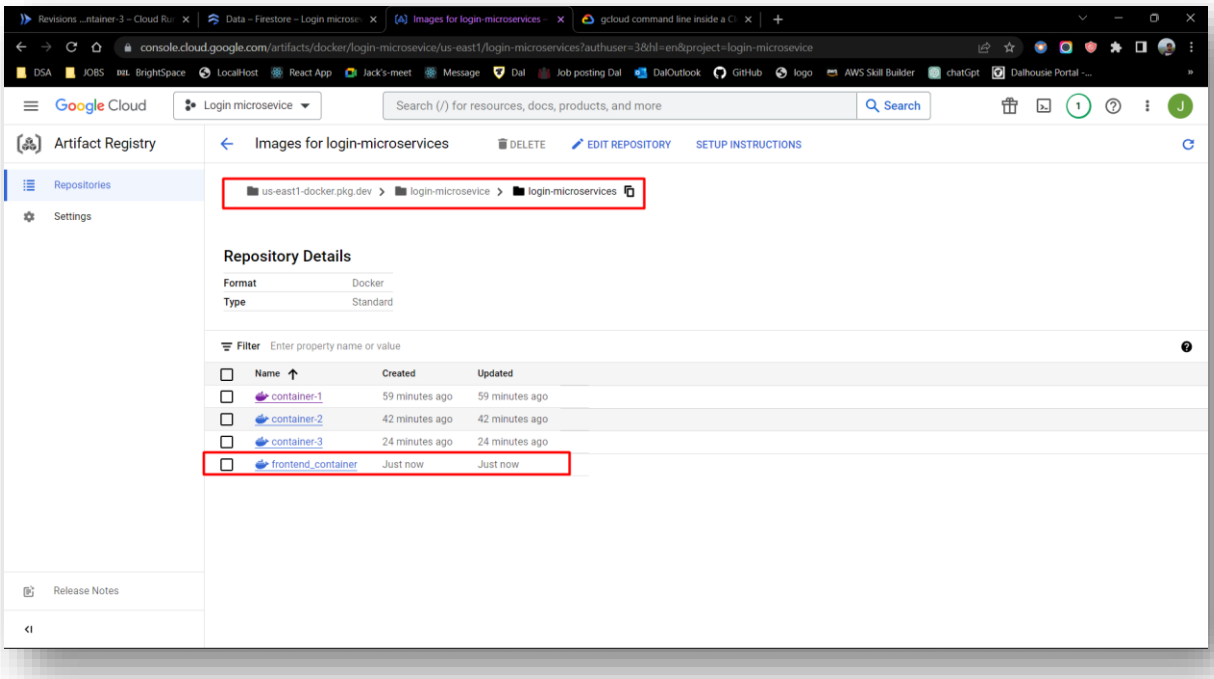

Figure 41 : Artifact Registry - Fronted Image.

- Create new Cloud Run Service for Frontend Container.

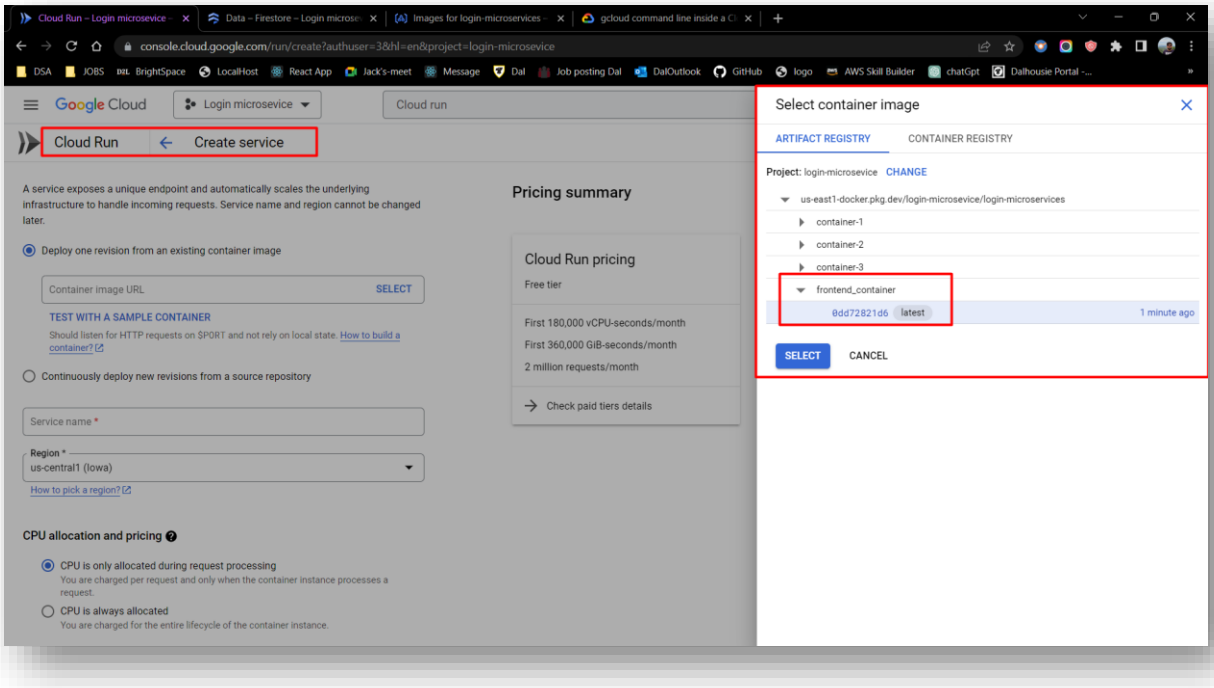

Figure 42 : Creating Cloud Run service for Frontend Container.

- Frontend Running on Cloud Run service.

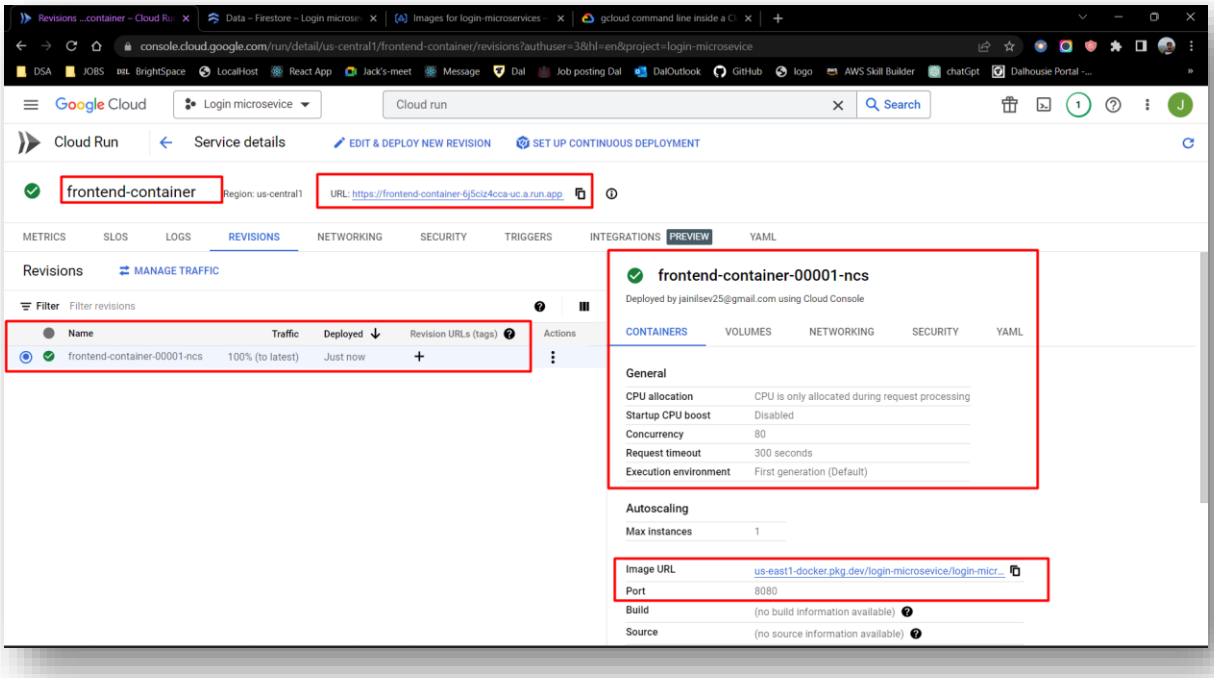

Figure 43: Frontend Running on Cloud Run.

- User Registration page:

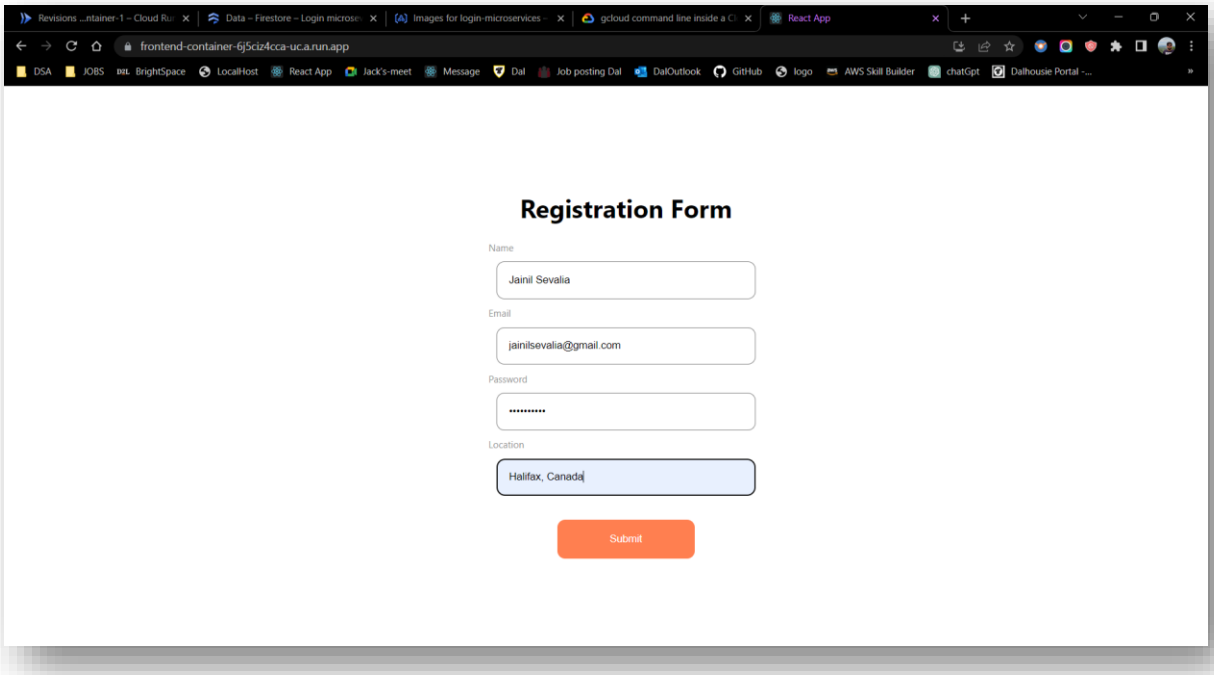

Figure 44: User Registration page(Build using React).

- User Document created successfully in Firestore Database.

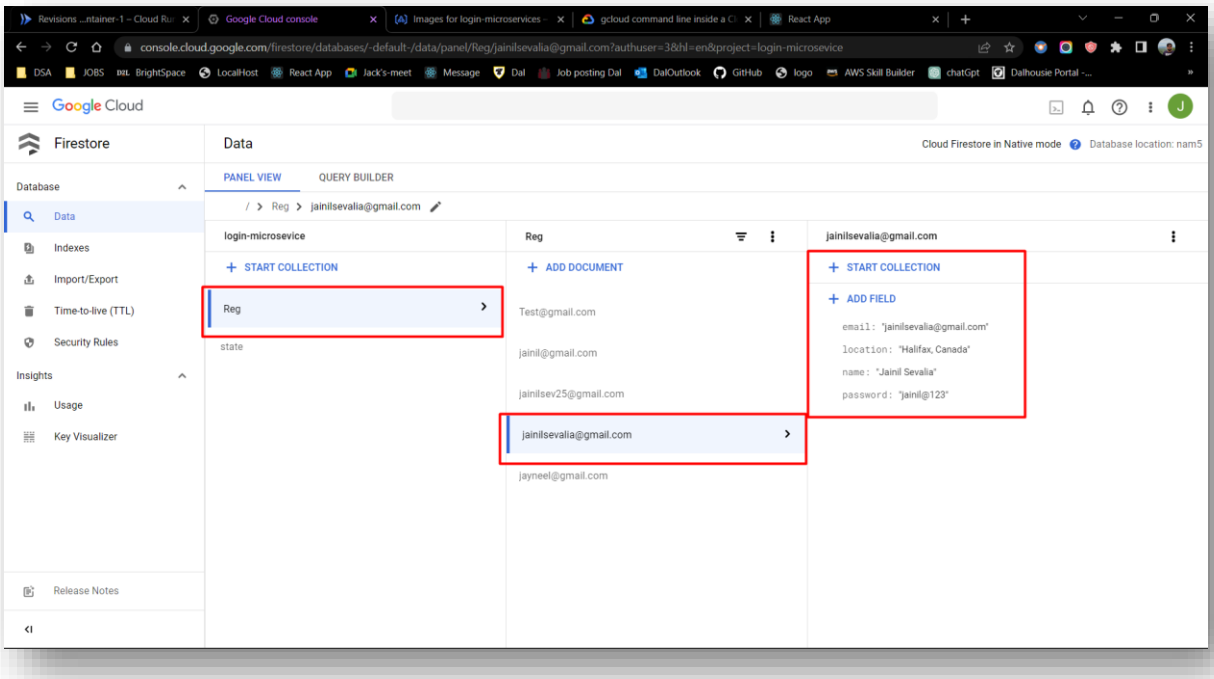

Figure 45 : User Document Successfully created in Firestore.

- Login Page:

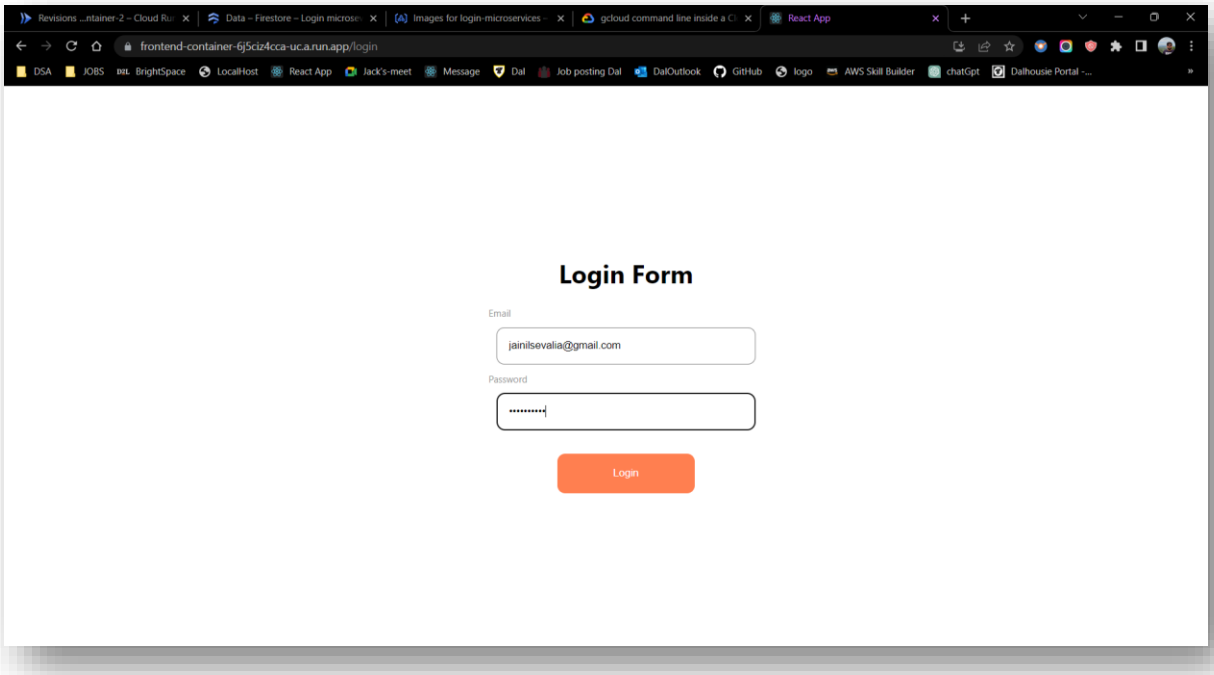

Figure 46 : Login Page.

- Online user page:

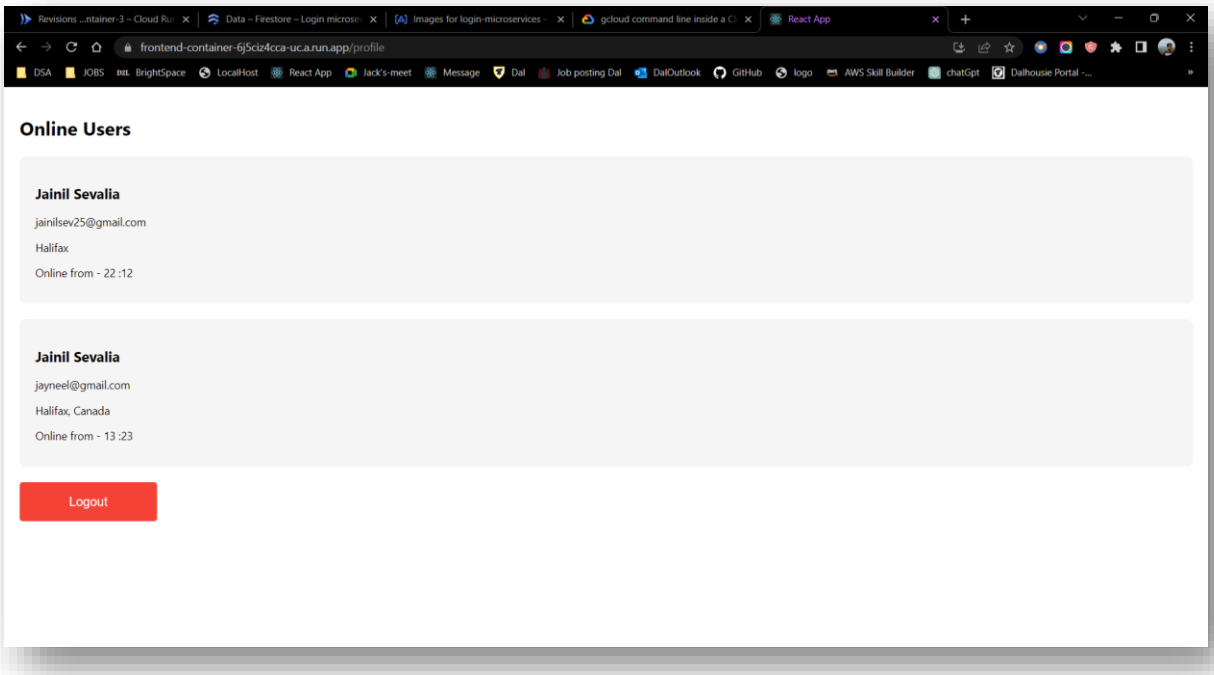

Figure 47 : Online User List after login.

I. Summary:

Google Cloud Run provided me with a serverless compute platform to run my stateless containers on a fully managed environment [1]. It handled infrastructure management, autoscaling, and load balancing, allowing me to focus on deploying and running my microservices [1]. I deployed my containerized microservices as Cloud Run services, benefiting from automatic scaling based on traffic and seamless load balancing.

To encapsulate my backend logic, I leveraged Docker containers. Docker allowed me to automate the deployment, scaling, and management of my applications using containerization [2]. I packaged each microservice into a separate Docker container, ensuring consistent behavior and easy deployment across different environments [2]. This approach simplified the deployment process and provided a lightweight and consistent runtime environment for my microservices.

For storing, managing, and distributing my container images, I used GCR/Artifact Registry. These managed container registries provided by Google Cloud Platform allowed me to push my Docker images, making them readily available for deployment [3]. By pushing the images to GCR/Artifact Registry, I centralized my repository and facilitated the deployment of my containerized microservices to Google Cloud Run [3].

I leveraged Google Cloud Run, Docker containers, and GCR/Artifact Registry to build and deploy my microservices-based application. Google Cloud Run provided a serverless compute platform, Docker enabled containerization and simplified deployment, and GCR/Artifact Registry served as a centralized repository for my container images [4]. This combination of technologies allowed for scalability, portability, and easy deployment while abstracting away infrastructure management concerns [4].

In my application, I utilized React as the frontend technology, leveraging its component-based architecture to develop three web pages - registration, login, and online users. React's efficiency and virtual DOM ensured a smooth user experience. For the backend, I employed Node.js as the runtime environment, coupled with Express.js as the web application framework. This combination allowed me to build containerized microservices responsible for user registration, login validation, and fetching online user information from the Firestore database. The integration of React, Node.js, Express.js and mocha.js [5] facilitated the development of a responsive and dynamic application, complementing the deployment of microservices on Google Cloud Run.

References:

[1] "Google cloud documentation", documentation. [Online]. Available: https://cloud.google.com/docs [Accessed Jul. 1, 2023].

[2] "Reference documentation," Docker Documentation. [Online]. Available: https://docs.docker.com/reference/ [Accessed Jul. 1, 2023].

[3] "Push and pull images", "artifact registry documentation", "google cloud," Google. [Online]. Available: https://cloud.google.com/artifact-registry/docs/docker/pushing-and-pulling [Accessed Jul. 1, 2023].

[4] "Cloud run documentation","cloud run documentation","google cloud," Google. [Online]. https://cloud.google.com/run/docs [Accessed Jul. 1, 2023].

[5] "The fun, simple, flexible JavaScript test framework," Mocha. [Online] https://mochajs.org/ [Accessed Jul. 1, 2023].