

Part B Summary and Steps

I completed the following tasks to build and deploy a containerized application using Google Cloud Platform (GCP):

Containerized Microservices: I created three microservices, each responsible for a specific backend logic of the application. The microservices were containerized using Docker, allowing for easy deployment and scalability.

Firestore Database: I chose Firestore as the database for my application. I created two collections: "Reg" to store user registration data and "state" to store user state information such as online/offline status and timestamp.

Container #1 - Registration: I developed Container #1 to handle user registration. It received registration details from the frontend and stored them in the "Reg" collection of Firestore.

Container #2 - Login Validation: Container #2 was responsible for validating user login information by comparing it with the values in the database. When a user successfully logged in, their state in Firestore was updated to "online".

Container #3 - State Extraction: I implemented Container #3 to extract state information from the Firestore database. It maintained the user's session from login to logout and updated the user's state to "offline" upon logout.

Docker Image Creation and Artifact Registry: I created Docker images for each container and pushed them to the Artifact Registry repository. This ensured that the container images were securely stored and versioned.

Deployment on Cloud Run: I deployed the container images on Cloud Run, a serverless platform provided by GCP. Cloud Run handled the scaling and management of the application, allowing it to run efficiently and handle incoming requests.

Web Page Development: I developed three simple web pages using the technology of my choice. These pages enabled users to register, login, and view online users based on the data retrieved from Firestore.

Testing and Test Case Creation: I wrote test cases to verify the functionality and reliability of the application. Using tools like Supertest, I conducted API testing and captured screenshots as evidence of the testing process.

Note:- Please wait for some time to load the Front end website, it takes up to 10 seconds to load

Front end Deployment link - <https://front-end-i6trs4yawa-uc.a.run.app>

Container1(Registration) Backend Hosted url - <https://container1-i6trs4yawa-uc.a.run.app>

Container 2(Login) Backend Hosted url - <https://container2-i6trs4yawa-uc.a.run.app>

Container 3(Status) Backend Hosted Url - <https://container3-i6trs4yawa-uc.a.run.app>

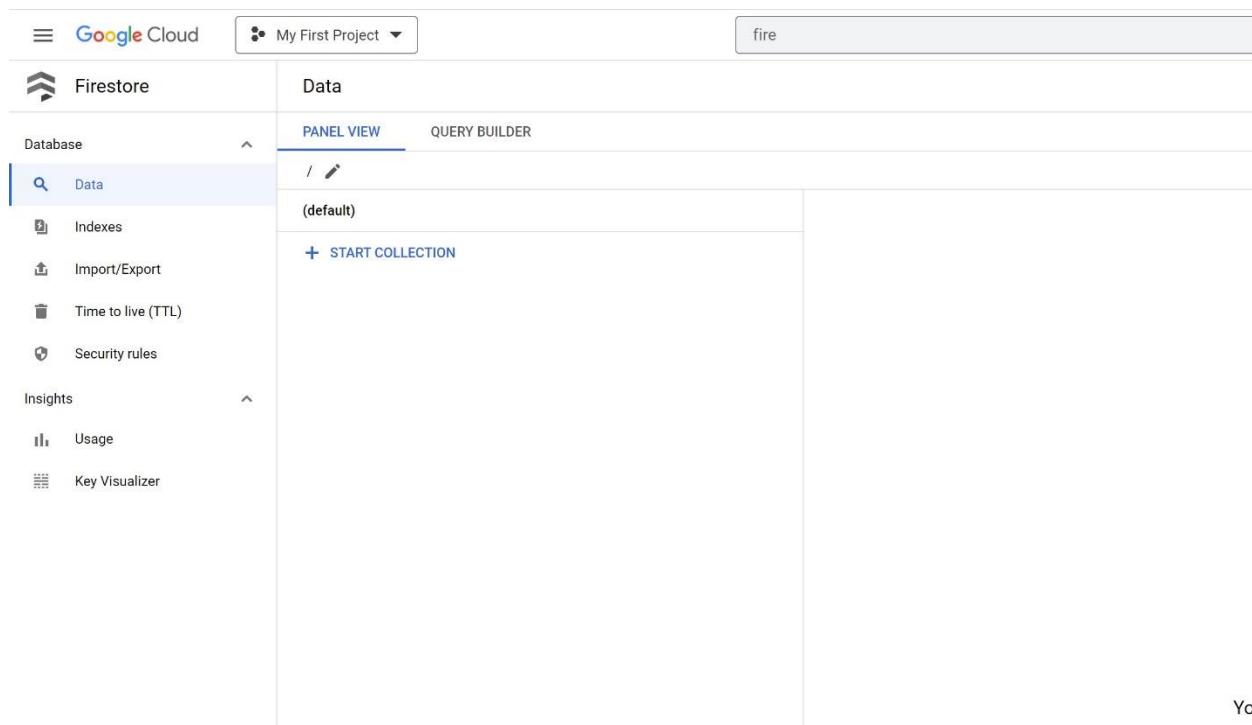
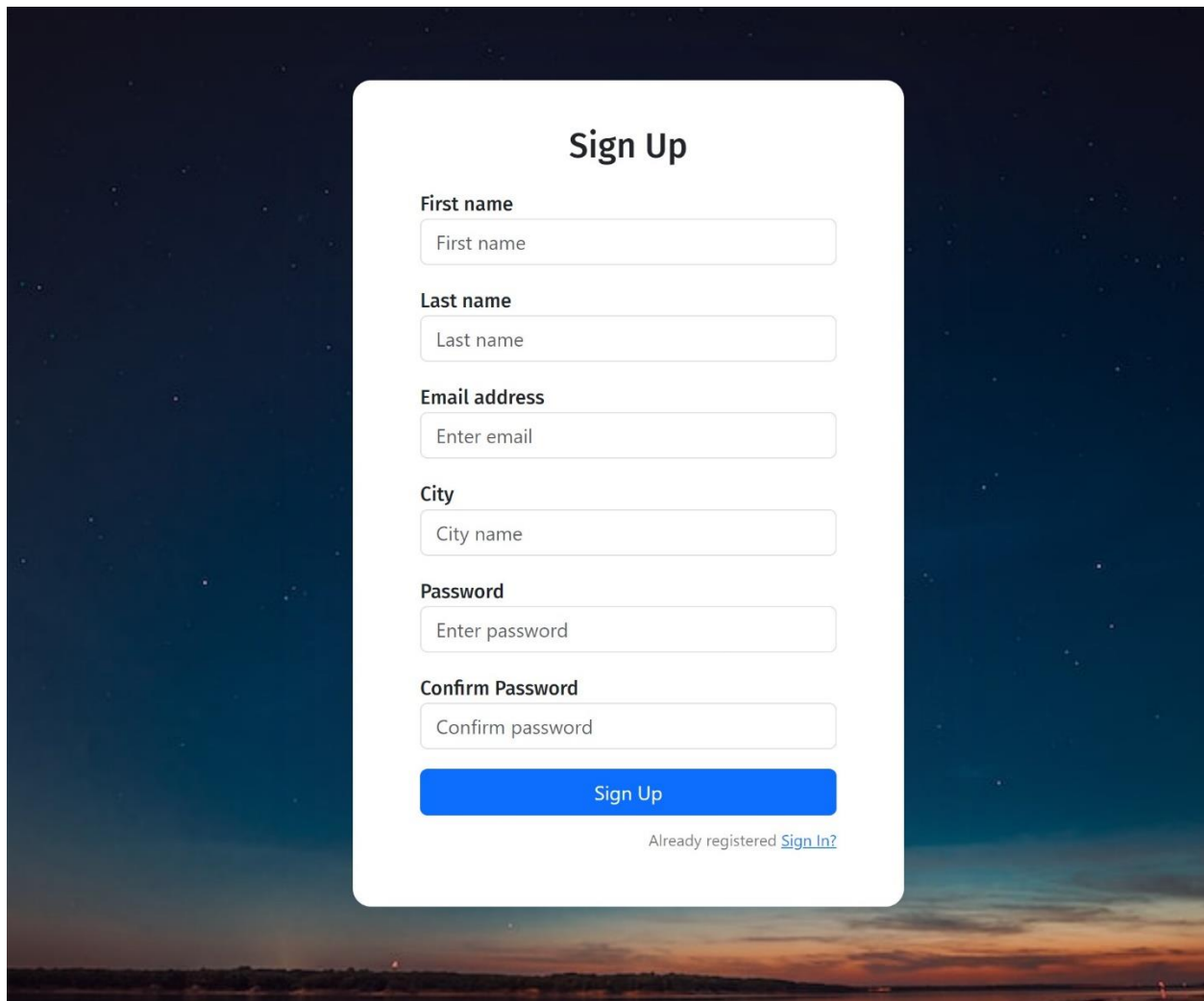


Fig 1: Empty Firestore DB (No data)

A user sign-up form titled "Sign Up" is centered on a dark blue background with a starry night sky and a sunset horizon. The form is a white rounded rectangle containing several input fields and a button. The fields are labeled "First name", "Last name", "Email address", "City", "Password", and "Confirm Password". Each label is followed by an input box with a placeholder text matching the label. Below the "Confirm Password" field is a blue "Sign Up" button. At the bottom right of the form, there is a link that says "Already registered [Sign In?](#)".

Sign Up

First name
First name

Last name
Last name

Email address
Enter email

City
City name

Password
Enter password

Confirm Password
Confirm password

Sign Up

Already registered [Sign In?](#)

Fig 2: User Sign Up Page

Sign Up

First name

First Name should contain only letters

Last name

Last Name should contain only letters

Email address

Please enter a valid email address

City

City Name should contain only letters

Password

Password should be at least 8 characters long and include at least one alpha-numeric and special characters.

Confirm Password

[Sign Up](#)

Already registered [Sign In?](#)

Fig 3: Implemented Front End Validation logic into Sign Up Page

front-end-i6trs4yawa-uc.a.run.app says
Successfully Registered

OK

Sign Up

First name
Fenil

Last name
Patel

Email address
fenil@dal.ca

City
Halifax

Password
.....

Confirm Password
.....

Sign Up

Already registered [Sign In?](#)

Fig 4: Successful Registration of the User

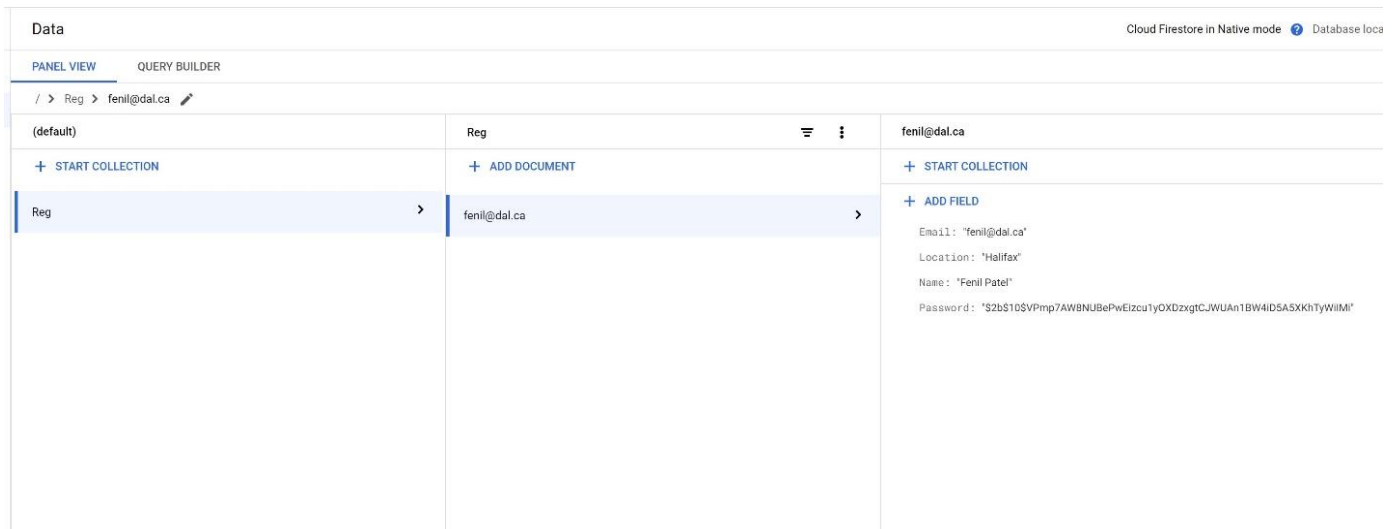


Fig 5 : Registration Entry into Reg Collection of Firestore db

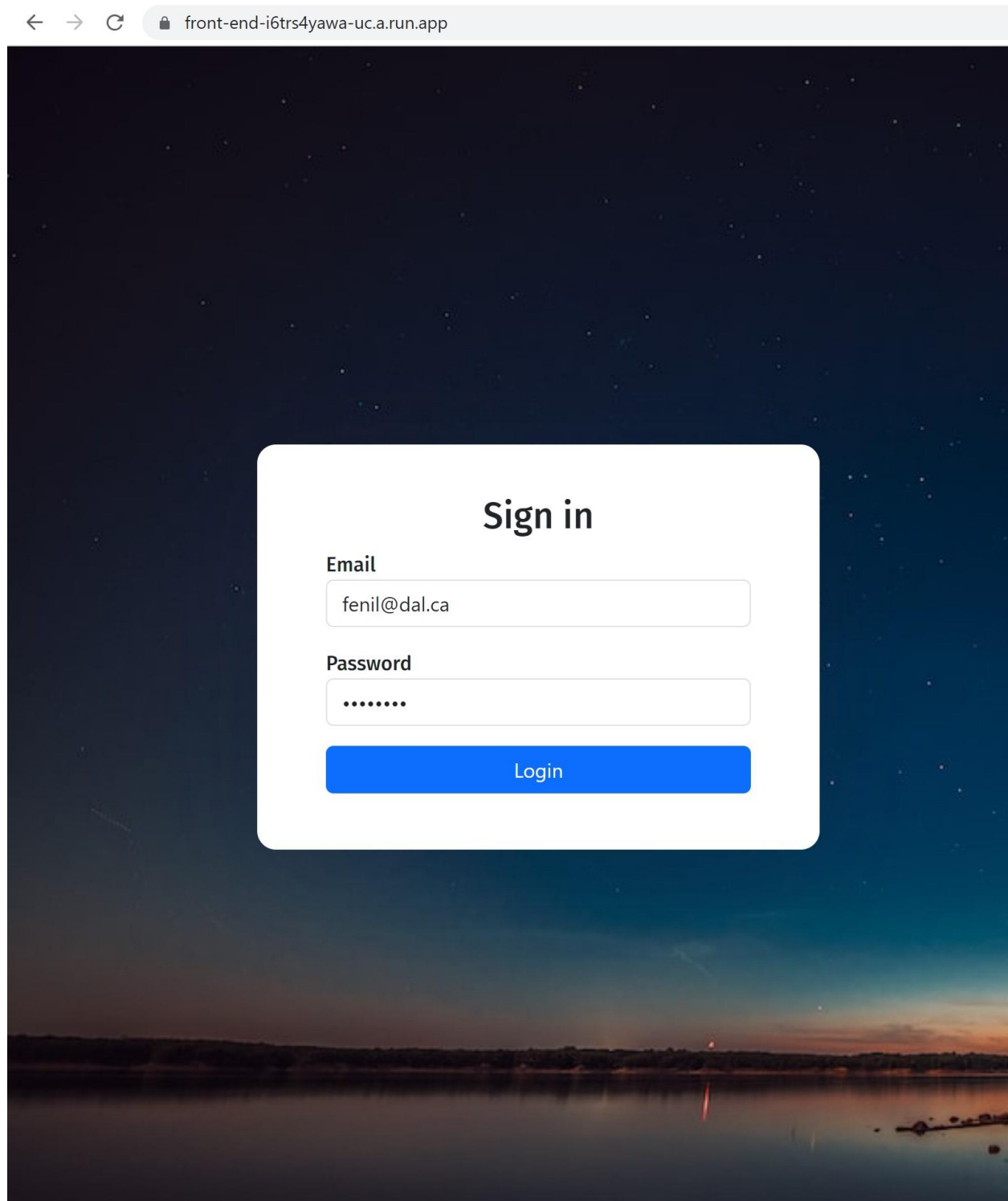


Fig 6 : User Redirects to Sign In Page After Sign Up

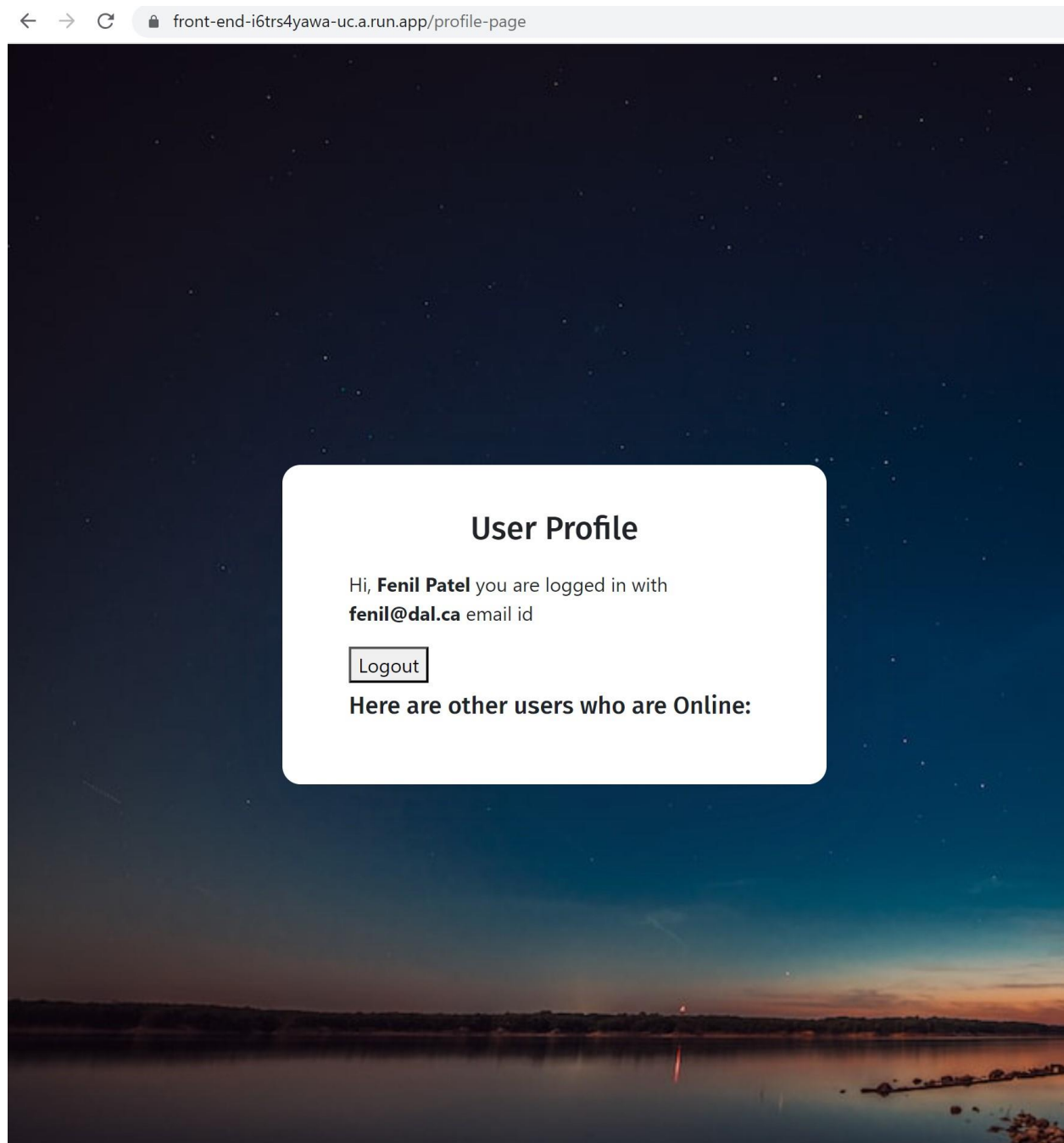


Fig 7: User Logs in, Sees User Profile Page

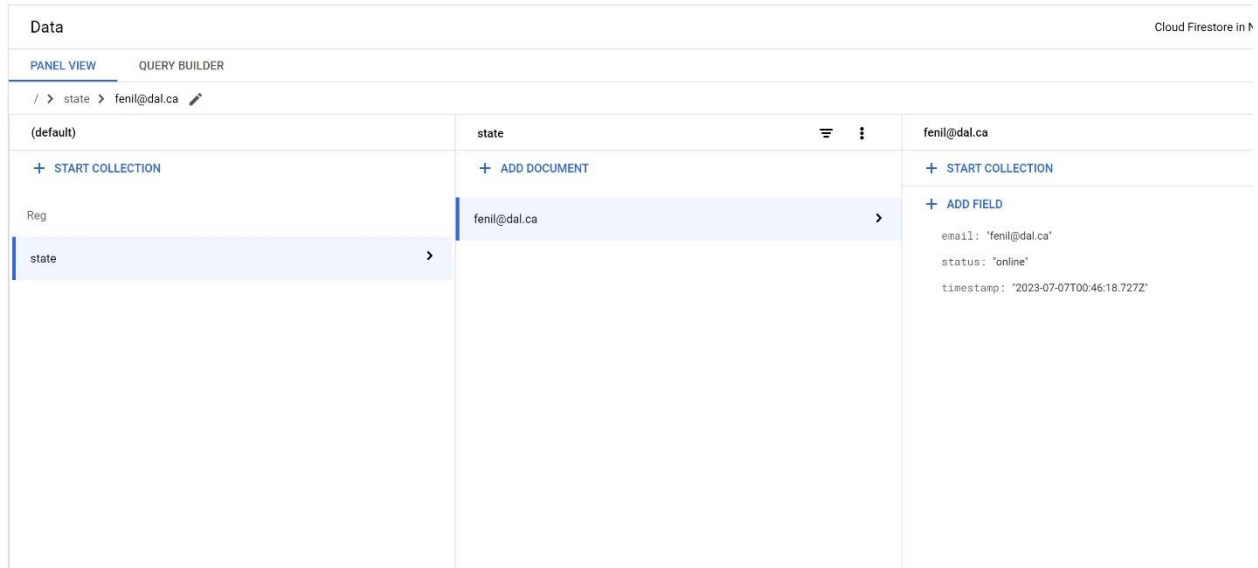


Fig 8 : After Login User state will created and Status will be shown

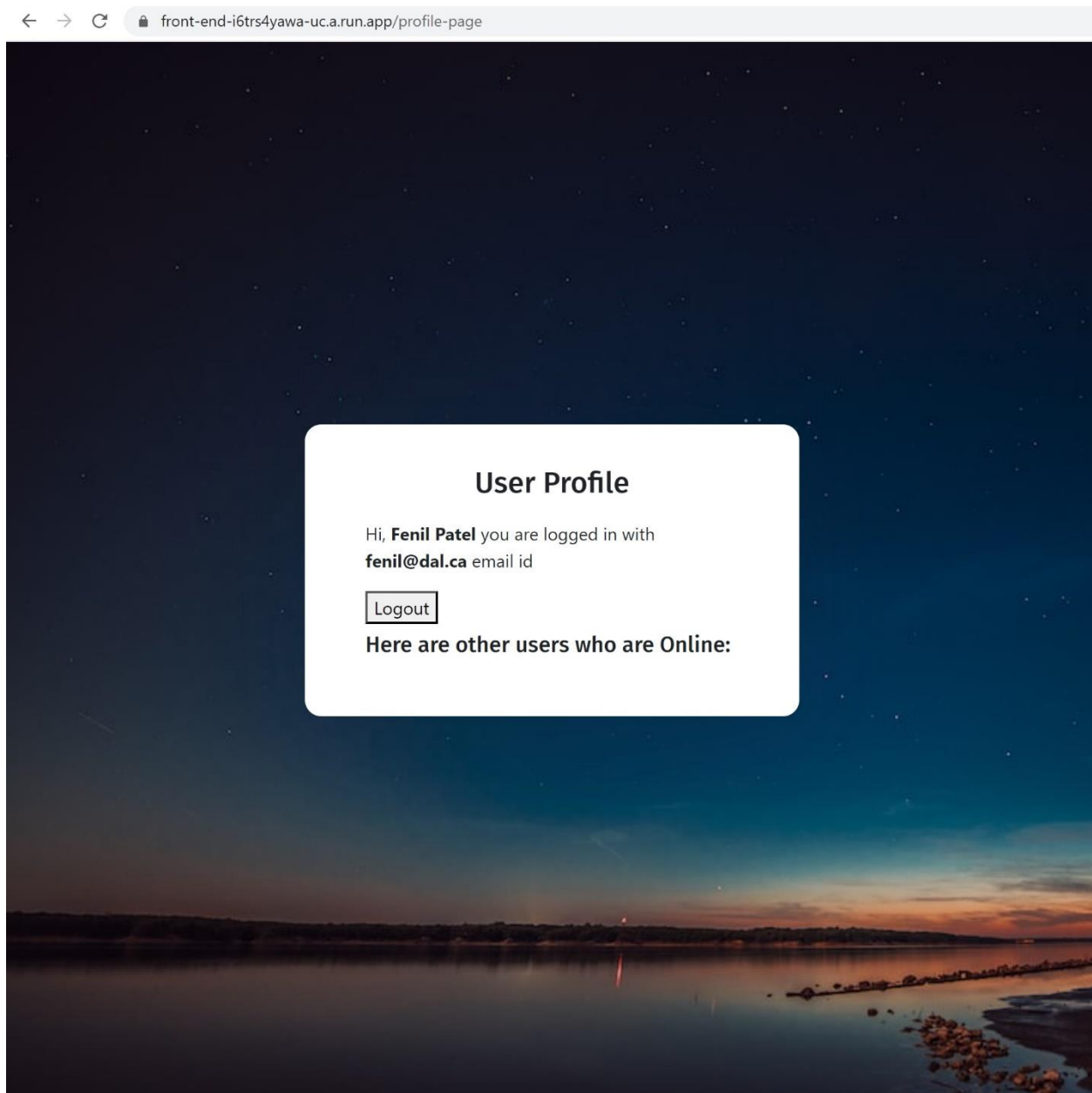


Fig 9 : User Logout through Logout button

state	fenil@dal.ca
+ ADD DOCUMENT	+ START COLLECTION
fenil@dal.ca	+ ADD FIELD
	email: "fenil@dal.ca"
	status: "offline"
	timestamp: "2023-07-07T00:48:25.303Z"

Fig 10: After Successful Log out, User Status will be turn into Offline

state	hp@dal.ca
+ ADD DOCUMENT	+ START COLLECTION
fenil@dal.ca	+ ADD FIELD
hp@dal.ca	email: "hp@dal.ca"
	status: "online"
	timestamp: "2023-07-07T00:51:43.751Z"
sarthak@dal.ca	

Fig 11: Showcasing Other users who are logged in and are online

state	fenil@dal.ca
+ ADD DOCUMENT	+ START COLLECTION
fenil@dal.ca	+ ADD FIELD
hp@dal.ca	email: "fenil@dal.ca"
sarthak@dal.ca	status: "online"
	timestamp: "2023-07-07T00:54:21.225Z"

Fig 12: Showcasing Other users who are logged in and are online

state	sarthak@dal.ca
+ ADD DOCUMENT	+ START COLLECTION
fenil@dal.ca	+ ADD FIELD
hp@dal.ca	email: "sarthak@dal.ca"
sarthak@dal.ca	status: "offline"
	timestamp: "2023-07-07T00:53:34.256Z"

Fig 13: User Sarthak Is offline

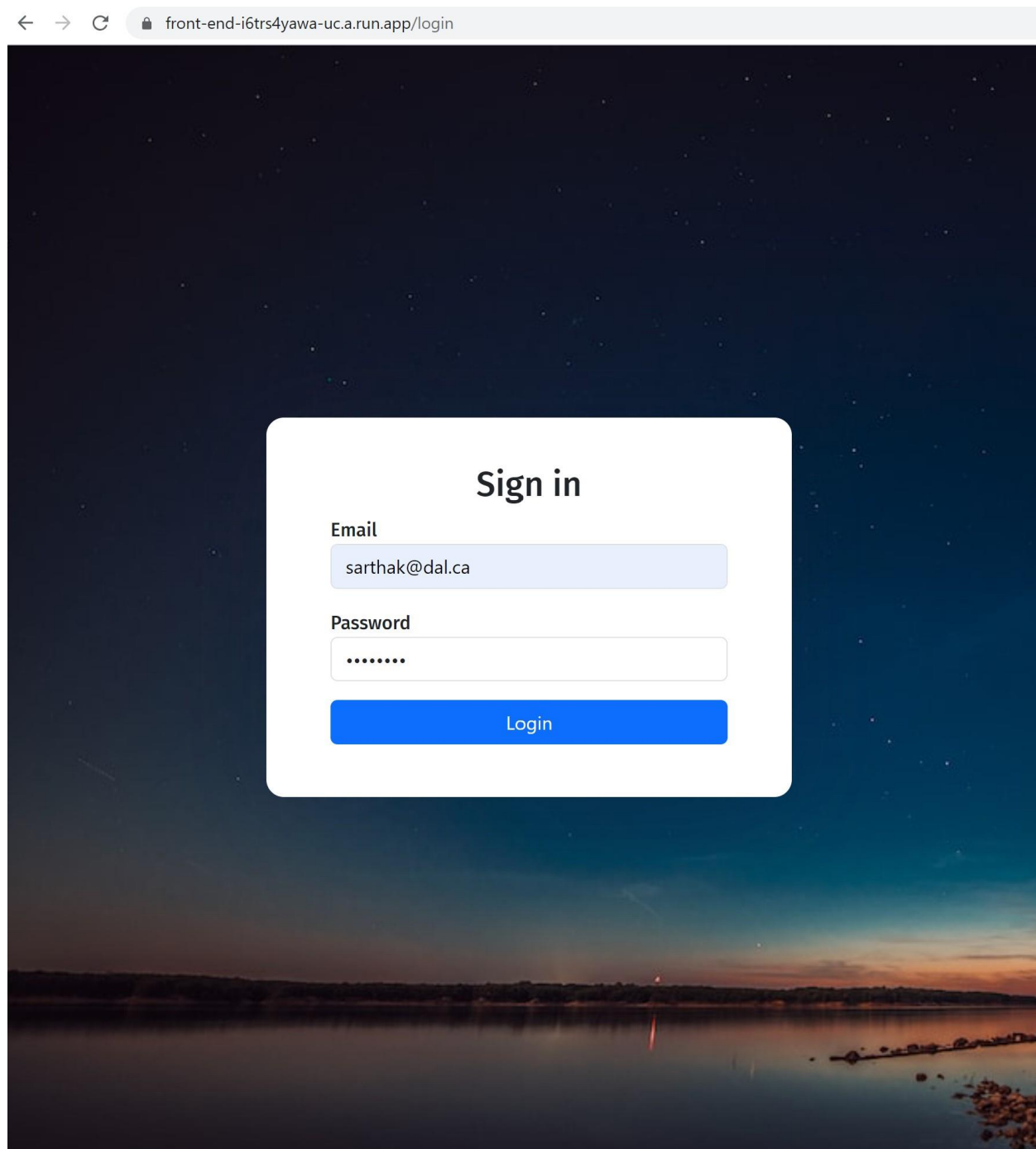


Fig 14: User Sarthak Logs in

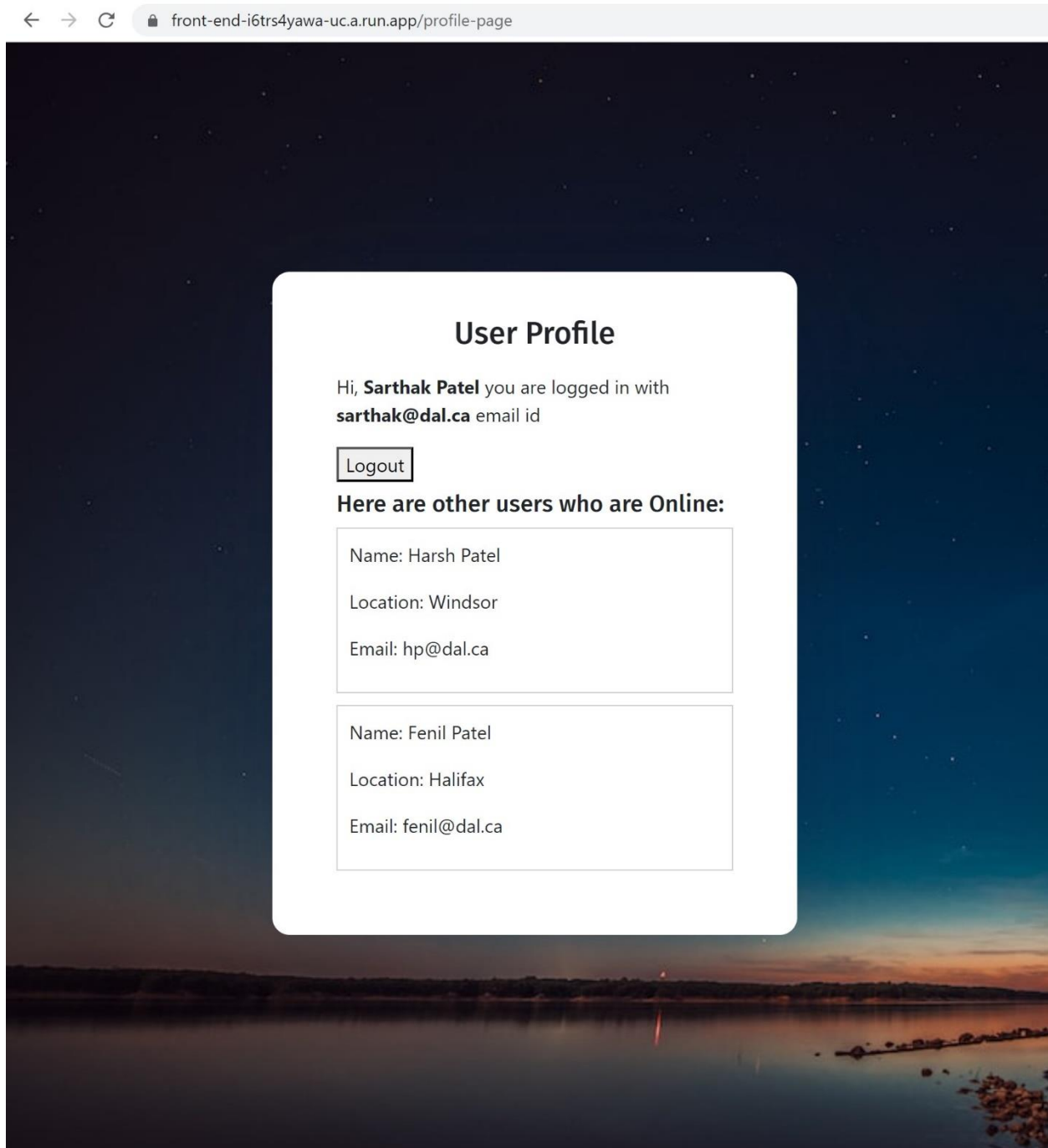


Fig 15 : After Successful Log in, User Profile Will be displayed, and Other Users, who are online will be displayed with their Name, Location and Email

state	sarthak@dal.ca
+ ADD DOCUMENT	+ START COLLECTION
fenil@dal.ca	+ ADD FIELD
hp@dal.ca	email: "sarthak@dal.ca"
sarthak@dal.ca	status: "online"
	timestamp: "2023-07-07T00:57:55.228Z"

Fig 16: User Sarthak's Status Will be turned to online after Login

state	sarthak@dal.ca
+ ADD DOCUMENT	+ START COLLECTION
fenil@dal.ca	+ ADD FIELD
hp@dal.ca	email: "sarthak@dal.ca"
sarthak@dal.ca	status: "offline"
	timestamp: "2023-07-07T00:57:55.228Z"

Fig 17: When User Sarthak Logs out, then His Status will be turned into offline

```

27 app.post( path: '/login', handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) : Promise<...> => {
28   try {
29     // Retrieve email and password from the request body
30     const email = req.body.email;
31     const password = req.body.password;
32
33     // Fetch user data from the Firestore collection
34     const userObj : DocumentSnapshot<DocumentData> = await usersDb.doc(email).get();
35
36     // Check if the user exists
37     if (!userObj.exists) {
38       return res.status( code: 404 ).json( body: { message: 'User not found' } );
39     }
40
41     // Retrieve stored password from user data
42     const userData = userObj.data();
43     const storedPassword = userData.Password;
44     // Compare the provided password with the stored password
45     const passwordMatch = await bcrypt.compare(password, storedPassword);
46     if (!passwordMatch) {
47       return res.status( code: 401 ).json( body: { message: 'Invalid password' } );
48     }
49     // Password is valid, proceed with login logic
50     // Update the user's status in the "state" collection of Firestore
51     const stateData : {...} = {
52       email: email,
53       status: 'online',
54       timestamp: new Date().toISOString()
55     };
56     const response : WriteResult = await stateDb.doc(email).set(stateData);
57
58     // Set sessionID cookie and send user data in the response
59     res.cookie( name: 'sessionID', email, options: { httpOnly: true } );
60     res.send( body: { userData: userData } );
61   } catch (error) {
62     res.status( code: 500 ).send( body: { message: 'Server error' } );
63   }
64 }

```

Fig 18: Login API Backend (Container 2) NodeJS Code

```

// Logout route
app.post( path: '/logout', handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) : Promise<void> => {
  try {
    // Retrieve email from the request body
    const email = req.body.email;

    // Update the user's status in the "state" collection to "offline"
    const stateDoc : DocumentReference<DocumentData> = stateDb.doc(email);
    await stateDoc.update( { status: 'offline' } );

    // Clear the sessionID cookie and send the success message in the response
    res.clearCookie( name: 'sessionID' );
    res.send( body: { message: 'Logout successful' } );
  } catch (error) {
    res.status( code: 500 ).send( body: { message: 'Server error' } );
  }
});

// Start the server
const port : string | number = process.env.PORT || 3003;
app.listen(port, hostname: () : void => {
  console.log( 'Server is running on port ${port}' );
});

// Export the app
module.exports = app;

```

Fig 19: Logout API Backend (Container 3) Nodejs Code


```
// Get online users route
app.get('/online-users', async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) : Promise<void> => {
  try {
    // Retrieve sessionID from the cookie
    const sessionId = req.cookies.sessionID;

    // Query the "state" collection to get online users
    const query : QuerySnapshot<DocumentData> = await stateDb.where( {fieldPath: 'status', opStr: '==', value: 'online'}).get();

    // Retrieve user data for each online user
    const onlineUsers : any[] = [];
    const promises : unknown[] = query.docs.map(async (doc) : Promise<void> => {
      const email = doc.data().email;
      const regDoc : DocumentSnapshot<DocumentData> = await usersDb.doc(email).get();
      const userData = regDoc.data();
      onlineUsers.push(userData);
    });

    await Promise.all(promises);

    // Send the online users data in the response
    res.send( {body: { onlineUsers: onlineUsers }});

  } catch (error) {
    res.status( {code: 500}).send( {body: { message: 'Server error' }});
  }
});
```

Fig 20 : online-users API Backend (Container 3) Nodejs Code

```
package.json  app.js  Dockerfile  package-lock.json  app.test.js  sigma-rarity.json

1  const request : function((Function | Server) , ... | {...} = require('supertest');
2  const app = require('./app'); // Import the main app file
3
4  describe('Registration', () :void => {
5    test('should register a new user', async () :Promise<void> => {
6      // Define the user data for registration
7      const userData : {...} = {
8        firstName: 'Narendra',
9        lastName: 'Modi',
10       email: 'namo@dal.ca',
11       password: 'namo@12345',
12       city: 'Delhi',
13     };
14
15     // Send a POST request to the '/create' endpoint with the user data
16     const response = await request(app)
17       .post('/create')
18       .send(userData);
19
20     // Assert the response
21     expect(response.status).toBe( {expected: 200}); // Expect the response status to be 200 (OK)
22     expect(response.body).toEqual( {expect.anything()}); // Expect the response body to be defined and not empty
23   });
24 });
25
```

Fig 21: Test Case for Registration API

```
console.log
{
  firstName: 'Narendra',
  lastName: 'Modi',
  at log (app.js:28:17)

PASS ./app.test.js
  Registration
    ✓ should register a new user (1034 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.931 s, estimated 2 s
Ran all test suites.
Jest did not exit one second after the test run has completed.
```

Fig 22: Shows Test Cases Passed for Registration API

```
15 describe('Login', () : void => {
16   test('should login a user', async () : Promise<void> => {
17     // Prepare test data
18     const email : string = 'namo@dal.ca';
19     const password : string = 'namo@12345';
20
21     // Create a user in the database
22     const userJson : { ... } = {
23       Name: 'Narendra',
24       Location: 'Delhi',
25       Email: email,
26       Password: await bcrypt.hash(password, salt: 10)
27     };
28     await usersDb.doc(email).set(userJson);
29
30     // Perform the login request
31     const response = await request(app)
32       .post('/login')
33       .send({ email, password });
34
35     // Assert the response
36     expect(response.status).toBe(expected: 200);
37     expect(response.body.userData).toBeDefined();
38     expect(response.body.userData.Email).toBe(email);
39     expect(response.headers['set-cookie']).toBeDefined();
40     expect(response.headers['set-cookie'][0]).toContain(expected: 'sessionID');
41
42     // Cleanup: Delete the user from the database
43     await usersDb.doc(email).delete();
44   });
45 }
```

Fig 23: Test Case for Login API

```
► test('should return 404 for non-existing user', async () : Promise<void> => {  
  // Perform the login request with a non-existing user  
  const response = await request(app)  
    .post('/login')  
    .send({ email: 'nonexisting@example.com', password: 'password123' });  
  
  // Assert the response  
  expect(response.status).toBe( expected: 404 );  
  expect(response.body.message).toBe( expected: 'User not found' );  
});  
  
► test('should return 401 for invalid password', async () : Promise<void> => {  
  // Prepare test data  
  const email : string = 'namo@dal.ca';  
  const password : string = 'namo@12345';  
  
  // Create a user in the database  
  const userJson : { ... } = {  
    Name: 'Narendra',  
    Location: 'Delhi',  
    Email: email,  
    Password: await bcrypt.hash(password, salt: 10)  
  };  
  await usersDb.doc(email).set(userJson);  
  
  // Perform the login request with an invalid password  
  const response = await request(app)  
    .post('/login')  
    .send({ email, password: 'wrongpassword' });  
  
  // Assert the response  
  expect(response.status).toBe( expected: 401 );  
  expect(response.body.message).toBe( expected: 'Invalid password' );  
  
  // Cleanup: Delete the user from the database  
  await usersDb.doc(email).delete();  
});
```

Fig 24: More Test Cases on Login API

```

PASS ./app.test.js
  Login
    ✓ should login a user (1231 ms)
    ✓ should return 404 for non-existing user (87 ms)
    ✓ should return 401 for invalid password (466 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        2.678 s, estimated 4 s
Ran all test suites.
Jest did not exit one second after the test run has completed.

```

Fig 25: Test Cases Passed for Login

```

1  const request :function((Function | Server),... | {...} = require('supertest');
2  const app = require('./app');
3
4  // Mocking Firebase Admin SDK
5  jest.mock('firebase-admin', () :{...} => {
6    // Mock Firestore functionality
7    const firestore :{...} = {
8      collection: jest.fn() :{...} => firestore,
9      doc: jest.fn() :{...} => firestore,
10     where: jest.fn() :{...} => firestore,
11     get: jest.fn() :Promise<{...}> => Promise.resolve( value: { docs: [] })),
12     update: jest.fn() :Promise<void> => Promise.resolve(),
13   };
14
15   return {
16     initializeApp: jest.fn(),
17     firestore: jest.fn() :{...} => firestore,
18     credential: {
19       cert: jest.fn(),
20     },
21   };
22 });
23

```

Fig 26: Mock for Test cases for Status container

```

24 // Test suite for Online Users
25 describe('Online Users', () :void => {
26   // Test case: should retrieve online users
27   it('should retrieve online users', async () :Promise<void> => {
28     // Mock data for online users
29     const mockData : {email: string, status: string} = [
30       {
31         email: 'gaurav@nirma.in',
32         status: 'online',
33         timestamp: '2023-07-04T19:08:50.987Z',
34       },
35     ];
36
37     // Mock the Firestore "get" method to return the mock data
38     const mockGet = jest.fn(() :Promise<{}> => Promise.resolve({ docs: mockData }));
39
40     // Mock the app.locals with the necessary mock implementations
41     app.locals = {
42       stateDb: {
43         where: jest.fn(() => app.locals.stateDb),
44         get: mockGet,
45       },
46       usersDb: {
47         doc: jest.fn(() => app.locals.usersDb),
48         get: jest.fn(() :Promise<{}> => Promise.resolve({ data: () : {Location: string, Name: string} => ({ Name: 'gaurav panchal', Location: 'ahmedabad' }) })),
49       },
50     };
51
52     // Make a request to the "/online-users" endpoint
53     const response : string[] = await request(app).get('/online-users');
54
55     // Assert that the response status is 200
56     expect(response.status).toBe(200);
57   });
58 }
59

```

Fig 27: Test Cases for Status Container

```

PASS ./app.test.js
Online Users
  ✓ should retrieve online users (43 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.75 s, estimated 2 s
Ran all test suites.
Jest did not exit one second after the test run has completed.

```

Fig 28: Test Cases Successfully Passed for Status Container

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL CODEWHISPERER REFERENCE LOG

> react-scripts build

Creating an optimized production build...
One of your dependencies, babel-preset-react-app, is importing the
"@babel/plugin-proposal-private-property-in-object" package without
declaring it in its dependencies. This is currently working because
"@babel/plugin-proposal-private-property-in-object" is already in your
node_modules folder for unrelated reasons, but it may break at any time.

babel-preset-react-app is part of the create-react-app project, which
is not maintained anymore. It is thus unlikely that this bug will
ever be fixed. Add "@babel/plugin-proposal-private-property-in-object" to
your devDependencies to work around this error. This will make this message
go away.

Compiled successfully.

File sizes after gzip:

=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/node:14
=> [auth] library/node:pull token for registry-1.docker.io
=> [1/5] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa
=> [internal] load build context
=> => transferring context: 354.79MB
=> CACHED [2/5] WORKDIR /app
=> [3/5] COPY package*.json ./
=> [4/5] RUN npm install
=> [5/5] COPY . .
=> exporting to image
=> => exporting layers
=> => writing image sha256:e3df3517322b08025b1b5033fa133f0406ec540392ed5ad8de3b1ab8bd20a4d8
=> => naming to us-central1-docker.pkg.dev/sigma-rarity-378302/my-docker-repo/front-end:latest
PS C:\Users\fenil\OneDrive\Desktop\Serverless\Assignment 2\Tutorial3> docker push us-central1-docker.pkg.dev/sigma-rarity-378302/my-docker-repo/front-end:latest
The push refers to repository [us-central1-docker.pkg.dev/sigma-rarity-378302/my-docker-repo/front-end]
14994d5082eb: Pushed
94239d2f6dad: Pushed
f8fd42ba3471: Pushed
0d5f5a015e5d: Pushed
3c777d951de2: Pushed
f8a91dd5fc84: Pushed
cb81227abde5: Pushed
e01a454893a9: Pushed
c45660adde37: Pushed
fe0fb3ab4a0f: Pushed
f1186e5061f2: Pushed
b2dba7477754: Pushed
latest: digest: sha256:e34c95023f5e86fe1e2c303b9fb825f34f4238320e2d6b4201ec2d84117f5687 size: 3056

```

Fig 29: Building and Pushing Front-End docker image into Artifact Registry

Artifact Registry

Images for my-docker-repo
DELETE
EDIT REPOSITORY
SETUP INSTRUCTIONS

Repositories

Settings

us-central1-docker.pkg.dev

>

sigma-rarity-378302

>

my-docker-repo

Repository details

Format

Docker

Type

Standard

Filter

Enter property name or value

<input type="checkbox"/>	Name	Created	Updated
<input type="checkbox"/>	container1	3 hours ago	2 hours ago
<input type="checkbox"/>	container2	2 hours ago	2 hours ago
<input type="checkbox"/>	container3	2 hours ago	2 hours ago
<input type="checkbox"/>	front-end	1 hour ago	1 hour ago

Fig 30: Docker Image is uploaded into Artifact Registry

```

=> => transferring dockerfile: 443B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/node:18.14.0
=> [auth] library/node:pull token for registry-1.docker.io
=> [1/6] FROM docker.io/library/node:18.14.0@sha256:5ab5e06df2c58dabb18b96363c2fab16b10b7150d13434a12024f599aa645bc
=> [internal] load build context
=> => transferring context: 80.69MB
=> CACHED [2/6] WORKDIR /app
=> [3/6] COPY . .
=> [4/6] RUN npm uninstall bcrypt
=> [5/6] RUN npm install bcryptjs
=> [6/6] RUN npm install bcrypt
=> exporting to image
=> => exporting layers
=> => writing image sha256:77702e6fdc05aa2e0cc0d89f3a964a31ec1ba2e7fc4c160c74c48767a449dc48
=> => naming to us-central1-docker.pkg.dev/sigma-rarity-378302/my-docker-repo/container1:latest
PS C:\Users\fenil\OneDrive\Desktop\Serverless\Assignment 2\registration> docker push us-central1-docker.pkg.dev/sigma-rarity-378302/my-docker-repo/container1:latest
The push refers to repository [us-central1-docker.pkg.dev/sigma-rarity-378302/my-docker-repo/container1]
a233209af893: Pushed
adaae10e4198: Pushed
2f63357e2903: Pushed
f15798e3fa04: Pushed
7adc16751dc3: Layer already exists
70c5854ab5fe: Layer already exists
a7e5b414b5d8: Layer already exists
aff26125b01e: Layer already exists
4c92897e605e: Layer already exists
0b6859e9fff1: Layer already exists
11829b3be9c0: Layer already exists
dc8e1d8b53e9: Layer already exists
9d49e0bc68a4: Layer already exists
8e396a1aad50: Layer already exists
latest: digest: sha256:806fe514a37069b7c86cb15b08daf8789b162d094c621aaef7a0d479609679c3 size: 3265
PS C:\Users\fenil\OneDrive\Desktop\Serverless\Assignment 2\registration> 

```

Fig 31: Docker Image of Registration code is built and being pushed to Artifact Registry

```

=> exporting to image
=> => exporting layers
=> => writing image sha256:01998889889380887bd8213485c49a49156ca67024f6f1309403b64677d0c9b3
=> => naming to us-central1-docker.pkg.dev/sigma-rarity-378302/my-docker-repo/container2:latest
PS C:\Users\fenil\OneDrive\Desktop\Serverless\Assignment 2\login> docker push us-central1-docker.pkg.dev/sigma-rarity-378302/my-docker-repo/container2:latest
The push refers to repository [us-central1-docker.pkg.dev/sigma-rarity-378302/my-docker-repo/container2]
782c51e652b7: Pushed
86a68a1be15c: Pushed
50125fa2eef0: Pushed
8e6611318f0d: Pushed
7adc16751dc3: Layer already exists
70c5854ab5fe: Layer already exists
a7e5b414b5d8: Layer already exists
aff26125b01e: Layer already exists
4c92897e605e: Layer already exists
0b6859e9fff1: Layer already exists
11829b3be9c0: Layer already exists
dc8e1d8b53e9: Layer already exists
9d49e0bc68a4: Layer already exists
8e396a1aad50: Layer already exists
latest: digest: sha256:0e987696ae6b36406f9a872d12077d98d18fa221c82dd7054d7800312a338a16 size: 3265
PS C:\Users\fenil\OneDrive\Desktop\Serverless\Assignment 2\login> 

```

Fig 32: Docker Image of the Login Code is built and being Pushed into Artifact Registry


```

PS C:\Users\fenil\OneDrive\Desktop\Serverless\Assignment 2\state> docker build -t us-central1-docker.pkg.dev/sigma-rarity-378302/my-docker-repo/container3:latest .
[+] Building 18.4s (12/12) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 441B
=> [internal] load .dockerignore
=> => transferring context: 28
=> [internal] load metadata for docker.io/library/node:18.14.0
=> [auth] library/node:pull token for registry-1.docker.io
=> [1/6] FROM docker.io/library/node:18.14.0@sha256:5ab5e86df2c58dabb18b963636c2fab16b10b7150d13434a12024f599aa645bc
=> [internal] load build context
=> => transferring context: 81.93MB
=> CACHED [2/6] WORKDIR /app
=> [3/6] COPY . .
=> [4/6] RUN npm uninstall bcrypt
=> [5/6] RUN npm install bcryptjs
=> [6/6] RUN npm install bcrypt
=> exporting to image
=> => exporting layers
=> => writing image sha256:021f3d45002c417ad865caf3f7a425a5919b47e105dd5120f7ac15e51abec40b
=> => naming to us-central1-docker.pkg.dev/sigma-rarity-378302/my-docker-repo/container3:latest
PS C:\Users\fenil\OneDrive\Desktop\Serverless\Assignment 2\state> docker push us-central1-docker.pkg.dev/sigma-rarity-378302/my-docker-repo/container3:latest
The push refers to repository [us-central1-docker.pkg.dev/sigma-rarity-378302/my-docker-repo/container3]
a1c851cad389: Pushed
59a1bc64cd2e: Pushed
792cf281168b: Pushed
13a516de65a9: Pushed
7adc16751dc3: Layer already exists
70c5854ab5fe: Layer already exists
a7e5b414b5d8: Layer already exists
aff26125b01e: Layer already exists
4c92897e605e: Layer already exists
0b6859e9fff1: Layer already exists
11829b3be9c0: Layer already exists
dc0e1d0b53e9: Layer already exists
9d49e0bc68a4: Layer already exists
8e396a1aad50: Layer already exists
latest: digest: sha256:232b4541e955b631496c7f9ed6e996273e9a77543440e681fb17a7817fe0a300 size: 3265
PS C:\Users\fenil\OneDrive\Desktop\Serverless\Assignment 2\state>

```

Fig 33: Docker Image of status code is built and being pushed to artifact registry

[←](#)
[Digests for container1](#)
[DELETE](#)
[SETUP INSTRUCTIONS](#)

[us-central1-docker.pkg.dev](#) > [sigma-rarity-378302](#) > [my-docker-repo](#) > [container1](#)

Filter Enter property name or value

<input type="checkbox"/>	Name	Description	Tags	Created	Updated ↓	Vulnerabilities ?
<input type="checkbox"/>	806fe514a370		latest	4 minutes ago	4 minutes ago	API disabled
<input type="checkbox"/>	f712a3ca2e68			2 hours ago	2 hours ago	API disabled

Fig 34: Docker Image of Container 1 in artifact Registry

← Digests for container2 DELETE [SETUP INSTRUCTIONS](#)

us-central1-docker.pkg.dev > sigma-rarity-378302 > my-docker-repo > container2

Filter Enter property name or value

<input type="checkbox"/>	Name	Description	Tags	Created	Updated ↓	Vulnerabilities ?
<input type="checkbox"/>	0e987696ae6b		latest	3 minutes ago	3 minutes ago	API disabled ⋮
<input type="checkbox"/>	87214f7da64d			2 hours ago	2 hours ago	API disabled ⋮

Fig 35: Docker Image of Container 2 in artifact Registry

← Digests for container3 DELETE [SETUP INSTRUCTIONS](#)

us-central1-docker.pkg.dev > sigma-rarity-378302 > my-docker-repo > container3

Filter Enter property name or value

<input type="checkbox"/>	Name	Description	Tags	Created	Updated ↓	Vulnerabilities ?
<input type="checkbox"/>	232b4541e955		latest	1 minute ago	1 minute ago	API disabled ⋮
<input type="checkbox"/>	a2b508736478			2 hours ago	2 hours ago	API disabled ⋮

Fig 36: Docker Image of Container 3 in Artifact Registry

us-central1-docker.pkg.dev > sigma-rarity-378302 > my-docker-repo > front-end

Filter Enter property name or value


<input type="checkbox"/>	Name	Description	Tags	Created	Updated ↓	Vulnerabilities ?
<input type="checkbox"/>	 e34c95023f5e		latest	1 hour ago	1 hour ago	API disabled

Fig 37: Docker Image of front-end code into Artifact Registry

console.cloud.google.com/run/create?project=sigma-rarity-378302

Google Cloud My First Project

Cloud Run Create service

A service exposes a unique endpoint and automatically scales the underlying infrastructure to handle incoming requests. Service name and region cannot be changed later.

☒ Deploy one revision from an existing container image

Container image URL [SELECT](#)

[TEST WITH A SAMPLE CONTAINER](#)

Should listen for HTTP requests on \$PORT and not rely on local state. [How to build a container](#)

☐ Continuously deploy new revisions from a source repository

Service name *

Region * [How to pick a region?](#)

CPU allocation and pricing

☒ CPU is only allocated during request processing
You are charged per request and only when the container instance processes a request.

☐ CPU is always allocated
You are charged for the entire lifecycle of the container instance.

Auto-scaling

Minimum number of instances * Maximum number of instances *

Set to one to reduce cold starts. [Learn more](#)

Fig 38: Creating a Cloud Run Service for Front-End Container image

Ingress control ?

- ☐ Internal
Allow traffic from VPCs and certain Google Cloud services in your project, Shared VPC, regional internal application load balancers and traffic allowed by VPC service controls.
[Learn more](#)
- ☒ All
Allow direct access to your service from the Internet

Authentication * ?

- ☒ Allow unauthenticated invocations
Tick this if you are creating a public API or website.
- ☐ Require authentication
Manage authorised users with Cloud IAM.

Container, Networking, Security



CONTAINER

NETWORKING

SECURITY



For adding more containers, use YAML-based deployment. [See Cloud Run YAML reference](#)

[DISMISS](#)

General

Container port

3000

Requests will be sent to the container on this port. We recommend that you listen on \$PORT instead of this specific number.

Container command

Leave blank to use the entry point command defined in the container image.

Container arguments

Fig 39: Configuring Front end Service

Memory
512 MiB

Memory to allocate to each instance of this container.

CPU
1

Number of vCPUs allocated to each instance of this container.

Request timeout
300

seconds

Time within which a response must be returned (maximum 3600 seconds).

Maximum concurrent requests per instance
80

The maximum number of concurrent requests that can reach each instance. [What is concurrency?](#)

Execution environment

The execution environment your container runs in. [Learn more](#)

☒ **Default**
Cloud Run will select a suitable execution environment for you.

☐ **First generation**
Faster cold starts.

☐ **Second generation**
Network file system support, full Linux compatibility, faster CPU and network performance.

Environment variables

+ ADD VARIABLE

Secrets ?

ADD A SECRET REFERENCE

Health checks ?

+ ADD HEALTH CHECK

Cloud SQL connections ?

+ ADD CONNECTION

CREATE

CANCEL

Fig 40 : Creating the Front end Service for Cloud Run

The screenshot displays the AWS Cloud Run console for a service named 'front-end' in the 'us-central-1' region. The URL is <https://front-end-i6trs4yawa-uc.a.run.app>. The 'REVISIONS' tab is active, showing a single revision 'front-end-00001-5mg' with 100% traffic and deployed 1 hour ago. The 'PREVIEW' tab is also visible, showing the service configuration.

front-end-00001-5mg
Deployed by fenilpatel61@gmail.com using Cloud Console

CONTAINERS | VOLUMES | NETWORKING | SECURITY | YAML

General

CPU allocation	CPU is only allocated during request processing
Startup CPU boost	Disabled
Concurrency	80
Request timeout	300 seconds
Execution environment	First generation (default)

Auto-scaling

Max. instances	100
----------------	-----

Image URL us-central1-docker.pkg.dev/sigma-rarity-378302/my-d...

Port 3000

Build (no build information available)

Source (no source information available)

Command and arguments (container entrypoint)

CPU limit 1

Memory limit 512MiB

Fig 41: Front end service is running on cloud run at port 3000.

The screenshot displays the AWS Cloud Run console for a service named 'container1' in the 'us-central-1' region. The URL is <https://container1-i6trs4yawa-uc.a.run.app>. The 'REVISIONS' tab is active, showing a single revision 'container1-00001-nf8' with 100% traffic and deployed 2 hours ago. The 'PREVIEW' tab is also visible, showing the service configuration.

container1-00001-nf8
Deployed by fenilpatel61@gmail.com using Cloud Console

CONTAINERS | VOLUMES | NETWORKING | SECURITY | YAML

General

CPU allocation	CPU is only allocated during request processing
Startup CPU boost	Disabled
Concurrency	80
Request timeout	300 seconds
Execution environment	First generation (default)

Auto-scaling

Max. instances	100
----------------	-----

Image URL us-central1-docker.pkg.dev/sigma-rarity-378302/my-d...

Port 3001

Build (no build information available)

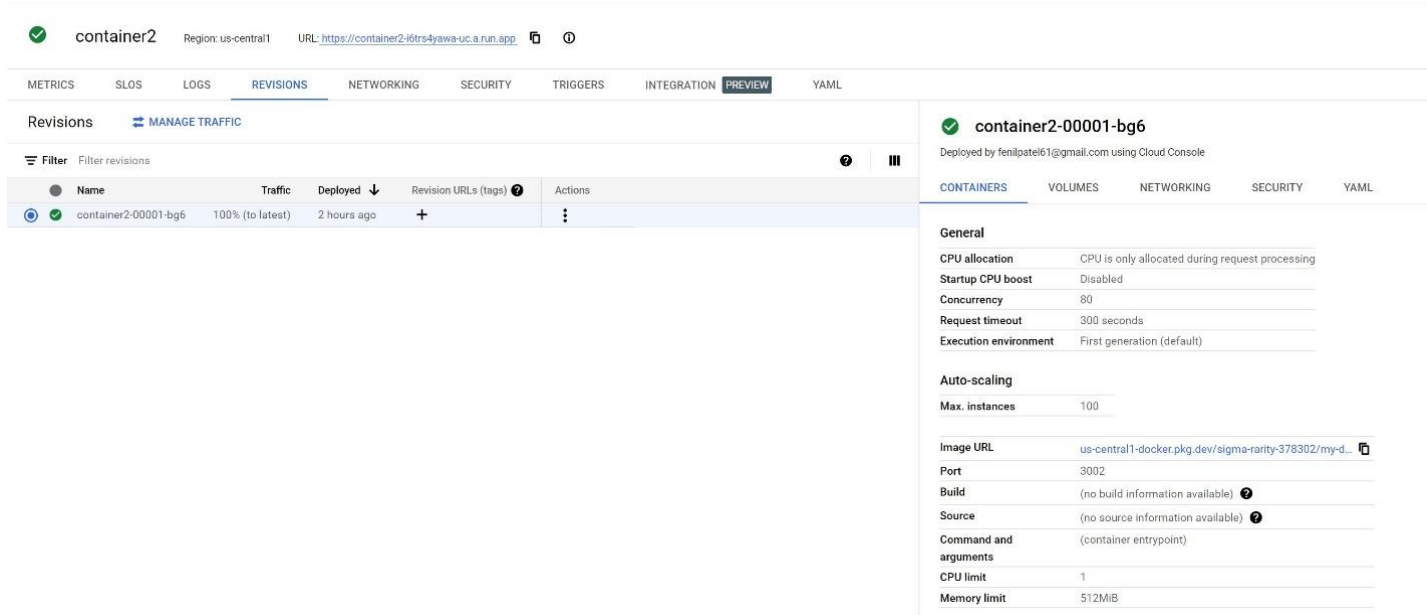
Source (no source information available)

Command and arguments (container entrypoint)

CPU limit 1

Memory limit 512MiB

Fig 42: Same configuration followed for creating a cloud run service for container 1 Image, and it is running at port 3001



The screenshot shows the Google Cloud Run console for a service named 'container2'. The 'REVISIONS' tab is active, displaying a table with one revision: 'container2-00001-bg6'. This revision is at 100% traffic, deployed 2 hours ago, and is the latest. The right-hand panel shows the configuration for this revision, including general settings (CPU allocation, startup boost, concurrency, timeout, environment) and auto-scaling (max instances: 100). The image URL is 'us-central1-docker.pkg.dev/sigma-rarity-378302/my-d...'. The port is 3002. The build and source information is unavailable.

Name	Traffic	Deployed	Revision URLs (tags)	Actions
container2-00001-bg6	100% (to latest)	2 hours ago	+	⋮

container2-00001-bg6
Deployed by fenilpatel61@gmail.com using Cloud Console

General

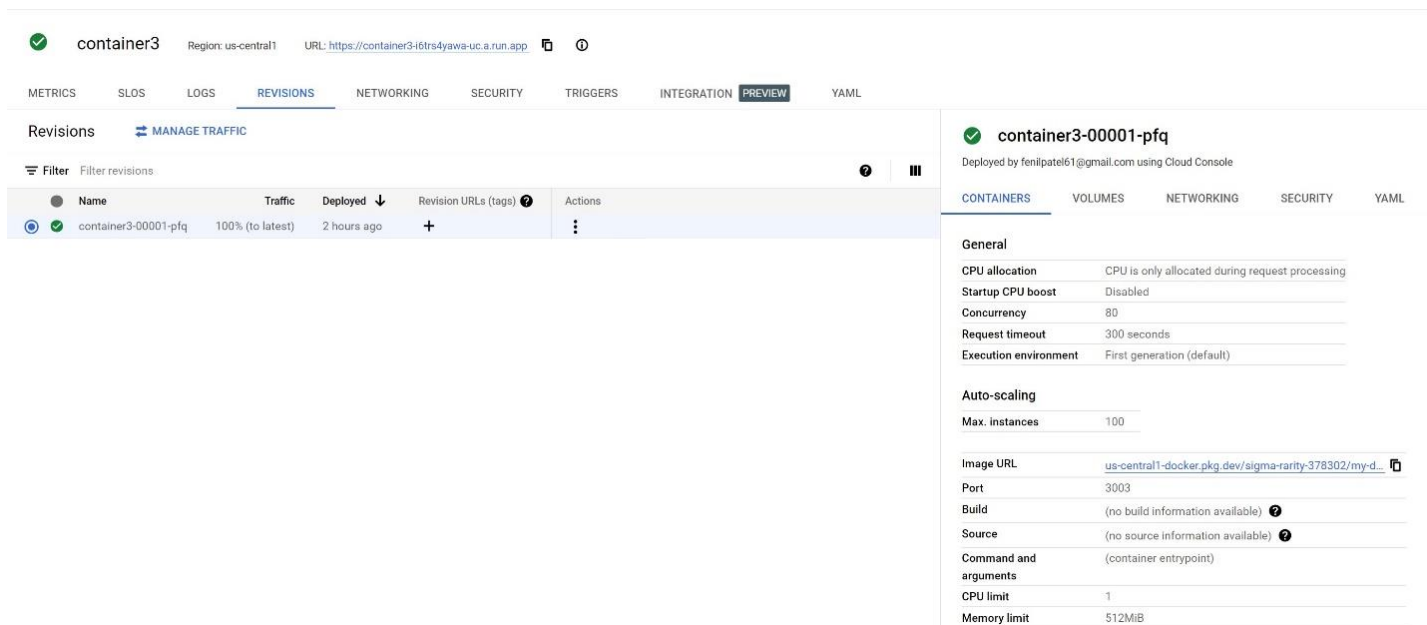
CPU allocation	CPU is only allocated during request processing
Startup CPU boost	Disabled
Concurrency	80
Request timeout	300 seconds
Execution environment	First generation (default)

Auto-scaling

Max. instances	100
----------------	-----

Image URL us-central1-docker.pkg.dev/sigma-rarity-378302/my-d...
Port 3002
Build (no build information available)
Source (no source information available)
Command and arguments (container entrypoint)
CPU limit 1
Memory limit 512MiB

Fig 43: Same configuration followed for creating a cloud run service for container 2 Image, and it is running at port 3002



The screenshot shows the Google Cloud Run console for a service named 'container3'. The 'REVISIONS' tab is active, displaying a table with one revision: 'container3-00001-pfq'. This revision is at 100% traffic, deployed 2 hours ago, and is the latest. The right-hand panel shows the configuration for this revision, including general settings (CPU allocation, startup boost, concurrency, timeout, environment) and auto-scaling (max instances: 100). The image URL is 'us-central1-docker.pkg.dev/sigma-rarity-378302/my-d...'. The port is 3003. The build and source information is unavailable.

Name	Traffic	Deployed	Revision URLs (tags)	Actions
container3-00001-pfq	100% (to latest)	2 hours ago	+	⋮

container3-00001-pfq
Deployed by fenilpatel61@gmail.com using Cloud Console

General

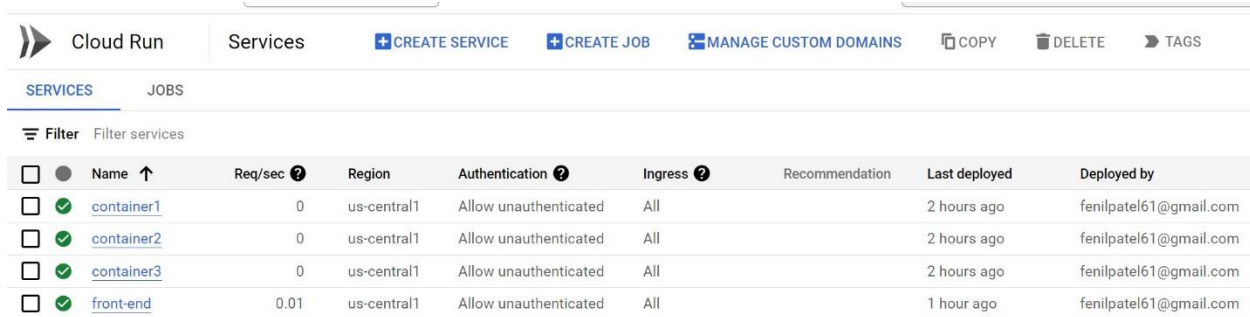
CPU allocation	CPU is only allocated during request processing
Startup CPU boost	Disabled
Concurrency	80
Request timeout	300 seconds
Execution environment	First generation (default)

Auto-scaling

Max. instances	100
----------------	-----

Image URL us-central1-docker.pkg.dev/sigma-rarity-378302/my-d...
Port 3003
Build (no build information available)
Source (no source information available)
Command and arguments (container entrypoint)
CPU limit 1
Memory limit 512MiB

Fig 44: Same configuration followed for creating a cloud run service for container 2 Image, and it is running at port 3003



The screenshot shows the Google Cloud Run 'Services' page. At the top, there are navigation links for 'Cloud Run', 'Services', and buttons for '+ CREATE SERVICE', '+ CREATE JOB', and 'MANAGE CUSTOM DOMAINS'. On the right, there are icons for 'COPY', 'DELETE', and 'TAGS'. Below the navigation bar, there are tabs for 'SERVICES' and 'JOBS'. A 'Filter' button is present. The main table lists four services: 'container1', 'container2', 'container3', and 'front-end'. Each row includes a checkbox, a status icon (green checkmark), the service name, request rate (Req/sec), region, authentication type, ingress type, recommendation, last deployment time, and the user who deployed it.

<input type="checkbox"/>	<input checked="" type="checkbox"/>	Name ↑	Req/sec ?	Region	Authentication ?	Ingress ?	Recommendation	Last deployed	Deployed by
<input type="checkbox"/>	<input checked="" type="checkbox"/>	container1	0	us-central1	Allow unauthenticated	All		2 hours ago	fenilpatel61@gmail.com
<input type="checkbox"/>	<input checked="" type="checkbox"/>	container2	0	us-central1	Allow unauthenticated	All		2 hours ago	fenilpatel61@gmail.com
<input type="checkbox"/>	<input checked="" type="checkbox"/>	container3	0	us-central1	Allow unauthenticated	All		2 hours ago	fenilpatel61@gmail.com
<input type="checkbox"/>	<input checked="" type="checkbox"/>	front-end	0.01	us-central1	Allow unauthenticated	All		1 hour ago	fenilpatel61@gmail.com

Fig 45: All the Services running on Cloud Run for the Front end, all three containers

References:

Deploying Containers to Cloud Run :- <https://medium.com/google-cloud/deploying-containers-to-cloud-run-in-5mins-b03f1d8d4a64>

How To Write Unit Tests In NodeJS With JEST Test Library :- <https://medium.com/bb-tutorials-and-thoughts/how-to-write-unit-tests-in-nodejs-with-jest-test-library-a201658829c7>