

Assignment 2

Part A

Meet Patel (B00899516)

Dalhousie University

Subject

CSCI 5410 (Serverless Data
Processing)

Professor

Dr. Saurabh Dey

Project Git Repository

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

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

Firebase Firestore Setup

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

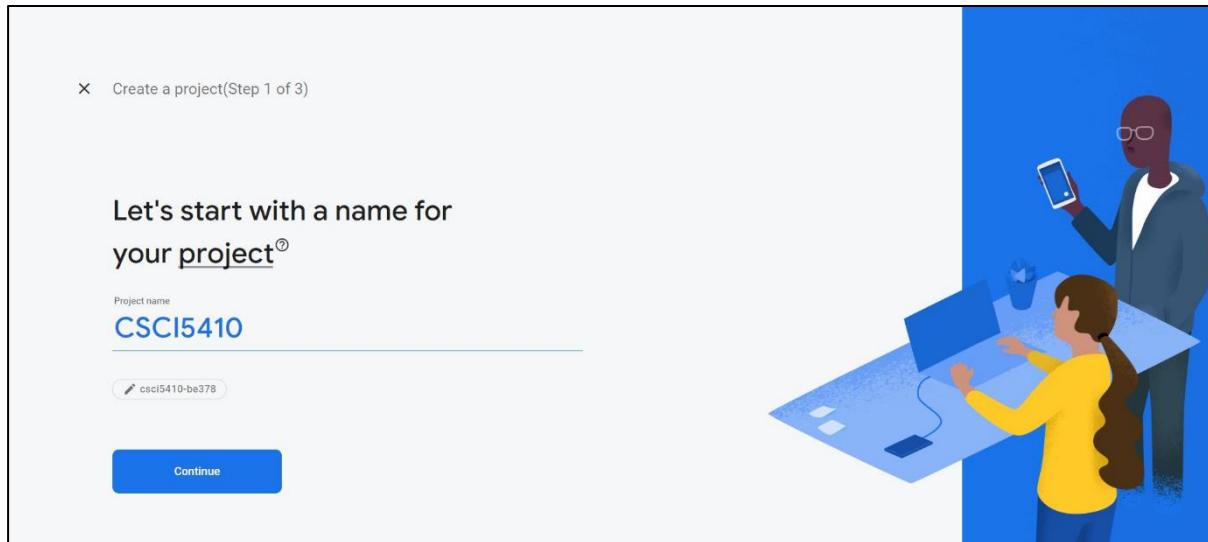


Figure 1: Firebase Firestore project creation with project name **CSCI5410**

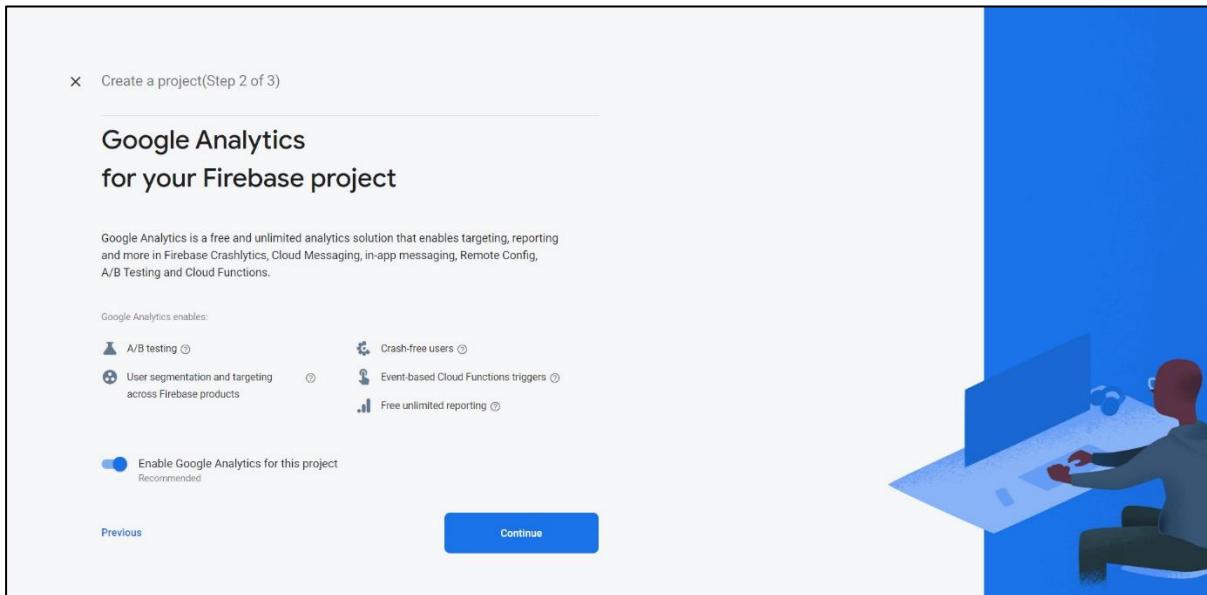


Figure 2: Firebase Firestore project creation with project name **CSCI5410**

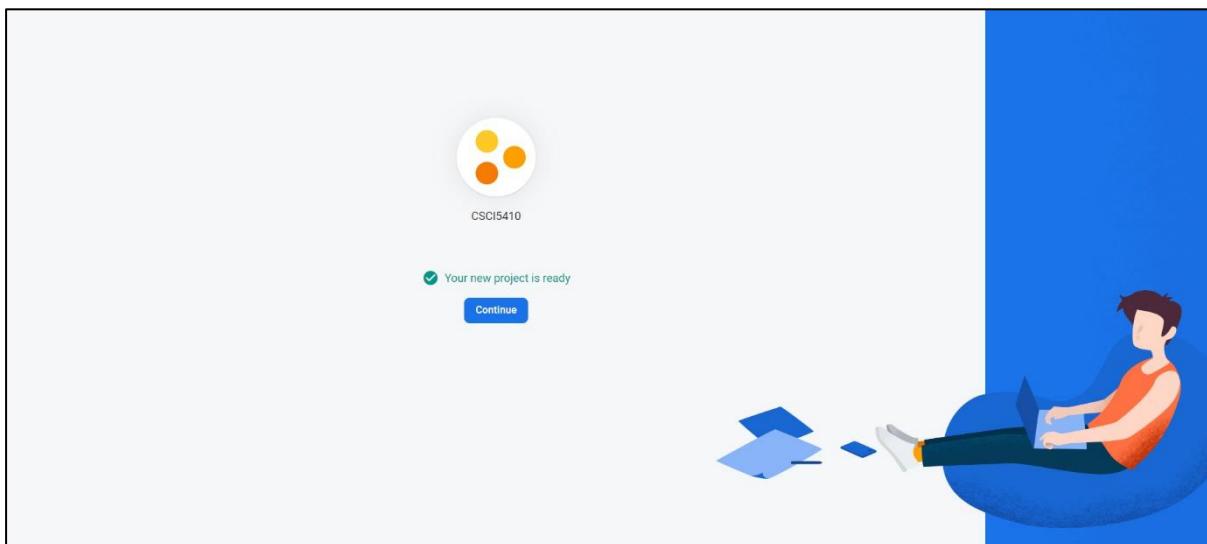


Figure 3: Firebase Firestore project creation with project name **CSCI5410**

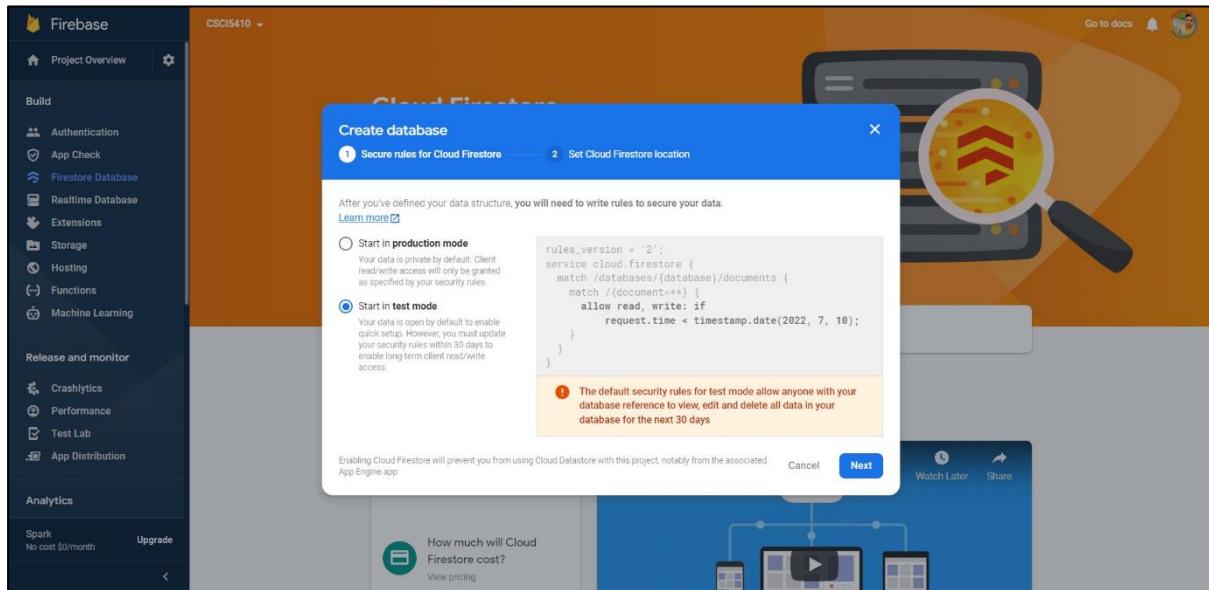


Figure 4: Firebase Firestore database creation

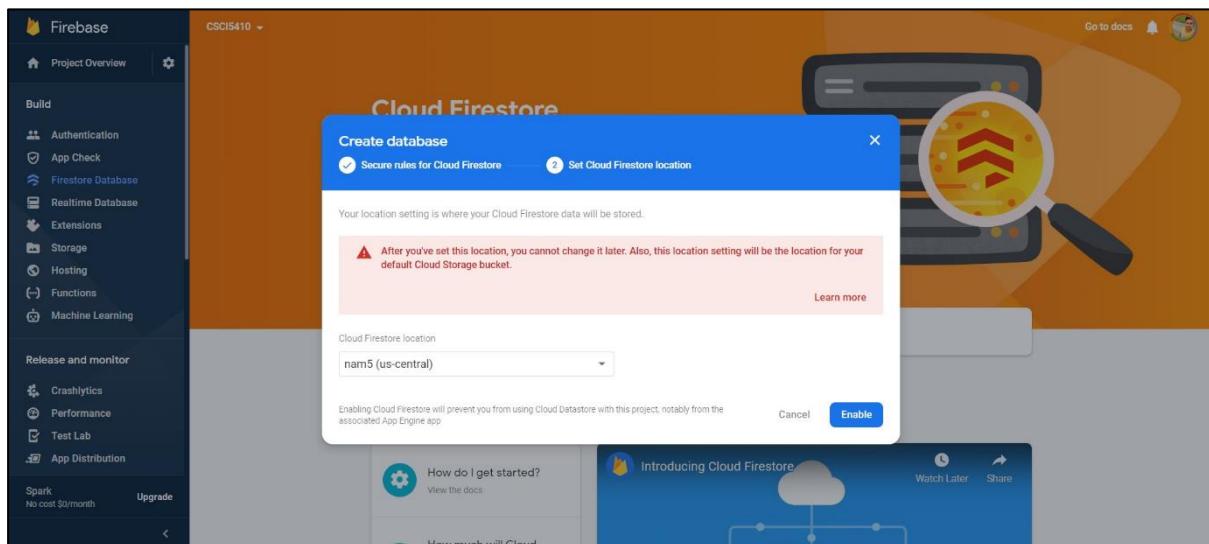


Figure 5: Firebase Firestore database creation

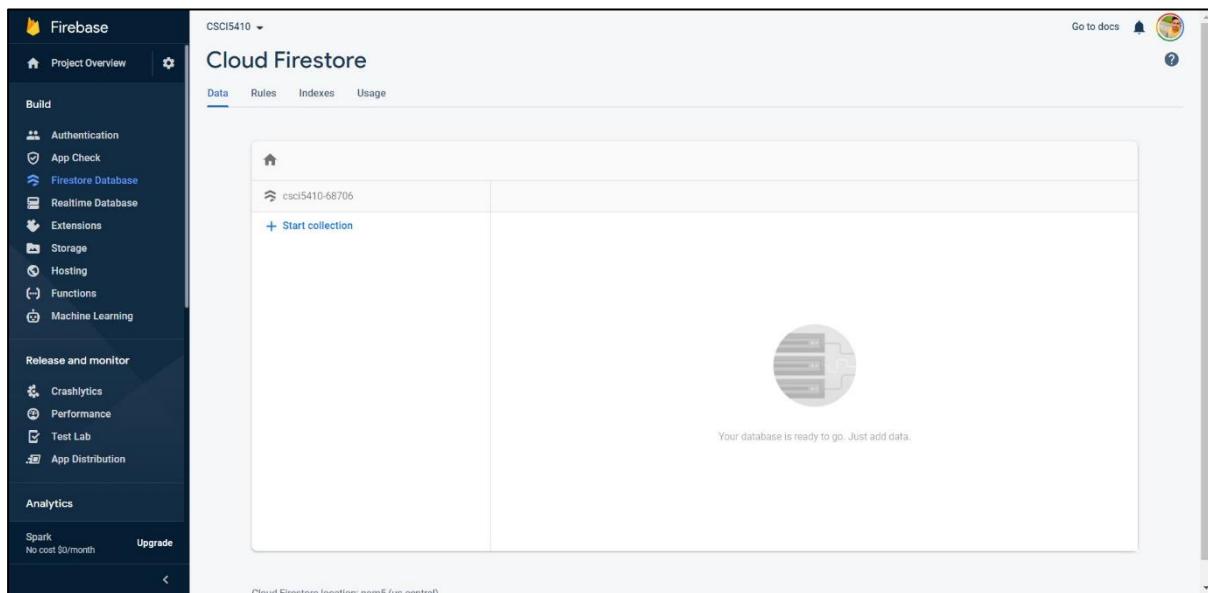


Figure 6: Firestore with empty database (Without users)

Creation of docker file for react application

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

The screenshot shows the Visual Studio Code interface with the Dockerfile open in the editor. The code is as follows:

```
FROM node:13.12.0-alpine
WORKDIR /container_1
ENV PATH /container_1/node_modules/.bin:$PATH
COPY package.json .
COPY package-lock.json .
RUN npm install
COPY . .
CMD ["npm", "start"]
```

The Explorer sidebar shows a project structure with a 'CONTAINER_1' folder containing 'node_modules', 'public', and 'src' directories. The 'src' directory contains 'component' and 'css' subfolders, along with files like 'Register.js', 'Register.css', 'App.css', 'App.js', 'firebaseConfiguration.js', 'index.css', 'index.js', and '.gitignore'. There are also 'Dockerfile', 'package-lock.json', 'package.json', and 'README.md' files.

The Terminal tab shows a PowerShell session with the command 'PS C:\Users\Alien\OneDrive\Desktop\Serverless\Assignment_1\Assignment2\assignment_2_code\container_1>'.

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

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

The screenshot shows the Visual Studio Code interface with the Dockerfile open in the editor. The code is as follows:

```
FROM node:13.12.0-alpine
WORKDIR /container_2
ENV PATH /container_2/node_modules/.bin:$PATH
COPY package.json .
COPY package-lock.json .
RUN npm install
COPY . .
CMD ["npm", "start"]
```

The Explorer sidebar shows a project structure with a 'CONTAINER_2' folder containing 'node_modules', 'public', and 'src' directories. The 'src' directory contains 'component' and 'css' subfolders, along with files like 'Login.js', 'Login.css', 'App.css', 'App.js', 'firebaseConfiguration.js', 'index.css', 'index.js', and '.gitignore'. There are also 'Dockerfile', 'package-lock.json', 'package.json', and 'README.md' files.

The Terminal tab shows a PowerShell session with the command 'PS C:\Users\Alien\OneDrive\Desktop\Serverless\Assignment_1\Assignment2\assignment_2_code\container_2>'.

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

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

The screenshot shows the Visual Studio Code interface with the Dockerfile open in the main editor area. The Dockerfile content is as follows:

```
FROM node:13.12.0-alpine
WORKDIR /container_3
ENV PATH /container_3/node_modules/.bin:$PATH
COPY package.json .
COPY package-lock.json .
RUN npm install
COPY . .
CMD ["npm", "start"]
```

The Explorer sidebar on the left shows the project structure for 'CONTAINER_3' containing files like 'node_modules', 'public', 'src', 'component', 'css', 'App.css', 'App.js', 'firebaseConfiguration.js', 'index.css', 'index.js', '.gitignore', 'Dockerfile', 'package-lock.json', 'package.json', and 'README.md'. The Terminal tab at the bottom shows a PowerShell session with the command 'PS C:\Users\Alien\OneDrive\Desktop\Serveless\Assignment_1\Assignment2\assignment_2_code\container_3>'.

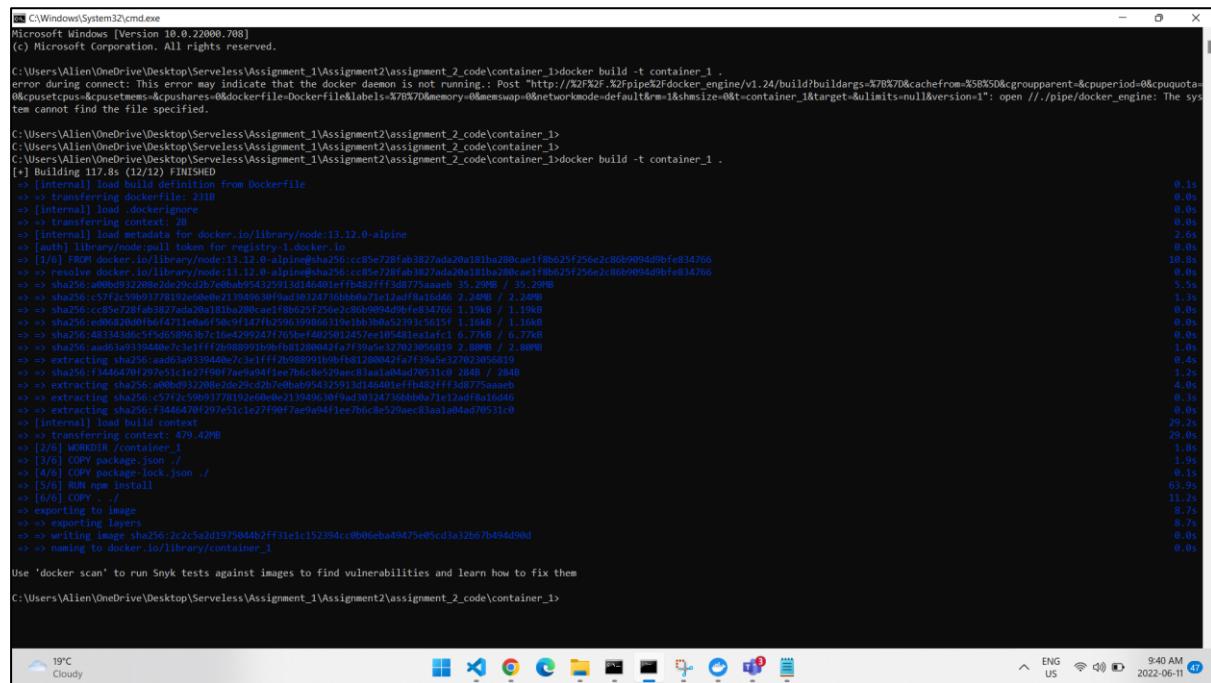
Figure 9: Docker file for UserProfile react application (container_3)

Creation of Containers Images using docker

Container 1 – Registration React Application

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

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



```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.708]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Alien\OneDrive\Desktop\Serverless\Assignment_1\Assignment_2_code\container_1>docker build -t container_1 .
error during connect: This error may indicate that the docker daemon is not running.: Post "http://<%2>%F.%2>pipe%2fdocker_engine/v1.24/build?buildargs=%5B%5D&cgroupparent=&cpuperiod=0&cpupquota=0&cpusetcpus=&cpusetcpu=&cpushares=&label=dockerfileLabel=%7E%7E%0memory=&memswap=&networkmode=default&r=&s=1&target=&ulimits=null&version=1": open //./pipe/docker_engine: The system cannot find the file specified.

C:\Users\Alien\OneDrive\Desktop\Serverless\Assignment_1\Assignment_2_code\container_1>C:\Users\Alien\OneDrive\Desktop\Serverless\Assignment_1\Assignment_2_code\container_1>docker build -t container_1 .
[+] Building 117.8s (12/12) FINISHED
--> [internal] load build definition from Dockerfile
--> transferring dockerfile: 231B
--> [internal] load .dockerignore
--> transferring context: 0B
--> [internal] load manifest for docker.io/library/node:13.12.0-alpine
--> [auth] library/library:pull token for registry.docker.io
--> [1/6] FROM docker.io/library/node:13.12.0-alpine@sha256:cc85e728fabab827ade20a181ba280cae1f8b625f256e2c8619094d9bfef834760
--> sha256:a00bd932208e2de9c2b7e0ba954291913d146401effb482fff3d8755aae 35.29M / 35.29M
--> sha256:c572c599d177819e2e00e21949e3019ad30124710bbba71e2adff8a1ed6d 2.49M / 2.49M
--> sha256:c85e728fabab827ade20a181ba280cae1f8b625f256e2c8619094d9bfef834766 1.19K / 1.19K
--> sha256:cd0082080bd16d4711e0af93dc47f7b598399661965bd04393c519 1.10M / 1.10M
--> sha256:6833151540935580865839a760824f16a22805760f46021f7765d5c5b151 1.77K / 1.77K
--> sha256:86a0393704d0009e203f12004f9e7f799fe2799259e6019 2.09M / 2.09M
--> sha256:0e9613394400e7c3e1442689901b0fbfb812800042e77305a532701305619 0.45M / 0.45M
--> extracting sha256:ad61a03394400e7c3e1442689901b0fbfb812800042e77305a532701305619
--> sha256:f3446470f297e51c27f907ae9a04flee7b6c8e529aec83aa1e94ad70531c0 2048 / 2048
--> extracting sha256:a00bd932208e2de9c2b7e0ba954325913d146401effb482fff3d8775aaeb
--> extracting sha256:c572c599d177819e2e00e21949e3019ad30124710bbba71e2adff8a1ed6d
--> extracting sha256:f3446470f297e51c27f907ae9a04flee7b6c8e529aec83aa1e94ad70531c0
[internal] load build context
--> transferring context: 479.429B
--> [2/2] WORKDIR container_1
--> [3/3] COPY package.json ./
--> [4/4] COPY package-lock.json ./
--> [5/5] RUN npm install
--> [6/6] COPY ./
--> exporting to image
--> exporting layers
--> writing image sha256:2c2c5a2d197504db2ff31e1c152394c:0b06eba49475e95cd3a32b67b494d98d
--> naming to docker.io/library/container_1

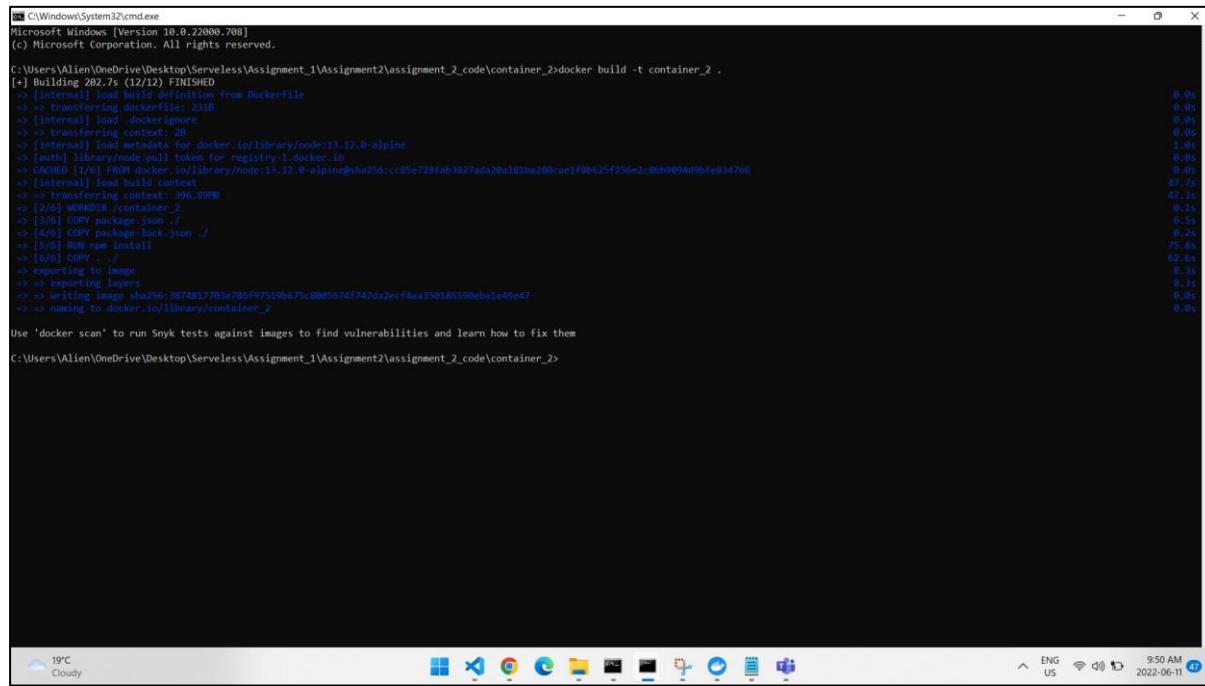
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
C:\Users\Alien\OneDrive\Desktop\Serverless\Assignment_1\Assignment_2\assignment_2_code\container_1>
```

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

Container 2 – Login React Application

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

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



```
[+] Building 202.7s (12/12) FINISHED
--> [internal] load build definition from Dockerfile
--> transferring dockerfile: 231B
--> [internal] load dockerignore
--> => transferring context: 2B
--> [internal] load build context
--> => transferring context: 396.89MB
--> [2/6] WORKDIR /container_2
--> [3/6] COPY package.json .
--> [4/6] COPY package-lock.json .
--> [5/6] RUN npm install
--> [6/6] COPY . .
--> => exporting layers
--> => writing image sha256:3874817783e780f97519b675c8809674f747da2ecf4ea35018059bebe1e49e47
--> => naming to docker.io/library/container_2

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

C:\Users\Alien\OneDrive\Desktop\Serverless\Assignment_1\Assignment_2_code\container_2>
```

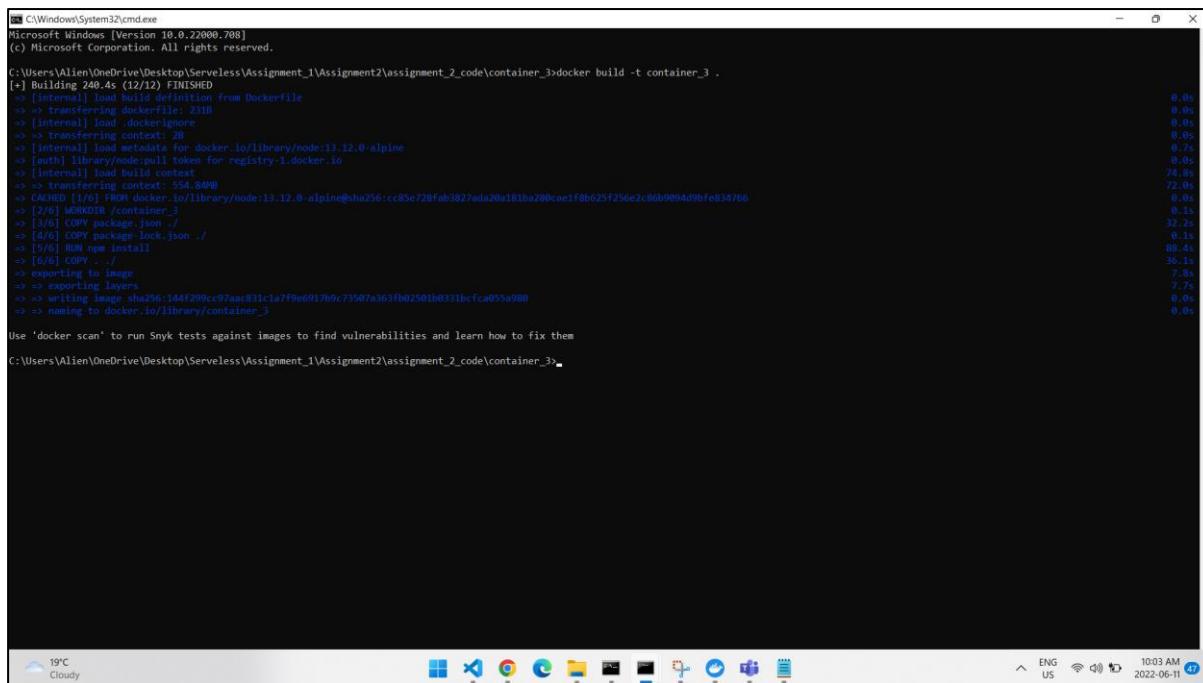
Figure 11:Successful creation of the docker image with name **Container_2** (Login React App)

Container 3 – UserProfile React Application

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

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

CSCI 5410: Assignment #2 (Part A) | Meet Patel B00899516



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.708]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Alien\OneDrive\Desktop\Serveless\Assignment_1\Assignment2\assignment_2_code\container_3>docker build -t container_3 .

[+] Building 240.4s (12/12) FINISHED
--> [internal] load build definition from Dockerfile
--> <-- transferring dockerfile: 23B
--> <-- transferring dockerignore
--> <-- transferring context: 2B
--> [internal] load metadata for docker.io/library/node:11.12.0-alpine
--> [auth] library/node:pull token for registry-1.docker.io
--> [internal] load build context
--> <-- transferring context: 554.84MB
--> CACHED [1/6] FROM docker.io/library/node:11.12.0-alpine@sha256:cc85ef28fab3827ada20a181ba700cae1f8b625f256e2c88b9094d96fe834760
--> [2/6] WORKDIR /container_3
--> [3/6] COPY package.json ./
--> [4/6] COPY package-lock.json ./
--> [5/6] RUN npm install
--> [6/6] COPY . .
--> <-- exporting to image
--> <-- exporting layers
--> <-- writing image sha256:144f299cc97aac831cia7ff9e691709c73507a363fb02501fb031bcfcfa055ad000
--> <-- naming to docker.io/library/container_3

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
C:\Users\Alien\OneDrive\Desktop\Serveless\Assignment_1\Assignment2\assignment_2_code\container_3>
```

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

Successfully Creation of 3 Docker Images

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

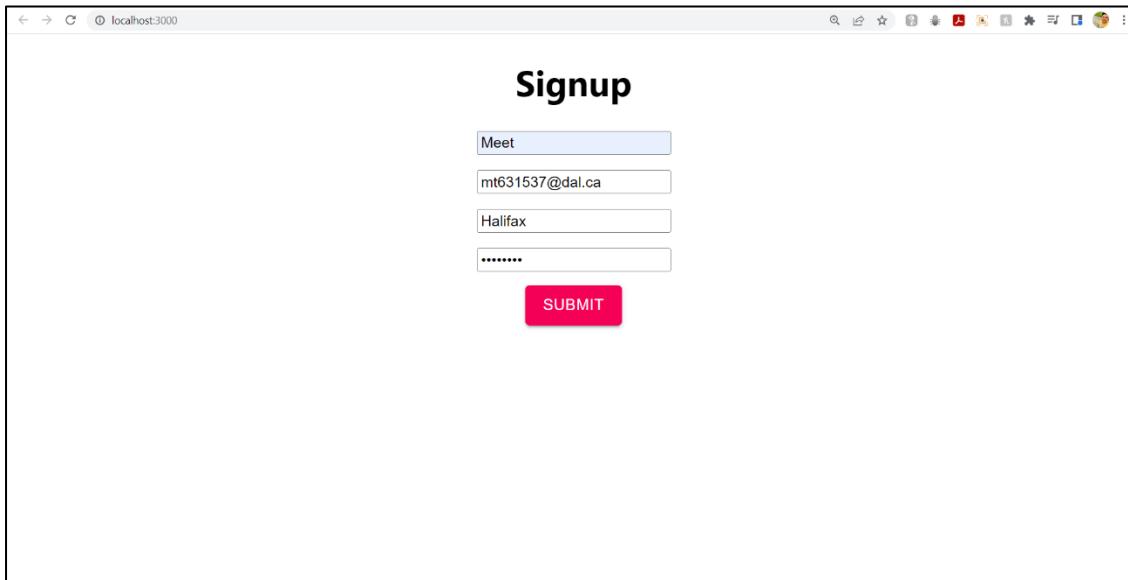
The screenshot shows the Docker Desktop interface with the 'Images' tab selected. The main pane displays a table of images on disk, categorized under 'LOCAL'. A red box highlights the last three entries in the table:

NAME	TAG	IMAGE ID	CREATED	SIZE
alpine/git	IN USE latest	241890ad72b1	23 days ago	38.23 MB
container_1	latest	594481675c95	about 3 hours ago	901.11 MB
container_2	latest	477d3a2b54c0	about 3 hours ago	818.61 MB
container_3	latest	9c29d285d344	about 3 hours ago	992.13 MB
my_updated_ubuntu	latest	4c3fcfc2b30	6 days ago	77.81 MB
ubuntu	IN USE latest	d2e4e1f51132	about 1 month ago	77.81 MB
us-east1-docker.pkg.dev/data54...	mp	594481675c95	about 3 hours ago	901.11 MB
us-east1-docker.pkg.dev/data54...	<none>	2c2c5a2d1975	about 4 hours ago	901.11 MB
us-east1-docker.pkg.dev/data54...	mp	477d3a2b54c0	about 3 hours ago	818.61 MB

Website Working (Docker Image Localhost) with Firebase Images

1. Registration React Application

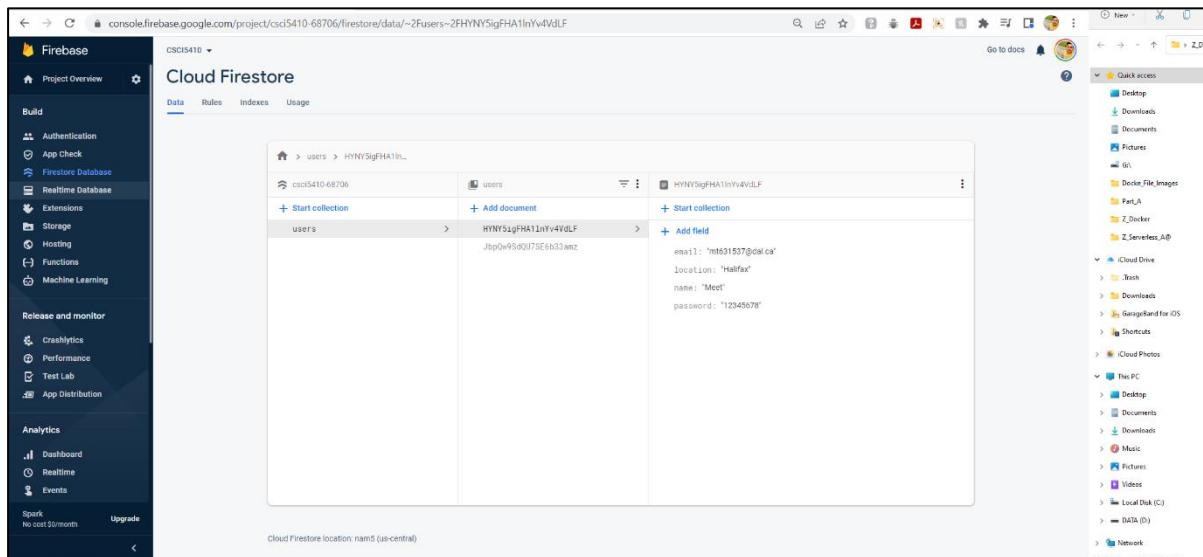
Figure 13 shows the **Registration Page (container_1)** using docker container on Localhost



The screenshot shows a registration page titled "Signup". It contains four input fields: "Meet", "mt631537@dal.ca", "Halifax", and a password field with five asterisks. Below the fields is a red "SUBMIT" button.

Figure 13: Registration page showed using container_1

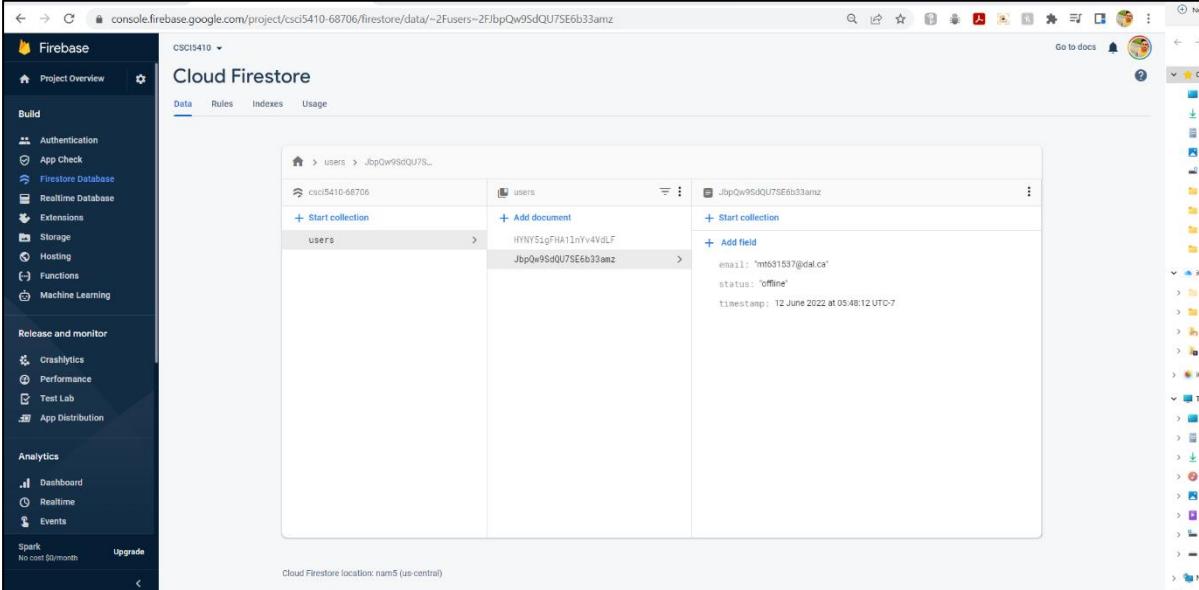
Figure 14 and 15 shows the successful registration of the user with status set to **offline** and the data is stored in firebase database



The screenshot shows the Firebase Cloud Firestore console. The left sidebar shows the project overview and various database options like Realtime Database, Firestore Database, and Storage. The main area shows the "Cloud Firestore" section with a "Data" tab selected. Under the "users" collection, there is a document with the ID "HYNY5igfHA1InYv4VdLF". The document contains the following data:

Field	Type	Value
email	String	"mt631537@dal.ca"
location	String	"Halifax"
name	String	"Meet"
password	String	"12345678"

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



The screenshot shows the Firebase Cloud Firestore interface. On the left, the navigation sidebar includes sections for Authentication, App Check, Firestore Database, Realtime Database, Extensions, Storage, Hosting, Functions, Machine Learning, Release and monitor, Crashlytics, Performance, Test Lab, and App Distribution. The Analytics section is also visible. The main area is titled "Cloud Firestore" and shows a "users" collection. A specific document, "JbpQw9SdQU7SE6b33amz", is selected. The document details are as follows:

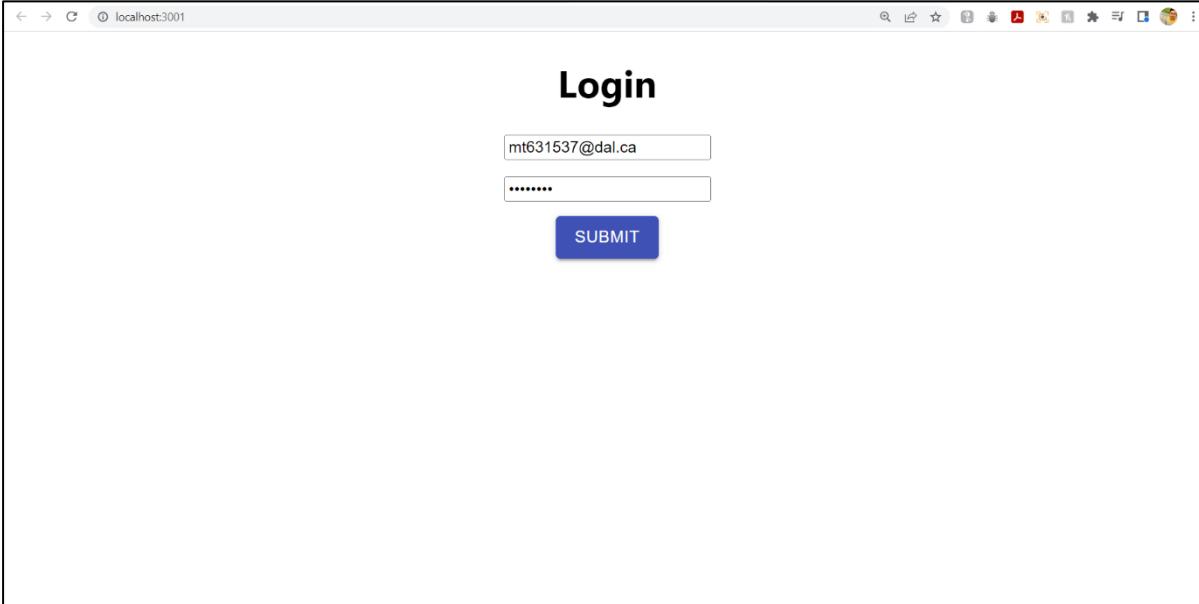
email	status	timestamp
"mt631537@dal.ca"	"offline"	12 June 2022 at 05:48:12 UTC-7

Cloud Firestore location: nam5 (us-central)

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

2. Login React Application

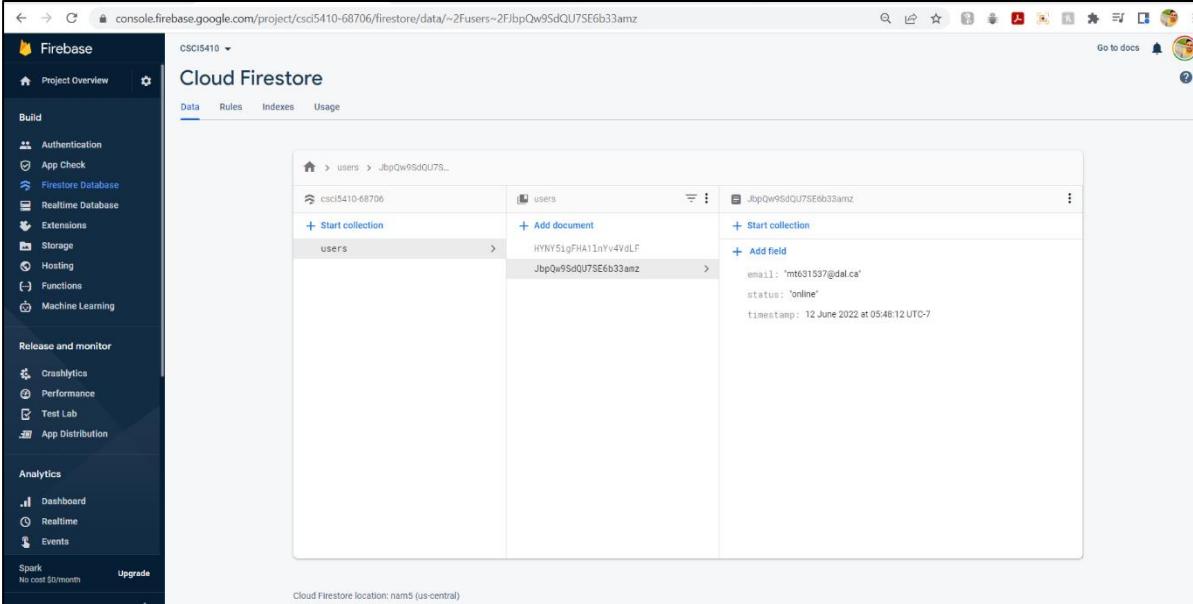
Figure 16 shows the **Login Page (container_2)** using docker container on Localhost



The screenshot shows a browser window with the URL "localhost:3001". The page title is "Login". It contains two input fields: one for "email" containing "mt631537@dal.ca" and one for "password" containing "*****". Below the inputs is a blue "SUBMIT" button.

Figure 16: Login page showed using container_2

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

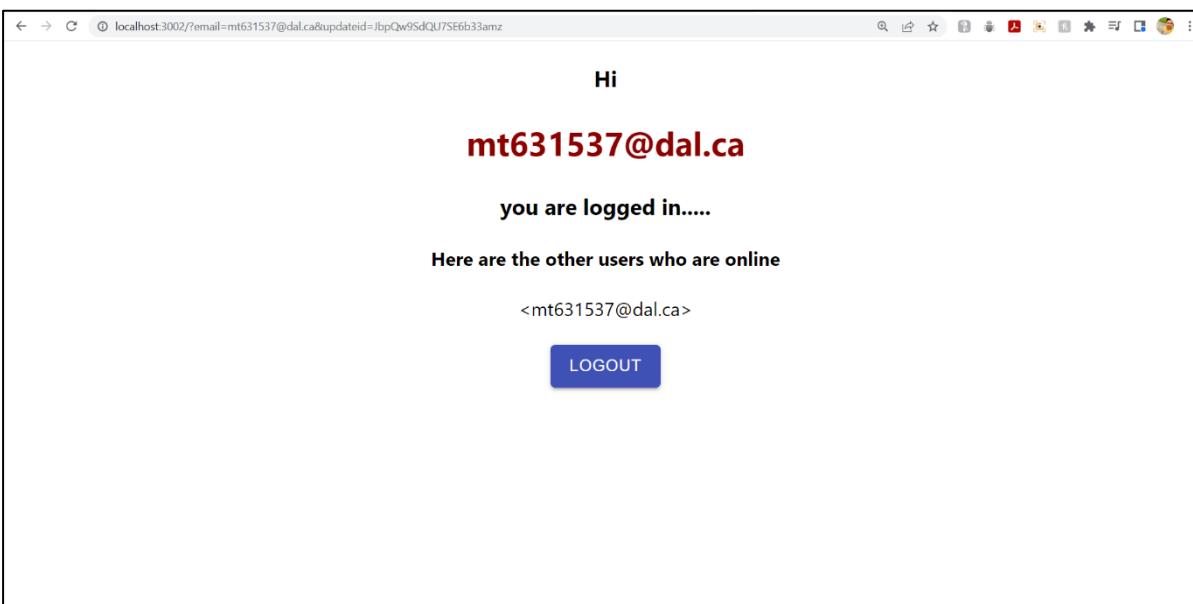


The screenshot shows the Firebase Cloud Firestore interface. On the left, the navigation sidebar includes sections for Authentication, App Check, Firestore Database, Realtime Database, Extensions, Storage, Hosting, Functions, Machine Learning, Release and monitor, Crashlytics, Performance, Test Lab, and App Distribution. The main area is titled "Cloud Firestore" and shows a hierarchical view of collections and documents under the "users" collection. A specific document, "JbpQw9SdQU7SE6b33amz", is expanded, displaying its fields: email ("mt631537@dal.ca"), status ("online"), and timestamp ("12 June 2022 at 05:48:12 UTC-7").

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

3. UserProfile React Application

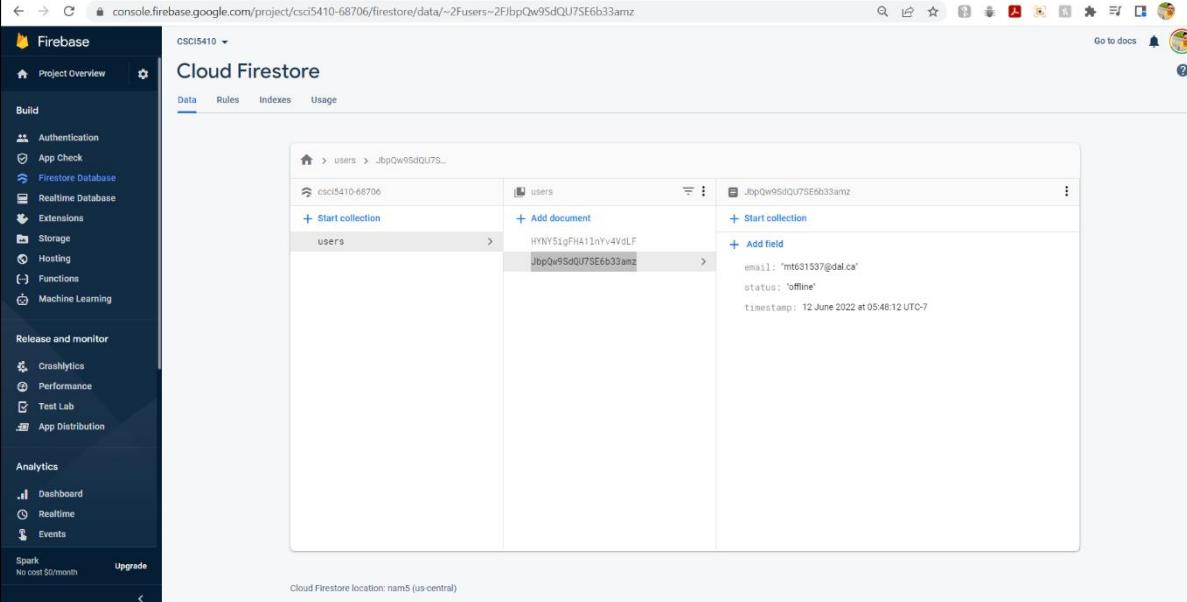
Figure 18 shows the **UserProfile Page (container_3)** using docker container on Localhost. This page shows the logged in user email address and a logout button.



The screenshot shows a web browser window with the URL "localhost:3002/?email=mt631537@dal.ca&updateid=JbpQw9SdQU7SE6b33amz". The page content includes the text "Hi", the user's email "mt631537@dal.ca", the message "you are logged in.....", and "Here are the other users who are online". It also lists the email "<mt631537@dal.ca>" and features a blue "LOGOUT" button.

Figure 18: UserProfile page showed using container_3

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



The screenshot shows the Firebase Cloud Firestore interface. On the left, the navigation sidebar includes options like Authentication, Firestore Database, Realtime Database, Storage, Hosting, Functions, Machine Learning, Crashlytics, Performance, Test Lab, App Distribution, and Analytics. The main area is titled "Cloud Firestore" and shows a collection named "users". Inside the "users" collection, there is a document with the ID "JbpQw9SdQU7SE6b33amz". The document details are as follows:

email	status	timestamp
"mt631537@dal.ca"	"offline"	12 June 2022 at 05:48:12 UTC-7

At the bottom of the interface, it says "Cloud Firestore location: nam5 (us-central)".

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

Google Could CLI Setup (Google Cloud SDK)

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



Figure 20: Google CLI Setup Step-1

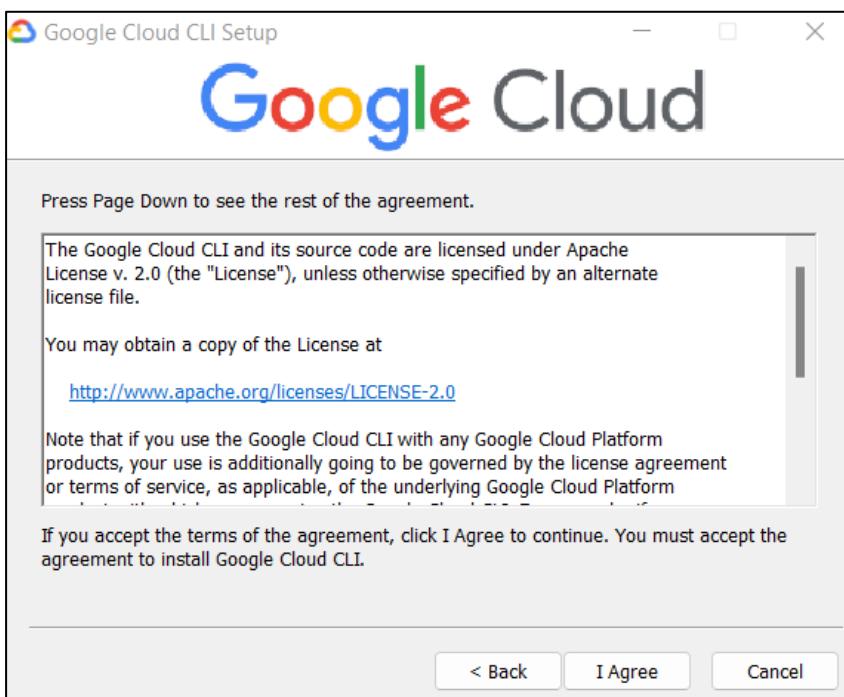


Figure 21: Google CLI Setup Step-2

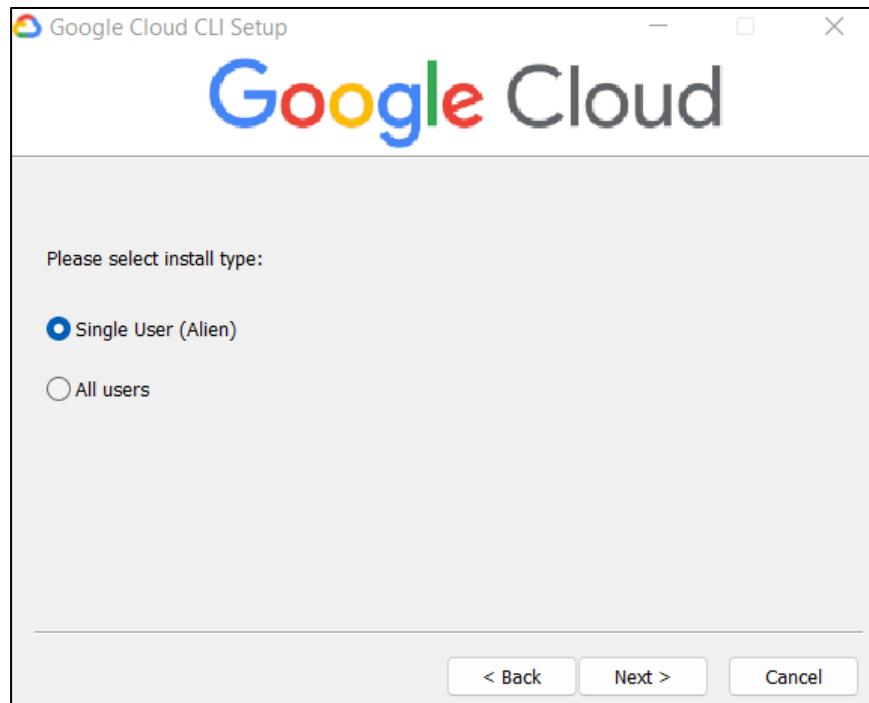


Figure 22: Google CLI Setup Step-3

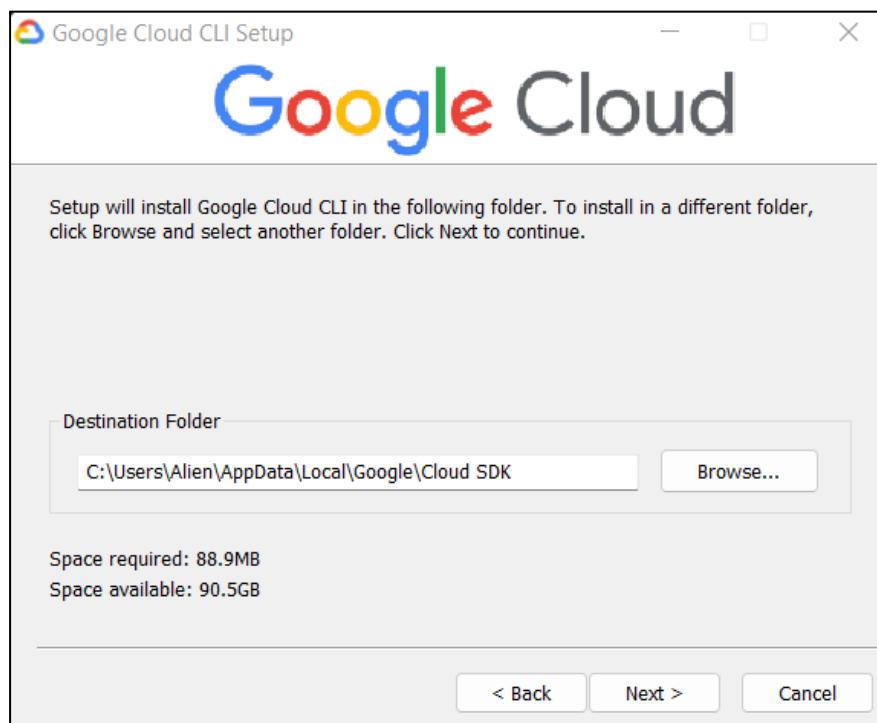


Figure 23: Google CLI Setup Step-4

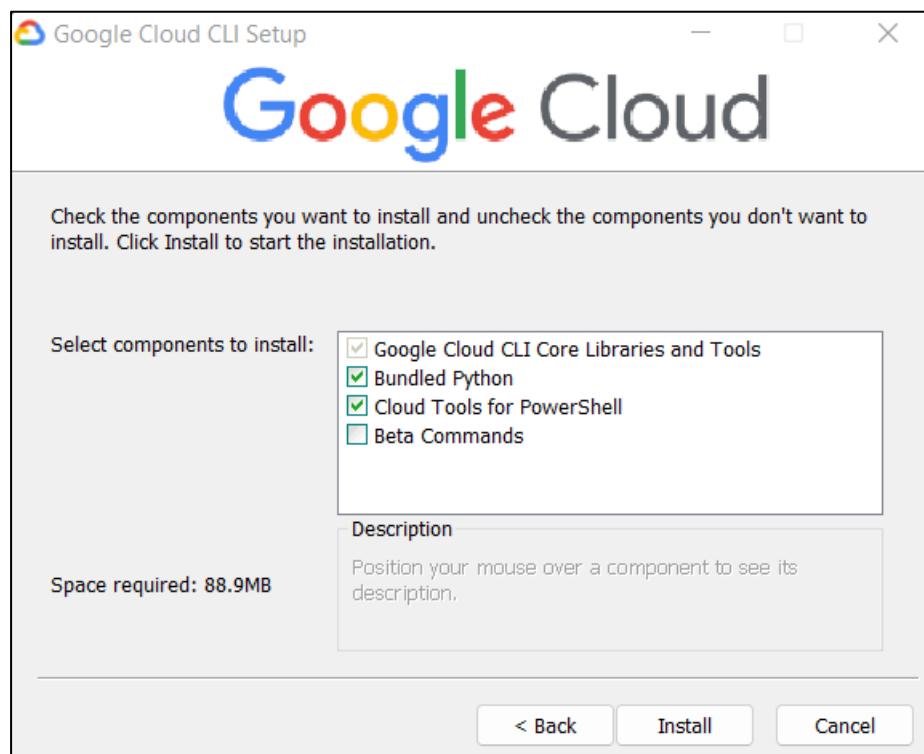


Figure 24: Google CLI Setup Step-5

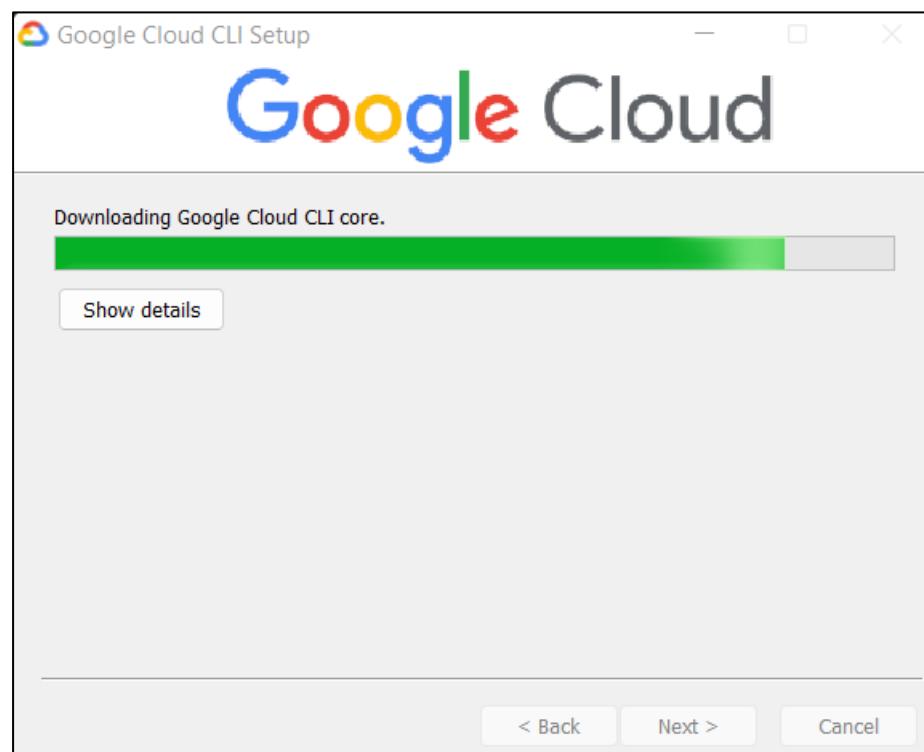


Figure 25: Google CLI Setup Step-6

Google Container Registry

Steps:

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

The screenshot shows a Windows Command Prompt window titled 'cmd.exe' with the path 'C:\WINDOWS\system32\cmd.exe'. The window displays the following text:

```
Your browser has been opened to visit:  
https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=325594059.apps.googleusercontent.com&redirect_uri=http%3A%2F%2Flocalhost%3A8085%2F&scope=openid+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.email+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fappengine.admin+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fservice.login+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcompute+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Faccounts.reauth&state=DRpZdexzXkm0tXZYluZvjQlyzVlaw&access_type=offline&code_challenge=S7_NvE3ltbtjtEnx;z5jeRFX-40vBkS2wO38vvCOVlk1k&code_challenge_method=S256  
You are logged in as: [mp.meetpatel98@gmail.com].  
Pick cloud project to use:  
[1] csci5410-68706  
[2] custom-mix-340905  
[3] data5408-b00899516  
[4] Enter a project ID  
[5] Create a new project  
Please enter numeric choice or text value (must exactly match list item): [1]  
Please enter a value between 1 and 5, or a value present in the list: 1  
Your current project has been set to: [csci5410-68706].  
Not setting default zone/region (this feature makes it easier to use  
[gcloud compute] by setting an appropriate default value for the  
--zone and --region flag).  
See https://cloud.google.com/compute/docs/gcloud-compute section on how to set  
default compute region and zone manually. If you would like [gcloud init] to be  
able to do this for you the next time you run it, make sure the  
Compute Engine API is enabled for your project on the  
https://console.developers.google.com/apis page.  
Created a default .boto configuration file at [C:\Users\Alien].boto]. See this file and  
[https://cloud.google.com/storage/docs/gsutil/commands/config] for more  
information about configuring Google Cloud Storage.  
Your Google Cloud SDK is configured and ready to use!  
* Commands that require authentication will use mp.meetpatel98@gmail.com by default  
* Commands will reference project csci5410-68706 by default  
Run 'gcloud help config' to learn how to change individual settings.  
This gcloud configuration is called [default]. You can create additional configurations if you work with multiple accounts and/or projects.  
Run 'gcloud topic configurations' to learn more.  
Some things to try next:  
* Run 'gcloud --help' to see the Cloud Platform services you can interact with. And run 'gcloud help COMMAND' to get help on any gcloud command.  
* Run 'gcloud topic --help' to learn about advanced features of the SDK like arg files and output formatting.  
* Run 'gcloud cheat-sheet' to see a roster of go-to gcloud commands.  
C:\Users\Alien\AppData\Local\Google\Cloud SDK>
```

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

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

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

```

C:\Windows\system32\cmd.exe
C:\Users\Alien\AppData\Local\Google\Cloud SDK>gcloud auth configure-docker us-east1-docker.pkg.dev
Adding credentials for: us-east1-docker.pkg.dev
After update, the following will be written to your Docker config file located at [C:\Users\Alien\.docker\config.json]:
{
  "credHelpers": {
    "us-east1-docker.pkg.dev": "gcloud"
  }
}

Do you want to continue (Y/n)? y
Docker configuration file updated.

C:\Users\Alien\AppData\Local\Google\Cloud SDK>docker tag container_1 us-east1-docker.pkg.dev/data5408-b00899516/csci5410a2/container_1:mp
The push refers to repository [us-east1-docker.pkg.dev/data5408-b00899516/csci5410a2/container_1]
14f2866e11cc: Pushed
40c855ad2435: Pushed
008bb3bae05c: Pushed
87108564fffd: Pushed
77ff75358ce0: Pushed
65d358b7de11: Pushed
f97384e8ccbc: Pushed
d56e5e7e20148: Pushed
beee9f30bc1f: Pushed
mp: digest: sha256:d5f3ab716da4e233a81de093ad4e167bdb3d843659118f0caa5fa9ca580c7cc size: 2208

C:\Users\Alien\AppData\Local\Google\Cloud SDK>

```

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

```

C:\Windows\system32\cmd.exe
C:\Users\Alien\AppData\Local\Google\Cloud SDK>gcloud auth configure-docker us-east1-docker.pkg.dev
Adding credentials for: us-east1-docker.pkg.dev
After update, the following will be written to your Docker config file located at [C:\Users\Alien\.docker\config.json]:
{
  "credHelpers": {
    "us-east1-docker.pkg.dev": "gcloud"
  }
}

Do you want to continue (Y/n)? y
Docker configuration file updated.

C:\Users\Alien\AppData\Local\Google\Cloud SDK>docker tag container_1 us-east1-docker.pkg.dev/data5408-b00899516/csci5410a2/container_1:mp
The push refers to repository [us-east1-docker.pkg.dev/data5408-b00899516/csci5410a2/container_1]
14f2866e11cc: Pushed
40c855ad2435: Pushed
008bb3bae05c: Pushed
87108564fffd: Pushed
77ff75358ce0: Pushed
65d358b7de11: Pushed
f97384e8ccbc: Pushed
d56e5e7e20148: Pushed
beee9f30bc1f: Pushed
mp: digest: sha256:d5f3ab716da4e233a81de093ad4e167bdb3d843659118f0caa5fa9ca580c7cc size: 2208

C:\Users\Alien\AppData\Local\Google\Cloud SDK>docker tag container_2 us-east1-docker.pkg.dev/data5408-b00899516/csci5410a2/container_2:mp
The push refers to repository [us-east1-docker.pkg.dev/data5408-b00899516/csci5410a2/container_2]
d70659018853: Pushed
54704b73683: Pushed
74735e9f7f70: Pushed
9f572ebf040: Pushed
60d0358b7de11: Pushed
65d358b7de11: Layer already exists
f97384e8ccbc: Layer already exists
d56e5e7e20148: Layer already exists
beee9f30bc1f: Layer already exists
mp: digest: sha256:b1651e7ae8fb65fdef33f71f39253df1d9846443cd0c9a25e307cd47a5616a size: 2207

C:\Users\Alien\AppData\Local\Google\Cloud SDK>

```

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

CSCI 5410: Assignment #2 (Part A) | Meet Patel B00899516

```
C:\WINDOWS\SYSTEM32\cmd.exe
Docker configuration file updated.

C:\Users\Alien\AppData\local\Google\Cloud SDK>docker tag container_1 us-east1-docker.pkg.dev/data5408-b00899516/csci5410a2/container_1:mp
The push refers to repository [us-east1-docker.pkg.dev/data5408-b00899516/csci5410a2/container_1]
14f2866e11c1: Pushed
40c855ad2435: Pushed
008bb3bae05c: Pushed
87108564fffd: Pushed
77ff5358ce0: Pushed
65d358b7de11: Pushed
f97384e8ccbc: Pushed
d56e5e720148: Pushed
beee9f80bc1f: Pushed
mp: digest: sha256:ed5f3ab716da4e233a81de093ad4e167bdb3d843659118f0caa5fa9ca580c7cc size: 2208

C:\Users\Alien\AppData\local\Google\Cloud SDK>docker tag container_2 us-east1-docker.pkg.dev/data5408-b00899516/csci5410a2/container_2:mp
The push refers to repository [us-east1-docker.pkg.dev/data5408-b00899516/csci5410a2/container_2]
d70659018853: Pushed
54704b773683: Pushed
72723451040: Pushed
0915f251040: Pushed
6d0e2459a387: Pushed
65d358b7de11: Layer already exists
f97384e8ccbc: Layer already exists
d56e5e720148: Layer already exists
beee9f80bc1f: Layer already exists
mp: digest: sha256:b1651e7a80fc65def33f571f39253df1d19846443dc9a25e307cd47a5616a size: 2207

C:\Users\Alien\AppData\local\Google\Cloud SDK>docker tag container_3 us-east1-docker.pkg.dev/data5408-b00899516/csci5410a2/container_3:mp
The push refers to repository [us-east1-docker.pkg.dev/data5408-b00899516/csci5410a2/container_3]
e25993ab4c61: Pushed
73db358e3508: Pushed
811a174f3f9a: Pushed
f23a573f446f: Pushed
3e679fcdb63: Pushed
65d358b7de11: Layer already exists
f97384e8ccbc: Layer already exists
d56e5e720148: Layer already exists
beee9f80bc1f: Layer already exists
mp: digest: sha256:3dcf80f79f49a45e314b22e7f71ea6ea941f8e44454c80ade9fe416636a7a191 size: 2208

C:\Users\Alien\AppData\local\Google\Cloud SDK>
```

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

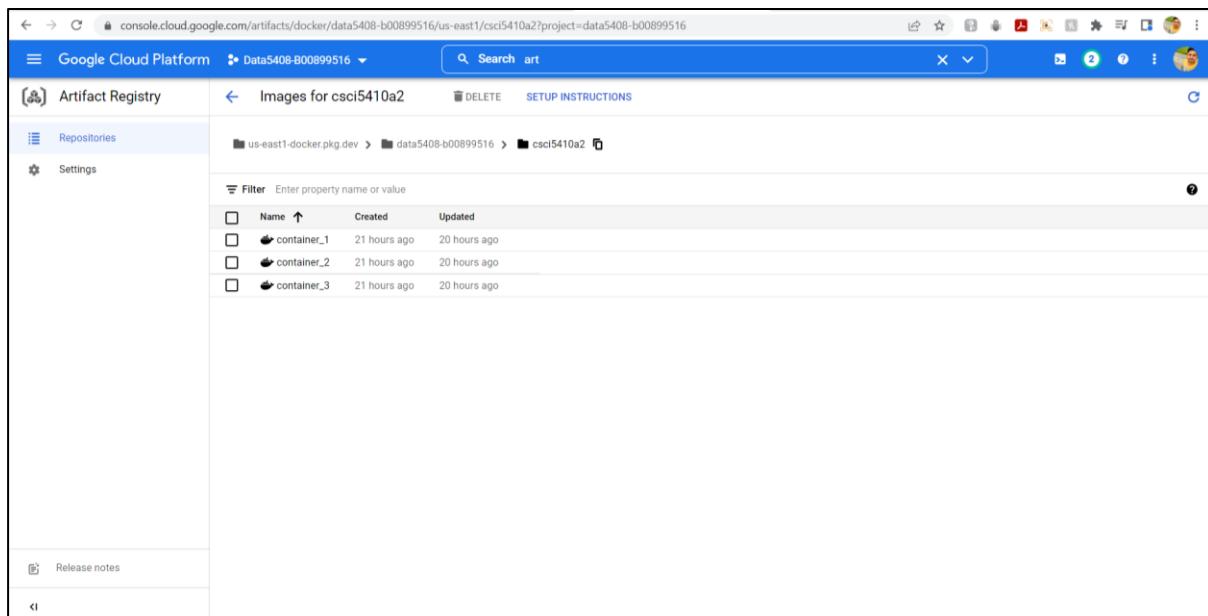


Figure 30: All Three Docker images uploaded to Google Container Registry (`csci5410a2`)

Google Cloud Run and Website hosted on GCR

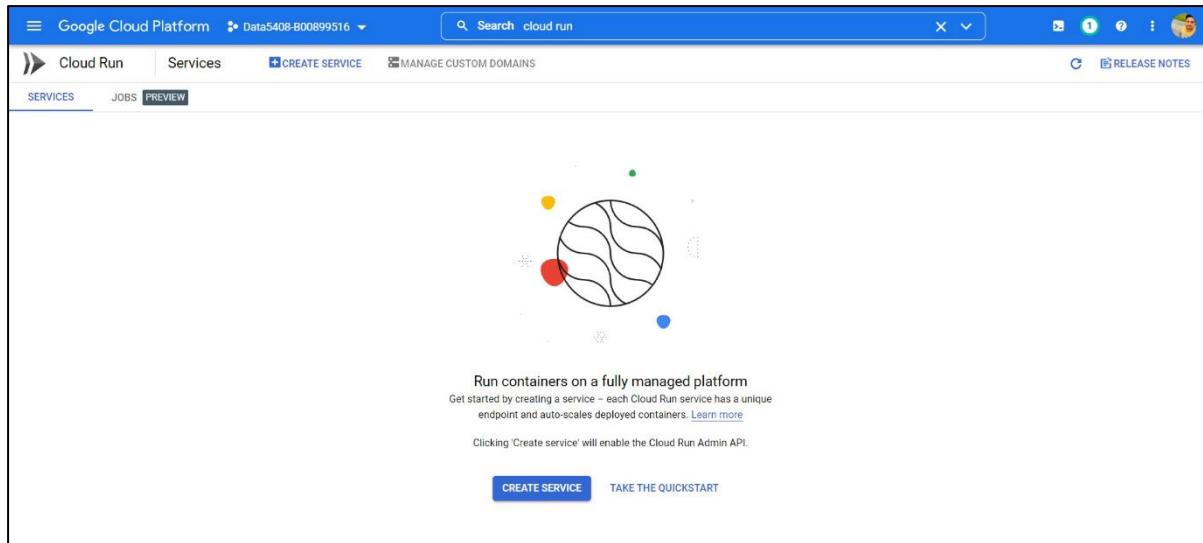


Figure 31: Google cloud run empty dashboard

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

 This screenshot shows the 'Create service' configuration page for 'Container_1'. The top section is titled 'Create service' with a back arrow. It includes fields for 'Service name' (set to 'container-1') and 'Region' (set to 'us-central1 (Iowa)'). To the right, a sidebar titled 'Cloud Run pricing' lists 'Free tier' benefits: 'First 180,000 vCPU-seconds/month', 'First 360,000 GiB-seconds/month', and '2 million requests/month'. A link 'Check paid tiers details' is also present. The main configuration area includes sections for 'CPU allocation and pricing' (with 'CPU is only allocated during request processing' selected), 'Auto-scaling' (with 'Minimum number of instances' at 0 and 'Maximum number of instances' at 100), 'Ingress' (with 'Allow all traffic' selected), and 'Authentication' (with 'Allow unauthenticated invocations' selected).

Figure 32: Container_1 Docker image Configuration on Google Cloud Run

CSCI 5410: Assignment #2 (Part A) | Meet Patel B00899516

Container, variables and secrets, connections, security ^

CONTAINER VARIABLES & SECRETS CONNECTIONS SECURITY

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
Arguments passed to the entry point command.

Capacity

Memory 512 MB CPU 1
Memory to allocate to each container instance. Number of vCPUs allocated to each container instance.

Request timeout 300 seconds
Time within which a response must be returned (maximum 3600 seconds).

Maximum requests per container 80
The maximum number of concurrent requests that can reach each container 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
 Second generation [PREVIEW](#) File system access, full Linux compatibility, faster CPU performance, slower cold starts.

CREATE CANCEL

Figure 33: Container_1 Docker image Configuration on Google Cloud Run

Google Cloud Platform Data5408-B00899516 ▾ Search Products, resources, docs (/)

Cloud Run Service details EDIT AND DEPLOY NEW REVISION SET UP CONTINUOUS DEPLOYMENT

container-1 Region: us-east1 URL: <https://container-1-anmmfpxoa-ue.a.run.app> ⚡ ⓘ

METRICS SLOS NEW LOGS REVISIONS TRIGGERS DETAILS YAML PERMISSIONS

Revisions MANAGE TRAFFIC

Filter Filter revisions

Name	Traffic	Deployed	Revision URLs (tags)	Actions
container-1-00001-zel	100% (to latest)	2 minutes ago	+	⋮

REVISIONS

container-1-00001-zel
Deployed by [mp.meetpatel98@gmail.com](#) using Cloud Console

CONTAINER VARIABLES & SECRETS CONNECTIONS SECURITY YAML

General

Image URL [us-east1-docker.pkg.dev/data5408-b00899516/csci5...](#) ⚡
Build (no build information available) ⓘ
Source (no source information available) ⓘ
Port 3000
Command and arguments (container entrypoint)

Capacity

CPU allocation CPU is only allocated during request processing
CPU 1
Memory 512MB
Concurrency 80
Request timeout 300 seconds
Execution environment First generation (default)

Figure 34: Hosted container_1 (Registration React Application) on GCR (Google Cloud Run)

CSCI 5410: Assignment #2 (Part A) | Meet Patel B00899516

The screenshot shows the Google Cloud Platform Cloud Run configuration interface for a service named 'Container_2'. The configuration includes:

- Container image URL:** us-east1-docker.pkg.dev/data5408-b00899516/csci5410a2/container
- TEST WITH A SAMPLE CONTAINER:** Should listen for HTTP requests on \$PORT and not rely on local state. [How to build a container](#)
- Cloud Run pricing:** Free tier, First 180,000 vCPU-seconds/month, First 360,000 GiB-seconds/month, 2 million requests/month. [Check paid tiers details](#)
- Service name:** container-2
- Region:** us-east1 (South Carolina)
- CPU allocation and pricing:** CPU is only allocated during request processing (selected). You are charged per request and only when the container instance processes a request.
- Auto-scaling:** Minimum number of instances: 0, Maximum number of instances: 100. Set to one to reduce cold starts. [Learn more](#)
- Ingress:** Allow all traffic (selected).
- Authentication:** Allow unauthenticated invocations (selected). Check this if you are creating a public API or website.
- Container, variables and secrets, connections, security:**
 - CONTAINER:** General settings include Container port: 3001 (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.), and Container arguments (Arguments passed to the entry point command.).
 - Capacity:** Memory: 512 MB, CPU: 1 (Memory to allocate to each container instance. Number of vCPUs allocated to each container instance.). Request timeout: 300 seconds (Time within which a response must be returned (maximum 3600 seconds)). Maximum requests per container: 80 (The maximum number of concurrent requests that can reach each container instance. [What is concurrency?](#))
 - Execution environment:** Default (selected). Cloud Run will select a suitable execution environment for you.
- Buttons:** CREATE (blue button), CANCEL

Figure 35: Container_2 Docker image Configuration on Google Cloud Run

CSCI 5410: Assignment #2 (Part A) | Meet Patel B00899516

The screenshot shows the Google Cloud Platform Cloud Run service details for 'container_2'. The service is deployed to the 'us-east1' region with the URL <https://container-2-anmifspxo-ue.a.run.app>. The 'REVISIONS' tab is selected, showing one revision named 'container-2-00001-voh' which was deployed just now. The 'CONTAINER' tab displays configuration details such as Image URL (us-east1-docker.pkg.dev/data5408-b00899516/csci5...), Build (no build information available), Source (no source information available), Port (3001), Command and arguments (container entrypoint), Capacity (CPU allocation: CPU is only allocated during request processing, CPU: 1, Memory: 512MB, Concurrency: 80), and Variables & Secrets.

Figure 36: Hosted container_2 (Login React Application) on GCR (Google Cloud Run)

The screenshot shows the 'Create service' page for 'Container_3'. It starts by prompting to 'Deploy one revision from an existing container image' or 'Continuously deploy new revisions from a source repository'. The 'Container image URL' field is set to 'us-east1-docker.pkg.dev/data5408-b00899516/csci5410a2/container'. The 'TEST WITH A SAMPLE CONTAINER' section indicates the service should listen on \$PORT. The 'Cloud Run pricing' sidebar shows the 'Free tier' includes First 180,000 vCPU-seconds/month, First 360,000 GiB-seconds/month, and 2 million requests/month. Below this, there's a link to 'Check paid tiers details'. The main configuration area includes fields for 'Service name' (set to 'container-3'), 'Region' (set to 'us-east1 (South Carolina)'), 'CPU allocation and pricing' (set to 'CPU is only allocated during request processing'), 'Auto-scaling' (set to 'Minimum number of instances: 0, Maximum number of instances: 100'), 'Ingress' (set to 'Allow all traffic'), and 'Authentication' (set to 'Allow unauthenticated invocations').

Figure 37 : Container_3 Docker image Configuration on Google Cloud Run

CSCI 5410: Assignment #2 (Part A) | Meet Patel B00899516

The screenshot shows the 'Container, variables and secrets, connections, security' configuration page for a Docker image. It includes sections for General (Container port: 3002), Capacity (Memory: 512 MB, CPU: 1), Request timeout (300 seconds), Maximum requests per container (80), and Execution environment (Default selected). Buttons for 'CREATE' and 'CANCEL' are at the bottom.

Figure 38: Container_3 Docker image Configuration on Google Cloud Run

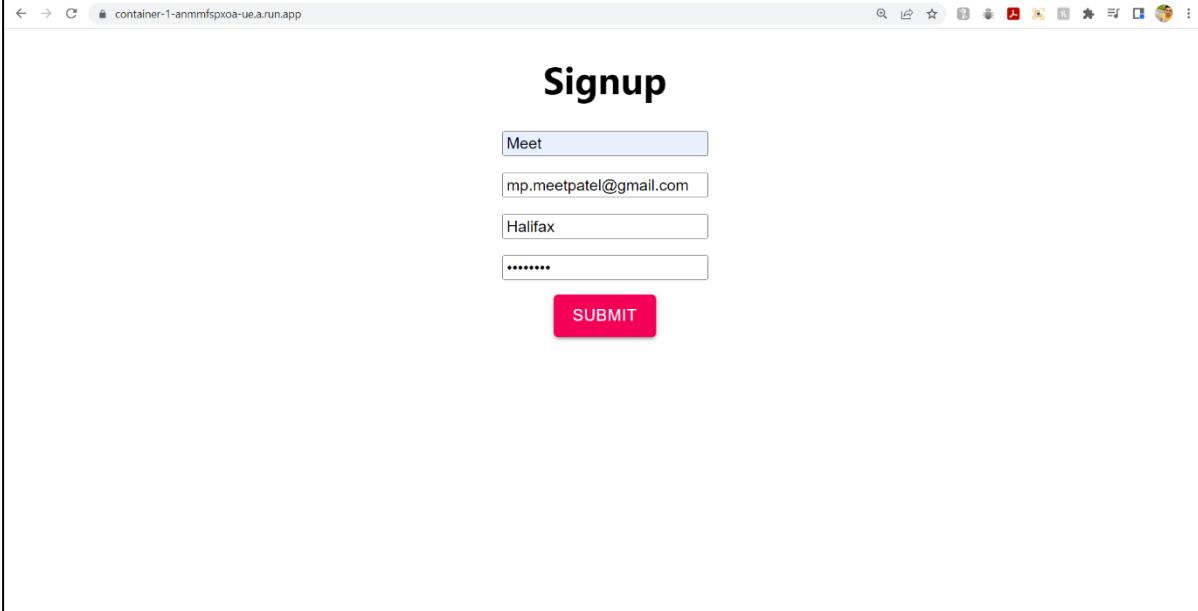
The screenshot shows the 'Service details' page for 'container-3'. It displays the URL (https://container-3-anmmfspxoae.a.run.app) and the revision history. The current revision is 'container-3-00001-def' (100% traffic, Just now). The right panel shows the detailed configuration for this revision, including General settings (Image URL: us-east1-docker.pkg.dev/data5408-b00899516/csci5..., Build: no build information available, Source: no source information available, Port: 3002, Command and arguments: (container endpoint)), Capacity settings (CPU allocation: CPU is only allocated during request processing, CPU: 1, Memory: 512MB, Concurrency: 80, Request timeout: 300 seconds), and Execution environment (First generation (default)).

Figure 39: Hosted container_3 (UserProfile React Application) on GCR (Google Cloud Run)

Website hosted on Google Cloud Run

1. Registration React Application

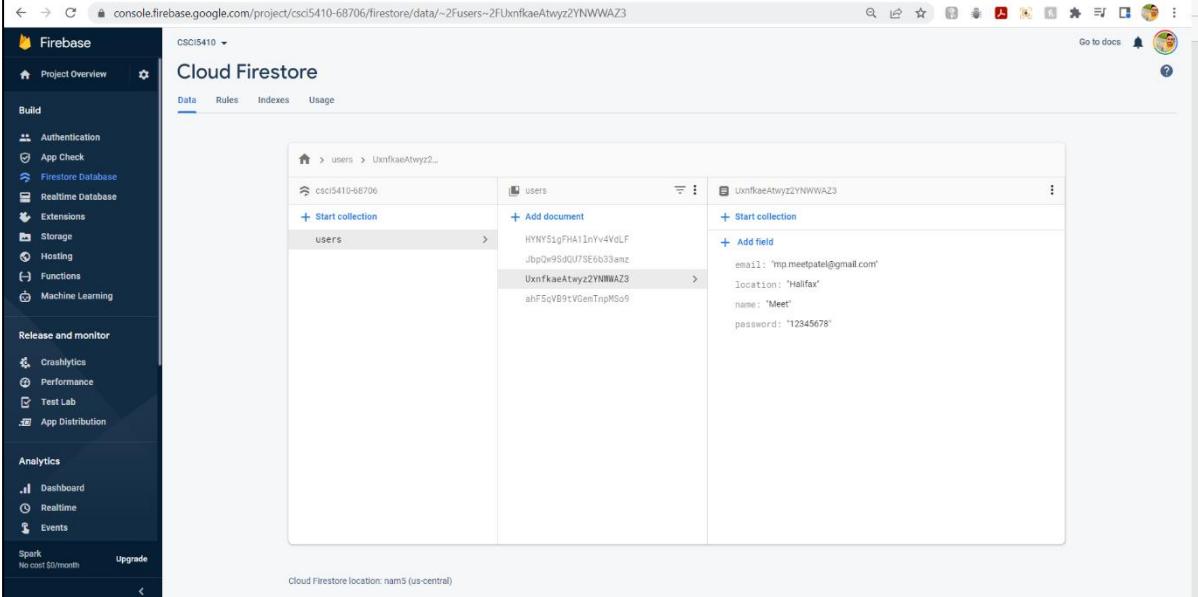
Figure 40 shows the **Registration Page (container_1)** hosted on Google Cloud Run



The screenshot shows a web browser window with the URL `container-1-anmmfspoxa-ue.a.run.app`. The page title is "Signup". It contains four input fields: "Meet" (name), "mp.meetpatel@gmail.com" (email), "Halifax" (location), and "*****" (password). Below the fields is a red "SUBMIT" button.

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

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



The screenshot shows the Firebase Cloud Firestore console. The left sidebar shows the project structure with "Authentication", "Firestore Database", "Functions", and "Machine Learning" selected. The main area shows the "Cloud Firestore" interface with a "Data" tab selected. Under the "users" collection, there is a document with the ID "UxnkaeAtwyz2YNWWAZ3". The document contains the following data:

Field	Type	Value
email	String	"mp.meetpatel@gmail.com"
location	String	"Halifax"
name	String	"Meet"
password	String	"12345678"

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

The screenshot shows the Firebase Cloud Firestore interface. On the left, the navigation sidebar includes sections for Authentication, App Check, Firestore Database, Realtime Database, Extensions, Storage, Hosting, Functions, and Machine Learning. Under the 'Release and monitor' section, Crashlytics, Performance, Test Lab, and App Distribution are listed. The main area displays the 'Cloud Firestore' interface with a 'Data' tab selected. A tree view shows a 'users' collection under 'csci5410-68706'. A specific document named 'ahF5qVB9tVGemTnpMSo9' is selected, showing fields: email ('mp.meetpatel@gmail.com'), status ('offline'), and timestamp ('12 June 2022 at 05:57:43 UTC-7').

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

2. Login React Application

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

The screenshot shows a web browser displaying a login form titled 'Login'. The form has two input fields: one for email ('mp.meetpatel@gmail.com') and one for password ('*****'). Below the fields is a blue 'SUBMIT' button.

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

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

The screenshot shows the Firebase Cloud Firestore interface. On the left, there's a sidebar with various services like Authentication, Firestore Database, Realtime Database, and Functions. The main area is titled 'Cloud Firestore' and shows a collection named 'users'. Inside the 'users' collection, there are several documents, one of which is selected and shown in detail. The document ID is 'ahF5qVB9tVGemTnpMSo9'. It contains fields: 'email' (set to 'mp.meetpatel@gmail.com'), 'status' (set to 'online'), and 'timestamp' (set to '12 June 2022 at 05:57:43 UTC-7').

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

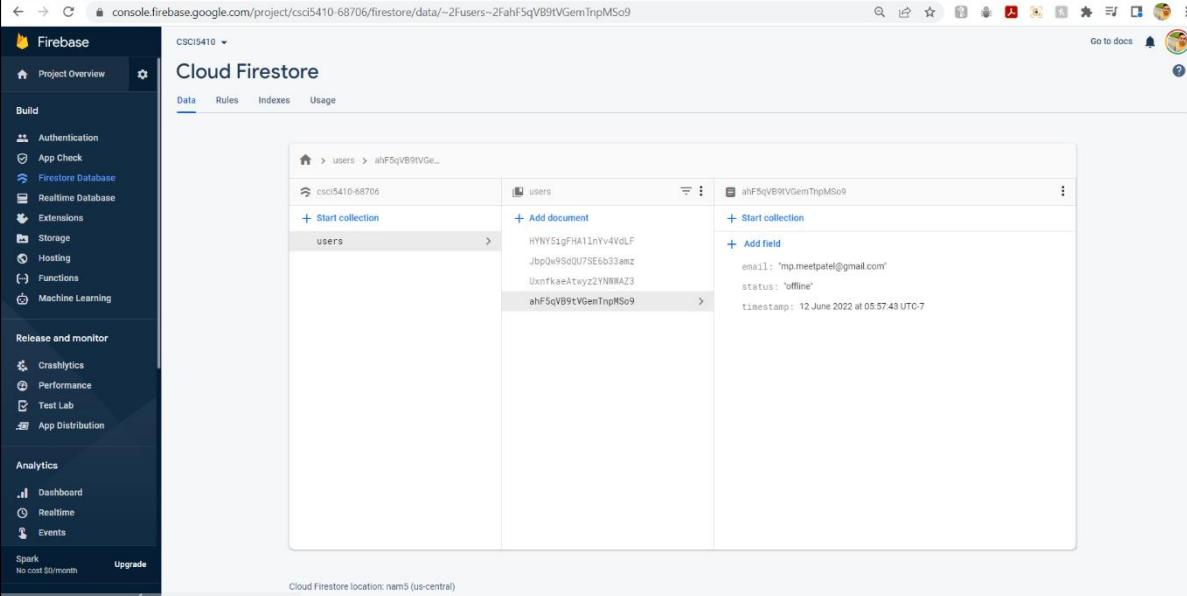
3. UserProfile React Application

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

The screenshot shows a web page titled 'Hi' with the user's email address, 'mp.meetpatel@gmail.com', displayed prominently. Below the email, it says 'you are logged in.....'. Underneath that, it lists 'Here are the other users who are online' with the email '<mt631537@dal.ca> <mp.meetpatel@gmail.com>'. At the bottom of the page is a blue 'LOGOUT' button.

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

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



The screenshot shows the Firebase Cloud Firestore interface. On the left, the navigation sidebar includes sections for Authentication, App Check, Firestore Database, Realtime Database, Extensions, Storage, Hosting, Functions, Machine Learning, Release and monitor, Crashlytics, Performance, Test Lab, and App Distribution. Below that are Analytics (Dashboard, Realtime, Events), and a Spark plan section indicating 'No cost \$0/month' with an 'Upgrades' button. The main area is titled 'Cloud Firestore' and shows a collection named 'users'. Under 'users', there is a document with the ID 'ahF5qVB9tVGemTnpMSo9'. This document contains fields: email ('mp.meetpatel@gmail.com'), status ('offline'), and timestamp ('12 June 2022 at 05:57:43 UTC-7'). The URL in the browser bar is <https://console.firebaseio.google.com/project/csci5410-68706/firestore/data/~2fusers~2fahF5qVB9tVGemTnpMSo9>.

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

Test Cases and Important Code

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

```
function validate(data) {  
  let errors = {};  
  
  if (!data.name.trim()) {  
    errors.name = "Name required";  
  }  
  
  if (!data.location.trim()) {  
    errors.location = "Location required";  
  }  
  
  if (!data.email) {  
    errors.email = "Email required";  
    // Regex Source link: https://regexr.com/3e48o  
  } else if (!/^\w+@\w+\.\w{2,4}$/.test(data.email)) {  
    errors.email = "Email address is invalid";  
  }  
  
  if (!data.password) {  
    errors.password = "Password required";  
  } else if (data.password.length < 8) {  
    errors.password = "Password needs to be 8 characters or more";  
  }  
  return errors;  
}
```

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

```
function validate (data) {  
  let errors = {};  
  
  if (!data.email) {  
    errors.email = 'Email required';  
  }
```

```
    }

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

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

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

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

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

```
);

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

5. Methods to fetch all the users who are online – [User Profile Page \(UserProfile.js\)](#)

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

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

Screenshot of the Validation Performed on Website

1. Signup page validations

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

The screenshot shows a web browser window with a "Signup" form. The form has four input fields: "Enter your name", "Enter your email", "Enter your location", and "Enter password". Each field has a red validation message below it: "Name required", "Email required", "Location required", and "Password required". A pink "SUBMIT" button is at the bottom.

Figure 47: All the fields are mandatory

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

The screenshot shows a web browser window with a "Signup" form. The "Enter your email" field contains "mt631537" and has a red validation message "Email address is invalid". The other three fields ("Enter your name", "Enter your location", and "Enter password") are empty. A pink "SUBMIT" button is at the bottom.

Figure 48: Email address must be valid

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

The screenshot shows a web browser window with the URL `container-1-anmmfspxo-ue.a.run.app`. The page title is "Signup". There are four input fields: "Meet" (text), "mt631537@dal.ca" (email), "Halifax" (text), and "....." (password). Below the password field is a red error message: "Password needs to be 8 characters or more". A red "SUBMIT" button is at the bottom.

Figure 49: Length of password must 8 character or more

2. Login page validation

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

The screenshot shows a web browser window with the URL `container-2-anmmfspxo-ue.a.run.app`. The page title is "Login". There are two input fields: "Enter your email" (text) with a red error message "Email required" below it, and "Enter your password" (text) with a red error message "Password required" below it. A blue "SUBMIT" button is at the bottom.

Figure 50: Email address and password are mandatory

Summary

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

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

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

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

Google Cloud Run

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

Artifact Registry

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

Docker Containers

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

References

- [1] Google, "Cloud Firestore," Google, [Online]. Available: <https://firebase.google.com/products/firestore>. [Accessed 11 June 2022].
- [2] Docker, "Developers Love Docker," Docker, [Online]. Available: <https://www.docker.com/>. [Accessed 11 June 2022].
- [3] Google, "Installing Cloud SDK," Google, [Online]. Available: <https://cloud.google.com/sdk/docs/install>. [Accessed 11 June 2022].
- [4] Google, "Container Registry," Google, [Online]. Available: <https://cloud.google.com/container-registry>. [Accessed 11 June 2022].
- [5] Google, "Cloud Run," Google, [Online]. Available: <https://cloud.google.com/run>. [Accessed 11 June October 2022]