CSCI 5410 Serverless Data Processing

Assignment 2 Report

Prepared by:
Vikram Venkatapathi - B00936916

Master of Applied Computer Science (Summer'23)
Faculty of Computer Science
Dalhousie University

GitLab Repo link : **https://git.cs.dal.ca/vikramv/csci5410-summer-23-b00936916/-/tree/A2?ref_type=heads**

Contents

Part I.

PART A

Provide summary

1. Introduction

The given paper titled as ***Performance Evaluation of Distributed Systems in Multiple Clouds using Docker Swarm*** authored by "**N. Naik**", focuses on Docker Swarm-based Distributed System in Multiple Clouds[1].

This report attempts to review the given literature and then provides a summary.

2. Summary

The authors present a study on designing distributed systems in multiple clouds using Docker Swarm. They emphasize the benefits of multi-cloud infrastructure and discuss the challenges it entails. **Docker** is introduced as an **efficient platform for application development** through **containerization**. **Docker Swarm** [15] is highlighted as a **clustering tool** that addresses critical issues in provisioning, configuration management, load balancing, and migration. Overall, the paper provides insights into building robust distributed systems across multiple cloud environments using Docker Swarm.

The paper specifically addresses the challenge of designing distributed systems across multiple cloud environments. It highlights the **issues** related to **provisioning, configuration management, load balancing, and** migration in such setups. The authors propose Docker Swarm as a solution to these challenges, as it provides a clustering mechanism that enables efficient application development through containerization. By leveraging Docker Swarm, the paper presents a framework for building robust distributed systems that can seamlessly operate in multi-cloud infrastructures.

The paper describes several experiments and studies conducted to validate the proposed framework. The authors set up a test environment consisting of multiple cloud providers**(AWS, Azure, GCP, Digital Ocean & Softlayer)** and deployed a distributed application using Docker Swarm. They **evaluated** the **performance and scalability** of the system by **measuring response times, throughput**, and **resource utilization** under different workload conditions. Additionally, they conducted experiments to analyze the impact of network latency and node failures on the overall system performance. The **results** of these experiments demonstrated the **effectiveness** of the proposed framework in achieving efficient resource utilization, **fault tolerance**, and **seamless scalability** in multi-cloud environments.

The authors' analysis and findings demonstrated that their **framework improved resource utilization** in multi-cloud environments by distributing tasks effectively. The experiments showed scalability and fault tolerance, with the system adapting to workload changes and recovering from failures. **Network latency** was identified as an **influential factor**, underscoring the importance of efficient communication and task allocation. Overall, the framework proved advantageous for efficient resource management and fault tolerance in multi-cloud settings.

In conclusion, the authors presented a framework for efficient resource management in multi-cloud environments. Their approach effectively distributed tasks, improved resource utilization, and demonstrated scalability and fault tolerance. The experiments highlighted the significance of network latency and emphasized the need for efficient communication and task allocation. The findings underscored the framework's ability to adapt to workload changes and recover from failures. Overall, the study showcased the advantages of the proposed framework in enhancing resource management and fault tolerance in multi-cloud settings.

My suggestions on areas of improvement:

The paper could **discuss the limitations of its experimental setup**. For example, the authors only tested the system with a small number of nodes. But according to [16], Kubernetes is a good choice, when the scale, complexity, and management of the containers increase. Hence, It would be interesting to see how the system performs with a larger number of nodes.
The paper could **suggest potential areas for future exploration**. For example, the authors could explore how Docker Swarm could be used to build distributed systems that are more secure or more efficient.
The paper could **provide alternative perspectives on the findings**. For example, the authors could compare the performance of Docker Swarm to other container orchestration tools, such as Kubernetes[17], and K3s[18].

Part II.

PART B

Build, deploy, and run a Containerized Application using GCP

3. Procedure followed for the given experiment

3.1. Flowchart

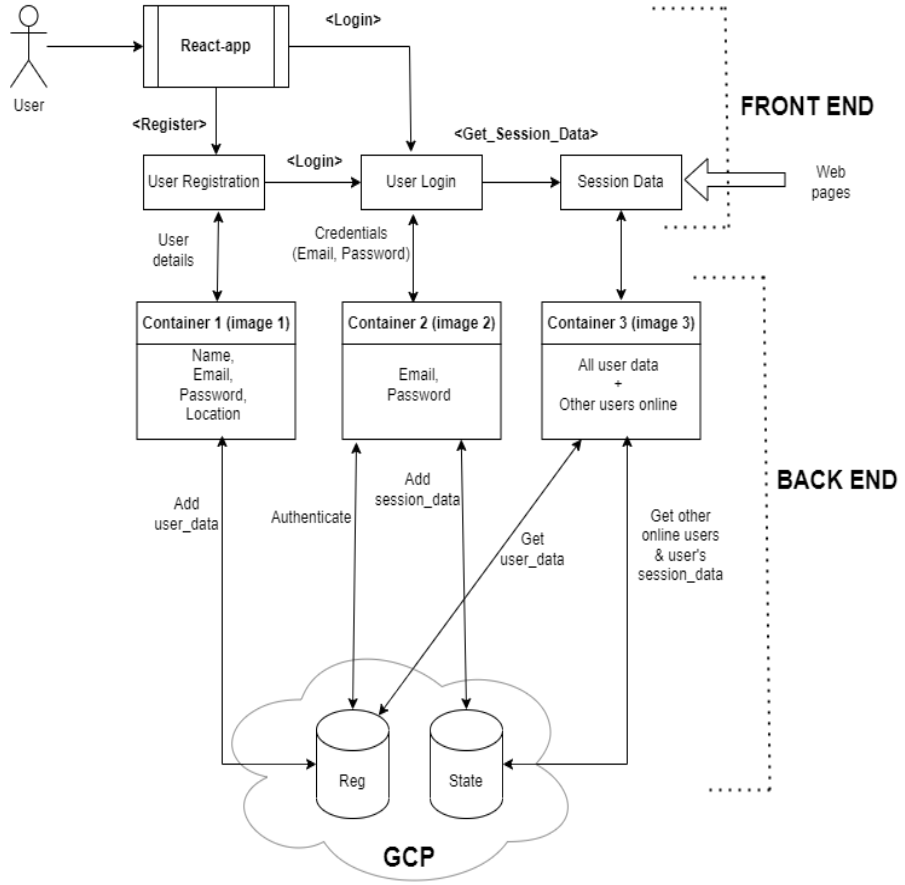

Figure 3.1.: ***Flowchart describing the operations that I have performed.***

3.2. Steps

3.2.1. Configure environment

1. Create images for all the given code and front-end (i have created the front-end in React)
   Note: Download the service account key from GCP IAM. In the docker file, copy the file to the container, and set it as an Environment variable as follows

```
COPY assignment-2-390620-60700558380c.json /app/assignment
    -2-390620-60700558380c.json

ENV GOOGLE_APPLICATION_CREDENTIALS /app/assignment-2-390620-60700558380c.
    json
```

Listing 3.1: Dockerfile Code snippet

   This will allow the containers to access the collection in Firestore.

2. Create separate repositories for all containers (3 containers: back-end; 1 container: front-end) in **Artifact Registry**.

3. Push the images to the respective repositories

4. Using the images, create services for all the containers, in **Cloud run** (with proper **PORT** and other configuration)

3.2.2. Control and data flow of the deployments

1. User performs registration

2. Send a POST request to Container-1 (API: /A2/register), with all *user_data*.

3. Registration by **Container-1**:

   a) Create record in collection **"Reg"** with *user_data* (shown below)

```
{
    'Name': <name>,
    'Password': <password>,
    'Email': <email>,
    'Location': <location>
}
```

   b) Return response (success/failure).

4. Front-end: re-direct to **login page** after successful registration (API: /A2/login).

5. User enters credentials.

6. Authentication by **Container-2**

   a) Fetch the respective document from the collection "Reg"

b) Validate credentials

c) Return Success/Failure

7. After successful authentication,

   a) If the user is logging in for the 1st time, create a *session_data* document(shown below) in the collection "state", with the user_email as the document id.

```
{
    'Online': True,
    'Offline': False,
    'Timestamp': <current_system_time>
}
```

   b) Else, just update the fields (toggle Online & Offline, update the current timestamp of system)

8. Redirect to **session_data page** (API: /A2/sessionData/<email>)

9. Session data from **Container-3**

   a) Fetch *user_data* from the collection "Reg", *session_data* from the collection "state", by matching with the <email>

   b) Display all the user details

   c) To display the other users, who are online, filter out all documents in the collection "state" where the key **"Online" is True**. (API: /A2/sessionData/usersOnline/<email >)

   d) Display the filtered document ids (except the current user who has logged in).

10. Logout by Container-3:

    a) On clicking logout, **end the current session**. (API: /A2/sessionData/<email >/logout)

    b) Update the *session_data* in the collection "state" (toggle Online & Offline fields)

    c) Re-direct to the Login page.

4. Screenshots

4.1. Environment setup

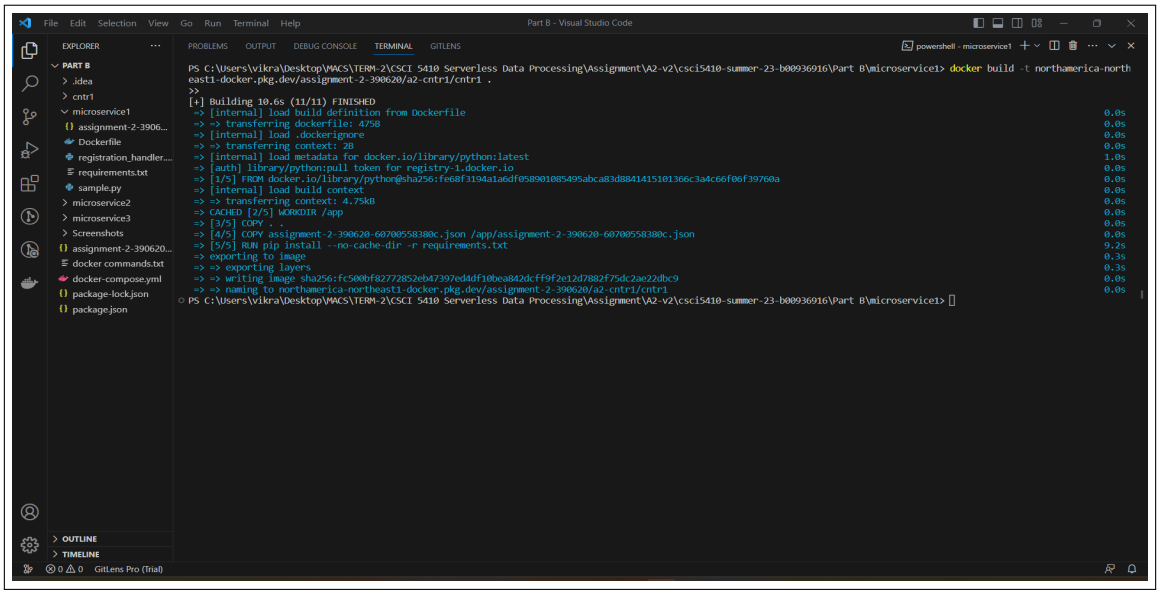

Figure 4.1.: ***Build container 1***

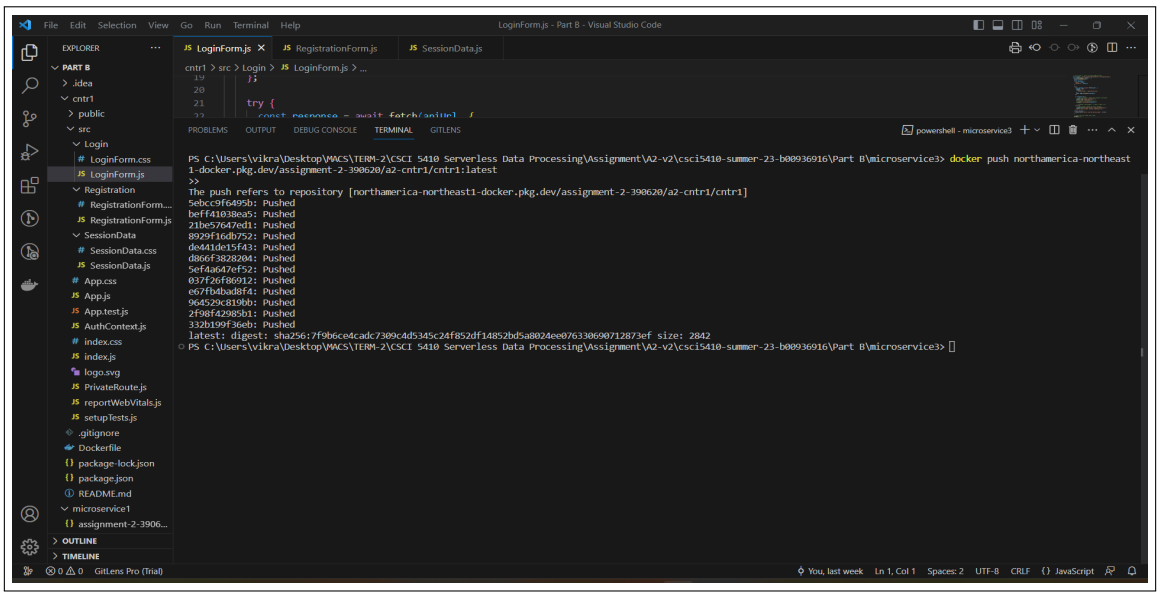

Figure 4.2.: ***Push container 1***

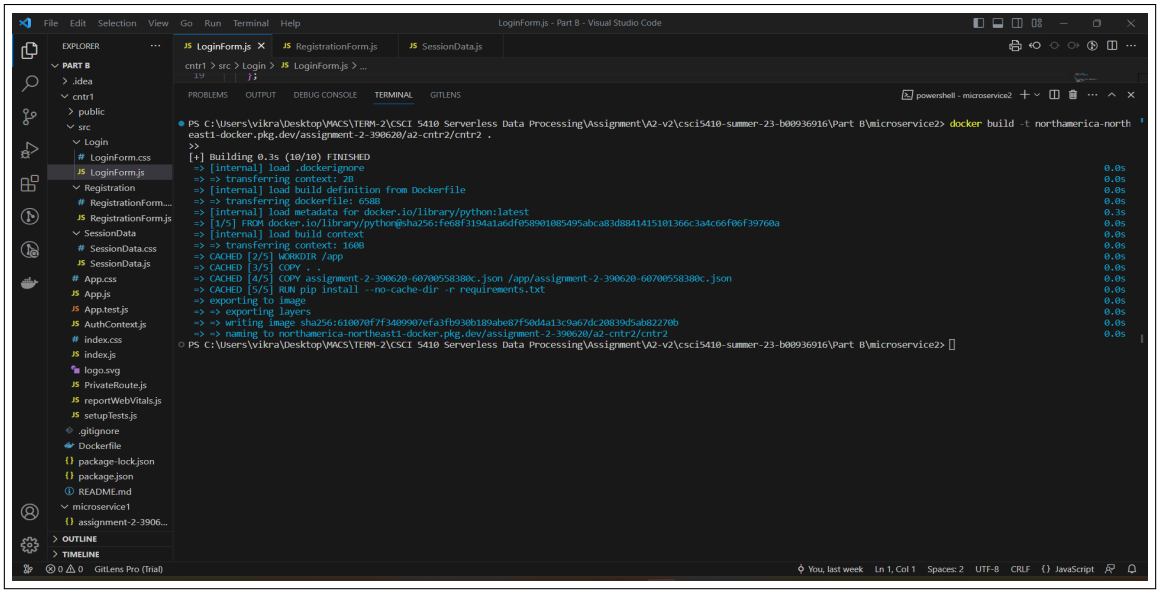

Figure 4.3.: ***Build container 2***

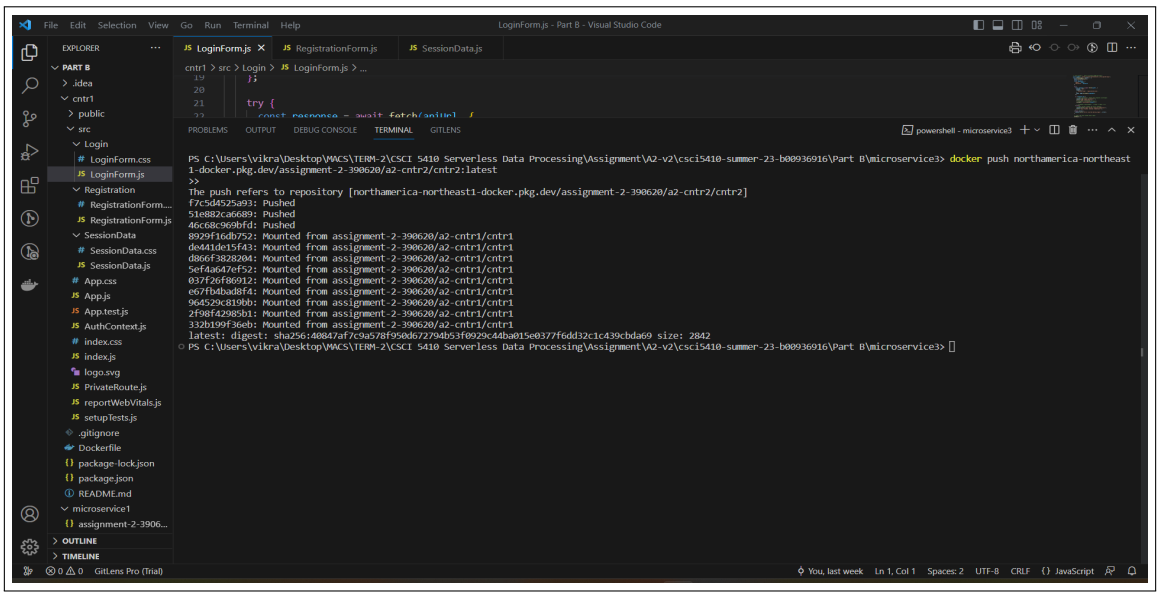

Figure 4.4.: ***Push container 2***

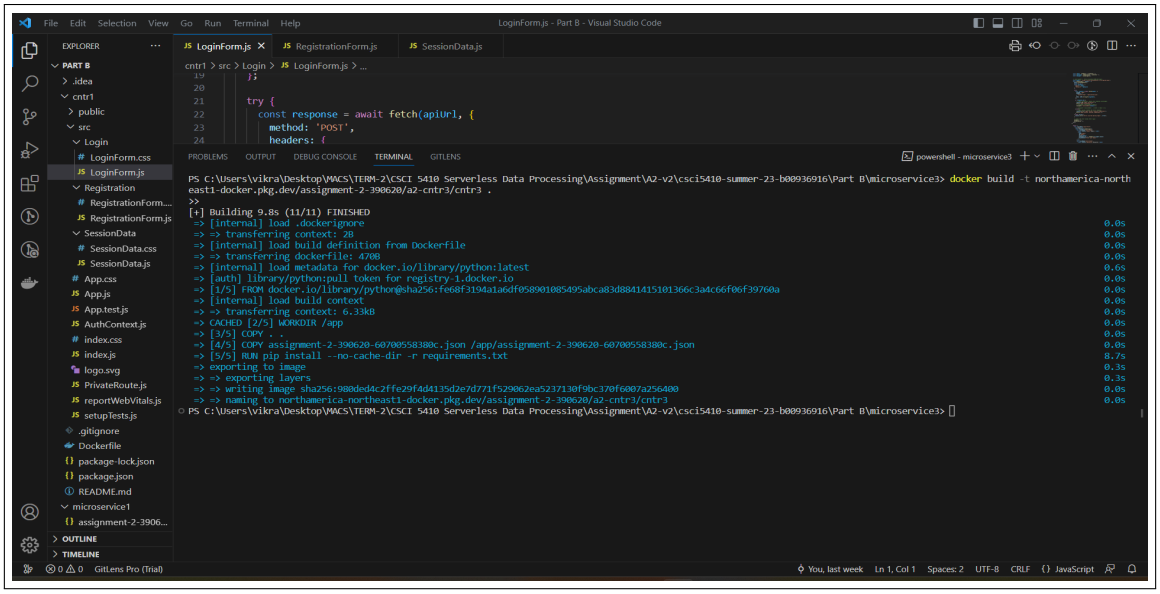

Figure 4.5.: ***Build container 3***

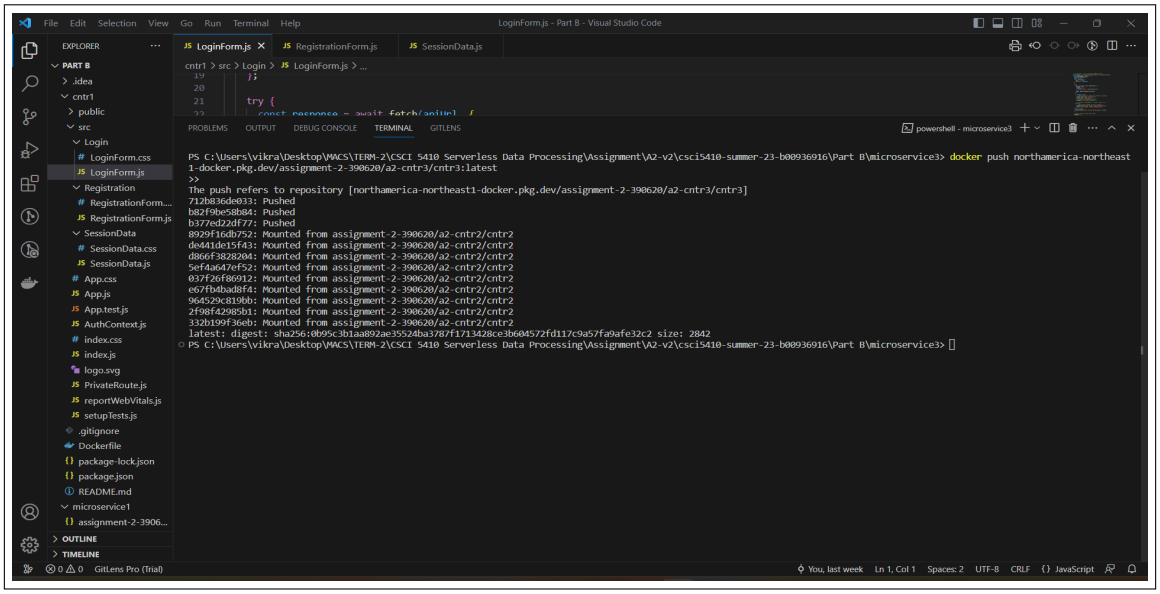

Figure 4.6.: ***Push container 3***

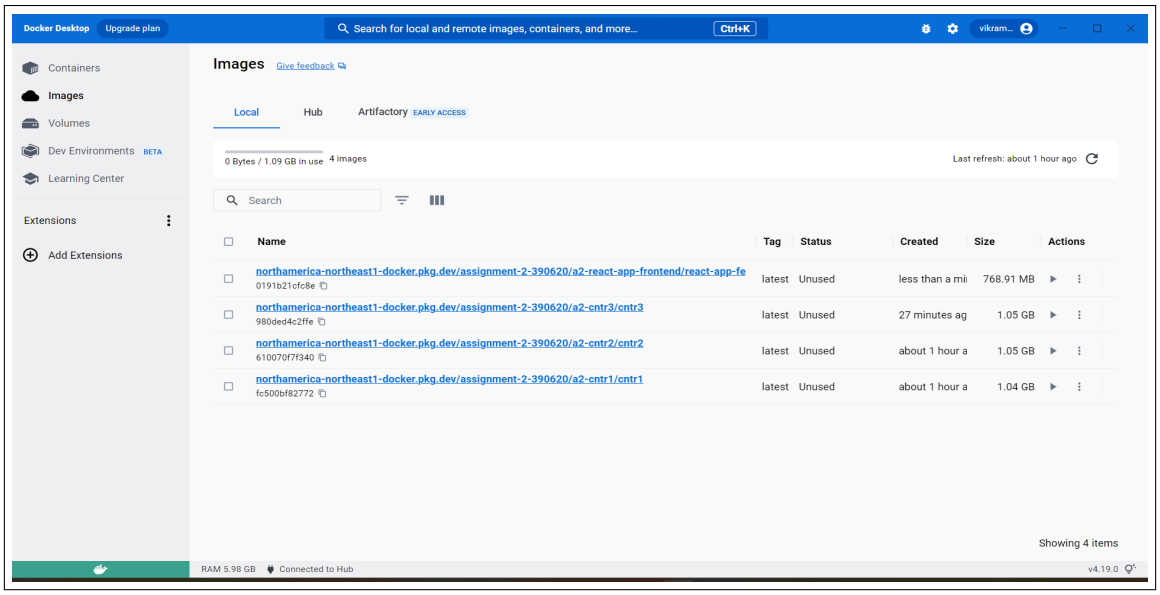

Figure 4.7.: ***Built images listed in Docker Desktop***

4.1.1. Collections & Repositories

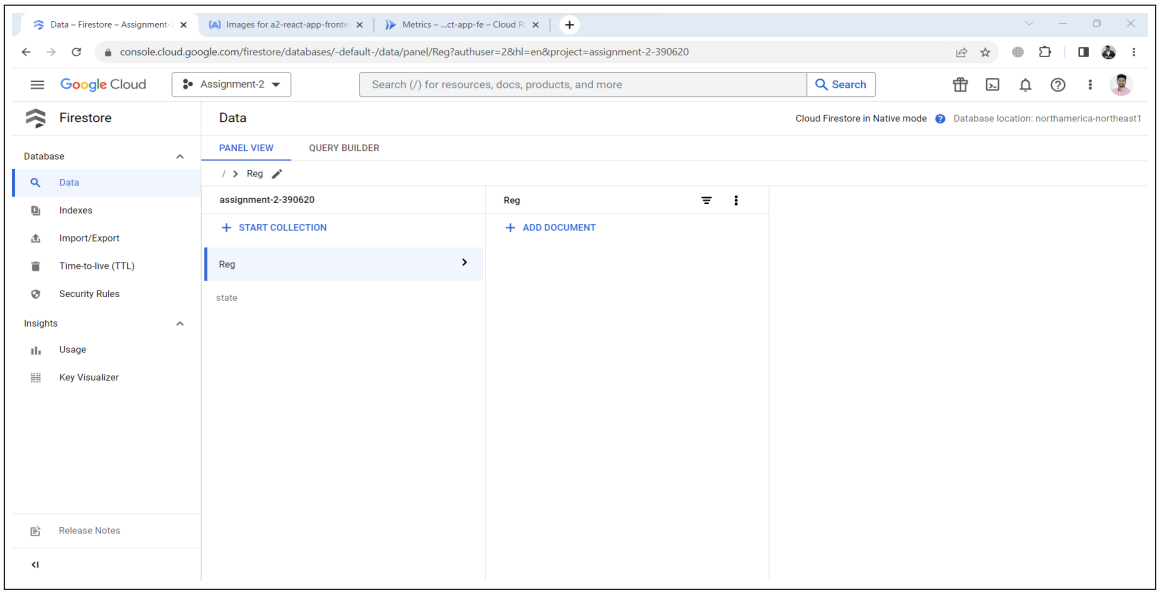

Figure 4.8.: ***Empty collection "Reg"***

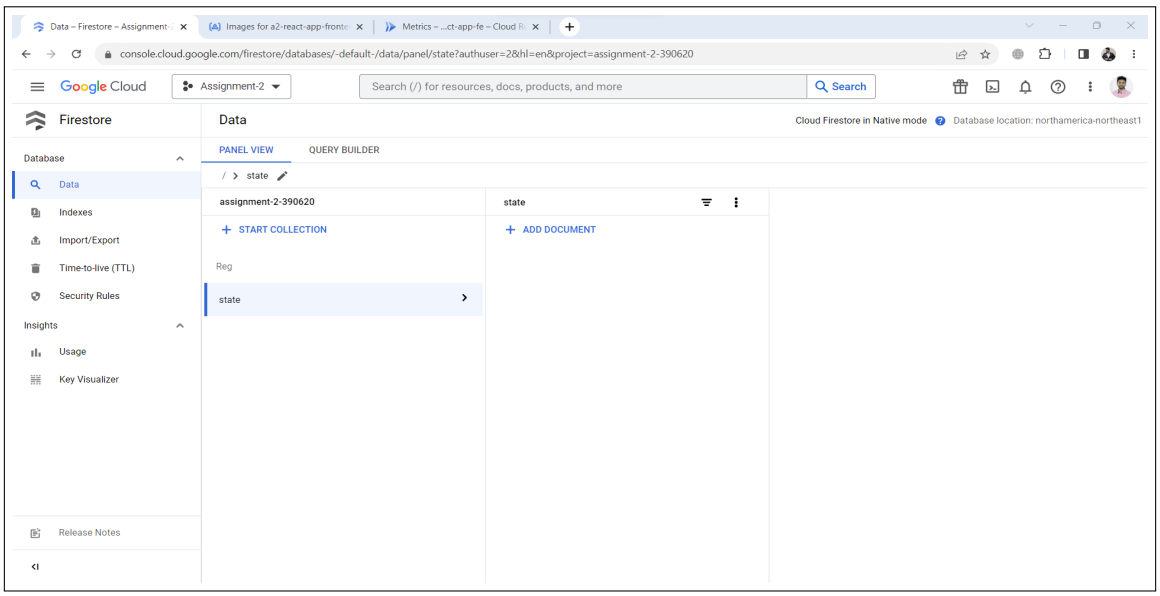

Figure 4.9.: ***Empty collection "state"***

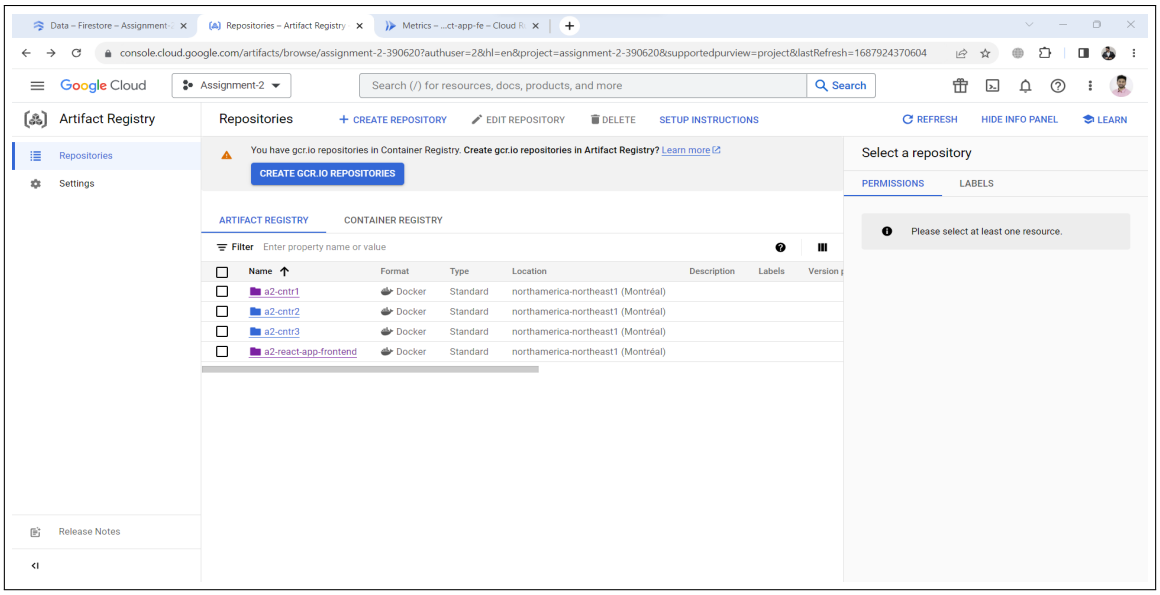

Figure 4.10.: ***Artifact registry repositories***

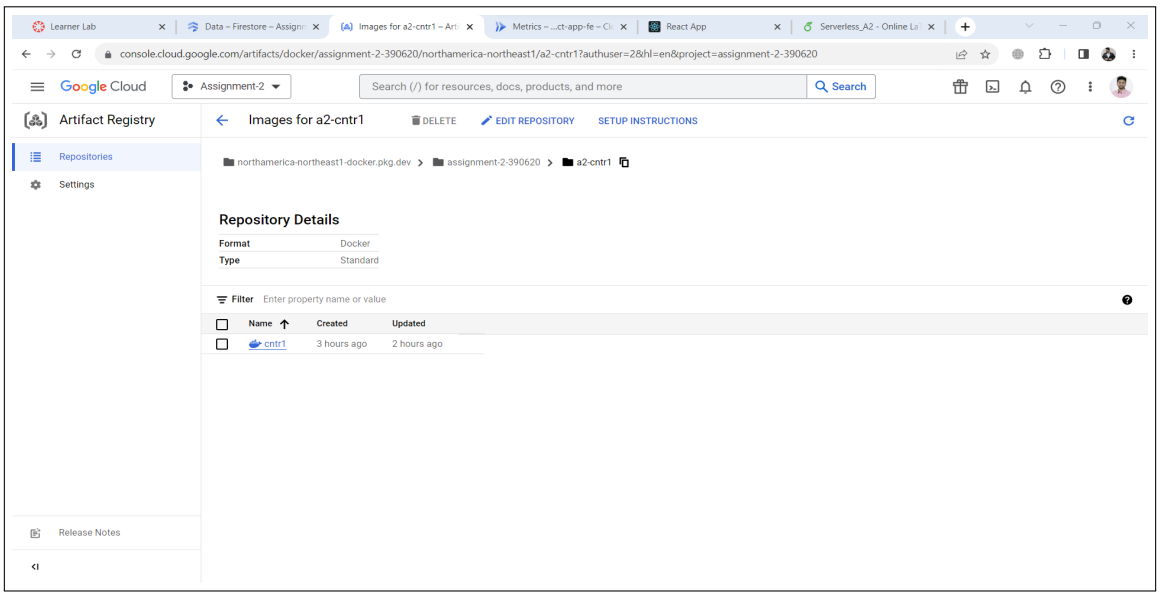

Figure 4.11.: ***Artifact registry repository - container 1***

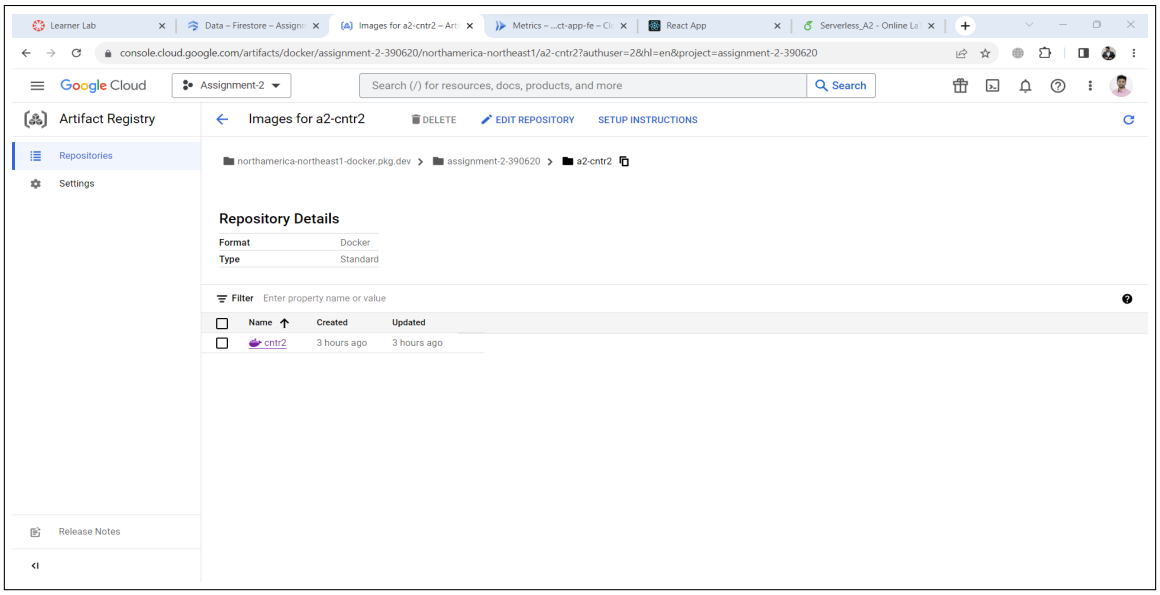

Figure 4.12.: ***Artifact registry repository - container 2***

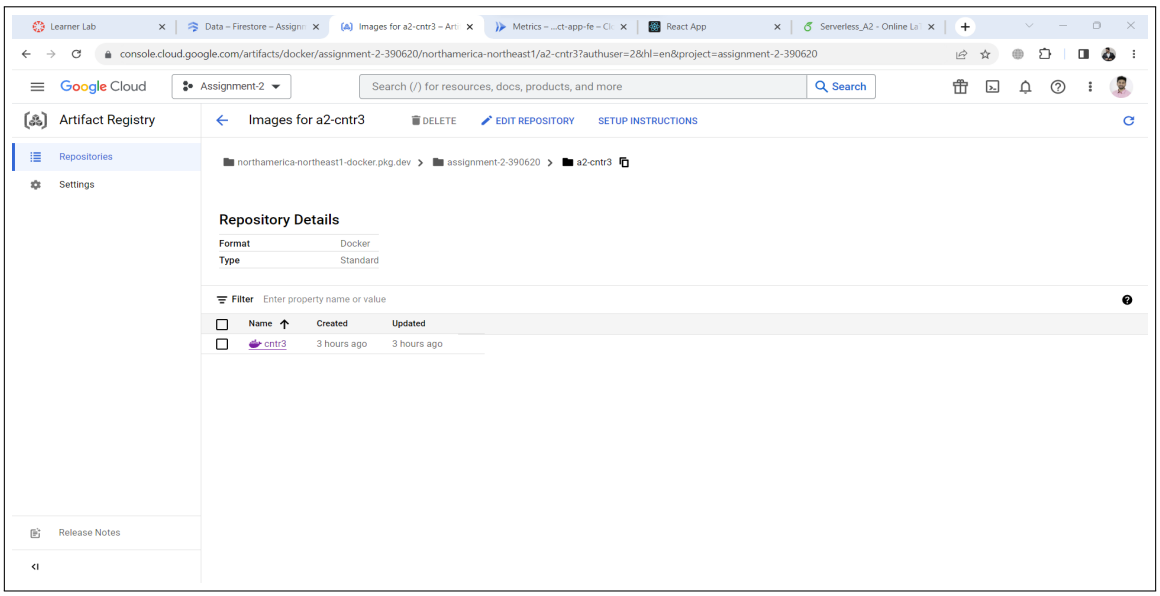

Figure 4.13.: ***Artifact registry repository - container 3***

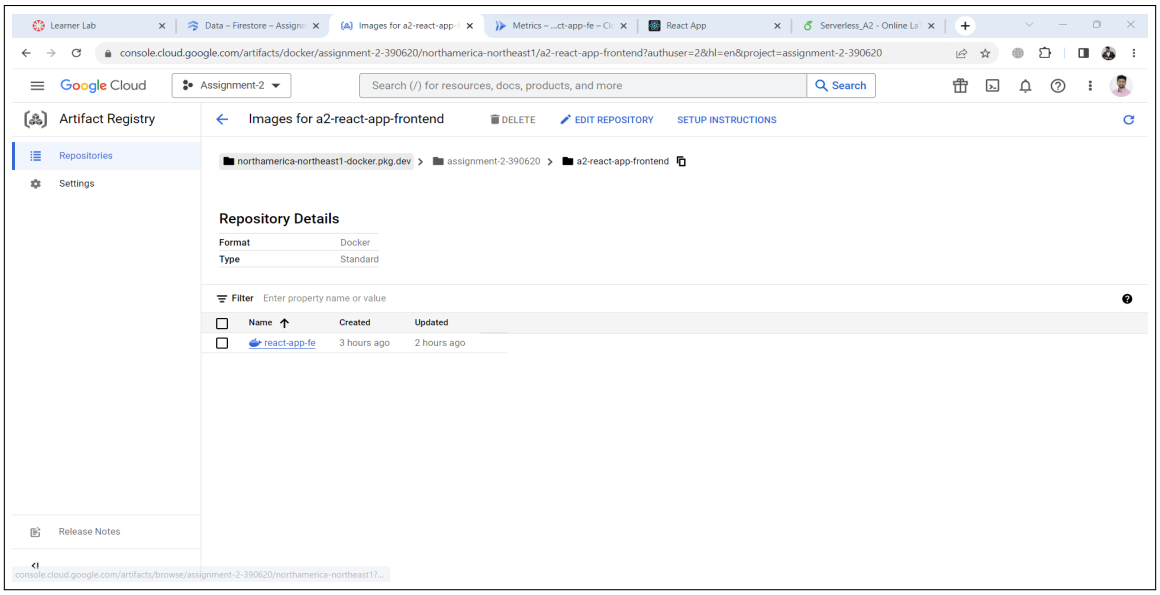

Figure 4.14.: ***Artifact registry repository - react-app frontend container***

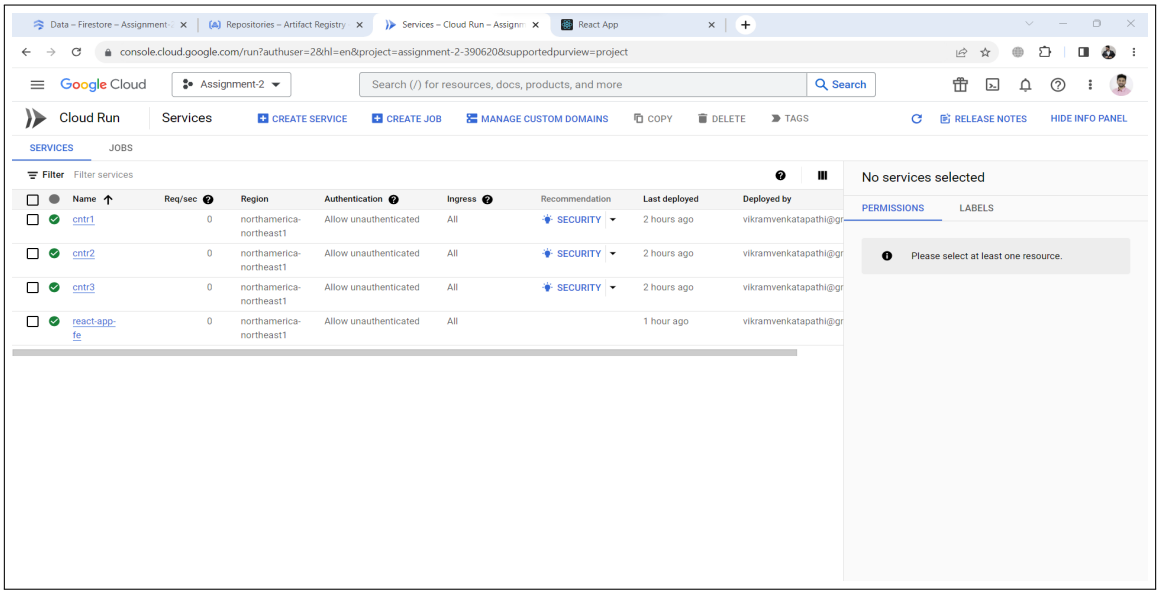

Figure 4.15.: ***Services created in Cloud Run***

4.2. Demo

NOTE: In some of the screenshots, the URLs displayed on the front end may vary due to the process of deleting and recreating services on Cloud Run. These changes were made during the documentation process, which resulted in different URLs being assigned to the deployed containers. Please note that despite the variations in the URLs shown in the screenshots, the overall functionality and deployment process remains consistent.

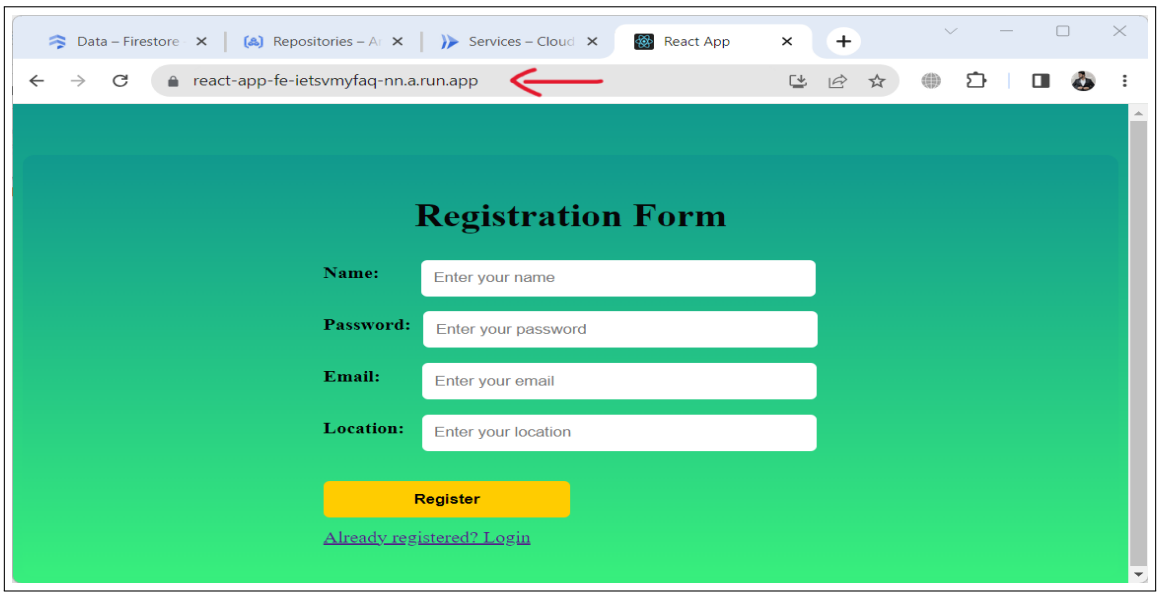

Figure 4.16.: ***Registration page***

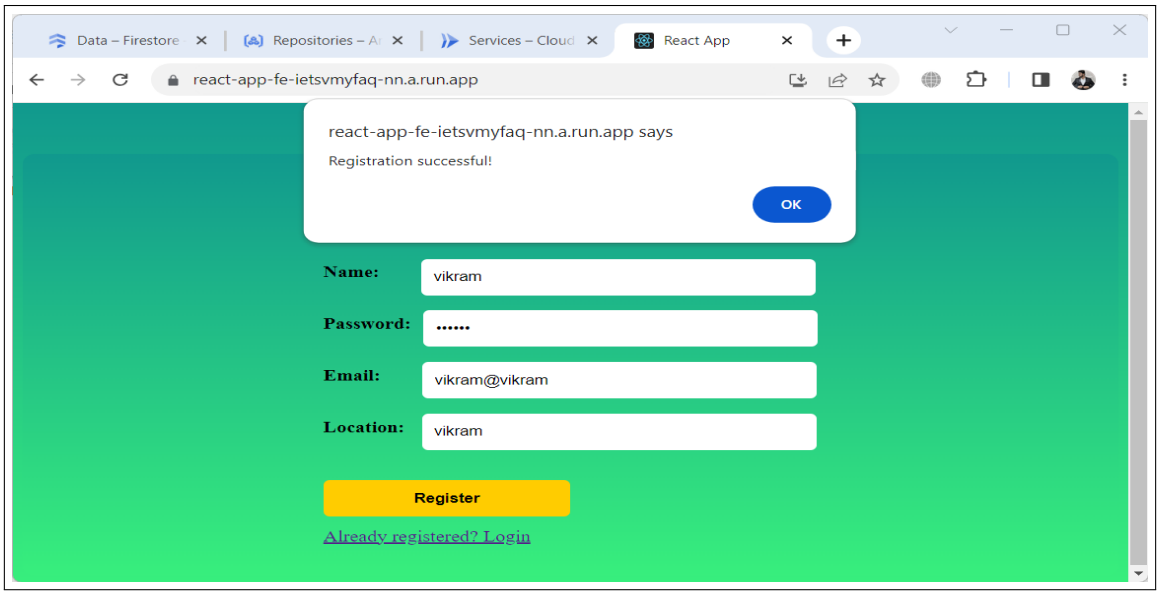

Figure 4.17.: ***User - Vikram: Registration success***

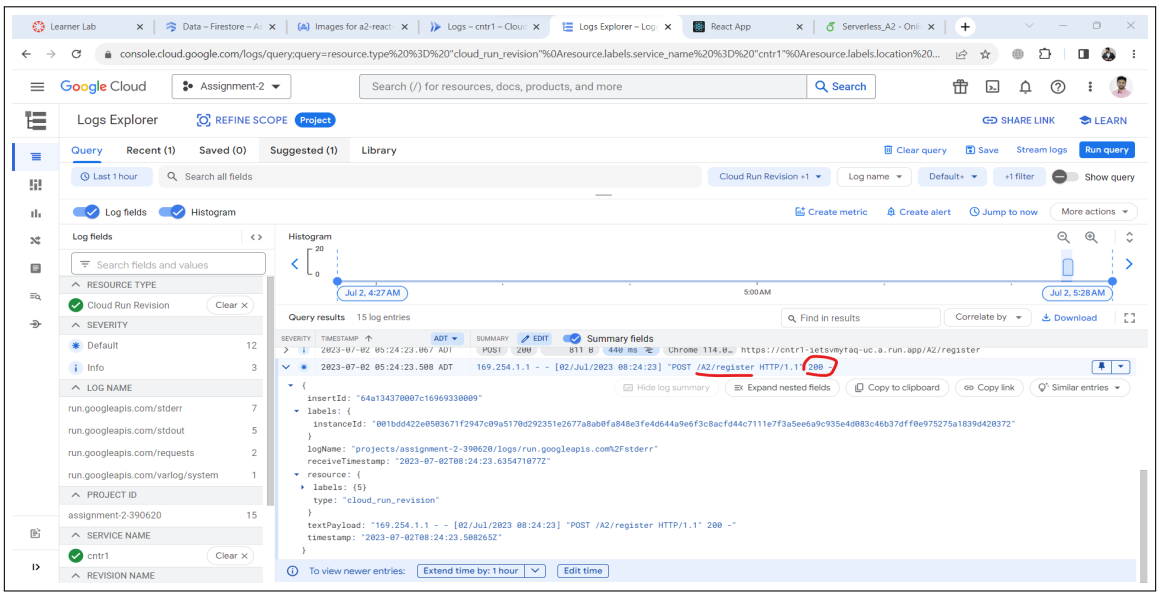

Figure 4.18.: ***User - Vikram: Registration Logs***

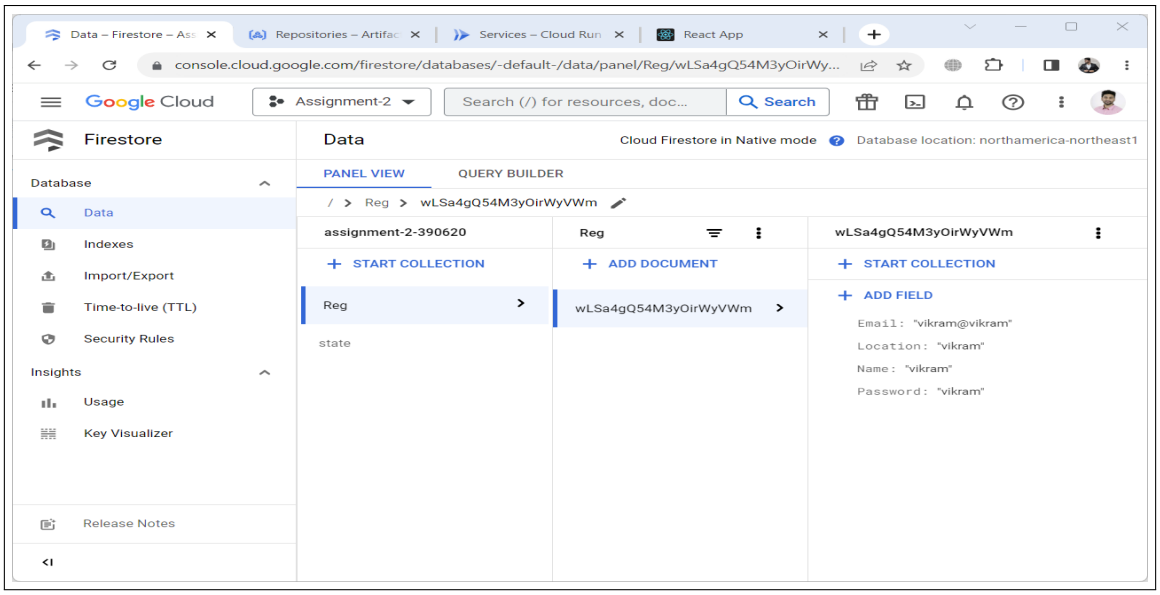

Figure 4.19.: ***User - Vikram: Document created in "Reg"***

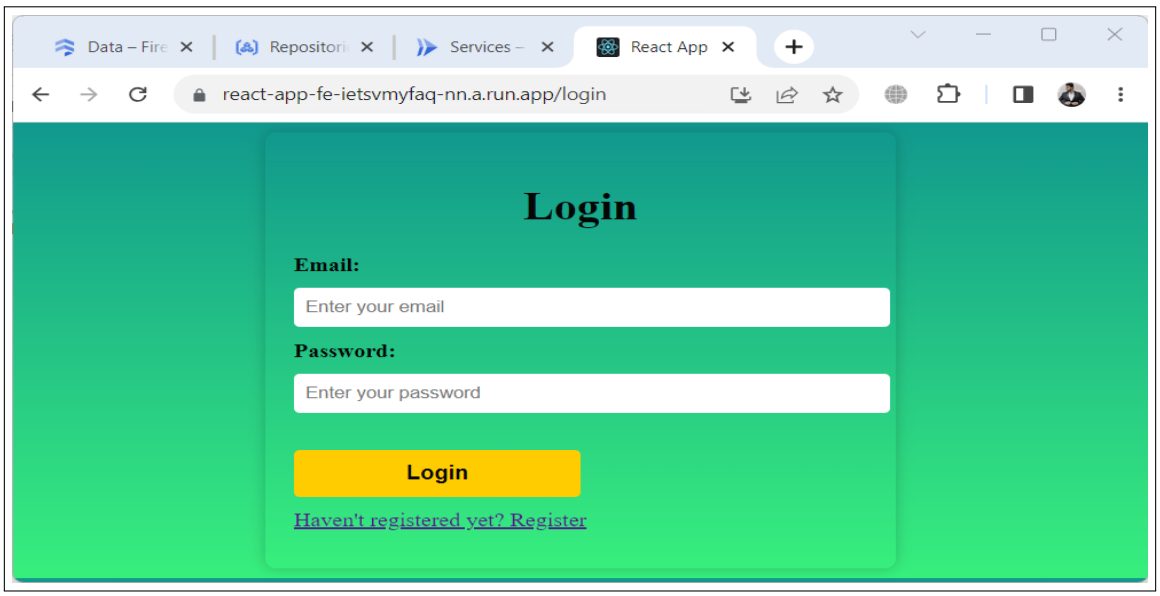

Figure 4.20.: ***Login page***

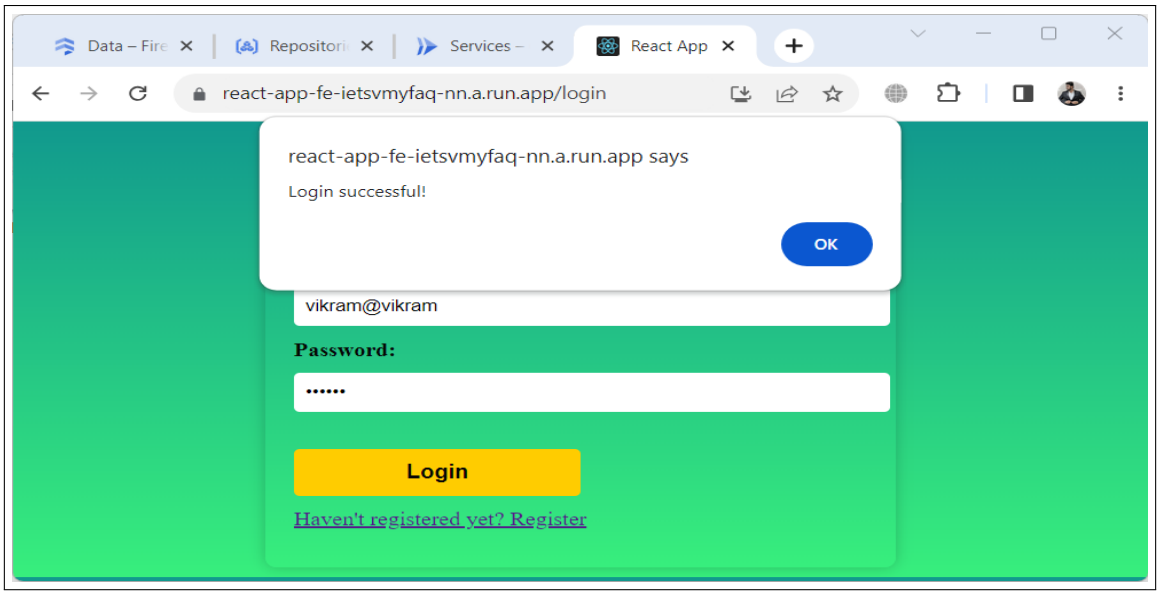

Figure 4.21.: ***User - Vikram: Login Success***

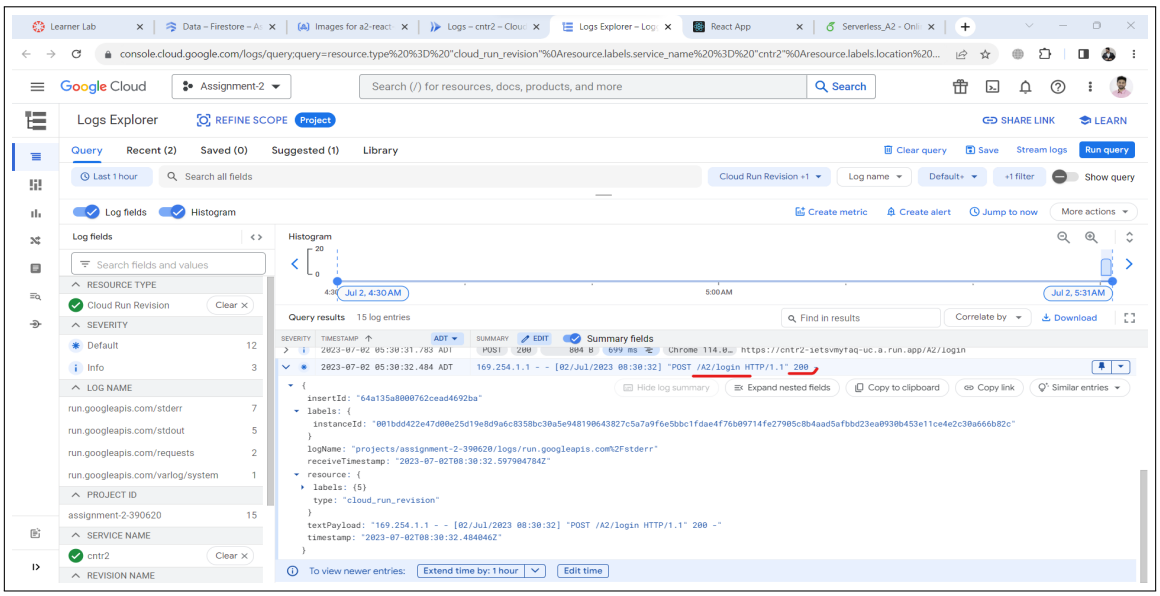

Figure 4.22.: ***User - Vikram: Login logs***

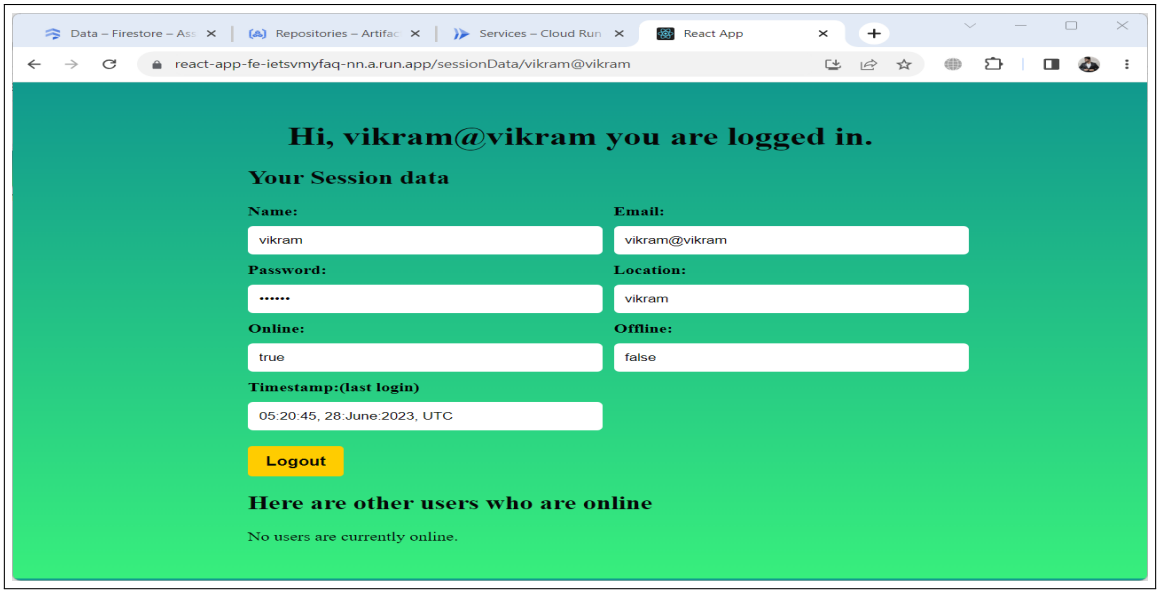

Figure 4.23.: ***User - Vikram: Session data page (no other users online)***

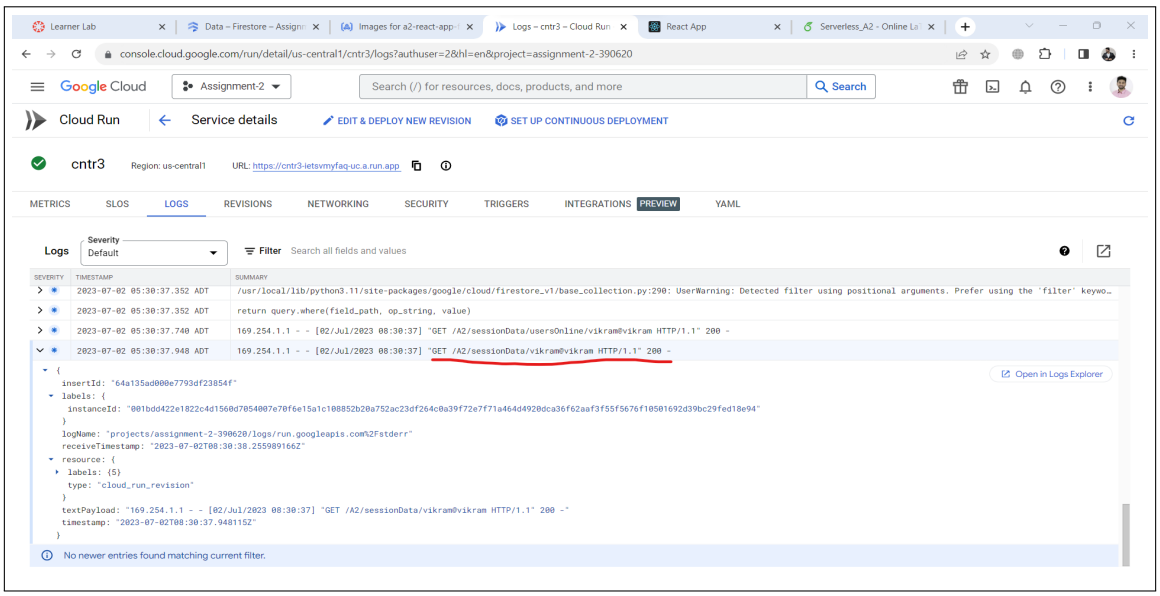

Figure 4.24.: ***User - Vikram: session logs***

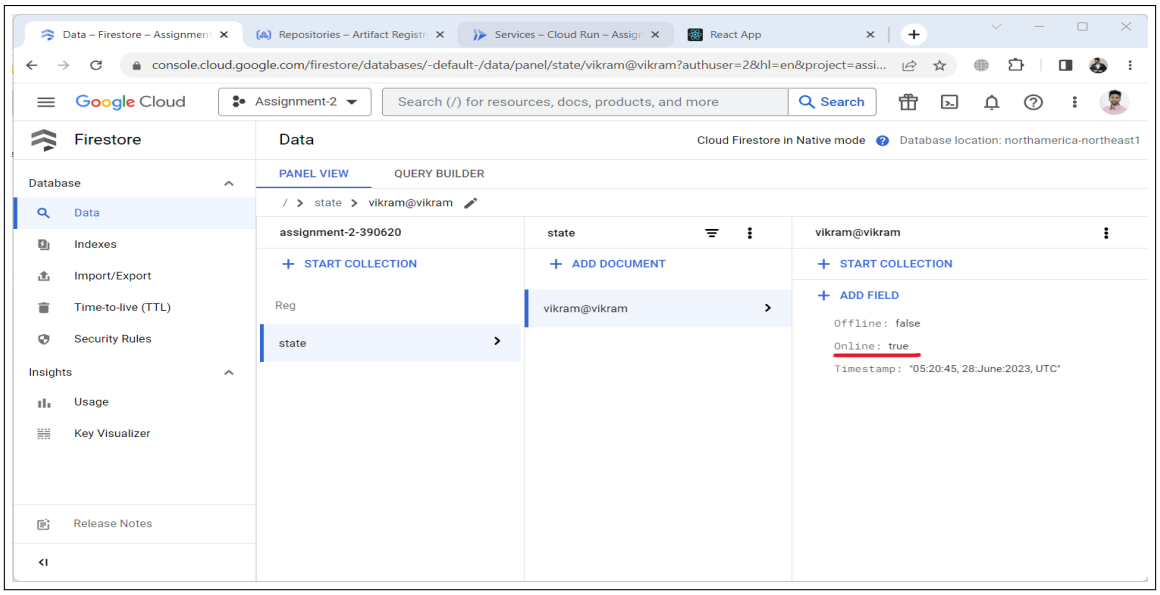

Figure 4.25.: ***User - Vikram: Session data document in "state"***

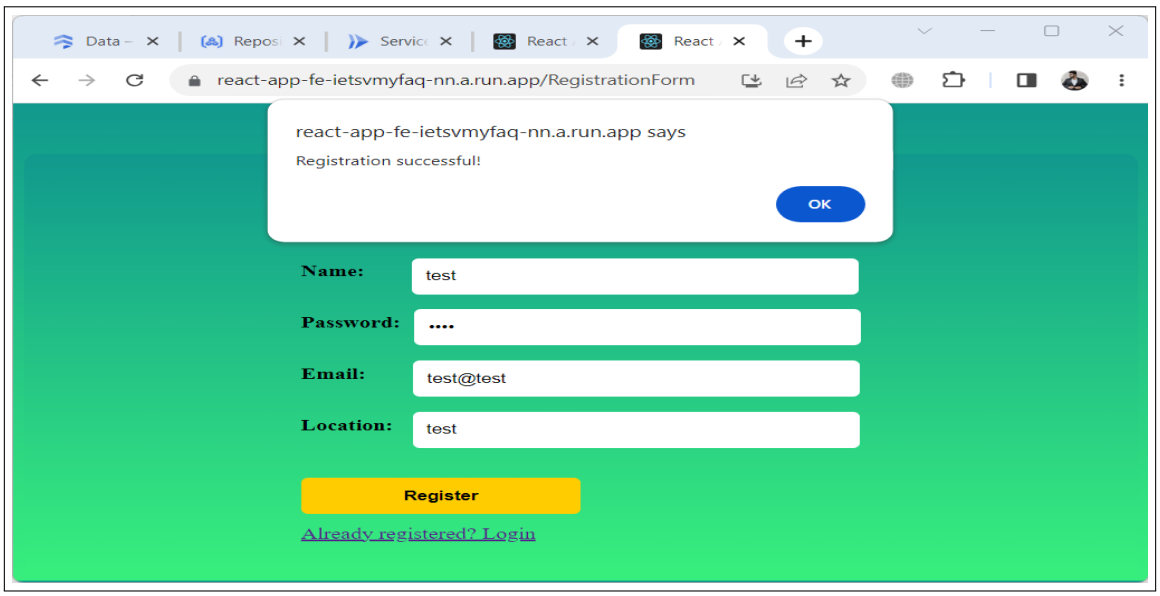

Figure 4.26.: ***User - test: Registration success***

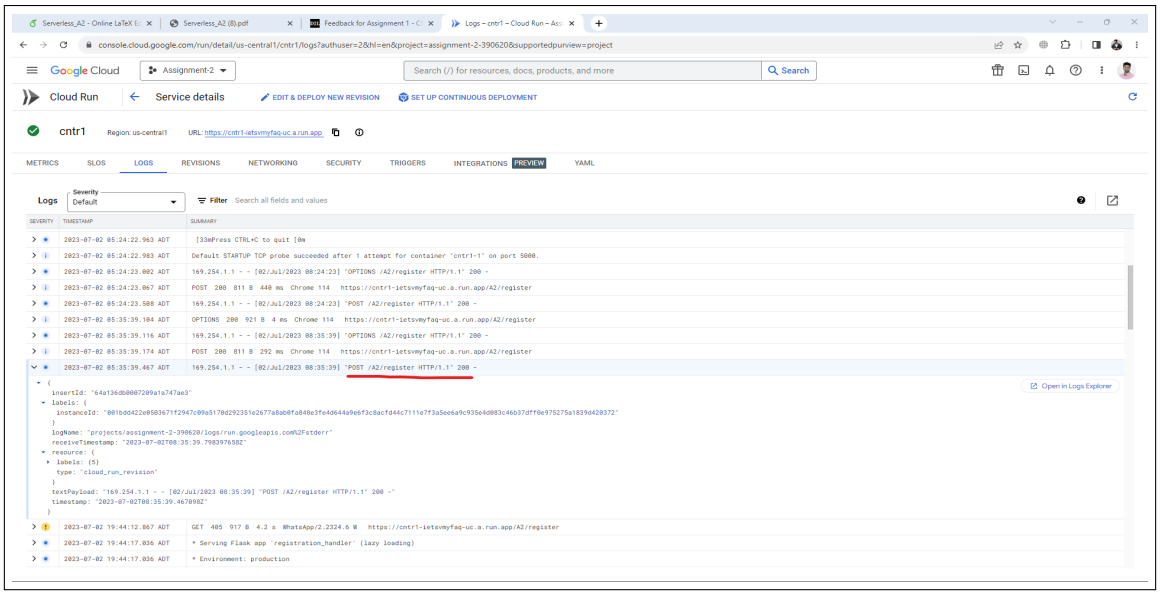

Figure 4.27.: ***User - test: Registration success logs***

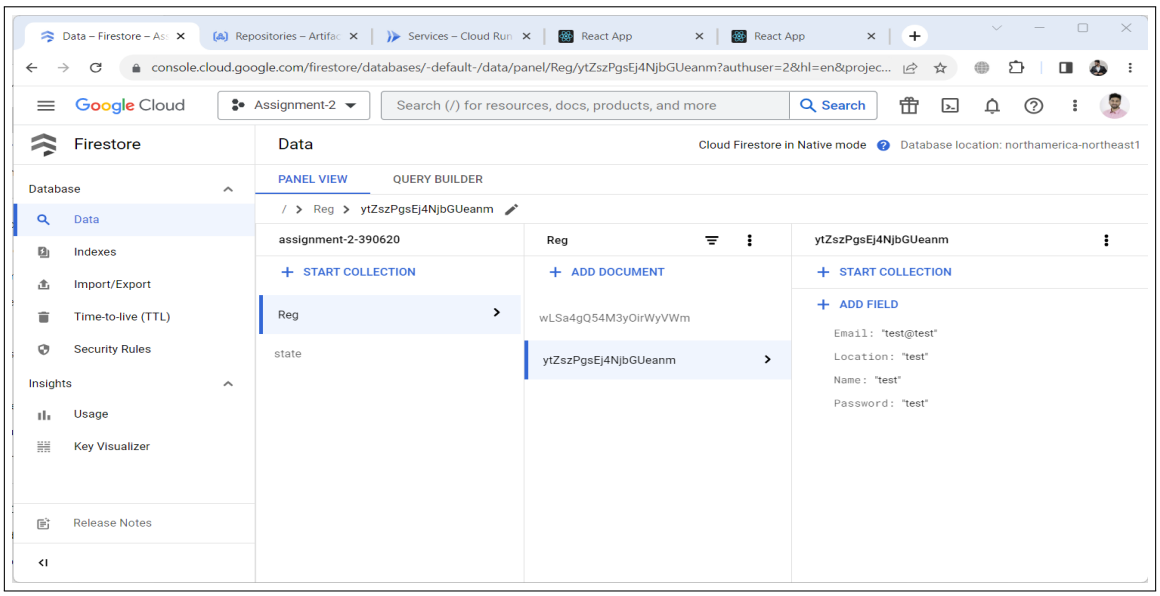

Figure 4.28.: ***User - test: document created in "Reg"***

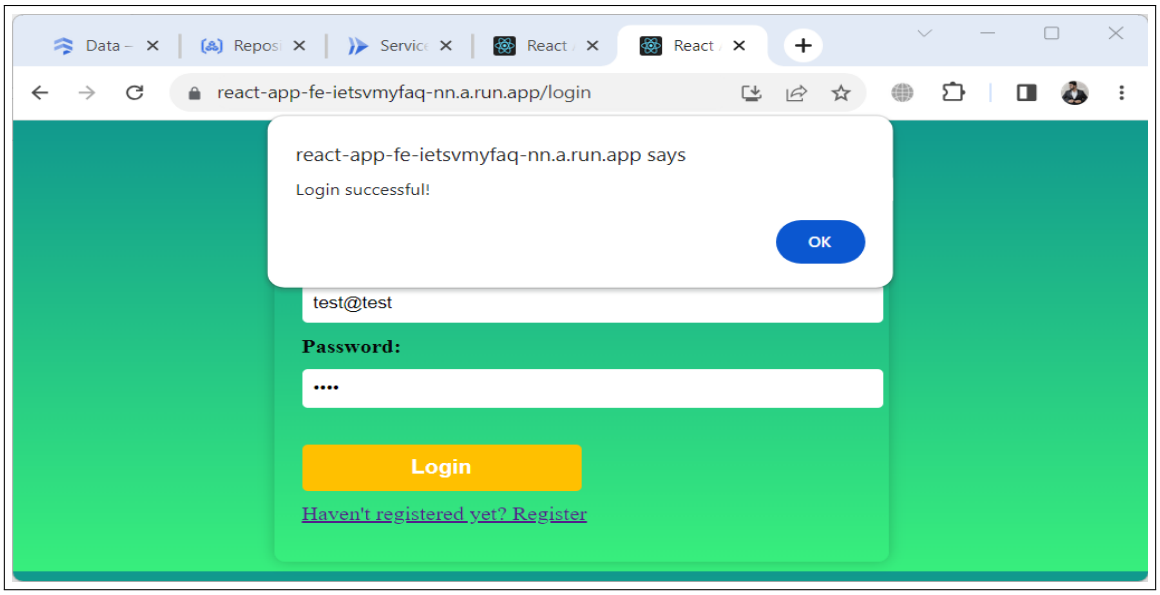

Figure 4.29.: ***User - test: Login success***

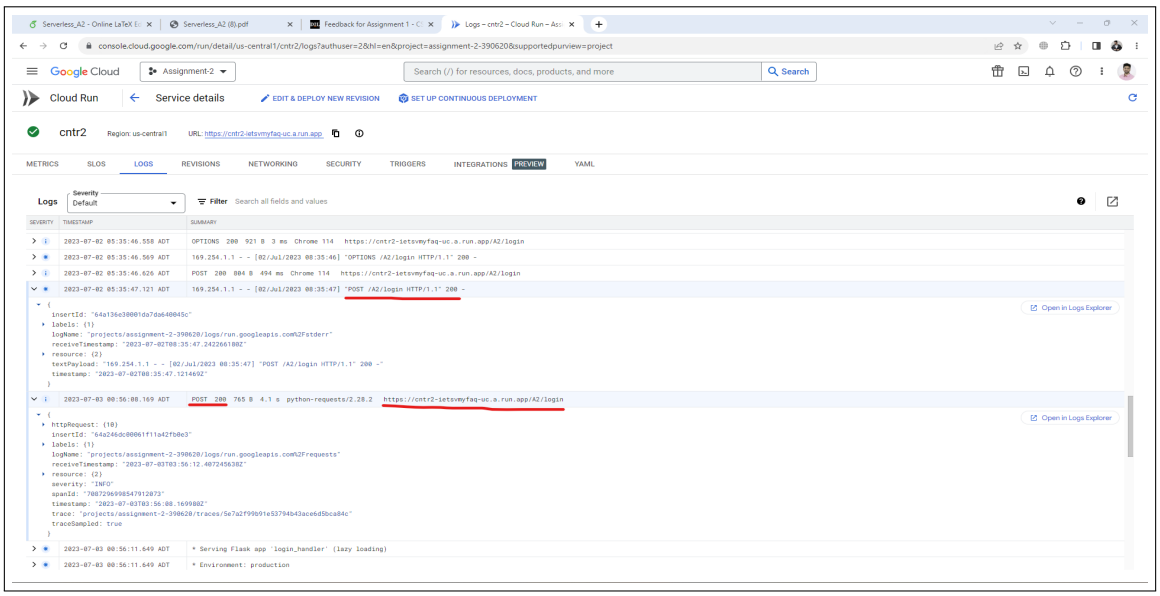

Figure 4.30.: ***User - test: Login success logs***

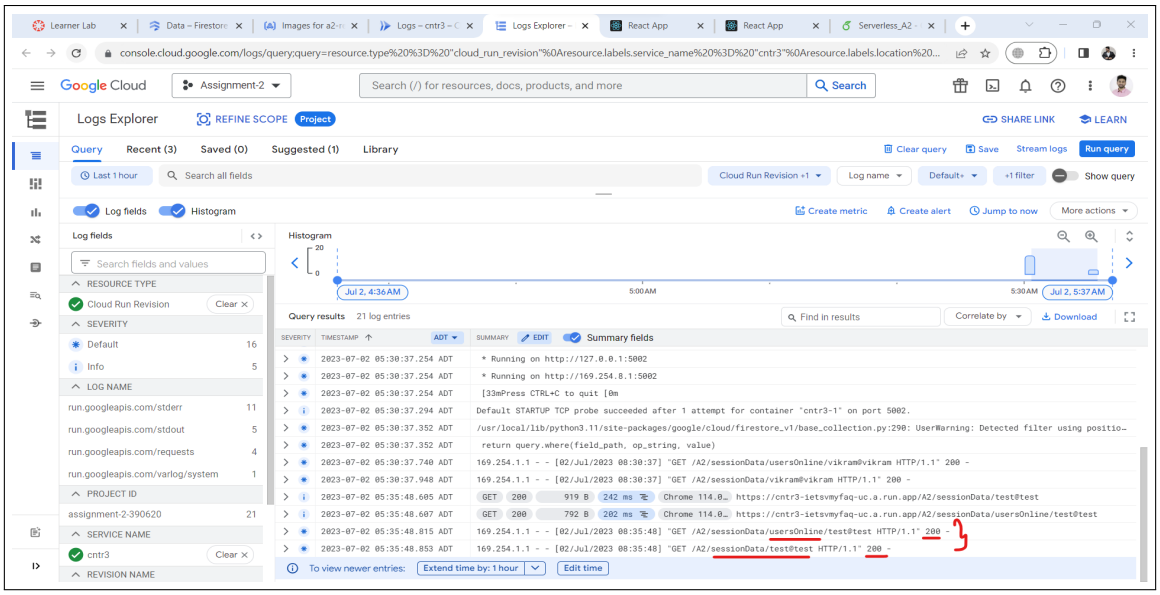

Figure 4.31.: ***User - test: Session data logs***

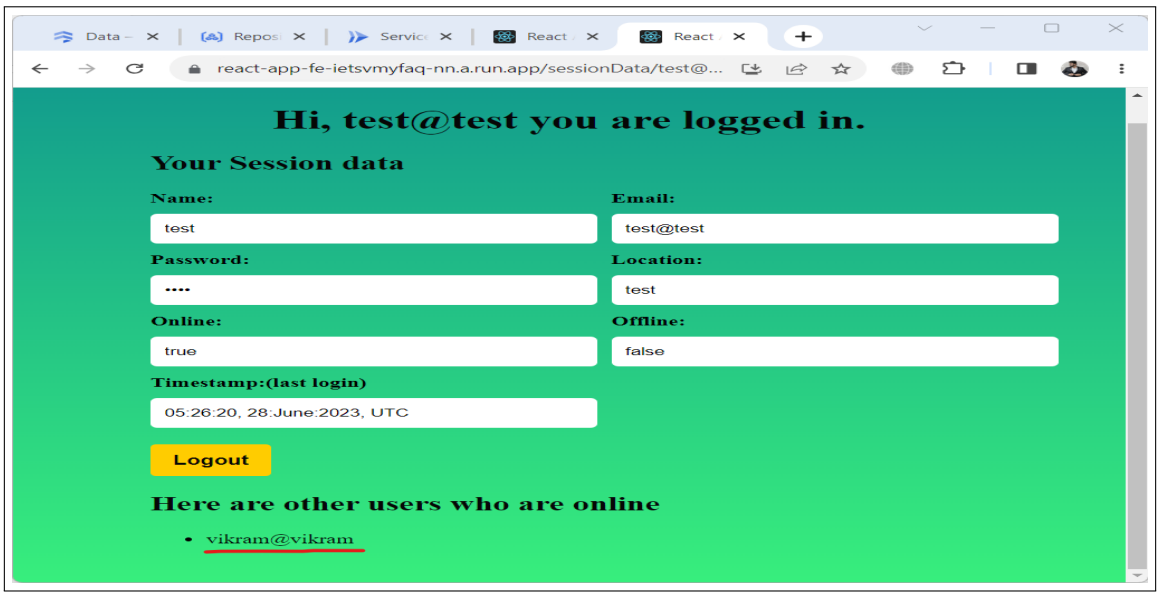

Figure 4.32.: *User - test: session data (other online user(s) - "Vikram" is displaying)*

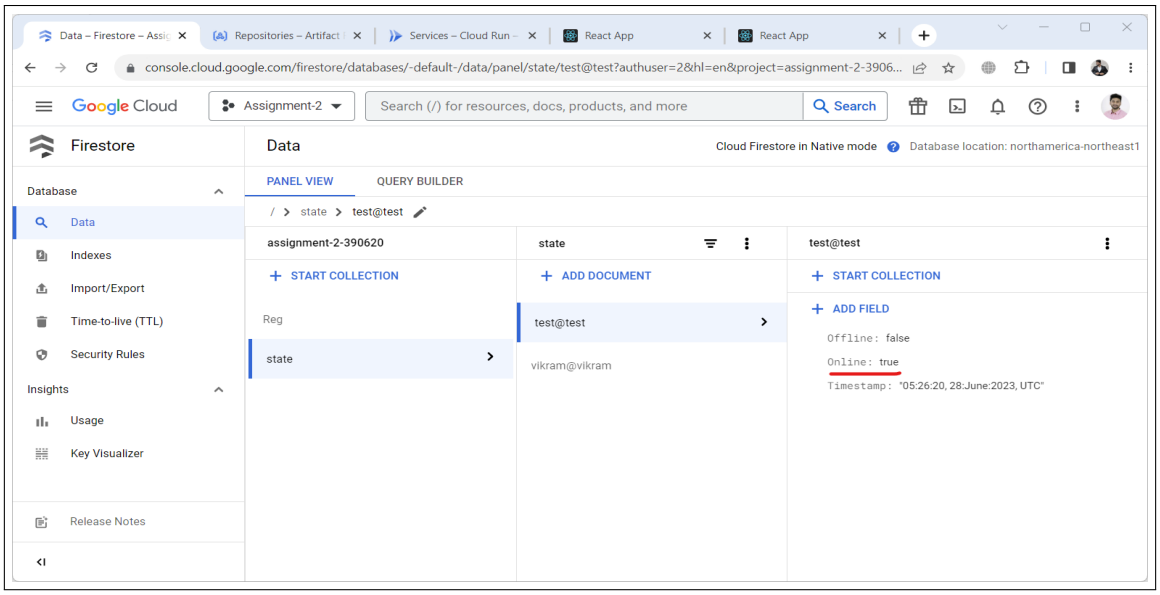

Figure 4.33.: *User - test: document created in "state"*

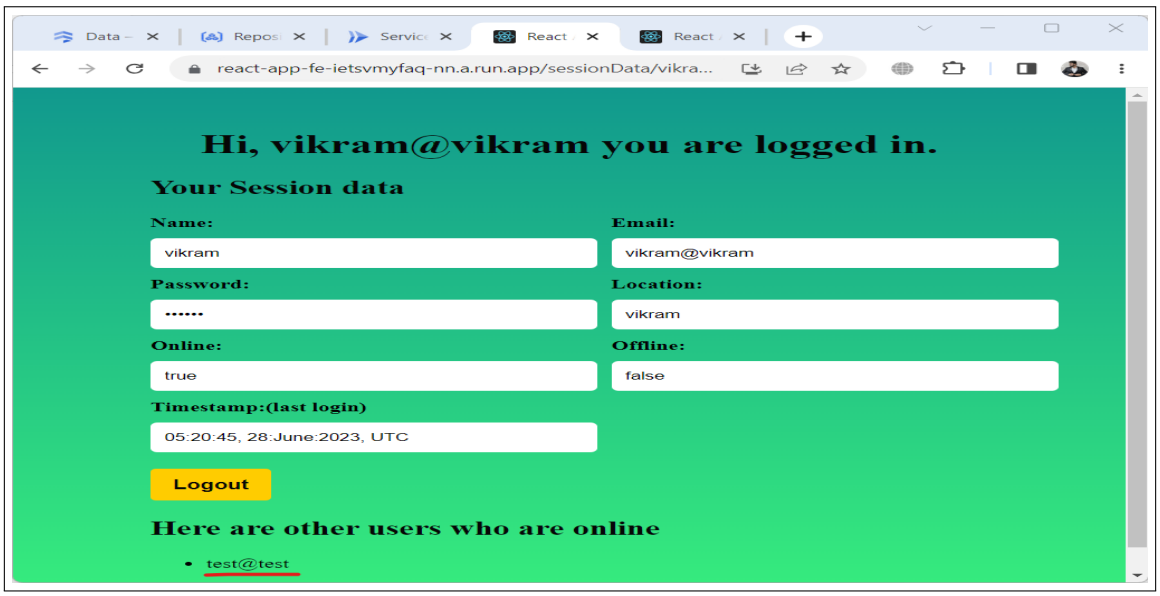

Figure 4.34.: *User - Vikram: session data (other online user(s) - "test" is displaying after refreshing the page)*

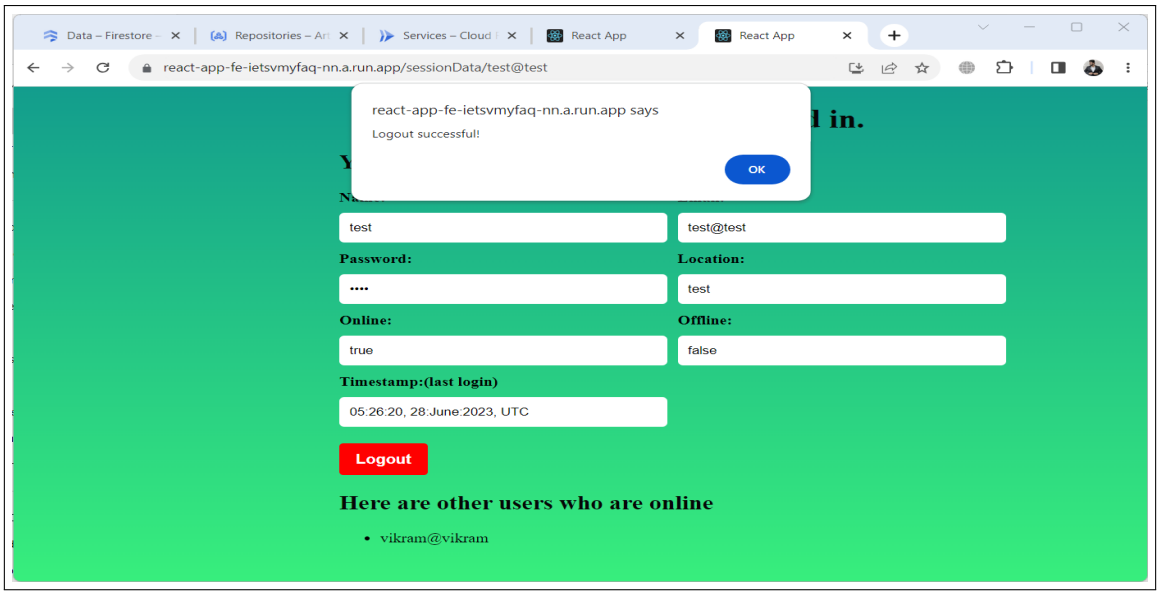

Figure 4.35.: *User - test: Logout success*

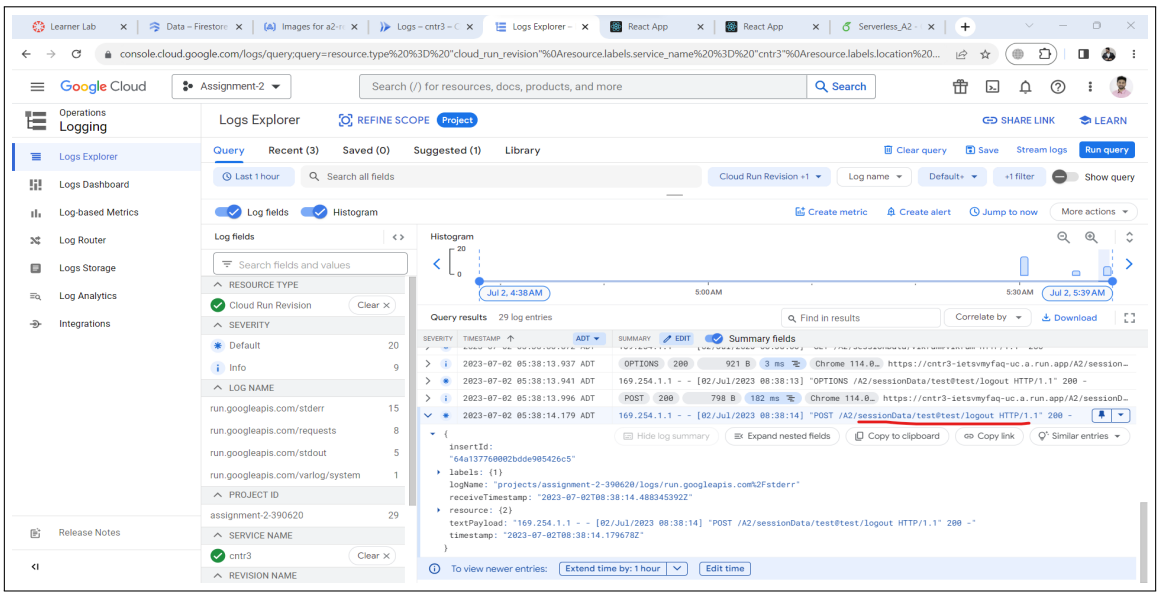

Figure 4.36.: ***User - test: Logout logs***

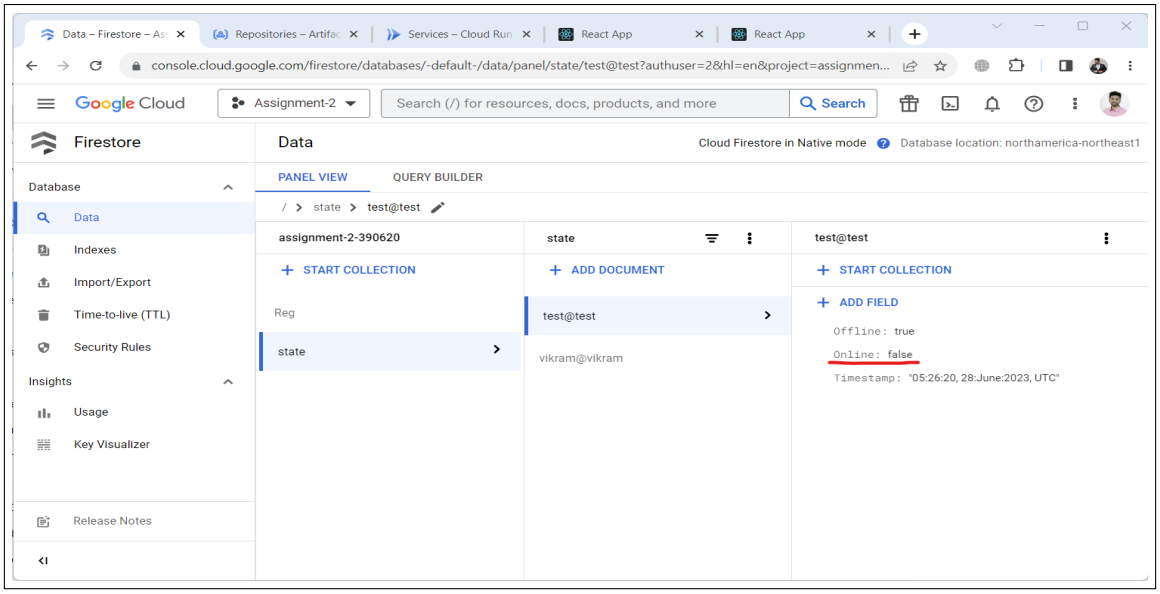

Figure 4.37.: ***User - test: Session is Offline***

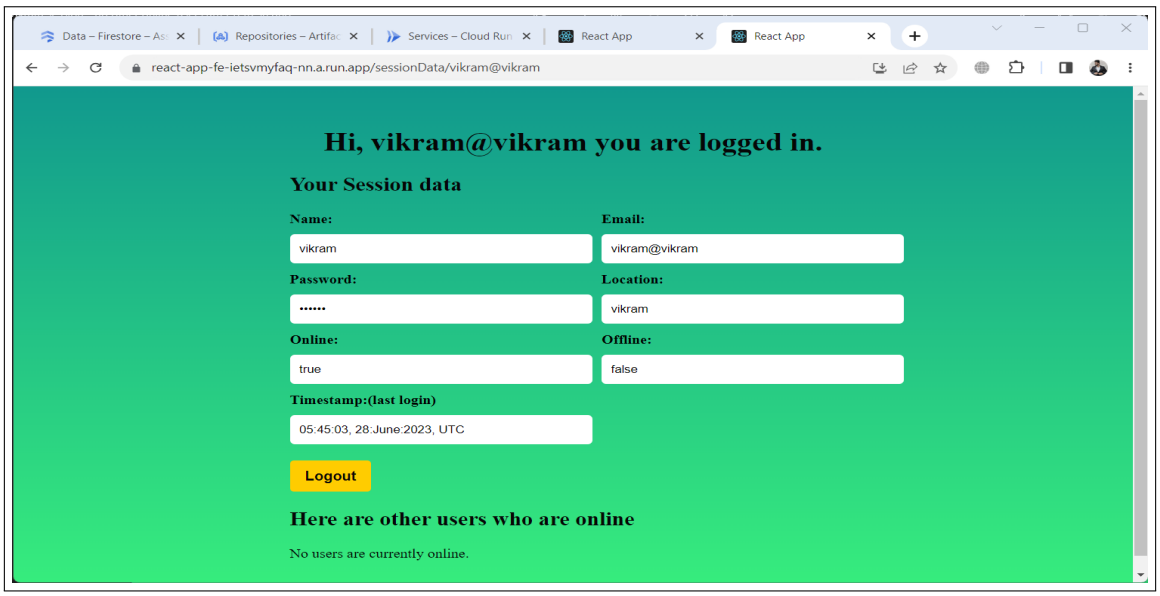

Figure 4.38.: ***User - Vikram: no users are online after user: test logged out (after refreshing the page)***

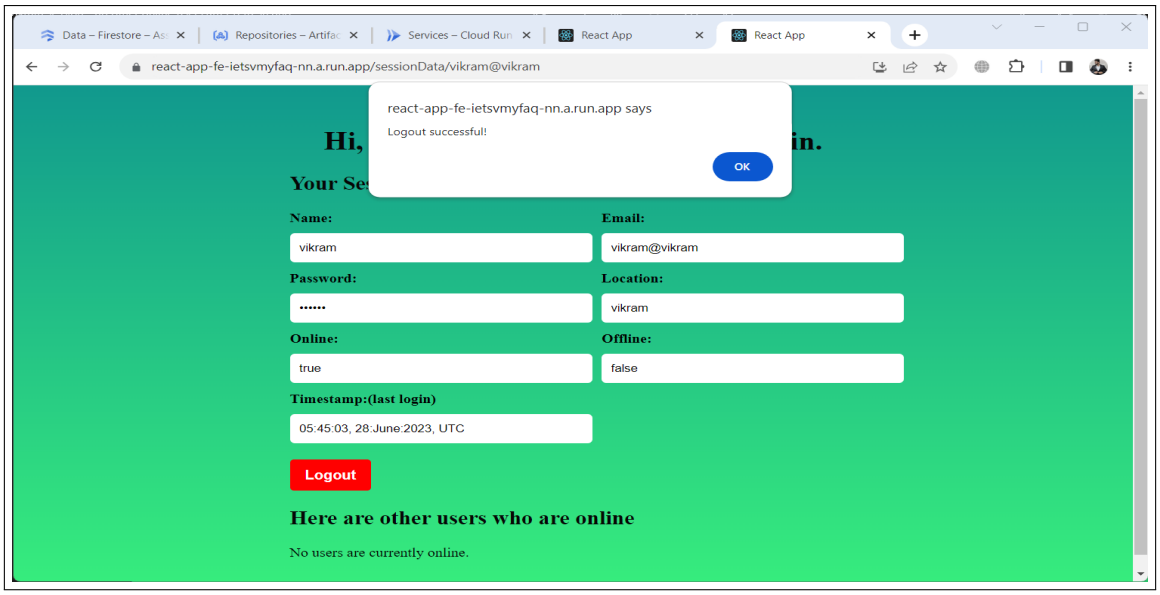

Figure 4.39.: ***User - Vikram: Logout success***

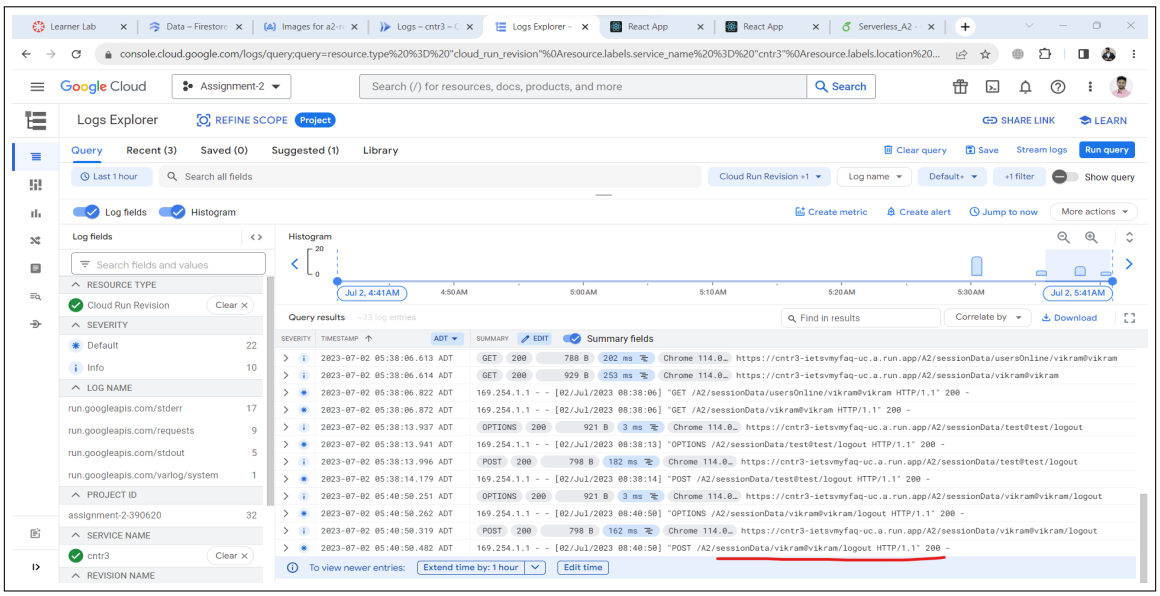

Figure 4.40.: ***User - Vikram: Logout logs***

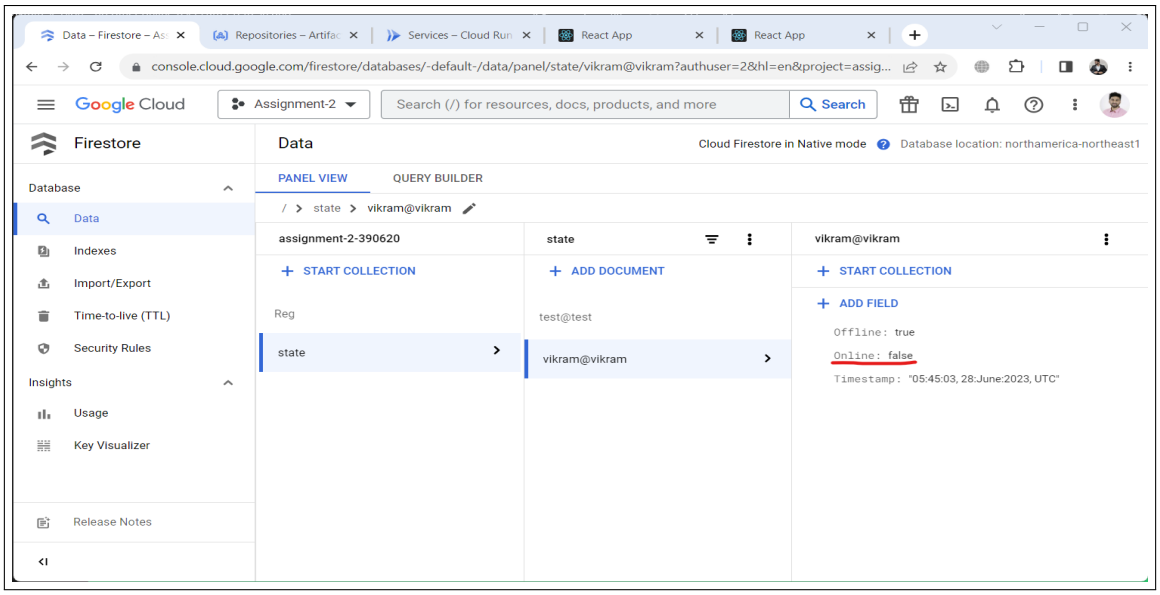

Figure 4.41.: ***User - Vikram: Session is offline***

4.3. Test cases

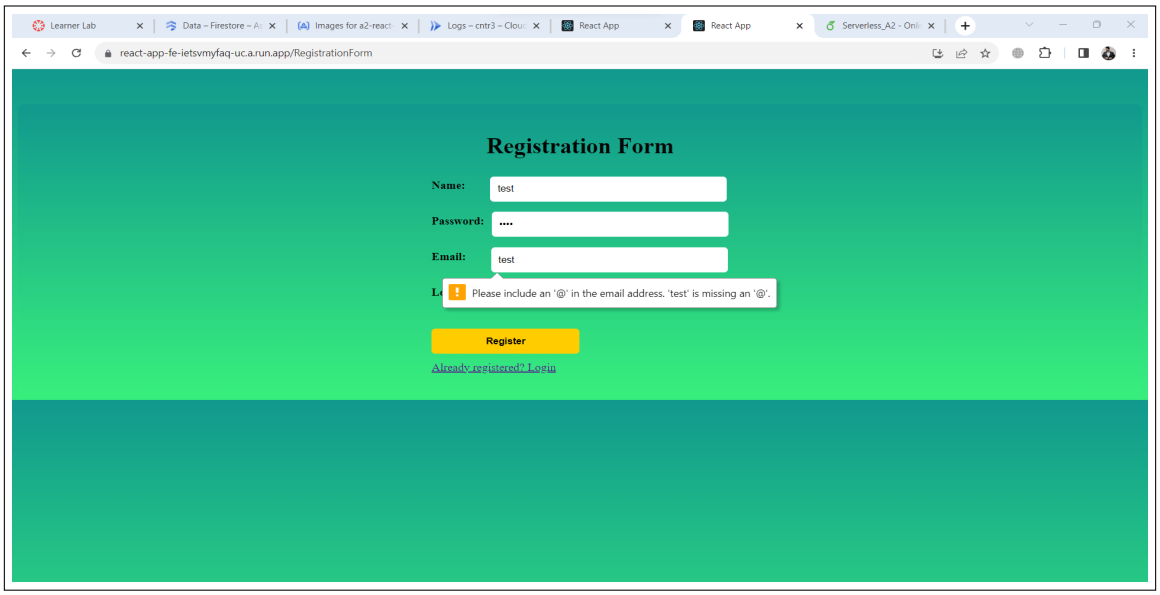

Figure 4.42.: ***Email validation - must have '@' symbol***

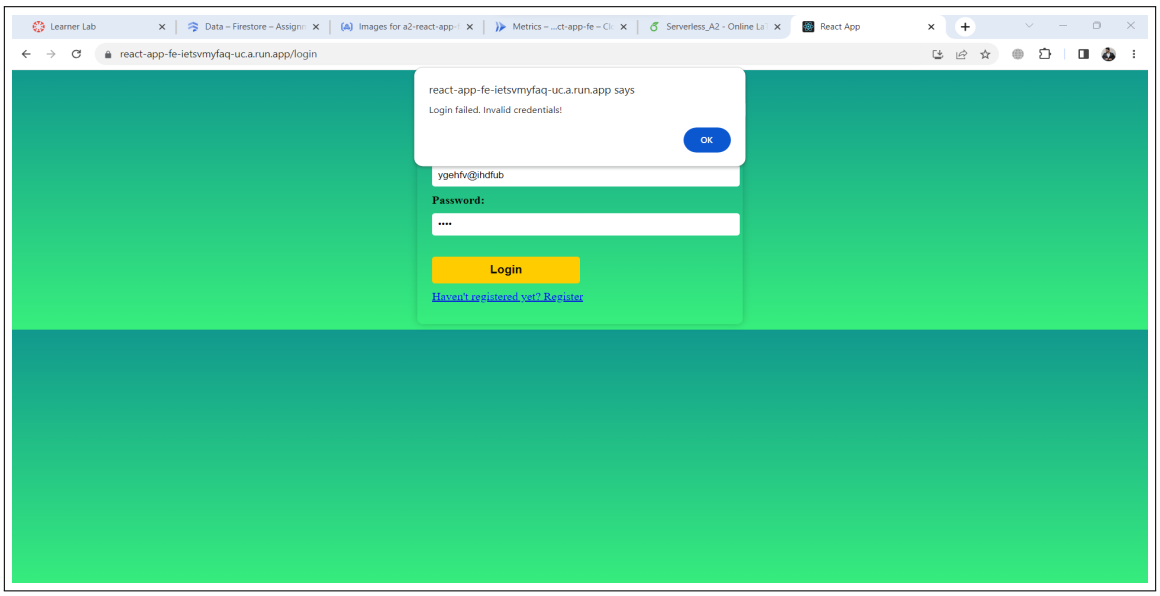

Figure 4.43.: ***Input: random credentials, Error: "Invalid credentials" - Since the record is not present in the collection "Reg"***

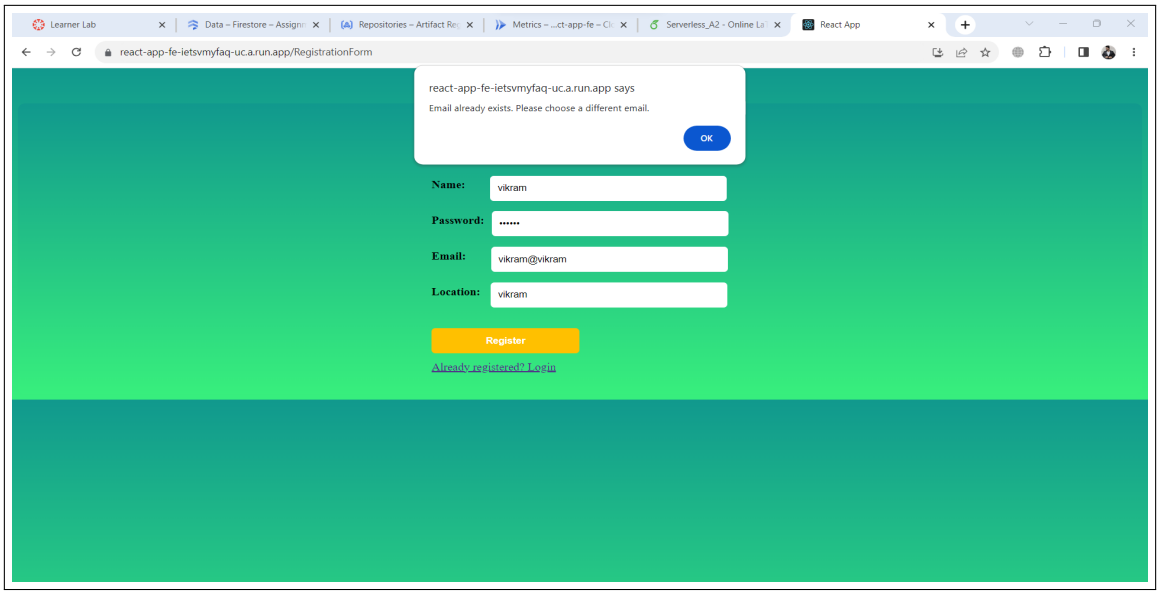

Figure 4.44.: ***Input: registration with an already existing email. Error: "Email already exists" - indicating that the entered email is already present in the collection "Reg"***

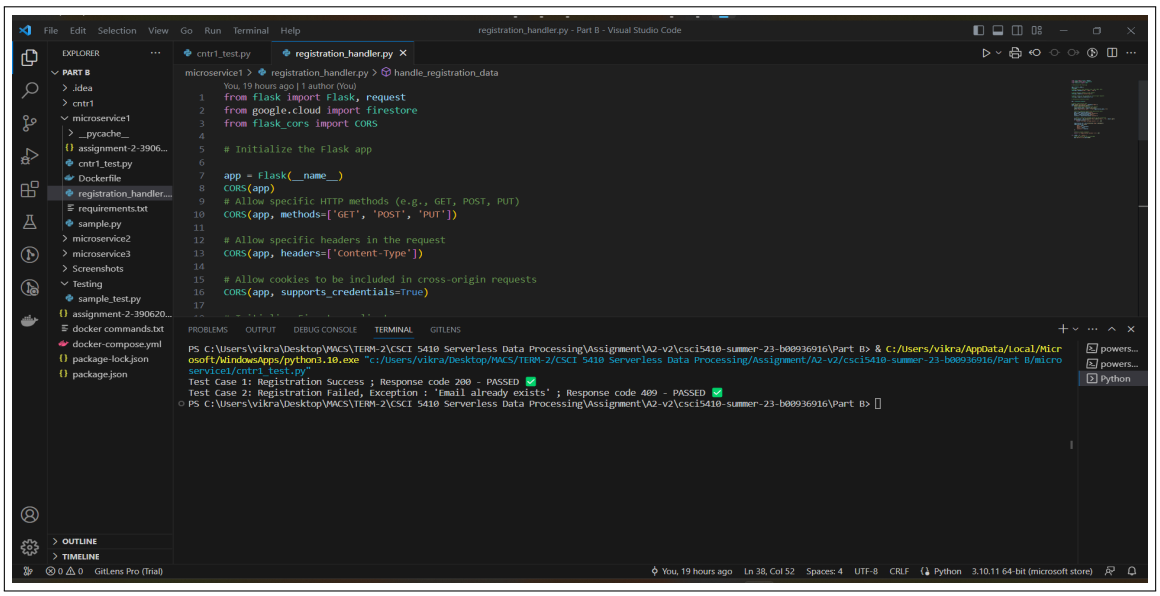

Figure 4.45.: ***Test cases for container 1***

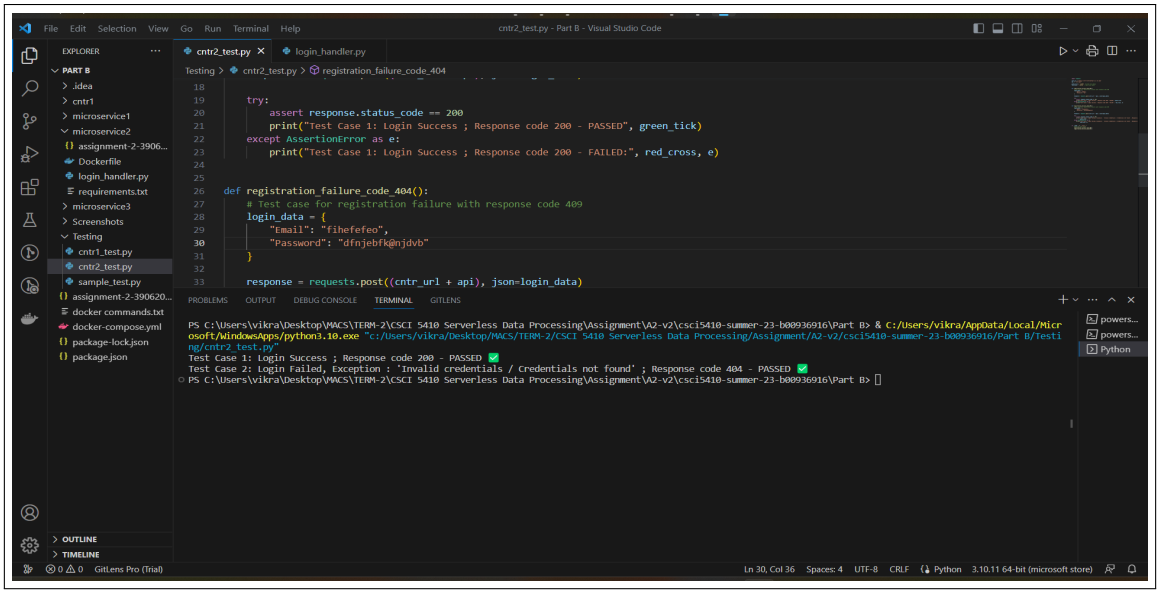

Figure 4.46.: ***Test cases for container 2***

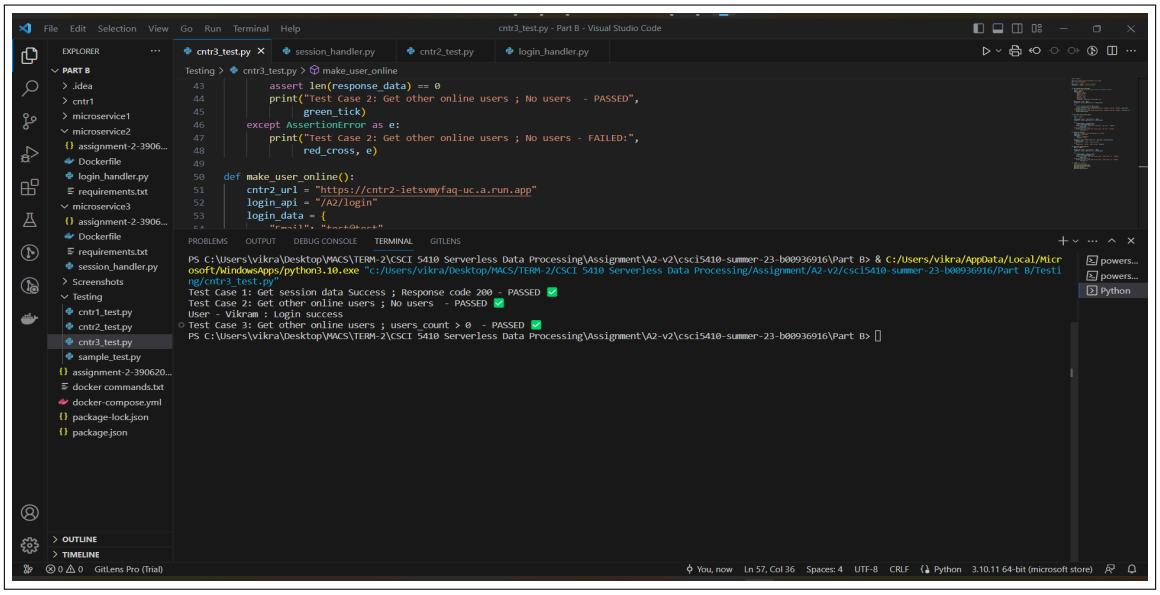

Figure 4.47.: ***Test cases for container 3***

4.4. My view on leveraging Google Cloud Run, GCR/Artifact Registry, and Docker Containers for Efficient Application Deployment

My application relies on the power of Google Cloud technologies to achieve seamless deployment and management. By utilizing Google Cloud Run[3], I can deploy my application as stateless containers, benefitting from automatic scaling and secure execution. This allows me to focus on developing my application code while leaving the infrastructure complexities to Cloud Run.

To store and manage my container images, I utilized Google Artifact Registry[2]. This central repository enabled me to version control my images, control access to containers, and efficiently distribute them across different environments. With easy push and pull capabilities[8], I can ensure smooth deployment and updates[7] of my container images.

Docker containers[6] play a vital role in the deployment process. They encapsulate the application code, dependencies, and configurations, making deployments consistent and portable. Docker's containerization technology provides isolation, scalability, and optimized resource utilization, contributing to improved performance and maintainability of the application.

My personal observation:
During the process of pushing Docker images to Artifact Registry, I observed an interesting behavior. When pushing the "cnt2" images, I noticed that instead of uploading all the layers from scratch, the system fetched some layers from the previously pushed "cntr1" images (Refer figures 4.2,4.4,4.6). Since these layers were identical, reusing them saved valuable time and resources. This optimization in the image pushing process demonstrates the efficiency of Google Artifact Registry in handling and

managing container images, further enhancing the overall deployment experience.

In conclusion, my application benefits from Google Cloud Run's scalability, Artifact Registry's image management capabilities, and Docker containers' consistency and portability. These technologies work harmoniously to provide me with an efficient, secure, and easily manageable application infrastructure. Additionally, Docker and Google Cloud Platform offer extensive documentation and resources, which enabled me to maximize the benefits of these tools in my application development and deployment process.

Part III.

PART C

Building a Chatbot: Using AWS Lex

5. Bot RideRequest

5.1. Actions taken to perform the bot creation operation

To create the "**TaxiRequest**" and "**SelfDriveRequest**" intents in AWS Lex, log in to your AWS account and access the AWS Management Console. From there, open the Amazon Lex service and either create a new bot. Once inside the bot, navigate to the "**Intents**" section and click on "**Create Intent**." For the "TaxiRequest" intent, name it accordingly and add sample utterances that users might say when requesting a taxi ride. Create the necessary slots to collect information like **pickup address, pickup date**, **pickup time**, and **vehicle type**. Configure prompts for each slot to prompt the user for the required details. Similarly, for the "SelfDriveRequest" intent, name it accordingly and add sample utterances related to self-drive requests. Create slots to collect information such as **arrival time**, **vehicle type**, and **number of vehicles**. Configure prompts for each slot to obtain the necessary details from the user. Define c**onfirmation prompts** for both intents using placeholders for variable values to confirm the user's request. Set up the **fulfillment logic** for each intent, including appropriate success and failure messages. Save the intent configurations and **build the bot** to apply the changes. These steps allow us to create the "TaxiRequest" and "SelfDriveRequest" intents in AWS Lex and we can customize them according to our requirements. Finally, **test the bot**, to check if it satisfies all the requirements.

5.2. Intents

5.2.1. TaxiRequest

The "TaxiRequest" intent allows users to request a taxi ride. It collects information such as the customer's address for pickup, pickup date, and pickup time.

Sample utterances:

Can you call me a taxi?
I need a taxi to the airport.
I need a ride to the train station.
I'm looking for a taxi service.

Create Custom Slot type Vehicle

1. Create a new, Custom slot type in the bot
2. Slot name: Vehicle

3. Slot type values(restrict to these values only) :
    a) Key = suv ; Values = SUV, suv, Suv
    b) Key = minivan ; Values = Minivan, MiniVan, minivan

Slots & Prompts

Slots	**Prompts**	**Slot Type**
CustomerAddress	Sure, you've selected a taxi ride. Could you please provide your address for pickup?	AMAZON.FreeFormInput
PickupDate	Thank you! When do you need the taxi? Please provide the pickup date.	AMAZON.Date
PickupTime	Great! What time would you like the taxi to arrive? Please provide the pickup time.	AMAZON.Time
VehicleType	What do you want today (SUV, Sedan, Minivan)?	AMAZON.Vehicle(custom built)
NumberOfVehicles	How many?	AMAZON.Number

Confirmation prompt:

Use this prompt with placeholders for variable values (indicated using {}), to get confirmation from the user(YES/NO).

```
You have requested for {NumberOfVehicles} {VehicleType} for pickup address
{CustomerAddress}. Your pickup date is {PickupDate}, and your pickup time is {
    PickupTime}. Please confirm.
```

Listing 5.1: Code snippet

Fulfillment :

On success	Your request has been placed successfully.
In case of failure	Oops!! Sorry, your request could not be placed at this moment. Please try again later.

5.2.2. SelfDriveRequest

The SelfDriveRequest intent is used to request a self-drive ride or car rental. It collects information about the vehicle type, number of vehicles, and pickup time.

Sample utterances:

I want to request a self-drive ride.
I'm looking to rent a car for the day.
I need a car to pick up my kids from school.
I'm going to the airport and need a car to get there.

Slots & Prompts

Slots	Prompts	Slot Type
ArrivalTime	Sure, you've selected a self-drive ride. When are you coming TODAY to get your vehicle?	AMAZON.Time
VehicleType	What do you want today (SUV, Sedan, Mini-van)?	AMAZON.Vehicle(custom built)
NumberOfVehicles	How many?	AMAZON.Number

Confirmation prompt:

Use this prompt with placeholders for variable values (indicated using {}), to get confirmation from the user (YES/NO).

```
You have requested for {NumberOfVehicles} {VehicleType}, and you will be
   arriving at {ArrivalTime}. Please confirm.
```

Listing 5.2: Code snippet

Fulfillment :

On success	Your request has been placed successfully.
In case of failure	Oops!! Sorry, your request could not be placed at this moment. Please try again later.

5.3. Screenshots

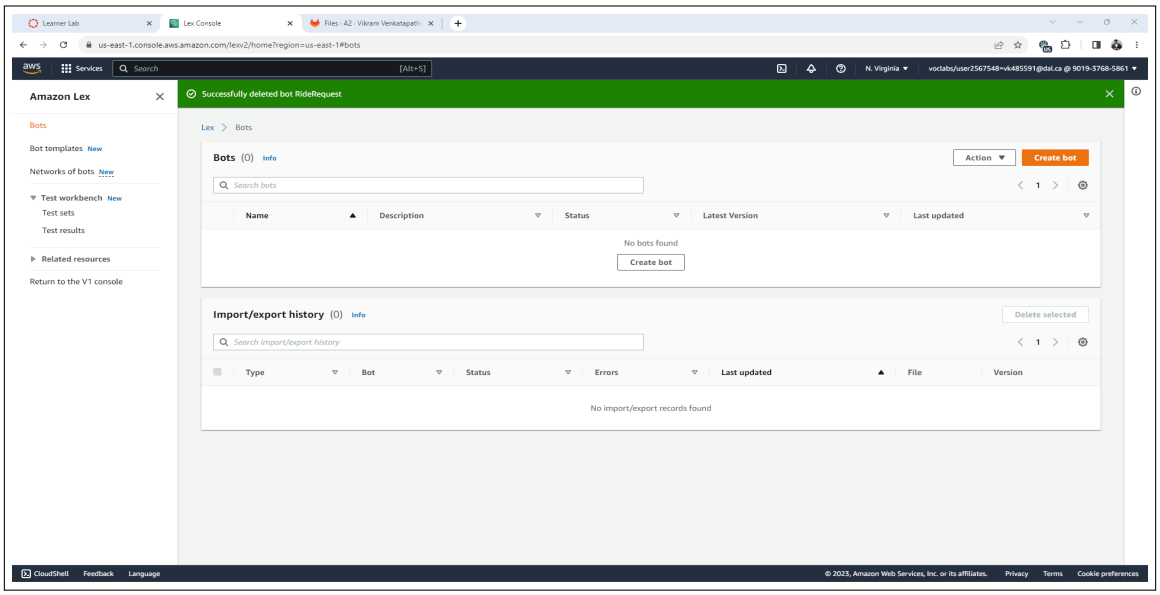

Figure 5.1.: ***Empty bots section***

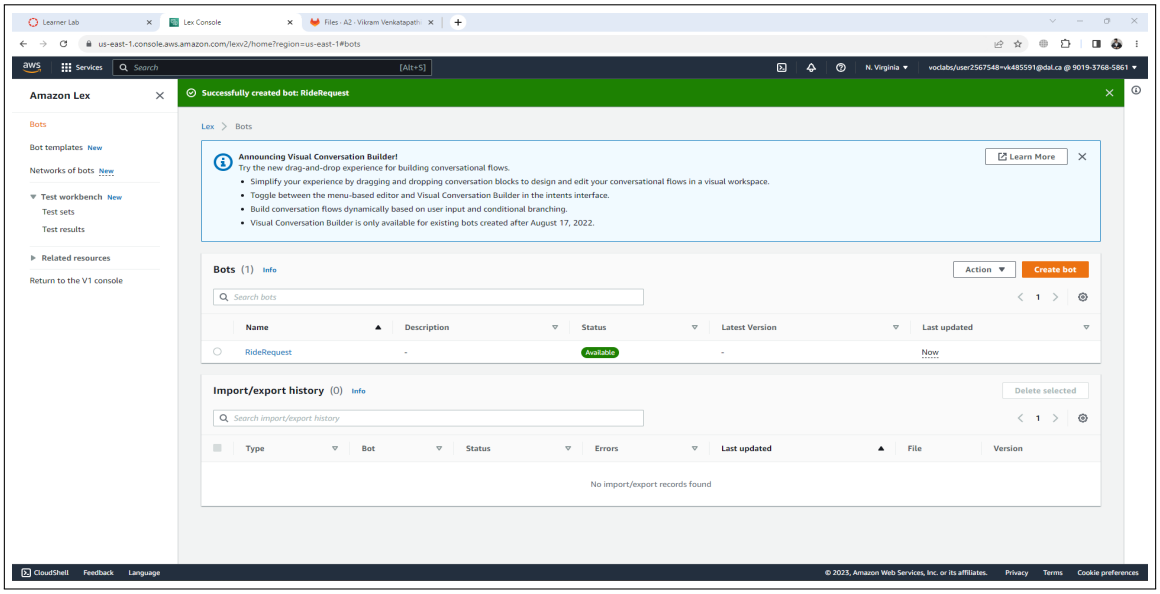

Figure 5.2.: ***Bot created***

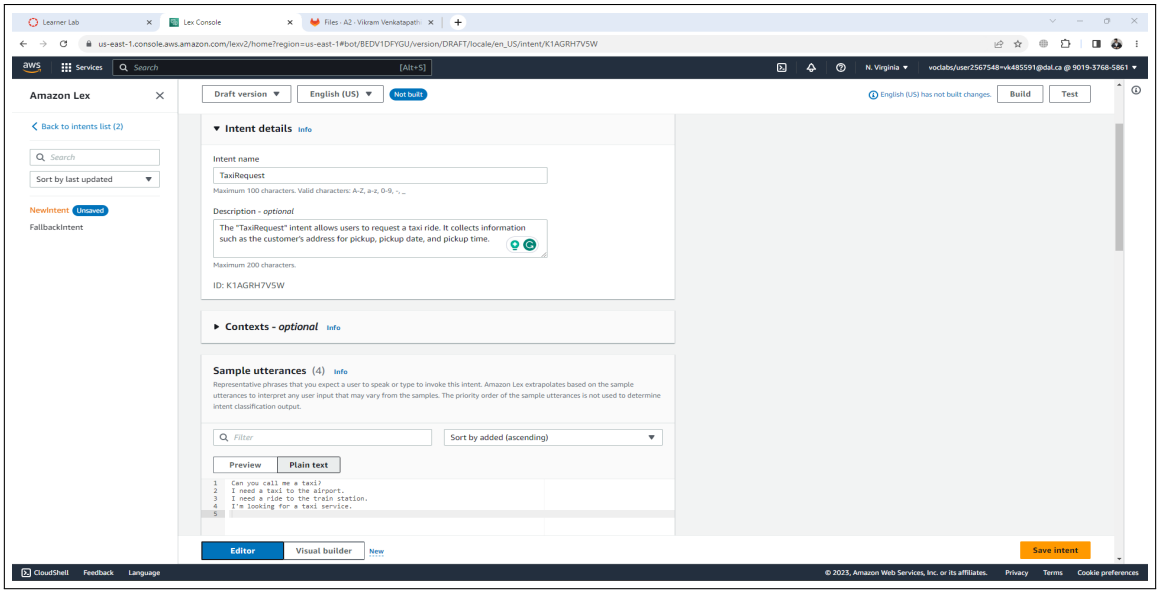

Figure 5.3.: ***TaxiRequest intent creation***

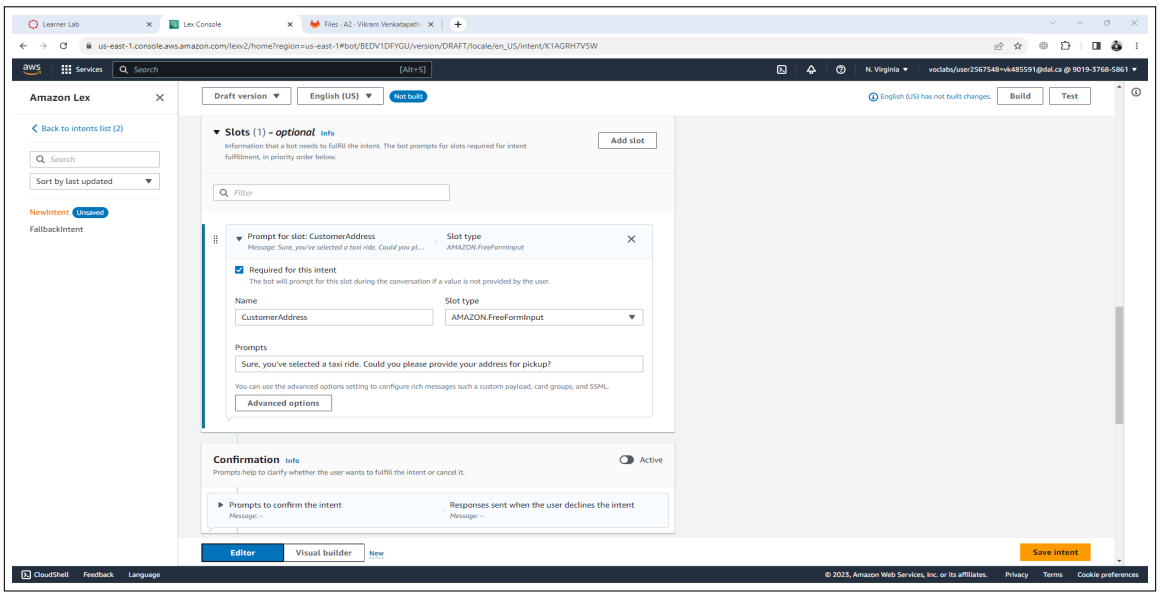

Figure 5.4.: ***TaxiRequest intent creation - CustomerAddress slot***

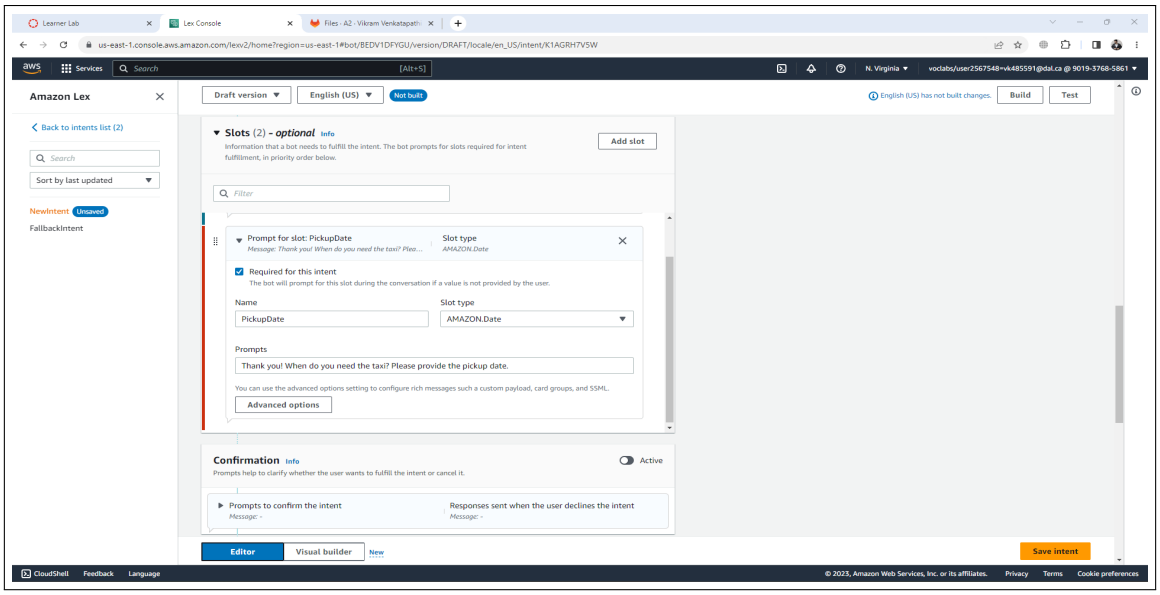

Figure 5.5.: ***TaxiRequest intent creation - PickupDate slot***

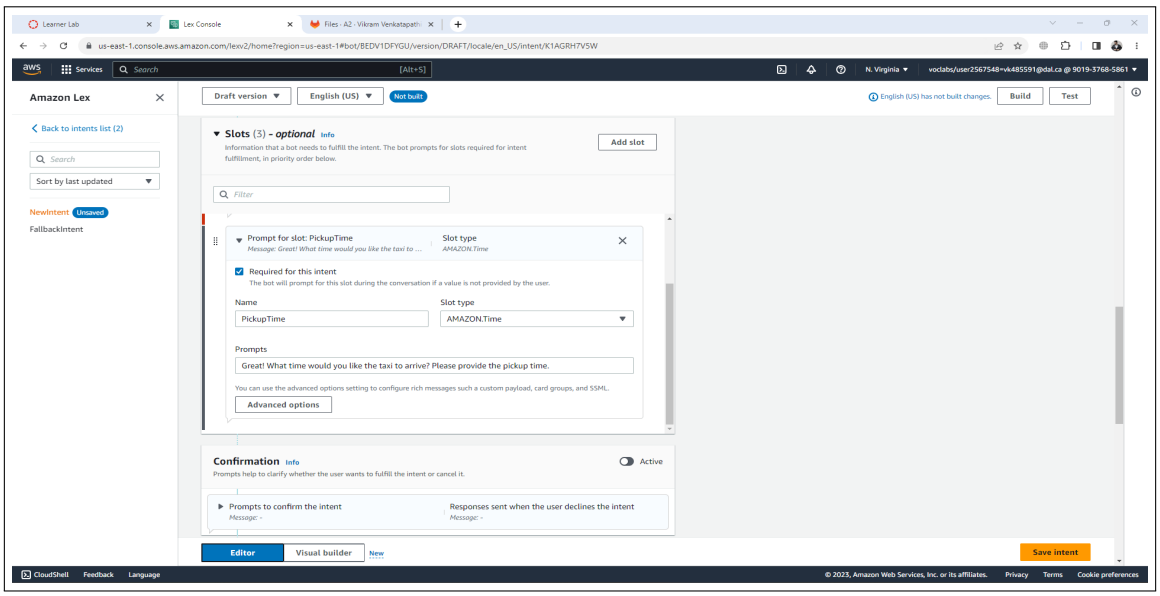

Figure 5.6.: ***TaxiRequest intent creation - PickupTime slot***

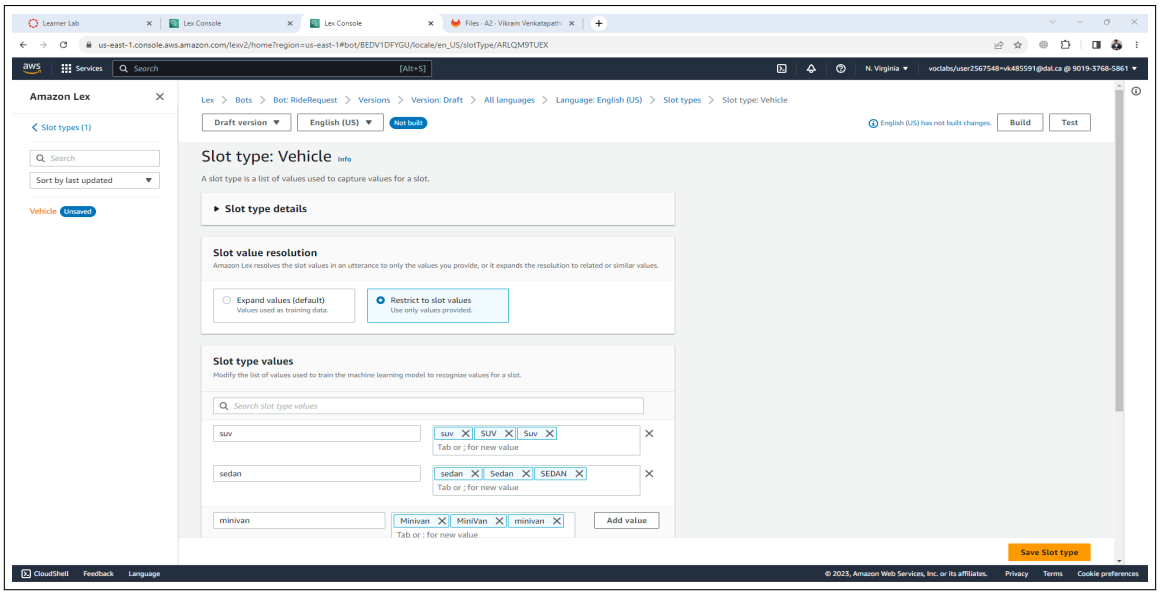

Figure 5.7.: ***TaxiRequest intent creation - Custom slot type 'Vehicle' creation***

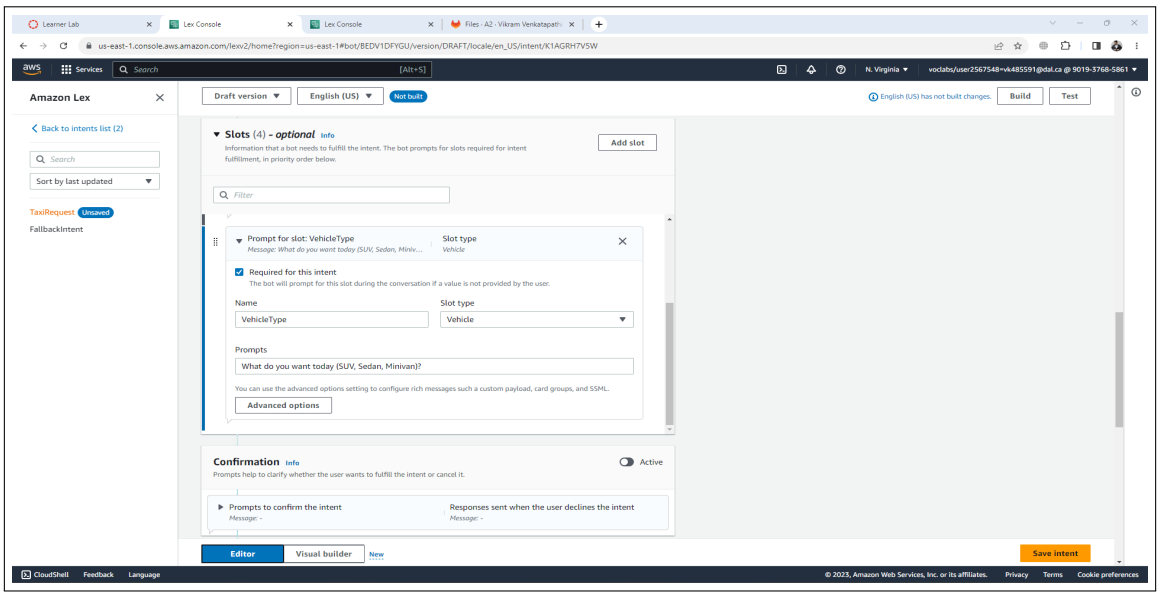

Figure 5.8.: ***TaxiRequest intent creation - VechicleType slot***

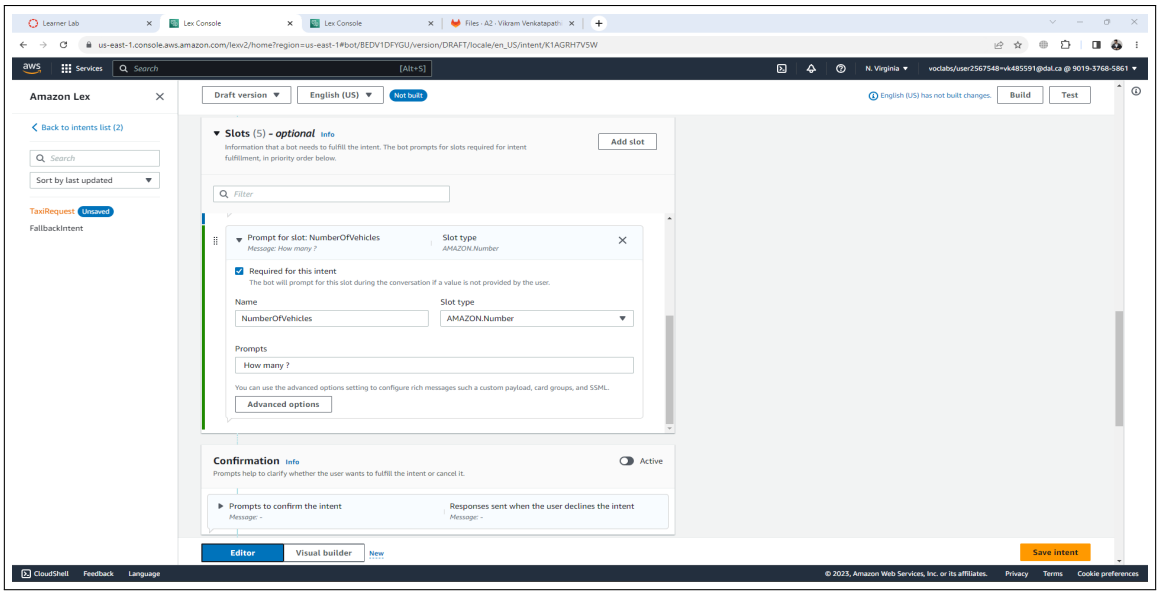

Figure 5.9.: ***TaxiRequest intent creation - NumberOfVehicles slot***

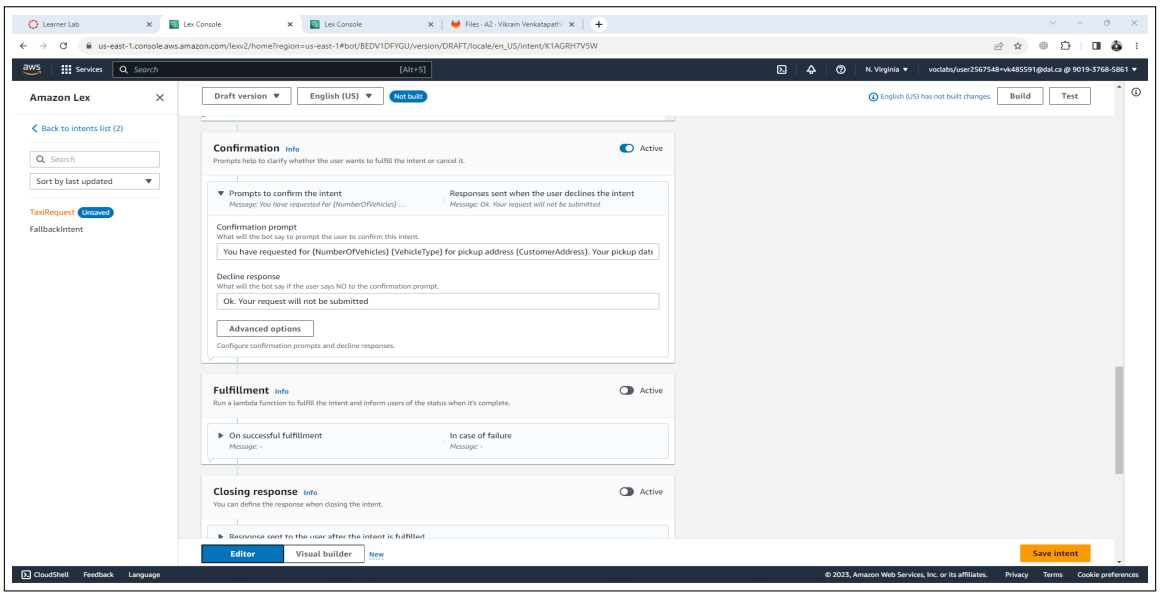

Figure 5.10.: ***TaxiRequest intent creation - Add Confirmation prompt with placeholder values***

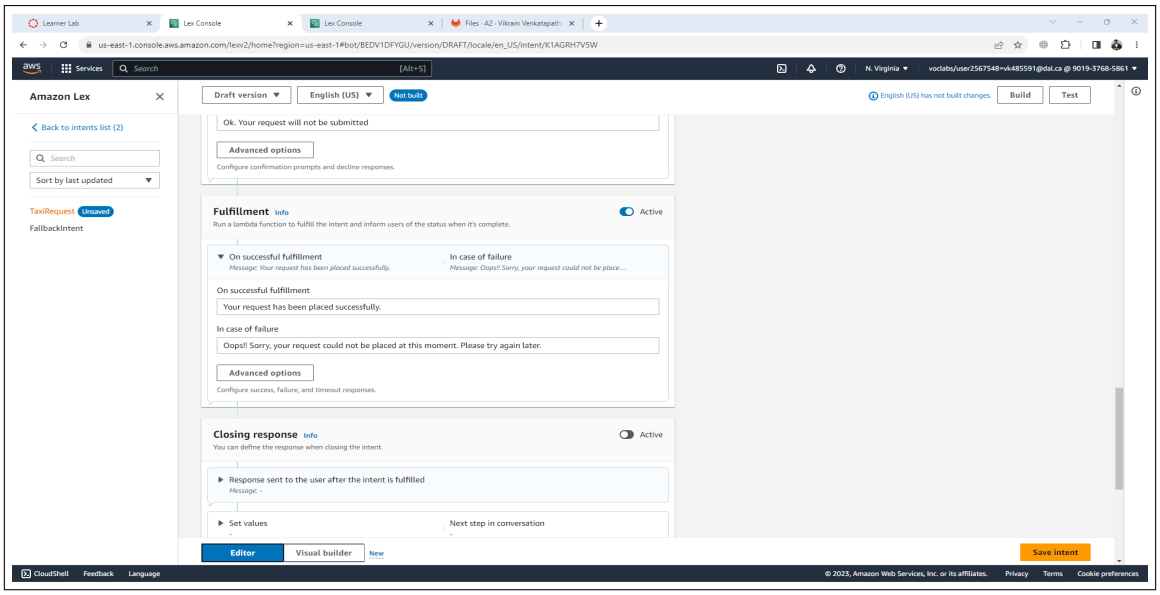

Figure 5.11.: ***TaxiRequest intent creation - Add fullfillment prompt***

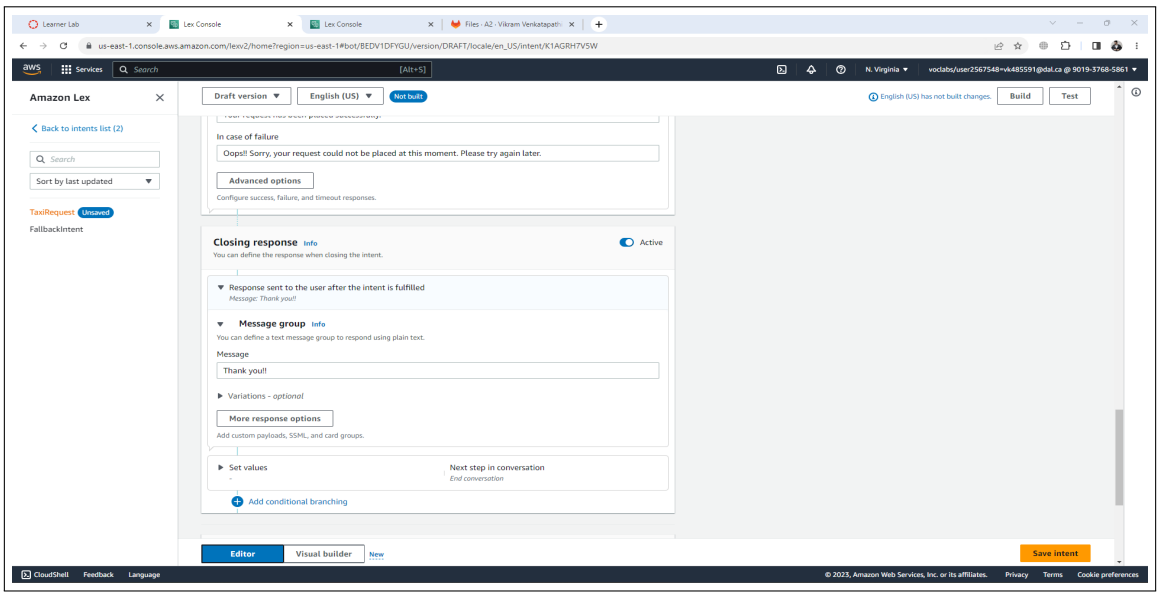

Figure 5.12.: ***TaxiRequest intent creation - Add closing response***

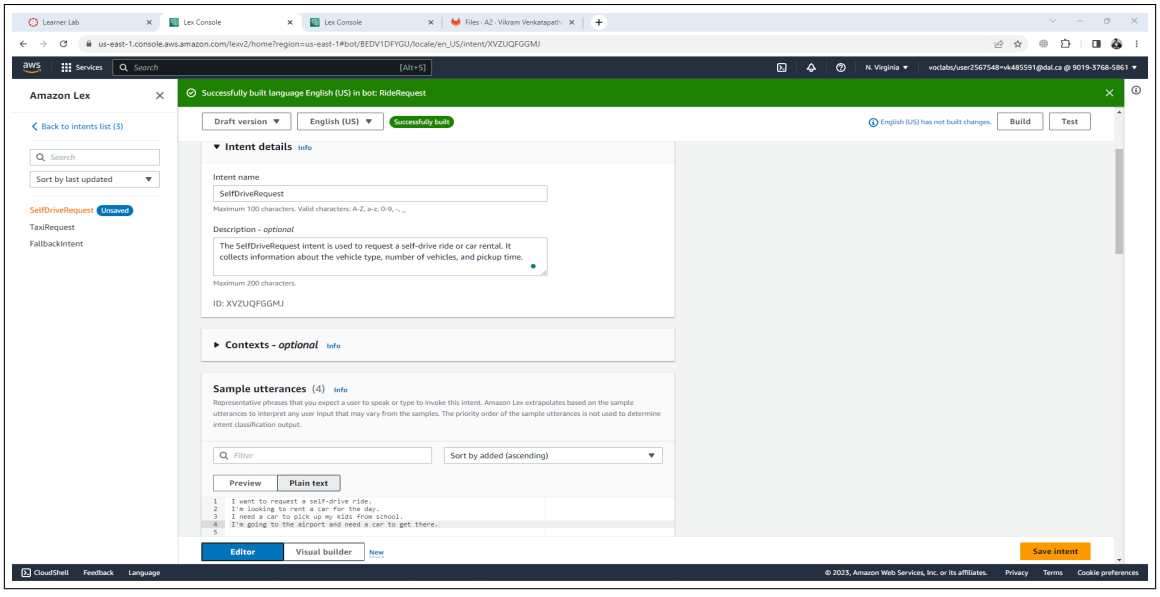

Figure 5.13.: ***SelfDriveRequest intent creation***

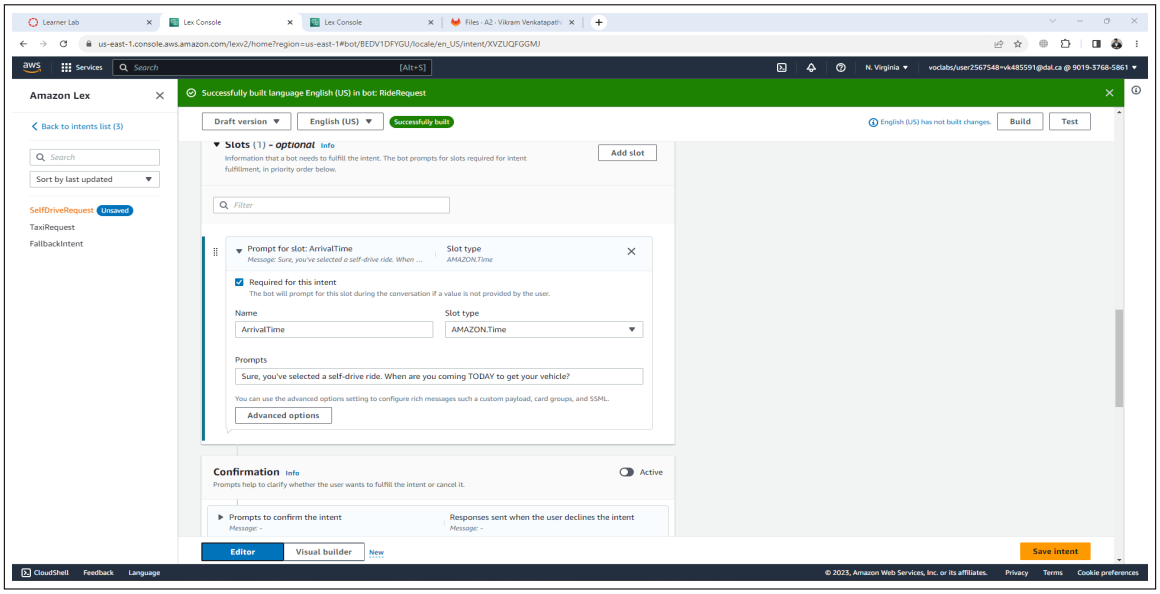

Figure 5.14.: ***SelfDriveRequest intent creation - ArrivalTime slot***

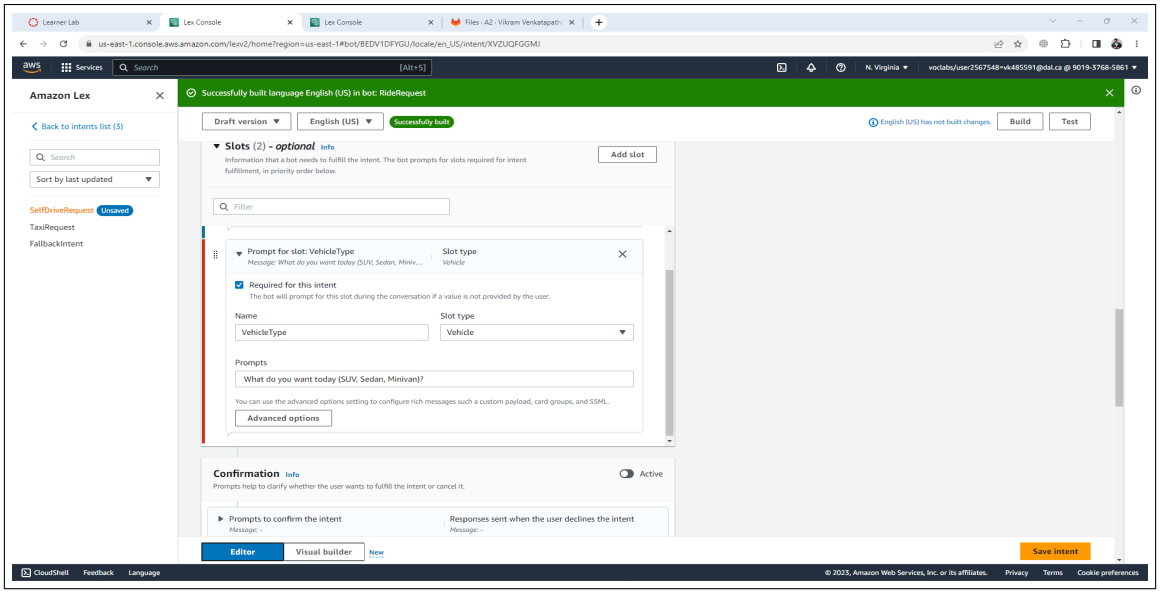

Figure 5.15.: ***SelfDriveRequest intent creation - VehicleType slot***

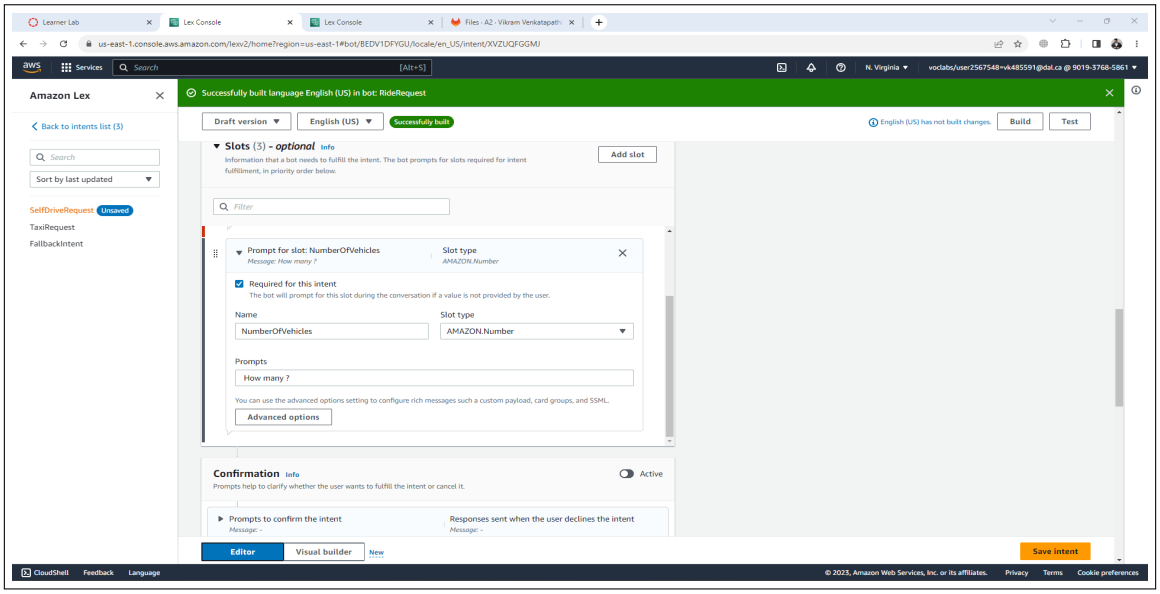

Figure 5.16.: ***SelfDriveRequest intent creation - NumberOfVehicles slot***

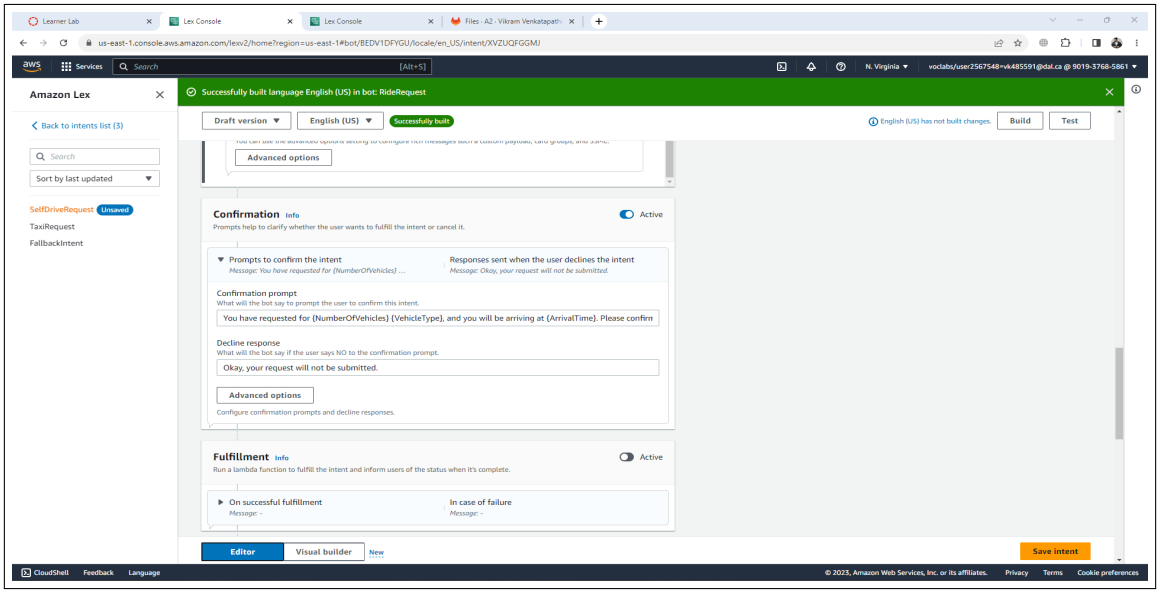

Figure 5.17.: ***SelfDriveRequest intent creation - Add Confirmation prompt with placeholder values***

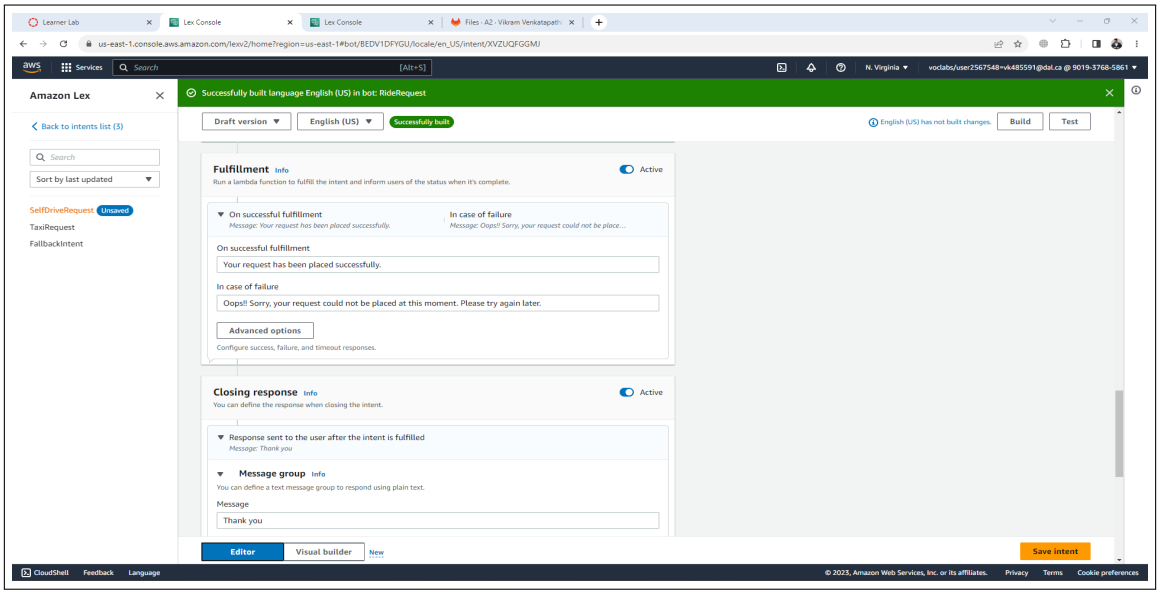

Figure 5.18.: ***SelfDriveRequest intent creation - Add Fullfilment prompt and Closing response***

5.4. Output

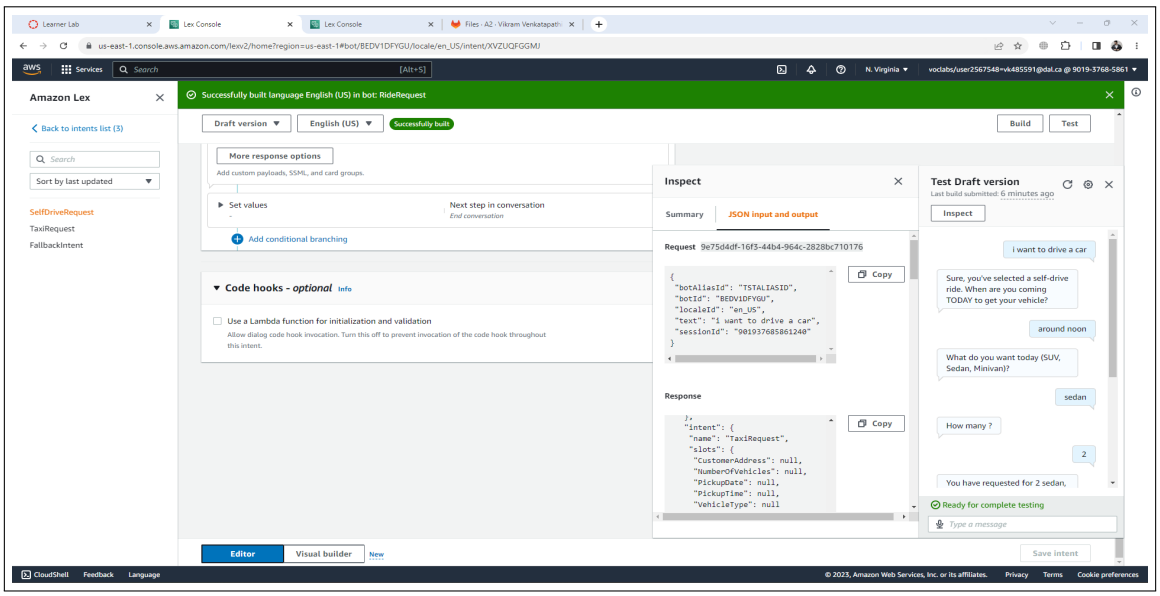

Figure 5.19.: ***SelfDrive - Confirm request 1.1***

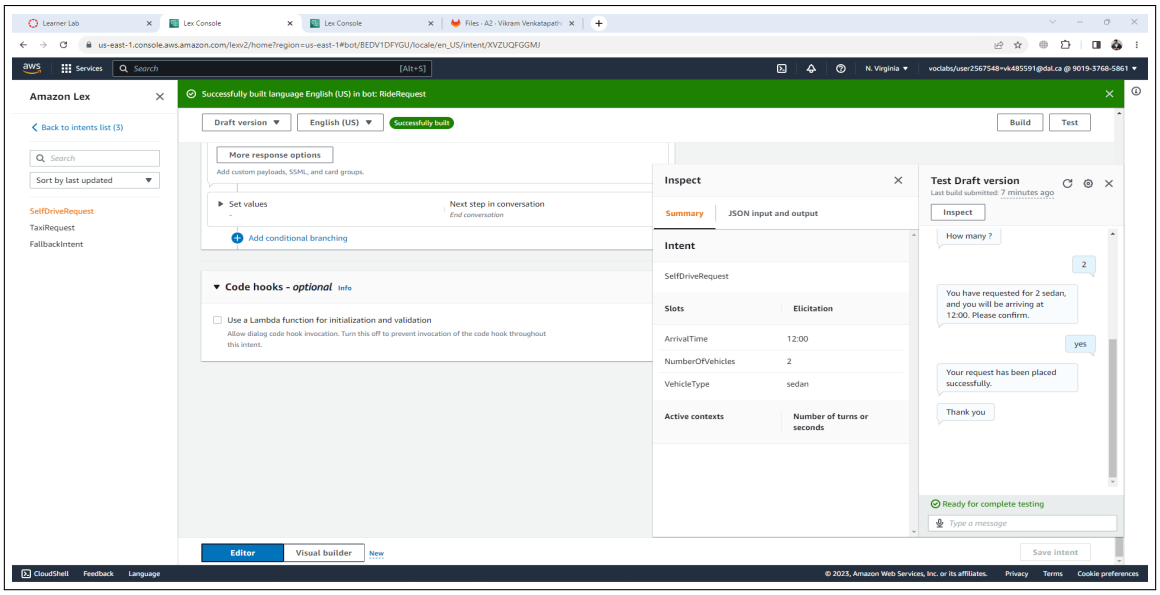

Figure 5.20.: ***SelfDrive - Confirm request 1.2***

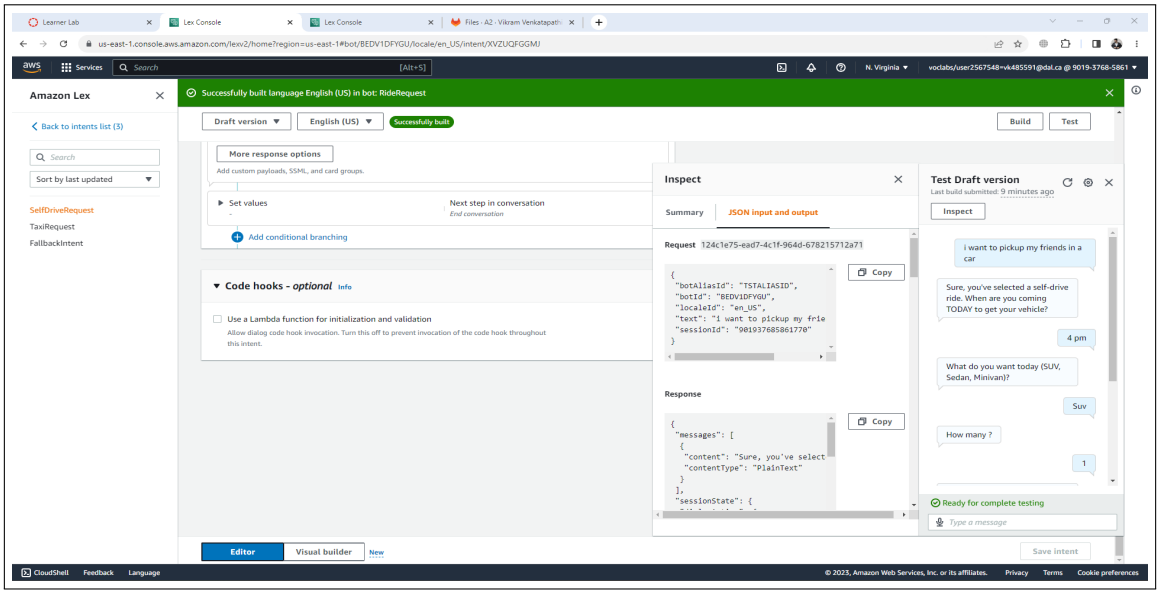

Figure 5.21.: ***SelfDrive - Cancel request 1.1***

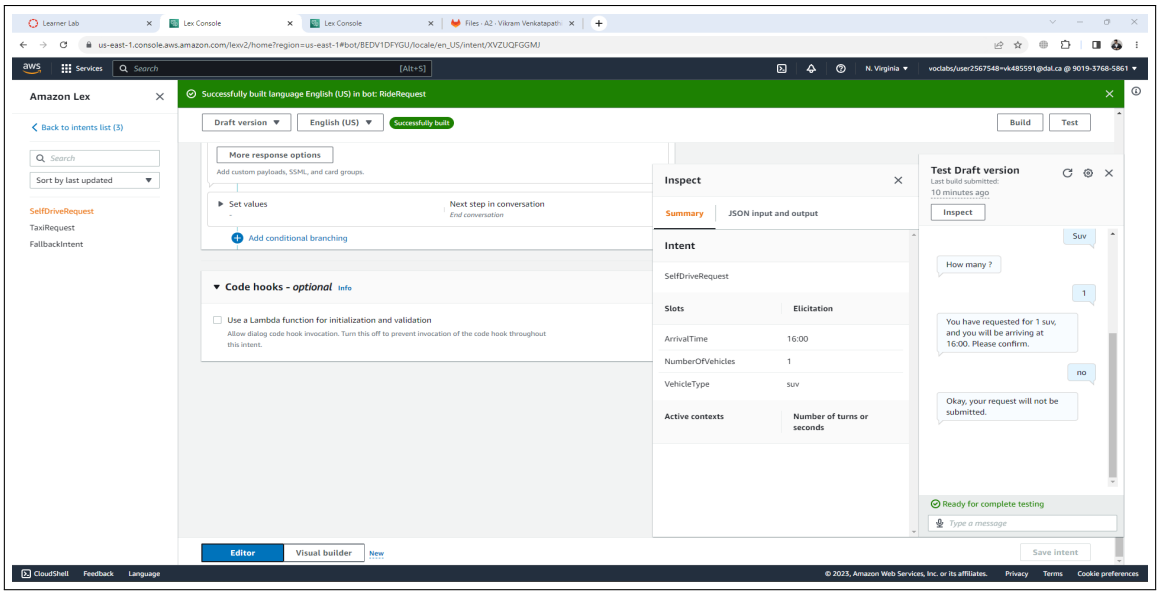

Figure 5.22.: ***SelfDrive - Cancel request 1.2***

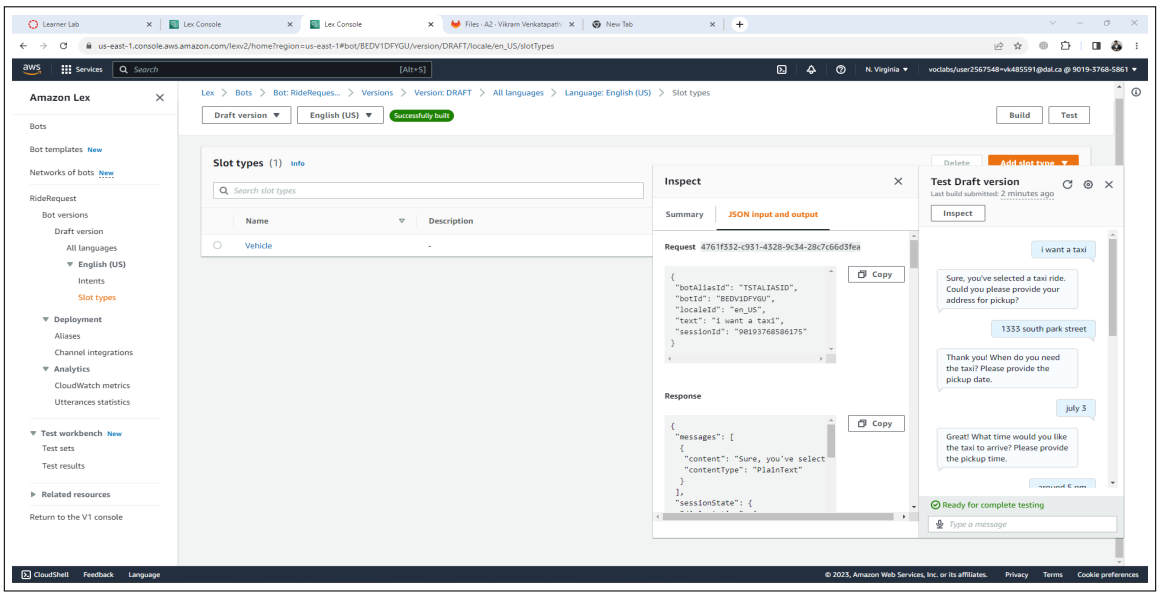

Figure 5.23.: ***Taxi - Confirm request 1.1***

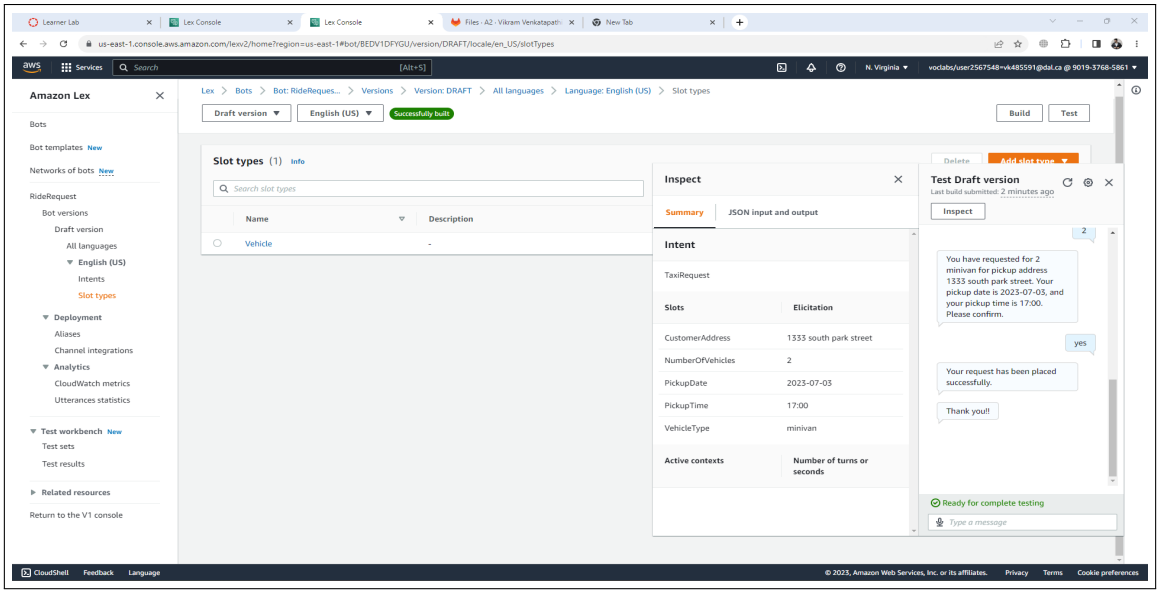

Figure 5.24.: ***Taxi - Confirm request 1.2***

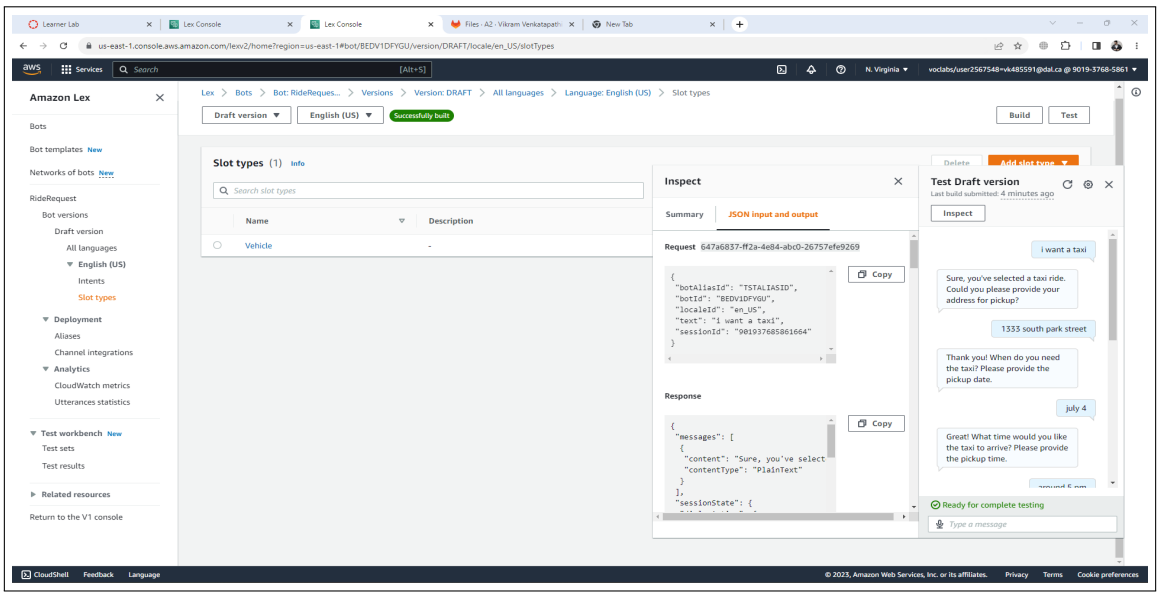

Figure 5.25.: ***Taxi - Cancel request 1.1***

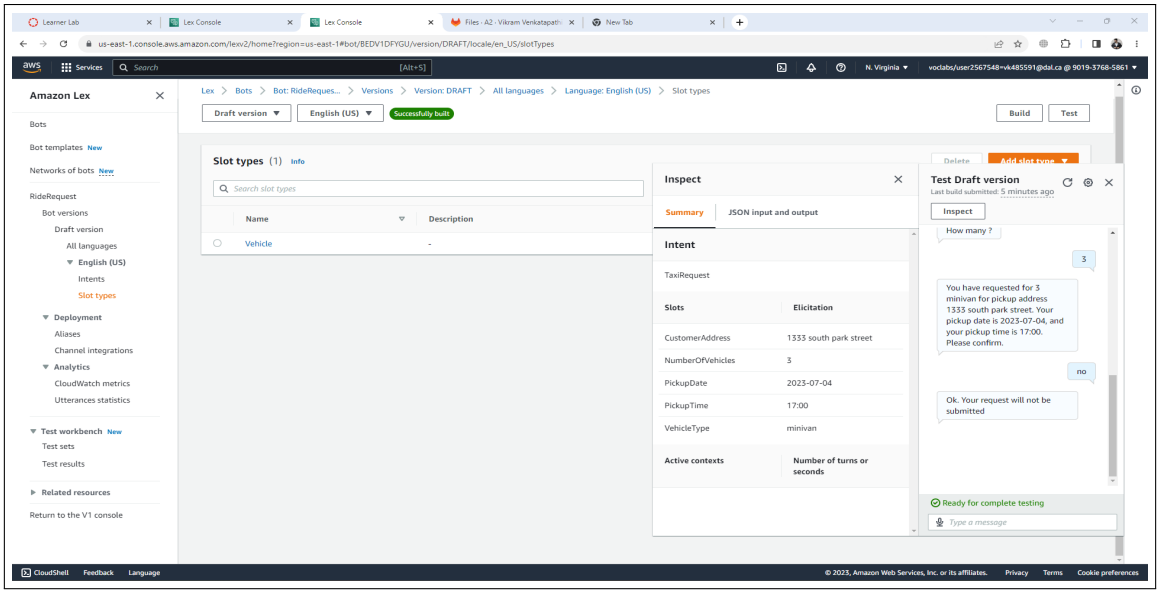

Figure 5.26.: ***Taxi - Cancel request 1.2***

Part IV.

REFERENCES

References

[1] N. Naik, "Performance Evaluation of Distributed Systems in Multiple Clouds using Docker Swarm," 2021 *IEEE International Systems Conference (SysCon)*, Vancouver, BC, Canada, 2021, pp. 1-6, doi: 10.1109/SysCon48628.2021.9447123

[2] "Create standard repositories," *Google Cloud*. [Online]. Available: https://cloud.google.com/artifact-registry/docs/repositories/create-repos.[Accessed: 29 June 2023].

[3] "Deploying to cloud run," *Google Cloud*. [Online]. Available: https://cloud.google.com/run/docs/deploying. [Accessed: 29 June 2023].

[4] "Add data to cloud firestore," *Firebase*. [Online]. Available: https://firebase.google.com/docs/firestore/manage-data/add-data. [Accessed: 29 June 2023].

[5] "Get data with cloud firestore," *Firebase*. [Online]. Available: https://firebase.google.com/docs/firestore/query-data/get-data. [Accessed: 29 June 2023].

[6] "Containerize an application," *Docker Documentation*, 28-Jun-2023. [Online]. Available: https://docs.docker.com/get-started/02_our_app/. [Accessed: 29 June 2023].

[7] "Update the application," *Docker Documentation*, 28-Jun-2023. [Online]. Available: https://docs.docker.com/get-started/03_updating_app/. [Accessed: 29 June 2023].

[8] "Push and pull images," *Google Cloud*. [Online]. Available: https://cloud.google.com/artifact-registry/docs/docker/pushing-and-pulling. [Accessed: 29 June 2023].

[9] "Create a new react app," *Reactjs.org*. [Online]. Available: https://legacy.reactjs.org/docs/create-a-new-react-app.html. [Accessed: 29 June 2023].

[10] S.Gandotra, "ReactJS router," *GeeksforGeeks*, 13-Dec-2019. [Online]. Available: https://www.geeksforgeeks.org/reactjs-router/. [Accessed: 29 June 2023].

[11] "LaTeX Listings package JSON formatting," *TeX Stack Exchange*, Available: https://tex.stackexchange.com/questions/560830/latex-listings-package-json-formatting. [Accessed 29 June 2023].

[12] V. Venkatapathi, "B00936916_VikramVenkatapathi_A1_Report.pdf," *Dalhousie University*, Jan. 2023. [Online]. Available: https://git.cs.dal.ca/vikramv/csci5408_w23_b00936916_vikram_-venkatapathi/-/blob/main/Assignment_1/B00936916_VikramVenkatapathi_A1_Report.pdf. [Accessed 29 June 2023].

[13] Shanmuganathan, Vishakan. "OOAD-Project." *GitHub*, 2021, https://github.com/svishakan/OOAD-Project. [Accessed 29 June 2023].

[14] AWS. "Amazon Lex V1",*Amazon.com*[Online]. Available: https://docs.aws.amazon.com/lex/latest/dg/gs-bp-create-bot.html. [Accessed: 29 June 2023].

[15] "Swarm mode overview," *Docker Documentation*, 30-Jun-2023. [Online]. Available: https://docs.docker.com/engine/swarm/. [Accessed: 03 July 2023].

[16] R. Powell, "Docker Swarm vs Kubernetes: how to choose a container orchestration tool," *CircleCI*, 12-Oct-2021. [Online]. Available: https://circleci.com/blog/docker-swarm-vs-kubernetes/. [Accessed: 03 July 2023].

[17] "Production-Grade Container Orchestration," *Kubernetes*. [Online]. Available: https://kubernetes.io/. [Accessed: 03 July 2023].

[18] "Lightweight kubernetes," *K3s.io*. [Online]. Available: https://k3s.io/. [Accessed: 03 July 2023].