

CSCI 5410 Serverless Data Processing

Assignment-2

PART-A REPORT

The paper “Performance Evaluation of Distributed Systems in Multiple Clouds using Docker Swarm” presents a performance evaluation of a Docker Swarm-based distributed system [1]. The study showcases the design and simulated development of the system, focusing on attributes such as high availability, fault tolerance, automatic scalability, load balancing, and maintainability of services [1]. While the evaluation was conducted on a small Docker Swarm cluster, future prospects include assessing the system's performance on larger clusters and exploring its deployment in multiple clouds to analyze communication and interoperability. The findings highlight the effectiveness of Docker Swarm as a natural choice for distributed systems and provide insights for further advancements in this area.

A Docker swarm cluster requires managers to manage resources and services, with one designated as the leader and the others as followers. Workers handle basic operations such as task execution and container-related data traffic. Managers oversee workers, schedule tasks, and perform periodic health checks. Swarm manager nodes use the Raft Consensus Algorithm to manage the swarm state [1]. The leader sends periodic heartbeats to other nodes, and in case of leader failure i.e., no communication over a period of time called as election timeout then new leader is elected in the presence of quorum of managers [1].

The authors simulated development of the distributed system using Docker Swarm involved creating five Docker machines within VirtualBox and configuring them to form a cluster. Inbuilt high availability and fault tolerances were tested by abruptly stopping the viable leader one by one, at point docker swam cluster could not continue its normal as there are less than 2 managers for successful failover procedure [1]. The fault tolerance of Docker Swarm is achieved through the Raft Consensus Algorithm, which allows the managers in the cluster to coordinate. By having $2f+1$ managers in the Docker Swarm cluster, the system can tolerate the failure of up to f managers [1]. This means that with an odd number of managers distributed across availability zones enhances fault-tolerance in the face of failures or maintenance scenarios [2].

The authors deployed two services (nginx and redis) in a Docker Swarm cluster and scaled them to five instances each [1]. Observing Docker Swarm's automatic load balancing, the replicas were efficiently allocated to nodes with optimal resources. Additionally, to test maintainability, the user abruptly stopped two nodes, and Docker Swarm automatically switched the instances running on those nodes to other available nodes. These experiments highlight Docker Swarm's capabilities in automatic scalability, load balancing, and maintaining services within a distributed system.

The user performed experiments comparing Docker Swarm and Google Kubernetes in terms of scalability performance on Cloud Container Cluster Common Benchmark framework by running 30,000 containers across 1,000 nodes [1]. The results showcased Docker Swarm's superior performance in container startup time and system responsiveness, particularly for small-scale workloads, while suggesting the usage of Google Kubernetes or other frameworks like Mesos for larger-scale deployments [1].

PART-B REPORT

GIT LAB Link - <https://git.cs.dal.ca/pusuluru/csci-5410-summer-23-b00913674.git>

1. Firestore has no collection(s) initially

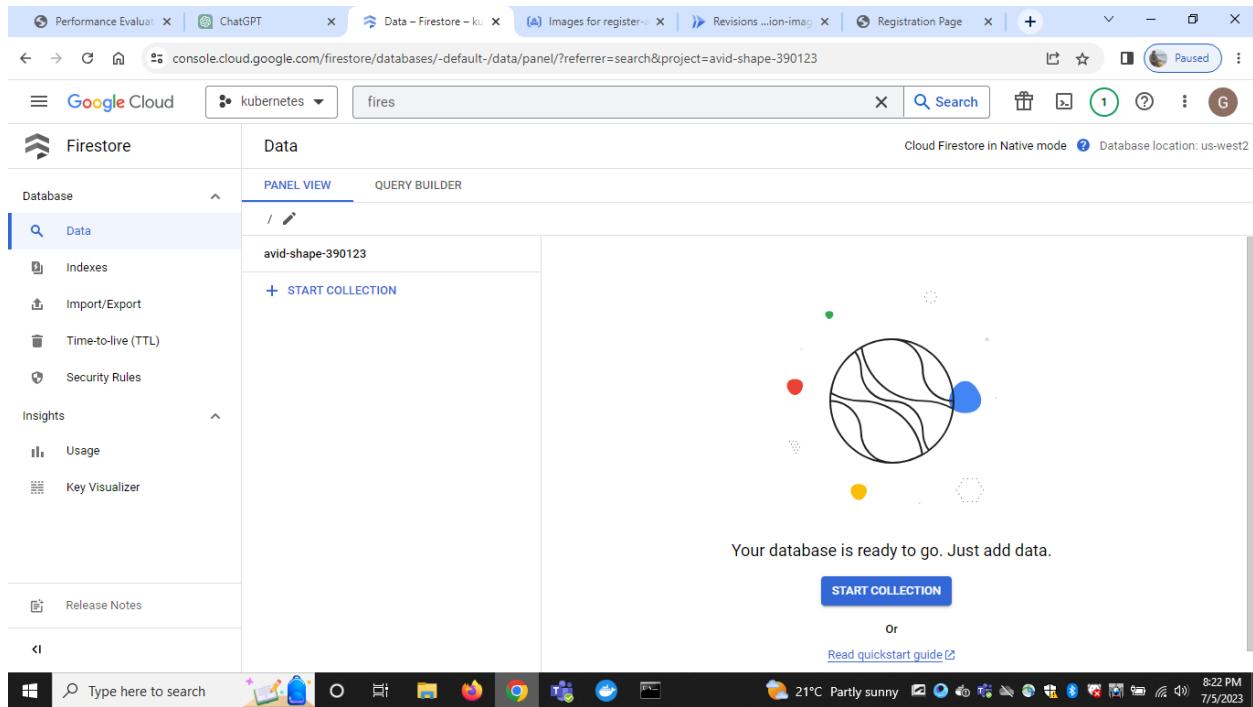


Figure 1: Firestore with no collections

2. Registration code docker image in artifact registry

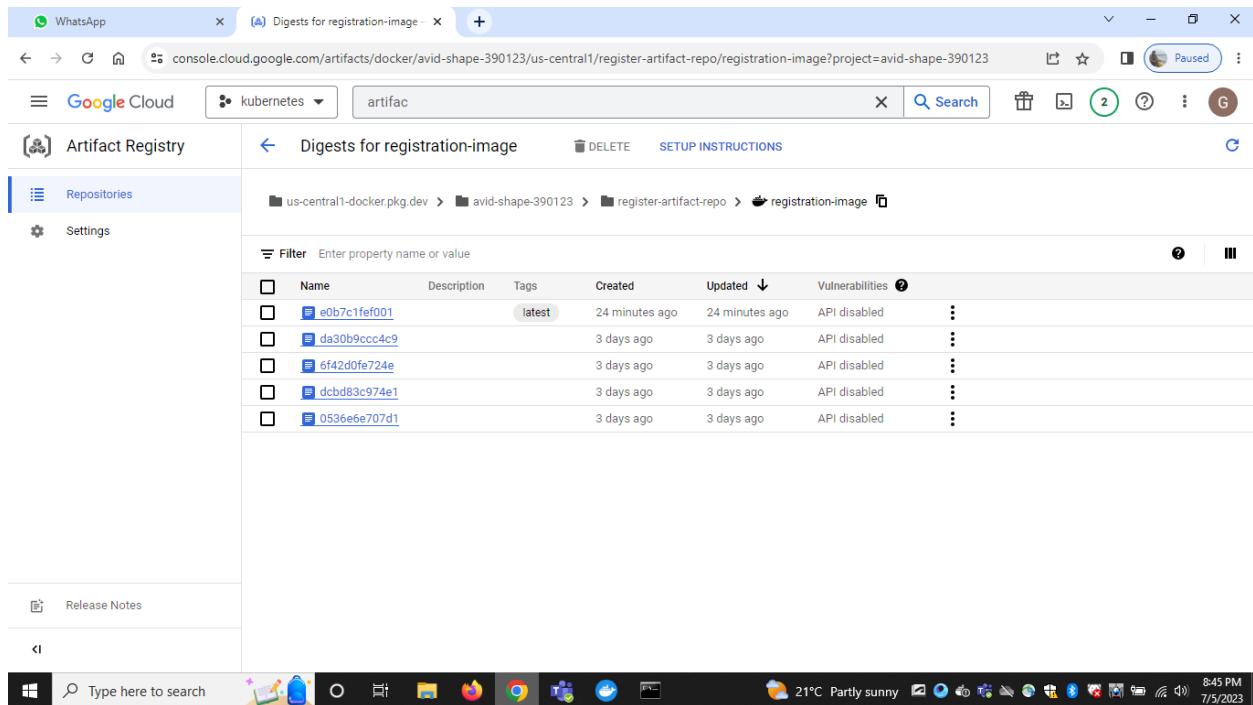


Figure 2: docker images of registration code under artifact registry

3. Login Code docker image in artifact registry

The screenshot shows the Google Cloud Artifact Registry interface. The left sidebar has 'Artifact Registry' selected under 'Repositories'. The main area shows the 'Digests for login-image' page. The breadcrumb navigation shows: us-central1-docker.pkg.dev > avid-shape-390123 > login-artifact-repo > login-image. A table lists five digest entries:

Name	Description	Tags	Created	Updated	Vulnerabilities
9c7da2a0b304		latest	29 minutes ago	29 minutes ago	API disabled
682738d2f9e2			3 days ago	3 days ago	API disabled
b4029efdf620f			3 days ago	3 days ago	API disabled
2c9b3c5320ae			3 days ago	3 days ago	API disabled

Below the table, there are sections for 'Release Notes' and a search bar. The taskbar at the bottom shows various icons and the system status.

Figure 3: docker images of login page code under artifact registry

4. Online available users' visibility code docker image in artifact registry

The screenshot shows the Google Cloud Artifact Registry interface, similar to Figure 3. The left sidebar has 'Artifact Registry' selected under 'Repositories'. The main area shows the 'Digests for home-image' page. The breadcrumb navigation shows: us-central1-docker.pkg.dev > avid-shape-390123 > home-artifact-repo > home-image. A table lists five digest entries:

Name	Description	Tags	Created	Updated	Vulnerabilities
d88ae47f5a21		latest	38 minutes ago	38 minutes ago	API disabled
8838c044d95e			43 minutes ago	43 minutes ago	API disabled
1258cd9f9601			2 days ago	2 days ago	API disabled
ef5646e7f78c			3 days ago	3 days ago	API disabled

Below the table, there are sections for 'Release Notes' and a search bar. The taskbar at the bottom shows various icons and the system status.

Figure 4: docker images of home page code under artifact registry

5. Registration Page after deploying docker image in the cloud run

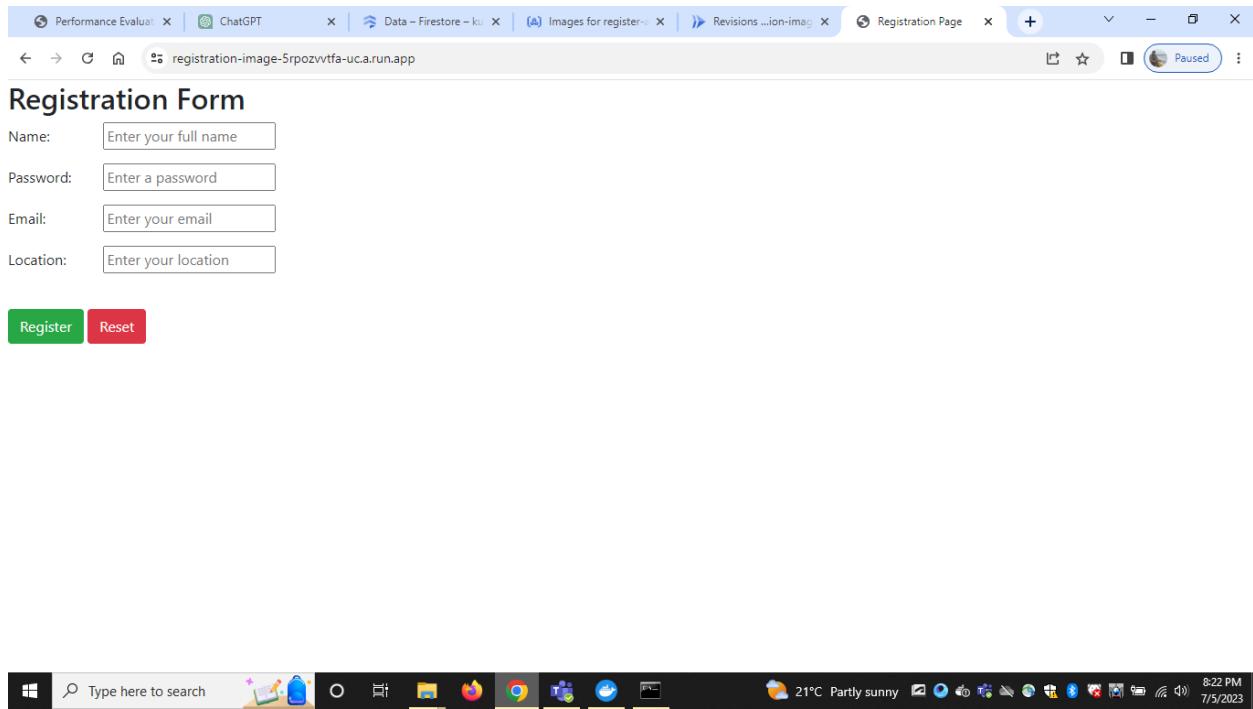


Figure 5: Registration form page

6. Registration Form Filled with details of user “Giri Sharan Reddy Pusuluru” and ready to submit

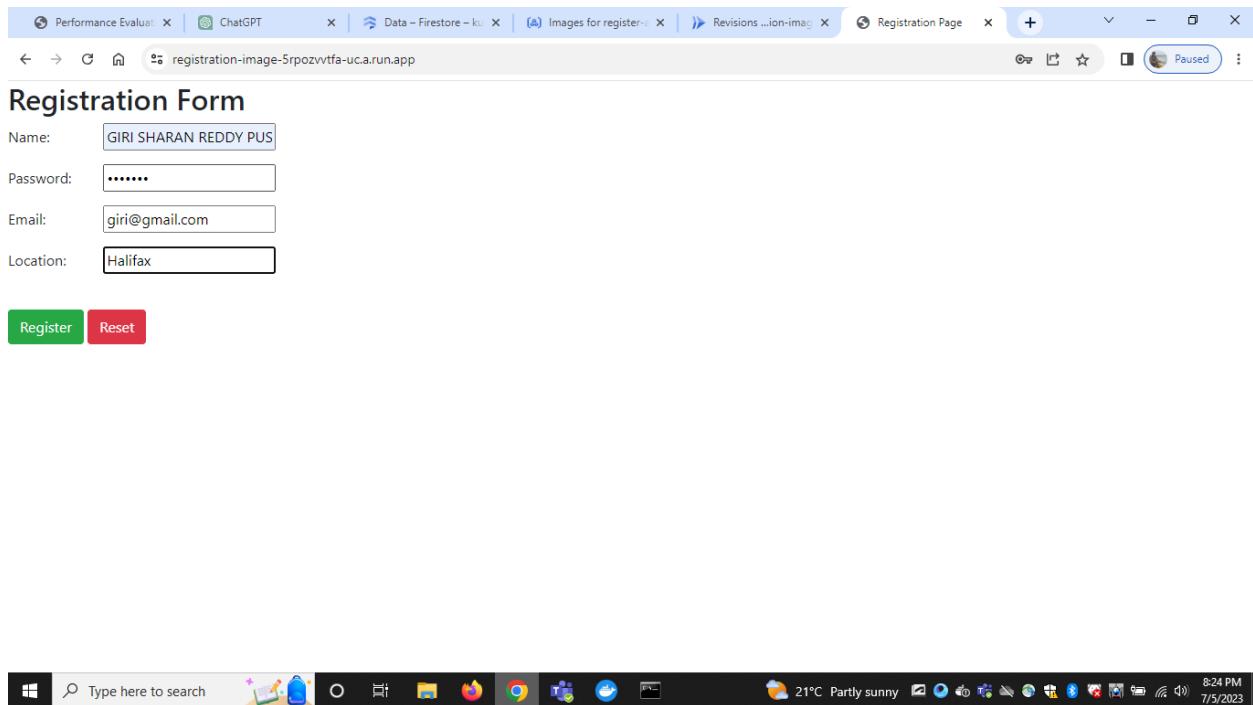


Figure 6: Details filled in registration form page

7. After submitting Registration Form, the details are getting stored in Firestore Database

The screenshot shows the Google Cloud Firestore interface. On the left, the sidebar includes 'Database' (selected), 'Indexes', 'Import/Export', 'Time-to-live (TTL)', 'Security Rules', 'Insights' (Usage, Key Visualizer), and 'Release Notes'. The main area displays a 'Data' view with 'PANEL VIEW' selected. The path is / > Reg > giri@gmail.com. The document 'giri@gmail.com' has fields: State, Full Name: "GIRI SHARAN REDDY PUSULURU", Location: "Halifax", and Password: "pass123".

Figure 7: Details updated in firestore collection Reg after user registration

As the user just got registers and did not login yet so the field “Logged_In” is offline, and also keeping track of Login and Logout Timestamp

The screenshot shows the Google Cloud Firestore interface. The sidebar is identical to Figure 7. The main area displays a 'Data' view with 'PANEL VIEW' selected. The path is / > State > giri@gmail.com. The document 'giri@gmail.com' has fields: State, Logged_In: "offline", Logged_In_Timestamp: null, Logged_Out: false, and Logged_Out_Timestamp: null. A note at the bottom right says 'Registration Timestamp: July 5, 2023 at 8:2...'.

Figure 8: Details updated in firestore collection State after user registration

8. Logs from cloud run of registration service

The screenshot shows the Google Cloud Console interface for a Cloud Run service named 'registration-image'. The 'Logs' tab is selected. The log entries are as follows:

Severity	Timestamp	Log Message
INFO	2023-07-05 20:22:37.322 ADT	[main] c.g.c.s.core.DefaultCredentialsProvider : Scopes in use by default credentials: [https://www.googleapis.com/...]
INFO	2023-07-05 20:22:37.546 ADT	[main] c.g.c.s.a.GcpContextAutoConfiguration : The default project ID is avid-shape-390123
INFO	2023-07-05 20:22:39.717 ADT	[main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 9001 (http) with context path ''
INFO	2023-07-05 20:22:39.730 ADT	[main] com.gcp.firebaseio.FirestoreApplication : Started FirestoreApplication in 9.493 seconds (process running for ...)
INFO	2023-07-05 20:22:39.740 ADT	Default STARTUP TCP probe succeeded after 1 attempt for container "registration-image-1" on port 9001.
INFO	2023-07-05 20:24:13.661 ADT	POST 302 700 B 2.1 s Chrome 114 https://registration-image-5rpozvvtfa-uc.a.run.app/register
INFO	2023-07-05 20:24:13.686 ADT	Project_Idavid-shape-390123
INFO	2023-07-05 20:24:13.712 ADT	Coming Here to update
WARN	2023-07-05 20:24:14.516 ADT	[mult-executor-0] i.g.n.s.i.n.u.internal.MacAddressUtil : Failed to find a usable hardware address from the network ...
INFO	2023-07-05 20:24:15.671 ADT	Successfully Registered on : 2023-07-05T23:24:15.551700002

No newer entries found matching current filter.

Figure 9: Logs in cloud run registration-image service

9. Once user is successfully registered, then will be redirected to Login page

The screenshot shows a browser window with the URL 'login-image-5rpozvvtfa-uc.a.run.app/login'. The page displays a simple login form with two input fields: 'Email:' and 'Password:', each with a placeholder 'Enter your email' and 'Enter your password' respectively. Below the inputs are two buttons: a green 'Login' button and a red 'Reset' button. The browser's taskbar at the bottom shows various pinned icons and the system status bar indicating '21°C Partly sunny' and the date '7/5/2023'.

Figure 10: Login Page

10. Filled login page details to check for authentication. If the authentication is failed then he will be in the same page else redirected to home page

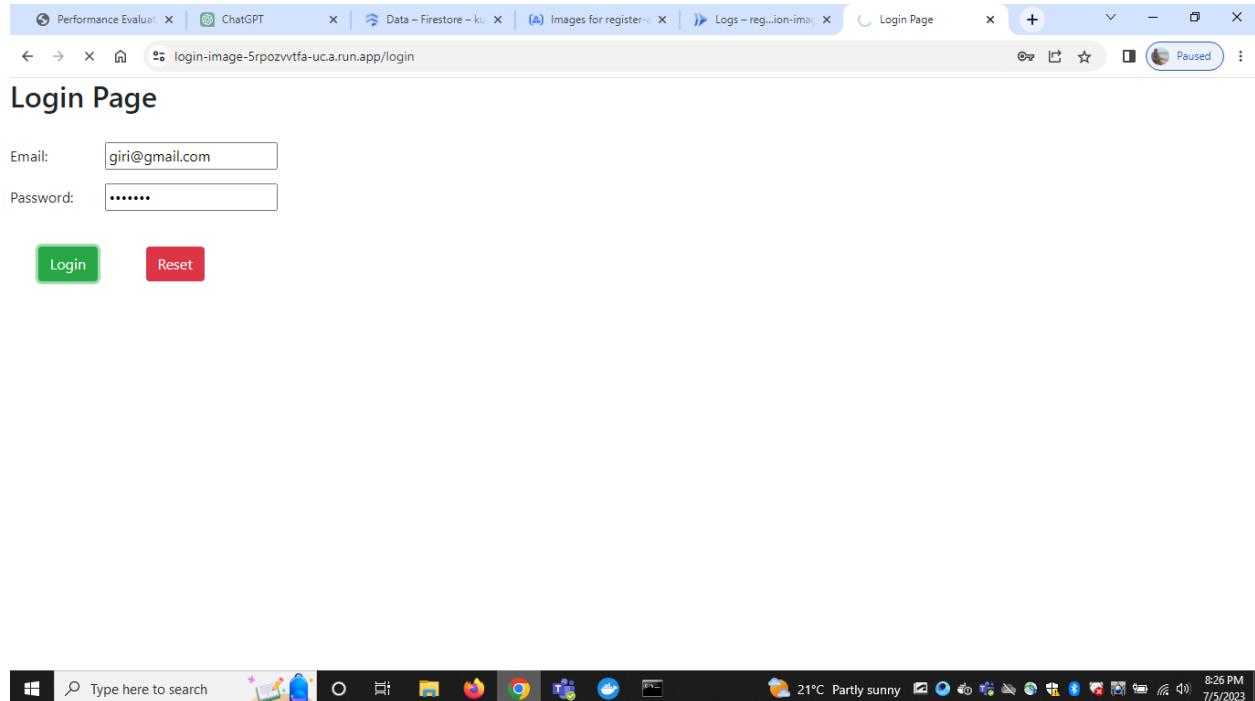


Figure 11: Details filled in login page for authentication

11. User Giri Sharan is successfully logged in and redirected to home page

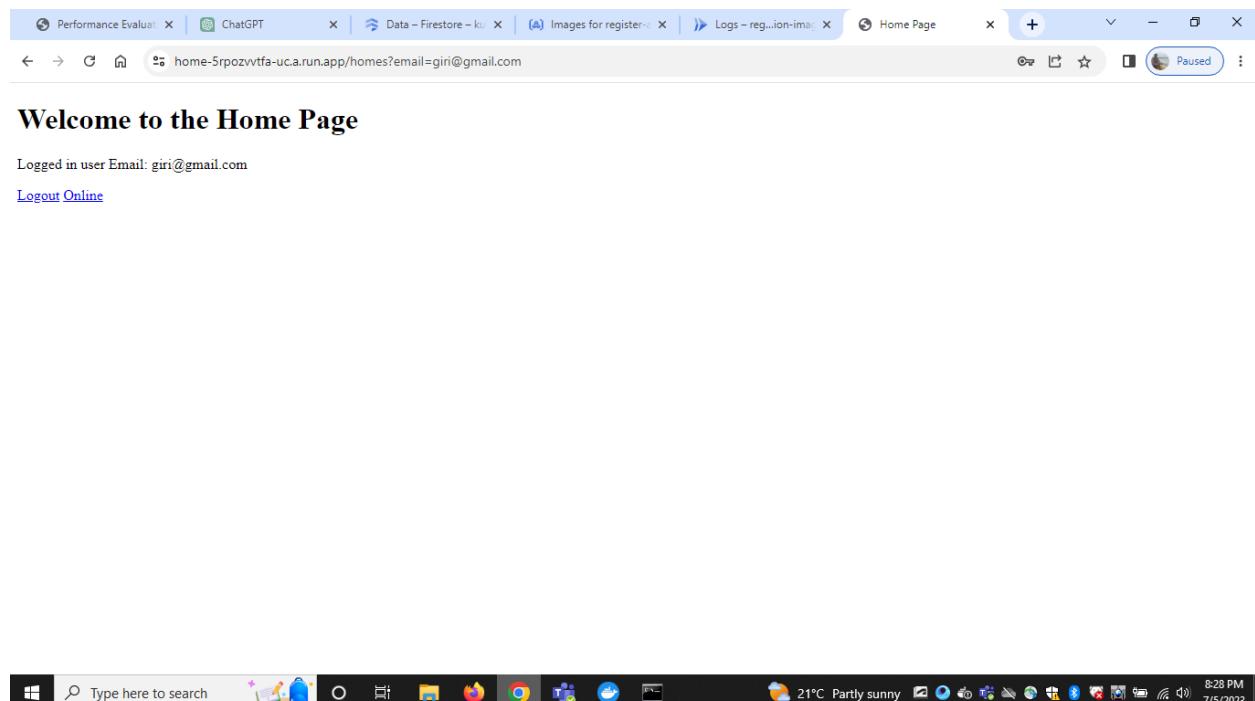


Figure 12: Home Page

12. Fields getting updated in Firestore Collection “State” of user specific document after user login

The screenshot shows the Google Cloud Firestore Data panel. On the left, the sidebar has 'Database' selected under 'Data'. The main area shows a 'PANEL VIEW' with a path: / > State > giri@gmail.com. A table lists a single document named 'giri@gmail.com' under the 'Reg' collection. The table columns are 'State' and 'giri@gmail.com'. To the right of the table, there are buttons for '+ START COLLECTION', '+ ADD DOCUMENT', '+ START COLLECTION', and '+ ADD FIELD'. Below the table, detailed log entries are visible:

- Logged_In: "online"
- Logged_In_Timestamp: July 5, 2023 at 8:26:38...
- Logged_Out: false
- Logged_Out_Timestamp: null
- Registration Timestamp: July 5, 2023 at 8:2...

The status bar at the bottom indicates it's 8:28 PM on 7/5/2023.

Figure 13: Fileds updated in State collection after user login

13. Registration Form Filled with details of user “Sankeerth” and then submitted

The screenshot shows a registration form titled 'Registration Form'. The form fields are filled with the following values:

- Name: sankeerth
- Password: (redacted)
- Email: sankeerthsharma@gmail.c
- Location: canada

At the bottom of the form are two buttons: 'Register' (green) and 'Reset' (red).

The status bar at the bottom indicates it's 8:31 on 05-07-2023.

Figure 14: Registration of a user

14. User Tupper getting registered

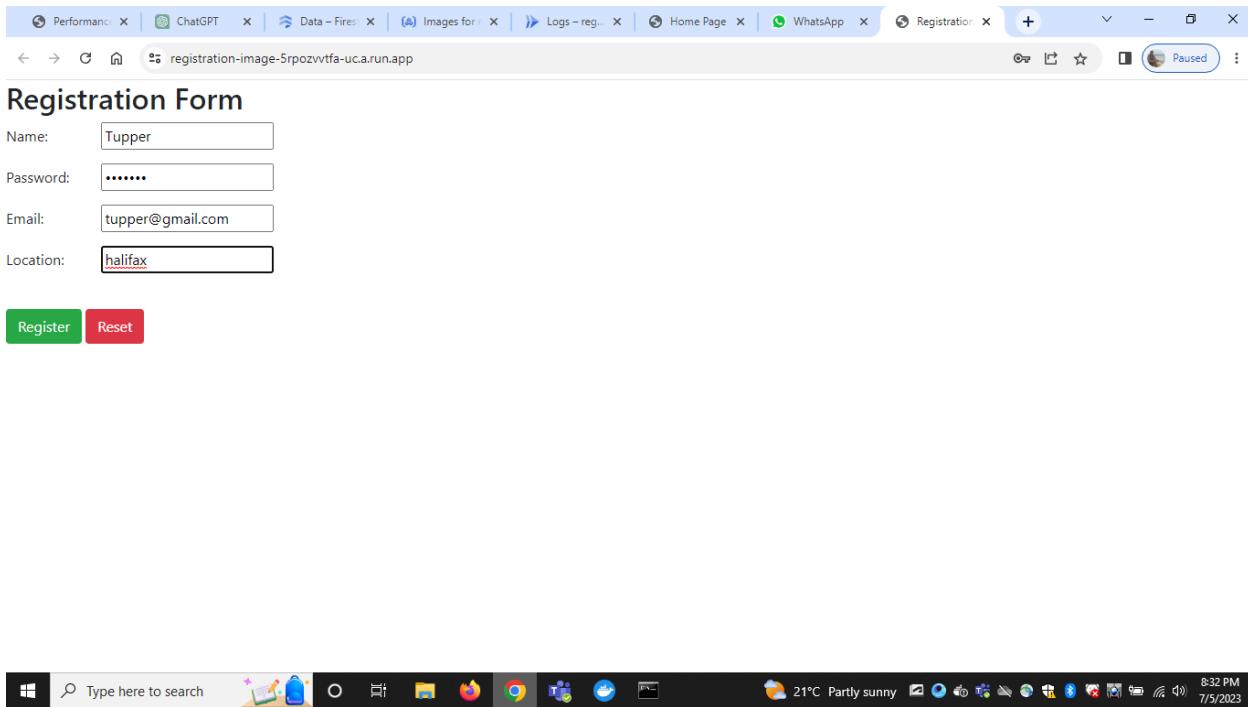


Figure 15: Registration of a user

15. Now user Sankeerth is logged in and clicking on button online in home page and able to see user Giri Sharan, who is also logged in but not Tupper because Tupper did not sign in

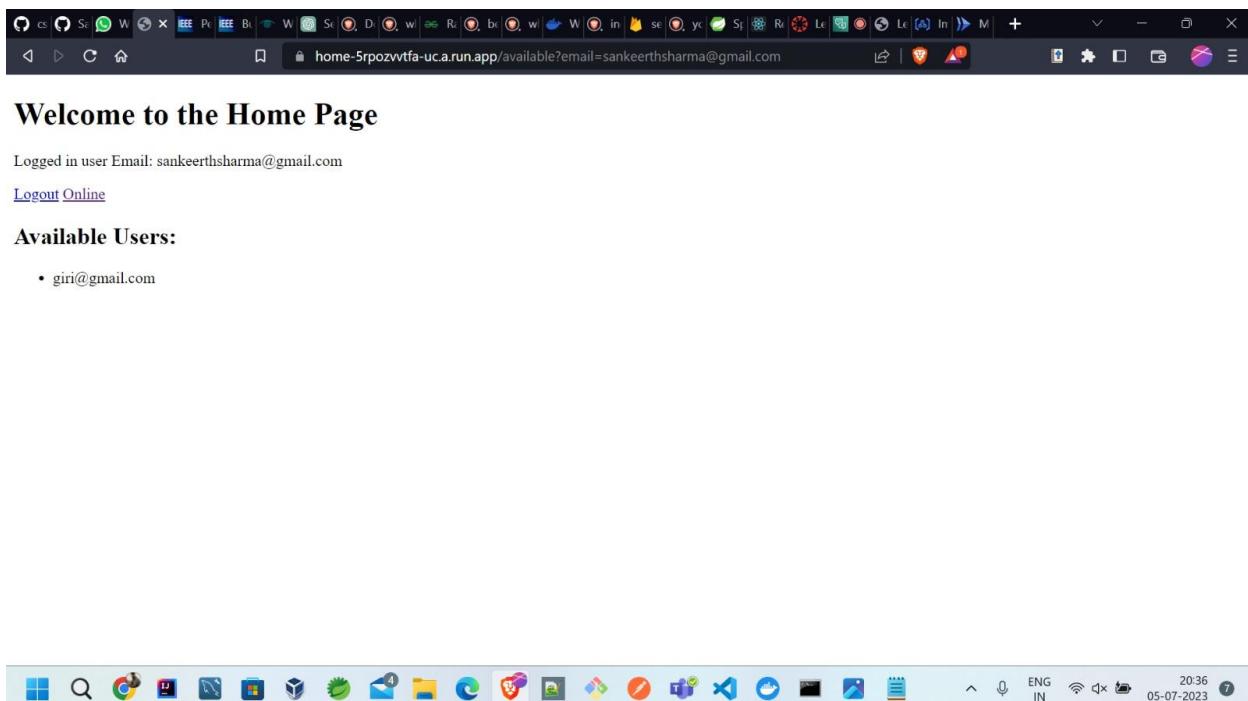


Figure 16: List of available users displayed in home page

16. User Sankeerth Field logged_in state got updated in Firestore collection of that user

The screenshot shows the Google Cloud Firestore interface. On the left, the sidebar has 'Database' selected under 'Data'. The main area shows a document structure under the collection 'State'. A new field 'Logged_In' was added with the value 'online'. Other fields visible include 'Logged_In_Timestamp' (July 5, 2023 at 8:31:56), 'Logged_Out' (false), and 'Registration Timestamp' (July 5, 2023 at 8:31:56).

Field	Type	Value
Logged_In	String	online
Logged_In_Timestamp	Timestamp	July 5, 2023 at 8:31:56...
Logged_Out	Boolean	false
Registration Timestamp	Timestamp	July 5, 2023 at 8:31:56...

Figure 17: user State after login

17. Checking in user Giri Sharan by clicking on online button provided in home page and now he is able to user Sankeerth as he signed in just now

The screenshot shows a web browser window displaying the home page. The URL is `home-5rpozvvtfa-uca.run.app/available?email=giri@gmail.com`. The page title is 'Welcome to the Home Page'. It displays a message 'Logged in user Email: giri@gmail.com' and a link 'Logout Online'. Below this, there is a section titled 'Available Users:' with a single item: 'sankeerthsharma@gmail.com'.

Figure 18: Displaying available users in home page

18. Reference that user Tupper is not logged in

The screenshot shows the Google Cloud Firestore interface. On the left, the sidebar includes sections for Database (Indexes, Import/Export, Time-to-live (TTL), Security Rules), Insights (Usage, Key Visualizer), and Release Notes. The main area displays a document under the 'State' collection for the user 'tupper@gmail.com'. The document contains the following fields:

Field	Type	Value
Logged_In	String	'offline'
Logged_In_Timestamp	Timestamp	null
Logged_Out	Boolean	false
Registration Timestamp	Timestamp	July 5, 2023 at 8:3...

Figure 19: Reference that user is not logged in

19. Logs from cloud run of Login service

The screenshot shows the Google Cloud Run logs for the 'login-image' service. The logs are displayed in a table format with columns for Severity, Timestamp, and Summary. The logs show several requests from different users (giri@gmail.com, sankeerthsharma@gmail.com) attempting to validate their user accounts.

Severity	Timestamp	Summary
> i	2023-07-05 20:26:34.467 ADT	POST 302 714 B 4.5 s Chrome 114 https://login-image-5rpozvvtfa-uc.a.run.app/validateUser
> *	2023-07-05 20:26:36.669 ADT	2023-07-05T23:26:36.669Z WARN 1 --- [ault-executor-0] i.g.n.s.i.n.u.internal.MacAddressUtil : Failed to find a usable hardware address from the network ...
> *	2023-07-05 20:26:38.342 ADT	giri@gmail.com---pass123
> *	2023-07-05 20:26:38.343 ADT	giri@gmail.com---pass123
> i	2023-07-05 20:31:30.899 ADT	GET 200 1.88 KB 13 ms Chrome 114 https://login-image-5rpozvvtfa-uc.a.run.app/login
> !	2023-07-05 20:31:31.492 ADT	GET 404 826 B 10 ms Chrome 114 https://login-image-5rpozvvtfa-uc.a.run.app/favicon.ico
> i	2023-07-05 20:31:55.535 ADT	POST 302 724 B 731 ms Chrome 114 https://login-image-5rpozvvtfa-uc.a.run.app/validateUser
> *	2023-07-05 20:31:55.810 ADT	giri@gmail.com---pass123
> *	2023-07-05 20:31:55.810 ADT	sankeerthsharma@gmail.com---sankeerth
> i	2023-07-05 20:32:44.014 ADT	GET 200 1.88 KB 7 ms Chrome 114 https://login-image-5rpozvvtfa-uc.a.run.app/login

Figure 20: Logs of cloud run login-image service

20. Logs from cloud run of home page service

The screenshot shows the Google Cloud Console interface for a Cloud Run service named 'home'. The service is deployed to the 'us-central1' region and has a URL of <https://home-5rpozvvtfa-uc.a.run.app>. The 'LOGS' tab is selected, displaying a list of log entries. The logs show various startup and user interaction events, such as TCP probe successes, application startup, and user logins.

Severity	Timestamp	Summary
> i	2023-07-05 20:27:09.933 ADT	Default STARTUP TCP probe succeeded after 1 attempt for container "home-image-1" on port 9201.
> *	2023-07-05 20:27:10.002 ADT	2023-07-05T23:27:10.001Z INFO 1 --- [main] c.googlefirestore.home.HomeApplication : Started HomeApplication in 7.401 seconds (process running for 9.486)
> i	2023-07-05 20:31:56.471 ADT	GET 200 1.04 KB 17 ms Chrome 114 https://home-5rpozvvtfa-uc.a.run.app/homes?email=sankeerthsharma@gmail.com
> !	2023-07-05 20:31:56.973 ADT	GET 404 827 B 14 ms Chrome 114 https://home-5rpozvvtfa-uc.a.run.app/favicon.ico
> i	2023-07-05 20:33:33.879 ADT	GET 200 1.13 KB 3 s Chrome 114 https://home-5rpozvvtfa-uc.a.run.app/available?email=giri@gmail.com
> *	2023-07-05 20:33:35.677 ADT	2023-07-05T23:33:35.677Z WARN 1 --- [ault-executor-0] i.g.n.s.i.n.u.internal.MacAddressUtil : Failed to find a usable hardware address from the network ...
> *	2023-07-05 20:33:36.872 ADT	giri@gmail.com---online
> *	2023-07-05 20:33:36.872 ADT	sankeerthsharma@gmail.com---online
> *	2023-07-05 20:33:36.872 ADT	Added User
> *	2023-07-05 20:33:36.872 ADT	tupper@gmail.com---offline

Figure 21: Logs of cloud run home service

21. Finally, the user can log out by clicking on button “logout” provided in home page

The screenshot shows a web browser window with the URL <https://home-5rpozvvtfa-uc.a.run.app/logout?email=giri@gmail.com>. The page displays a message: "Thank you for attending Serverless Data Processing Course class". This indicates that the user has successfully logged out of the service.

Figure 22: Display after user logout

22. Fields got updated after user gets signed out

The screenshot shows the Google Cloud Firestore interface. On the left, the sidebar includes 'Database' (selected), 'Indexes', 'Import/Export', 'Time-to-live (TTL)', and 'Security Rules'. Under 'Insights', there are 'Usage' and 'Key Visualizer' sections. The main area shows a 'Data' panel with a 'PANEL VIEW' tab selected. The path is / > State > giri@gmail.com. A table lists documents under the collection 'State'. One document, 'giri@gmail.com', has its fields expanded. The 'Logged_In' field is set to 'offline', and the 'Logged_In_Timestamp' is July 5, 2023 at 8:35:57. The 'Logged_Out' field is set to true, and the 'Logged_Out_Timestamp' is July 5, 2023 at 8:37:1. The 'Registration Timestamp' is also listed as July 5, 2023 at 8:2... . The bottom of the screen shows the Windows taskbar with various pinned icons.

Figure 23: Fileds getting updated after user logout

Google Cloud Run is a powerful serverless compute platform that allows developers for building and deploying applications quickly and easily, without the need to manage infrastructure or worry about scaling, pay for the exact resources our application consumes during its execution. Google Cloud Run is apart from other similar services because of its ability to run both stateless HTTP-driven containers and long-running background tasks, providing developers with great flexibility[3]. Additionally, Cloud Run offers auto-scaling capabilities, ensuring that our application scales seamlessly to handle any level of traffic. It integrates seamlessly with other Google Cloud services, such as Cloud Build for automated builds and deployments. Other notable features of Google Cloud Run include easy deployment through a simple command-line interface or via continuous integration and delivery pipelines [3]. I have used to Continuous deploy feature of cloud run where when I push my code to Github then it will automatically build docker image and stores that in artifact registry and then deploy in Cloud run.

Yaml file used to build the pipeline

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: firestore
  namespace: '213864788863'
```

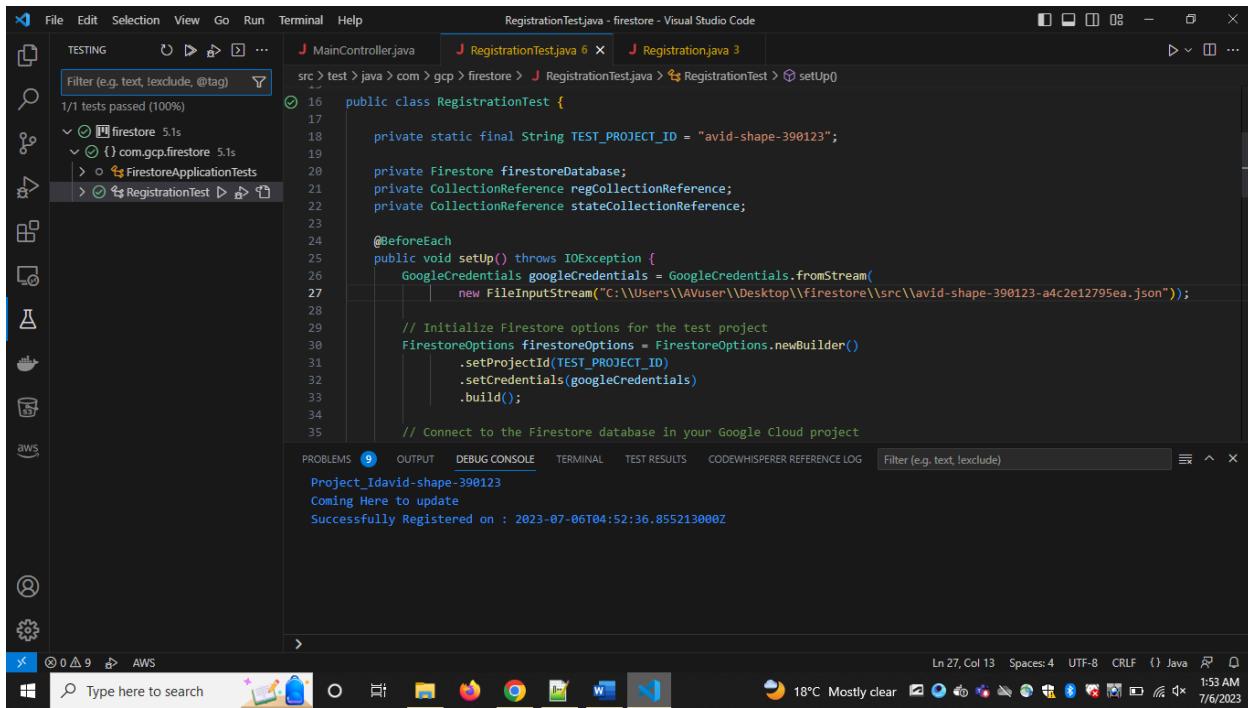
```
selfLink: /apis/serving.knative.dev/v1/namespaces/213864788863/services/firestore
  uid: a93118ce-f2c0-476b-832f-9ef64cf73f7c
  resourceVersion: AAX/yStraqU
  generation: 6
  creationTimestamp: '2023-07-06T03:08:04.335892Z'
  labels:
    managed-by: gcp-cloud-build-deploy-cloud-run
    gcb-trigger-id: c0151ee1-0fff-4371-8211-53f7d15033d8
    gcb-trigger-region: global
    commit-sha: 75d30bee54826e87f988870ee78e102b5d1aaba4
    gcb-build-id: e8fc1074-9300-4184-a55f-cc02997b1b8f
    cloud.googleapis.com/location: us-central1
  annotations:
    run.googleapis.com/client-name: gcloud
    serving.knative.dev/creator: giri.pusuluru@gmail.com
    serving.knative.dev/lastModifier: 213864788863@cloudbuild.gserviceaccount.com
    run.googleapis.com/client-version: 437.0.1
    run.googleapis.com/operation-id: d0e20354-e625-4e14-96e9-babc1909b59f
    run.googleapis.com/ingress: all
    run.googleapis.com/ingress-status: all
spec:
  template:
    metadata:
      name: firestore-00006-pex
    labels:
      managed-by: gcp-cloud-build-deploy-cloud-run
      gcb-trigger-id: c0151ee1-0fff-4371-8211-53f7d15033d8
      gcb-trigger-region: global
      commit-sha: 75d30bee54826e87f988870ee78e102b5d1aaba4
      gcb-build-id: e8fc1074-9300-4184-a55f-cc02997b1b8f
      run.googleapis.com/startupProbeType: Default
    annotations:
      run.googleapis.com/client-name: gcloud
      run.googleapis.com/client-version: 437.0.1
      autoscaling.knative.dev/maxScale: '100'
  spec:
    containerConcurrency: 80
    timeoutSeconds: 300
    serviceAccountName: 213864788863-compute@developer.gserviceaccount.com
    containers:
      - image: us-central1-docker.pkg.dev/avid-shape-390123/cloud-run-source-
deploy/firestore/firestore:75d30bee54826e87f988870ee78e102b5d1aaba4
    ports:
      - name: http1
```

```
        containerPort: 9001
      resources:
        limits:
          cpu: 1000m
          memory: 512Mi
      startupProbe:
        timeoutSeconds: 240
        periodSeconds: 240
        failureThreshold: 1
        tcpSocket:
          port: 9001
    traffic:
      - percent: 100
        latestRevision: true
  status:
    observedGeneration: 6
    conditions:
      - type: Ready
        status: 'True'
        lastTransitionTime: '2023-07-06T03:32:45.427869Z'
      - type: ConfigurationsReady
        status: 'True'
        lastTransitionTime: '2023-07-06T03:32:21.709797Z'
      - type: RoutesReady
        status: 'True'
        lastTransitionTime: '2023-07-06T03:32:45.522597Z'
    latestReadyRevisionName: firestore-00006-pex
    latestCreatedRevisionName: firestore-00006-pex
  traffic:
    - revisionName: firestore-00006-pex
      percent: 100
      latestRevision: true
  url: https://firestore-5rpozvvtfa-uc.a.run.app
  address:
    url: https://firestore-5rpozvvtfa-uc.a.run.app
```

Artifact Registry is a tool provided by Google Cloud designed specifically for securely storing and managing software artifacts like container images and packages. It integrates with popular build and deployment tools, allowing easy building and deploying of applications. It offers fine-grained advanced features for artifact management, such as access controls, versioning, and tagging, and integrates seamlessly with Google Cloud services for easy deployment within a cloud infrastructure [4].

Docker Containers provide a lightweight and portable environment that encapsulates all the dependencies and components required to run an application, making it consistent and reliable across different environments. One key advantage of Docker containers is their isolation, ensuring that each container operates independently, preventing conflicts and providing security. Containers are highly scalable, allowing applications to be seamlessly scaled up or down based on demand [4]. They promote efficient resource utilization by running multiple containers on a single host, maximizing hardware efficiency. With Docker, applications become easily portable, as containers can be run on any system with Docker installed, irrespective of the underlying infrastructure [4].

Test case of registration form code:



```

File Edit Selection View Go Run Terminal Help RegistrationTest.java - firestore - Visual Studio Code
TESTING Filter (e.g. text, exclude, @tag) ...
src > test > java > com > gcp > firestore > J RegistrationTest.java 6 | J Registration.java 3
src > test > java > com > gcp > firestore > J RegistrationTest.java > RegistrationTest > setUp()
16 public class RegistrationTest {
17
18     private static final String TEST_PROJECT_ID = "avid-shape-390123";
19
20     private Firestore firestoreDatabase;
21     private CollectionReference regCollectionReference;
22     private CollectionReference stateCollectionReference;
23
24     @BeforeEach
25     public void setUp() throws IOException {
26         GoogleCredentials googleCredentials = GoogleCredentials.fromStream(
27             new FileInputStream("C:\\Users\\AVuser\\Desktop\\firestore\\src\\avid-shape-390123-a4c2e12795ea.json"));
28
29         // Initialize Firestore options for the test project
30         FirestoreOptions firestoreOptions = FirestoreOptions.newBuilder()
31             .setprojectId(TEST_PROJECT_ID)
32             .setCredentials(googleCredentials)
33             .build();
34
35         // Connect to the Firestore database in your Google Cloud project
}

```

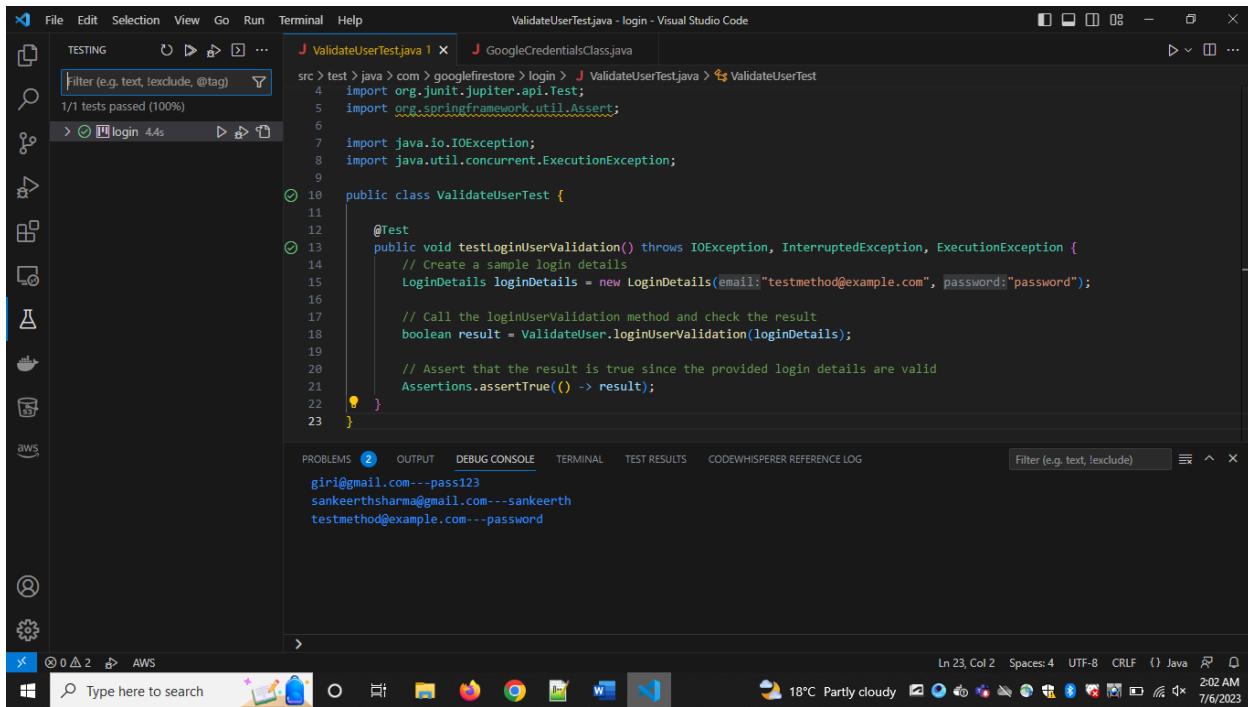
PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL TEST RESULTS CODEWHISPERER REFERENCE LOG Filter (e.g. text, exclude)

Project_Idavid-shape-390123
Coming Here to update
Successfully Registered on : 2023-07-06T04:52:36.855Z13000Z

Type here to search 18°C Mostly clear 1:53 AM 7/6/2023

Figure 24: Test case for registration of a user

Test case of login page validation code:



The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "TESTING" containing "ValidateUserTest.java" and "GoogleCredentialsClass.java".
- Code Editor:** Displays the content of `ValidateUserTest.java`:


```

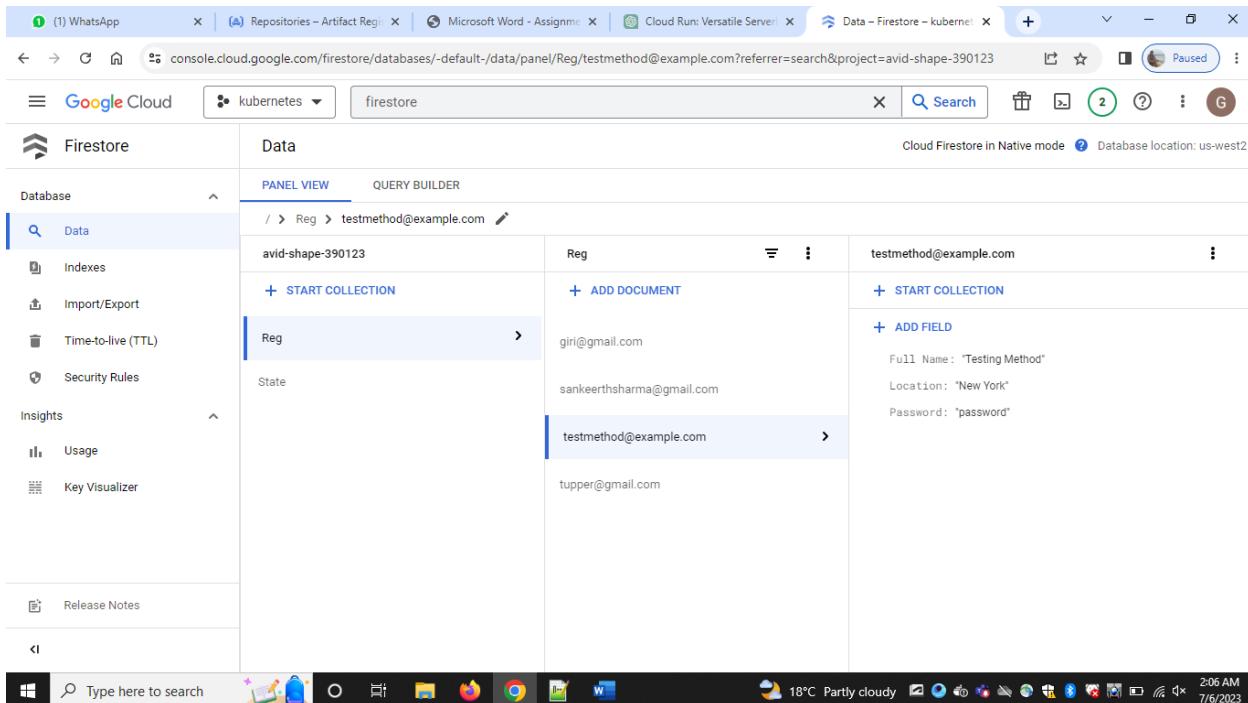
src > test > java > com > googlefirestore > login > J ValidateUserTest.java > ValidateUserTest
1/1 tests passed (100%)
> O login 44s
```

```

10 public class ValidateUserTest {
11
12     @Test
13     public void testLoginUserValidation() throws IOException, InterruptedException, ExecutionException {
14         // Create a sample login details
15         LoginDetails loginDetails = new LoginDetails(email:"testmethod@example.com", password:"password");
16
17         // Call the loginUserValidation method and check the result
18         boolean result = ValidateUser.loginUserValidation(loginDetails);
19
20         // Assert that the result is true since the provided login details are valid
21         Assertions.assertTrue(() -> result);
22     }
23 }
```
- Terminal:** Shows the command `giri@gmail.com--pass123`.
- Output:** Shows the output of the test execution.
- Problems:** Shows 2 errors.
- Debug Console:** Shows the command `sankeerthsharma@gmail.com---sankeerth`.
- Test Results:** Shows 1/1 tests passed (100%).
- CodeWhisperer Reference Log:** Shows the command `testmethod@example.com---password`.
- Bottom Bar:** Includes icons for AWS, a search bar, and system status.

Figure 25: Test case for login of a user

Reference that user is created from test case of registration code



The screenshot shows the Google Cloud Firestore console with the following details:

- Left Sidebar:** Shows sections for Database, Indexes, Import/Export, Time-to-live (TTL), and Security Rules.
- Panel View:** Shows a collection named "Reg" under the database "avid-shape-390123". It contains documents for "giri@gmail.com", "sankeerthsharma@gmail.com", "testmethod@example.com", and "tupper@gmail.com".
- Right Panel:** Shows the document "testmethod@example.com" with fields:
 - Full Name: "Testing Method"
 - Location: "New York"
 - Password: "password"
- Bottom Bar:** Includes icons for AWS, a search bar, and system status.

Figure 26: user details in firestore

Reference that user login details are validated successfully and the status of user is changed

The screenshot shows the Google Cloud Firestore console interface. On the left, there's a sidebar with 'Firestore' selected under 'Database'. The main area is titled 'Data' and shows a hierarchical view of collections and documents. Under the 'State' collection, there are three documents: 'giri@gmail.com', 'sankeerthsharma@gmail.com', and 'testmethod@example.com'. The 'testmethod@example.com' document has the following fields:

Field	Type	Value
Logged_In	(string)	"online"
Logged_In_Timestamp		July 6, 2023 at 2:01:57...
Logged_Out		false
Logged_Out_Timestamp		null
Registration Timestamp		July 6, 2023 at 1:5...

Test case of checking available online users

The screenshot shows a Java test file named 'CheckAvailabilityTest.java' open in Visual Studio Code. The code is as follows:

```
public class CheckAvailabilityTest {
    @Test
    public void testOnlineCheck() throws IOException, InterruptedException, ExecutionException {
        // Define the user email for which availability needs to be checked
        String userEmail = "test@example.com";

        // Call the onlineCheck method to check the availability of online users
        List<String> availableUsers = CheckAvailability.onlineCheck(userEmail);

        System.out.println(availableUsers.toString());

        // Assert that the availableUsers list is not null
        assertNotNull(availableUsers);

        // Assert that the availableUsers list does not contain the user email being checked
        assertFalse(availableUsers.contains(userEmail));
    }
}
```

The 'PROBLEMS' tab shows the following output:

```
giri@gmail.com---offline
sankeerthsharma@gmail.com---offline
testmethod@example.com---online
Added User
tupper@gmail.com---offline
[ testmethod@example.com ]
```

Figure 27: Test case for checking available online users

PART-C REPORT

Ride Request chat bot is built where it takes fulfills two types of requests

- 1) Taxi service booking
- 2) Self-driving car booking

Basic Terminology

Intent – An intent represents an action that the user wants to perform like booking a cab.

Utterances - These are the textual representations of what a user types or says in order to request a service. One intent can contain many different utterances, allowing users to trigger the bot using different phrases.

Slot - A slot type is a list of value(s) that Amazon Lex takes as input and are used to validate, compare or fulfill request.

Initial Greeting

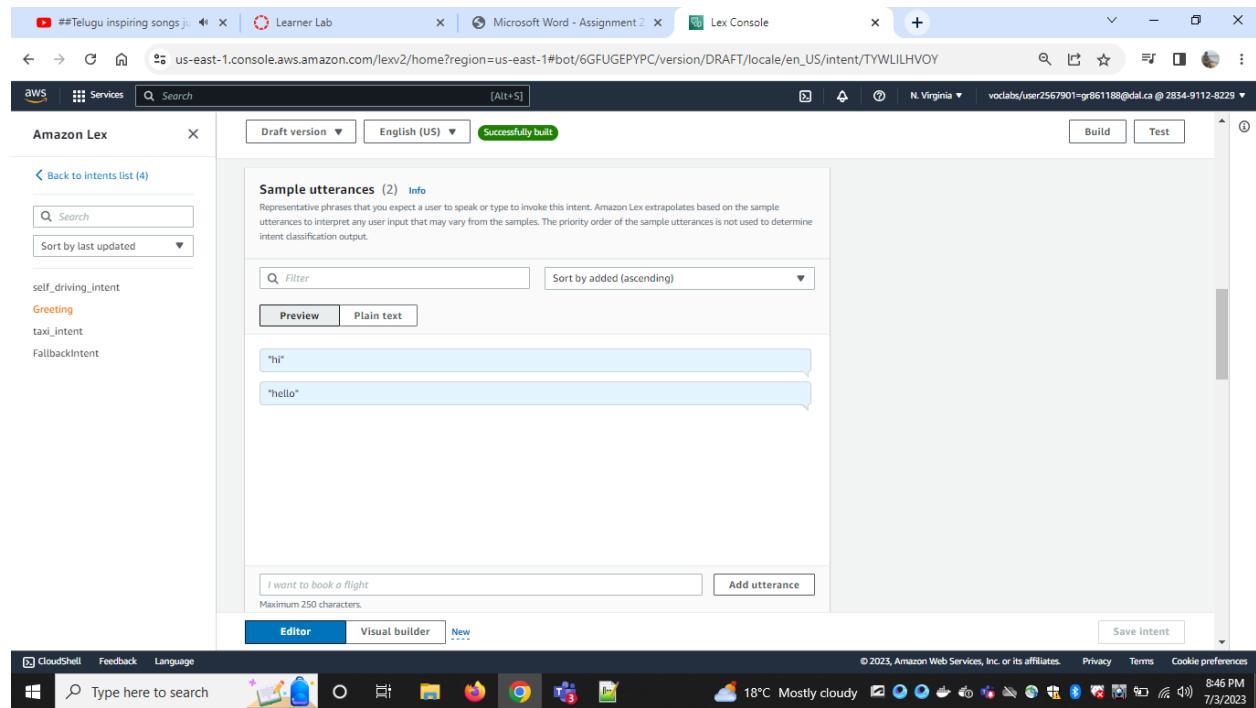


Figure 28: Greeting utterances

Reply for greeting utterances

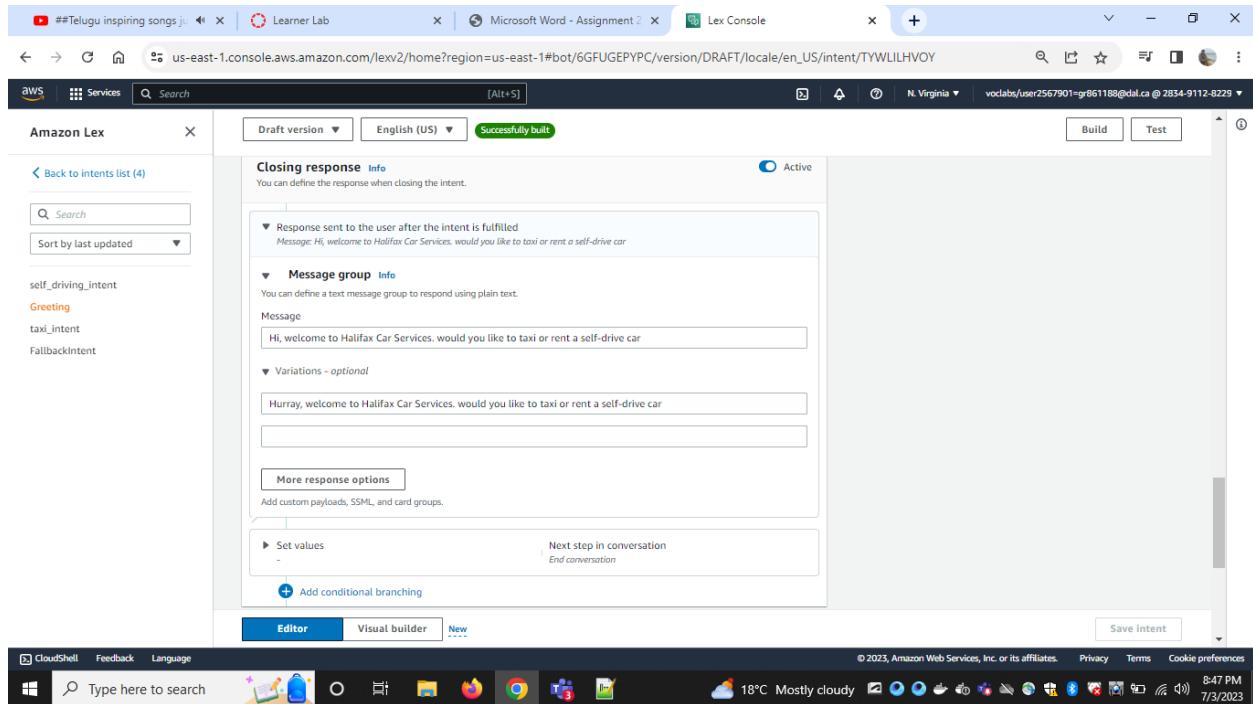


Figure 29: Reply for greeting utterances

Steps Involved in Building Taxi Service Bot

1. Sample Utterances for accepting the taxi/cab request

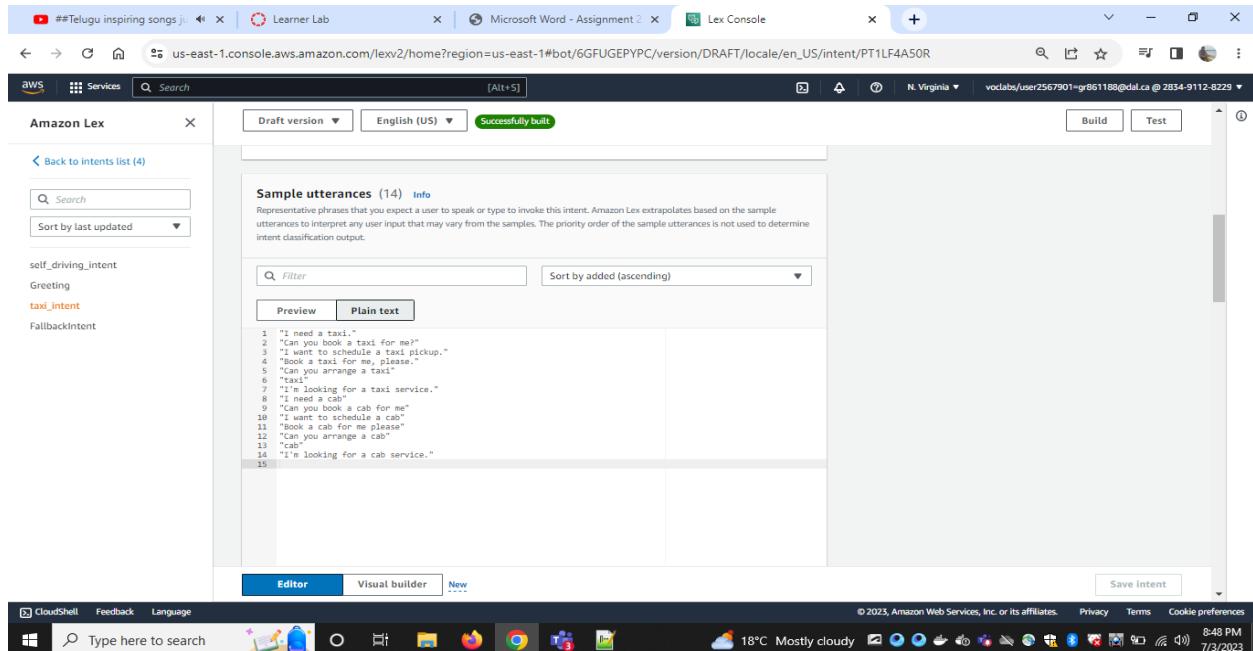


Figure 30: Sample utterances

2. Initial response from bot once user has requested for taxi service

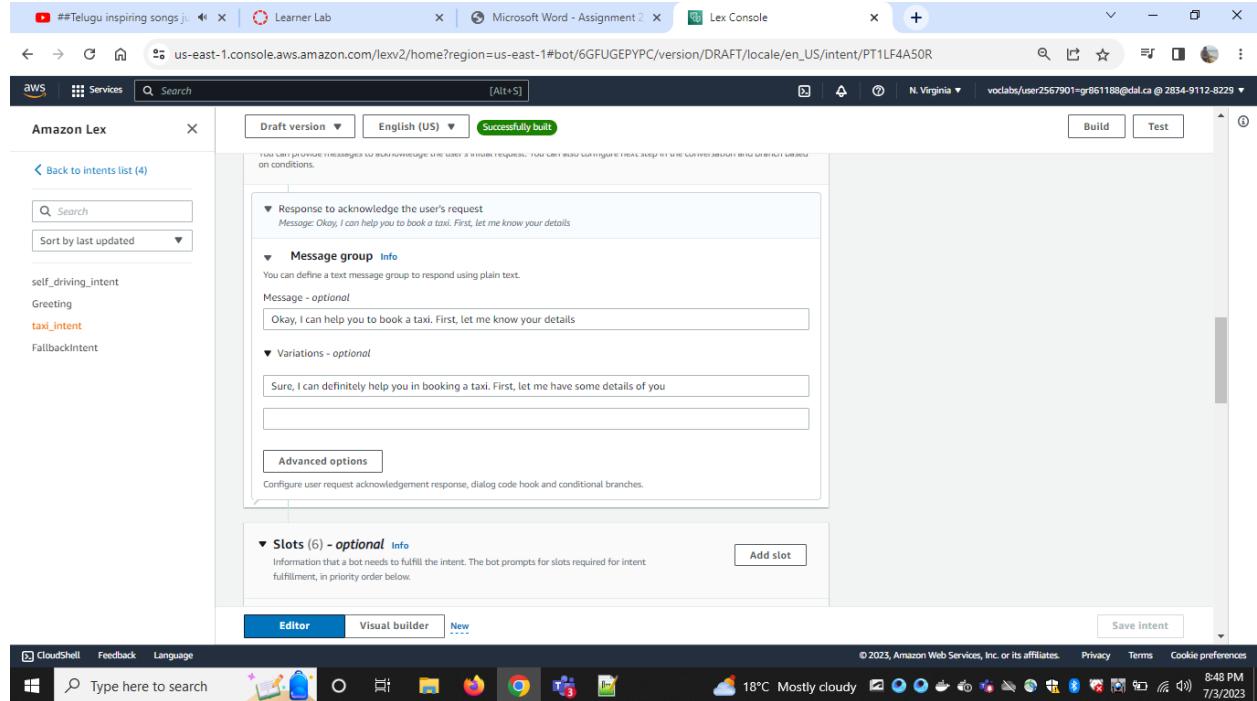


Figure 31: Initial response from bot

3. Inputs that are required in order to book a taxi

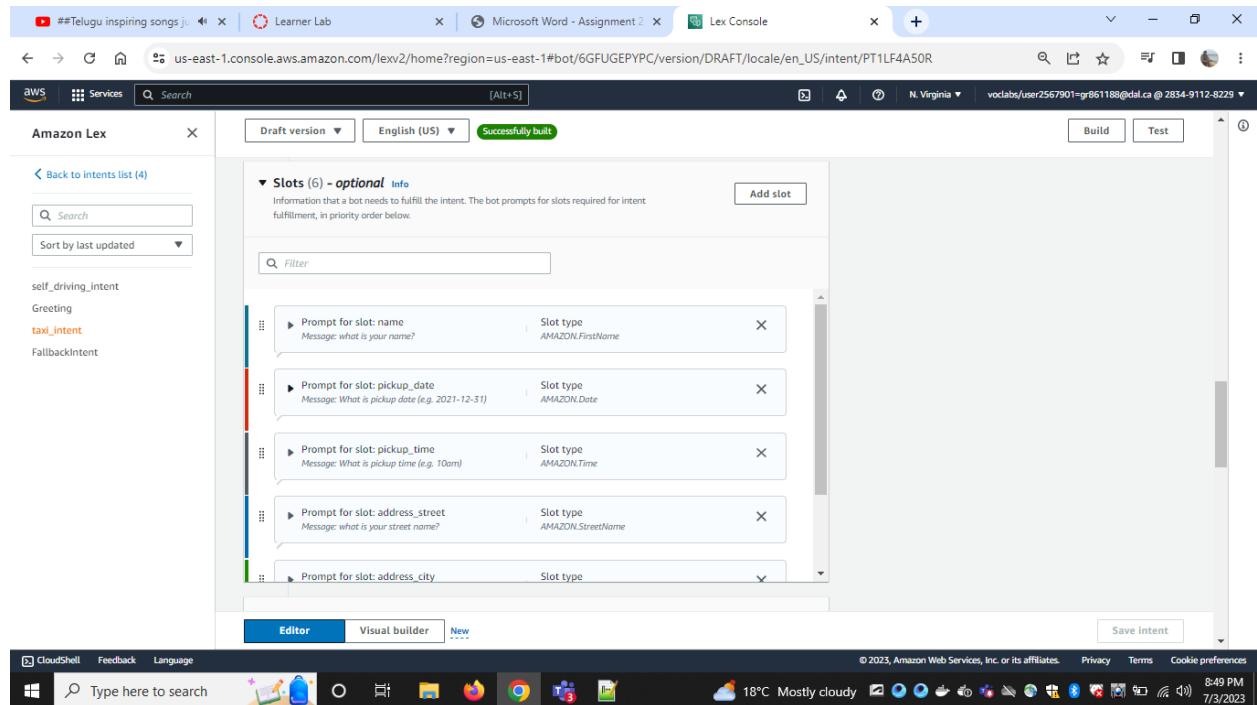


Figure 32: Slots for booking taxi

4. Once user provided all the slots, bot asks for confirmation

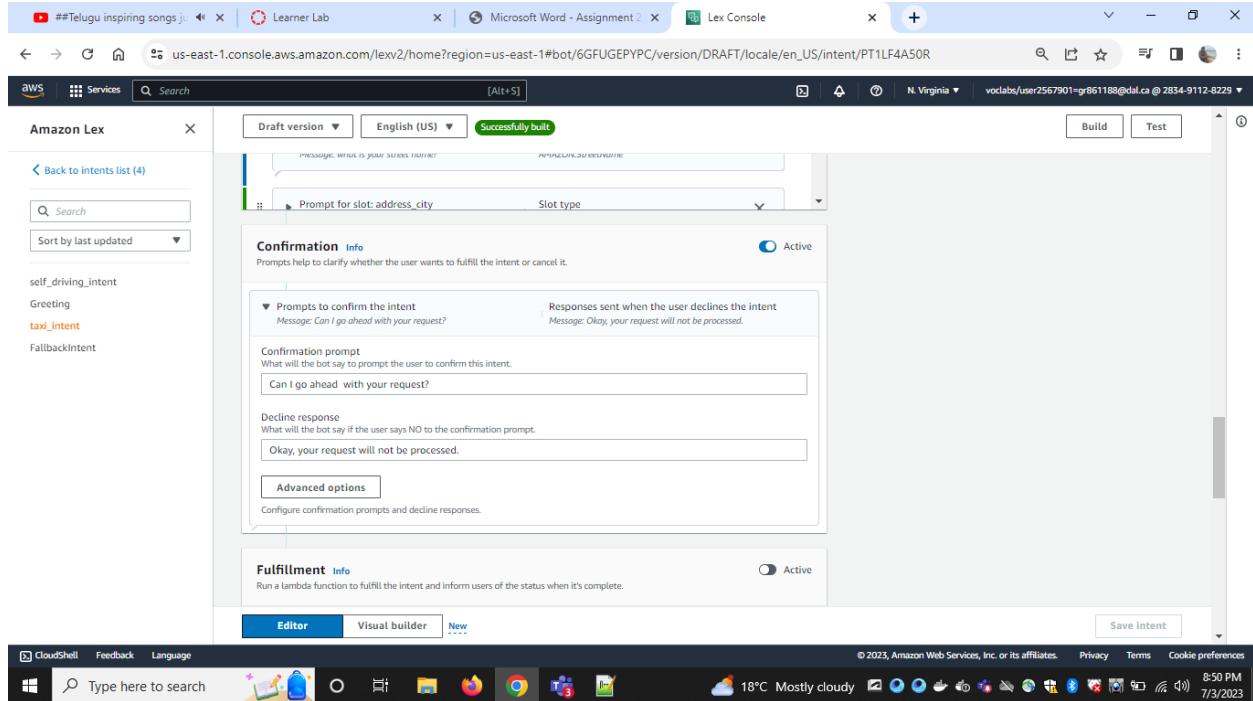


Figure 33: Request for getting confirmation

5. After confirmation, closing response is provided

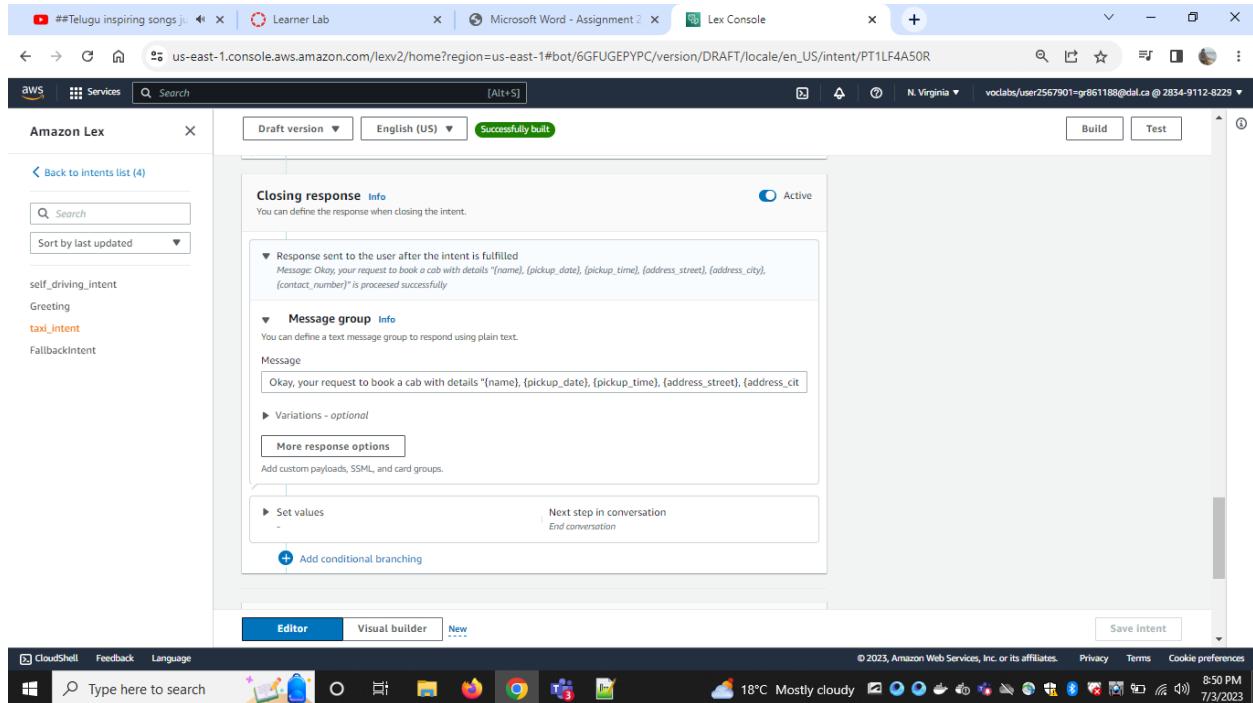


Figure 34: Closing message after service fulfillment

Taxi Booking Bot Execution

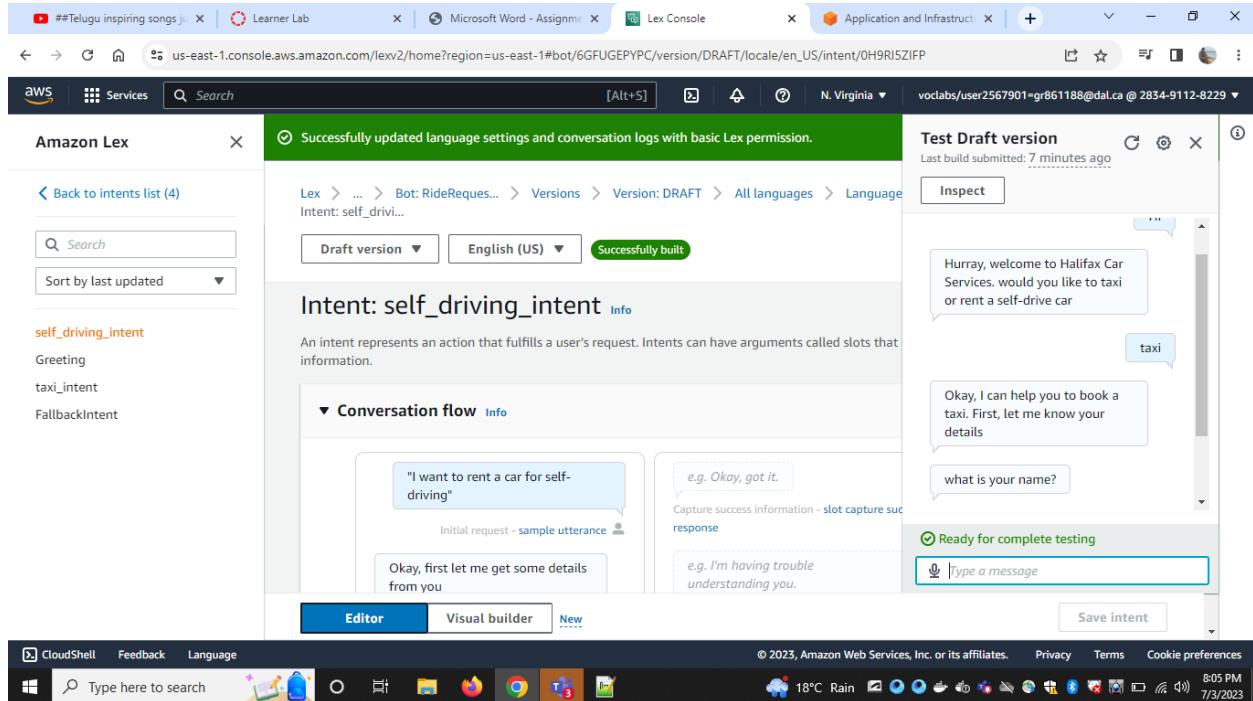


Figure 35: Bot execution-1

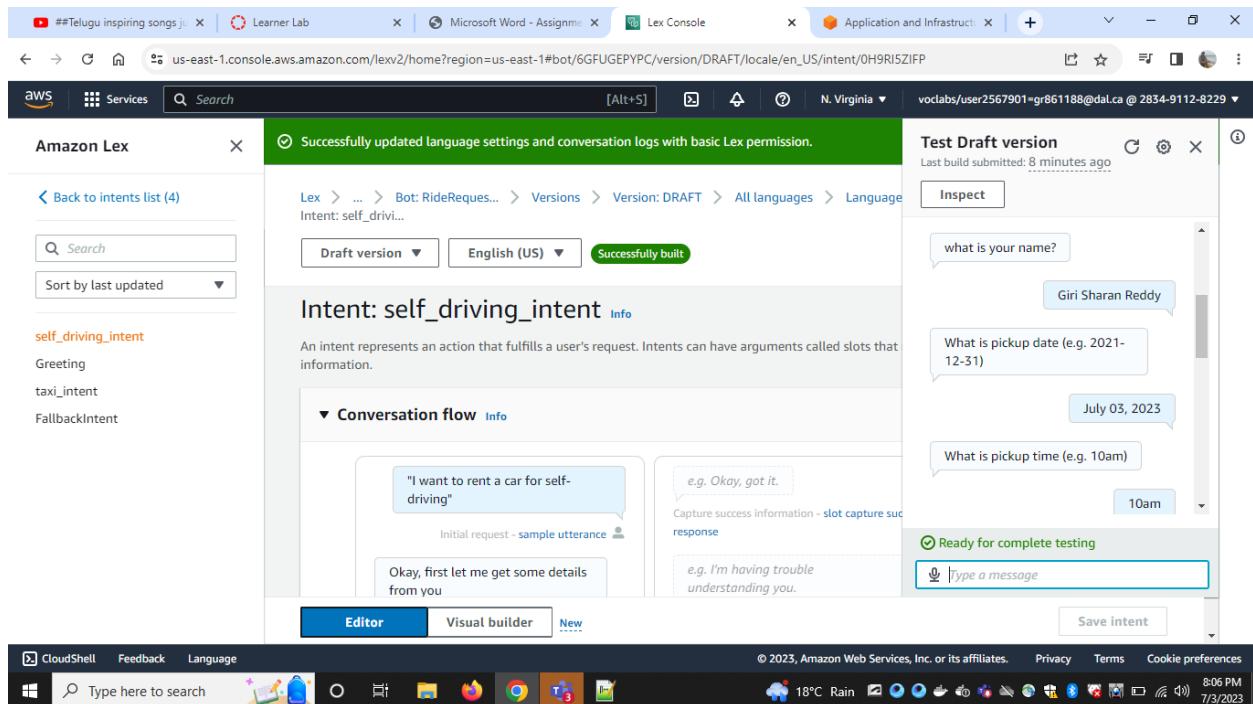


Figure 36: Bot execution-2

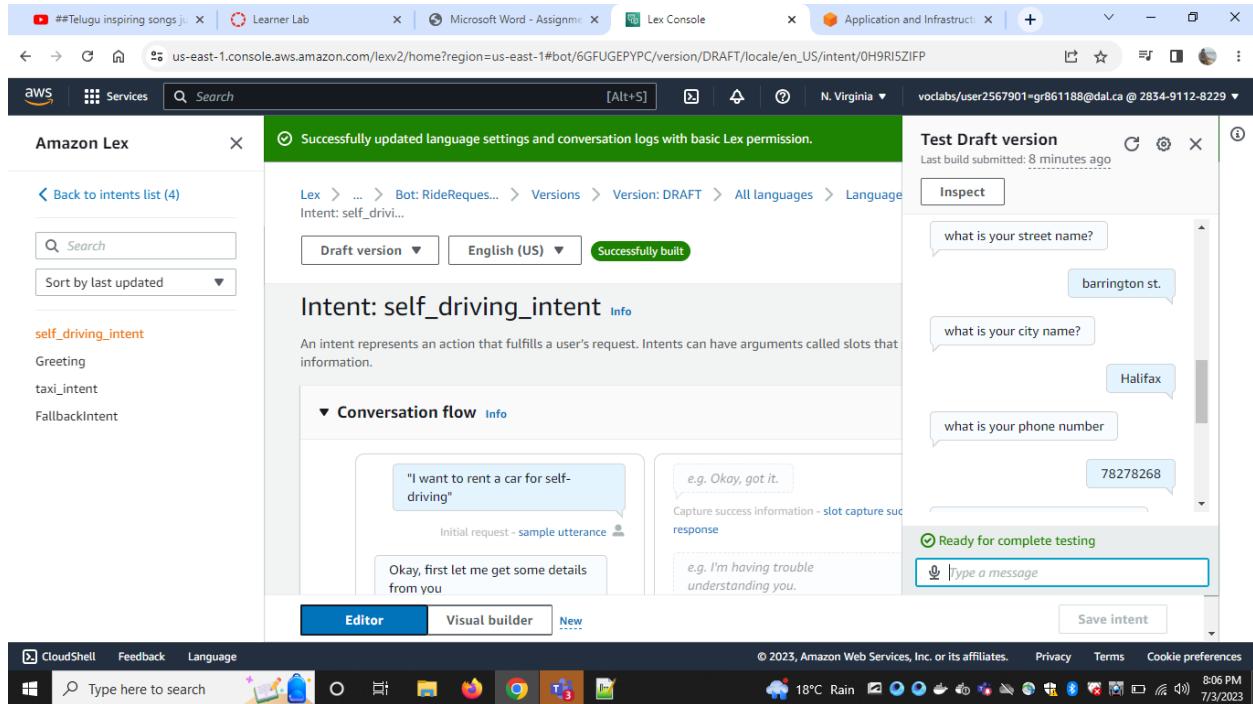


Figure 37: Bot execution-3

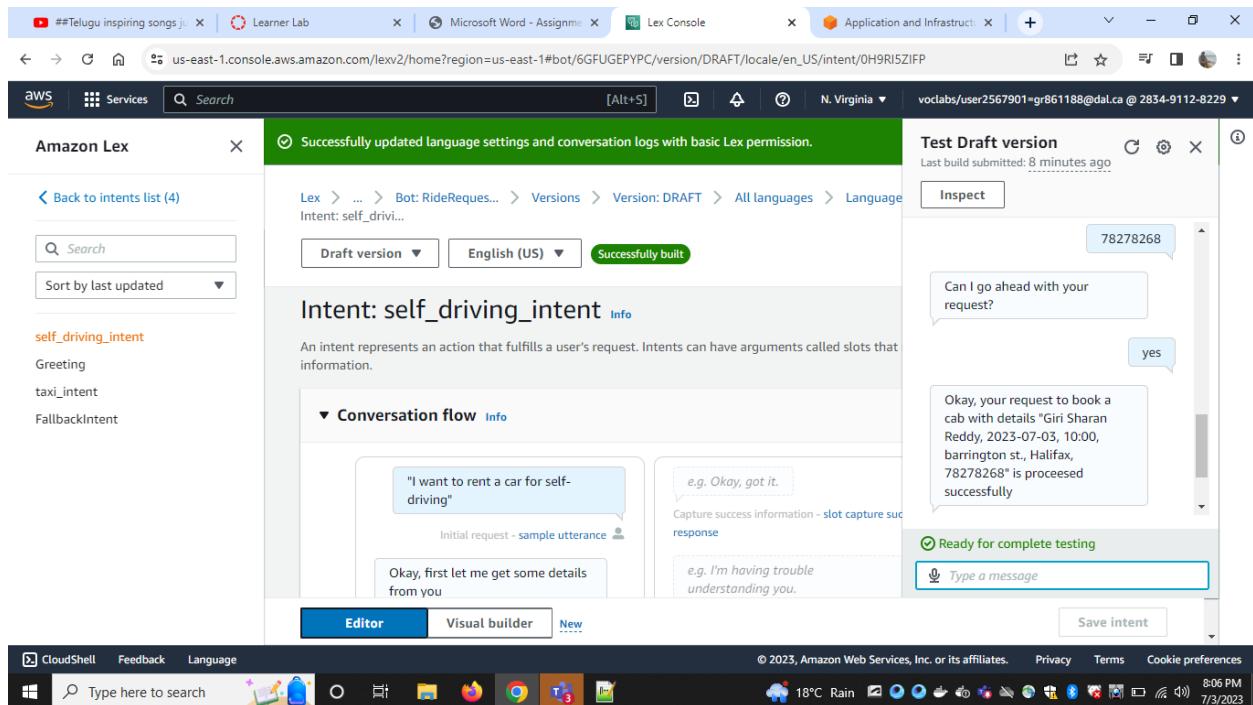


Figure 38: Bot execution-4

Steps Involved in Building Self-driving Car Booking Service Bot

1. Sample Utterances for accepting the self-driving car request

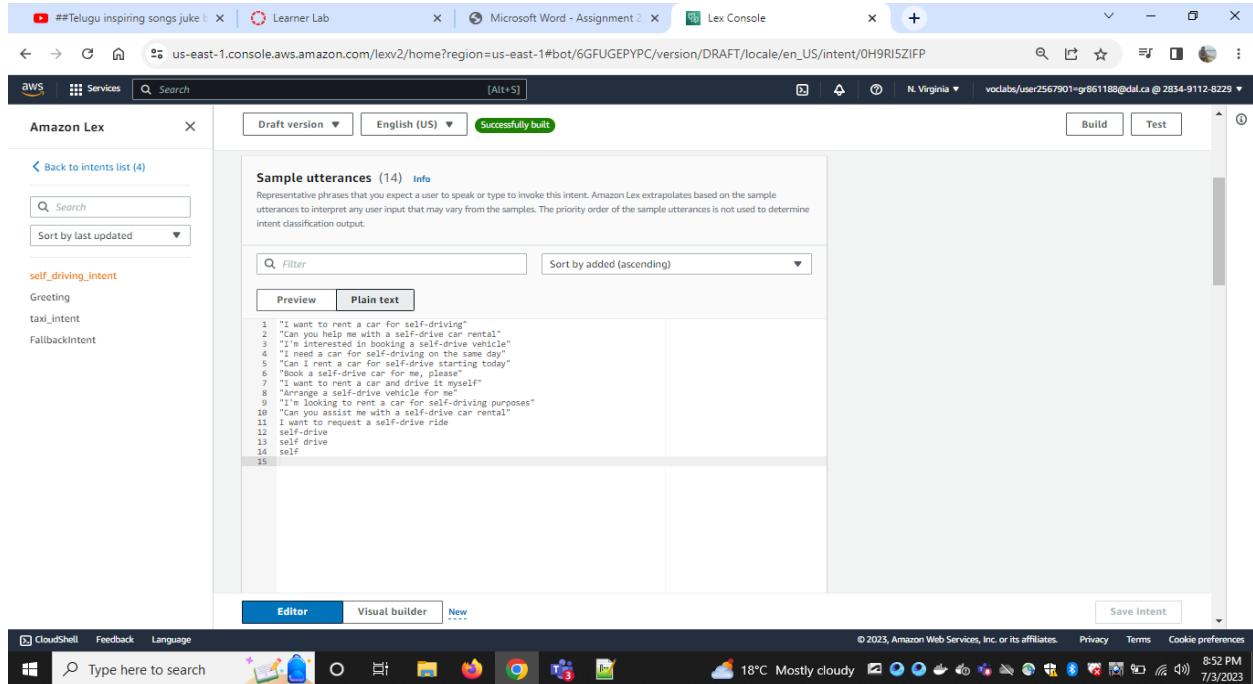


Figure 39: Sample utterances

2. Inputs that are required in order to book a taxi

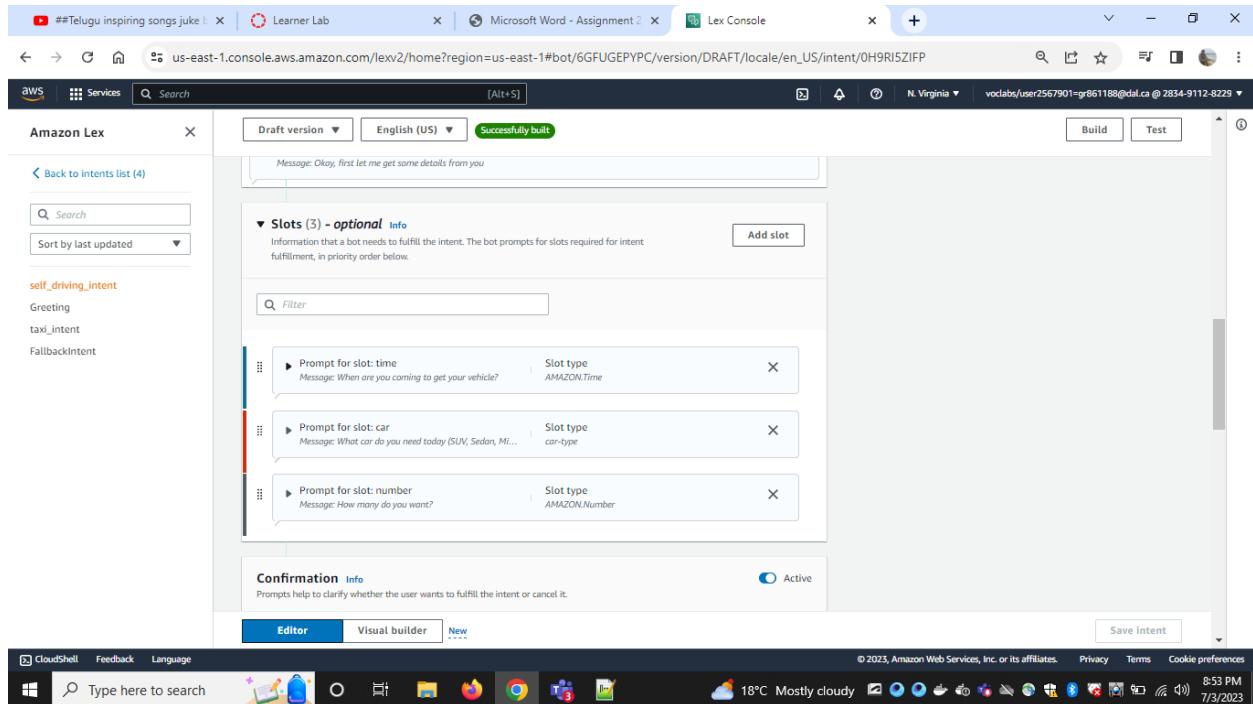


Figure 40: Slots for self-driving car service

3. Once user provided all the slots, bot asks for confirmation

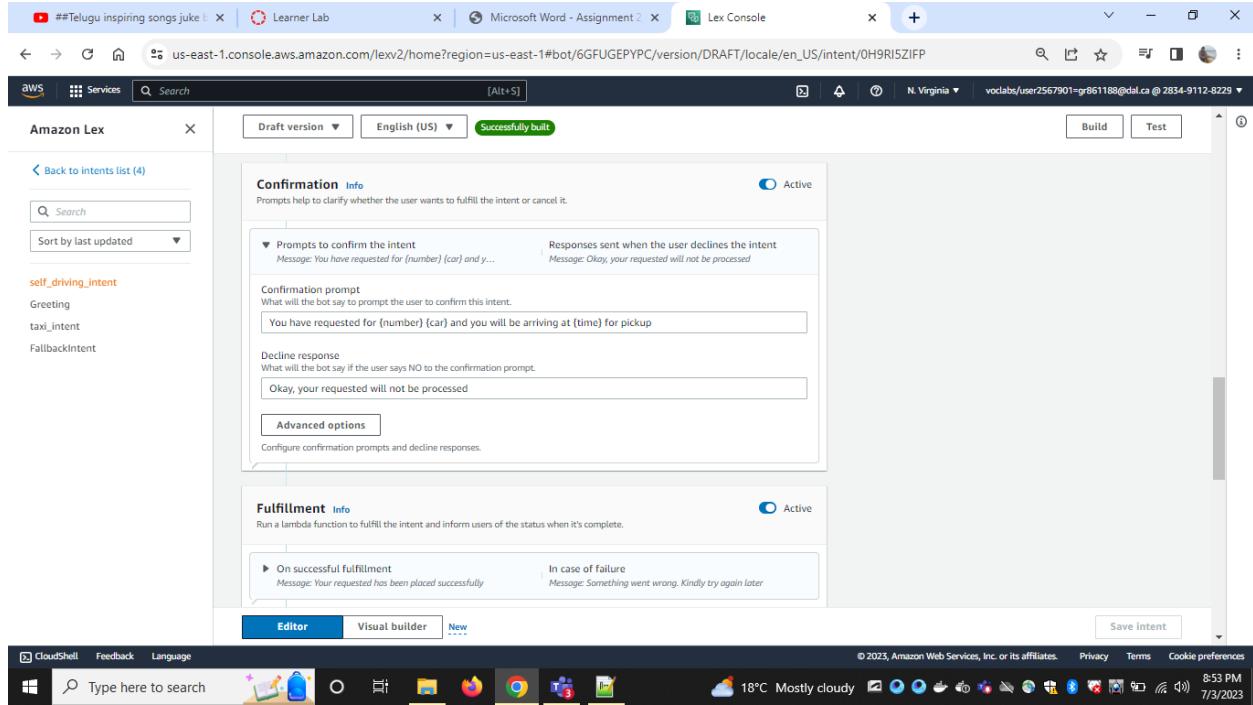


Figure 41: Request for getting confirmation

4. After confirmation, fulfillment response is provided else failure message

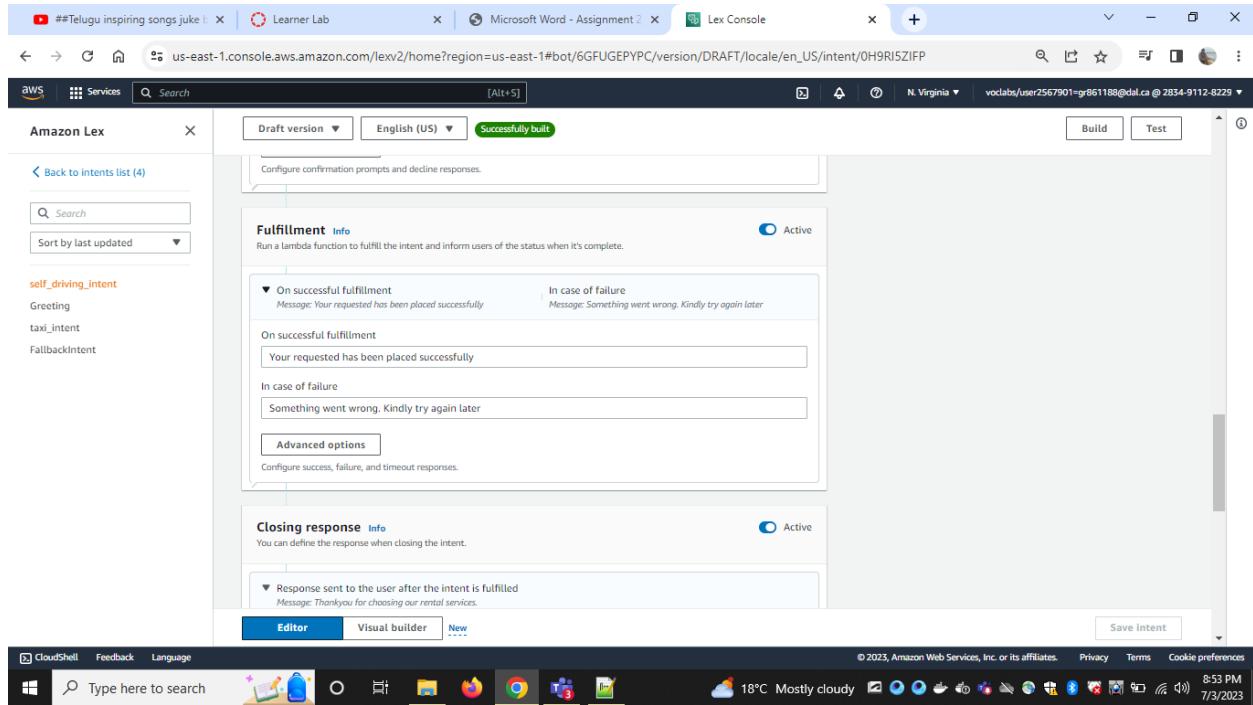


Figure 42: Fulfillment message to user

5. Closing response at the end after fulfillment of the service

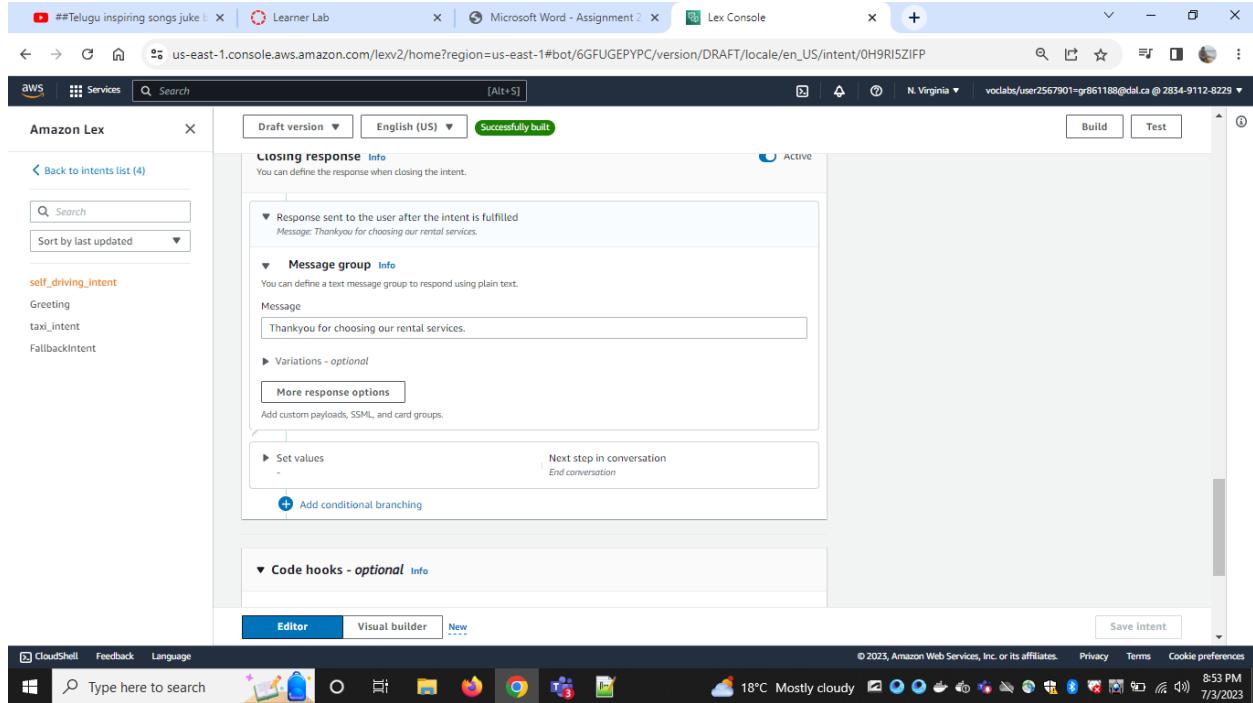


Figure 43: Closing response

Self-Driving Car Booking Bot Execution

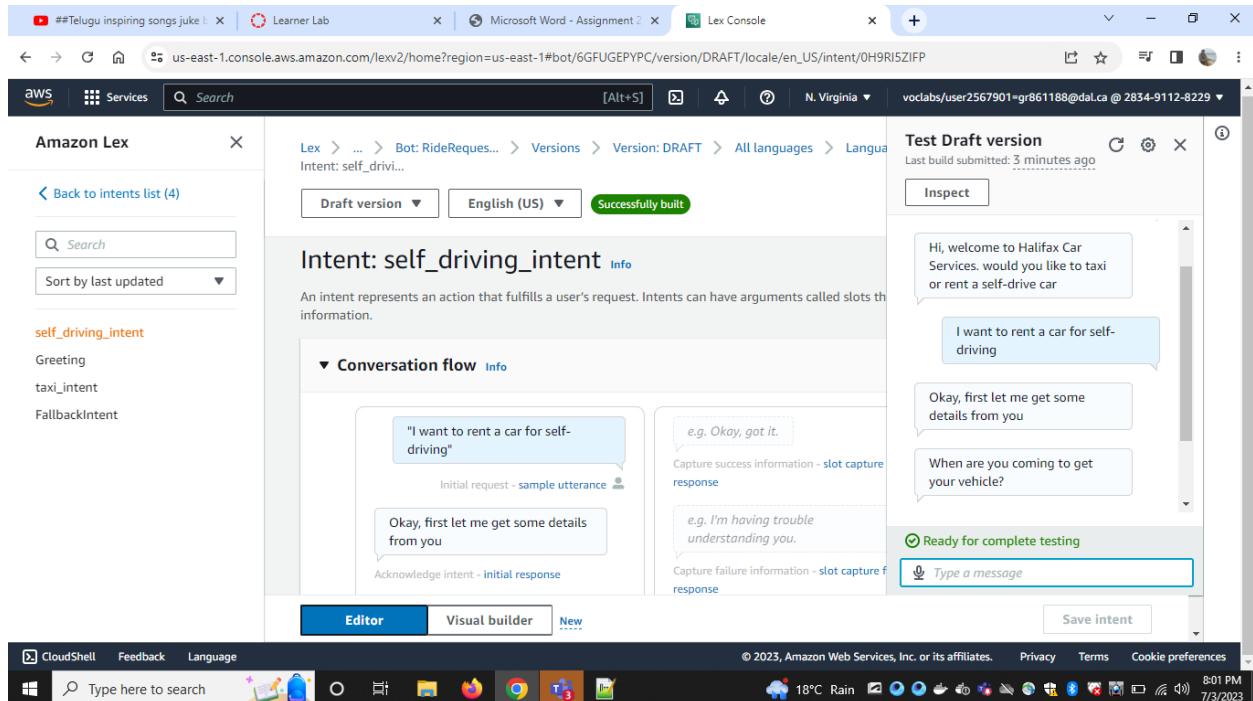


Figure 44: Bot execution-1

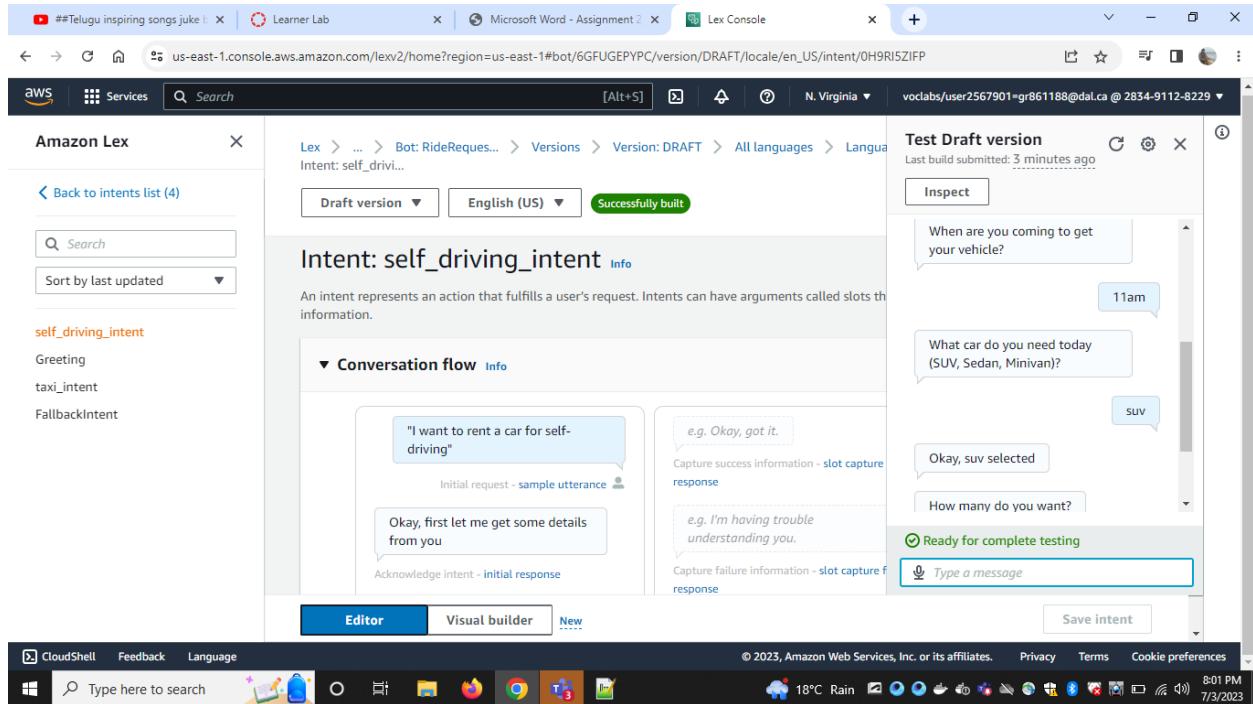


Figure 45: Bot execution-2

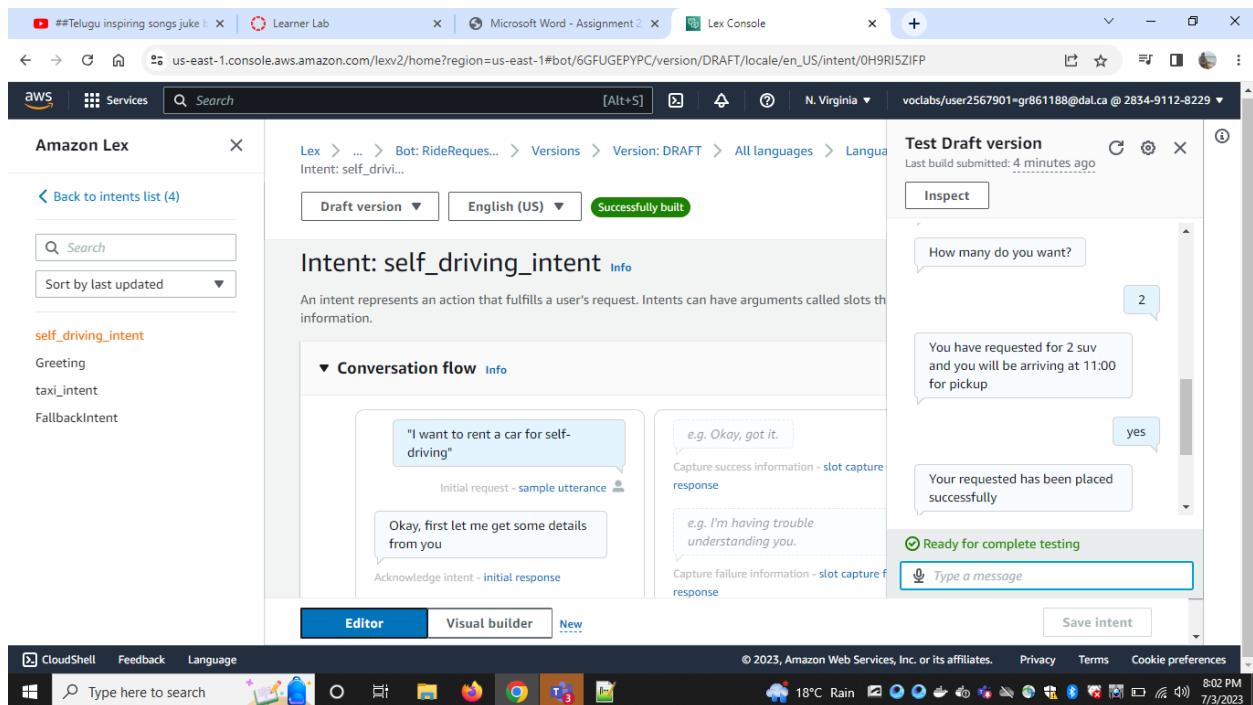


Figure 46: Bot execution-3

Procedure followed to build a bot

To build an Amazon Lex chatbot [6], we need to follow a step-by-step procedure. First, we should have the purpose of our chatbot by determining the main goal it should accomplish and the tasks or conversations it should handle. This will help us to understand the specific user inputs and intents your chatbot needs to recognize. Next, designing the conversation flow by creating a flowchart or diagram roughly that visualizes the interaction between the chatbot and users.

Once we have a clear understanding of your chatbot's purpose and conversation flow, we can proceed to set up the AWS Management Console. Sign in to the AWS Management Console and navigate to the Amazon Lex service [6].

In the Amazon Lex console, we can create a new bot by clicking on the "Create" button. Provide a name and description for our chatbot to help us in identifying and managing it later on. After creating chatbot, involves creating intents, which represent user goals or tasks, and defining sample utterances that users might those intents. We can also set up slots, which are used to gather specific pieces of information from users during the conversation.

Finally, we can test our chatbot using the built-in testing feature in the Amazon Lex console. This allows you to interact with your chatbot and see how it responds to different user inputs.

References

- [1] N. Naik, "Performance evaluation of distributed systems in multiple clouds using docker swarm," in *2021 IEEE International Systems Conference (SysCon)*, 2021, pp. 1–6.
- [2] "Administer and maintain a swarm of Docker Engines," *Docker Documentation*, 05-Jul-2023. [Online]. Available: https://docs.docker.com/engine/swarm/admin_guide/. [Accessed: 06-Jul-2023].
- [3] "What is cloud run," *Google Cloud*. [Online]. Available: <https://cloud.google.com/run/docs/overview/what-is-cloud-run>. [Accessed: 06-Jul-2023].
- [4] "Artifact registry overview," *Google Cloud*. [Online]. Available: <https://cloud.google.com/artifact-registry/docs/overview>. [Accessed: 06-Jul-2023].
- [5] "What is a container?," *Docker*, 11-Nov-2021. [Online]. Available: <https://www.docker.com/resources/what-container/>. [Accessed: 06-Jul-2023].
- [6] *Amazon.com*. [Online]. Available: <https://docs.aws.amazon.com/lexv2/latest/dg/what-is.html>. [Accessed: 06-Jul-2023].