



# **CSCI 5410 Serverless Data Processing**

## **Assignment-2**

**June 10, 2020**

**Submitted By**

Vamsi Gamidi, B00834696

## Table of Contents

Part A.....	3
Summary .....	4
Screenshots.....	4
Source Code Snippets .....	8
Steps to run the Application .....	8
Snippet 1: Changing the User Status to Online.....	8
Snippet 2: Setting Status to offline after Logout .....	9
Snippet 3: Sending Post Request for Registration .....	9
Testing.....	10
Test Case 1 .....	10
Test Case 2 .....	10
Test Case 3 .....	11
Part B.....	12
Summary .....	12
Screenshots.....	12
References .....	17

## Table of Figures

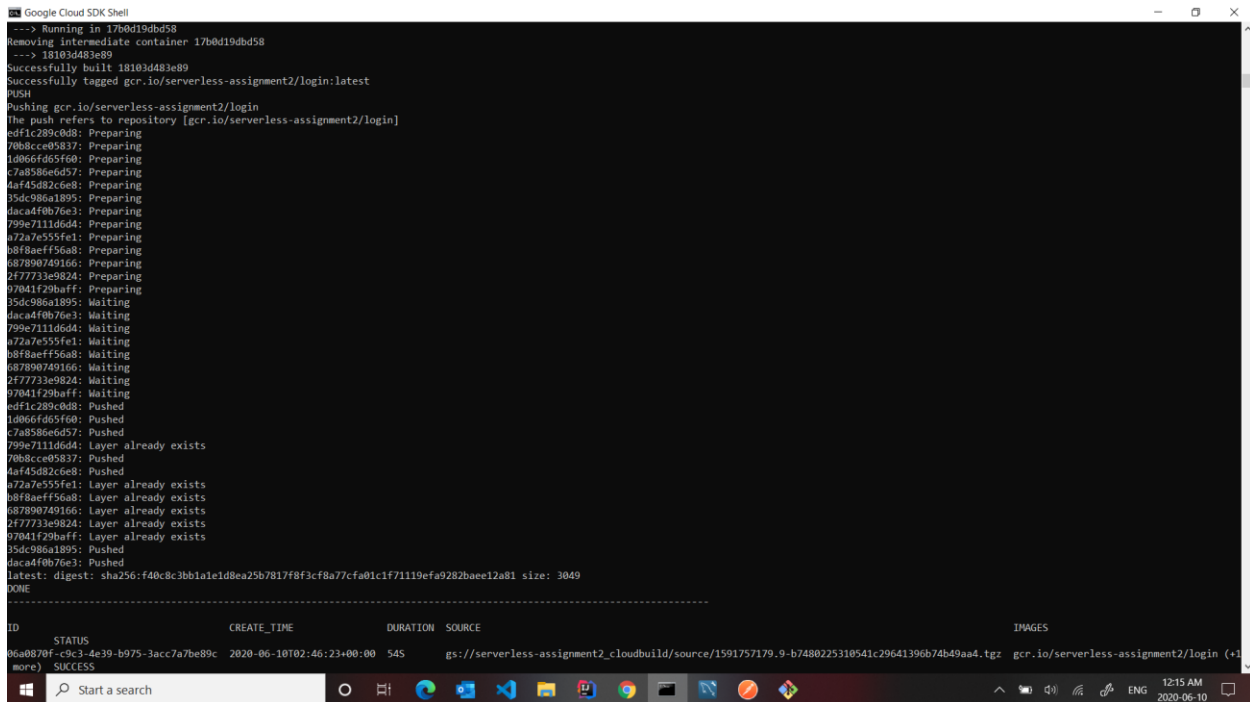
Figure 1: Login Container .....	4
Figure 2: Registration Container .....	5
Figure 3: State Information Container .....	5
Figure 4: Docker Images List .....	6
Figure 5: Services in Google Cloud Run.....	6
Figure 6: RDS MySQL Database.....	7
Figure 7: User Table data .....	7
Figure 8: Status Table data .....	8
Figure 9: Registration Page .....	10
Figure 10: Login Page .....	11
Figure 11: State Information Screen.....	11
Figure 12: Bot Creation Home Page.....	12
Figure 13: Bot Details.....	13
Figure 14: Bot Home Page after Creation .....	13
Figure 15: Intent Creation.....	14
Figure 16: Adding slot to intent .....	14
Figure 17: Adding Utterance, Slots, Responses for Order Pickup.....	15
Figure 18: Testing bot for order pickup .....	15
Figure 19: Adding Utterance, Slots, Response for Order Delivery.....	16
Figure 20: Testing bot for Order Delivery .....	16

## Part A

### Summary

I have developed the front end of the application i.e. login page, user registration page, state information page using React JS. For handling database operations and handling the get and post requests from front end, I have written services using Node JS. To sending http requests from front end to back end, I have used Axios library. I have also created a MySQL database in Amazon RDS and created 2 tables: user, and status. In the user table, I am storing the details given by the user while registration and in the status table, I am storing the email id, time stamp, whether the user is online or offline. After writing the separate services for login, registration, and state information, I have created docker images for each service [6]. By creating docker images with google cloud SDK shell, the images will be automatically uploaded to google container registry [4]. I have created 3 services namely login, register, stateinfo in google cloud run [3]. While creating the services, I have attached the images that were uploaded to google container registry earlier. After the services are created, I have taken the URL generated by google cloud run and used them in my front end to send http get and post requests. I have run the react app using 'npm start' and tested the application to check all the login, registration, and state information features. I have presented the screenshots for all the steps that I have performed in this task. I find google cloud easier to use when compared to AWS due to the integration of features such as Google Container Registry, Google SDK Shell, Google Run. The service creation and attaching the images to services is also straight forward and the documentation is also clear.

### Screenshots



```
Google Cloud SDK Shell
---> Running in 17b0d19dbd58
Removing intermediate container 17b0d19dbd58
--> 18103d483e89
Successfully built 18103d483e89
Successfully tagged gcr.io/serverless-assignment2/login:latest
PUSH
Pushing gcr.io/serverless-assignment2/login
The push refers to repository [gcr.io/serverless-assignment2/login]
edf1c289c0d8: Preparing
70b8cce95837: Preparing
1d066fd65f60: Preparing
c7a8586e6d57: Preparing
4af45d82c6e8: Preparing
35dc986a1895: Preparing
daca4f0b76e3: Preparing
799e711d6d4: Preparing
a72a7e555fe1: Preparing
b8f8aeff56a8: Preparing
687890749166: Preparing
2f77733e9824: Preparing
97041f29baff: Preparing
35dc986a1895: Waiting
daca4f0b76e3: Waiting
799e711d6d4: Waiting
a72a7e555fe1: Waiting
b8f8aeff56a8: Waiting
687890749166: Waiting
2f77733e9824: Waiting
97041f29baff: Waiting
edf1c289c0d8: Pushed
1d066fd65f60: Pushed
c7a8586e6d57: Pushed
799e711d6d4: Layer already exists
70b8cce95837: Pushed
4af45d82c6e8: Pushed
a72a7e555fe1: Layer already exists
b8f8aeff56a8: Layer already exists
687890749166: Layer already exists
2f77733e9824: Layer already exists
97041f29baff: Layer already exists
35dc986a1895: Pushed
daca4f0b76e3: Pushed
latest: digest: sha256:f40c8c3bb1a1c1d8ea25b7817f8f3cf8a77cf01c1f71119efa9282baee12a81 size: 3049
DONE
-----
ID                                STATUS                                CREATE_TIME                                DURATION                                SOURCE                                IMAGES
06a8870f-c9c3-4e39-b975-3acc7a7be89c  SUCCESS                                2020-06-10T02:46:23+00:00  54s                                gs://serverless-assignment2_cloudbuild/source/1591757179-9-b7480225310541c29641396b74b49aa4.tgz  gcr.io/serverless-assignment2/login (+1
more)
```

Figure 1: Login Container

```
Google Cloud SDK Shell
Alternative official "docker" images, including multiple versions across
multiple platforms, are maintained by the Docker Team. For details, please
visit https://hub.docker.com/_/docker.

***** END OF NOTICE *****

The push refers to repository [gcr.io/serverless-assignment2/register]
4697f176ef5b: Preparing
1dc6217af96: Preparing
75affadb6c80: Preparing
0a83f9d67dc3: Preparing
4af45d82c6e8: Preparing
35dc986a1895: Preparing
daca4f0b76e3: Preparing
799e711d6d4: Preparing
a72a7e555fe1: Preparing
b8f8aef56a8: Preparing
687890749166: Preparing
2f77733e9824: Preparing
97041f29baff: Preparing
35dc986a1895: Waiting
daca4f0b76e3: Waiting
799e711d6d4: Waiting
a72a7e555fe1: Waiting
b8f8aef56a8: Waiting
2f77733e9824: Waiting
97041f29baff: Waiting
687890749166: Waiting
4af45d82c6e8: Layer already exists
35dc986a1895: Layer already exists
daca4f0b76e3: Layer already exists
799e711d6d4: Layer already exists
a72a7e555fe1: Layer already exists
b8f8aef56a8: Layer already exists
687890749166: Layer already exists
2f77733e9824: Layer already exists
97041f29baff: Layer already exists
0a83f9d67dc3: Pushed
4697f176ef5b: Pushed
75affadb6c80: Pushed
1dc6217af96: Pushed
latest: digest: sha256:35b66fd93d60c35b152da59081c0bc62a36dbaf5a5767e6aeba3db3c6d948743 size: 3051
DONE

-----
ID              STATUS              CREATE_TIME          DURATION  SOURCE                                     IMAGES
4147290f-288b-453a-9425-bde789ee00c6  2020-06-10T02:53:10+00:00  51s          gs://serverless-assignment2_cloudbuild/source/1591757586.59-f1111ecaeald462786f3f5073d12f8f0.tgz  gcr.io/serverless-assignment2/register
(+1 more)  SUCCESS
```

Figure 2: Registration Container

```
Google Cloud SDK Shell
Removing intermediate container 7f9e64b3891e
--> 96451cb4762c
Successfully built 96451cb4762c
Successfully tagged gcr.io/serverless-assignment2/stateinfo:latest
PUSH
Pushing gcr.io/serverless-assignment2/stateinfo
The push refers to repository [gcr.io/serverless-assignment2/stateinfo]
0e67acad42a9: Preparing
e840c2deb67c: Preparing
5b19053f8523: Preparing
ed798b9b0348: Preparing
4af45d82c6e8: Preparing
35dc986a1895: Preparing
daca4f0b76e3: Preparing
799e711d6d4: Preparing
a72a7e555fe1: Preparing
b8f8aef56a8: Preparing
687890749166: Preparing
2f77733e9824: Preparing
97041f29baff: Preparing
35dc986a1895: Waiting
daca4f0b76e3: Waiting
799e711d6d4: Waiting
a72a7e555fe1: Waiting
b8f8aef56a8: Waiting
687890749166: Waiting
2f77733e9824: Waiting
97041f29baff: Waiting
4af45d82c6e8: Layer already exists
35dc986a1895: Layer already exists
daca4f0b76e3: Layer already exists
799e711d6d4: Layer already exists
a72a7e555fe1: Layer already exists
b8f8aef56a8: Layer already exists
687890749166: Layer already exists
2f77733e9824: Layer already exists
97041f29baff: Layer already exists
0e67acad42a9: Pushed
ed798b9b0348: Pushed
5b19053f8523: Pushed
e840c2deb67c: Pushed
latest: digest: sha256:3ee356d0fe1bd3d031a779cb53e7342078017371afffd81e50cf636dfef8e833 size: 3049
DONE

-----
ID              STATUS              CREATE_TIME          DURATION  SOURCE                                     IMAGES
ed2a2e85-58df-4dca-9658-671ad8a4dc7b  2020-06-10T02:54:28+00:00  49s          gs://serverless-assignment2_cloudbuild/source/1591757665.23-ab67c24b215e4c2eaa8f642a4e5172e.tgz  gcr.io/serverless-assignment2/stateinfo
(+1 more)  SUCCESS
```

Figure 3: State Information Container

```
Google Cloud SDK Shell
C:\Users\vamsi\Term-3\Serverless Data Processing\Assignment-2\assign2\authentication\stateinfo>gcloud container images list
NAME
gcr.io/serverless-assignment2/login
gcr.io/serverless-assignment2/register
gcr.io/serverless-assignment2/stateinfo
Only listing images in gcr.io/serverless-assignment2. Use --repository to list images in other repositories.

C:\Users\vamsi\Term-3\Serverless Data Processing\Assignment-2\assign2\authentication\stateinfo>
```

Figure 4: Docker Images List

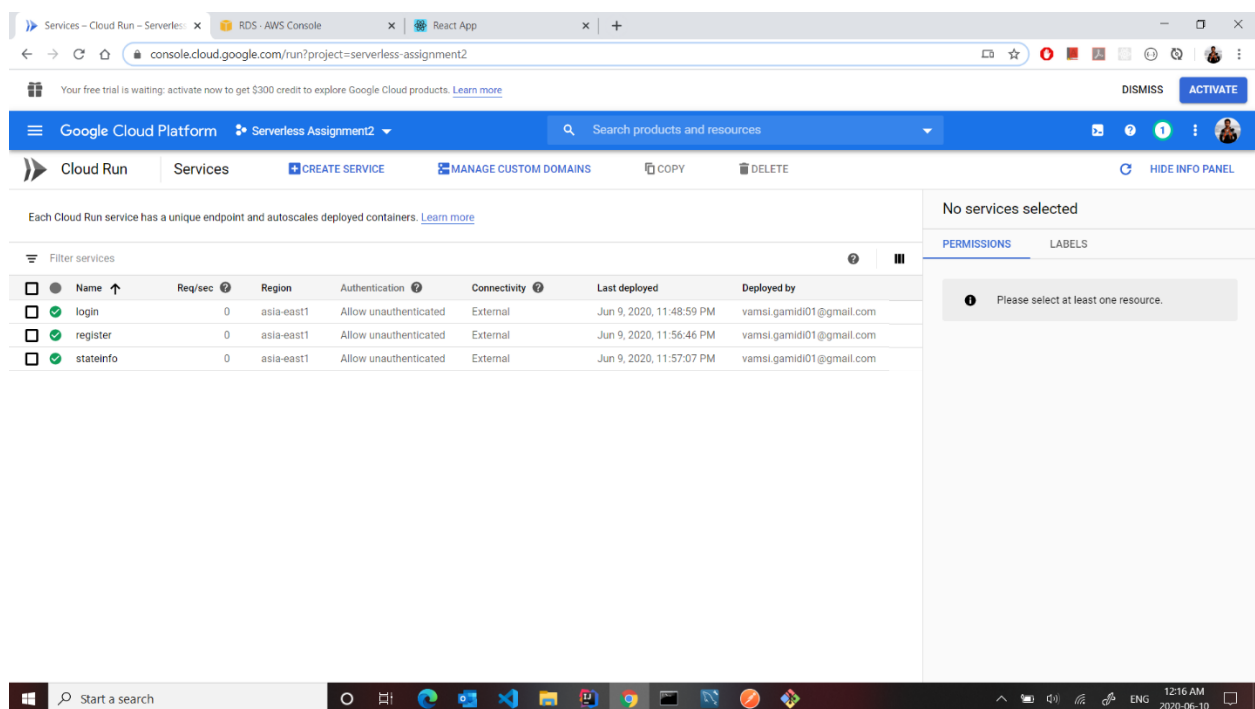


Figure 5: Services in Google Cloud Run

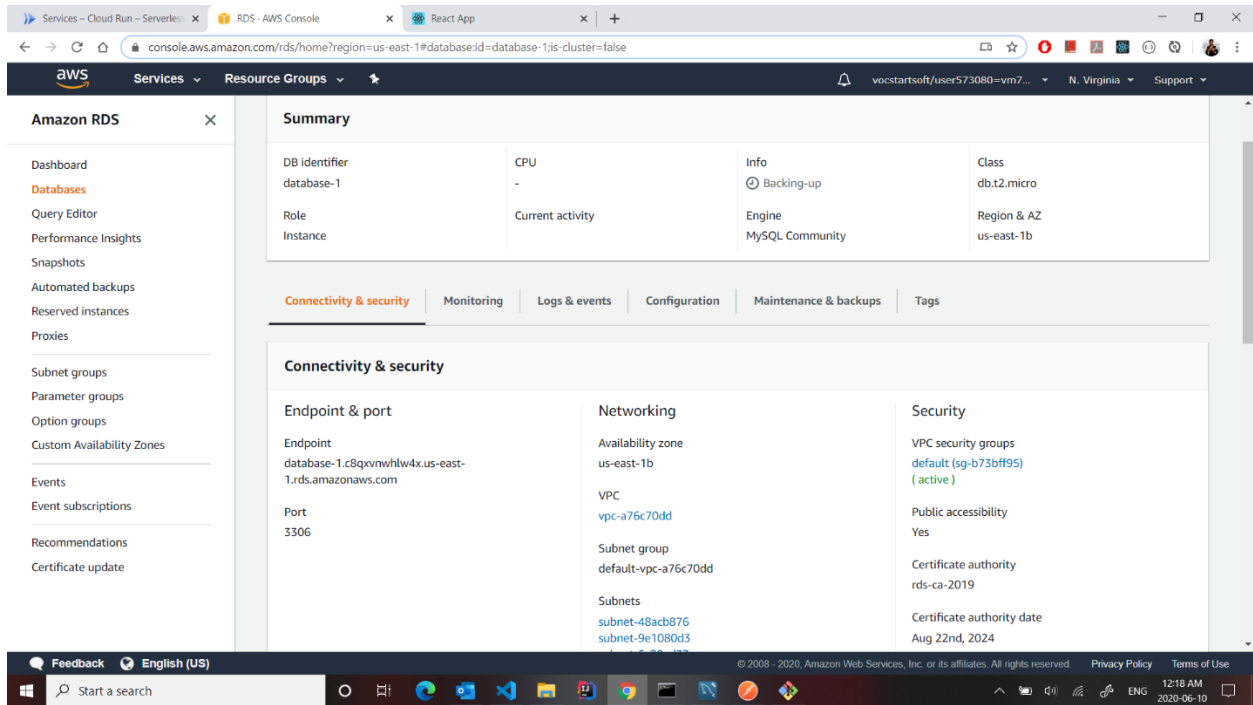


Figure 6: RDS MySQL Database

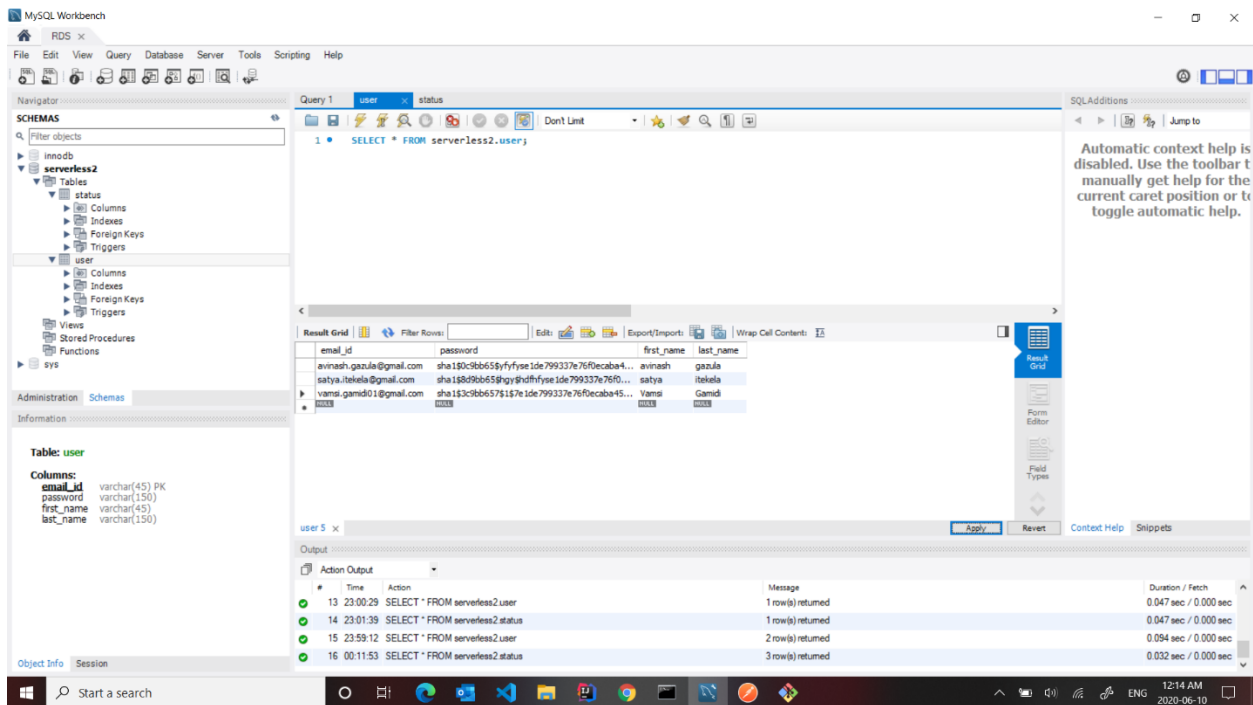


Figure 7: User Table data

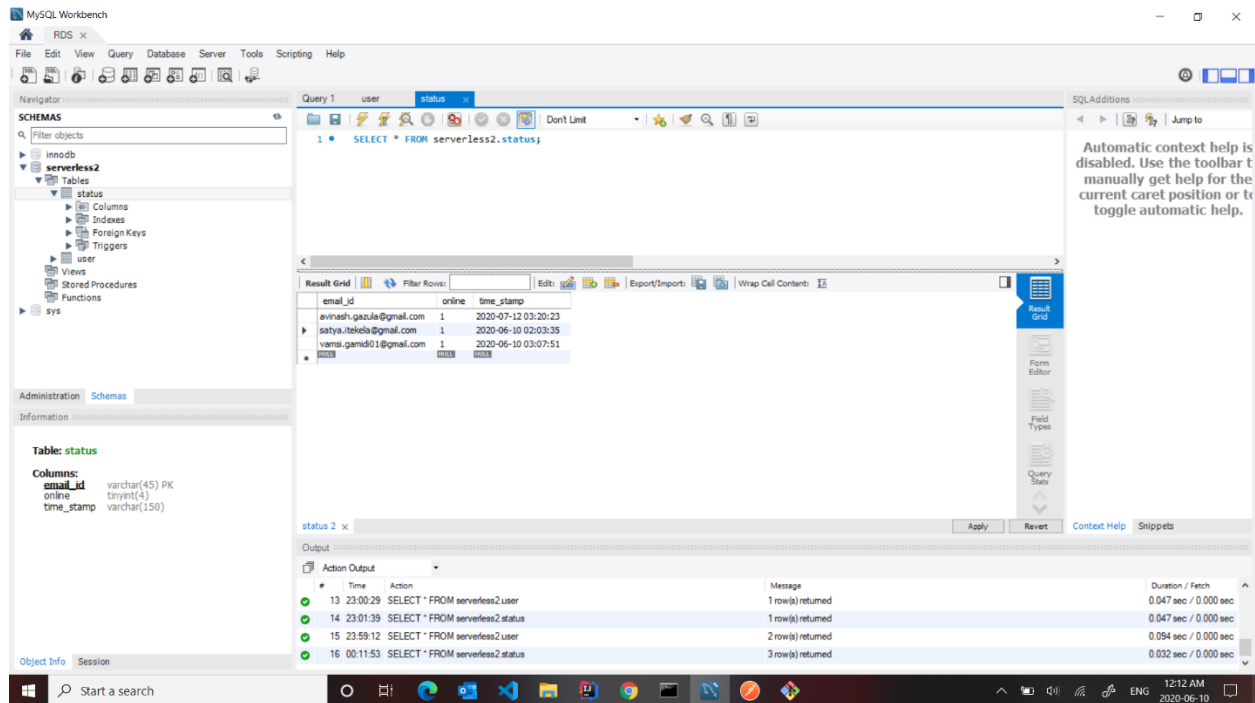


Figure 8: Status Table data

## Source Code Snippets

### Steps to run the Application

To run the application (front-end), please follow the below steps:

- Download the zip file
- Unzip the file
- Open the folder in VS Code
- Open terminal in VS Code
- Run 'cd client'
- Run 'npm install'
- Run 'npm start'
- Application will be opened in the browser

### Snippet 1: Changing the User Status to Online

```
exports.setOnline = (req, res, next) => {
  var sql = `insert into status values("${req.params.emailId}", true, now())
  on duplicate key update online=true, time_stamp=now()`;
```



```

connection.query(sql, function (err, result){
  console.log(sql);
  if(!err){
    console.log("Status Updated");
    return res.status(200).json({
      success: true
    });
  }
  else{
    console.log(err);
  }
})
}

```

Snippet 2: Setting Status to offline after Logout

```

exports.logout = (req, res, next) => {
  var sql = `update status set online=false, time_stamp=now() where email_id="${req.params.emailId}"`;
  try {
    connection.query(sql, function (err, result) {
      if (!err) {
        console.log(result);
        console.log("Logged Out");
        return res.status(200).json({
          success: true,
        });
      } else {
        console.log(err);
      }
    });
  } catch (e) {
    console.log(e);
  }
};

```

Snippet 3: Sending Post Request for Registration

```

const handleSubmit = (e) => {
  e.preventDefault();
  axios.post('https://register-y6nn3qcdoq-de.a.run.app/users/register', {
    emailId,
    password,
  })
}

```

```

        firstName,
        lastName
    }).then(response => {
        history.push('/login')
        console.log(response);
    }).catch(
        error => console.log(error)
    );
}

```

## Testing

### Test Case 1

User should be able to register by giving details such as email id, password, first name, last name.

The screenshot displays a web browser window with the URL `localhost:3000/register`. The page is titled "State Management" and features a registration form. The form includes four input fields: the first contains "Vamsi", the second contains "Gamidi", the third contains the email "vamsi.gamidi01@gmail.com", and the fourth is a password field with masked characters. A "REGISTER" button is positioned below the password field. At the bottom of the form, there is a link that reads "Already have an account? Login". The browser's taskbar at the bottom indicates the time is 12:11 AM on 2020-06-10.

Figure 9: Registration Page

### Test Case 2

After successful registration, user should be taken to login screen. In the login screen, by entering valid credentials, user should be able to login.

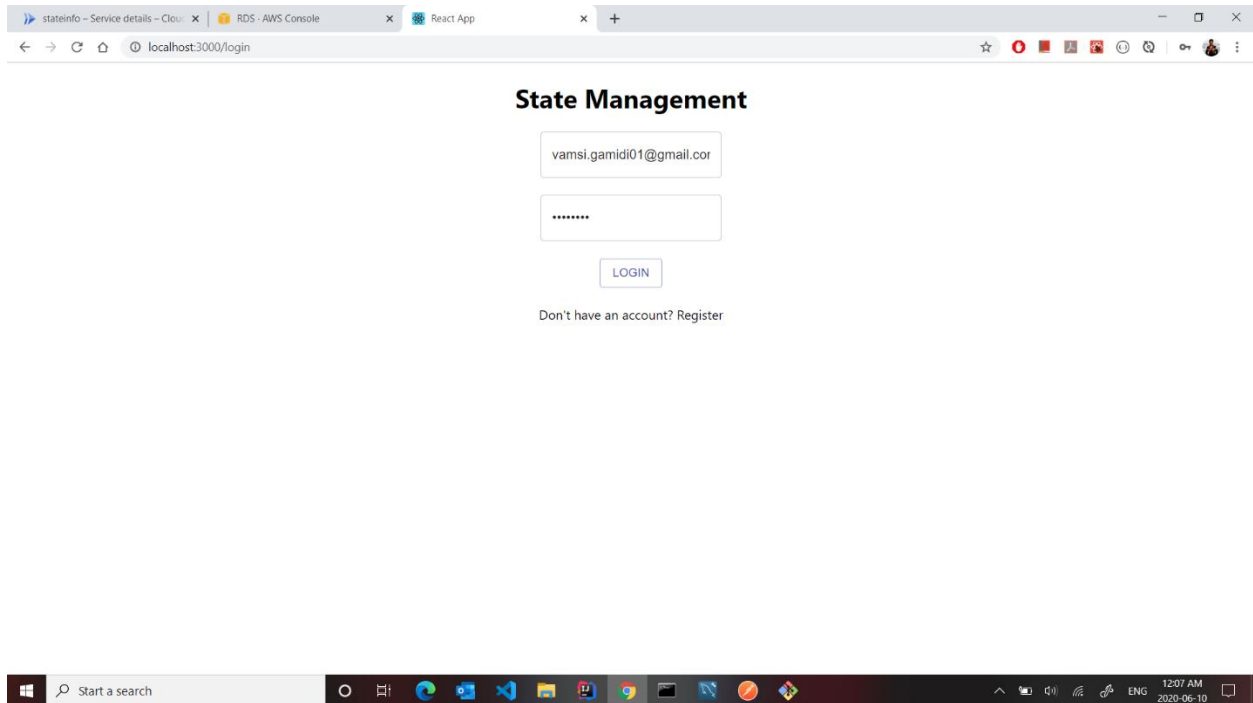


Figure 10: Login Page

### Test Case 3

After logging in by entering valid credentials, user should be taken to state information screen where the list of all online users will be displayed.

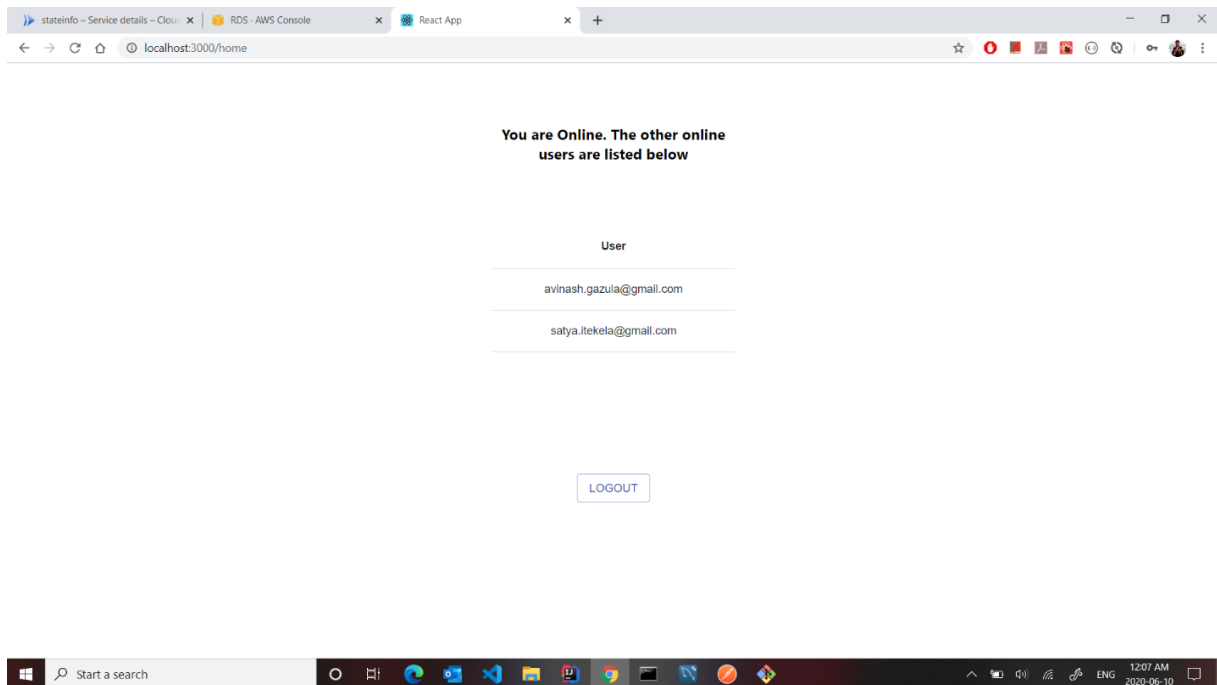


Figure 11: State Information Screen

## Part B

### Summary

Amazon Lex is a service provided by Amazon Web Services for building chat bots. Amazon Lex supports integration with services such as AWS Lambda, AWS MobileHub, AWS CloudWatch [1]. By using Amazon Lex, I have created a custom bot with the name 'PizzaForAll' and session timeout period of 10 mins [2]. After creating the bot, I have created an intent for pizza pickup. I have added a custom slot type 'PizzaType' for specifying the type of the pizza. Later, I have added the utterance message, slots, and Response messages. After building the bot, I have tested it and provided screenshots. After completing the pizza pickup task, I have created another intent for delivery task. I have added utterance message, slots, and response message relevant to delivery task and tested the bot. Please go through the screenshots provided in which I have demonstrated every step in creation and testing of the bot.

### Screenshots

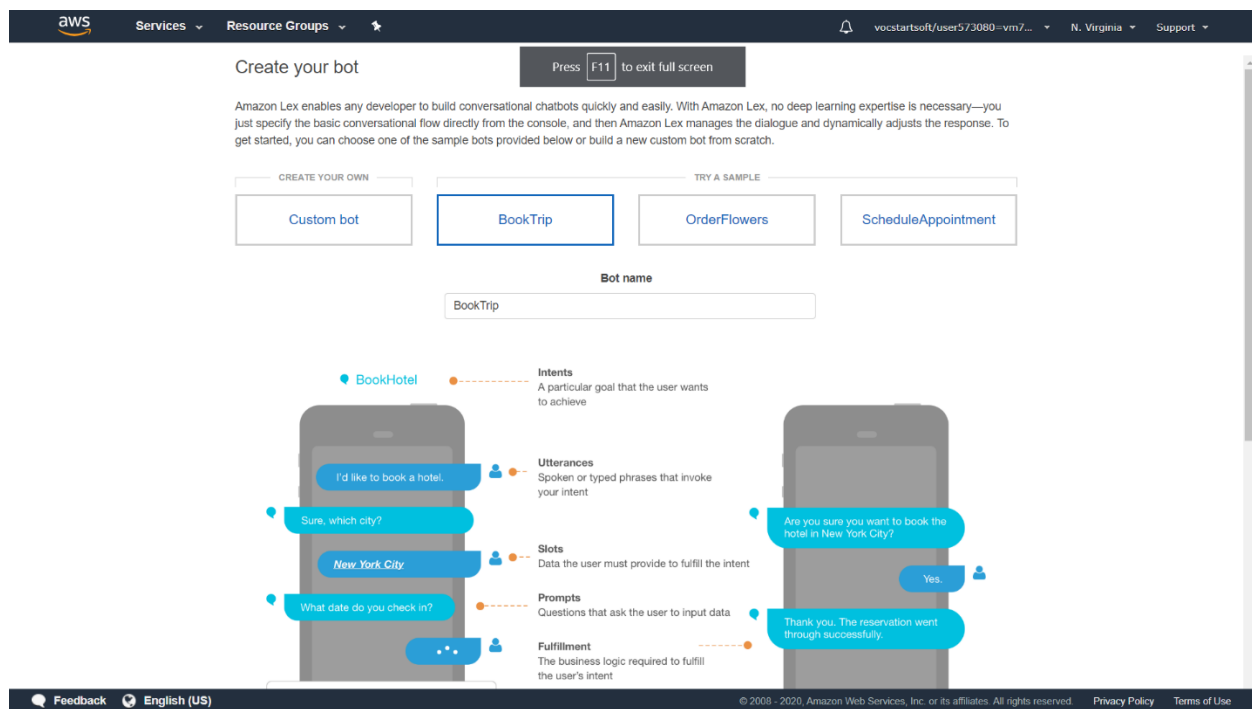


Figure 12: Bot Creation Home Page

The screenshot shows the 'Bot Details' page for a bot named 'PizzaForAll'. The page is divided into two main sections: 'CREATE YOUR OWN' and 'TRY A SAMPLE'. Under 'CREATE YOUR OWN', there are four buttons: 'Custom bot' (highlighted), 'BookTrip', 'OrderFlowers', and 'ScheduleAppointment'. The 'Bot name' field is set to 'PizzaForAll'. The 'Language' is set to 'English (US)'. The 'Output voice' is set to 'None. This is only a text based applicati...'. The 'Session timeout' is set to '10 min'. The 'Sentiment analysis' is set to 'No'. The 'IAM role' is 'AWSServiceRoleForLexBots'. The 'COPPA' section has a 'No' selection. There is a 'Tags' section at the bottom. At the bottom right, there are 'Cancel' and 'Create' buttons.

Figure 13: Bot Details

The screenshot shows the 'Bot Home Page' for the 'PizzaForAll' bot. The page has a navigation bar with 'Editor', 'Settings', 'Channels', and 'Monitoring'. The 'Editor' tab is selected. On the left, there are sections for 'Intents' (No intents created), 'Slot types' (No slots created), and 'Error Handling'. The main content area is titled 'Getting started with your bot' and includes a 'Welcome to your bot editor' message. Below this, there is a 'Components of your bot' diagram. The diagram shows a sequence of interactions: an intent 'BookHotel', an utterance 'I'd like to book a hotel.', a slot 'New York City', a prompt 'What date do you check in?', and a fulfillment 'November 30th.'. The diagram also shows a confirmation dialog 'Are you sure you want to book the hotel in New York City?' with a 'Yes' response, and a final message 'Thank you. The reservation went through successfully.'.

Figure 14: Bot Home Page after Creation

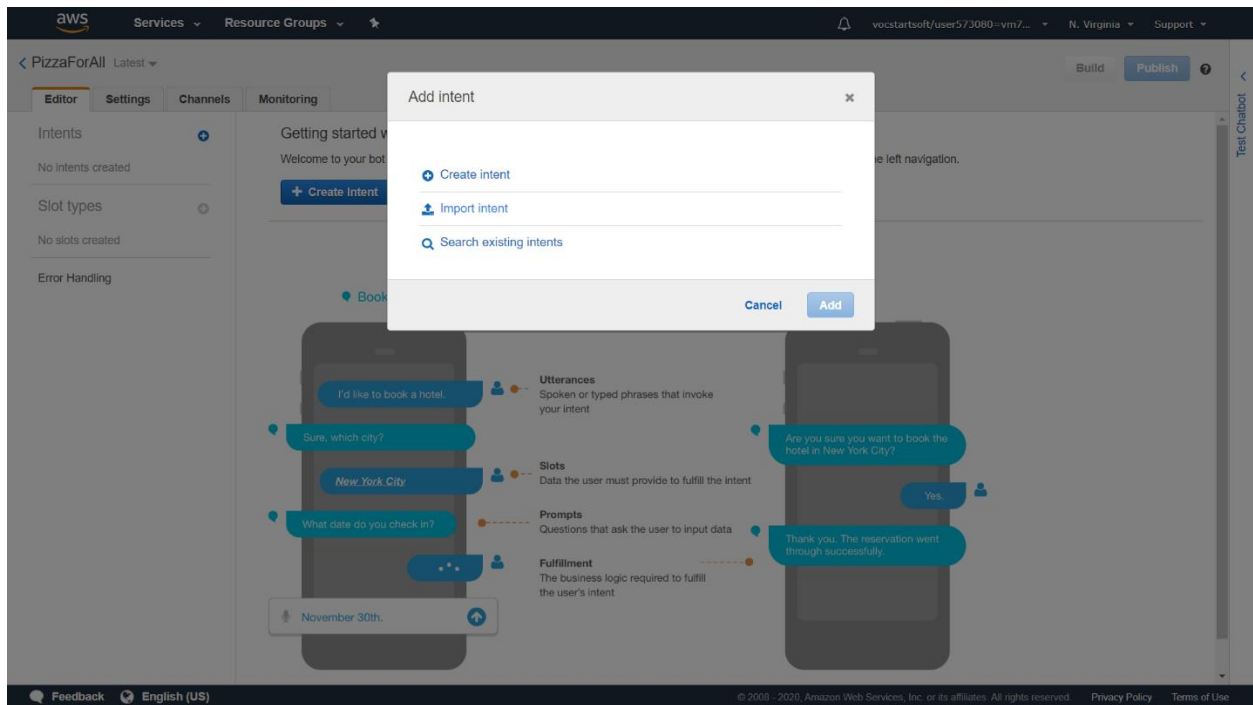


Figure 15: Intent Creation

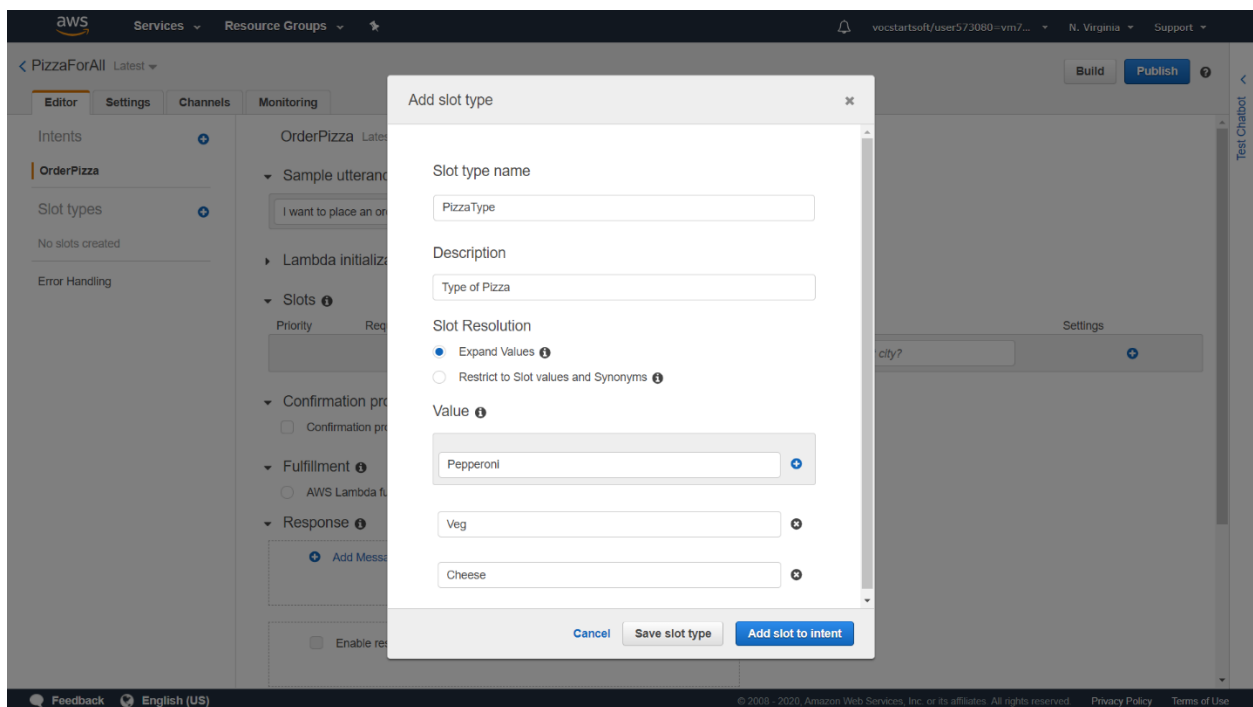


Figure 16: Adding slot to intent

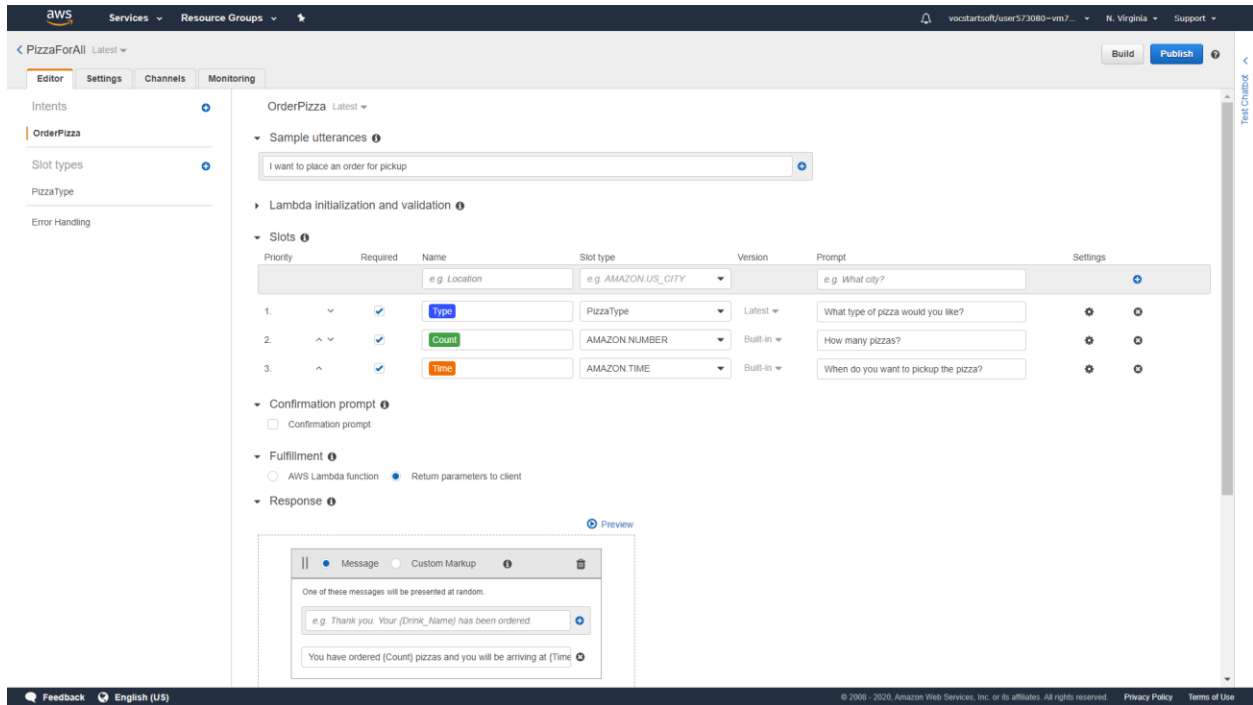


Figure 17: Adding Utterance, Slots, Responses for Order Pickup

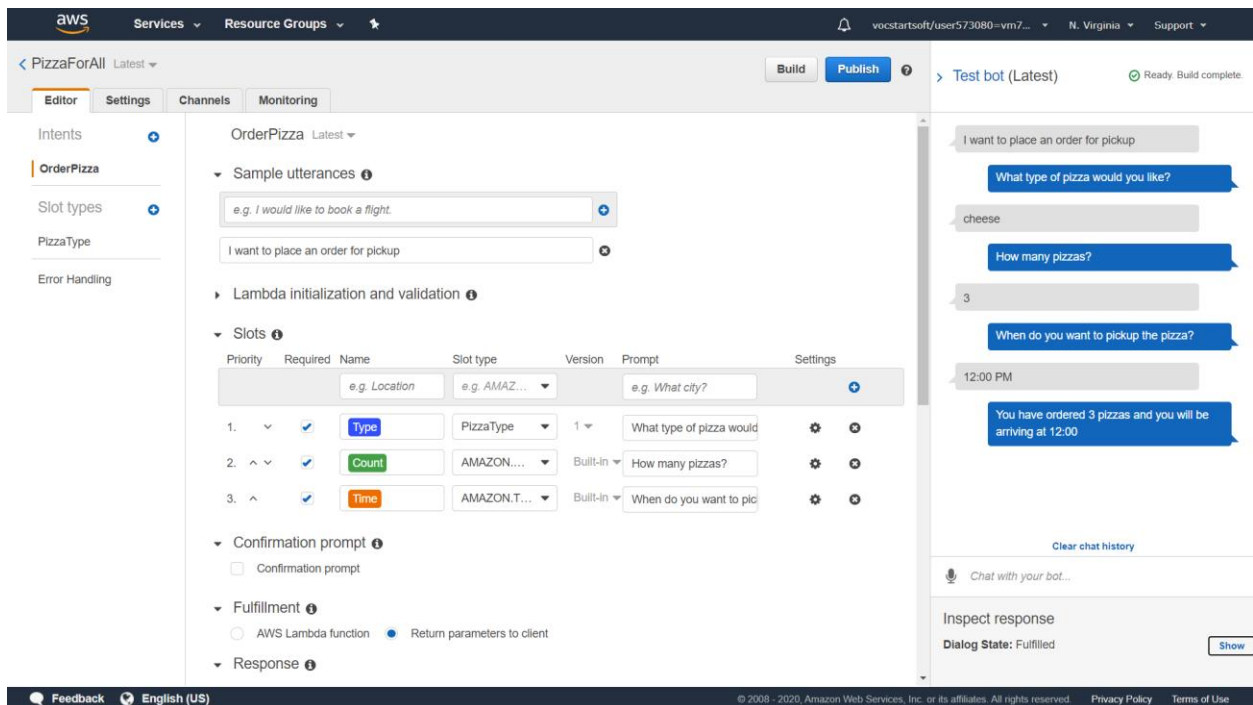


Figure 18: Testing bot for order pickup

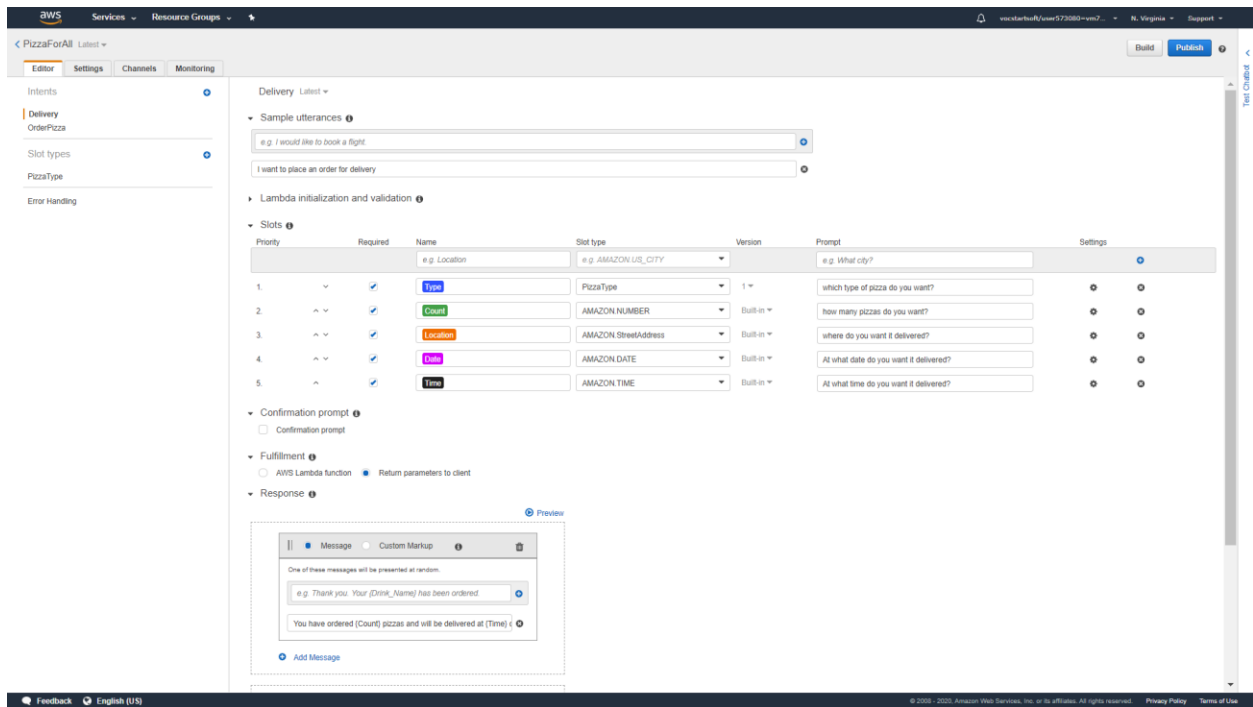


Figure 19: Adding Utterance, Slots, Response for Order Delivery

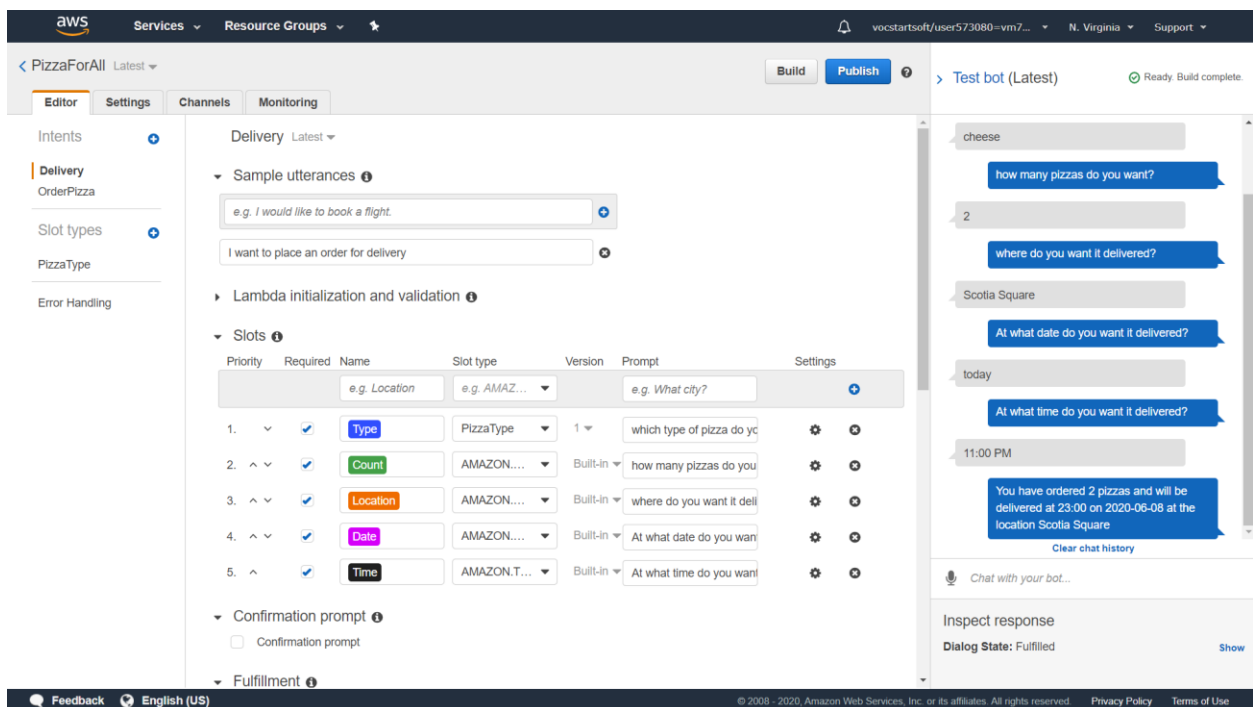


Figure 20: Testing bot for Order Delivery



## References

- [1]"Amazon Lex – Build Conversation Bots", *Amazon Web Services, Inc.*, 2020. [Online]. Available: <https://aws.amazon.com/lex/>. [Accessed: 08- Jun- 2020]
- [2]"Step 1: Create an Amazon Lex Bot (Console) - Amazon Lex", *Docs.aws.amazon.com*, 2020. [Online]. Available: <https://docs.aws.amazon.com/lex/latest/dg/gs-bp-create-bot.html>. [Accessed: 08- Jun- 2020].
- [3]"Cloud Run: Container to production in seconds | Google Cloud", *Google Cloud*, 2020. [Online]. Available: <https://cloud.google.com/run>. [Accessed: 08- Jun- 2020]
- [4]"Container Registry | Google Cloud", *Google Cloud*, 2020. [Online]. Available: <https://cloud.google.com/container-registry>. [Accessed: 09- Jun- 2020]
- [5]"Pushing and pulling images | Container Registry Documentation", *Google Cloud*, 2020. [Online]. Available: <https://cloud.google.com/container-registry/docs/pushing-and-pulling>. [Accessed: 09- Jun- 2020]
- [6]"Deploy and run a container with Cloud Run on Node.js", *Codelabs.developers.google.com*, 2020. [Online]. Available: <https://codelabs.developers.google.com/codelabs/cloud-run-hello/#0>. [Accessed: 09- Jun- 2020].