CSCI-5410: ASSIGNMENT2

Pratheep kumar Manoharan
B00837436
10-Jun-2020

Contents

Figures

Part A-Containerized Application Development

Objective

To develop serverless containerized meeting application that stores and retrieves data from MySQL database.

Technologies Used

The technologies used in this application are captured in the following grid.

S No	Application Section	Technology Used	Deployed Location
1	Database	MySQL	Cloud SQL
2	Backend	NodeJS with ExpressJS	Cloud Run
3	Frontend	React Framework	Cloud Run

Database

A new Cloud SQL instance was created for this application. The sequence of steps followed to create the database instance are as follows:

Step 1: Login to Google Cloud Platform (GCP). Click on the SQL service in the main menu as shown in image 1.

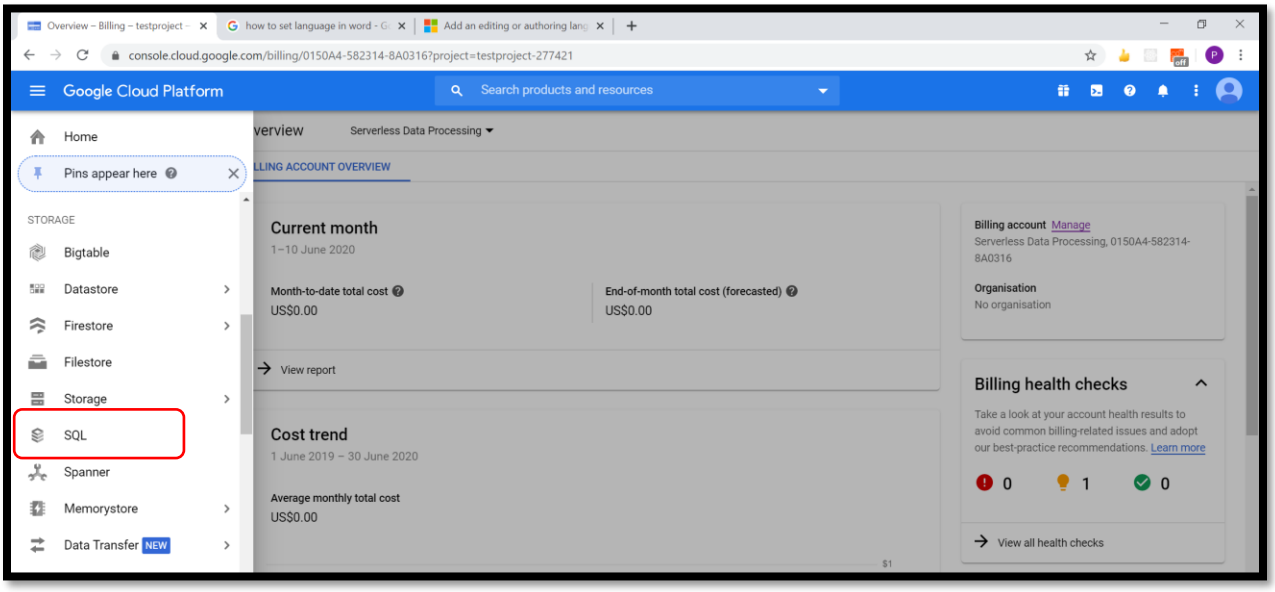


Figure 1: Click on SQL

Step 2: As per assignment requirement, select the MySQL database. Figure 2 highlights the MySQL database inside Cloud SQL.

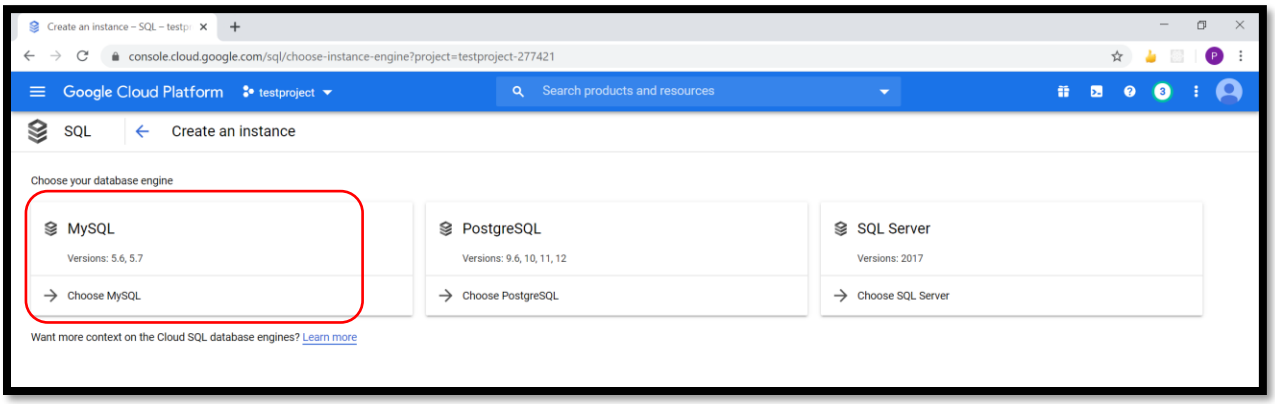


Figure 2: Select MySQL database

Step 3: Create the instance with basic details as displayed in figure 3. The important detail that need to be noted while creating the instance is to provide access to ports and IP address of the containers accessing the SQL instance.

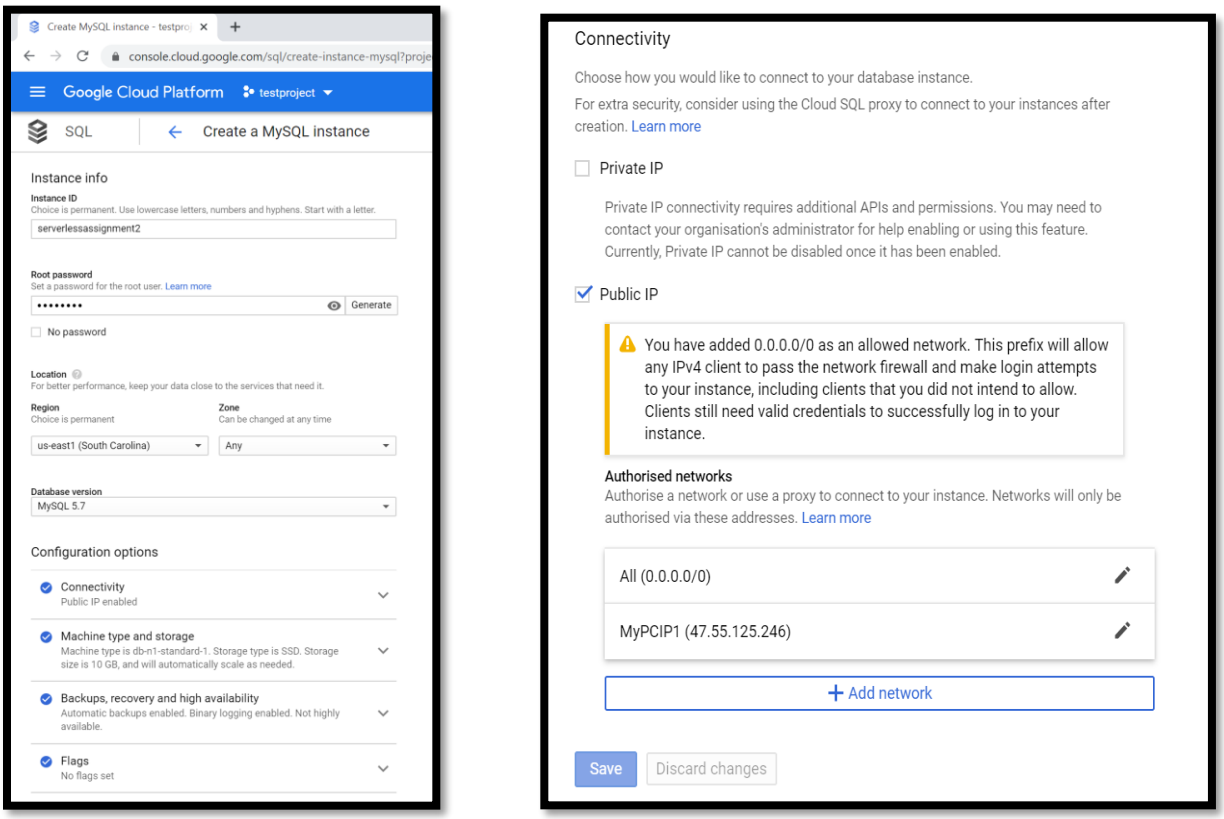


Figure 3: Creation of Instance

Step 4: Image 4 shows the created instance in the Cloud SQL page.

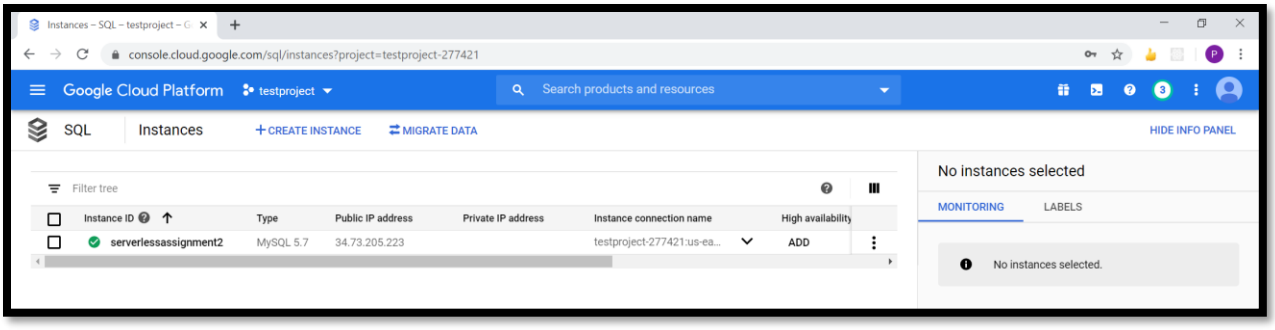


Figure 4: Created SQL Instance

Step 6: Connect to the MySQL instance from MySQL workbench. Figure 5 shows the success message after getting connected to the SQL instance.

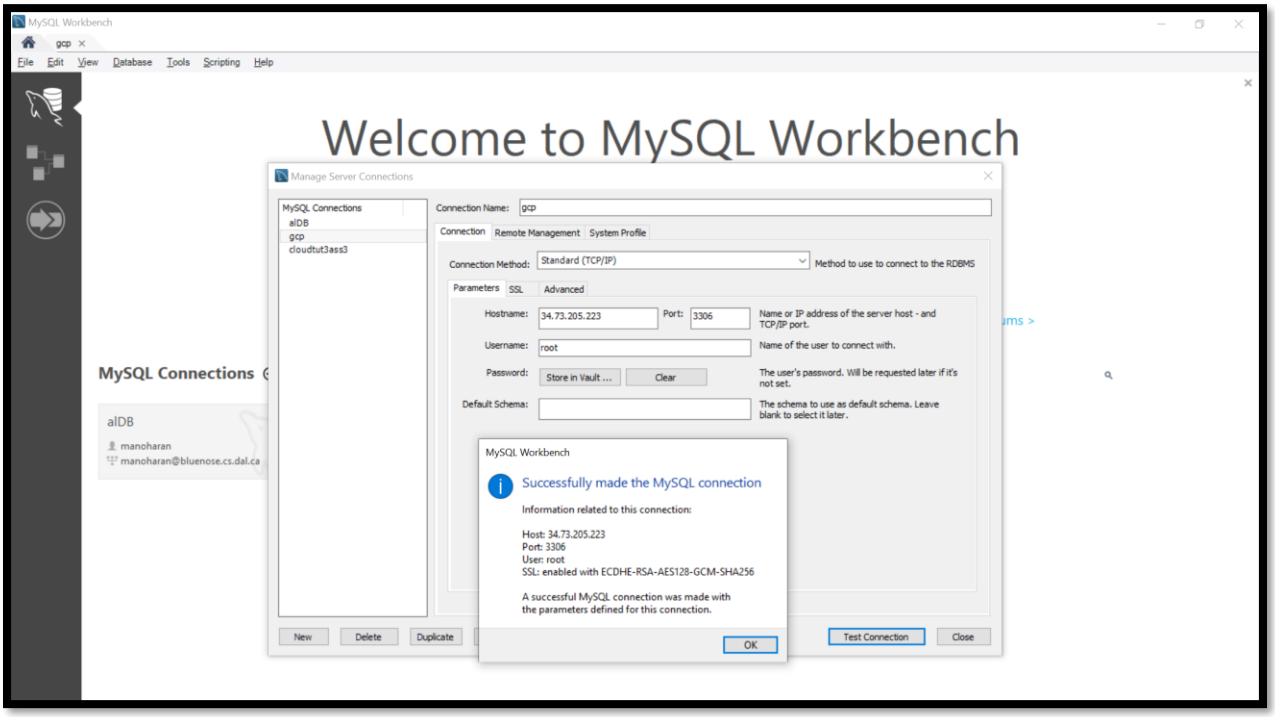


Figure 5: MySQL workbench connection

Step 7: Execute the scripts attached in Appendix 1 in the instance as shown in figure 6.

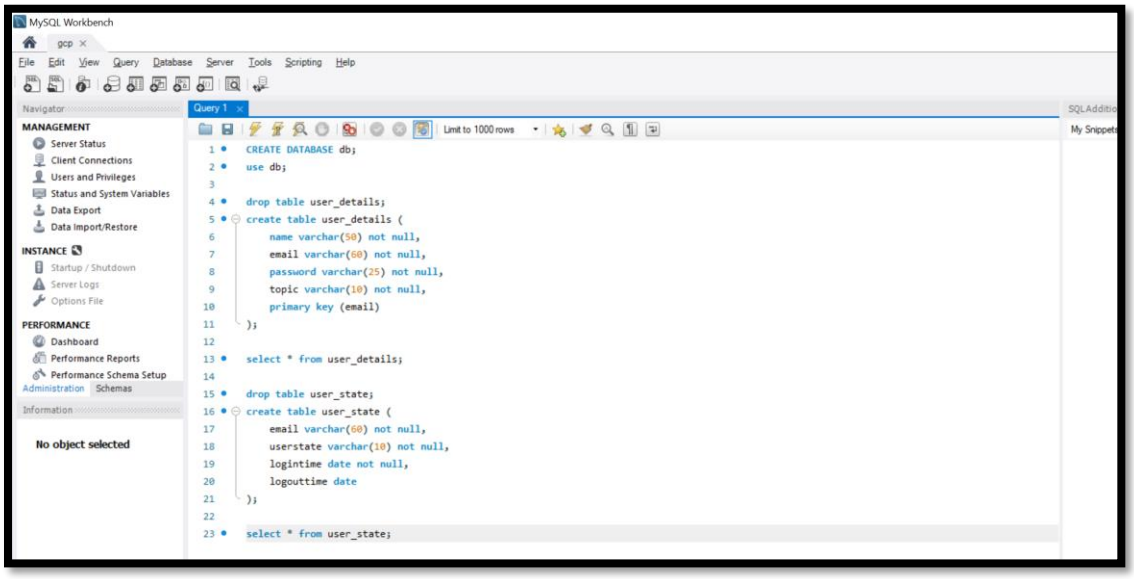


Figure 6: Scripts

Container 1: Registration

The registration container is the backend container responsible for handling the registration page requests of the application. The user will be providing the registration details in the frontend. The container will be called after validating the registration details provided by user. Among all the details, email address should be unique and if an existing email address is used by another user for registration, the application will not allow. The email address provided by the user will be validated with the email addresses already in database. After all validations, the details will be persisted in the database. The codebase of container 1 is added in Appendix 2. The task flow to deploy the codebase into the GCP Cloud Run are as follows.

Step 1: Develop an app to capture the post request from frontend.

Step 2: Add validations to check the data and persist the data in database.

Step 3: Containerize the codebase in local and test the same.

Step 4: Build the image and upload the image to the GCP container registry. Figure 7 captures the command line interface building container 1 image.

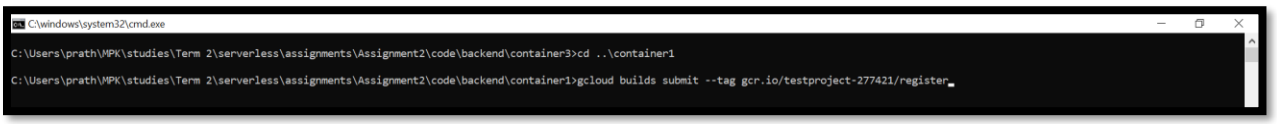


Figure 7: Build GCP image of container 1

Step 5: Figure 8 shows the build success message.

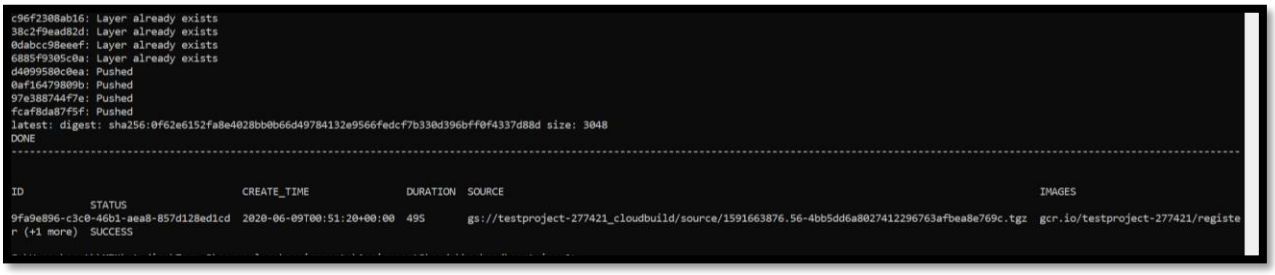


Figure 8: Build success Container 1

Step 6: Deploying the container image into the cloud run. Create a cloud run container as shown in images 9.

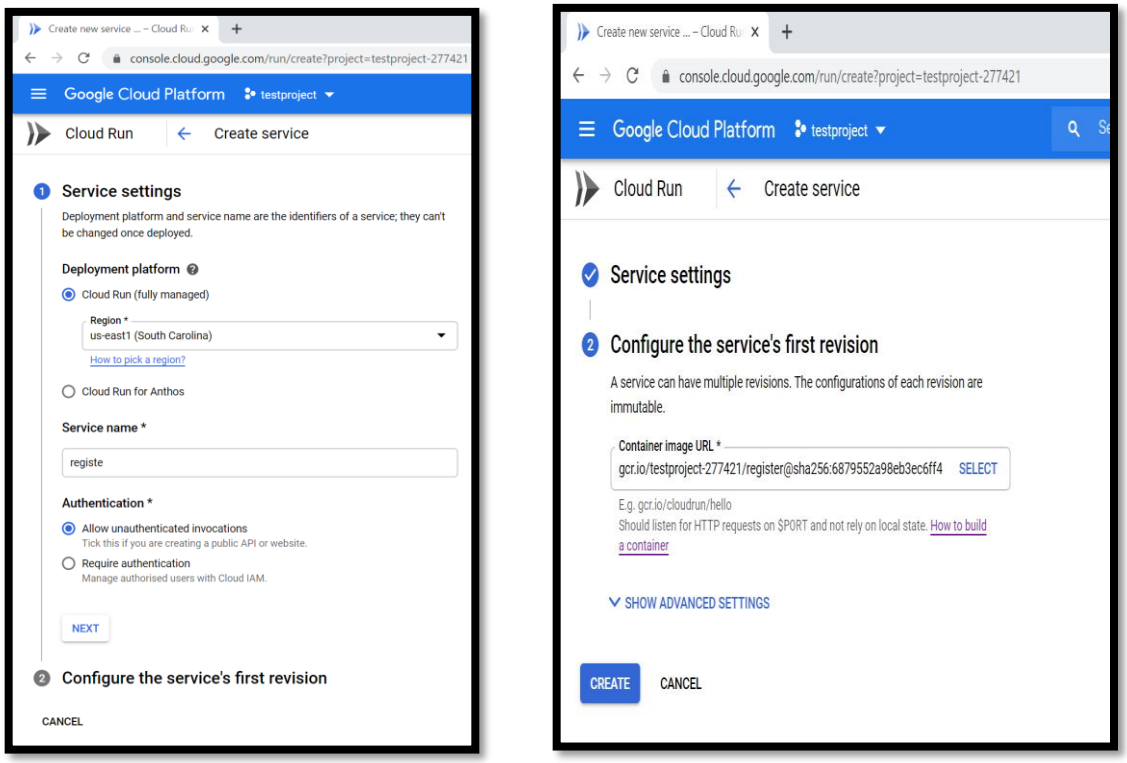


Figure 9: Container 1 creation

Step 7: Container will be created as shown in image 10 and to check the deployment a home page is created in the container as captured in image 11.

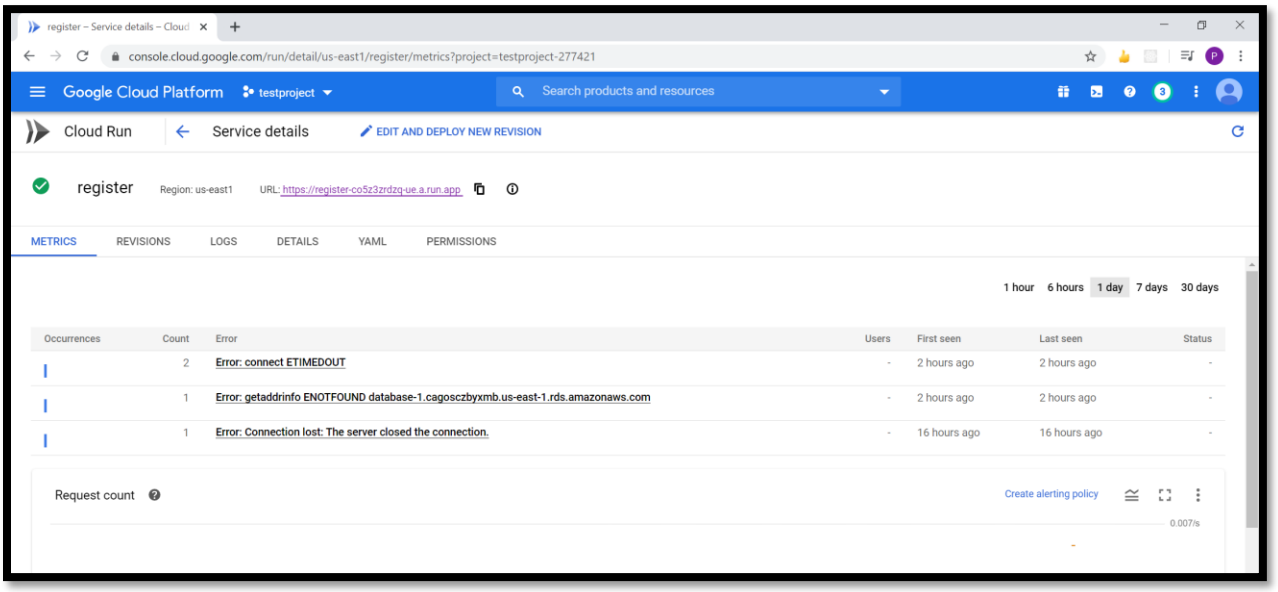


Figure 10: Container 1

Figure 11: Container 1 home page

Container 2: Login

The login container is responsible for handling the login requests and force login requests of the application. The user must be registered and provide the correct credentials in the login page to login the application. The container will be called to validate the credentials provided by user. The credentials provided by the user will be validated against data in the database. After all the validations, the system will take the user to the home page. The codebase of container 2 is added in Appendix 3. The steps to deploy the container 2 are as follows.

Step 1: Build a codebase to capture the post request from frontend.

Step 2: Add validations to check the data against the data in database and update appropriately.

Step 3: Containerize the codebase in local and test the same.

Step 4: Build the image and upload the image to the GCP container register. Figure 12 captures the command line building of container 2 image.

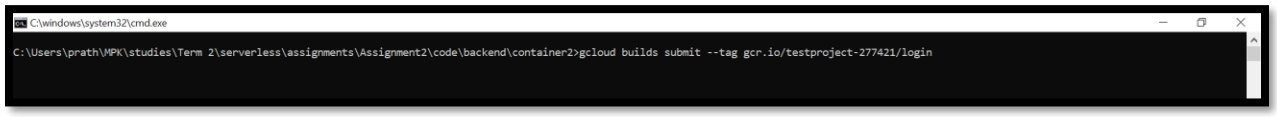


Figure 12: Build GCP image of container 2

Step 5: Figure 13 shows the build success message.

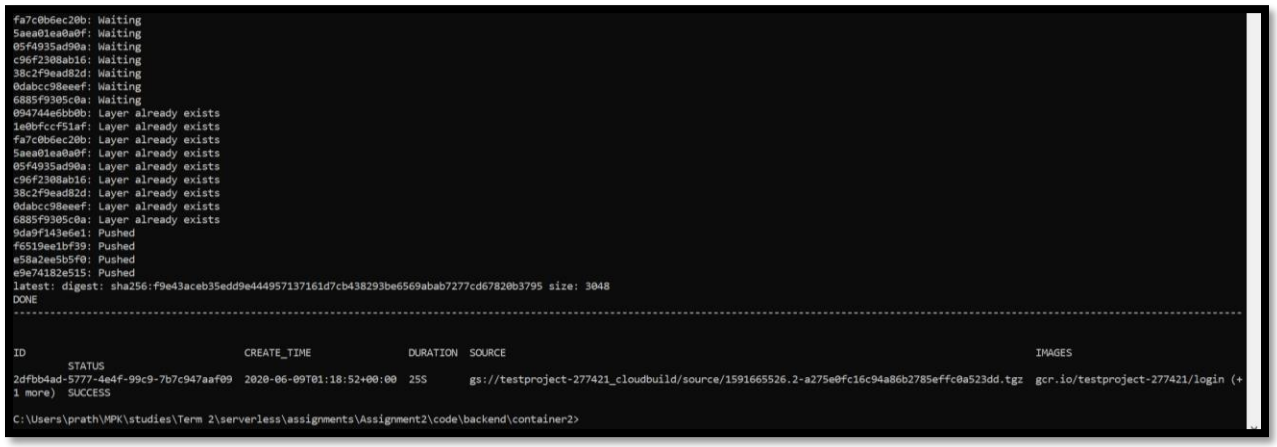


Figure 13: Build success Container 2

Step 6: Deploying the container image into the cloud run. Create a cloud run container as shown in images 14 and 15.

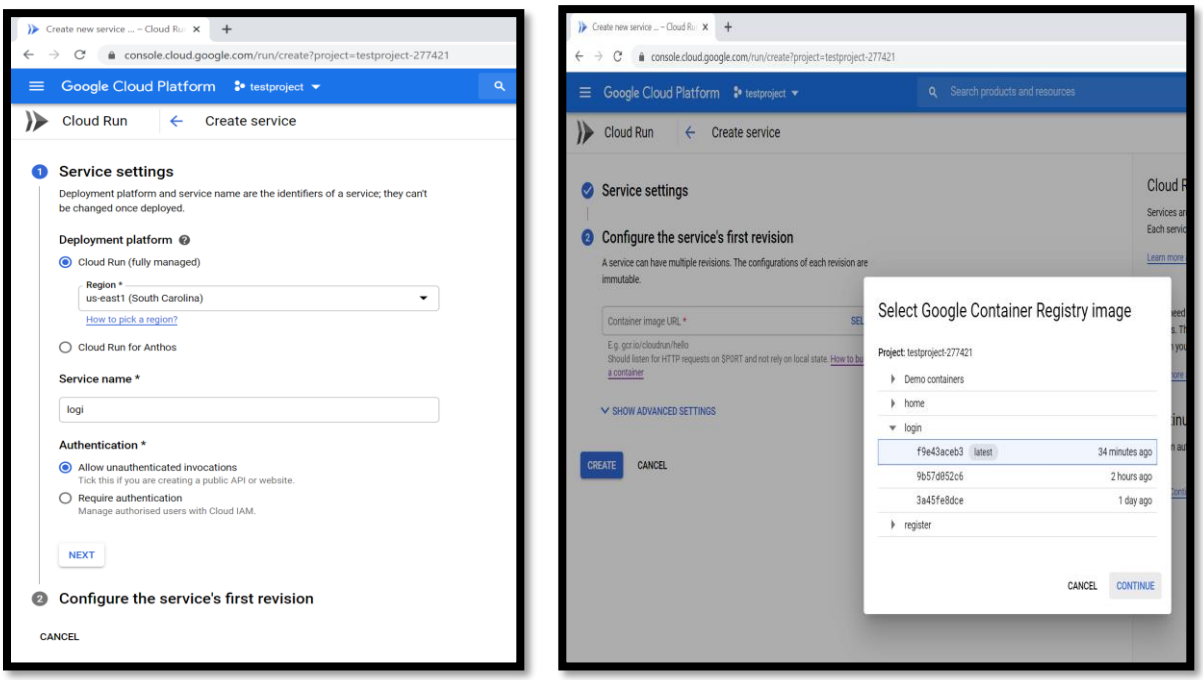


Figure 14: Container 2 creation

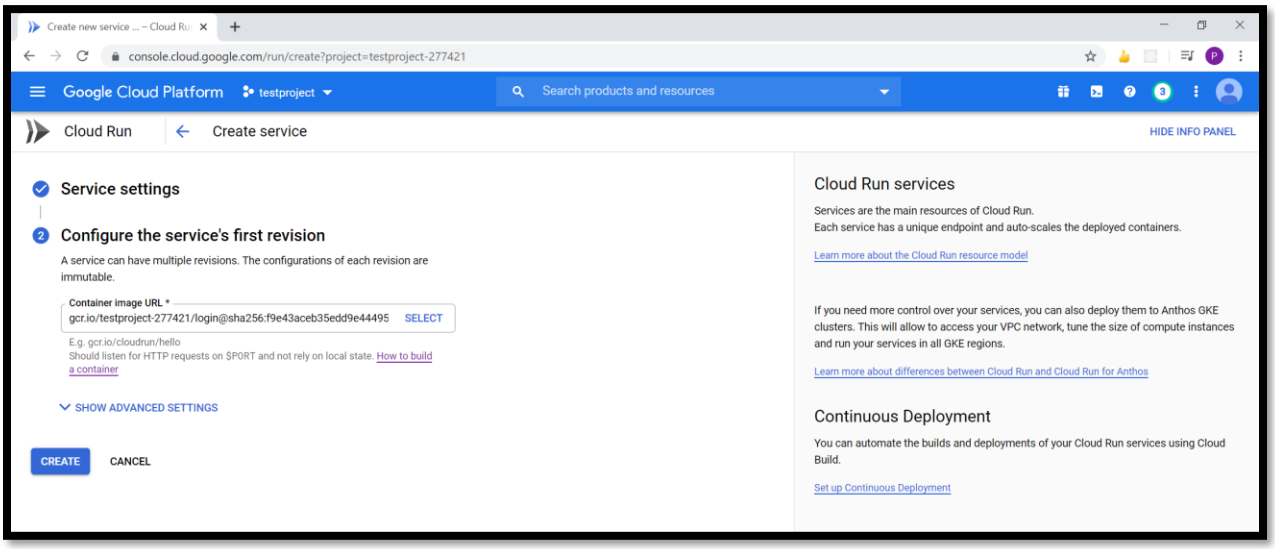


Figure 15: Container 2 configuration

Step 7: Container will be created as shown in image 16 and to check the deployment a home page is created in the container, which is tested, and results are captured in image 17.

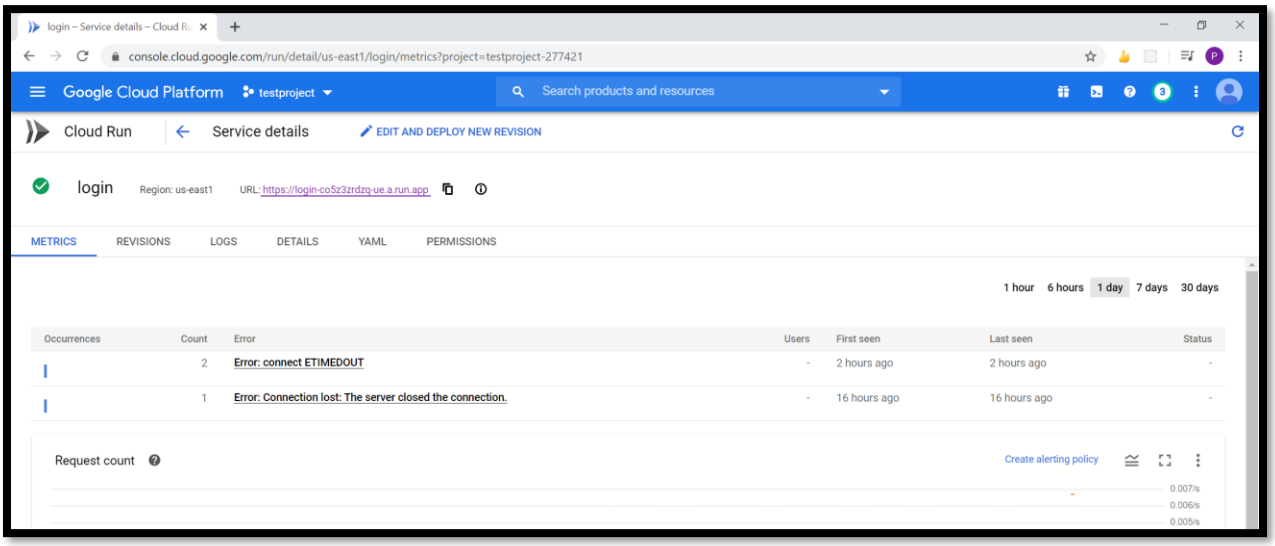


Figure 16: Container 2

Figure 17: Container 2 home page

Container 3: Home

The home container is responsible for handling the home page requests to fetch the online user details and logout. The codebase of container 3 is added in Appendix 4. The steps followed to deploy the container 3 are captured below.

Step 1: Build a codebase to capture the get request from frontend.

Step 2: Add validations to check the data against the data in database and update appropriately.

Step 3: Containerize the codebase in local and test the same.

Step 4: Build the image and upload the image to the GCP container register. Figure 18 captures the command line building of container 3 image.

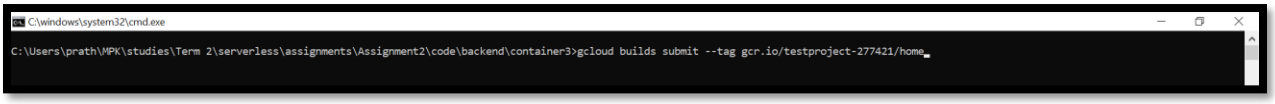


Figure 18: Build GCP image of container 3

Step 5: Figure 19 shows the build success message.

Figure 19: Build success Container 3

Step 6: Deploying the container image into the cloud run. Create a cloud run container as shown in images 20.

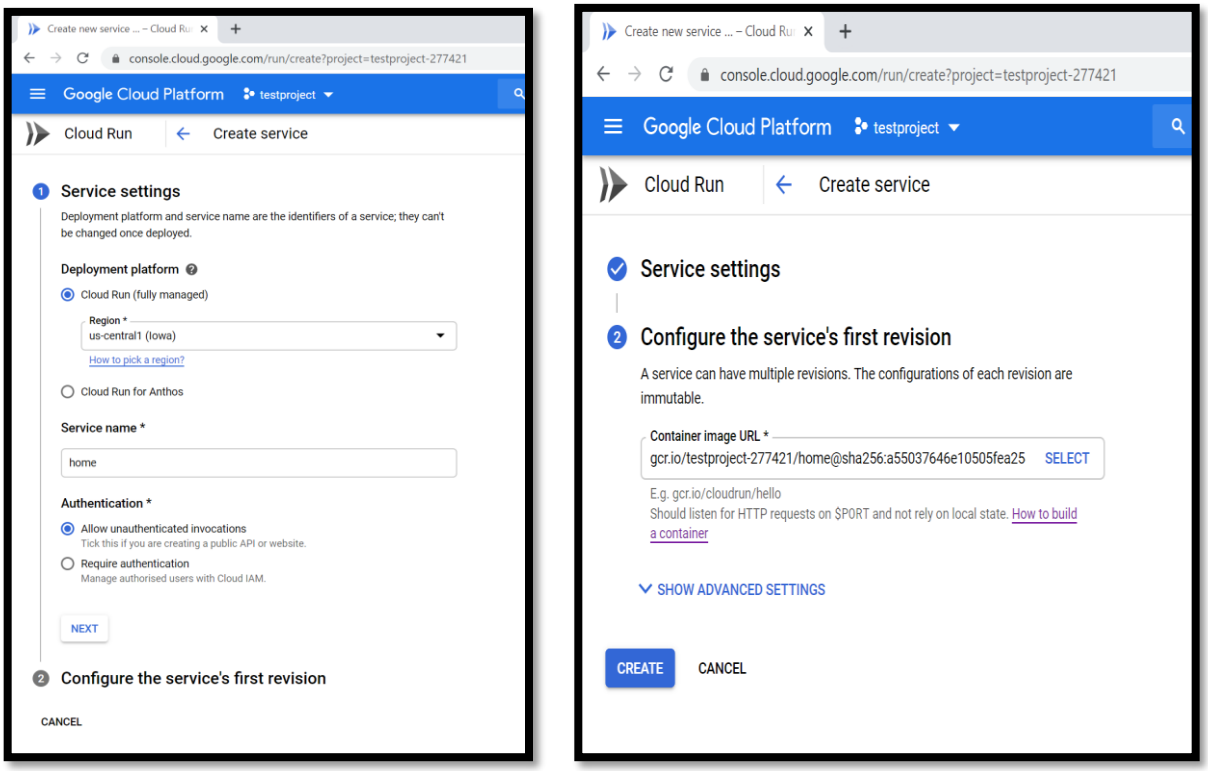


Figure 20: Container 3 creation

Step 7: Container will be created as shown in image 21 and to check the deployment a home page is created in the container, which is tested, and results are captured in image 22.

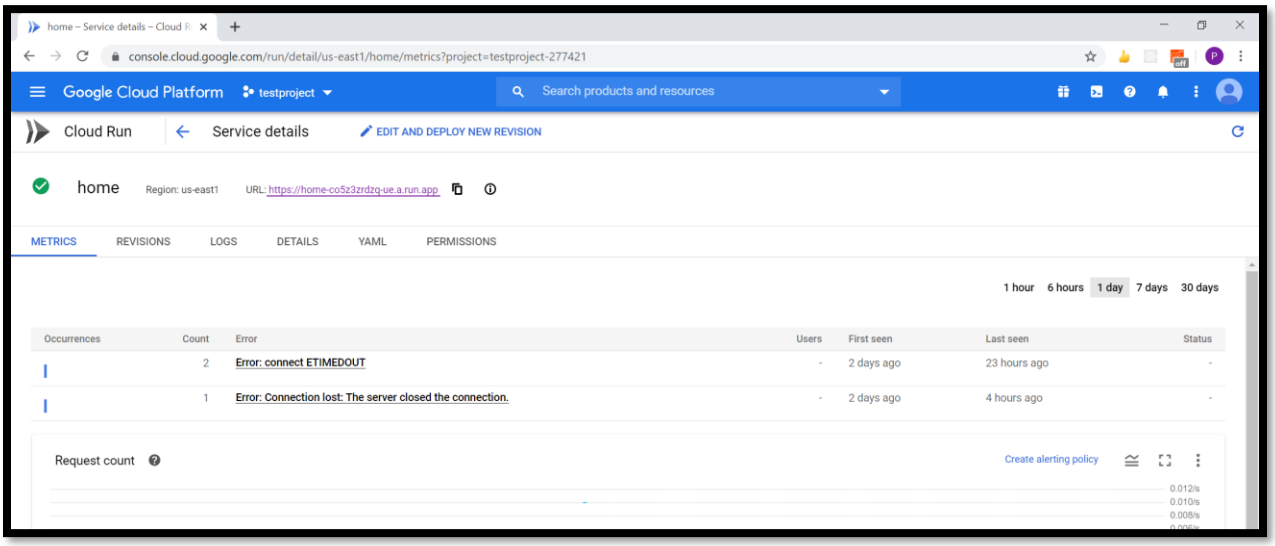


Figure 21: Container 3

Figure 22: Container 3 home page

Container 4: Frontend

The frontend container is responsible for the web pages of the application. The codebase of container 4 is added in Appendix 5. The sequence of steps followed to containerize the frontend are as described below.

Step 1: Build a codebase with react framework to show the web pages.

Step 2: Add validations to check the user data and call the backend API.

Step 3: Containerize the codebase in local and test the same.

Step 4: Build the image and upload the image to the GCP container register. Figure 23 captures the command line interface building of container 4 image.

Figure 23: Build GCP image of container 4

Step 5: Figure 24 shows the build success message.

Figure 24: Build success Container 4

Step 6: Deploying the container image into the cloud run. Create a cloud run container as shown in image 25.

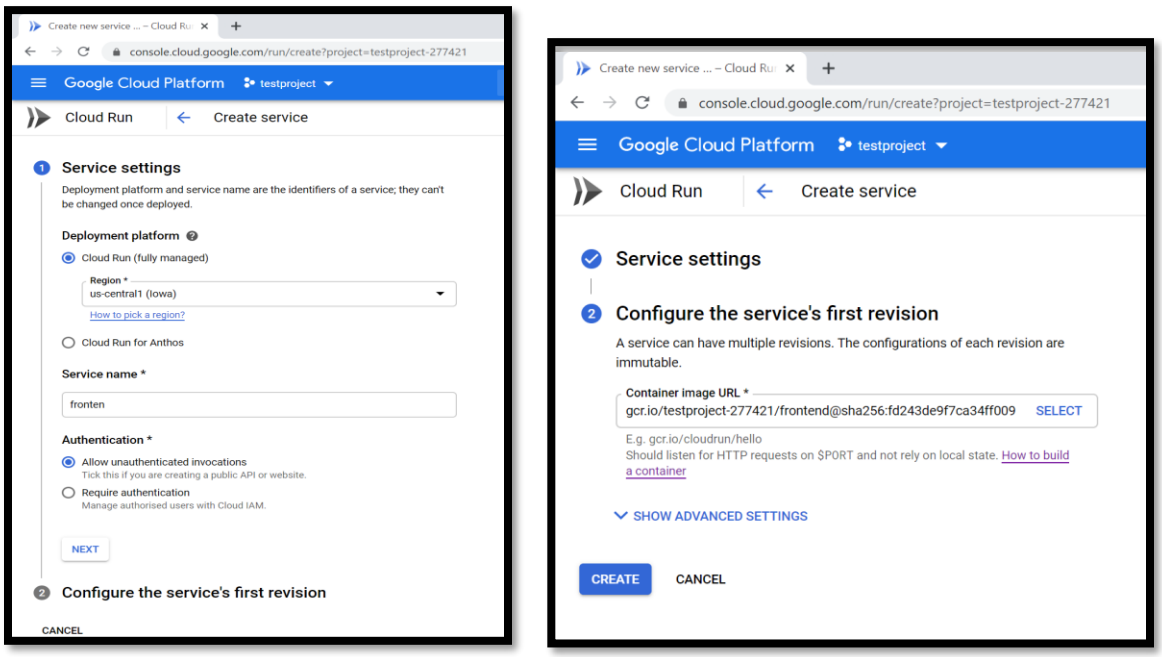


Figure 25: Container 4 creation

Step 7: Container will be created as shown in image 26 and the application is tested after deployment and the result is captured in image 27.

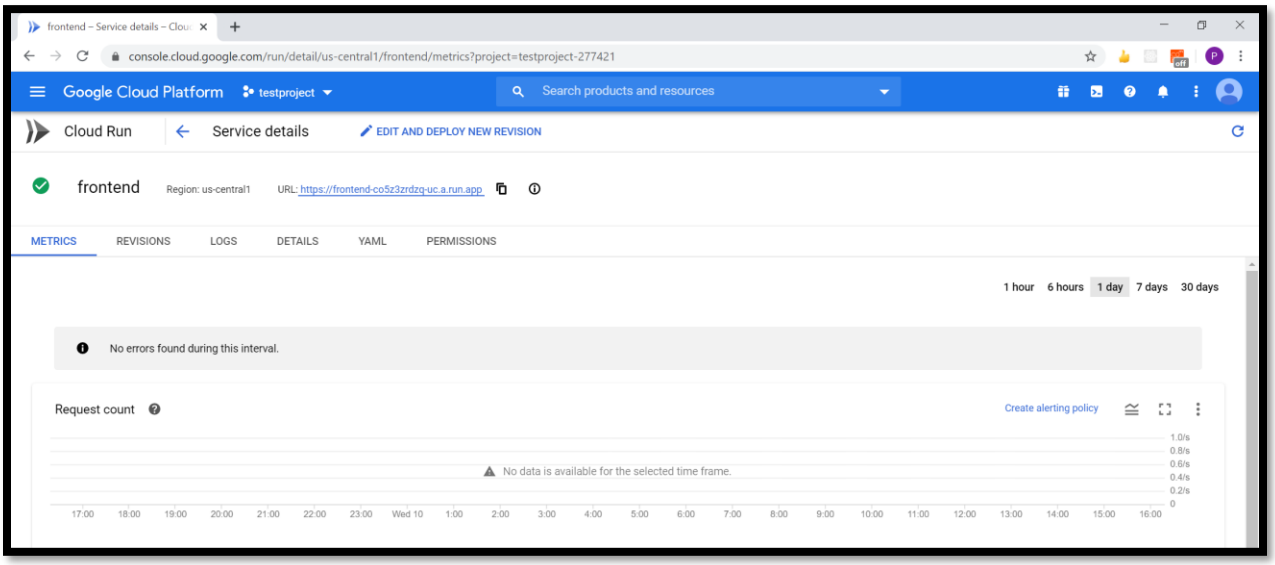


Figure 26: Container 4

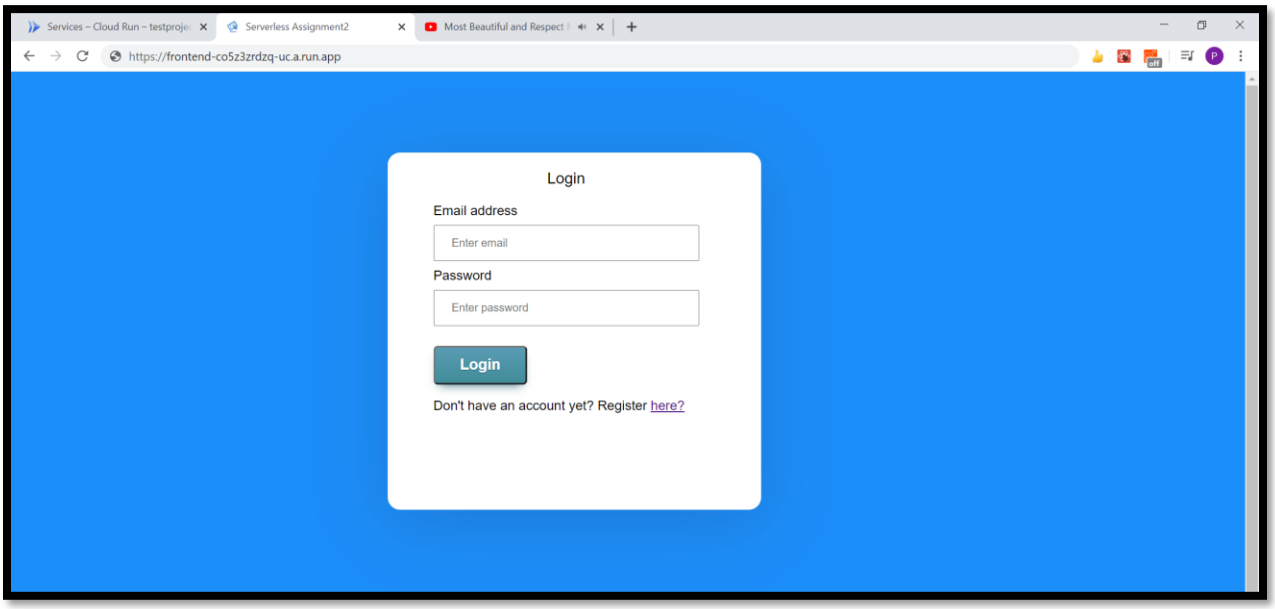


Figure 27: Container 4 home page

GCP

Figure 28 shows the docker images uploaded into GCP container registry.

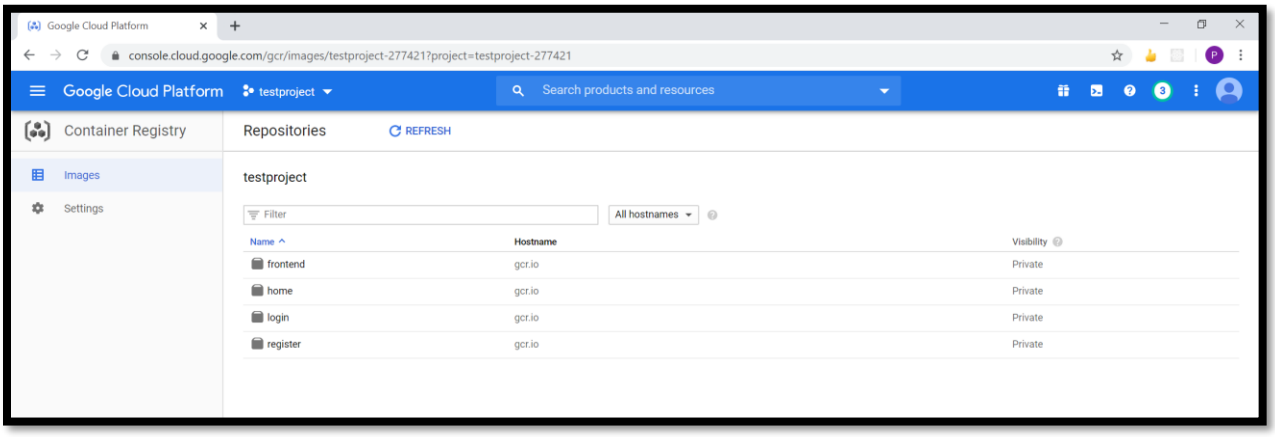


Figure 28: GCR images

Figure 29 shows the containers created in Cloud Run in GCP.

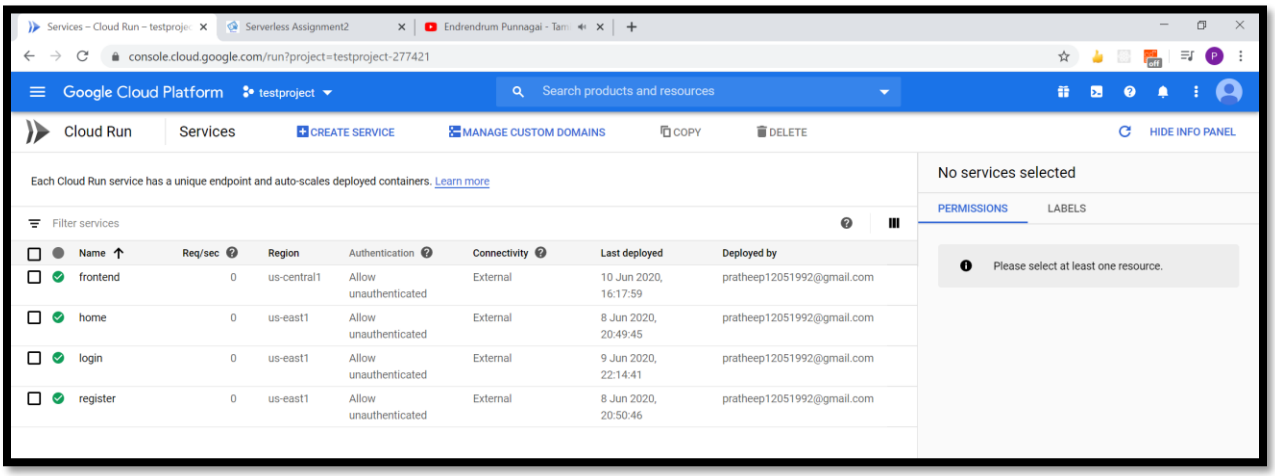


Figure 29: GCP Cloud Run Containers

Test Cases

The test cases prepared to test the application are added below.

S No	Test scenario	Expected result	Actual result
1	Launch the application with container 4 URL	Landing page should appear	pass
	Registration Page		
2	Click on register link to register	the system should route the user to the registration page	pass
3	Submit the registration page without any details filled.	form should not get submitted and should throw an error	pass
4	Submit the registration page without name.	form should not get submitted and should throw an error	pass
5	Submit the registration page with invalid email address.	form should not get submitted and should throw an error	pass
6	Give all valid details	form should get submitted and the system should render login page	pass
7	Give an existing email	form should not get submitted and should throw an error	pass
	Login Page		
8	Submit without any fields	form should not get submitted and should throw an error	pass
9	Give non existing user credentials	form should not get submitted and should throw an error	pass
10	Give valid user credentials	form should get submitted and system sshould render home page	pass
	Force login		
11	Login with a user, without logging out, try to do force login in next tab.	second attempt by the user should be prompted with force login button. Upon force login the user should be logged in.	pass
	Home Page		
12	Login with multiple users and check the online users in database. Logout and check whether the online users are modified accordingly.	The users should be same. After logout the user should not be shown.	Pass

Testing

Scenario 1: Launch the application with container 4 URL. Figure 30 displays the result of the test scenario 1.

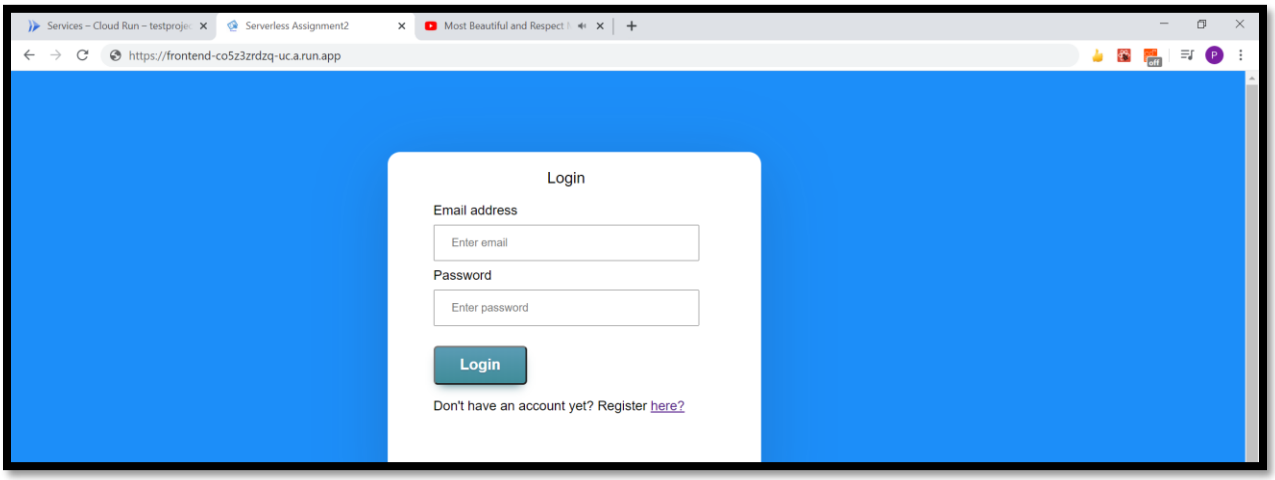


Figure 30: Test scenario 1

Scenario 2: Click on register link to register. The test scenario 2 result is shown in figure 31.

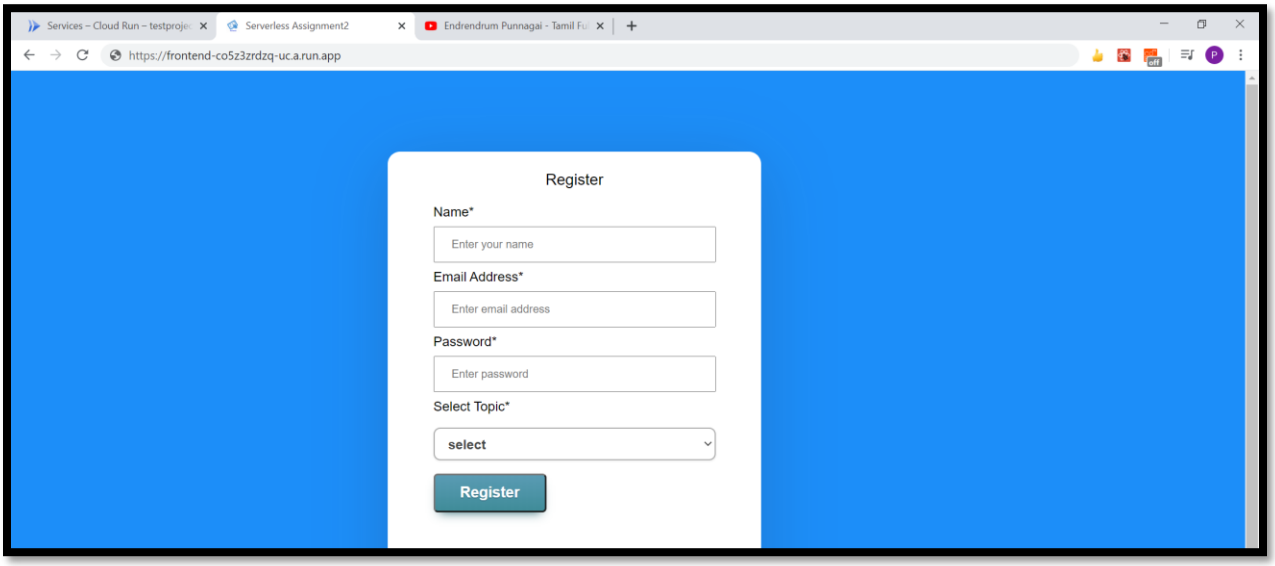


Figure 31: Test scenario 2

Scenario 3: Submit the registration page without any details filled. Figure 32 captures the result of scenario 3.

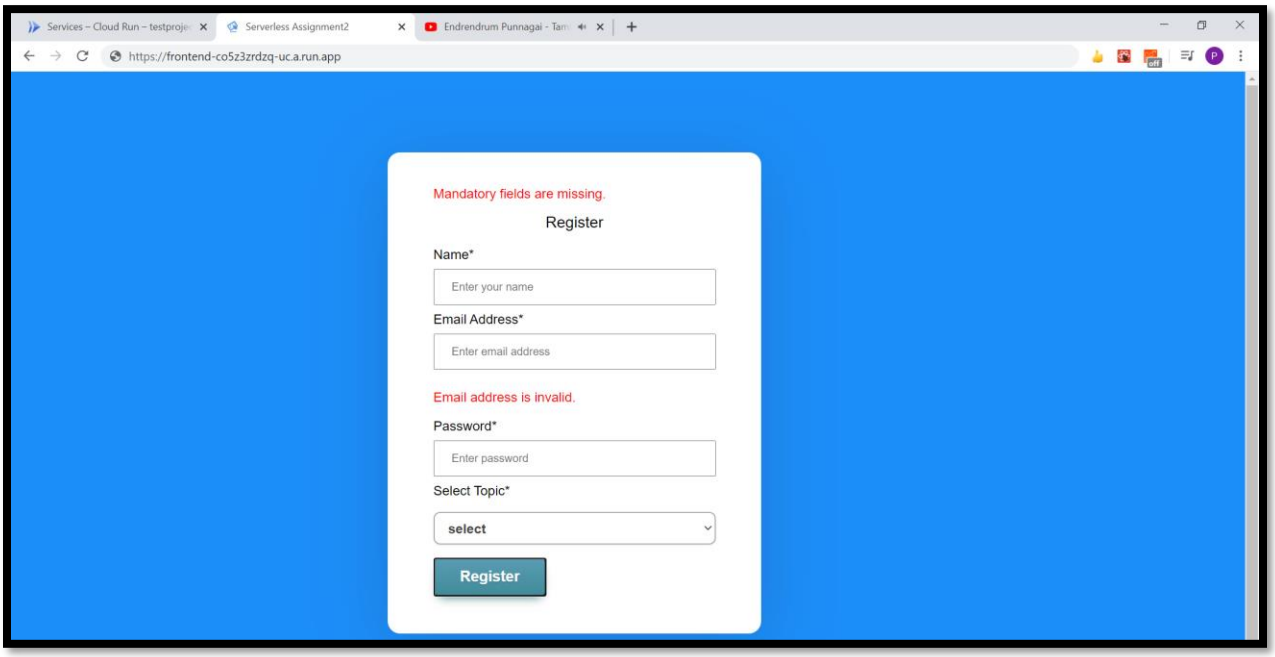


Figure 32: Test scenario 3

Scenario 4: Submit the registration page without name. Figure 33 captures the result of scenario 4. Similarly, all fields in the registration page are mandatory.

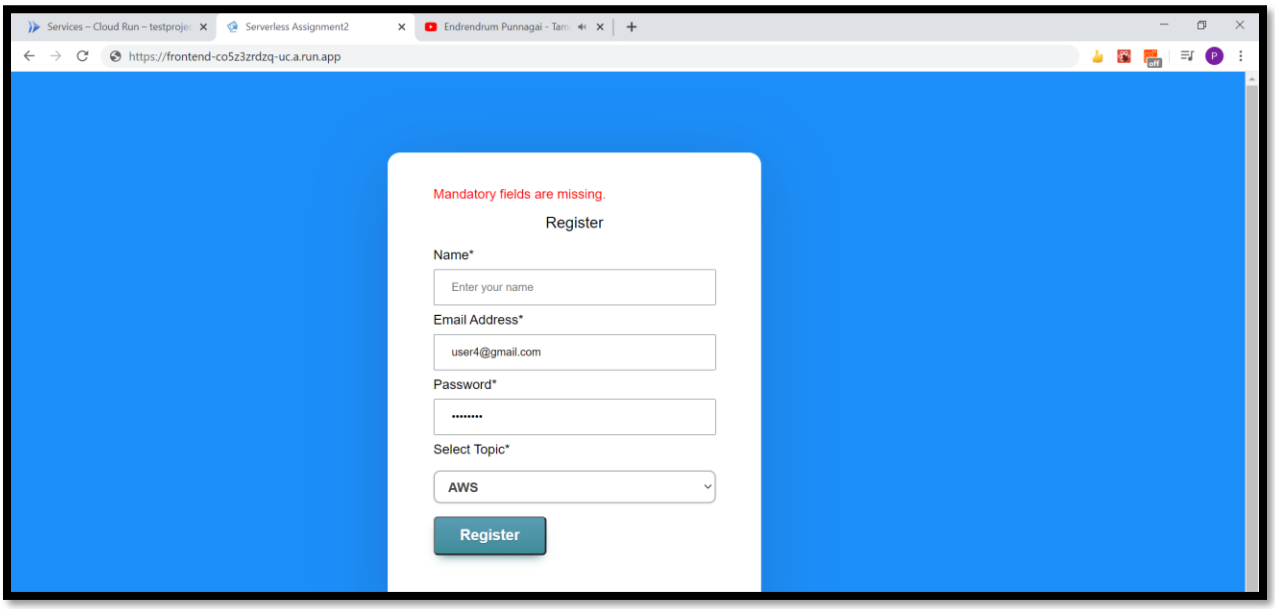


Figure 33: Test scenario 4

Scenario 5: Submit the registration page with invalid email address. Figure 34 captures the result of scenario 5.

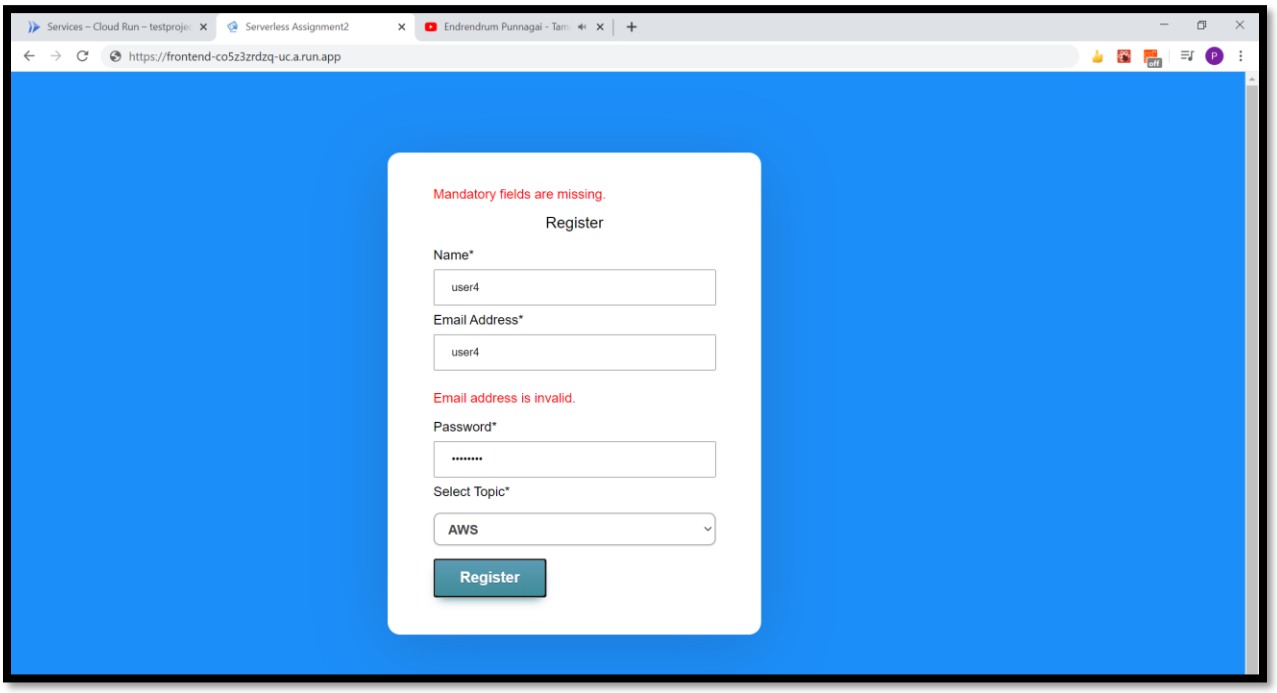


Figure 34: Test scenario 5

Scenario 6: Give all valid details. After submitting the details will be stored in the database in user details table. Figure 35 captures the result of scenario 6.

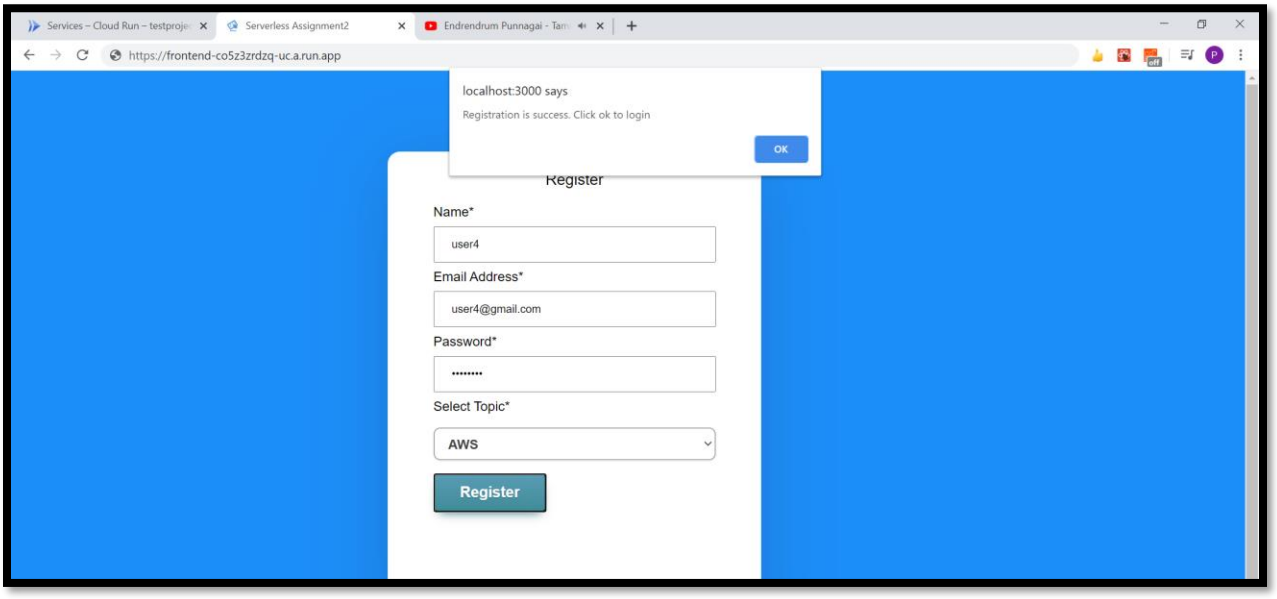


Figure 35: Test scenario 6

Figure 36 shows the user details added in database.

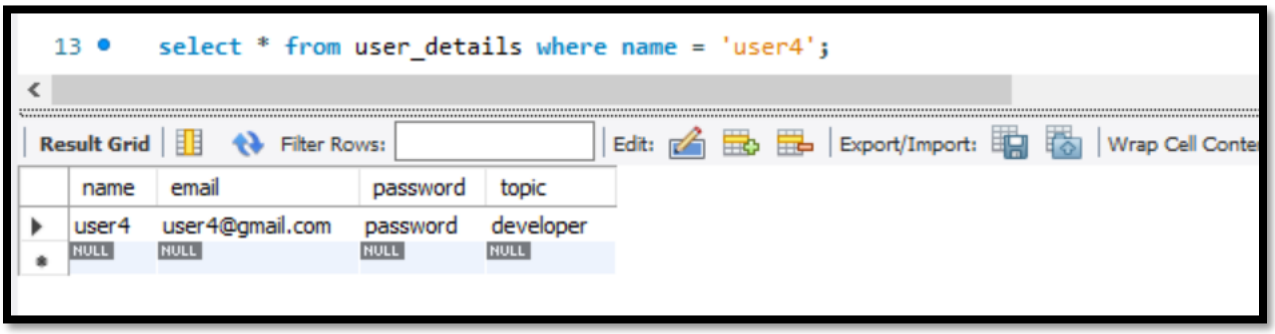


Figure 36: Register details in user

Scenario 7: Give an existing email. The web page will call the backend which in turn will check the details in database. Figure 37 captures the result of scenario 7.

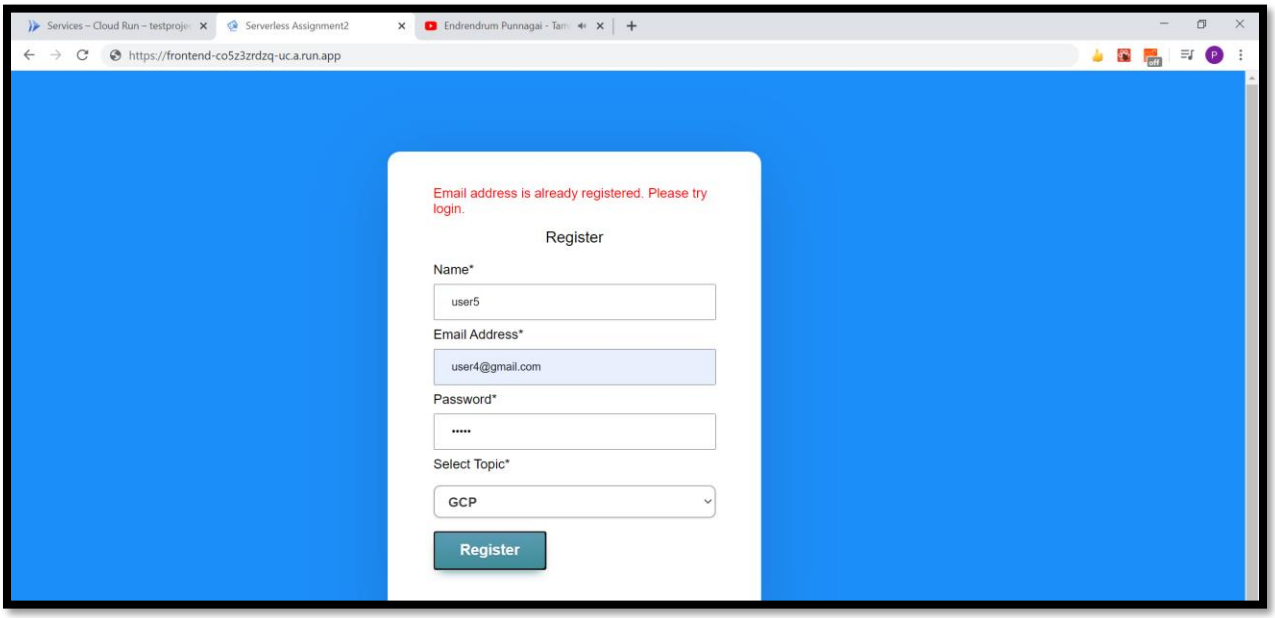


Figure 37: Test scenario 7

Scenario 8: Submit without any fields in login page. Figure 38 captures the result of scenario 8.

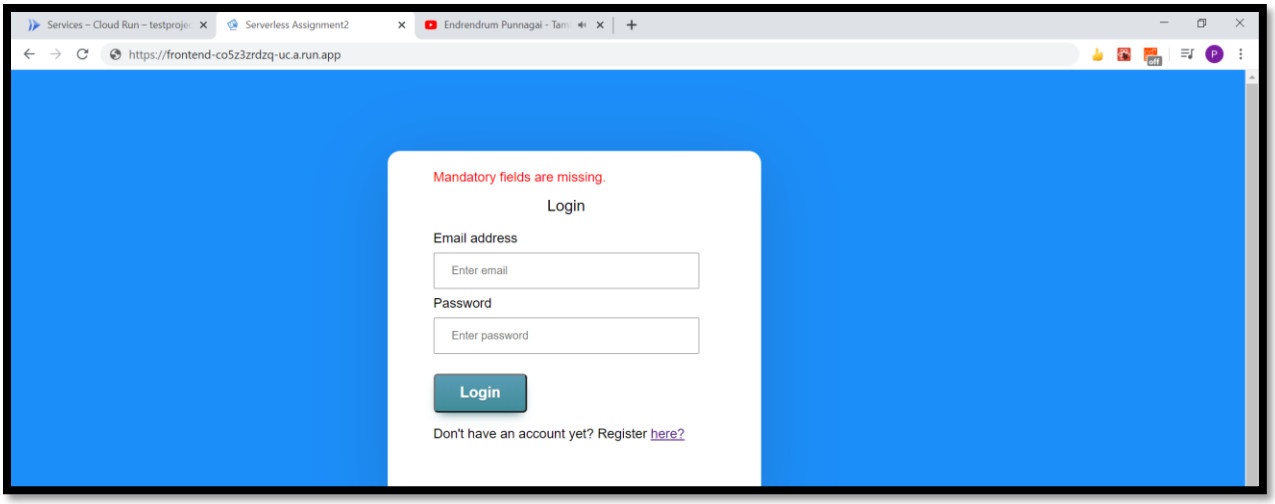


Figure 38: Test scenario 8

Scenario 9: Give non existing user credentials. Once the user clicks login, the web page will call the backend which in turn will call the database to check the database. Figure 39 captures the result of scenario 9.

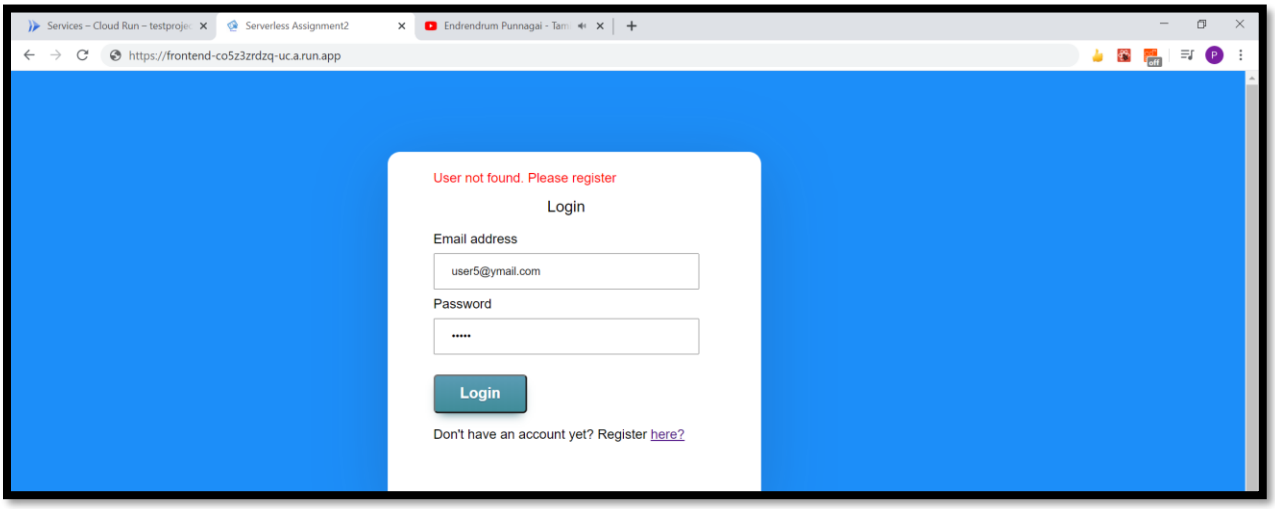


Figure 39: Test scenario 9

Scenario 10: Give valid user credentials to login. Once the user clicks the login, the web page will call the backend which in turn validates the data in database. Valid response will be sent from backend to

frontend. Based on the response the system navigates the user to home page. Figure 40 captures the result of scenario 10. Figure 41 confirms that no users are online from database side.

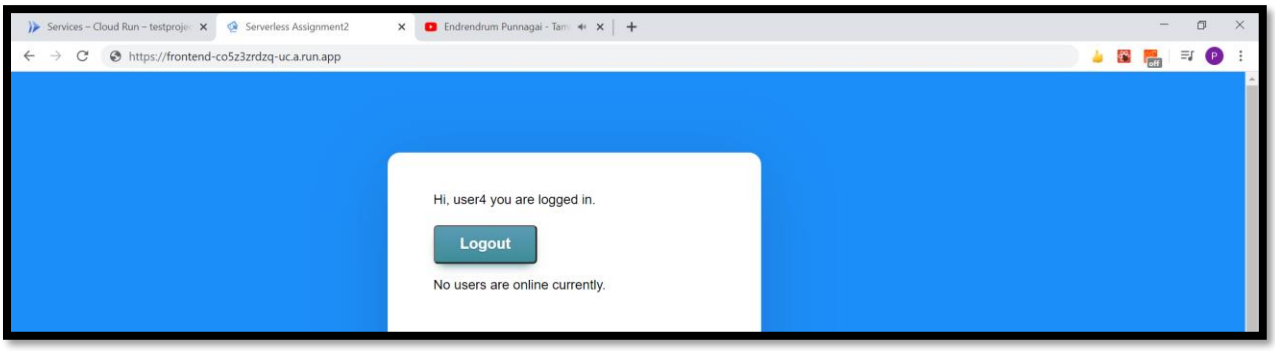


Figure 40: Test scenario 10

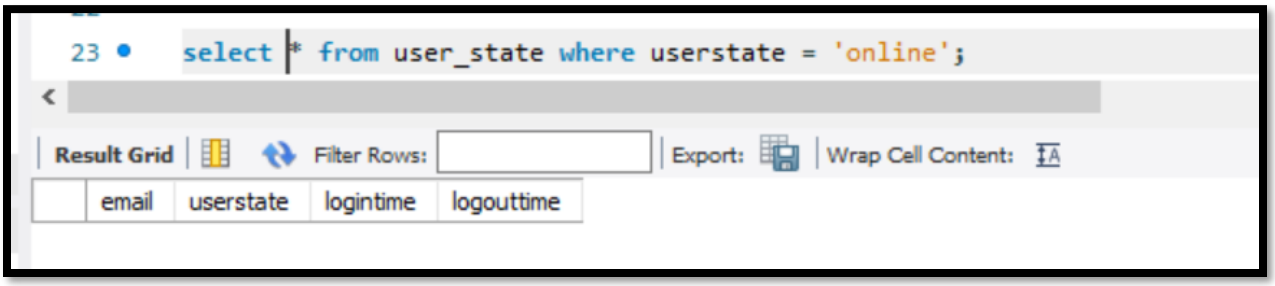


Figure 41: Database result of test scenario 10

Scenario 11: Login with a user, without logging out, try to do force login in next tab. Figures 42, 43, and 44 shows the test results of scenario 11.

Figure 42 shows the first login attempt with user2.

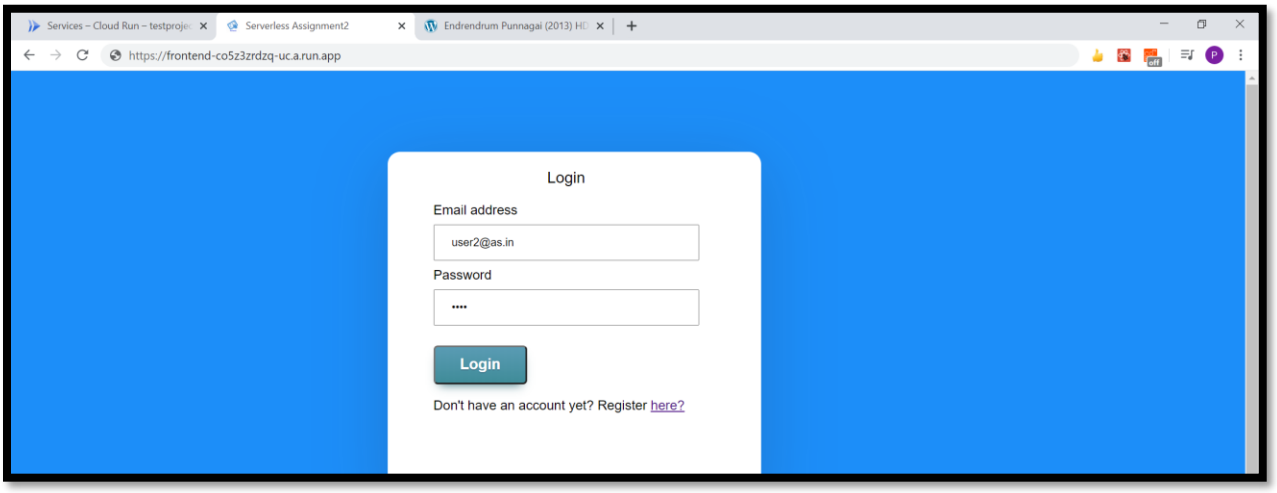


Figure 42: first login with user2

Figure 43 shows the home page of first login attempt.

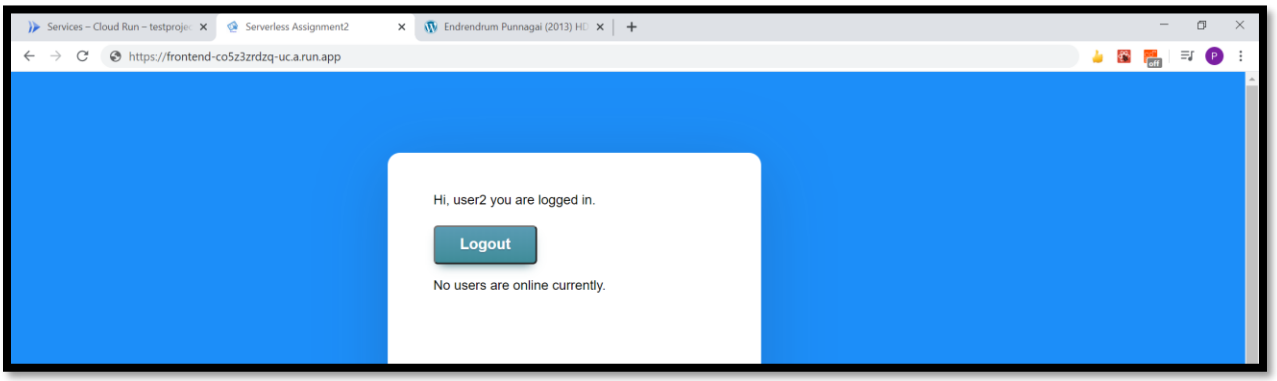


Figure 43: home page

Figure 44 shows the second login done by user2 in different tab.

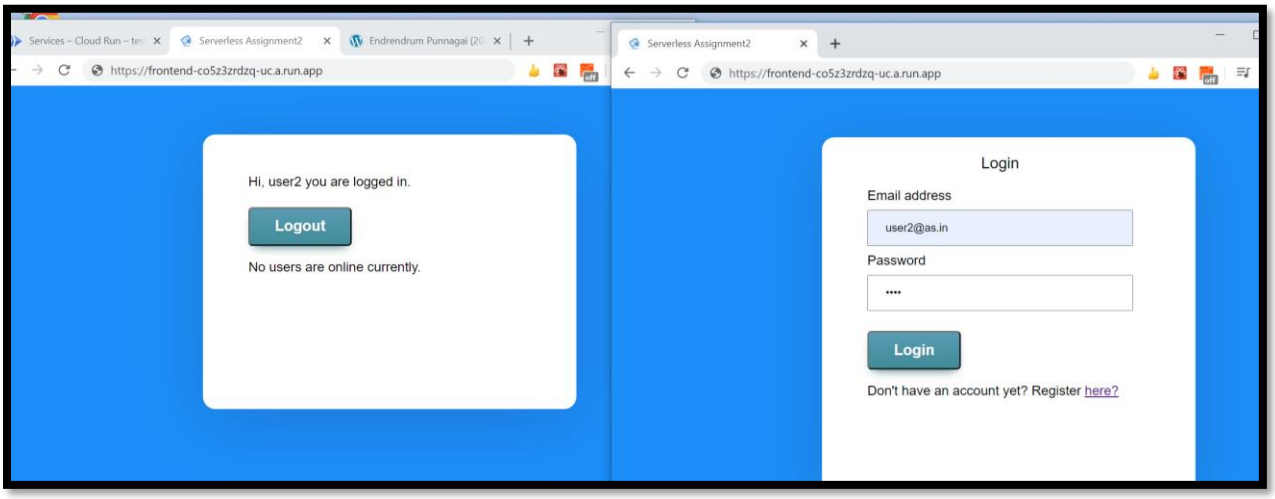


Figure 44: second login in different tab

Figure 45 shows the force login button since the user2 is already logged in in different tab.

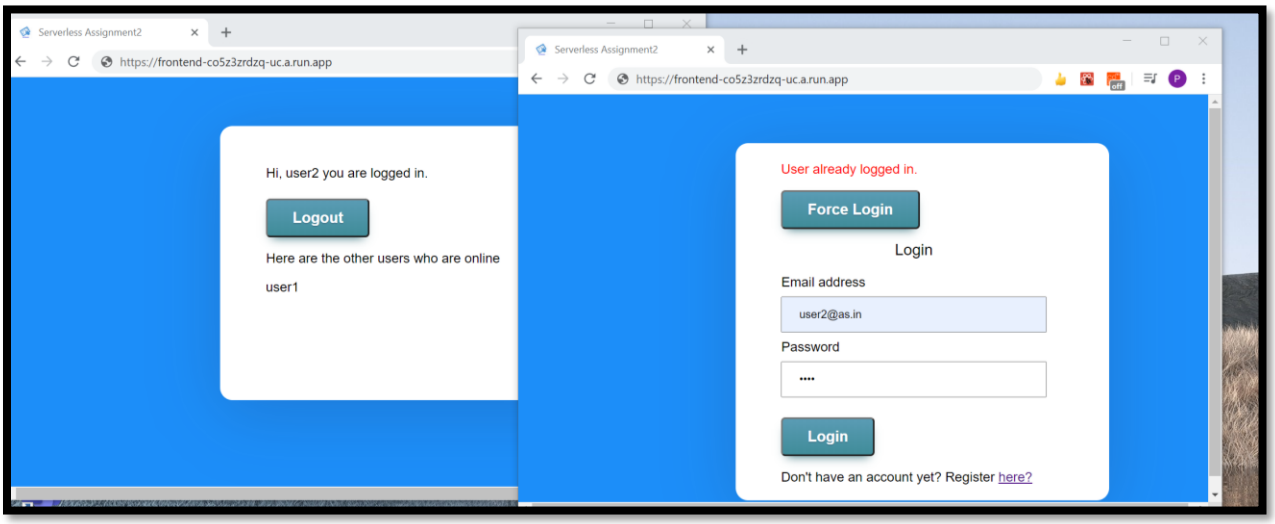


Figure 45: Force login enabled

Figure 46 shows the user2 force logged into the application.

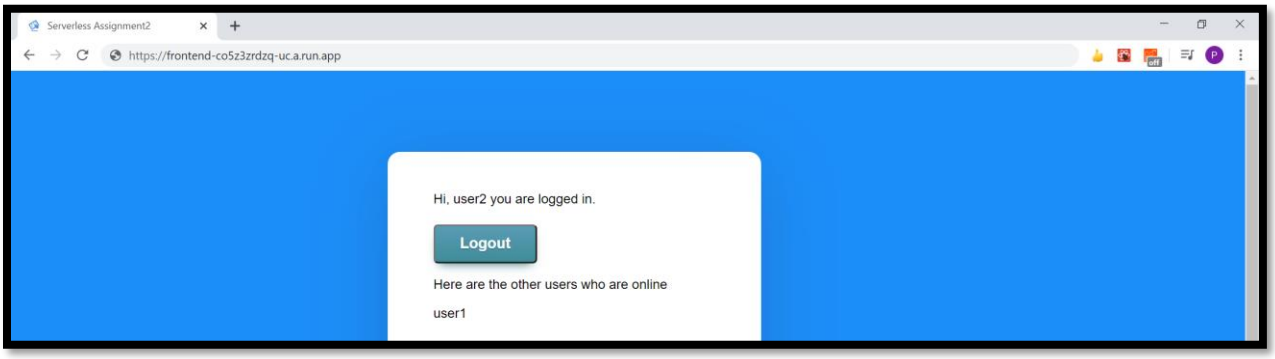


Figure 46: Force logged in

Scenario 12: Login with multiple users and check the online users in database.

User3 logged into the application. It is captured in figure 47.

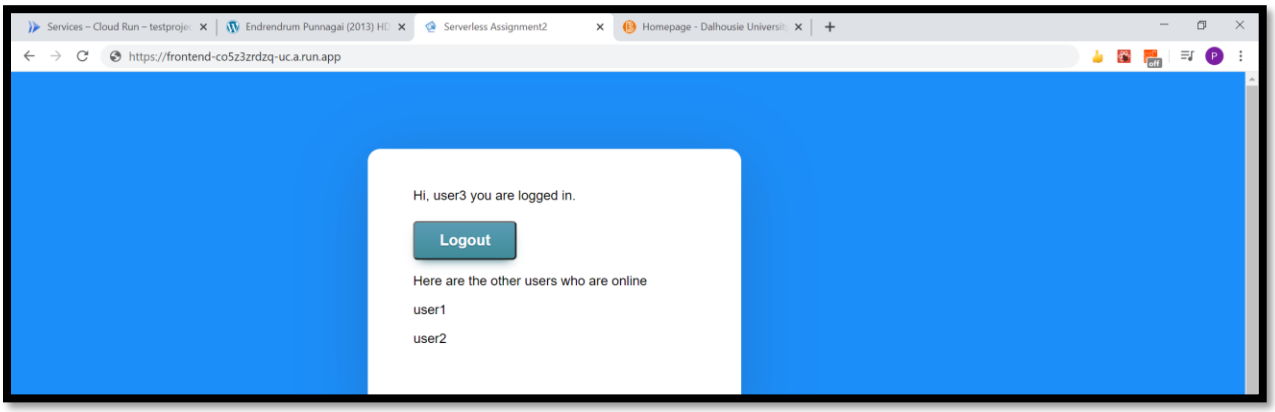


Figure 47: User 3 logged in

The online user list from database is shown in figure 48.

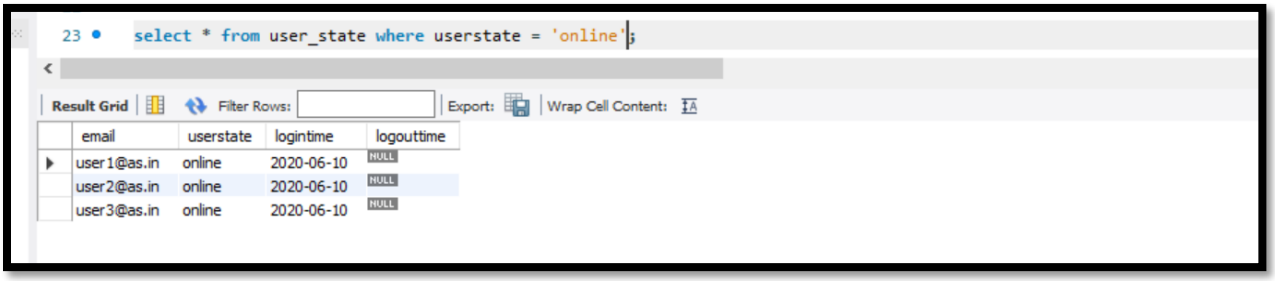


Figure 48: User list from database

The user list from database after user3 logs out from application is shown in figure 49.

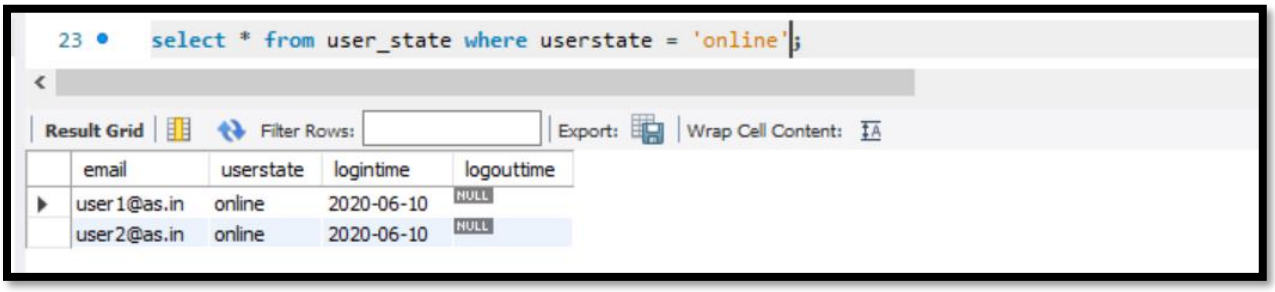


Figure 49: User list from database after user3 logs out

Summary

The meeting account application is built completely with serverless technologies. The major technologies and their roles are explained below.

Docker Containers

A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another [1]. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings [1].

The frontend, the registration, the login, and the home functionalities are built as separate applications. The testing of the applications is done in local. Then the applications are dockerized in local with docker. The docker images are deployed into separate containers in local. Again, the applications are tested. The docker tool is free of cost.

Google Containers Registry (GCR)

Container Registry is a single place for your team to manage Docker images, perform vulnerability analysis, and decide who can access what with fine-grained access control in GCP [2]. The docker images build in local are tagged into GCR using GCP CLI and are pushed into the GCR. the purpose of pushing the images is to access the images for deployment into GCP services. The price for Standard buckets is about $0.026 per GB per month.

Google Cloud Run

Cloud Run is a fully managed compute platform for deploying and scaling containerized applications quickly and securely [3]. It provides enhanced developer experience and supports all language codebase, libraries. As per assignment requirement, the docker images created in local and uploaded into GCR are deployed into GCP Cloud Run. The cloud run is simple to deploy the application and the application link is added in the cloud run page. Using Cloud Run we can create containerized application quickly. The scale up and scale down is very easy with Cloud Run and can be set automatic. Cloud Run comes with pay-per-use, with an always-free tier, rounded up to the nearest 100 milliseconds. Total cost is the sum of used CPU, Memory, Requests and Networking.

Part B: Building Chatbot

Objective

To build a chatbot for a pizza place in Amazon Lex.

OrderPizza Chatbot

The Chatbot is created in Amazon Web Service (AWS). The sequence of steps followed is explained below.

Step 1: Select Amazon Lex under machine learning module as shown in figure 50.

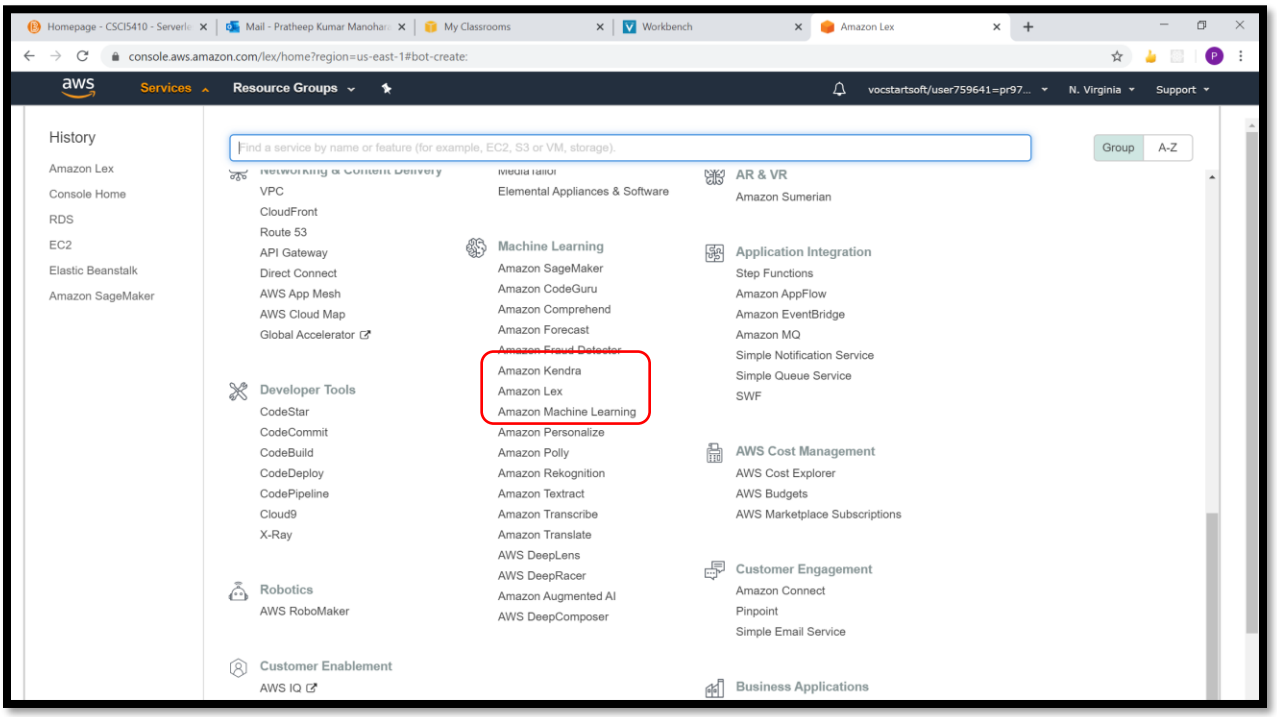


Figure 50: AWS Services

Step 2: Figure 51 shows the home page of Amazon Lex without any chatbot.

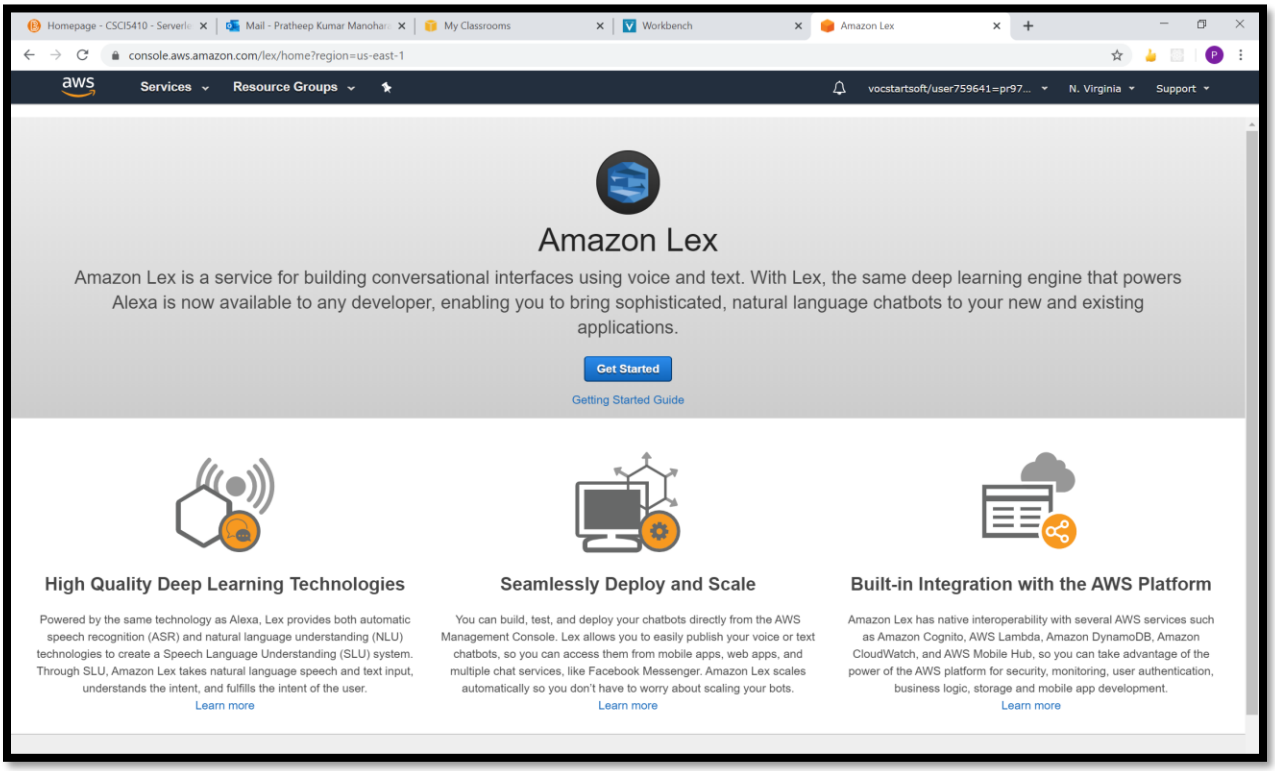


Figure 51: Lex homepage

Step 3: In the Lex homepage, click on Get Started to create a chatbot. Figure 52 captures the options available in lex page. Click on the custom bot highlighted in image 52.

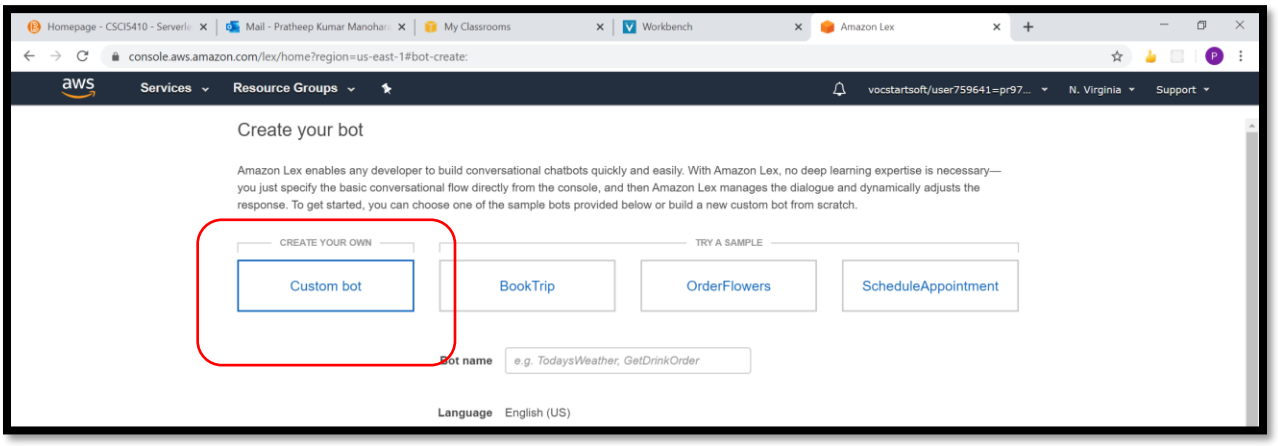


Figure 52: Chat options

Step 4: Fill in the basic details of the chatbot as shown in figure 53.

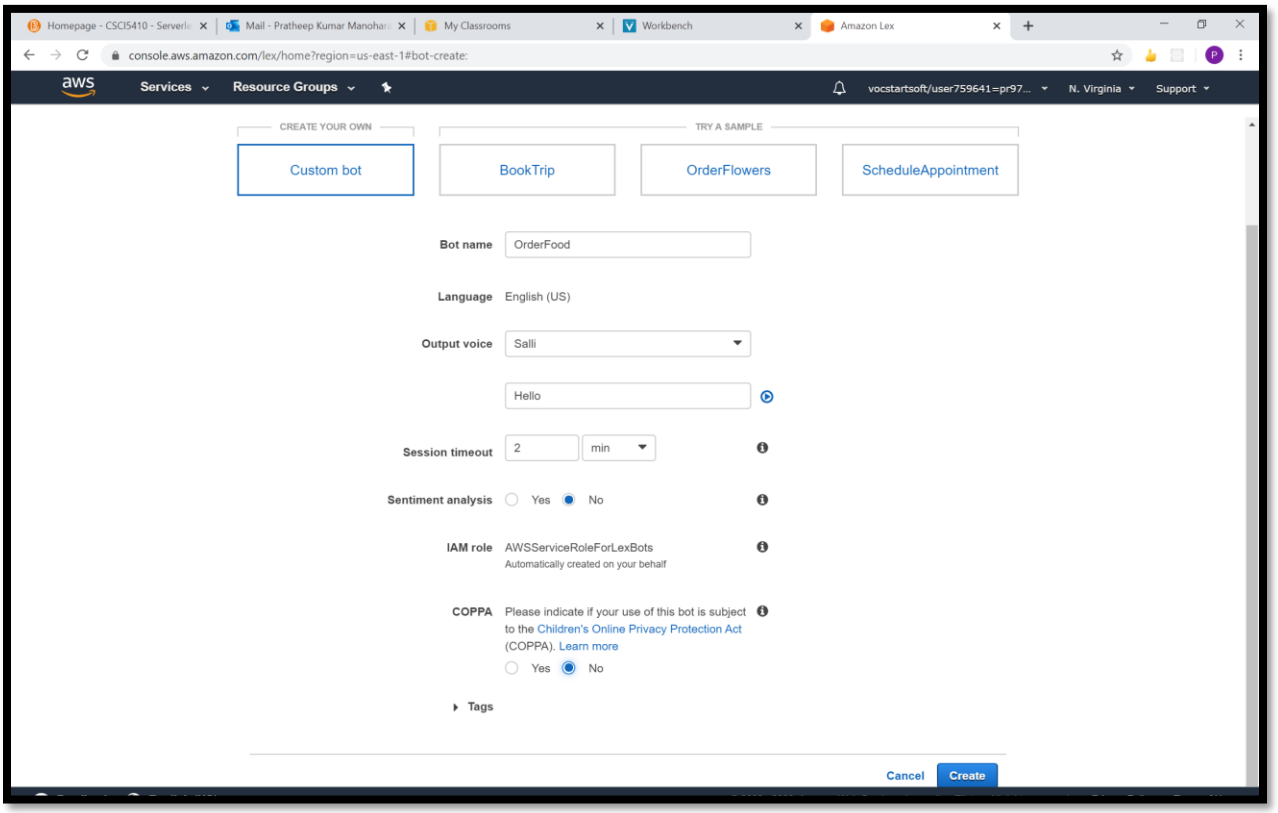


Figure 53: Chatbot creation

Step 5: The newly created chatbot is captured in figure 54.

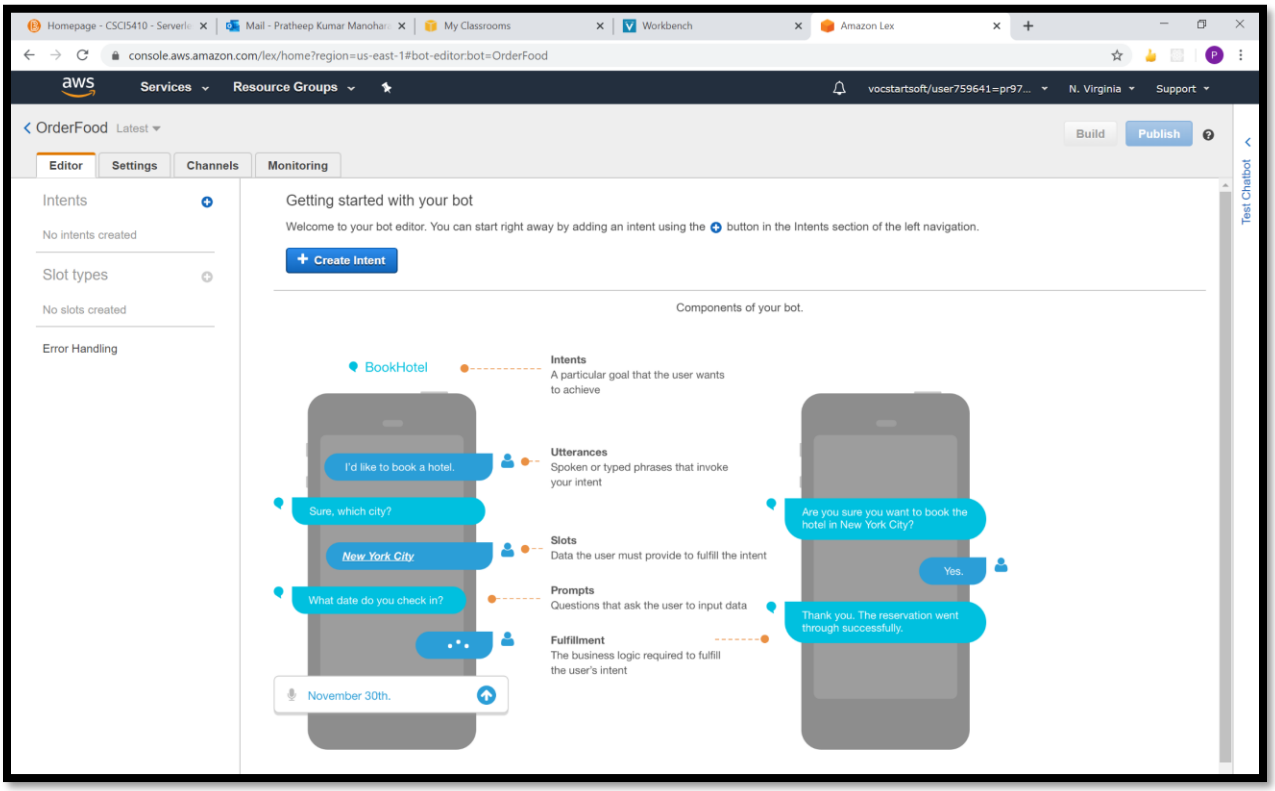


Figure 54: New chatbot

Intent 1: Greetings

Sample Utterances: Figure 55 shows the utterances that triggers the greetings intent.

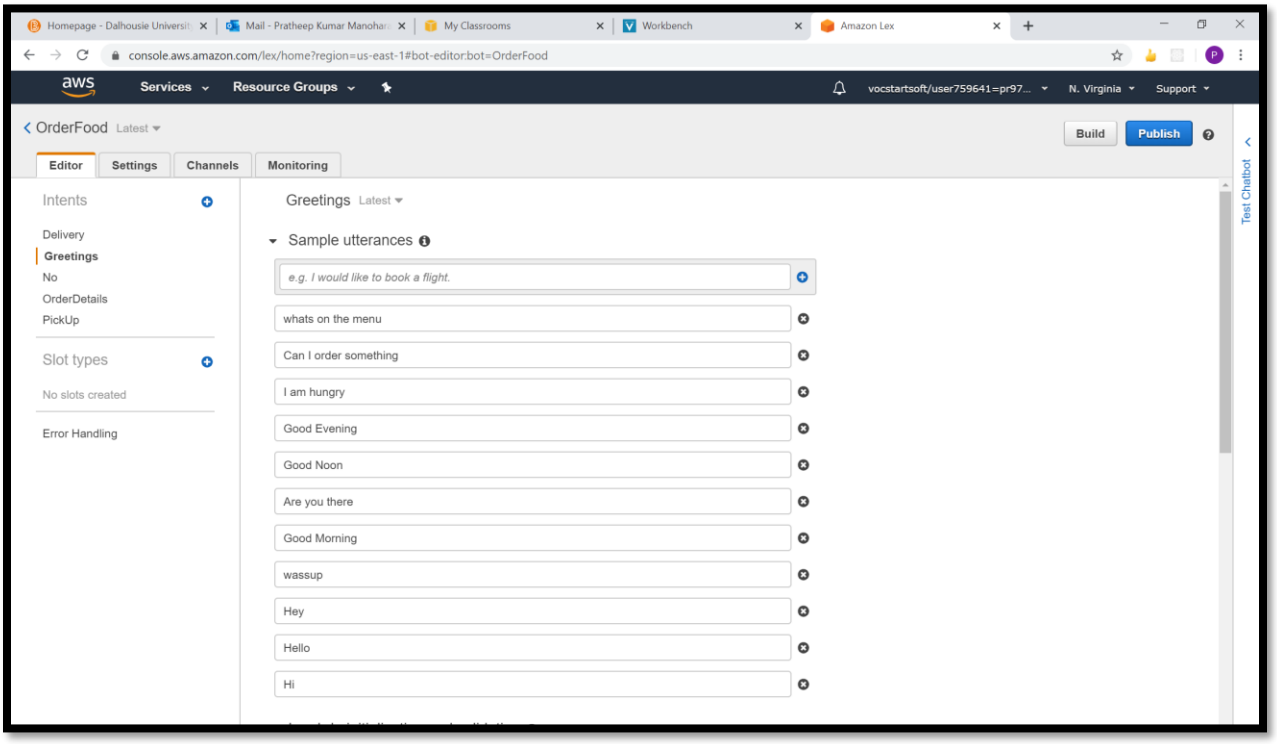


Figure 55: Sample utterances of greetings intent

Figure 56 shows the slots and confirmation prompt of greetings intent. Only one slot named Name is created to get the username is created.

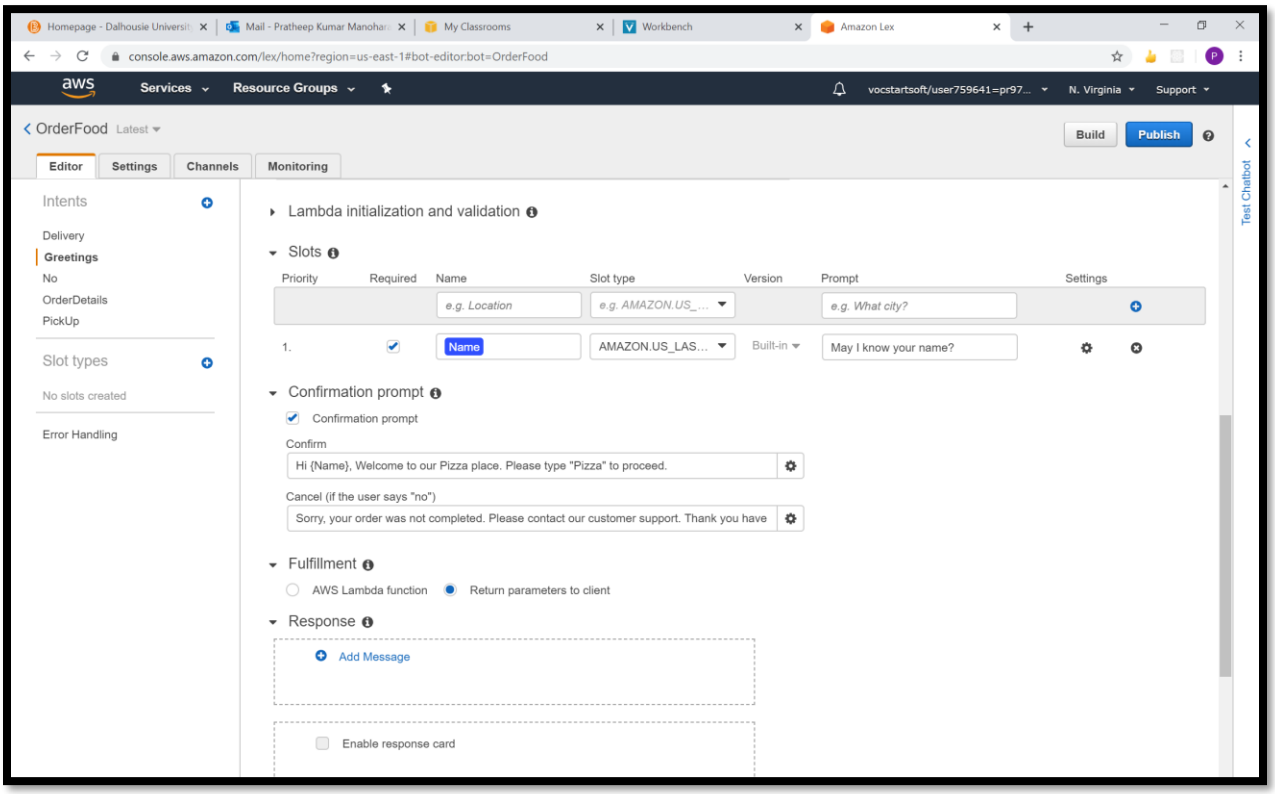


Figure 56: Slots and Confirmation prompt of greetings intent

Intent 2: OrderDetails

Sample utterances of orderdetails intent is shown in figure 57.

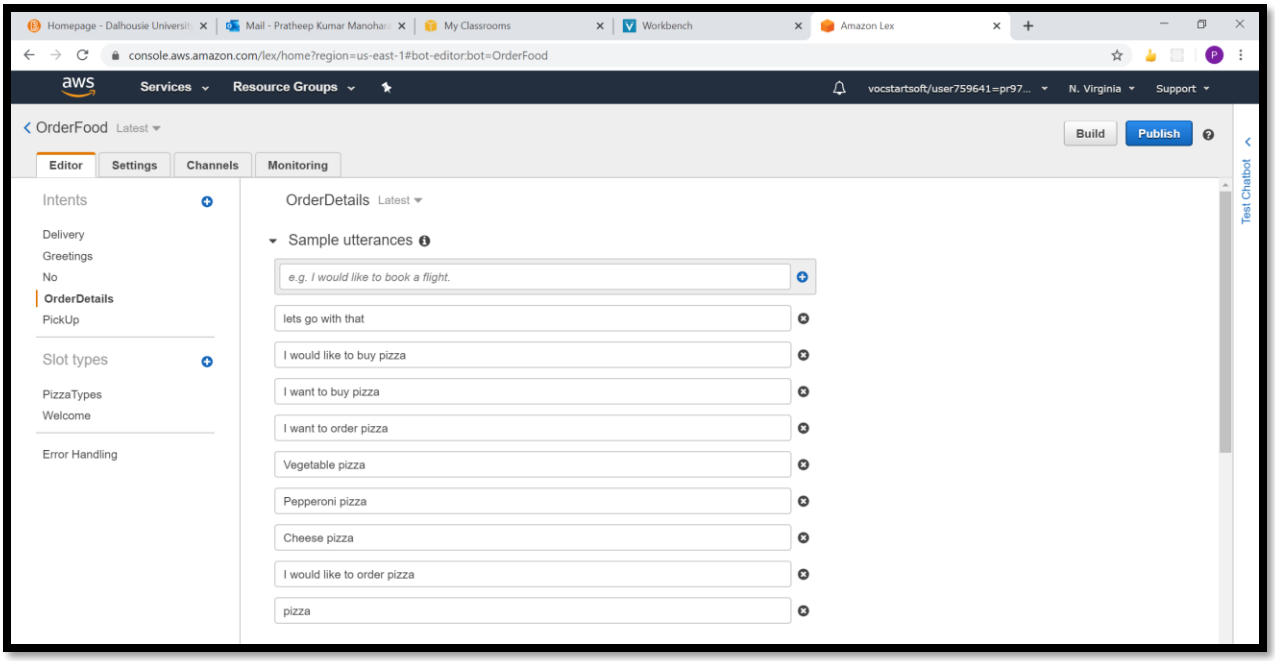


Figure 57: Sample utterances of OrderDetails intent

Figure 58 shows the slots and confirmation prompt of orderdetails intent. This intent is used to capture all the orderdetails of the order. The type of Pizza, type of crust, and the type of delivery are captured in this intent.

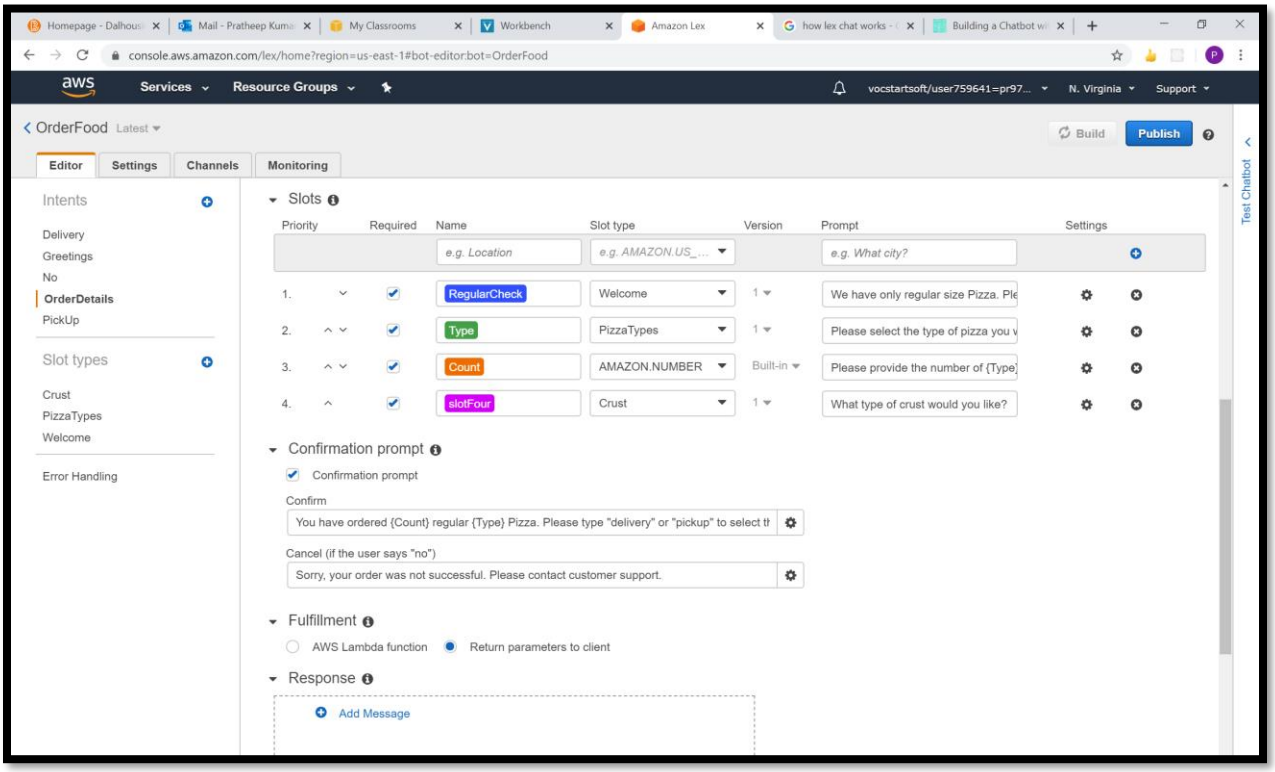


Figure 58: Slots and confirmation prompt of OrderDetails intent

Intent 3: Delivery

Sample utterances of the delivery intent is shown in figure 59.

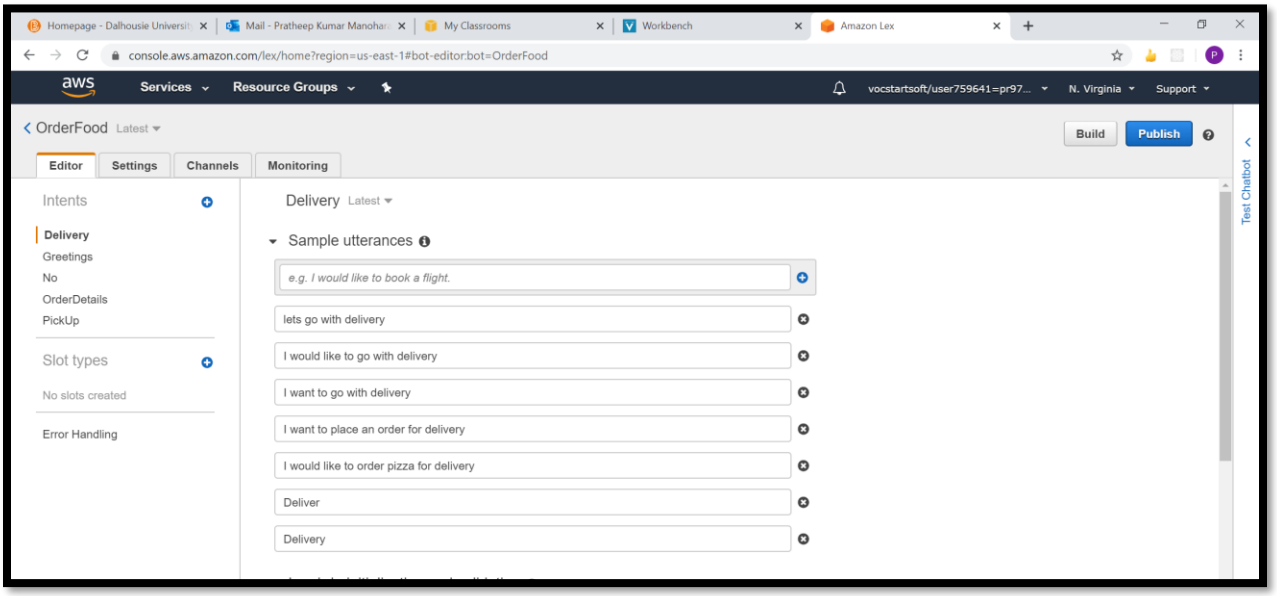


Figure 59: Sample utterances of delivery intent

Figure 60 shows the slots and confirmation prompt of delivery intent. The delivery address, date, time, and contact number are retrieved in this intent.

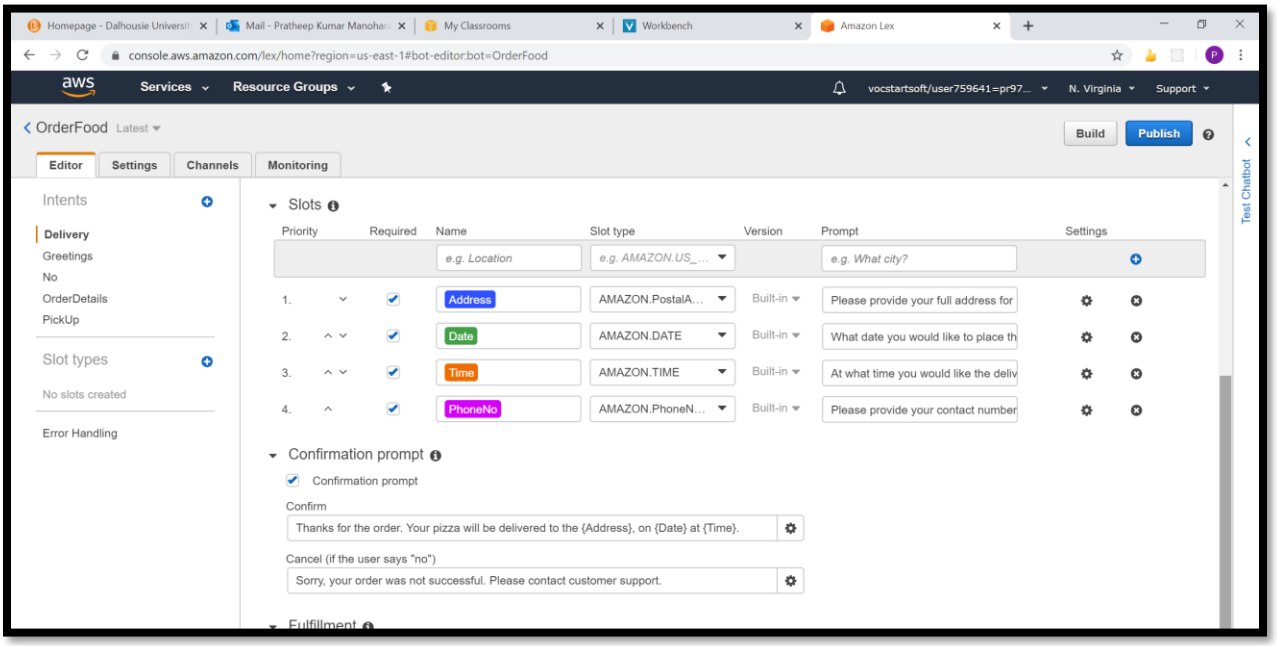


Figure 60: Slots and confirmation prompt of delivery intent

Intent 4: PickUp intent

Sample utterances of the pickup intent is shown in figure 61.

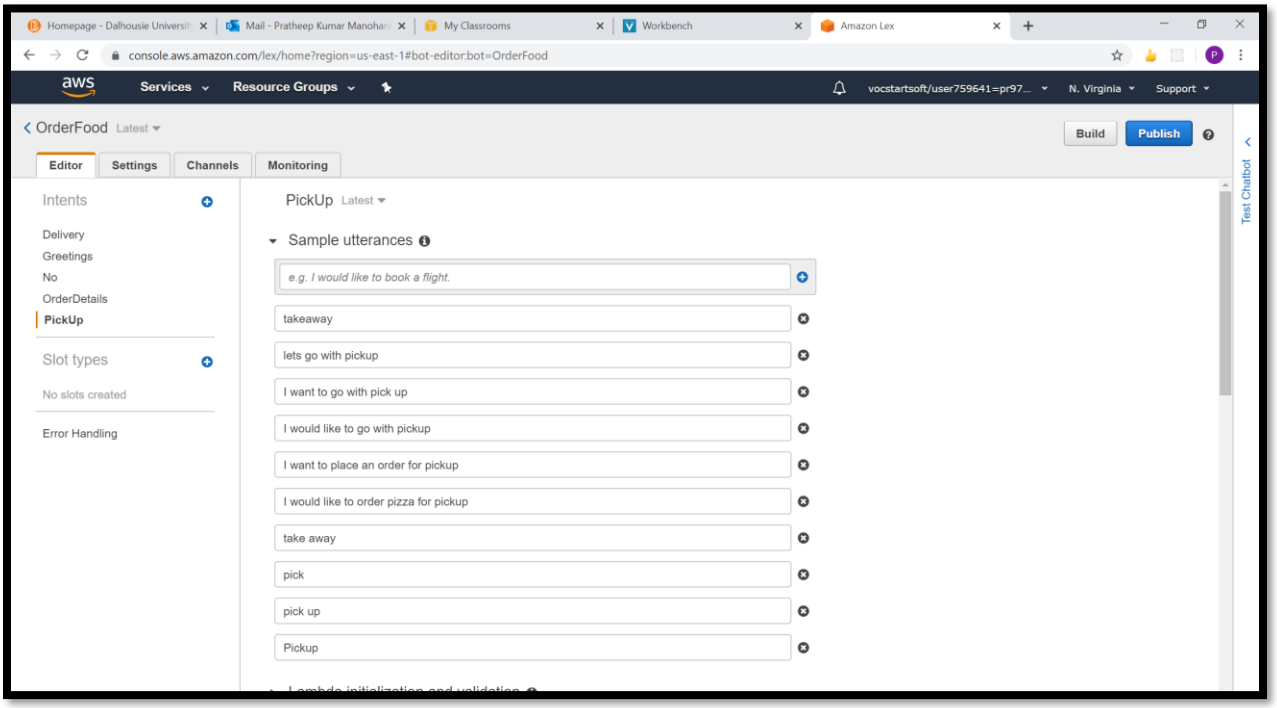


Figure 61: Sample utterances of pickup intent

Figure 62 shows the slots and confirmation prompt of pickup intent. The pickup time and contact number are retrieved from the users in this intent.

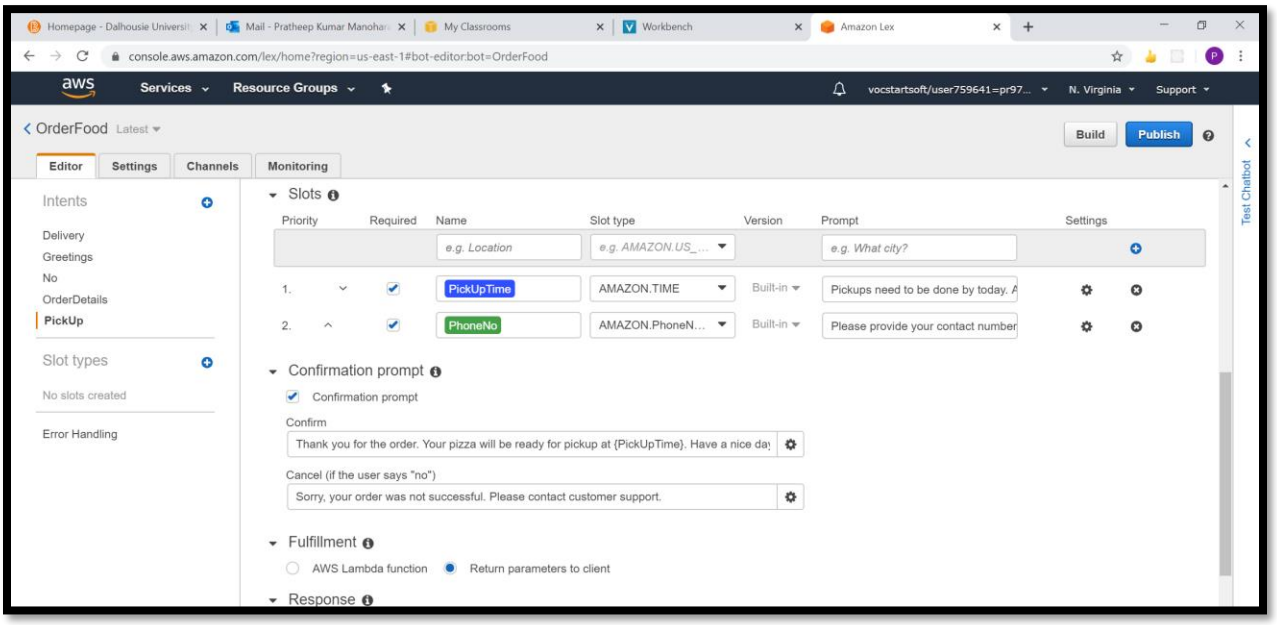


Figure 62: Slots and confirmation prompt of pickup intent

Intent 5: No

The details of No intent are shown in figure 63.

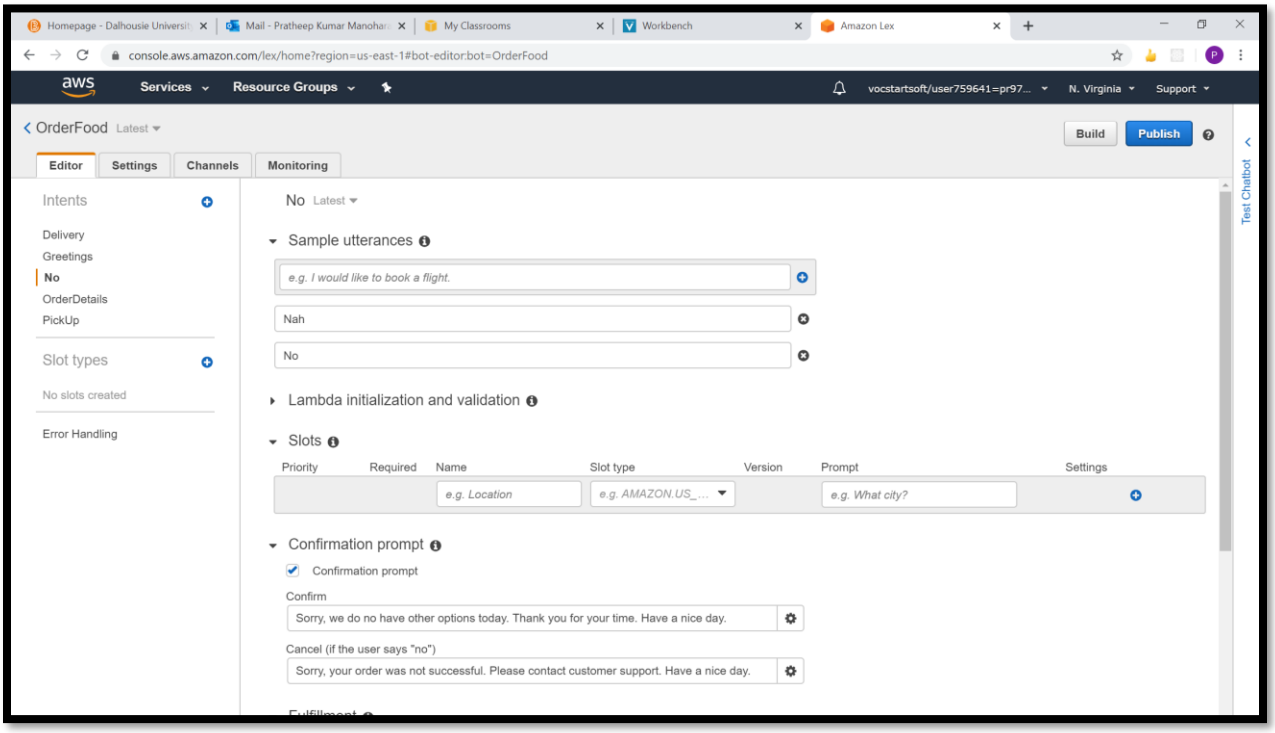


Figure 63: No intent

Error Handling

Figure 64 shows the error messages of the chatbot.

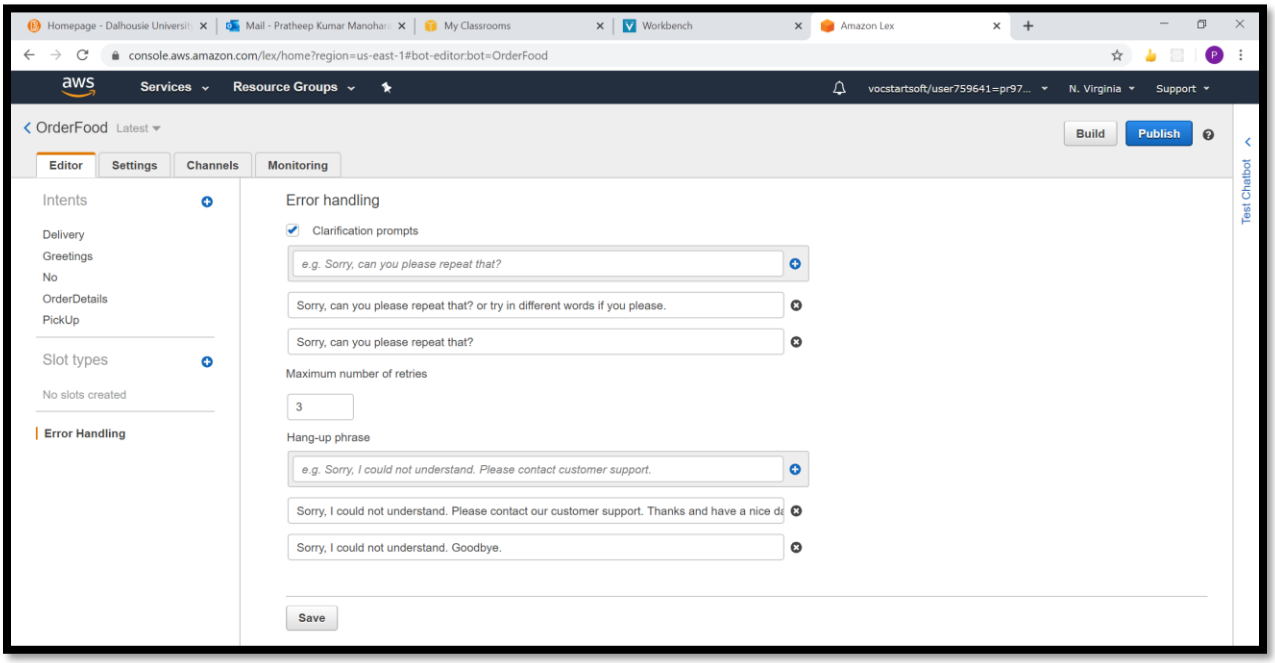


Figure 64: Error handling of chatbot

Test Cases

The test cases used to test the chatbot are mentioned below.

S No	Test scenario	Expected result	Actual result
1	order 1 regular size veg pizza for delivery with tomorrow and noon	order should be successful, and the user should be prompted with appropriate success message.	pass
2	whats on the menu order 4 regular size cheese pizza for delivery with three days and pm time	order should be successful, and the user should be prompted with appropriate success message.	pass
3	I am hungry order 3 regular size pepperoni pizza for delivery diff format date and time in 24 formats	order should be successful, and the user should be prompted with appropriate success message.	pass

4	I would like to buy pizza order three regular size veg pizza for pickup with time pm	order should be successful, and the user should be prompted with appropriate success message.	pass
5	cheese pizza order regular size "cheese" pizza for pickup with evening	order should be successful, and the user should be prompted with appropriate success message.	pass
6	order regular size pepperoni pizza for pickup with time	order should be successful, and the user should be prompted with appropriate success message.	pass
7	couple of other utterances to start	order should be successful, and the user should be prompted with appropriate success message.	pass
8	no for regular pizza	order should be cancelled, and user should be prompted with appropriate error message	pass
9	no for order confirmation	order should be cancelled, and user should be prompted with appropriate error message	

Testing

Scenario 1: Order 1 regular size veg pizza for delivery tomorrow noon. Figures 65, and 66 shows the scenario 1.

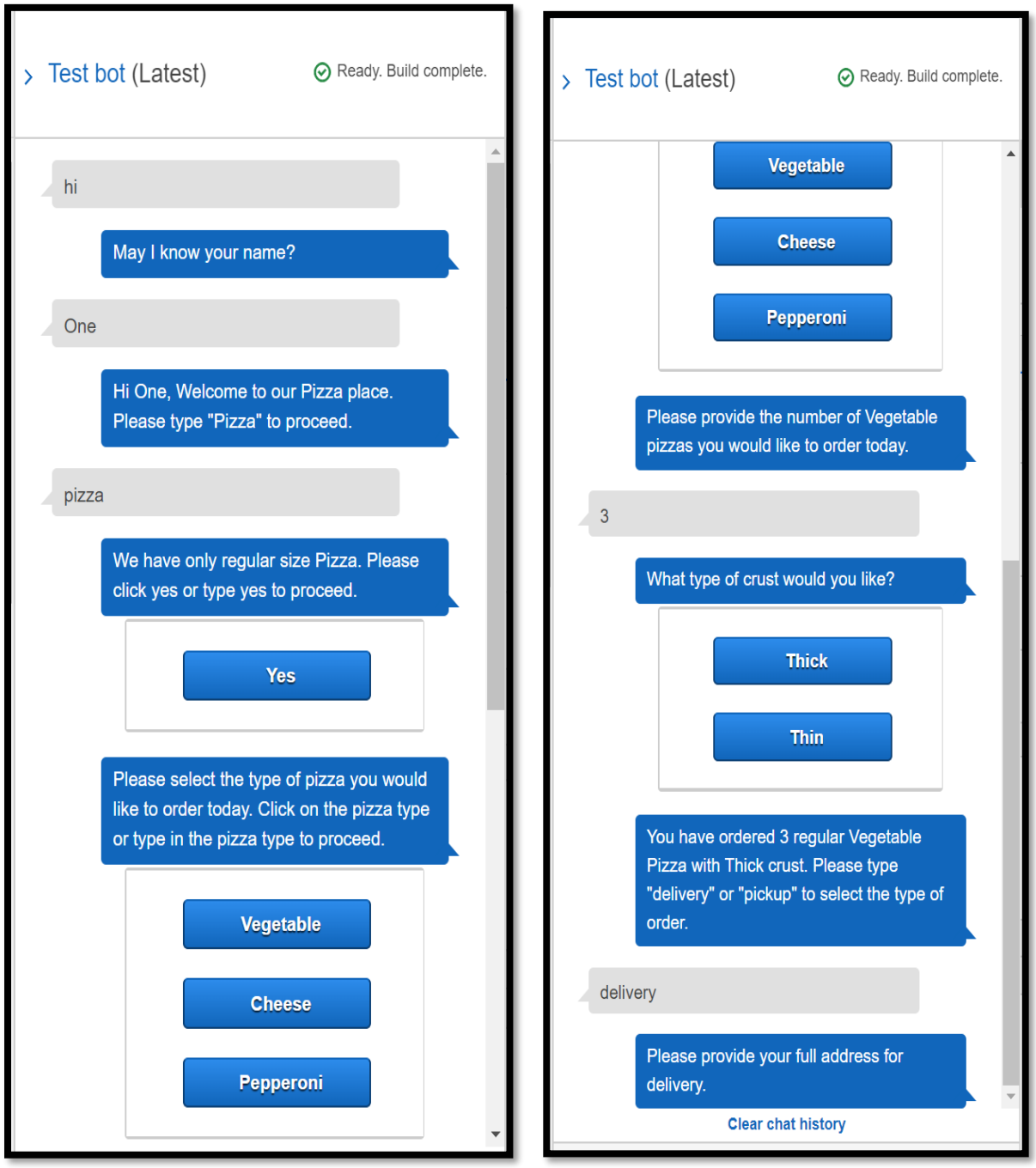


Figure 65: Test scenario 1

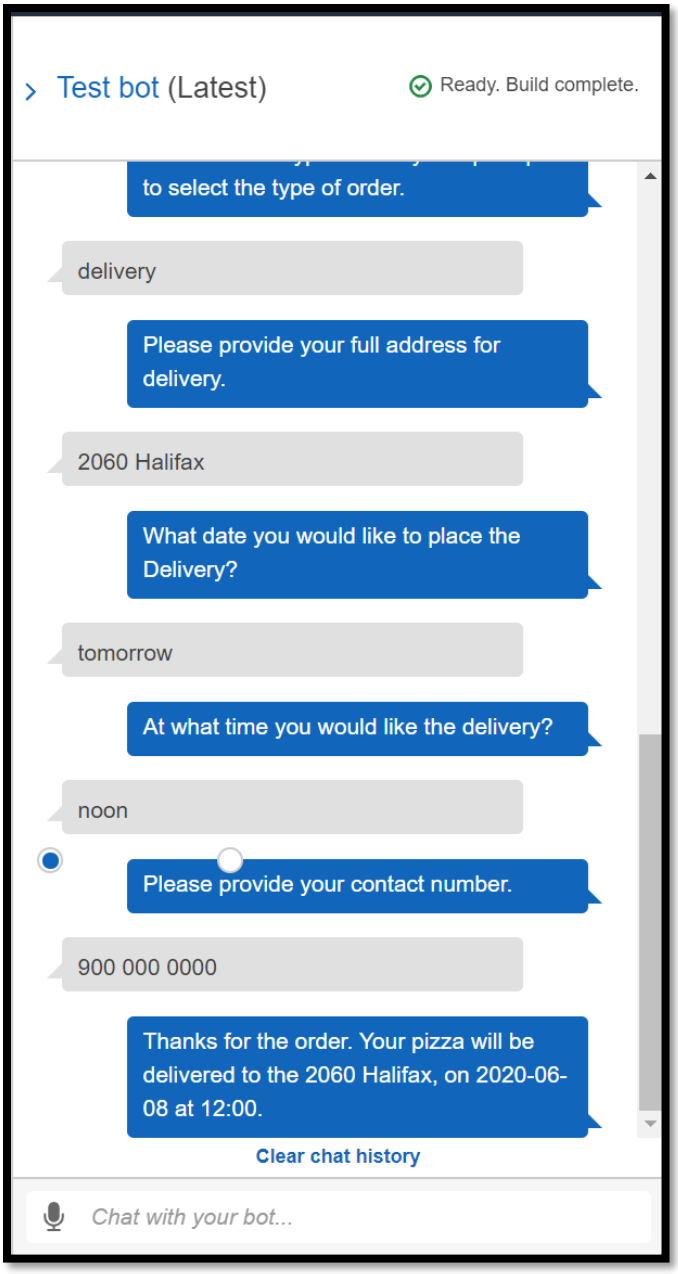


Figure 66: Test scenario 1 image 2

Scenario 2: Order 4 regular size cheese pizza for delivery with three days and pm time. Figures 67 and 68 shows the results of scenario 2.

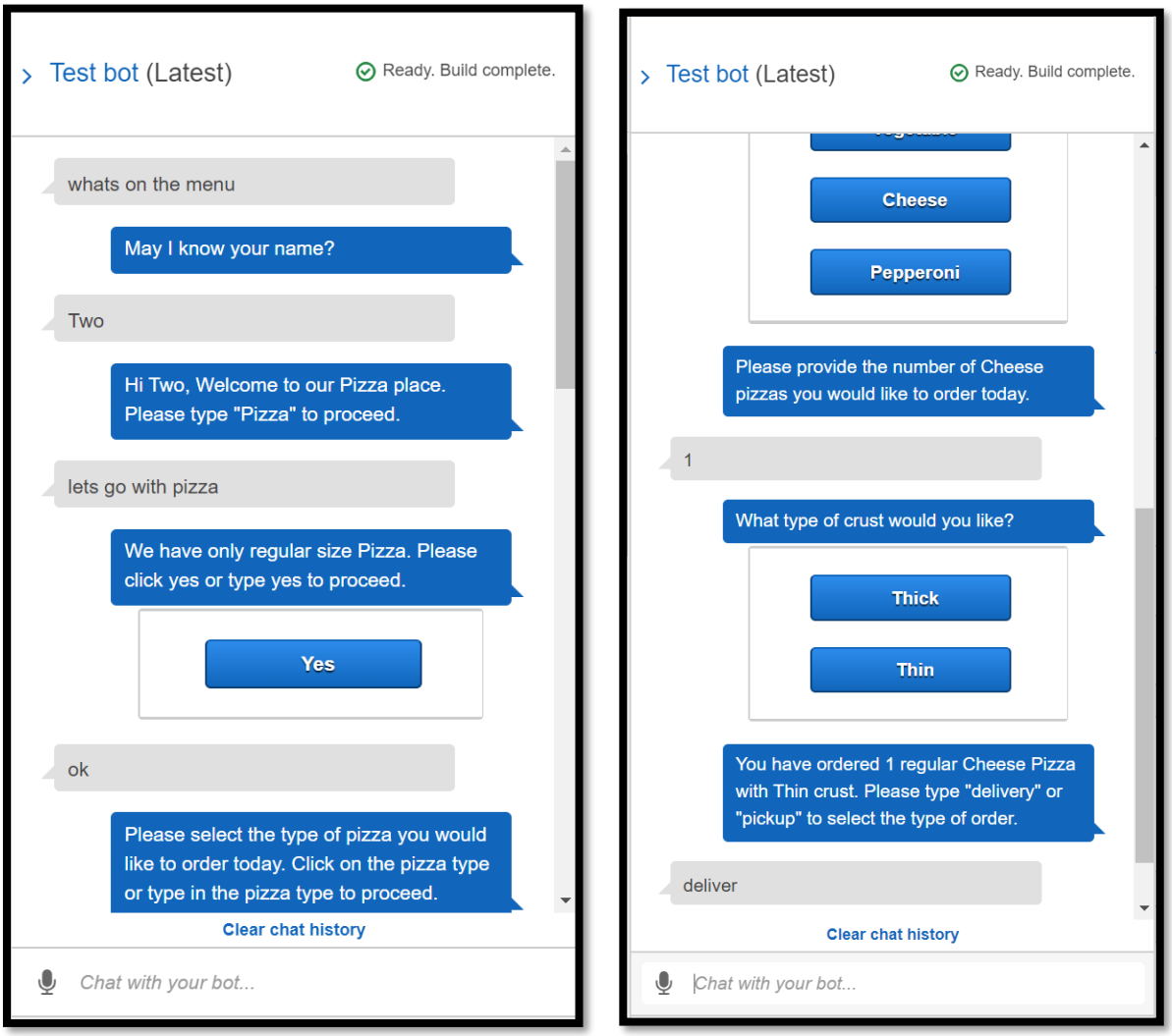


Figure 67: Test scenario 2 image 1

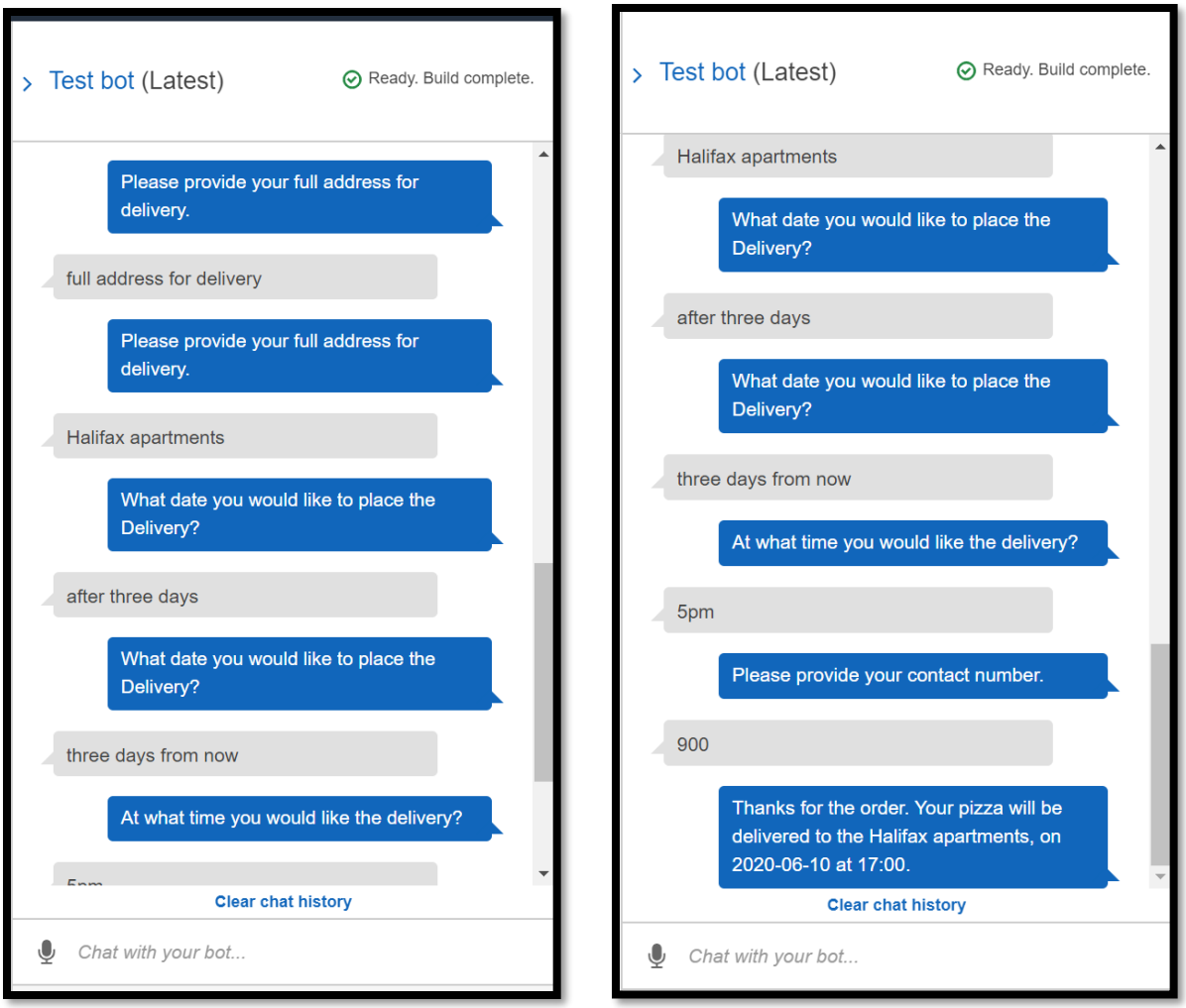


Figure 68: Test scenario 2 image 2

Scenario 3: Order three regular size pepperoni pizza for delivery diff format date and time in 24-hour time format. Figures 69 and 70 shows the results of scenario 3.

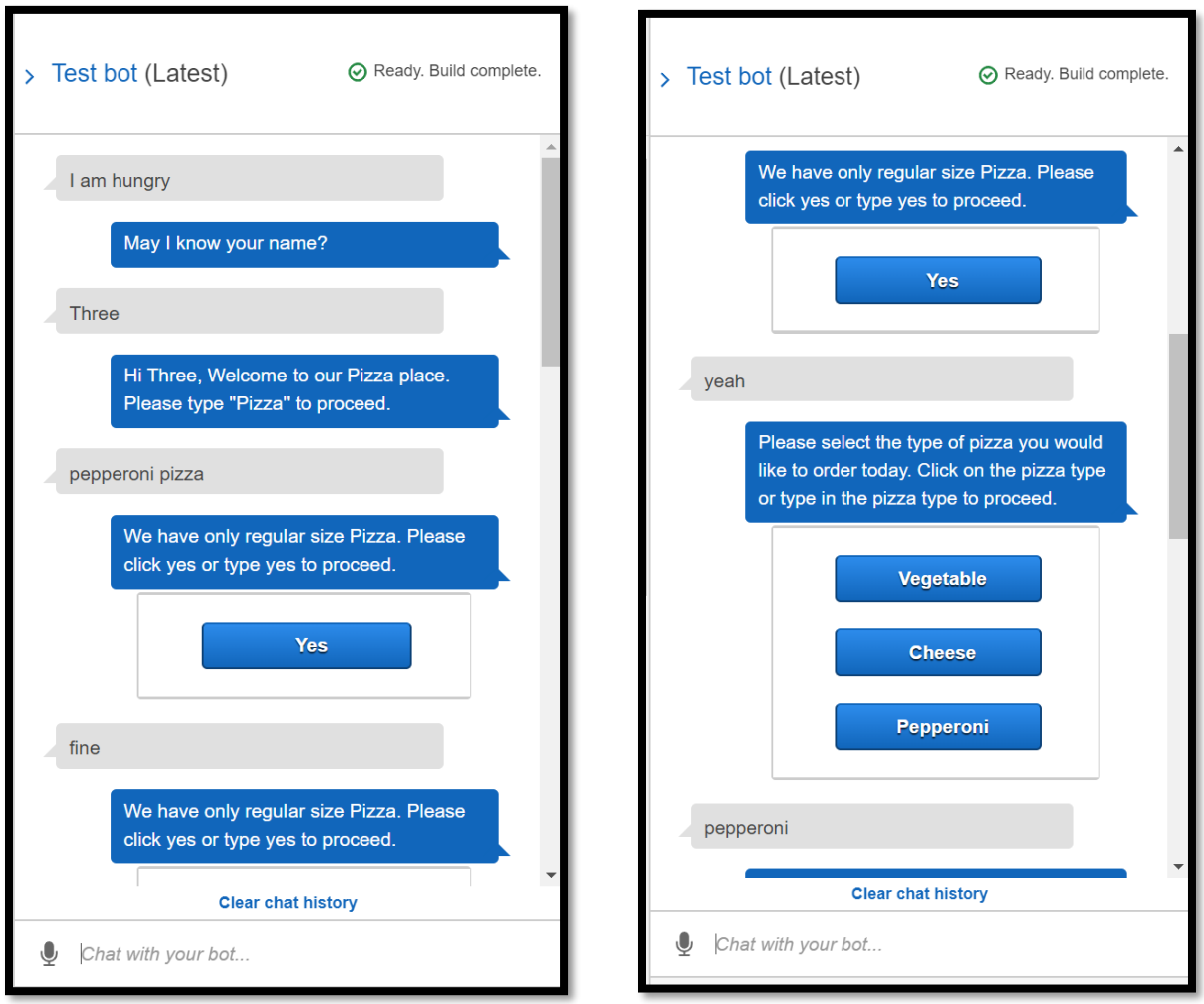


Figure 69: Test scenario 3 image 1

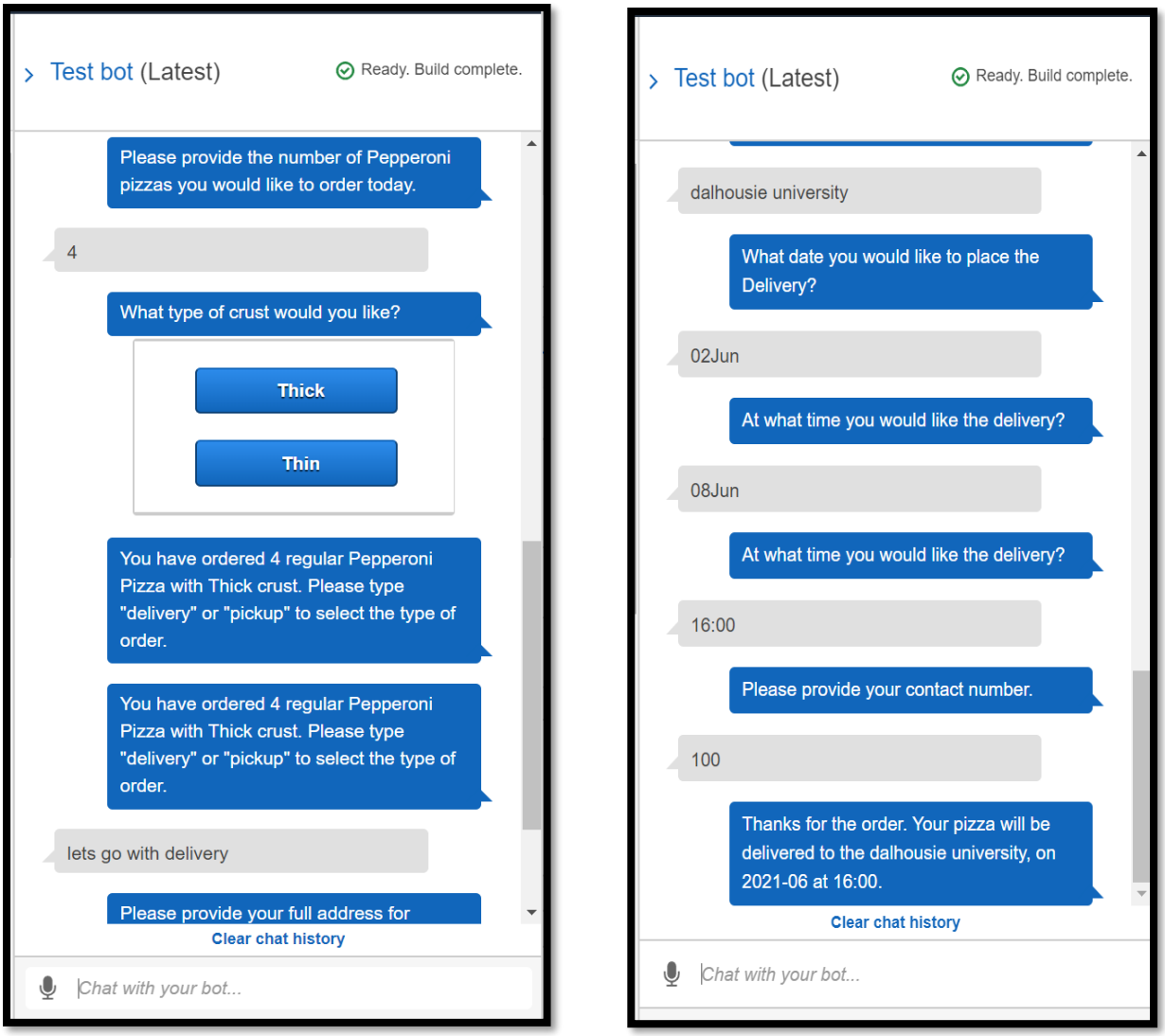


Figure 70: Test scenario 3 image 2

Scenario 4: Order three regular size veg pizza for pickup with time pm. Figures 71 and 72 shows the results of scenario 4.

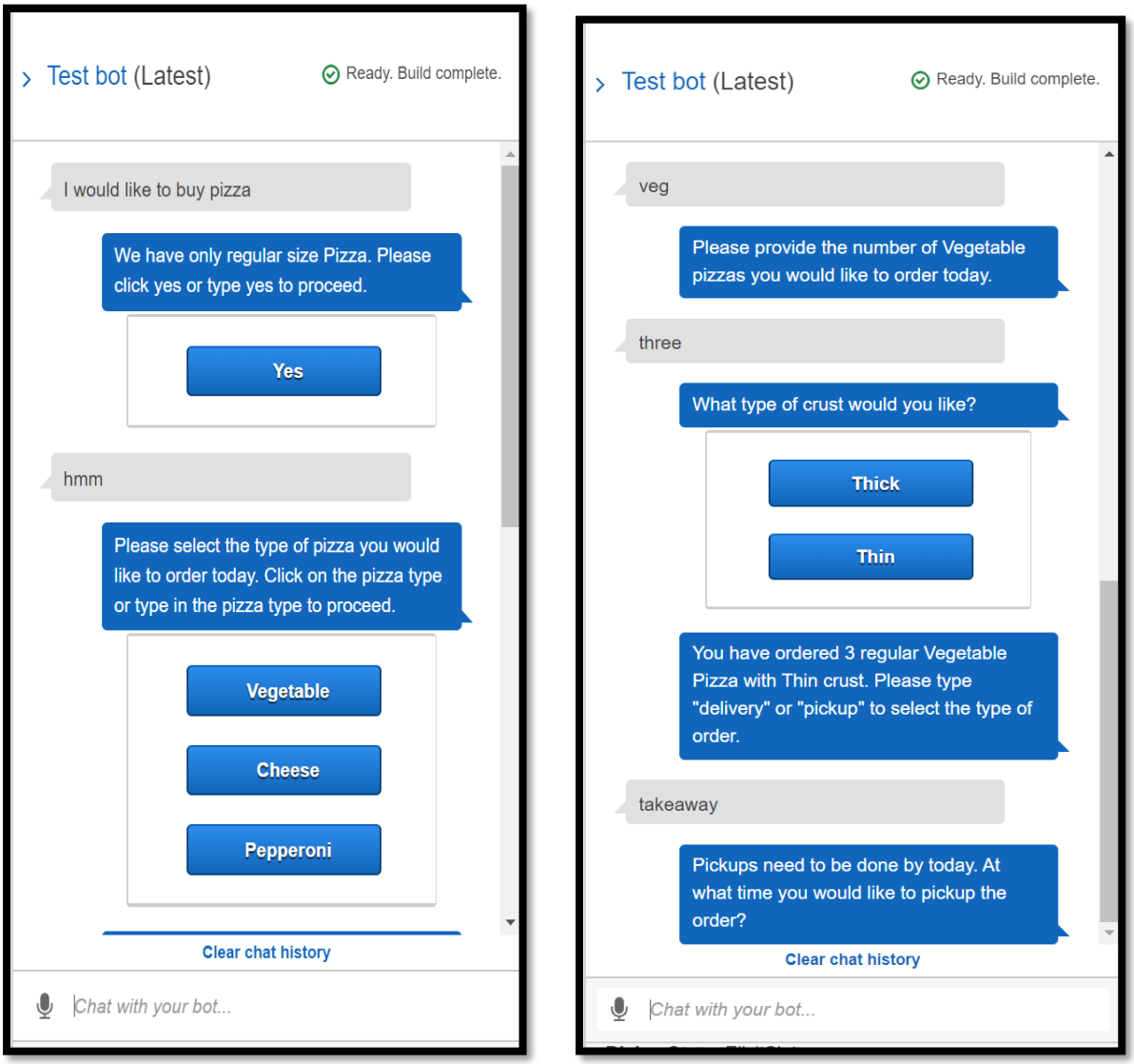


Figure 71: Test scenario 4 image 1

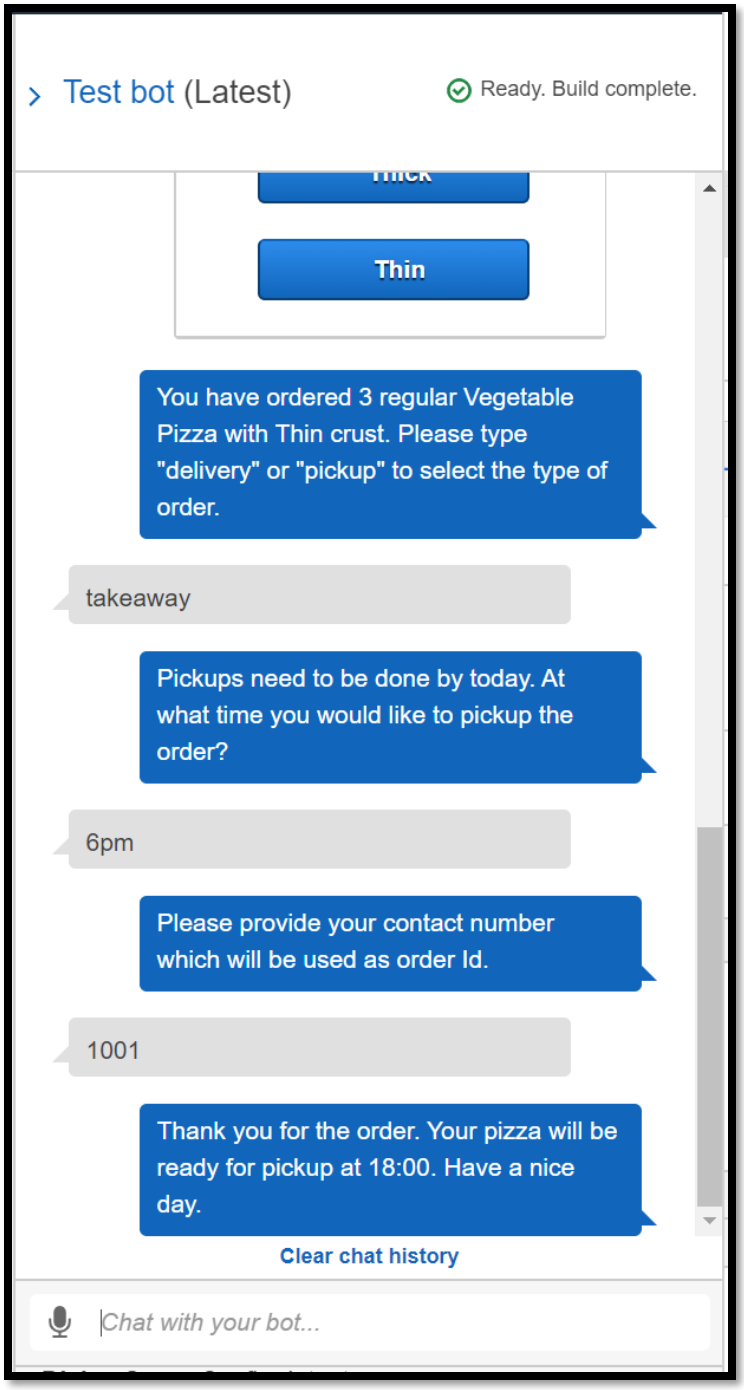


Figure 72: Test scenario 4 image 2

Scenario 5: Order regular size "cheese" pizza for pickup with evening. Figures 73 and 74 shows the results of scenario 5.

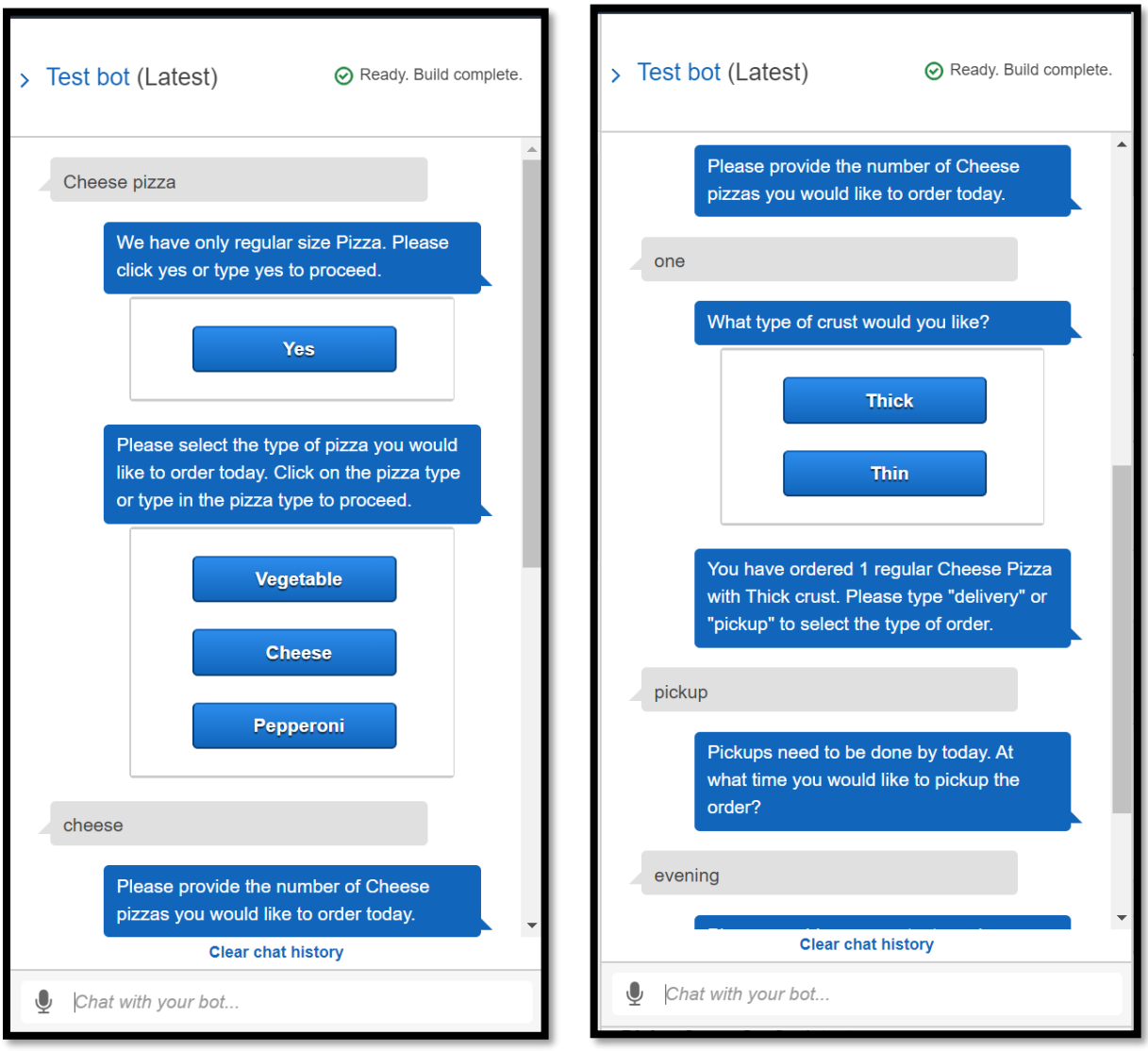


Figure 73: Test scenario 5 image 1

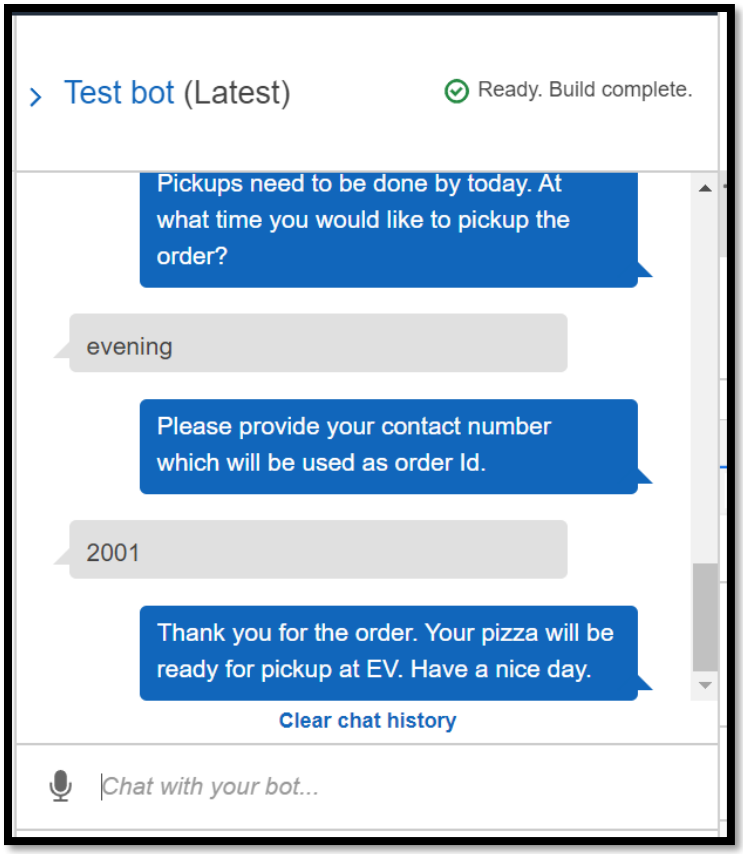
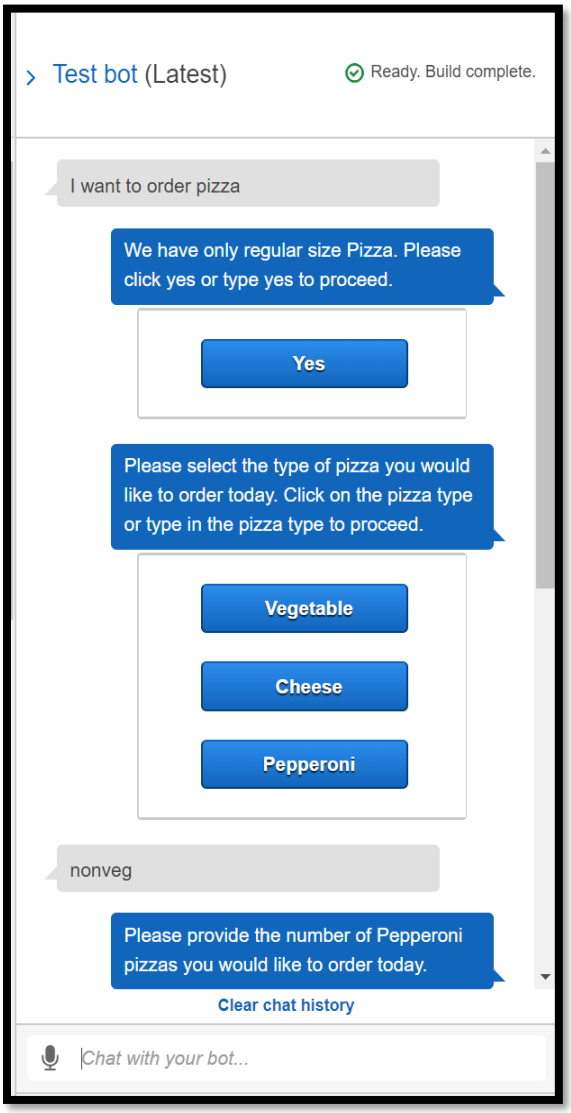


Figure 74: Test scenario 5 image 2

Scenario 6: Order regular size pepperoni pizza for pickup with time. Figures 75 and 76 show the results of scenario 6.



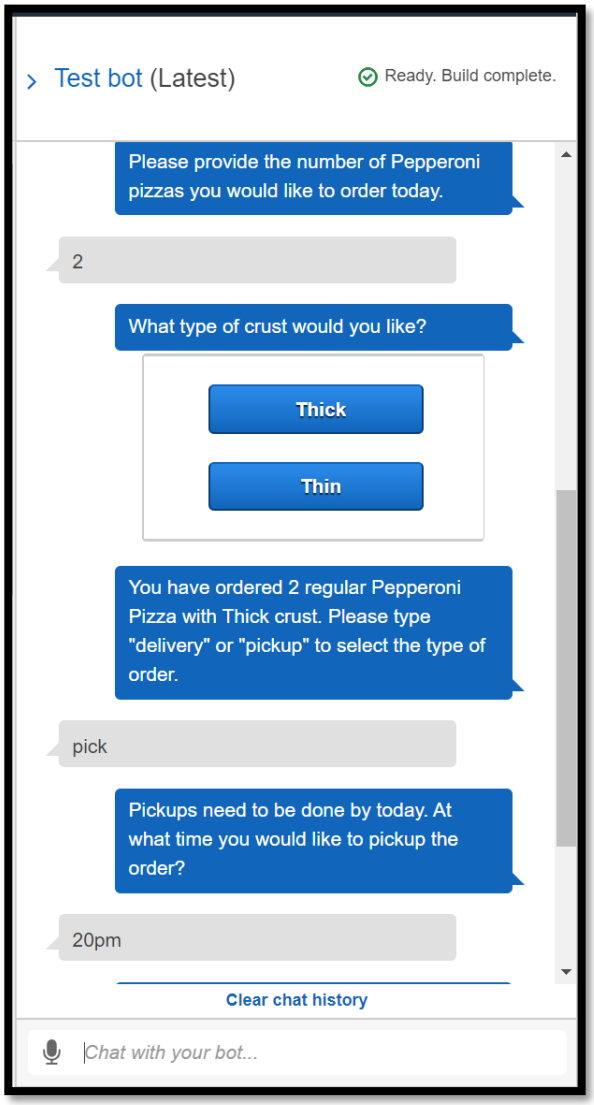


Figure 75: Test scenario 6 image 1

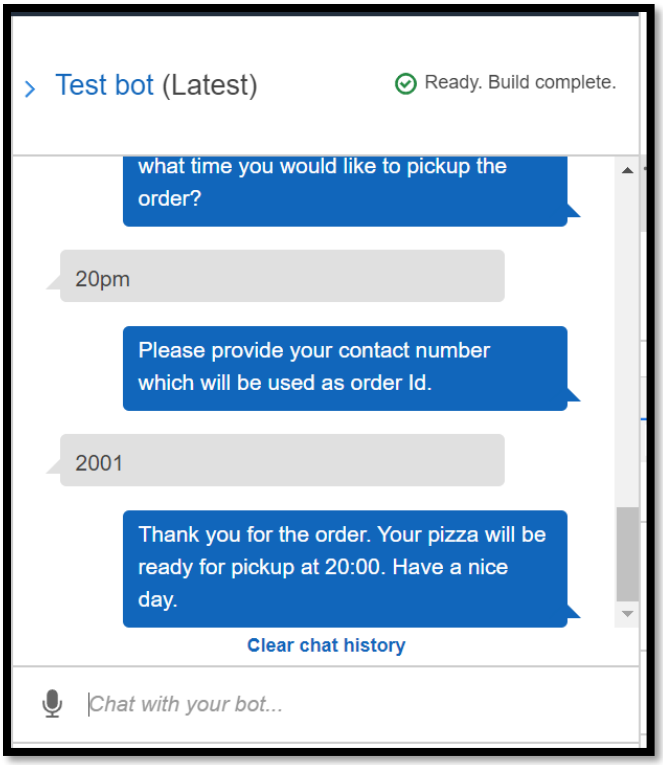


Figure 76: Test scenario 6 image 2

Scenario 7: Couple of other utterances to start. Figures 77 and 78 show the results of scenario 7.

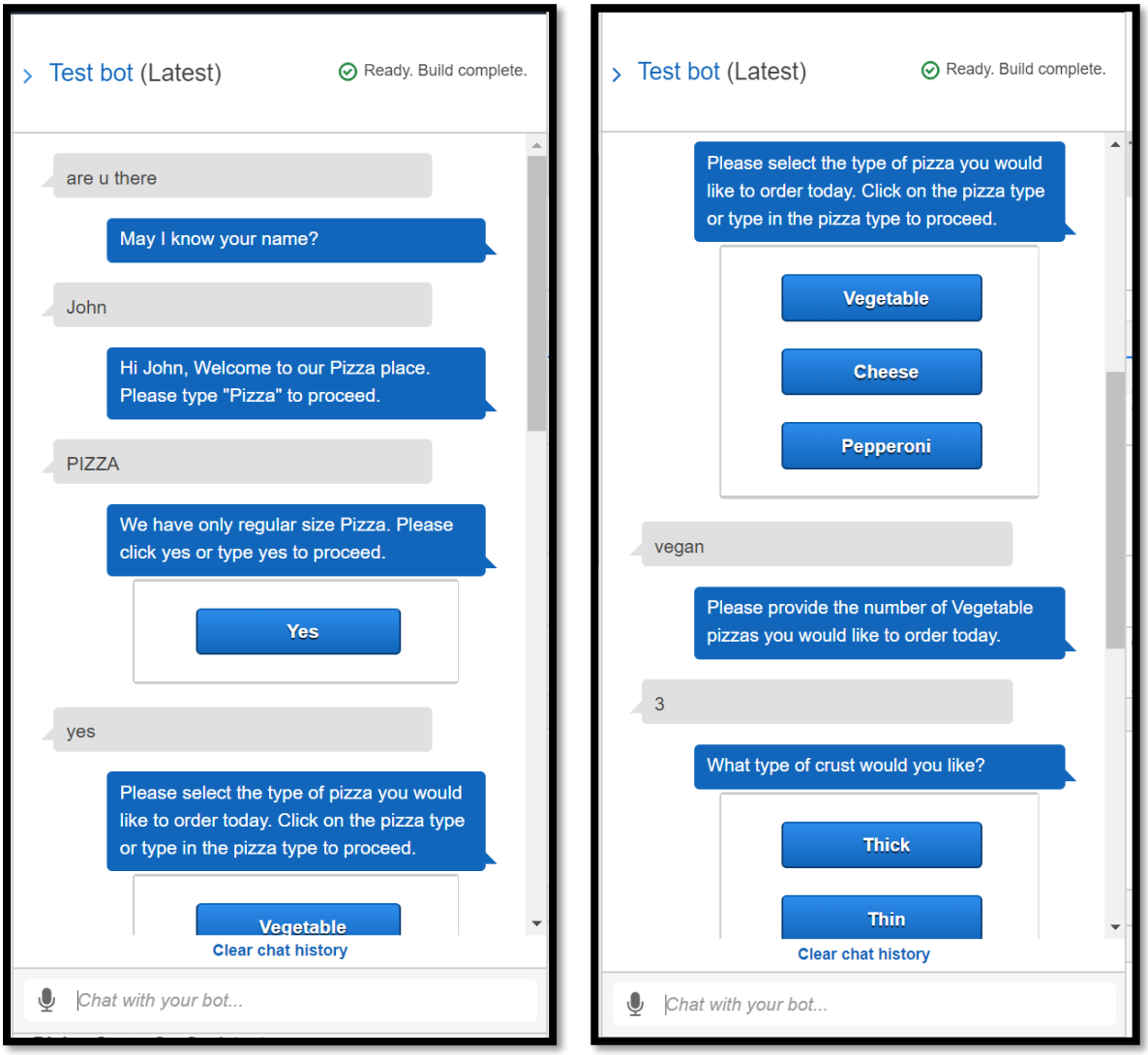


Figure 77: Test scenario 7 image 1

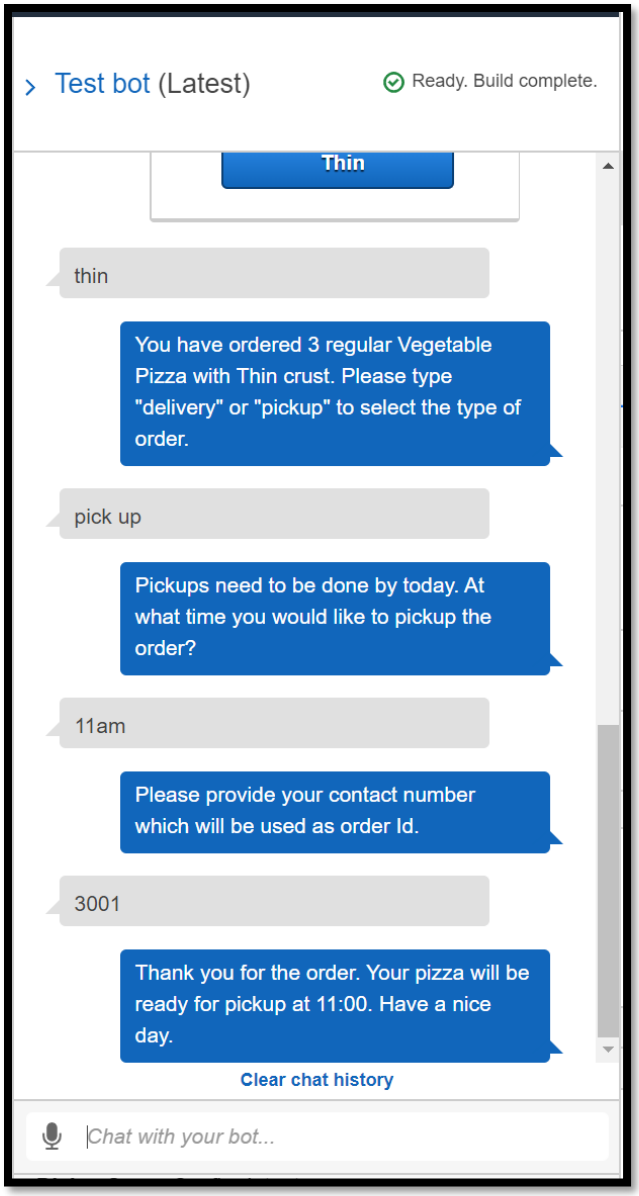


Figure 78: Test scenario 7 image 2

Scenario 8: No intent testing. Figure 79 shows the result of scenario 8.

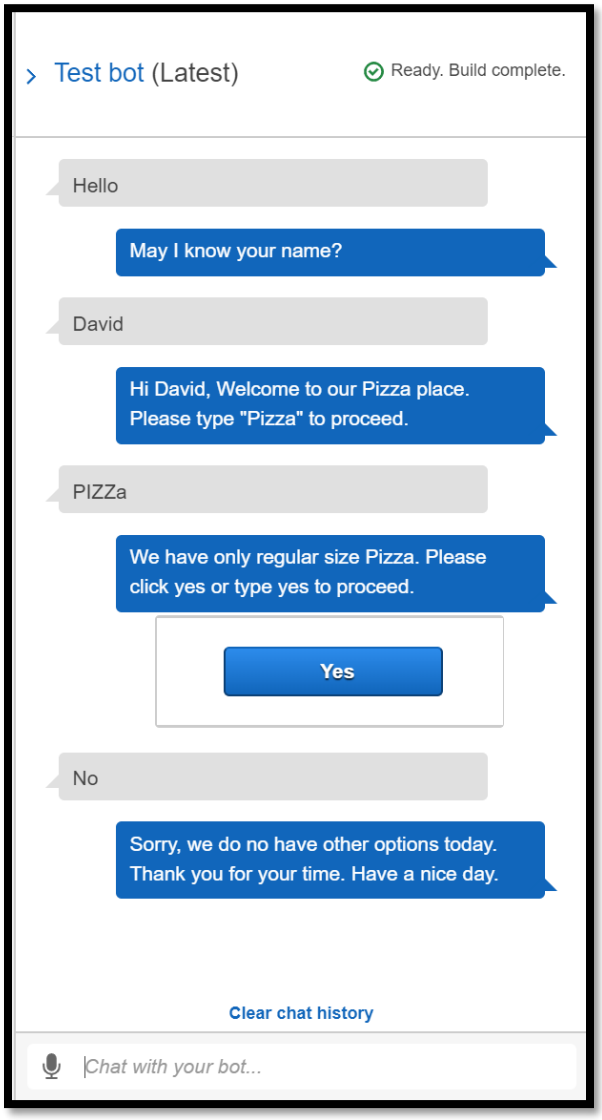


Figure 79: Test scenario 8

Scenario 9: Sample No intent in different order. Figures 80 and 81 show the result of scenario 9.

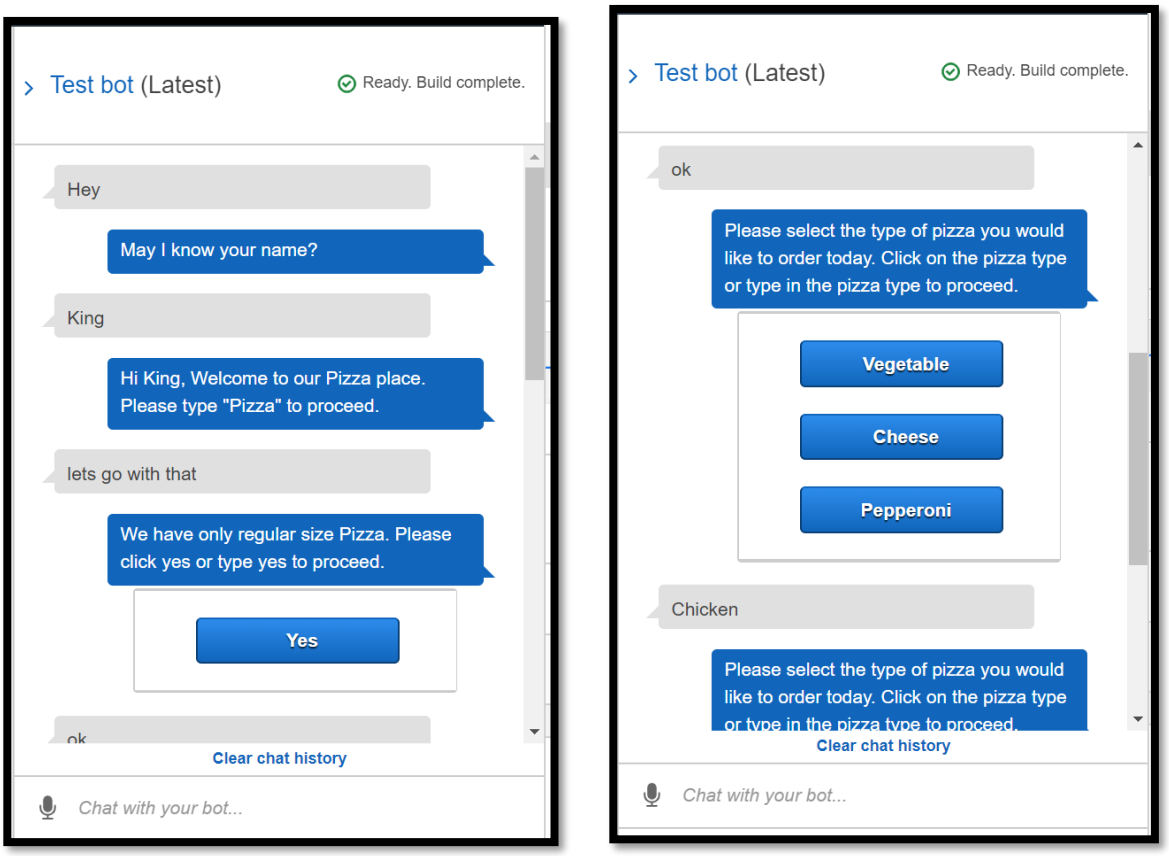


Figure 80: Test scenario 9 image 1

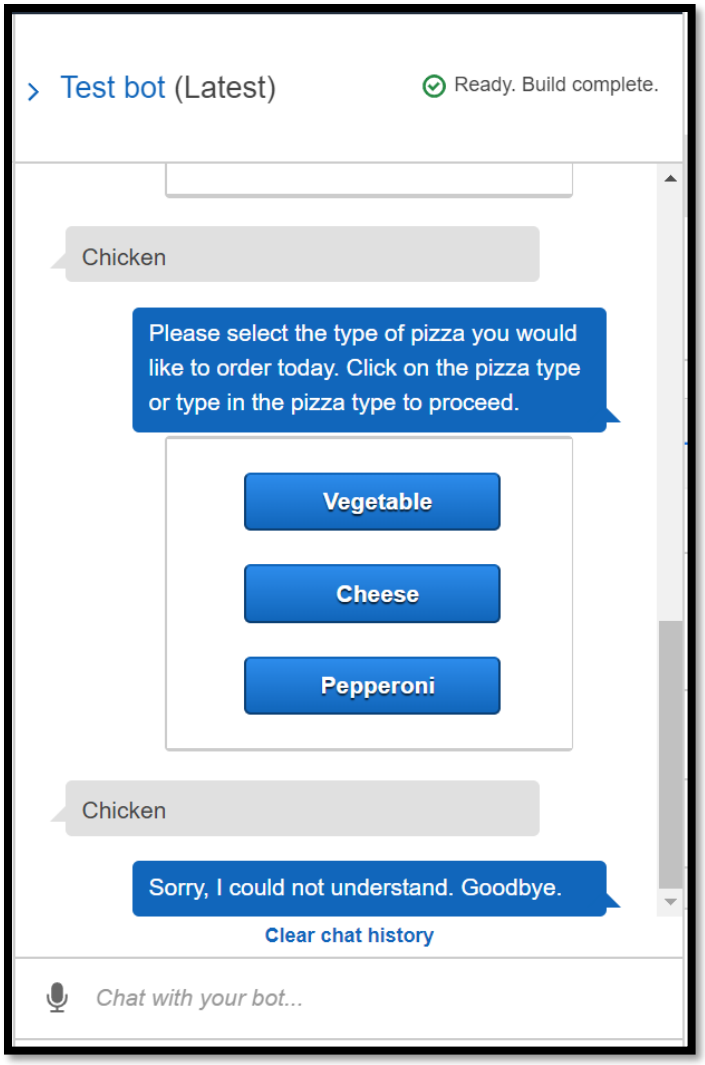


Figure 81: Test scenario 9 image 2

Operations Done

Amazon Lex enables us to build applications using a speech or text interface powered by the same technology that powers Amazon Alexa [4]. Following are the typical steps we need to perform when working with Amazon Lex [5].

Greetings Intent: It is an optional intent. The intent is used to welcome users.

Sample utterances: Hi, Hello, what is on the menu.

Slots: Name is used to store username

Confirmation prompt: Hi {Name}, Welcome to our Pizza place. Please type "Pizza" to proceed.

Control flow: Once the user types in or say Pizza the bot will go to the next intent

OrderDetails Intent: The OrderDetails Intent gets all the order details.

Sample utterances: Pizza, I would like to buy pizza, Cheese pizza

Slots: RegualrCheck to check user is ok with only regular size

Type: to get the type of pizza (cheese, vegetable, pepperoni)

Count: to store the number of pizzas that user wants

Crus: to know the type of crust that user would like to order (thick, thin)

Confirmation prompt: You have ordered {Count} regular {Type} Pizza with {Crus} crust. Please type "delivery" or "pickup" to select the type of order.

Control flow: after getting all the details, confirmation prompt will be shown to user. User has to type in delivery or pickup to take the control flow to next Intent.

Delivery Intent:If the user types in delivery at the end of OrderDetails intent then the delivery intent will be triggered.

Sample utterances: Delivery, Deliver, lets go with delivery

Slots: Address – to store user delivery address

Date - to store date of delivery

Time – to store the time of delivery

Confirmation prompt: Thanks for the order. Your pizza will be delivered to the {Address}, on {Date} at {Time}.

Control flow: The flow ends here.

PickUp Intent: If the user types in pickup at the end of OrderDetails intent then the delivery intent will be triggered.

Sample utterances: PickUp, pick, takeaway

Slots: Time – to store the time of pickup

Confirmation prompt: Thank you for the order. Your pizza will be ready for pickup at {PickUpTime}. Have a nice day.

Control flow: The flow ends here.

No Intent: In the conversation, if the user types in No at any time, the No intent will be triggered to stop the conversation.

Sample Utterances: No

Confirmation prompt: Sorry, we do no have other options today. Thank you for your time. Have a nice day.

Control flow: the flow ends here.

Error Handling: In case the user provides a response, which is not configured in the bot, then the messages from error handling will be sent as prompt to user. Added couple of error messages to the bot.

Test the bot: We can use the test window client provided by the Amazon Lex console to test the bot. The sample test scenarios and results are added in the testing section of the document.

Appendix 1: Database scripts

```
CREATE DATABASE db;
use db;

create table user_details (
        name varchar(50) not null,
        email varchar(60) not null,
        password varchar(25) not null,
        topic varchar(10) not null,
        primary key (email)
);

create table user_state (
        email varchar(60) not null,
        userstate varchar(10) not null,
        logintime date not null,
        logouttime date
);
```

Appendix 2: Codebase for Container 1: Registration

Codebase structure

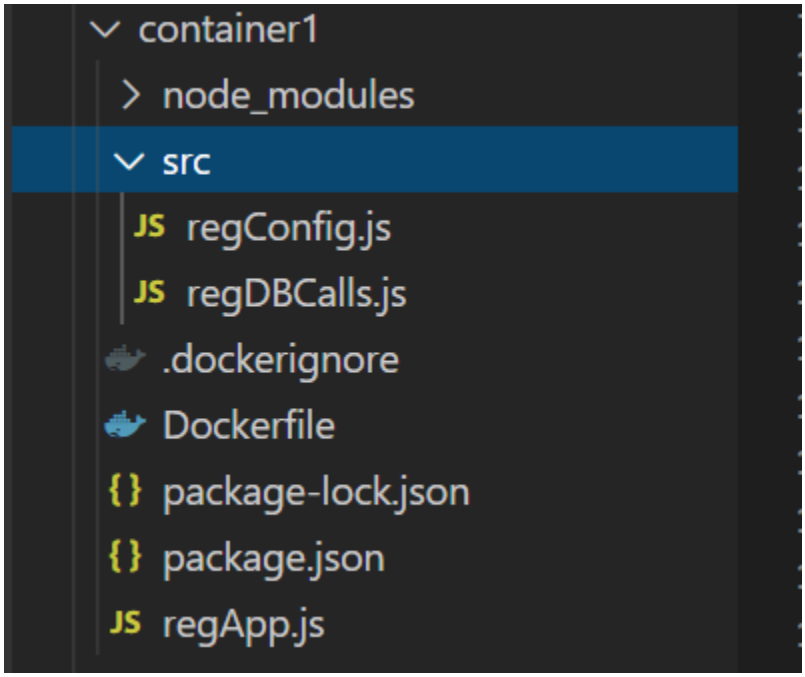


Figure 82: Codebase structure of container 1

package.json

```
{
  "name": "register",
  "version": "1.0.0",
  "description": "This container will handle all registration requests.",
  "main": "regApp.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Pratheep",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "mysql": "^2.18.1"
```

```
  }
}
```

Dockerfile

```
FROM node:12.16

WORKDIR /app

COPY package.json /app

RUN npm install

COPY  .  /app

CMD node regApp.js

EXPOSE 3001
```

regApp.js

```
const express = require('express');
const app = express();
const dbcalls = require('./src/regDBCalls');
const cors = require('cors');
const bodyParser = require('body-parser');

app.use(cors());
app.use(bodyParser.json());

app.get('/', (req, res) => {
   return res.send('Welcome to backend server. Backend server is up and running.');
});



app.post('/register', (req, res) => {

   const obj = req.body;

   dbcalls.register(obj, (output) => {
      return res.send(output);
   });
```

```
});

const port = process.env.PORT || 3001;

app.listen(port, () => {
    console.log(`listening to ${port}...`);
});

module.exports = app;
```

regDBCalls.js

```
const config = require('./regConfig');
const mysql = require('mysql');

let dbname = "use " + config.dbname;

let connec = mysql.createConnection({
    host : config.host,
    user : config.user,
    password : config.password,
    port : config.port
});

connec.connect(function(error1) {
    if (error1){
        throw error1;
    }
    connec.query(dbname, function (error2, result) {
        if (error2) {
            throw error2;
        }
    });
});

module.exports.register = function (object, callback) {

    let selectUserDetails = `select * From user_details where email = '${object.email}';`;

    connec.query(selectUserDetails, (error1, result1) => {
        if(error1){
            return callback('0');
        }
```

```
        else if(result1 && result1.length > 0) {
            return callback('5');
        }
        else {

            let insertQuery = `insert into user_details (name, email, password, topic) values
                        ('${object.name}', '${object.email}', '${object.password}', '${object.topic}');`;

            connec.query(insertQuery, (error2, result2) => {
                if(error2) {
                    return callback('0');
                }
                else{
                    return callback('6');
                }
            });
        }
    });

};
```

regConfig.js

```
module.exports = {

    host: '34.73.205.223',
    user:'root',
    password:'password',
    dbname:'db',
    port:3306

};
```

Appendix 3

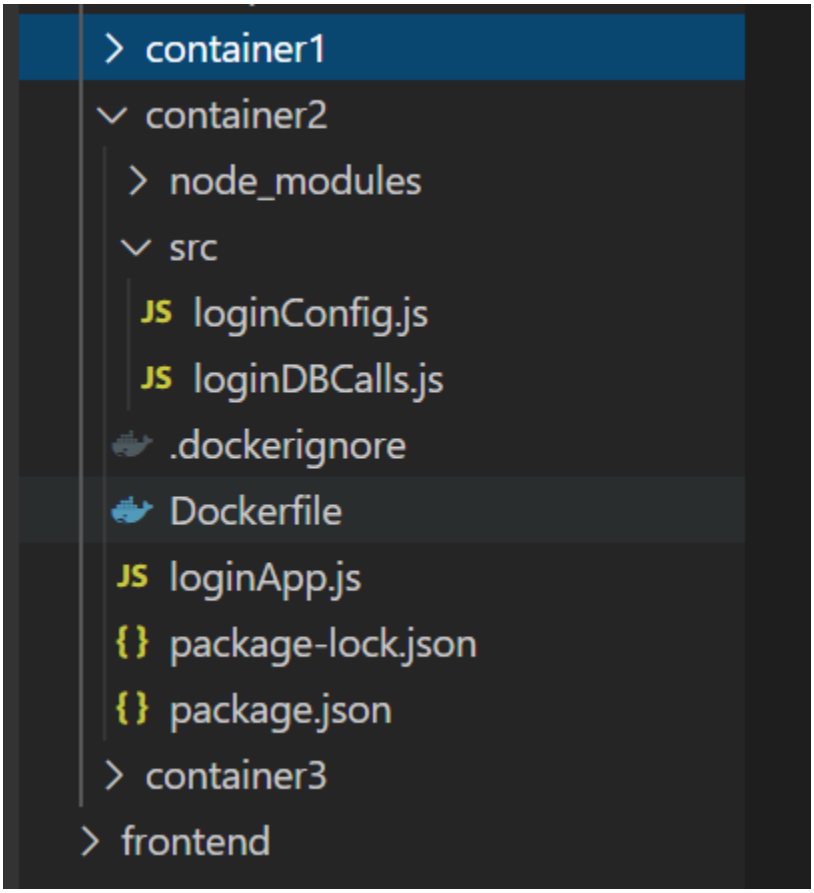


Figure 83: Codebase structure of container 2

package.json

```
{
  "name": "login",
  "version": "1.0.0",
  "description": "This container will handle all login requests.",
  "main": "loginApp.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Pratheep",
  "license": "ISC",
```

```
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "mysql": "^2.18.1"
  }
}
```

Dockerfile

```
FROM node:12.16

WORKDIR /app

COPY package.json /app

RUN npm install

COPY  .  /app

CMD node loginApp.js

EXPOSE 3002
```

loginApp.js

```
const express = require('express');
const app = express();
const dbcalls = require('./src/loginDBCalls');
const cors = require('cors');
const bodyParser = require('body-parser');

app.use(cors());

app.use(bodyParser.json());

app.get('/', (req, res) => {
    return res.send('Welcome to backend server. Backend server is up and running.');
});


app.post('/login', (req, res) => {

    console.log('login 1');
```

```
    let obj = req.body;

    dbcalls.login(obj, (output1)=>{
        console.log('login 2');
        if(output1 === '3'){
            console.log('login 3');
            obj.state='online';

            dbcalls.insertState(obj, (insertOutput) => {
                console.log('login 4');
                res.send(insertOutput);
            });
        } else {
            console.log('login 5');
            res.send(output1);
        }
    });
});


app.post('/forcelogin', (req, res) => {

    let obj = req.body;
    console.log('for ob 1', obj);
    dbcalls.updateState(obj, (output1)=>{

        if(output1 === '7'){
            console.log('for ob 2', obj);
            dbcalls.login(obj, (output2)=>{

                if(output2 === '3'){

                    obj.state= 'online';

                    dbcalls.insertState(obj, (output3) => {
                        console.log('for ob 3', obj);
                        return res.send(output3);
                    });
                } else {
                    return res.send(output2);
                }
            });
        } else {
            return res.send(output1);
        }
```

```
    });
    console.log('for ob 4', obj);
});


const port = 3002;

app.listen(port, () => {
    console.log(`listening to ${port}...`);
});

module.exports = app;
```

loginDBCalls.js

```
const config = require('./loginConfig');
const mysql = require('mysql');

let dbname = "use " + config.dbname;

let connec = mysql.createConnection({
    host : config.host,
    user : config.user,
    password : config.password,
    port : config.port
});

connec.connect(function(error1) {
    if (error1){
        console.log('error1',error1);
        throw error1;
    }
    connec.query(dbname, function (error2, result) {
        if (error2) {
            console.log('error2',error2);
            throw error2;
        }
    });
});


module.exports.login = function (object, callback) {
```

```
    let userDetailsQuery = `select * from user_details where email = '${object.email}'
                and password = '${object.password}' ;`;

    connec.query(userDetailsQuery, (error1, result1) => {

        if(error1) {
            return callback('0');
        } else if(result1 && result1.length > 0) {

            let userStateQuery = `select * From user_state where email = '${object.email}' and logouttime is null;`;
            connec.query(userStateQuery, (error2, result2) => {
                if(error2){
                    return callback('0');
                } else if(result2 && result2.length > 0){
                    return callback('2');
                } else {
                    return callback('3');
                }
            });

        } else {
            return callback('1');
        }
    });

};


module.exports.insertState = function (object, callback) {

    let insertUserState = `insert into user_state (email, userstate, logintime, logouttime) values
            ('${object.email}', '${object.state}', sysdate(), null);`;

    connec.query(insertUserState, (error1, result1) => {
        if(error1) {
            console.log(error1);
            return callback('0');
        }
        else{
            return callback('8');
        }
    });

};
```

```
module.exports.updateState = function (object, callback) {

   let updateState = `update user_state set logouttime = sysdate(), userstate='offline' where email = '${object.e
mail}'
                        and userstate = 'online';`;

       connec.query(updateState, (error1, result1) => {
          if(error1) {
             return callback('0');
          }
          else {
             return callback('7');
          }
       });

};
```

loginConfig.js

```
module.exports = {

   host: '34.73.205.223',
   user:'root',
   password:'password',
   dbname:'db',
   port:3306

};
```

Appendix 4: Codebase of Container 3-Home

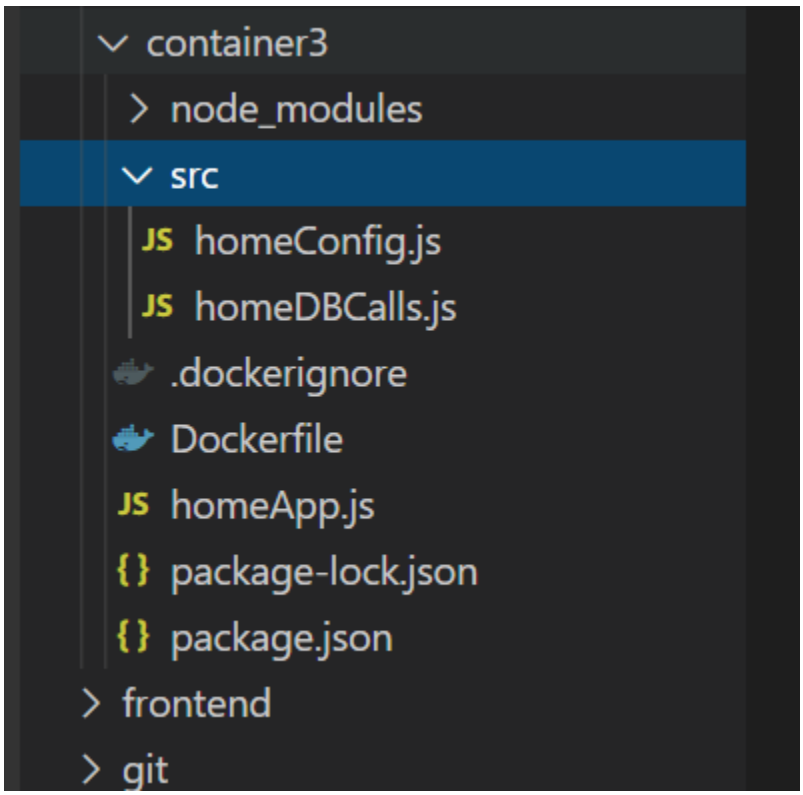


Figure 84: Codebase structure of container 3

package.json

```
{
  "name": "home",
  "version": "1.0.0",
  "description": "This container will handle all home page requests.",
  "main": "homeApp.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Pratheep",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "mysql": "^2.18.1"
  }
```

```
}
```

Dockerfile

```
FROM node:12.16

WORKDIR /app

COPY package.json /app

RUN npm install

COPY . /app

CMD node homeApp.js

EXPOSE 3003
```

homeApp.js

```
const express = require('express');
const app = express();
const dbcalls = require('./src/homeDBCalls');
const cors = require('cors');
const bodyParser = require('body-parser');

app.use(cors());
app.use(bodyParser.json());

app.get('/', (req, res) => {
    return res.send('Welcome to backend server. Backend server is up and running.');
});

app.get('/home', (req, res) => {

    console.log('get home');
    let returnObj;

    dbcalls.getUserDetails((getUserDetailsOutput) => {
        if(getUserDetailsOutput !== '0' || getUserDetailsOutput !== 0){
            returnObj = {
```

```
            status: '9',
            records: getUserDetailsOutput
          }
          console.log('returnObj', returnObj);
          return res.send(returnObj);
        } else {
          returnObj = {
            status: '0'
          }
          console.log('returnObj', returnObj);
          return res.send(returnObj);
        }
    });
});


app.post('/logout', (req, res) => {

    console.log('logout');
    console.log('req', req);

    let obj = req.body;

    console.log('obj', obj);

    console.log('email', obj.email);

    dbcalls.logout(obj.email, (output1)=>{
        return res.send(output1);
    });
});

const port = process.env.PORT || 3003;

app.listen(port, () => {
    console.log(`listening to ${port}...`);
});

module.exports = app;
```

homeDBCalls.js

```
const config = require('./homeConfig');
const mysql = require('mysql');
```

```
let dbname = "use " + config.dbname;

let connec = mysql.createConnection({
   host : config.host,
   user : config.user,
   password : config.password,
   port : config.port
});


connec.connect(function(error1) {
   if (error1){
      throw error1;
   }
   connec.query(dbname, function (error2, result) {
      if (error2) {
         throw error2;
      }
   });
});

module.exports.logout = function (email, callback) {

   let updateState = `update user_state set logouttime = sysdate(), userstate='offline' where email = '${email}'
                        and userstate = 'online';`;

   connec.query(updateState, (error1, result1) => {
      if(error1) {
         return callback('0');
      }
      else {
         return callback('12');
      }
   });
};


module.exports.getUserDetails = function (callback) {

   let getUserDetailsQuery = `select dtls.name as name, dtls.email as email from user_state state, user_details d
tls
                           where dtls.email = state.email and state.userstate = 'online';`;

   connec.query(getUserDetailsQuery, (error, result) => {
```

```
        if(error){
            return callback('0');
        } else {
            return callback(result);
        }
    });

}
```

homeConfig.js

```
module.exports = {

    host: '34.73.205.223',
    user:'root',
    password:'password',
    dbname:'db',
    port:3306

};
```

Appendix 5: Codebase of Container 4-Frontend

Referenced react tutorial from react website [6].

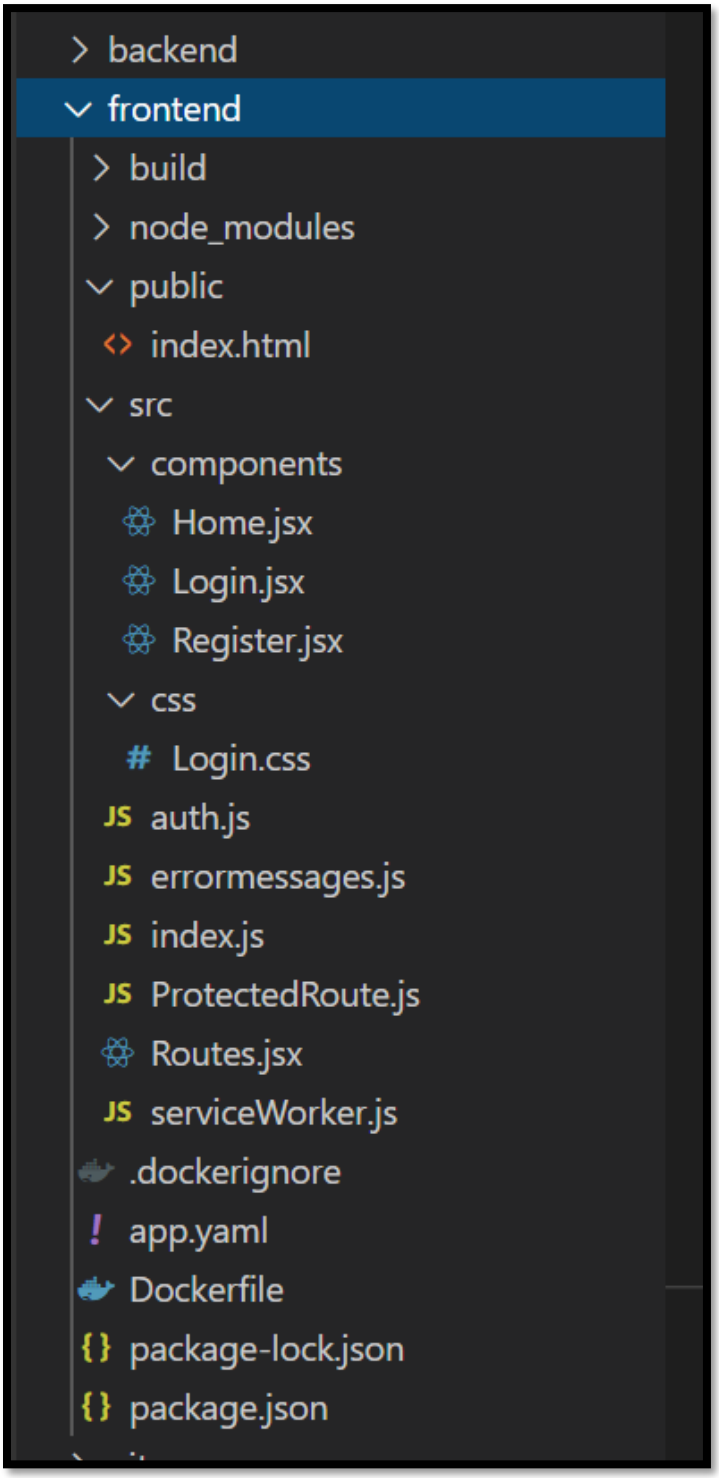


Figure 85: Codebase structure of container 4

package.json

```
{
  "name": "frontend",
  "version": "1.0.0",
  "description": "The frontend part of the serverless assignment.",
  "main": "index.js",
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "author": "Pratheep",
  "license": "ISC",
  "dependencies": {
    "axios": "^0.19.2",
    "boostrap": "^2.0.0",
    "react": "^16.13.1",
    "react-bootstrap": "^1.0.1",
    "react-dom": "^16.13.1",
    "react-native": "^0.62.2",
    "react-router-dom": "^5.2.0",
    "react-scripts": "^3.4.1"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Dockerfile

```
FROM node:12.16

WORKDIR /app

COPY package.json .

RUN npm install

COPY  . .

CMD ["npm", "start"]
```

Home.jsx

```
import React, { Component } from "react";
import auth from "../auth";
import '../css/Login.css';
import axios from 'axios';

class Home extends Component {

    constructor(props){
      super(props);
      this.state = {
         email: this.props.location.state.email,
         currentUser: '',
         onlineUserList: [],
      };

      this.backendCall();

      console.log('3', this.state);
   }

   async backendCall(){

      console.log('backendCall');
      let config = {
         headers: {
            'Content-Type' : 'application/json'
      }};
```

```
      const res = await axios.get(`https://home-co5z3zrdzq-
ue.a.run.app/home`, JSON.stringify(this.state), config);
      console.log('res', res);
      let data = res.data;
      let status = data.status;
      if(data){
        if(status === '9' || status === 9){
          let records = data.records;

          this.prepareDetails(records);

        }
      }


  }

   prepareDetails(records){

      let i = 0;
      let onlineUserList=[];
      let currentUser;

      for(i=0; i<records.length; i++){
        if(records[i].email === this.state.email){
          currentUser = records[i].name;
        } else {
          onlineUserList.push(records[i].name);
        }
      }

      this.setState({
        onlineUserList: onlineUserList,
        currentUser: currentUser
      });
      console.log('this.state 2',this.state);

  }

  async logoutBackend() {

      let config = {
        headers: {
           'Content-Type' : 'application/json'
        }
```

```
    };
    console.log('this.state.email logout', this.state.email);
    const res = await axios.post('https://home-co5z3zrdzq-
ue.a.run.app/logout', JSON.stringify(this.state), config);
    let data = res.data;
    if(data === '12' || data === 12){

      auth.logout( () => {
        this.props.history.push({
          pathname: '/',
        });
      });
    }
  }

  onLogout = (e) => {

    e.preventDefault();
    this.logoutBackend();

  };

  render(){
    return(

      <div>

      <div className="homebox">
      <div className="home">

        <form>

          <div>
            <h4 className="homeMsg">Hi, {this.state.currentUser} you are logged in.</h4>
          </div>

          <div>
            <button className="myButton" onClick={this.onLogout}>Logout</button>
          </div>
          <p />
          <div className="form-group">
            {this.state.onlineUserList.length > 0 ?
              <p>Here are the other users who are online
                {this.state.onlineUserList.map((result) => (
                <p>{result}</p>
```

```
                    ))}</p>
                  : <p>No users are online currently.</p>
                  }
              </div>
            </form>
            </div>
          </div>
        </div>
        );
    }
}

export default Home;
```

Login.jsx

```
import React, { Component } from 'react';
import '../css/Login.css';
import axios from 'axios';
import errMsg from '../errormessages';
import auth from '../auth';

class Login extends Component {

    constructor(props){
        super(props);

        this.state = {
            email: '',
            password: '',
            flag: false,
            errorMsg:'',
            forceLogin: false,
            userList:[],
        }

    }

    componentDidMount(){

    }
```

```
  onChange = (e) => {
    this.setState({
      [e.target.name]: e.target.value
    });
  }

  async forceLoginBackend(){
    var config = {
      headers: {
        'Content-Type' : 'application/json',
        'Access-Control-Allow-Origin': '*'
      }};
    try{
      //const res = await axios.post('http://192.168.99.100:3002/forcelogin', JSON.stringify(this.state), config
);
      //const res = await axios.post('http://localhost:3002/forcelogin', JSON.stringify(this.state), config);
      const res = await axios.post('https://login-co5z3zrdzq-
ue.a.run.app/forcelogin', JSON.stringify(this.state), config);
      let data = res.data;

      if(data === '8' || data === 8){

        auth.login( () => {
          this.props.history.push({
            pathname: '/home',
            state: {
              email : this.state.email,
            }
          });
        });
        this.setState({
          email:'',
          password:''
        });

      } else{
        if(data === '2' || data === 2){
          this.setState({
            forceLogin: true
          });
        }
        this.setState({
          errorMsg: errMsg[data]
        });
      }
```

```
      } catch(err){
         console.log(err);
      }
   }

   async backendCall(){
      let config = {
         headers: {
            'Content-Type' : 'application/json',
           //  'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Methods':'GET,PUT,POST,DELETE,PATCH,OPTIONS',
         }
      };
      try{
         //const res = await axios.post('http://192.168.99.100:3002/login', JSON.stringify(this.state), config);
         //const res = await axios.post('http://localhost:3002/login', JSON.stringify(this.state), config);
         const res = await axios.post('https://login-co5z3zrdzq-
ue.a.run.app/login', JSON.stringify(this.state), config);

         let data = res.data;
         console.log('data',data);
         if(data){
            if(data === 8 || data === '8'){

               auth.login( () => {
                  this.props.history.push({
                     pathname: '/home',
                     state: {
                        email : this.state.email
                     }
                  });
               });
               this.setState({
                  email:'',
                  password:''
               });

            } else{
               if(data === 2 || data === '2'){
                  console.log('inside 2 block');
                  this.setState({
                     forceLogin: true
                  });
               }
               this.setState({
```

```
                    errorMsg: errMsg[data]
                });
            }
        } else {
            this.setState({
                errorMsg: errMsg['10']
            });
        }
    } catch(err){
        console.log(err);
        this.setState({
            errorMsg: errMsg['11']
        });
    }
}

onForceLogin = (e) => {
    e.preventDefault();
    this.setState({
        flag: false,
        errorMsg:''
    });

    if(this.state && this.state.email && this.state.password &&
        this.state !== {} && this.state.email !== '' && this.state.password !== ''){
            this.forceLoginBackend();
    } else {
        this.setState({
            flag:true,
            errorMsg: errMsg['13']
        });
    }
}

onLogin = (e) => {
    e.preventDefault();
    this.setState({
        flag: false,
        errorMsg:''
    });

    if(this.state && this.state.email && this.state.password &&
        this.state !== {} && this.state.email !== '' && this.state.password !== ''){
            this.backendCall();
    } else {
```

```
        this.setState({
          flag:true,
          errorMsg: errMsg['13']
        });
    }
}

render() {
    return(
        <div>
          <div className="loginbox">
          <div className="login">

              <form>


                <div>
                  <p style={{ color: "red" }}>
                    {this.state.errorMsg ? this.state.errorMsg:null}
                  </p>

                  {this.state.forceLogin?
                  <button type="submit" onClick={this.onForceLogin}
                    className="myButton">Force Login</button>
                  :null}
                </div>

                <p/>
                <h3>Login</h3>

                <div className="form-group">
                  <label>Email address</label>
                </div>
                <div className="form-group">
                  <input type="email" name="email" onChange={this.onChange}
                    className="input" placeholder="Enter email" />
                </div>

                <div className="form-group">
                  <label>Password</label>
                </div>
                <div className="form-group">
                  <input type="password" name="password" onChange={this.onChange}
                    className="input" placeholder="Enter password" />
                </div>
```

```
                    <p/>
                    <button type="submit" onClick={this.onLogin}
                        className="myButton">Login</button>

                    <p className="register text-left">
                        Don't have an account yet? Register <a href="/register">here?</a>
                    </p>

                </form>
                </div>
            </div>
        </div>
      );
    }
}

export default Login;
```

Register.jsx

```
import React, { Component } from 'react';
import axios from 'axios';
import { Redirect } from "react-router-dom";
import errMsg from '../errormessages';


class Register extends Component {

    constructor(props){
        super(props);

        this.state = {
            name:'',
            email:'',
            password:'',
            topic:'',
            validateFlag:'',
            registerFlag: false,
            validEmail:true,
            errorMsg:'',
        }
    }
```

```
onChange = (e) => {
    this.setState ( {
        [e.target.name] : e.target.value
    } );
}

async apiCall(){
    var config = {
        headers: { 'Content-Type' : 'application/json' }
       };

    try{
        let mess= '';
        //await axios.post('http://192.168.99.100:3001/register', JSON.stringify(this.state), config)
        //await axios.post('http://localhost:3001/register', JSON.stringify(this.state), config)
        await axios.post('https://register-co5z3zrdzq-ue.a.run.app/register', JSON.stringify(this.state), config)
                .then(response => {
                    console.log('response',response);
                    console.log('response.data',response.data);
                    mess=response.data;
                    return response.data
                })
                .catch((err) => {
                    console.log('err',err);
                    console.log('err.response',err.response);
                    console.log('err.response.data',err.response.data);
                    mess=err.response.data;
                    throw err.response.data
                });

        if(mess === 6){

            alert(
                "Registration is success. Click ok to login"
            );
            this.setState({
                name:'',
                email:'',
                password:'',
                topic:'',
                registerFlag:true
            });
        } else {
            this.setState({
                validateFlag: true,
```

```
                errorMsg: errMsg[mess]
            });
        }

    } catch(err){
        console.log(err);
        this.setState({
            validateFlag: true,
            errorMsg: errMsg['11']
        });
    }
}

onRegister = (e) => {
    e.preventDefault();

    this.setState({
        validateFlag: false
    });

    const re = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
    let validEmail = re.test(String(this.state.email).toLowerCase());

    if(!validEmail){
        this.setState({
            validEmail:validEmail,
            errorMsg: errMsg['14']
        });
    }

    if(this.state && this.state.name && this.state.email && this.state.password && this.state.topic &&
        this.state!=={} && this.state.name !==" && this.state.email !== " && validEmail
            && this.state.password !== " && this.state.topic !== "){

            this.apiCall();

    } else {
        this.setState({
            validateFlag: true,
            errorMsg:errMsg['13']
        });
    }
}
```

```
render() {

   return(
      <div>

         <div className="registerbox">

         {this.state.registerFlag ?

         <div>
            <Redirect to ='/login' />
         </div>
         :null}

         <div className="registerDiv">
            <div className="registerCard">
               <div className="registerCard-body">

                  <form >

                     {this.state.validateFlag &&
                        <p className="errorMsg" style={{ color: "red" }}>
                           {this.state.errorMsg}
                        </p>
                     }

                     <h3>Register</h3>

                     <div className="form-group">
                        <label>Name*</label>
                     </div>

                     <div className="form-group">
                        <input required type="name" name="name"
                           onChange={this.onChange} value={this.state.name}
                           className="input" placeholder="Enter your name"
                           />
                     </div>

                     <div className="form-group">
                        <label>Email Address*</label>
                     </div>

                     <div className="form-group">
                        <input required type="email" name="email"
```

```
                        onChange={this.onChange}
                        className="input" placeholder="Enter email address" />
                    </div>

                    {!this.state.validEmail &&
                      <p className="errorMsg" style={{ color: "red" }}>
                        Email address is invalid.
                      </p>
                    }

                    <div className="form-group">
                      <label>Password*</label>
                    </div>

                    <div className="form-group">
                      <input required type="password" name="password"
                        onChange={this.onChange}
                        className="input" placeholder="Enter password" />
                    </div>

                    <div className="form-group">
                      <label>Select Topic*</label>
                      <p/>
                    </div>

                    <div className="form-group">
                      <select required type="select" name="topic"
                        onChange={this.onChange}
                        className="select-css" >

                        <option value="">select</option>
                        <option value="developer">AWS</option>
                        <option value="manager">GCP</option>
                        <option value="manager">Azure</option>

                      </select>
                    </div>
                    <p></p>
                    <button onClick={this.onRegister} type="submit" className="myButton">Register</butt
on>

                  </form>

                </div>
              </div>
```

```
            </div>
            </div>
          </div>
      );
    }
}

export default Register;
```

References

[1] Docker, “What is a Container,” [Online]. Available: https://www.docker.com/resources/what-container. [Accessed: 08-Jun-2020].

[2] Google Cloud, “Container registry,” [Online]. Available: https://cloud.google.com/container-registry. [Accessed: 07-Jun-2020].

[3] Google Cloud, “Cloud Run,” [Online]. Available: https://cloud.google.com/run. [Accessed: 08-Jun-2020].

[4] AWS, “Amazon Lex: How it works,” [Online]. Available: https://docs.aws.amazon.com/lex/latest/dg/how-it-works.html. [Accessed: 07-Jun-2020].

[5] Saurabh Dey, “How to create chatbot,” Class Lecture Module 3 on May 25, 2020, Dalhousie University, Canada. [Accessed: 04-Jun-2020].

[6] React JS, “Create a New React App,” [Online]. Available: https://reactjs.org/docs/create-a-new-react-app.html. [Accessed: 17-May-2020].