FINAL PROJECT REPORT

CSCI 5410 Serverless Data Processing

Submitted To: Dr. Saurabh Dey

Submitted By: Group 2

Ketul Patel - B00900957

Faiza Umatiya – B00899642

Parul Raich - B00903203

Contents

Table of figures

Table of contents

1. Feature Specifications

1.1 User Management Module

1.1.1 Registration

The system will offer the registration process for multiple users at the same time. 3-Factor authentication system will be implemented to be to register user successfully database. Firstly, users will provide information about the type of user, email, password, name, address, and phone number. Then, email address and password will be validated and stored in Cognito User Pools using **AWS Cognito** [4]. The second factor will be questions and answers, where the user needs to choose between the provided 3 questions and provide the answer for the selected question. I have stored questions and answers for that user in **Firstore** [3] database using **CloudFunction** [5]. Lastly, the User provided 4-character key and plain text will be stored in **DynamoDB** [3] database using Lambda function. Following, User will receive cipher text computed from lambda function using Columnar Cipher that can be used during authentication.

Test Cases:

Table 1: Test case scenarios for Registration Module

Test Scenarios	Expected behavior
Type of user is not selected.	Alert user with proper message.
Name is in invalid format.	Alert user with proper message.
Email-id is in invalid format.	Alert user with proper message.
Phone number in invalid format.	Alert user with proper message.
Password is in invalid format.	Alert user with proper message.
Address is in invalid format.	Alert user with proper message.
Email-id already exists.	Alert user with proper message.
All basic details are valid.	Send user to next sign-up process.
Question not selected in question-answer.	Alert user with proper message.
Answer is not valid in question-answer.	Alert user with proper message.
Question-Answer is valid.	Store selected question and respected answer in **Firestore**[3] for user and send user to next sign-up process.
Character key is in invalid format.	Alert user with proper message.
Plaint text is in invalid format.	Alert user with proper message.

Character key and plain text are valid.	Send use cipher text and store key and plain text in **DynamoDB** [3].
Successfully Registered user.	Provide Success information with proper message and redirect user to login page.

Screenshots for test cases:

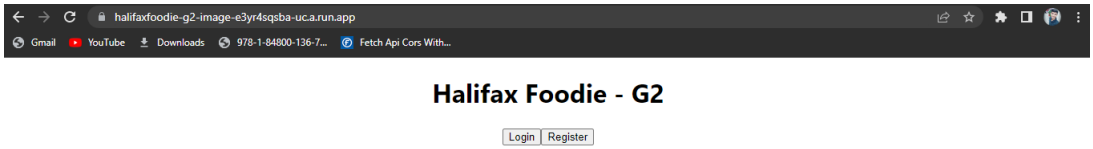

Figure 1: Home page of the application.

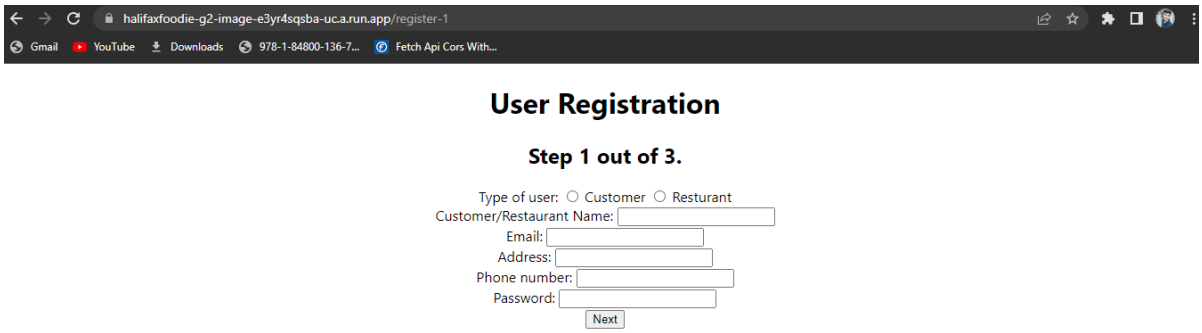

Figure 2: Register 1st page.

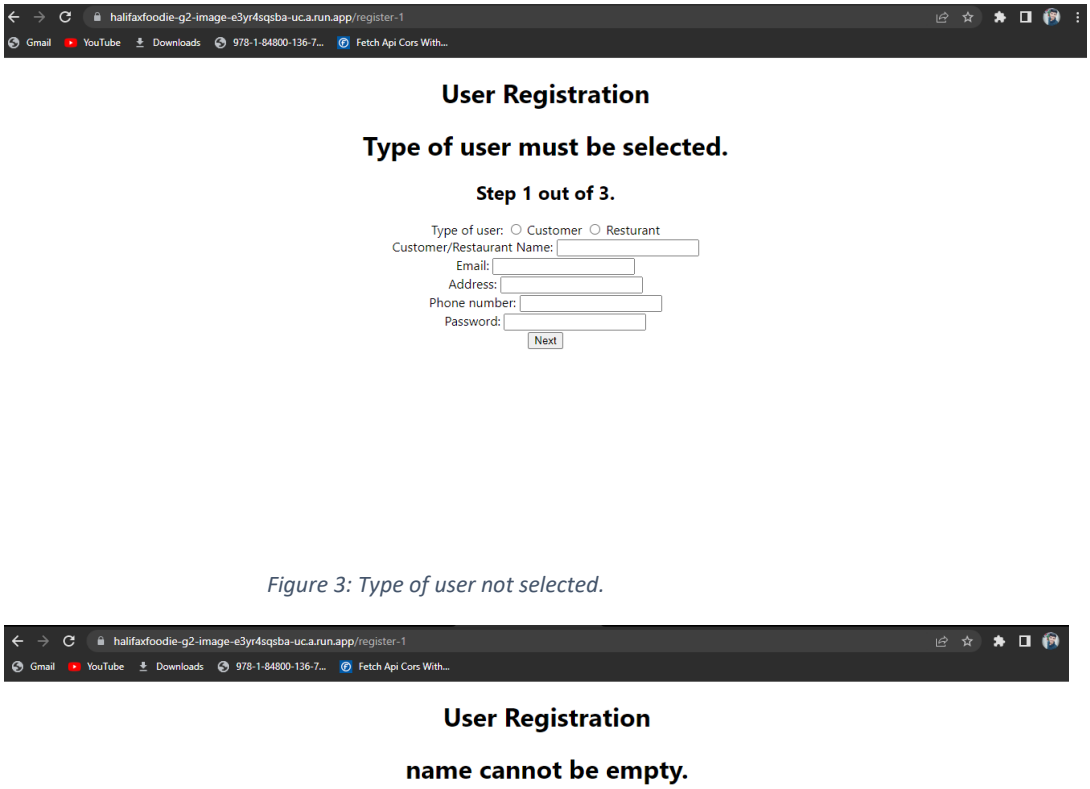

Figure 3: Type of user not selected.

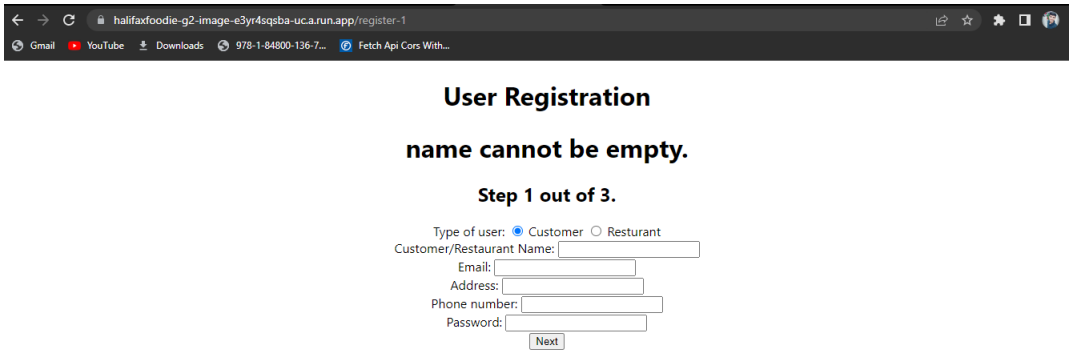

Figure 4: Name is empty.

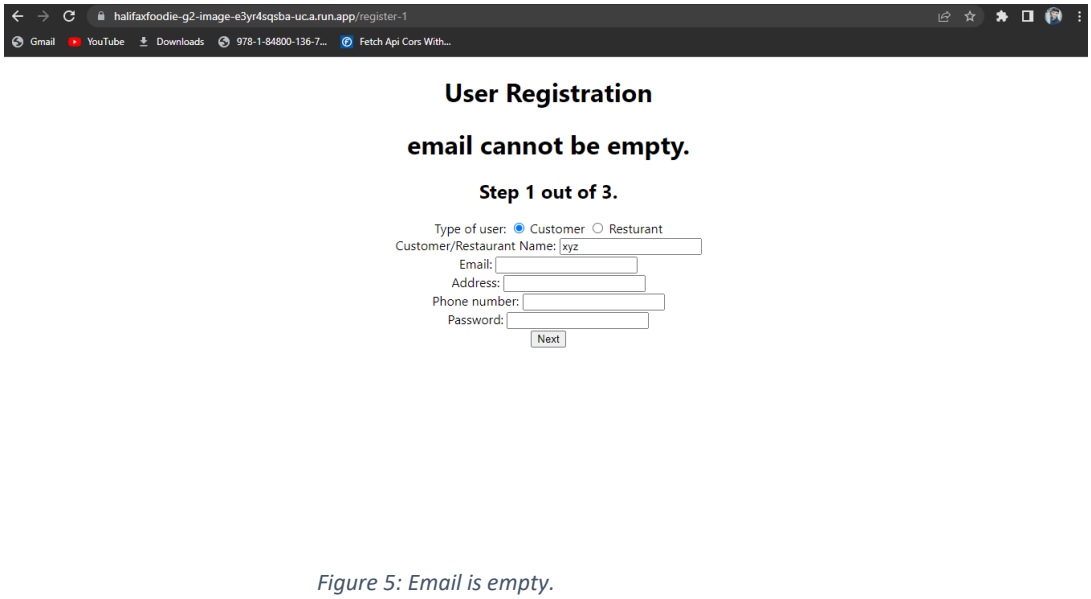

Figure 5: Email is empty.

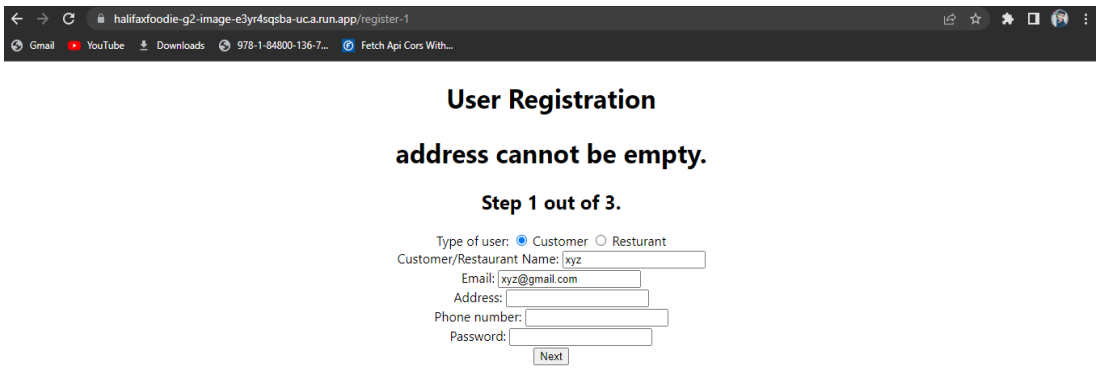

Figure 6: Address is empty.

Figure 7: Phone number is not 10 digits.

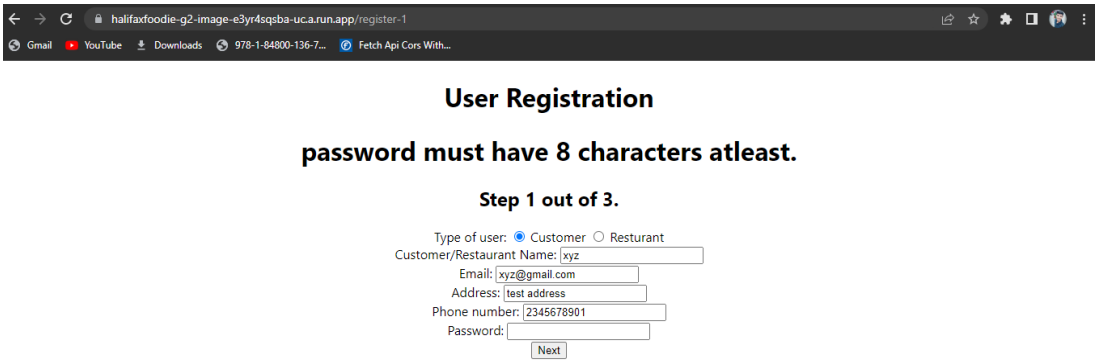

Figure 8: Password do not have 8 characters.

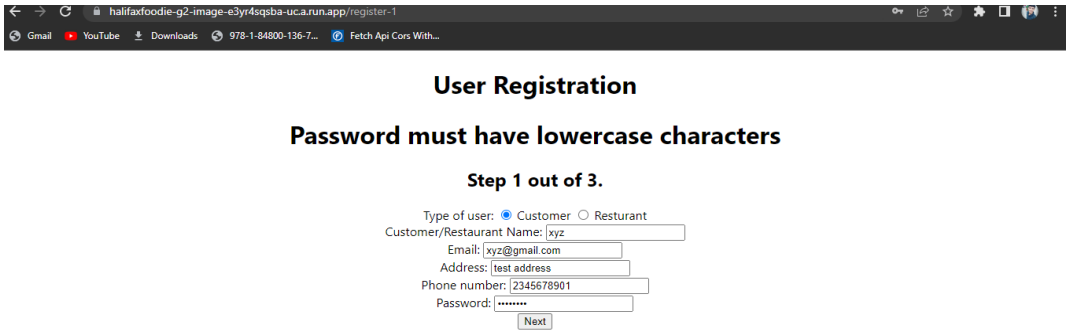

Figure 9: Password do not have lower letter.

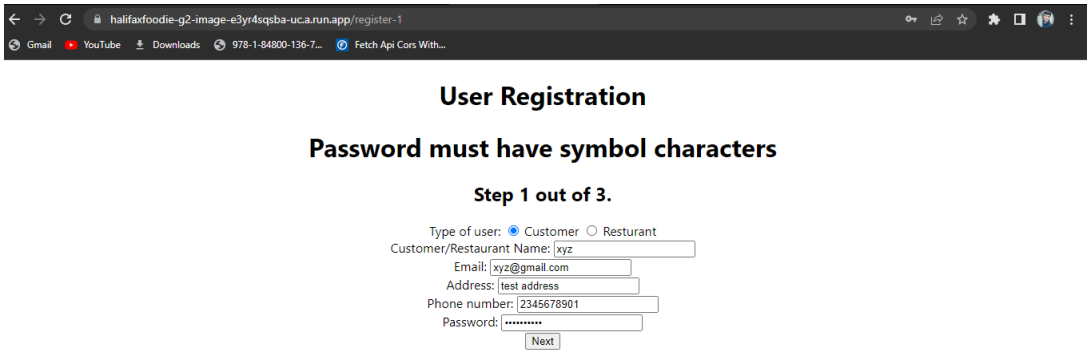

Figure 10: Password do not have symbol character.

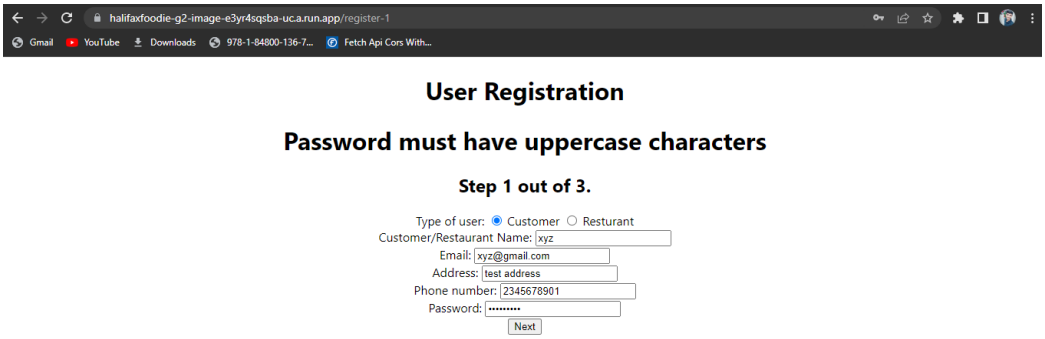

Figure 11: Password do not have upper character.

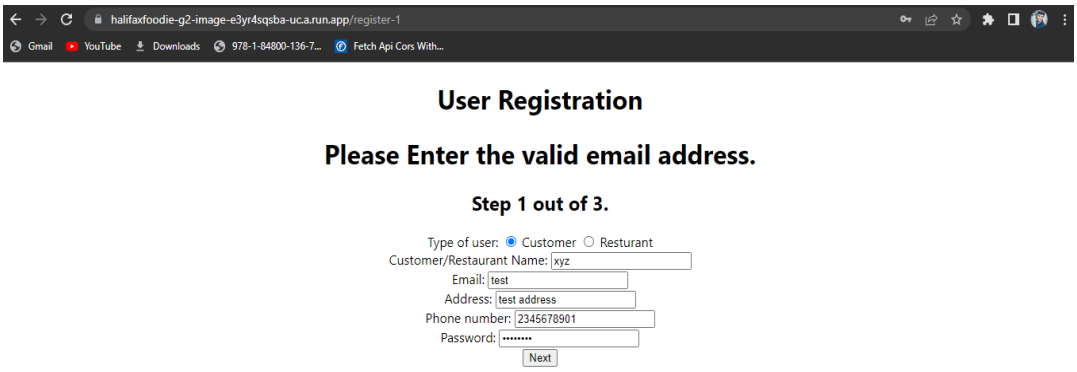

Figure 12: Email address is not valid.

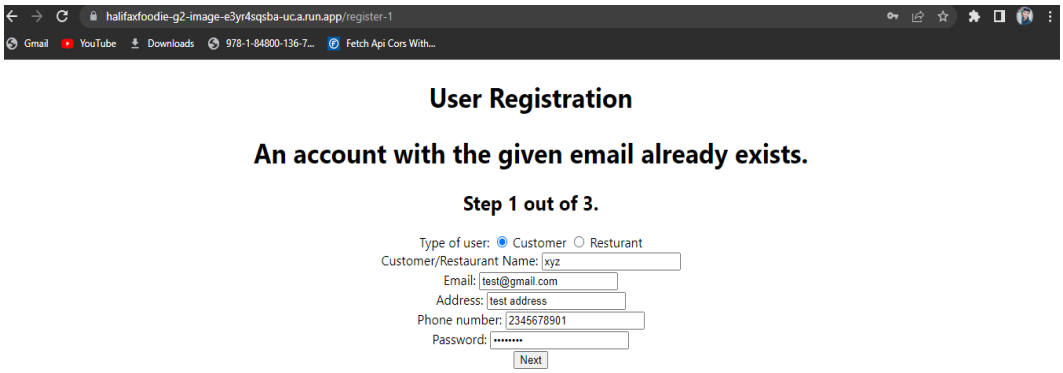

Figure 13: Email address already exist.

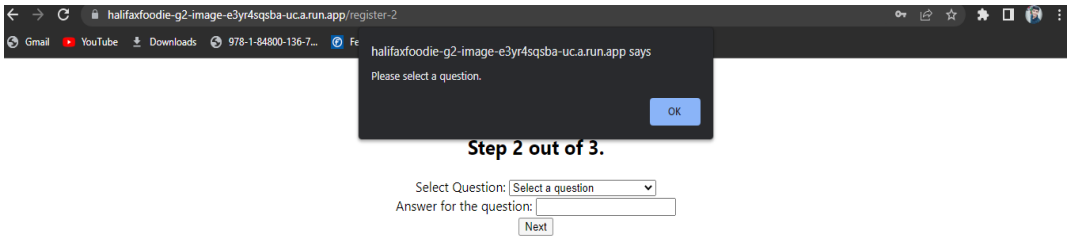

Figure 14: Question not selected.

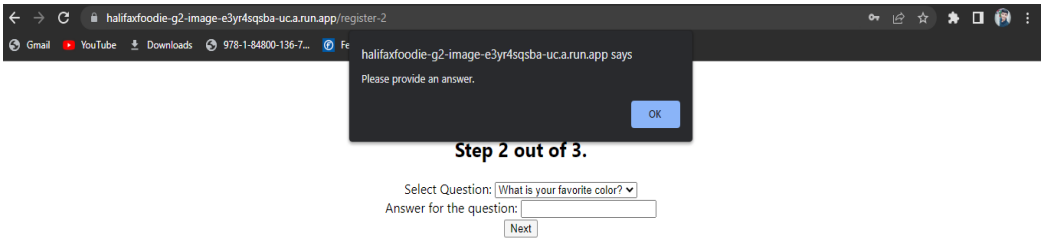

Figure 15: Answer not provided.

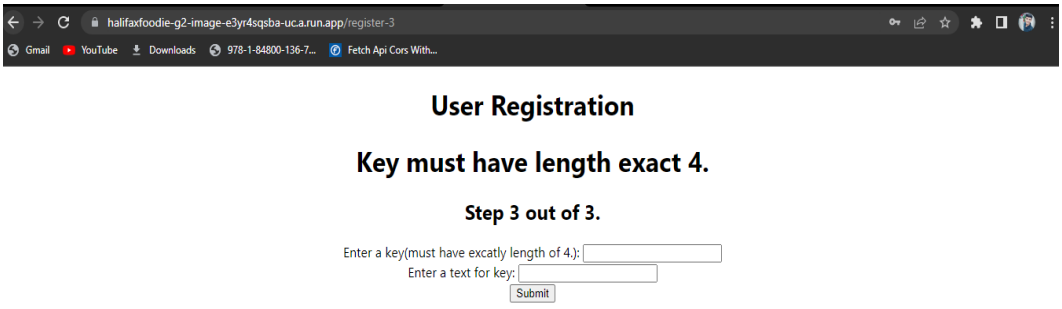

Figure 16: Key length is not equal to 4.

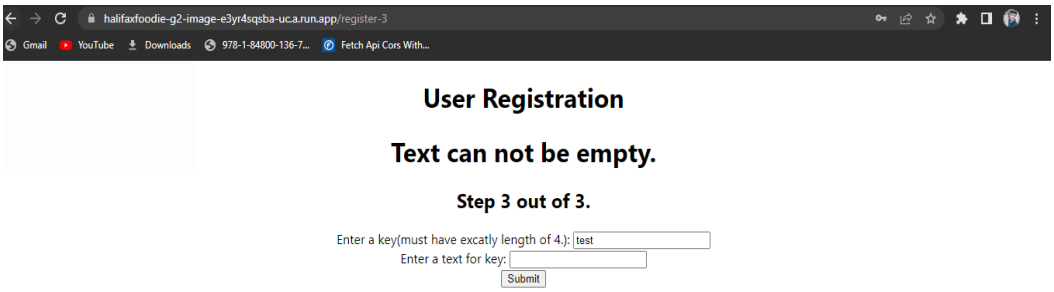

Figure 17: Provided text is empty.

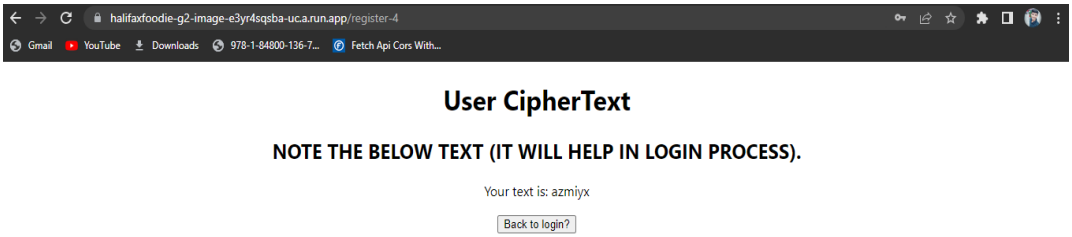

Figure 18: Everything is correct in registration.

1.1.2 Authentication

The login page for the serverless Food service web app is where the user ends up after registering. The user must initially register to use all of the system's features. Upon entering the app, the registered user authenticates themselves. There are multiple steps involved in the authentication.

Firstly, the user needs to provide email and password for the 1st step of Authentication. The authentication was completed using "**AWS Cognito"** [4]. The control shifts to the second layer of security if the first step is completed successfully. Then, the cloud function will be triggered to find the question selected by user in **Firestore** [3] during registration and displayed to the user. After the user provides the answer for the question, correctness of answers will be checked using "**GCP Cloud Functions"** [5] and which carry out question-answer checking, are now in charge in this tier [5].

After this layer is successfully passed, "**columnar cypher"** validation is carried out [14]. "**AWS Lamba"** [11] are used for this validation. The user will successfully log in once all three layers have been verified and directed to the home page according to the type of user.

Test Cases:

Table 2: Test case scenarios for Authentication Module

Test case	Expected behavior
User enters invalid ID or password during first stage of authentication.	Return to login page and alert user to enter proper login credentials.
User enters proper login credentials during first stage of authentication.	Direct user to 2nd factor authentication page.
User enters incorrect answers during 2nd factor authentication.	Alert user with message stating "incorrect answer" asks to re-enter again.
User enters correct answer during 2nd factor authentication.	Direct user to 3rd factor authentication page.
User enters an incorrect key or plain text during 3rd factor authentication.	Alert user by stating "login failed".

Screenshots for test cases:

Figure 19: Home page of login.

Figure 20: Incorrect email address.

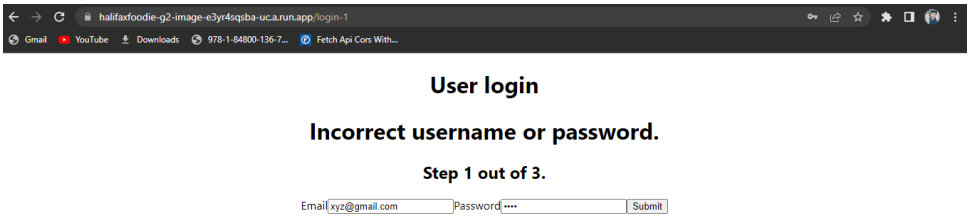

Figure 21: Incorrect password.

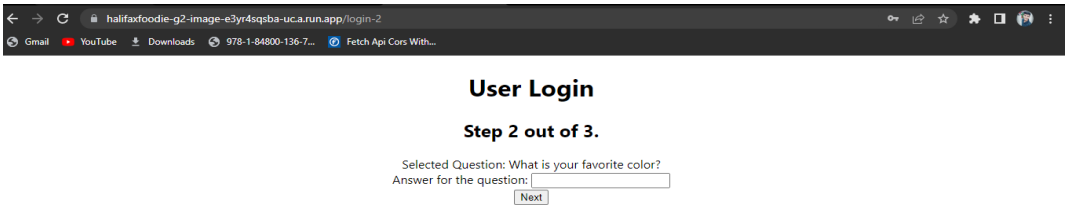

Figure 22: Correct email and password.

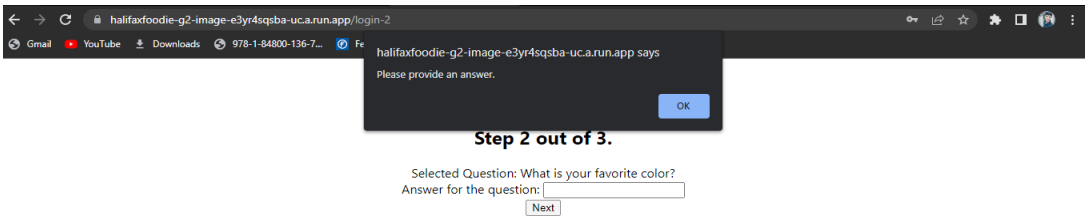

Figure 23: Answer is not provided.

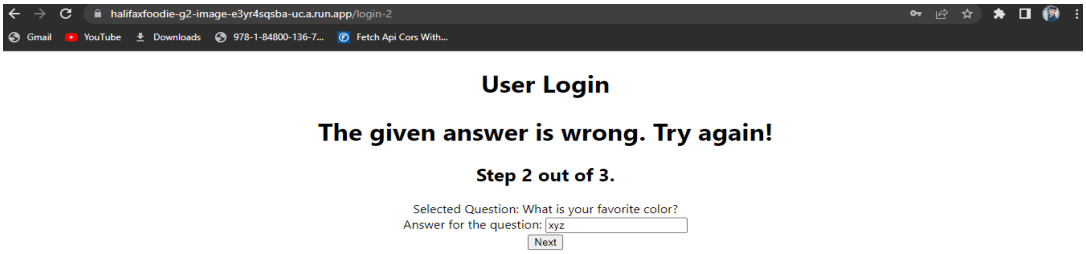

Figure 24: Answer is incorrect.

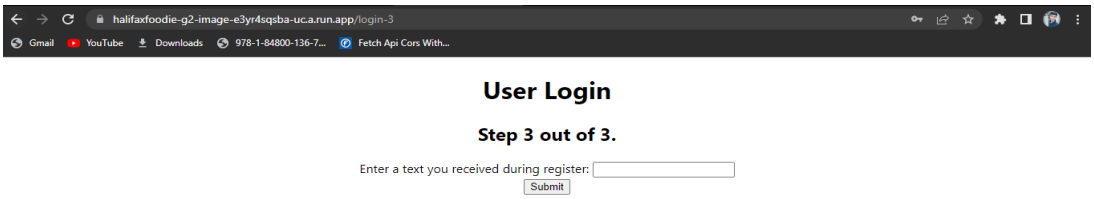

Figure 25: Answer is correct.

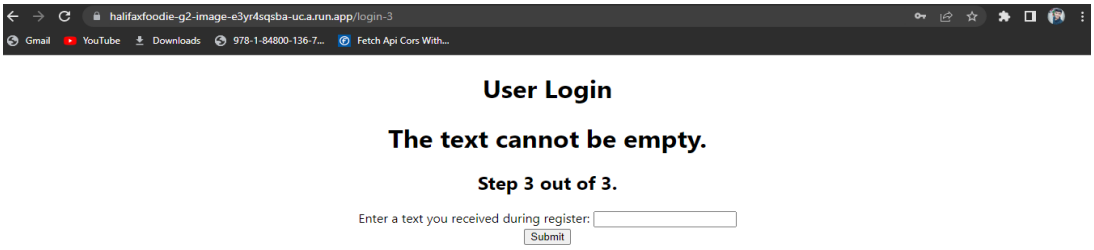

Figure 26: Text is empty.

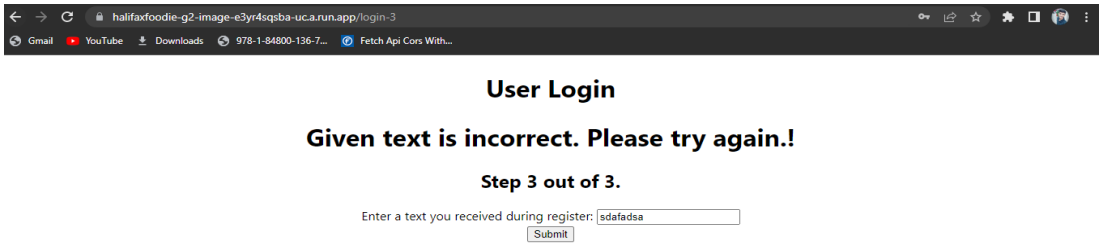

Figure 27: Text is incorrect.

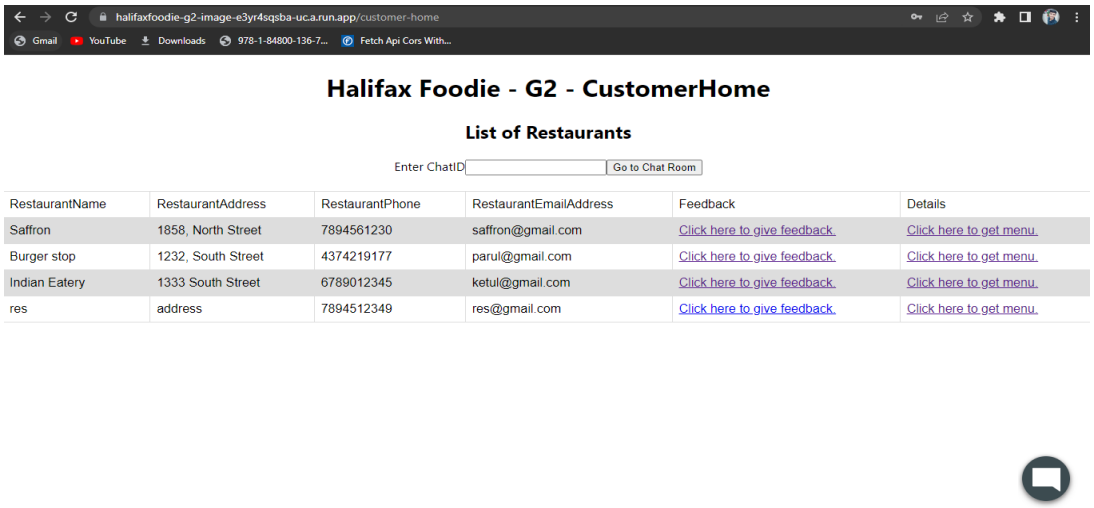

Figure 28: Everything is correct, and User is Customer.

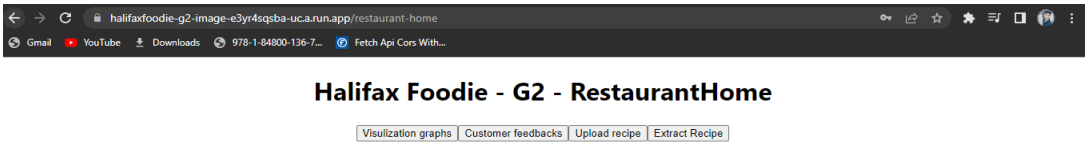

Figure 29: Everything is correct, and User is Restaurant owner.

1.2 Online Support Module

This module was created using the cloud services "**AWS Lex"** [13], and "**AWS Lambda"** Function [11].

The module's primary goal is to provide the user with virtual support by sending out bots to address all of their questions. All user types can make use of the func tionality.

The chatbot verifies a user's identification when they register before allowing them to continue their order online. These users can rate an order based on its order number and ask for navigational assistance. Visitors can inquire about directions from the chatbot. Restaurant owners should be allowed to provide information in their dynamic responses regarding the ingredients they use as well as their expenses. If the customer or user's question is regarding the complaint, the chatbot should start a new conversation using the chat module.

Implementation Details:

- After users register, the bot will assist them with navigation and solve any registration problems. The bot will authenticate the userID for logged-in users. After authentication, the bot will respond to the queries regarding order tracking and reviews. After "**AWS Lambda"** is activated [11], All of the data is retrieved using "**AWS DynamoDB"** [3]. Following receipt of all the information from the client, a second AWS Lambda function will be called, which will display all the tracking data. The names and costs of recipes can be added by restaurant owners. The name of the recipe will be entered and inserted into the database using an "**AWS Lambda"** function [11].
- If the problem or complaint is related to an order, the bot will end the current virtual help session and offer the "Chat with our associate" option. Once the user clicks the chat button, the chat module will start.

Test Cases:

Table 3: Test case scenarios for Online Support Module

Test Scenarios	Expected behavior
Customer tracking the online order	Bot should validate the UserID and provide the required information.
The owner tries to add the recipe and its prices along with the recipe types.	Bot should first validate the restaurant owner and then guide the restaurant owner to add recipes,

	name and price with dynamic response.
Customer adding food order rating.	Bot should validate UserID and asks customer to add rating.

Screenshots for test cases:

Test case – 1: Customer giving food rating.

Step-1: I created intents for the bot **"Online Virtual Assistant"** which is shown in the fig 30.

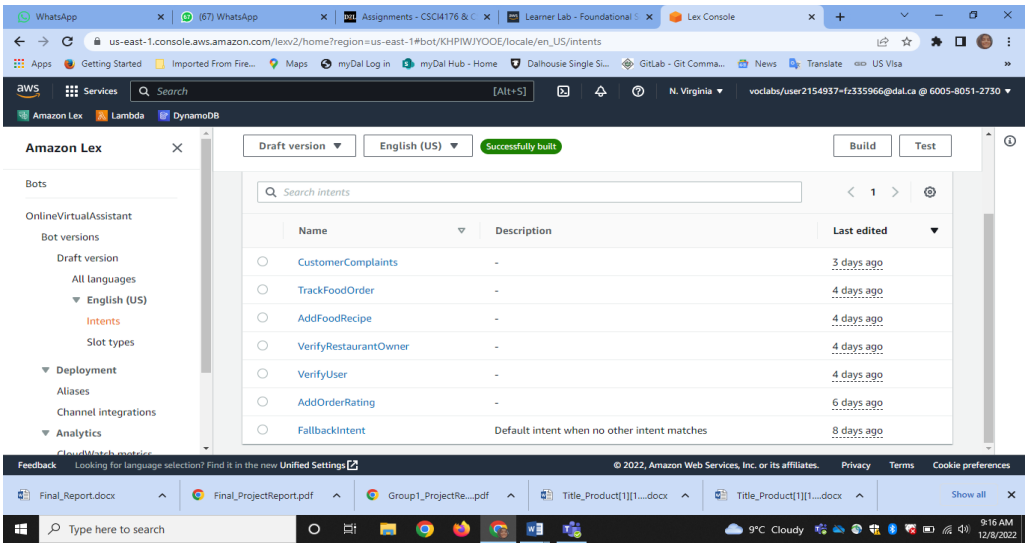

Figure 30: Intents for bot "Online Virtual Assistant"

Step-2: I created tables in the AWS DynamoDB to store the user data, food order rating, recipe, restaurant owner data etc as shown in the fig 31

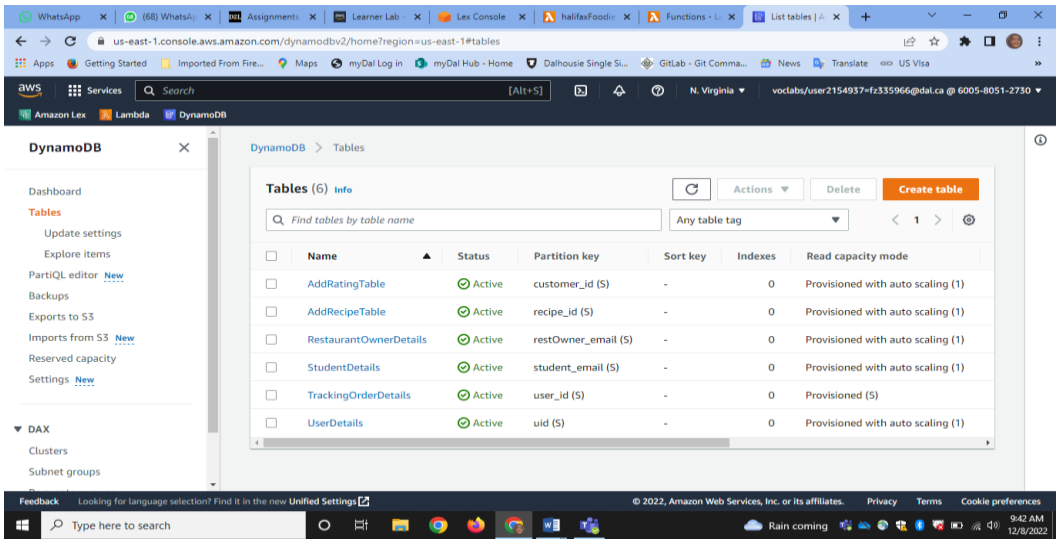

Figure 31: Tables in DynamoDB

Step-3:

- I created lambda function **"Halifax foodie"** which has multiple functionalities like verifying customer, verifying restaurant owner, tracking food order, adding food rating, adding food recipes along with its cost and type as shown in the fig 32.
- For tracking I first validate the user and then user can track the order as shown in the fig 33-39

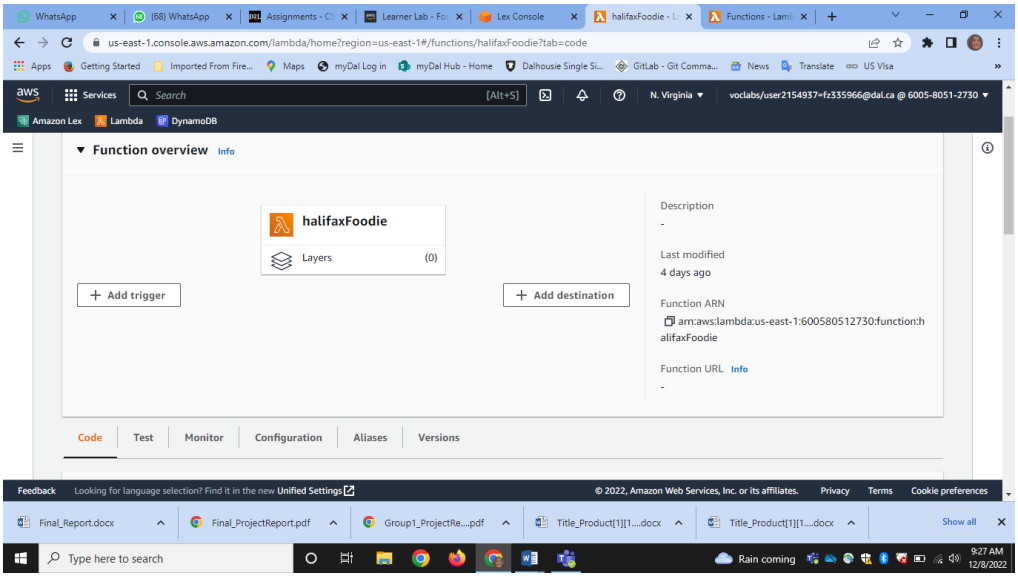

Figure 32: Lambda function - "HalifaxFoodie"

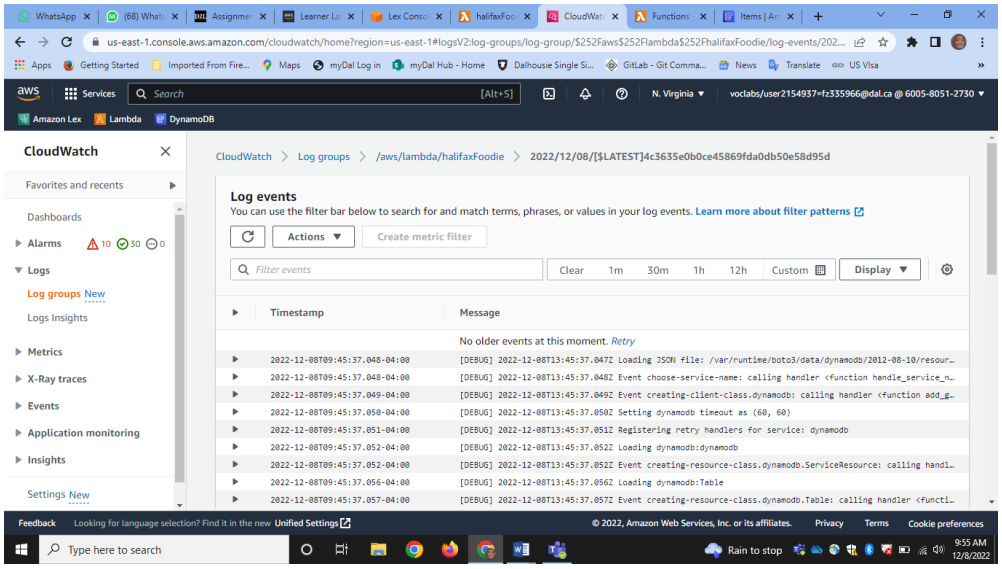

Figure 33: Lamda Function is triggered to verify user

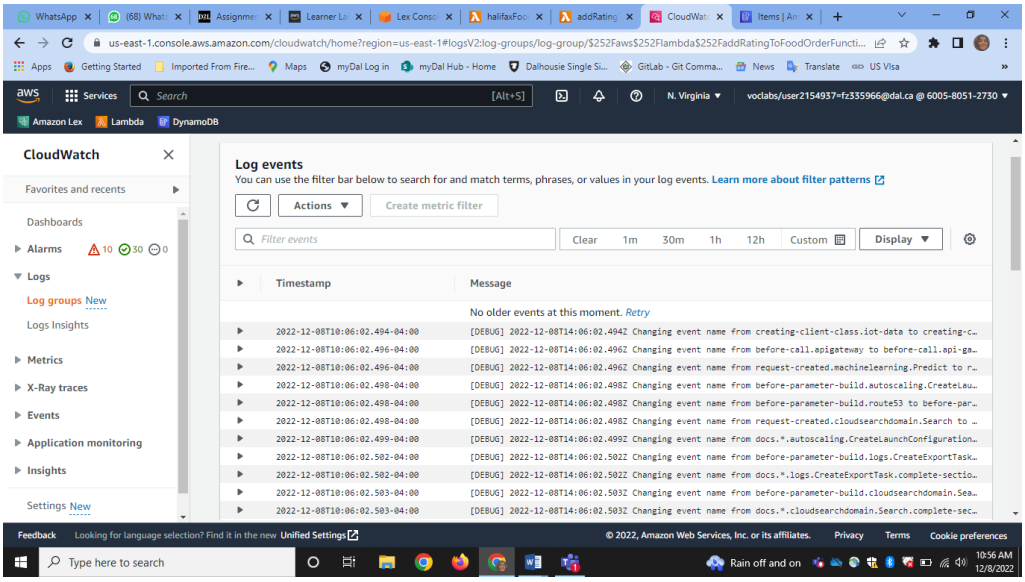

Figure 34: Lambda function is triggered to add rating

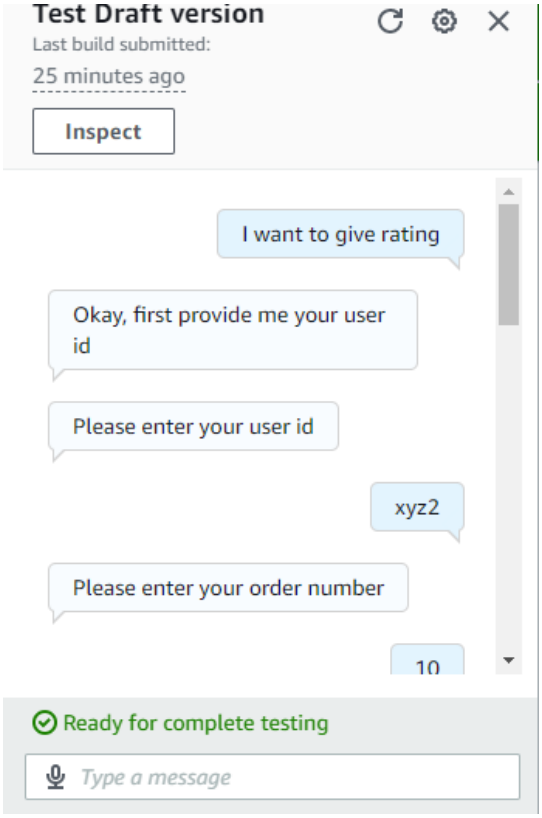

Figure 35: Test case to verify user and add rating

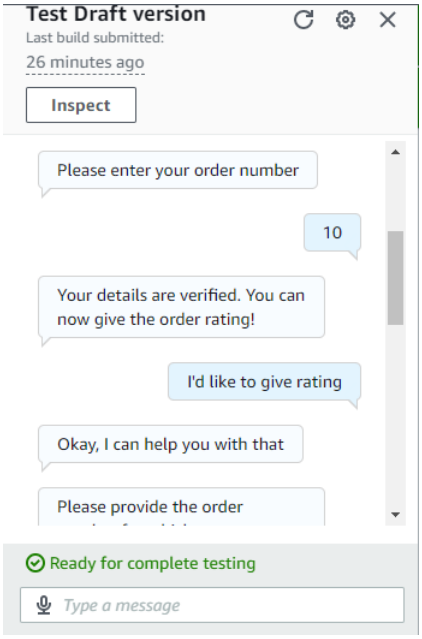

Figure 36: Test case to verify user and add rating

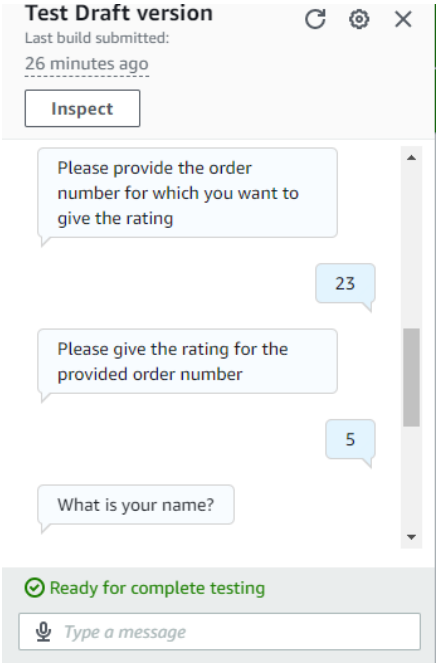

Figure 37: Test case to verify user and add rating

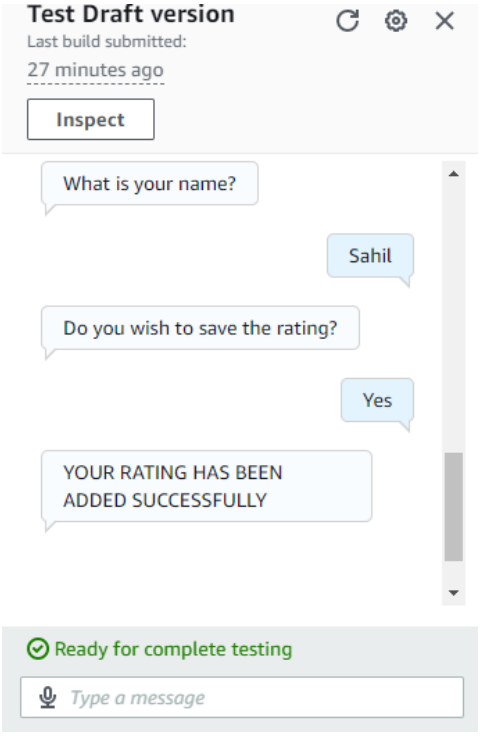

Figure 38: Test case to verify user and add rating

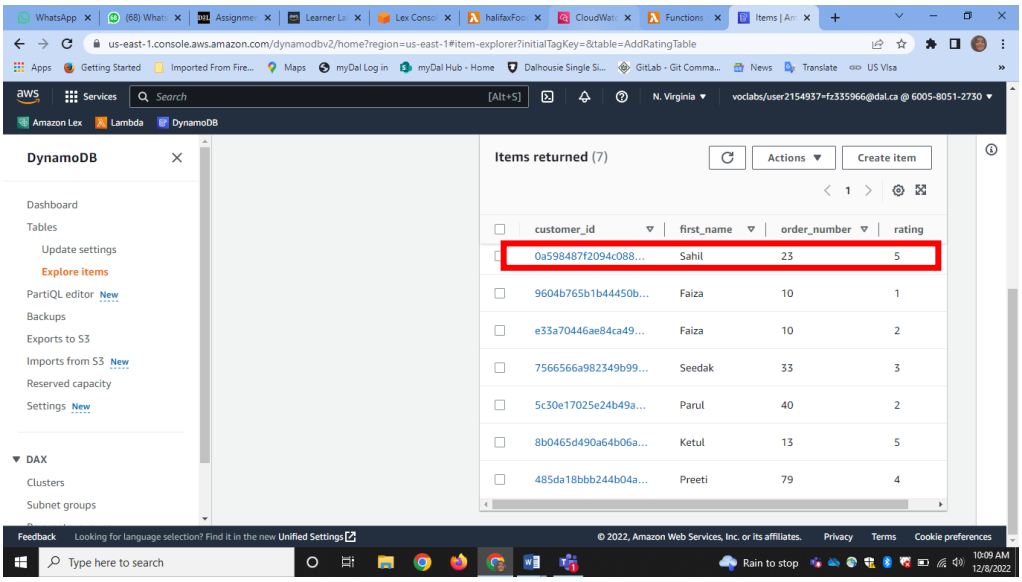

Figure 39: Rating added in the DynamoDB successfully

Test case 2: Customer wants to track order

Step-1: Verify the user before user tracks the order as shown in the fig 42 and 43.

Step-2: After verifying user tracks the order as shown in the fig 44.

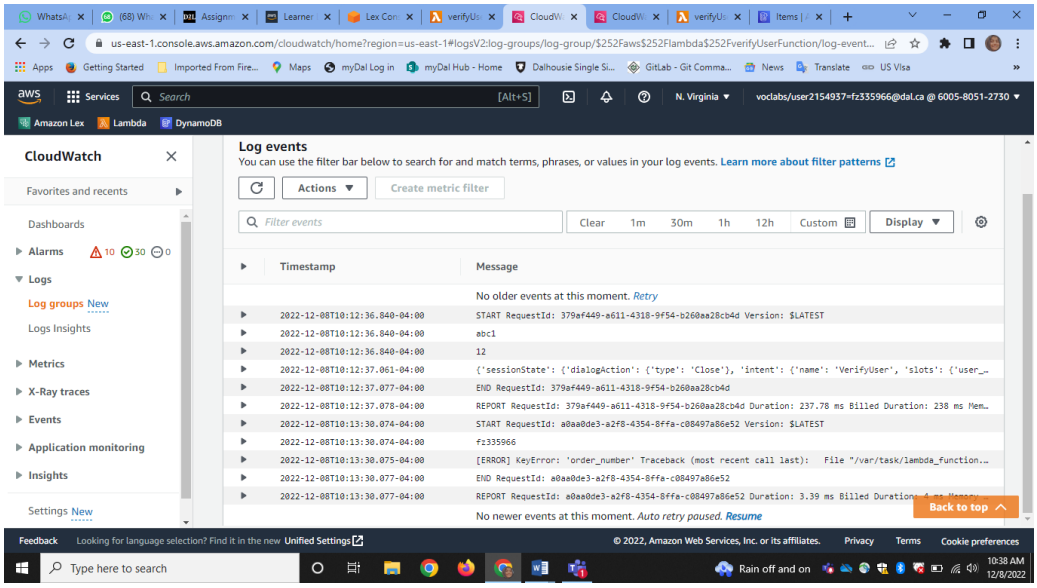

Figure 40: Lambda is triggered to verify user

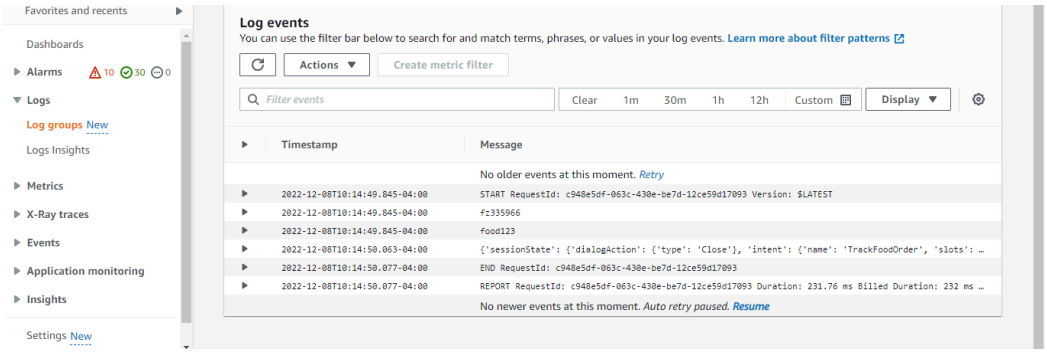

Figure 41: Lambda is triggered to track order

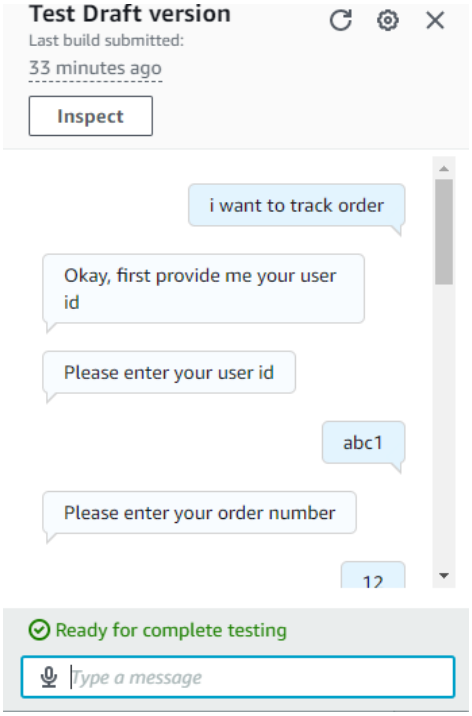

Figure 42: Test case to verify user and track order

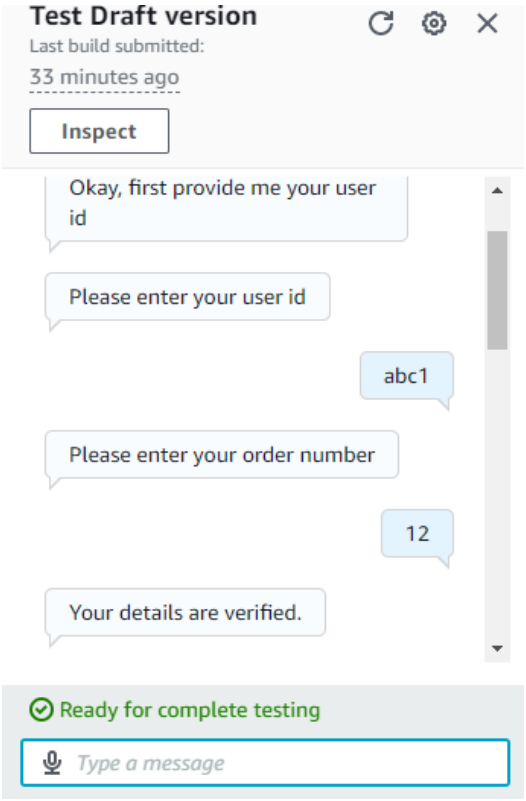

Figure 43: Test case to verify user and track order

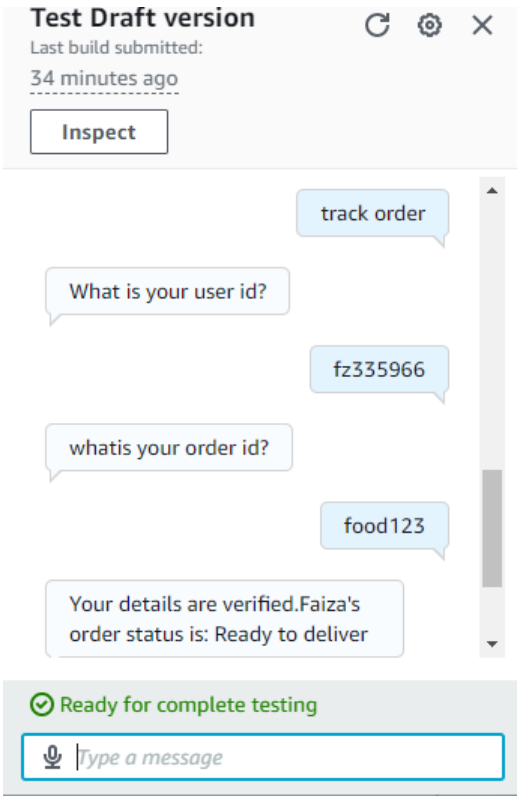

Figure 44: Test case to verify user and track order

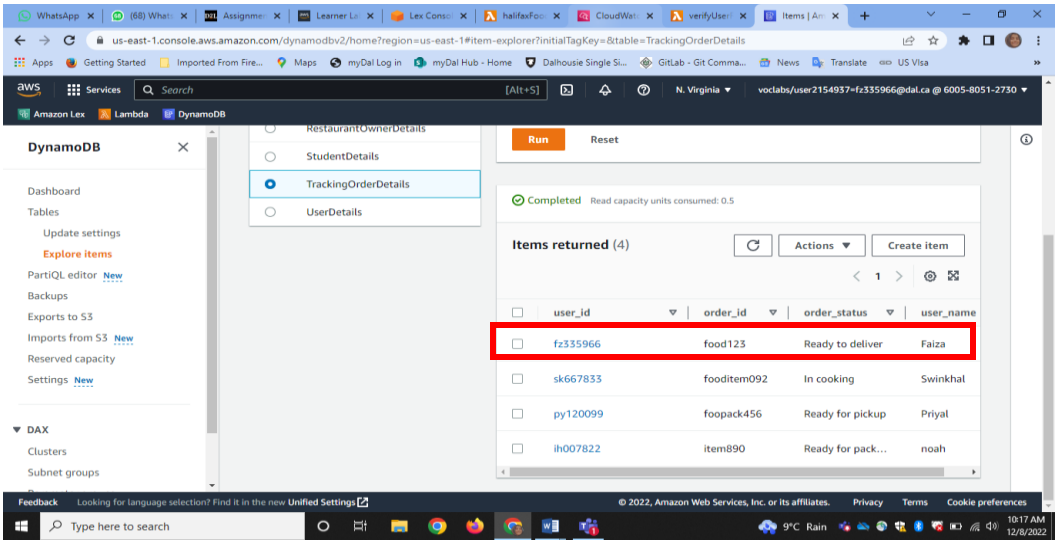

Figure 45: Table verifies the order status of the user

Test case – 3: Restaurant owner adding food recipe.

Step-1: Verify the restaurant owner before restaurant owner adds the food recipe and cost as shown in the fig 48 and 49.

Step-2: After the restaurant owner is verified, restaurant owner can add food recipes and cost as shown in the fig 50-52.

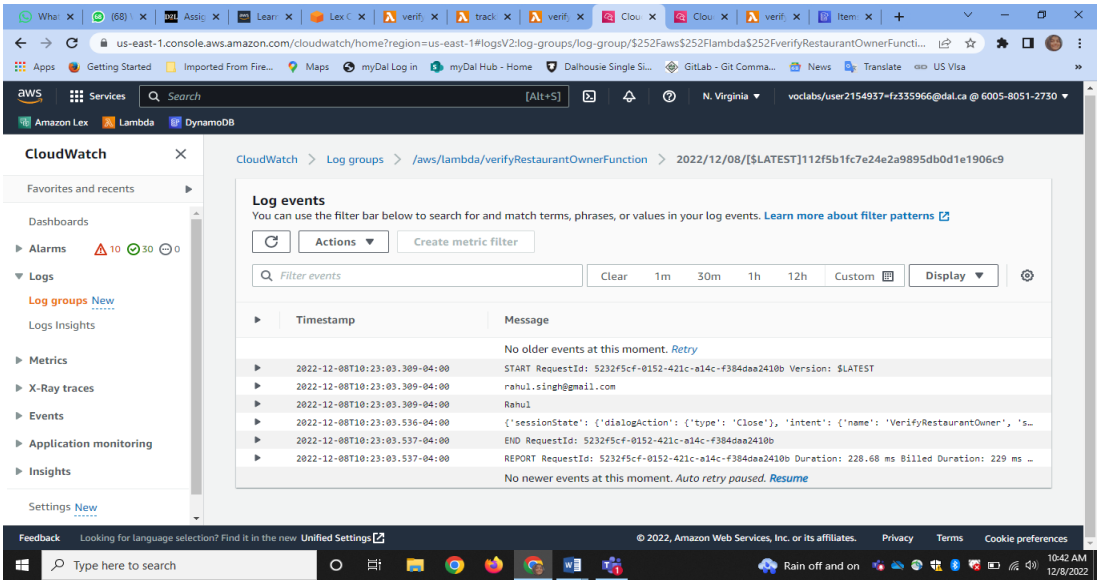

Figure 46: Lambda is triggered when restaurant owner is verified

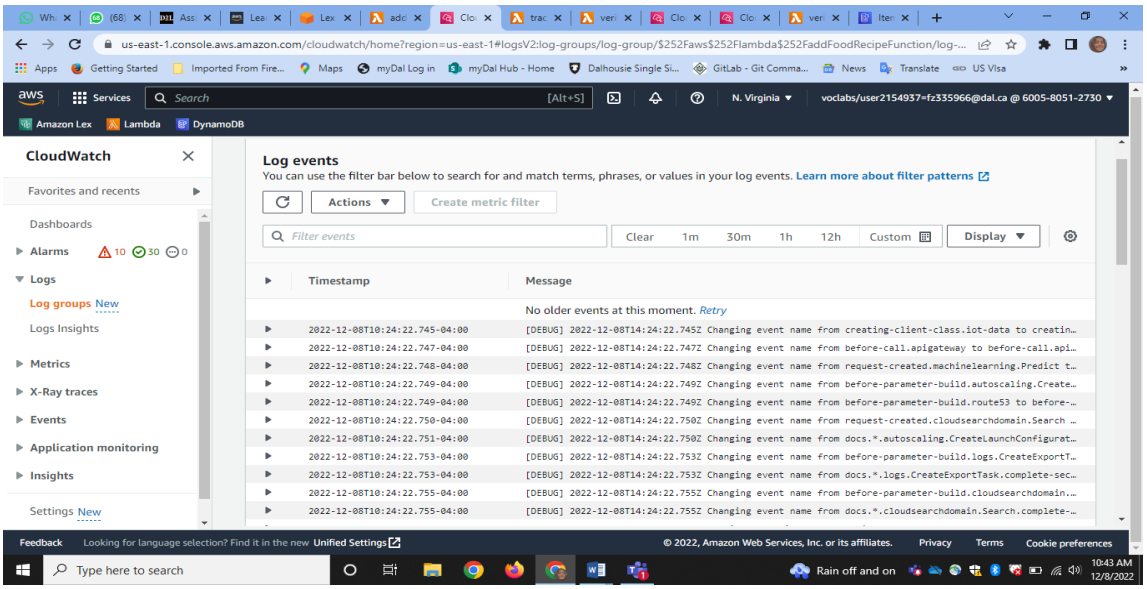

Figure 47: Lambda is triggered to add the food recipe and its cost

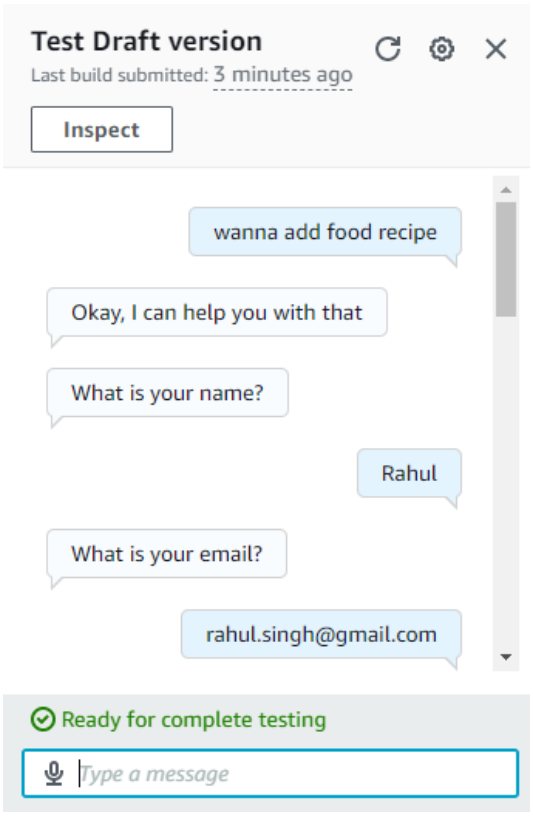

Figure 48: Test case to verify restaurant owner and to add food recipe with its cost

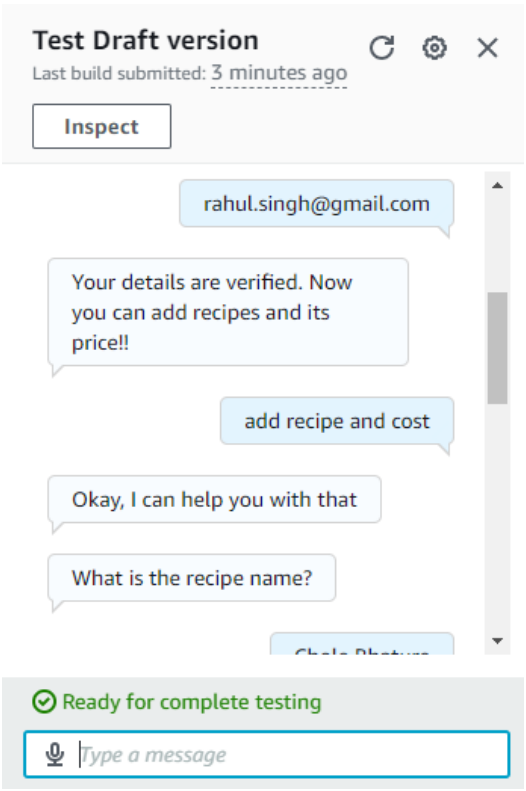

Figure 49: Test case to verify restaurant owner and to add food recipe with its cost

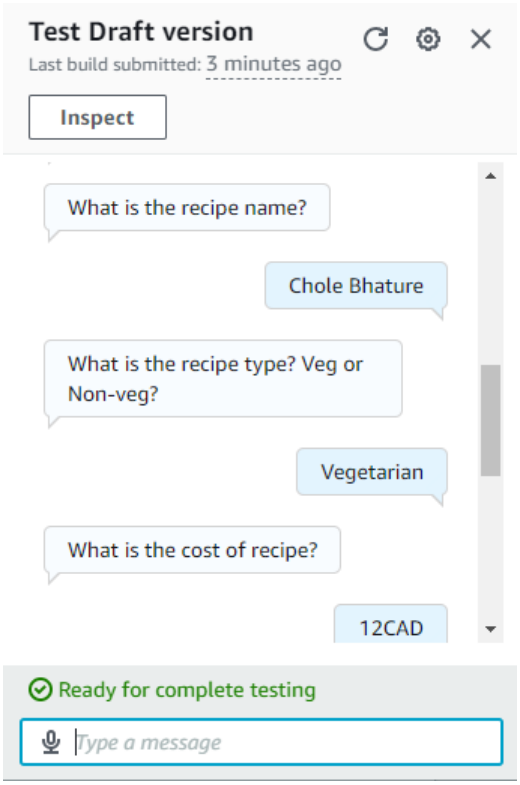

Figure 50: Test case to verify restaurant owner and to add food recipe with its cost

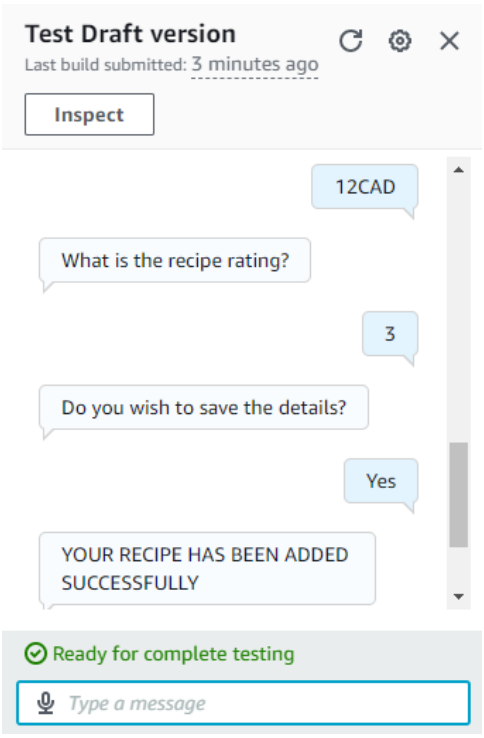

Figure 51: Test case to verify restaurant owner and to add food recipe with its cost

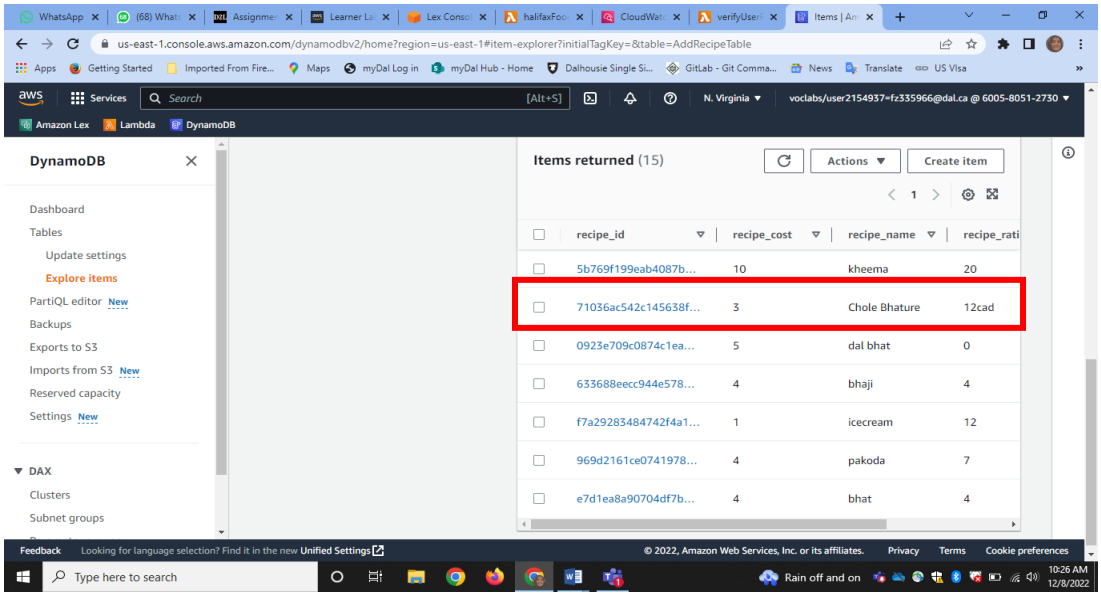

Figure 52: Food recipe and cost is added in the DynamoDB successfully

<u>Test case 4: User add complaints.</u>

- Customer adds name of the restaurant to which complaint is made as shown in the fig 54. Fig 56 shows the details of that restaurant.
- Chatroom id is generated to chat with the restaurant admin/customer service for complaints as shown in the fig 55.

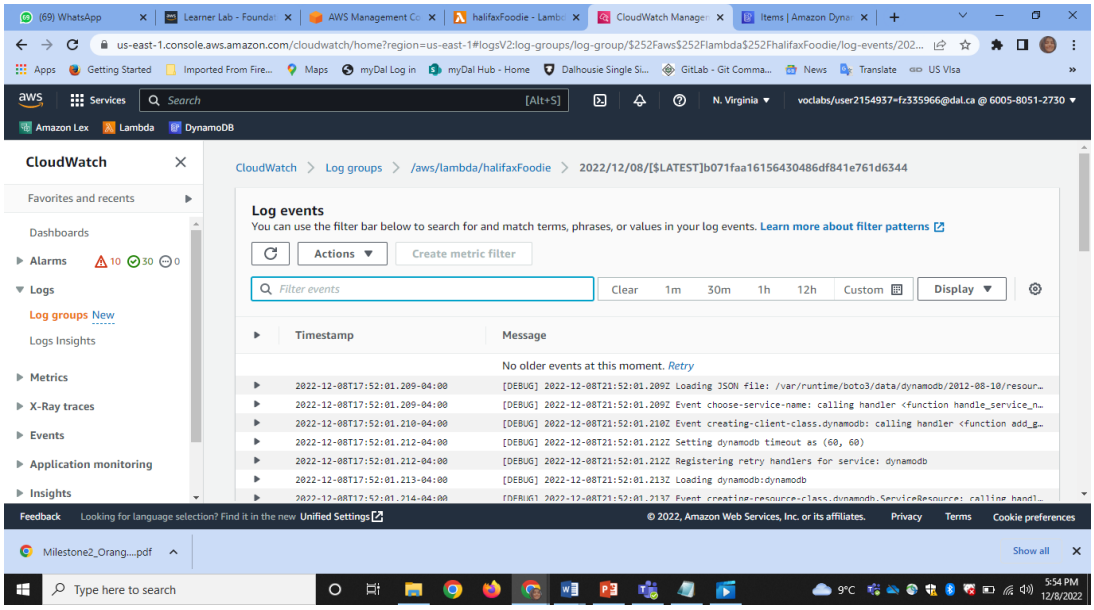

Figure 53: Lambda is triggered to add customer complaint

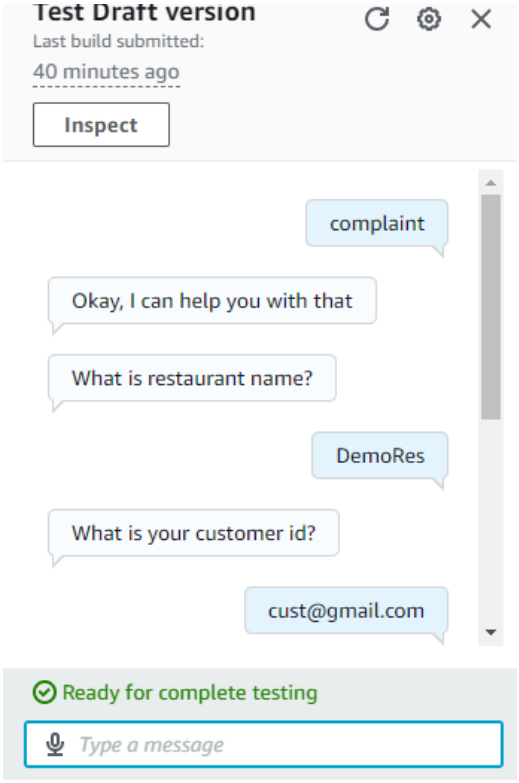

Figure 54: Test case to add customer complaint

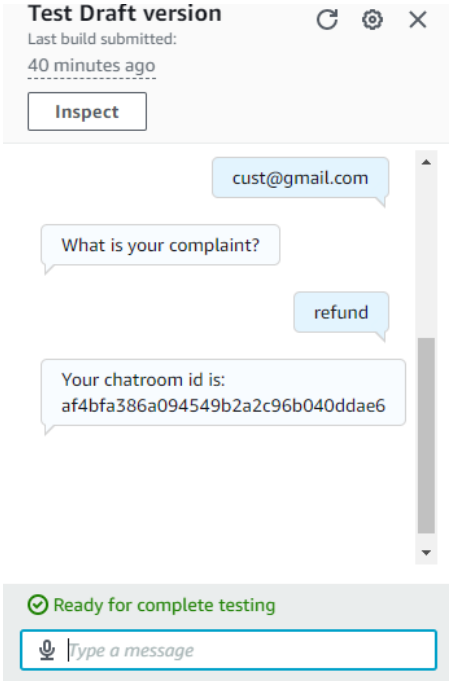

Figure 55: Test case to add customer complaint

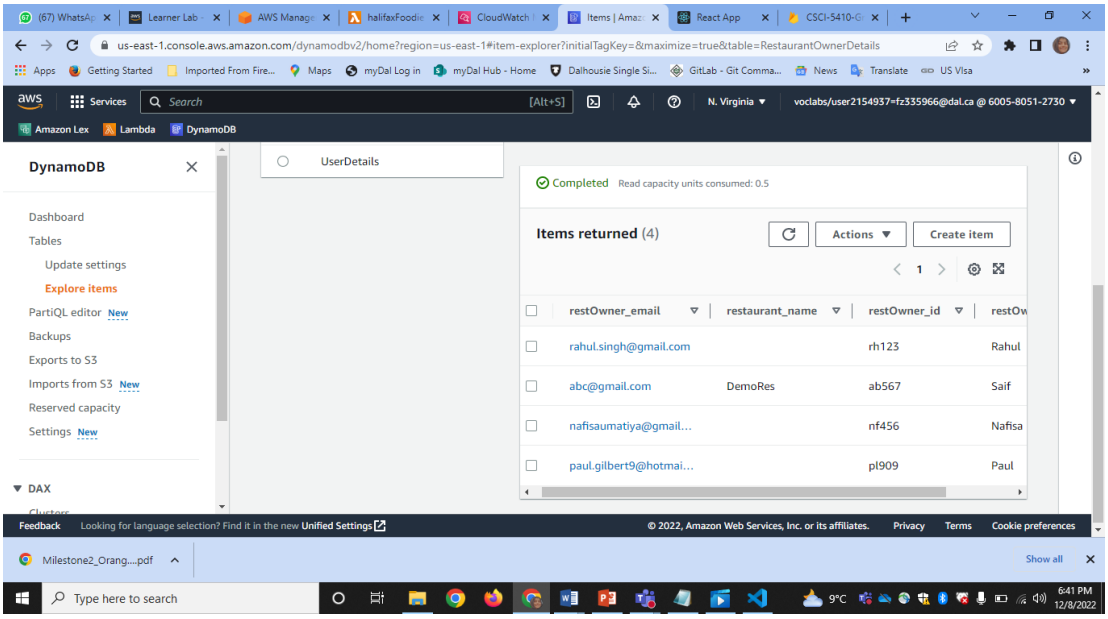

Figure 56: Customer complaint for the highlighted restaurant

1.3 Chat Module

With the help of this module, customers and the restaurant owner can interact in a live chat room. Customers use this feature to communicate issues with meal delivery to restaurant owners or customer service representatives. For instance, if customers receive their orders late, they can use a chatroom to speak with the restaurant manager who oversees that. There will be several logged-in sessions for this module, including client and restaurant operator sessions. A minimum of two logged-in accounts is needed to create a chat room; one must be a customer and the other must be a restaurant customer care professional. To make the dynamic chat chatrooms we would be using react applications coupled with "**Google Firebase"** at the backend [12].

Implementation Details:

- The logged-in customers will access the system in the first stage. Initially, they will select "Chat with us" from the menu if they have any problems with the current order. By pressing this button, the AWS Lambda function will start the chat room on the customer's end. The key-value pair containing all the information required for the setup of the chat room and the conversation will be kept in "**Google's Firebase"** [12] database.
- Another "**AWS Lambda"** function will be executed after receiving all the client-provided information, allowing the restaurant's customer service person to enter the chat session [11]. This module can only be used by authorised users. The necessary information will be retrieved via lambda functions from the order information pertaining to the customer and restaurant. Text messages will be used by them to communicate with one another.

Test Cases:

Table 4: Test case scenarios for Chat Module

Test Scenarios	Expected behavior
Customer/user tries to initiate a chat. (Customer and Customer service representative is online).	A chat should be initiated with the required restaurant's customer service representative.
Customer/user tries to end the chat.	Chat should be terminated from both ends (customer and restaurant's customer service representative).

Test-case screenshots:

- The chatroom is initiated between the customer and the restaurant customer service/ restaurant admin for the complaints as shown in the fig 57 and 58.

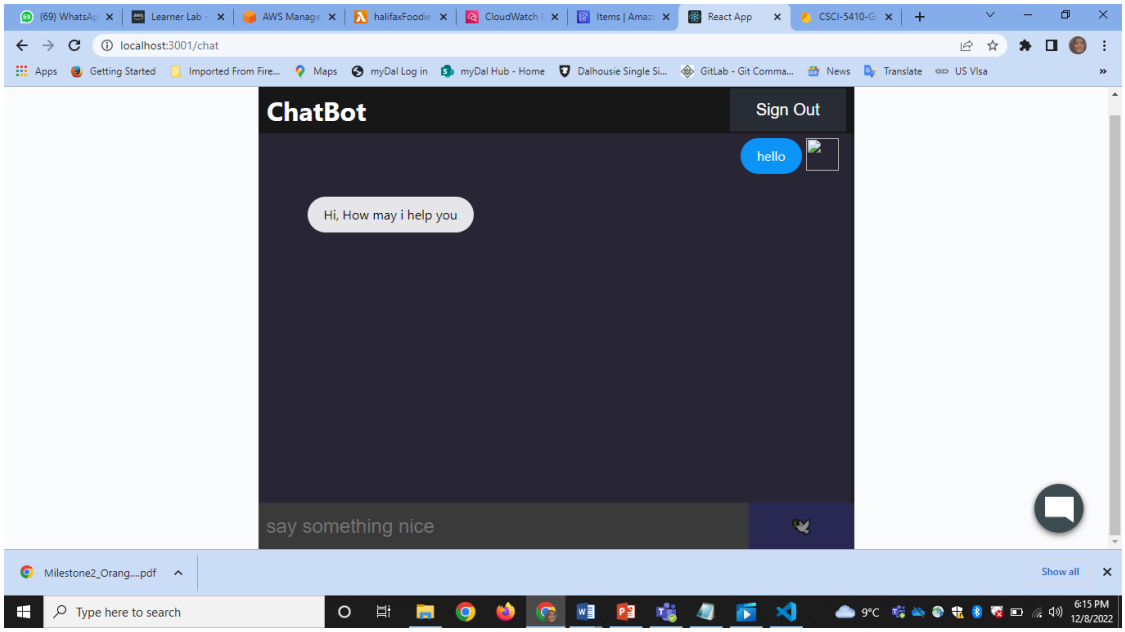

Figure 57: ChatRoom is generated for customer complaints

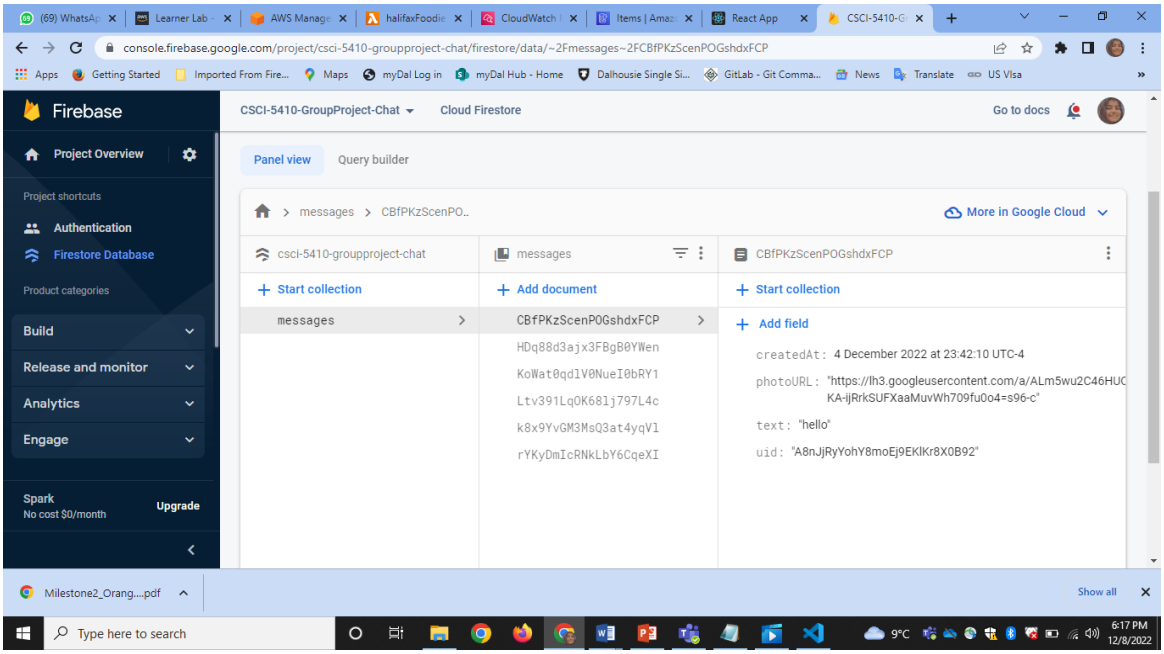

Figure 58: Firebase having restaurant messages data

1.4 Data Processing Module

In this feature, an "**AWS lambda function"** would be triggering the services required to extract and process the data to be used for quick retrieval [11]. This service will be limited to the restaurant owners wherein they would be allowed to extract "key" ingredients from the recipe. restaurant owners using lambda function, would upload recipe through front-end, which would be saved in the **"S3"** bucket [6]. Then, Amazon "**Comprehend**" will analyze the data from S3 to extract the title or the key ingredients from the recipe (**named entities**) [8], and store them in "**AWS DynamoDB"** along with name of the recipe for easy searching [3]. Machine learning module could use this data for further analyses. The uppercase entities would be considered as "named" entities.

Test Cases:

Table 5: Test case scenarios for Data Processing Module

Test Scenarios	Expected behavior
Named entities correctly identified	Shows the requested recipe and ingredients to restaurant owners.
Named entities exists but not identified	Named entity should have been identified and displayed.
Named entities falsely identified	Entities those were not part of named entities criteria were identified.
No named entity exists and not identified	No ingredients or recipe will be shown to restaurant owner.

Screenshots of test cases:

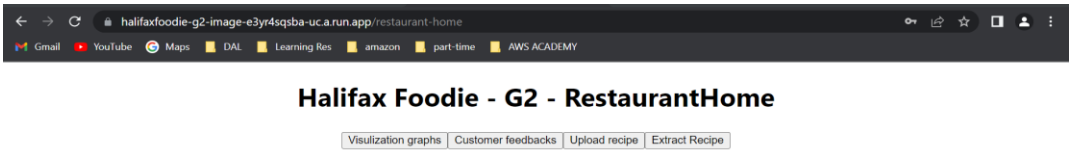

Figure 59: restaurant homepage providing upload and extract button

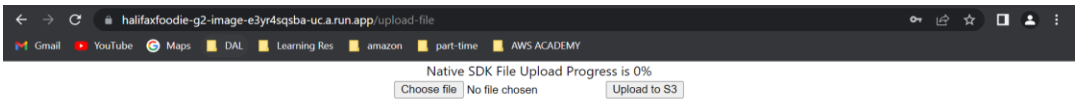

Figure 60: when the user clicks on upload page, the application redirects to this page

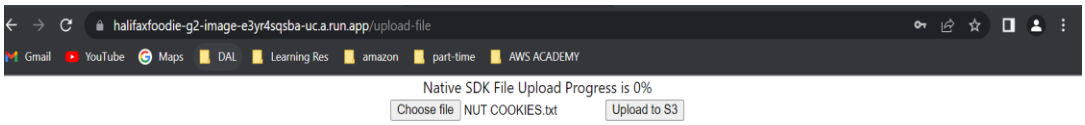

Figure 61: upload page with a file just added through choose files

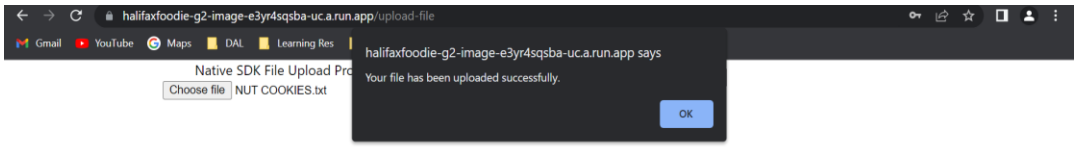

Figure 62: User receiving a prompt after they click on upload

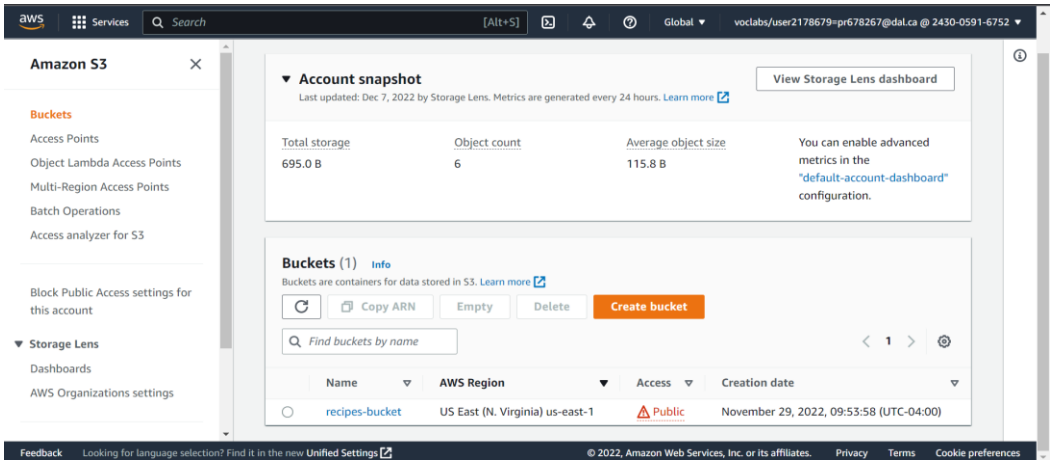

Figure 63: S3 bucket

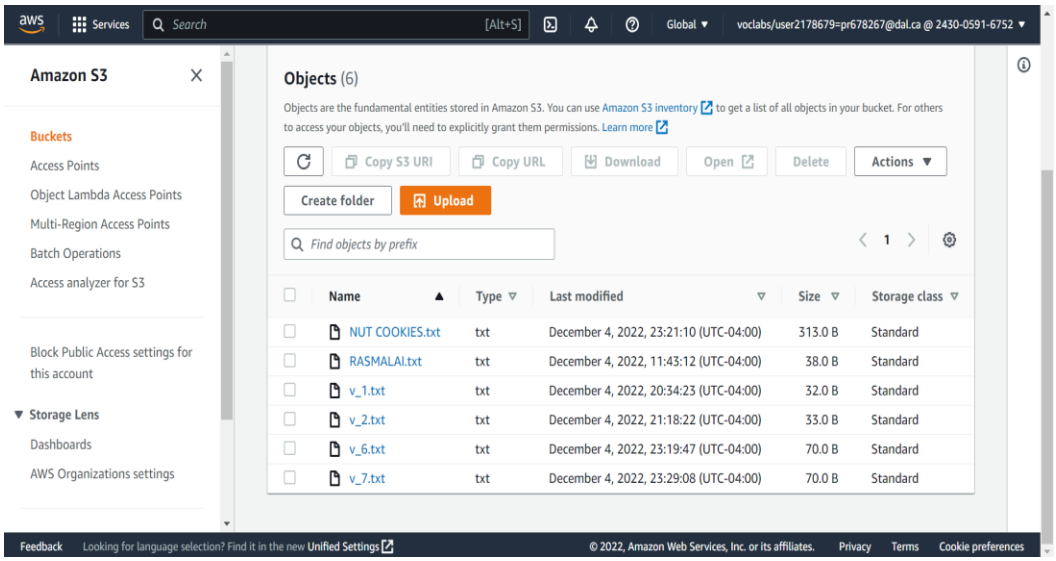

Figure 64: Nut Cookies txt uploaded successfully in S3 Bucket

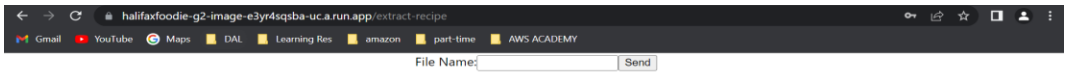

Figure 65: Extract recipe page

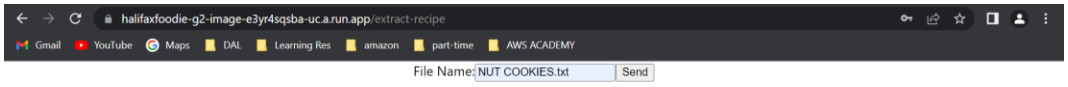

Figure 66: Added the name of file uploaded earlier in text box for extraction

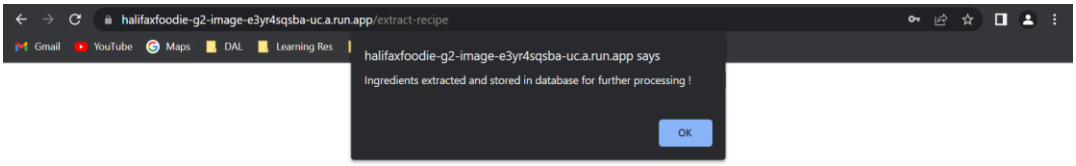

Figure 67: NUT COOKIES.txt being passed on to be extracted

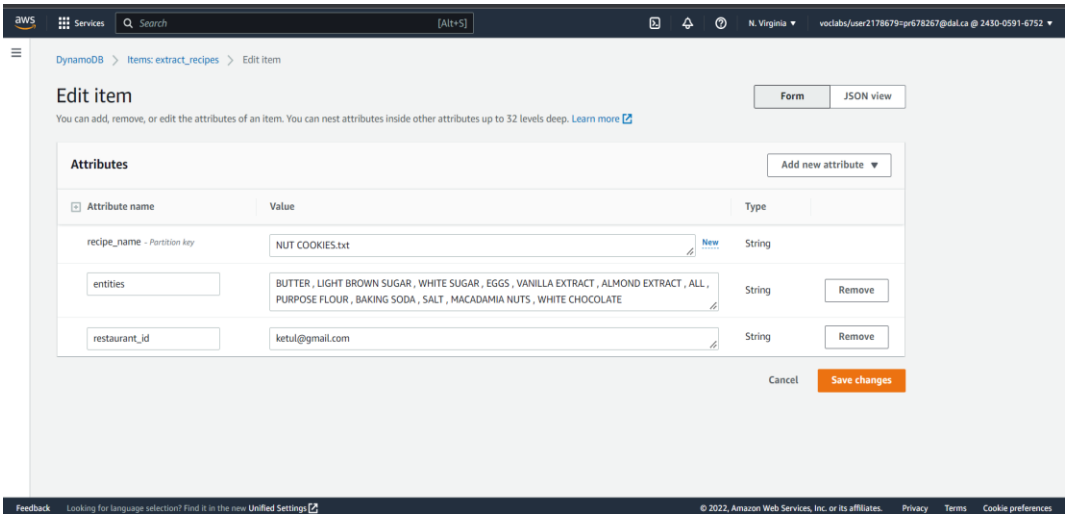

Figure 68: contents from the file in previous steps, saved in DYNAMODB after analysis

1.5 Machine Learning Module

1.5.1 Similarity Score

In this feature, similarity between two recipes uploaded in "**AWS S3"** bucket will be identified [6]. "**GCP AutoML"** will be used to calculate similarity of recipes based on different measurements such as Euclidean distance [7]. The same recipes will be classified under the same group based on the threshold value. The value for threshold will be decided after checking model performance, available data, and features.

Test Cases:

Table 6: Test case scenarios for Machine Learning Module (Similarity Score)

Test Scenarios	Expected behavior
No similar recipe is found for recipe.	Maximum of 3 Random recipes of current Restaurant watched by user will be displayed.
One similar recipe is found.	Display only that recipe.
Two similar recipes found.	Display only those 2 similar recipes.
More than two similar recipes found.	Display only top 3 similar recipes.

Screenshots for test cases:

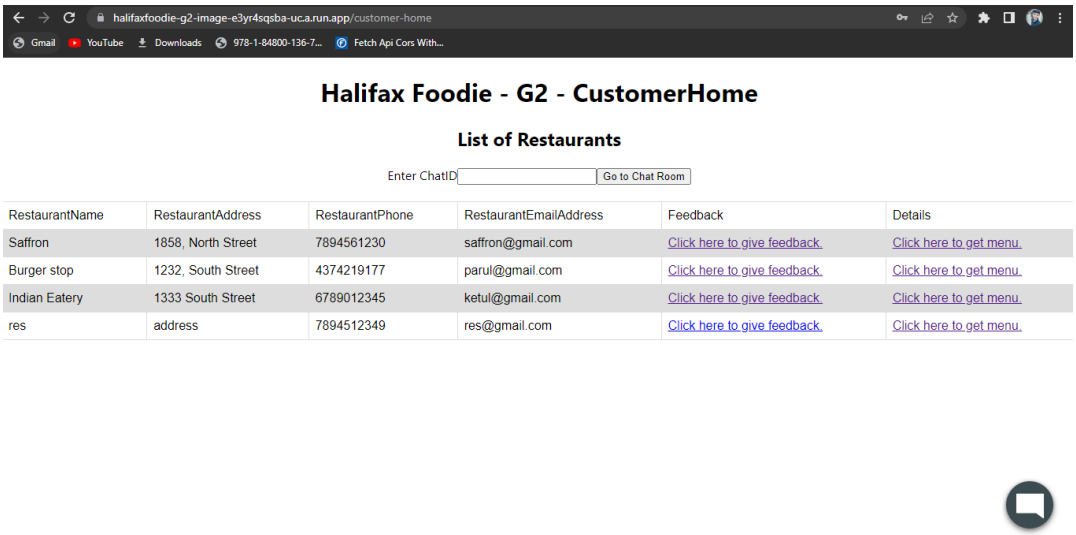

Figure 69: Customer Home page.

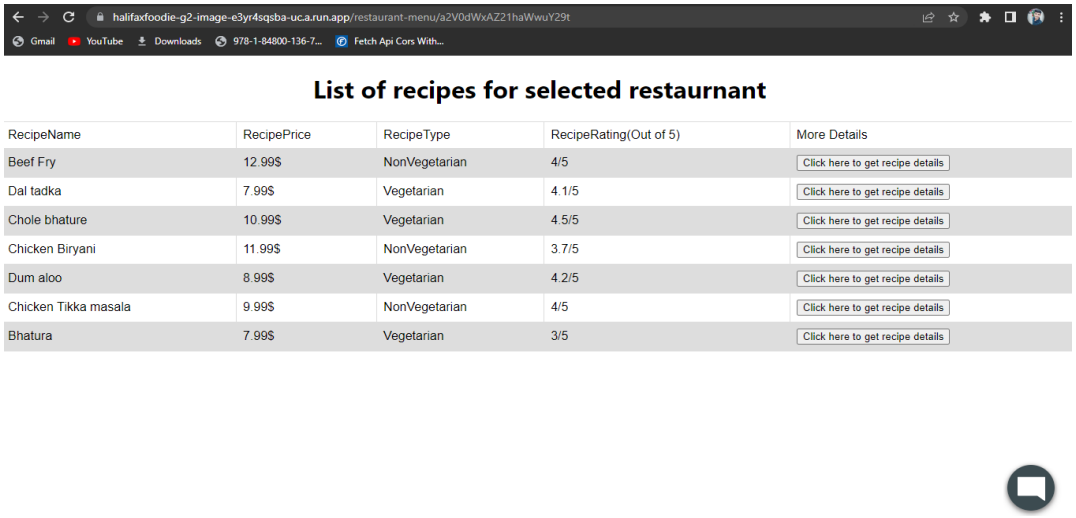

Figure 70: Selects one of the listed restaurants from home page.

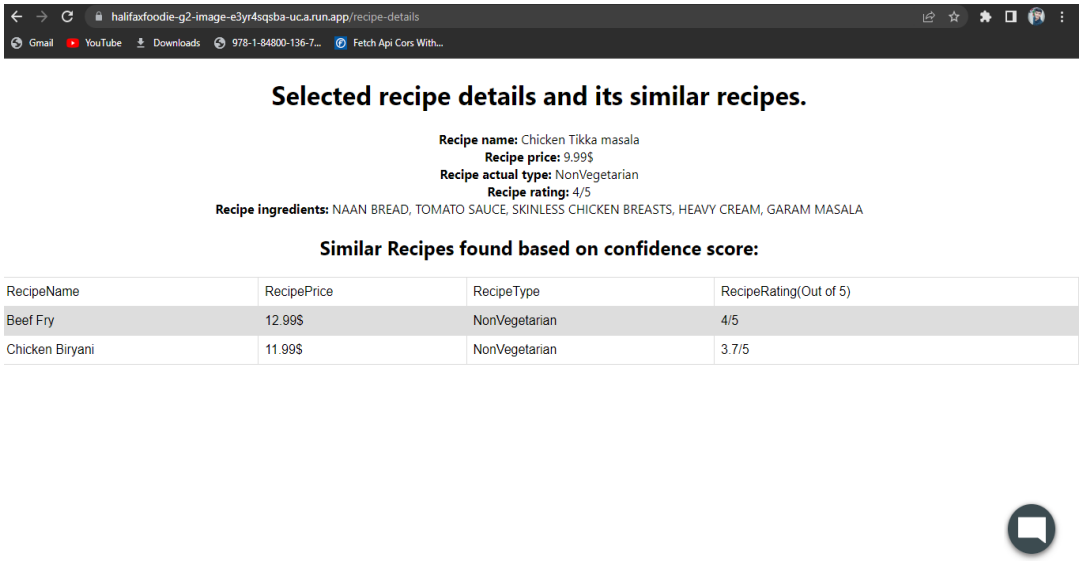

Figure 71: Selects Non-Vegetarian recipe from list recipes.

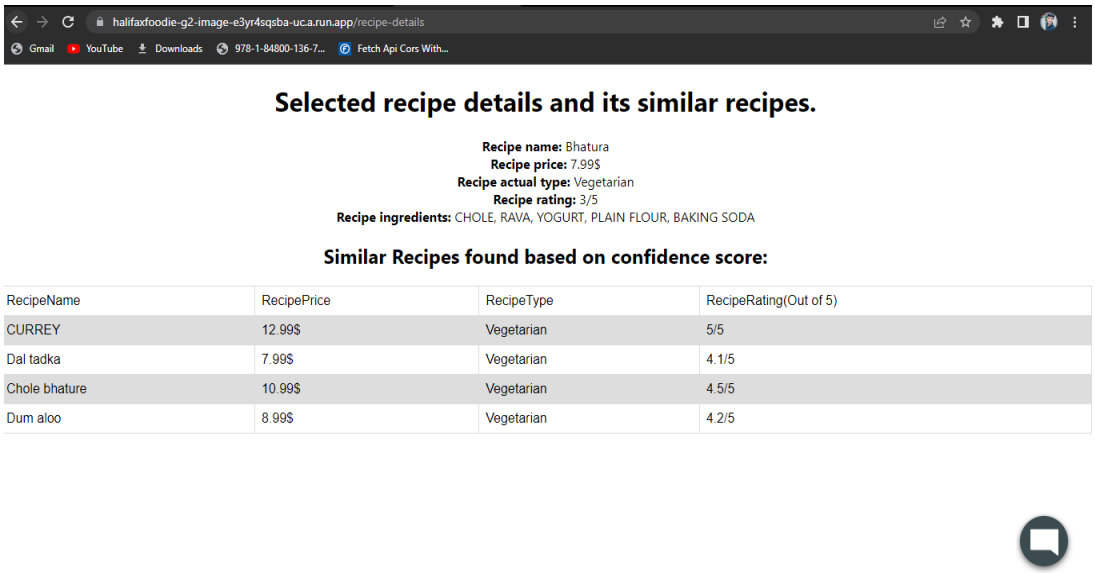

Figure 72: Selects Vegetarian recipe from list of recipes.

1.5.2 Polarity

The main aim of the feature is to find the polarity of the feedback provided by customers**. "AWS Comprehend"** will be used to calculate the polarity [8] and will be visualized using **AWS "QuickSight"** [9]. Restaurant owners will be able to check polarity by clicking on the option of view visualize customer feedback polarity.

Test Cases:

Table 7: Test case scenarios or Machine Learning Module (Polarity)

Test Scenarios	Expected behavior
No customer feedback available.	Alert user with proper message.
Customer feedback polarity is positive.	Display green color happy face icon.
Customer feedback polarity is negative.	Display red color sad face icon.
Customer feedback polarity is neutral.	Display yellow color neutral face icon.

Screenshots for test cases:

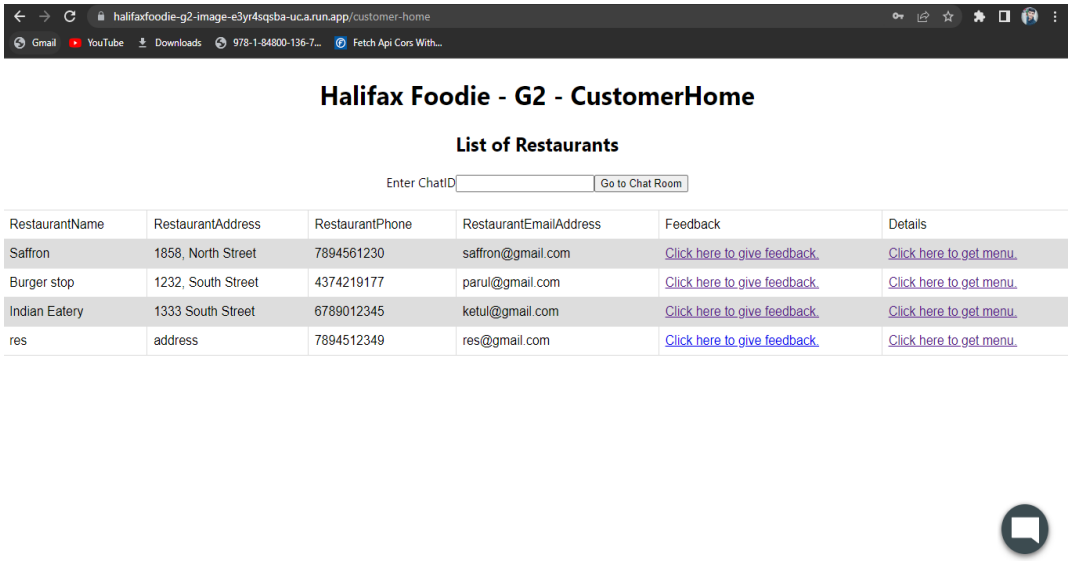

Figure 73: Customer Home page.

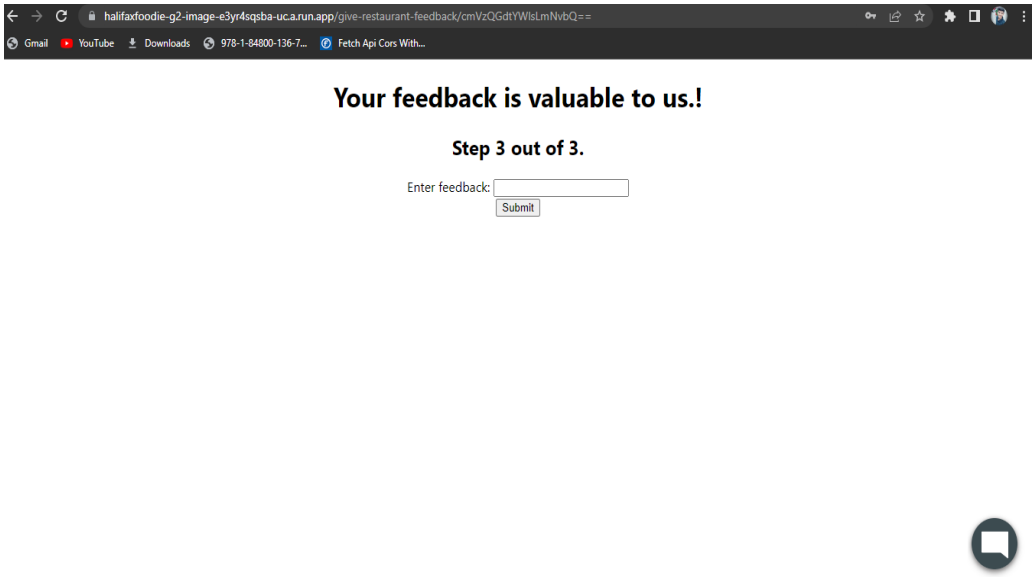

Figure 74: Selects one of the listed Restaurant from list.

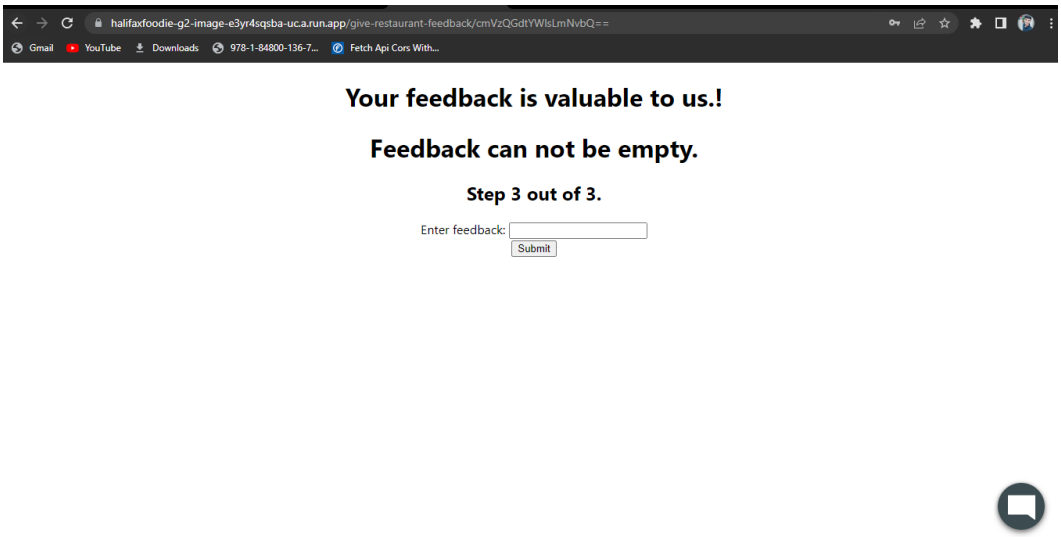

Figure 75: Provides empty feedback.

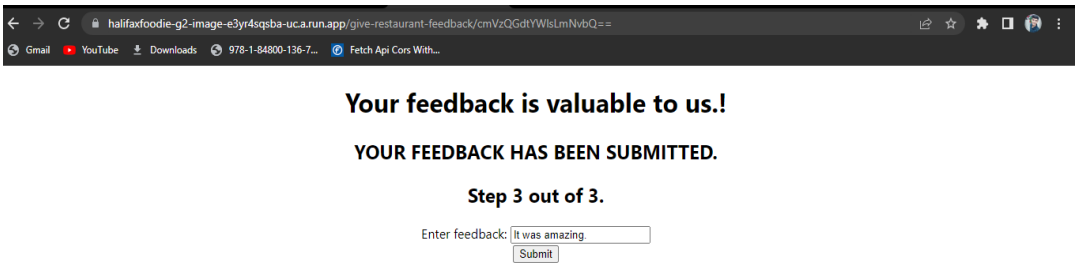

Figure 76: Provides any feedback.

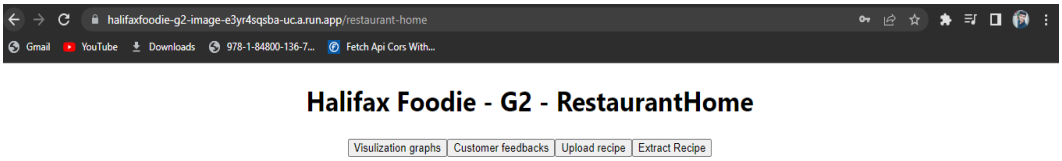

Figure 77: Restaurant Home page.

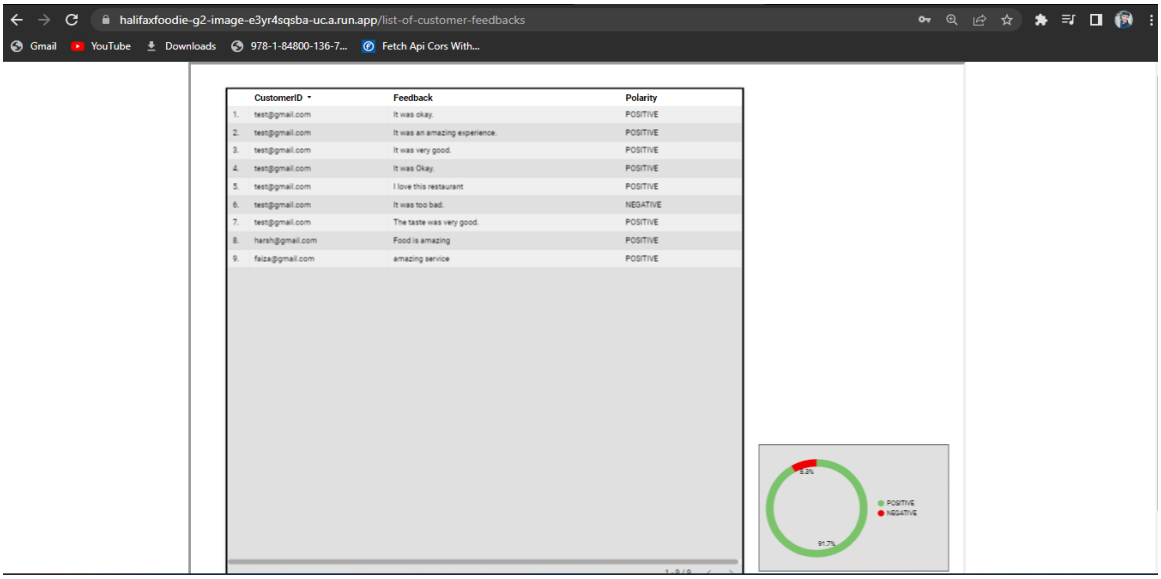

Figure 78: Selects Customer Feedbacks button.

1.6 Visualization Module

1.6.1 Login Statistics

The feature will provide graphs/charts of traffic of Restaurant and Users, and login activity of user. It will be visualized using "**GCP Data Studio"** [10]. The graphs/charts will be displayed on the dashboard page of Restaurant admin.

Test Cases:

Table 8: Test case scenarios for Visualization Module (Login Statistics)

Test Scenarios	Expected behavior
No data is available.	No graphs will be displayed with proper message.
Data is available.	Related charts/graphs will be displayed.

Screenshots of test cases:

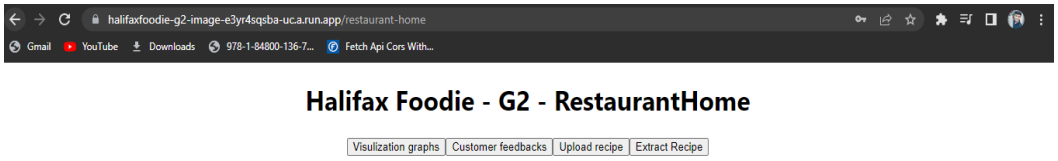

Figure 79: Restaurant Home page

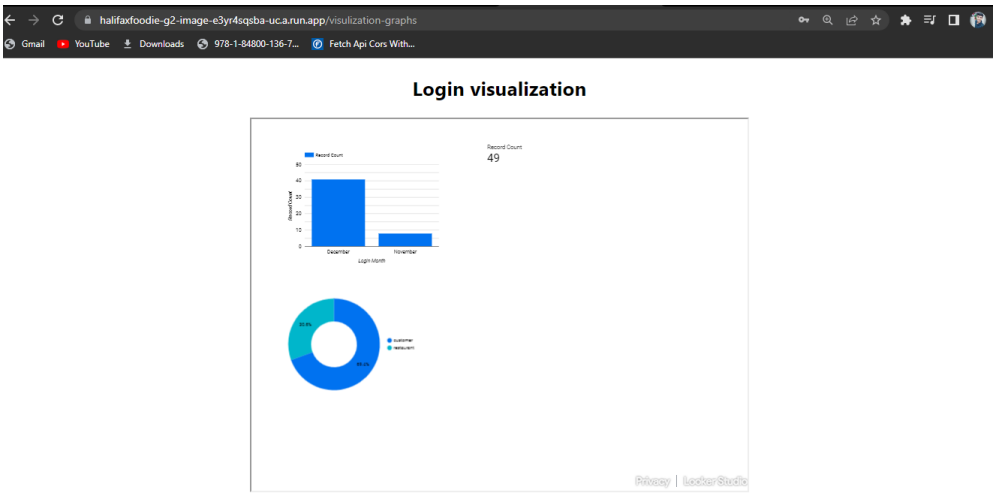

Figure 80: Login visualization.

1.6.2 Recipe Visualization

Recipe uploaded by will be highlighted using proper graphs. The examples of the graphs can be top rated recipes, most liked recipes, most profitable recipes, etc. "**GCP Data Studio"** will be used to show graphs on the dashboard page of Restaurant admin [10].

Test Cases:

Table 9: Test case scenarios for Visualization Module (Recipe)

Test Scenarios	Expected behavior
No data is available.	No graphs will be displayed with proper message.
Data is available.	Related charts/graphs will be displayed.

Screenshots of test cases:

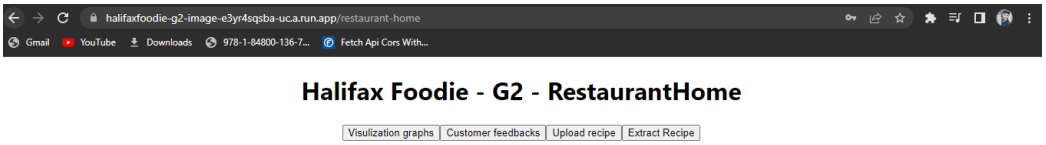

Figure 81: Restaurant Home page.

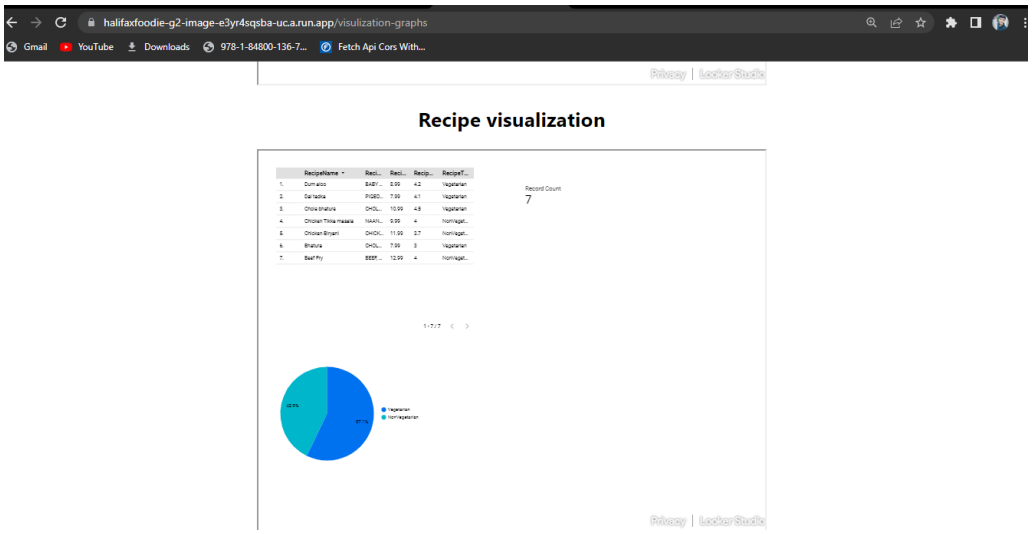

Figure 82: Recipe Visualization.

1.7 Message Passing Module

This module is “messaging-oriented middleware” that could be used as a queue to exchange data and enable communication “asynchronously and “reliably” between different micro-services [2]. Different applications could be publisher or subscriber or both to create or respond to an event to send or receive data. In this step, the publisher applications would publish data to a topic (“predefined” or “dynamic” based on the context of message passing) , and subscriber would subscribe to the topics and ingest data whenever needed [2].We will be using “**GCP pub/sub”** in our project to achieve the messaging passing and queuing [2].

Test Cases:

Table 10: Test case scenarios for Message Passing Module

Test Scenarios	Expected behavior
Publisher published the data to the topic and subscriber accessed the data when needed from the topic.	The publisher publishes the data and generates event to let the subscribers know the about update in the subscribed topic. The data is read correctly by the subscriber.
Publisher published the data but the subscriber could not access data when needed from topic.	The publisher publishes the data and generates event to let the subscribers know the about update in the subscribed topic. The data could not be read by the subscriber when it tried to access it.
Publisher could not publish the data	Publish could not publish data and subscriber not notified about the event.

Screenshots of test cases:

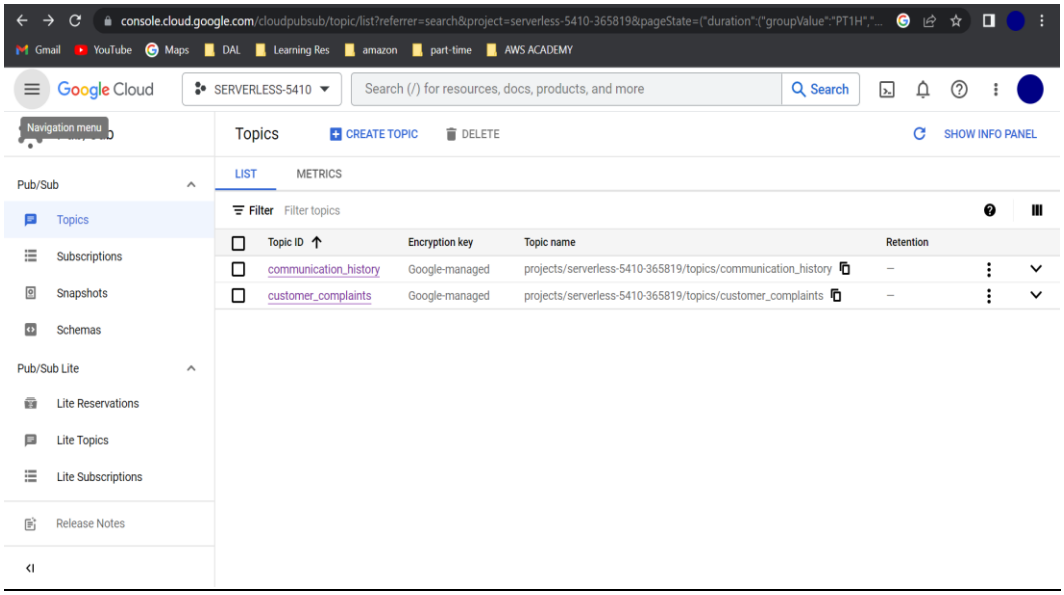

Figure 83: Topics

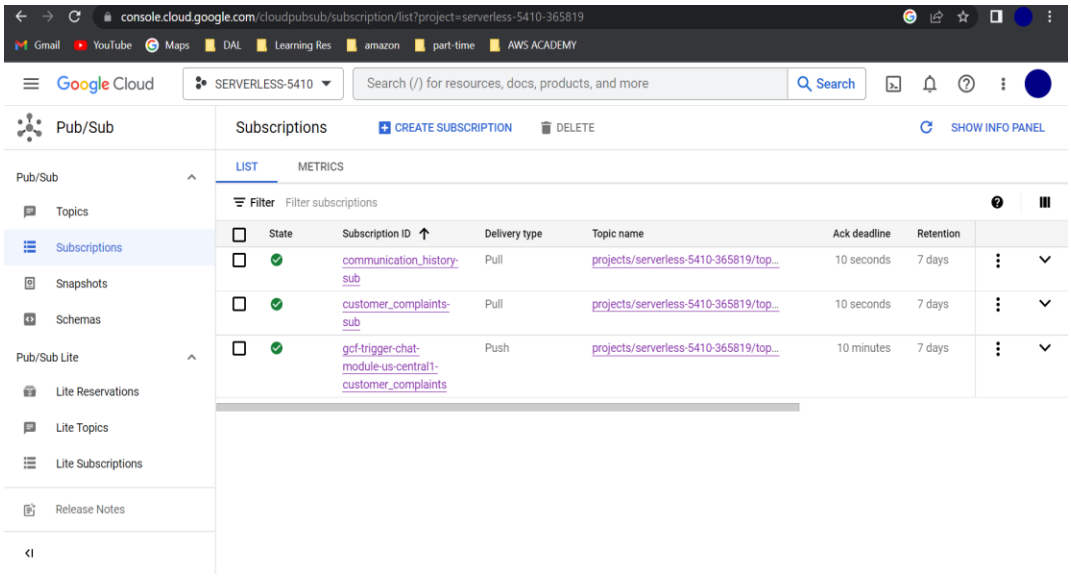

Figure 84: details of subscribers, I used the push type trigger to save basic details of chatroom in firestore.

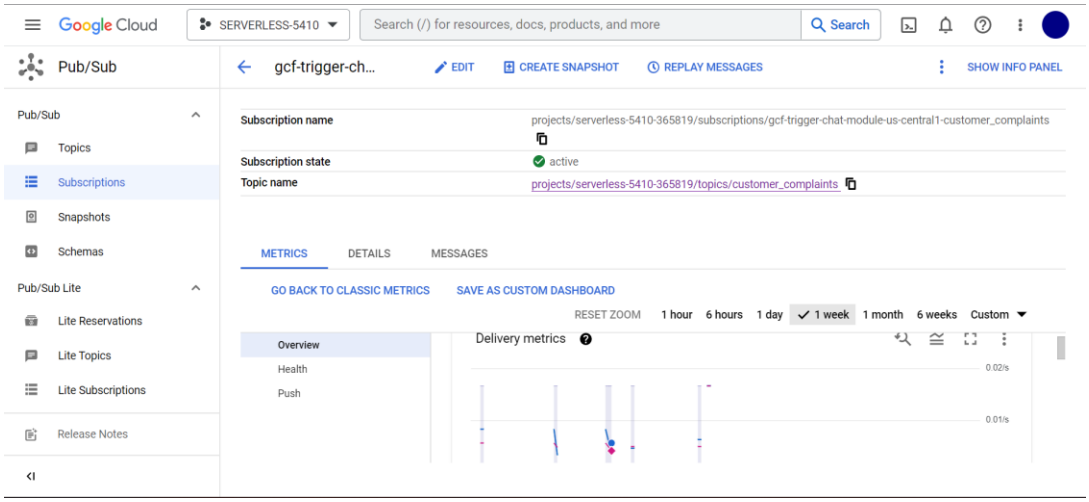

Figure 85: details of the subscriber we used to store the Chatroom data in firestore

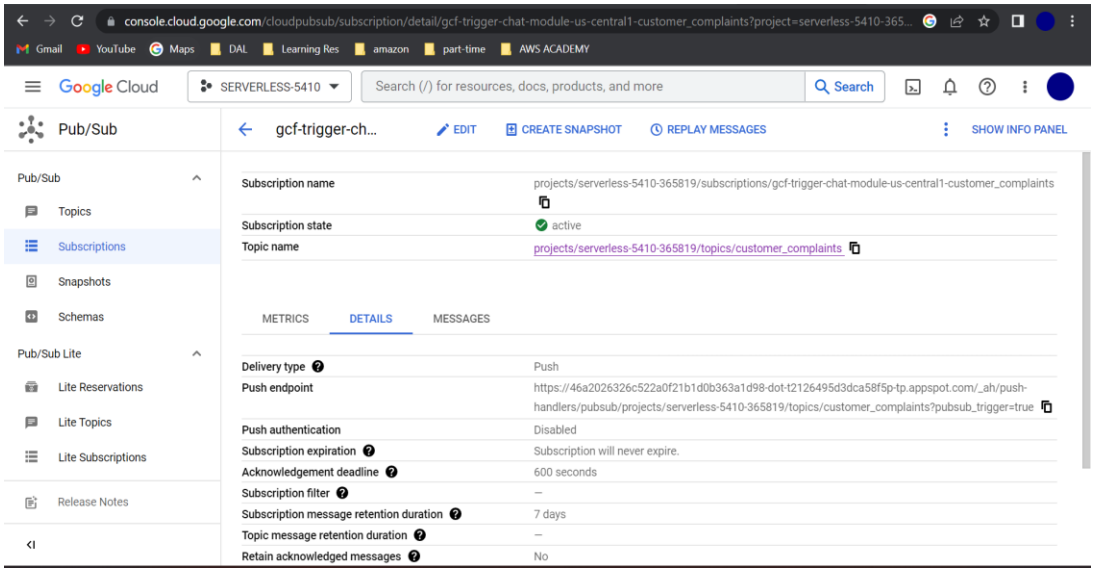

Figure 86: More details of the push subscriber that stores data in firestore

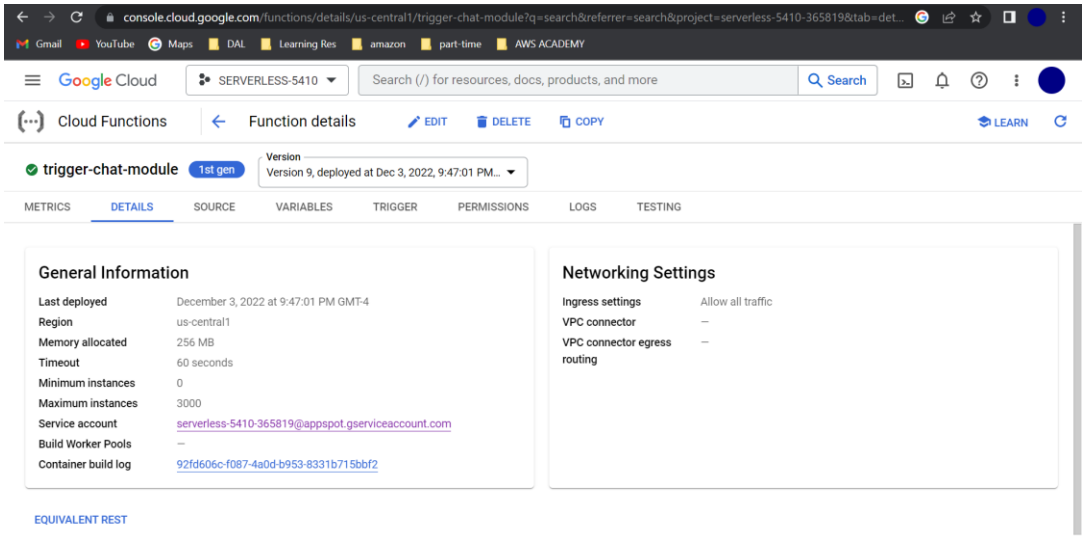

Figure 87: More detail on the cloud function that is a subscriber of push type to the topic

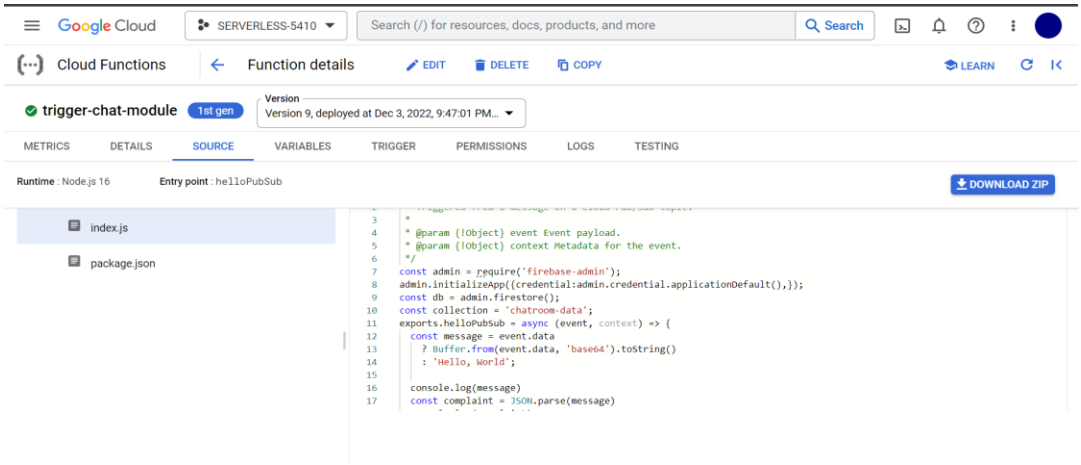

Figure 88: source snippet of a part of cloud function that is subscriber

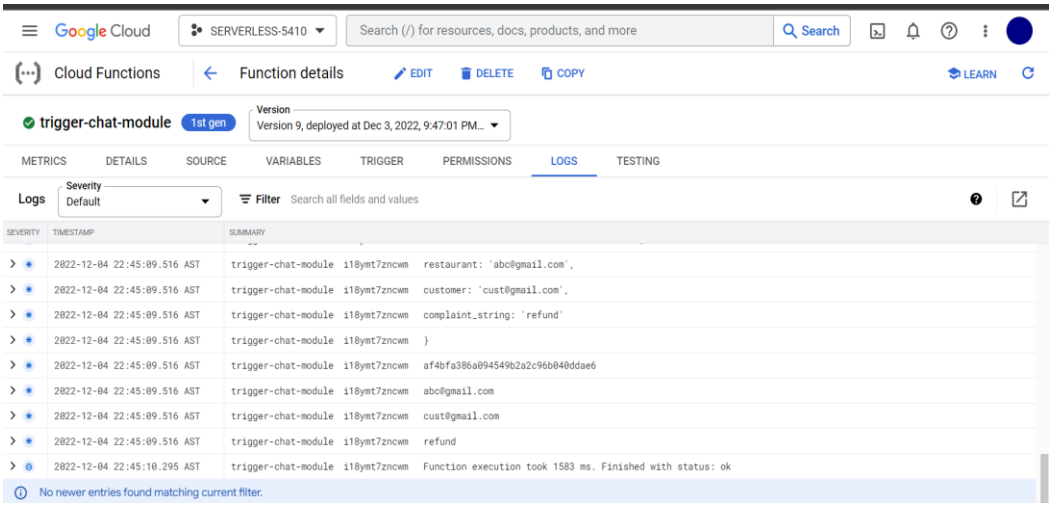

Figure 89: Logs of the cloud function, showing the cloud function was triggered by message on topic and details of message printed on the log to be same as sent by chatbot in case of complaint intent

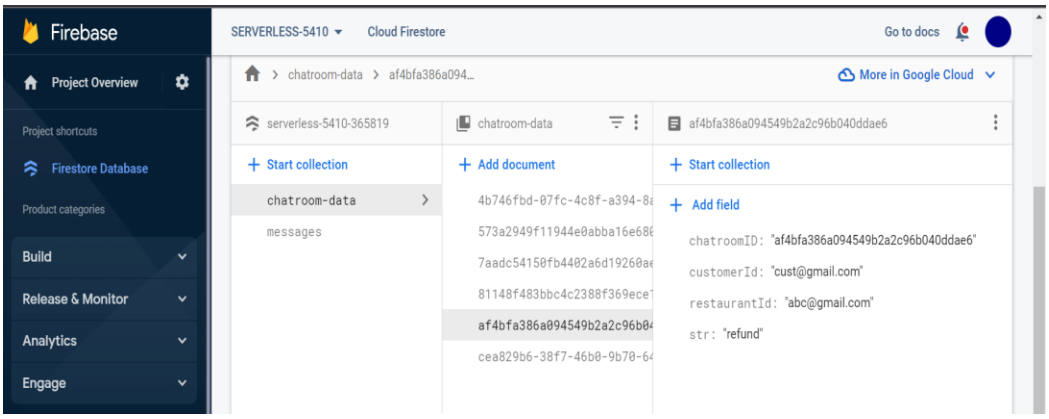

Figure 90: entries about the chatroom successfully added to the "chatroom-data" table

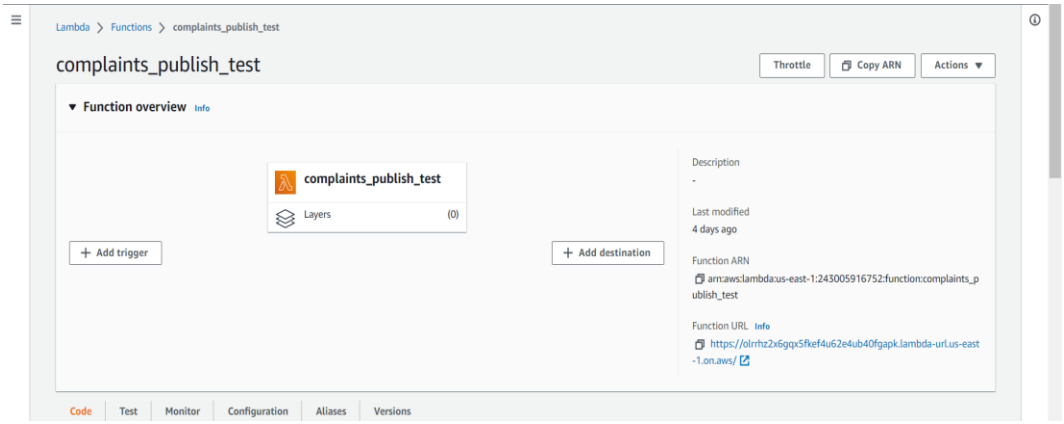

Figure 91: publisher lambda function, that publishes to the google pub sub topic

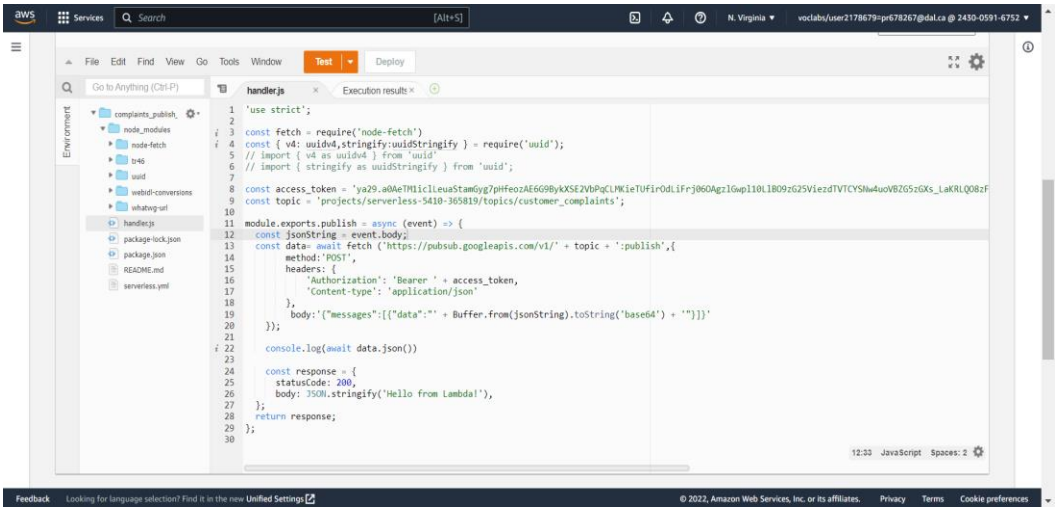

Figure 92: snippet of code in publisher lambda function , the data is published using the REST POST call

2. Application Roadmap

2.1 Registration

The users of the application are of two types: customer and restaurant owners, Although the front-end page presented to both the users will be different, the data collection for authentication would be same.

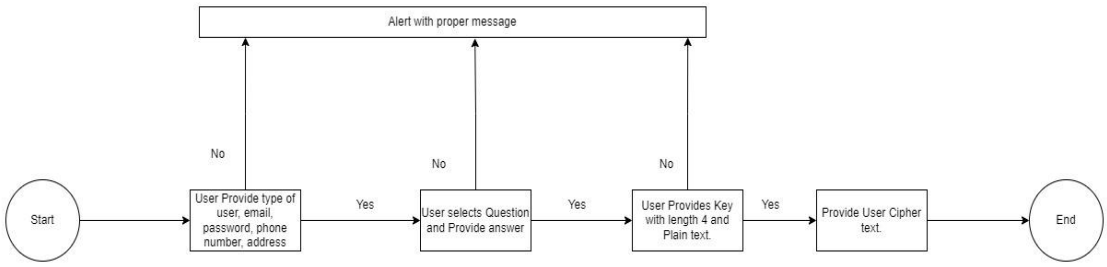

Figure 93: Application Module for Registration Module [1]

Table 11: What was planned v/s What was built for registration.

What was planned	What was built
1st Factor registration Using AWS Cognito	1st Factor registration Using AWS Cognito
2nd Factor registration Using GCP Cloud	2nd Factor registration Using GCP Cloud
3rd Factor registration Using AWS lambda	3rd Factor registration Using AWS Lambda

2.2 Authentication

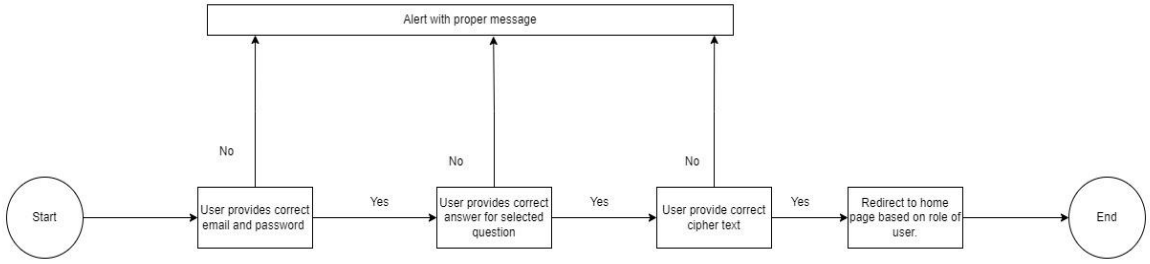

Figure 94: Application Module for Authentication Module [1]

Table 12: What was planned v/s What was built for authentication.

What was planned	What was built
1st Factor authentication Using AWS Cognito	1st Factor authentication Using AWS Cognito
2nd Factor authentication Using GCP Cloud	2nd Factor authentication Using GCP Cloud
3rd authentication Factor Using AWS lambda	3rd authentication Factor Using AWS Lambda

2.3 Online Support

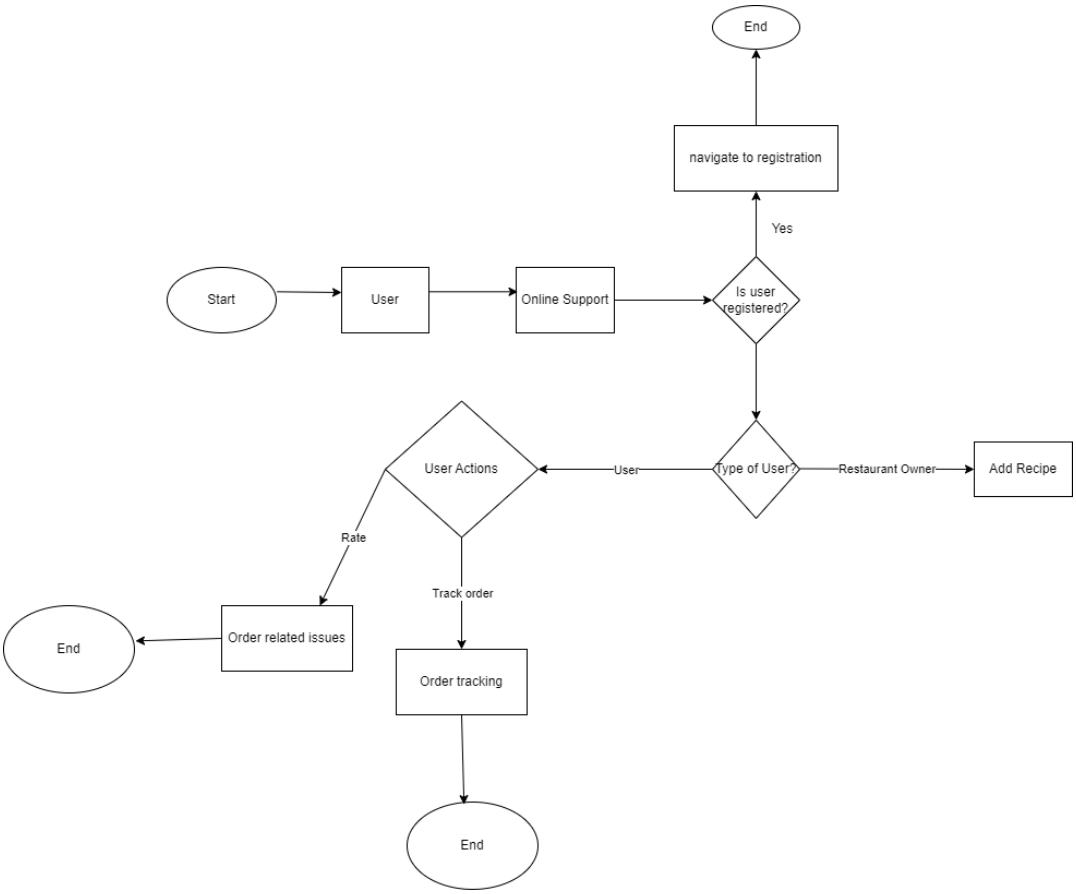

Figure 95: Application diagram for Online Support Module [1]

Table 13: What was planned v/s What was built for Online support module.

What was planned	What was built
Verify user and track food order.	Verify user and track food order.
Verify user and add food order rating.	Verify user and add food order rating.

Verify restaurant owner and add food recipe and its cost.	Verify restaurant owner and add food recipe and its cost.
Add customer complaints.	Add customer complaints.

2.4 Chat Module

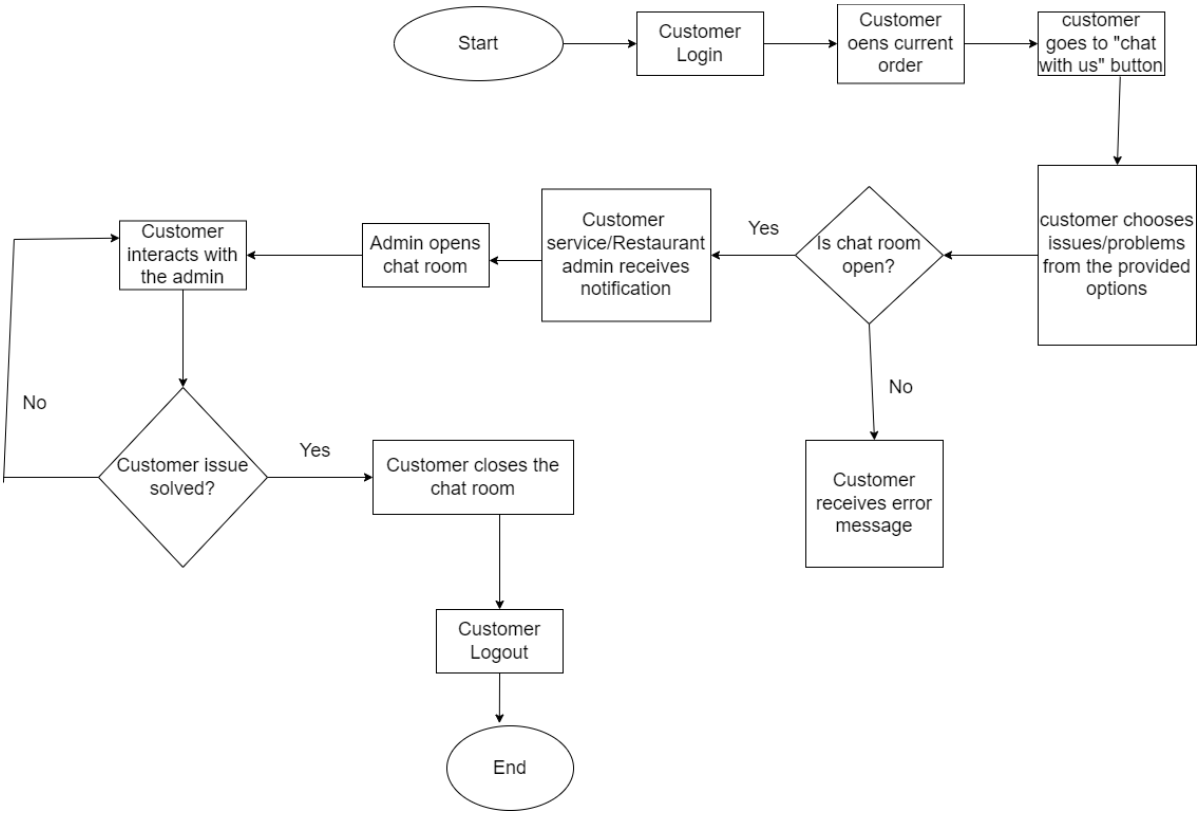

Figure 96: Application diagram for Chat Module [1]

Table 14: What was planned v/s What was built for Chat Module

What was planned	What was built
Customer signs in.	Customer signs in.
Generate chatroom id to open the chat session.	Generate chatroom id to open the chat session.
Customer chats with the Customer Service/Restaurant admin.	Customer chats with the Customer Service/Restaurant admin.
Customer signs out.	Customer signs out.

2.5 Data Processing

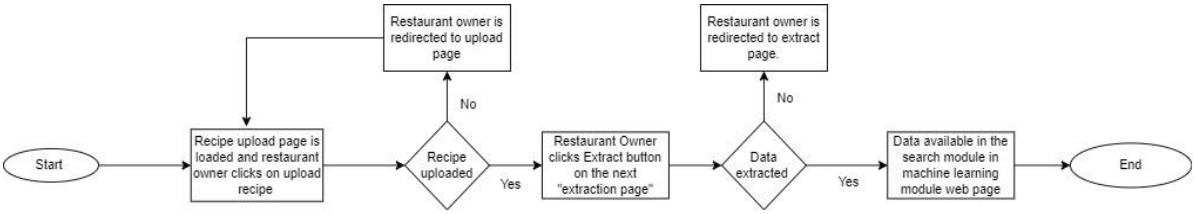

Figure 97: Application diagram for Data processing Module [1]

Table 15: What was planned v/s What was built for Data processing

What was planned	What was built
Upload recipe file to S3 bucket	Upload recipe file to S3 bucket
Extract button triggering lambda to extract named entities from recipe and store the details about recipe and entities to DYNAMODB	Extract button triggering lambda to extract named entities from recipe and store the details about recipe and entities to DYNAMODB

2.6 Machine learning

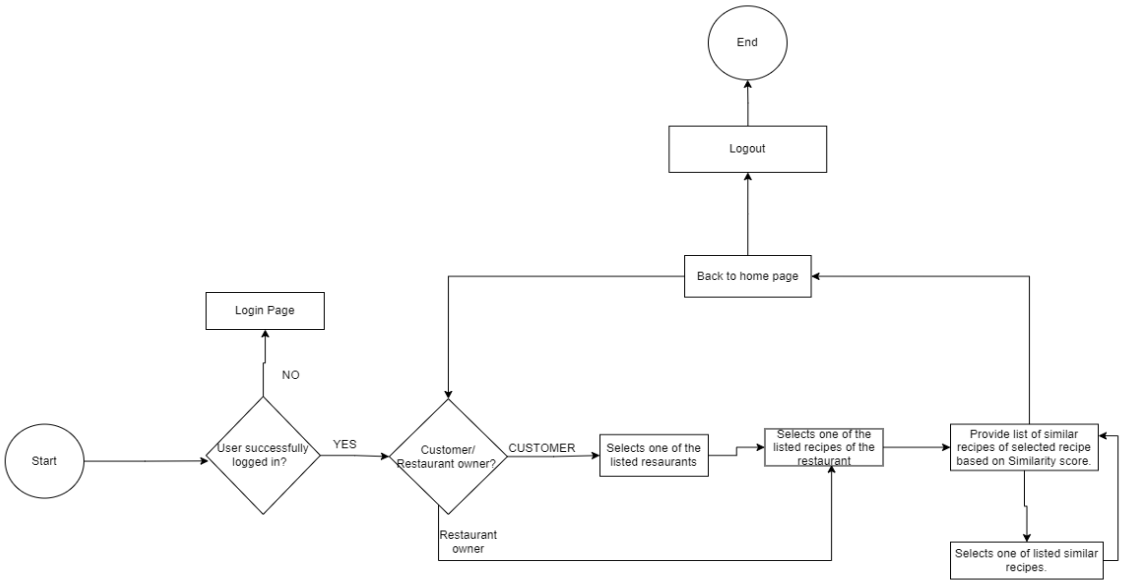

Figure 98: Application Diagram for Similarity Score [1]

Table 16: What was planned v/s What was built for Similarity Score.

What was planned	What was built
Similarity score using GCP AutoML	Similarity score using GCP AutoML

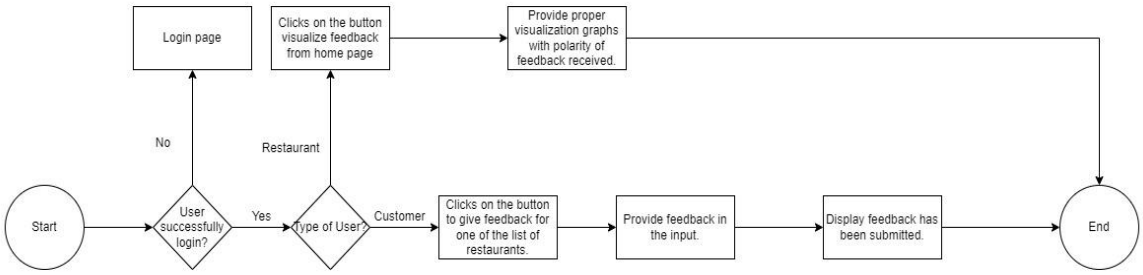

Figure 99: Application Diagram for Polarity of Customer feedback [1]

Table 17: What was planned v/s What was built for Customer Feedback polarity.

What was planned	What was built
Customer feedback polarity using AWS Comprehend	Customer feedback using AWS Comprehend
Customer feedback polarity visualization using AWS Quicksight	Customer feedback polarity visualization using GCP Datastudio

2.7 Visualization

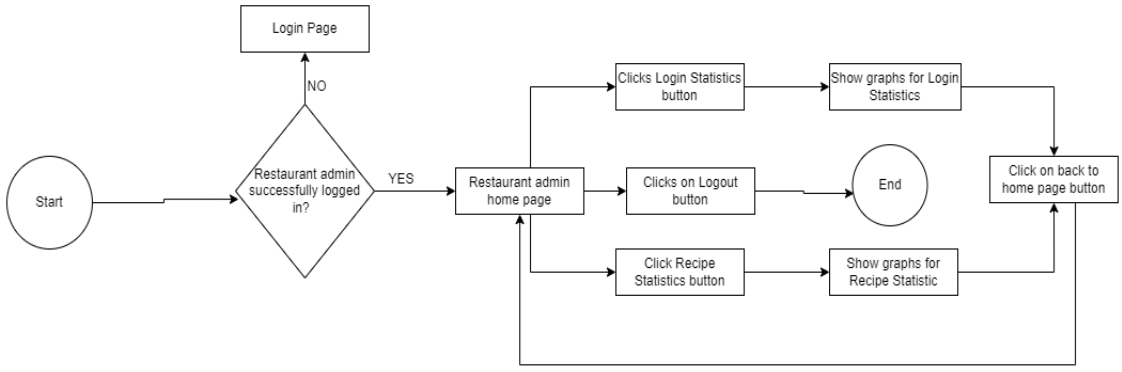

Figure 100: Application Diagram for Visualization [1]

Table 18: What was planned v/s What was built for Visualization.

What was planned	What was built
Login statistics visualization using GCP Datastudio	Login statistics visualization using GCP Datastudio
Recipe visualization using GCP Datastudio	Recipe visualization using GCP Datastudio

3.8 Message Passing

No application diagram for this feature as it will not be accessible from the front end and is internal to the system. The lambda attached to lex chat bot on fulfillment of “complaints” intent would do a post request to the publisher “LAMBDA” function, which would then publish the data to the GCP pub subtopic. The pub subtopic on arrival of messages, would trigger the attached “PUSH” subscriber and this subscriber would store the basic details about chatroom data to the firestore, which can be used by the chat module to get the details about the customer and restaurant in conversation.

Table 19: What was planned v/s What was built for GCP pub sub

What was planned	What was built
Scenario: 1	Scenario: 1
LEX publishing message to the GCP pub sub topic.	Lambda attached to Lex publishing data to a publisher lambda, which in turn would publish data to the GCP pub sub topic.
GCP cloud function initiating chatroom	The Lambda attached to lex would return a chatroom ID which would be used to chat with respective restaurants and google cloud function attached to the topic as a subscriber would create an entry with chatroom ID, restaurant ID, customer ID and complaint string to create a entry in firebase, which would be fetched by the chat module for the further initiation.

3. Final Architecture

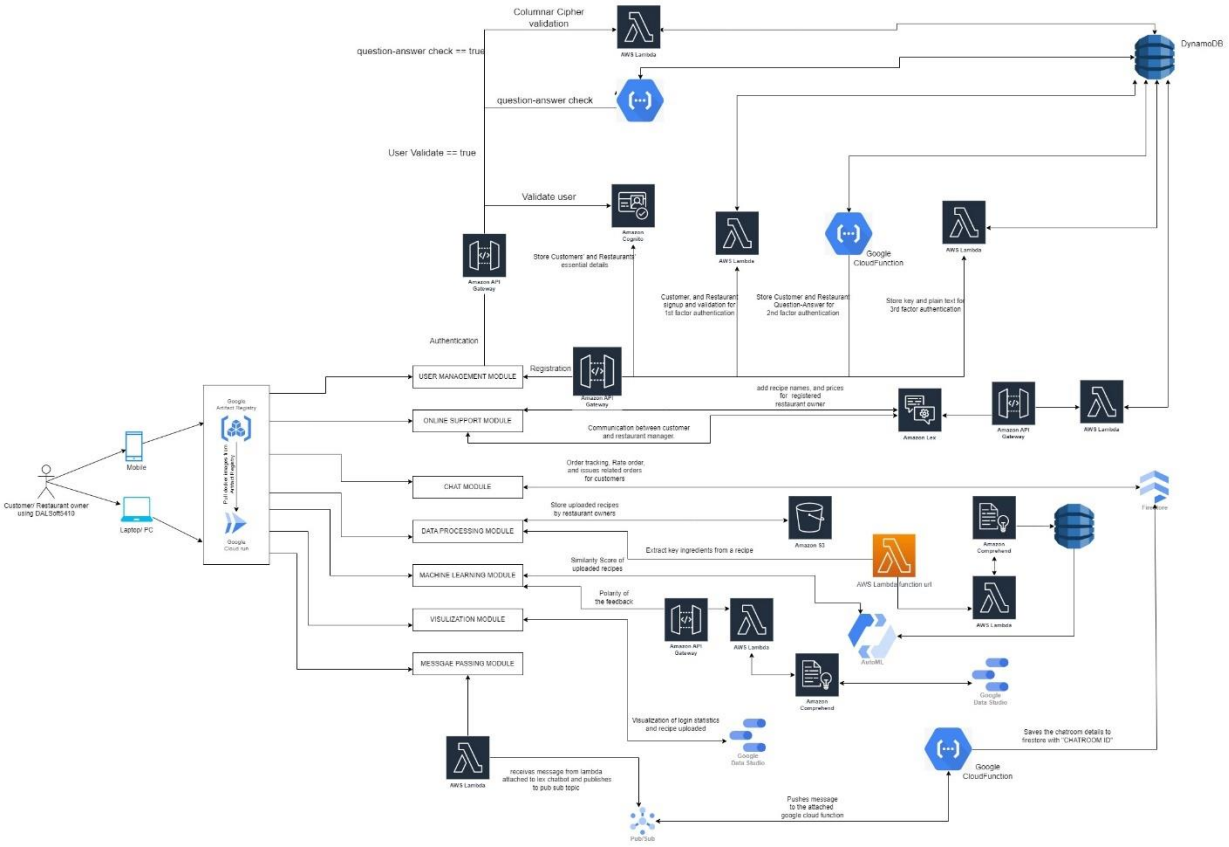

Figure 101: Cloud Architecture for DALSoft5410 [1].

4. Worksheet

4.1 Worksheet table

Table 20: Worksheet

Module	Team member
User management	Ketul Patel
Online Support	Faiza Umatiya
Chat	Faiza Umatiya
Data processing	Parul Raich
Machine learning	Ketul Patel
Visualization	Ketul Patel
Message passing	Parul Raich
Documentation	All team members

4.2 Commits Screenshots

Commit from Ketul:

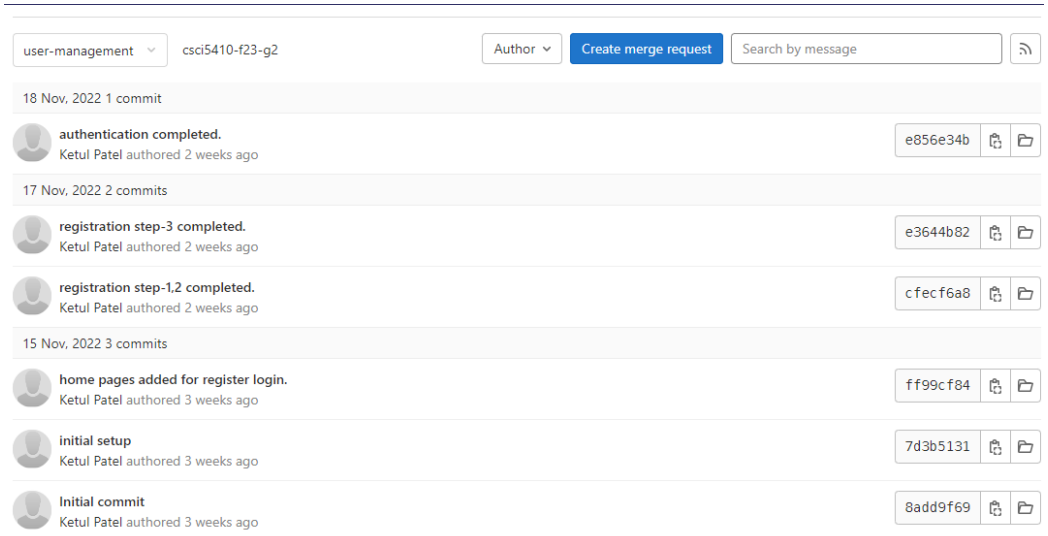

Figure 102: User-Management Branch.

Commits from Faiza:

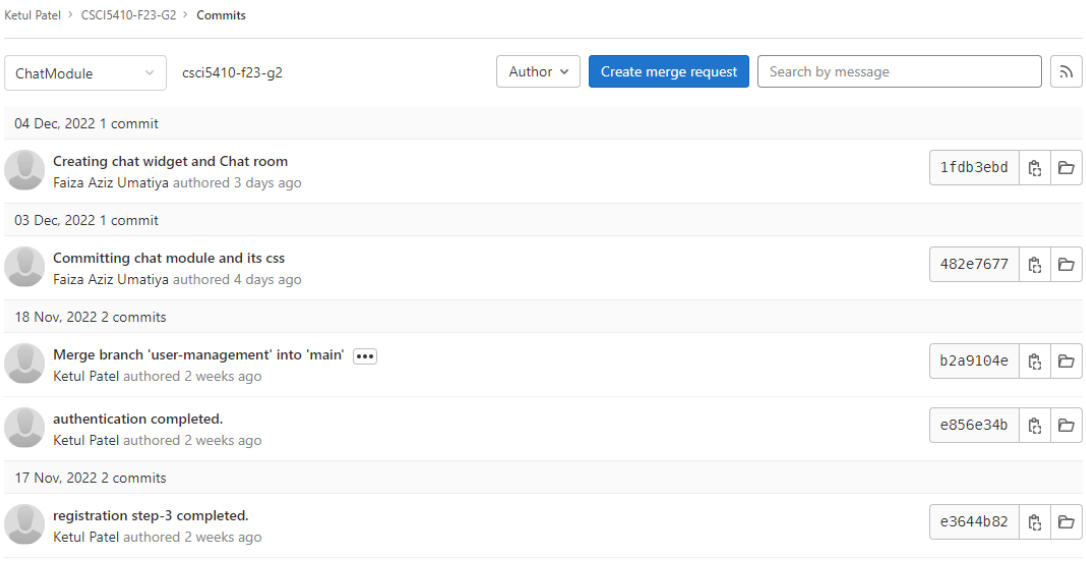

Figure 103: ChatModule branch.

Commits from Parul:

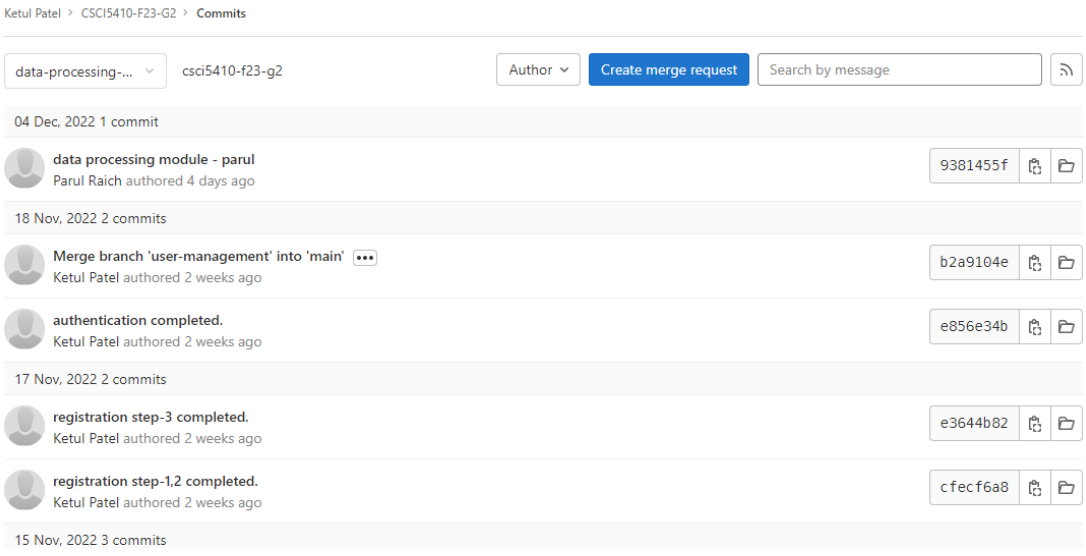

Figure 104: Data-processing branch.

Commits from team to dev branch:

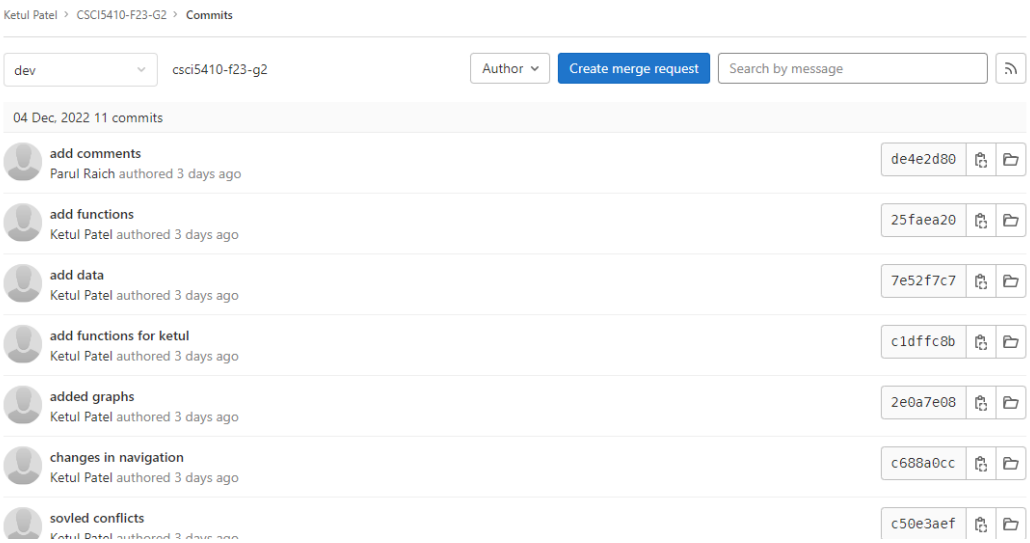

Figure 105: Development branch.

Commits graph for the project:

Figure 106: Commit graph from the team.

Merge Requests from team:

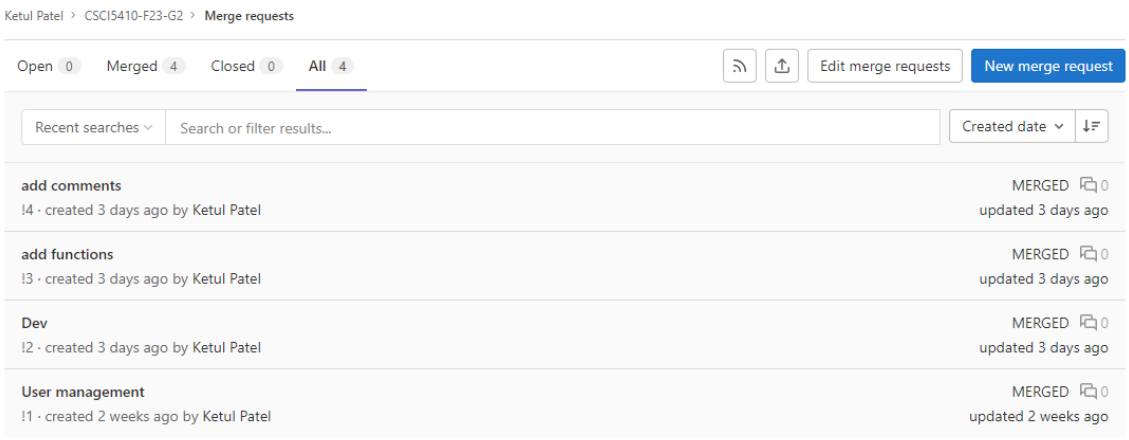

Figure 107: List of merge requests.

5. Sprint Plan

Table 21: Sprint plan

Sprint	Date	Task
1	Oct 01 - Oct 10	• Meet and greet for project planning • Requirement gathering and project analysis • Exploring cloud technologies and distributing modules amongst us.
2	Oct 12 - Oct 21	• Setting up the environment and Project configuration • Architecture planning and designing project structure.
3	Oct 23 - Nov 04	• User management Module implementation • Chat module implementation
4	Nov 05- Nov 14	• Building the front-end of the application • Preliminary stage of deployment and testing.
5	Nov 15- Nov 25	• Final testing and deployment • Start with the Report documentation and visualization.
6	Nov 26 - Dec 02	• Preparation of final Report and design documentation. • Preparation of final video presentation.

6. Demo URL

https://halifaxfoodie-g2-image-e3yr4sqsba-uc.a.run.app/

7. Meeting Logs

7.1. Meeting 1: Offline

We met offline at Goldberg Computer Science building for initial set up for the application on 14th of November and decided the flow of the application.

7.2. Meeting 2: Offline

We met at Goldberg Computer Science building on the November 25th to know about how much part of the project is done and how much is left.

7.3 Meeting 3: Offline

We met at Goldberg Computer Science building for final integration of the project on December 3rd.

7.4 Meeting 4: Offline

We met at Goldberg Computer Science building for demo video of the project on December 4th.

8. References

[1] “Flowchart maker & online diagram software,” Diagrams.net. [Online]. Available: https://app.diagrams.net/. [Accessed: 24-Oct-2022].

[2] “What is Pub/Sub?,” Google Cloud. [Online]. Available: https://cloud.google.com/pubsub/docs/overview. [Accessed: 24-Oct-2022].

[3] Wikipedia contributors, “Amazon DynamoDB,” Wikipedia, The Free Encyclopedia, 09-Oct-2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Amazon_DynamoDB&oldid=1115015722.

[4] Amazon.com. [Online]. Available: https://aws.amazon.com/cognito/. [Accessed: 24-Oct-2022].

[5] “Cloud functions,” Google Cloud. [Online]. Available: https://cloud.google.com/functions. [Accessed: 24-Oct-2022].

[6] Amazon.com. [Online]. Available: https://aws.amazon.com/s3/. [Accessed: 24-Oct-2022].

[7] Y. Li, Z. Wang, B. Ding, and C. Zhang, “AutoML: A Perspective where Industry Meets Academy,” in Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 2021.

[8] Amazon.com. [Online]. Available: https://aws.amazon.com/comprehend/. [Accessed: 24-Oct-2022].

[9] Amazon.com. [Online]. Available: https://aws.amazon.com/quicksight/. [Accessed: 24-Oct-2022].

[10] “Looker studio connect to data,” Google.com. [Online]. Available: https://datastudio.google.com/data. [Accessed: 24-Oct-2022].

[11] Amazon.com. [Online]. Available: https://aws.amazon.com/lambda/. [Accessed: 24-Oct-2022].

[12] “Firebase API reference,” Firebase. [Online]. Available: https://firebase.google.com/docs/reference/. [Accessed: 24-Oct-2022].

[13] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/lexv2/latest/dg/what-is.html. [Accessed: 24-Oct-2022].

[14] Åhlén, “Columnar Transposition Cipher (online tool) | Boxentriq,” Columnar Transposition Cipher (online tool) | Boxentriq. [Online]. Available: https://www.boxentriq.com/code-breaking/columnar-transposition-cipher. [Accessed: Oct. 24, 2022]

Appendix

Appendix A

Name: **PARUL RAICH**
Banner id: **B00903203**

What did I learn?

I worked on the "Data Processing" and "Message Passing" modules in the group project. Reactjs was new to me and learning to integrate and write code in javascript felt challenging at first but official documents and existing starter code snippets provided in them made the learning process easy. While working on the data processing module, I learned about various methods that could be used to add data to S3 and Dynamo DB. I used the "AWS SDK" method, imported the AWS-SDK into the project, and used the in-built methods from it to add files to the bucket. Furthermore, for data processing, the usage of "AWS Comprehend" was crucial. I got to learn about "AWS comprehend", the various inbuilt functions it provides, the possibilities of adding custom algorithms in comprehend, and how to integrate it with a lambda function. The project also helped me understand how to work in multi-cloud architecture and components. For my message passing module, for one of the scenarios, I had to publish a message using the AWS lambda that custom user data from the lex chatbot to the GCP pub-sub topic and then consume the message from the pub-sub subscribers. This whole experience gave me clarity on how the message-passing concept works in a serverless environment. The whole learning experience was made more enriching by my teammates, who helped me solve any technical difficulties by sharing their own similar experiences.

What went wrong?

While working on the project I faced various problems that made my learning deeper in the subject. The tutorials that were available online on various topics used AWS endpoint to expose the various services they wanted. But I realized that as a LabRole user of AWS academy, I could not use the AWS endpoints and after reading through many documentations and blogs on alternatives for it, I got to know about the "function URL" method for exposing the AWS lambda. Since the front-end application offered through our application was in reactjs, I got to know about the various configurations such as "CORS" that I needed to enable before exposing services through a publicly available to be allowed to accept requests from various other services uninterruptedly. In addition, another issue I faced was choosing the type of subscribers for the scenarios that had to be considered for the pub sub-module. Although the official documentation and videos turned out to be helpful for me to understand the topic in great depth and clarity. Another important problem I faced was in the integration of the pub sub module with the online support and chat module, but the existing blogs and documentation again proved to help guide me in various ways to successfully integrate and test the integration. Another problem I faced was sometimes the google cloud function and AWS lambda's existing compilers/interpreters were unable to locate certain inbuilt or imported modules, to solve this I explored the method of adding various layers on lambda or even uploading a whole new small code with dependencies in lambda functions and google cloud functions.

What was my unique approach?

The approach I took for the pub-sub module was based on the REST-based methods in order to publish the message to the topic and I published the message from a Cross-platform function that is "AWS lambda" to the topic. Although usage of the available client libraries would have helped make the whole process of publishing a little easy but using the REST methods and the await fetch method of the nodejs library inside the Nodejs AWS LAMBDA, made me more clarity and control of the data to be published on the topic.

How does the project map into my course learning process?

The main objective of the course was on various serverless services and how to integrate the multi-cloud serverless services and projects in many instances demanded of the same, which helped me understand the whole need and concept of this in a clearer way. Working on this project helped me get an industry-level hands-on experience in a serverless based environment and various challenges and solutions to overcome those challenges.

Appendix B

Name: **FAIZA UMATIYA**
Banner id: **B00899642**

What did I learn?

For the CSCI-5410 Serverless Data Processing project, my module was Online Support Module and Chat Module. While working on the project I got to learn many technologies and AWS services like AWS Lex, AWS Lambda, AWS DynamoDB, Firestore, Google cloud functions and GCP Pub/Sub. For the Online Support Module, I learnt more about AWS lambda and AWS lex which is the main component for building bot. Apart from the backend, I learnt react for the front-end. There were many errors and problems I faced while working with the chatbot. The most common error I faced was the intent fallback which I can easily resolve by checking logs from the CloudWatch. Apart from this I faced few errors from the front-end such as integrating the AWS lex to the homepage of the front-end. But because of these difficulties I got to learn more about the module which helped me improved my knowledge and cleared the concepts. Additionally, working with my team members was best experience as they helped me whenever I get stuck and helped me stay calm in stressful situations.

What went wrong?

There were many problems I faced while working on my modules. For Online Support Module, I created AWS Lambda functions for each feature. For instance, lambda function to verify user, lambda function to track food order, lambda function to add food recipes and so on. With this approach I wasn't able to connect all the lambdas to jump from one feature to another. So, I made a single lambda function having all the features. Another problem I faced with the same module was the integration of the AWS Lex with the front-end. This was because of the AWS policy issues where I wasn't permitted to add the access token key of the particular region. This made me unable to add Lex to the front-end. For Chat Module, I faced problem in generating chatroom id using function URL. After trying multiple times on the demo code and realized the problem, I tried generating the chatroom id from the final lambda function and it worked. Lastly, during Chat Module integration, I faced problem with the CSS, as the style was not only affecting the Chatbot, but it was changing the CSS of the other modules too. But we tried solving the problem and resolved it before the time of submission.

What was my unique approach?

The approach I took for the Online Support Module is to create individual lambda functions for each feature. By doing this, I can test each feature individually and can resolve any errors easily without going in depth looking for the errors in other feature. For instance, if I want to track the food order, then I will only call the lambda function to track the order and test the bot. I implemented same approach for the others lambda and solved the errors individually. Lastly, I merged all the lambda functions for a better flow and deployed it. This caused less error and was resolved easily. Apart from this, I added extra components like recipe type to have some data for visualization. For the customer complaints intent, we had different lambda function for it, so I connected this lambda function with the other lambda functions using function URL. Because the customer complaints lambda function was written in Node.js and other lambda functions were written in Python.

How does the project map into my course learning process?

Because of this Serverless project, I got to learn about different services and technologies like AWS and GCP services. I got to learn which services works better for which scenarios like when to use GCP Pub/Sub and AWS SNS/SQS. As my module was the Online Support Module and Chat Module, I got to learn about how I can create a dynamic and static chatroom for the user as well as the admin. Additionally, I got to learn how Backend-as-a-Service works through this Serverless project.

Appendix C

Name: **KETUL PATEL**
Banner id: **B00900957**

What did I learn?

To implement the project, I implemented User Management Module, Machine Learning, and Visualization model. Firstly, I had to interact with AWS Cognito where I learned about User Pools and how they work and can be helpful in a project where someone does not need to build a backend part to store user credentials Then, implementation of the second part which was related to question and answer where I used cloud functions to interact with Firestore database that helped to get insights of how cloud function can be used to interact with GCP services. Adding to that module, for the last part I stored the other details such as name, phone number, and address of the User to the DynamoDB database using lambda function where I learned about how boto3 services can be used to interact with DynamoDB database. Secondly, the implementation of the Machine Learning module helped me to get to know about the GCP AutoML services where I uploaded text files and then trained the model. To get a prediction from the model, I used the GCP cloud function that can interact with the endpoint of the trained model. Moreover, I learned about AWS Comprehend which helped to get the polarity of the feedback entered by User and GCP data studio which was used to visualize the polarity in form of graphs. Lastly, I learned about the GCP data studio and different ways to pass the data to the GCP data studio that can help to make interactive graphs for Login statistics and Recipe Uploaded by the Restaurant Owner on dashboard page of Restaurant Owner.

What went wrong?

I had to face two main issues while implementation of whole project. The first problem was trying to implement the cloud function. As cloud function implementation was new to me and I had not prior experience working with it, I tried to use the inline editor provided by GCP Cloud Function run it online. However, to update minor changes in the code, deployment time was nearly 3 to 5 minutes. If there was syntax problem in writing cloud function, then the deployment would fail, and the changes were removed from the code automatically. Moreover, it was very hard for me to debug the GCP cloud function as logs were no efficient as they were in AWS lambda function. The second issue I faced during the implementation was I had to use gspread module from Python to store data to the Google spreadsheet to make visualization graphs as I had not access of the AWS Quick Sight. I had to add layer in AWS lambda function to use to gspread module. I tried different approaches for that. However, nothing was working for me. After 6 hours of hard work, I was finally able to add layer and use gspread module.

What was my unique approach?

According to me, the unique approach was to run and debug the cloud function locally rather than running it online. After searching for methods to debug, I got to know about function-framework that can be used in react to run the cloud functions in the local environment. The way was, I was running my project on port 3000 and cloud function on port number 8080. Then, I was making the HTTP request to that port number to get the output. The process helped to save a ton of time and made it easier for me to debug the GCP cloud function.

How does the project map into my course learning process?

I was always curious to know about cloud computing and that was one of the main reasons I enrolled in the course to gain practical knowledge. The project implementation helped me to gain knowledge about various GCP and AWS cloud services. As the project was multi-Cloud based, it helped me how I can use multiple

cloud services in one project and connect them to make an efficient web application. In fact, things I learnt during lectures and lectures slides eased my project implementation.