



# DALHOUSIE UNIVERSITY

## Sprint 3 Team Report

### Team 36

Christin Saji - B00977669

Jay Jagdishbhai Patel - B00981520

Jeet Jani - B00972192

Kuldeep Rajeshbhai Gajera – B00962793

**GitLab Repo Link:** [https://git.cs.dal.ca/patel45/serverless\\_group\\_36/-/tree/main?ref\\_type=heads](https://git.cs.dal.ca/patel45/serverless_group_36/-/tree/main?ref_type=heads)

## Table of Contents

<b>DETAILS OF RESEARCH .....</b>	<b>4</b>
MODULE 3: MESSAGE PASSING .....	4
MODULE 4: NOTIFICATION .....	4
MODULE 5: DATA ANALYSIS AND VISUALIZATION .....	5
MODULE 6: WEB APPLICATION BUILDING AND DEPLOYMENT.....	6
<b>HOW WE HAVE IMPLEMENTED:.....</b>	<b>7</b>
MODULE 1: USER MANAGEMENT & AUTHENTICATION MODULE .....	7
MODULE 2: VIRTUAL ASSISTANCE MODULE .....	7
MODULE 3: MESSAGE PASSING .....	8
MODULE 4: NOTIFICATION .....	9
MODULE 5: DATA ANALYSIS AND VISUALIZATION .....	10
MODULE 6: WEB APPLICATION BUILDING AND DEPLOYMENT.....	12
<b>INDIVIDUAL CONTRIBUTION:.....</b>	<b>13</b>
<b>GITLAB CODE REPOSITORY:.....</b>	<b>18</b>
<b>SYSTEM ARCHITECTURE: .....</b>	<b>18</b>
<b>PSEUDOCODE/ALGORITHM FOR IMPORTANT MODULES' LOGIC:.....</b>	<b>19</b>
MODULE 2: VIRTUAL ASSISTANT .....	19
<i>HandoffCustomerSupport</i> .....	19
MODULE 3: MESSAGE PASSING .....	20
<i>AdminFetcher</i> .....	20
<i>AgentAssigner</i> .....	21
<i>ChatLogger</i> .....	24
<i>ChatRetriever</i> .....	26
MODULE 5: DATA ANALYSIS AND VISUALIZATION .....	28
<i>Lambda Function: Post Feedback</i> .....	28
<i>Cloud Function: Extract Sentiment</i> .....	30
<i>Lambda Function: post-authentication-trigger</i> .....	31
<i>Lambda Function: storeLoginStatistics</i> .....	32
<i>Lambda Function: fetchUserData</i> .....	33
<b>TEST CASES: .....</b>	<b>34</b>
LAMBDA TEST POST FEEDBACK EVENT: .....	34
CLOUD FUNCTION EXTRACT SENTIMENT TEST EVENT: .....	35
LAMBDA TEST POST AUTHENTICATION LOGGING EVENT: .....	35
LAMBDA TEST STORE LOGIN STATISTICS EVENT: .....	36
LAMBDA TEST FETCH USER DATA EVENT: .....	37
HANDOFFCUSTOMERSUPPORT .....	38
AGENTASSIGNER: .....	39
ADMINFETCHER: .....	39
CHATLOGGER.....	40
CHATRETRIEVER .....	40
NOTIFICATION TESTING USING SNS PUBLISH TOPIC .....	41
<b>API TESTING: .....</b>	<b>42</b>
POST FEEDBACK:.....	42
EXTRACT SENTIMENT: .....	43

STORE LOGIN STATISTICS:.....	43
FETCH USER DATA:.....	44
FETCH ADMINS: .....	44
LOG CHATS: .....	45
RETRIEVE CHATS:.....	45
<b>EVIDENCE OF TESTING FOR COMPLETED MODULES:.....</b>	<b>46</b>
NOTIFICATION MODULE TESTING:.....	46
DATA ANALYSIS AND VISUALIZATION MODULE: .....	52
<i>All users can access Feedback of Room:</i> .....	52
<i>Property Agent Side:</i> .....	55
<i>Property Agent Side:</i> .....	59
<i>Customer Side updated chat:</i> .....	60
<b>REFERENCES.....</b>	<b>69</b>

## Details of Research

### Module 3: Message Passing

Research Area	Source	URL
Creating a Pub/Sub Topic	Google Cloud Pub/Sub	<a href="https://cloud.google.com/publish/docs/create-topic">https://cloud.google.com/publish/docs/create-topic</a>
Publishing in a Pub/Sub Topic from AWS	Google Cloud Pub/Sub	<a href="https://cloud.google.com/publish/docs/publisher">https://cloud.google.com/publish/docs/publisher</a>
Setting a Cloud Function as a Subscriber	Google Cloud – Cloud Functions	<a href="https://cloud.google.com/functions/docs/calling/publish#:~:text=If%20you%20are%20deploying%20using%20the%20Google%20Cloud%20console%2C%20you,trigger%20calls%20to%20your%20function.">https://cloud.google.com/functions/docs/calling/publish#:~:text=If%20you%20are%20deploying%20using%20the%20Google%20Cloud%20console%2C%20you,trigger%20calls%20to%20your%20function.</a>
Storing Information in Firestore using Cloud Functions	Firebase - Firestore	<a href="https://firebase.google.com/docs/firestore/extend-with-functions">https://firebase.google.com/docs/firestore/extend-with-functions</a>
Implementing Chat System using Firestore	FreeCodeCamp	<a href="https://www.freecodecamp.org/news/building-a-real-time-chat-app-with-reactjs-and-firebase/">https://www.freecodecamp.org/news/building-a-real-time-chat-app-with-reactjs-and-firebase/</a>

### Module 4: Notification

Research Area	Source	URL
What is Amazon SNS?	AWS Documentation	<a href="https://docs.aws.amazon.com/sns/latest/dg/welcome.html">https://docs.aws.amazon.com/sns/latest/dg/welcome.html</a>
What is Amazon SQS?	AWS Documentation	<a href="https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html">https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html</a>
Sending Amazon SNS messages to an Amazon SQS queue in different accounts	AWS Documentation	<a href="https://docs.aws.amazon.com/sns/latest/dg/sns-sqs-as-subscriber.html">https://docs.aws.amazon.com/sns/latest/dg/sns-sqs-as-subscriber.html</a>

How to set up a serverless notification system on AWS	Medium, Towards Data Science	<a href="https://lo0o0p.medium.com/build-a-serverless-push-and-sms-notification-service-on-aws-f92fdbf7ece3">https://lo0o0p.medium.com/build-a-serverless-push-and-sms-notification-service-on-aws-f92fdbf7ece3</a>
---	------------------------------	---

## Module 5: Data Analysis and Visualization

Research Area	Source	URL
AWS Cognito SDK to interact with Cognito Services	NPM (Node Package Manager) Repository	<a href="https://www.npmjs.com/package/amazon-cognito-identity-js">https://www.npmjs.com/package/amazon-cognito-identity-js</a>
Exploring different APIs and Features available to call for different task	GCP Cloud Natural Processing Documentation	<a href="https://cloud.google.com/natural-language/docs/basics#interpreting_sentiment_analysis_values">https://cloud.google.com/natural-language/docs/basics#interpreting_sentiment_analysis_values</a>
Calling analyzeSentiment API using NodeJS.	GCP Cloud Natural Processing Documentation	<a href="https://cloud.google.com/natural-language/docs/analyzing-sentiment">https://cloud.google.com/natural-language/docs/analyzing-sentiment</a>
To Know more about NPM library offered by Google for NLP Tasks	NPM Documentation of GCP NLP Library	<a href="https://www.npmjs.com/package/@google-cloud/language">https://www.npmjs.com/package/@google-cloud/language</a>
Cognito Post Authentication Lambda Trigger	AWS Documentation	<a href="https://docs.aws.amazon.com/cognito/latest/developerguide/user-pool-lambda-post-authentication.html">https://docs.aws.amazon.com/cognito/latest/developerguide/user-pool-lambda-post-authentication.html</a>
Update Google Sheets using Apps Script	Google Sheets Documentation	<a href="https://developers.google.com/sheets/api/guides/batchupdate">https://developers.google.com/sheets/api/guides/batchupdate</a>
Connect Google Sheets with Locker Studio for Data Analytics and Visualizations	Locker Studio Support	<a href="https://support.google.com/looker-studio/answer/7450249?hl=en#zippy=%2Cin-this-article">https://support.google.com/looker-studio/answer/7450249?hl=en#zippy=%2Cin-this-article</a>
Features	Locker Studio Documentation	<a href="https://developers.looker.com/embed/getting-started">https://developers.looker.com/embed/getting-started</a>

## Module 6: Web Application Building and Deployment

Research Area	Source	URL
Google Cloud Run	Google Cloud Run Documentation	<a href="https://cloud.google.com/build/docs/deploying-builds/deploy-cloud-run">https://cloud.google.com/build/docs/deploying-builds/deploy-cloud-run</a>
Using Google Cloud Build Service to build Docker Image and push to Google Artifact Repository	Google Cloud Build Documentation	<a href="https://cloud.google.com/build/docs/building/build-containers">https://cloud.google.com/build/docs/building/build-containers</a>
Create a Gitlab pipeline to push to Google Artifact Registry	Gitlab Documentation	<a href="https://docs.gitlab.com/ee/tutorials/create_gitlab_pipeline_push_to_google_artifact_registry/">https://docs.gitlab.com/ee/tutorials/create_gitlab_pipeline_push_to_google_artifact_registry/</a>
Use Gitlab CI/CD to build Application	Gitlab Documentation	<a href="https://docs.gitlab.com/ee/topics/build_your_application.html">https://docs.gitlab.com/ee/topics/build_your_application.html</a>

# How we have Implemented:

## Module 1: User Management & Authentication Module

We have extended this module further by adding Post Authentication Trigger which will log the user login statistics in the DynamoDB Table. This stored login statistics will help us to visualize further in Data analysis and Visualization Module.

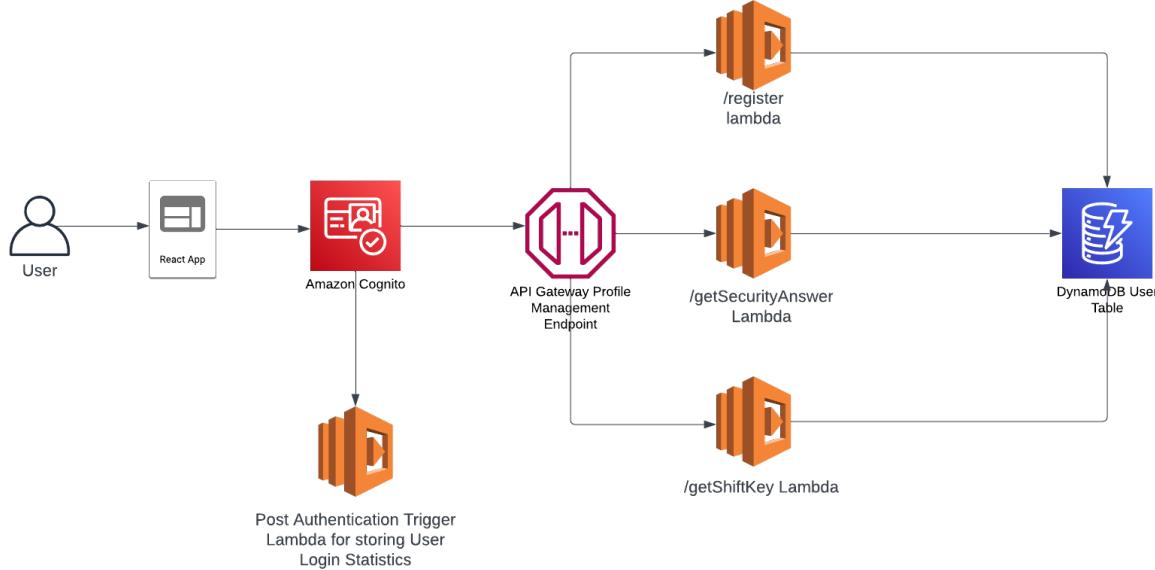


Figure 1: Architecture Diagram of User Management and Authentication Module

## Module 2: Virtual Assistance Module

This module was extended in Sprint 3 to fulfil the customer concern functionality. HandoffCustomerSupport was modified to publish the user and concern details to GCP Pub/Sub topic [1][2]. IntentHandler and FetchBookingDetails were modified to support the changes made for role-based response generation.

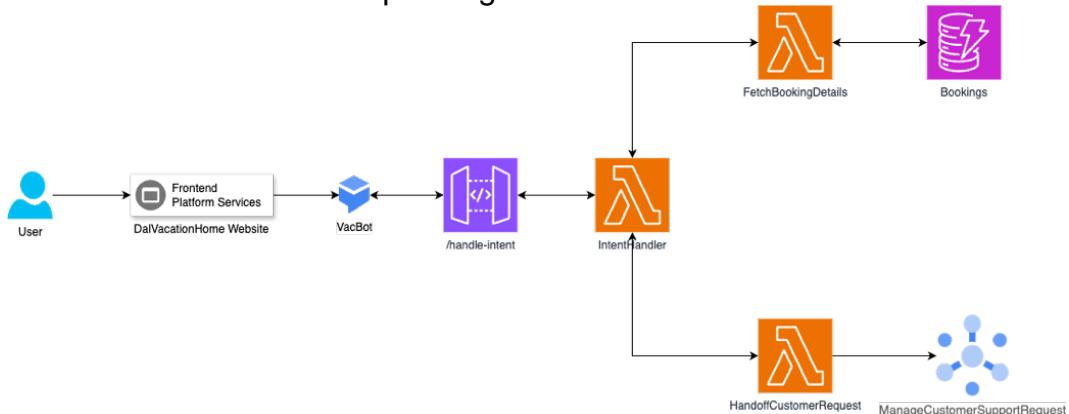


Figure 2: Architecture Diagram of Virtual Assistant Module

## Module 3: Message Passing

As implemented in the previous module, when a user posts their concern to the chatbot, the HandoffCustomerSupport Lambda function dealt with sending back an appropriate response to the user. This module comprises two parts - publishing the concern to create a ticket for the user and implementing a live chat.

The first functionality is an extension of the second module and closely associated with it. HandoffCustomerSupport will publish the user details like the email id, concern along with other metadata through a GCP Pub/Sub topic - ManageCustomerSupportRequest [1][2]. A Cloud Function – AgentAssigner, is subscribed to this topic and receives the information passed on by it [3].

Following that, AgentAssigner will fetch the list of admins from the DynamoDB table User, using a Lambda Function – AdminFetcher, and assign a random agent to this customer support request. It will create and store this new concern in a collection called Issues inside the default Firestore database [4]. Each support request is stored as a separate issue, having a unique, auto-assigned document ID. That concludes the first functionality.

The second feature involves creating a new page in the frontend which users can use to keep track of all their posted concerns and chat with the assigned agent. Every message that is sent by the user or the agent, is sent as an event to a Cloud Function called ChatLogger. This adds every message as a JSON object inside the chat array, created by AgentAssigner. Another Cloud Function – ChatRetriever fetches all the previous chats between them.

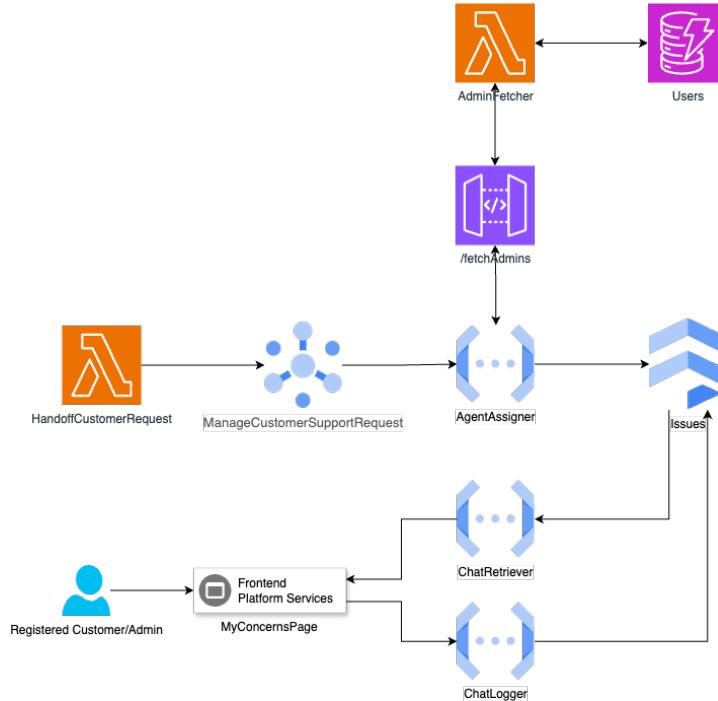


Figure 3: Architecture Diagram of Message Passing Module

## Module 4: Notification

In the implementation of our notification module, we leveraged a combination of Amazon Simple Notification Service (SNS), Amazon Simple Queue Service (SQS), and AWS Lambda to create a robust and scalable notification system [6][7]. This design choice was driven by the need for reliable real-time communication to users for various system events such as registration confirmations and booking updates. SNS facilitates the management and dispatch of notification messages to different channels, ensuring broad reach and high deliverability. Each notification type is associated with a specific SNS topic that routes messages to an SQS queue, which serves as a buffer to absorb any fluctuations in message volume and maintain system resilience against spikes in demand [8].

Upon arrival of messages in SQS, AWS Lambda functions are triggered to process these messages asynchronously [9]. This setup allows for the decoupling of notification logic from the application's core functionality, enhancing maintainability and scalability. The Lambda functions handle the task of parsing messages from SQS, formatting them appropriately, and executing the delivery mechanism, which could range from sending emails, pushing mobile notifications, or interfacing with other AWS services for additional actions. The use of SQS ensures that no message is lost, even during periods of high load, providing a fail-safe mechanism that reinforces the reliability of our notification system. This architecture not only meets our current needs but is also designed to scale seamlessly as user demand grows, exemplifying a modern cloud-native approach to application development and system design.

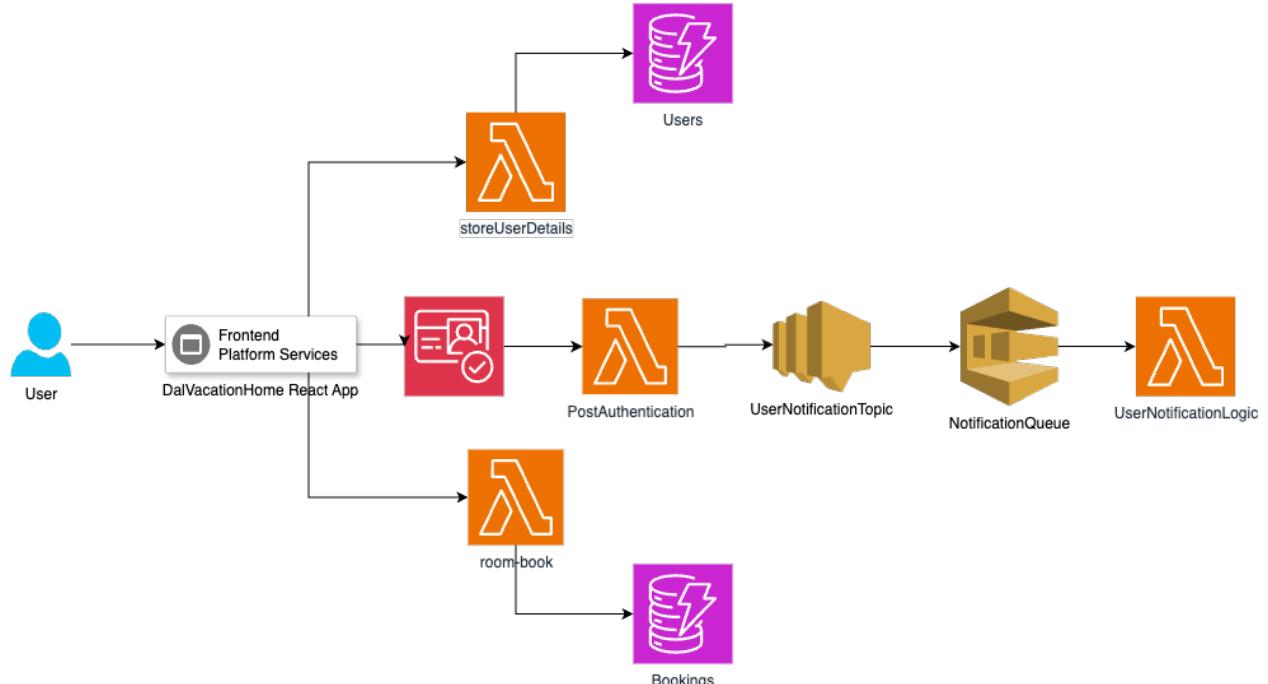


Figure 4: Architecture diagram for notification module

## Module 5: Data Analysis and Visualization

After going over the official documentation of various cloud services, we got an overall idea of how we will be implementing the Data Analytics and Visualization module.

We have used Google Looker Studio, which allows us to present our raw data in informative, easy-to-read ways such that we can get insights from the data by visualizing the data using different diagrams and displayed pie charts, and interactive table. This tool is used for Data Analytics and Visualization purposes [16][17].

This tool provides a feature to visualize Google Sheets, so we have written a Google App Script that will periodically fetch data from Lambda function via API Gateway and populate the Google Sheets. We then use Looker Studio to analyze that Google Sheet and then embed that Looker Studio Report in React App using an iFrame [15].

According to the requirement, we have displayed Login Statistics of the users on the Admin Page. We have stored login statistics in the AWS DynamoDB after Login. After the successful login into the application using AWS Cognito, we have set up a post-authentication trigger which will run a Lambda function to store data about Login Statistics of the user in DynamoDB [14].

We also need to display feedback of each room in a tabular structure in the front end according to the requirement. To accomplish this, we have used Google NLP API, which we expose using GCP Cloud Function and called that cloud function in the AWS Lambda for analyzing the user feedback. So as soon as the user posts feedback for a particular room, it will first come in Lambda and in Lambda, we called the cloud function with the feedback text to analyze the sentiment and then we stored the feedback along with the score and magnitude of the feedback in DynamoDB [11][12].

Moreover, we also need to analyze all feedback and show the sentiment of each feedback in the front-end. For this, we have gone through the Google Natural Language Processing API Documentation. The Natural Language API has several methods for performing analysis and annotation on your text. Out of all, the analyzeSentiment method takes given text as input and in response provides score (-1 to 1) and magnitude (0 to 1) values. Different score values give different sentiment levels like Clearly positive (above 0.8), Clearly Negative (below -0.6), Neutral (0.1), Mixed (0.0) [11].

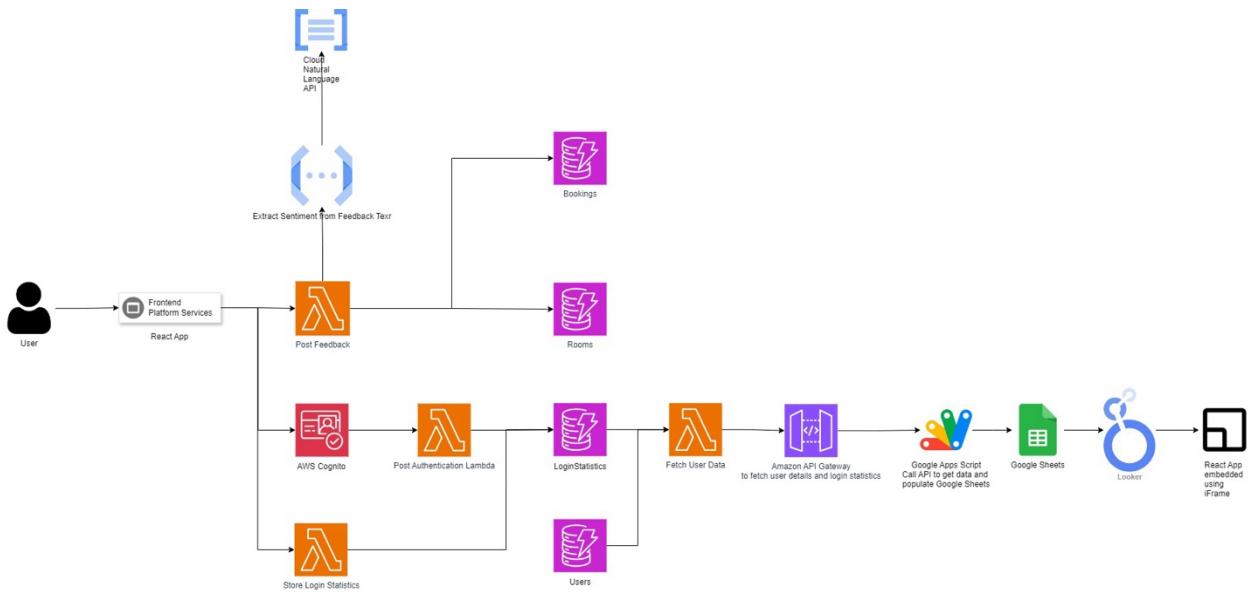


Figure 5: Architecture Diagram of Data Analytics and Visualization Module

## Module 6: Web Application Building and Deployment

After going over the official documentation of Google Cloud Run and blogs on how to deploy React App on Cloud Run, we have configured CI/CD pipeline from GitLab and deploy React App on Google Cloud Run.

Google Cloud Build is used for building images of applications, and that image is deployed on Cloud Run. We first need to create a Dockerfile to create the image of the React App and then push that image to Google Cloud Build or Google Cloud Repository, and then from that, we run a container created out of that image on Cloud Run. Every step is automated using GitLab CI/CD pipeline and terraform for automatic deployment of Google Cloud Resources and React App on Google Cloud Run [18][19][20][21]. Additionally, AWS resources are created using AWS CloudFormation, ensuring a robust and repeatable infrastructure setup.

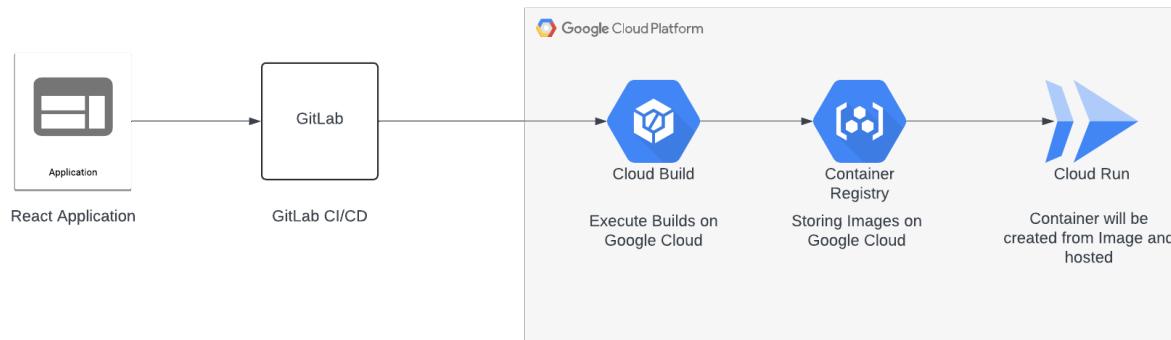


Figure 6: Architecture Diagram of Web Application Building and Deployment Module

# Individual Contribution:

**Christin Saji:**

## 1. Data Analysis and Visualization:

- Created 'post-authentication-logging' Lambda function to store user login statistics, triggered post-authentication in Cognito.
- Developed 'storeLoginStatistics' Lambda function to log user actions (security question completion, Caesar cipher completion, logout) via API Gateway.
- Stored login statistics in DynamoDB table 'LoginStatistics'.
- Integrated Lambda function to fetch user and login statistics from DynamoDB and transfer to Google Sheets via Apps Script, utilized in Looker Studio for frontend visualization.

## 2. Lambda Functions:

- **post-authentication-trigger:** Logs user login attempts in the DynamoDB.
- **storeLoginStatistics:** Appends login actions to DynamoDB.
- **fetchUserData:** Fetches user and login statistics from DynamoDB.

## 3. Infrastructure as Code (IaC):

- Created CloudFormation stacks for all Lambda functions and DynamoDB setup.
- Utilized Terraform to deploy Dialogflow virtual assistant.
- Stored Lambda function zip files in S3 and referenced them in CloudFormation templates for automated setup

## 4. Frontend Development:

- Integrated EventLoggingService to trigger login events and log via API Gateway to Lambda.
- Enhanced various aspects of the user interface for better usability and visual appeal and resolved chatbot rendering issues.

## 5. GitLab Board Management:

- Actively managed the progress of the Data Analysis and Visualization and Web Application and Deployment Module on the GitLab board.
- Regularly commented on the progress of tasks and made frequent commits to ensure transparency and continuous integration.
- Collaborated with team members to track milestones, review code, and ensure timely completion of project goals.

## 6. Documentation:

- Updated the Data Analysis and Visualization module architecture diagram and integrated that in final architecture diagram.
- Maintained comprehensive meeting logs for Sprint 3, capturing agendas, outcomes of discussions, and links to meeting recordings.

## **Jay Jagdishbhai Patel:**

### **1. Data Analysis and Visualization:**

- Implemented Google Cloud function and called the GCP NLP API using cloud language library to analyze the sentiment of the text and return the score and magnitude of it.
- And created a lambda function which will call Cloud function with feedback text as request and store the return response (score, magnitude) and feedback for a particular room in the DynamoDB Room Table.

### **2. Web Application Deployment:**

- Build CI/CD pipeline from Gitlab to Google cloud. Written script for gitlab-ci.yml, Dockerfile, Cloudbuild.yaml, and nginx.conf. Used Cloud Build for executing build process and create Docker image and push it in Artifact Registry and deploy on Cloud Run. These above steps are automated using Gitlab CI/CD pipeline.

### **3. Lambda Functions:**

- a. **post-feedback:** Allows Customers to provide feedback for room after their stay ends. Also written logic to process sentiment of feedback by fetching response from cloud function and store in DynamoDB.

### **4. Cloud Functions:**

- **extract-sentiment:** take feedback text as request and used GCP NLP API to extract sentiment out of it and return the score (-1 to 1) and magnitude (0 to 1).

### **5. Infrastructure as Code (IaC):**

- **User Management and Authentication Module:** Used Cloud formation to automatically create stack for entire Cognito module (even post authentication trigger lambda)
- **Virtual Assistance:** Used Terraform for creating script for automatic deployment of Dialog Flow virtual assistant service.
- **Pub/Sub Module:** Used Terraform for automatic deployment of Pub/Sub service.
- **Cloud Functions:** User Terraform for automatic deployment of all 4 Cloud functions needed in this project. Upload cloud function zip to cloud storage and reference the bucket and object name in Terraform Script for creating Cloud function.

### **6. Frontend Development:**

- Contributed to designing and developing the Data analysis and visualization frontend using React.
- Integrated the Lambda API endpoints with the frontend for real-time data interaction.

- Integrated Looker Studio in the frontend using iFrame for displaying analytics about login statistics.
- Connected Pub/Sub module in the frontend and enabled asynchronous chat between property agent and customer. Design frontend page for customer and property agent side to view raised tickets.

## **7. GitLab Board Management:**

- Managed the project workflow using the GitLab board.
- Organized tasks, tracked progress, and ensured timely completion of project milestones.
- Facilitated team collaboration and done code reviews.

## **8. Improve Code Quality and Commenting:**

Contributed to Improve Code Quality and add Doc String for every Lambda and Cloud Function.

## **Jeet Jani**

### **1. Virtual Assistant Module:**

- Updated the Dialogflow ES chatbot, defined the intents, parameters, entity types, training phrases and bot responses for various user requests like retrieving booking information, or submitting customer concerns.
- Updated Lambda functions to correctly handle the backend logic for various user requests.
- Updated an API endpoint for the IntentHandler Lambda function and set it as the webhook URL for connecting the chatbot with the function, and sending parameters passed by the users.

### **2. Message Passing Module:**

- Created Pub/Sub topic ManageCustomerSupportRequest to Handoff customer support request to a Cloud Function set as a subscriber, which can proceed with the flow.
- Created AgentAssigner to assign fetch admins and assign a random agent to manage the customer concern.
- Created a Lambda function to assist AgentAssigner to fetch agent details.
- Wrote cloud functions to store and retrieve chats from a Firestore collection to offer users a complete chat system.

### **3. Lambda Functions**

- Created AdminFetcher function to solve the multi-cloud credentials security problem. Cloud function AgentAssigner will invoke AdminFetcher using the endpoint /fetchAdmins, and AdminFetcher will query the Users DynamoDB table. It will search for users with the role = 1 and return the retrieved results to AgentAssigner.
- Updated HandoffCustomerSupport function to handle the Customer Support Request Intent. It hands over the concern and user metadata to the GCP

- Pub/Sub topic ManageCustomerSupportRequest, and notifies the user that the query has been submitted.
- Updated IntentHandler function to receive the parameter passed by the user from the bot. IntentHandler determines what intent has been called and calls other Lambda functions to execute the necessary logic. It also returns the output given by these Lambda functions back to the chatbot interface.
- Updated FetchBookingDetails function to handle the Booking Info Intent. It queries the DynamoDB table, Users, and returns the room details if found in the table. Else returns the message that no such booking has been made.

#### **4. Cloud Functions:**

- Wrote AgentAssigner which is triggered whenever a message is published in ManageCustomerSupportRequest. Then it calls AdminFetcher to fetch the agents' list. It will assign a random agent to the concern and store the metadata along with the assigned agent details in a Firestore collection - Issues. It will also create an empty array of chats which will be updated if any chat is recorded between the user and the assigned agent.
- Wrote ChatLogger that logs the messages sent by the user and the agent in the Issues collection. The chat array gets appended with this new message and metadata in the form of JSON documents. Each document represents a separate message.
- Wrote ChatRetriever that retrieves all the chats between the user and the agent for every concern requested.

#### **5. GitLab Board Management:**

- Used GitLab board to manage project tasks.
- Kept track of tasks, progress, and made sure we met deadlines.
- Helped the team work together and reviewed code.

#### **6. Documentation:**

- Designed architecture diagrams for the Virtual Assistant module and the entire project, including plans for future service integrations.
- Assisted teammates in formatting the final report in a well-structured manner.

## **Kuldeep Gajera**

### **1. User Management & Authentication Module with AWS Cognito:**

- Implemented user and property agent sign-up features on the frontend. This involved integrating the signup forms with AWS Lambda functions and DynamoDB to handle data storage and retrieval securely and efficiently.
- Provided support to team members for smooth integration of user management features.

### **2. Lambda Functions:**

- **StoreUserDetails:** The function for storing user registration details in DynamoDB, ensuring data consistency and security.

- **GetShiftKey:** The function to retrieve a user's shift key from DynamoDB, facilitating secure user verification processes.
- **GetSecurityAnswer:** The function to fetch the security answer to a user's security question from DynamoDB, crucial for user authentication.
- **DeleteRoomWithImage:** The function to delete a room and its associated image from Amazon S3, maintaining the integrity and freshness of room data.
- **UpdateRoomDetailsWithImage:** The function to update room details and optionally an image in S3, allowing dynamic updates to room listings.
- **UserNotificationLogic :** The function to send notifications to end users upon successful registration, successful login, successful booking and booking failer.

### **3. Notification Module:**

- Designed and implementation of a scalable notification system using AWS SNS, SQS, and Lambda, ensuring efficient communication across user activities such as registrations, logins, and bookings. I configured SNS topics linked to SQS queues to handle message surges and implemented Lambda functions for processing these messages.
- Contribution extended to integrating these services to trigger notifications for key user actions—successfully registering, logging in, confirming bookings, and handling booking failures. This integration was critical in enabling real-time, reliable notifications that enhanced the user experience by providing timely feedback on their interactions with our application.

### **4. Documentation**

- Added references and wrote captions for documentation images.
- Helped format and manage the project report for clarity and professionalism.

# GitLab Code Repository:

[https://git.cs.dal.ca/patel45/serverless\\_group\\_36/-/tree/main?ref\\_type=heads](https://git.cs.dal.ca/patel45/serverless_group_36/-/tree/main?ref_type=heads)

## System architecture:

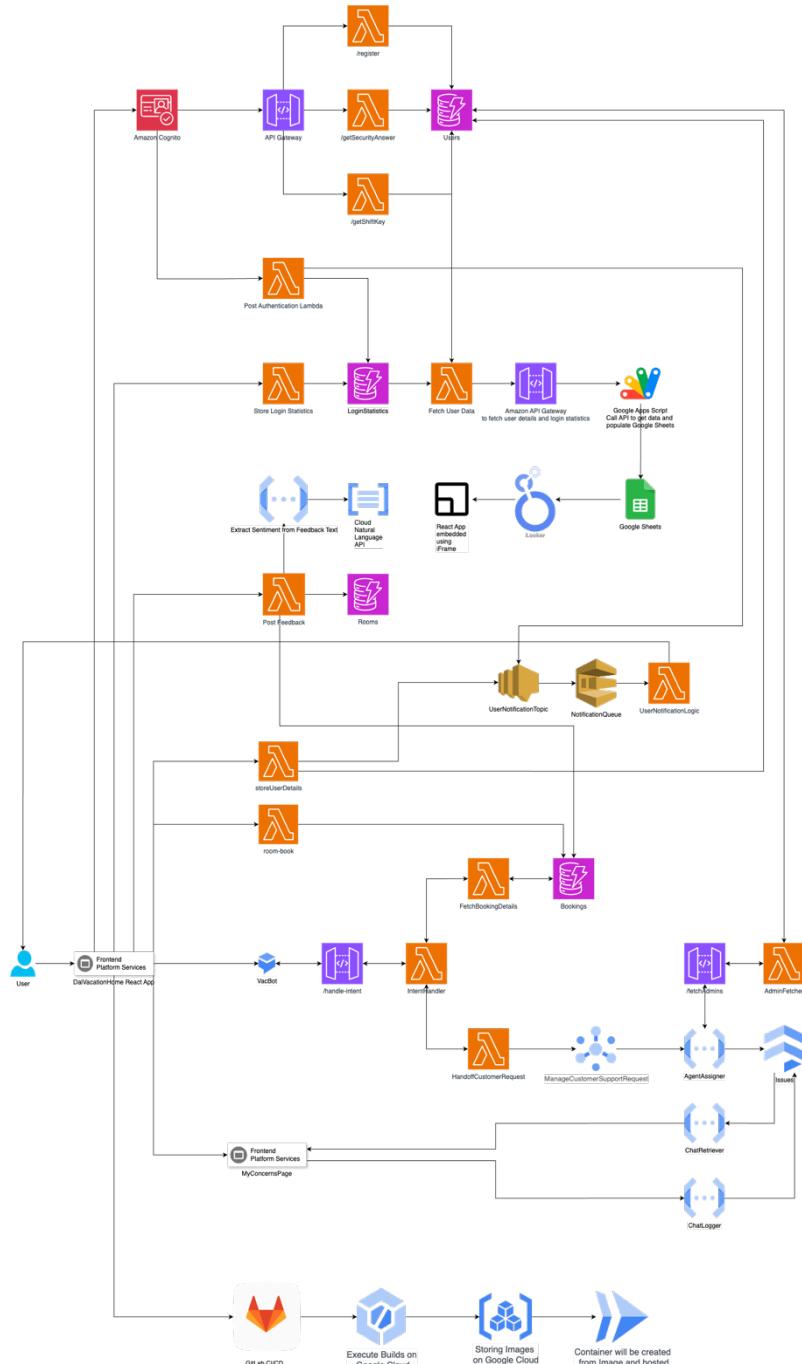


Figure 7: System Architecture Diagram of Application

# Pseudocode/Algorithm for Important Modules' logic:

## Module 2: Virtual Assistant

### HandoffCustomerSupport

*Pseudocode:*

```
FUNCTION lambda_handler(event):
```

```
TRY:
```

```
    IF event is a string:
```

```
        Parse event to JSON and assign to body
```

```
    ELSE:
```

```
        Assign event to body
```

```
    Extract issue from body.Issue
```

```
    Extract bookingReferenceCode from body.BookingReferenceCode
```

```
    Log triggered by IntentHandler
```

```
    Log issue received
```

```
    Log booking reference code received
```

```
Store issue and bookingReferenceCode in DynamoDB table 'CustomerConcerns'
```

```
    RETURN success message 'Your query has been submitted. A representative  
will contact you shortly.'
```

```
CATCH error:
```

```
    Log error
```

```
    RETURN failure message 'There was an error processing your request. Please  
try again later.'
```

```
END FUNCTION lambda_handler
```

### *Algorithm*

1. Initialize Variables
2. Parse Event
  - a. If event is a string:
    - i. Parse event to JSON and assign to body.
  - b. Else:
    - i. Assign event to body.
3. Extract Parameters
  - a. Extract issue from body.Issue.
  - b. Extract bookingReferenceCode from body.BookingReferenceCode.
4. Log Details
  - a. Log 'triggered by IntentHandler'.
  - b. Log issue received.
  - c. Log booking reference code received.

5. Store Customer Concerns
  - a. Store issue and bookingReferenceCode in DynamoDB table 'CustomerConcerns'.
6. Return Response
  - a. Return success message 'Your query has been submitted. A representative will contact you shortly.'
7. Handle Errors
  - a. In the catch block:
    - i. Log error.
    - ii. Return failure message 'There was an error processing your request. Please try again later.'

## Module 3: Message Passing

AdminFetcher

*Pseudocode:*

```

FUNCTION handler(event)
  INITIALIZE AWS SDK
  INITIALIZE DynamoDB DocumentClient

  SET params
    TableName = 'Users'
    FilterExpression = '#role = :role'
    ExpressionAttributeNames = {'#role': 'role'}
    ExpressionAttributeValues = {':role': '1'}

  TRY
    LOG 'Fetching admins with params:' and params
    FETCH data FROM dynamoDb.scan(params)
    EXTRACT admins FROM data.Items

    LOG 'Fetched Admins:' and admins

  RETURN
    statusCode = 200
    body = JSON.stringify(admins)
  CATCH error
    LOG 'Error fetching admins:' and error
    RETURN
      statusCode = 500
      body = JSON.stringify({ error: 'Could not fetch admins' })
  END FUNCTION

```

*Algorithm:*

1. Initialize AWS SDK
2. Initialize DynamoDB DocumentClient.
3. Set Parameters
4. Define params with the following properties:
  - a. TableName set to 'Users'.
  - b. FilterExpression set to '#role =
5. '.
  - a. ExpressionAttributeNames with '#role' mapped to 'role'.
  - b. ExpressionAttributeValues with '
6. ' mapped to '1'.
7. Log Fetching Details
8. Log the message 'Fetching admins with params:' followed by the params.
9. Fetch Data
10. Use dynamoDb.scan(params).promise() to fetch data from DynamoDB.
11. Extract and Log Admins
12. Extract admins from data.Items.
13. Log the message 'Fetched Admins:' followed by the admins.
14. Return Success Response
15. Return an object with:
  - a. statusCode set to 200.
  - b. body set to JSON stringified admins.
16. Handle Errors
17. In the catch block:
  - a. Log the message 'Error fetching admins:' followed by the error.
  - b. Return an object with:
    - i. statusCode set to 500.
    - ii. body set to JSON stringified { error: 'Could not fetch admins' }.

## AgentAssigner

*Pseudocode:*

IMPORT necessary libraries and modules

INITIALIZE Firestore client

FUNCTION fetch\_admins  
    SET url to API endpoint  
    TRY  
        SEND GET request to url  
        CHECK response for HTTP errors

```

PARSE response JSON to get admins
RETURN admins
CATCH RequestException as e
LOG error message
RETURN empty list

FUNCTION hello_pubsub(cloud_event)
DECODE pubsub_data from base64
TRY
PARSE pubsub_data as JSON to get message_data

EXTRACT bookingReferenceCode, issue, and email from message_data

LOG bookingReferenceCode, issue, and email

CALL fetch_admins to get list of admins
IF admins list is not empty
SELECT a random admin from the list
EXTRACT name and email of the selected admin
LOG selected admin's name and email
ELSE
SET assigned_admin_name to "No Admin Available"
SET assigned_admin_email to "No Admin Available"
LOG "No Admin Available"

INITIALIZE empty chats list
GET current date and time as dateRaised
SET isActive to True

ADD document to Firestore 'Issues' collection with extracted and generated
data
GET document ID of the added document
LOG success message with document ID
CATCH JSONDecodeError as e
LOG error message
CATCH Exception as e
LOG error message

```

*Algorithm:*

1. Import Libraries
2. Import necessary libraries: base64, json, random, requests, functions\_framework, google.cloud.firestore, and datetime.
3. Initialize Firestore Client
4. Initialize a Firestore client.
5. Define fetch\_admins Function
6. Set API Endpoint

- a. Define url for the API endpoint.
- 7. Try to Fetch Admins
  - a. Send a GET request to the API endpoint.
  - b. Check for HTTP errors.
  - c. Parse the response JSON to get admins.
  - d. Return the list of admins.
- 8. Handle Errors
  - a. Catch RequestException.
  - b. Log the error message.
  - c. Return an empty list.
- 9. Define hello\_pubsub Function
- 10. Decode Pub/Sub Data
  - a. Decode pubsub\_data from base64.
- 11. Try to Process Data
  - a. Parse pubsub\_data as JSON to get message\_data.
  - b. Extract bookingReferenceCode, issue, and email from message\_data.
  - c. Log bookingReferenceCode, issue, and email.
  - d. Fetch Admins
    - i. Call fetch\_admins to get the list of admins.
    - ii. If admins is not empty:
      - 1. Select a random admin from the list.
      - 2. Extract the name and email of the selected admin.
      - 3. Log the selected admin's name and email.
    - iii. Else:
      - 1. Set assigned\_admin\_name to "No Admin Available".
      - 2. Set assigned\_admin\_email to "No Admin Available".
      - 3. Log "No Admin Available".
  - e. Initialize Variables
    - i. Initialize an empty chats list.
    - ii. Get the current date and time as dateRaised.
    - iii. Set isActive to True.
  - f. Store Issue in Firestore
    - i. Add a document to the Firestore 'Issues' collection with the extracted and generated data.
    - ii. Get the document ID of the added document.
    - iii. Log a success message with the document ID.
- 12. Handle Errors
  - a. Catch JSONDecodeError.
  - b. Log the error message.
  - c. Catch other exceptions.
  - d. Log the error message.

## ChatLogger

### Pseudocode:

```
IMPORT necessary libraries and modules

INITIALIZE Firestore client

DEFINE hello_http(request)
    ALLOW cross-origin requests

    PARSE request JSON body

    IF request JSON is empty
        RETURN error response "Invalid request, JSON body is required" with
        status 400

    TRY
        EXTRACT concernId and chats from request JSON

        IF concernId is empty
            RETURN error response "Invalid request, concernId is required" with
            status 400

        IF chats is empty or not a list
            RETURN error response "Invalid request, 'chats' should be a list and
            cannot be empty" with status 400

        FOR each chat in chats
            IF chat does not have 'from', 'message', and 'timestamp'
                RETURN error response "Invalid request, each chat must have 'from',
                'message', and 'timestamp'" with status 400

        FETCH existing issue document from Firestore using concernId
        IF document does not exist
            RETURN error response "Issue not found" with status 404

        CONVERT document to dictionary

        APPEND new chats to existing chats in the issue document

        UPDATE issue document in Firestore with the new chats array

        RETURN success response "Chats updated successfully" with status 200

    CATCH Exception as e
        RETURN error response with exception message and status 500
```

*Algorithm:*

1. Import Libraries
2. Import necessary libraries: functions\_framework, google.cloud.firestore, flask.jsonify, flask.request, and flask\_cors.cross\_origin.
3. Initialize Firestore Client
4. Initialize a Firestore client.
5. Define hello\_http Function
6. Allow Cross-Origin Requests
  - a. Decorate the function with @cross\_origin() to allow cross-origin requests.
7. Parse Request JSON
  - a. Parse the JSON body from the request.
  - b. If the JSON body is empty:
    - i. Return an error response with message "Invalid request, JSON body is required" and status 400.
8. Try to Process Data
  - a. Extract concernId and chats from the request JSON.
  - b. If concernId is empty:
    - i. Return an error response with message "Invalid request, concernId is required" and status 400.
  - c. If chats is empty or not a list:
    - i. Return an error response with message "Invalid request, 'chats' should be a list and cannot be empty" and status 400.
  - d. Validate Chats
    - i. For each chat in chats:
      1. If the chat does not have 'from', 'message', and 'timestamp':
        - a. Return an error response with message "Invalid request, each chat must have 'from', 'message', and 'timestamp'" and status 400.
  - e. Fetch Existing Issue Document
    - i. Fetch the existing issue document from Firestore using concernId.
    - ii. If the document does not exist:
      1. Return an error response with message "Issue not found" and status 404.
  - f. Append and Update Chats
    - i. Convert the document to a dictionary.
    - ii. Append new chats to the existing chats in the issue document.
    - iii. Update the issue document in Firestore with the new chats array.
  - g. Return Success Response
    - i. Return a success response with message "Chats updated successfully" and status 200.
9. Handle Errors
  - a. Catch exceptions.
  - b. Return an error response with the exception message and status 500.

## ChatRetriever

*Pseudocode:*

```
IMPORT necessary libraries and modules

INITIALIZE Firestore client

DEFINE ChatRetriever(request)
    ALLOW cross-origin requests

    PARSE request JSON body and request arguments

    INITIALIZE customer_email and agent_email to None

    IF request JSON body exists
        SET customer_email from request JSON
        SET agent_email from request JSON
    ELSE IF request arguments exist
        SET customer_email from request arguments
        SET agent_email from request arguments

    IF customer_email is None AND agent_email is None
        RETURN error response "customerEmail or agentEmail is required!" with
        status 400

    TRY
        INITIALIZE reference to Firestore 'Issues' collection

        IF customer_email exists
            CREATE query to filter issues by customer_email
        ELSE
            CREATE query to filter issues by agent_email

        EXECUTE query and get documents

        INITIALIZE empty issues list

        FOR each document in query result
            CONVERT document to dictionary
            ADD document ID to dictionary as concernId
            APPEND dictionary to issues list

        RETURN success response with issues list and status 200

    CATCH Exception as e
        RETURN error response with exception message and status 500
```

*Algorithm:*

1. Import Libraries
2. Import necessary libraries: functions\_framework, google.cloud.firestore, flask.jsonify, and flask\_cors.cross\_origin.
3. Initialize Firestore Client
4. Initialize a Firestore client.
5. Define ChatRetriever Function
6. Allow Cross-Origin Requests
  - a. Decorate the function with @cross\_origin() to allow cross-origin requests.
7. Parse Request Data
  - a. Parse the JSON body and arguments from the request.
  - b. Initialize customer\_email and agent\_email to None.
  - c. If the request JSON body exists:
    - i. Set customer\_email from the request JSON.
    - ii. Set agent\_email from the request JSON.
  - d. Else if the request arguments exist:
    - i. Set customer\_email from the request arguments.
    - ii. Set agent\_email from the request arguments.
8. Validate Emails
  - a. If both customer\_email and agent\_email are None:
    - i. Return an error response with message "customerEmail or agentEmail is required!" and status 400.
9. Try to Retrieve Chats
  - a. Initialize a reference to the Firestore 'Issues' collection.
  - b. If customer\_email exists:
    - i. Create a query to filter issues by customer\_email.
  - c. Else:
    - i. Create a query to filter issues by agent\_email.
  - d. Execute the query and get the documents.
  - e. Initialize an empty issues list.
  - f. For each document in the query result:
    - i. Convert the document to a dictionary.
    - ii. Add the document ID to the dictionary as concernId.
    - iii. Append the dictionary to the issues list.
  - g. Return a success response with the issues list and status 200.
10. Handle Errors
  - a. Catch exceptions.
  - b. Return an error response with the exception message and status 500.

## Module 5: Data Analysis and Visualization

### Lambda Function: Post Feedback

*Pseudocode:*

```
FUNCTION lambda_handler(event, context):

    DEFINE apiUrl as 'https://us-central1-dalvacationhome-
        429314.cloudfunctions.net/extract-sentiment'

    INITIALIZE body to undefined
    INITIALIZE statusCode to 200
    INITIALIZE headers with "Content-Type" set to "application/json"

    TRY:
        IF event.body is a string:
            PARSE event.body to JSON and assign to requestJSON
        ELSE:
            ASSIGN event.body directly to requestJSON

        DEFINE payload with text set to requestJSON.feedbackText

        DEFINE response as result of POST request to apiUrl with payload

        IF response is not OK:
            THROW Error with message containing response status

        PARSE response JSON to data

        DEFINE feedback with properties from requestJSON and data (including
        current date)

        DEFINE result as result of UpdateCommand to roomsTable to append
        feedback

        DEFINE resultUpdateFeedbackStatus as result of UpdateCommand to
        bookingsTable to set feedbackReceived to true
        SET body to success message with roomid from requestJSON

        CATCH error:
            SET statusCode to 400
            SET body to error message

        FINALLY:
            CONVERT body to JSON string
            RETURN response with statusCode, body, and headers

    END FUNCTION lambda_handler
```

*Algorithm:*

1. Initialize Variables
  - a. Define apiUrl for the sentiment analysis service.
  - b. Initialize body to undefined.
  - c. Set statusCode to 200.
  - d. Define response headers with content type as JSON.
2. Parse Event Body
  - a. If event.body is a string, parse it to JSON and assign it to requestJSON.
  - b. Otherwise, assign event.body directly to requestJSON.
3. Prepare Payload for Sentiment Analysis
  - a. Create a payload object with text set to requestJSON.feedbackText.
4. Call Sentiment Analysis API
  - a. Send a POST request to apiUrl with the payload.
  - b. If the response is not OK, throw an error with the response status.
5. Parse API Response
  - a. Parse the JSON response from the API to data.
6. Create Feedback Object
  - a. Create a feedback object with properties from requestJSON and data, including the current date.
7. Update DynamoDB Tables
  - a. Update Rooms Table:
    - i. Append the feedback object to the feedback list in the roomsTable.
  - b. Update Bookings Table:
    - i. Mark the feedback as received in the bookingsTable.
8. Set Response Body
  - a. Set the response body to a success message using the roomid from requestJSON.
9. Handle Errors
  - a. In the catch block:
    - i. Set statusCode to 400.
    - ii. Set the response body to the error message.
10. Convert Response Body
  - a. Convert the response body to a JSON string.
11. Return Response
  - a. Return a JSON response with statusCode, body, and headers.

## Cloud Function: Extract Sentiment

### Pseudocode:

```
FUNCTION helloHttp(req, res):
    EXTRACT text from req.body
    INITIALIZE client as new instance of language.LanguageServiceClient
    DEFINE document with content set to text and type set to 'PLAIN_TEXT'

    TRY:
        CALL analyzeSentiment method on client with document
        ASSIGN result to the first element of the response array
        EXTRACT sentiment from result.documentSentiment
        DEFINE response object with text, sentimentScore, and
        sentimentMagnitude from sentiment
        RETURN response with 200 status code and JSON body
    CATCH error:
        LOG error
        RETURN error message with 500 status code
    END FUNCTION helloHttp
```

### Algorithm:

1. Parse Request Body
  - a. Extract text from req.body.
2. Initialize Language Service Client
  - a. Initialize a new instance of language.LanguageServiceClient.
3. Prepare Document for Sentiment Analysis
  - a. Define a document object with:
    - i. content set to the extracted text.
    - ii. type set to 'PLAIN\_TEXT'.
4. Analyze Sentiment
  - a. Use a try-catch block to handle potential errors:
    - i. Try Block:
      1. Call the analyzeSentiment method on the client with the document.
      2. Assign result to the first element of the response array.
      3. Extract sentiment from result.documentSentiment.
      4. Define a response object with:
        - a. text set to the original text.
        - b. sentimentScore set to sentiment.score.
        - c. sentimentMagnitude set to sentiment.magnitude.
      5. Return the response with a 200 status code and JSON body.
    - ii. Catch Block:
      1. Log the error.
      2. Return an error message with a 500 status code.
  5. Return Response

- a. Based on the outcome of the sentiment analysis, return the appropriate HTTP status code and message:
  - i. 200 with the sentiment analysis result if successful.
  - ii. 500 with an error message if there is an issue with the sentiment analysis.

## Lambda Function: post-authentication-trigger

### Pseudocode:

```

FUNCTION handler(event, context):
    INITIALIZE dynamoDBClient, dynamoDocument, snsClient

    EXTRACT email, name, roleCode from event.request.userAttributes

    DETERMINE role based on roleCode
    GET current date and time

    DEFINE params for updating DynamoDB table LoginStatistics

    TRY:
        SEND UpdateCommand to dynamoDocument with params
        LOG success message

        CREATE notification message
        DEFINE snsParams with TopicArn and message

        SEND PublishCommand to snsClient with snsParams
        LOG success message

    CATCH error:
        LOG error message

    RETURN event
END FUNCTION
  
```

### Algorithm:

1. Initialize Variables:
  - a. Initialize dynamoDBClient, dynamoDocument, and snsClient.
2. Extract User Information:
  - a. Extract email, name, and roleCode from event.request.userAttributes.
3. Determine User Role:
  - a. Determine role based on roleCode.
4. Get Current Date and Time:
  - a. Get the current date and time.
5. Define DynamoDB Update Parameters:
  - a. Define params for updating the LoginStatistics table in DynamoDB.
6. Try Block:

- a. Update DynamoDB:
  - i. Send UpdateCommand to dynamoDocument with params.
  - ii. Log success message.
- b. Send Notification:
  - i. Create notification message.
  - ii. Define snsParams with TopicArn and message.
  - iii. Send PublishCommand to snsClient with snsParams.
  - iv. Log success message.
- 7. Catch Block:
  - a. Log error message if an exception occurs.
- 8. Return Event:
  - a. Return the event object.

## Lambda Function: storeLoginStatistics

*Pseudocode:*

```

FUNCTION handler(event, context):
    INITIALIZE client, dynamo
    INITIALIZE body

    IF event.body is a string:
        PARSE event.body to JSON and assign to body
    ELSE:
        ASSIGN event.body to body

    EXTRACT email, date, time, action from body

    DEFINE params for updating DynamoDB table LoginStatistics

    TRY:
        SEND UpdateCommand to dynamo with params
        RETURN success response with result attributes
    CATCH error:
        LOG error message
        RETURN error response
END FUNCTION
  
```

*Algorithm:*

1. Initialize Variables:
  - a. Initialize client and dynamo.
2. Parse Event Body:
  - a. If event.body is a string, parse it to JSON and assign to body.
  - b. Otherwise, assign event.body to body.
3. Extract Parameters:
  - Extract email, date, time, and action from body.

4. Define DynamoDB Update Parameters:
  - a. Define params for updating the LoginStatistics table in DynamoDB.
5. Try Block:
  - a. Update DynamoDB:
    - i. Send UpdateCommand to dynamo with params.
    - ii. Return success response with result attributes.
6. Catch Block:
  - a. Log error message if an exception occurs.
  - b. Return error response.

## Lambda Function: fetchUserData

*Pseudocode:*

```

FUNCTION handler(event, context):
    INITIALIZE client, dynamo

    TRY:
        DEFINE usersParams for scanning Users table
        SCAN Users table and assign result to users

        DEFINE loginStatsParams for scanning LoginStatistics table
        SCAN LoginStatistics table and assign result to loginStats

        RETURN success response with users and loginStats
    CATCH error:
        LOG error message
        RETURN error response
    END FUNCTION
  
```

*Algorithm:*

1. Initialize Variables:
  - a. Initialize client and dynamo.
2. Try Block:
  - a. Fetch Users:
    - i. Define usersParams for scanning the Users table.
    - ii. Scan Users table and assign the result to users.
  - b. Fetch Login Statistics:
    - i. Define loginStatsParams for scanning the LoginStatistics table.
    - ii. Scan LoginStatistics table and assign the result to loginStats.
  - c. Return Success Response:
    - i. Return success response with users and loginStats.
3. Catch Block:
  - a. Log error message if an exception occurs.
  - b. Return error response.

# Test Cases:

## Lambda Test Post Feedback Event:

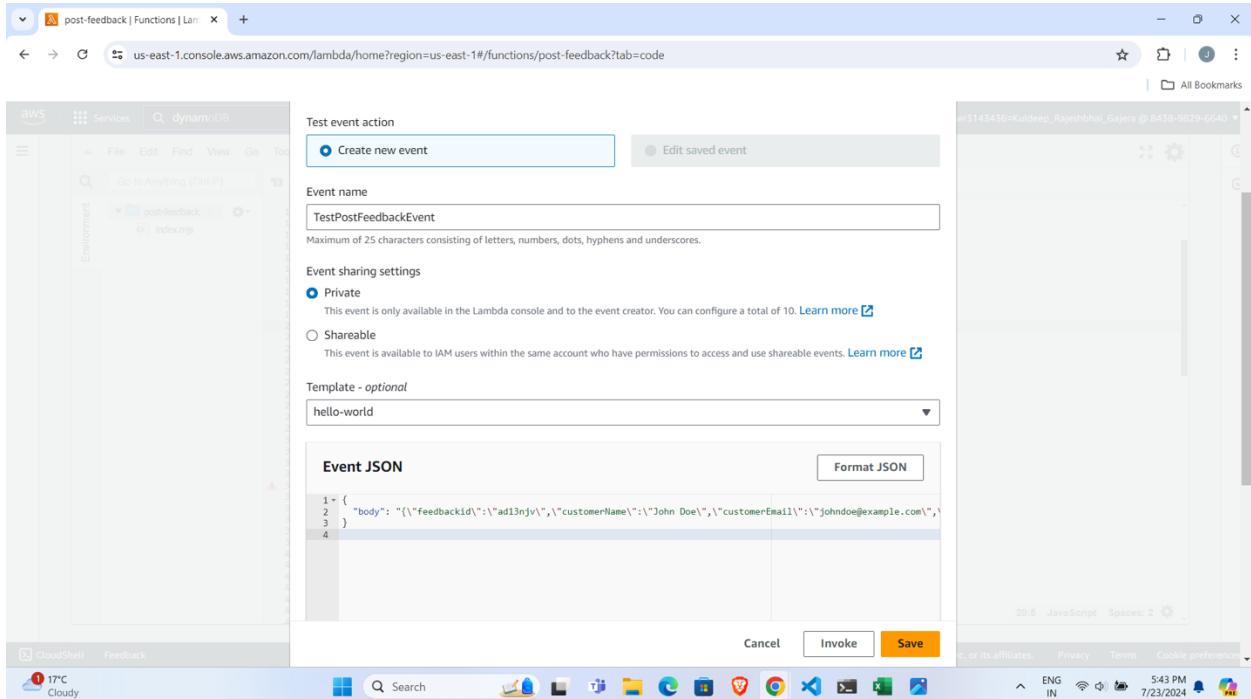


Figure 8: Setup and execution of the Post Feedback test event in AWS Lambda.

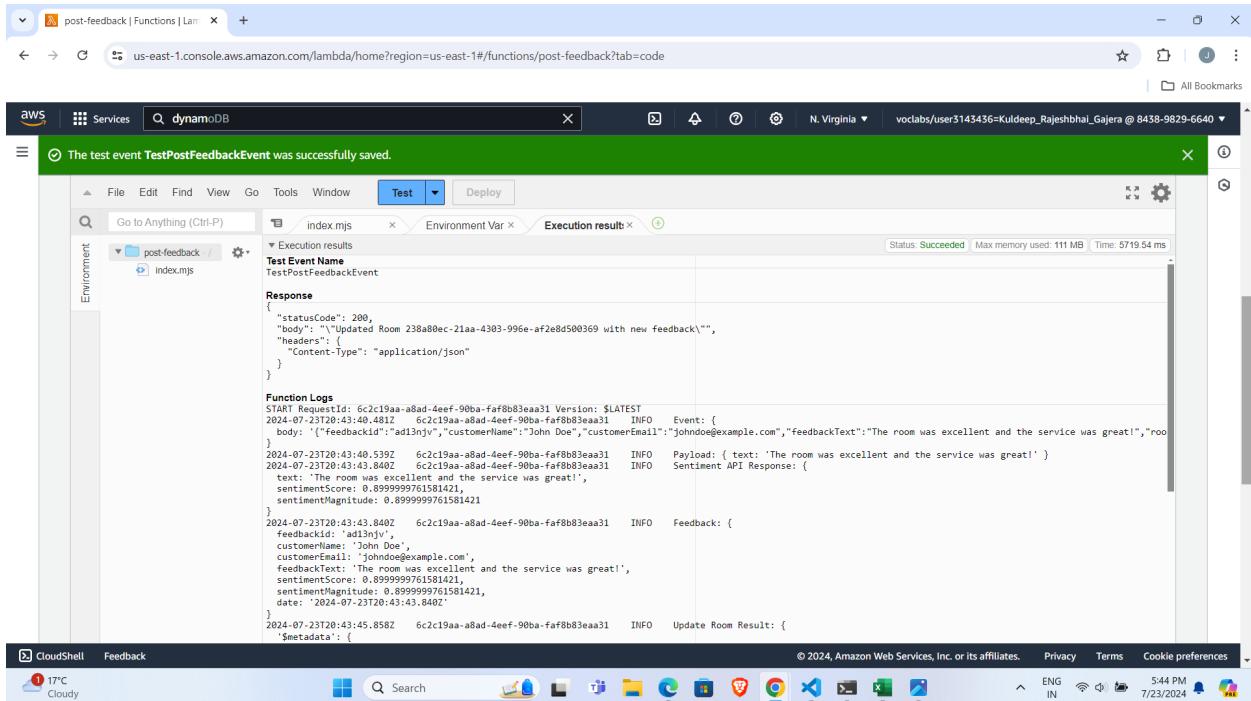


Figure 9: CloudWatch log showing the results of the Post Feedback test event execution.

## Cloud Function Extract Sentiment Test Event:

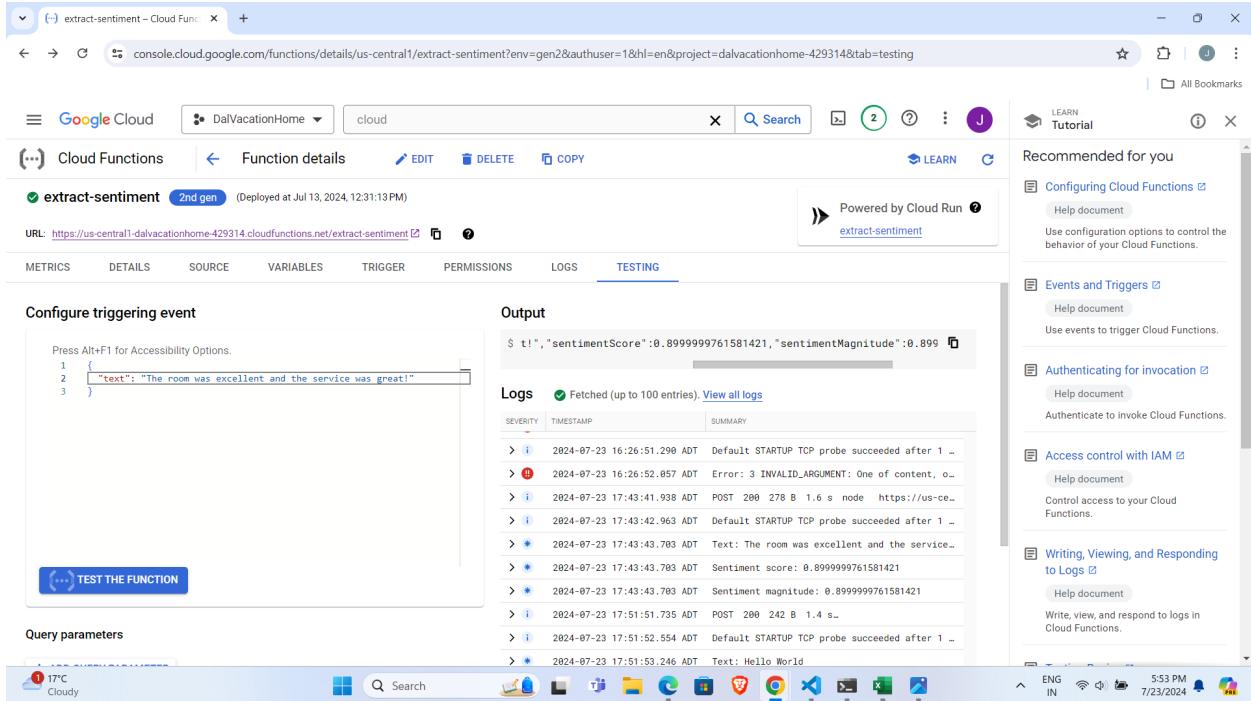


Figure 10: Configuring and executing Extract Sentiment Cloud Function test event in GCP Console.

## Lambda Test Post Authentication Logging Event:

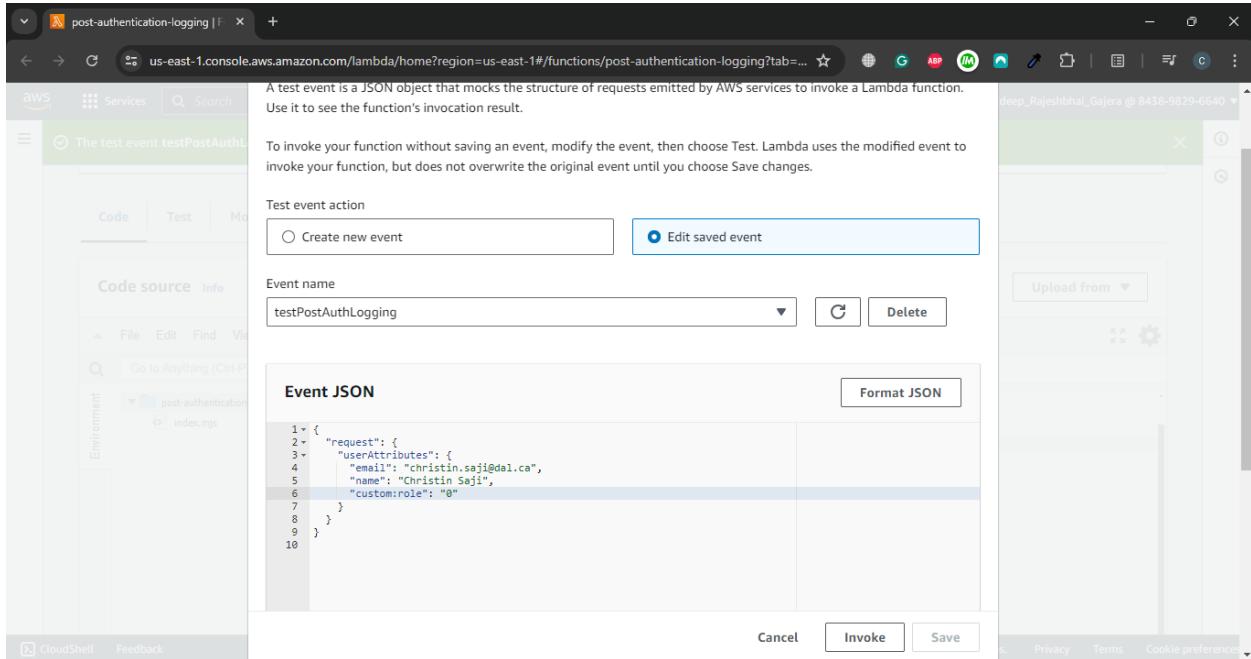


Figure 11: Setup and execution of the Post Authentication Logging test event in AWS Lambda.

```

{
  "request": {
    "userAttributes": {
      "email": "christin.saji@dai.ca",
      "name": "Christin Saji",
      "custom:role": "0"
    }
  }
}

Function Logs
START RequestId: 1e26f0bc-845c-49d4-afcc-763743bc345c Version: $LATEST
2024-07-23T21:23:03.298Z 1e26f0bc-845c-49d4-afcc-763743bc345c INFO in lambda function {
  request: {
    userAttributes: {
      email: 'christin.saji@dai.ca',
      name: 'Christin Saji',
      'custom:role': '0'
    }
  }
}
2024-07-23T21:23:03.321Z 1e26f0bc-845c-49d4-afcc-763743bc345c INFO email name, roleCode christin.saji@dai.ca Christin Saji 0
2024-07-23T21:23:04.279Z 1e26f0bc-845c-49d4-afcc-763743bc345c INFO Login statistics stored successfully
2024-07-23T21:23:04.821Z 1e26f0bc-845c-49d4-afcc-763743bc345c INFO Notification sent successfully
END RequestId: 1e26f0bc-845c-49d4-afcc-763743bc345c
REPORT RequestId: 1e26f0bc-845c-49d4-afcc-763743bc345c Duration: 1562.55 ms Billed Duration: 1563 ms Memory Size: 128 MB Max Memory Used: 93 MB
Request ID
1e26f0bc-845c-49d4-afcc-763743bc345c

```

Figure 12: CloudWatch log showing the results of the Post Authentication Logging test event execution.

## Lambda Test Store Login Statistics Event:

```

[{"body": {"email": "christin.saji@dai.ca", "date": "2024-07-23", "time": "12:00:00", "action": "Login"}]

```

Figure 13: Setup and execution of the Store Login Statistics test event in AWS Lambda.

The screenshot shows the AWS Lambda console interface. The top navigation bar includes tabs for Code, Test, Monitor, Configuration, Aliases, and Versions. The main area displays the 'Code source' tab with the file 'index.mjs'. Below the code editor is a 'Test' button, which is highlighted in blue. To the right of the code editor, there is a 'Execution result' section. This section shows the execution details: Status Succeeded, Max memory used: 91 MB, and Time: 991.35 ms. The log output is as follows:

```

Execution results
Test Event Name
TestStoreLoginStatistics

Response
{
  "statusCode": 200,
  "headers": {
    "Access-Control-Allow-Headers": "Content-Type",
    "Access-Control-Allow-Origin": "*",
    "Access-Control-Allow-Methods": "OPTIONS,POST,GET"
  },
  "body": "[{"loghistory": [{"date": "2024-07-23", "action": "registration", "time": "12:48:50"}, {"date": "2024-07-23", "action": "login", "time": "12:48:50"}]}]"
}

Function Logs
START RequestId: 3529b56c-34ad-4625-86a1-39b03c829c27 Version: $LATEST
END RequestId: 3529b56c-34ad-4625-86a1-39b03c829c27
REPORT RequestId: 3529b56c-34ad-4625-86a1-39b03c829c27 Duration: 991.35 ms Billed Duration: 992 ms Memory Size: 128 MB Max Memory Used: 91 MB

Request ID
3529b56c-34ad-4625-86a1-39b03c829c27

```

Figure 14: CloudWatch log showing the results of the Store Login Statistics test event execution.

## Lambda Test Fetch User Data Event:

The screenshot shows the AWS Lambda console interface. The top navigation bar includes tabs for Code, Test, Monitor, Configuration, Aliases, and Versions. The main area displays the 'Code source' tab with the file 'index.js'. Below the code editor is a 'Test' button, which is highlighted in blue. To the right of the code editor, there is a 'Configure test event' dialog box. The dialog box has the following fields:

- Test event action:** An input field with two options:  Create new event and  Edit saved event.
- Event name:** A dropdown menu with the value 'TestFetchUserData'.
- Event JSON:** A text area containing the JSON value '1 {}'.

At the bottom of the dialog box are buttons for 'Cancel', 'Invoke', and 'Save'.

Figure 15: Setup and execution of the Fetch User Data test event in AWS Lambda.

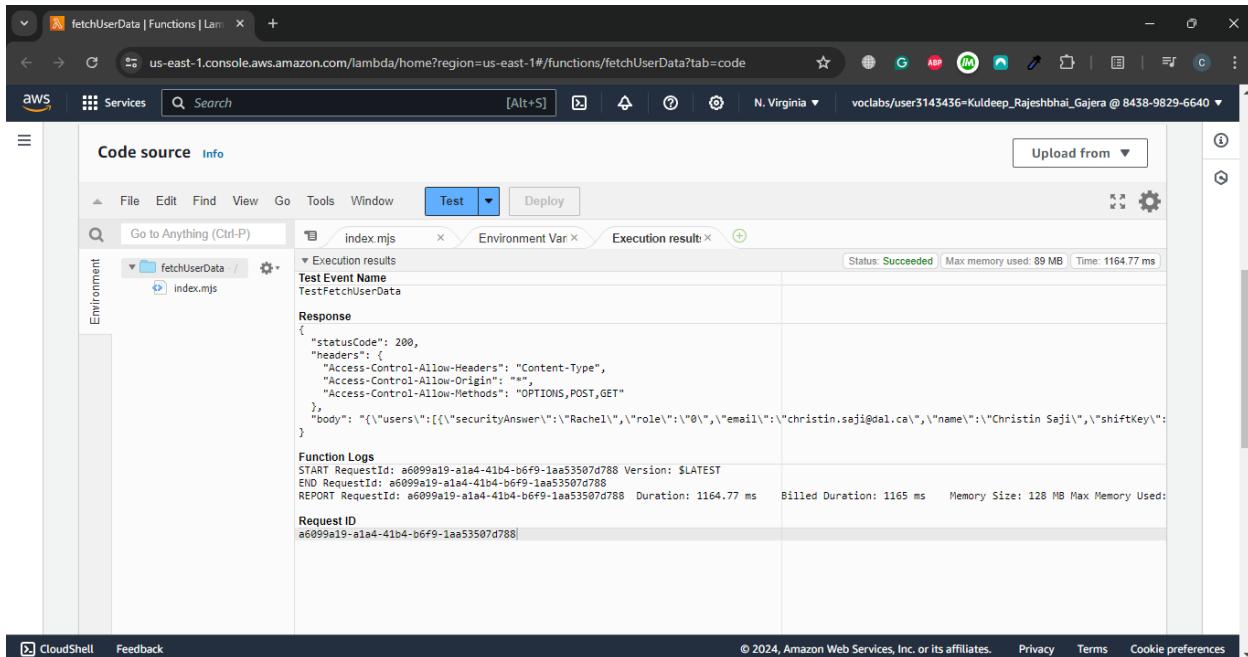


Figure 16: CloudWatch log showing the results of the Fetch User Data test event execution.

## HandoffCustomerSupport

The screenshot shows the AWS Lambda console interface. On the left, under 'Test event', there's a 'Create new event' button and a dropdown for 'Event name' set to 'SaveCustomerRequestTest'. Below that is the 'Event JSON' section with the following content:

```

1: {
2:   "Issue": "Raise service request",
3:   "BookingReferenceCode": "B123456"
4: }

```

On the right, the 'Executing function: succeeded' section shows the execution details:

- Code SHA-256: A24PHDmObt5LXfRwvTfQUtQUNvLSDsgufHPfBdA+
- Request ID: 39063f14779-41b8-664a-b42d8b008db
- Init duration: 1143.76 ms
- Billed duration: 3606 ms
- Max memory used: 115 MB
- Execution time: 1 minute ago (July 23, 2024 at 06:54 PM AEST)
- Function version: \$LATEST
- Duration: 3679.26 ms
- Resources configured: 128 MB

The 'Log output' section contains the following log entries:

```

1: [2024-07-23T22:54:13.652Z] 98b7c1-479-41b8-664a-b42d8b008db Version: $LATEST
2: [2024-07-23T22:54:13.652Z] 98b7c1-479-41b8-664a-b42d8b008db 202 Triggered by Interpolator
3: [2024-07-23T22:54:13.652Z] 98b7c1-479-41b8-664a-b42d8b008db 202 Event received: New customer request
4: [2024-07-23T22:54:13.652Z] 98b7c1-479-41b8-664a-b42d8b008db 202 Function code received: B123456

```

Figure 17: Executing the HandoffCustomerSupport test with corresponding CloudWatch log outcomes.

## AgentAssigner:

The screenshot shows the Google Cloud Functions console for the 'AgentAssigner' function. The 'Logs' tab is active, showing a table of log entries. The columns include Severity, Timestamp, and Log. The log entries detail interactions with Firestore, such as writing data to Firestore successfully and reading documents like 'Customer Email: jeetjani@dal.ca'. Other entries show API requests and system logs related to Cloud Functions and Cloud Run.

Figure 18: Setting up and executing the IntentHandler test, with results shown in the CloudWatch logs.

## AdminFetcher:

The screenshot shows the Google Cloud Functions console for the 'AdminFetcher' function. The execution details page is displayed, showing a successful execution. The log output section shows a JSON response with three security answers. The summary section provides execution time (32 seconds ago), function version (\$LATEST), duration (852.00 ms), and resources configured (128 MB).

Figure 19: Running the FetchBookingDetails test and displaying execution logs in CloudWatch.

# ChatLogger

ChatLogger 2nd gen (Deployed at Jul 20, 2024, 11:16:01 PM) URL: <https://us-central1-thematic-answer-427612-gcf.cloudfunctions.net/ChatLogger>

Powered by Cloud Run  
chatlogger

METRICS	DETAILS	SOURCE	VARIABLES	TRIGGER	PERMISSIONS	LOGS	TESTING																																													
<h3>Configure triggering event</h3> <p>Press Option+F1 for Accessibility Options.</p> <pre>1  { 2     "concernId": "XsvDrPcSJcdkInuQW9t2", 3     "concernText": "Leaky faucet in the kitchen", 4     "customerName": "Alice", 5     "customerEmail": "alice@example.com", 6     "agentName": "John Doe", 7     "agentEmail": "john.doe@example.com", 8     "dateRaised": "2023-07-16", 9     "isActive": true, 10    "chats": [ 11      { 12        "from": "agent", 13        "message": "We will fix the leaky faucet tomorrow.", 14        "timestamp": "2023-07-16T10:00:00Z" 15      } 16    ] }</pre> <p><b>TEST THE FUNCTION</b></p> <h3>Query parameters</h3> <p>+ ADD QUERY PARAMETER</p> <h3>Headers</h3> <p>+ ADD HEADER</p> <h3>Output</h3> <p>Logs Fetched (up to 100 entries); <a href="#">View all logs</a></p> <table border="1"><thead><tr><th>SEVERITY</th><th>TIMESTAMP</th><th>SUMMARY</th></tr></thead><tbody><tr><td>&gt;</td><td>2024-07-28 23:34:09.373 ADT</td><td>POST 200 239 B 113 ms Chrome 126 https://us-central1-th...</td></tr><tr><td>&gt;</td><td>2024-07-28 23:43:43.109 ADT</td><td>OPTIONS 200 298 B 5 ms Chrome 126 https://us-central1-th...</td></tr><tr><td>&gt;</td><td>2024-07-28 23:43:43.185 ADT</td><td>POST 200 239 B 131 ms Chrome 126 https://us-central1-th...</td></tr><tr><td>&gt;</td><td>2024-07-22 19:58:59.571 ADT</td><td>OPTIONS 200 298 B 2.4 s Chrome 126 https://us-central1-th...</td></tr><tr><td>&gt;</td><td>2024-07-22 19:59:02.681 ADT</td><td>Default STARTUP TCP probe succeeded after 1 attempt for con...</td></tr><tr><td>&gt;</td><td>2024-07-22 19:59:02.733 ADT</td><td>POST 200 239 B 195 ms Chrome 126 https://us-central1-th...</td></tr><tr><td>*</td><td>2024-07-22 19:59:02.744 ADT</td><td>WARNING: All log messages before abs1::InitializeLog() are c...</td></tr><tr><td>*</td><td>2024-07-22 19:59:02.744 ADT</td><td>100000 00:00:1721689142.744383 6 config.cc:[230] gRPC e...</td></tr><tr><td>*</td><td>2024-07-22 19:59:02.764 ADT</td><td>100000 00:00:1721689142.765288 6 check_gcp_environment...</td></tr><tr><td>&gt;</td><td>2024-07-22 28:00:03.729 ADT</td><td>OPTIONS 200 339 B 2 ms Chrome 126 https://us-central1-th...</td></tr><tr><td>&gt;</td><td>2024-07-22 28:00:03.887 ADT</td><td>POST 200 271 B 184 ms Chrome 126 https://us-central1-th...</td></tr><tr><td>&gt;</td><td>2024-07-23 18:57:00.847 ADT</td><td>POST 400 280 B 1.9 s Google-Cloud-Functions https://us...</td></tr><tr><td>&gt;</td><td>2024-07-23 18:57:02.862 ADT</td><td>Default STARTUP TCP probe succeeded after 1 attempt for con...</td></tr><tr><td>&gt;</td><td>2024-07-23 19:00:05.069 ADT</td><td>POST 400 280 B 2 ms Google-Cloud-Functions https://us...</td></tr></tbody></table>								SEVERITY	TIMESTAMP	SUMMARY	>	2024-07-28 23:34:09.373 ADT	POST 200 239 B 113 ms Chrome 126 https://us-central1-th...	>	2024-07-28 23:43:43.109 ADT	OPTIONS 200 298 B 5 ms Chrome 126 https://us-central1-th...	>	2024-07-28 23:43:43.185 ADT	POST 200 239 B 131 ms Chrome 126 https://us-central1-th...	>	2024-07-22 19:58:59.571 ADT	OPTIONS 200 298 B 2.4 s Chrome 126 https://us-central1-th...	>	2024-07-22 19:59:02.681 ADT	Default STARTUP TCP probe succeeded after 1 attempt for con...	>	2024-07-22 19:59:02.733 ADT	POST 200 239 B 195 ms Chrome 126 https://us-central1-th...	*	2024-07-22 19:59:02.744 ADT	WARNING: All log messages before abs1::InitializeLog() are c...	*	2024-07-22 19:59:02.744 ADT	100000 00:00:1721689142.744383 6 config.cc:[230] gRPC e...	*	2024-07-22 19:59:02.764 ADT	100000 00:00:1721689142.765288 6 check_gcp_environment...	>	2024-07-22 28:00:03.729 ADT	OPTIONS 200 339 B 2 ms Chrome 126 https://us-central1-th...	>	2024-07-22 28:00:03.887 ADT	POST 200 271 B 184 ms Chrome 126 https://us-central1-th...	>	2024-07-23 18:57:00.847 ADT	POST 400 280 B 1.9 s Google-Cloud-Functions https://us...	>	2024-07-23 18:57:02.862 ADT	Default STARTUP TCP probe succeeded after 1 attempt for con...	>	2024-07-23 19:00:05.069 ADT	POST 400 280 B 2 ms Google-Cloud-Functions https://us...
SEVERITY	TIMESTAMP	SUMMARY																																																		
>	2024-07-28 23:34:09.373 ADT	POST 200 239 B 113 ms Chrome 126 https://us-central1-th...																																																		
>	2024-07-28 23:43:43.109 ADT	OPTIONS 200 298 B 5 ms Chrome 126 https://us-central1-th...																																																		
>	2024-07-28 23:43:43.185 ADT	POST 200 239 B 131 ms Chrome 126 https://us-central1-th...																																																		
>	2024-07-22 19:58:59.571 ADT	OPTIONS 200 298 B 2.4 s Chrome 126 https://us-central1-th...																																																		
>	2024-07-22 19:59:02.681 ADT	Default STARTUP TCP probe succeeded after 1 attempt for con...																																																		
>	2024-07-22 19:59:02.733 ADT	POST 200 239 B 195 ms Chrome 126 https://us-central1-th...																																																		
*	2024-07-22 19:59:02.744 ADT	WARNING: All log messages before abs1::InitializeLog() are c...																																																		
*	2024-07-22 19:59:02.744 ADT	100000 00:00:1721689142.744383 6 config.cc:[230] gRPC e...																																																		
*	2024-07-22 19:59:02.764 ADT	100000 00:00:1721689142.765288 6 check_gcp_environment...																																																		
>	2024-07-22 28:00:03.729 ADT	OPTIONS 200 339 B 2 ms Chrome 126 https://us-central1-th...																																																		
>	2024-07-22 28:00:03.887 ADT	POST 200 271 B 184 ms Chrome 126 https://us-central1-th...																																																		
>	2024-07-23 18:57:00.847 ADT	POST 400 280 B 1.9 s Google-Cloud-Functions https://us...																																																		
>	2024-07-23 18:57:02.862 ADT	Default STARTUP TCP probe succeeded after 1 attempt for con...																																																		
>	2024-07-23 19:00:05.069 ADT	POST 400 280 B 2 ms Google-Cloud-Functions https://us...																																																		

*Figure 20: Running the FetchBookingDetails test and displaying execution logs in CloudWatch.*

# ChatRetriever

Figure 21: Running the FetchBookingDetails test and displaying execution logs in CloudWatch.

# Notification testing using SNS publish topic

Amazon SNS > Topics > userNotification > Publish message

## Publish message to topic

**Message details**

Topic ARN  
arn:aws:sns:us-east-1:843898296640:userNotification

Subject - optional  
test  
Maximum 100 printable ASCII characters

Time to Live (TTL) - optional Info  
This setting applies only to mobile application endpoints. The number of seconds that the push notification service has to deliver the message to the endpoint.

**Message body**

Message structure  
 Identical payload for all delivery protocols.  
The same payload is sent to endpoints subscribed to the topic, regardless of their delivery protocol.  
 Custom payload for each delivery protocol.  
Different payloads are sent to endpoints subscribed to the topic, based on their delivery protocol.

Message body to send to the endpoint

```
1 {
2   "to": "kuldeepgajera15@gmail.com",
3   "subject": "Test Email",
4   "body": "This is a test email from SQS"
5 }
```

Figure 22: Running the FetchBookingDetails test and displaying execution logs in CloudWatch.

CloudWatch

Favorites and recents

Dashboards

Alarms △ 10 ○ 30 ○ 0

In alarm

All alarms

Billing

Logs

Log groups

Log Anomalies

Live Tail

Logs Insights

Contributor Insights

Metrics

X-Ray traces

Events

Application Signals New

Network monitoring

Insights

2024-07-24T00:50:04.700Z 2024-07-24T00:50:04.700Z c72b8ae-b-fca1-59ad-b06e-d5f1af4d46ce INFO in lambda event { Records: [ { messageId:...

START RequestId: c72b8ae-b-fca1-59ad-b06e-d5f1af4d46ce Version: \$LATEST

2024-07-24T00:50:04.701Z 2024-07-24T00:50:04.701Z c72b8ae-b-fca1-59ad-b06e-d5f1af4d46ce INFO after snsMessage { "to": "kuldeepgajera15@gmail.com", "subject": "Test Email", "body": "This is a test email from SQS" }

2024-07-24T00:50:04.701Z 2024-07-24T00:50:04.701Z c72b8ae-b-fca1-59ad-b06e-d5f1af4d46ce INFO msg: { to: 'kuldeepgajera15@gmail.com', ... }

2024-07-24T00:50:06.363Z 2024-07-24T00:50:06.363Z c72b8ae-b-fca1-59ad-b06e-d5f1af4d46ce INFO Email sent: 250 2.0.0 OK 1721782206 6a18...

6a1893d4083f44-6b70c76370sm52810716d6:52 - smtp

2024-07-24T00:50:06.375Z END RequestId: c72b8ae-b-fca1-59ad-b06e-d5f1af4d46ce

END RequestId: c72b8ae-b-fca1-59ad-b06e-d5f1af4d46ce Duration: 1674.97 ms Billed Duration: 1674.97 ms

REPORT RequestId: c72b8ae-b-fca1-59ad-b06e-d5f1af4d46ce Duration: 1674.97 ms Billed Duration: 1674.97 ms

Figure 23: Running the FetchBookingDetails test and displaying execution logs in CloudWatch.

# API Testing:

## Post Feedback:

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'Aws Auction Service', 'AWS serverless-lab', 'iMessage clone', 'kubernetes', 'lcodev', 'Mens100m', 'MernEstore', 'student registration', and 'vatsal'. The main area displays a POST request to the URL <https://fa7721ywbk.execute-api.us-east-1.amazonaws.com/feedback>. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2   "feedbackId": "ahdjjjd12",
3   "customerName": "jay",
4   "customerEmail": "jayate145677@gmail.com",
5   "feedbackText": "The room was comfortable and the amenities were satisfactory.",
6   "roomid": "4212473f-ec38-402e-879a-6b20a08311ec",
7   "bookingid": "BHI97994"
8 }
```

The response section shows a status of 200 OK with a message: "Updated Room 4212473f-ec38-402e-879a-6b20a08311ec with new feedback". The bottom of the window shows the Windows taskbar with various pinned icons.

Figure 24: API call execution to Post Feedback for a specific booking of Room, highlighting the request and response.

## Extract Sentiment:

The screenshot shows the Postman interface for an API call to extract sentiment. The URL is `https://us-central1-dalvacationhome-429314.cloudfunctions.net/extract-sentiment`. The method is POST. The request body is JSON, containing the text "Very less satisfied with room, but staff was very helpful". The response status is 200 OK, with a sentiment score of -0.4000000059604645 and a magnitude of 0.4000000059604645.

Figure 25: API interaction for extracting sentiment (score and magnitude) of feedback text, displaying request and response details.

## Store Login Statistics:

The screenshot shows the Postman interface for an API call to store login statistics. The URL is `https://audigx4717.execute-api.us-east-1.amazonaws.com/dev/loginEvent`. The method is POST. The request body is JSON, containing user login information: email (christin.saji@dal.ca), date (2023-07-23), time (11:08:00), and action (login). The response status is 200 OK, indicating successful logging.

Figure 26: API call to storeLoginStatistics lambda to store logging events of the user

## Fetch User Data:

The screenshot shows the Postman interface with a dark theme. On the left, the sidebar displays 'My Workspace' with collections like 'Cloud Computing Assignment 3' and 'Cloud Computing Kubernetes'. Under 'Serverless Project', there are two items: 'Web Development Tutorial 5' and 'Web Development Tutorial 7'. The main area shows a POST request to 'https://auidgx4717.execute-api.us-east-1.amazonaws.com/dev/userStats'. The response is a GET request to the same URL. The response body is a JSON array of user objects:

```
1 "users": [
2   {
3     "securityAnswer": "Rachel",
4     "role": "0",
5     "email": "chxistin.saji@dal.ca",
6     "name": "Christin Saji",
7     "shiftKey": "2"
8   },
9   {
10    "securityAnswer": "Blue",
11    "role": "1",
12    "email": "blindbasics@gmail.com",
13    "name": "Kuldeep Gajera",
14    "shiftKey": "3"
15  },
16  {
17    "securityAnswer": "holla",
18    "role": "0",
19    "email": "jaypatel45677@gmail.com",
20    "name": "jay",
21    "shiftKey": "3"
22  }
23 ]
```

Figure 27: API call to `fetchUserData` lambda to fetch the user details and the logging events of the user

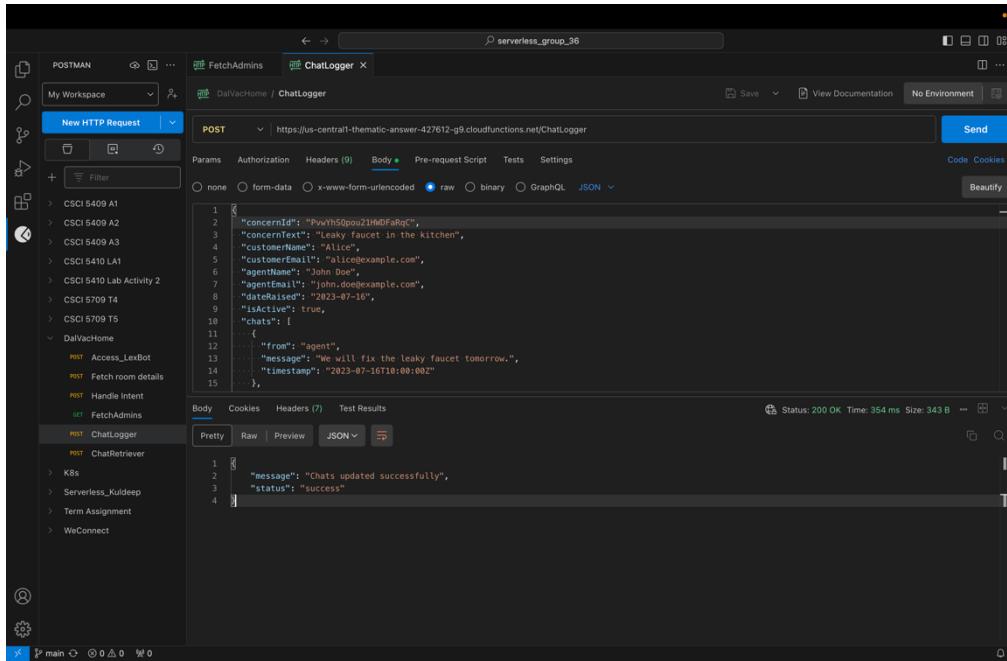
## Fetch Admins:

The screenshot shows the Postman interface with a dark theme. The sidebar lists various projects and environments. The main area shows a GET request to 'https://bnnk80cuma.execute-api.us-east-1.amazonaws.com/v1/fetchAdmins'. The response body is a JSON array of admin objects:

```
1 "users": [
2   {
3     "securityAnswer": "Blue",
4     "role": "1",
5     "email": "blindbasics@gmail.com",
6     "name": "Kuldeep Gajera",
7     "shiftKey": "3"
8   },
9   {
10    "securityAnswer": "Rachel",
11    "role": "1",
12    "email": "christinsaji01@gmail.com",
13    "name": "Christin Saji",
14    "shiftKey": "2"
15  },
16  {
17    "securityAnswer": "holla",
18    "role": "1",
19    "email": "dalvacationhome36@gmail.com",
20    "name": "dal property agent",
21    "shiftKey": "2"
22  },
23  {
24    "securityAnswer": "holla",
25    "role": "1",
26    "email": "jaypatel08215@gmail.com",
27    "name": "jay",
28    "shiftKey": "2"
29  }
30 ]
```

Figure 28: API call to `fetchUserData` lambda to fetch the user details and the logging events of the user

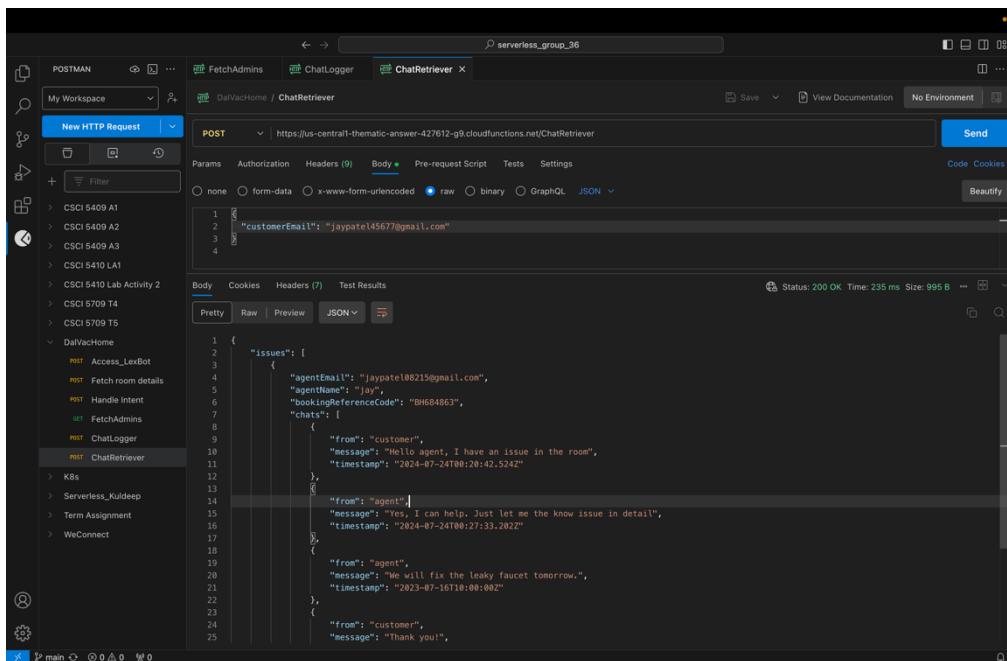
## Log Chats:



The screenshot shows the Postman application interface. On the left, there's a sidebar with a tree view of various API endpoints under categories like 'My Workspace', 'CSCI 5409 A1' through 'CSCI 5709 T5', and 'DalVachome'. The main panel shows a request to 'ChatLogger' at 'https://us-central1-thematic-answer-427612-g9.cloudfunctions.net/ChatLogger'. The request method is POST, and the URL is https://us-central1-thematic-answer-427612-g9.cloudfunctions.net/ChatLogger. The body is set to raw JSON, containing a message about a leaky faucet. The response status is 200 OK, with a timestamp of 2023-07-16T10:00:00Z and a size of 343B.

Figure 29: API call to fetchUserData lambda to fetch the user details and the logging events of the user

## Retrieve Chats:

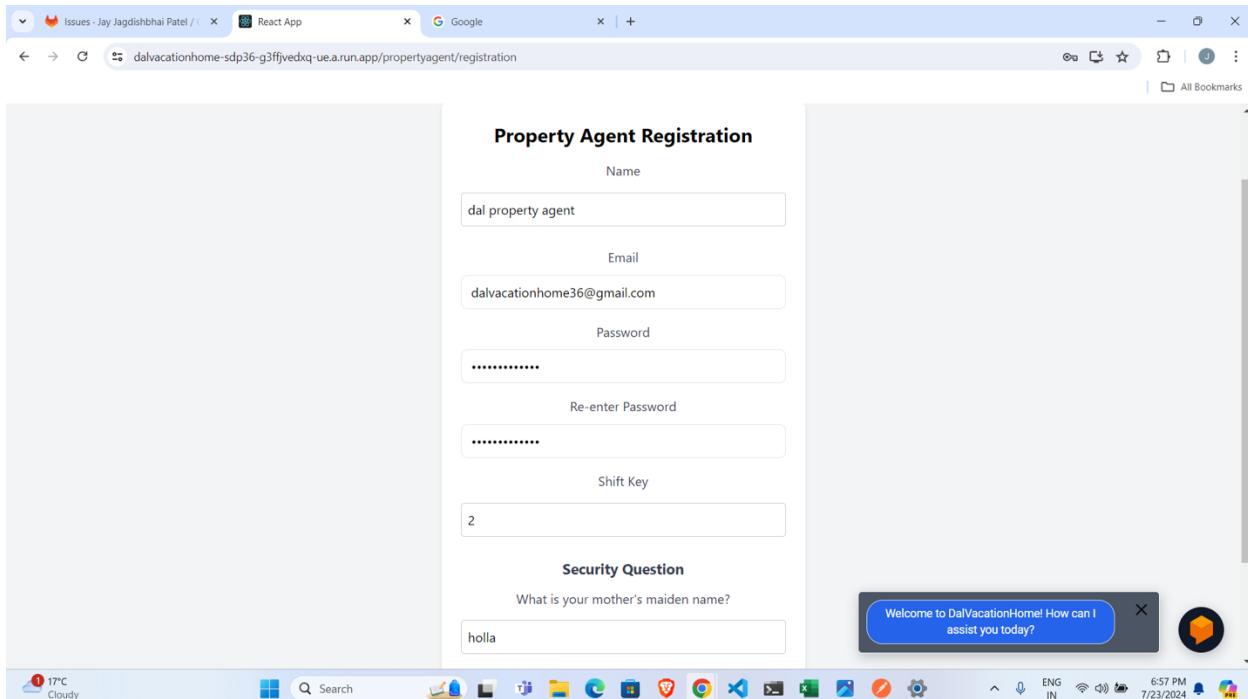


The screenshot shows the Postman application interface. The sidebar has the same structure as Figure 29. The main panel shows a request to 'ChatRetriever' at 'https://us-central1-thematic-answer-427612-g9.cloudfunctions.net/ChatRetriever'. The request method is POST, and the URL is https://us-central1-thematic-answer-427612-g9.cloudfunctions.net/ChatRetriever. The body is set to raw JSON, containing a customer email address. The response status is 200 OK, with a timestamp of 2024-07-24T00:20:42.524Z and a size of 995B.

Figure 30: API call to fetchUserData lambda to fetch the user details and the logging events of the user

# Evidence of Testing for completed modules:

## Notification module Testing:

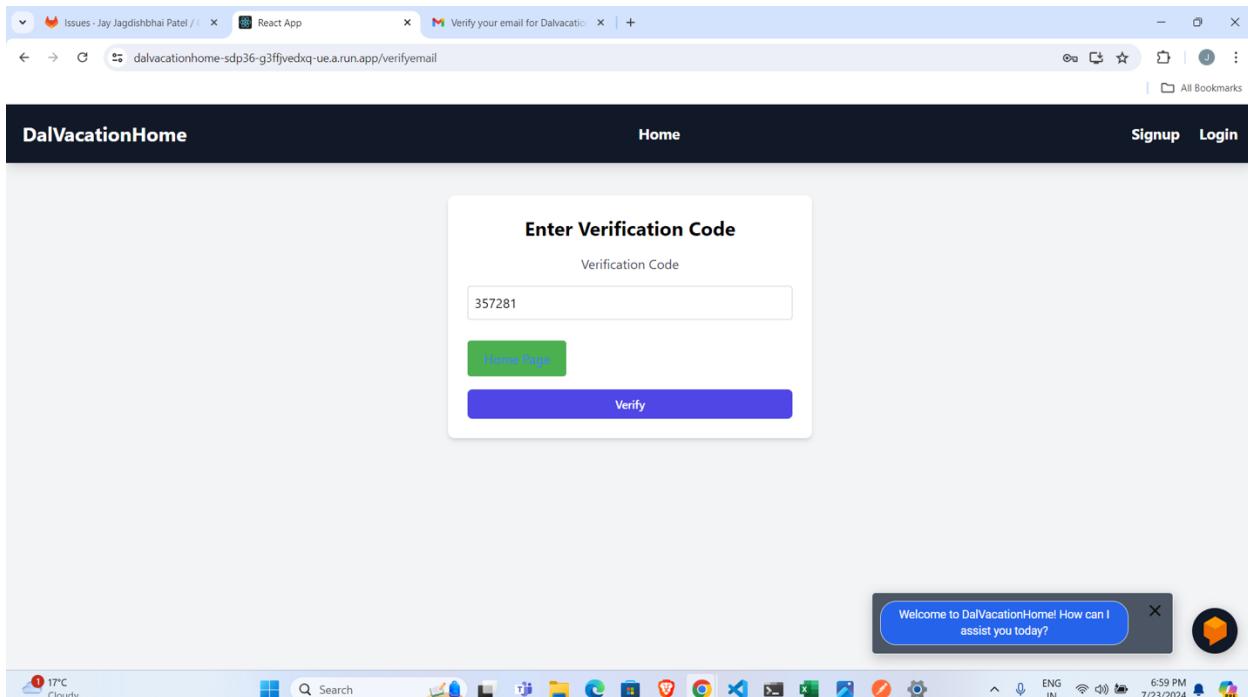


The screenshot shows a web browser window with the title "Property Agent Registration". The form contains the following fields:

- Name: dal property agent
- Email: dalvacationhome36@gmail.com
- Password: (redacted)
- Re-enter Password: (redacted)
- Shift Key: 2
- Security Question: What is your mother's maiden name? (Answer: holla)

A blue notification bar at the bottom right says "Welcome to DalVacationHome! How can I assist you today?"

Figure 31: Screenshot of the Property Agent registration form, including fields for shift key and security question.



The screenshot shows a web browser window with the title "Verify your email for Dalvacation". The page has a dark header with "DalVacationHome", "Home", "Signup", and "Login" buttons. A central modal dialog titled "Enter Verification Code" contains the following fields:

- Verification Code: 357281

Below the input field are two buttons: "Home Page" (green) and "Verify" (blue).

A blue notification bar at the bottom right says "Welcome to DalVacationHome! How can I assist you today?"

Figure 32: Display of the verification code received via email during registration.

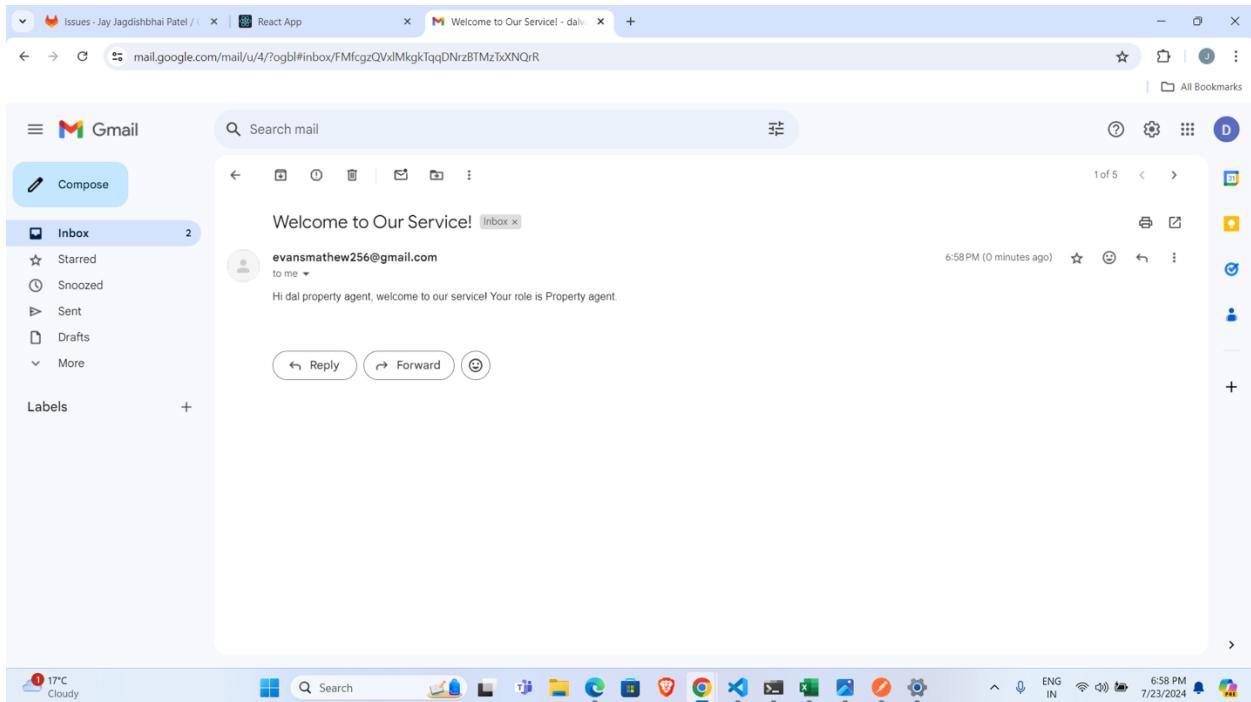


Figure 33: Display of the verification code received via email during registration.

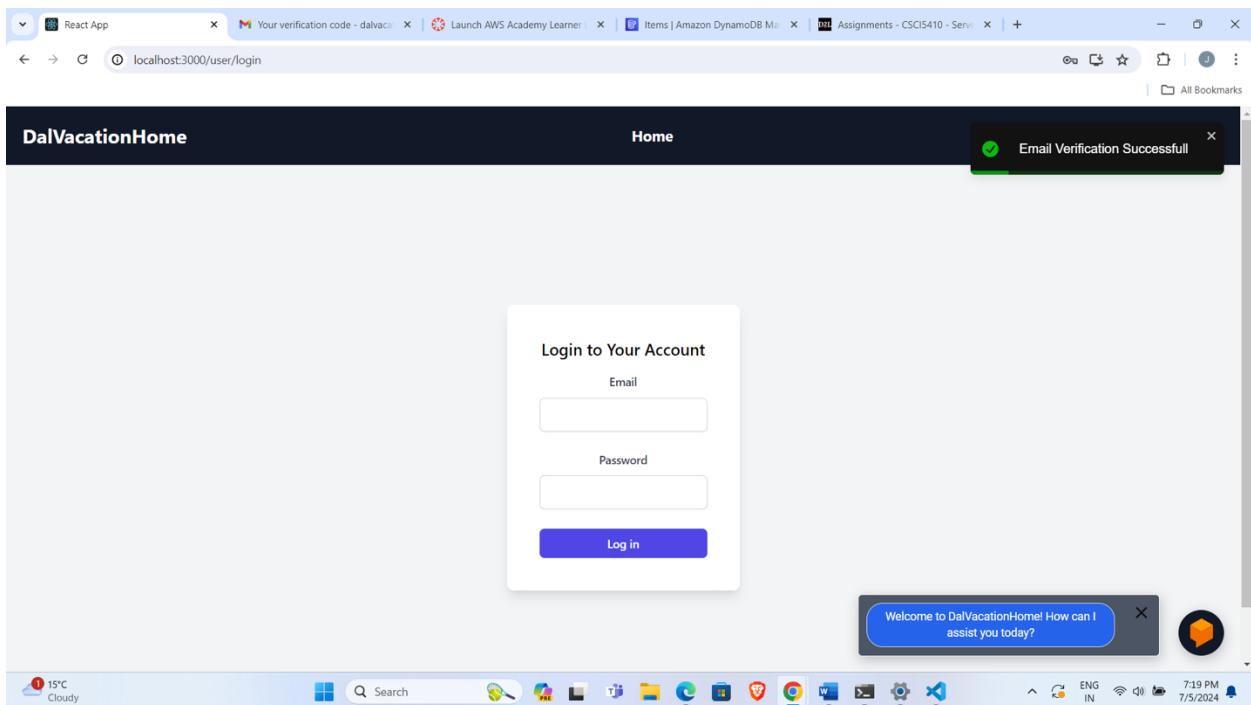


Figure 34: Login screen capture, featuring fields for email and password.

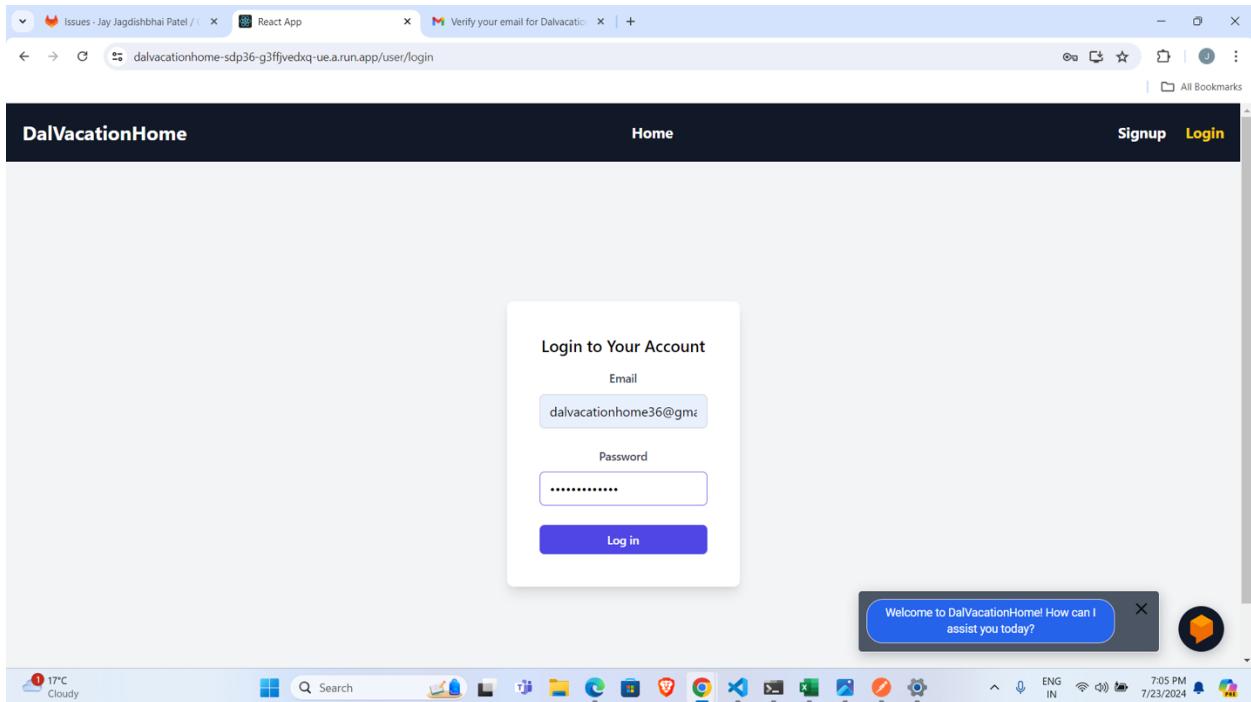


Figure 35: Attempt to log in showing the screen during the process.

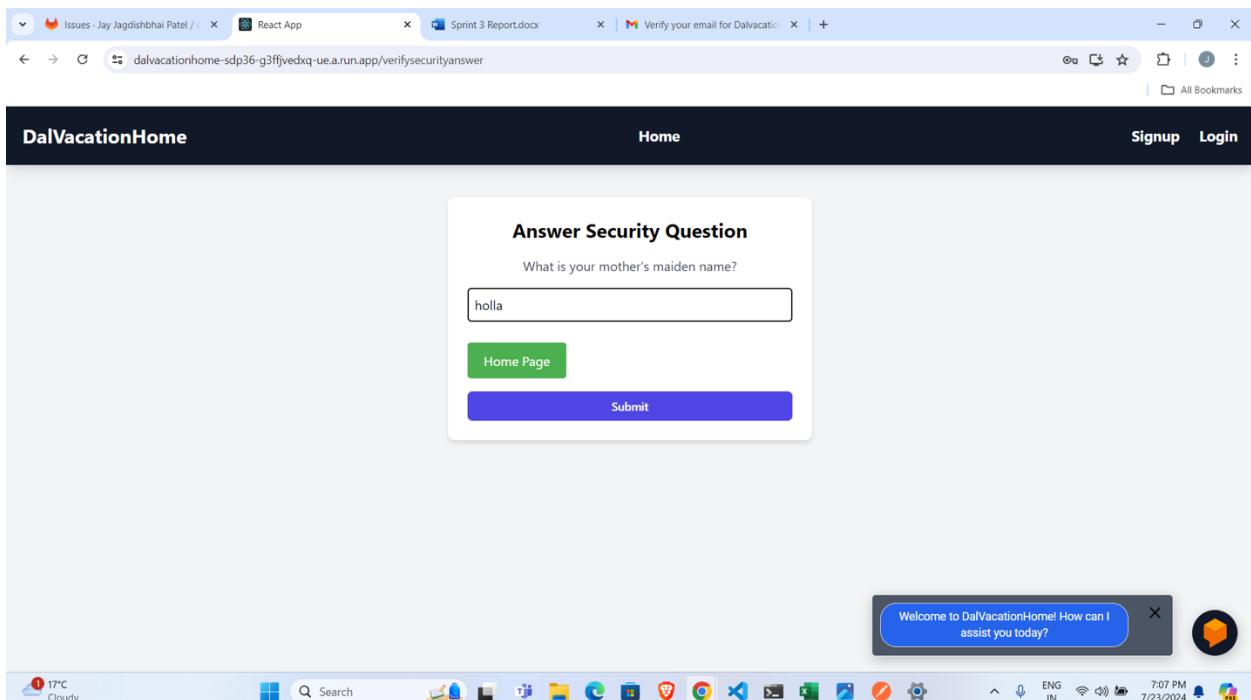


Figure 36: Security question prompt during login for additional verification.

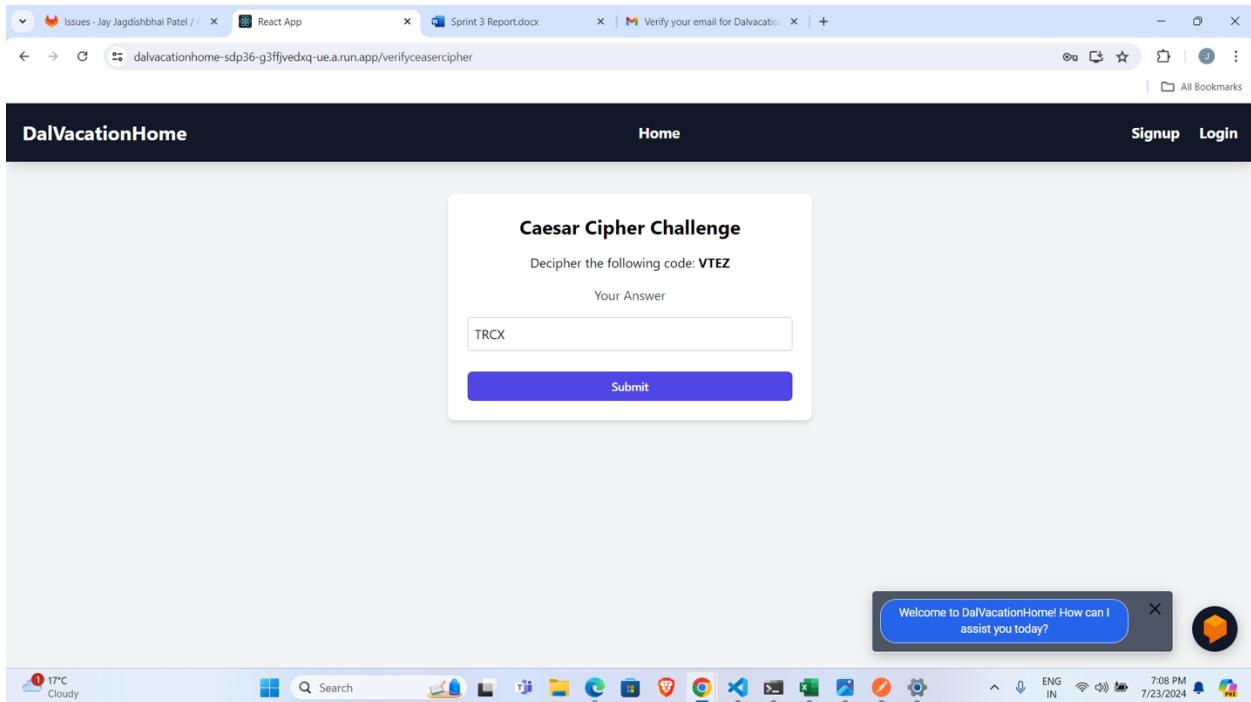


Figure 37: Caesar cipher challenge presented to the user as a login verification step.

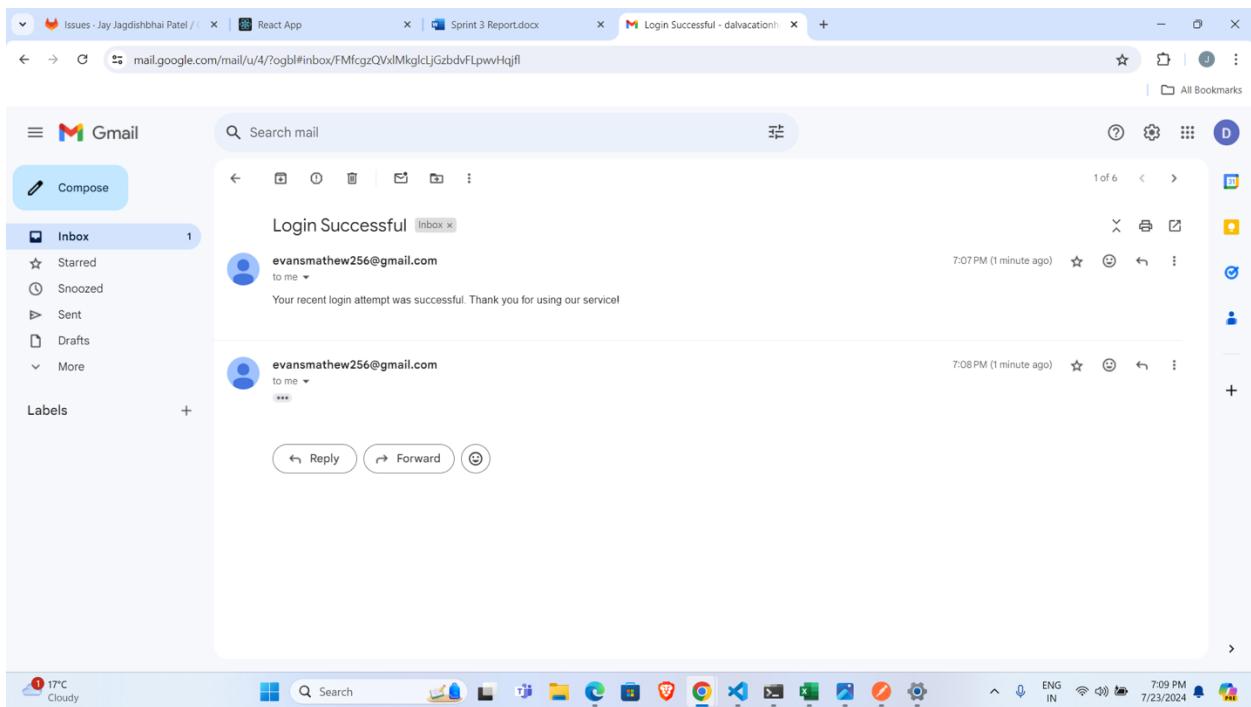


Figure 38: Login successful email.

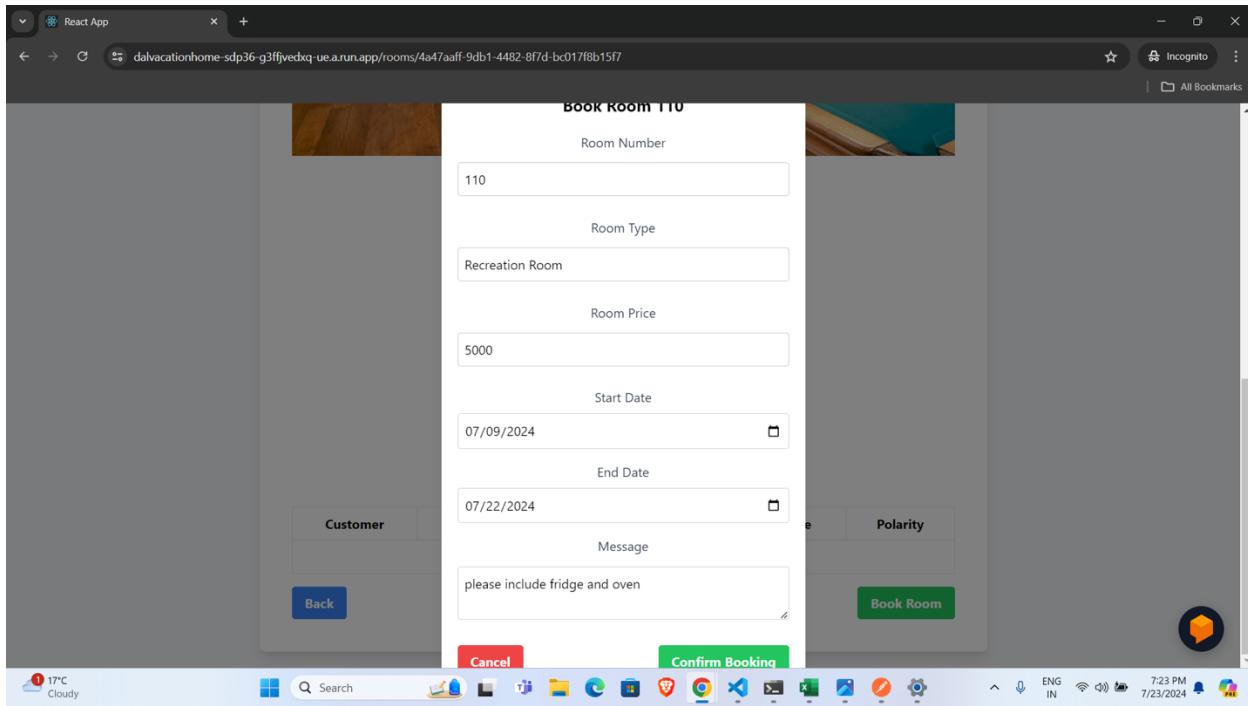


Figure 39: Page for booking confirmation by user.

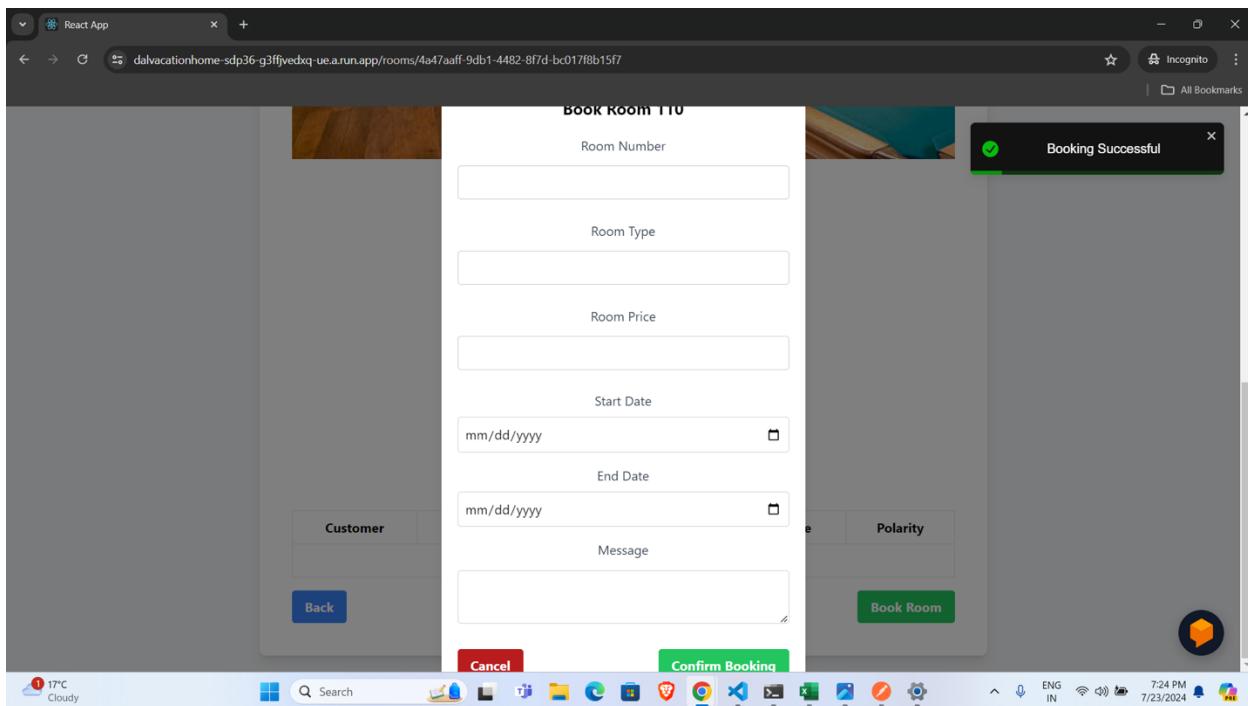


Figure 40: Page 2 for booking confirmation by user

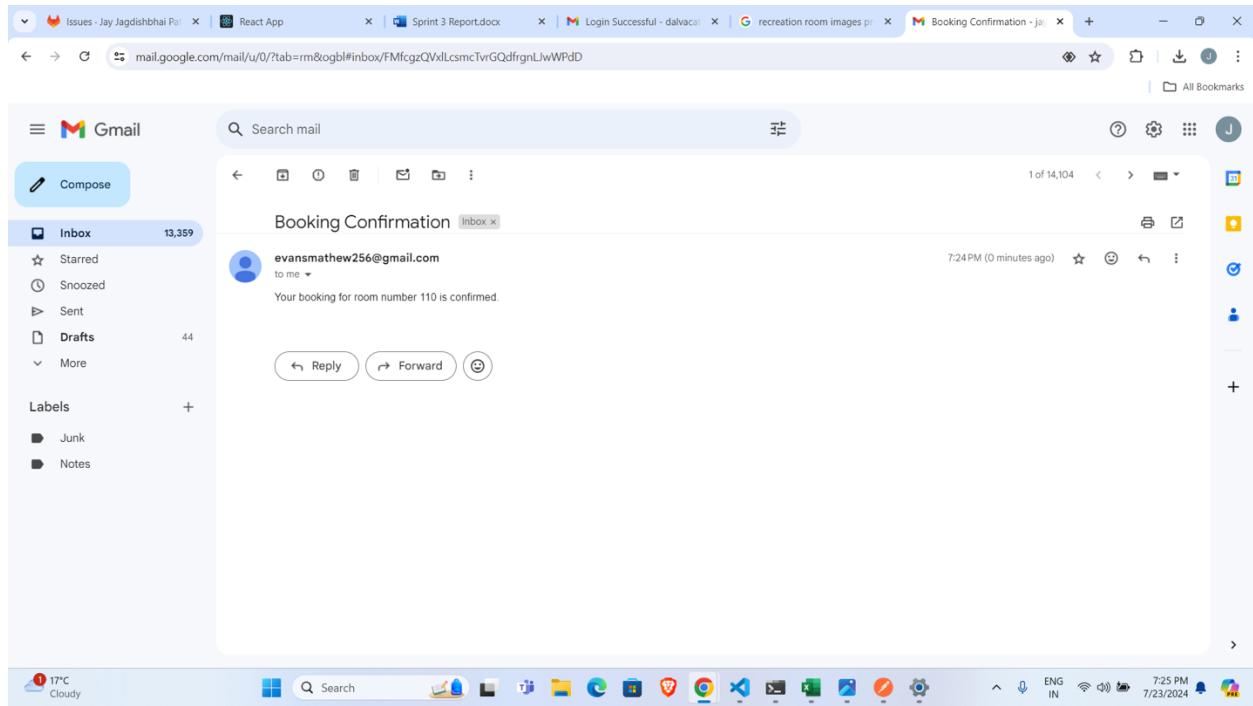


Figure 41: Email notification for confirming booking by user.

## Data Analysis and Visualization Module:

All users can access Feedback of Room:

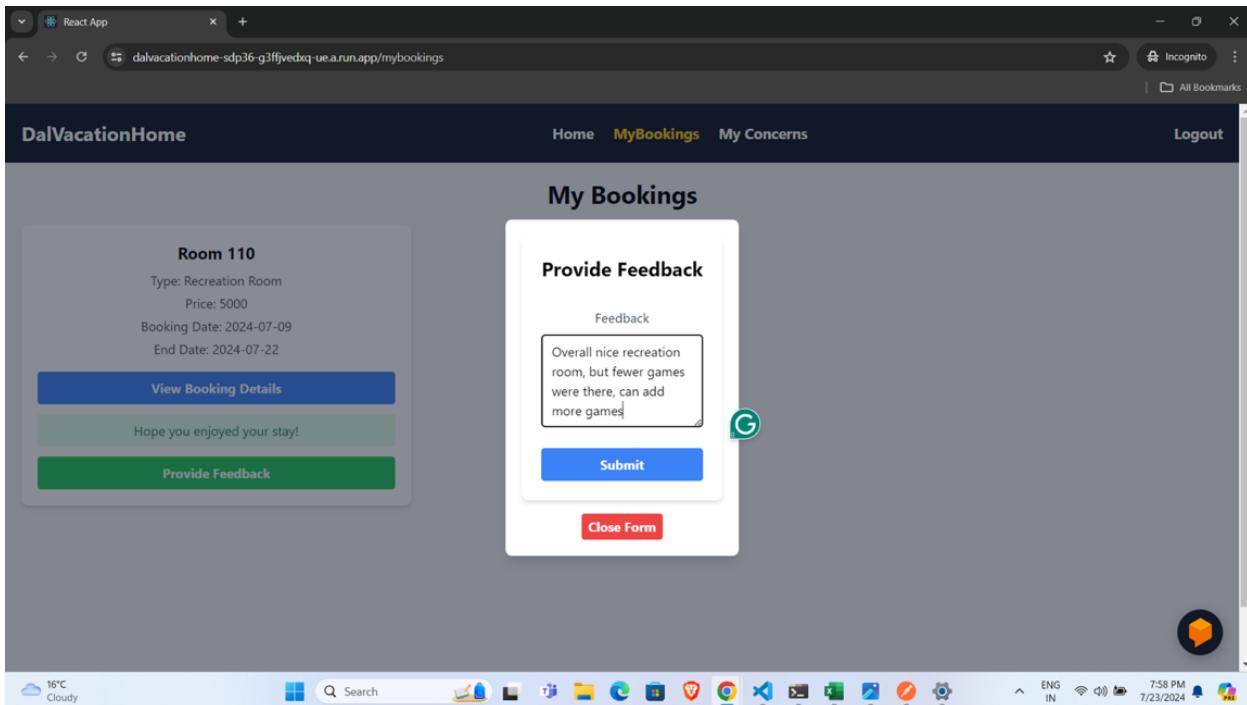


Figure 42: Page for providing feedback

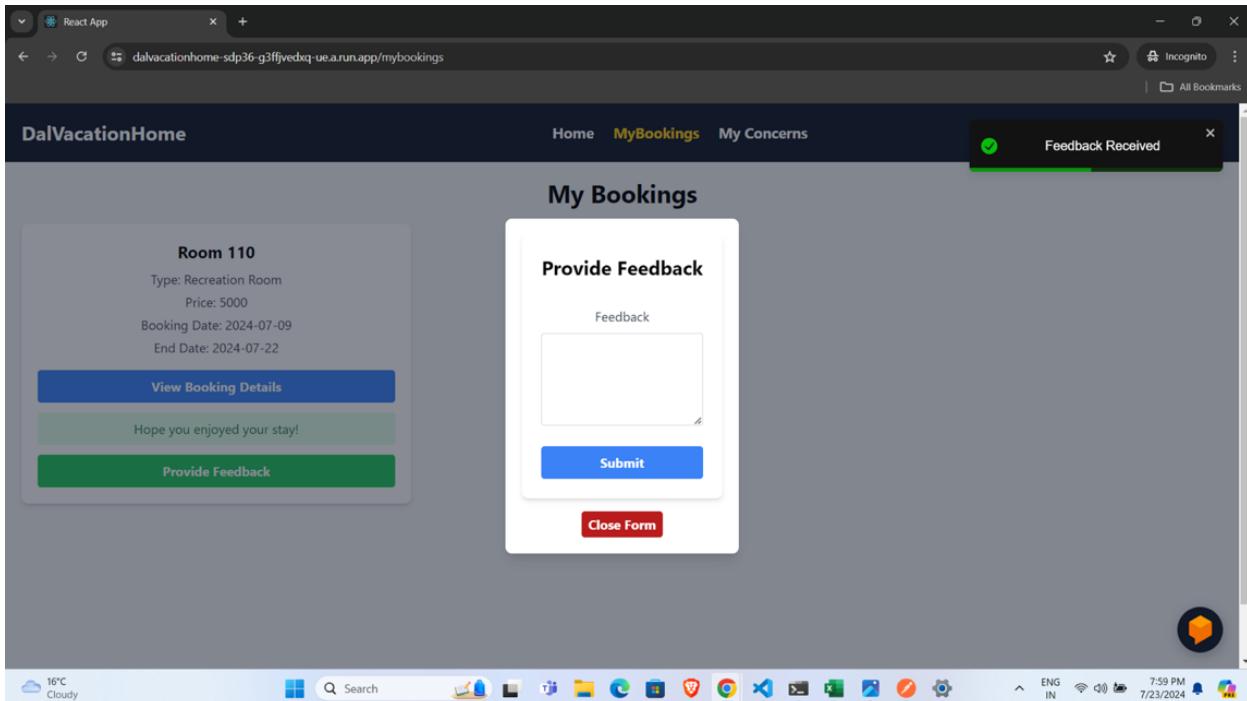


Figure 43: Feedback provided by user form.

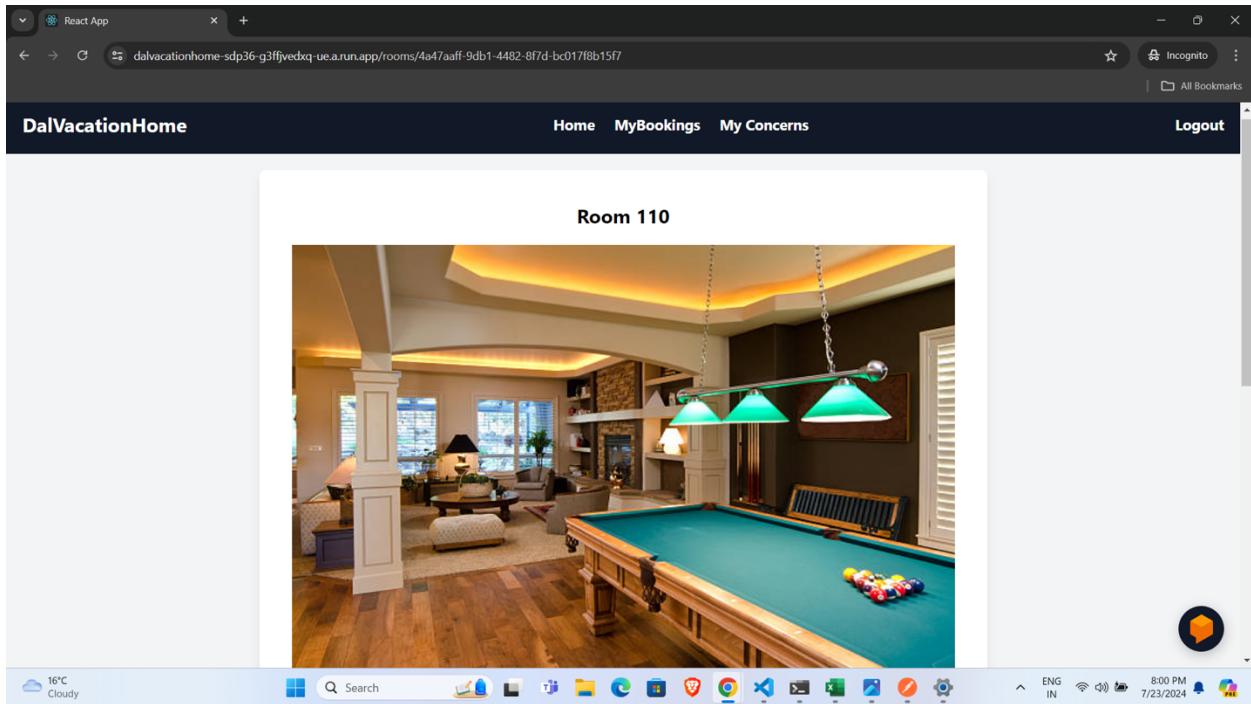


Figure 44: Room Image along with room number on the homepage.

The screenshot shows a detailed view of Room 110. At the top, there's a large image of the room. Below it, the room type is listed as "Recreation Room". The description reads "Very nice place to spend quality time with family". The original price is \$5000, and the discounted price is \$3750.00 (25% off). Availability is marked as "Not Available for Booking". A section titled "Features" lists "Pool Table", "Table tennis", "Chess", "AC", and "Swimming pool". Another section titled "Feedbacks" contains a table with one row. The table has columns for "Customer", "Feedback", "Date", "Score", "Magnitude", and "Polarity". The entry shows a customer named "jay" with the feedback "Overall nice recreation room, but fewer games were there, can add more games", dated 2024-07-23T22:59:52.131Z, a score of 0.10, a magnitude of 0.10, and a polarity of "Neutral". A "Back" button is located at the bottom left of the feedback section. The Windows taskbar at the bottom shows various pinned icons and the date/time as 7/23/2024.

Figure 45: Book details such as number, type, price, feedback, etc.

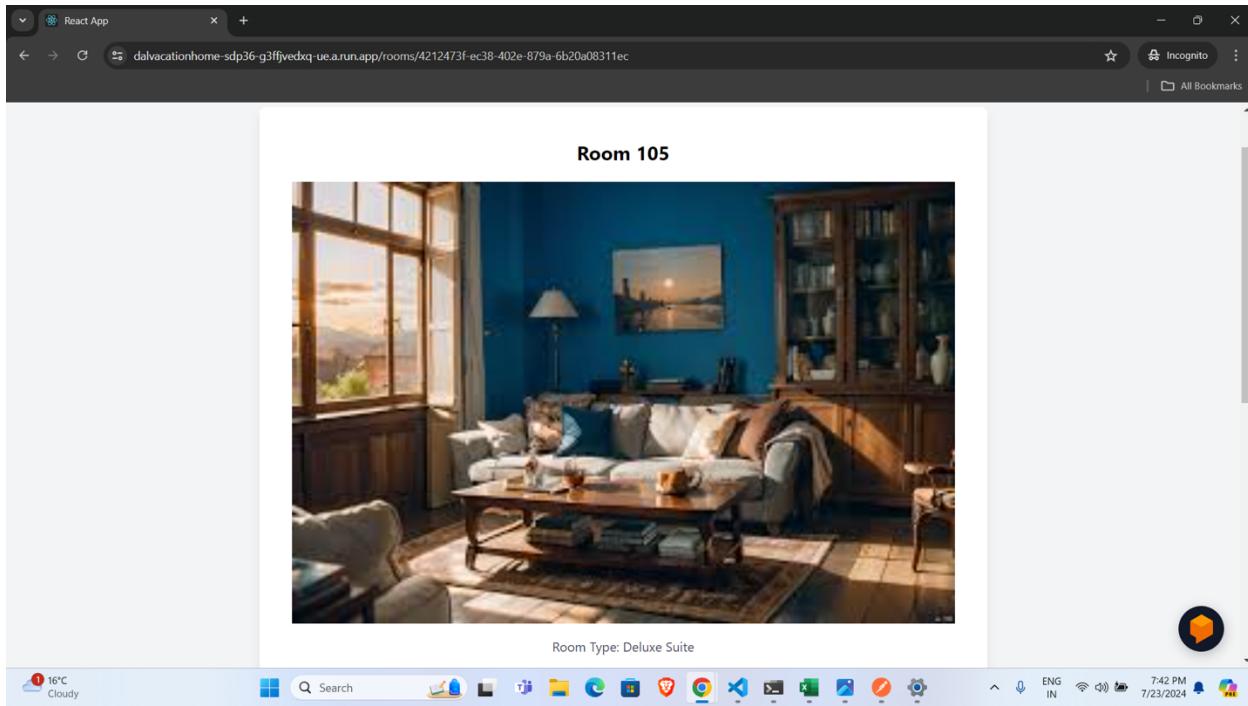


Figure 46: Room 105 with its image.

Customer	Feedback	Date	Score	Magnitude	Polarity
jay	Highly Recommended this room	2024-07-13T16:51:50.836Z	0.90	0.90	Clearly Positive
jay	The room was comfortable and the amenities were satisfactory.	2024-07-23T21:00:52.610Z	0.50	0.50	Slightly Positive
jeet	Not really like the room. Room space is so small.	2024-07-23T21:00:52.610Z	-0.80	1.70	Clearly Negative

Figure 47: Room 105 details with positive and negative feedbacks.

## Property Agent Side:

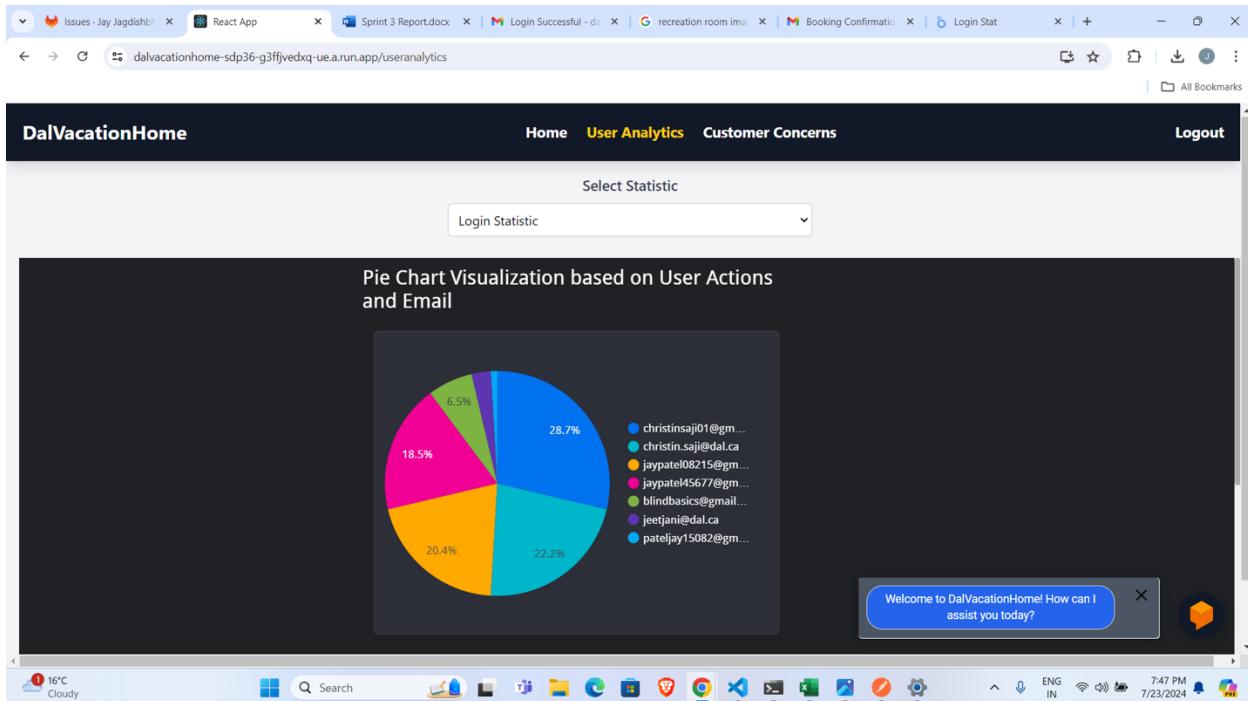


Figure 48: Visualization of user Actions and Email.

The screenshot shows a web browser window with multiple tabs open. The active tab is titled "User Analytics". The page has a dark header with "DalVacationHome", "Home", "User Analytics", "Customer Concerns", and "Logout" buttons. Below the header is a "Select Statistic" dropdown set to "Login Statistic". A table titled "User Login Statistics Table" is displayed, showing the following logins for the date Jul 23, 2024:

Email	Date	Time	Action
christinsaji01@gmail.com	Jul 23, 2024	2:59:26	login
christinsaji01@gmail.com	Jul 23, 2024	2:51:39	logout
christinsaji01@gmail.com	Jul 23, 2024	2:50:49	ceaser cipher passed
christinsaji01@gmail.com	Jul 23, 2024	2:50:37	security question passed
christinsaji01@gmail.com	Jul 23, 2024	2:50:32	login
christinsaji01@gmail.com	Jul 23, 2024	2:49:44	registration
christinsaji01@gmail.com	Jul 23, 2024	2:44:54	login
christin.saji@dal.ca	Jul 23, 2024	21:23:03	login
jay Patel 45677@gmail.com	Jul 23, 2024	18:23:00	ceaser cipher passed
jay Patel 45677@gmail.com	Jul 23, 2024	18:22:52	security question passed

Figure 49: User login Statistics Table.

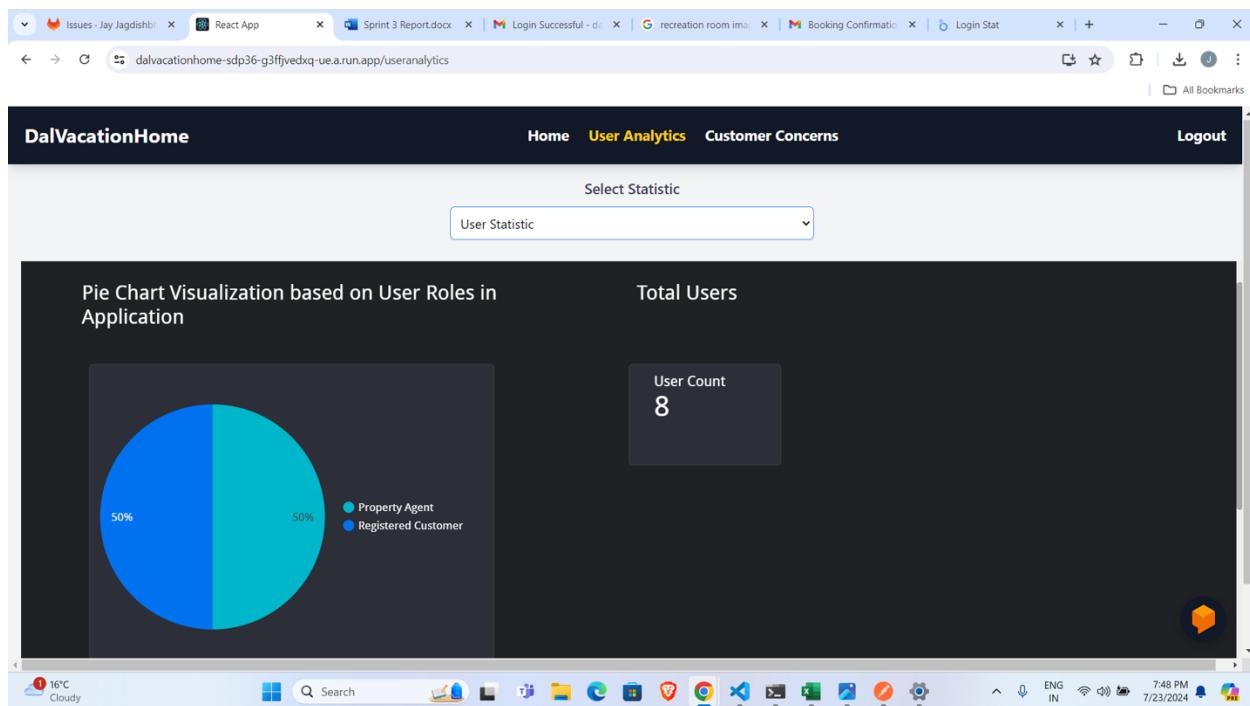


Figure 50: Pie chart visualization based on roles in the application and user count.

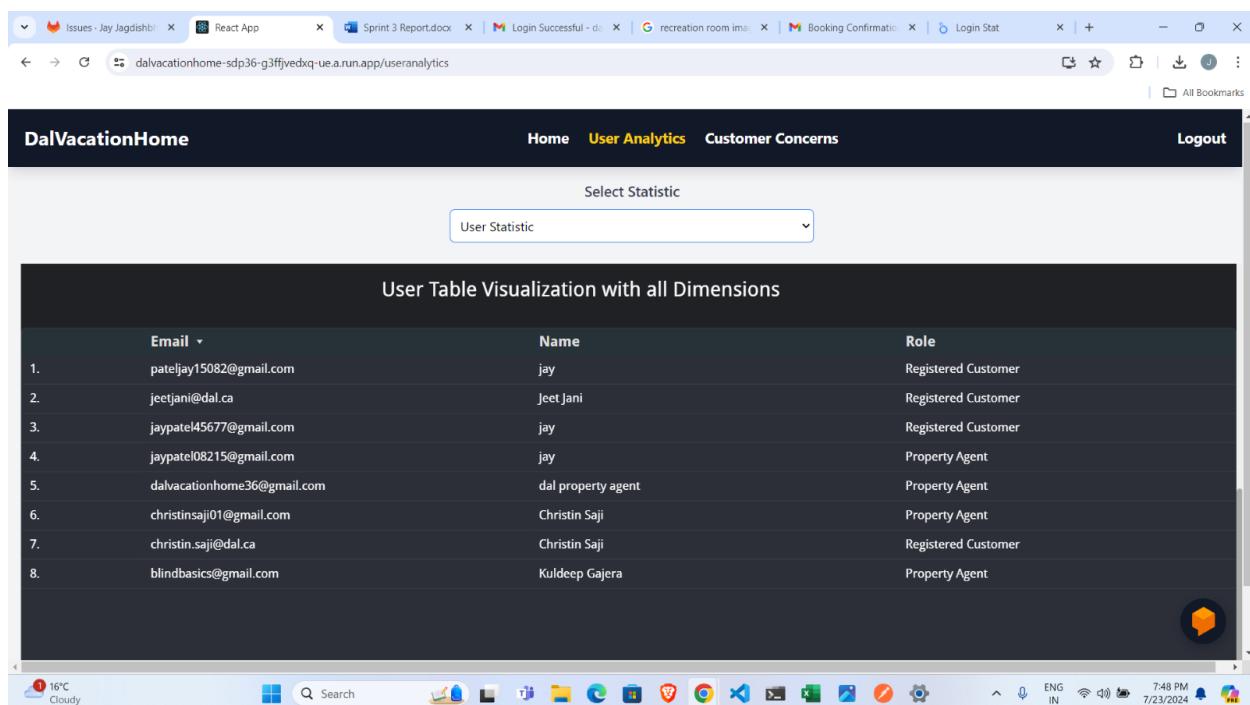


Figure 51: User table visualization with all the dimensions.

## Virtual Assistance and Message Passing Module: Customer Side:

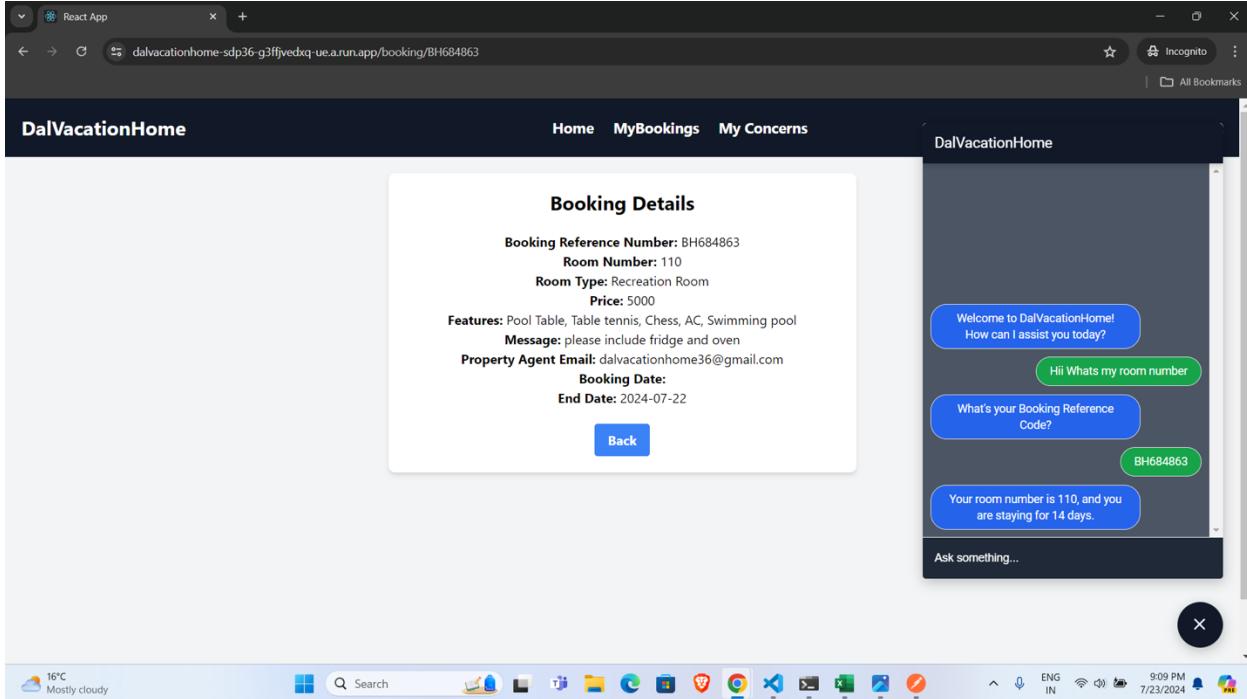


Figure 52: Booking details and chatting with chatbot from customer side.

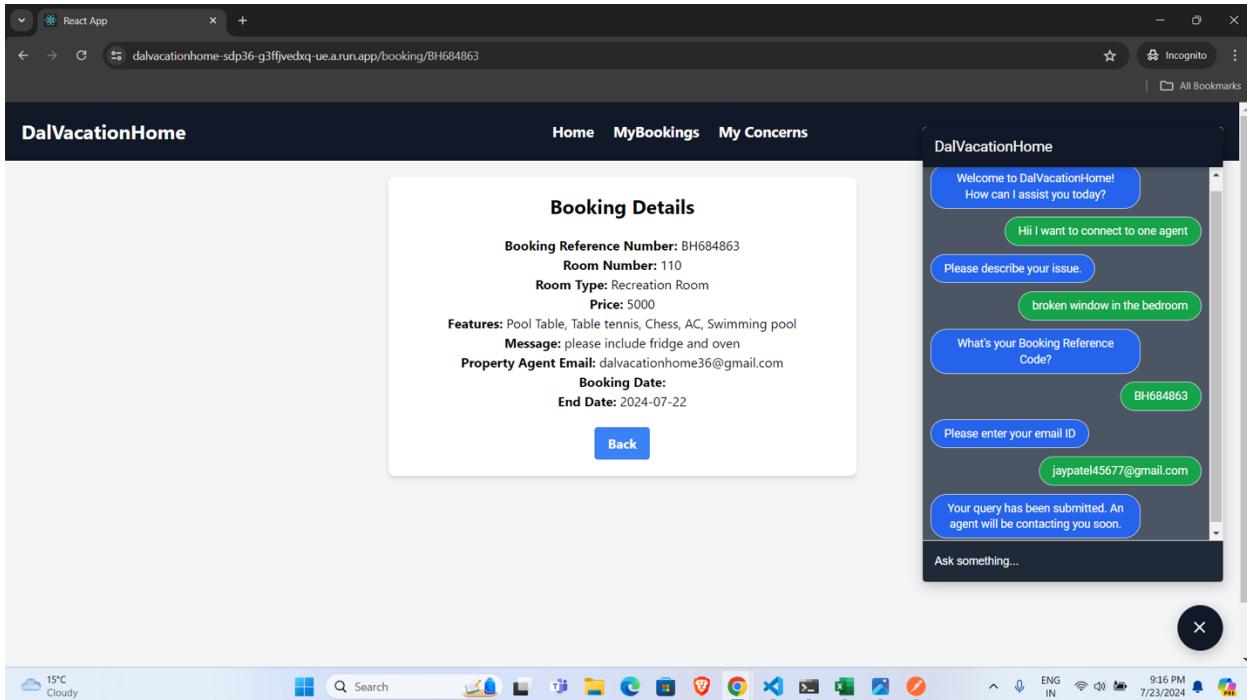


Figure 53: Conversing with chatbot about connecting to property agent.

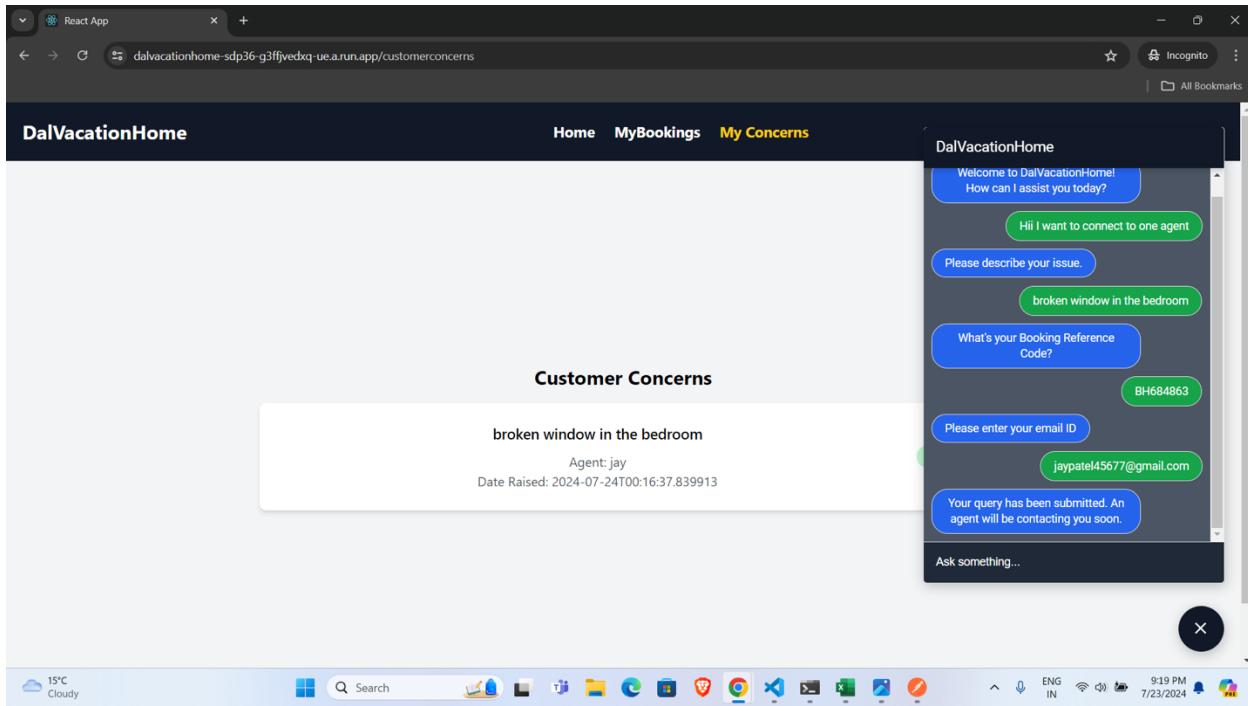


Figure 54: Issue provided with booking id and email id

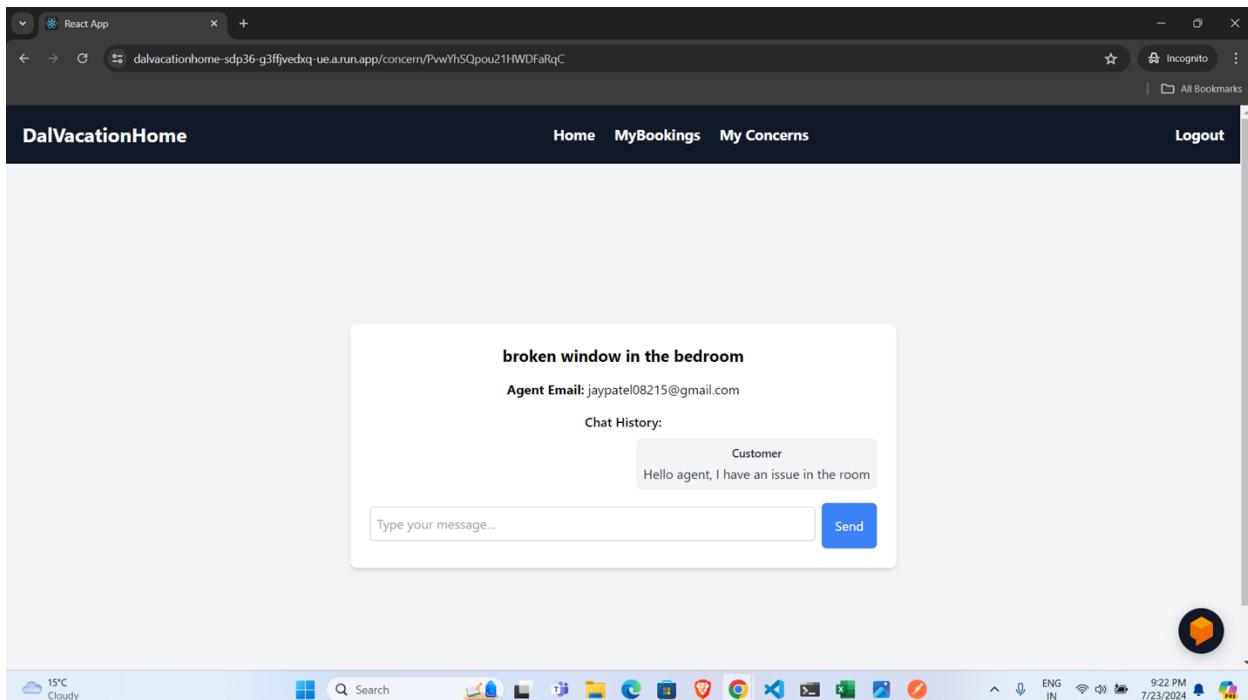


Figure 55: Chatting with property agent about the issue.

## Property Agent Side:

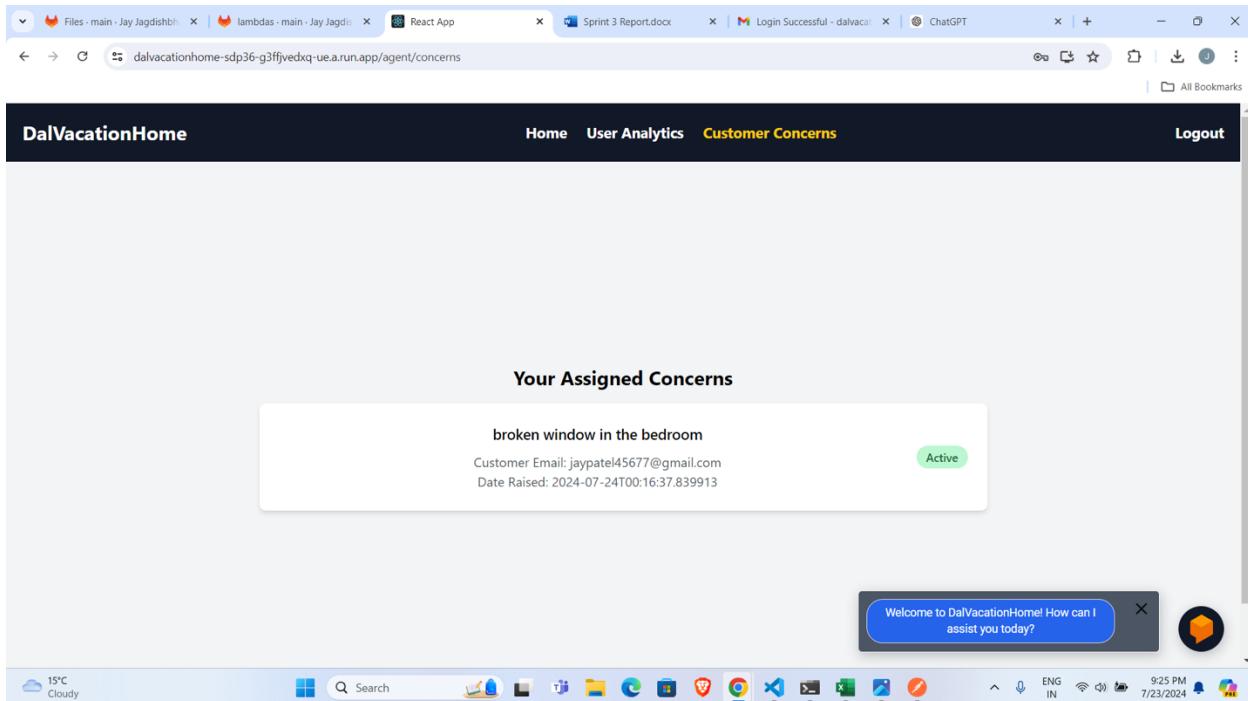


Figure 56: ticket concern active between customer and property agent.

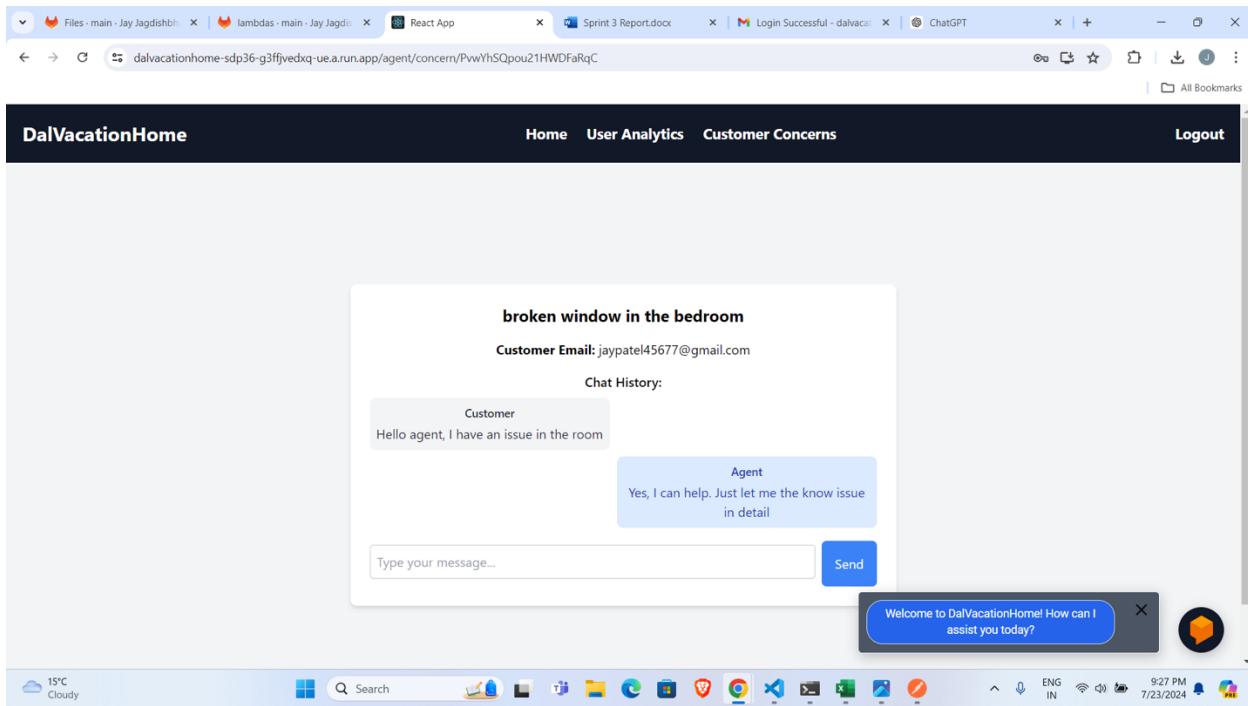


Figure 57: Chat history between customer and property agent.

## Customer Side updated chat:

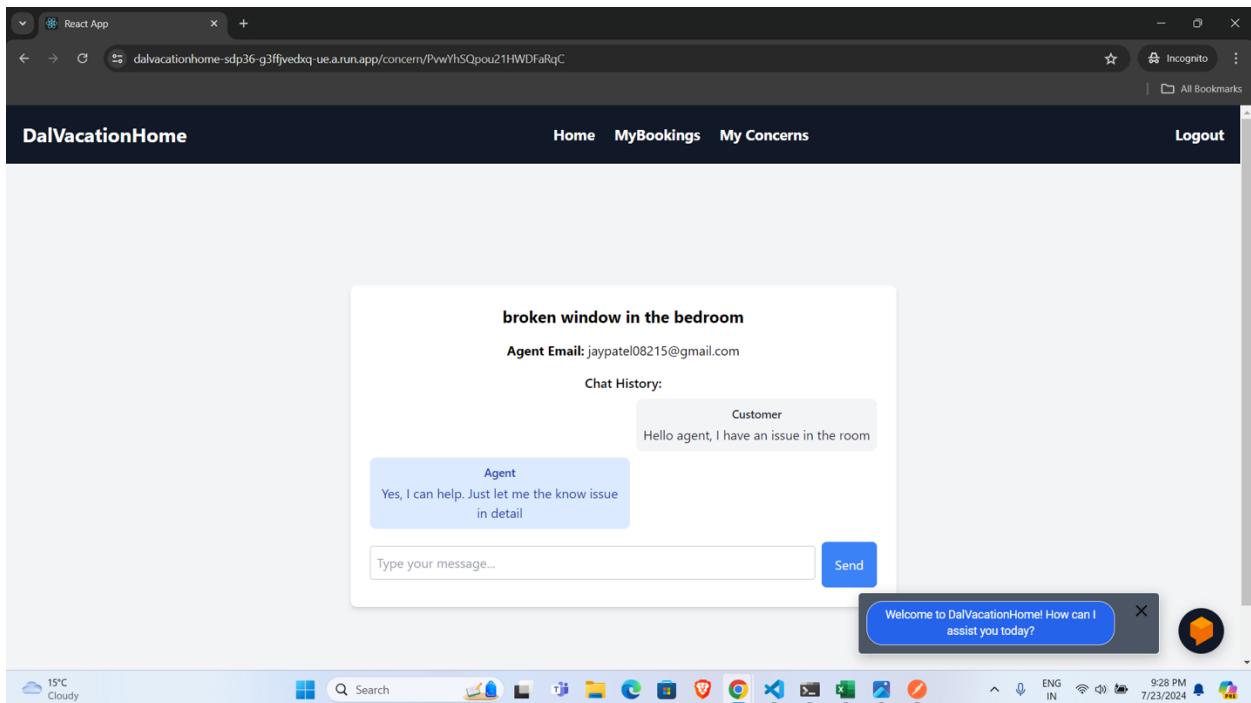


Figure 58: Property agent giving response to customer's query.

## Meeting Logs:

Agenda	Outcome of discussion	Link to meeting recordings
<ul style="list-style-type: none"> <li>Share status of each module.</li> <li>Discuss completed and pending parts.</li> <li>Identify issues in the chatbot.</li> <li>Prepare for the upcoming meeting with Head TA.</li> <li>Plan live meeting for module presentations.</li> </ul>	<ul style="list-style-type: none"> <li>Shared status of each module.</li> <li>Identified completed and pending parts.</li> <li>Discussed and identified issues in the chatbot that need fixing.</li> <li>Prepared for the meeting with Head TA.</li> <li>Decided to hold a live meeting where everyone will describe their module's functionality.</li> </ul>	<a href="#">Call with Serverless Team 36-20240715 204120-Meeting Recording.mp4</a>
<ul style="list-style-type: none"> <li>Explain each module for better understanding.</li> <li>Demonstrate module setups.</li> </ul>	<ul style="list-style-type: none"> <li>Each member explained their module, enhancing team understanding.</li> <li>Demonstrated module setups, facilitating better comprehension of workflows.</li> <li>Prepared for the meeting with the TA.</li> </ul>	<a href="#">Call with Serverless Team 36-20240716 192449-Meeting Recording.mp4</a>
<ul style="list-style-type: none"> <li>Discuss SNS &amp; SQS module.</li> <li>Plan and start the report.</li> <li>Share access to final architecture.</li> <li>Identify remaining project tasks.</li> <li>Plan final demo video recording.</li> <li>Finalize Sprint 3 report.</li> <li>Identify and plan to fix bugs in the application.</li> </ul>	<ul style="list-style-type: none"> <li>Discussed the workings of the SNS &amp; SQS module.</li> <li>Planned the report structure and initiated the writing process.</li> <li>Shared access to the final architecture for module modifications.</li> <li>Identified remaining project tasks</li> <li>Planned the final demo video recording and scheduled a meeting to finalize the Sprint 3 report.</li> <li>Identified bugs in the application and planned necessary fixes.</li> </ul>	<a href="#">Meeting in Serverless Team 36-20240722 192536-Meeting Recording.mp4</a>

# Chat Screenshots:

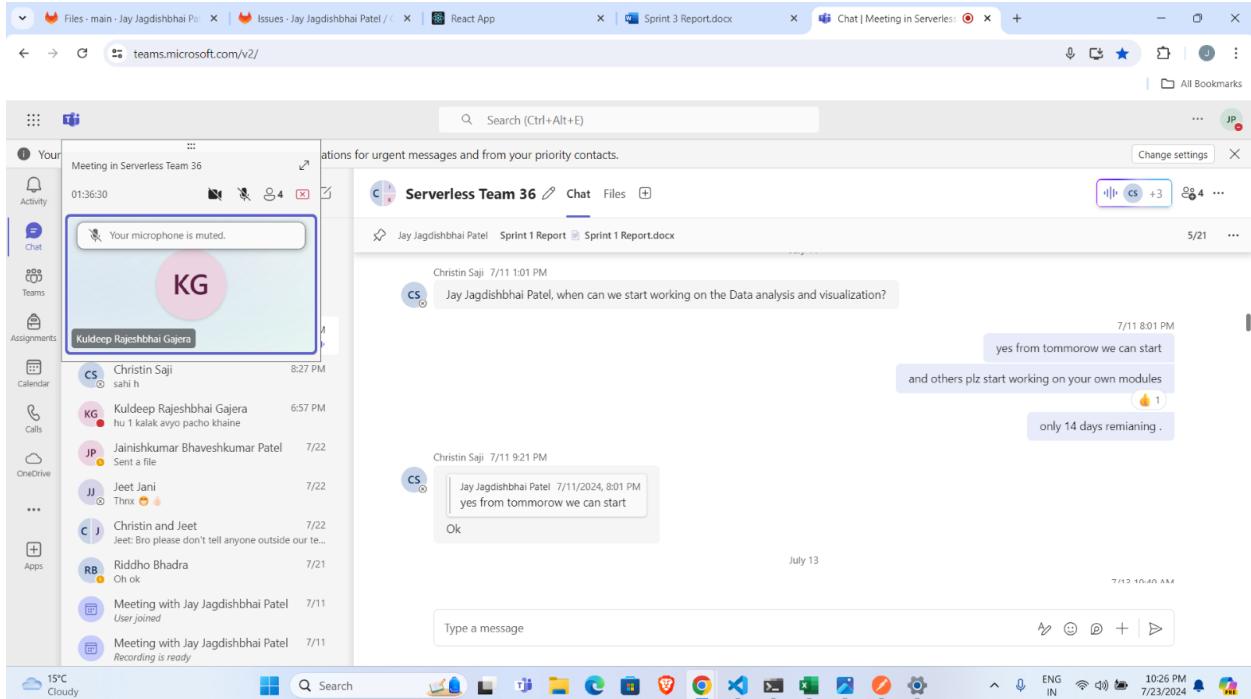


Figure 59: Team members discussing project updates and strategies in a team chat.

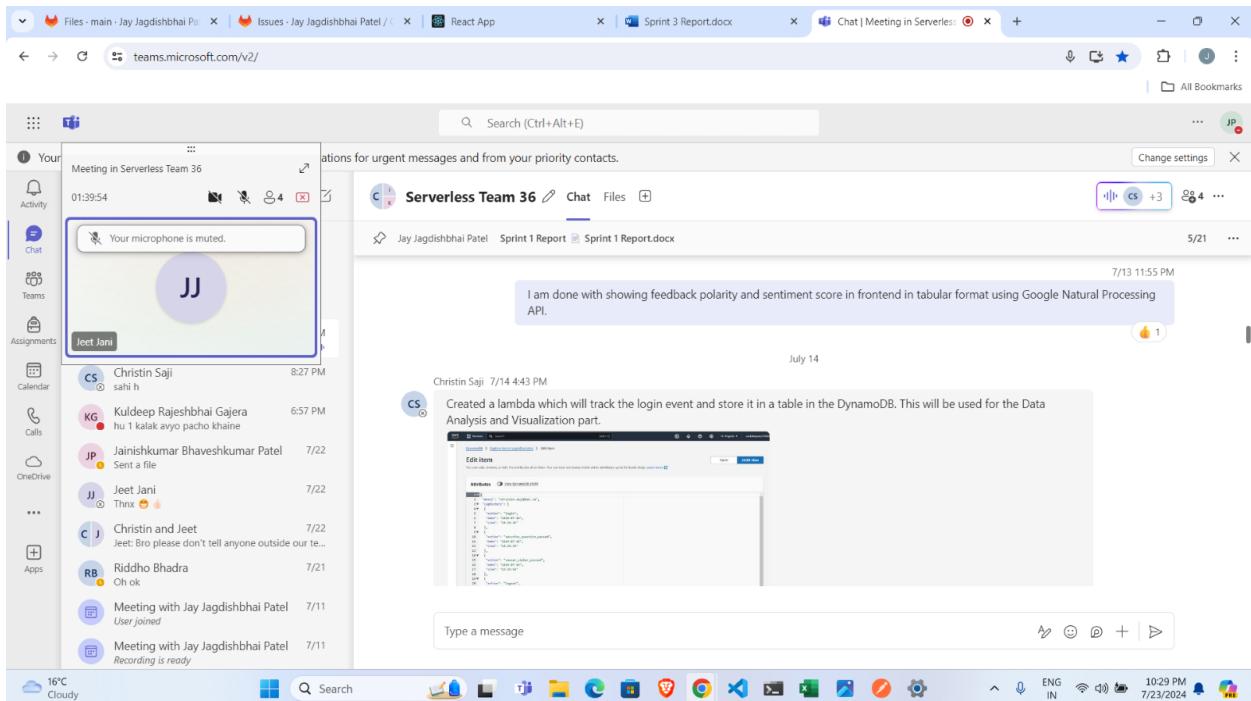


Figure 60: Team members discussing project updates and strategies in a team chat.

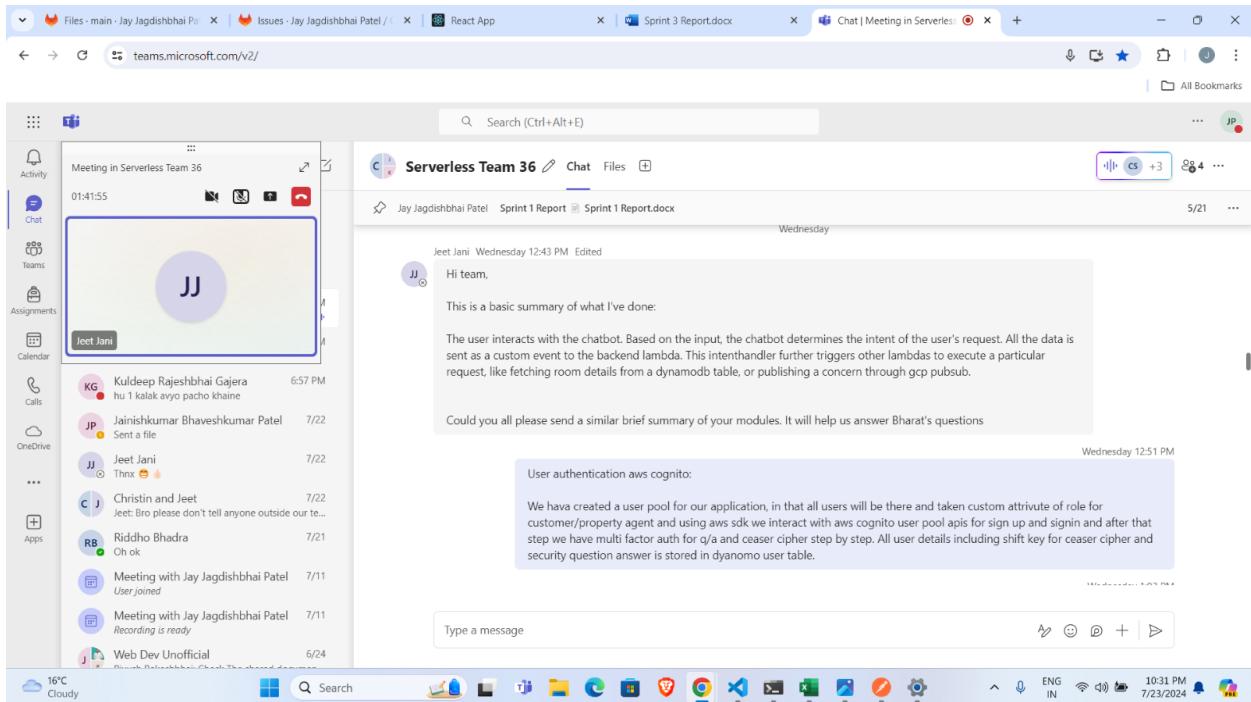


Figure 61: Team members discussing project updates and strategies in a team chat.

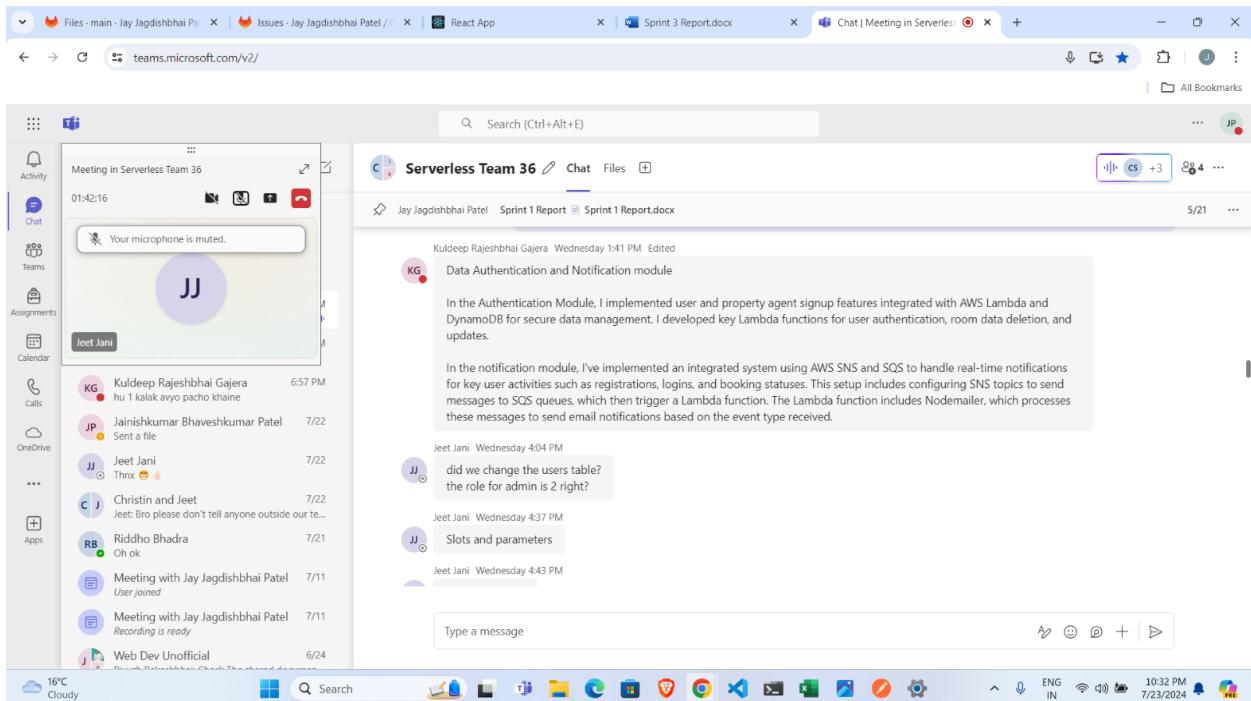


Figure 62: Team members discussing project updates and strategies in a team chat.

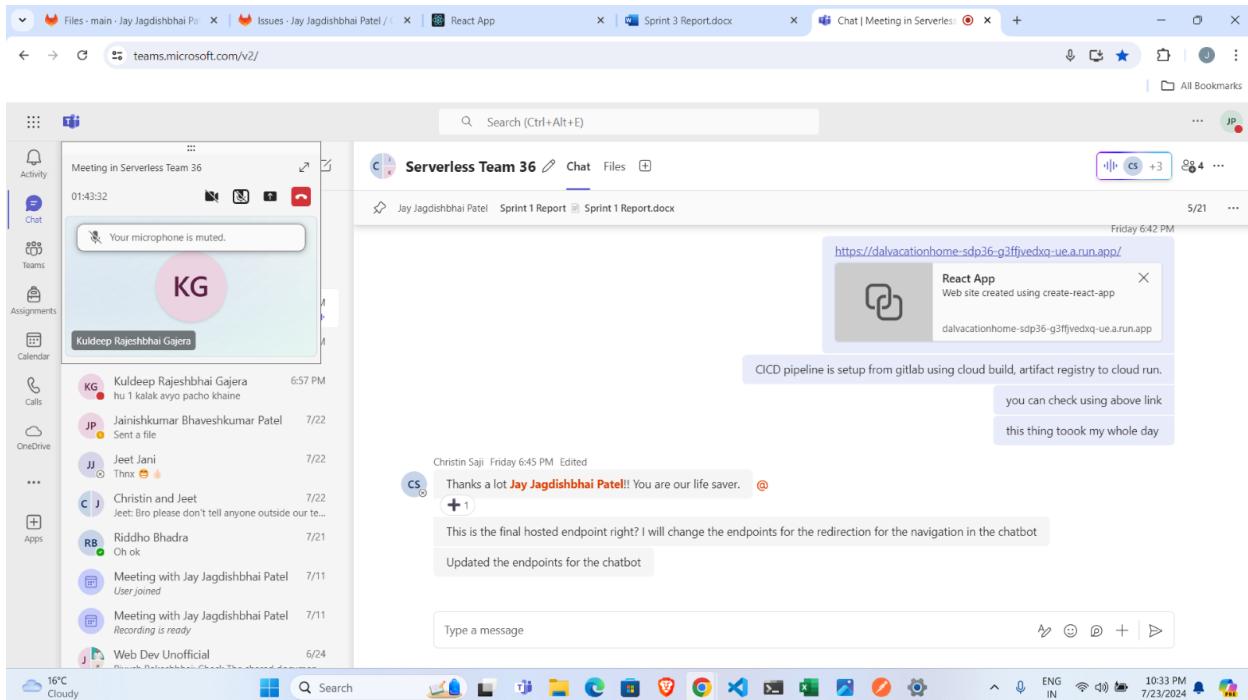


Figure 63: Team members discussing project updates and strategies in a team chat.

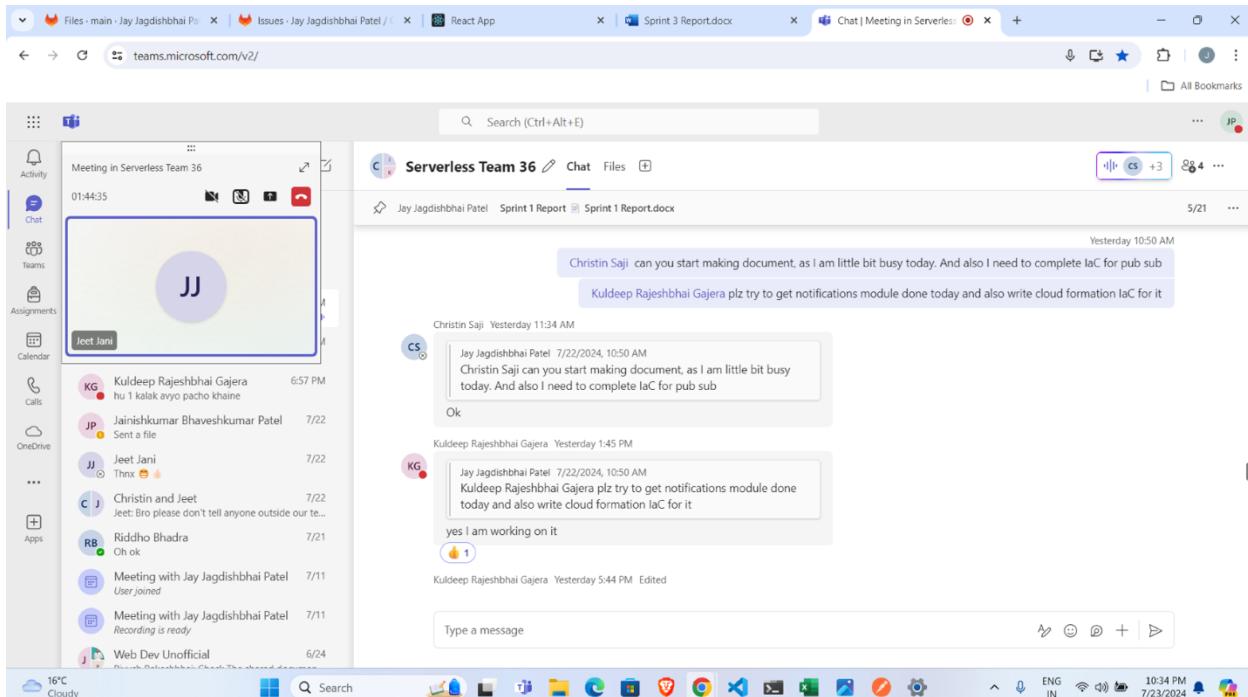
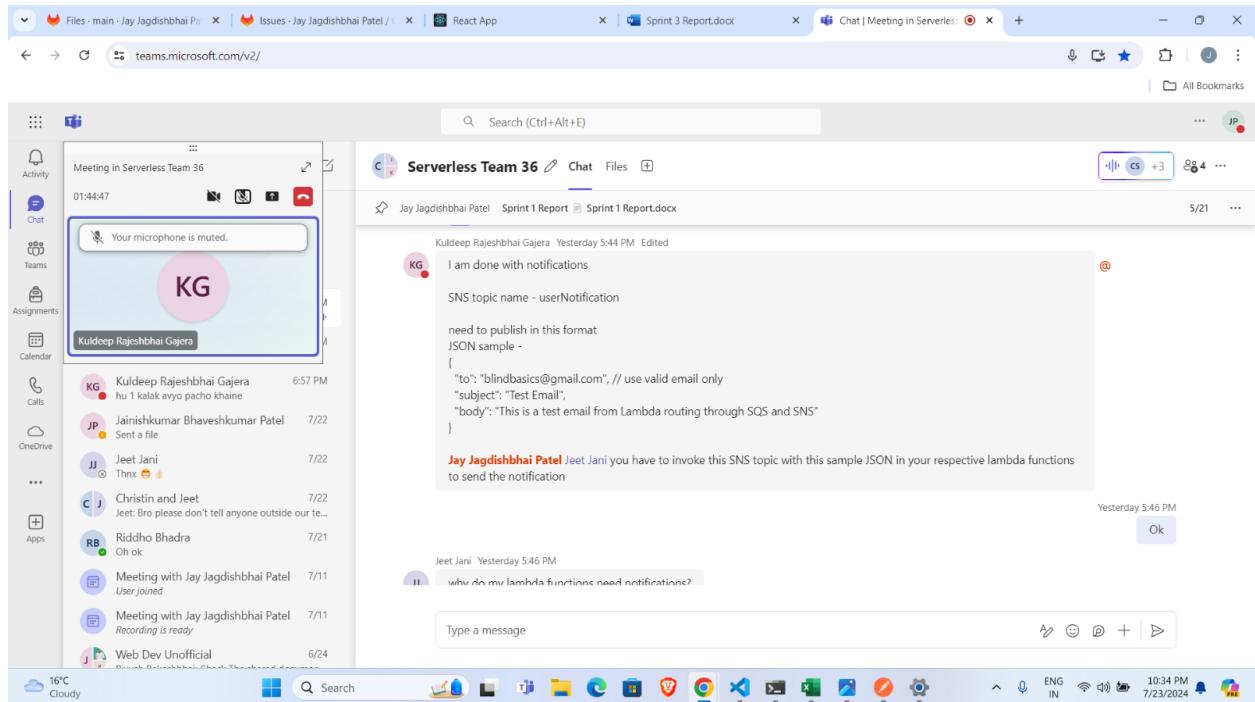


Figure 64: Team members discussing project updates and strategies in a team chat.



*Figure 65: Team members discussing project updates and strategies in a team chat.*

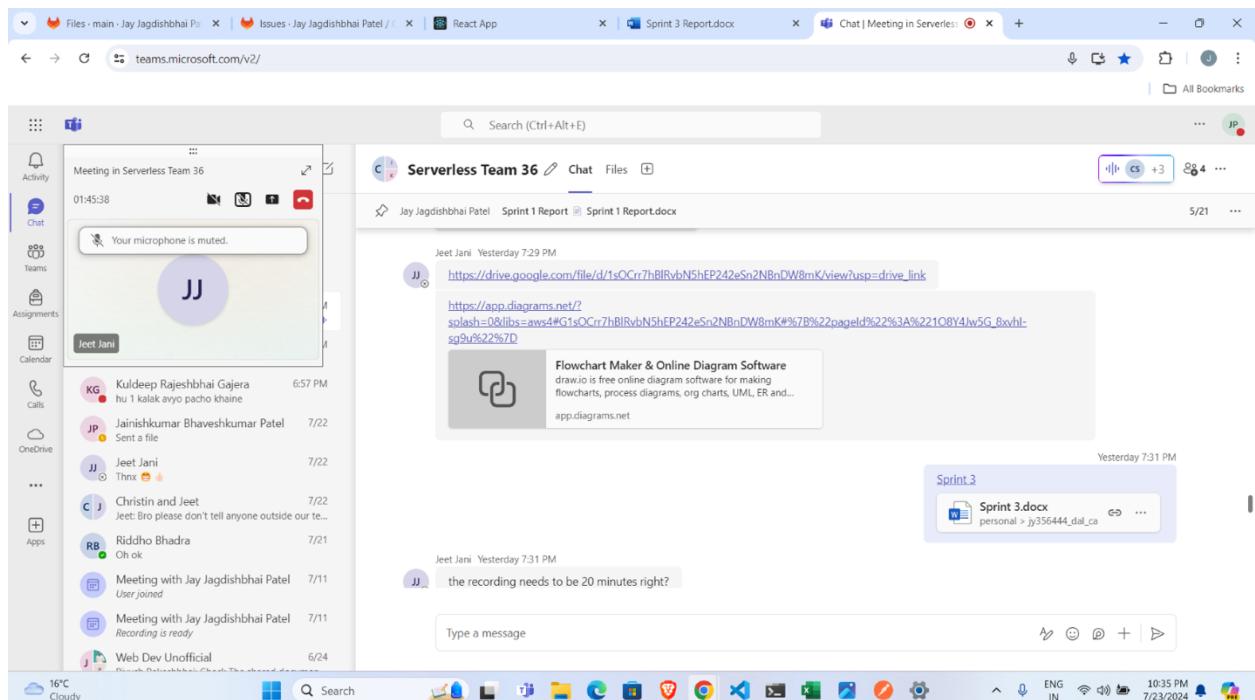


Figure 66: Team members discussing project updates and strategies in a team chat.

# Project Management (Sprint 3) Gitlab Board:

The screenshot shows the Gitlab interface for Sprint 3. On the left, the sidebar includes 'Project' (CSCI5410-S24-SDP-36), 'Pinned', 'Issues' (0), 'Manage', 'Plan', 'Issues' (0), and 'Issue boards'. Under 'Issue boards', 'Milestones', 'Wiki', and 'Requirements' are listed. Below them are 'Code', 'Build', 'Secure', 'Deploy', and 'Help'. The main area displays two columns: 'Open' and 'Closed'. The 'Closed' column contains five items, each with a 'Completed' status badge: 'Message Passing Module (GCP Pub/Sub)' (#21, Today), 'Web Application Building and Deployment of Cloud Services and React Frontend (GCP Cloud Run, Cloud Formation, Terraform)' (#6, Today), 'Notifications' (#10, Today), 'Data Analytics and Visualization Module' (#9, Jul 21), and 'Virtual Assistant Module' (highlighted with a red underline). A search bar at the top right allows filtering by labels and grouping. A 'Create list' button is also present.

Figure 67: Gitlab Board for Sprint 3 (all issues are closed and completed).

The screenshot shows the Gitlab interface for the project's milestones. The sidebar includes 'Project' (CSCI5410-S24-SDP-36), 'Pinned', 'Issues' (0), 'Manage', 'Plan', 'Issues' (0), and 'Milestones'. Under 'Milestones', 'Wiki', 'Requirements', 'Code', 'Build', 'Secure', 'Deploy', and 'Help' are listed. The main area displays three sprints: 'Sprint 3' (Jul 6, 2024–Jul 23, 2024), 'Sprint 2' (May 30, 2024–Jul 5, 2024), and 'Sprint 1' (May 18, 2024–May 29, 2024). Each sprint has a progress bar indicating '28 Issues · 7 Merge requests' (100% complete), '12 Issues · 5 Merge requests' (100% complete), and '5 Issues · 0 Merge requests' (100% complete) respectively. Each sprint is labeled as 'Closed' and associated with the user 'Jay Jagdishbhai Patel / CSCI5410-S24-SDP-36'. A 'New milestone' button is visible at the top right. A search bar and filter options are at the top, and a toolbar with various icons is at the bottom.

Figure 68: Gitlab Board: All Milestone (Sprints) are completed on time and closed.

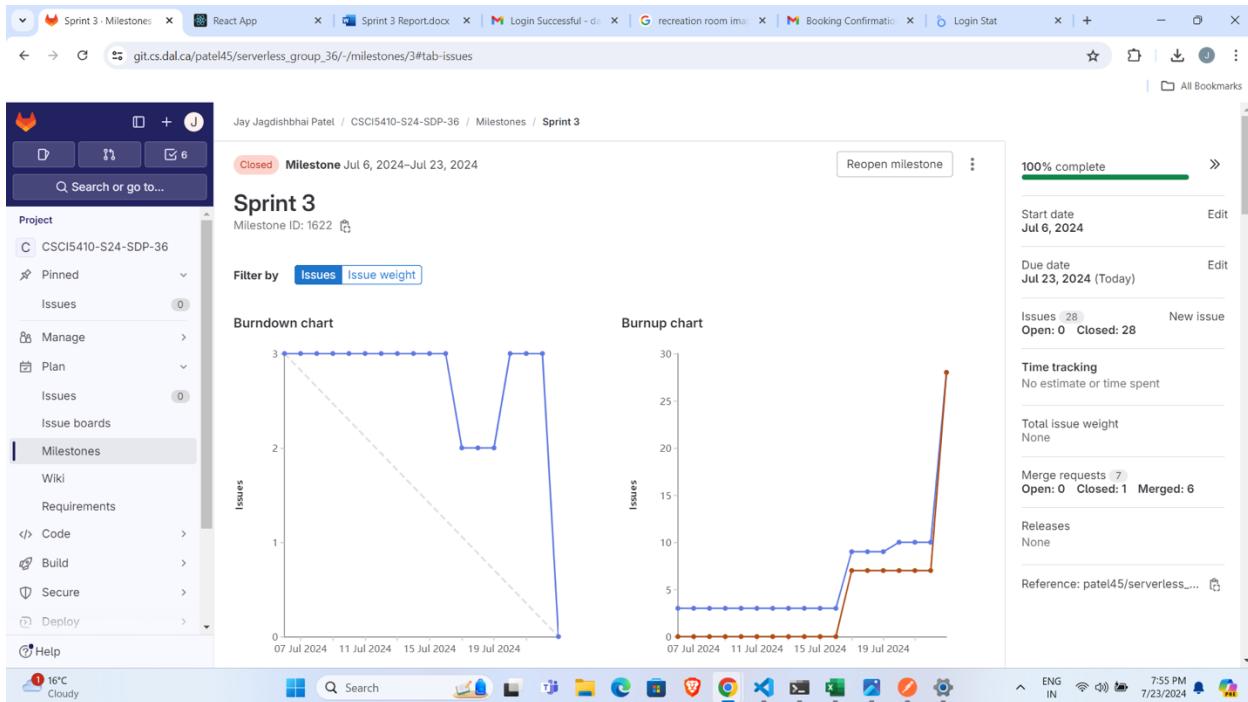


Figure 69: GitLab board and Milestone snapshot displaying Milestone statuses and progress for Sprint 3 of our project.

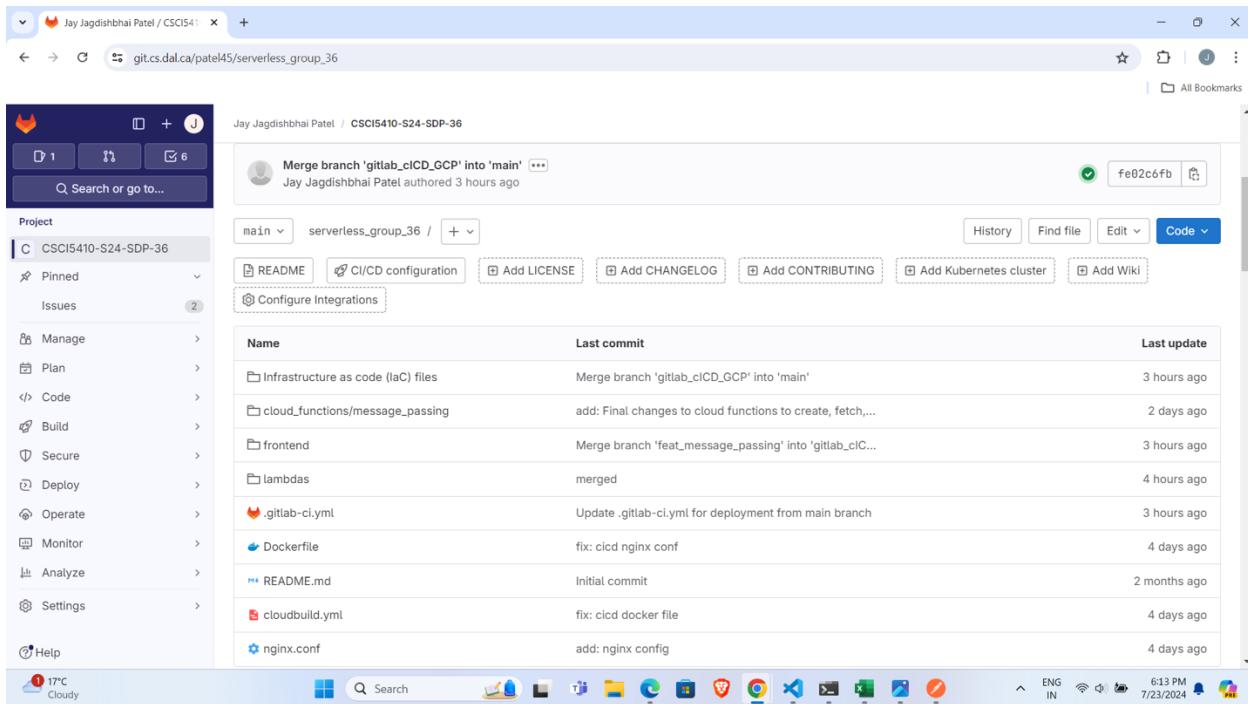


Figure 70: Main branch of our Gitlab Repository and CICD pipeline is working, and application is deployed.

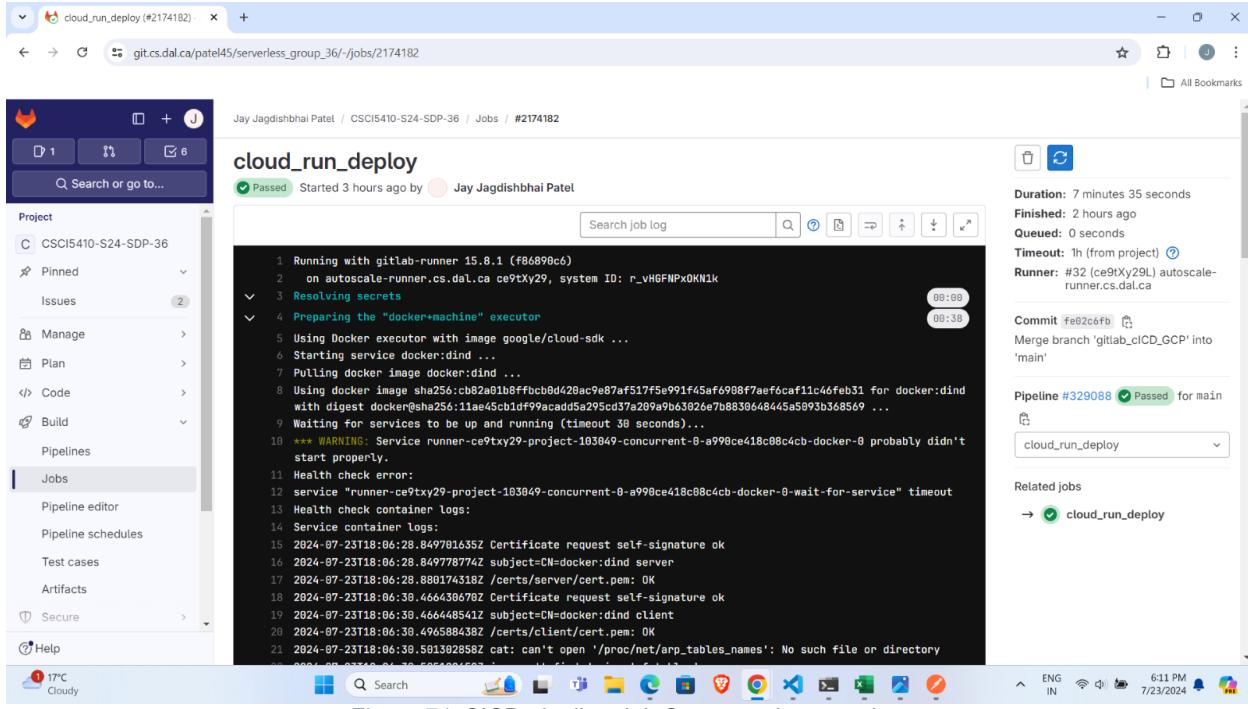


Figure 71: CI/CD pipeline Job Succussed screenshot.

## References

- [1] "Google Cloud Pub/Sub Documentation," Google Cloud. [Online]. Available: <https://cloud.google.com/pubsub/docs/create-topic>. [Accessed: July 23, 2024].
- [2] "Google Cloud Pub/Sub Documentation," Google Cloud. [Online]. Available: <https://cloud.google.com/pubsub/docs/publisher>. [Accessed: July 23, 2024].
- [3] "Calling Cloud Pub/Sub from Google Cloud Functions," Google Cloud. [Online]. Available: <https://cloud.google.com/functions/docs/calling/pubsub#:~:text=If%20you%20are%20deploying%20using%20the%20Google%20Cloud%20console%2C%20you,trigger%20calls%20to%20your%20function>. [Accessed: July 23, 2024].
- [4] "Extend Cloud Firestore with Cloud Functions," Firebase. [Online]. Available: <https://firebase.google.com/docs/firestore/extend-with-functions>. [Accessed: July 23, 2024].
- [5] A. Prasad, "Building a Real-Time Chat App with React.js and Firebase," freeCodeCamp, May 14, 2020. [Online]. Available: <https://www.freecodecamp.org/news/building-a-real-time-chat-app-with-reactjs-and-firebase/>. [Accessed: July 23, 2024].
- [6] "Amazon Simple Notification Service Developer Guide," AWS. [Online]. Available: <https://docs.aws.amazon.com/sns/latest/dg/welcome.html>. [Accessed: July 23, 2024].
- [7] "Amazon Simple Queue Service Developer Guide," AWS. [Online]. Available: <https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html>. [Accessed: July 23, 2024].
- [8] "Using Amazon SNS and SQS as Subscribers," AWS. [Online]. Available: <https://docs.aws.amazon.com/sns/latest/dg/sns-sqs-as-subscriber.html>. [Accessed: July 23, 2024].
- [9] A. Gupta, "Build a Serverless Push and SMS Notification Service on AWS," Medium, March 10, 2020. [Online]. Available: <https://lo0o0p.medium.com/build-a-serverless-push-and-sms-notification-service-on-aws-f92fdbf7ece3>. [Accessed: July 23, 2024].

- [10] "Amazon Cognito Identity SDK for JavaScript," npm. [Online]. Available: <https://www.npmjs.com/package/amazon-cognito-identity-js>. [Accessed: July 23, 2024].
- [11] "Analyzing Sentiment," Google Cloud. [Online]. Available: <https://cloud.google.com/natural-language/docs/analyzing-sentiment>. [Accessed: July 23, 2024].
- [12] "Natural Language Basics," Google Cloud. [Online]. Available: [https://cloud.google.com/natural-language/docs/basics#interpreting\\_sentiment\\_analysis\\_values](https://cloud.google.com/natural-language/docs/basics#interpreting_sentiment_analysis_values). [Accessed: July 23, 2024].
- [13] "Node.js Client for Google Cloud Natural Language," npm. [Online]. Available: <https://www.npmjs.com/package/@google-cloud/language>. [Accessed: July 23, 2024].
- [14] "Cognito User Pool Lambda Trigger," AWS. [Online]. Available: <https://docs.aws.amazon.com/cognito/latest/developerguide/user-pool-lambda-post-authentication.html>. [Accessed: July 23, 2024].
- [15] "Batch Update," Google Sheets API. [Online]. Available: <https://developers.google.com/sheets/api/guides/batchupdate>. [Accessed: July 23, 2024].
- [16] "Embedding in Looker Studio," Google Developers. [Online]. Available: <https://developers.looker.com/embed/getting-started>. [Accessed: July 23, 2024].
- [17] "Looker Studio Help," Google Support. [Online]. Available: <https://support.google.com/looker-studio/answer/7450249?hl=en#zippy=%2Cin-this-article>. [Accessed: July 23, 2024].
- [18] "Deploying Builds on Cloud Run," Google Cloud. [Online]. Available: <https://cloud.google.com/build/docs/deploying-builds/deploy-cloud-run>. [Accessed: July 23, 2024].
- [19] "Building Containers," Google Cloud. [Online]. Available: <https://cloud.google.com/build/docs/building/build-containers>. [Accessed: July 23, 2024].
- [20] "Create a GitLab Pipeline to Push to Google Artifact Registry," GitLab Documentation. [Online]. Available: [https://docs.gitlab.com/ee/tutorials/create\\_gitlab\\_pipeline\\_push\\_to\\_google\\_artifact\\_registry/](https://docs.gitlab.com/ee/tutorials/create_gitlab_pipeline_push_to_google_artifact_registry/). [Accessed: July 23, 2024].

- [21] "Build Your Application," GitLab Documentation. [Online]. Available: [https://docs.gitlab.com/ee/topics/build\\_your\\_application.html](https://docs.gitlab.com/ee/topics/build_your_application.html). [Accessed: July 23, 2024].
- [22] "Flowchart maker & online diagram software," *Diagrams.net*. [Online]. Available: <https://app.diagrams.net/>. [Accessed: July 4, 2024]].
- [23] "Thunder Client," Thunder Client. [Online]. Available: <https://www.thunderclient.com/>. [Accessed: July 1, 2024].