CSCI 5410

Serverless Data Processing

Dal Vacation Home

Group 6: Sprint-3 Report

Group Members:

Nikita Davies	nk548914@dal.ca
Mrunal Mangesh Patkar	mr396180@dal.ca
Rameez Parkar	rameez.parkar@dal.ca
Axata Darji	ax583820@dal.ca

GitLab Link: https://git.cs.dal.ca/ndavies/csci5410-s24-sdp-6

Deployed Application URL: https://dalvac-eawtiwfwgq-uc.a.run.app/

Table of Contents

Details of Research

Message passing:

A publisher application creates and sends messages to a topic. Pub/Sub offers at-least-once message delivery and best-effort ordering to existing subscribers.

The general flow for a publisher application is:

1. Create a message containing your data.
2. Send a request to the Pub/Sub server to publish the message to the specified topic.
3. To get the permissions to publish messages to a topic, the Pub/Sub Publisher (roles/pubsub.publisher) IAM role on topic is required.[1]
4. In Cloud Functions, a Pub/Sub trigger enables a function to be called in response to Pub/Sub messages. When you specify a Pub/Sub trigger for a function, you also specify a Pub/Sub topic. Your function will be called whenever a message is published to the specified topic.
5. A Pub/Sub trigger is implemented as an Cloud Event function, in which the Pub/Sub event data is passed to your function in the Cloud Events format, and the Cloud Event data payload is of type MessagePublishedData.[2]

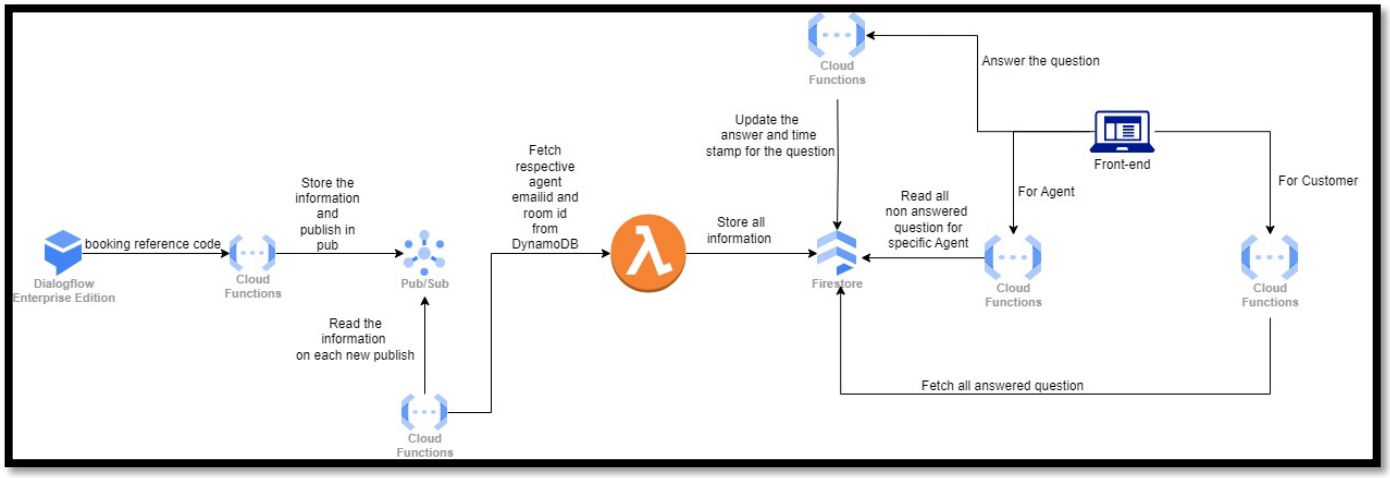

Figure 1: Architecture for Message passing module

Looker Studio:

Looker Studio (formerly known as Google Data Studio) is a powerful tool for data visualization and dashboard creation. It allows users to connect, visualize, and share data from various sources.
Looker Studio Community Connectors enable direct connections from Looker Studio to any internet accessible data source. Community Connectors to create reports and dashboards with data from sources such as:

- Platforms for social media, CRM, search, finance, HR, advertising, etc.
- Public and other open data sets.
- Private company data.

- Any data source or service that can be accessed over the internet using Apps Script.[3]

In Looker Studio, connecting to data involves the following components:

- **Connectors** connect Looker Studio to underlying data. Connecting to data creates a *data source* in Looker Studio.
- **Data sources** represent a particular instance of a connector: for example, a connection to a specific BigQuery table or query, a Google Analytics property, or a Google Sheet. Data sources let you configure the fields and options provided by the connector used to create that connection instance. In addition, the data source gives a secure way to share information and insights with report viewers who may not be able to directly access the underlying data.

- **Credentials** determine who can access the data provided by a data source.[4]

  We have used Google Cloud storage as Data source.

Dialogflow

A Dialogflow CX agent is a virtual agent that handles concurrent conversations with end-users. It is a natural language understanding module that understands the nuances of human language. Dialogflow translates end-user text or audio during a conversation to structured data that apps and services can understand. Design and build a Dialogflow agent to handle the types of conversations required for the system.

A Dialogflow agent is similar to a human call center agent. Train them both to handle expected conversation scenarios, and training does not need to be overly explicit. Flows are used to define these topics and the associated conversational paths. Every agent has one flow called the Default Start Flow. This single flow may be all you need for a simple agent. More complicated agents may require additional flows, and different development team members can be responsible for building and maintaining these flows.[5]

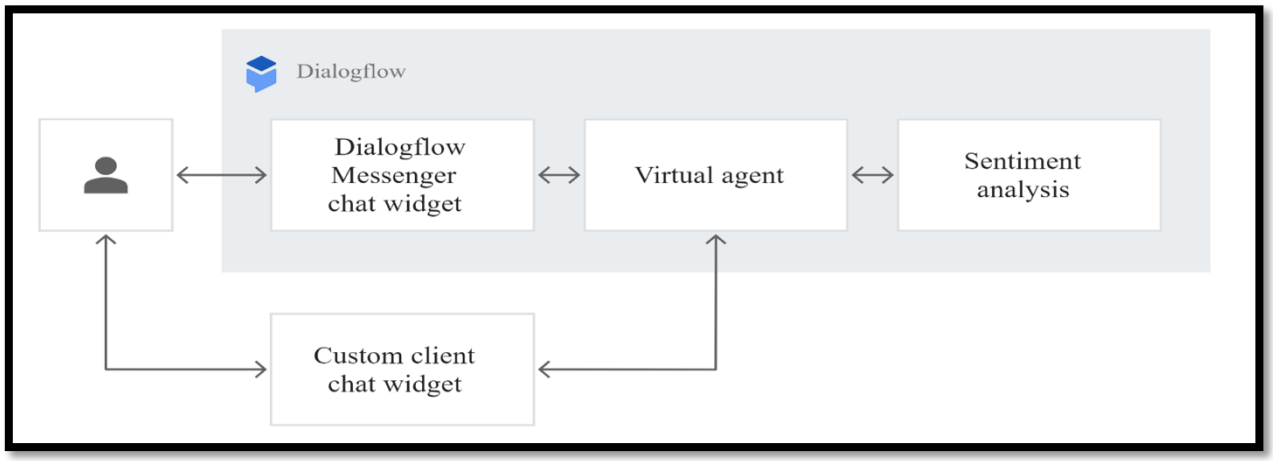

Figure 2: Dialogflow flowchart

Google Cloud Run

What is Google Cloud Run:

Google Cloud Run is a fully managed platform that scales stateless containers automatically. It enables developers to deploy containerized applications in a serverless environment, removing the need for server management. Cloud Run supports any language or framework and integrates seamlessly with other Google Cloud services. Features include custom domains, automatic SSL, and detailed monitoring [6].

How to Use the Cloud Run Console:

The Cloud Run console allows developers to deploy and manage services, configure traffic management, and monitor performance. It simplifies scaling, setting environment variables, and integrating with CI/CD pipelines. The console enhances productivity and operational efficiency [7].

Ultimately, Google Cloud Run provides a flexible and scalable platform for deploying containerized applications, with seamless integration and robust security features.

AWS CloudFormation

What is AWS CloudFormation:

AWS CloudFormation is an infrastructure-as-code service that simplifies provisioning and managing AWS resources using templates. It supports a wide range of AWS services and integrates with AWS management and monitoring tools [8].

How to Use the CloudFormation Console:

The CloudFormation console is used to create, manage, and monitor stacks. Developers can upload templates, launch stacks, and track resource provisioning status, facilitating troubleshooting and ensuring consistent deployments [9].

In conclusion, AWS CloudFormation provides a robust way to manage AWS infrastructure as code, with powerful templating capabilities and seamless integration with AWS tools.

System Architecture

Final Architecture

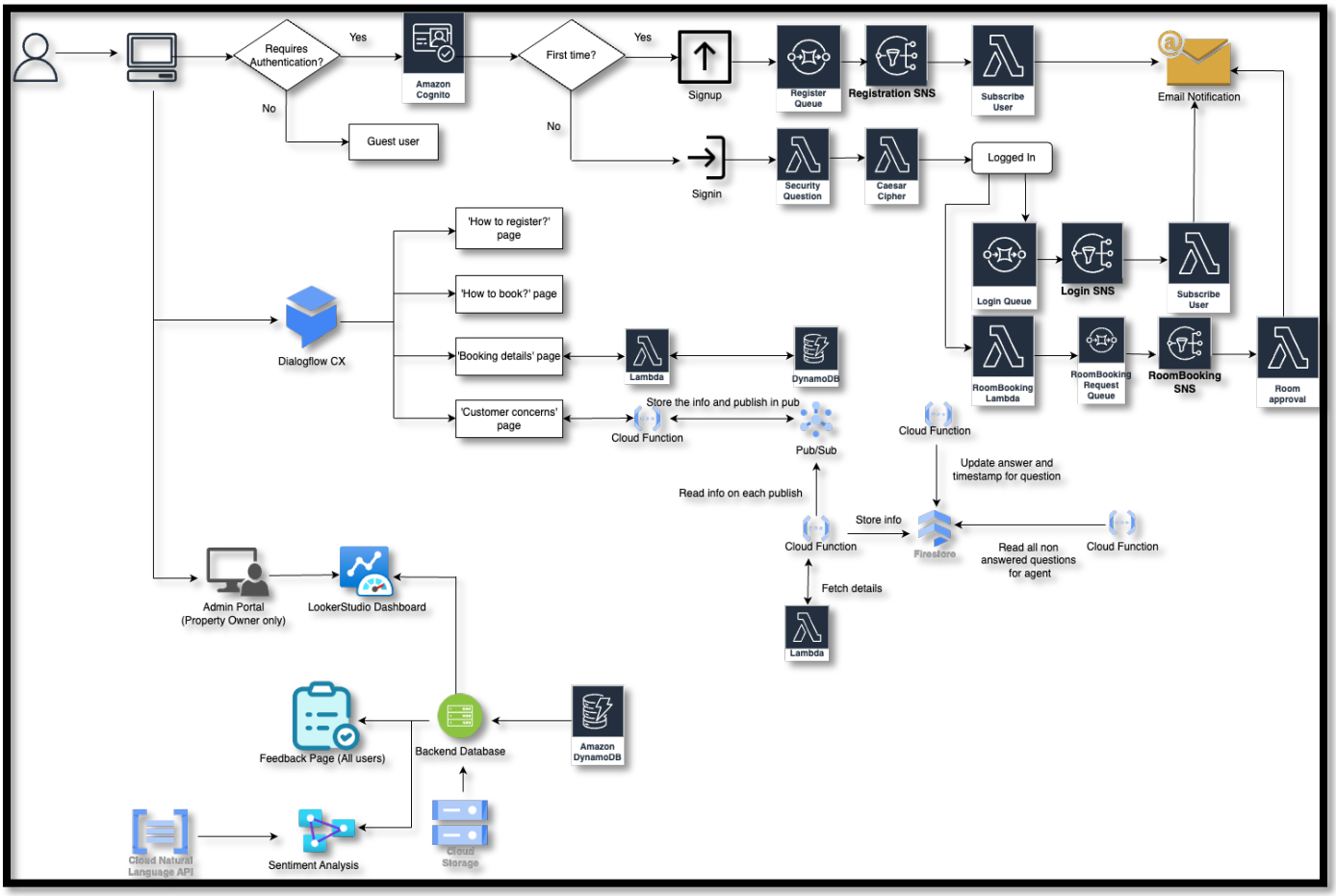

Figure 3: Final Architecture

Pseudocode/Algorithm for important modules

Algorithm for AWS Cognito authentication (sign up and sign in):

1. **User Registration Process:**

   - **Front-End App:** User submits the registration form with details such as email, password, name, security question, security answer, cipher key, and other attributes.
   - **API Call:** The front-end app sends a request to AWS Cognito for user sign-up with the provided details.
   - **Example Request Data:**

```
{
  "ClientID": "userid",
  "Username": "mmp3299@hotmail.com",
```

```
    "Password": "password@3",
    "UserAttributes": [
        {"Name": "email", "Value": "mmp3299@hotmail.com"},
        {"Name": "custom:name", "Value": "Userame"},
        {"Name": "custom:security_question", "Value": "q1"},
        {"Name": "custom:security_answer", "Value": "answer"},
        {"Name": "custom:cipher_key", "Value": "3"},
        {"Name": "custom:user_group", "Value": "RegisteredCustomers"}
      ]
}
```

2. **User Sign-Up Verification**

   - **User Sign-Up Verification Process:**
     - **Front-End App:** User submits the verification code received via email.
     - **API Call:** The front-end app sends a request to AWS Cognito to confirm the user sign-up with the verification code.
   - **Example Request Data:**

```
{
    "ClientId": "userID",
    "Username": "mmp3299@hotmail.com",
    "ConfirmationCode": "123456"
}
```

3. **Post Confirmation Lambda Function**

   - **Trigger:** AWS Cognito triggers the Post Confirmation Lambda function after successful user sign-up confirmation.
   - **Lambda Function Logic:**
     - **Extracting User Attributes:**
       - Retrieve the user attributes from the event.
       - Identify the user group from the custom attribute (`custom:user_group`).
   - **Assign User to Group:** Assigning The user to the appropriate group in Cognito. (Registered Customer or property agents)
   - **Store Additional User Data in DynamoDB:** Store the data, including email, name, security question, security answer, cipher key, and other attributes.

4. **User Sign-In**
   - **User Sign-In Process:**
     - **Front-End App:** User submits the sign-in form with email and password.
     - **API Call:** The front-end app sends a request to AWS Cognito to authenticate the user with the provided email and password.
   - **Example Request Data:**

```
{
    "AuthFlow": "USER_PASSWORD_AUTH",
    "ClientId": "example_client_id",
    "AuthParameters": {
      "USERNAME": "user@example.com",
      "PASSWORD": "password123"
}
```

Algorithm for Security Question and answer:

1. During registration, user is displayed a dropdown menu of security questions.
2. User selects one of the security questions from the list.
3. User enters the answer for the security question in a textbox.
4. This information along with other registration data is inserted in DynamoDB.
5. During signin, after user passes the correct credentials for email and password, they are presented with security question.
6. User enters the answer to the security question in the text box.
7. This user email and security question's answer is passed as an API through API Gateway to the lambda.
8. The lambda then fetches the answer entered by the user from DynamoDB based on the email.
9. The lambda then verifies the correct answer with the answer sent during login.
10. If the answer is correct user is then taken to the third stage of authentication, i.e. Caesar cipher, else authentication fails.

Algorithm for Caesar Cipher:

1. During registration, user is asked to enter a cipher key which is a numeric value.
2. This information along with other registration data is inserted in DynamoDB UserDetails table.
3. During signin, after user passes the correct credentials for email and password, and answers correctly for the security question, they are presented with the encoded string which the user must decode using the Caesar cipher key entered during registration.
4. After user enters the decoded string, the user email, encoded and decoded string are passed as an API through API Gateway to lambda.
5. The lambda then checks if the decoded value is correct by fetching the key from DynamoDB and decoding the encoded string and comparing both these decoded values.
6. If the answer is correct, user is then successfully signed in since all three factors of authentication are successfully executed.

Algorithm for notifying user upon successful login/register:

1. **User Registration/Login**
   - **Front-End App:** User submits registration or login form.
   - **API Call:** The front-end app sends a request to the backend API with user details (e.g., email ID) for registration or login.
2. **API Call to Lambda**
   - **API Gateway:** The API Gateway receives the request and triggers the Lambda function with the user details.
   - **Example Request Data:**

     ```
     {
         "email": "user@example.com",
     }
     ```
   - **Lambda Function (Submit to SQS):** The Lambda function processes the request and sends a message to an SQS queue.
3. **SQS Queue to SNS Topic**
   - **SQS Queue:** The message from the Lambda function is placed in the SQS queue.
   - **SNS Topic Subscription:** The SQS queue is subscribed to the SNS topic. When a new message arrives in the SQS queue, it triggers the SNS topic.
4. **Lambda for Subscription**
   - **Lambda Function (Subscribe User)**

- **Lambda Function 2:** The SNS topic triggers another Lambda function.
- **Retrieve Message:** Extract the email and event from the SQS message.
- **Check Subscription:** Check if the email is already subscribed to the SNS topic.
- **If not subscribed:** Subscribe the user to the SNS topic.
- **Publish Notification:** Publish a notification message to the SNS topic about the successful login or registration.

5. **Email Notification via SNS Topic**
    - **SNS Topic (Send Email):** SNS topic sends the notification message to all subscribers, including the email address provided.

Algorithm for add/update room – property agents:

1. After logging in, the property agent can add or update details of the room.
2. To add a new room, the user must enter details like roomType, capacity, price, furnishedType, additionalFeature, discountCode.
3. This data is passed to add-new-room lambda via API Gateway. In addition to the above data, details like roomId, feedbackId, PolarityOfFeedback and propertyAgentId are also added to the DynamoDB rooms table.

Algorithm for room booking and notifications:

1. Room Booking

- **Front-End App:** User clicks on the book room button.
- **API Call:** The front-end app sends a request to the backend API with the required room details needed for booking.

2. API Call to Lambda

- **API Gateway**
- **API Gateway:** The API Gateway receives the request and triggers the Lambda function named room-booking.
- **Example Request Data:**

```
{
  "userId": "2",
  "roomId": "room2208",
  "bookingStartDate": "2024-07-15",
  "bookingEndDate": "2024-07-21",
  "totalBookingDays": 5
}
```

- **Lambda Function (RoomBooking) logic:** This Lambda function processes the request to book the room by the user and sends a message to an SQS queue.

3. SQS Queue

- **SQS Queue:** The message from the Lambda function is placed in the SQS queue. The SQS queue is added as trigger for the invokation of the second lambda function roomApproval.

4. Lambda Function (RoomApproval)

- **Lambda Function (roomApproval) logic:** This Lambda function receives the message pushed to the SQS queue and it checks the roomId, bookingStartDate and bookingEndDate to check whether the requested room is available for the given dates. If the room is available, then the booking is confirmed, the booking details is added to the **DynamoDB table registration** and a confirmation email notification is send to user through SNS. If the room is not available, then a rejection email notification is emailed to the user.

5. Confirmation/Rejection Email Notification via SNS Topic

- **SNS Topic (RoomBookingRequest):** The SNS topic sends the notification message to the user notifying the confirmation/rejection of the requested room booking.

Algorithm for message passing module:

1. User Interaction in Dialogflow

- **User Input**: User enters a question along with a booking reference code in Dialogflow.
- **Dialogflow Intent**: Dialogflow detects the intent and extracts the question and booking reference code.

2. Publish Message to Pub/Sub

- **Cloud Function Trigger**: A Cloud Function is triggered by the Dialogflow intent.
- **Message Publishing**:
  - The Cloud Function formats the extracted question and booking reference code into a message.
  - It publishes the message to a specific Pub/Sub topic

3. Trigger Cloud Function by Pub/Sub

1. **Pub/Sub Trigger**: Another Cloud Function is triggered by the Pub/Sub message.
2. **Call AWS Lambda**:
   - The Cloud Function extracts the booking reference code from the message.
   - It calls an AWS Lambda function, passing the booking reference code.
3. **AWS Lambda Execution**:
   - The AWS Lambda function queries DynamoDB to fetch the agentID and roomID based on the booking reference code.
   - The Lambda function returns the agentID and roomID.

4. **Store Data in Firestore**

   - The Cloud Function receives the agentID and roomID from AWS Lambda.
   - It combines this data with the original question and booking reference code.
   - The Cloud Function stores this combined data along with timestamp in a Firestore collection.

5. **Agent Logs in and Fetches Concerns**

1. **Agent Login**: When an agent logs in, a Cloud Function is triggered.
2. **Fetch Concerns**:
   - The Cloud Function queries Firestore for records matching the agentID with an answer field set to null.
   - It returns these records to the agent's front end.

6. **Agent Answers the Question**

1. **Agent Answer Submission**: The agent submits an answer through the front end.
2. **Update Firestore**:
    - A Cloud Function is triggered by the agent's submission.
    - It updates the corresponding Firestore record with the provided answer.

7. **User Logs In and Fetches Answers**

1. **User Login**: When the user logs in, a Cloud Function is triggered.
2. **Fetch Answers**:
    - The Cloud Function queries Firestore for records with the user's booking reference code where the answer field is not null.
    - It returns these records to the user's front end.

Algorithm for LookerStudio module:

1: **User Interaction and Login**

- User Input: User logs in using Cognito.
- Capture Login Details: Cognito captures and stores the login details.

2: **Store Login Details in DynamoDB**

- Trigger Lambda (Logging): On user login, a Lambda function called logging is triggered.
- Store in DynamoDB: Lambda function logs the login details in login_statistics table.

3: **Export Data to CSV**

- DynamoDB Trigger: when a row of data is inserted in the table login_statistics, a lambda named uploadCSV is configured to be triggered
- Lambda (UploadCSV): Exports the rows of the table login_statistics to a CSV file.

4: **Store CSV in GCP Cloud Storage**

Upload to Cloud Storage: Lambda function uploads the CSV file to a GCP Cloud Storage bucket.

5: **Configure LookerStudio for Analytics**

- Import Data: LookerStudio imports data from the CSV file in GCP Cloud Storage.
- Visualize Data: Fields and styles are configured in LookerStudio to:
    - Display user login data as a bar chart.
    - Show total number of users.

Algorithm for analyzing feedback using Google Natural Language API

1. **View Feedbacks**

- **Front-End App:** User clicks on the view reviews button.
- **API Call:** The front-end app sends a request to the backend API with the required roomId to fetch all the feedbacks added by the users for that particular room. The feedbacks are listed in frontend.

2. **API Call to Lambda Function (getFeedbacks)**

- **API Gateway:** The API Gateway receives the request and triggers the Lambda function named room-booking

- **Example Request Data:**

```
{
  "roomId": "room2208",
}
```

- **Lambda Function (getFeedbacks) logic:** This Lambda function accepts the roomId and returns all the feedbacks received for that room.

3. **Add Feedbacks**

- **Front-End App:** User clicks on the add feedback button.
- **API Call:** The front-end app sends a request to the backend API with the feedback details.

4. **API Call to Lambda Function (addFeedbacks)**

- **API Gateway:** The API Gateway receives the request and triggers the Lambda function named addFeedbacks.
- **Example Request Data:**

```
{
    "roomId": "room2208",
    "roomType": "room",
    "feedback": "Wondeful experience and great room",
    "rating": 4,
    "userId": "1"
}
```

1. **Lambda Function (addFeedbacks) logic**

- This Lambda function accepts the feedback details and adds it to the **dynamoDB Table feedback.**
- **Google Natural Language API Invocation**
- In the addFeedback lambda function, the google natural language API is called which analyse the feedback provided and adds a sentiment score and sentiment magnitude to the feedback row
- For using the Google Natural Language API in lambda function, a service account was created in GCP and the JSON Key was downloaded.
- The downloaded json key was added as secret in AWS Secret Key Manager to store it securely when used in the lambda function.
- An environment variable GOOGLE_APPLICATION_CREDENTIALS was created to fetch and verify the GCP credentials.

2. **Lambda Function (getFeedbackPolarity)**

- This Lambda function fetches all the feedbacks sentiment scores for a particular room and calculates the average score to the polarity of the feedback.
- The individual feedbacks, their score and the overall feedback polarity is displayed in the frontend for all the users.

Algorithm for fetching Booking details using a booking code:

1.Front-End App: User submits a request to retrieve booking details using a booking reference code.
2. API Call: The front-end app sends a request to the AWS API Gateway, which triggers the Lambda function. Example Request Data:

```
{
  "bookingCode": "102"
}
```

3. Lambda Function Execution

- **Initialize DynamoDB Resource:**
  - The boto3 library is used to initialize a connection to the DynamoDB service.
  - Define the table names for users, rooms, and bookings.

- **Retrieve Booking Details:**
  - **Fetch Booking Details:**
    - Access the Booking table using the booking reference code provided in the event.
    - Retrieve booking details based on the booking code.
- **Handle Invalid Booking Code:** If the booking code is invalid (i.e., no such code found in the Booking table), return an error message indicating the invalid booking code.

4. Retrieve User Details:

- **Fetch User Details:**
  - At start, extract the userId from the booking details, then access the UserDetails table using the userId.
  - Retrieve user details based on the userId.
- **Handle Invalid User:**
  - If the user is not found (i.e., no relevant data in the UserDetails table), return an error message indicating the user not found.

5. Retrieve Room Details:

- **Fetch Room Details:**
  - Extract the roomId from the booking details.
  - Access the Rooms table using the roomId.
  - Retrieve room details based on the roomId.
- **Handle Invalid Room:**
  - If the room is not found (i.e., no roomID found in the Rooms table), return an error message indicating the room not found.

6. Construct and Return Result:

- **Construct Result:**
  - Combine the booking details, user details, and room details into a single result object.

- Extract relevant fields such as booking start date, booking end date, total booking days, status, room type, capacity, price, furnished type, username, email, and address.

- **Return Result:**
  - Return the constructed result as the response.

Algorithm for Virtual Assistant module:

1. The user clicks on the Dialogflow chat button on bottom right corner of the screen.
2. After the chat screen opens up, user mentions the intent for the chat.
3. Our DialogFlow CX chatbot has been set up with the following 5 intents in mind:
   - **HowToRegister intent:**
     1. User can ask question like “How to Register?” or “How to sign up?”
     2. The DialogFlow CX identifies the intent in the flow and returns the response, i.e. the steps to signup to the application.
   - **HowToBookRoom intent:**
     1. User can ask question like “How do I book a room?” or “room booking”.
     2. The DialogFlow CX identifies the intent in the flow and returns the response, i.e. the steps to book a room in the application.
   - **BookingDetails intent:**
     1. User can trigger the intent with statements like “Booking details” or “I want details about my booking”.
     2. The DialogFlow CX identifies the intent in the flow and asks the user for their booking reference code.
     3. This booking reference code is then stored in the parameter.
     4. This value is then passed to the lambda via the Dialogflow CX webhook which fetches the data from DynamoDB.
     5. The response is then returned to the user.
   - **CustomerConcerns intent:**
     1. User can trigger the intent with statements like “concern” or “I have a query regarding my booking”.
     2. The DialogFlow CX identifies the intent in the flow and asks the user for their email id, booking reference code and their question.
     3. These values are then stored in the respective parameters.
     4. This value are then passed to the cloud function via the Dialogflow CX webhook which passes this data to the Pub/Sub module.
     5. The cloud function returns the ticket id which the user can then use later to follow up.
   - **CheckTicketResolutionStatus intent:**
     1. User can trigger the intent with statements like “check ticket resolution status”.
     2. The DialogFlow CX identifies the intent in the flow and asks the user for their ticket id.
     3. This ticket id is then stored in the parameter.
     4. This value is then passed to the cloud function via the Dialogflow CX webhook which fetches the data from DynamoDB.
     5. The response is then returned to the user.

Algorithm for Cloud run deployment:

1. Build the React application for production, the Dockerfile and nginx.conf are configured to use Nginx to serve the React app on port 8080.
2. The Docker image is built using the configured Dockerfile and nginx.conf.
3. The Docker image is pushed to Google Container Registry after authenticating Docker with Google Cloud.
4. Necessary APIs are enabled and permissions are granted for the user in Google Cloud.
5. The Docker image is deployed to Google Cloud Run, specifying the appropriate region and allowing unauthenticated access.
6. The deployment process is monitored through Google Cloud Console or CLI logs, and the service is verified to be running correctly and accessible via the provided URL.

To automate service deployment, we have created **CloudFormation scripts** for deploying all AWS services such as Cognito, all Lambda functions, Register and Login SQS-SNS, Room Booking SQS-SNS, and DynamoDB. We used **GCP Deployment Manager** scripts to automate the GCP Pub/Sub module. The application is deployed on Cloud Run.

Deployed URL: https://dalvac-eawtiwfwgq-uc.a.run.app/

Testcases and Evidence of Testing completed modules

Table 1: List of test cases

S.no	Test Case	Result
1.	Guest Users, Registered Customers and Property agents can access the deployed Dashboard of DALVacationHome application	Pass
2.	Guest Users, Registered Customers and Property agents can view all the available rooms	Pass
3.	Guest Users, Registered Customers and Property agents can view the details of a particular room	Pass
4.	Users can successfully sign up to the application	Pass
5.	Users get a verification code in their registered email to sign up	Pass
6.	On successful sign up, users get a welcome email notification	Pass
7.	Verified users get sorted into their respective user groups in the backend	Pass
8.	Registered Customers and Property Agents can successfully sign-in to the application	Pass
9.	2FA authentication is successfully triggered during sign-in	Pass
10.	3FA authentication is successfully triggered during sign-in after 2FA	Pass
11.	On successful sign-in, users receive a email notification	Pass
12.	Auth token and user details gets stored in the local storage for session management	Pass
13.	Auth token and user details should get deleted from the local storage once the user clicks on the sign-out button	Pass
14.	Log table gets populated once a user signs in successfully (log	Pass

	details to be used for analytics)	
15.	If the user signs up as a property agent then they get redirected to the Property agent dashboard	Pass
16.	Property agents can view the rooms created by them	Pass
17.	Property agents can create rooms successfully	Pass
18.	Property agents can edit rooms successfully	Pass
19.	Guest Users, Registered Customers and Property Agents can view reviews given for a particular	Pass
20.	Registered Customers and Property Agents can add review for a particular room.	Pass
21.	Guest Users, Registered Customers and Property Agents can view the overall polarity of reviews for a particular room.	Pass
22.	Property Agents can see Login statistics and total users in Admin Dashboard page using LookerStudio.	Pass
23.	Guest Users, Registered Customers and Property Agents can ask chatbot how to register and chatbot prompty responds to it	Pass
24.	Guest Users, Registered Customers and Property Agents can ask chatbot how to book a room and chatbot prompty responds to it.	Pass
25.	Guest Users, Registered Customers and Property Agents can ask chatbot for booking details and chatbot prompty responds to it.	Pass
26.	Users can ask chatbot for concerns and chatbot prompty responds to it.	Pass
27.	User asks status of the submitted concern to the chatbot and chatbot promptly responds	Pass
28.	User is able to initiate a ticket through Chatbot	Pass
29.	Property Agent can view the raised concerns on customer concern tab	Pass
30.	Property Agent answers the raised query	Pass
31.	Registered Customers can see answers to their raised concerns	Pass
32.	Registered Customers can successfully submit a room booking request	Pass
33.	On successful booking, user gets a booking confirmation to their registered email	Pass
34.	User gets a booking rejection to their registered email on rejection of booking request.	Pass
35.	Application is successfully deployed using Cloud Run	Pass

Dashboard

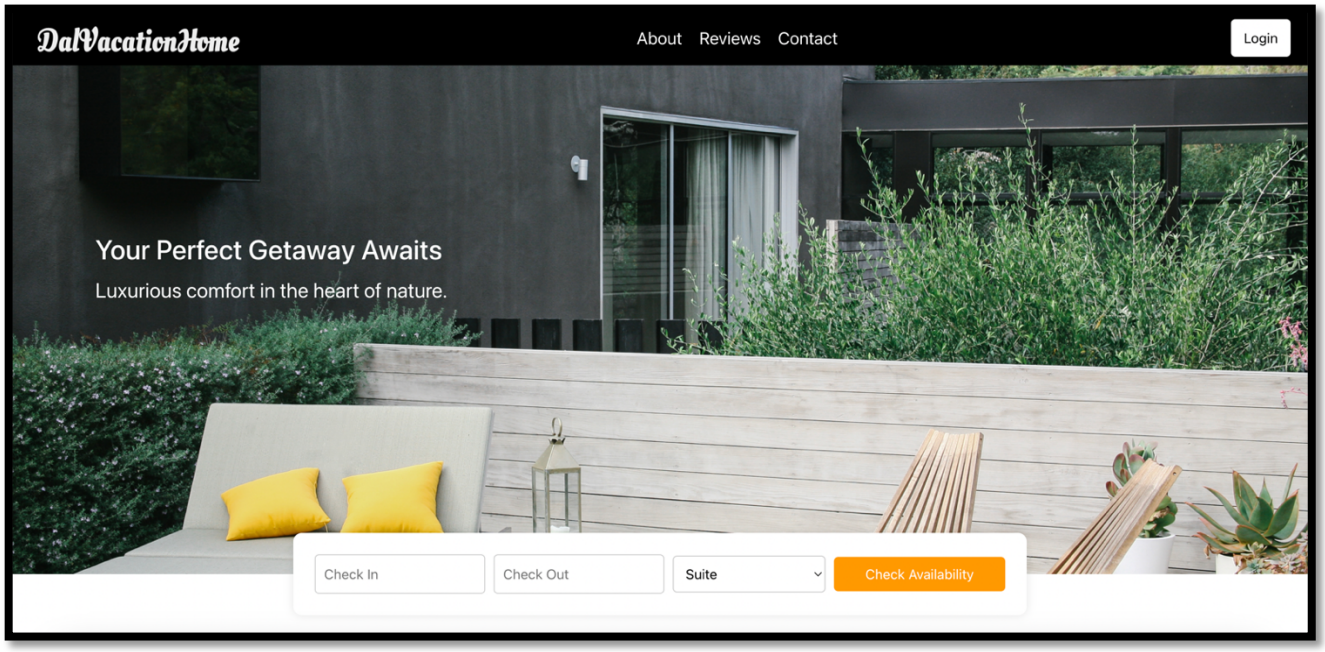

Figure 4: Dashboard for DALVacationHome Application

Rooms List and API integration

Testcase 1: Successful fetching of list of available rooms

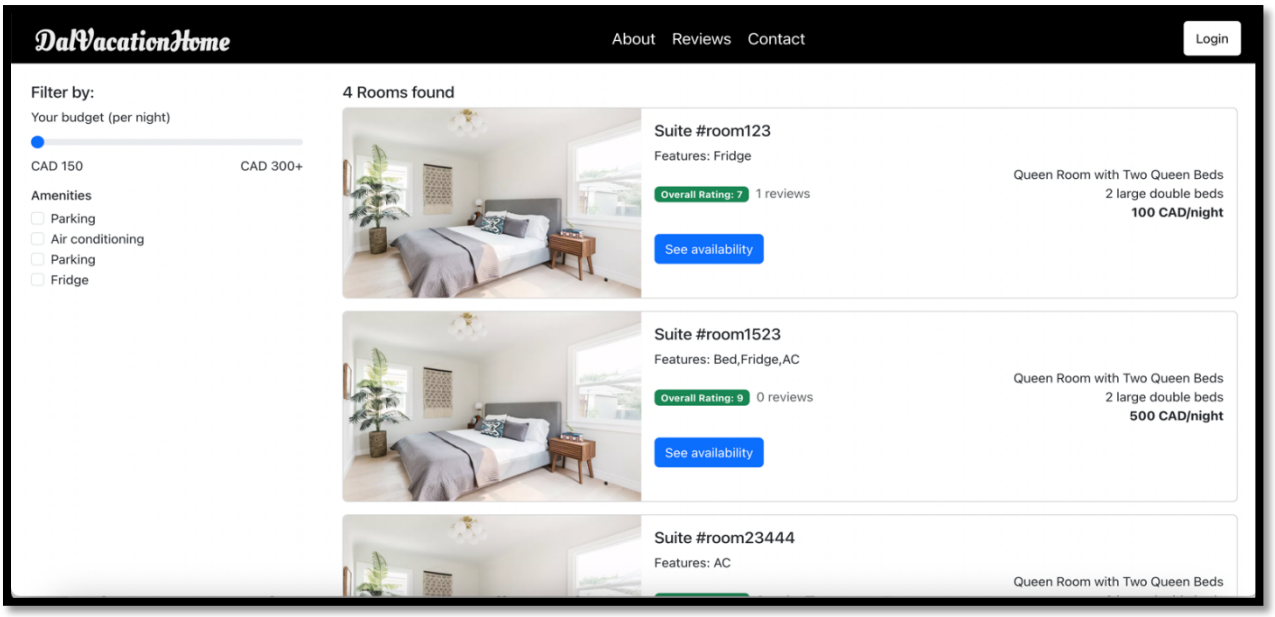

Figure 5: Available Rooms List UI

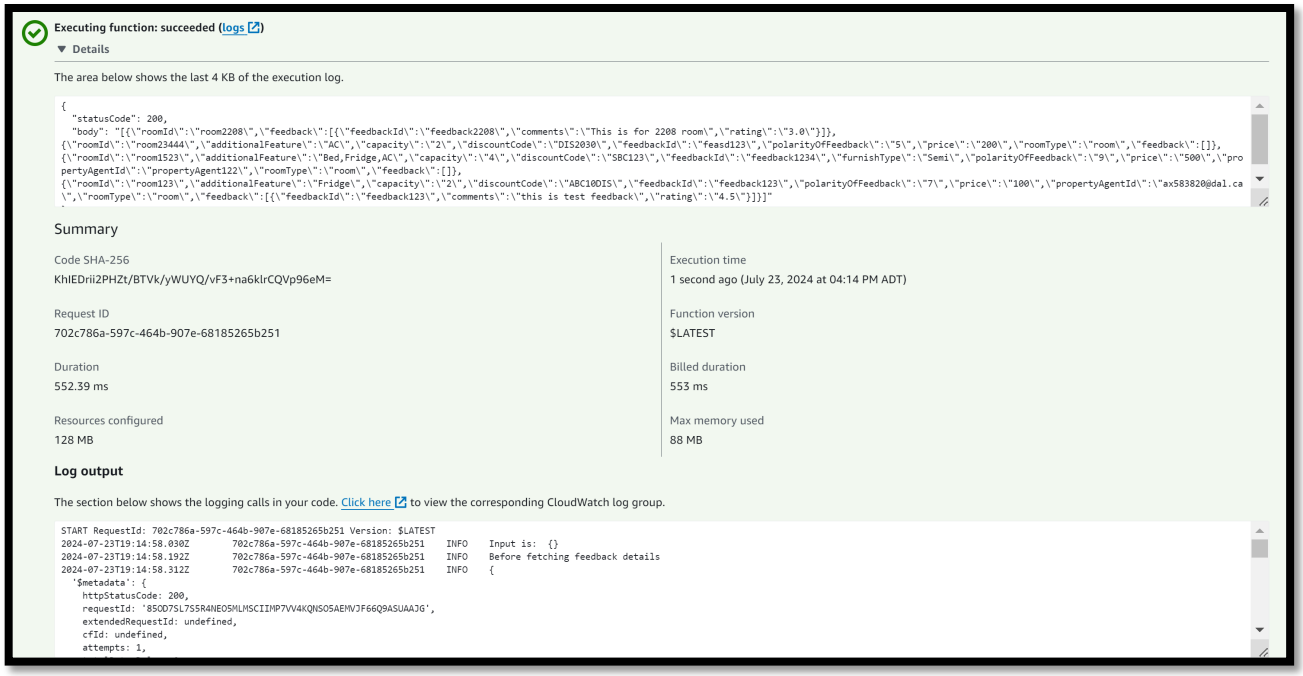

Figure 6: Lambda execution for fetching list of all available properties

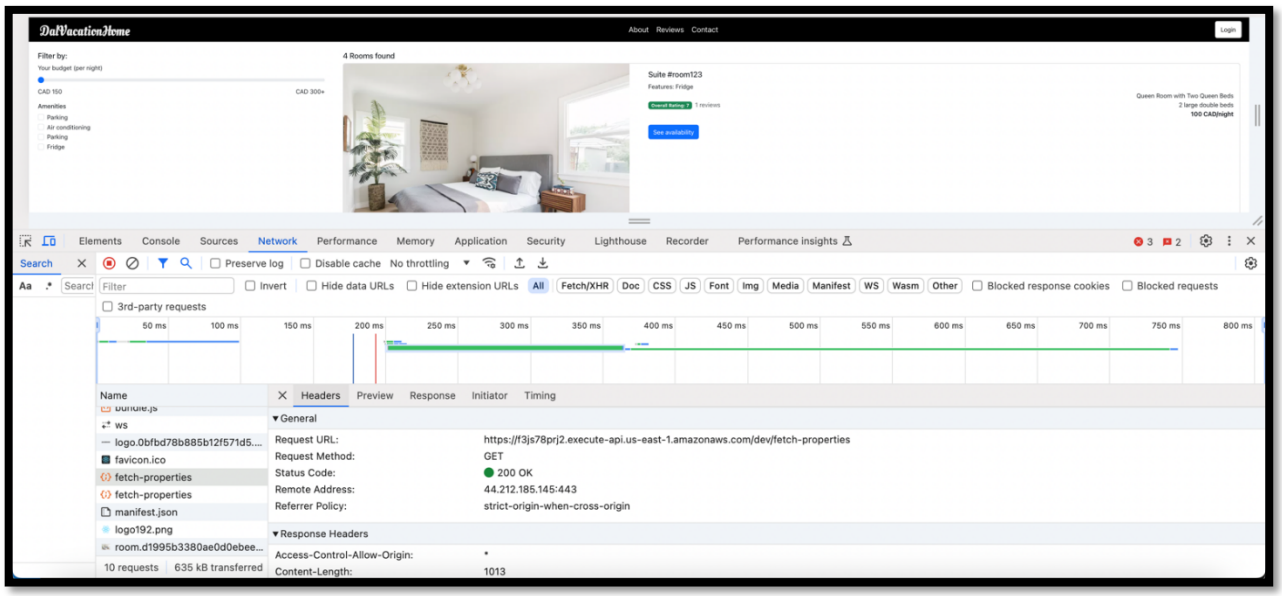

Figure 7: API integration of fetch properties

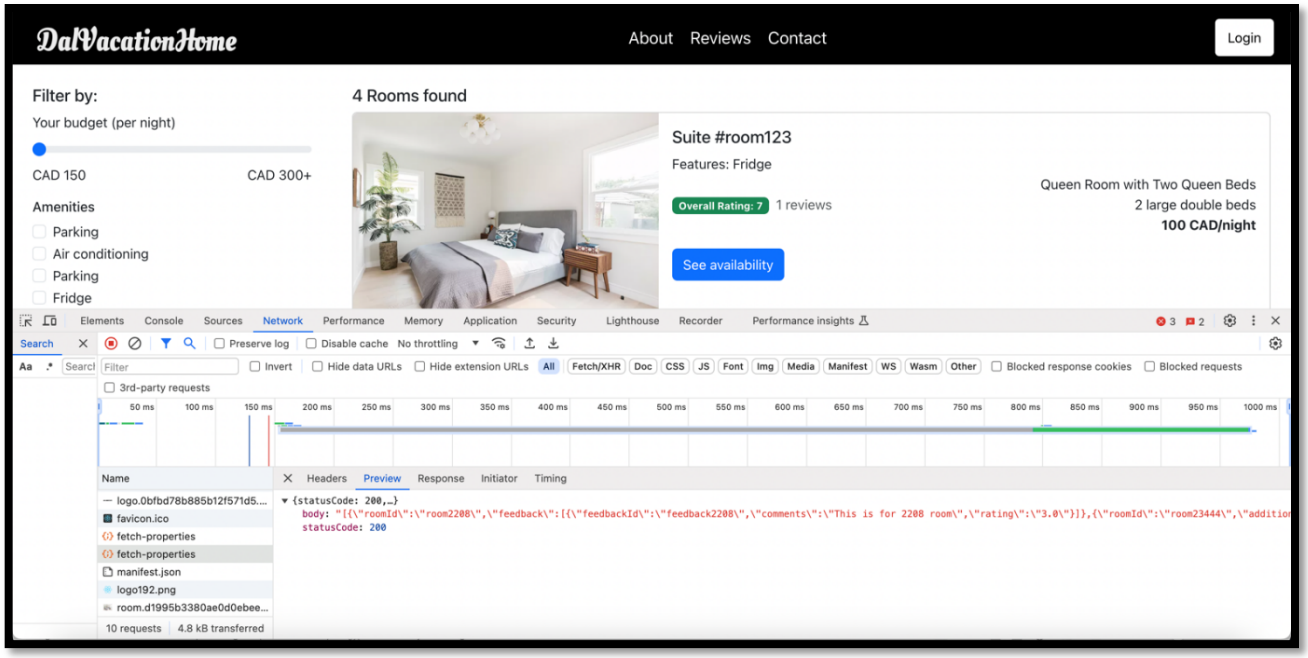

Figure 8: Response of fetchProperties API

Testcase 2: Successful fetching of room details

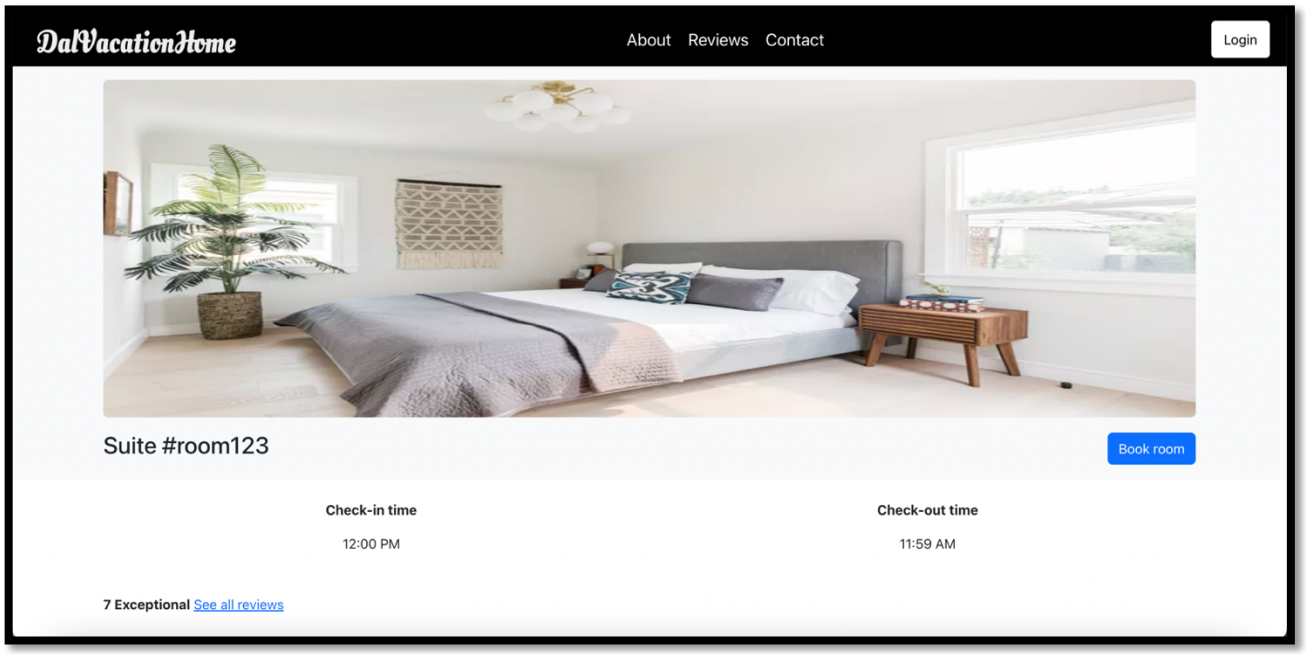

Figure 9: Room Details UI

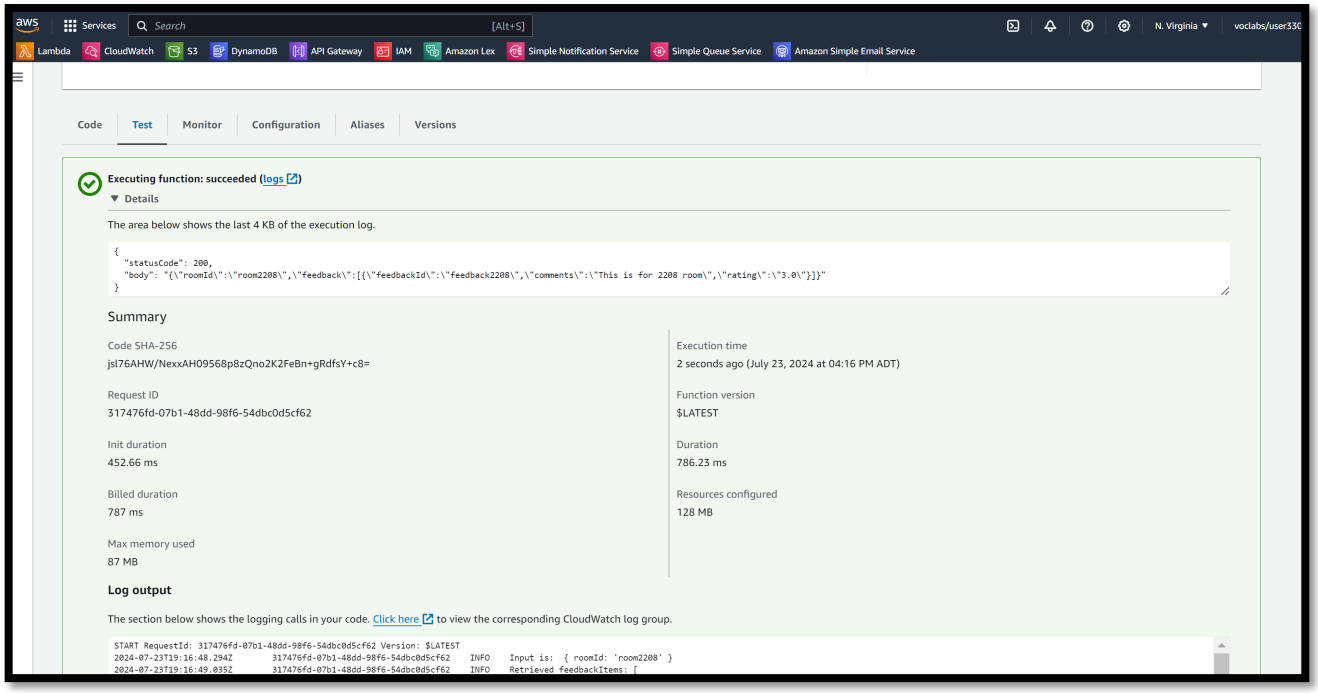

Figure 10: Lambda execution for fetching single property detail

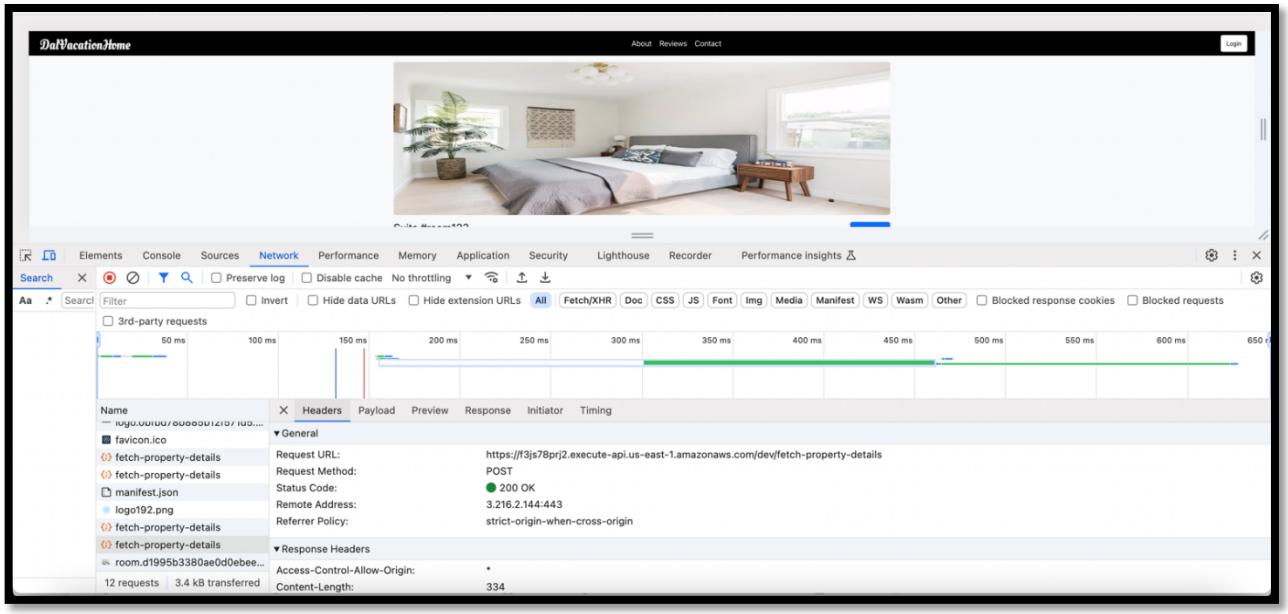

Figure 11: API integration of fetch properties details API

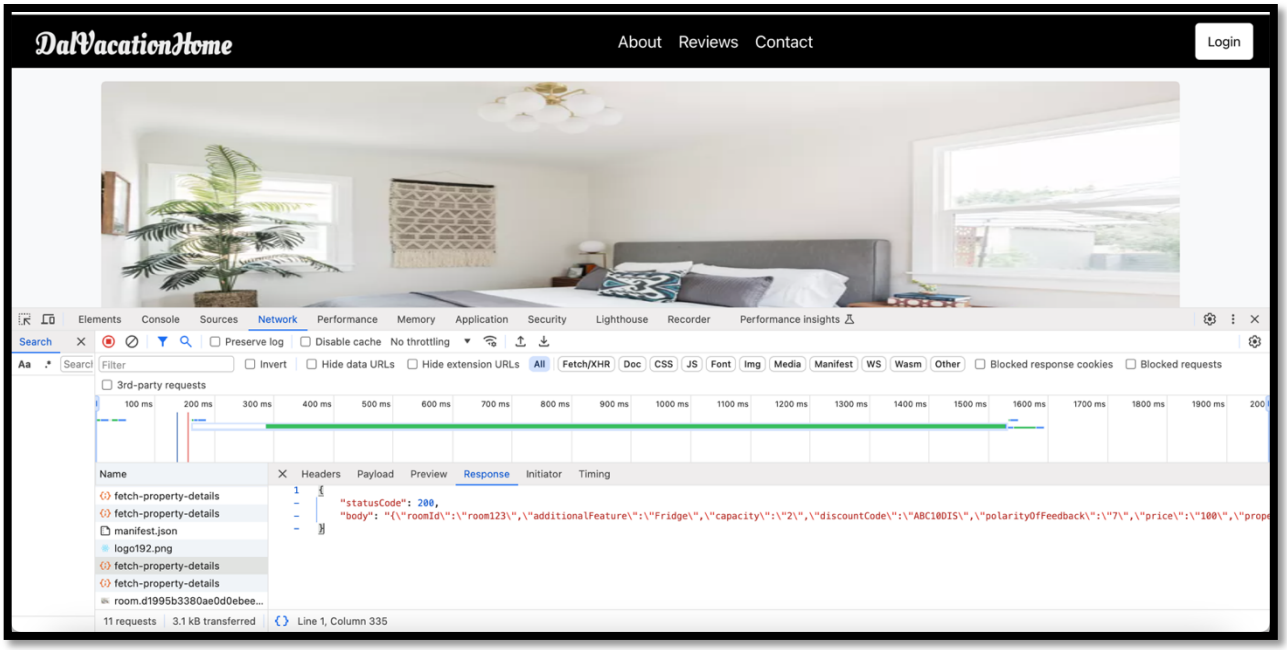

Figure 12: Response of fetch property details API

Create/Edit Rooms by Property Agent

Testcase 1: Create Room by Property Agent

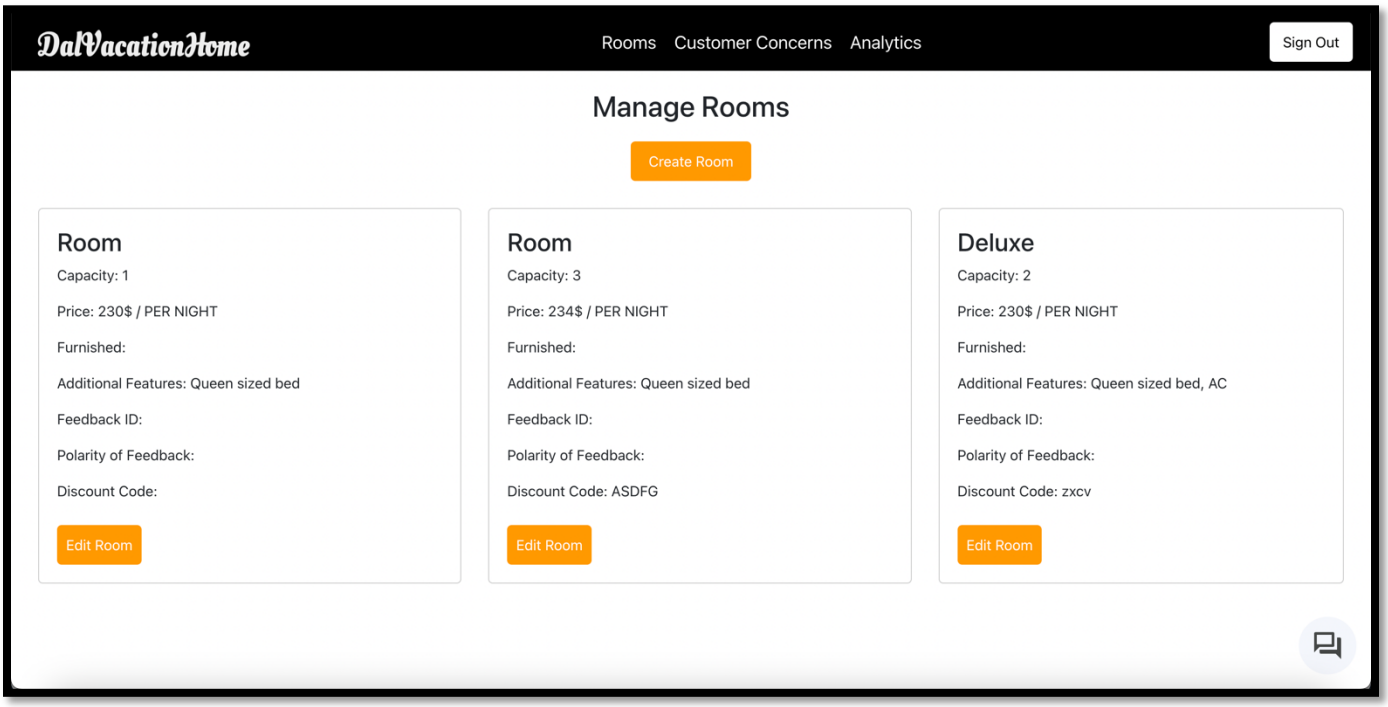

Figure 13: Rooms Added by a Property Agent

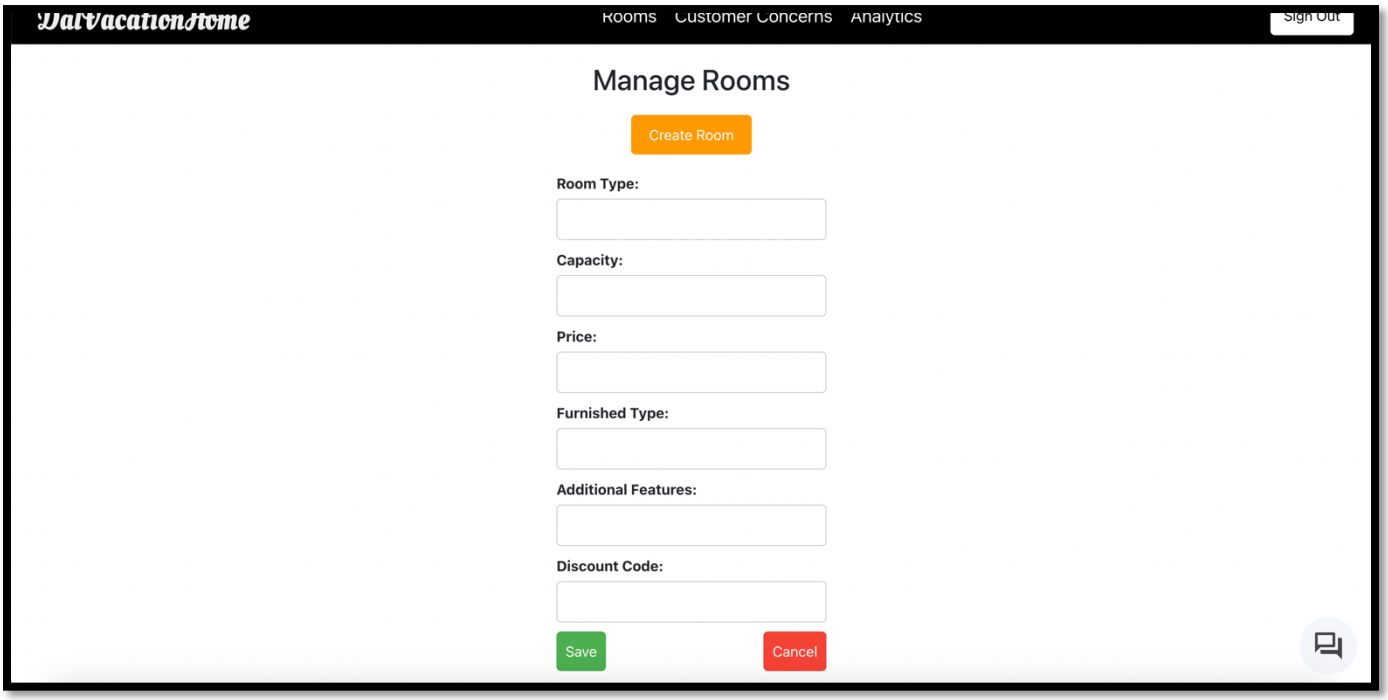

Figure 14: Create Room UI

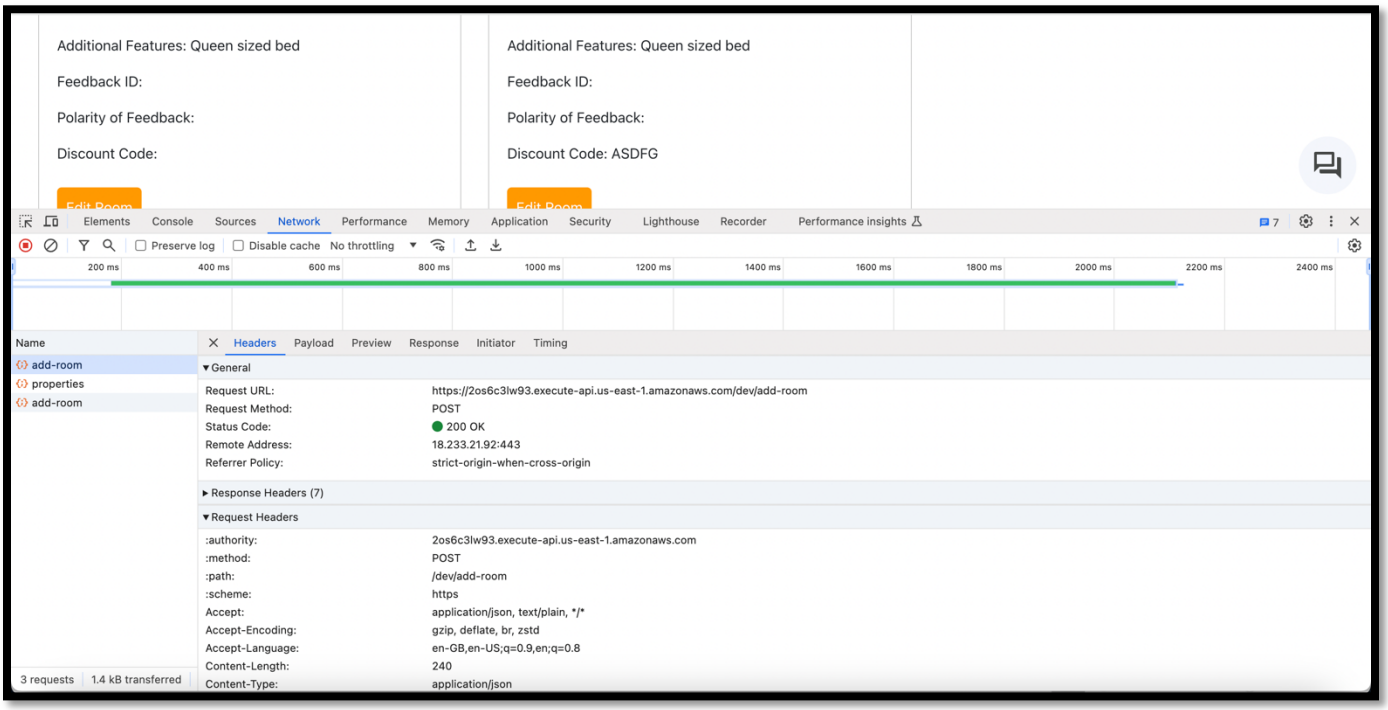

Figure 15: API integration of add room

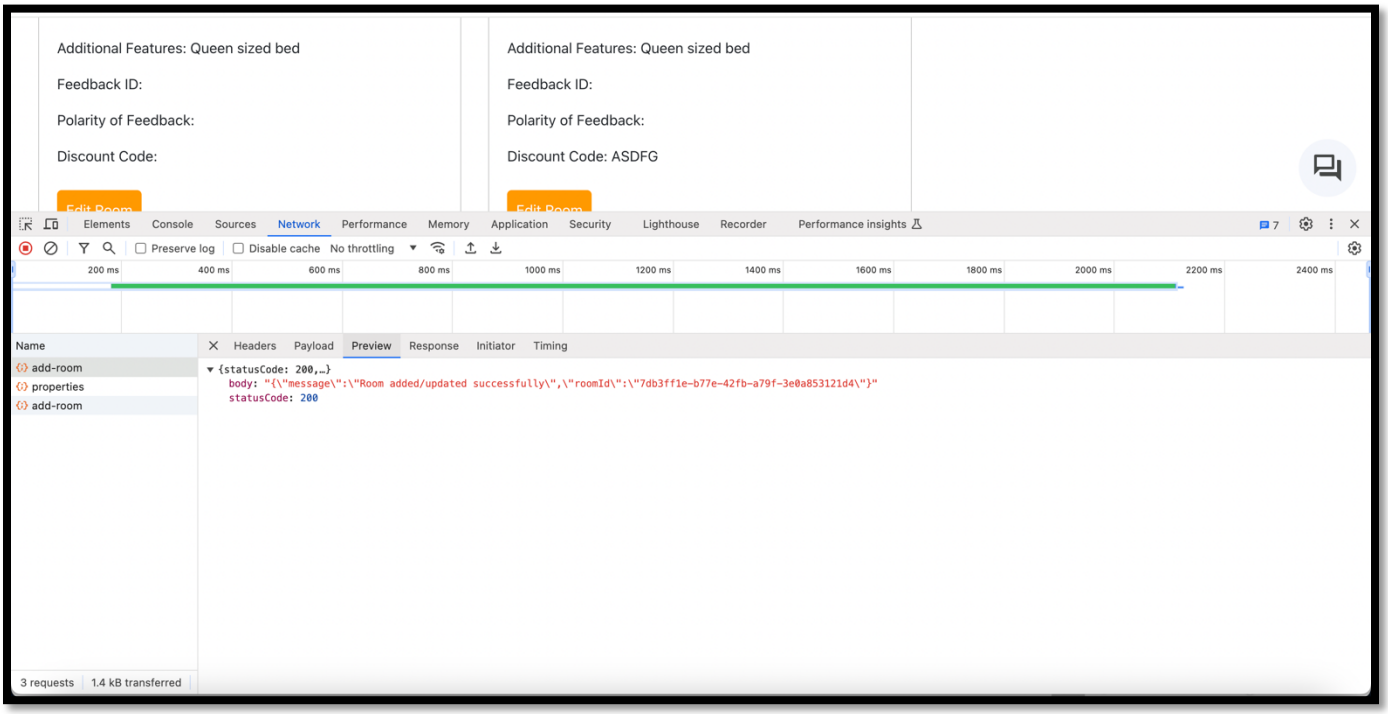

Figure 16: Response of addRoom API

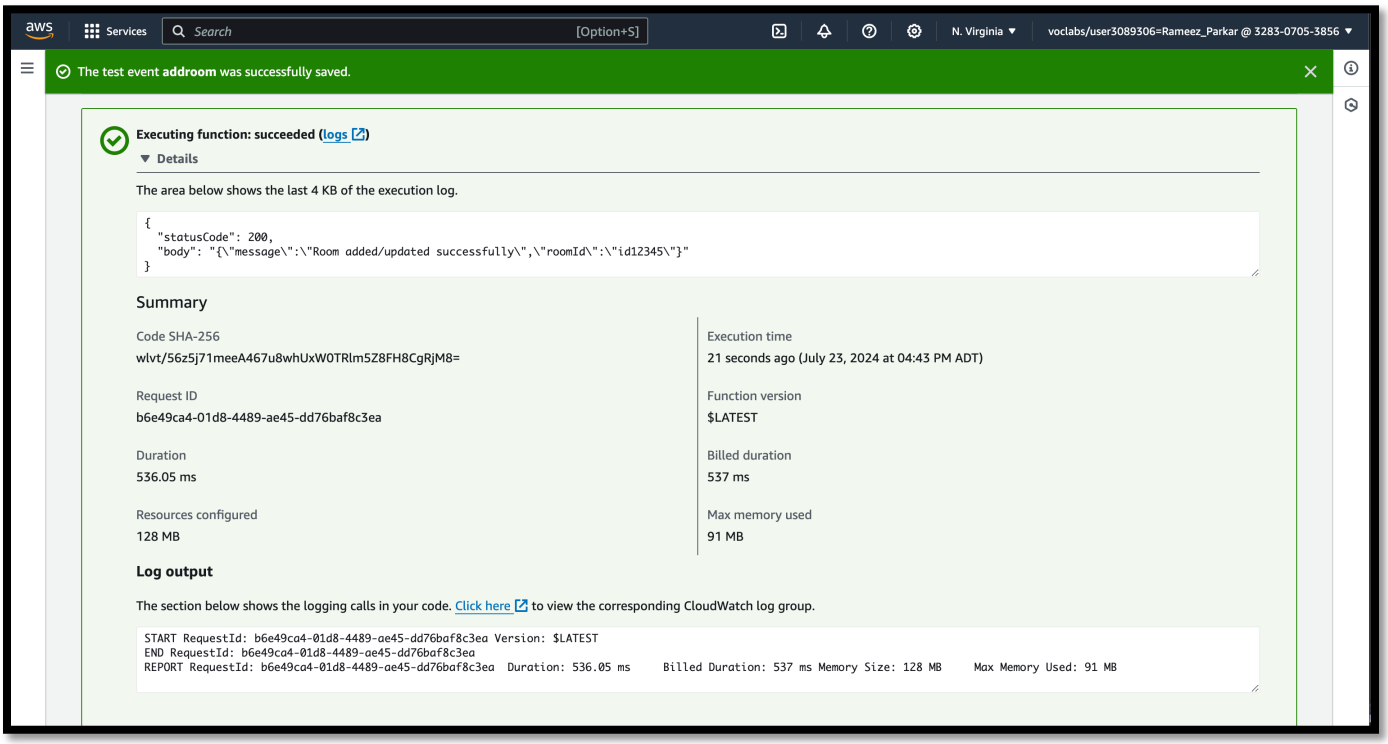

Figure 17: Lambda for add room

Testcase 2: Edit Room by Property Agent

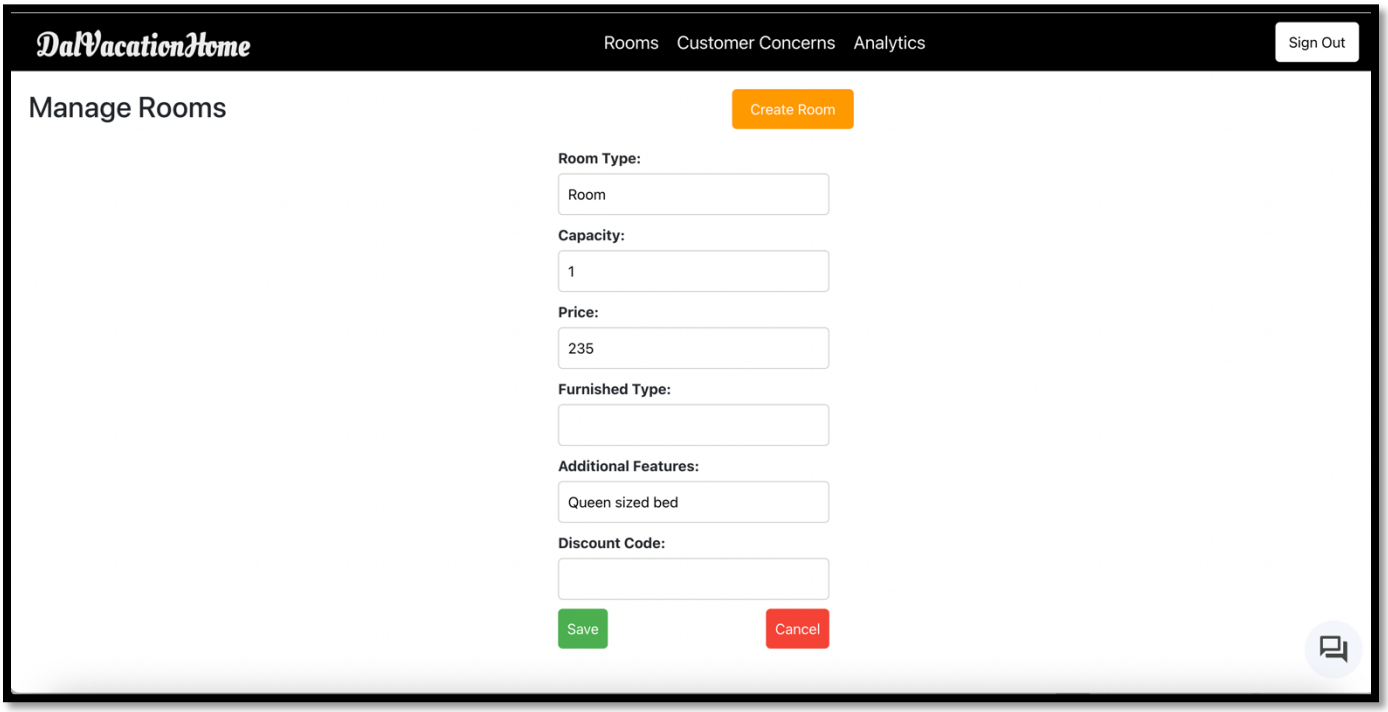

Figure 18: Edit Room UI For PropertyAgent

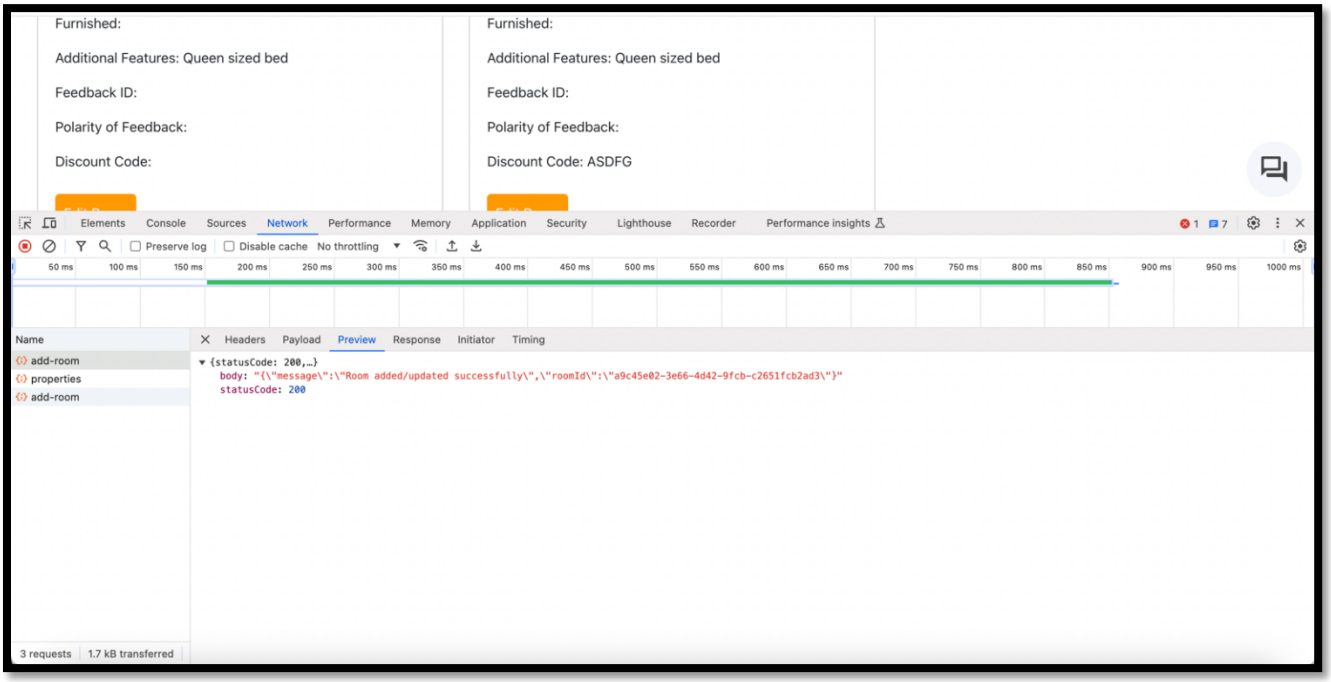

Figure19: API Response after editing the room

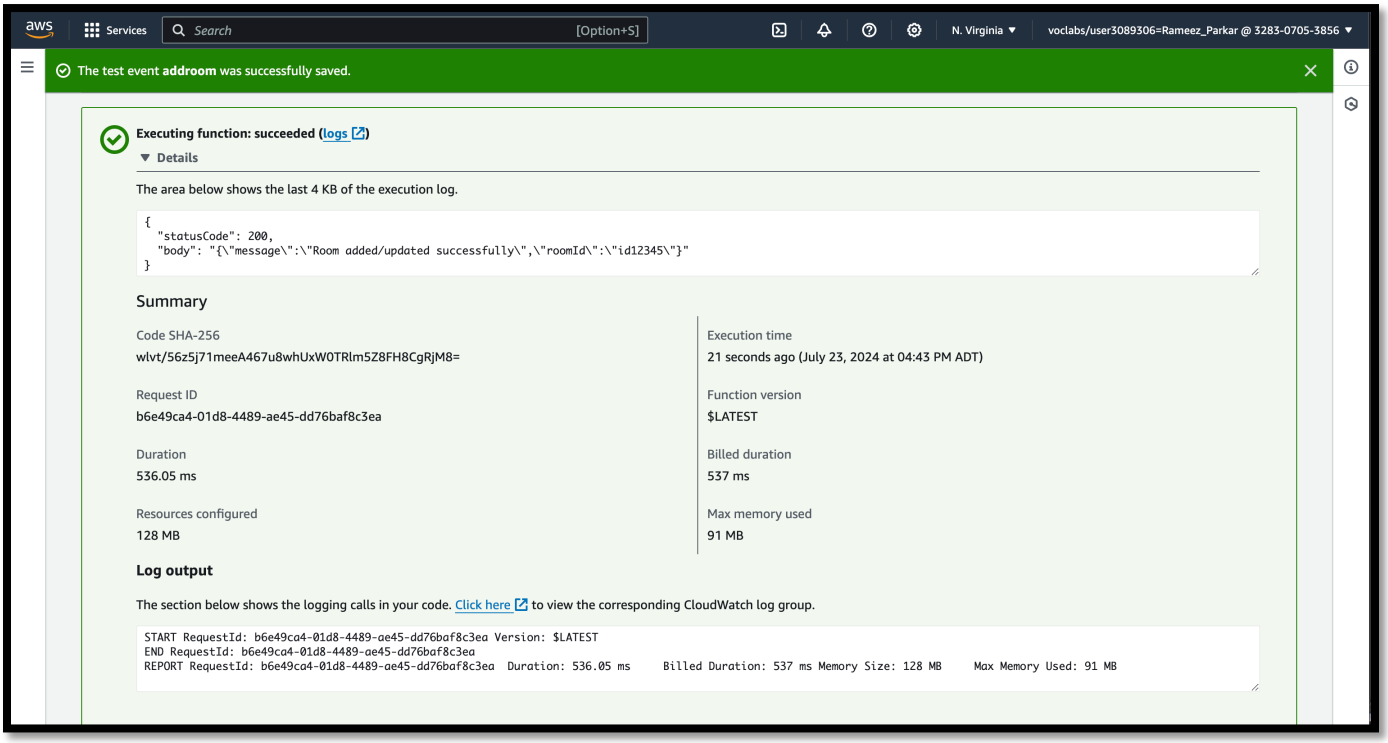

Figure 20: Lambda for update room

Module: Data Analysis

Feedbacks and API integration

Testcase 1: Successful fetching of feedbacks

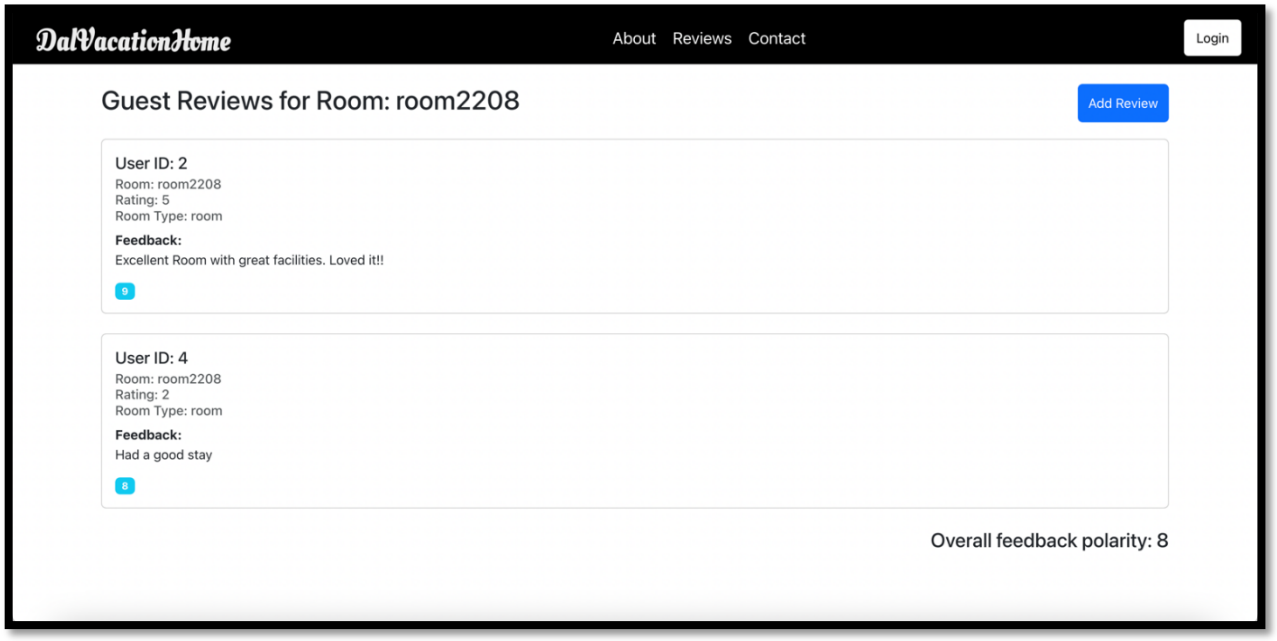

Figure 21: Guest Reviews UI

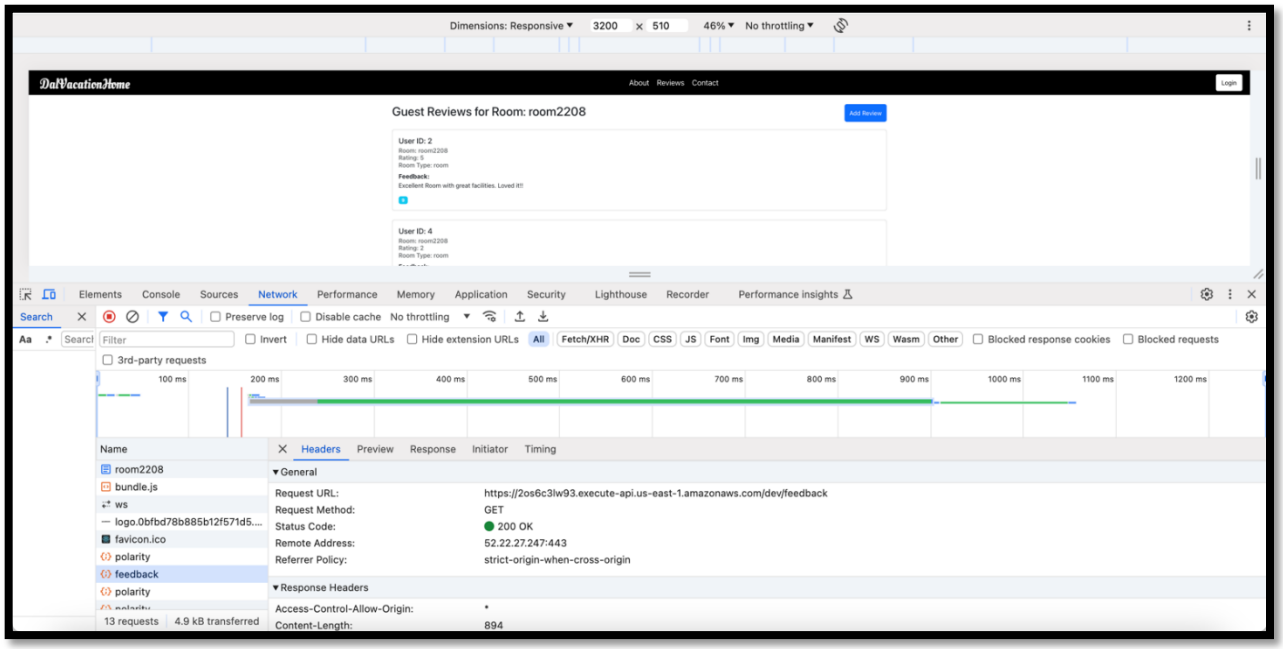

Figure 22: Feedback API Integration

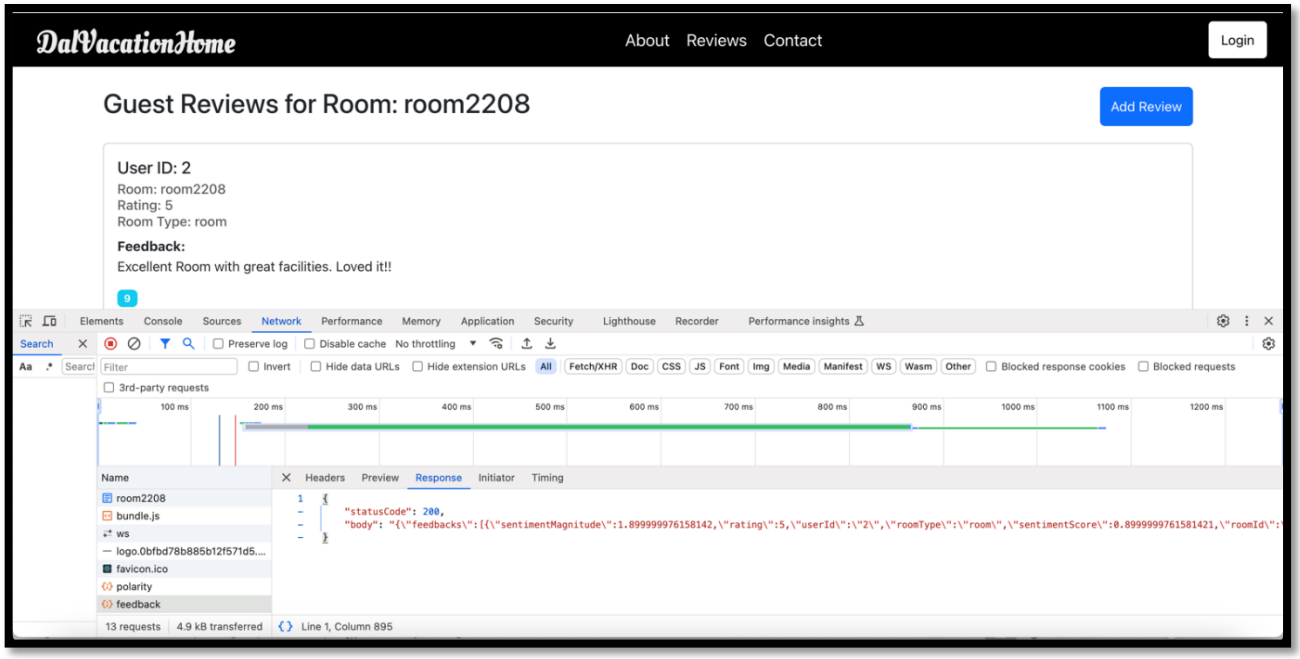

Figure 23: Response of feedback API

Testcase 2: Successful fetching of feedbacks overall polarity

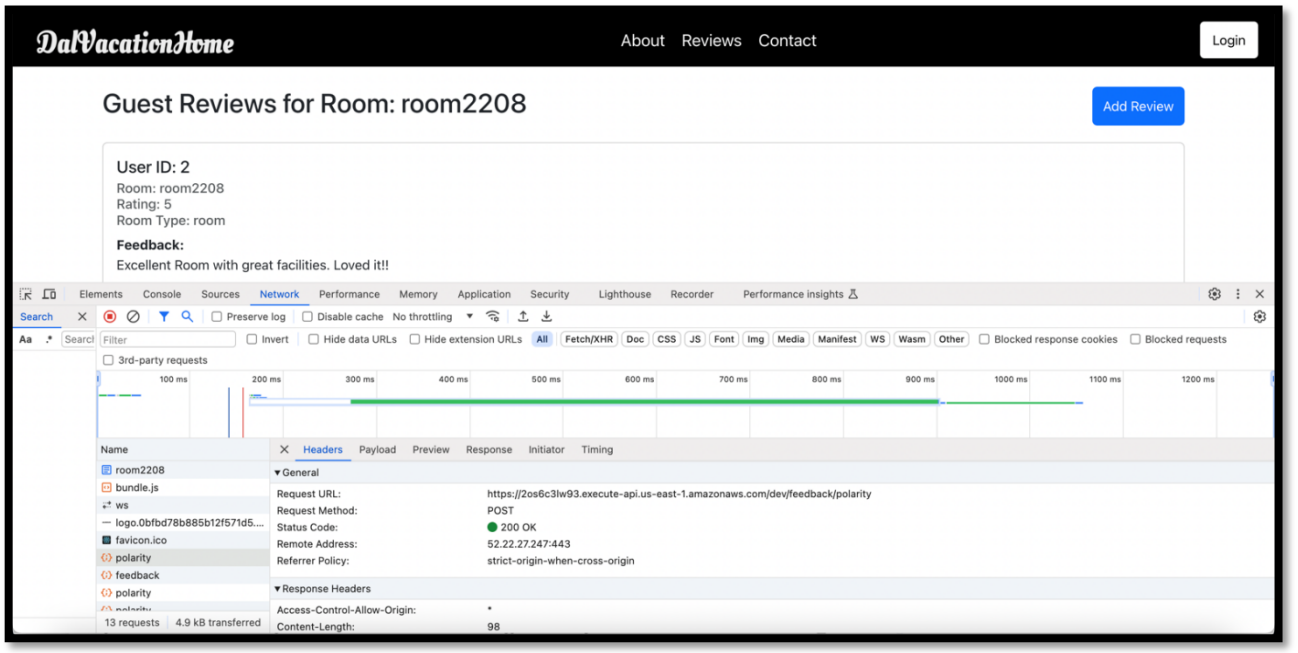

Figure 24: Feedback Polarity API Integration

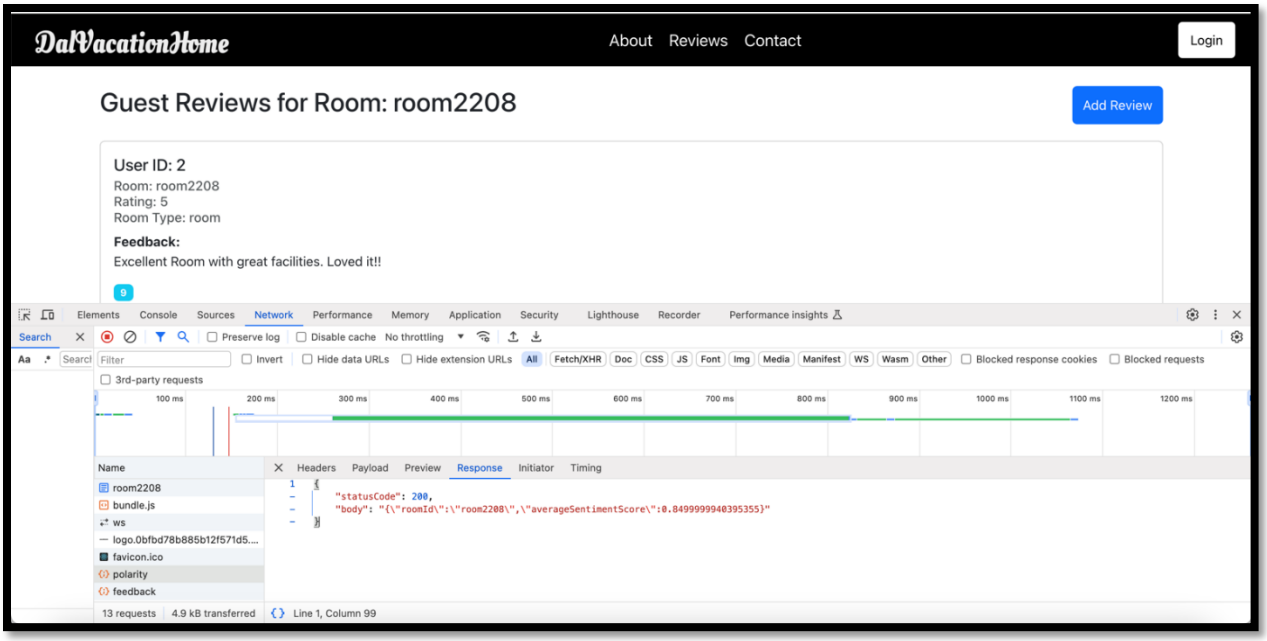

Figure 25: Response of getFeedbackPolarity API

Testcase 3: Registered Customers can add Reviews

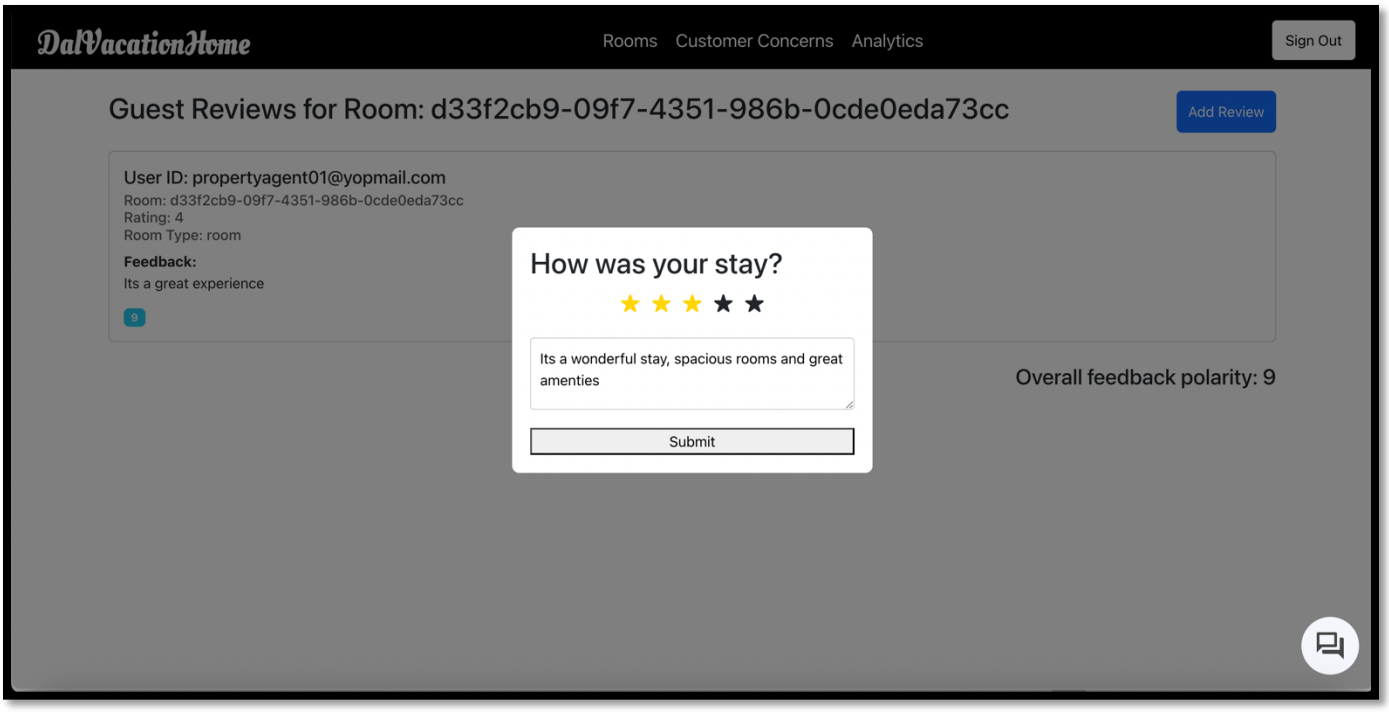

Figure 26: Add Review UI

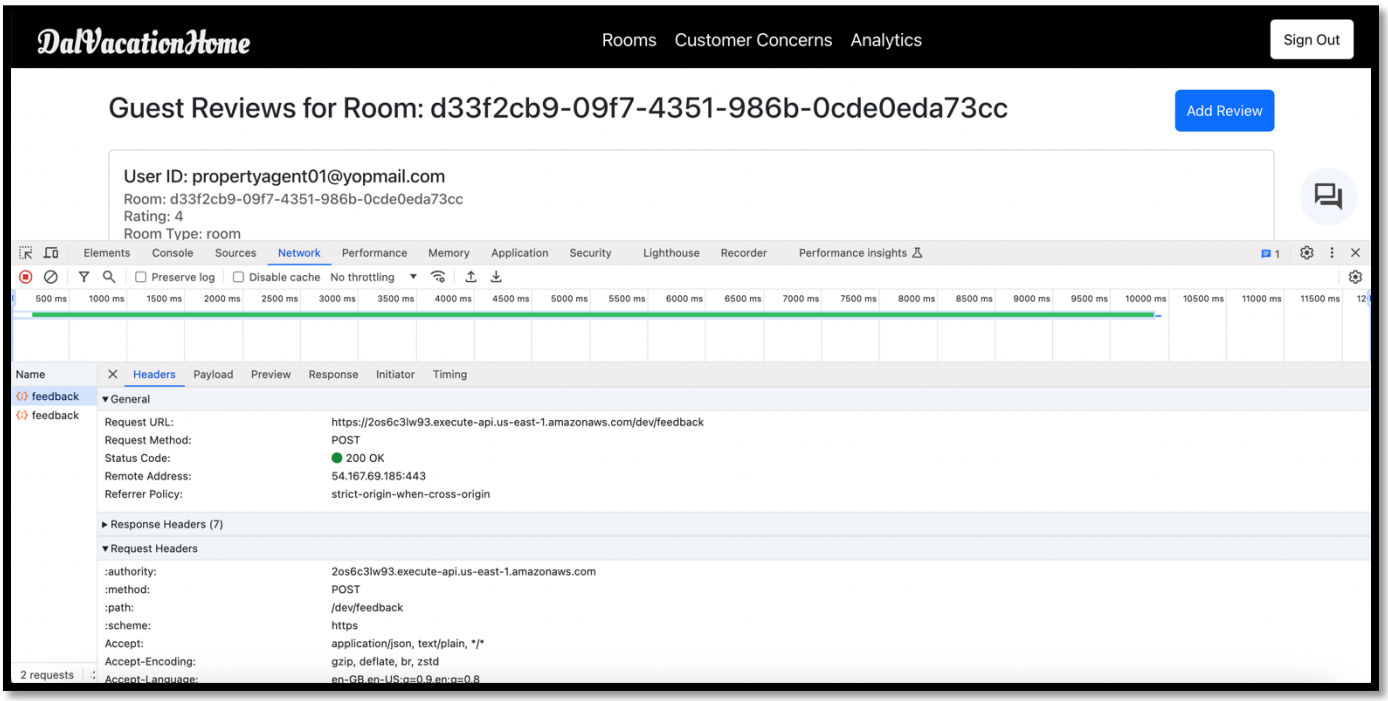

Figure 27: API Integration of add feedback

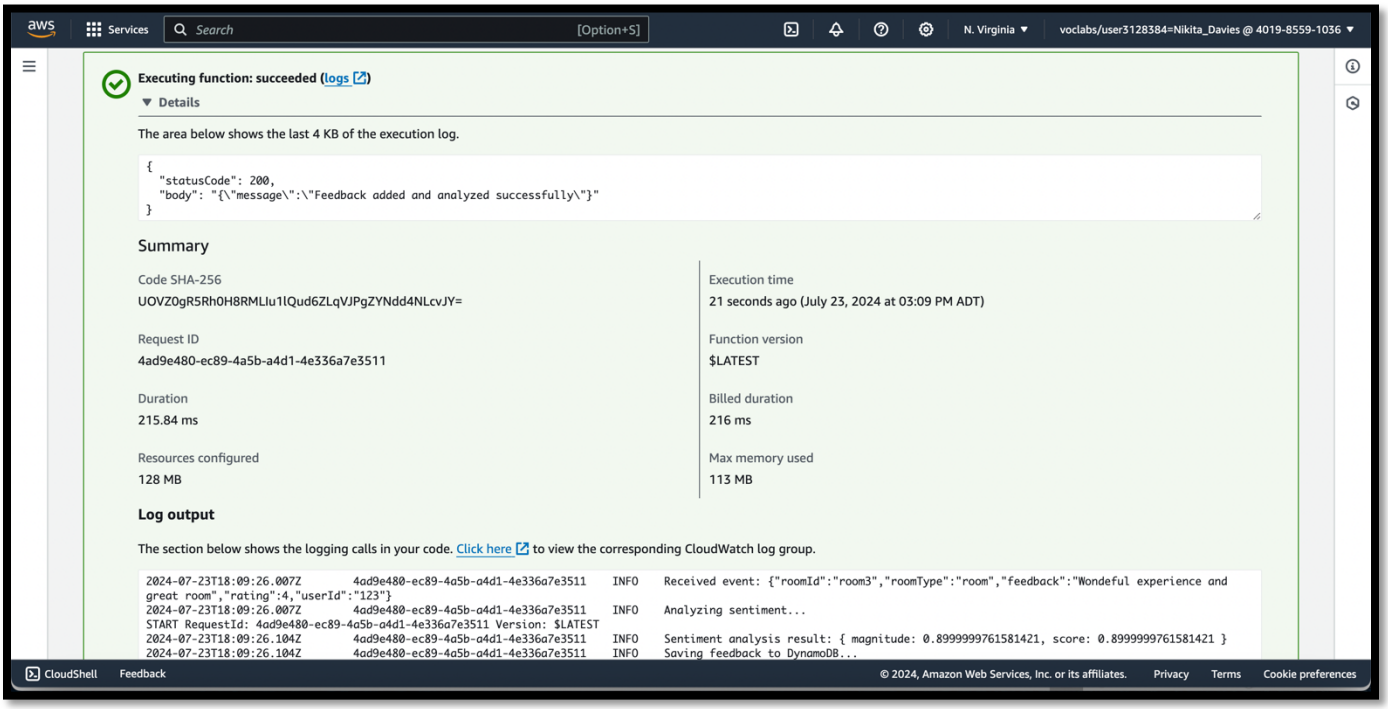

Figure 28: Lambda invocation test for addFeedback

Testcase 4: Data Analytics using LookerStudio

Figure 29: Admin dashboard with embedded Lookerstudio url

Module: Virtual Assistant using DialogFlow CX

Testcase 1: User asks chatbot how to register

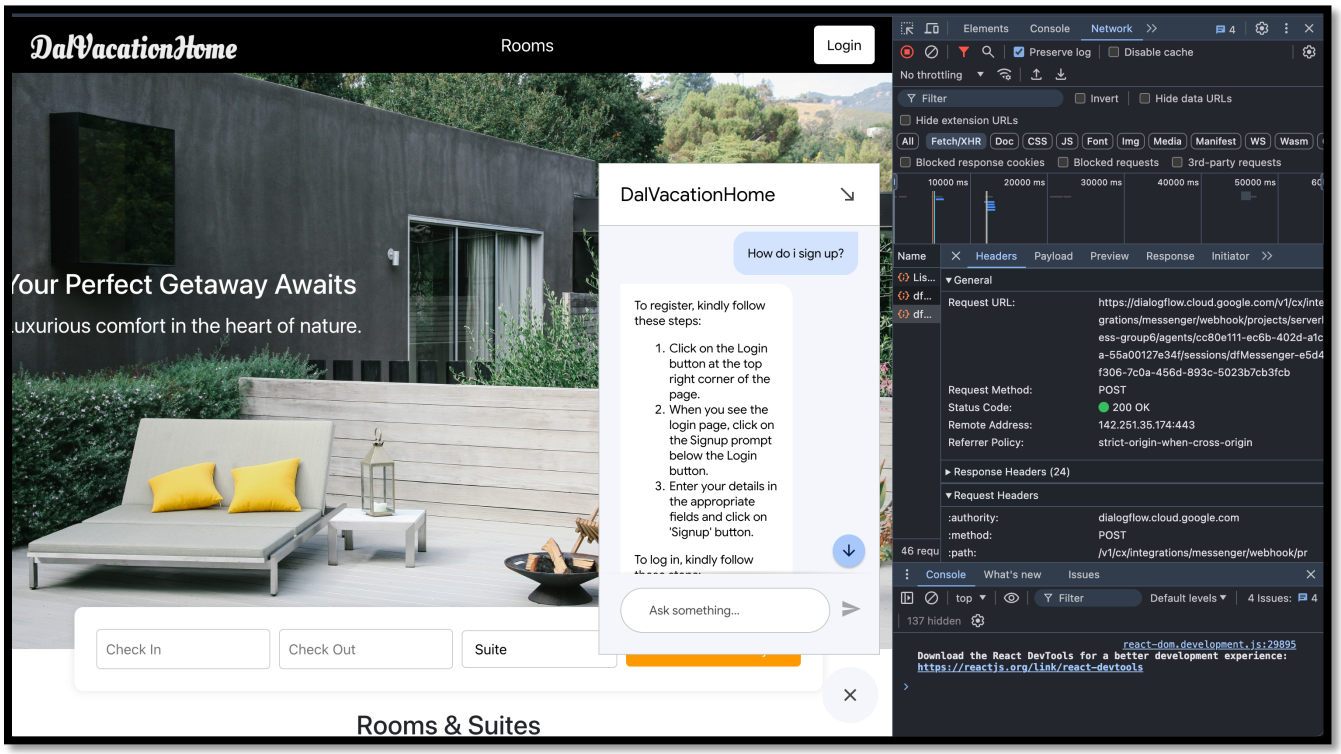

Figure 30: Dialogflow Chatbot for How to Register

Testcase 2: User asks chatbot how to book a room

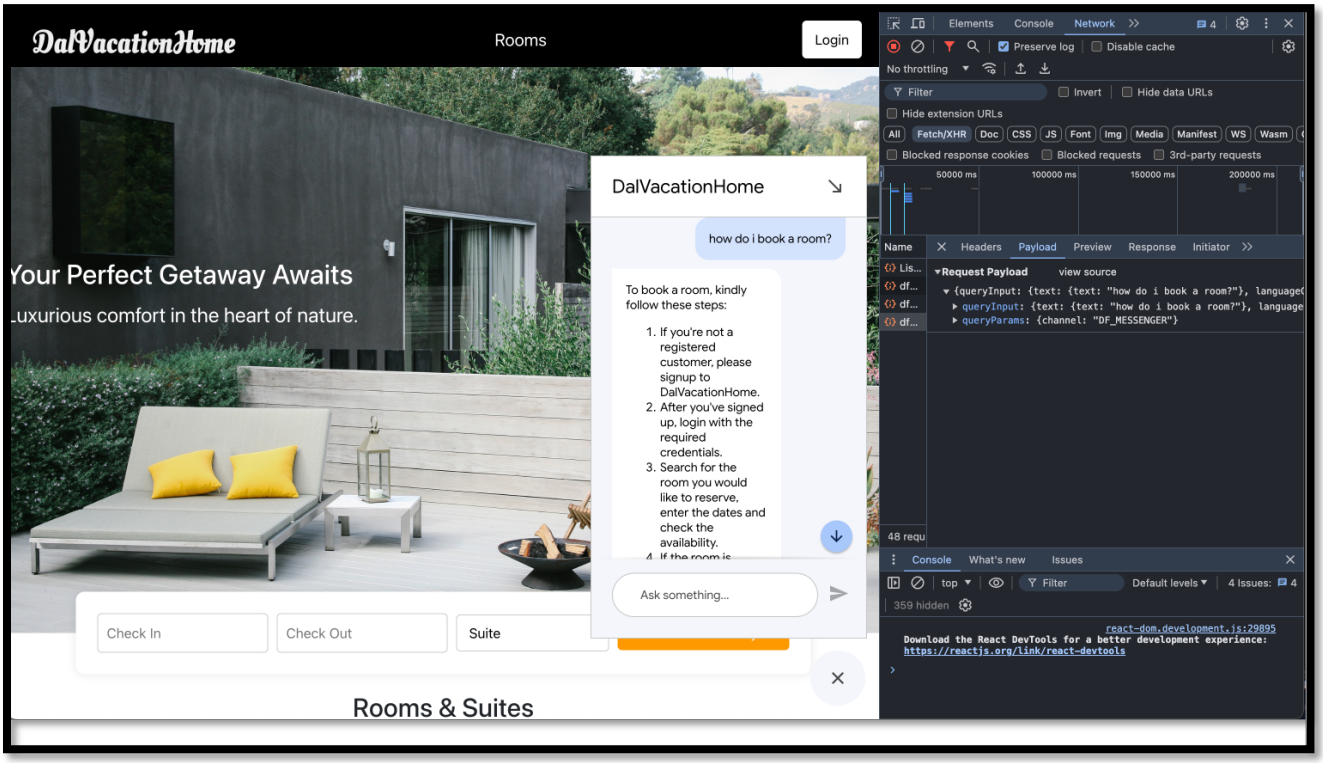

Figure 31: Dialogflow Chatbot for How to Book Room

Testcase 3: User asks chatbot for their booking details

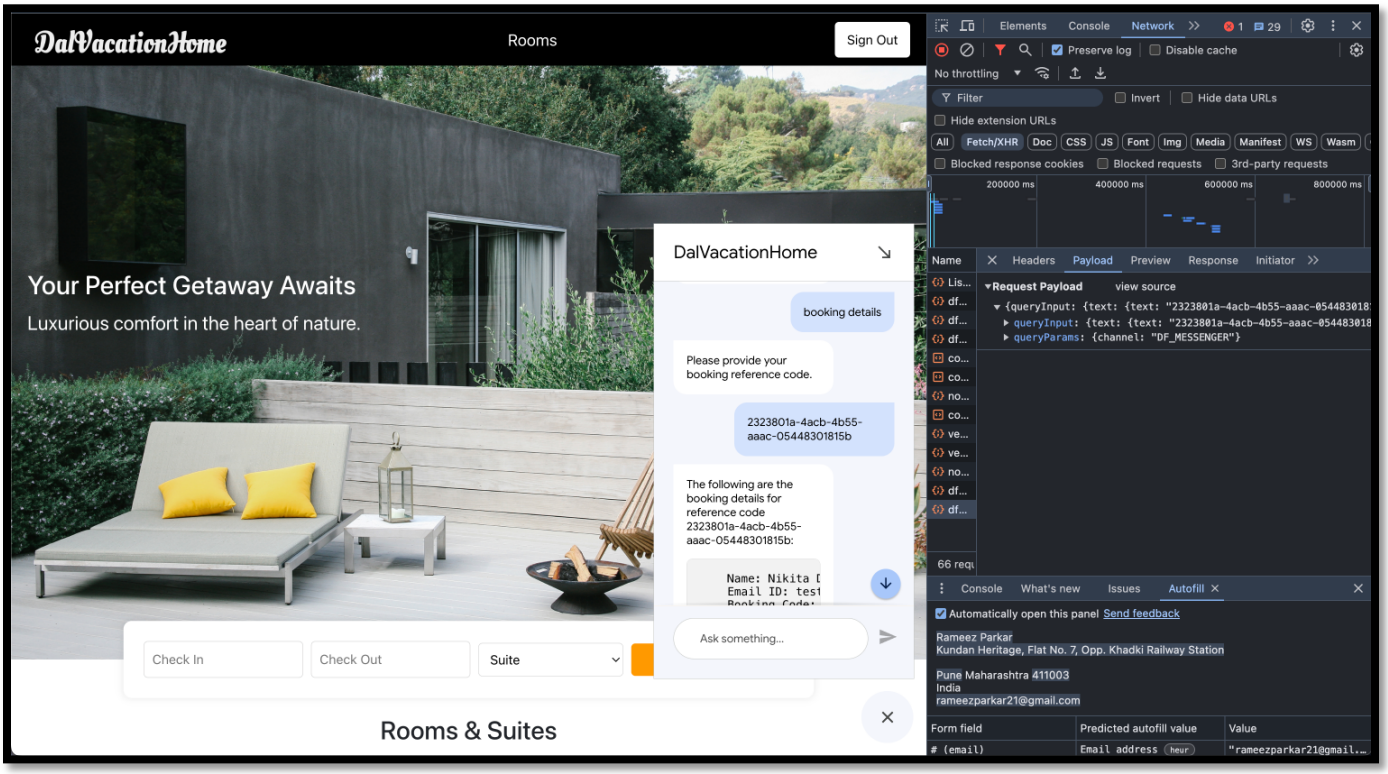

Figure 32: Dialogflow Chatbot for Booking Details

Testcase 4: User mentions a concern to the chatbot

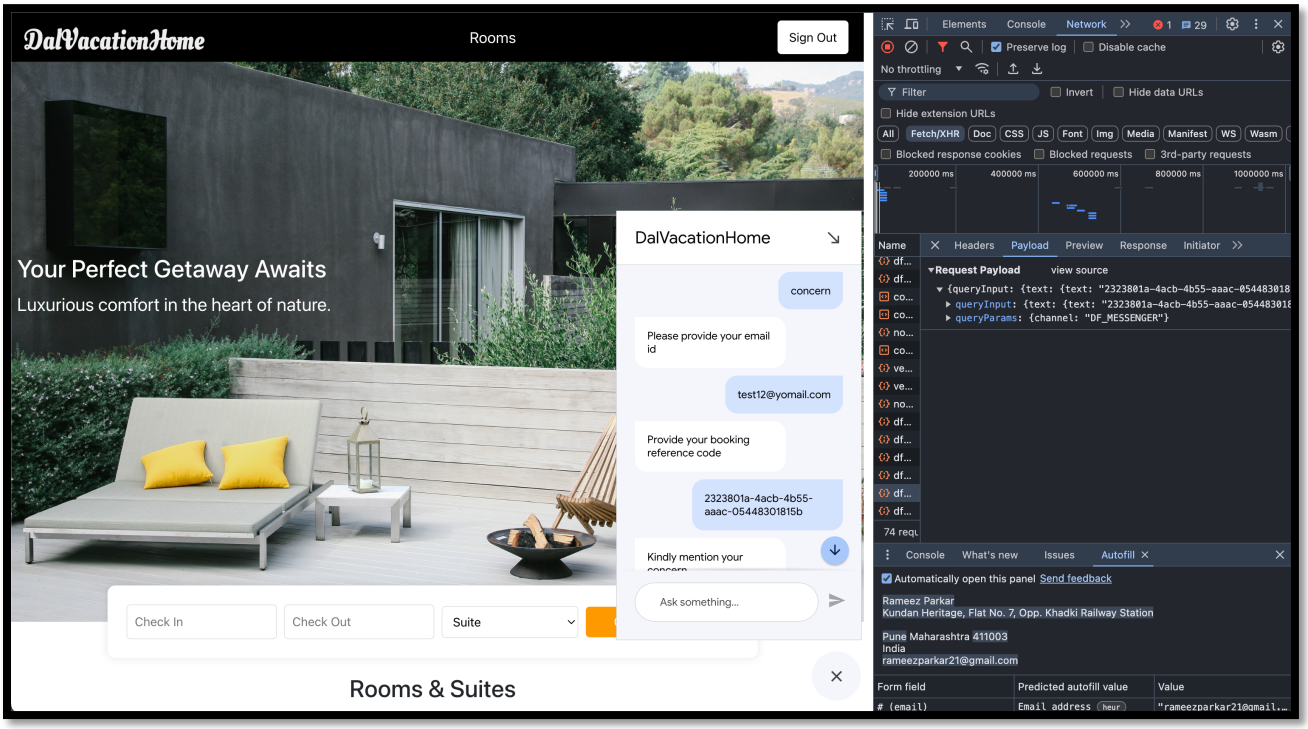

Figure 33: Dialogflow Chatbot for customer concerns

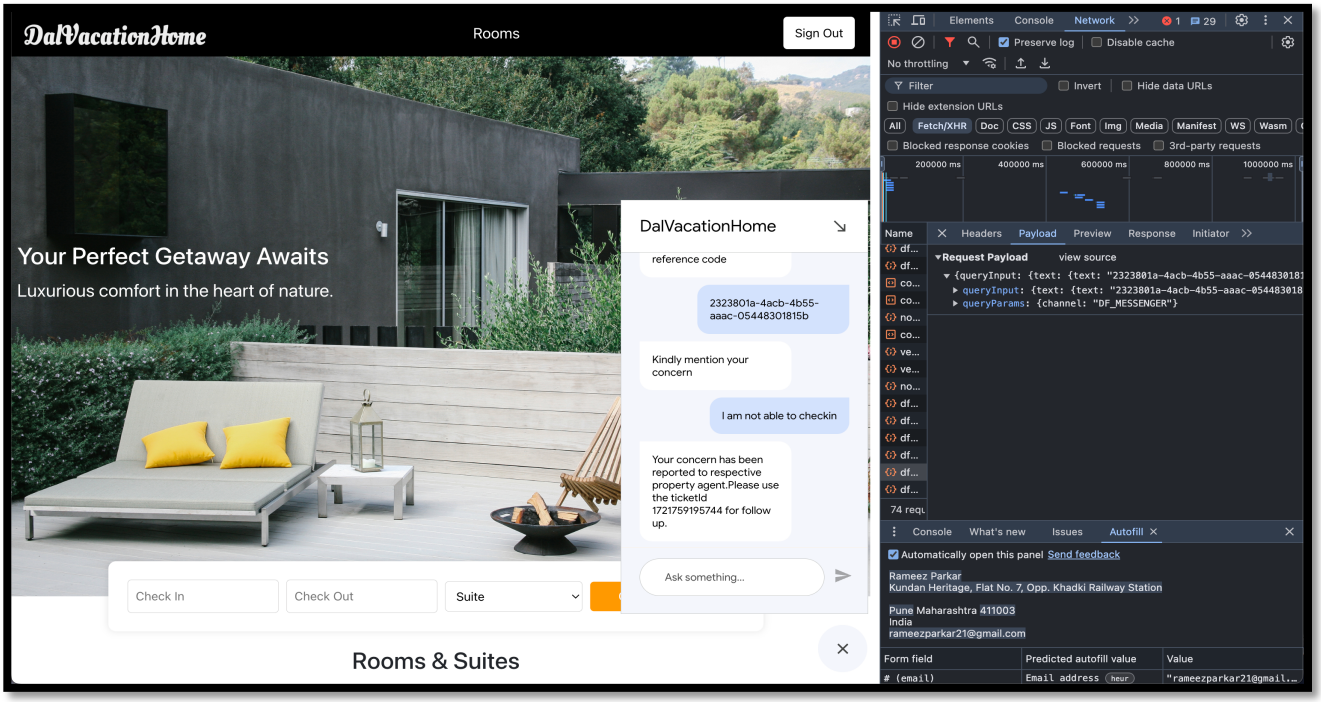

Figure 34: Dialogflow Chatbot for customer concerns

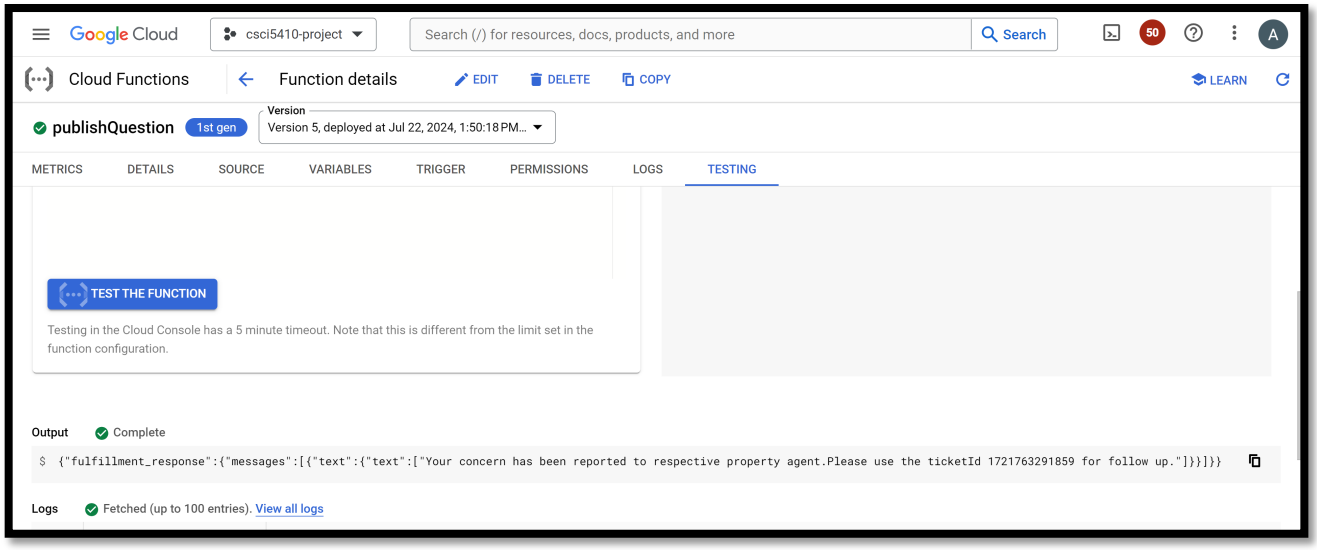

Figure 35: Cloud function for publish question

Testcase 5: User asks status of the submitted concern to the chatbot

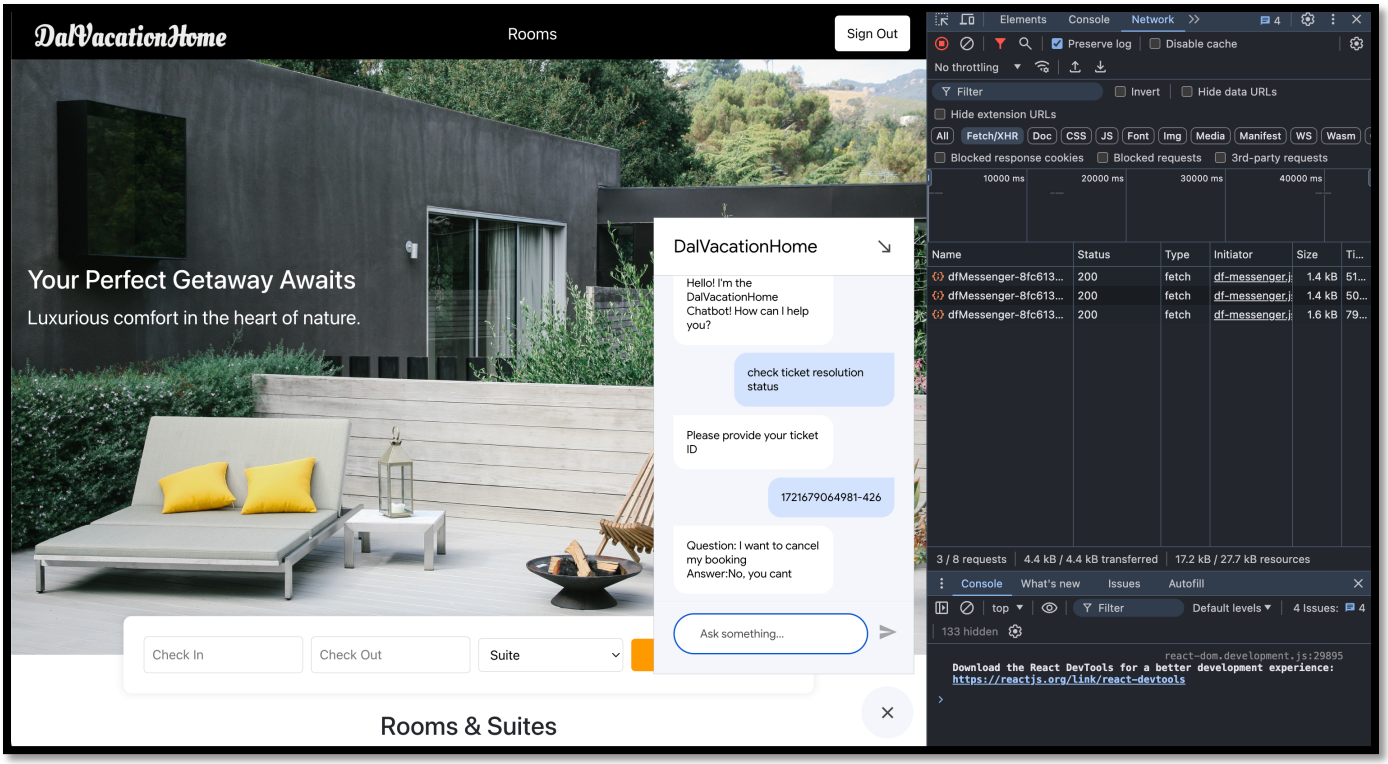

Figure 36: Dialogflow to check ticket resolution status

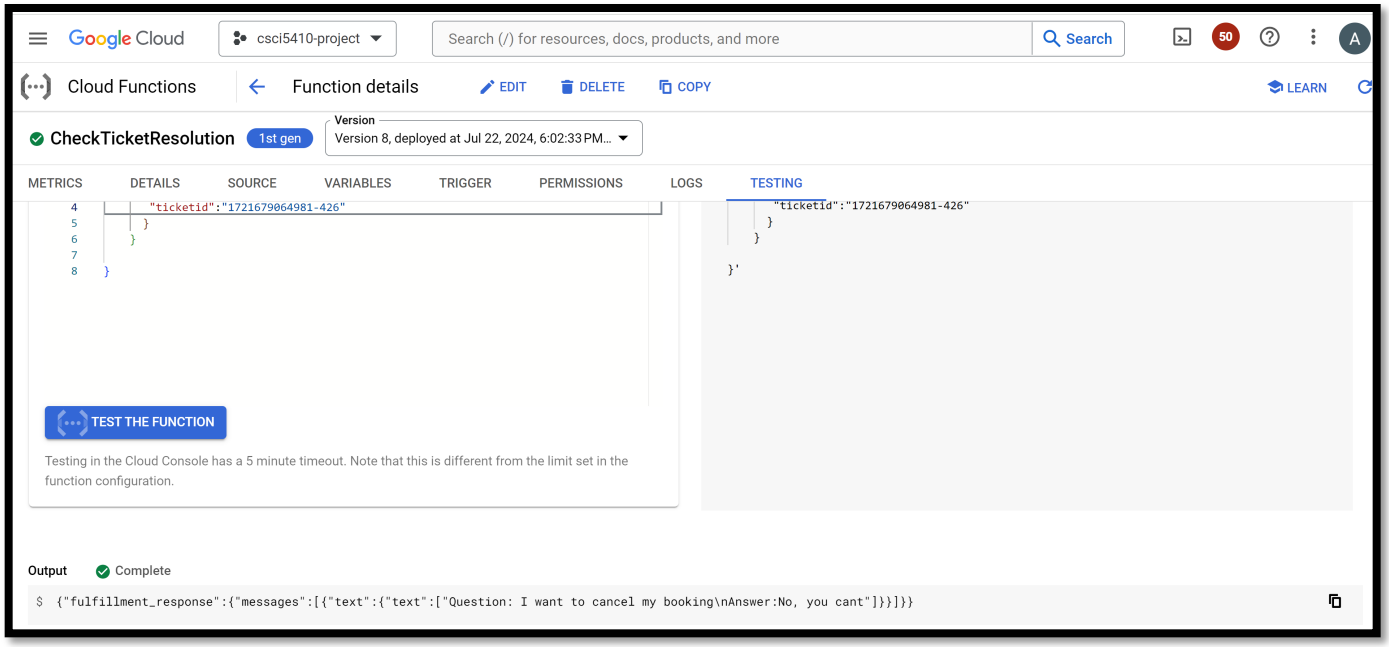

Figure 37: Cloud function call to fetch question and answer for submitted concern

Testcase 6: Handling negative scenario in chatbot

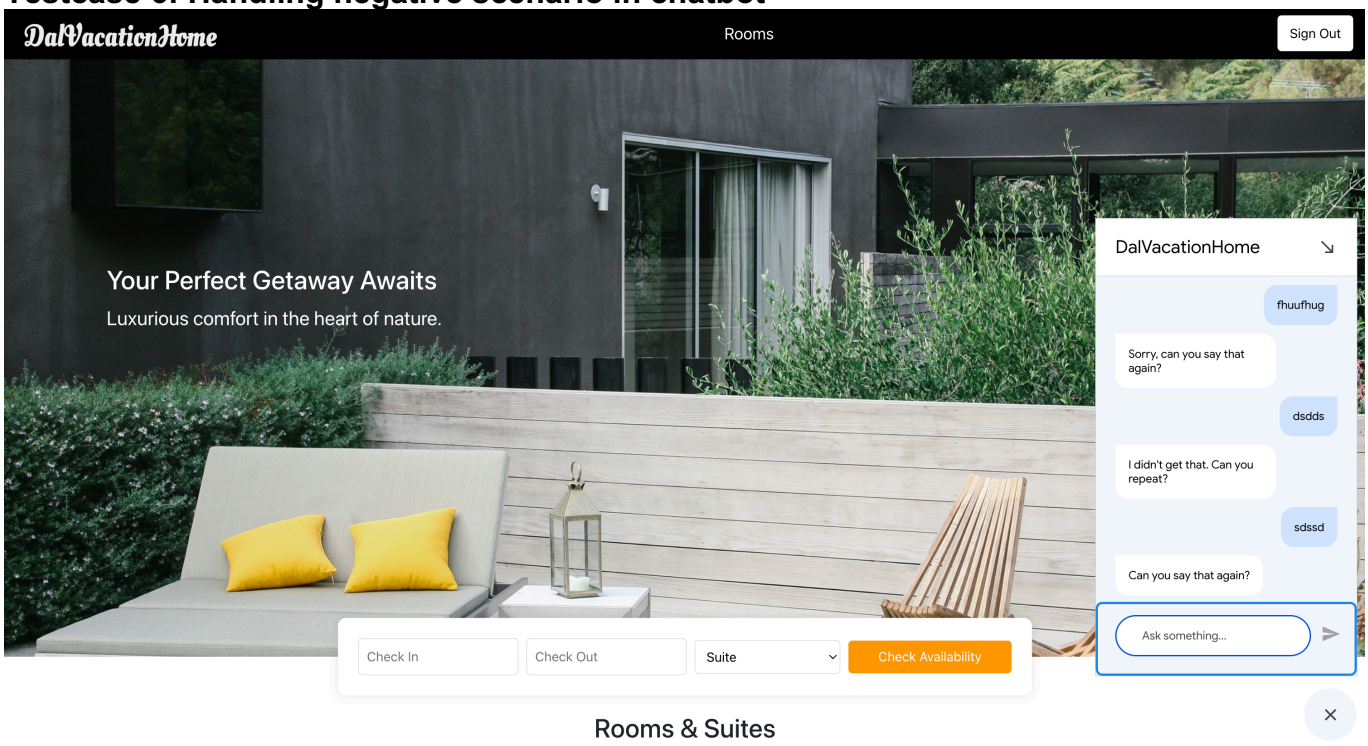

Figure 38: Handling negative scenario for chatbot

Module: Message Passing using GCP Pub/Sub

Testcase 1: User is able to initiate a ticket through Chatbot

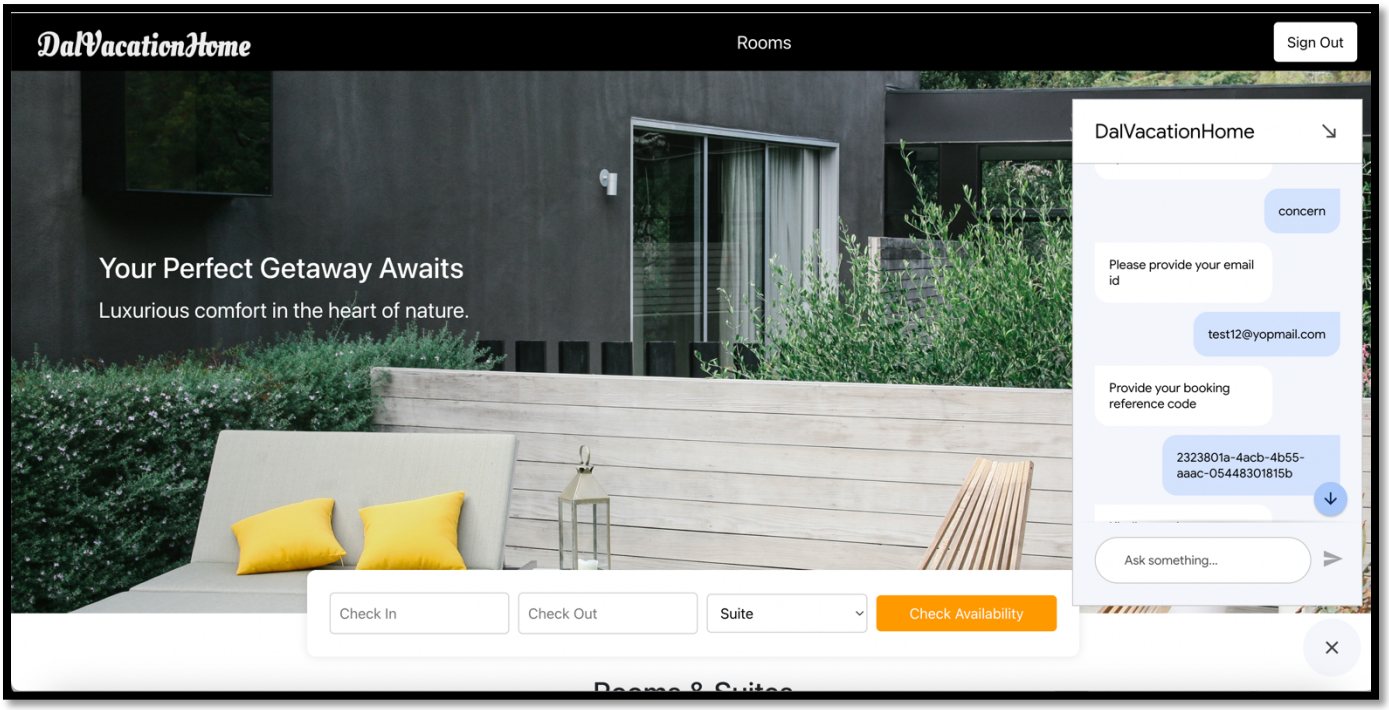

Figure 39: Initiate the concern from Chat bot

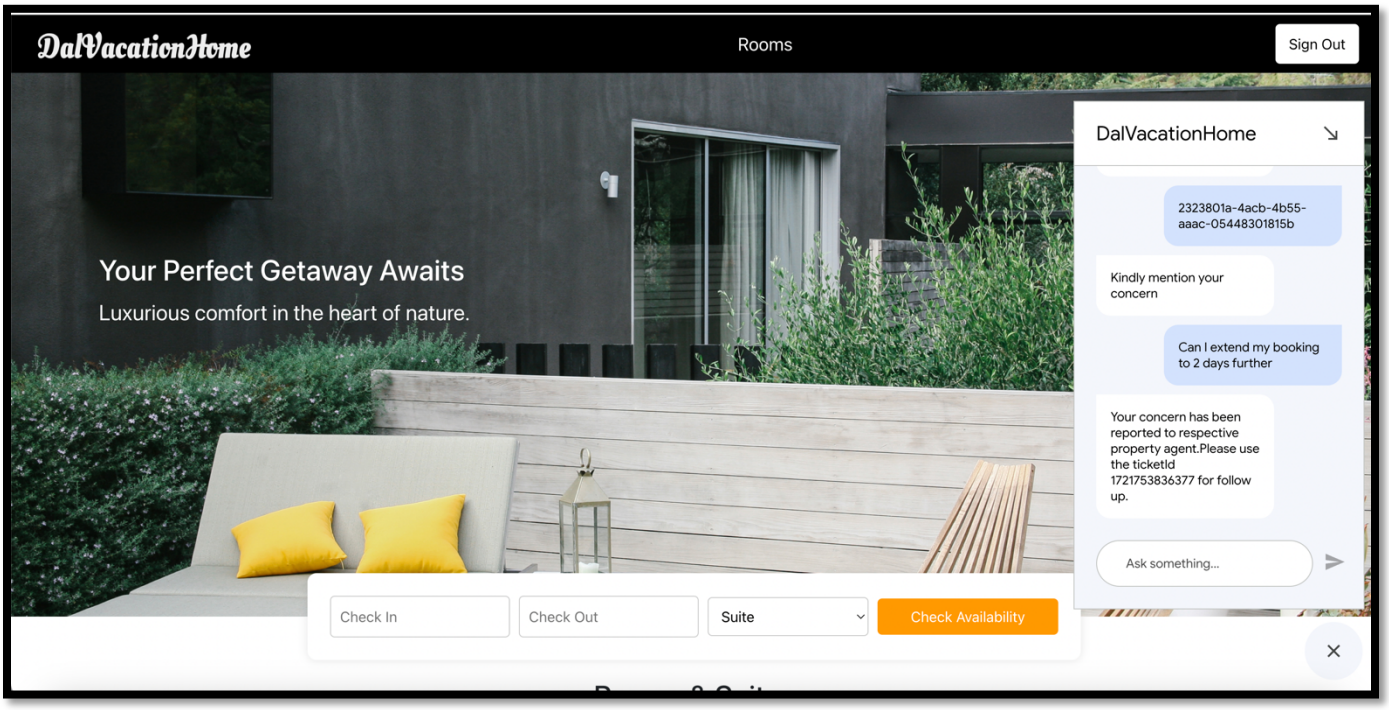

Figure 40: Provide relevant information about the concern

Testcase 2: Property Agent views the concern on customer concern tab

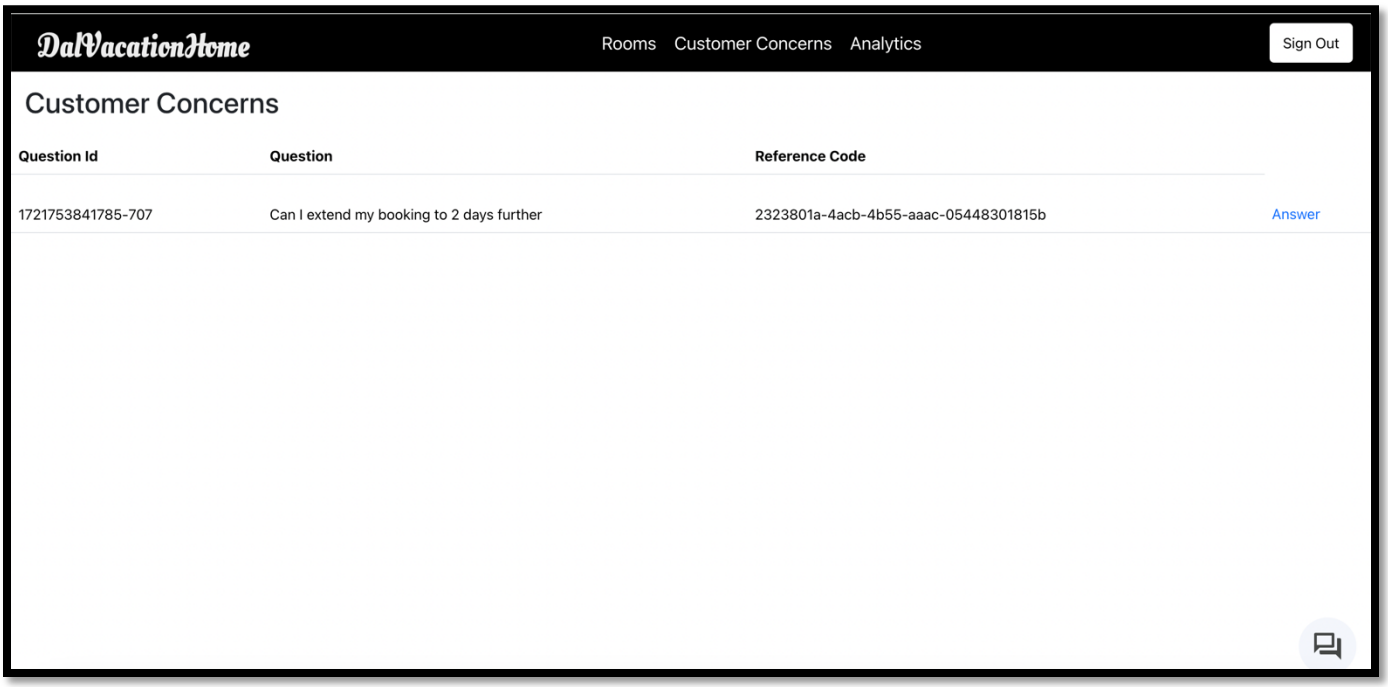

Figure 41: Customer concerns UI for Property Agent

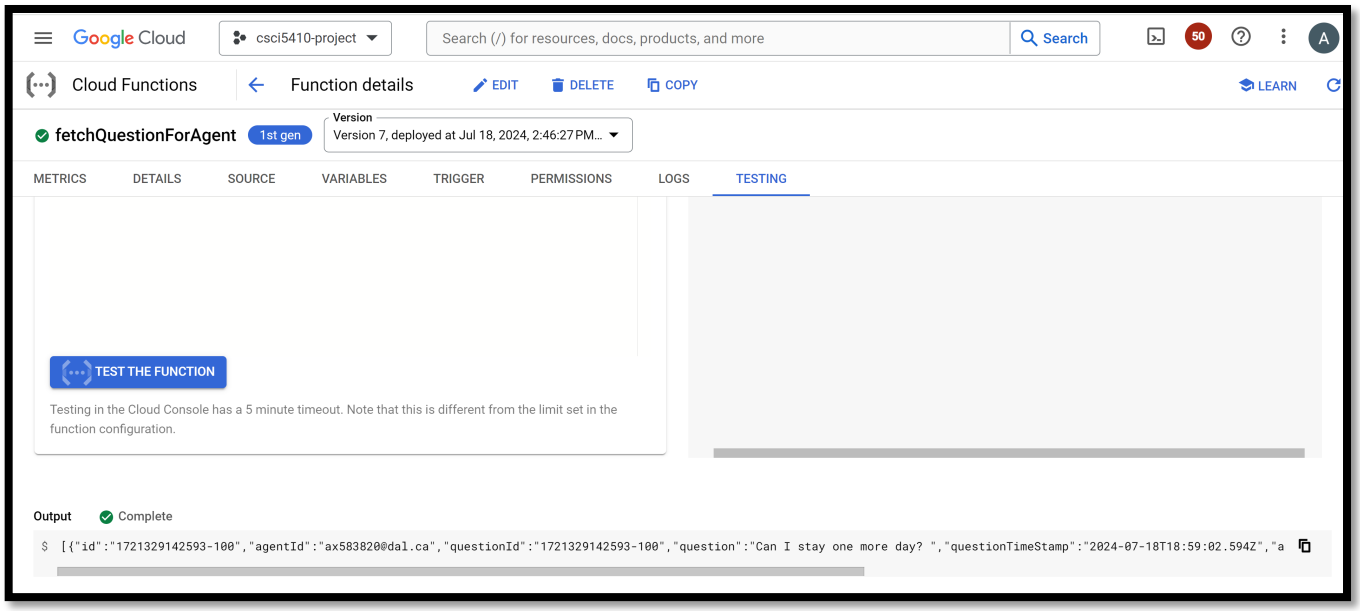

Figure 42: Cloud function call to fetch all posted concerns

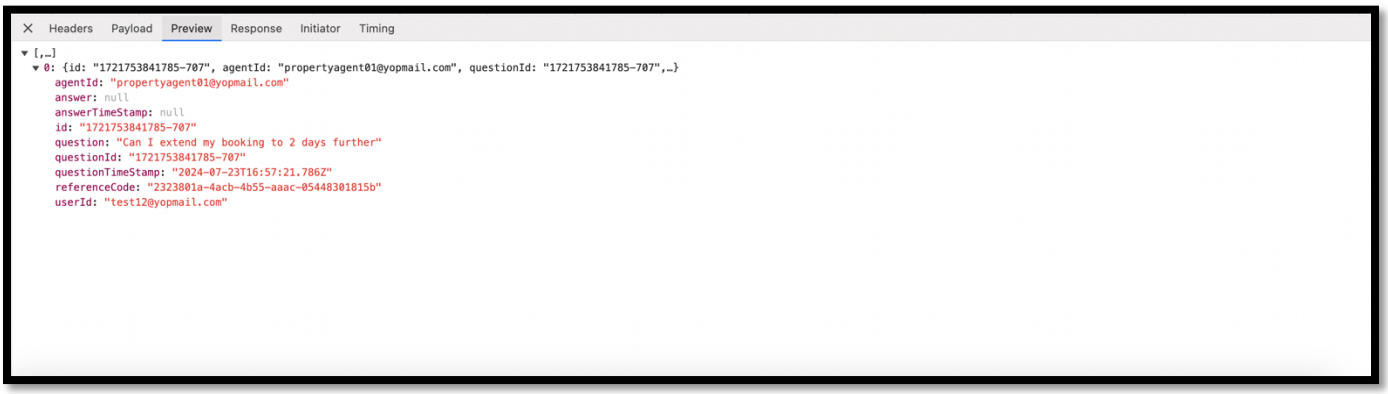

Figure 43: API Response of fetching all raised concerns for a property agent

Testcase 3: Property Agent answers the raised query

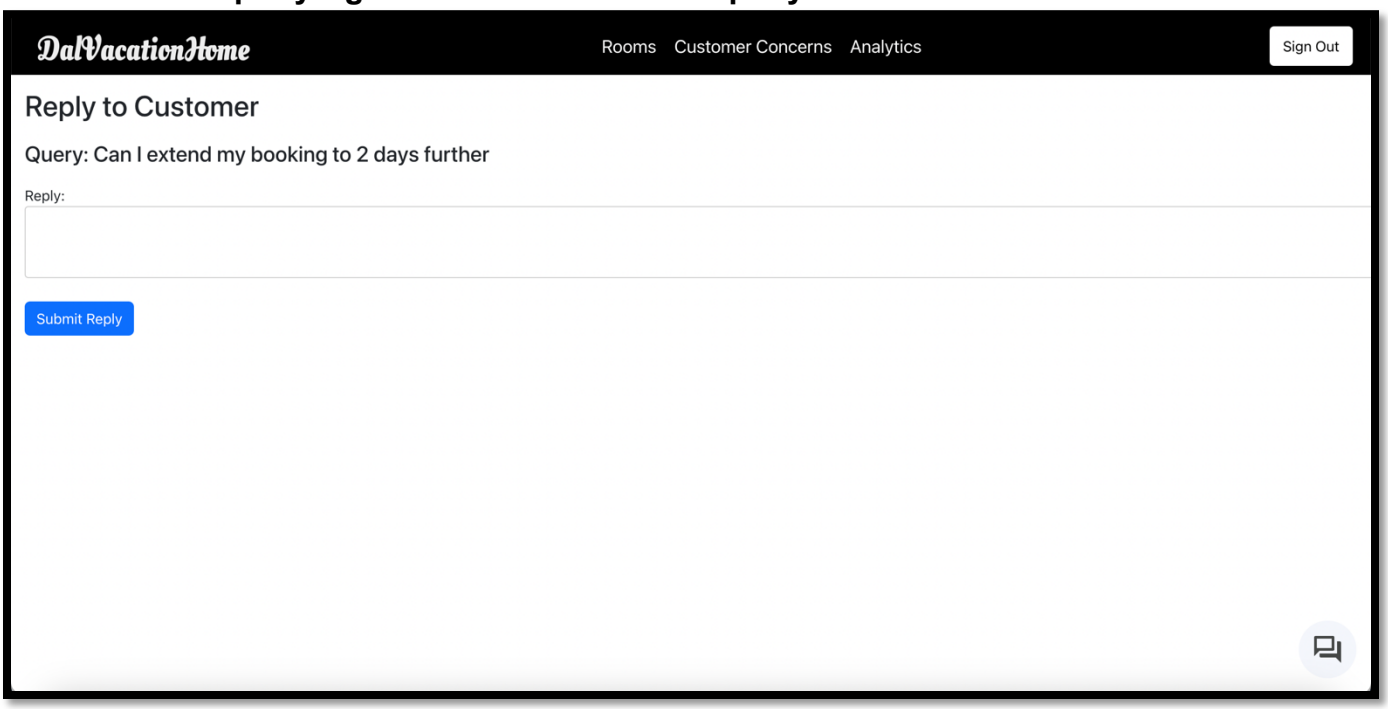

Figure 44: Property Agent can reply to a concern

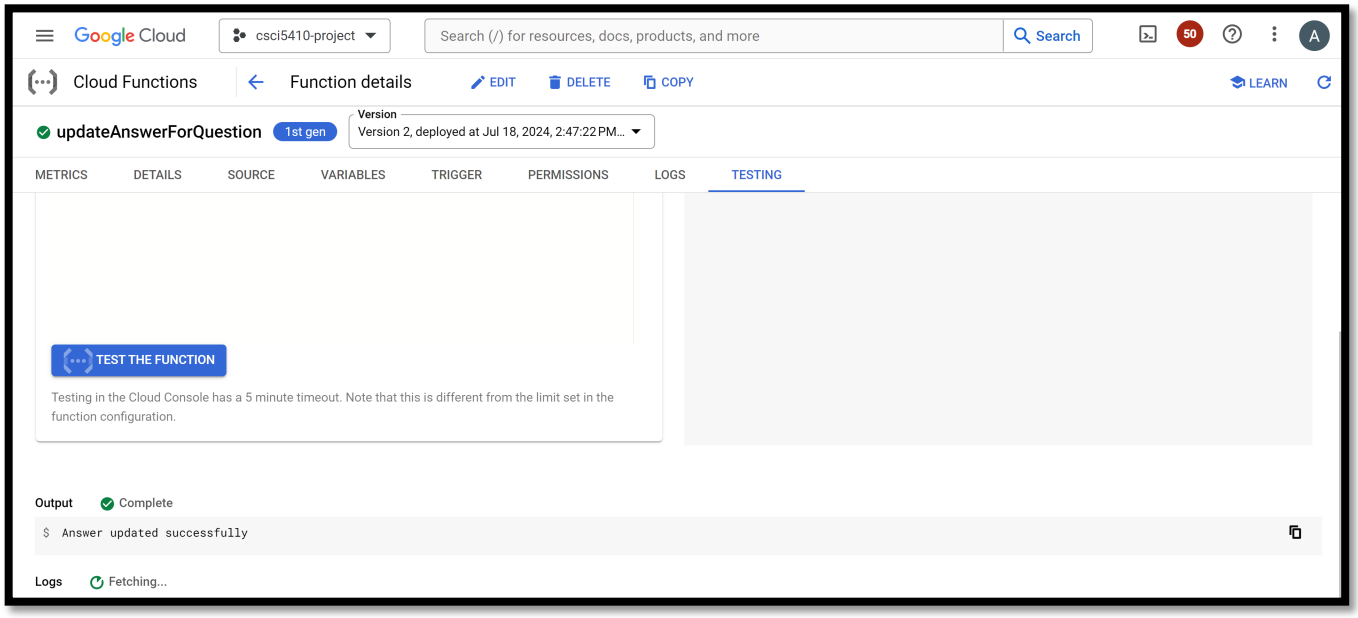

Figure 45: Cloud function call to update answer for specific concern

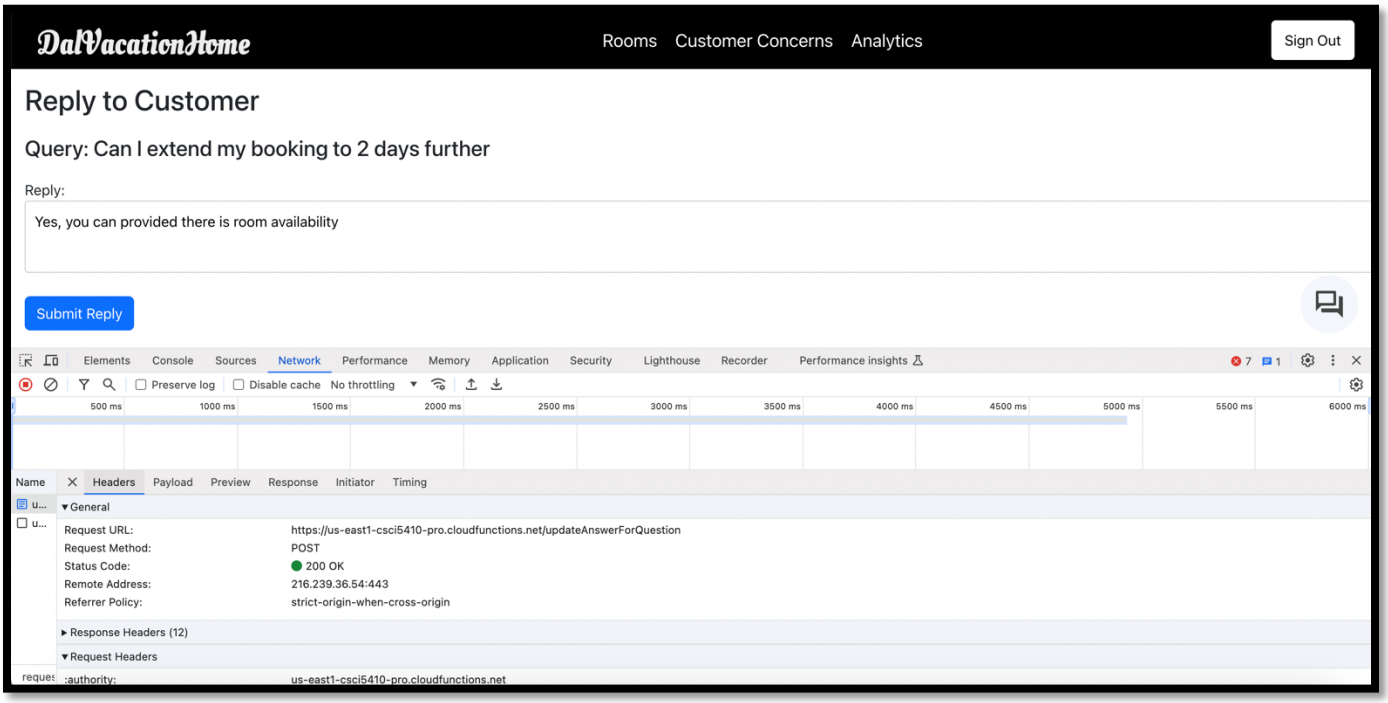

Figure 46: API Integration for a storing the reply from Property Agent

Testcase 4: Registered Customers can see answers to their raised concerns

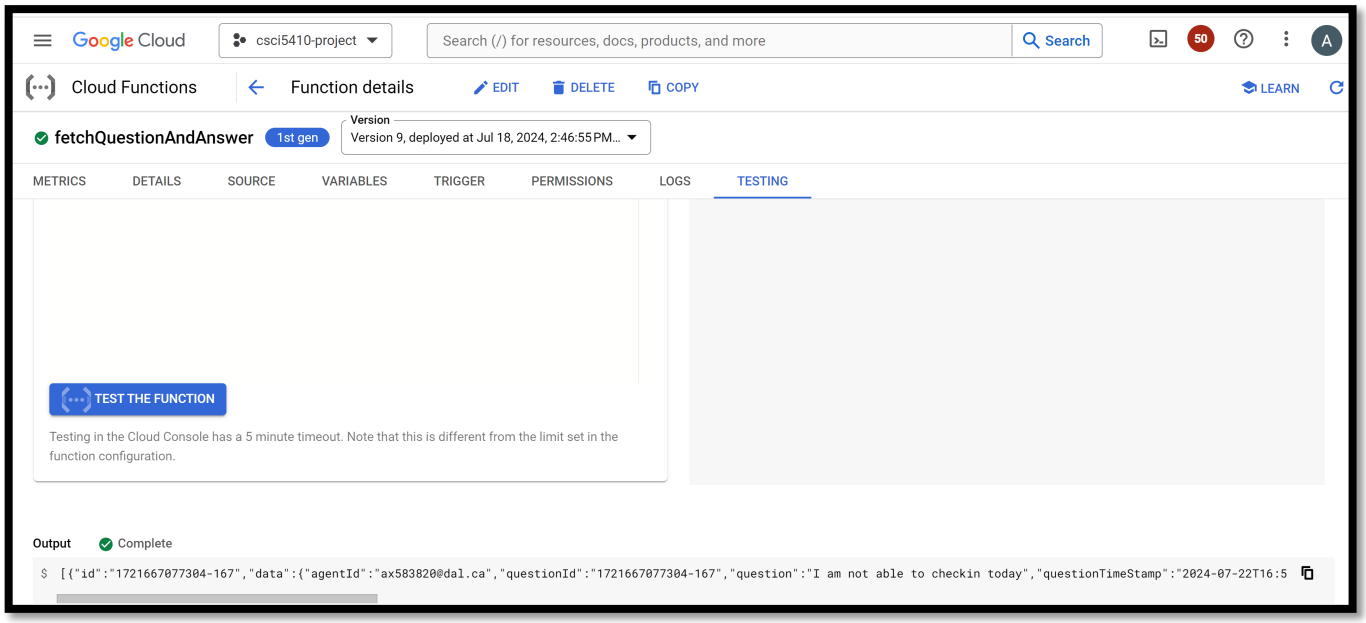

Figure 47: Cloud function call to fetch all concerns which are answered by property agent

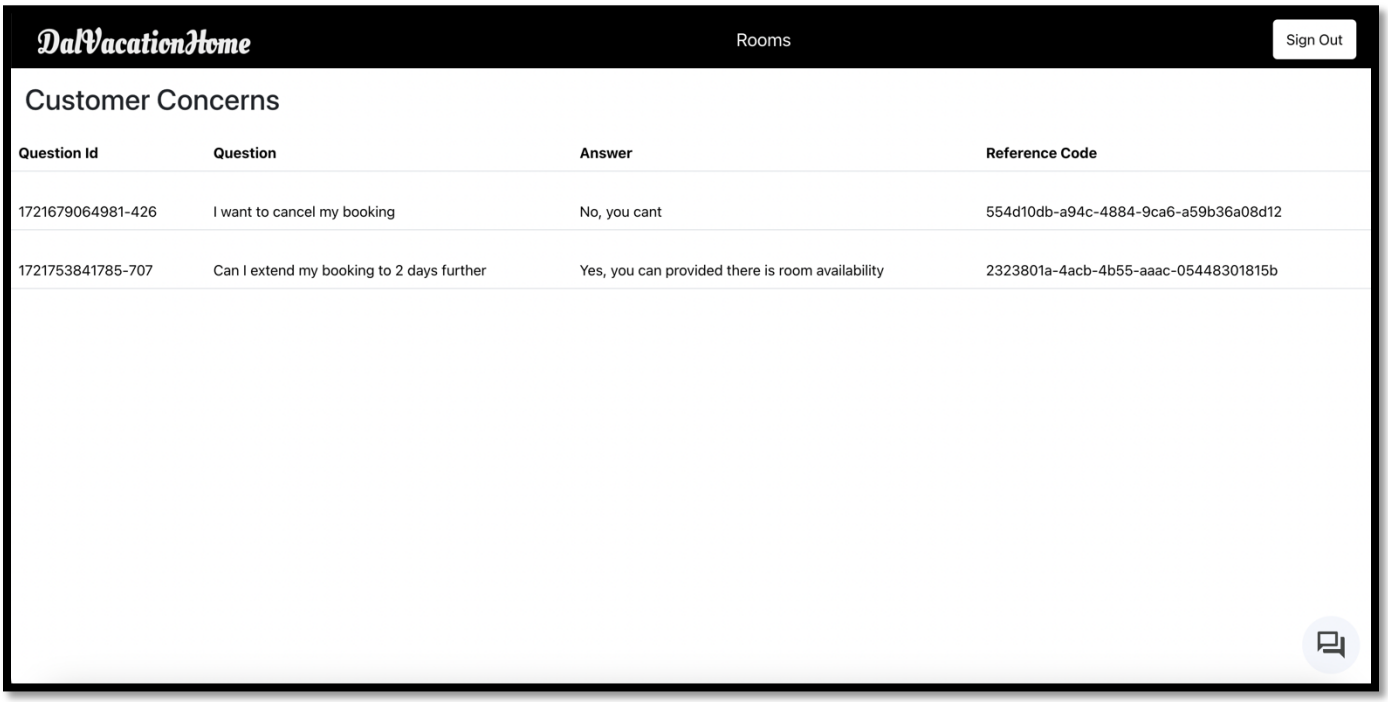

Figure 48: Customer Concern UI for a customer

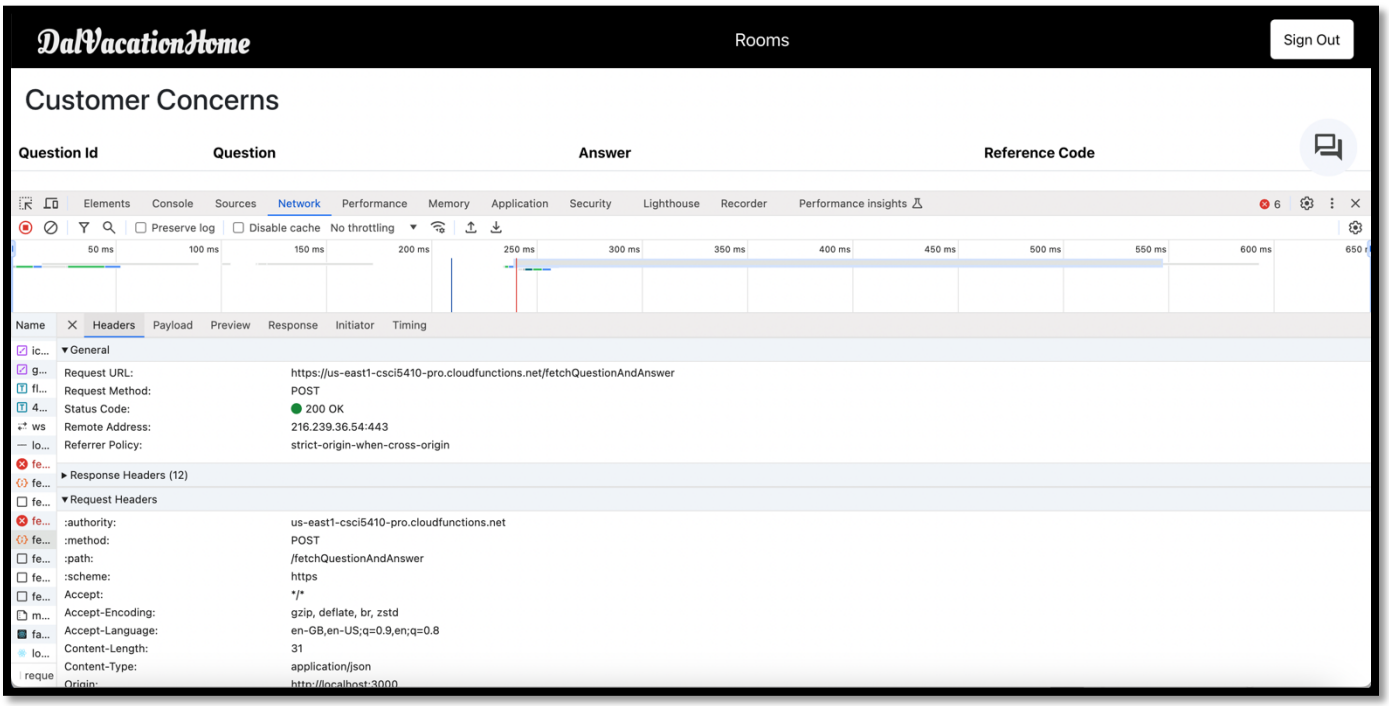

Figure 49: API Integration for fetching all raised concerns and thier answers for a customer

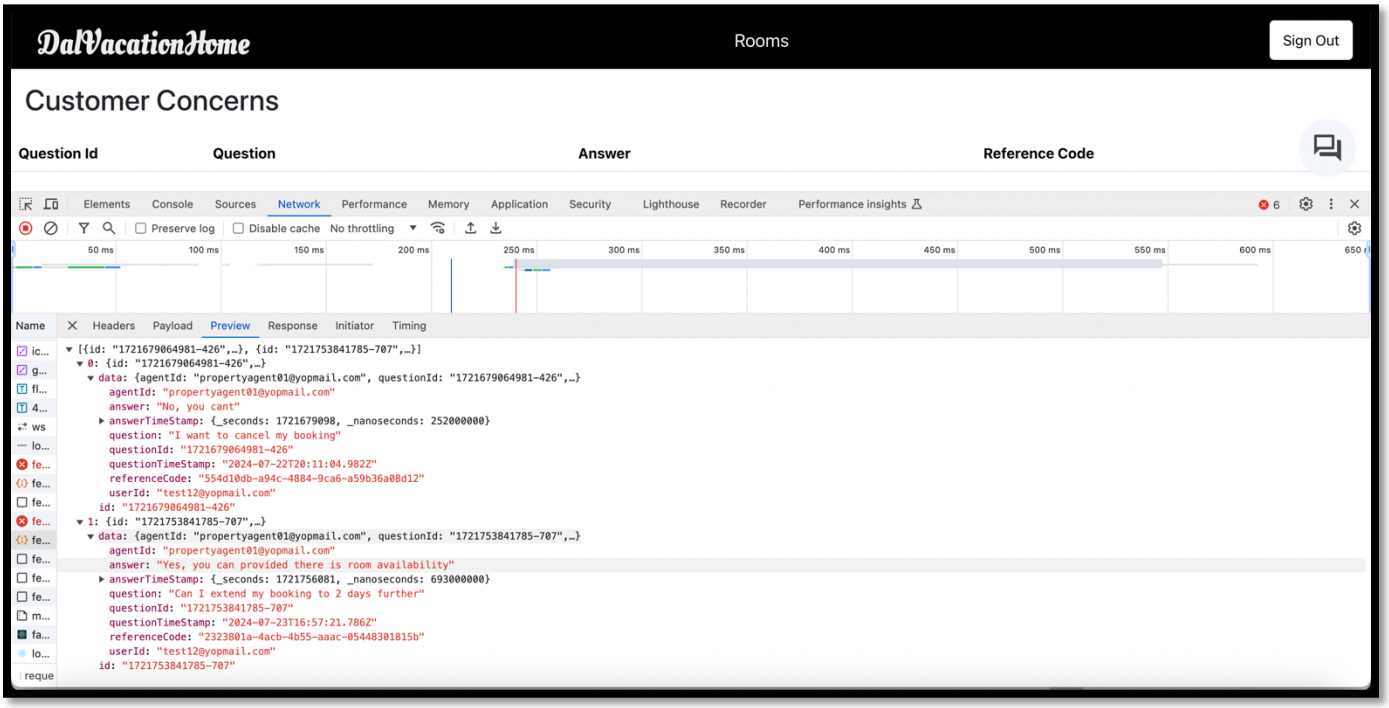

Figure 50: API response for fetching all concerns and answers

Module: Notifications

Room Booking and API integration

Testcase 1: Successful Booking of Room

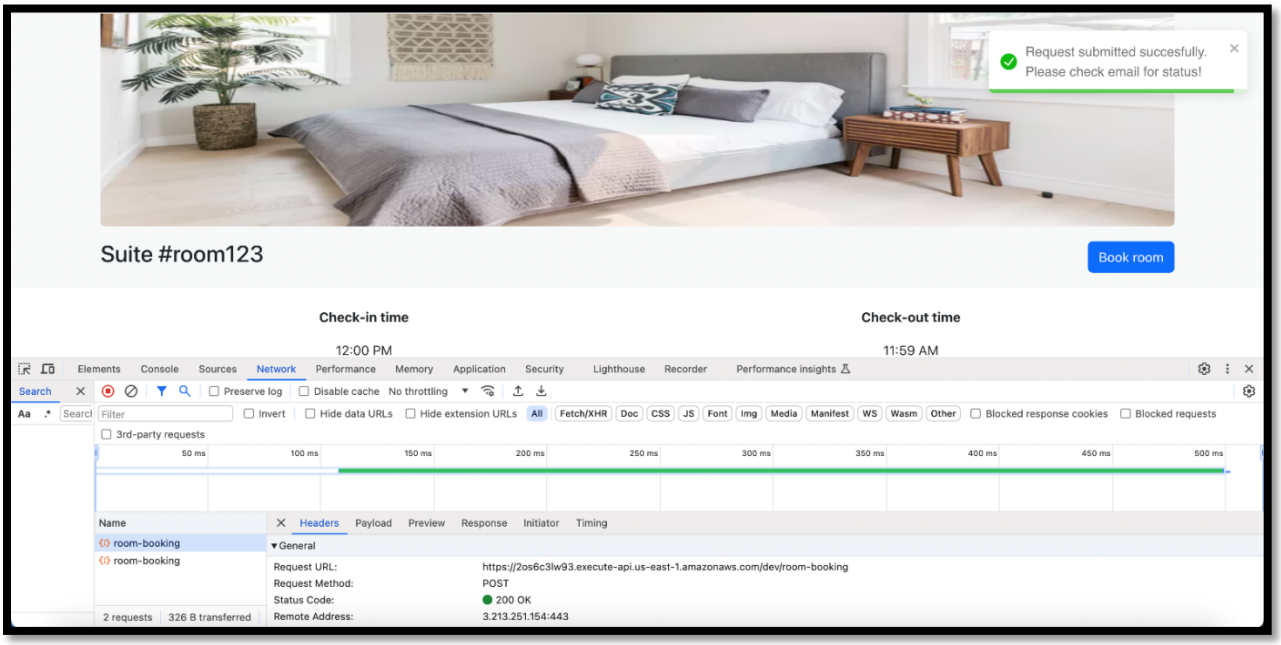

Figure 51: Room Booking API integration

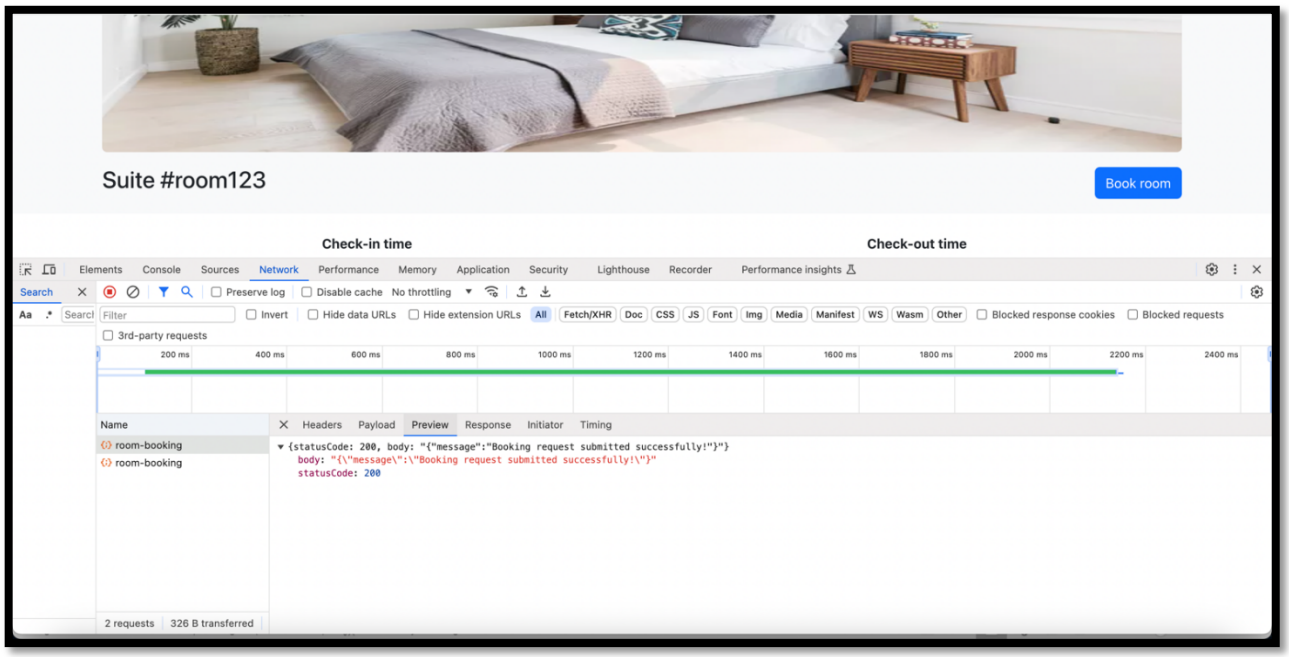

Figure 52: Room Booking API response

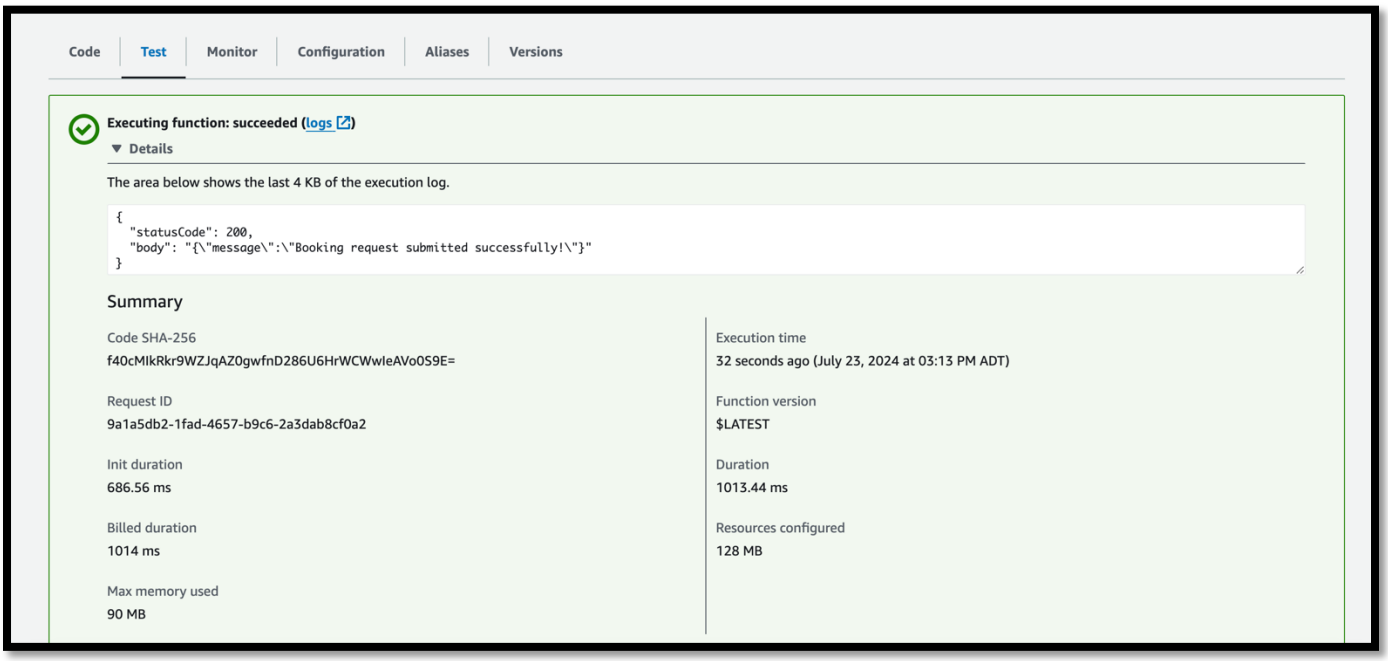

Figure 53: Lambda testing for pushing a booking request to SQS

Testcase 2: Booking Request Confirmation

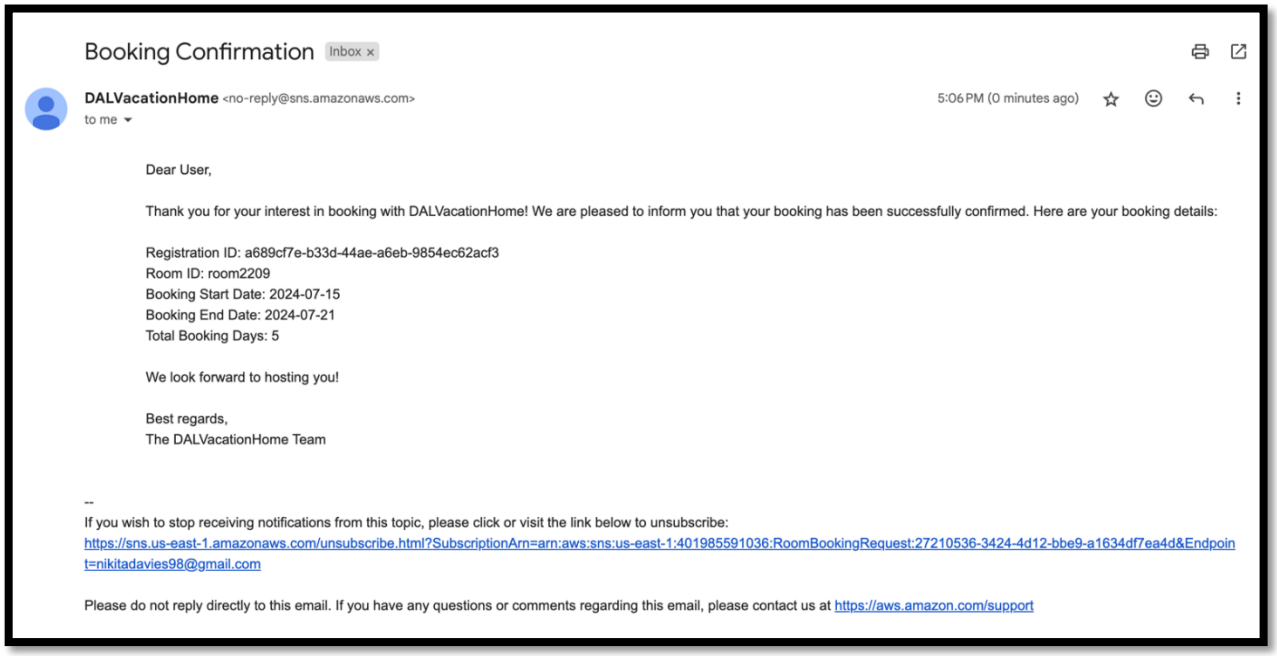

Figure 54: Email for booking Confirmation

Testcase 3: Booking Request Rejection

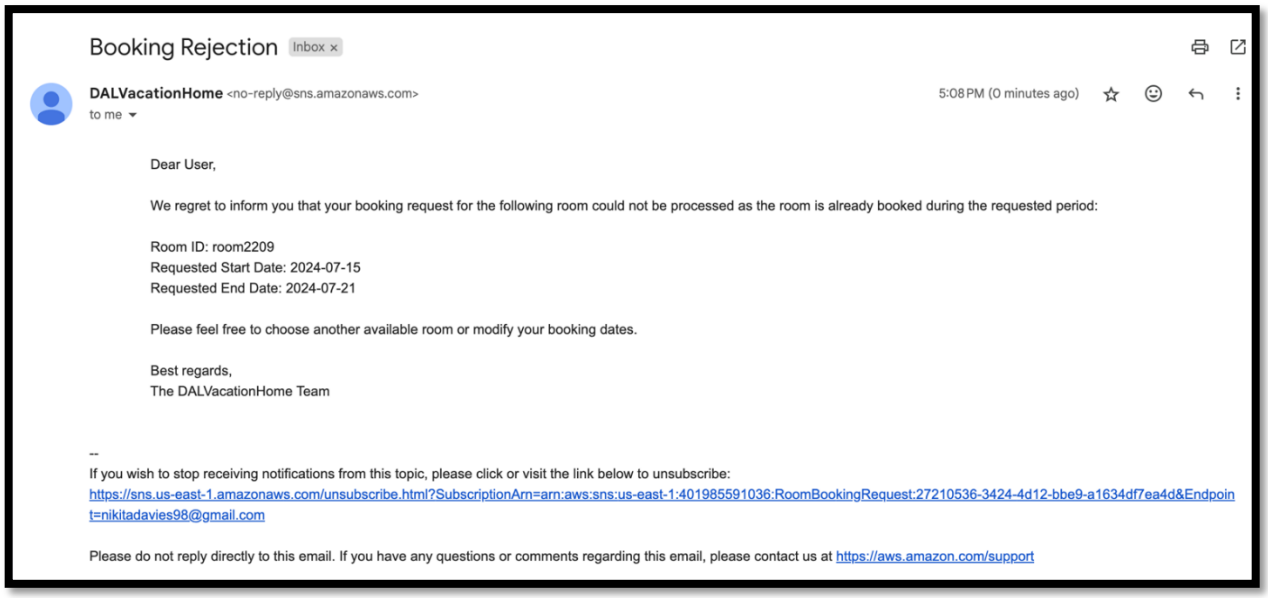

Figure 55: Email notification for Booking Rejection

Authentication module test cases:

Test case 1:

User sign up data entry before email verification

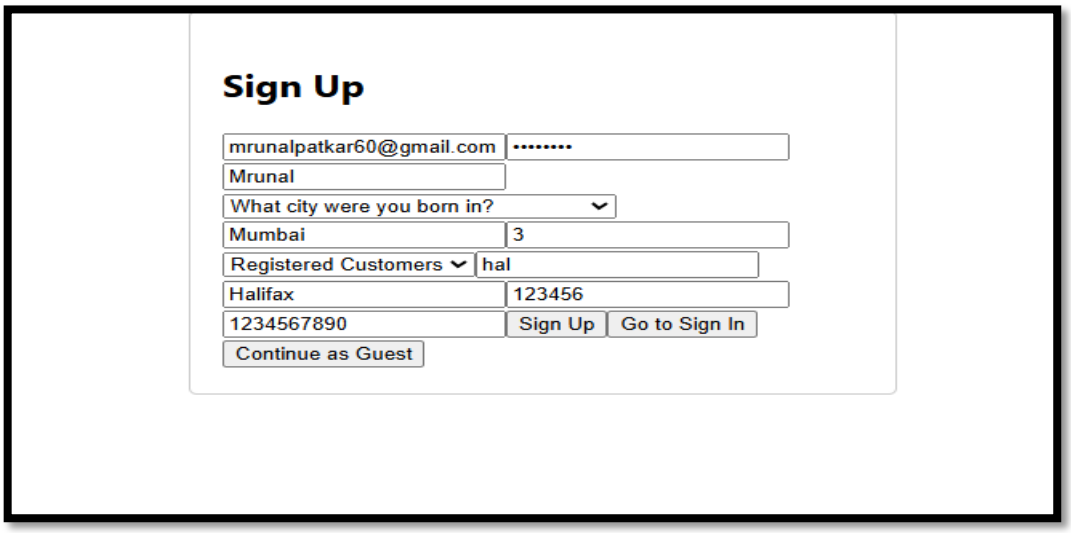

Figure 56. User sign up form.

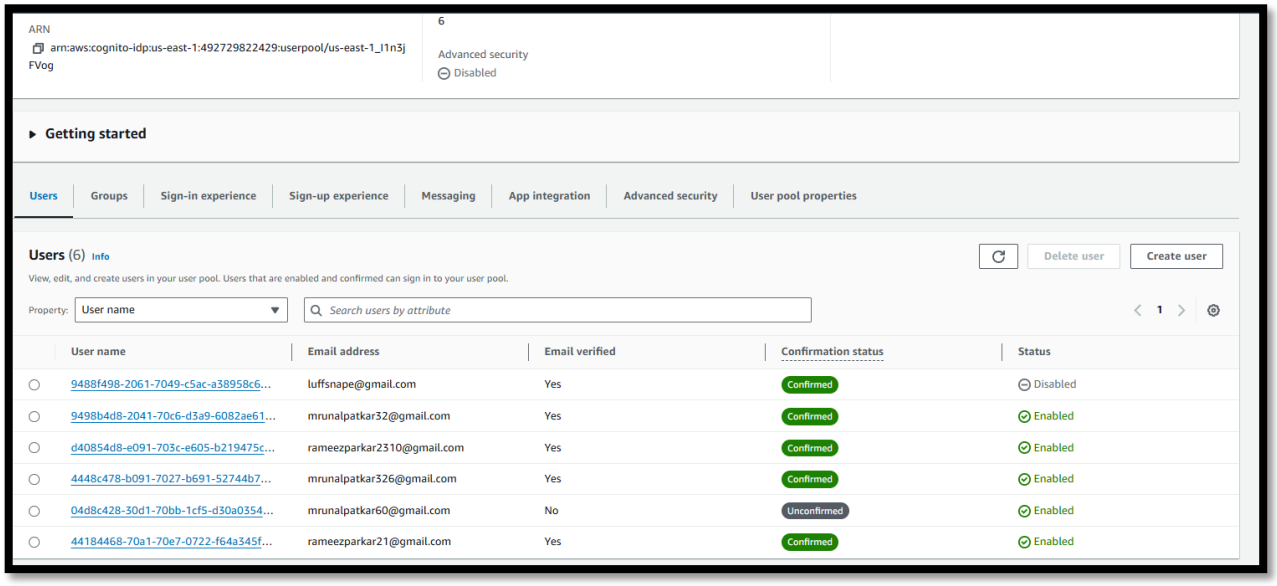

Figure 57. New user is inserted in the Cognito User pool(unverified)

Test case 2:

User sign-up data entry after email verification

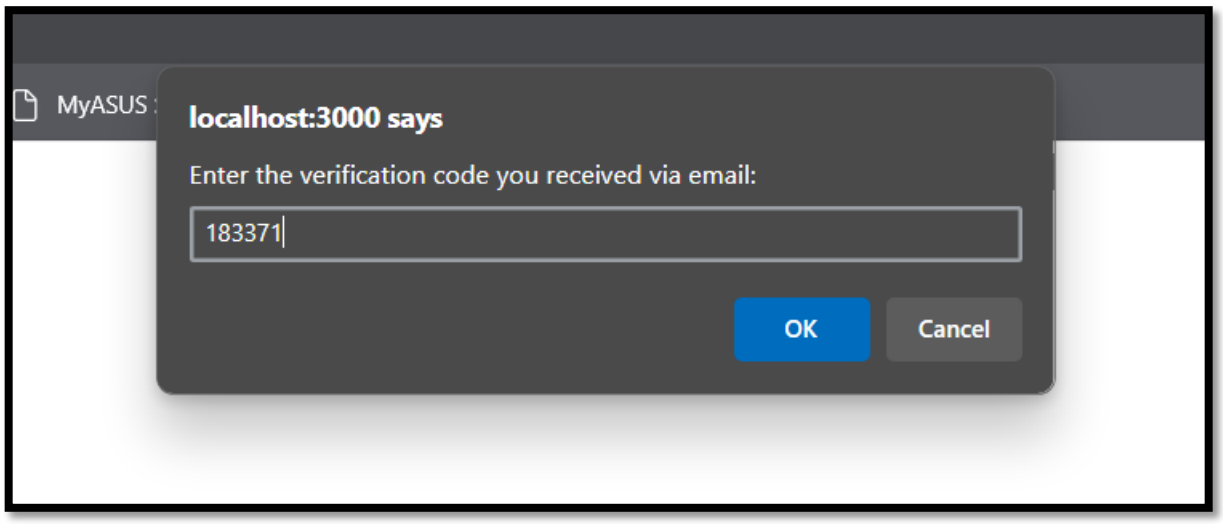

Figure 58: Verification code inserted.

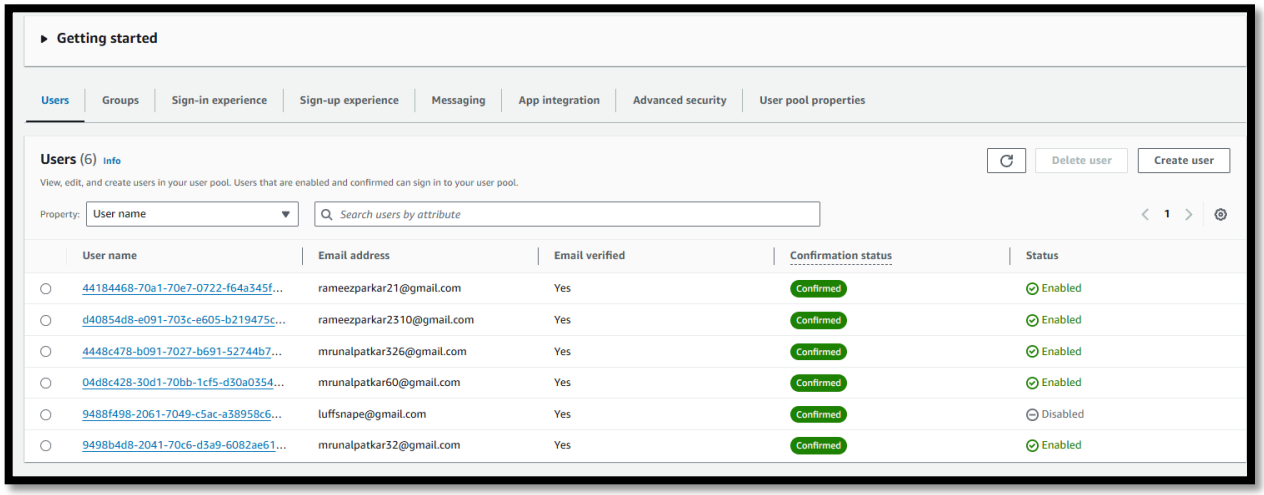

Figure 59: Email verified in cognito

Test case 3:

User Signed up successfully (Welcome mail triggered)

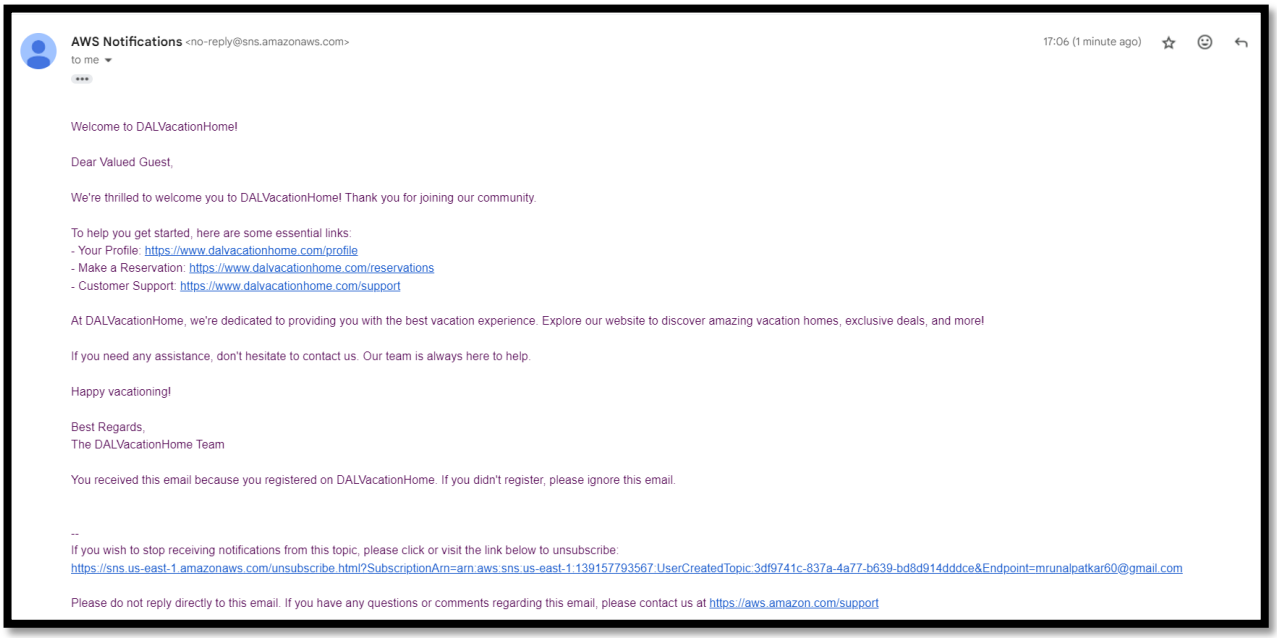

Figure 60: Welcome mail is sent after successful sign up

Welcome mail (Sign-up) Lambda manual testing

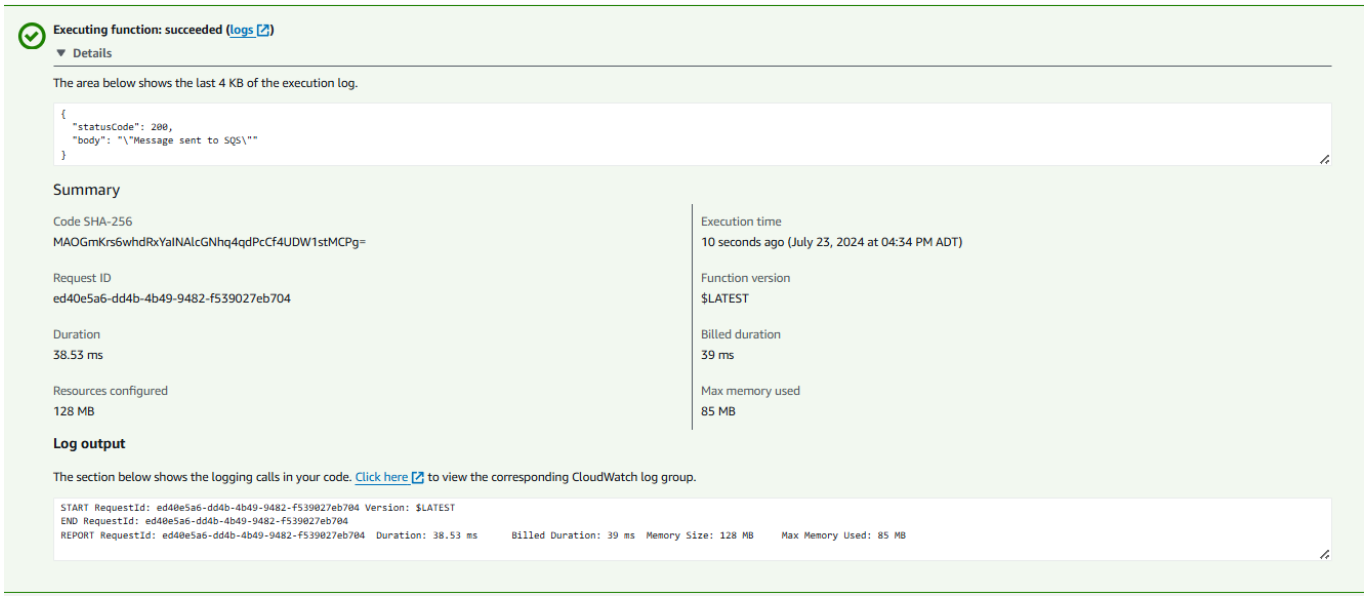

Figure 61: 4Register user notification lambda testing

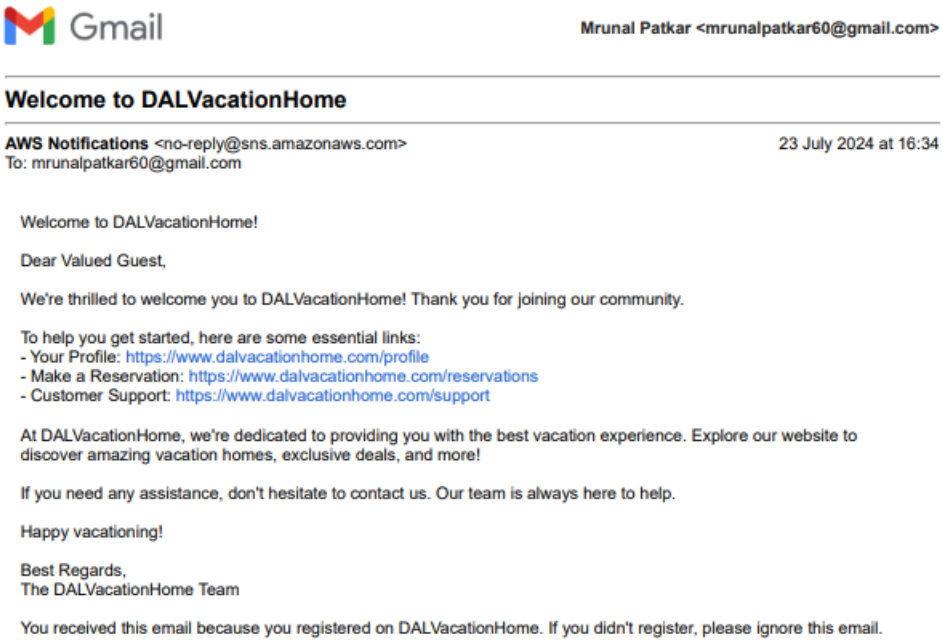

Figure 62: Welcome mail triggered by the manual testing

Login mail (Sign in) Lambda manual testing

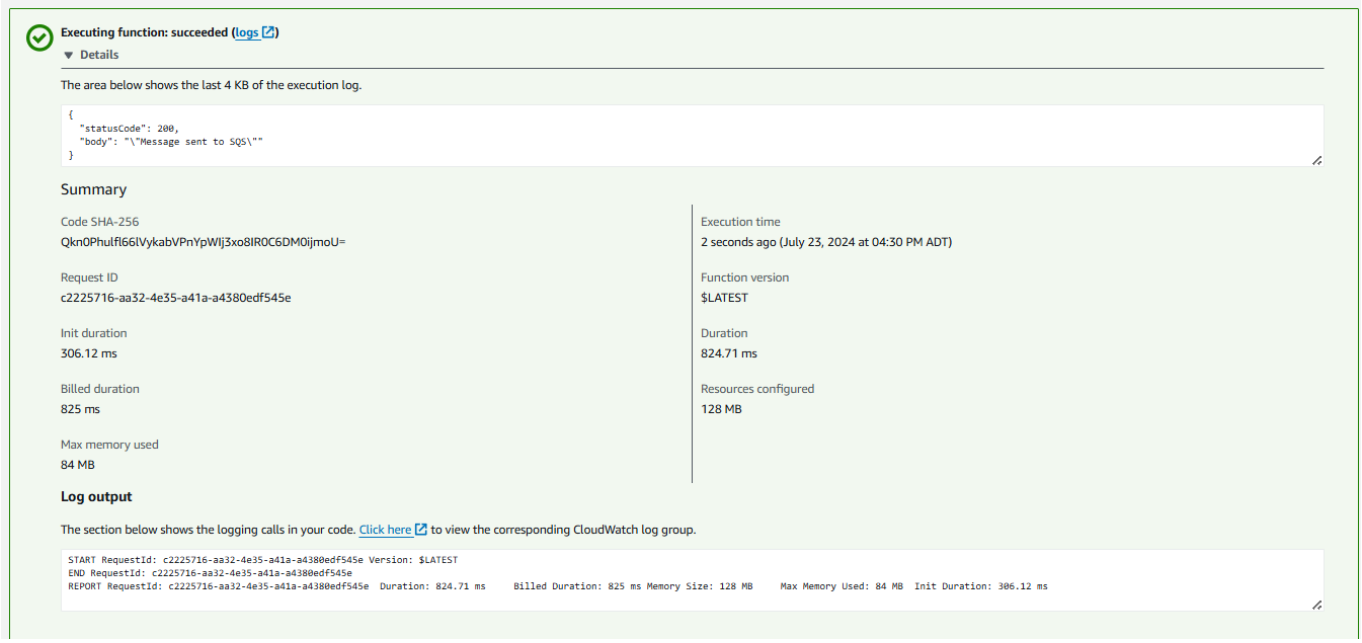

Figure 63: Login notification lambda testing

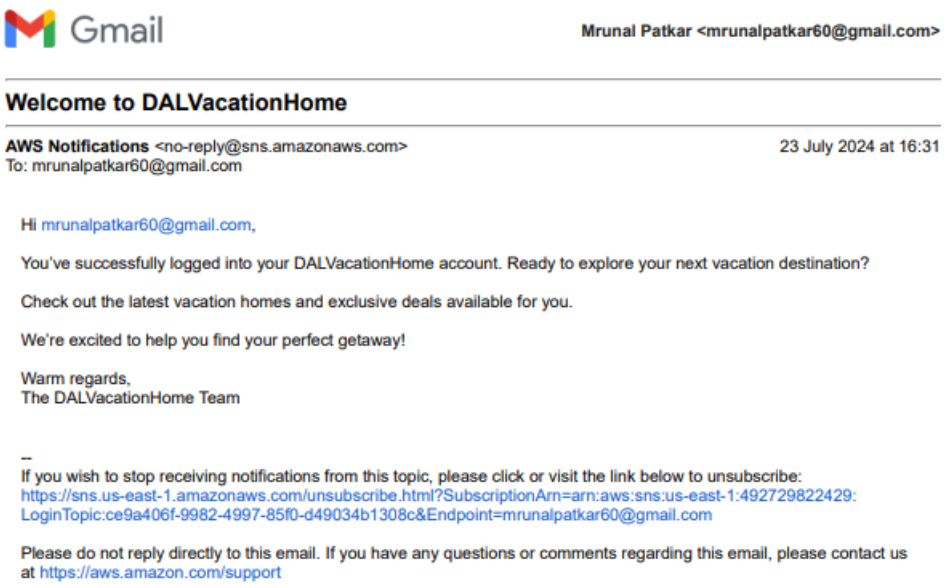

Figure 64: Login mail triggered by the manual testing.

Test case 4:

Verified users get sorted into their respective user groups in the backend.

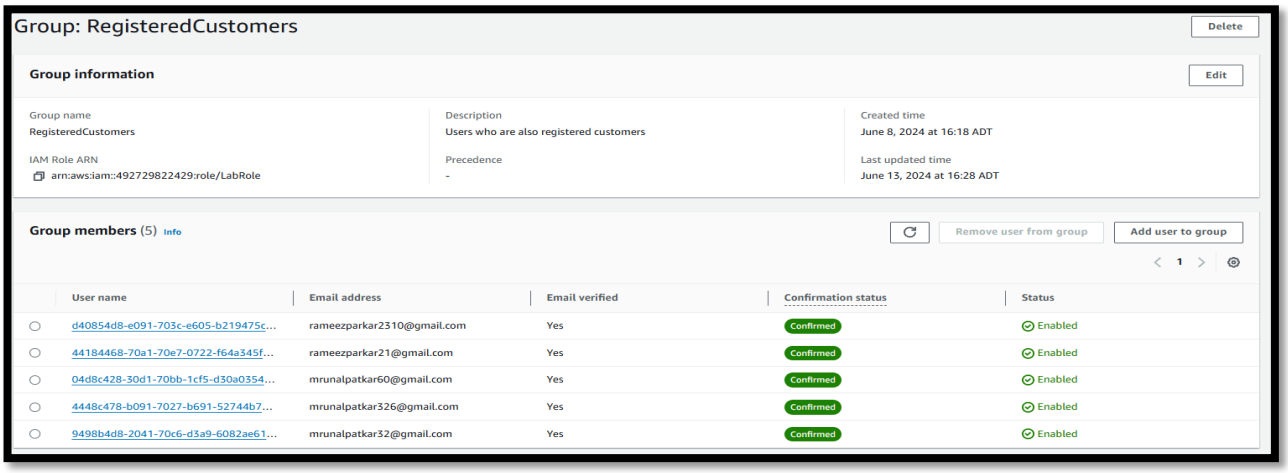

Figure 65: User sorted in respective groups

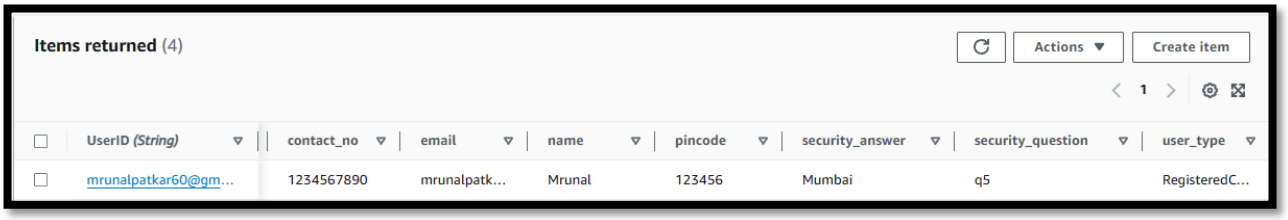

Figure 66: User data is entered in the dynamodb upon successful sign-up.

Test case 4:

2 factor authentication front end

Figure 67: 2FA Frontend component.

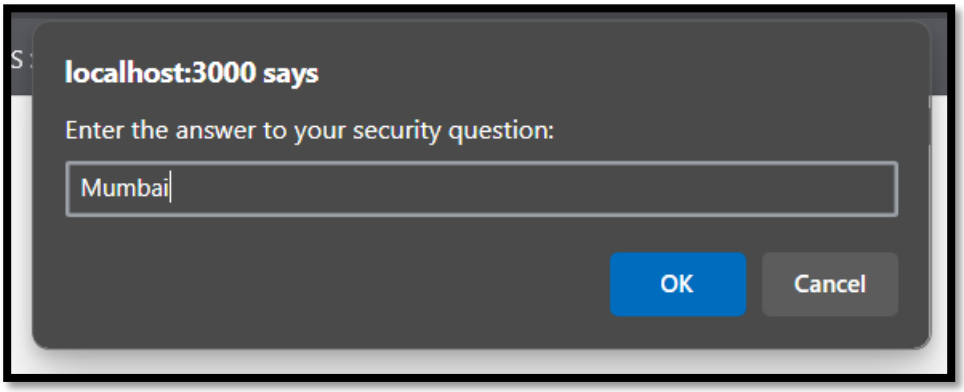

Figure 68: 2FA security answer

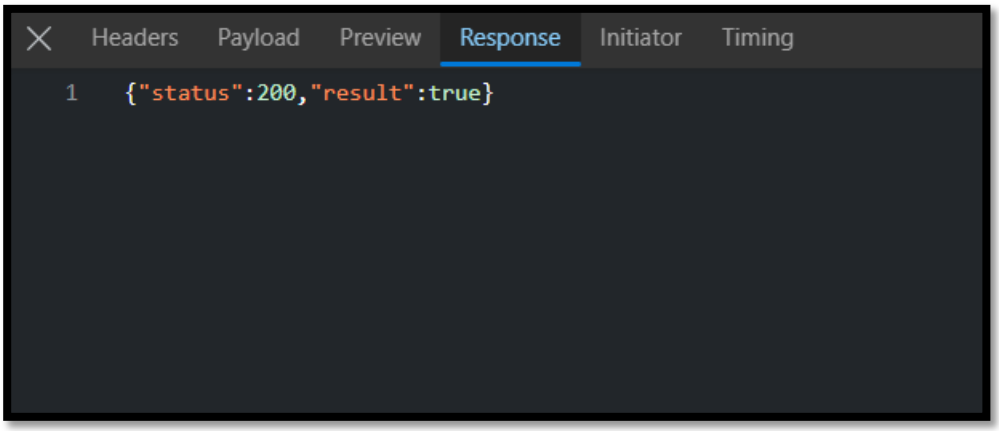

Figure 69: 2FA success response

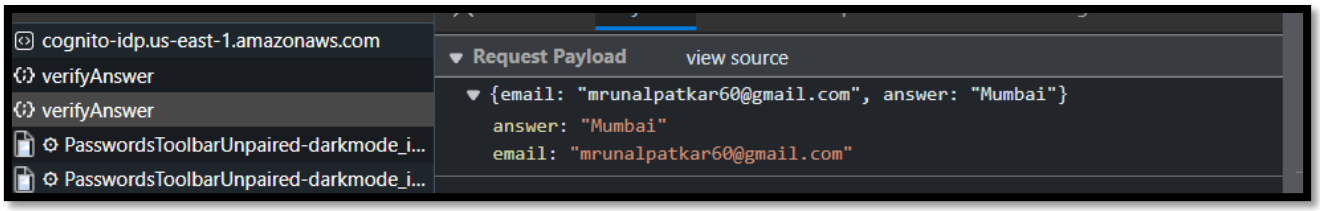

Figure 70: 2FA frontend payload.

2FA lambda manual testing

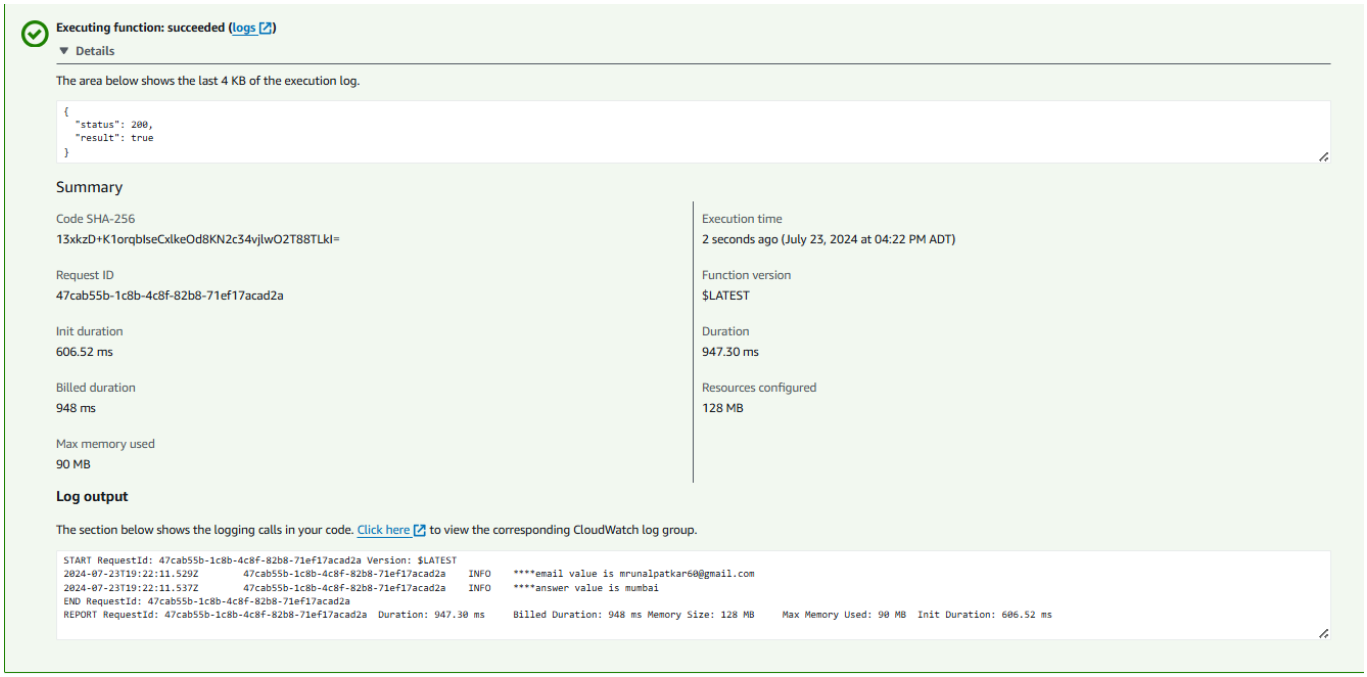

Figure 71: 2FA manual testing

Test case 5:

3 factor authentication front end:

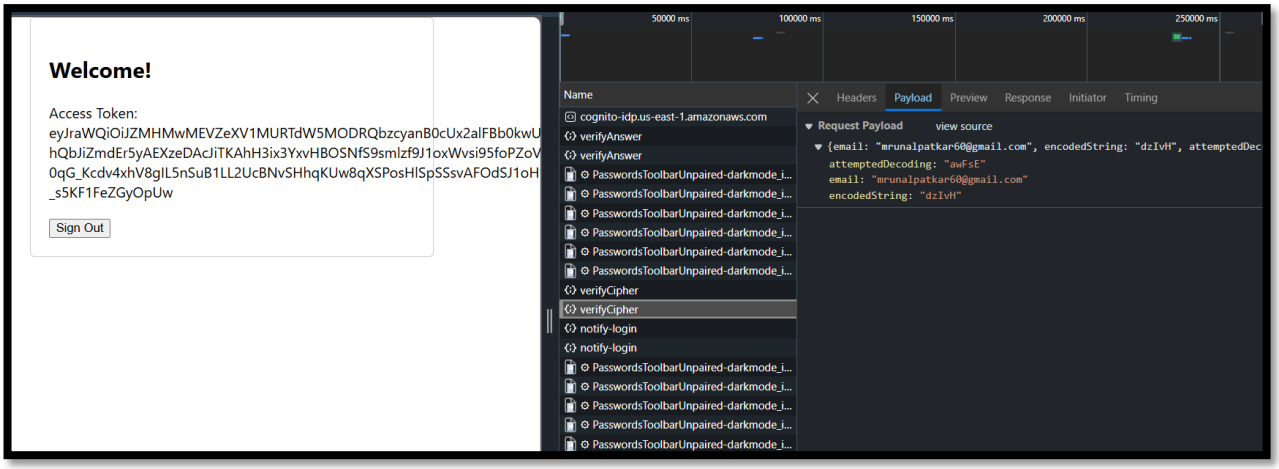

Figure 72: 3FA frontend payload

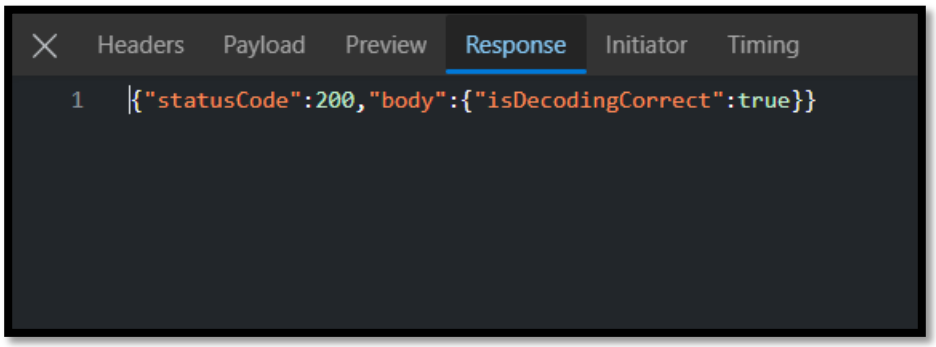

Figure 73: 3FA success response.

3FA Lambda manually tested

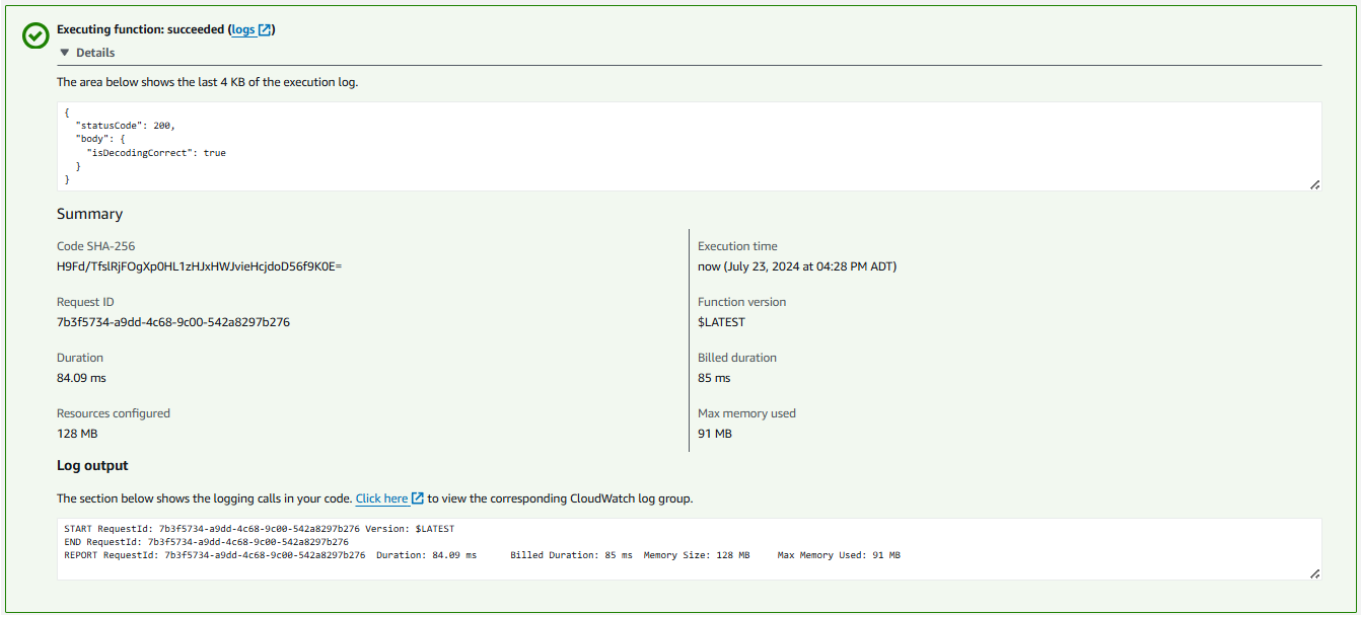

Figure 74: 3FA Manually tested

Test case 6:

Verified users get a log-in successful message when successfully logged in.

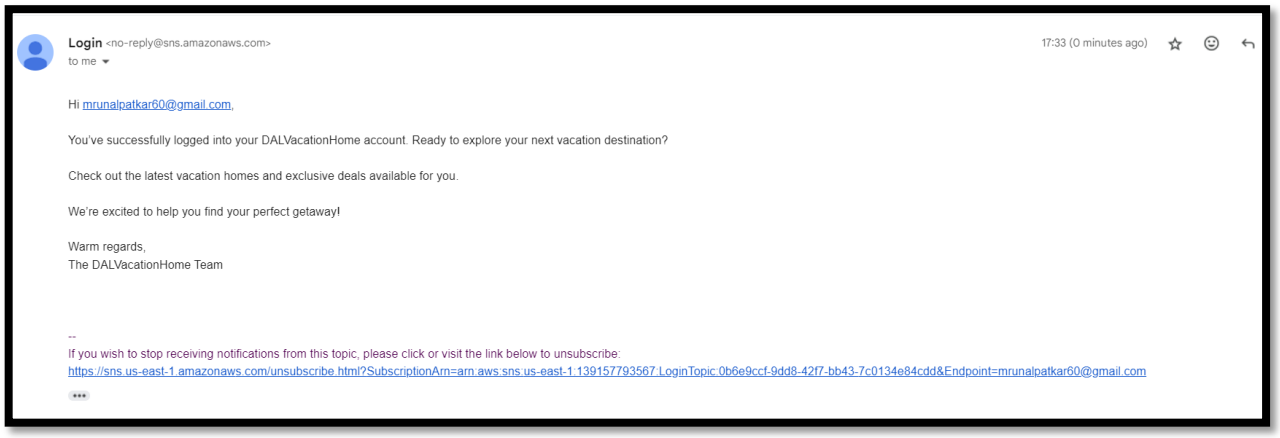

Figure 75: Login successfully mail

Fully Integrated sign-in flow:

Test case 1:

Auth token and user details gets stored in the local storage for session management

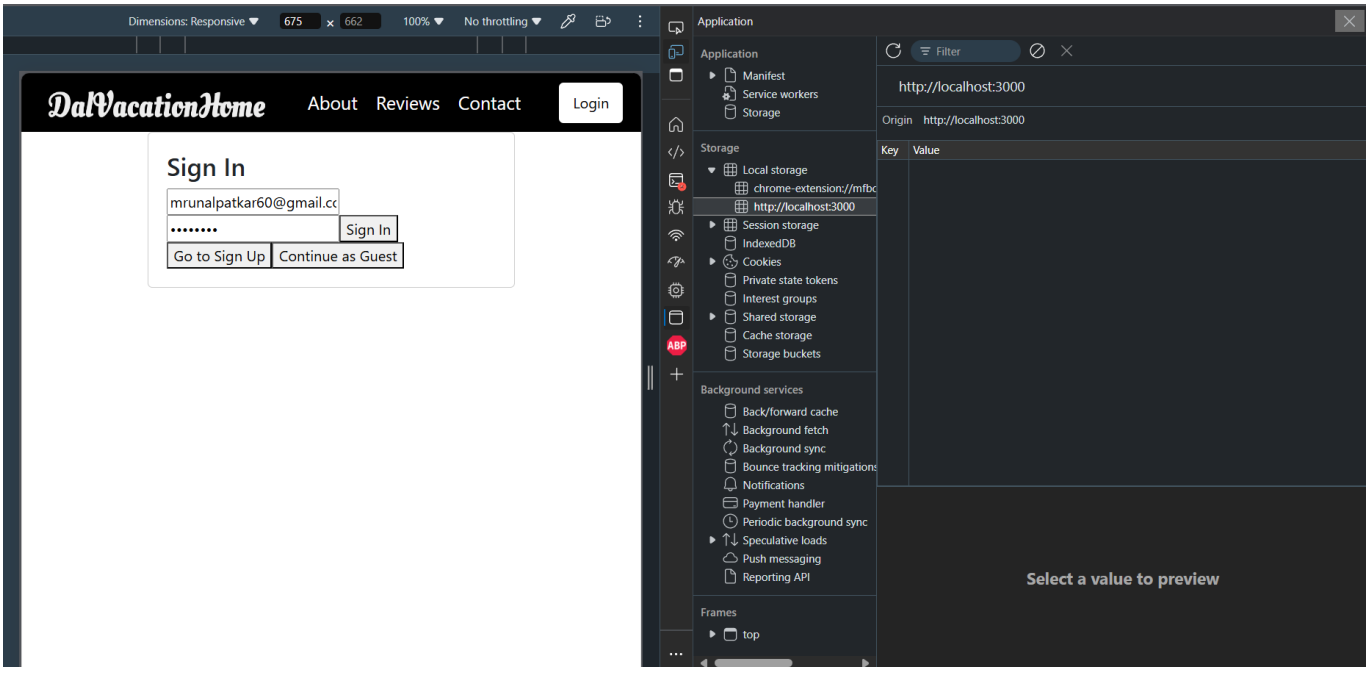

Figure 76: Empty local storage before the user signs in

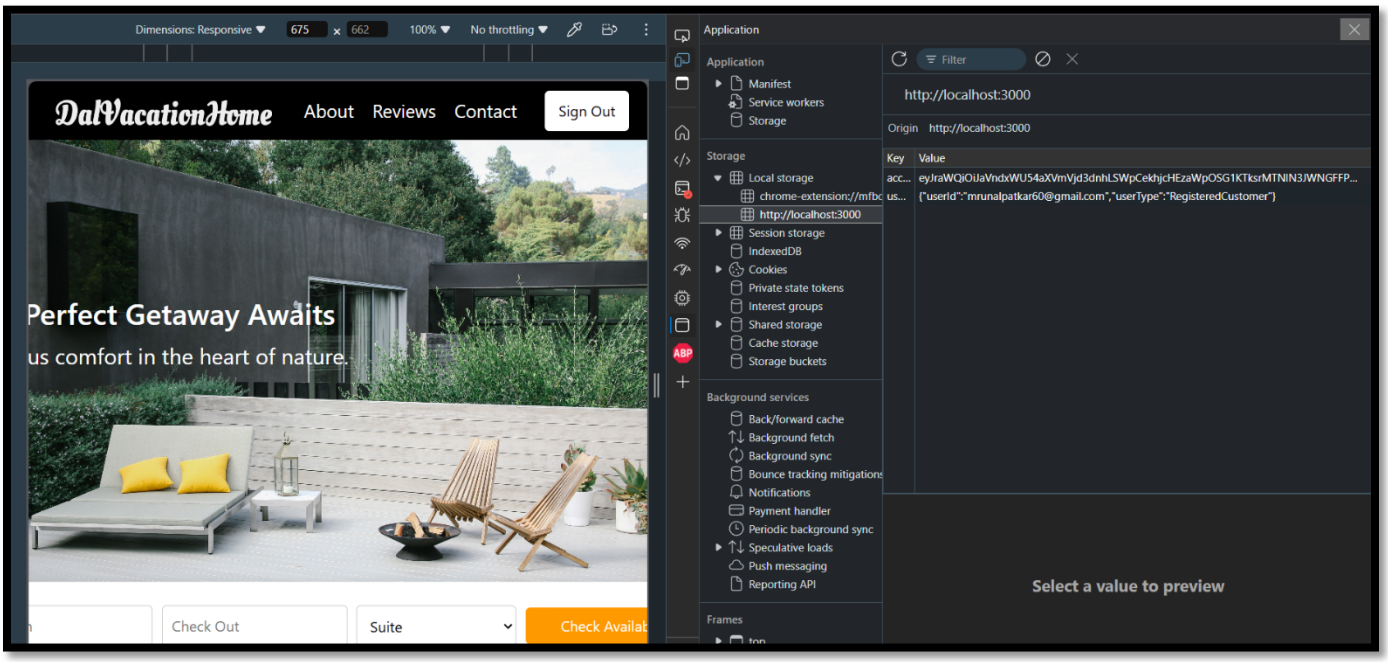

Figure 77: Auth token and user details are stored once the user signs in successfully

Test case 2:

Auth token and user details should get deleted from the local storage once the user clicks on the sign-out button

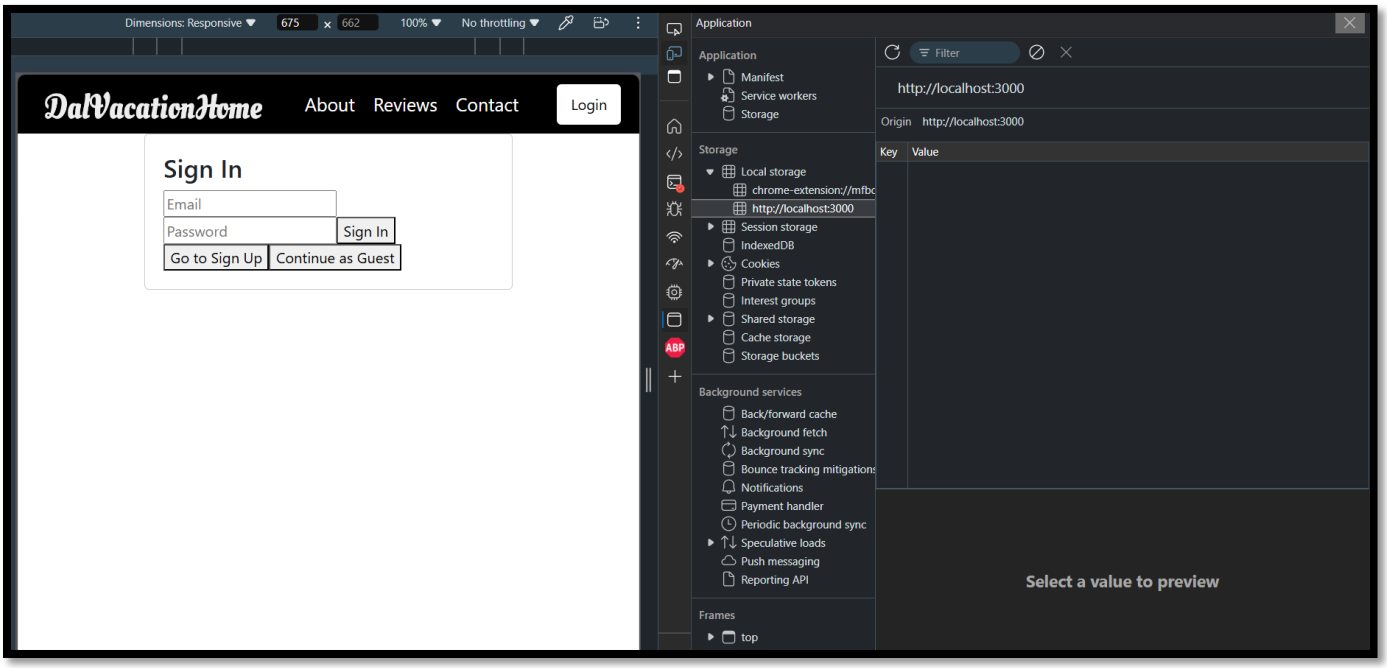

Figure 78: Auth token and User details get deleted after clicking on the "Sign-out" button

Test case 3:
Log table gets populated once a user signs in successfully (log details to be used for analytics)

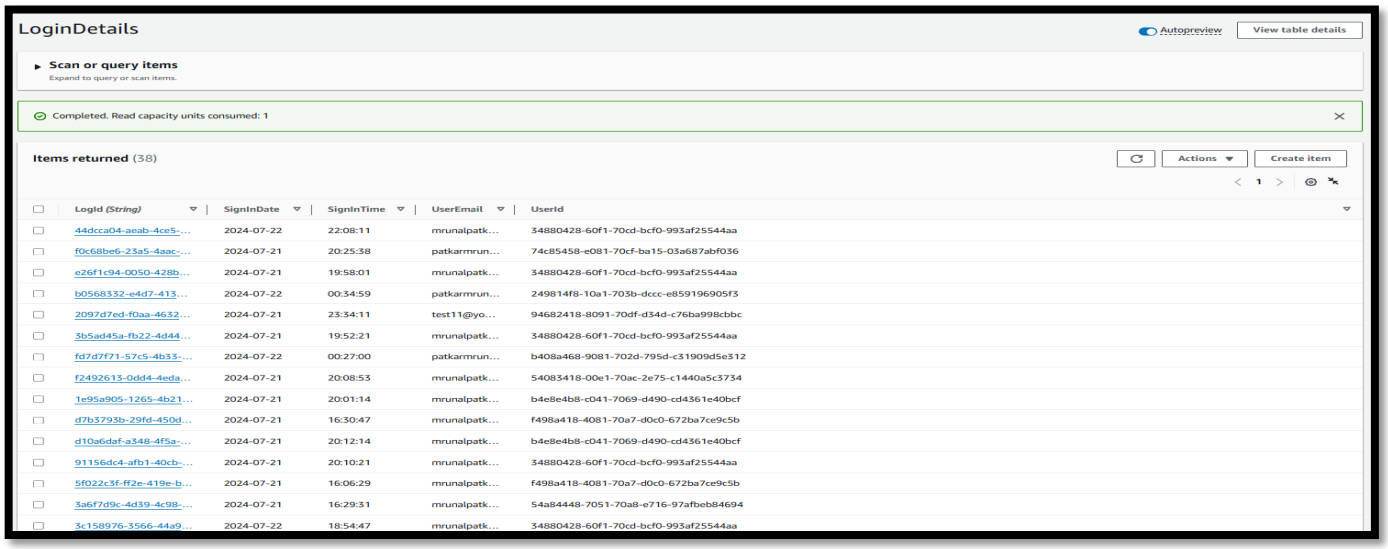

Figure 79: Sign in the activity log table

Test case 4:
If the user signs up as a property agent then they get redirected to the Property agent dashboard

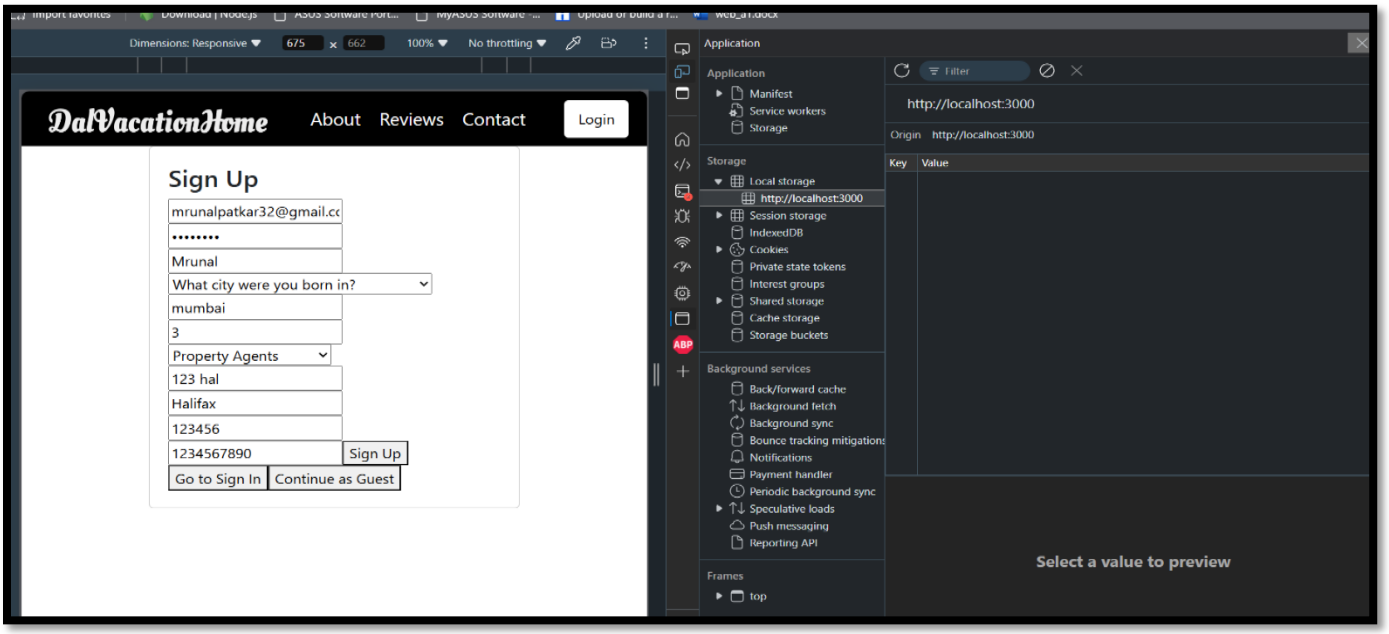

Figure 80: The user signs up as a property agent

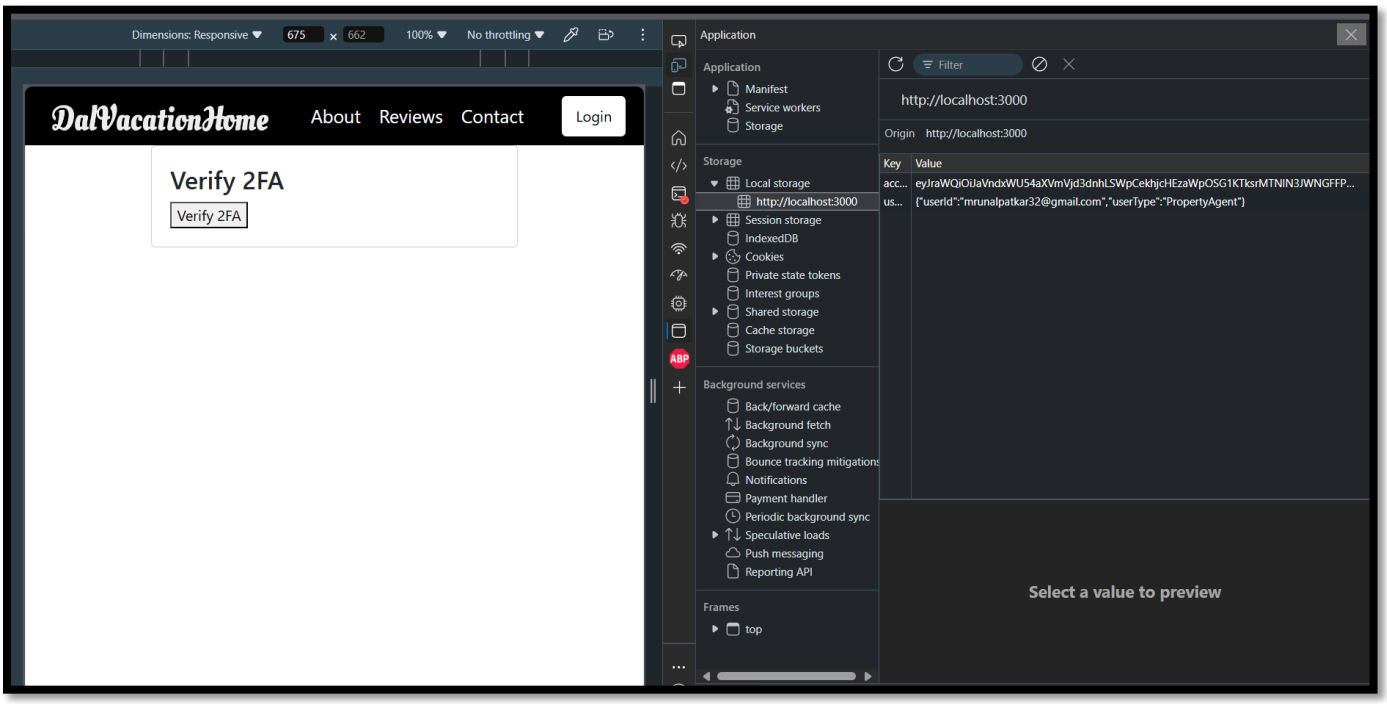

Figure 81: Successful sign-up

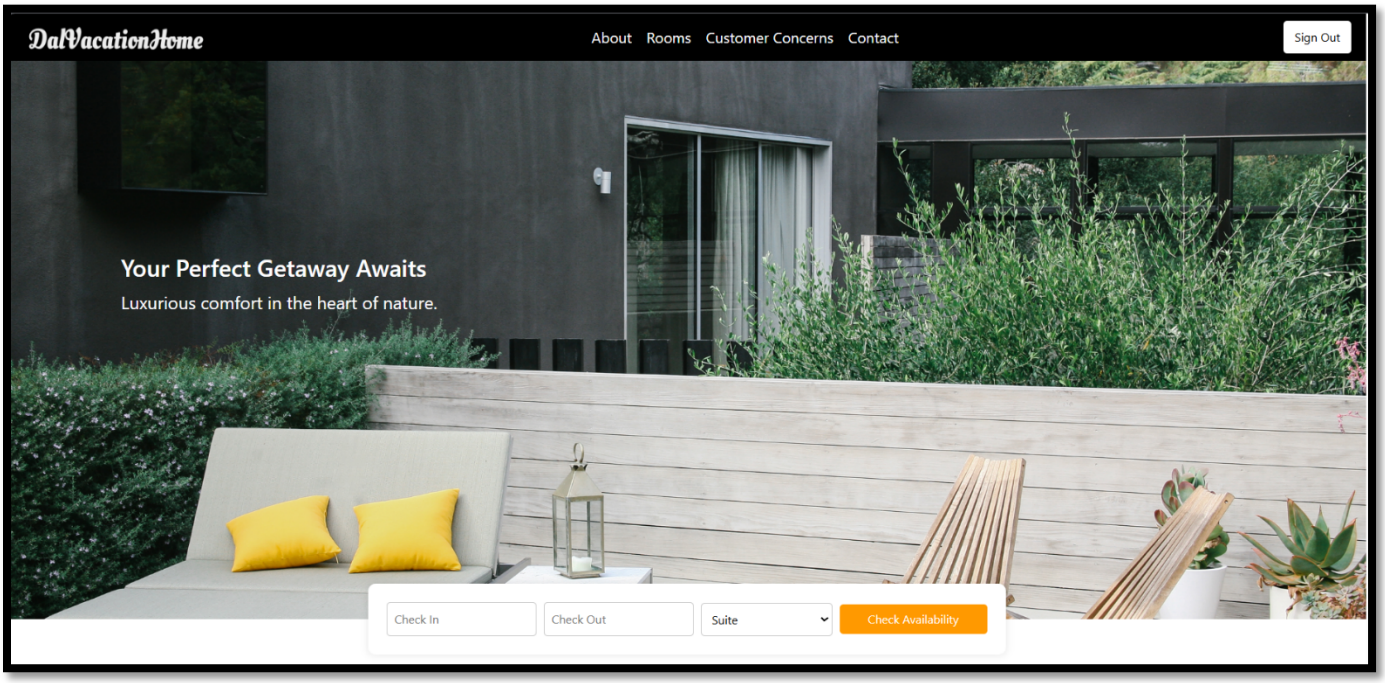

Figure 82: Property agent dashboard.

Fetching booking details based using a booking code:

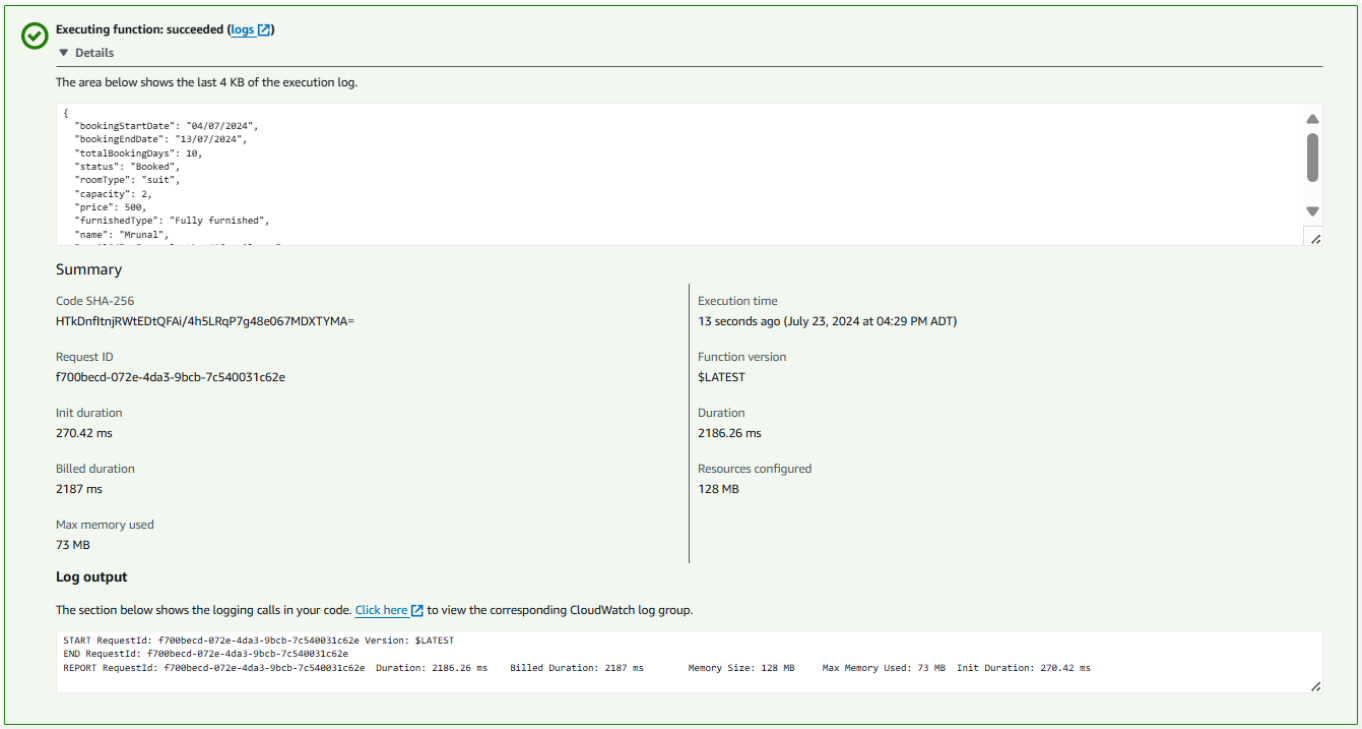

Figure 83: Manually testing the lambda to fetch booking details using the booking codes.

Individual Contribution

Table 2: Individual Contribution

Team Member	Contribution
Nikita Davies	• Explored LookerStudio and the options to automate the workflow. • Implemented the automated flow of LookerStudio and its integration and connectivity from AWS to GCP. • Implemented the frontend for customer-concerns and handled the frontend integration for GCP pub/sub message passing. • Created CloudFormation Scripts for DynamoDB, Created CloudFormation Scripts for SQS-SNS Room Booking Flow, Lambda functions and API Gateway. • Integrated all the AWS and GCP services into the frontend along with its API integration.
Mrunal Mangesh Patkar	• Explored Google Cloud run, AWS CloudFormation, and how to automate resource provisioning and deployment. • Developed Cognito (including post auth, post confirmation lambda integration and FE app integration) CloudFormation script. • Developed SQS, and SNS CloudFormation scripts for Login and Sign-up notification feature. • Developed the CloudFormation script for 2nd and 3rd-factor authentication Lambda endpoints. • Integrated the Sign and sign up with the front along with dynamic switching of the dashboard based on the “User type”. • Created the front end for room creation and editing.
Rameez Parkar	• Explored the virtual assistant module and researched on Dialogflow CX. • Planned the flows of each context and how it is triggered by user.Created the pages for HowToRegister, HowToBook, CustomerConcerns, BookingDetails, CheckTicketResolutionStatus virtual assistant intents. • Created the lambdas associated with the above pages for webhooks in order to fetch required data. • Helped resolve issues in message passing pub/sub module. • Created the final architecture of the application.
Axata Darji	• Explored and worked on module 3 Pub/Sub and its triggers. • Explored the connectivity of AWS from GCP. Explored on connecting, storing and retrieving from firestore using Cloud functions. • Worked on some cloud formation scripts for the lambdas which is created by me. • Created the structure for the sprint 3 report and added information about module 3. Added all important module pseudocode. • Scheduling and maintaining meeting logs and recordings.

Project Meeting Logs

Table 3: Project Meeting Logs

Meeting Date	Meeting Time (Start)	Meeting Time (End)	Meeting Place	Outcomes	Attendees	Recording link
17/05/2024	6:00PM	6:30PM	Online	Introduction and plan for sprint1	All present	https://dalu-my.sharepoint.com/personal/ax583820_dal_ca/_layouts/15/stream.aspx?id=%2Fpersonal%2Fax583820%5Fdal%5Fca%2FDocuments%2FRecordings%2FCSCI%205410%2D%20Team%206%20Introduction%20call%2D20240517%5F180313%2DMeeting%20Recording%2Emp4&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview%2Ed151eddf%2D3ac8%2D431c%2Db50a%2D4acdfd69b525&ga=1
22/05/2024	1:30PM	2:30PM	Online	Plan for sprint 1 report and update on the individual task	All present	Meeting in _General_-20240522_140828-Meeting Recording.mp4 (sharepoint.com)
26/05/2024	7:00PM	7:30PM	Online	Discussed on individual task update and plan for final project report for sprint 1	All present	CSCI 5410- Team 6 Sprint 1 report meeting-20240526_190119-Meeting Recording.mp4 (sharepoint.com)
28/05/2024	2:00PM	3:30PM	Online	Finalized the architecture and the project planning	All present	https://dalu.sharepoint.com/teams/CSCI5410_Serverless_Team6/_layouts/15/stream.aspx?id=%2Fteams%2FCSCI5410%5FServerless%5FTeam6%2FShared%20Documents%2FGeneral%2FMeeting%20Recordings%2FCSCI%205410%2D%20Group%206%20sprint1%20report%20final%20call%2D20240528%5F143126%2DMeeting%20Recording%2Emp4&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview%2E46f1b306%2Df018%2D4077%2Da481%2D9744141b3111
30/05/2024	3:00PM	4:00 PM	online	Sprint 2 planning and user story assignment	All present	https://dalu.sharepoint.com/teams/CSCI5410_Serverless_Team6/Shared Documents/General/../../../../:v:/r/t

						eams/CSCI5410_Serverless_Team6/Shared%20Documents/General/Meeting%20Recordings/CSCI%205410-%20GroupProject-Sprint2%20planning-20240530_150058-Meeting%20Recording.mp4?csf=1&web=1&e=Pz4B8W
09/06/2024	9:00 AM	9:30 AM	online	Discussion of database design and plan for upcoming week	All present	https://dalu.sharepoint.com/teams/CSCI5410_Serverless_Team6/Shared Documents/General/../../../../:v:/r/teams/CSCI5410_Serverless_Team6/Shared%20Documents/General/Meeting%20Recordings/CSCI%205410-%20GroupProject-Sprint2%20planning-20240530_150058-Meeting%20Recording.mp4?csf=1&web=1&e=Pz4B8W
18/06/2024	1:30 PM	2:30 PM	online	Discussion on how to share common services if multiple people working on same feature	All present	https://dalu.sharepoint.com/:v:/r/teams/CSCI5410_Serverless_Team6/Shared%20Documents/General/Meeting%20Recordings/CSCI%205410-%20Project%20development%20approach%20discussion-20240618_134317-Meeting%20Recording.mp4?csf=1&web=1&e=xfAye3
25/06/2024	2:30 PM	3:00 PM	online	Discussion on sprint 2 individual updates and deliverables	All present	https://dalu.sharepoint.com/:v:/r/teams/CSCI5410_Serverless_Team6/Shared%20Documents/General/Meeting%20Recordings/CSCI%205410-%20Weekly%20project%20meeting-20240625_143106-Meeting%20Recording.mp4?csf=1&web=1&e=oItrCF
04/07/2024	4:00 PM	4:20PM	online	Discussion on update for sprint 2 report	All present	https://dalu.sharepoint.com/:v:/r/teams/CSCI5410_Serverless_Team6/Shared%20Documents/General/Meeting%20Recordings/CSCI%205410-%20Sprint%202%20Update%20and%20Plan%20for%20report-20240704_160120-Meeting%20Recording.mp4?csf=1&web=1&e=XMbi93

14/07/2024	9:00AM	9:30AM	online	Discussion on sprint 3 planning	All present	https://dalu.sharepoint.com/:v:/r/teams/CSCI5410_Serverless_Team6/Shared%20Documents/General/Meeting%20Recordings/CSCI%205410-%20Development%20time-20240714_090333-Meeting%20Recording.mp4?csf=1&web=1&e=PEsHB7
15/07/2024	2:45PM	4:45PM	online	Development time and doubt solving session	All present	https://dalu.sharepoint.com/:v:/r/teams/CSCI5410_Serverless_Team6/Shared%20Documents/General/Meeting%20Recordings/CSCI%205410%20-%20Serverless%20project-20240715_144351-Meeting%20Recording.mp4?csf=1&web=1&e=pCR6c8
17/07/2024	6:30PM	7:45PM	Online	Discussion on pending items and overall built architecture	All present	https://dalu.sharepoint.com/:v:/r/teams/CSCI5410_Serverless_Team6/Shared%20Documents/General/Meeting%20Recordings/CSCI%205410%20-%20Serverless%20project%20Sprint%20call%20preparation-20240717_184441-Meeting%20Recording.mp4?csf=1&web=1&e=jlrhmO
21/07/2024	9:00AM	9:30AM	Online	Discussion on the updates for all modules and preparation for sprint 3 report and all other pending items	All present	CSCI 5410 - Sprint 3 final call-20240721_090144-Meeting Recording.mp4
23/07/2024	2:00PM	2:30PM	Online	Sprint 3 report updates and pending items	All present	CSCI 5410- Project sprint 3 report-20240723_141216-Meeting Recording.mp4

Project management tool

We have decided to use Gitlab as our project management tool.

1. Project issues spread across 6 modules

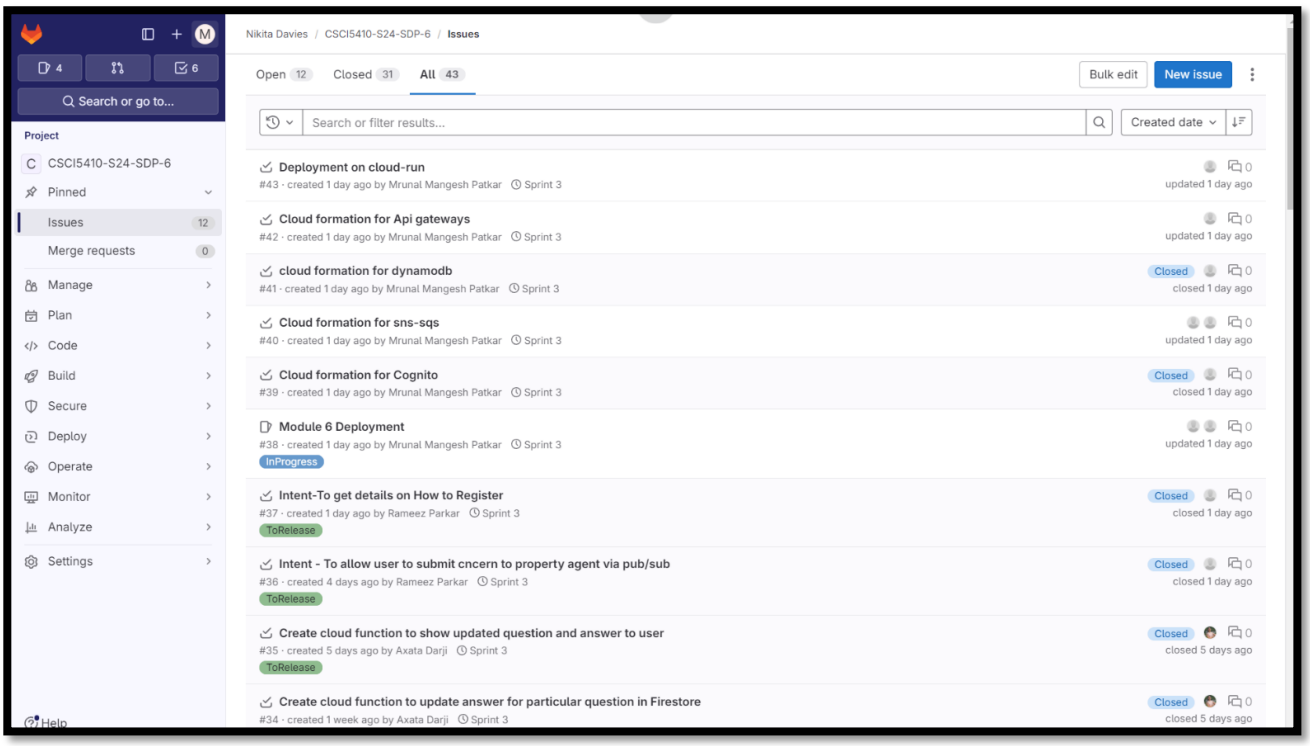

Figure 84: All issues

2. Closed Project issues

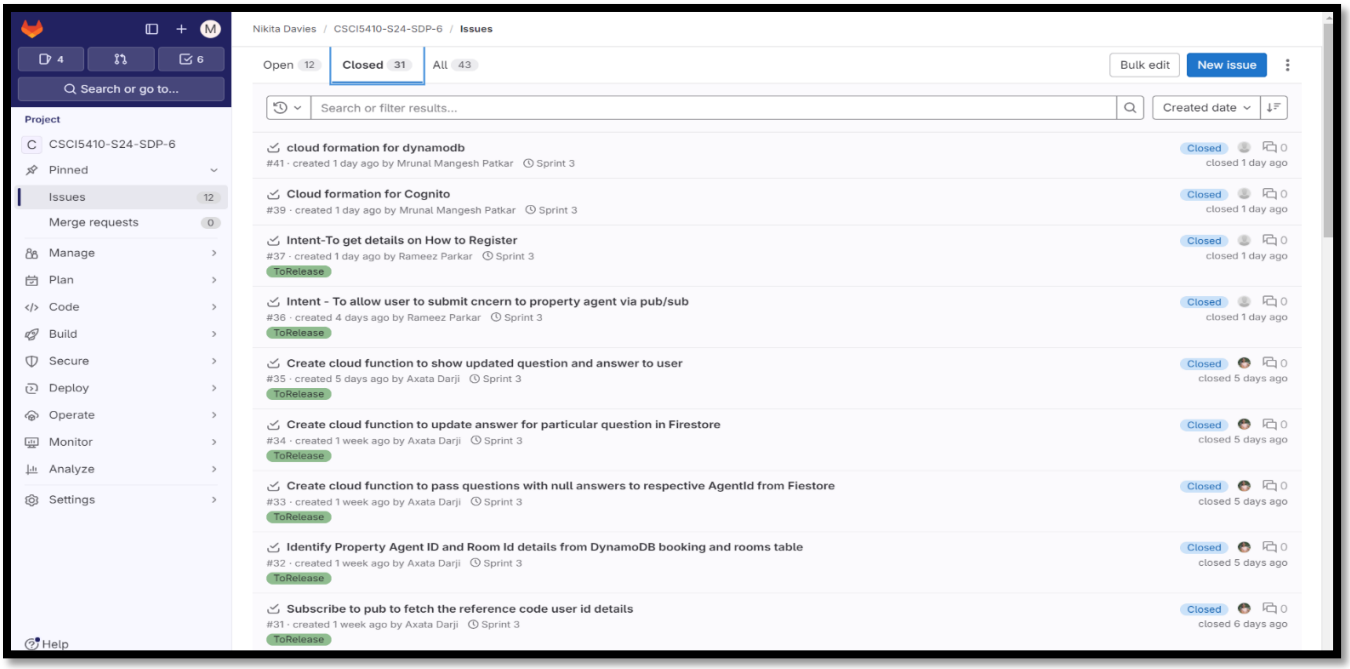

Figure 85: Closed Issues

3. Development board

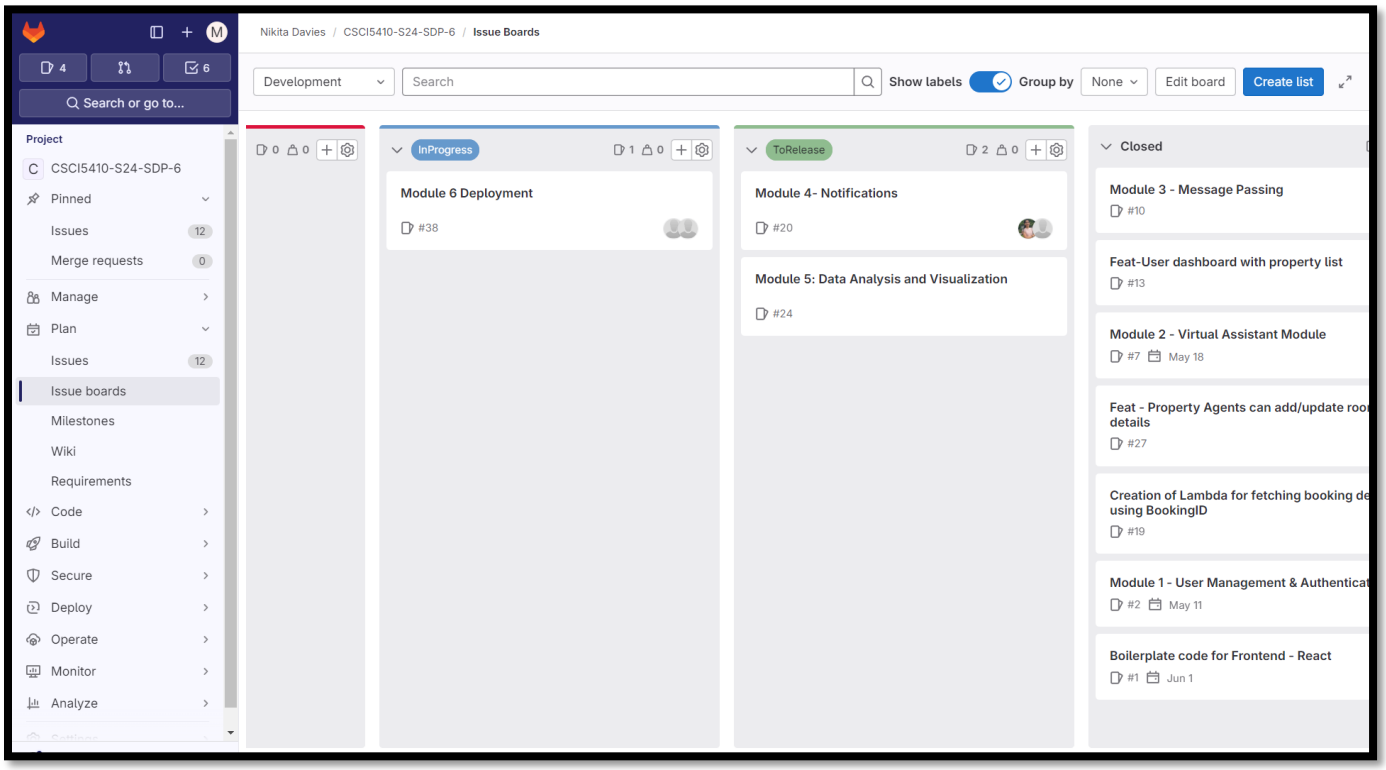

Figure 86: Development board

4. Team Board:

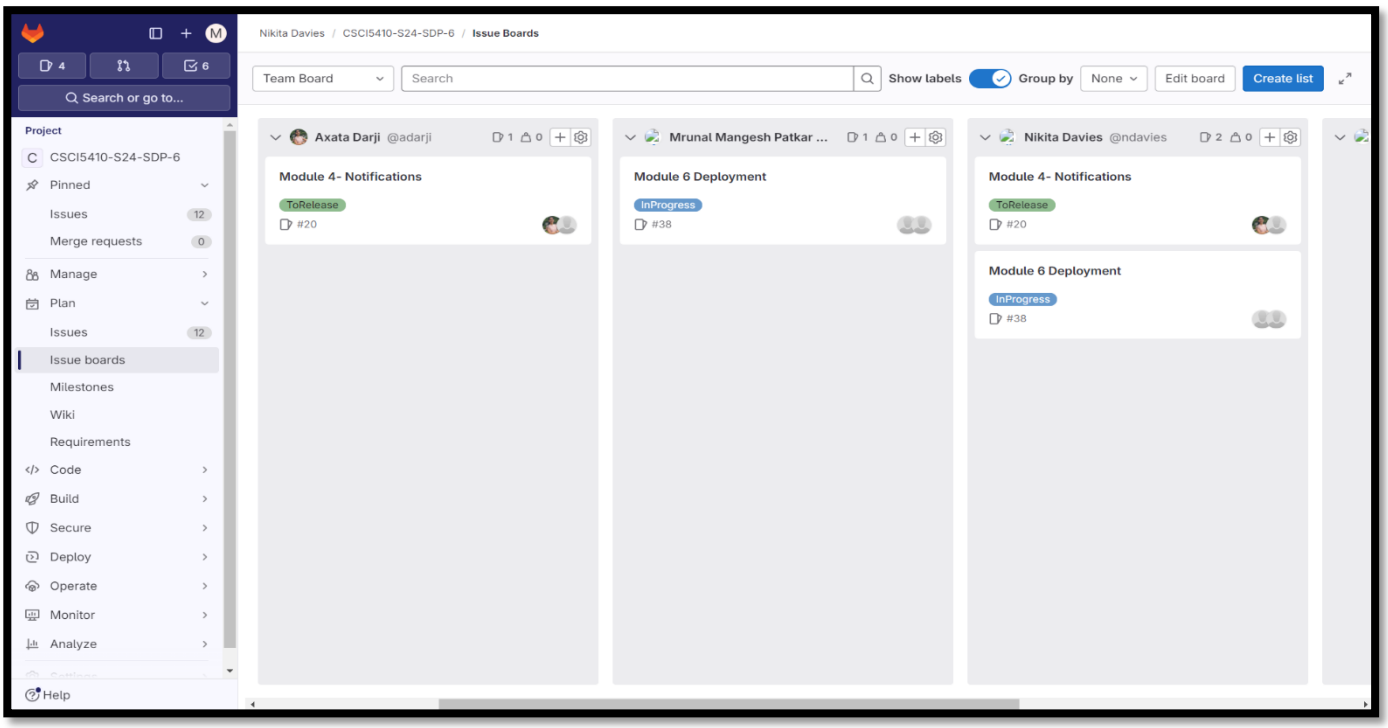

Figure 87: Individual boards at the end

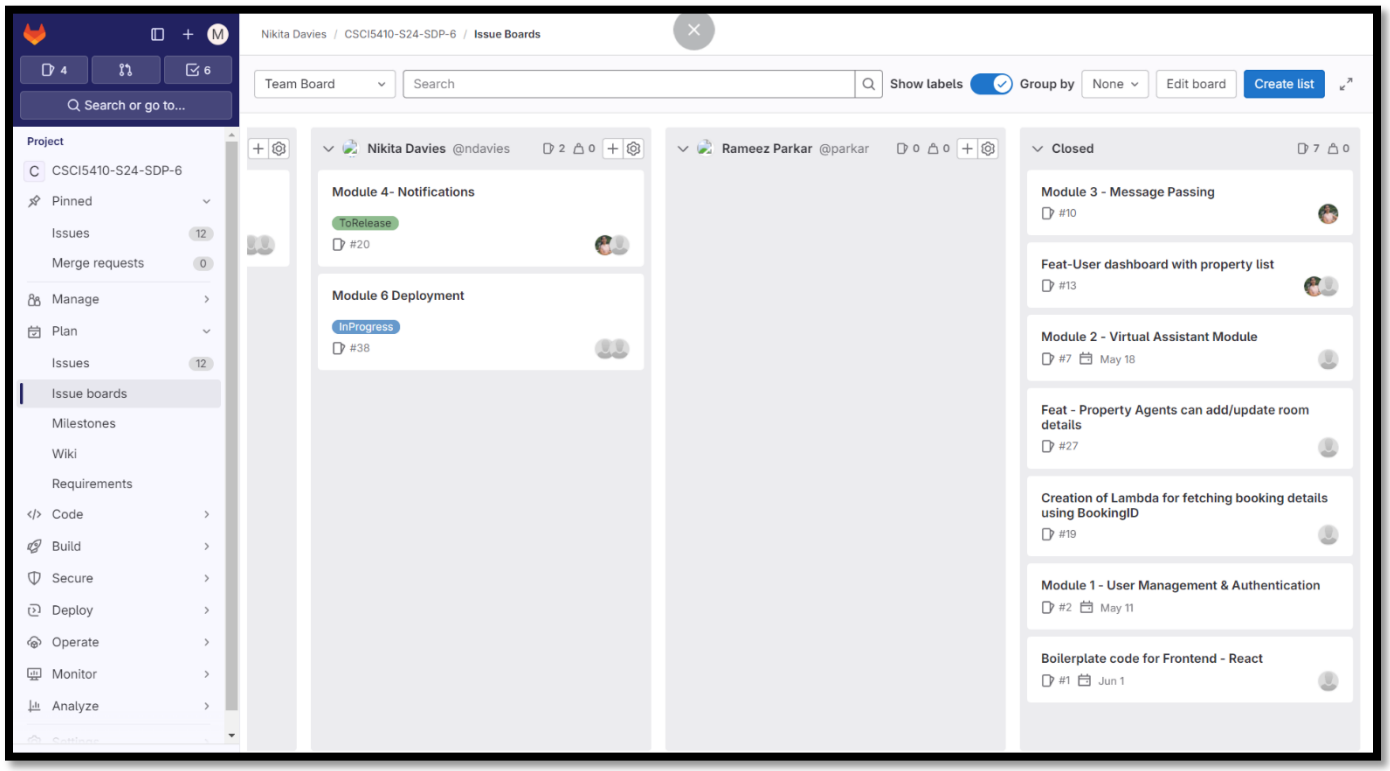

Figure 88: Individual boards with the closed issues list

5. Sprint board:

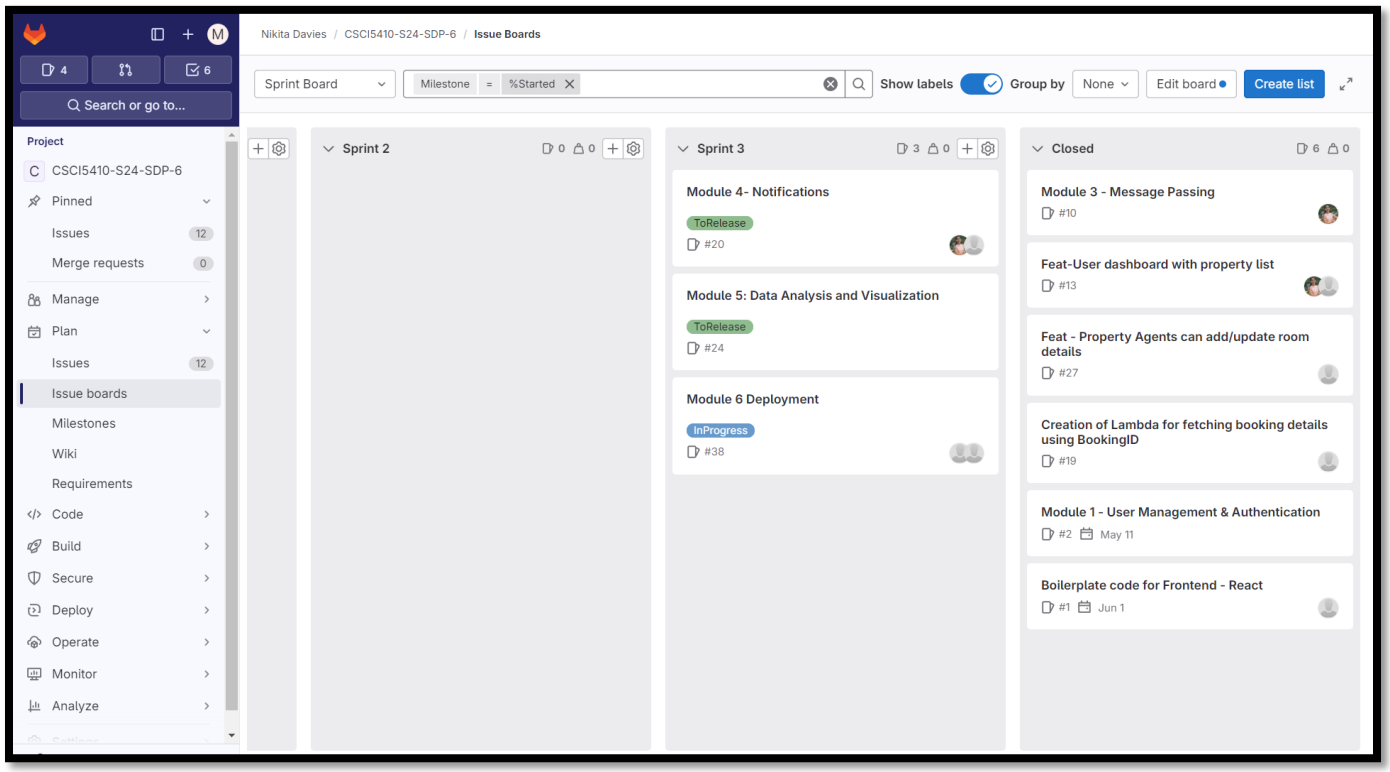

Figure 89: Sprint board at the end

Screenshot of team chats

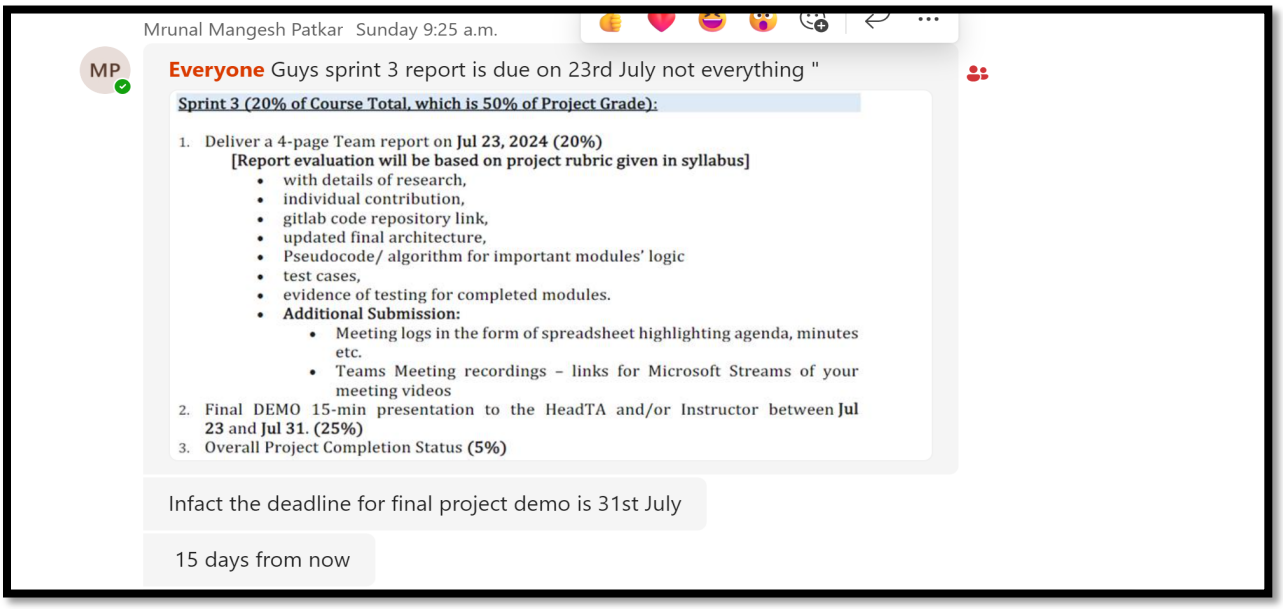

Figure 90: Screenshot of teams group chat

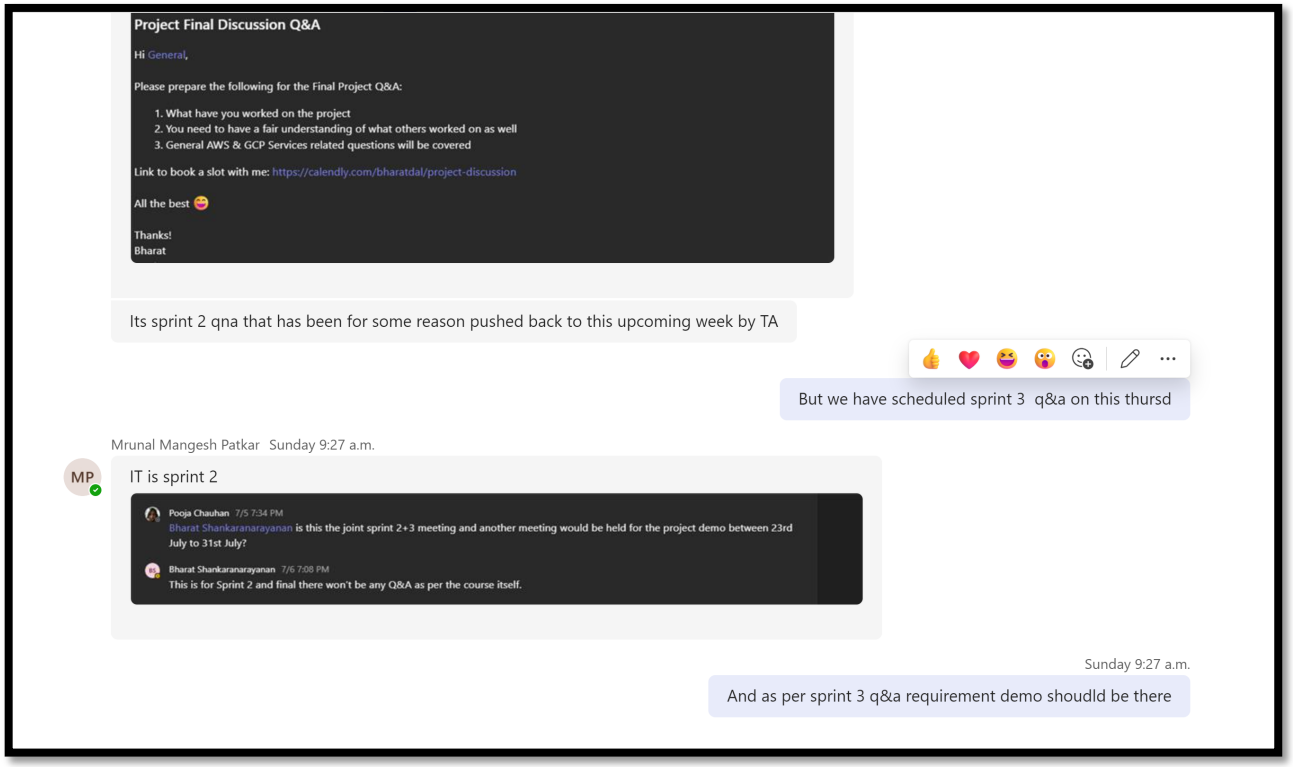

Figure 91: Screenshot of teams group chat

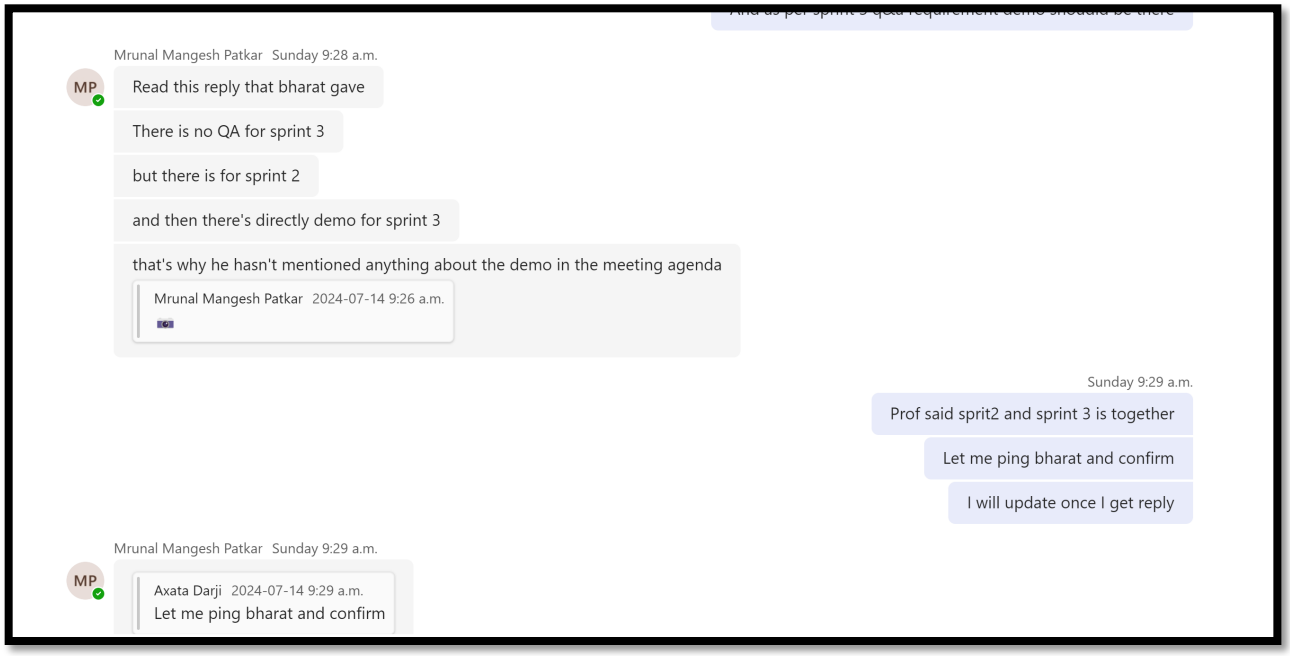

Figure 92: Screenshot of teams group chat

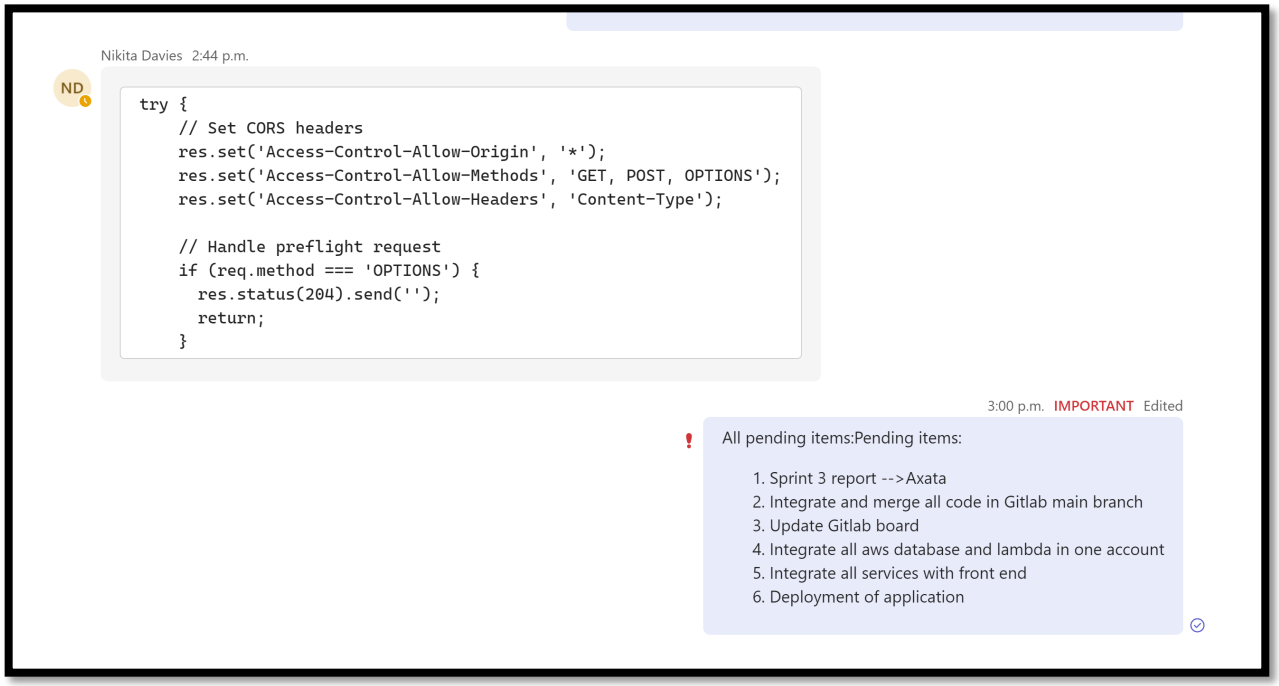

Figure 93: Screenshot of teams group chat

References

1. "Publish messages to topics", Google cloud. Accessed: July 13, 2024. [Online]. Available: https://cloud.google.com/pubsub/docs/publisher

2. "Pub/Sub triggers(2nd gen)", Google cloud. Accessed: July 16, 2024. [Online]. Available: https://cloud.google.com/functions/docs/calling/pubsub

3. “Looker Studio: Community Connectors”, Google cloud. Accessed: July 12, 2024. [Online]. Available: https://developers.google.com/looker-studio/connector .

4. "About connectors, data sources and credentials," Looker Studio. Accessed: July 13, 2024 [Online]. Available: https://support.google.com/lookerstudio/answer/12269110?hl=en

5. "Dialogflow CX basics," Google Cloud. Accessed: July 16, 2024 [Online]. Available: https://cloud.google.com/dialogflow/cx/docs/basics

6. Google Cloud, "Cloud Run Documentation," Google Cloud, 2023. [Online]. Available: https://cloud.google.com/run/docs [Accessed: 18-Jul-2024].

7. Google Cloud, "Managing services in Cloud Run," Google Cloud, 2023. [Online]. Available: https://cloud.google.com/run/docs/deploying [Accessed: 20-Jul-2024].

8. Amazon Web Services, "What is AWS CloudFormation," AWS, 2023. [Online]. Available: https://aws.amazon.com/cloudformation/. [Accessed: 18-Jul-2024].

9. Amazon Web Services, "Using AWS CloudFormation Console," AWS, 2023. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-console.html. [Accessed: 19-Jul-2024].