

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-eawtiwfqq-uc.a.run.app/>

Table of Contents

Details of Research	3
System Architecture.....	6
Pseudocode/Algorithm for important modules.....	6
Testcases and Evidence of testing for completed modules.....	15
Individual Contribution	57
Project Meeting Logs	58
Project Management Tool.....	61
Screenshot of team chats	64
References.....	66

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]

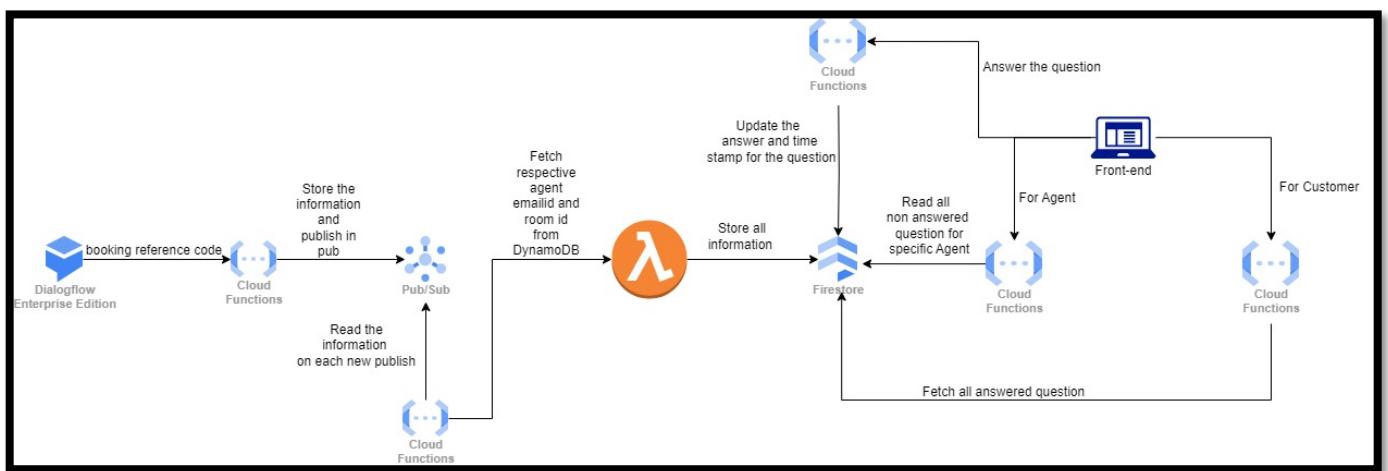


Figure 1: Architecture for Message passing module

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]

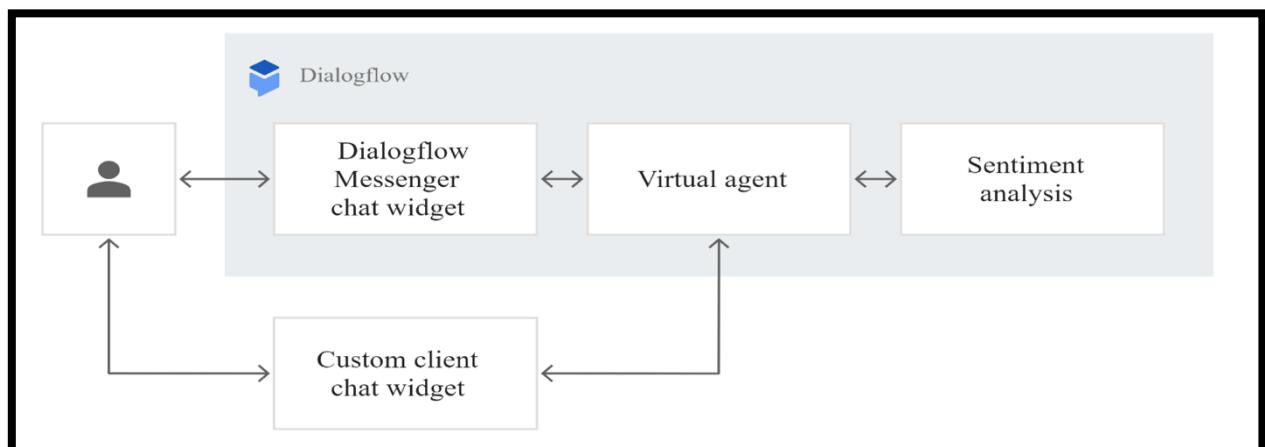


Figure 2: Dialogflow flowchart

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

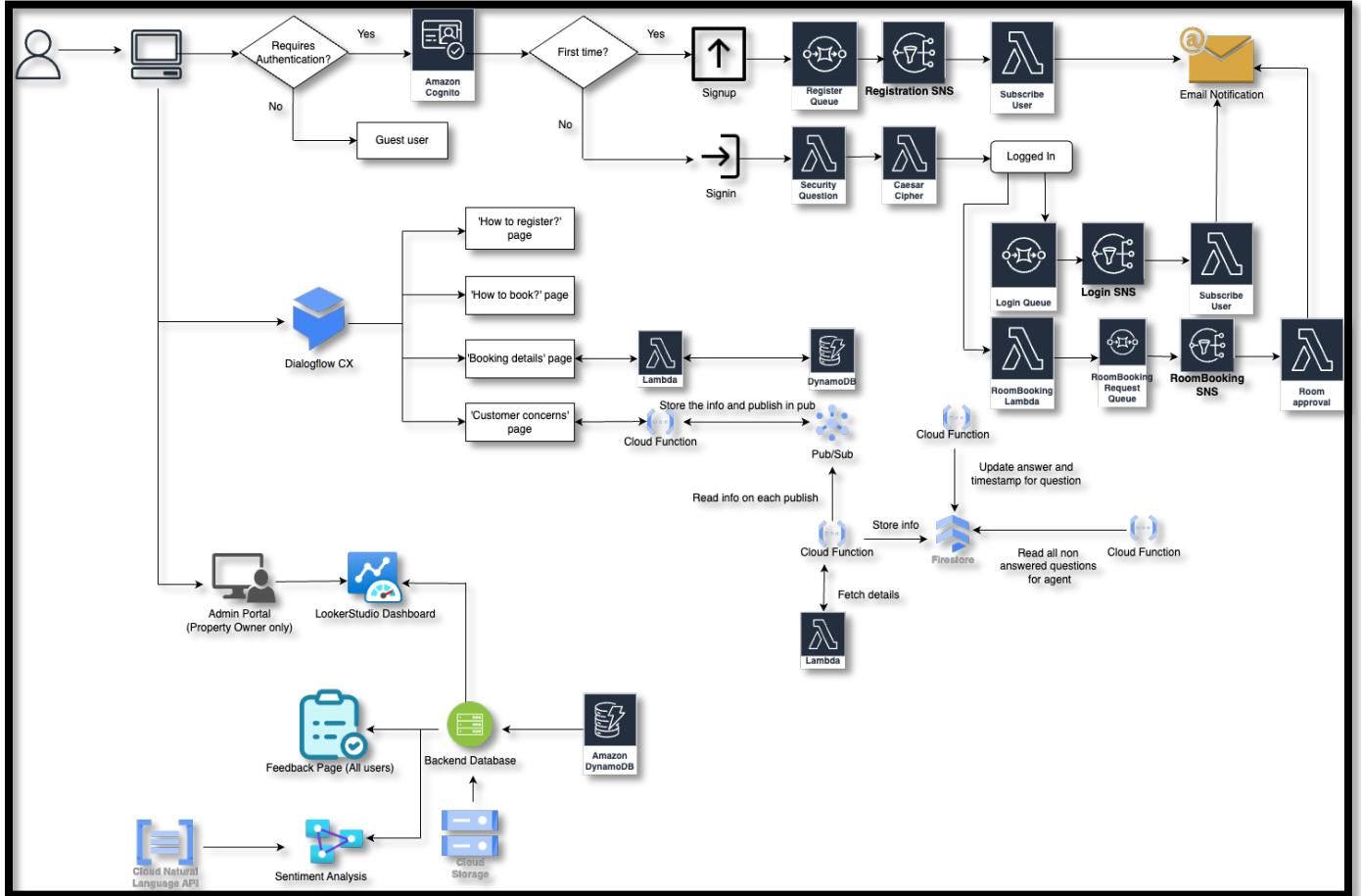


Figure 3: Final Architecture

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": "Username"},
        {"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 invocation 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": "Wondful 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-eawtiwfqq-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 an 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 promptly responds to it	Pass
24.	Guest Users, Registered Customers and Property Agents can ask chatbot how to book a room and chatbot promptly responds to it.	Pass
25.	Guest Users, Registered Customers and Property Agents can ask chatbot for booking details and chatbot promptly responds to it.	Pass
26.	Users can ask chatbot for concerns and chatbot promptly 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

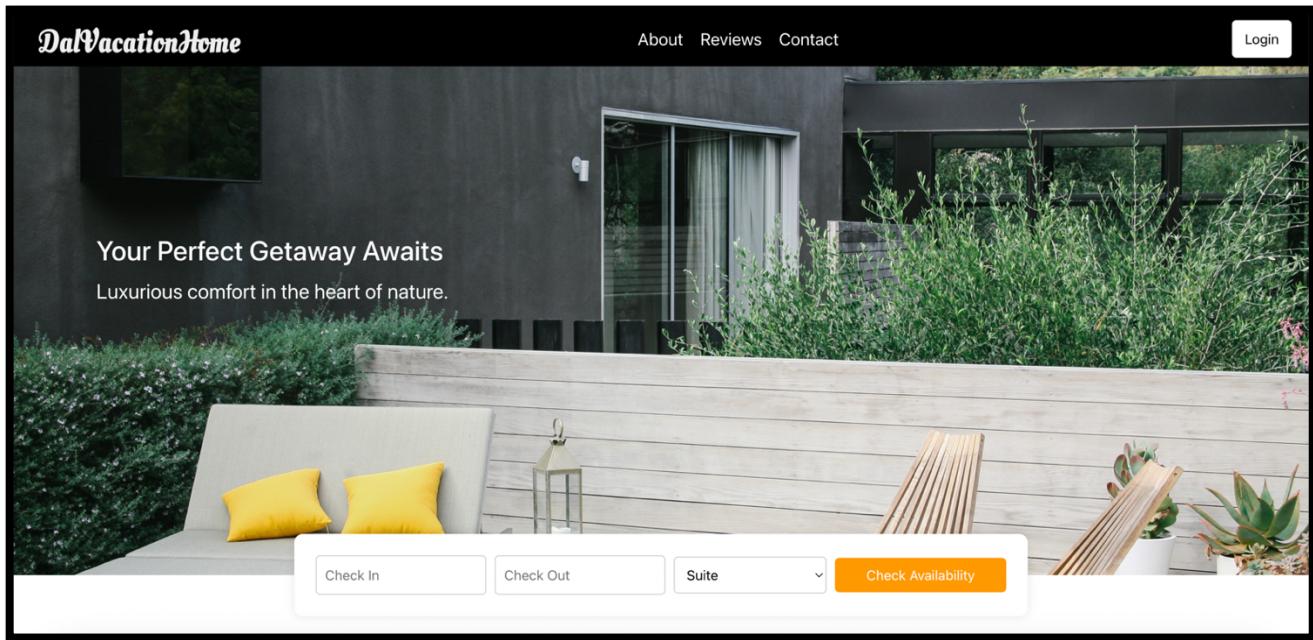


Figure 4: Dashboard for DALVacationHome Application

Rooms List and API integration

Testcase 1: Successful fetching of list of available rooms

The screenshot shows the 'Rooms List' page of the DALVacationHome application. At the top, there is a navigation bar with links for 'About', 'Reviews', 'Contact', and a 'Login' button. On the left, there is a sidebar titled 'Filter by:' with sections for 'Your budget (per night)' (radio buttons for 'CAD 150' and 'CAD 300+' with 'CAD 150' selected), 'Amenities' (checkboxes for 'Parking', 'Air conditioning', 'Parking', and 'Fridge'), and a 'See availability' button. The main area displays a grid of three room listings. Each listing includes a thumbnail image of the room, the room number, a list of features, the overall rating, the number of reviews, and a 'See availability' button. The first listing is 'Suite #room123' with features 'Fridge', overall rating 7, 1 review, and a price of '100 CAD/night'. The second listing is 'Suite #room1523' with features 'Bed,Fridge,AC', overall rating 9, 0 reviews, and a price of '500 CAD/night'. The third listing is 'Suite #room23444' with features 'AC', overall rating 9, 0 reviews, and a price of '500 CAD/night'. The bottom right corner of the page has a small note: 'Queen Room with Two Queen Beds 2 large double beds 500 CAD/night'.

Figure 5: Available Rooms List UI

Executing function: succeeded ([Logs](#))

▼ Details

The area below shows the last 4 KB of the execution log.

```
{
  "statusCode": 200,
  "body": "[{"roomId": "room2208", "feedback": [{"feedbackId": "feedback2208", "comments": "This is for 2288 room", "rating": "3.0"}]}, {"roomId": "room1523", "additionalFeature": "AC", "capacity": "4", "discountCode": "DIS200", "feedbackId": "feedback123", "polarityOfFeedback": "5", "price": "200", "roomType": "room", "feedback": []}, {"roomId": "room1523", "additionalFeature": "Fridge, AC", "capacity": "4", "discountCode": "SBC123", "feedbackId": "feedback1234", "furnishType": "Sem", "polarityOfFeedback": "9", "price": "500", "propertyAgentId": "room123", "additionalFeature": "Fridge", "capacity": "2", "discountCode": "ABC10015", "feedbackId": "feedback123", "polarityOfFeedback": "7", "price": "100", "propertyAgentId": "ax583820@dal.ca", "roomType": "room", "feedback": [{"feedbackId": "feedback123", "comments": "this is test feedback", "rating": "4.5"}]}]"
}
```

Summary

Code SHA-256
KhlEDrii2PHZt/BTVk/yWUYQ/vF3+na6klrCQVp96eM=

Request ID
702c786a-597c-464b-907e-68185265b251

Duration
552.39 ms

Resources configured
128 MB

Execution time
1 second ago (July 23, 2024 at 04:14 PM ADT)

Function version
\$LATEST

Billed duration
553 ms

Max memory used
88 MB

Log output

The section below shows the logging calls in your code. [Click here](#) to view the corresponding CloudWatch log group.

```
START RequestId: 702c786a-597c-464b-907e-68185265b251 Version: $LATEST
2024-07-23T19:14:58.038Z          702c786a-597c-464b-907e-68185265b251 INFO  Input is: {}
2024-07-23T19:14:58.192Z          702c786a-597c-464b-907e-68185265b251 INFO  Before fetching feedback details
2024-07-23T19:14:58.312Z          702c786a-597c-464b-907e-68185265b251 INFO  {
  '$metadata': {
    httpStatusCodes: 200,
    requestId: '850075L755R4NE05MLMSCIIMP7VV4KQN50AEM/JF66Q9ASUAJG',
    extendedRequestId: undefined,
    cfId: undefined,
    attempts: 1,
  }
}
```

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

Figure 7: API integration of fetch properties

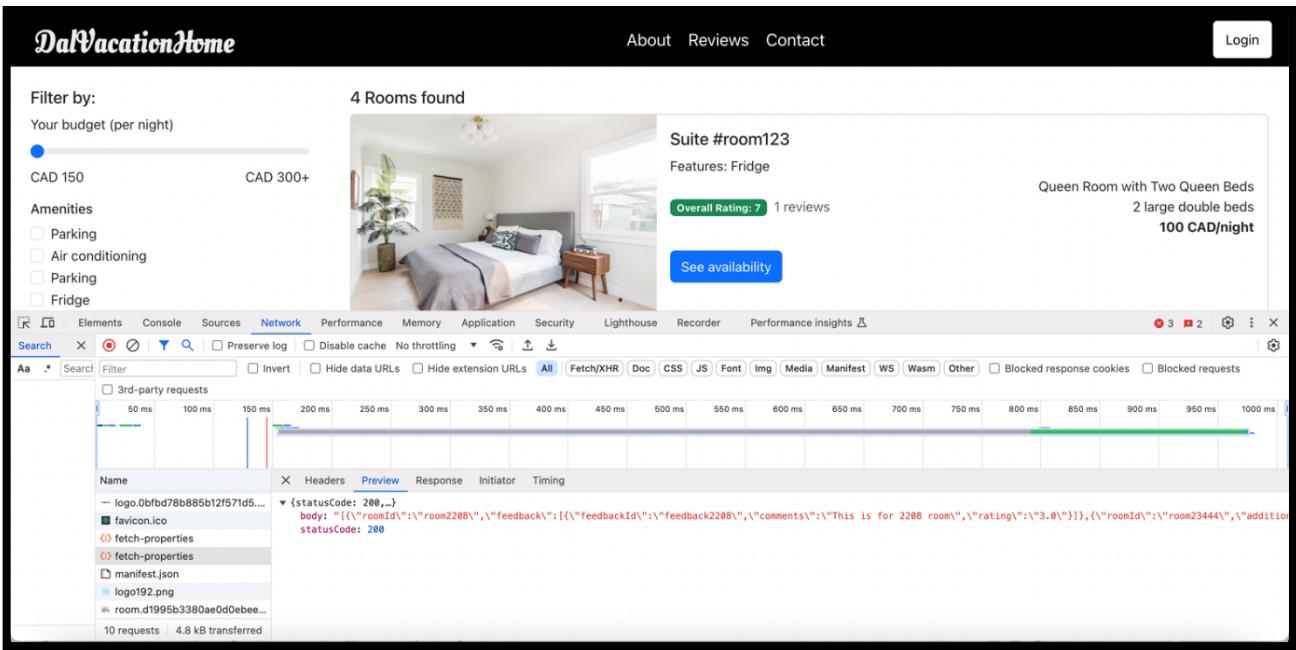


Figure 8: Response of `fetchProperties` API

Testcase 2: Successful fetching of room details

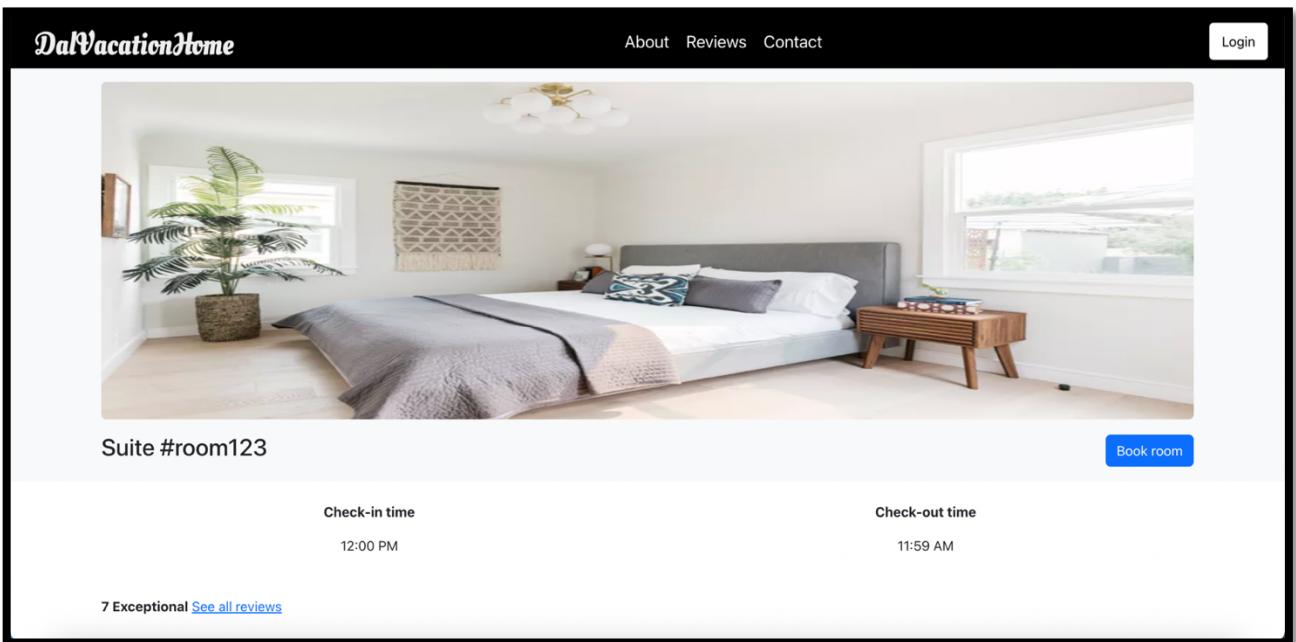


Figure 9: Room Details UI

The screenshot shows the AWS Lambda execution details page for a function named "fetch-property-details". The status is "Executing function: succeeded (logs)". The log output shows a JSON response with a statusCode of 200 and a body containing room2008 feedback items. The summary section includes metrics like execution time (2 seconds ago), function version (\$LATEST), duration (786.23 ms), and resources configured (128 MB). The log output section shows CloudWatch logs for the request and response.

Figure 10: Lambda execution for fetching single property detail

The screenshot shows the Network tab in the developer tools for a web application. It displays a timeline of network requests. One notable request is for "fetch-property-details" with a duration of 150 ms. The details panel for this request shows the following information:

Name	Request URL	Response URL
fetch-property-details	https://f3js78prj2.execute-api.us-east-1.amazonaws.com/dev/fetch-property-details	https://f3js78prj2.execute-api.us-east-1.amazonaws.com/dev/fetch-property-details
Request Method	POST	
Status Code	200 OK	
Remote Address	3.216.2.144:443	
Referrer Policy	strict-origin-when-cross-origin	

Figure 11: API integration of fetch properties details API

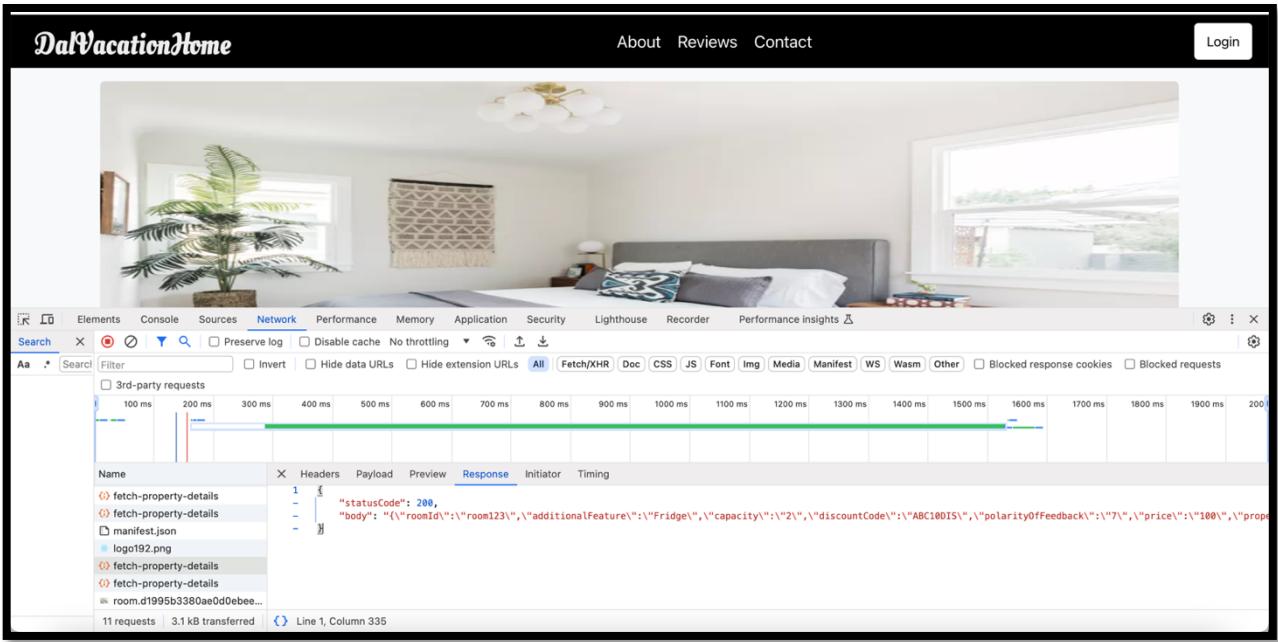


Figure 12: Response of fetch property details API

Create/Edit Rooms by Property Agent

Testcase 1: Create Room by Property Agent

The screenshot shows the 'Manage Rooms' section of the DalVacationHome application. At the top right are navigation links for 'Rooms', 'Customer Concerns', 'Analytics', and a 'Sign Out' button. A large orange 'Create Room' button is centered above three room cards. The first card is for a 'Room' with capacity 1, price 230\$, and queen-sized bed features. The second card is for a 'Room' with capacity 3, price 234\$, and queen-sized bed features. The third card is for a 'Deluxe' room with capacity 2, price 230\$, queen-sized bed features, AC, and discount code zxcv.

Figure 13: Rooms Added by a Property Agent

The screenshot shows the 'Create Room' modal window. It includes fields for 'Room Type' (a dropdown menu), 'Capacity' (text input), 'Price' (text input), 'Furnished Type' (dropdown menu), 'Additional Features' (text input), and 'Discount Code' (text input). At the bottom are 'Save' and 'Cancel' buttons.

Figure 14: Create Room UI

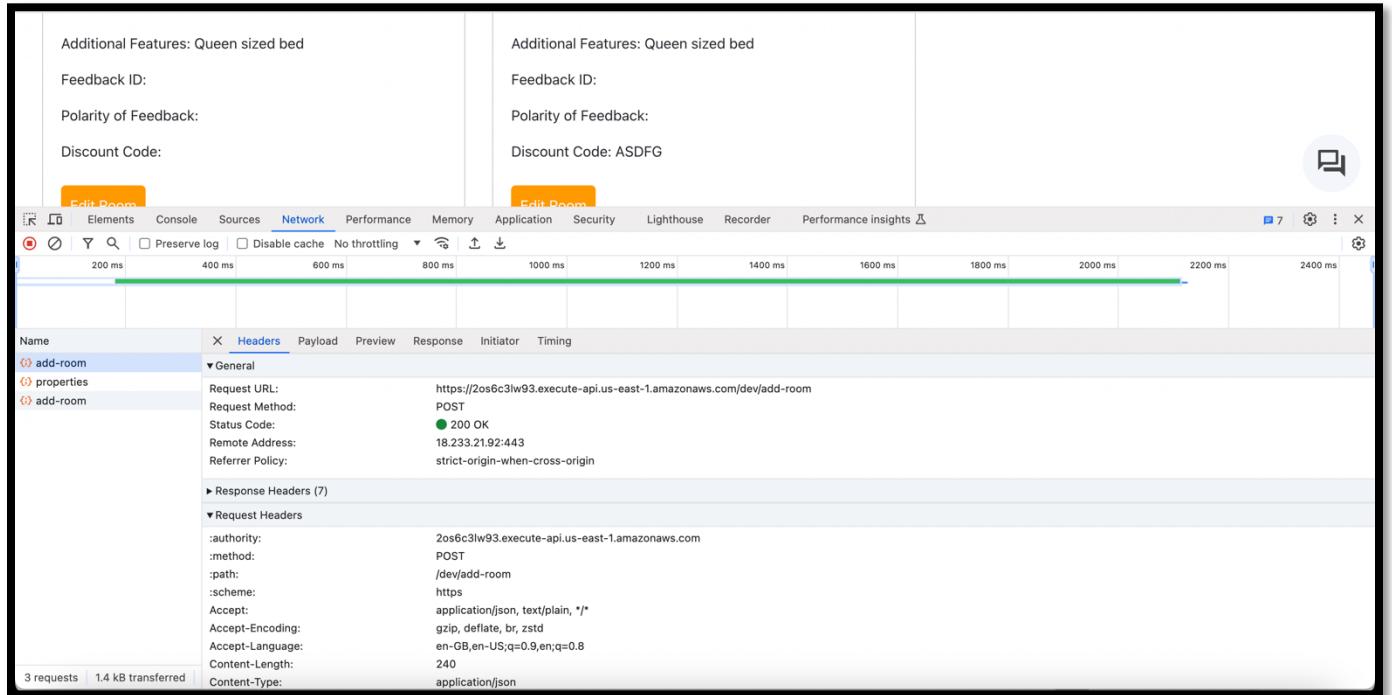


Figure 15: API integration of add room

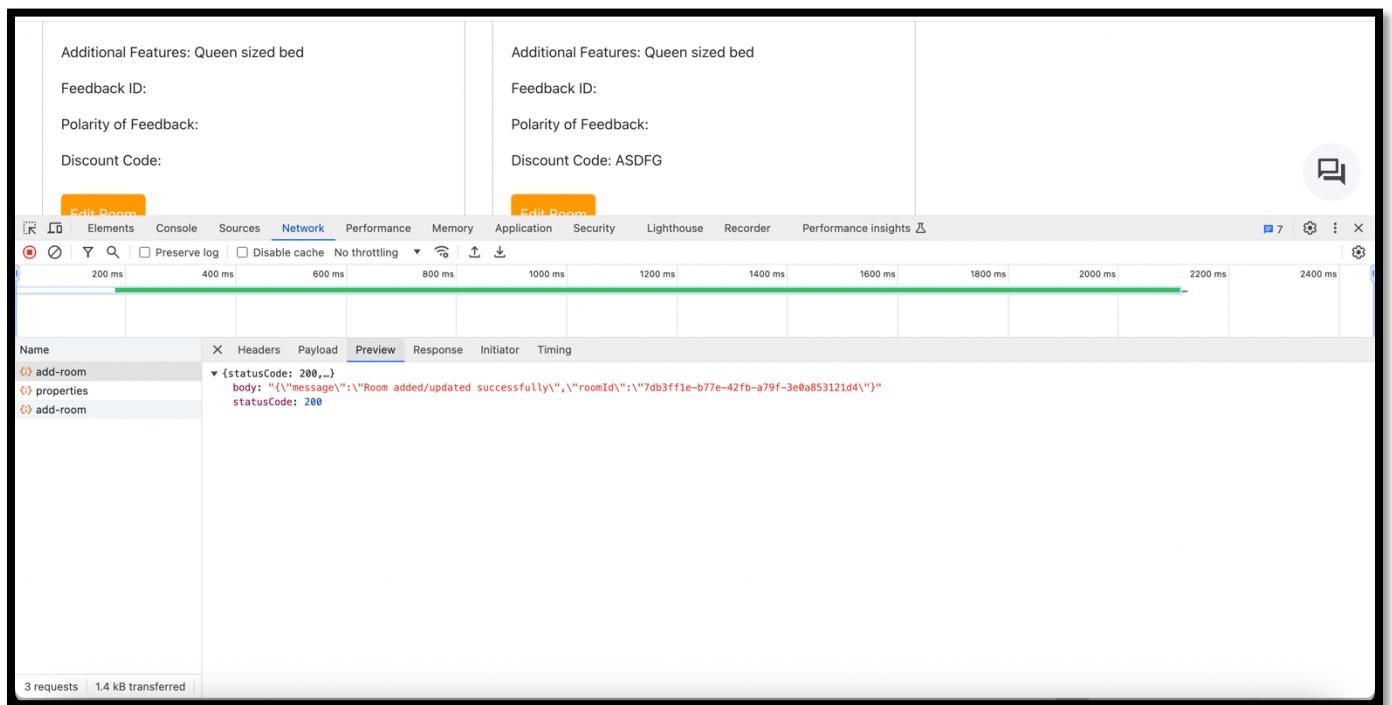


Figure 16: Response of addRoom API

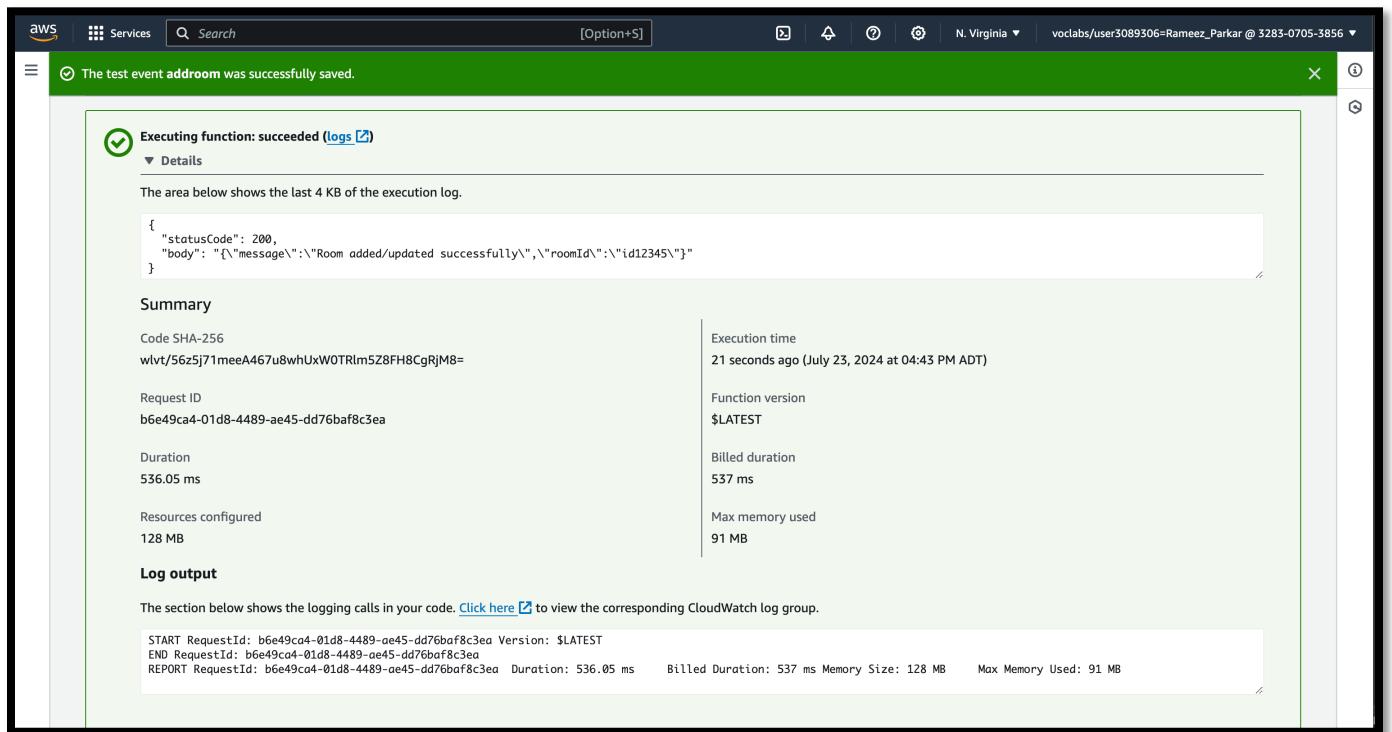


Figure 17: Lambda for add room

Testcase 2: Edit Room by Property Agent

DalVacationHome

Rooms Customer Concerns Analytics

Sign Out

Manage Rooms

Create Room

Room Type:

Capacity:

Price:

Furnished Type:

Additional Features:

Discount Code:

Save **Cancel**

Figure 18: Edit Room UI For PropertyAgent

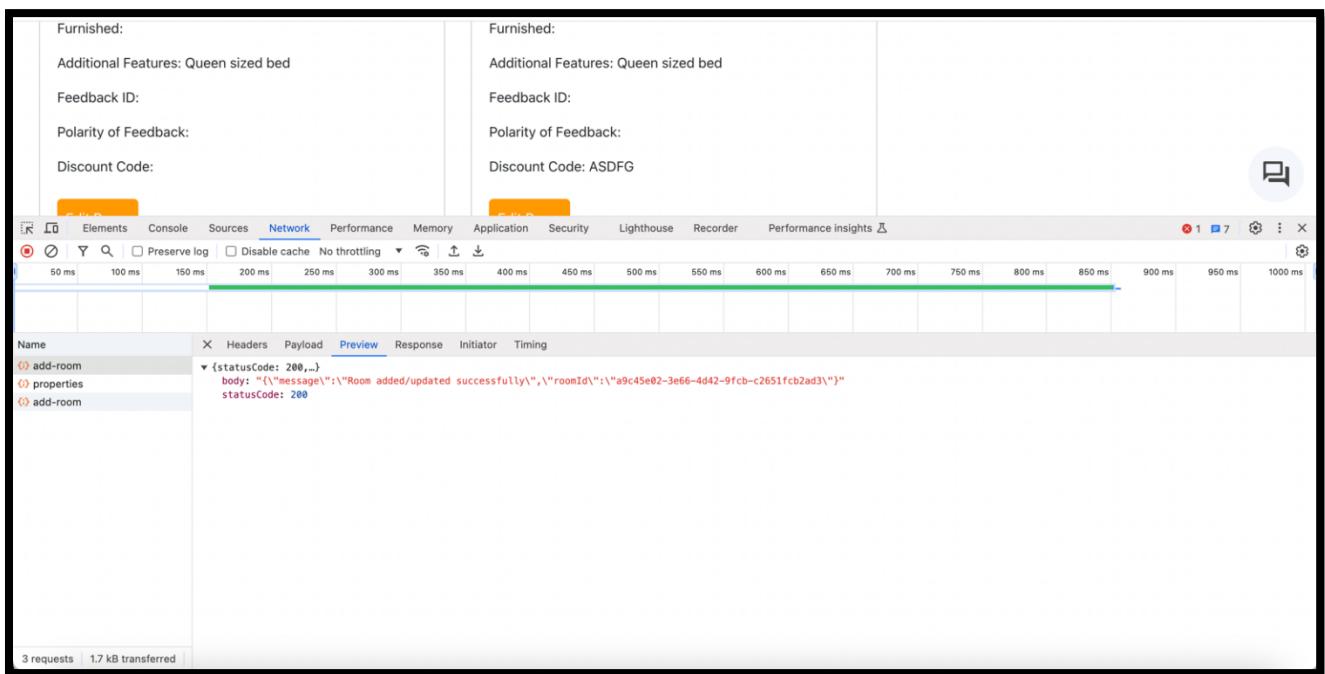


Figure 19: API Response after editing the room

Summary	Execution time
Code SHA-256 wlvt/56z5j71meeA467u8whUxWOTRlm5Z8FH8CgRjM8=	21 seconds ago (July 23, 2024 at 04:43 PM ADT)
Request ID b6e49ca4-01d8-4489-ae45-dd76baf8c3ea	Function version \$LATEST
Duration 536.05 ms	Billed duration 537 ms
Resources configured 128 MB	Max memory used 91 MB

Figure 20: Lambda for update room

Module: Data Analysis

Feedbacks and API integration

Testcase 1: Successful fetching of feedbacks

The screenshot shows a web application interface for 'DafVacationHome'. At the top, there is a navigation bar with links for 'About', 'Reviews', and 'Contact', along with a 'Login' button. Below the navigation bar, the title 'Guest Reviews for Room: room2208' is displayed, followed by a large blue 'Add Review' button. The main content area contains two review cards. The first card is for User ID: 2, who rated the room 5 and wrote, 'Excellent Room with great facilities. Loved it!!'. The second card is for User ID: 4, who rated the room 2 and wrote, 'Had a good stay'. Both cards show a small blue circular icon with the number 9. In the bottom right corner of the content area, the text 'Overall feedback polarity: 8' is displayed.

Figure 21: Guest Reviews UI

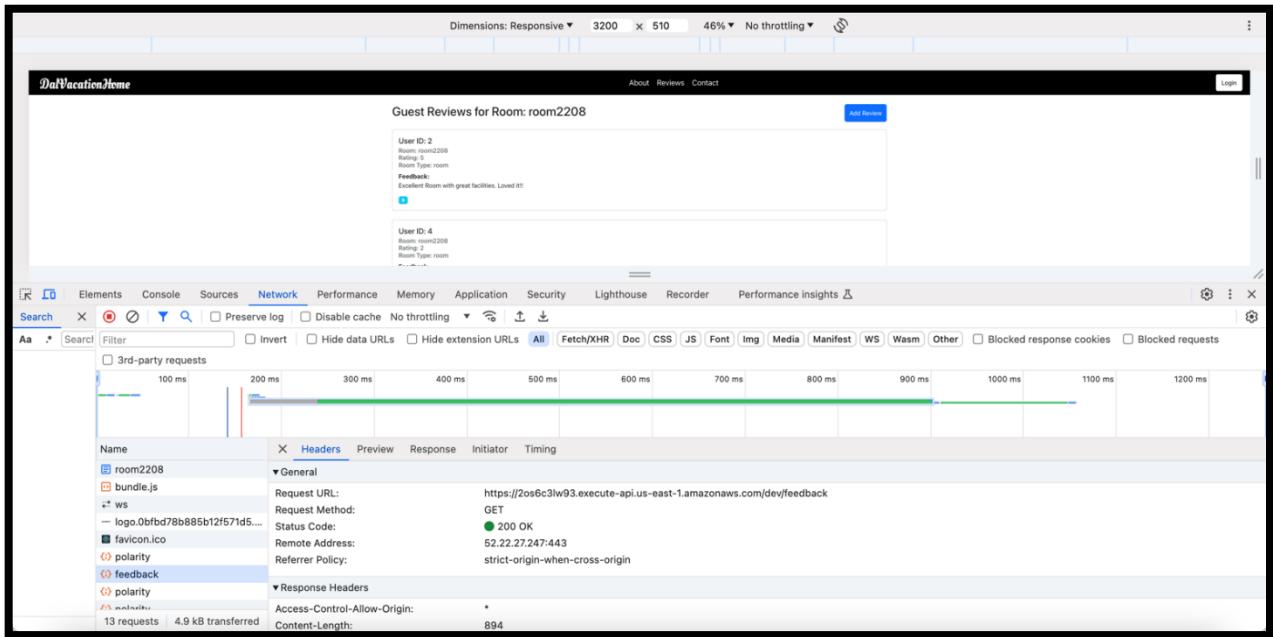


Figure 22: Feedback API Integration

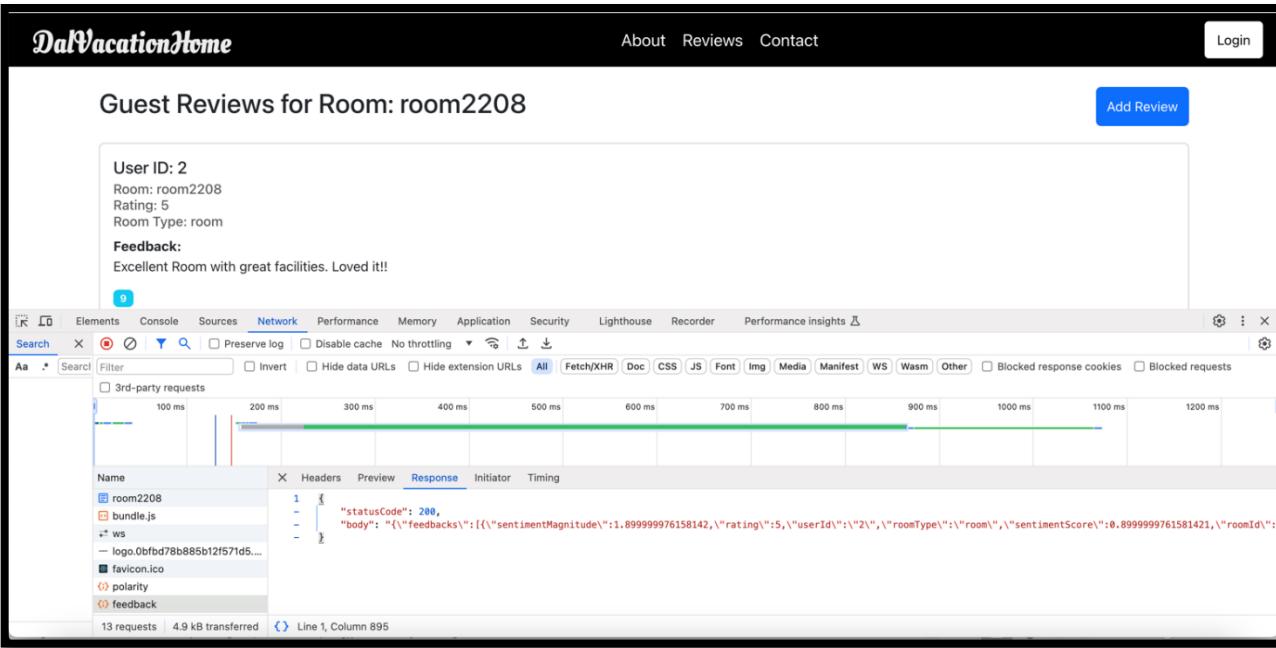


Figure 23: Response of feedback API

Testcase 2: Successful fetching of feedbacks overall polarity

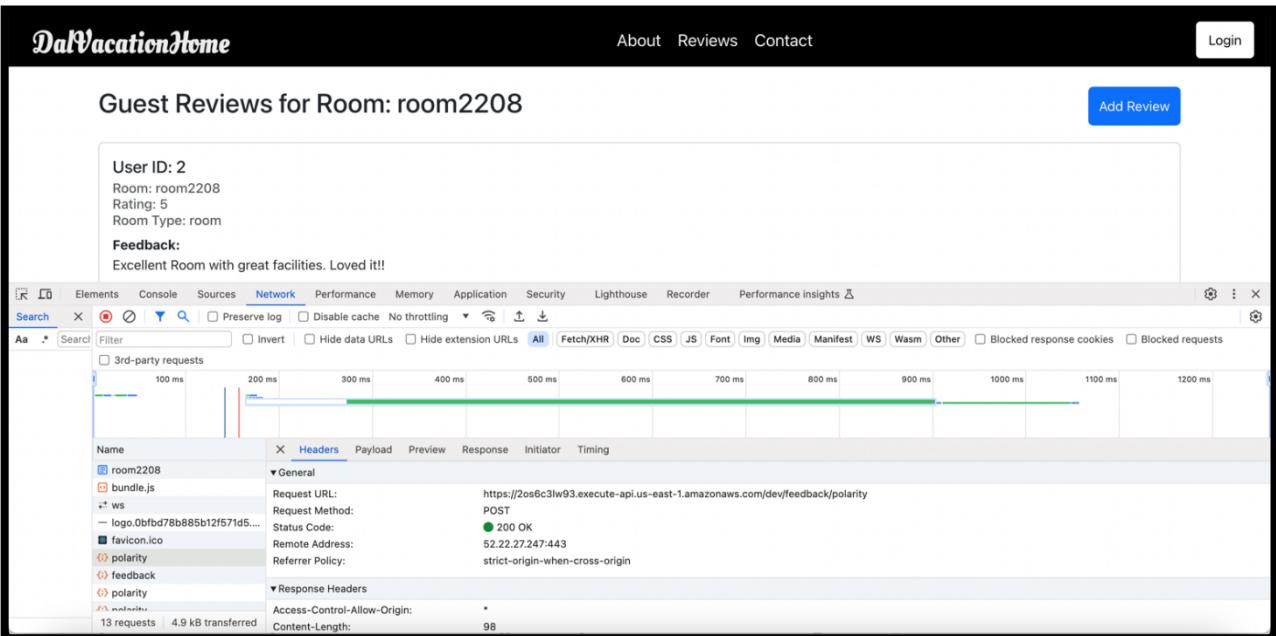


Figure 24: Feedback Polarity API Integration

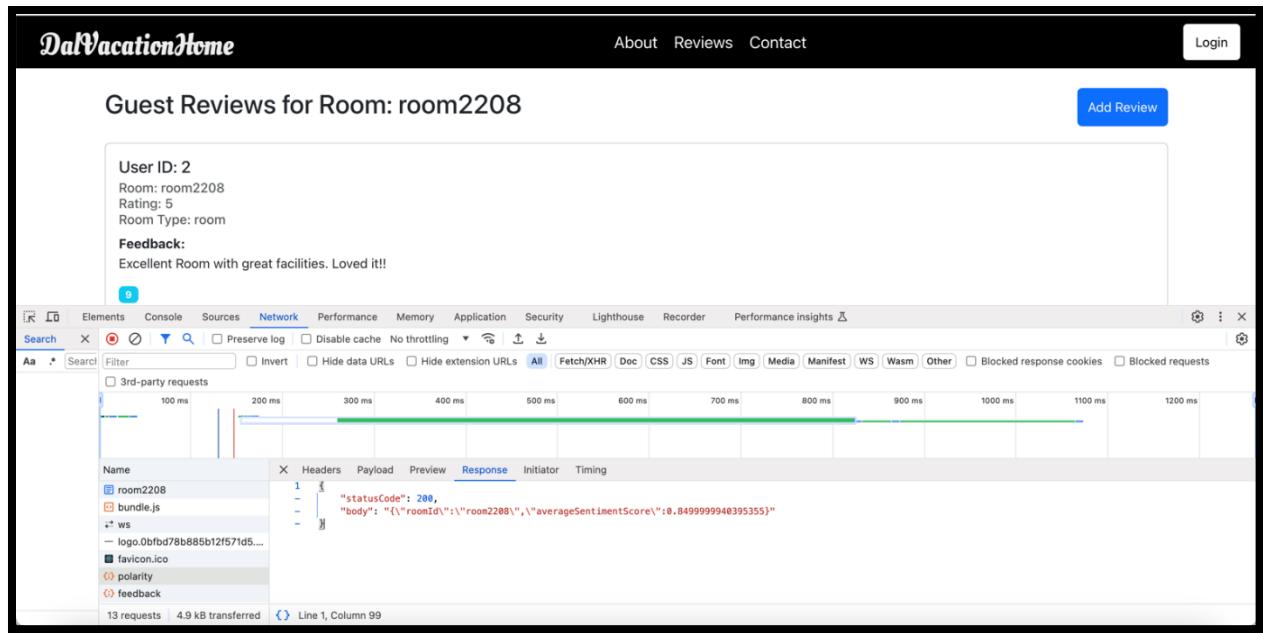


Figure 25: Response of getFeedbackPolarity API

Testcase 3: Registered Customers can add Reviews

User ID: propertyagent01@yopmail.com
Room: d33f2cb9-09f7-4351-986b-0cde0eda73cc
Rating: 4
Room Type: room

Feedback:
Its a great experience

9

How was your stay?

★★★★★

Its a wonderful stay, spacious rooms and great amenities

Submit

Overall feedback polarity: 9

Figure 26: Add Review UI

The screenshot shows a web browser interface for a vacation rental platform. At the top, there's a navigation bar with 'DalVacationHome' logo, 'Rooms', 'Customer Concerns', 'Analytics', and 'Sign Out'. Below the navigation is a section titled 'Guest Reviews for Room: d33f2cb9-09f7-4351-986b-0cde0eda73cc' with a 'Add Review' button. The main content area displays a Network tab in the developer tools, showing a POST request to 'https://20s6c3lw93.execute-api.us-east-1.amazonaws.com/dev/feedback'. The request details show the following parameters:

- Name: feedback
- Request URL: https://20s6c3lw93.execute-api.us-east-1.amazonaws.com/dev/feedback
- Request Method: POST
- Status Code: 200 OK
- Remote Address: 54.167.69.185:443
- Referrer Policy: strict-origin-when-cross-origin
- Response Headers (7):

 - :authority: 20s6c3lw93.execute-api.us-east-1.amazonaws.com
 - :method: POST
 - :path: /dev/feedback
 - :scheme: https
 - Accept: application/json, text/plain, */*
 - Accept-Encoding: gzip, deflate, br, zstd
 - Accept-Language: en-US;q=0.9, en-n=0.8

- Timing: 2 requests

Figure 27: API Integration of add feedback

The screenshot shows the AWS Lambda function execution details for the 'addFeedback' function. The execution was successful, indicated by a green checkmark icon and the message 'Executing function: succeeded (logs)'. The log output shows a single log entry:

```
{
  "statusCode": 200,
  "body": "{\"message\":\"Feedback added and analyzed successfully\"}"
}
```

The summary section provides the following details:

Code SHA-256	Execution time
UOVZ0gR5Rh0H8RMLiu1lQud6ZLqVJPGZYNd4NLcvJY=	21 seconds ago (July 23, 2024 at 03:09 PM ADT)
Request ID	Function version
4ad9e480-ec89-4a5b-a4d1-4e336a7e3511	\$LATEST
Duration	Billed duration
215.84 ms	216 ms
Resources configured	Max memory used
128 MB	113 MB

The Log output section shows the CloudWatch logs for the function invocation:

```

2024-07-23T18:09:26.007Z 4ad9e480-ec89-4a5b-a4d1-4e336a7e3511 INFO Received event: {"roomId": "room3", "roomType": "room", "feedback": "Wondful experience and great room", "rating": 4, "userId": "123"}
2024-07-23T18:09:26.007Z 4ad9e480-ec89-4a5b-a4d1-4e336a7e3511 INFO Analyzing sentiment...
STAR RequestId: 4ad9e480-ec89-4a5b-a4d1-4e336a7e3511 Version: $LATEST
2024-07-23T18:09:26.104Z 4ad9e480-ec89-4a5b-a4d1-4e336a7e3511 INFO Sentiment analysis result: { magnitude: 0.8999999761581421, score: 0.8999999761581421 }
2024-07-23T18:09:26.104Z 4ad9e480-ec89-4a5b-a4d1-4e336a7e3511 INFO Saving feedback to DynamoDB...

```

Figure 28: Lambda invocation test for addFeedback

Testcase 4: Data Analytics using LookerStudio

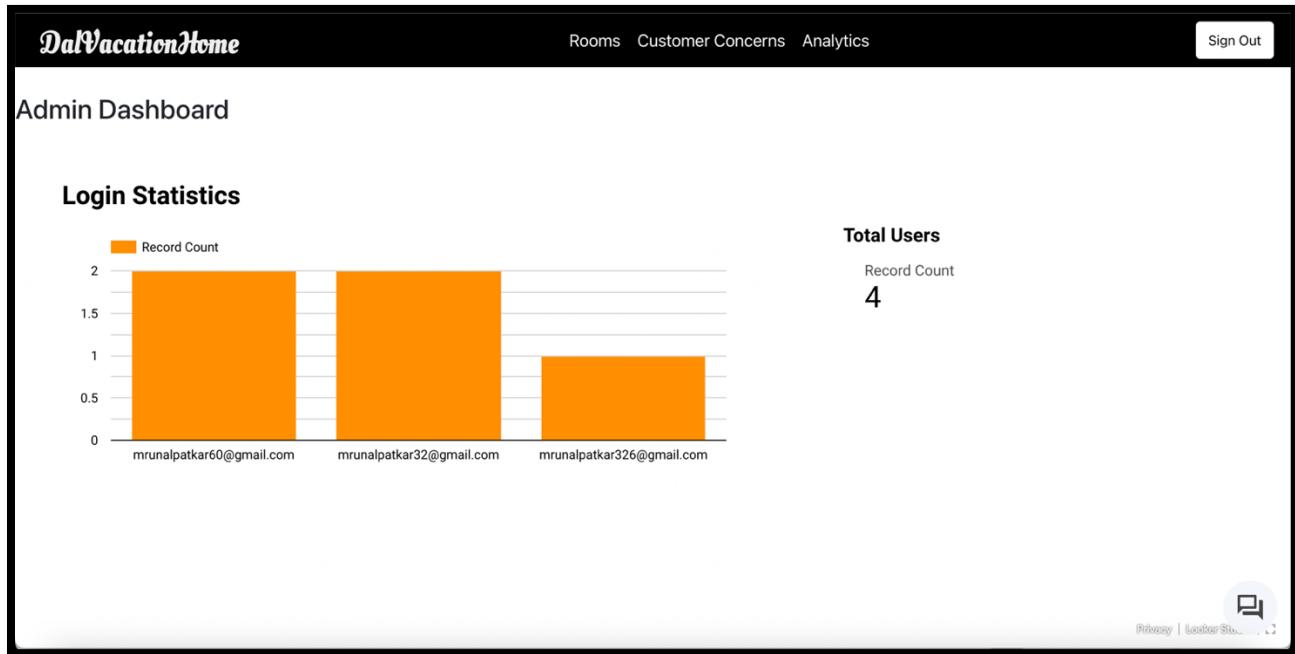


Figure 29: Admin dashboard with embedded Lookerstudio url

Module: Virtual Assistant using DialogFlow CX

Testcase 1: User asks chatbot how to register

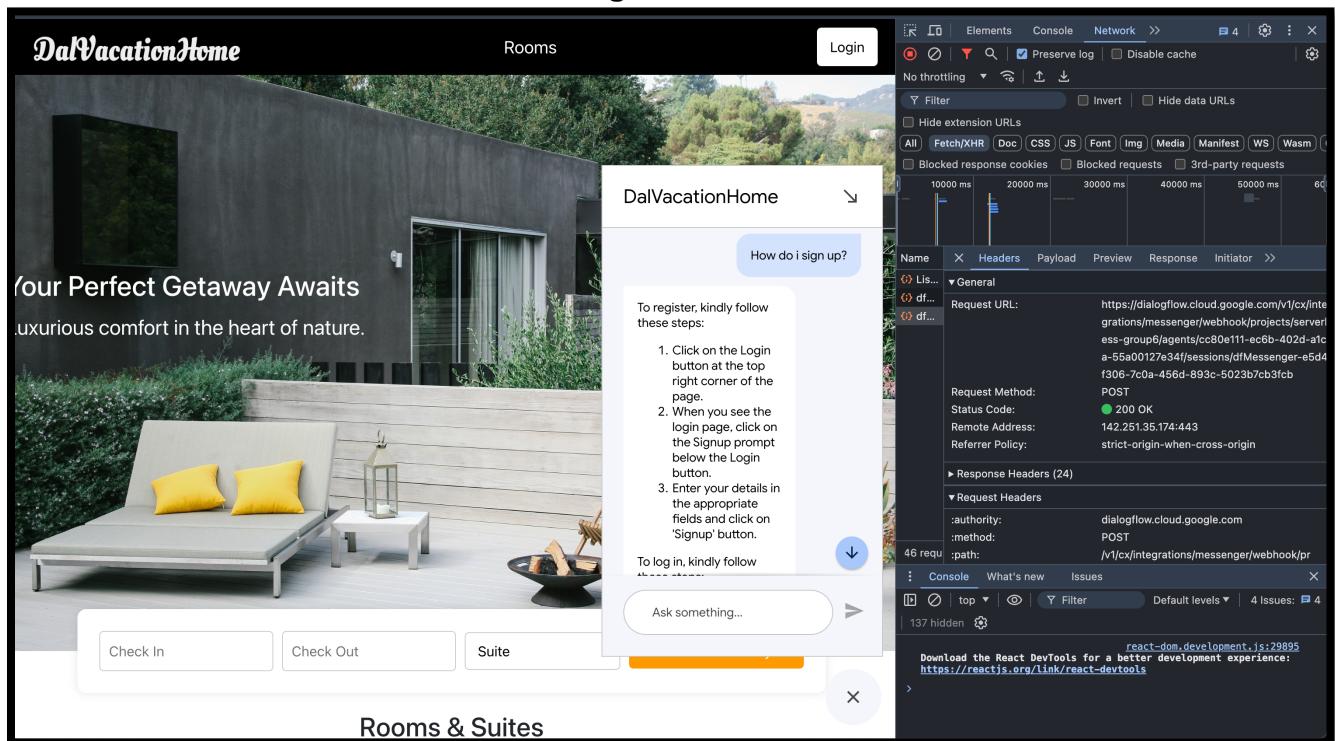


Figure 30: Dialogflow Chatbot for How to Register

Testcase 2: User asks chatbot how to book a room

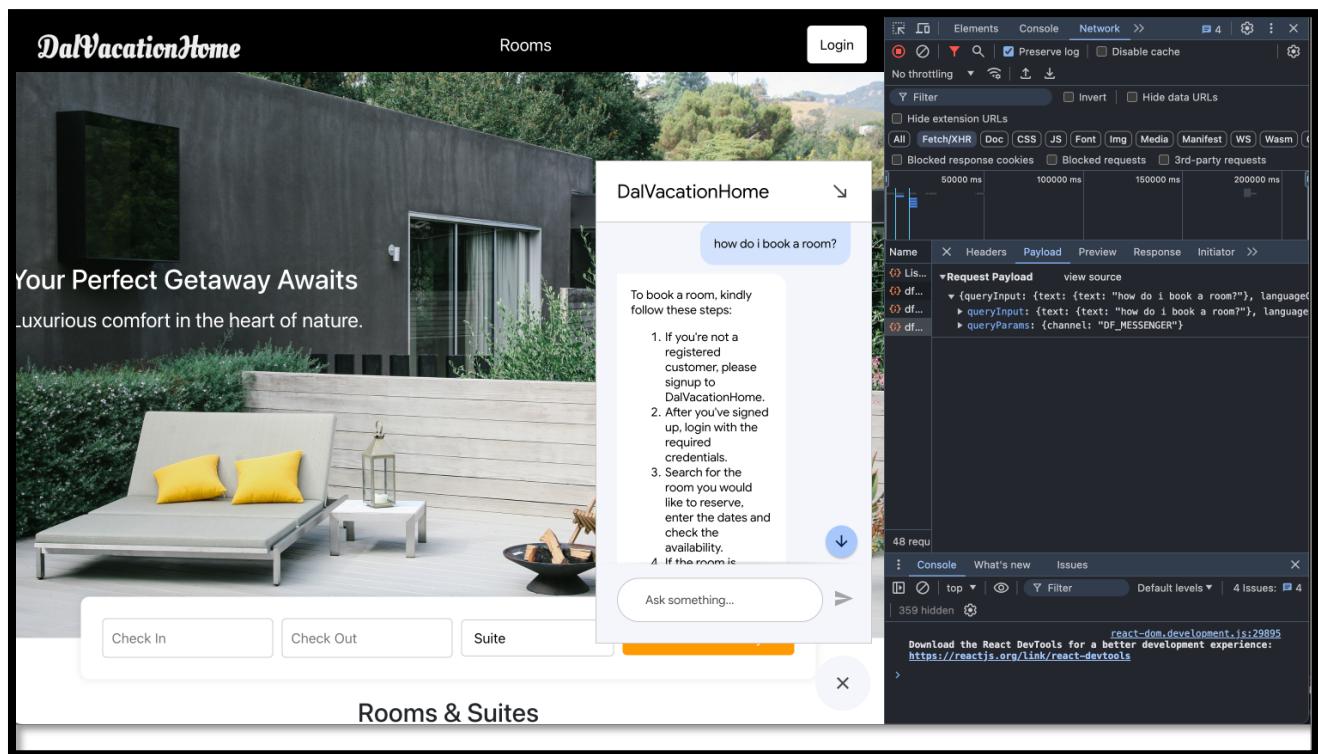


Figure 31: Dialogflow Chatbot for How to Book Room

Testcase 3: User asks chatbot for their booking details

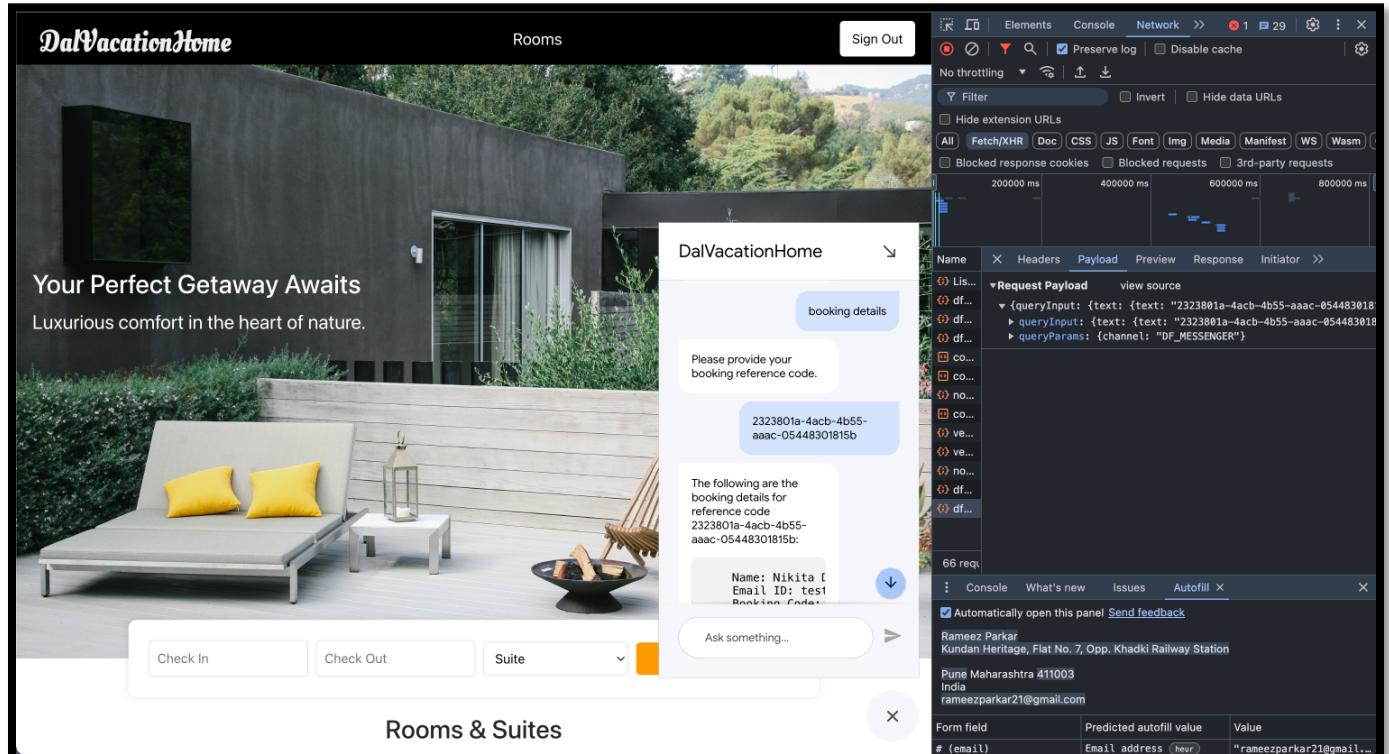


Figure 32: Dialogflow Chatbot for Booking Details

Testcase 4: User mentions a concern to the chatbot

Your Perfect Getaway Awaits
Luxurious comfort in the heart of nature.

DalVacationHome

Please provide your email id

test12@yomail.com

Provide your booking reference code

2323801a-4acb-4b55-aaac-05448301815b

Kindly mention your concern

I am not able to checkin

Ask something...

Rooms & Suites

Request Payload

```
{
  "queryInput": {
    "text": "2323801a-4acb-4b55-aaac-05448301815b"
  },
  "queryParams": {
    "channel": "DF_MESSENGER"
  }
}
```

Name Headers Payload Preview Response Initiator

Request Payload

```
{
  "queryInput": {
    "text": "2323801a-4acb-4b55-aaac-05448301815b"
  },
  "queryParams": {
    "channel": "DF_MESSENGER"
  }
}
```

Console What's new Issues Autofill

Rameez Parkar
Kundan Heritage, Flat No. 7, Opp. Khadki Railway Station
Pune Maharashtra 411003
India
rameezparkar21@gmail.com

Form field Predicted autofill value Value
(email) Email address (hour) "rameezparkar21@gmail.com"

Figure 33: Dialogflow Chatbot for customer concerns

Your Perfect Getaway Awaits
Luxurious comfort in the heart of nature.

DalVacationHome

reference code

2323801a-4acb-4b55-aaac-05448301815b

Kindly mention your concern

I am not able to checkin

Your concern has been reported to respective property agent. Please use the ticketId 1721759195744 for follow up.

Ask something...

Rooms & Suites

Request Payload

```
{
  "queryInput": {
    "text": "2323801a-4acb-4b55-aaac-05448301815b"
  },
  "queryParams": {
    "channel": "DF_MESSENGER"
  }
}
```

Name Headers Payload Preview Response Initiator

Request Payload

```
{
  "queryInput": {
    "text": "2323801a-4acb-4b55-aaac-05448301815b"
  },
  "queryParams": {
    "channel": "DF_MESSENGER"
  }
}
```

Console What's new Issues Autofill

Rameez Parkar
Kundan Heritage, Flat No. 7, Opp. Khadki Railway Station
Pune Maharashtra 411003
India
rameezparkar21@gmail.com

Form field Predicted autofill value Value
(email) Email address (hour) "rameezparkar21@gmail.com"

Figure 34: Dialogflow Chatbot for customer concerns

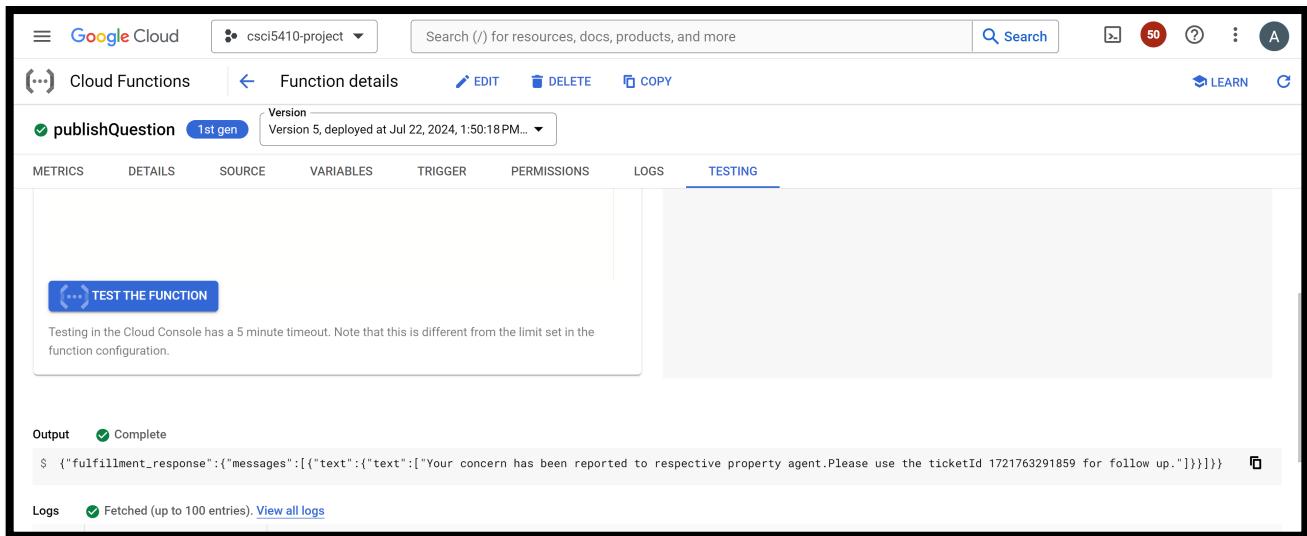


Figure 35: Cloud function for publish question

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

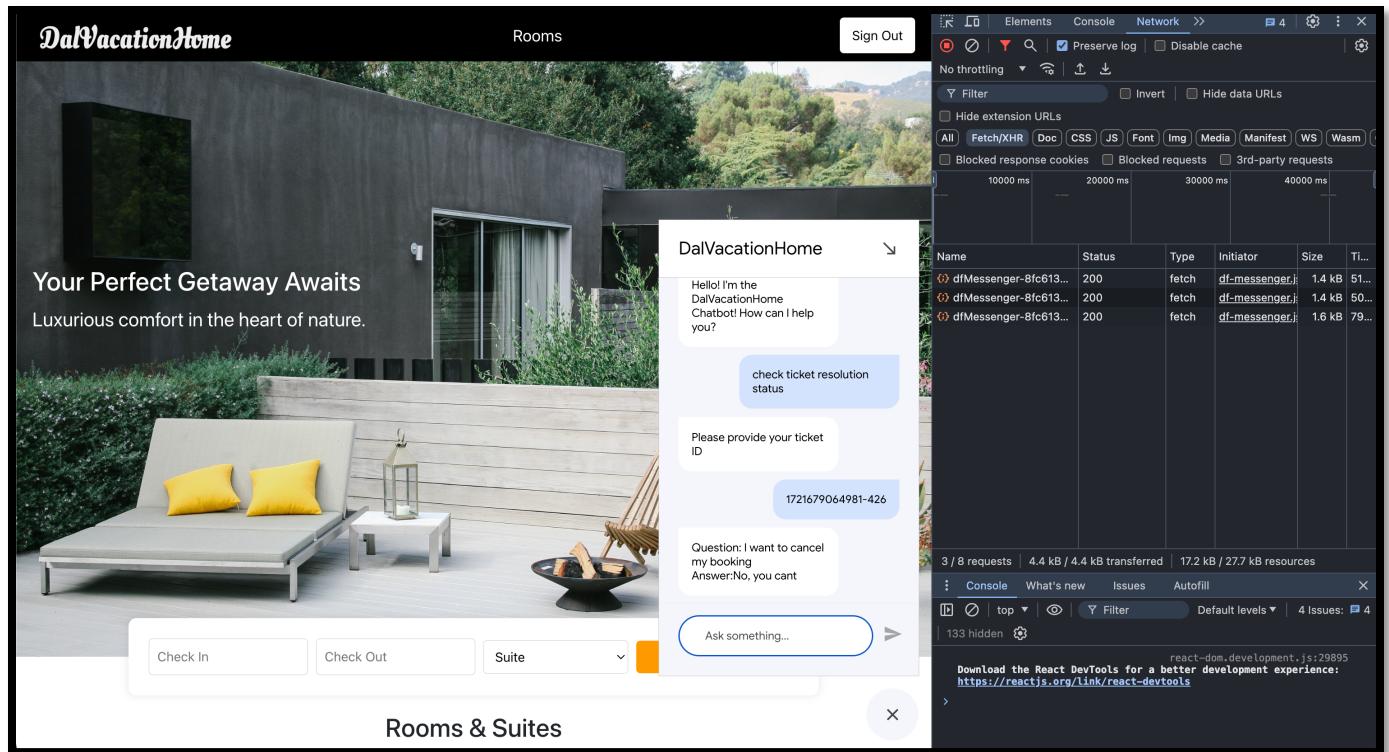


Figure 36: Dialogflow to check ticket resolution status

The screenshot shows the Google Cloud Functions interface. At the top, there's a navigation bar with 'Google Cloud', a project dropdown 'csci5410-project', a search bar 'Search (/) for resources, docs, products, and more', and various status indicators. Below the navigation is a header for 'Function details' with tabs for 'Cloud Functions', 'Function details', 'EDIT', 'DELETE', and 'COPY'. A sub-header indicates the function is 'CheckTicketResolution' (1st gen), version 8, deployed at Jul 22, 2024, 6:02:33 PM... The main area has tabs for 'METRICS', 'DETAILS', 'SOURCE', 'VARIABLES', 'TRIGGER', 'PERMISSIONS', 'LOGS', and 'TESTING'. The 'TESTING' tab is active, showing a code editor with the following JavaScript code:

```

4  "ticketid": "1721679064981-426"
5  }
6  }
7  }
8 }

```

Below the code editor is a button labeled 'TEST THE FUNCTION'. A note below it says: 'Testing in the Cloud Console has a 5 minute timeout. Note that this is different from the limit set in the function configuration.' At the bottom, there's an 'Output' section with a green checkmark and the text 'Complete', followed by a log entry: '\$ {"fulfillment_response": {"messages": [{"text": {"text": ["Question: I want to cancel my booking\\nAnswer:No, you cant"]}}]}}'.

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

Testcase 6: Handling negative scenario in chatbot

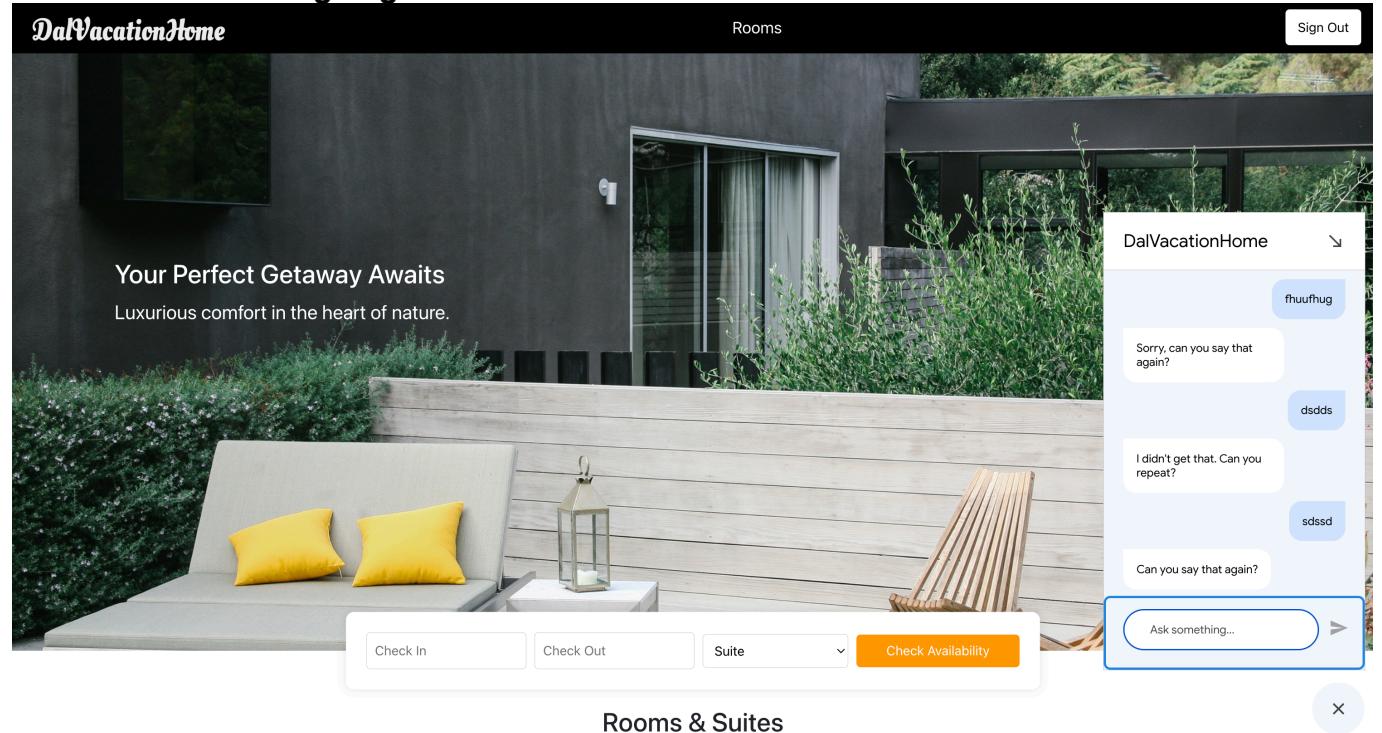


Figure 38: Handling negative scenario for chatbot

Module: Message Passing using GCP Pub/Sub

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

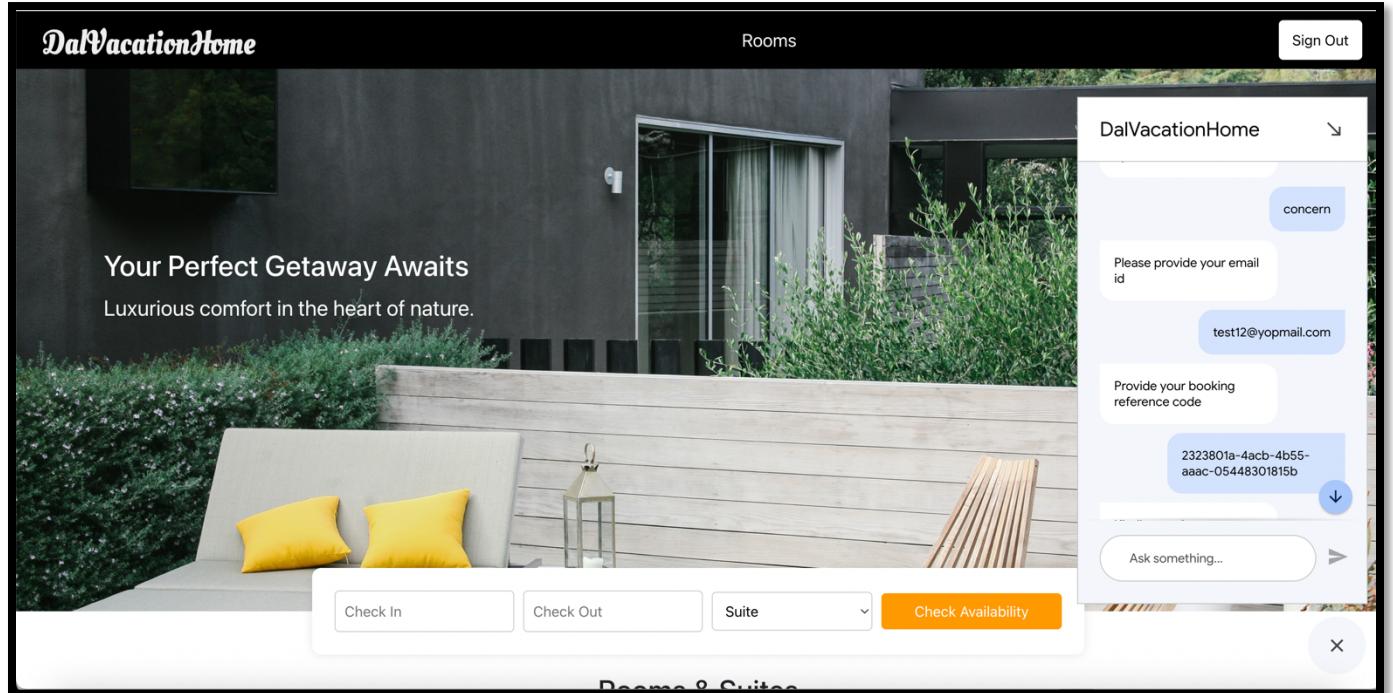


Figure 39: Initiate the concern from Chat bot

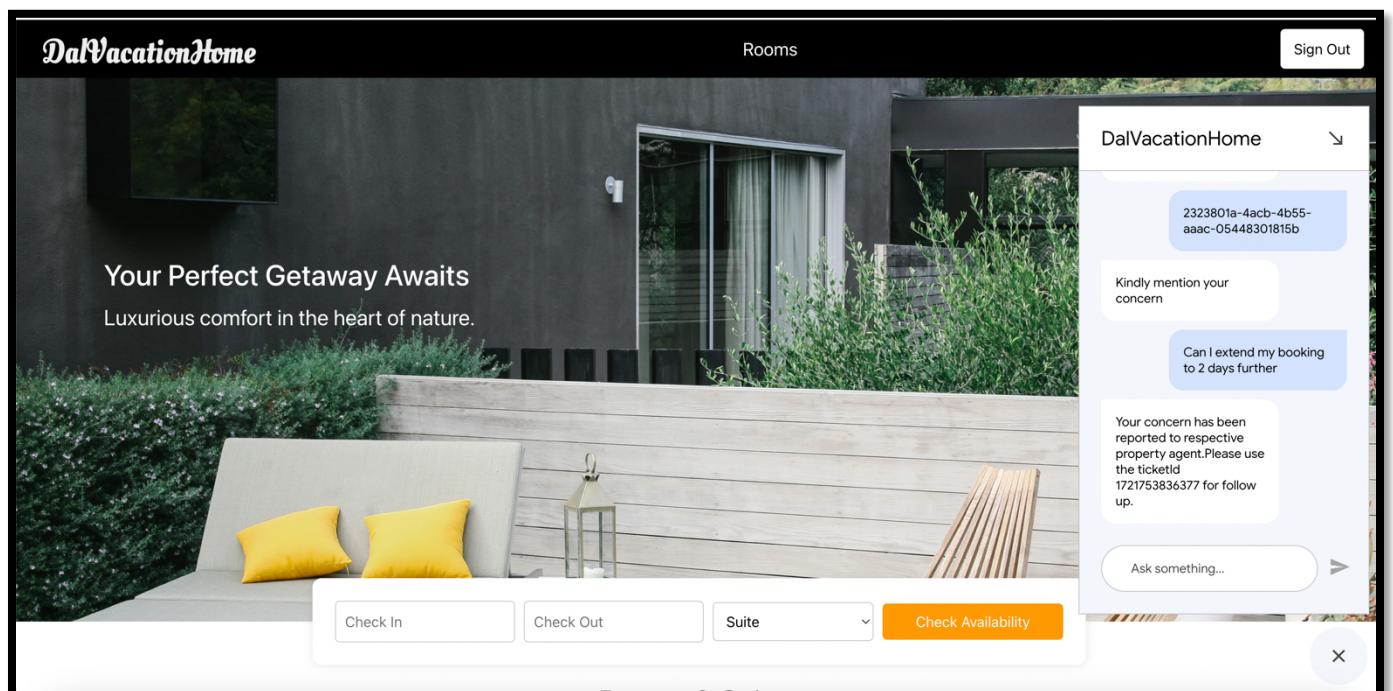


Figure 40: Provide relevant information about the concern

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

The screenshot shows a web application interface for a property agent. At the top, there is a navigation bar with the logo 'DalVacationHome' on the left, and 'Rooms', 'Customer Concerns', and 'Analytics' links on the right, along with a 'Sign Out' button. Below the navigation bar is a section titled 'Customer Concerns'. This section contains a table with three columns: 'Question Id', 'Question', and 'Reference Code'. There is one row of data: '1721753841785-707' in 'Question Id', 'Can I extend my booking to 2 days further' in 'Question', and '2323801a-4acb-4b55-aaac-05448301815b' in 'Reference Code'. To the right of the 'Question' column, there is a blue link labeled 'Answer'. The bottom right corner of the main content area has a small circular icon with a speaker symbol.

Figure 41: Customer concerns UI for Property Agent

The screenshot shows the Google Cloud Functions console. The top navigation bar includes the 'Google Cloud' logo, the project name 'csci5410-project', a search bar, and various navigation icons. The main area is titled 'Cloud Functions' and shows a single function named 'fetchQuestionForAgent'. It indicates this is '1st gen' and shows 'Version 7, deployed at Jul 18, 2024, 2:46:27PM...'. Below the function name are tabs for 'METRICS', 'DETAILS', 'SOURCE', 'VARIABLES', 'TRIGGER', 'PERMISSIONS', 'LOGS', and 'TESTING'. The 'TESTING' tab is currently selected. A large blue button labeled 'TEST THE FUNCTION' is visible. A note below it states: 'Testing in the Cloud Console has a 5 minute timeout. Note that this is different from the limit set in the function configuration.' At the bottom, there is an 'Output' section with a status message: 'Output Complete' and a JSON output block containing the following data:

```
$ [{"id": "1721329142593-100", "agentId": "ax583820@dal.ca", "questionId": "1721329142593-100", "question": "Can I stay one more day?", "questionTimeStamp": "2024-07-18T18:59:02.594Z", "a
```

Figure 42: Cloud function call to fetch all posted concerns

The screenshot shows a browser's developer tools Network tab with the 'Preview' tab selected. The response body is a JSON array containing one element:

```

[{"id": "1721753841785-707", "agentIds": "propertyagent01@yopmail.com", "questionId": "1721753841785-707", "agentId": "propertyagent01@yopmail.com", "answer": null, "answerTimeStamp": null, "id": "1721753841785-707", "question": "Can I extend my booking to 2 days further?", "questionId": "1721753841785-707", "questionTimeStamp": "2024-07-23T16:57:21.786Z", "referenceCode": "2323801a-4acb-4b55-aaac-05448301815b", "userId": "test12@yopmail.com"}]

```

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

Testcase 3: Property Agent answers the raised query

The screenshot shows a web application interface for a property agent. The top navigation bar includes 'Rooms', 'Customer Concerns', 'Analytics', and a 'Sign Out' button. The main content area has a title 'Reply to Customer' and a query: 'Query: Can I extend my booking to 2 days further'. Below the query is a text input field labeled 'Reply:' and a blue 'Submit Reply' button. In the bottom right corner of the page, there is a small circular icon with a speech bubble symbol.

Figure 44: Property Agent can reply to a concern

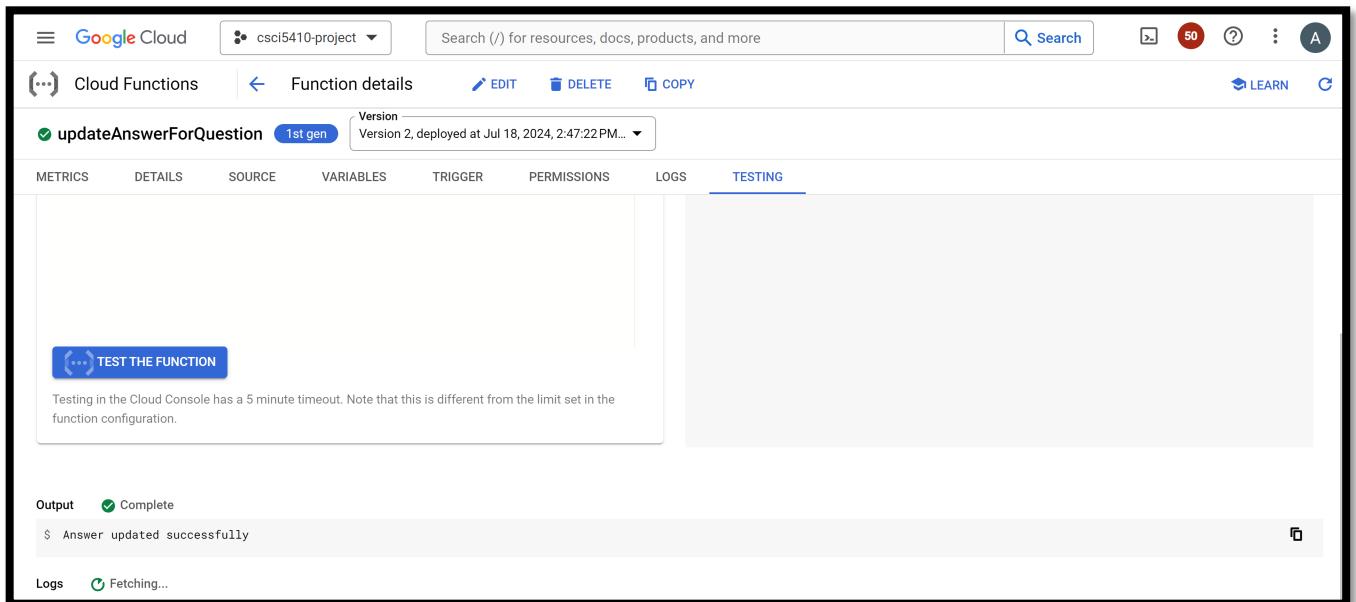


Figure 45: Cloud function call to update answer for specific concern

The screenshot shows a web application for 'DalVacationHome'. The top navigation bar includes 'Rooms', 'Customer Concerns', 'Analytics', and 'Sign Out'. The main content area is titled 'Reply to Customer' and contains the query: 'Query: Can I extend my booking to 2 days further'. Below the query is a text input field containing the response: 'Yes, you can provided there is room availability'. A 'Submit Reply' button is located below the input field. To the right of the input field is a small icon of a person speaking. At the bottom of the screen is a developer's browser toolbar with Network, Performance, and Memory tabs selected. The Network tab shows a single request entry:

Name	Headers	Payload	Preview	Response	Initiator	Timing
Request URL:				https://us-east1-csci5410-pro.cloudfunctions.net/updateAnswerForQuestion		
Request Method:				POST		
Status Code:				200 OK		
Remote Address:				216.239.36.54:443		
Referrer Policy:				strict-origin-when-cross-origin		
► Response Headers (12)						
▼ Request Headers						
request:authority:				us-east1-csci5410-pro.cloudfunctions.net		

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

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

The screenshot shows the Google Cloud Functions console. At the top, there's a navigation bar with 'Google Cloud', a dropdown for 'csci5410-project', a search bar, and various icons. Below the navigation is a header for 'Cloud Functions' with tabs for 'Function details', 'EDIT', 'DELETE', and 'COPY'. A sub-header shows the function name 'fetchQuestionAndAnswer', version '1st gen', and 'Version Version 9, deployed at Jul 18, 2024, 2:46:55PM...'. Below this are tabs for 'METRICS', 'DETAILS', 'SOURCE', 'VARIABLES', 'TRIGGER', 'PERMISSIONS', 'LOGS', and 'TESTING' (which is underlined). In the main area, there's a button labeled 'TEST THE FUNCTION'. A note below it says: 'Testing in the Cloud Console has a 5 minute timeout. Note that this is different from the limit set in the function configuration.' Under the 'TESTING' tab, there's an 'Output' section with a status of 'Complete' and a green checkmark. It displays the following JSON response:

```
S [{"id": "1721667077304-167", "data": {"agentId": "ax583820@dal.ca", "questionId": "1721667077304-167", "question": "I am not able to checkin today", "questionTimeStamp": "2024-07-22T16:5"}]
```

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

The screenshot shows a web application interface for 'DalVacationHome'. At the top, there's a logo 'DalVacationHome', a 'Rooms' link, and a 'Sign Out' button. Below the header, the title 'Customer Concerns' is displayed. A table lists customer concerns, each with a question, answer, and reference code. The columns are: Question Id, Question, Answer, and Reference Code. The data in the table is as follows:

Question Id	Question	Answer	Reference Code
1721679064981-426	I want to cancel my booking	No, you cant	554d10db-a94c-4884-9ca6-a59b36a08d12
1721753841785-707	Can I extend my booking to 2 days further	Yes, you can provided there is room availability	2323801a-4acb-4b55-aaac-05448301815b

Figure 48: Customer Concern UI for a customer

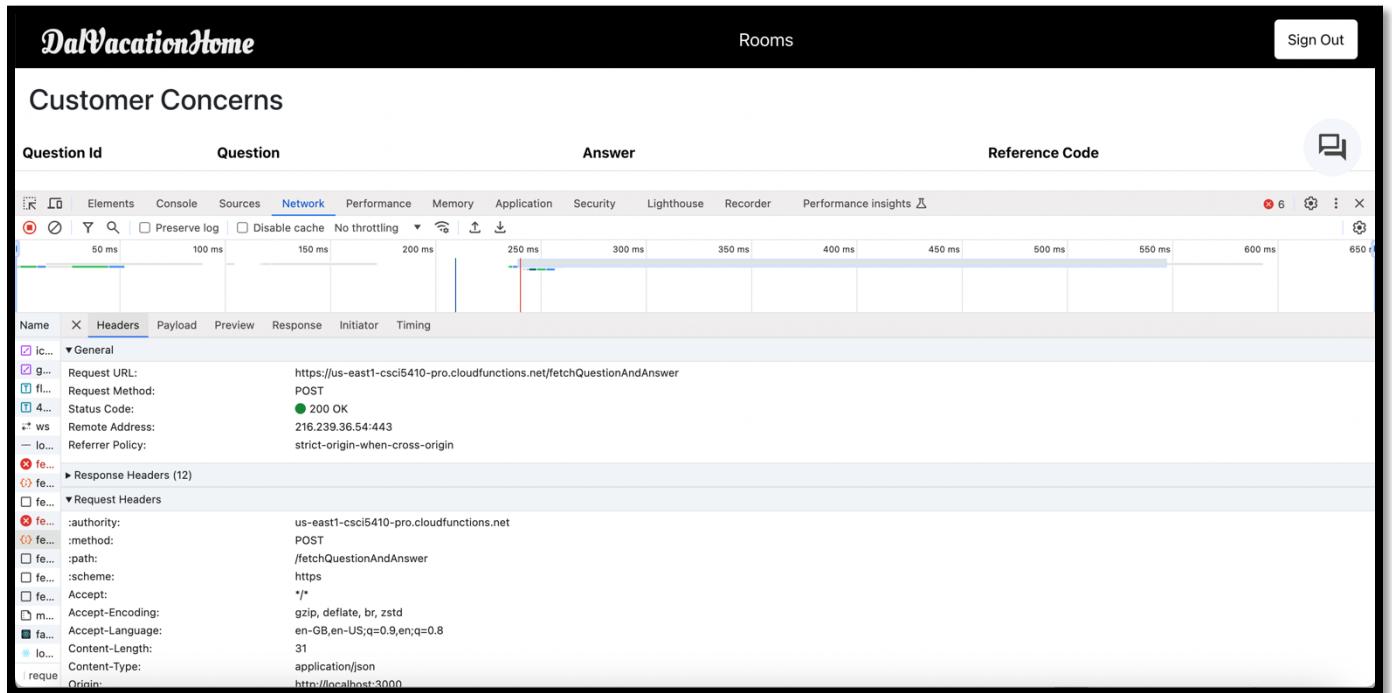


Figure 49: API Integration for fetching all raised concerns and thier answers for a customer

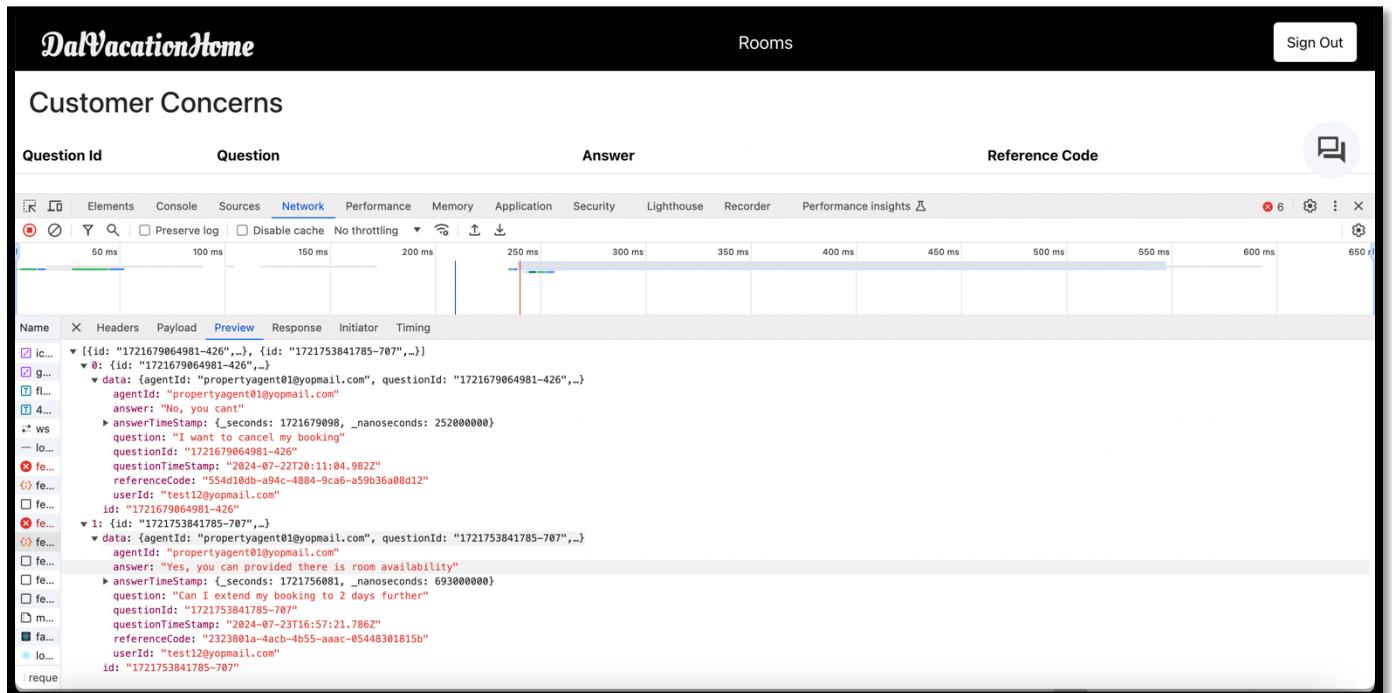
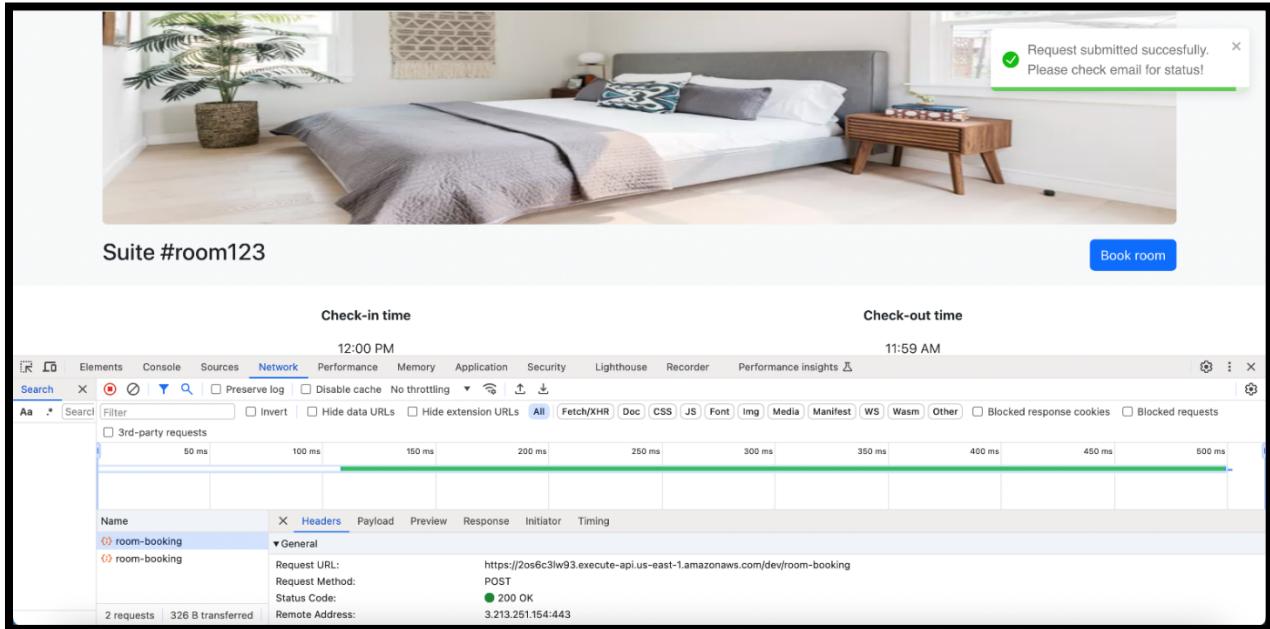


Figure 50: API response for fetching all concerns and answers

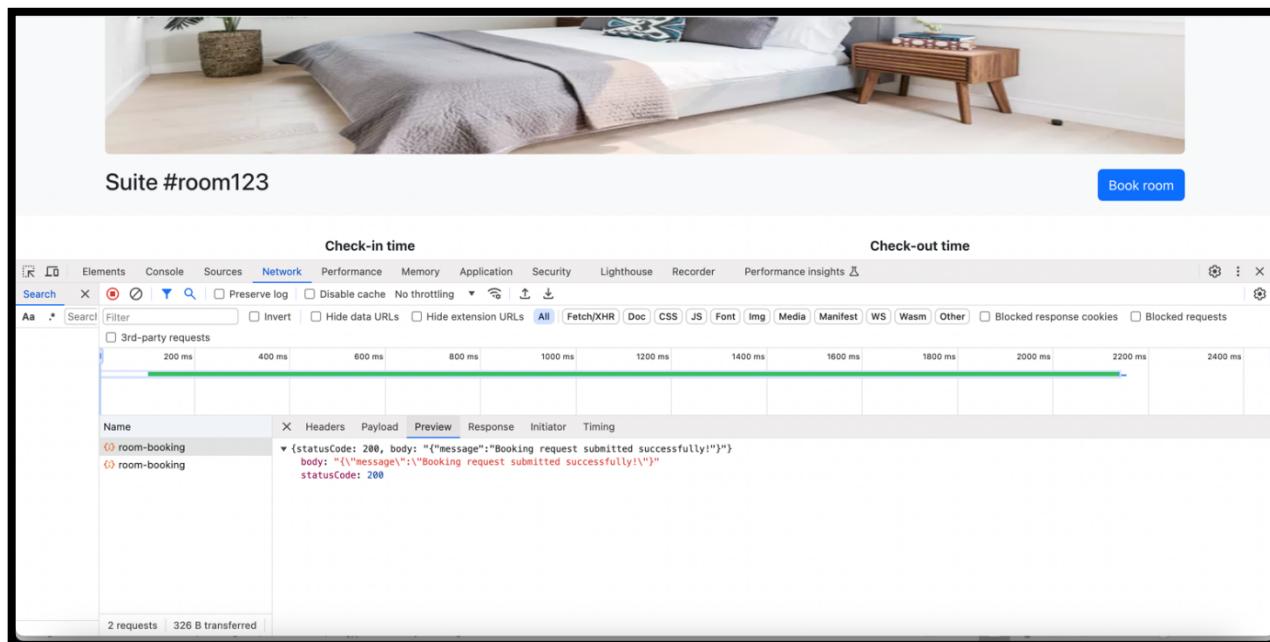
Module: Notifications

Room Booking and API integration Testcase 1: Successful Booking of Room



The screenshot shows a room booking interface. At the top right, a green success message box displays "Request submitted successfully. Please check email for status!". Below the message, a button labeled "Book room" is visible. The main content area shows a photograph of a bedroom with a double bed, a potted plant, and a small wooden side table. Below the photo, the text "Suite #room123" is displayed. A "Check-in time" field shows "12:00 PM" and a "Check-out time" field shows "11:59 AM". At the bottom, a browser's Network tab is open, showing a log of requests. One request from "room-booking" is highlighted, showing a successful POST request to "https://2os6c3lw93.execute-api.us-east-1.amazonaws.com/dev/room-booking" with a status code of 200 OK.

Figure 51: Room Booking API integration



This screenshot is similar to Figure 51, showing the same room booking interface. The success message at the top right has disappeared. The "Check-in time" field now shows "12:00 PM" and the "Check-out time" field shows "11:59 AM". The "Book room" button is still present. The Network tab in the browser shows the same successful POST request to the room booking API, but the response payload is now visible in the "Preview" tab of the Network tool. The response body is a JSON object: {statusCode: 200, body: {"message": "Booking request submitted successfully!"}}. The status code is also explicitly listed as 200.

Figure 52: Room Booking API response

The screenshot shows the AWS Lambda test interface. The top navigation bar includes tabs for Code, Test (which is selected), Monitor, Configuration, Aliases, and Versions. The main content area displays a green checkmark icon indicating the function succeeded. A link to the logs is provided. Below this, a section titled "Details" shows the execution log content:

```
{
  "statusCode": 200,
  "body": "{\"message\":\"Booking request submitted successfully!\\"}"
}
```

Under the "Summary" section, various metrics are listed:

- Code SHA-256: f40cMlkRkr9WZJqAZ0gwfnD286U6HrWCWwleAVoOS9E=
- Request ID: 9a1a5db2-1fad-4657-b9c6-2a3dab8cf0a2
- Init duration: 686.56 ms
- Billed duration: 1014 ms
- Max memory used: 90 MB
- Execution time: 32 seconds ago (July 23, 2024 at 03:13 PM ADT)
- Function version: \$LATEST
- Duration: 1013.44 ms
- Resources configured: 128 MB

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

Testcase 2: Booking Request Confirmation

The screenshot shows an email from DALVacationHome. The subject is "Booking Confirmation". The email body starts with "Dear User," and a thank you message for the booking. It then lists booking details: Registration ID, Room ID, Booking Start Date, Booking End Date, and Total Booking Days. The message concludes with a note about hosting and ends with "Best regards, The DALVacationHome Team". At the bottom, there is an unsubscribe link and a note about not replying directly.

Booking Confirmation Inbox x

DALVacationHome <no-reply@sns.amazonaws.com> to me 5:06 PM (0 minutes ago) ☆ ☺ ↵ ⋮

Dear User,

Thank you for your interest in booking with DALVacationHome! We are pleased to inform you that your booking has been successfully confirmed. Here are your booking details:

Registration ID: a689cf7e-b33d-44ae-a6eb-9854ec62acf3
Room ID: room2209
Booking Start Date: 2024-07-15
Booking End Date: 2024-07-21
Total Booking Days: 5

We look forward to hosting you!

Best regards,
The DALVacationHome Team

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
<https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:401985591036:RoomBookingRequest:27210536-3424-4d12-bbe9-a1634df7ea4d&EndpointToken=nikitadavies98@gmail.com>

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at <https://aws.amazon.com/support>

Figure 54: Email for booking Confirmation

Testcase 3: Booking Request Rejection

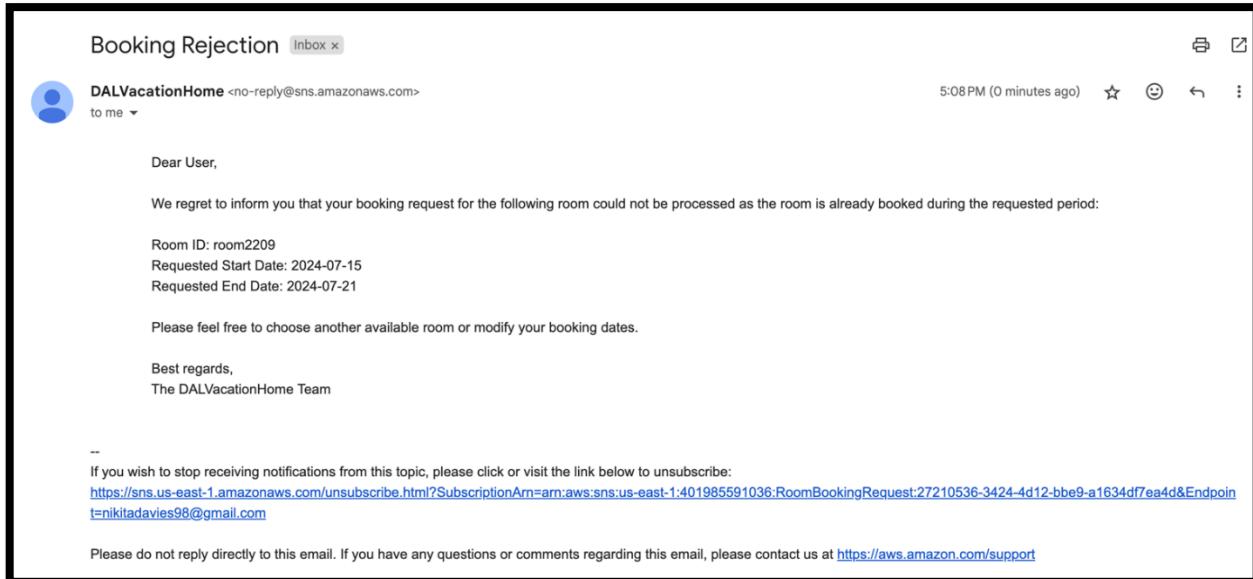


Figure 55: Email notification for Booking Rejection

Authentication module test cases:

Test case 1:

User sign up data entry before email verification

The screenshot shows a "Sign Up" form with the following fields:

mrunalpatkar60@gmail.com
Mrunal	
What city were you born in?	(dropdown menu)
Mumbai	3
Registered Customers	hal
Halifax	123456
1234567890	Sign Up
Go to Sign In	
Continue as Guest	

Figure 56. User sign up form.

ARN	6																																			
arn:aws:cognito-idp:us-east-1:492729822429:userpool/us-east-1_11n3jFVog	Advanced security Disabled																																			
▶ Getting started																																				
Users Groups Sign-in experience Sign-up experience Messaging App integration Advanced security User pool properties																																				
Users (6) <small>Info</small> View, edit, and create users in your user pool. Users that are enabled and confirmed can sign in to your user pool.																																				
Property: <input type="button" value="User name"/> <input type="text" value="Search users by attribute"/>																																				
<table border="1"> <thead> <tr> <th>User name</th> <th>Email address</th> <th>Email verified</th> <th>Confirmation status</th> <th>Status</th> </tr> </thead> <tbody> <tr> <td>9488f498-2061-7049-c5ac-a38958c...</td> <td>luffsnape@gmail.com</td> <td>Yes</td> <td>Confirmed</td> <td>Disabled</td> </tr> <tr> <td>9498b4d8-2041-70c6-d3a9-6082ae61...</td> <td>mrunalpatkar32@gmail.com</td> <td>Yes</td> <td>Confirmed</td> <td>Enabled</td> </tr> <tr> <td>d40854d8-e091-703c-e605-b219475c...</td> <td>rameezparkar2310@gmail.com</td> <td>Yes</td> <td>Confirmed</td> <td>Enabled</td> </tr> <tr> <td>4448c478-b091-7027-b691-52744b7...</td> <td>mrunalpatkar326@gmail.com</td> <td>Yes</td> <td>Confirmed</td> <td>Enabled</td> </tr> <tr> <td>04d8c428-30d1-70bb-1cf5-d30a0354...</td> <td>mrunalpatkar60@gmail.com</td> <td>No</td> <td>Unconfirmed</td> <td>Enabled</td> </tr> <tr> <td>44184468-70a1-70e7-0722-f64a345f...</td> <td>rameezparkar21@gmail.com</td> <td>Yes</td> <td>Confirmed</td> <td>Enabled</td> </tr> </tbody> </table>		User name	Email address	Email verified	Confirmation status	Status	9488f498-2061-7049-c5ac-a38958c...	luffsnape@gmail.com	Yes	Confirmed	Disabled	9498b4d8-2041-70c6-d3a9-6082ae61...	mrunalpatkar32@gmail.com	Yes	Confirmed	Enabled	d40854d8-e091-703c-e605-b219475c...	rameezparkar2310@gmail.com	Yes	Confirmed	Enabled	4448c478-b091-7027-b691-52744b7...	mrunalpatkar326@gmail.com	Yes	Confirmed	Enabled	04d8c428-30d1-70bb-1cf5-d30a0354...	mrunalpatkar60@gmail.com	No	Unconfirmed	Enabled	44184468-70a1-70e7-0722-f64a345f...	rameezparkar21@gmail.com	Yes	Confirmed	Enabled
User name	Email address	Email verified	Confirmation status	Status																																
9488f498-2061-7049-c5ac-a38958c...	luffsnape@gmail.com	Yes	Confirmed	Disabled																																
9498b4d8-2041-70c6-d3a9-6082ae61...	mrunalpatkar32@gmail.com	Yes	Confirmed	Enabled																																
d40854d8-e091-703c-e605-b219475c...	rameezparkar2310@gmail.com	Yes	Confirmed	Enabled																																
4448c478-b091-7027-b691-52744b7...	mrunalpatkar326@gmail.com	Yes	Confirmed	Enabled																																
04d8c428-30d1-70bb-1cf5-d30a0354...	mrunalpatkar60@gmail.com	No	Unconfirmed	Enabled																																
44184468-70a1-70e7-0722-f64a345f...	rameezparkar21@gmail.com	Yes	Confirmed	Enabled																																

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

Test case 2:

User sign-up data entry after email verification

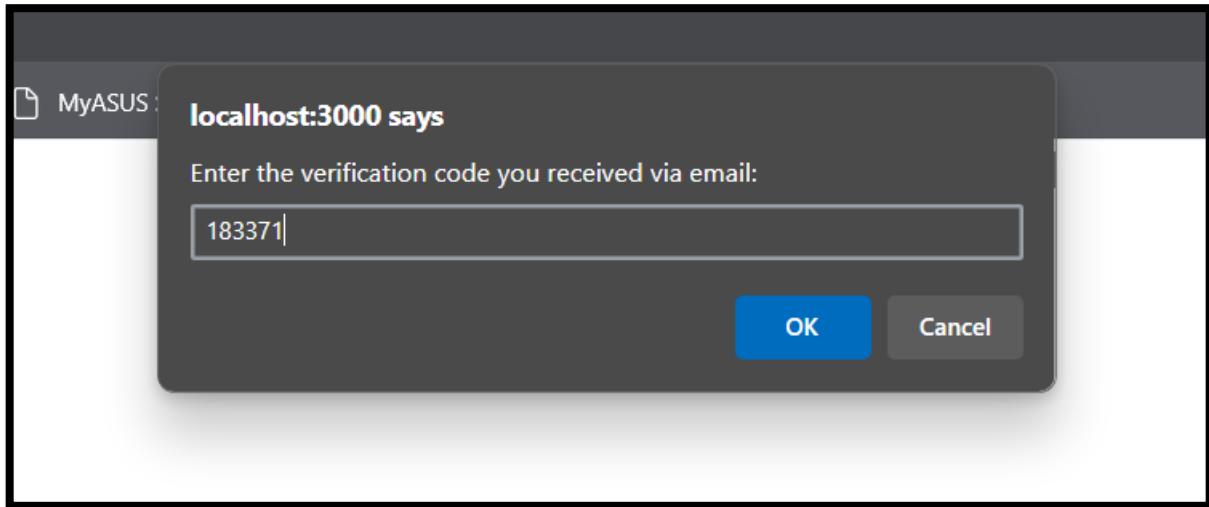


Figure 58: Verification code inserted.

Getting started																																				
Users																																				
Groups																																				
Sign-in experience																																				
Sign-up experience																																				
Messaging																																				
App integration																																				
Advanced security																																				
User pool properties																																				
<h3>Users (6) <small>Info</small></h3> <p>View, edit, and create users in your user pool. Users that are enabled and confirmed can sign in to your user pool.</p> <p>Property: <input type="button" value="User name"/> <input type="text" value="Search users by attribute"/></p> <table><thead><tr><th>User name</th><th>Email address</th><th>Email verified</th><th>Confirmation status</th><th>Status</th></tr></thead><tbody><tr><td>44184468-70a1-70e7-0722-f64a345f...</td><td>rameezparkar21@gmail.com</td><td>Yes</td><td>Confirmed</td><td>Enabled</td></tr><tr><td>d40854d8-e091-703c-e605-b219475c...</td><td>rameezparkar2310@gmail.com</td><td>Yes</td><td>Confirmed</td><td>Enabled</td></tr><tr><td>4448c478-b091-7027-b691-52744b7...</td><td>mrunalpatkar32@gmail.com</td><td>Yes</td><td>Confirmed</td><td>Enabled</td></tr><tr><td>04d8c428-30d1-70bb-1cf5-d30a0354...</td><td>mrunalpatkar60@gmail.com</td><td>Yes</td><td>Confirmed</td><td>Enabled</td></tr><tr><td>9488f498-2061-7049-c5ac-a38958c6...</td><td>luffsnape@gmail.com</td><td>Yes</td><td>Confirmed</td><td>Disabled</td></tr><tr><td>9498b4d8-2041-70c6-d3a9-6082ae61...</td><td>mrunalpatkar32@gmail.com</td><td>Yes</td><td>Confirmed</td><td>Enabled</td></tr></tbody></table>		User name	Email address	Email verified	Confirmation status	Status	44184468-70a1-70e7-0722-f64a345f...	rameezparkar21@gmail.com	Yes	Confirmed	Enabled	d40854d8-e091-703c-e605-b219475c...	rameezparkar2310@gmail.com	Yes	Confirmed	Enabled	4448c478-b091-7027-b691-52744b7...	mrunalpatkar32@gmail.com	Yes	Confirmed	Enabled	04d8c428-30d1-70bb-1cf5-d30a0354...	mrunalpatkar60@gmail.com	Yes	Confirmed	Enabled	9488f498-2061-7049-c5ac-a38958c6...	luffsnape@gmail.com	Yes	Confirmed	Disabled	9498b4d8-2041-70c6-d3a9-6082ae61...	mrunalpatkar32@gmail.com	Yes	Confirmed	Enabled
User name	Email address	Email verified	Confirmation status	Status																																
44184468-70a1-70e7-0722-f64a345f...	rameezparkar21@gmail.com	Yes	Confirmed	Enabled																																
d40854d8-e091-703c-e605-b219475c...	rameezparkar2310@gmail.com	Yes	Confirmed	Enabled																																
4448c478-b091-7027-b691-52744b7...	mrunalpatkar32@gmail.com	Yes	Confirmed	Enabled																																
04d8c428-30d1-70bb-1cf5-d30a0354...	mrunalpatkar60@gmail.com	Yes	Confirmed	Enabled																																
9488f498-2061-7049-c5ac-a38958c6...	luffsnape@gmail.com	Yes	Confirmed	Disabled																																
9498b4d8-2041-70c6-d3a9-6082ae61...	mrunalpatkar32@gmail.com	Yes	Confirmed	Enabled																																

Figure 59: Email verified in cognito

Test case 3:

User Signed up successfully (Welcome mail triggered)

 AWS Notifications <no-reply@sns.amazonaws.com>17:06 (1 minute ago) ☆ ✉ ↶

to me ▼

Welcome to DALVacationHome!

Dear Valued Guest,

We're thrilled to welcome you to DALVacationHome! Thank you for joining our community.

To help you get started, here are some essential links:

- Your Profile: <https://www.dalvacationhome.com/profile>
- Make a Reservation: <https://www.dalvacationhome.com/reservations>
- Customer Support: <https://www.dalvacationhome.com/support>

At DALVacationHome, we're dedicated to providing you with the best vacation experience. Explore our website to discover amazing vacation homes, exclusive deals, and more!

If you need any assistance, don't hesitate to contact us. Our team is always here to help.

Happy vacationing!

Best Regards,
The DALVacationHome Team

You received this email because you registered on DALVacationHome. If you didn't register, please ignore this email.

--

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:139157793567:UserCreatedTopic_3d9741c-837a-4a77-b639-bd8d914dd1ce&EndPoint=mrunalpatkar60@gmail.com

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at <https://aws.amazon.com/support>

Figure 60: Welcome mail is sent after successful sign up

Welcome mail (Sign-up) Lambda manual testing

The screenshot shows the AWS Lambda function execution details for a successful run. It includes the execution log, summary metrics like execution time and memory usage, and log output.

Executing function: succeeded (Logs [\[2\]](#))

Details

The area below shows the last 4 KB of the execution log.

```
{ "statusCode": 200, "body": "{\"Message sent to SQS\"}" }
```

Summary

Code SHA-256	Execution time
MAOGmKrs6whdRxYalNAlcGNhq4qdPcCf4UDW1stMCPg=	10 seconds ago (July 23, 2024 at 04:34 PM ADT)
Request ID	Function version
ed40e5a6-dd4b-4b49-9482-f539027eb704	\$LATEST
Duration	Billed duration
38.53 ms	39 ms
Resources configured	Max memory used
128 MB	85 MB

Log output

The section below shows the logging calls in your code. [Click here \[2\]](#) to view the corresponding CloudWatch log group.

```
START RequestId: ed40e5a6-dd4b-4b49-9482-f539027eb704 Version: $LATEST
END RequestId: ed40e5a6-dd4b-4b49-9482-f539027eb704
REPORT RequestId: ed40e5a6-dd4b-4b49-9482-f539027eb704 Duration: 38.53 ms Billed Duration: 39 ms Memory Size: 128 MB Max Memory Used: 85 MB
```

Figure 61: 4Register user notification lambda testing

The screenshot shows a Gmail inbox with a welcome email from DALVacationHome. The email is triggered by AWS notifications and contains a message to a valued guest, links for getting started, and information about the vacation home service.

Gmail

Mrunal Patkar <mrunalpatkar60@gmail.com>

Welcome to DALVacationHome

AWS Notifications <no-reply@sns.amazonaws.com> **23 July 2024 at 16:34**
To: mrunalpatkar60@gmail.com

Welcome to DALVacationHome!

Dear Valued Guest,

We're thrilled to welcome you to DALVacationHome! Thank you for joining our community.

To help you get started, here are some essential links:

- Your Profile: <https://www.dalvacationhome.com/profile>
- Make a Reservation: <https://www.dalvacationhome.com/reservations>
- Customer Support: <https://www.dalvacationhome.com/support>

At DALVacationHome, we're dedicated to providing you with the best vacation experience. Explore our website to discover amazing vacation homes, exclusive deals, and more!

If you need any assistance, don't hesitate to contact us. Our team is always here to help.

Happy vacationing!

Best Regards,
The DALVacationHome Team

You received this email because you registered on DALVacationHome. If you didn't register, please ignore this email.

Figure 62: Welcome mail triggered by the manual testing

Login mail (Sign in) Lambda manual testing

The screenshot shows the AWS Lambda function execution details for a successful execution. The summary includes:

- Code SHA-256: QknOPhulfI66lVykabVPnYpWlj3xo8IROC6DM0ijmoU=
- Request ID: c2225716-aa32-4e35-a41a-a4380edf545e
- Init duration: 306.12 ms
- Billed duration: 825 ms
- Max memory used: 84 MB
- Execution time: 2 seconds ago (July 23, 2024 at 04:30 PM ADT)
- Function version: \$LATEST
- Duration: 824.71 ms
- Resources configured: 128 MB

The log output shows the following message:

```
START RequestId: c2225716-aa32-4e35-a41a-a4380edf545e Version: $LATEST
END RequestId: c2225716-aa32-4e35-a41a-a4380edf545e Duration: 824.71 ms      Billed Duration: 825 ms Memory Size: 128 MB      Max Memory Used: 84 MB  Init Duration: 306.12 ms
REPORT RequestId: c2225716-aa32-4e35-a41a-a4380edf545e Duration: 824.71 ms      Billed Duration: 825 ms Memory Size: 128 MB      Max Memory Used: 84 MB  Init Duration: 306.12 ms
```

Figure 63: Login notification lambda testing

The screenshot shows an incoming email in the Gmail inbox. The email is from AWS Notifications <no-reply@sns.amazonaws.com> and is addressed to Mrunal Patkar <mrunalpatkar60@gmail.com>. The subject is "Welcome to DALVacationHome". The email body contains:

Hi mrunalpatkar60@gmail.com,

You've successfully logged into your DALVacationHome account. Ready to explore your next vacation destination?

Check out the latest vacation homes and exclusive deals available for you.

We're excited to help you find your perfect getaway!

Warm regards,
The DALVacationHome Team

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
<https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:492729822429:LoginTopic:ce9a406f-9982-4997-85f0-d49034b1308c&Endpoint=mrunalpatkar60@gmail.com>

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at <https://aws.amazon.com/support>

Figure 64: Login mail triggered by the manual testing.

Test case 4:

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

Group information				
Group name	RegisteredCustomers	Description	Users who are also registered customers	
IAM Role ARN	arn:aws:iam::492729822429:role/LabRole	Precedence	-	
Group members (5) Info		Edit		
User name	Email address	Email verified	Confirmation status	Status
d40854d8-e091-703c-e605-b219475c...	rameezparkar2310@gmail.com	Yes	Confirmed	Enabled
44184468-70a1-70e7-0722-f64a345f...	rameezparkar21@gmail.com	Yes	Confirmed	Enabled
04d8c428-30d1-70bb-1cf5-d30a0354...	mrunalpatkar60@gmail.com	Yes	Confirmed	Enabled
4448c478-b091-7027-b691-52744b7...	mrunalpatkar32@gmail.com	Yes	Confirmed	Enabled
9498b4d8-2041-70c6-d3a9-6082ae61...	mrunalpatkar32@gmail.com	Yes	Confirmed	Enabled

Figure 65: User sorted in respective groups

Items returned (4)								Actions ▾	Create item
	User ID (String)	contact_no	email	name	pincode	security_answer	security_question	user_type	
	mrunalpatkar60@gm...	1234567890	mrunalpatk...	Mrunal	123456	Mumbai	q5	RegisteredC...	

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

Test case 4:

2 factor authentication front end

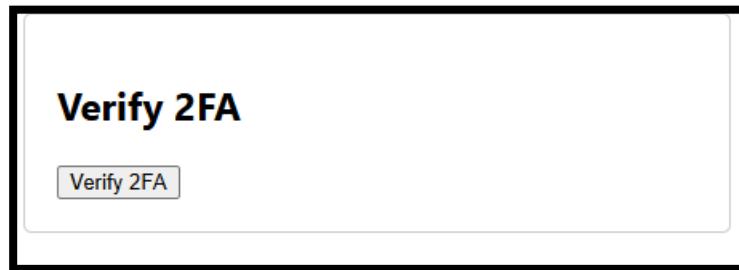


Figure 67: 2FA Frontend component.

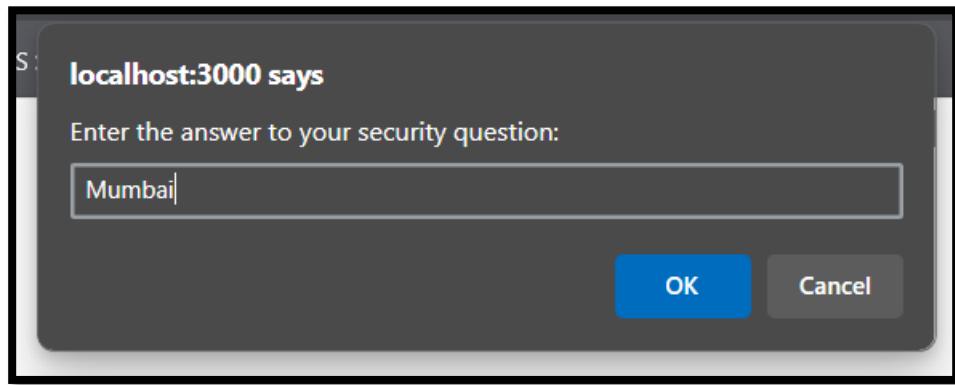


Figure 68: 2FA security answer

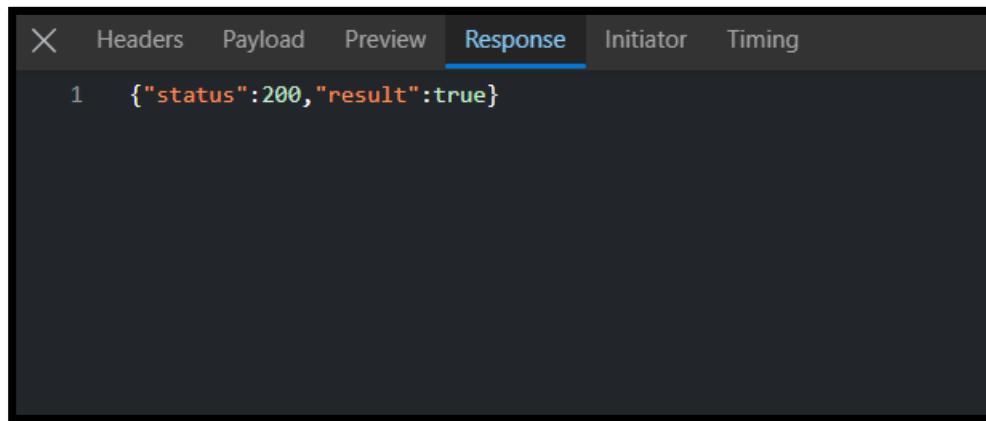


Figure 69: 2FA success response

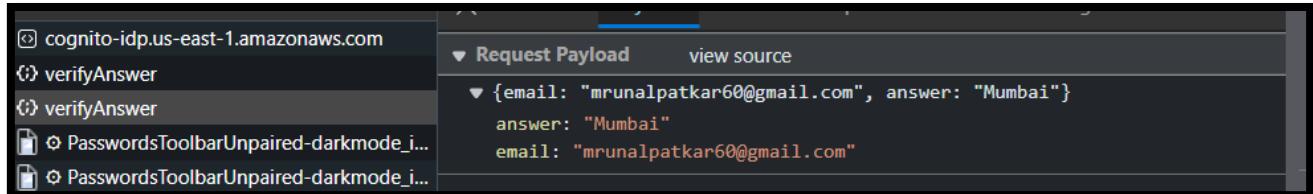


Figure 70: 2FA frontend payload.

2FA lambda manual testing

The screenshot shows the AWS Lambda execution details for a successful function execution. The summary includes:

- Code SHA-256: 13xkzD+K1orqblseCxIkeOd8KN2c34vjlwO2T88TLkl=
- Request ID: 47cab55b-1c8b-4c8f-82b8-71ef17acad2a
- Init duration: 606.52 ms
- Billed duration: 948 ms
- Max memory used: 90 MB
- Execution time: 2 seconds ago (July 23, 2024 at 04:22 PM ADT)
- Function version: \$LATEST
- Duration: 947.30 ms
- Resources configured: 128 MB

Log output

The section shows CloudWatch logs with the following entries:

```
START RequestId: 47cab55b-1c8b-4c8f-82b8-71ef17acad2a Version: $LATEST
2024-07-23T09:22:11.529Z        47cab55b-1c8b-4c8f-82b8-71ef17acad2a    INFO    ****email value is mrunalpatkar60@gmail.com
2024-07-23T09:22:11.537Z        47cab55b-1c8b-4c8f-82b8-71ef17acad2a    INFO    ****answer value is mumbai
END RequestId: 47cab55b-1c8b-4c8f-82b8-71ef17acad2a
REPORT RequestId: 47cab55b-1c8b-4c8f-82b8-71ef17acad2a Duration: 947.30 ms    Billed Duration: 948 ms Memory Size: 128 MB    Max Memory Used: 90 MB    Init Duration: 606.52 ms
```

Figure 71: 2FA manual testing

Test case 5:

3 factor authentication front end:

The screenshot shows a browser window with a "Welcome!" message and an "Access Token" field. Below the token field is a "Sign Out" button.

The Network tab shows several requests:

Name	Headers	Payload	Preview	Response	Initiator	Timing
cognito-idp.us-east-1.amazonaws.com						
verifyAnswer						
verifyAnswer						
PasswordsToolbarUnpaired-darkmode_i...						
PasswordsToolbarUnpaired-darkmode_i...						
PasswordsToolbarUnpaired-darkmode_i...						
PasswordsToolbarUnpaired-darkmode_i...						
PasswordsToolbarUnpaired-darkmode_i...						
verifyCipher						
verifyCipher						
notify-login						
notify-login						
PasswordsToolbarUnpaired-darkmode_i...						
PasswordsToolbarUnpaired-darkmode_i...						
PasswordsToolbarUnpaired-darkmode_i...						
PasswordsToolbarUnpaired-darkmode_i...						

Figure 72: 3FA frontend payload

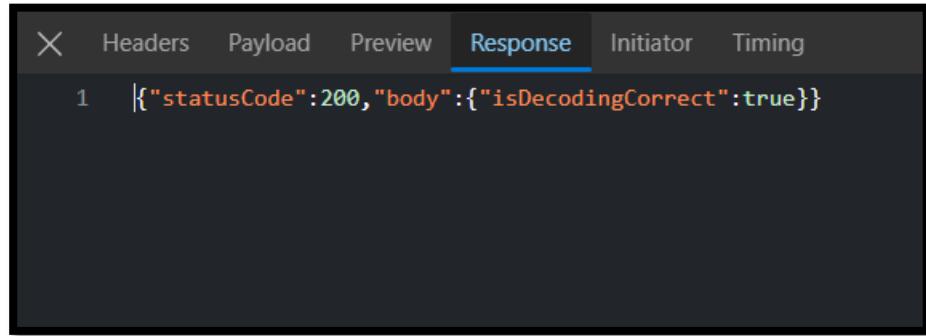


Figure 73: 3FA success response.

3FA Lambda manually tested

Summary	Log output
Request ID: 7b3f5734-a9dd-4c68-9c00-542a8297b276 Duration: 84.09 ms Resources configured: 128 MB	Execution time: now (July 23, 2024 at 04:28 PM ADT) Function version: \$LATEST Billed duration: 85 ms Max memory used: 91 MB
The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group. <pre> START RequestId: 7b3f5734-a9dd-4c68-9c00-542a8297b276 Version: \$LATEST END RequestId: 7b3f5734-a9dd-4c68-9c00-542a8297b276 REPORT RequestId: 7b3f5734-a9dd-4c68-9c00-542a8297b276 Duration: 84.09 ms Billed Duration: 85 ms Memory Size: 128 MB Max Memory Used: 91 MB </pre>	

Figure 74: 3FA Manually tested

Test case 6:

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

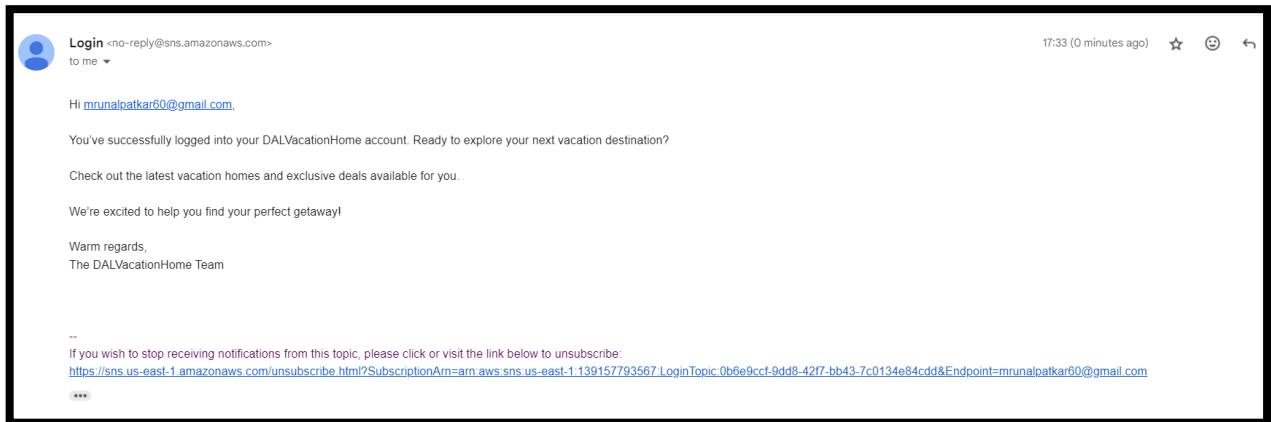


Figure 75: Login successfully mail

Fully Integrated sign-in flow:

Test case 1:

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

A screenshot of the Chrome DevTools Storage panel. The left sidebar shows the application manifest, service workers, and storage sections. Under 'Storage', the 'Local storage' section is expanded, showing entries for 'chrome-extension://mfbc' and 'http://localhost:3000'. The 'Session storage' section is also visible. The right panel displays a table with columns 'Key' and 'Value', which is currently empty. A message at the bottom right says 'Select a value to preview'. The main window shows a 'DalVacationHome' sign-in page with fields for email and password, and buttons for 'Sign In', 'Go to Sign Up', and 'Continue as Guest'.

Figure 76: Empty local storage before the user signs in

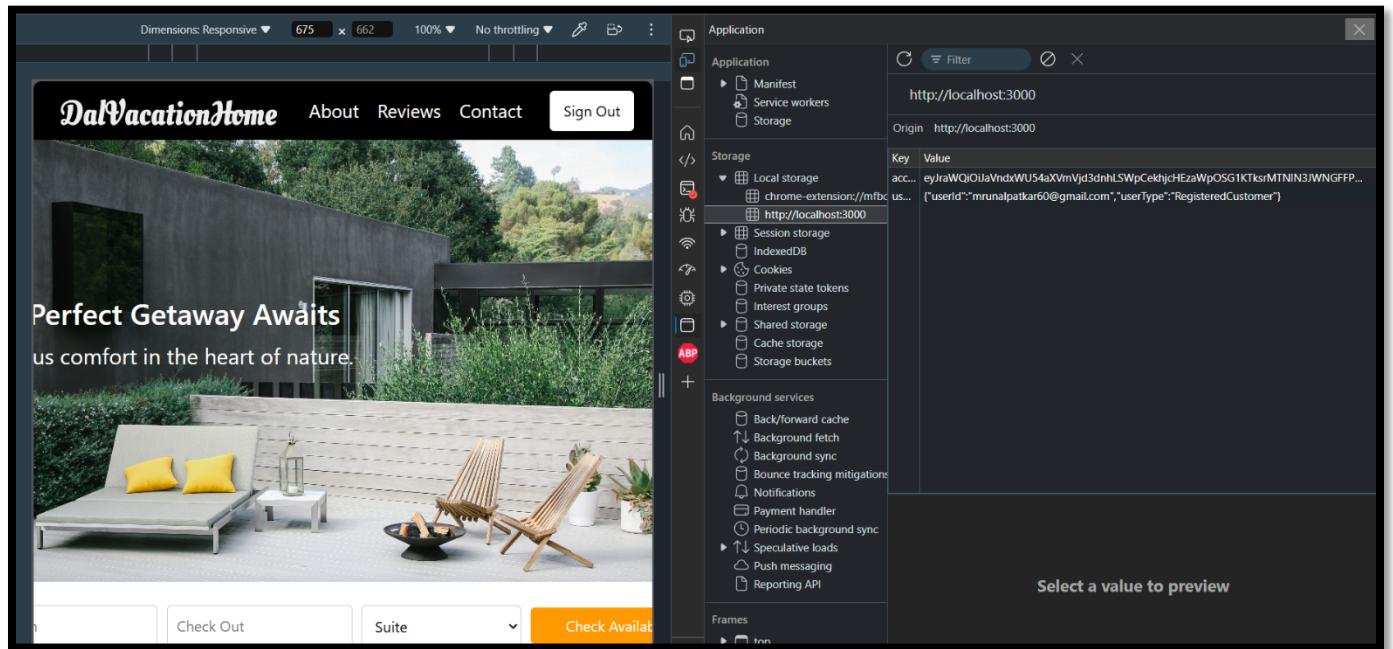


Figure 77: Auth token and user details are stored once the user signs in successfully

Test case 2:

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

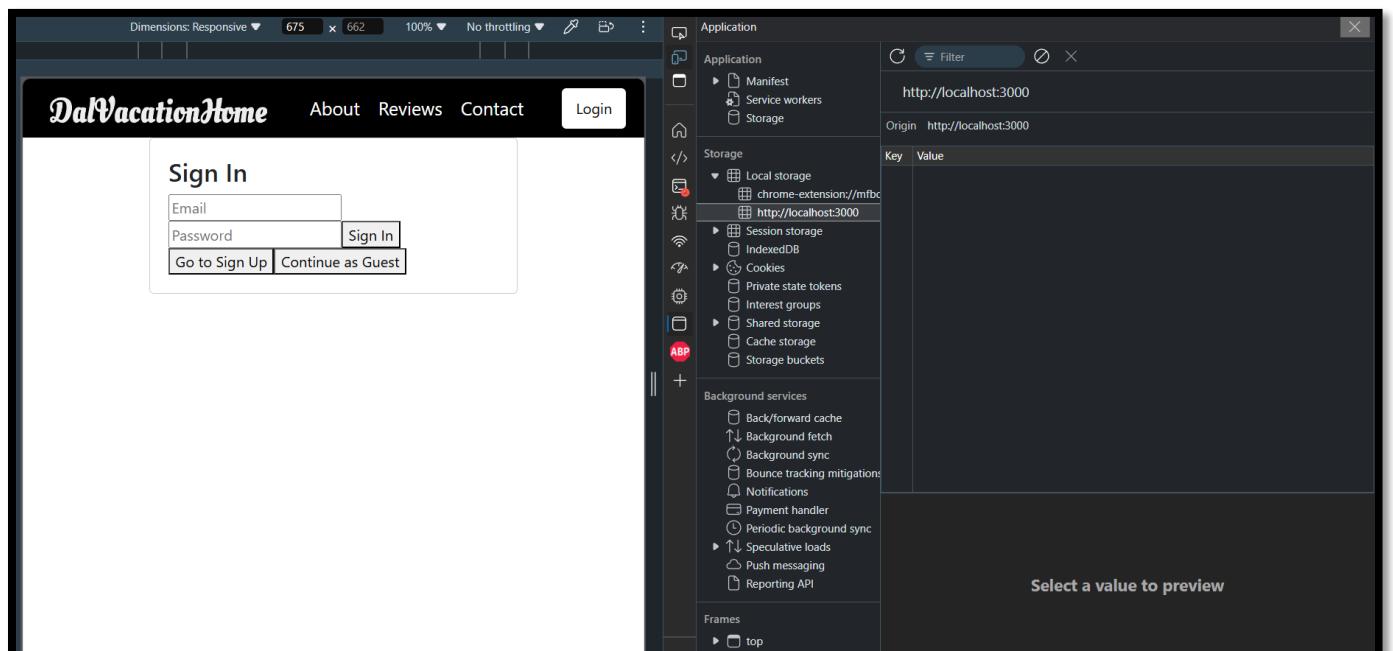


Figure 78: Auth token and User details get deleted after clicking on the "Sign-out" button

Test case 3:

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

The screenshot shows the AWS CloudWatch Log Details interface. At the top, there's a header bar with tabs like 'Autopreview' and 'View table details'. Below the header, a message says 'Completed. Read capacity units consumed: 1'. The main area is titled 'Items returned (38)' and contains a table with columns: LogId (String), SignInDate, SignInTime, UserEmail, and UserId. The table lists 38 rows of sign-in data, such as 'mrunalpatkar32@gmail.com' signing in at 2024-07-22 22:08:11.

LogId (String)	SignInDate	SignInTime	UserEmail	UserId
44dcra04-aeab-4ce5...	2024-07-22	22:08:11	mrunalpatk...	34880428-60f1-70cd-bcf0-993af25544aa
f0c68be6-23a5-4aac...	2024-07-21	20:25:38	patkarmrun...	74c85458-e081-70cf-ba15-03a687abf036
e26f1c94-0050-428b...	2024-07-21	19:58:01	mrunalpatk...	34880428-60f1-70cd-bcf0-993af25544aa
b0568332-e4d7-413...	2024-07-22	00:34:59	patkarmrun...	249814f8-10a1-703b-dccc-e859196905f3
2097d7ed-f0aa-4652...	2024-07-21	23:34:11	test11@yo...	94662418-8091-70df-d34d-c76ba998ccbc
3b5ad45a-fb22-4d44...	2024-07-21	19:52:21	mrunalpatk...	34880428-60f1-70cd-bcf0-993af25544aa
fd7d7ff1-57e5-4b35...	2024-07-22	00:27:00	patkarmrun...	b408a468-9081-702d-795d-c3190d5e5e12
f2492613-0dd4-4eda...	2024-07-21	20:06:53	mrunalpatk...	54083418-00e1-70ac-2e75-c1440a5c3754
1e95a905-1265-4b21...	2024-07-21	20:01:14	mrunalpatk...	b4e8e4b8-c041-7069-d490-cd4361e40bcf
d7b3793b-29fd-450d...	2024-07-21	16:30:47	mrunalpatk...	f498a418-4081-70a7-d0c0-672ba7ce95b
d10a6d6f-a348-4f5a...	2024-07-21	20:12:14	mrunalpatk...	b4e8e4b8-c041-7069-d490-cd4361e40bcf
91156dc4-afb1-40cb...	2024-07-21	20:10:21	mrunalpatk...	34880428-60f1-70cd-bcf0-993af25544aa
5f022c3f-f2e4-479e-b...	2024-07-21	16:06:29	mrunalpatk...	f498a418-4081-70a7-d0c0-672ba7ce95b
3a6f769c-4d59-4c98...	2024-07-21	16:29:31	mrunalpatk...	54a84448-7051-70ab-e716-97a7fbbe84694
3c158976-3566-44a9...	2024-07-22	18:54:47	mrunalpatk...	34880428-60f1-70cd-bcf0-993af25544aa

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

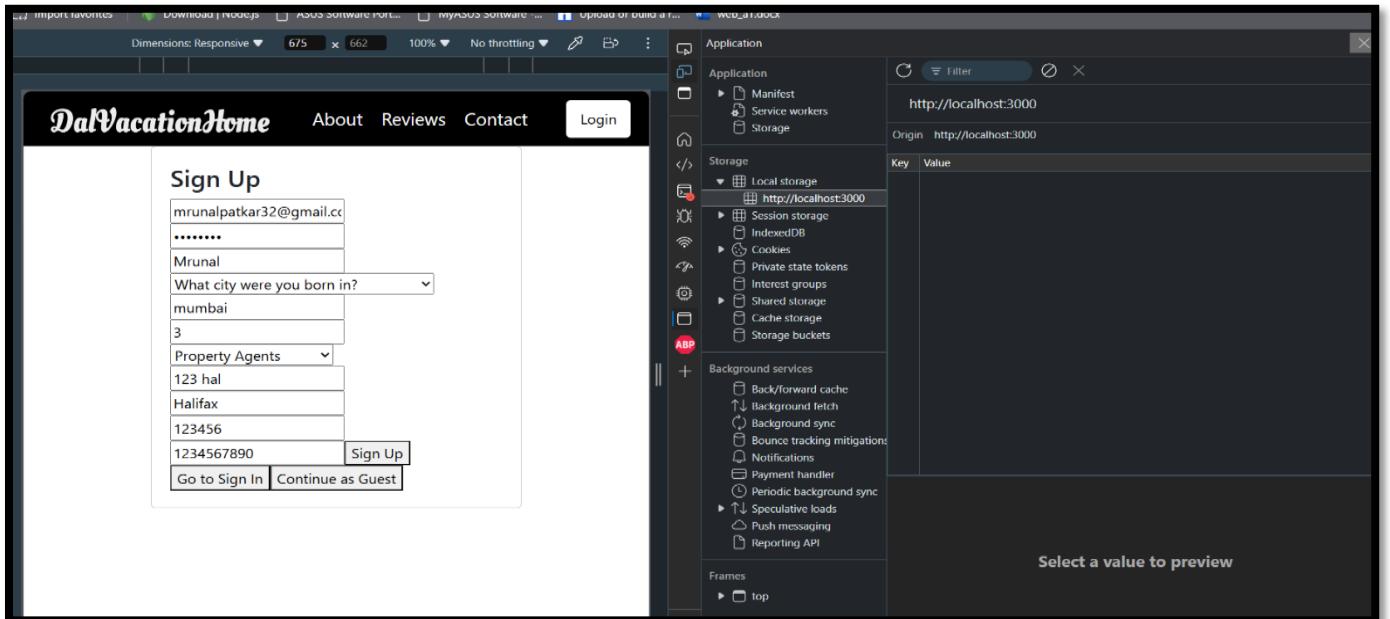


Figure 80: The user signs up as a property agent

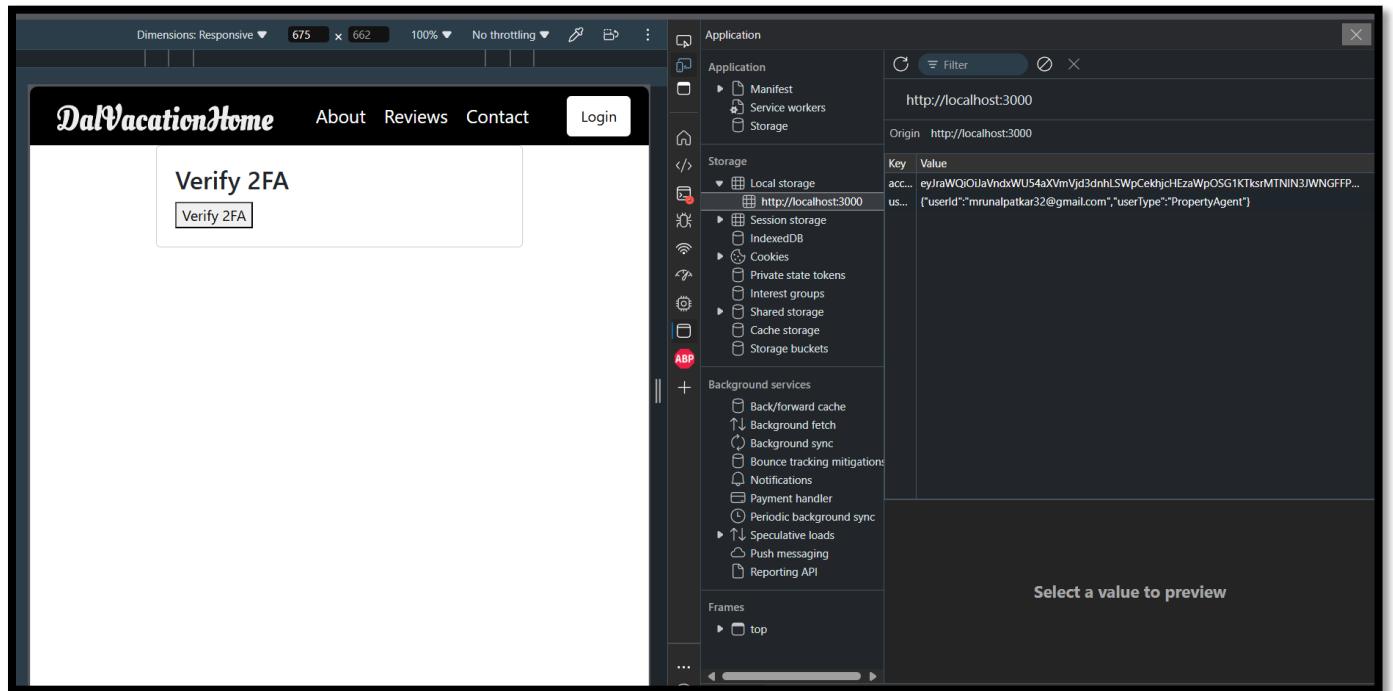


Figure 81: Successful sign-up

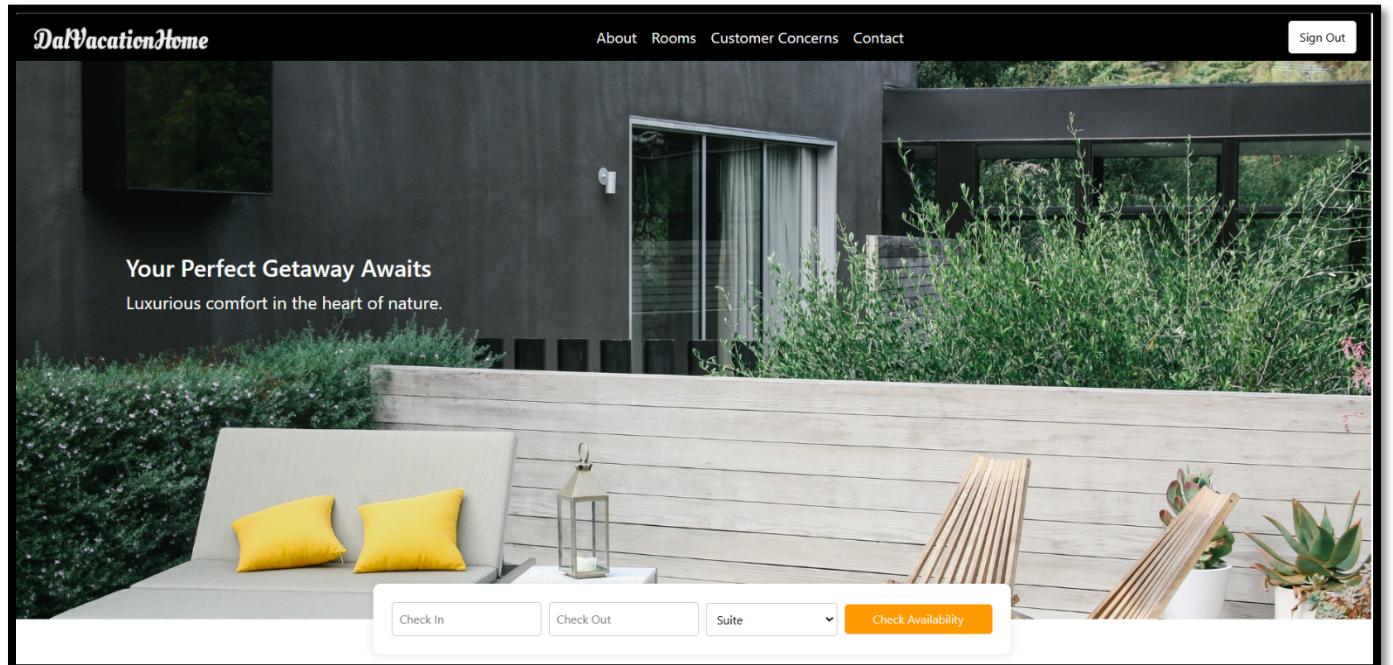


Figure 82: Property agent dashboard.

Fetching booking details based using a booking code:

Executing function: succeeded ([logs](#))

▼ Details

The area below shows the last 4 KB of the execution log.

```
{ "bookingStartDate": "04/07/2024", "bookingEndDate": "13/07/2024", "totalBookingDays": 10, "status": "Booked", "roomType": "suit", "capacity": 2, "price": 500, "furnishedType": "Fully furnished", "name": "Munali", }
```

Summary

Code SHA-256	Execution time
HTkDnfitnjRWtEDtQFAi/4h5LRqP7g48e067MDXTYMA=	13 seconds ago (July 23, 2024 at 04:29 PM ADT)
Request ID	Function version
f700becd-072e-4da3-9bcb-7c540031c62e	\$LATEST
Init duration	Duration
270.42 ms	2186.26 ms
Billed duration	Resources configured
2187 ms	128 MB
Max memory used	
73 MB	

Log output

The section below shows the logging calls in your code. [Click here](#) to view the corresponding CloudWatch log group.

```
START RequestId: f700becd-072e-4da3-9bcb-7c540031c62e Version: $LATEST
END RequestId: f700becd-072e-4da3-9bcb-7c540031c62e
REPORT RequestId: f700becd-072e-4da3-9bcb-7c540031c62e Duration: 2186.26 ms Billed Duration: 2187 ms Memory Size: 128 MB Max Memory Used: 73 MB Init Duration: 270.42 ms
```

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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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%5FServeless%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

						https://dalu.sharepoint.com/:v/r/teams/CSCI5410_Serverless_Team6/Shared%20Documents/General/Meeting%20Recordings/CSI%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/:v/r/teams/CSCI5410_Serverless_Team6/Shared%20Documents/General/..../..:/v:/r/teams/CSCI5410_Serverless_Team6/Shared%20Documents/General/Meeting%20Recordings/CSI%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/CSI%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/CSI%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/CSI%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/:r/tteams/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/:r/tteams/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/:r/tteams/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

The screenshot shows the GitLab Issues page for the project 'CSCI5410-S24-SDP-6'. The left sidebar includes sections for Project, Issues (12), Merge requests (0), Manage, Plan, Code, Build, Secure, Deploy, Operate, Monitor, Analyze, and Settings. The main area displays 43 issues under the 'All' tab. The issues are categorized by module:

- Deployment on cloud-run**: #43 - created 1 day ago by Mrunal Mangesh Patkar (Sprint 3)
- Cloud formation for Api gateways**: #42 - created 1 day ago by Mrunal Mangesh Patkar (Sprint 3)
- cloud formation for dynamodb**: #41 - created 1 day ago by Mrunal Mangesh Patkar (Sprint 3) (Closed)
- Cloud formation for sns-sqs**: #40 - created 1 day ago by Mrunal Mangesh Patkar (Sprint 3) (Closed)
- Cloud formation for Cognito**: #39 - created 1 day ago by Mrunal Mangesh Patkar (Sprint 3) (Closed)
- Module 6 Deployment**: #38 - created 1 day ago by Mrunal Mangesh Patkar (Sprint 3) (In Progress)
- Intent-To get details on How to Register**: #37 - created 1 day ago by Rameez Parkar (Sprint 3) (To Release)
- Intent - To allow user to submit concern to property agent via pub/sub**: #36 - created 4 days ago by Rameez Parkar (Sprint 3) (To Release)
- Create cloud function to show updated question and answer to user**: #35 - created 5 days ago by Axata Darji (Sprint 3) (To Release)
- Create cloud function to update answer for particular question in Firestore**: #34 - created 1 week ago by Axata Darji (Sprint 3) (To Release)

At the top, there are filters for Open (12), Closed (31), and All (43). A search bar and a 'New issue' button are also present.

Figure 84: All issues

2. Closed Project issues

The screenshot shows the GitLab Issues page for the project 'CSCI5410-S24-SDP-6', specifically filtered to show only closed issues. The left sidebar is identical to Figure 84. The main area displays 31 closed issues across the same modules:

- cloud formation for dynamodb**: #41 - created 1 day ago by Mrunal Mangesh Patkar (Sprint 3) (Closed)
- Cloud formation for Cognito**: #39 - created 1 day ago by Mrunal Mangesh Patkar (Sprint 3) (Closed)
- Intent-To get details on How to Register**: #37 - created 1 day ago by Rameez Parkar (Sprint 3) (Closed)
- Intent - To allow user to submit concern to property agent via pub/sub**: #36 - created 4 days ago by Rameez Parkar (Sprint 3) (Closed)
- Create cloud function to show updated question and answer to user**: #35 - created 5 days ago by Axata Darji (Sprint 3) (Closed)
- Create cloud function to update answer for particular question in Firestore**: #34 - created 1 week ago by Axata Darji (Sprint 3) (Closed)
- Create cloud function to pass questions with null answers to respective AgentId from Firestore**: #33 - created 1 week ago by Axata Darji (Sprint 3) (Closed)
- Identify Property Agent ID and Room Id details from DynamoDB booking and rooms table**: #32 - created 1 week ago by Axata Darji (Sprint 3) (Closed)
- Subscribe to pub to fetch the reference code user id details**: #31 - created 1 week ago by Axata Darji (Sprint 3) (Closed)

At the top, the 'Closed' filter is selected, showing 31 issues. A search bar and a 'New issue' button are also present.

Figure 85: Closed Issues

3. Development board

The screenshot shows a Jira-like development board for project CSCI5410-S24-SDP-6. The left sidebar includes sections for Project, Pinned, Issues (12), Merge requests (0), Manage, Plan, Issues (12), Issue boards, Milestones, Wiki, Requirements, Code, Build, Secure, Deploy, Operate, Monitor, and Analyze. The main area has three columns: 'InProgress' (Module 6 Deployment, issue #38), 'ToRelease' (Module 4- Notifications, issue #20; Module 5: Data Analysis and Visualization, issue #24), and 'Closed' (Module 3 - Message Passing, issue #10; Feat-User dashboard with property list, issue #13; Module 2 - Virtual Assistant Module, issue #7; Feat - Property Agents can add/update room details, issue #27; Creation of Lambda for fetching booking details using BookingID, issue #19; Module 1 - User Management & Authentication, issue #2; Boilerplate code for Frontend - React, issue #1). A search bar at the top right allows filtering by labels and grouping.

Figure 86: Development board

4. Team Board:

The screenshot shows a team board for the same project. The left sidebar is identical to Figure 86. The main area displays three individual boards: 'Axata Darji @adarji' (Module 4- Notifications, ToRelease, issue #20), 'Mrunal Mangesh Patkar ...' (Module 6 Deployment, InProgress, issue #38), and 'Nikita Davies @ndavies' (Module 4- Notifications, ToRelease, issue #20; Module 6 Deployment, InProgress, issue #38). The interface is similar to Figure 86, with a search bar and various filter options at the top.

Figure 87: Individual boards at the end

The screenshot shows the Jira Issue Boards interface. On the left, there's a sidebar with project navigation and a search bar. The main area displays three separate boards:

- Nikita Davies (@ndavies)**: Contains two items: "Module 4- Notifications" (status: ToRelease) and "Module 6 Deployment" (status: InProgress).
- Rameez Parkar (@parkar)**: Contains one item: "Module 3 - Message Passing" (status: ToRelease).
- Closed**: A list of completed issues:
 - Module 3 - Message Passing (#10)
 - Feat-User dashboard with property list (#13)
 - Module 2 - Virtual Assistant Module (#7) (due May 18)
 - Feat - Property Agents can add/update room details (#27)
 - Creation of Lambda for fetching booking details using BookingID (#19)
 - Module 1 - User Management & Authentication (#2) (due May 11)
 - Boilerplate code for Frontend - React (#1) (due Jun 1)

Figure 88: Individual boards with the closed issues list

5. Sprint board:

The screenshot shows the Jira Issue Boards interface with a "Sprint Board" selected. The main area displays three sprints:

- Sprint 2**: Contains two items: "Module 4- Notifications" (status: ToRelease) and "Module 5: Data Analysis and Visualization" (status: ToRelease).
- Sprint 3**: Contains one item: "Module 6 Deployment" (status: InProgress).
- Closed**: A list of completed issues (same as in Figure 88).

Figure 89: Sprint board at the end

Screenshot of team chats

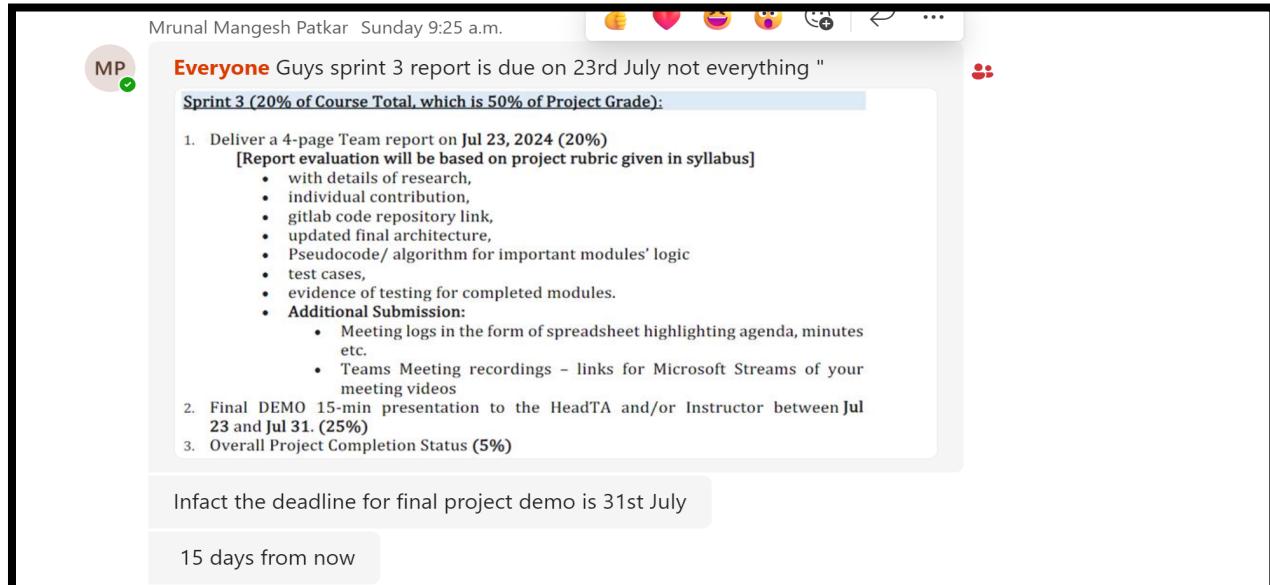


Figure 90: Screenshot of teams group chat

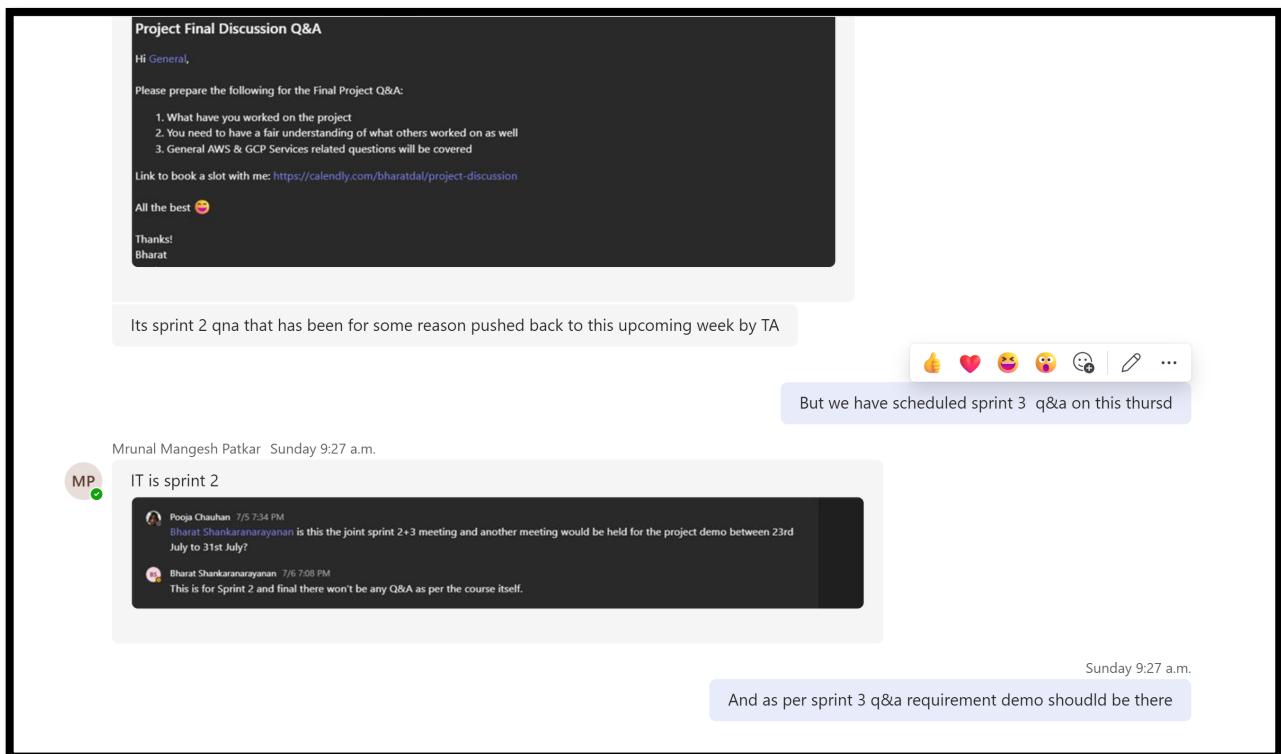


Figure 91: Screenshot of teams group chat

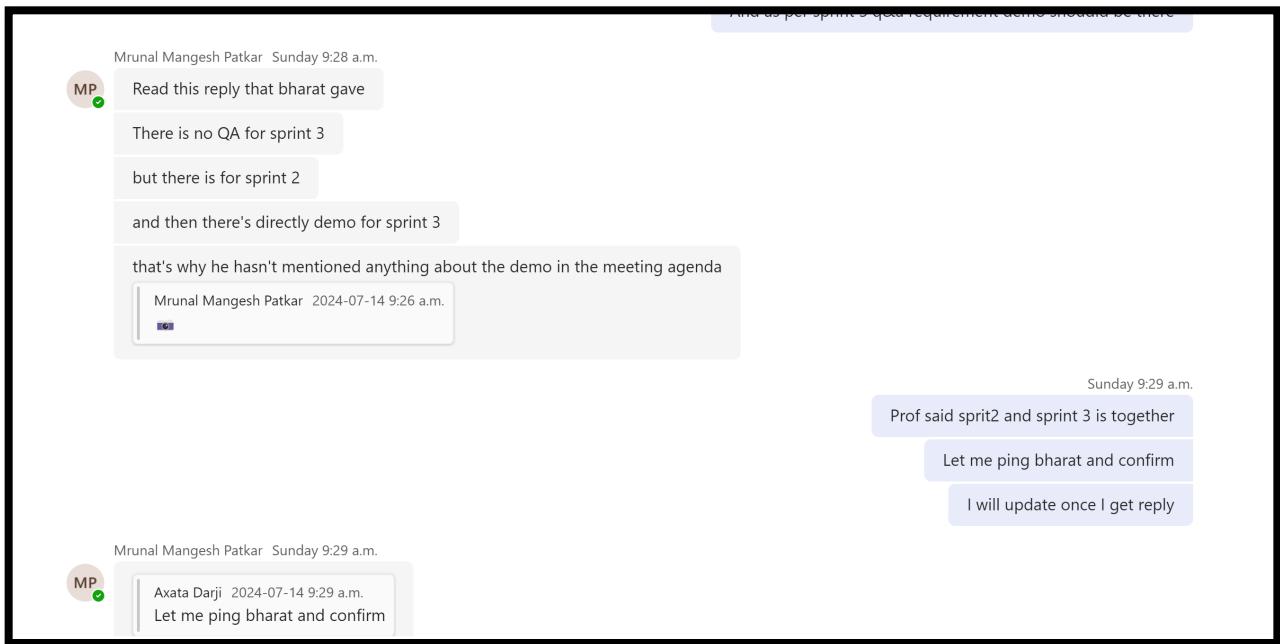


Figure 92: Screenshot of teams group chat

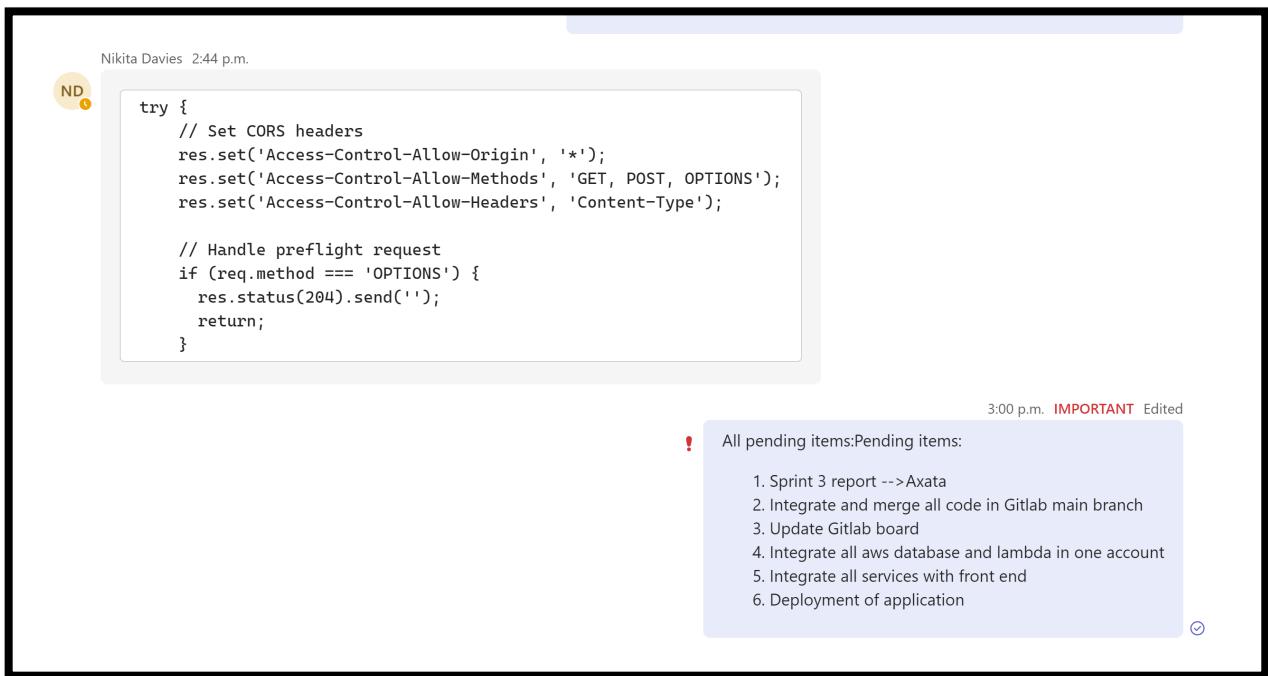


Figure 93: Screenshot of teams group chat

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].