

CSCI 5410

Serverless Data Processing

DALVacationHome

Group 6: Sprint-2 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>

Table of Contents

Details of Research.....	3
System Architecture.....	7
Pseudocode/Algorithm for important modules	12
Testcases and Evidence of testing for completed modules.....	19
Individual Contribution	34
Meeting Logs - Sprint 2.....	35
Screenshot of team chats.....	36
References.....	39

Details of Research

For notifying the user:

Amazon SNS works closely with Amazon Simple Queue Service (Amazon SQS). These services provide different benefits for developers. Amazon SNS allows applications to send time-critical messages to multiple subscribers through a “push” mechanism, eliminating the need to periodically check or “poll” for updates. Amazon SQS is a message queue service used by distributed applications to exchange messages through a polling model and can be used to decouple sending and receiving components—without requiring each component to be concurrently available. Using Amazon SNS and Amazon SQS together, messages can be delivered to applications that require immediate notification of an event, and also persisted in an Amazon SQS queue for other applications to process at a later time.[1]

In a nutshell:

- SQS is like a shared mailbox where messages are stored and retrieved by different parties.
- SNS is like a loudspeaker that broadcasts messages to a group of people simultaneously.

A subscription filter policy allows you to specify property names and assign a list of values to each property name. When Amazon SNS evaluates message attributes or message body properties against the subscription filter policy, it ignores the ones that aren't specified in the policy.

A subscription accepts a message under the following conditions:

- When the filter policy scope is set to MessageAttributes, each property name in the filter policy matches a message attribute name. For each matching property name in the filter policy, at least one property value matches the message attribute value.
- When the filter policy scope is set to MessageBody, each property name in the filter policy matches a message body property name. For each matching property name in the filter policy, at least one property value matches the message body property value.
- We can apply a filter policy to an Amazon SNS subscription using the Amazon SNS console. Or, to apply policies programmatically, we can use the Amazon SNS API, the AWS Command Line Interface (AWS CLI), or any AWS SDK that supports Amazon SNS.

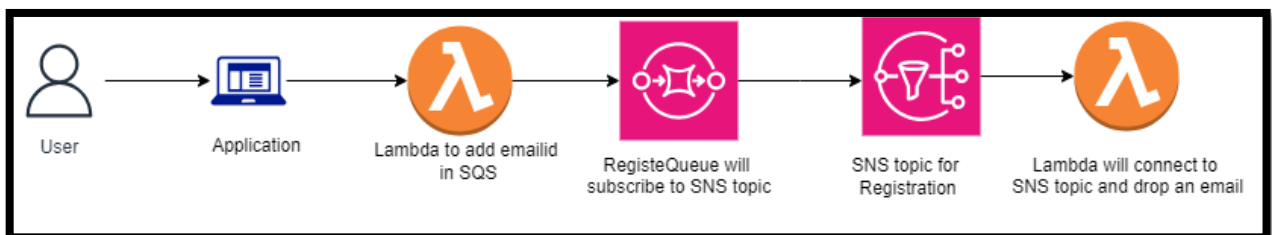


Figure 1: SNS and SQS Email Notification for registration

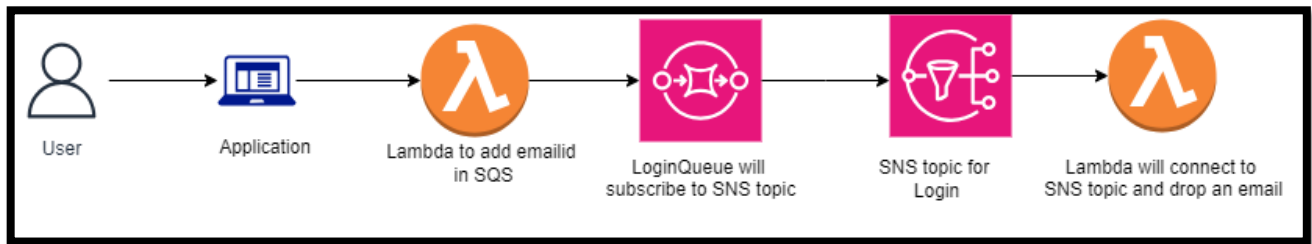


Figure 2: SNS and SQS Email Notification for login

Dialogflow

Dialogflow is a natural language understanding platform that makes it easy to design and integrate a conversational user interface into your mobile app, web application, device, bot, interactive voice response system, and so on. Dialogflow can analyze multiple types of input from your customers, including text or audio inputs (like from a phone or voice recording). It can also respond to your customers in a couple of ways, either through text or with synthetic speech.

Dialogflow CX and ES provide virtual agent services for chatbots and contact centers. Agent Assist provides real-time suggestions for human agents while they are in conversations with end-user customers. The Agent Assist API is implemented as an extension of the Dialogflow ES API. Even though Agent Assist is an extension of the Dialogflow ES API, we can use a Dialogflow CX agent type as the virtual agent for Agent Assist. If you are only using a Dialogflow virtual agent, you can ignore these extensions.

Whether the users want to ask common questions or access specific information, text virtual agents offer an instant and satisfying experience for users who want quick and accurate responses.[2]

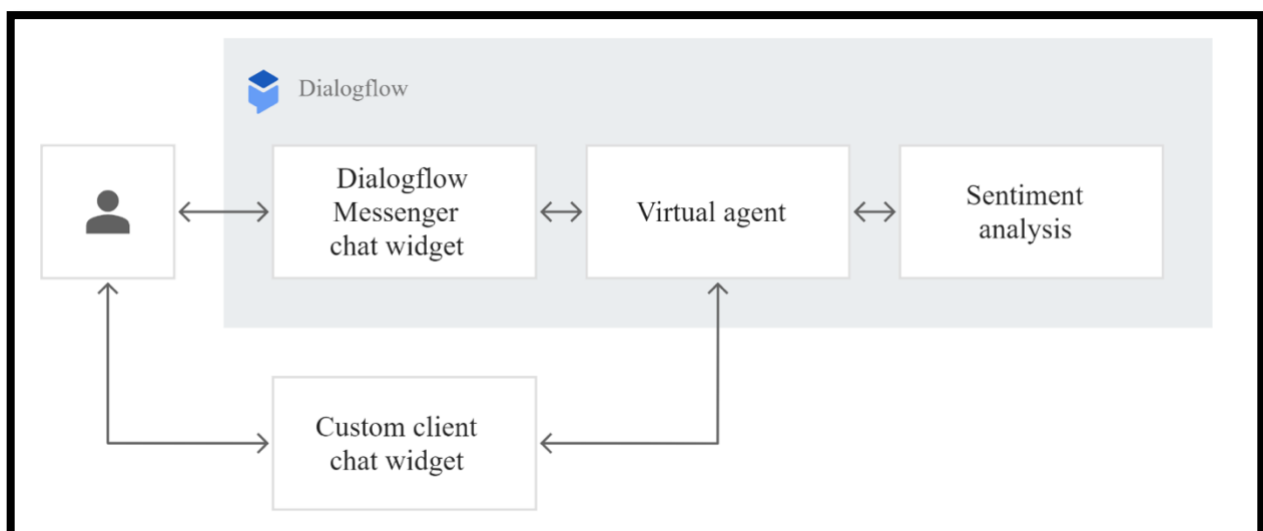


Figure 3: Dialogflow flowchart

Google Natural Language API

What is Google Natural Language API:

The Google Natural Language API is a powerful cloud-based service that provides natural language understanding technologies to developers. This API offers various features for analyzing and understanding text, making it an essential tool for applications that need to process and derive insights from large volumes of textual data.

The Google Natural Language API can be easily integrated into applications through RESTful API calls. It supports multiple languages and can process both plain text and HTML documents. Developers can leverage client libraries provided by Google for various programming languages, including Python, Java, Node.js, and more, to simplify integration and implementation.[3]

How to use Google Natural Language API in AWS Lambda Function:

In GCP, create a Service Account through the Google Cloud Console using the IAM & Admin section. Select Service Accounts and create a new service account. Create a JSON key for the service account. The Cloud Natural Language API also must be enabled in GCP. The downloaded JSON key must be stored in the AWS Secrets Manager. This key can be used in the Lambda function to retrieve the service account key from AWS Secrets Manager and configure it for the Google Cloud client library. This ensures secure and streamlined integration between AWS Lambda and Google Cloud services.

In our application, natural language API can be effectively used to analyse the feedbacks provided by users for various rooms and thus helps us to find the overall polarity of feedbacks to the customer.

Amazon Cognito service

What is Amazon Cognito:

Amazon Cognito is a powerful authentication service from AWS that simplifies the process of adding user sign-up and login functionality to web and mobile applications. It consists of two main components: user pools and identity pools. User pools act as user directories, managing authentication and user data, while identity pools provide users with access to other AWS services. This structure allows developers to implement robust authentication systems without the need to build complex infrastructure from scratch. Amazon Cognito also supports multi-factor authentication and encryption for secure user management. It integrates seamlessly with other AWS services and provides features such as social identity providers (e.g., Facebook, Google), user migration, and customization of the authentication process [4].

How to Use the Cognito Console:

The Cognito console serves as the central management interface for these authentication resources. Through this console, developers can configure user pools and identity pools, we can adjust security settings, and tailor the authentication process to our application's specific requirements. It offers tools for user management, performance analytics, and integration with other AWS services. The console streamlines the setup process for features such as multi-factor authentication and social identity provider integration, making it a valuable asset in the development workflow[5].

Amazon Cognito User Pools:

User pools, a key feature of Cognito, offer a versatile approach to user management. They support various sign-up and sign-in methods, ranging from traditional username and password combinations to authentication through social media platforms. User pools handle critical security functions like password policies and account recovery and can be customized with additional attributes and workflows. Upon successful authentication, user pools issue JSON Web Tokens, facilitating seamless integration with modern application architectures. This blend of flexibility, security, and ease of use makes Cognito an attractive solution for developers seeking to implement efficient authentication in their applications.

We can define custom user attributes, set password policies, and configure account recovery options. In our current implementation we have added a lot of custom attributes that we are going to store after successful authentication. User pools also support triggers that allow you to execute AWS Lambda functions at various stages of the user lifecycle, such as pre-sign-up, post-confirmation, and pre-token generation. In our implementation we are using a post confirmation lambda to assign the users from the user pool to two distinct user Groups (Registered Users and Property Agents) as stated in the project requirements document[6]. In a nutshell, Amazon Cognito provides a robust and scalable authentication solution for our application, ensuring secure user management and seamless integration with other AWS services.

Triggering lambda functions from API Gateway

We need to create an API Gateway API and setup a resource method before calling a Lambda function from it. This will include defining the endpoint and configuring the method (GET, POST) that will trigger Lambda. Once we have created the method, we then integrate this with the desired lambda function. To do this you need to provide the integration settings for specifying the Lambda function ARN. After adding integration now deploy the API to any of its stages so that it can be accessed from anywhere. If called through its endpoint, this deployment step exposes, preparing it for invocation by invoking another service which is backing it up.

When an API Gateway endpoint is invoked, API Gateway makes a request to a particular Lambda function as indicated in such an invocation process. The request contains path parameters, query parameters, headers and body among other things which are given as event data to Lambda function. Then it processes this event data and runs through its logic, before getting back into on API Gateway's response side. In order to respond differently based on some set criterion in case there is need for customizing this response later on. [7]

System Architecture

Completed Modules

User Management and Authentication Module

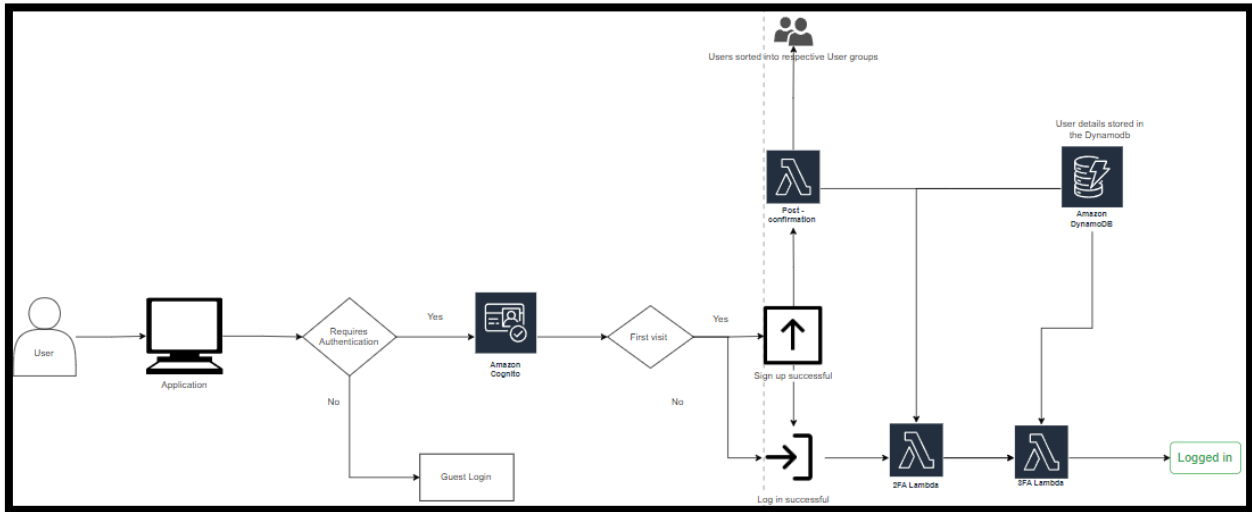


Figure 4: User Management and Authentication Module Architecture

The registered user or guest wanting to register to the app would first connect to Amazon Cognito. Amazon Cognito would perform the first step of the sign-up/sign-in process with username and password and on successful authentication, it would return a token. This token would be passed as an authorization header on any further calls made to lambda. We have used step functions to streamline the signup and sign-in steps to reach the appropriate lambdas for security questions and Caesar cipher. The lambdas would connect to DynamoDB to set and retrieve data. [8][9]

Notifications module:

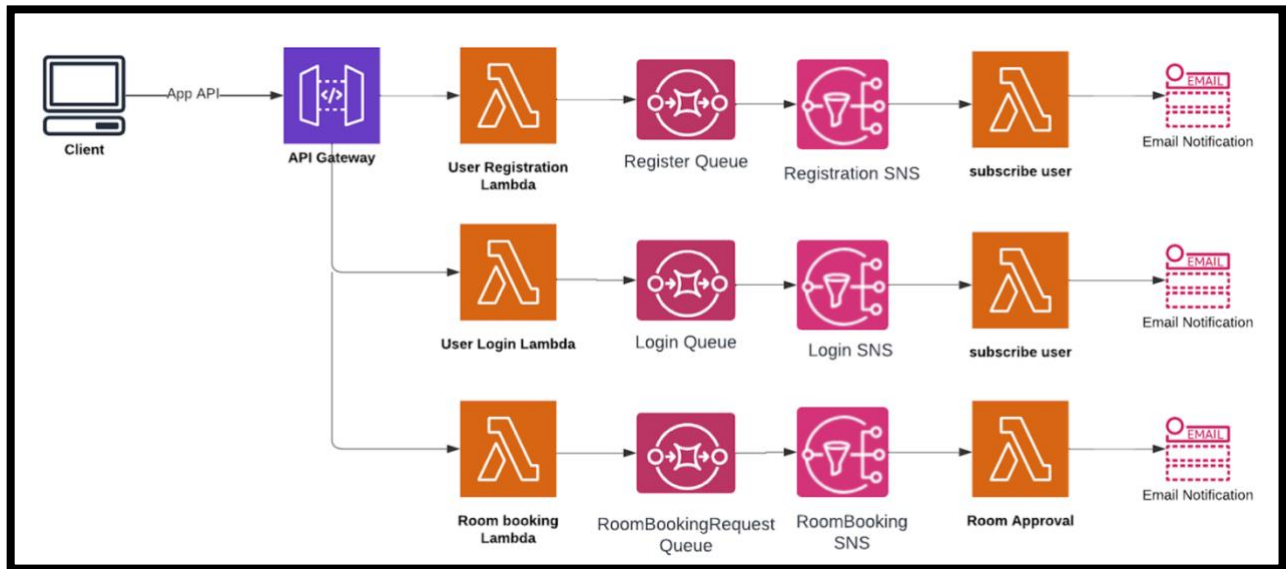


Figure 5: Notification Module Architecture

When users register or log in, the app would send email notifications via SNS . When the user successfully registers or logs in, a lambda function would be triggered which sends messages to SQS queue . The SQS queue is subscribed to the SNS topic and would trigger another Lambda functions to send success emails. For bookings, requests would go to a lambda function which sends messages to an SQS queue, triggering a Lambda function to process them. The second lambda function handles the logic to check if the selected dates are available for booking and accordingly sends confirmation/rejection email notifications through SNS and users would get an email about their booking status. This system would ensure users are promptly notified about their registration, login, and booking status [10][11].

Partially Completed

Data Analysis & Visualization:

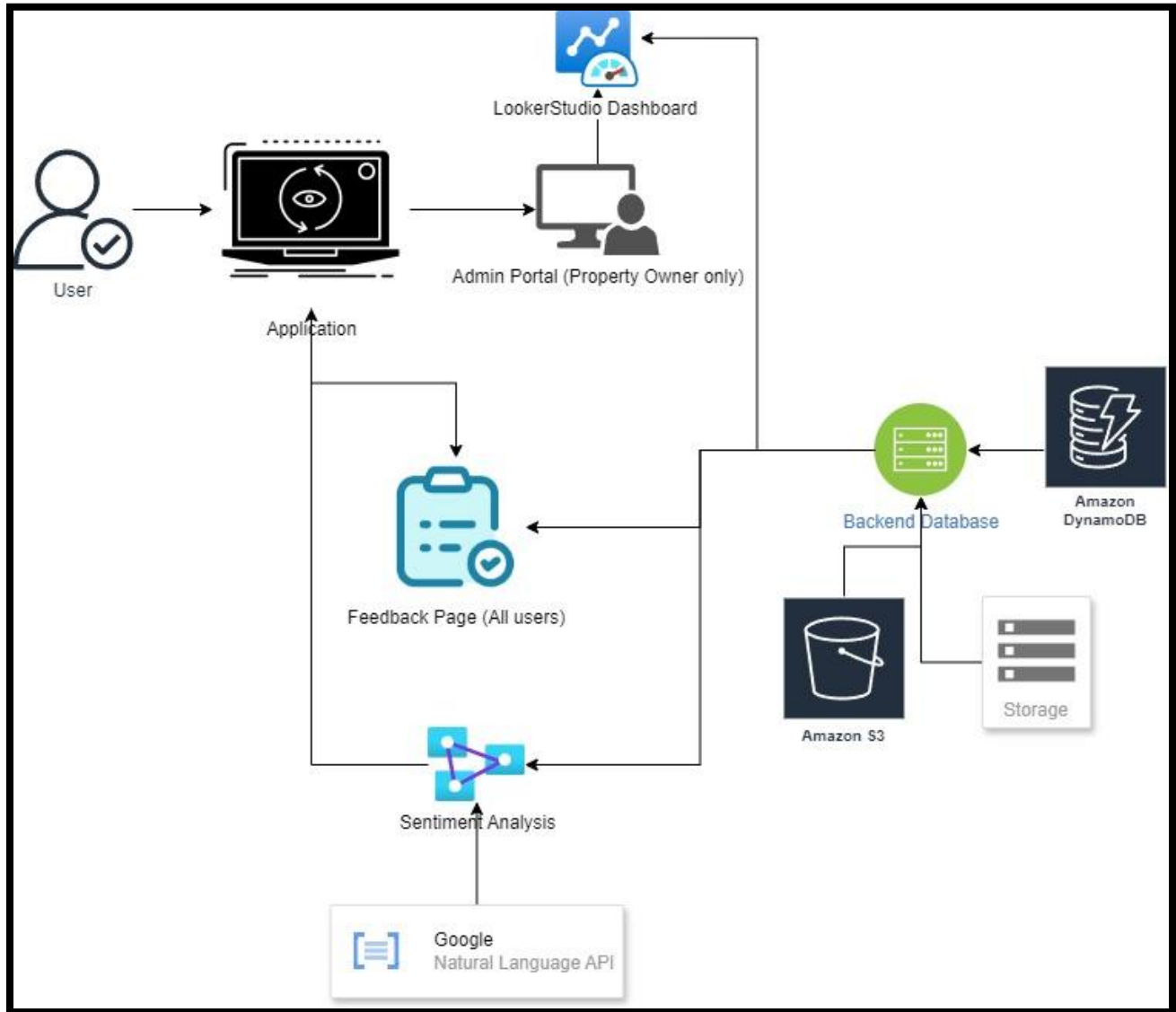


Figure 6: Data Analysis and Visualization Module Architecture

Property agents would be able to check user and login stats on an admin page with a LookerStudio dashboard. All users, including guests and customers, would be able to view customer feedback in a table format on the frontend, with data retrieved from DynamoDB, S3, or other storage services. Feedback would be analyzed using Google's Natural Language API to determine sentiment, and this analysis would be displayed for all users on the frontend.

In this module, the feedback analysis using Google Natural Language Processing part is completed. The Looker Studio section is still in progress [12].

Planned Sprint 3 modules

Virtual Assistant Module

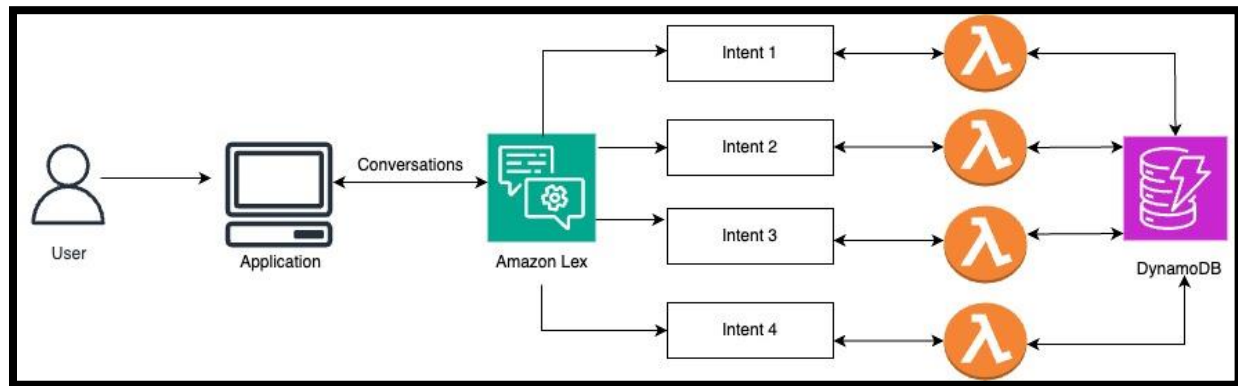


Figure 7: Virtual Assistant Module Architecture

We would set up various intents on Amazon Lex such as for reservation details, queries, etc. in order to give our chatbot the context of some common questions and how to answer them. We would create our own UI for the chatbot, and the application would interact with Amazon Lex. For some questions such as reservation details, Lex would need to fetch data from DynamoDB. To achieve this, Lex can connect to Lambda first, which can fetch data from DynamoDB, process it and return the necessary output to Lex. Lex can then return this response to the user as a response to their question.[13]

Message Passing Module

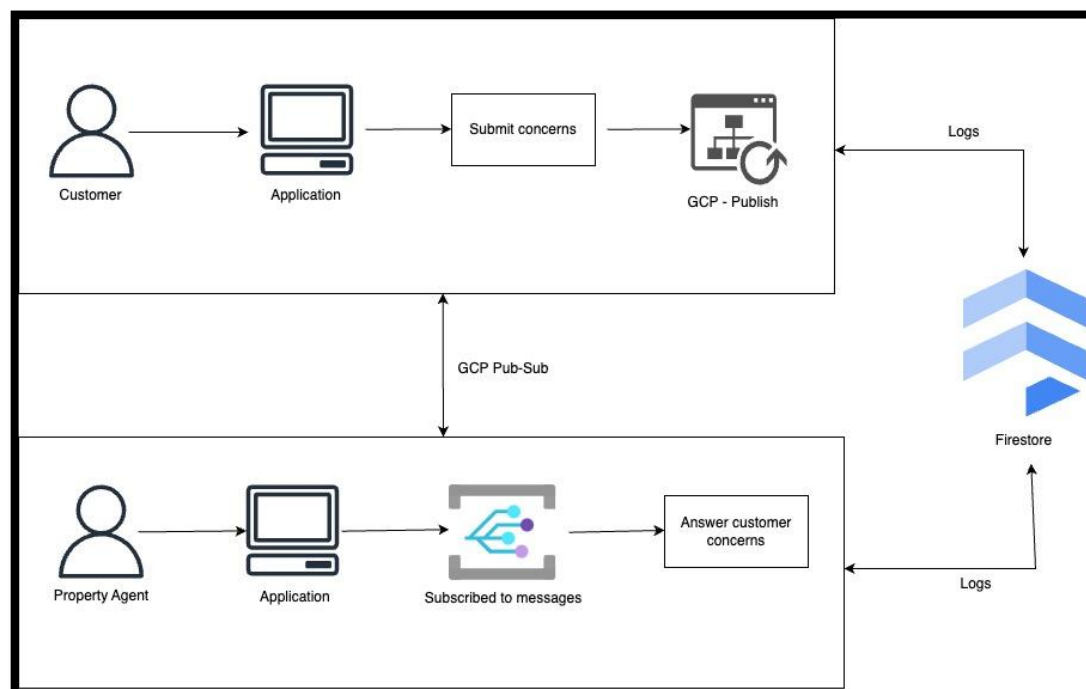


Figure 8: Message Passing Module Architecture

The registered customer would publish their concerns on the portal based on the booking code which would reach the property agent that has subscribed to it. The property agent would then answer the concern/query which would reach the customer. The communication/message passing is carried out through GCP Pub-Sub. The logs of these conversations are maintained on Firestore [14][15].

Web Application Building and Deployment:

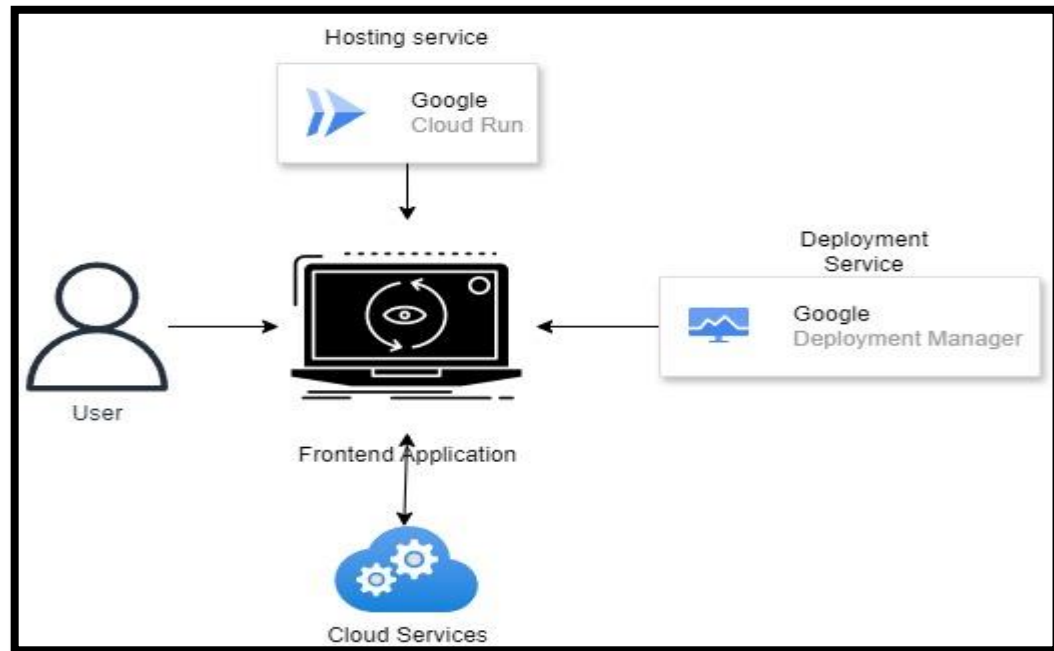


Figure 9: Frontend deployment using CloudRun

The entire application and user/client-facing interface would be hosted using GCP Cloud Run. Automation of service deployment would be handled using CloudFormation or GCP Cloud Deployment Manager [16][17].

Pseudocode/Algorithm for important modules

Algorithm for notifying upon successful login/register:

- **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.
- **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.
- **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.
- **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.
- **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 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 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": "Wonderful experience and great room",  
  "rating": 4,  
  "userId": "1"  
}
```

5. 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.

6. 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 AWS Cognito authentication (sign up and sign in):

1. User Registration Process:

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

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

2. User Sign-Up Verification

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

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

3. Post Confirmation Lambda Function

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

4. User Sign-In

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

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


Algorithm for Security Question and answer:

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

Algorithm for Caesar Cipher:

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

Algorithm for 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 fetching Booking details using a booking code:

1. API Call to Lambda Function:

- **API Call:**

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

- **Example Request Data:**

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

2. 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.

- **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.

- **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.

- **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.

Testcases and Evidence of Testing completed modules

Dashboard

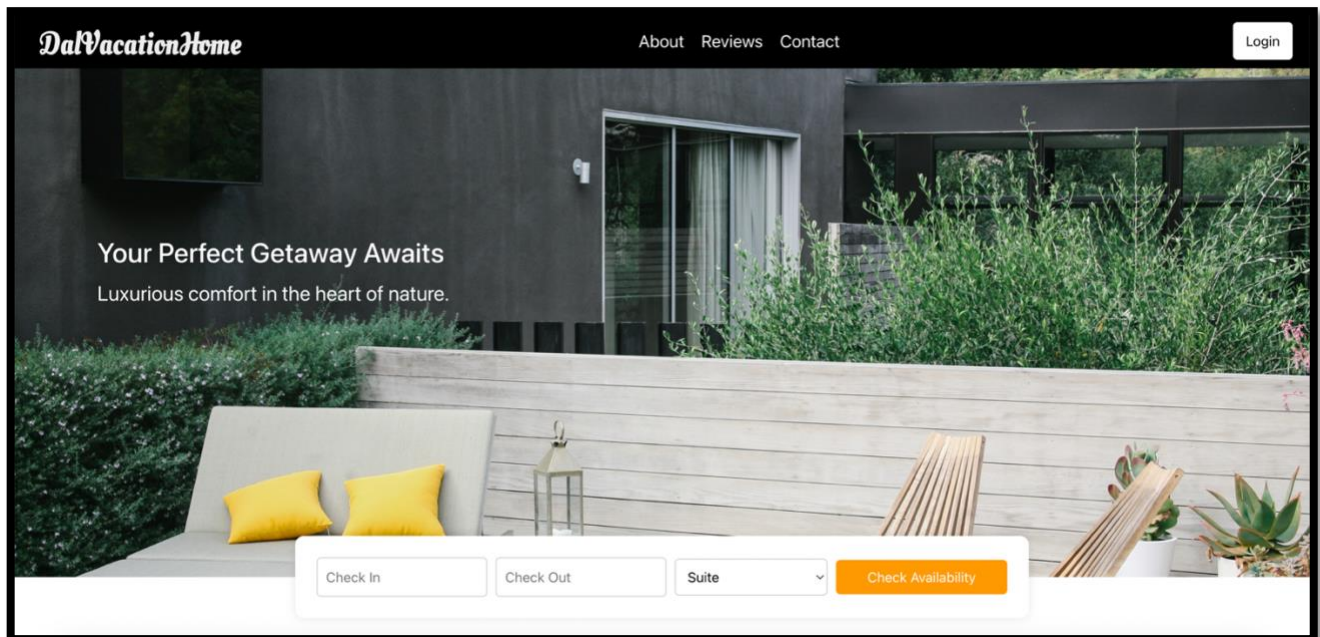


Figure 10: Dashboard for DALVacationHome Application

Rooms List and API integration

Testcase 1: Successful fetching of list of available rooms

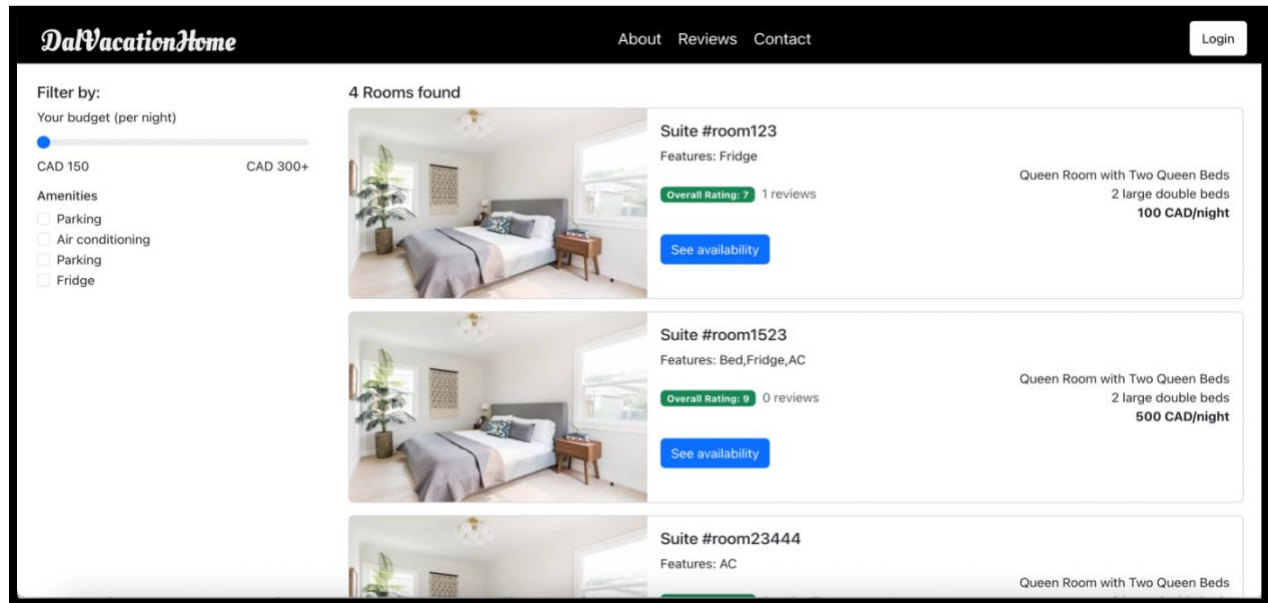


Figure 11: Available Rooms List UI

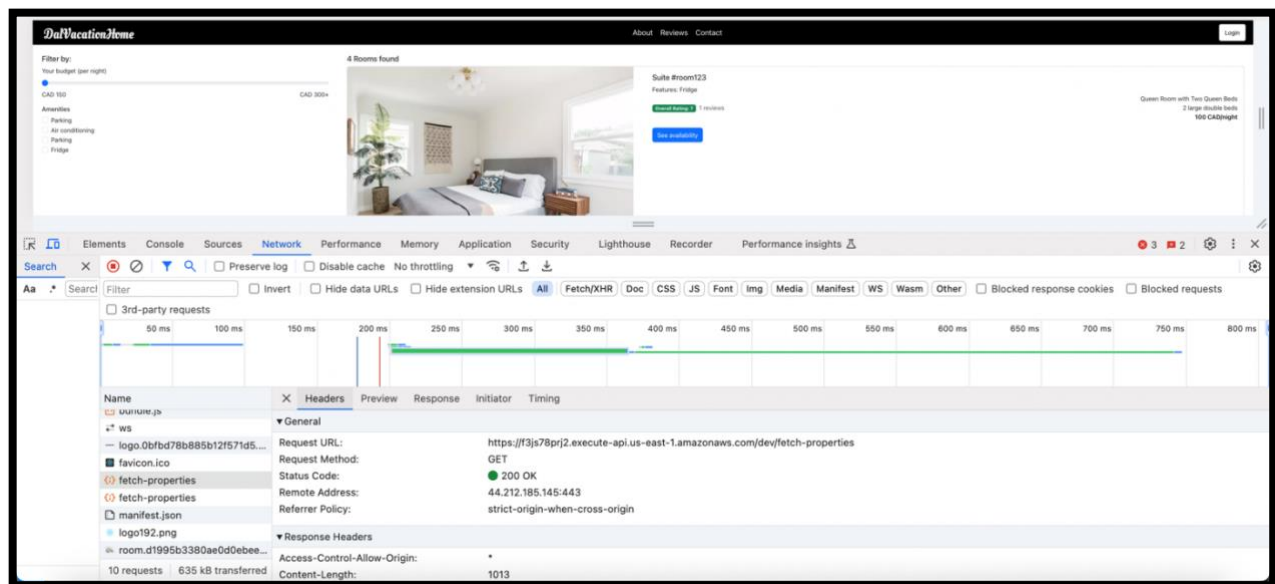


Figure 12: API integration of fetch properties

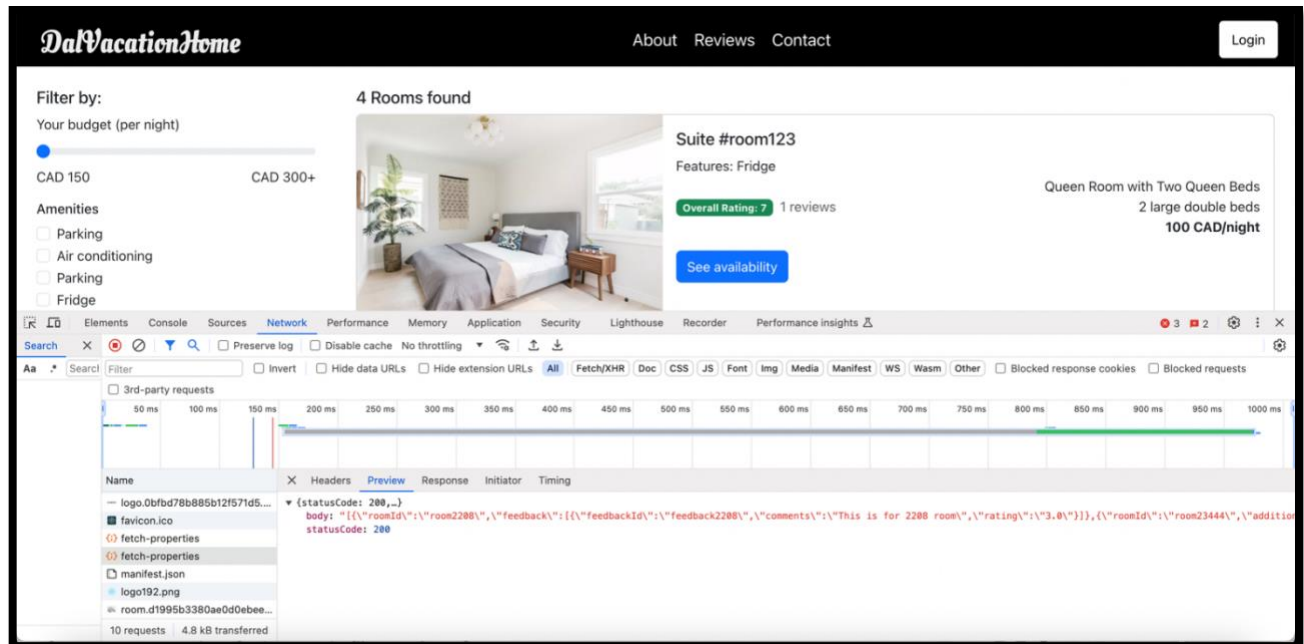


Figure 13: Response of fetchProperties API

Room Detail and API integration

Testcase 2: Successful fetching of room details

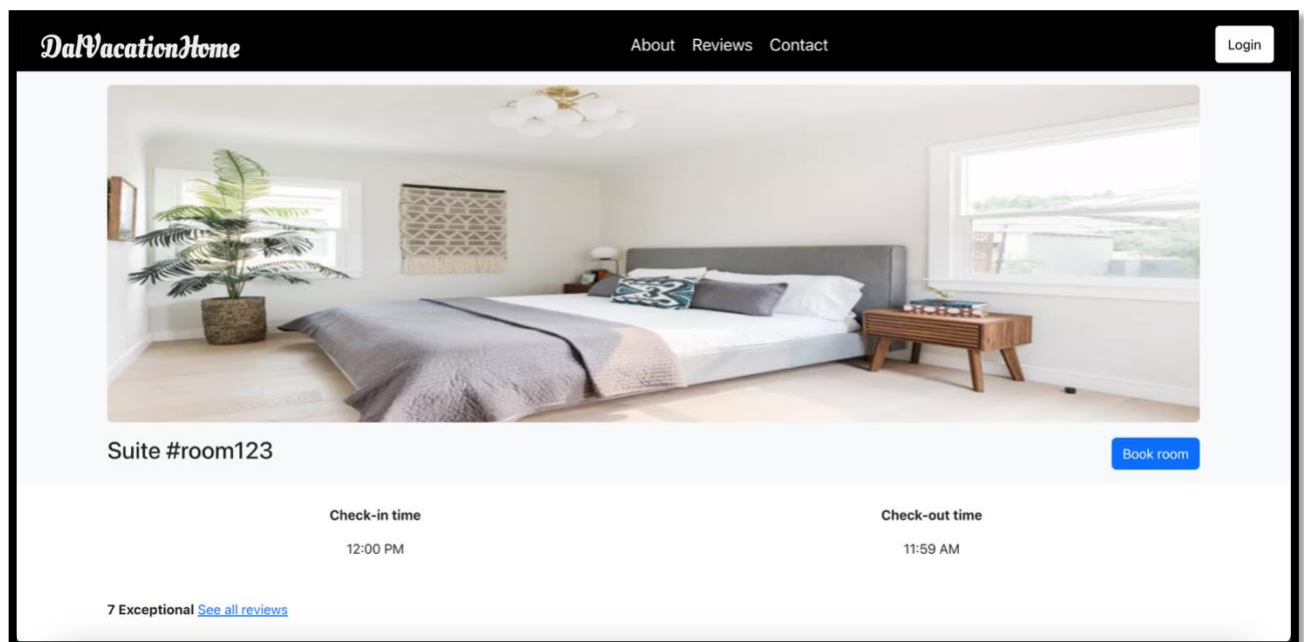


Figure 14: Room Details UI

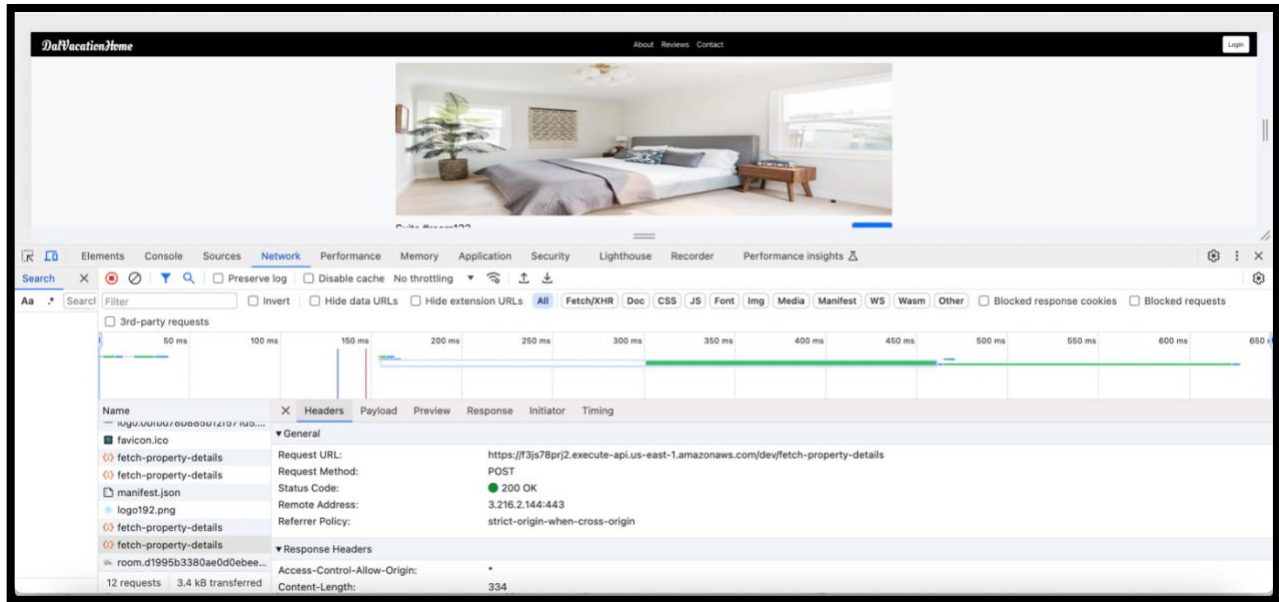


Figure 15: API integration of fetch properties details API

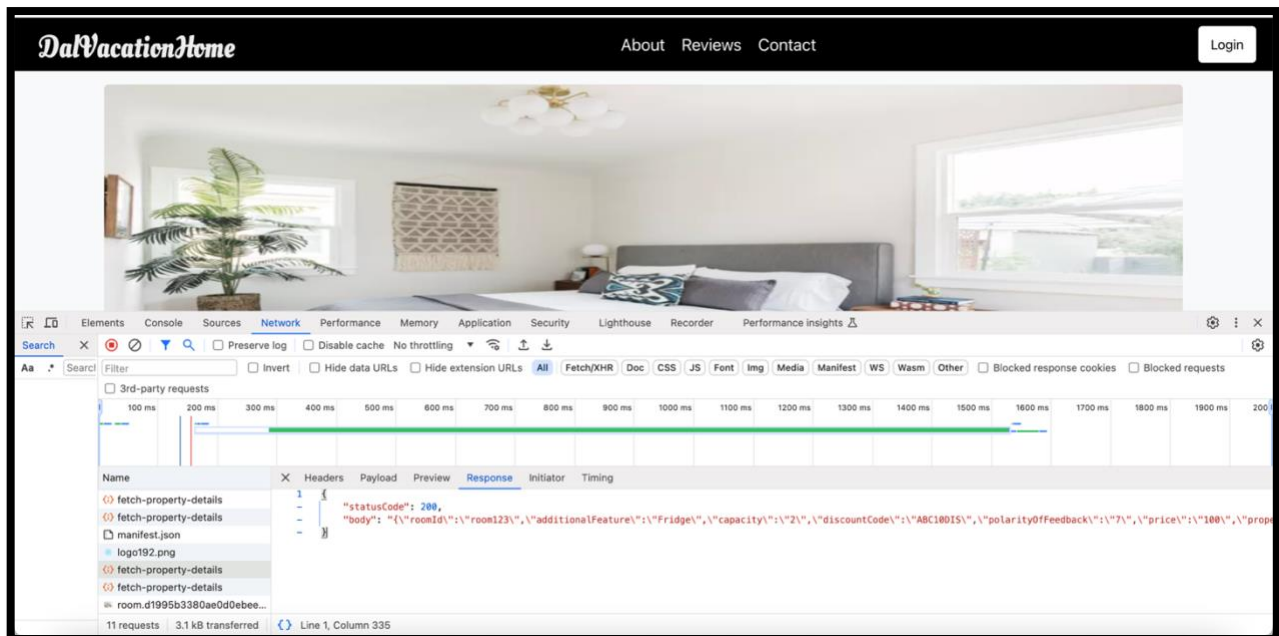


Figure 16: Response of fetch property details API

Module: Data Analysis

Feedbacks and API integration

Testcase 1: Successful fetching of feedbacks

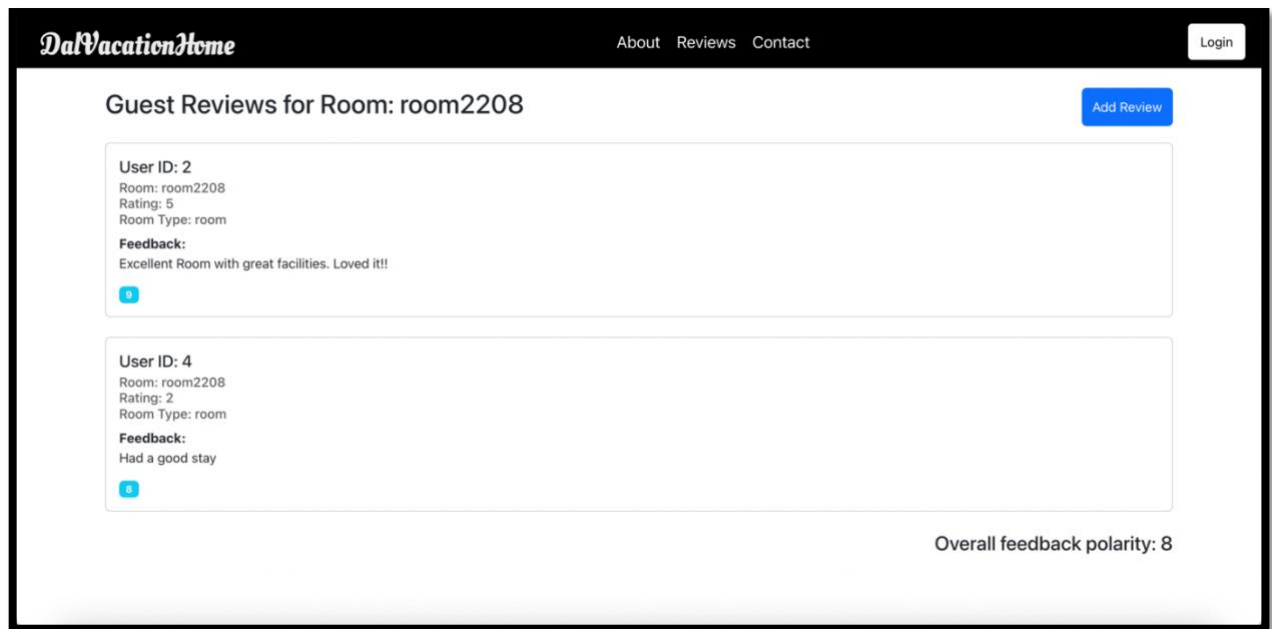


Figure 17: Guest Reviews UI

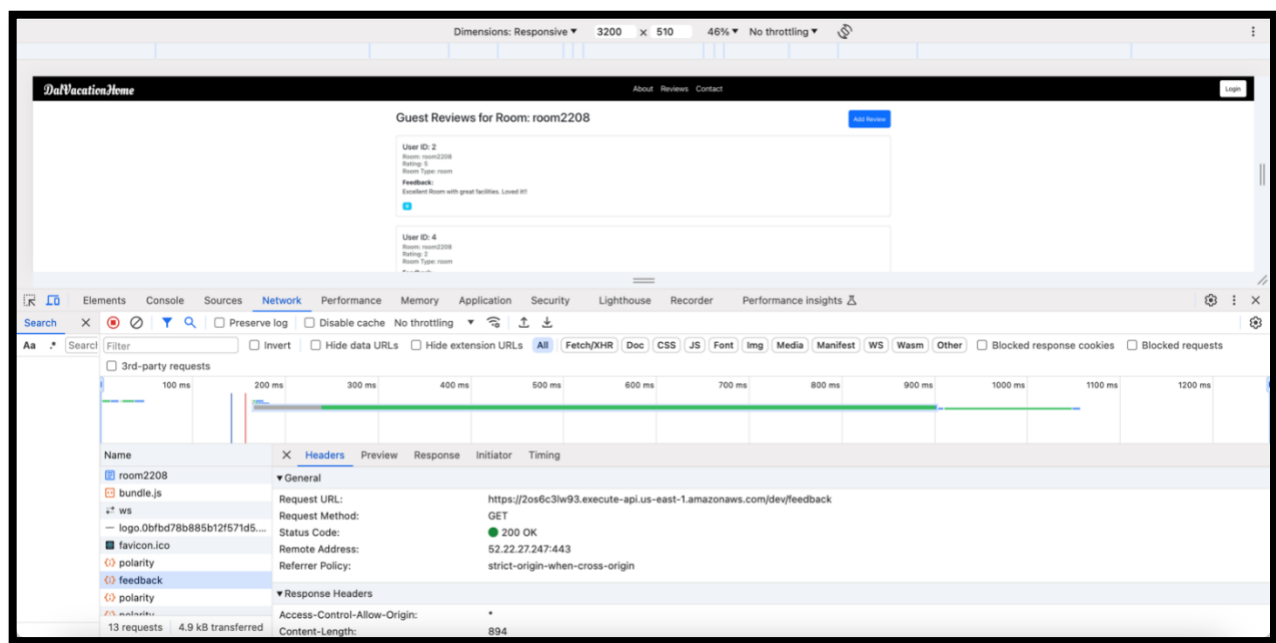


Figure 18: Feedback API Integration

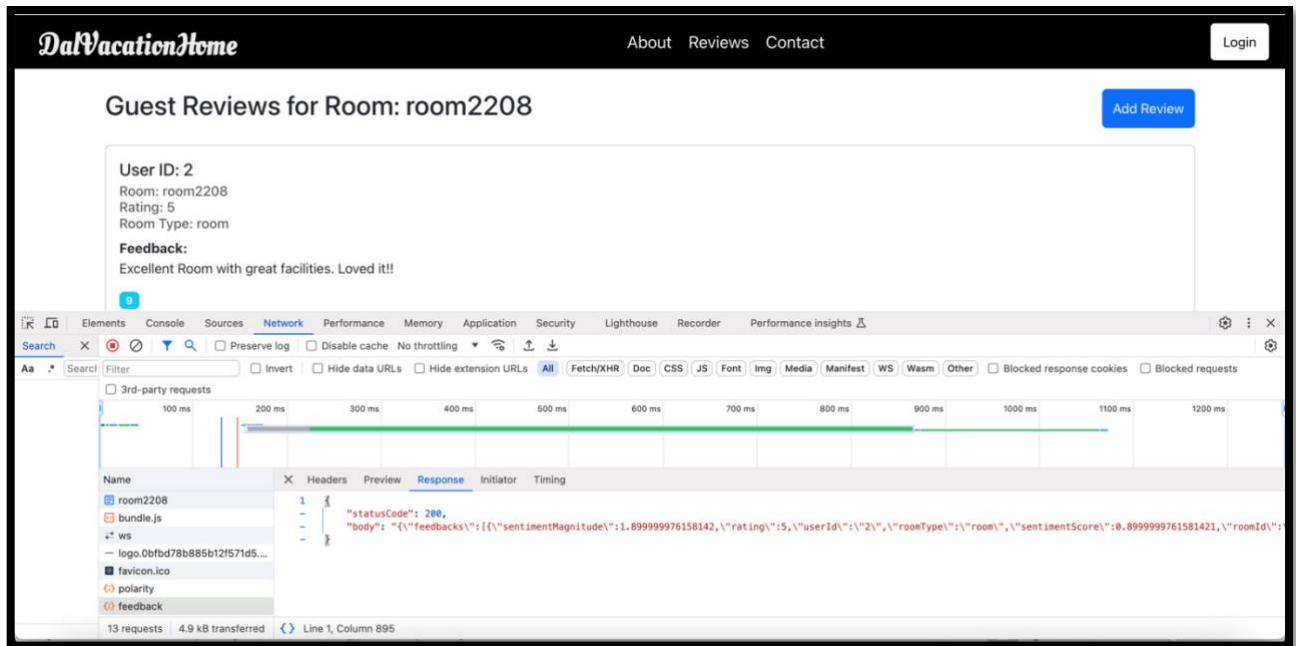


Figure 19: Response of feedback API

Testcase 2: Successful fetching of feedbacks overall polarity

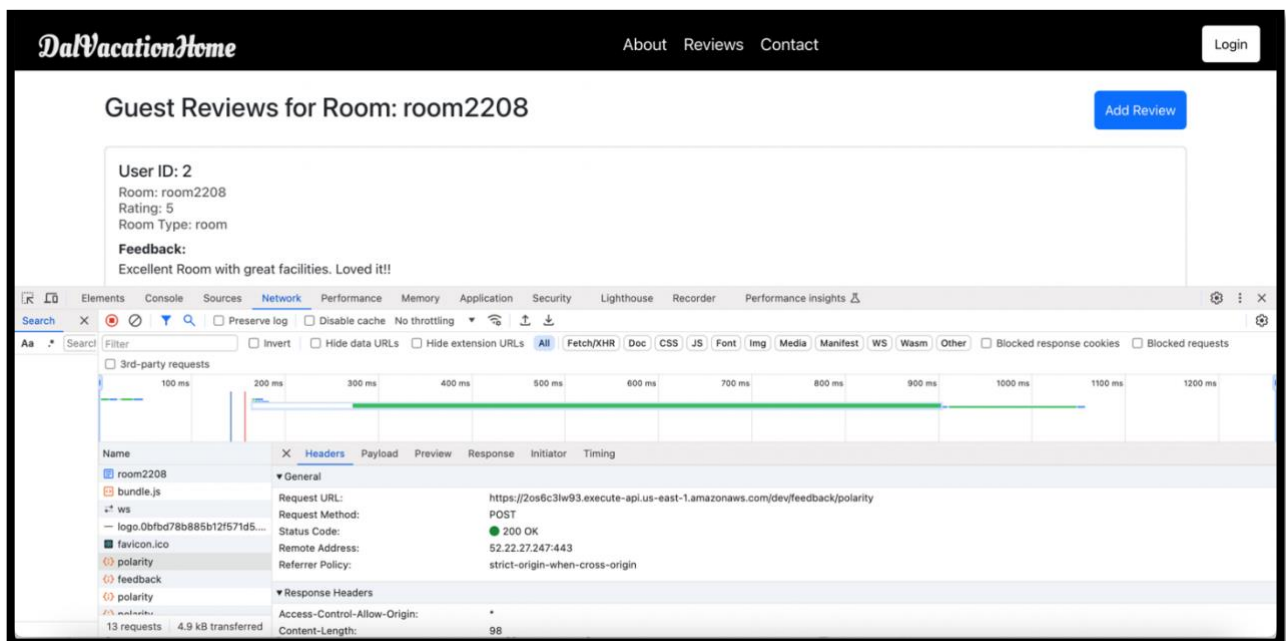


Figure 20: Feedback Polarity API Integration

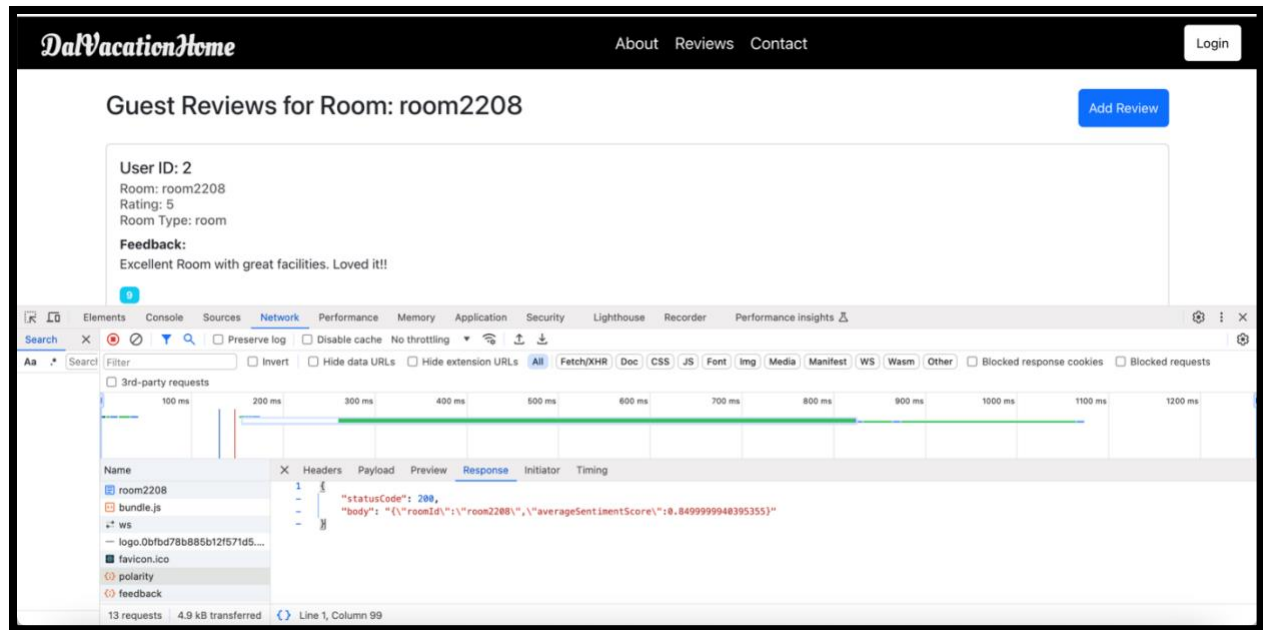


Figure 21: Response of getFeedbackPolarity API

Module: Notifications

Room Booking and API integration

Test case 1: Successful Booking of Room

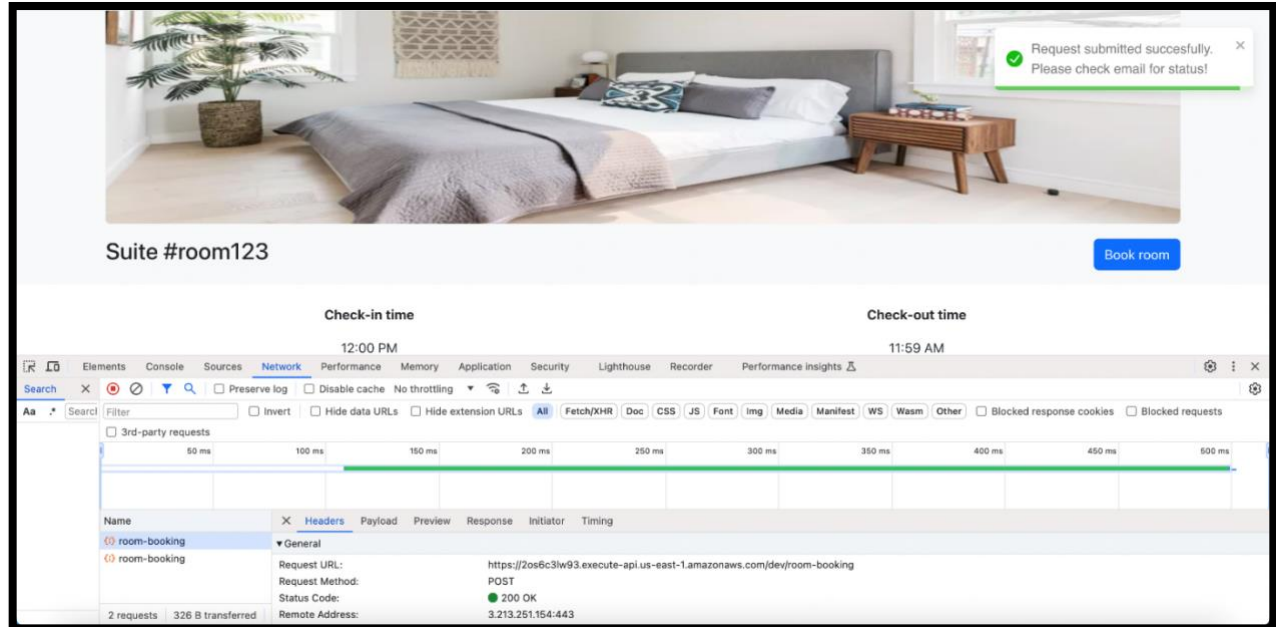


Figure 22: Room Booking API integration

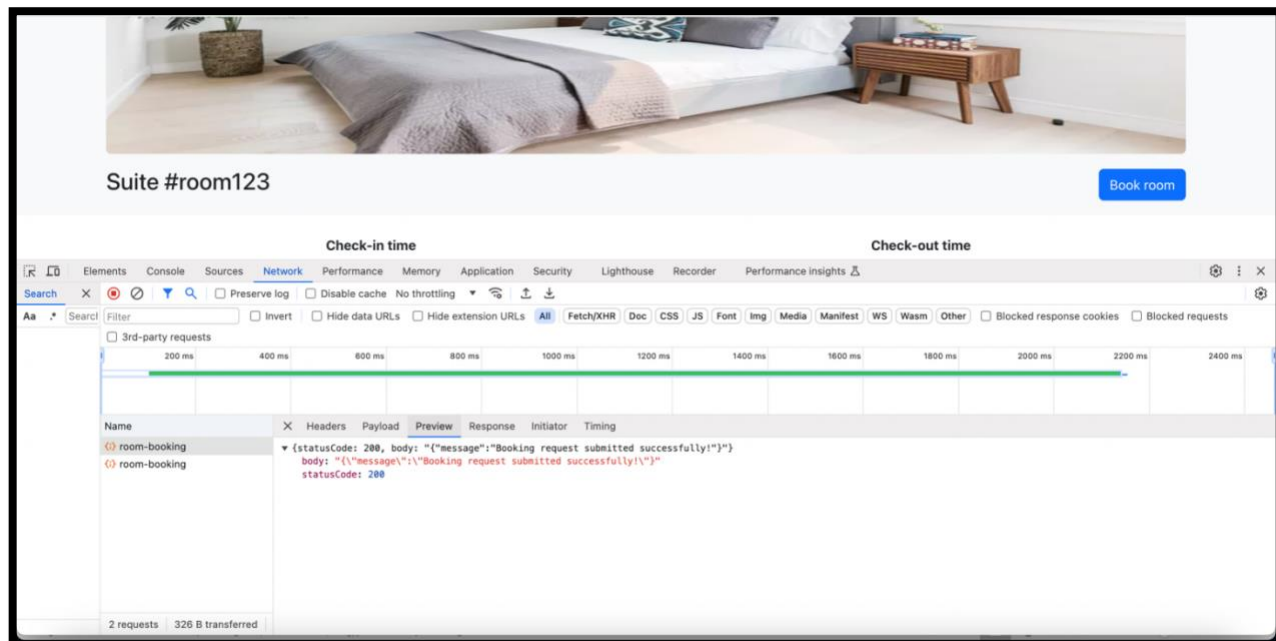


Figure 23: Room Booking API response

Test case 6: Booking Request Confirmation

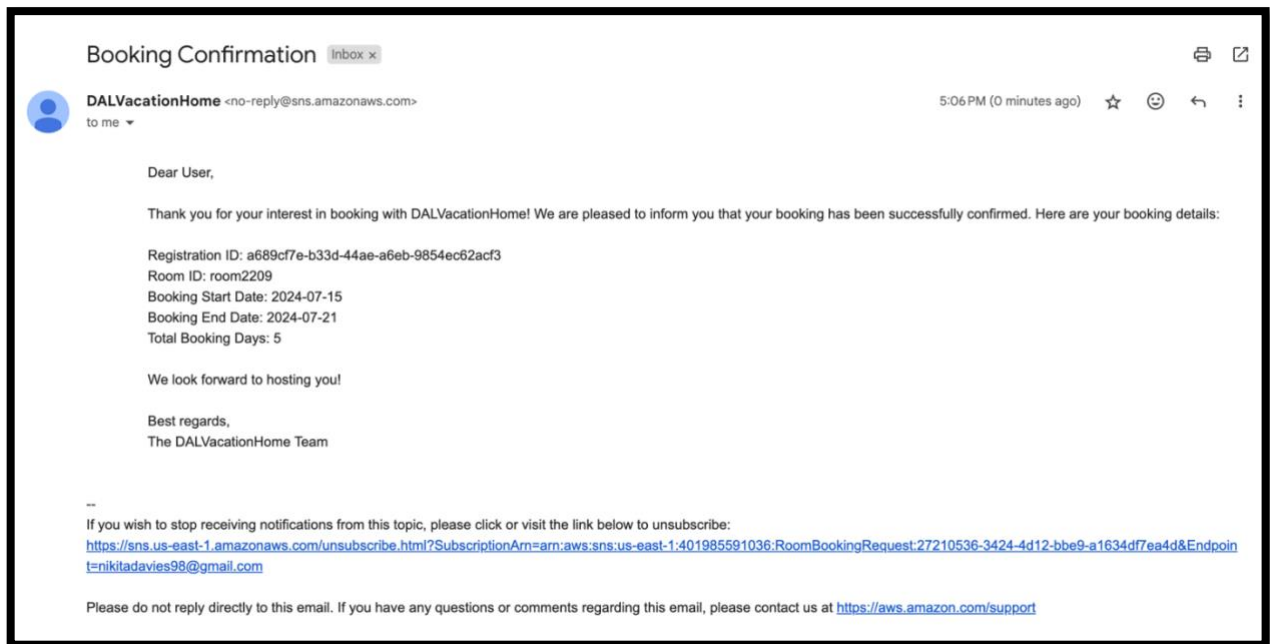


Figure 24: Email for booking Confirmation

Test case 7: Booking Request Rejection

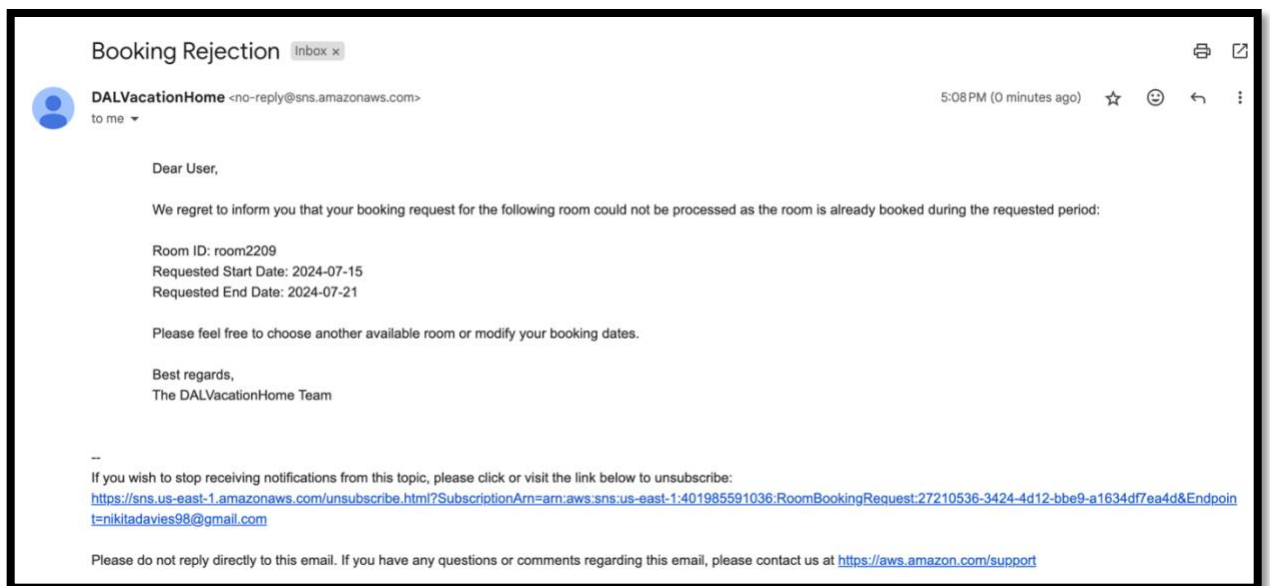


Figure 25: Email notification for Booking Rejection

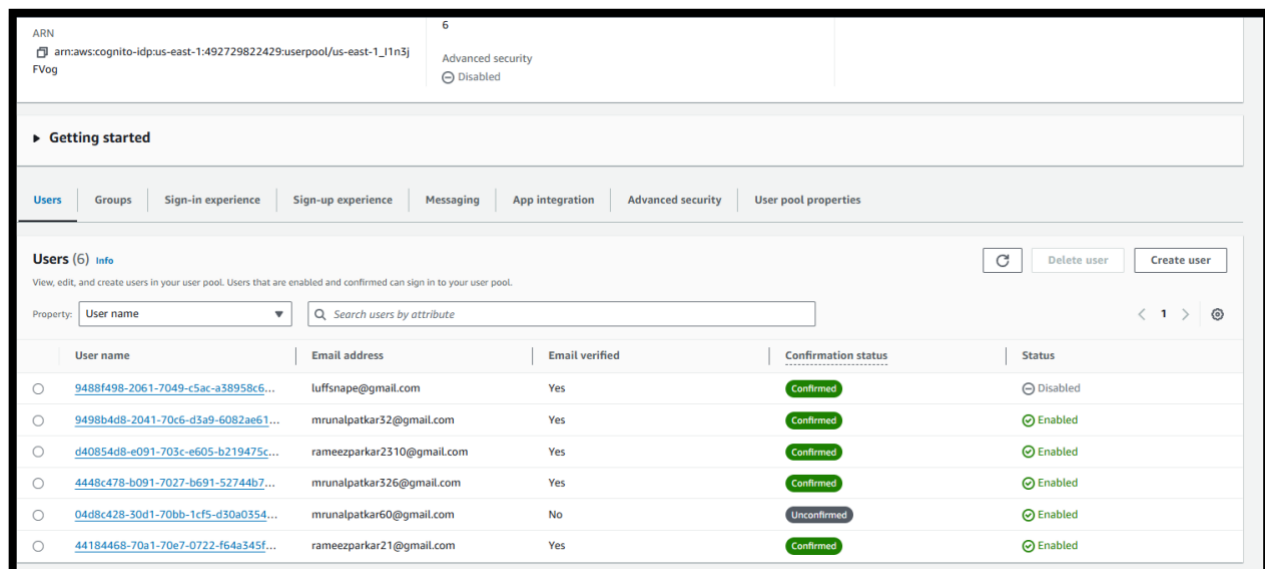
Authentication module test cases:

Test case 1:

User sign up data entry before email verification



Figure 26. User sign up form.



User name	Email address	Email verified	Confirmation status	Status
9488f498-2061-7049-c5ac-a38958c6...	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 27. New user is inserted in the Cognito User pool(unverified)

Test case 2:
User sign-up data entry after email verification

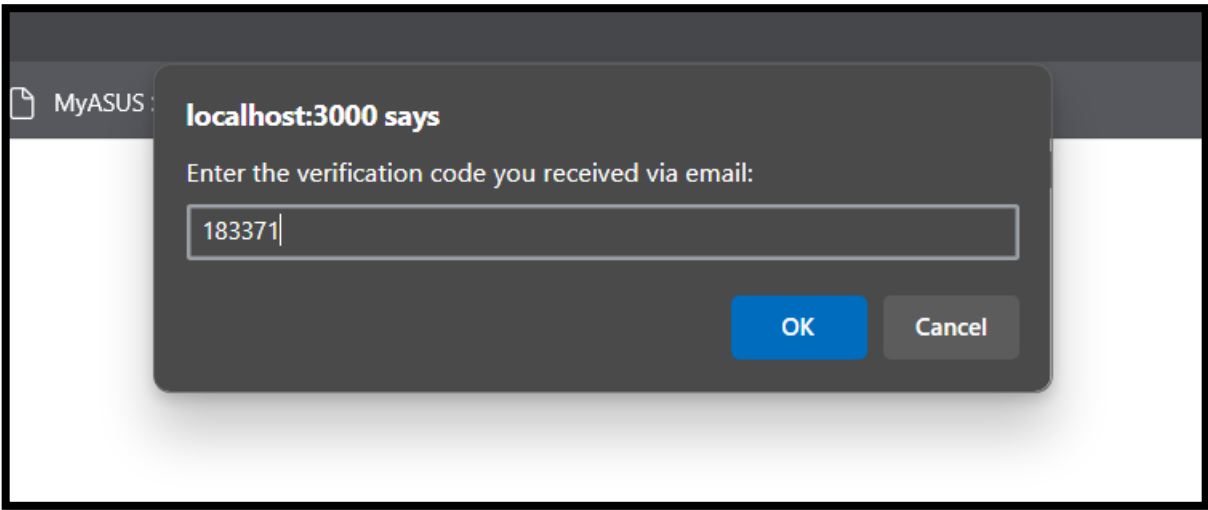


Figure 28. Verification code inserted.

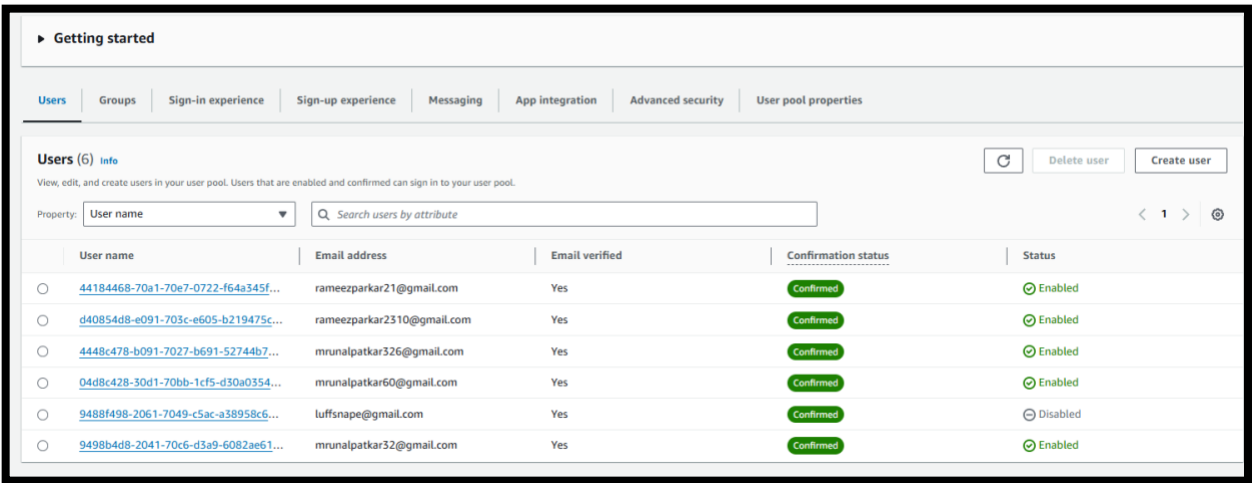


Figure 29. Email verified in cognito

Test case 4:

User Signed up successfully (Welcome mail triggered)

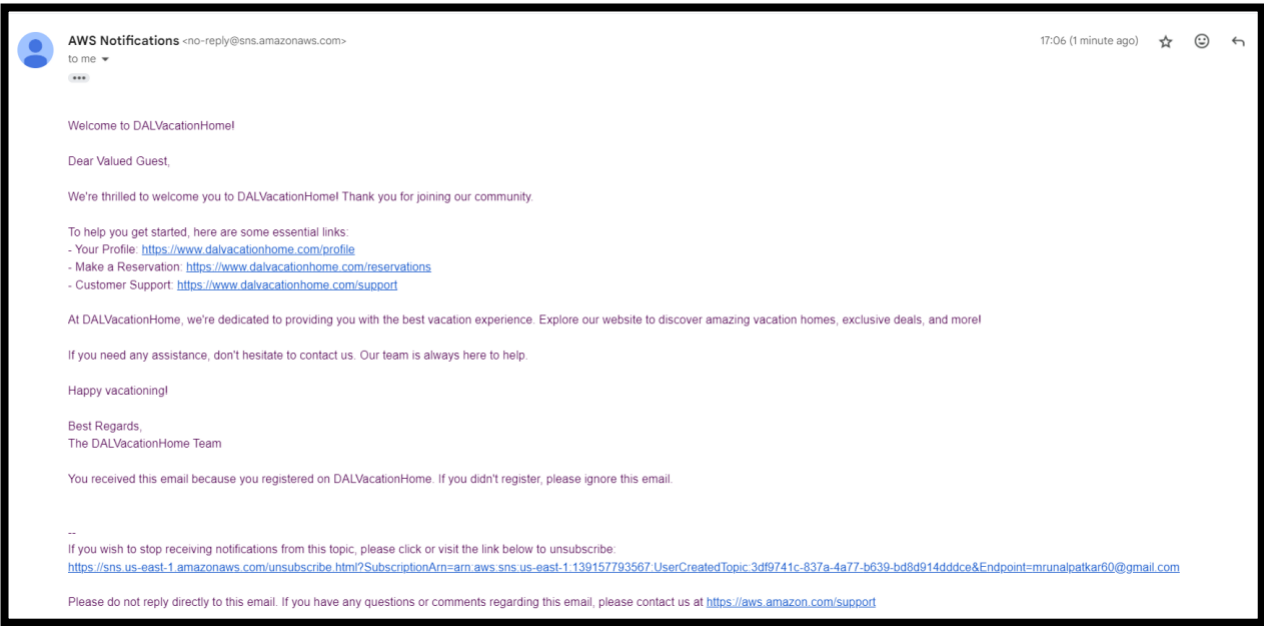


Figure 30. Welcome mail is sent after successful sign up

Test case 4:

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

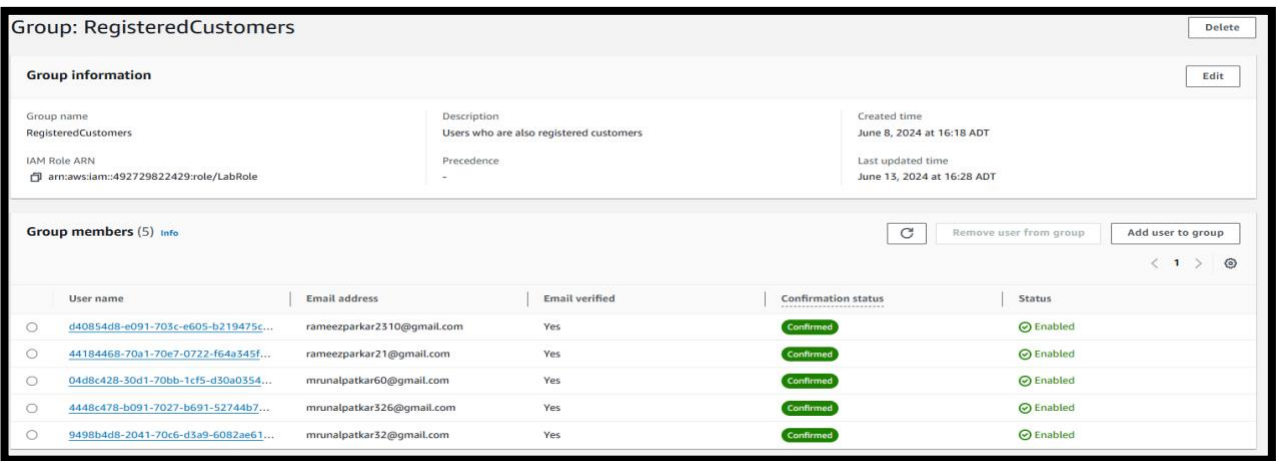


Figure 31. User sorted in respective groups

Items returned (4)

	UserID (String)	contact_no	email	name	pincode	security_answer	security_question	user_type
<input type="checkbox"/>	mrunalpatkar60@gm...	1234567890	mrunalpatk...	Mrunal	123456	Mumbai	q5	RegisteredC...

Figure 32. User data is entered in the dynamodb upon successful sign-up.

Test case 4:

2 factor authentication front end

Verify 2FA

Figure 33. 2FA Frontend component.

localhost:3000 says

Enter the answer to your security question:

Figure 34. 2FA security answer

```

X Headers Payload Preview Response Initiator Timing
1 {"status":200,"result":true}
  
```

Figure 35. 2FA success response

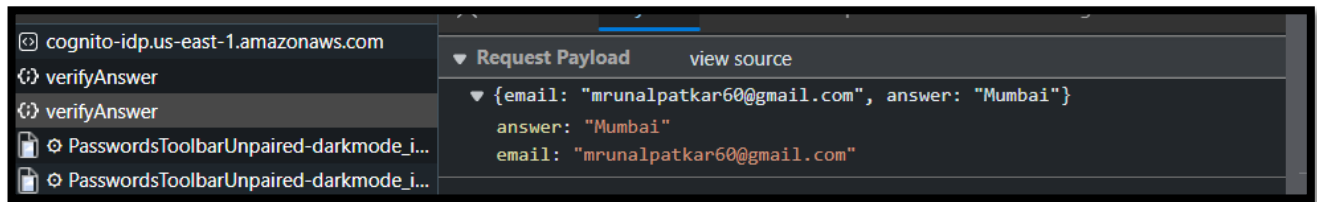


Figure 36. 2FA frontend payload.

Test case 5:

3 factor authentication front end:

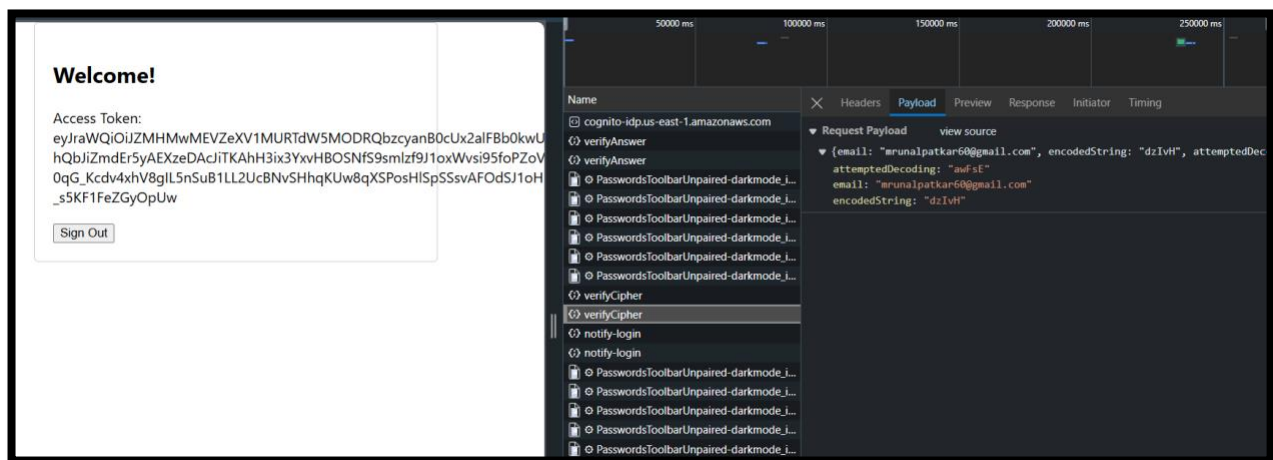


Figure 37. 3FA frontend payload

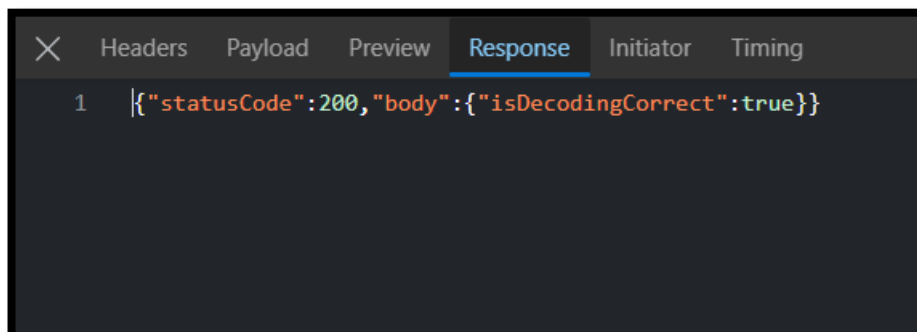


Figure 38. 3FA success response.

Test case 6:

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

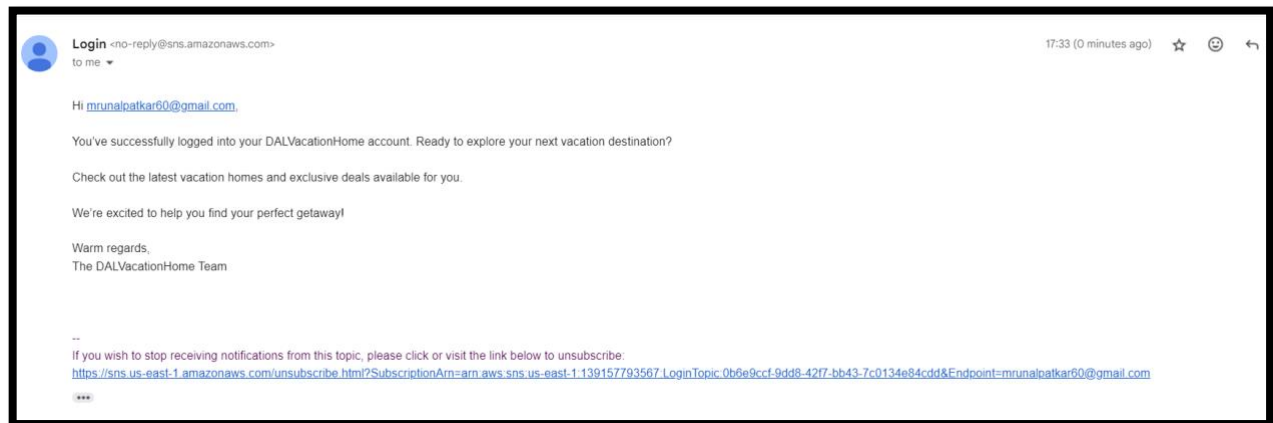


Figure 39. Login successfully mail

Individual Contribution

Table 1: Individual Contribution

Team Member	Contribution
Nikita Davies	<ul style="list-style-type: none">• Worked on the notification module specifically the room booking part and sending of confirmation/rejection notifications.• Worked on the data analysis module. Implemented the posting and analyzing of feedbacks using Google Natural Language API.• Explored and worked on AWS Lex before it got disabled• Implemented the front-end UI and API integration for the dashboard, rooms listing, rooms details and reviews pages.
Mrunal Mangesh Patkar	<ul style="list-style-type: none">• Explored Cognito user pool, user groups, and authentication flow• Provisioned and configured Cognito authentication service for the sign-up and sign-in process.• Developed the post-confirmation Lambda for Cognito to sort Users into their respective user groups and insert the user data into Dynamodb.• Developed a Lambda to fetch all the booking details for a particular Booking code (To be done by the Property agent user).
Rameez Parkar	<ul style="list-style-type: none">• Designed the code structure and flow for the Authentication module.• Created the logic, lambda and DynamoDB for the security question and answer part of authentication.• Created the logic, lambda and DynamoDB for the Caesar Cipher part of authentication.• Designed the structure for the property agents flow.• Created the Lambdas and DynamoDB to allow the property agent to add or update the room details.
Axata Darji	<ul style="list-style-type: none">• Explored and worked on AWS Lex before it got disabled• Explored Dialogflow to replace with AWS Lex• Worked on notification module such that after user has registered/logged in email notification will be shared.• Scheduling and maintaining meeting logs and recordings

Meeting Logs - Sprint 2

Table 2: Meeting Logs for Sprint 2

Meeting Date	Meeting Time (Start)	Meeting Time (End)	Meeting Place	Outcomes	Attendees	Recording link
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/teams/CSCI5410_Serverless_Team6/Shared%20Documents/General/Meeting%20Recordings/CSCI%205410-%20GroupProject-Sprint2%20planning-20240530_150058-Meeting%20Recording.mp4?csf=1&web=1&e=Pz4B8W
09/06/2024	9:00 AM	9:30 AM	online	Discussion of database design and plan for upcoming week	All present	https://dalu.sharepoint.com/teams/CSCI5410_Serverless_Team6/Shared Documents/General/../../../../v:/r/teams/CSCI5410_Serverless_Team6/Shared%20Documents/General/Meeting%20Recordings/CSCI%205410-%20GroupProject-Sprint2%20planning-20240530_150058-Meeting%20Recording.mp4?csf=1&web=1&e=Pz4B8W
18/06/2024	1:30 PM	2:30 PM	online	Discussion on how to share common services if multiple people working on same feature	All present	https://dalu.sharepoint.com/:v:/r/teams/CSCI5410_Serverless_Team6/Shared%20Documents/General/Meeting%20Recordings/CSCI%205410-%20Project%20development%20approach%20discussion-20240618_134317-Meeting%20Recording.mp4?csf=1&web=1&e=xfAye3
25/06/2024	2:30 PM	3:00 PM	online	Discussion on sprint 2 individual updates and deliverables	All present	https://dalu.sharepoint.com/:v:/r/teams/CSCI5410_Serverless_Team6/Shared%20Documents/General/Meeting%20Recordings/CSCI%205410-%20Weekly%20project%20meeting-20240625_143106-Meeting%20Recording.mp4?csf=1&web=1&e=oltrCF

04/07/2024	4:00 PM	4:20PM	online	Discussion on update for sprint 2 report	All present	https://dalu.sharepoint.com/:v:/r/teams/CSCI5410_Serverless_Team6/Shared%20Documents/General/Meeting%20Recordings/CSCI%205410-%20Sprint%202%20Update%20and%20Plan%20for%20report-20240704_160120-Meeting%20Recording.mp4?csf=1&web=1&e=XMbi93
------------	---------	--------	--------	--	-------------	---

Screenshot of team chats

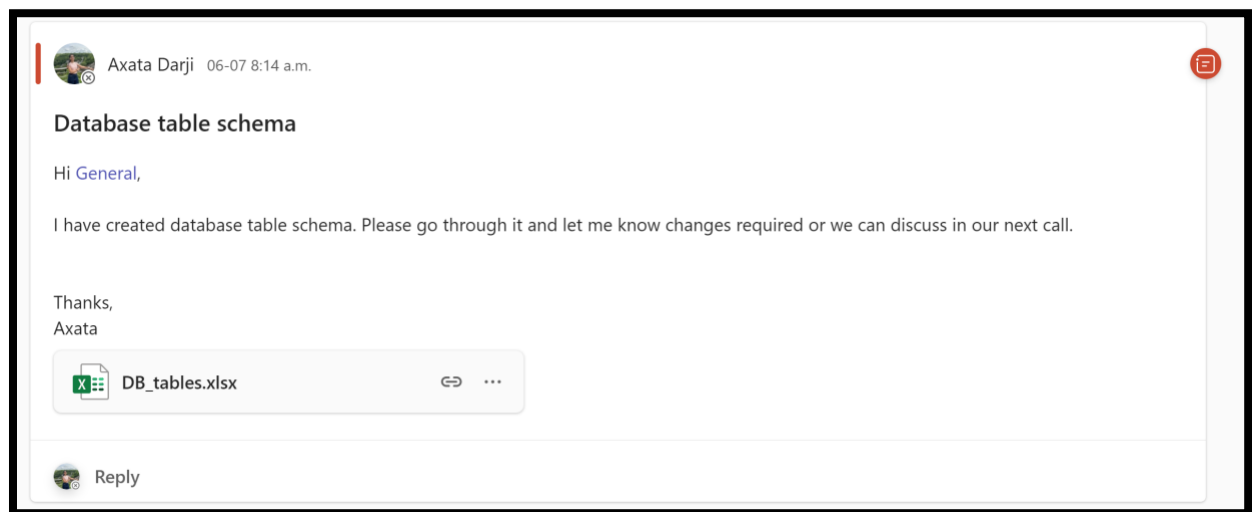


Figure 40: Screenshot of teams group chat

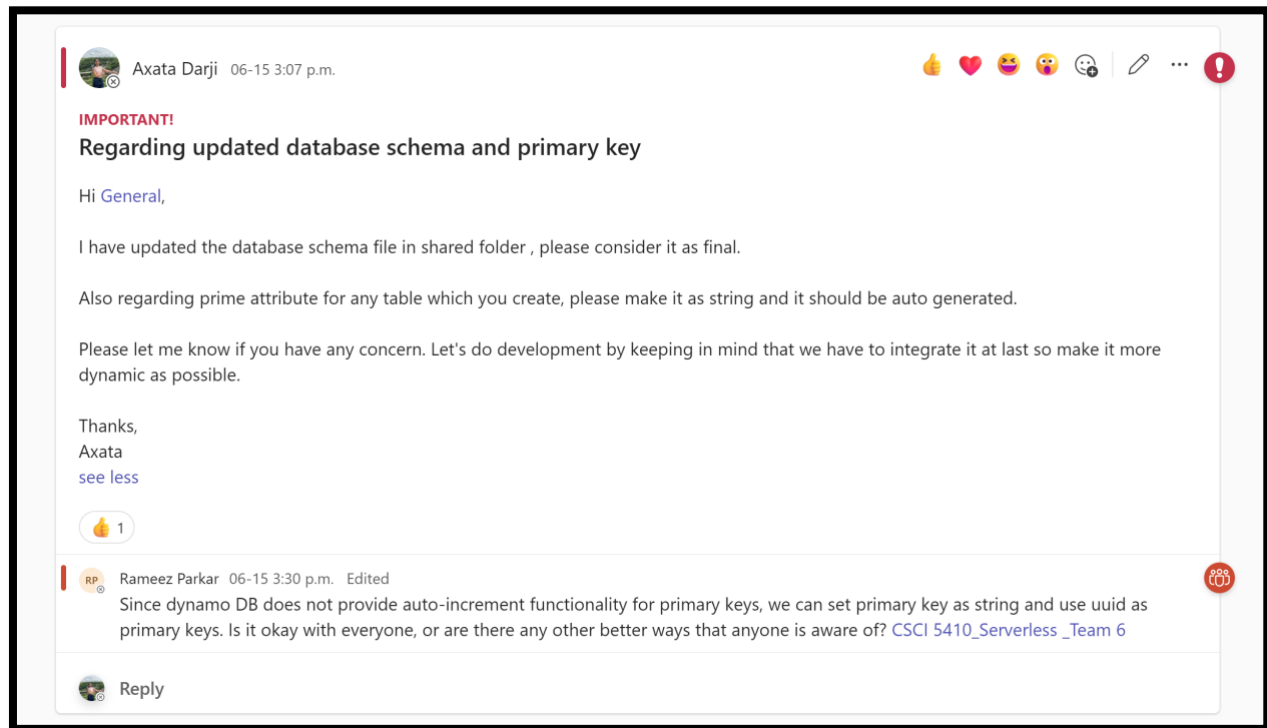


Figure 41: Screenshot of teams group chat

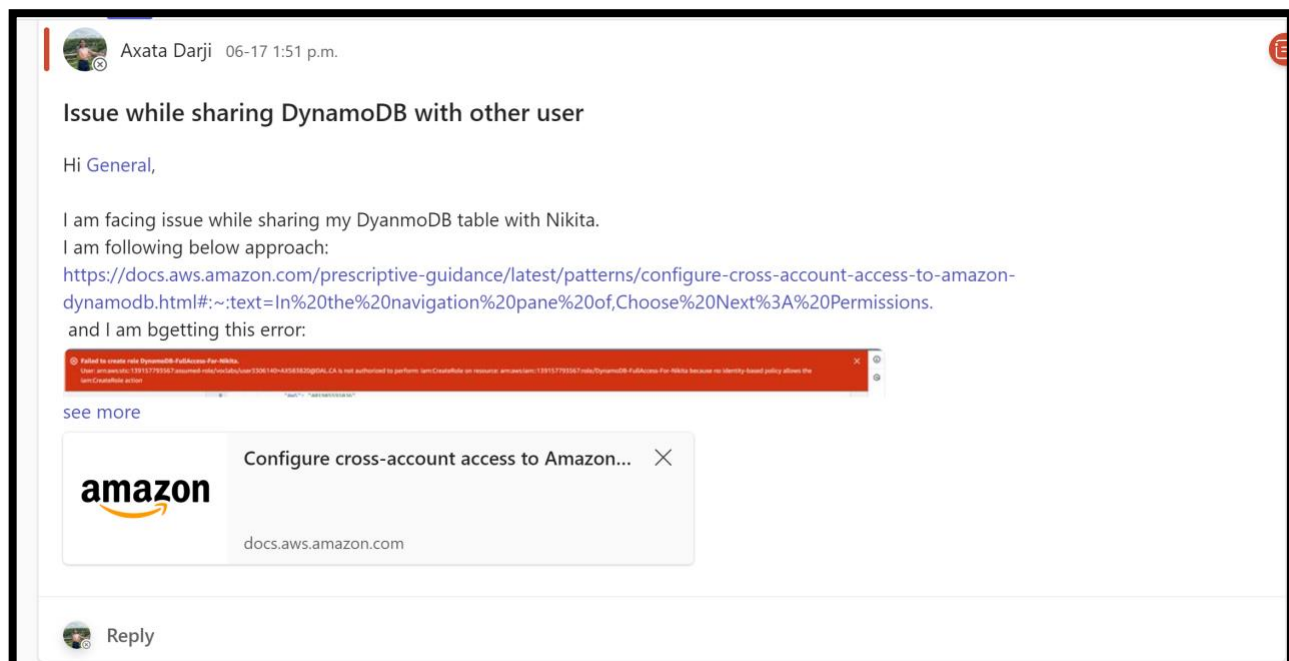


Figure 42: Screenshot of teams group chat

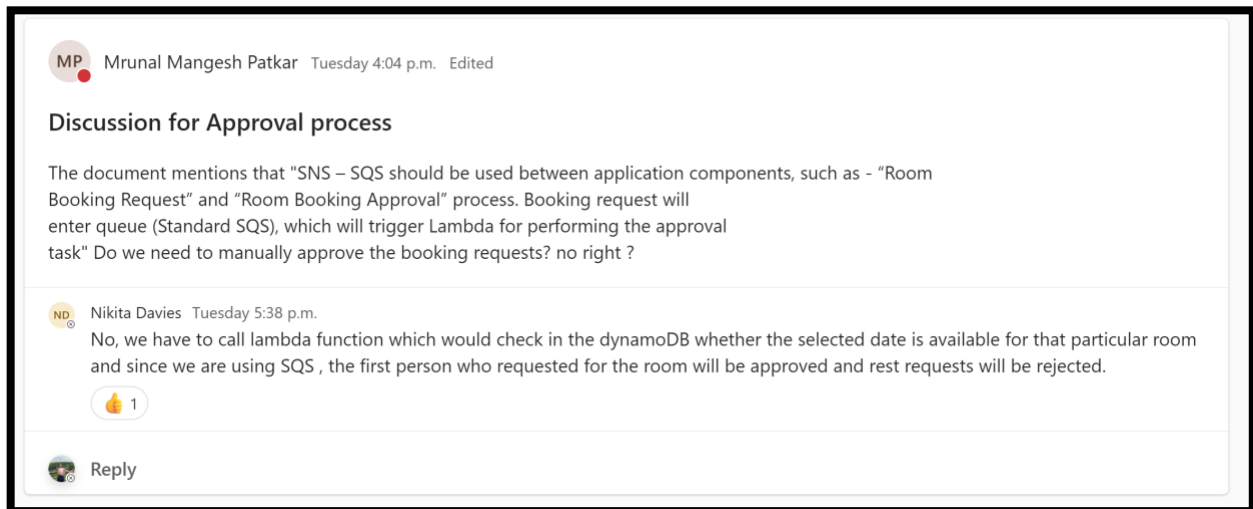


Figure 43: Screenshot of teams group chat

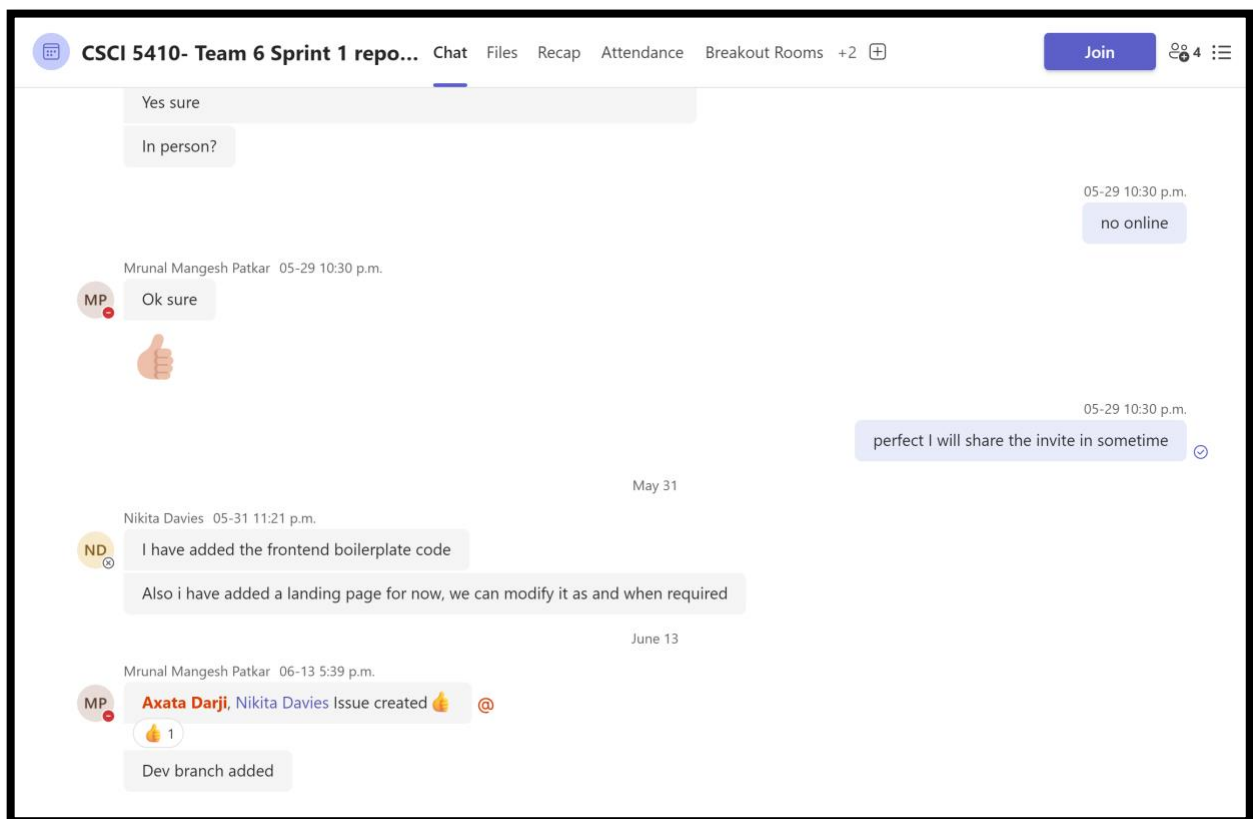


Figure 44: Screenshot of teams group chat

References

1. "Fanout to Amazon SQS queues", AWS. Accessed: June 25, 2024. [Online]. Available: <https://docs.aws.amazon.com/sns/latest/dg/sns-sqs-as-subscriber.html>.
2. "Dialogflow", Google cloud. Accessed: July 2, 2024. [Online]. Available: <https://cloud.google.com/dialogflow#benefits>
3. "Natural Language API", Google cloud. Accessed: July 5, 2024. [Online]. Available: <https://cloud.google.com/natural-language?hl=en> .
4. "Amazon Cognito," AWS Prescriptive Guidance. [Online]. Available: <https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-net-applications-security/cognito.html>
5. "Amazon Cognito Console," Amazon Web Services, Inc. [Online]. Available: <https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-console.html>
6. "Amazon Cognito User Pools," Amazon Web Services, Inc. [Online]. Available: <https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-identity-pools.html/>
7. "Invoking the Lambda Function using an API Gateway Endpoint", Amazon Web Services, Inc. [Online]. Available: <https://docs.aws.amazon.com/lambda/latest/dg/services-apigateway.html>
8. "What is Amazon DynamoDB?", AWS. . Accessed: July 5, 2024. [Online]. Available: <https://aws.amazon.com/pm/dynamodb/>
9. "AWS Lambda", AWS. . Accessed: July 5, 2024. [Online]. Available: <https://aws.amazon.com/lambda/>
10. "What is Amazon SNS?", AWS. Accessed: July 5, 2024. [Online]. Available: <https://docs.aws.amazon.com/sns/latest/dg/welcome.html>
11. "Amazon simple queue service documentation", AWS, Accessed: July 5, 2024. [Online]. Available: <https://docs.aws.amazon.com/sqs/>
12. "Create Reports from Amazon DynamoDB Data in Looker Studio", Cdata. Accessed: May 26, 2024. [Online]. Available: <https://www.cdata.com/kb/tech/dynamodb-cloud-google-data-studio.rst>
13. "What is Amazon Lex?", AWS. Accessed: May 25, 2024. [Online]. Available: <https://docs.aws.amazon.com/lex/latest/dg/what-is.html#CPMA>
14. "Pub/Sub", Google Cloud. Accessed: May 26, 2024. [Online]. Available: <https://cloud.google.com/pubsub?hl=en>

15. "Pub.Sub in GCP", Medium. Accessed: May 26, 2024. [Online]. Available: <https://medium.com/@teja.ravi474/pub-sub-in-gcp-27a21554d158>
16. Harsh Ananad,Satyam Biradar,Naveen Prajapat,Prof. Shripad G Desai, "Deployment of a Serverless Web Application using AWS services," American Journal of Science and Engineering. Accessed: May 25, 2024. [Online]. Available: https://ajse.us/wp-content/uploads/2023/03/AJSE-V3-I4_paper1.pdf
17. "Build applications or websites quickly on a fully managed platform", Google Cloud. Accessed: May 26, 2024. [Online]. Available: <https://cloud.google.com/natural-language?hl=en>