CSCI 5410

Serverless Data Processing

Sprint 3 Report

Team Members

Kaushik Chanabhai Dhola (B00956419)
Luv Prakashkumar Patel (B00950942)
Jaishankar Mohanraj(B00975938)
MD Samshad Rahman (B00968344)

Gitlab Repo link : https://git.cs.dal.ca/mohanraj/serverless_group1
Deployed URL : https://dalvac-qual5imuuq-uc.a.run.app/

DALHOUSIE UNIVERSITY

HALIFAX

Table of contents

Table of figures:

Virtual Assistant Module - [Kaushik Chanabhai Dhola]:

Research Details:

Introduction to usage of Virtual Assistant:

1. **Objective:** The objective of the virtual assistant (DalBot) is to assist users in retrieving booking details and providing guidance on using the application effectively.

2. **Market Research:** I have conducted research to understand user needs and preferences in interacting with virtual assistants for booking services and application guidance. It incorporates how the existing Assistants are responding based on the requests and question provided.
   To design and develop an effective user interface (UI) and ensure user experience (UX) for DalBot, I have conducted proper market research, particularly focusing on industry leaders like Expedia. The research aimed to understand best practices in chatbot interaction and scalability in handling a large user base. Specifically, insights were gathered from Expedia's Help Center [1] regarding:
   - **UI/UX Design:** Analyzed Expedia's UI/UX strategies to optimize user engagement and navigation.
   - **Chatbot Interaction:** Studied the mechanisms incorporated by Expedia to interact with users through chatbots.
   - **Scalability:** Explored strategies used by Expedia to manage and scale chatbot interactions across a diverse user demographic.

   This research has been essential in developing the Virtual assistant for our system as it provided proper insights as well as the workflow that the global leaders have incorporated.

3. **Technology Stack:**
   a. **Amazon Lex:** Used for natural language understanding and conversation management. This Amazon services automatically processes the request message and extract the meaning using which the specified intent would be triggered to perform the operation.
   b. **AWS Lambda:** The lambda function is used to serves as a serverless compute service for executing backend logic. Right now, I am retrieving the booking details based on the details provided by the user.
   c. **DynamoDB:** This includes the booking details and user information. This is from where lambda function gets data to present to users on chatbot.
   d. **React:** This is the JavaScript library used for building the User Interface for the Chatbot. I have not used any pre-built libraries.
   e. **JavaScript, HTML, CSS:** Some frontend technologies used for developing the user interface.

Individual Contribution [Kaushik Chanabhai Dhola]:

My responsibilities:

1. I am responsible for development and integration Amazon Lex with Lambda function and DynamoDB frontend, and coordinating Front-end Chatbot UI which is implemented with React, JavaScript, HTML, CSS.
2. I have also implemented the admin dashboard with frontend and backend functionalities:
    a. Add New Room: This form adds new type of room based on the capacity, availability, type, features and cost.
    b. Delete Room: This form deletes room from based on the combination of type and capacity given.
    c. Update Room Details: This function enables user to update the existing room details such as feature, cost, availability based on the combination of type and cost given.

My tasks:

- I have implemented Amazon Lex integration with a Lambda function to facilitate interaction with DynamoDB for booking details retrieval.
- I have also developed the frontend chatbot UI using Kommunicate AI powered platform ensuring full functionality. This functionality supports features such as state management and Interaction History Archive.
- I have developed frontend for admin dashboard using React, HTML/CSS, JavaScript.
- I have implemented the frontend for the admin dashboard for Add Room Details, Update Room Details, Delete Room using for Lambda Functions AddNewRoom, UpdateRoomDetails, DeleteRoom and FetchRoomDetails. These all functions were implemented using python 3.12 and by attaching library layers to the lambda functions.
- I have worked on the deployment process with Jaishankar to deploy the project. Also worked on the Terraform (IaC with Jaishankar) for the automated provisioning of deployment resources.

As shown below, git issue has the given 3 tasks have been implemented and closed successfully.

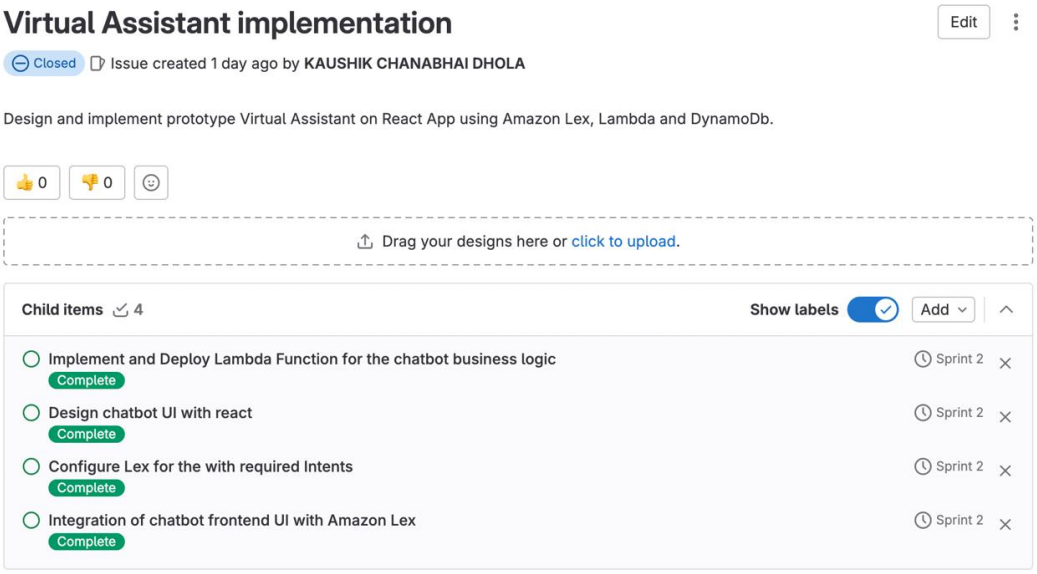

Figure 1 Virtual Assistant Issue- Git board

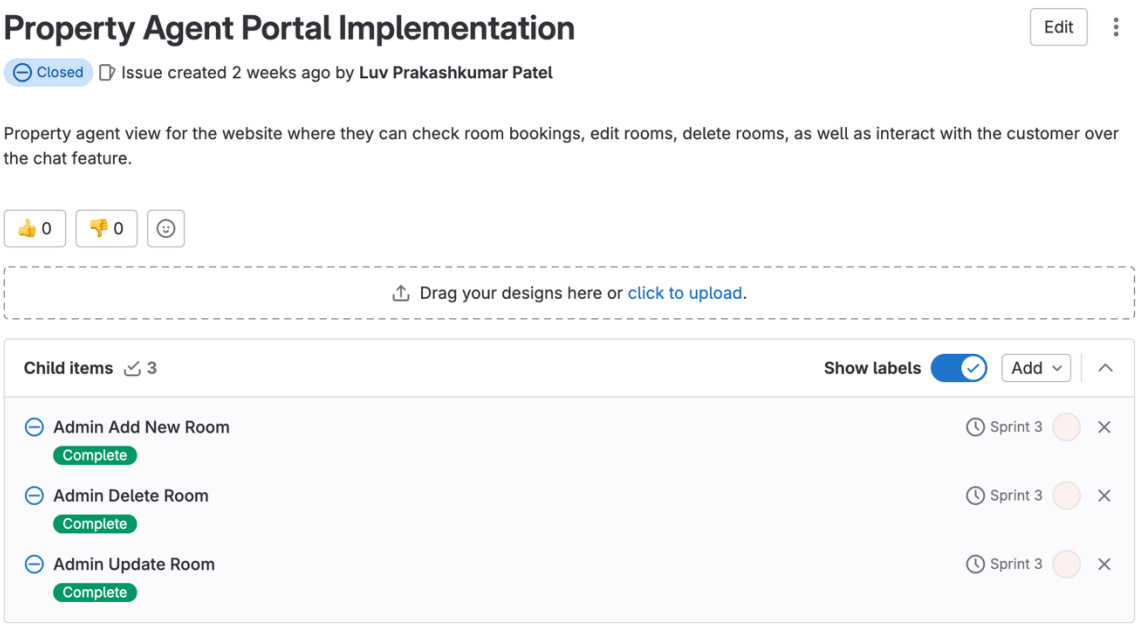

Figure 2 Admin Dashboard Issue- Git board

Architecture Design:

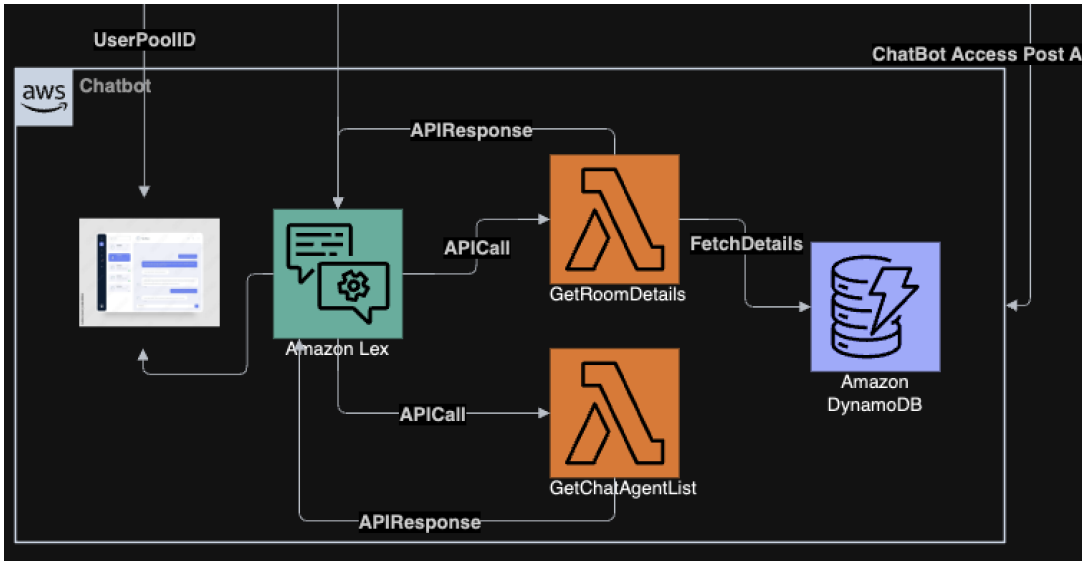

Figure 3 Chatbot Architecture

Pseudocode/ Algorithm for important modules' logic:

Lex DalBot Intents Flow:

1. bookRoom Intent:

Below figure shows the dynamic workflow of the intent for the DalBot. In this workflow, you can see the logic behind how the request is handled and how the response is generated for the bot. The request is first forwarded to lambda function which will process the query of request and generates the response based on the business logic applied on the data passed in event and data store in the DynamoDB.

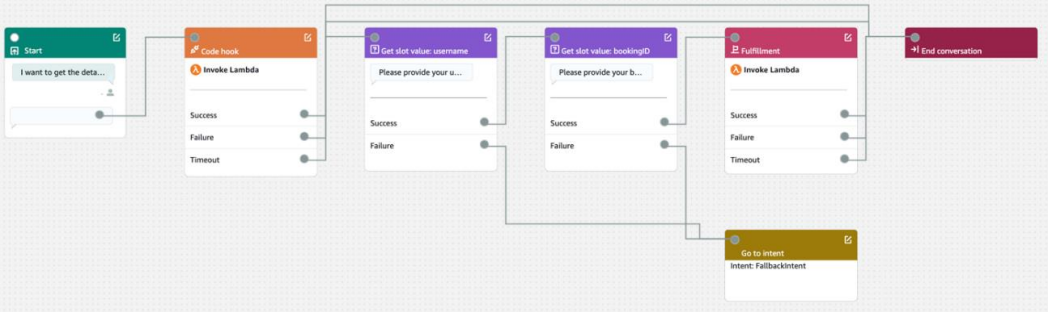

Figure 4 bookRoom Lex Intent Workflow

2. RegisterGuide Intent:

The below given workflow for the register guide is quite naïve as it required static response to be generated.

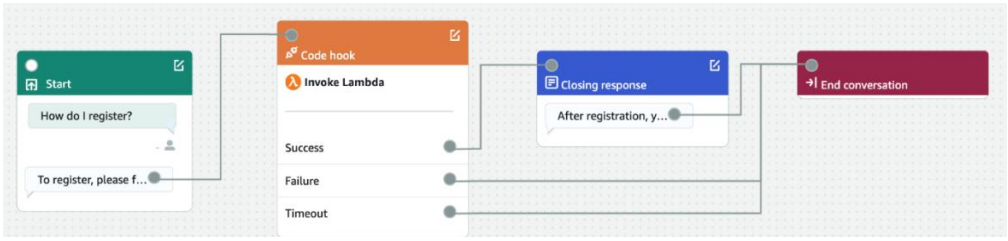

Figure 5 RegisterGuide Lex Intent Workflow

3. Greeting Intent:

The below given workflow for the Greetings to the user.

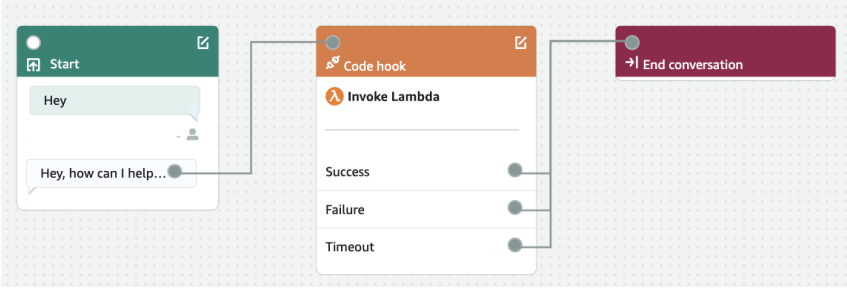

Figure 6 Greeting Lex Intent Workflow

4. TalkToAgent Intent:

The below given workflow is for the redirecting the user to the live chat conversation page.

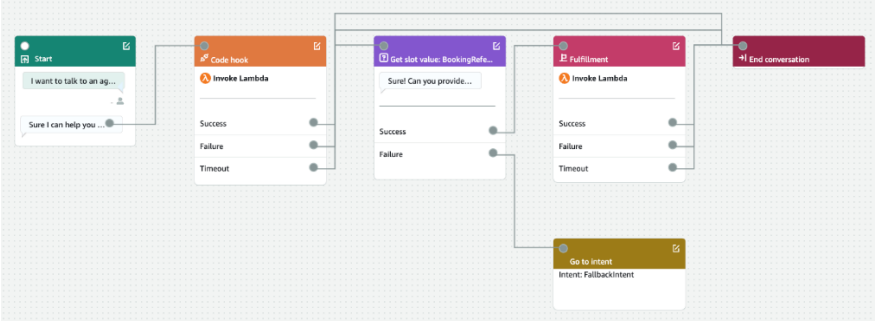

Figure 7 Talk to Agent Intent

5. Navigation Booking Intent:

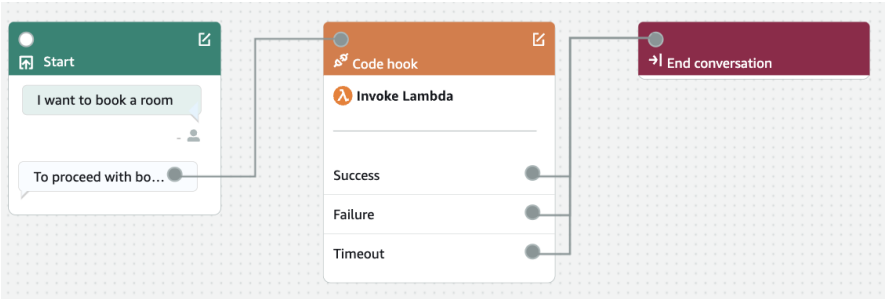

Figure 8 Navigation Booking Intent

6. Show Available Rooms Intent:

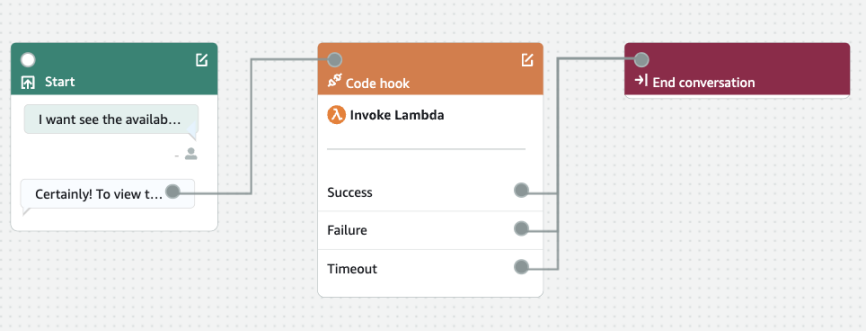

Figure 9 Show Available Rooms Intent

7. Feature Navigation Intent:

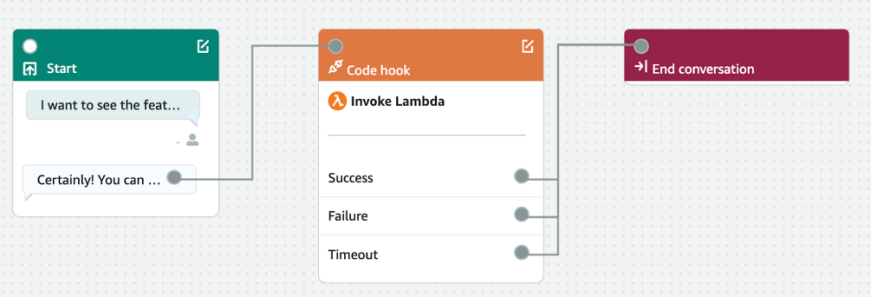

Figure 10 Feature Navigation Intent

Test Cases for Virtual Assistant and Admin Dashboard:

The implemented solution showcases a fully integrated chatbot UI with Amazon Lex, achieving end-to-end functionality. This integration leverages Kommunicate AI to facilitate the front-end and serverless interactions between Amazon Lex and AWS services such as Lambda and DynamoDB.

Successful Retrieval of booking information:

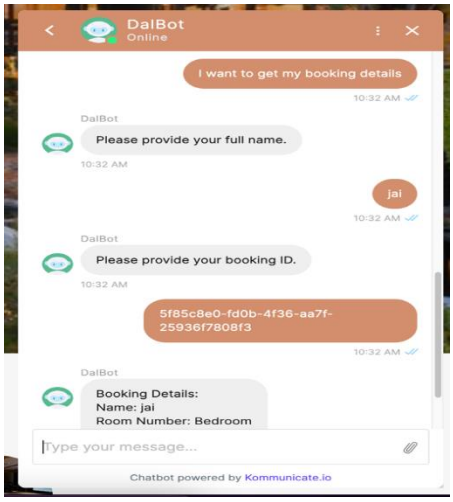

Figure 11 Test 1 Successful Retrieval of booking Information

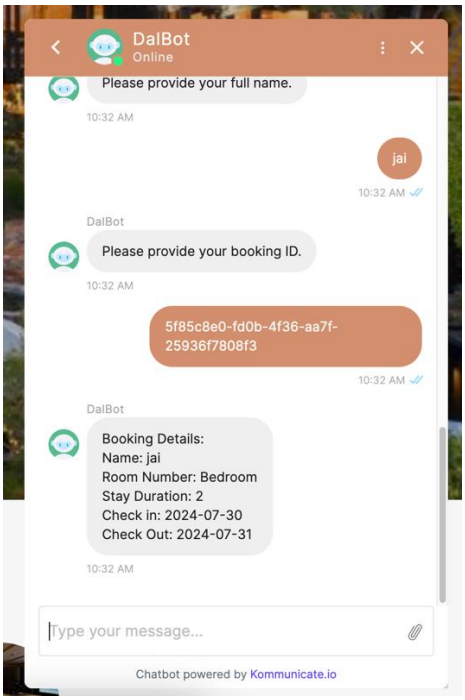

Figure 12 Test 1 Successful Retrieval of booking Information 1

Whenever user request to get the booking information from the bot, it will first check if the user exists in the system, followed by the booking ID reference check. If these both checks are valid then only the booking details can be retrieved from the DynamoDB.

If the details are not valid then the below response would be generated from the lambda function:

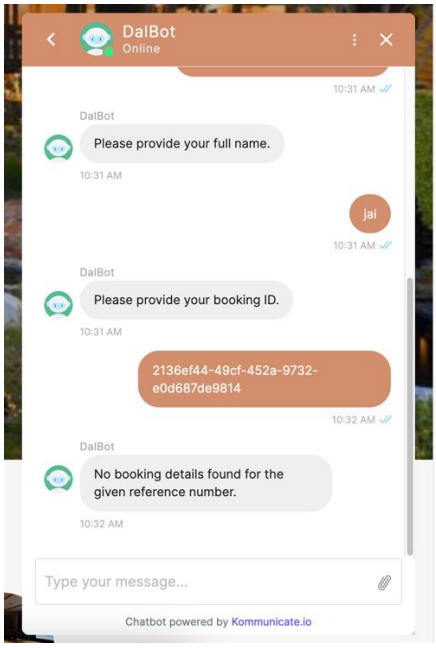

Figure 13 Test 2 Invalid Response from Chatbot

User Guide Chat:

If the user wants to know the details on web application FAQs or information regarding application access, then they can easily navigate to the chatbot and ask whatever question they might have in their mind. Below is the scenario where user asks, “How to register for a account” and gets response on the procedure for the registration.

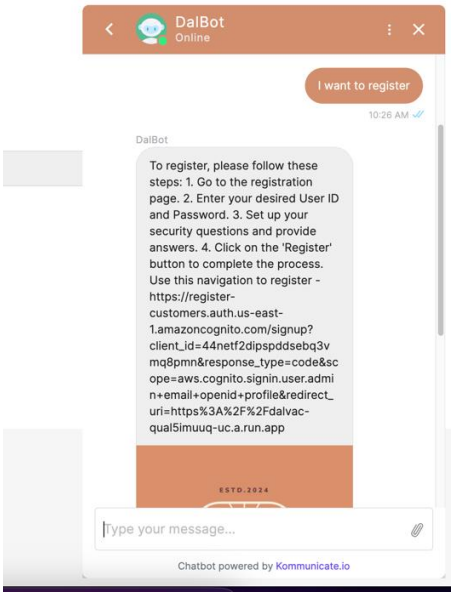

Figure 14 Test 3 User Guide Chat

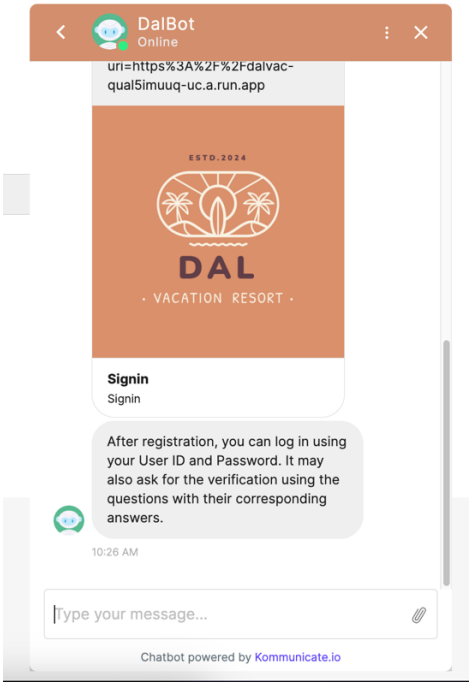

Figure 15 Test 3 User Guide Chat 1

Available Room Guide:

If user wants to inquire about the available rooms, then they can directly ask a query in the chatbot which will provide a user guide to know about the available rooms.

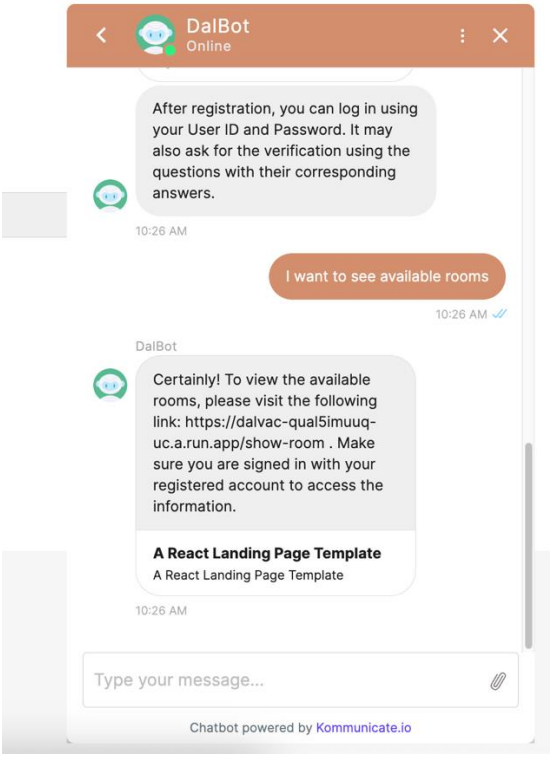

Figure 16 Test 3 User Guide Chat 2

Book a Room:

If user wants to know about the booking process, then they can directly ask a query in the chatbot which will provide a user guide to know about the book room procedure.

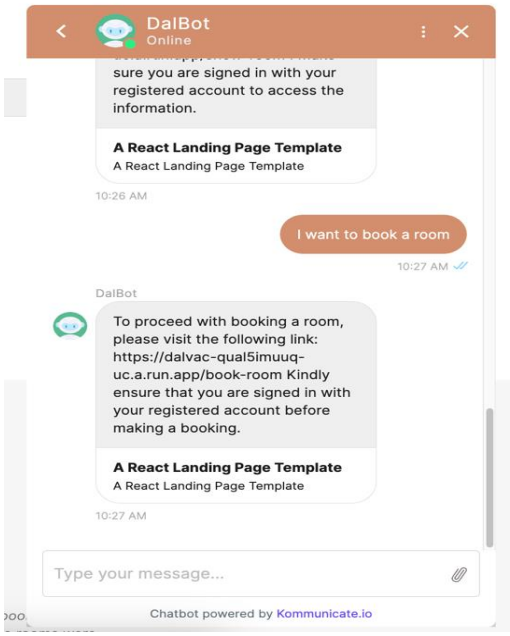

Figure 17 Test 3 User Guide Chat 3.

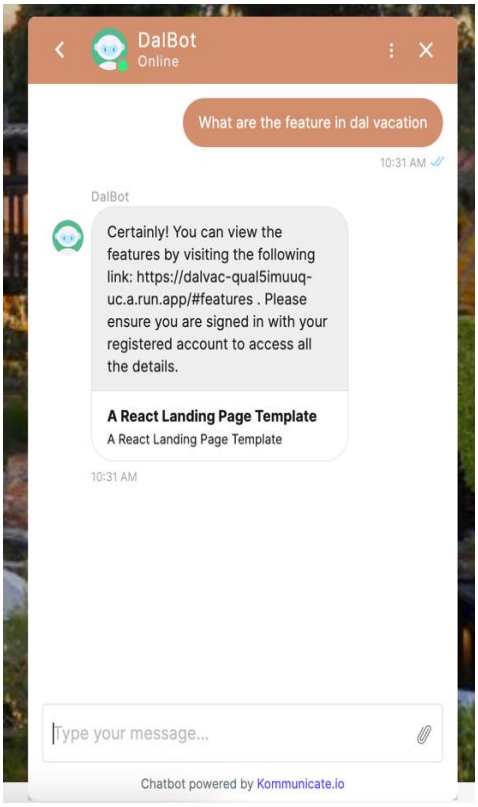

Figure 18 Test 3 User Guide Chat 4

Session wise chat history:

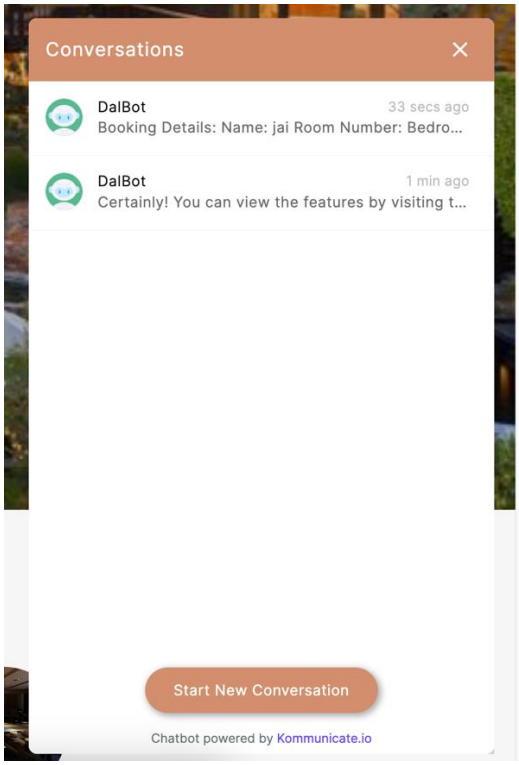

Figure 19 Session wise chat history

Admin Dashboard Testing:

Add Room Testing:

Here, the below page is for adding new room with the type, capacity, feature, available rooms and cost. It is only accessed by the admin.

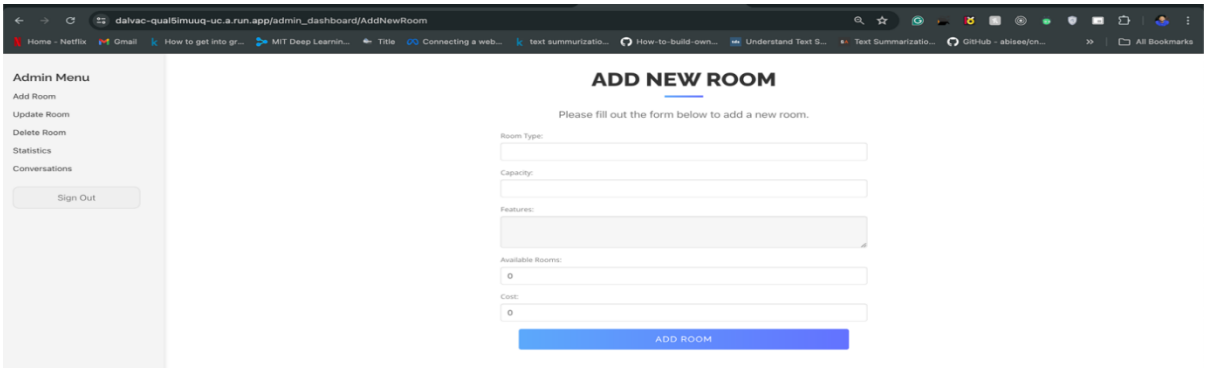

Figure 20 Admin Add Room Test Case 1

In the below snapshot, you can see that I can add the product using the given form.

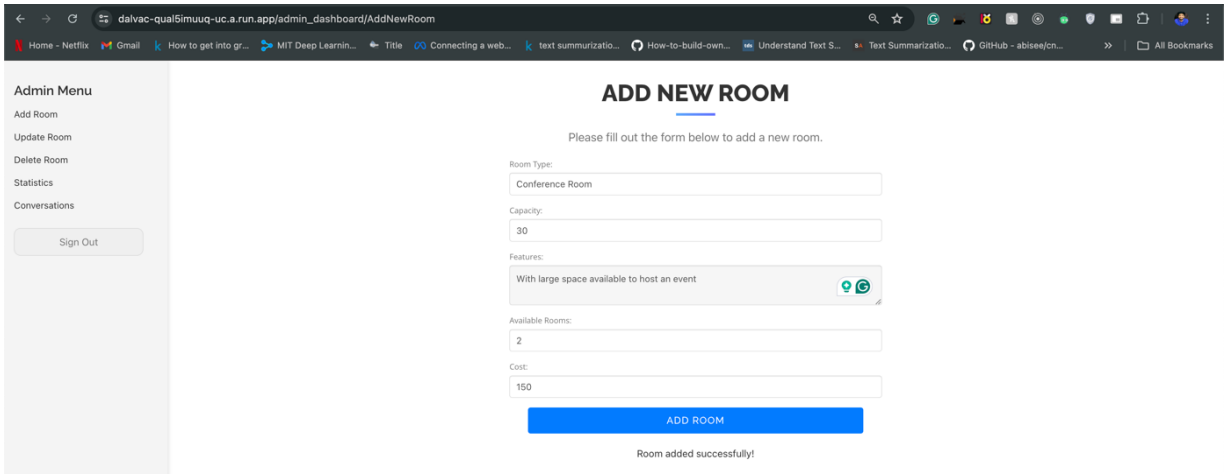

Figure 21 Admin Add Room Test Case 2

The below given screenshot shows the data being added in the DynamoDB.

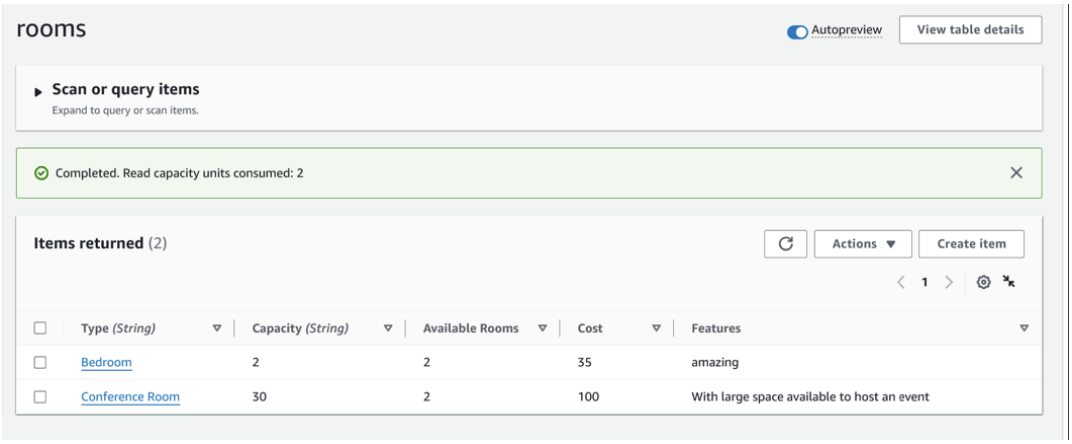

Figure 22 Added in the DynamoDB

Update Room Testing:

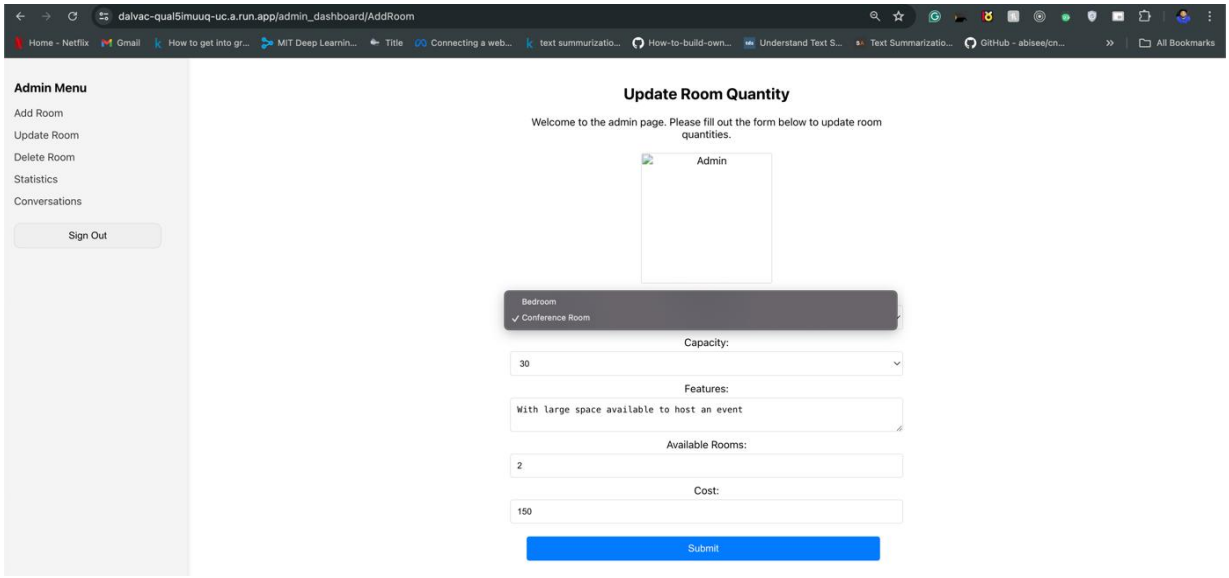

Figure 23 Admin Update Room Test 1

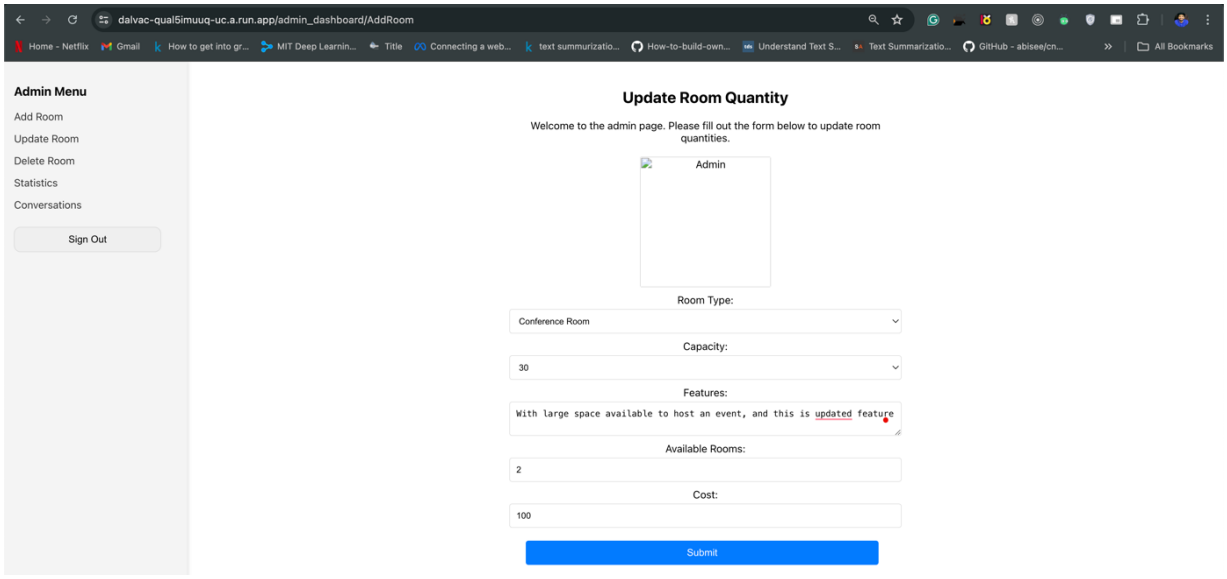

Figure 22 Admin Update Room Test 2

The below given screenshot shows the existing data being updated in the DynamoDB.

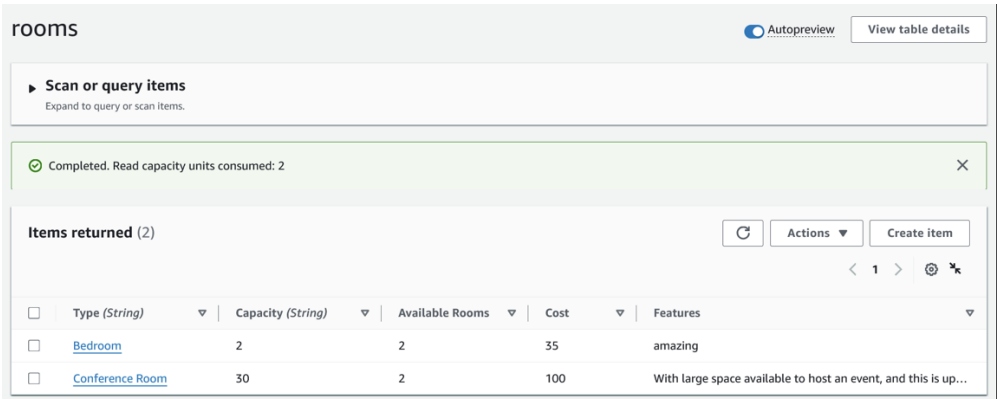

Figure 24 Updated in DynamoDB

Admin Delete Room Testing:

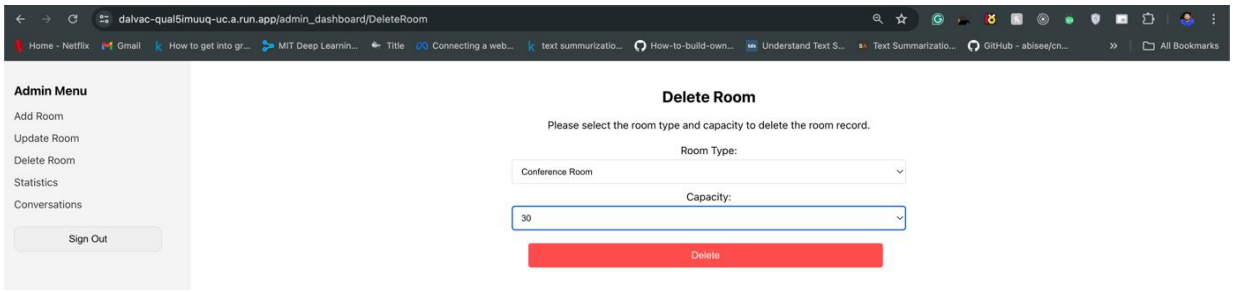

Figure 25 Admin Delete Room Test 1

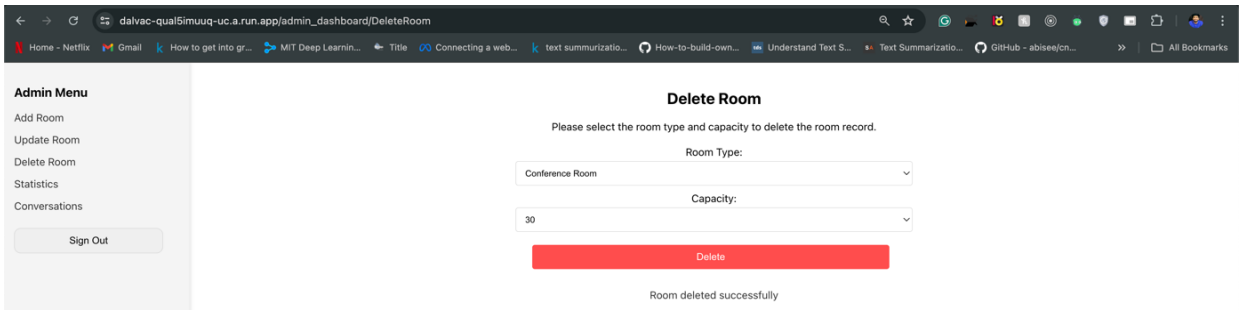

Figure 26 Admin Delete Room Test 2

The below given screenshot shows the data being deleted in the DynamoDB.

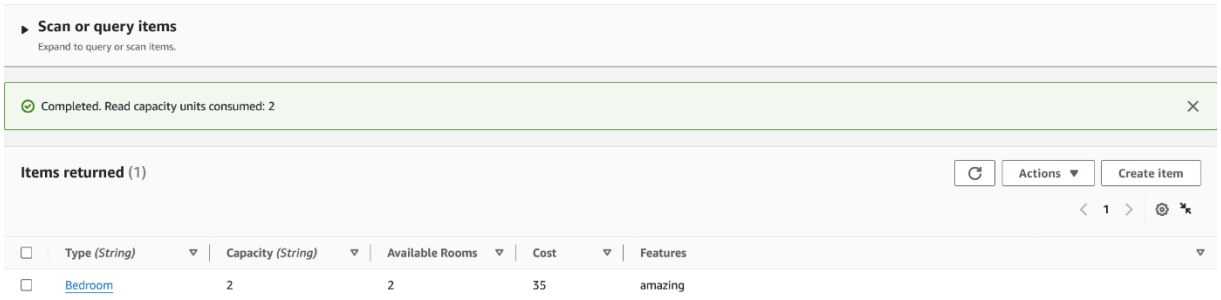

Figure 27 Delete record from DynamoDB

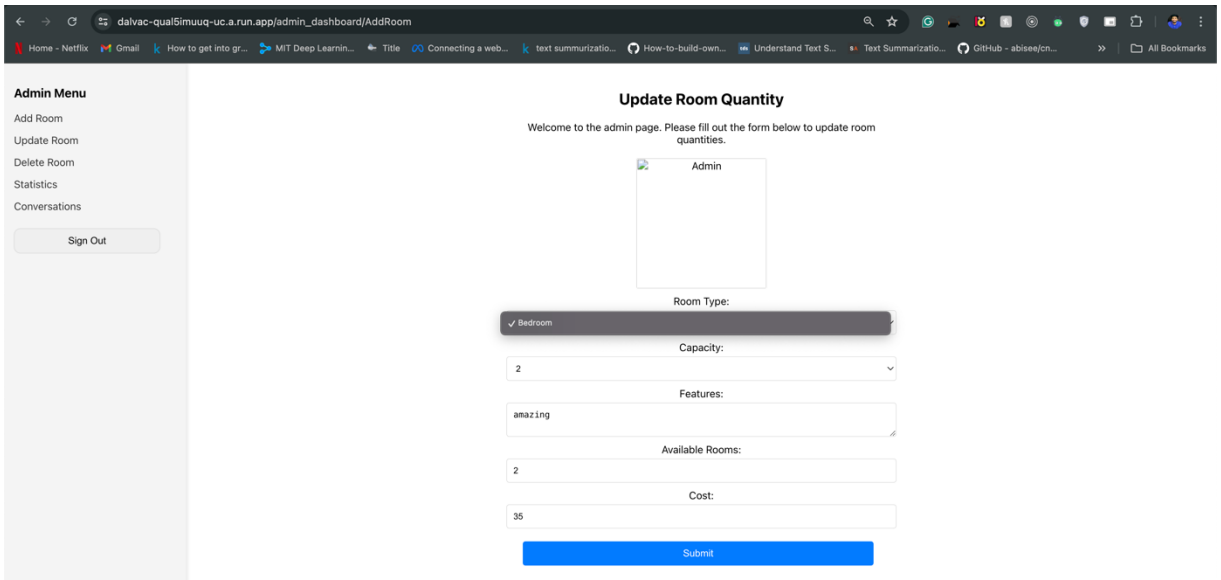

Figure 28 Updated Record in the Update Form

User Authentication [Luv Patel]

Research Details:

Introduction to User Authentication :

1. **Objective :** The objective of the user authentication is to allow only the authenticated users to access the features which are not available to the guest users who are just browsing the website. The goal is the implement 3-layer authentication system to verify the users.

2. **Market Research :** I have done research on how to leverage the power of the AWS cognito to perform the first layer user authentication. The second layer required the security question and answer page where the user needs to enter his/her security answer for the question which they set during registering. Next the 3rd layer is the ceaser cipher authentication step, where the user needs to decrypt the cipher that is provided to them using a decrypt key value.

Once, all these steps are successfully completed, the user is fully authenticated and can now access all the features of the website. To design and develop the required task, I have thoroughly gone through the secure government website like Canada Revenue Agency (CRA)[2] who already has this kind of multi layer authentication.

- Examined CRA's flow for authentication and researched on how similar thing can be done our project.
- Learned the architecture to implement the required task while focusing on increasing the user engagement and experience.

This exploration played a crucial role in helping me understand the multi factor authentication authentication flow for user. I also got to know how it is currently being implemented by one of the government bodies where security and privacy are crucial aspects.

Following the documentation of the AWS for cognito, I came across the user groups that I can create, so that i could manage the permissions of each user group separately, by giving specific access to each one. Moreover, I discovered that I need to configure the callback URLs for the signin and signup processes. This URLs should match with the URLs mentioned in the redirect URL that is present in the query string of the calling string. If this URls are different then the cognito will not allow the redirection, thus stopping the program execution and the flow[4].

Technology Stack :

a. **AWS Cognito :** Used to handle the authentication flow, specifically for the 1st layer authentication which is taken care by cognito. This service by AWS manages the user pool, user groups, and tokens of the user for seamless authentication and management.
b. **AWS Lambda :** The lambda functions are used to implement the business logic by running the serverless compute service which handles the backend code, without having to worry about the server.
c. **DynamoDB :** This is the database used for storing the information like the user security Q&A , as well as other information like the userRole, etc. This all functionality is achieved by the lambda functions which triggers the storage of data from the frontend to the database.
d. **React :** It is a frontend framework based on JavaScript, used for development of the frontend for the DalVacationHome.

Individual Contribution [Luv Patel]

My responsibilities: I was assigned the task of implementing the multi-layer user authentication and integrating it with front-end technology like React which will also be developed by me. Moreover, I was responsible for integrating the backend lambda functions and database with the frontend to offer a complete experience to the users of the website.

My Task:

- I have implemented the 3 layer user authentication for the DalVacationHome web application using the AWS services like AWS Cognito, AWS Lambda as well as AWS DynamoDB.
- Development of the frontend part for integrating with the backend to offer a seamless experience to the users using out website.
- Creating user pool for the implementation so that users can authenticate themselves.
- Implementing lambda functions to interact with the dynamoDB and Cognito.
- Moreover, I have implemented the protected routes that restricts the users from accessing any page that they are restricted to or are not allowed unless they are authenticated to do so.
- Also, on the admin side, i have implemented teh sidebar navigation, that enables the admin to navigate between various components on the admin dashboard

Below shown screenshot shows the issue created on gitlab for project management as well as the corresponding tasks.

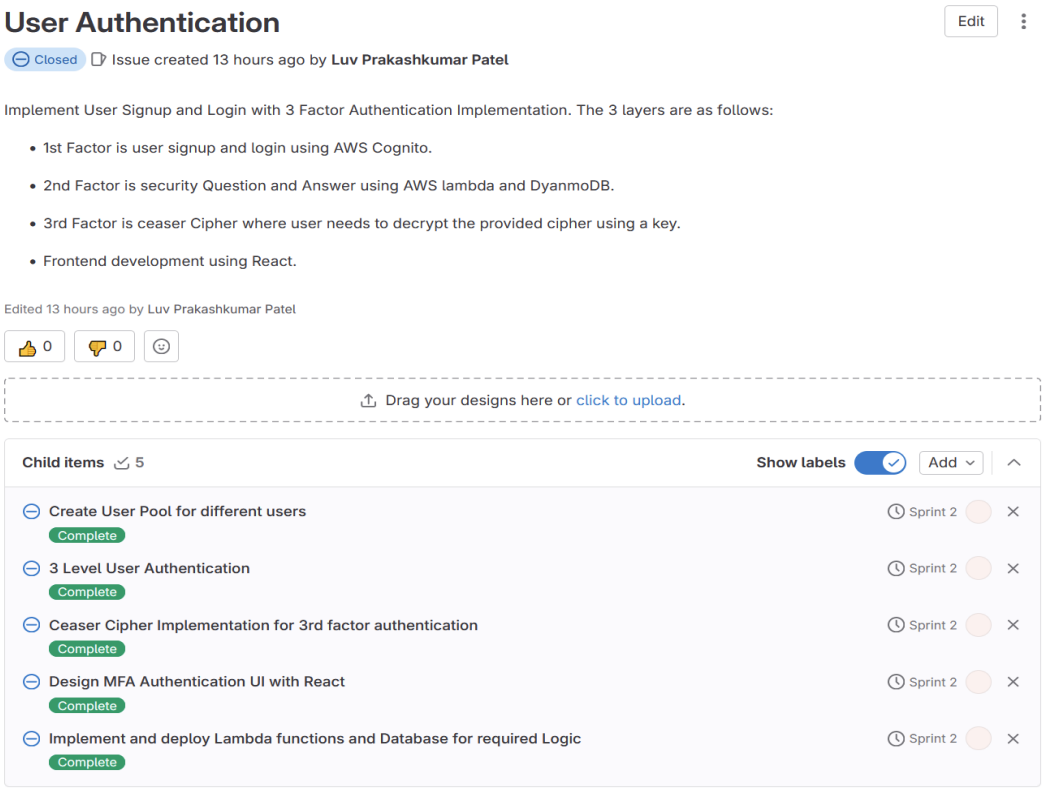

Figure 29 Issue created in the Issueboard

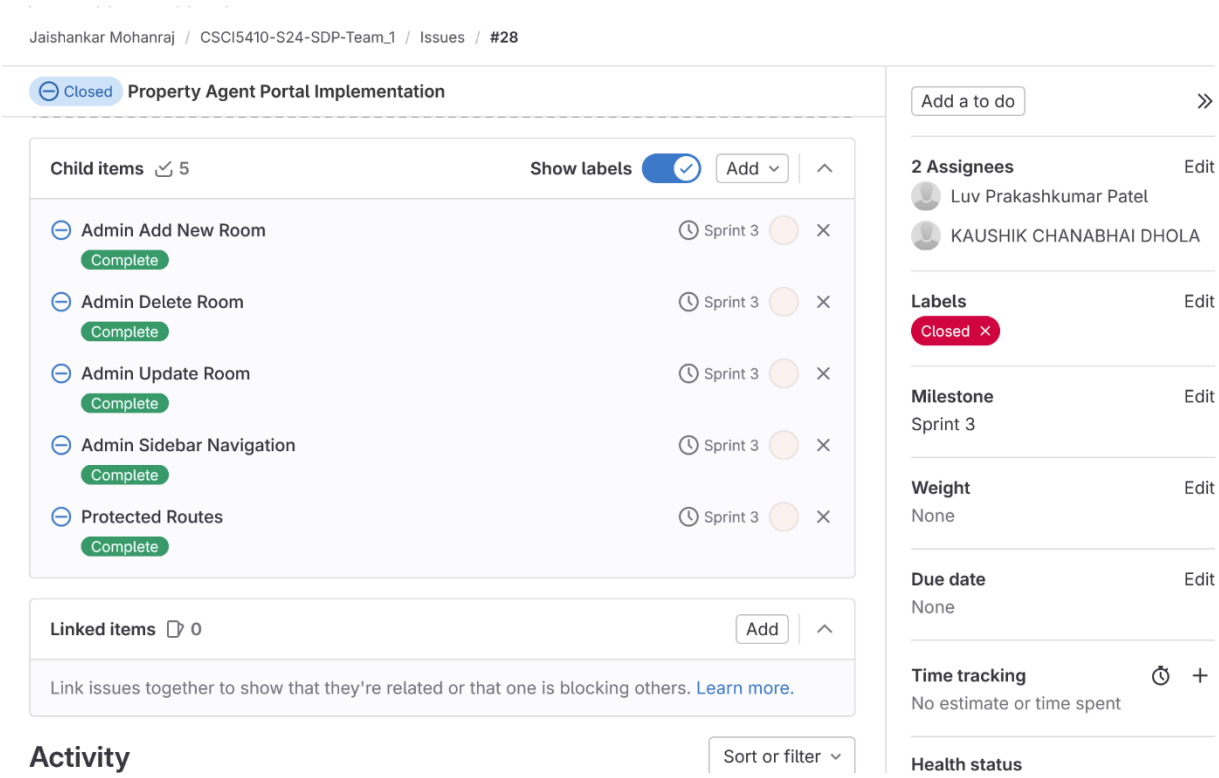

Figure 30 Issue Created in the issueboard

Algorithm for important module Logic:

1). Signin Module:

```
if (code) {
  try {
    const response = await axios.post('https://register-customers.auth.us-east-1.amazoncognito.com/oauth2/token'
      new URLSearchParams({
        grant_type: 'authorization_code',
        client_id: '44netf2dipspddsebq3vmq8pmn',
        code: code,
        redirect_uri: 'http://localhost:3000/signin' // The redirect URI
      }), {
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded'
      }
    });
```

Figure 31: Algorithm for signin module authentication

=> The above displayed piece of code I believe, is the most important aspect of the project among many others, this is because this code is handling the authentication of the code that is returned by the AWS cognito after the user is successfully authenticated. Ideally, when the user authenticates themselves using cognito, it returns the users to the redirect URLs, along with a authentication code as a query string parameter, that helps identify the user authentication in the project frontend.

The above mentioned code takes in account the oauth2 link and calls with while sending the grant type, the code generated after the authentication is successful by the cognito, and the client ID for which app client this step needs to take place. This is where the exchange of the authentication tokens takes place. Without verification and exchange of the authentication tokens, we cannot move forward with the other tasks that will happen in the project .

2). User pool groups assigning :

```
def add_user_to_group(user_id, user_role):
    try:
        group_name = 'RegisteredUsers' if user_role == 'RegisteredUser' else 'PropertyAgents'

        response = cognito.admin_add_user_to_group(
            UserPoolId='us-east-1_uBDCBcAXG',
            Username=user_id,
            GroupName=group_name
        )
        print(f'User added to group {group_name} in Cognito User Pool')
    except Exception as e:
        print(f'Error adding user to group in Cognito User Pool: {str(e)}')
        raise
```

Figure 32: Algorithm for user group assignment

This is another important code, where the users are categorized in the groups as per the custom attributes in the congito pool. This is where the users will be added to the respective group which will have different setting and permission on what they can perform in the websites and what they can access.

Architecture Diagram :

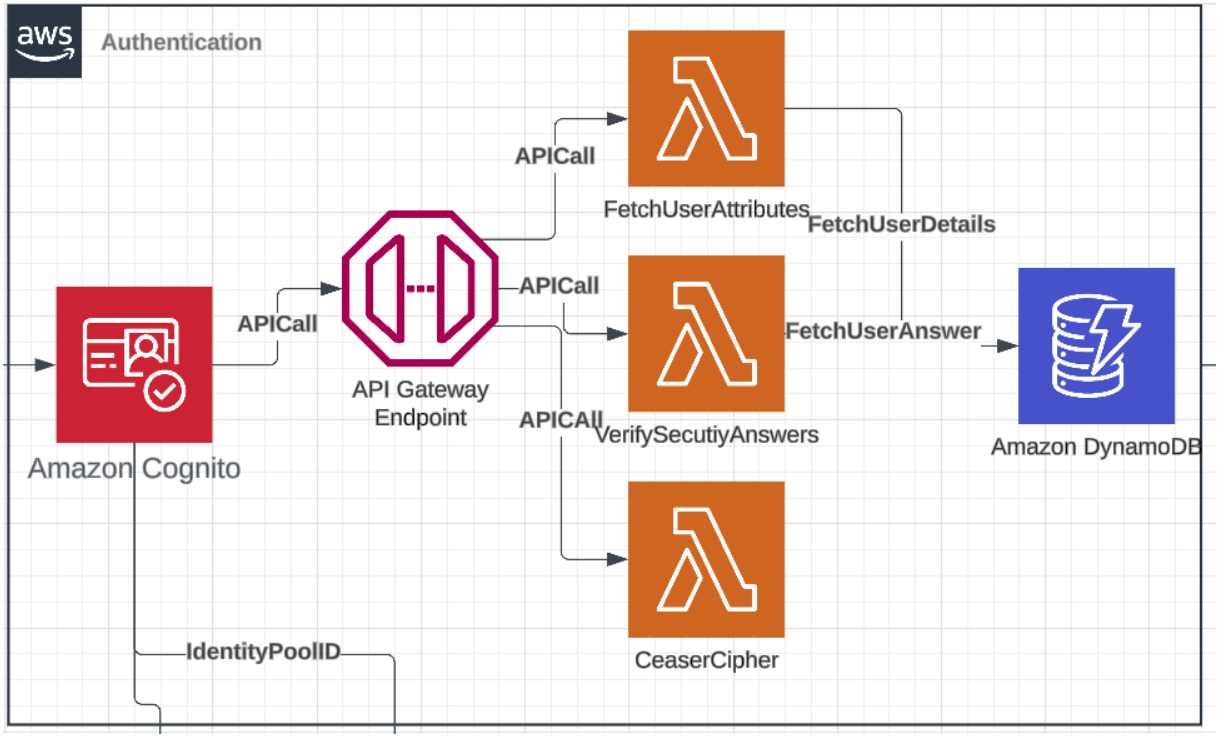

Figure 33: Authentication Architecture Diagram

Test Cases:

1). Integrating the Signin component with the Homepage Navigation bar. So that the user can click on the signIn button

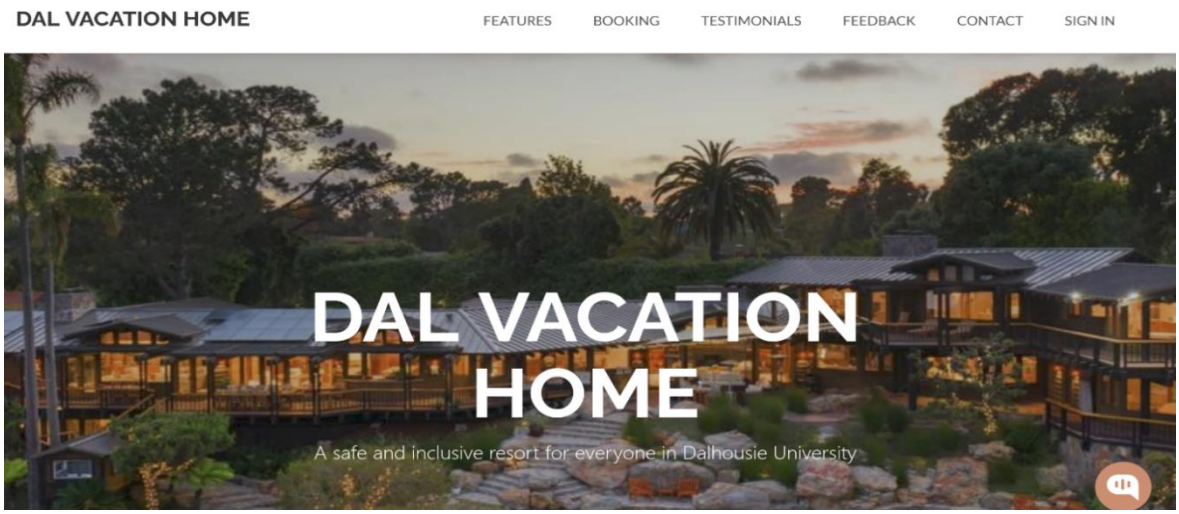

Figure 34: Homepage when user enters the website

=> Upon Clicking on the signin , the user will be redirected to the signin page, which is shown as below.

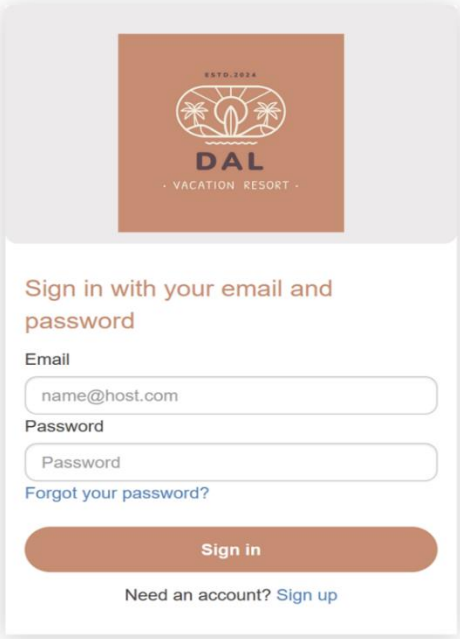

Figure 35: Application Sign in Page

=> If the user credentials are wrong/ User does not exist, then prompts for wrong/invalid credentials.

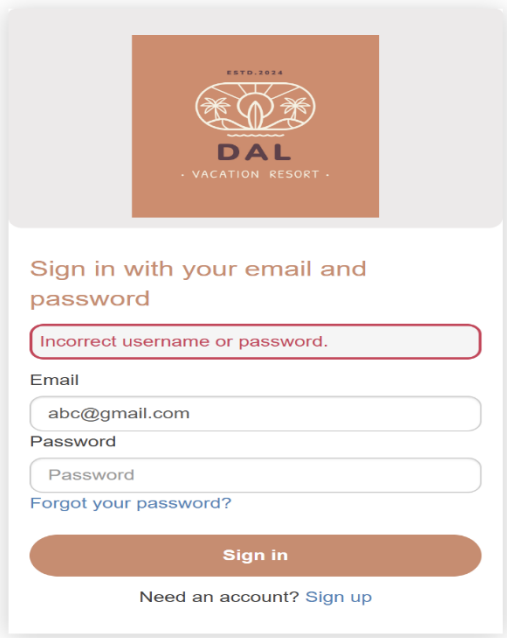

Figure 36:User enters incorrect credentials

=> User enters the registered credentials to signing to the application:

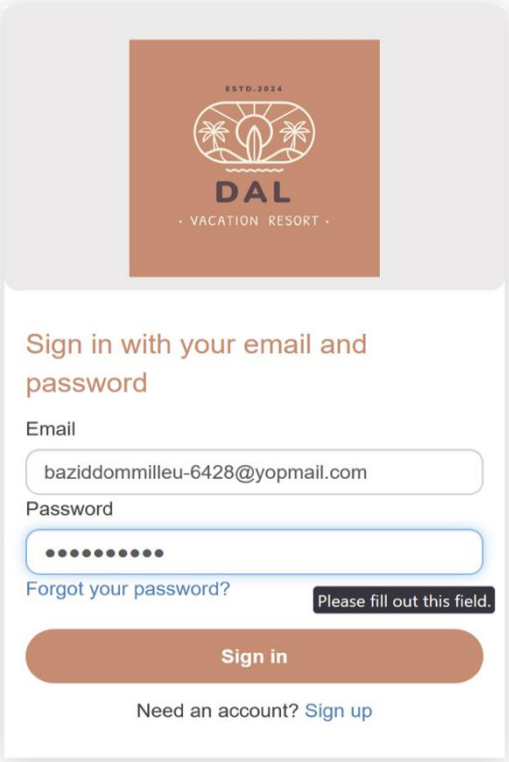

Figure 37: Entering correct credentials

=> Upon successful 1st layer authentication using AWS cognito, the user is redirected to the security answer page where they need to answer the security question they setup during registration :

Figure 38: Security Answer Page

=> Answering the security question correctly, redirects the user to the 3rd authentication layer that is, ceaser cipher verification.

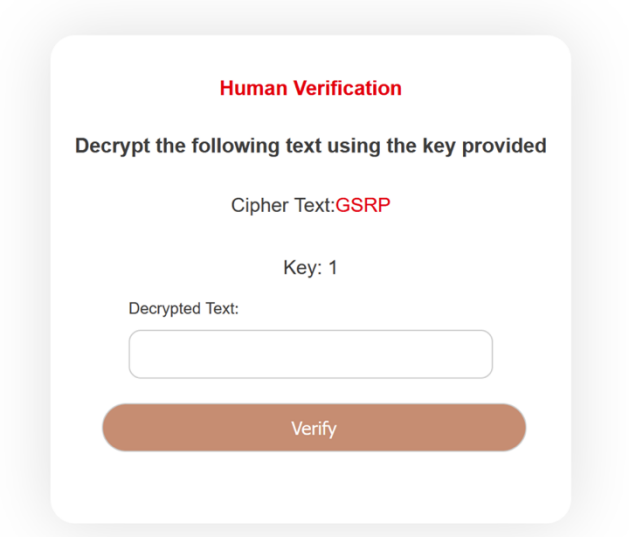

Figure 39:Decrypt Cipher text

=> User decrypts the cipher using the key provided on the screen.

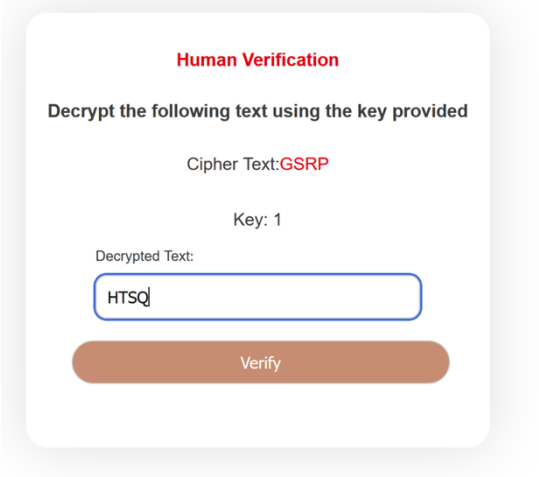

Figure 40 Entering the correct text

=> After successful authentication , the user is redirected back to the homepage where, now the signing button changes to the signout button as the user is fully authenticated.

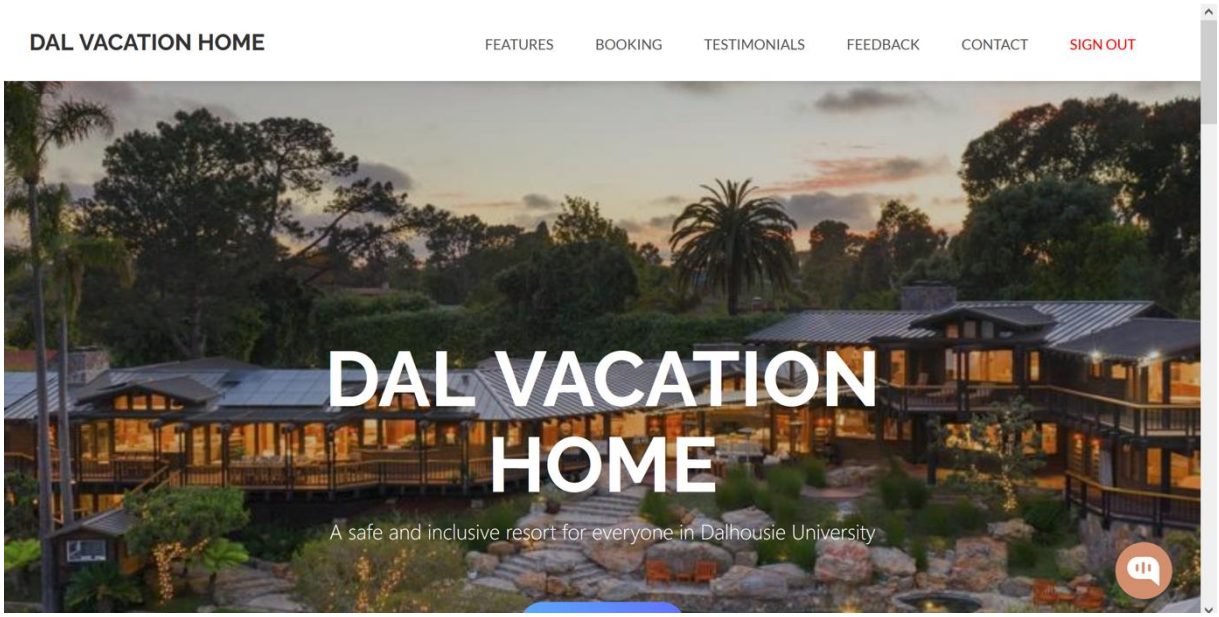

Figure 41 Homepage with signout component

=> If the user wants to sign up for the first time , then he needs to click on the signup button where he will need to enter required details as shown below:

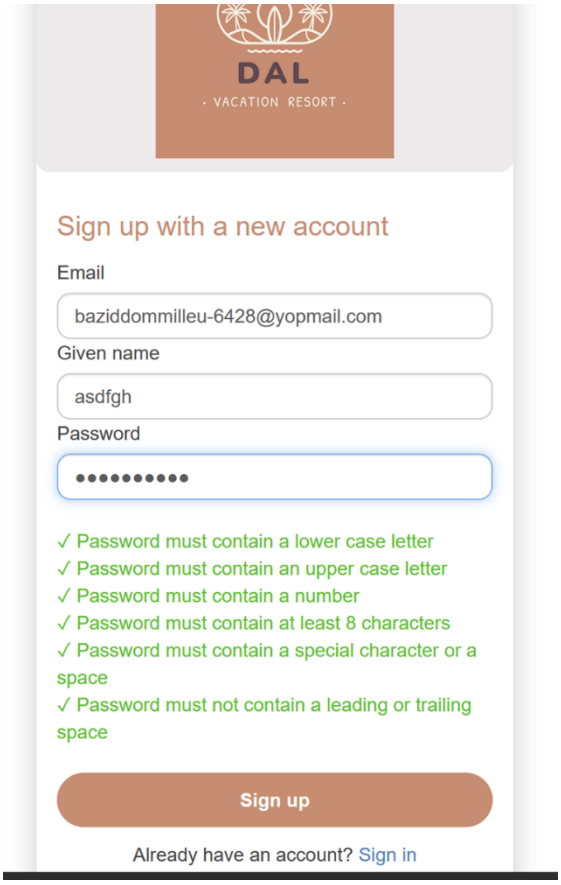

Figure 42 Cognito SignUp page with valid credentials

=> After fulfilling the details, if the user clicks on the signup button , he will be prompted to verify the account :

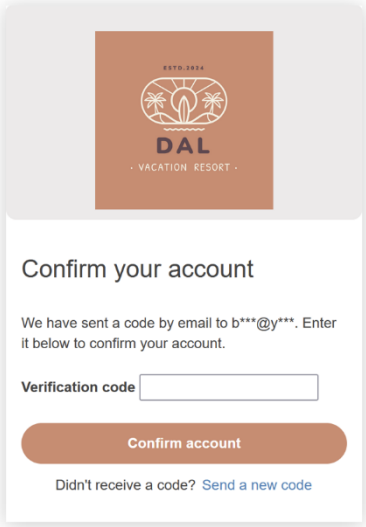

Figure 43 User account verification code page

=> Below is the verification code that is sent to the user's email:

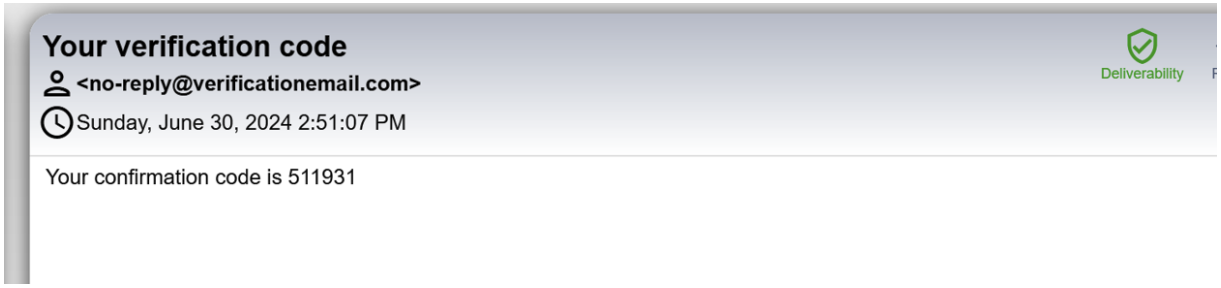

Figure 44 Verification code mail to user

=> Upon entering the verification code, the user will be successfully verified.

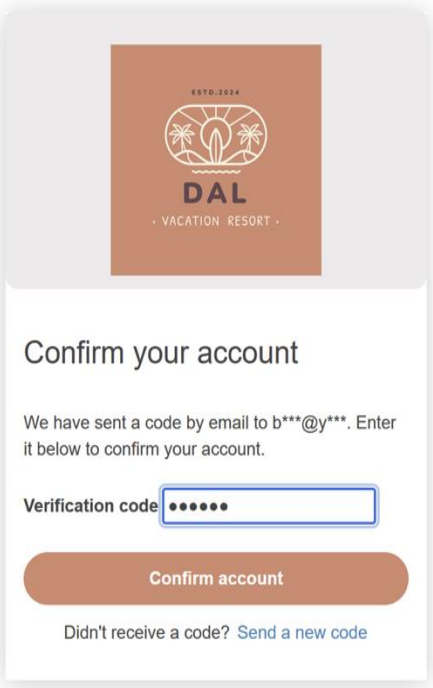

Figure 45 Enter the correct verification code

=> After successfully adding the verification code, the user will then be prompted to set his security question and answer as shown below and decide the user role based on his requirements:

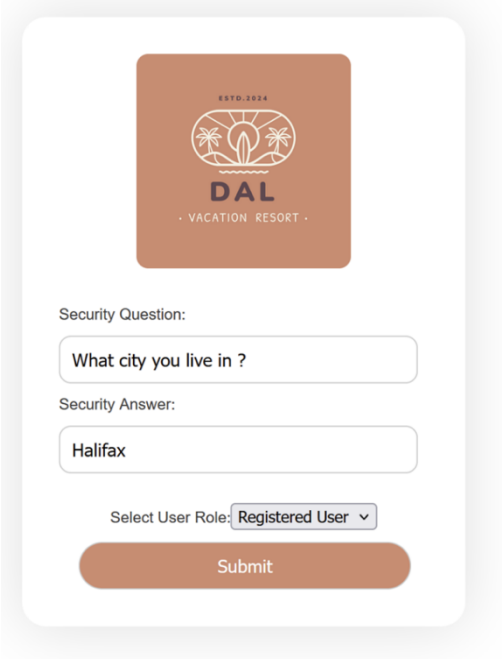

Figure 46 User security Question Setup page

=> After setting the security factors, the user will then be prompted to signin using his new credentials:

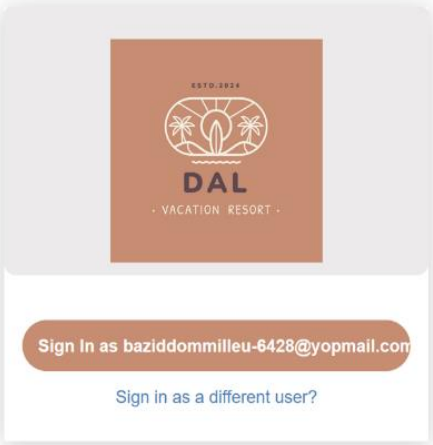

Figure 47 User sign in prompt page

=> For the user security question, if the user enters a wrong answer in the answer input, then the below mentioned error is displayed:

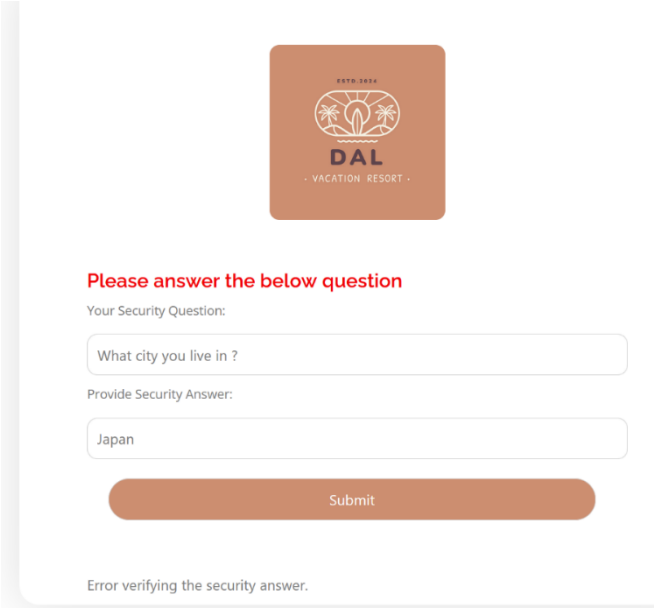

Figure 48 User security answer page

=> if the user does not satisfy the password requirement, then he will not be able to click the signup button until the password requirement is met.

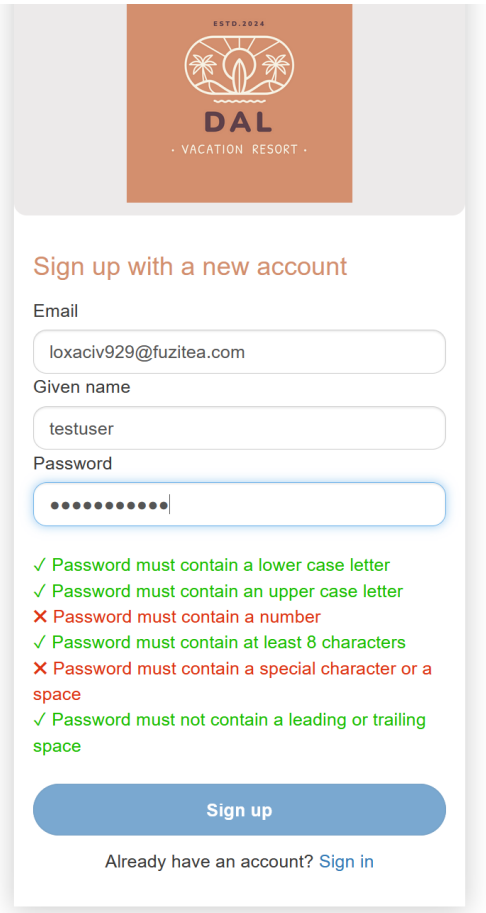

Figure 49 User not satisfying the password requirement

=> What if the user enters a wrong verification code during the registration process

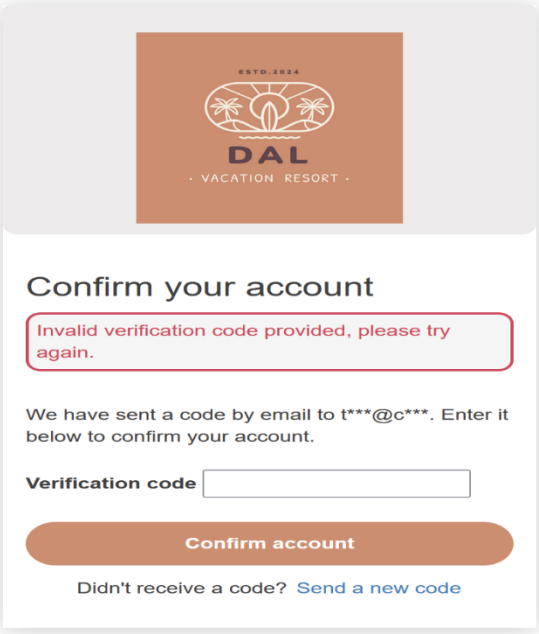

Figure 50 User enters wrong verification code

=> What if the user enters an incorrect word formed using the key provided, then there will be an alert saying that the incorrect Decryption!

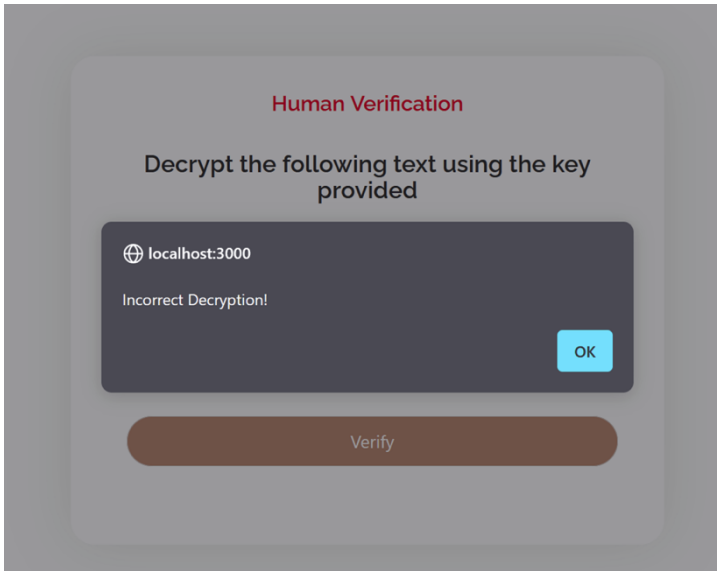

Figure 51 User enters incorrect decrypt code

=> If a user tries to signin with an already registered email address, then he will get error saying user already exists:

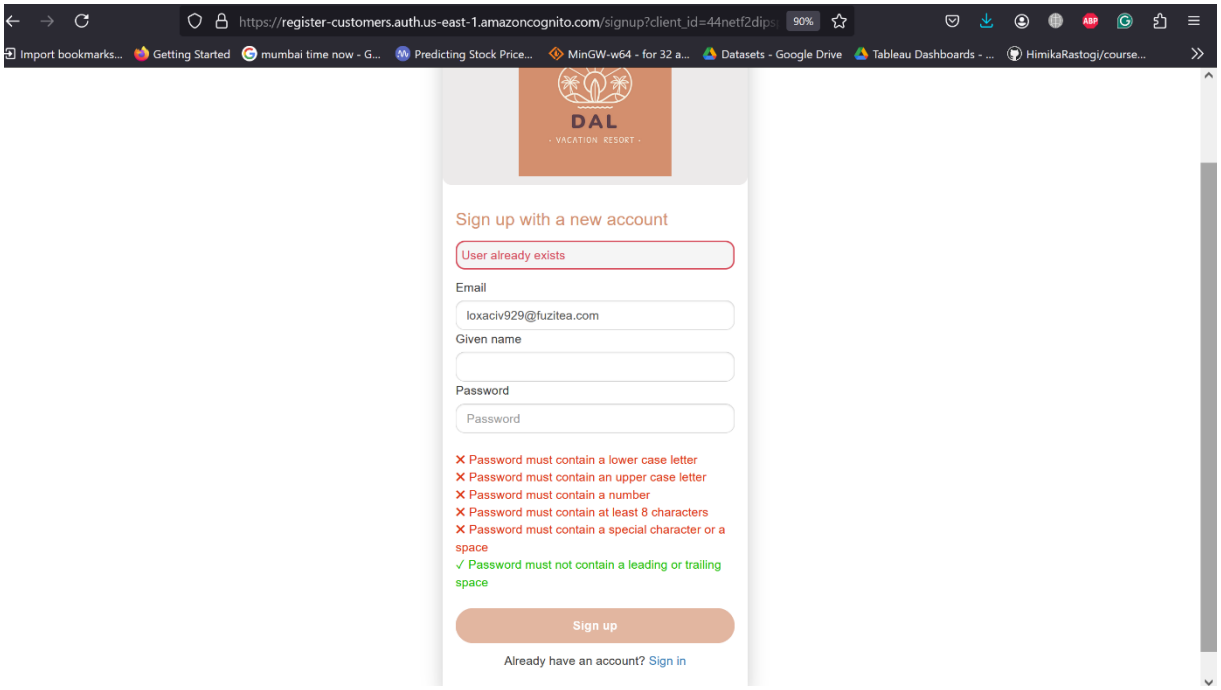

Figure 52 User signup with already registered account

=> Admin side sidebar navigation for the admin to navigate to the different components of the admin dashboard.

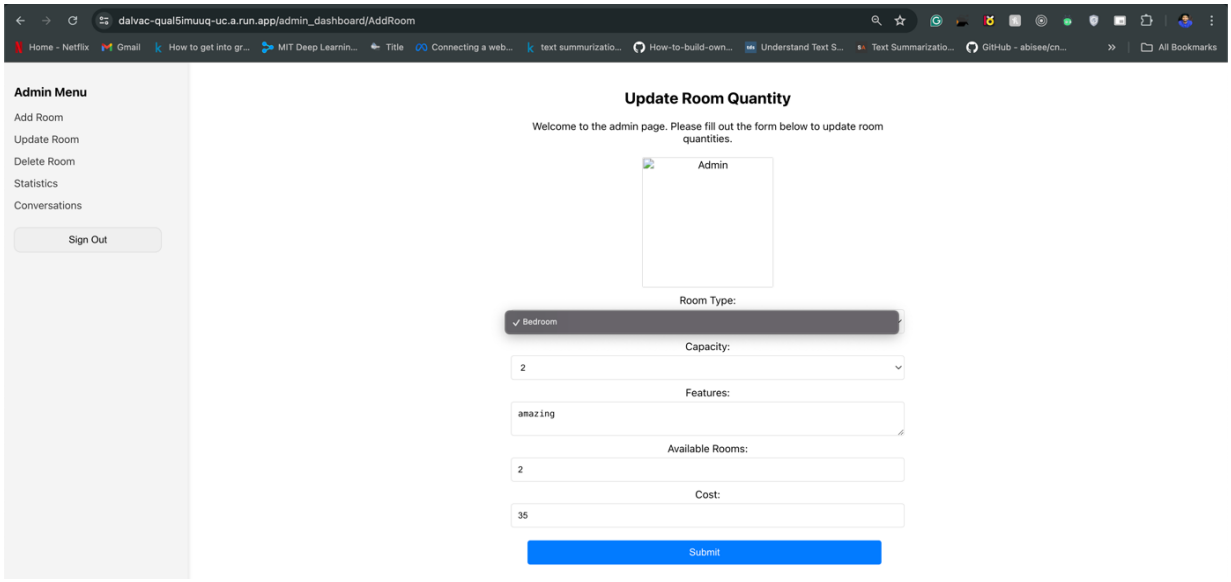

Figure 53 Admin dashboard sidebar navigation

=> Also, the routes are protected from unauthorized access. So, if the users are authenticated, then only one will be able to navigate to different pages else they will be prompted to sign in and authenticate.

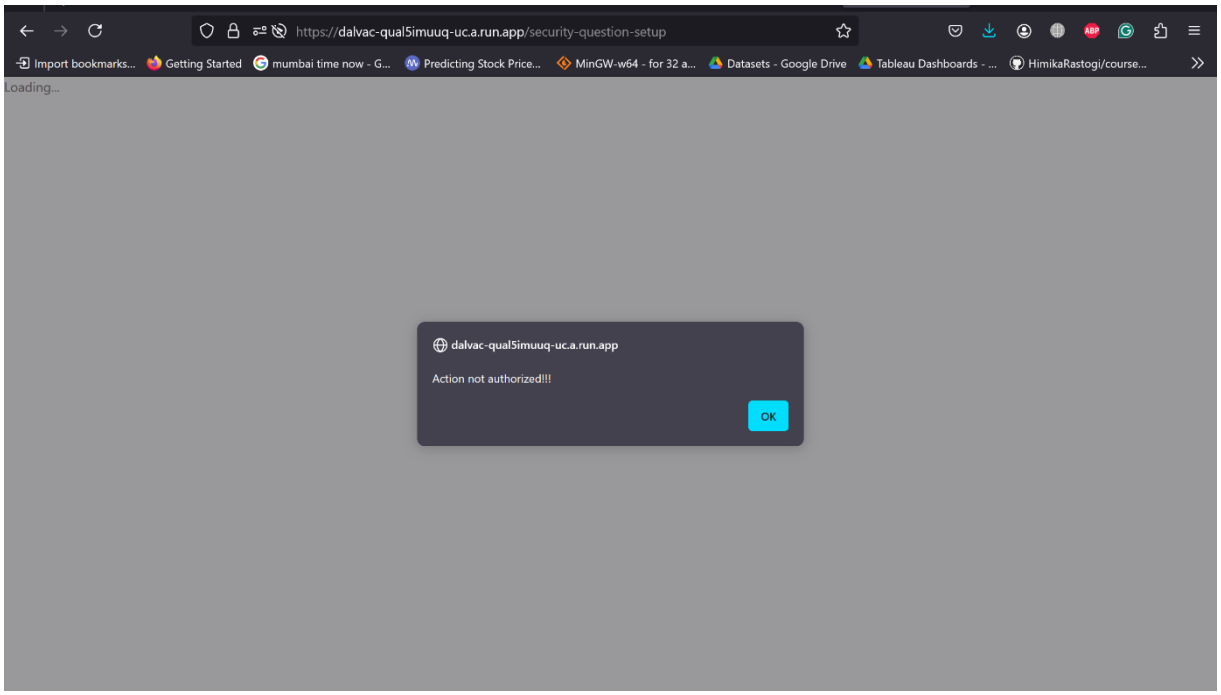

Figure 54 User tries to navigate manually without signin

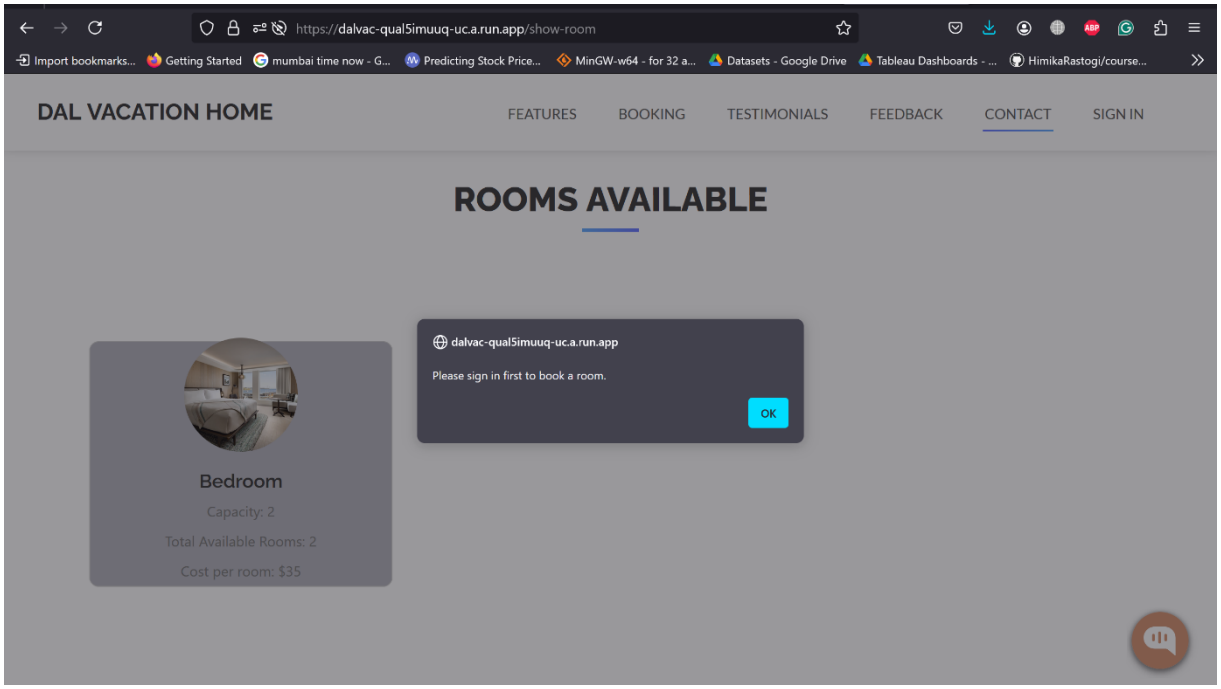

Figure 55 User signup with already registered account

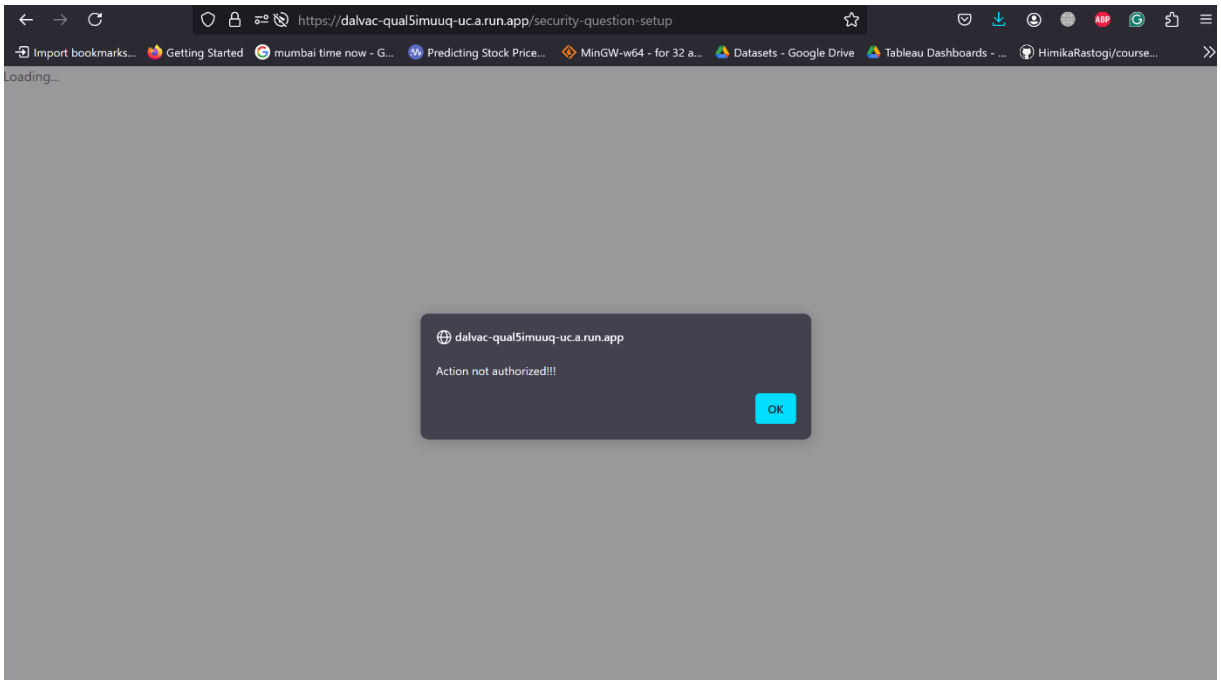

Figure 56 User tries to navigate manually without signin

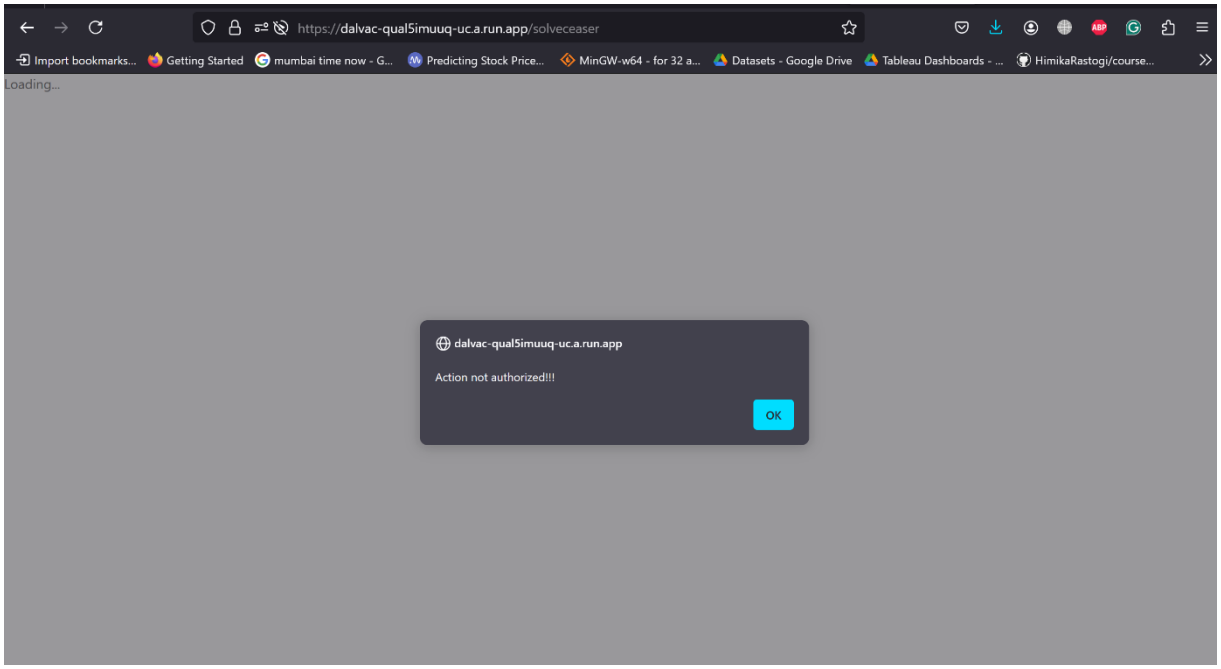

Figure 57 User tries to navigate manually without signin

=> Now coming to the database table side, below are the database tables that we have created in AWS dynamoDB tables.

=> This table stores the users information which they provide for the authentication, such as name , email, security question and answer, etc.

userId (String)	email	email_verified	given_name	hasSecurityQuestion	securityAnswer	securityQuestion	sub	TopicArn	userRole
14c83468-2051-70...	mufakannitteu-8232@yopmail.c...	true	asdfg	true	Halifax	What is your birth pl...	14c83468-...		RegisteredUser
d46884e8-e071-7...	sapsehirka@gufum.com	true	sapsehirka	true	Halifax	City?	d46884e8-...	arn:aws:sns:...	RegisteredUser
94b8a418-40f1-70...	nicouppiffeddu-2824@yopmail.c...	true	Qwertyu	true	arijit	favourite singer	94b8a418-...		
04583498-4091-7...	seuttauttussoge-1872@yopmail....	true	asdfgh	true	Halifax	What is your favouri...	04583498-...		PropertyAgent
b4980478-3041-7...	loxaciv929@fuzitea.com	true	batman	true	Halifax	Which city you live in ?	b4980478-...	arn:aws:sns:...	RegisteredUser

Figure 58 User entries when the user creates and account using MFA

=> Also, we are storing the user logins such as the identity of user as well as the timestamp when they logged in .

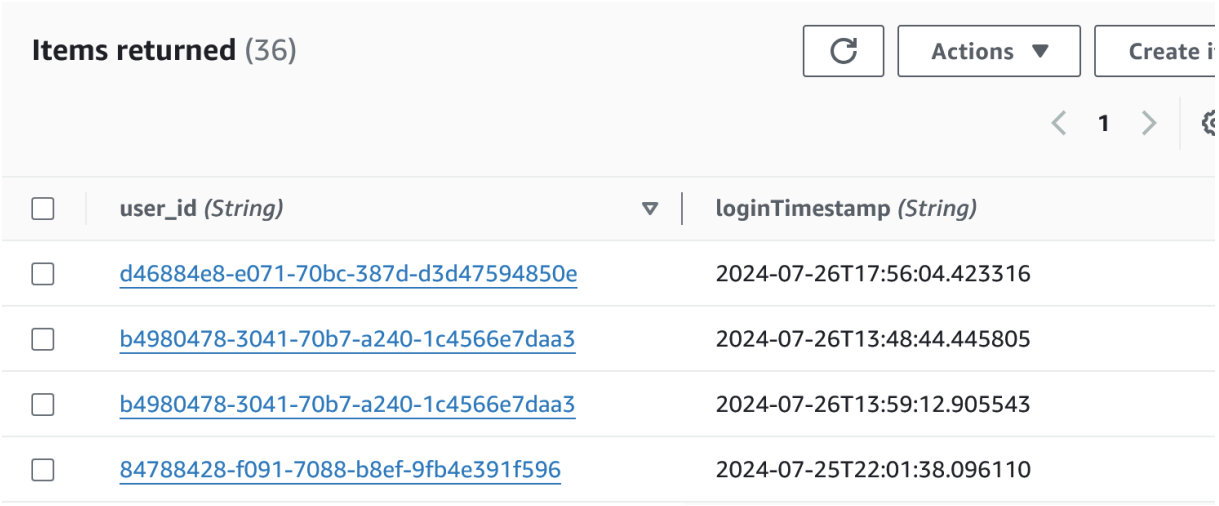

Figure 59 User login timestamp gets recorded every time they login to the application

General Tasks [Jaishankar Mohanraj]:

Tasks performed under general tasks for sprint 3:

- Frontend template and setup.

Frontend template and setup:

We used a frontend template that is available in [3]. The setup process included changing the sections contents to the needed contents and images. This also included creating routes to the different pages used for this project.

Technology used:

1. React: It is a frontend framework based on JavaScript, used for development of the frontend for the DalVacationHome.

Create Bookings for rooms:

Objectives:

Registered users need to create bookings for rooms that they want to rent out by creating a BaaS (Backend as a Service). The users should also be able to view the available types of rooms and their features.

Market Research:

A booking system generally uses an SQL database like mySQL but since the objective was to use serverless applications. We used AWS dynamoDB and lambda functions to access the tables. Hence, I have done my research on how to achieve this in a serverless architecture.

To ensure the users know all the different types of rooms and variants of each room, I have thoroughly gone through different hotel booking websites such as Hotel Halifax [4].

Pseudocode:

Getrooms:

- Scan the DynamoDB and get all the rooms.

Bookrooms:

- Scan the available rooms to check if the selected room exists.
- Scan the users table to confirm the user exists.
- Count the number of bookings present on the selected date for the selected room specifications.
- Check if the number of bookings is greater than available rooms, if yes then return error.
- If the number of rooms are lower than available, then create a booking document and return the booking ID.

Technology Stack:

1. AWS DynamoDB: This is used to store all data required for this task. This includes Room details, User details, bookings.
2. AWS lambda: The lambda functions perform backend tasks in the business logic layer such as fetching details from the database, creating new entries, and storing them in the database.
3. API gateway: API gateway is used to group related tasks together and provide a general URL for these tasks.
4. React: The javascript based framework is used to create the frontend for this task.

SQS [Jaishankar Mohanraj:

Objective: Set up booking queues to ensure race conditions are handled properly and the bookings are created.
Setup When the user requests for a booking a lambda function verifies the user details and the room details from the database, if this information checks out then the lambda function publishes a message to the booking verification queue which triggers a lambda function to check if the room is available on the date that the user has requested, if the room is available then a message is sent to the booking confirmation queue which updates the booking table and calls the send notification lambda function with the details.

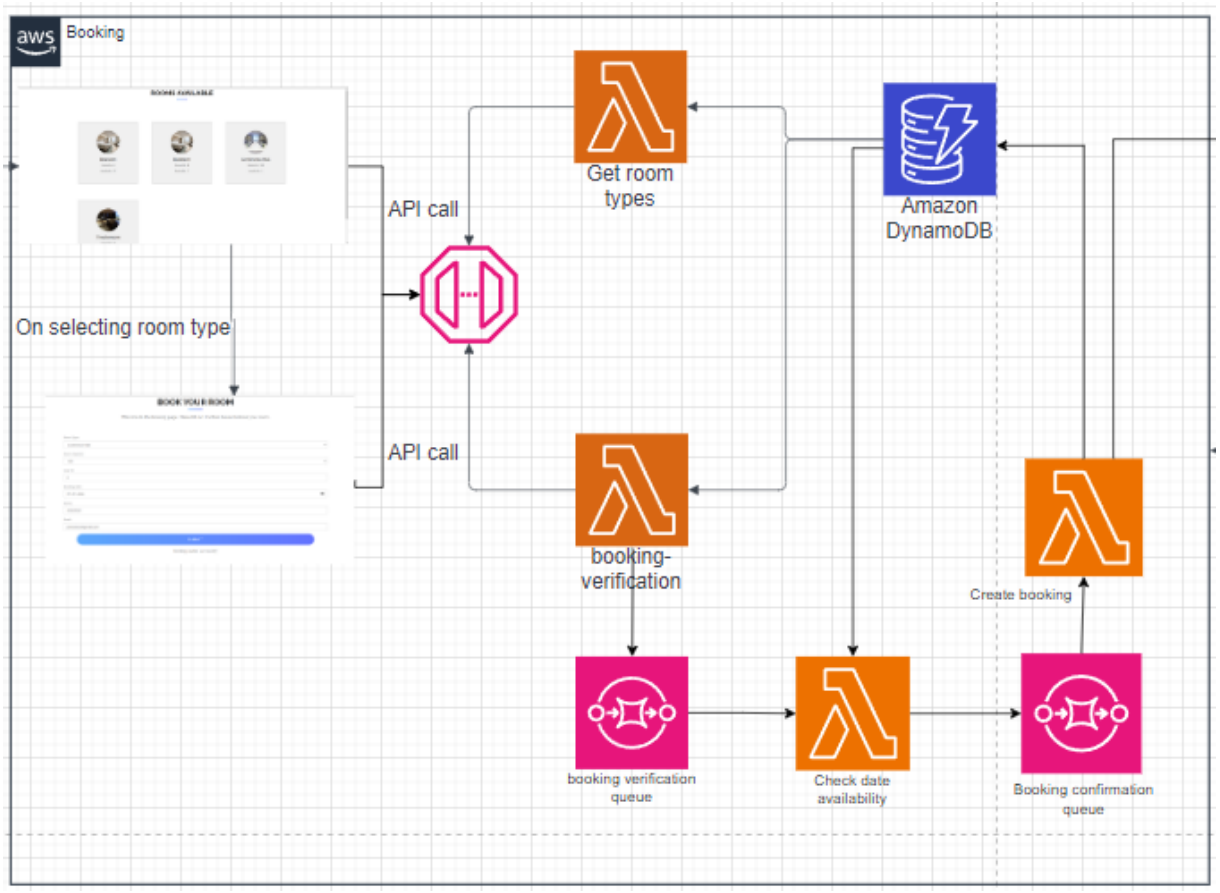

Figure 60: backend architecture for bookings

Notifications [Jaishankar Mohanraj]:

Objective: Set up AWS SNS to send notifications to users for various tasks performed by the user.

Notification instances:

1. On creating a new user, a subscription confirmation email is sent to the user.
2. When the user loges into the website an email is sent to the user along with the timestamp.
3. When the user initiates a booking, an email is sent along with the booking reference number to the user if the booking is successful else, an email is sent stating the reason for booking failure.
4. When the user requests to chat with an agent an email is sent with the link that the user can use to chat with an agent.

Setup: When the user table is updated with a new user, a lambda function is triggered that creates a new SNS topic with the userID and Updates the Record with the generated topic ARN. This

topic ARN is then used to send notifications through an endpoint that triggers a lambda. This setup allows the code for sending notifications to be reused.

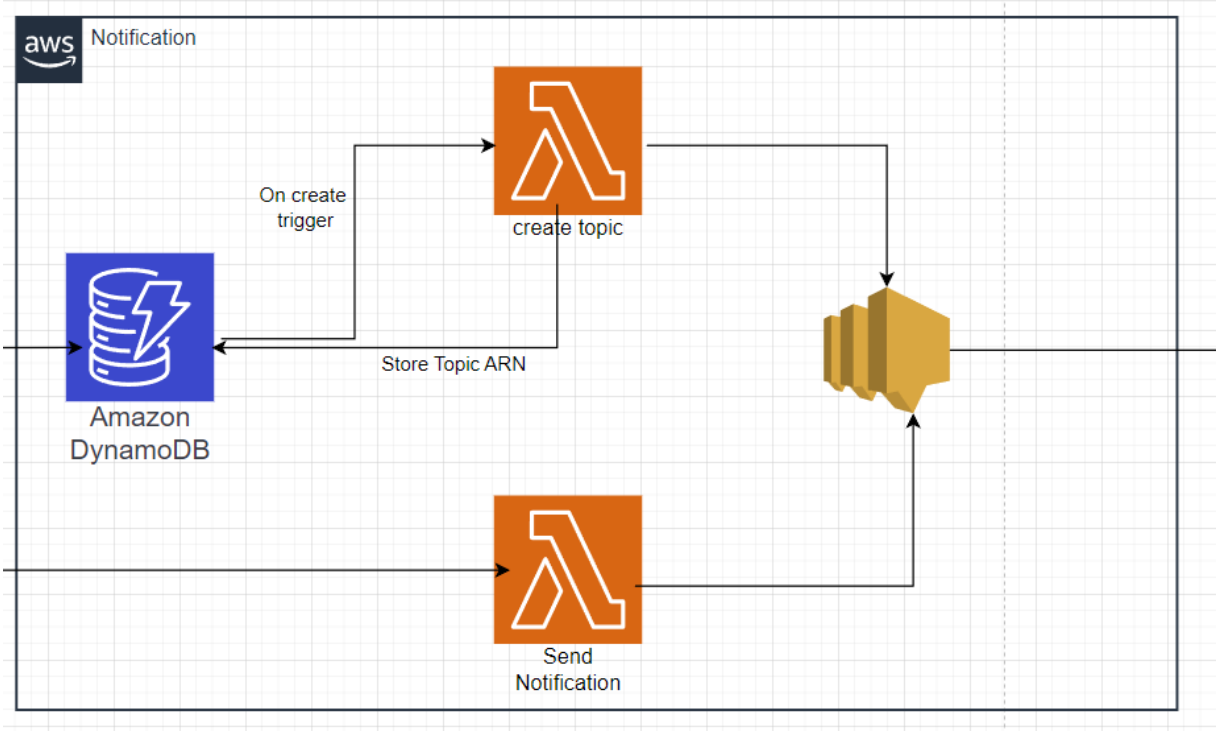

Figure 61:SNS Architecture

Message Passing Module [Jaishankar Mohanraj]:

Objective: A user can submit queries regarding their booking and will be able to get into a live chat with an agent that can address the users concerns.

Market Research: I conducted research in ways to pass information between multiple cloud providers to implement a multicloud approach, how to implement synchronous chat on a website between 2 entities.

To create a chat system effectively and accurately I investigated popular messaging services such as WhatsApp and Facebook messenger. The research aimed to understand the issues with real time chat and how it has been mitigated.

Technology Stack:

1. AWS Lambda: This is used to select a random agent from the pool and communicate with GCP.

2. Google cloud function: This is used to receive data from lambda, publish to pub/sub and store data to firestore.
3. Google Pub/Sub: This is used to ensure that the order of requests is maintained.
4. Google Firestore: This is Used to store the conversations between the agent and the user.
5. Google Firebase Messaging SDK: Used to create the live chat between user and agent.
6. React: The Javascript based framework to create the frontend for this task.

Backend architecture:

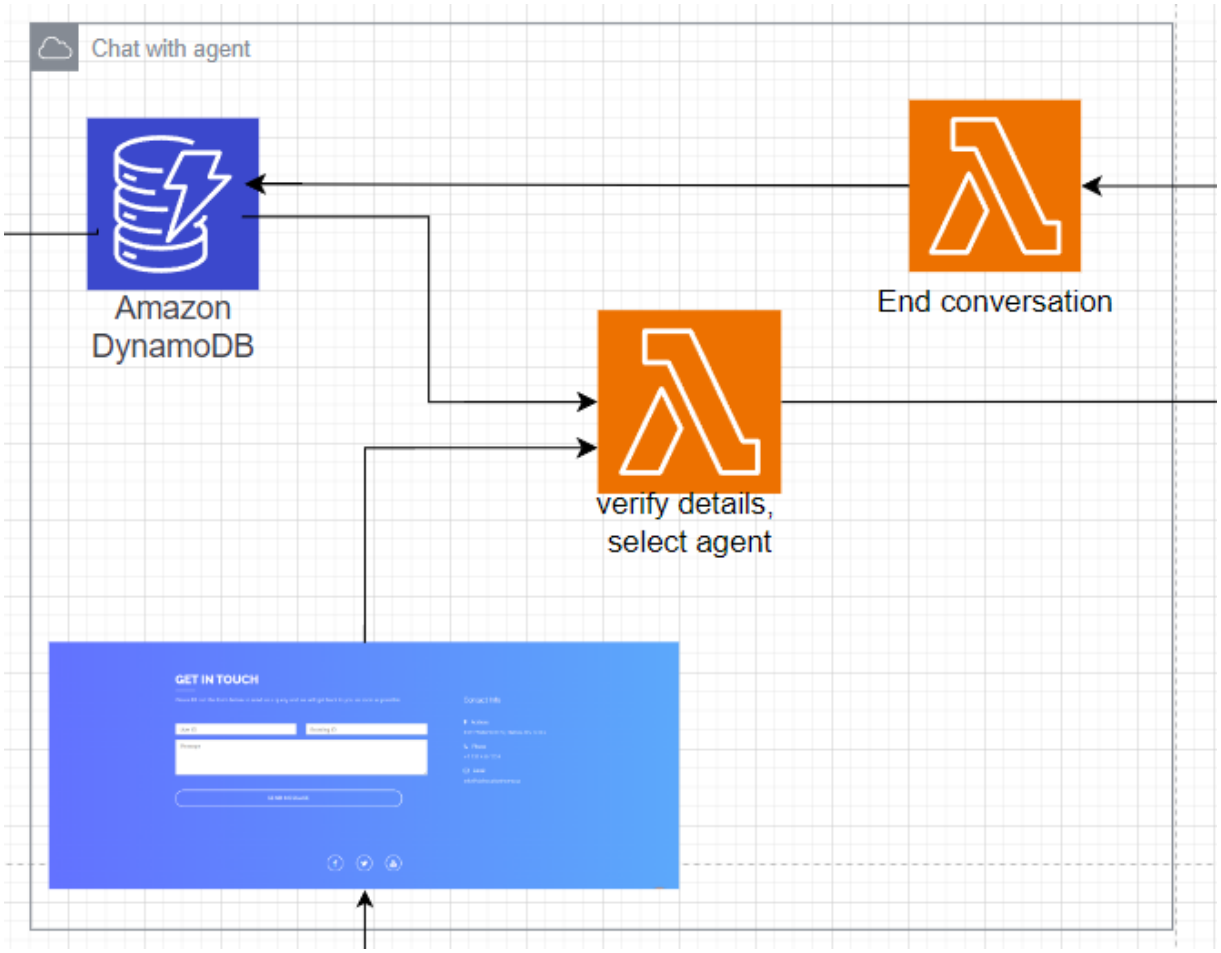

Figure 62: AWS part of the backend for live chat

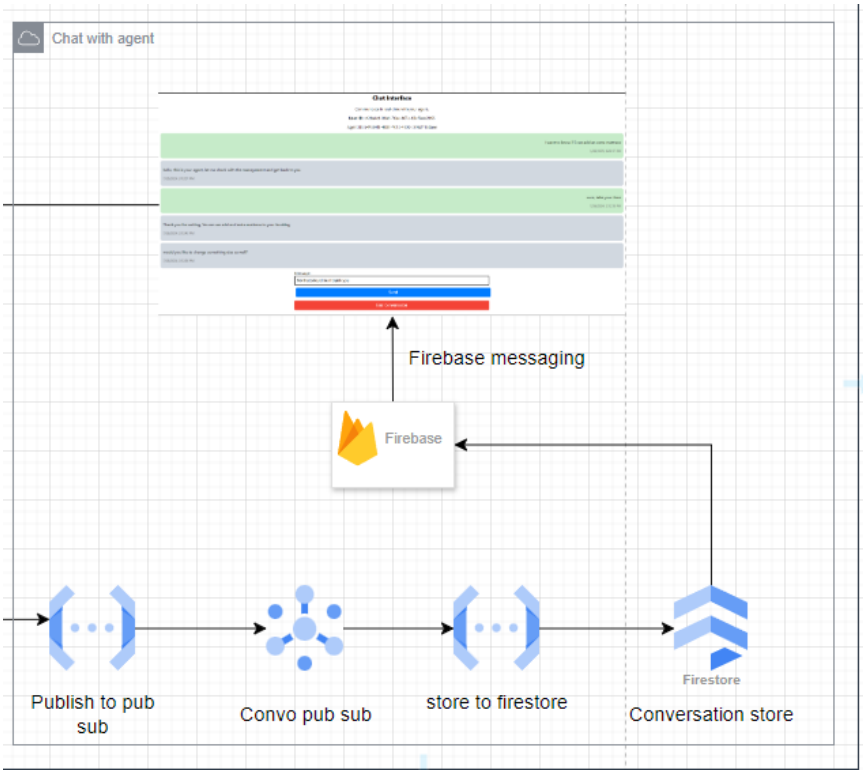

Figure 63: GCP part of the backend for live chat

Individual contribution [Jaishankar Mohanraj]:

My Responsibilities: I was assigned with the task to create a base frontend for the application and perform the base tasks of the application such as viewing rooms and creating bookings. I also worked on notifications and chat with agent.

Home-Page:

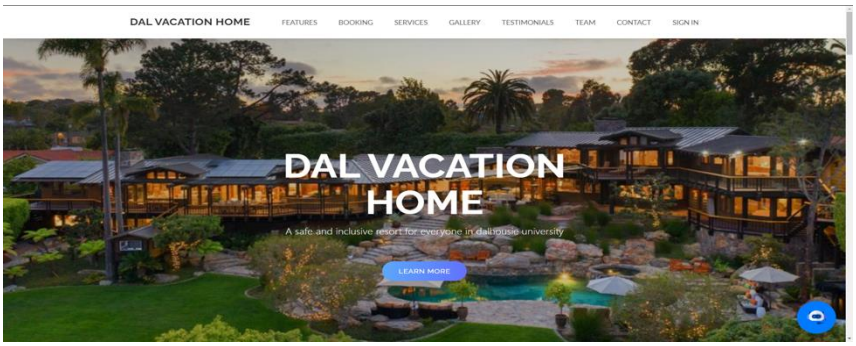

Figure 64: Landing page of the website

Features of the resort and bookings:

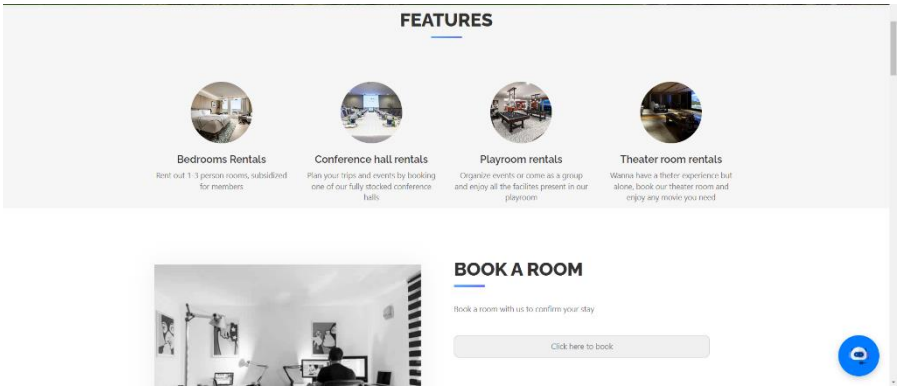

Figure 65: Features of the resort

Room details page:

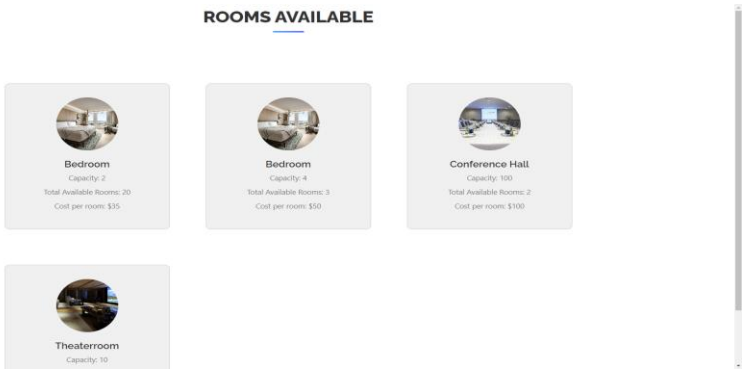

Figure 66: view rooms page

Bookings page and test cases:

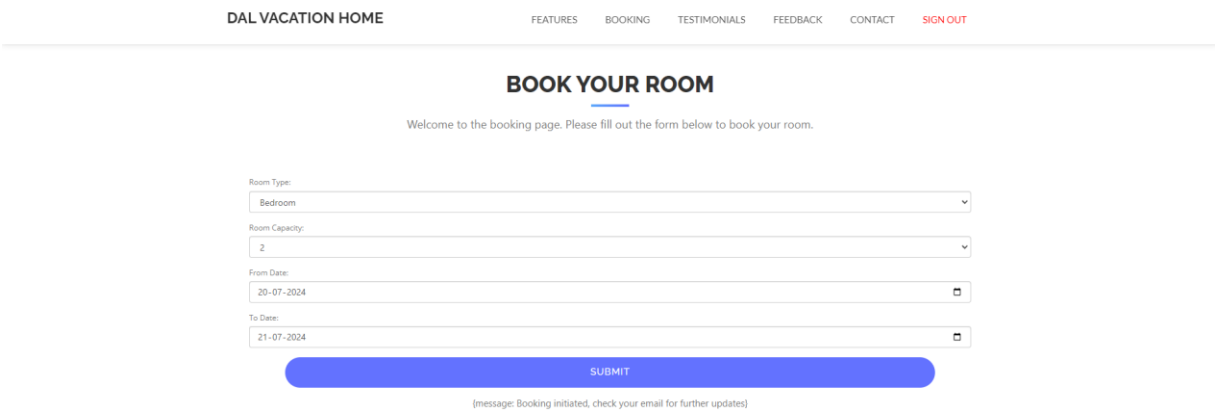

Figure 67: Booking page.

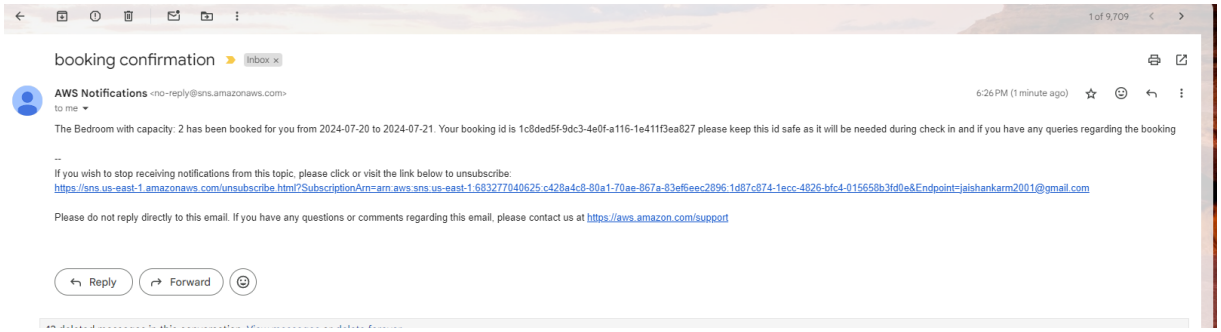

Figure 68: booking confirmation email.

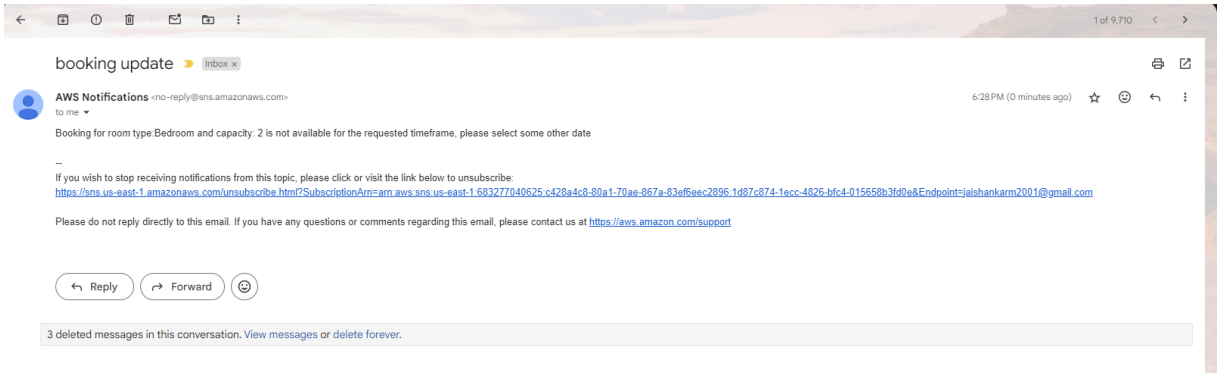

Figure 69: Booking unavailable email.

Message passing test cases:

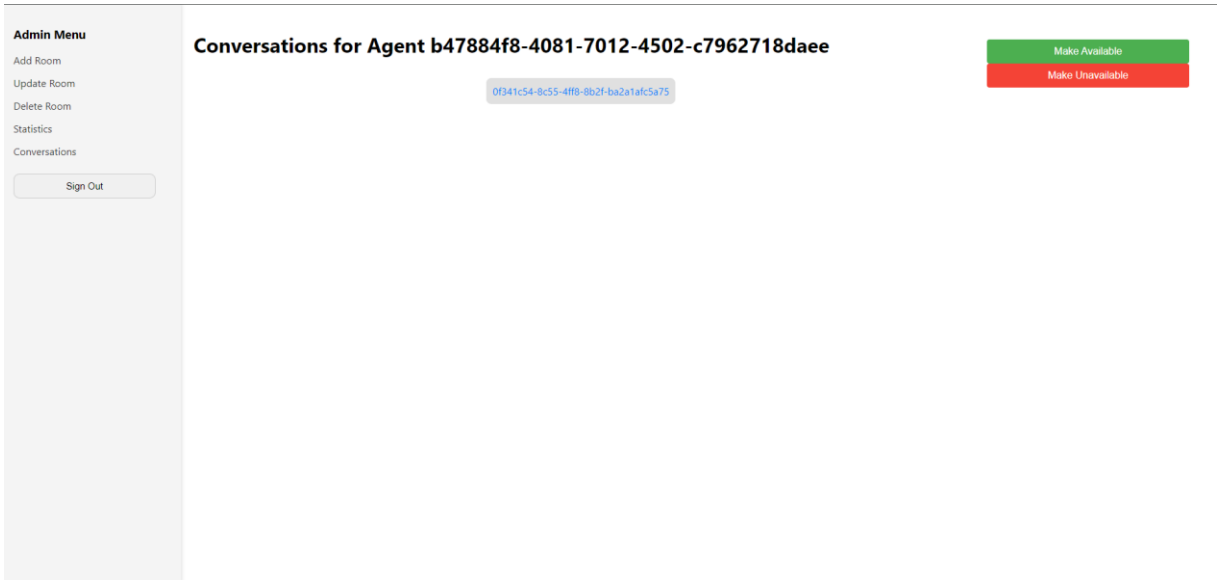

Figure 70: Agent side active conversation list

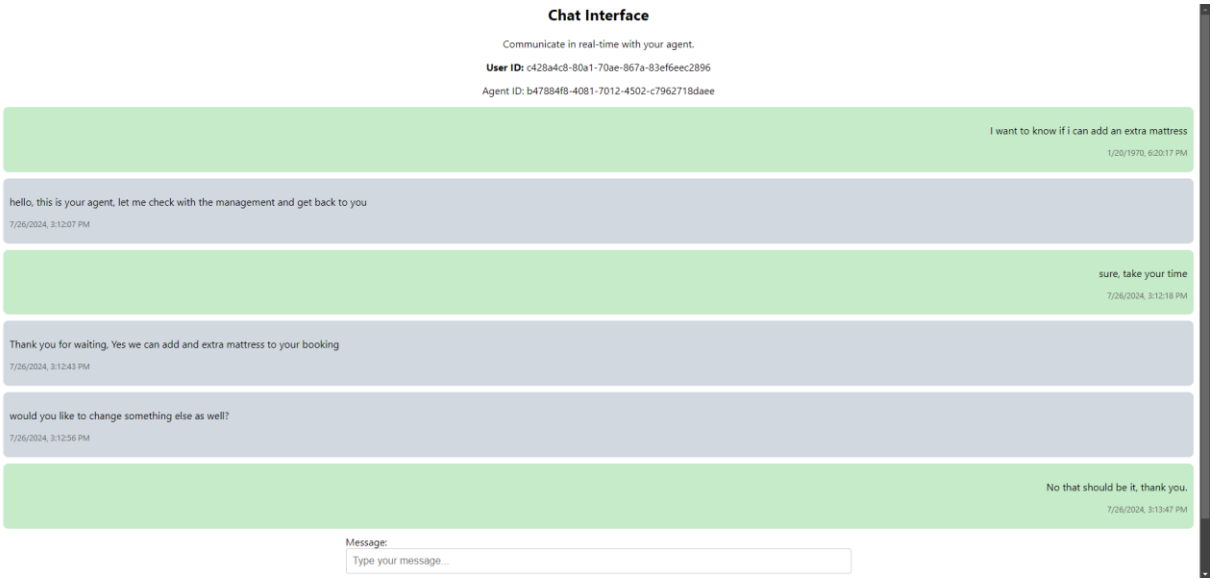

Figure 71: Chat Page

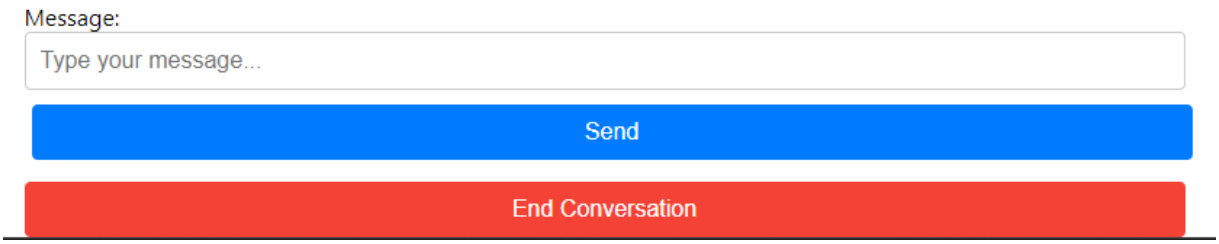

Figure 72: End conversation button

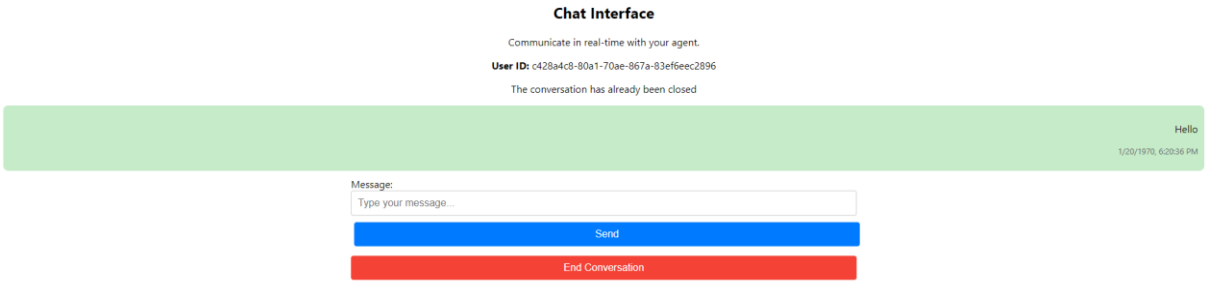

Figure 73: Conversation has already been closed message

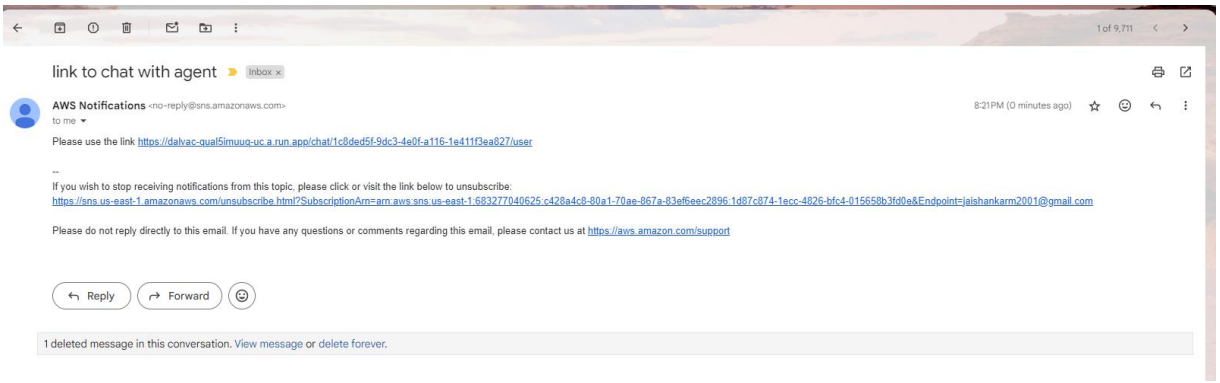

Figure 74: Email sent to user with link for live chat with agent.

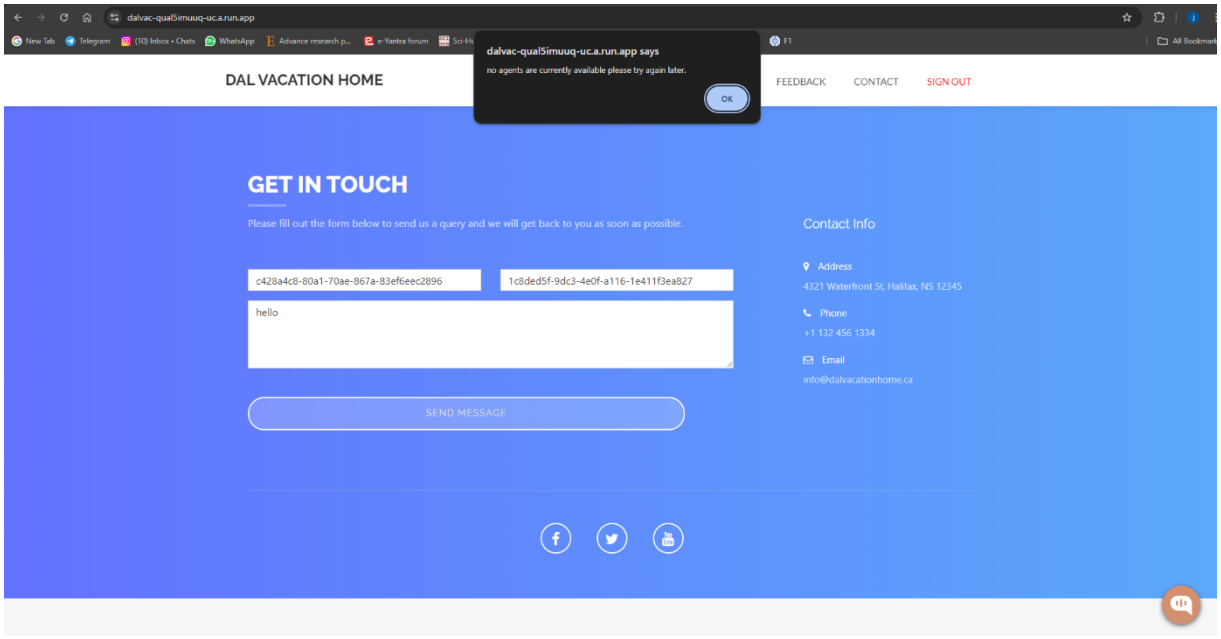

Figure 75: No available agents now message

Login notification:

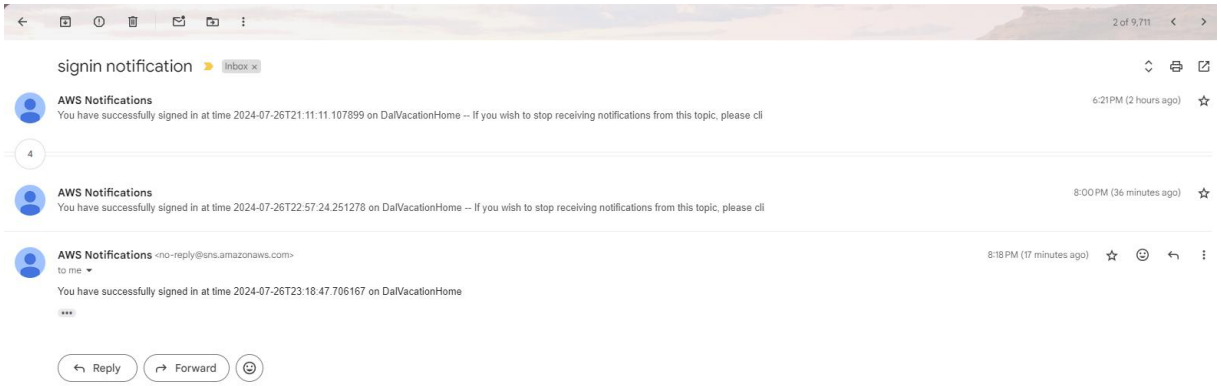

Figure 76: login notifications

GitLab issue board:

As shown below the 3 issues for this task have been completed and merged to main.

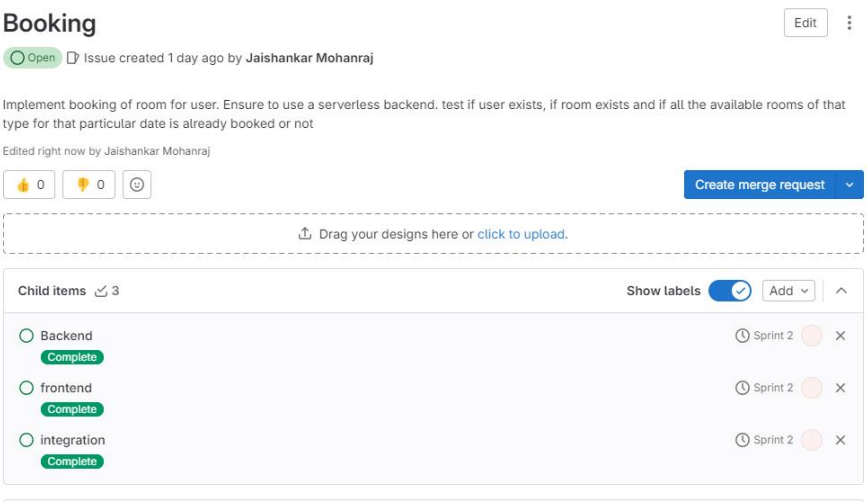

Figure 77: Booking issue.

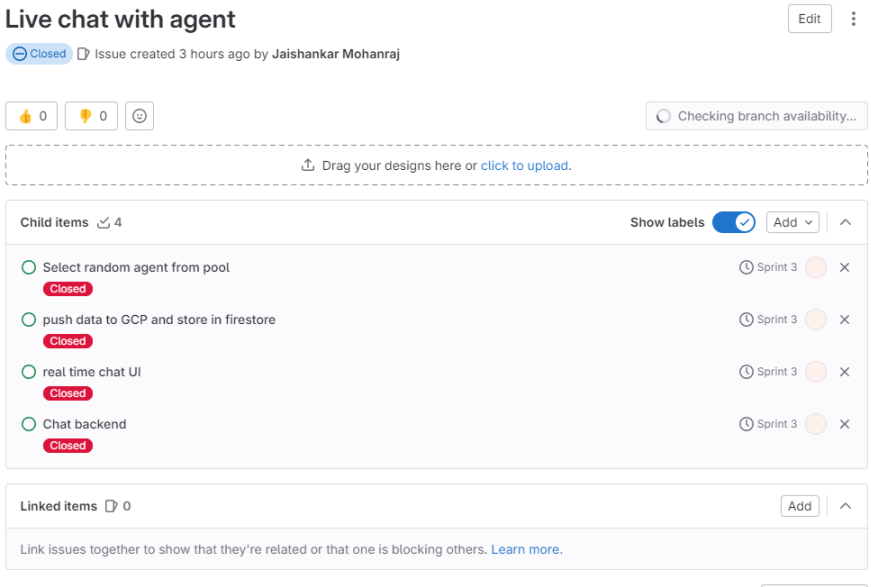

Figure 78: Chat module

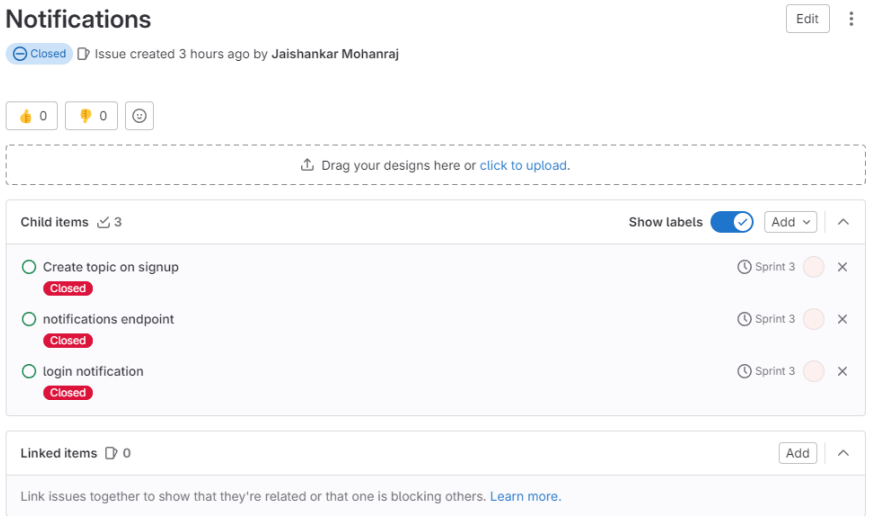

Figure 79: Notifications

Data Analysis and Visualization: [MD Samshad Rahman]

I was responsible for implementing a serverless architecture for data analysis and visualization. As a result, I implemented a feedback feature and admin dashboard where admins can check user's login statistics.

Objectives:

- Non-registered or not logged-in users will be able to read all the feedback already submitted, along with their polarity, in a tabular format.
- Logged-in users can submit feedback. Additionally, they can check their submitted feedbacks and delete if they want.
- Logged-in admin can see a dashboard with users login statistics along with the total user count.

Market research:
I conducted research on effective strategies for decoupling system architectures to enhance performance, scalability, and maintainability. Based on my findings, I have chosen to implement the following approaches:

Function-Specific Tasks: Decoupling systems into smaller, single-responsibility functions increases modularity and simplifies debugging. So, I divided the entire system into smaller tasks, ensuring each function is responsible for only one task. This approach allows for better scalability and easier maintenance.

Handler Functions for Task Management: Utilizing handler functions to manage tasks allows for more organized and efficient processing of requests. Handler functions were employed to streamline the task management process, ensuring tasks are executed efficiently and effectively.

Dashboard Update with Lambda and EventBridge: Regularly updating dashboard data is crucial for real-time monitoring and reporting. Using scheduled tasks ensures data is consistently refreshed without manual intervention. A Lambda function with a scheduler using EventBridge was written to update the dashboard data every 5 minutes (or as specified). This approach ensures that the dashboard remains current and provides accurate insights.

System Architecture for Data analysis (Feedback):

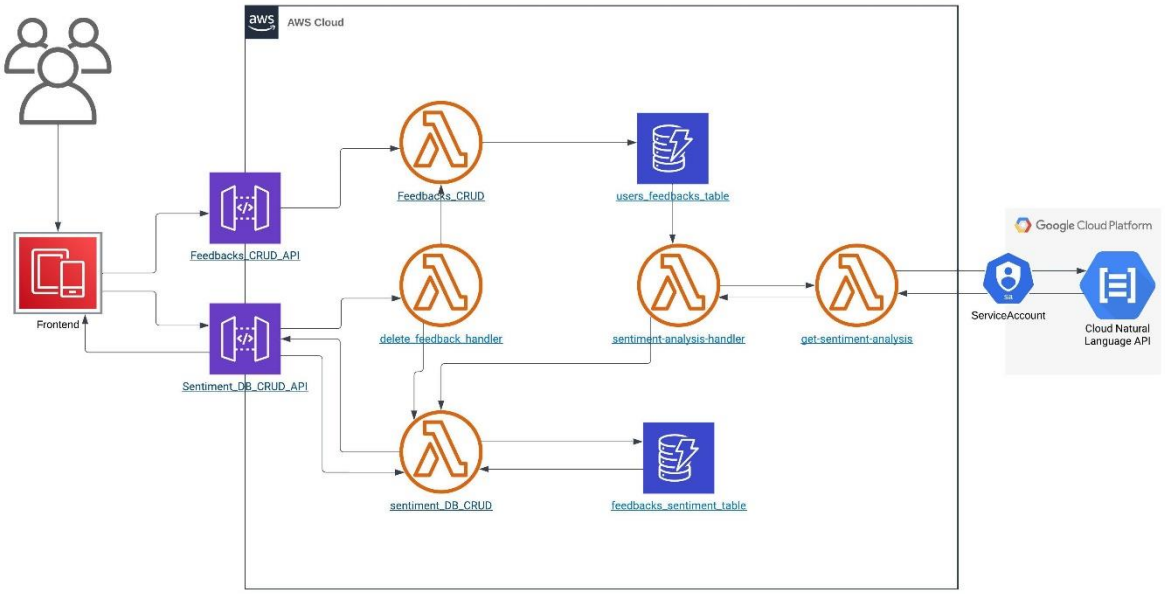

Figure 80: System architecture for feedback

Explanation:

I have worked on a serverless solution for processing feedback submission/retrieval from database and analyze the feedback with sentiment analysis. Writing a brief explanation of the architecture.

1. Frontend Interaction:

- Users submit feedback through the front-end application (React App).
- The frontend makes HTTP requests to the backend APIs.
- Users can fetch/see all the feedbacks with their polarity in a tabular format.

2. Backend APIs:

- Feedbacks_CRUD_API: This API handles inserting, reading, and deleting feedback entries inside users_feedbacks_table (DynamoDB table).
- Sentiment_DB_CRUD_API: This API handles inserting, reading, and deleting feedback entries inside feedbacks_sentiment_table (DynamoDB table).

3. Lambda Functions:

Feedbacks_CRUD:

- When a user submits feedback, this Lambda function is triggered.
- It stores the feedback data into the users_feedbacks_table DynamoDB table.
- This insert will also trigger the sentiment-analysis-handler to analyze the sentiment of the new feedback.

delete_feedback_handler:

- This function is responsible for deleting feedback entries.
- When a delete request is received, it calls DELETE API with the feedback_id to both feedbacks_CRUD and sentiment_DB_CRUD to delete the feedback entries from users_feedbacks_table and feedbacks_sentiment_table.

sentiment-analysis-handler:

- This function is triggered when new feedback is created.
- It sends the feedback text to get-sentiment-analysis (Lambda) for sentiment analysis.
- It stores the analyzed sentiment data into the feedbacks_sentiment_table.

get-sentiment-analysis:

- This function gets the feedback text and sends the feedback text to the Google Cloud Natural Language API for sentiment analysis and returns the polarity and score.

4. DynamoDB Tables:

- users_feedbacks_table: Stores the raw feedback data from users.
- feedbacks_sentiment_table: Stores the sentiment analysis results of the feedback along with feedback data.

Test cases:

Test case 1:

Non-logged-in users can see all the existing feedbacks in a tabular format.

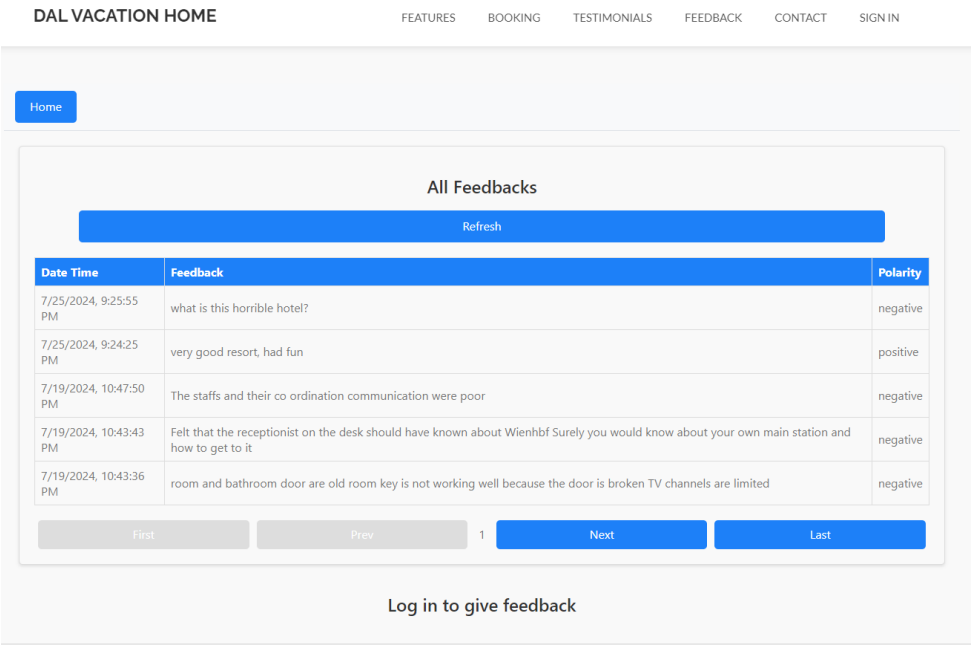

Figure 81 All existing feedbacks with polarity

Test case 2:
Logged-in users can see their own feedback with polarity and a delete option.

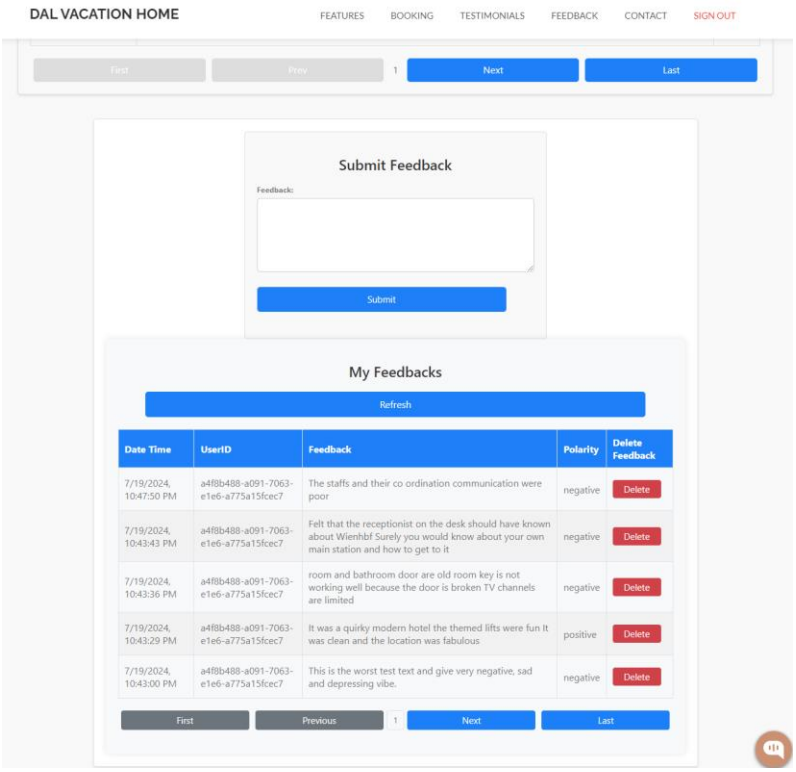

Figure 82 Logged-in user's own feedbacks with delete button

Test case 3:
Logged-in users can submit feedback.

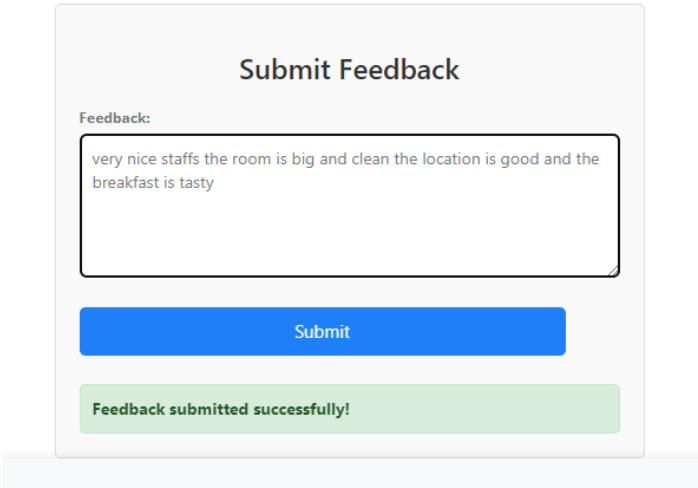

Figure 83 Submit feedback

Test case 4:
Logged-in users can delete feedback.

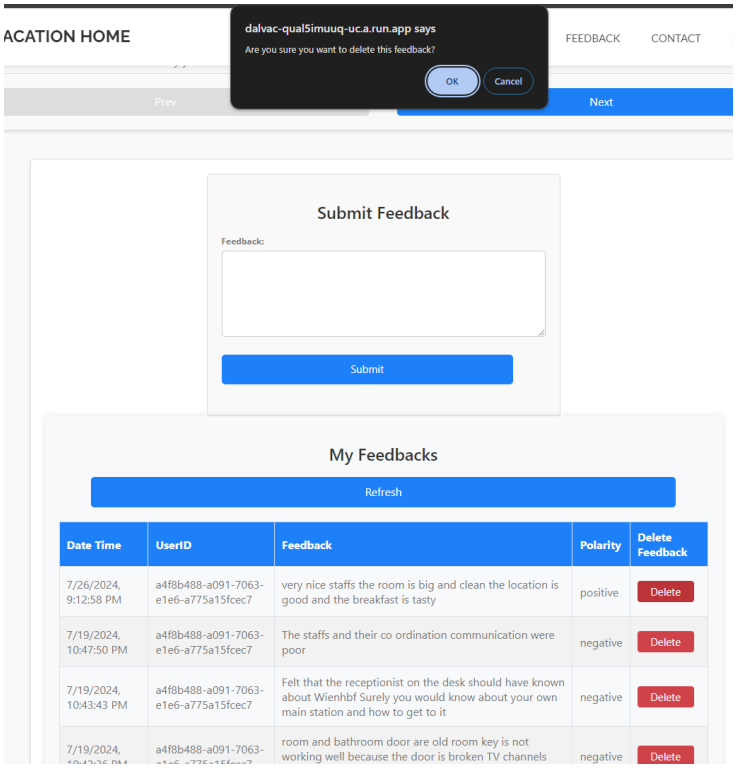

Figure 84 Delete feedback

System Architecture for Data visualization (Admin Dashboard):

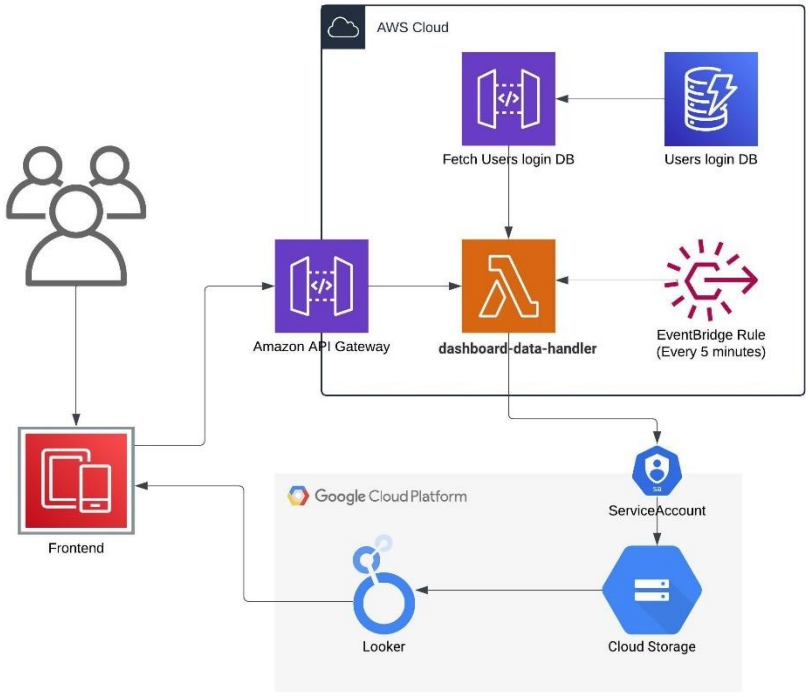

Figure 85 System Architecture for Data visualization (Admin Dashboard)

Explanation:

1. **User Interaction**: Users interact with the React frontend, which renders the dashboard using an iframe.
2. **API Gateway Trigger**: When an admin logs in system sends a GET call to dashboard-data-handler function to trigger and update the dashboard data.
3. **Dashboard Data Handler**: This function handles the dashboard data and has two key connections:
   - **EventBridge Rule**: Triggers every 5 minutes to update data automatically.
   - **Google Cloud Platform**: Dumps the user login data as a csv for data analysis to the Cloud Storage.
4. **Looker Studio**: Reads the csv file from Cloud Storage and renders the dashboard for admins. By default, the Looker studio updates the data every 15 minutes.

Test cases:

Test case 1:

Dashboard renders on the front end for admins.

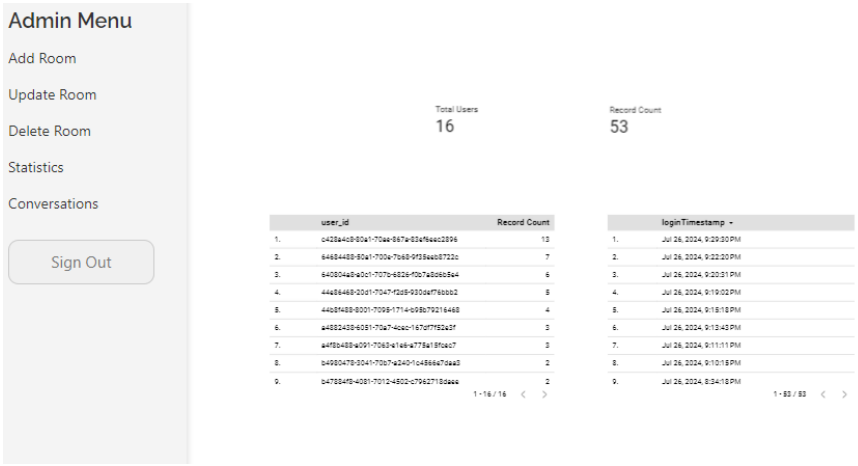

Figure 86 Admin Dashboard

Test case 2:
When an admin logs in manually lambda gets triggered to update csv using API Gateway.

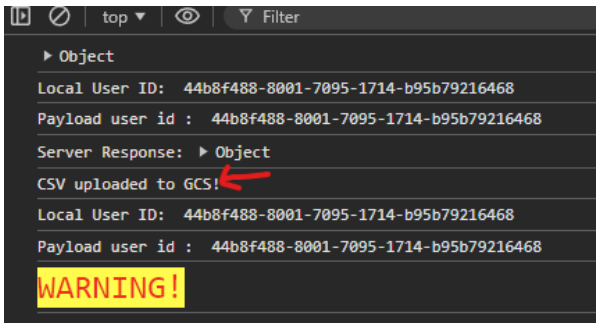

Figure 87 Manual lambda trigger to update CSV

Test case 3:
EventBridge triggers every 5 minutes to update the data.

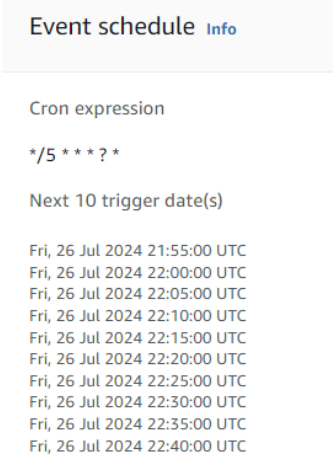

Figure 88 Cron job expression

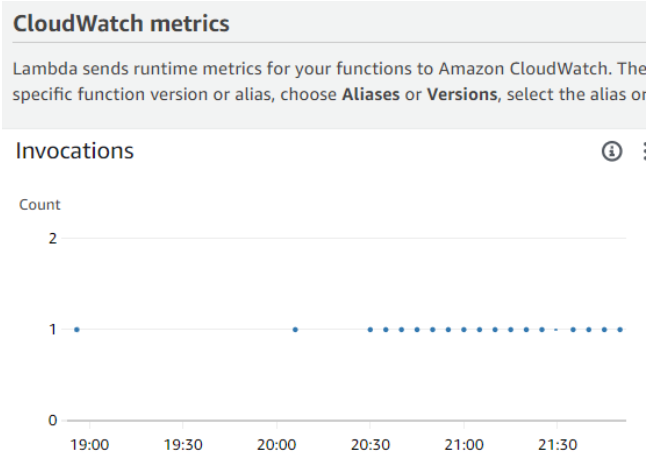

Figure 89 Lambda Invocation every 5 minutes

Deployment:

IaC:

We researched cloudRun to deploy our application on.
We created a terraform IaC script to spin up our cloudRun resource using this script and deployed our containerized application on GCP. The docker image was deployed on dockerhub. This image is pulled in our terraform script.

Figure 90: Terraform IaC for frontend

Deployment screenshots:

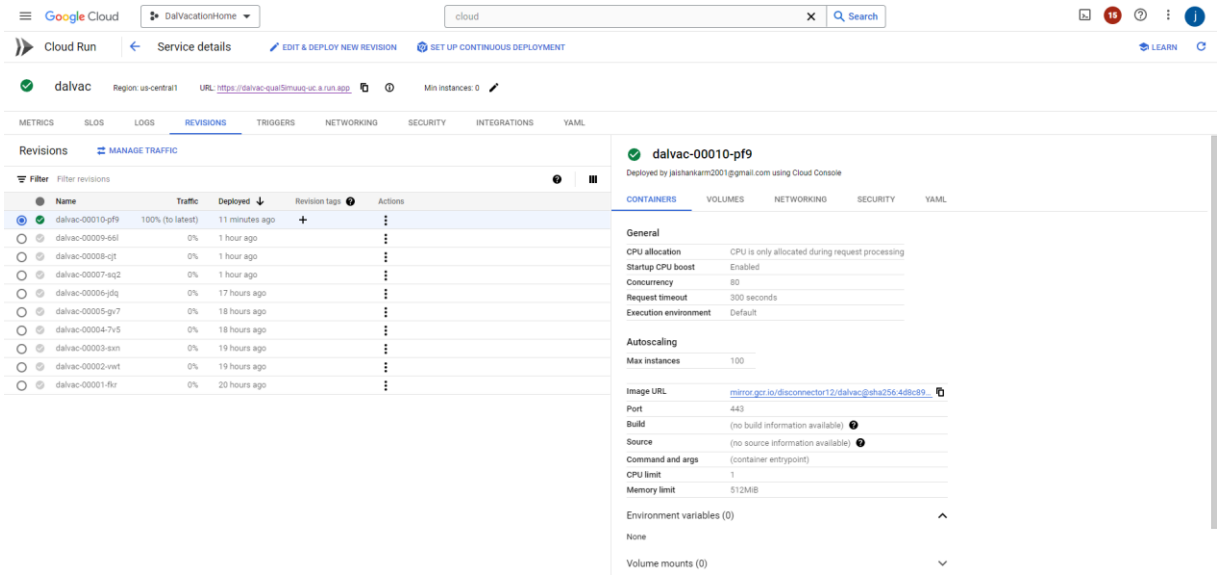

Figure 91: Frontend has been deployed using google cloud run

proj > Code > serverless_group1 > frontend > deploy.tf

```
provider "google" {
  project = "dalvacationhome-427921" #add your project name
  region  = " us-central1" #add your region name
}

resource "google_cloud_run_service" "default" {
  name     = "newtrial" #add cloud run service name
  location = "us-central1" #add location

  template {
    spec {
      containers {
        image = "docker.io/disconnector12/dalvac:latest" #replace with yours Jaishankar

        ports {
          container_port = 443
        }
      }
    }
  }

  traffic {
    percent         = 100
    latest_revision = true
  }
}

resource "google_cloud_run_service_iam_member" "noauth" {
  service    = google_cloud_run_service.default.name
  location   = google_cloud_run_service.default.location
  role       = "roles/run.invoker"
  member     = "allUsers"
}

output "cloud_run_url" {
  value = google_cloud_run_service.default.status[0].url
}

```

Figure 92: terraform script to create new deployment

proj > Code > serverless_group1 > frontend > Dockerfile

```
# pull official base image
FROM node:13.12.0-alpine

# set working directory
WORKDIR /frontend

# add `/app/node_modules/.bin` to $PATH
ENV PATH /fronend/node_modules/.bin:$PATH

# install app dependencies
COPY package.json ./
COPY package-lock.json ./
RUN npm install --silent
RUN npm install react-scripts@3.4.1 -g --silent

# add app
COPY . ./

# Expose the port the app runs on
EXPOSE 3000

# Define the environment variable for the port
ENV PORT 3000

# start app
CMD ["npm", "start"]
```

Figure 93: Dockerfile to create image

Final Architecture Diagram:

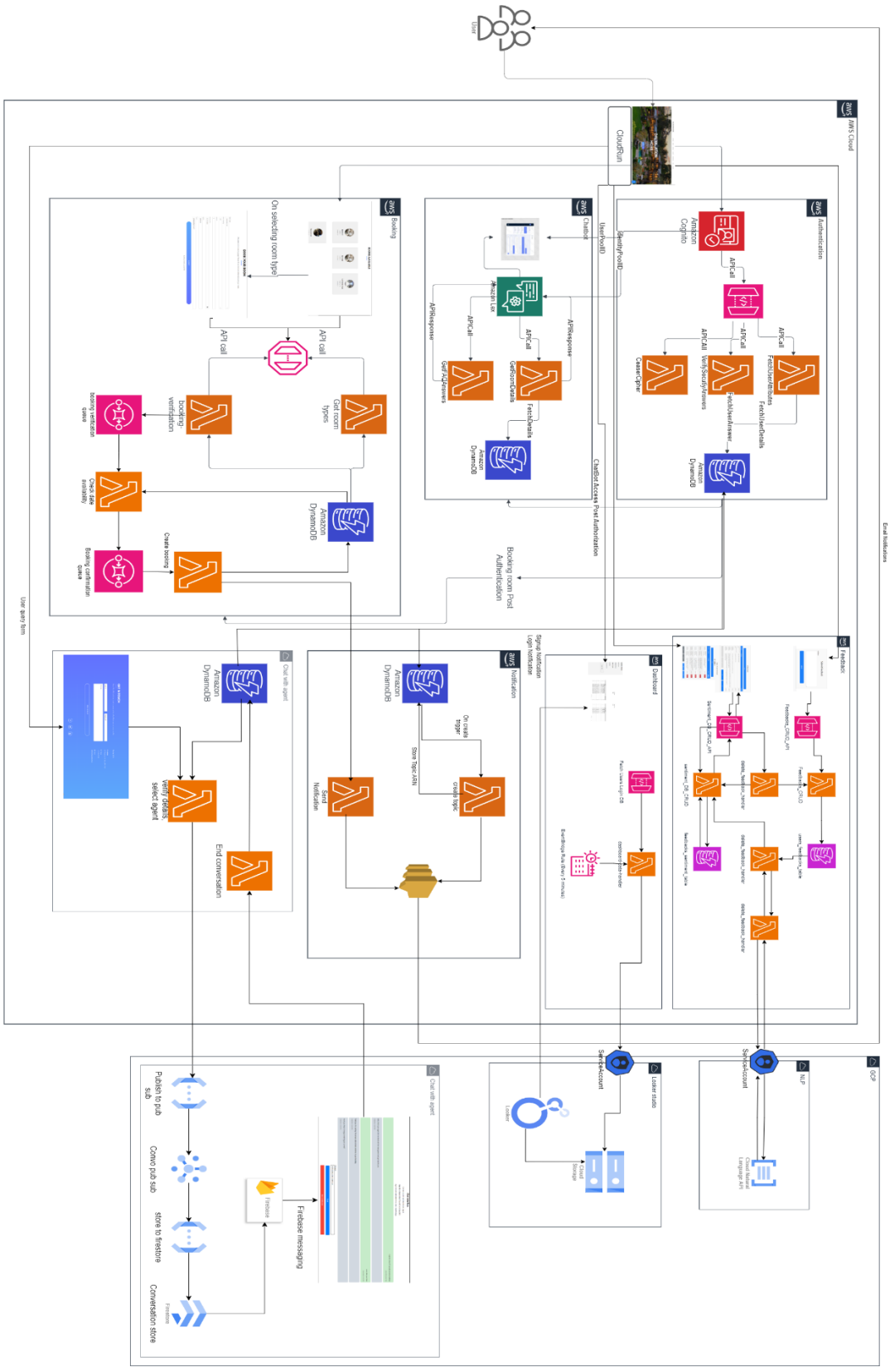

Figure 94: Final architecture diagram

Meeting Logs:

Agenda	Outcome	Link
Discussion about individual progress and further steps for integration.	Decided how to integrate the different modules with the frontend and discussion about IAC.	https://dalu-my.sharepoint.com/:v:/g/personal/ks328392_dal_ca/EUePkxoBqLVAieoSbHM9kbkB6BSnosPnl4qcB-56xK-dbQ?referrer=Teams.TEAMS-ELECTRON&referrerScenario=MeetingChicletGetLink.view.view
Creation of sprint 2 document and creation of system architecture	Created the sprint 2 document	https://dalu-my.sharepoint.com/:v:/g/personal/lv455130_dal_ca/Ebx4seKQ3_ZEmRBbWEmyEisBMxvr0gvKG3qJGV_Sb05VEQ?referrer=Teams.TEAMS-ELECTRON&referrerScenario=MeetingChicletGetLink.view.view
Complete creation of sprint 2 document and system architecture	Completed the bookkeeping tasks	https://dalu-my.sharepoint.com/:v:/g/personal/ks328392_dal_ca/EW51Jx6tIj5Pv1rofRmSWr4BaSBLYN7YOd8e2yx-pUWkXw?referrer=Teams.TEAMS-ELECTRON&referrerScenario=MeetingChicletGetLink.view.view
Meeting for knowing Updates	Knowledge on what everyone is working on	https://dalu-my.sharepoint.com/:v:/g/personal/lv455130_dal_ca/Ebx4seKQ3_ZEmRBbWEmyEisBMxvr0gvKG3qJGV_Sb05VEQ?referrer=Teams.TEAMS-ELECTRON&referrerScenario=MeetingChicletGetLink.view.view

References:

[1] Expedia, “Help Center,” *Expedia*. [Online]. Available : https://Https://www.expedia.ca/helpcenter/ [Accessed Jul. 04, 2024].

[2] “CRA Services”, *Canada Revenue Agency* [Online], Available: https://www.canada.ca/en/revenue-agency/services/e-services/cra-login-services.html [Accessed : Jun. 30, 2024].

[3] I. Kattan, ”React Landing Page Template”, *Github* [Online], Available : https://github.com/issaafalkattan/React-Landing-Page-Template [Accessed: Jun. 28, 2024]

[4] “Getting Started with user pools”, *Amazon Web Services* [Online]. Available: https://docs.aws.amazon.com/cognito/latest/developerguide/getting-started-user-pools.html [Accessed : Jul. 01, 2024]

[5] “Welcome to Hotel Halifax”, *Hotel Halifax* [Online] Available: https://www.hotelhalifax.ca/ [Accessed: Jul. 01, 2024]

[6] A. Hitchon, “AWS_RestLambdaAPI,” *GitHub* [Online], Available: https://github.com/hitchon1/AWS_RestLambdaAPI/blob/main/lambda_function.py [Accessed Jul. 2, 2024].

[7] “Intelligent diagramming,” *Lucidchart*. [Online]. Available: https://www.lucidchart.com/pages/ [Accessed: 30-May-2024].