



---

# CSCI 5410

---

SERVERLESS DATA PROCESSING



**PROJECT REPORT  
GROUP 4**

## Table of Contents

<b>Table of Figures.....</b>	<b>2</b>
<b>1. Gitlab Repository Link.....</b>	<b>5</b>
<b>2. Hosted Web Application Link .....</b>	<b>5</b>
<b>3. Implementation Details.....</b>	<b>5</b>
<b>3.1 User Management Module [1] .....</b>	<b>5</b>
<b>3.2 User Authentication Module.....</b>	<b>6</b>
<b>3.3 Online Support Module [1] .....</b>	<b>6</b>
<b>3.4 Message Passing Module [1].....</b>	<b>7</b>
<b>3.5 Machine Learning Module [1] .....</b>	<b>8</b>
<b>3.6 Web Application Building and Hosting [1].....</b>	<b>9</b>
<b>3.7 Other essential modules-testing, report generation and visualizations.....</b>	<b>9</b>
<b>5. Pseudo Code .....</b>	<b>11</b>
<b>5.1 User Management .....</b>	<b>11</b>
<b>5.2 User Authentication (3 Stage Verification).....</b>	<b>11</b>
<b>5.4 Message Passing Module .....</b>	<b>13</b>
<b>Tour Package Recommender:.....</b>	<b>13</b>
<b>Sentiment Analysis: .....</b>	<b>14</b>
<b>5.6 Web Application Building and Hosting .....</b>	<b>14</b>
<b>Room Service.....</b>	<b>14</b>
<b>Kitchen Service (Get Food):.....</b>	<b>15</b>
<b>Kitchen Service (OrderFood): .....</b>	<b>15</b>
<b>Tour Service .....</b>	<b>15</b>
<b>5.7 Other Essential Module (Report/Visualization).....</b>	<b>16</b>
<b>6. Flowchart/Activity Diagram [1].....</b>	<b>17</b>
<b>7. Individual and Team Contribution .....</b>	<b>21</b>
<b>8. Meeting Logs .....</b>	<b>22</b>
<b>9. Test Cases and Evidence of Testing.....</b>	<b>26</b>
<b>9.1 User Login and Registration .....</b>	<b>26</b>
<b>9.4 Message Passing .....</b>	<b>36</b>
<b>9.5 Machine Learning.....</b>	<b>37</b>
<b>9.6 Web Application Building and Hosting .....</b>	<b>41</b>

9.7 Other Essential Module (Report/Visualization) .....	64
10. Limitations.....	67
11. References .....	68

## Table of Figures

Figure 1: Cloud Architecture Diagram [1][2].....	10
Figure 2 Unregistered Customer Roadmap [3][4][5] .....	17
Figure 3 Registered Customer Roadmap [3][4][5] .....	17
Figure 4 Admin Roadmap [3][4][5] .....	18
Figure 5 Flow diagram[2] of services like Amazon Lex[10] and Lambda function[8] interacting with each other for unauthorised users.....	19
Figure 6 Flow diagram[2] of services like Amazon Lex[10], Lambda function[8], and DynamoDB[7] and interacting with each other for authorised users.....	19
Figure 7 Meeting 2 Log .....	22
Figure 8 Meeting 3 Log.....	22
Figure 9 Meeting log 4 .....	23
Figure 10 Meeting log 5 .....	23
Figure 11 Meeting log 6 .....	24
Figure 12 Meeting log 7 .....	24
Figure 13 Login Authentication Failed Test Case .....	26
Figure 14 User Already Signed Up Test case .....	26
Figure 15 Password policy not matched .....	27
Figure 16 Caesar cipher failed.....	27
Figure 17 Screenshot of chatbot successfully deploying on the website.....	28
Figure 18 Screenshot of chatbot successfully deploying on the website.....	28
Figure 19 . Screenshot of chatbot performing unauthorized function: room details. ....	29
Figure 20 . Screenshot of chatbot performing unauthorized function: navigation/help. ....	29
Figure 21 Screenshot of chatbot performing unauthorized function: navigation/help .....	30
Figure 22 Screenshot of activation of authorized chatbot after login.....	31
Figure 23 Screenshot of chatbot performing authorized function: order food. .....	31
Figure 24 Screenshot of chatbot performing authorized function: order food with confirmation message. ....	32
Figure 25 Screenshot of chatbot performing authorized function: book room. ....	32
Figure 26 Screenshot of chatbot performing authorized function: book room with confirmation message. ....	33
Figure 27 User Notifications .....	36
Figure 28 Booking Confirmation Email .....	36
Figure 29 Notifications in Table .....	37
Figure 30 Tour Package Recommender Front End .....	37

Figure 31 Tour Package Recommender Front End .....	38
Figure 32 Tour Package Recommender Results.....	38
Figure 33 Tour Package Recommender Cloud Function.....	39
Figure 34 Tour Package Recommended Vertex AI.....	39
Figure 35 Sentiment Analysis Front End .....	40
Figure 36 Sentiment Analysis Results in AWS DynamoDB.....	40
Figure 37 Sentiment Analysis Cloud Function.....	40
Figure 38 Room booking home page .....	41
Figure 39 : User enters the room booking details .....	41
Figure 40 Room booked successfully message .....	42
Figure 41 Validations for the fields in the room booking form .....	42
Figure 42 Validations for the fields in the room booking form .....	43
Figure 43 Room booking details in the Amazon DynamoDB table .....	43
Figure 44 Tour package home page for user .....	44
Figure 45 Tour package home page for user .....	44
Figure 46 Booking confirmation check for the user.....	45
Figure 47 Booking date confirmation for the user.....	45
Figure 48 Data inserted in Amazon DynamoDB table for Tour booking.....	46
Figure 49 Food Order home page .....	46
Figure 50 Order confirmation details form for the user .....	47
Figure 51 Validations in the order food form .....	47
Figure 52 Food order details stored in Amazon DynamoDB table .....	48
Figure 53 Lambda Functions for all the modules.....	48
Figure 54 API created in AWS for all the modules .....	49
Figure 55 API gateway trigger added for room booking backend lambda function.....	49
Figure 56 GET, POST, PUT and DELETE API requests for the routes .....	50
Figure 57 Integrations of the requests with AWS Lambda function.....	50
Figure 58 API gateway trigger for food order lambda function.....	51
Figure 59 Routes for the food order lambda function API's.....	51
Figure 60 Integrations of the requests with AWS Lambda function in food order .....	52
Figure 61 API gateway trigger for the Tour Booking lambda function .....	52
Figure 62 Routes for tour booking .....	53
Figure 63 Integrations of the requests with AWS Lambda function in tour booking .....	53
Figure 64 Caesar Cipher Lambda function and API gateway trigger .....	54
Figure 65 API details of Caesar cipher lambda function .....	54
Figure 66 Integrations of the requests with AWS Lambda function in caesar cipher .....	55
Figure 67 feedback lambda function .....	55
Figure 68 orderMyfood lambda function .....	56
Figure 69 available rooms lambda function .....	56
Figure 70 API gateway trigger for security answer lambda function .....	57
Figure 71 Integrations of the requests with security answer lambda function.....	58
Figure 72 Website navigation lambda function.....	58
Figure 73 room booking lambda function .....	59
Figure 74 Active tables in DynamoDB.....	59

Figure 75 Caesar Cipher table in DynamoDB .....	60
Figure 76 User feedback table in dynamodb .....	60
Figure 77 Food details table in dynamodb .....	61
Figure 78 Food order table in dynamodb .....	61
Figure 79 room details table in dymaodb.....	62
Figure 80 roomdetails table in dynamodb.....	62
Figure 81 security answer table in dynamodb.....	63
Figure 82 Tour booking details table in dynamodb .....	63
Figure 83 Room Bookings 1.....	64
Figure 84 Room Bookings 2.....	64
Figure 85 Food Orderings 1.....	65
Figure 86 Food Orderings 2.....	65
Figure 87 Tour Bookings 1.....	66
Figure 88 Tour Bookings 2.....	66

## 1. Gitlab Repository Link

[Group 4 GitLab Repository](#)

## 2. Hosted Web Application Link

[Serverless B&B](#)

## 3. Implementation Details

### 3.1 User Management Module [1]

- The main functionality of this module is to handle the user registration process along with dynamically assigning the User to the specific room booked by the user. The AWS Cognito [7] will be used for authentication, authorization, and user management. The users will be able to sign in using their username and password. After this step, the user will be asked to setup the security questions followed by the Caesar Cipher key. This key will be generated randomly, and the user is supposed to remember the key which will be useful at the time of login. The user pool and Identity pool will be used together to provide the essential services for the application. Furthermore, we will be also securely storing and maintaining the user details. AWS DynamoDB [8] service will be utilized to store the information of these users. The application initially will display the home page where the user will either log in to the application or can register themselves.
- If the user chooses to register themselves then the application will ask the user for details like Username, Phone No, Email ID (It will be a unique User ID), Password, Security Questions, and the Caesar Cipher Key for encryption of the text. There will be on-the-go validations done for the details entered by the user like for a valid email or for a strong password.
- User registration logic for Serverless B&B application:
  1. According to the requirements, when the user enters the username, email and other details, the next section will be followed by the Security Questions which the user has to remember at the time of login. Along with that, the final part which will be required for authentication is the Caesar cipher key which will be randomly generated by the system. The user is supposed to remember this key which the user will be using to encrypt the plain text that the system generated at the time of login. When the plain text is generated in the third part of authentication, then the user will encrypt the plain text with the randomly generated key and perform Caesar cipher on the text. If the entered details match with that in the database, then the user will be able to successfully login and access the application to order food, book rooms and even book for tours.

- Now, all the information entered by the User, such as the username, email and password, security questions and the Caesar cipher key along with the information regarding the room bookings, food orders and the tour bookings that are provided by the user is stored securely in the AWS DynamoDB [8].

### **3.2 User Authentication Module**

Users must successfully complete all three phases of authentication before being granted access to any of the services offered by the B&B application, and the user authentication logic uses a multi-cloud paradigm to enable multi-factor authentication for registered users.

#### **1. ID and Password Validation (AWS Cognito):**

The first step requests the user's login email and password, then checks it against the information in AWS Cognito.

#### **2. Question and Answer verification (AWS Lambda ):**

The second stage displays the user's chosen question from the two options they were given on registering and checks the response. The user-selected query and response are saved in AWS DynamoDb, and the API calls the Cloud Function to validate the outcomes before proceeding to the third phase of authentication.

#### **3. Caesar Cipher Validation (AWS DynamoDB ):**

The user will be required to utilise the key that was generated and given to them when they registered to decrypt the Caesar-ciphered text as the last step in the authentication process.

### **3.3 Online Support Module [1]**

Online assistance is a web-based technique for offering customer service. It is currently more popular than calling or going in person to communicate with businesses. Online support is a part of customer relationship management. The best feature of online support is the round-the-clock customer service it provides. The consumer can instantly get in touch with the agent if there is a problem or if they have any questions about the service or product.

The module's responsibility is to provide online help using Amazon Lex to both authenticated and unauthenticated users [10]. Lex will assist with developing clever comments to the recorded key words. To answer accurately and assist consumers in the best way possible, the service will offer a range of user contact scenarios.

As per the project specifications, two bots are created:

**1. BnbAuth:**

This bot will be for authorized users, i.e., activated after user has registered/login. This bot will help in booking room by taking all the details like email, arrival date, departure date, room numbers and finally gives a confirmation message of booking the room with message with all the details.

**2. BnbUnauth:**

This bot will be for unauthorized users, i.e., users who have not yet logged into the website portal. It will help in navigating through the website by giving the options like login, signup and more by giving direct links to the pages. Also, it will help to get details about any room like what kind of rooms the website offers.

For deploying the bot into the application, an amazon deployment document [16] was followed that involved creating Cloud Formation [3] stack using the available template. The formation had a snippet code which was added in the index.html for successful deployment of IU for chatbot.

### 3.4 Message Passing Module [1]

Giving authorized users access to the hotel management i.e. food ordering and room booking, and tour operators via this message passing module is helpful. Using the service, Pub/Sub, provided by GCP, we can build systems of event producers and customers, known as publishers and subscribers. Without considering how or when these events should be processed, publishers transmit events to the Pub/Sub service. Then, Pub/Sub distributes events to all the services that must respond to them. As a result, if employed as a messaging service, this service can be incredibly beneficial and effective.

In the Serverless B&B application, whenever the users use services such as room booking, food ordering or tour booking, the front-end, hosted on GCP CloudRun, will send a request to the Lambda function to store the details of the order in the DynamoDB table. Once the successful response comes to the application, it will send another request to the GCP Cloud Function which passes the message to the GCP Pub/Sub service. Whenever there's a message available in the Publisher queue, the Subscriber service would fetch the message from the queue and send it to the AWS Lambda function. This Lambda function will create a confirmation message and store it in the notification table in the DynamoDB database to store the user-specific notifications. After storing the notification in the table, the Lambda function would send an email containing the order details to the registered user's email address.

### 3.5 Machine Learning Module [1]

*To identify the similarity of stay duration of customers and propose a tour package.*

Google cloud platform offers various Machine Learning services which club into Vertex AI service [6]. To implement this module, we will use Vertex AI's Auto ML services [6]. Auto ML provides various methods to train, test, and evaluate the model. From the choice of many, we will be using the Tabular form of data to identify the suitable tour package and recommend it to the customers [6]. We will be using a classification algorithm to train the model. The process of developing this module is described below:

- We will create the dataset that will contain information on the stay duration of customers, and we will also set the pipeline that will continue updating the database with new entries about customers' stay duration.
- Clean that database and modify it to best fit the model.
- Apply classification algorithm on Vertex AI's Auto ML service [6].
- Train the model with the training dataset.
- Test the model with the test dataset and evaluate the performance by considering the Confusion matrix and Feature importance matrix [6].
- Generate the Endpoint to access the trained model and do the prediction.
- By entering the data into the model, we will receive a prediction from the model that will tell us what package is suitable for the customer based on their stay duration [6].

*To identify the polarity of customer feedback and to add appropriate scores.*

We will be using Vertex AI's Auto ML service to build this module [5]. Here, we must calculate the polarity of customer feedback with its score of it [5]. Therefore, we need to use the Text format of the data. We can use Text format with Auto ML and can build the machine learning model which takes text data and can give the polarity of customer feedback [5]. The implementation of this module is described below:

- We will create a Text dataset by adding data from various users to our application's database and collecting those data into one directory [4].
- After collecting the data, we have to label all the text files and divide them based on the polarity of the feedback [4].
- Cleaning of the data by removing stopwords, and other less important details from the data files.
- Build the model by using Sentiment Analysis on customer feedback data [4].
- Train the model using the training dataset.

- Test the model by using a test dataset with various matrix that defines the accuracy of the model.
- Generate the endpoint from which we can predict the result of the input data [1].

### **3.6 Web Application Building and Hosting [1]**

We are going to use ReactJS for the front-end development and NodeJS with different Cloud Function as well as Lambda Function for the API connection as a back-end environment. Users will interact with the web application via the front-end module [4][5]. A good choice of front-end and back-end technologies is necessary when developing dynamic web applications [4][5]. Therefore, to achieve application flexibility and scalability, we have chosen to design the front-end using React JS. To access the functionality of GCP Cloud function services on the back end, we will be using Express JS, Node JS, and Lambda services [2][3]. This web application will be hosted in Google Cloud Run via a docker image.

### **3.7 Other essential modules-testing, report generation and visualizations**

For our application, we ran validation tests and function tests. Username-password checks, operations that can only be carried out by authorised users, whether a guest user can carry out tasks that are not assigned to them, whether the chatbot can provide the users with the relevant information they are looking for, whether the users are notified in the appropriate situations, whether the users can obtain specific booking ids and order ids when booking the room and ordering the food, and many other validations and tests. All potential test cases necessary for the application to operate effectively are included in the testing module.

For the report generation module, the AWS QuickSight service is used, which comes with business intelligence dashboards and applications, for the report generation module. Reports on user login and access statistics are generated by this module. The report can be generated by the application's authorized users. Each user activity will be recorded using the AWS CloudWatch service. The Lambda function will store all the user login activities in the DynamoDB database from AWS CloudWatch. The QuickSight service receives the user activity data in order to produce the reports.

To visualize the statistical data reports, we have used GCP Data Studio, BigQuery, AWS Lambda functions, GCP Cloud functions, AWS S3 and GCP CloudStorage. We have used the Lambda function to extract the data from the DynamoDB tables and store it in the AWS S3 buckets. For example, to generate the room booking visualization, the function will extract the room booking data from DynamoDB and store it in CSV format in the S3 bucket. Another function will send this file to CloudStorage. Whenever the data arrives in the CloudStorage bucket, a cloud function will be triggered. This cloud function will extract the file from the GCP bucket and create a data set and table in BigQuery for report visualization. From the BigQuery service, the data is fetched into the GCP Data Studio to visualize in different forms such as tabular forms, pie charts, donut charts etc.

## 4. Cloud Architecture Diagram [1]

Figure 1 depicts the cloud architecture of the application. The diagram is designed using draw.io [2] by gathering knowledge from sources Google Cloud Documentation [2], and AWS Documentation [3]. As we can see in the figure, the users will interact with the front-end of the application developed in ReactJS. The front-end of the application is hosted on GCP Cloud Run via the docker image which is stored in the Container Registry of GCP. AWS API gateway is used to navigate the requests from various services to AWS Lambda and GCP API Gateway. GCP API Gateway is used to direct the requests to the GCP Cloud Functions. Amazon DynamoDB is used for the storage of data for the application [1].

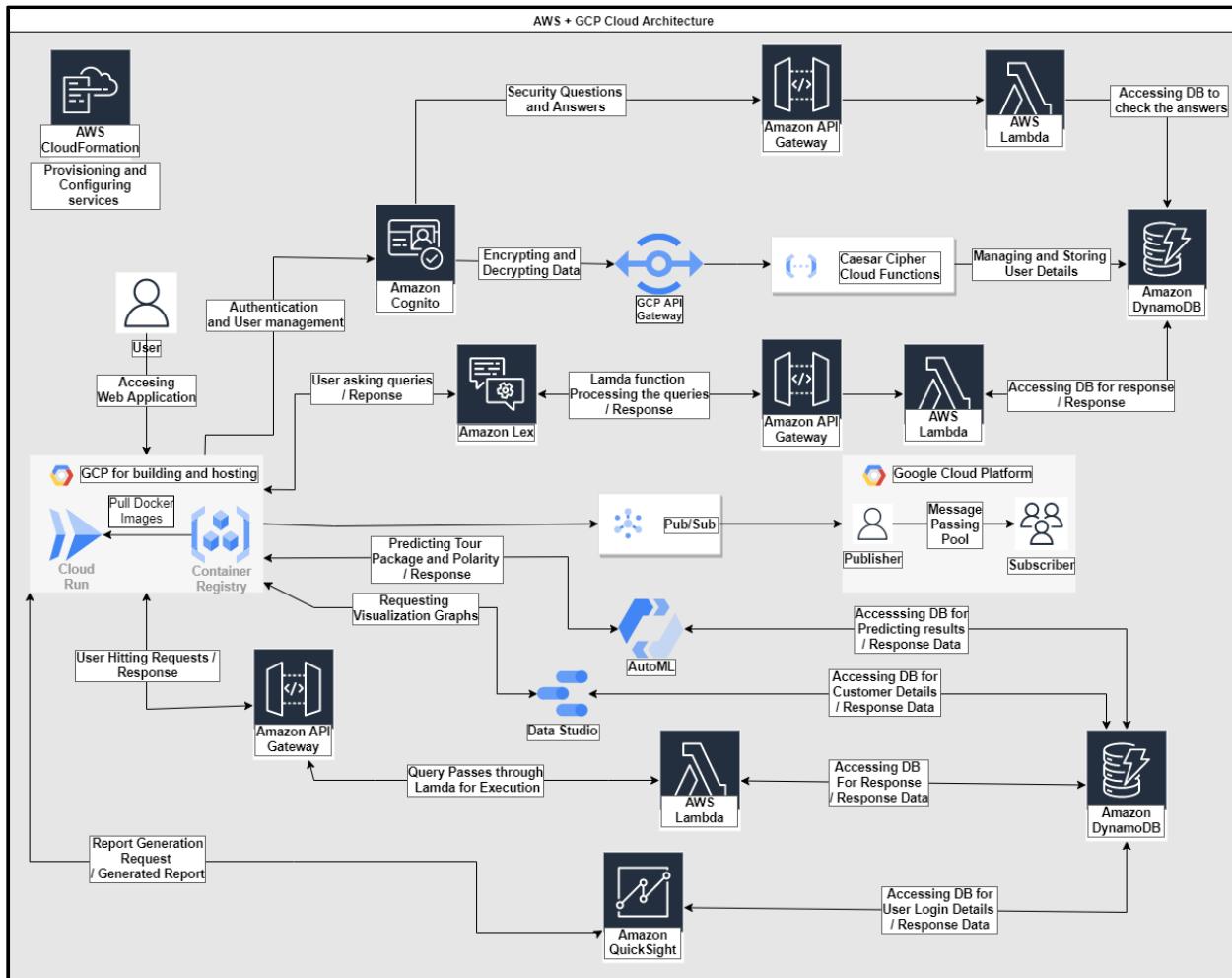


Figure 1: Cloud Architecture Diagram [1][2]

## 5. Pseudo Code

### 5.1 User Management

1. User enters the username, email, password
2. The entered username, email and password is checked for validations and for proper format for email and password.
3. If the format is wrong, the system throws an error; If not, the user is proceeded to the next phase.
4. In the next step, the user enters the answers for the security questions asked by the system.
5. The user must remember the answers to the questions required when they try to login.
6. In the next step, the system generates a key for the Caesar cipher verification.
7. The user is supposed to remember the key generated by the system.
8. When all the details are correctly entered by the user which match the requirements of the system, the user is successfully registered to the system.
9. The registered data is stored in the database.
10. If the details are not valid and do not follow the system requirements, then the user is thrown an error and redirected back to the registration page.
11. On successful registration the user is redirected to the login page.
12. If the user is unregistered, then also the user can access the system
13. In case of unregistered user, the user will get minimum access wherein the user can access a smaller number of features in the application.

### 5.2 User Authentication (3 Stage Verification)

1. Request the person's email address as both their username and password.
2. Call the AWS Cognito API to verify your credentials.
3. If the credentials are wrong, go back to step 1 or continue to step 4.
4. Request the user to enter their security question and response.
5. If the response to the security questions is wrong, go back to step 4; otherwise, continue.

6. Request that the user use the registration-generated key to decrypt the Caesar cypher text.
7. Retry step 1 if the user is unable to decrypt the text. Otherwise, direct the visitor to the application's main page.

### **5.3 Online Support Module:**

Pseudo code for Lambda functions:

bookMyRoom.js

1. First, it will scan through the ‘roomDetails’ table where information related to room is stored and roomNumber is stored as the key.
2. After taking all the inputs from the user in lex intent(s) [as it is set as fulfilment trigger for BnbAuth’s BookRoom intent], it will record all the data and combine data of the intent and step 1 as an ‘item’.
3. This ‘item’ will be passed as an argument to putItem and the details will be recorded into ‘room’ table where all the room booking information resides.\

orderMyFood.js

1. First, it will scan through the ‘foodDetails’ table where information related to food is stored and foodName is stored as the key.
2. After taking all the inputs from the user in lex intent(s) [as it is set as fulfilment trigger for BnbAuth’s OrderFood intent], it will record all the data and combine data of the intent and step 1 as an ‘item’.
3. This ‘item’ will be passed as an argument to putItem and the details will be recorded into ‘foodorder’ table where all the food order information resides.

navigateWebsite.js

1. This function will just scan the input by the user in the ‘NavigateWebsite’ intent and send the message i.e., the link of the page user desired from the menu in chatbot UI.

## 5.4 Message Passing Module

1. User clicks on room booking/food ordering/tour booking.
2. System makes a request with required data to Lambda function.
3. Lambda function stores the data in DynamoDB and sends back the success message.
4. System makes another request with data to Cloud Function.
5. Cloud Function publish the message to the relevant topic.
6. Subscriber service fetches message from the topic queue and pass it to AWS Lambda function.
7. Lambda function prepares notification message and store the notification to a DynamoDB table.
8. Once the data is stored, the Lambda function sends email to the user.
9. User navigates to notification page.
10. System makes a call to Lambda function.
11. Lambda function fetches the user's notification from DynamoDB and sends to the application.
12. System displays the notifications.

## 5.5 Machine Learning Module

The below implementation has been done by referring the official documentation provided from sources [13][14][15].

### Tour Package Recommender:

1. User clicks enter data from the front-end application and post request will be hit to the cloud function.
2. Cloud function will be invoked and fetch the request json body and get all the parameters.
3. Cloud function will hit the machine learning model's endpoint and send the instance\_dict to that model to get the prediction.
4. Here, function will get all the prediction and store them in dictionary format.
5. Form here, function will determine which class has the best score and return that result to the front-end.
6. Result of tour recommendation model will be displayed to the user.

### **Sentiment Analysis:**

1. User enter the feedback from the front-end.
2. After hitting the submit the cloud function will be invoked.
3. Cloud function get the text data from the request JSON body.
4. Cloud function calls Cloud Natural Language Processing library provided by GCP and fetch the sentiment score of the text.
5. Cloud function save that score and add the sentiment score to the dynamodb table.
7. Return with the 200 status.

## **5.6 Web Application Building and Hosting**

### **Room Service**

1. User clicks on room booking then a list of rooms which are available will be displayed on the page.
2. Once the user selects a particular room and book it then that room with the given user id will get stored in the dynamodb.
3. When the user clicks on the book button then a call will be made to the API Gateway as different routes/endpoints are assigned in the roomapi on the aws.
4. Each API endpoints are integrated with the Lambda function and the lambda function will post the data in the database.
5. User clicks on the book button.
6. API endpoint will be hit and a dedicated function will be called.
7. At the successful storing of user information with room info and date from which the reservation is made.
8. If everything is ok then room booked successfully message will be seen on the website.
9. Once the room is reserved then that room will be deducted from the available room database.

### **Kitchen Service (Get Food):**

1. Loading the NodeJS AWS-SDK.
2. Using aws.config.update to configure AWS credentials.
3. Making a service object for DynamoDB.
4. Create a JSON object with the parameters needed to search for items in the table, such as TableName: "getfood"
5. Calling the procedure getMenu while supplying a JSON object
6. In particular Function
  - a. Calling the scan method of the DynamoDB service object
  - b. Creating a JSON menu string by iterating the response.
  - c. Give the menu back.
7. In response to an API request, returning the menu.

### **Kitchen Service (OrderFood):**

1. Loading the NodeJS AWS-SDK.
2. Using aws.config.update to configure AWS credentials.
3. Making a service object for DynamoDB.
4. Getting order items from a JSON request
5. Iterating the order items to figure out how long it takes to prepare an order.
6. Making an order ID.
7. Extracting order information from the JSON request.
8. Making a JSON object with the parameters required to store the object in DynamoDB.
9. Call the function that orders orders from DynamoDB.

## **Tour Service**

1. User clicks on tour booking then a list of tours which are available on the season will be displayed on the page.
2. The following images of tour and its information will be get or displayed using the getapi which takes information from DynamoDB.

3. When the user clicks on the book button then a call will be made to the API Gateway as different routes/endpoints are assigned in the tourapi on the aws.
4. Each API endpoints are integrated with the Lambda function and the lambda function will post the data in the database.
5. User clicks on the book button.
6. Api endpoint will be hit and a dedicated function will be called.
7. At the successful storing of user information with tour info and date from which the reservation is made.
8. If everything is ok then tour will be booked successfully and a message will be seen on the website.

## **5.7 Other Essential Module (Report/Visualization)**

1. Lambda functions checks periodically and extracts data from DynamoDB to store it in AWS S3.
2. Another function sends the extracted data from S3 to GCP Cloud Storage.
3. Cloud function gets triggered as the data arrives in GCP Cloud Storage.
4. Cloud function takes data from storage and creates a dataset and table in the BigQuery.
5. BigQuery service performs query on the generated table.
6. GCP Data Studio takes data from BigQuery.
7. Data Studio generates visuals from the data.
8. User clicks on the visualisation.
9. The system displays the visualisations created by the Data Studio.

## 6. Flowchart/Activity Diagram [1]

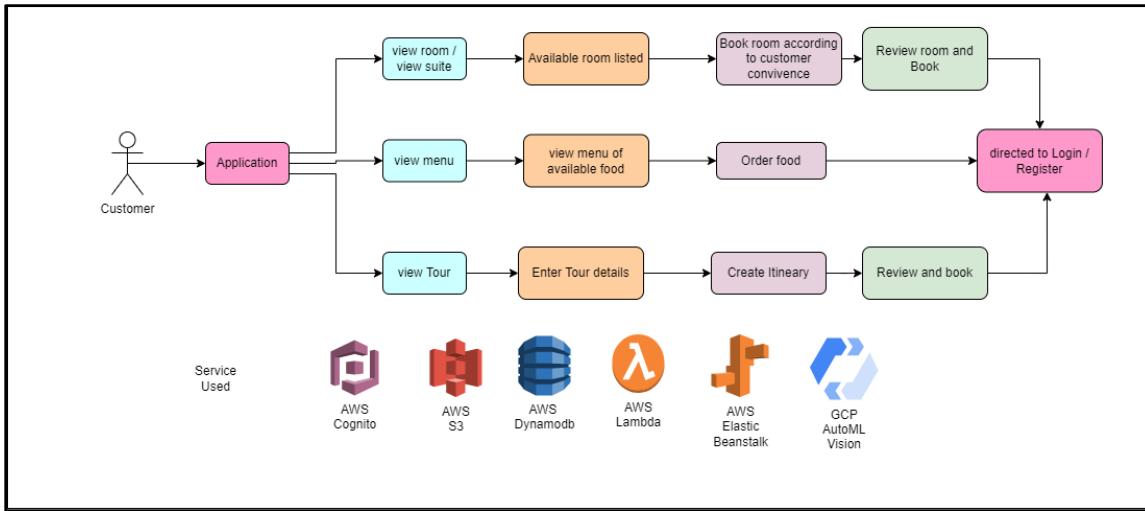


Figure 2 Unregistered Customer Roadmap [3][4][5]

- Non-registered users

Users who are not signed up will be able to browse rooms, menus, and tours. The user will see all the available rooms and their specifications when they access the rooms. The user will be sent to the login/registration page to proceed when attempting to select and book the room [4][5]. The menu can also be viewed by the user. When a user examines the menu, all the available breakfast options are displayed along with information like pricing. Additionally, the user can view recommended tours by selecting the view tours option. It will request the user's stay time and then display results based on how closely that duration matches the user's input [4][5].

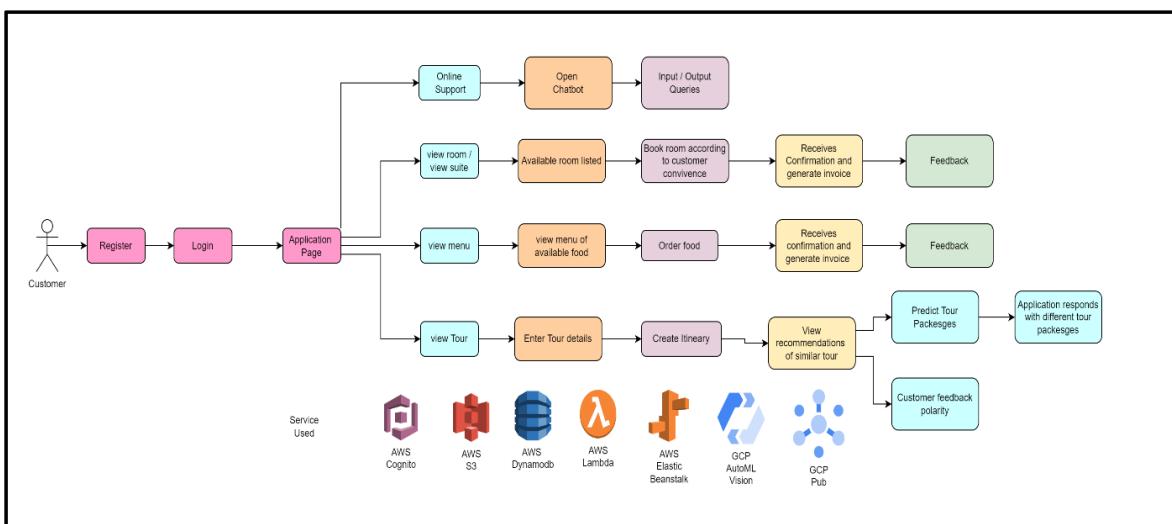


Figure 3 Registered Customer Roadmap [3][4][5]

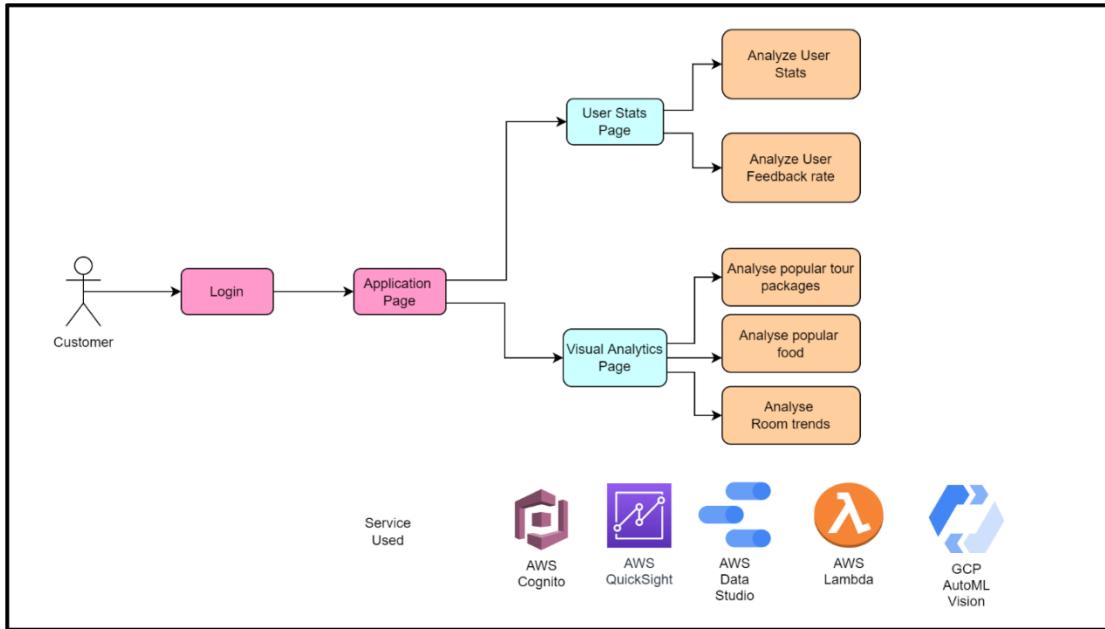


Figure 4 Admin Roadmap [3][4][5]

- **Registered users**

The register is chosen by the user from the navigation bar. They answer security questions and input their login and password. After submitting the form, Cognito will be used by a Lambda function through an AWS gateway to register the user [4][5]. The responses to the security question will also be kept in DynamoDB by Lambda. Additionally, upon successful registration, they will receive a random number that will be saved in DynamoDB [4][5].

Once the user is registered then he/she does not have to sign up again as data is already present in the DynamoDB so the user will sign in the next time he /she is using the application [4][5]. Now we will illustrate different scenarios of the interaction of users with the services available on our application [4][5].

## I. User interaction with a chatbot:

The user will be able to communicate with a chatbot after logging in. Users can use this bot to search for available rooms and browse the website [4][5]. Additionally, it will give them suggestions and useful guidance for booking reservations and exclusive placing meal orders [4][5].

- **Unauthorised users:** The flow diagram below shows how various services run and communicate with others in cloud environment for BnbUnauth bot.

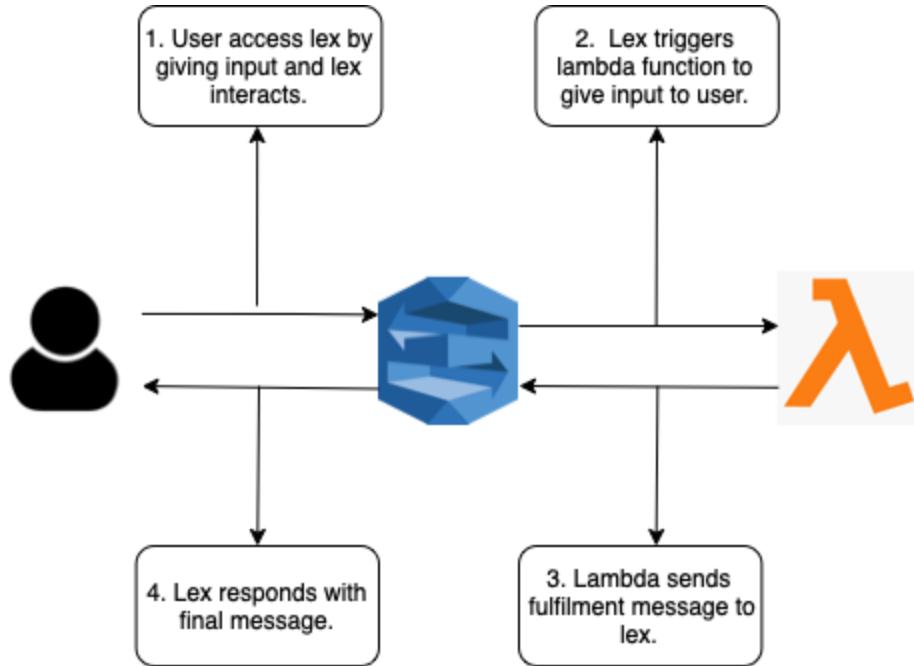


Figure 5 Flow diagram[2] of services like Amazon Lex[10] and Lambda function[8] interacting with each other for unauthorised users.

- **Authorised users:** The flow diagram below shows how various services run and communicate with others in cloud environment for BnbAuth bot.

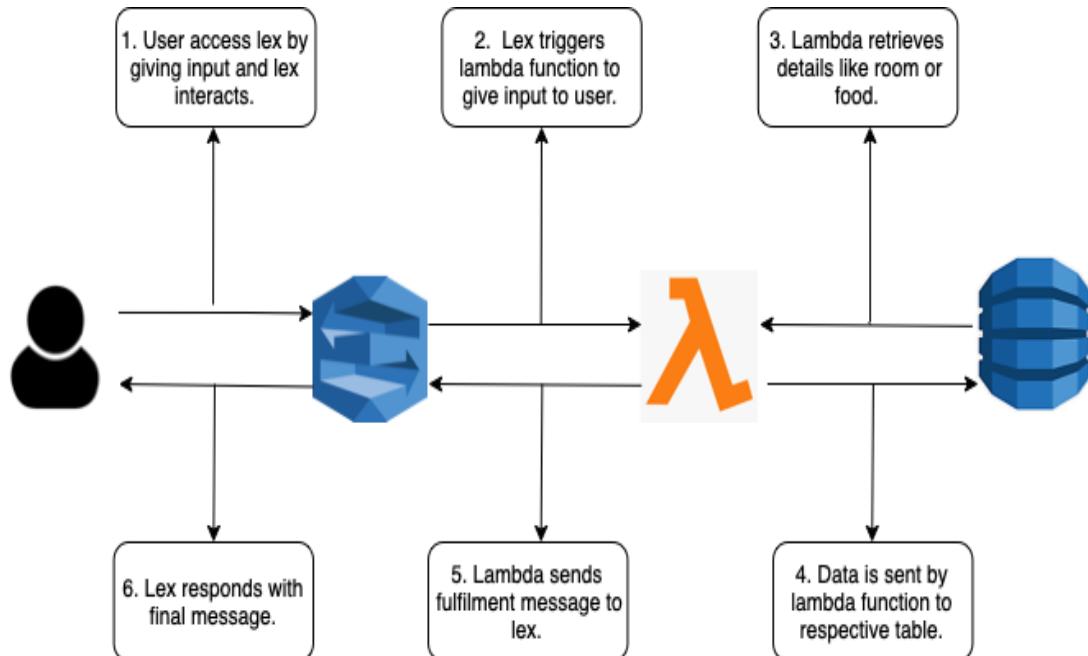


Figure 6 Flow diagram[2] of services like Amazon Lex[10], Lambda function[8], and DynamoDB[7] and interacting with each other for authorised users.

**II. User booking a room:**

The chatbot will let the user book a room or a bed as they interact with it. Once the user selects the option they want to use, this will happen automatically. Once the confirmation is received, the information about the reservation will be updated in DynamoDB [4][5]. After the reservation, the consumer will be asked for feedback. Sentiment analysis will be done on the feedback, which will be stored in DynamoDB [4][5].

**III. User ordering food(breakfast):**

Users will be able to examine a pre-defined menu that includes information on the breakfast alternatives that are offered, as well as their specifics and cost. Users will be able to choose from the menu and place meal orders. After a successful order, DynamoDB will save the order data, and the user will get an order confirmation notification [4][5]. They will also be able to see the order's current status, including whether it is being placed, prepared, ready, or delivered. Invoices are produced and saved to S3 buckets after orders are delivered [4][5].

**IV. User viewing suggested tours:**

Following the completion of the order, the customer will be requested to provide feedback [4][5]. This feedback will be kept in DynamoDB and based on the data, sentiment analysis will be performed to determine if the feedback was good or negative [4][5].

## 7. Individual and Team Contribution

Module name	Team member
<b>User Management Module</b>	Adarsh Kannan Iyengar
<b>User Authentication Module</b>	Meet Patel
<b>Online Virtual Assistance Module</b>	Ruhi Tyagi
<b>Message Passing Module</b>	Kunj Patel
<b>Machine Learning Module</b>	Fenil Parmar
<b>Room Booking, Tour Booking Module</b>	Jenish Patel
<b>Web Application building and hosting</b>	All Team Members
<b>Other Essential Module</b>	Kunj Patel
<b>Reports and Documentations</b>	All Team Members
<b>Demo Video</b>	All Team Members

## 8. Meeting Logs

### Meeting 1

Date: 17<sup>th</sup> May 2022

Agenda: Team Introduction

### Meeting 2

Date: 25<sup>th</sup> May 2022

Agenda: Discussion about the project specifications and getting to know the requirements.



Figure 7 Meeting 2 Log

### Meeting 3

Date: 29<sup>th</sup> May 2022

Agenda: Task distribution and discussion regarding the cloud services that are to be used in the project.

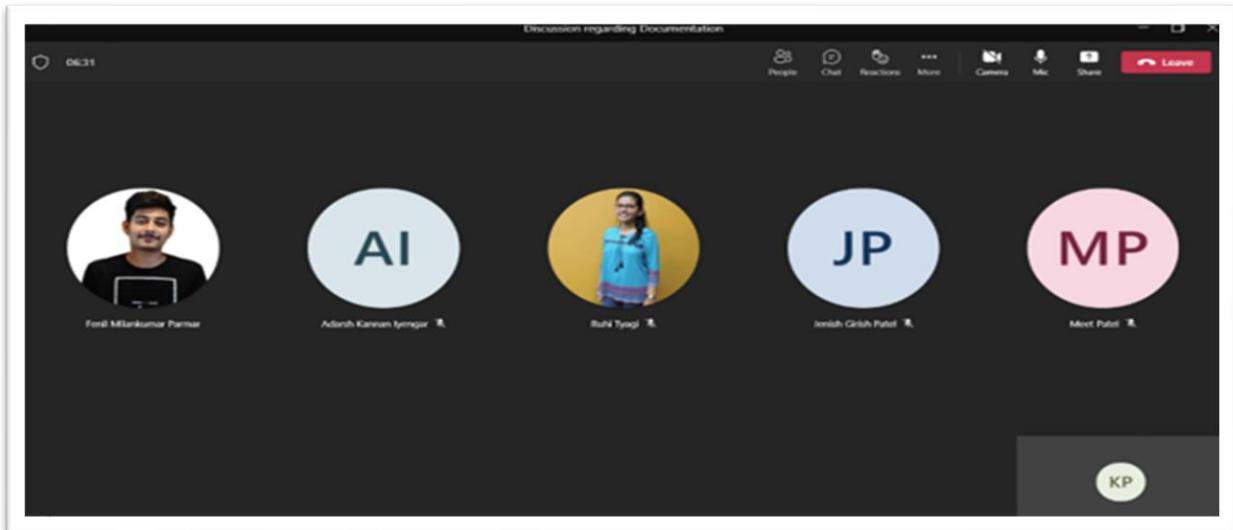


Figure 8 Meeting 3 Log

## Meeting 4

Date: 1st June 2022

Agenda: Reviewing the conceptual design report

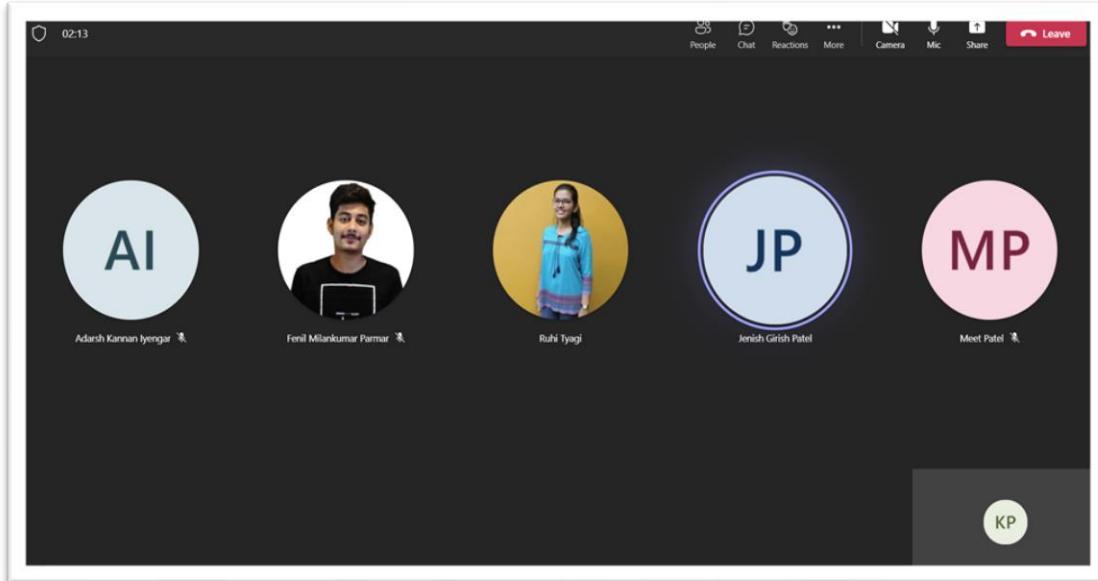


Figure 9 Meeting log 4.

## Meeting 5

Date: 28th June 2022

Agenda: Discussion regarding conception report feedback and dividing the modules amongst the team members.

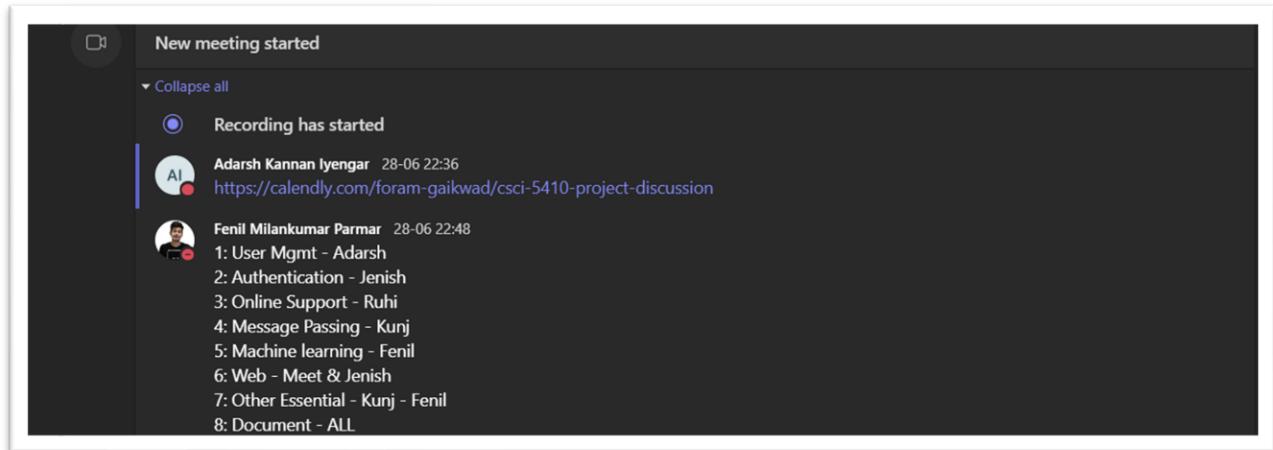


Figure 10 Meeting log 5

## Meeting 6

Date: 4th July 2022

Agenda: Discussion regarding the design document and dividing the documentation tasks amongst the team members.

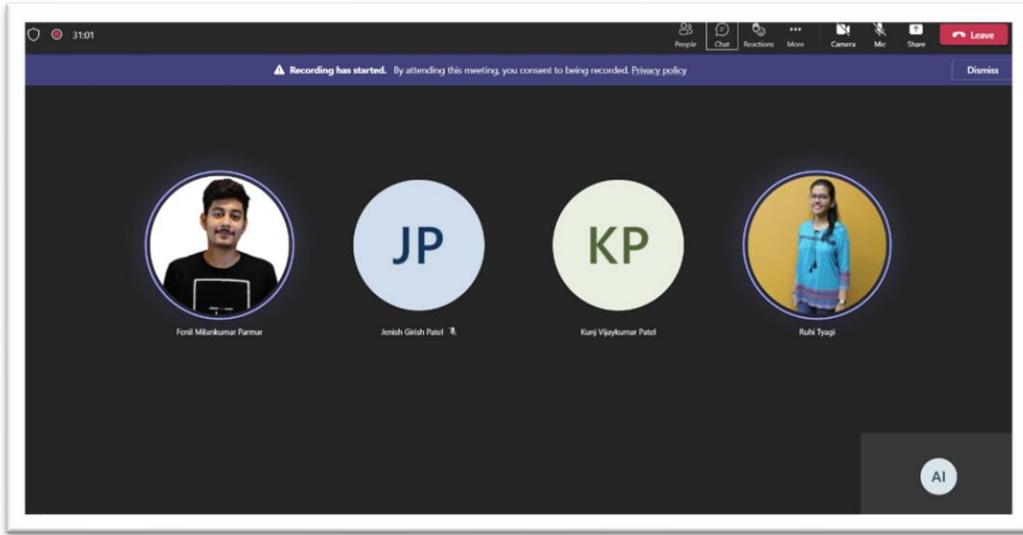


Figure 11 Meeting log 6

## Meeting 7

Date: 5th July 2022

Agenda: Final meeting regarding the design document formatting and submission

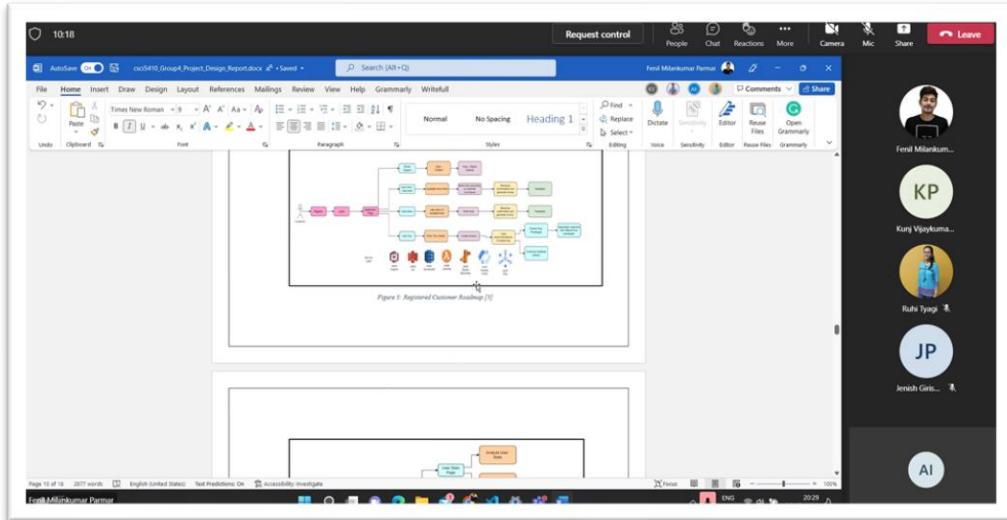


Figure 12 Meeting log 7

## **Meeting 8**

Date: 18th July 2022

Agenda: In Person Meeting regarding project implementation and integration

## **Meeting 9**

Date: 23rd July 2022

Agenda: In Person meeting regarding demo video and final report documentation

## 9. Test Cases and Evidence of Testing

### 9.1 User Login and Registration

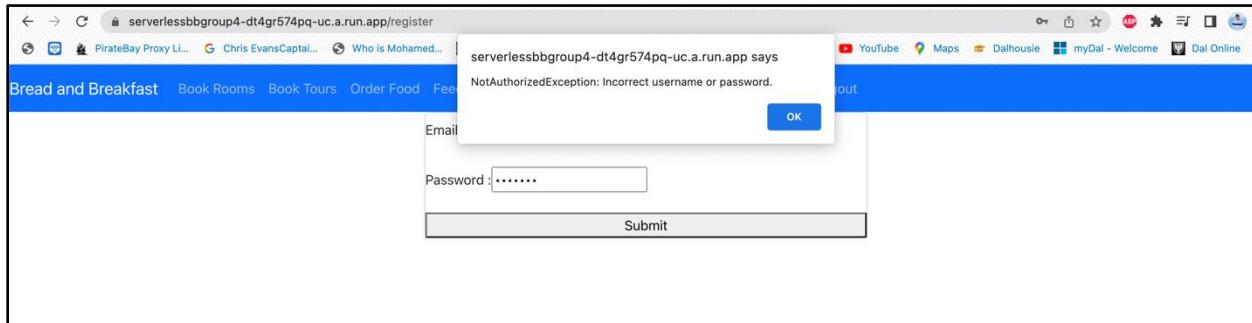


Figure 13 Login Authentication Failed Test Case

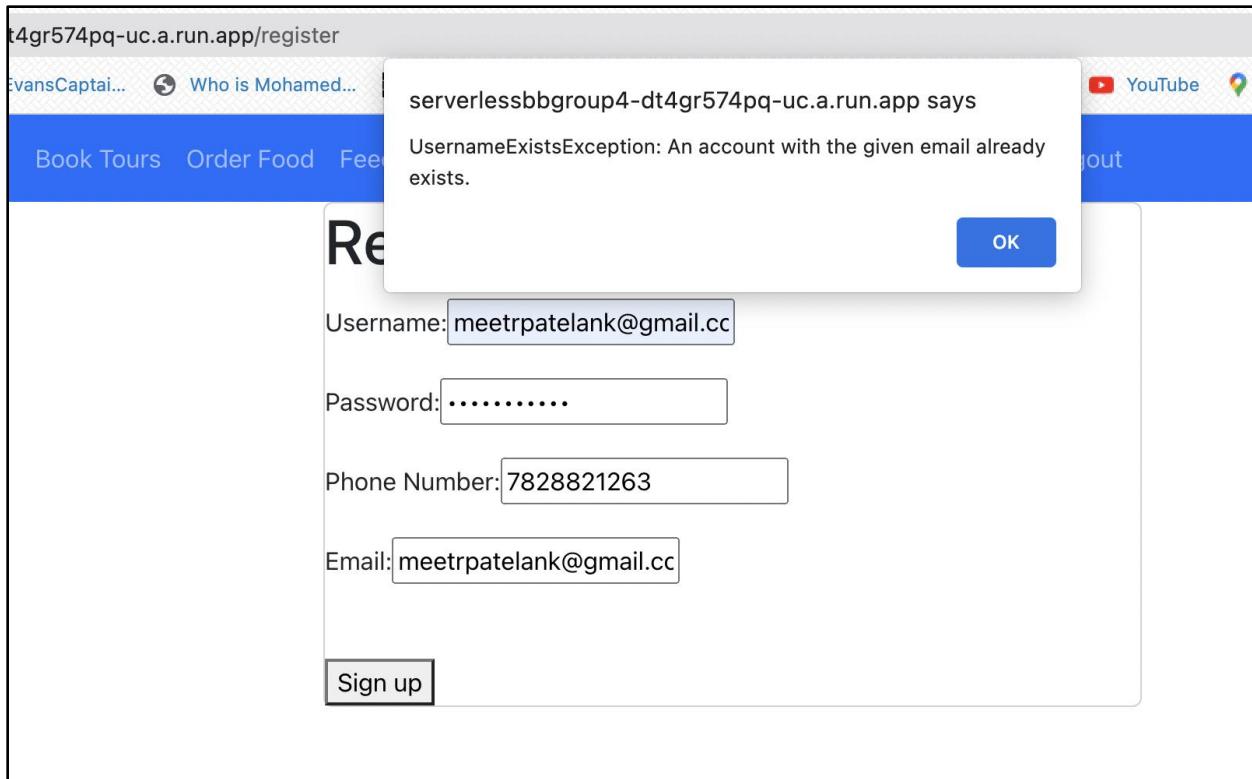


Figure 14 User Already Signed Up Test case

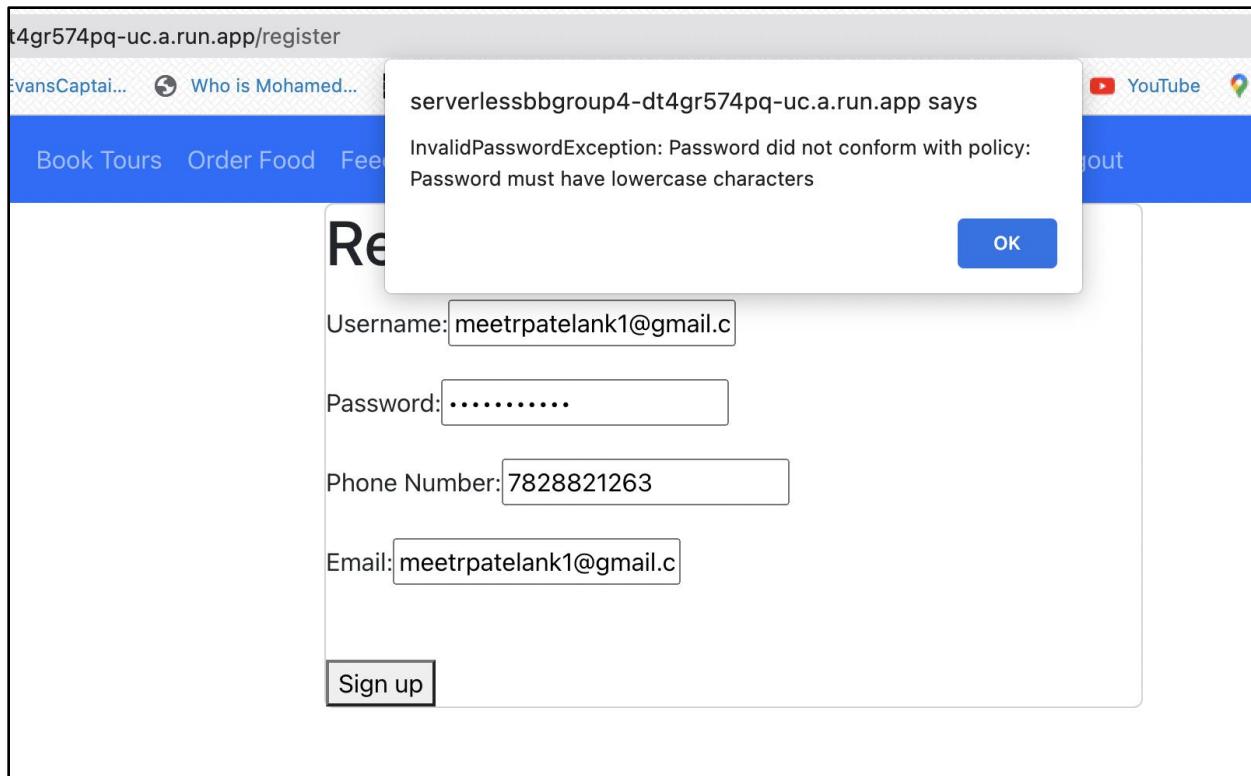


Figure 15 Password policy not matched

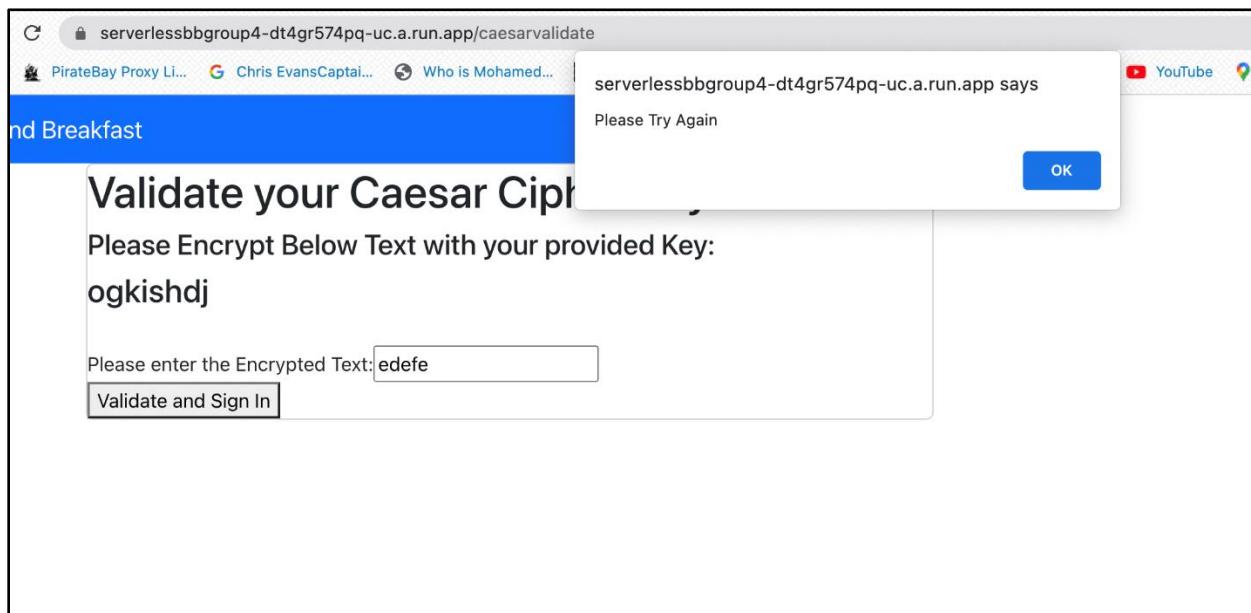


Figure 16 Caesar cipher failed.

### 9.3 Online Support Module:

Screenshot 17-21 show the unauthorized functionalities of bot.

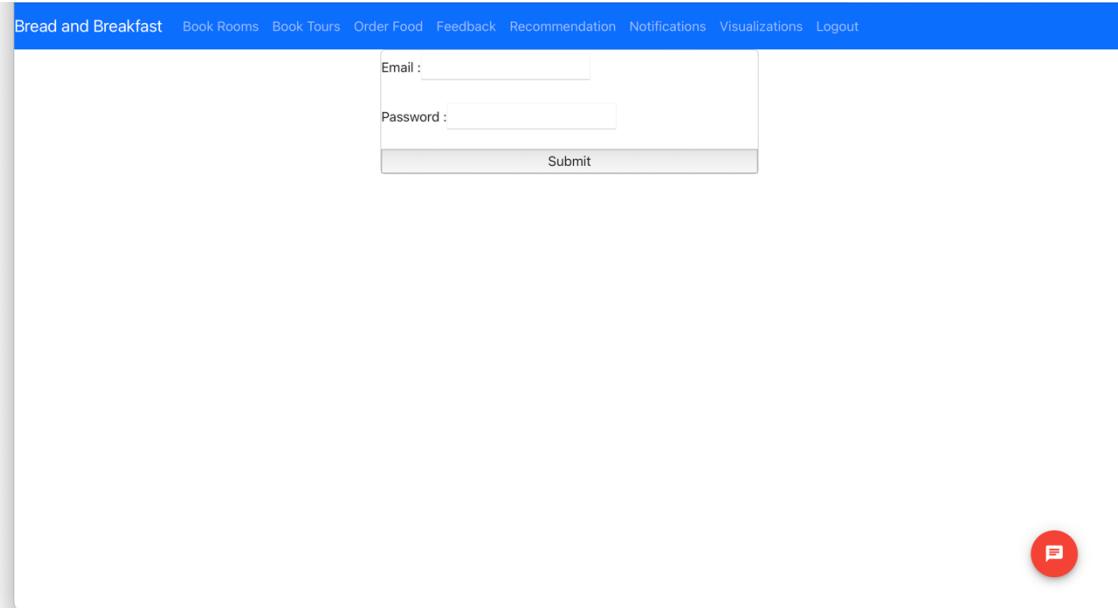


Figure 17 Screenshot of chatbot successfully deploying on the website.

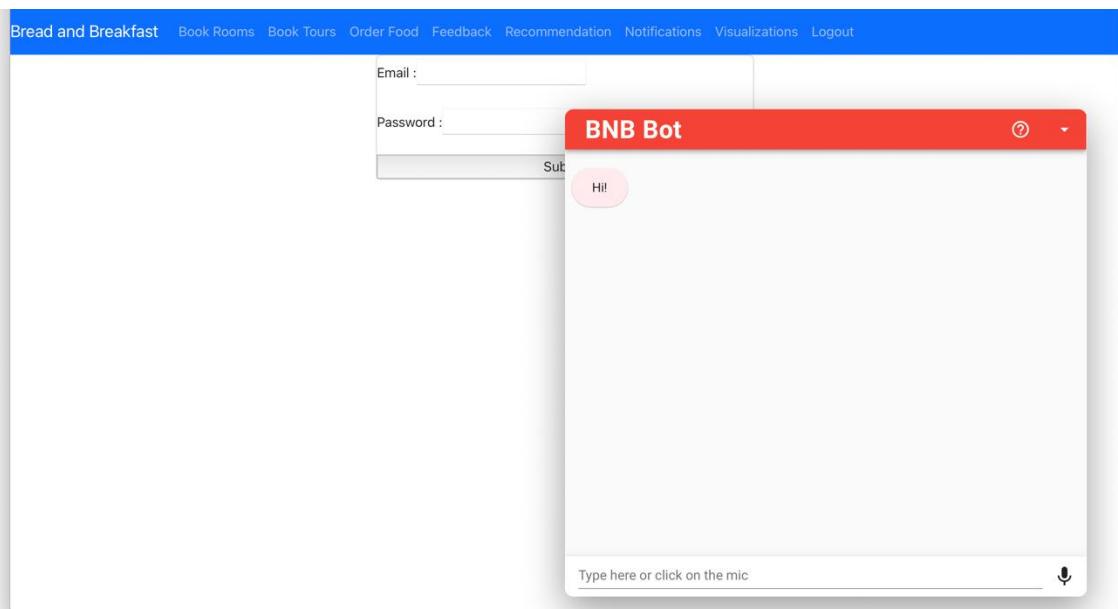


Figure 18 Screenshot of chatbot successfully deploying on the website.

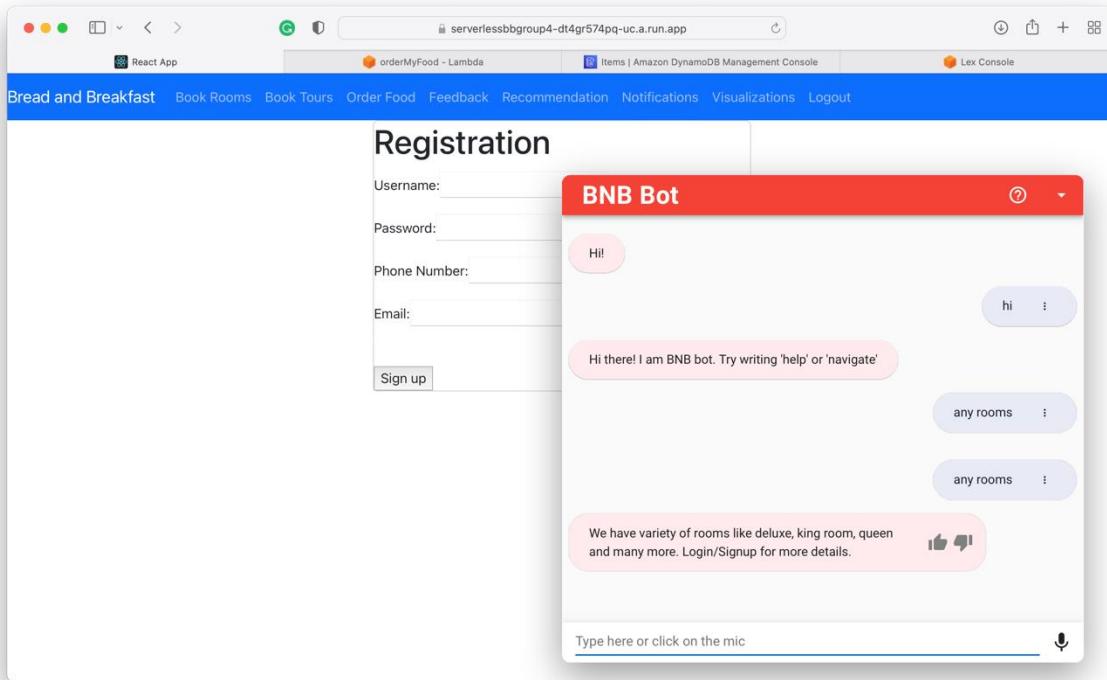


Figure 19 . Screenshot of chatbot performing unauthorized function: room details.

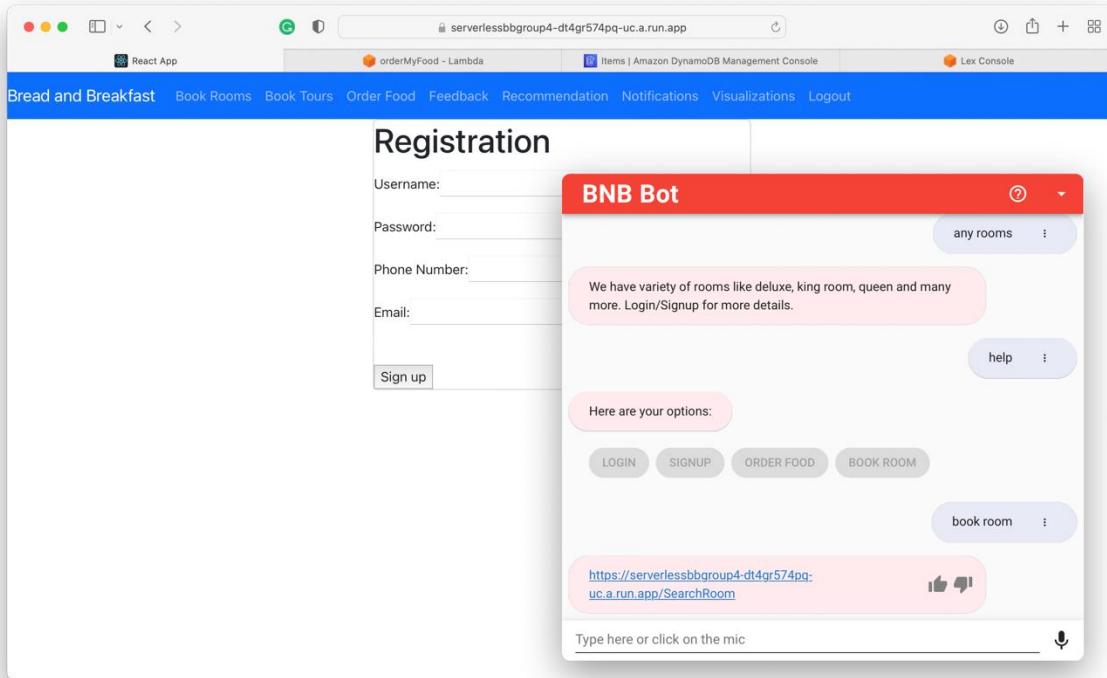


Figure 20 . Screenshot of chatbot performing unauthorized function: navigation/help.

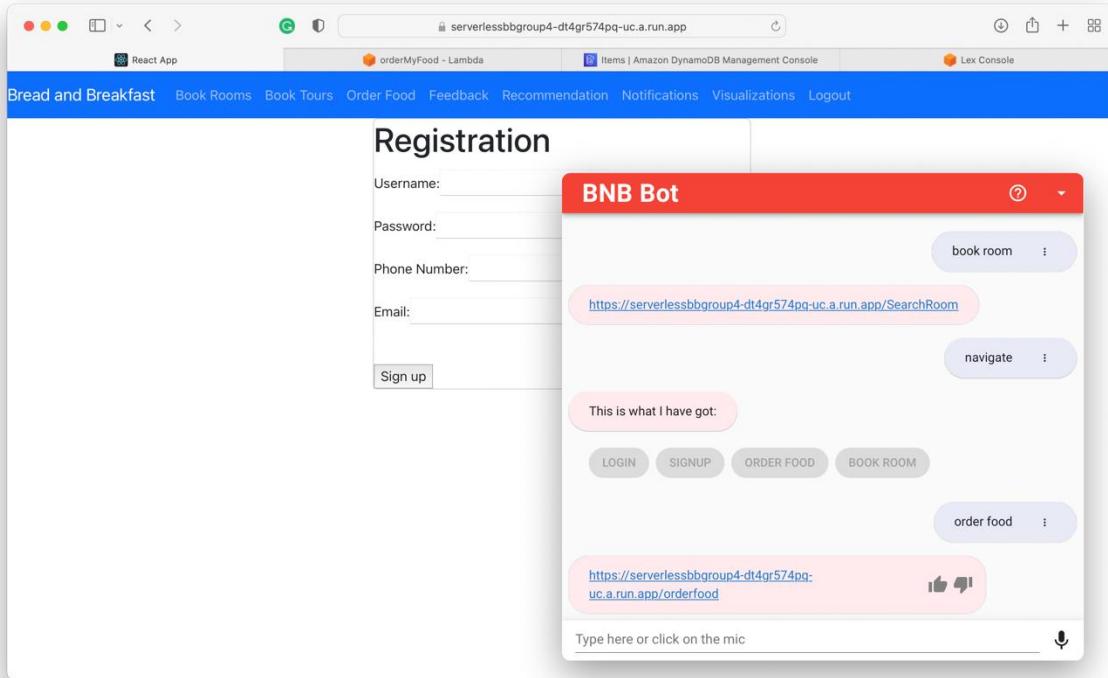


Figure 21 Screenshot of chatbot performing unauthorized function: navigation/help

Screenshot 22-26 show the authorized functionalities of bot.

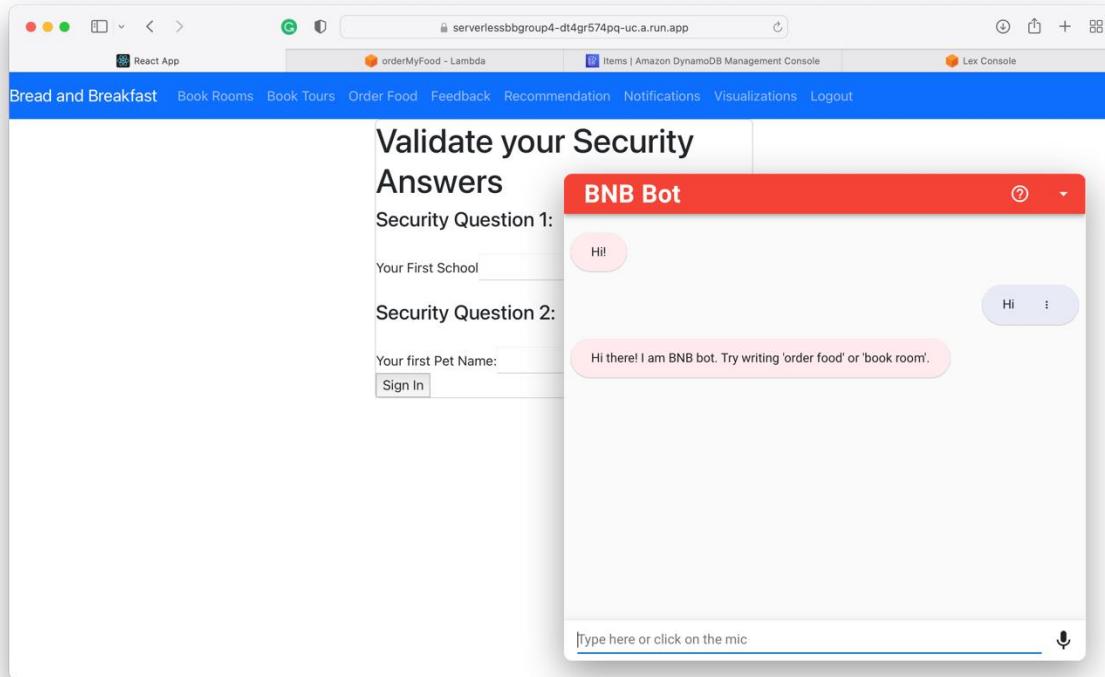


Figure 22 Screenshot of activation of authorized chatbot after login.

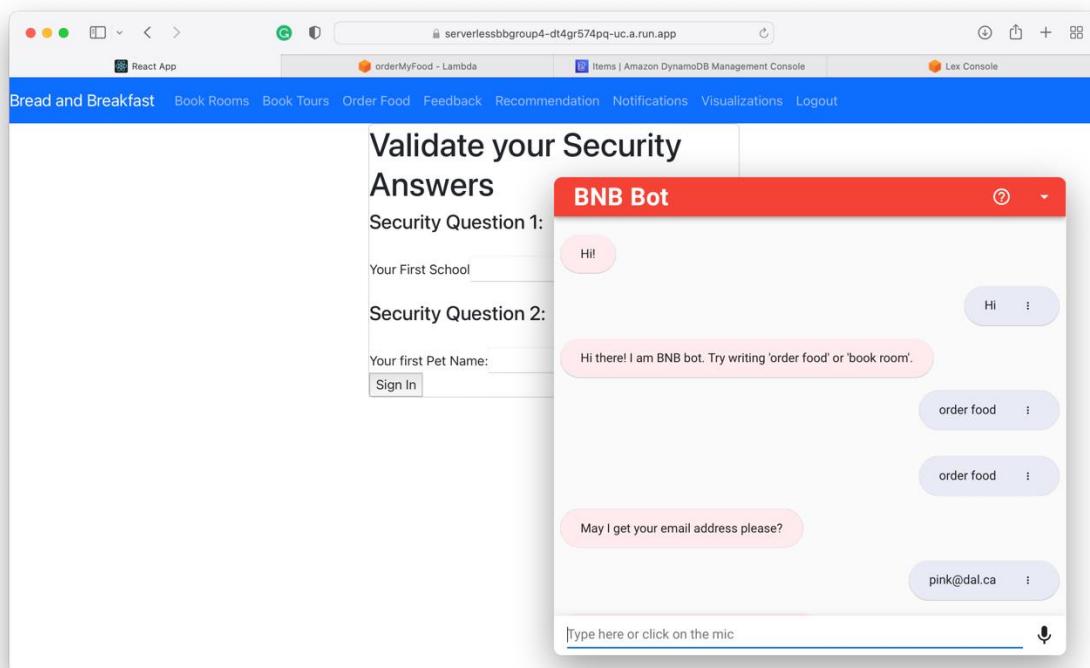


Figure 23 Screenshot of chatbot performing authorized function: order food.

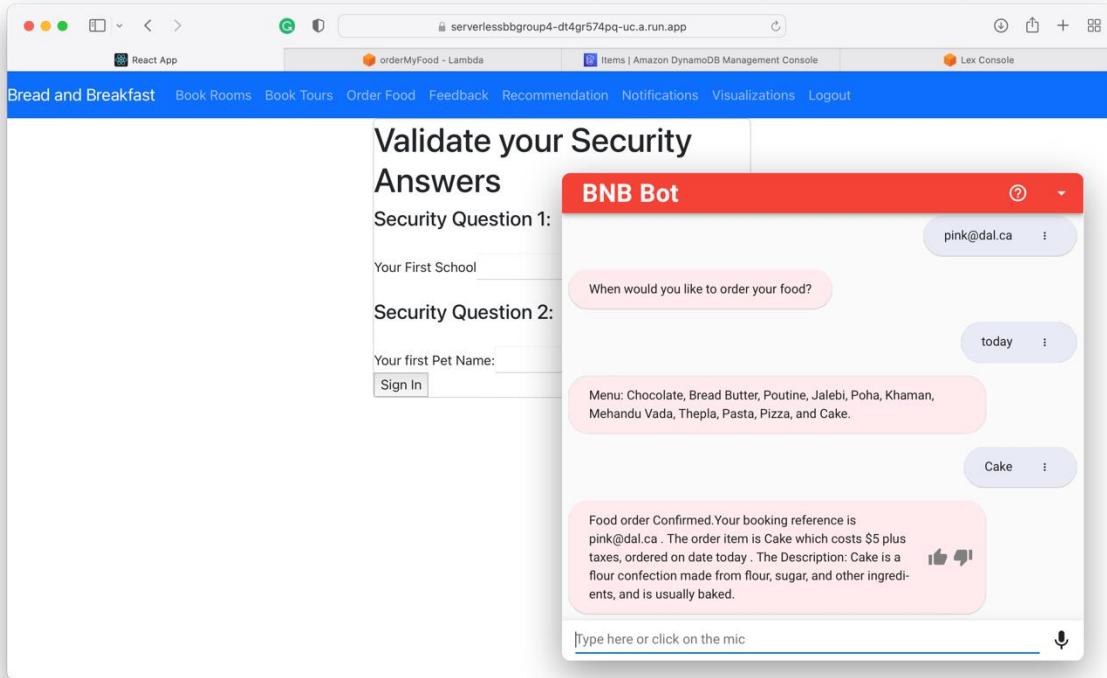


Figure 24 Screenshot of chatbot performing authorized function: order food with confirmation message.

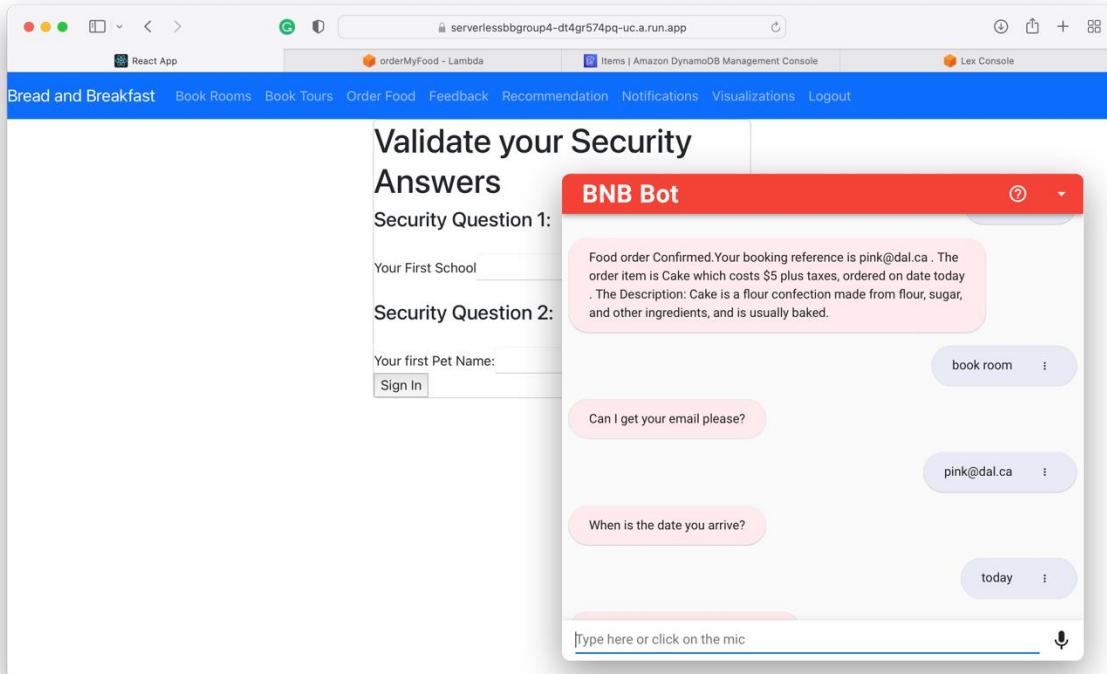


Figure 25 Screenshot of chatbot performing authorized function: book room.

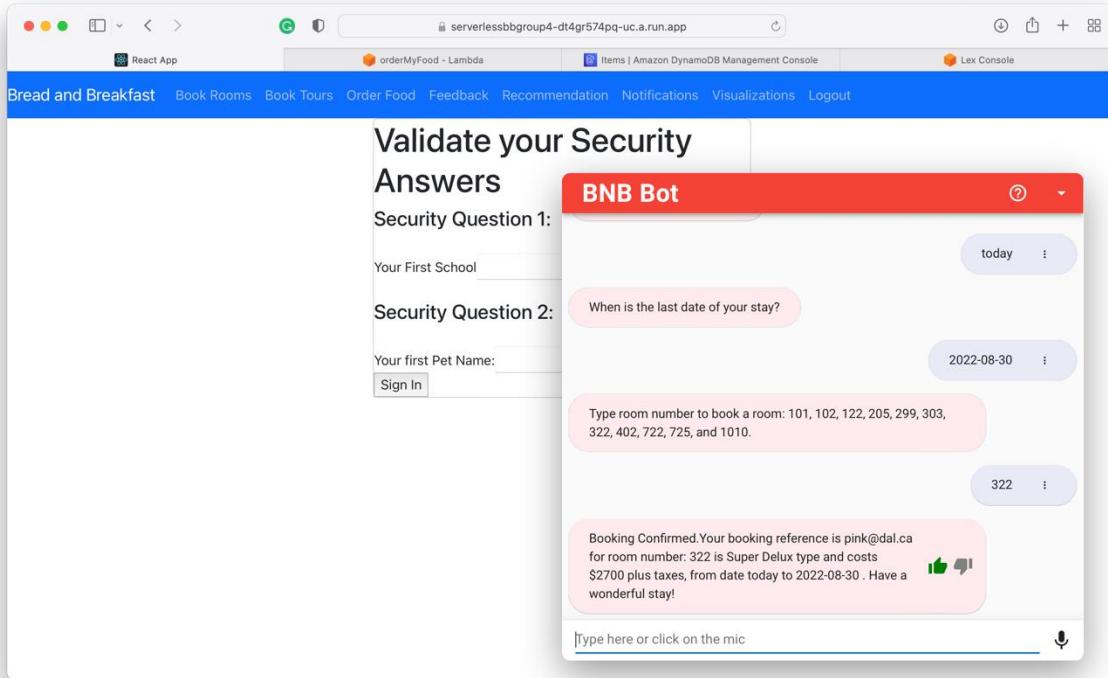


Figure 26 Screenshot of chatbot performing authorized function: book room with confirmation message.

Screenshot 27-22 show the unauthorized action attempted by user from chatbot and how website handled it.

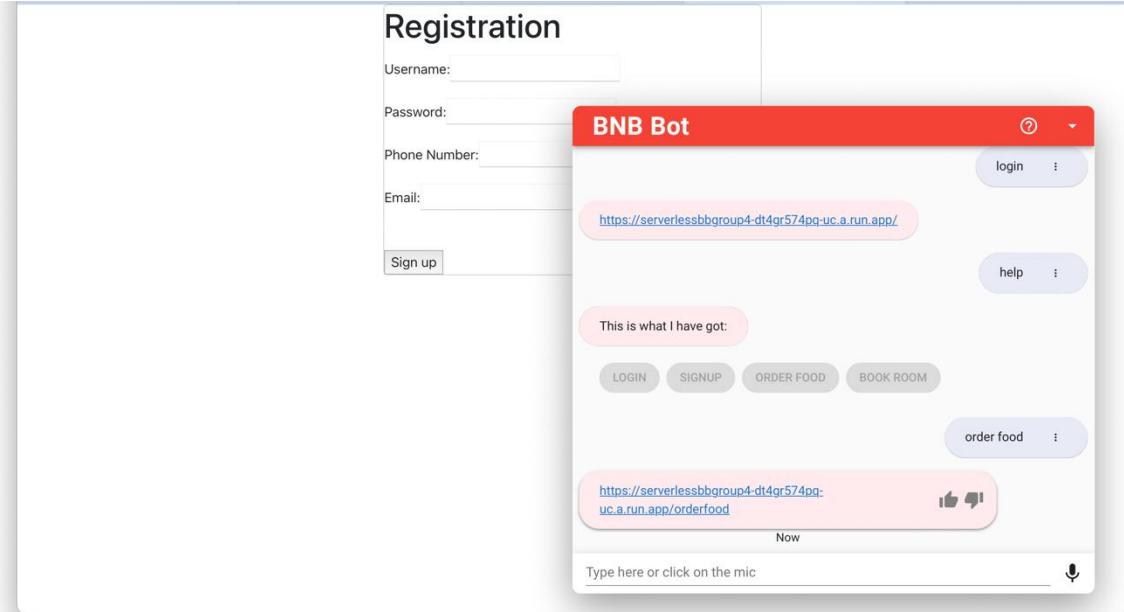


Figure 24. Screenshot of user trying to navigate to orderfood page before login.

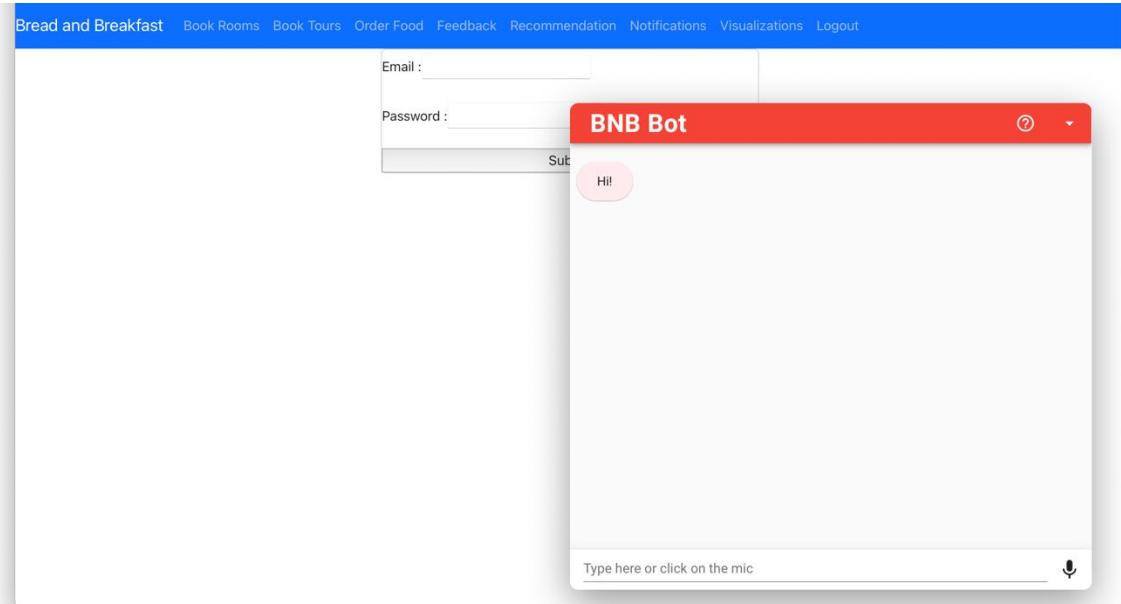
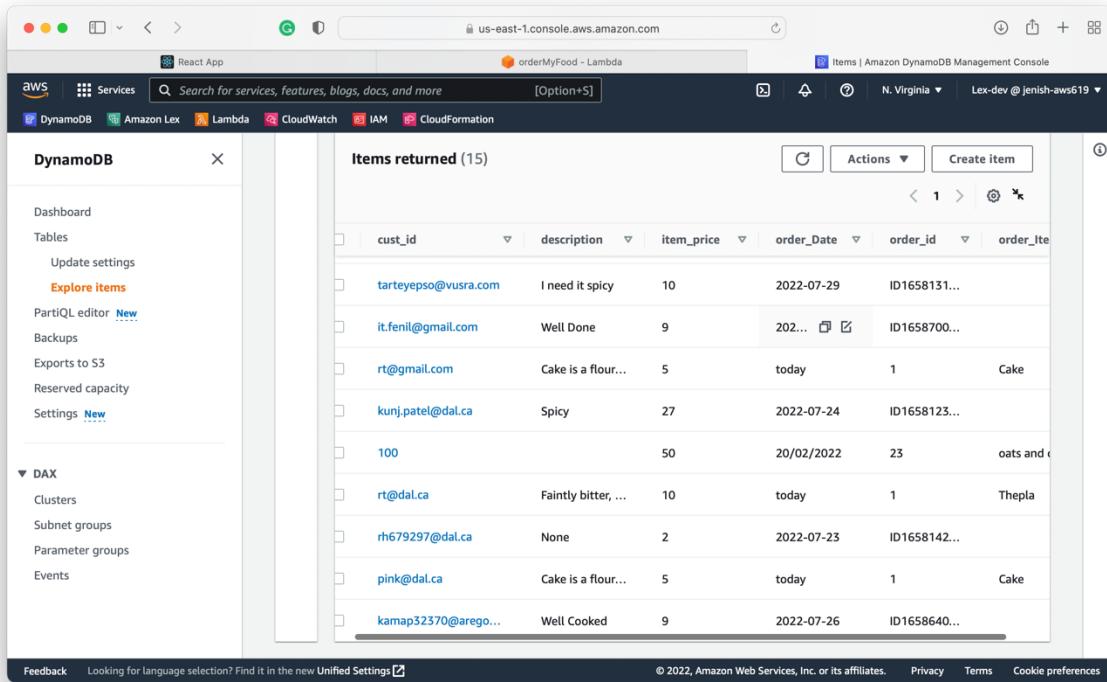


Figure 25. Screenshot of website landing on login page after user trying to access authorized functionality and chatbot successfully deployed.

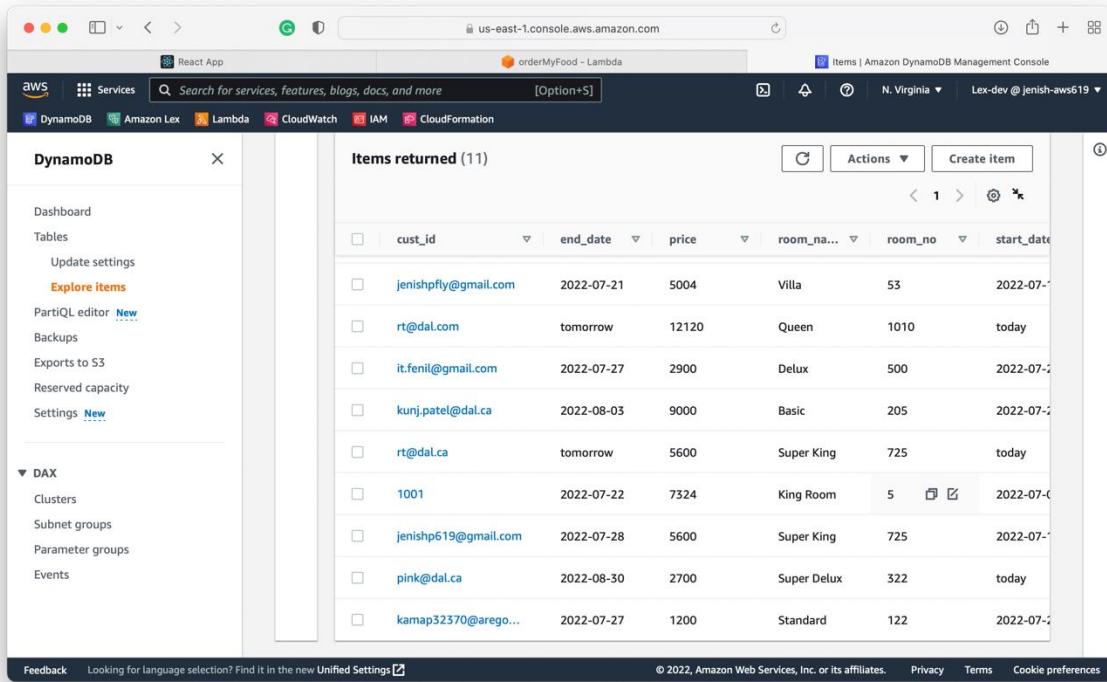
Screenshot 23-24 show the entry of records taken from the chatbot into respective tables.



The screenshot shows the AWS Lambda function 'orderMyFood - Lambda' running in the us-east-1 region. The Lambda function is triggered by an S3 event. The Lambda code is a simple Node.js script that reads a JSON file from S3 and inserts it into a DynamoDB table named 'foodorder'. The table has 15 items, each representing a food item with fields like cust\_id, description, item\_price, order\_Date, order\_id, and order\_lte.

Items returned (15)						
	cust_id	description	item_price	order_Date	order_id	order_lte
□	tarteyepso@vusra.com	I need it spicy	10	2022-07-29	ID1658131...	
□	it.fenil@gmail.com	Well Done	9	202...	ID1658700...	
□	rt@gmail.com	Cake is a flour...	5	today	1	Cake
□	kunj.patel@dal.ca	Spicy	27	2022-07-24	ID1658123...	
□	100		50	20/02/2022	23	oats and c...
□	rt@dal.ca	Faintly bitter, ...	10	today	1	Thepla
□	rh679297@dal.ca	None	2	2022-07-23	ID1658142...	
□	pink@dal.ca	Cake is a flour...	5	today	1	Cake
□	kamap32370@arego...	Well Cooked	9	2022-07-26	ID1658640...	

Figure 26. Screenshot of successful entry of record '[pink@dal.ca](#)' into the foodorder table in DynamoDB.



The screenshot shows the AWS Lambda function 'orderMyFood - Lambda' running in the us-east-1 region. The Lambda function is triggered by an S3 event. The Lambda code is a simple Node.js script that reads a JSON file from S3 and inserts it into a DynamoDB table named 'room'. The table has 11 items, each representing a room with fields like cust\_id, end\_date, price, room\_na..., room\_no, and start\_date.

Items returned (11)						
	cust_id	end_date	price	room_na...	room_no	start_date
□	jenishfly@gmail.com	2022-07-21	5004	Villa	53	2022-07-21
□	rt@dal.com	tomorrow	12120	Queen	1010	today
□	it.fenil@gmail.com	2022-07-27	2900	Delux	500	2022-07-27
□	kunj.patel@dal.ca	2022-08-03	9000	Basic	205	2022-07-27
□	rt@dal.ca	tomorrow	5600	Super King	725	today
□	1001	2022-07-22	7324	King Room	5	2022-07-22
□	jenishp619@gmail.com	2022-07-28	5600	Super King	725	2022-07-28
□	pink@dal.ca	2022-08-30	2700	Super Delux	322	today
□	kamap32370@arego...	2022-07-27	1200	Standard	122	2022-07-27

Figure 27. Screenshot of successful entry of record '[pink@dal.ca](#)' into the room table in DynamoDB.

## 9.4 Message Passing

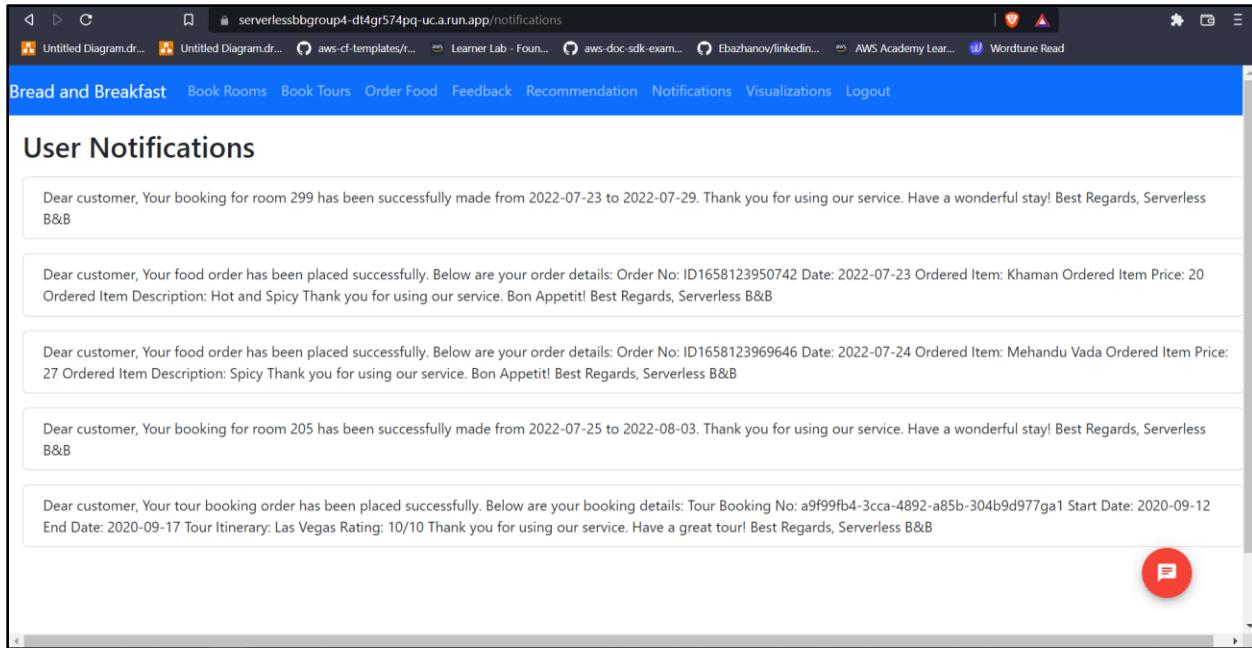


Figure 27 User Notifications

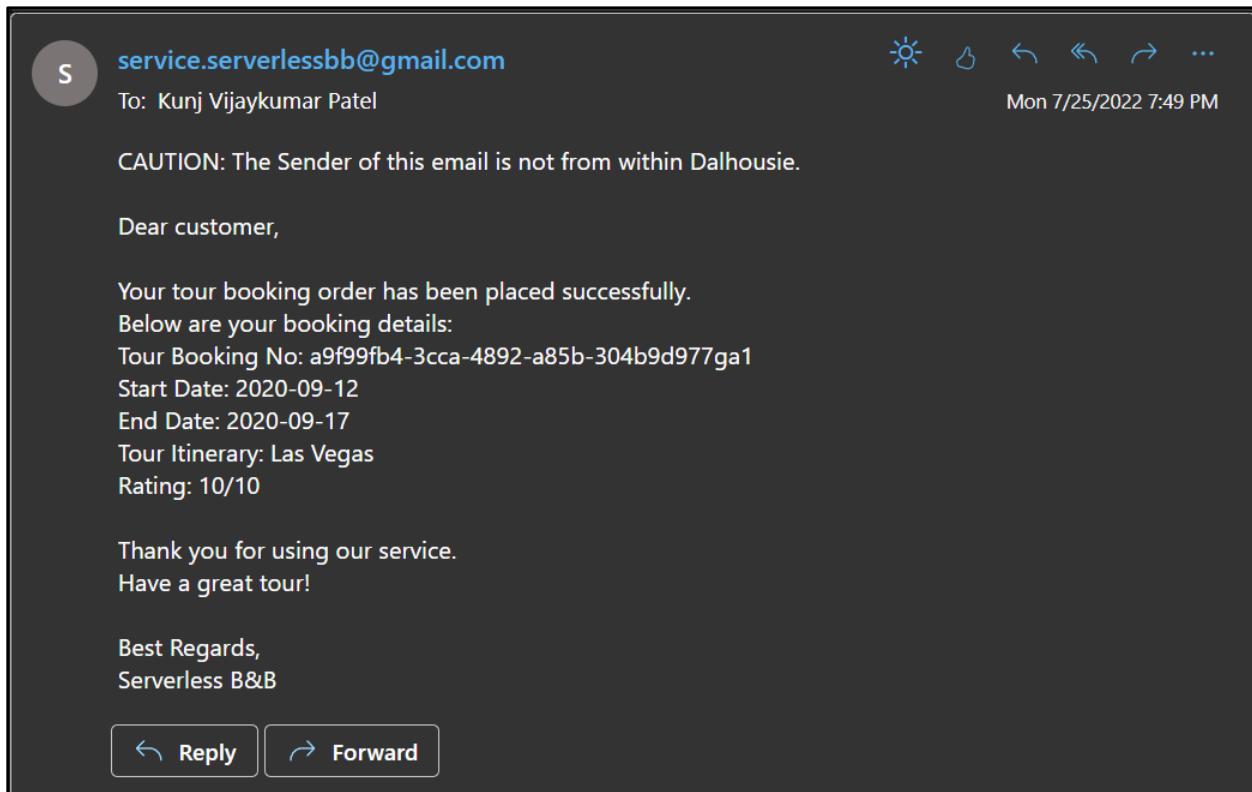


Figure 28 Booking Confirmation Email

The screenshot shows the AWS DynamoDB console with a table named 'serverless\_bb\_notifications'. The table has two columns: 'user\_id' and 'notification'. There are 7 items returned. The notifications contain messages such as 'Dear customer, Your food order has been placed' and 'Your booking for room 299 is confirmed'.

user_id	notification
rh679297@dal.ca	[{"S": "Dear customer,\n\nYour food order has been pl..."}]
tarteyepso@vusra.com	[{"S": "Dear customer,\n\nYour food order has been pl..."}]
it.fenil@gmail.com	[{"S": "Dear customer,\n\nYour food order has been pl..."}]
kunj.patel@dal.ca	[{"S": "Dear customer,\n\nYour booking for room 299 ..."}]
jenishp619@gmail.com	[{"S": "Dear customer,\n\nYour booking for room 122 ..."}]
kamap32370@arego...	[{"S": "Dear customer,\n\nYour booking for room 122 ..."}]
kanepatel8@gmail.com	[{"S": "Dear Customer, Thank you for booking the roo..."}]

Figure 29 Notifications in Table

## 9.5 Machine Learning

- Tour Recommendation

The screenshot shows a web application titled 'Tour Package Recommender'. The page displays a form with the following input fields:

- StaysInWeekendNights: 10
- StaysInWeekNights: 2
- Adults: 4
- Children: 8
- Babies: 1

Figure 30 Tour Package Recommender Front End

The screenshot shows a web browser window titled "React App" with the URL "serverlessbbgroup4-dt4gr574pq-uca.run.app/predict/tourPackage". The page contains several input fields for tour package details:

- StaysInWeekendNights: 2
- Adults\*: 4
- Children\*: 8
- Babies\*: 2
- IsRepeatedGuest (0/1)\*: 1
- RequiredCarParkingSpaces\*: 1

Below these fields are two sets of radio buttons:

- Customer Type:** Transient (unchecked), Transient-Party (unchecked), Group (checked), Contract (unchecked)
- Payment Type:** NoDeposit (unchecked), Refundable (checked), NonRefundable (unchecked)

A large blue button at the bottom right is labeled "PREDICT".

Figure 31 Tour Package Recommender Front End

The screenshot shows a web browser window titled "React App" with the URL "serverlessbbgroup4-dt4gr574pq-uca.run.app/predict/tourPackage". The top navigation bar includes links for "Breakfast and Breakfast", "Book Rooms", "Book Tours", "Order Food", "Feedback", "Recommendation", "Notifications", and "Visualizations".

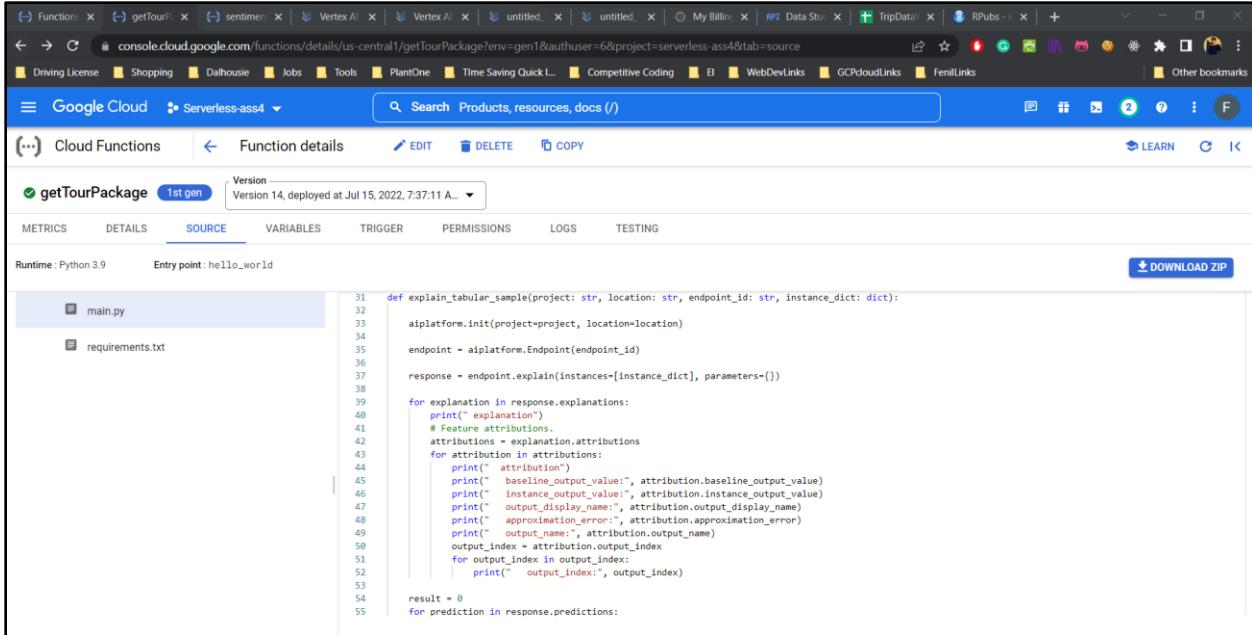
The main content area displays the results of the tour package recommendation:

**Tour Package Recommender**  
**Recommended Tour: Super King**

The same input fields as in Figure 31 are present:

- StaysInWeekendNights\*: 10
- StaysInWeekNights\*: 2
- Adults\*: 4
- Children\*: 8
- Babies\*: 2

Figure 32 Tour Package Recommender Results

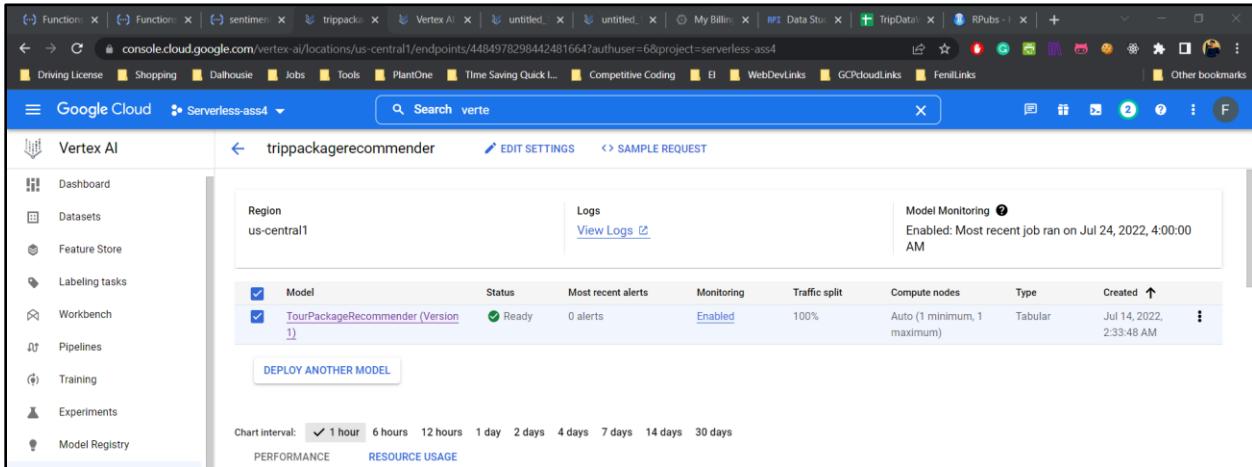


```

31 def explain_tabular_sample(project: str, location: str, endpoint_id: str, instance_dict: dict):
32     aiplatform.init(project=project, location=location)
33
34     endpoint = aiplatform.Endpoint(endpoint_id)
35
36     response = endpoint.explain(instance_dict, parameters={})
37
38     for explanation in response.explanations:
39         print("explanation")
40         # Feature attributions.
41         attributions = explanation.attributions
42         for attribution in attributions:
43             print("attribution")
44             print("baseline_output_value:", attribution.baseline_output_value)
45             print("instance_output_value:", attribution.instance_output_value)
46             print("output_display_name:", attribution.output_display_name)
47             print("approximation_error:", attribution.approximation_error)
48             print("output_name:", attribution.output_name)
49             output_index = attribution.output_index
50             for output_index in output_index:
51                 print("output_index:", output_index)
52
53     result = 0
54
55     for prediction in response.predictions:

```

Figure 33 Tour Package Recommender Cloud Function



Model	Status	Most recent alerts	Monitoring	Traffic split	Compute nodes	Type	Created
TourPackageRecommender (Version 1)	Ready	0 alerts	Enabled	100%	Auto (1 minimum, 1 maximum)	Tabular	Jul 14, 2022, 2:33:48 AM

Figure 34 Tour Package Recommended Vertex AI

- Sentiment Analysis

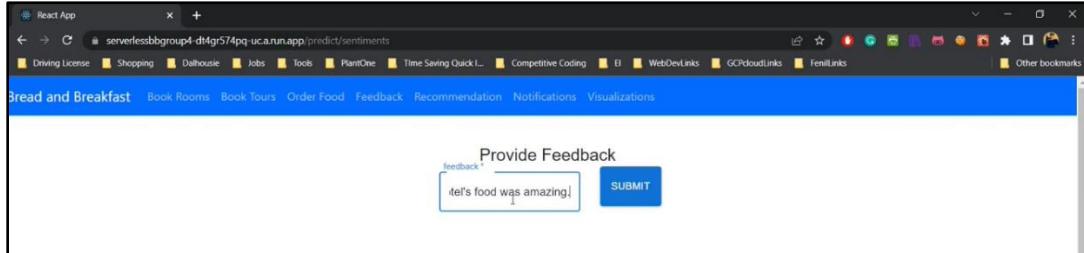


Figure 35 Sentiment Analysis Front End

A screenshot of the AWS Lambda Functions interface. The URL is "us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1&item\_explorer?initialLogKey=%26Table%3Dfeedbacks". The left sidebar shows the "DynamoDB" service and lists tables: caesaripher, feedbacks (selected), foodDetails, foodorder, room, roomDetails, securityanswer, Tour\_Booking, and Users-farm. The main pane displays the "Items returned (21)" table with columns: date, feedback, and prediction. The table contains 21 rows of sentiment analysis results from the "feedbacks" table.

	date	feedback	prediction
70@arego...	165864513...	This hotel's food was amazing.	0.899999761581421
70@arego...	165864294...	This is Bad.	-0.1000000149011612
70@arego...	165864261...	This Hotel was Amazing.	0.800000011920929
70@arego...	165863974...	This Food is Awesome.	0.899999761581421
mail.com16...	165861596...	Las Vegas Trip was Awesome.	0.899999761581421
mail.com16...	165861103...	Hotel Was awesome.	0.800000011920929
mail.com16...	165817065...	Restaurant is awesome.	0.899999761581421
mail.com16...	165814396...	LasVegas was amazing.	0.899999761581421

Figure 36 Sentiment Analysis Results in AWS DynamoDB

A screenshot of the Google Cloud Functions interface. The URL is "console.cloud.google.com/functions/details/us-central1/sentimentAnalysis?env=gen1&authuser=6&project=serverless-ass4&tab=source". The function name is "sentimentAnalysis" (1st gen). The code editor shows the "main.py" file with Python 3.9 code for sentiment analysis using the Google Cloud Language API. The "requirements.txt" file is also listed.

```

16     from google.cloud import language_v1
17     import six
18
19     def analyze(content):
20         """Run a sentiment analysis request on text."""
21         client = language_v1.LanguageServiceClient()
22
23         if isinstance(content, six.binary_type):
24             content = content.decode("utf-8")
25
26         type_ = language_v1.Document.Type.PLAIN_TEXT
27         document = {"type": type_, "content": content}
28
29         response = client.analyze_sentiment(request={"document": document})
30         sentiment = response.document_sentiment
31         res = str(sentiment.score)
32
33         # If sentiment.score >= 0:
34         #   res = res + "Positive: The Sentence \"content\"\nhas a sentiment score of \"str(sentiment.score)\"
35         # else:
36         #   res = res + "Negative: The Sentence \"content\"\nhas a sentiment score of \"str(sentiment.score)\"
37
38         # res = res + "\nOverall Sentiment: score of \"str(sentiment.magnitude)\" magnitude."
39         print("Score: {}").format(sentiment.score)
40         print("Magnitude: {}").format(sentiment.magnitude)

```

Figure 37 Sentiment Analysis Cloud Function

## 9.6 Web Application Building and Hosting

### Room Booking:

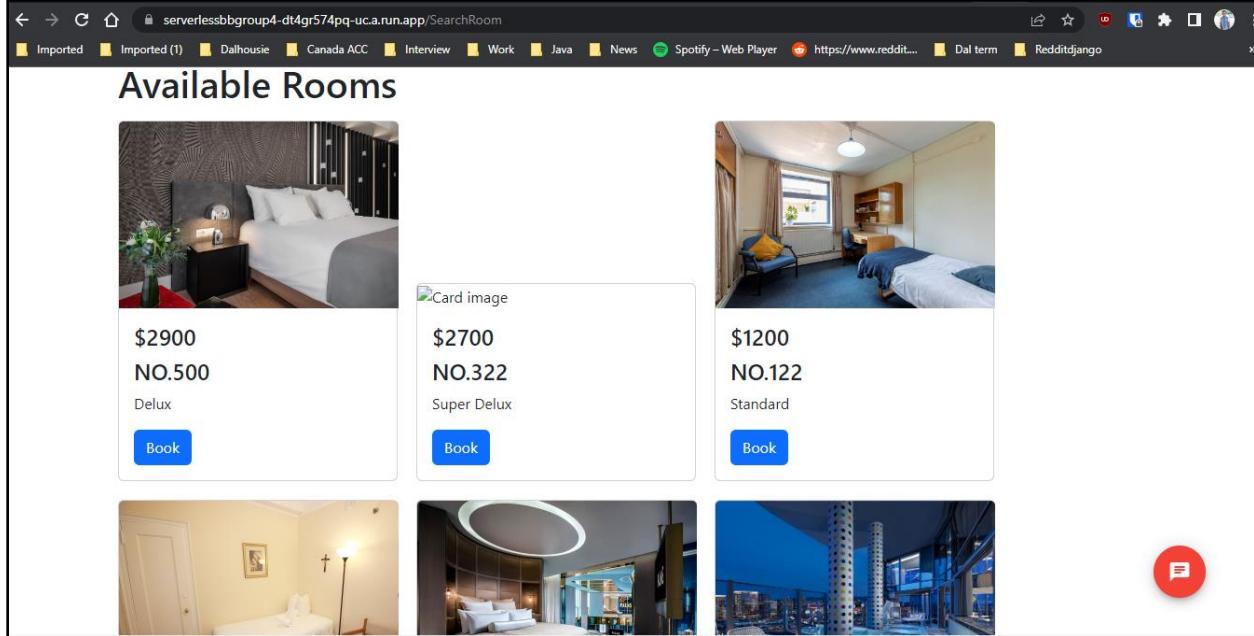


Figure 38 Room booking home page

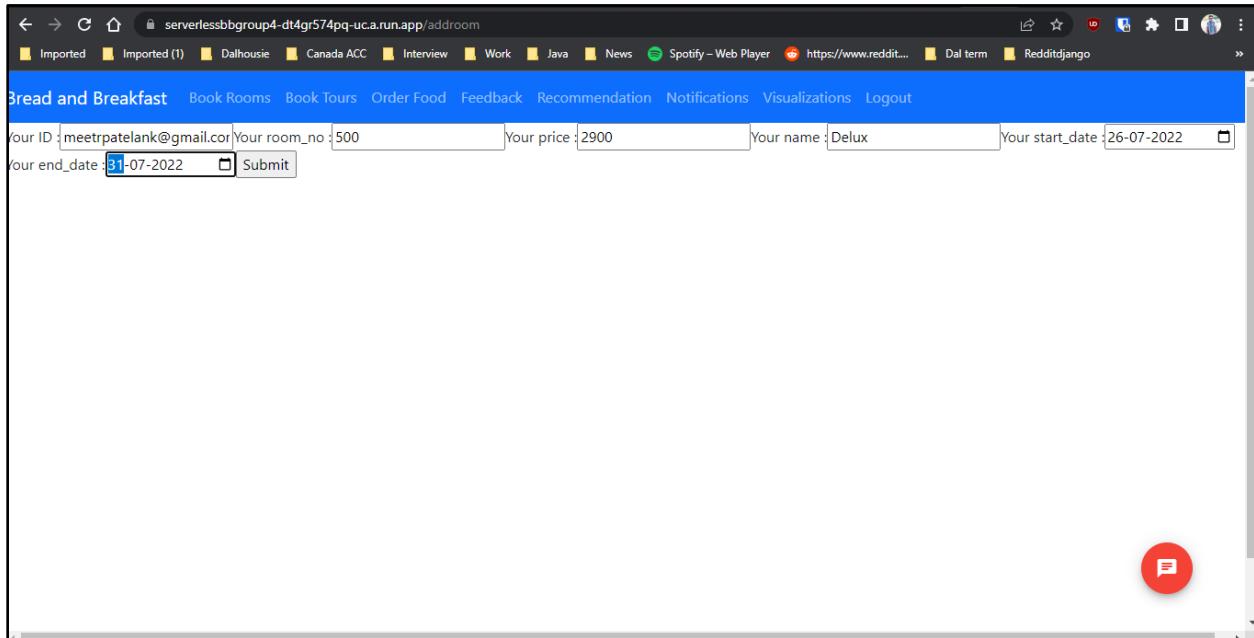


Figure 39 : User enters the room booking details

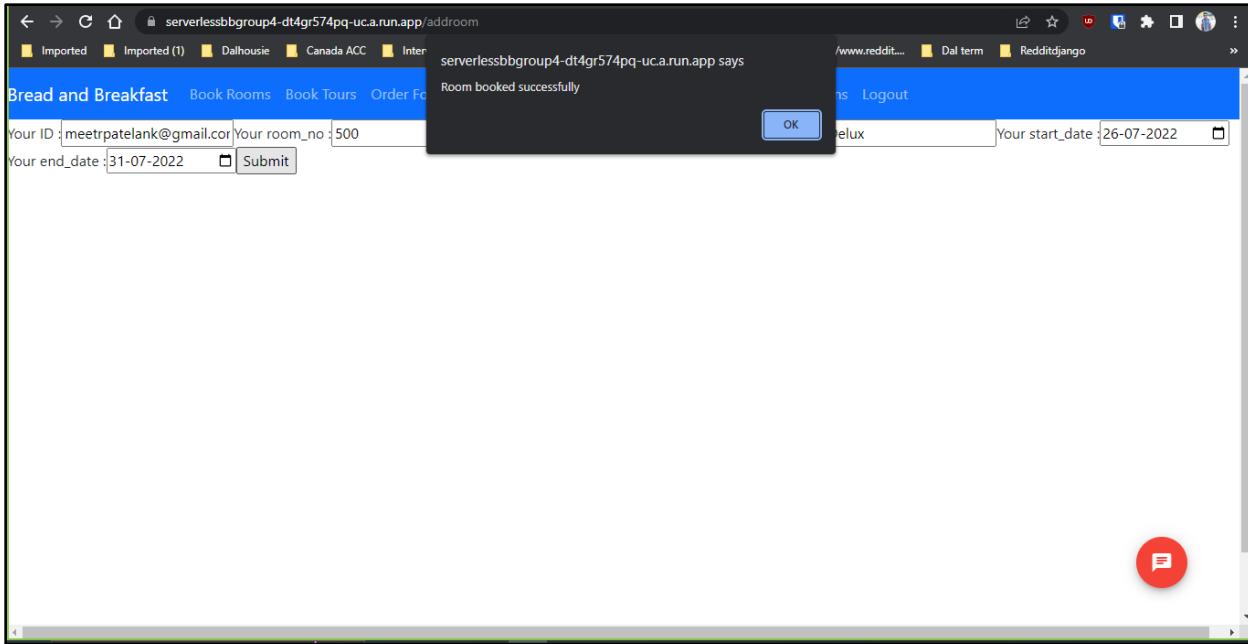


Figure 40 Room booked successfully message

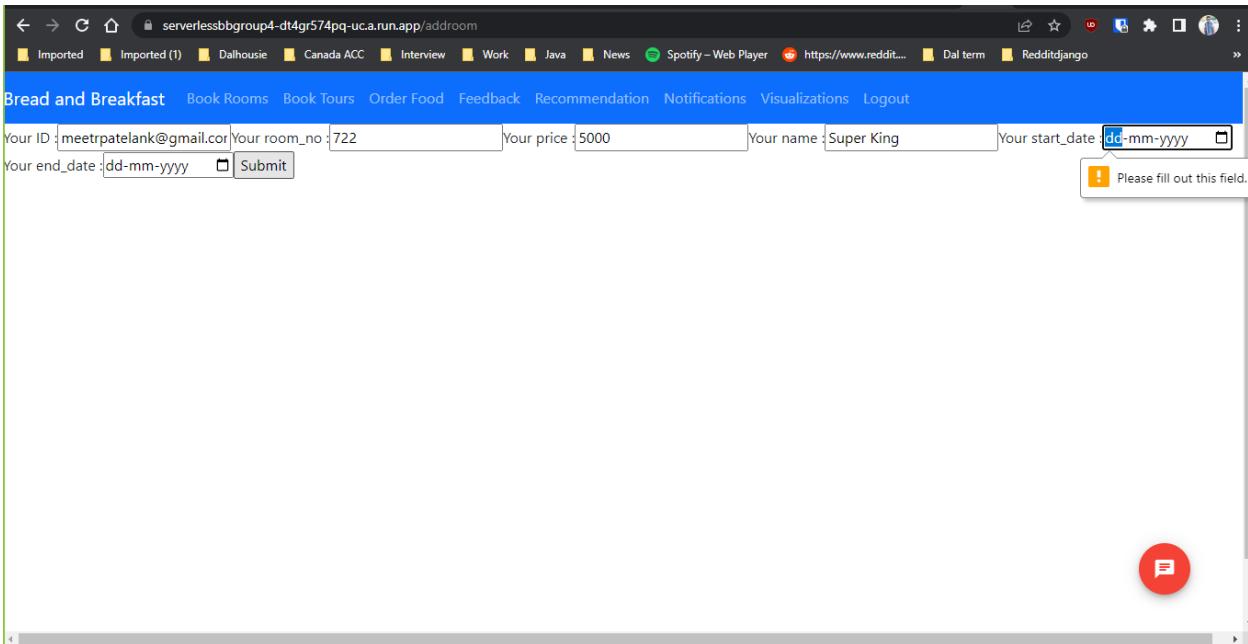


Figure 41 Validations for the fields in the room booking form

The screenshot shows a web browser window with a blue header bar containing various links like 'Imported', 'Dalhouse', 'Canada ACC', etc. Below the header is a navigation bar for 'Bread and Breakfast' with links for 'Book Rooms', 'Book Tours', 'Order Food', 'Feedback', 'Recommendation', 'Notifications', 'Visualizations', and 'Logout'. The main content area contains several input fields: 'Your ID' (meetrpatelank@gmail.com), 'Your room\_no' (722), 'Your price' (5000), 'Your name' (Super King), and 'Your start\_date' (26-07-2022). Below these is a field for 'Your end\_date' with the placeholder 'dd-mm-yyyy'. A red validation message box is overlaid on the page, containing a yellow exclamation mark icon and the text 'Please fill out this field.' A small red circular icon with a white 'F' is visible in the bottom right corner of the browser window.

Figure 42 Validations for the fields in the room booking form

The screenshot shows the AWS DynamoDB console interface. The left sidebar has a tree view with 'DynamoDB' selected, showing 'Dashboard', 'Tables' (with 'Explore items'), 'PartiQL editor', 'Backups', 'Exports to S3', 'Reserved capacity', and 'Settings'. Under 'Tables', 'room' is selected. The main panel shows a table titled 'Items returned (11)' with columns: cust\_id, end\_date, price, and room\_na... (partially visible). One row is selected, showing 'meetrpatelank@gmail.com' as the cust\_id, '2022-07-31' as the end\_date, '2900' as the price, and 'Delux' as the room name. There are buttons for 'Run' and 'Reset' at the top of the table view.

cust_id	end_date	price	room_na...
meetrpatelank@gmail.com	2022-07-31	2900	Delux
domramrap@gmail.com	2022-07-17	7324	King Room
jenishpfly@gmail.com	2022-07-21	5004	Villa
rt@dal.com	tomorrow	12120	Queen
it_fan@outlook.com	2022-07-27	3000	Delux
it_fan@outlook.com	2022-07-27	3000	Delux
it_fan@outlook.com	2022-07-27	3000	Delux
it_fan@outlook.com	2022-07-27	3000	Delux
it_fan@outlook.com	2022-07-27	3000	Delux
it_fan@outlook.com	2022-07-27	3000	Delux

Figure 43 Room booking details in the Amazon DynamoDB table

## Tour Booking:

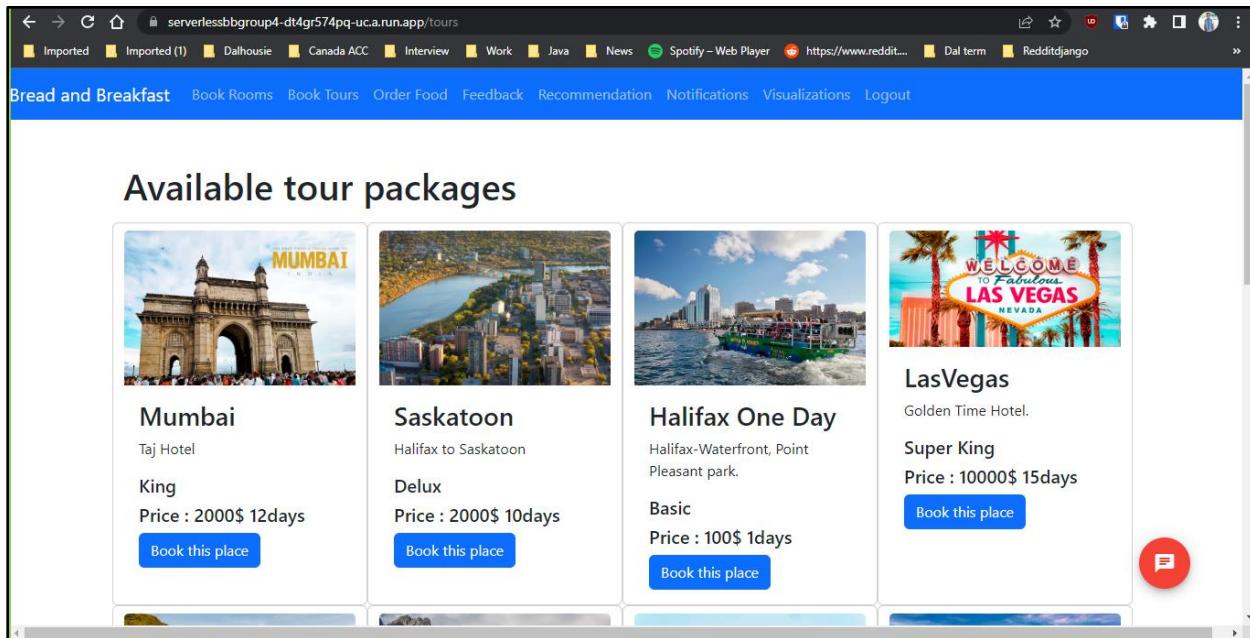


Figure 44 Tour package home page for user

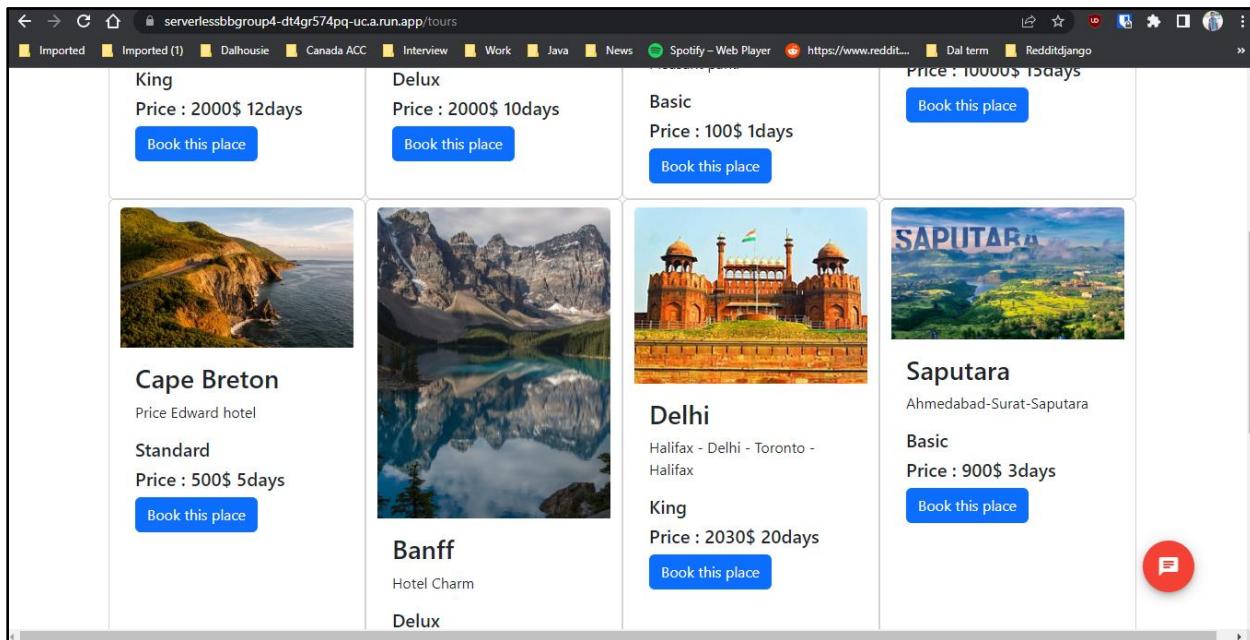


Figure 45 Tour package home page for user

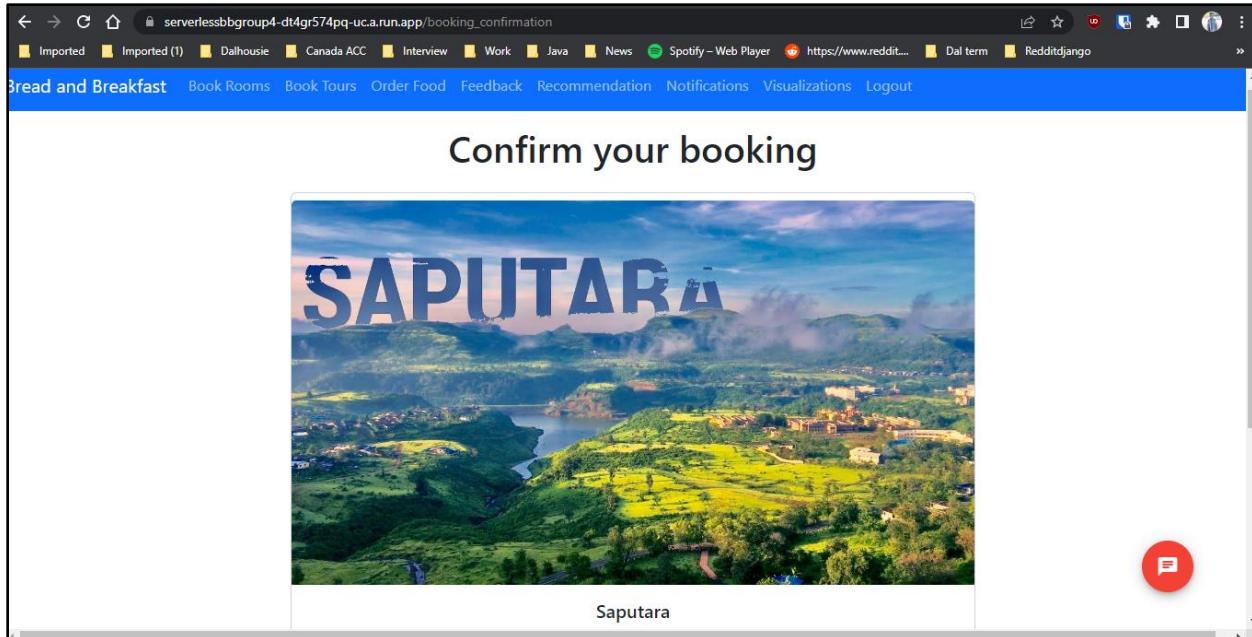


Figure 46 Booking confirmation check for the user

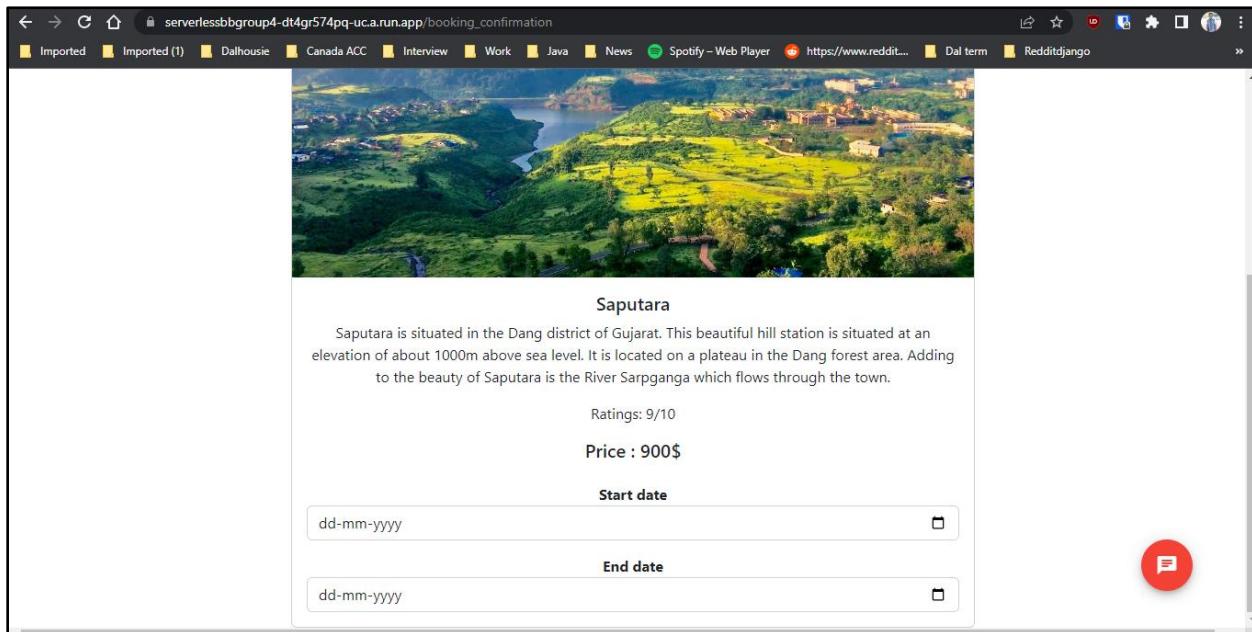


Figure 47 Booking date confirmation for the user

The screenshot shows the AWS DynamoDB console. On the left, the navigation pane is open for the 'DynamoDB' service, showing options like 'Dashboard', 'Tables', 'Update settings', and 'Explore items'. Under 'Tables', 'Tour\_Booking' is selected. The main area displays the 'Items returned (6)' table. Each item has columns for 'cust\_id', 'end\_Date', 'Itinerary', and 'rating'. The data is as follows:

	cust_id	end_Date	Itinerary	rating
1	meetrpatelank@gmail.com...	2022-12-20	Mumbai	10
2	jenishpfly@gmail.com...	2022-07-23	Surat	10/10
3	ajju1632000@gmail.c...	2022-07-14	Surat	10/10
4	it.fenil@gmail.com	2022-09-17	Cape Breton	7
5	kunj.patel@dal.ca	2022-08-09	Banff	10
6	200	2022-07-31	Surat	10/10

Figure 48 Data inserted in Amazon DynamoDB table for Tour booking

## Food Order :

The screenshot shows the 'Food Order' application's home page. The top navigation bar includes links for 'read and Breakfast', 'Book Rooms', 'Book Tours', 'Order Food', 'Feedback', 'Recommendation', 'Notifications', 'Visualizations', and 'Logout'. Below the navigation, the heading 'Available Breakfast' is displayed. Three breakfast items are listed in a grid:

- Chocolate**: Price: \$2, Name: Chocolate. A 'Book' button is present.
- Bread Butter**: Price: \$7, Name: Bread Butter. A 'Book' button is present.
- Poutine**: Price: \$10, Name: Poutine. A 'Book' button is present.

Figure 49 Food Order home page

The screenshot shows a web browser window with the URL `serverlessbbgroup4-dt4gr574pq-uc.a.run.app/orderfood#`. The page title is "Bread and Breakfast". The main content area is titled "Confirm your order". It contains three input fields: "Quantity" (with value "3"), "Special Instruction" (with value "Hot"), and "Date of Order" (with placeholder "dd-mm-yyyy"). To the right of the Date field is a "Place Order" button. A red circular icon with a white speech bubble is visible in the bottom right corner of the page.

Figure 50 Order confirmation details form for the user

The screenshot shows the same web browser window as Figure 50. The "Date of Order" field now has a validation error message: "Please fill out this field." The rest of the form and the red circular icon are identical to Figure 50.

Figure 51 Validations in the order food form

The screenshot shows the AWS DynamoDB console. On the left, the navigation pane lists tables: feedbacks, foodDetails, foodorder (selected), room, roomDetails, securityanswer, Tour\_Booking, and Users-farm. The main area displays the contents of the foodorder table. A message at the top says "Completed Read capacity units consumed: 0.5". Below it, a table titled "Items returned (15)" shows 15 rows of data. The columns are: cust\_id, description, item\_price, and order\_Date. The first few rows are: RUHI@dal.ca (description: Cake is a flour..., item\_price: 5, order\_Date: today), sky@dal.ca (description: Faintly bitter, ..., item\_price: 10, order\_Date: today), 38da4cad-4e44-4d20... (description: , item\_price: 15, order\_Date: 2022-07-14), and meetrpatelank@gmail.com (description: Hot, item\_price: 15, order\_Date: 2022-07-27).

Figure 52 Food order details stored in Amazon DynamoDB table

## Lambda Functions used in the Project:

The screenshot shows the AWS Lambda console. The left sidebar shows "Lambda > Functions". The main area displays a table titled "Functions (38)". A search bar at the top says "Filter by tags and attributes or search by keyword". The table has columns: Function name, Description, Package type, Runtime, and Last modified. Some of the function names listed are: roombackend, lex-web-ui-cloudrunUnauth-C-CodeBuildStarterLambda-ZvPSUdwGsPM, lex-web-ui-auth-Cognitold-CognitoUserPoolUpdatesFu-cJerW8NmQO46, lex-web-ui-auth-Cognitolden-CleanStackNameFunction-IVXP29qsXoI, caesarcipher, bookMyRoom, Tour\_Booking, and getFeedback.

Figure 53 Lambda Functions for all the modules

## APIs created in the project:

The screenshot shows the AWS API Gateway console with a list of APIs. The table has columns for Name, Description, ID, Protocol, Endpoint type, and Created. The APIs listed are:

Name	Description	ID	Protocol	Endpoint type	Created
caesarkey		yc1uf17dt5	HTTP	Regional	2022-07-24
foodorder		nbobbbkh492	HTTP	Regional	2022-07-13
login-backend-gateway		d1bi5isu09	REST	Regional	2022-07-03
roomapi		r3g1kc5x8e	HTTP	Regional	2022-07-13
securityquestion		pcg45mec4	HTTP	Regional	2022-07-23
sns-order		4j7sqduetg	REST	Regional	2022-07-19
tour		ddrxhagqw4	HTTP	Regional	2022-07-13

Figure 54 API created in AWS for all the modules

## Room Booking Backend Process:

The screenshot shows the AWS Lambda function overview page for 'roombackend'. The function has three triggers from the API Gateway. The right sidebar displays details such as Description, Last modified (9 days ago), Function ARN (arn:aws:lambda:us-east-1:651609363924:function:roombackend), and Function URL (Info).

Figure 55 API gateway trigger added for room booking backend lambda function

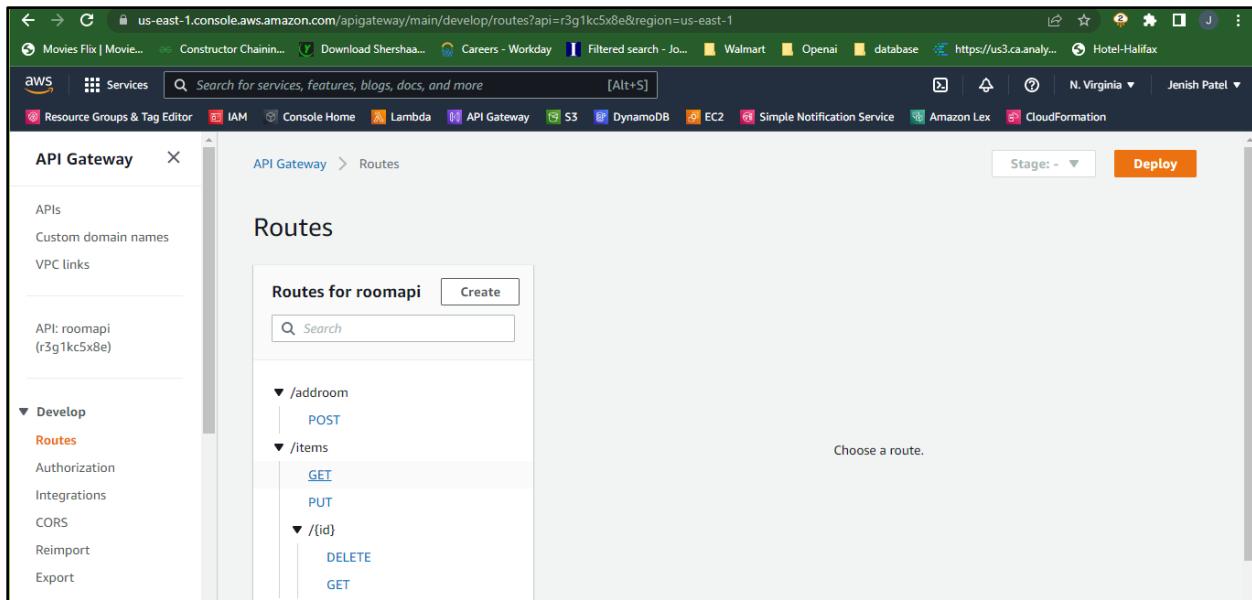


Figure 56 GET, POST, PUT and DELETE API requests for the routes

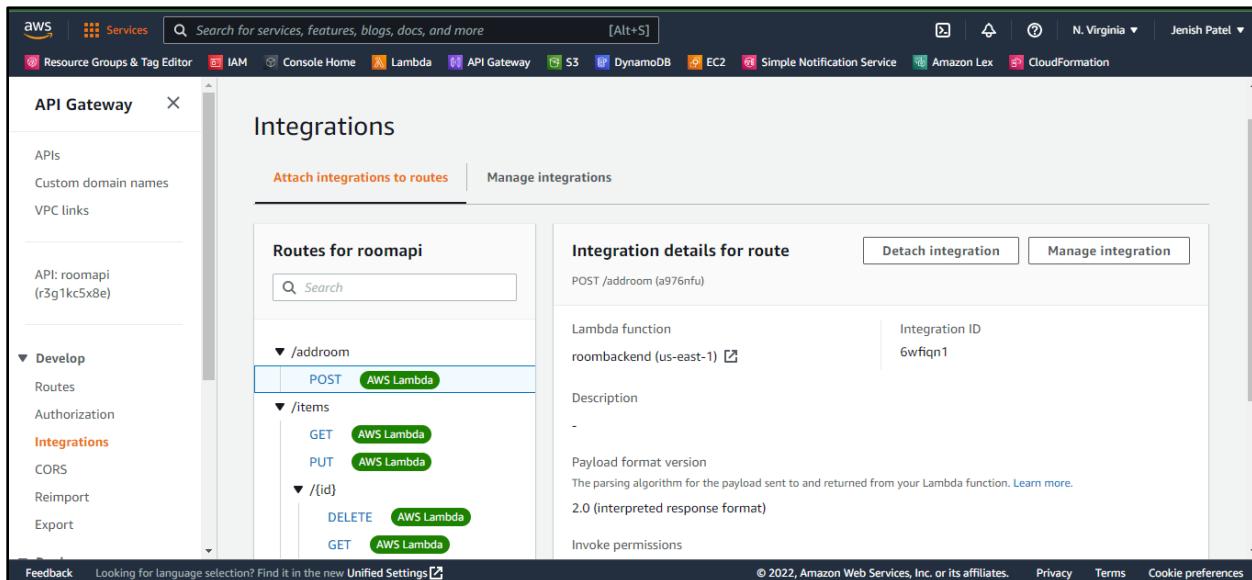


Figure 57 Integrations of the requests with AWS Lambda function

## Food Order Backend Process:

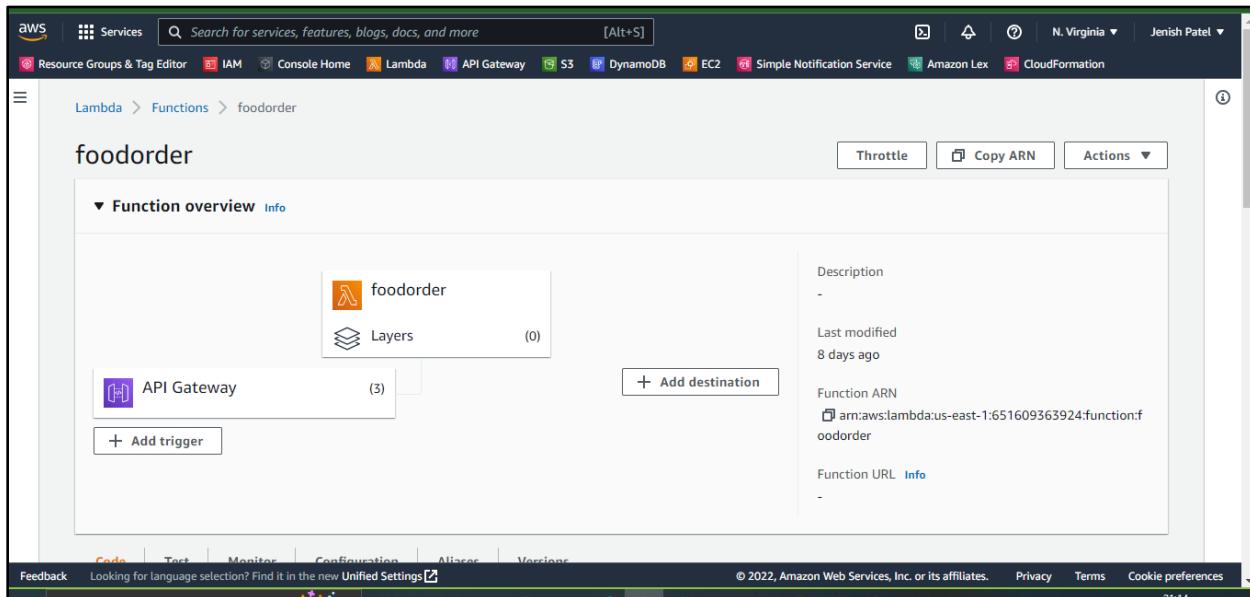


Figure 58 API gateway trigger for food order lambda function

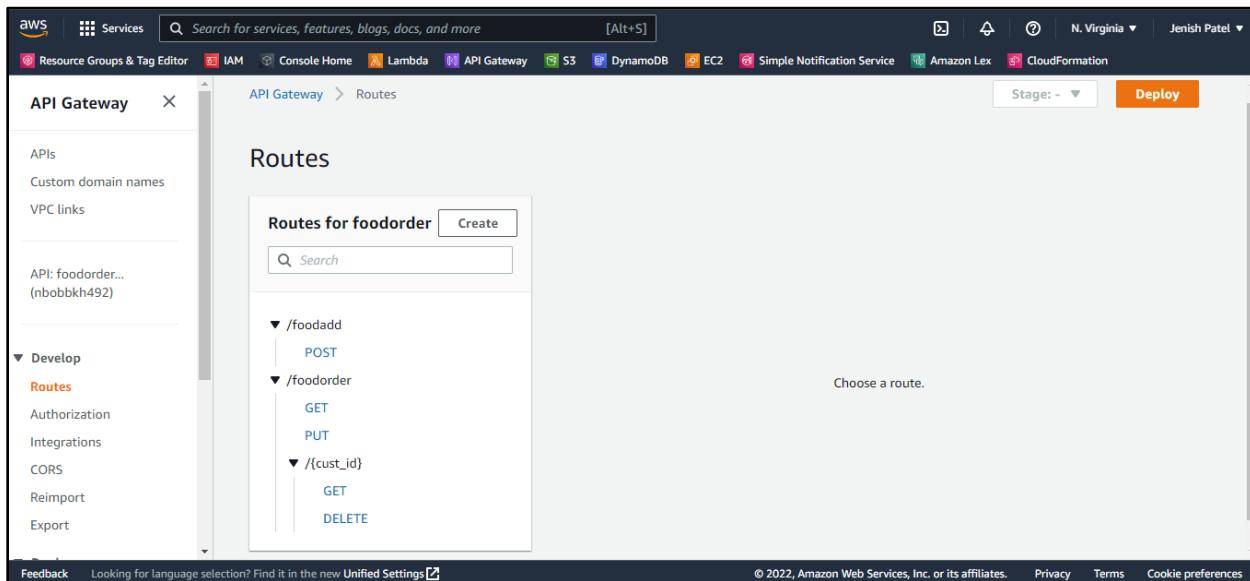


Figure 59 Routes for the food order lambda function API's

The screenshot shows the AWS API Gateway Integrations page. On the left, a sidebar menu includes 'APIs', 'Custom domain names', 'VPC links', and a 'Develop' section with 'Routes', 'Authorization', 'Integrations' (which is selected), 'CORS', 'Reimport', and 'Export'. The main content area is titled 'Integrations' and shows 'Routes for foodorder'. It lists three routes: POST /foodadd (AWS Lambda), GET /foodorder (AWS Lambda), and GET /{cust\_id} (AWS Lambda). To the right, 'Integration details for route' for the POST /foodadd route are displayed, showing it's a Lambda function named 'foodorder' from 'us-east-1' with an integration ID '0iv4kp5'. Other tabs include 'Attach integrations to routes' and 'Manage integrations'.

Figure 60 Integrations of the requests with AWS Lambda function in food order

## Tour Booking Backend Process:

The screenshot shows the AWS Lambda Functions page. The 'Tour\_Booking' function is selected. In the 'Function overview' section, it shows an icon for 'Tour\_Booking', 'Layers (0)', and an 'API Gateway' trigger. Below this, there are tabs for 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. The 'Code source' tab is active, displaying the 'index.js' file content:

```

1  const AWS = require('aws-sdk');
2  const dynamo = new AWS.DynamoDB.DocumentClient();
3
4  exports.handler = async (event, context) => {
5    try {
6      let statusCode = 200;
7      const headers = {
8        'Content-Type': "application/json"
9      };
10     ...
11   }
12 }

```

Figure 61 API gateway trigger for the Tour Booking lambda function

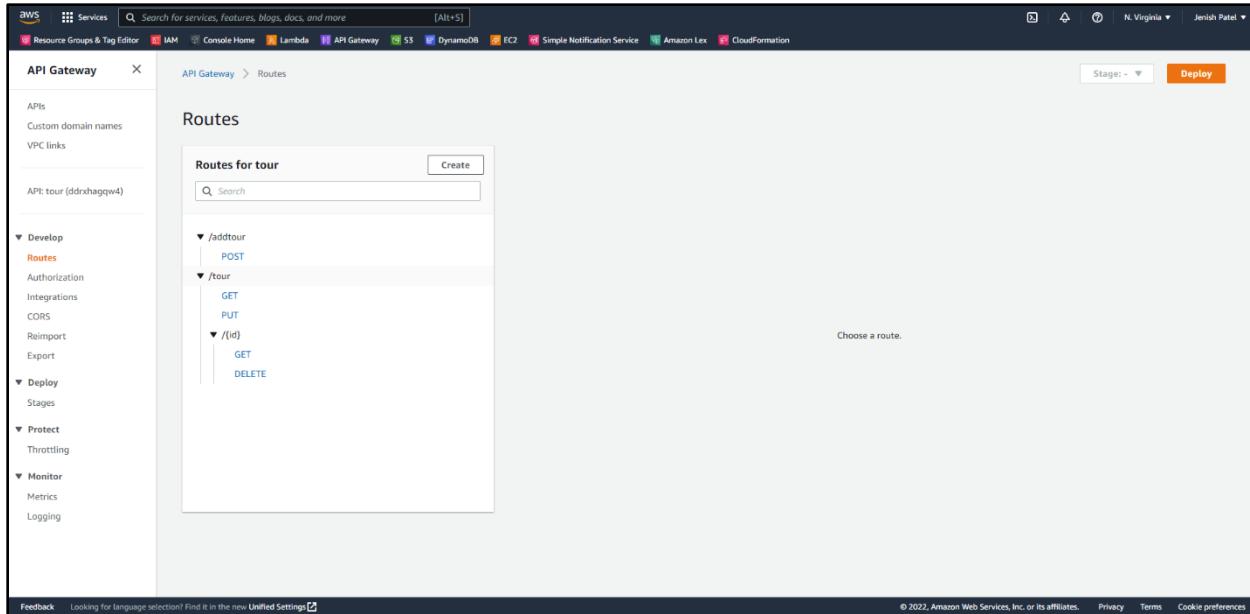


Figure 62 Routes for tour booking

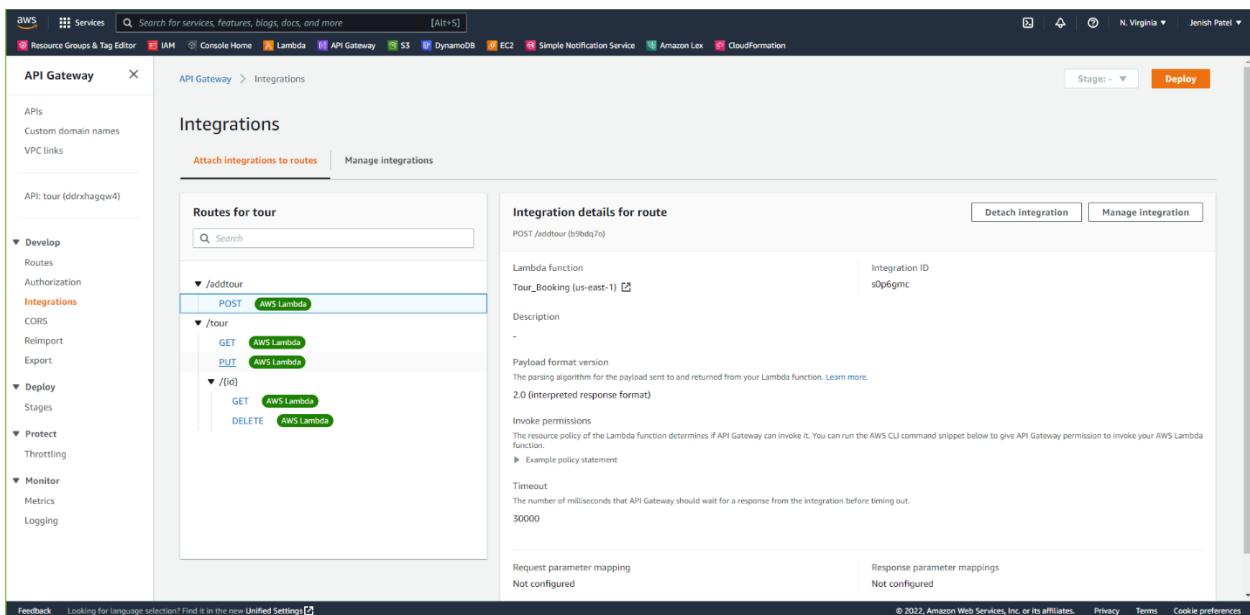


Figure 63 Integrations of the requests with AWS Lambda function in tour booking

## Caesar Cipher Backend Process:

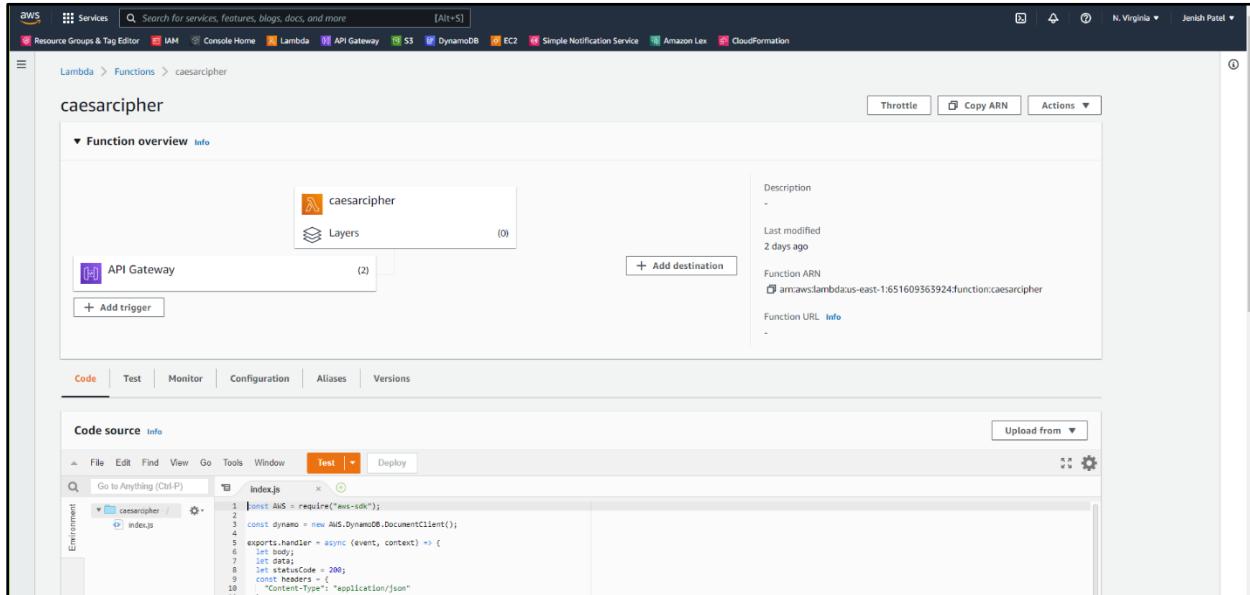


Figure 64 Caesar Cipher Lambda function and API gateway trigger

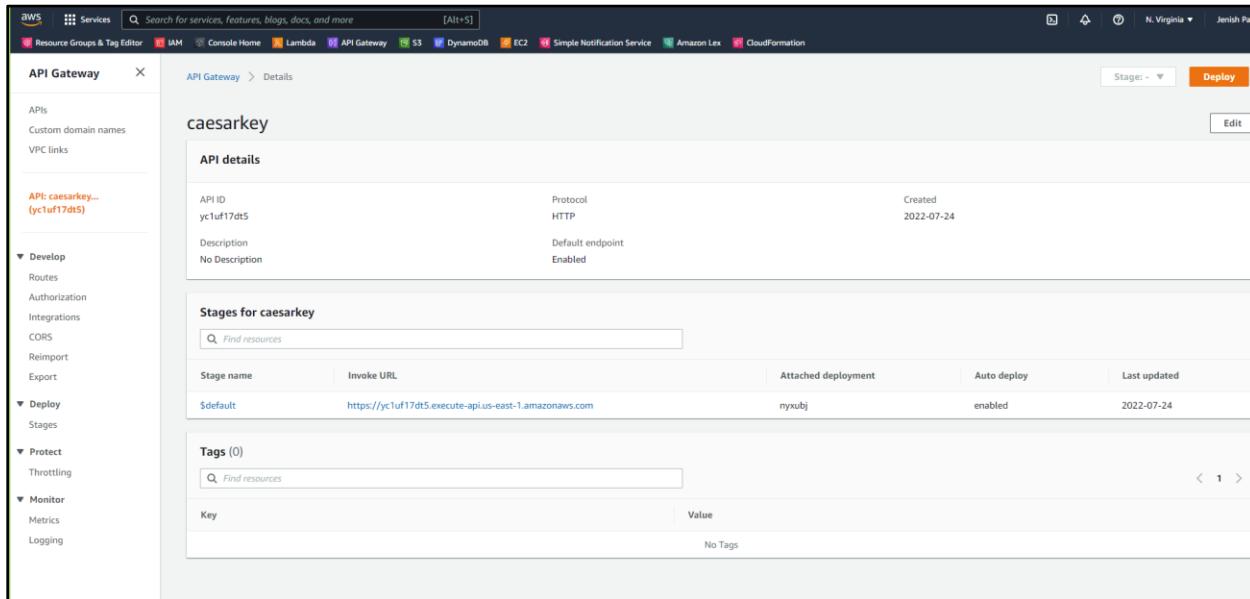


Figure 65 API details of Caesar cipher lambda function

The screenshot shows the AWS API Gateway Integrations page. On the left, the navigation menu is expanded to show 'Integrations' under 'Develop'. The main area displays 'Routes for caesarkey' with two entries: a POST route for '/addkey' and a GET route for '/(cust\_id)'. The right panel shows 'Integration details for route' for the POST /addkey route, which points to a Lambda function named 'caesarcipher' (us-east-1). The Lambda function has an integration ID of b0vb8gg. Other details include payload format version 2.0, invoke permissions, a timeout of 30000, and request/response parameter mappings.

Figure 66 Integrations of the requests with AWS Lambda function in caesar cipher

## User Feedback Backend Process:

The screenshot shows the AWS Lambda Functions page. It displays a single function named 'getFeedback'. The 'Function overview' tab is selected, showing basic details like the ARN, last modified (8 days ago), and function URL. The 'Code source' tab is active, showing the code editor with an 'index.js' file containing the following code:

```

const AWS = require("aws-sdk");
const dynamo = new AWS.DynamoDB.DocumentClient();
exports.handler = async (event, context) => {
  let body;
  let statusCode = 200;
  const headers = {
    "Content-Type": "application/json"
  };
  try {
    const params = {
      TableName: "FeedbackTable",
      Key: event.params.id
    };
    const result = await dynamo.get(params).promise();
    body = JSON.stringify(result.Item);
  } catch (err) {
    statusCode = 400;
    body = JSON.stringify({ error: err.message });
  }
  return {
    statusCode,
    headers,
    body
  };
}
  
```

Figure 67 feedback lambda function

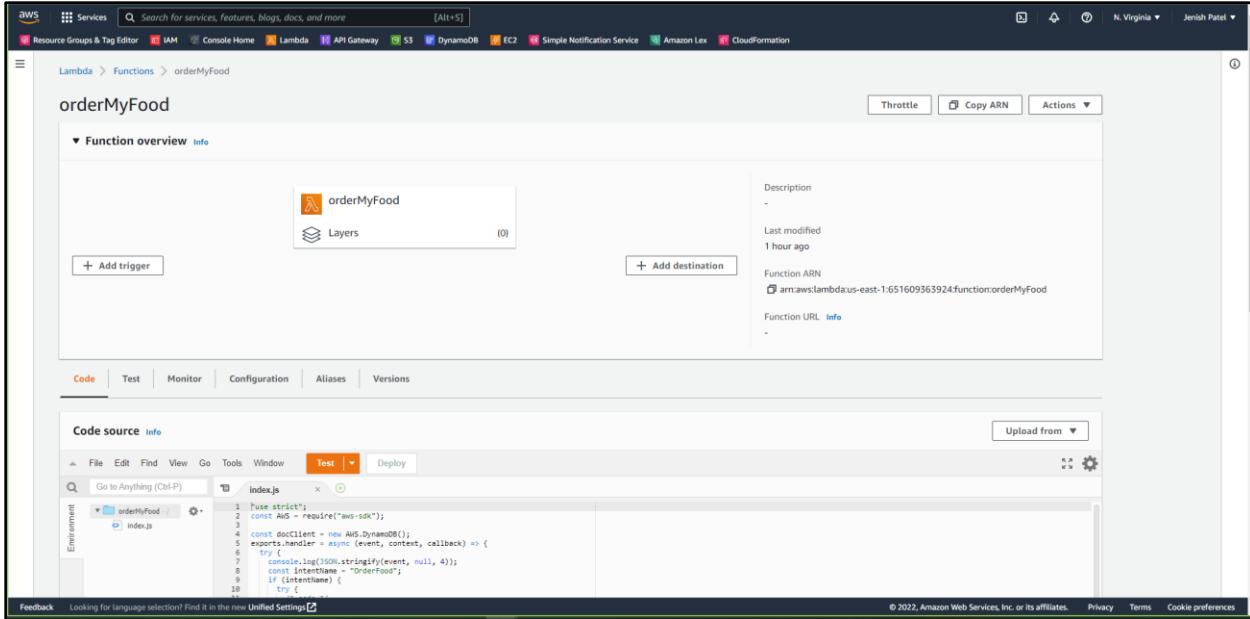


Figure 68 orderMyfood lambda function

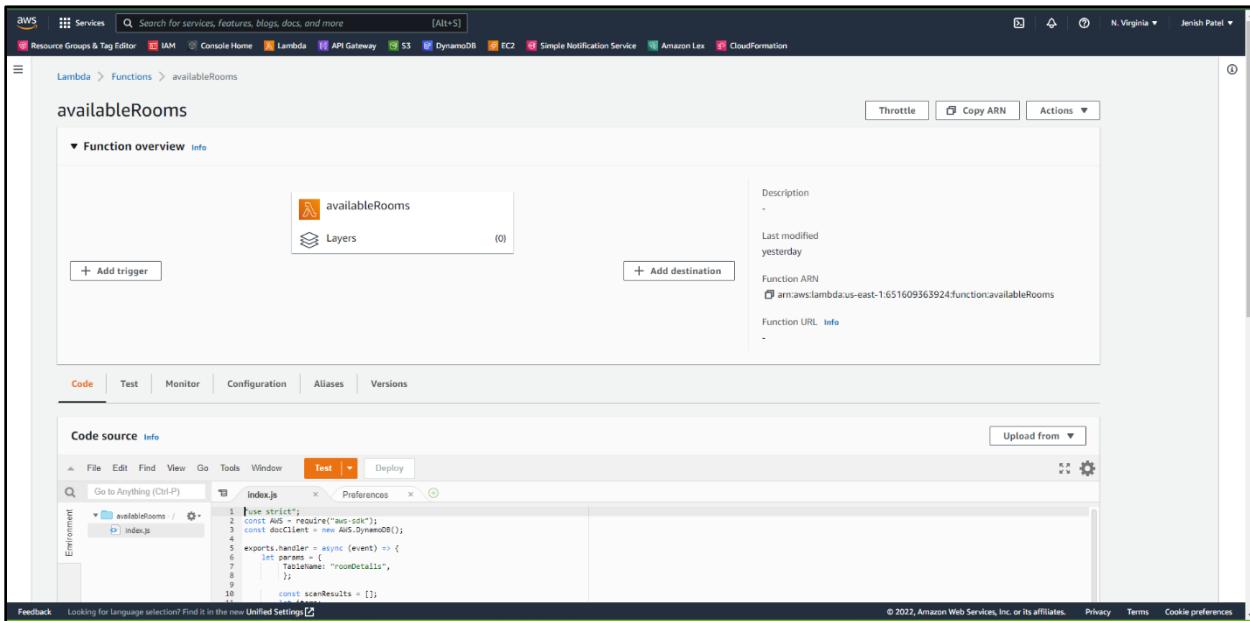


Figure 69 available rooms lambda function

## Security Question Answer Backend Process:

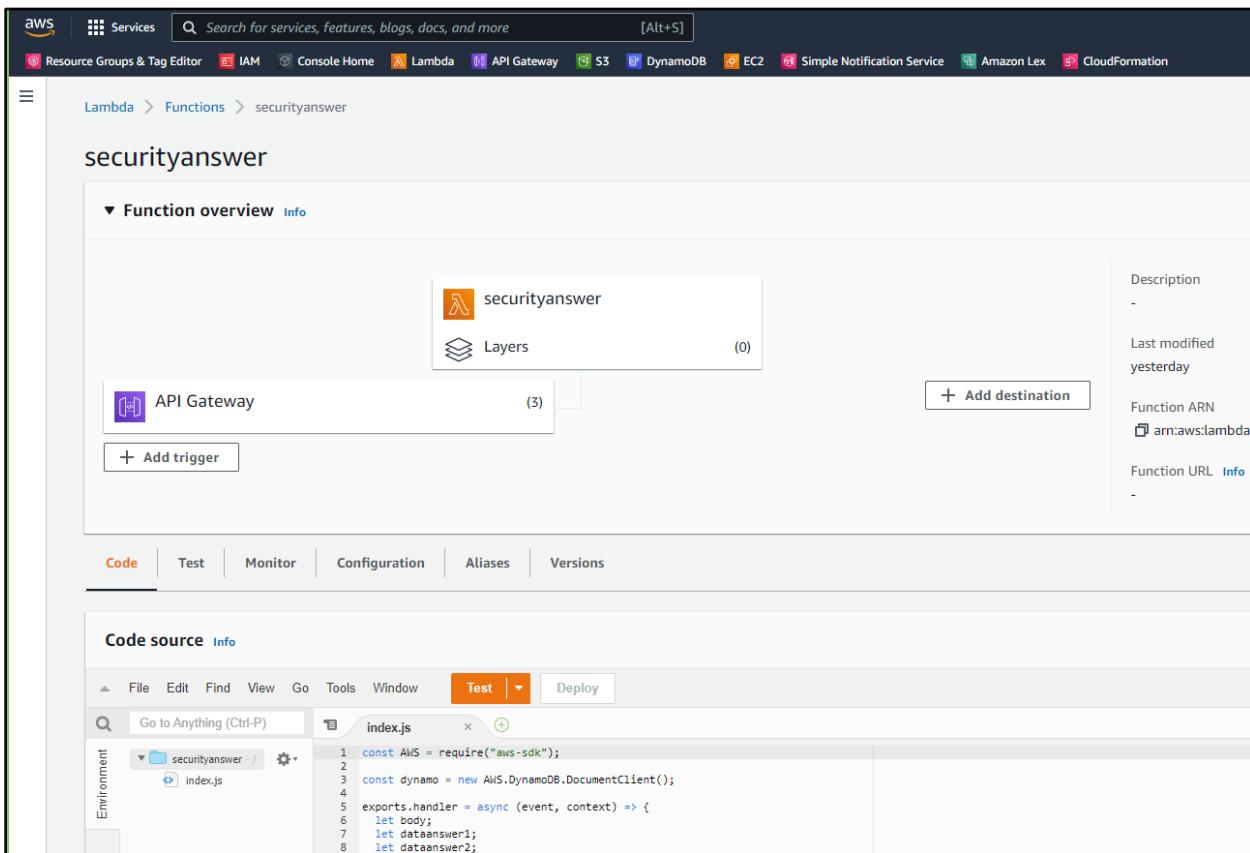


Figure 70 API gateway trigger for security answer lambda function

The screenshot shows the AWS API Gateway Integrations page. On the left, the navigation menu is expanded to show 'Integrations' under 'Develop'. The main area displays a tree structure of routes. A route for 'securityquestion' is selected, showing its integration details. The integration is a Lambda function named 'securityanswer' (us-east-1) with the ARN 'gp9c0y6'. The payload format version is set to 2.0 (interpreted response format). The invoke permissions section shows an example policy statement. The timeout is set to 30000 milliseconds. Request and response parameter mappings are both set to 'Not configured'.

Figure 71 Integrations of the requests with security answer lambda function

The screenshot shows the AWS Lambda Functions page. A function named 'navigateWebsite' is selected. The 'Function overview' tab is active, showing the function ARN 'arn:aws:lambda:us-east-1:651609363924:function:navigateWebsite' and a function URL. The 'Code' tab is selected, showing the code editor with an 'index.js' file. The code is as follows:

```

1  'use strict';
2  const AWS = require('aws-sdk');
3
4  exports.handler = async (event) => {
5
6    let Options = event['interpretations'][0]['intent']['slots'][0]['options'][0]['originalValue'];
7
8    let result = '';
9    switch(Options){
10      case 'login':
11        break;
12    }
13
14    return {
15      statusCode: 200,
16      body: JSON.stringify(result)
17    };
18}

```

Figure 72 Website navigation lambda function

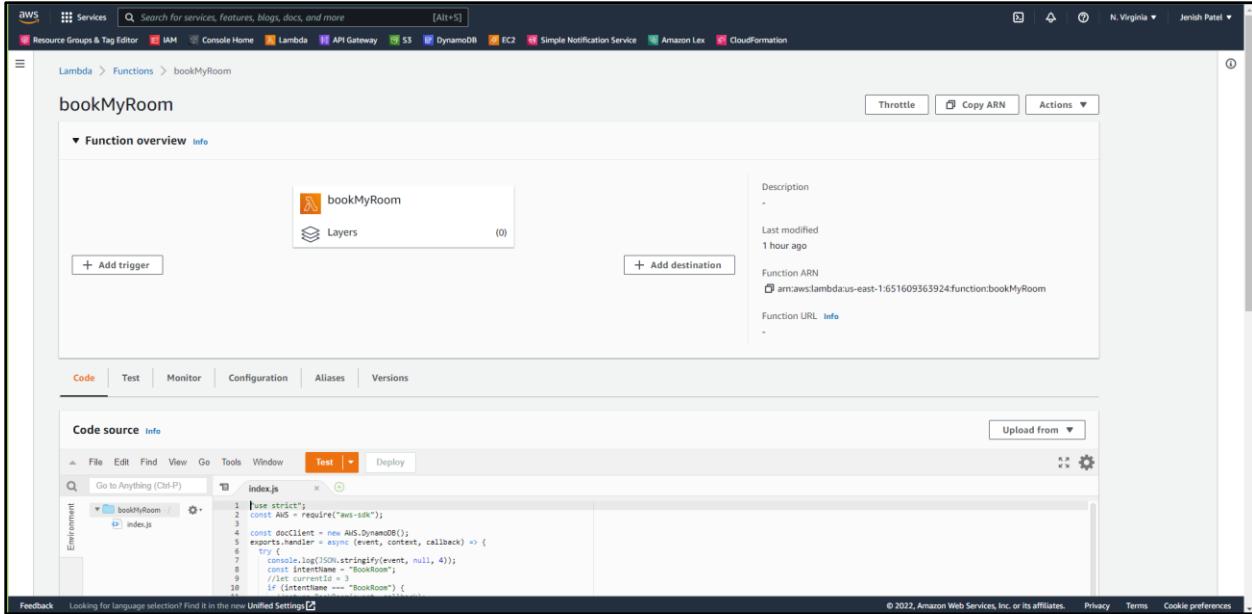


Figure 73 room booking lambda function

## Tables stored in DynamoDB for the project:

Tables (9) <small>Info</small>									
	Name	Status	Partition key	Sort key	Indexes	Read capacity mode	Write capacity mode	Size	Table class
	caesarcipher	Active	cust_id (\$)	-	0	Provisioned with auto scaling (1)	Provisioned with auto scaling (1)	193 bytes	DynamoDB Standard
	feedbacks	Active	feedbackId (\$)	-	0	Provisioned with auto scaling (1)	Provisioned with auto scaling (1)	3 kilobytes	DynamoDB Standard
	foodDetails	Active	foodName (\$)	-	0	Provisioned with auto scaling (1)	Provisioned with auto scaling (1)	1.8 kilobytes	DynamoDB Standard
	foodorder	Active	cust_id (\$)	-	0	Provisioned with auto scaling (1)	Provisioned with auto scaling (1)	1.7 kilobytes	DynamoDB Standard
	room	Active	cust_id (\$)	-	0	Provisioned with auto scaling (1)	Provisioned with auto scaling (1)	951 bytes	DynamoDB Standard
	roomDetails	Active	roomNumber (\$)	-	0	Provisioned with auto scaling (1)	Provisioned with auto scaling (1)	573 bytes	DynamoDB Standard
	securityanswer	Active	cust_id (\$)	-	0	Provisioned with auto scaling (1)	Provisioned with auto scaling (1)	283 bytes	DynamoDB Standard
	Tour_Booking	Active	cust_id (\$)	-	0	Provisioned with auto scaling (1)	Provisioned with auto scaling (1)	643 bytes	DynamoDB Standard

Figure 74 Active tables in DynamoDB

DynamoDB > Items > caesarcipher

**Tables (9)**

- Any table tag
- Find tables by table name
- caesarcipher (selected)
- feedbacks
- foodDetails
- foodorder
- room
- roomDetails
- securityanswer
- Tour\_Booking
- Users-farm

**caesarcipher**

Scan/Query items

Scan/Query a table or index

Scan | Query | caesarcipher

Filters

Run | Reset

Completed Read capacity units consumed: 0.5

Items returned (6)

	cust_id	key
<input type="checkbox"/>	terdomafye@vusra.com	21
<input type="checkbox"/>	brplate2210@gmail.c...	19
<input type="checkbox"/>	jeetpadia123@gmail....	19
<input type="checkbox"/>	tarteyepso@vusra.com	11

Figure 75 Caesar Cipher table in DynamoDB

DynamoDB > Items > feedbacks

**Tables (9)**

- Any table tag
- Find tables by table name
- caesarcipher
- feedbacks (selected)
- foodDetails
- foodorder
- room
- roomDetails
- securityanswer
- Tour\_Booking
- Users-farm

**feedbacks**

Scan/Query items

Scan/Query a table or index

Scan | Query | feedbacks

Filters

Run | Reset

Completed Read capacity units consumed: 0.5

Items returned (24)

	feedbackId	date	feedback	prediction	user_id
<input type="checkbox"/>	it.fenil@gmail.com16...	165808808...	This is good.	0.80000001...	it.fenil@gmail.com
<input type="checkbox"/>	meetrpatelank@gmail...	165810203...	good	0.80000001...	meetrpatelank@gmail.com
<input type="checkbox"/>	kamap32370@arego...	165863974...	This Food is...	0.89999997...	kamap32370@aregods.com
<input type="checkbox"/>	it.fenil@gmail.com16...	165808731...	Discount sy...	0.89999997...	it.fenil@gmail.com

Figure 76 User feedback table in dynamodb

DynamoDB > Items > foodDetails

**foodDetails**

Scan/Query items

Scan/Query a table or index

Scan    Query    foodDetails

Run    Reset

Completed Read capacity units consumed: 0.5

Items returned (11)

Actions	Create item	
Pasta	Pasta is a type of f...	20
Poutine	Poutine, a Canada...	10
Cake	Cake is a flour conf...	5
Poha	Poha is one of Indi...	15

Figure 77 Food details table in dynamodb

DynamoDB > Items > foodorder

**foodorder**

Scan/Query items

Scan/Query a table or index

Scan    Query    foodorder

Run    Reset

Completed Read capacity units consumed: 0.5

Items returned (15)

Actions	Create item				
RUHI@dal.ca	Cake is a flour...	5	today	1	Cake
sky@dal.ca	Faintly bitter, ...	10	today	1	Thepla
38da4cad-4e44-4d20...		15	2022-07-14	2	Poha

Figure 78 Food order table in dynamodb

DynamoDB > Items > room

**Tables (9)**

- Any table tag
- Find tables by table name
- caesarcipher
- feedbacks
- foodDetails
- foodorder
- room
- roomDetails
- securityanswer
- Tour\_Booking
- Users-farm

**room**

Scan/Query items

Scan/Query a table or index

Scan Query room

Filters

Run Reset

Completed Read capacity units consumed: 0.5

Items returned (11)

	cust_id	end_date	price	room_na...	room_no	start_date_
	meetrpatelank@gmail...	2022-07-31	2900	Delux	500	2022-07-26
	domramrap@gmail.com	2022-07-17	7324	King Room	5	2022-07-17
	jenishfly@gmail.com	2022-07-21	5004	Villa	55	2022-07-13
	rt@dal.com	tomorrow	12120	Queen	1010	today

Figure 79 room details table in dymaodb

aws Services Search for services, features, blogs, docs, and more [Alt+S]

Resource Groups & Tag Editor IAM Console Home Lambda API Gateway S3 DynamoDB EC2 Simple Notification Service Amazon Lex CloudFormation N. Virginia Jenish Patel

**DynamoDB**

Dashboard Tables Update settings Explore items PartiQL editor New Backups Exports to S3 Reserved capacity Settings New

DAX Clusters Subnet groups Parameter groups Events

Any table tag Find tables by table name

caesarcipher feedbacks foodDetails foodorder room roomDetails securityanswer Tour\_Booking Users-farm

**roomDetails**

Scan/Query items

Scan/Query a table or index

Scan Query roomDetails

Filters

Run Reset

Completed Read capacity units consumed: 0.5

Items returned (11)

	roomNumber	roomName	roomPrice
	1010	Queen	12120
	102	Basic	800
	303	King	2200
	725	Super King	5600
	299	Delux	1029
	205	Basic	9000

Figure 80 roomdetails table in dynamodb

cust_id	answer1	answer2
terandomaye@vusra.com	jk	jk
brpatel2210@gmail.com	SGV	SGV
jeepadia123@gmail.com	SGV	Tommy
tareyepso@vusra.com	svg	jimmy
it.fenil@gmail.com	abc	abc
jenishp619@gmail.com	kv	kv

Figure 81 security answer table in dynamodb

cust_id	end_Date	Itinerary	rating	start_Date	tour_id
meetrpatelank@gmail.com	2022-12-20	Mumbai	10	2022-12-12	3541eac4-0396-4a77-a17f-205ec404b896
jenishpfly@gmail.com	2022-07-25	Surat	10/10	2022-07-16	a9ff9fb4-3cca-4892-a85b-304b9d977fa
aju1632000@gmail.com	2022-07-14	Surat	10/10	2022-07-10	89a63771-aaba-4d11-901c-daf26055d545
it.fenil@gmail.com	2022-09-17	Cape Breton	7	2022-09-12	65ea19af-85dc-4717-9d84-131555e14a66
kunj.patel@dal.ca	2022-08-09	Banff	10	2022-08-01	072e242d-1aff-45ac-90b5-effe557c7d6
200	2022-07-31	Surat	10/10	2022-07-23	3527b85b-3689-4cb0-9d47-ff8fa39a0ab9

Figure 82 Tour booking details table in dynamodb

## 9.7 Other Essential Module (Report/Visualization)

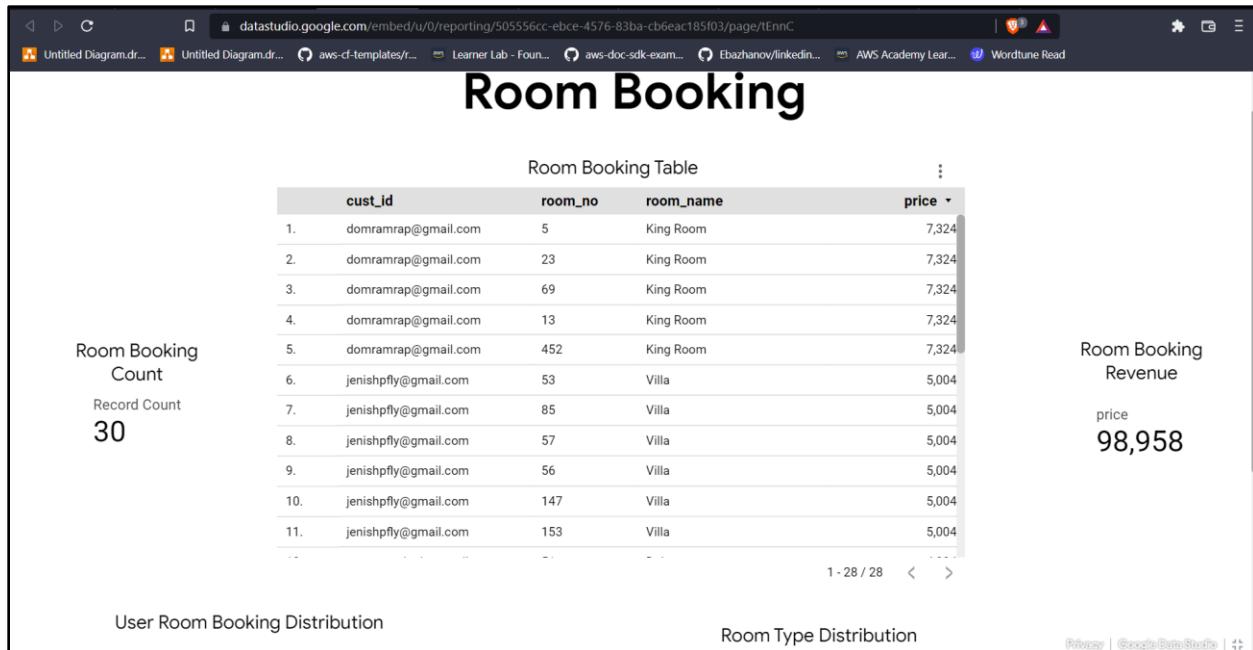


Figure 83 Room Bookings 1

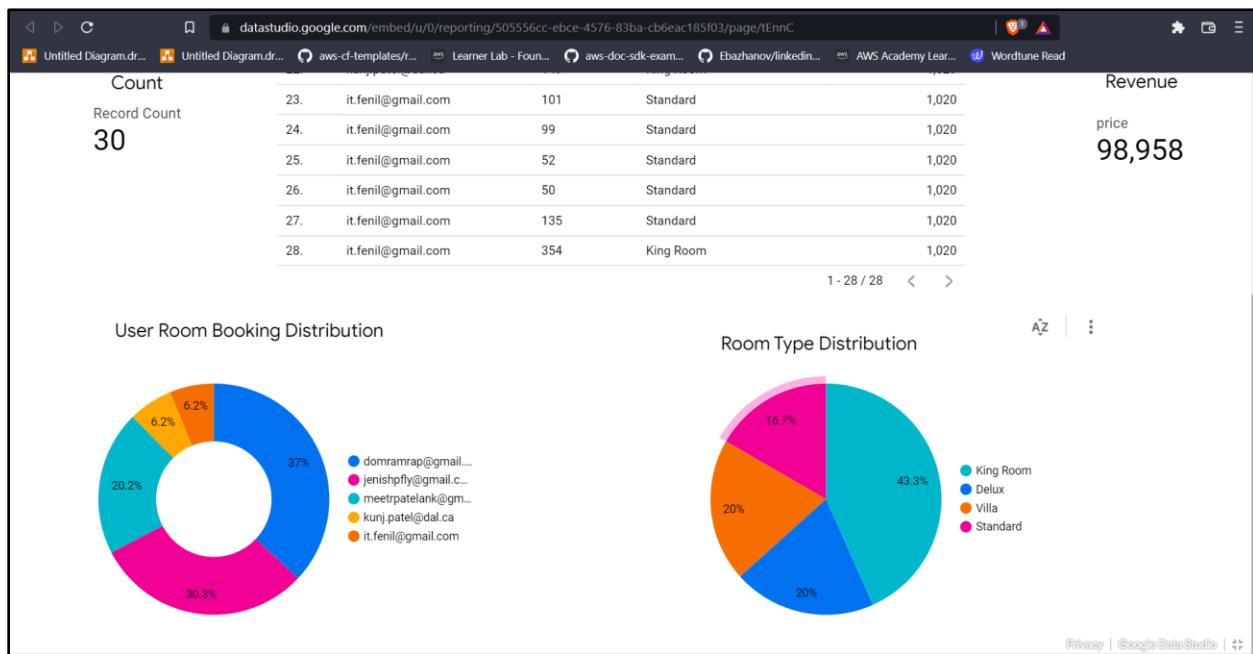


Figure 84 Room Bookings 2

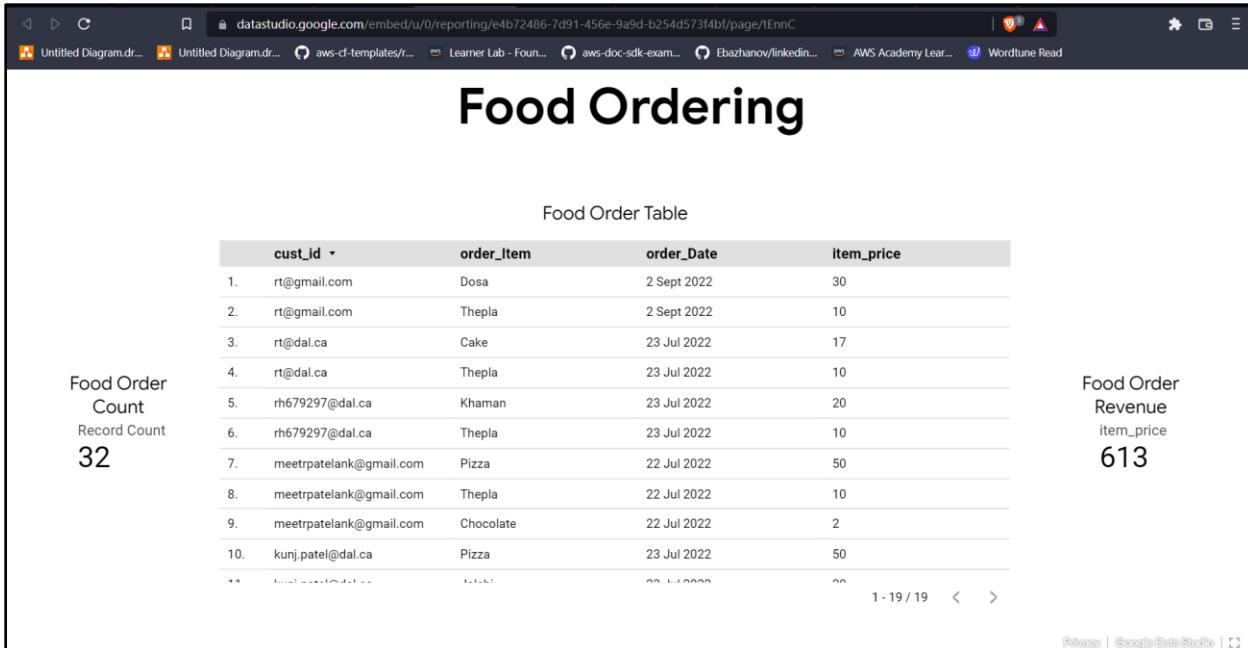


Figure 85 Food Orderings 1

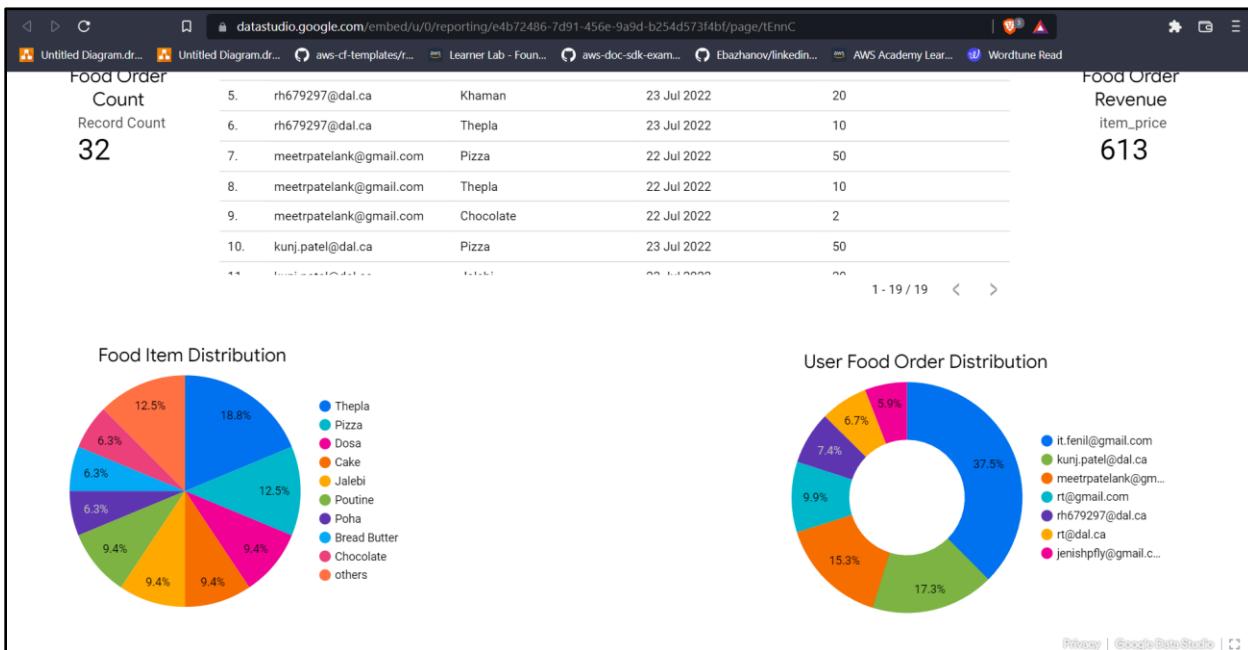
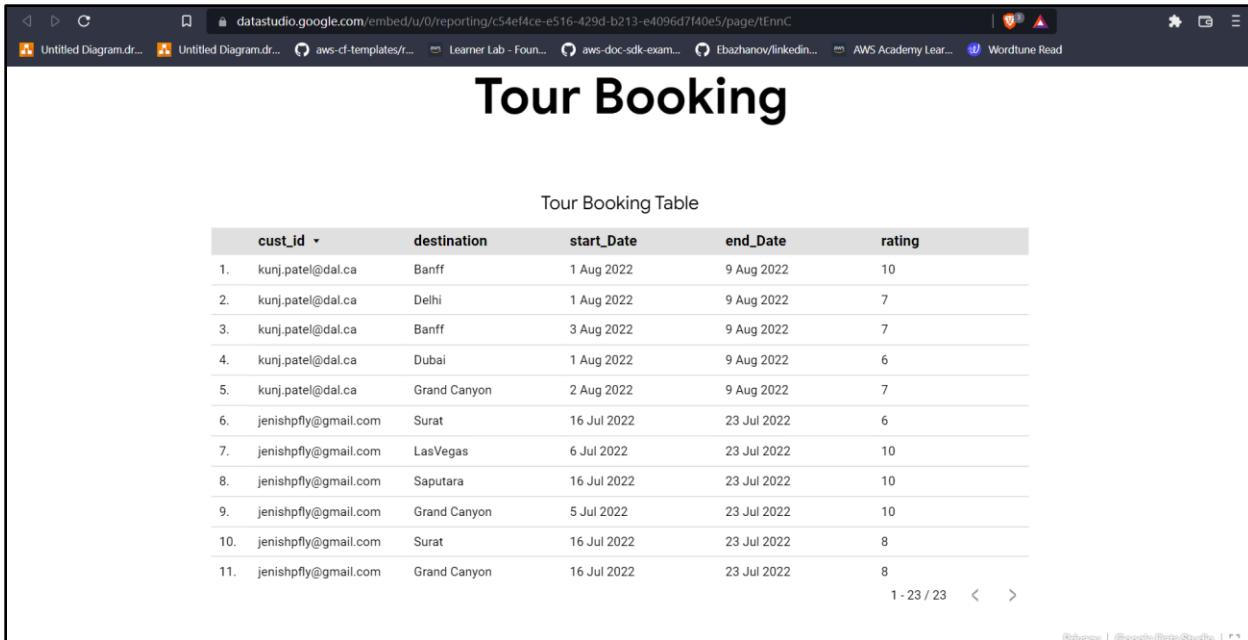


Figure 86 Food Orderings 2



The screenshot shows a Google Data Studio report titled "Tour Booking". At the top, there is a title "Tour Booking" and below it, a subtitle "Tour Booking Table". The table has columns: cust\_id, destination, start\_Date, end\_Date, and rating. The data consists of 11 rows. The last row shows pagination "1 - 23 / 23".

cust_id	destination	start_Date	end_Date	rating
1. kunj.patel@dal.ca	Banff	1 Aug 2022	9 Aug 2022	10
2. kunj.patel@dal.ca	Delhi	1 Aug 2022	9 Aug 2022	7
3. kunj.patel@dal.ca	Banff	3 Aug 2022	9 Aug 2022	7
4. kunj.patel@dal.ca	Dubai	1 Aug 2022	9 Aug 2022	6
5. kunj.patel@dal.ca	Grand Canyon	2 Aug 2022	9 Aug 2022	7
6. jenishpfly@gmail.com	Surat	16 Jul 2022	23 Jul 2022	6
7. jenishpfly@gmail.com	LasVegas	6 Jul 2022	23 Jul 2022	10
8. jenishpfly@gmail.com	Saputara	16 Jul 2022	23 Jul 2022	10
9. jenishpfly@gmail.com	Grand Canyon	5 Jul 2022	23 Jul 2022	10
10. jenishpfly@gmail.com	Surat	16 Jul 2022	23 Jul 2022	8
11. jenishpfly@gmail.com	Grand Canyon	16 Jul 2022	23 Jul 2022	8

Figure 87 Tour Bookings 1

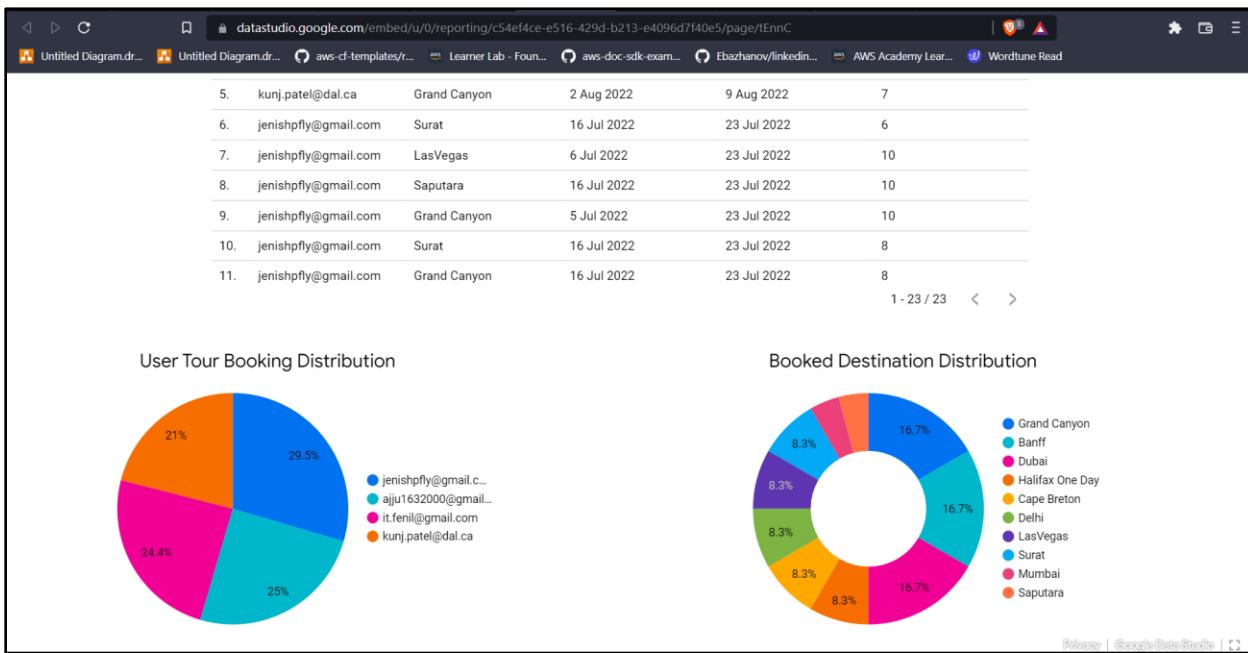


Figure 88 Tour Bookings 2

## 10. Limitations

Some of the limitations that we have in our application are as follows:

- Currently, we are not providing forgot password feature as it was not required by mentioned requirements documentation. However, if user still want to change their password, they cannot do from our application.
- In our application, manager or admin of the hotel must manually make the rooms available to book. That is quite stressful for admin to keep eye on the checkout data and that is also one of the limitations of our application.
- Even though we are booking rooms in accordance with user requests, we are not showing earlier reservations. The user would only see active bookings on the booking details page as a result.
- To manage sessions after a user registers or logs in, we use the local storage of the browser, which is prone to exploitation. JSON Web Tokens for data transmission across the network stream can be used to improve this.
- In the current implementation, once a user has requested to order food or book a room, we simply accept the request without integrating a payment method that can be used in real-time scenarios.
- At present, we don't currently have an interface for our application where we can show user profile information. As a result, the user lacks access to an interface for updating personal information like their address or phone number.
- For bot, it can perform limited functionalities and the connection of intents is yet to be implemented. Also the chatbot UI needs to be improved matching the website UI.
- Bot is not able to intelligently recognize the user input and trigger the respective intents.

## 11. References

- [1] A Iyengar, K Patel, J Patel,M Patel,R Tyagi,F Parmar “*Project Design Report*,” Dalhousie University, [Online document], 2022 [Accessed: Jul. 25, 2022].
- [2] "Flowchart Maker & Online Diagram Software", *App.diagrams.net*, 2022. [Online]. Available: <https://app.diagrams.net/>. [Accessed: 05- Jul- 2022].
- [3] "Google Cloud documentation | Documentation", *Google Cloud*, 2022. [Online]. Available: <https://cloud.google.com/docs>. [Accessed: 05-Jul-2022].
- [4] "AWS Documentation", *AWS Docs*, 2022. [Online]. Available: <https://docs.aws.amazon.com/>. [Accessed: 05-Jul-2022].
- [5] “Vertex AI,” *Google Cloud*. [Online]. Available: <https://cloud.google.com/vertex-ai>. [Accessed: 05-Jul-2022].
- [6] “AWS Cognito,” Amazon.com. [Online]. Available: <https://aws.amazon.com/cognito/>. [Accessed: 05-Jul-2022].
- [7] “AWS DynamoDB,” Amazon.com. [Online]. Available: <https://aws.amazon.com/dynamodb/>. [Accessed: 05-Jul-2022].
- [8] “AWS Lambda,” Amazon.com. [Online]. Available: <https://aws.amazon.com/lambda/>. [Accessed: 05-Jul-2022].
- [9] “What is Pub/Sub?,” *Google Cloud*. [Online]. Available: <https://cloud.google.com/pubsub/docs/overview>. [Accessed: 05-Jul-2022].
- [10] "What is Amazon Lex V2?", Amazon Lex, 2022. [Online]. Available: <https://docs.aws.amazon.com/lexv2/latest/dg/what-is.html>. [Accessed: 05 - Jul- 2022]
- [11] Amazon.com. [Online]. Available: <https://aws.amazon.com/quicksight/>. [Accessed: 05-Jul-2022].
- [12] Amazon.com. [Online]. Available: <https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-dynamodb.html>. [Accessed: 20-Jul-2022].
- [13] “RPubs - Hotel Demand Dataset,” *Rpubs.com*. [Online]. Available: <https://rpubs.com/saianne/832882>. [Accessed: 26-Jul-2022].

- [14] "Sentiment analysis tutorial," *Google Cloud*. [Online]. Available: <https://cloud.google.com/natural-language/docs/sentiment-tutorial>. [Accessed: 26-Jul-2022].
- [15] "Get online predictions from AutoML models," *Google Cloud*. [Online]. Available: <https://cloud.google.com/vertex-ai/docs/predictions/online-predictions-automl>. [Accessed: 26-Jul-2022].
- [16] "Deploy a Web UI for Your Chatbot | Amazon Web Services", Amazon Web Services, 2022. [Online]. Available: <https://aws.amazon.com/blogs/machine-learning/deploy-a-web-ui-for-your-chatbot/>. [Accessed: 22- Jul- 2022]