CSCI5410

Serverless Data Processing

Group Project – DALVacationHome Sprint 2

Submitted By:-
Divyank Shah – B00966377
Vivek Sonani – B00953064
Zeel Ravalani – B00917373
Yuci Wang – B00965074

Table of Contents

Research 3

  Service: Google Cloud Dailogflow 3

  Service: Amazon API Gateway 3

  User Registration 3

  User Confirmation 3

  Login – 1st Authentication 4

  Login – 2nd Authentication 4

  Login – 3rd Authentication 4

  User Registration Notification 4

Individual Contribution 5

System Architecture 5

Gitlab Link 5

Pseudocode 6

Test Cases 11

Evidence of Testing 12

References 17

Research

Service: Google Cloud Dailogflow

Dialogflow is a natural language understanding platform that makes it easy to design and integrate a conversational user interface into your mobile app, web application, device, bot, interactive voice response system, and so on. Using Dialogflow, you can provide new and engaging ways for users to interact with your product. [1]
Best Practices: To effectively use Google Dialogflow, design clear intents with diverse training phrases and manage conversations using contexts. Integrate webhooks for complex logic and use Dialogflow's versioning, testing, and analytics tools to optimize performance. Ensure data privacy and provide user guidance for a smooth experience. Regularly refine your agent based on user interactions to improve understanding and engagement. [2]
Use Case: Google Cloud Dialogflow will be used to create the Virtual Assistant in the DalVacationHome project.

Service: Amazon API Gateway

Amazon API Gateway allows you to create and deploy scalable REST and WebSocket APIs, accessing AWS services or other web services. It supports robust, secure API development for both internal use and third-party applications. [3]
Best Practices: When developing with API Gateway, follow these security best practices: implement least privilege access using IAM policies, utilize CloudWatch Logs or Amazon Data Firehose for logging API requests, set up CloudWatch alarms to monitor API metrics and trigger notifications, enable AWS CloudTrail to record detailed information on API Gateway actions, use AWS Config to track resource configurations and set compliance rules, and monitor security practices and compliance using AWS Security Hub's security controls and standards. [4].
Use Case: In DalVacationHome Amazon API Gateway will be used to create the API endpoints for the lambda function to call them from the frontend.

User Registration

This module will collect the information of the users and store it in DynamoDB. The information we collect includes user type, full name, email, encrypted password, username, phone number, gender, age, address, security question, answer, and key. The password is encrypted using AES-256 encryption [5]. We then register the user in AWS Cognito and store their details in DynamoDB. The environment used is NodeJS 20.x. We researched the AWS SDK for JavaScript v3 for interfaces for DynamoDB, Cognito Identity Provider, and encryption with the crypto library. We also referred to CloudFormation templates for the Lambda function definition. Detailed steps on encryption, user registration in Cognito, and DynamoDB operations were followed from AWS documentation.

User Confirmation

This module confirms user registration through AWS Cognito by using a confirmation code. Following user registration, Cognito sends a confirmation code via email verifies the user's email address and complete the registration process. The user enters this code in the application or confirmation page, which triggers the Lambda function. This function, utilizing the AWS SDK for JavaScript (v3) [6], interacts with the Cognito Identity provider [7] by constructing an *AdminConfirmSignUpCommand* with the user's email, and confirmation code. The Lambda function sends the confirmation command to Cognito, which validates the confirmation code. If valid, Cognito confirms the user's registration, and the Lambda function returns a success message. If invalid or an error occurs, the Lambda function returns an error message. Research references include AWS SDK for JavaScript v3 documentation, Cognito documentation, and Lambda function setup guides

Login – 1st Authentication

This module facilitates user authentication through AWS Cognito, initiating a first-factor login process. Upon the user's submission of their email and password, the Lambda function interacts with the AWS SDK for JavaScript (v3) [6], utilizing an *InitiateAuthCommand* object. This object specifies USER_PASSWORD_AUTH as the authentication flow and AuthParameters (Email and Password). The function sends this command to Cognito, which responds with tokens (such as access tokens) used as the session for subsequent authorized API calls, thereby authenticating logged-in users. Upon successful authentication, the Lambda function retrieves user security question data from DynamoDB using the email address as the key for 2nd factor authentication. Response handling includes returning the Cognito authentication response and security question data with appropriate status codes: 200 for success. Research references include AWS Cognito User Pools authentication flow documentation and AWS SDK for JavaScript (v3) Cognito Identity Provider Client documentation.

Login – 2nd Authentication

This module will get answers from the user for user's security questions. And it will compare the user submitted answer with the stored answer in the DynamoDB. One user will have only one security question chosen from provided list. And only one answer will be stored in DynamoDB.

Login – 3rd Authentication

This module will use Caeser Cipher [8] to validate user's answer with random word we generated. User will choose the key for the Caeser Cipher between 1 and 10. User needs to remember the key and use it to solve the problem.

User Registration Notification

In addition to the research on user registration and authentication, development is underway for a notification system to send a welcome message upon successful user registration confirmation. This approach utilizes a combination of AWS services: SQS (Simple Queue Service)[9] and SNS (Simple Notification Service)[10] along with authentication module Lambda functions.

Here's the breakdown of the approach:

- Upon Confirming User Registration in Cognito: The user is subscribed to the SNS topic with a filter policy including their email address. This ensures messages are only sent to confirmed users. This Lambda function instead of directly publishing to SNS, it sends a message to an SQS queue containing user information.

- Queue Consumer Lambda: Another Lambda function listens to the SQS queue. Upon receiving a message, it verifies the user's subscription confirmation status with SNS to ensure they are subscribed to the topic. If subscribed, the Lambda function publishes a welcome message to the SNS topic, triggering the welcome email containing the user-chosen Caesar Cipher authentication key (set during registration).

Note: This notification development is still in progress.

Individual Contribution

1. **Divyank Shah**
   - Developed the frontend for the Authentication module.
   - Worked on Amazon Lex and Lambda functions.
   - Researched the working of the Google Cloud Dialogflow.
   - Integrated frontend and backend for the Authentication module.
   - Worked on API Gateway.
   - Developed landing page (Home page) for all type of users.
2. **Vivek Sonani**
   - Worked on the frontend and backend of the Room booking system.
   - Worked on Amazon Lex and Lambda functions.
   - Worked on the overall application architecture design.
3. **Zeel Ravalani**
   - Developed the lambda functions for the signup flow and login 1st factor.
   - Developed the API Gateway for lambda functions.
   - Developed CloudFormation.
   - Worked on the SNS service for notifications.
4. **Yuci Wang**
   - Developed the lambda functions for the 2nd and 3rd factor for login.
   - Developed the API Gateway for lambda functions.
   - Developed CloudFormation.

System Architecture

Figure 1 displays the overall architecture of the application. The frontend will first interact with the API gateway and then lambda functions to use different services. The frontend will directly access the virtual assistant Amazon Lex which will trigger the lambda function to perform specific tasks.

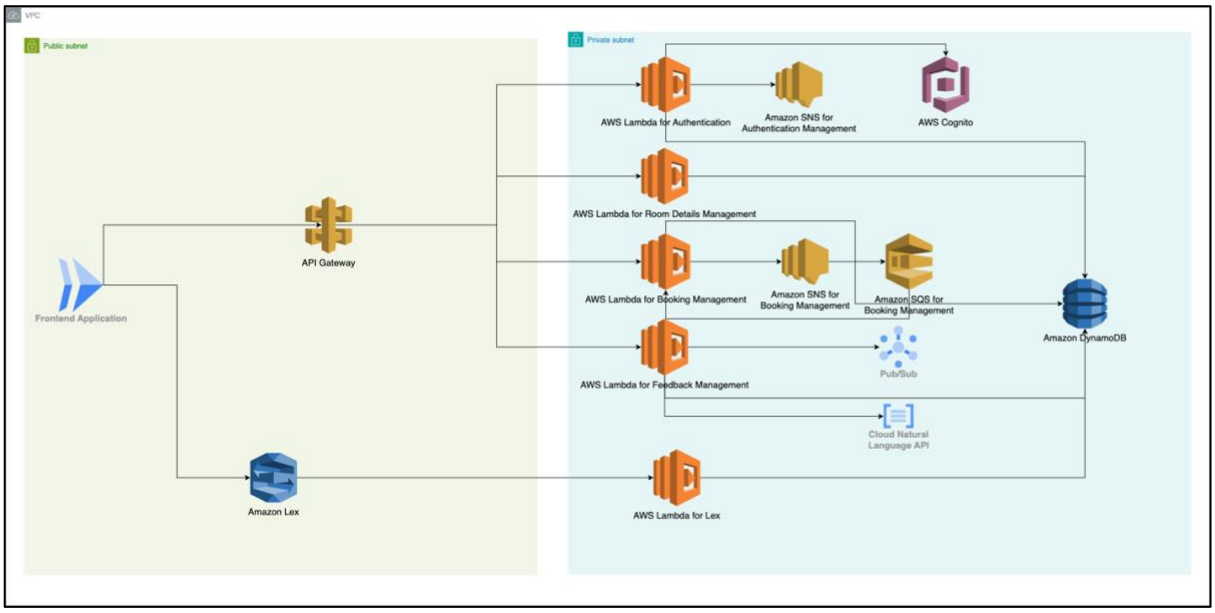

VPC
Public subnet
Private subnet
AWS Lambda for Authentication
Amazon SNS for Authentication Management
AWS Cognito
AWS Lambda for Room Details Management
API Gateway
Frontend Application
AWS Lambda for Booking Management
Amazon SNS for Booking Management
Amazon SQS for Booking Management
Amazon DynamoDB
Pub/Sub
AWS Lambda for Feedback Management
Cloud Natural Language API
Amazon Lex
AWS Lambda for Lex

Figure 1. Overall System Architecture [11].

Gitlab Link

Link: https://git.cs.dal.ca/yuci/CSCI5410-S24-SDP-7

Pseudocode

Signup – User Registration

```
Algorithm: user registration process
Input: User signup details
Output: HTTP response containing the status of registration
    1. Initialize DynamoDB client
    2. Initialize Cognito Identity Provider client
    3. Initialize variables for response body and status code
    4. Fetch necessary environment variables (e.g., DynamoDB table name, Cognito client ID)
    5. Destructure event data for user signup details (user_type, fullname, email, password, userName, phone_no, gender, age, address, question, answer, key)
    6. Encrypt the password using AES256 algorithm and a predefined cipher key
    7. Define SignUpParams for Cognito registration with provided user attributes
    8. Send SignUpCommand to CognitoIdentityProviderClient to register user in Cognito
    9. Check if user already exists in DynamoDB using email as unique identifier
    10. If user not found in DynamoDB:
            • Prepare userItem object with signup details including encrypted password
            • Create PutCommand to store userItem in DynamoDB
            • Send PutCommand to DynamoDB to store user details
            • Set response body to JSON.stringify(signUpResponse) and status code to 200 (Success)
    11. Else:
            • Set response body to "User already exists" and status code to 403 (Forbidden)
    12. Catch any errors during signup or database operations:
            • Set response body to error message and status code to 500 (Internal Server Error)
    13. Prepare HTTP response object with statusCode, headers, and responseBody
    14. Return the HTTP response object
```

Signup – User Confirmation

```
Algorithm: confirm user registration
Input: User email and confirmation code
Output: HTTP response containing the status of confirmation
    1. Initialize Cognito Identity Provider client
    2. Fetch necessary environment variables (e.g., Cognito user pool ID, Cognito client ID)
    3. Initialize variables for response body and status code
    4. Destructure event data for user email and confirmation code (email, confirmationCode)
    5. Define confirmParams with UserPoolId, Username, ConfirmationCode, and ClientId
    6. Create AdminConfirmSignUpCommand using confirmParams
    7. Send AdminConfirmSignUpCommand to CognitoIdentityProviderClient to confirm user registration
    8. Set response body to "User confirmed successfully" and status code to 200 (Success) upon successful confirmation
    9. Catch any errors during confirmation process:
            • Set response body to error message and status code to 500 (Internal Server Error)
    10. Prepare HTTP response object with statusCode, headers, and responseBody
    11. Return the HTTP response object
```

Login – 1st Factor

Algorithm: Login 1st Factor Authentication
Input: User email and password
Output: Http response containing status of authentication (from Cognito) and security question (for 2nd Factor authentication)

```
1. Initialize DynamoDB client
2. Initialize Cognito Identity Provider client
3. Initialize variables for response body and status code
4. Fetch necessary environment variables (e.g., DynamoDB table name, Cognito client
   ID)
5. Destructure event data for login details (email, password)
6. Check if user exists in DynamoDB using email as unique identifier
7. If user exists in DynamoDB:
     • Define authParams for Cognito authentication with provided email and
       password
     • Send InitiateAuthCommand to CognitoIdentityProviderClient to authenticate
       user in Cognito
     • Set userQAQuestion to the security question retrieved from DynamoDB
8. Else:
     • Set response body to "User not found" and status code to 404 (Not Found)
9. Set response body to JSON object containing authResponse and userQAQuestion
10. Set status code to 200 (Success)
11. Catch any errors during authentication or database operations:
     • Set response body to error message and status code to 500 (Internal Server
       Error)
12. Prepare HTTP response object with statusCode, headers, and responseBody
13. Log the response object for debugging
14. Return the HTTP response object
```

Login – 2nd Factor

Algorithm : Question and answer authentication
Input: User email and answer to the security question
Output: Http response containing status of authentication

```
1. Email, answer ← event passed from frontend
2. User ← DynamoDB. Get(email)
3. If user not exists
4. Return 'User not found'
5. End if
6. If user.item.answer == answer
7. Return 'Success'
8. Else
9. Return 'Wrong answer'
10. End if
```

Login – 3rd Factor

Algorithm: Caesar Cipher
Input: Word, Key
Output: A decrypted word with number of key movements

```
1. Email, answer ← event passed from frontend
2. I ← 0
```

```
3. While (i<length of the word)
      a. Word[i]←word[i]-key
4. End while
5. Return word
```

Room Booking System – Get all room details

1. **Initialize DynamoDB Client**:
   - Create a DynamoDB client instance using the DynamoDBClient class.
   - Create a DynamoDB Document client from the DynamoDB client instance.
2. **Define the Table Name**:
   - Set the table name to Rooms.
3. **Lambda Handler Function**:
   - Define an asynchronous handler function to handle the Lambda event.
   - Inside the function, perform the following steps:
     1. **Setup Scan Parameters**:
        - Define scan parameters with the table name Rooms.
     2. **Scan DynamoDB Table**:
        - Use the DynamoDB Document client to send a ScanCommand with the defined scan parameters.
        - Store the result of the scan.
     3. **Return Success Response**:
        - If the scan is successful, return a response with status code 200 and the scanned items as the body.
     4. **Error Handling**:
        - If an error occurs during the scan, catch the error and return a response with status code 500 and an error message.

Room Booking System – Get room details by room ID

1. **Initialize DynamoDB Client**:
   - Create a DynamoDB client instance using the DynamoDBClient class.
   - Create a DynamoDB Document client from the DynamoDB client instance.
2. **Define the Table Name**:
   - Set the table name to Rooms.
3. **Lambda Handler Function**:
   - Define an asynchronous handler function to handle the Lambda event.
   - Inside the function, perform the following steps:
     1. **Extract Room ID**:
        - Retrieve room_id from the query string parameters of the event.
     2. **Setup Get Parameters**:
        - Define parameters for the GetCommand with the table name Rooms and the room ID as the key.
     3. **Get Room Details**:
        - Use the DynamoDB Document client to send a GetCommand with the defined parameters.
        - Store the result of the get operation.
     4. **Check if Room Exists**:
        - If no item is found, return a response with status code 404 and a message indicating the room was not found.
     5. **Return Success Response**:

    - If the get operation is successful, return a response with status code 200 and the room details as the body.
6. **Error Handling**:
    - If an error occurs during the get operation, catch the error and return a response with status code 500 and an error message.

Room Booking System – Add room details

1. **Initialize DynamoDB Client**:
    - Create a DynamoDB client instance using the DynamoDBClient class.
    - Create a DynamoDB Document client from the DynamoDB client instance.
2. **Define the Table Name**:
    - Set the table name to Rooms.
3. **Lambda Handler Function**:
    - Define an asynchronous handler function to handle the Lambda event.
    - Inside the function, perform the following steps:
        1. **Extract Room Details**:
            - Parse the event body to retrieve room_number, room_type, price, and features.
        2. **Check for Existing Room**:
            - Define scan parameters to check if a room with the given room_number already exists.
            - Use the DynamoDB Document client to send a ScanCommand with the defined parameters.
            - If an existing room is found, return a response with status code 400 and a message indicating the room number already exists.
        3. **Generate New Room ID**:
            - Define scan parameters to retrieve all room IDs.
            - Use the DynamoDB Document client to send a ScanCommand with the defined parameters.
            - Calculate the maximum room ID and generate a new room ID by incrementing the maximum ID by 1.
        4. **Setup Put Parameters**:
            - Define parameters for the PutCommand with the table name Rooms and the new room details, including the new room ID.
        5. **Add New Room**:
            - Use the DynamoDB Document client to send a PutCommand with the defined parameters to add the new room.
        6. **Return Success Response**:
            - If the room is added successfully, return a response with status code 200 and a message indicating the room was added successfully, along with the new room ID.
        7. **Error Handling**:
            - If an error occurs during the process, catch the error and return a response with status code 500 and an error message.

Room Booking System – Update room details

1. **Initialize DynamoDB Client**:
    - Create a DynamoDB client instance using the DynamoDBClient class.
    - Create a DynamoDB Document client from the DynamoDB client instance.
2. **Define the Table Name**:

    - Set the table name to Rooms.
3. **Lambda Handler Function**:
    - Define an asynchronous handler function to handle the Lambda event.
    - Inside the function, perform the following steps:
        1. **Extract Room Details**:
            - Parse the event body to retrieve room_id, room_number, room_type, price, and features.
        2. **Check if Room Exists**:
            - Define get parameters to check if a room with the given room_id exists.
            - Use the DynamoDB Document client to send a GetCommand with the defined parameters.
            - If the room does not exist, return a response with status code 404 and a message indicating the room was not found.
        3. **Setup Update Parameters**:
            - Define parameters for the UpdateCommand with the table name Rooms, the room ID as the key, and the new room details in the UpdateExpression.
            - Include the new room details as expression attribute values.
        4. **Update Room Details**:
            - Use the DynamoDB Document client to send an UpdateCommand with the defined parameters to update the room details.
            - Store the result of the update operation.
        5. **Return Success Response**:
            - If the room is updated successfully, return a response with status code 200 and a message indicating the room was updated successfully, along with the updated room details.
        6. **Error Handling**:
            - If an error occurs during the process, catch the error and return a response with status code 500 and an error message.

Room Booking System – Delete room details

1. **Initialize DynamoDB Client**:
    - Create a DynamoDB client instance using the DynamoDBClient class.
    - Create a DynamoDB Document client from the DynamoDB client instance.
2. **Define the Table Name**:
    - Set the table name to Rooms.
3. **Lambda Handler Function**:
    - Define an asynchronous handler function to handle the Lambda event.
    - Inside the function, perform the following steps:
        1. **Extract Room ID**:
            - Retrieve room_id from the query string parameters of the event.
        2. **Setup Delete Parameters**:
            - Define parameters for the DeleteCommand with the table name Rooms and the room ID as the key.
        3. **Delete Room**:
            - Use the DynamoDB Document client to send a DeleteCommand with the defined parameters to delete the room.

4. **Return Success Response**:
    - If the room is deleted successfully, return a response with status code 200 and a message indicating the room was deleted successfully.
5. **Error Handling**:
    - If an error occurs during the process, catch the error and return a response with status code 500 and an error message.

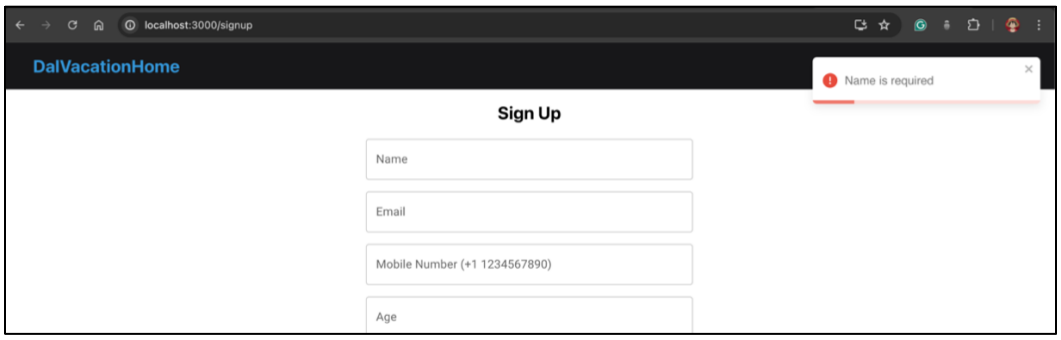
Test Cases

Sr no.	Test case Description	Expected Output	Actual Output
1.	If the user leaves any field empty, then user should not be able to register.	User should receive an error message in toast container.	User receives an error message in toast container.
2.	If the user enters wrong confirmation code then his status should remain unconfirmed in cognito.	User should receive an error message in toast container that invalid confirmation code.	User receives an error message in toast container that invalid confirmation code.
3.	If the user enters wrong credentials then they should not be able to proceed to next stage.	User should receive an error message invalid credentials in toast container.	User receives an error message invalid credentials in toast container.
4.	If user enters wrong answer for the MFA question then they should not be able to proceed to next stage.	User should receive an error message stating wrong answer.	User receives an error message stating wrong answer.
5.	If user enters wrong decrypted word for the given random word then they should not be able to login.	User should receive an error message decryption failed in toast container.	User receives an error message decryption failed in toast container.
6.	Property agent should be able to see all the room details.	All room details	All room details
7.	Property agent should be able to add room details including room number, room type, price, and features.	Added room details	Added room details
8.	Property agent should be able to update room details including room number, room type, price, and features.	Updated room details	Updated room details
9.	Property agent should be able to delete selected room details.	Deleted room details	Deleted room details

Evidence of Testing

Authentication Module

Signup page – Frontend


localhost:3000/signup

DalVacationHome

Name is required

Sign Up

Name

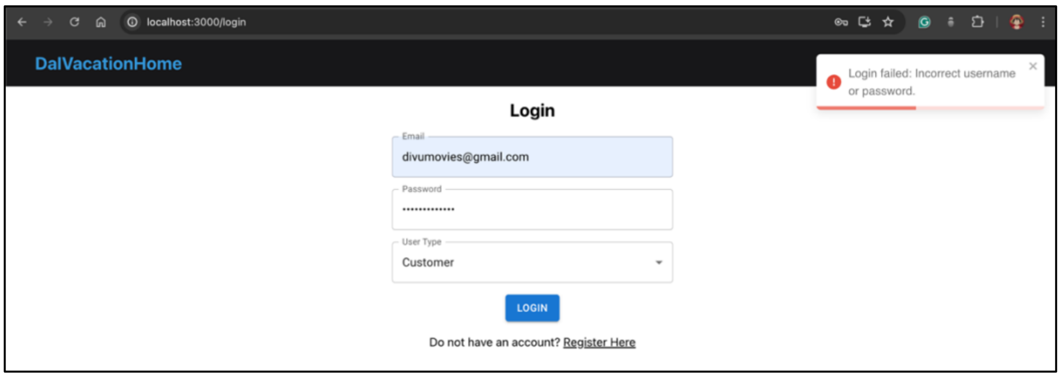
Email

Mobile Number (+1 1234567890)

Age

Figure 2. Test Case showing signup page with form validation

Login Page – Frontend


localhost:3000/login

DalVacationHome

Login failed: Incorrect username or password.

Login

Email: divumovies@gmail.com

Password: •••••••••••••

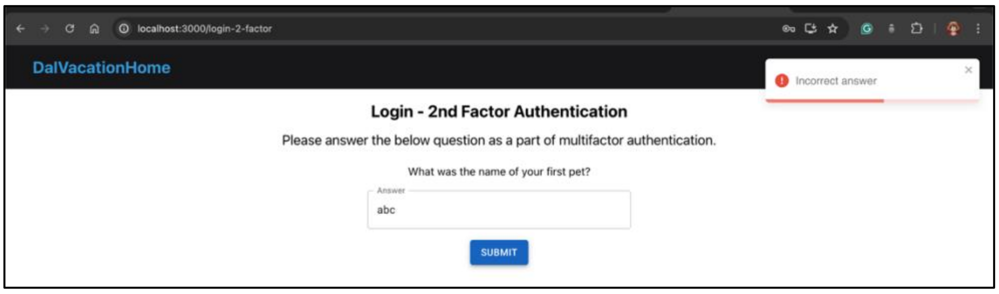
User Type: Customer

LOGIN

Do not have an account? Register Here

Figure 3. Test case for login page showing error for incorrect login credentials

Login – 2nd Factor Authentication – Frontend


localhost:3000/login-2-factor

DalVacationHome

Incorrect answer

Login - 2nd Factor Authentication

Please answer the below question as a part of multifactor authentication.

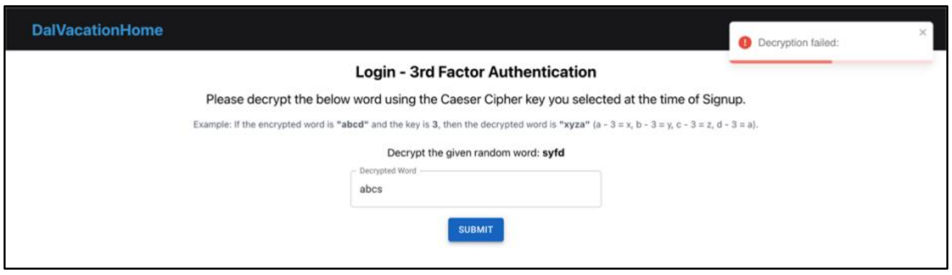
What was the name of your first pet?

Answer: abc

SUBMIT

Figure 4. Test case for 2nf factor authentication in login.

Login – 3rd Factor Authentication – Frontend


DalVacationHome

Decryption failed:

Login - 3rd Factor Authentication

Please decrypt the below word using the Caeser Cipher key you selected at the time of Signup.

Example: If the encrypted word is **"abcd"** and the key is **3**, then the decrypted word is **"xyza"** (a - 3 = x, b - 3 = y, c - 3 = z, d - 3 = a).

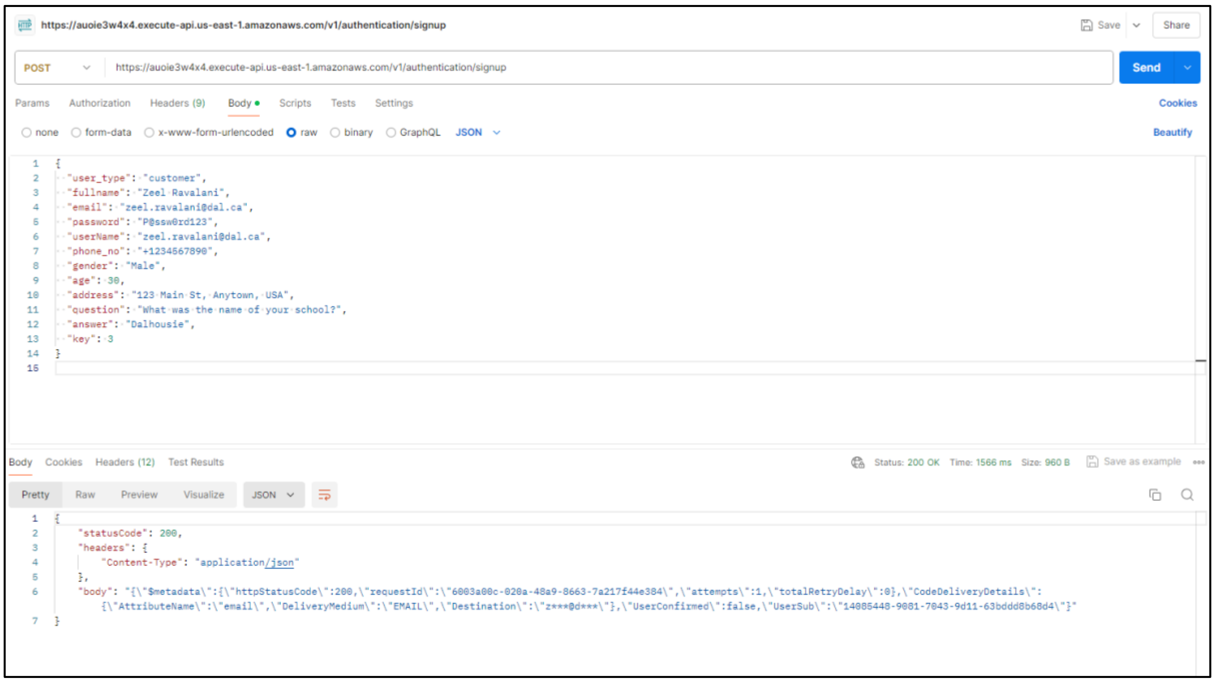
Decrypt the given random word: **syfd**

Decrypted Word: abcs

SUBMIT

Figure 5. Test case for the login 3rd-factor authentication.

Signup - Backend


https://auoie3w4x4.execute-api.us-east-1.amazonaws.com/v1/authentication/signup

POST https://auoie3w4x4.execute-api.us-east-1.amazonaws.com/v1/authentication/signup

Body: raw JSON

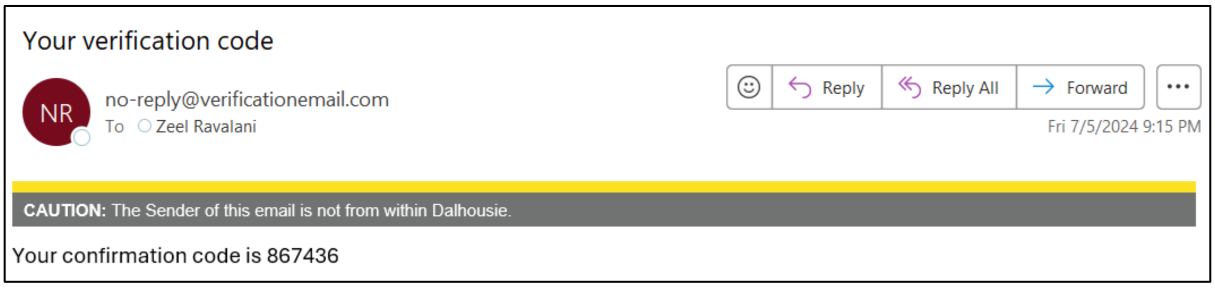
```
{
  "user_type": "customer",
  "fullname": "Zeel Ravalani",
  "email": "zeel.ravalani@dal.ca",
  "password": "P@ssw0rd123",
  "userName": "zeel.ravalani@dal.ca",
  "phone_no": "+1234567890",
  "gender": "Male",
  "age": 30,
  "address": "123 Main St, Anytown, USA",
  "question": "What was the name of your school?",
  "answer": "Dalhousie",
  "key": 3
}
```

Status: 200 OK Time: 1566 ms Size: 960 B

```
{
    "statusCode": 200,
    "headers": {
        "Content-Type": "application/json"
    },
    "body": "{\"$metadata\":{\"httpStatusCode\":200,\"requestId\":\"6003a00c-020a-48a9-8663-7a217f44e384\",\"attempts\":1,\"totalRetryDelay\":0},\"CodeDeliveryDetails\":
        {\"AttributeName\":\"email\",\"DeliveryMedium\":\"EMAIL\",\"Destination\":\"z***@d***\"},\"UserConfirmed\":false,\"UserSub\":\"14085448-9081-7043-9d11-63bddd8b68d4\"}"
}
```

Figure 6. Test case and result for user registration.

Cognito – Confirmation Code Email


Your verification code

no-reply@verificationemail.com

To Zeel Ravalani

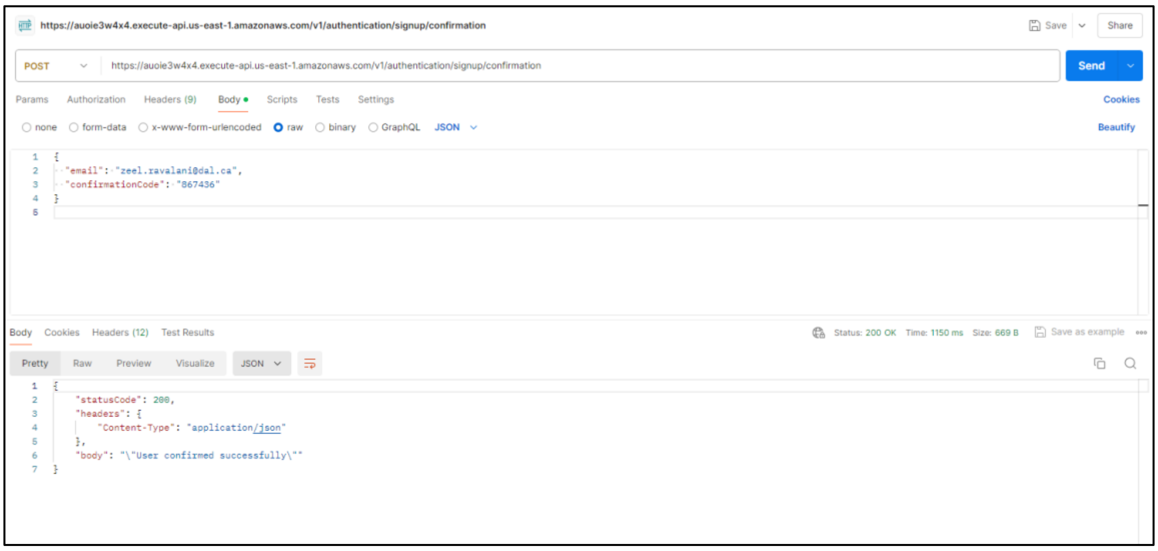
Fri 7/5/2024 9:15 PM

CAUTION: The Sender of this email is not from within Dalhousie.

Your confirmation code is 867436

Figure 7. Confirmation code sent by Cognito.

Signup - User confirmation


https://auoie3w4x4.execute-api.us-east-1.amazonaws.com/v1/authentication/signup/confirmation

POST https://auoie3w4x4.execute-api.us-east-1.amazonaws.com/v1/authentication/signup/confirmation

Body: raw JSON

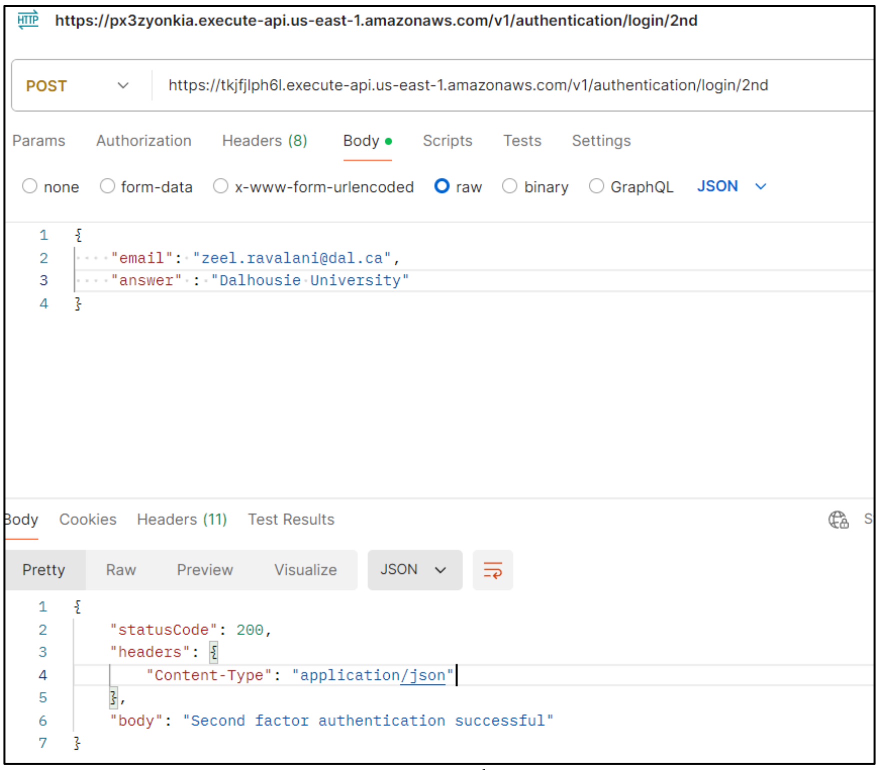
```
{
  "email": "zeel.ravalani@dal.ca",
  "confirmationCode": "867436"
}
```

Status: 200 OK Time: 1150 ms Size: 669 B

```
{
    "statusCode": 200,
    "headers": {
        "Content-Type": "application/json"
    },
    "body": "\"User confirmed successfully\""
}
```

Figure 8. Test case and result for confirm user registration

Login Backend – 2nd Factor Authentication – Backend


https://px3zyonkia.execute-api.us-east-1.amazonaws.com/v1/authentication/login/2nd

POST https://tkjfjlph6l.execute-api.us-east-1.amazonaws.com/v1/authentication/login/2nd

Body: raw JSON

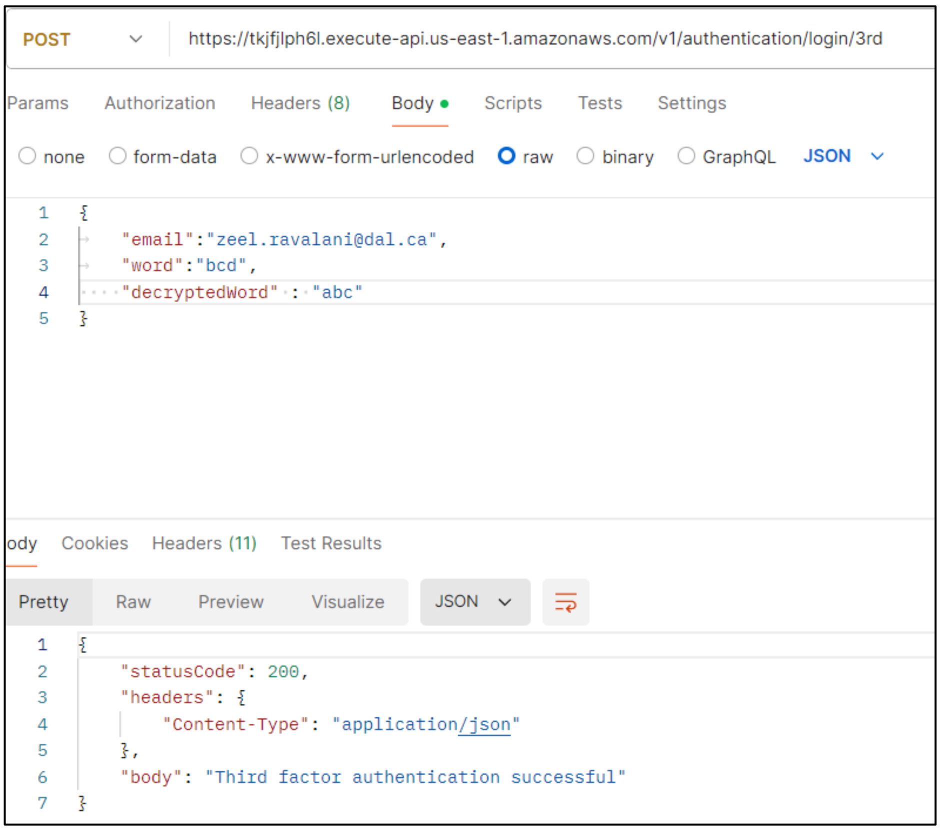
```
{
    "email": "zeel.ravalani@dal.ca",
    "answer" : "Dalhousie University"
}
```

Response:

```
{
    "statusCode": 200,
    "headers": {
        "Content-Type": "application/json"
    },
    "body": "Second factor authentication successful"
}
```

Figure 9. Test case for the backend of 2nd-factor authentication of the login

Login – 3rd Factor Authentication – Backend


POST https://tkjfjlph6l.execute-api.us-east-1.amazonaws.com/v1/authentication/login/3rd

Body: raw JSON

```
{
    "email":"zeel.ravalani@dal.ca",
    "word":"bcd",
    "decryptedWord" : "abc"
}
```

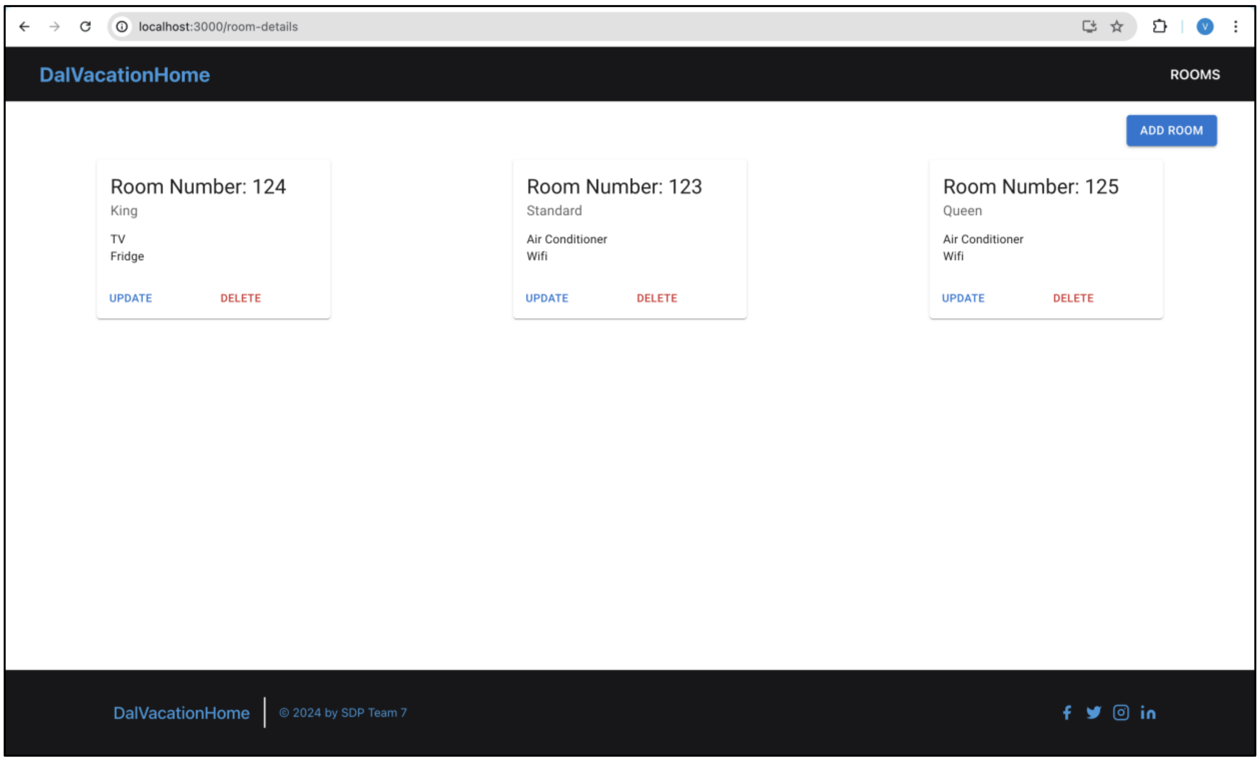
Response:

```
{
    "statusCode": 200,
    "headers": {
        "Content-Type": "application/json"
    },
    "body": "Third factor authentication successful"
}
```

Figure 10. Test case for the backend of 3rd-factor authentication of the login.

Room booking system

View All Rooms


localhost:3000/room-details

DalVacationHome

ROOMS

ADD ROOM

Room Number: 124
King
TV
Fridge
UPDATE DELETE

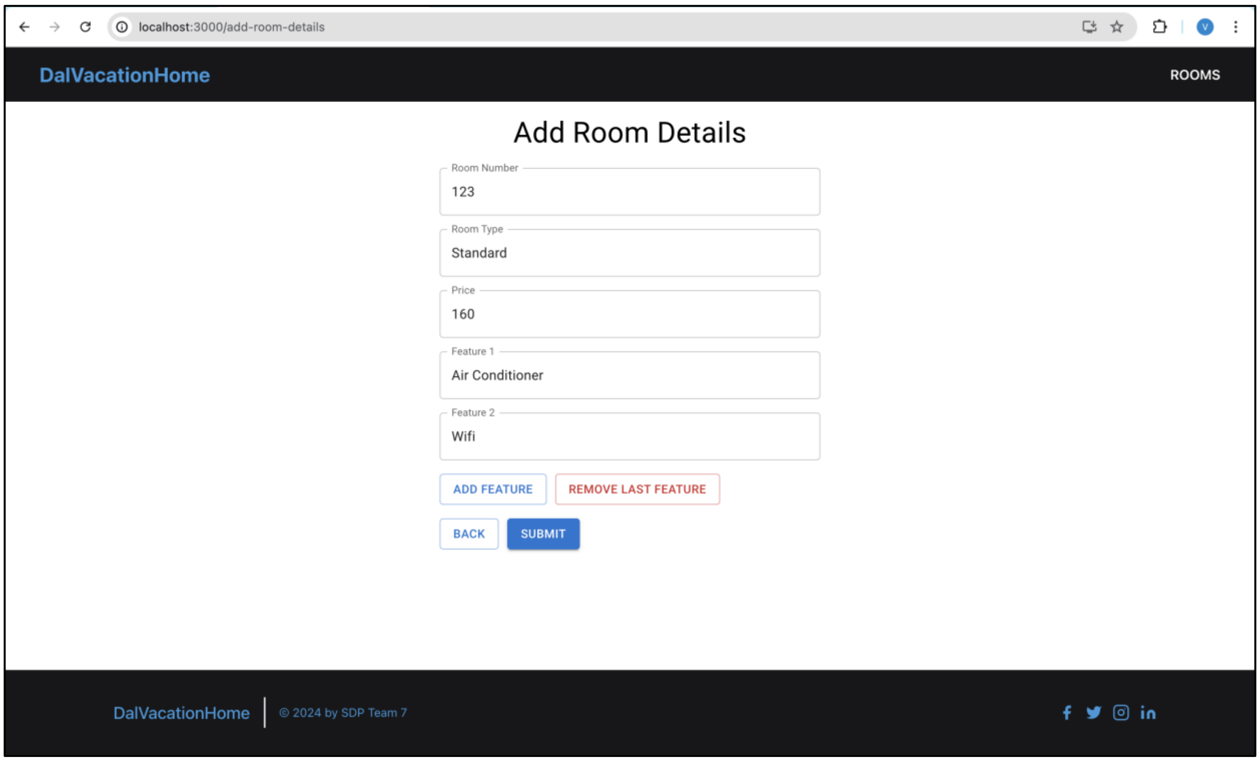
Room Number: 123
Standard
Air Conditioner
Wifi
UPDATE DELETE

Room Number: 125
Queen
Air Conditioner
Wifi
UPDATE DELETE

DalVacationHome | © 2024 by SDP Team 7

Figure 11. Test case view all rooms.

Add Room Details


localhost:3000/add-room-details

DalVacationHome

ROOMS

Add Room Details

Room Number: 123
Room Type: Standard
Price: 160
Feature 1: Air Conditioner
Feature 2: Wifi

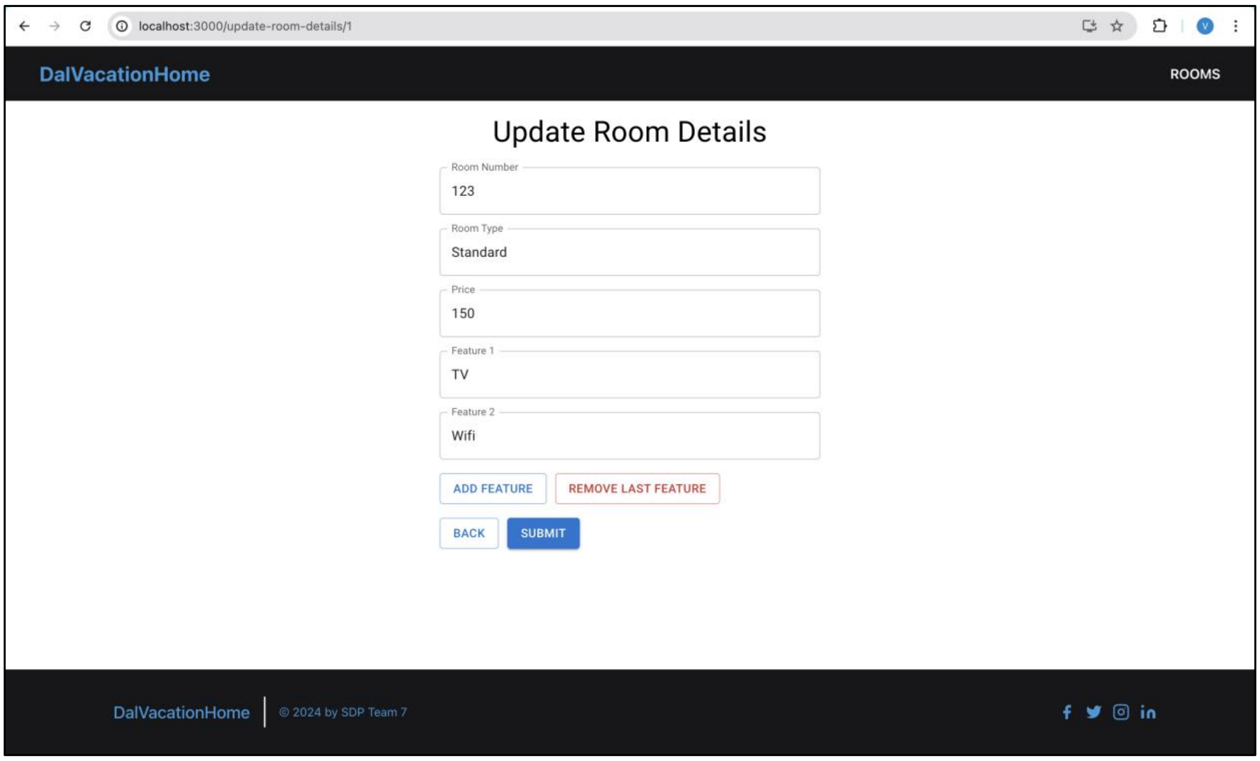
ADD FEATURE REMOVE LAST FEATURE

BACK SUBMIT

DalVacationHome | © 2024 by SDP Team 7

Figure 12. Testcase add room details

Update Room Details


localhost:3000/update-room-details/1

Update Room Details

Room Number: 123

Room Type: Standard

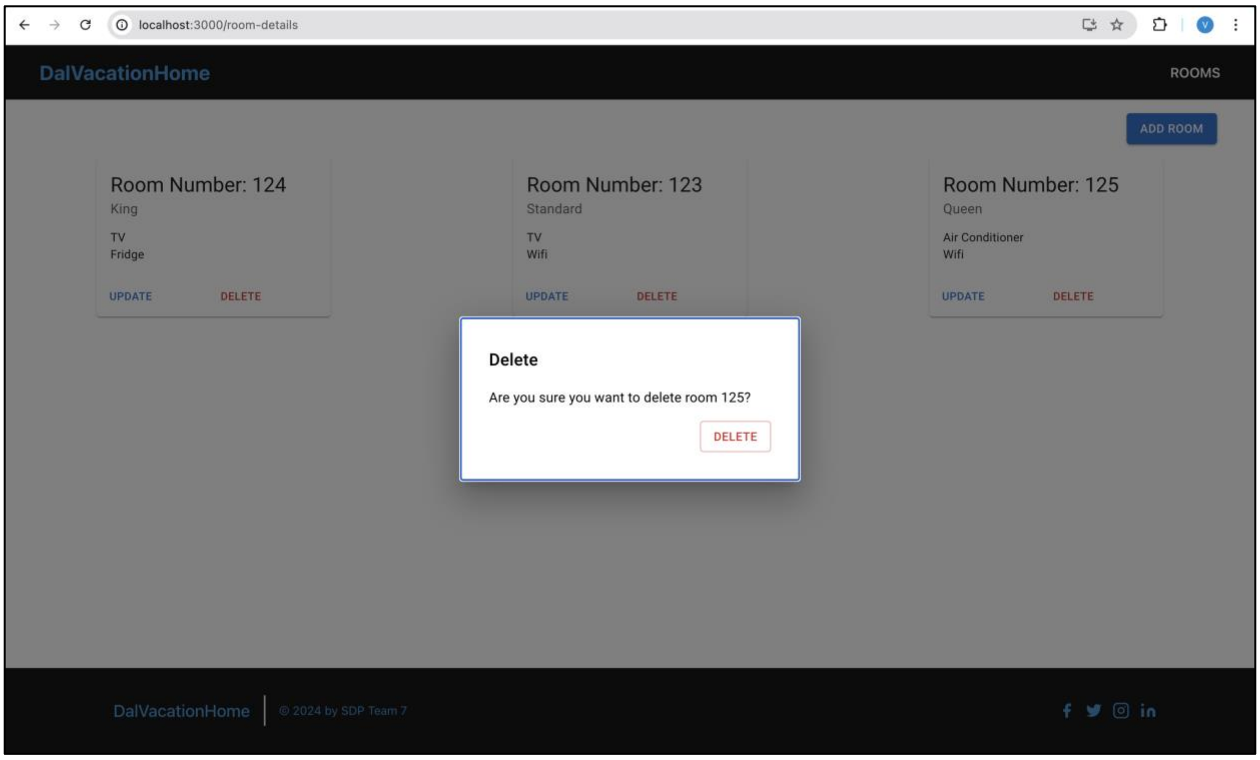
Price: 150

Feature 1: TV

Feature 2: Wifi

Figure 13. Test case update room details.

Delete Room Details


localhost:3000/room-details

Room Number: 124
King
TV
Fridge

Room Number: 123
Standard
TV
Wifi

Room Number: 125
Queen
Air Conditioner
Wifi

Delete

Are you sure you want to delete room 125?

Figure 14. Testcase delete rooms.

References

[1] “Dialogflow documentation | google cloud,” *Google* [Online]. Available: https://cloud.google.com/dialogflow/docs. [Accessed: Jul. 5, 2024].

[2] “Service use best practices | Dialogflow CX | google cloud,” *Google* [Online]. Available: https://cloud.google.com/dialogflow/cx/docs/concept/best-practices. [Accessed: Jul. 5, 2024].

[3] “Amazon API Gateway Documentation,” *AWS Documentation* [Online]. Available: https://docs.aws.amazon.com/apigateway/. [Accessed: Jul. 5, 2024].

[4] “Security best practices in amazon API gateway - amazon API gateway,” *AWS Documentation* [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/security-best-practices.html. [Accessed: Jul. 5, 2024].

[5] “What is Node crypto.createCipher(algorithm, password[, options])?,” *Educative* [Online]. Available: https://www.educative.io/answers/what-is-node-cryptocreatecipheralgorithm-password-options. [Accessed: Jul. 05, 2024]

[6] “AWS SDK for JavaScript v3,” *docs.aws.amazon.com* [Online]. Available: https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/. [Accessed: Jul. 05, 2024]

[7] “Using Amazon Cognito Identity to Authenticate Users - AWS SDK for JavaScript,” *docs.aws.amazon.com* [Online]. Available: https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/loading-browser-credentials-cognito.html. [Accessed: Jul. 05, 2024].

[8] "Tutorial: Create a CRUD HTTP API with Lambda and DynamoDB," AWS, 2024. [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-dynamo-db.html. [Accessed: Jun. 29, 2024].

[9] “What is Amazon Simple Queue Service? - Amazon Simple Queue Service,” *docs.aws.amazon.com* [Online]. Available: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html. [Accessed: Jul. 5, 2024]

[10] AWS, “What is Amazon SNS? - Amazon Simple Notification Service,” *docs.aws.amazon.com* [Online]. Available: https://docs.aws.amazon.com/sns/latest/dg/welcome.html. [Accessed: Jul. 5, 2024]

[11] “draw.io - documentation,” *Draw.io documentation* [Online]. Available: https://www.drawio.com/doc/. [Accessed: Jul. 5, 2024].

[12] Mak, "How to encrypt data that needs to be decrypted in node.js?," Stack Overflow, 3 Jun 2020. [Online]. Available: https://stackoverflow.com/questions/6953286/how-to-encrypt-data-that-needs-to-be-decrypted-in-node-js. [Accessed: Jun. 29, 2024].

[13] T. John.S, "Implementing Caesar Cipher in JavaScript for Secure Communication," Medium, 21 Jul 2023. [Online]. Available: https://medium.com/@stheodorejohn/implementing-caesar-cipher-in-javascript-for-secure-communication-6f82bbe914c2. [Accessed: Jul. 2, 2024].

[14] “What is AWS CloudFormation? - AWS CloudFormation,” *docs.aws.amazon.com* [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html. [Accessed: Jul. 5, 2024]

Serverless Data Processing Team - 7 Meeting Logs

Meeting Date	Meeting Agenda	Minutes of Meeting	Attendance	Meeting Recordings
4 June, 2024	Gitlab Board Issues creation	1. Created the GitLab board issues for the Authentication module and Virtual Assitant Module. 2. Divided the development task among the team members. 3. Finalized the backend development Tech stack.	All Members Present	Call with SDP_Team- 7-20240604_193320-Meeting Recording.mp4
26 June, 2024	Discussion about the progress in the project.	1. Discussed every team member's progress. 2. Discussed what other task needs to be completed. 3. Discussed the topics that are needed to be added to the report. 4. Discussed about any blockers for any team member. 5. Solved team members queries regarding the use of services and forntend backend integration.	All Members Present	Call with SDP_Team- 7-20240626_190641-Meeting Recording.mp4
1 July, 2024	Team members progress discussion	1. Discussed the workflow of Caecer Cipher in the authentication module. 2. Discussed about the blockers related to SNS. 3. Understanding the cloud formation and how to use it. 4. Properly understood the working of Caeser Cipher and how to use it in the authentication module.	All Members Present	SDP7 TeamMeetUp-20240701_140215-Meeting Recording.mp4
3 July, 2024	Project progress discussion	1. Discussed the report for the sprint 2 submission. 2. Discussed the project progress and solved team member's query.	All Members Present	SDP7 Meetup-20240703_193202-Meeting Recording.mp4