

## TravelGo: A Cloud-Powered Real-Time Travel Booking Platform Using AWS

### Project Description:

**TravelGo** is a full-stack, cloud-based travel booking platform designed to simplify the process of reserving buses, trains, flights, and hotels through a unified interface. Built using Flask as the backend framework, the application is deployed on Amazon EC2 and leverages DynamoDB for efficient storage of user data and bookings. TravelGo allows users to register, log in, search for transportation and accommodation options, and book their travel with ease. Once a booking is confirmed or cancelled, users receive real-time email notifications powered by AWS Simple Notification Service (SNS), keeping them informed throughout their journey.

The platform's user-friendly interface supports dynamic seat selection for buses, hotel filtering based on preferences such as luxury or budget, and provides booking summaries along with centralized cancellation management. By combining cloud scalability, responsive design, and secure session handling, TravelGo delivers a seamless and real-time travel planning experience for users.

### Scenario 1: Hassle-Free Multi-Mode Travel Booking Experience

**TravelGo** offers users a unified platform to search and book buses, trains, flights, and hotels all in one place. For instance, a user planning a trip from Hyderabad to Bangalore can log in, select their preferred mode of transport, choose from available options, and proceed to booking. Flask manages the backend operations such as retrieving travel listings and processing user input in real-time. Hosted on AWS EC2, the platform remains responsive even during high-traffic hours like weekends or holiday seasons, allowing multiple users to browse and book without delay.

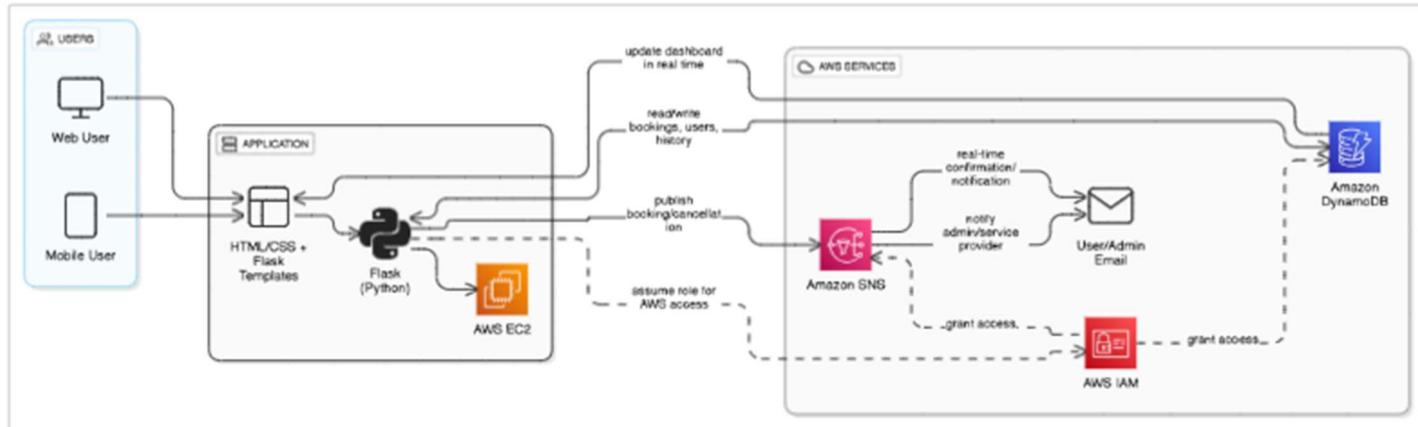
### Scenario 2: Real-Time Booking Confirmation with AWS SNS

Once a booking is made—whether it's a train ticket or a hotel stay—TravelGo uses AWS SNS to instantly notify the user. For example, after a student books a hotel in Chennai, SNS sends a real-time email notification confirming the booking with all the relevant details. This notification is triggered from the Flask backend after the booking is successfully recorded in DynamoDB. Additionally, SNS can alert admin or service providers, ensuring transparency and real-time updates on every transaction.

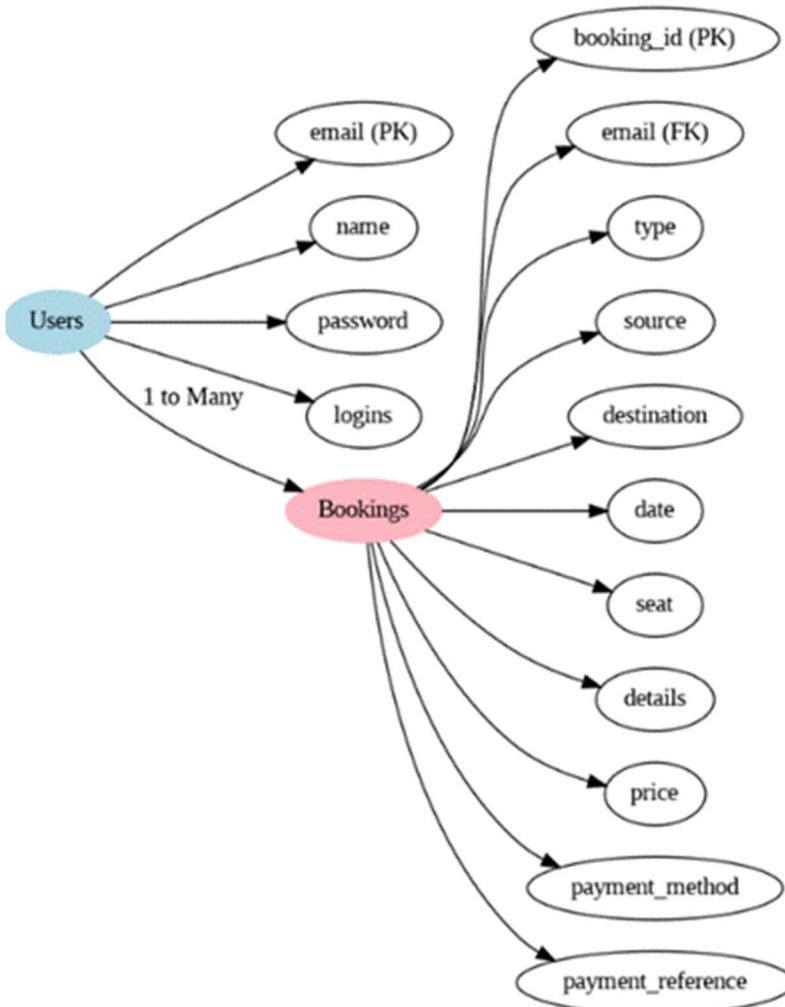
### Scenario 3: Dynamic Dashboard with Personal Travel History

TravelGo features a dynamic user dashboard that displays all past and upcoming bookings for the logged-in user. For example, a user who has booked a flight and a hotel can view these bookings categorized by type, along with dates, price, and cancellation options. Flask fetches this data from AWS DynamoDB, which persistently stores all user bookings. The dashboard UI, powered by responsive HTML/CSS and Flask templates, ensures users can review or manage bookings anytime, from any device, with real-time updates and quick cancellation workflows supported.

## AWS ARCHITECTURE



Entity Relationship (ER)Diagram:



## Pre-requisites:

1. AWS Account Setup: [AWS Account Setup](#)
2. Understanding IAM: [IAM Overview](#)
3. Amazon EC2 Basics: [EC2 Tutorial](#)
4. DynamoDB Basics: [DynamoDB Introduction](#)
5. SNS Overview: [SNS Documentation](#)
6. Git Version Control: [Git Documentation](#)

## Project WorkFlow:

### 1. AWS Account Setup and Login

**Activity 1.1:** Set up an AWS account if not already done.

**Activity 1.2:** Log in to the AWS Management Console

### 2. DynamoDB Database Creation and Setup

**Activity 2.1:** Create a DynamoDB Table.

**Activity 2.2:** Configure Attributes for User Data and Book Requests.

### 3. SNS Notification Setup

**Activity 3.1:** Create SNS topics for book request notifications.

**Activity 3.2:** Subscribe users and library staff to SNS email notifications.

### 4. Backend Development and Application Setup

**Activity 4.1:** Develop the Backend Using Flask.

**Activity 4.2:** Integrate AWS Services Using boto3.

### 5. IAM Role Setup

**Activity 5.1:** Create IAM Role

**Activity 5.2:** Attach Policies

### 6. EC2 Instance Setup

**Activity 6.1:** Launch an EC2 instance to host the Flask application.

**Activity 6.2:** Configure security groups for HTTP, and SSH access.

### 7. Deployment on EC2

**Activity 7.1:** Upload Flask Files

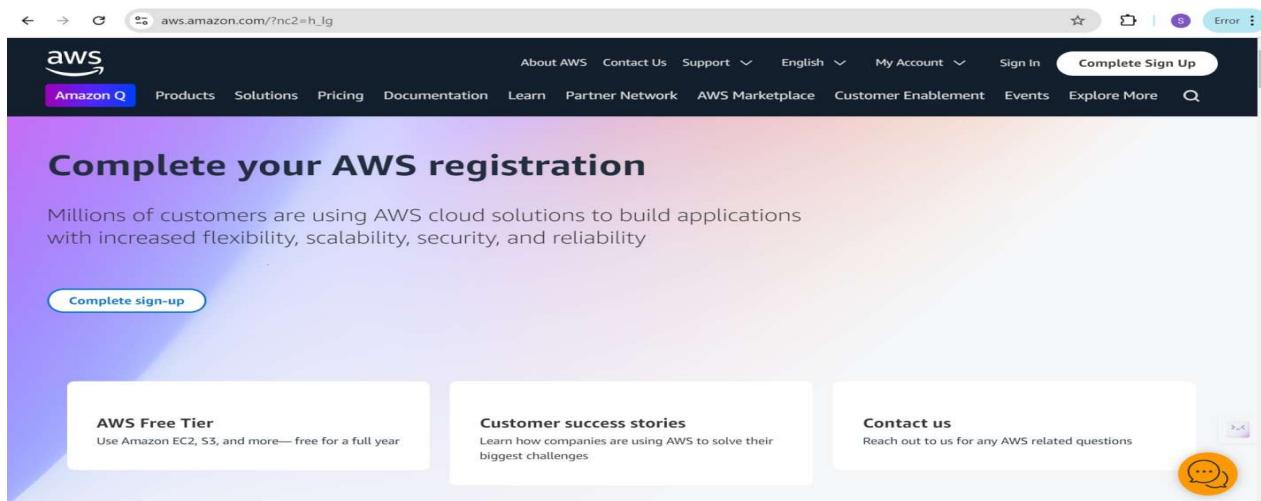
**Activity 7.2:** Run the Flask App

## 8. Testing and Deployment

**Activity 8.1:** Conduct functional testing to verify user registration, login, book requests, and notifications.

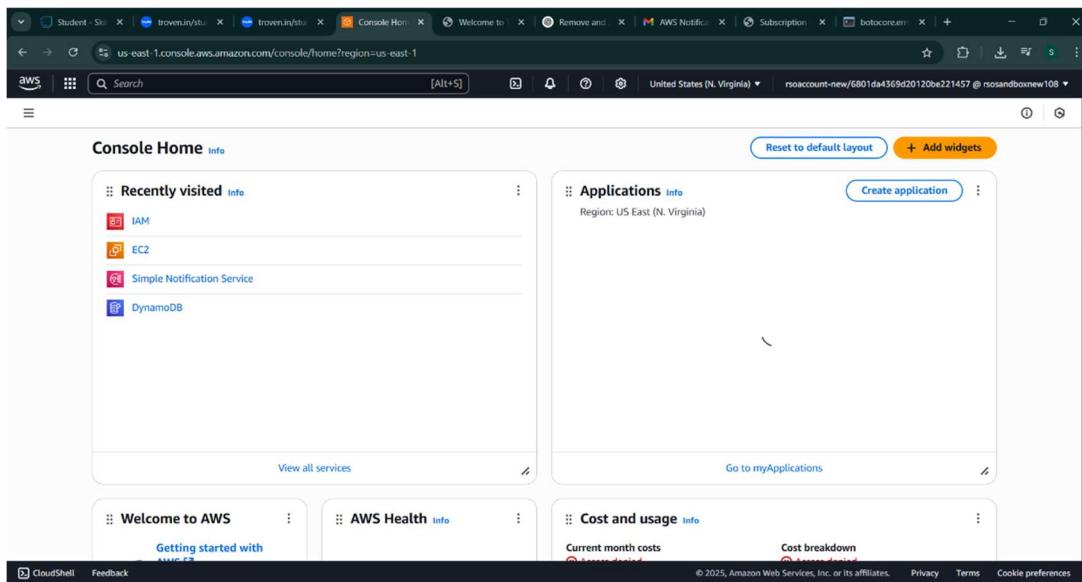
### Milestone 1: AWS Account Setup and Login

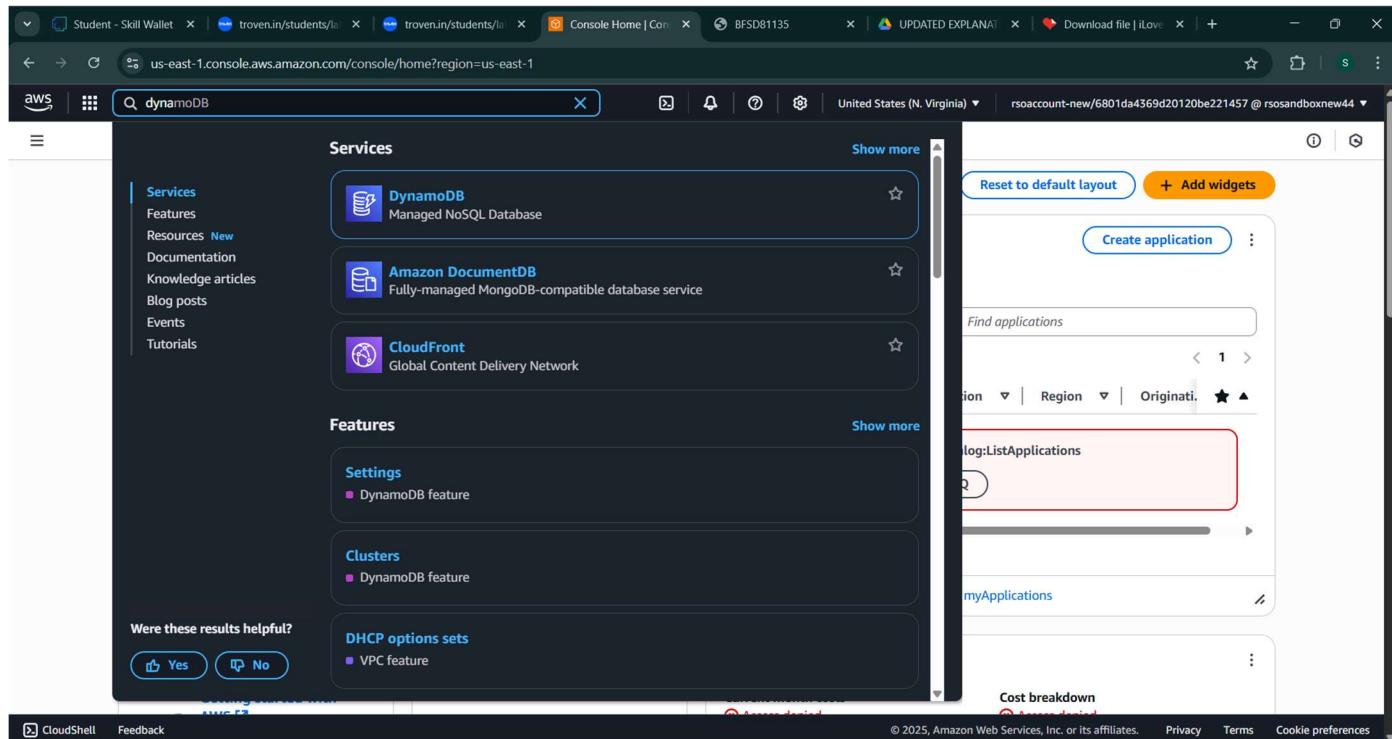
- **Activity 1.1: Set up an AWS account if not already done.**
  - Sign up for an AWS account and configure billing settings.



- **Activity 1.2: Log in to the AWS Management Console**

- After setting up your account, log in to the [AWS Management Console](#).





The screenshot shows the AWS Console search results for 'dynamoDB'. The search bar at the top contains 'dynamoDB'. The results are categorized under 'Services' and 'Features'.

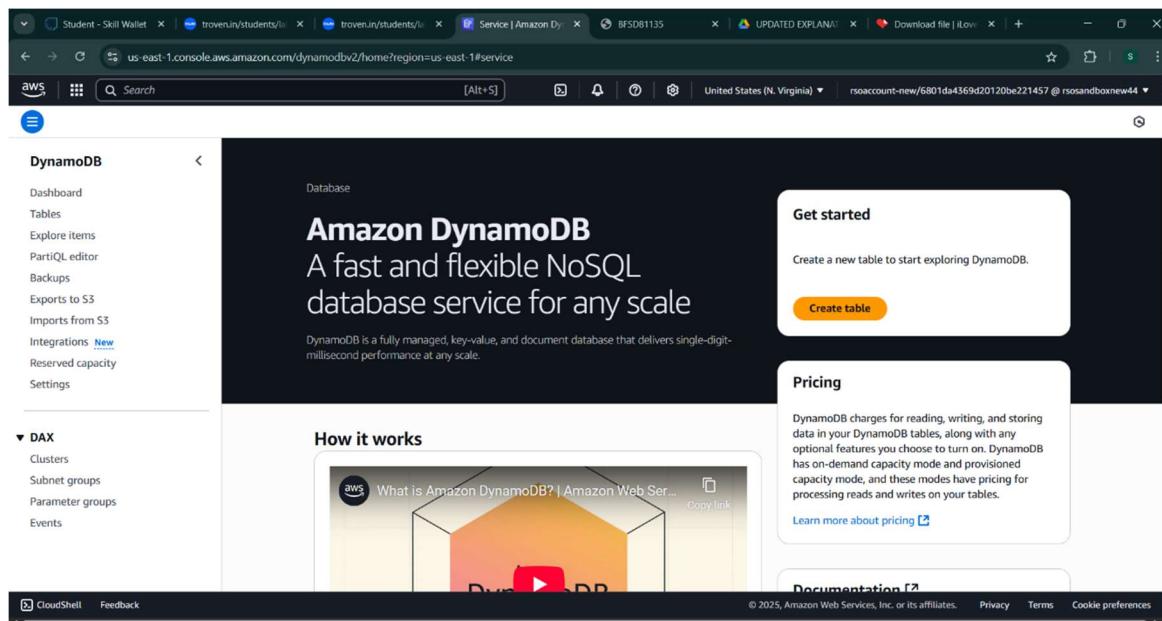
- Services:**
  - DynamoDB**: Managed NoSQL Database
  - Amazon DocumentDB**: Fully-managed MongoDB-compatible database service
  - CloudFront**: Global Content Delivery Network
- Features:**
  - Settings**: DynamoDB feature
  - Clusters**: DynamoDB feature
  - DHCP options sets**: VPC feature

At the bottom left, there are 'Were these results helpful?' buttons ('Yes' or 'No'). On the right side, there are options to 'Reset to default layout' and '+ Add widgets'. Below the search bar, there are links for 'Create application', 'Find applications', and 'Region'. A red box highlights the 'Find applications' search bar.

## Milestone 2: DynamoDB Database Creation and Setup

- **Activity 2.1: Navigate to the DynamoDB**

- In the AWS Console, navigate to DynamoDB and click on create tables.



The screenshot shows the Amazon DynamoDB service page. The left sidebar has a 'DynamoDB' section with links like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Below that is a 'DAX' section with Clusters, Subnet groups, Parameter groups, and Events.

The main content area features a large heading 'Amazon DynamoDB' with the subtext 'A fast and flexible NoSQL database service for any scale'. It includes a 'Get started' button and a 'Pricing' section. A video player titled 'What is Amazon DynamoDB? | Amazon Web Services' is embedded in the page.

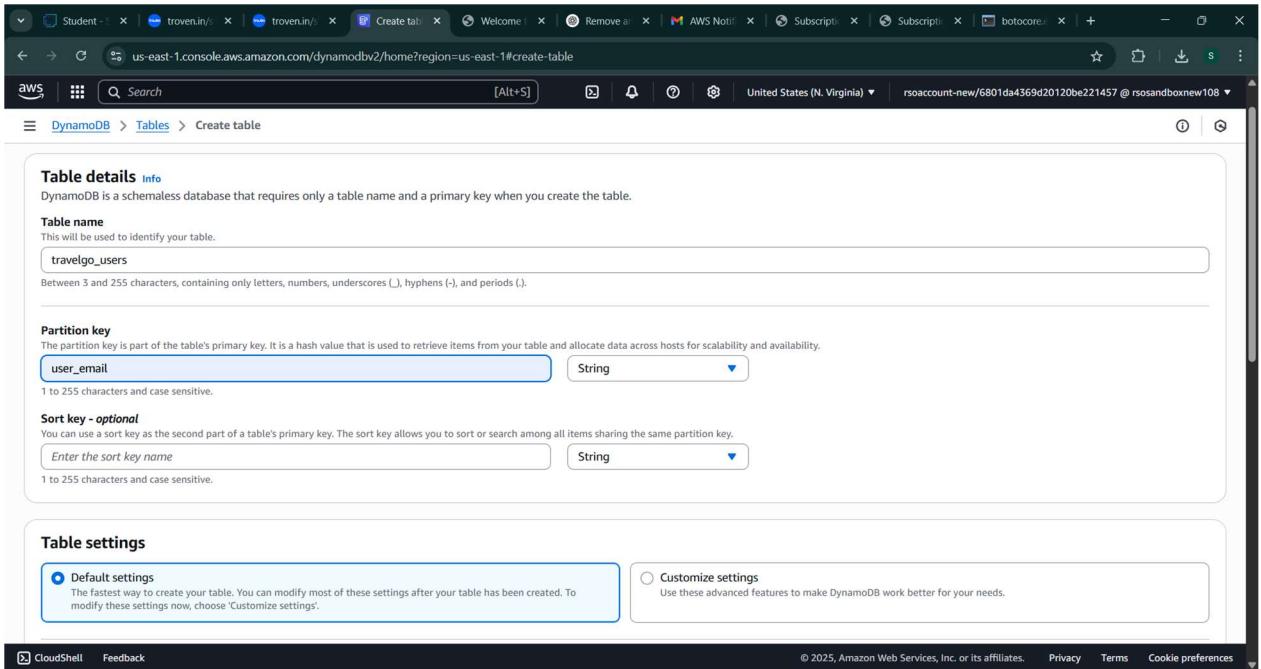
On the right, there are links for 'Documentation' and 'Learn more about pricing'. At the bottom, there are standard AWS footer links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

9.  
10.

- **Activity 2.2:Create a DynamoDB table for storing registration details and book requests.**

- Create Users table with partition key “user\_email” with type String and click on create tables.

**11.**



The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The 'Table details' step is active. The table name is set to 'travelgo\_users'. The partition key is 'user\_email' of type String. There is no sort key defined. Under 'Table settings', the 'Default settings' option is selected. The bottom of the screen includes standard AWS navigation links like CloudShell, Feedback, and copyright information.



Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

### Tags

Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

[Add new tag](#)

You can add 50 more tags.

[Cancel](#)

[Create table](#)

- Follow the same steps to create a requests table with `user_email` as the primary key for book requests data.

us-east-1.console.aws.amazon.com/dynamodbv2/home#create-table

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

**Table name**  
 This will be used to identify your table.  
 Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.)

**Partition key**  
 The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.  
 String  
 1 to 255 characters and case sensitive.

**Sort key - optional**  
 You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.  
 String  
 1 to 255 characters and case sensitive.

**Table settings**

**Default settings**  
 The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these settings now, choose 'Customize settings'.

**Customize settings**  
 Use these advanced features to make DynamoDB work better for your needs.

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

us-east-1.console.aws.amazon.com/dynamodbv2/home#create-table

**Create table**

**Table details** Info  
 DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

**Table name**  
 This will be used to identify your table.  
 Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.)

**Partition key**  
 The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.  
 String  
 1 to 255 characters and case sensitive.

**Sort key - optional**  
 You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.  
 String  
 1 to 255 characters and case sensitive.

**Table settings**

**Default settings**  
 The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these settings now, choose 'Customize settings'.

**Customize settings**  
 Use these advanced features to make DynamoDB work better for your needs.

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Screenshot of the AWS DynamoDB 'Create table' wizard.

The configuration table shows the following settings:

Maximum write capacity units	-	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	AWS owned key	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

**Tags**

Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

[Add new tag](#)

You can add 50 more tags.

ⓘ This table will be created with auto scaling deactivated. You do not have permissions to turn on auto scaling.

[Cancel](#) [Create table](#)

## Milestone 3: SNS Notification Setup

- **Activity 3.1: Create SNS topics for sending email notifications to users**

Screenshot of the AWS Simple Notification Service (SNS) console.

The left sidebar shows the navigation menu for SNS, including Services, Features, Resources, Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials.

The main content area displays the following services:

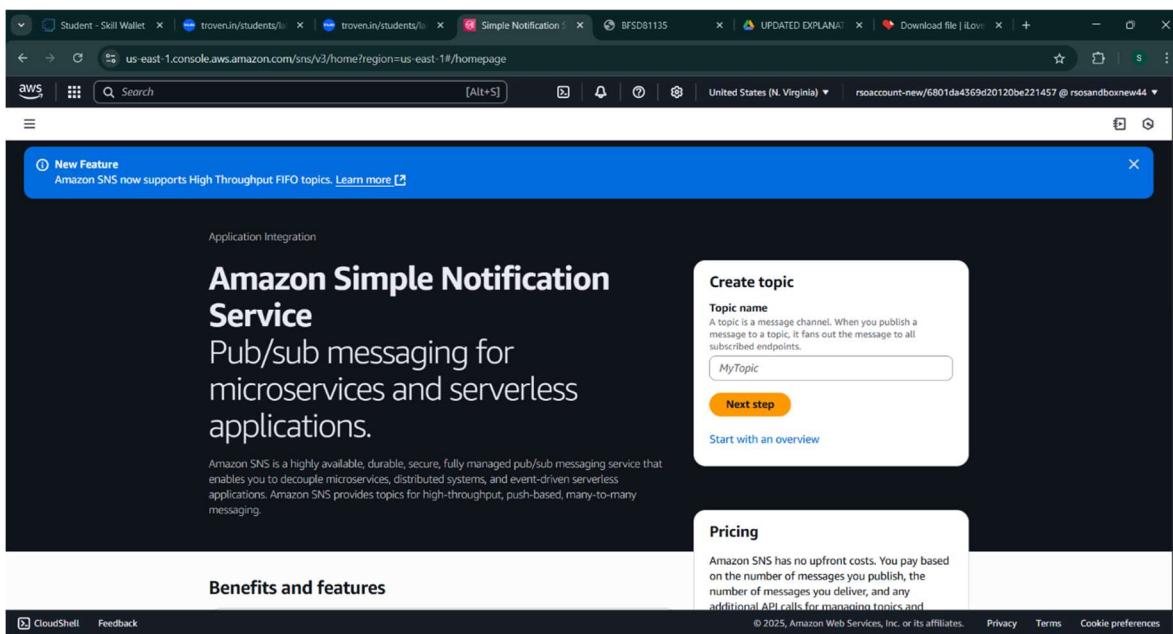
- Services**
  - Simple Notification Service**: SNS managed message topics for Pub/Sub.
  - Route 53 Resolver**: Resolve DNS queries in your Amazon VPC and on-premises network.
  - Route 53**: Scalable DNS and Domain Name Registration.
- Features**
  - Events**: ElastiCache feature.
  - SMS**: AWS End User Messaging feature.
  - Hosted zones**: Route 53 feature.

A modal window is open on the right side, showing the configuration for a new topic. The configuration includes:

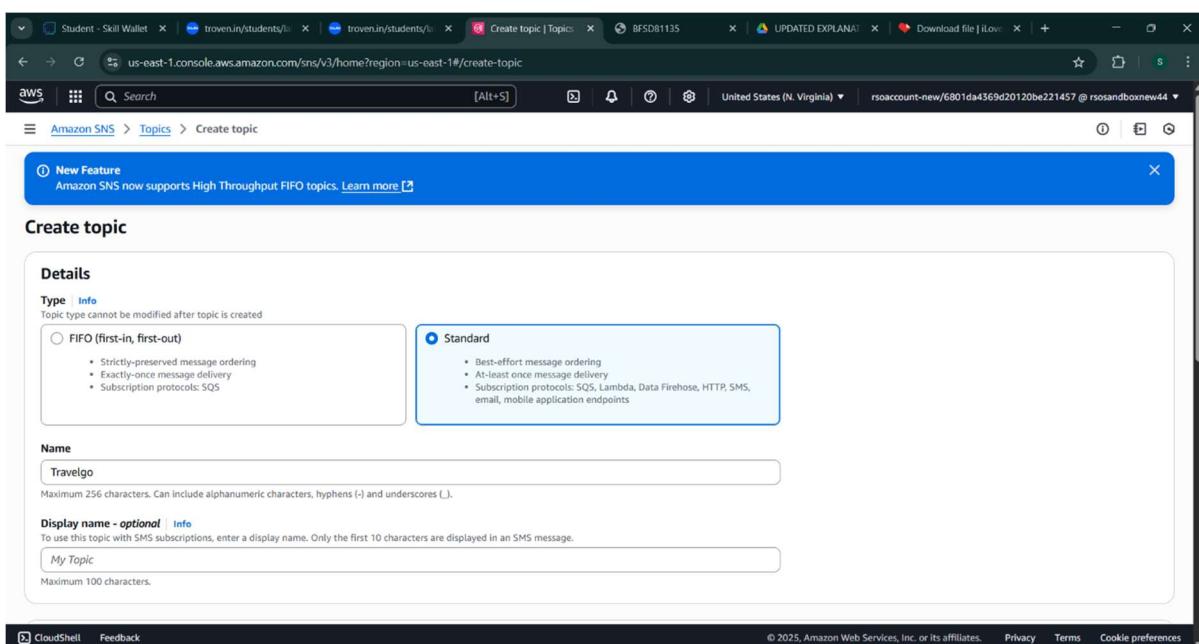
- Topic name: Any tag value
- Region: United States (N. Virginia)
- Amazon Regions: Off
- Deletion protection: Off
- Favorite: On-demand
- Read capacity: Off
- Write capacity: Off

[Actions](#) [Delete](#) [Create topic](#)

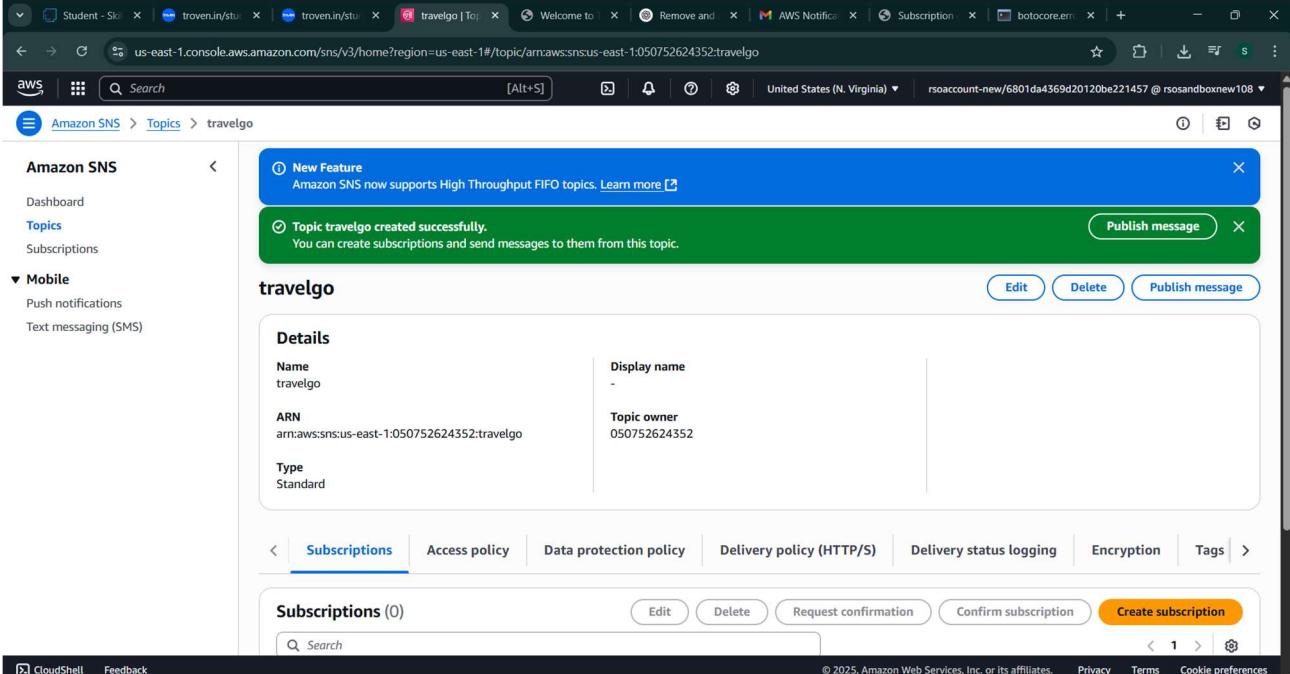
- In the AWS Console, search for SNS and navigate to the SNS Dashboard.



- Click on **Create Topic** and choose a name for the topic.

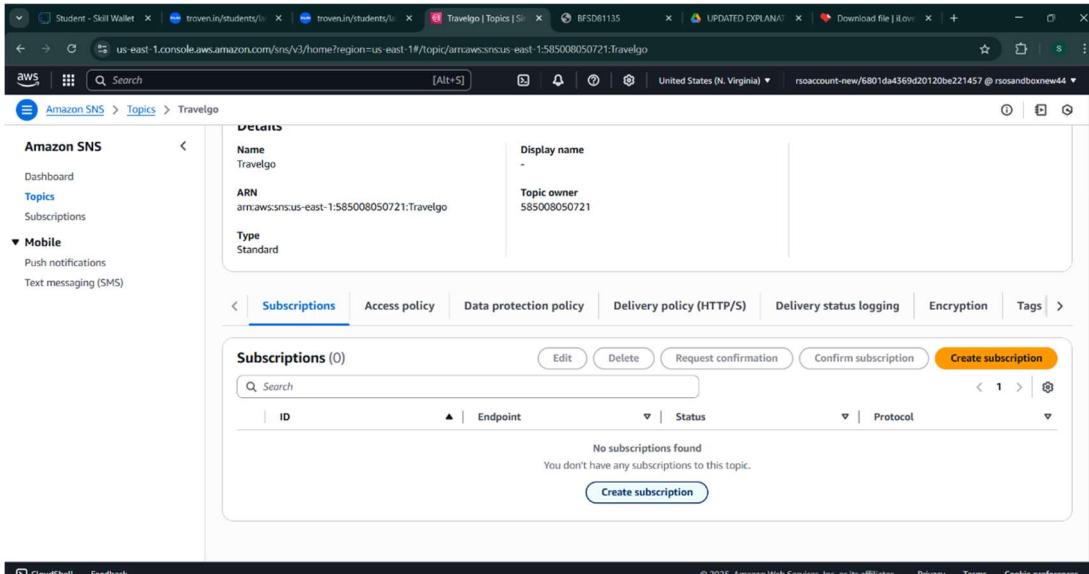


- Choose Standard type for general notification use cases and Click on Create Topic.



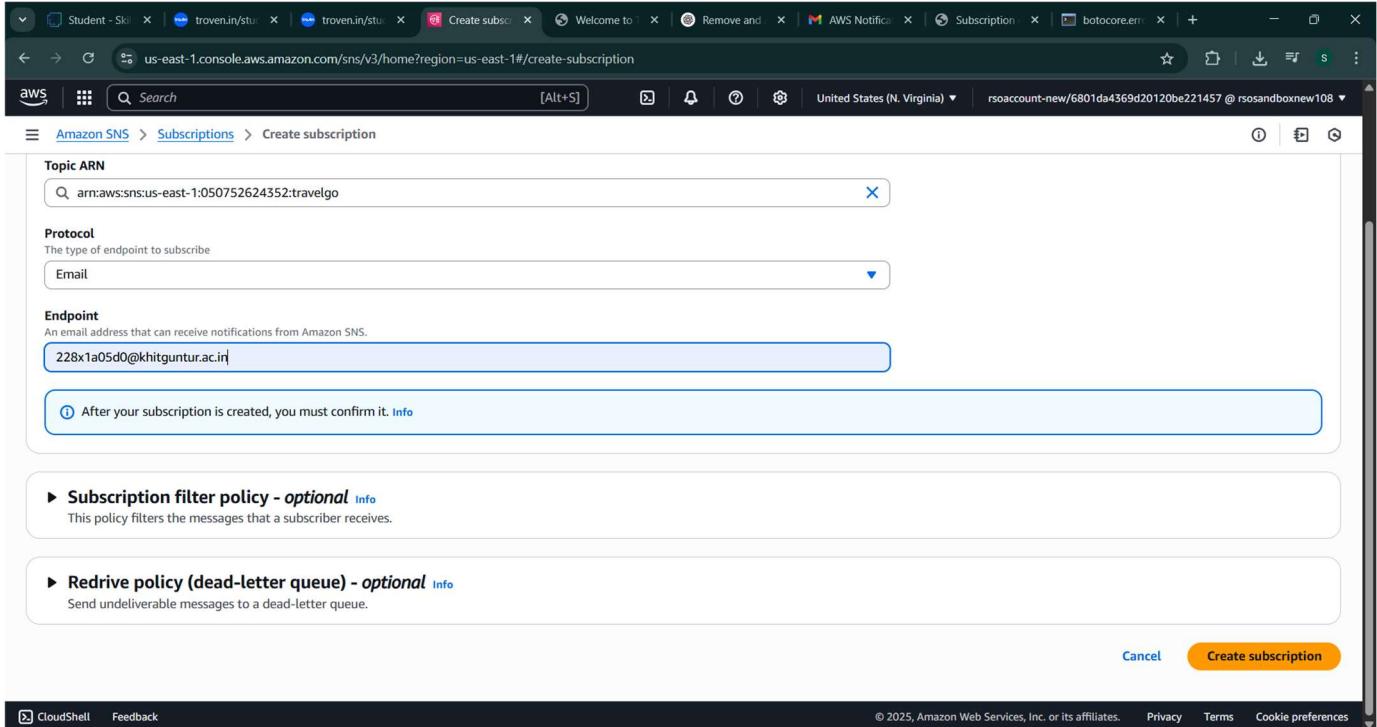
The screenshot shows the AWS SNS Topics page. A success message at the top right states: "Topic travelgo created successfully. You can create subscriptions and send messages to them from this topic." Below this, the "Details" section for the "travelgo" topic is shown, including its Name (travelgo), ARN (arn:aws:sns:us-east-1:050752624352:travelgo), and Type (Standard). The "Display name" and "Topic owner" fields are also listed. At the bottom of the page, there are tabs for "Subscriptions", "Access policy", "Data protection policy", "Delivery policy (HTTP/S)", "Delivery status logging", "Encryption", and "Tags". The "Subscriptions" tab is selected, showing a table with one row: "Subscriptions (0)". There are buttons for "Edit", "Delete", "Request confirmation", "Confirm subscription", and "Create subscription".

- Configure the SNS topic and note down the **Topic ARN**.
- **Activity 3.2: Subscribe users relevant SNS topics to receive real-time notifications when a book request is made.**



The screenshot shows the same AWS SNS Topics page for the "Travelgo" topic. The "Subscriptions" section is now active, displaying a table with the header: "ID", "Endpoint", "Status", and "Protocol". A message below the table states: "No subscriptions found. You don't have any subscriptions to this topic." There is a "Create subscription" button at the bottom of the table.

- Subscribe users to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails.



Topic ARN  
arn:aws:sns:us-east-1:050752624352:travelgo

Protocol  
Email

Endpoint  
228x1a05d0@khitguntur.ac.in

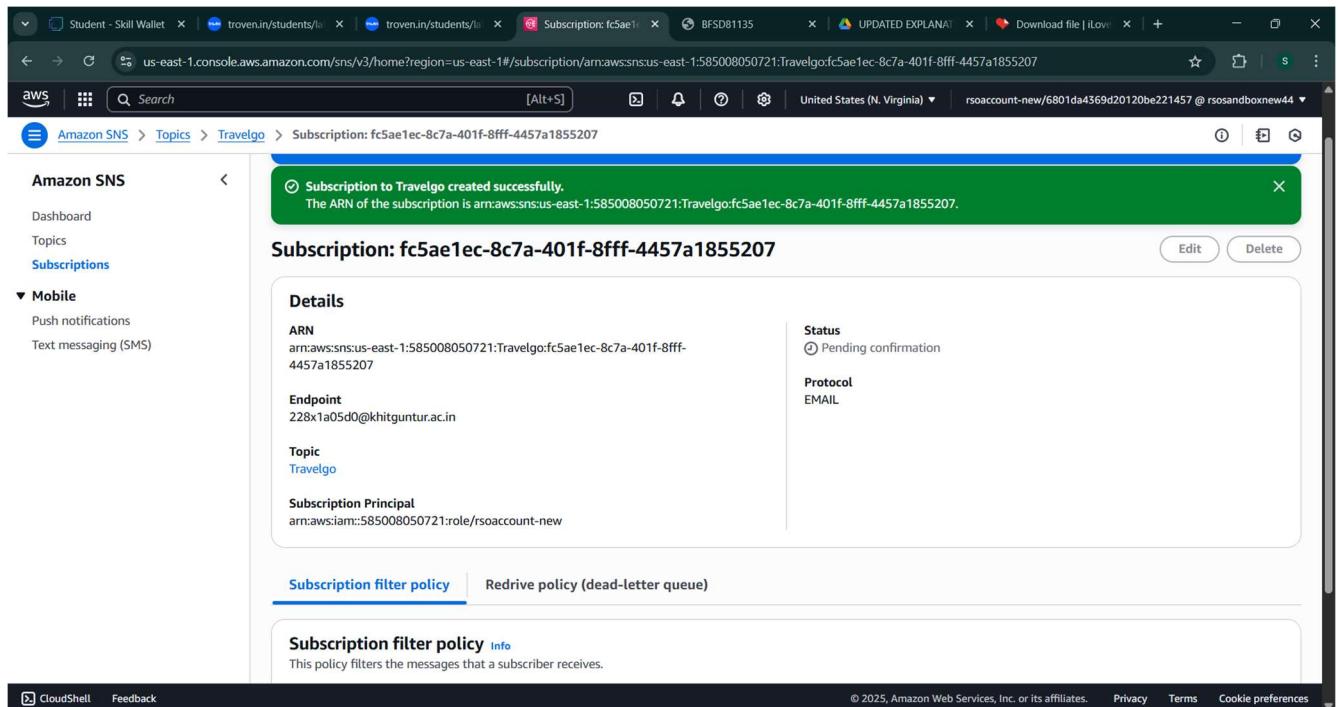
After your subscription is created, you must confirm it. [Info](#)

► Subscription filter policy - *optional* [Info](#)  
This policy filters the messages that a subscriber receives.

► Redrive policy (dead-letter queue) - *optional* [Info](#)  
Send undeliverable messages to a dead-letter queue.

[Cancel](#) [Create subscription](#)

- After subscription request for the mail confirmation



Subscription to Travelgo created successfully.  
The ARN of the subscription is arn:aws:sns:us-east-1:585008050721:Travelgo:fc5ae1ec-8c7a-401f-8fff-4457a1855207.

**Subscription: fc5ae1ec-8c7a-401f-8fff-4457a1855207**

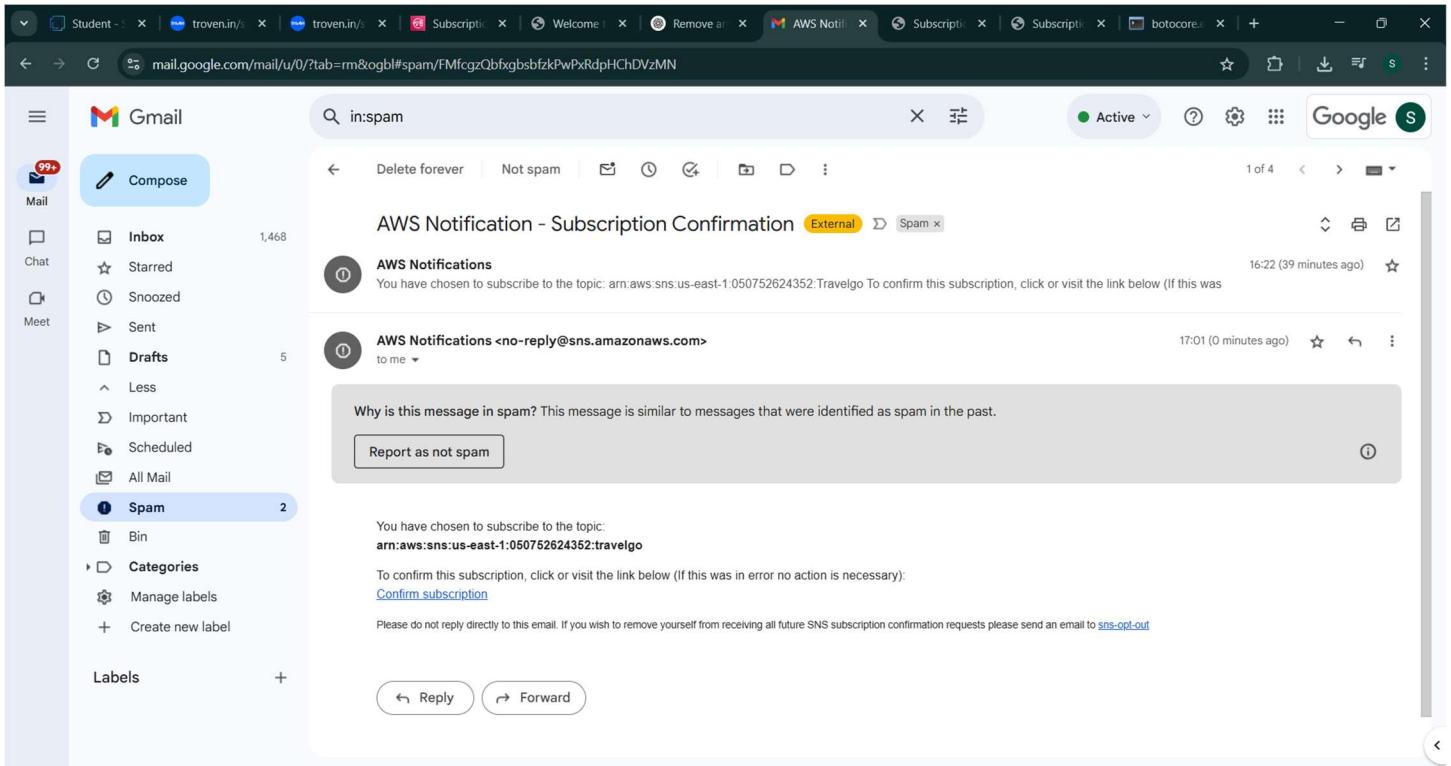
Details	Status
ARN arn:aws:sns:us-east-1:585008050721:Travelgo:fc5ae1ec-8c7a-401f-8fff-4457a1855207	Pending confirmation
Endpoint 228x1a05d0@khitguntur.ac.in	Protocol EMAIL
Topic Travelgo	
Subscription Principal arn:aws:iam::585008050721:role/rsoaccount-new	

[Edit](#) [Delete](#)

[Subscription filter policy](#) [Redrive policy \(dead-letter queue\)](#)

**Subscription filter policy** [Info](#)  
This policy filters the messages that a subscriber receives.

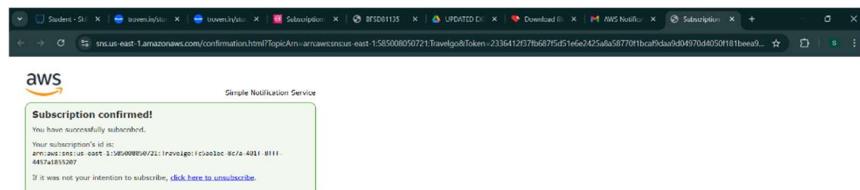
- Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.



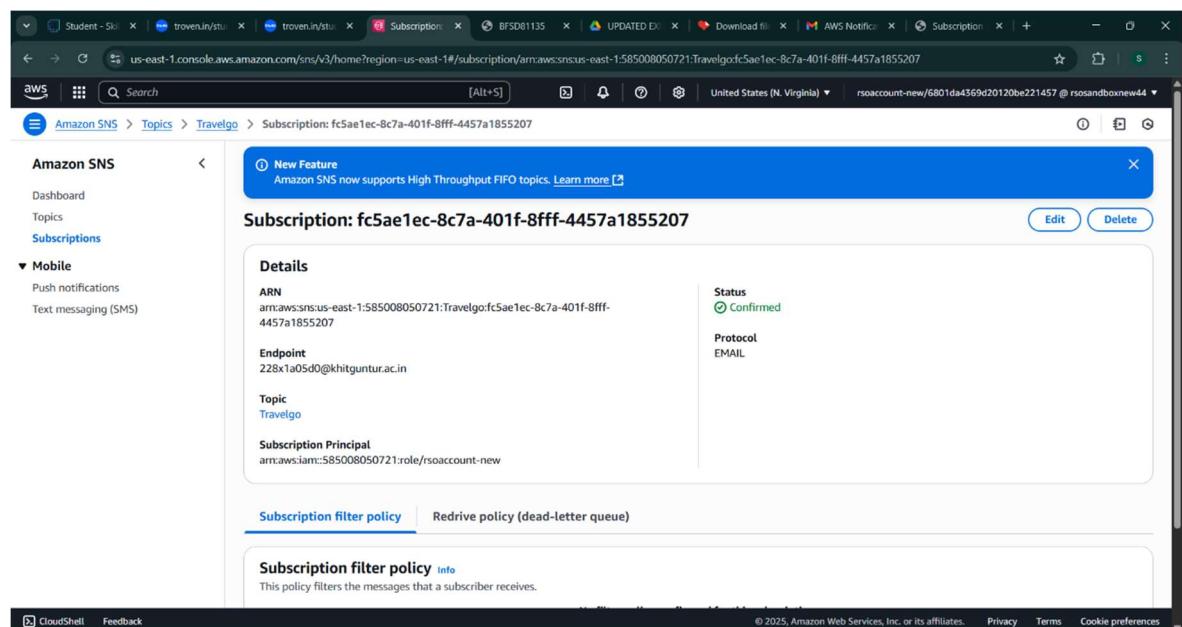
The screenshot shows a Gmail inbox with the search bar set to "in:spam". There are four messages listed:

- AWS Notification - Subscription Confirmation** (External, Spam) - Sent at 16:22 (39 minutes ago). It contains a link to confirm the subscription: <https://aws.amazon.com/sns/confirm-subscription/?arn=arn:aws:sns:us-east-1:050752624352:travelgo>.
- AWS Notifications <no-reply@sns.amazonaws.com>** (to me) - Sent at 17:01 (0 minutes ago).
- Why is this message in spam? This message is similar to messages that were identified as spam in the past.**
- Report as not spam**

The left sidebar shows the navigation menu: Mail (99+), Chat, Meet, Compose, Inbox (1,468), Starred, Snoozed, Sent, Drafts (5), Less, Important, Scheduled, All Mail, and **Spam** (2). Labels section has a plus sign.



- Successfully done with the SNS mail subscription and setup, now store the ARN link.

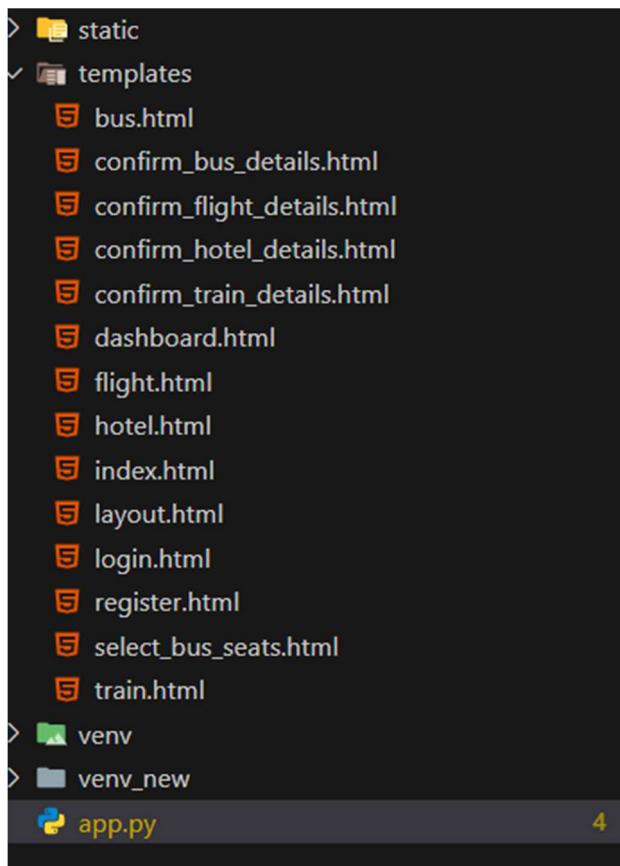


The screenshot shows the AWS Amazon SNS console with the URL <https://us-east-1.console.aws.amazon.com/sns/v3/home?region=us-east-1#/subscription/arm:aws:sns:us-east-1:585008050721:Travelgo:fc5ae1ec-8c7a-401f-8fff-4457a1855207>. The page displays the details of a subscription named "Subscription: fc5ae1ec-8c7a-401f-8fff-4457a1855207". The subscription ARN is listed as `arn:aws:sns:us-east-1:585008050721:Travelgo:fc5ae1ec-8c7a-401f-8fff-4457a1855207`. The endpoint is `228x1a05-d0@khitguntur.ac.in`. The topic is `Travelgo`. The subscription principal is `arn:aws:iam::585008050721:role/ssoaccount-new`. The status is **Confirmed**. The protocol is set to **EMAIL**.

## Milestone 4:Backend Development and Application Setup

- **Activity 4.1: Develop the backend using Flask**

- File Explorer Structure



**Description:** set up the **TravelGO** project with an app.py file, a static/ folder for assets, and a templates/ directory containing all required HTML pages like home, login, register, booking-specific pages (e.g., bus.html, train.html)

### Description of the code :

- **Flask App Initialization**

```
from flask import Flask, render_template, request, redirect, url_for, session, jsonify, flash
import boto3
from boto3.dynamodb.conditions import Key, Attr
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime
from decimal import Decimal
import uuid
import random
```

**Description:** import essential libraries including Flask utilities for routing, Boto3 for DynamoDB operations,

```
app = Flask(__name__)
```

**Description:** initialize the Flask application instance using Flask(\_\_name\_\_) to start building the web app.

- **Dynamodb Setup:**

```
2 # AWS Setup using IAM Role
3 REGION = 'us-east-1' # Replace with your actual AWS region
4 dynamodb = boto3.resource('dynamodb', region_name=REGION)
5 sns_client = boto3.client('sns', region_name=REGION)
6
7 users_table = dynamodb.Table('travelgo_users')
8 trains_table = dynamodb.Table('trains') # Note: This table is declared bu
9 bookings_table = dynamodb.Table('bookings')
```

**Description:** initialize the DynamoDB resource for the us-east-1 region and set up access to the Users and Requests tables for storing user details and book requests.

- **SNS Connection**

```
SNS_TOPIC_ARN = 'arn:aws:sns:us-east-1:418272775181:TravelGo:b7d6f5bc-7710-49de-97bc-ddecff5fd023'
topic ARN

# Function to send SNS notifications

def send_sns_notification(subject, message):
    try:
        sns_client.publish(
            TopicArn=SNS_TOPIC_ARN,
            Subject=subject,
            Message=message
        )
    except Exception as e:
        print(f"SNS Error: Could not send notification - {e}")
        # Optionally, flash an error message to the user or log it more robustly.
    # Function
```

**Description:** Configure SNS to send notifications when a book request is submitted. Paste your stored ARN link in the sns\_topic\_arn space, along with the region\_name where the SNS topic is created.

- **Routes for Web Pages**

- **Home Route:**

```
# Home route redirects to Registration page
@app.route('/')
def home():
    |   return redirect(url_for('register'))
```

**Description:** define the home route / to automatically redirect users to the register page when they access the base URL.

- Register Route:

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        # existing = users_collection.find_one({'email': email}) # MongoDB
        existing = users_table.get_item(Key={'email': email}) # DynamoDB
        if 'Item' in existing:
            flash('Email already exists!', 'error')
            return render_template('register.html')
        hashed_password = generate_password_hash(password)
        # users_collection.insert_one({'email': email, 'password': hashed_password}) # MongoDB
        users_table.put_item(Item={'email': email, 'password': hashed_password}) # DynamoDB
        flash('Registration successful! Please log in.', 'success')
        return redirect(url_for('login'))
    return render_template('register.html')
```

**Description:** define /register route to validate registration form fields, hash the user password using Bcrypt, store the new user in DynamoDB with a login count, and send an SNS notification on successful registration

- login Route (GET/POST):

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if 'username' in session:
        session.pop('username', None)
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        # user = users_collection.find_one({'email': email}) # MongoDB
        user = users_table.get_item(Key={'email': email}) # DynamoDB
        if 'Item' in user and check_password_hash(user['Item']['password'], password):
            session['email'] = email
            flash('Logged in successfully!', 'success')
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid email or password!', 'error')
            return render_template('login.html')
    return render_template('login.html')
```

**Description:** define /login route to validate user credentials against DynamoDB, check the password using Bcrypt, update the login count on successful authentication, and redirect users to the home page

- Home,Bus,Train,Flight,Hotel Routes:

```

@app.route('/dashboard')
def dashboard():
    if 'email' not in session:
        return redirect(url_for('login'))

    user_email = session['email']
    # user_bookings = list(bookings_collection.find({'user_email': user_email}).sort('booking_date', -1)) # MongoDB
    response = bookings_table.query(
        KeyConditionExpression=Key('user_email').eq(user_email),
        ScanIndexForward=False
    )
    bookings = response.get('Items', [])
    for booking in bookings:
        if 'total_price' in booking:
            try:
                booking['total_price'] = float(booking['total_price'])
            except Exception:
                booking['total_price'] = 0.0

    for booking in bookings:
        if '_id' in booking and isinstance(booking['_id'], ObjectId):
            booking['_id'] = str(booking['_id'])

        # Determine vehicle_type for display based on booking_type
        if booking.get('booking_type') == 'bus':
            booking['vehicle_type'] = booking.get('type', 'Bus')
            # Add seat information for bus bookings
            if booking.get('selected_seats'):
                booking['seats_display'] = ', '.join(booking['selected_seats'])
            else:
                booking['seats_display'] = 'N/A'
        elif booking.get('booking_type') == 'train':
            booking['vehicle_type'] = f"Train {booking.get('train_number', '')}" if booking.get('train_number') else "Train"
            # Add seat information for train bookings
            if booking.get('selected_seats'):
                booking['seats_display'] = ', '.join(booking['selected_seats'])
            else:
                booking['seats_display'] = 'N/A'
        elif booking.get('booking_type') == 'flight':
            booking['vehicle_type'] = f"Flight {booking.get('flight_number', '')} ({booking.get('airline', '')})"
            # Add seat information for flights
            if booking.get('selected_seats'):
                booking['seats_display'] = ', '.join(booking['selected_seats'])
            else:
                booking['seats_display'] = 'N/A'
        else:
            booking['vehicle_type'] = booking.get('booking_type', 'N/A')

    return render_template('dashboard.html', username=user_email, bookings=bookings)

```

**Description:** define /dashboard-page to render the main homepage, to handle booking selection and redirection.

- confirmbooking Routes:

```

@app.route('/train')
def train():
    if 'email' not in session:
        return redirect(url_for('login'))
    return render_template('train.html')

@app.route('/confirm_train_details')
def confirm_train_details():
    if 'email' not in session:
        return redirect(url_for('login'))

    name = request.args.get('name')
    train_number = request.args.get('trainNumber')
    source = request.args.get('source')
    destination = request.args.get('destination')
    departure_time = request.args.get('departureTime')
    arrival_time = request.args.get('arrivalTime')
    price_per_person = float(request.args.get('price'))
    travel_date = request.args.get('date')
    num_persons = int(request.args.get('persons'))
    train_id = request.args.get('trainId')

    total_price = price_per_person * num_persons

    booking_details = {
        'name': name,
        'train_number': train_number,
        'source': source,
        'destination': destination,
        'departure_time': departure_time,
        'arrival_time': arrival_time,
        'price_per_person': Decimal(price_per_person),
        'travel_date': travel_date,
        'num_persons': num_persons,
        'total_price': Decimal(total_price),
        'item_id': train_id,
        'booking_type': 'train',
        'user_email': session['email']
    }
    session['pending_booking'] = booking_details
    return render_template('confirm_train_details.html', booking=booking_details)
    
```

**Description:** define /request-form route to capture book request details from users, store the request in DynamoDB, send a thank-you email to the user, notify the admin, and confirm submission with a success message.

#### Exit Route:

```

@app.route('/logout')
def logout():
    session.pop('email', None)
    flash('You have been logged out.', 'info')
    return redirect(url_for('index'))
    
```

**Description:** define /logout route the index.html page to render when the user chooses to leave or close the application.

### Deployment Code:

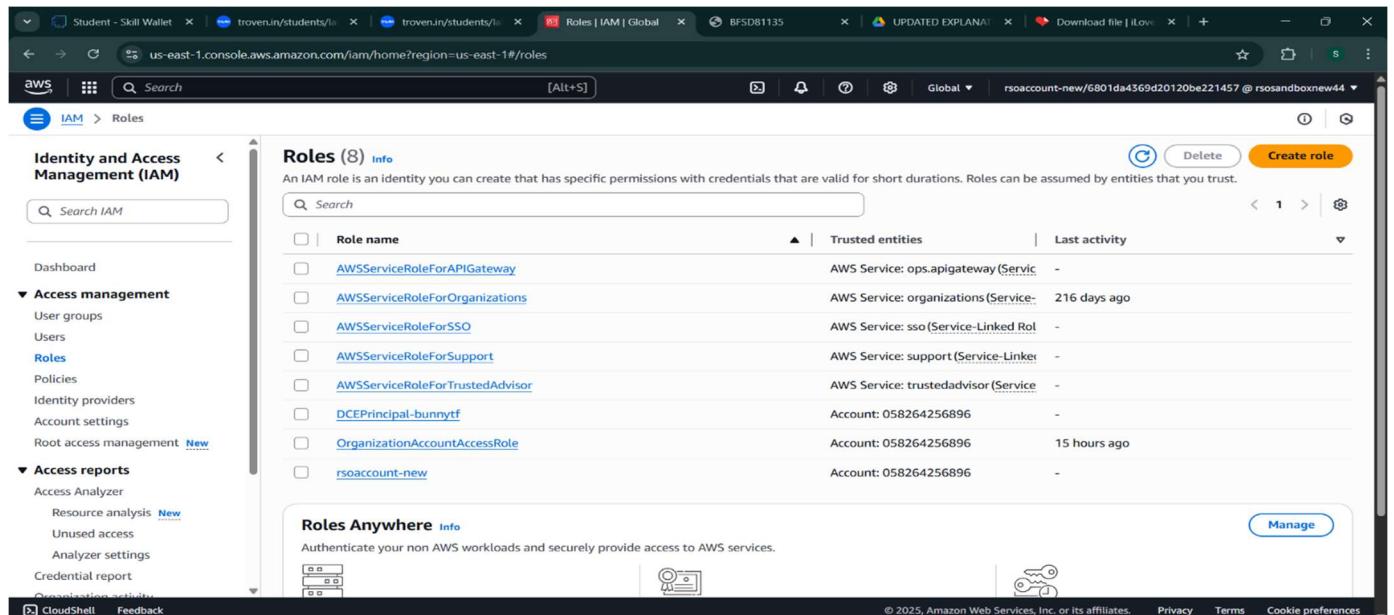
```
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

**Description:** start the Flask server to listen on all network interfaces (0 . 0 . 0 . 0) with debug mode enabled for development and testing.

## Milestone 5: IAM Role Setup

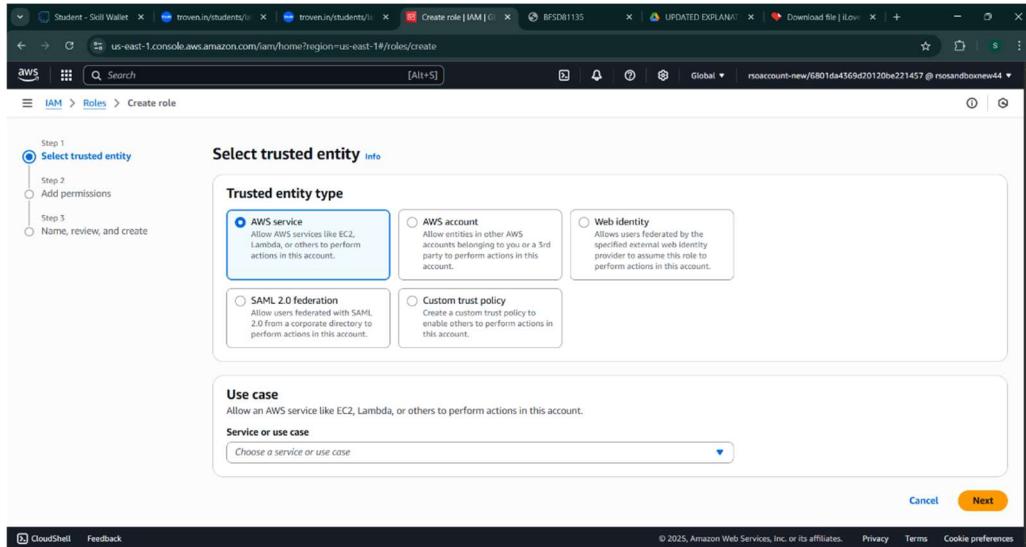
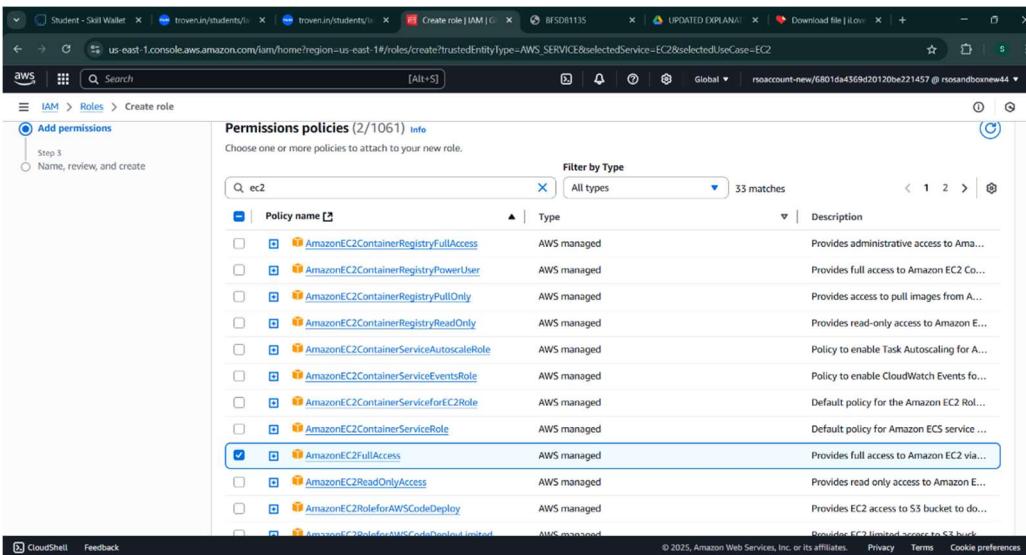
- **Activity 5.1:Create IAM Role.**

- In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.



The screenshot shows the AWS IAM Roles page with 8 roles listed:

Role name	Trusted entities	Last activity
AWSServiceRoleForAPIGateway	AWS Service: ops.apigateway (Service)	-
AWSServiceRoleForOrganizations	AWS Service: organizations (Service)	216 days ago
AWSServiceRoleForSSO	AWS Service: sso (Service-Linked Role)	-
AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service)	-
DCEPrincipal-bunnytf	Account: 058264256896	-
OrganizationAccountAccessRole	Account: 058264256896	15 hours ago
rsoaccount-new	Account: 058264256896	-

## ● Activity 5.2: Attach Policies.

Attach the following policies to the role:

- **AmazonDynamoDBFullAccess:** Allows EC2 to perform read/write operations on DynamoDB.
- **AmazonSNSFullAccess:** Grants EC2 the ability to send notifications via SNS.

Screenshot of the AWS IAM 'Create role' wizard, Step 2: Add permissions.

The search bar shows 'dyna'. The results list includes:

- AmazonDynamoDBFullAccess** (AWS managed) - Provides full access to Amazon DynamoDB.
- AmazonDynamoDBFullAccess\_v2** (AWS managed) - Provides full access to Amazon DynamoDB.
- AmazonDynamoDBFullAccessWithDataPipeline** (AWS managed) - This policy is on a deprecation path. See...
- AmazonDynamoDBReadOnlyAccess** (AWS managed) - Provides read only access to Amazon DynamoDB.
- AWSLambdaDynamoDBExecutionRole** (AWS managed) - Provides list and read access to DynamoDB.
- AWSLambdaInvocation-DynamoDB** (AWS managed) - Provides read access to DynamoDB Stream.

Buttons at the bottom: Cancel, Previous, Next.

Screenshot of the AWS IAM 'Create role' wizard, Step 2: Add permissions.

The search bar shows 'sns'. The results list includes:

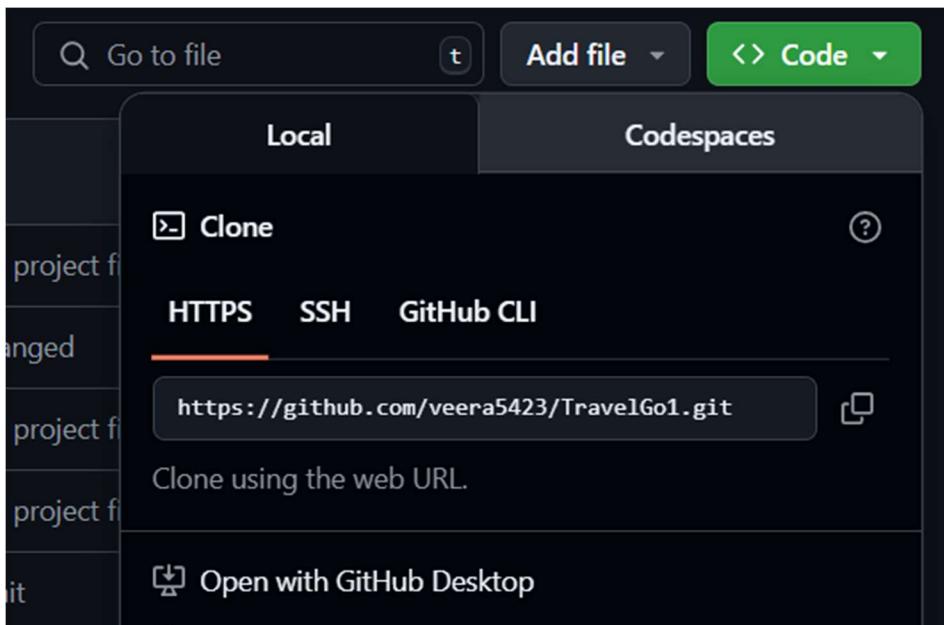
- AmazonSNSFullAccess** (AWS managed) - Provides full access to Amazon SNS via...
- AmazonSNSReadOnlyAccess** (AWS managed) - Provides read only access to Amazon SNS.
- AmazonSNSRole** (AWS managed) - Default policy for Amazon SNS service...
- AWSElasticBeanstalkRoleSNS** (AWS managed) - (Elastic Beanstalk operations role) Allo...
- AWSIoTDeviceDefenderPublishFindingsToSN...** (AWS managed) - Provides messages publish access to S...

Buttons at the bottom: Cancel, Previous, Next.

## Milestone 6: EC2 Instance Setup

- Note: Load your Flask app and Html files into GitHub repository.

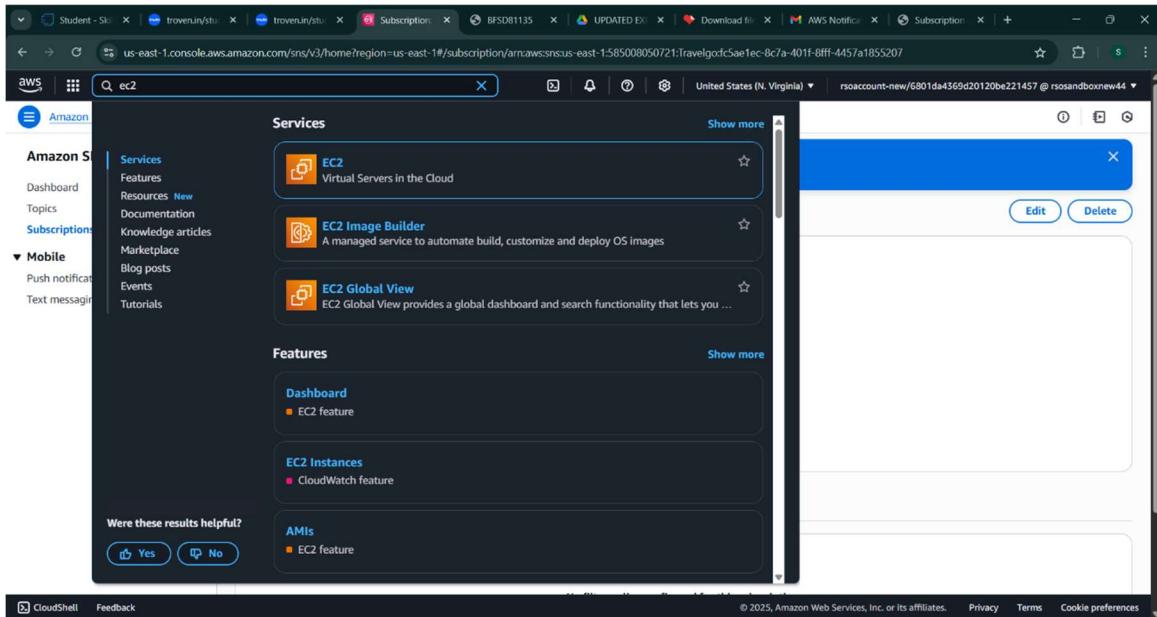
static	Added full project files
templates	app.py changed
venv	Added full project files
venv_new	Added full project files
README.md	first commit
app.py	Update app.py



- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

- **Launch EC2 Instance**

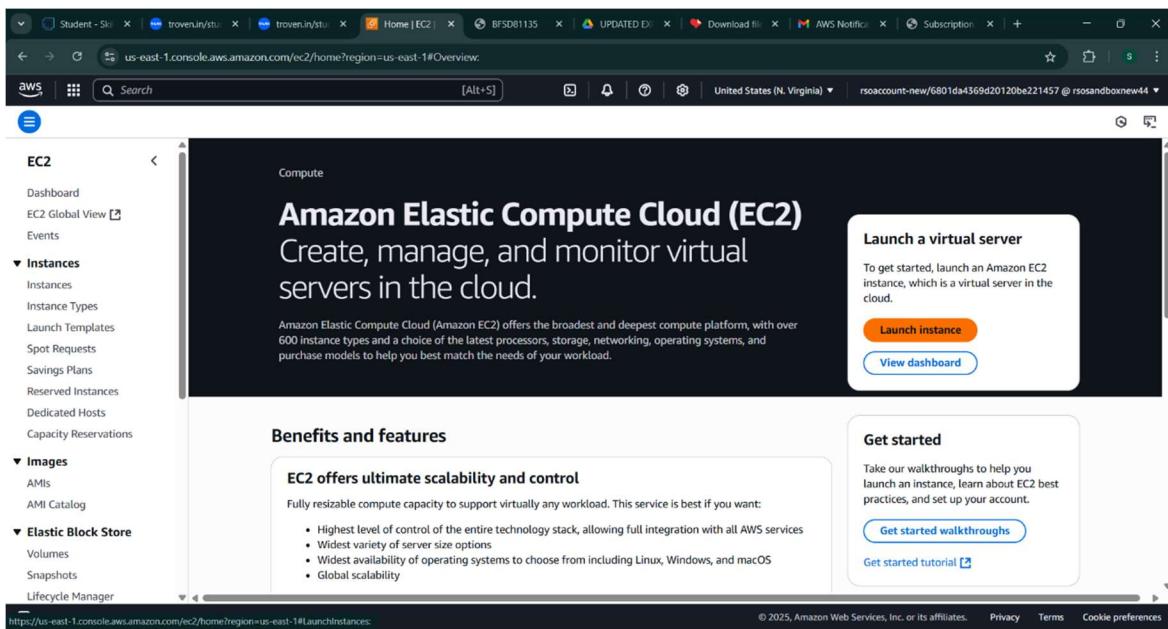
- In the AWS Console, navigate to EC2 and launch a new instance.



The screenshot shows the AWS search interface with the query 'ec2' entered. The results list includes:

- EC2**: Virtual Servers in the Cloud
- EC2 Image Builder**: A managed service to automate build, customize and deploy OS images
- EC2 Global View**: EC2 Global View provides a global dashboard and search functionality that lets you...

On the left sidebar, the 'Amazon Services' section is expanded, showing options like Dashboard, Topics, Subscriptions, Mobile, Push notifications, and Text messaging. The 'EC2' service card is highlighted.



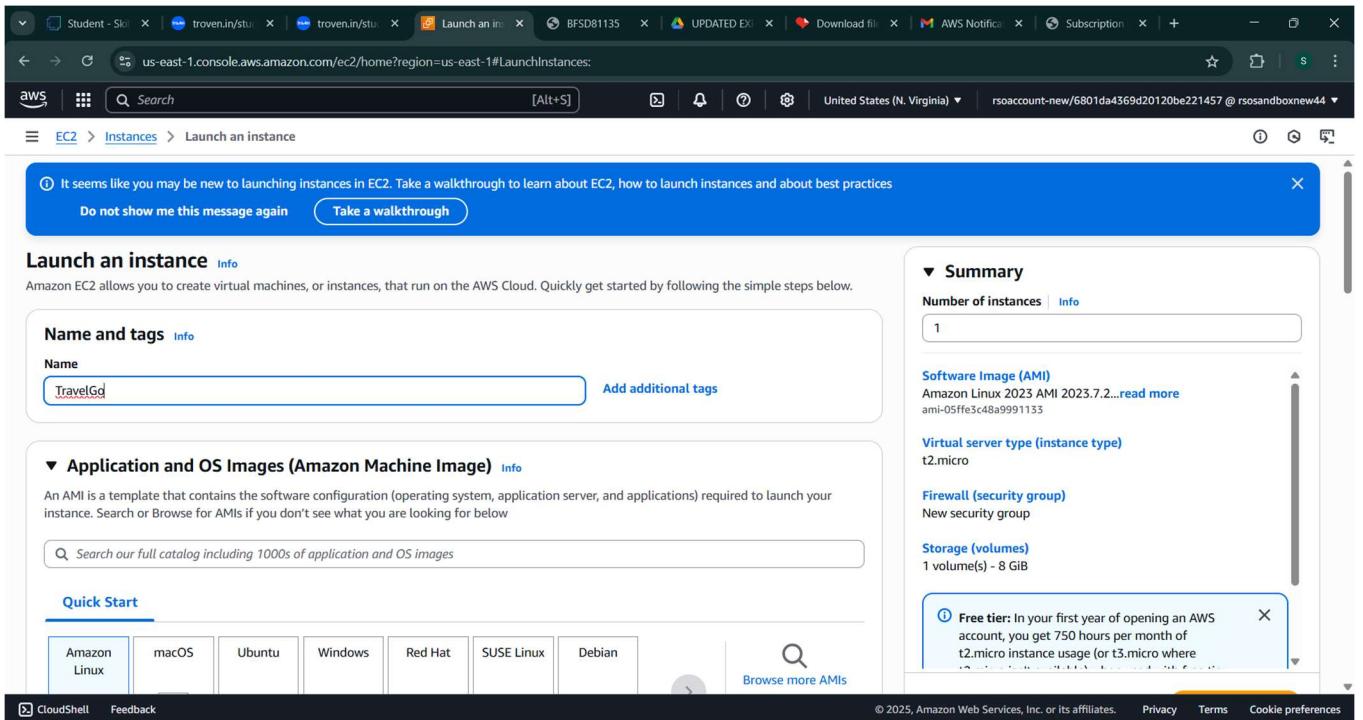
The screenshot shows the AWS EC2 home page. The left sidebar navigation includes:

- EC2
- Dashboard
- EC2 Global View
- Events
- Instances
- Instances Types
- Launch Templates
- Spot Requests
- Savings Plans
- Reserved Instances
- Dedicated Hosts
- Capacity Reservations
- Images
- AMIs
- AMI Catalog
- Elastic Block Store
- Volumes
- Snapshots
- Lifecycle Manager

The main content area features the **Amazon Elastic Compute Cloud (EC2)** heading with the subtext "Create, manage, and monitor virtual servers in the cloud." It includes a brief description of EC2's capabilities and a large "Launch a virtual server" button.

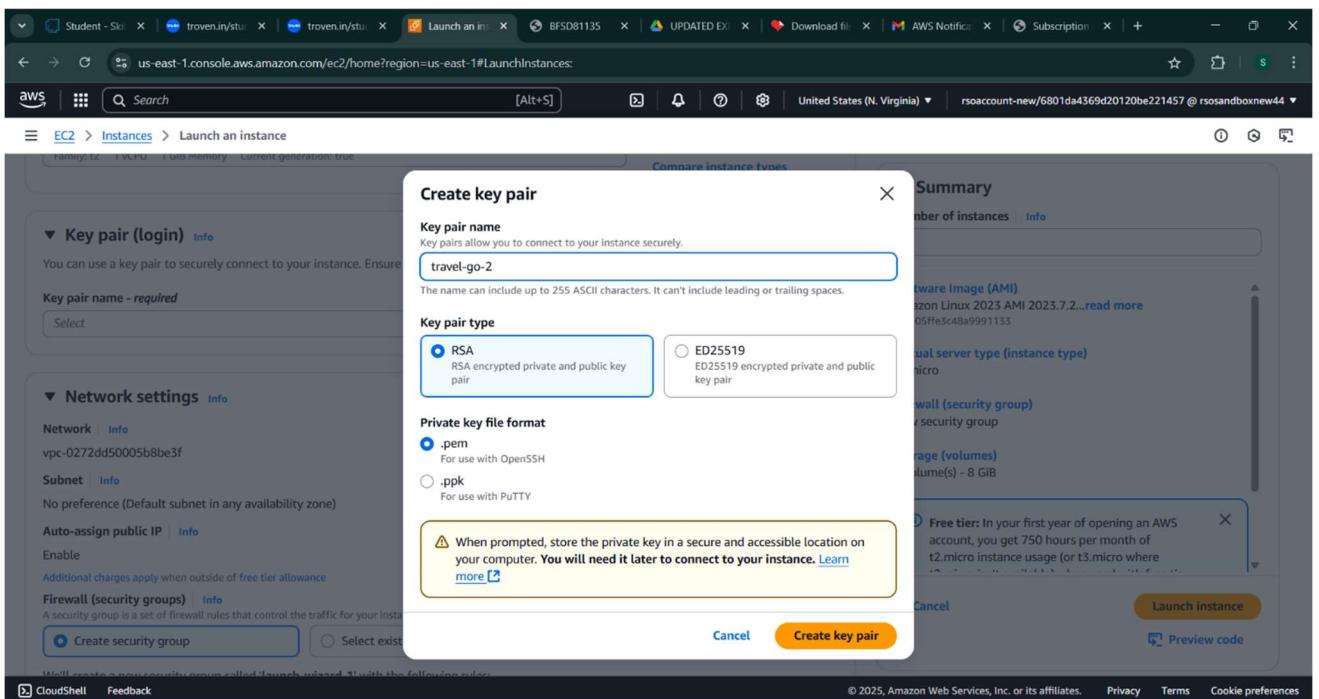
- Click on Launch instance to launch EC2 instance

- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).

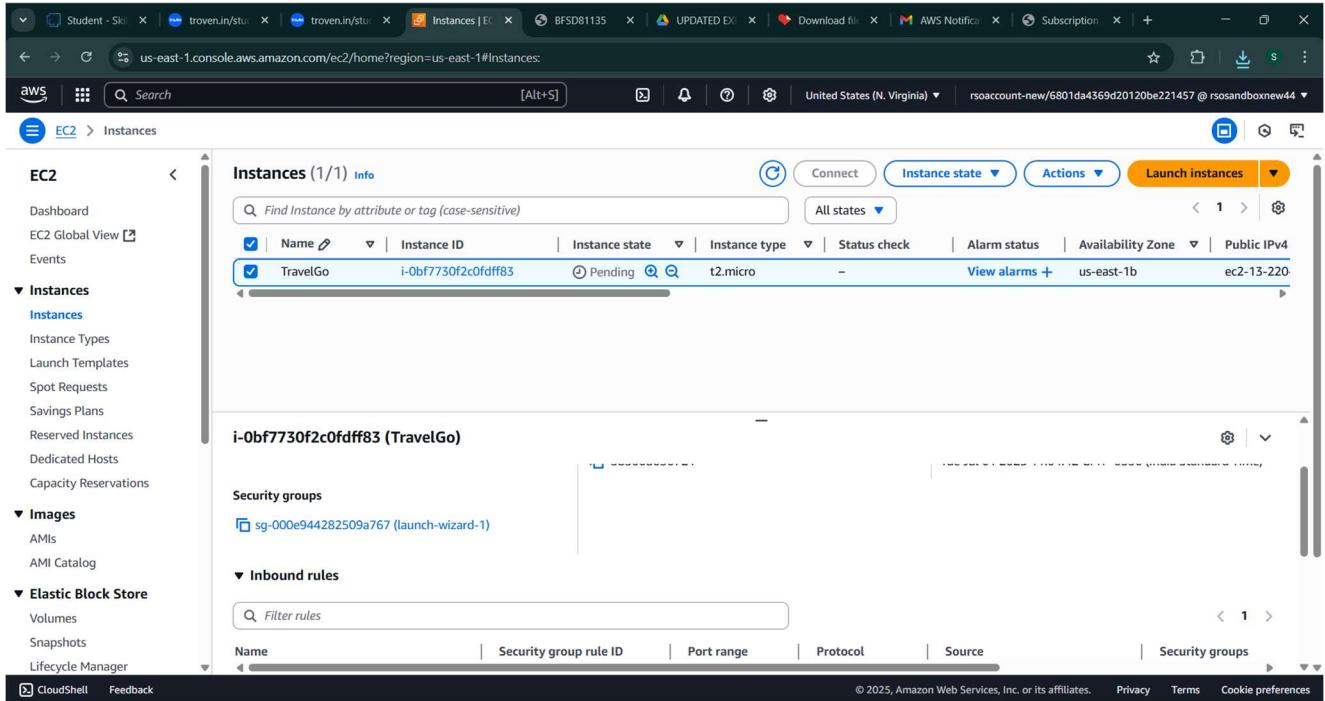


The screenshot shows the AWS EC2 Instances page with the 'Launch an instance' wizard open. In the 'Name and tags' section, the name 'TravelGo' is entered. Under 'Application and OS Images (Amazon Machine Image)', 'Amazon Linux' is selected. The 'Virtual server type (instance type)' is set to 't2.micro'. A summary panel on the right shows 1 instance, the selected AMI, and the chosen instance type. A note about the free tier is visible.

- Create and download the key pair for Server access.

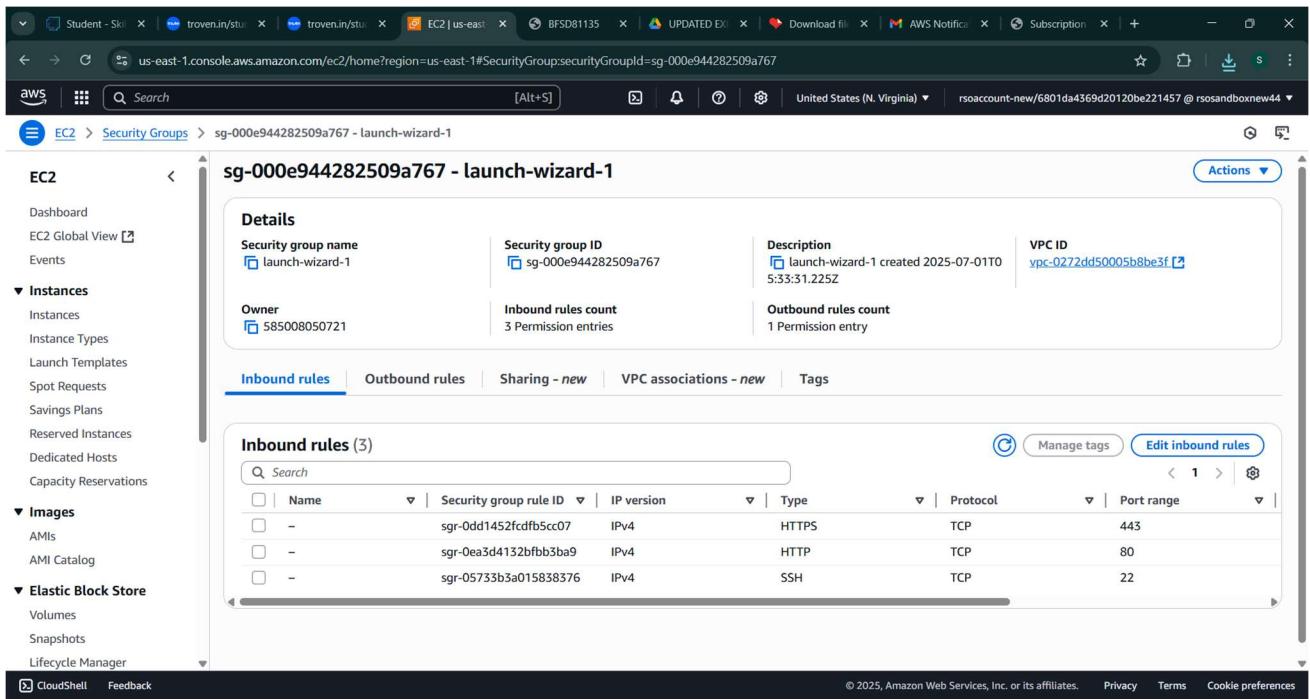


The screenshot shows the 'Create key pair' dialog. The 'Key pair name' is 'travel-go-2'. The 'Key pair type' section has 'RSA' selected. A note at the bottom of the dialog says: 'When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance.' A 'Create key pair' button is at the bottom right.



The screenshot shows the AWS EC2 Instances page. On the left, there is a navigation sidebar with options like Dashboard, EC2 Global View, Events, Instances (with sub-options for Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations), Images (AMIs, AMI Catalog), and Elastic Block Store (Volumes, Snapshots, Lifecycle Manager). The main content area displays the 'Instances (1/1) Info' section. It lists one instance named 'TravelGo' with the ID 'i-0bf7730f2c0fdff83'. The instance is currently 'Pending' and is of type 't2.micro'. It is associated with a security group 'sg-000e944282509a767 (launch-wizard-1)' and is located in the 'us-east-1b' availability zone. There are tabs for 'Actions' and 'Launch instances' at the top of the instance card.

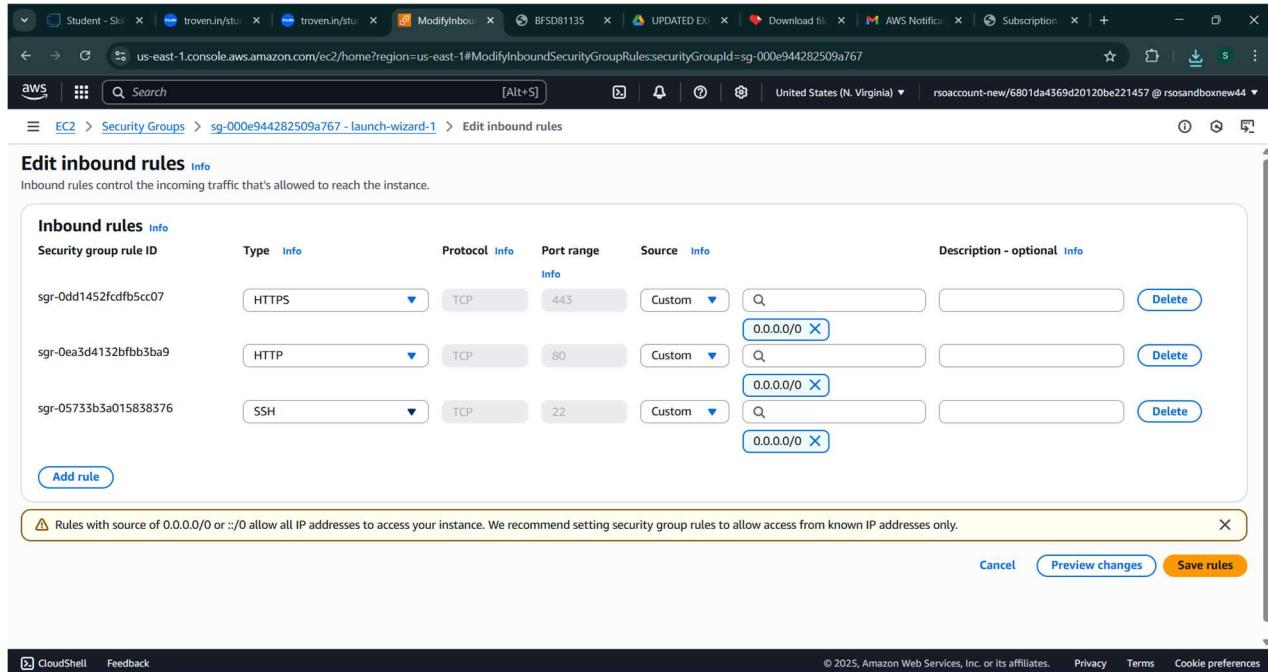
- **Activity 6.2:Configure security groups for HTTP, and SSH access.**



The screenshot shows the AWS Security Groups page. The left sidebar includes options for EC2 Instances, Images, and Elastic Block Store. The main content shows the details for the security group 'sg-000e944282509a767 - launch-wizard-1'. The 'Details' section provides information such as the security group name ('launch-wizard-1'), ID ('sg-000e944282509a767'), owner ('585008050721'), and counts of inbound and outbound rules. The 'Inbound rules' tab is active, listing three rules:

Name	Security group rule ID	IP version	Type	Protocol	Port range
-	sgr-0dd1452fcdfb5cc07	IPv4	HTTPS	TCP	443
-	sgr-0ea3d4132bfb3ba9	IPv4	HTTP	TCP	80
-	sgr-05733b3a015838376	IPv4	SSH	TCP	22

- **Activity 6.2:Configure security groups for HTTP, and SSH access.**



**Edit inbound rules** Info

Inbound rules control the incoming traffic that's allowed to reach the instance.

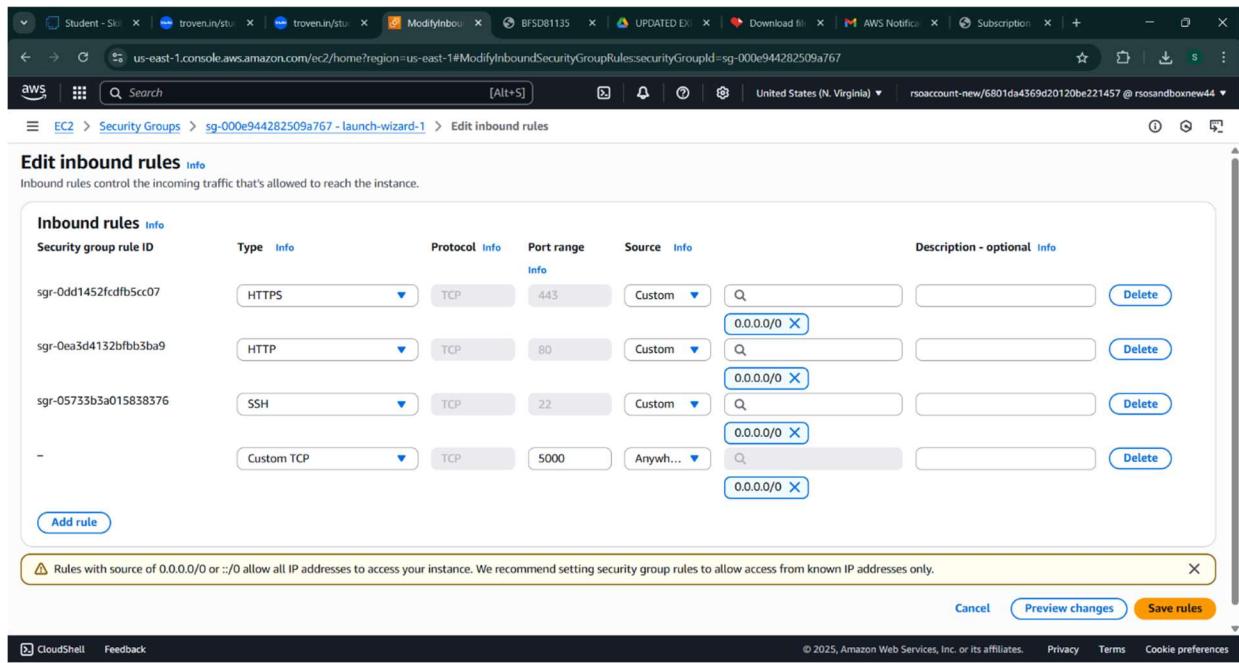
Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0dd1452fcdb5cc07	HTTPS	TCP	443	Custom	0.0.0.0/0
sgr-0ea3d4132bfb3ba9	HTTP	TCP	80	Custom	0.0.0.0/0
sgr-05733b3a015838376	SSH	TCP	22	Custom	0.0.0.0/0

**Add rule**

⚠ Rules with source of 0.0.0.0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Cancel Preview changes **Save rules**

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences



**Edit inbound rules** Info

Inbound rules control the incoming traffic that's allowed to reach the instance.

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0dd1452fcdb5cc07	HTTPS	TCP	443	Custom	0.0.0.0/0
sgr-0ea3d4132bfb3ba9	HTTP	TCP	80	Custom	0.0.0.0/0
sgr-05733b3a015838376	SSH	TCP	22	Custom	0.0.0.0/0
-	Custom TCP	TCP	5000	Anywh...	0.0.0.0/0

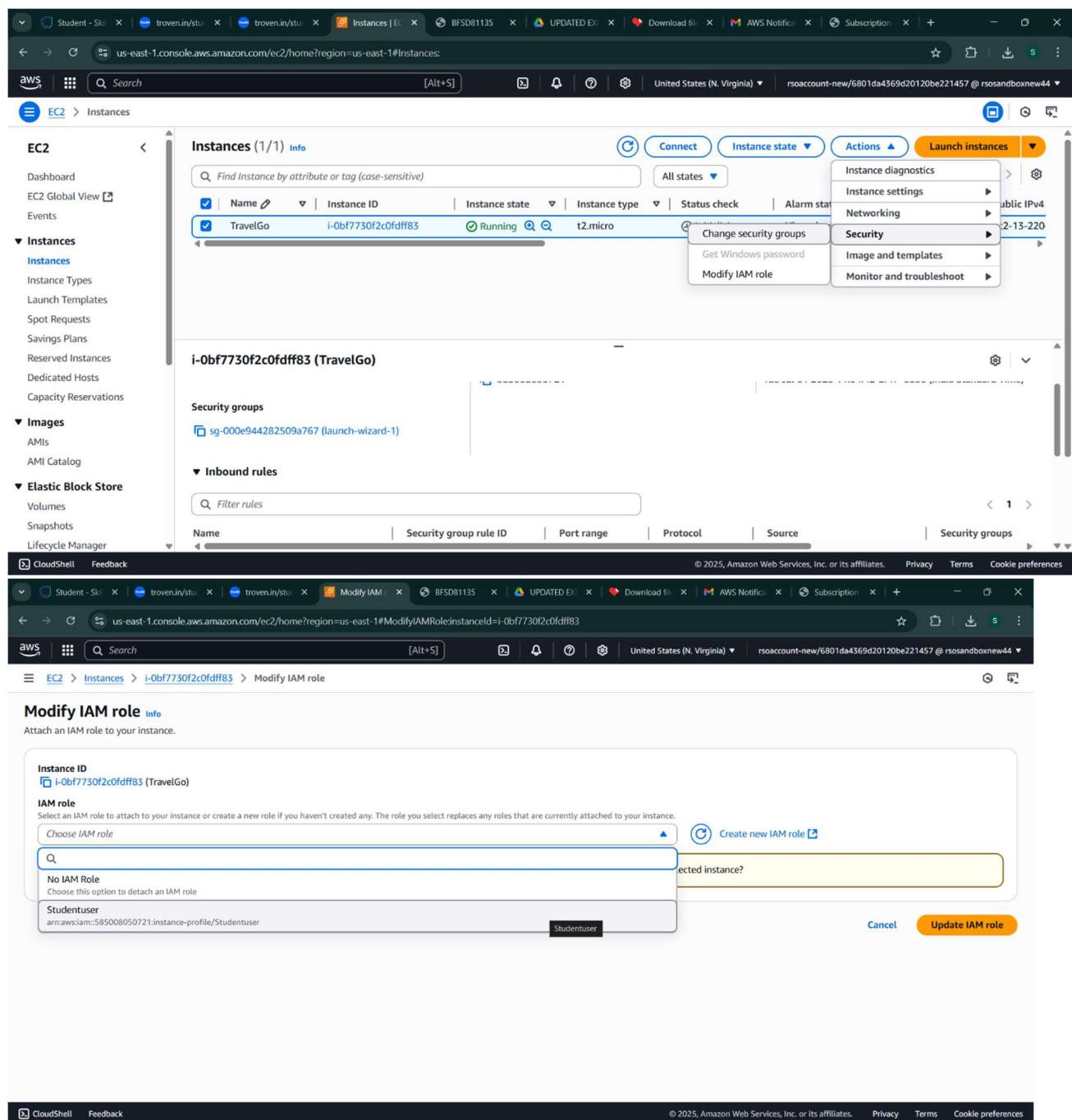
**Add rule**

⚠ Rules with source of 0.0.0.0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Cancel Preview changes **Save rules**

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the IAM role is connected, navigate to the **EC2** section in the **AWS Management Console**. Select the **EC2 instance** you wish to connect to. At the top of the **EC2 Dashboard**, click the **Connect** button. From the connection methods presented, choose **EC2 Instance Connect**. Finally, click **Connect** again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.

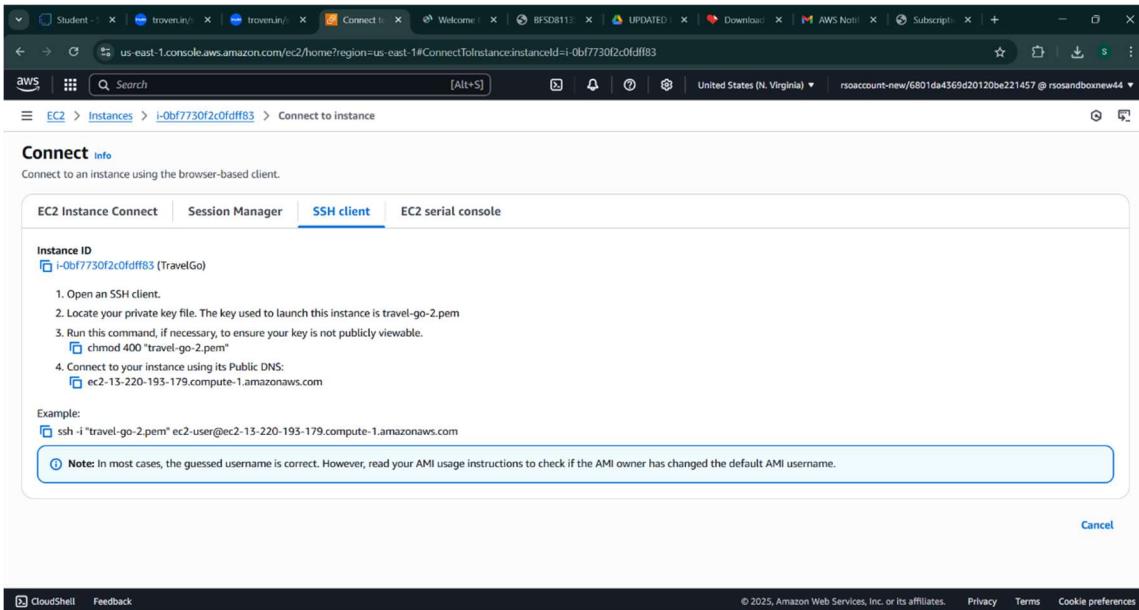


The screenshot shows two separate browser tabs of the AWS Management Console.

The top tab displays the **Instances** page under the **EC2** service. It lists one instance named **TravelGo** (i-0bf7730f2c0fdff83) which is **Running** and of type **t2.micro**. The **Actions** menu is open, showing options like **Instance diagnostics**, **Instance settings**, **Networking**, **Security** (which is currently selected), **Get Windows password**, **Image and templates**, and **Monitor and troubleshoot**.

The bottom tab displays the **Modify IAM role** dialog for the **TravelGo** instance. It shows the **Instance ID** as **i-0bf7730f2c0fdff83 (TravelGo)**. Under the **IAM role** section, there is a dropdown menu labeled **Choose IAM role** with an option **No IAM Role** and a note about choosing an option to detach an IAM role. A second dropdown menu shows the **Studentuser** role. To the right of the dropdowns, there is a note about selecting an instance and a **Create new IAM role** button. At the bottom right of the dialog are **Cancel** and **Update IAM role** buttons.

- Now connect the EC2 with the files



The screenshot shows the AWS EC2 Connect interface. At the top, it displays the URL [us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#ConnectToInstance:instanceId=i-0bf7730f2c0fdff83](https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#ConnectToInstance:instanceId=i-0bf7730f2c0fdff83). Below this, the breadcrumb navigation shows **EC2 > Instances > i-0bf7730f2c0fdff83 > Connect to instance**.

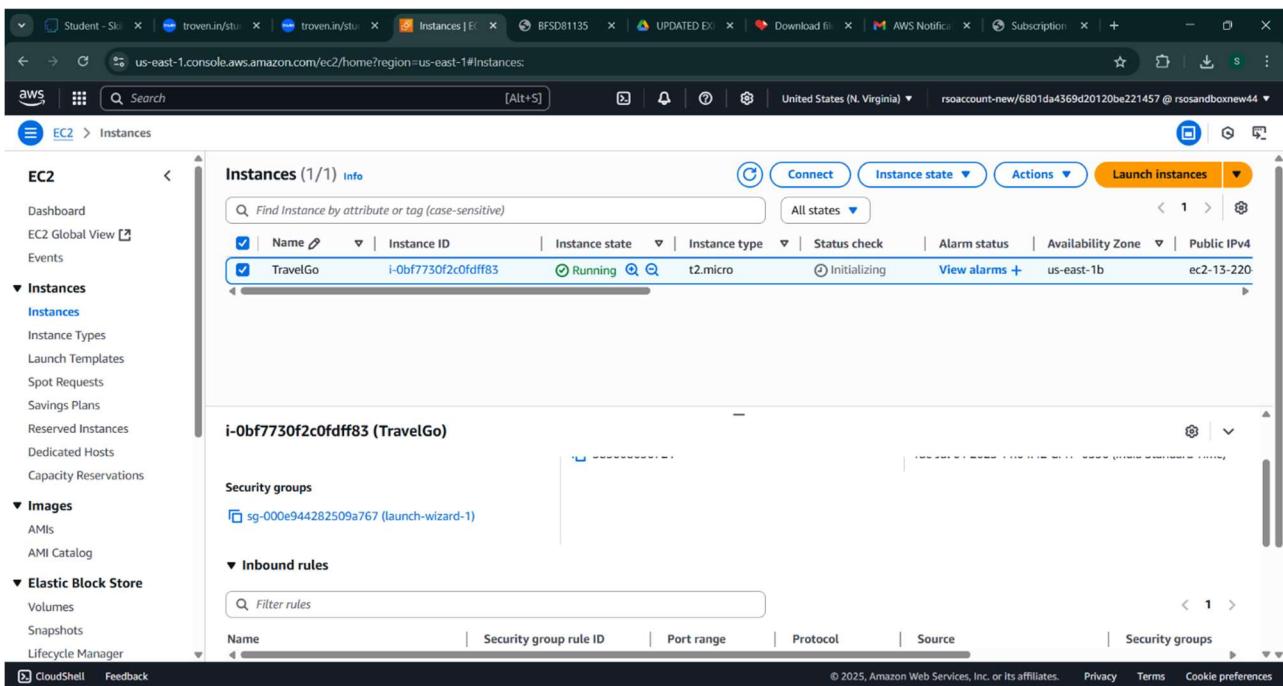
The main content area is titled **Connect Info**, which says "Connect to an instance using the browser-based client." It includes tabs for **EC2 Instance Connect**, **Session Manager**, and **SSH client** (which is selected). The **EC2 serial console** tab is also present.

Under the **SSH client** tab, the **Instance ID** is listed as **i-0bf7730f2c0fdff83 (TravelGo)**. Below this, there are four numbered steps:

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is `travel-go-2.pem`.
3. Run this command, if necessary, to ensure your key is not publicly viewable.  
`chmod 400 "travel-go-2.pem"`
4. Connect to your instance using its Public DNS:  
`ec2-13-220-193-179.compute-1.amazonaws.com`

Below these steps, there is an **Example:** section with a link to `ssh -i "travel-go-2.pem" ec2-user@ec2-13-220-193-179.compute-1.amazonaws.com`. A note at the bottom states: **Note:** In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

At the bottom right of the interface is a **Cancel** button.



The screenshot shows the AWS EC2 Instances page. The left sidebar menu is expanded, showing sections like **Dashboard**, **EC2 Global View**, **Events**, **Instances** (which is selected), **Images**, and **Elastic Block Store**. The main content area is titled **Instances (1/1) Info**.

Find Instance by attribute or tag (case-sensitive)	All states
<input checked="" type="checkbox"/> Name	Instance ID
<input checked="" type="checkbox"/> TravelGo	i-0bf7730f2c0fdff83
	Instance state
	Running
	Instance type
	t2.micro
	Status check
	Initializing
	Alarm status
	View alarms
	Availability Zone
	us-east-1b
	Public IPv4
	ec2-13-220

Below the table, the instance details for **i-0bf7730f2c0fdff83 (TravelGo)** are shown. It lists the **Security groups** (`sg-000e944282509a767 (launch-wizard-1)`) and **Inbound rules**. A filter bar for rules is present, along with columns for **Name**, **Security group rule ID**, **Port range**, **Protocol**, **Source**, and **Security groups**.

## Milestone 7: Deployment on EC2

### Activity 7.1: Install Software on the EC2 Instance

Install Python3, Flask, and Git:

On Amazon Linux 2:

```
sudo yum update -y
sudo yum install python3 git
sudo pip3 install flask boto3
```

Verify Installations:

```
flask --version
git --version
```

### Activity 7.2: Clone Your Flask Project from GitHub

**Clone your project repository from GitHub into the EC2 instance using Git.**

Run: 'git clone <https://github.com/vour-github-username/vour-repository-name.git>'

Note: change your-github-username and your-repository-name with your credentials

here: 'git clone https://github.com/AlekhyaPenubakula/InstantLibrary.git'

- This will download your project to the EC2 instance.

**To navigate to the project directory, run the following command:**

```
cd InstantLibrary
```

**Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:**

**Run the Flask Application**

```
sudo flask run --host=0.0.0.0 --port=80
```

```

ec2-user@ip-172-31-84-183:~ % 
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\VIPPA> ssh -i "C:\Users\VIPPA\Downloads\travel-go-app.pem" ec2-user@ec2-13-218-73-47.compute-1.amazonaws.com

      _#_
     ~\_\_ #####_      Amazon Linux 2023
     ~~ \_######\_
     ~~ \|###|
     ~~ \#/  ___  https://aws.amazon.com/linux/amazon-linux-2023
     ~~   V~' '-->
     ~~~   /
     ~~_.-_-/
     ~/--/
     _/`-/
     _/m'` 

Last login: Mon Jun 30 11:06:19 2025 from 223.185.81.172
[ec2-user@ip-172-31-84-183 ~]$ sudo yum install git -y
Last metadata expiration check: 1:17:53 ago on Mon Jun 30 11:07:05 2025.
Package git-2.47.1-1.amzn2023.0.3.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-84-183 ~]$ git clone https://github.com/veera5423/TravelGo1.git
fatal: destination path 'TravelGo1' already exists and is not an empty directory.
[ec2-user@ip-172-31-84-183 ~]$ cd TravelGo1
[ec2-user@ip-172-31-84-183 TravelGo1]$ sudo yum install python3 -y
sudo yum install python3-pip -y

```

## Verify the Flask app is running:

<http://your-ec2-public-ip>

- Run the Flask app on the EC2 instance

```

ec2-user@ip-172-31-84-183:~ % 
.38.46->boto3) (2.8.1)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.39.0,>=1.38.46->boto3) (1.25.10)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.39.0,>=1.38.46->boto3) (1.15.0)
[ec2-user@ip-172-31-84-183 TravelGo1]$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.84.183:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 594-419-608
223.185.81.172 -- [30/Jun/2025 12:27:17] "GET / HTTP/1.1" 200 -
223.185.81.172 -- [30/Jun/2025 12:27:17] "GET /static/images/flight-icon.jpeg HTTP/1.1" 304 -
223.185.81.172 -- [30/Jun/2025 12:27:17] "GET /static/images/train-icon.jpeg HTTP/1.1" 304 -
223.185.81.172 -- [30/Jun/2025 12:27:18] "GET /static/images/bus-icon.jpeg HTTP/1.1" 304 -
223.185.81.172 -- [30/Jun/2025 12:27:18] "GET /static/images/hotel.jpg HTTP/1.1" 304 -
223.185.81.172 -- [30/Jun/2025 12:27:53] "GET /login HTTP/1.1" 200 -
223.185.81.172 -- [30/Jun/2025 12:27:55] "GET /register HTTP/1.1" 200 -
223.185.81.172 -- [30/Jun/2025 12:28:13] "POST /register HTTP/1.1" 200 -
223.185.81.172 -- [30/Jun/2025 12:28:21] "GET /login HTTP/1.1" 200 -
223.185.81.172 -- [30/Jun/2025 12:28:29] "POST /login HTTP/1.1" 302 -
223.185.81.172 -- [30/Jun/2025 12:28:29] "GET /dashboard HTTP/1.1" 200 -
223.185.81.172 -- [30/Jun/2025 12:28:39] "POST /cancel_booking HTTP/1.1" 302 -
223.185.81.172 -- [30/Jun/2025 12:28:39] "GET /dashboard HTTP/1.1" 200 -
223.185.81.172 -- [30/Jun/2025 12:28:46] "GET /bus HTTP/1.1" 200 -

```

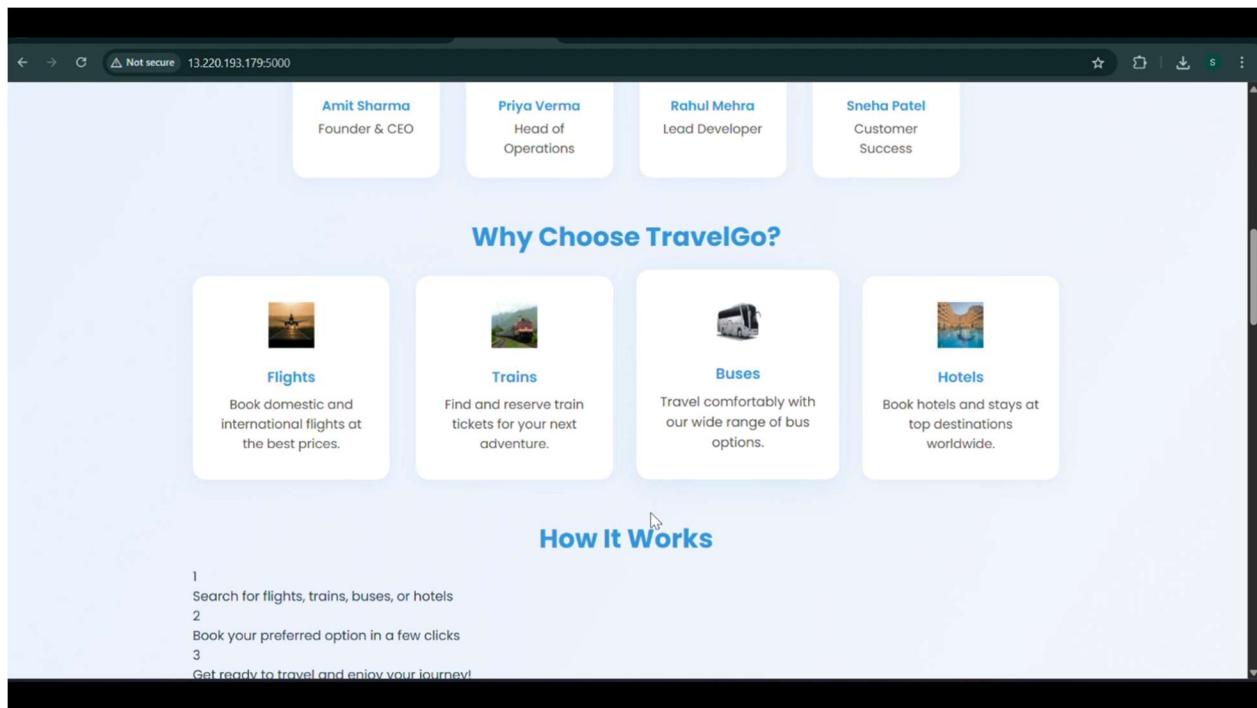
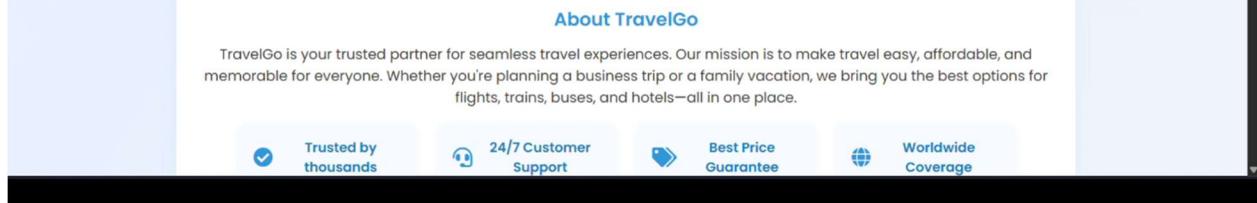
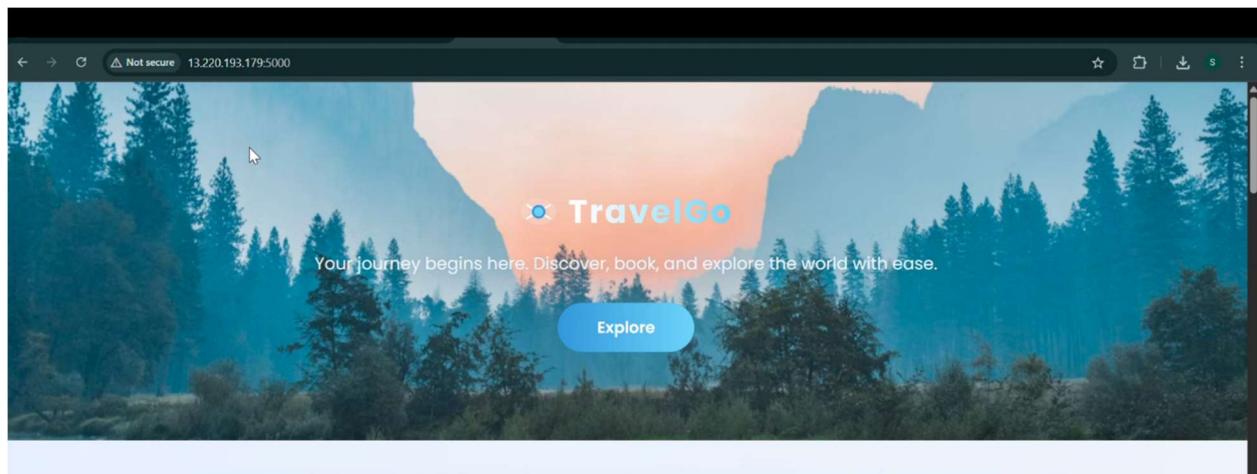
Access the website through:

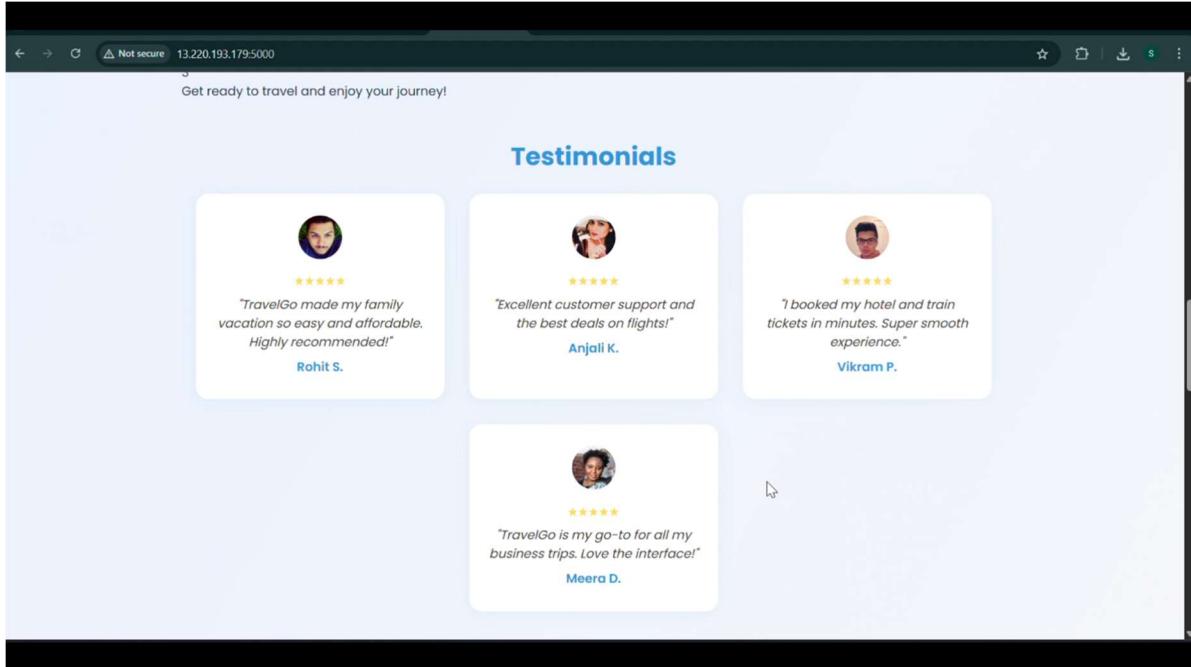
Public IPs: [http://13.220.193.179:5000//](http://13.220.193.179:5000/)

## Milestone 8: Testing and Deployment

- **Activity 8.1: Conduct functional testing to verify user registration, login, book requests, and notifications.**

Welcome Page:





Not secure 13.220.193.179:5000

Get ready to travel and enjoy your journey!

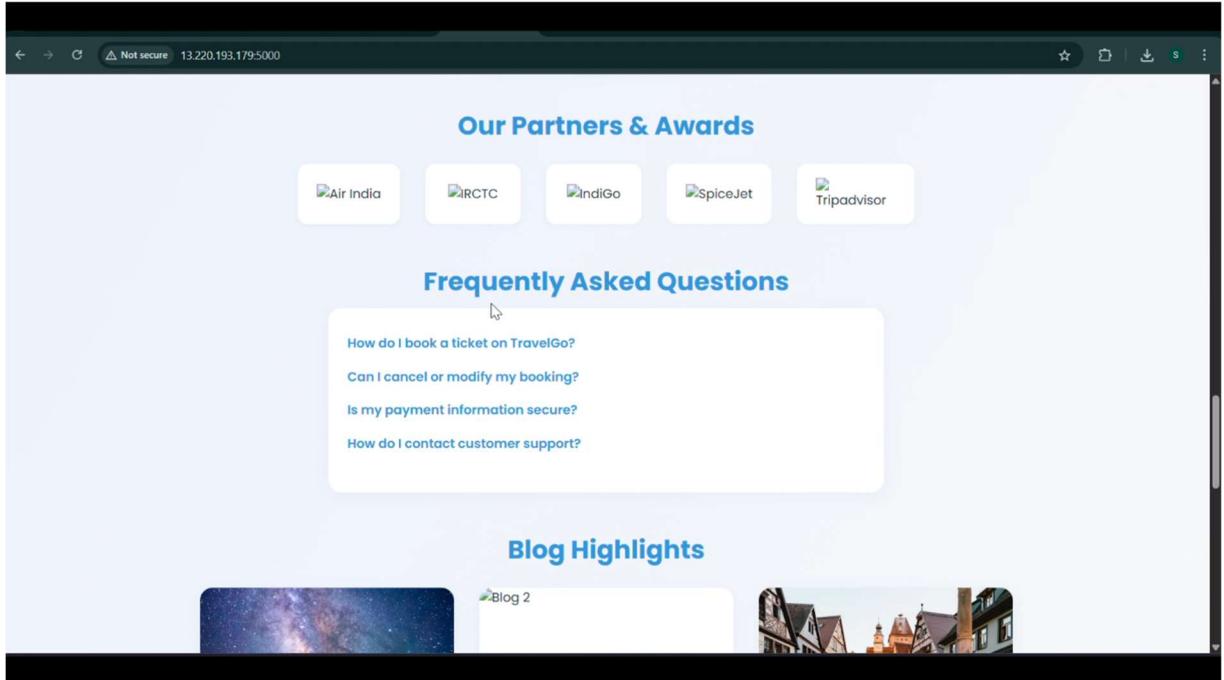
## Testimonials

 ★★★★  
"TravelGo made my family vacation so easy and affordable. Highly recommended!"  
Rohit S.

 ★★★★  
"Excellent customer support and the best deals on flights!"  
Anjali K.

 ★★★★  
"I booked my hotel and train tickets in minutes. Super smooth experience."  
Vikram P.

 ★★★★  
"TravelGo is my go-to for all my business trips. Love the interface!"  
Meera D.



Not secure 13.220.193.179:5000

## Our Partners & Awards



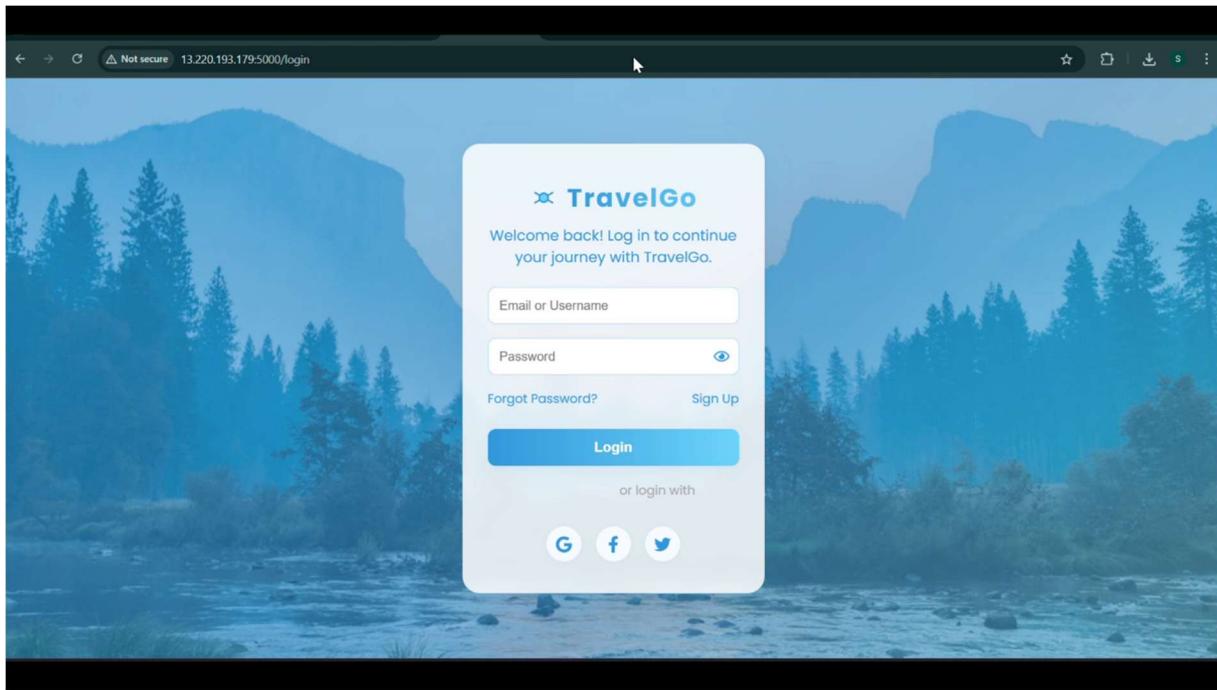
## Frequently Asked Questions

How do I book a ticket on TravelGo?  
Can I cancel or modify my booking?  
Is my payment information secure?  
How do I contact customer support?

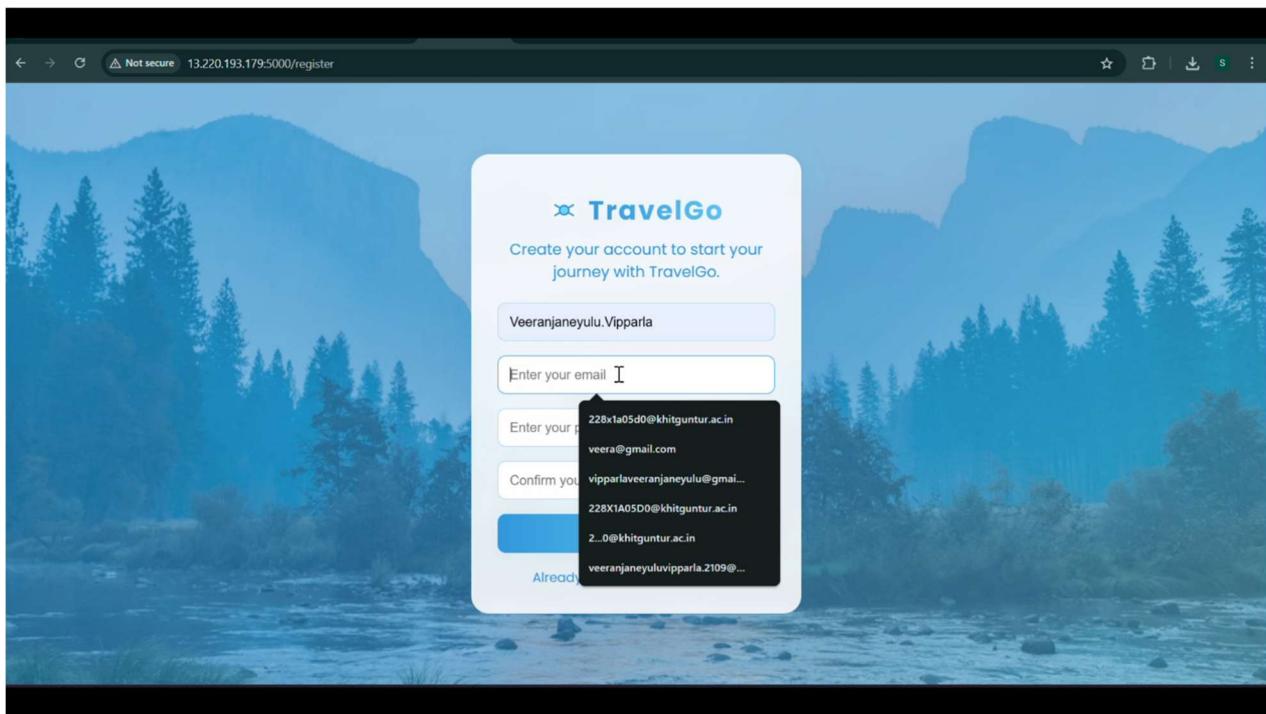
## Blog Highlights

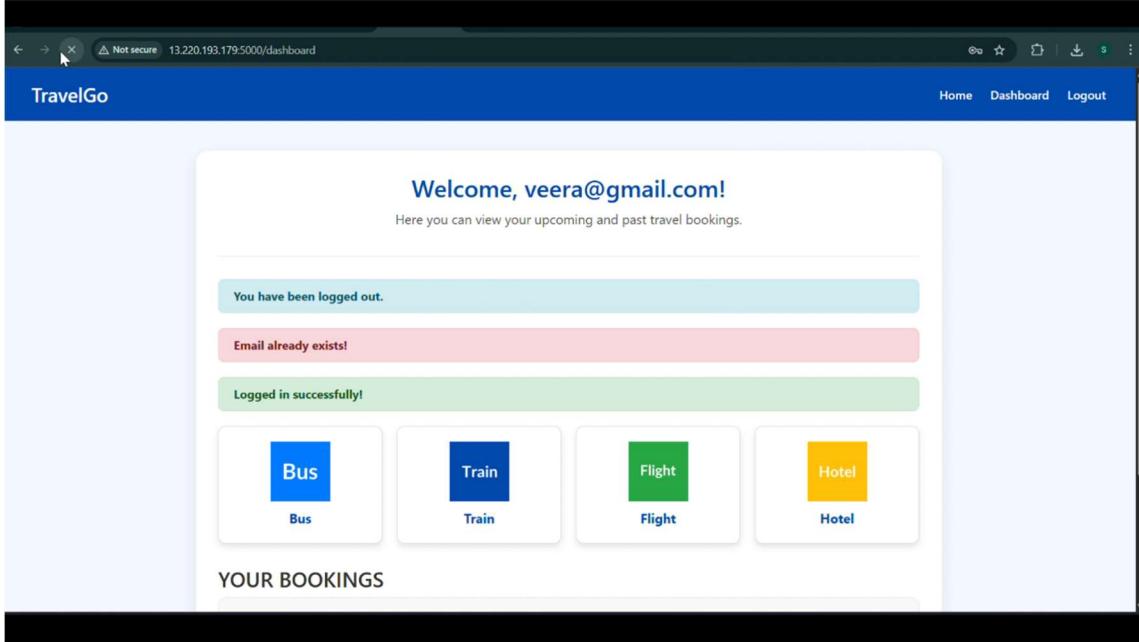


### Login Page:

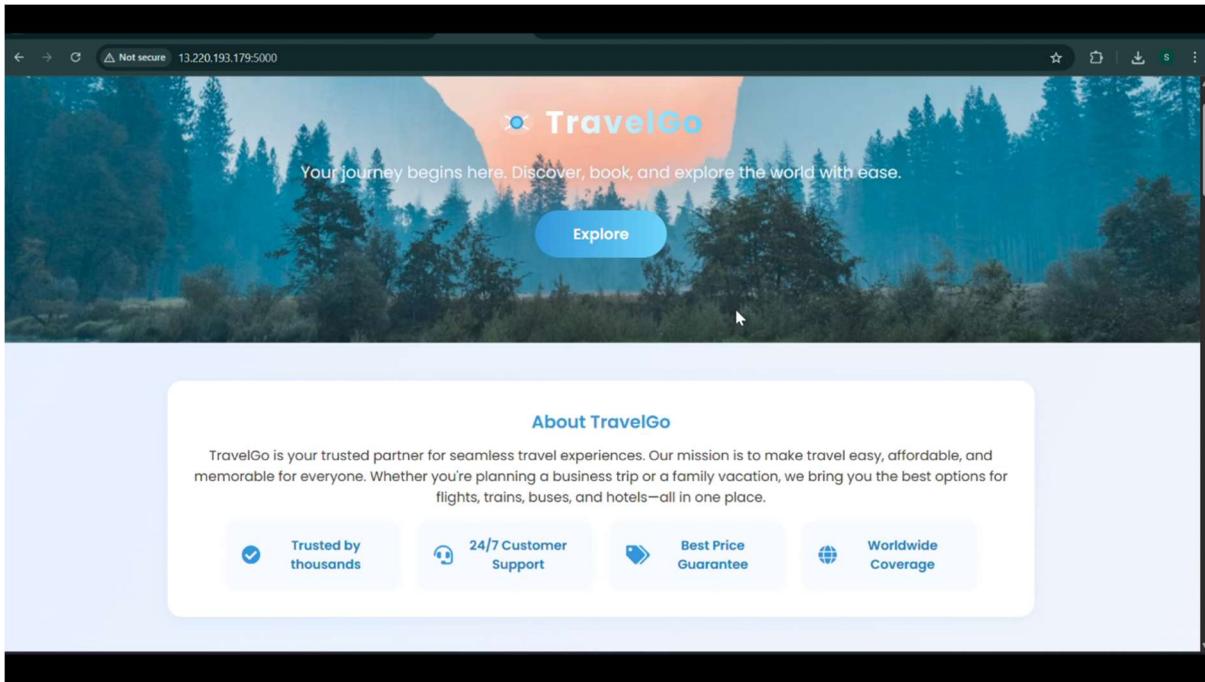


### Register Page:



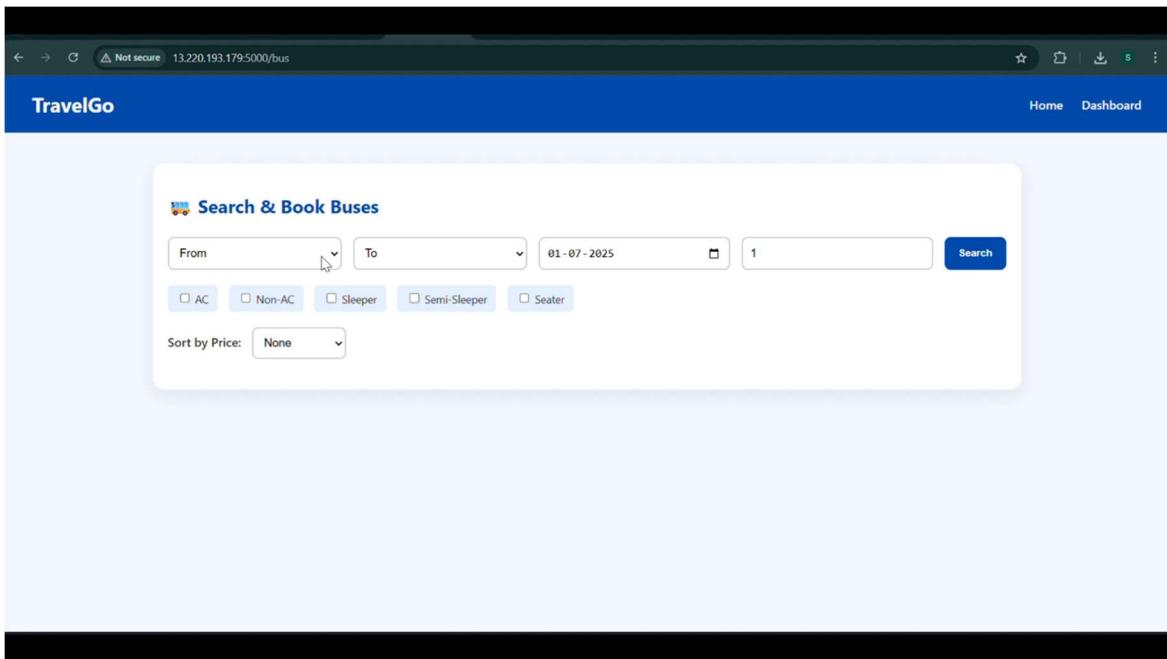
**Dashboard page:**

The screenshot shows a web browser window for 'TravelGo' at the URL 13.220.193.179:5000/dashboard. The title bar says 'TravelGo'. The top navigation bar includes 'Home', 'Dashboard', and 'Logout'. A central message box displays 'Welcome, veera@gmail.com!' followed by 'Here you can view your upcoming and past travel bookings.' Below this, three error messages are shown in colored boxes: a green box for 'You have been logged out.', a red box for 'Email already exists!', and a green box for 'Logged in successfully!'. Below the message box are four service icons: 'Bus' (blue), 'Train' (dark blue), 'Flight' (green), and 'Hotel' (yellow). At the bottom, a section titled 'YOUR BOOKINGS' is partially visible.

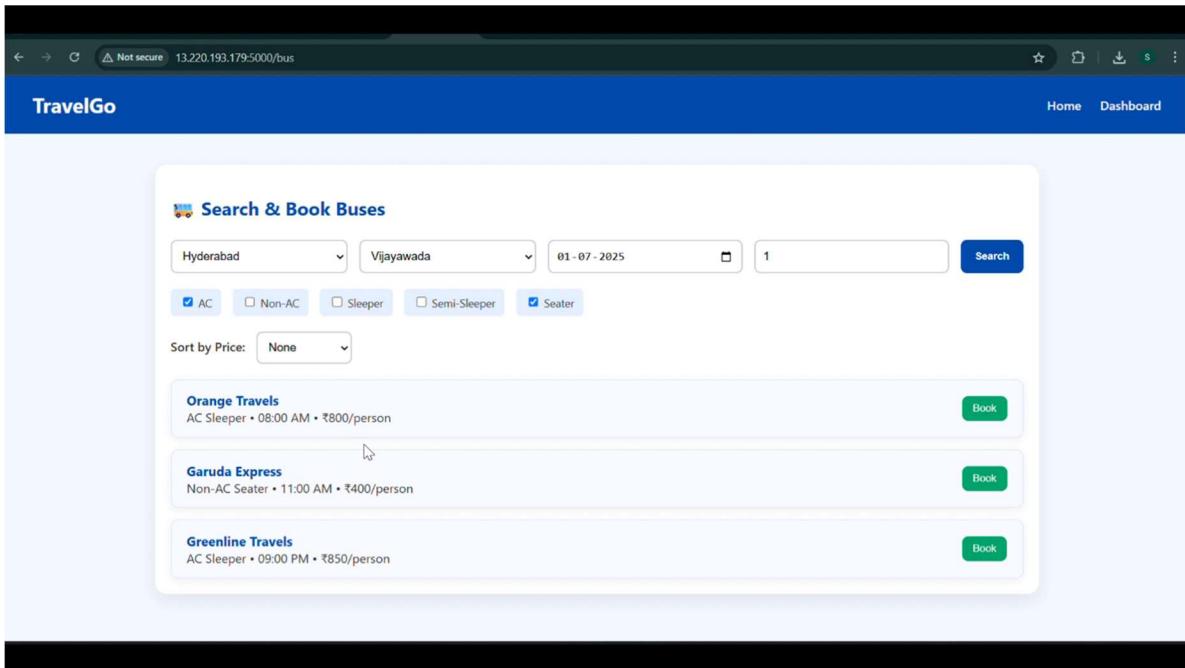
**About Us page:**

The screenshot shows a web browser window for 'TravelGo' at the URL 13.220.193.179:5000. The title bar says 'TravelGo'. The main header features a scenic forest background with the text 'Your journey begins here. Discover, book, and explore the world with ease.' and a large blue 'Explore' button. Below this is a white callout box titled 'About TravelGo'. It contains the text: 'TravelGo is your trusted partner for seamless travel experiences. Our mission is to make travel easy, affordable, and memorable for everyone. Whether you're planning a business trip or a family vacation, we bring you the best options for flights, trains, buses, and hotels—all in one place.' At the bottom of the callout box are five service highlights: 'Trusted by thousands' (checkmark icon), '24/7 Customer Support' (ear icon), 'Best Price Guarantee' (key icon), and 'Worldwide Coverage' (globe icon).

## Bus Booking Page:

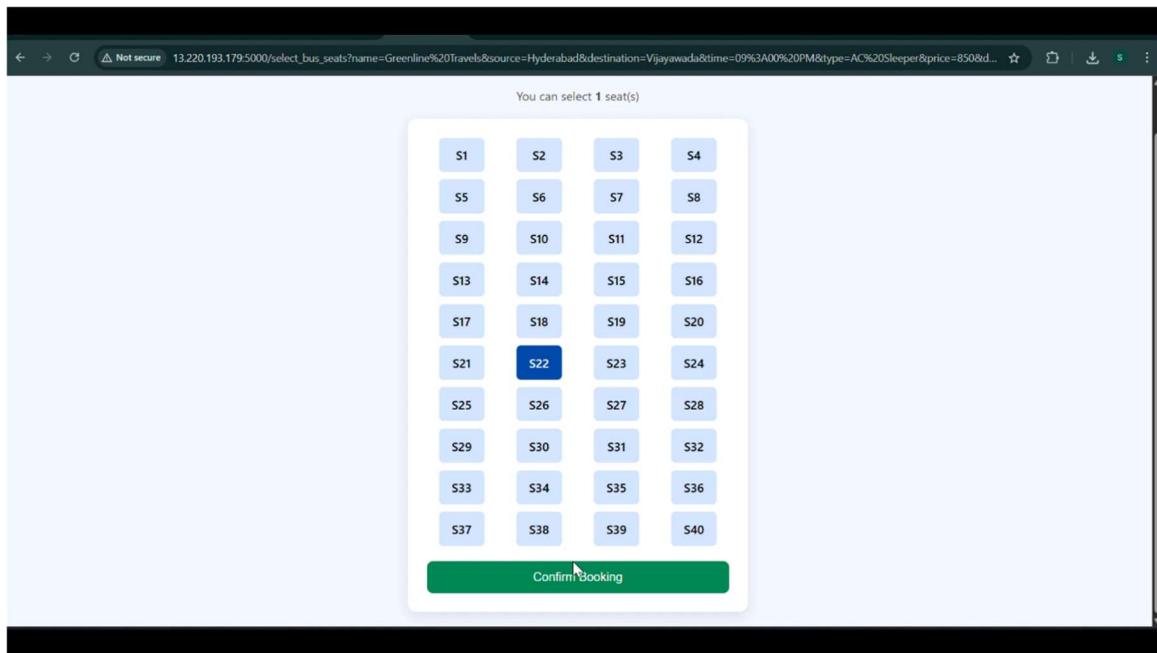


The screenshot shows the initial search interface for TravelGo. It includes fields for 'From' (dropdown), 'To' (dropdown), date ('01-07-2025'), passengers ('1'), a 'Search' button, and filters for AC, Non-AC, Sleeper, Semi-Sleeper, and Seater. A dropdown for 'Sort by Price' is set to 'None'.

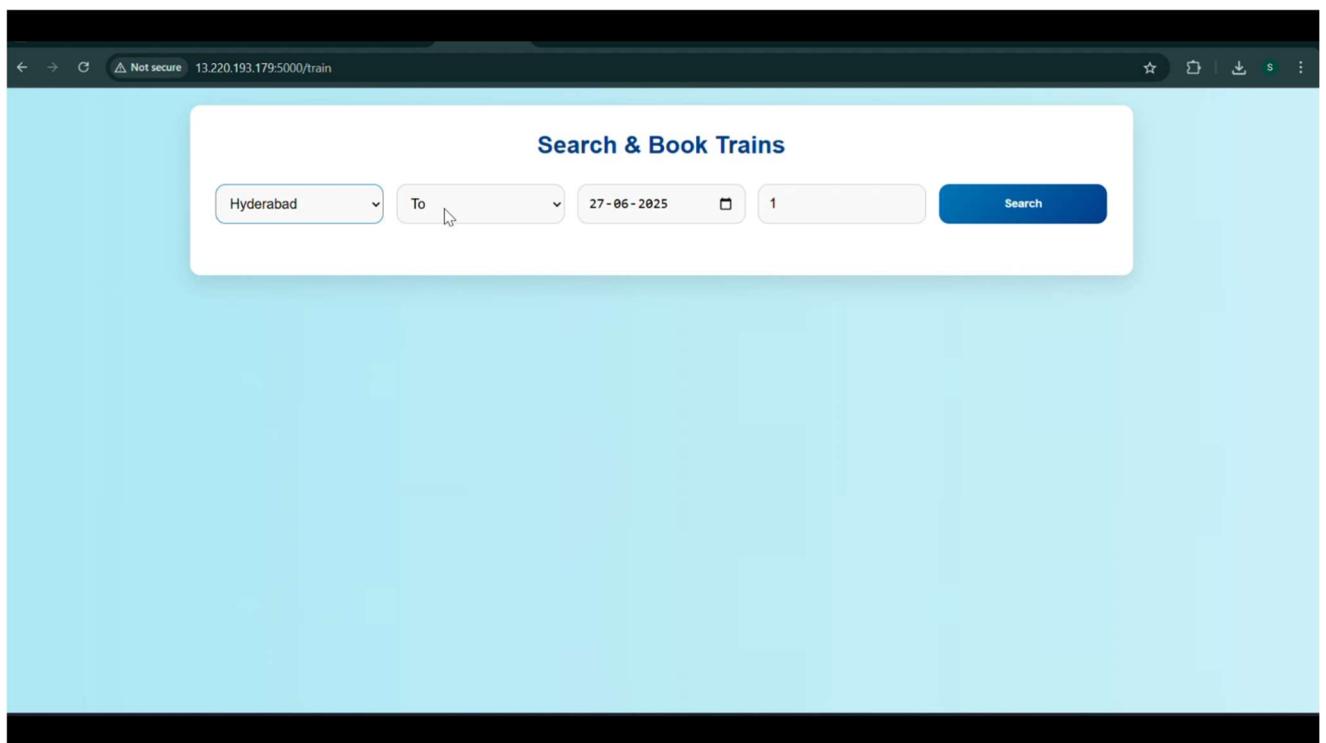


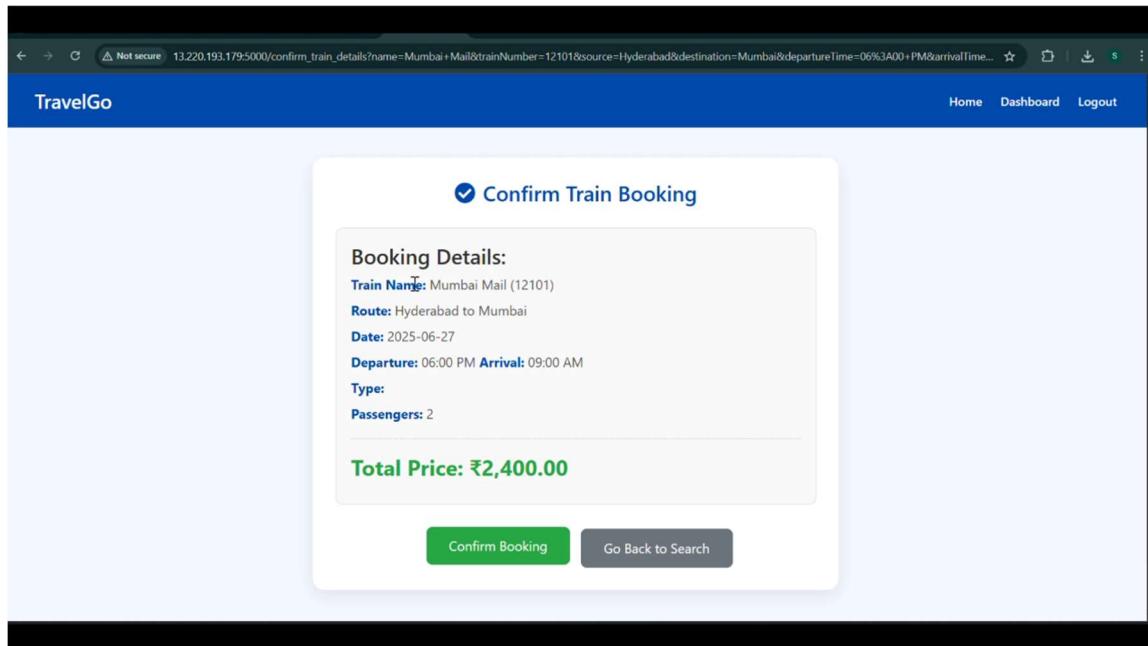
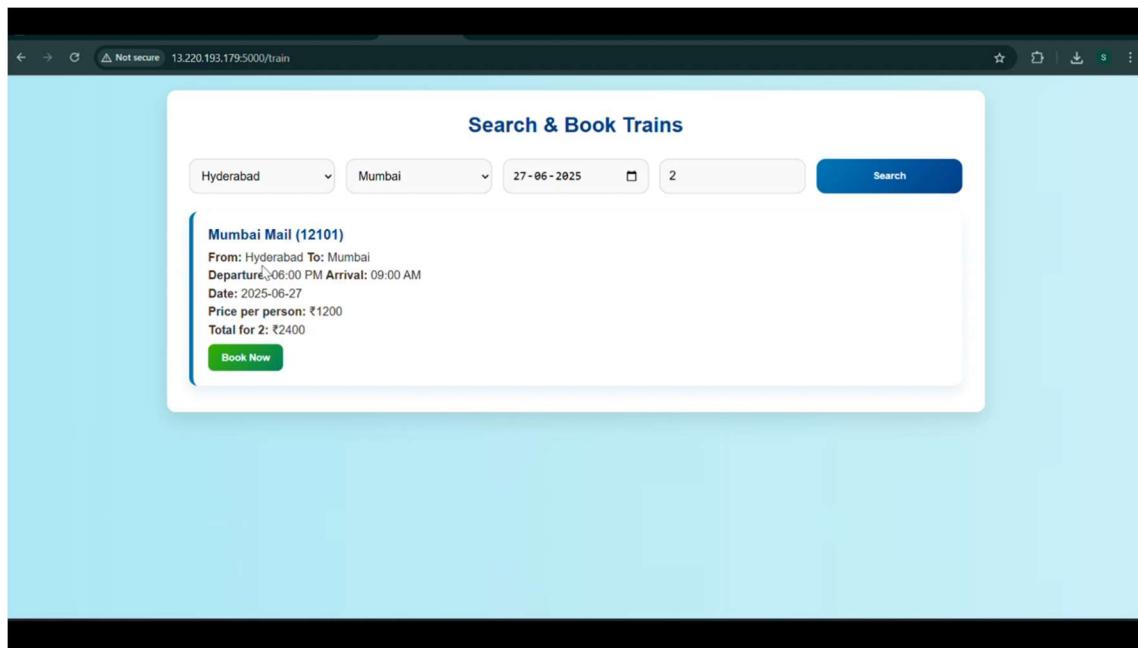
The screenshot shows the search results for a bus journey from Hyderabad to Vijayawada on 01-07-2025 for 1 passenger. The search filters include AC, Non-AC, Sleeper, Semi-Sleeper, and Seater selected. The results list three operators: Orange Travels, Garuda Express, and Greenline Travels, each with their respective bus details and a 'Book' button.

Operator	Bus Type	Departure Time	Fare	Action
Orange Travels	AC Sleeper	08:00 AM	₹800/person	Book
Garuda Express	Non-AC Seater	11:00 AM	₹400/person	Book
Greenline Travels	AC Sleeper	09:00 PM	₹850/person	Book

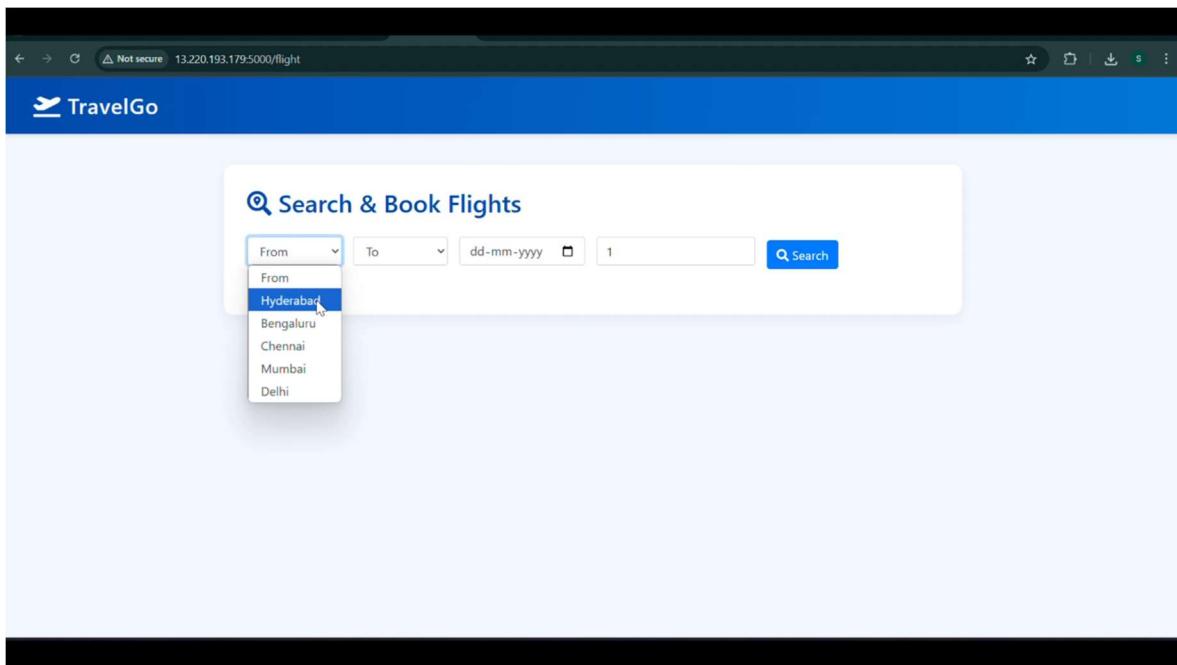


**Train Booking page:**

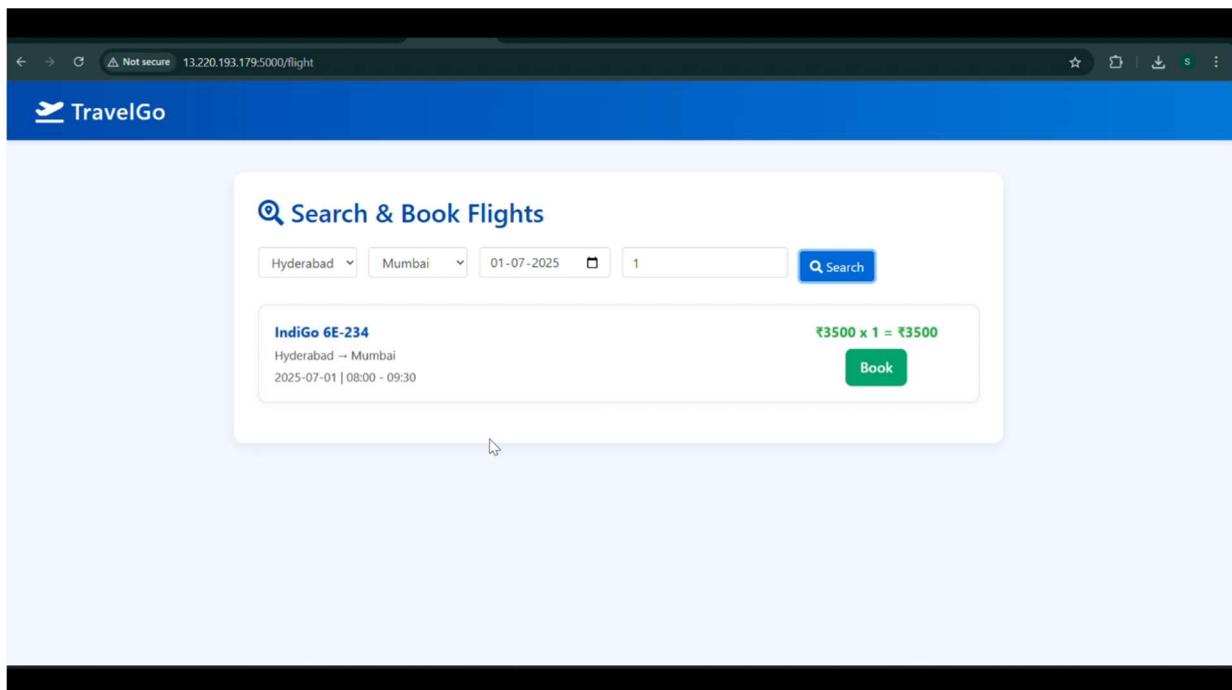




## Flight Booking Page:



The screenshot shows a web browser window for 'TravelGo'. The URL bar indicates the page is not secure (Not secure) and shows the address 13.220.193.179:5000/flight. The main content area is titled 'Search & Book Flights'. A dropdown menu is open under the 'From' field, listing several Indian cities: Hyderabad, Bengaluru, Chennai, Mumbai, and Delhi. The 'Hyderabad' option is currently selected.



The screenshot shows the same 'TravelGo' search interface after a search has been performed. The 'From' field is set to 'Hyderabad' and the 'To' field is set to 'Mumbai'. The travel date is '01-07-2025'. The search button has been clicked, and the results are displayed below. One flight result is shown: 'IndiGo 6E-234' from 'Hyderabad → Mumbai' on '2025-07-01' at '08:00 - 09:30'. The price for one ticket is '₹3500 x 1 = ₹3500'. A green 'Book' button is visible next to the price.

Not secure 13.220.193.179:5000/dashboard

Flight booking confirmed successfully!

Bus
Train
Flight
Hotel

**YOUR BOOKINGS**

**Flight Booking**

Flight: IndiGo 6E-234  
 Route: Hyderabad → Mumbai  
 Date: 2025-07-01  
 Departure: 08:00  
 Arrival: 09:30  
 Passengers: 1  
 Booked On: 2025-07-01

**₹3,500.00**

**Cancel Booking**

**Train Booking**

Train: Mumbai Mail (12101)  
 Route: Hyderabad → Mumbai  
 Date: 2025-06-27  
 Time: 06:00 PM - 09:00 AM  
 Passengers: 2  
 Seats: S71, S94

**₹2,400.00**

**Cancel Booking**

13.220.193.179:5000/hotel

### Hotel Booking Page:

Not secure 13.220.193.179:5000/hotel

**TravelGo**

Home Dashboard Hotels

**Book Your Perfect Stay**

Select City

- Hyderabad
- Mumbai
- Delhi
- Bangalore

01-07-2025

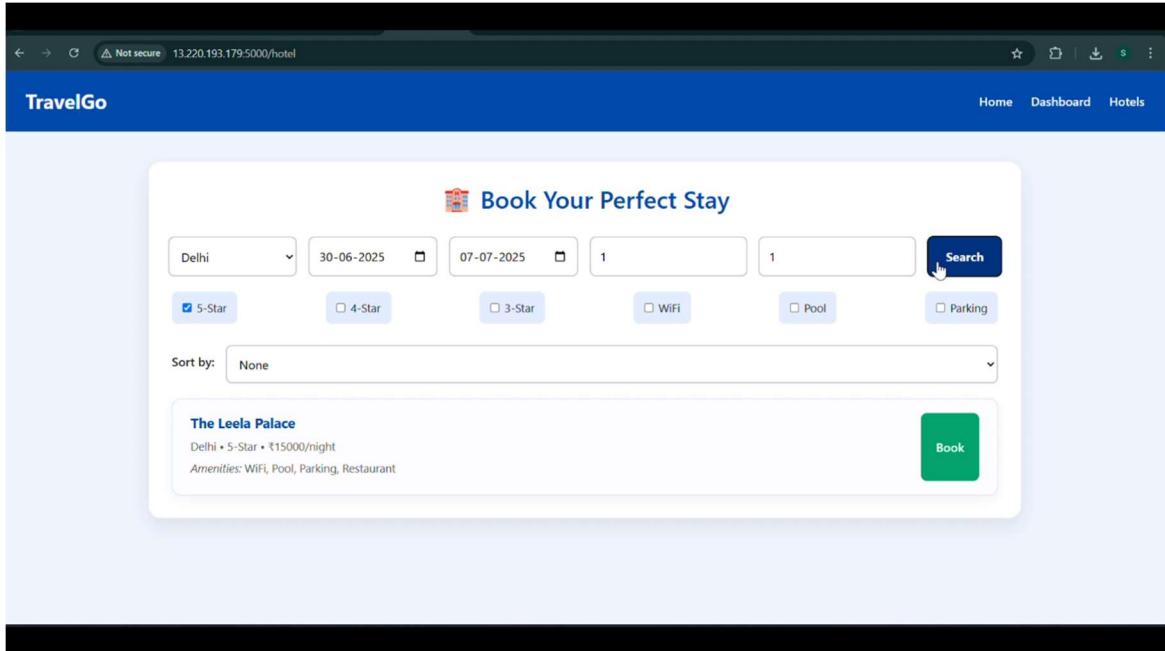
02-07-2025

1

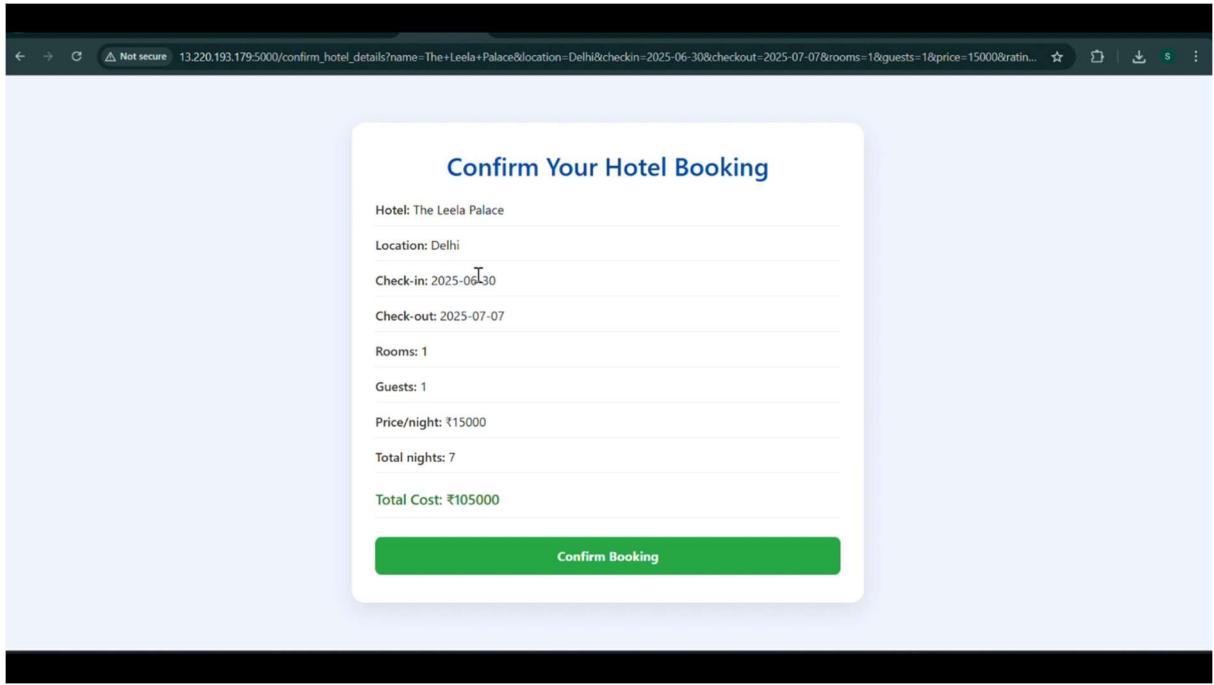
1

**Search**

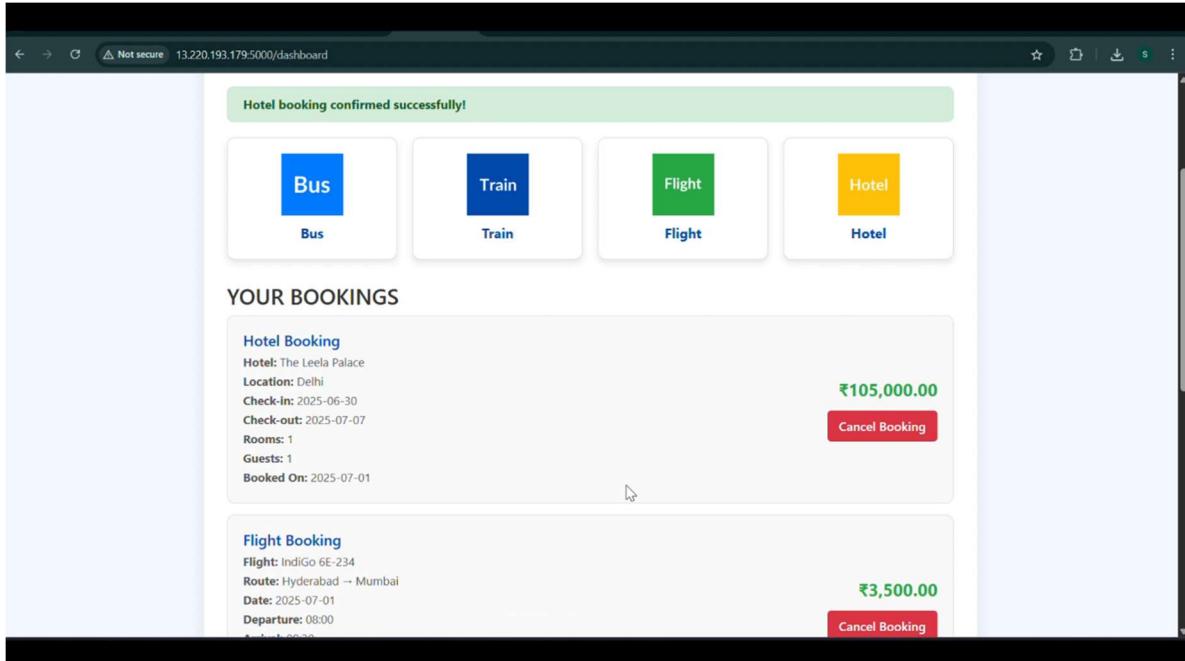
4-Star
 3-Star
 WiFi
 Pool
 Parking



The screenshot shows a web browser window for 'TravelGo' with a blue header bar. The URL bar indicates the page is not secure (13.220.193.179:5000/hotel). The main content area has a white background and features a search form titled 'Book Your Perfect Stay'. The form includes fields for destination ('Delhi'), check-in date ('30-06-2025'), check-out date ('07-07-2025'), number of rooms ('1'), and number of guests ('1'). A large blue 'Search' button is positioned to the right of these fields. Below the search form are several filter buttons: '5-Star' (checked), '4-Star', '3-Star', 'WiFi', 'Pool', and 'Parking'. A dropdown menu labeled 'Sort by' with 'None' selected is also present. A result card for 'The Leela Palace' is displayed, showing it is a 5-star hotel in Delhi at ₹15000/night, with amenities WiFi, Pool, Parking, and Restaurant. A green 'Book' button is on the right side of the card.



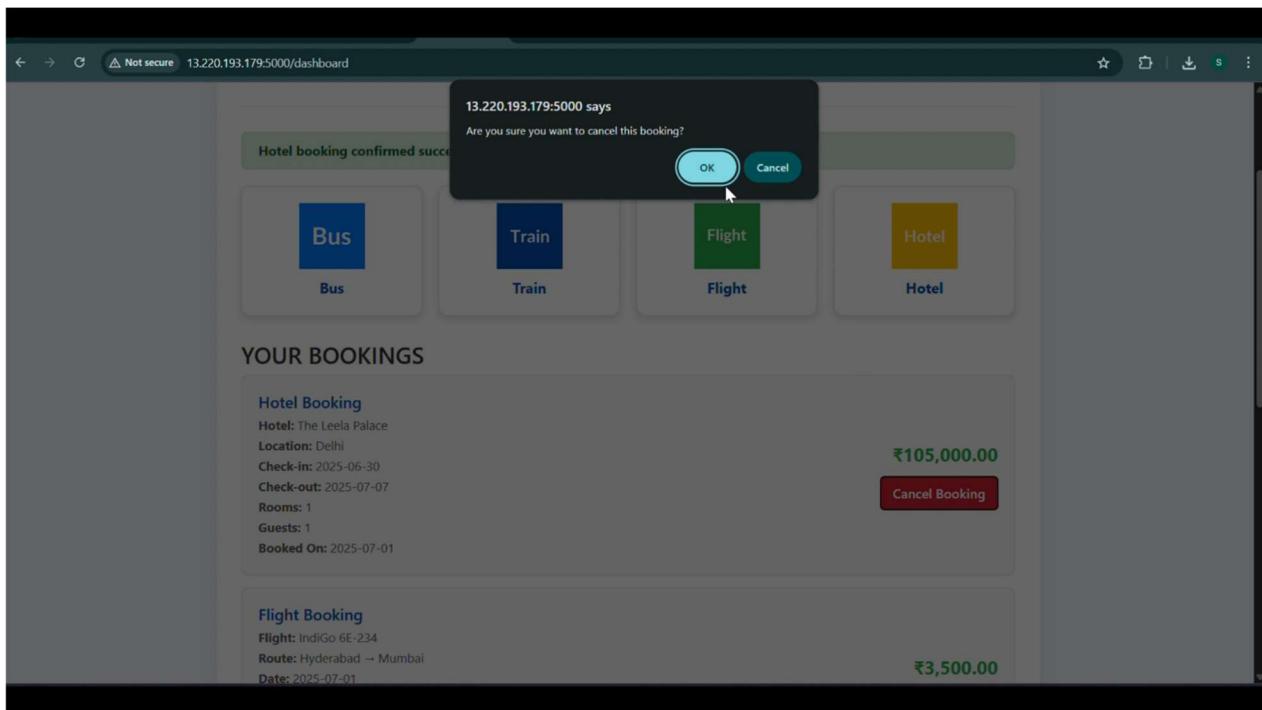
The screenshot shows a web browser window for 'TravelGo' with a blue header bar. The URL bar indicates the page is not secure (13.220.193.179:5000/confirm\_hotel\_details?name=The+Leela+Palace&location=Delhi&checkin=2025-06-30&checkout=2025-07-07&rooms=1&guests=1&price=15000&ratin...'). The main content area has a white background and features a confirmation message 'Confirm Your Hotel Booking'. It lists the following details:  
Hotel: The Leela Palace  
Location: Delhi  
Check-in: 2025-06-30  
Check-out: 2025-07-07  
Rooms: 1  
Guests: 1  
Price/night: ₹15000  
Total nights: 7  
Total Cost: ₹105000  
A large green 'Confirm Booking' button is at the bottom of the confirmation box.



The screenshot shows a web-based dashboard for booking management. At the top, there are four main categories: Bus, Train, Flight, and Hotel. Below these, a green banner displays the message "Hotel booking confirmed successfully!". The "YOUR BOOKINGS" section contains two entries:

- Hotel Booking**: Details include Hotel: The Leela Palace, Location: Delhi, Check-in: 2025-06-30, Check-out: 2025-07-07, Rooms: 1, Guests: 1, Booked On: 2025-07-01. The total amount is ₹105,000.00. A red "Cancel Booking" button is present.
- Flight Booking**: Details include Flight: IndiGo 6E-234, Route: Hyderabad → Mumbai, Date: 2025-07-01, Departure: 08:00, Arrival: 09:00. The total amount is ₹3,500.00. A red "Cancel Booking" button is present.

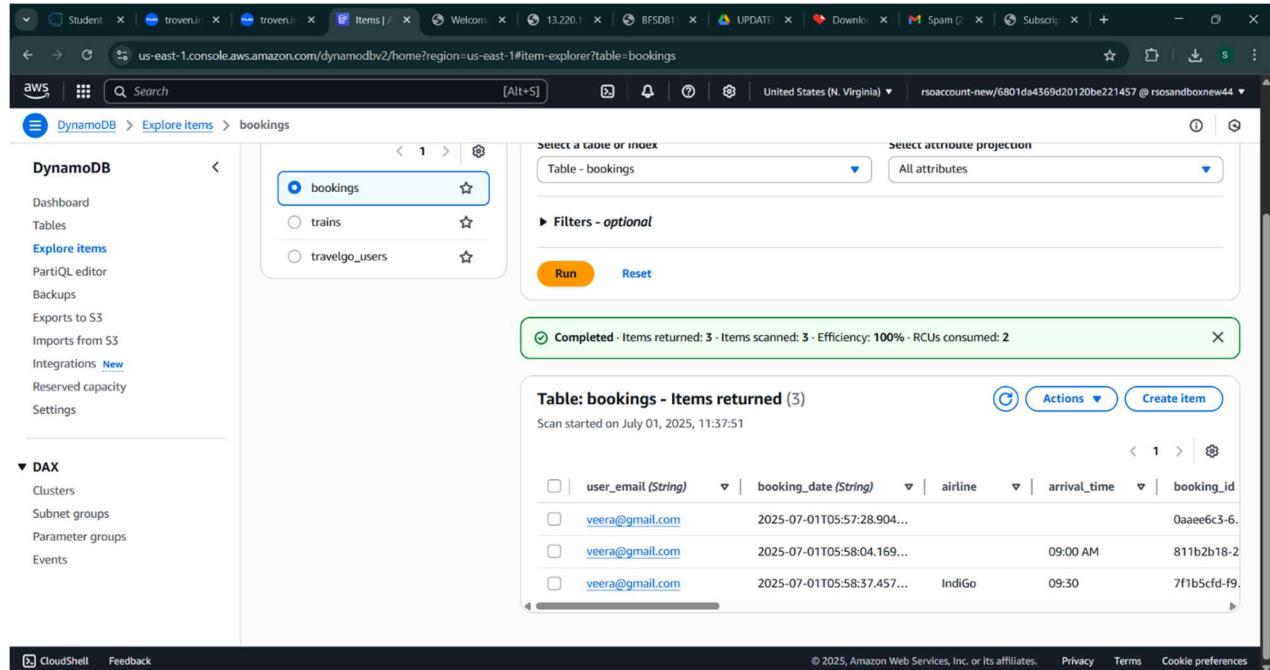
### Cancelation of Bookings:



A confirmation dialog box is overlaid on the dashboard, asking "Are you sure you want to cancel this booking?". It contains two buttons: "OK" (highlighted with a blue oval) and "Cancel". The background shows the same two bookings as the previous screenshot. The "OK" button is being clicked, which will likely trigger the cancellation process.

## Dynamodb Database updatons :

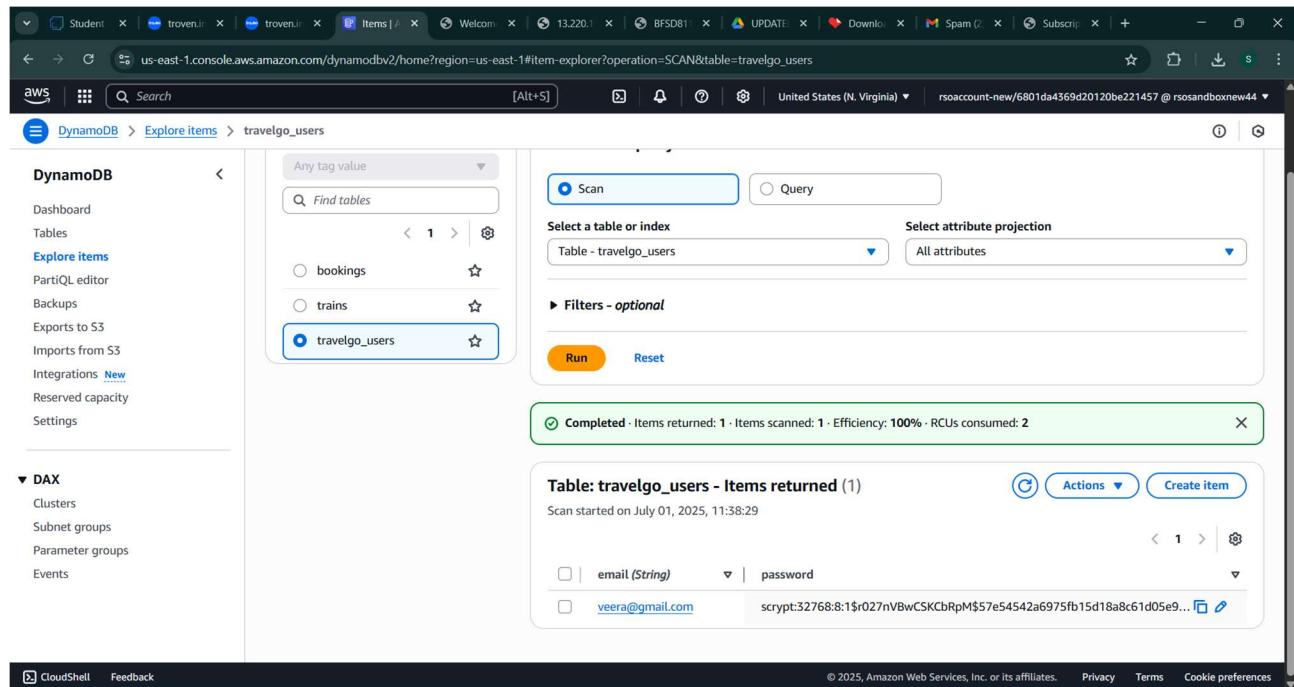
### 1. Bookings table :



The screenshot shows the AWS DynamoDB console with the 'bookings' table selected. The table contains the following data:

user_email (String)	booking_date (String)	airline	arrival_time	booking_id
veera@gmail.com	2025-07-01T05:57:28.904...			Oaaee6c3-6...
veera@gmail.com	2025-07-01T05:58:04.169...		09:00 AM	811b2b18-2...
veera@gmail.com	2025-07-01T05:58:37.457...	IndiGo	09:30	7f1b5cfdf9...

### 2. Users table :



The screenshot shows the AWS DynamoDB console with the 'travelgo\_users' table selected. The table contains the following data:

email (String)	password
veera@gmail.com	script:32768:8:1:r027nVBwCSKCbRpM\$57e54542a6975fb15d18a8c61d05e9...

## Conclusion:

The **TravelGo** Website has been successfully developed and deployed using a scalable and cloud-native architecture. Leveraging AWS services such as EC2 for hosting, DynamoDB for real-time data management, and SNS for instant booking and cancellation notifications, the platform provides a seamless travel booking experience for users. TravelGo enables registered users to search and book buses, trains, flights, and hotels in a centralized, intuitive interface, eliminating the complexities of navigating multiple travel services.

The cloud infrastructure ensures high availability and smooth performance even during peak usage, while the Flask backend ensures efficient handling of user authentication, dynamic booking flows, and data transactions. Real-time notification integration via AWS SNS allows users to receive booking confirmations and cancellations immediately via email, improving communication and user engagement.

In summary, the **TravelGo** Website offers a modern, reliable, and user-friendly solution for managing travel and accommodation needs. It highlights the potential of cloud-based platforms in building unified travel systems, simplifying operations, and enhancing the overall user experience.

