

## 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.

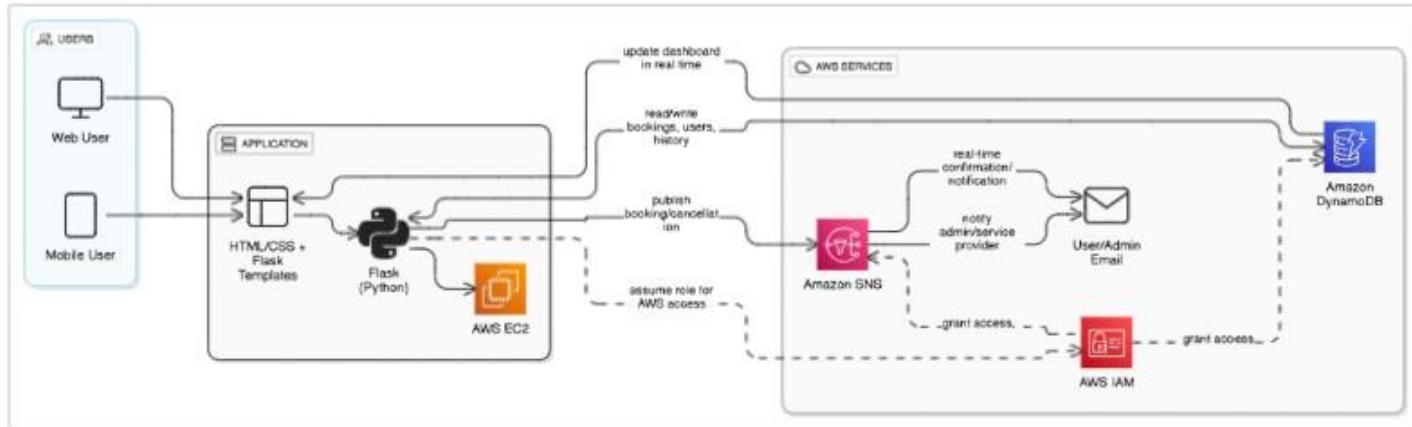
### Scenario 2: Real-Time Booking Confirmation with AWS SNS

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.

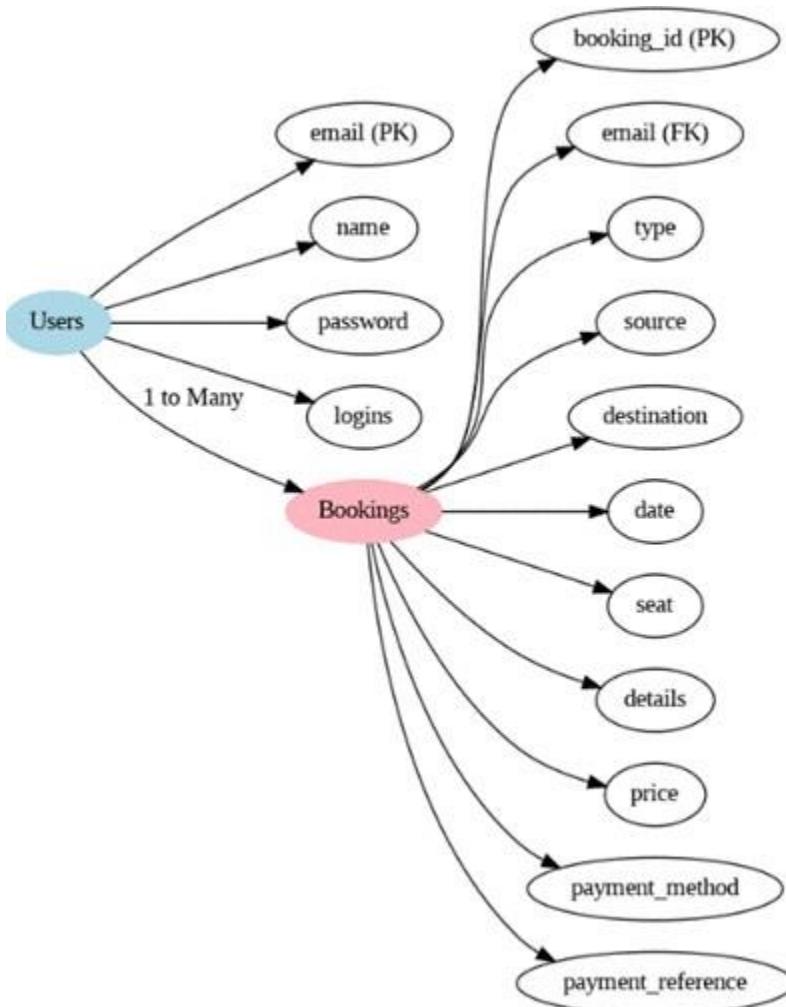
### Scenario 3: Dynamic Dashboard with Personal Travel History

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

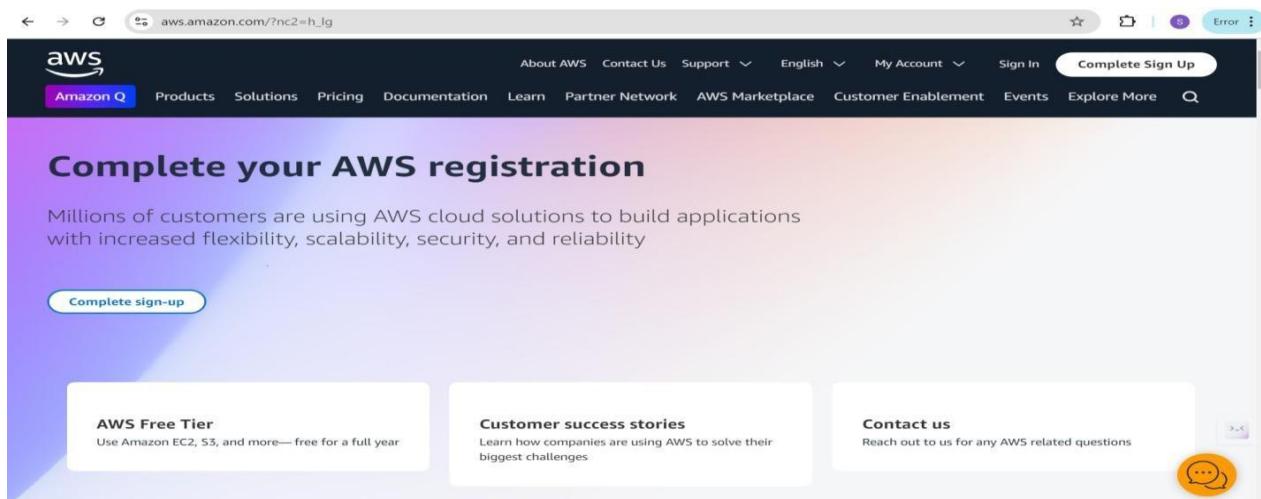
**Activity 7.2:** Run the Flask App

## 8. Testing and Deployment

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

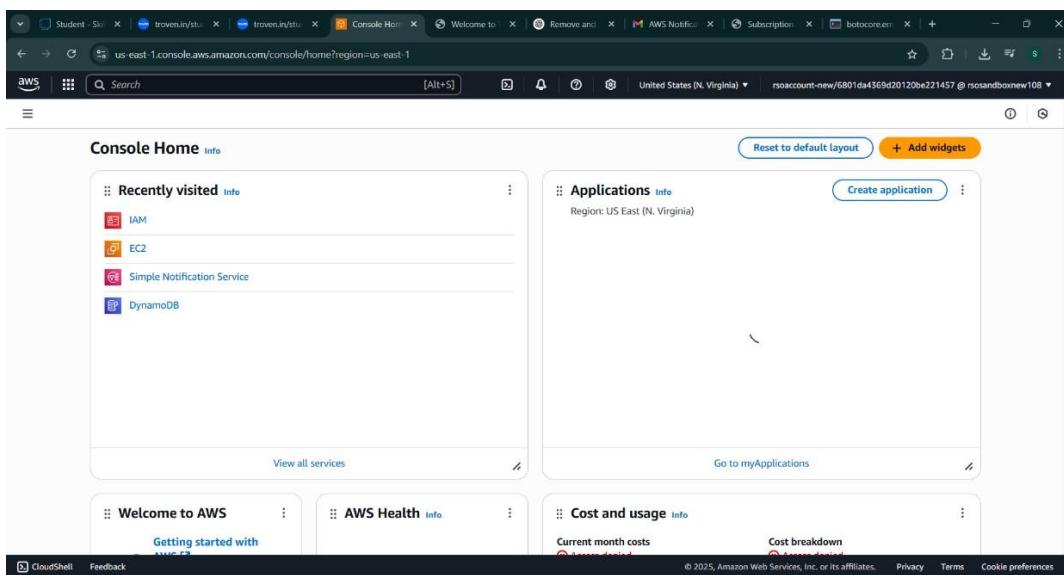
### Milestone 1: AWS Account Setup and Login

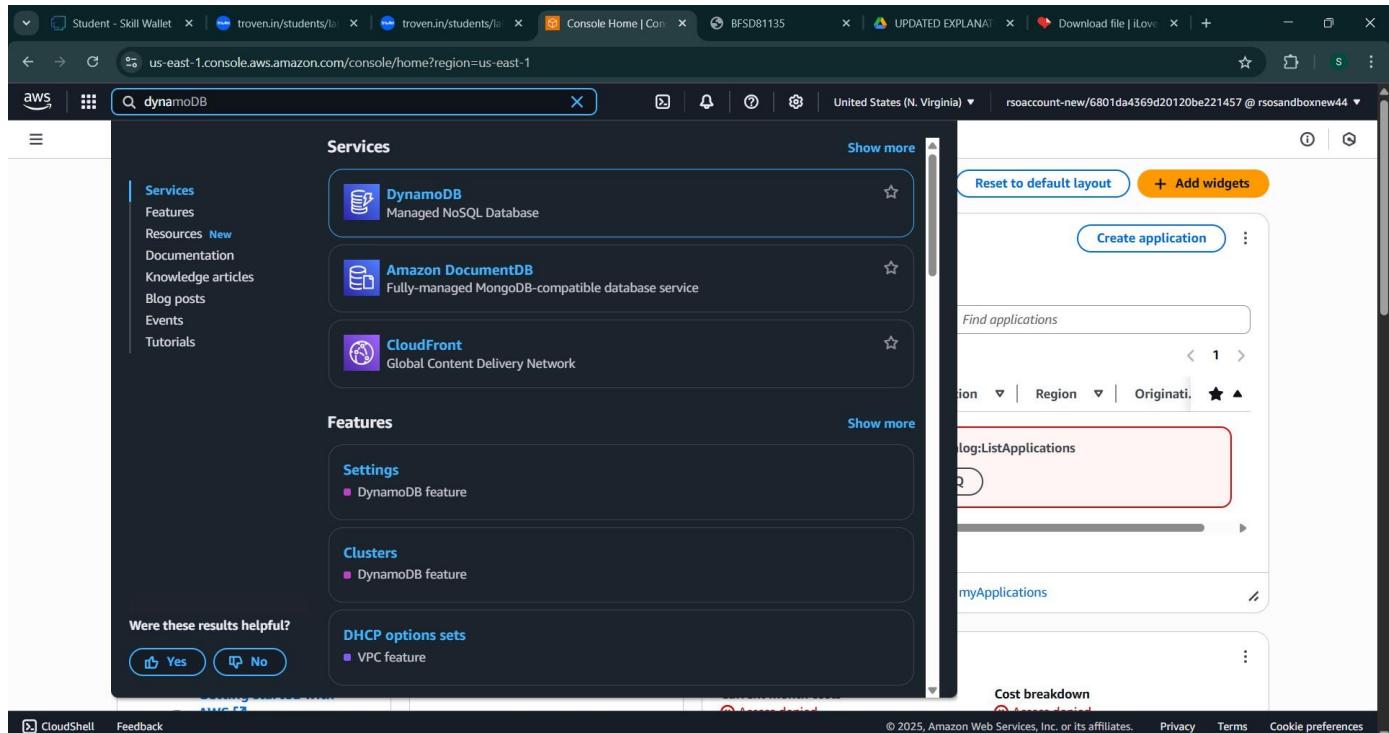
- **Activity 1.1: Set up an AWS account if not already done.**
  - Sign up for an AWS account and configure billing settings.



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

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



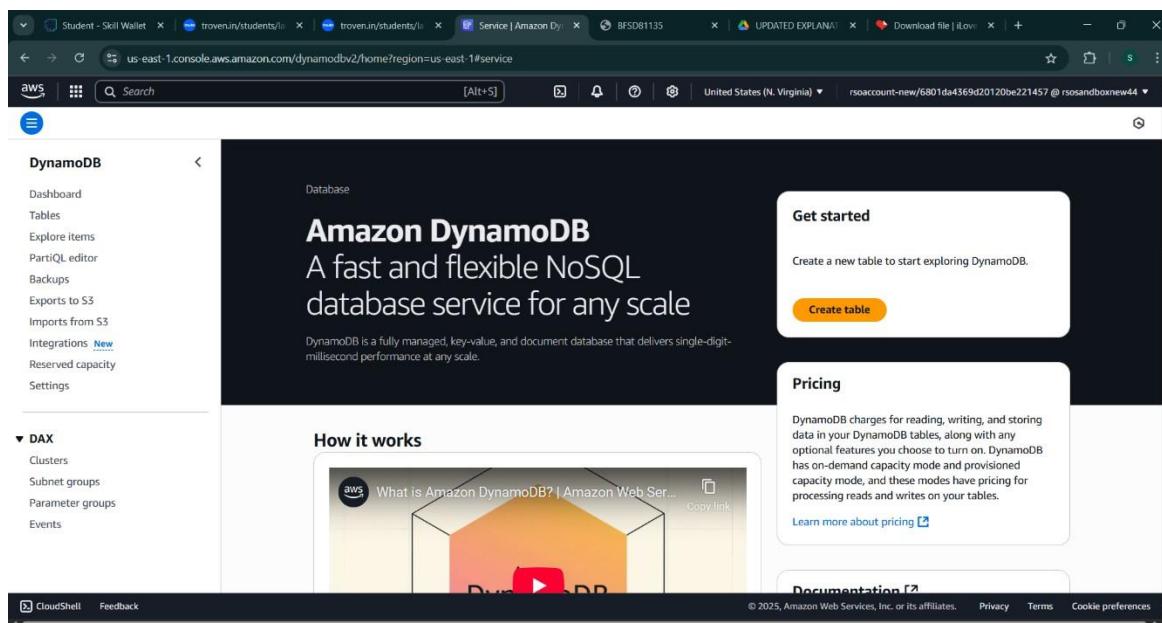


The screenshot shows the AWS Management Console with the search bar set to "dynamoDB". The main content area displays the "Services" section under the "DynamoDB" heading, which is described as a "Managed NoSQL Database". Below this, there are sections for "Amazon DocumentDB" (a "Fully-managed MongoDB-compatible database service") and "CloudFront" (a "Global Content Delivery Network"). The "Features" section includes "Settings" (with a note about the "DynamoDB feature"), "Clusters" (with a note about the "DynamoDB feature"), and "DHCP options sets" (with a note about the "VPC feature"). On the right side, there is a sidebar with options like "Reset to default layout", "Create application", and a "Find applications" search bar. A red box highlights the "myApplications" section.

## 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.



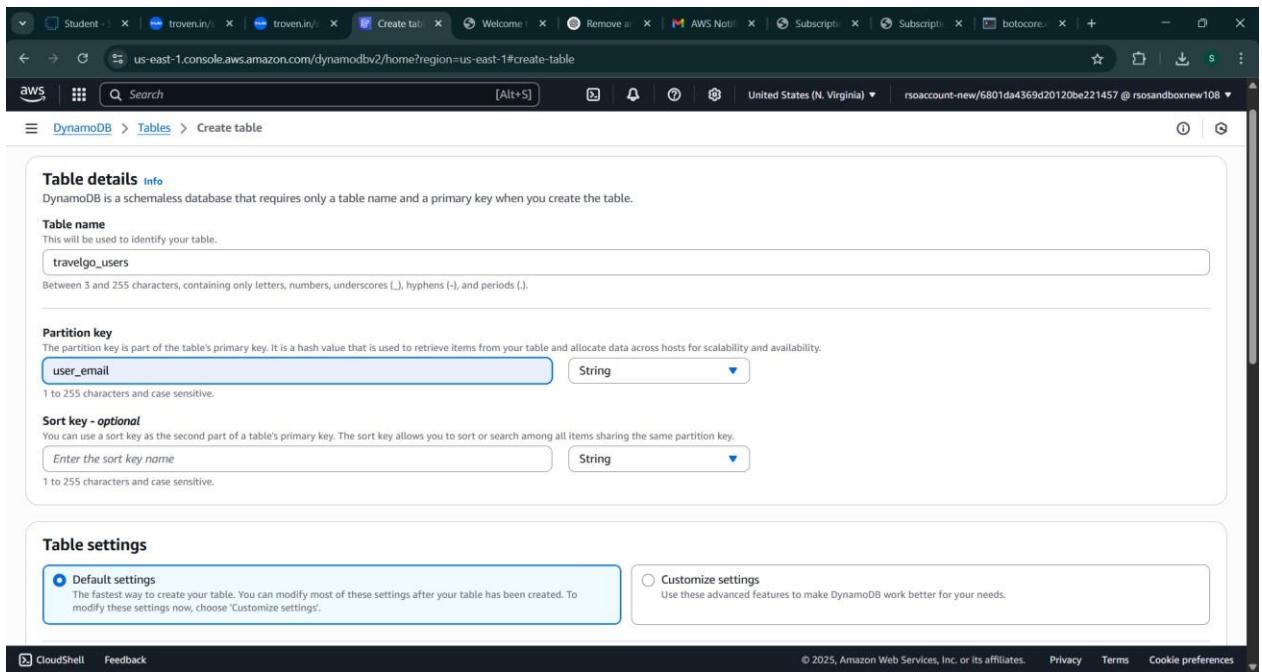
The screenshot shows the "Amazon DynamoDB" service page. The left sidebar lists navigation options: Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, Settings, DAX (Clusters, Subnet groups, Parameter groups, Events), and Documentation. The main content area features a large "Amazon DynamoDB" title with the subtitle "A fast and flexible NoSQL database service for any scale". It includes a brief description of DynamoDB as a fully managed, key-value, and document database. To the right, there are sections for "Get started" (with a "Create table" button) and "Pricing" (describing how DynamoDB charges for reading, writing, and storing data). A video player titled "What is Amazon DynamoDB? | Amazon Web Services" is embedded in the "How it works" section. The bottom right corner contains copyright information: "© 2025, Amazon Web Services, Inc. or its affiliates." and links for 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 'travelpgo\_users'. The partition key is 'user\_email' of type 'String'. There is an optional sort key field, but it is currently empty. Under 'Table settings', the 'Default settings' option is selected. The page includes standard AWS navigation and footer links.



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 > Tables > 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

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

DynamoDB > Tables > 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 'Table settings' step is shown, with the following configuration:

Setting	Value
Maximum write capacity units	-
Local secondary indexes	-
Global secondary indexes	-
Encryption key management	AWS owned key
Deletion protection	Off
Resource-based policy	Not active

**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 SNS service navigation menu, including 'Simple Notification Service', 'Route 53 Resolver', and 'Route 53'. The main content area displays the 'Topics' list:

Topic ARN	Name	Region	Deletion protection	Last updated
arn:aws:sns:us-east-1:6801da4369d20120be221457:TestTopic	TestTopic	us-east-1	Off	2023-09-01 10:00:00
arn:aws:sns:us-east-1:6801da4369d20120be221457:TestTopic2	TestTopic2	us-east-1	Off	2023-09-01 10:00:00
arn:aws:sns:us-east-1:6801da4369d20120be221457:TestTopic3	TestTopic3	us-east-1	Off	2023-09-01 10:00:00

**Services**

- Simple Notification Service
- Route 53 Resolver
- Route 53

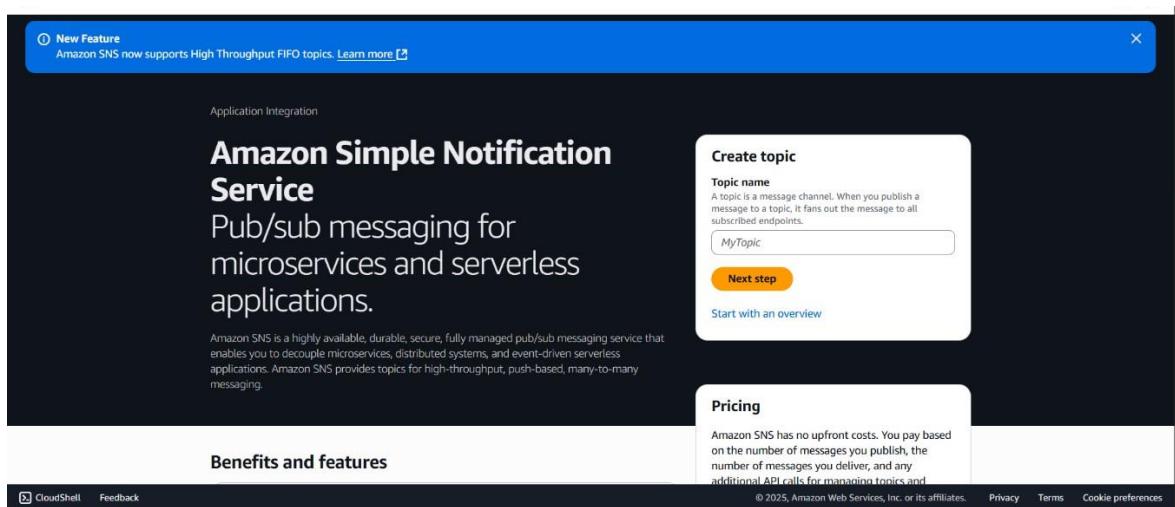
**Features**

- Events
- SMS
- Hosted zones

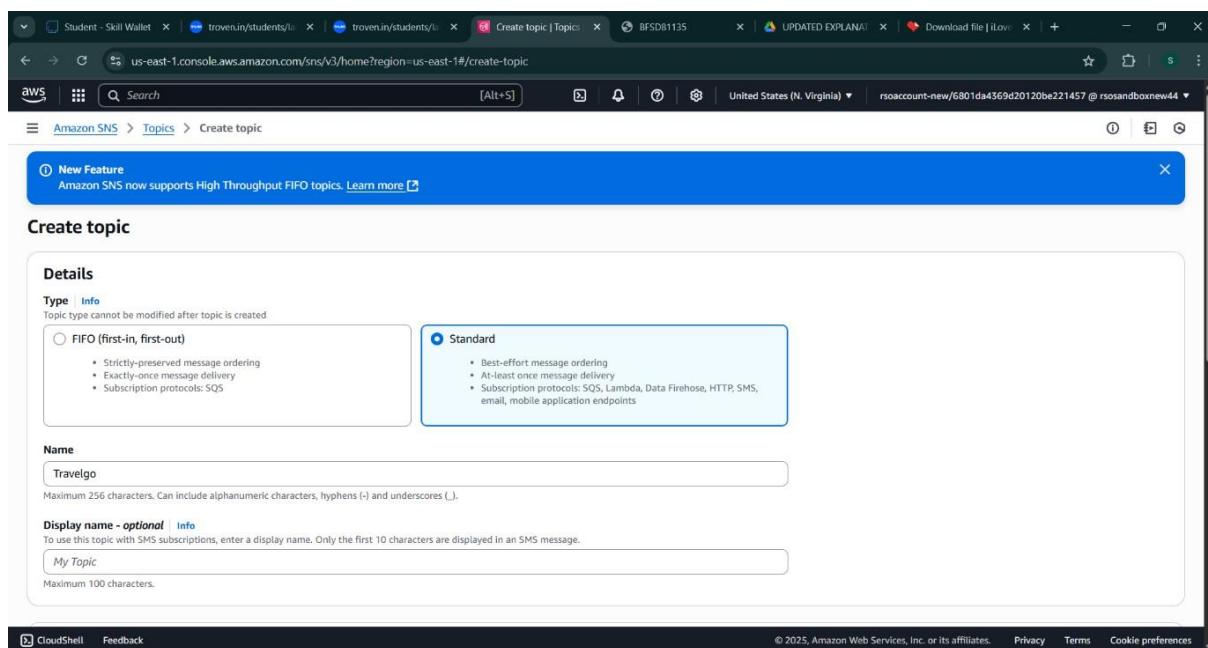
**Were these results helpful?**

[Yes](#) [No](#)

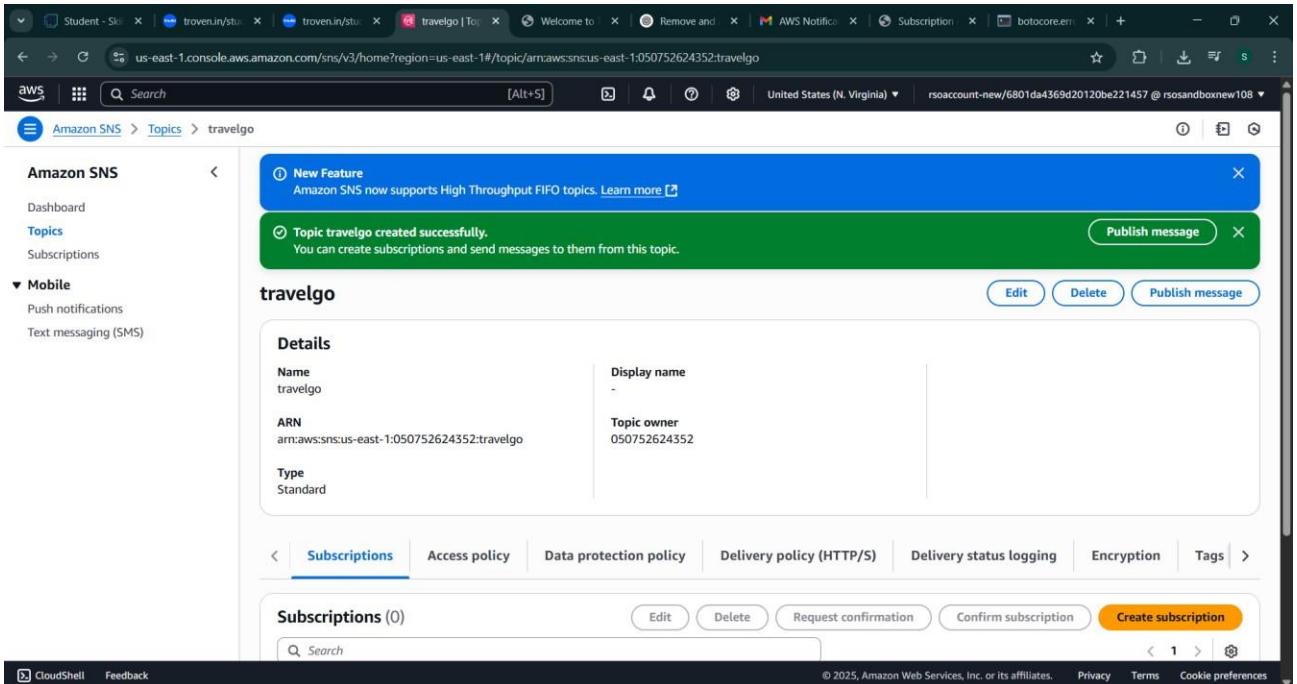
- In the AWS Console, search for SNS and navigate to the SNS Dashboard.



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

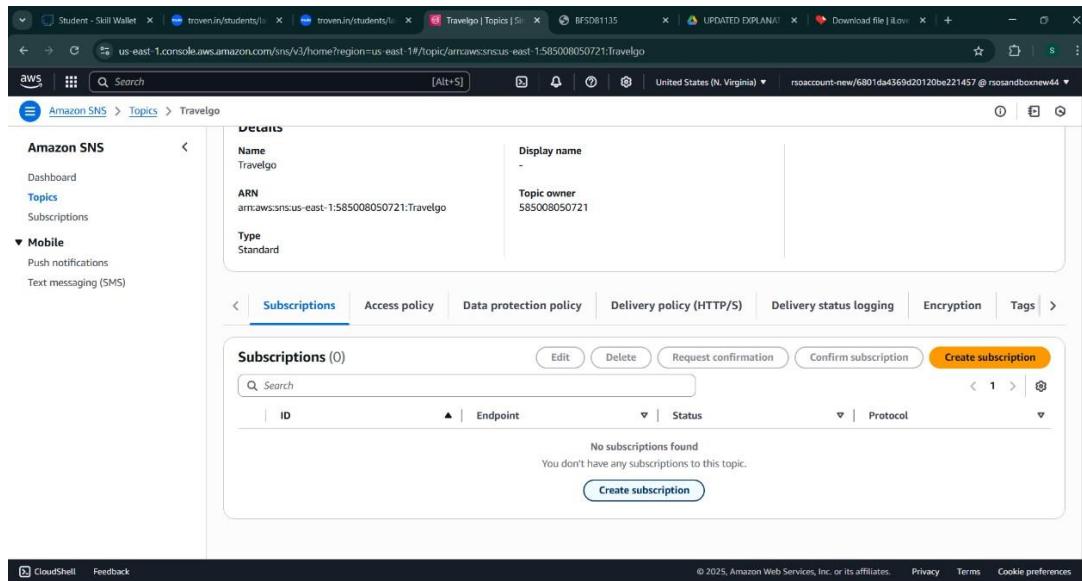


- Choose Standard type for general notification use cases and Click on Create Topic.



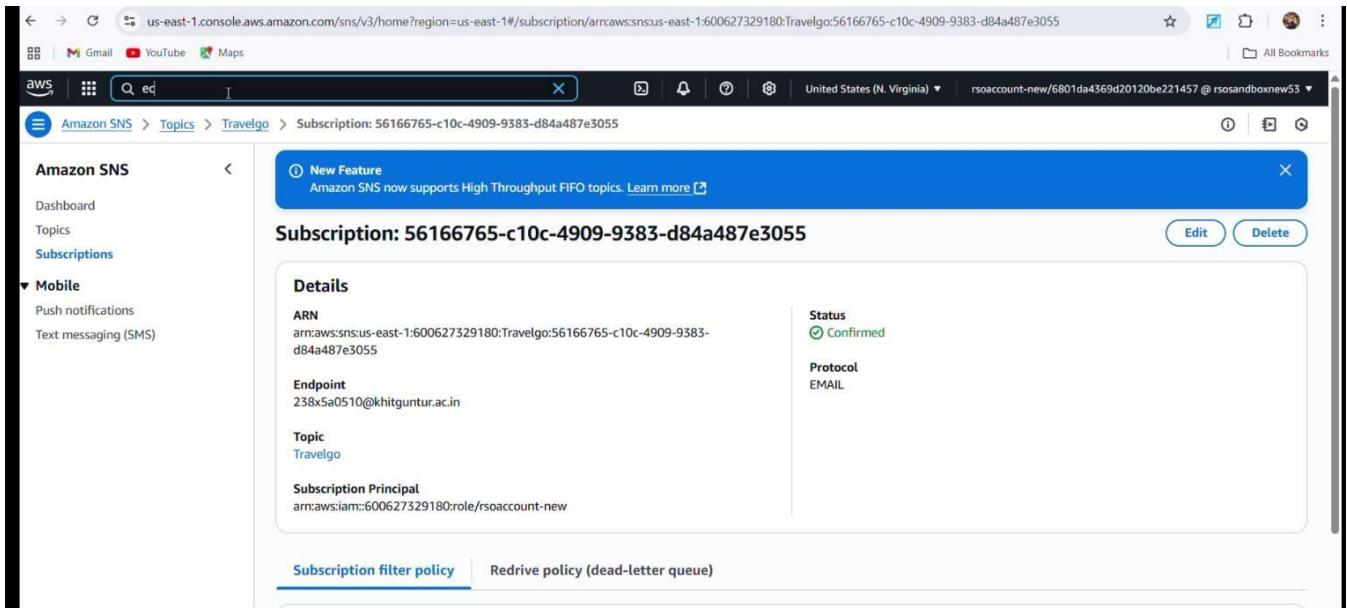
The screenshot shows the AWS SNS Topics page. A success message indicates that the topic 'travelgo' was created successfully. The 'Subscriptions' tab is selected, showing 0 subscriptions. The ARN of the topic is displayed as arn:aws:sns:us-east-1:050752624352:travelgo.

- 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 AWS SNS Topics page for the 'Travelgo' topic. The 'Subscriptions' tab is selected, showing 0 subscriptions. A message at the bottom states 'No subscriptions found. You don't have any subscriptions to this topic.' A 'Create subscription' button is located at the bottom right of the subscriptions table.

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

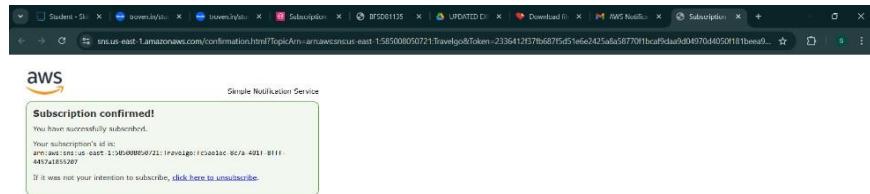


The screenshot shows the AWS SNS console with a subscription details page. The URL in the browser is `us-east-1.console.aws.amazon.com/sns/v3/home?region=us-east-1#/subscription/armaws:sns:us-east-1:600627329180:Travelgo:56166765-c10c-4909-9383-d84a487e3055`. The page title is "Subscription: 56166765-c10c-4909-9383-d84a487e3055". The left sidebar shows "Amazon SNS" with "Subscriptions" selected. The main content area displays the following subscription details:

Details	Status
ARN arn:aws:sns:us-east-1:600627329180:Travelgo:56166765-c10c-4909-9383-d84a487e3055	Confirmed
Endpoint 238x5a0510@khitguntur.ac.in	
Topic Travelgo	EMAIL
Subscription Principal arn:aws:iam::600627329180:role/rsoaccount-new	

Below the table, there are tabs for "Subscription filter policy" and "Redrive policy (dead-letter queue)".

After subscription request for the mail confirmation

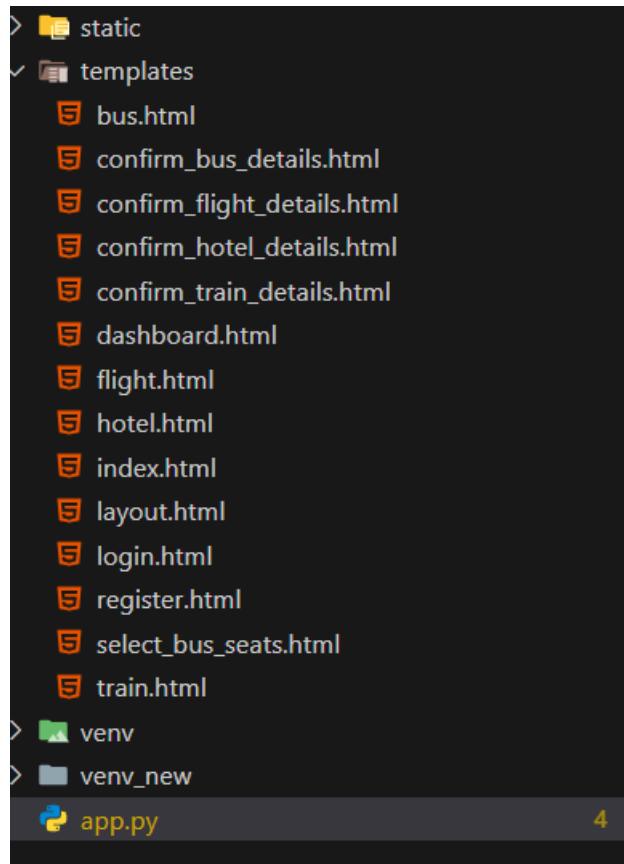


- Successfully done with the SNS mail subscription and setup, now store the ARN link.

## 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:**

```
3 # AWS Setup using IAM Role
4 REGION = 'us-east-1' # Replace with your actual AWS region
5 dynamodb = boto3.resource('dynamodb', region_name=REGION)
6 sns_client = boto3.client('sns', region_name=REGION)
7
8 users_table = dynamodb.Table('travelgo_users')
9 trains_table = dynamodb.Table('trains') # Note: This table is declared by
0 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:

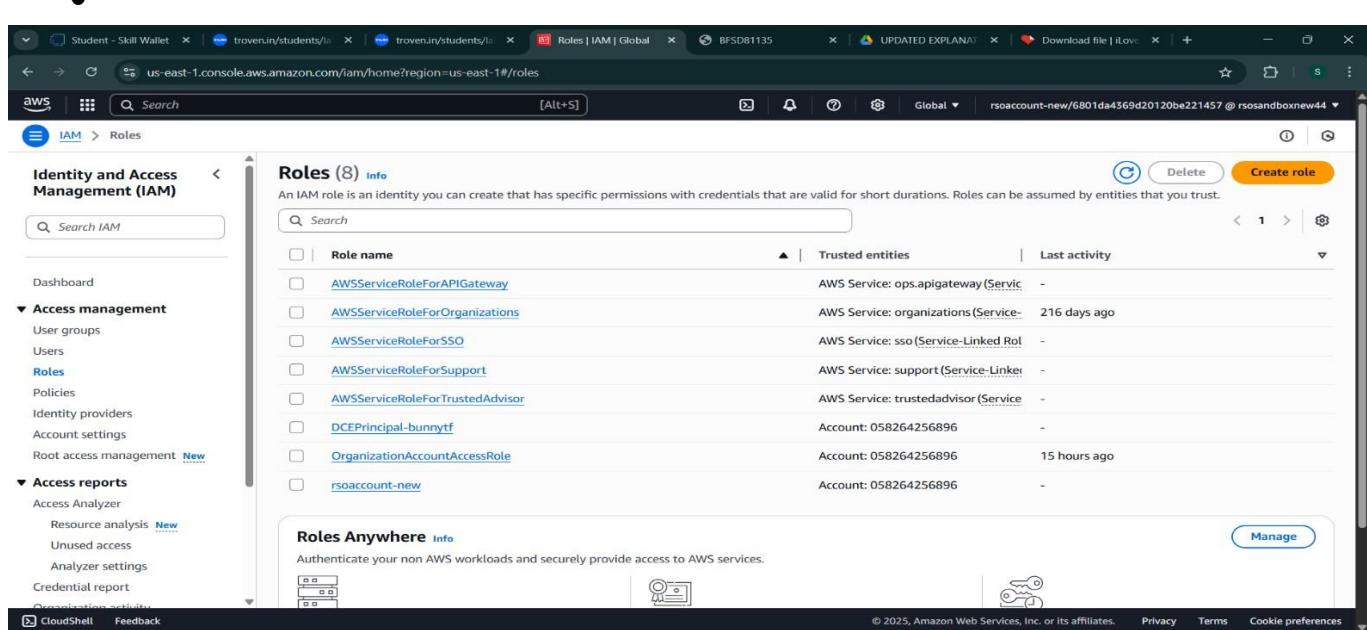
```
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

**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.**

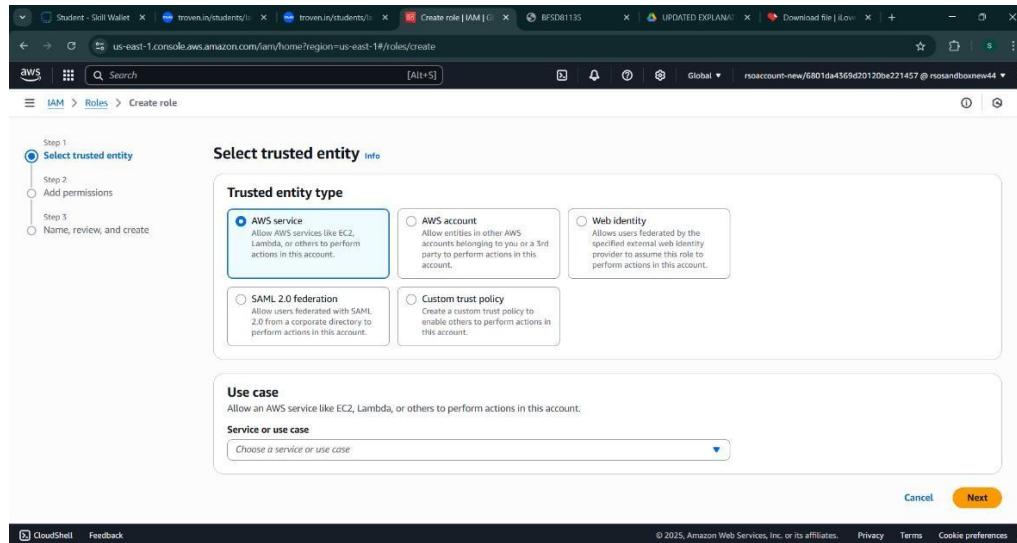
- In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.



The screenshot shows the AWS IAM Roles page with the following details:

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	-

Below the table, there is a section titled "Roles Anywhere" with a "Manage" button.



Step 1  
 **Select trusted entity**

Step 2  
 Add permissions

Step 3  
 Name, review, and create

**Select trusted entity**

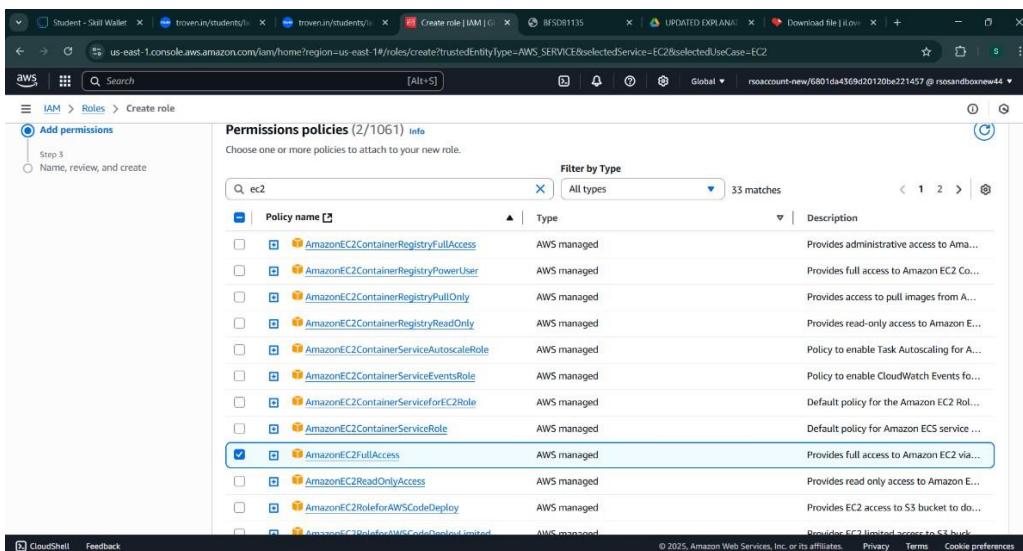
**Trusted entity type**

- AWS service**  
Allow AWS services like EC2, Lambda, or others to perform actions in this account.
- AWS account**  
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
- Web identity**  
Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.
- SAML 2.0 federation**  
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.
- Custom trust policy**  
Create a custom trust policy to enable others to perform actions in this account.

**Use case**  
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

**Service or use case**  
Choose a service or use case

Cancel **Next**



Step 1  
 **Select trusted entity**

Step 2  
 Add permissions

Step 3  
 Name, review, and create

**Permissions policies (2/1061)**

Choose one or more policies to attach to your new role.

Policy name	Type	Description
<input checked="" type="checkbox"/> <a href="#">AmazonEC2FullAccess</a>	AWS managed	Provides full access to Amazon EC2 via...
<input type="checkbox"/> <a href="#">AmazonEC2ContainerRegistryFullAccess</a>	AWS managed	Provides administrative access to Amazon...
<input type="checkbox"/> <a href="#">AmazonEC2ContainerRegistryPowerUser</a>	AWS managed	Provides full access to Amazon EC2 Co...
<input type="checkbox"/> <a href="#">AmazonEC2ContainerRegistryPullOnly</a>	AWS managed	Provides access to pull images from Am...
<input type="checkbox"/> <a href="#">AmazonEC2ContainerRegistryReadOnly</a>	AWS managed	Provides read-only access to Amazon E...
<input type="checkbox"/> <a href="#">AmazonEC2ContainerServiceAutoscaleRole</a>	AWS managed	Policy to enable Task Autoscaling for A...
<input type="checkbox"/> <a href="#">AmazonEC2ContainerServiceEventsRole</a>	AWS managed	Policy to enable CloudWatch Events fo...
<input type="checkbox"/> <a href="#">AmazonEC2ContainerServiceForEC2Role</a>	AWS managed	Default policy for the Amazon EC2 Rol...
<input type="checkbox"/> <a href="#">AmazonEC2ContainerServiceRole</a>	AWS managed	Default policy for Amazon ECS service ...
<input type="checkbox"/> <a href="#">AmazonEC2ReadonlyAccess</a>	AWS managed	Provides read only access to Amazon E...
<input type="checkbox"/> <a href="#">AmazonEC2RoleforAWSCodeDeploy</a>	AWS managed	Provides EC2 access to S3 bucket to do...
<input type="checkbox"/> <a href="#">AmazonEC2RoleforAWSCodeDeployLambda</a>	AWS managed	Provides EC2 limited access to S3 buck...

CloudShell Feedback

## • 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 table lists six AWS managed policies:

Policy name	Type	Description
<input checked="" type="checkbox"/>  AmazonDynamoDBFullAccess	AWS managed	Provides full access to Amazon Dynamo...
<input type="checkbox"/>  AmazonDynamoDBFullAccess_v2	AWS managed	Provides full access to Amazon Dynam...
<input type="checkbox"/>  AmazonDynamoDBFullAccessWithDataPipeline	AWS managed	This policy is on a deprecation path. Se...
<input type="checkbox"/>  AmazonDynamoDBReadOnlyAccess	AWS managed	Provides read only access to Amazon D...
<input type="checkbox"/>  AWSLambdaDynamoDBExecutionRole	AWS managed	Provides list and read access to Dynam...
<input type="checkbox"/>  AWSLambdaInvocation-DynamoDB	AWS managed	Provides read access to DynamoDB Str...

**Set permissions boundary - optional**

Buttons: Cancel, Previous, Next

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

The search bar shows 'sns'. The results table lists five AWS managed policies:

Policy name	Type	Description
<input checked="" type="checkbox"/>  AmazonSNSFullAccess	AWS managed	Provides full access to Amazon SNS via...
<input type="checkbox"/>  AmazonSNSReadOnlyAccess	AWS managed	Provides read only access to Amazon S...
<input type="checkbox"/>  AmazonSNSRole	AWS managed	Default policy for Amazon SNS service...
<input type="checkbox"/>  AWSElasticBeanstalkRoleSNS	AWS managed	(Elastic Beanstalk operations role) Allo...
<input type="checkbox"/>  AWSIoTDeviceDefenderPublishFindingsToSN...	AWS managed	Provides messages publish access to S...

**Set permissions boundary - optional**

Buttons: Cancel, Previous, Next

## Milestone 6: EC2 Instance Setup

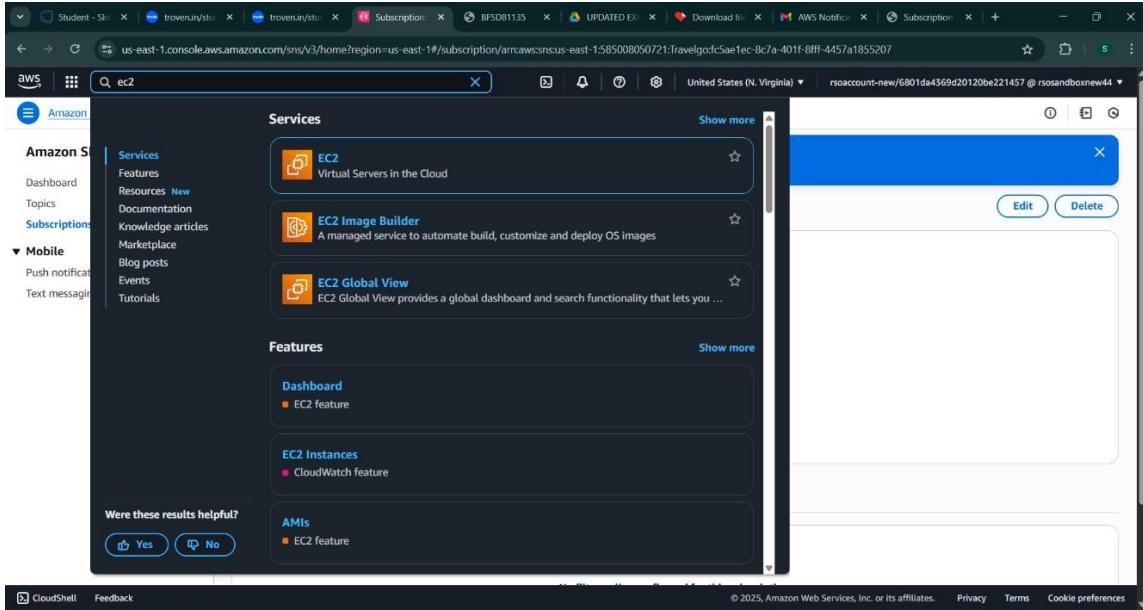
- Note: Load your Flask app and Html files into GitHub repository.

 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

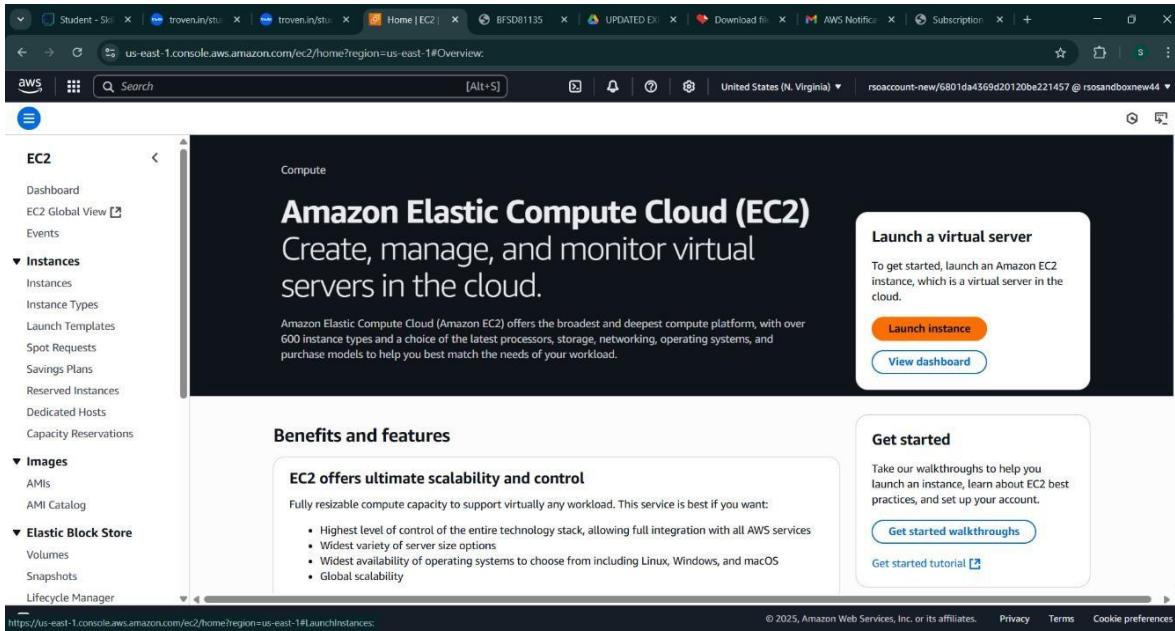
- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

- **Launch EC2 Instance**

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



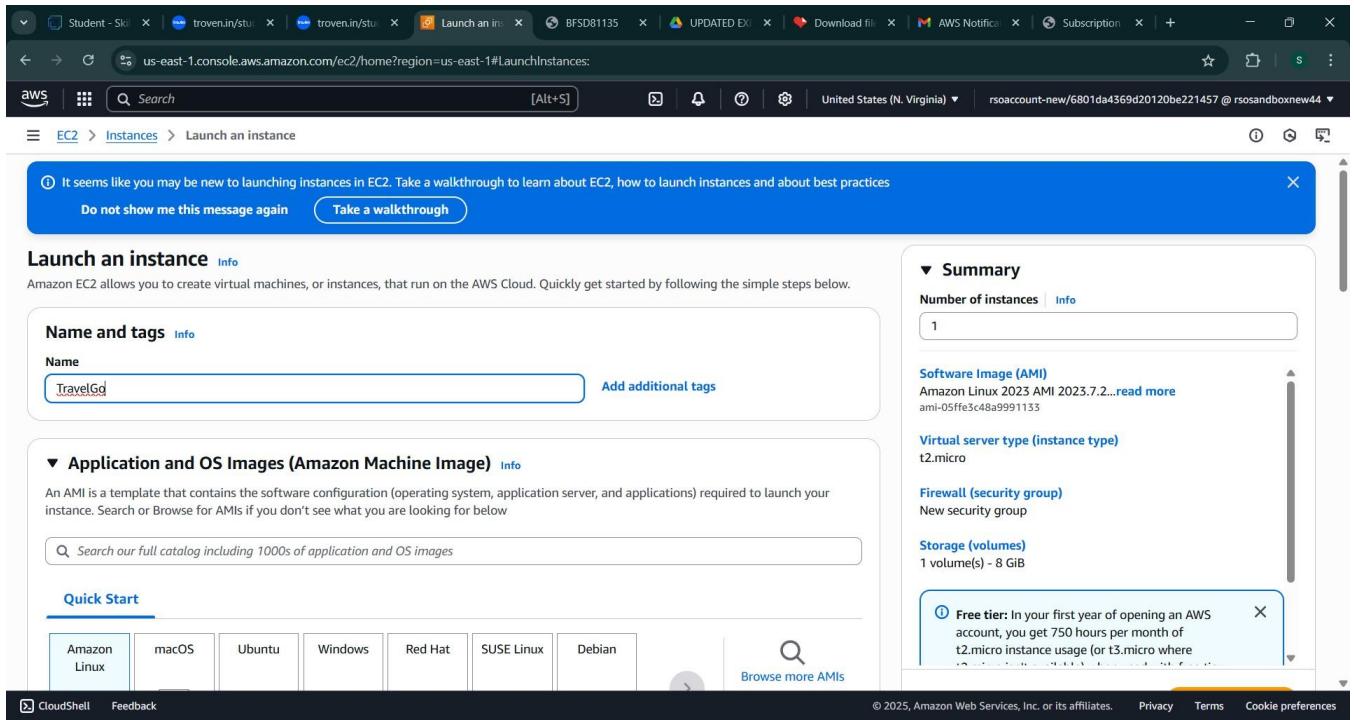
The screenshot shows the AWS search results for 'ec2'. The left sidebar has a 'Mobile' section. The main area lists services under 'Services': EC2 (Virtual Servers in the Cloud), EC2 Image Builder (A managed service to automate build, customize and deploy OS images), and EC2 Global View (EC2 Global View provides a global dashboard and search functionality that lets you ...). Under 'Features', there are sections for 'Dashboard' (EC2 feature) and 'EC2 Instances' (CloudWatch feature). At the bottom, there are 'AMIs' (EC2 feature) and a feedback section asking 'Were these results helpful?' with 'Yes' and 'No' buttons.



The screenshot shows the AWS EC2 home page. The left sidebar has sections for 'Compute' (Dashboard, EC2 Global View, Events), 'Instances' (Instances 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 features a large heading 'Amazon Elastic Compute Cloud (EC2)' with the subtext 'Create, manage, and monitor virtual servers in the cloud.' It includes a callout for 'Launch a virtual server' with 'Launch instance' and 'View dashboard' buttons. Below this, there are sections for 'Benefits and features' (with a sub-section 'EC2 offers ultimate scalability and control') and 'Get started' (with 'Get started walkthroughs' and 'Get started tutorial' buttons). The URL in the address bar is <https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#LaunchInstances>.

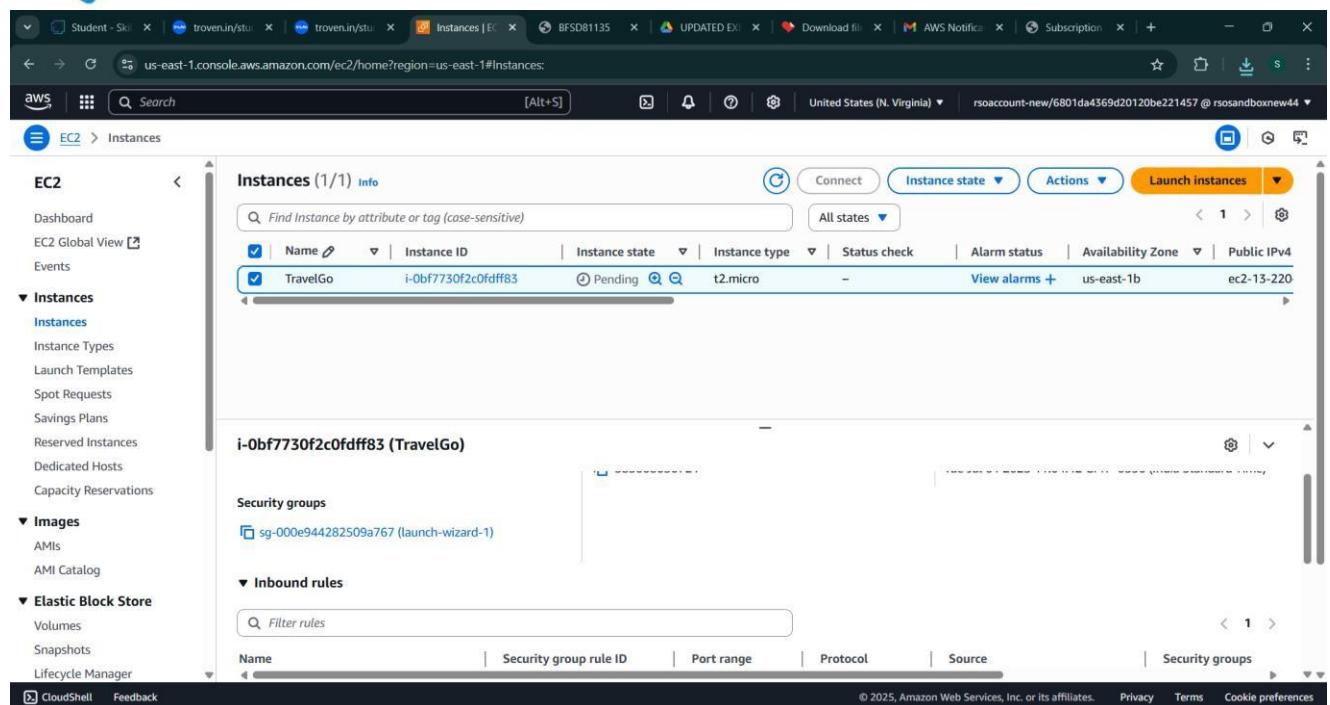
- Click on Launch instance to launch EC2 instance

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



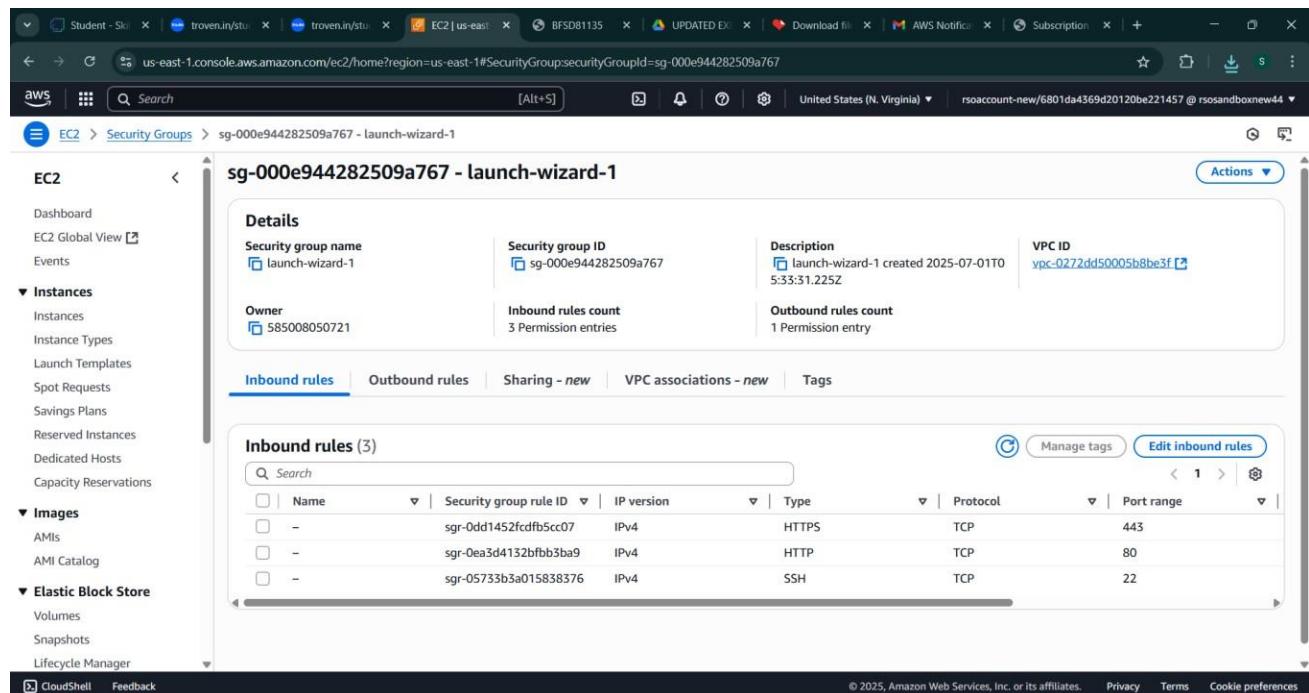
The screenshot shows the AWS EC2 'Launch an instance' wizard. The current step is 'Name and tags'. A message at the top says: 'It seems like you may be new to launching instances in EC2. Take a walkthrough to learn about EC2, how to launch instances and about best practices'. There are 'Do not show me this message again' and 'Take a walkthrough' buttons. Below this, the 'Launch an instance' section starts with 'Name and tags' and a 'Name' input field containing 'TravelGd'. To the right is a 'Add additional tags' button. The next section is 'Application and OS Images (Amazon Machine Image)', which includes a search bar and a 'Quick Start' grid of operating system icons: Amazon Linux, macOS, Ubuntu, Windows, Red Hat, SUSE Linux, and Debian. To the right of the wizard is a sidebar titled 'Summary' showing 'Number of instances' (1), 'Software Image (AMI)' (Amazon Linux 2023 AMI 2023.7.2...), 'Virtual server type (instance type)' (t2.micro), 'Firewall (security group)' (New security group), and 'Storage (volumes)' (1 volume(s) - 8 GiB). A 'Free tier' message is also visible.

- Create and download the key pair for Server access.



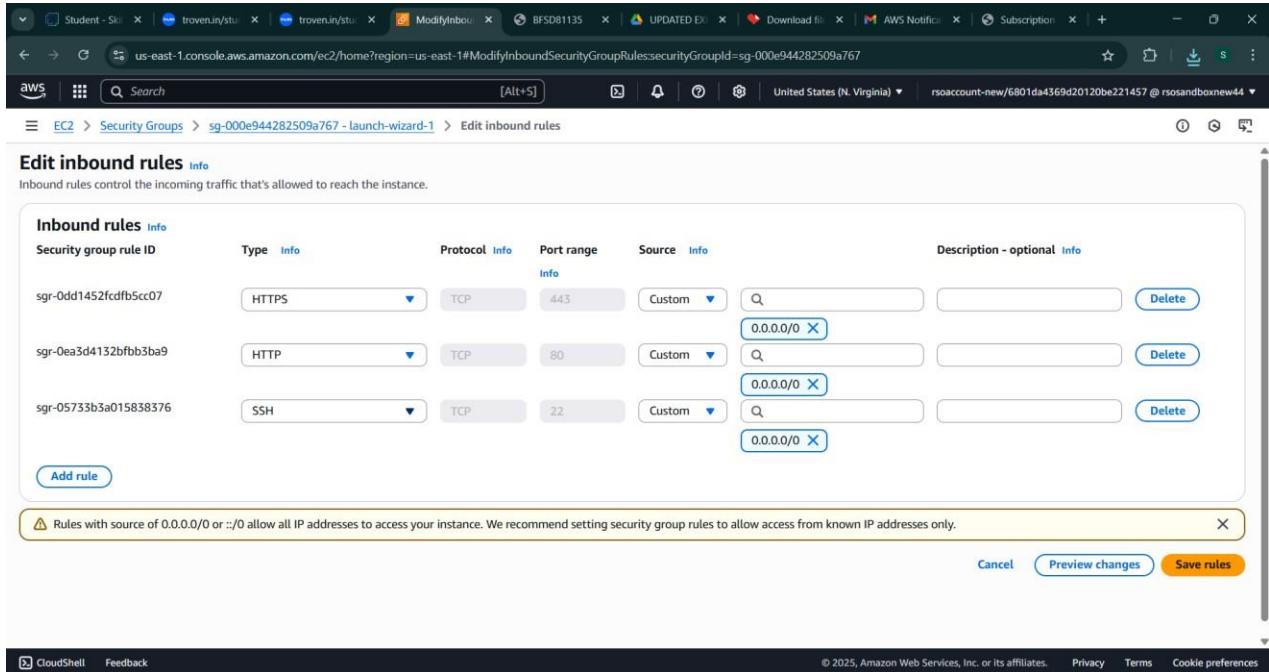
The screenshot shows the AWS EC2 Instances page. On the left, there is a navigation sidebar with options like Dashboard, EC2 Global View, Events, Instances (selected), Images, and Elastic Block Store. The main content area displays a table titled "Instances (1/1) Info" with one row for "TravelGo". The instance has an ID of i-0bf7730f2c0fdff83, is pending, and is a t2.micro type. It is associated with the "us-east-1b" availability zone and has a public IPv4 address ec2-13-220. Below the table, there is a detailed view for the selected instance, showing its security groups (sg-000e944282509a767) and inbound rules.

- **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 Instances, Images, and Elastic Block Store. The main area is focused on the "sg-000e944282509a767 - launch-wizard-1" security group. It shows details like the security group name (launch-wizard-1), ID (sg-000e944282509a767), owner (585008050721), and a count of 3 permission entries. The "Inbound rules" tab is selected, displaying three rules: one for HTTPS (port 443), one for HTTP (port 80), and one for SSH (port 22).

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



**Edit inbound rules** Info

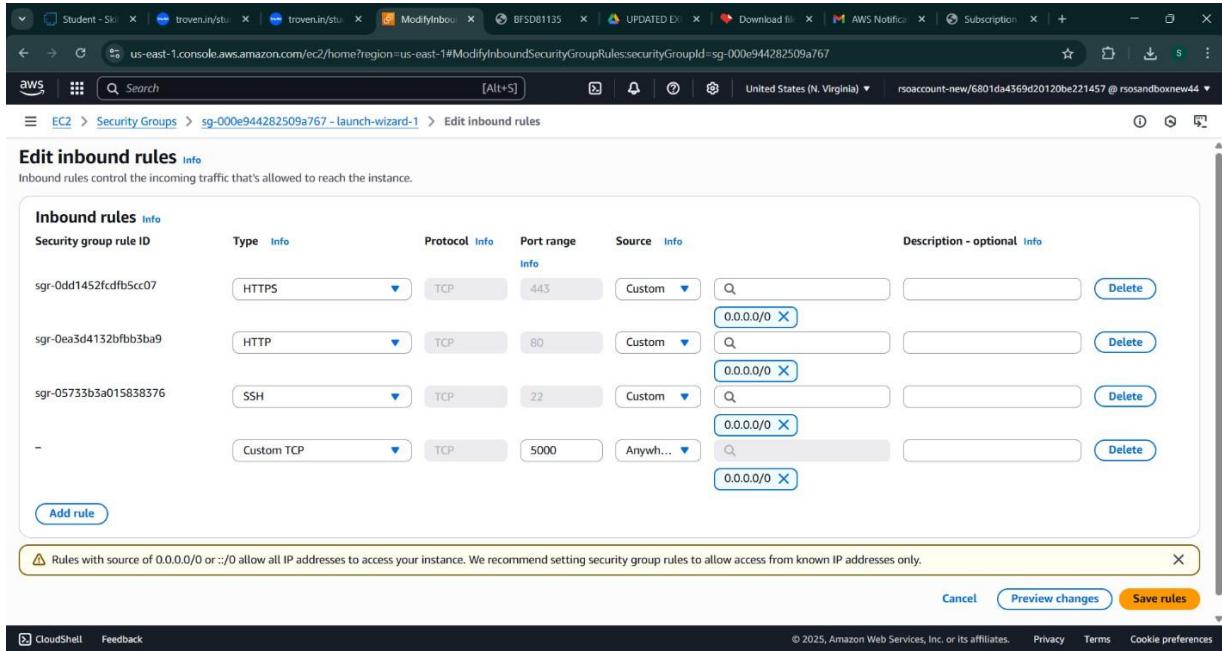
Inbound rules control the incoming traffic that's allowed to reach the instance.

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0dd1452fcdfb5cc07	HTTPS	TCP	443	Custom	<input type="text"/> 0.0.0.0/0 <span style="color: red;">X</span>
sgr-0ea3d4132bfb3ba9	HTTP	TCP	80	Custom	<input type="text"/> 0.0.0.0/0 <span style="color: red;">X</span>
sgr-05733b3a015838376	SSH	TCP	22	Custom	<input type="text"/> 0.0.0.0/0 <span style="color: red;">X</span>

[Add rule](#)

⚠ Rules with source of 0.0.0.0/0 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](#)



**Edit inbound rules** Info

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

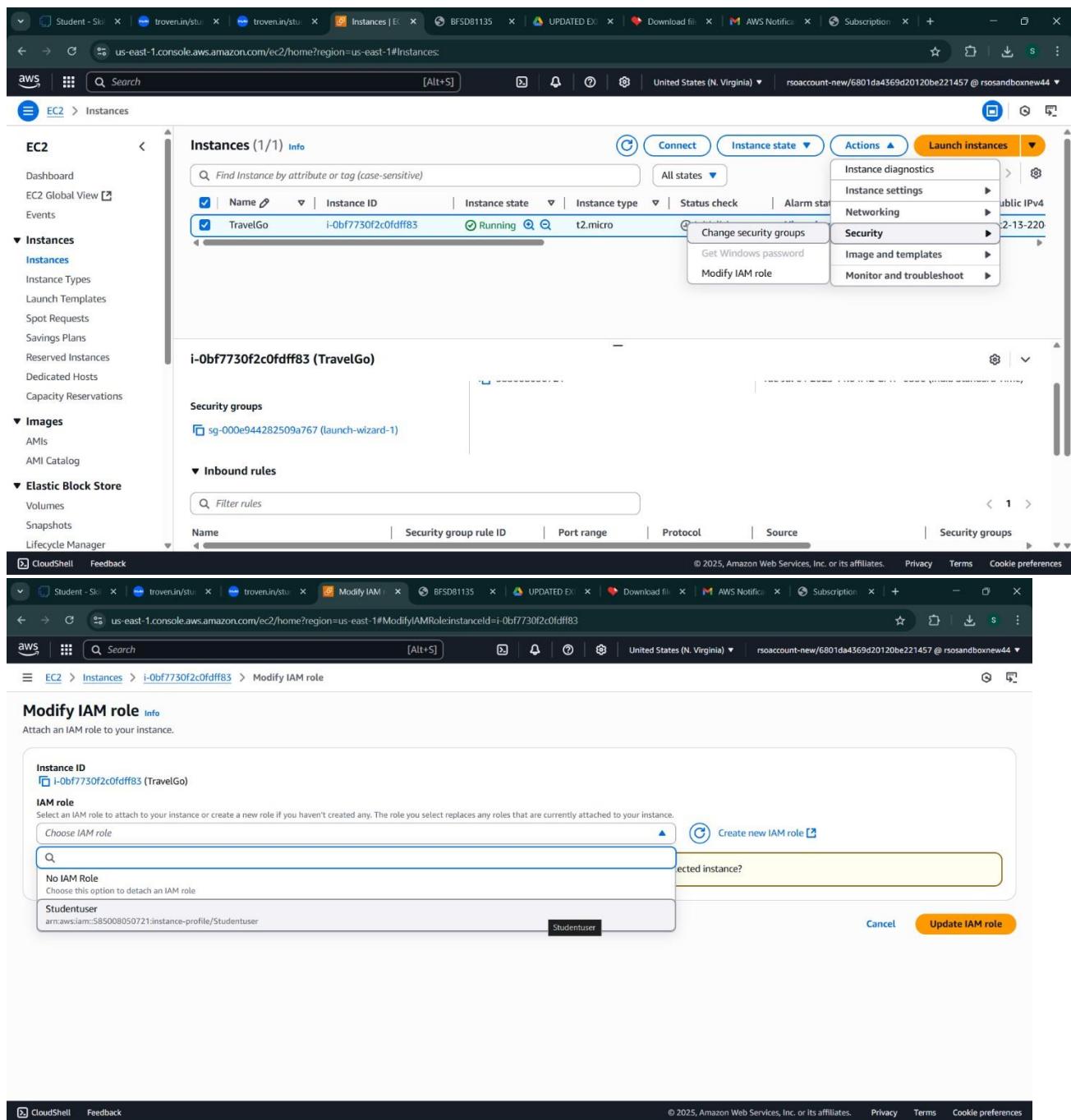
Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0dd1452fcdfb5cc07	HTTPS	TCP	443	Custom	<input type="text"/> 0.0.0.0/0 <span style="color: red;">X</span>
sgr-0ea3d4132bfb3ba9	HTTP	TCP	80	Custom	<input type="text"/> 0.0.0.0/0 <span style="color: red;">X</span>
sgr-05733b3a015838376	SSH	TCP	22	Custom	<input type="text"/> 0.0.0.0/0 <span style="color: red;">X</span>
-	Custom TCP	TCP	5000	Anywhere	<input type="text"/> 0.0.0.0/0 <span style="color: red;">X</span>

[Add rule](#)

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

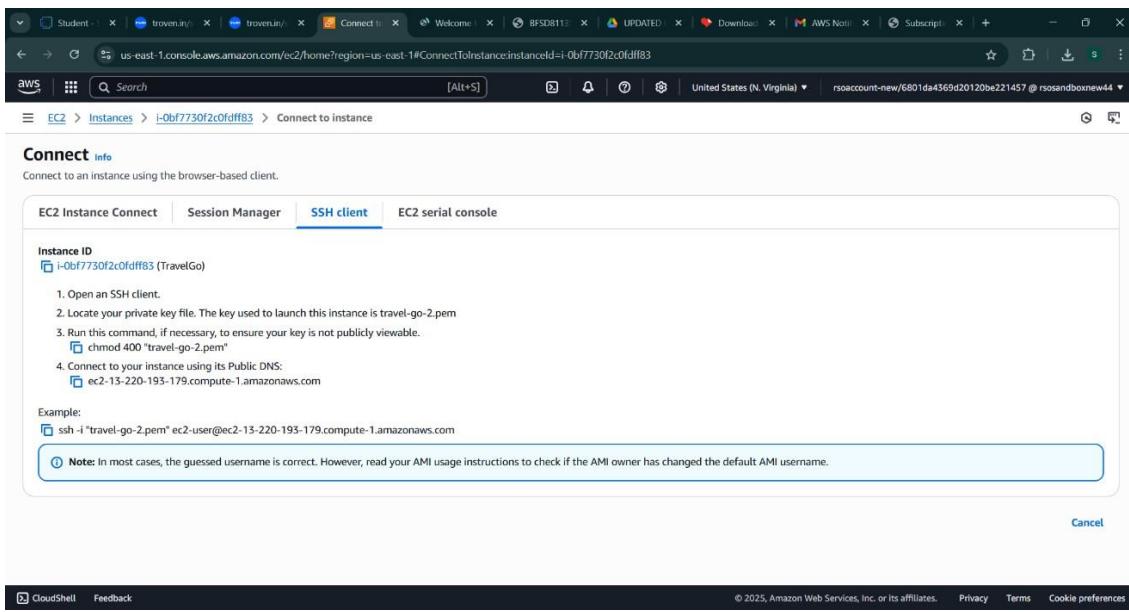
[Cancel](#) [Preview changes](#) [Save rules](#)

- 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 browser tabs. The top tab displays the AWS Management Console EC2 Instances page for the 'TravelGo' instance (i-0bf7730f2c0fdff83). The instance is running and has the t2.micro type. A context menu is open over the instance, with the 'Security' option selected, which highlights the 'Modify IAM role' option. The bottom tab shows the 'Modify IAM role' dialog for the same instance. In the dialog, the 'Instance ID' field is populated with 'i-0bf7730f2c0fdff83 (TravelGo)'. The 'IAM role' section contains a dropdown menu with 'Choose IAM role' and a search bar, and a link to 'Create new IAM role'. Below this, there is a note about selecting an IAM role or creating a new one. The 'Selected instance?' dropdown is set to 'Studentuser'. 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 the navigation bar, it shows the path **EC2 > Instances > i-0bf7730f2c0fdff83 > Connect to instance**.

**Connect** Info

Connect to an instance using the browser-based client.

**SSH client** (selected)

**EC2 Instance Connect** | **Session Manager** | **SSH client** | **EC2 serial console**

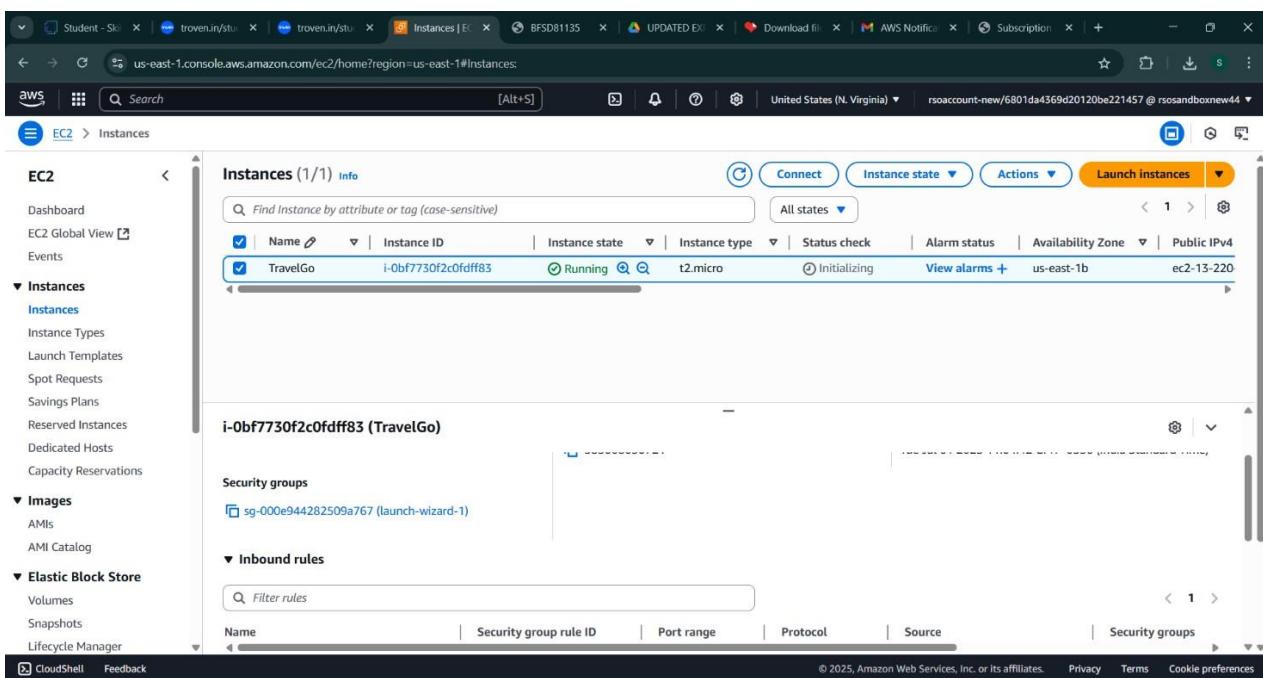
**Instance ID**: **i-0bf7730f2c0fdff83 (TravelGo)**

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:  
`ssh -i "travel-go-2.pem" ec2-user@ec2-13-220-193-179.compute-1.amazonaws.com`

**Example:**  
`ssh -i "travel-go-2.pem" ec2-user@ec2-13-220-193-179.compute-1.amazonaws.com`

**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.

[Cancel](#)



The screenshot shows the AWS EC2 Instances page. The left sidebar shows the navigation menu: **EC2**, Dashboard, EC2 Global View, Events, **Instances** (selected), 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 shows the **Instances (1/1) Info** section. It lists one instance: **i-0bf7730f2c0fdff83 (TravelGo)**. The instance is **Running** (t2.micro, Initializing). It is associated with the security group **sg-000e944282509a767 (launch-wizard-1)**.

Below the instance details, the **Inbound rules** section is shown, with a table for filtering rules.

At the bottom, there are links for CloudShell and Feedback, and standard footer links for Privacy, Terms, and Cookie preferences.

## 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/your-github-username/your-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
```

## Verify the Flask app is running:

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

- Run the Flask app on the EC2 instance

```
  Downloading zipp-3.23.0-py3-none-any.whl (10 kB)
Installing collected packages: zipp, markupsafe, werkzeug, jinja2, itsdangerous, importlib-metadata, click, blinker, flask
Successfully installed blinker-1.9.0 click-8.1.8 flask-3.1.1 importlib-metadata-8.7.0 itsdangerous-2.2.0 jinja2-3.1.6 markupsafe-3.0.2 werkzeug-3.1.3 zipp-3.23.0
[ec2-user@ip-172-31-26-55 Travel-go]$ pip install boto3
Defaulting to user installation because normal site-packages is not writeable
Collecting boto3
  Downloading boto3-1.39.3-py3-none-any.whl (139 kB)
    ━━━━━━━━━━━━━━━━ 139 kB 16.3 MB/s
Collecting s3transfer<0.14.0,>=0.13.0
  Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB)
    ━━━━━━━━━━━━━━ 85 kB 7.6 MB/s
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (from boto3) (0.10.0)
Collecting botocore<1.40.0,>=1.39.3
  Downloading botocore-1.39.3-py3-none-any.whl (13.8 MB)
    ━━━━━━━━━━━━━━ 13.8 MB 39.1 MB/s
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.3->boto3) (1.25.10)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.3->boto3) (2.8.1)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.40.0,>=1.39.3->boto3) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed boto3-1.39.3 botocore-1.39.3 s3transfer-0.13.0
[ec2-user@ip-172-31-26-55 Travel-go]$ 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://172.31.26.55:5000
Press CTRL+C to quit
 * Restarting with stat
```

## Milestone 8: Testing and Deployment

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

**Welcome Page:**

**Register Page:**

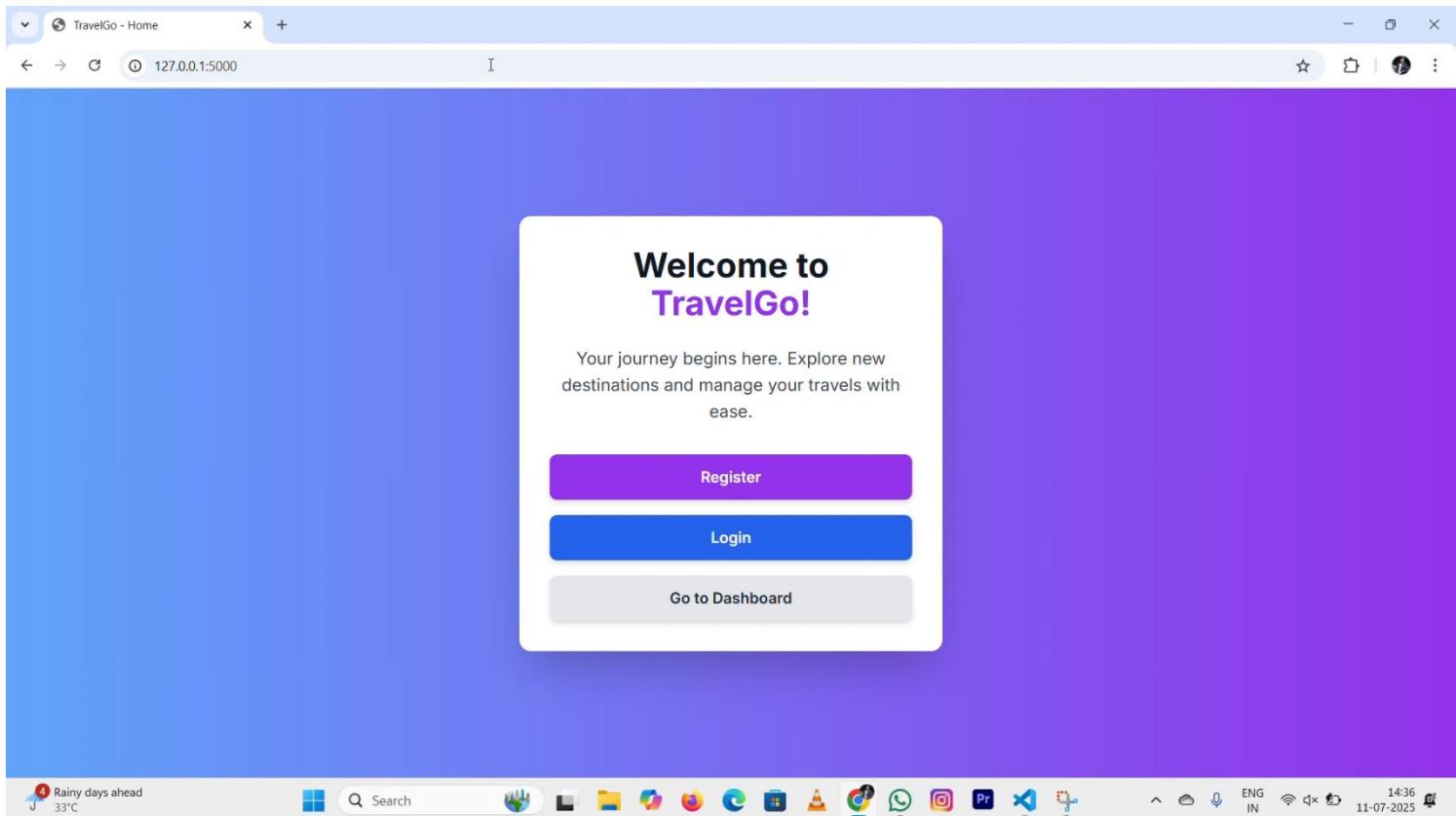
**Login Page:**

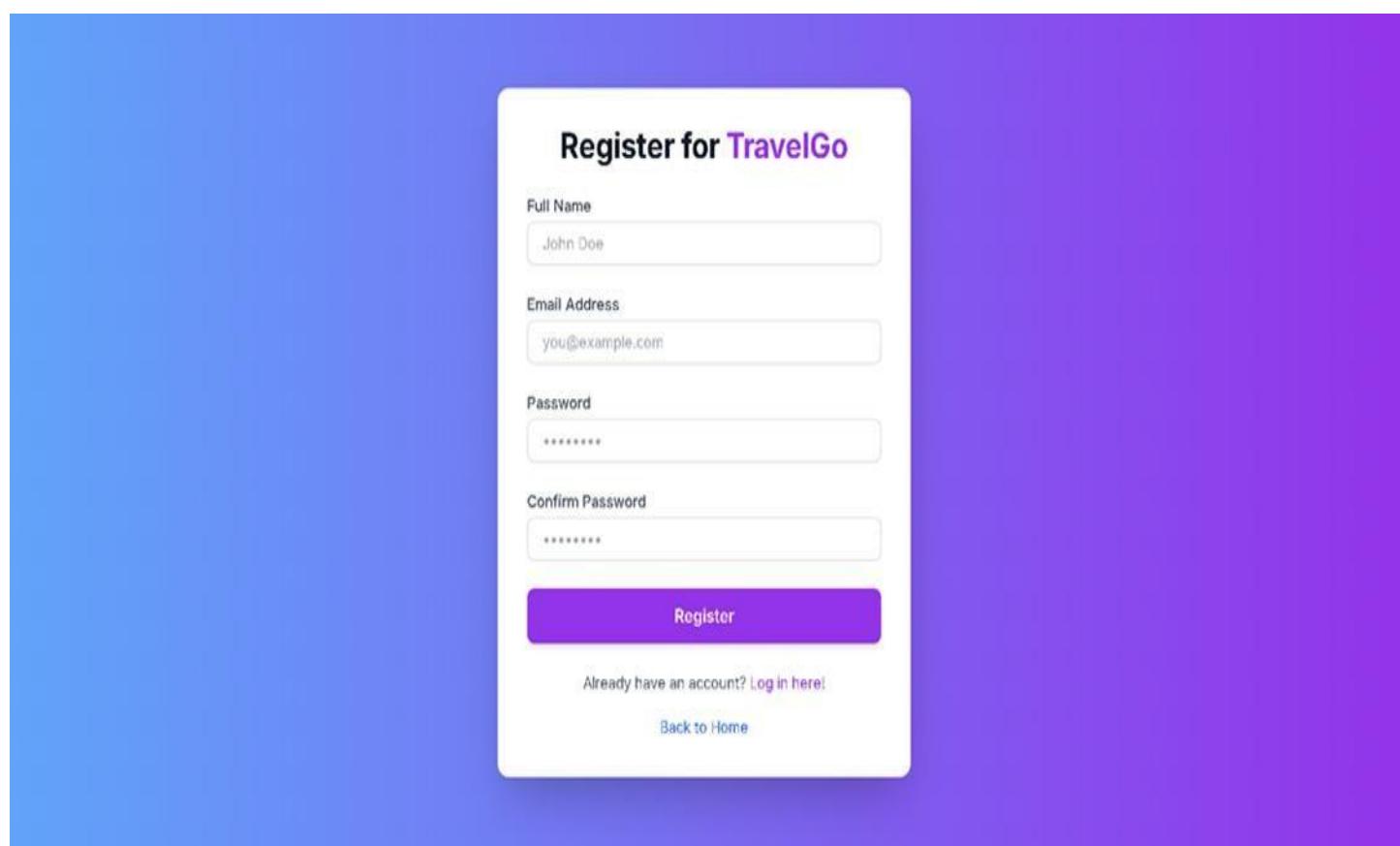
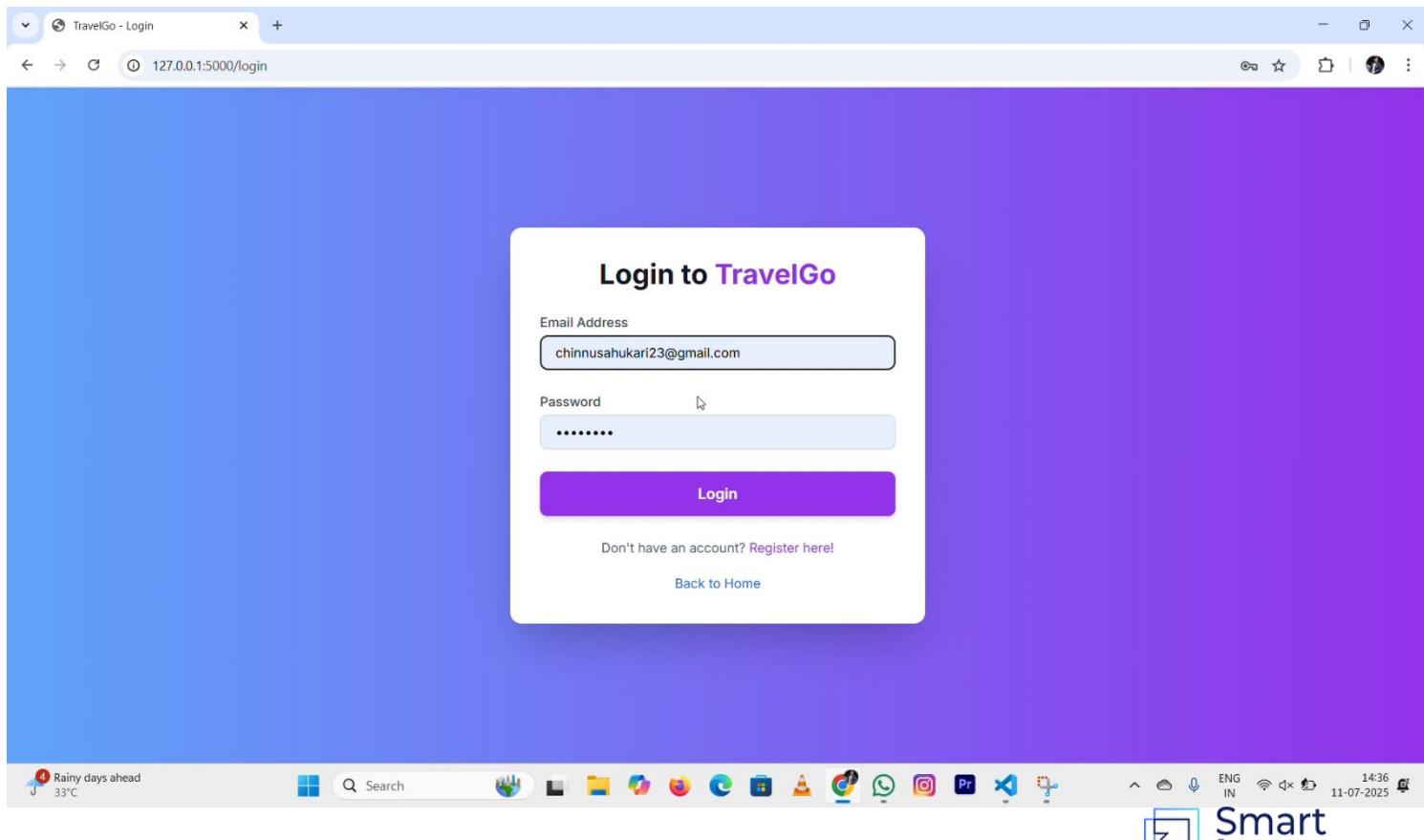
**Bus Booking Page:**

**Train Booking page:**

**Flight Booking Page:**

**Hotel Booking Page:**





Dashboard-TravelGo

127.0.0.1:5000/dashboard

TravelGo

Welcome,Chinnu sahukari

Bus Train Flight Hotel

YOUR BOOKINGS

**Bus Booking**

<b>Booking Date:</b> 2025-07-11T14:36:58.273937	<b>Travel Date:</b> 2025-07-11	<b>Bus Name:</b> Orange Travels	<b>Route:</b> Hyderabad to Vijayawada
<b>Time:</b> 08:00 AM	<b>Persons:</b> 1	<b>Seats:</b> A5	<b>Total Price:</b> ₹800.0
<b>Status:</b> Confirmed	Cancel		

Rainy days ahead  
33°C

Search

ENG IN

14:37 11-07-2025

### Booking Summary

Type: Bus (AC Sleeper)

Route: Hyderabad to Vijayawada

Travel Date: 2025-07-09

Departure/Time: 08:00 AM

Price per Person: ₹800.00

Number of Passengers: 1

Please select 1 seat(s).

Available Selected Booked

Driver

Door

A	1	2	3	4	5	6
B	1	2	3	4	5	6
C	1	2	3	4	5	6
D	1	2	3	4	5	6
E	1	2	3	4	5	6

**Proceed to Confirm Booking**

### Confirm Your Bus Booking

Bus Name: **Orange Travels**  
Route: **Hyderabad → Vijayawada**  
Departure Time: **08:00 AM**  
Bus Type: **AC Sleeper**  
Travel Date: **2025-07-09**  
Number of Persons: **1**  
Price per Person: **₹800.0**

**Total Price:** **₹800.0**

**Confirm Booking**

**Go Back**

### 🚂 Search & Book Trains

Hyderabad ▾ Vijayawada ▾ 09-07-2025 ▾ 1 ▾ **Search**

Express  Superfast  AC  Sleeper

Sort by: **None**

**Charminar Express (12759)**  
Express Sleeper • 18:30 - 00:30 • ₹300.00

**Book**

## Select Your Seats

### Booking Summary

Type: Train (Train)

Route: Hyderabad to Vijayawada

Travel Date: 2025-07-09

Departure/Time: 18:30 - 00:30

Price per Person: ₹300.00

Number of Passengers: 1

Please select 1 seat(s).

Available  Selected  Booked

Driver

Door

A

1

2

3

4

5

6

### Search & Book Flights

Hyderabad (HYD)

Mumbai (BOM)

09-07-2025

1

Search

Indigo

Vistara

Air India

Direct

1 Stop

Sort by:

None

Indigo (6E 234)

Direct Economy • 08:00 - 09:30 • Direct • ₹3500.00

Book

### Find & Book Hotel Rooms

Hyderabad      09-07-2025      10-07-2025      1

**Search**

5-Star     4-Star     3-Star     WiFi     Pool     Parking

Sort by: **None**

#### Taj Falaknuma Palace

Hyderabad • 5-Star • ₹25000.00/night

Amenities: WiFi, Pool, Parking, Restaurant

**Book**

**Total for 1 nights, 1 rooms: ₹25000.00**

#### Novotel Hyderabad Convention Centre

Hyderabad • 4-Star • ₹7000.00/night

Amenities: WiFi, Pool, Parking, Gym

**Book**

**Total for 1 nights, 1 rooms: ₹7000.00**

## 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.

# THANK YOU