

# Travel Go: A Cloud-Based Instant Travel Reservation Platform

## Project Overview:

Travel Go aims to streamline the travel planning process by offering a centralized platform to book buses, trains, flights, and hotels. Catering to the growing need for real-time and convenient travel solutions, it utilizes cloud infrastructure. The application employs Flask for backend services, AWS EC2 for scalable hosting, DynamoDB for fast data handling, and AWS SNS for immediate notifications. Users can sign up, log in, search, and book transport and accommodations. Additionally, they can review booking history and pick seats interactively, ensuring a user-friendly and responsive system.

## Use Case Scenarios:

### Scenario 1: Instant Travel Booking Experience

After logging into Travel Go, the user picks a travel type (bus/train/flight/hotel) and fills in preferences. Flask manages backend operations by retrieving matching options from DynamoDB. Once a booking is confirmed, AWS EC2 ensures swift performance even under high usage. This real-time setup allows users to secure bookings efficiently, even during rush periods.

### Scenario 2: Instant Email Alerts

Upon booking confirmation, Travel Go uses AWS SNS to instantly send emails with booking info. For example, when a flight is booked, Flask handles the transaction and SNS notifies the user via email. The booking details are saved securely in DynamoDB. This smooth integration boosts reliability and enhances user confidence.

### Scenario 3: Easy Booking Access and Control

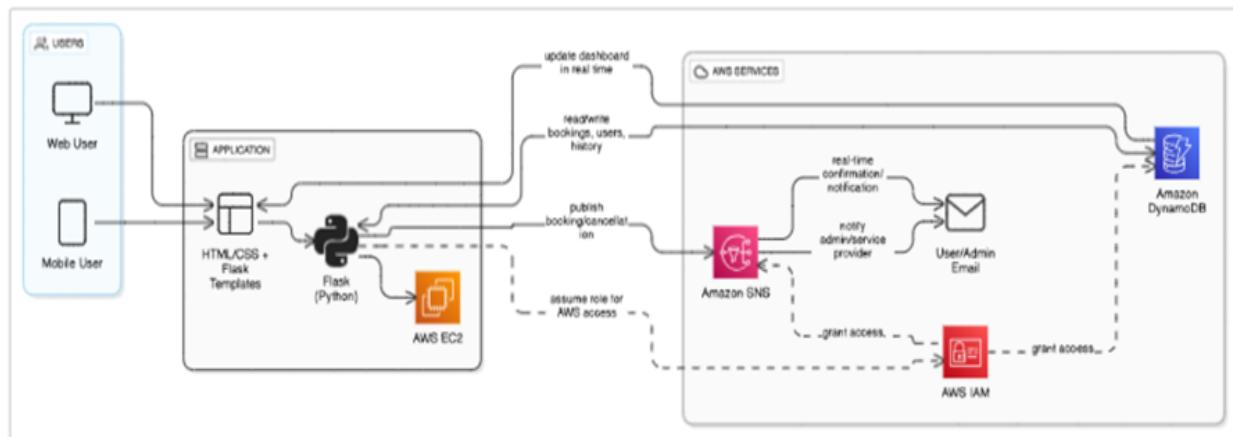
Users can return anytime to manage previous bookings. For instance, a user checks hotel reservations made last week. Flask quickly fetches this from DynamoDB. Thanks to its cloud-first approach, Travel Go provides uninterrupted access, letting users easily cancel or make new plans. EC2 hosting guarantees consistent performance across multiple users.

## AWS Architecture Diagram

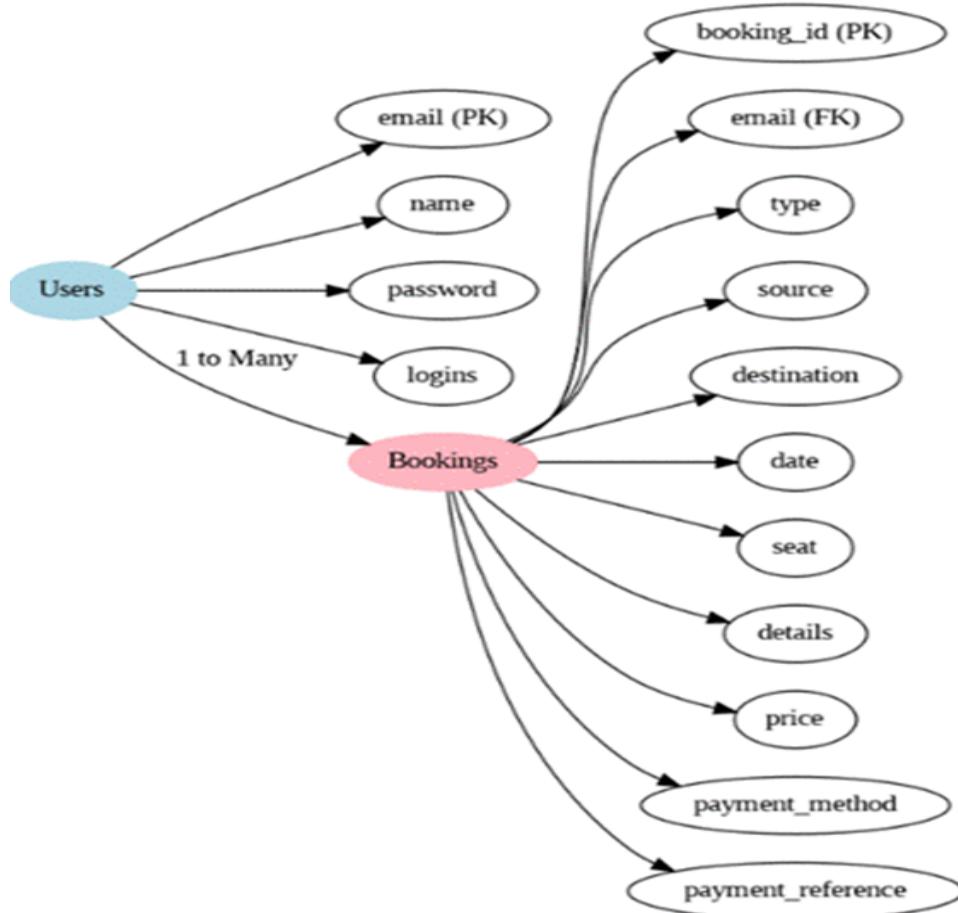
To build a scalable and responsive travel booking platform, a well-structured cloud architecture is essential. Travel Go leverages core AWS services such as EC2, DynamoDB, IAM, and SNS to ensure high availability and fault tolerance. The architecture is designed to support real-time booking, seamless data flow, and secure

operations. Below is the visual representation of the AWS setup used in the project.

## AWS Architecture



## Entity Relationship Diagram



## **Requirements:**

1. AWS Account Configuration
2. IAM Basics Overview
3. EC2 Introduction
4. DynamoDB Fundamentals
5. SNS Concepts
6. Git Version Control

## **Development Steps:**

### **1. AWS Account Setup and Login**

- Activity 1.1: Register for an AWS account if you haven't already.
- Activity 1.2: Access the AWS Management Console using your credentials.

### **2. DynamoDB Database Initialization**

- Activity 2.1: Create a DynamoDB table.
- Activity 2.2: Define attributes for users and travel bookings.

### **3. SNS Notification Configuration**

- Activity 3.1: Create SNS topics to notify users upon booking.
- Activity 3.2: Add user and provider emails as subscribers for updates.

### **4. Backend Development using Flask**

- Activity 4.1: Build backend logic with Flask.
- Activity 4.2: Connect AWS services using the boto3 SDK.

### **5. IAM Role Configuration**

- Activity 5.1: Set up IAM roles with specific permissions.
- Activity 5.2: Assign relevant policies to each role.

### **6. EC2 Instance Launch**

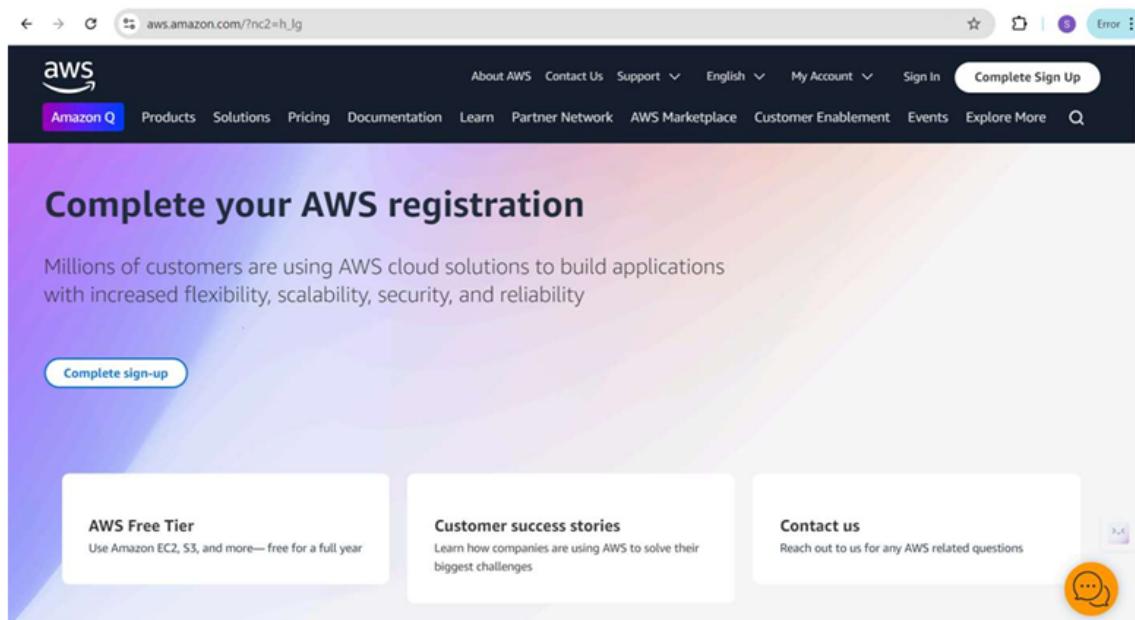
- Activity 6.1: Start an EC2 instance to host the backend.
- Activity 6.2: Set up security rules to allow HTTP and SSH traffic.

## 7. Flask Application Deployment

- Activity 7.1: Upload your Flask files to the EC2 instance.
- Activity 7.2: Launch your Flask server from the instance.

## 8. Testing and Launch

### 1. AWS Account Setup and Login



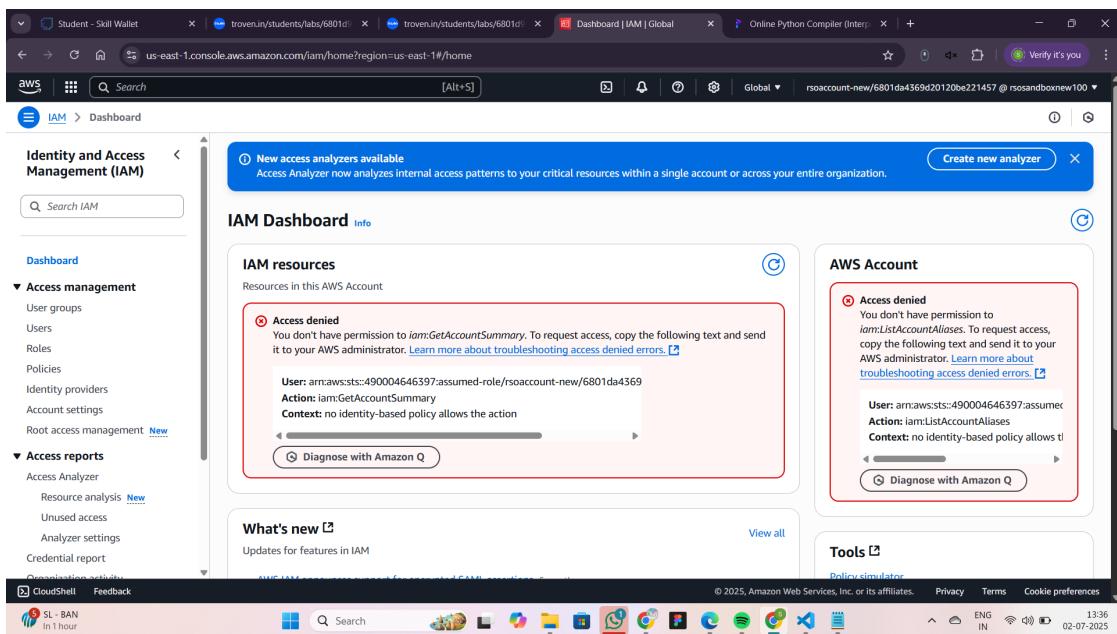
### 2. Log in to the AWS Management Console

The screenshot shows the AWS sign-in page. It features the AWS logo at the top, followed by a 'Sign in' section. Inside, there are two radio button options: 'Root user' (selected) and 'IAM user'. The 'Root user' option is described as an account owner performing tasks requiring unrestricted access. Below this is a 'Root user email address' input field containing 'username@example.com' and a 'Next' button. At the bottom, a note states: 'By continuing, you agree to the AWS Customer Agreement or other agreement for AWS services, and the Privacy Notice. This site uses essential cookies. See our Cookie Notice for more information.' To the right of the sign-in form is a dark sidebar with a purple-to-black gradient. It features the heading 'AI Use Case Explorer' and the subtext 'Discover AI use cases, customer success stories, and expert-curated implementation plans'. At the bottom of the sidebar is a 'Explore now >' button.

## After logging into the AWS Console:

Once you're logged in, you can access a wide range of AWS services from the dashboard. Use the search bar at the top to quickly navigate to services like EC2, DynamoDB, SNS, or IAM as needed for your project setup.

### 3. IAM Roles Creation



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

**Service or use case**

EC2

**Choose a use case for the specified service.**

**EC2**  
Allows EC2 instances to call AWS services on your behalf.

**EC2 Role for AWS Systems Manager**  
Allows EC2 instances to call AWS services like CloudWatch and Systems Manager on your behalf.

**EC2 Spot Fleet Role**  
Allows EC2 Spot Fleet to request and terminate Spot Instances on your behalf.

**EC2 - Spot Fleet Auto Scaling**  
Allows Auto Scaling to access and update EC2 spot fleets on your behalf.

**EC2 - Spot Fleet Tagging**  
Allows EC2 to launch spot instances and attach tags to the launched instances on your behalf.

**EC2 - Spot Instances**  
Allows EC2 Spot Instances to launch and manage spot instances on your behalf.

**EC2 - Spot Fleet**  
Allows EC2 Spot Fleet to launch and manage spot fleet instances on your behalf.

**EC2 - Scheduled Instances**  
Allows EC2 Scheduled instances to manage instances on your behalf.

**Cancel** **Next**

Before assigning permissions, ensure that the IAM role is created and ready to be attached with appropriate AWS-managed policies for each required service.

**Add permissions** Info

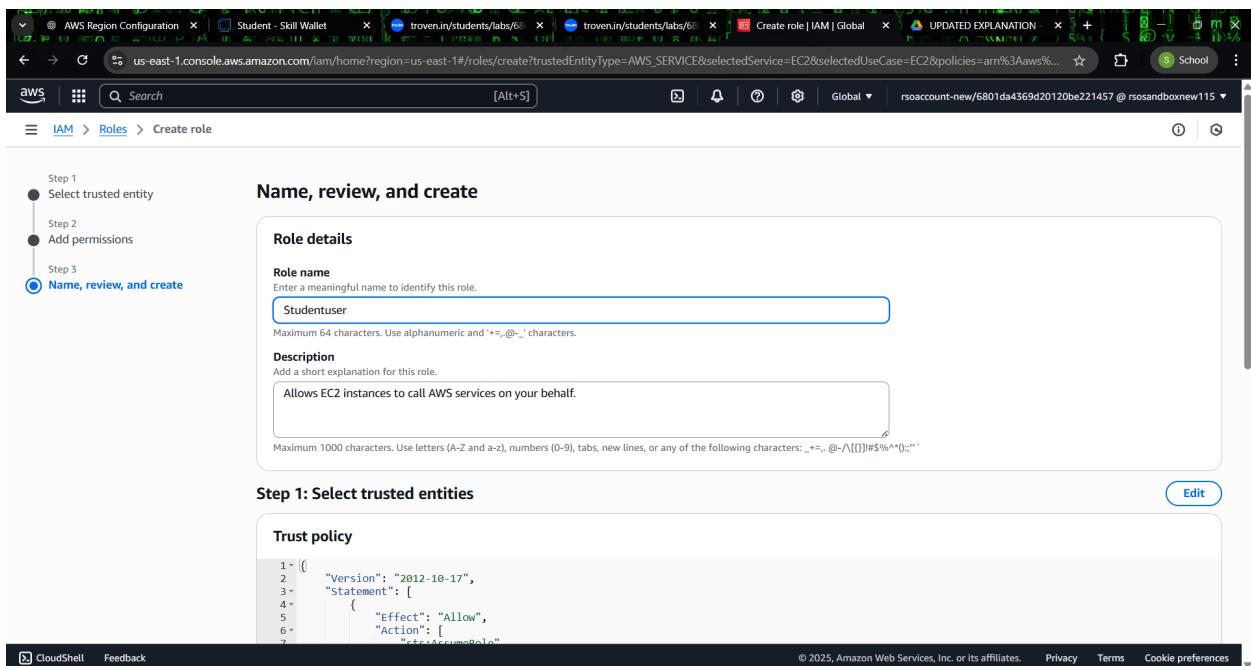
Permissions policies (3/1060) Info

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

**Filter by Type**

Policy name	Type	Description
AmazonEC2ContainerRegistryFullAccess	AWS managed	Provides administrative access to Amazon E...
AmazonEC2ContainerRegistryPowerUser	AWS managed	Provides full access to Amazon EC2 Co...
AmazonEC2ContainerRegistryPullOnly	AWS managed	Provides access to pull images from A...
AmazonEC2ContainerRegistryReadOnly	AWS managed	Provides read-only access to Amazon E...
AmazonEC2ContainerServiceAutoscaleRole	AWS managed	Policy to enable Task Autoscaling for A...
AmazonEC2ContainerServiceEventsRole	AWS managed	Policy to enable CloudWatch Events fo...
AmazonEC2ContainerServiceforEC2Role	AWS managed	Default policy for the Amazon EC2 Rol...
AmazonEC2ContainerService	AWS managed	Default policy for Amazon ECS service ...
<input checked="" type="checkbox"/> <b>AmazonEC2FullAccess</b>	AWS managed	Provides full access to Amazon EC2 via...
AmazonEC2ReadonlyAccess	AWS managed	Provides read only access to Amazon E...

**CloudShell** **Feedback**



## 4. DynamoDB Database Creation and Setup

Familiarize yourself with the DynamoDB service interface. DynamoDB offers a fast and flexible NoSQL database ideal for serverless applications like Travel Go. You will now proceed to create tables to store user and booking data. In this setup, you'll define partition keys to identify records uniquely—such as using “Email” for users or “Train Number” for trains. Properly configuring your attributes is crucial to ensure efficient querying and data retrieval. DynamoDB’s schema-less design allows flexibility while maintaining performance, making it well-suited for handling diverse booking data types in Travel Go.

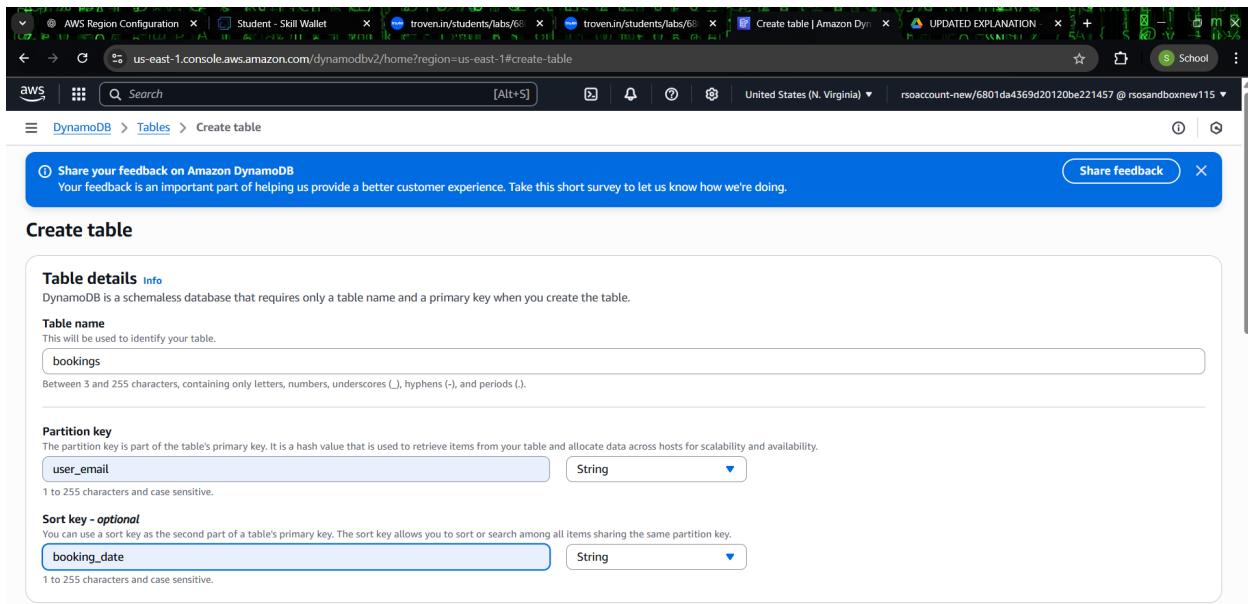
- In the AWS Console, navigate to DynamoDB and click on create tables

## Create a DynamoDB table for storing registration details and book Requests

1. Create Users table with partition key "Email" with type String and click on create tables.

**Repeat the same procedure to create additional tables:**

- **Train Table:** Use “Train Number” as the partition key to uniquely identify each train.
- **Bookings Table:** Set “User Email” as the partition key and “Booking Date” as the sort key to efficiently organize and retrieve individual user booking records based on date.



The screenshot shows the 'Create table' step in the AWS DynamoDB console. The table name is 'bookings'. The primary key is 'user\_email' of type String. A sort key is optional, set to 'booking\_date' of type String. The table is being created in the us-east-1 region.

**Table details**

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.

1 to 255 characters and case sensitive.

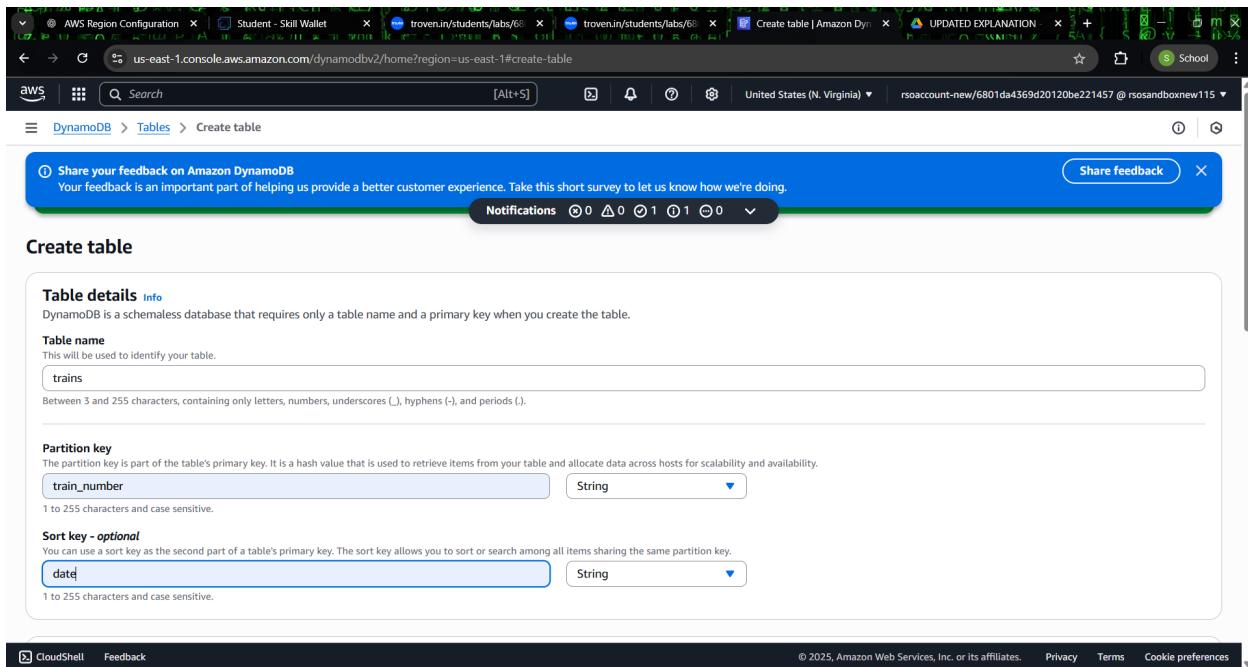
**Sort key - optional**

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.

1 to 255 characters and case sensitive.

**Table settings**

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

The screenshot shows the 'Create table' step in the AWS DynamoDB console. The table name is 'trains'. The primary key is 'train\_number' of type String. A sort key is optional, set to 'date' of type String. The table is being created in the us-east-1 region.

**Table details**

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.

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.

1 to 255 characters and case sensitive.

**Table settings**

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

The screenshot shows the AWS DynamoDB console. On the left, there's a navigation sidebar 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 section for 'DAX' with 'Clusters', 'Subnet groups', 'Parameter groups', and 'Events'. The main area is titled 'Tables (3) info' and lists three tables: 'bookings' (Active, user\_email (\$), booking\_date (\$), 0, 0, Off, On-di), 'trains' (Active, train\_number (\$), date (\$), 0, 0, Off, On-di), and 'travelgo\_users' (Active, email (\$), -, 0, 0, Off, On-di). At the bottom, there are links for 'CloudShell', 'Feedback', and copyright information.

## 5. SNS Notification Setup

### 1. Create SNS topics for sending email notifications to users.

The screenshot shows the AWS search results for 'sns'. The search bar at the top contains 'sns'. The results are categorized under 'Services' and 'Features'. Under 'Services', the 'Simple Notification Service' is listed as 'SNS managed message topics for Pub/Sub'. Under 'Features', there are sections for 'Events' (ElastiCache feature), 'SMS' (AWS End User Messaging feature), and 'Hosted zones' (Route 53 feature).

New Feature  
Amazon SNS now supports High Throughput FIFO topics. [Learn more](#)

Application Integration

## Amazon Simple Notification Service

Pub/sub messaging for microservices and serverless applications.

Amazon SNS is a highly available, durable, secure, fully managed pub/sub messaging service that enables you to decouple microservices, distributed systems, and event-driven serverless applications. Amazon SNS provides topics for high-throughput, push-based, many-to-many messaging.

**Benefits and features**

**Create topic**

**Topic name**  
A topic is a message channel. When you publish a message to a topic, it fans out the message to all subscribed endpoints.

**Next step**

[Start with an overview](#)

**Pricing**

Amazon SNS has no upfront costs. You pay based on the number of messages you publish, the number of messages you deliver, and any additional API calls for managing topics and

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

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

**Create topic**

**Details**

Type | Info  
Topic type cannot be modified after topic is created

FIFO (first-in, first-out)  
• Strictly-preserved message ordering  
• Exactly-once message delivery  
• Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints

Standard  
• Best-effort message ordering  
• At-least once message delivery  
• Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints

**Name**  
TravelId  
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (\_).

**Display name - optional** | Info  
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.  
My Topic  
Maximum 100 characters.

**Encryption - optional**  
Amazon SNS provides in-transit encryption by default. Enabling server-side encryption adds at-rest encryption to your topic.

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

2. Click on create topic

To begin configuring notifications, navigate to the SNS dashboard and click on "Create

Topic." Choose the topic type (Standard or FIFO) based on your requirements. Provide a meaningful name for the topic that reflects its purpose (e.g., booking-alerts). This topic will serve as the communication channel for sending notifications to subscribed users.

The screenshot shows the 'Create topic' configuration page for AWS SNS. It displays several optional policy sections:

- Access policy - optional** Info  
This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.
- Data protection policy - optional** Info  
This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.
- Delivery policy (HTTP/S) - optional** Info  
The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.
- Delivery status logging - optional** Info  
These settings configure the logging of message delivery status to CloudWatch Logs.
- Tags - optional**  
A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. [Learn more](#)
- Active tracing - optional** Info  
Use AWS X-Ray active tracing for this topic to view its traces and service map in Amazon CloudWatch. Additional costs apply.

At the bottom right, there are 'Cancel' and 'Create topic' buttons.

3. Configure the SNS topic and note down the **Topic ARN**.

4. Click on create subscription.

The screenshot shows the 'Create subscription' configuration page for AWS SNS. It displays two optional policy sections:

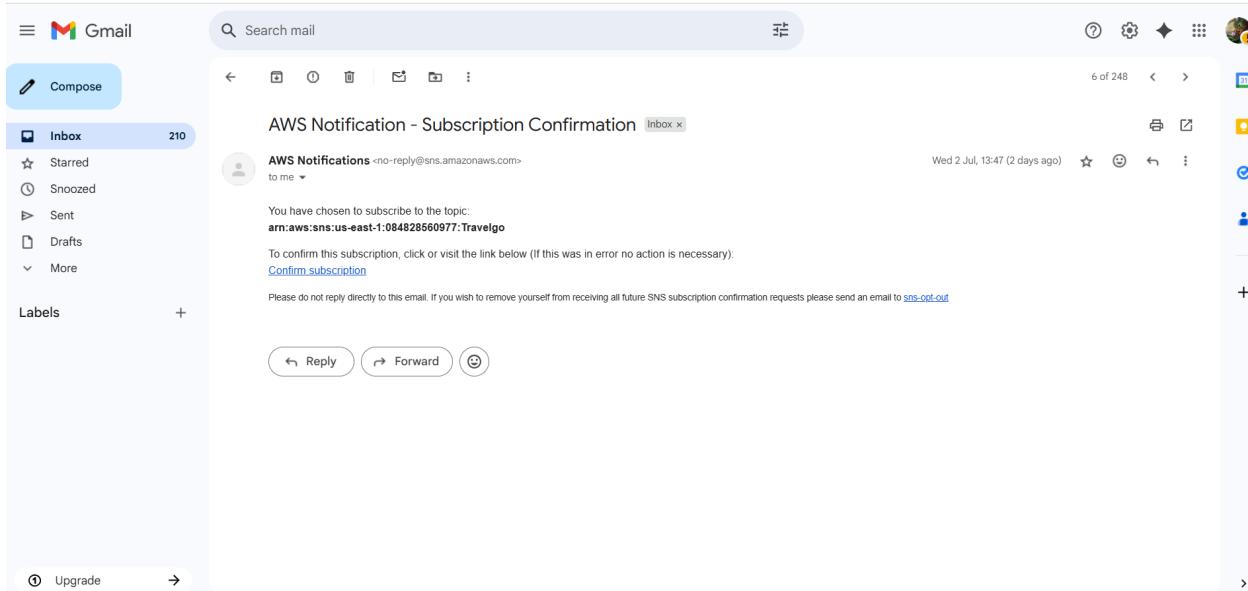
- Subscription filter policy - optional** Info  
This policy filters the messages that a subscriber receives.
- Redrive policy (dead-letter queue) - optional** Info  
Send undeliverable messages to a dead-letter queue.

At the bottom right, there are 'Cancel' and 'Create subscription' buttons.

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

Once the email subscription is confirmed, the endpoint becomes active for receiving notifications. This ensures that users will instantly get booking confirmations or alerts. You can manage or remove subscriptions anytime via the SNS dashboard. It's

recommended to test the setup by publishing a sample message to verify successful delivery.



## Backend Configuration and Coding

```
1  from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify
2  from pymongo import MongoClient
3  from werkzeug.security import generate_password_hash, check_password_hash
4  from datetime import datetime
5  from bson.objectid import ObjectId
6  from bson.errors import InvalidId
```

- Flask: Initiates web application
- render\_template: Loads Jinja2 HTML templates
- request: Collects input from forms/API
- redirect/url\_for: Page redirection logic
- session: Keeps user-specific info
- jsonify: Returns data in JSON structure

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

Begin building the web application by initializing the Flask app instance using `Flask(__name__)`, which sets up the core of your backend framework.

```
18     users_table = dynamodb.Table('travelgo_users')
19     trains_table = dynamodb.Table('trains') # Note: This table is declared but not used in the provided routes.
20     bookings_table = dynamodb.Table('bookings')
```

## SNS connection:

This function sends alerts using AWS SNS by publishing a subject and message to a predefined topic. It uses sns\_client.publish() with error handling to catch failures and print debug info. This ensures users receive real-time booking notifications.

```
22     SNS_TOPIC_ARN = 'arn:aws:sns:us-east-1:490004646397:Travelgo:372db6a7-7131-4571-8397-28b62c9e08a8'
23
24     # Function to send SNS notifications
25     # This function is duplicated in the original code, removing the duplicate.
26     def send_sns_notification(subject, message):
27         try:
28             sns_client.publish(
29                 TopicArn=SNS_TOPIC_ARN,
30                 Subject=subject,
31                 Message=message
32             )
33         except Exception as e:
34             print(f"SNS Error: Could not send notification - {e}")
35             # Optionally, flash an error message to the user or log it more robustly.
```

## Routes:

### Route Overview of TravelGo:

The application begins with user onboarding through the Register and Login routes, securing user access. Once authenticated, users are redirected to the Dashboard, which serves as the central hub. From there, they can explore and book through Bus, Train, Flight, and Hotel modules. Each booking passes through a Confirmation step where details are reviewed before final submission. The Final Confirmation route stores the booking and triggers email alerts. Users also have the option to manage or cancel their reservations via the Cancel Booking route, ensuring full control and flexibility.

Let's take a closer look at the backend code that powers these application routes.

```
38 @app.route('/')
39 def index():
40     return render_template('index.html')
41
42 @app.route('/register', methods=['GET', 'POST'])
43 def register():
44     if request.method == 'POST':
45         email = request.form['email']
46         password = request.form['password']
47
48         # Check if user already exists
49         # This uses get_item on the primary key 'email', so no GSI needed.
50         existing = users_table.get_item(Key={'email': email})
51         if 'Item' in existing:
52             flash('Email already exists!', 'error')
53             return render_template('register.html')
54
55         # Hash password and store user
56         hashed_password = generate_password_hash(password)
57         users_table.put_item(Item={'email': email, 'password': hashed_password})
58         flash('Registration successful! Please log in.', 'success')
59         return redirect(url_for('login'))
60     return render_template('register.html')
```

```
62 @app.route('/login', methods=['GET', 'POST'])
63 def login():
64     if request.method == 'POST':
65         email = request.form['email']
66         password = request.form['password']
67
68         # Retrieve user by email (primary key)
69         user = users_table.get_item(Key={'email': email})
70
71         # Authenticate user
72         if 'Item' in user and check_password_hash(user['Item']['password'], password):
73             session['email'] = email
74             flash('Logged in successfully!', 'success')
75             return redirect(url_for('dashboard'))
76         else:
77             flash('Invalid email or password!', 'error')
78             return render_template('login.html')
79     return render_template('login.html')
```

```
81     @app.route('/logout')
82     def logout():
83         session.pop('email', None)
84         flash('You have been logged out.', 'info')
85         return redirect(url_for('index'))
86
87     @app.route('/dashboard')
88     def dashboard():
89         if 'email' not in session:
90             return redirect(url_for('login'))
91         user_email = session['email']
92
93         # Query bookings for the logged-in user using the primary key 'user_email'
94         # No GSI is needed here as 'user_email' is likely the partition key for the bookings_table.
95         response = bookings_table.query(
96             KeyConditionExpression=Key('user_email').eq(user_email),
97             ScanIndexForward=False # Get most recent bookings first
98         )
99         bookings = response.get('Items', [])
100
101        # Convert Decimal types from DynamoDB to float for display if necessary
102        for booking in bookings:
103            if 'total_price' in booking:
104                try:
105                    booking['total_price'] = float(booking['total_price'])
106                except (TypeError, ValueError):
107                    booking['total_price'] = 0.0 # Default value if conversion fails
108        return render_template('dashboard.html', username=user_email, bookings=bookings)
```

```
110     @app.route('/train')
111     def train():
112         if 'email' not in session:
113             return redirect(url_for('login'))
114         return render_template('train.html')
```

```

116 @app.route('/confirm_train_details')
117 def confirm_train_details():
118     if 'email' not in session:
119         return redirect(url_for('login'))
120
121     booking_details = {
122         'name': request.args.get('name'),
123         'train_number': request.args.get('trainNumber'),
124         'source': request.args.get('source'),
125         'destination': request.args.get('destination'),
126         'departure_time': request.args.get('departureTime'),
127         'arrival_time': request.args.get('arrivalTime'),
128         'price_per_person': Decimal(request.args.get('price')),
129         'travel_date': request.args.get('date'),
130         'num_persons': int(request.args.get('persons')),
131         'item_id': request.args.get('trainId'), # This is the train ID
132         'booking_type': 'train',
133         'user_email': session['email'],
134         'total_price': Decimal(request.args.get('price')) * int(request.args.get('persons'))
135     }

```

```

492 @app.route('/cancel_booking', methods=['POST'])
493 def cancel_booking():
494     if 'email' not in session:
495         return redirect(url_for('login'))
496
497     booking_id = request.form.get('booking_id')
498     user_email = session['email']
499     booking_date = request.form.get('booking_date') # This is crucial as it's the sort key
500
501     if not booking_id or not booking_date:
502         flash("Error: Booking ID or Booking Date is missing for cancellation.", 'error')
503         return redirect(url_for('dashboard'))
504
505     try:
506         # Delete item using the primary key (user_email and booking_date)
507         # This does not use GSI, so it remains unchanged.
508         bookings_table.delete_item(
509             Key={'user_email': user_email, 'booking_date': booking_date}
510         )
511         flash(f"Booking {booking_id} cancelled successfully!", 'success')
512     except Exception as e:
513         flash(f"Failed to cancel booking {booking_id}: {str(e)}", 'error')
514
515     return redirect(url_for('dashboard'))

```

**Note:** The implementation logic for train booking, train details, and cancellation is consistent across other modules such as bus, flight, and hotel. Each follows a similar structure for handling user input, storing booking data, and managing cancellations.

using the same backend principles. This modular approach promotes code reusability and simplifies maintenance across the application. Minor adjustments are made in each module to accommodate specific fields like transport type or room preferences. Overall, the core flow—search, select, book, and cancel—remains consistent for all services.

## Deployment code:

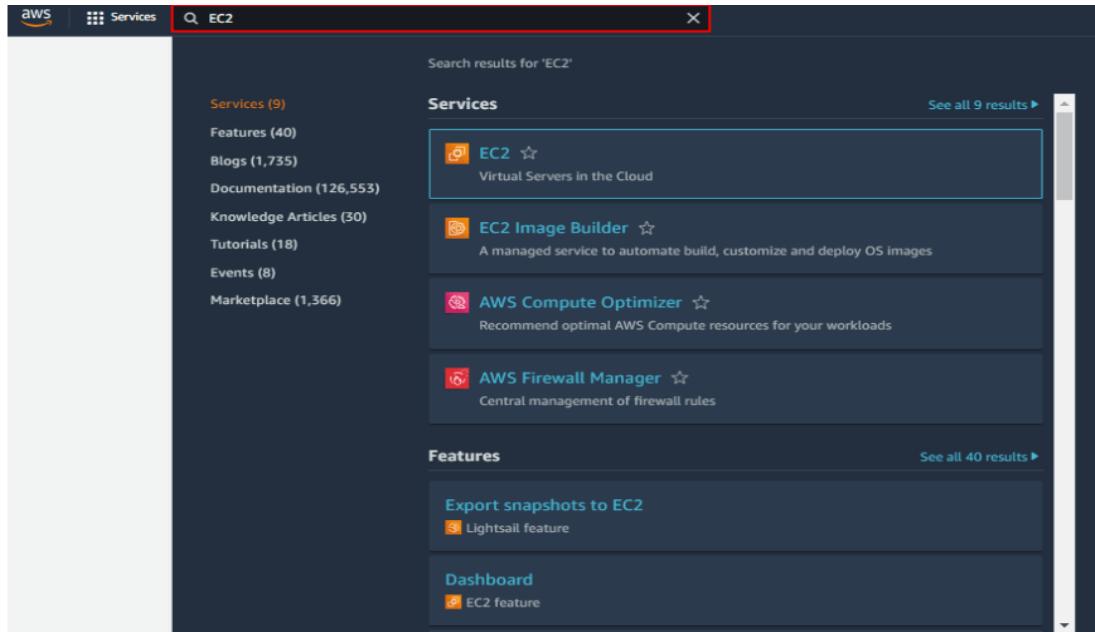
```
518     if __name__ == '__main__':
519         # IMPORTANT: In a production environment, disable debug mode and specify a production-ready host.
520         app.run(debug=True, host='0.0.0.0')
```

- Start the Flask server by configuring it to run on all network interfaces (0.0.0.0) at port 5000, with debug mode enabled to support development and live testing.

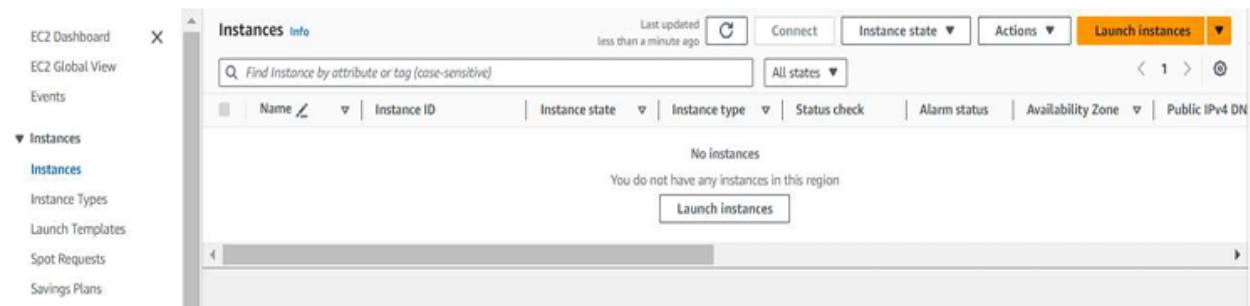
## EC2 Instance Setup:

Rohitha2527 Update app.py with latest changes		5344d4a · 2 days ago	11 Commits
static	Resolve conflicts and force push final TravelGo project	4 days ago	
templates	Resolve conflicts and force push final TravelGo project	4 days ago	
venv	Resolve conflicts and force push final TravelGo project	4 days ago	
venv_new	Resolve conflicts and force push final TravelGo project	4 days ago	
README.md	First commit	4 days ago	
app.py	Update app.py with latest changes	2 days ago	

Launch an EC2 instance to host the Flask application.

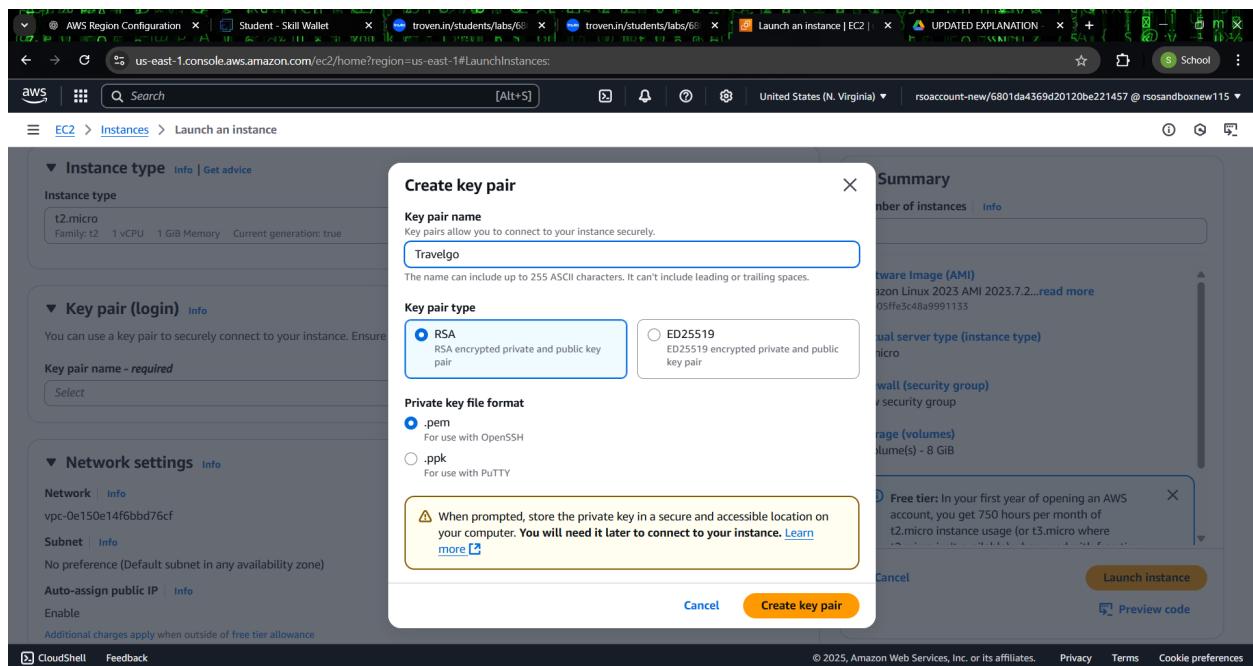


## 1. Click on launch Instance:

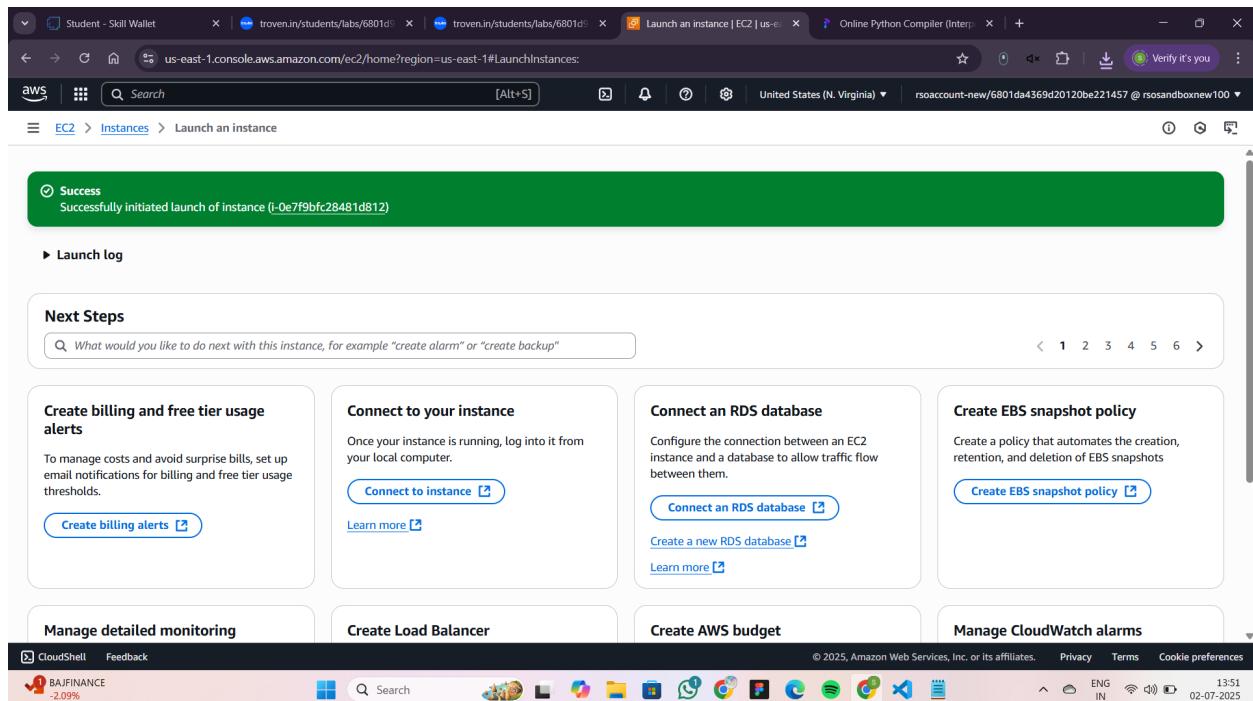


Once the EC2 instance is launched, it acts as the server backbone for your application. Naming the instance as Travelgoproject helps in easy identification during future scaling or monitoring. You can manage performance, logs, and connectivity from the EC2 dashboard. This instance will host and run your Flask backend reliably.

>Create a key pair named Travelgo to securely connect to your EC2 instance via SSH. Download and store the .pem file safely, as it will be required for future logins. While setting up the firewall, configure the security group to allow inbound traffic on ports 22 (SSH) and 5000 (Flask). This ensures both secure access and proper functioning of the web application. It's recommended to restrict SSH access to your specific IP range for added security. The Flask port (5000) should be open to all only during development—limit access in production. Properly configured security groups help prevent unauthorized access while keeping your app responsive and accessible.



Wait for the confirmation message like “Successfully initiated launch of instance i-0e7f9bfc28481d812,” indicating the instance is being provisioned. During this process, create a key pair named Travelgo (RSA type) with .pem file format. Save this private key securely, as it will be needed for future SSH access.



## Edit Inbound Rules:

Select the EC2 instance you just launched and ensure it's in the "running" state. Navigate to the "Security" tab, then click "Edit inbound rules." Add a new rule with the following settings: Type – Custom TCP, Protocol – TCP, Port Range – 5000, Source – Anywhere (IPv4) 0.0.0.0/0. This allows external access to your Flask application running on port 5000.

The screenshot shows the AWS EC2 Security Groups interface under the 'Edit inbound rules' section. The table lists four existing rules:

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-01084cb5c8716b9df	SSH	TCP	22	Custom	0.0.0.0/0
sgr-0ce5f84230320a5a2	HTTP	TCP	80	Custom	0.0.0.0/0
sgr-0bfd52d31ccca377e	HTTPS	TCP	443	Custom	0.0.0.0/0

A new rule is being added at the bottom:

-	Custom TCP	TCP	5000	Anywh...	0.0.0.0/0
---	------------	-----	------	----------	-----------

Below the table is a note: "⚠ 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."

At the bottom right are buttons for 'Cancel', 'Preview changes', and 'Save rules'.

Inbound security group rules successfully modified on security group (sg-069e3954cb2b6f875 | launch-wizard-1)

**sg-069e3954cb2b6f875 - launch-wizard-1**

**Details**

Security group name launch-wizard-1	Security group ID sg-069e3954cb2b6f875	Description launch-wizard-1 created 2025-07-02T08:18:46.654Z	VPC ID vpc-0e150e14f6bbd76cf
Owner 084828560977	Inbound rules count 4 Permission entries	Outbound rules count 1 Permission entry	

**Inbound rules (4)**

Name	Security group rule ID	IP version	Type	Protocol	Port range
-	sgr-01084cb5c8716b9df	IPv4	SSH	TCP	22
-	sgr-0aff0e76f524cd65	IPv4	Custom TCP	TCP	5000
-	sgr-0ce5f84230320a5a2	IPv4	HTTP	TCP	80

## Modify IAM ROLE

Attach the IAM role Studentuser to your EC2 instance to grant secure access to AWS services like DynamoDB and SNS. This avoids hardcoding credentials and ensures your app functions with the required permissions. Make sure the role includes all necessary policies for smooth integration.

**Instances (1/1) Info**

**i-0e7f9bfc28481d812 (TravelGoproject)**

**Actions**

- Launch instances
- Instance diagnostics
- Instance settings
- Networking
- Security
- Image and templates
- Modify IAM role
- Monitor and troubleshoot

**Inbound rules**

The screenshot shows the 'Modify IAM role' page in the AWS IAM console. The 'Instance ID' dropdown is set to 'i-0426bada174a7ef93 (TravelGoproject)'. The 'IAM role' dropdown is set to 'Studentuser'. A green confirmation message at the top states 'Successfully attached Studentuser to instance i-0e7f9bfc28481d812'. Below this, the 'Instances (1/1)' section shows a table with one row: 'TravelGoproject' (Instance ID: i-0e7f9bfc28481d812, State: Running, Type: t2.micro). The details tab for the instance shows its public IP (18.205.24.220), private IP (172.31.82.161), and DNS name (ec2-18-205-24-220.compute-1.amazonaws.com).

Wait until the confirmation message appears indicating that the Studentuser role has been successfully attached to the instance. This confirms that all required permissions are now active. Once attached, proceed to connect to your EC2 instance and run your GitHub-hosted Flask application code.

The screenshot shows the AWS EC2 Connect interface for an instance with ID i-0e7f9bf28481d812. The 'SSH client' tab is selected. It provides instructions for connecting via SSH:

- Open an SSH client.
- Locate your private key file. The key used to launch this instance is Travelgo.pem.
- Run this command, if necessary, to ensure your key is not publicly viewable.  
chmod 400 "Travelgo.pem"
- Connect to your instance using its Public DNS:  
ec2-18-205-24-220.compute-1.amazonaws.com

Below these instructions is an example command:

```
ssh -i "Travelgo.pem" ec2-user@ec2-18-205-24-220.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."

The following are the terminal outputs after executing the commands shown below the image. These outputs indicate that the setup steps have been successfully carried out.

The commands summary will be provided at last.

The screenshot shows a Windows PowerShell window titled "Windows PowerShell". The output of the session is as follows:

```
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\Rohi> ssh -i "C:\Users\Rohi\Downloads\Travelgo (1).pem" ec2-user@ec2-3-95-225-23.compute-1.amazonaws.com
The authenticity of host 'ec2-3-95-225-23.compute-1.amazonaws.com (3.95.225.23)' can't be established.
ED25519 key fingerprint is SHA256:w/MQ9m0FT+9KP9NF+Xf2qCz31XBo3N2Rnf8oHwf58.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-95-225-23.compute-1.amazonaws.com' (ED25519) to the list of known hosts.

  _/\_ ###
  ~~~ \#####
  ~~~ \####
  ~~~ \#/
  ~~~ \#> https://aws.amazon.com/linux/amazon-linux-2023
  ~~~ /
  ~~~ /_/
  ~~~ /_/
  ~~~ /_/
Last login: Wed Jul 2 08:26:18 2025 from 18.206.107.28
[ec2-user@ip-172-31-94-13 ~]$ sudo yum install git -y
Amazon Linux 2023 Kernel Livepatch repository
Dependencies resolved.
=====
 Package           Architecture      Version       Repository      Size
=====
 Installing:
  git              x86_64          2.47.1-1.amzn2023.0.3    amazonlinux   52 k
 Installing dependencies:
  git-core          x86_64          2.47.1-1.amzn2023.0.3    amazonlinux   4.5 M
  git-core-doc      noarch         2.47.1-1.amzn2023.0.3    amazonlinux   2.8 M
  perl-Error        noarch         1:0.17029-5.amzn2023.0.2  amazonlinux   41 k
  perl-File-Find    noarch         1.37-477.amzn2023.0.7    amazonlinux   25 k
  perl-Git          noarch         2.47.1-1.amzn2023.0.3    amazonlinux   40 k
  perl-TermReadKey x86_64          2.38-9.amzn2023.0.2     amazonlinux   36 k
  perl-Lib          x86_64          0.65-477.amzn2023.0.7    amazonlinux   15 k

Transaction Summary
=====
```

```

Windows PowerShell

Installed:
git-2.47.1-1.amzn2023.0.3.x86_64
git-core-doc-2.47.1-1.amzn2023.0.3.noarch
perl-File-Find-1.37-477.amzn2023.0.7.noarch
perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64
perl-core-2.47.1-1.amzn2023.0.3.x86_64
perl-Error-1.0.17029-5.amzn2023.0.2.noarch
perl-Git-2.47.1-1.amzn2023.0.3.noarch
perl-lib-0.65-477.amzn2023.0.7.x86_64

Complete!
[ec2-user@ip-172-31-94-133 ~]$ git clone https://github.com/Rohitha2527/Travel-Go
Cloning into 'Travel-Go'...
remote: Enumerating objects: 3266, done.
remote: Counting objects: 100% (3266/3266), done.
remote: Compressing objects: 100% (2016/2016), done.
remote: Total 3266 (delta 1241), reused 3250 (delta 1231), pack-reused 0 (from 0)
Receiving objects: 100% (3266/3266), 11.06 MiB | 22.56 MiB/s, done.
Resolving deltas: 100% (1241/1241), done.
[ec2-user@ip-172-31-94-133 ~]$ cd Rohitha2527
-bash: cd: Rohitha2527: No such file or directory
[ec2-user@ip-172-31-94-133 ~]$ cd Travel-Go
[ec2-user@ip-172-31-94-133 Travel-Go]$
sudo yum install python3 -y
Last metadata expiration check: 0:05:06 ago on Wed Jul 2 08:34:01 2025.
Package python3-3.9.23-1.amzn2023.0.1.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-94-133 Travel-Go]$ sudo yum install python3-pip -y
Last metadata expiration check: 0:05:24 ago on Wed Jul 2 08:34:01 2025.
Dependencies resolved.
=====
Package           Arch      Version          Repository      Size
=====
Installing:
python3-pip      noarch   21.3.1-2.amzn2023.0.11    amazonlinux   1.8 M
Installing weak dependencies:
libcrypt-compat  x86_64   4.4.33-7.amzn2023.0.11.n  amazonlinux   92 k

Transaction Summary
=====
Install 2 Packages


```

- sudo yum install git -y
- git clone your\_repository\_url
- cd your\_project\_directory
- sudo yum install python3 -y
- sudo yum install python3-pip -y

```

Windows PowerShell

Total download size: 1.9 M
Installed size: 11 M
Downloading Packages:
(1/2): libcrypt-compat-4.4.33-7.amzn2023.x 2.5 MB/s | 92 kB     00:00
(2/2): python3-pip-21.3.1-2.amzn2023.0.11.n 31 MB/s | 1.8 MB   00:00
Total                                         20 MB/s | 1.9 MB  00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing : 1/1
Installing : libcrypt-compat-4.4.33-7.amzn2023.x86_64 1/2
Installing : python3-pip-21.3.1-2.amzn2023.0.11.noarch 2/2
Running scriptlet: python3-pip-21.3.1-2.amzn2023.0.11.noarch 2/2
Verifying  : libcrypt-compat-4.4.33-7.amzn2023.x86_64 1/2
Verifying  : python3-pip-21.3.1-2.amzn2023.0.11.noarch 2/2

Installed:
libcrypt-compat-4.4.33-7.amzn2023.x86_64
python3-pip-21.3.1-2.amzn2023.0.11.noarch

Complete!
[ec2-user@ip-172-31-94-133 Travel-Go]$ pip install flask
Defaulting to user installation because normal site-packages is not writeable
Collecting flask
  Downloading flask-3.1.1-py3-none-any.whl (103 kB)
Collecting itsdangerous>=2.2.0
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting markupsafe>=2.1.1
  Downloading MarkupSafe-2.0.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (20 kB)
Collecting click<8.1.3
  Downloading click-8.1.8-py3-none-any.whl (98 kB)
Collecting blinker>=1.9.0
  Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Collecting importlib-metadata>=3.6.0
  Downloading importlib_metadata-3.6.0-py3-none-any.whl (27 kB)
Collecting jinja2>=3.1.2
```

```
| Windows PowerShell      + - x
Collecting werkzeug>=3.1.0
  Downloading werkzeug-3.1.3-py3-none-any.whl (224 kB)
    |██████████| 224 kB 42.3 MB/s
Collecting zipp>=3.20
  Downloading zipp-3.23.0-py3-none-any.whl (10 kB)
Installing collected packages: zip, 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
[ec2-user@ip-172-31-94-133 Travel-Go]$ pip install boto3
Defaulting to user installation because normal site-packages is not writeable
Collecting boto3
  Downloading boto3-1.39.1-py3-none-any.whl (139 kB)
    |██████████| 139 kB 15.3 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 s3transfer<0.14.0,>=0.13.0
  Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB)
    |██████████| 85 kB 8.5 MB/s
Collecting botocore<1.40.0,>=1.39.1
  Downloading botocore-1.39.1-py3-none-any.whl (13.8 MB)
    |██████████| 13.8 MB 42.6 MB/s
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.1->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,>=1.39.1->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,>=1.39.1->boto3) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed boto3-1.39.1 botocore-1.39.1 s3transfer-0.13.0
[ec2-user@ip-172-31-94-133 Travel-Go]$ python3 app.py
-bash: python3 app.py: command not found
[ec2-user@ip-172-31-94-133 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://172.31.94.133:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 664-821-455
client_loop: send disconnect: Connection reset
PS C:\Users\Rohi> |
```

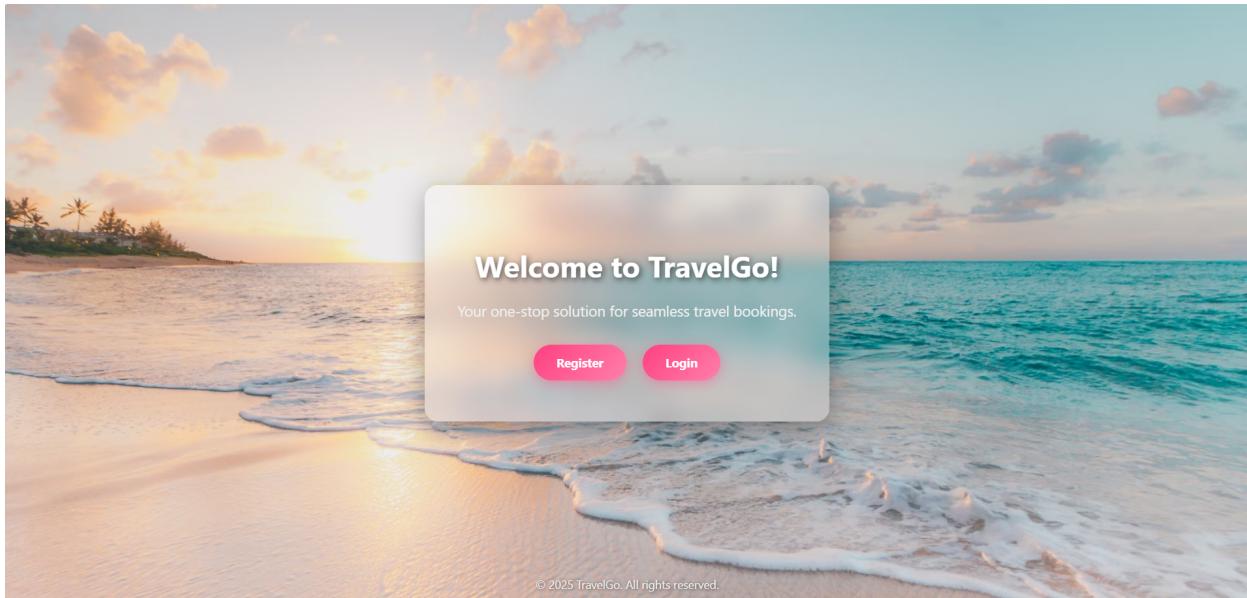
- pip install flask
- pip install boto3
- python3 app.py

## Deployment Steps Summary:

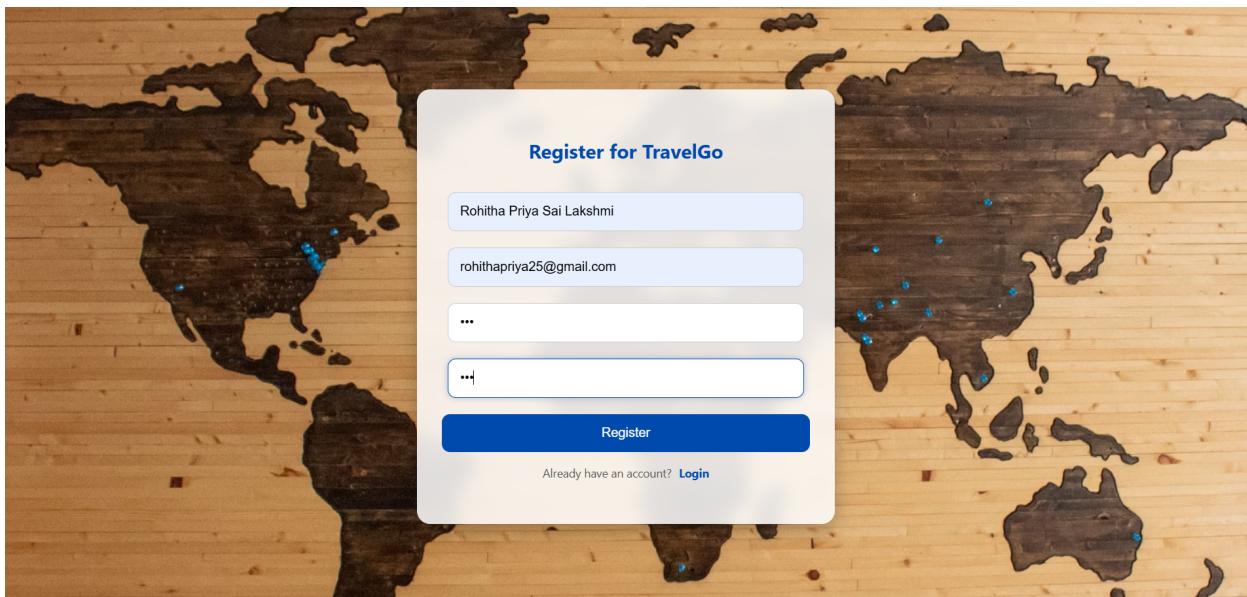
- sudo yum install git -y
- git clone your\_repository\_url
- cd your\_project\_directory
- sudo yum install python3 -y
- sudo yum install python3-pip -y
- pip install flask
- pip install boto3
- python3 app.py

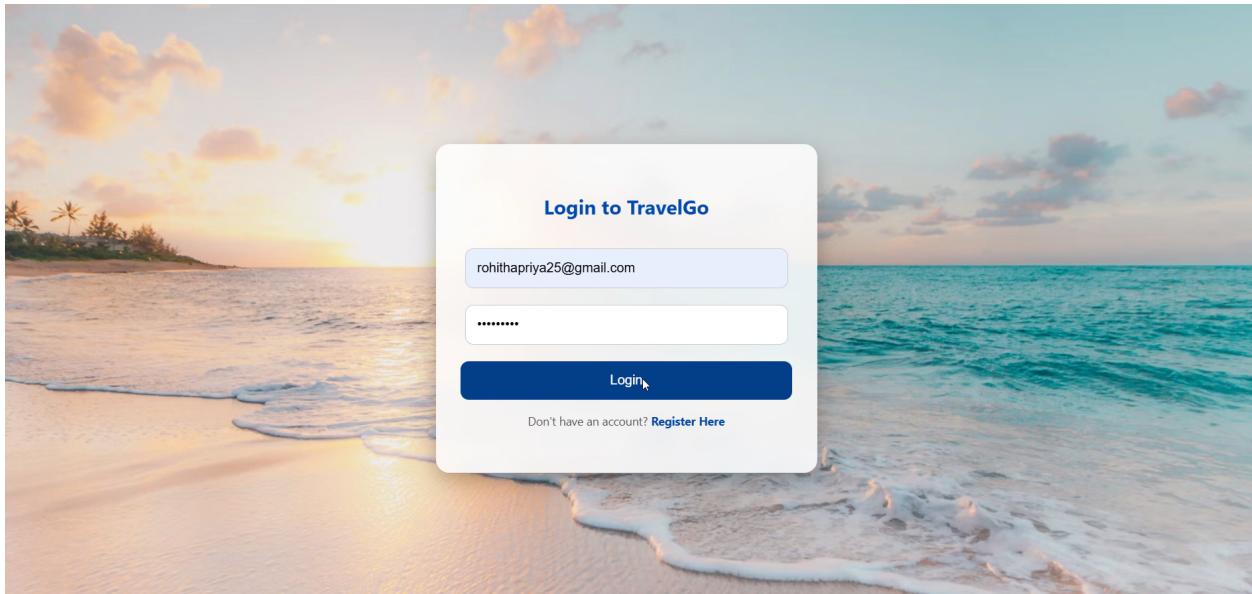
## User Interface Screens:

- Homepage:



- Register and Login Portals:





- Dashboard View:

The dashboard serves as the central hub for users to access all travel services in one place. It displays a personalized greeting along with quick navigation cards for Bus, Train, Flight, and Hotel bookings. A success alert confirms the user's login, enhancing the user experience. Below, recent and upcoming bookings are listed with full travel details and cancellation options. This intuitive layout ensures users can manage their journeys effortlessly and efficiently.

Lets have a look at my dashboard page!!!

Welcome, rohithapriya25@gmail.com!

Dashboard Logout

User rohithapriya25@gmail.com

Total Bookings	Upcoming	Completed	Cancelled
0	0	0	0

Bus Train Flight Hotel

All Bus Train Flight Hotel dd-mm-yyyy

Your Bookings

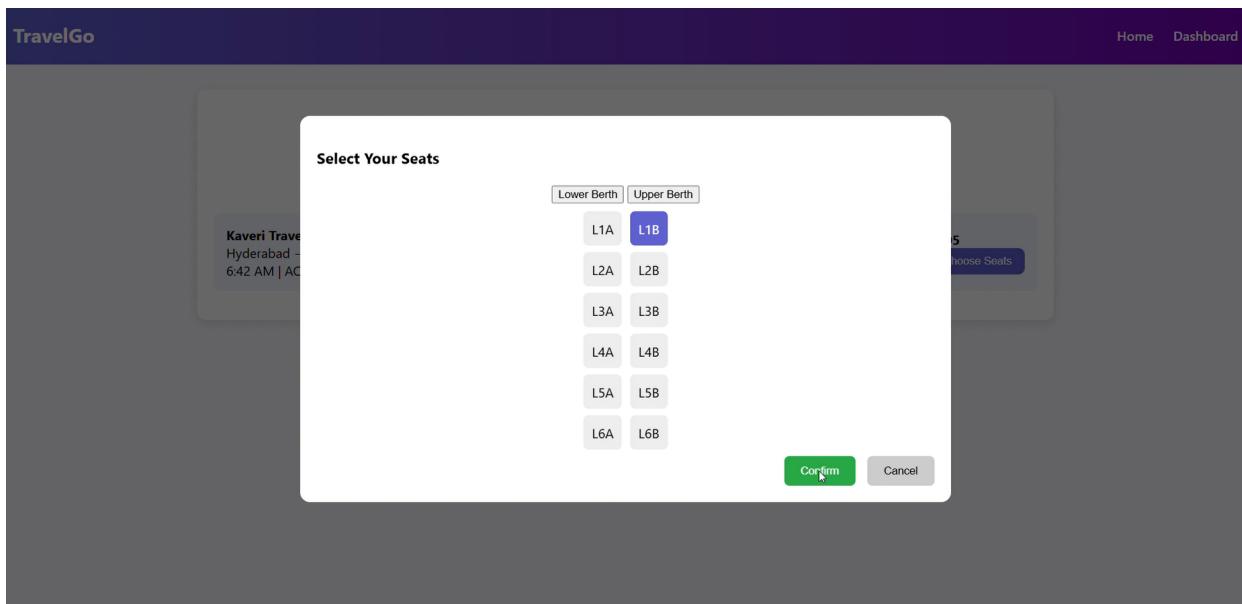
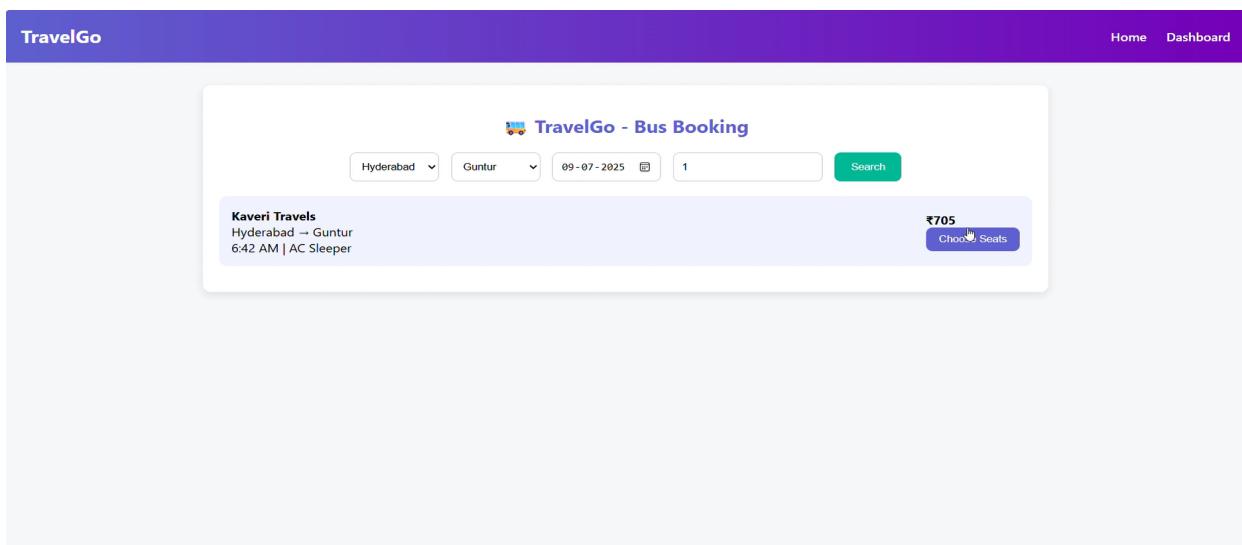
You haven't made any bookings yet. Start your travel planning now!

- **Booking Tabs: Bus, Train, Flight, Hotel**

For demonstration purposes, a seat selection section has also been included in the bus booking module. This allows users to choose their preferred seats during the booking process, enhancing the overall user experience.

This interactive feature simulates real-world booking platforms and showcases the system's dynamic capabilities. It also helps users visualize the layout before confirming their reservation.

- **Bus booking:**



Welcome, rohithapriya25@gmail.com!

User rohithapriya25@gmail.com

Total Bookings 1	Upcoming 0	Completed 0	Cancelled 0
Bus	Train	Flight	Hotel

All Bus Train Flight Hotel dd-mm-yyyy

Your Bookings

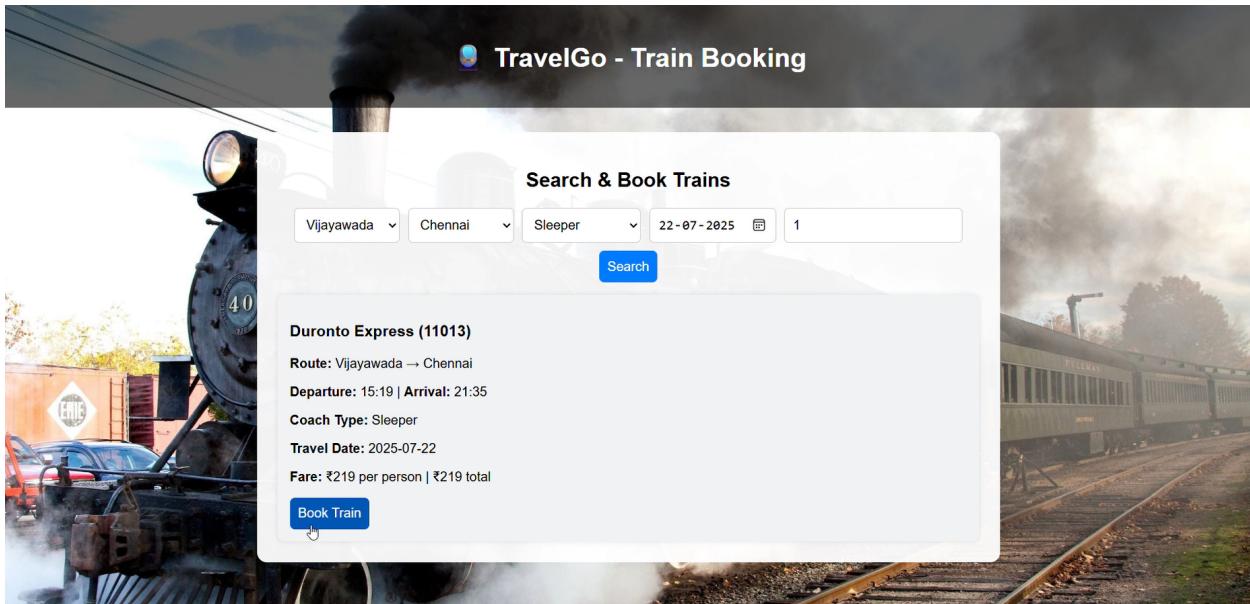
**Bus - Kaveri Travels**

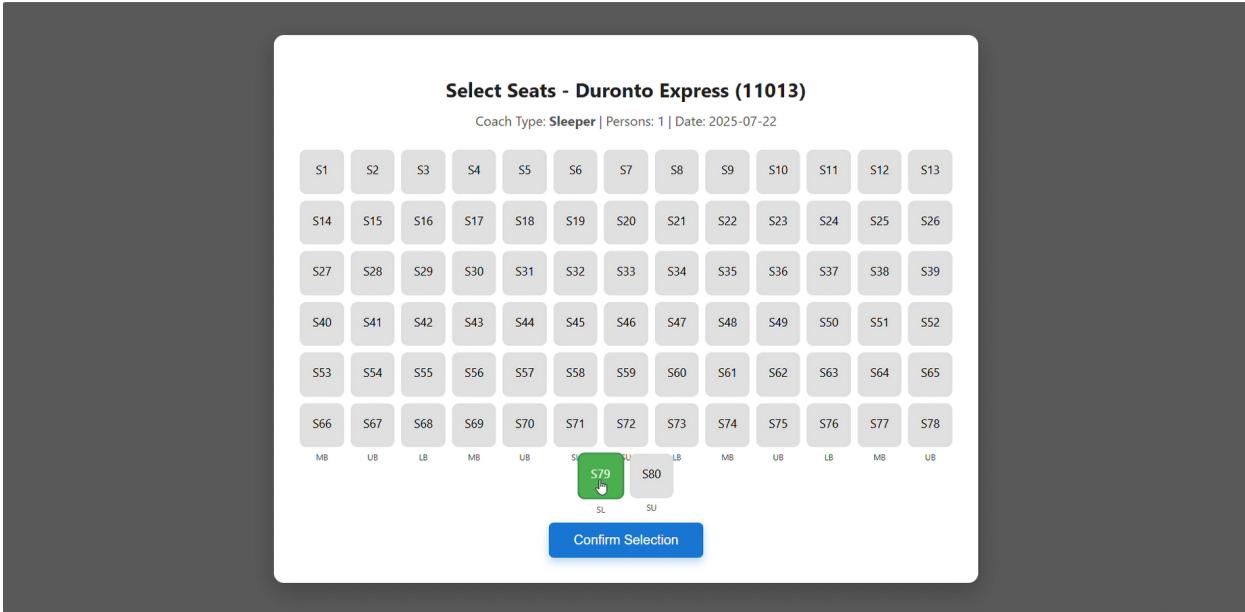
Route: Hyderabad → Guntur  
Date: 2025-07-09  
Seats: L1B  
Passengers: 1  
Total: ₹705.00

Cancel

### • Train booking:

The train booking module enables users to search and reserve train tickets by selecting routes, dates, and times. It fetches real-time data and displays available options tailored to user input. Once a booking is made, the system confirms the reservation and stores the details in the database. Bookings are reflected instantly on the user dashboard for easy tracking. This module ensures a seamless and efficient booking experience for train travelers.





Welcome, rohithapriya25@gmail.com!

User
Logout

Total Bookings  
2

Upcoming  
0

Completed  
0

Cancelled  
0

Bus

Train

Flight

Hotel

All
Bus
Train
Flight
Hotel
dd-mm-yyyy

Your Bookings

**Train - Duronto Express**

Route: Vijayawada → Chennai  
 Date: 2025-07-22  
 Time: 15:19 – 21:35  
 Coach: Sleeper  
 Seats: S79  
 Passengers: 1  
 Total: ₹219.00

### • Flight booking:

→ The flight booking section enables users to search and reserve flights quickly based on their preferred source, destination, and date. Upon entering the details, the system fetches available flight options and displays them with timing, pricing, and route information. The interface is clean and responsive, allowing users to confirm bookings with just a few clicks. Once booked, flight details are stored securely and reflected in the dashboard. This module ensures a smooth and efficient booking experience tailored for

air travel.

## ✈ TravelGo - Flight Booking

**Search & Book Flights**

Hyderabad → Mumbai | 29 - 07 - 2025 | 1 | Search

**IndiGo 6E-234**  
Hyderabad → Mumbai  
2025-07-29 | 06:00 - 07:30  
₹4724 x 1 = ₹4724

**Vistara UK-876**  
Hyderabad → Mumbai  
2025-07-29 | 08:30 - 10:15  
₹3468 x 1 = ₹3468

**Air India AI-543**  
Hyderabad → Mumbai

**Select Your Seats for IndiGo (6E-234)**  
Date: 2025-07-29 | Passengers: 1

Available   Selected   Booked

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

**Welcome, rohithapriya25@gmail.com!**

Dashboard   Logout

User: rohithapriya25@gmail.com

Total Bookings: 3   Upcoming: 0   Completed: 0   Cancelled: 0

Bus   Train   Flight   Hotel

All   Bus   Train   Flight   Hotel

dd-mm-yyyy

**Your Bookings**

**Flight - IndiGo**  
Route: Hyderabad → Mumbai  
Date: 2025-07-29  
Time: 06:00 – 07:30  
Seats: 1A  
Passenger: 1

- **Hotel booking:**

□ The hotel booking section allows users to search and reserve accommodations based on their destination and travel dates. The interface displays available hotels with details such as location, price, and amenities. Users can easily compare options and proceed with booking in just a few clicks. Once confirmed, the booking details appear on the dashboard for easy tracking. The process is designed to be seamless, intuitive, and efficient for all types of travelers.

The screenshot shows the TravelGo Hotel Booking interface. At the top, there is a header bar with the logo "TravelGo" and navigation links "Home" and "Dashboard". Below the header, the main content area has a title "Travel Go - Hotel Booking". It features a search form with fields for "Mumbai" (destination), "29-07-2025" and "31-07-2025" (check-in and check-out dates), "1" (number of guests), and a "Search" button. Below the search form are filters: "5-Star" (checked), "4-Star", "3-Star", "WiFi", "Pool", and "Parking". A dropdown menu for sorting is set to "None". A result card for "ITC Grand Central" is displayed, showing its location in Mumbai, room type (Luxury Suite), price (₹12000/night), and amenities (WiFi, Pool, Parking, Spa). A "Book" button is at the bottom of the card. The bottom half of the screenshot shows a "Confirm Your Hotel Booking" dialog box with the following details:

- Hotel:** ITC Grand Central
- Location:** Mumbai
- Check-in:** 2025-07-29
- Check-out:** 2025-07-31
- Room Type:** (empty)
- Rooms:** 1
- Guests:** 1
- Price per night:** ₹12000.0
- Total nights:** 2
- Total Price:** ₹24000.0

A green "Confirm Booking" button is at the bottom of the dialog.

◇ At the bottom of the dashboard, users can view a summary of all their bookings, including travel details, dates, and status. A dedicated “Cancel” button is provided next to each entry, allowing users to easily cancel any upcoming reservations if needed.

This section provides a centralized view for managing bookings efficiently. It ensures transparency by displaying every confirmed ticket in an organized format. The cancel feature updates the backend in real time, removing the booking from the list and database. This functionality enhances user control and flexibility while planning or modifying travel. It also improves overall convenience by reducing the need to revisit individual booking sections.

## All Bookings:

**Flight - IndiGo**  
Route: Hyderabad → Mumbai  
Date: 2025-07-29  
Time: 06:00 – 07:30  
Seats: 1A  
Passengers: 1  
Total: ₹4,724.00

**Train - Duronto Express**  
Route: Vijayawada → Chennai  
Date: 2025-07-22  
Time: 15:19 – 21:35  
Coach: Sleeper  
Seats: S79  
Passengers: 1  
Total: ₹219.00

**Bus - Kaveri Travels**  
Route: Hyderabad → Guntur  
Date: 2025-07-09  
Seats: L1B  
Passengers: 1  
Total: ₹705.00

## Cancel Bookings:

Total Bookings  
**4**

127.0.0.1:5000 says  
Cancel this booking?

0 0 0 0

Cancelled  
**0**

Bus Train Flight Hotel

All Bus Train Flight Hotel dd-mm-yyyy

Your Bookings

**Bus - Kaveri Travels**  
Route: Hyderabad → Guntur  
Date: 2025-07-09  
Seats: L1B  
Passenger: 1  
Total: ₹705.00

Once a booking is cancelled, a confirmation message appears indicating the cancellation was successful. This real-time feedback reassures the user that their action has been completed without issues. It also helps avoid confusion or repeated

attempts. The booking entry is immediately removed from the dashboard view. This seamless flow enhances the overall usability and responsiveness of the application.

Let's have a look at the page indicating the cancellation was successful.

The screenshot shows the TravelGo dashboard. At the top, there is a header bar with the logo 'TravelGo' on the left, and 'Dashboard' and 'Logout' on the right. A user profile icon and the email 'rohithapriya25@gmail.com' are also present. Below the header, a blue banner displays the welcome message 'Welcome, rohithapriya25@gmail.com!'. The main content area features several cards: 'Total Bookings' (0), 'Upcoming' (0), 'Completed' (0), and 'Cancelled' (0). Below these are four category cards: 'Bus' (with a bus icon), 'Train' (with a train icon), 'Flight' (with a plane icon), and 'Hotel' (with a hotel building icon). A navigation bar below the category cards includes buttons for 'All', 'Bus', 'Train', 'Flight', and 'Hotel'. To the right of the navigation bar is a date input field 'dd-mm-yyyy' with a calendar icon. The section titled 'Your Bookings' contains a message: 'You haven't made any bookings yet. Start your travel planning now!'.

## Final Conclusion – Elevating Travel with TravelGo:

TravelGo stands as a dynamic and cloud-powered solution that redefines how users plan and book their journeys. By seamlessly integrating Flask for backend logic, AWS EC2 for scalable deployment, DynamoDB for reliable data storage, and SNS for real-time notifications, the platform delivers a smooth, responsive, and user-centric experience.

From login to booking, and from instant alerts to effortless cancellations, every feature is built with efficiency and accessibility in mind. Whether it's buses, trains, flights, or hotels—TravelGo offers a unified platform that adapts to diverse travel needs. Its modern architecture ensures high performance, security, and scalability even under peak load conditions.

**With TravelGo, travel planning is no longer a hassle—it's an experience.**  
Smart. Fast. Reliable. That's the future of travel.