

# 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.

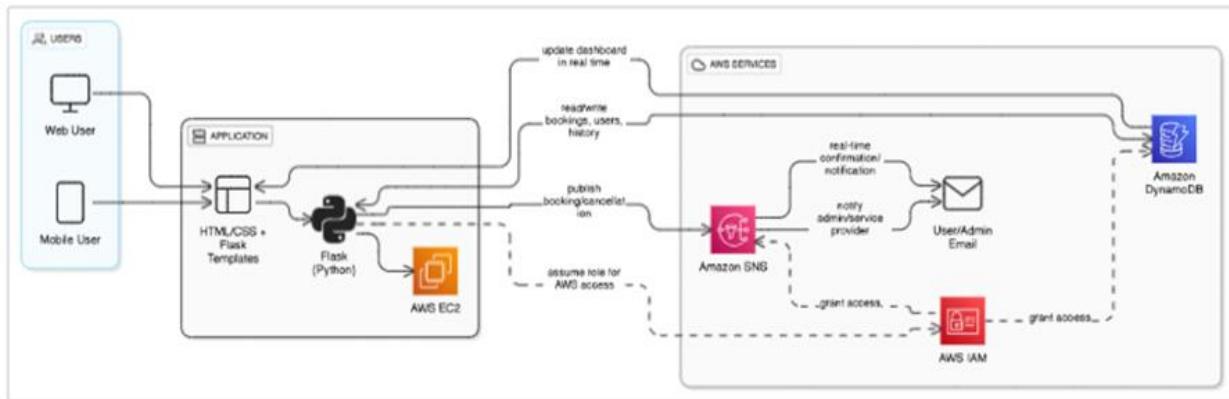
### Scenario 3: Easy Booking Access and Control

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.

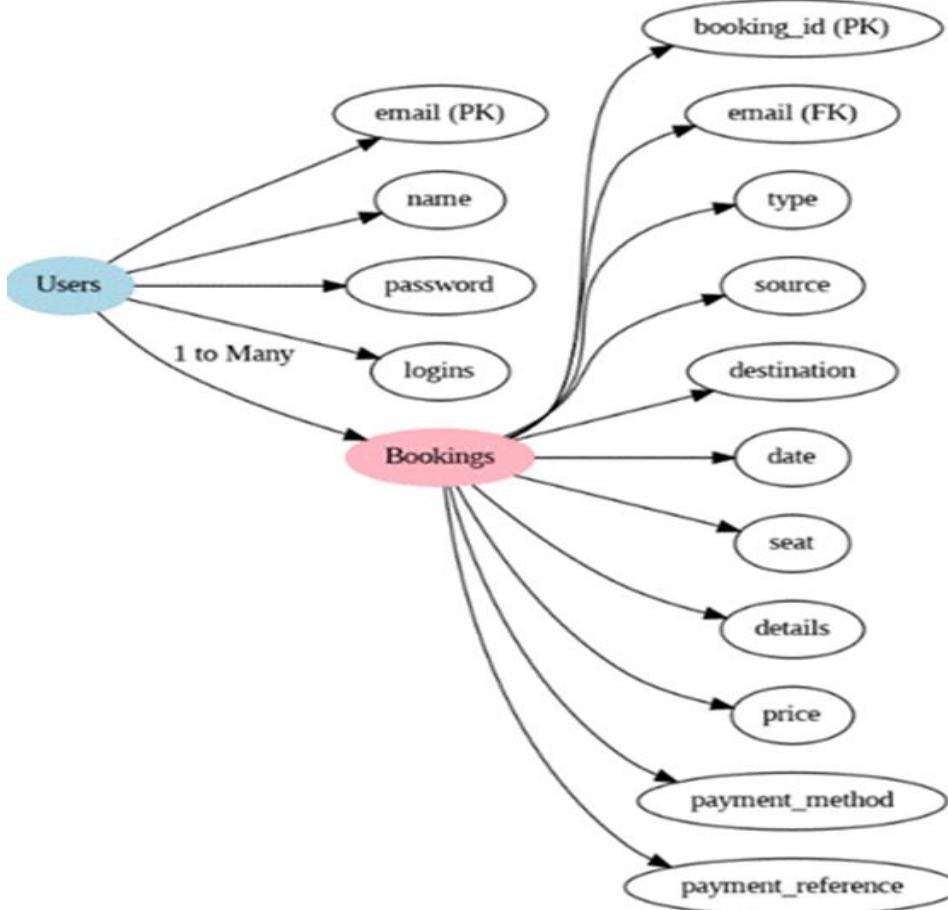
## AWS Architecture Diagram

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.

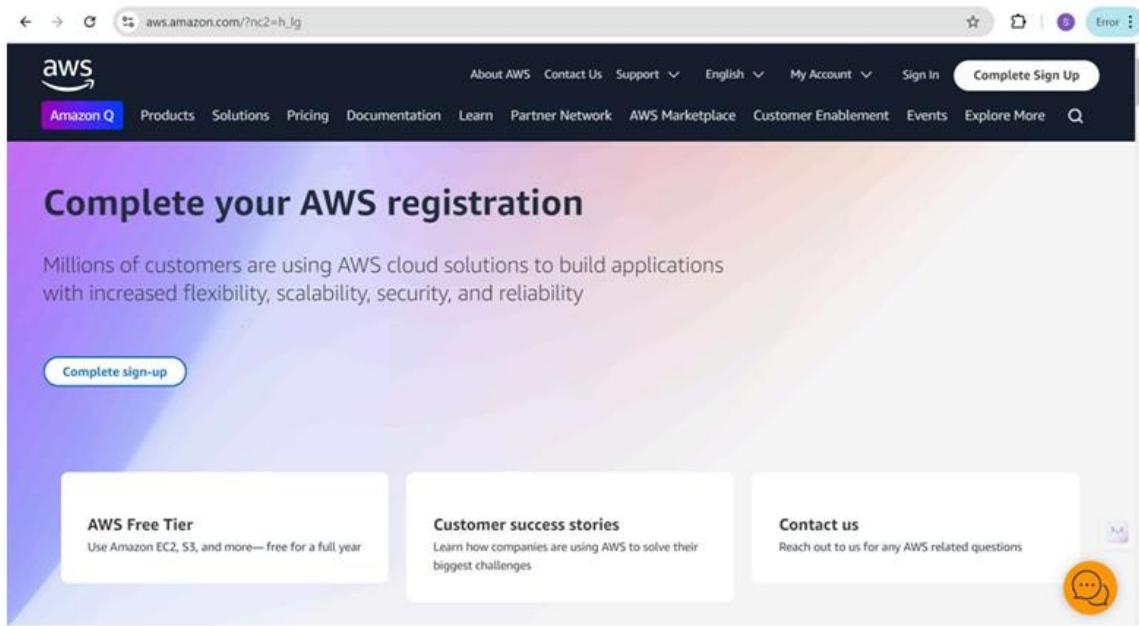
### 7. Flask Application Deployment

- 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

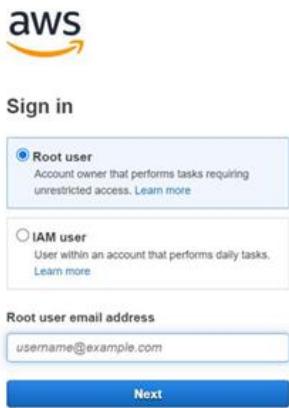
### 1. AWS Account Setup and Login



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

**After**

Once you  
Use the  
IAM as needed for your project setup.



The screenshot shows the AWS sign-in interface. It has a "Sign in" header, two radio button options ("Root user" and "IAM user"), a "Root user email address" input field containing "username@example.com", and a "Next" button at the bottom. A small legal notice is visible at the bottom of the form.



The screenshot shows the AI Use Case Explorer landing page. The title is "AI Use Case Explorer". The text reads: "Discover AI use cases, customer success stories, and expert-curated implementation plans". There is a "Explore now" button at the bottom.

### 3. IAM Roles Creation

The screenshot shows two browser windows side-by-side. The left window displays the AWS IAM Dashboard with a sidebar containing 'Access management' and 'Access reports'. The main area shows 'IAM resources' with a red box highlighting an 'Access denied' error message: 'You don't have permission to iam:GetAccountSummary. To request access, copy the following text and send it to your AWS administrator.' Below this is a 'Diagnose with Amazon Q' button. To the right, there's an 'AWS Account' section with another 'Access denied' message for 'iam>ListAccountAliases'. The right window shows a 'Create role' wizard step titled 'Use case'. It asks 'Allow an AWS service like EC2, Lambda, or others to perform actions in this account.' A dropdown menu is set to 'EC2'. Under 'Service or use case', the 'EC2' option is selected. Other options include 'EC2 Role for AWS Systems Manager', 'EC2 Spot Fleet Role', 'EC2 - Spot Fleet Auto Scaling', 'EC2 - Spot Fleet Tagging', 'EC2 - Spot Instances', 'EC2 - Spot Fleet', and 'EC2 - Scheduled Instances'. At the bottom right of the wizard are 'Cancel' and 'Next' buttons.

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**

**Permissions policies (3/1060)**

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

Policy name	Type	Description
<a href="#">AmazonEC2ContainerRegistryFullAccess</a>	AWS managed	Provides administrative access to Amazon EC2 Container Registry
<a href="#">AmazonEC2ContainerRegistryPowerUser</a>	AWS managed	Provides full access to Amazon EC2 Container Registry
<a href="#">AmazonEC2ContainerRegistryPullOnly</a>	AWS managed	Provides access to pull images from Amazon EC2 Container Registry
<a href="#">AmazonEC2ContainerRegistryReadOnly</a>	AWS managed	Provides read-only access to Amazon EC2 Container Registry
<a href="#">AmazonEC2ContainerServiceAutoscaleRole</a>	AWS managed	Policy to enable Task AutoScaling for Amazon ECS
<a href="#">AmazonEC2ContainerServiceEventsRole</a>	AWS managed	Policy to enable CloudWatch Events for Amazon ECS
<a href="#">AmazonEC2ContainerServiceforEC2Role</a>	AWS managed	Default policy for the Amazon EC2 Role for Amazon ECS
<a href="#">AmazonEC2ContainerServiceRole</a>	AWS managed	Default policy for Amazon ECS service role
<a href="#">AmazonEC2FullAccess</a>	AWS managed	Provides full access to Amazon EC2 via the AWS SDKs and command-line interface
<a href="#">AmazonEC2ReadOnlyAccess</a>	AWS managed	Provides read only access to Amazon EC2

**Name, review, and create**

**Role details**

**Role name**  
Studentuser

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

**Step 1: Select trusted entities**

**Trust policy**

```

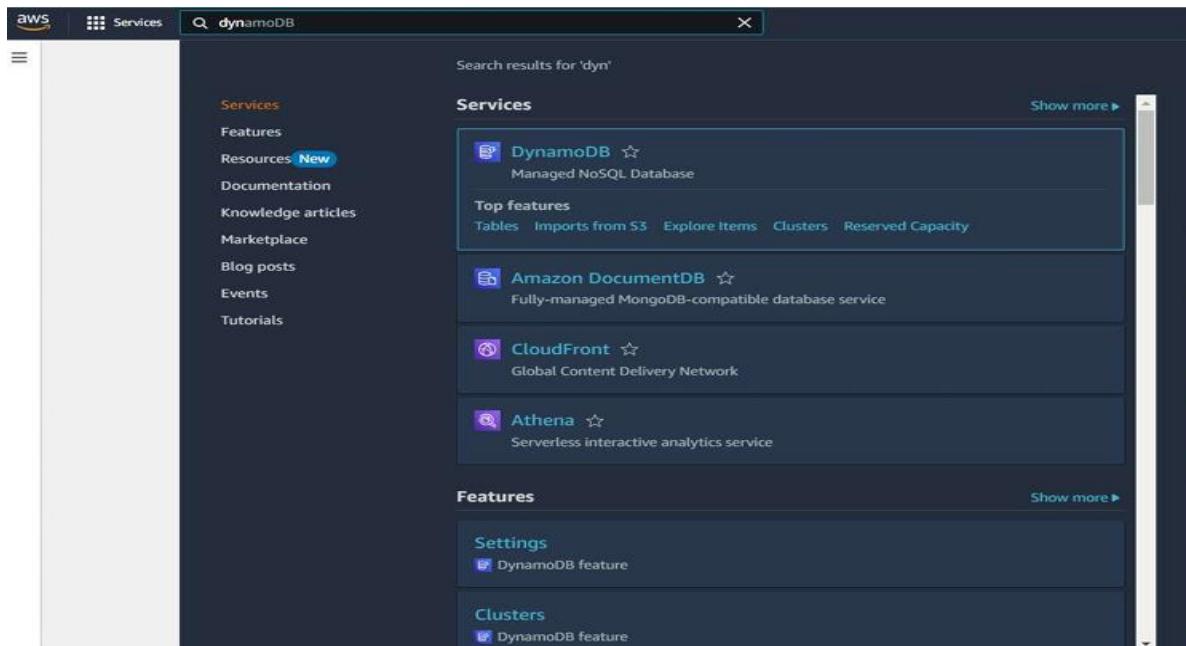
1 = [
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "sts:AssumeRole"
8       ]
9     }
10  ]
11 ]
  
```

## 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 uniquely identify records—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.

The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. In the 'Table details' section, the table name is set to 'travelgo\_users'. The 'Partition key' is defined as 'email' of type 'String'. The 'Sort key - optional' field is empty. At the bottom, there are tabs for 'Table settings', 'CloudShell', and 'Feedback'.

**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.

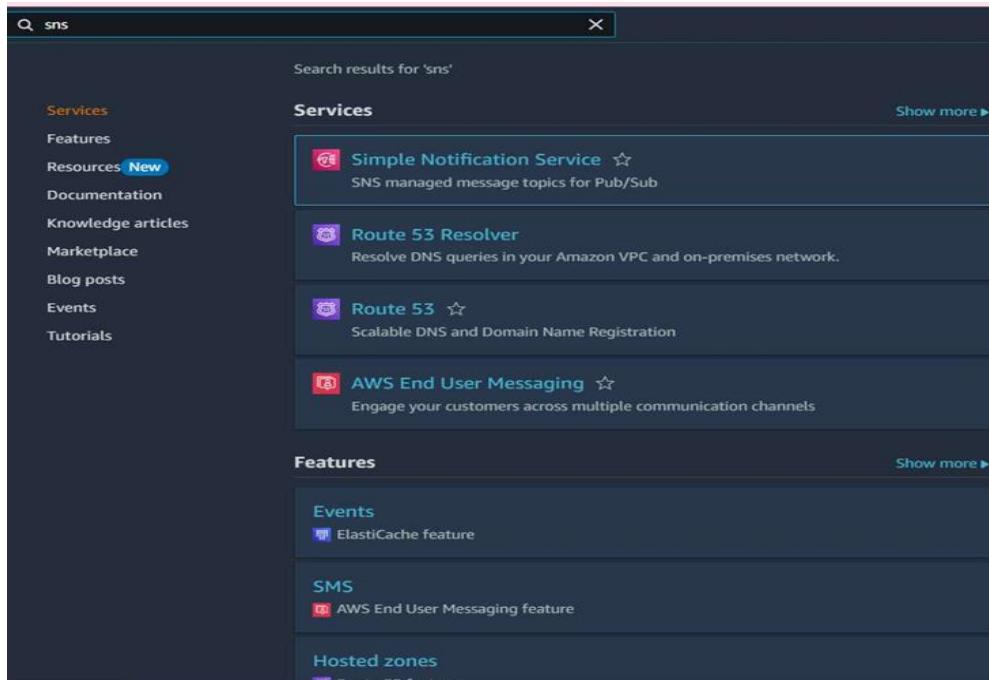
The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. In the 'Table details' section, the table name is set to 'bookings'. The 'Partition key' is defined as 'user\_email' of type 'String'. The 'Sort key - optional' field contains 'booking\_date' of type 'String'. Both fields have a note indicating they are between 1 and 255 characters and case sensitive. At the bottom, there are tabs for 'Table settings', 'CloudShell', and 'Feedback'.

The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The 'Table details' step is active, with a blue header bar at the top containing a feedback survey link. The main form includes fields for 'Table name' (set to 'trains'), 'Partition key' (set to 'train\_number' of type 'String'), and 'Sort key - optional' (set to 'date' of type 'String'). The 'Table details' section also contains a note about DynamoDB being schemaless and requires only a table name and primary key.

The screenshot shows the 'Tables' page in the AWS DynamoDB console. The left sidebar shows navigation links for 'Dashboard', 'Tables' (selected), 'Explore items', 'PartiQL editor', 'Backups', 'Exports to S3', 'Imports from S3', 'Integrations', 'Reserved capacity', and 'Settings'. The main area displays a table titled 'Tables (3) Info' with columns for Name, Status, Partition key, Sort key, Indexes, Replication Regions, Deletion protection, Favorite, and Read. The table lists three tables: 'bookings', 'trains', and 'travelgo\_users', all in Active status with various key and index configurations.

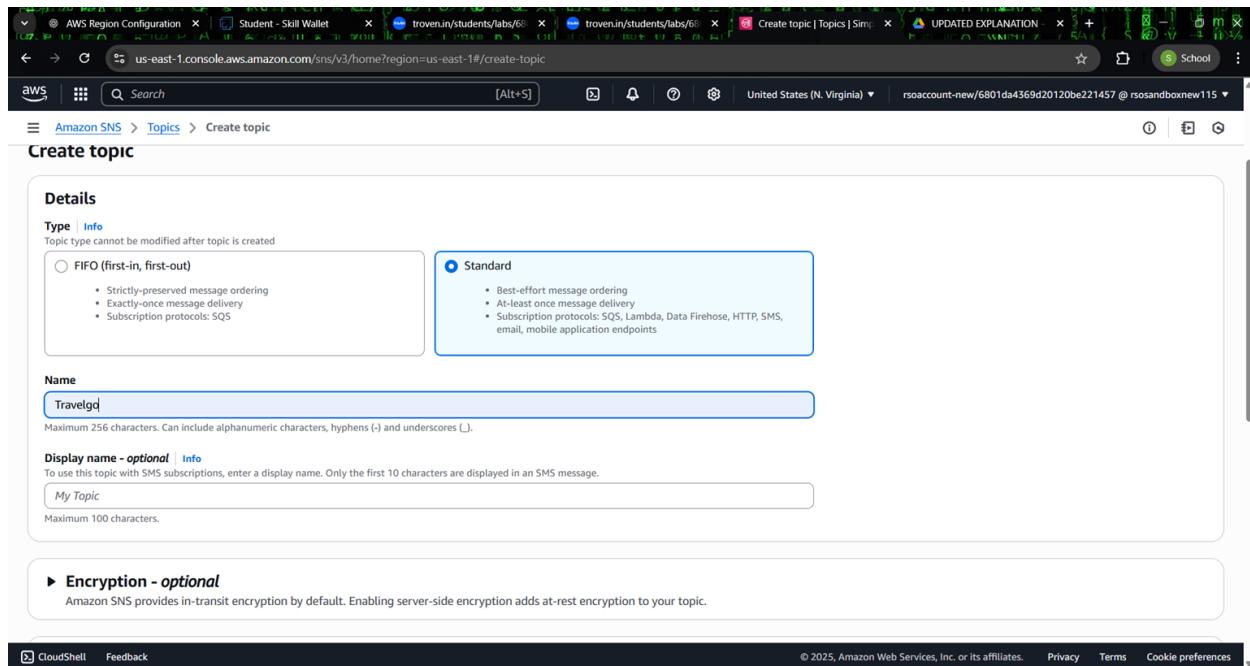
## 5. SNS Notification Setup

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



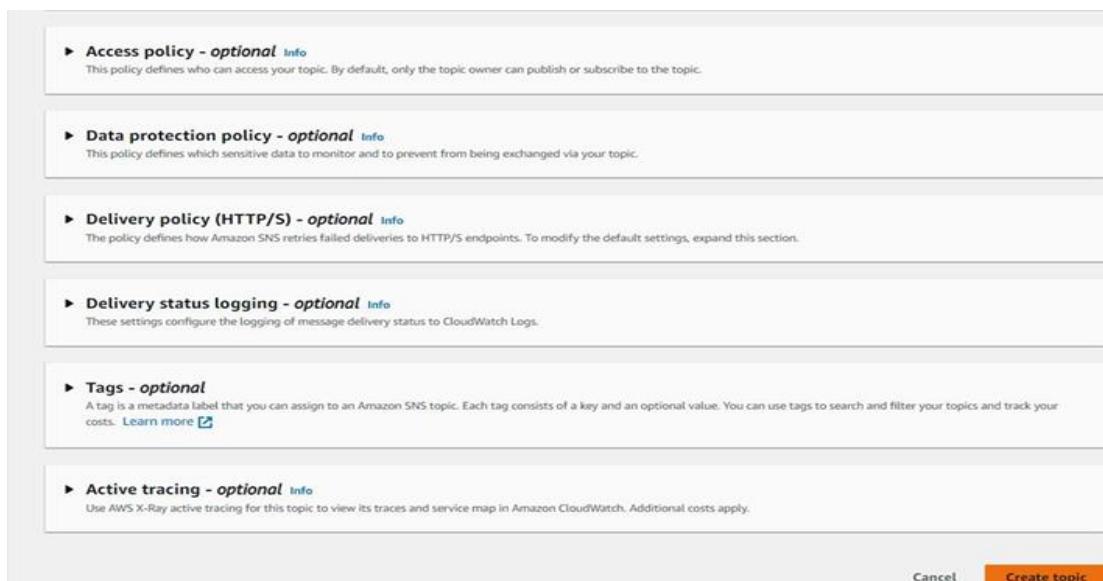
A screenshot of the Amazon Simple Notification Service (SNS) homepage. At the top, there is a blue banner with a 'New Feature' message: 'Amazon SNS now supports High Throughput FIFO topics. Learn more'. Below the banner, the page title is 'Amazon Simple Notification Service' with the subtitle 'Pub/sub messaging for microservices and serverless applications.' To the right, there is a 'Create topic' form with a 'Topic name' input field containing 'MyTopic', a 'Next step' button, and a 'Start with an overview' link. Further down, there is a 'Pricing' section with a detailed description of the service's cost structure. The bottom of the page includes standard browser navigation bars like CloudShell, Feedback, and a weather widget showing 29°C Mostly cloudy, along with a footer containing copyright information and a date of 01-07-2025.

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



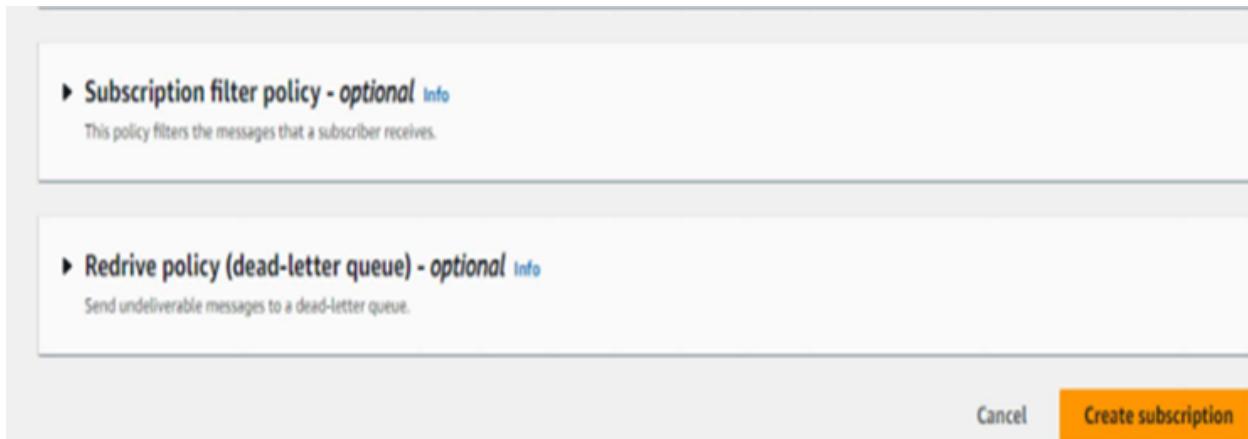
## 2. Click on create topic

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.



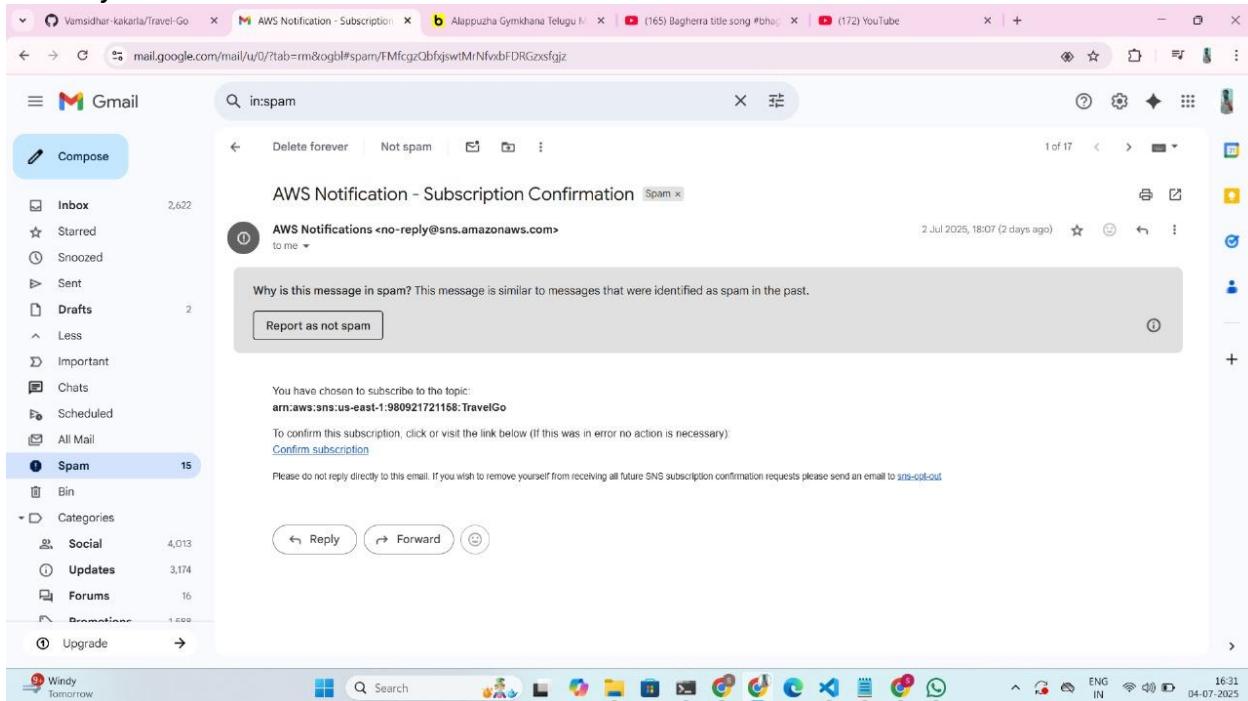
## 3. Configure the SNS topic and note down the Topic ARN.

4. Click on create subscription.



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
80     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
109        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
115    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:

The screenshot shows a GitHub repository interface. At the top, there are buttons for 'main' (selected), '1 Branch', '0 Tags', 'Go to file', 'Add file', and 'Code'. Below is a list of commits:

Commit	Message	Date
static	Initial commit	3 days ago
templates	Initial commit	3 days ago
venv	Initial commit	3 days ago
venv_new	Initial commit	3 days ago
README.md	Initial commit	3 days ago
app.py	Update app.py with latest changes	2 days ago

Launch an EC2 instance to host the Flask application.

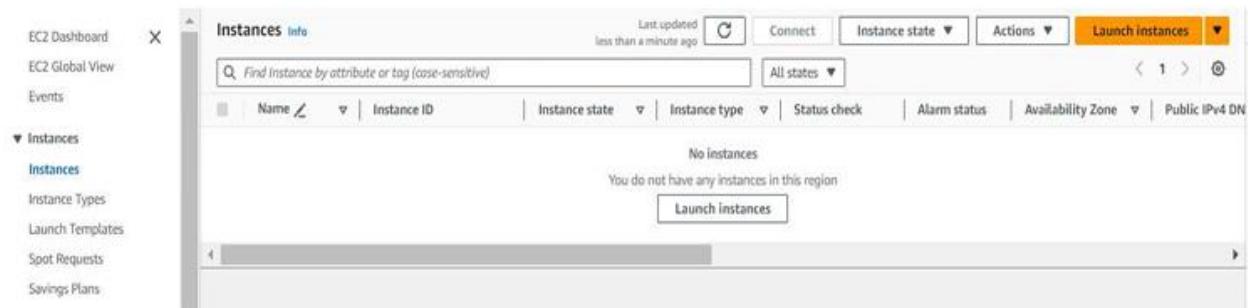
The screenshot shows the AWS Services Catalog search results for 'EC2'. The search bar at the top contains 'EC2'. On the left, there is a sidebar with categories: Services (9), Features (40), Blogs (1,735), Documentation (126,553), Knowledge Articles (30), Tutorials (18), Events (8), and Marketplace (1,366). The main area displays the search results under the 'Services' heading:

Service	Description	Actions
EC2	Virtual Servers in the Cloud	See all 9 results ▶
EC2 Image Builder	A managed service to automate build, customize and deploy OS images	
AWS Compute Optimizer	Recommend optimal AWS Compute resources for your workloads	
AWS Firewall Manager	Central management of firewall rules	

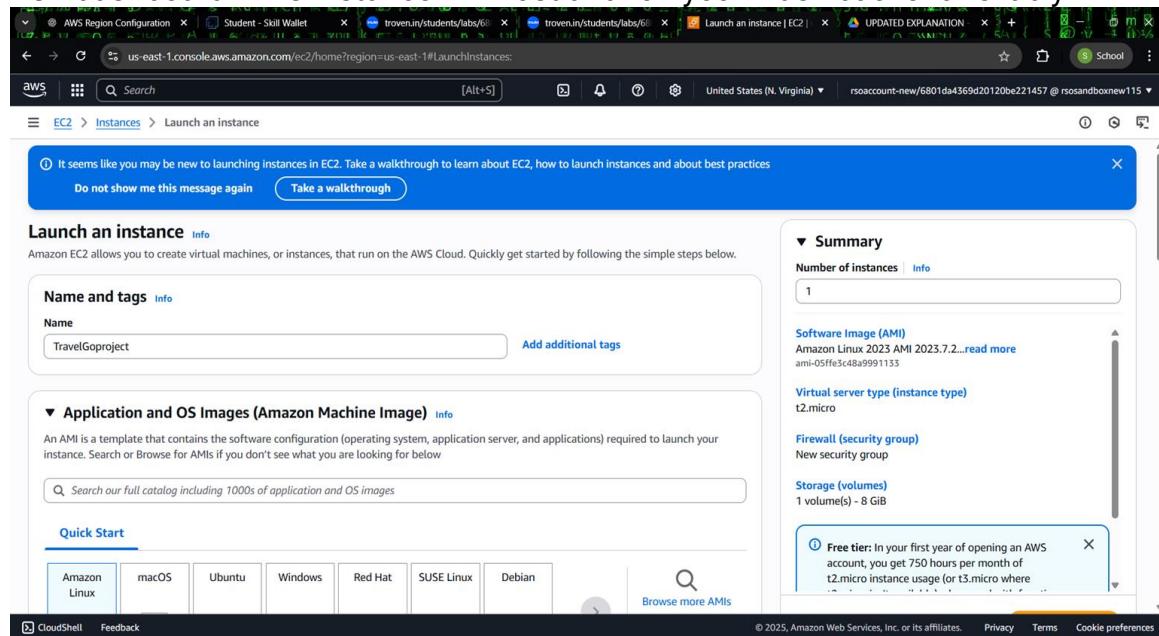
Below the services, there is a section for 'Features':

Feature	Description	Actions
Export snapshots to EC2	Lightsail feature	See all 40 results ▶
Dashboard	EC2 feature	

## 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.

The screenshot shows the AWS EC2 'Launch an instance' wizard. In the 'Auto-assign public IP' section, 'Enable' is selected. Under 'Firewall (security groups)', a new security group 'launch-wizard-1' is being created with three rules: 'Allow SSH traffic from Anywhere', 'Allow HTTPS traffic from the internet', and 'Allow HTTP traffic from the internet'. In the 'Configure storage' section, an 8 GiB gp3 root volume is selected. On the right, the 'Summary' pane shows one instance launching with an Amazon Linux 2023 AMI and a t2.micro instance type. A tooltip indicates a free tier benefit of 750 hours per month for the first year.

The screenshot shows the 'Create key pair' step of the wizard. A key pair named 'Travelgo' is being created with RSA type and .pem file format. A tooltip provides instructions for storing the private key securely. The 'Summary' pane on the right remains the same as the previous step, showing one instance launching with an Amazon Linux 2023 AMI and a t2.micro instance type.

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.

The screenshot shows the AWS EC2 Instances Launch wizard. A green success message at the top states: "Successfully initiated launch of instance (i-0e7f9bfc28481d812)". Below this, there's a "Launch log" section. Under "Next Steps", there are four cards: "Create billing and free tier usage alerts", "Connect to your instance", "Connect an RDS database", and "Create EBS snapshot policy". At the bottom, there are links for "Manage detailed monitoring", "Create Load Balancer", "Create AWS budget", and "Manage CloudWatch alarms". The browser status bar indicates the URL is us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#LaunchInstances: and the session is rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew100.

## 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 Security Groups Edit inbound rules page. It lists existing rules and allows adding new ones. A new rule has been added with the following details:

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

A note at the bottom says: "⚠️ 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." Buttons for "Cancel", "Preview changes", and "Save rules" are at the bottom right.

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**

<input checked="" type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm state
<input checked="" type="checkbox"/>	TravelGoproject	i-0e7f9bfc28481d812	Running	t2.micro		

**Actions**

- Instance diagnostics
- Instance settings
- Networking
- Security (selected)
- Image and templates
- Monitor and troubleshoot

**i-0e7f9bfc28481d812 (TravelGoproject)**

Public IP: 490004646397

Launched: Wed Jul 02 2025 13:50:57 GMT+0530 (India Standard Time)

**Security groups**

- sg-067a674658fc816b (launch-wizard-1)

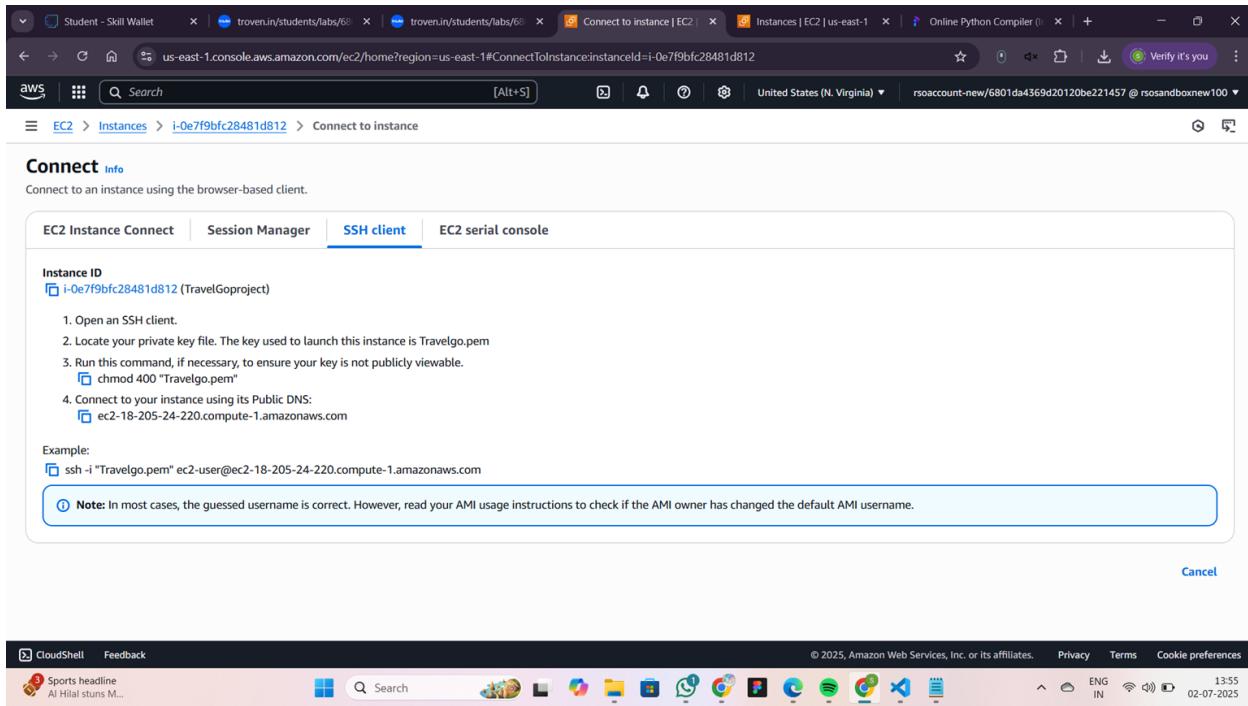
**Inbound rules**

The screenshot shows the AWS CloudShell interface. At the top, there are several tabs open, including "AWS Region Configuration", "Student - Skill Wallet", "troven.in/students/labs/68", "troven.in/students/labs/68", "Modify IAM role | EC2 | us-east-1", "UPDATED EXPLANATION", and others. The main content area shows the "Modify IAM role" page for an instance with ID i-0426bada174a7ef93. A dropdown menu is open, showing "Studentuser" selected under "IAM role". Below the dropdown are buttons for "Create new IAM role" and "Update IAM role". At the bottom right of the page are "Cancel" and "Update IAM role" buttons.

The screenshot shows the "Instances" page in the AWS Management Console. The left sidebar is collapsed, showing "EC2" as the active service. The main pane displays a table of instances. A green success message at the top states "Successfully attached Studentuser to instance i-0e7f9bfc28481d812". The table has columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IPv4. One row is highlighted for "TravelGoproject" with instance ID i-0e7f9bfc28481d812, which is "Running" on a "t2.micro" instance type in "us-east-1b" availability zone, with a public IP of 18.205.24.220. Below the table, a detailed view for instance i-0e7f9bfc28481d812 (TravelGoproject) is shown, including sections for Details, Status and alarms, Monitoring, Security, Networking, Storage, and Tags. The "Details" tab is selected. Under "Instance summary", it shows the instance ID i-0e7f9bfc28481d812, a blank "IPv6 address" field, and the "Public IPv4 address" 18.205.24.220 with a link to "open address". It also shows the "Instance state" as "Running". To the right, it lists "Private IPv4 addresses" (172.31.82.161), "Public DNS" (ec2-18-205-24-220.compute-1.amazonaws.com), and another link to "open address".

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 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.

- 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)
|████████████████████████████████████████████████████████████████| 103 kB 15.1 MB/s
Collecting itsdangerous>=2.2.0
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting markupsafe>=2.1.1
  Downloading MarkupSafe-3.0.2-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)
|████████████████████████████████████████████████████████████████| 98 kB 12.3 MB/s
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-8.7.0-py3-none-any.whl (27 kB)
Collecting jinja2>=3.1.2
```

```
Windows PowerShell x + 
Python 3.9.23
[ec2-user@ip-172-31-88-102 Travel-Go]$ python app.p
-bash: python: command not found
[ec2-user@ip-172-31-88-102 Travel-Go]$ python app.py
-bash: python: command not found
[ec2-user@ip-172-31-88-102 Travel-Go]$ Python app.py
-bash: Python: command not found
[ec2-user@ip-172-31-88-102 Travel-Go]$ python app.py
-bash: python: command not found
[ec2-user@ip-172-31-88-102 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://172.31.88.102:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 526-078-484
103.42.250.170 - - [01/Jul/2025 08:30:33] "GET / HTTP/1.1" 200 -
103.42.250.170 - - [01/Jul/2025 08:30:34] "GET /static/images/travel-bg.jpg HTTP/1.1" 404 -
103.42.250.170 - - [01/Jul/2025 08:30:34] "GET /favicon.ico HTTP/1.1" 404 -
103.42.250.170 - - [01/Jul/2025 08:30:39] "GET /login HTTP/1.1" 200 -
103.42.250.170 - - [01/Jul/2025 08:30:43] "GET /register HTTP/1.1" 200 -
103.42.250.170 - - [01/Jul/2025 08:30:58] "POST /register HTTP/1.1" 500 -
Traceback (most recent call last):
  File "/home/ec2-user/.local/lib/python3.9/site-packages/flask/app.py", line 1536, in __call__
    return self.wsgi_app(environ, start_response)
  File "/home/ec2-user/.local/lib/python3.9/site-packages/flask/app.py", line 1514, in wsgi_app
    response = self.handle_exception(e)
  File "/home/ec2-user/.local/lib/python3.9/site-packages/flask/app.py", line 1511, in wsgi_app
    response = self.full_dispatch_request()
  File "/home/ec2-user/.local/lib/python3.9/site-packages/flask/app.py", line 919, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "/home/ec2-user/.local/lib/python3.9/site-packages/flask/app.py", line 917, in full_dispatch_request
    rv = self.dispatch_request()
  File "/home/ec2-user/.local/lib/python3.9/site-packages/flask/app.py", line 902, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args) # type: ignore[no-any-return]
  File "/home/ec2-user/Travel-Go/app.py", line 59, in register
    existing = users_table.get_item(Key={'email': email})
```

- pip install flask
  - pip install boto3

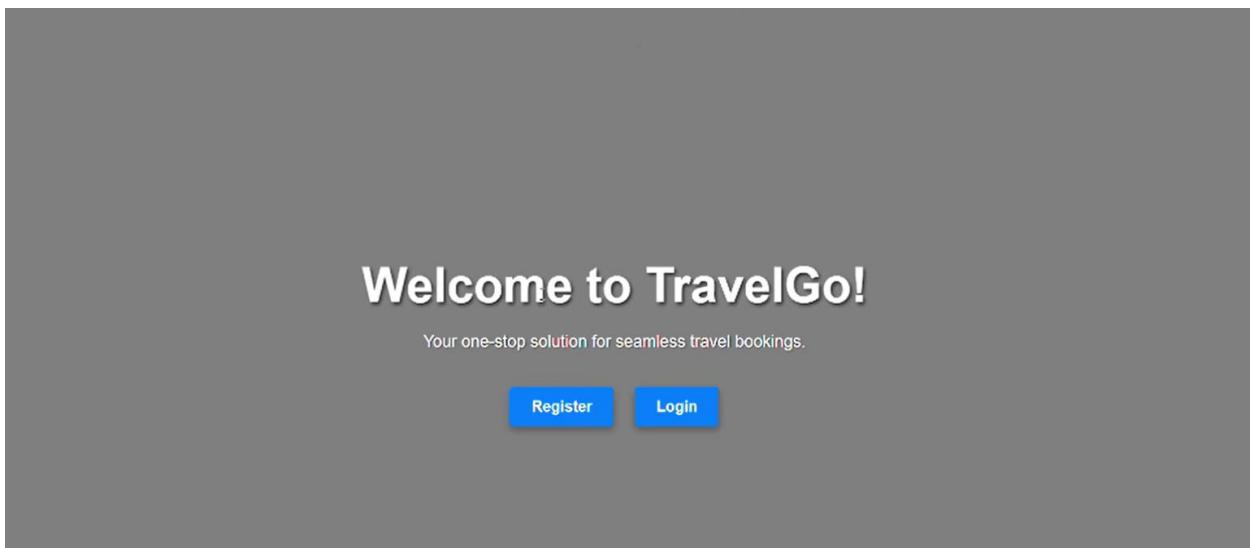
- python3 app.py

### Deployment Steps Summary:

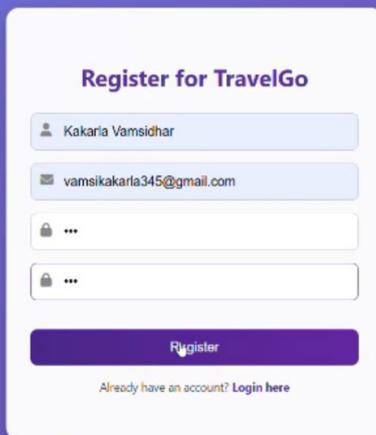
- sudo yum install git -y
- git clone your\_repository\_url
- cd your\_project\_directory
- sudo yum install python3 -y
- sudo yum install python3-pip -y
- pip install flask
- pip install boto3
- python3 app.py

### User Interface Screens:

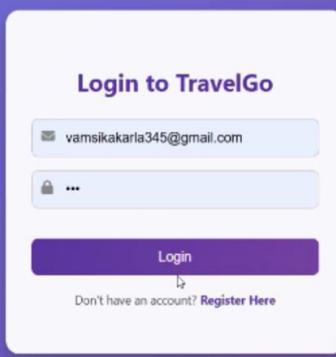
- Homepage:



- Register and Login Portals:



The image shows the 'Register for TravelGo' form. It features four input fields: a first name field containing 'Kakaria Vamsidhar', an email field containing 'vamsikakarla345@gmail.com', and two password fields. Below the fields is a purple 'Register' button. At the bottom, there is a link 'Already have an account? [Login here](#)'.



The image shows the 'Login to TravelGo' form. It has two input fields: an email field containing 'vamsikakarla345@gmail.com' and a password field. Below the fields is a purple 'Login' button. At the bottom, there is a link 'Don't have an account? [Register Here](#)'.

- 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, vamsikakarla345@gmail.com!

Here you can view your upcoming and past travel bookings.

Email already exists!

Logged in successfully!

Bus

Bus

Train

Train

Flight

Flight

Hotel

Hotel

#### YOUR BOOKINGS

You have no bookings yet. Start by searching for a trip!

### • 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:

**Search & Book Buses**

Hyderabad → Vijayawada → 02-07-2025 1 Search

AC  Non-AC  Sleeper  Semi-Sleeper  Seater

Sort by Price: None

<b>Orange Travels</b> AC Sleeper • 08:00 AM • ₹800/person	↳ <span style="border: 1px solid #0072BD; color: white; padding: 2px 10px; border-radius: 5px;">Book</span>
<b>Garuda Express</b> Non-AC Seater • 11:00 AM • ₹400/person	↳ <span style="border: 1px solid #0072BD; color: white; padding: 2px 10px; border-radius: 5px;">Book</span>
<b>Greenline Travels</b> AC Sleeper • 09:00 PM • ₹850/person	↳ <span style="border: 1px solid #0072BD; color: white; padding: 2px 10px; border-radius: 5px;">Book</span>

You can select 1 seat(s)

S1	S2	S3	S4
S5	S6	S7	S8
S9	S10	S11	S12
S13	<b>S14</b>	S15	S16
S17	S18	S19	S20
S21	S22	S23	S24
S25	S26	S27	S28
S29	S30	S31	S32
S33	S34	S35	S36
S37	S38	S39	S40

**Confirm Booking**

TravelGo

Home Dashboard Logout

Welcome, vamsikakarla345@gmail.com!

Here you can view your upcoming and past travel bookings.

Bus booking confirmed successfully!

**Bus**      **Train**      **Flight**      **Hotel**

**YOUR BOOKINGS**

**Bus Booking**  
 Bus: Orange Travels  
 Route: Hyderabad → Vijayawada  
 Date: 2025-07-02  
 Time: 08:00 AM  
 Seats: S14

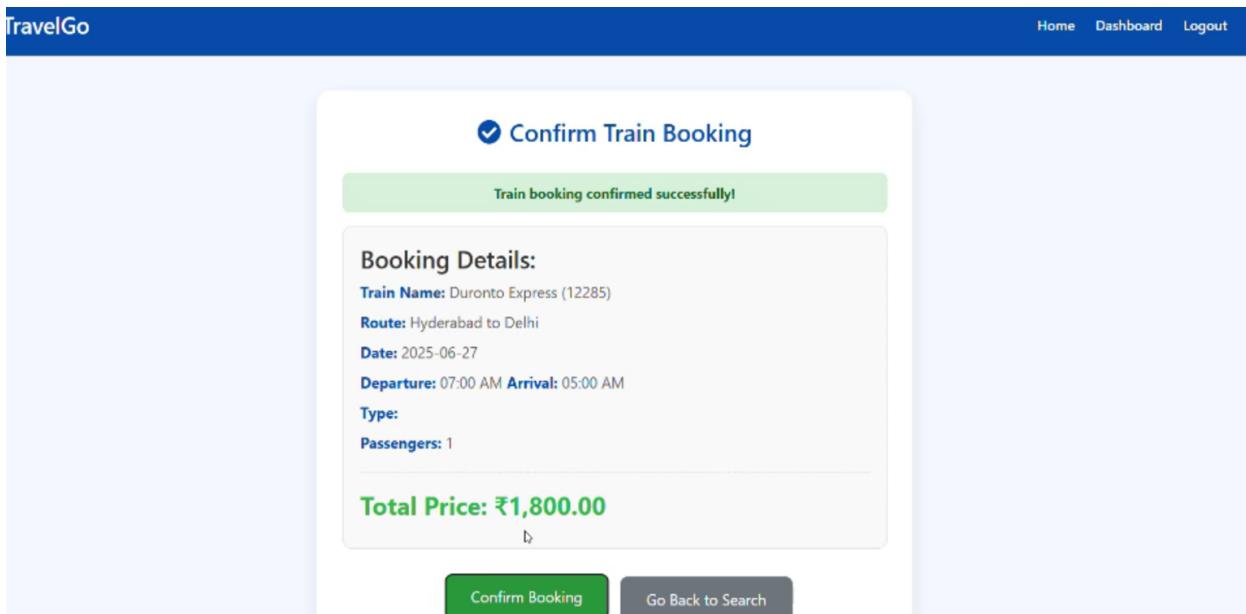
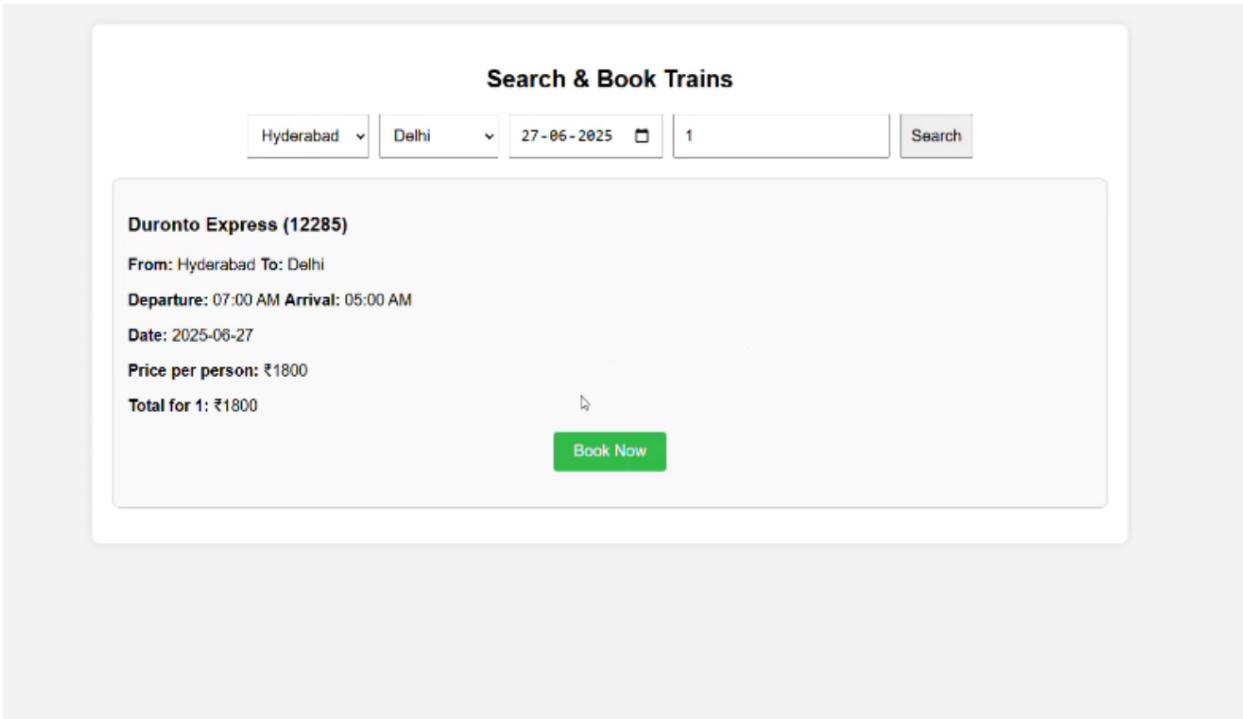
₹800.00

**Cancel Booking**

- Train booking:

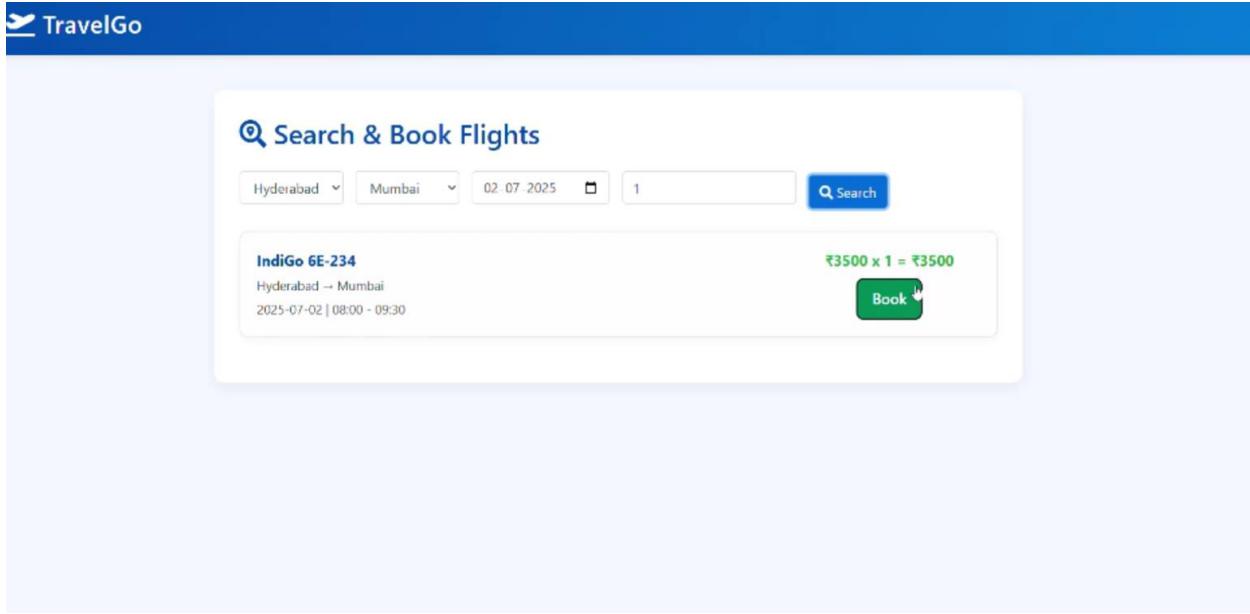
The train booking module enables users to search and reserve train tickets by selecting routes, dates, and times. It fetches real-time data and displays available options tailored to user input. Once a booking is made, the system confirms the reservation and stores the details in the database. Bookings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

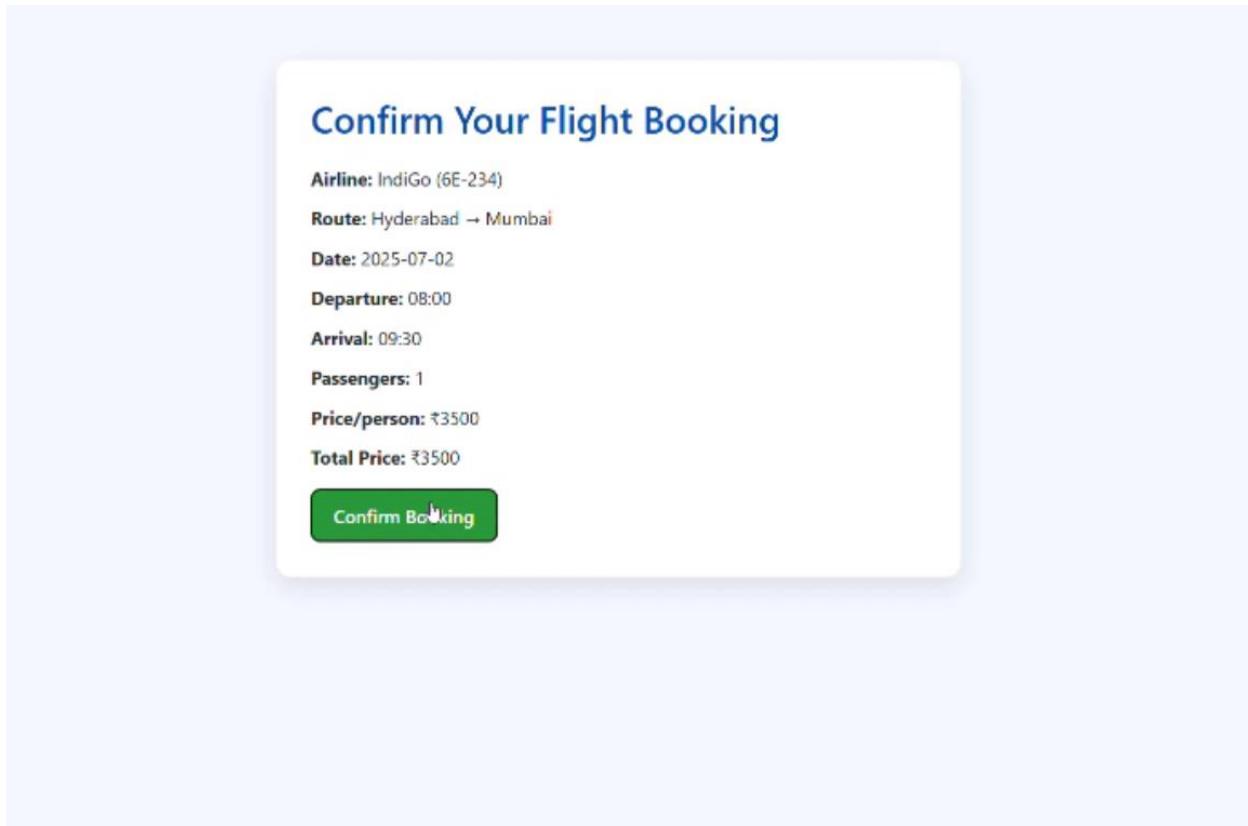
the user dashboard for easy tracking. This module ensures a seamless and efficient booking experience for train travelers.



- 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.





- **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 displays a travel booking application interface. At the top, there is a search bar with fields for location ("Hyderabad"), check-in date ("02-07-2025"), check-out date ("03-07-2025"), number of rooms ("1"), and number of guests ("1"). Below the search bar are several filters: "5-Star", "4-Star", "3-Star", "WiFi", "Pool", and "Parking". A "Sort by" dropdown is set to "None".

The search results show two hotel options:

- Taj Falaknuma Palace**: Hyderabad • 5-Star • ₹25000/night. Amenities: WiFi, Pool, Parking, Restaurant. A green "Book" button is available.
- The Park**: Hyderabad • 3-Star • ₹5500/night. Amenities: WiFi. A green "Book" button is available.

Below the search results, a modal window titled "Confirm Your Hotel Booking" provides detailed booking information:

- Hotel: Taj Falaknuma Palace
- Location: Hyderabad
- Check-in: 2025-07-02
- Check-out: 2025-07-03
- Rooms: 1
- Guests: 1
- Price/night: ₹25000
- Total nights: 1
- Total Cost: ₹25000

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

- 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:

The screenshot displays a dashboard titled 'All Bookings' with three distinct sections: 'Flight Booking', 'Train Booking', and 'Bus Booking'. Each section contains detailed booking information and a red 'Cancel Booking' button.

- Flight Booking:**
  - Flight: IndiGo GE 234
  - Route: Hyderabad → Mumbai
  - Date: 2025-07-02
  - Departure: 08:00
  - Arrival: 09:30
  - Passengers: 1
  - Booked On: 2025-07-02

₹3,500.00

**Cancel Booking**
- Train Booking:**
  - Train: Duronto Express (12285)
  - Route: Hyderabad → Delhi
  - Date: 2025-06-27
  - Time: 07:00 AM - 05:00 AM
  - Passengers: 1
  - Seats: S26
  - Booked On: 2025-07-02

₹1,800.00

**Cancel Booking**
- Bus Booking:**
  - Bus: Orange Travels
  - Route: Hyderabad → Vijayawada
  - Date: 2025-07-02
  - Time: 08:00 AM
  - Seats: S14
  - Passengers: 1
  - Booked On: 2025-07-02

₹800.00

**Cancel Booking**

## Cancel Bookings:

The screenshot shows a confirmation dialog box titled '98.81.163.132:5000 says' asking if the user is sure they want to cancel the booking. The dialog has 'OK' and 'Cancel' buttons. In the background, the main dashboard is visible, showing the same three booking entries as the previous screenshot.

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.

Welcome, vamsikakarla345@gmail.com!

Here you can view your upcoming and past travel bookings.

Booking ebbaa9e5-0736-4477-836d-19d9a9b60589 cancelled successfully!

Bus

Bus

Train

Train

Flight

Flight

Hotel

Hotel

#### YOUR BOOKINGS

You have no bookings yet. Start by searching for a trip!

## 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.