

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.

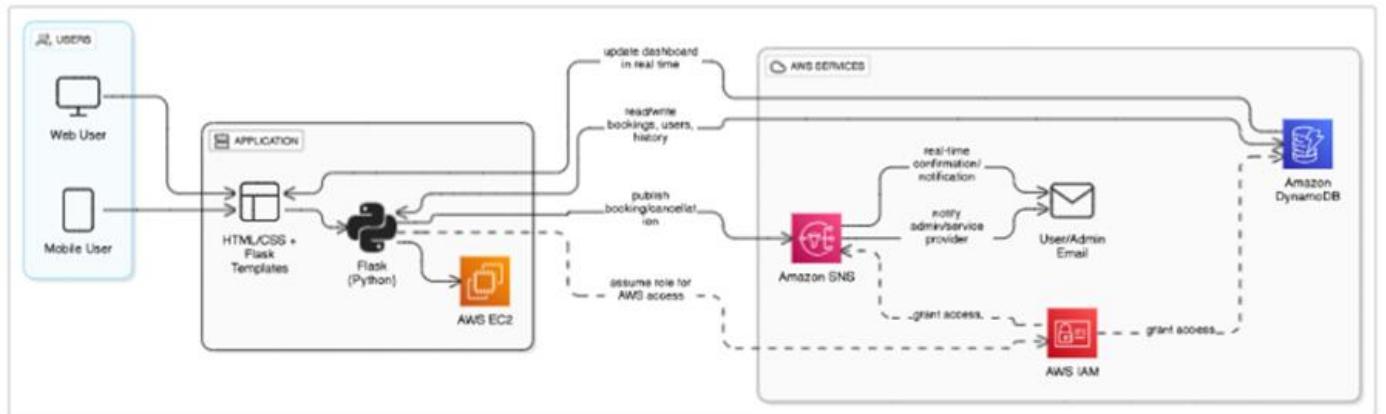
Scenario 3: Easy Booking Access and Control

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.

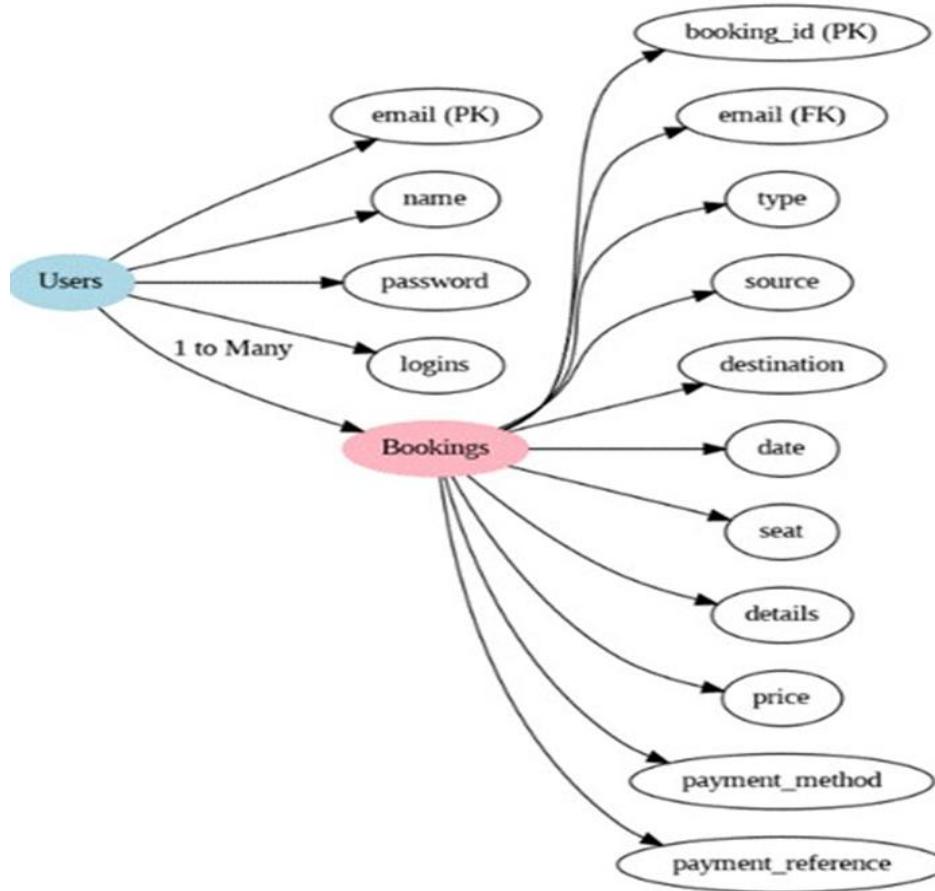
AWS Architecture Diagram

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

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

2. DynamoDB Database Initialization

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

3. SNS Notification Configuration

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

4. Backend Development using Flask

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

5. IAM Role Configuration

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

6. EC2 Instance Launch

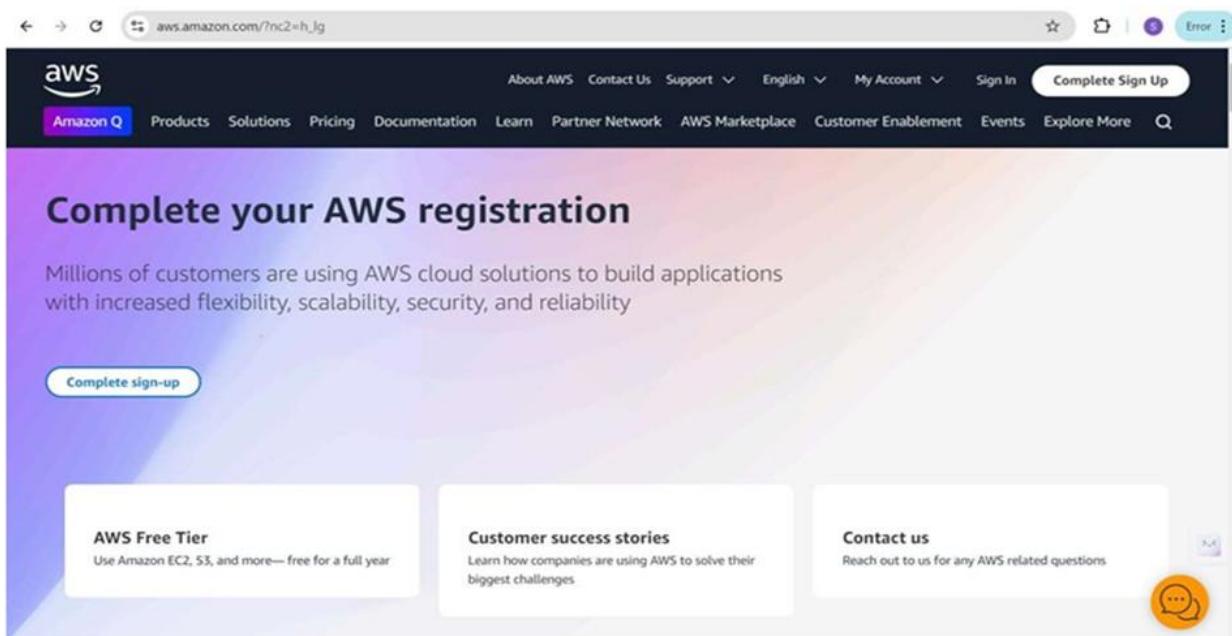
1. Activity 6.1: Start an EC2 instance to host the backend.
2. Activity 6.2: Set up security rules to allow HTTP and SSH traffic.

7. Flask Application Deployment

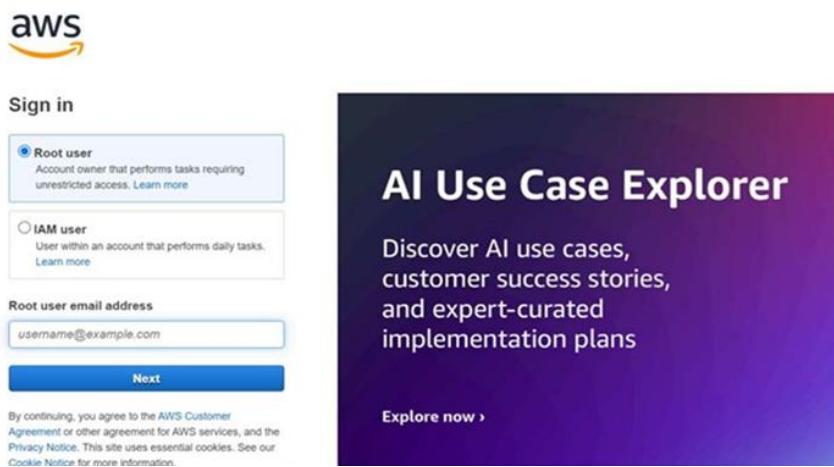
1. Activity 7.1: Upload your Flask files to the EC2 instance.
2. Activity 7.2: Launch your Flask server from the instance.

8. Testing and Launch:

1. AWS Account Setup and Login



2. Log in to the AWS Management Console



After 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

The screenshot shows the AWS IAM Dashboard. On the left, there's a sidebar with navigation links like 'Identity and Access Management (IAM)', 'Access management', 'Access reports', and 'CloudShell'. The main area has a blue header bar with 'New access analyzers available' and a 'Create new analyzer' button. Below it, the 'IAM Dashboard' section includes 'Security recommendations' (with items like 'Root user has MFA' and 'Add MFA for yourself'), 'AWS Account' (with account ID 211125402868), 'IAM resources' (showing 1 user, 4 roles, 0 policies, and 0 identity providers), and 'Quick Links' for managing security credentials.

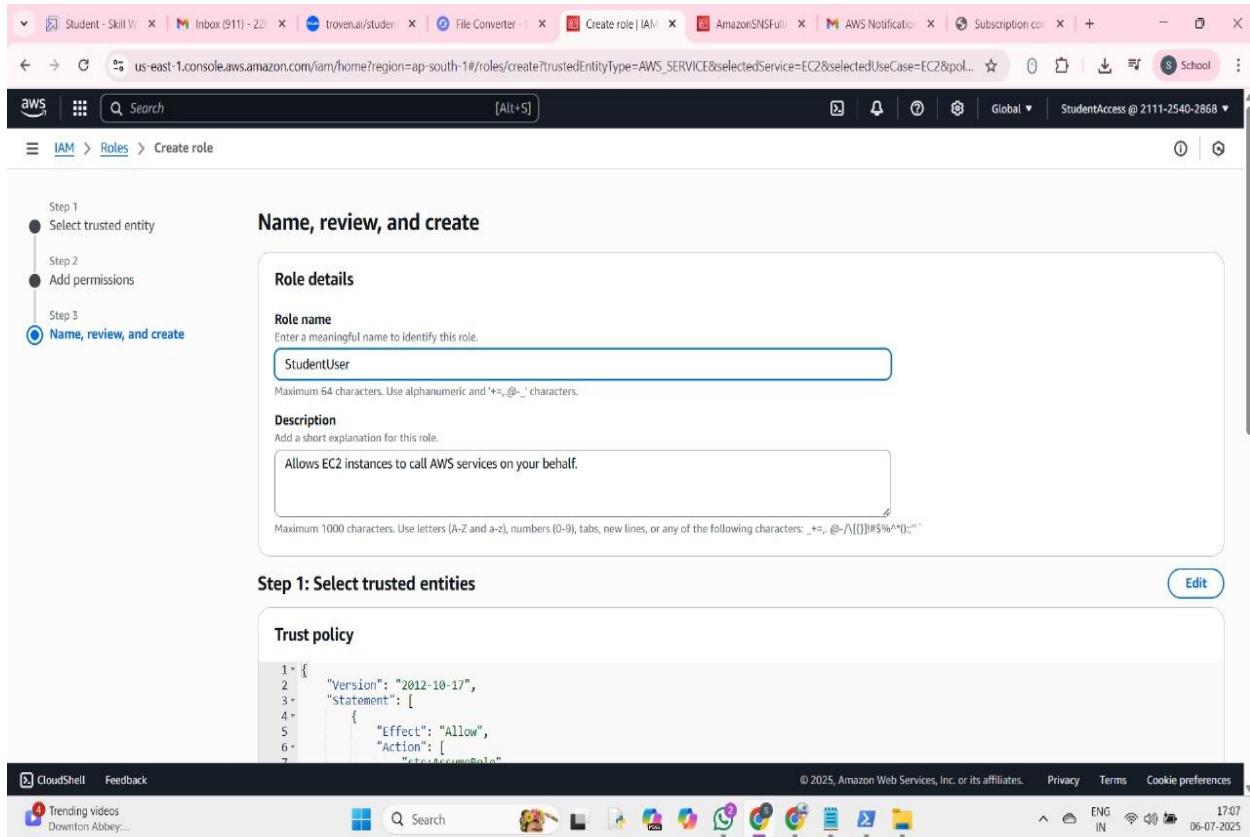
This screenshot shows the 'Create role' wizard at Step 1: 'Select trusted entity'. It lists three options: 'Select trusted entity' (selected), 'Add permissions', and 'Name, review, and create'. A dropdown menu titled 'Filter service or use case' is open, showing 'Commonly used services' with 'EC2' selected and highlighted. Other services listed include Lambda, Amazon Aurora, Amazon EMR, Amazon OpenSearch, Amazon Q, Amplify, API Gateway, AppFabric, Application Auto Scaling, Application Discovery Service, Application Migration Service, and AppStream 2.0. A tooltip for 'Web identity' explains it allows federated users from external providers to assume the role.

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

The screenshot shows the AWS IAM 'Create role' wizard at Step 2: 'Add permissions'. The title is 'Add permissions' with a 'Info' link. A sub-section titled 'Permissions policies (3/1058) Info' displays a list of AWS managed policies. The list includes:

Policy name	Type	Description
AdministratorAccess	AWS managed - job function	Provides full access to AWS services an...
AdministratorAccess-Amplify	AWS managed	Grants account administrative permisi...
AdministratorAccess-AWSElasticBeanstalk	AWS managed	Grants account administrative permisi...
AIOpsAssistantPolicy	AWS managed	Provides ReadOnly permissions requir...
AIOpsConsoleAdminPolicy	AWS managed	Grants full access to Amazon AI Opera...
AIOpsOperatorAccess	AWS managed	Grants access to the Amazon AI Opera...
AIOpsReadOnlyAccess	AWS managed	Grants ReadOnly permissions to the A...
AlexaForBusinessDeviceSetup	AWS managed	Provide device setup access to AlexaFo...
AlexaForBusinessFullAccess	AWS managed	Grants full access to AlexaForBusiness ...

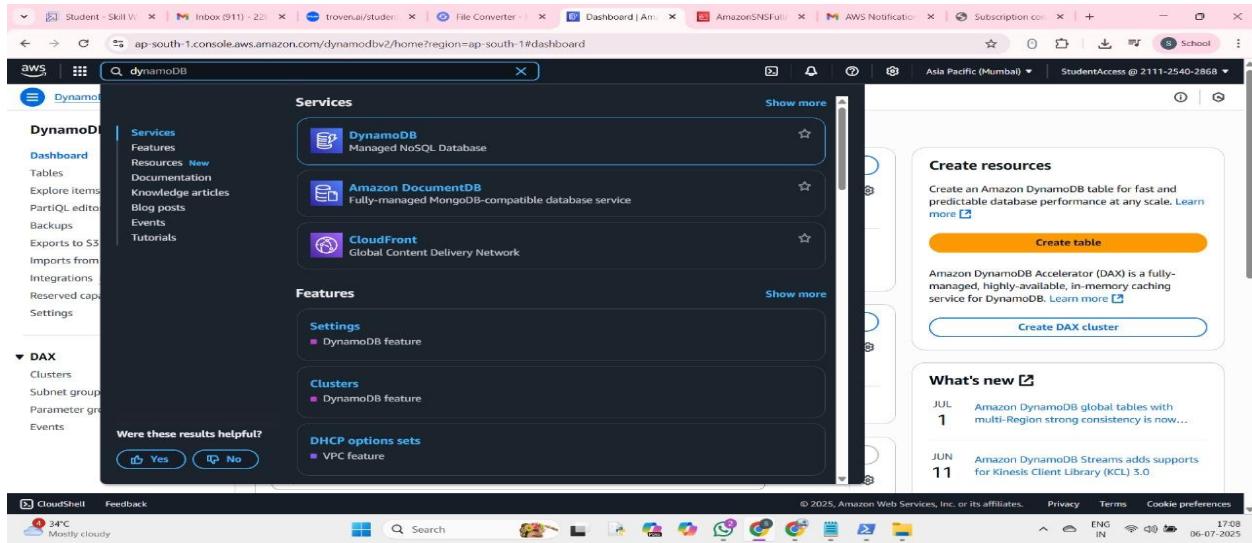
The sidebar on the left shows the steps: Step 1 (Select trusted entity), Step 2 (Add permissions, which is selected), and Step 3 (Name, review, and create). The bottom of the screen shows the Windows taskbar with various pinned icons and system status information.



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.

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.

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.

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.

Table settings

Default settings
The fastest way to create your table. You can modify most of these settings after your table has been created. To learn more about table settings, see Table settings.

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

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' step, a table named 'bookings' is being created with a primary key 'user_email' of type String. A sort key 'booking_date' is also defined. In the 'Table settings' step, the 'Default settings' tab is selected, indicating the creation of a new table.

The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. A table named 'travelgo_user' is being created with a primary key 'train_number' of type String. A sort key 'Enter the sort key name' is also defined. In the 'Table settings' step, the 'Default settings' tab is selected, indicating the creation of a new table.

The trains table was created successfully.

DynamoDB

Tables (5) Info

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite
Bookings	Active	user_email (\$)	booking_date (\$)	0	0	Off	☆
MovieMagic_Bookings	Active	booking_id (\$)	-	0	0	Off	☆
MovieMagic_Users	Active	email (\$)	-	0	0	Off	☆
trains	Active	train_number (\$)	-	0	0	Off	☆
travelgo_user	Active	email (\$)	-	0	0	Off	☆

CloudShell Feedback 34°C Mostly cloudy © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG IN 17:11 06-07-2025

5. SNS Notification Setup

The screenshot shows the AWS SNS console homepage. On the left sidebar, there are links for Services, Dashboard, Topics, Subscriptions, Mobile (Push notifications, Text messaging), and Tutorials. The main content area has two sections: 'Services' and 'Features'. Under 'Services', there are three items: 'Simple Notification Service' (SNS managed message topics for Pub/Sub), 'Route 53 Resolver' (Resolve DNS queries in your Amazon VPC and on-premises network), and 'Route 53' (Scalable DNS and Domain Name Registration). Under 'Features', there are three items: 'Events' (ElastiCache feature), 'SMS' (AWS End User Messaging feature), and 'Hosted zones' (Route 53 feature). A modal window titled 'Create topic' is open on the right side, showing fields for 'Topic name' (with a placeholder 'MyTopic') and a 'Next step' button. At the bottom of the page, there is a feedback section asking 'Were these results helpful?' with 'Yes' and 'No' buttons, and a footer with copyright information and navigation links.

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

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

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

MyTopic

Next step

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

Start with an overview

CloudShell Feedback

29°C Mostly cloudy

Search

ENG IN 01-07-2025

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

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

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

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

Name: MyTopic

Maximum 256 characters, alphanumeric characters, hyphens (-) and underscores (_).

Display name: Travelgo

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.

CloudShell Feedback

Light rain At night

Search

ENG IN 17:12 06-07-2025

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 an AWS SNS topic. It includes sections for Access policy, Data protection policy, Delivery policy (HTTP/S), Delivery status logging, Tags, and Active tracing. Each section has an 'optional' note and an 'Info' link. At the bottom right are 'Cancel' and 'Create topic' buttons.

- ▶ **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.

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

Cancel

Create topic

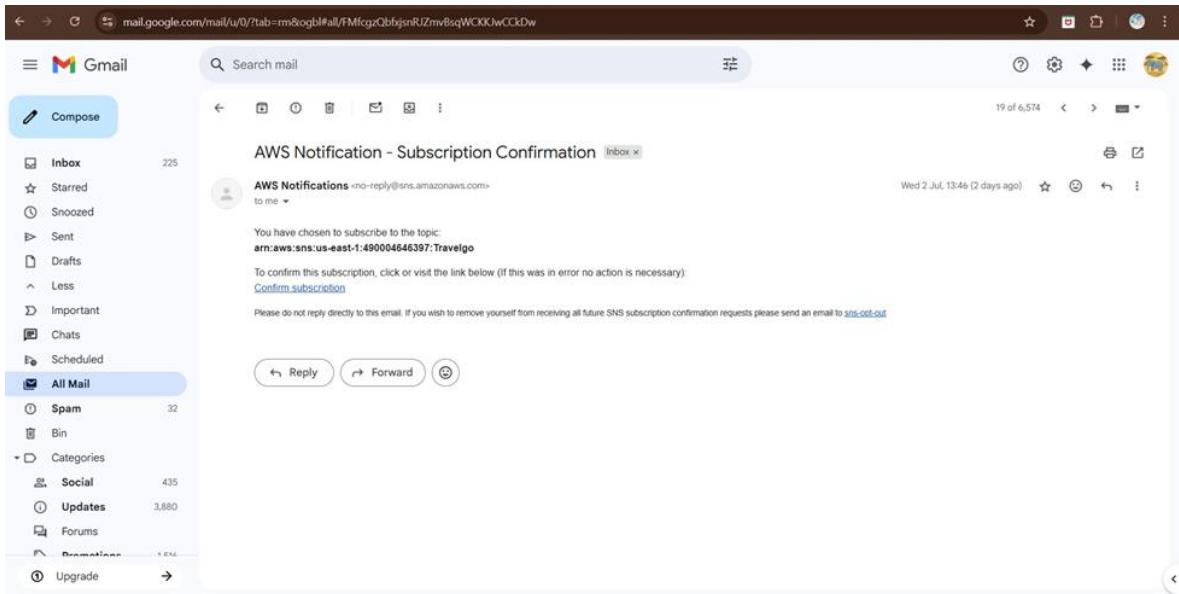
4. Click on create subscription.

The screenshot shows the 'Create subscription' configuration page for an AWS SNS topic. It includes sections for Subscription filter policy and Redrive policy (dead-letter queue). Each section has an 'optional' note and an 'Info' link. At the bottom right are 'Cancel' and 'Create subscription' buttons.

- ▶ **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.

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
61     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        return render_template('dashboard.html', username=user_email, bookings=bookings)
109
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:

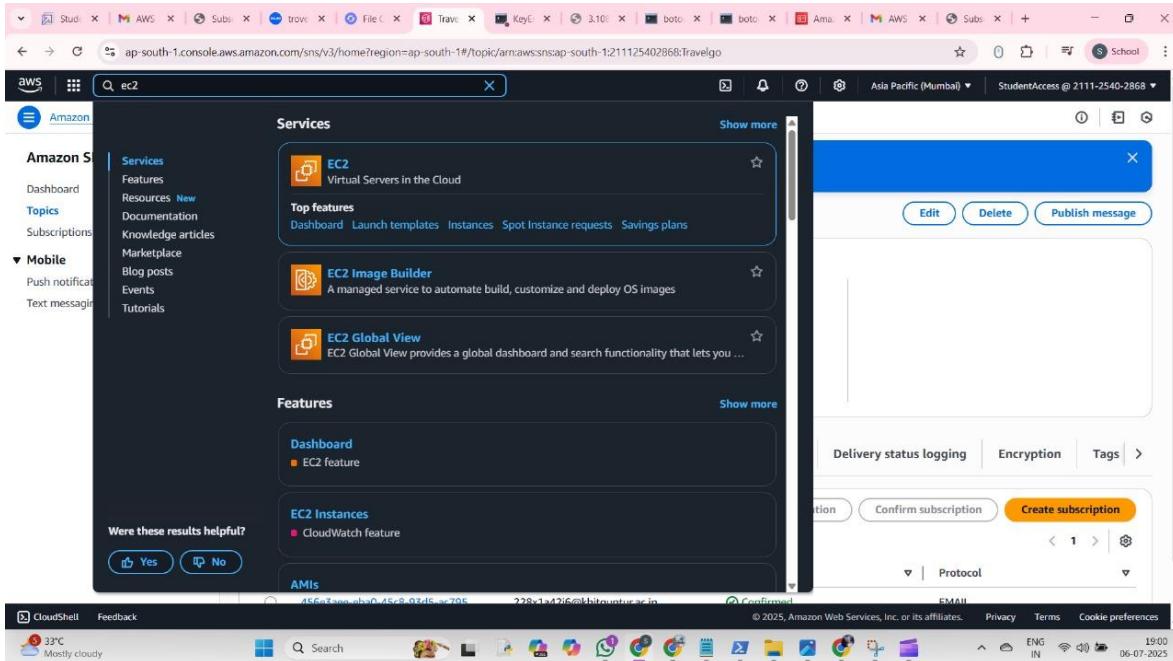
```
518     if __name__ == '__main__':
519         # IMPORTANT: In a production environment, disable debug mode and specify a production-ready host.
520         app.run(debug=True, host='0.0.0.0')
```

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. [6. EC2 Instance](#)

Setup:

Harini-devx09 Update app.py		
static	Initial commit: TravelGo project uploaded	3 days ago
templates	Initial commit: TravelGo project uploaded	3 days ago
venv	Initial commit: TravelGo project uploaded	3 days ago
venv_new	Initial commit: TravelGo project uploaded	3 days ago
README.md	First commit	3 days ago
app.py	Update app.py	yesterday

Launch an EC2 instance to host the Flask application.



1. Click on launch Instance:

The screenshot shows the AWS EC2 Instances dashboard. On the left, there's a sidebar with options like EC2 Dashboard, EC2 Global View, Events, Instances (which is selected), Instance Types, Launch Templates, Spot Requests, and Savings Plans. The main area has tabs for Instances and Info. It displays a search bar, filters for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IPv4 DNS. A message says "No instances" and "You do not have any instances in this region". There's a prominent orange "Launch instances" button.

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.

The screenshot shows the "Launch an instance" wizard. At the top, it says "Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below." Below this, there are two main sections: "Name and tags" and "Application and OS Images (Amazon Machine Image)". In the "Name and tags" section, the name "TravelGoproject" is entered. In the "Application and OS Images (Amazon Machine Image)" section, a search bar is shown, followed by a grid of OS icons: Amazon Linux, macOS, Ubuntu, Windows, Red Hat, SUSE Linux, and Debian. To the right, there's a summary panel with fields for "Number of instances" (set to 1), "Software Image (AMI)" (Amazon Linux 2023 AMI 2023.7.2...), "Virtual server type (instance type)" (t2.micro), "Firewall (security group)" (New security group), and "Storage (volumes)" (1 volume(s) - 8 GiB). At the bottom right of the summary panel is an orange "Launch instance" button.

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.

The screenshot shows the AWS EC2 Instances page with a green success message: "Successfully initiated launch of instance i-0f95c15501f00bcad". Below this, there's a "Launch log" section and a "Next Steps" summary. The summary includes links for "Create billing usage alerts", "Connect to your instance", "Connect an RDS database", "Create EBS snapshot policy", "Manage detailed monitoring", "Create Load Balancer", "Create AWS budget", and "Manage CloudWatch alarms". The status bar at the bottom indicates "Hot days ahead 34°C".

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 page for a specific security group. It displays an "Inbound rules" table with the following data:

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-09d54498e961354af	HTTPS	TCP	443	Custom	0.0.0.0/0
sgr-089fcf1743166b570	SSH	TCP	22	Custom	0.0.0.0/0
sgr-0acec6809fe0352e2	HTTP	TCP	80	Custom	0.0.0.0/0
-	Custom TCP	TCP	5000	Anywh...	0.0.0.0/0

A note at the bottom states: "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 visible at the bottom right.

The screenshot shows the AWS EC2 Security Groups console. A green success message at the top right states: "Inbound security group rules successfully modified on security group (sg-0fcf7b578ac8848ff | launch-wizard-4) Details". The main page displays the details for the security group "sg-0fcf7b578ac8848ff - launch-wizard-4". The "Details" section includes fields for Security group name (launch-wizard-4), Security group ID (sg-0fcf7b578ac8848ff), Description (launched-wizard-4 created 2025-07-06T11:44:38.633Z), Owner (211125402868), Inbound rules count (4 Permission entries), and Outbound rules count (1 Permission entry). Below this, there are tabs for Inbound rules, Outbound rules, Sharing - new, VPC associations - new, and Tags. The Inbound rules table lists four entries:

Name	Security group rule ID	IP version	Type	Protocol	Port range
-	sgr-02e856c0cb571b2a3	IPv4	Custom TCP	TCP	5000
-	sgr-08ea015b2a212a07f	IPv4	HTTP	TCP	80
-	sgr-064a030a5933672a5	IPv4	SSH	TCP	22

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.

Screenshot of the AWS EC2 Instances page showing a list of running t2.micro instances. The instance 'TravelGoproject' (i-0f93c15501f00bca4) is selected. A context menu is open over this instance, with the 'Modify IAM role' option highlighted.

Instances (1/4) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm s
AWSPrepZone	i-06ead1e14a4eb83e	Running	t2.micro	OK	
check	i-0fd873ce2b8502c8f	Running	t2.micro	OK	
Stocker	i-0b8aace42901dd045	Running	t2.micro	OK	
TravelGoproject	i-0f93c15501f00bca4	Running	t2.micro	OK	Initializing

Actions ▾ **Launch instances** ▾

- Instance diagnostics
- Instance settings
- Networking
- Security** ▾
 - ec2-15-2(1)
 - ec2-15-2(2)
 - ec2-15-2(3)
- Image and templates
- Monitor and troubleshoot

Security details

IAM Role: - Owner ID: 211125402868 Launch time: Sun Jul 06 2025 17:17:29 GMT+0530 (India Standard Time)

Security groups: sg-0fcf7b578ac8848ff (launch-wizard-4)

Inbound rules

Screenshot of the 'Modify IAM role' configuration page for the selected instance.

Modify IAM role Info

Attach an IAM role to your instance.

Instance ID: i-0f93c15501f00bca4 (TravelGoproject)

IAM role: Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

StudentUser

The screenshot shows the AWS EC2 Instances page. A green success message at the top states: "Successfully attached StudentUser to instance i-0f93c15501f00bca4". Below this, a table lists four instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IP
AWSPrepZone	i-06ead1e14a4eb83e	Running	t2.micro	2/2 checks passed	View alarms	ap-south-1b	ec2-15-2(1)
check	i-0fd873ce2b8502c8f	Running	t2.micro	2/2 checks passed	View alarms	ap-south-1b	ec2-15-2(1)
Stocker	i-0b8aaece42901dd045	Running	t2.micro	2/2 checks passed	View alarms	ap-south-1b	ec2-15-2(1)

The instance **i-0f93c15501f00bca4 (TravelGoproject)** is selected. Its details are shown in the main pane, including its Public IPv4 address (3.108.51.57), Instance state (Running), and Public DNS (ec2-3-108-51-57.ap-south-1.compute.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 page for the instance **i-0f93c15501f00bca4**. It provides instructions for connecting using an SSH client:

- 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:
`ec2-3-108-51-57.ap-south-1.compute.amazonaws.com`

An example command is provided:

```
ssh -i "Travelgo.pem" ec2-user@ec2-3-108-51-57.ap-south-1.compute.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.

- sudo yum install git -y
 - git clone your_repository_url
 - cd your_project_directory

```

[ec2-user@ip-172-31-10-246:~/TravelGo]
(3/8): perl-Error-0.17029-5_amzn2023.0.3.noarch.rpm          1.1 MB/s | 41 kB  00:00
(4/8): git-core-2.47.1-1.amzn2023.0.3.x86_64.rpm           42 kB/s | 4.5 MB  00:00
(5/8): perl-File-Find-1.37-477_amzn2023.0.7.noarch.rpm      579 kB/s | 25 kB  00:00
(6/8): perl-Git-2.47.1-1.amzn2023.0.3.noarch.rpm            949 kB/s | 40 kB  00:00
(7/8): perl-TermReadKey-2.38-9_amzn2023.0.2.x86_64.rpm       1.7 MB/s | 36 kB  00:00
(8/8): perl-lib-0.65-477_amzn2023.0.7.x86_64.rpm            680 kB/s | 15 kB  00:00
Total                                         46 MB/s | 7.5 MB  00:00

Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing : 1/1
Installing  : git-core-2.47.1-1.amzn2023.0.3.x86_64          1/1
Installing  : git-core-doc-2.47.1-1.amzn2023.0.3.noarch        2/8
Installing  : perl-lib-0.65-477_amzn2023.0.7.x86_64          3/8
Installing  : perl-TermReadKey-2.38-9_amzn2023.0.2.noarch      4/8
Installing  : perl-file-Find-1.37-477_amzn2023.0.7.noarch      5/8
Installing  : perl-Git-2.47.1-1.amzn2023.0.3.noarch            6/8
Installing  : perl-lib-0.65-477_amzn2023.0.7.x86_64          7/8
Installing  : git-2.47.1-1.amzn2023.0.3.x86_64                8/8
Running scriptlets: git-2.47.1-1.amzn2023.0.3.x86_64          1/8
Verifying   : git-2.47.1-1.amzn2023.0.3.x86_64                2/8
Verifying   : git-core-2.47.1-1.amzn2023.0.3.noarch            3/8
Verifying   : perl-Error-1.9.17029-5_amzn2023.0.2.noarch      4/8
Verifying   : perl-file-Find-1.37-477_amzn2023.0.7.noarch      5/8
Verifying   : perl-Git-2.47.1-1.amzn2023.0.3.noarch            6/8
Verifying   : perl-TermReadKey-2.38-9_amzn2023.0.2.x86_64      7/8
Verifying   : perl-lib-0.65-477_amzn2023.0.7.x86_64          8/8

Installed:
git-2.47.1-1.amzn2023.0.3.x86_64                  git-core-2.47.1-1.amzn2023.0.3.x86_64
git-core-doc-2.47.1-1.amzn2023.0.3.noarch          perl-Error-1.9.17029-5_amzn2023.0.2.noarch
perl-file-Find-1.37-477_amzn2023.0.7.noarch        perl-Git-2.47.1-1.amzn2023.0.3.noarch
perl-TermReadKey-2.38-9_amzn2023.0.2.x86_64        perl-lib-0.65-477_amzn2023.0.7.x86_64

Complete!
[ec2-user@ip-172-31-31-5 ~]$ git clone https://github.com/Amulya-456/TravelGo.git
Cloning into 'TravelGo'...
remote: Enumerating objects: 3698, done.
remote: Counting objects: 100% (3698/3698), done.
remote: Compressing objects: 100% (2112/2112), done.
remote: Writing objects: 100% (3698/3698), (delta 1560), pack-reused 0 (from 0)
Receiving objects: 100% (3698/3698), 11.53 MB / 28.66 MiB/s, done.
Resolving deltas: 100% (1602/1602), done.
[ec2-user@ip-172-31-31-5 ~]$ cd TravelGo
[ec2-user@ip-172-31-31-5 TravelGo]$ client_loop: send disconnect: Connection reset

```

System tray icons: 33°C, Mostly cloudy, Search, File Explorer, Task View, Taskbar icons, Network, Battery, ENG IN, Wi-Fi, 19:03, 06-07-2025.

- sudo yum install python3 -y
- sudo yum install python3-pip -y

```

[ec2-user@ip-172-31-10-246:~/TravelGo]
Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Collecting click==8.1.3
  Downloading click-8.1.8-py3-none-any.whl (98 kB) 98 kB 87.0 MB/s
Collecting jinja2>=3.1.2
  Downloading jinja2-3.1.6-py3-none-any.whl (134 kB) 134 kB 87.0 MB/s
Collecting zipp>=3.20
  Downloading zipp-3.23.0-py3-none-any.whl (10 kB)
Installing collected packages: blinker, markupsafe, werkzeug, jinja2, itsdangerous, importlib-metadata, click, flask
Successfully installed blinker-1.9.0 click-8.1.8 flask-3.1.1 importlib-metadata-3.7.0 itsdangerous-2.2.0 jinja2-3.1.6 markupsafe-3.0.2 werkzeug-3.1.3 zipp-3.23.0
[ec2-user@ip-172-31-10-246 TravelGo]$ pip install boto3
Requirement already satisfied: boto3<1.40.0,>=1.39.3 in /usr/lib/python3.9/site-packages (from boto3) (0.10.0)
Collecting botocore<1.40.0,>=1.39.3
  Downloading botocore-1.39.3-py3-none-any.whl (13.8 kB) 13.8 kB 58.8 MB/s
Collecting s3transfer<0.14.0,>=0.13.0
  Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB) 85 kB 12.0 MB/s
Requirement already satisfied: urllib3<2.0.0,>=1.26.4 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.3>botocore) (1.25.10)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.3>botocore) (2.8.1)
Requirement already satisfied: six<1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1>botocore<1.40.0,>=1.39.3>botocore) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed botocore-1.39.3 s3transfer-0.13.0
[ec2-user@ip-172-31-10-246 TravelGo]$ 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://172.31.10.246:5000
Overwriting config to ...
* Restarting with stat
* Debugger is active!
* Debugger PIN: 610-907-662
49.205.107.54 - - [06/Jul/2025 12:36:08] "GET / HTTP/1.1" 200 -
49.205.107.54 - - [06/Jul/2025 12:36:08] "GET /static/images/travel_bg.jpg HTTP/1.1" 404 -
49.205.107.54 - - [06/Jul/2025 12:36:08] "GET /favicon.ico HTTP/1.1" 404 -
49.205.107.54 - - [06/Jul/2025 12:36:13] "GET /login HTTP/1.1" 200 -
49.205.107.54 - - [06/Jul/2025 12:36:51] "GET /register HTTP/1.1" 200 -
49.205.107.54 - - [06/Jul/2025 12:37:04] "POST /register HTTP/1.1" 500 -

```

System tray icons: 33°C, Mostly cloudy, Search, File Explorer, Task View, Taskbar icons, Network, Battery, ENG IN, Wi-Fi, 19:03, 06-07-2025.

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

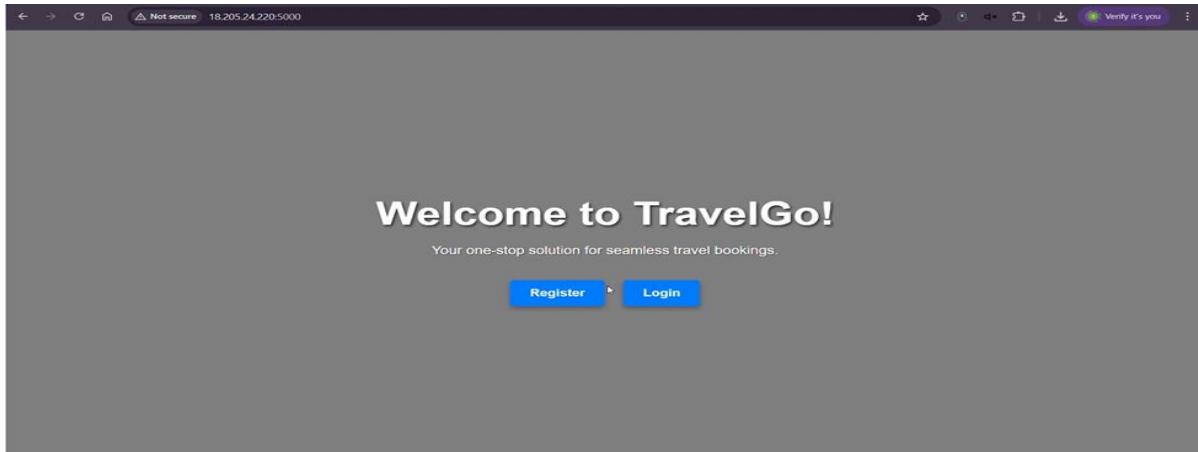
Deployment Steps Summary:

- sudo yum install git -y
- git clone your_repository_url
- cd your_project_directory

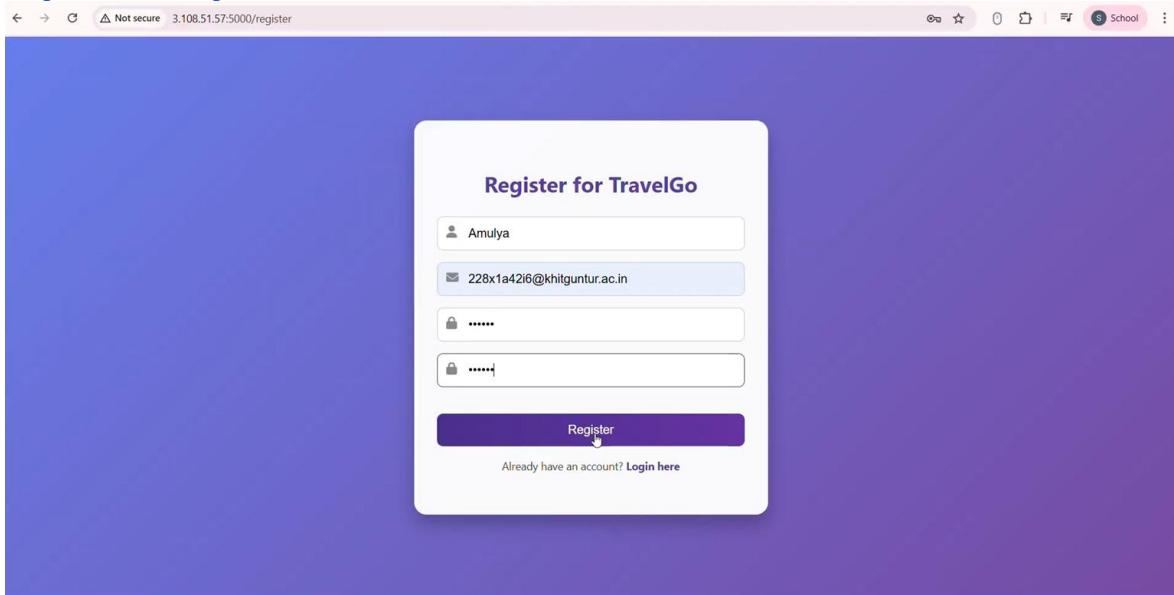
- sudo yum install python3 -y
- sudo yum install python3-pip -y
- pip install flask
- pip install boto3 • python3 app.py

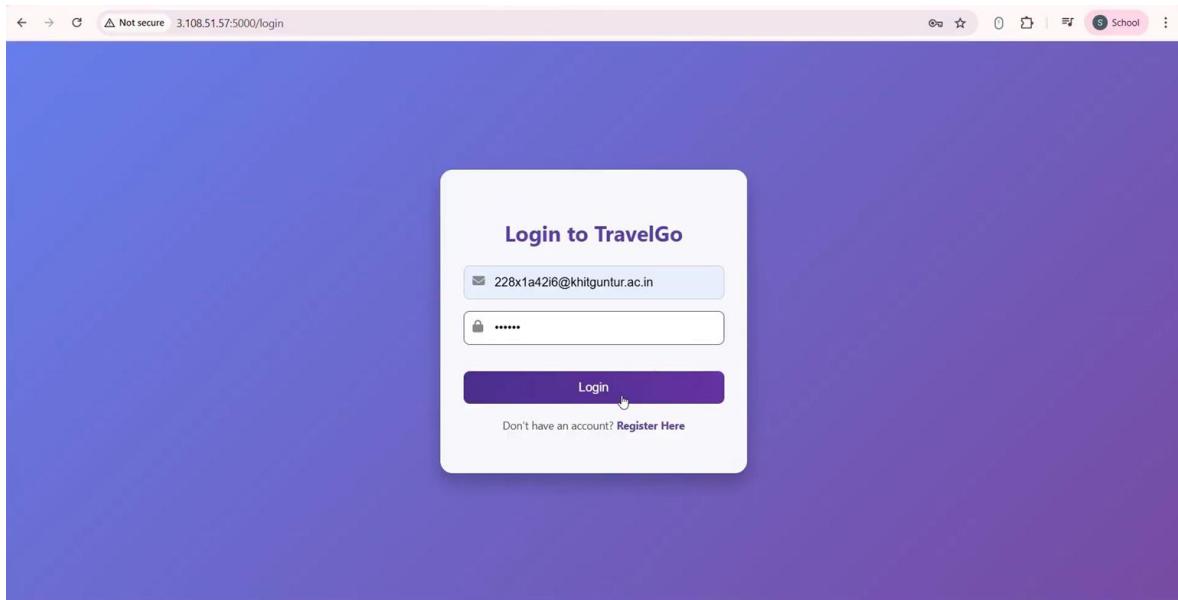
User Interface Screens:

- Homepage:



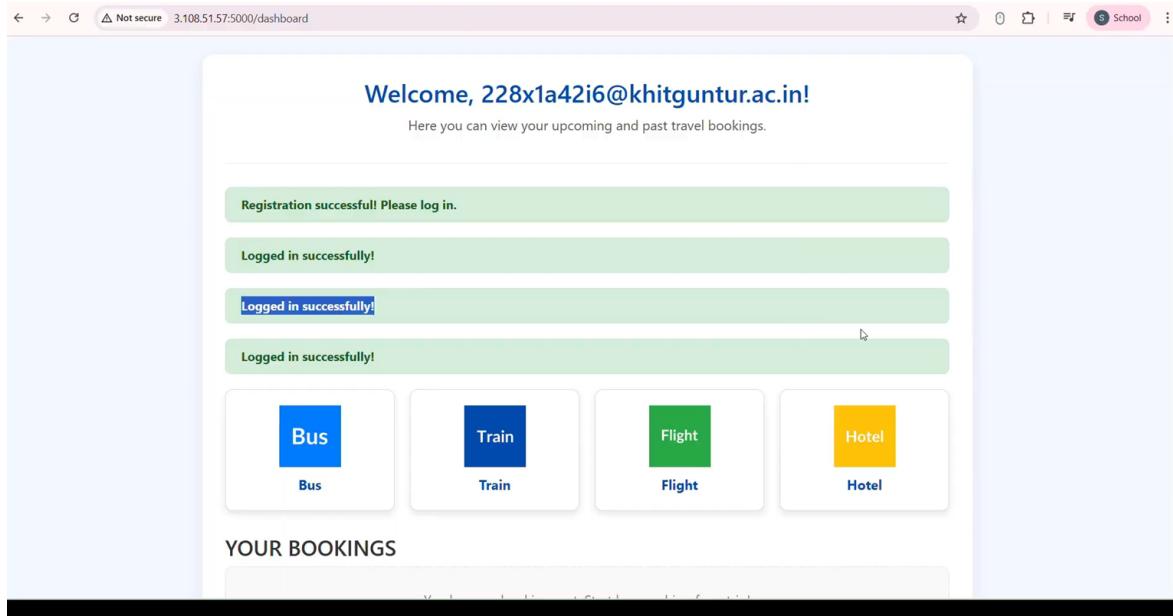
Register and Login Portals:





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!!!



- **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:**

Not secure 3.108.51.57:5000/bus

TravelGo

Home Dashboard

Search & Book Buses

Hyderabad ▾ Vijayawada ▾ 06-07-2025 ▾ 1 Search

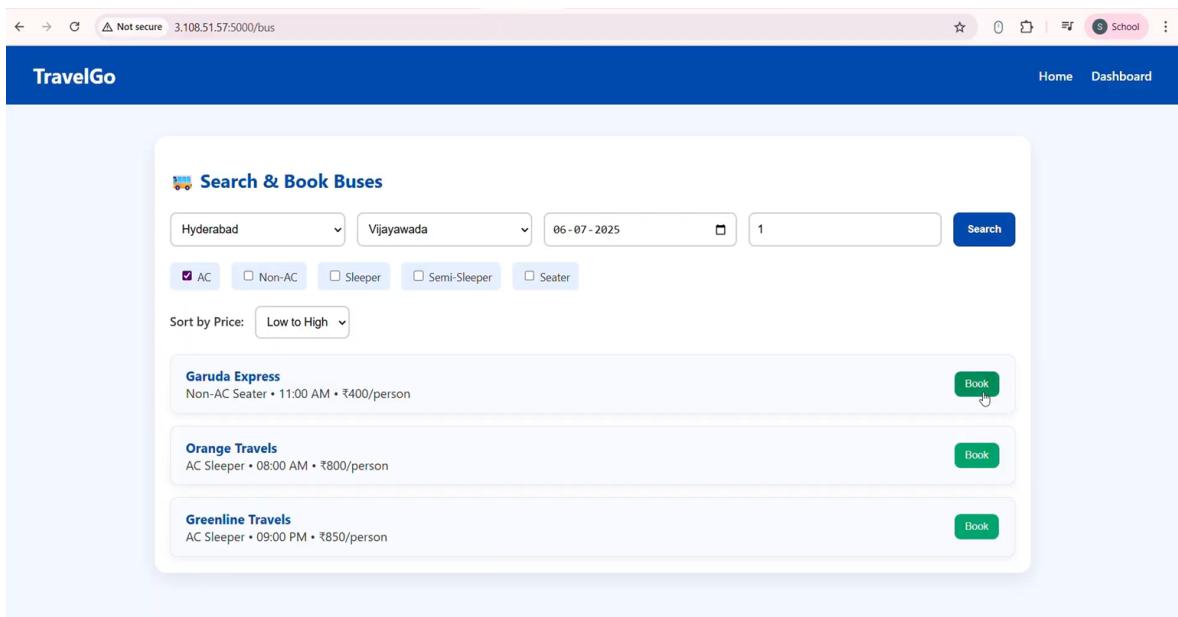
AC Non-AC Sleeper Semi-Sleeper Seater

Sort by Price: Low to High ▾

Garuda Express
Non-AC Seater • 11:00 AM • ₹400/person Book

Orange Travels
AC Sleeper • 08:00 AM • ₹800/person Book

Greenline Travels
AC Sleeper • 09:00 PM • ₹850/person Book

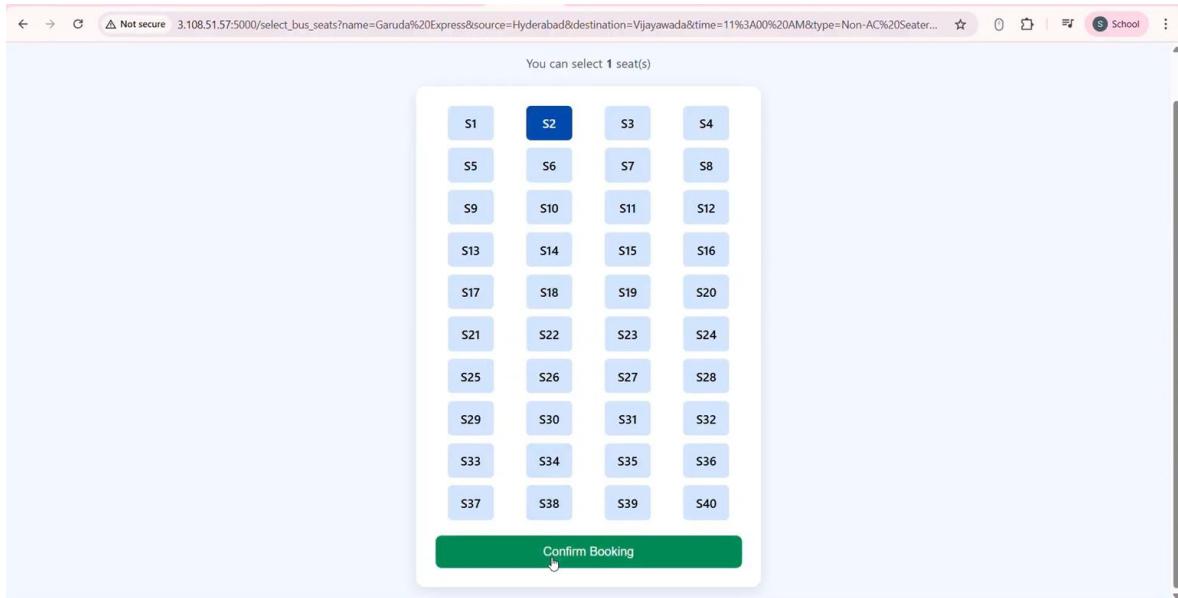


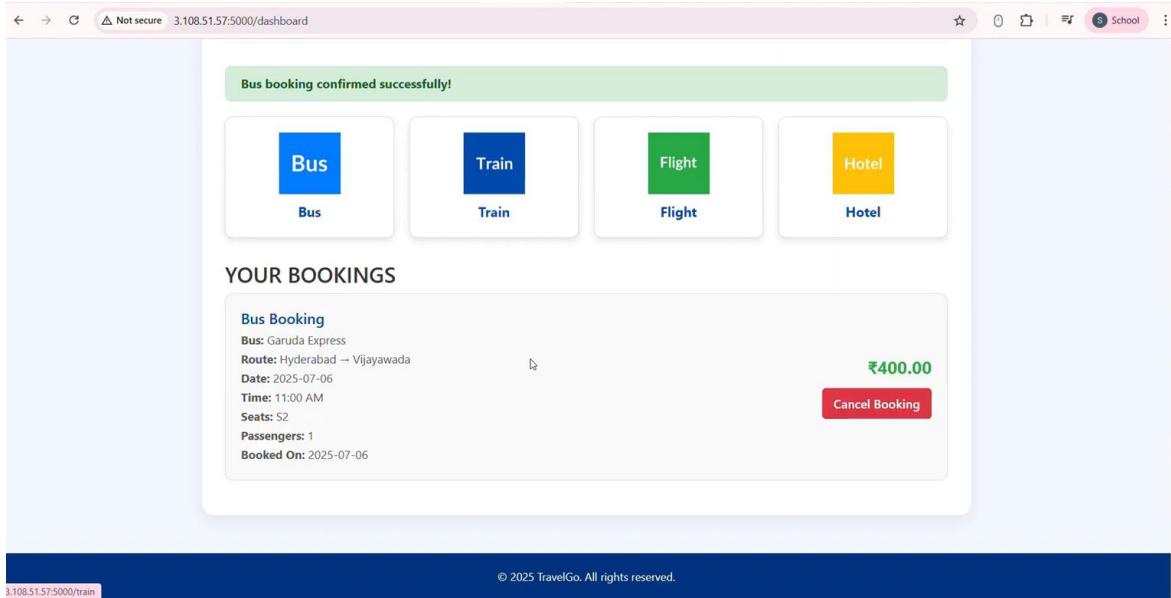
Not secure 3.108.51.57:5000/select_bus_seats?name=Garuda%20Express&source=Hyderabad&destination=Vijayawada&time=11%3A00%20AM&type=Non-AC%20Seater... ☆ 0 School

You can select 1 seat(s)

S1	S2	S3	S4
S5	S6	S7	S8
S9	S10	S11	S12
S13	S14	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





- **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.

← → ⌂ Not secure 3.108.51.57:5000/dashboard

Welcome, 228x1a42i6@khitguntur.ac.in!

Here you can view your upcoming and past travel bookings.

Bus booking confirmed successfully!

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

YOUR BOOKINGS

Bus Booking

Bus: Garuda Express
Route: Hyderabad → Vijayawada
Date: 2025-07-06
Time: 11:00 AM
Seats: S2
Passengers: 1
Booked On: 2025-07-06

₹400.00

Cancel Booking

3.108.51.57:5000/train

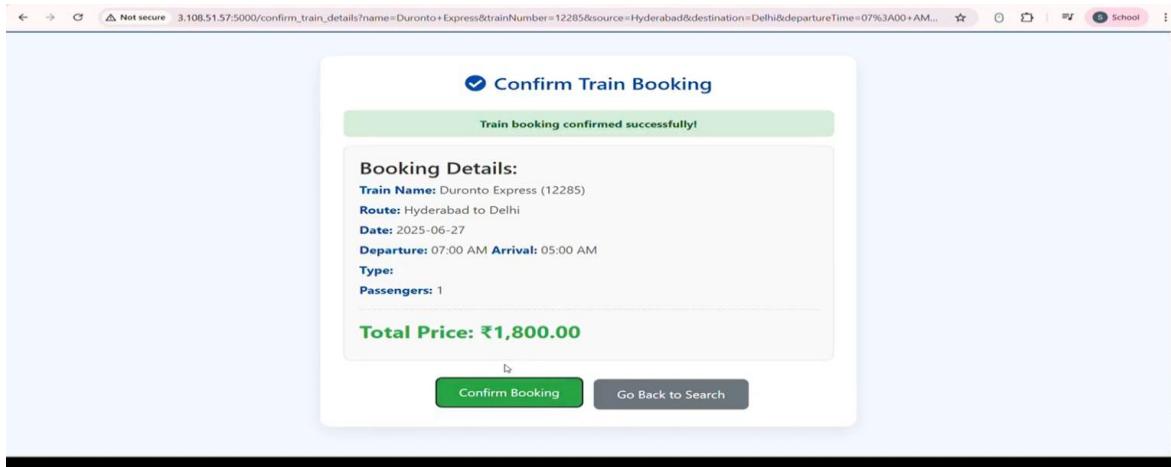
← → ⌂ Not secure 3.108.51.57:5000/train

Search & Book Trains

Hyderabad ▾ Delhi ▾ 27-06-2025 ▾ 1 Search

Duronto Express (12285)
From: Hyderabad **To:** Delhi
Departure: 07:00 AM **Arrival:** 05:00 AM
Date: 2025-06-27
Price per person: ₹1800
Total for 1: ₹1800

Book Now



- **Flight booking:**

Welcome, 228x1a42i6@khitguntur.ac.in!

Here you can view your upcoming and past travel bookings.

Bus Train Flight Hotel

YOUR BOOKINGS

Train Booking

Train: Duronto Express (12285)
Route: Hyderabad → Delhi
Date: 2025-06-27
Time: 07:00 AM - 05:00 AM
Passenger: 1
Seat: S38
Booked On: 2025-07-06

₹1,800.00 Cancel 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.

Not secure 3.108.51.57:5000/dashboard

TravelGo

Welcome, 228x1a42i6@khitguntur.ac.in!

Here you can view your upcoming and past travel bookings.

[Bus](#)

[Train](#)

[Flight](#)

[Hotel](#)

YOUR BOOKINGS

Train Booking

Train: Duronto Express (12285)
Route: Hyderabad → Delhi
Date: 2025-06-27
Time: 07:00 AM - 05:00 AM
Passengers: 1
Seats: S38
Booked On: 2025-07-06

₹1,800.00

[Cancel Booking](#)

Not secure 3.108.51.57:5000/confirm_flight_details?flight_id=FLT001&airline=IndiGo&flight_number=6E-234&source=Hyderabad&destination=Mumbai&departure=08:00... School

Confirm Your Flight Booking

Airline: IndiGo (6E-234)
Route: Hyderabad → Mumbai
Date: 2025-07-07
Departure: 08:00
Arrival: 09:30
Passengers: 1
Price/person: ₹3500
Total Price: ₹3500

[Confirm Booking](#)

- 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.

This screenshot shows the TravelGo dashboard for a user named 228x1a42i6@khitguntur.ac.in. A green notification bar at the top left says "Flight booking confirmed successfully!". Below it are four buttons: Bus, Train, Flight, and Hotel. The "Flight" button is highlighted with a green border. Underneath is a section titled "YOUR BOOKINGS" showing a flight booking for IndiGo 6E-234 from Hyderabad to Mumbai on 07-07-2025 at 08:00. The total price is ₹3,500.00 and there is a "Cancel Booking" button. The URL in the address bar is 3.108.51.57:5000/dashboard.

This screenshot shows the TravelGo hotel booking search interface. It features a search form with fields for destination (Hyderabad), check-in date (06-07-2025), check-out date (07-07-2025), guests (1), rooms (1), and a "Search" button. Below the form are filters for star ratings (5-Star selected, 4-Star, 3-Star), amenities (WiFi, Pool, Parking), and a "Sort by" dropdown set to "None". A result card for "Taj Falaknuma Palace" in Hyderabad is shown, describing it as a 5-star hotel at ₹25000/night with WiFi, Pool, and Restaurant amenities. A large green "Book" button is on the right. The URL in the address bar is 3.108.51.57:5000/hotel.

The screenshot shows a web browser window with a title bar indicating 'Not secure' and the URL '3.108.51.57:5000/confirm_hotel_details?name=Taj+Falaknuma+Palace&location=Hyderabad&checkin=2025-07-06&checkout=2025-07-07&rooms=1&guests=1...'. The main content is a white card titled 'Confirm Your Hotel Booking'. It lists the following details:

- Hotel:** Taj Falaknuma Palace
- Location:** Hyderabad
- Check-in:** 2025-07-06
- Check-out:** 2025-07-07
- Rooms:** 1
- Guests:** 1
- Price/night:** ₹25000
- Total nights:** 1
- Total Cost:** ₹25000

A large green button at the bottom right of the card says 'Confirm Booking'.

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.

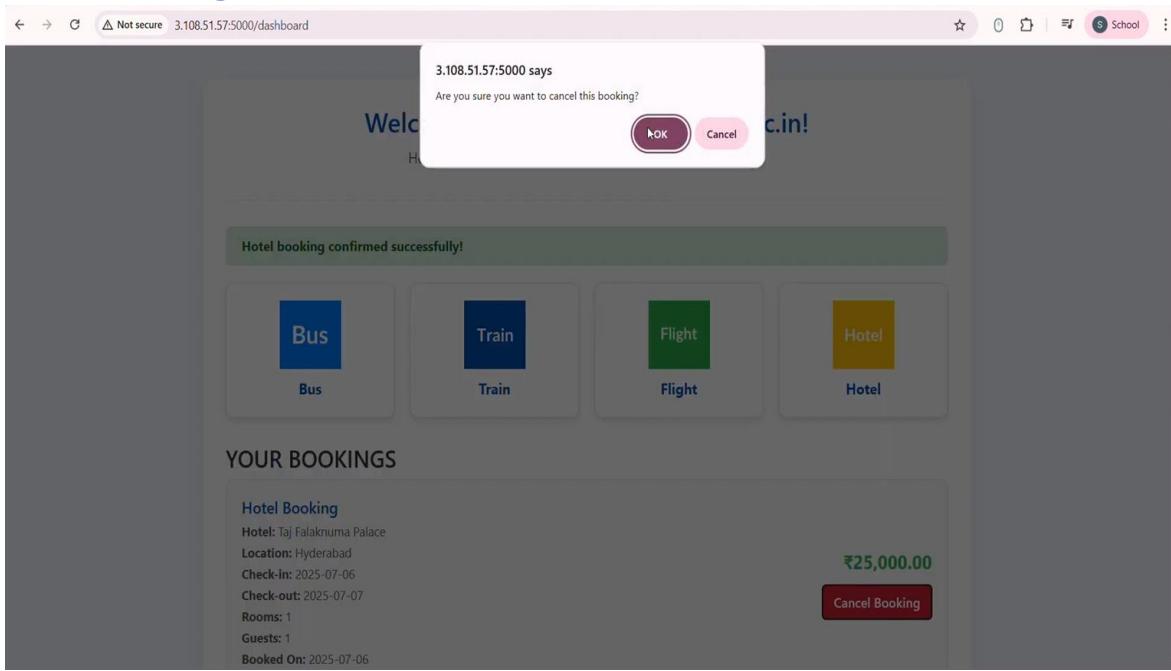
This section provides a centralized view for managing bookings efficiently. It ensures transparency by displaying every confirmed ticket in an organized format. The cancel feature updates the backend in real time, removing the booking from the list and database. This functionality enhances user control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

The screenshot shows a web browser window with a title bar indicating 'Not secure' and the URL '3.108.51.57:5000/dashboard'. The main content is a white card titled 'YOUR BOOKINGS'.

It displays three booking entries:

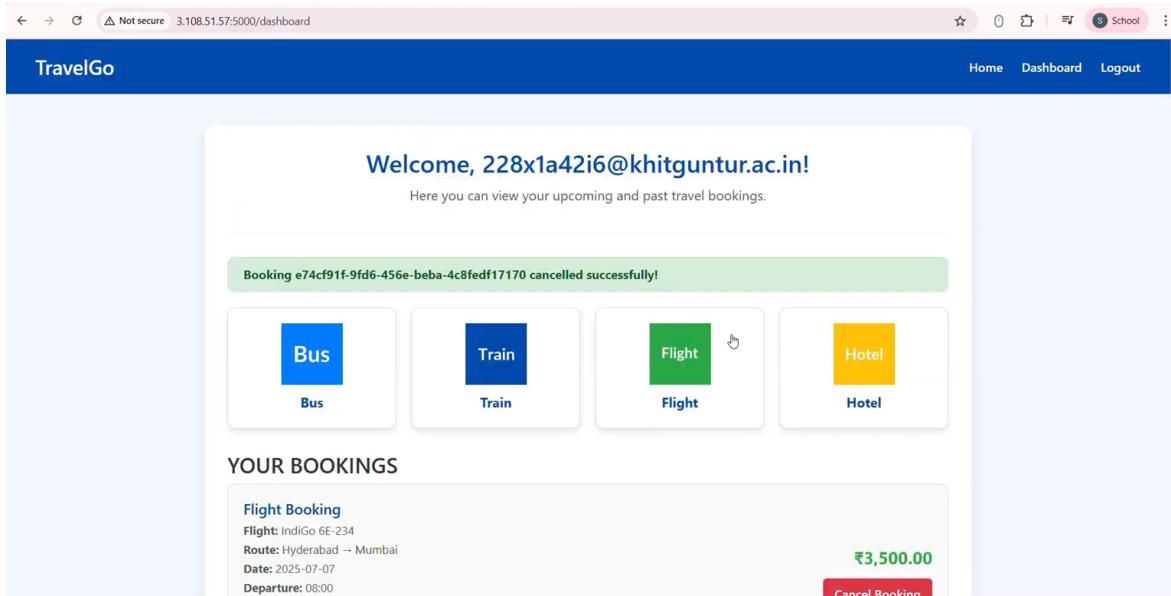
- Hotel Booking**
Hotel: Taj Falaknuma Palace
Location: Hyderabad
Check-in: 2025-07-06
Check-out: 2025-07-07
Rooms: 1
Guests: 1
Booked On: 2025-07-06
₹25,000.00
Cancel Booking
- Flight Booking**
Flight: IndiGo 6E-234
Route: Hyderabad → Mumbai
Date: 2025-07-07
Departure: 08:00
Arrival: 09:30
Passengers: 1
Booked On: 2025-07-06
₹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
₹1,800.00
Cancel Booking

Cancel Bookings:



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.



Final Conclusion – Elevating Travel with TravelGo:

The TravelGo platform has been thoughtfully designed and successfully deployed using a cloudnative, scalable architecture that meets the dynamic needs of modern travelers. By integrating key AWS services — including EC2 for robust hosting, DynamoDB for real-time and flexible data management, and SNS for instant email notifications — the platform delivers a smooth, all-inone travel booking experience.

Users can seamlessly register, log in, and book buses, trains, flights, and hotels through a unified and intuitive interface, eliminating the hassle of navigating multiple apps or sites. The backend, powered by Flask, efficiently manages user sessions, dynamic data transactions, and booking flows — ensuring performance and responsiveness at every step.

With real-time email alerts triggered by AWS SNS, users stay informed about their bookings and cancellations the moment they occur. This instant feedback loop not only boosts trust but also enhances overall usability.

In essence, TravelGo stands as a smart, reliable, and user-friendly travel assistant. It demonstrates how cloud technology can unify complex travel services into one cohesive platform — streamlining operations and elevating the end-user experience.

 **With TravelGo, travel planning is no longer a hassle—it's an experience.**

Smart. Fast. Reliable. That's the future of travel.