

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

Hardware Required:

Processor: Intel i5 or equivalent (minimum). RAM: 4 GB (8 GB recommended for Full Stack MERN). Storage: 128 GB SSD or 128 GB HDD. Internet Connectivity: High-speed internet (minimum 10 Mbps per system). Additional: Audio-visual setup for interactive sessions (microphone, speakers, etc.).

Software Required:

Processor: Intel i5 or equivalent (minimum). RAM: 4 GB (8 GB recommended for Full Stack MERN). Storage: 128 GB SSD or 128 GB HDD. Internet Connectivity: High-speed internet (minimum 10 Mbps per system). Additional: Audio-visual setup for interactive sessions (microphone, speakers, etc.).

System Required:

Projector and Audio System for presentations in all labs/classrooms Classrooms/Labs are equipped with systems or provisions for students to join sessions with their own laptops.

DESCRIPTION:

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

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

Scenarios:

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

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

Scenario 2: Real-Time Booking Confirmation with AWS SNS

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

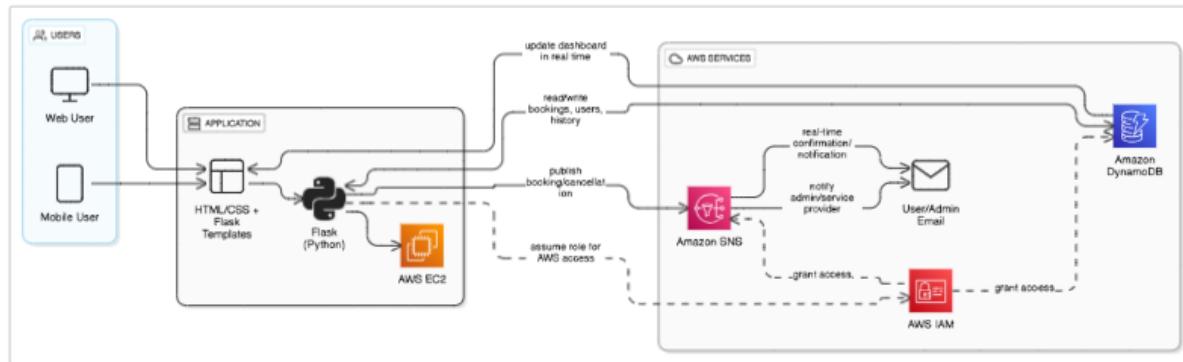
Scenario 3: Dynamic Dashboard with Personal Travel History

TravelGo features a dynamic user dashboard that displays all past and upcoming bookings for the logged-in user. For example, a user who has booked a flight and a hotel can view these bookings categorized by type, along with dates, price, and cancellation options. Flask fetches this data from AWS DynamoDB, which persistently stores all user

bookings. The dashboard UI, powered by responsive HTML/CSS and Flask templates, ensures users can review or manage bookings anytime, from any device, with real-time updates and quick cancellation workflows supported.

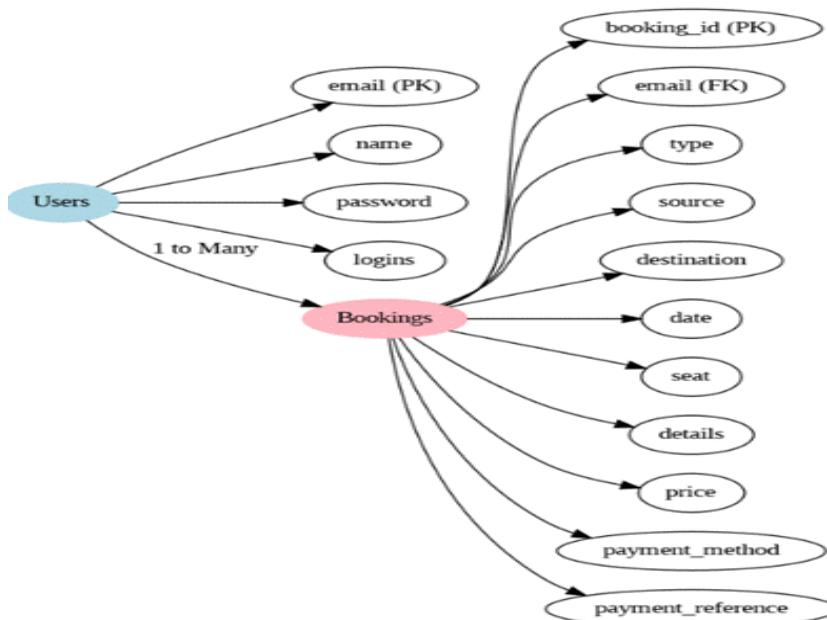
ARCHITECTURE

This AWS-based architecture powers a scalable and secure web application using Amazon EC2 for hosting the backend, with a lightweight framework like Flask handling core logic. Application data is stored in Amazon DynamoDB, ensuring fast, reliable access, while user access is managed through AWS IAM for secure authentication and control. Real-time alerts and system notifications are enabled via Amazon SNS, enhancing communication and user engagement.



ENTITY RELATIONSHIP (ER)DIAGRAM

An ER (Entity-Relationship) diagram visually represents the logical structure of a database by defining entities, their attributes, and the relationships between them. It helps organize data efficiently by illustrating how different components of the system interact and relate. This structured approach supports effective database normalization, data integrity, and simplified query design.



Pre-requisites

- **AWS Account Setup**
<https://docs.aws.amazon.com/accounts/latest/reference/getting-started.html>
- **AWS IAM (Identity and Access Management)**
<https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>
- **AWS EC2 (Elastic Compute Cloud)**
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>
- **AWS DynamoDB**
<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>
- **Amazon SNS (Simple Notification Service)**
<https://docs.aws.amazon.com/sns/latest/dg/welcome.html>
- **Git Documentation**
<https://git-scm.com/doc>
- **Visual Studio Code Download**
<https://code.visualstudio.com/download>

PROJECT WORKFLOW:

Milestone 1. Backend Development and Application Setup

- Develop the Backend Using Flask.
- Integrate AWS Services Using boto3.

Milestone 2. AWS Account Setup and Login

- Set up an AWS account if not already done.
- Log in to the AWS Management Console

Milestone 3. DynamoDB Database Creation and Setup

- Create a DynamoDB Table.
- Configure Attributes for User Data and Book Requests.

Milestone 4. SNS Notification Setup

- Create SNS topics for book request notifications.
- Subscribe users and library staff to SNS email notifications.

Milestone 5. IAM Role Setup

- Create IAM Role
- Attach Policies

Milestone 6. EC2 Instance Setup

- Launch an EC2 instance to host the Flask application.
- Configure security groups for HTTP, and SSH access.

Milestone 7. Deployment on EC2

- Upload Flask Files

- Run the Flask App

Milestone 8. Testing and Deployment

- Conduct functional testing to verify user registration, login, book requests, and notifications.

MILESTONE 1: WEB APPLICATION DEVELOPMENT AND SETUP

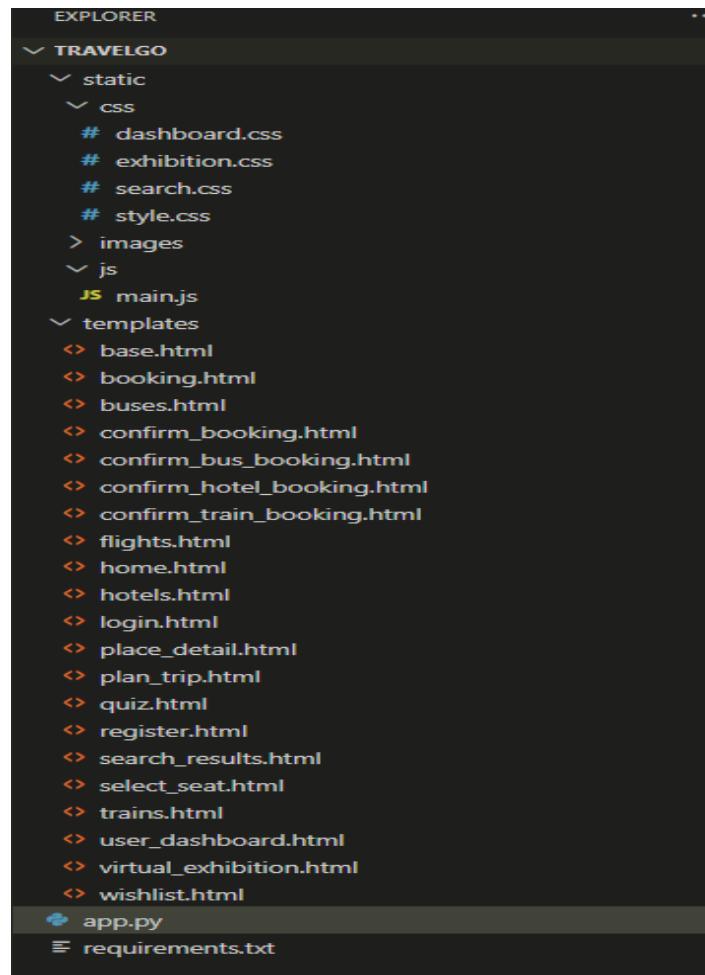
Backend Development and Application Setup focuses on establishing the core structure of the application. This includes configuring the backend framework, setting up routing, and integrating database connectivity. It lays the groundwork for handling user interactions, data management, and secure access.

Important Instructions:

- Start by creating the necessary HTML pages and Flask routes (app.py) to build the core functionality of your application.
- During the initial development phase, store and retrieve data using Python dictionaries or lists locally. This will allow you to design, test, and validate your application logic without external database dependencies.
- Ensure your app runs smoothly with local data structures before integrating any cloud services.

LOCAL DEPLOYMENT

- File Explorer:



Description: Organize the project with HTML templates for each feature (e.g., login, wishlist, quiz, checkout) under the templates folder and manage backend logic in application.py.

- **Activity 3.1:Develop the Backend Using Flask**

Import libraries:

```
app.py > ...
1 from flask import Flask, render_template, request, redirect, session, url_for, flash, jsonify, abort
2 from pymongo import MongoClient
3 from werkzeug.security import generate_password_hash, check_password_hash
4 from datetime import datetime
5 from functools import wraps
6 from flask_caching import Cache
7 from bson.objectid import ObjectId # ✅ Needed for _id lookups
8 import os
9 |
```

Description: Import essential Flask modules for web handling, Boto3 for AWS integration, Werkzeug for password hashing, and datetime for timestamp management

Flask App Initialization:

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

Description: initialize the Flask application instance using Flask(__name__) to start building the web app.

Database Configuration:

```
# Initialize DynamoDB resource
dynamodb = boto3.resource('dynamodb', region_name='ap-south-1')

# DynamoDB Tables
users_table = dynamodb.Table('Users') # Ensure the 'Users' table
requests_table = dynamodb.Table('Requests') # Ensure the 'Reques
```

Description: Connect to DynamoDB using Boto3 and define references to the UserTable and WishlistTable for user and wishlist data operations.

Home and registration routes:

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

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

```
# Routes
@app.route('/')
def home():
    return render_template('home.html', logged_in='email' in session)

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        email = request.form['email']
        username = request.form['username']
        password = request.form['password']

        if users_col.find_one({"email": email}):
            return render_template('register.html', error="Email already registered.")

        hashed_password = generate_password_hash(password)
        users_col.insert_one({
            'email': email,
            'username': username,
            'hashed_password': hashed_password,
            'login_count': 0
        })
        return redirect(url_for('login'))

    return render_template('register.html')

# --- Login ---
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = users_col.find_one({"email": email})

        if user and check_password_hash(user['hashed_password'], password):
            session['user_id'] = str(user['_id'])
            session['email'] = user['email']
            session['username'] = user['username']
            users_col.update_one({"email": email}, {"$inc": {"login_count": 1}})
            flash('Login successful!', 'success')
            return redirect(url_for('user_dashboard'))

    return render_template('login.html', error="Invalid credentials")

return render_template('login.html')
```

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

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

Dashboard and other routes:



SMARTBRIDGE
Data Design for You

```
app.py > login
106
107     # --- Dashboard ---
108     @app.route('/user_dashboard')
109     @login_required
110     def user_dashboard():
111         email = session['email']
112         bookings = list(bookings_col.find({"email": email}).sort("booked_at", -1))
113         bus = list(db['bus_bookings'].find({"email": email}).sort("booked_at", -1))
114         train = list(train_bookings_col.find({"email": email}))
115         hotels = list(hotel_bookings_col.find({"email": email}).sort("booked_at", -1))
116         trips = list(trip_plans_col.find({"email": email}).sort("created_at", -1))
117
118         return render_template('user_dashboard.html', bookings=bookings, bus_bookings=bus,
119                               train_bookings=train, hotel_bookings=hotels, trip_plans=trips)
120
121
122     # --- Search Places ---
123     @app.route('/search_places', methods=['GET'])
124     def search_places():
125         query = request.args.get('location', '').strip()
126         category = request.args.get('category', '').strip()
127
128         if not query:
129             return render_template("search_results.html", results=[], message="Please enter a location to search.")
130
131         try:
132             # Use Atlas Search if available
133             pipeline = [{"$search": {"index": "default", "text": {"query": query, "path": ["location", "name"]}}}]
134         except:
135             # Fallback to basic regex match if $search fails
136             pipeline = [{"$match": {
137                 "$or": [
138                     {"location": {"$regex": query, "$options": "i"}},
139                     {"name": {"$regex": query, "$options": "i"}}
140                 ]
141             }}]
142
143         if category:
144             pipeline.append({"$match": {"category": {"$regex": category, "$options": "i"}}})
145
146         pipeline.append({"$limit": 20})
147         results = list(db.places.aggregate(pipeline))
148         for r in results:
149             r['_id'] = str(r['_id'])


```

Description: The /dashboard route provides a personalized view where logged-in users can see all their current bookings—flights, buses, trains, and hotels. It fetches data from the database and allows users to manage or cancel their reservations.

The /search (or /booking) route lets users search for travel options by submitting a unified form. Based on the selected tab (flight, bus, train, or hotel), it filters results from available data and shows relevant options for booking.

Request booking Routes:

```
app.py > login
13
14     @app.route('/booking', methods=['GET', 'POST'])
15     def booking():
16         active_tab = 'flights' # default tab
17         flights = buses = trains = hotels = []
18
19         if request.method == 'POST':
20             session['check_in'] = request.form.get('check_in')
21             session['check_out'] = request.form.get('check_out')
22             session['passenger'] = int(request.form.get('passenger', 1))
23             active_tab = request.form.get('active_tab', 'flights')
24             form_type = request.form.get('form_type')
25
26             if form_type == 'flight':
27                 origin = request.form.get('origin')
28                 destination = request.form.get('destination')
29                 date = request.form.get('date').strip()
30                 passengers = int(request.form.get('passenger', 1))
31                 flights = [f for f in flights_data if f['origin'] == origin and f['destination'] == destination and f['departure_time'].startswith(date)]
32                 return render_template('booking.html', flights=flights, active_tab=active_tab, passengers=passenger)
33
34             elif form_type == 'bus':
35                 origin = request.form.get('origin')
36                 destination = request.form.get('destination')
37                 date = request.form.get('date')
38                 buses = [b for b in buses_data if b['origin'] == origin and b['destination'] == destination and b['departure'] == date]
39                 return render_template('booking.html', buses=buses, active_tab=active_tab)
40
41             elif form_type == 'train':
42                 origin = request.form.get('origin')
43                 destination = request.form.get('destination')
44                 date = request.form.get('date')
45                 trains = [t for t in trains_data if t['origin'] == origin and t['destination'] == destination and t['date'] == date]
46                 return render_template('booking.html', trains=trains, active_tab=active_tab)
47
48             elif form_type == 'hotel':
49                 location = request.form.get('location')
50                 check_in = request.form.get('check_in')
51                 check_out = request.form.get('check_out')
52                 hotels = [h for h in hotels_data if h['location'].lower() == location.lower()]
53                 return render_template('booking.html', hotels=hotels, active_tab=active_tab)
54
55
56     @app.route('/confirm_booking/<int:flight_id>', methods=['GET', 'POST'])


```

Description: The /booking route accepts both GET and POST requests. On GET, it shows the search interface with tabs for flights, buses, trains, and hotels. On POST, it processes the form

submission, filters available options based on origin, destination, date, and passengers, and renders the results accordingly.

Train booking route:

```
# app.py > ⚡ login
76  @app.route('/train/confirm/<int:train_id>', methods=['GET', 'POST'])
77  def confirm_train_booking(train_id):
78      train = next((t for t in trains_data if t['id'] == train_id), None)
79      if not train:
80          return "Train not found", 404
81
82      try:
83          passenger_count = int(request.args.get('passengers') or session.get('passengers', 1))
84      except ValueError:
85          passenger_count = 1
86
87      if request.method == 'POST':
88          action = request.form.get('action')
89          selected_seats = request.form.getlist('seats[]')
90
91          if action == 'cancel':
92              flash("Train booking cancelled.", "info")
93              return redirect(url_for('booking'))
94
95          if not selected_seats:
96              flash("Please select at least one seat before confirming.", "warning")
97              return redirect(request.url)
98
99          if len(selected_seats) > passenger_count:
100             flash(f"You selected {len(selected_seats)} seats but only {passenger_count} passenger(s).", "danger")
101             return redirect(request.url)
102
103         for seat in selected_seats:
104             if train['seats'].get(seat) == 'booked':
105                 flash(f"Seat {seat} is already booked. Please choose another.", "danger")
106                 return redirect(request.url)
107
108         passenger_names = request.form.getlist('passenger_names[]')
109         if len(passenger_names) != len(selected_seats):
110             flash("Number of passenger names does not match selected seats.", "danger")
111             return redirect(request.url)
112
113         for seat, name in zip(selected_seats, passenger_names):
114             train['seats'][seat] = 'booked'
115             train_bookings_col.insert_one({
116                 'email': session['email'],
117                 'train_id': train['id'],
118                 'train_number': train['train_number'],
119                 'origin': train['origin'],
120                 'destination': train['destination'],
121             })
122
123     return render_template('confirm_train_booking.html', train=train, selected_seats=selected_seats, passenger_count=passenger_count)
```

Description: The /train/confirm/<int:train_id> route handles train booking. It shows train info and seat selection on GET, and processes booking or cancellation on POST. It validates seat availability, matches passenger details, saves the booking, and redirects to the dashboard.

Bus booking route:

```
# app.py > ⚡ login
20  @app.route('/bus/confirm/<int:bus_id>', methods=['GET', 'POST'])
21  def confirm_bus_booking(bus_id):
22      bus = next((b for b in buses_data if b['id'] == bus_id), None)
23      passenger_names = request.form.getlist('passenger_names[]')
24      if not bus:
25          return "Bus not found", 404
26
27      if request.method == 'POST':
28          action = request.form.get('action')
29          selected_seats = request.form.getlist('seats') # ✅ get list of selected seats
30
31          if action == 'confirm':
32              if not selected_seats:
33                  flash("Please select at least one seat.", 'danger')
34                  return redirect(request.url)
35
36              for seat in selected_seats:
37                  if bus['seats'].get(seat) == 'booked':
38                      flash(f"Seat {seat} already booked.", "danger")
39                      return redirect(request.url)
40
41          # Book seats
42          for i, seat in enumerate(selected_seats):
43              bus['seats'][i] = 'booked'
44              bus_bookings_col.insert_one({
45                  'email': session['email'],
46                  'bus_id': bus['id'],
47                  'bus_number': bus['bus_number'],
48                  'origin': bus['origin'],
49                  'destination': bus['destination'],
50                  'departure_time': bus['departure_time'],
51                  'arrival_time': bus['arrival_time'],
52                  'price': bus['price'],
53                  'seat': seat,
54                  'passenger_name': passenger_names[i] if i < len(passenger_names) else '',
55                  'booked_at': datetime.utcnow()
56              })
57
58          flash(f'Bus booking confirmed! Seats: ({", ".join(selected_seats)}), success')
59          return redirect(url_for('user_dashboard'))
60      else:
61          flash('Bus booking cancelled.', 'info')
62          return redirect(url_for('booking'))
63
64      return render_template('confirm_bus_booking.html', bus=bus, passengers=session.get('passengers', 1))
```



SMARTBRIDGE
Enriching the Way

Description: The /bus/confirm/<int:bus_id> route manages bus bookings. It displays bus details and available seats, handles booking confirmation or cancellation, checks seat availability, and stores confirmed bookings in the database before redirecting to the user dashboard.

Hotel booking routes:

```
❶ app.py > ⌂ login
736  @app.route('/hotel/confirm/<int:hotel_id>', methods=['GET', 'POST'])
737  def confirm_hotel_booking(hotel_id):
738      hotel = next((h for h in hotels_data if h['id'] == hotel_id), None)
739      if not hotel:
740          return "Hotel not found", 404
741
742      # Get check-in and check-out dates from session
743      check_in = session.get('check_in')
744      check_out = session.get('check_out')
745
746      # Calculate total nights
747      try:
748          nights = (datetime.strptime(check_out, '%Y-%m-%d') - datetime.strptime(check_in, '%Y-%m-%d')).days
749          if nights <= 0:
750              nights = 1
751      except (TypeError, ValueError):
752          nights = 1 # fallback if dates are missing or malformed
753
754      if request.method == 'POST':
755          if request.form['action'] == 'confirm':
756              hotel_bookings_col.insert_one({
757                  'email': session['email'],
758                  'hotel_id': hotel['id'],
759                  'hotel_name': hotel['name'],
760                  'location': hotel['location'],
761                  'price': hotel['price_per_night'],
762                  'rating': hotel['rating'],
763                  'check_in': check_in,
764                  'check_out': check_out,
765                  'booked_at': datetime.utcnow()
766              })
767              flash('Hotel booking confirmed!', 'success')
768              return redirect(url_for('user_dashboard'))
769          else:
770              flash('Hotel booking cancelled.', 'info')
771              return redirect(url_for('booking'))
772
773      return render_template(
774          'confirm_hotel_booking.html',
775          hotel=hotel,
776          check_in=check_in,
777          check_out=check_out,
778          nights=nights
779      )
```

Description: The /hotel/confirm/<int:hotel_id> route handles hotel booking confirmation. It retrieves hotel info, calculates nights based on check-in/check-out dates, and on confirmation, saves the booking in the database, then redirects to the user dashboard.

Exit Route:

```
❶ # ----- LOGOUT -----
45
46  @app.route('/logout')
47  def logout():
48      session.pop('email', None)
49      return redirect(url_for('login'))
50
```

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



Deployment Code:

```
850
851  # ----- RUN -----
852  if __name__ == '__main__':
853  |   app.run(host='0.0.0.0', port=80, debug=True)
854
```

Description: start the Flask server to listen on all network interfaces (0.0.0.0) at port 80 with debug mode enabled for development and testing.

Milestone 2 : AWS Account Setup

Important Notice: Use Troven Labs for AWS Access

Students are strictly advised not to create their own AWS accounts, as doing so may incur charges. Instead, we have set up a dedicated section called “Labs” on the Troven platform, which provides temporary and cost-free access to AWS services.

Once your website is locally deployed and fully functional, you must proceed with integrating AWS services only through the Troven Labs environment. This ensures secure, controlled access to AWS resources without any risk of personal billing.

All steps involving AWS (such as deploying to EC2, connecting to DynamoDB, or using SNS) must be carried out within the Troven Labs platform, as we've configured temporary credentials for each student.

AWS Account Setup and Login

- Go to the AWS website (<https://aws.amazon.com/>).
- Click on the "Create an AWS Account" button.
- Follow the prompts to enter your email address and choose a password.
- Provide the required account information, including your name, address, and phone number.
- Enter your payment information. (Note: While AWS offers a free tier, a credit card or debit card is required for verification.)
- Complete the identity verification process.
- Choose a support plan (the basic plan is free and sufficient for starting).
- Once verified, you can sign in to your new AWS accounts.



SMARTBRIDGE
Let's Bridge the Gap

aws.amazon.com/?nc2=hJg

About AWS Contact Us Support English My Account Sign In Complete Sign Up

Amazon Q Products Solutions Pricing Documentation Learn Partner Network AWS Marketplace Customer Enablement Events Explore More

Complete your AWS registration

Millions of customers are using AWS cloud solutions to build applications with increased flexibility, scalability, security, and reliability

[Complete sign-up](#)

AWS Free Tier
Use Amazon EC2, S3, and more—free for a full year

Customer success stories
Learn how companies are using AWS to solve their biggest challenges

Contact us
Reach out to us for any AWS related questions

aws

Explore Free Tier products with a new AWS account.
To learn more, visit aws.amazon.com/free.

Sign up for AWS

Root user email address
Used for account recovery and as described in the [AWS Privacy Notice](#)

AWS account name
Choose a name for your account. You can change this name in your account settings after you sign up.

Verify email address

OR

Sign in to an existing AWS account

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

aws

Sign in

Root user
Account owner that performs tasks requiring unrestricted access. [Learn more](#)

IAM user
User within an account that performs daily tasks. [Learn more](#)

Root user email address

[Next](#)

By continuing, you agree to the AWS Customer Agreement or other agreement for AWS services, and the Privacy Notice. This site uses essential cookies. See our [Cookie Notice](#) for more information.

AI Use Case Explorer

Discover AI use cases, customer success stories, and expert-curated implementation plans

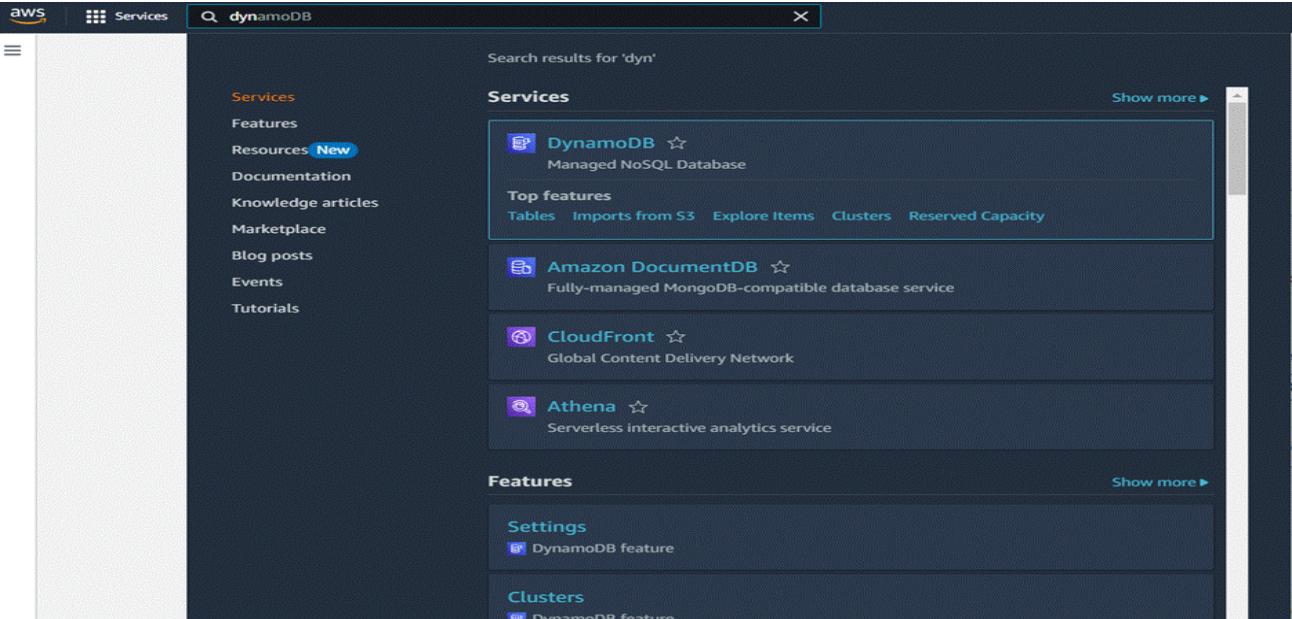
[Explore now >](#)

Milestone 3 : DynamoDB Database Creation and Setup

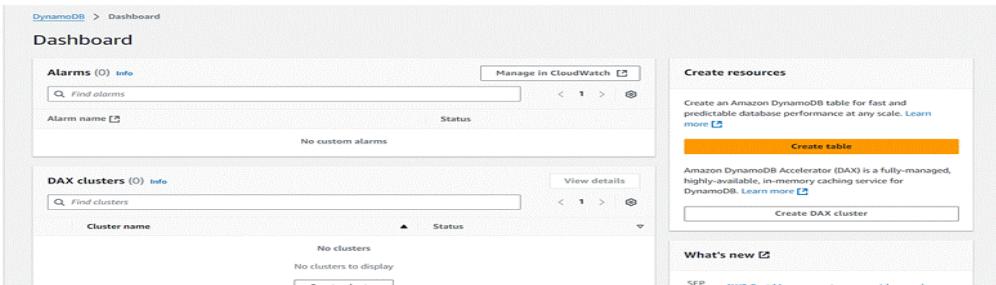
Database Creation and Setup involves initializing a cloud-based NoSQL database to store and manage application data efficiently. This step includes defining tables, setting primary keys, and configuring read/write capacities. It ensures scalable, high-performance data storage for seamless backend operations.

Navigate to the DynamoDB

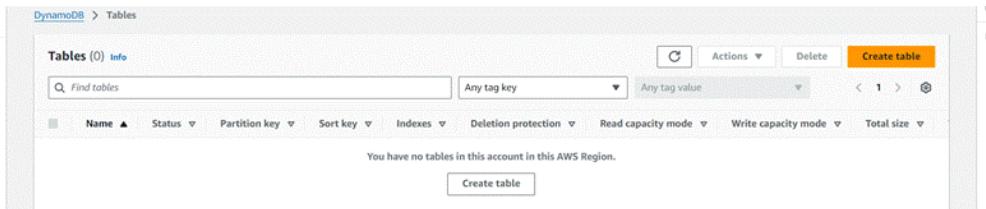
- In the AWS Console, navigate to DynamoDB and click on create tables.



The screenshot shows the AWS search interface with the query 'dynamodb' entered in the search bar. The top result is 'DynamoDB' under the 'Services' category, described as a 'Managed NoSQL Database'. Other services listed include Amazon DocumentDB, CloudFront, and Athena. Below the services, there are sections for 'Features' (Settings, DAX clusters) and 'Clusters'.



The screenshot shows the DynamoDB Dashboard. On the left, there's a sidebar with links for Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. The main area is titled 'Dashboard' and shows sections for 'Alarms' (0), 'DAX clusters' (0), and 'Create resources'. A 'Create table' button is prominently displayed.



The screenshot shows the 'Tables' page within the DynamoDB service. The sidebar on the left is identical to the dashboard. The main content area shows a table header with columns: Name, Status, Partition key, Sort key, Indexes, Deletion protection, Read capacity mode, Write capacity mode, and Total size. A message at the bottom states, 'You have no tables in this account in this AWS Region.' A 'Create table' button is located at the bottom of the table list.

Create an DynamoDB table for storing data

- Create a travel-Users table for storing registered user details with partition key “Email” with type String and click on create tables.

☰ [DynamoDB](#) > [Tables](#) > [Create table](#)



Create table

Table details Info

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

Table name

This will be used to identify your table.

travel-Users

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.)

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

email

String ▾

1 to 255 characters and case sensitive.

Sort key - optional

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

Enter the sort key name

String ▾

1 to 255 characters and case sensitive.

- Follow the same steps to create a Bookings for storing booking records with email as the partition key and booking_id as the sort key.

Create table

Table details Info

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

Table name

This will be used to identify your table.

Bookings

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.)

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

email

String ▾

1 to 255 characters and case sensitive.

Sort key - optional

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

booking_id

String ▾

1 to 255 characters and case sensitive.

Table settings

Default settings

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

Customize settings

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



SMARTBRIDGE
Data Science for All

The screenshot shows the AWS DynamoDB console with the URL <https://us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#tables>. The left sidebar has 'DynamoDB' selected under 'Tables'. The main area displays a table titled 'Tables (9) Info' with columns: Name, Status, Partition key, Sort key, Indexes, Replication Regions, Deletion protection, Favorite, and Read capacity. The tables listed are: bookings, bus_bookings, flights_data, hotels_bookings, places, train_bookings, trip_plans, users, and wishlists. Each table row includes a checkbox, a status icon (green for active), and a status name (e.g., Active, email (S)). The 'Deletion protection' column shows 'Off' for all tables.

Milestone 4 : SNS Notification Setup

Amazon SNS is a fully managed messaging service that enables real-time notifications through channels like SMS, email, or app endpoints. You create topics, configure subscriptions, and integrate SNS into your app to send notifications based on specific events.

SNS Topics for email notifications

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

The screenshot shows the AWS search results page with the search term 'sns' entered in the search bar. The results are filtered by 'Services'. The top result is 'Simple Notification Service' with the subtext 'SNS managed message topics for Pub/Sub'. Below it is 'Route 53 Resolver' with the subtext 'Resolve DNS queries in your Amazon VPC and on-premises network.' Other visible links include 'Features', 'Resources New', 'Documentation', 'Knowledge articles', and 'Marketplace'.



SMARTBRIDGE
Data Bridge for the Web

Amazon SNS

New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics. [Learn more](#)

Dashboard Topics Subscriptions

▼ Mobile Push notifications Text messaging (SMS)

Application Integration

Amazon Simple Notification Service

Pub/sub messaging for microservices and serverless applications.

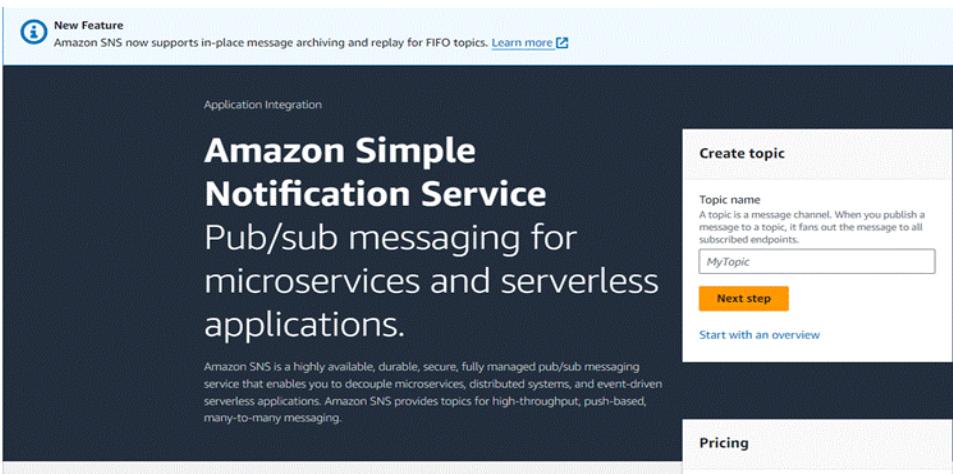
Amazon SNS is a highly available, durable, secure, fully managed pub/sub messaging service that enables you to decouple microservices, distributed systems, and event-driven serverless applications. Amazon SNS provides topics for high-throughput, push-based, many-to-many messaging.

Create topic

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

Next step Start with an overview

Pricing



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

Amazon SNS

New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics. [Learn more](#)

Dashboard Topics Subscriptions

▼ Mobile Push notifications Text messaging (SMS)

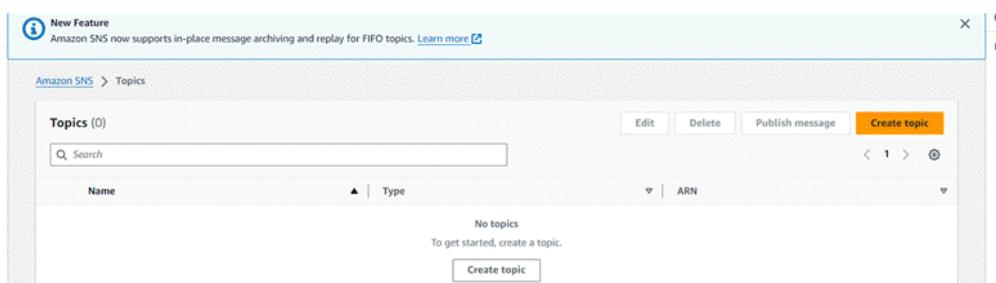
Topics (0)

Search

Name Type ARN

No topics To get started, create a topic.

Create topic



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

Create topic

Details

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

FIFO (first-in, first-out)
• Strictly-preserved message ordering
• Exactly-once message delivery
• Subscription protocols: SQS

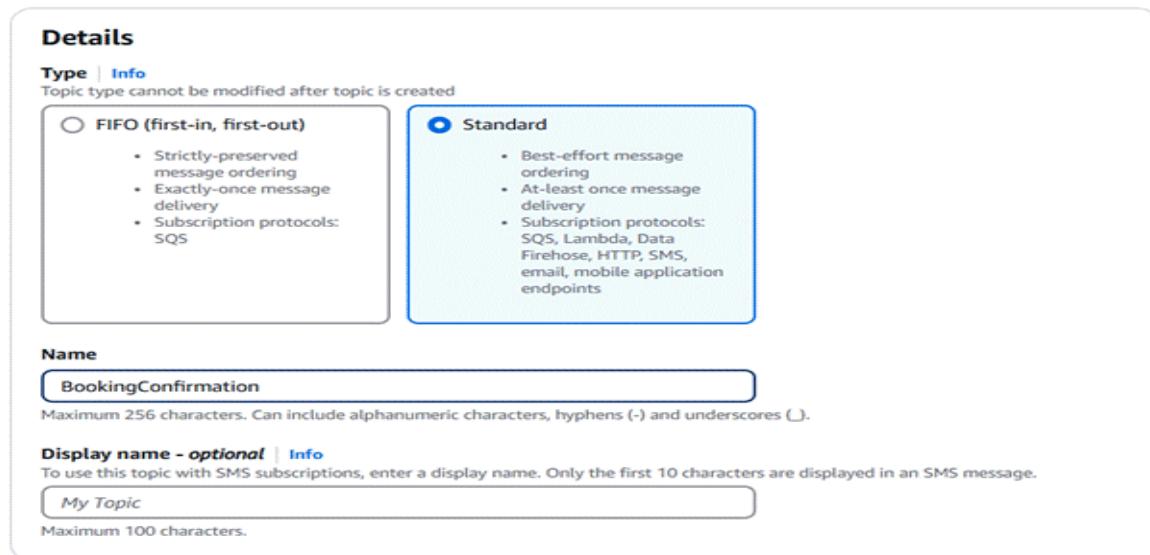
Standard
• Best-effort message ordering
• At-least once message delivery
• Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints

Name

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

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

Maximum 100 characters.



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

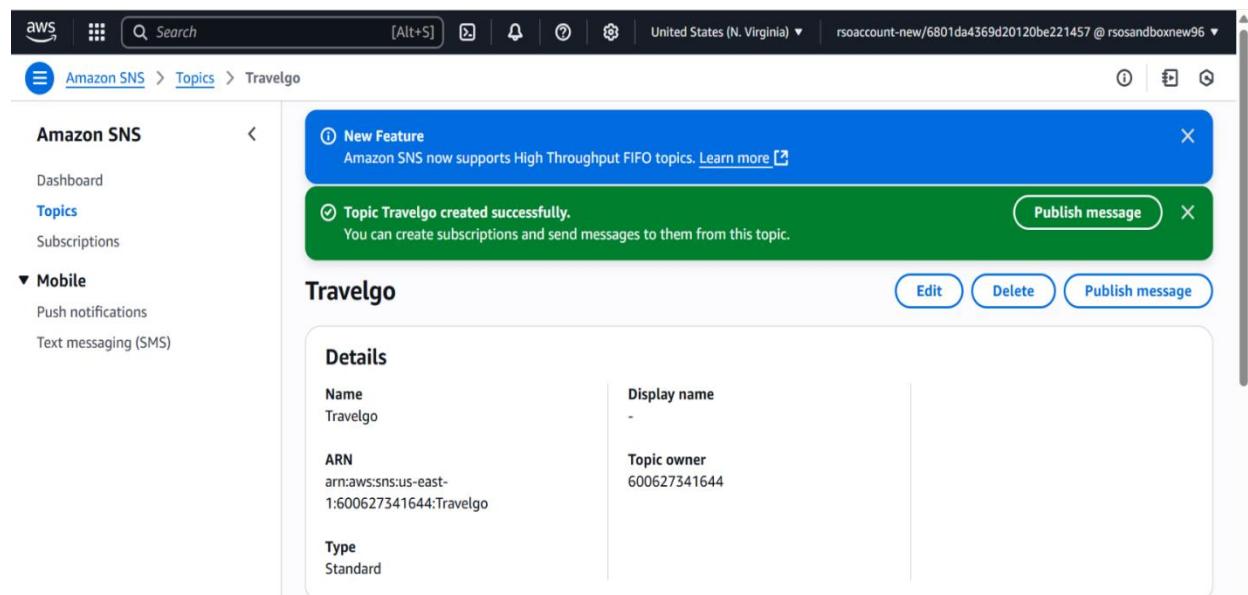
A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. [Learn more](#)

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.

[Cancel](#) [Create topic](#)

? Configure the SNS topic and note down the Topic ARN.



The screenshot shows the AWS SNS Topics page. On the left, there's a sidebar with options like Dashboard, Topics, Subscriptions, and Mobile (Push notifications, Text messaging (SMS)). The main area shows a list of topics under 'Topics'. A new topic, 'Travelgo', has just been created, as indicated by a green notification bar at the top right: 'Topic Travelgo created successfully. You can create subscriptions and send messages to them from this topic.' Below this, there's a 'Publish message' button. The 'Travelgo' topic card shows its details: Name: Travelgo, ARN: arn:aws:sns:us-east-1:600627341644:Travelgo, and Type: Standard. There are also 'Edit' and 'Delete' buttons.

Subscribe users and admin

- Subscribe users (or admin staff) to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails.
- After subscription request for the mail confirmation



SMARTBRIDGE
Data Bridge One Step

AWS Notifications<no-reply@sns.amazonaws.com>

To: You

Mon 6/30/2025 9:44 AM

You have chosen to subscribe to the topic:

arn:aws:sns:us-east-1:586794467071:Travelgo

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):

[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)

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



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

The screenshot shows the AWS SNS Topics page. On the left, there's a sidebar with 'Amazon SNS' navigation: Dashboard, Topics (which is selected), Subscriptions, and Mobile (Push notifications, Text messaging (SMS)). The main area shows a topic named 'Travelgo'. A blue banner at the top says 'New Feature: Amazon SNS now supports High Throughput FIFO topics. Learn more'. Below that, a green banner says 'Topic Travelgo created successfully. You can create subscriptions and send messages to them from this topic.' with a 'Publish message' button. The 'Travelgo' topic card has 'Edit', 'Delete', and 'Publish message' buttons. The 'Details' section shows the following information:

- Name: Travelgo
- ARN: arn:aws:sns:us-east-1:600627341644:Travelgo
- Type: Standard
- Display name: -
- Topic owner: 600627341644



SMARTBRIDGE
Data Science Hub

The screenshot shows the AWS SNS console with a successful subscription creation message: "Subscription to Travelgo created successfully. The ARN of the subscription is arn:aws:sns:us-east-1:586794467071:Travelgo:90e78453-f4b5-485a-85bd-dc379e38a683." Below this, the "Subscription: 90e78453-f4b5-485a-85bd-dc379e38a683" details are displayed, including ARN, Endpoint, Topic, and Subscription Principal. The status is "Pending confirmation" and the protocol is "EMAIL". At the bottom, there are tabs for "Subscription filter policy" and "Redrive policy (dead-letter queue)".

Milestone 5 : IAM Role Setup

IAM (Identity and Access Management) role setup involves creating roles that define specific permissions for AWS services. To set it up, you create a role with the required policies, assign it to users or services, and ensure the role has appropriate access to resources like EC2, S3, or RDS. This allows controlled access and ensures security best practices in managing AWS resources.

Create IAM Role.

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

The screenshot shows the AWS search results for 'iam'. The search bar at the top contains 'Q. iam'. The left sidebar lists services such as Features, Resources (New), Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials. The main search results section displays four services: IAM (Manage access to AWS resources), IAM Identity Center (Manage workforce user access to multiple AWS accounts and cloud applications), Resource Access Manager (Share AWS resources with other accounts or AWS Organizations), and AWS App Mesh (Easily monitor and control microservices). A 'Show more' link is visible on the right side of the search results.



SMARTBRIDGE
Cloud Bridge for AWS

The screenshot shows the AWS Identity and Access Management (IAM) service. On the left, there's a sidebar with a search bar and a 'Dashboard' link. The main area is titled 'Roles (6)' and contains a table with the following columns: Role name, Trusted entities, and Last activity. There are six rows in the table, each representing a different role. At the top right of the table, there are buttons for 'Create role', 'Delete', and other actions.

This screenshot shows the 'Select trusted entity' step of the 'Create role' wizard. It has three tabs: Step 1 (Select trusted entity), Step 2 (Add permissions), and Step 3 (Name, review, and create). The Step 1 tab is active. It includes sections for 'Trusted entity type' (with options for AWS service, IAM account, or Web identity), 'Service or use case' (set to EC2), and 'Use case' (with several EC2-related options like 'EC2 instances to call AWS services on your behalf'). At the bottom are 'Cancel' and 'Next' buttons.

This screenshot shows the 'Add permissions' step of the 'Create role' wizard. It has three tabs: Step 1 (Select trusted entity), Step 2 (Add permissions), and Step 3 (Name, review, and create). The Step 2 tab is active. It shows a list of 'Permissions policies (1/955)' with a search bar and a filter for 'All types'. Two policies are listed: 'AmazonDynamoDBFullAccess' and 'AmazonDynamoDBReadOnlyAccess'. Below the list is a note about setting a 'permissions boundary'. At the bottom are 'Cancel', 'Previous', and 'Next' buttons.

Attach Policies.

Attach the following policies to the role:

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

Step 1
Select trusted entity

Add permissions Info

Permissions policies (2/955) Info

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

Filter by Type

| Name | Type |
|--|-------------|
| <input checked="" type="checkbox"/> AmazonSNSFullAccess | AWS managed |
| <input type="checkbox"/> AmazonSNSReadOnlyAccess | AWS managed |
| <input type="checkbox"/> AmazonSNSRole | AWS managed |
| <input type="checkbox"/> AWSLambdaBasicExecutionRole | AWS managed |
| <input type="checkbox"/> AWSIoTDevice Defender PublishFindingsToSNSIntegrationAction | AWS managed |

Set permissions boundary - optional

[Cancel](#) [Previous](#) [Next](#)

Microsoft Edge | Tanuja-04311 | Khit_3rd_Year | app.py for final | Student - Skill | troven.in/studio | troven.in/studio | Roles | IAM | Home - Travel | +

aws https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles

Search [Alt+S] Global rsoaccount-new/5801da4569d20120be221457 @ rsandboxnew29

IAM > Roles

Identity and Access Management (IAM)

Search IAM

Roles (14) Info

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

| Role name | Trusted entities | Last activity |
|-------------------------------|-----------------------|----------------|
| DCEPrincipal-bunny/tf | Account: 058264256896 | - |
| OrganizationAccountAccessRole | Account: 058264256896 | 1 hour ago |
| rsoaccount-new | Account: 058264256896 | 13 minutes ago |
| Studentuser | AWS Service: ec2 | 18 minutes ago |

Roles Anywhere Info

Authenticate your non AWS workloads and securely provide access to AWS services.

Access AWS from your non AWS workloads

Operate your non AWS workloads using the same authentication and authorization strategy that you use within AWS.

X.509 Standard

Use your own existing PKI infrastructure or use AWS Certificate Manager Private Certificate Authority to authenticate identities.

Temporary credentials

Use temporary credentials with ease and benefit from the enhanced security they provide.

CloudShell Feedback

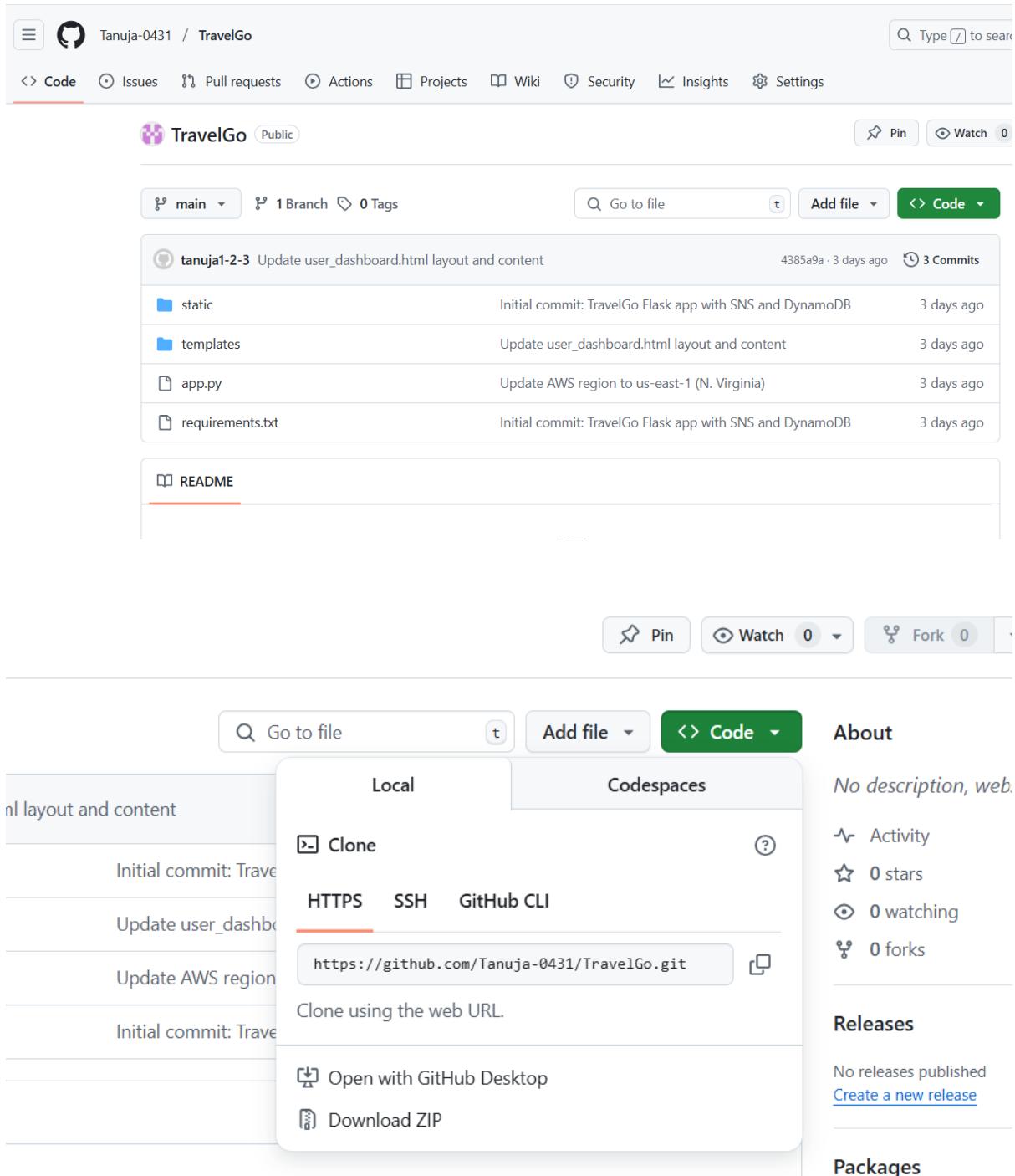
Quick search

28°C Mostly cloudy ENG JN 30-06-2025

Milestone 6 : EC2 Instance Setup

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

To set up a public EC2 instance, choose an appropriate Amazon Machine Image (AMI) and instance type. Ensure the security group allows inbound traffic on necessary ports (e.g., HTTP/HTTPS for web applications). After launching the instance, associate it with an elastic IP for consistent public access, and configure your application or services to be publicly accessible.



The screenshot shows a GitHub repository page for 'TravelGo'. At the top, there's a navigation bar with links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below the navigation is the repository name 'TravelGo' and its status as 'Public'. A search bar at the top right says 'Type / to search'. Underneath, there's a commit history for a single branch named 'main'. The commits are as follows:

- tanuja1-2-3 Update user_dashboard.html layout and content (4385a9a - 3 days ago)
- static Initial commit: TravelGo Flask app with SNS and DynamoDB (3 days ago)
- templates Update user_dashboard.html layout and content (3 days ago)
- app.py Update AWS region to us-east-1 (N. Virginia) (3 days ago)
- requirements.txt Initial commit: TravelGo Flask app with SNS and DynamoDB (3 days ago)

Below the commit history is a 'README' file. On the right side of the page, there are buttons for Pin, Watch (0), Fork (0), and a dropdown menu. A modal window is open in the center, titled 'Local' and 'Codespaces'. It contains a 'Clone' section with three options: HTTPS (selected), SSH, and GitHub CLI. The HTTPS URL is shown as <https://github.com/Tanuja-0431/TravelGo.git>. Below the URL are buttons for 'Open with GitHub Desktop' and 'Download ZIP'. To the right of the modal, there are sections for 'About', 'Activity', 'Stars', 'Watching', 'Forks', 'Releases', and 'Packages'.

Launch An EC2 instance to host The flask

Load your Project Files to Github

- Launch EC2 Instance



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

The screenshot shows the AWS Services search results for 'ec2'. The EC2 service card is highlighted, showing its name, icon, and description: 'Virtual Servers in the Cloud'.

- Click on Launch instance to launch EC2 instance

The screenshot shows the EC2 Dashboard Instances page. It displays a message 'No instances' and a 'Launch instances' button.

- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).
- Create and download the key pair for Server access.

The screenshot shows the 'Launch instance' wizard. Step 1: Instance type. It shows the t2.micro instance type selected, which is free-tier eligible. Step 2: Key pair (login). It shows a dropdown for 'Key pair name - required' with 'Select' and a 'Create new key pair' button.



SMARTBRIDGE
Data Bridge for AWS

Earlier this week



travel-go-application.pem

Description

Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.

Architecture: 64-bit (x86) Boot mode: uefi-preferred AMI ID: ami-078264b8ba71bc45e Username: ec2-user Verified provider

Instance type

t2.micro Family: t2 1 vCPU 1 GiB Memory Current generation: true Free tier eligible On-Demand Linux base pricing: 0.0124 USD per Hour On-Demand Windows base pricing: 0.0117 USD per Hour On-Demand RHEL base pricing: 0.02611 USD per Hour On-Demand SUSE base pricing: 0.0124 USD per Hour Additional costs apply for AMIs with pre-installed software

All generations Compare instance types

Key pair (login)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required InstantLibrary Create new key pair

Summary

Number of instances: 1

Software Image (AMI): Amazon Linux 2023 AMI 2023.5.2... read more ami-078264b8ba71bc45e

Virtual server type (instance type): t2.micro

Firewall (security group): New security group

Storage (volumes): 1 volume(s) - 8 GiB

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 750 hours of public IPv4 address usage per month, 30 GiB of EBS storage, 2 million IOPS, 1 GiB of snapshots, and 100 GiB of bandwidth to the internet.

Cancel Preview code Launch instance

Success Successfully initiated launch of instance i-048d39bf1e76c4077

Launch log

Next Steps

What would you like to do next with this instance, for example "create alarm" or "create backup"

Create billing and free tier usage alerts

To manage costs and avoid surprise bills, set up email notifications for billing and free tier usage thresholds.

Create billing alerts

Connect to your instance

Once your instance is running, log into it from your local computer.

Connect to instance

Learn more

Connect an RDS database

Configure the connection between an EC2 instance and a database to allow traffic flow between them.

Connect an RDS database

Create a new RDS database

Learn more

Create EBS snapshot policy

Create a policy that automates the creation, retention, and deletion of EBS snapshots

Create EBS snapshot policy

Manage detailed monitoring

Create Load Balancer

Create AWS budget

Manage CloudWatch alarms

CloudShell Feedback

Quick search

ENG IN

15:17 30-06-2025



SMARTBRIDGE
Engineering the Future

Configure security groups for HTTP, and SSH access

To ensure the server remains both accessible and manageable, the security group must allow specific types of network traffic. HTTP access (port 80) enables users to open the website in their browser, while SSH access (port 22) allows secure remote login for deployment and server maintenance. These rules, configured within AWS security groups, help maintain a balance between usability and system security.

Inbound rules info

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

| Security group rule ID | Type | Protocol | Port range | Source | Description - optional |
|------------------------|------------|----------|------------|--------|------------------------|
| sgr-0db5b532b0c13ae52 | Custom TCP | TCP | 5000 | Custom | 0.0.0.0/0 |
| sgr-05a381ef15f419538 | SSH | TCP | 22 | Custom | 0.0.0.0/0 |
| sgr-0a2c7518bca3fac13 | HTTPS | TCP | 443 | Custom | 0.0.0.0/0 |

Add rule

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

Cancel Preview changes Save rules

EC2 > Security Groups > sg-0e5e82a22bc61d34b - launch-wizard-1

Inbound security group rules successfully modified on security group (sg-0e5e82a22bc61d34b | launch-wizard-1)

Details

| Security group name | Security group ID | Description | VPC ID |
|---------------------|----------------------|---|-----------------------|
| launch-wizard-1 | sg-0e5e82a22bc61d34b | launch-wizard-1 created 2025-06-30T09:46:14.58Z | vpc-0808cf43f556c9f0f |
| Owner | 586794467071 | Inbound rules count | Outbound rules count |
| | | 3 Permission entries | 1 Permission entry |

Inbound rules (3)

| Name | Security group rule ID | IP version | Type | Protocol | Port range |
|------|------------------------|------------|------------|----------|------------|
| - | sgr-0db5b532b0c13ae52 | IPv4 | Custom TCP | TCP | 5000 |
| - | sgr-05a381ef15f419538 | IPv4 | SSH | TCP | 22 |
| - | sgr-0a2c7518bca3fac13 | IPv4 | HTTPS | TCP | 443 |

- To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then



navigating to Security and selecting Modify IAM Role to attach the appropriate role. After the IAM role is connected, navigate to the EC2 section in the AWS Management Console. Select the EC2 instance you wish to connect to. At the top of the EC2 Dashboard, click the Connect button. From the connection methods presented, choose EC2 Instance Connect. Finally, click Connect again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.

The screenshot shows the 'Modify IAM role' page in the AWS Management Console. The 'Instance ID' dropdown is set to 'i-048d39bf1e76c4077 (TravelGoProject)'. The 'IAM role' dropdown is set to 'Studentuser'. There are 'Cancel' and 'Update IAM role' buttons at the bottom.

The screenshot shows the 'Instances' page in the AWS Management Console. A green success message at the top states 'Successfully attached Studentuser to instance i-048d39bf1e76c4077'. The main table shows one instance: 'TravelGoProject' (i-048d39bf1e76c4077), which is 'Running'. The 'Actions' dropdown menu for this instance includes 'Connect'. The 'Details' tab is selected for the instance, showing its summary information including IP addresses and DNS names.

- Now connect the EC2 with the files



Connect to instance

Connect to your instance i-0c95169c57a272969 (travel-booking) using any of these options

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

Instance ID
 i-0c95169e57a272969 (travel-booking)

Connection Type

Connect using EC2 Instance Connect
 Connect using the EC2 Instance Connect browser-based client, with a public IPv4 or IPv6 address.

Connect using EC2 Instance Connect Endpoint
 Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

Public IPv4 address
 3.110.28.122

IPv6 address

Username
 Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ubuntu.

Note: In most cases, the default username, ubuntu, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

Milestone 7: Deployment using EC2

Deployment on an EC2 instance involves launching a server, configuring security groups for public access, and uploading your application files. After setting up necessary dependencies and environment variables, start your application and ensure it's running on the correct port. Finally, bind your domain or use the public IP to make the application accessible online.

Install Software on the EC2 Instance

Install Python3, Flask, and Git:

- **On Amazon Linux 2:**

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

- **Verify Installations:**

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

Clone Your Flask Project from GitHub

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

- Run: '[git clone https://github.com/Tanuja-0431/TravelGo.git](https://github.com/Tanuja-0431/TravelGo.git)'
- **This will download your project to the EC2 instance.**
- To navigate to the project directory, run the following command:
cd <your-folder> i.e.; Travelgo
- Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:
- Run the Flask Application
- **export SNS_TOPIC_ARN= arn:aws:sns:us-east-1:586794467071:Travelgo:90e78453-f4b5-485a-85bd-dc379e38a683**
- git pull origin main (Only if you made changes in git and want to reflect them in EC2 terminal.)

```
# Install requirements
pip install flask boto3
• Launch the Flask app:
sudo -E venv/bin/python3 app.py
```

Verify the Flask app is running:

<http://your-ec2-public-ip> i.e.; <http://3.84.5.171:5000/>

- Run the Flask app on the EC2 instance



SMARTBRIDGE
Data Bridge for the Web

```
[ec2-user@ip-172-31-81-119: ~] + ~
Error updating places: An error occurred (ResourceNotFoundException) when calling the Scan operation: Requested resource not found
Error updating flights: An error occurred (ResourceNotFoundException) when calling the Scan operation: Requested resource not found
* Serving Flask app 'app'
* Debug mode: on
Permission denied
[ec2-user@ip-172-31-81-119 TravelGo]$ cd TravelGo
-bash: cd: TravelGo: No such file or directory
[ec2-user@ip-172-31-81-119 TravelGo]$ cd ~/TravelGo
[ec2-user@ip-172-31-81-119 TravelGo]$ git pull
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 441 bytes | 441.00 KiB/s, done.
From https://github.com/faujan-a-0431/TravelGo
   e199802...a3f6971 main       -> origin/main
Updating e199802...a3f6971
Fast-forward
 app.py | 6 +----
 1 files changed, 3 insertions(+), 3 deletions(-)
[ec2-user@ip-172-31-81-119 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://172.31.81.119:5000
Press CTRL+C to quit.
* Restarting with stat
Default fields added where missing.
Seat layout added to all flights.
* Debugger is active!
* Debugger PIN: 8465-0804
115.244.132.21 -- [30/Jun/2025 10:52:53] "GET / HTTP/1.1" 200 -
115.244.132.21 -- [30/Jun/2025 10:52:53] "GET /static/css/style.css HTTP/1.1" 200 -
115.244.132.21 -- [30/Jun/2025 10:52:54] "GET /favicon.ico HTTP/1.1" 404 -
103.168.81.62 -- [30/Jun/2025 10:56:55] "GET / HTTP/1.1" 200 -
103.168.81.62 -- [30/Jun/2025 10:56:55] "GET /static/css/style.css HTTP/1.1" 200 -
103.168.81.62 -- [30/Jun/2025 10:56:56] "GET /favicon.ico HTTP/1.1" 404 -
103.168.81.62 -- [30/Jun/2025 10:56:57] "GET /register HTTP/1.1" 200 -
103.168.81.62 -- [30/Jun/2025 10:57:07] "GET /register HTTP/1.1" 200 -
103.168.199.199 -- [30/Jun/2025 10:57:07] "GET /register HTTP/1.1" 200 -
103.168.199.199 -- [30/Jun/2025 10:57:07] "GET /static/css/style.css HTTP/1.1" 200 -
103.168.199.199 -- [30/Jun/2025 10:57:07] "GET /favicon.ico HTTP/1.1" 404 -
223.228.99.148 -- [30/Jun/2025 10:57:09] "GET / HTTP/1.1" 200 -
223.228.99.148 -- [30/Jun/2025 10:57:09] "GET /static/css/style.css HTTP/1.1" 200 -
152.59.199.199 -- [30/Jun/2025 10:57:10] "GET /register HTTP/1.1" 200 -
223.228.99.148 -- [30/Jun/2025 10:57:10] "GET /register HTTP/1.1" 200 -
152.59.199.199 -- [30/Jun/2025 10:57:10] "GET /register HTTP/1.1" 200 -
223.228.99.148 -- [30/Jun/2025 10:57:12] "GET /favicon.ico HTTP/1.1" 404 -
223.228.99.148 -- [30/Jun/2025 10:57:12] "GET /favicon.ico HTTP/1.1" 404 -
```

28°C Mostly cloudy Quick search 16:32 ENG IN 30-06-2025

```
return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args) # type: ignore[no-any-return]
File "/home/ec2-user/TravelGo/app.py", line 92, in wrap
    return f(*args, **kwargs)
File "/home/ec2-user/TravelGo/app.py", line 171, in user_dashboard
    return render_template('user_dashboard.html',
File "/home/ec2-user/.local/lib/python3.9/site-packages/flask/templating.py", line 149, in render_template
    template = app.jinja_env.get_or_select_template(template_name_or_list)
File "/home/ec2-user/.local/lib/python3.9/site-packages/jinja2/environment.py", line 1087, in get_or_select_template
    return self.get_template(template_name_or_list, parent, globals)
File "/home/ec2-user/.local/lib/python3.9/site-packages/jinja2/environment.py", line 1016, in get_template
    return self._load_template(name, globals)
File "/home/ec2-user/.local/lib/python3.9/site-packages/jinja2/environment.py", line 975, in _load_template
    template = self.loader.load(name, self.make_globals(globals))
File "/home/ec2-user/.local/lib/python3.9/site-packages/jinja2/loaders.py", line 138, in load
    code = environment.compile(source, name, filename)
File "/home/ec2-user/.local/lib/python3.9/site-packages/jinja2/environment.py", line 771, in compile
    self.handle_exception(source=source_hint)
File "/home/ec2-user/.local/lib/python3.9/site-packages/jinja2/environment.py", line 942, in handle_exception
    raise rewrite_traceback_stack(sources=source)
File "/home/ec2-user/TravelGo/templates/user_dashboard.html", line 149, in template
    * train.depends_on_time(datetime.datetime.now() - {train.arrival_time|datetimeformat })
jinja2.exceptions.TemplateAssertionError: No filter named 'datetimeformat'
123.228.99.148 -- [30/Jun/2025 10:57:40] "GET /user/dashboard?_debugger_=yes&cmd=resource&f=style.css HTTP/1.1" 200 -
123.228.99.148 -- [30/Jun/2025 10:57:49] "GET /user/dashboard?_debugger_=yes&cmd=resource&f=debugger.js HTTP/1.1" 200 -
123.228.99.148 -- [30/Jun/2025 10:57:50] "GET /user/dashboard?_debugger_=yes&cmd=resource&f=console.png HTTP/1.1" 200 -
123.228.99.148 -- [30/Jun/2025 10:57:50] "GET /user/dashboard?_debugger_=yes&cmd=resource&f=console.png HTTP/1.1" 304 -
103.244.132.21 -- [30/Jun/2025 10:58:01] "GET /static/css/style.css HTTP/1.1" 200 -
103.244.132.21 -- [30/Jun/2025 10:58:02] "GET /static/css/style.css HTTP/1.1" 200 -
103.244.132.21 -- [30/Jun/2025 10:58:03] "GET /static/css/style.css HTTP/1.1" 200 -
103.244.132.21 -- [30/Jun/2025 10:58:04] "GET /static/css/style.css HTTP/1.1" 200 -
103.244.132.21 -- [30/Jun/2025 10:58:05] "GET /register HTTP/1.1" 200 -
103.244.132.21 -- [30/Jun/2025 10:58:06] "GET /register HTTP/1.1" 200 -
103.244.132.21 -- [30/Jun/2025 10:58:07] "GET /register HTTP/1.1" 200 -
103.244.132.21 -- [30/Jun/2025 10:58:08] "GET /register HTTP/1.1" 200 -
103.244.132.21 -- [30/Jun/2025 10:58:09] "GET /register HTTP/1.1" 200 -
103.244.132.21 -- [30/Jun/2025 10:58:10] "GET /register HTTP/1.1" 200 -
103.244.132.21 -- [30/Jun/2025 10:58:11] "GET /register HTTP/1.1" 200 -
10.77.167.283 -- [30/Jun/2025 11:01:00] "GET //users.txt HTTP/1.1" 404 -
12.167.144.217 -- [30/Jun/2025 11:01:00] "GET / HTTP/1.1" 200 -
10.77.190.141 -- [30/Jun/2025 11:01:18] "GET /static/css/style.css HTTP/1.1" 200 -
52.167.144.176 -- [30/Jun/2025 11:01:37] "GET /login HTTP/1.1" 200 -
```

28°C Mostly cloudy Quick search 16:36 ENG IN 30-06-2025

- Access the website through:

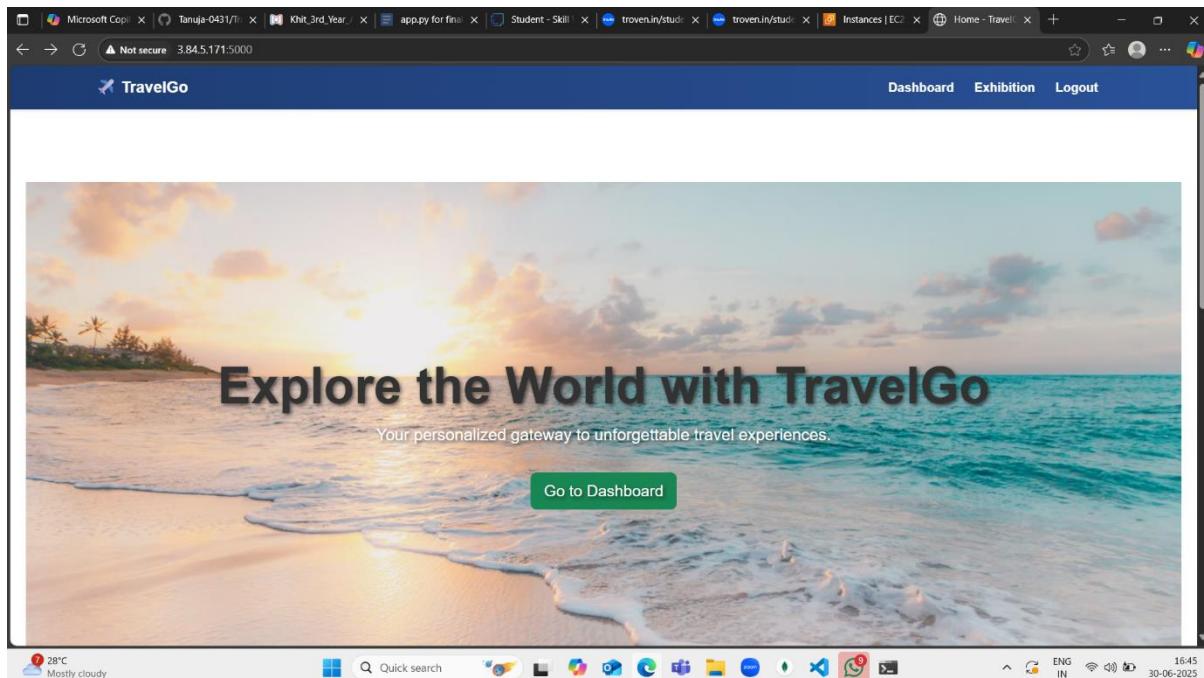
Public IPs: <http://3.84.5.171:5000/>

Milestone 8 : Testing and Deployment

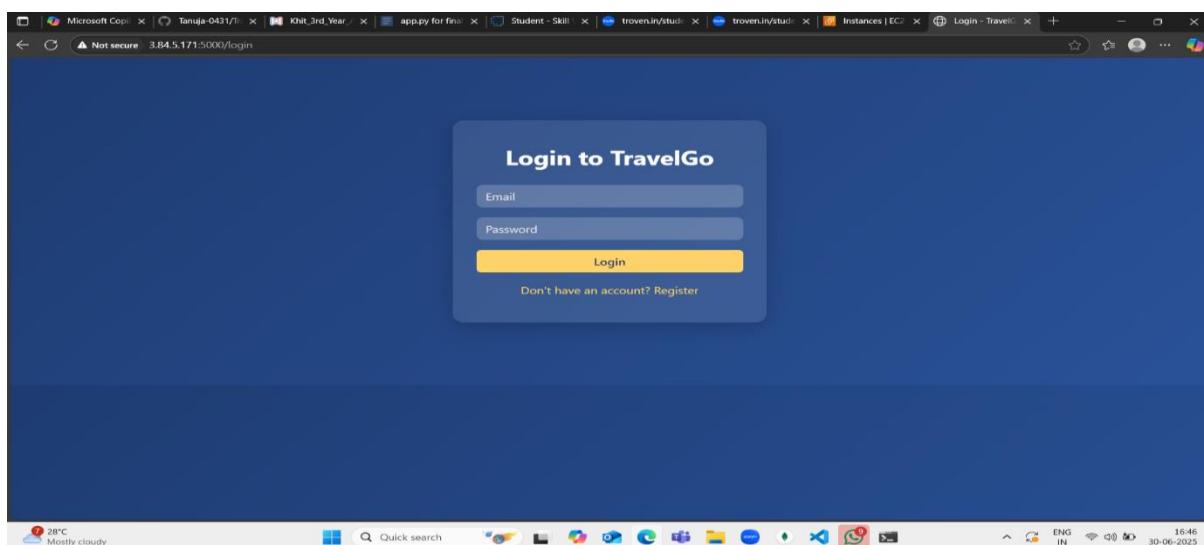
Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and perform a final round of live testing to confirm everything runs smoothly in the production environment.

Functional Testing to verify the Project

Home Page:



Login Page:





Register Page:

28°C
Mostly cloudy

Quick search

ENG IN 16:46 30-06-2025

Dashboard page:

Login successful!

Welcome, Tanuja!

Find your next destination or revisit your favorites.

Search destination... All Categories Search Now

AI102
Delhi → Mumbai
Passenger: N/A

Cancel Booking

Your Bookings

My Trips

View and manage your past and upcoming travel bookings.

Virtual Tours

Explore places virtually before visiting them in person.

Wishlist

Save destinations to your wishlist for later planning.

Bookings page:

TravelGo

Plan Your Next Trip
Book flights, buses, trains, and hotels – all in one place.

Flights **Buses** **Trains** **Hotels**

Search Flights

From: e.g. Delhi To: e.g. Mumbai Departure Date: dd-mm-yyyy

Passengers: 1

Q Search

No flights match your search. Try different cities or dates.

TravelGo

Plan Your Next Trip
Book flights, buses, trains, and hotels – all in one place.

Flights **Buses** **Trains** **Hotels**

Search Flights

From: e.g. Delhi To: e.g. Mumbai Departure Date: dd-mm-yyyy

Passengers: 1

Q Search

Available Flights

A101
Delhi → Mumbai
⌚ 2025-07-01 08:00 - 2025-07-01 10:00
₹ 5000

A102
Delhi → Mumbai
⌚ 2025-07-01 12:00 - 2025-07-01 14:00
₹ 5500

Book

TravelGo

Plan Your Next Trip
Book flights, buses, trains, and hotels – all in one place.

Flights **Buses** **Trains** **Hotels**

Confirm Your Booking

A101
Delhi → Mumbai
⌚ Departure 2025-07-01 08:00
⌚ Arrival 2025-07-01 10:00
₹ 5000

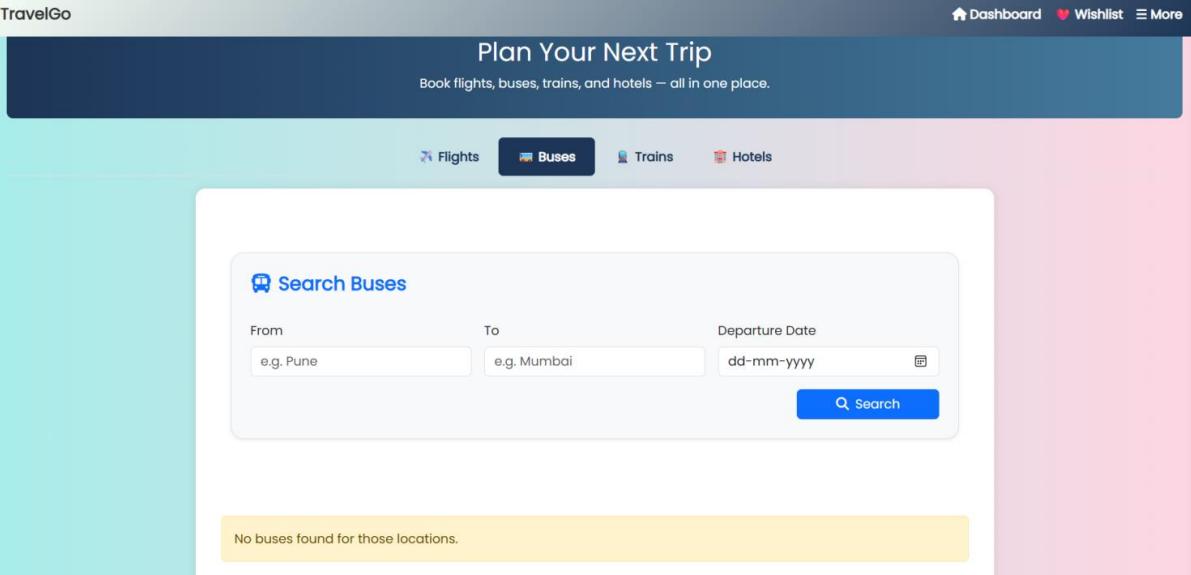
| | | | |
|----|----|----|----|
| A1 | A2 | A3 | A4 |
| B1 | B2 | B3 | B4 |
| C1 | C2 | C3 | C4 |
| D1 | D2 | D3 | D4 |
| E1 | E2 | E3 | E4 |
| F1 | F2 | F3 | F4 |

Passenger Details
Passenger 1 (Seat F3)
tonuja

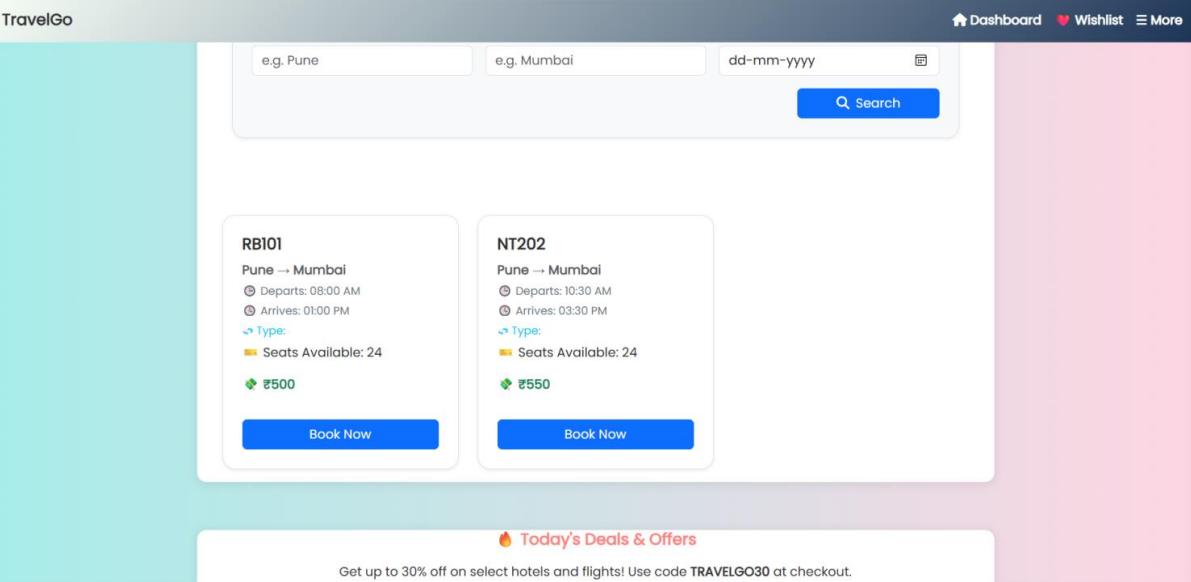
Passenger 2 (Seat F4)
sample

✓ Confirm **✗ Cancel**

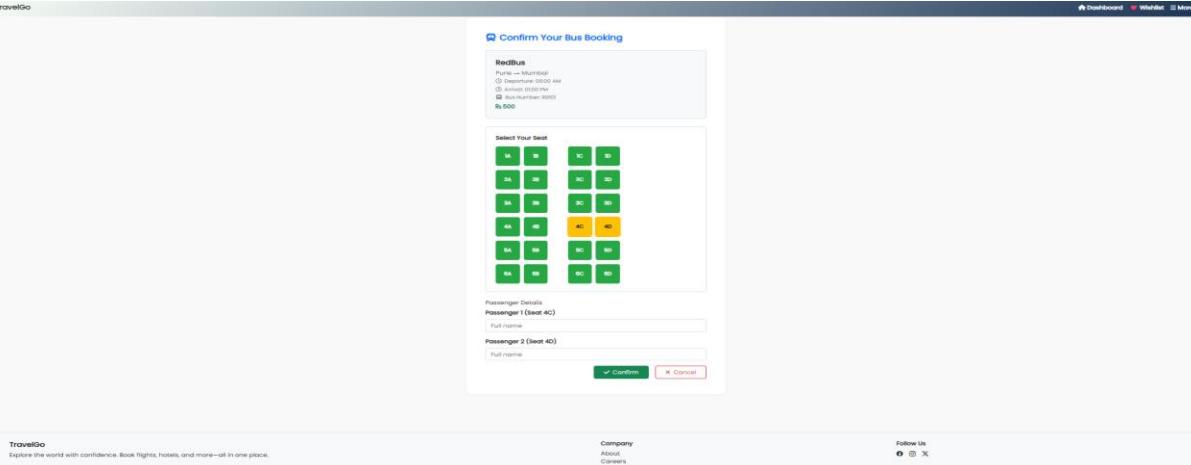
Bus bookings page:



The screenshot shows the TravelGo interface for bus bookings. At the top, there's a navigation bar with 'TravelGo' on the left and 'Dashboard', 'Wishlist', and 'More' on the right. Below the navigation is a banner with the text 'Plan Your Next Trip' and 'Book flights, buses, trains, and hotels – all in one place.' Underneath the banner are tabs for 'Flights', 'Buses' (which is selected), 'Trains', and 'Hotels'. A search form titled 'Search Buses' is centered, with fields for 'From' (e.g. Pune) and 'To' (e.g. Mumbai), a date selector for 'Departure Date' (dd-mm-yyyy), and a 'Search' button. Below the search form is a yellow message box stating 'No buses found for those locations.'



This screenshot shows the same TravelGo interface after a search. The search form fields now contain 'e.g. Pune' for 'From' and 'e.g. Mumbai' for 'To'. Below the search form, two bus route cards are displayed: 'RB101' and 'NT202'. Both cards show the route 'Pune → Mumbai', departure time (08:00 AM), arrival time (01:00 PM), seat availability (24 seats available), and a price of ₹500. Each card has a 'Book Now' button at the bottom. At the bottom of the page is a promotional banner for 'Today's Deals & Offers' with the text 'Get up to 30% off on select hotels and flights! Use code TRAVELGO30 at checkout.'



This screenshot shows the 'Confirm Your Bus Booking' page. It starts with a summary box for 'RedBus' showing the route 'Pune → Mumbai', departure time '08:00 AM', arrival time '01:00 PM', and a fare of ₹500. Below this is a 'Select Your Seat' grid where seat 4C is highlighted in orange. A 'Passenger Details' section lists 'Passenger 1 (Seat 4C)' and 'Passenger 2 (Seat 4D)', both with empty 'Full name' fields. At the bottom are 'Confirm' and 'Cancel' buttons. The footer of the page includes links for 'Company', 'About', 'Contact', and social media icons for 'Follow Us'.



Train bookings page:

The screenshot shows the 'Plan Your Next Trip' interface. At the top, there are tabs for Flights, Buses, Trains (which is selected), and Hotels. Below the tabs is a search form titled 'Search Trains'. The form includes fields for 'From' (e.g. Delhi), 'To' (e.g. Mumbai), 'Departure Date' (dd-mm-yyyy), and 'Passengers' (1). A 'Search' button is located to the right of the passengers field. Below the search form, a section titled 'Available Trains' displays a single train entry: '12345' from Delhi to Mumbai at 05:00 AM - 08:00 PM for Rs 850. A 'Book Now' button is next to it. A note 'Please fill out this field.' is displayed below the passenger field.

The screenshot shows the 'Plan Your Next Trip' interface on TravelGo. The layout is similar to the Smartbridge version, with tabs for Flights, Buses, Trains (selected), and Hotels. A search form for 'Search Trains' is present. Below it, a section titled 'Available Trains' shows a single train entry: '12345' from Delhi to Mumbai at 05:00 AM - 08:00 PM for Rs 850. A 'Book Now' button is available. A note 'Please fill out this field.' is displayed below the passenger field. A promotional banner for 'Today's Deals & Offers' is visible, stating 'Get up to 30% off on select hotels and flights! Use code TRAVELGO000 at checkout.' Below the search area, a link 'Explore the World' is shown.

The screenshot shows the 'Confirm Your Train Booking' page. It displays a summary of the booking: '12345' from Delhi to Mumbai at 05:00 AM - 08:00 PM for Rs 850. The status is 'Available'. Below this, a seating chart for 'Coach A' shows 16 green seats arranged in a 4x4 grid. A note says 'You can only select up to 1 seat(s.)'. A 'Passenger Details' field with placeholder 'Passenger Name' is shown. At the bottom, there are 'Confirm' and 'Cancel' buttons. The footer contains links for 'Company', 'About', 'Careers', and 'Contact', along with social media icons for Facebook, Twitter, and LinkedIn. A note '© TravelGo. All rights reserved.' is also present.



Hotels booking page:

Plan Your Next Trip
Book flights, buses, trains, and hotels — all in one place.

Flights Buses Trains Hotels

Search Hotels

Location: e.g. Goa Check-in Date: dd-mm-yyyy Check-out Date: dd-mm-yyyy

Min Price: e.g. 1000 Max Price: e.g. 5000 Room Type: Any

Minimum Rating: Any

No hotels match your search. Try different dates or location.

Today's Deals & Offers
Get up to 30% off on select hotels and flights! Use code TRAVELGO30 at checkout.

Explore the World

TravelGo

Dashboard Wishlist More

Available Hotels

The Grand Palace
New York
Rooms: 25 | ★ 4.5/5
₹ 180 per night

City Center Suites
New York
Rooms: 18 | ★ 4.6/5
₹ 200 per night

Today's Deals & Offers
Get up to 30% off on select hotels and flights! Use code TRAVELGO30 at checkout.

TravelGo

Dashboard Wishlist More

Confirm Hotel Booking

City Center Suites
New York
Rooms: 18 | ★ 4.6/5 | ₹ 200 per night
Check-in: 2025-06-26 | Check-out: 2025-06-29

Room Type: Standard
Total Nights: 3
Total Price: ₹ 600



SMARTBRIDGE
Data Bridge One Step

Recent bookings:

TravelGo

Welcome, Tanuja!

Find your next destination or revisit your favorites.

Search destination... All Categories Search Now

A102 Delhi — Mumbai Passenger N/A Seats 1 07/07/2023 12:00 - 07/07/2023 14:00 ₹ 1500 Cancel Booking

A102 Delhi — Mumbai Passenger N/A Seats A2 07/07/2023 12:00 - 07/07/2023 14:00 ₹ 1500 Cancel Booking

A102 Delhi — Mumbai Passenger N/A Seats A1 07/07/2023 12:00 - 07/07/2023 14:00 ₹ 1500 Cancel Booking

TravelGo

Your Bus Bookings

NT202 Pune — Mumbai Passenger sample! Seats 10 10:00 AM - 03:00 PM ₹ 1500 Cancel Booking

Your Train Bookings

98765 Kolkata — Patna Seats 10 11:15 AM - 08:00 PM ₹ 1500 Cancel Booking

12345 Delhi — Mumbai Seats 41 Passengers tanuja 08:00 AM - 08:00 PM ₹ 1500 Cancel Booking

Your Hotel Bookings

City Center Suites New York 07/08/2023 - 08/08/2023 ₹ 1500 Cancel Booking

You haven't planned any trips yet.

My Trips View and manage your past and upcoming travel

Virtual Tours Explore places virtually before visiting them in person

Wishlist Save destinations to your wishlist for later planning.

Conclusion

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

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

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