Name : ALLAM AJAY KUMAR

EMAIL : 238x5a0516@khitguntur.ac.in

COLLEGE : KALLAM HARANADHREDDY INSTITUTE OF TECHNOLOGY

Project Name : TravelGo: A Cloud-Powered Real-TimeTravel Booking PlatformUsing AWS

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

Project Description:

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

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

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

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

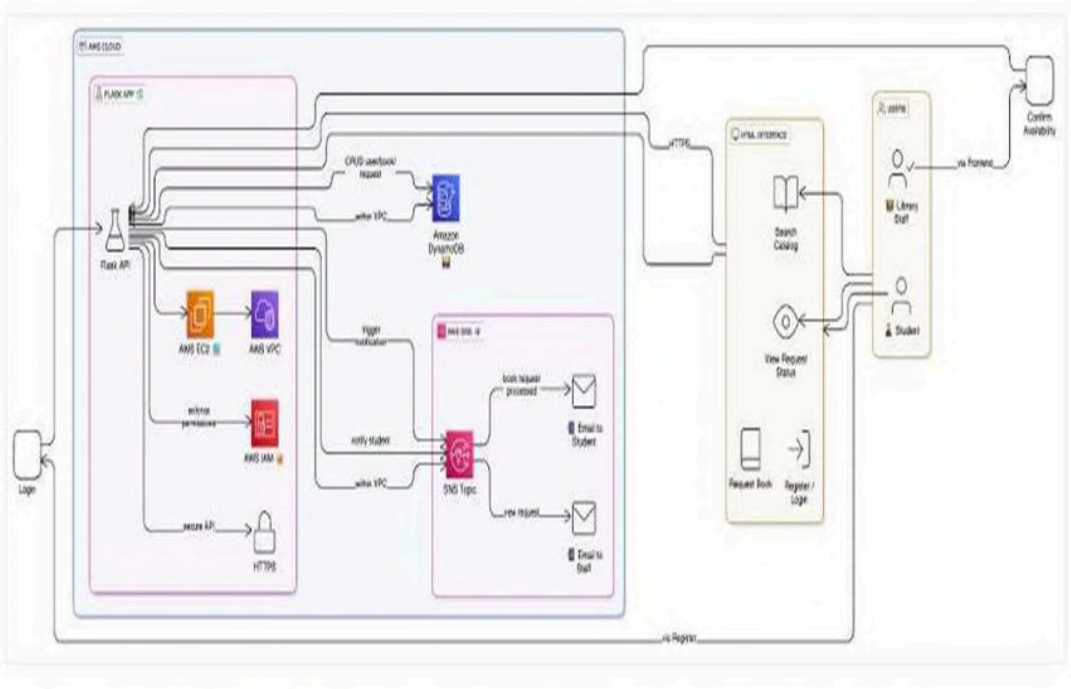
Scenario 2: Real-Time BookingConfirmation with AWS SNS

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.

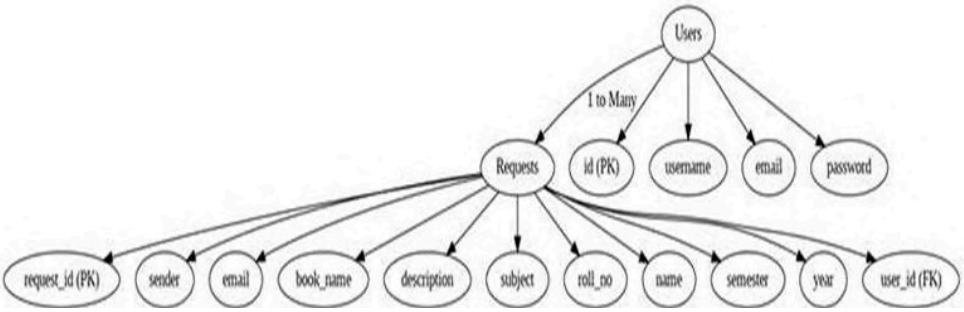
Scenario 3: Dynamic Dashboard with Personal TravelHistory

TravelGo features a dynamic user dashboard that displays all past and upcoming bookings for the logged-in user. For example, a user who has booked a flight and a hotel can view these bookings categorized by type, along with dates, price, and cancellation options. Flask fetches this data from AWS DynamoDB, which persistently stores all user bookings. The dashboard UI, powered by responsive HTML/CSS and Flask templates, ensures users can review or manage bookings anytime, from any device, with real-time updates and quick cancellation workflows supported.

AWS ARCHITECTURE



Entity Relationship (ER)Diagram:



Project WorkFlow:

1. **AWS troven access Setup and Login**

a. : Set up an troven access

b. : Open the AWS Management Console

2. **DynamoDB DatabaseCreation and Setup**

a. : Create a DynamoDB Table.

b. : Configure Attributes for User Data and BookingRequests.

3. **SNS Notification Setup Activity**

a. : Create SNS topics.
b. : Subscribe users and library staff to SNS email notifications.

4. **Backend Development and Application Setup**

4.1:Develop the Backend Using Flask.
4.2: IntegrateAWS Services Using boto3.

5. **IAM Role Setup**

a. : Create IAM Role
b. : Attach Policies

6. **EC2 InstanceSetup Activity**

a. : Launch an EC2 instance to host the Flask application.

b. : Configure security groups for HTTP and SSH access.

7. **Deployment on EC2 Activity**

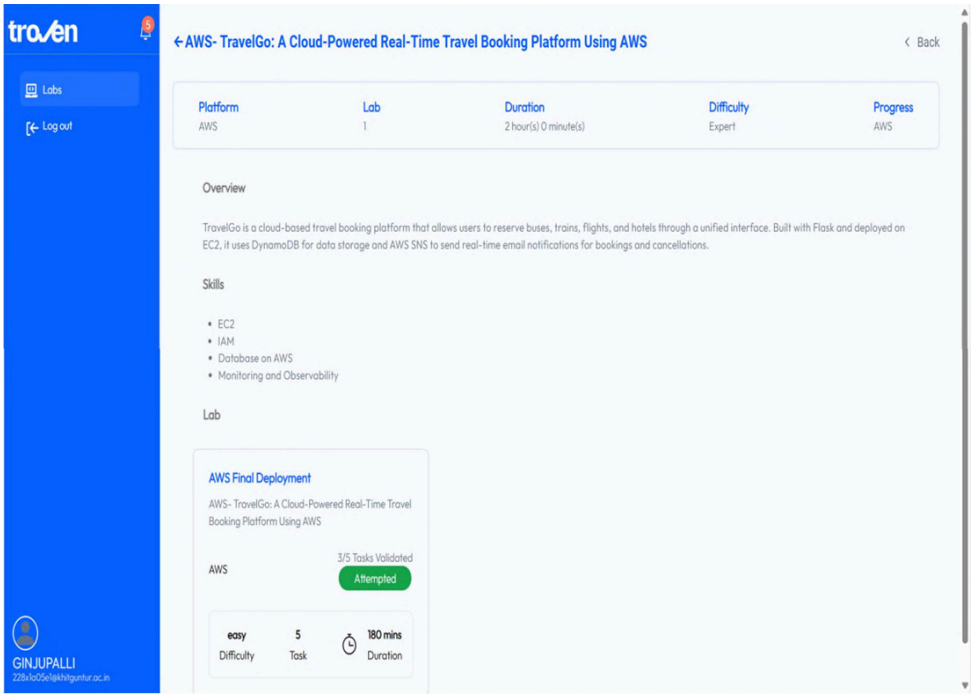
a. :Upload Flask Files Activity

b. : Run the Flask App

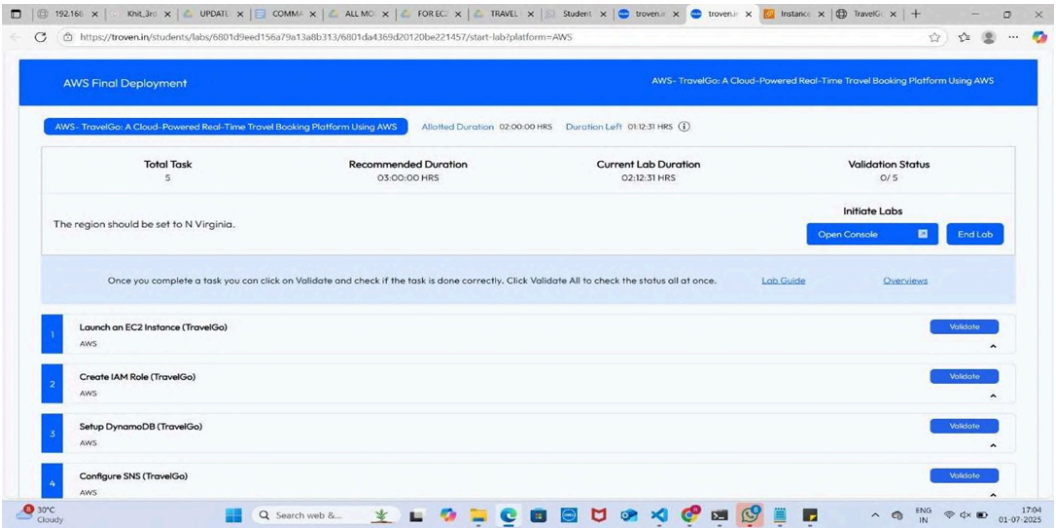
8. **Testing and Deployment Activity:** Conduct functional testing to verify user registration,login,bookings, and notifications.

1. **AWS troven accessSetup and Login**

Activity 1.1: Set up an troven access



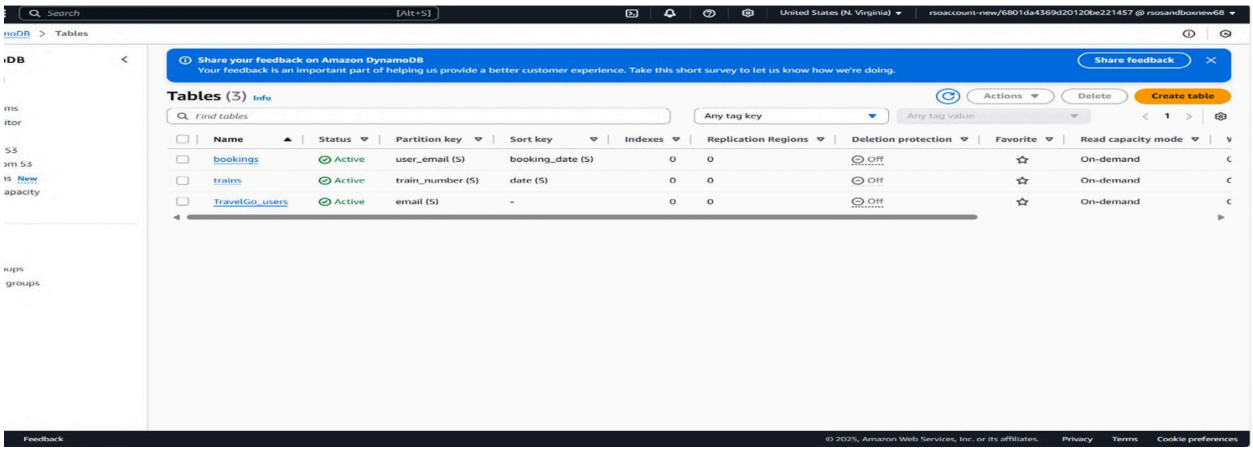
Activity 1.2: Open the AWS Management Console



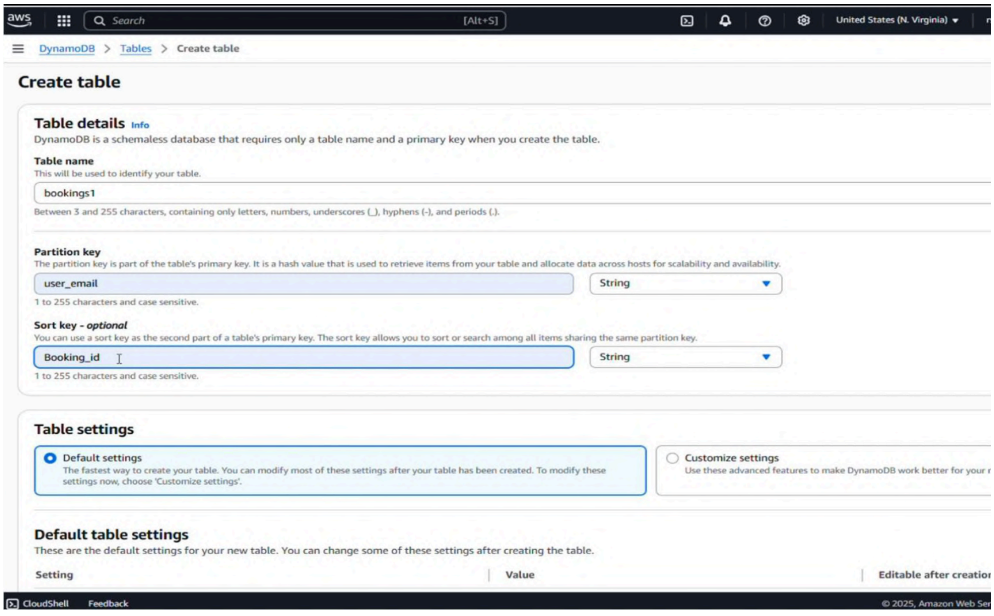
2. DynamoDB Database Creation and Setup

Activity 2.1: Create a DynamoDB Table

To create a DynamoDB table,go to the AWS Management Console, navigate to **DynamoDB**, and click **"Create table."** Specify a **table name** and define a **primary key** (e.g., Email as Partition Key). Choose additional settings like **read/write capacity mode** (on-demand or provisioned) and click **"Create."** Your table will be ready in seconds for data operations.



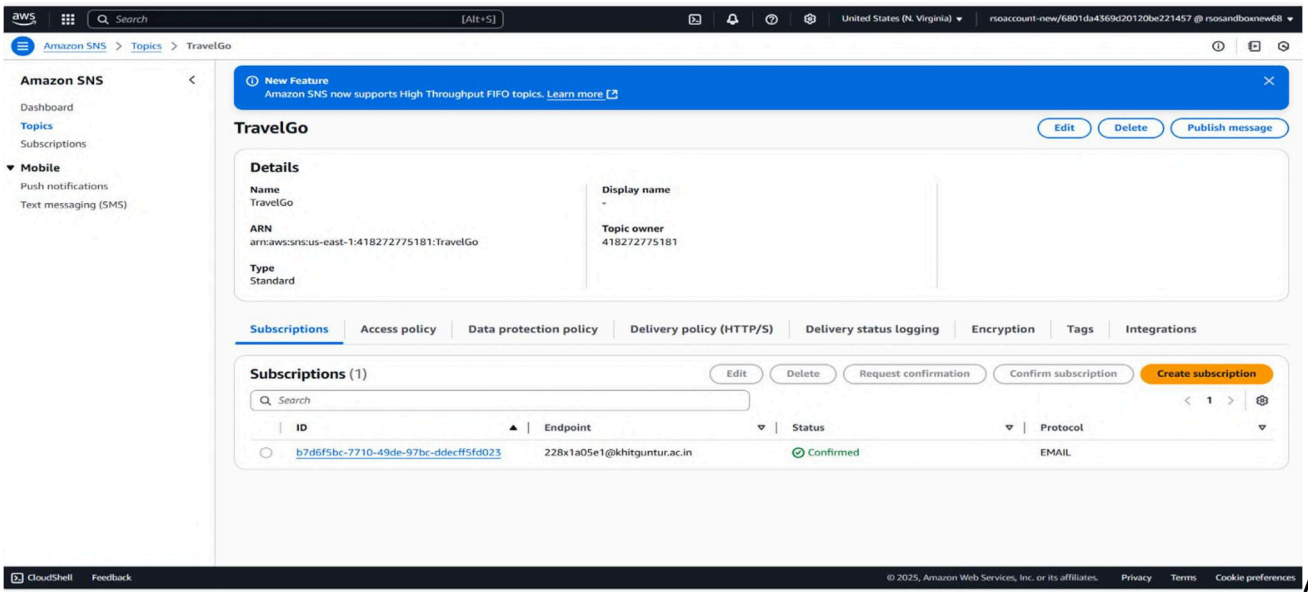
Activity 2.2: ConfigureAttributes for User Data and Booking Requests



3. SNS Notification Setup

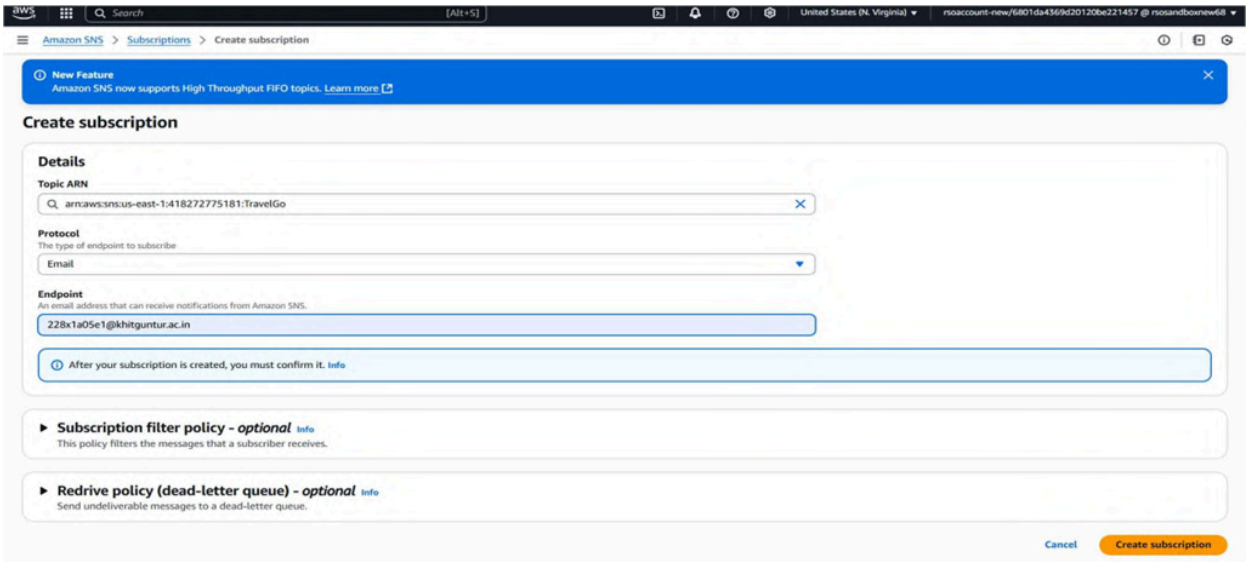
For **SNS Notification Setup**, go to the AWS Console and open the **Simple Notification Service (SNS)** dashboard. Click **“Create topic”**, choose the **Standard** type, and name your topic. After creating it, copy the **Topic ARN**. Then, **subscribe email addresses** (students/admins) to the topic and confirm the subscription via email.

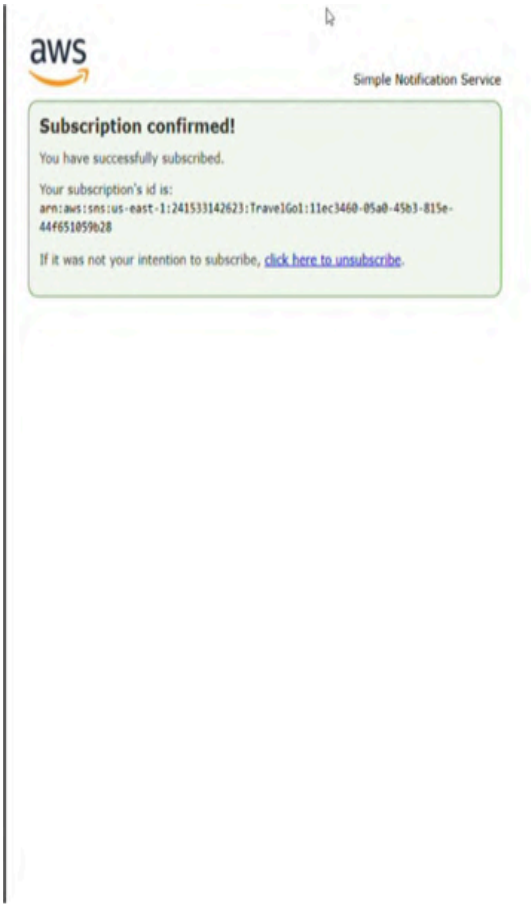
Activity 3.1: Create SNS topics.



A

Activity 3.2: Subscribe users to SNS email notifications

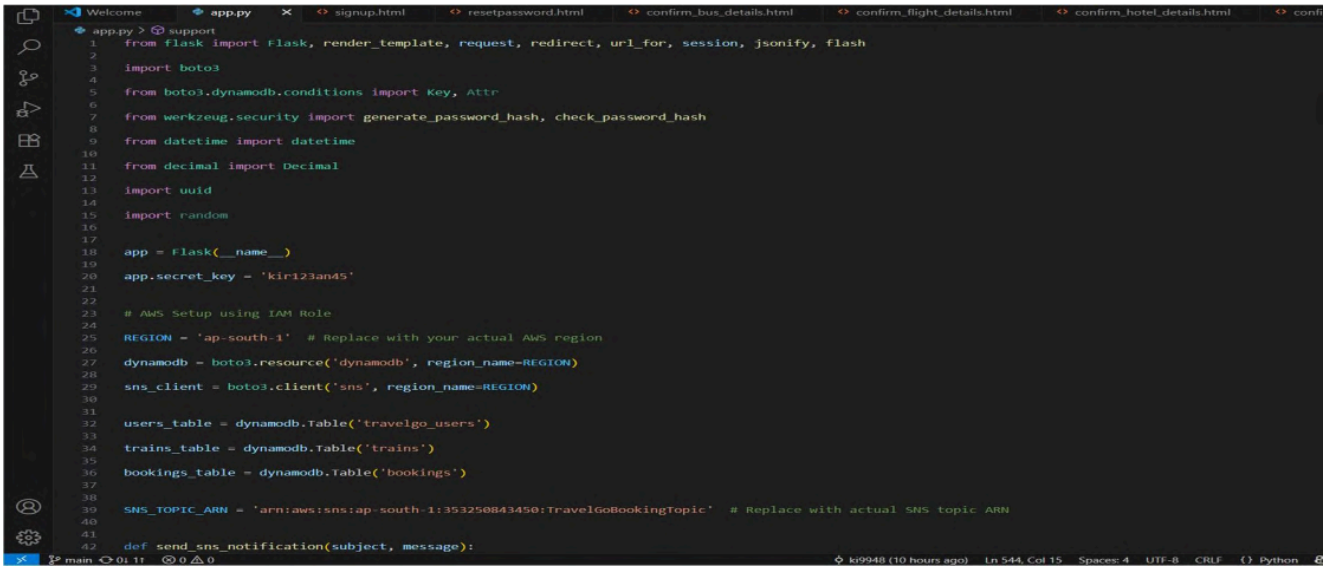




4. Backend Development and Application Setup

In the Backend Development and Application Setup,use Flask to build the web application and integrate AWS services via the boto3 library. Set up routes for user registration, login, book requests, and page navigation. Connect to DynamoDB to store user and request data, and configure AWS SNS to send notifications. Organize files into app.py, templates/, and static/ folders for a clean project structure.

Activity 4.1:Develop the Backend UsingFlask.



```python
from flask import Flask, render_template, request, redirect, url_for, session, jsonify, flash

import boto3

from boto3.dynamodb.conditions import Key, Attr

from werkzeug.security import generate_password_hash, check_password_hash

from datetime import datetime

from decimal import Decimal

import uuid

import random

app = Flask(__name__)

app.secret_key = 'kir123an45'

# AWS Setup using IAM Role

REGION = 'ap-south-1'  # Replace with your actual AWS region

dynamodb = boto3.resource('dynamodb', region_name=REGION)

sns_client = boto3.client('sns', region_name=REGION)

users_table = dynamodb.Table('travelgo_users')

trains_table = dynamodb.Table('trains')

bookings_table = dynamodb.Table('bookings')

SNS_TOPIC_ARN = 'arn:aws:sns:ap-south-1:353250843450:TravelGoBookingTopic'  # Replace with actual SNS topic ARN

def send_sns_notification(subject, message):
```

Activity 4.2: Integrate AWS Services Using boto3

```python
from flask import Flask, render_template, request, redirect, url_for, session, jsonify, flash
import boto3
from boto3.dynamodb.conditions import Key, Attr
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime
from decimal import Decimal
import uuid
import random

app = Flask(__name__)
app.secret_key = 'kir123an45' # IMPORTANT: Change this to a strong, random key in production!

# AWS Setup using IAM Role
REGION = 'us-east-1'  # Replace with your actual AWS region
dynamodb = boto3.resource('dynamodb', region_name=REGION)
sns_client = boto3.client('sns', region_name=REGION)

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

SNS_TOPIC_ARN = 'arn:aws:sns:us-east-1:418272775181:TravelGo:b7d6f5bc-7710-49de-97bc-ddecff5fd023'  # Replace with actual SNS topic
```

C for SNS

```python
# This function is duplicated in the original code, removing the duplicate.
def send_sns_notification(subject, message):
    try:
        sns_client.publish(
            TopicArn=SNS_TOPIC_ARN,
            Subject=subject,
            Message=message
        )
    except Exception as e:
        print(f"SNS Error: Could not send notification - {e}")
        # Optionally, flash an error message to the user or log it more robustly.

# Routes
@app.route('/')
```

CODE for Register

```python
@app.route('/')
def index():
    return render_template('index.html')

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

        # Check if user already exists
        # This uses get_item on the primary key 'email', so no GSI needed.
        existing = users_table.get_item(Key={'email': email})
        if 'Item' in existing:
            flash('Email already exists!', 'error')
            return render_template('register.html')

        # Hash password and store user
        hashed_password = generate_password_hash(password)
        users_table.put_item(Item={'email': email, 'password': hashed_password})
        flash('Registration successful! Please log in.', 'success')
        return redirect(url_for('login'))
    return render_template('register.html')

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

        # Retrieve user by email (primary key)
        user = users_table.get_item(Key={'email': email})

        # Authenticate user
        if 'Item' in user and check_password_hash(user['Item']['password'], password):
            session['email'] = email
            flash('Logged in successfully!', 'success')
            return redirect(url_for('dashboard'))
```

Code for Dashboard

```python

@app.route('/dashboard')
def dashboard():
    if 'email' not in session:
        return redirect(url_for('login'))
    user_email = session['email']

    # Query bookings for the logged-in user using the primary key 'user_email'
    # No GSI is needed here as 'user_email' is likely the partition key for the bookings_table.
    response = bookings_table.query(
        KeyConditionExpression=Key('user_email').eq(user_email),
        ScanIndexForward=False # Get most recent bookings first
    )
    bookings = response.get('Items', [])

    # Convert Decimal types from DynamoDB to float for display if necessary
    for booking in bookings:
        if 'total_price' in booking:
            try:
                booking['total_price'] = float(booking['total_price'])
            except (TypeError, ValueError):
                booking['total_price'] = 0.0 # Default value if conversion fails
    return render_template('dashboard.html', username=user_email, bookings=bookings)
```

Code for Cancel booking

```python

@app.route('/cancel_booking', methods=['POST'])
def cancel_booking():
    if 'email' not in session:
        return redirect(url_for('login'))

    booking_id = request.form.get('booking_id')
    user_email = session['email']
    booking_date = request.form.get('booking_date') # This is crucial as it's the sort key

    if not booking_id or not booking_date:
        flash("Error: Booking ID or Booking Date is missing for cancellation.", 'error')
        return redirect(url_for('dashboard'))

    try:
        # Delete item using the primary key (user_email and booking_date)
        # This does not use GSI, so it remains unchanged.
        bookings_table.delete_item(
            Key={'user_email': user_email, 'booking_date': booking_date}
        )
        flash(f"Booking {booking_id} cancelled successfully!", 'success')
    except Exception as e:
        flash(f"Failed to cancel booking {booking_id}: {str(e)}", 'error')

    return redirect(url_for('dashboard'))


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

```

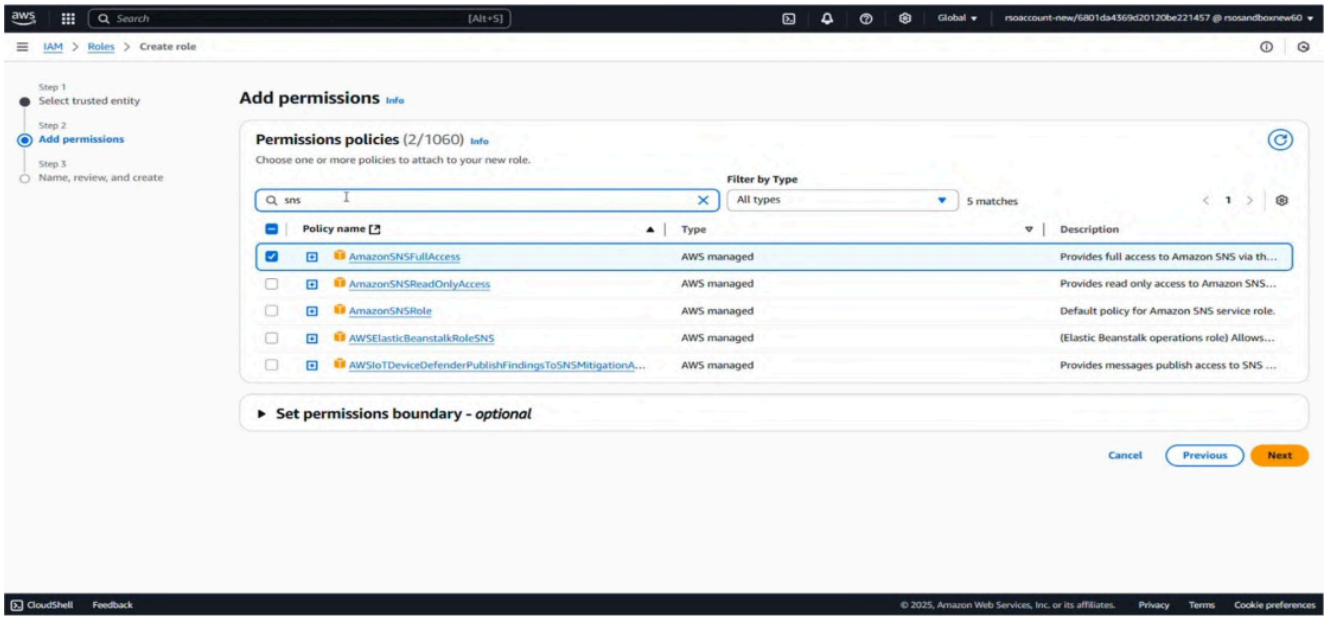
5. IAM Role Setup

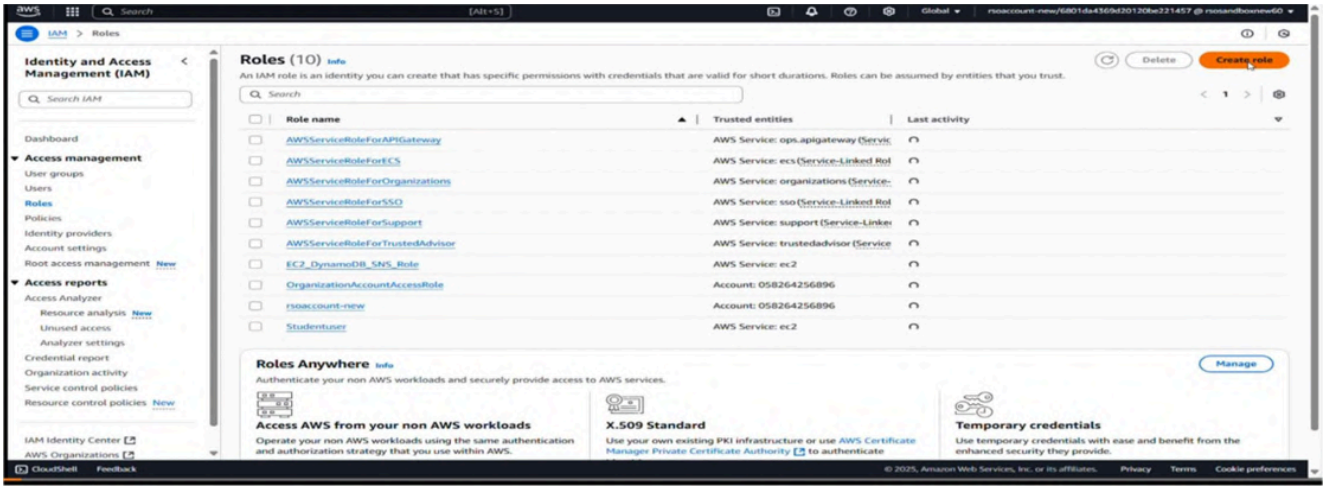
For IAM Role Setup,go to the AWS Console and create a new IAM Role for your EC2 instance.

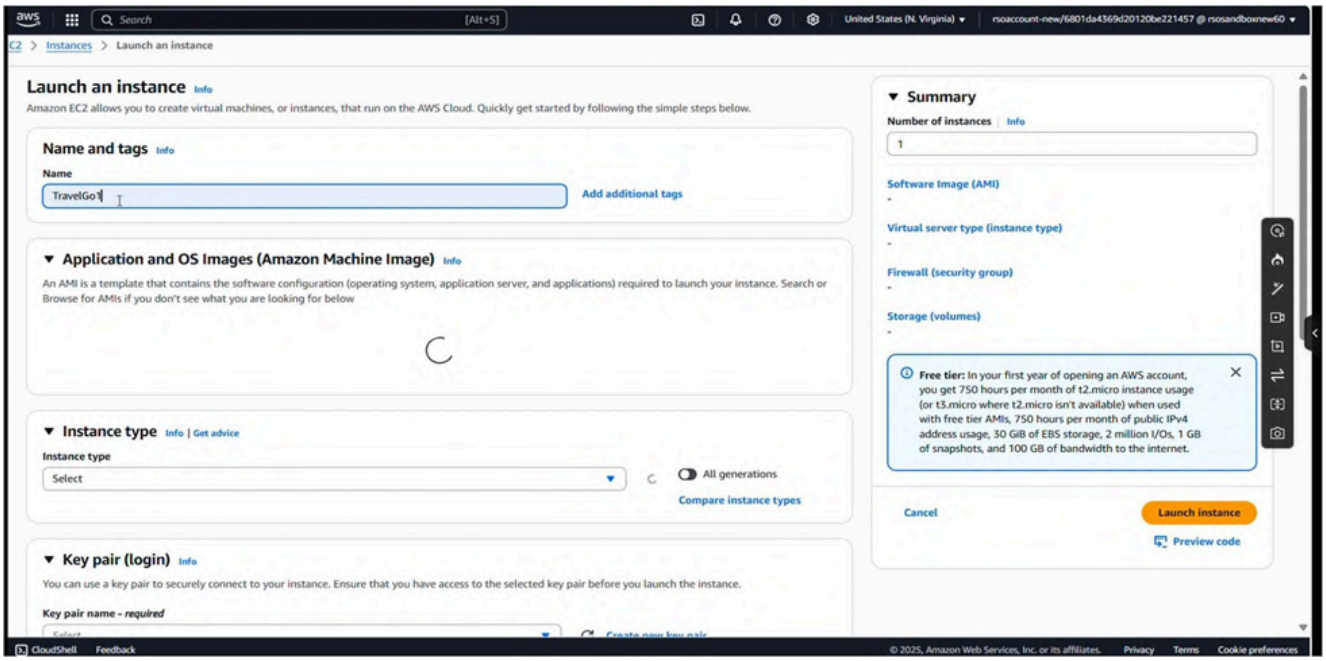
Attach AmazonDynamoDBFullAccess and AmazonSNSFullAccess policies to allow interaction with DynamoDB and SNS. Assign this role to your EC2 instance during or after launch. This ensures secure and authorized access to AWS resources from your Flask backend.

Activity 5.1: CreateIAM Role

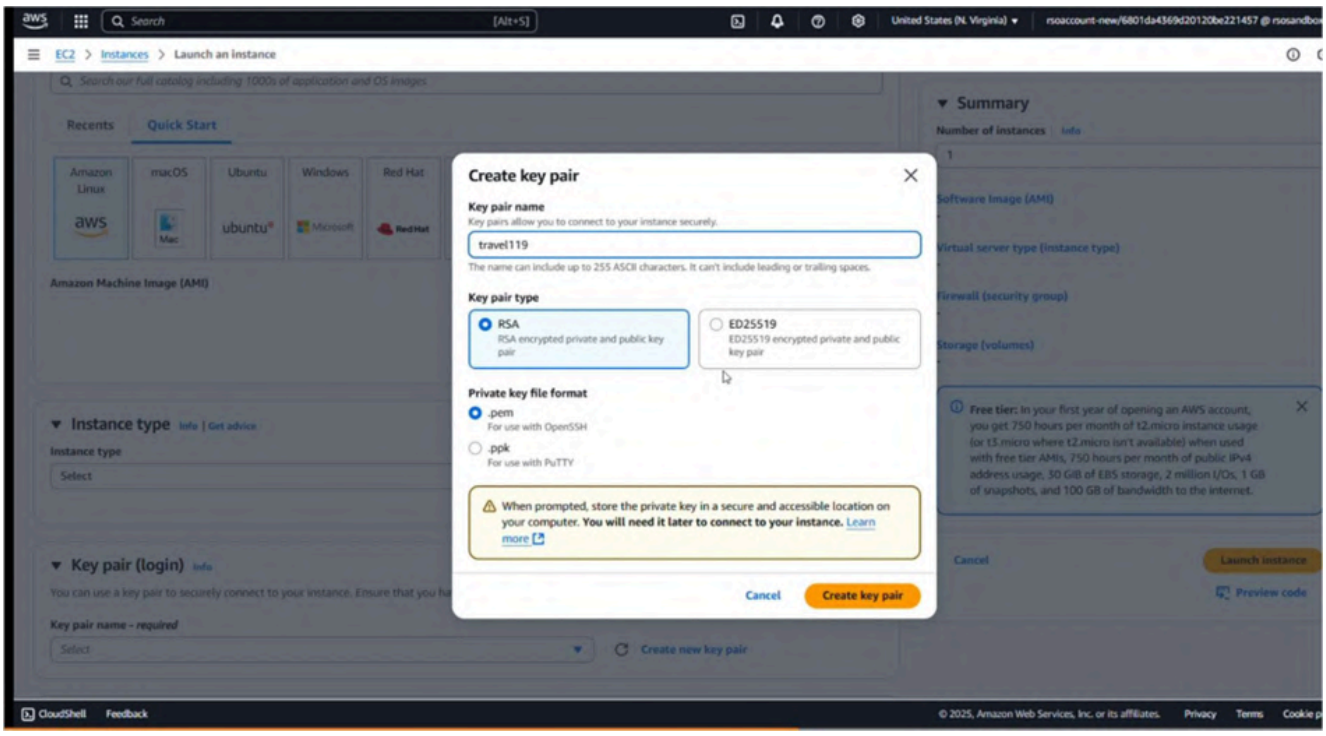
Activity 5.2: Attach Policies



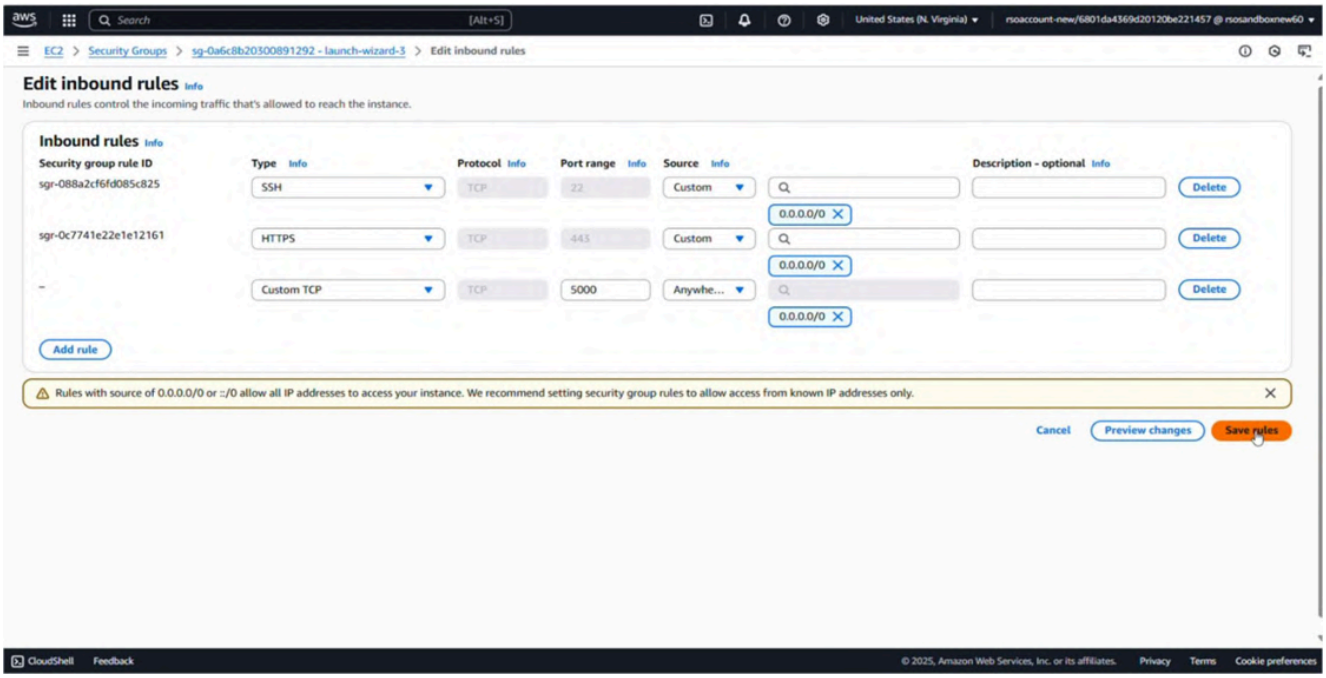


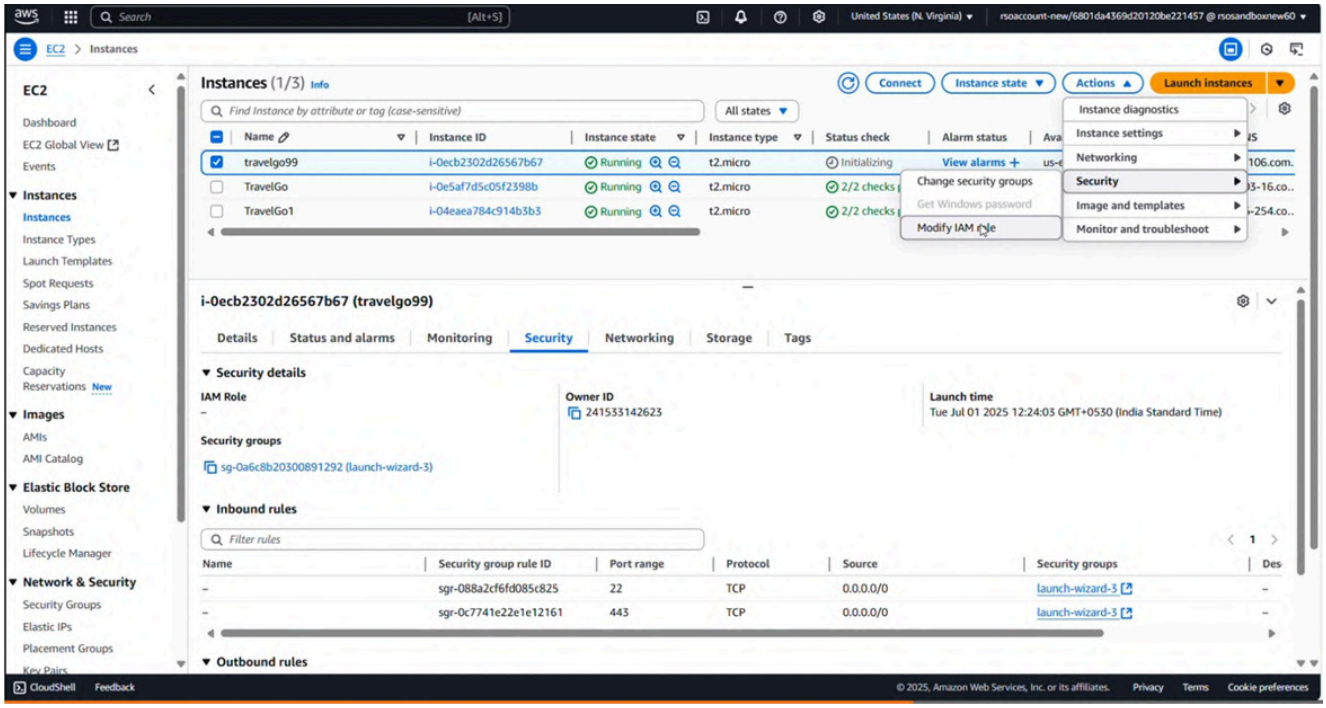


create a key pair:

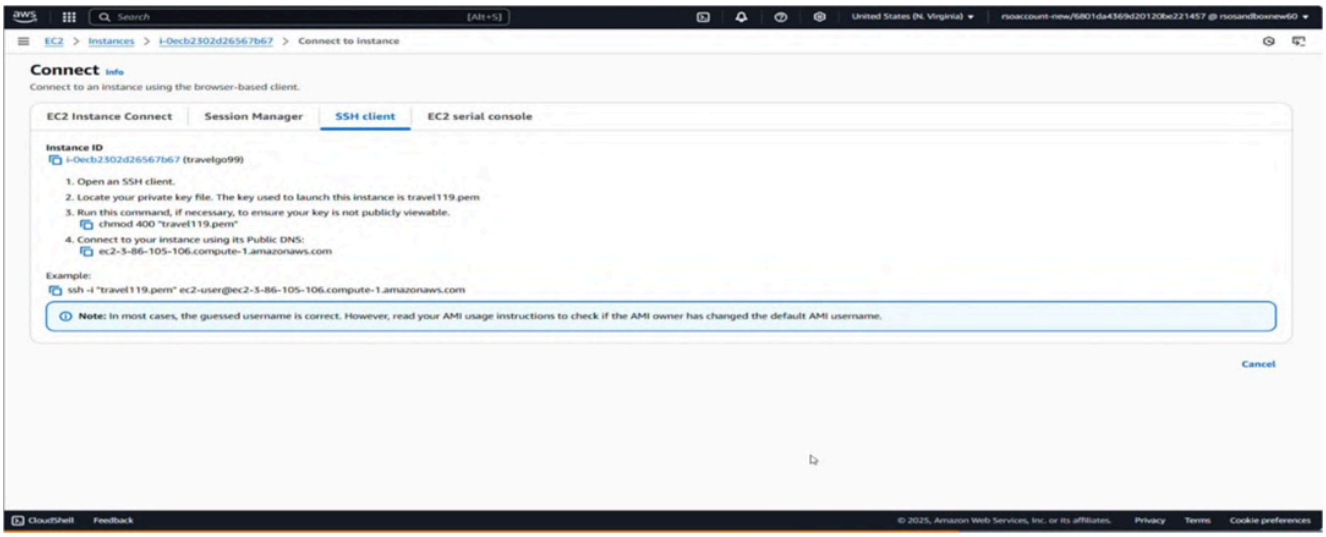


Edit of Security group:





EC2 Connect:



Milestone 7: Deployment on EC2

Activity 7.1: Install Software on the EC2 Instance

Install Python3,Flask, and Git: On AmazonLinux 2:

sudo yum update

-y sudo yum

install python3

git sudo pip3 install flask

boto3 Verify Installations:

flask --versiongit --version

Activity 7.2:Clone Your Flask Project from GitHub

Clone your project repository from GitHub into the EC2 instance usingGit.

Run: ‘git clone https://github.com/your-github-username/your-repository-name.git’

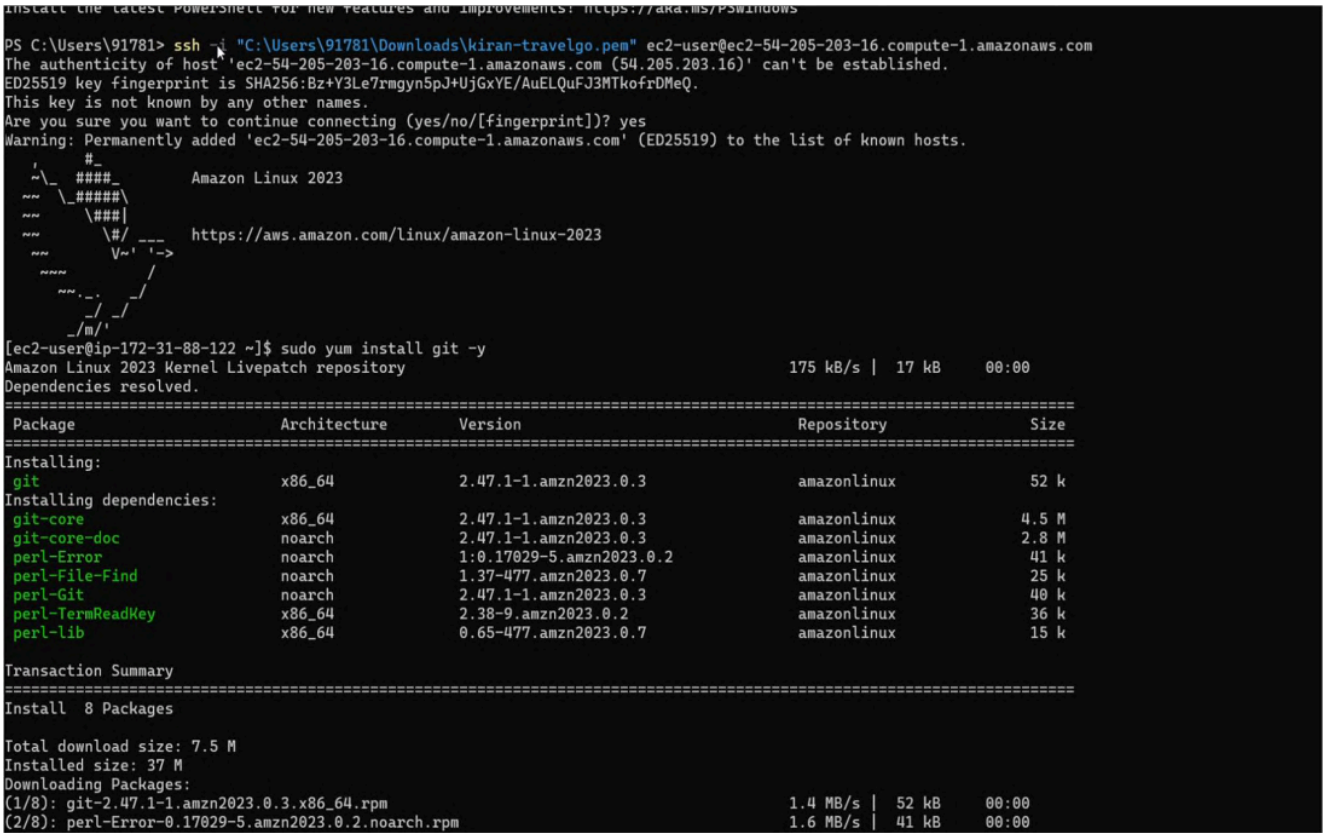
Note: change your-github-username and your-repository-name with your credentials here: ‘git clone https://github.com/AlekhyaPenubakula/InstantLibrary.git’

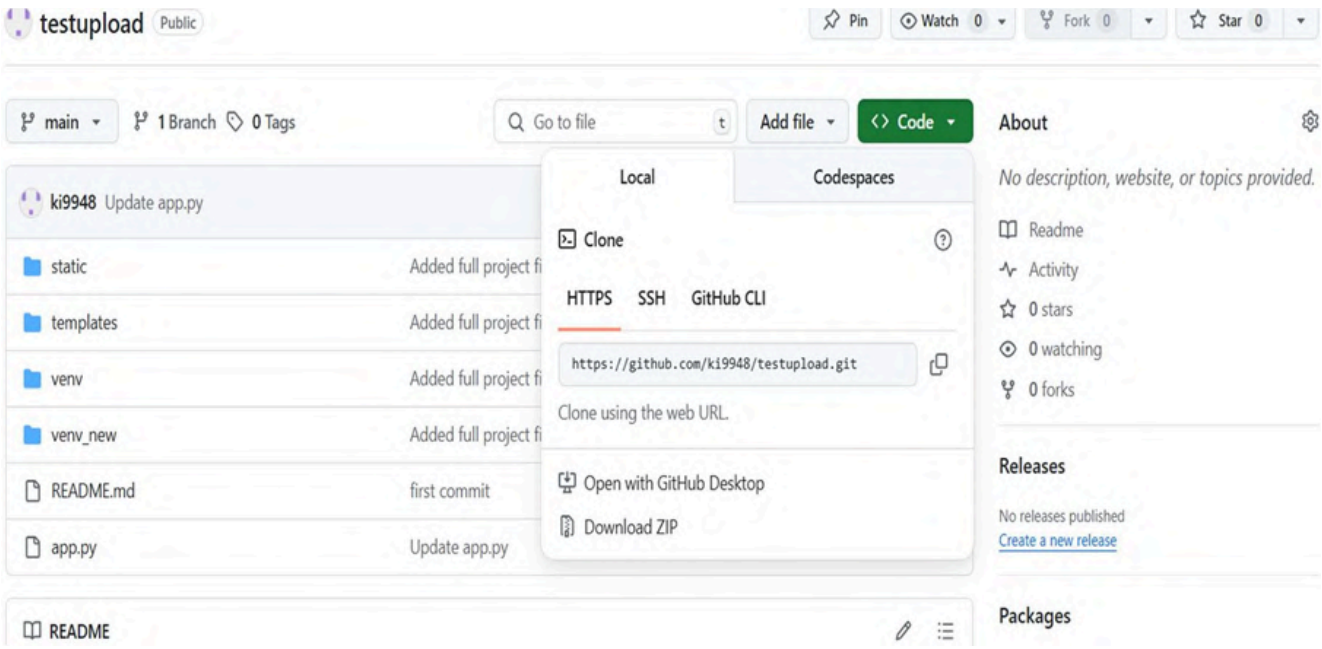
This will download your project to the EC2 instance.

To navigate to the project directory, run the following command:

cd InstantLibrary

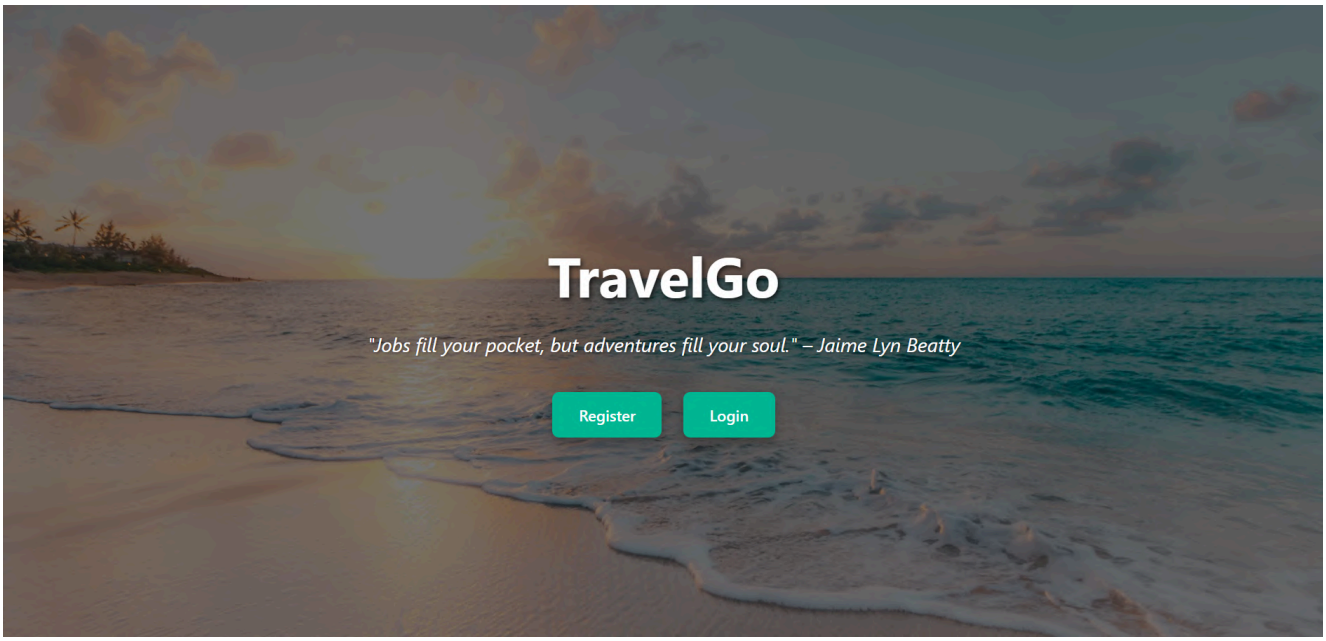
Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:Run the Flask Application : sudo flask run --host=0.0.0.0 --port=80



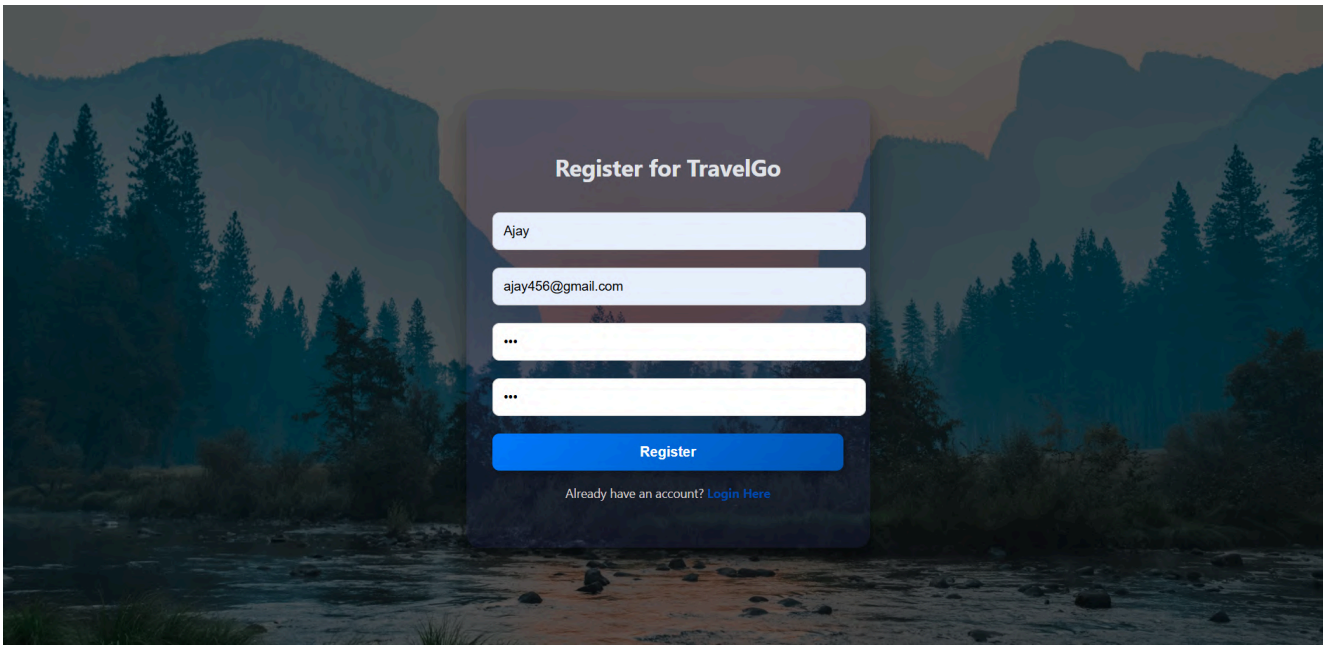



```
Collecting itsdangerous>=2.2.0
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting blinker>=1.9.0
  Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Collecting werkzeug>=3.1.0
  Downloading werkzeug-3.1.3-py3-none-any.whl (224 kB)
     |████████████████████████████████| 224 kB 56.0 MB/s
Collecting zipp>=3.20
  Downloading zipp-3.23.0-py3-none-any.whl (10 kB)
Installing collected packages: zipp, markupsafe, werkzeug, jinja2, itsdangerous, importlib-metadata, click, blinker, flask
Successfully installed blinker-1.9.0 click-8.1.8 flask-3.1.1 importlib-metadata-8.7.0 itsdangerous-2.2.0 jinja2-3.1.6 markupsafe-3.0.2 werkzeug-3.1.3 zipp-3
.23.0
[ec2-user@ip-172-31-88-122 ~]$ pip install boto3
Defaulting to user installation because normal site-packages is not writeable
Collecting boto3
  Downloading boto3-1.39.0-py3-none-any.whl (139 kB)
     |████████████████████████████████| 139 kB 13.8 MB/s
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (from boto3) (0.10.0)
Collecting botocore<1.40.0,>=1.39.0
  Downloading botocore-1.39.0-py3-none-any.whl (13.8 MB)
     |████████████████████████████████| 13.8 MB 46.6 MB/s
Collecting s3transfer<0.14.0,>=0.13.0
  Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB)
     |████████████████████████████████| 85 kB 6.3 MB/s
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.0->boto3) (1.25.10)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.0->boto3) (2.8.1)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.40.0,>=1.39.0->boto3) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed boto3-1.39.0 botocore-1.39.0 s3transfer-0.13.0
[ec2-user@ip-172-31-88-122 ~]$ cd TravelGok
[ec2-user@ip-172-31-88-122 TravelGok]$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.88.122:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 696-443-691
```

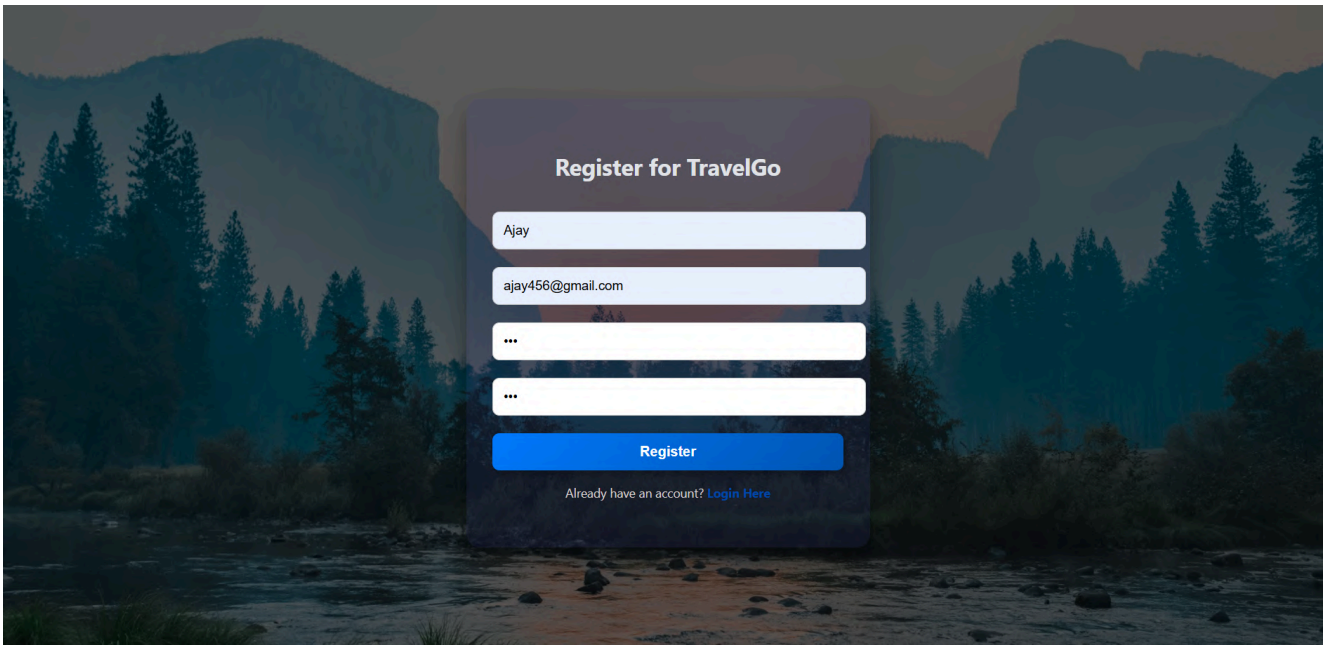
Index Page:



Register Page :



Login Page :



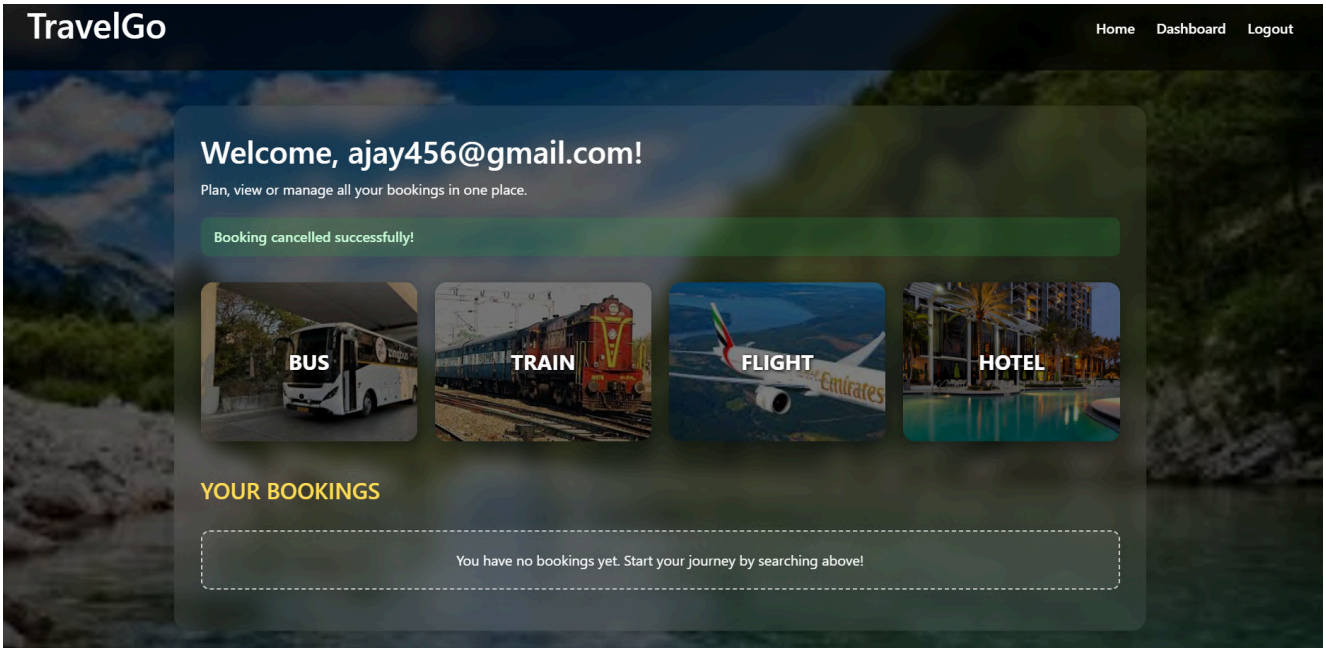
Home Code:

```
templates > home.html > html > head > style
<html lang="en">
<head>
    <style>

        .dashboard-btn:hover {
            background-color: #f0c040;
            color: #000;
        }

        .content {
            height: 100%;
            display: flex;
            justify-content: center;
            align-items: center;
            text-align: center;
            color: white;
            padding: 20px;
        }

        .content h1 {
            font-size: 60px;
            margin-bottom: 20px;
        }

        .content p {
            font-size: 24px;
        }
    </style>
</head>
<body>
    <div class="overlay">
        <a href="/dashboard" class="dashboard-btn">Dashboard</a>
        <div class="content">
            <div>
                <h1>Welcome to TravelGo</h1>
                <p>Start Booking your Ticket and Safe Journey!</p>
            </div>
        </div>
    </div>
</body>
</html>
```

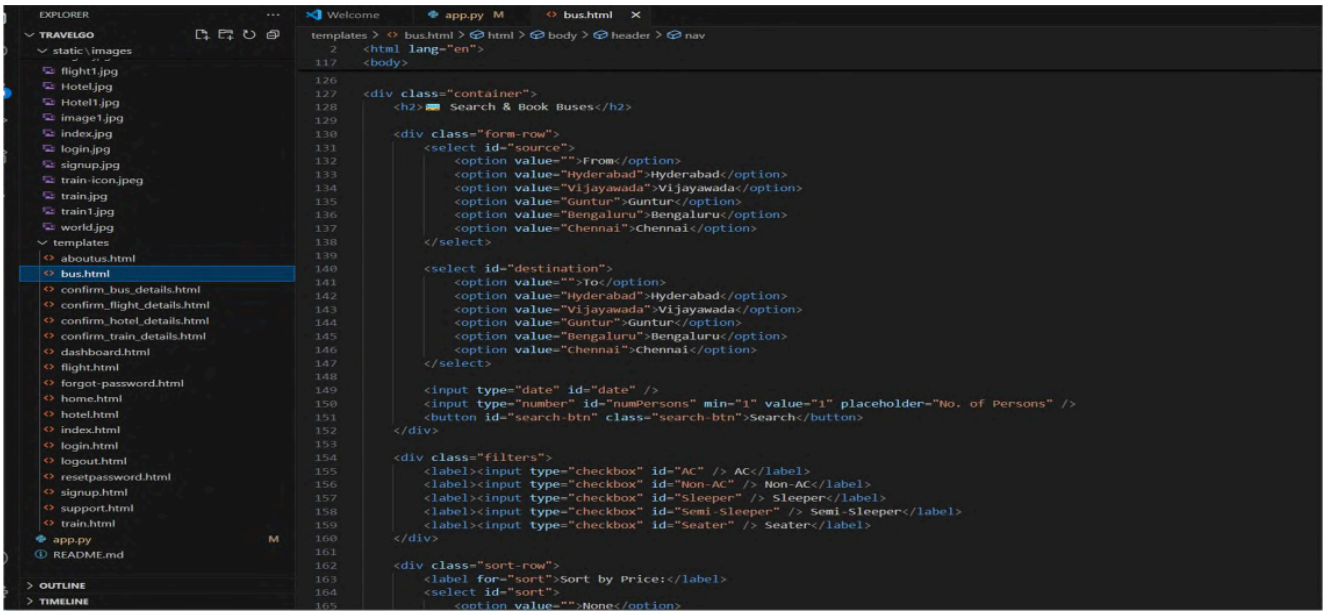
Dashboard Code :

```
Welcome  app.py M  dashboard.html
templates > dashboard.html > html > body > div.dashboard-container > div.bookings > div.booking-item > div.booking-actions > form > button.cancel-
  2   <html>
153   <body>
154
155     <div class="navbar">
156       <h1>TravelGo</h1>
157       <nav>
158         <a href="/home">Home</a>
159         <a href="/aboutus">About Us</a>
160         <a href="/support">Support</a>
161         <a href="/logout" class="logout">Logout</a>
162       </nav>
163     </div>
164
165     <div class="dashboard-container">
166       <p class="Welcome">Welcome, {{ username }}!</p>
167
168       <div class="booking-options">
169         <a href="/bus"><img src="../static/images/bus1.jpg" alt="Bus"></a>
170         <a href="/train"><img src="../static/images/train1.jpg" alt="Train"></a>
171         <a href="/flight"><img src="../static/images/flight1.jpg" alt="Flight"></a>
172         <a href="/hotel"><img src="../static/images/Hotel1.jpg" alt="Hotel"></a>
173       </div>
174
175       <div class="bookings">
176         <h3>YOUR BOOKINGS</h3>
177         {% if bookings %}
178           {% for booking in bookings %}
179             <div class="booking-item">
180               <div class="booking-details-summary">
181                 <p><strong>{{ booking.booking_type|capitalize }} Booking</strong></p>
182
183                 {% if booking.booking_type == 'hotel' %}
184                   <p><strong>Hotel:</strong> {{ booking.name }}</p>
185                   <p><strong>Location:</strong> {{ booking.location }}</p>
186                   <p><strong>Check-in:</strong> {{ booking.checkin_date }}</p>
187                   <p><strong>Check-out:</strong> {{ booking.checkout_date }}</p>
188                   <p><strong>Rooms:</strong> {{ booking.num_rooms }}</p>
189                   <p><strong>Guests:</strong> {{ booking.num_guests }}</p>
190
191                 {% elif booking.booking_type == 'train' %}
192                   <p><strong>Train:</strong> {{ booking.name }} ({{ booking.train_number }})</p>
193                   <p><strong>Route:</strong> {{ booking.source }} → {{ booking.destination }}</p>
```

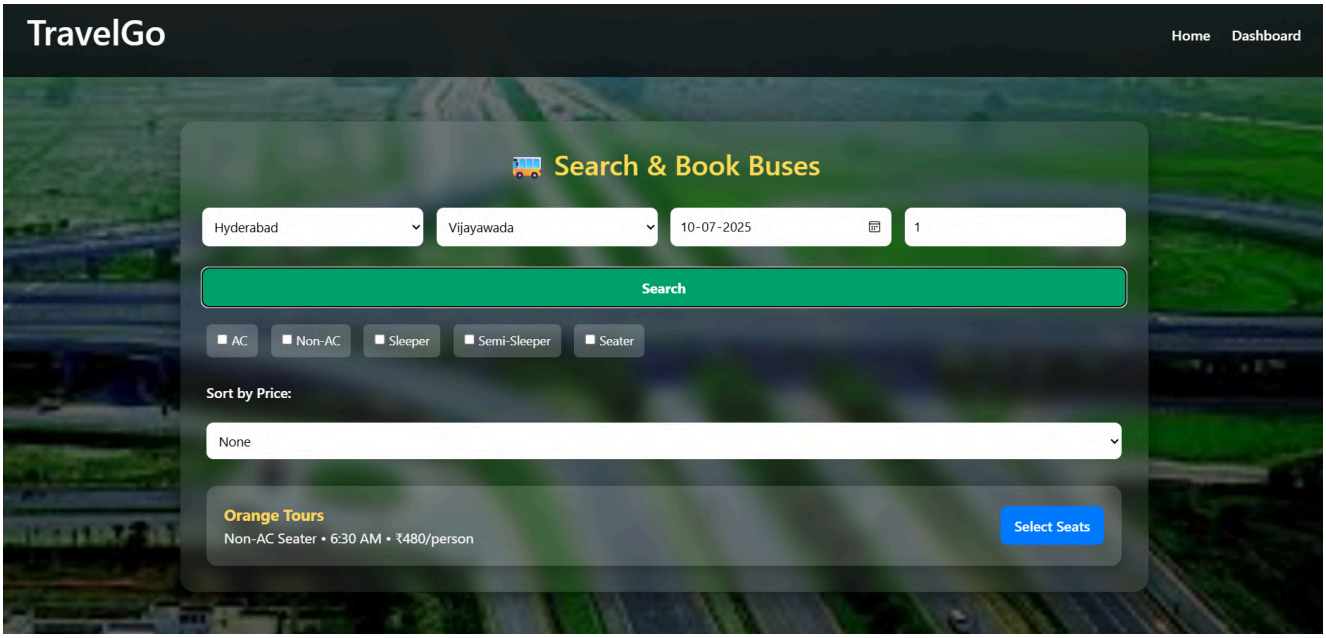
Dashboard Page :



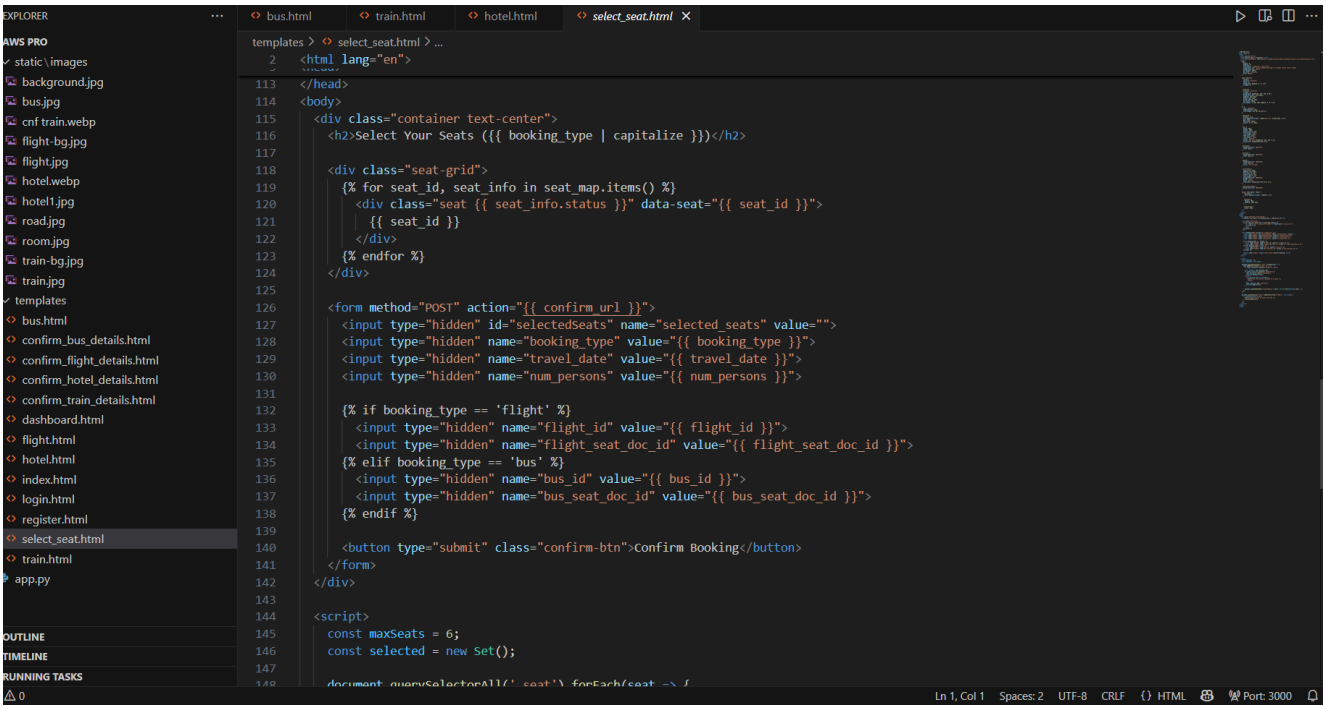
Bus Booking Code :



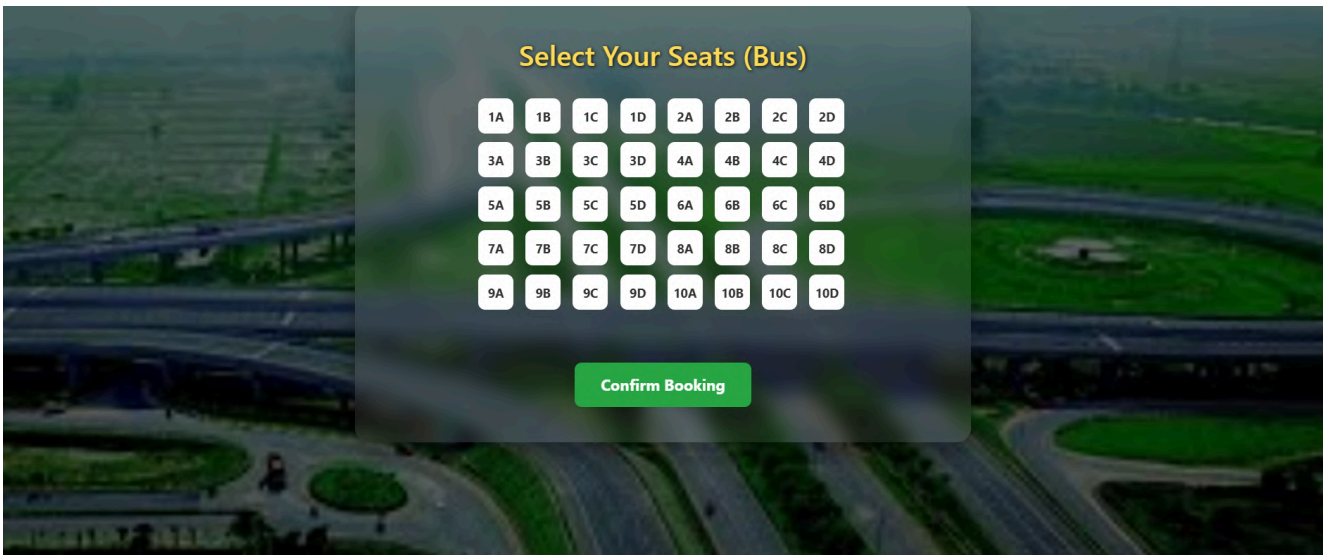
Bus Booking Page :



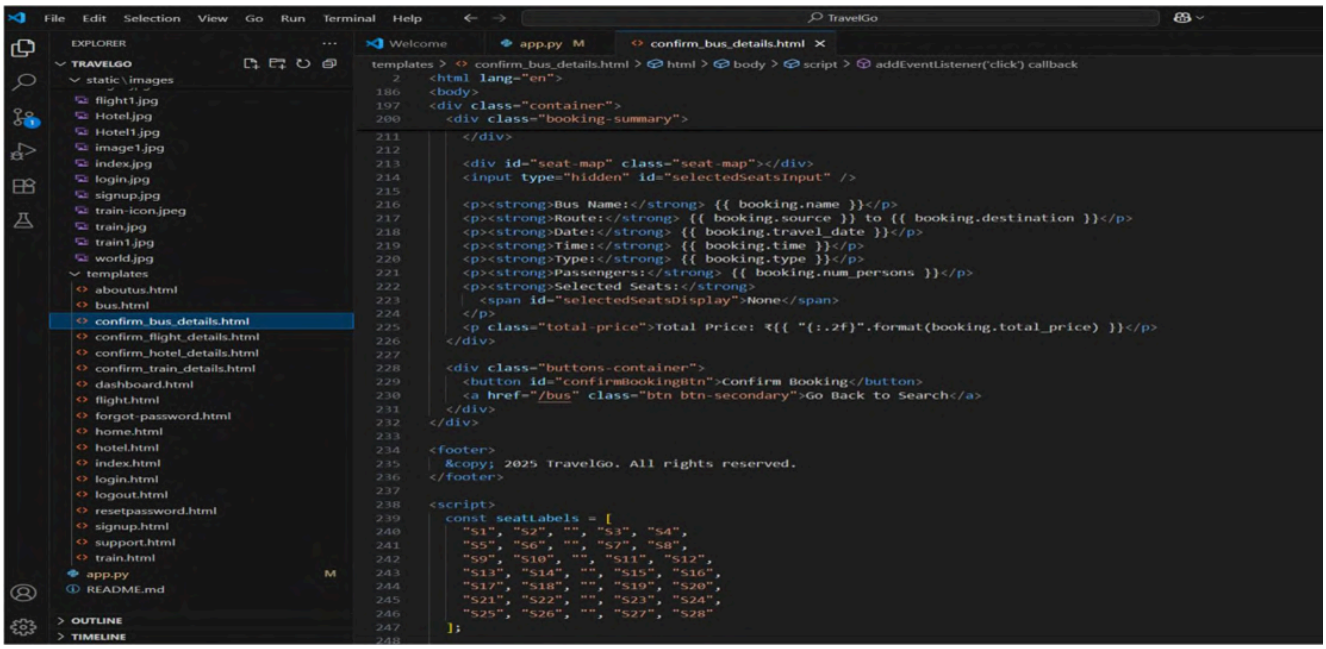
Selecting Seat Code :



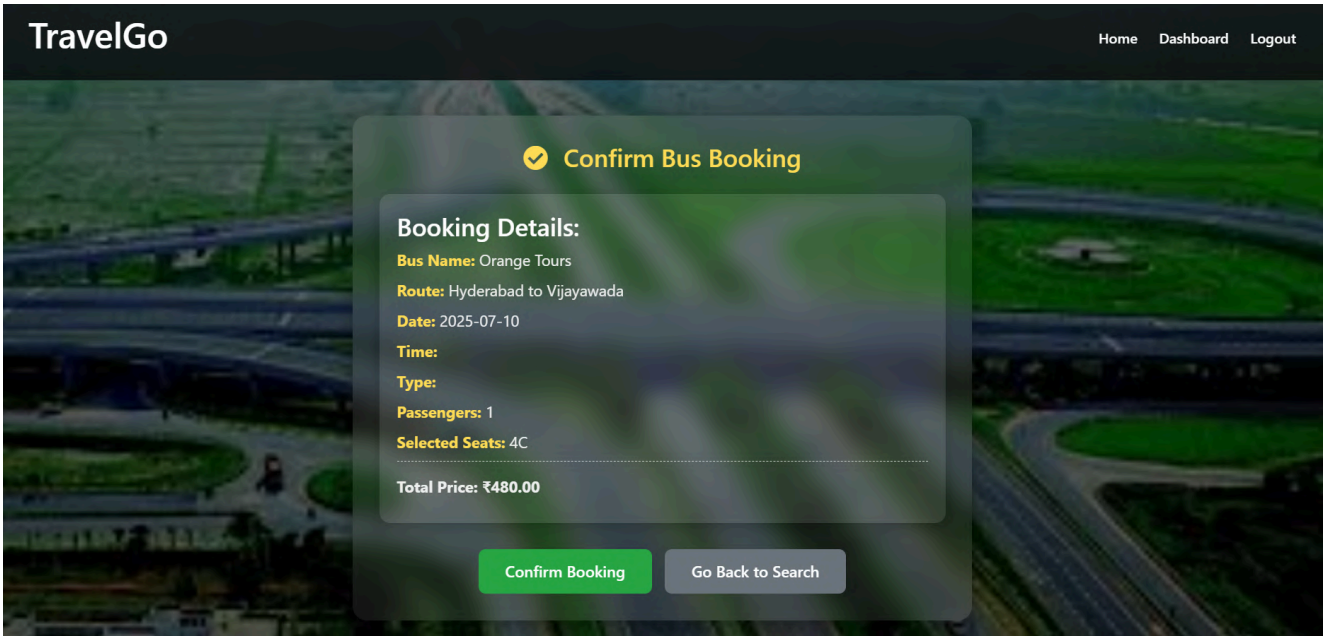
Selecting Seat Page :



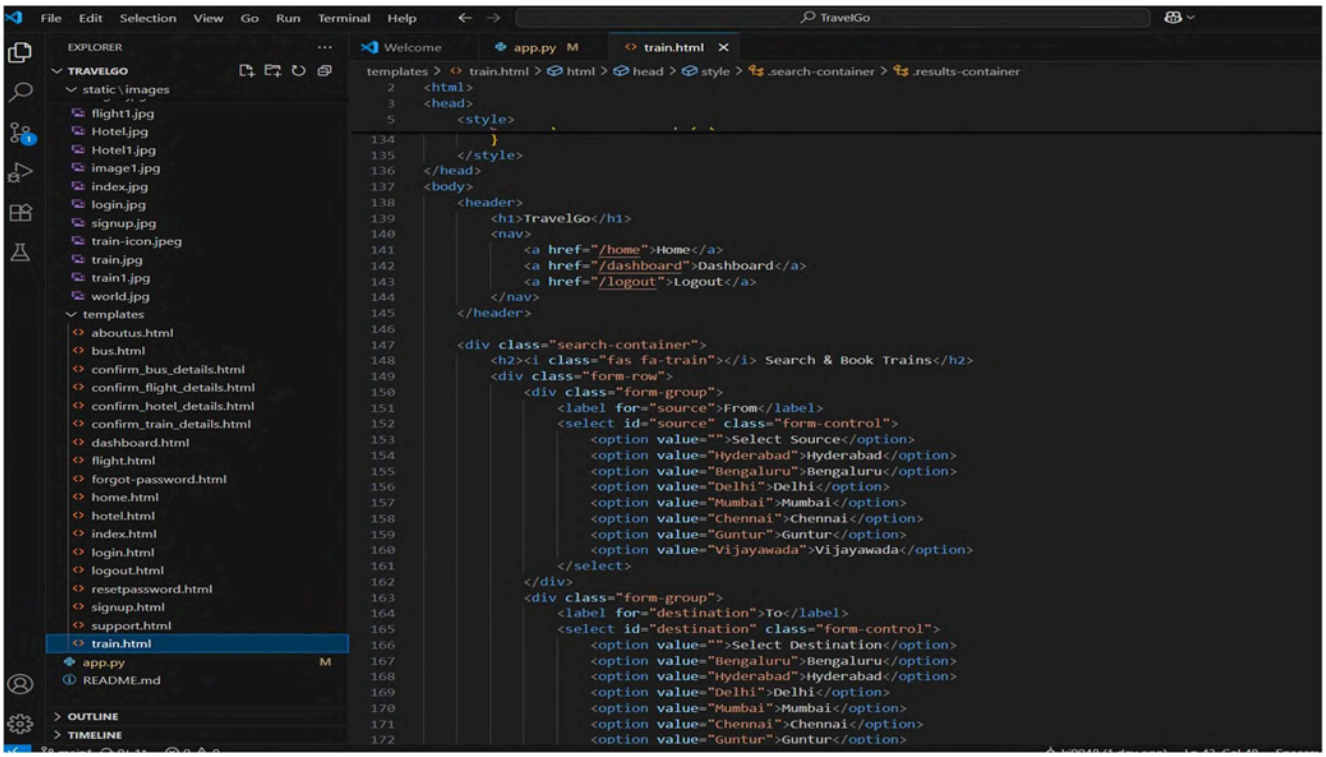
Bus Confirm Booking Code :



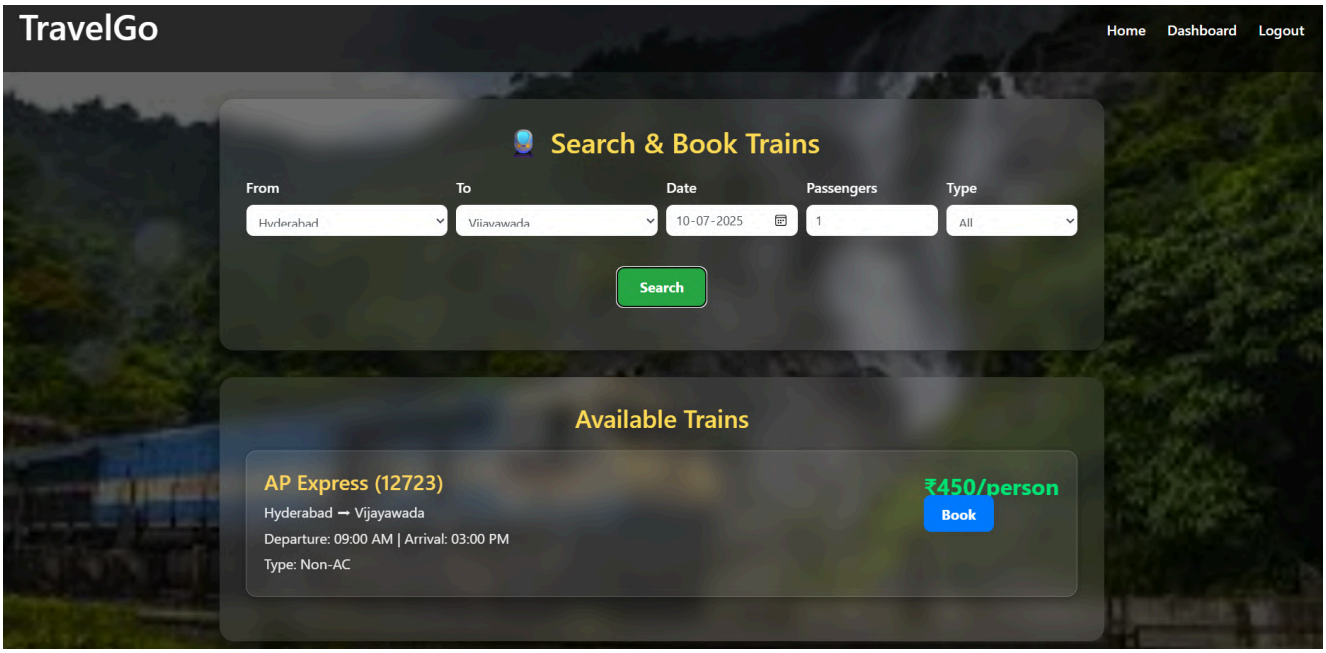
Bus Confirm Booking Page :



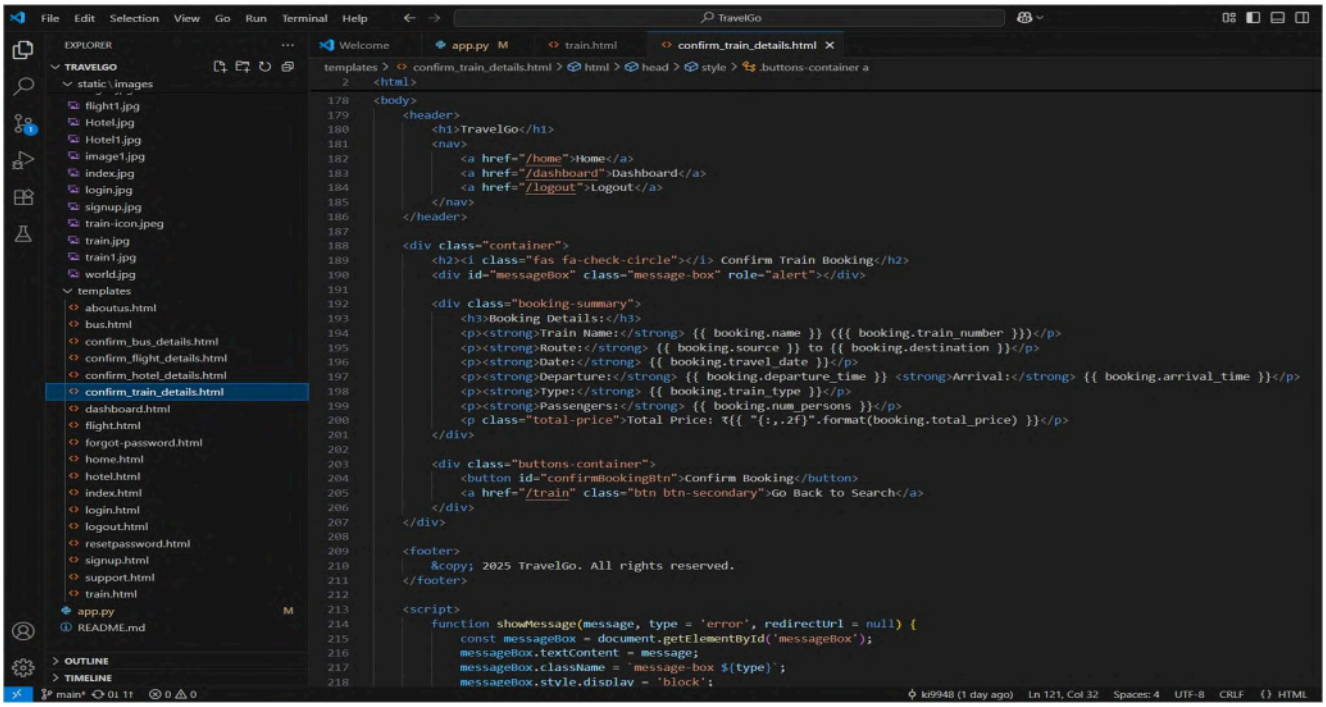
Train Booking Code :



Train Booking Page :

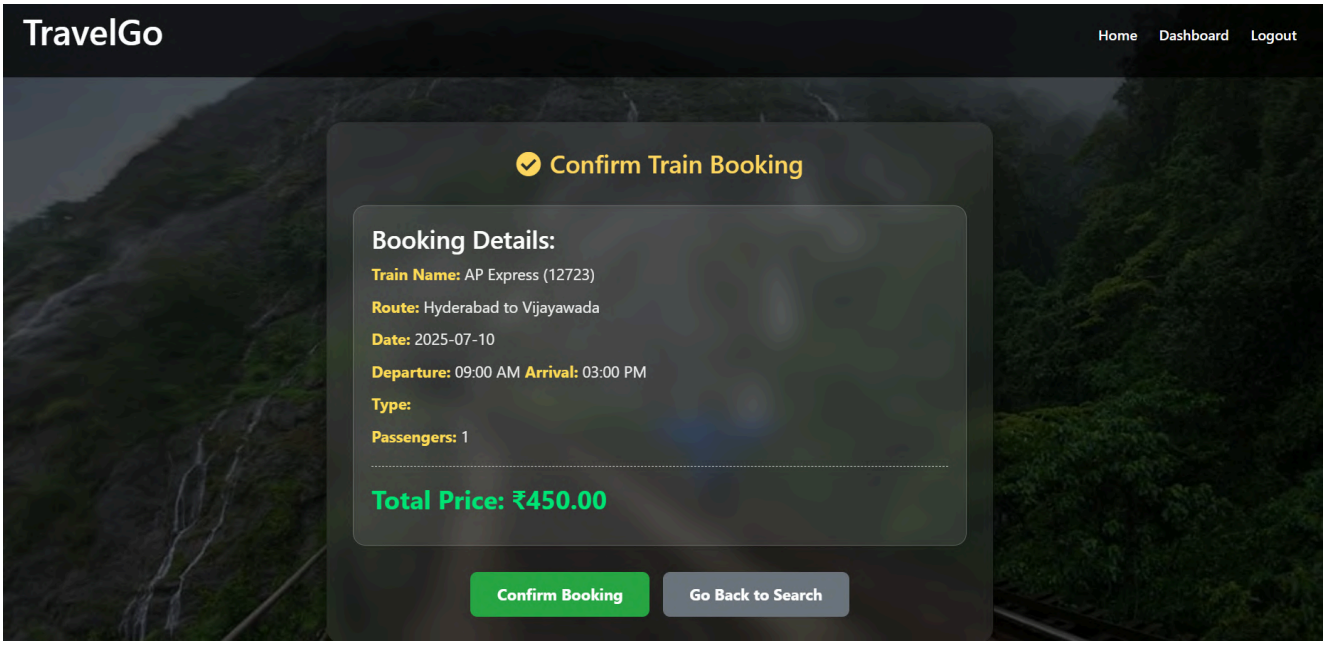


Train Confirm Booking Code :

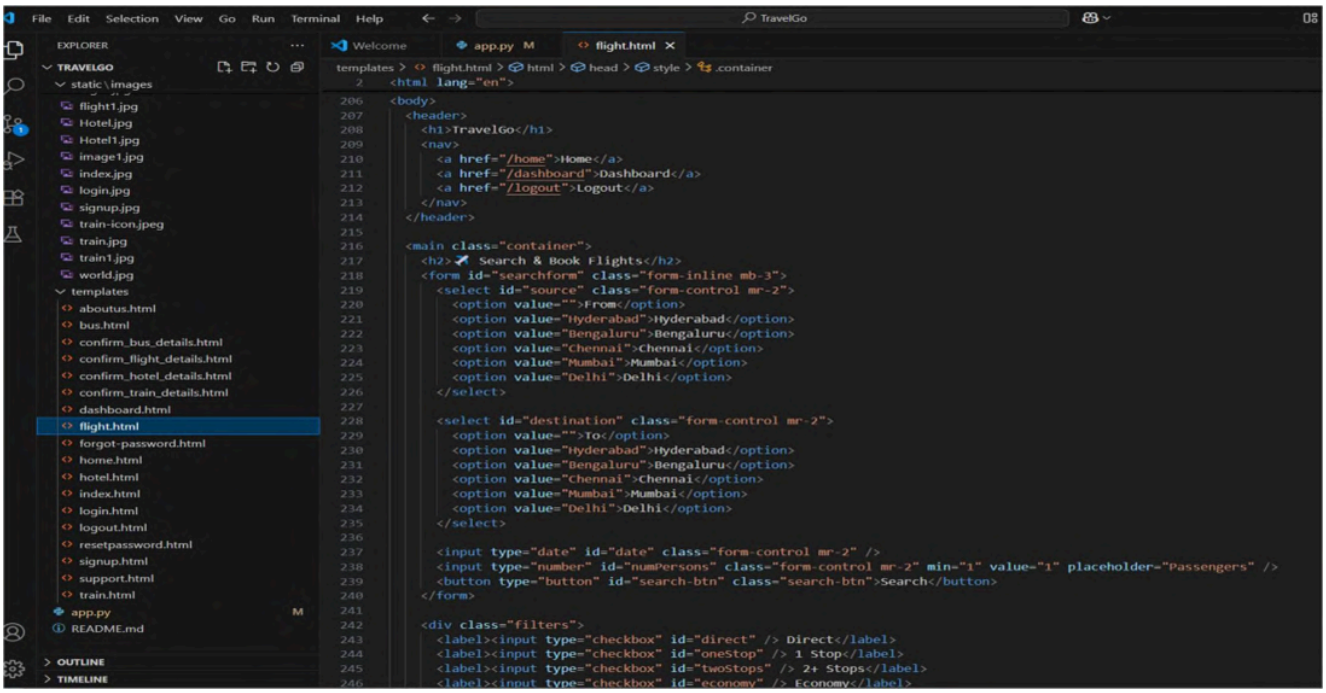


```
<body>
    <header>
        <h1>TravelGo</h1>
        <nav>
            <a href="/home">Home</a>
            <a href="/dashboard">Dashboard</a>
            <a href="/logout">Logout</a>
        </nav>
    </header>

    <div class="container">
        <h2><i class="fas fa-check-circle"></i> Confirm Train Booking</h2>
        <div id="messageBox" class="message-box" role="alert"></div>

        <div class="booking-summary">
            <h3>Booking Details:</h3>
            <p><strong>Train Name:</strong> {{ booking.name }} ({{ booking.train_number }})</p>
            <p><strong>Route:</strong> {{ booking.source }} to {{ booking.destination }}</p>
            <p><strong>Date:</strong> {{ booking.travel_date }}</p>
            <p><strong>Departure:</strong> {{ booking.departure_time }} <strong>Arrival:</strong> {{ booking.arrival_time }}</p>
            <p><strong>Type:</strong> {{ booking.train_type }}</p>
            <p><strong>Passengers:</strong> {{ booking.num_persons }}</p>
            <p class="total-price">Total Price: ₹{{ "{:,.2f}".format(booking.total_price) }}</p>
        </div>

        <div class="buttons-container">
            <button id="confirmBookingBtn">Confirm Booking</button>
            <a href="/train" class="btn btn-secondary">Go Back to Search</a>
        </div>
    </div>

    <footer>
        &copy; 2025 TravelGo. All rights reserved.
    </footer>

    <script>
        function showMessage(message, type = 'error', redirectUrl = null) {
            const messageBox = document.getElementById('messageBox');
            messageBox.textContent = message;
            messageBox.className = `message-box ${type}`;
            messageBox.style.display = 'block';
```

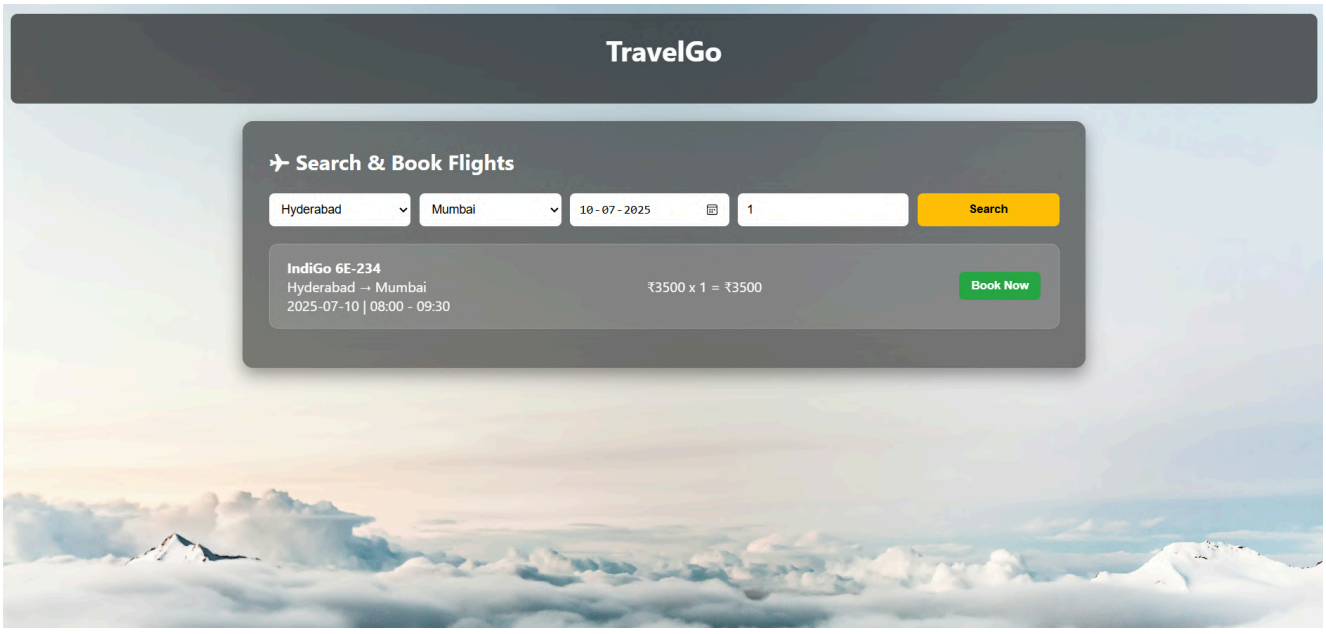
Train Confirm Booking Page :



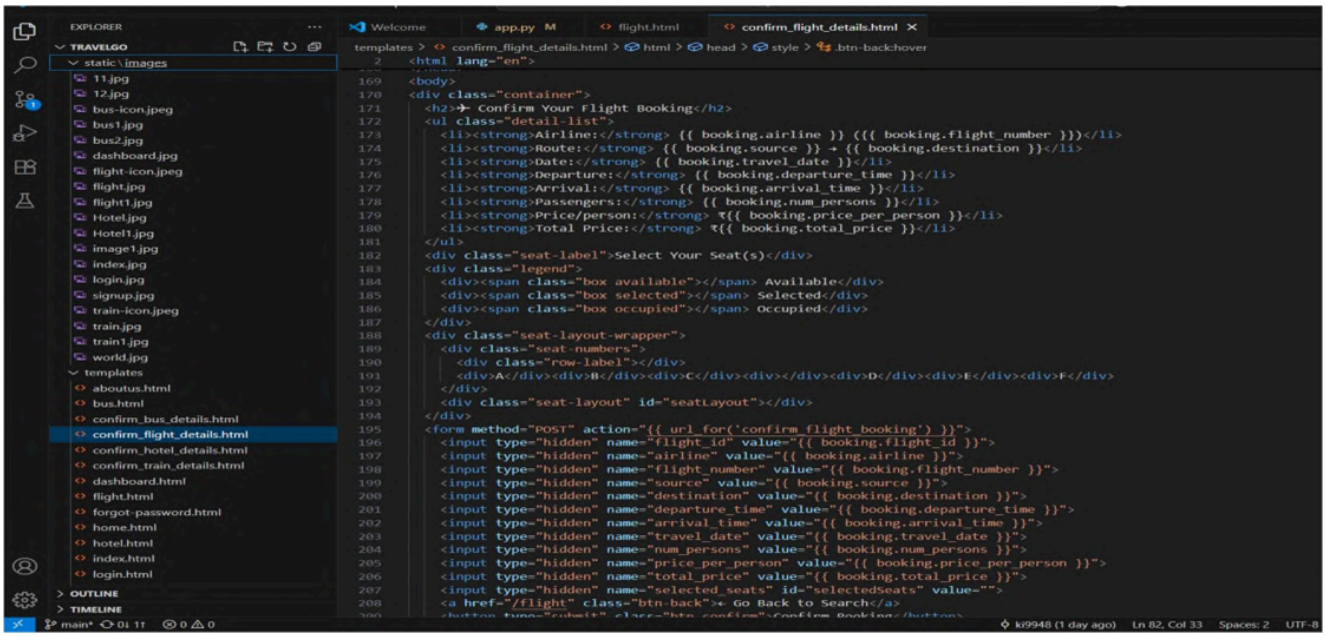
Flight Booking Code :



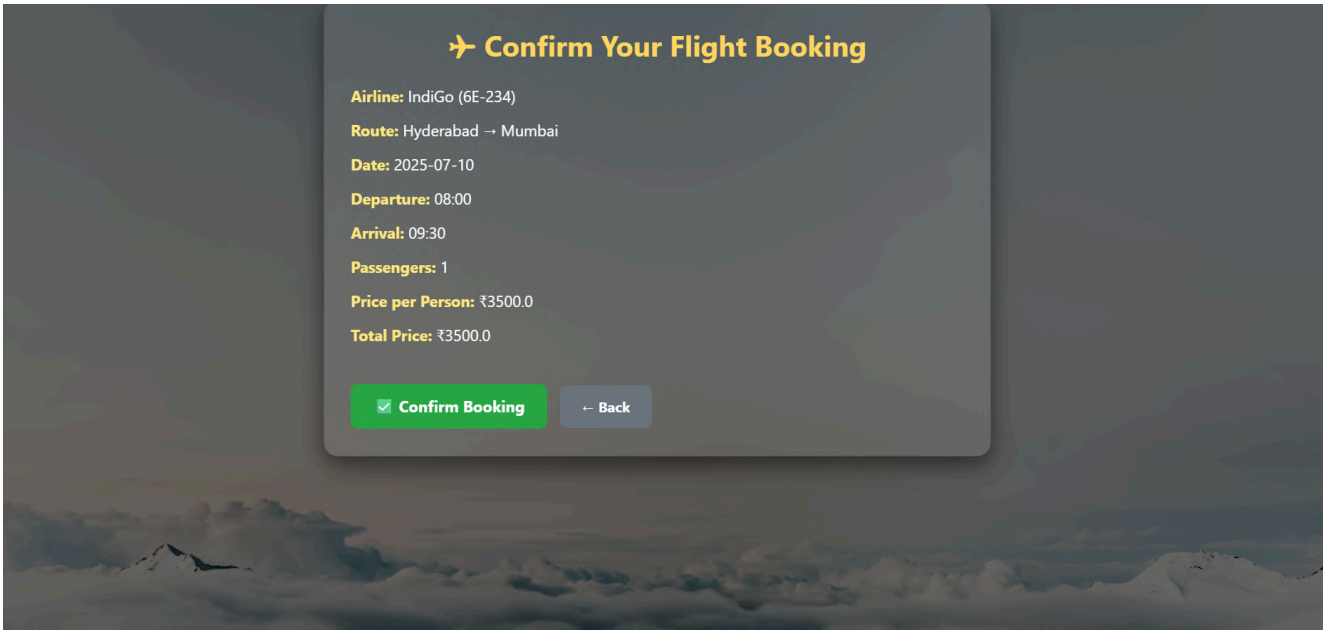
Flight Booking Page :



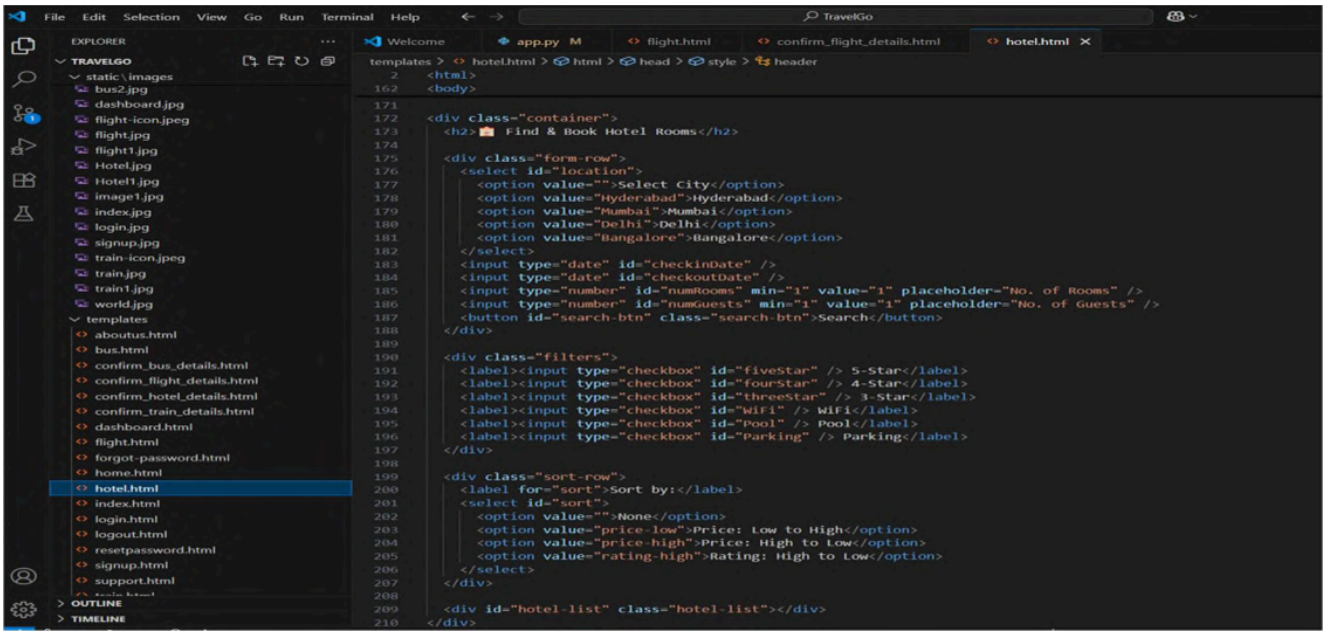
Flight Confirm Booking Code :



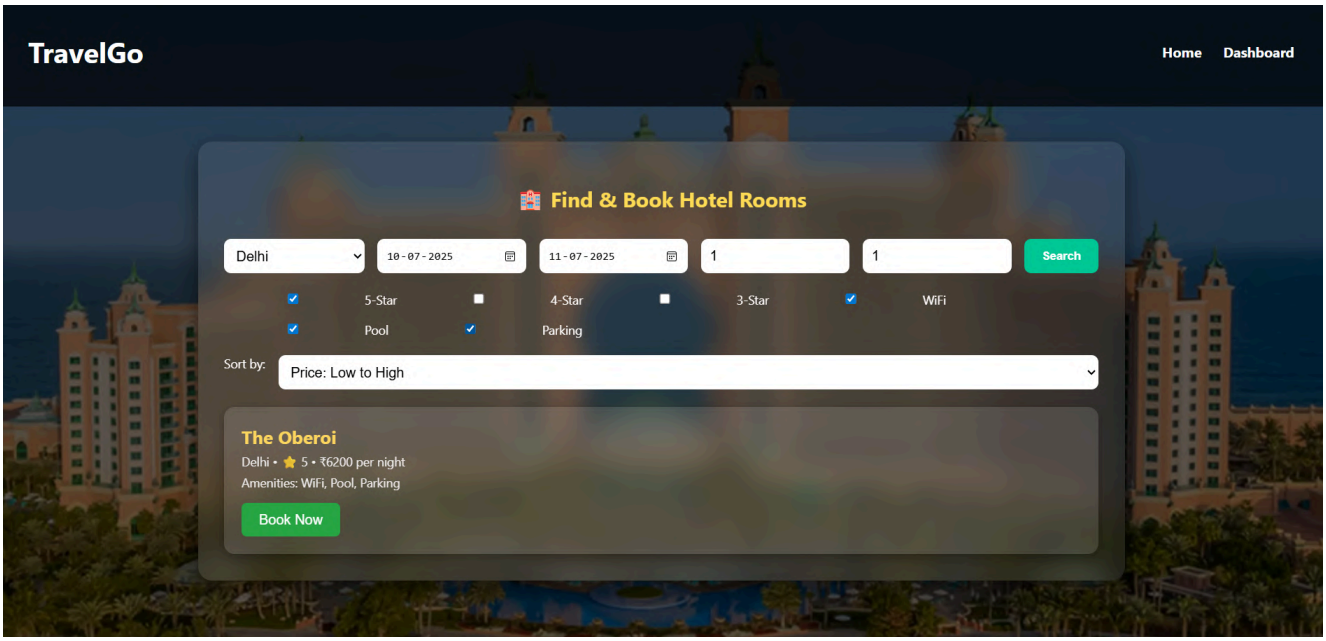
Flight Confirm Booking Page :



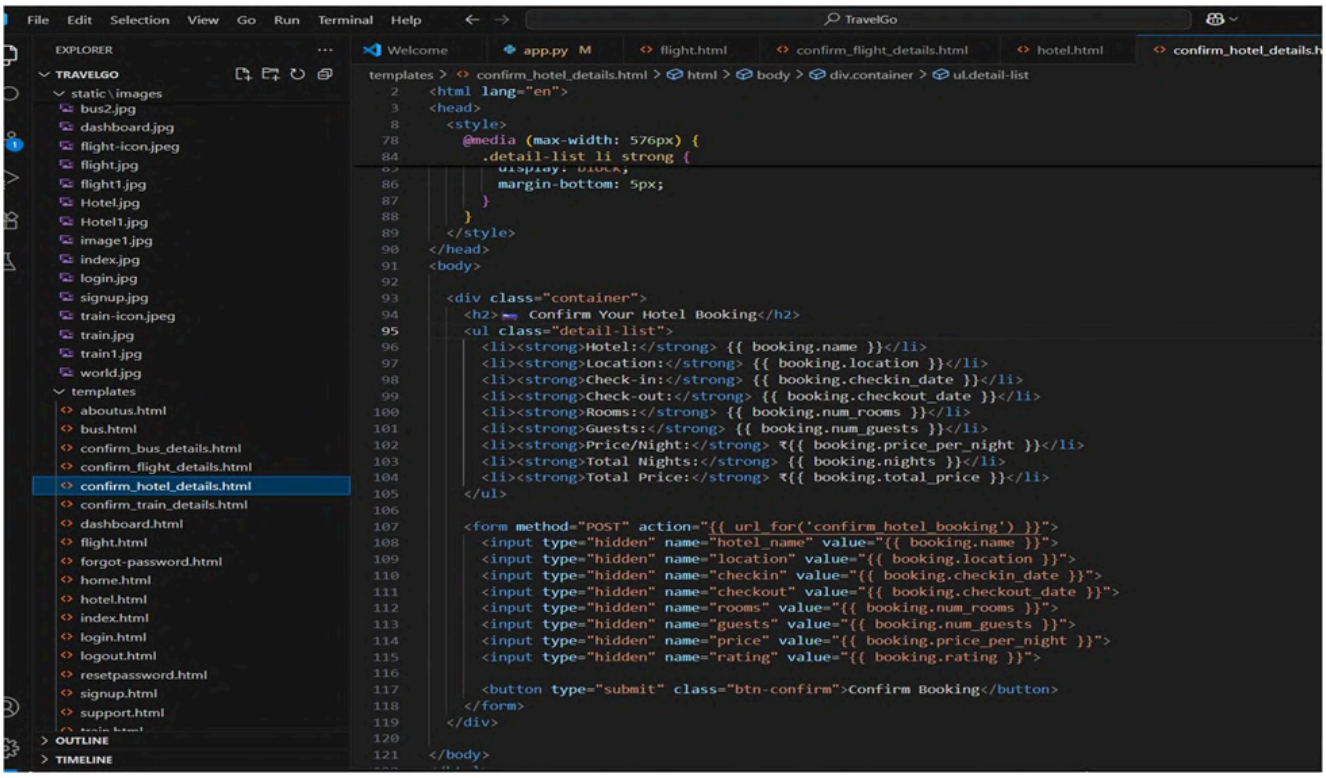
Hotel Page Code :



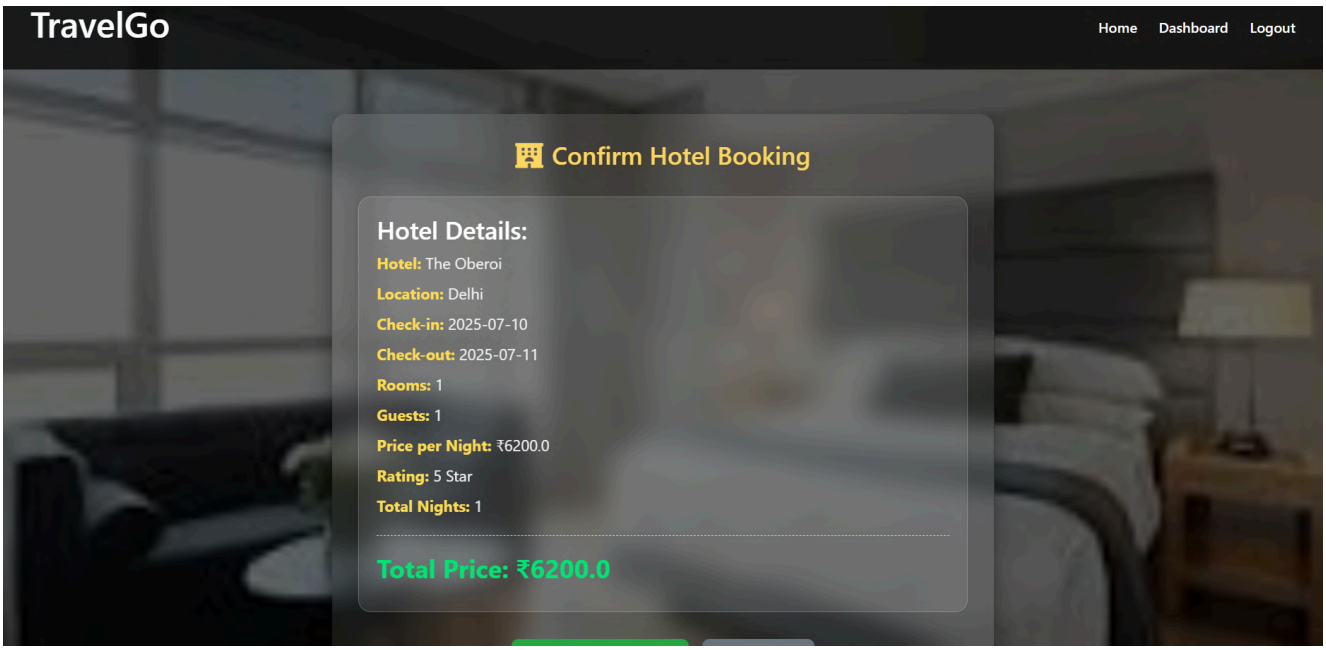
Hotel Page :



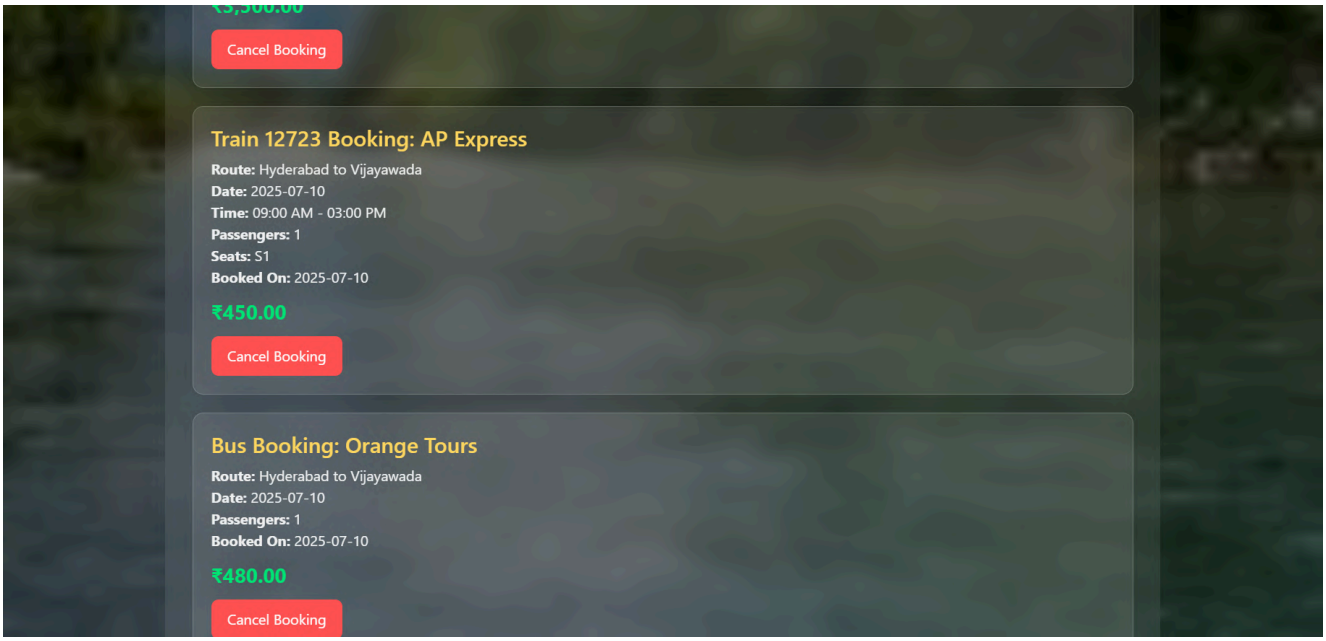
Hotel Confirming Booking Code :



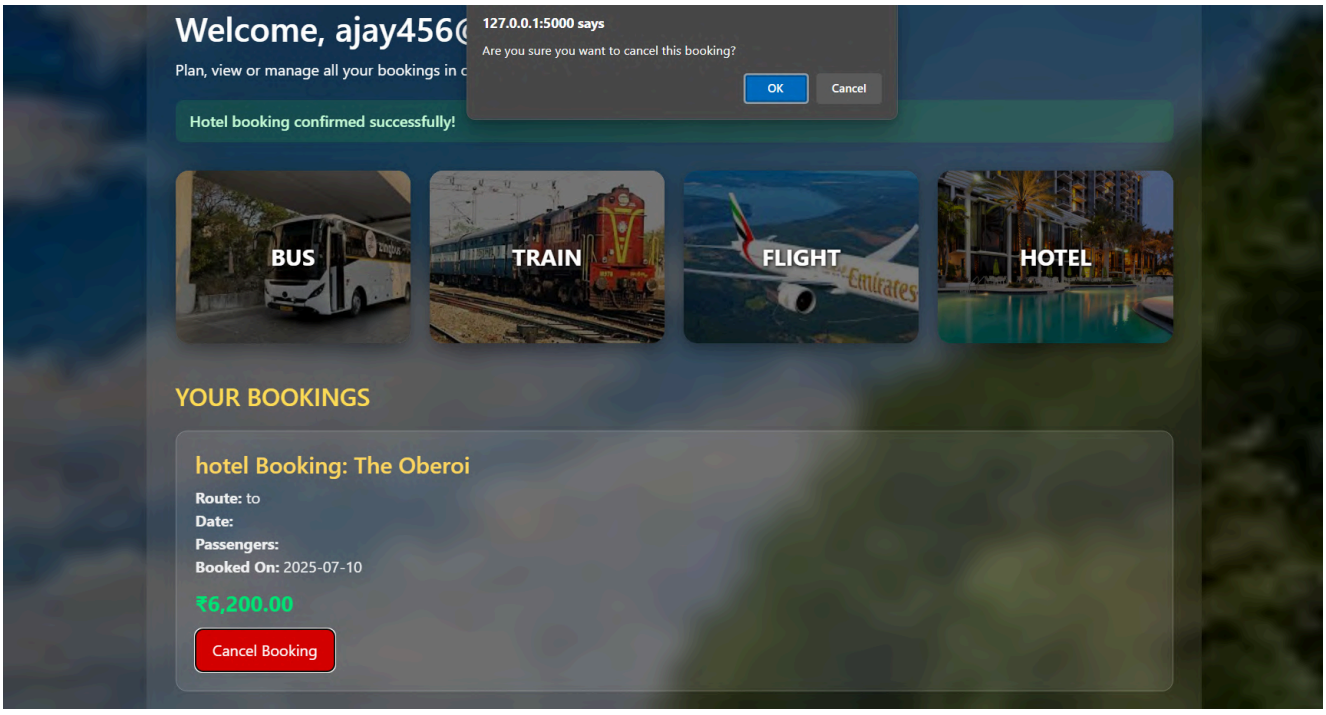
Hotel Confirming Booking Page :



Booking Pages :



Cancel Booking Page :



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.