Name : GINJUPALLI Kiran

Email : 228x1a05e1@khitguntur.ac.in

College : Kallam Haranadhareddy Institute of Technology

Branch : CSE

Project Name : TravelGo: A Cloud-Powered Real-Time Travel Booking Platform Using 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 Booking Confirmation with AWS SNS

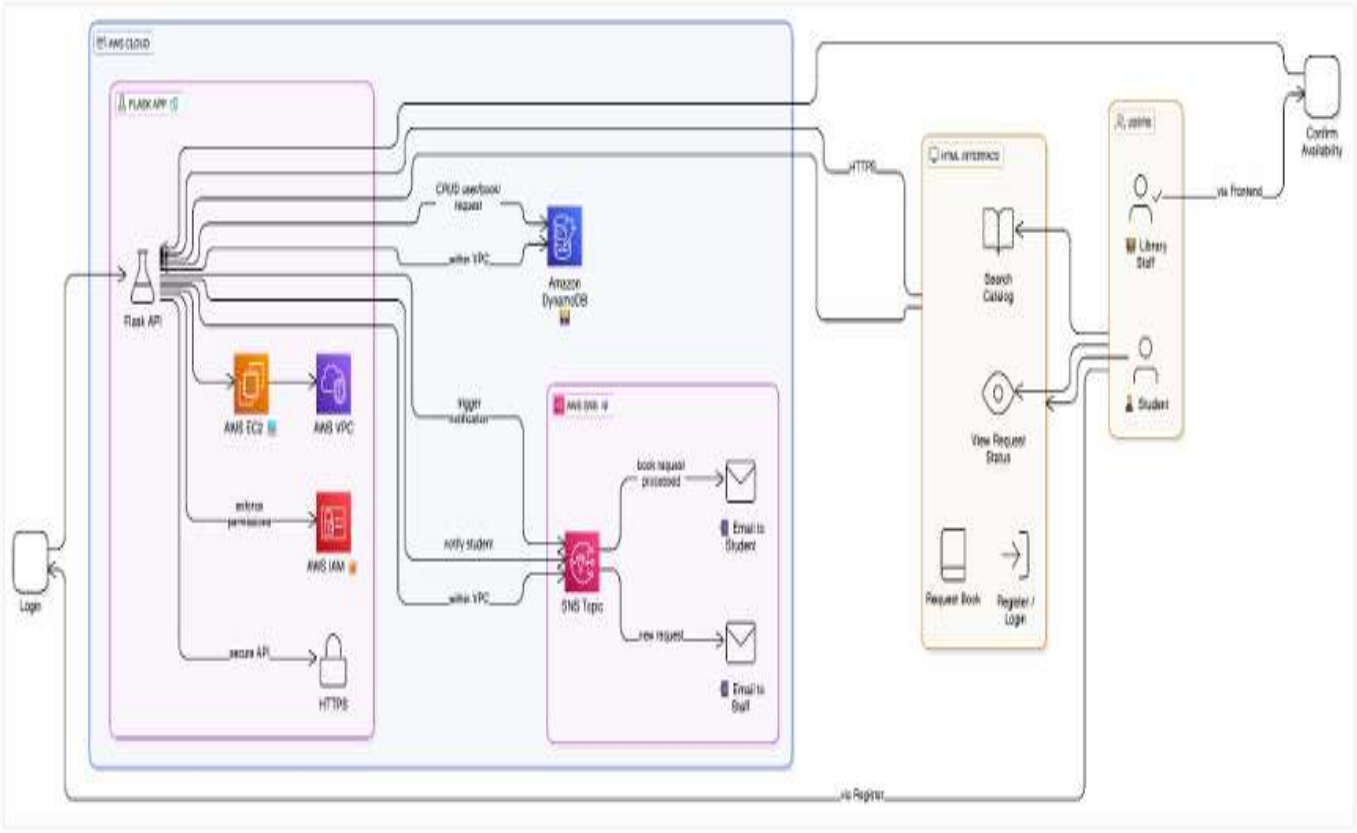
Once a booking is made—whether it's a train ticket or a hotel stay—TravelGo uses AWS SNS to instantly notify the user. For example, after a student books a hotel in Chennai, SNS sends a real-time email notification confirming the booking with all the relevant details. This notification is triggered from 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.

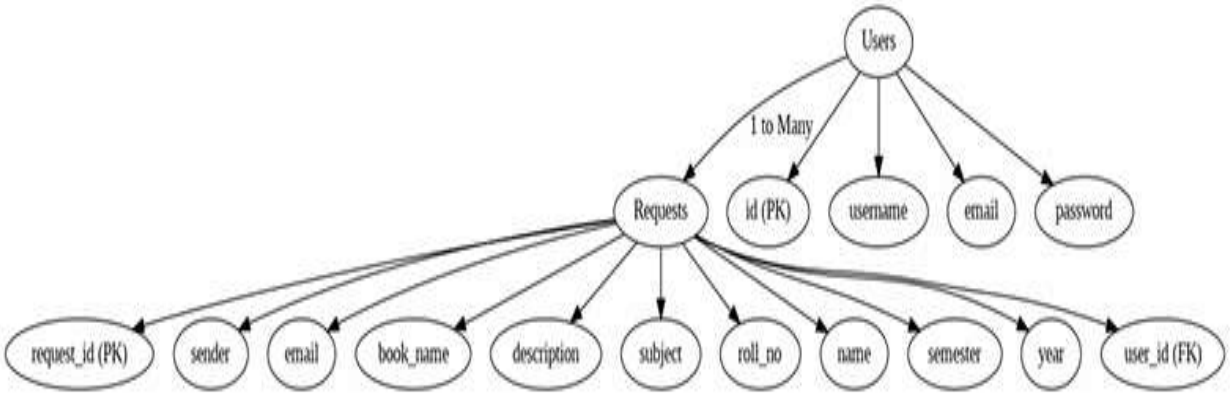
Scenario 3: Dynamic Dashboard with Personal Travel History

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.

AWS ARCHITECTURE



Entity Relationship (ER)Diagram:



Project WorkFlow:

1. AWS troven access Setup and Login
    1.1: Set up an troven access
    1.2: Open the AWS Management Console
2. DynamoDB Database Creation and Setup
    2.1: Create a DynamoDB Table.
    2.2: Configure Attributes for User Data and Booking Requests.
3. SNS Notification Setup Activity
    3.1: Create SNS topics.
    3.2: Subscribe users and library staff to SNS email notifications.
4. Backend Development and Application Setup
    4.1:Develop the Backend Using Flask.
    4.2: Integrate AWS Services Using boto3.
5. IAM Role Setup
    5.1: Create IAM Role
    5.2: Attach Policies
6. EC2 Instance Setup Activity
    6.1: Launch an EC2 instance to host the Flask application.
    6.2: Configure security groups for HTTP, and SSH access.

7. Deployment on EC2 Activity

7.1:Upload Flask Files Activity

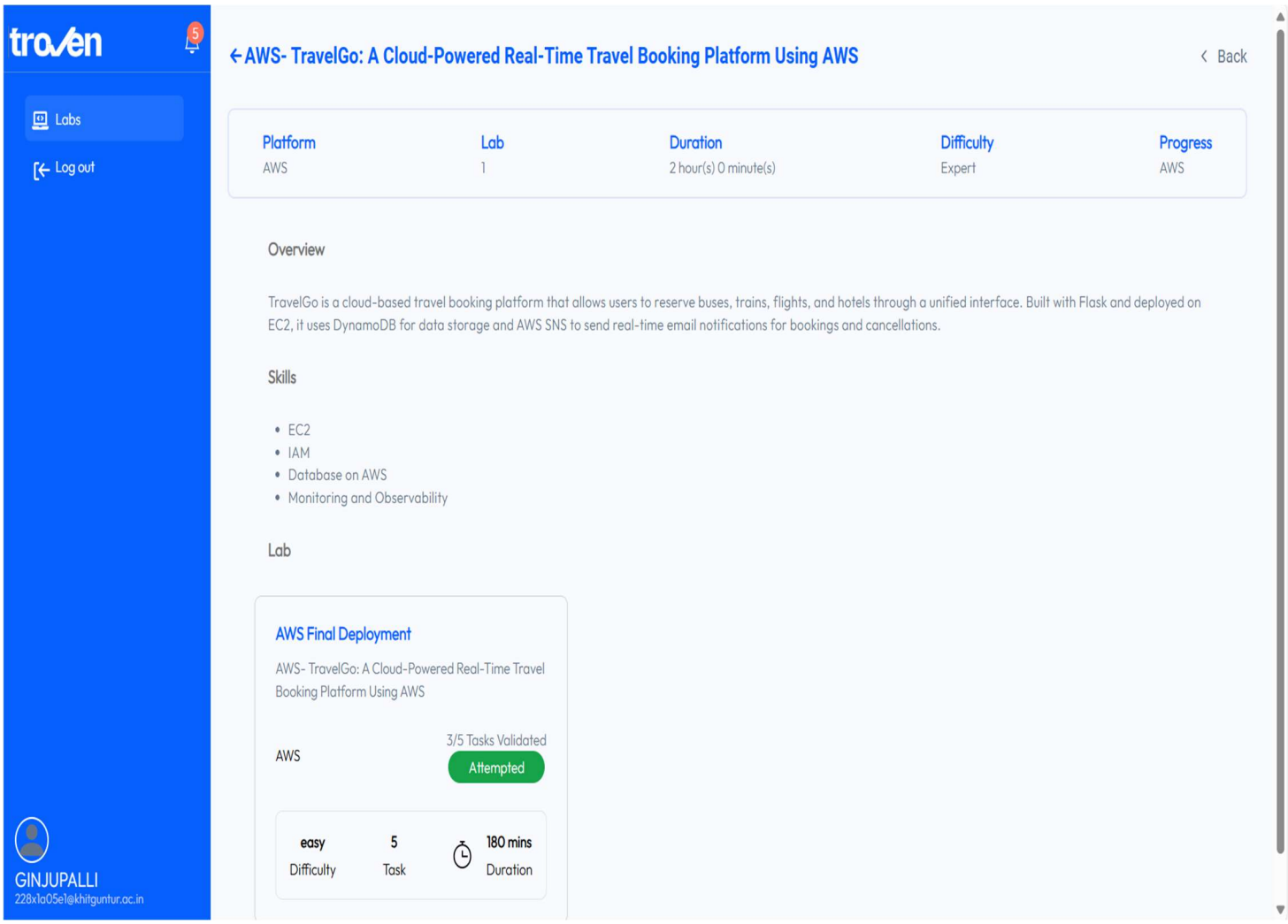
7.2: Run the Flask App

8. Testing and Deployment Activity

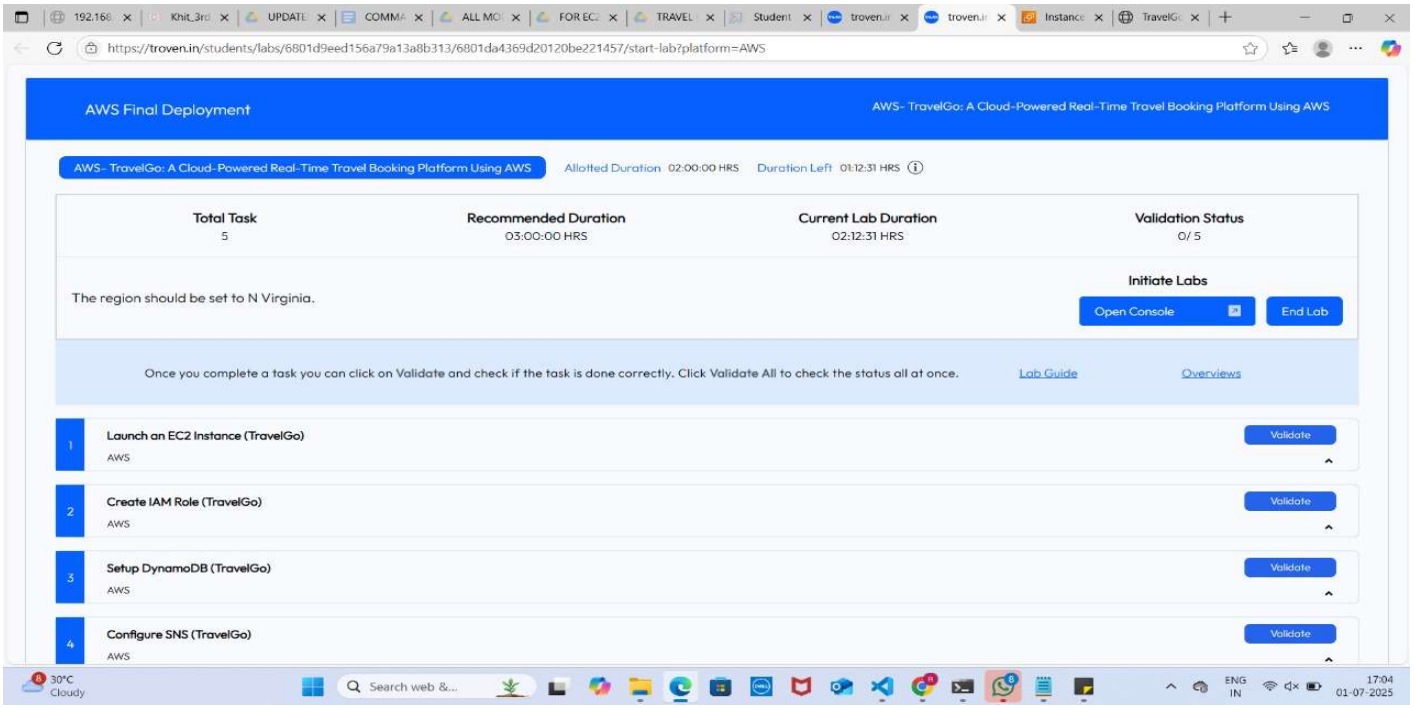
8.1: Conduct functional testing to verify user registration, login, bookings, and notifications.

1. AWS troven access Setup and Login

- Activity 1.1: Set up an troven access



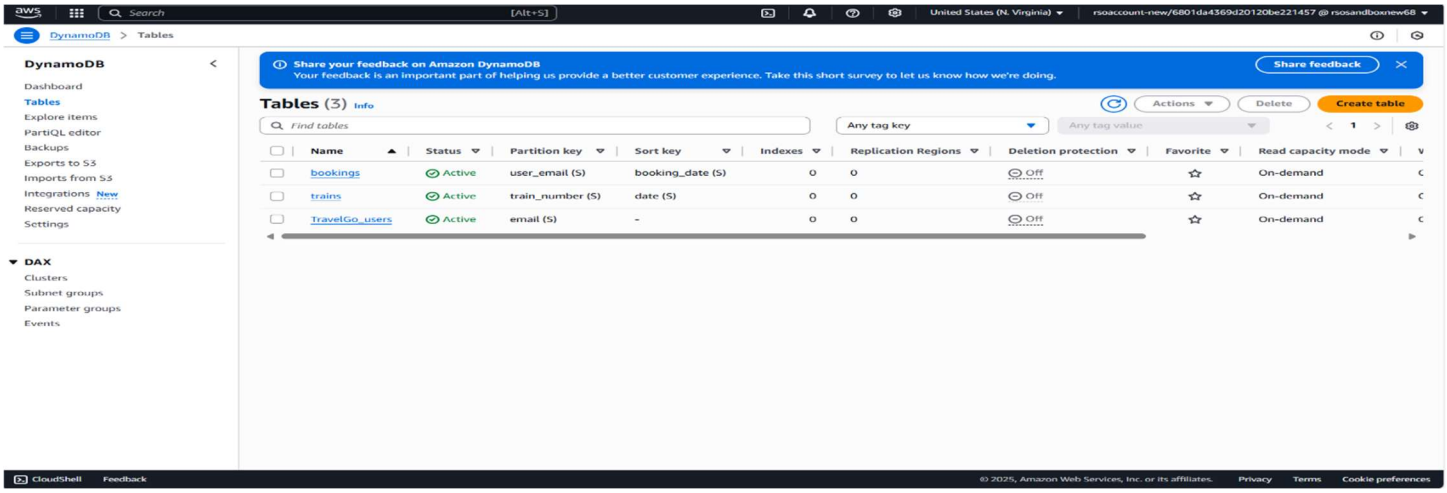
- Activity 1.2: Open the AWS Management Console



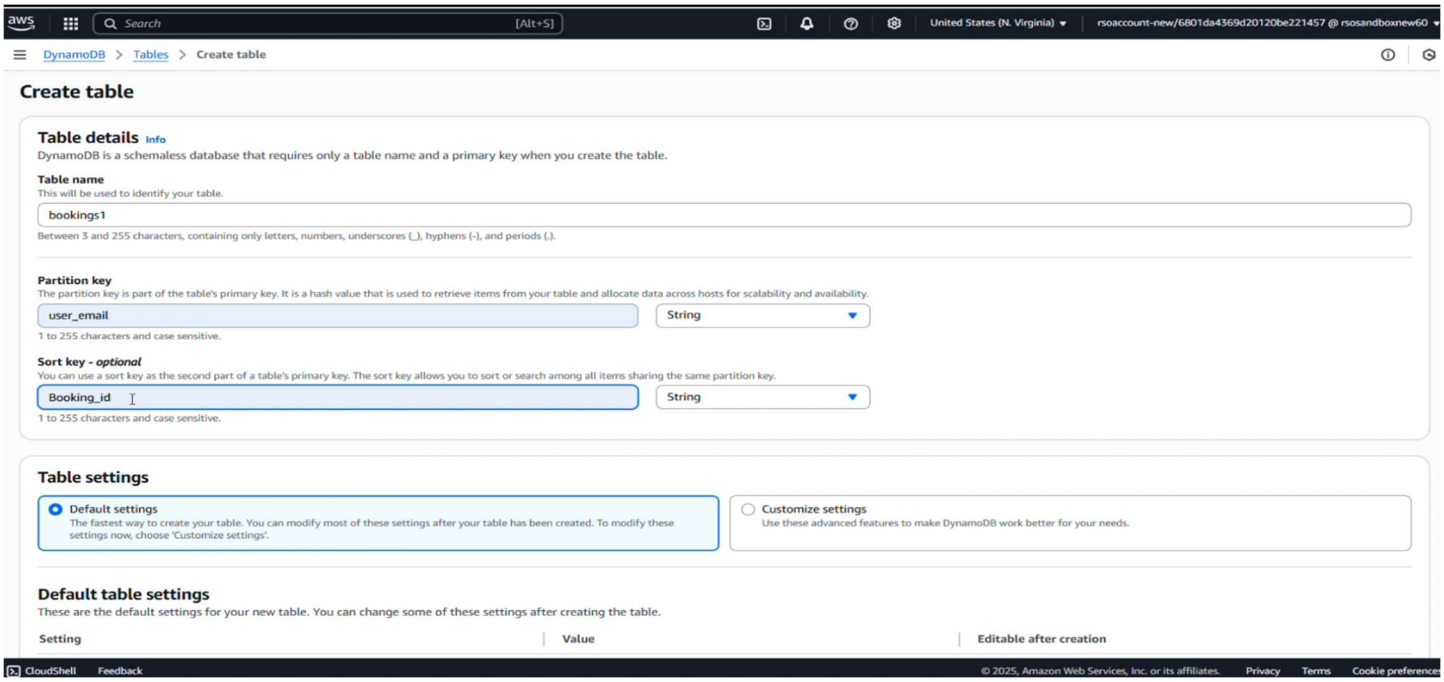
2. DynamoDB Database Creation and Setup

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**.**



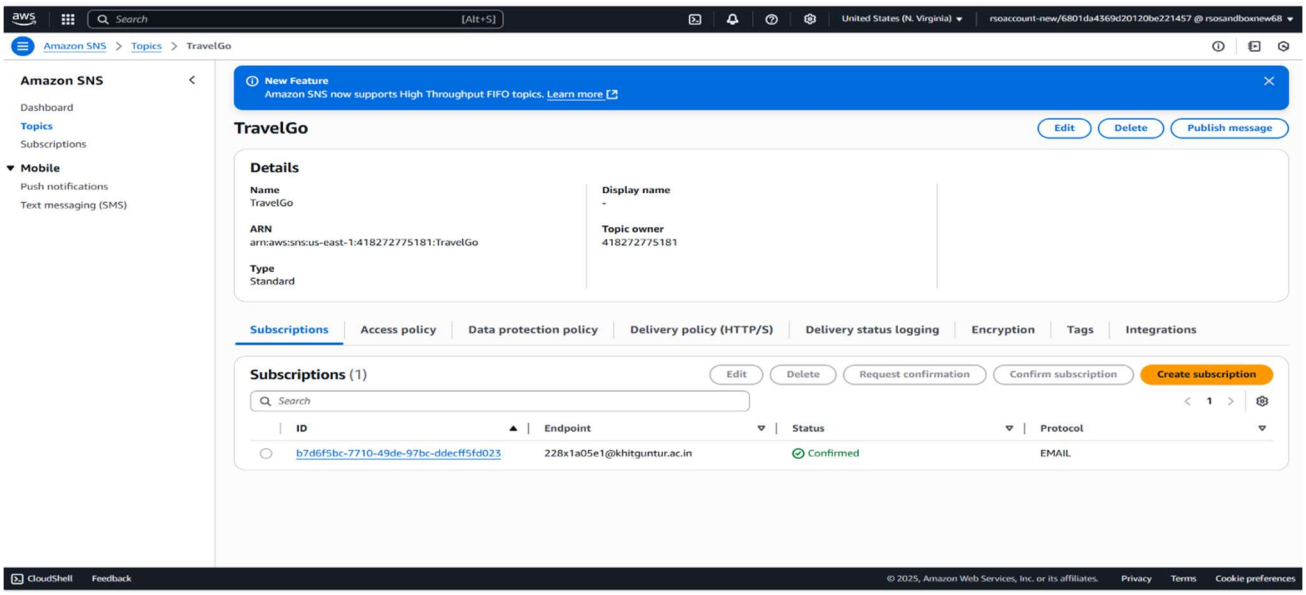
Activity 2.2: Configure Attributes for User Data and Booking Requests



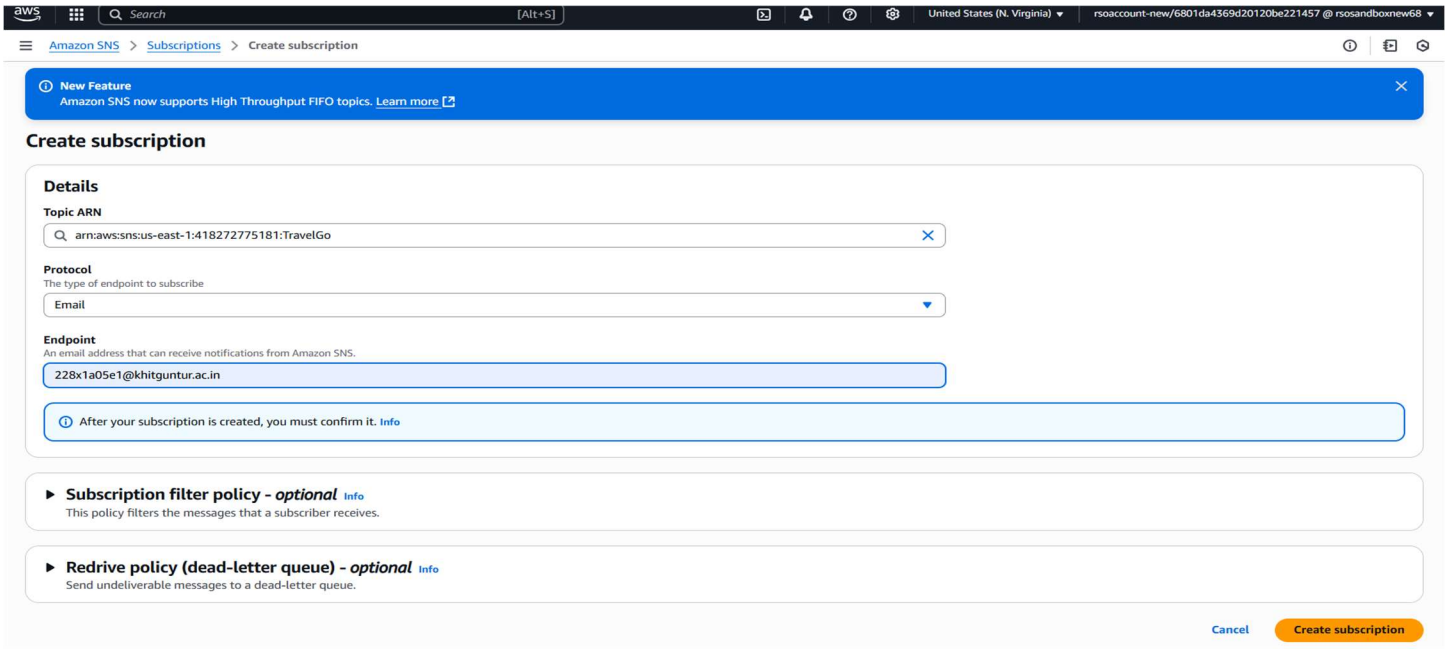
3. SNS Notification Setup

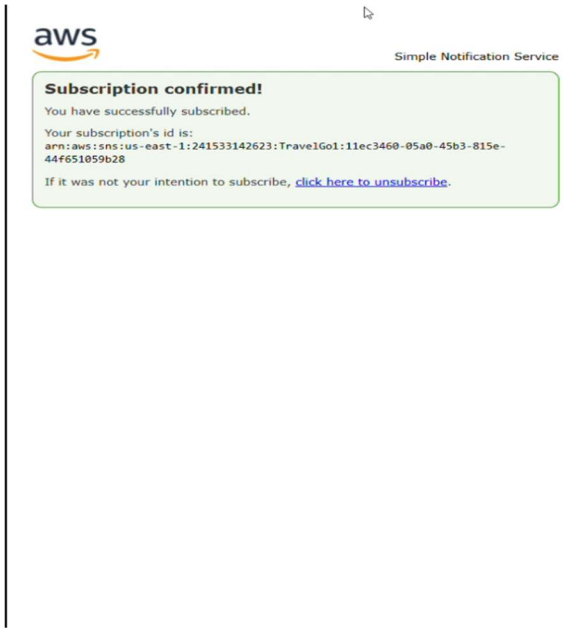
For **SNS Notification Setup**, go to the AWS Console and open the **Simple Notification Service (SNS)** dashboard. Click **"Create topic"**, choose the **Standard** type, and name your topic. After creating it, copy the **Topic ARN**. Then, **subscribe email addresses** (students/admins) to the topic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

Activity 3.1: Create SNS topics.



Activity 3.2: Subscribe users to SNS email notifications

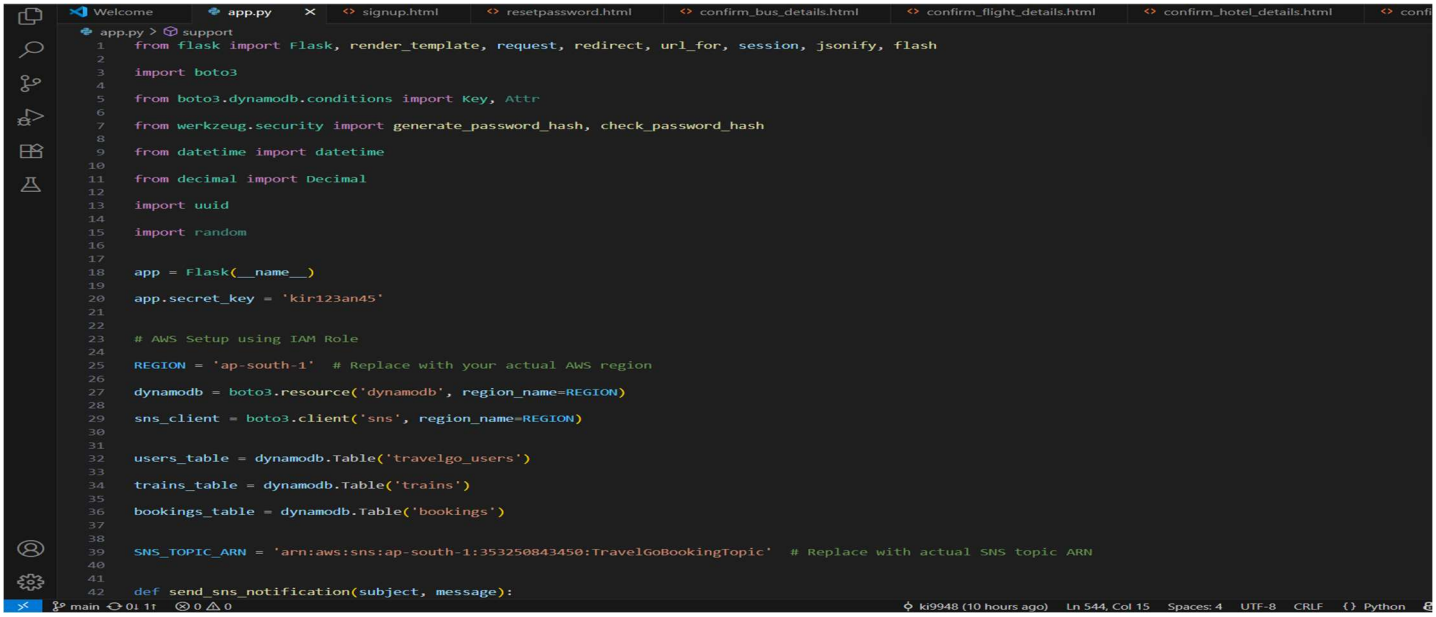




4. Backend Development and Application Setup

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 Using Flask.



Activity 4.2: Integrate AWS Services Using boto3

app.py > ...

```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
```

Code 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

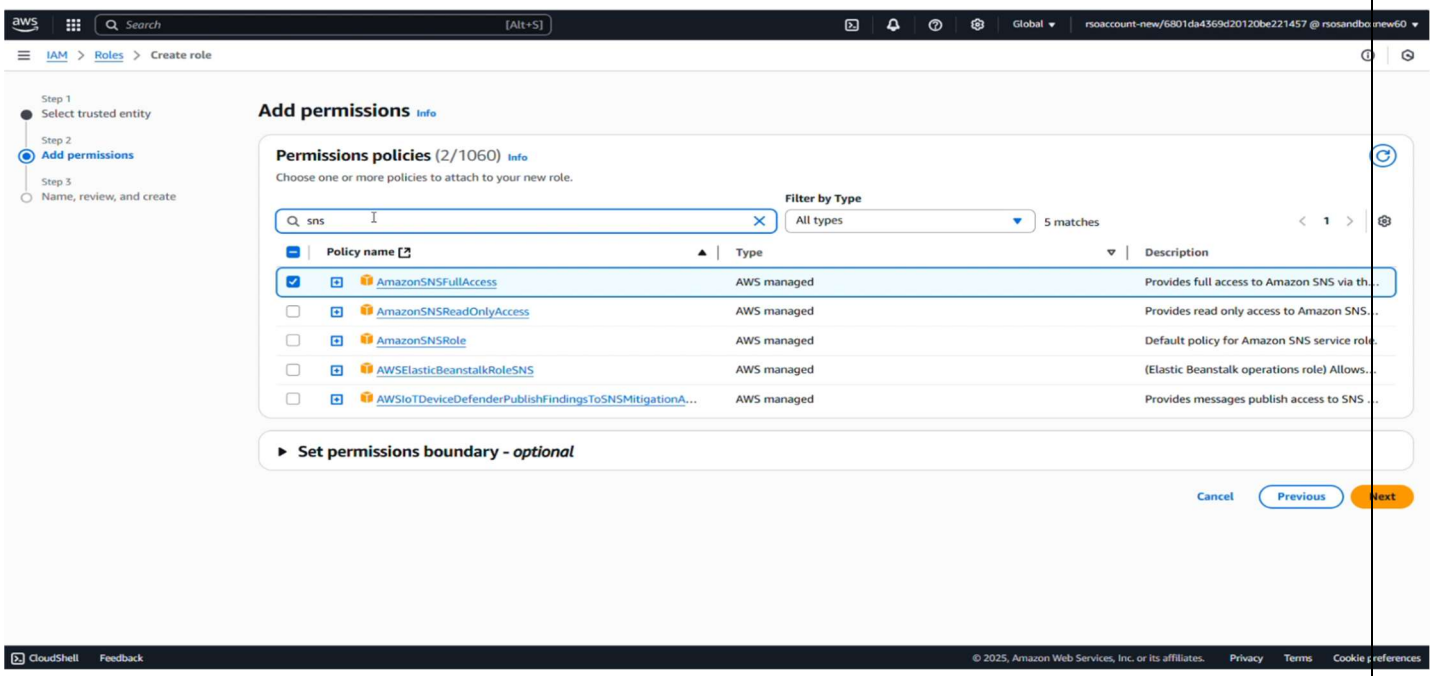
```python

@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')

```

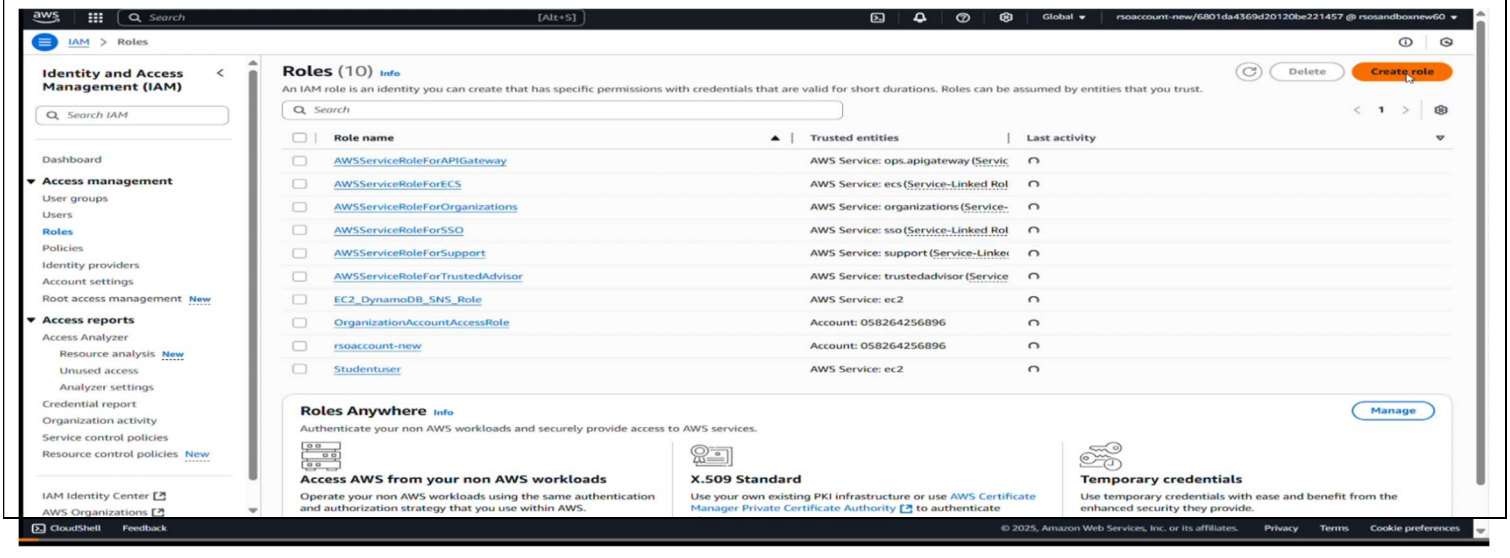
5. IAM Role Setup

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: Create IAM Role

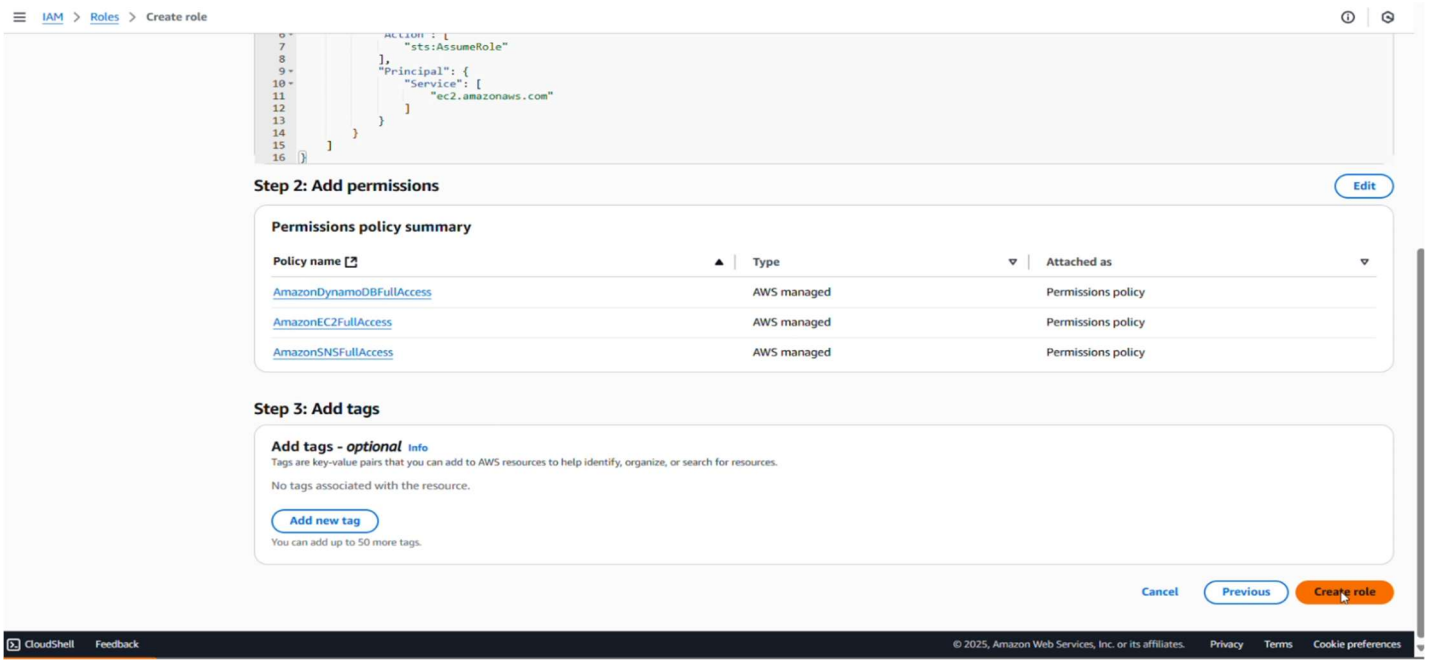
Activity 5.2: Attach Policies

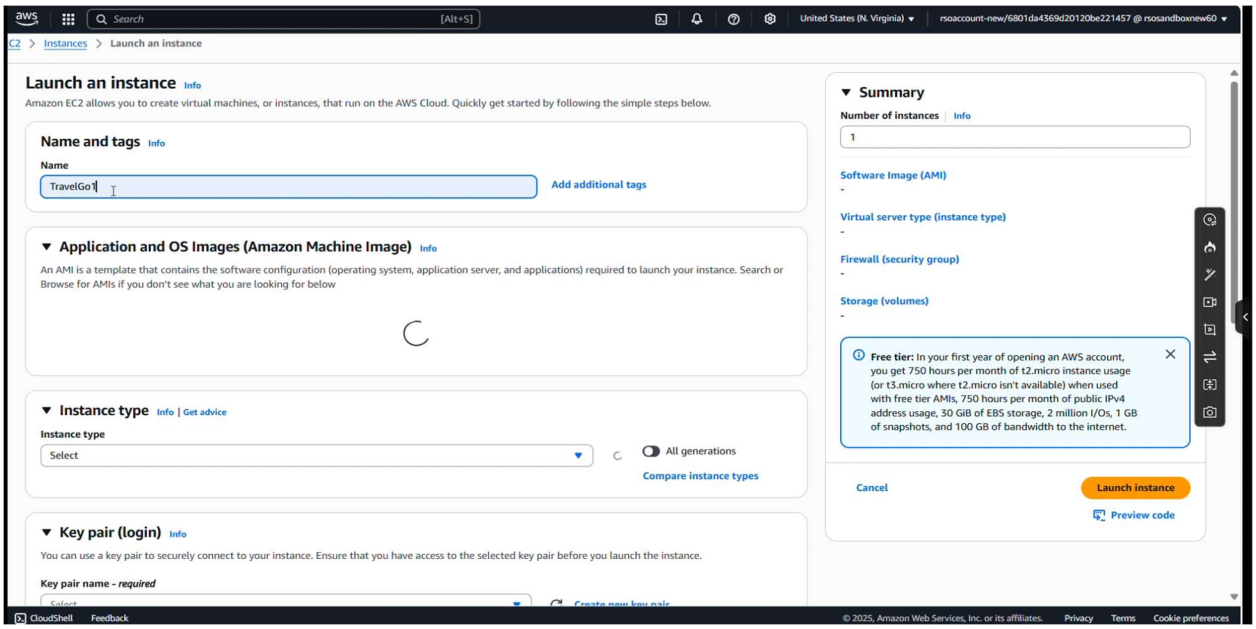




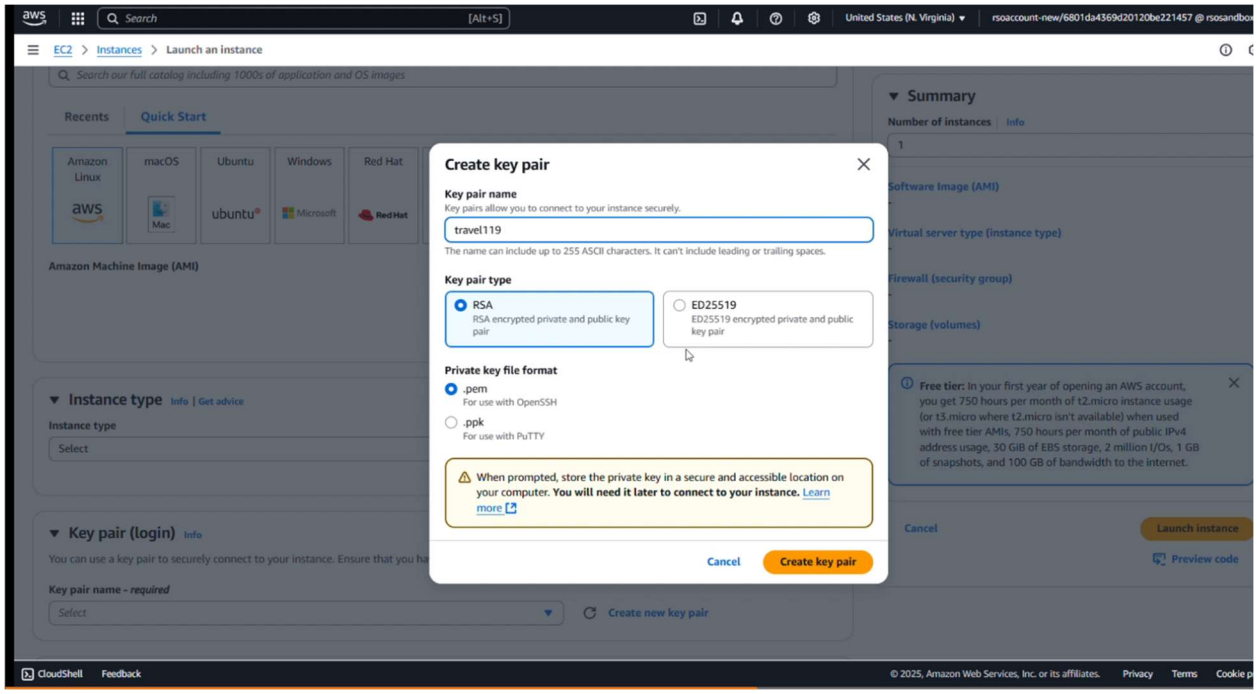
6. EC2 Instance Setup Activity

In the **EC2 Instance Setup**, launch a new EC2 instance using **Amazon Linux 2** or **Ubuntu**, and choose the **t2.micro** type for free-tier eligibility. Configure a **key pair** for secure SSH access and set **security groups** to allow HTTP (port 80) and SSH (port 22). Attach the appropriate **IAM role** to the instance. This setup prepares the server environment to host your Flask application.

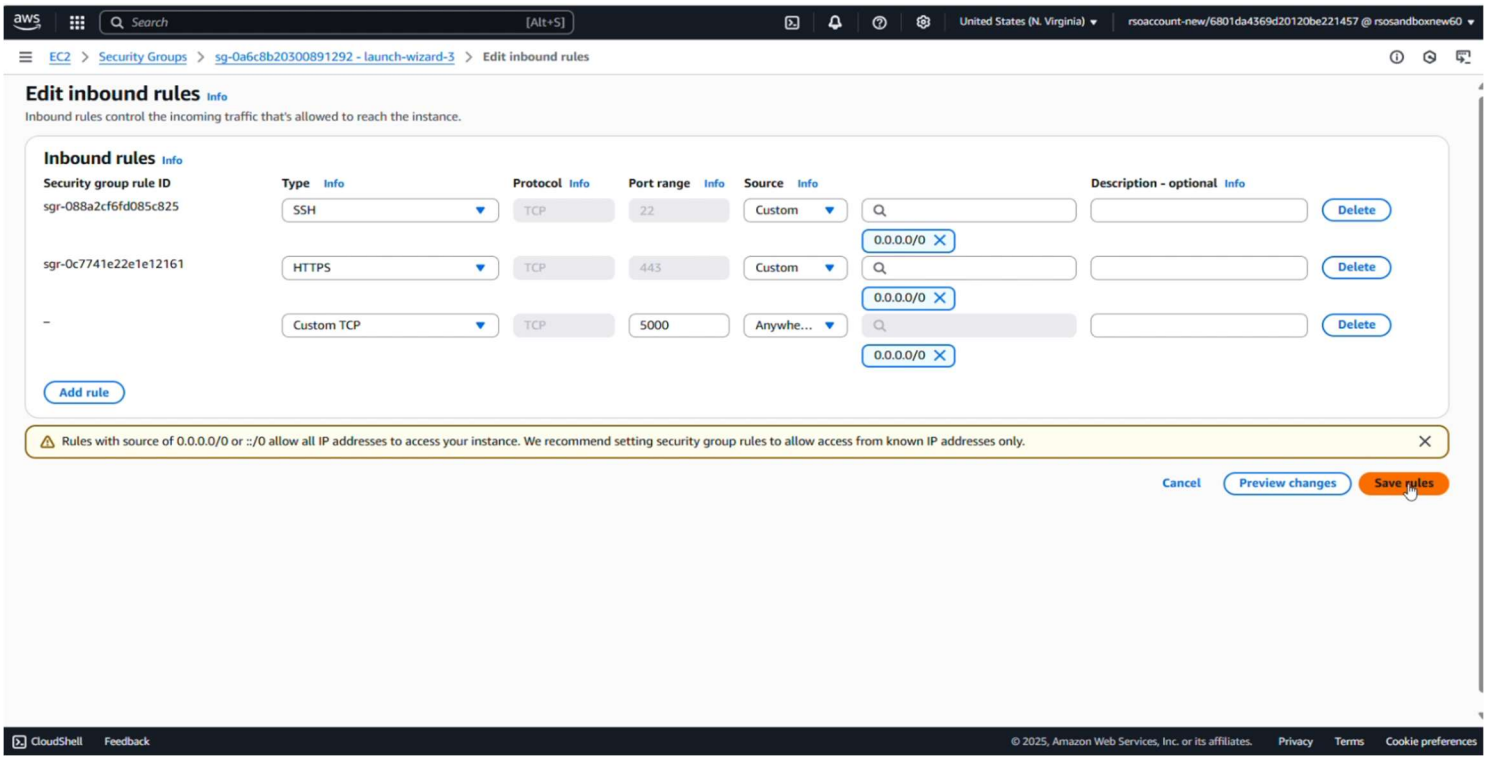


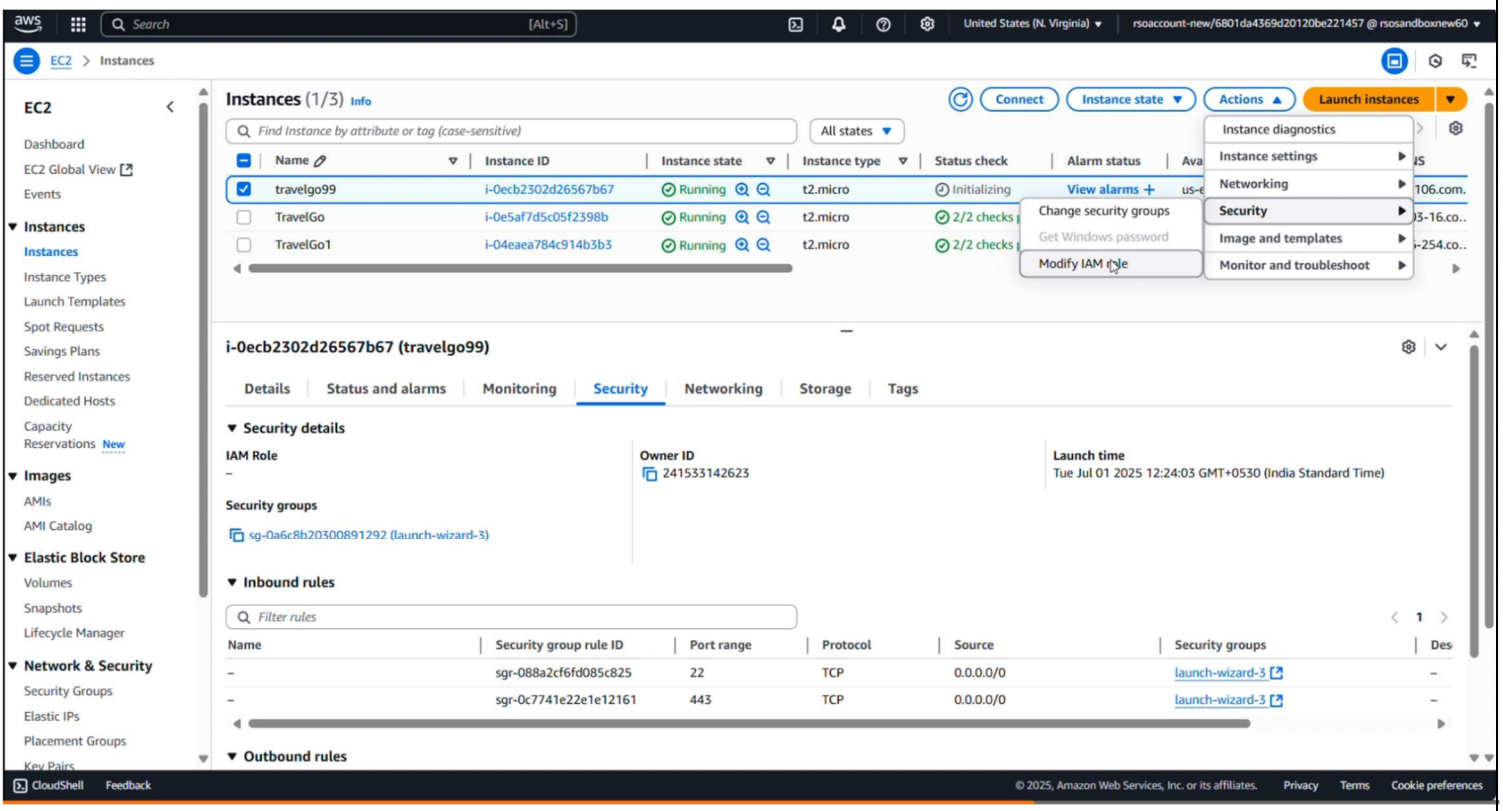


create a key pair:

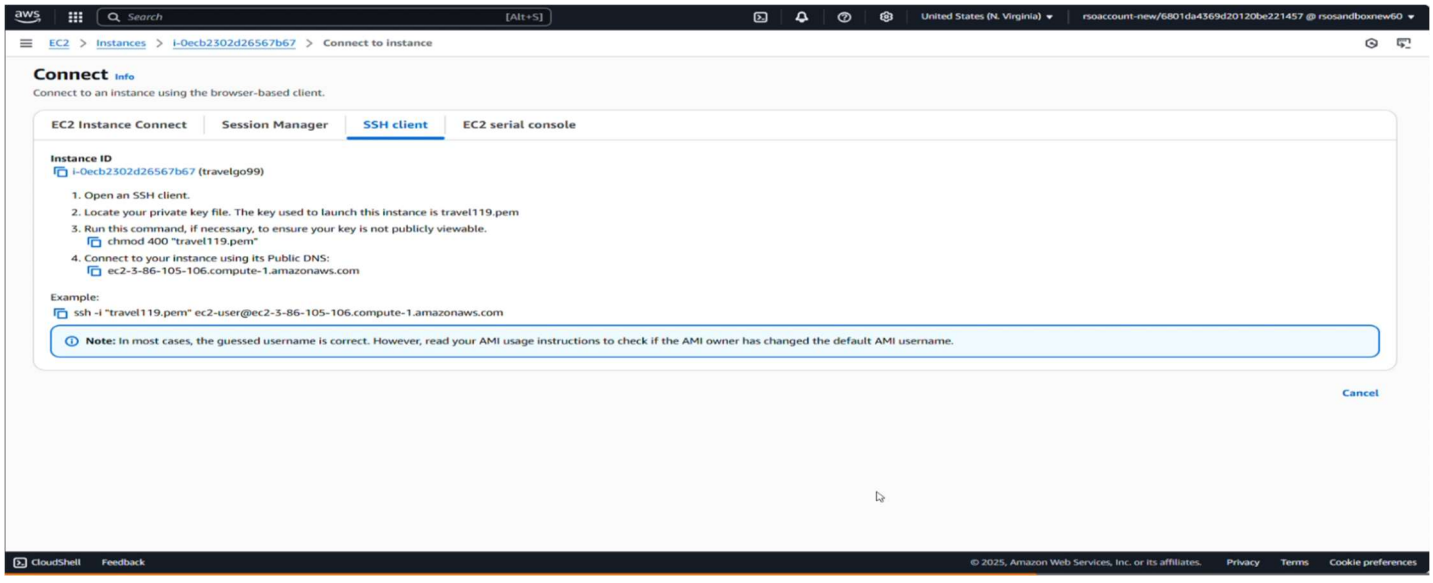


Edit of Security group:




Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew60
EC2 > Instances
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations New
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Instances (1/3) Info
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
travelgo99
i-0ecb2302d26567b67
Running
t2.micro
Initializing
View alarms
TravelGo
i-0e5af7d5c05f2398b
Running
t2.micro
2/2 checks
TravelGo1
i-04eaea784c914b3b3
Running
t2.micro
2/2 checks
Instance diagnostics
Instance settings
Networking
Security
Image and templates
Monitor and troubleshoot
Change security groups
Get Windows password
Modify IAM role
i-0ecb2302d26567b67 (travelgo99)
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Security details
IAM Role
Owner ID
241533142623
Launch time
Tue Jul 01 2025 12:24:03 GMT+0530 (India Standard Time)
Security groups
sg-0a6c8b20300891292 (launch-wizard-3)
Inbound rules
Filter rules
Name
Security group rule ID
Port range
Protocol
Source
Security groups
sgr-088a2cf6fd085c825
22
TCP
0.0.0.0/0
launch-wizard-3
sgr-0c7741e22e1e12161
443
TCP
0.0.0.0/0
launch-wizard-3
Outbound rules
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

EC2 Connet:


Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew60
EC2 > Instances > i-0ecb2302d26567b67 > Connect to instance
Connect Info
Connect to an instance using the browser-based client.
EC2 Instance Connect
Session Manager
SSH client
EC2 serial console
Instance ID
i-0ecb2302d26567b67 (travelgo99)
1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is travel119.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
chmod 400 "travel119.pem"
4. Connect to your instance using its Public DNS:
ec2-3-86-105-106.compute-1.amazonaws.com
Example:
ssh -i "travel119.pem" ec2-user@ec2-3-86-105-106.compute-1.amazonaws.com
Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.
Cancel
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Milestone 7: Deployment on EC2

Activity 7.1: 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

Activity 7.2:Clone Your Flask Project from GitHub

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

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:

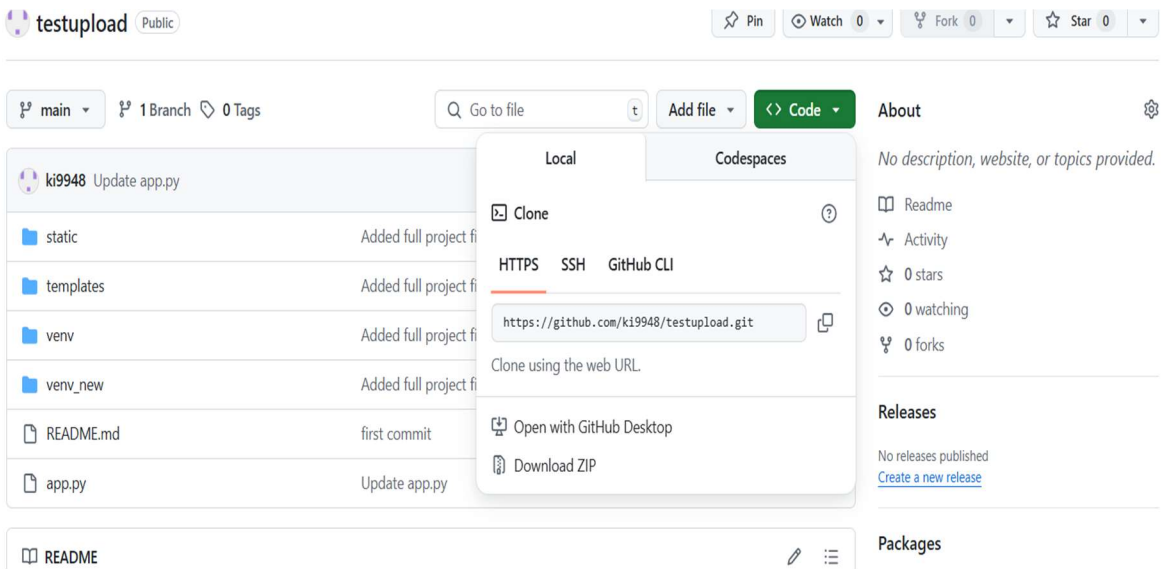
cd InstantLibrary

Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:

Run the Flask Application

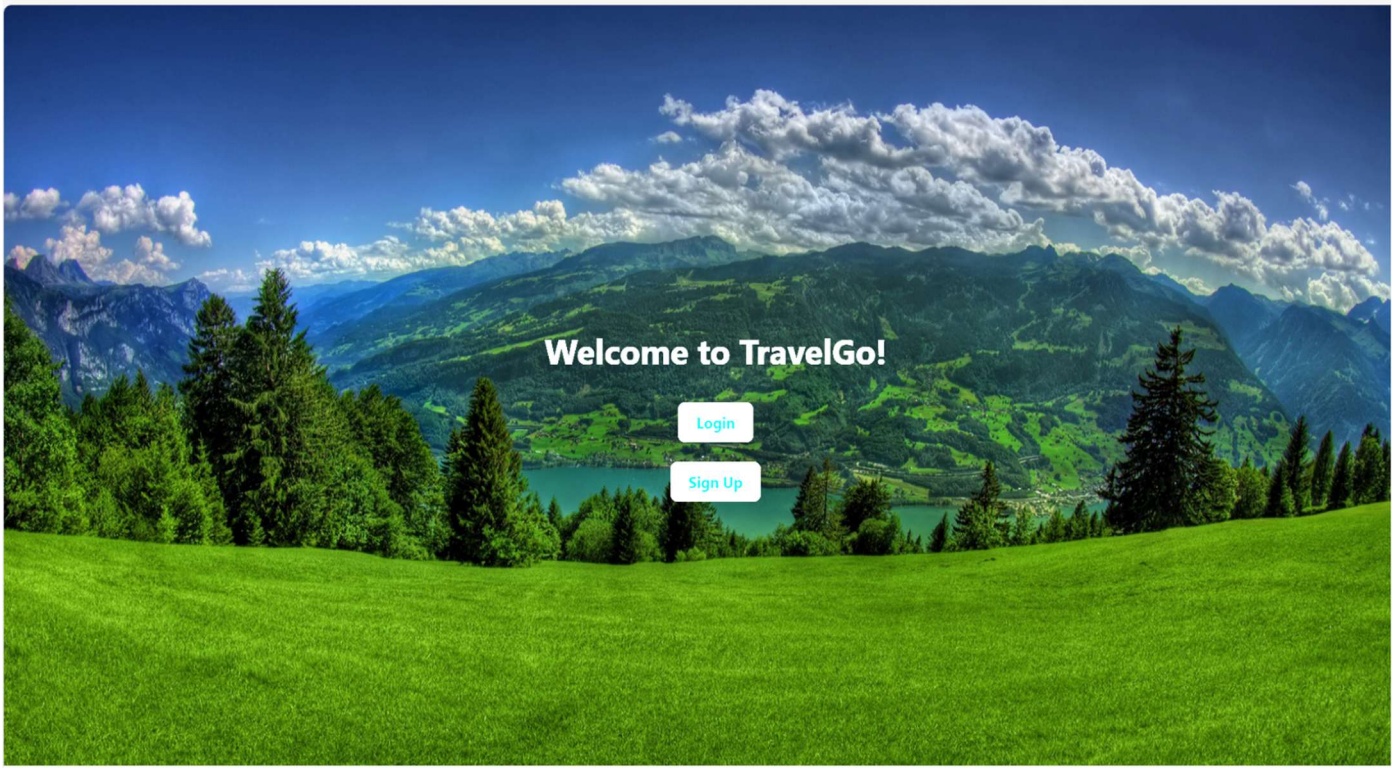
sudo flask run --host=0.0.0.0 --port=80

```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\91781> ssh -i "C:\Users\91781\Downloads\kiran-travelgo.pem" ec2-user@ec2-54-205-203-16.compute-1.amazonaws.com
The authenticity of host 'ec2-54-205-203-16.compute-1.amazonaws.com (54.205.203.16)' can't be established.
ED25519 key fingerprint is SHA256:Bz+Y3Le7rmgyn5pJ+UjGxYE/AuELQuFJ3MTkofrDMeQ.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-54-205-203-16.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-88-122 ~]$ sudo yum install git -y
Amazon Linux 2023 Kernel Livepatch repository                    175 kB/s |  17 kB     00:00
Dependencies resolved.
================================================================================================
 Package                 Architecture        Version                        Repository      Size
================================================================================================
Installing:
 git                     x86_64              2.47.1-1.amzn2023.0.3          amazonlinux     52 k
Installing dependencies:
 git-core                x86_64              2.47.1-1.amzn2023.0.3          amazonlinux    4.5 M
 git-core-doc            noarch              2.47.1-1.amzn2023.0.3          amazonlinux    2.8 M
 perl-Error              noarch              1:0.17029-5.amzn2023.0.2       amazonlinux     41 k
 perl-File-Find          noarch              1.37-477.amzn2023.0.7          amazonlinux     25 k
 perl-Git                noarch              2.47.1-1.amzn2023.0.3          amazonlinux     40 k
 perl-TermReadKey        x86_64              2.38-9.amzn2023.0.2            amazonlinux     36 k
 perl-lib                x86_64              0.65-477.amzn2023.0.7          amazonlinux     15 k

Transaction Summary
================================================================================================
Install  8 Packages

Total download size: 7.5 M
Installed size: 37 M
Downloading Packages:
(1/8): git-2.47.1-1.amzn2023.0.3.x86_64.rpm                      1.4 MB/s |  52 kB     00:00
(2/8): perl-Error-0.17029-5.amzn2023.0.2.noarch.rpm              1.6 MB/s |  41 kB     00:00
```

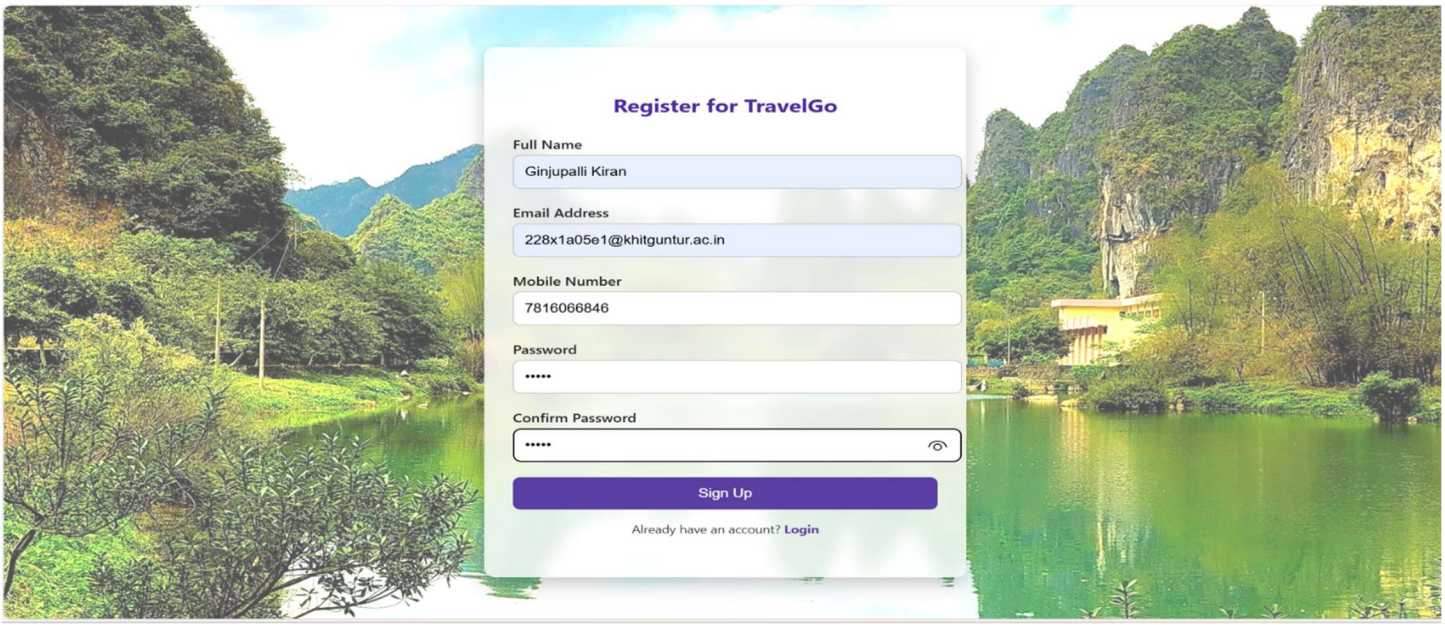


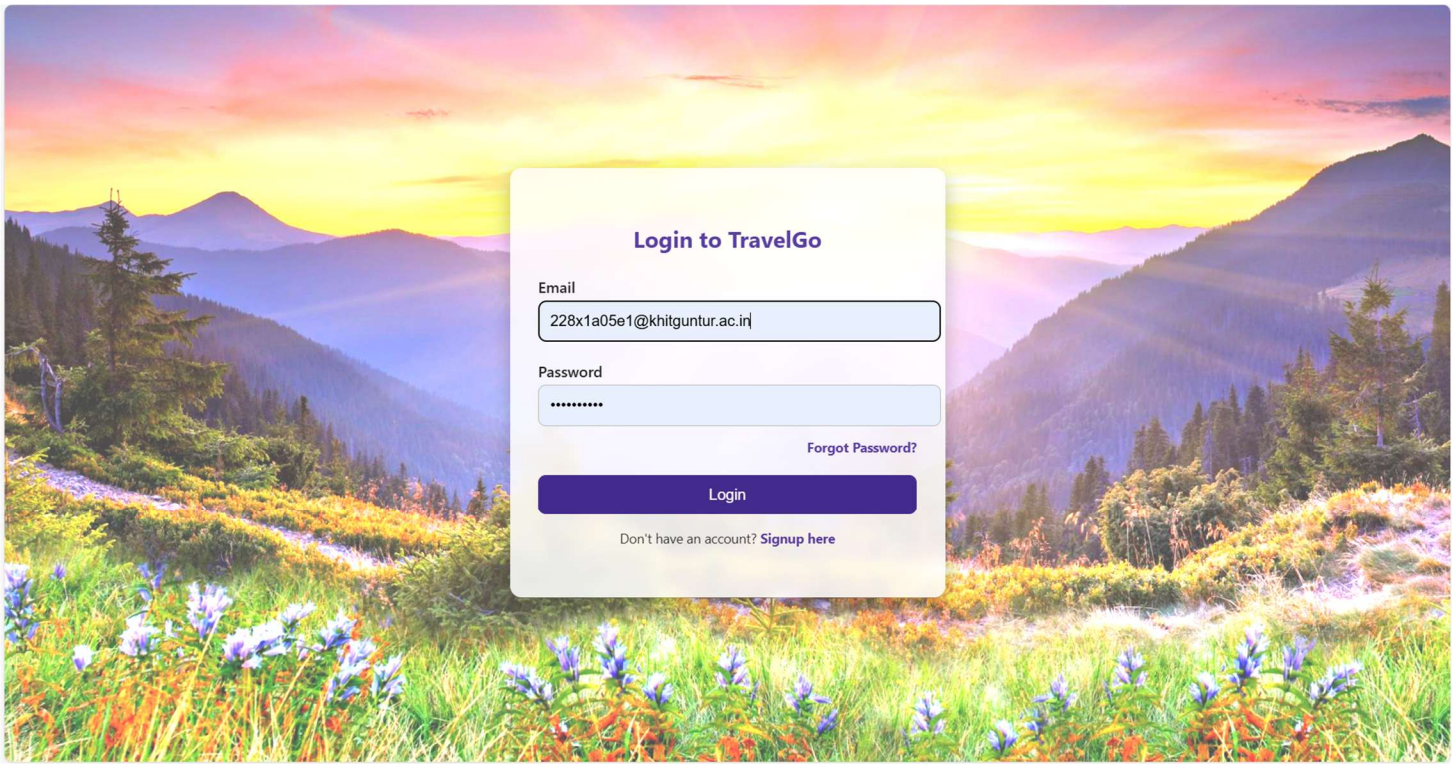
```
     |████████████████████████████████| 134 kB 49.1 MB/s
Collecting itsdangerous>=2.2.0
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting blinker>=1.9.0
  Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Collecting werkzeug>=3.1.0
  Downloading werkzeug-3.1.3-py3-none-any.whl (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:

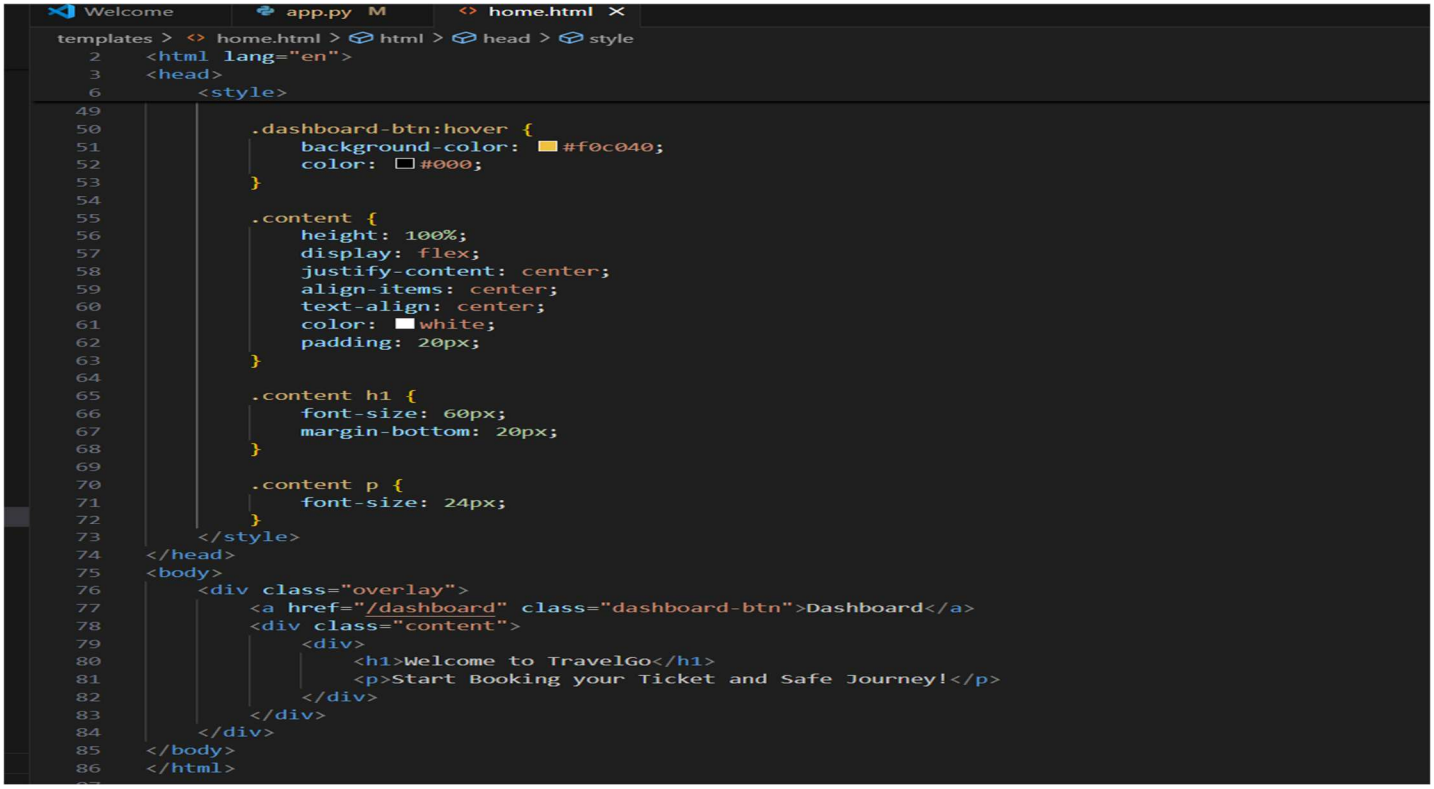


Register page:





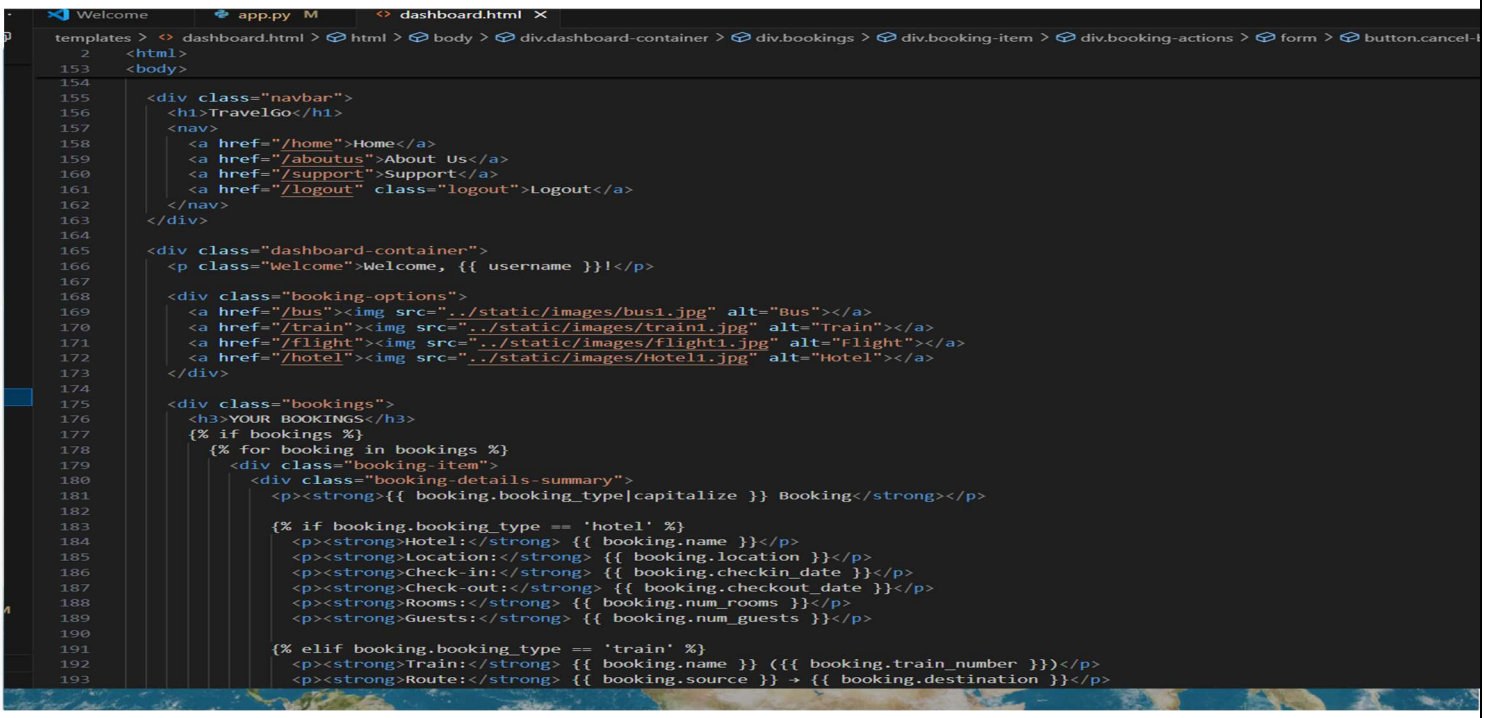
Login page:



```
templates > <> home.html > html > head > style
  2   <html lang="en">
  3   <head>
  6       <style>
 49
 50           .dashboard-btn:hover {
 51               background-color: #f0c040;
 52               color: #000;
 53           }
 54
 55           .content {
 56               height: 100%;
 57               display: flex;
 58               justify-content: center;
 59               align-items: center;
 60               text-align: center;
 61               color: white;
 62               padding: 20px;
 63           }
 64
 65           .content h1 {
 66               font-size: 60px;
 67               margin-bottom: 20px;
 68           }
 69
 70           .content p {
 71               font-size: 24px;
 72           }
 73       </style>
 74   </head>
 75   <body>
 76       <div class="overlay">
 77           <a href="/dashboard" class="dashboard-btn">Dashboard</a>
 78           <div class="content">
 79               <div>
 80                   <h1>Welcome to TravelGo</h1>
 81                   <p>Start Booking your Ticket and Safe Journey!</p>
 82               </div>
 83           </div>
 84       </div>
 85   </body>
 86   </html>
```

Home Code:

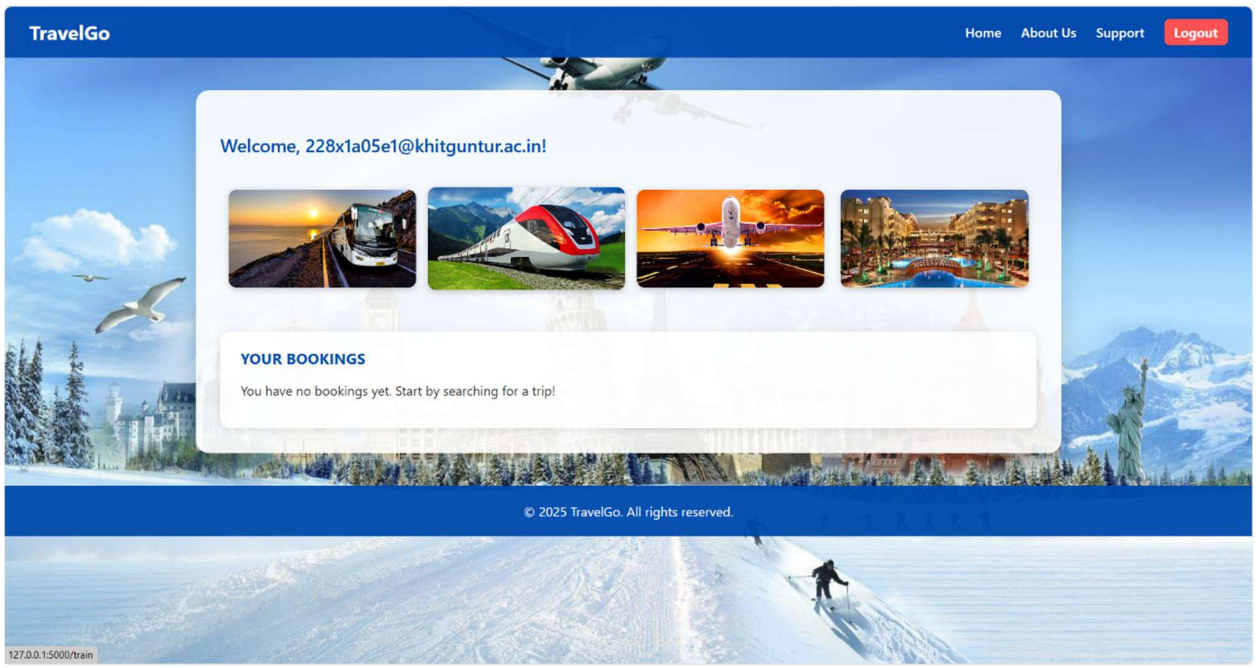
Home page:



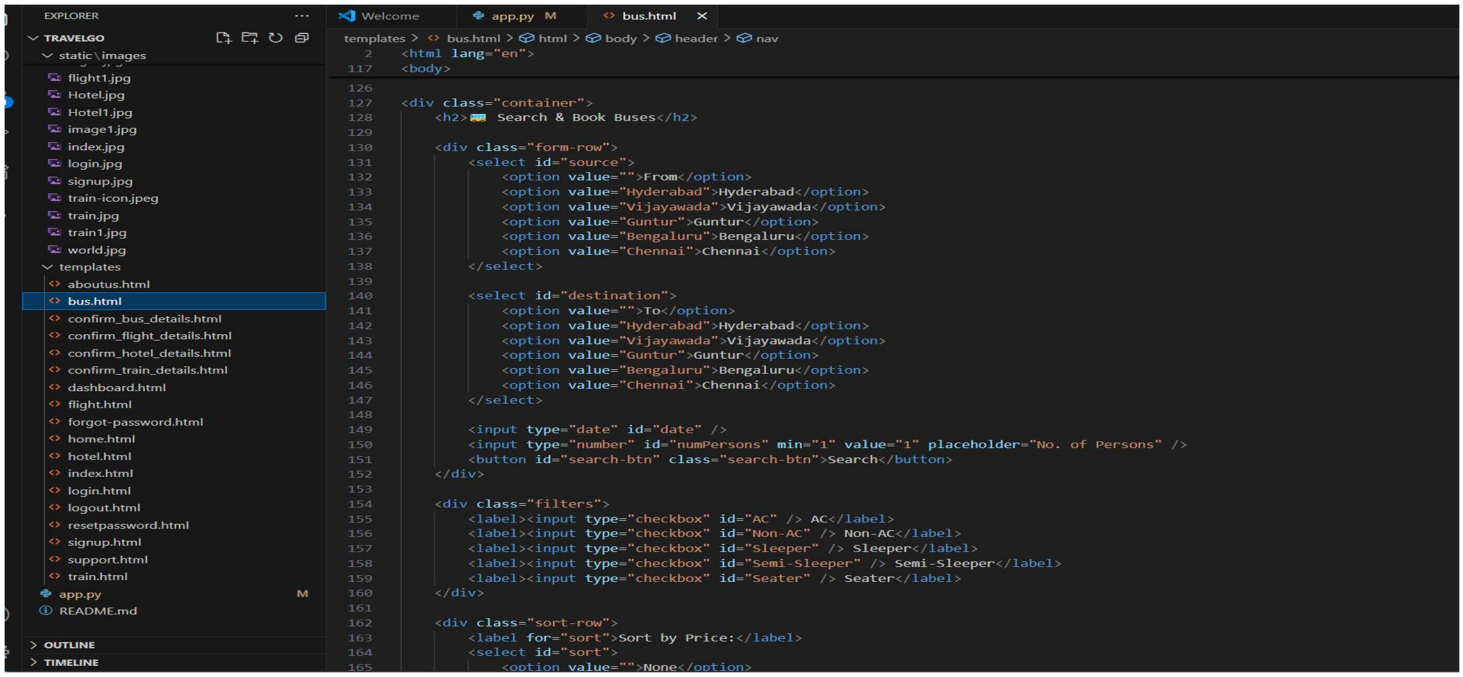
```
templates > dashboard.html > html > body > div.dashboard-container > div.bookings > div.booking-item > div.booking-actions > form > button.cancel-
  2   <html>
153   <body>
154
155     <div class="navbar">
156       <h1>TravelGo</h1>
157       <nav>
158         <a href="/home">Home</a>
159         <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 code:

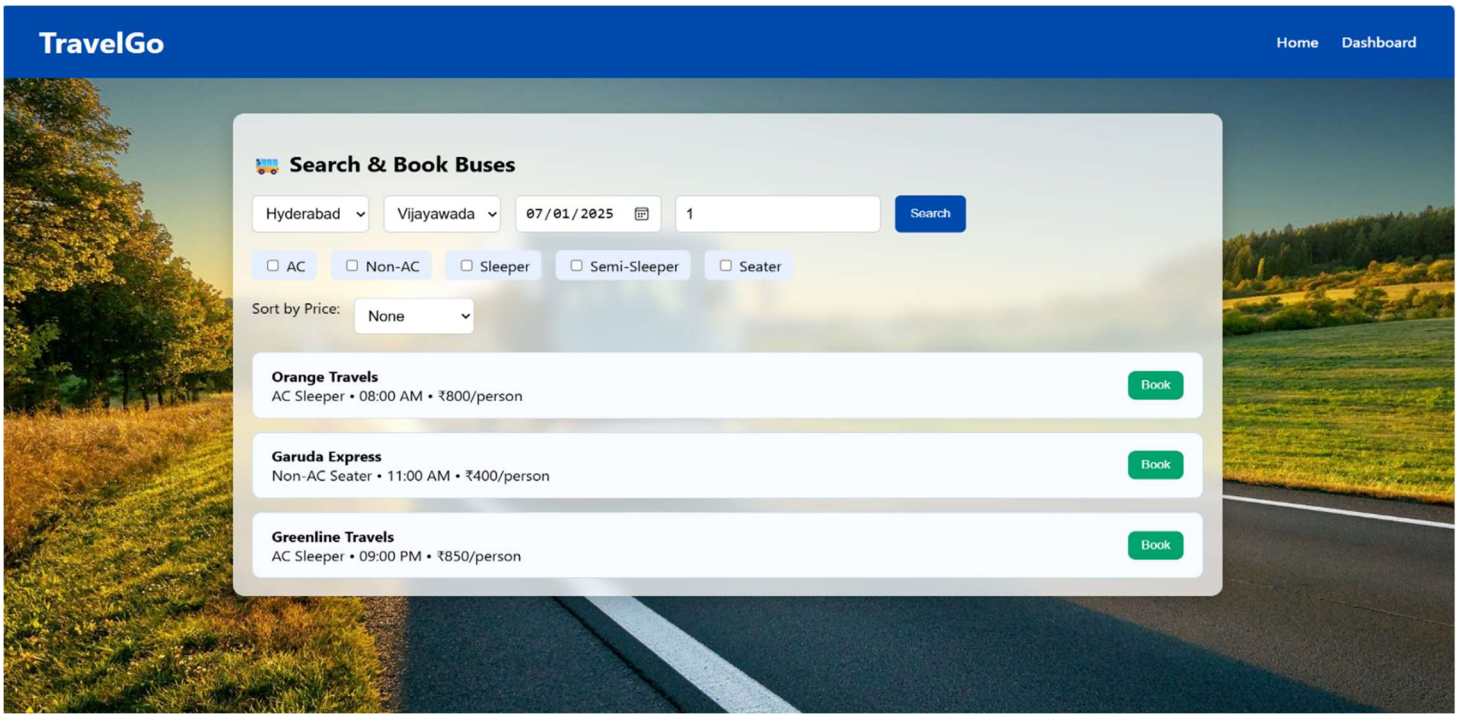
Dashboard page:


TravelGo
Home
About Us
Support
Logout
Welcome, 228x1a05e1@khitguntur.ac.in!
YOUR BOOKINGS
You have no bookings yet. Start by searching for a trip!
© 2025 TravelGo. All rights reserved.
127.0.0.1:5000/train

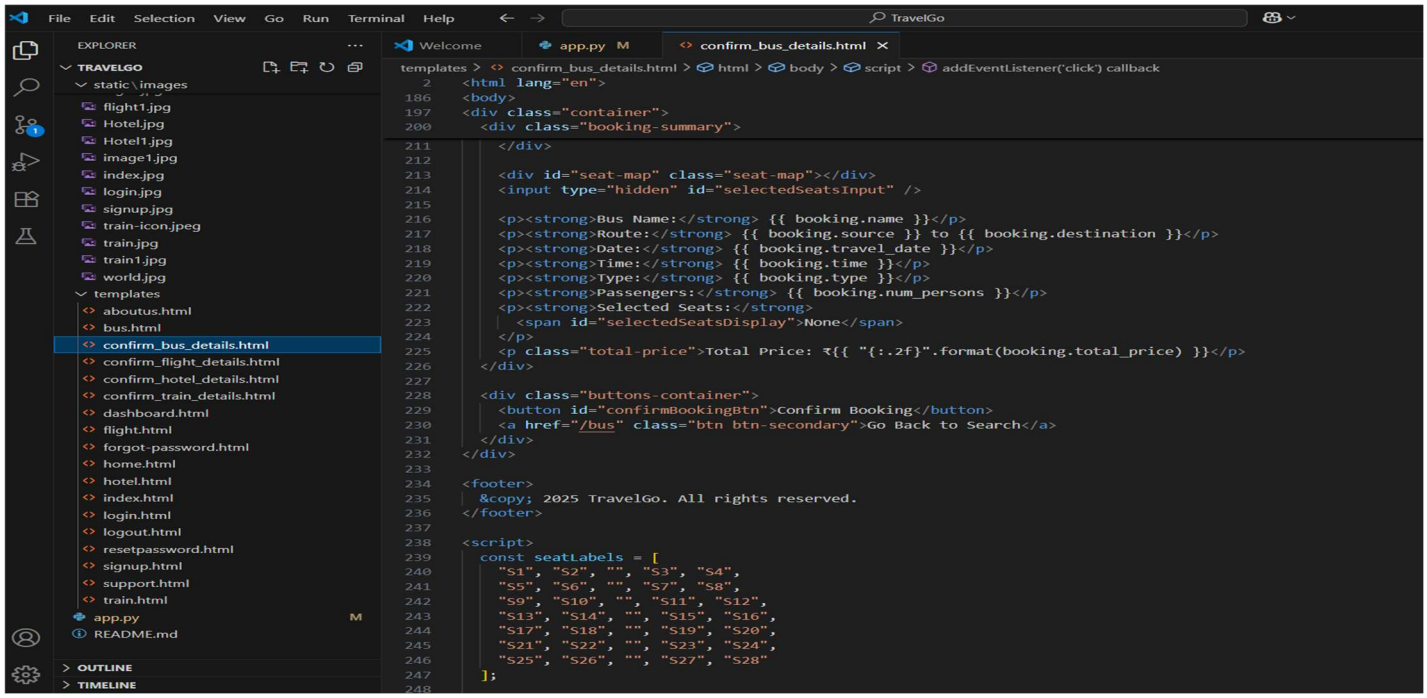
Bus Booking code:



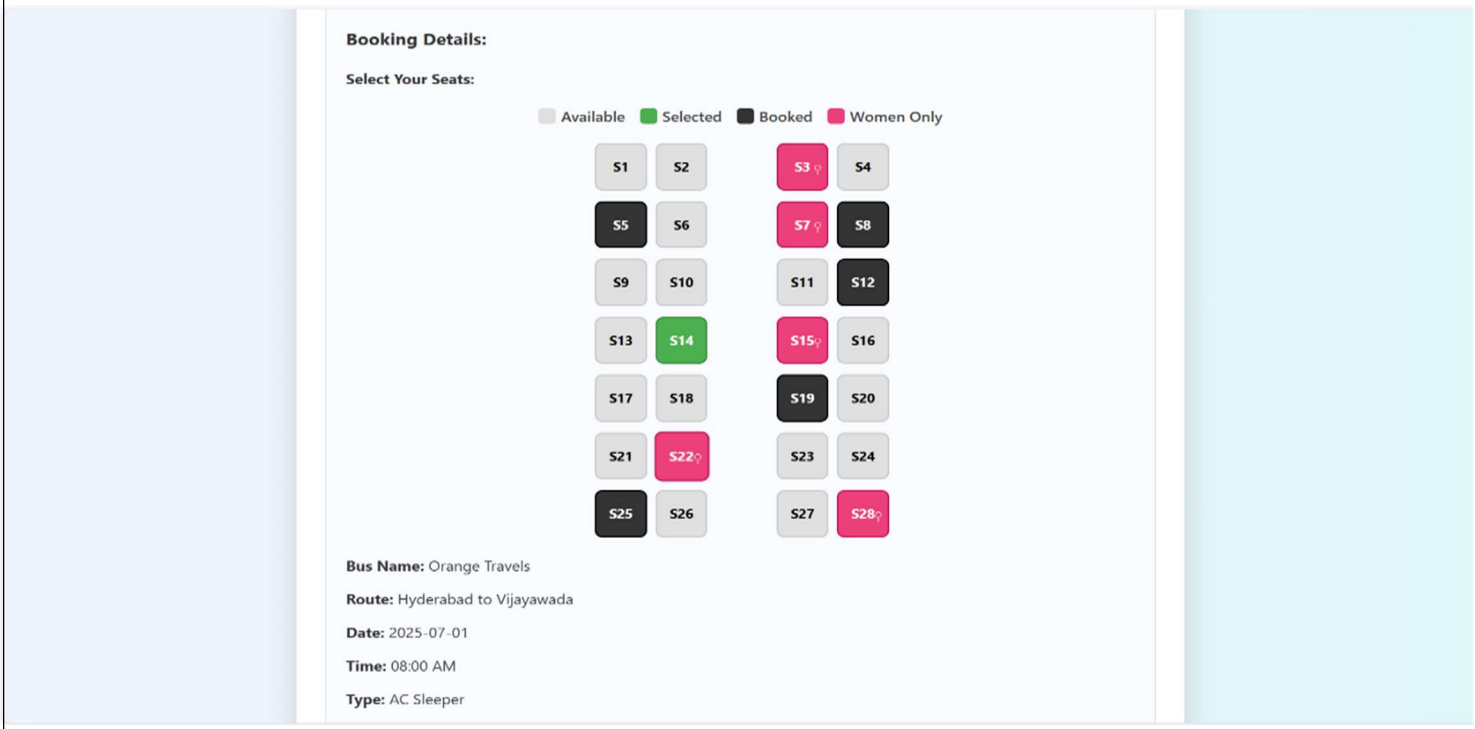
Bus booking page:



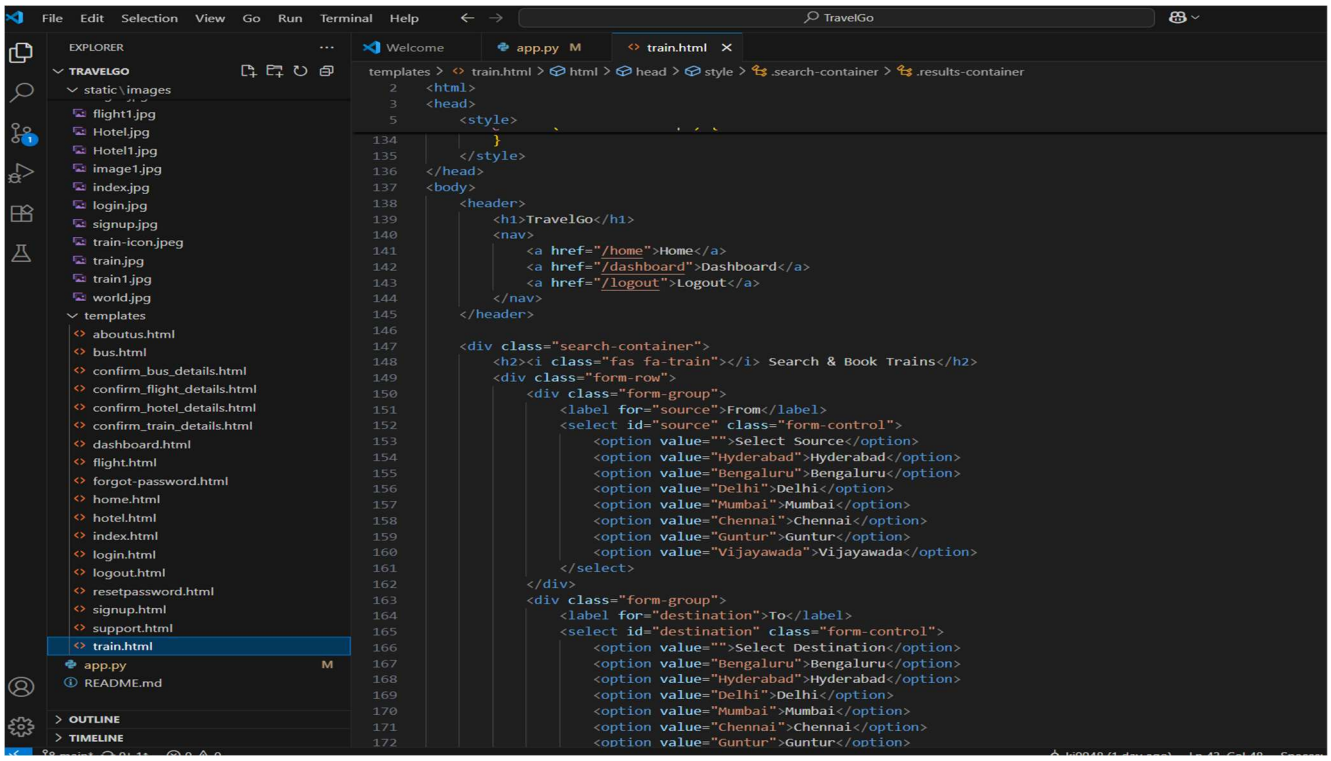
Bus Confirm Booking code:



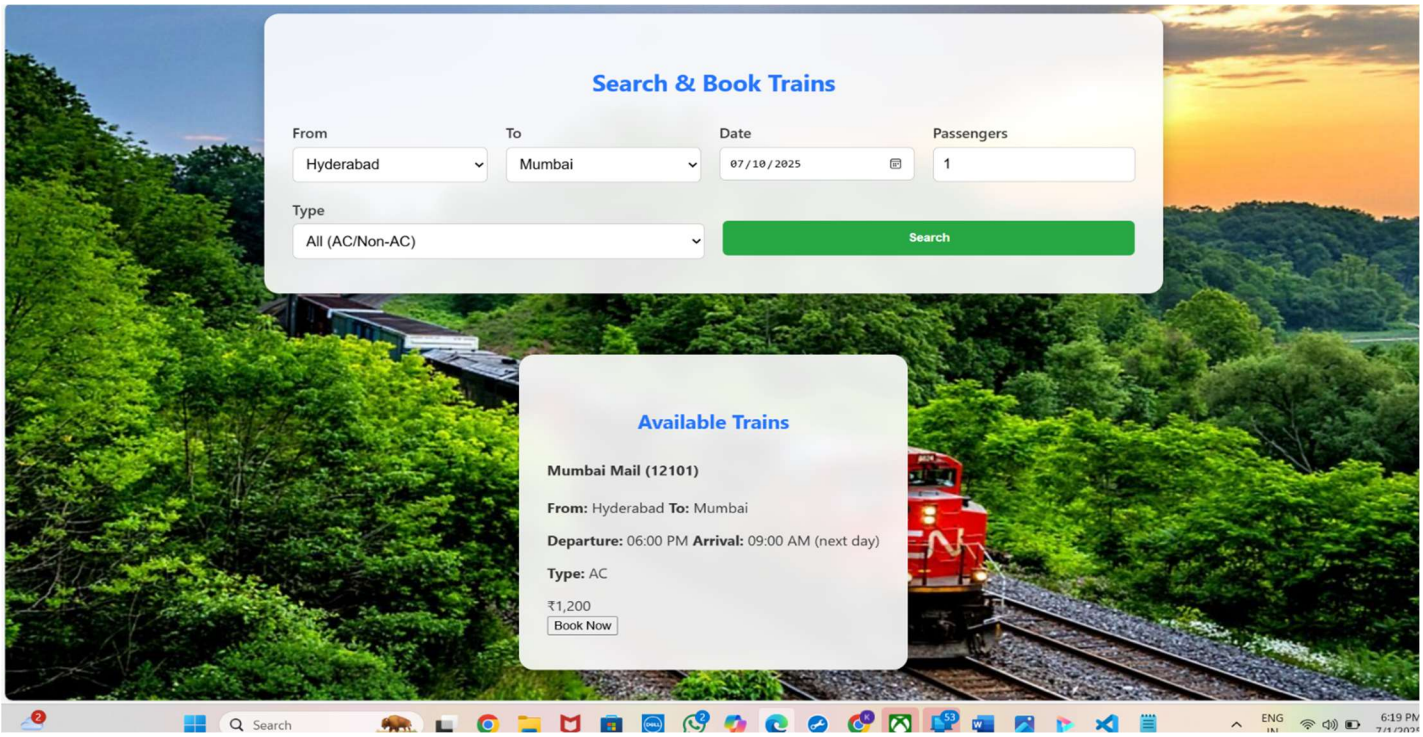
Bus confirm booking page:



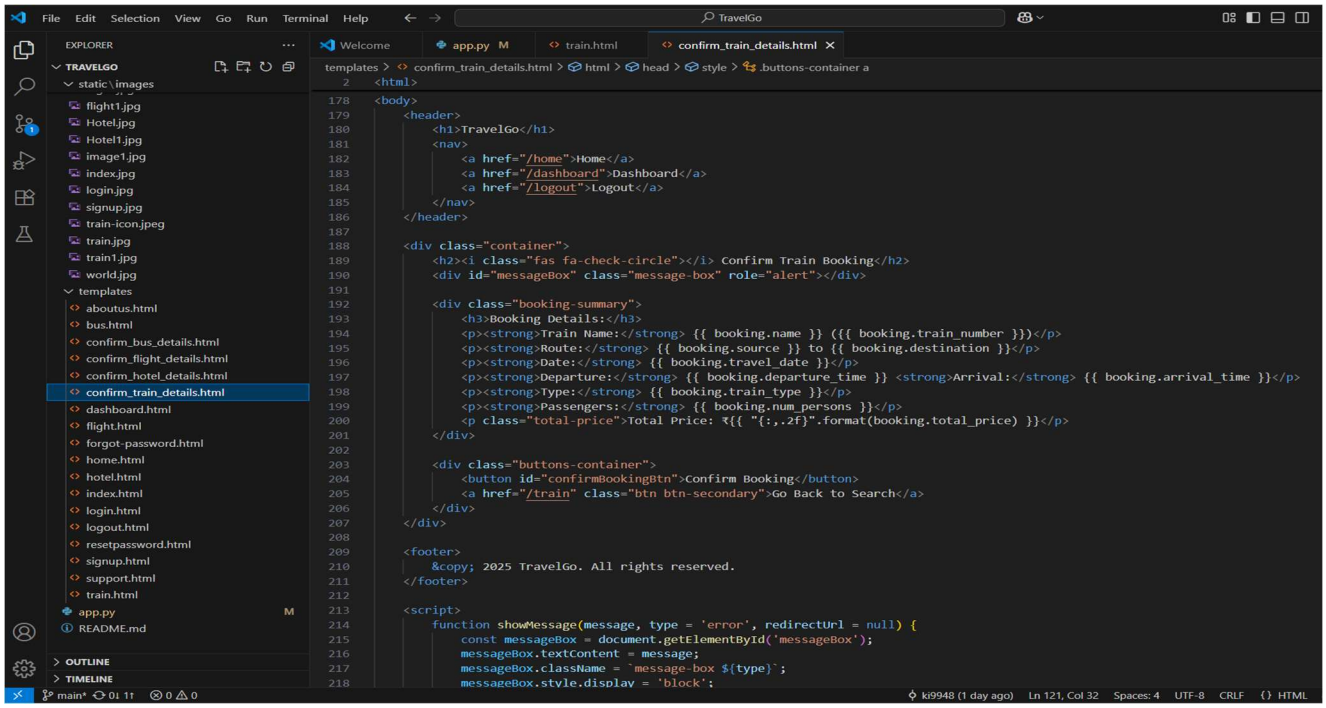
Train code :



Train booking page:

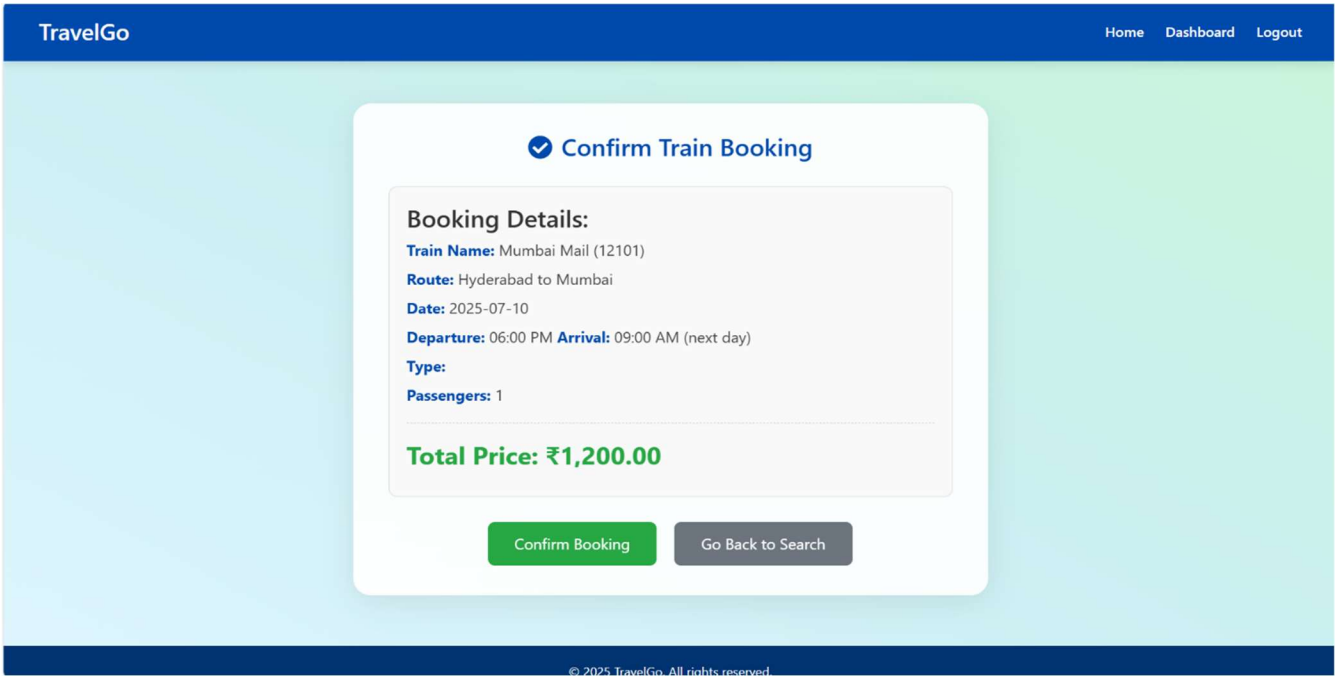


Train confirm booking code:

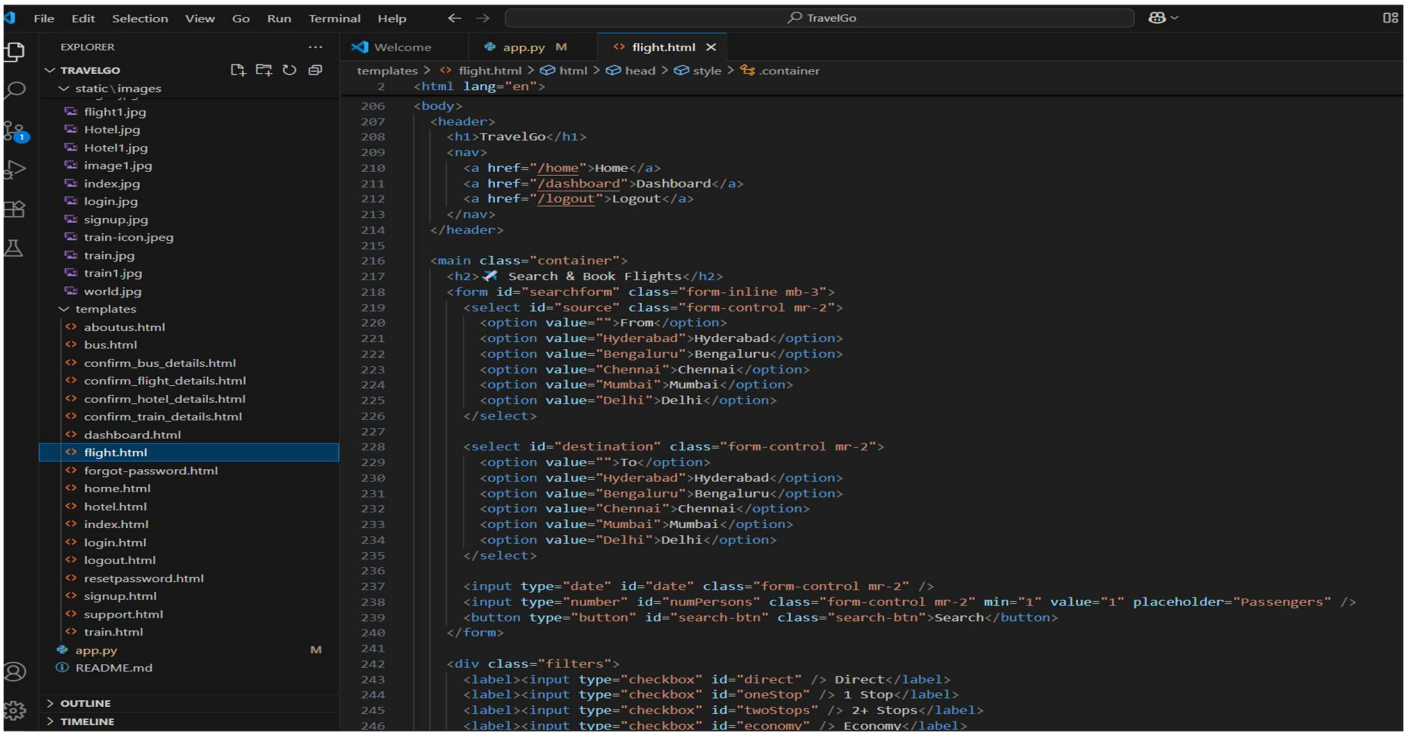


```
<html>
<body>
    <header>
        <h1>TravelGo</h1>
        <nav>
            <a href="/home">Home</a>
            <a href="/dashboard">Dashboard</a>
            <a href="/logout">Logout</a>
        </nav>
    </header>

    <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';
```

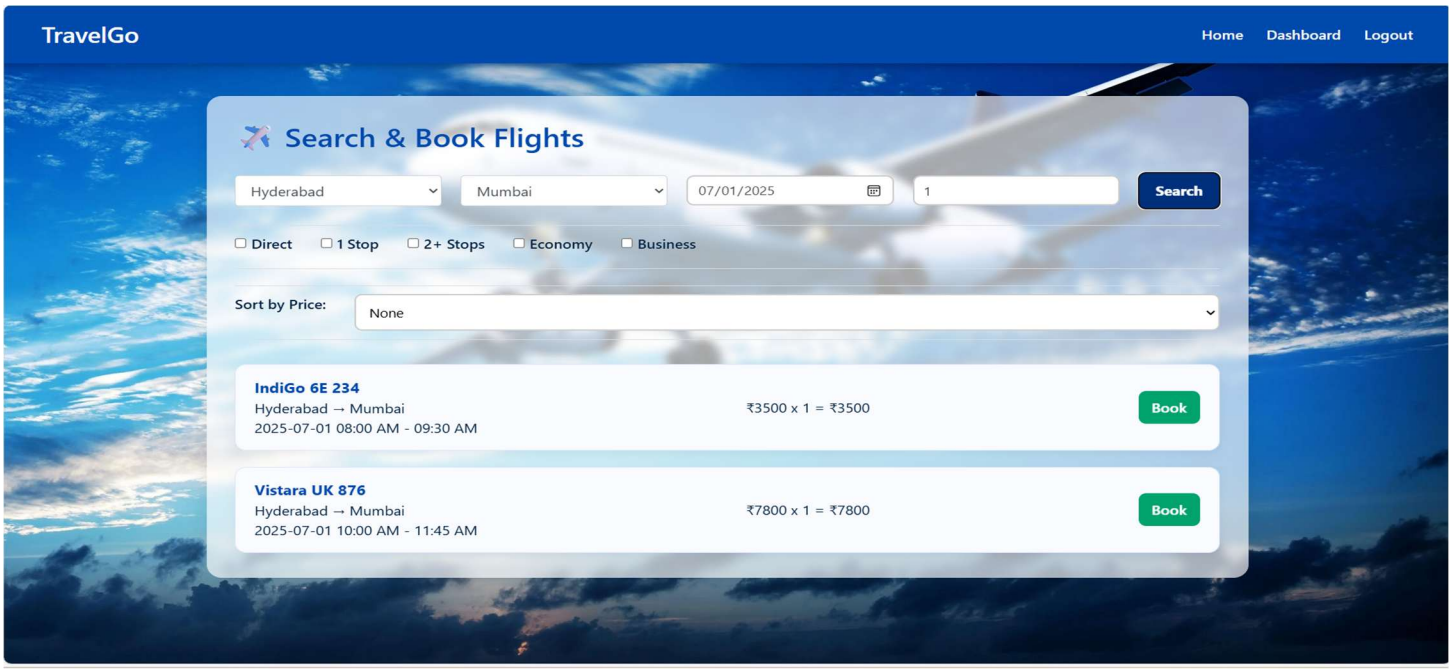
Train confirm booking page:



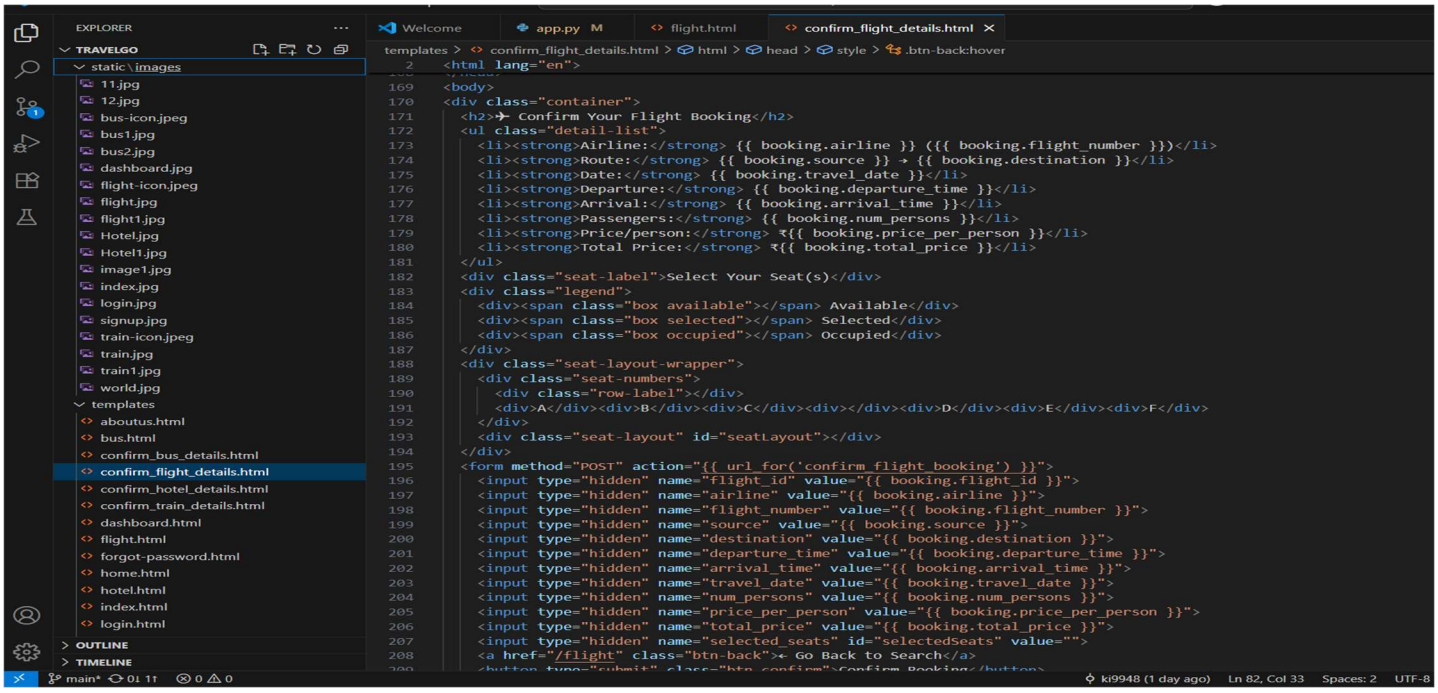
Flight code:



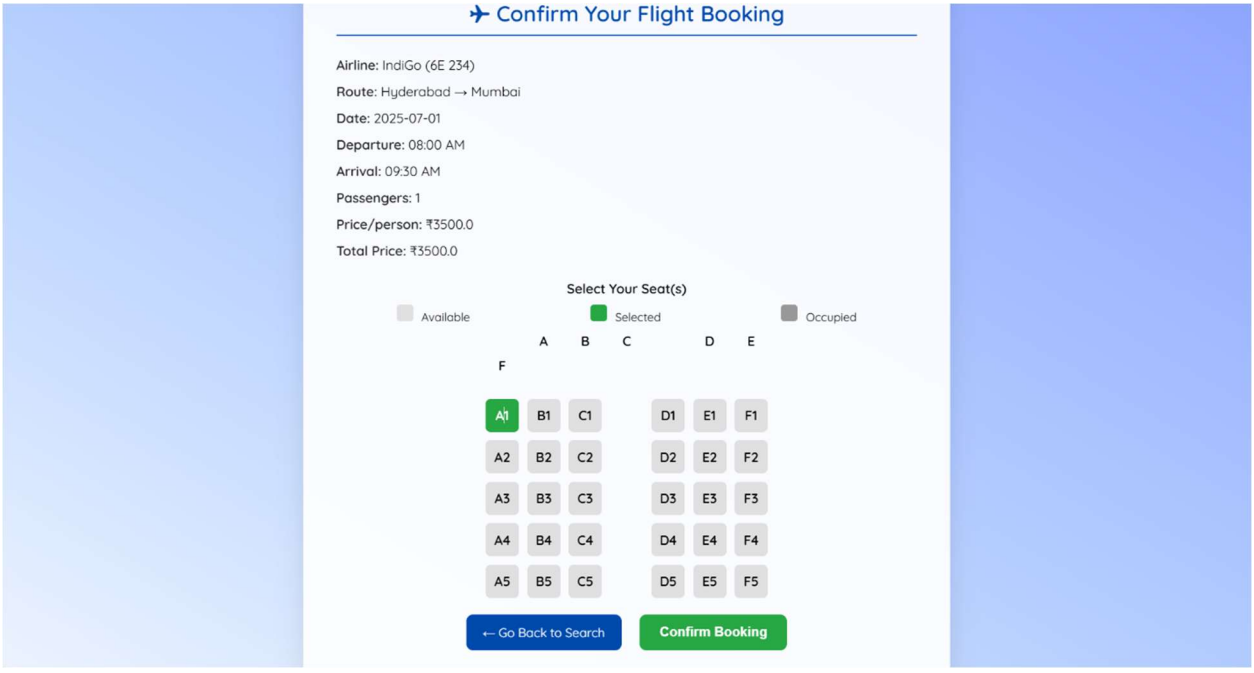
Flight page:



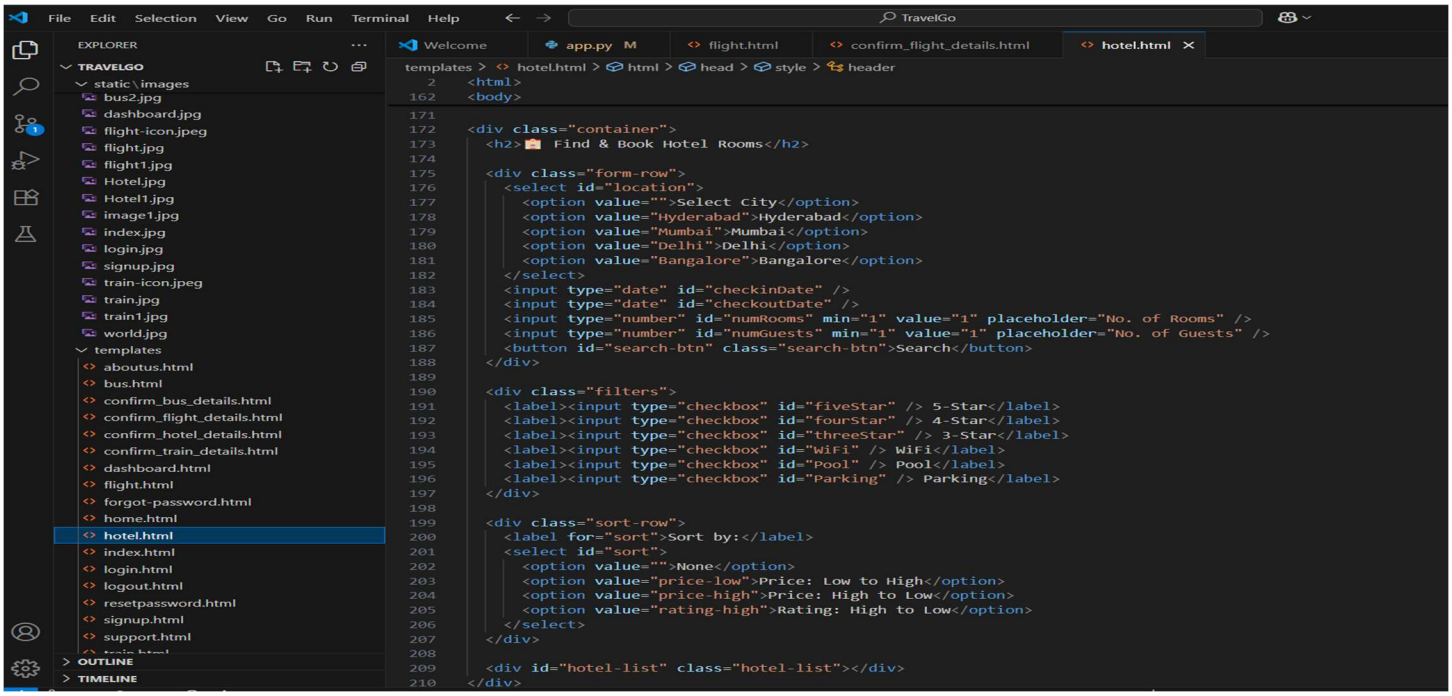
Flight confirm booking code:



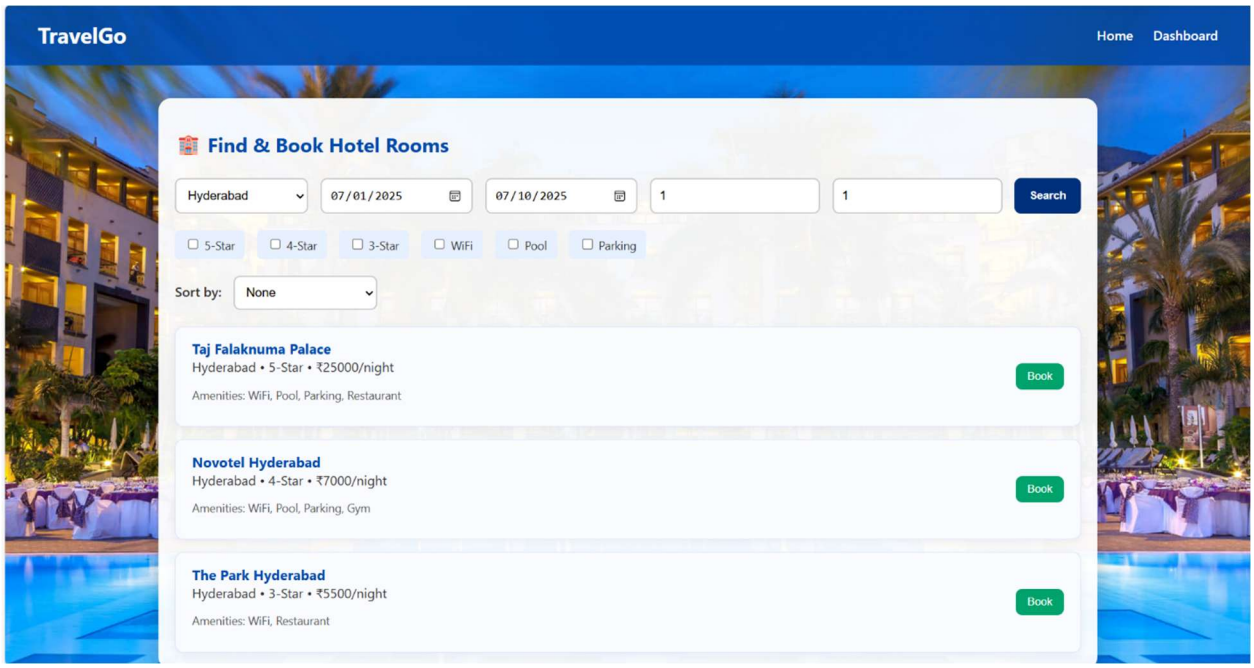
Flight confirm booking page:



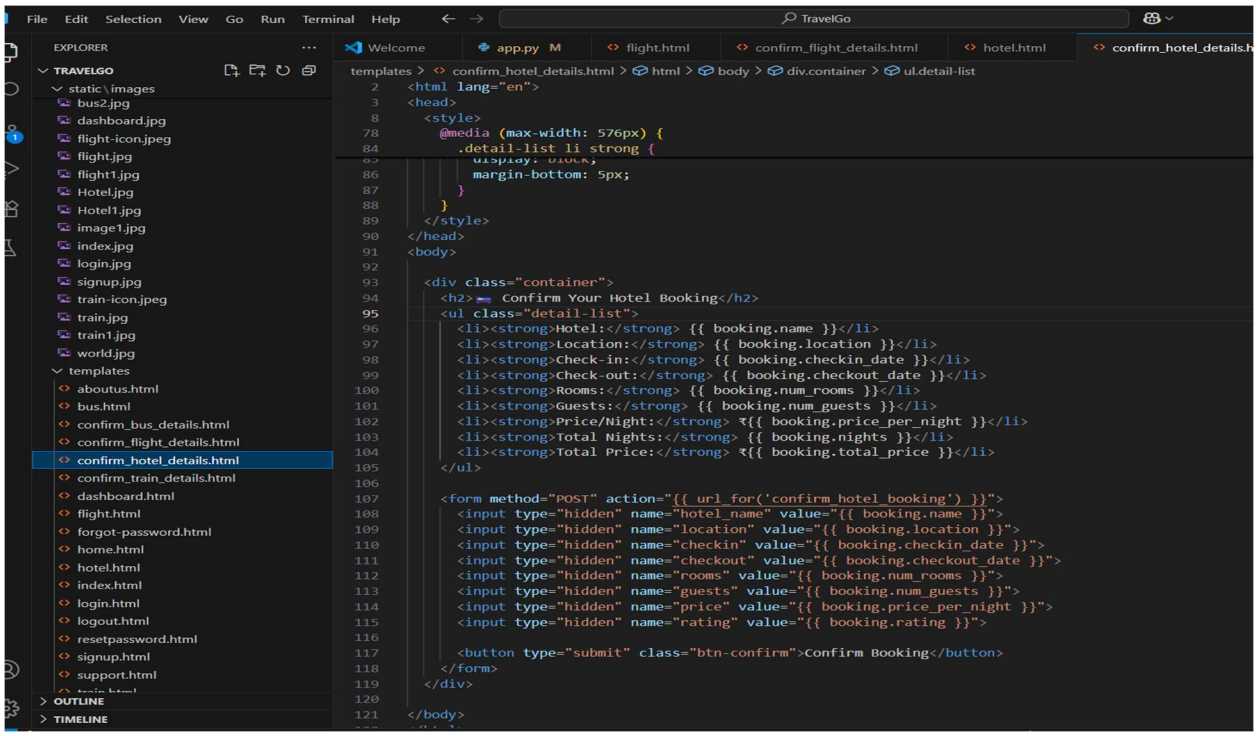
Hotel code:



Hotel page:

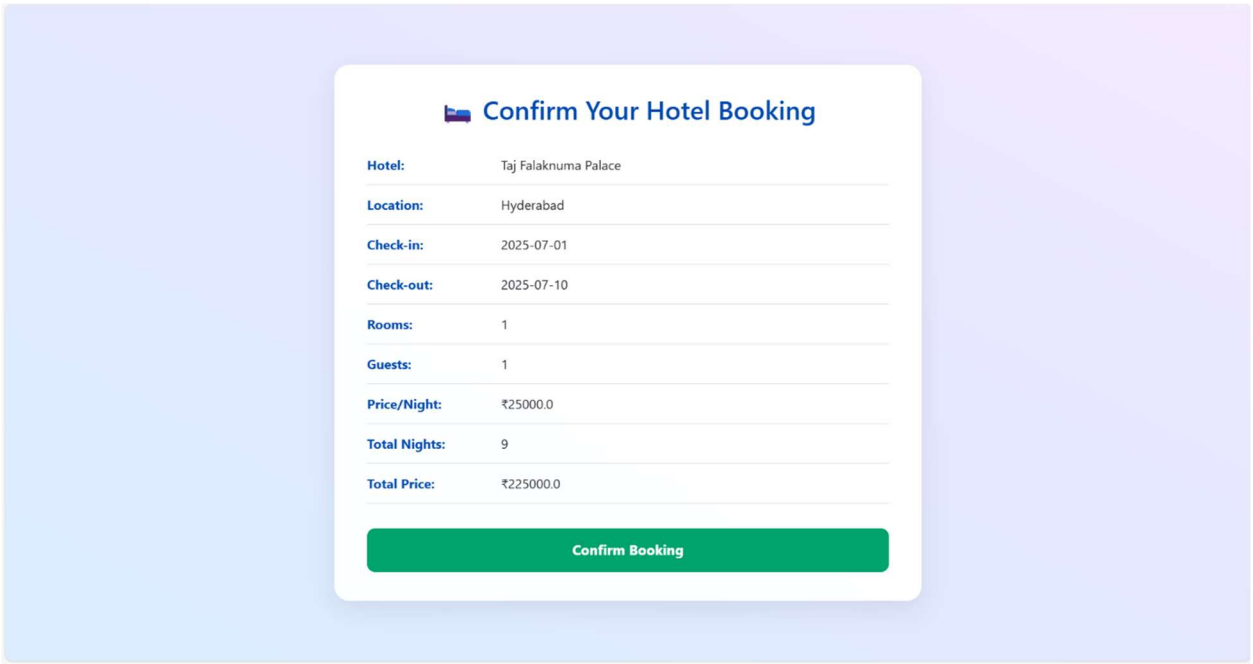


Hotel confirm booking code:

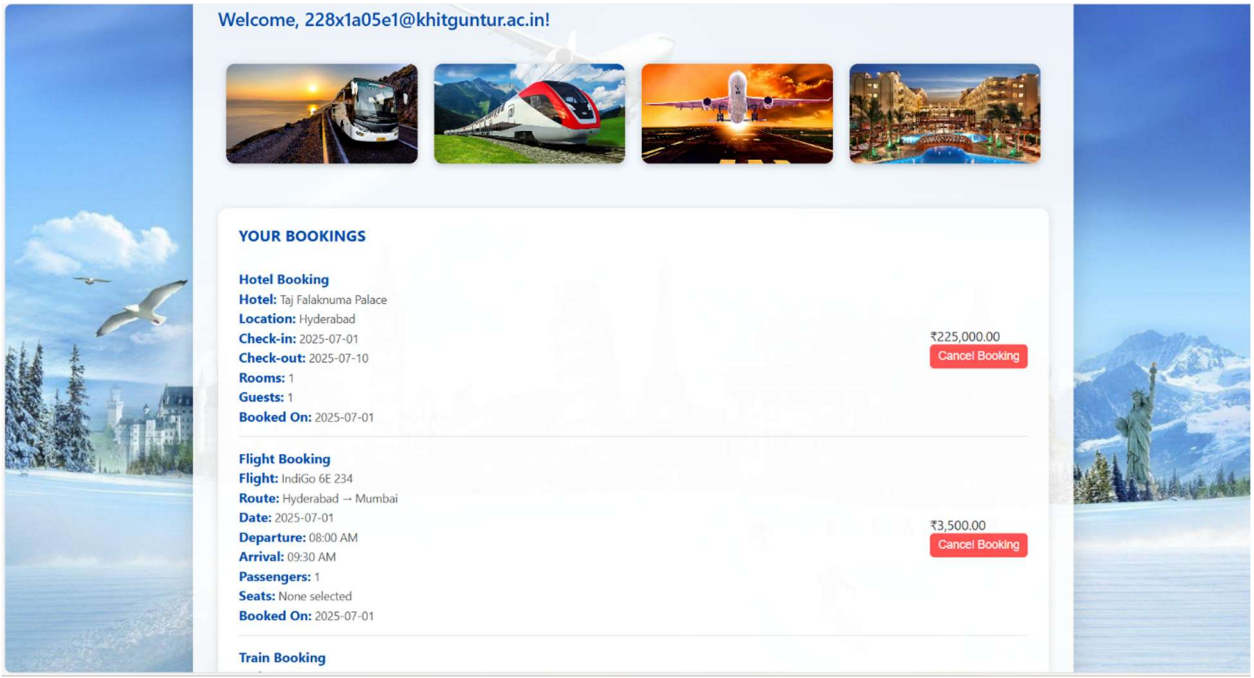


```
templates > confirm_hotel_details.html > html > body > div.container > ul.detail-list
<html lang="en">
<head>
  <style>
    @media (max-width: 576px) {
      .detail-list li strong {
        display: block;
        margin-bottom: 5px;
      }
    }
  </style>
</head>
<body>

  <div class="container">
    <h2>🛏️ Confirm Your Hotel Booking</h2>
    <ul class="detail-list">
      <li><strong>Hotel:</strong> {{ booking.name }}</li>
      <li><strong>Location:</strong> {{ booking.location }}</li>
      <li><strong>Check-in:</strong> {{ booking.checkin_date }}</li>
      <li><strong>Check-out:</strong> {{ booking.checkout_date }}</li>
      <li><strong>Rooms:</strong> {{ booking.num_rooms }}</li>
      <li><strong>Guests:</strong> {{ booking.num_guests }}</li>
      <li><strong>Price/Night:</strong> ₹{{ booking.price_per_night }}</li>
      <li><strong>Total Nights:</strong> {{ booking.nights }}</li>
      <li><strong>Total Price:</strong> ₹{{ booking.total_price }}</li>
    </ul>

    <form method="POST" action="{{ url_for('confirm_hotel_booking') }}">
      <input type="hidden" name="hotel_name" value="{{ booking.name }}">
      <input type="hidden" name="location" value="{{ booking.location }}">
      <input type="hidden" name="checkin" value="{{ booking.checkin_date }}">
      <input type="hidden" name="checkout" value="{{ booking.checkout_date }}">
      <input type="hidden" name="rooms" value="{{ booking.num_rooms }}">
      <input type="hidden" name="guests" value="{{ booking.num_guests }}">
      <input type="hidden" name="price" value="{{ booking.price_per_night }}">
      <input type="hidden" name="rating" value="{{ booking.rating }}">

      <button type="submit" class="btn-confirm">Confirm Booking</button>
    </form>
  </div>

</body>
```

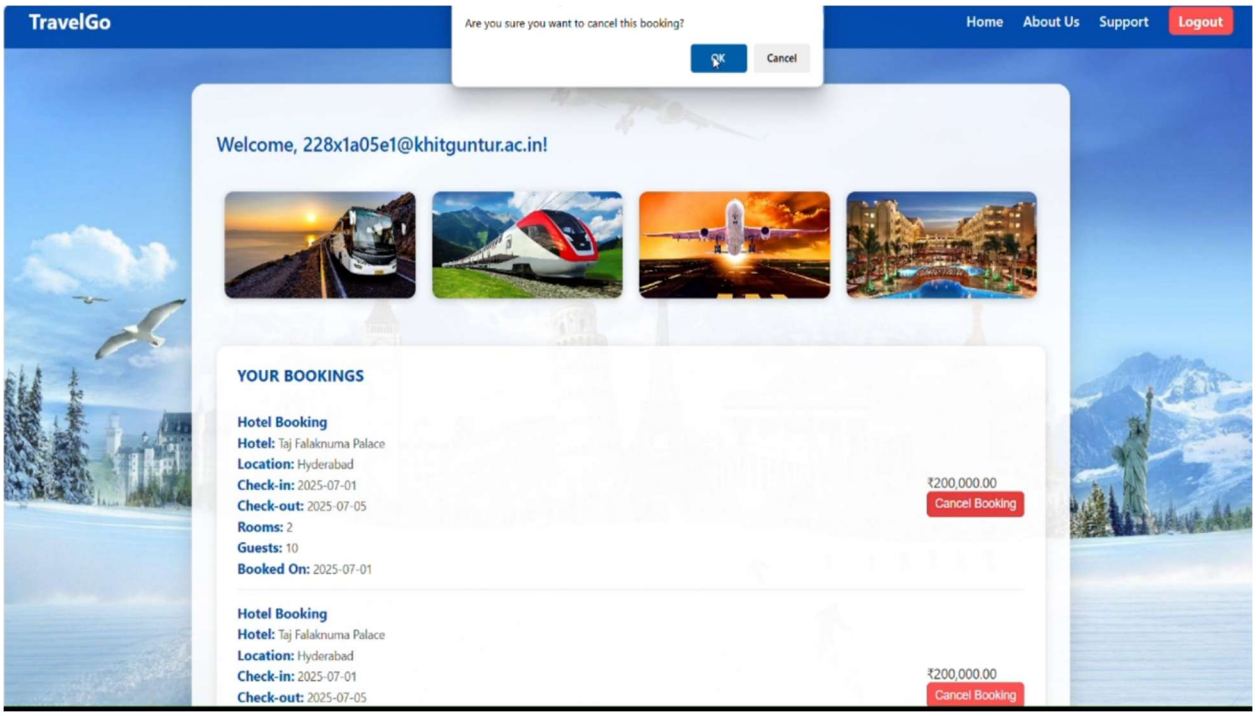
Hotel confirm booking page:



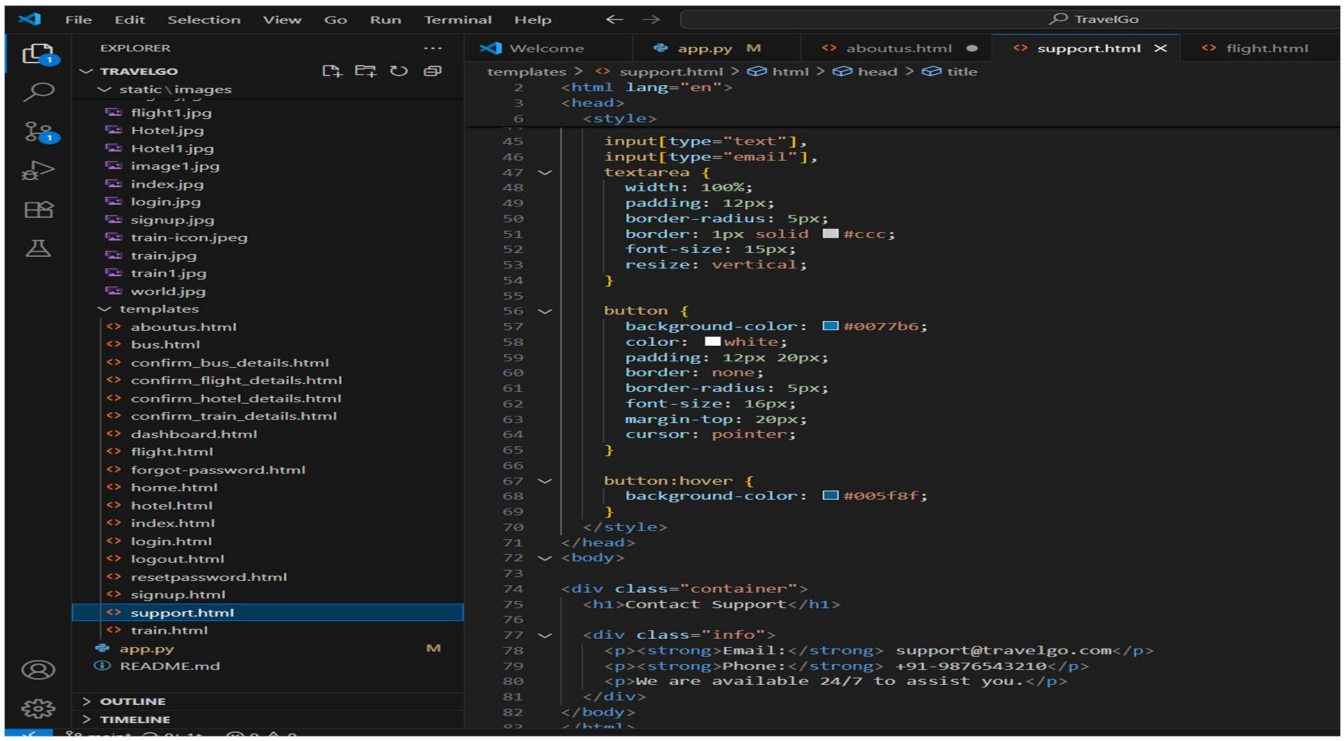
Bookings page:



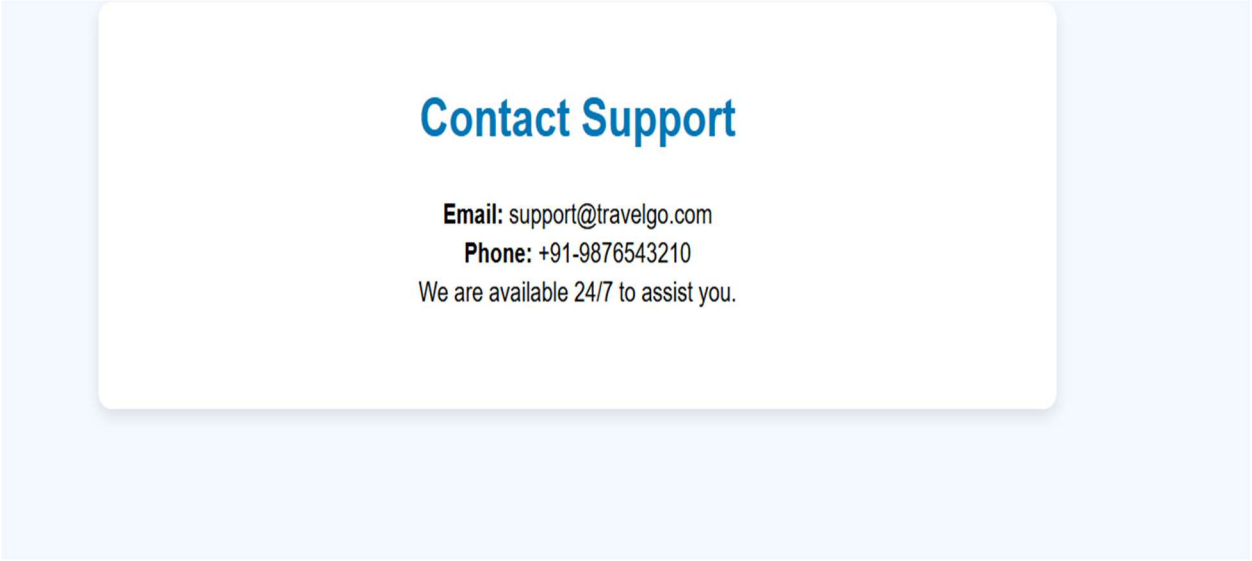
Cancel booking p



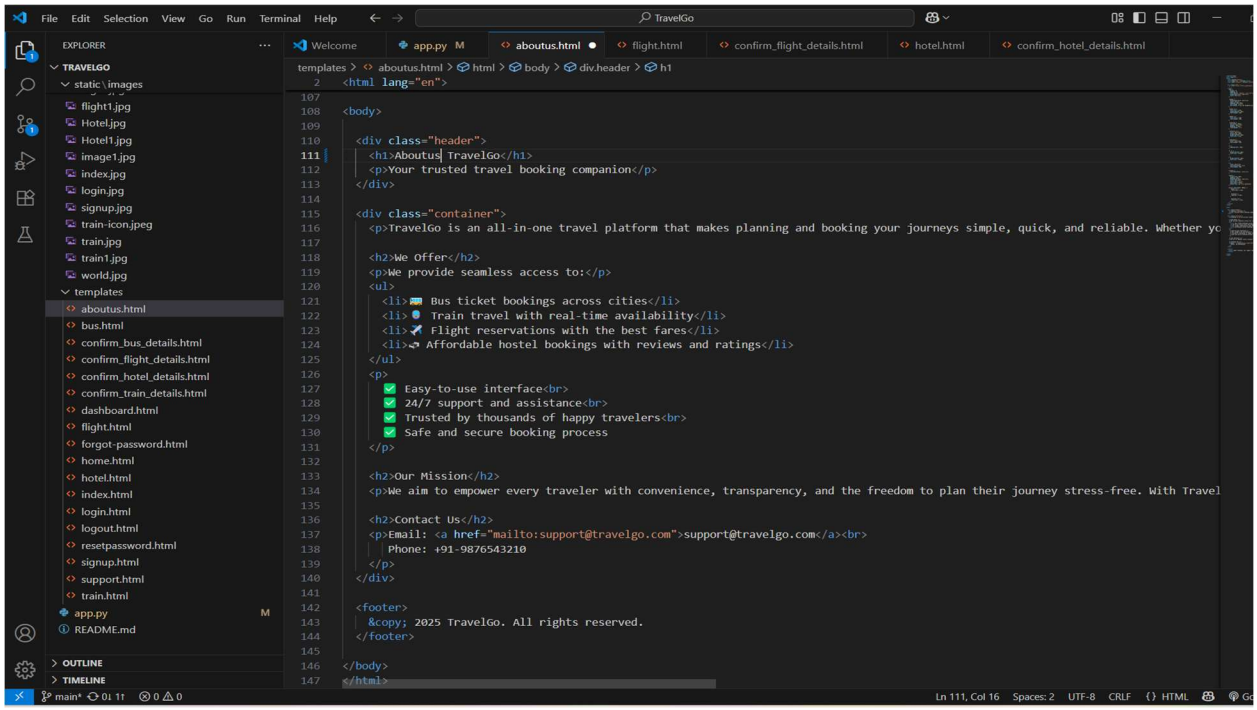
Support code:



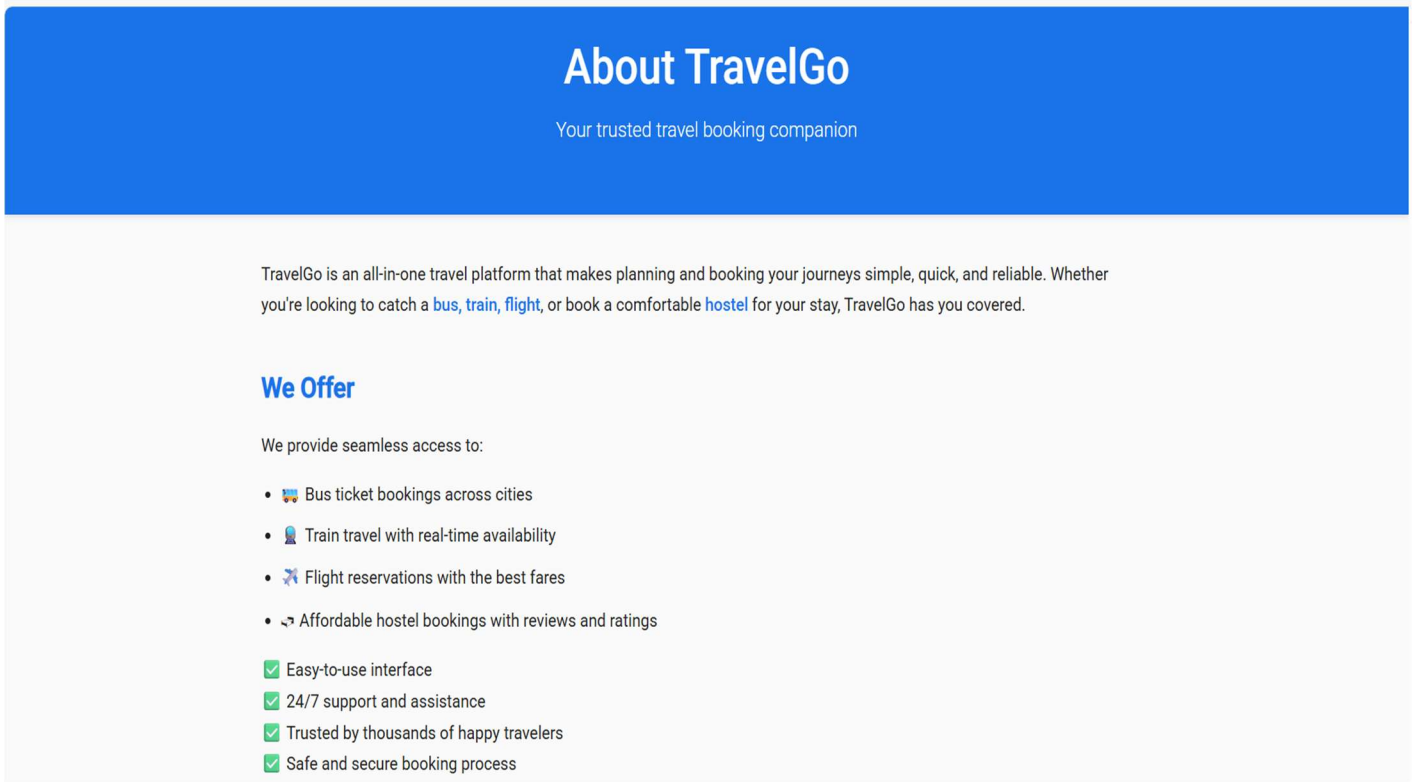
Support page:



aboutus code:



about us 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.