

Name : ALLAM AJAY KUMAR

EMAIL : 238x5a0516@khitguntur.ac.in

COLLEGE : KALLAM HARANADHREDDY INSTITUTE OF TECHNOLOGY

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.

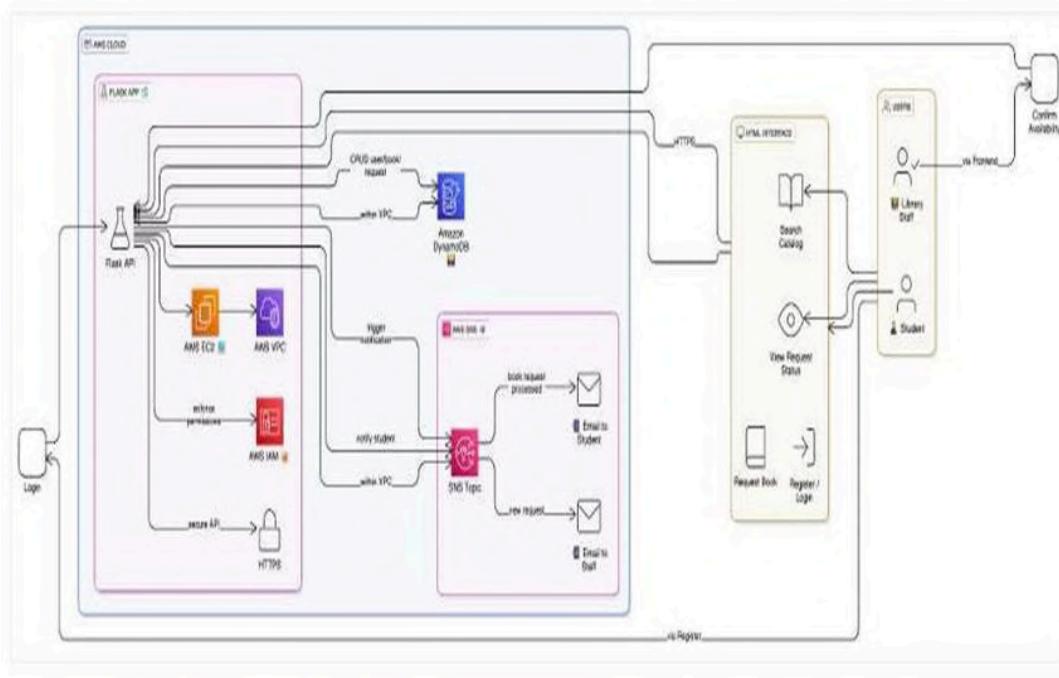
Scenario 2: Real-Time Booking Confirmation with AWS SNS

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.

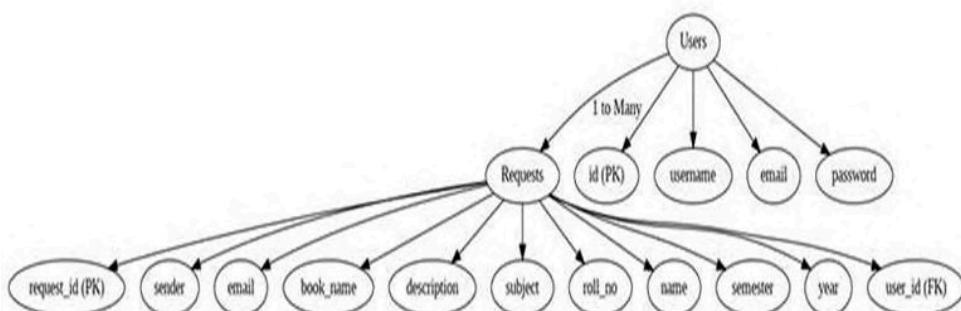
Scenario 3: Dynamic Dashboard with Personal TravelHistory

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

The screenshot shows the troven platform interface for setting up a lab. On the left, there's a sidebar with a blue background and a user profile icon labeled 'GINJUPALLI' and '226x105@lakshmi@iitg.ac.in'. The main content area has a white background with a header: '← AWS- TravelGo: A Cloud-Powered Real-Time Travel Booking Platform Using AWS' and a back arrow. Below the header is a table with columns: Platform (AWS), Lab (1), Duration (2 hour(s) 0 minute(s)), Difficulty (Expert), and Progress (AWS). The 'Overview' section contains a brief description of TravelGo as a cloud-based travel booking platform built with Flask and deployed on EC2, using DynamoDB for storage and AWS SNS for notifications. The 'Skills' section lists EC2, IAM, Database on AWS, and Monitoring and Observability. The 'Lab' section is titled 'AWS Final Deployment' and shows 'AWS- TravelGo: A Cloud-Powered Real-Time Travel Booking Platform Using AWS'. It indicates 3/5 Tasks Validated and shows an 'Attempted' button. Below this, it shows 'easy' difficulty, 5 tasks, and a duration of 180 mins.

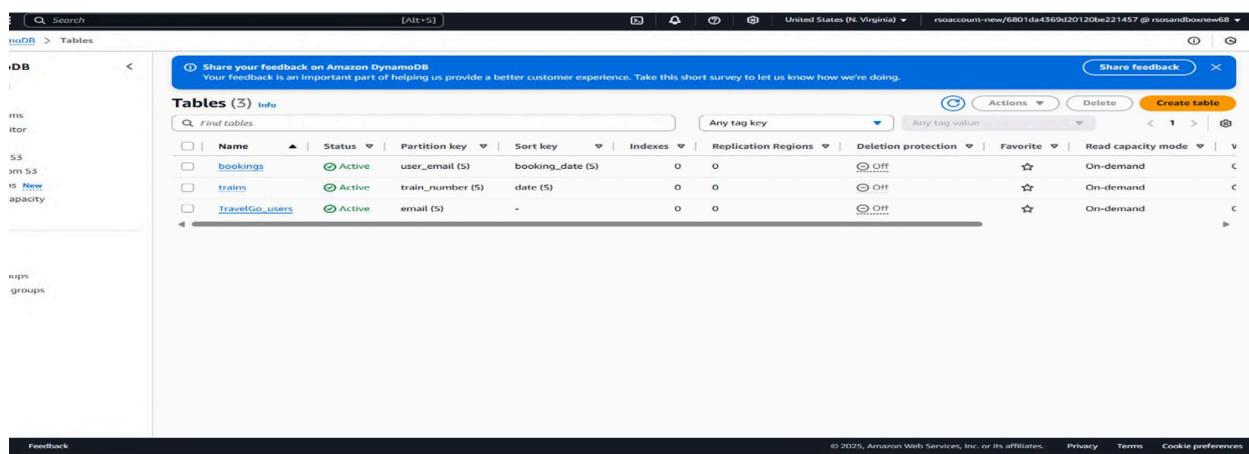
Activity 1.2: Open the AWS Management Console

The screenshot shows the AWS Management Console within a browser window. The title bar reads 'AWS- TravelGo: A Cloud-Powered Real-Time Travel Booking Platform Using AWS'. The main content area is titled 'AWS Final Deployment' and shows 'AWS- TravelGo: A Cloud-Powered Real-Time Travel Booking Platform Using AWS'. It displays 'Allotted Duration: 02:00:00 HRS' and 'Duration Left: 01:12:31 HRS'. A table provides details for the total task: 'Total Task: 5', 'Recommended Duration: 03:00:00 HRS', 'Current Lab Duration: 02:12:31 HRS', and 'Validation Status: 0/5'. A note says 'The region should be set to N Virginia.' Below this, there are buttons for 'Initiate Labs', 'Open Console', and 'End Lab'. A message at the bottom says 'Once you complete a task you can click on Validate and check if the task is done correctly. Click Validate All to check the status all at once.' A sidebar on the left lists four steps: 'Launch an EC2 Instance (TravelGo)', 'Create IAM Role (TravelGo)', 'Setup DynamoDB (TravelGo)', and 'Configure SNS (TravelGo)', each with a 'Validate' button. The browser taskbar shows various open tabs and system icons.

2. DynamoDB Database Creation and Setup

Activity 2.1: Create a DynamoDB Table

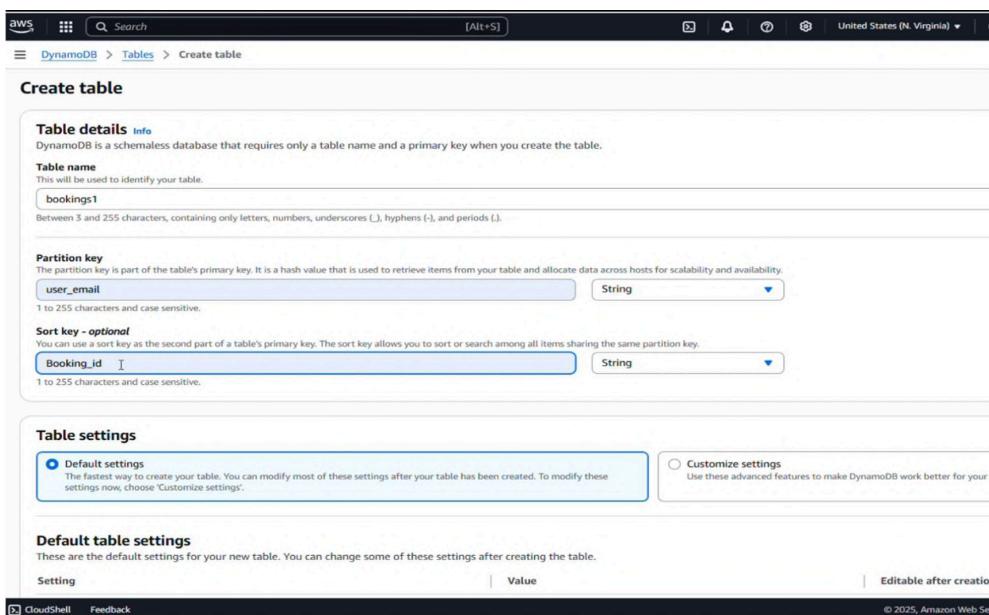
To create a DynamoDB table, go to the AWS Management Console, navigate to **DynamoDB**, and click "**Create table**." Specify a **table name** and define a **primary key** (e.g., Email as Partition Key). Choose additional settings like **read/write capacity mode** (on-demand or provisioned) and click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



The screenshot shows the AWS DynamoDB management console. On the left, there's a sidebar with navigation links like 'Tables', 'AWS Lambda', 'Amazon S3', 'Amazon Kinesis', 'Amazon RDS', and 'Amazon CloudWatch'. The main area is titled 'Tables (3) Info' and lists three tables:

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity mode
bookings	Active	user_email (\$)	booking_date (\$)	0	0	Off	☆	On-demand
trains	Active	train_number (\$)	date (\$)	0	0	Off	☆	On-demand
TravelGo_users	Active	email (\$)	-	0	0	Off	☆	On-demand

Activity 2.2: Configure Attributes for User Data and Booking Requests



The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The steps are:

- Table details**:
 - Table name**: bookings1
 - Partition key**: user_email (String)
 - Description: 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.
 - Setting: user_email
 - Description: 1 to 255 characters and case sensitive.
 - Sort key - optional**: Booking_id (String)
 - Description: 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.
 - Setting: Booking_id
 - Description: 1 to 255 characters and case sensitive.
- Table settings**:
 - Default settings**:
 - Description: 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'.
 - Setting: None
 - Description: These are the default settings for your new table. You can change some of these settings after creating the table.
 - Customize settings**:
 - Description: Use these advanced features to make DynamoDB work better for your application.
 - Setting: None
- Default table settings**:
 - Setting: None
 - Description: These are the default settings for your new table. You can change some of these settings after creating the table.
- Editable after creation**:
 - Setting: None
 - Description: These settings can be modified after the table is created.

3. SNS Notification Setup

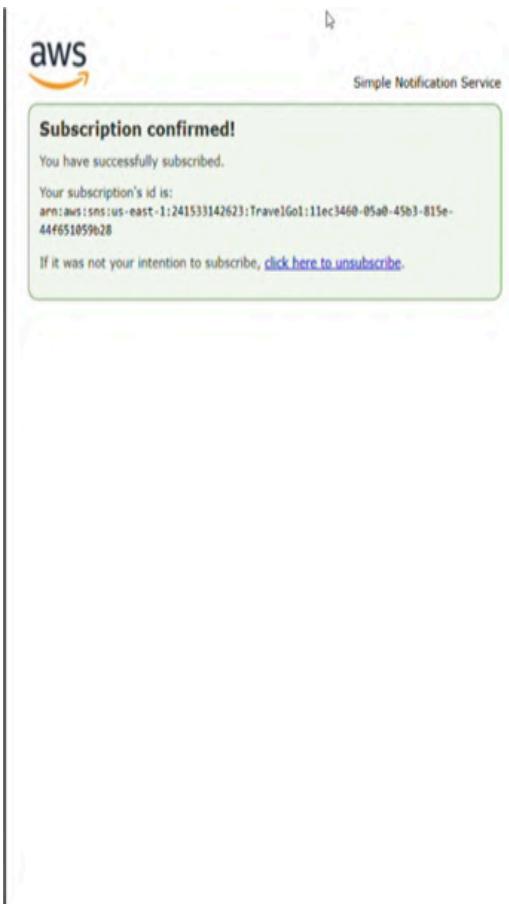
For **SNS Notification Setup**, go to the AWS Console and open the **Simple Notification Service (SNS)** dashboard. Click “**Create topic**”, choose the **Standard** type, and name your topic. After creating it, copy the **Topic ARN**. Then, **subscribe email addresses** (students/admins) to the topic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

Activity 3.1: Create SNS topics.

The screenshot shows the AWS SNS Topics page. A new topic named "TravelGo" has been created. The "Details" section shows the Name as "TravelGo", ARN as "arnaws:sns:us-east-1:418272775181:TravelGo", and Type as "Standard". The "Subscriptions" tab is selected, showing one confirmed subscription with the ID "b7d6f5bc-7710-49de-97bc-ddecff5fd023" and endpoint "228x1a05e1@khitguntur.ac.in". Other tabs include Access policy, Data protection policy, Delivery policy (HTTP/S), Delivery status logging, Encryption, Tags, and Integrations.

Activity 3.2: Subscribe users to SNS email notifications

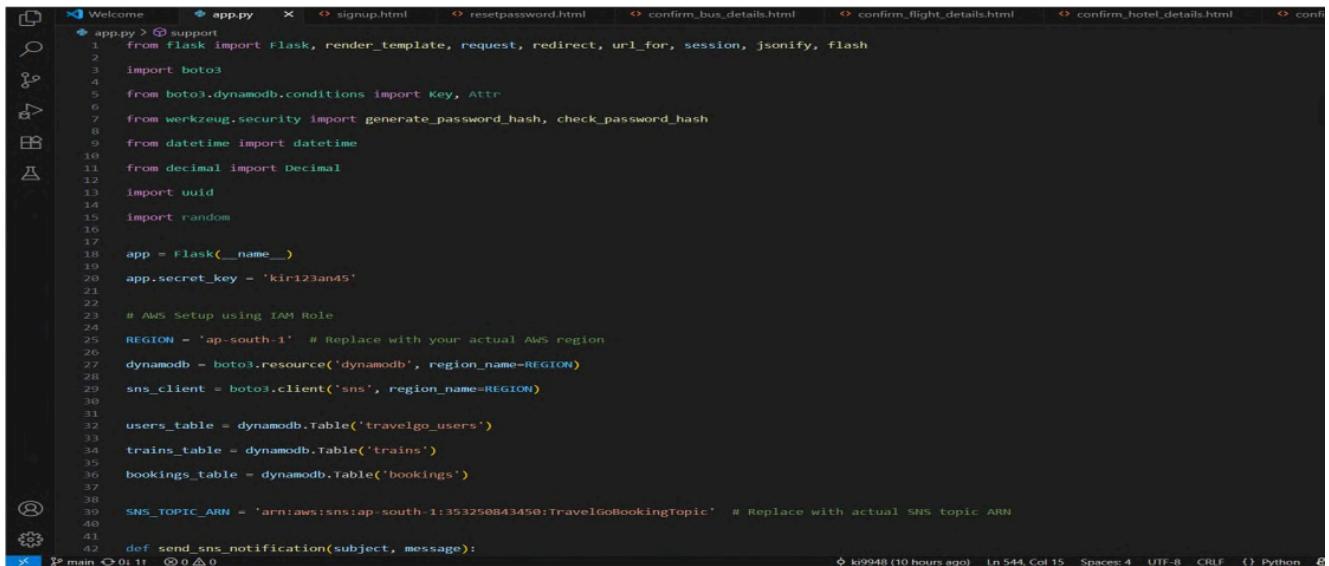
The screenshot shows the "Create subscription" page. It is configured to subscribe the topic ARN "arnaws:sns:us-east-1:418272775181:TravelGo" to an "Email" endpoint at "228x1a05e1@khitguntur.ac.in". The "Subscription filter policy - optional" and "Redrive policy (dead-letter queue) - optional" sections are present but empty. At the bottom right are "Cancel" and "Create subscription" buttons.



4. Backend Development and Application Setup

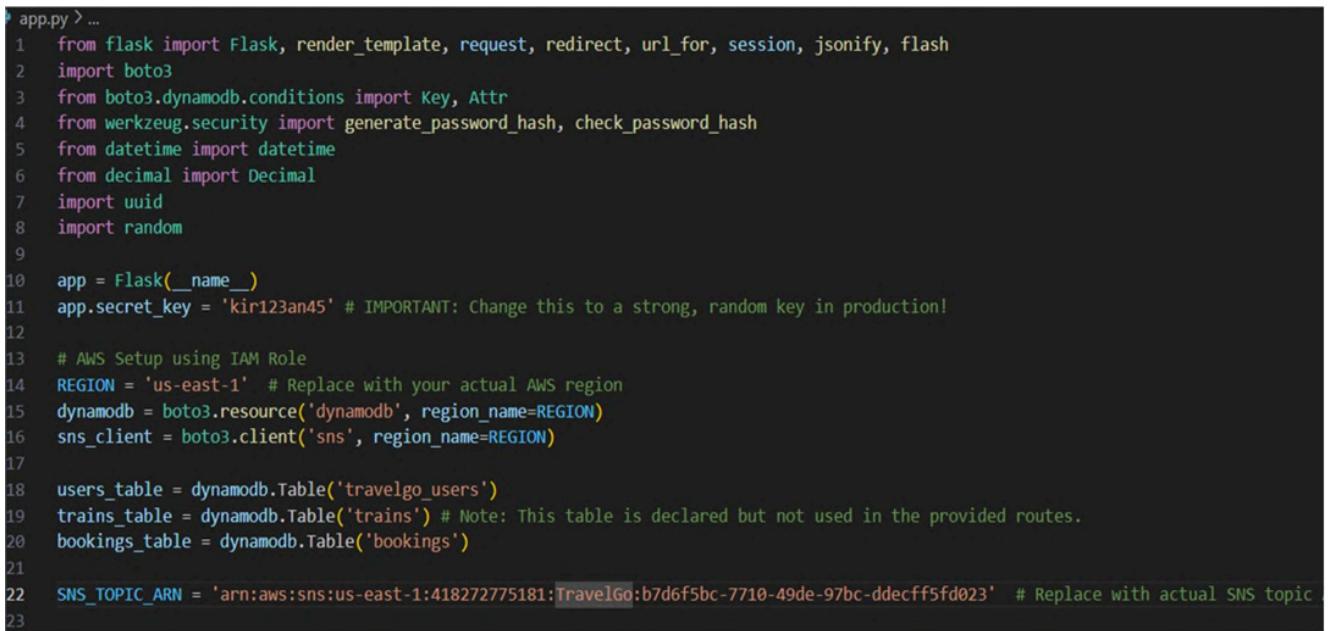
In the Backend Development and Application Setup, use Flask to build the web application and integrate AWS services via the boto3 library. Set up routes for user registration, login, book requests, and page navigation. Connect to DynamoDB to store user and request data, and configure AWS SNS to send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

Activity 4.1: Develop the Backend Using Flask.



```
app.py > ...  
1 from flask import Flask, render_template, request, redirect, url_for, session, jsonify, flash  
2 import boto3  
3 from boto3.dynamodb.conditions import Key, Attr  
4 from werkzeug.security import generate_password_hash, check_password_hash  
5 from datetime import datetime  
6 from decimal import Decimal  
7 import uuid  
8 import random  
9  
10 app = Flask(__name__)  
11 app.secret_key = 'kir123an45'  
12  
13 # AWS Setup using IAM Role  
14 REGION = 'ap-south-1' # Replace with your actual AWS region  
15 dynamodb = boto3.resource('dynamodb', region_name=REGION)  
16 sns_client = boto3.client('sns', region_name=REGION)  
17  
18 users_table = dynamodb.Table('travelgo_users')  
19 trains_table = dynamodb.Table('trains')  
20 bookings_table = dynamodb.Table('bookings')  
21  
22 SNS_TOPIC_ARN = 'arn:aws:sns:ap-south-1:353250843450:TravelGoBookingTopic' # Replace with actual SNS topic ARN  
23  
24 def send sns_notification(subject, message):  
25     pass
```

Activity 4.2: Integrate AWS Services Using boto3



```
app.py > ...  
1 from flask import Flask, render_template, request, redirect, url_for, session, jsonify, flash  
2 import boto3  
3 from boto3.dynamodb.conditions import Key, Attr  
4 from werkzeug.security import generate_password_hash, check_password_hash  
5 from datetime import datetime  
6 from decimal import Decimal  
7 import uuid  
8 import random  
9  
10 app = Flask(__name__)  
11 app.secret_key = 'kir123an45' # IMPORTANT: Change this to a strong, random key in production!  
12  
13 # AWS Setup using IAM Role  
14 REGION = 'us-east-1' # Replace with your actual AWS region  
15 dynamodb = boto3.resource('dynamodb', region_name=REGION)  
16 sns_client = boto3.client('sns', region_name=REGION)  
17  
18 users_table = dynamodb.Table('travelgo_users')  
19 trains_table = dynamodb.Table('trains') # Note: This table is declared but not used in the provided routes.  
20 bookings_table = dynamodb.Table('bookings')  
21  
22 SNS_TOPIC_ARN = 'arn:aws:sns:us-east-1:418272775181:TravelGo:b7d6f5bc-7710-49de-97bc-ddecff5fd023' # Replace with actual SNS topic  
23
```

C for SNS

```
# 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

```
@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

```
86
87 @app.route('/dashboard')
88 def dashboard():
89     if 'email' not in session:
90         return redirect(url_for('login'))
91     user_email = session['email']
92
93     # Query bookings for the logged-in user using the primary key 'user_email'
94     # No GSI is needed here as 'user_email' is likely the partition key for the bookings_table.
95     response = bookings_table.query(
96         KeyConditionExpression=Key('user_email').eq(user_email),
97         ScanIndexForward=False # Get most recent bookings first
98     )
99     bookings = response.get('Items', [])
100
101    # Convert Decimal types from DynamoDB to float for display if necessary
102    for booking in bookings:
103        if 'total_price' in booking:
104            try:
105                booking['total_price'] = float(booking['total_price'])
106            except (TypeError, ValueError):
107                booking['total_price'] = 0.0 # Default value if conversion fails
108
109    return render_template('dashboard.html', username=user_email, bookings=bookings)
```

Code for Cancel booking

```
1
2 @app.route('/cancel_booking', methods=['POST'])
3 def cancel_booking():
4     if 'email' not in session:
5         return redirect(url_for('login'))
6
7     booking_id = request.form.get('booking_id')
8     user_email = session['email']
9     booking_date = request.form.get('booking_date') # This is crucial as it's the sort key
10
11     if not booking_id or not booking_date:
12         flash("Error: Booking ID or Booking Date is missing for cancellation.", 'error')
13         return redirect(url_for('dashboard'))
14
15     try:
16         # Delete item using the primary key (user_email and booking_date)
17         # This does not use GSI, so it remains unchanged.
18         bookings_table.delete_item(
19             Key={'user_email': user_email, 'booking_date': booking_date}
20         )
21         flash(f"Booking {booking_id} cancelled successfully!", 'success')
22     except Exception as e:
23         flash(f"Failed to cancel booking {booking_id}: {str(e)}", 'error')
24
25     return redirect(url_for('dashboard'))
26
27
28 if __name__ == '__main__':
29     # IMPORTANT: In a production environment, disable debug mode and specify a production-ready host.
30     app.run(debug=True, host='0.0.0.0')
```

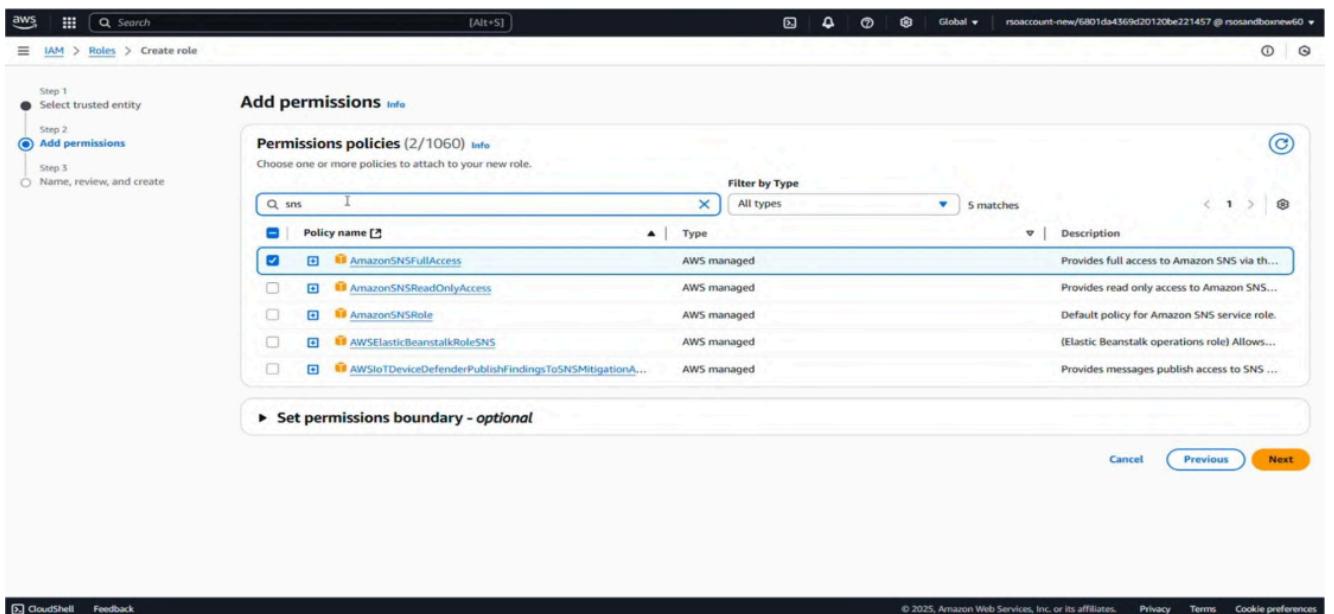
5. IAM Role Setup

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

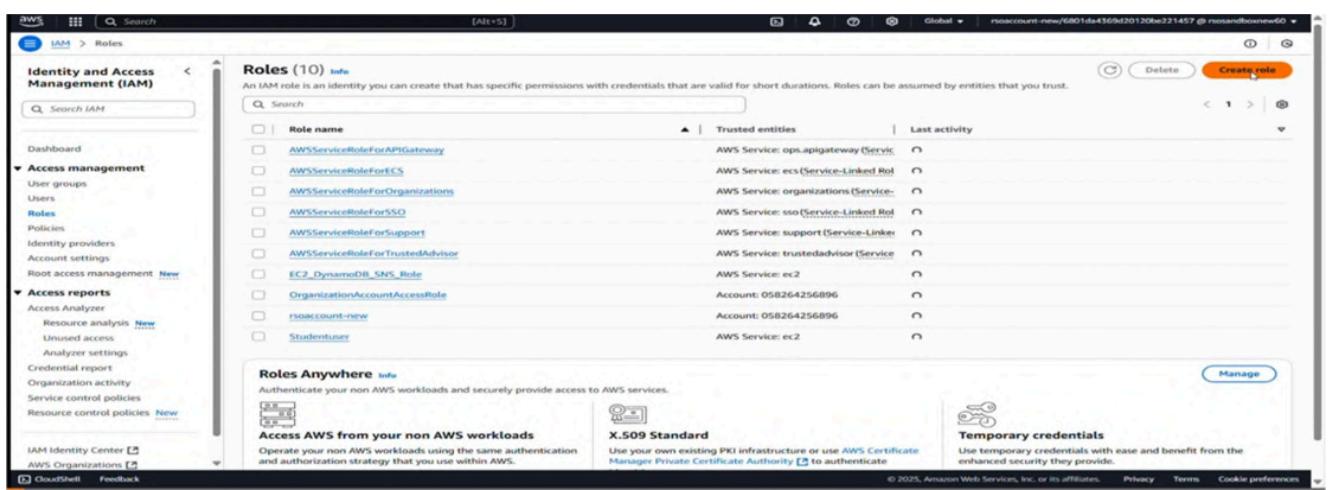
Attach AmazonDynamoDBFullAccess and AmazonSNSFullAccess policies to allow interaction with DynamoDB and SNS. Assign this role to your EC2 instance during or after launch. This ensures secure and authorized access to AWS resources from your Flask backend.

Activity 5.1: Create IAM Role

Activity 5.2: Attach Policies



The screenshot shows the 'Add permissions' step of creating a new IAM role. On the left, a sidebar lists steps: Step 1 (Select trusted entity), Step 2 (Add permissions, which is selected), and Step 3 (Name, review, and create). The main area is titled 'Add permissions' and shows a search bar with 'sns' and a list of 'Permissions policies (2/1060)'. A checked checkbox next to 'AmazonSNSFullAccess' indicates it is selected. Other policies listed include 'AmazonSNSReadOnlyAccess', 'AmazonSNSRole', 'AWSelasticBeanstalkRoleSNS', and 'AWSIoTDeviceDefenderPublishFindingsToSNSSignatureVerificationRole'. Below the list is a section titled 'Set permissions boundary - optional' with 'Cancel', 'Previous', and 'Next' buttons.



The screenshot shows the 'Roles' page in the AWS IAM console. The sidebar includes sections for Identity and Access Management (IAM), Access management, Access reports, and more. The main area displays a table of 10 roles, each with a 'Role name', 'Trusted entities', and 'Last activity'. Roles listed include 'AWSLambdaRoleForAPIGateway', 'AWSLambdaRoleForECS', 'AWSLambdaRoleForOrganizations', 'AWSLambdaRoleForSSO', 'AWSLambdaRoleForSupport', 'AWSLambdaRoleForTrustedAdvisor', 'EC2_DynamoDB_SNS_Role', 'OrganizationAccountAccessRole', 'rsoaccount-new', and 'Studentuser'. Below the table are sections for 'Roles Anywhere', 'Access AWS from your non AWS workloads', and 'Temporary credentials'.

The screenshot shows the 'Launch an instance' wizard in the AWS Management Console. The current step is 'Name and tags'. A blue box highlights the 'Name' field, which contains 'TravelGo1'. To the right, there's a 'Add additional tags' button. Below this, the 'Application and OS Images (Amazon Machine Image)' section is shown, featuring a search bar and a list of AMIs. The 'Instance type' section follows, with a dropdown menu set to 'Select' and a 'Compare instance types' link. The 'Key pair (login)' section is also visible. On the right side, a summary panel shows '1' instance selected, and a detailed 'Free tier' information box is open, stating: 'In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where t2.micro isn't available) when used with free tier AMIs, 750 hours per month of public IPv4 address usage, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.' At the bottom right are 'Cancel', 'Launch instance', and 'Preview code' buttons.

create a key pair:

The screenshot shows the 'Create key pair' dialog box overlaid on the EC2 instance launch wizard. The dialog has a title 'Create key pair' and a 'Key pair name' input field containing 'travel119'. Below it is a note: 'The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.' The 'Key pair type' section shows 'RSA' (selected) and 'ED25519' as options. The 'Private key file format' section offers '.pem' (selected) and '.ppk' as choices. A note below says: 'When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance.' At the bottom are 'Cancel' and 'Create key pair' buttons. The background shows the 'Summary' section of the instance launch wizard, which includes fields for 'Software Image (AMI)', 'Virtual server type (instance type)', 'Firewall (security group)', and 'Storage (volumes)'. A 'Free tier' information box is also present. The bottom right of the dialog has 'Launch instance' and 'Preview code' buttons.

Edit of Security group:

The screenshot shows the 'Edit inbound rules' section of the AWS EC2 console. It lists three security group rule entries:

Type	Protocol	Port range	Source	Description - optional
SSH	TCP	22	Custom	0.0.0.0/0
HTTPS	TCP	443	Custom	0.0.0.0/0
Custom TCP	TCP	5000	Anywhere	0.0.0.0/0

Buttons for 'Add rule' and 'Delete' are available for each entry. A note at the bottom left says: '⚠ Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.' At the bottom right are 'Cancel', 'Preview changes', and 'Save rules' buttons.

The screenshot shows the 'Instances' page in the AWS EC2 console. It displays a list of three instances:

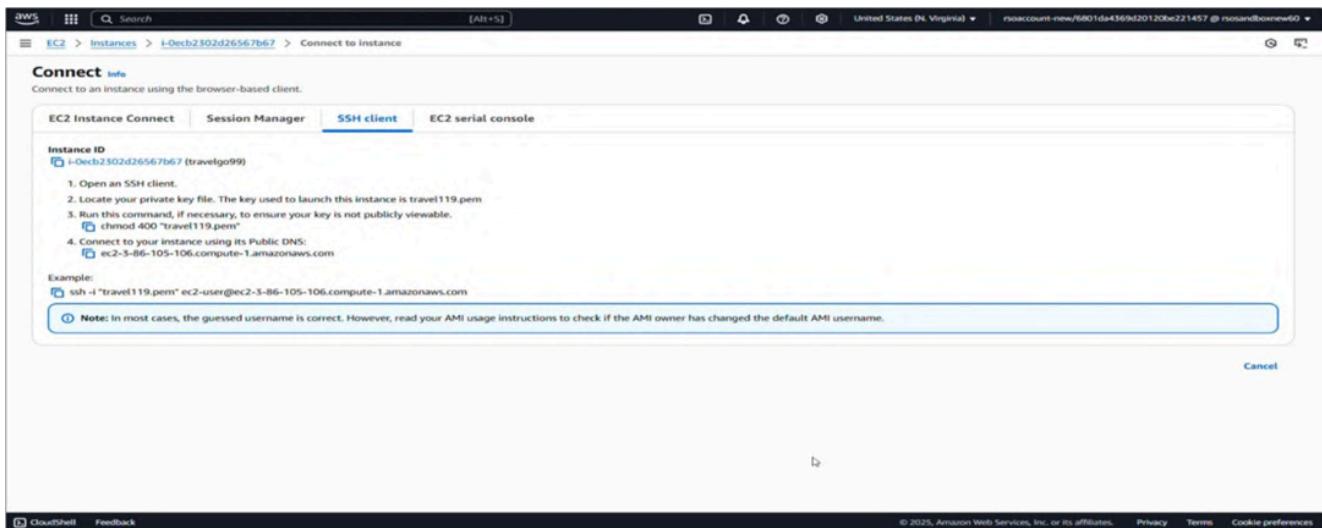
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Avg. CPU usage
travelgo99	i-0ecb2302d26567b67	Running	t2.micro	Initializing	View alarms +	1%
TravelGo	i-0e5af7d5c05f2398b	Running	t2.micro	2/2 checks	Change security groups	1%
TravelGo1	i-04aea784c914b3b3	Running	t2.micro	2/2 checks	Get Windows password	1%

For the selected instance 'travelgo99' (i-0ecb2302d26567b67), the 'Security' tab is active. It shows the IAM Role (empty) and the Security groups assigned: 'sg-0a6c8b20300891292 (launch-wizard-3)'. The 'Inbound rules' section lists two 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

At the bottom right of the page are 'Launch time' and 'Launch time' details: 'Tue Jul 01 2025 12:24:05 GMT+0530 (India Standard Time)'.

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 --version
git --version

Activity 7.2:Clone Your Flask Project from GitHub

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

 testupload Public

main 1 Branch 0 Tags Go to file Add file Code About

ki9948 Update app.py

static Added full project fi
templates Added full project fi
venv Added full project fi
venv_new Added full project fi
README.md first commit
app.py Update app.py

Local Codespaces

Clone HTTPS SSH GitHub CLI https://github.com/ki9948/testupload.git Clone using the web URL.

Open with GitHub Desktop Download ZIP

No description, website, or topics provided.

Readme Activity 0 stars 0 watching 0 forks

Releases No releases published Create a new release

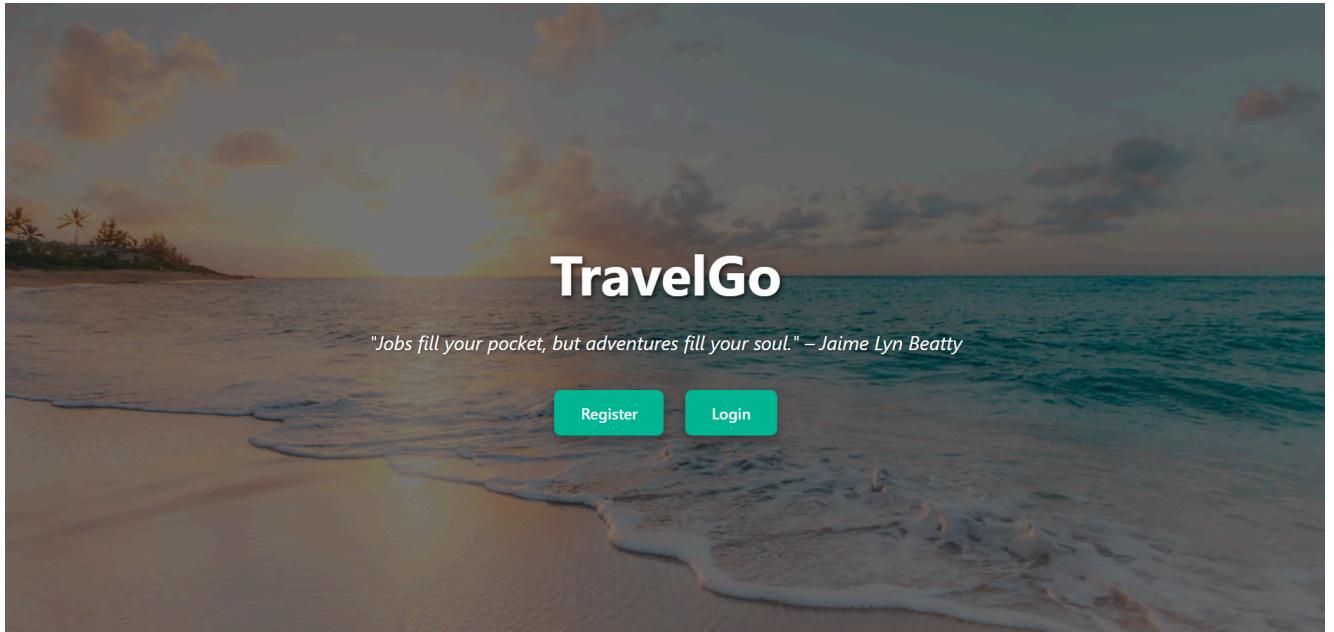
Packages

```

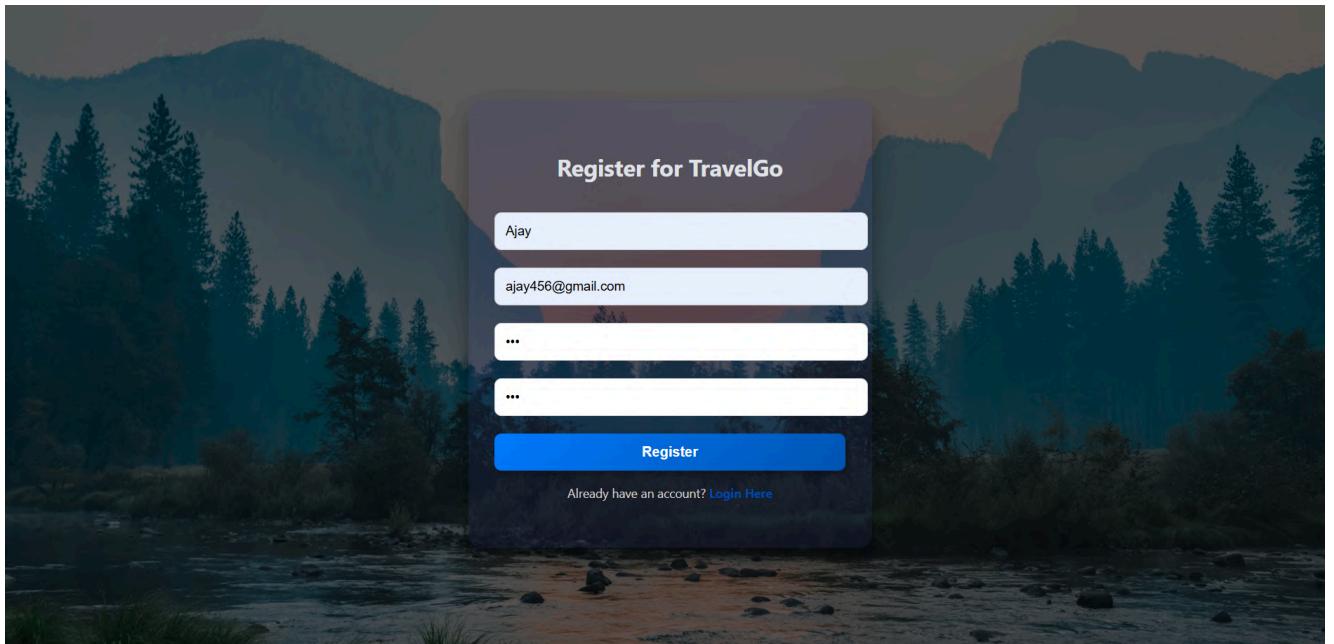
Collecting itsdangerous>=2.2.0
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting blinker<=1.9.0
  Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Collecting werkzeug>=3.1.0
  Downloading werkzeug-3.1.3-py3-none-any.whl (224 kB) | 224 kB 56.0 MB/s
Collecting zipp>=3.20
  Downloading 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 writable
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
127.0.0.1 - - [03/01/2025 05:40:24] "GET / HTTP/1.1" 200

```

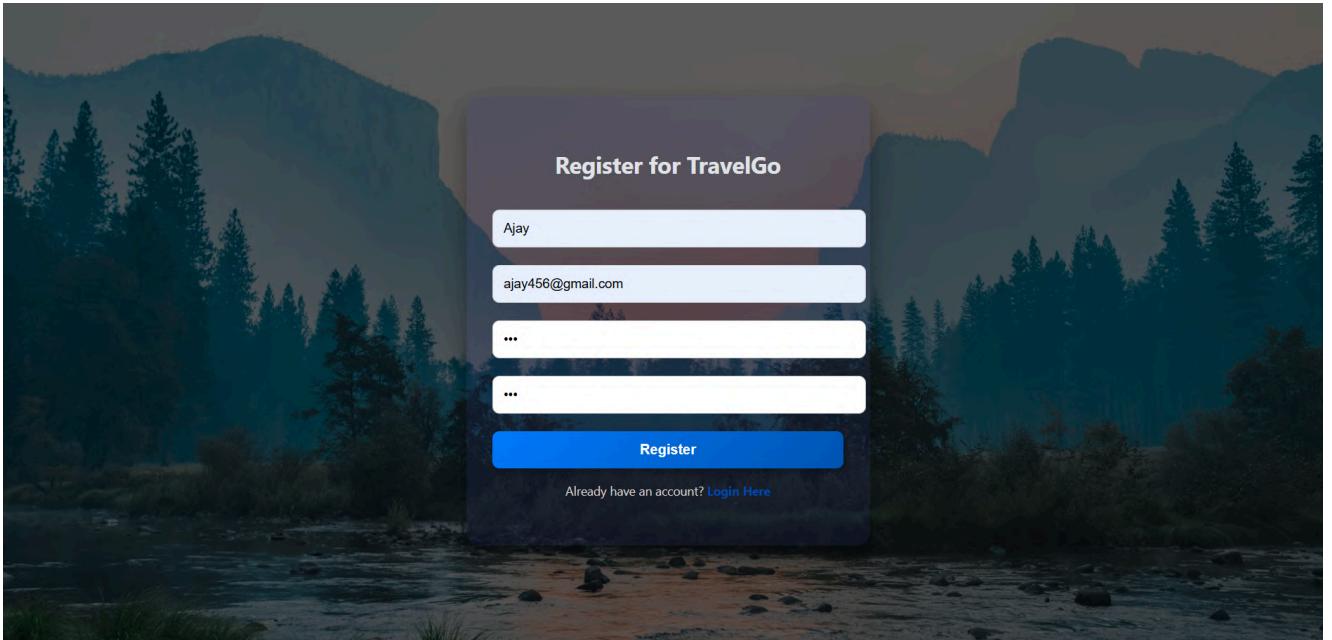
Index Page:



Register Page :



Login Page :



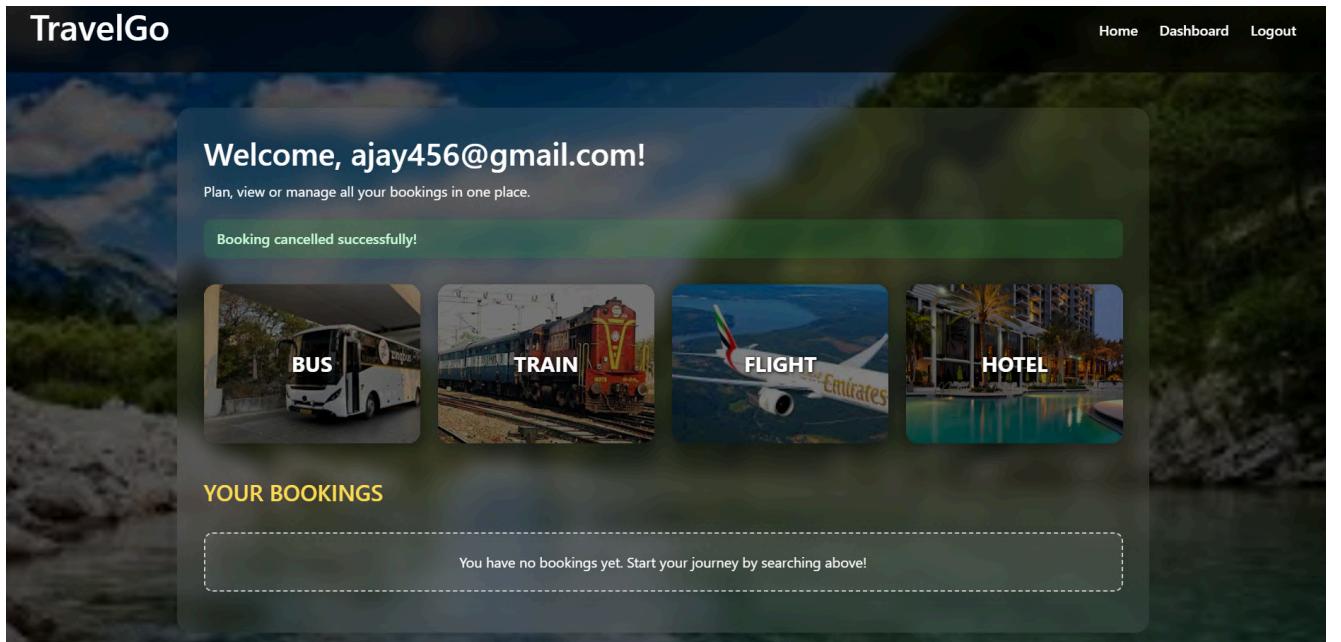
Home Code:

```
 1 Welcome   app.py M   home.html X
2 templates > home.html > html > head > style
3   <html lang="en">
4     <head>
5       <style>
6         .dashboard-btn:hover {
7           background-color: #f0c040;
8           color: #000;
9         }
10
11         .content {
12           height: 100%;
13           display: flex;
14           justify-content: center;
15           align-items: center;
16           text-align: center;
17           color: white;
18           padding: 20px;
19         }
20
21         .content h1 {
22           font-size: 60px;
23           margin-bottom: 20px;
24         }
25
26         .content p {
27           font-size: 24px;
28         }
29     </style>
30   </head>
31   <body>
32     <div class="overlay">
33       <a href="/dashboard" class="dashboard-btn">Dashboard</a>
34       <div class="content">
35         <div>
36           <h1>Welcome to TravelGo</h1>
37           <p>Start Booking your Ticket and Safe Journey!</p>
38         </div>
39       </div>
40     </div>
41   </body>
42 </html>
```

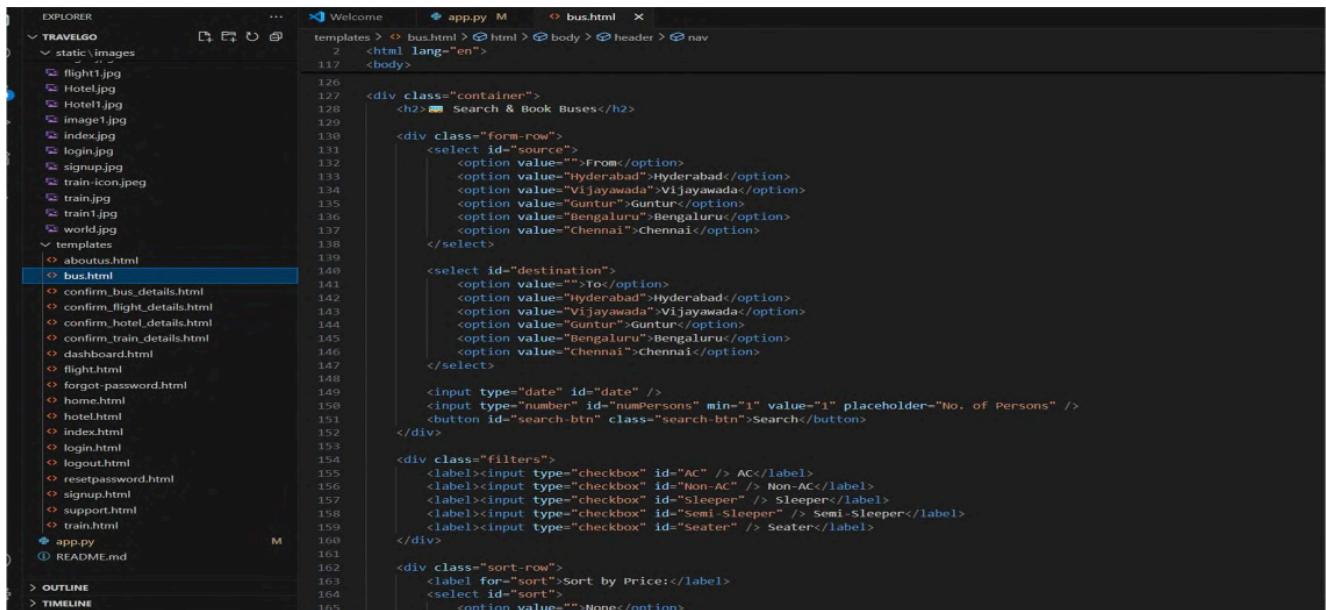
Dashboard Code :

```
 2 <html>
 3   <body>
 4     <div class="dashboard-container">
 5       <div class="booking-item">
 6         <div class="booking-actions">
 7           <form>
 8             <button type="button" class="cancel-btn">Cancel</button>
 9           </form>
10         </div>
11       </div>
12     </div>
13   </body>
14 </html>
```

Dashboard Page :



Bus Booking Code :

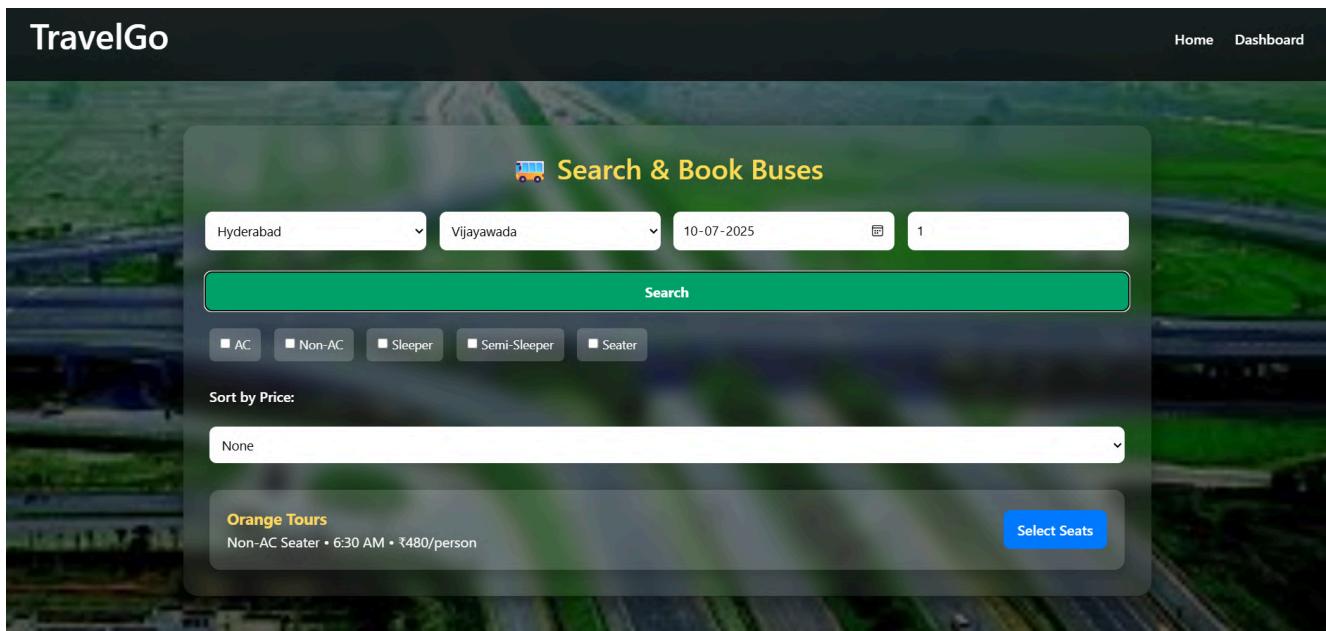


The screenshot shows a code editor with the file `bus.html` open. The code is an HTML template for a bus booking search page. It includes fields for source and destination locations, a date input, a person count input, and a search button. Below these are checkboxes for different seat types: AC, Non-AC, Sleeper, Semi-Sleeper, and Seater. A dropdown menu for sorting by price is also present. The code uses Bootstrap classes like `form-row`, `filters`, and `sort-row`.

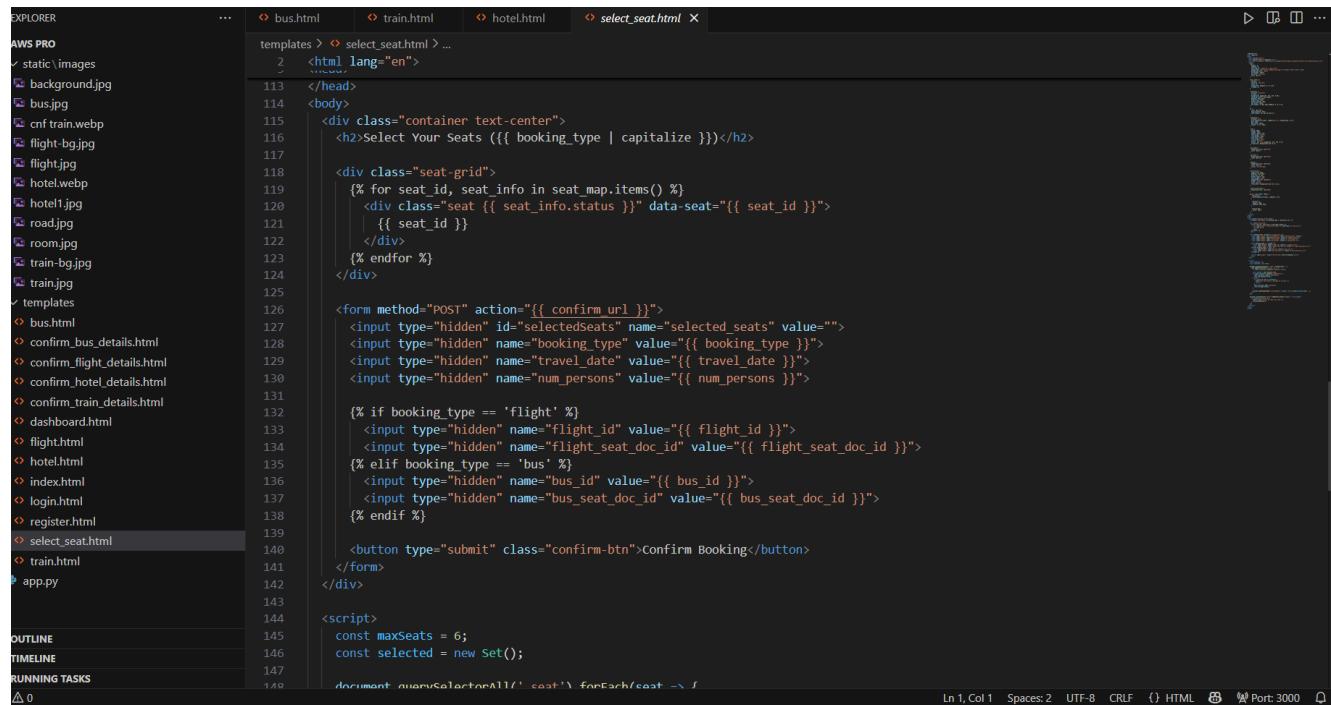
```
<div class="container">
    <h2> Search & Book Buses</h2>
    <div class="form-row">
        <select id="source">
            <option value="">From</option>
            <option value="Hyderabad">Hyderabad</option>
            <option value="Vijayawada">Vijayawada</option>
            <option value="Guntur">Guntur</option>
            <option value="Bengaluru">Bengaluru</option>
            <option value="Chennai">Chennai</option>
        </select>
        <select id="destination">
            <option value="">To</option>
            <option value="Hyderabad">Hyderabad</option>
            <option value="Vijayawada">Vijayawada</option>
            <option value="Guntur">Guntur</option>
            <option value="Bengaluru">Bengaluru</option>
            <option value="Chennai">Chennai</option>
        </select>
        <input type="date" id="date" />
        <input type="number" id="numPersons" min="1" value="1" placeholder="No. of Persons" />
        <button id="search-btn" class="search-btn">Search</button>
    </div>
    <div class="filters">
        <label><input type="checkbox" id="AC" /> AC</label>
        <label><input type="checkbox" id="Non-AC" /> Non-AC</label>
        <label><input type="checkbox" id="Sleeper" /> Sleeper</label>
        <label><input type="checkbox" id="Semi-Sleeper" /> Semi-Sleeper</label>
        <label><input type="checkbox" id="Seater" /> Seater</label>
    </div>
    <div class="sort-row">
        <label>Sort by Price:</label>
        <select id="sort">
            <option value="">None</option>
        </select>
    </div>

```

Bus Booking Page :



Selecting Seat Code :

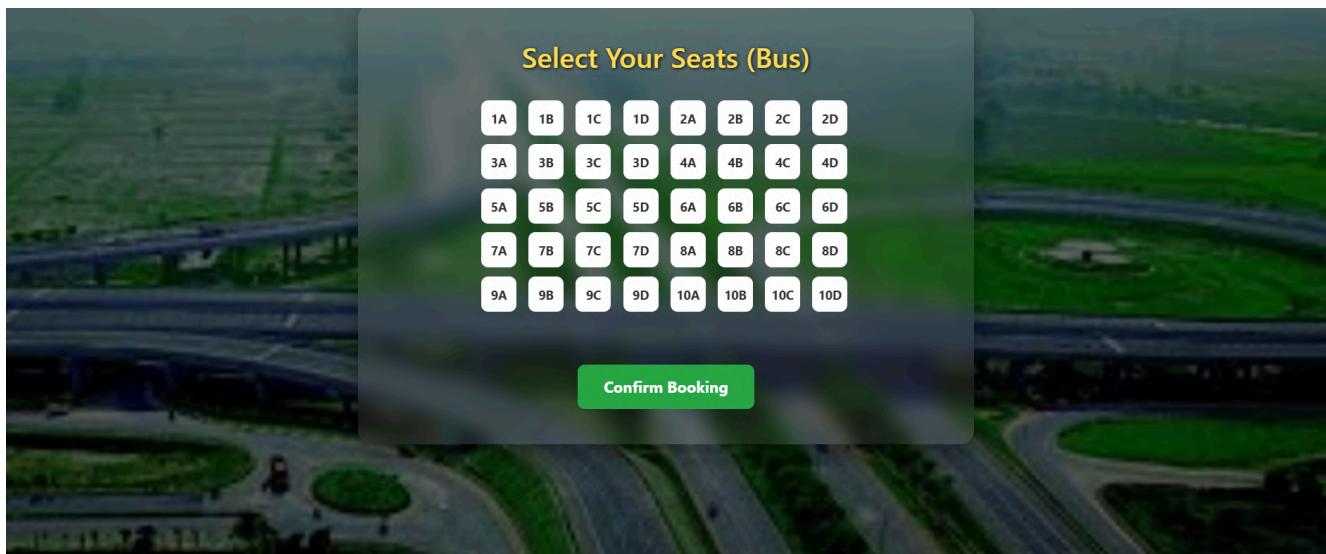


```
EXPLORER ... bus.html train.html hotel.html select_seat.html ...
AWS PRO
✓ static\images
  background.jpg
  bus.jpg
  cmt_train.webp
  flight-bg.jpg
  flight.jpg
  hotel.webp
  hotel.jpg
  road.jpg
  room.jpg
  train-bg.jpg
  train.jpg
✓ templates
  bus.html
  confirm_bus_details.html
  confirm_flight_details.html
  confirm_hotel_details.html
  confirm_train_details.html
  dashboard.html
  flight.html
  hotel.html
  index.html
  login.html
  register.html
  select_seat.html
  train.html
  app.py
OUTLINE
TIMELINE
RUNNING TASKS
  △ 0
```

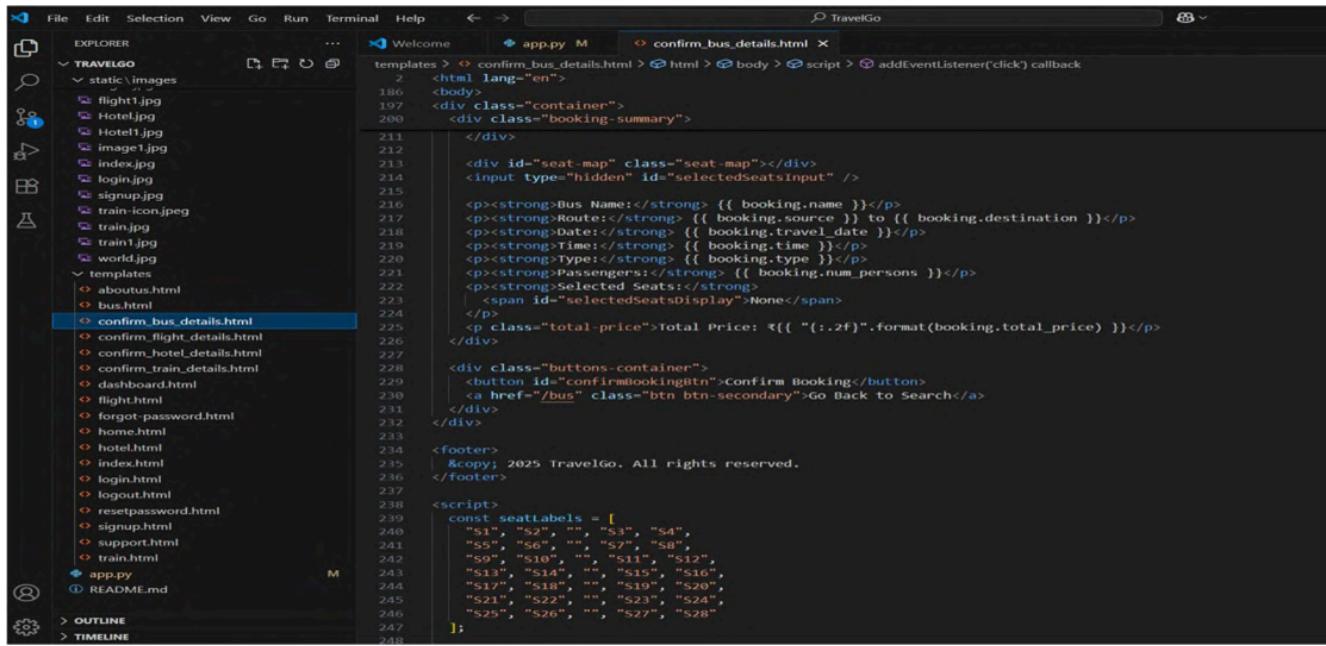
```
113 </head>
114 <body>
115   <div class="container text-center">
116     <h2>Select Your Seats ({{ booking_type | capitalize }})</h2>
117
118   <div class="seat-grid">
119     {% for seat_id, seat_info in seat_map.items() %}
120       <div class="seat {{ seat_info.status }}" data-seat="{{ seat_id }}>
121         {{ seat_id }}
122       </div>
123     {% endfor %}
124   </div>
125
126   <form method="POST" action="{{ confirm_url }}>
127     <input type="hidden" id="selectedSeats" name="selected_seats" value="">
128     <input type="hidden" name="booking_type" value="{{ booking_type }}>
129     <input type="hidden" name="travel_date" value="{{ travel_date }}>
130     <input type="hidden" name="num_persons" value="{{ num_persons }}>
131
132     {% if booking_type == 'flight' %}
133       <input type="hidden" name="flight_id" value="{{ flight_id }}>
134       <input type="hidden" name="flight_seat_doc_id" value="{{ flight_seat_doc_id }}>
135     {% elif booking_type == 'bus' %}
136       <input type="hidden" name="bus_id" value="{{ bus_id }}>
137       <input type="hidden" name="bus_seat_doc_id" value="{{ bus_seat_doc_id }}>
138     {% endif %}
139
140     <button type="submit" class="confirm-btn">Confirm Booking</button>
141   </form>
142
143 <script>
144   const maxSeats = 6;
145   const selected = new Set();
146
147   document.querySelectorAll('seat').forEach(seat => {
148     seat.addEventListener('click', () => {
149       if (!selected.has(seat.id)) {
150         selected.add(seat.id);
151         seat.classList.add('selected');
152       } else {
153         selected.delete(seat.id);
154         seat.classList.remove('selected');
155       }
156     });
157   });
158
159 </script>
```

Ln 1, Col 1 Spaces: 2 UTF-8 CRLF {} HTML 🌐 Port: 3000

Selecting Seat Page :

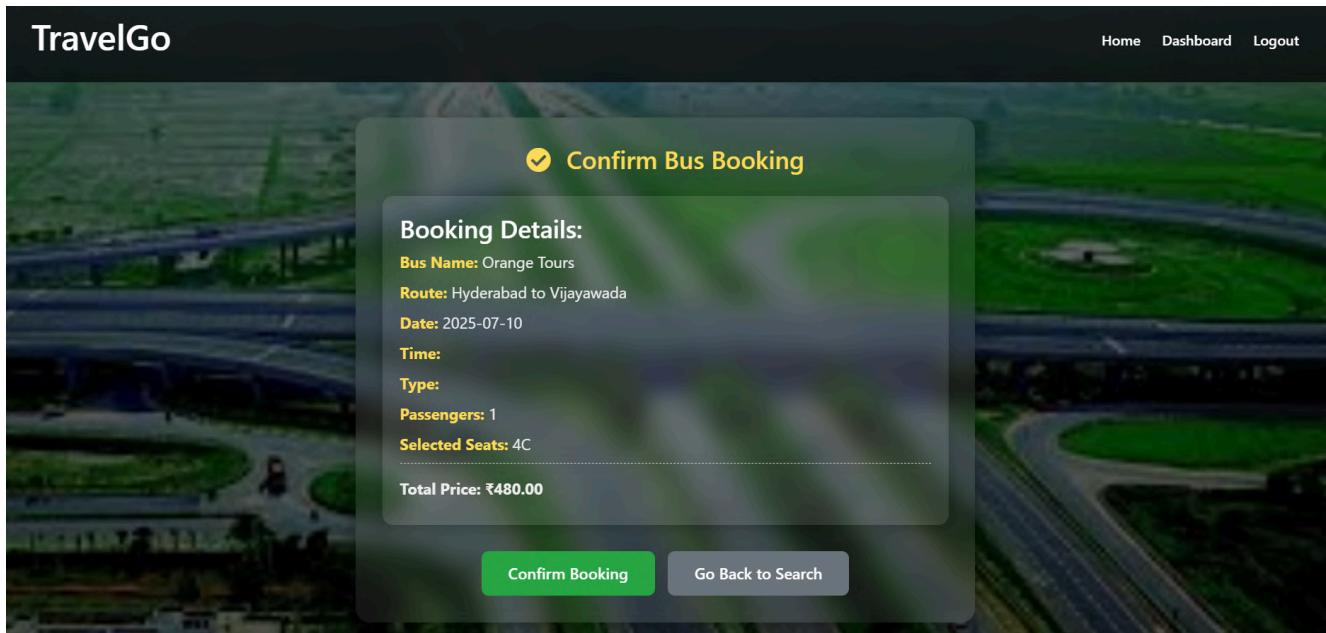


Bus Confirm Booking Code :

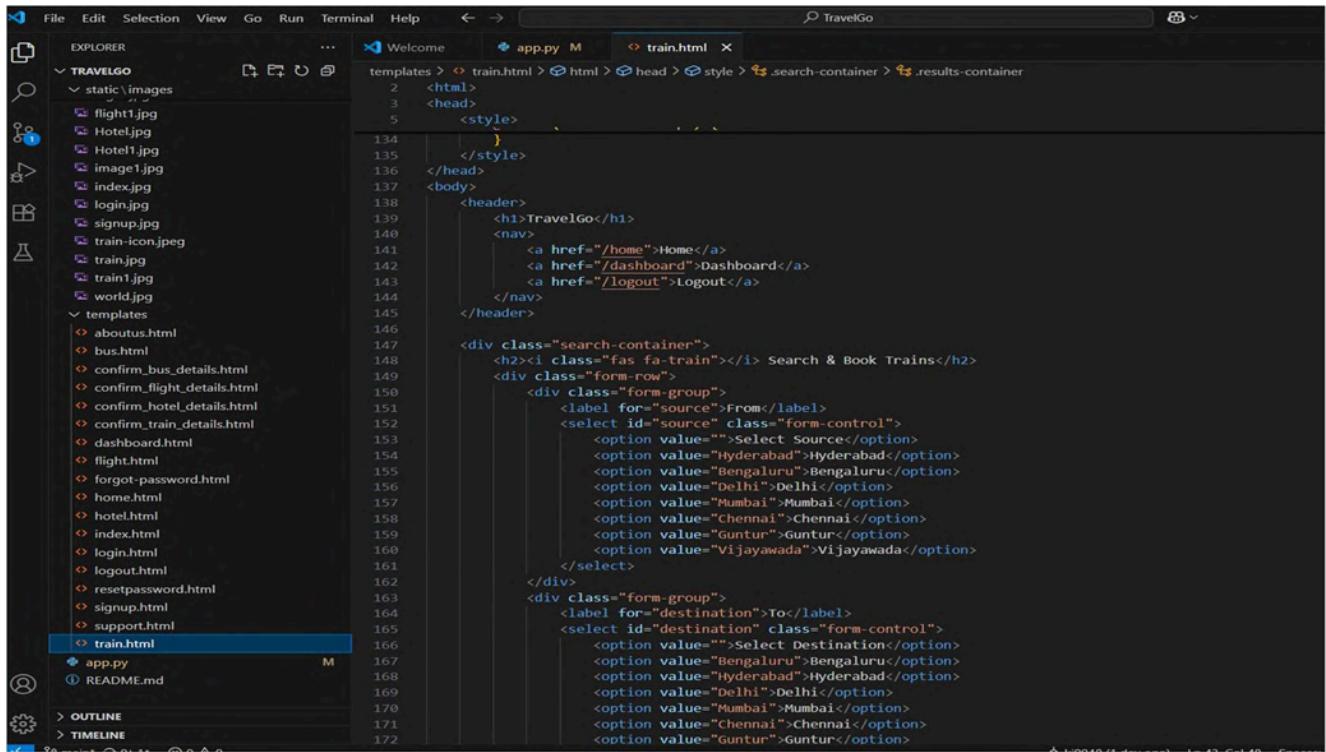


```
File Edit Selection View Go Run Terminal Help < > TravelGo
EXPLORER ... Welcome app.py M confirm_bus_details.html
templates > confirm_bus_details.html > html > body > script > addEventListener('click') callback
186 <html lang="en">
187   <body>
188     <div class="container">
189       <div class="booking-summary">
190         </div>
191       <div id="seat-map" class="seat-map"></div>
192       <input type="hidden" id="selectedSeatsInput" />
193       <p><strong>Bus Name:</strong> {{ booking.name }}</p>
194       <p><strong>Route:</strong> {{ booking.source }} to {{ booking.destination }}</p>
195       <p><strong>Date:</strong> {{ booking.travel_date }}</p>
196       <p><strong>Time:</strong> {{ booking.time }}</p>
197       <p><strong>Type:</strong> {{ booking.type }}</p>
198       <p><strong>Passengers:</strong> {{ booking.num_persons }}</p>
199       <p><strong>Selected Seats:</strong> <span id="selectedSeatsDisplay">None</span></p>
200     </div>
201     <p class="total-price">Total Price: ₹{{ "(:.2f)".format(booking.total_price) }}</p>
202   </div>
203   <div class="buttons-container">
204     <button id="confirmBookingBtn">Confirm Booking</button>
205     <a href="/bus" class="btn btn-secondary">Go Back to Search</a>
206   </div>
207 </div>
208 <footer>
209   &copy; 2025 TravelGo. All rights reserved.
210 </footer>
211 <script>
212   const seatLabels = [
213     "S1", "S2", "", "S3", "S4",
214     "S5", "S6", "", "S7", "S8",
215     "S9", "S10", "", "S11", "S12",
216     "S13", "S14", "", "S15", "S16",
217     "S17", "S18", "", "S19", "S20",
218     "S21", "S22", "", "S23", "S24",
219     "S25", "S26", "", "S27", "S28"
220   ];
221 </script>
```

Bus Confirm Booking Page :



Train Booking Code :

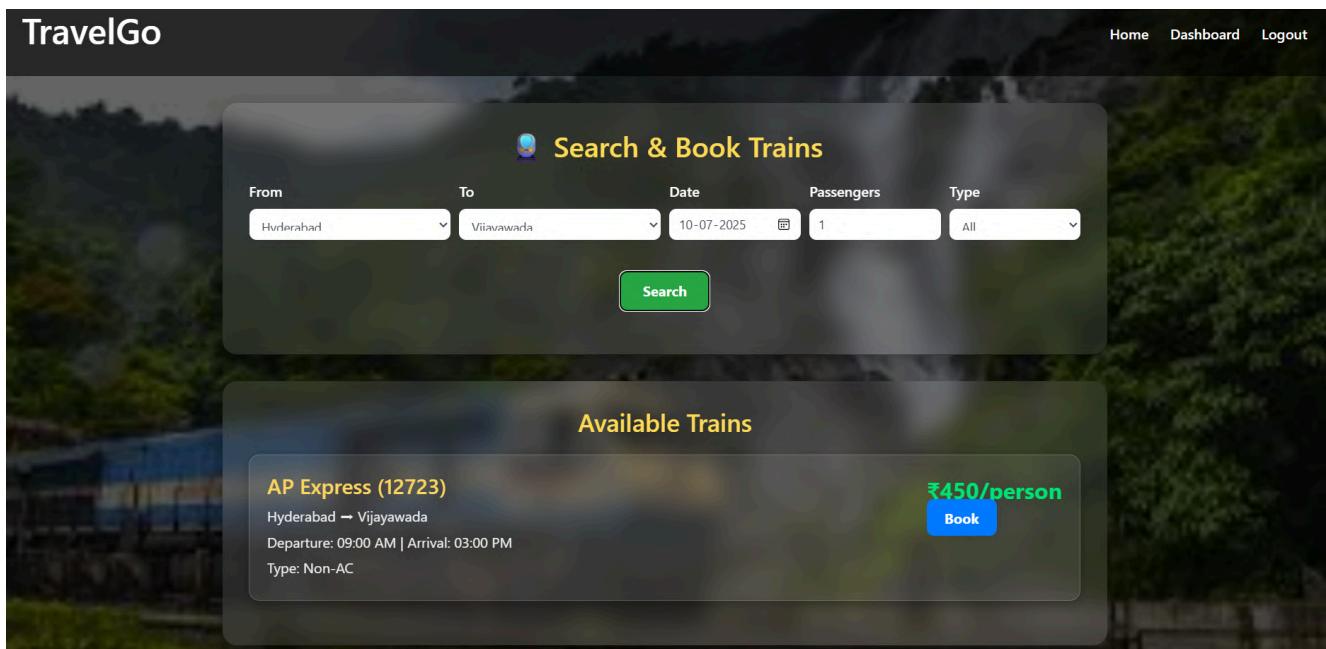


The screenshot shows a code editor interface with the following details:

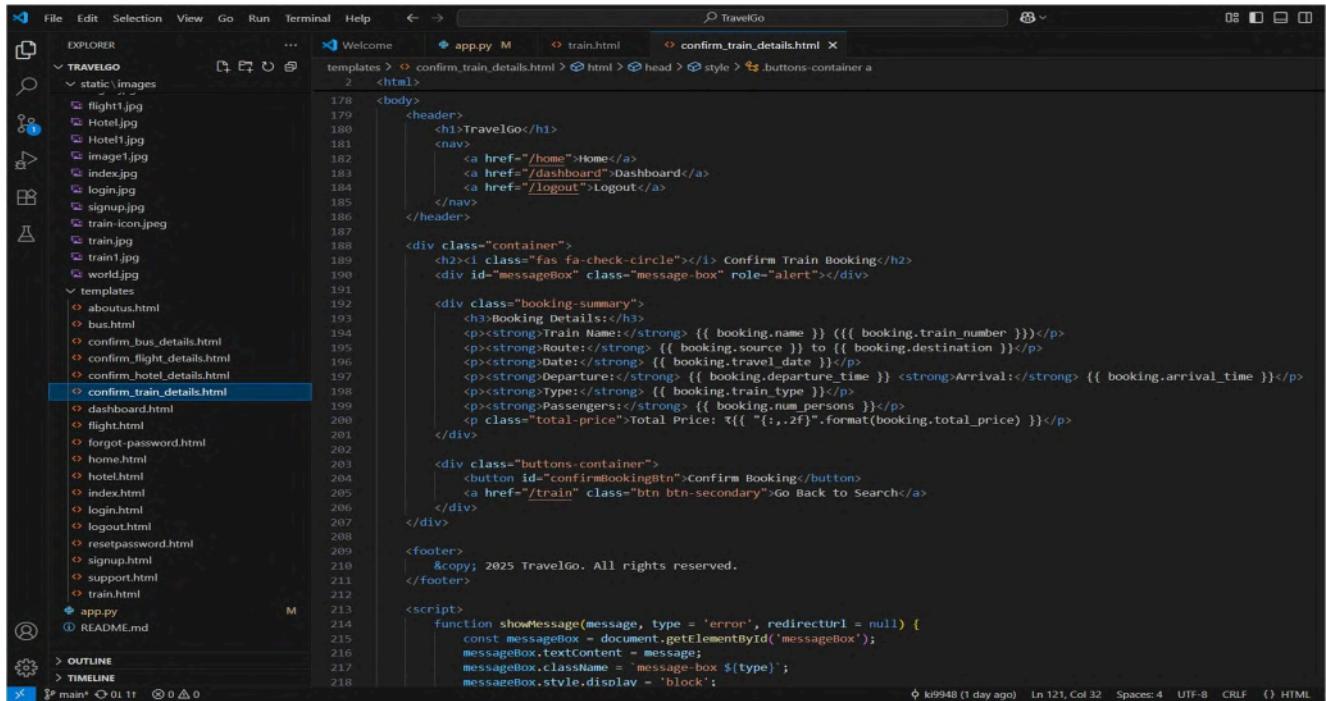
- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Explorer:** Shows a tree view of files and folders under the TRAVELGO project, including static/images and templates subfolders containing various HTML files like aboutus.html, bus.html, etc.
- Code Editor:** The current file is train.html, which contains the following code:

```
<html>
  <head>
    <style>
      ...
    </style>
  </head>
  <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="search-container">
      <h2><i class="fas fa-train"></i> Search & Book Trains</h2>
      <div class="form-row">
        <div class="form-group">
          <label for="source">From</label>
          <select id="source" class="form-control">
            <option value="">Select Source</option>
            <option value="Hyderabad">Hyderabad</option>
            <option value="Bengaluru">Bengaluru</option>
            <option value="Delhi">Delhi</option>
            <option value="Mumbai">Mumbai</option>
            <option value="Chennai">Chennai</option>
            <option value="Guntur">Guntur</option>
            <option value="Vijayawada">Vijayawada</option>
          </select>
        </div>
        <div class="form-group">
          <label for="destination">To</label>
          <select id="destination" class="form-control">
            <option value="">Select Destination</option>
            <option value="Bengaluru">Bengaluru</option>
            <option value="Hyderabad">Hyderabad</option>
            <option value="Delhi">Delhi</option>
            <option value="Mumbai">Mumbai</option>
            <option value="Chennai">Chennai</option>
            <option value="Guntur">Guntur</option>
          </select>
        </div>
      </div>
      <div class="form-group">
        <label for="date">Date</label>
        <input type="text" value="10-07-2025" id="date" />
      </div>
      <div class="form-group">
        <label for="passenger">Passengers</label>
        <input type="text" value="1" id="passenger" />
      </div>
      <div class="form-group">
        <label for="type">Type</label>
        <input type="text" value="All" id="type" />
      </div>
      <button type="button" class="Search">Search
    </div>
  </body>
</html>
```

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

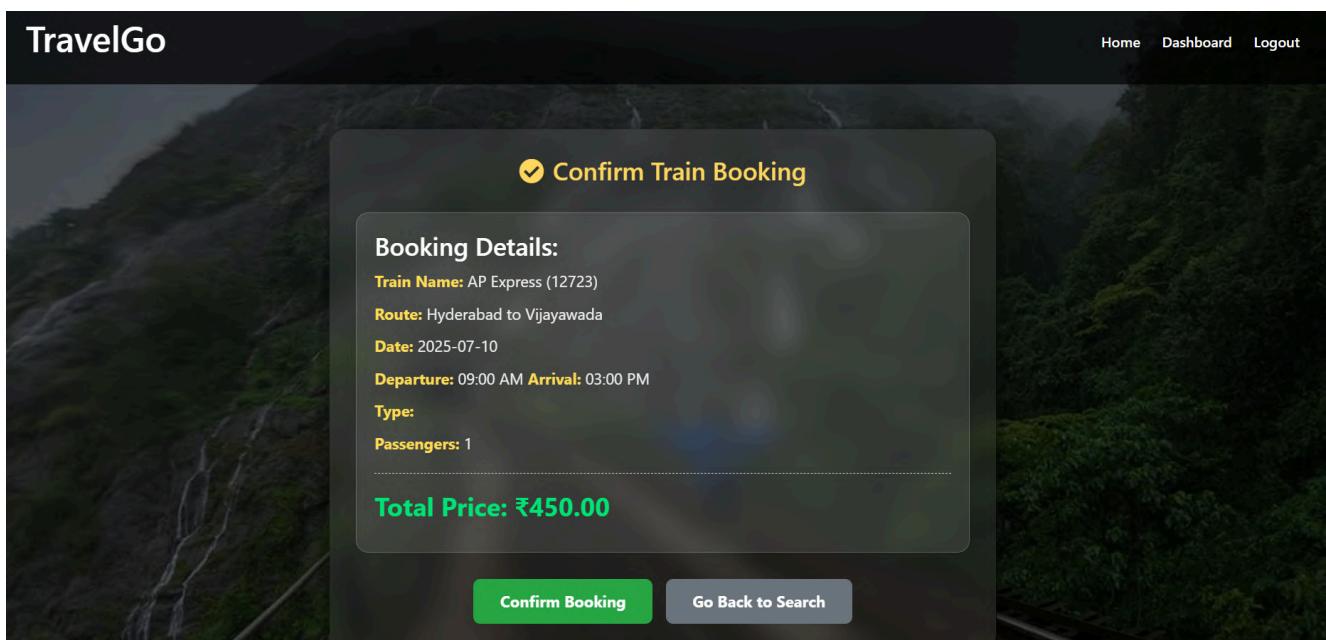
      <div class="booking-summary">
        <h3>Booking Details:</h3>
        <p><strong>Train Name:</strong> {{ booking.name }} ({{ booking.train_number }})</p>
        <p><strong>Route:</strong> {{ booking.source }} to {{ booking.destination }}</p>
        <p><strong>Date:</strong> {{ booking.travel_date }}</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>Passenger:</strong> {{ booking.num_persons }}</p>
        <p>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>

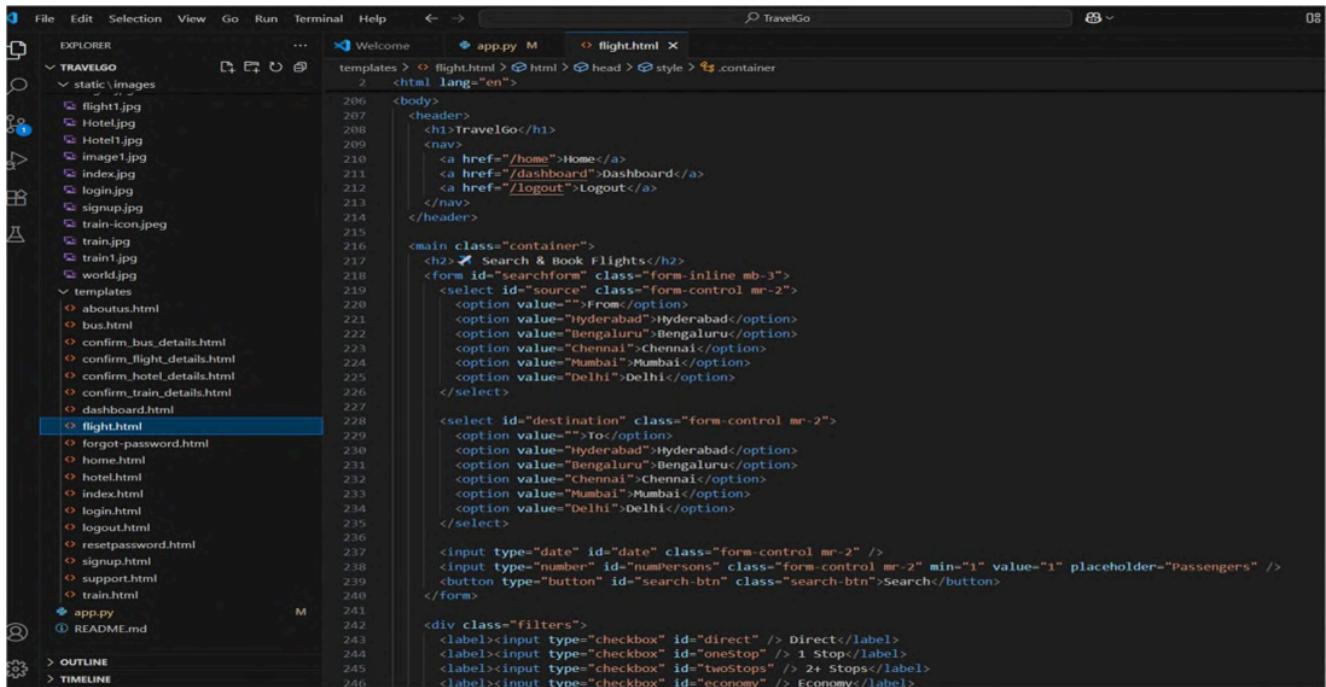
    <footer>
      &copy; 2025 TravelGo. All rights reserved.
    </footer>
  </body>
</html>

<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';
  }
</script>
```

Train Confirm Booking Page :

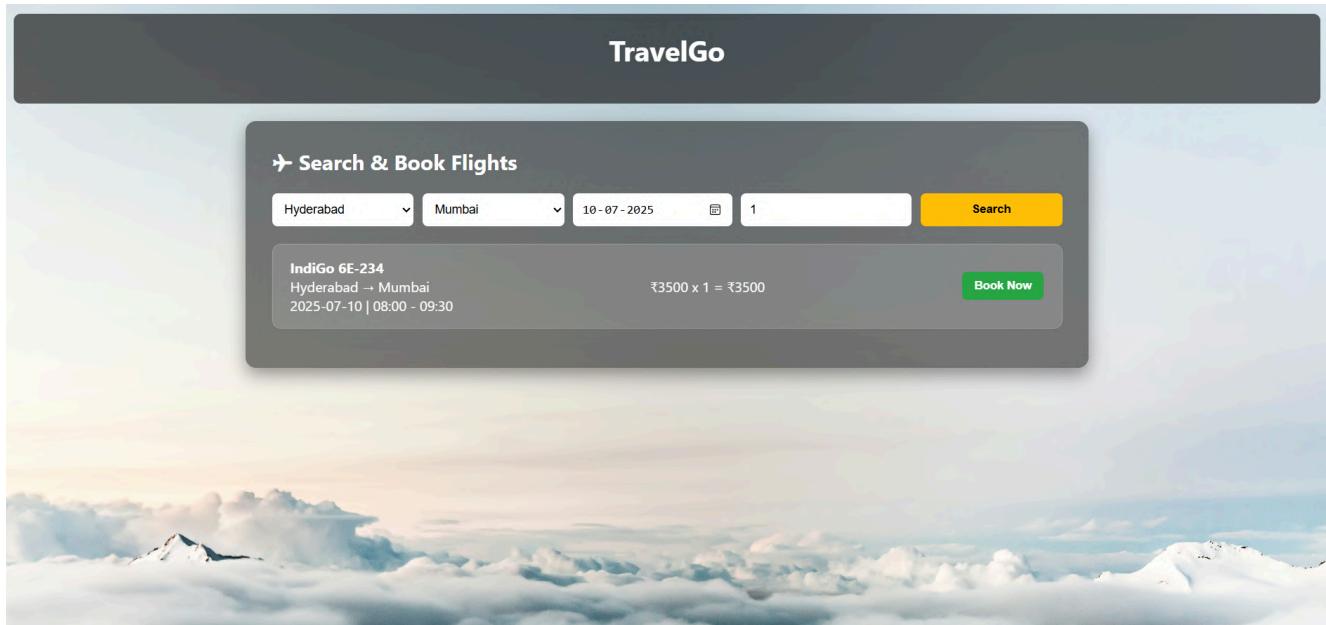


Flight Booking Code :



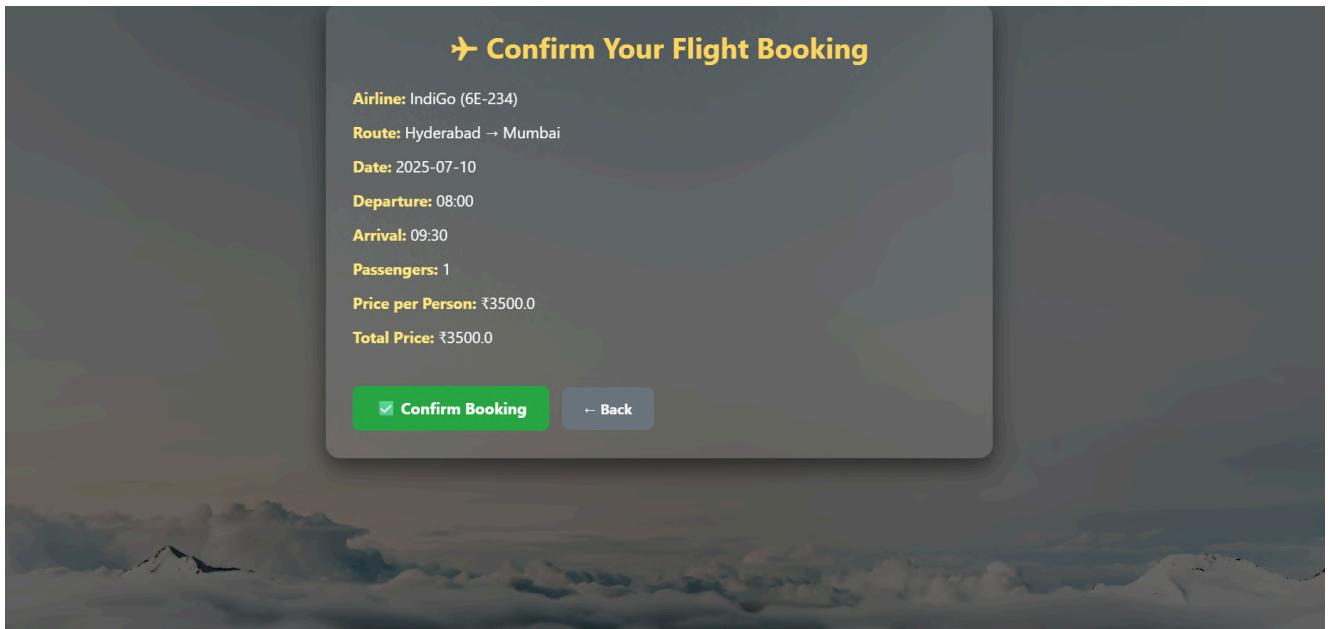
```
File Edit Selection View Go Run Terminal Help < > TravelGo
EXPLORER templates > flight.html > app.py M flight.html X
static\images
flight1.jpg
Hotel.jpg
Hotel1.jpg
image1.jpg
index.jpg
login.jpg
signup.jpg
train-icon.jpeg
train.jpg
train1.jpg
world.jpg
templates
aboutus.html
bus.html
confirm_bus_details.html
confirm_flight_details.html
confirm_hotel_details.html
confirm_train_details.html
dashboard.html
flight.html
forgot-password.html
home.html
hotel.html
index.html
login.html
logout.html
resetpassword.html
signup.html
support.html
train.html
app.py
README.md
OUTLINE
TIMELINE
2 <html lang="en">
  <body>
    <header>
      <h1>TravelGo</h1>
      <nav>
        <a href="/home">Home</a>
        <a href="/dashboard">Dashboard</a>
        <a href="/logout">Logout</a>
      </nav>
    </header>
    <main class="container">
      <h2>Search & Book Flights</h2>
      <form id="searchform" class="form-inline mb-3">
        <select id="source" class="form-control mr-2">
          <option value="">From</option>
          <option value="Hyderabad">Hyderabad</option>
          <option value="Bengaluru">Bengaluru</option>
          <option value="Chennai">Chennai</option>
          <option value="Mumbai">Mumbai</option>
          <option value="Delhi">Delhi</option>
        </select>
        <select id="destination" class="form-control mr-2">
          <option value="">To</option>
          <option value="Hyderabad">Hyderabad</option>
          <option value="Bengaluru">Bengaluru</option>
          <option value="Chennai">Chennai</option>
          <option value="Mumbai">Mumbai</option>
          <option value="Delhi">Delhi</option>
        </select>
        <input type="date" id="date" class="form-control mr-2" />
        <input type="number" id="numPersons" class="form-control mr-2" min="1" value="1" placeholder="Passengers" />
        <button type="button" id="search-btn" class="search-btn">Search</button>
      </form>
      <div class="filters">
        <label><input type="checkbox" id="direct" /> Direct</label>
        <label><input type="checkbox" id="oneStop" /> 1 Stop</label>
        <label><input type="checkbox" id="twoStops" /> 2+ Stops</label>
        <label><input type="checkbox" id="economy" /> Economy</label>
      </div>
    </main>
  </body>
</html>
```

Flight Booking Page :



Flight Confirm Booking Code :

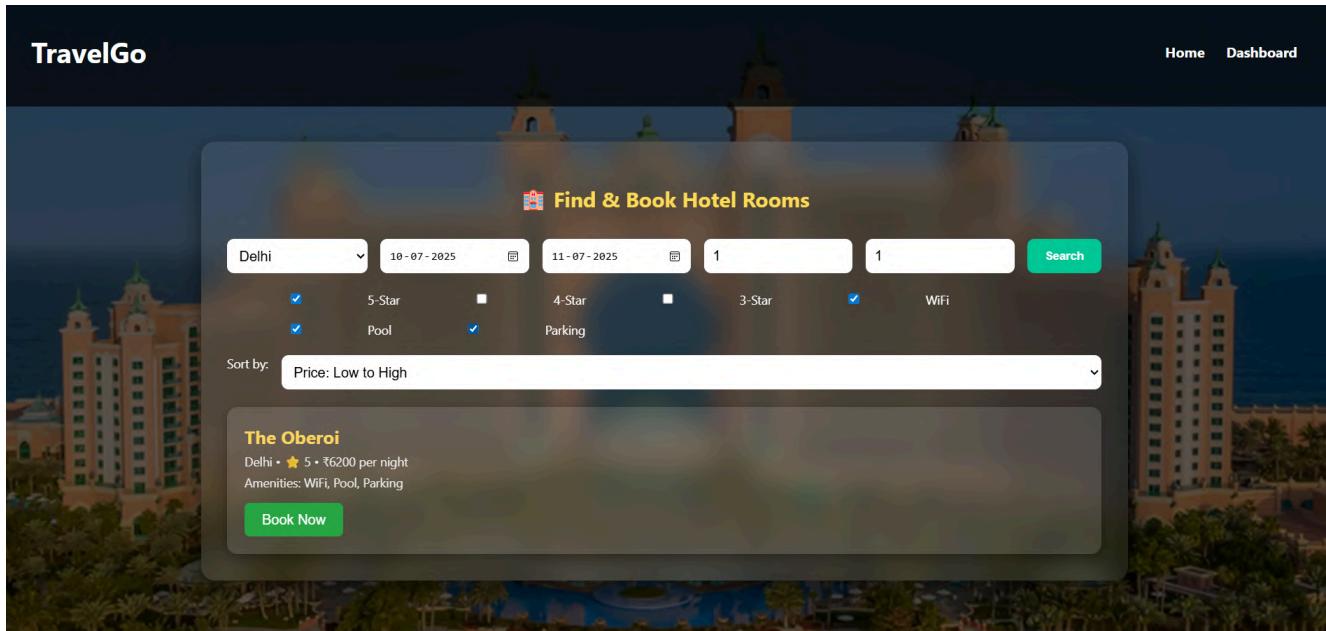
Flight Confirm Booking Page :



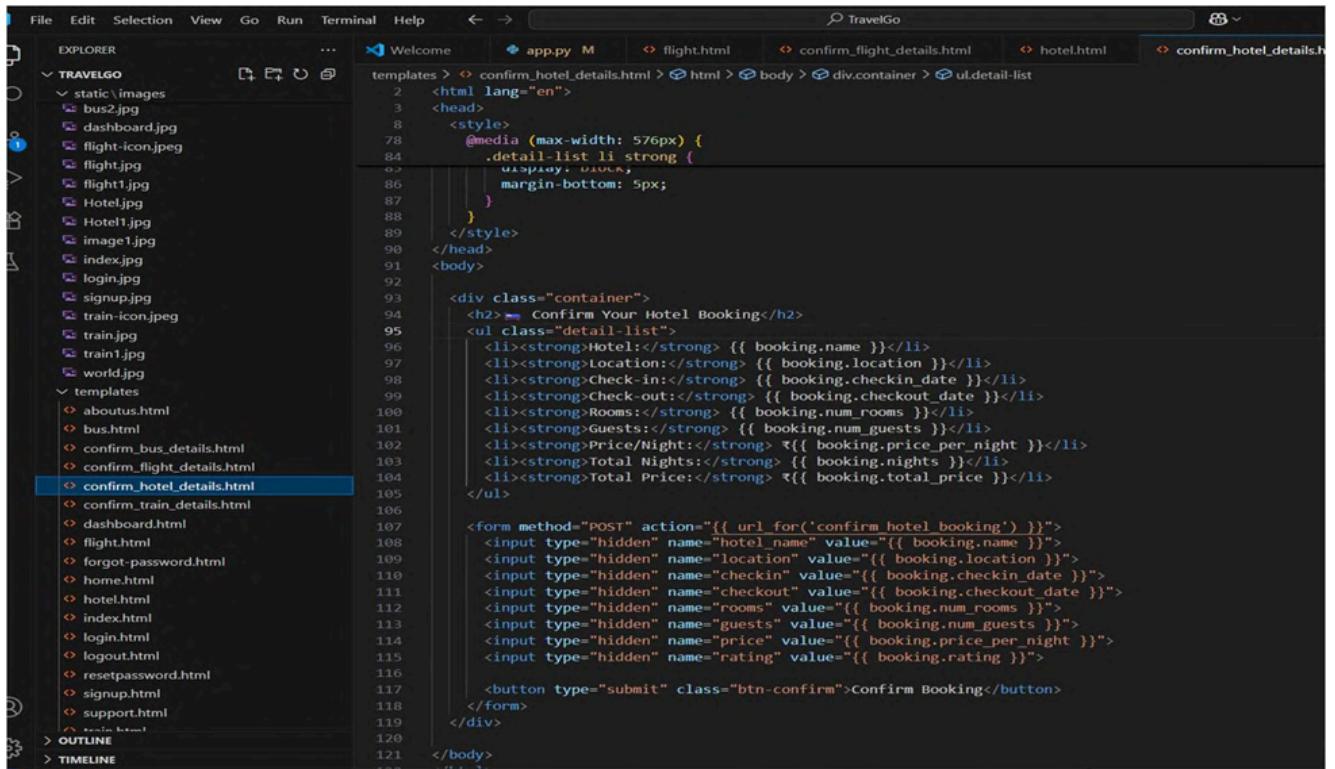
Hotel Page Code :

```
<div class="container">
  <h2> Find & Book Hotel Rooms</h2>
  <div class="form-row">
    <select id="location">
      <option value="">Select City</option>
      <option value="Hyderabad">Hyderabad</option>
      <option value="Mumbai">Mumbai</option>
      <option value="Delhi">Delhi</option>
      <option value="Bangalore">Bangalore</option>
    </select>
    <input type="date" id="checkinDate" />
    <input type="date" id="checkoutDate" />
    <input type="number" id="numRooms" min="1" value="1" placeholder="No. of Rooms" />
    <input type="number" id="numGuests" min="1" value="1" placeholder="No. of Guests" />
    <button id="search-btn" class="search-btn">Search</button>
  </div>
  <div class="filters">
    <label><input type="checkbox" id="fiveStar" /> 5-Star</label>
    <label><input type="checkbox" id="fourStar" /> 4-Star</label>
    <label><input type="checkbox" id="threeStar" /> 3-Star</label>
    <label><input type="checkbox" id="WiFi" /> WiFi</label>
    <label><input type="checkbox" id="Pool" /> Pool</label>
    <label><input type="checkbox" id="Parking" /> Parking</label>
  </div>
  <div class="sort-row">
    <label for="sort">Sort by:</label>
    <select id="sort">
      <option value="">None</option>
      <option value="price-low">Price: Low to High</option>
      <option value="price-high">Price: High to Low</option>
      <option value="rating-high">Rating: High to Low</option>
    </select>
  </div>
  <div id="hotel-list" class="hotel-list"></div>
</div>
```

Hotel Page :



Hotel Confirming Booking Code :

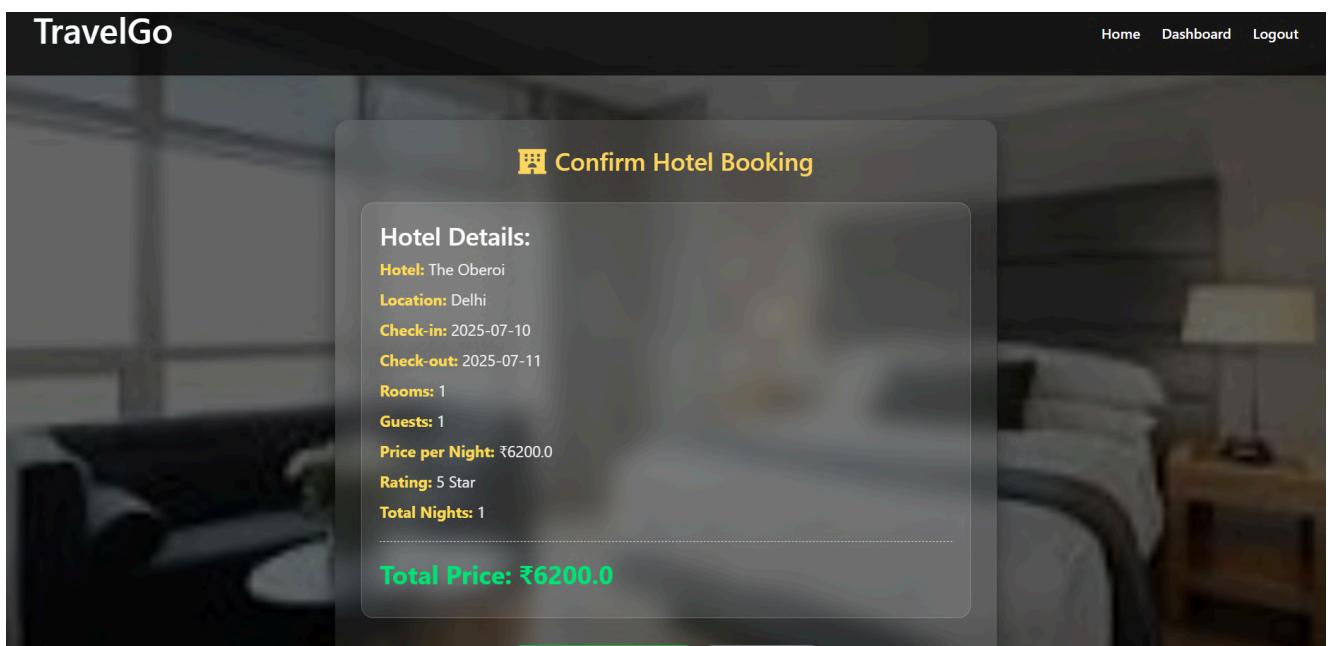


The screenshot shows a code editor interface with the following details:

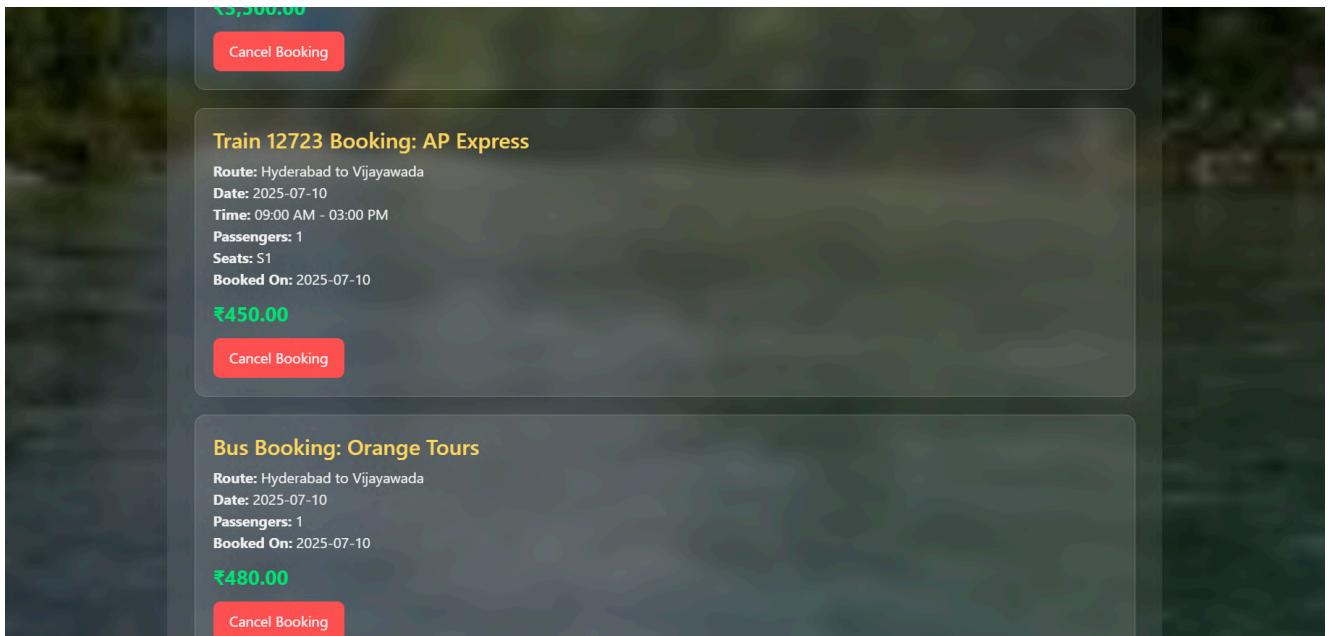
- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Toolbar:** Welcome, app.py, flight.html, confirm_flight_details.html, hotel.html, confirm_hotel_details.html.
- Explorer:** Shows a file tree for a project named "TRAVELGO". It includes static images like bus2.jpg, dashboard.jpg, flight-icon.jpeg, flight1.jpg, flight1i.jpg, Hotel.jpg, Hotel1.jpg, image1.jpg, index.jpg, login.jpg, signup.jpg, train-icon.jpeg, train.jpg, train1.jpg, world.jpg, and various templates such as aboutus.html, bus.html, confirm_bus_details.html, confirm_flight_details.html, and confirm_hotel_details.html.
- Code Editor:** Displays the HTML and CSS code for the "confirm_hotel_details.html" template. The code includes a header with styles for media queries, a container div, and a list of booking details (Hotel, Location, Check-in, Check-out, Rooms, Guests, Price/Night, Total Nights, Total Price). Below this is a form for submitting the booking information.

```
2 <html lang="en">
3 <head>
4   <style>
5     @media (max-width: 576px) {
6       .detail-list li strong {
7         margin-bottom: 5px;
8       }
9     }
10   </style>
11 </head>
12 <body>
13
14   <div class="container">
15     <h2>Confirm Your Hotel Booking</h2>
16     <ul class="detail-list">
17       <li><strong>Hotel:</strong> {{ booking.name }}</li>
18       <li><strong>Location:</strong> {{ booking.location }}</li>
19       <li><strong>Check-in:</strong> {{ booking.checkin_date }}</li>
20       <li><strong>Check-out:</strong> {{ booking.checkout_date }}</li>
21       <li><strong>Rooms:</strong> {{ booking.num_rooms }}</li>
22       <li><strong>Guests:</strong> {{ booking.num_guests }}</li>
23       <li><strong>Price/Night:</strong> ₹{{ booking.price_per_night }}</li>
24       <li><strong>Total Nights:</strong> {{ booking.nights }}</li>
25       <li><strong>Total Price:</strong> ₹{{ booking.total_price }}</li>
26     </ul>
27
28     <form method="POST" action="{{ url_for('confirm_hotel_booking') }}">
29       <input type="hidden" name="hotel_name" value="{{ booking.name }}"/>
30       <input type="hidden" name="location" value="{{ booking.location }}"/>
31       <input type="hidden" name="checkin" value="{{ booking.checkin_date }}"/>
32       <input type="hidden" name="checkout" value="{{ booking.checkout_date }}"/>
33       <input type="hidden" name="rooms" value="{{ booking.num_rooms }}"/>
34       <input type="hidden" name="guests" value="{{ booking.num_guests }}"/>
35       <input type="hidden" name="price" value="{{ booking.price_per_night }}"/>
36       <input type="hidden" name="rating" value="{{ booking.rating }}"/>
37
38       <button type="submit" class="btn-confirm">Confirm Booking</button>
39     </form>
40   </div>
41
42 </body>
```

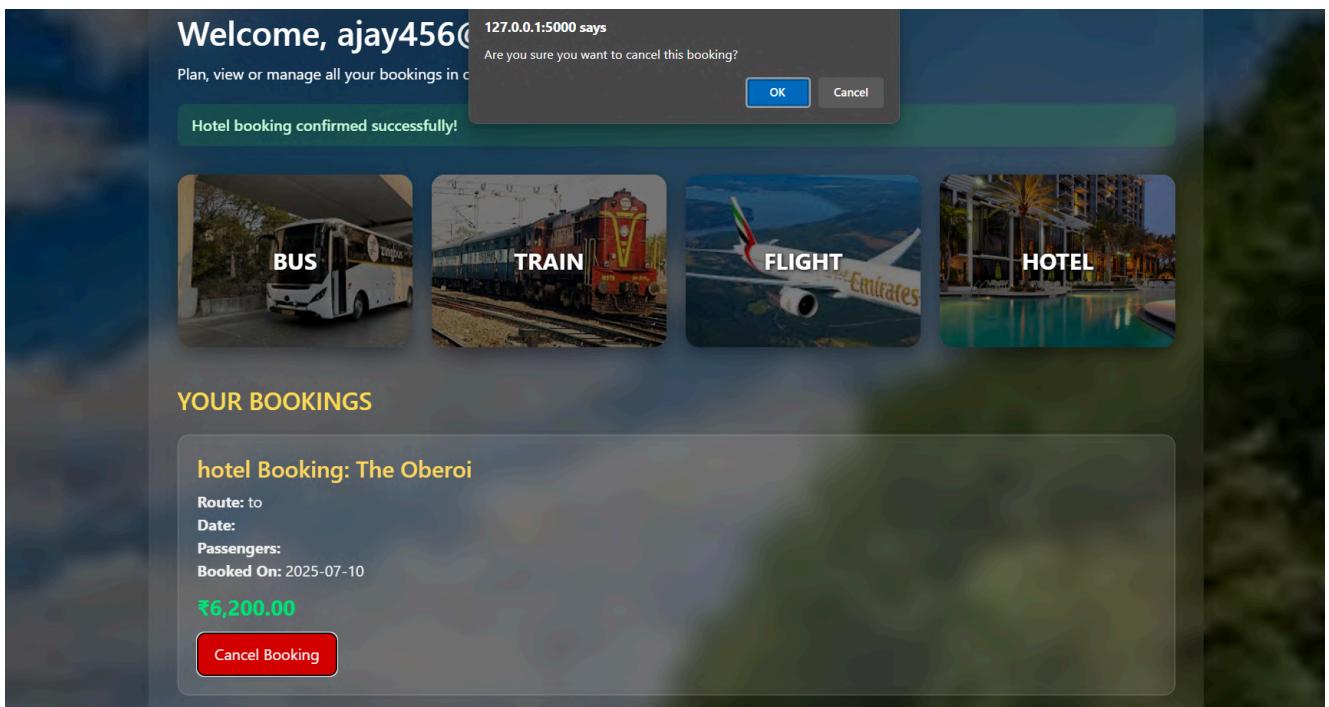
Hotel Confirming Booking Page :



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.

