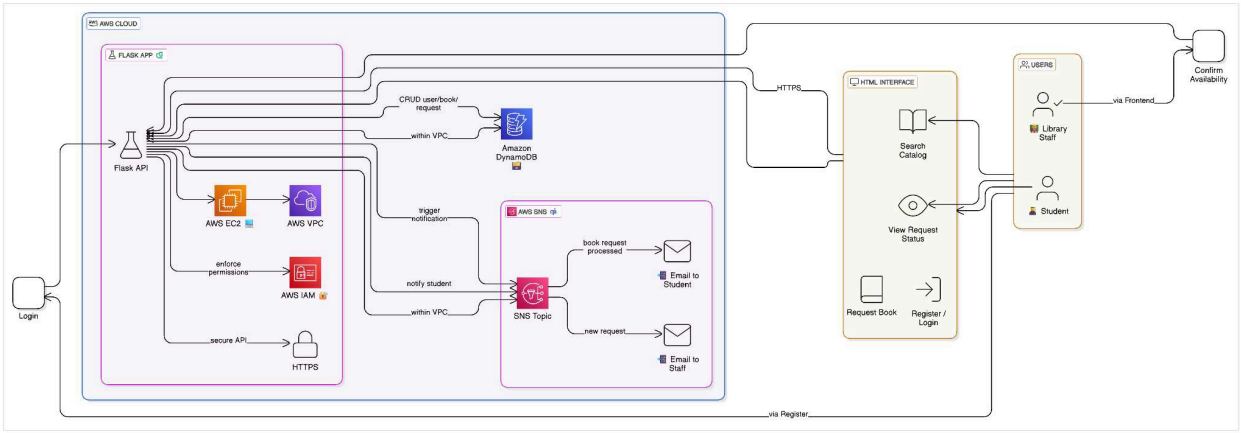
Travel Go: A Cloud-Powered Real-Time Travel Booking Platform

Project Description:

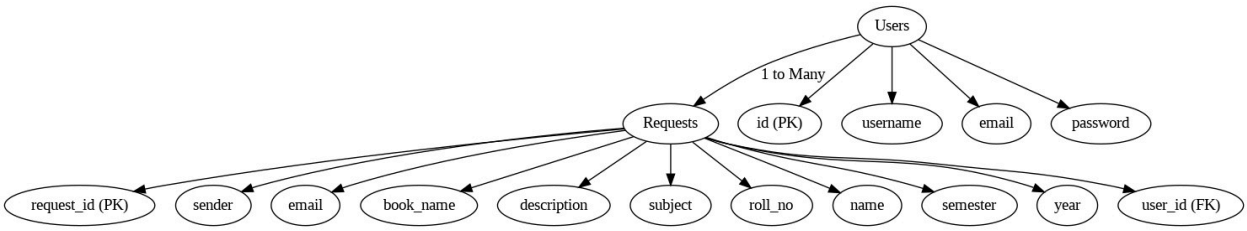
Travel Go is designed to simplify travel planning by providing a unified platform for booking buses, trains, flights, and hotels. With growing demand for real-time,hassle-free travel services, Travel Go addresses this by leveraging the cloud technologies. The application uses Flask for backend development, AWS EC2 for reliable deployment, DynamoDB for efficient data storage, and AWS SNS for real time notifications. Users can register, log in, search, and book their preferred mode of transport and accommodations. The system also offers booking history tracking and dynamic seat selection features, providing a smooth, responsive, and accessible experience for all travelers.

Scenario 1: Real-Time Booking Experience for Users Using Travel Go, a user logs into the platform and selects the travel category (bus/train/flight/hotel), and submits their booking preferences. Flask handles the backend logic—fetching available options from DynamoDB based on user input. Once the user confirms a booking, AWS EC2 ensures quick response times even after the booking is made.
under heavy traffic. The real-time nature of the system enables users to book their travel efficiently without long wait times or page loads, even during peak seasons.
Scenario 2: Automated Email Confirmations and Alerts. After confirming a booking, Travel Go uses AWS SNS to send instant email notifications to the user with booking details. For instance, a user books a flight, and Flask captures the transaction while AWS SNS sends an email confirming the booking to the user and alerting the service provider if needed. All booking data is securely stored in DynamoDB for future reference. This seamless integration ensures timely communication, improving user trust and experience.
Scenario 3: Effortless Booking Management and Access Users can log in anytime to view or manage their previous bookings. For example, a user accesses their dashboard to check hotel reservations made a week ago. Flask retrieves this data from DynamoDB instantly. Travel Go's cloud-backed design ensures uninterrupted access, and its intuitive interface allows users to cancel bookings or plan new trips easily. Hosting on AWS EC2 guarantees smooth performance even when multiple users access the platform concurrently.

AWS Architecture:



Entity Diagram:



Pre-requisites:

1. AWS Account Setup
2. Understanding IAM
3. Amazon EC2 Basics
4. DynamoDB Basics
5. SNS Overview
6. Git Version Control

Project Work Flow:

1. AWS Account Setup and Login
Activity 1.1: Set up an AWS account if not already done.
Activity 1.2: Log in to the AWS Management Console

2. DynamoDB Database Creation and Setup
Activity 2.1: Create a DynamoDB Table.
Activity 2.2: Configure Attributes for User Data and Book Requests.

3. SNS Notification Setup
Activity 3.1: Create SNS topics for book request notifications.
Activity 3.2: Subscribe users and library staff to SNS email notifications.

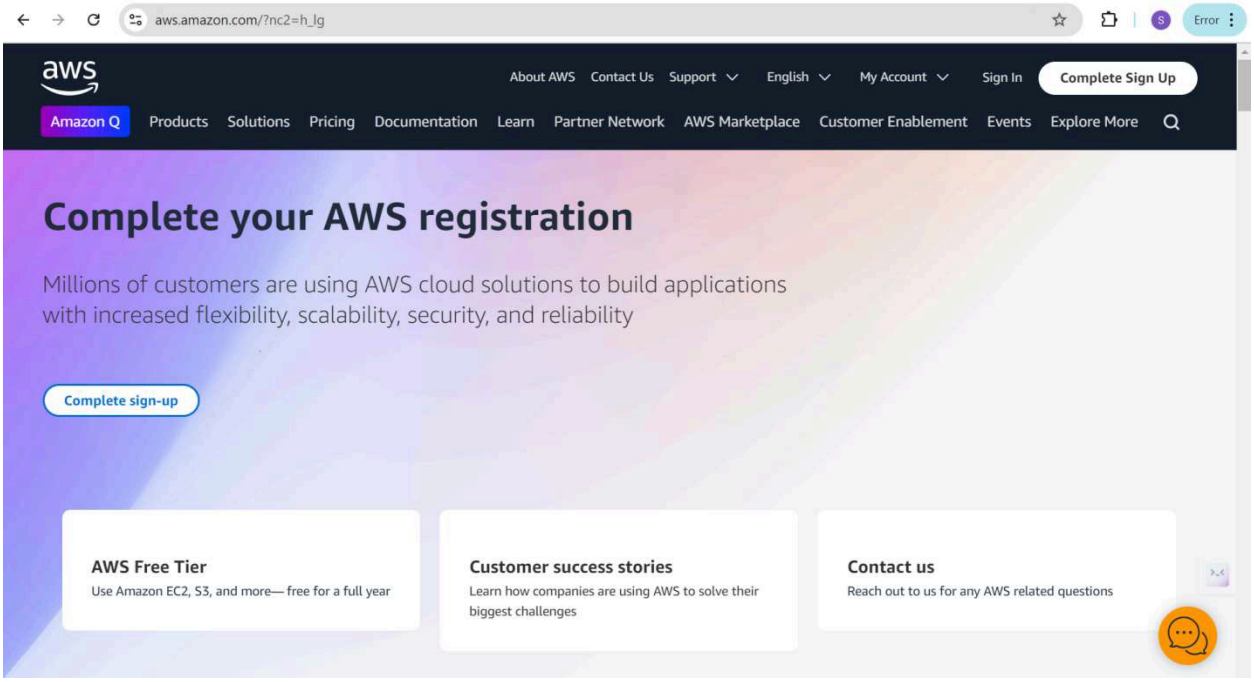
4. Backend Development and Application Setup
Activity 4.1: Develop the Backend Using Flask.
Activity 4.2: Integrate AWS Services Using boto3.

5. IAM Role Setup
Activity 5.1: Create IAM Role
Activity 5.2: Attach Policies

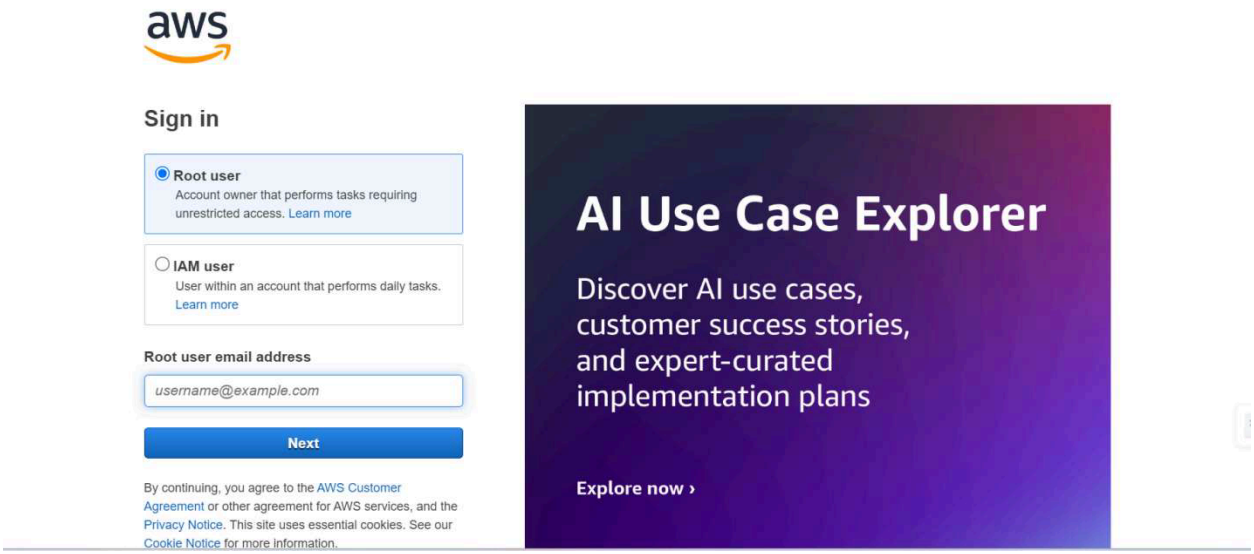
6. EC2 Instance Setup
Activity 6.1: Launch an EC2 instance to host the Flask application.
Activity 6.2: Configure security groups for HTTP, and SSH access.

7. Deployment on EC2
Activity 7.1: Upload Flask Files
Activity 7.2: Run the Flask App

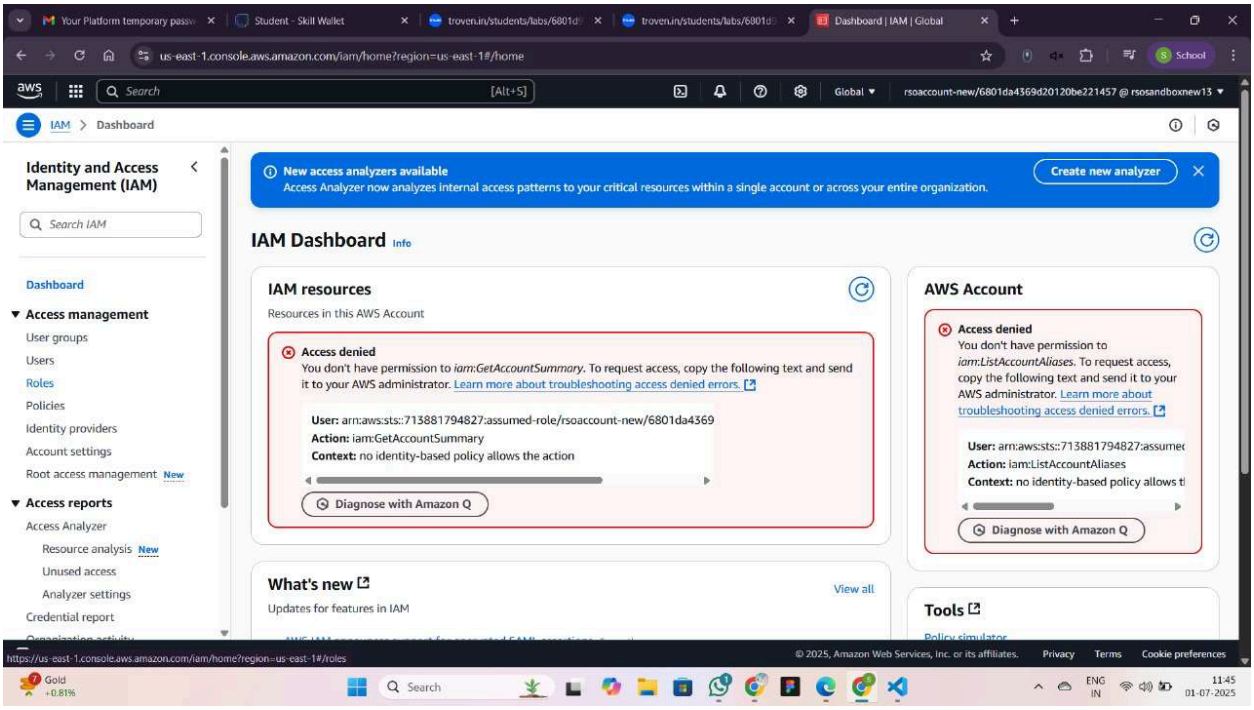
8. Testing and Deployment
AWS Account Setup and Login

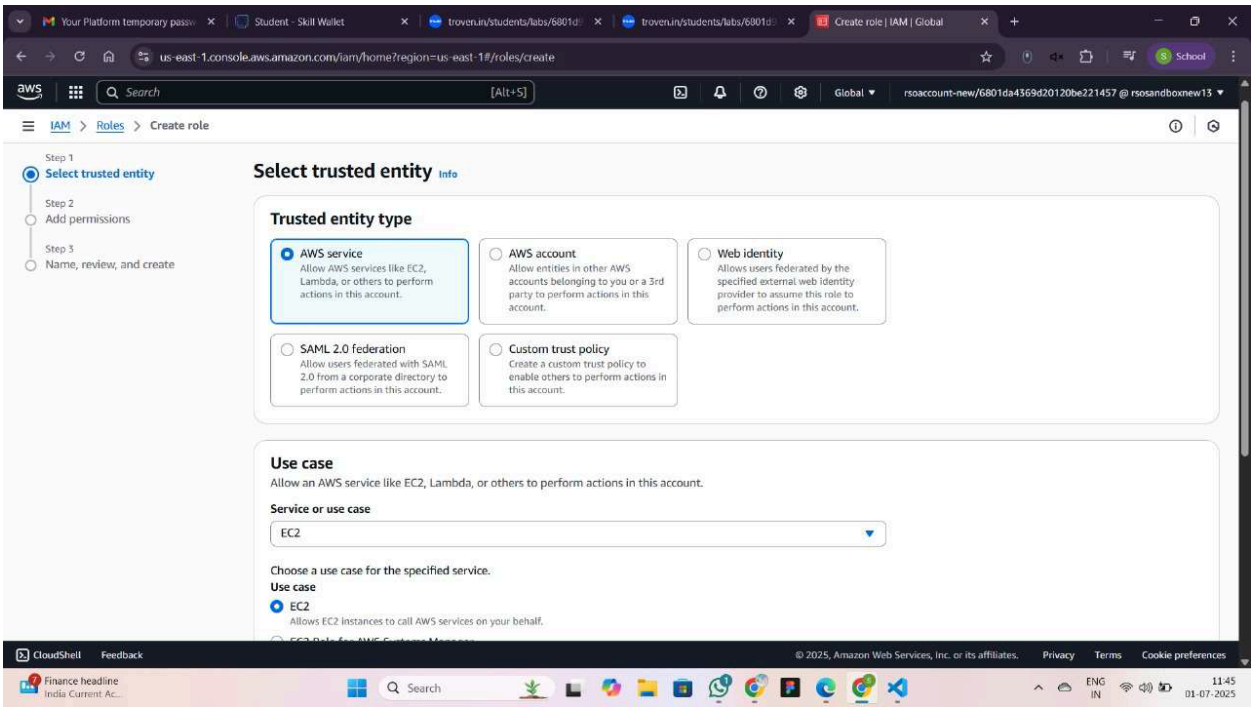


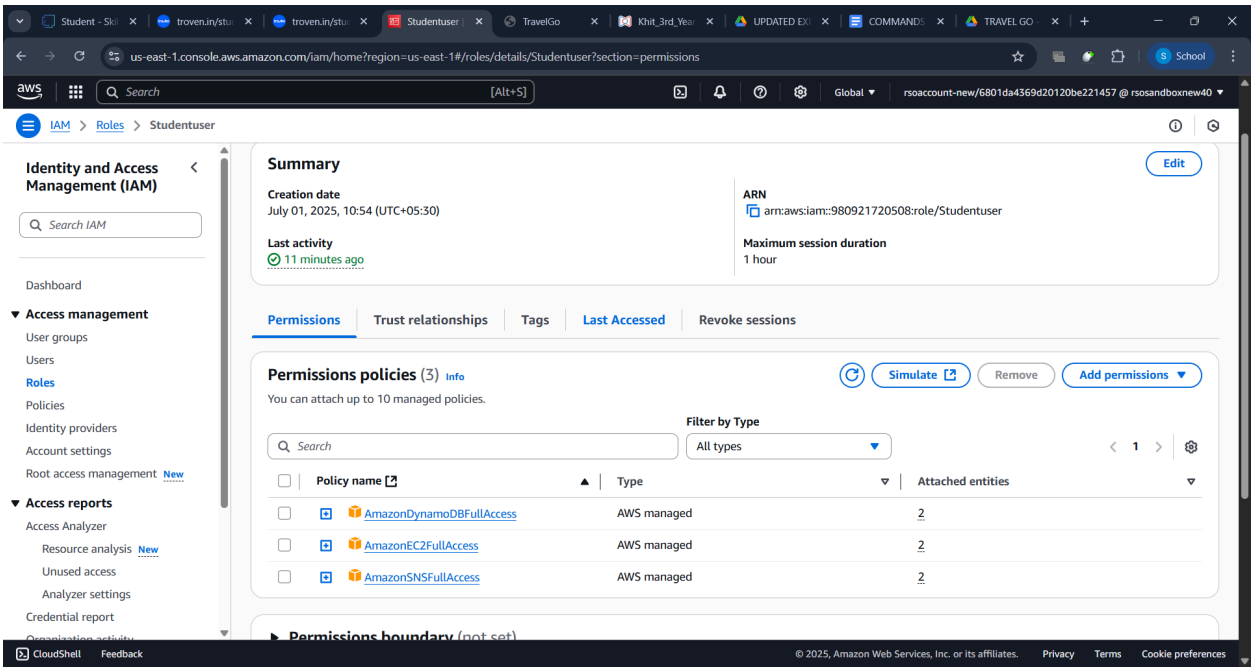
Log in to the AWS Management Console



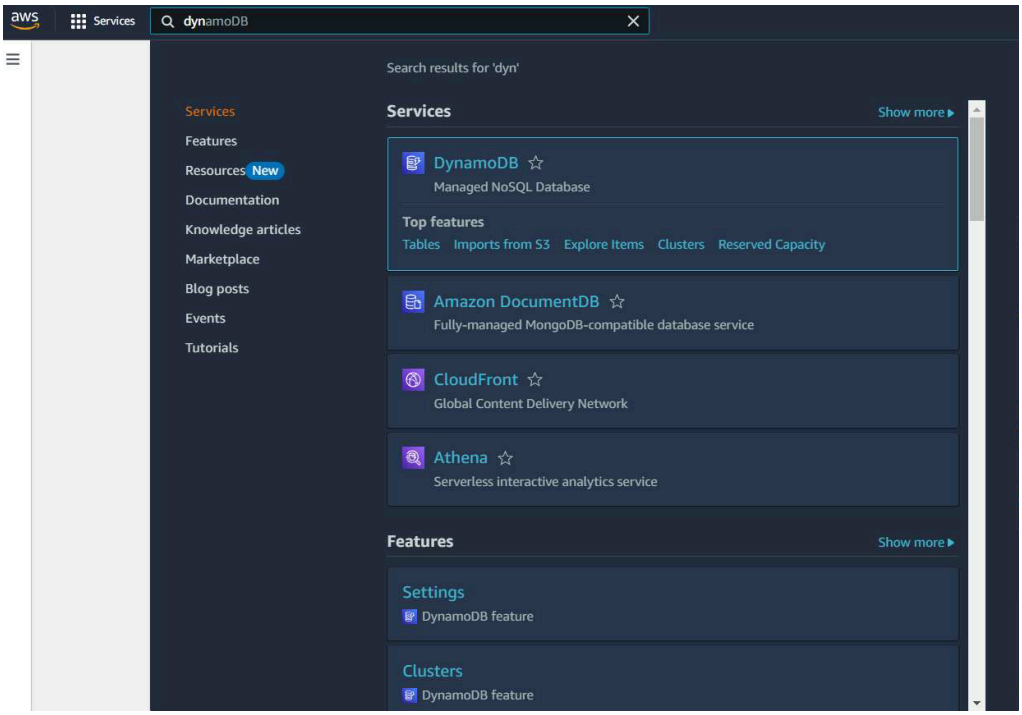
IAM role creation:







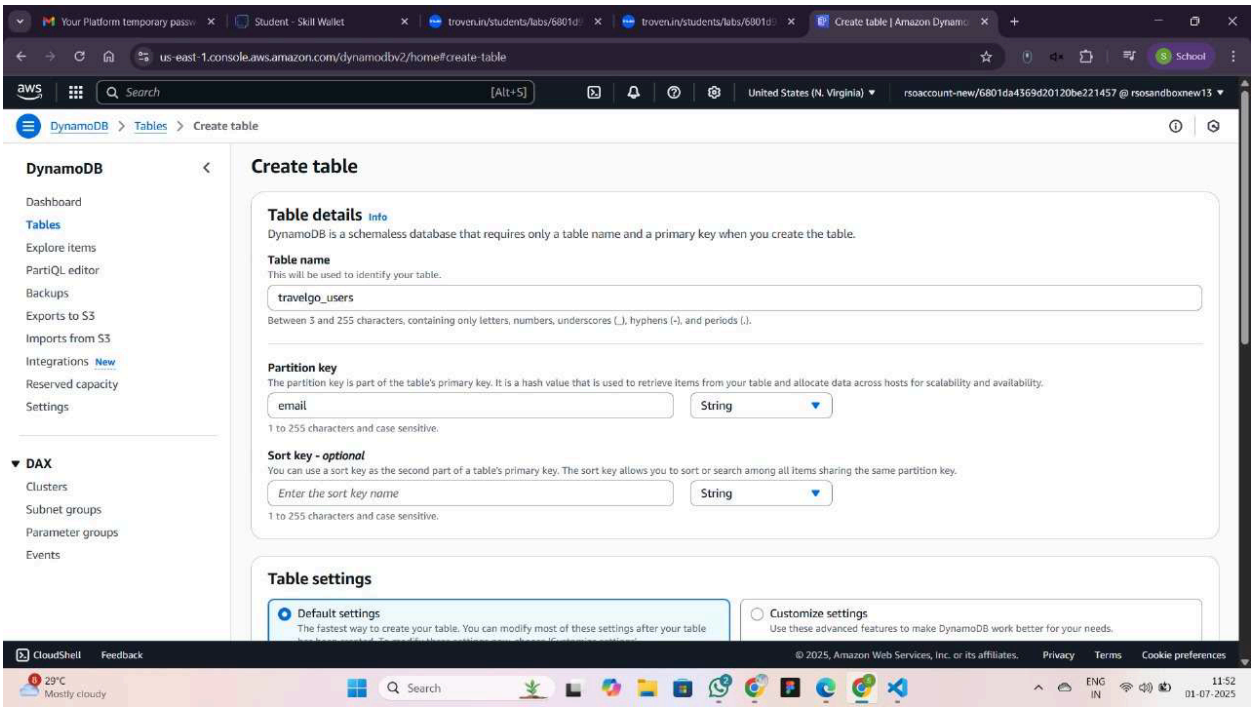
DynamoDBDatabase Creation and Setup



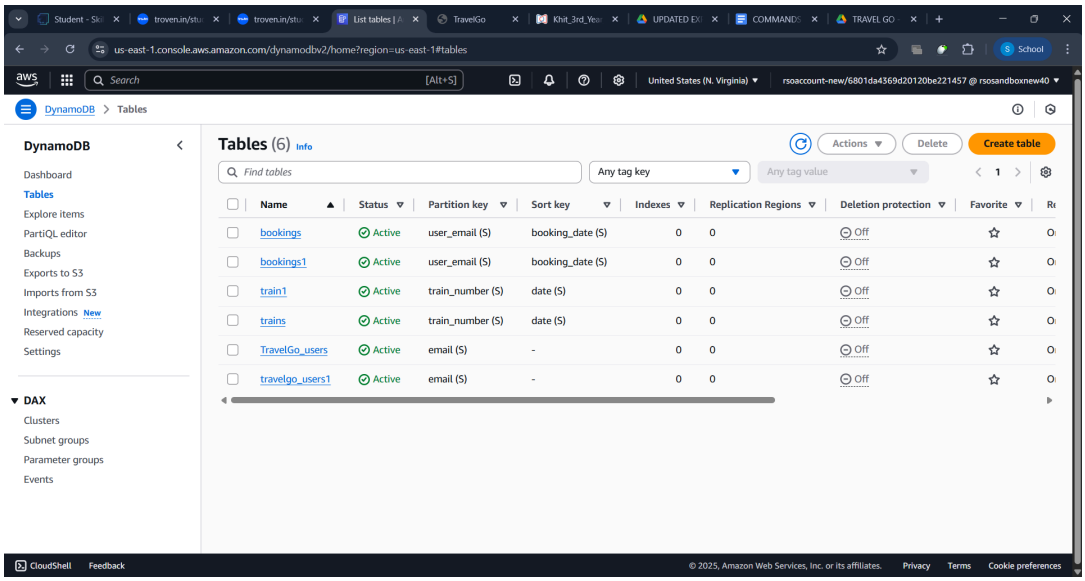
Create a DynamoDB table for storing registration details and book Requests

1. Create a Users table with partition key “Email” with type String and click on create tables

.

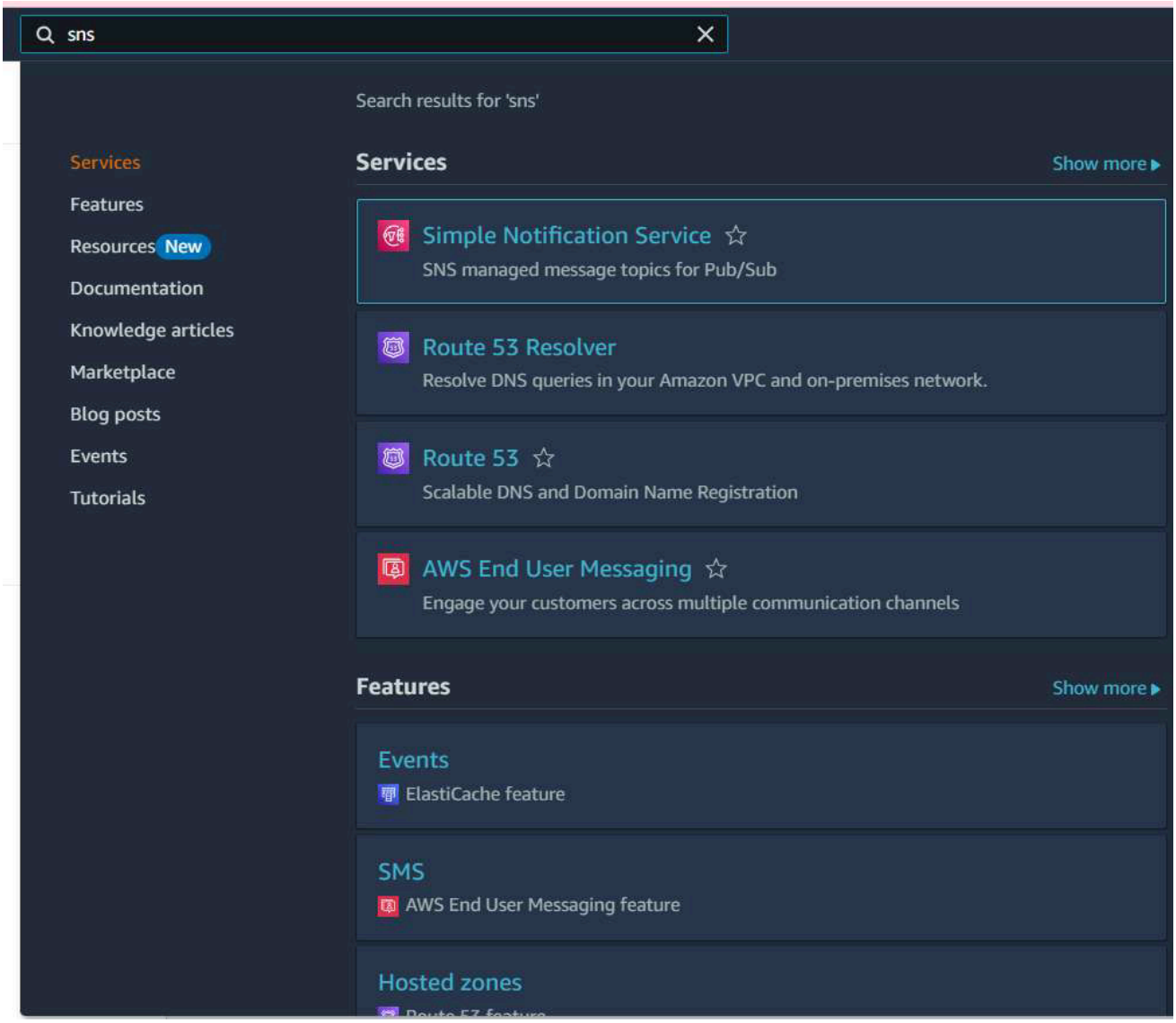


Follow the same steps to create a train table with train number as the partition key and bookings table with partition key as user email and sort key as booking date.



SNS Notification Setup

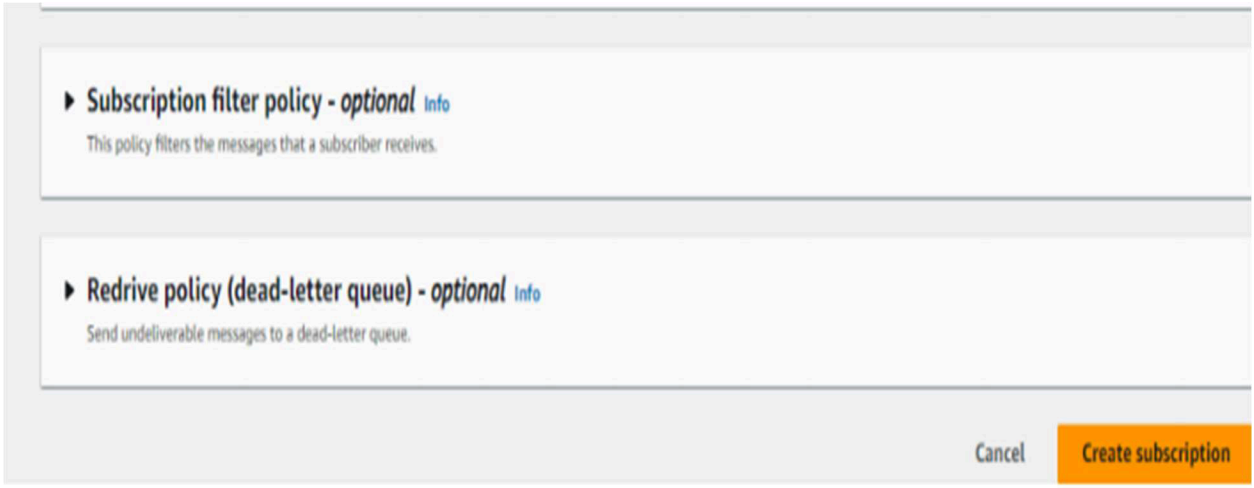
Create SNS topics for sending email notifications to users.

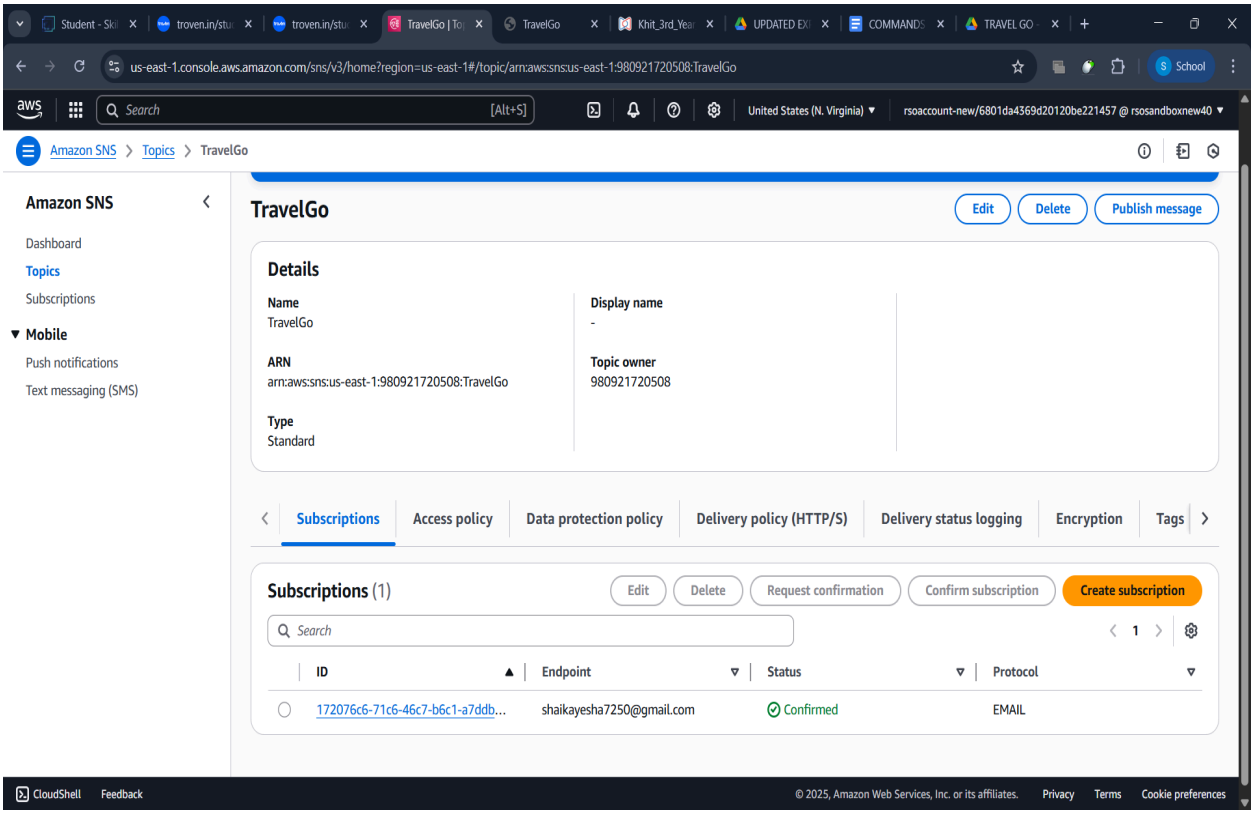


- Click on Create Topic and choose a name for the topic.
- Click on create topic.
- configure the SNS topic and note down the Topic ARN.

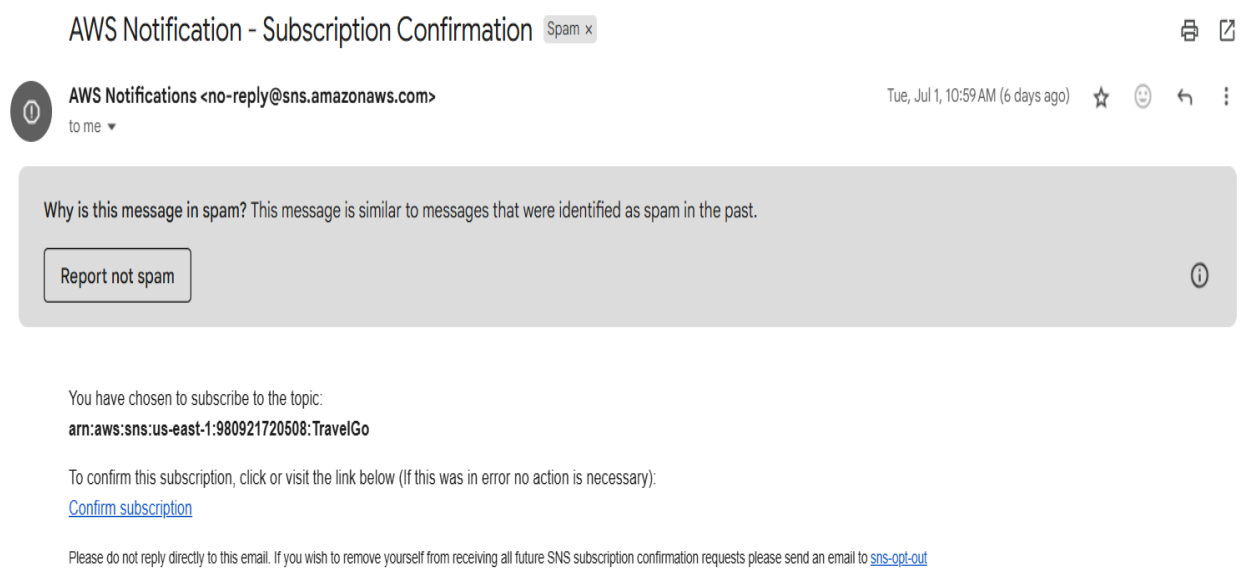
arn:are:as:Sns:us-east-1:980921720508:TravelGo:172076c6-71c6-46c7-b6c1-a7dba0f164e

- Click on create subscription

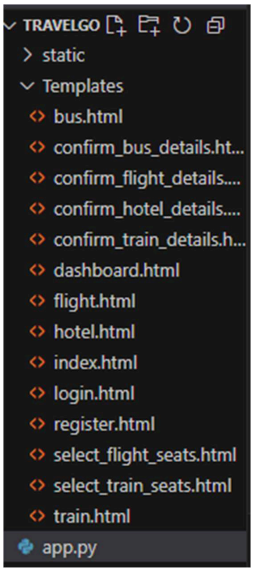




Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail



Backend Development and Application Setup




```
from flask import Flask, render_template, request, redirect, url_for, session, jsonify
from pymongo import MongoClient
from werkzeug.security import generate_password_hash, check_password_hash
import json
from datetime import datetime
from bson.objectid import ObjectId # Still needed for other potential ObjectIds from DB
from bson.errors import InvalidId # Import this for specific error handling
```

Flask: Initializes the web application.
render template: Renders HTML templates using Jinja2.
request: Handles incoming data from forms or API calls.

redirect and url_for: Used for redirecting between pages (e.g., login → dashboard).
session: Stores user-specific data across routes (e.g., user login session).
Jsonify: Converts Python data into JSON format (commonly used in API responses).

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

Initialize the Flask application instance using Flask(__name__) to start building the web app.

```
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 connection:

```
# Function to send SNS notifications
# 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('/')
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')
```

```python
@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'))
        else:
            flash('Invalid email or password!', 'error')
            return render_template('login.html')
    return render_template('login.html')

@app.route('/logout')
def logout():
    session.pop('email', None)
    flash('You have been logged out.', 'info')
    return redirect(url_for('index'))
```

```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)
```

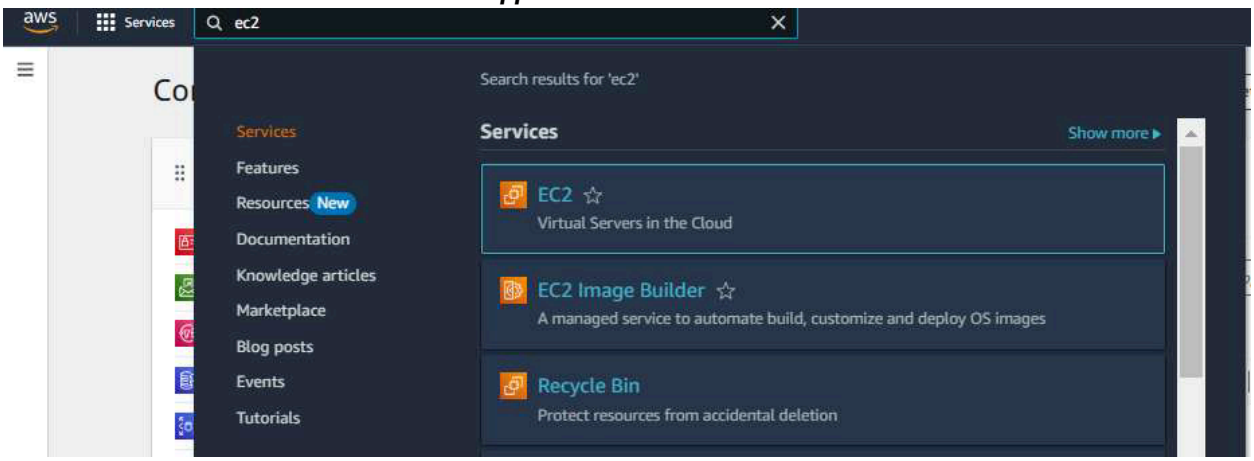
```python
@app.route('/train')
def train():
    if 'email' not in session:
        return redirect(url_for('login'))
    return render_template('train.html')

@app.route('/confirm_train_details')
def confirm_train_details():
    if 'email' not in session:
        return redirect(url_for('login'))

    booking_details = {
        'name': request.args.get('name'),
        'train_number': request.args.get('trainNumber'),
        'source': request.args.get('source'),
        'destination': request.args.get('destination'),
        'departure_time': request.args.get('departureTime'),
        'arrival_time': request.args.get('arrivalTime'),
        'price_per_person': Decimal(request.args.get('price')),
        'travel_date': request.args.get('date'),
        'num_persons': int(request.args.get('persons')),
        'item_id': request.args.get('trainId'), # This is the train ID
        'booking_type': 'train',
        'user_email': session['email'],
        'total_price': Decimal(request.args.get('price')) * int(request.args.get('persons'))
    }
```

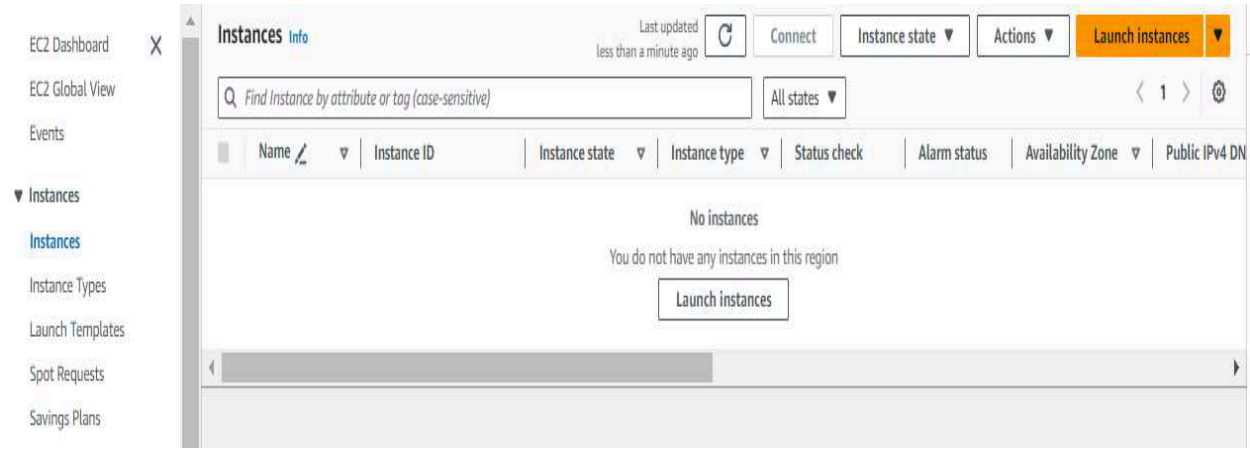
```python
@app.route('/cancel_booking', methods=['POST'])
def cancel_booking():
    if 'email' not in session:
        return redirect(url_for('login'))

    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'))
```

Start the Flask server to listen on all network interfaces (0.0.0.0) at port 5000 with debug mode enabled for development and testing.

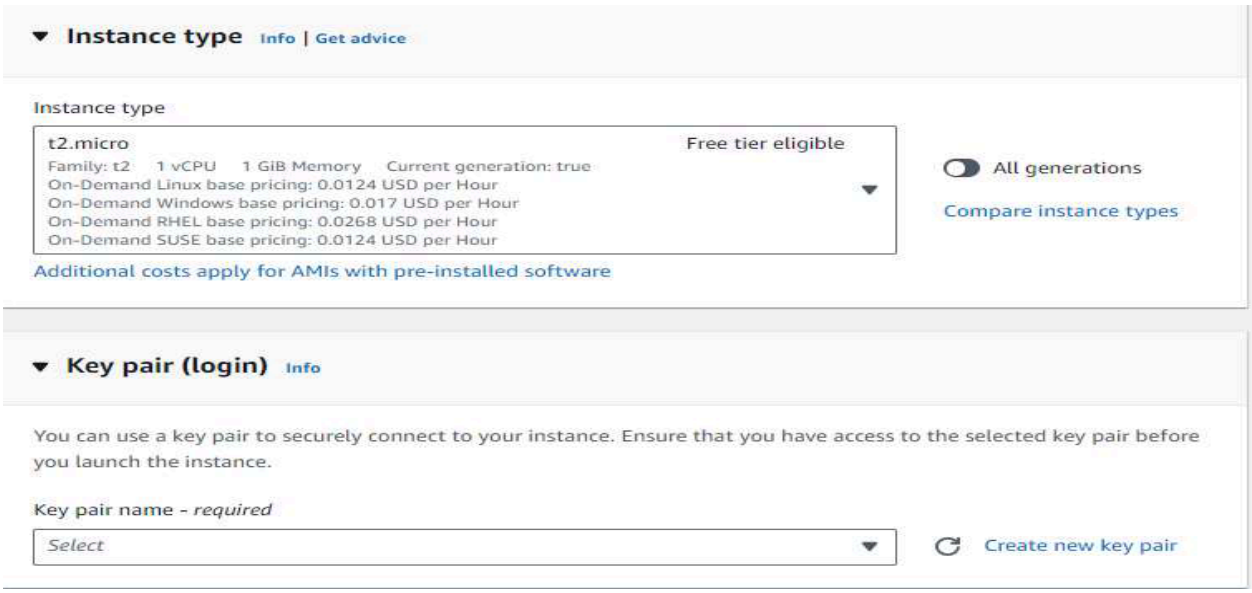
static	Initial commit
templates	Update statistics.html
app.py	Update app.py

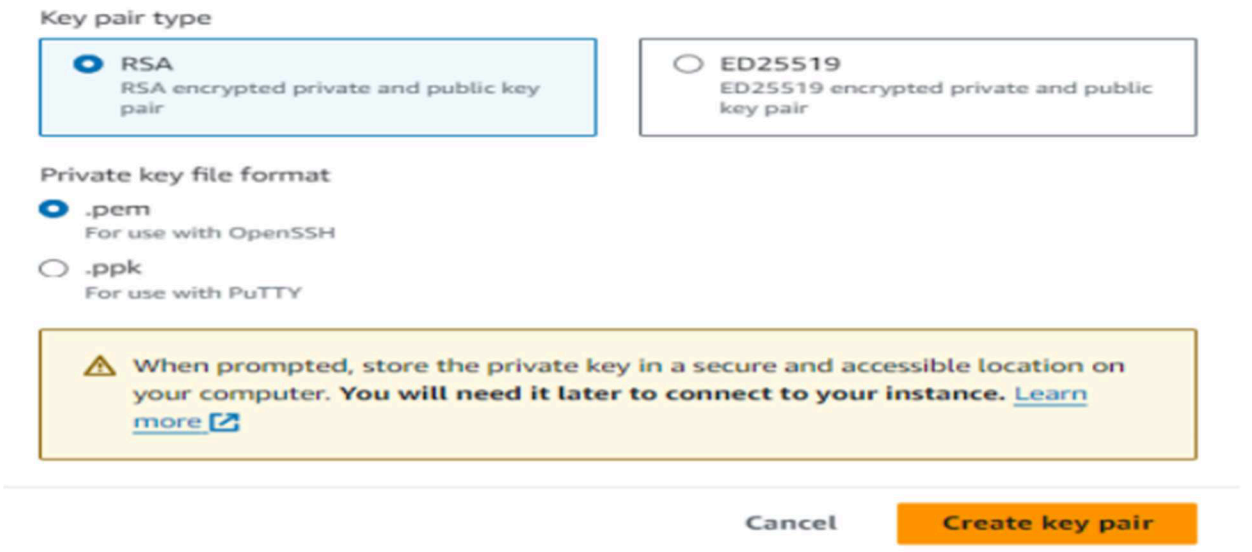
Launch an EC2 instance to host the Flask application.

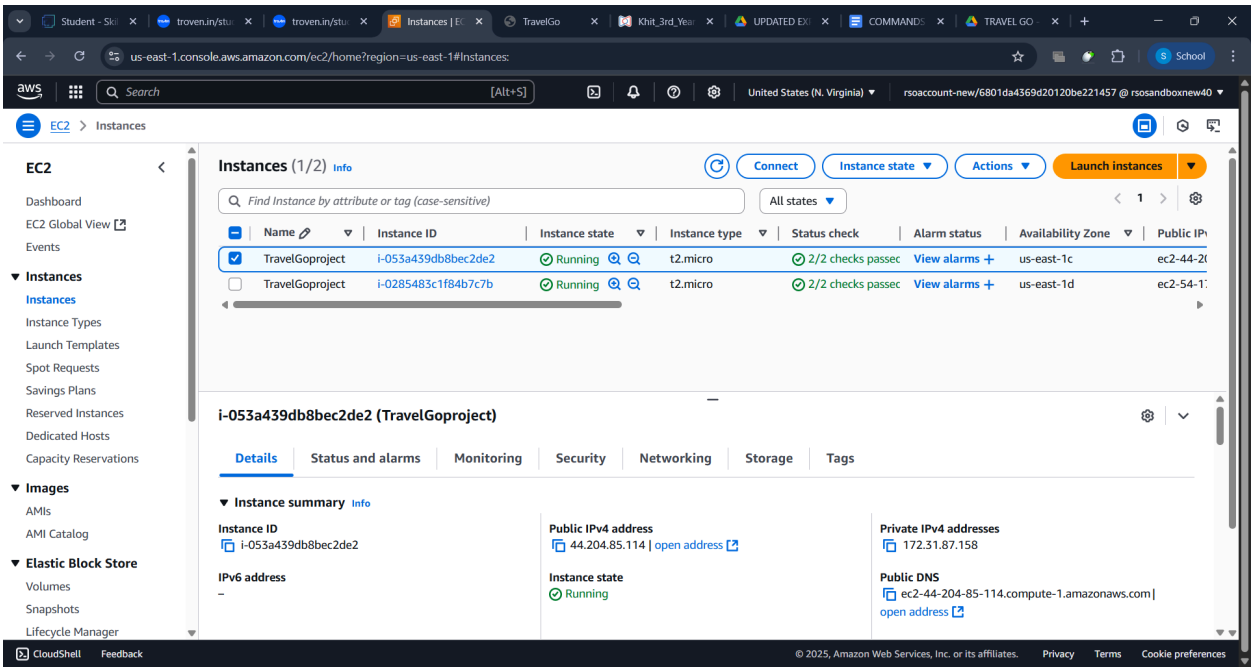




1.Modify IAM roles
2.create Key Pair


Instance type Info | Get advice
Instance type
t2.micro
Free tier eligible
Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Linux base pricing: 0.0124 USD per Hour
On-Demand Windows base pricing: 0.017 USD per Hour
On-Demand RHEL base pricing: 0.0268 USD per Hour
On-Demand SUSE base pricing: 0.0124 USD per Hour
All generations
Compare instance types
Additional costs apply for AMIs with pre-installed software
Key pair (login) Info
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.
Key pair name - required
Select
Create new key pair


Key pair type
RSA
RSA encrypted private and public key pair
ED25519
ED25519 encrypted private and public key pair
Private key file format
.pem
For use with OpenSSH
.ppk
For use with PuTTY
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. Learn more
Cancel
Create key pair

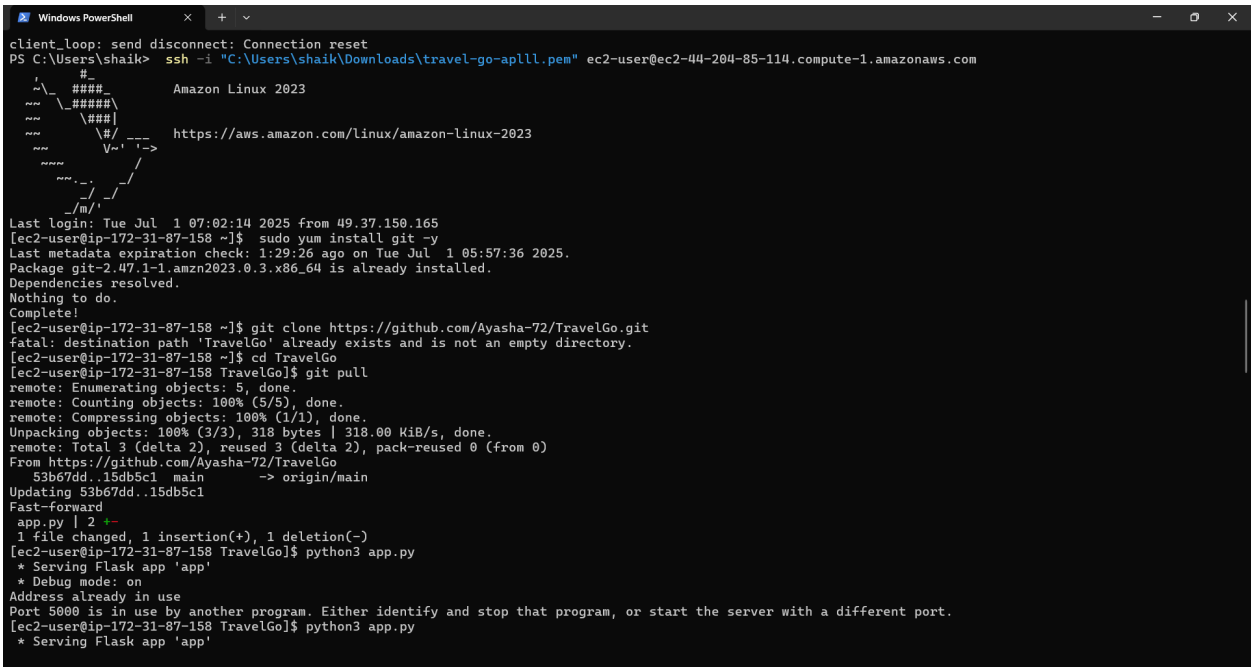


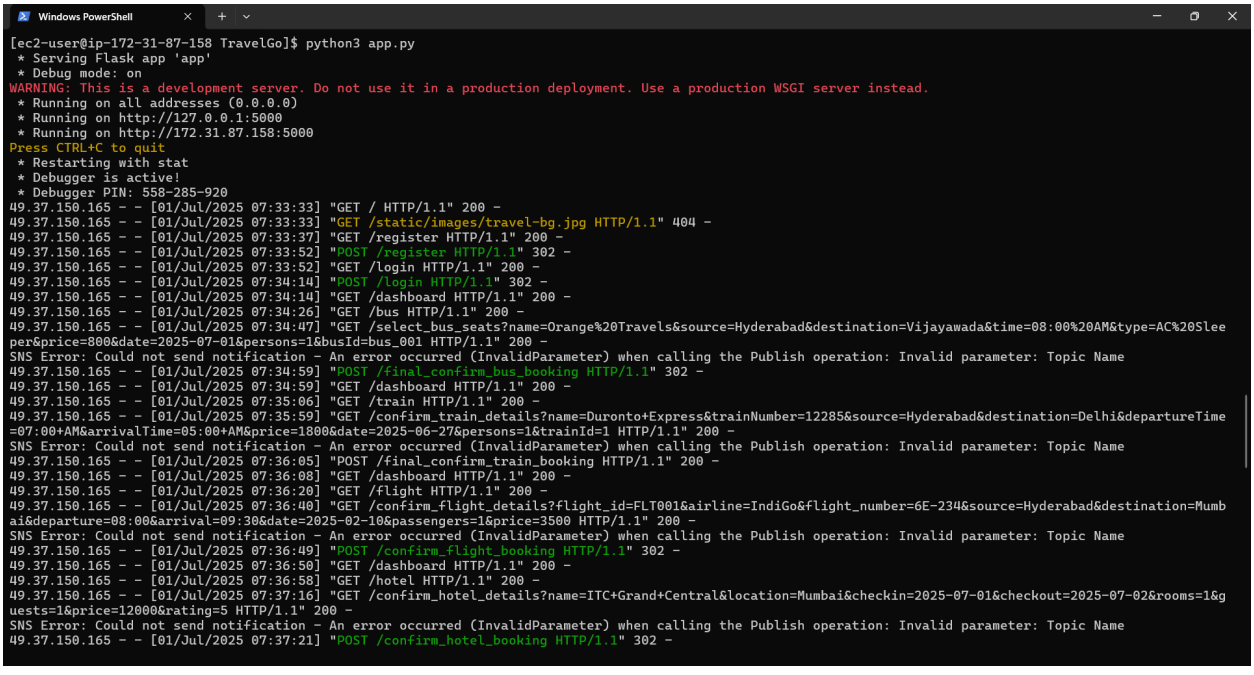
To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and selecting Modify IAM Role to attach the appropriate role.

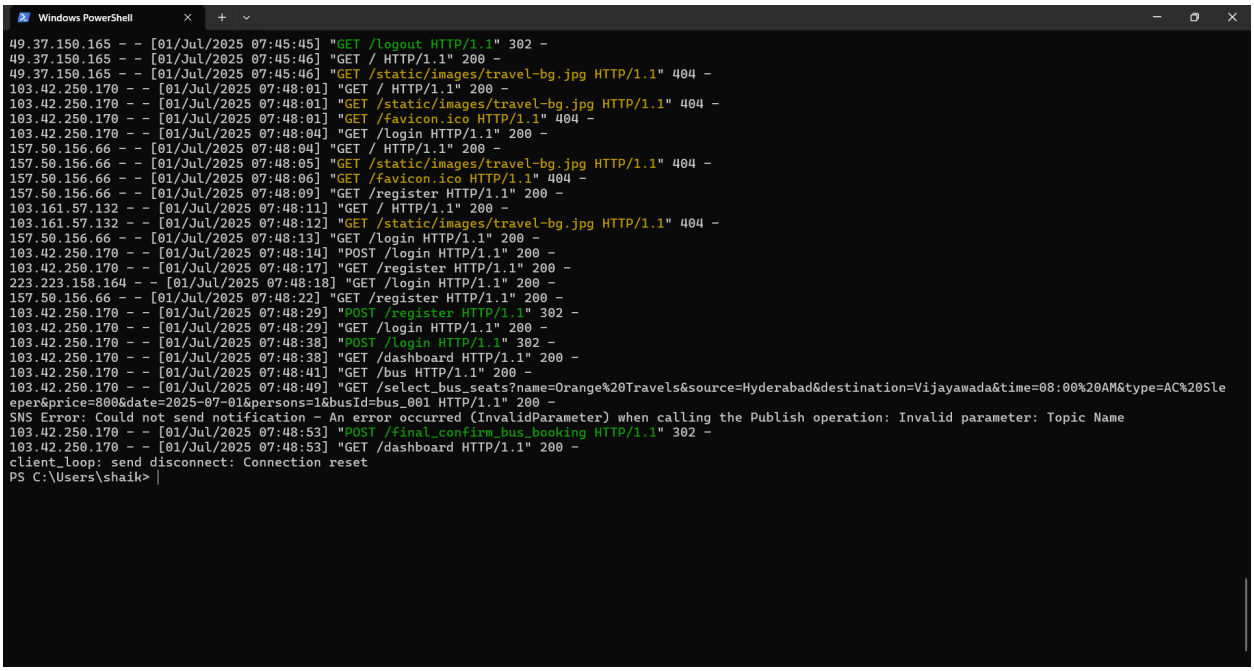
After the IAM role is connected, navigate to the EC2 section in the AWS Management Console. Select the EC2 instance you wish to connect to. At the top of the EC2 Dashboard, click the Connect button. From the connection methods presented, choose EC2 Instance Connect.

Finally, click Connect again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.

Connecting to EC2 With Power shell:


Windows PowerShell
client_loop: send disconnect: Connection reset
PS C:\Users\shaik> ssh -i "C:\Users\shaik\Downloads\travel-go-aplll.pem" ec2-user@ec2-44-204-85-114.compute-1.amazonaws.com
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
Last login: Tue Jul 1 07:02:14 2025 from 49.37.150.165
[ec2-user@ip-172-31-87-158 ~]$ sudo yum install git -y
Last metadata expiration check: 1:29:26 ago on Tue Jul 1 05:57:36 2025.
Package git-2.47.1-1.amzn2023.0.3.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-87-158 ~]$ git clone https://github.com/Ayasha-72/TravelGo.git
fatal: destination path 'TravelGo' already exists and is not an empty directory.
[ec2-user@ip-172-31-87-158 ~]$ cd TravelGo
[ec2-user@ip-172-31-87-158 TravelGo]$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (1/1), done.
Unpacking objects: 100% (3/3), 318 bytes | 318.00 KiB/s, done.
remote: Total 3 (delta 2), reused 3 (delta 2), pack-reused 0 (from 0)
From https://github.com/Ayasha-72/TravelGo
53b67dd..15db5c1 main -> origin/main
Updating 53b67dd..15db5c1
Fast-forward
app.py | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)
[ec2-user@ip-172-31-87-158 TravelGo]$ python3 app.py
* Serving Flask app 'app'
* Debug mode: on
Address already in use
Port 5000 is in use by another program. Either identify and stop that program, or start the server with a different port.
[ec2-user@ip-172-31-87-158 TravelGo]$ python3 app.py
* Serving Flask app 'app'


Windows PowerShell
[ec2-user@ip-172-31-87-158 TravelGo]$ python3 app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.31.87.158:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 558-285-920
49.37.150.165 - - [01/Jul/2025 07:33:33] "GET / HTTP/1.1" 200 -
49.37.150.165 - - [01/Jul/2025 07:33:33] "GET /static/images/travel-bg.jpg HTTP/1.1" 404 -
49.37.150.165 - - [01/Jul/2025 07:33:37] "GET /register HTTP/1.1" 200 -
49.37.150.165 - - [01/Jul/2025 07:33:52] "POST /register HTTP/1.1" 302 -
49.37.150.165 - - [01/Jul/2025 07:33:52] "GET /login HTTP/1.1" 200 -
49.37.150.165 - - [01/Jul/2025 07:34:14] "POST /login HTTP/1.1" 302 -
49.37.150.165 - - [01/Jul/2025 07:34:14] "GET /dashboard HTTP/1.1" 200 -
49.37.150.165 - - [01/Jul/2025 07:34:26] "GET /bus HTTP/1.1" 200 -
49.37.150.165 - - [01/Jul/2025 07:34:47] "GET /select_bus_seats?name=Orange%20Travels&source=Hyderabad&destination=Vijayawada&time=08:00%20AM&type=AC%20Sleeper&price=800&date=2025-07-01&persons=1&busId=bus_001 HTTP/1.1" 200 -
SNS Error: Could not send notification - An error occurred (InvalidParameter) when calling the Publish operation: Invalid parameter: Topic Name
49.37.150.165 - - [01/Jul/2025 07:34:59] "POST /final_confirm_bus_booking HTTP/1.1" 302 -
49.37.150.165 - - [01/Jul/2025 07:34:59] "GET /dashboard HTTP/1.1" 200 -
49.37.150.165 - - [01/Jul/2025 07:35:06] "GET /train HTTP/1.1" 200 -
49.37.150.165 - - [01/Jul/2025 07:35:59] "GET /confirm_train_details?name=Duronto+Express&trainNumber=12285&source=Hyderabad&destination=Delhi&departureTime=07:00+AM&arrivalTime=05:00+AM&price=1800&date=2025-06-27&persons=1&trainId=1 HTTP/1.1" 200 -
SNS Error: Could not send notification - An error occurred (InvalidParameter) when calling the Publish operation: Invalid parameter: Topic Name
49.37.150.165 - - [01/Jul/2025 07:36:05] "POST /final_confirm_train_booking HTTP/1.1" 200 -
49.37.150.165 - - [01/Jul/2025 07:36:08] "GET /dashboard HTTP/1.1" 200 -
49.37.150.165 - - [01/Jul/2025 07:36:20] "GET /flight HTTP/1.1" 200 -
49.37.150.165 - - [01/Jul/2025 07:36:40] "GET /confirm_flight_details?flight_id=FLT001&airline=IndiGo&flight_number=6E-234&source=Hyderabad&destination=Mumbai&departure=08:00&arrival=09:30&date=2025-02-10&passengers=1&price=3500 HTTP/1.1" 200 -
SNS Error: Could not send notification - An error occurred (InvalidParameter) when calling the Publish operation: Invalid parameter: Topic Name
49.37.150.165 - - [01/Jul/2025 07:36:49] "POST /confirm_flight_booking HTTP/1.1" 302 -
49.37.150.165 - - [01/Jul/2025 07:36:50] "GET /dashboard HTTP/1.1" 200 -
49.37.150.165 - - [01/Jul/2025 07:36:58] "GET /hotel HTTP/1.1" 200 -
49.37.150.165 - - [01/Jul/2025 07:37:16] "GET /confirm_hotel_details?name=ITC+Grand+Central&location=Mumbai&checkin=2025-07-01&checkout=2025-07-02&rooms=1&guests=1&price=12000&rating=5 HTTP/1.1" 200 -
SNS Error: Could not send notification - An error occurred (InvalidParameter) when calling the Publish operation: Invalid parameter: Topic Name
49.37.150.165 - - [01/Jul/2025 07:37:21] "POST /confirm_hotel_booking HTTP/1.1" 302 -

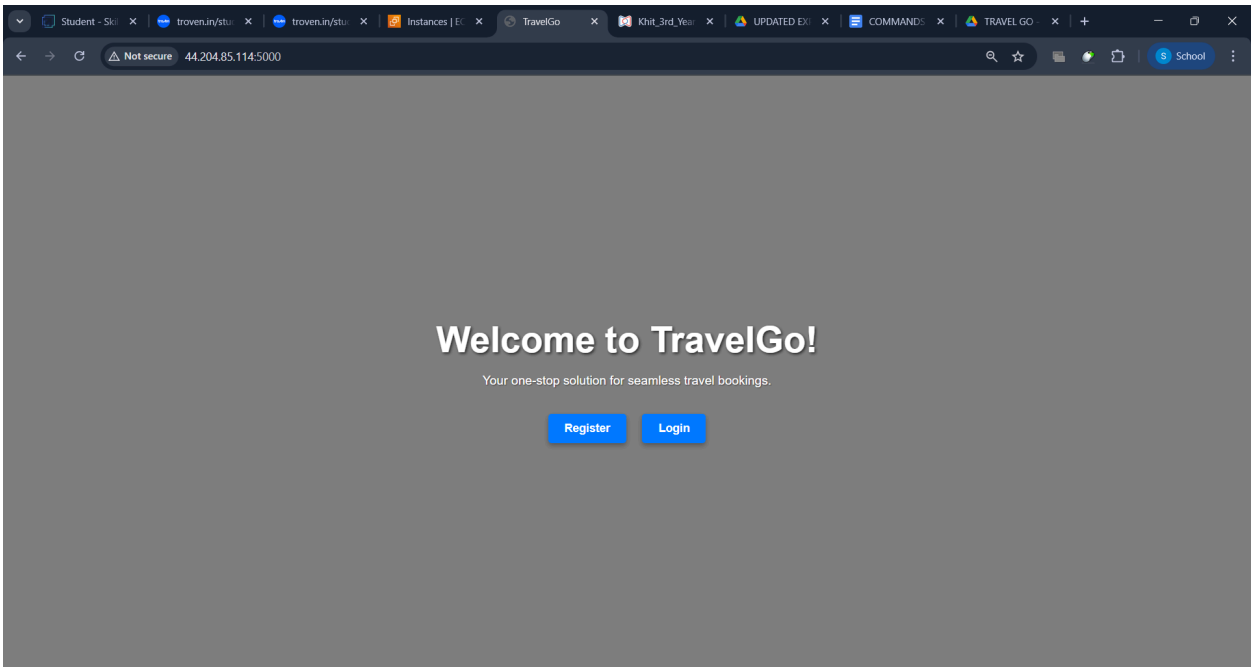


Commands Used:

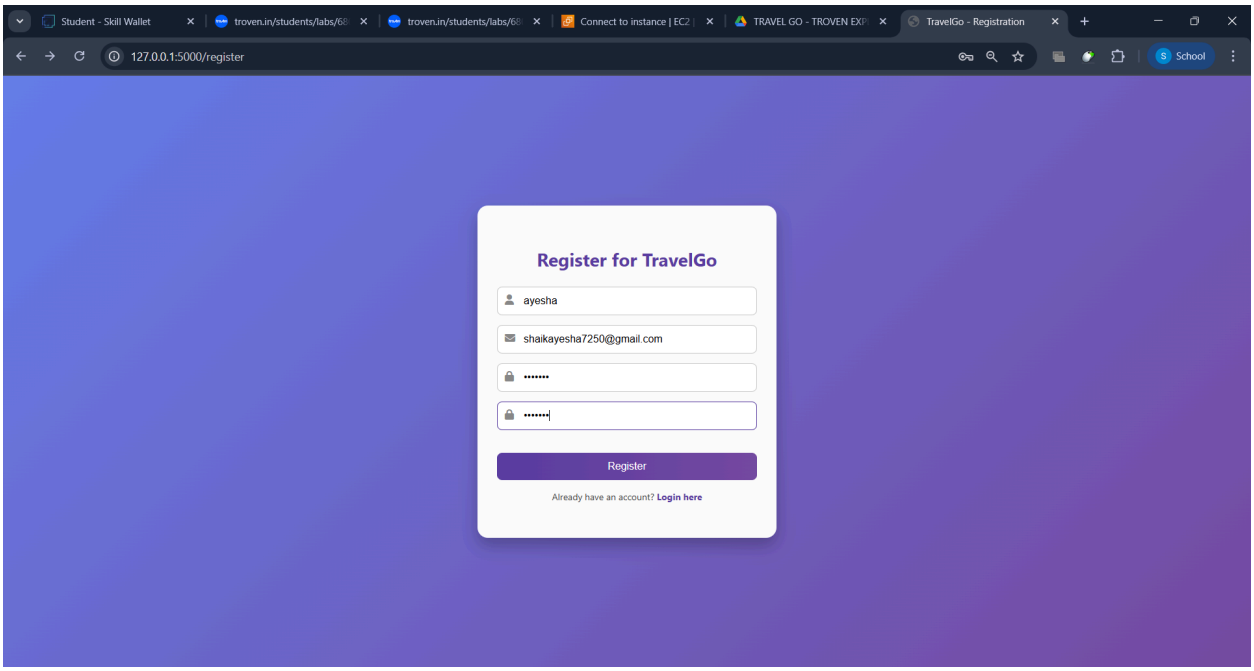
1. *sudo yum install git -y*
2. *git clone you_github_repository*
3. *cd your_repository_name*
4. *sudo yum install python3 -y*
5. *sudo yum install python3-pip -y*
6. *pip install flask*
7. *pip install boto3*
8. *python3 app.py*

Testing and Deployment:

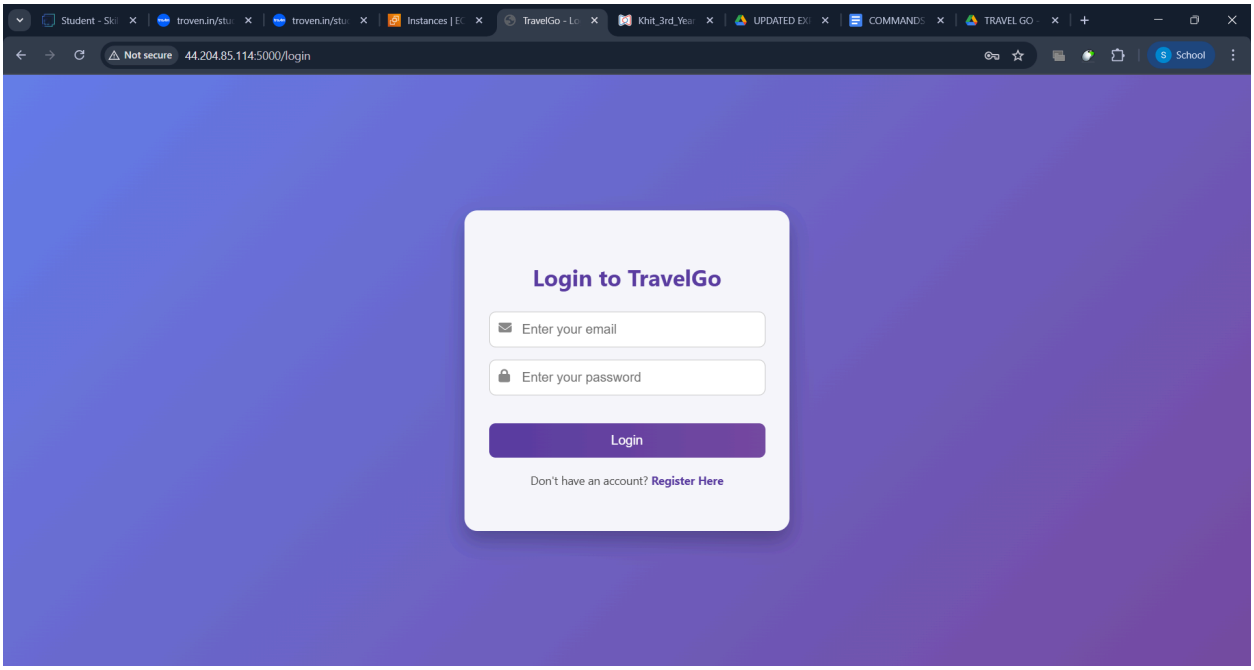
Home Page:



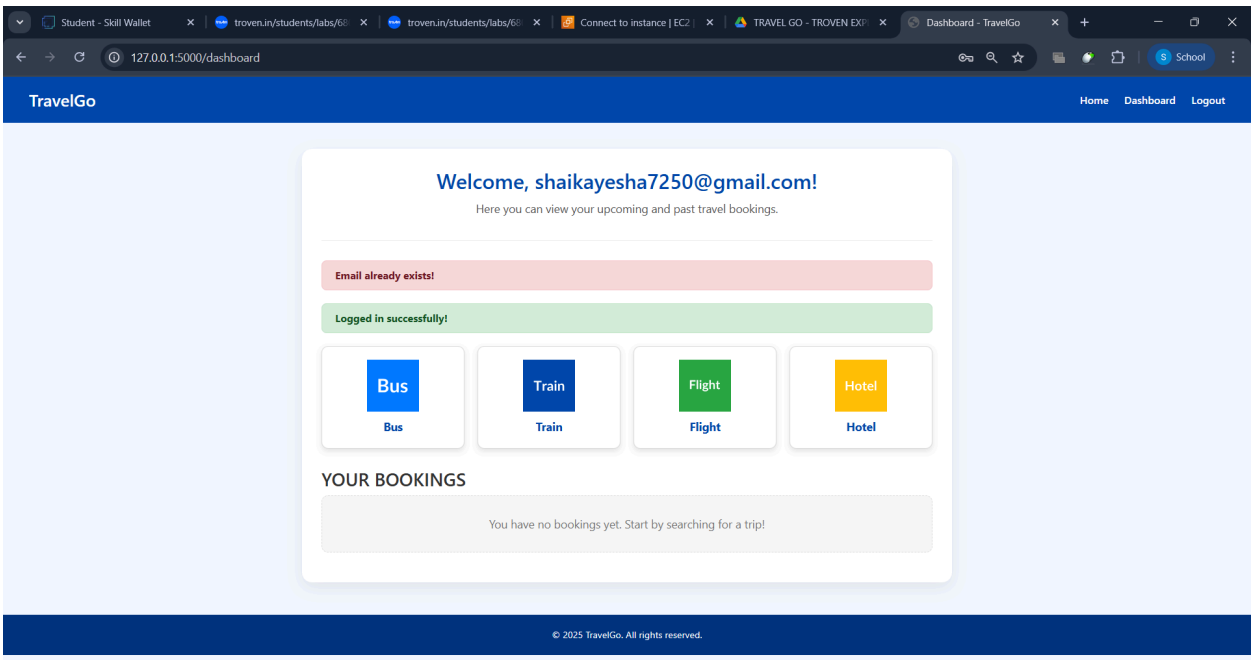
Registration Page:



Login page:

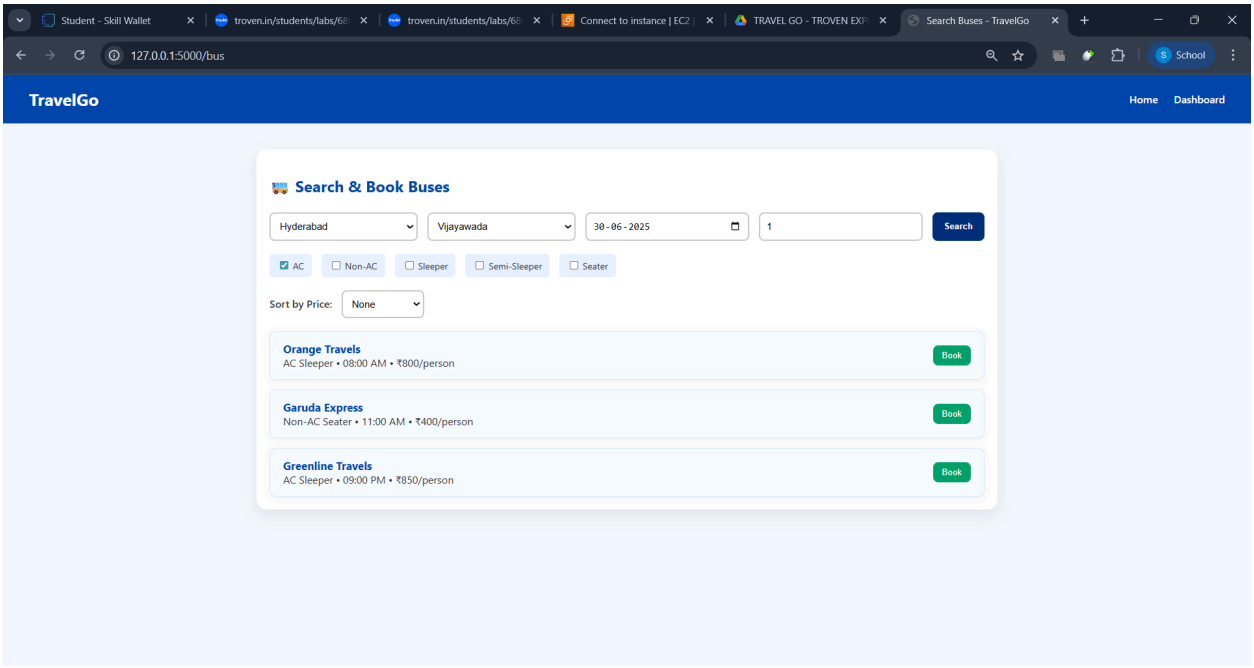


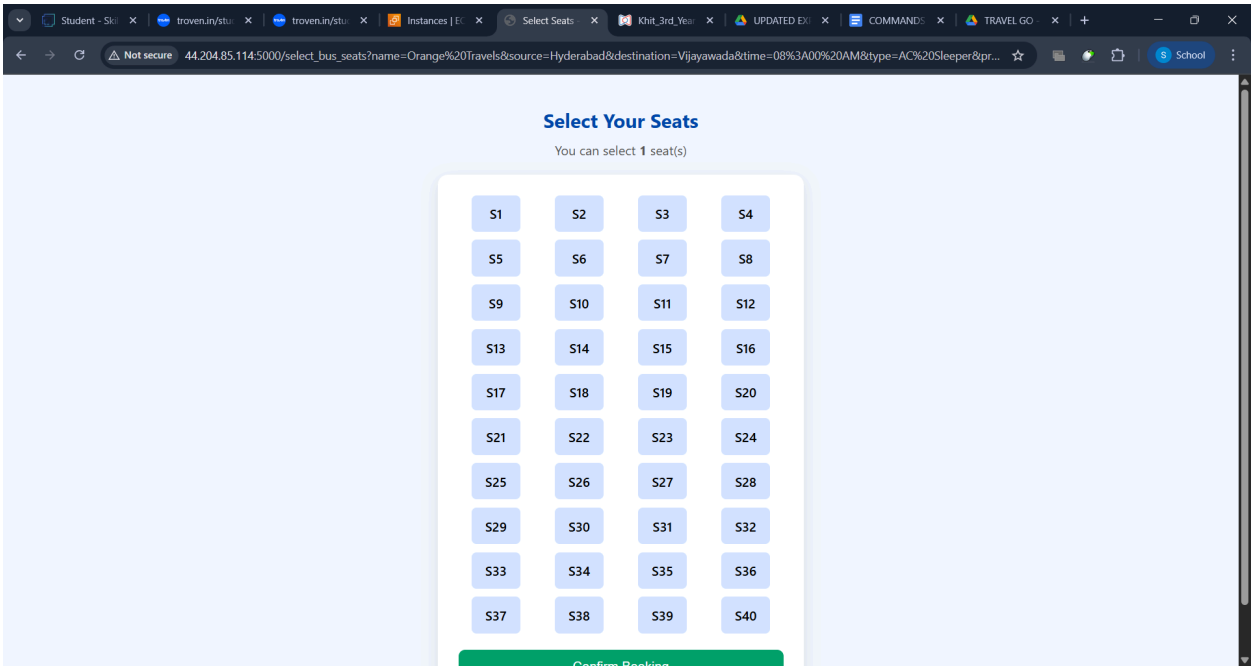
Dashboard Page:



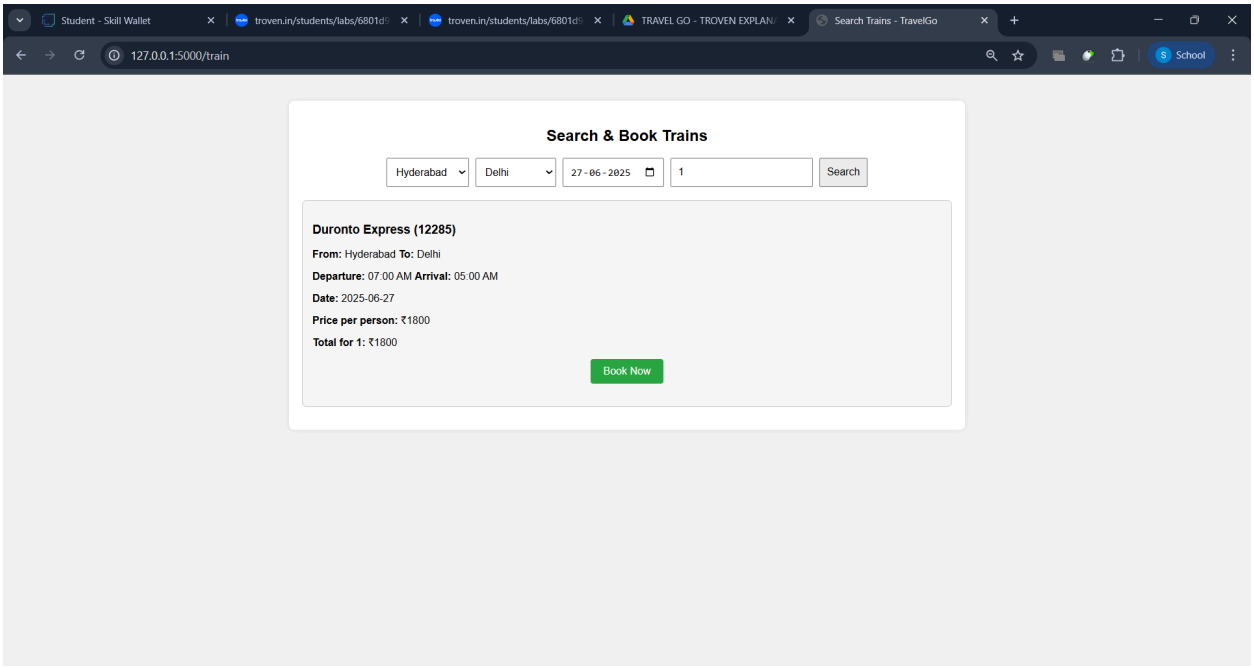
Booking pages:

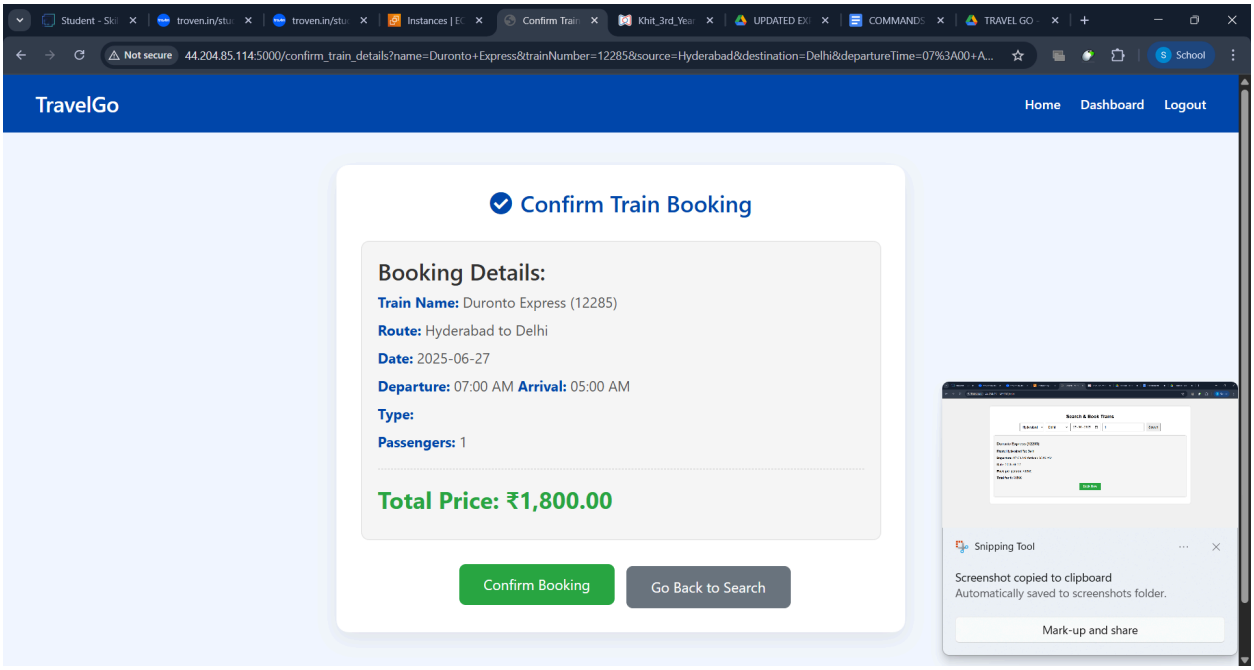
Bus:



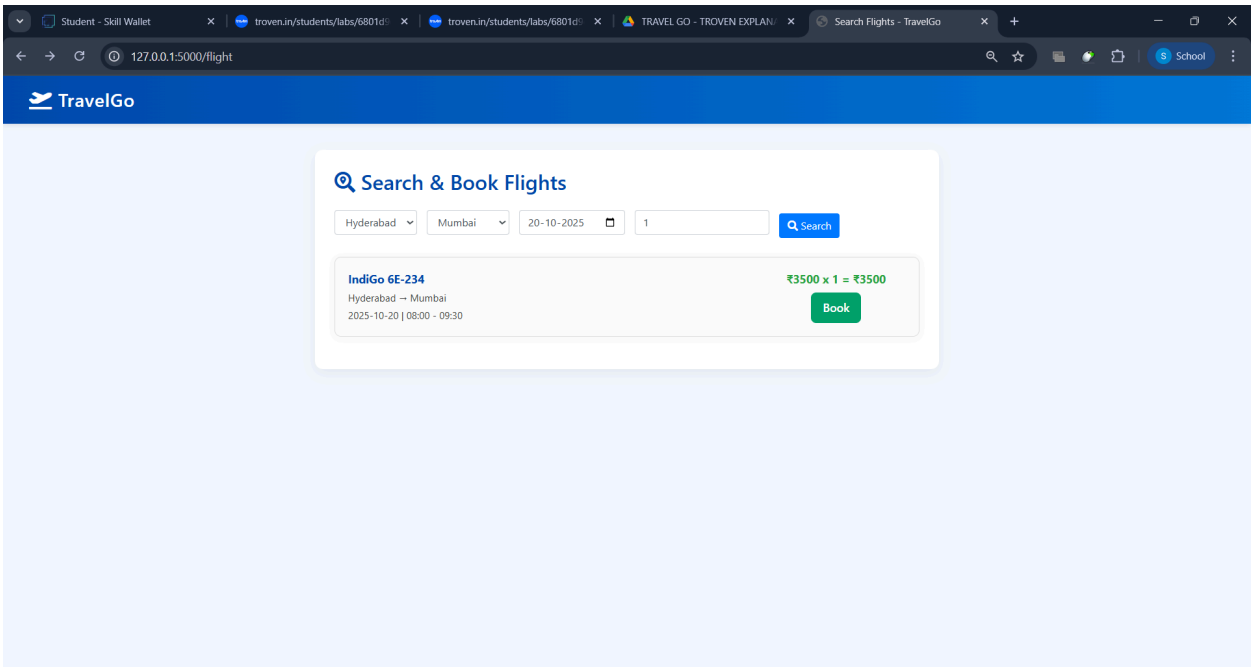


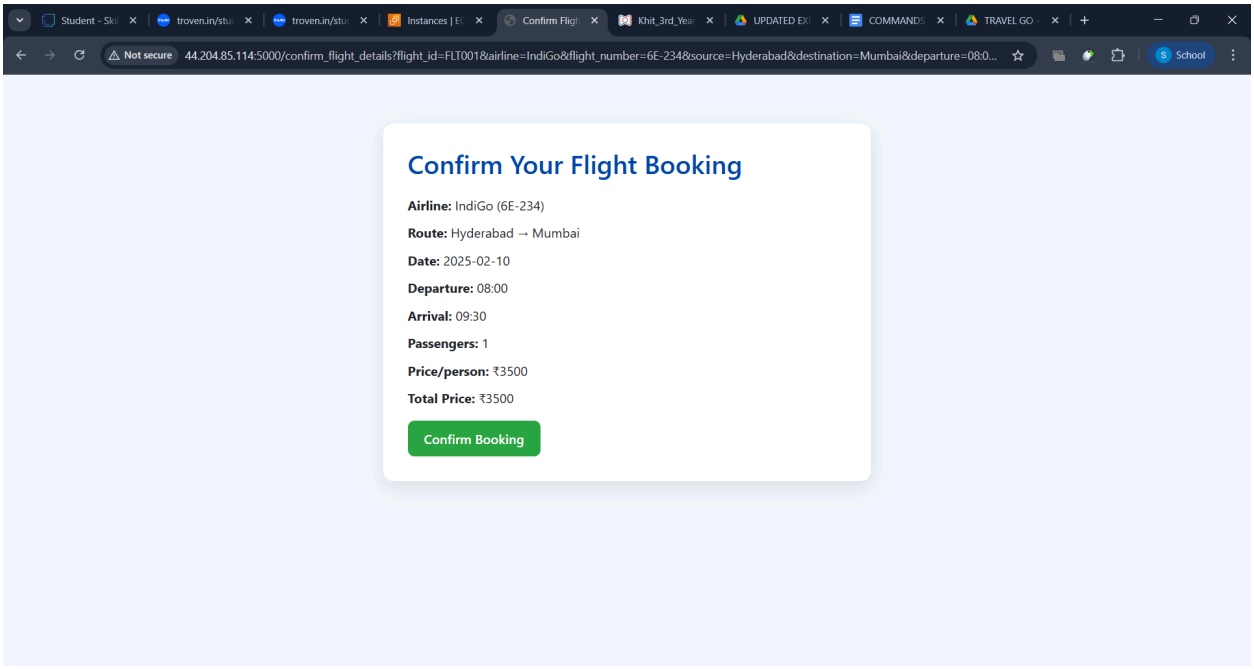
Train:


Search Trains - TravelGo
127.0.0.1:5000/train
Search & Book Trains
Hyderabad
Delhi
27-06-2025
1
Search
Duronto Express (12285)
From: Hyderabad To: Delhi
Departure: 07:00 AM Arrival: 05:00 AM
Date: 2025-06-27
Price per person: ₹1800
Total for 1: ₹1800
Book Now

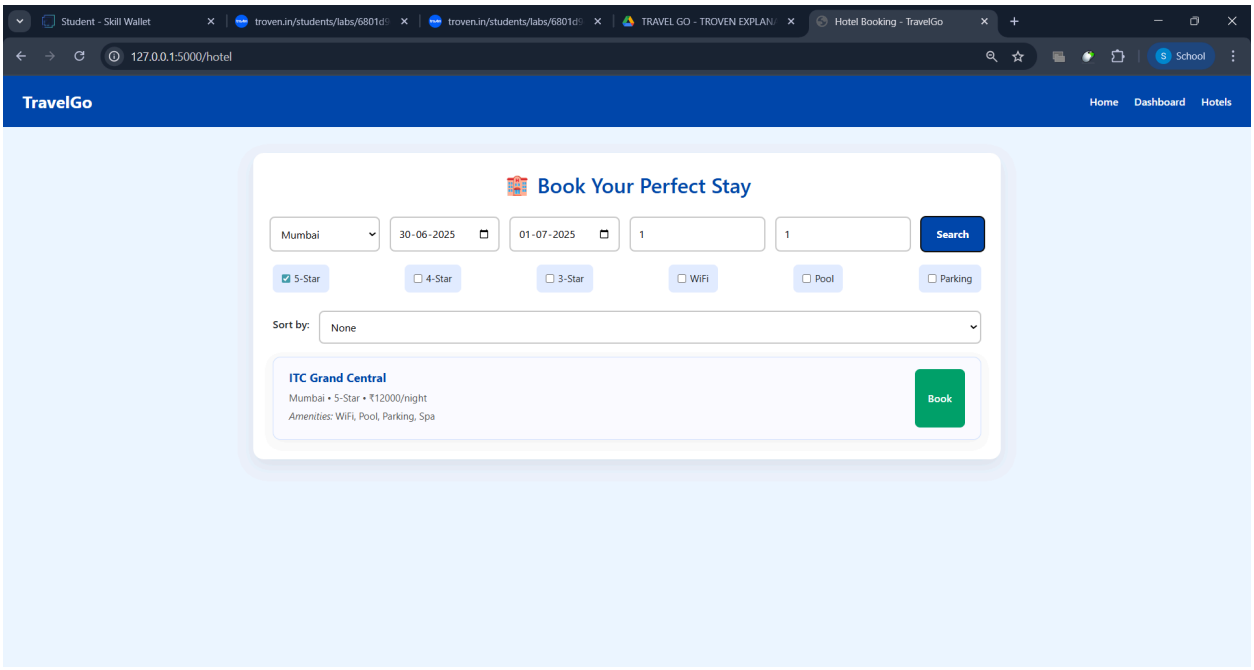

Not secure
44.204.85.114:5000/confirm_train_details?name=Duronto+Express&trainNumber=12285&source=Hyderabad&destination=Delhi&departureTime=07%3A00+A...
TravelGo
Home
Dashboard
Logout
Confirm Train Booking
Booking Details:
Train Name: Duronto Express (12285)
Route: Hyderabad to Delhi
Date: 2025-06-27
Departure: 07:00 AM Arrival: 05:00 AM
Type:
Passengers: 1
Total Price: ₹1,800.00
Confirm Booking
Go Back to Search
Snipping Tool
Screenshot copied to clipboard
Automatically saved to screenshots folder.
Mark-up and share

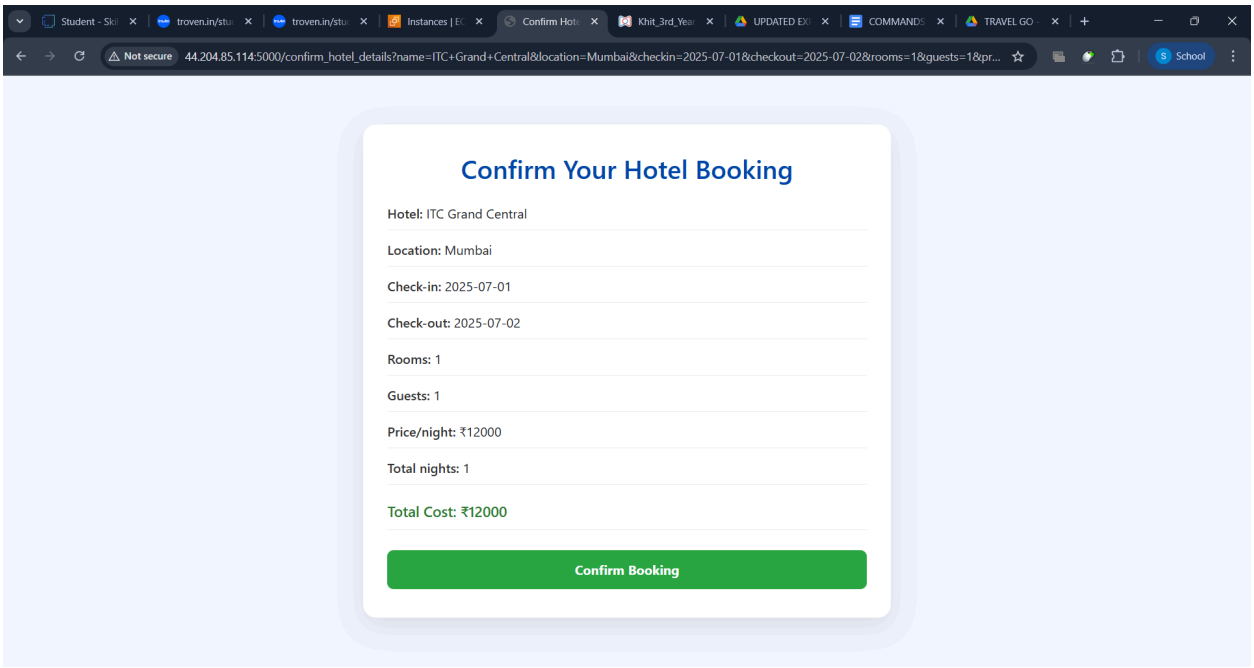
Flight:


Student - Skill Wallet
Search Flights - TravelGo
127.0.0.1:5000/flight
School
TravelGo
Search & Book Flights
Hyderabad
Mumbai
20-10-2025
1
Search
IndiGo 6E-234
Hyderabad → Mumbai
2025-10-20 | 08:00 - 09:30
₹3500 x 1 = ₹3500
Book

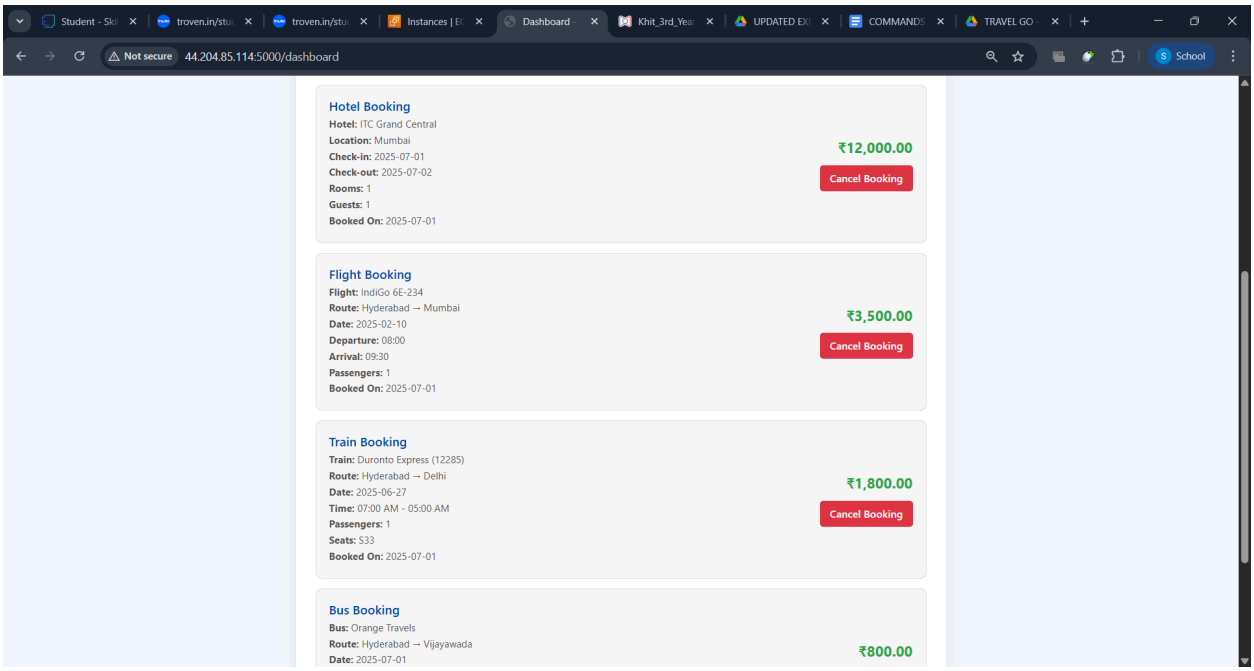

Not secure
44.204.85.114:5000/confirm_flight_details?flight_id=FLT001&airline=IndiGo&flight_number=6E-234&source=Hyderabad&destination=Mumbai&departure=08:0...
School
Confirm Your Flight Booking
Airline: IndiGo (6E-234)
Route: Hyderabad → Mumbai
Date: 2025-02-10
Departure: 08:00
Arrival: 09:30
Passengers: 1
Price/person: ₹3500
Total Price: ₹3500
Confirm Booking

Hotel:


Student - Skill Wallet
Hotel Booking - TravelGo
127.0.0.1:5000/hotel
School
TravelGo
Home
Dashboard
Hotels
Book Your Perfect Stay
Mumbai
30-06-2025
01-07-2025
1
1
Search
5-Star
4-Star
3-Star
WiFi
Pool
Parking
Sort by:
None
ITC Grand Central
Mumbai • 5-Star • ₹12000/night
Amenities: WiFi, Pool, Parking, Spa
Book


Not secure
44.204.85.114:5000/confirm_hotel_details?name=ITC+Grand+Central&location=Mumbai&checkin=2025-07-01&checkout=2025-07-02&rooms=1&guests=1&pr...
School
Confirm Your Hotel Booking
Hotel: ITC Grand Central
Location: Mumbai
Check-in: 2025-07-01
Check-out: 2025-07-02
Rooms: 1
Guests: 1
Price/night: ₹12000
Total nights: 1
Total Cost: ₹12000
Confirm Booking

My Bookings:



Conclusion:

The Travel Go project successfully delivers a robust, cloud-powered travel booking platform that simplifies the process of booking buses, trains, flights, and hotels for users in real time. By integrating Flask for backend development, AWS EC2 for reliable deployment, DynamoDB for scalable data storage, and AWS SNS for instant notifications, the system offers a seamless and responsive user experience.

From user authentication to booking management and real-time alerts, Travel Go addresses the critical challenges of modern travel planning with efficiency and flexibility. The platform's architecture ensures scalability, security, and performance—making it a practical solution for travelers and administrators alike.

This project demonstrates how cloud technologies can be effectively used to build smart, user-centric applications that operate smoothly under real-world conditions.