

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.

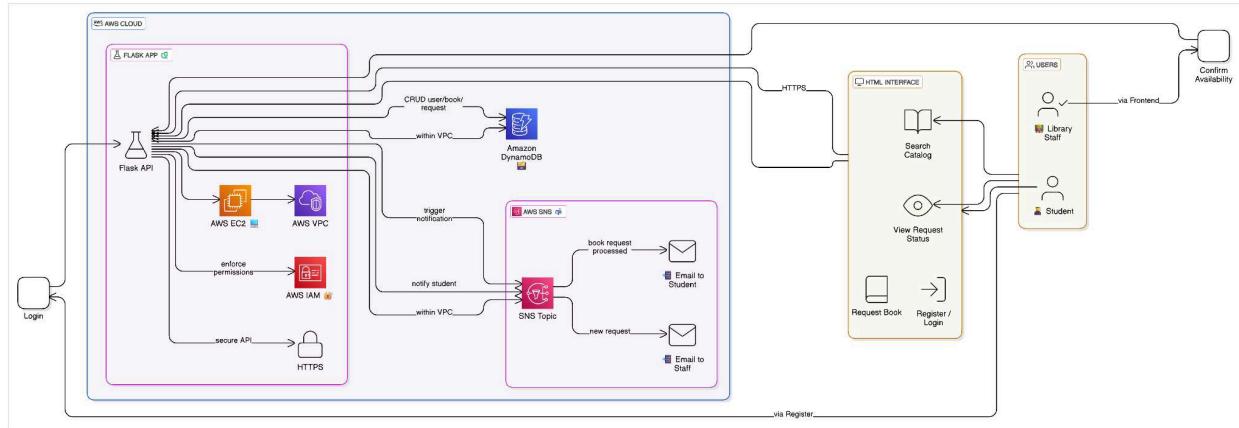
Scenario 1: Real-Time Booking Experience for Users Using Travel Go, a user logs into the platform and selects the travel category (bus/train/flight/hotel), and submits their booking preferences. Flask handles the backend logic—fetching available options from DynamoDB based on user input. Once the user 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.

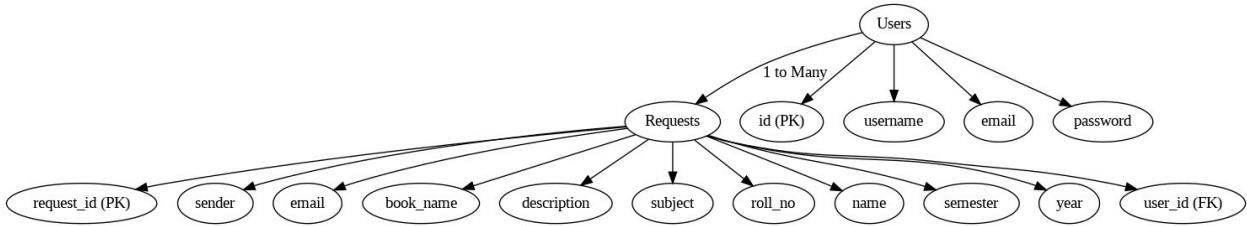
Scenario 2: Automated Email Confirmations and Alerts. After confirming a booking, Travel Go uses AWS SNS to send instant email notifications to the user with booking details. For instance, a user books a flight, and Flask captures the transaction while AWS SNS sends an email confirming the booking to 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.

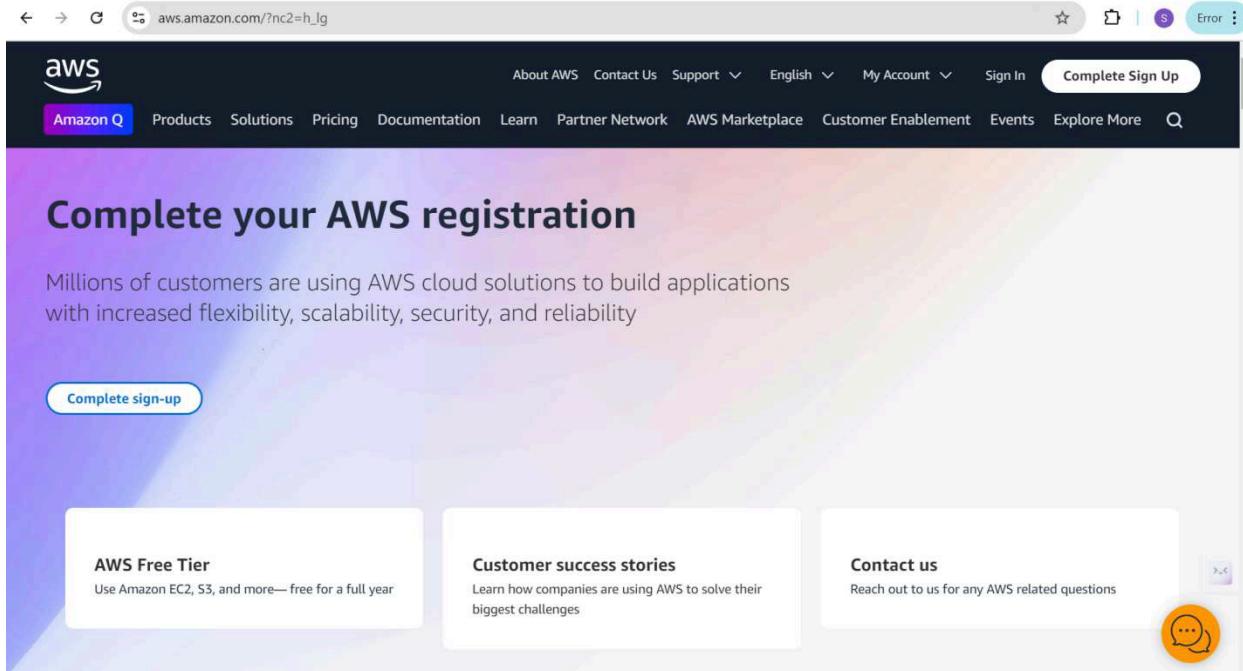
7. Deployment on EC2

Activity 7.1: Upload Flask Files

Activity 7.2: Run the Flask App

8. Testing and Deployment

AWS Account Setup and Login



The screenshot shows the AWS registration sign-up page. At the top, there's a navigation bar with links for About AWS, Contact Us, Support, English, My Account, Sign In, and Complete Sign Up. Below the navigation is a search bar labeled "Amazon Q" and a menu with options like Products, Solutions, Pricing, Documentation, Learn, Partner Network, AWS Marketplace, Customer Enablement, Events, Explore More, and a magnifying glass icon.

Complete your AWS registration

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

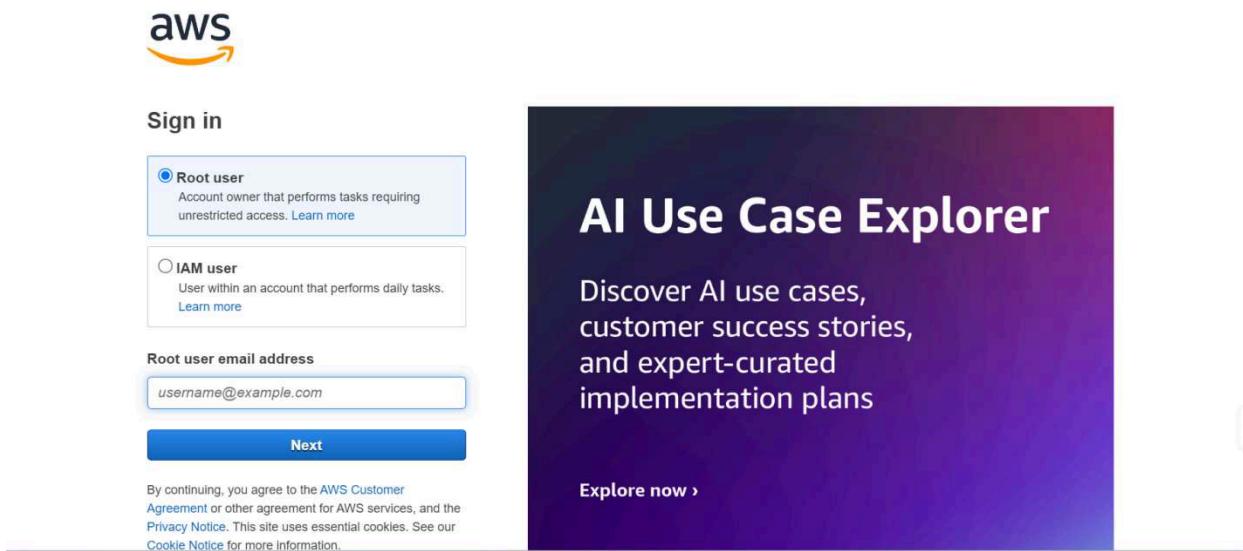
[Complete sign-up](#)

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

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

Contact us
Reach out to us for any AWS related questions

Log in to the AWS Management Console



The screenshot shows the AWS sign-in page. It features the AWS logo at the top left. Below it is a "Sign in" section with two radio button options: "Root user" (selected) and "IAM user". The "Root user" option is described as an account owner that performs tasks requiring unrestricted access. The "IAM user" option is described as a user within an account that performs daily tasks. A "Root user email address" input field contains "username@example.com". A blue "Next" button is located below the input fields. At the bottom of the sign-in form, there is a small note about cookie consent and legal agreements.

AI Use Case Explorer

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

[Explore now >](#)

IAM role creation:

The screenshot shows the AWS IAM Dashboard. On the left, a sidebar lists 'Access management' options like User groups, Roles, Policies, and Identity providers. The main area displays 'IAM resources' and 'AWS Account' sections, both containing 'Access denied' error messages. A prominent blue banner at the top right says 'New access analyzers available' with a 'Create new analyzer' button. The bottom status bar shows the URL as <https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles>.

The screenshot shows the 'Create role' wizard, Step 1: 'Select trusted entity'. It has three tabs: Step 1 (selected), Step 2 (Add permissions), and Step 3 (Name, review, and create). The 'Trusted entity type' section contains five options: 'AWS service' (selected), 'AWS account', 'Web identity', 'SAML 2.0 federation', and 'Custom trust policy'. Below this is a 'Use case' section with a dropdown set to 'EC2'. The bottom status bar shows the URL as <https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles/create>.

Identity and Access Management (IAM)

Summary

Creation date
July 01, 2025, 10:54 (UTC+05:30)

Last activity
11 minutes ago

ARN
arn:aws:iam::980921720508:role/Studentuser

Maximum session duration
1 hour

Permissions | Trust relationships | Tags | Last Accessed | Revoke sessions

Permissions policies (3) Info

You can attach up to 10 managed policies.

Policy name	Type	Attached entities
AmazonDynamoDBFullAccess	AWS managed	2
AmazonEC2FullAccess	AWS managed	2
AmazonSNSFullAccess	AWS managed	2

Permissions boundary (not set)

DynamoDB Database Creation and Setup

Search results for 'dyn'

Services

- Features
- Resources **New**
- Documentation
- Knowledge articles
- Marketplace
- Blog posts
- Events
- Tutorials

Services

- DynamoDB** ☆
Managed NoSQL Database
- Top features: Tables, Imports from S3, Explore Items, Clusters, Reserved Capacity
- Amazon DocumentDB** ☆
Fully-managed MongoDB-compatible database service
- CloudFront** ☆
Global Content Delivery Network
- Athena** ☆
Serverless interactive analytics service

Features

- Settings**
DynamoDB feature
- Clusters**
DynamoDB feature

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

The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. In the 'Table details' section, the table name is set to 'travelgo_users'. The 'Partition key' is defined as 'email' of type 'String'. In the 'Table settings' section, the 'Default settings' tab is selected. The table has 6 items listed in the main pane.

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.

The screenshot shows the 'Tables' page in the AWS DynamoDB console, displaying a list of 6 tables:

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite
bookings	Active	user_email (\$)	booking_date (\$)	0	0	Off	Off
bookings1	Active	user_email (\$)	booking_date (\$)	0	0	Off	Off
train1	Active	train_number (\$)	date (\$)	0	0	Off	Off
trains	Active	train_number (\$)	date (\$)	0	0	Off	Off
TravelGo_users	Active	email (\$)	-	0	0	Off	Off
travelgo_users1	Active	email (\$)	-	0	0	Off	Off

SNS Notification Setup

Create SNS topics for sending email notifications to users.

The screenshot shows the AWS search interface with the query 'sns' entered in the search bar. The results are categorized under 'Services' and 'Features'. The top result is 'Simple Notification Service' (SNS), which is described as 'SNS managed message topics for Pub/Sub'. Below it are other services like Route 53 Resolver and AWS End User Messaging. Under 'Features', there are sections for 'Events' (ElastiCache feature), 'SMS' (AWS End User Messaging feature), and 'Hosted zones' (Route 53 feature).

Search results for 'sns'

Services

Simple Notification Service ☆
SNS managed message topics for Pub/Sub

Route 53 Resolver
Resolve DNS queries in your Amazon VPC and on-premises network.

Route 53 ☆
Scalable DNS and Domain Name Registration

AWS End User Messaging ☆
Engage your customers across multiple communication channels

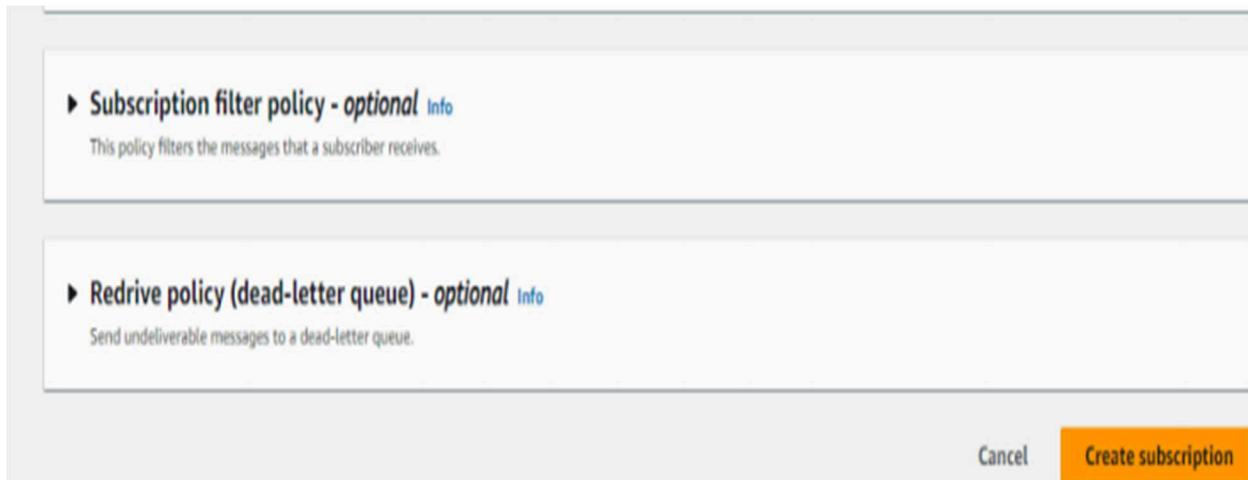
Features

Events
ElastiCache feature

SMS
AWS End User Messaging feature

Hosted zones
Route 53 feature

- 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:aws:sns:us-east-1:980921720508:TravelGo:172076c6-71c6-46c7-b6c1-a7dba0f164e
- Click on create subscription



Amazon SNS < TravelGo

Details

Name	TravelGo	Display name	-
ARN	arn:aws:sns:us-east-1:980921720508:TravelGo	Topic owner	980921720508
Type	Standard		

Subscriptions (1)

ID	Endpoint	Status	Protocol
172076c6-71c6-46c7-b6c1-a7ddb...	shaikayesha7250@gmail.com	Confirmed	EMAIL

[Edit](#) [Delete](#) [Request confirmation](#) [Confirm subscription](#) Create subscription

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

AWS Notification - Subscription Confirmation Spam x

AWS Notifications <no-reply@sns.amazonaws.com> to me ▾ Tue, Jul 1, 10:59 AM (6 days ago) star smile undo more

Why is this message in spam? This message is similar to messages that were identified as spam in the past.

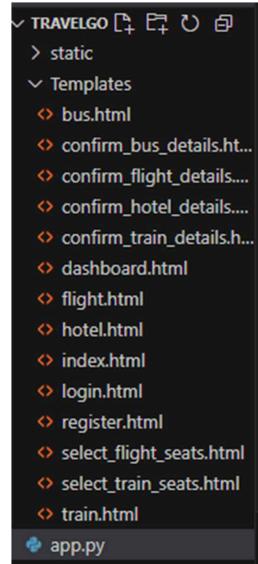
Report not spam info

You have chosen to subscribe to the topic:
`arn:aws:sns:us-east-1:980921720508:TravelGo`

To confirm this subscription, click or visit the link below (if this was in error no action is necessary):
[Confirm subscription](#)

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

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')
```

```

@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'))

```

```

@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)

```

```

@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'))
    }

```

```

@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')

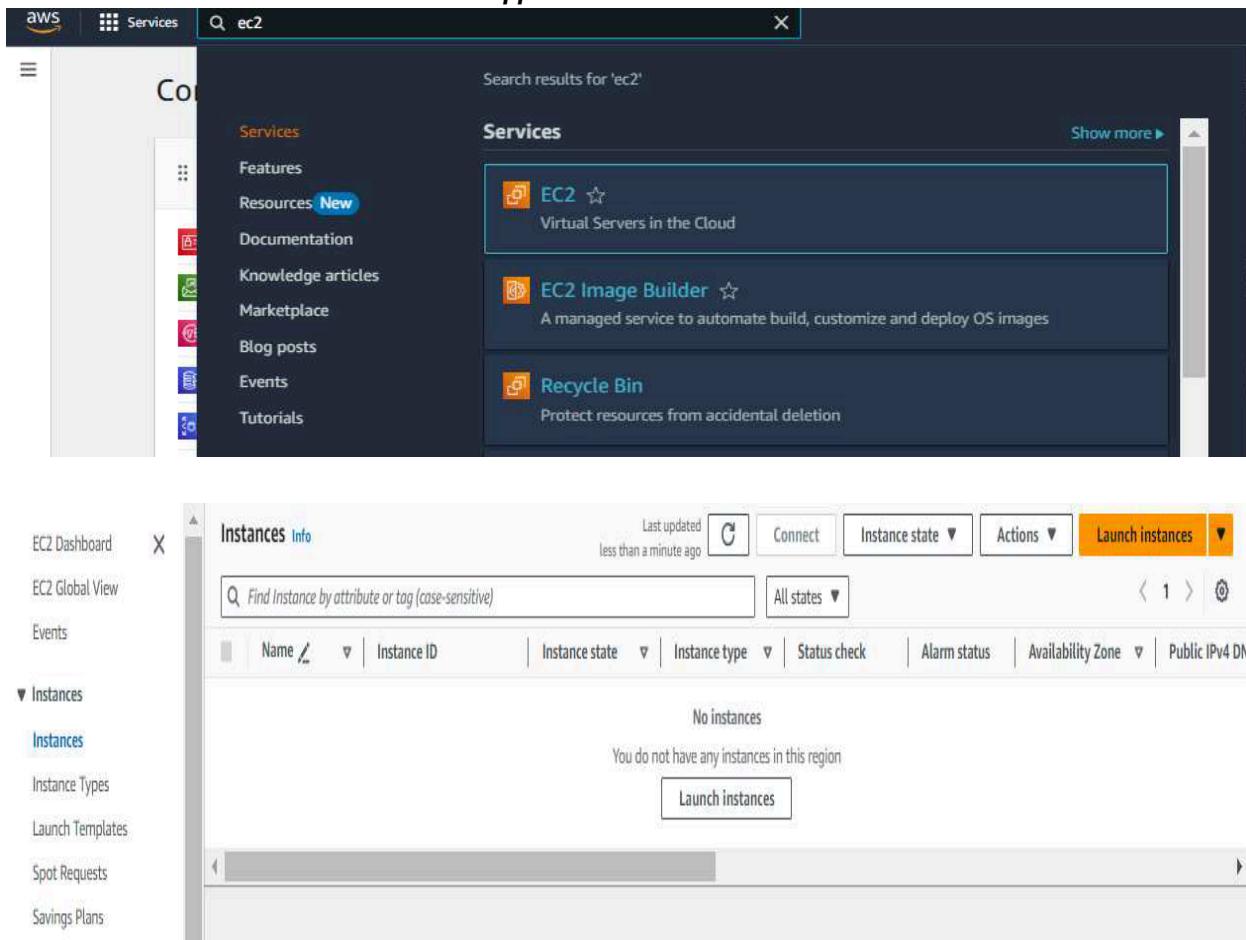
    return redirect(url_for('dashboard'))

```

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.



The screenshot shows two windows side-by-side. The left window is the AWS Cloud9 IDE, displaying a terminal session with the command "git log" output:

```
commit 1e3a2f3 (HEAD, local branch master)
Author: [REDACTED] <[REDACTED]>
Date:   Mon Jun 10 10:45:10 2019 -0700

    Initial commit

commit 1e3a2f3 (HEAD, local branch master)
Author: [REDACTED] <[REDACTED]>
Date:   Mon Jun 10 10:45:10 2019 -0700

    Update statistics.html

commit 1e3a2f3 (HEAD, local branch master)
Author: [REDACTED] <[REDACTED]>
Date:   Mon Jun 10 10:45:10 2019 -0700

    Update app.py
```

The right window is the AWS Management Console, specifically the EC2 service. It shows the search results for 'ec2' and the EC2 service details:

Services

- EC2
- EC2 Image Builder
- Recycle Bin

Instances

No instances

You do not have any instances in this region

Launch instances

1.Modify IAM roles

2.create Key Pair

▼ Instance type [Info](#) | [Get advice](#)

Instance type

t2.micro
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

Free tier eligible

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](#)

The screenshot shows the AWS EC2 Instances page. The left sidebar is collapsed, and the main area displays two EC2 instances under the 'Instances' section. The first instance, 'TravelGoproject i-053a439db8bec2de2', is listed as 'Running' with an 't2.micro' instance type, located in 'us-east-1c' with a public IP 'ec2-44-204-85-114'. The second instance, 'TravelGoproject i-0285483c1f84b7c7b', is also 'Running' with an 't2.micro' instance type, located in 'us-east-1d' with a public IP 'ec2-54-1'. The top navigation bar includes buttons for 'Connect', 'Actions', and 'Launch instances'. The bottom of the page includes standard AWS footer links like CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

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 x + 
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
~~~ /_/_/_
~~~ /_/_/_
/_m/.

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..15db5c main       -> origin/main
Updating 53b67dd..15db5c
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 x + 
[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
49.37.158.165 - [01/Jul/2025 07:33:33] "GET / HTTP/1.1" 200 -
49.37.158.165 - [01/Jul/2025 07:33:33] "GET /static/images/travel-bg.jpg HTTP/1.1" 404 -
49.37.158.165 - [01/Jul/2025 07:33:37] "GET /register HTTP/1.1" 200 -
49.37.158.165 - [01/Jul/2025 07:33:52] "POST /register HTTP/1.1" 302 -
49.37.158.165 - [01/Jul/2025 07:33:52] "GET /login HTTP/1.1" 200 -
49.37.158.165 - [01/Jul/2025 07:34:14] "POST /Login HTTP/1.1" 302 -
49.37.158.165 - [01/Jul/2025 07:34:14] "GET /dashboard HTTP/1.1" 200 -
49.37.158.165 - [01/Jul/2025 07:34:26] "GET /bus HTTP/1.1" 200 -
49.37.158.165 - [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
49.37.158.165 - [01/Jul/2025 07:34:59] "POST /final_confirm_bus_booking HTTP/1.1" 302 -
49.37.158.165 - [01/Jul/2025 07:34:59] "GET /dashboard HTTP/1.1" 200 -
49.37.158.165 - [01/Jul/2025 07:35:06] "GET /train HTTP/1.1" 200 -
49.37.158.165 - [01/Jul/2025 07:35:59] "GET /confirm_train_details?name=Duronto+Express&trainNumber=l2285&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
49.37.158.165 - [01/Jul/2025 07:36:05] "POST /final_confirm_train_booking HTTP/1.1" 200 -
49.37.158.165 - [01/Jul/2025 07:36:08] "GET /dashboard HTTP/1.1" 200 -
49.37.158.165 - [01/Jul/2025 07:36:26] "GET /flight HTTP/1.1" 200 -
49.37.158.165 - [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
49.37.158.165 - [01/Jul/2025 07:36:49] "POST /confirm_flight_booking HTTP/1.1" 302 -
49.37.158.165 - [01/Jul/2025 07:36:50] "GET /dashboard HTTP/1.1" 200 -
49.37.158.165 - [01/Jul/2025 07:36:58] "GET /hotel HTTP/1.1" 200 -
49.37.158.165 - [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
49.37.158.165 - [01/Jul/2025 07:37:21] "POST /confirm_hotel_booking HTTP/1.1" 302 -
```

```
| Windows PowerShell | + -
```

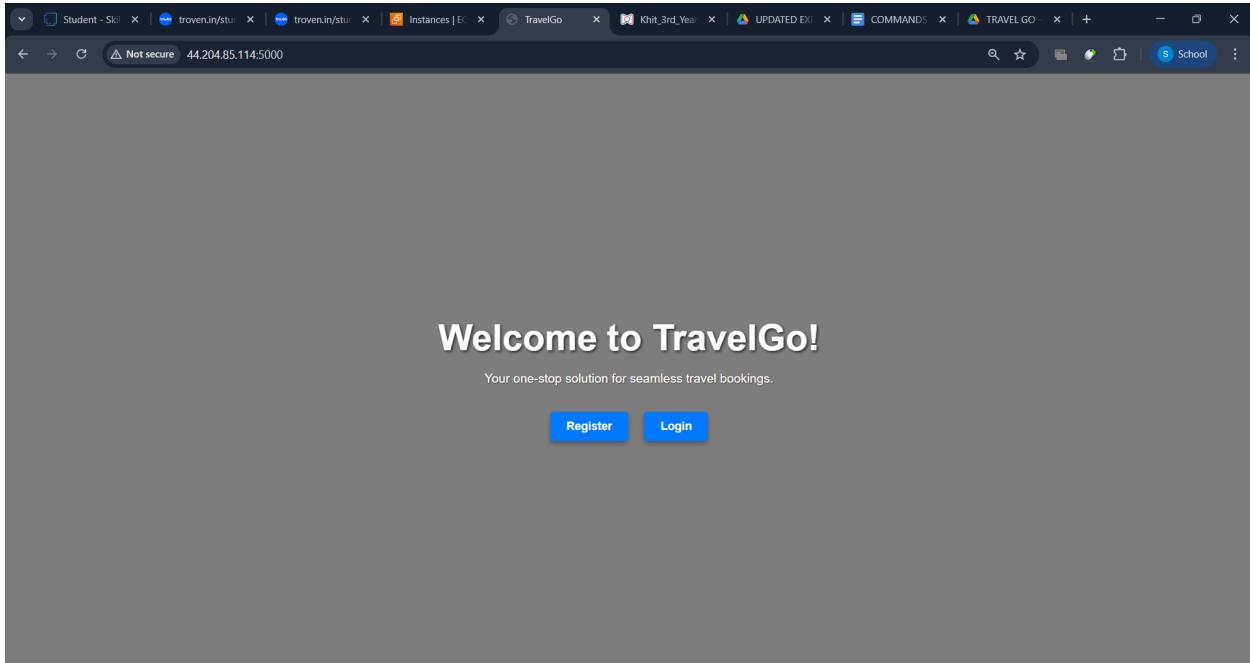
```
49.37.150.165 -- [01/Jul/2025 07:45:45] "GET /logout HTTP/1.1" 302 -
49.37.150.165 -- [01/Jul/2025 07:45:46] "GET / HTTP/1.1" 200 -
49.37.150.165 -- [01/Jul/2025 07:45:46] "GET /static/images/travel-bg.jpg HTTP/1.1" 404 -
103.42.250.170 -- [01/Jul/2025 07:48:01] "GET / HTTP/1.1" 200 -
103.42.250.170 -- [01/Jul/2025 07:48:01] "GET /static/images/travel-bg.jpg HTTP/1.1" 404 -
103.42.250.170 -- [01/Jul/2025 07:48:01] "GET /favicon.ico HTTP/1.1" 404 -
103.42.250.170 -- [01/Jul/2025 07:48:04] "GET /Login HTTP/1.1" 200 -
157.50.156.66 -- [01/Jul/2025 07:48:04] "GET /static/images/travel-bg.jpg HTTP/1.1" 404 -
157.50.156.66 -- [01/Jul/2025 07:48:06] "GET /favicon.ico HTTP/1.1" 404 -
157.50.156.66 -- [01/Jul/2025 07:48:09] "GET /register HTTP/1.1" 200 -
103.161.57.132 -- [01/Jul/2025 07:48:11] "GET / HTTP/1.1" 200 -
103.161.57.132 -- [01/Jul/2025 07:48:12] "GET /static/images/travel-bg.jpg HTTP/1.1" 404 -
157.50.156.66 -- [01/Jul/2025 07:48:13] "GET /login HTTP/1.1" 200 -
103.42.250.170 -- [01/Jul/2025 07:48:14] "POST /login HTTP/1.1" 200 -
103.42.250.170 -- [01/Jul/2025 07:48:17] "GET /register HTTP/1.1" 200 -
223.223.158.164 -- [01/Jul/2025 07:48:18] "GET /login HTTP/1.1" 200 -
157.50.156.66 -- [01/Jul/2025 07:48:22] "GET /register HTTP/1.1" 200 -
103.42.250.170 -- [01/Jul/2025 07:48:29] "POST /register HTTP/1.1" 302 -
103.42.250.170 -- [01/Jul/2025 07:48:29] "GET /Login HTTP/1.1" 200 -
103.42.250.170 -- [01/Jul/2025 07:48:38] "POST /Login HTTP/1.1" 302 -
103.42.250.170 -- [01/Jul/2025 07:48:38] "GET /dashboard HTTP/1.1" 200 -
103.42.250.170 -- [01/Jul/2025 07:48:41] "GET /bus HTTP/1.1" 200 -
103.42.250.170 -- [01/Jul/2025 07:48:49] "GET /select_bus_seats?name=Orange%20Travels&source=Hyderabad&destination=Vijayawada&time=08:00%20AM&type=AC%20Se
eprice&price=8000&date=2025-07-01&persons=1&busId=bus_001 HTTP/1.1" 200 -
SNS Error: Could not send notification - An error occurred (InvalidOperationException) when calling the Publish operation: Invalid parameter: Topic Name
103.42.250.170 -- [01/Jul/2025 07:48:53] "POST /final_confirm_bus_booking HTTP/1.1" 302 -
103.42.250.170 -- [01/Jul/2025 07:48:53] "GET /dashboard HTTP/1.1" 200 -
client_loop: send disconnect: Connection reset
PS C:\Users\shaik> |
```

Commands Used:

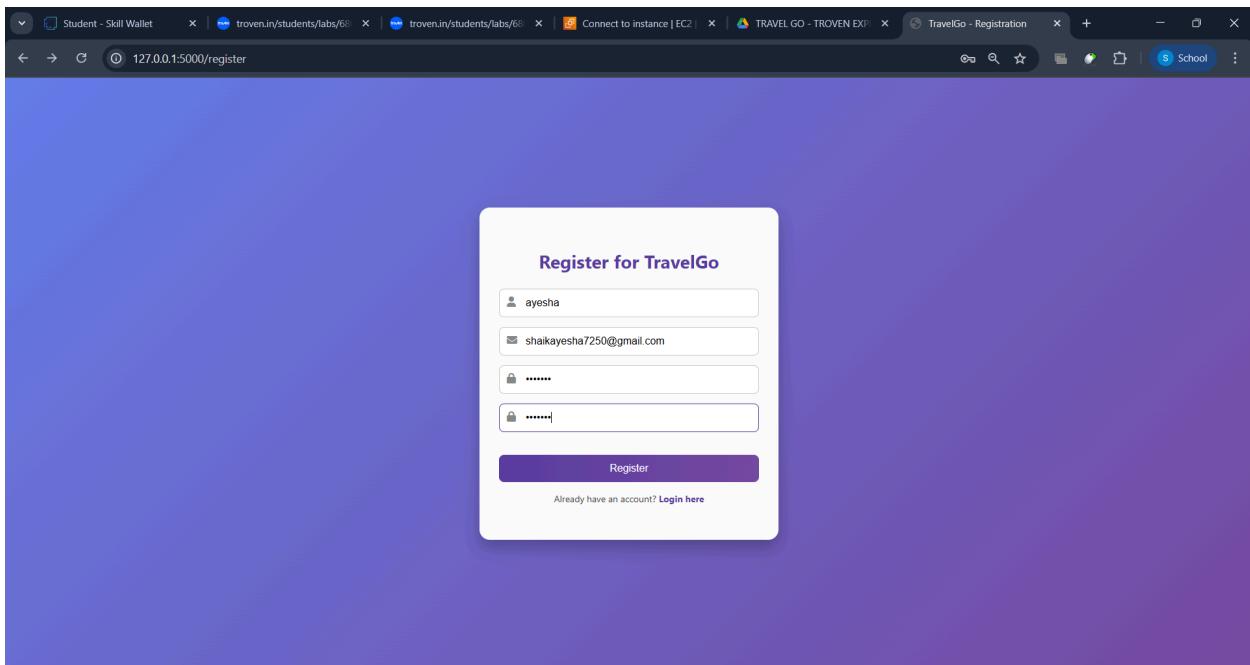
1. `sudo yum install git -y`
2. `git clone you_github_repository`
3. `cd your_repository_name`
4. `sudo yum install python3 -y`
5. `sudo yum install python3-pip -y`
6. `pip install flask`
7. `pip install boto3`
8. `python3 app.py`

Testing and Deployment:

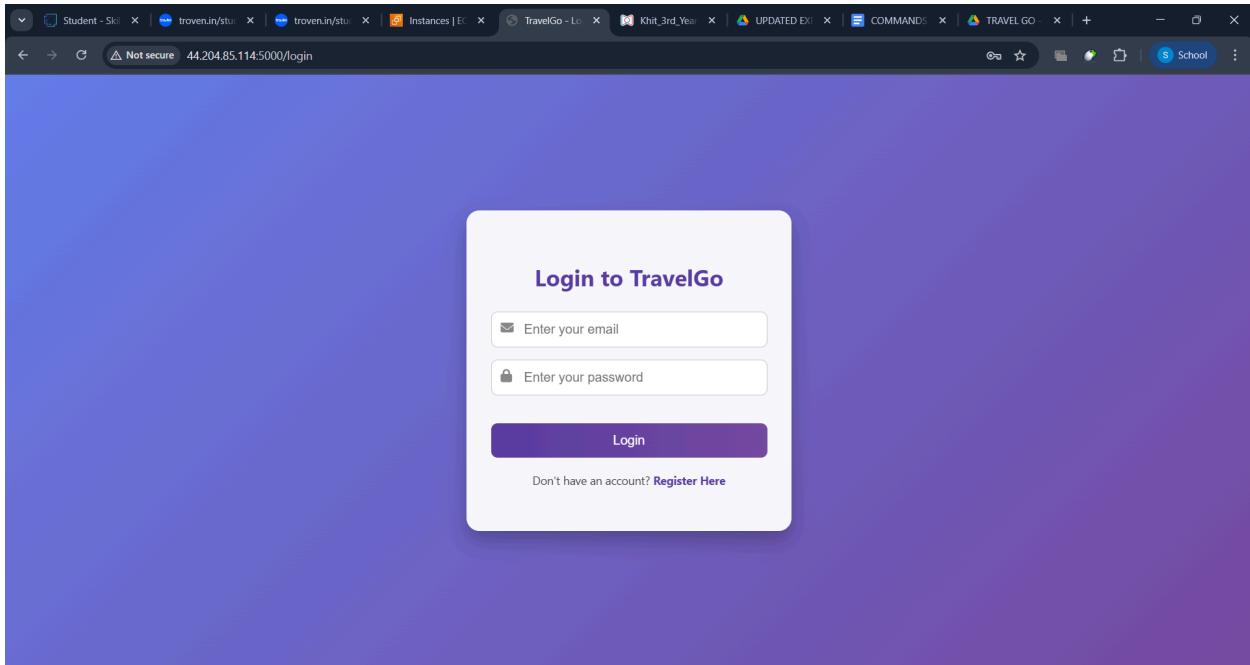
Home Page:



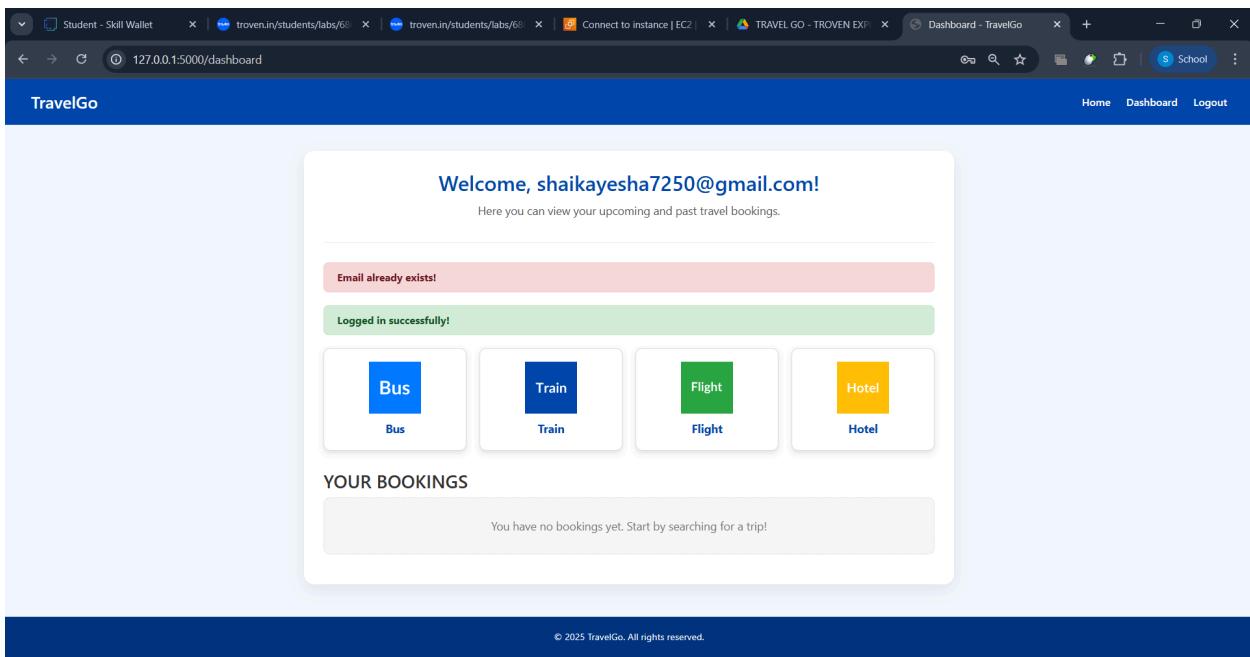
Registration Page:



Login page:



Dashboard Page:



Booking pages:

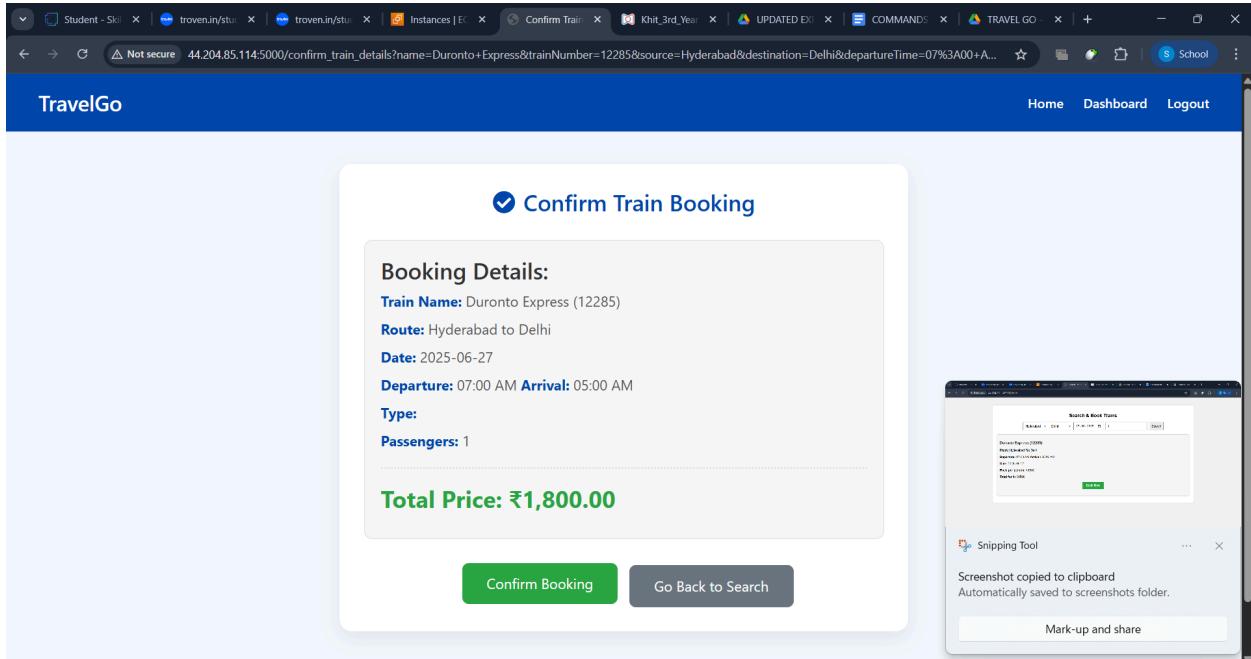
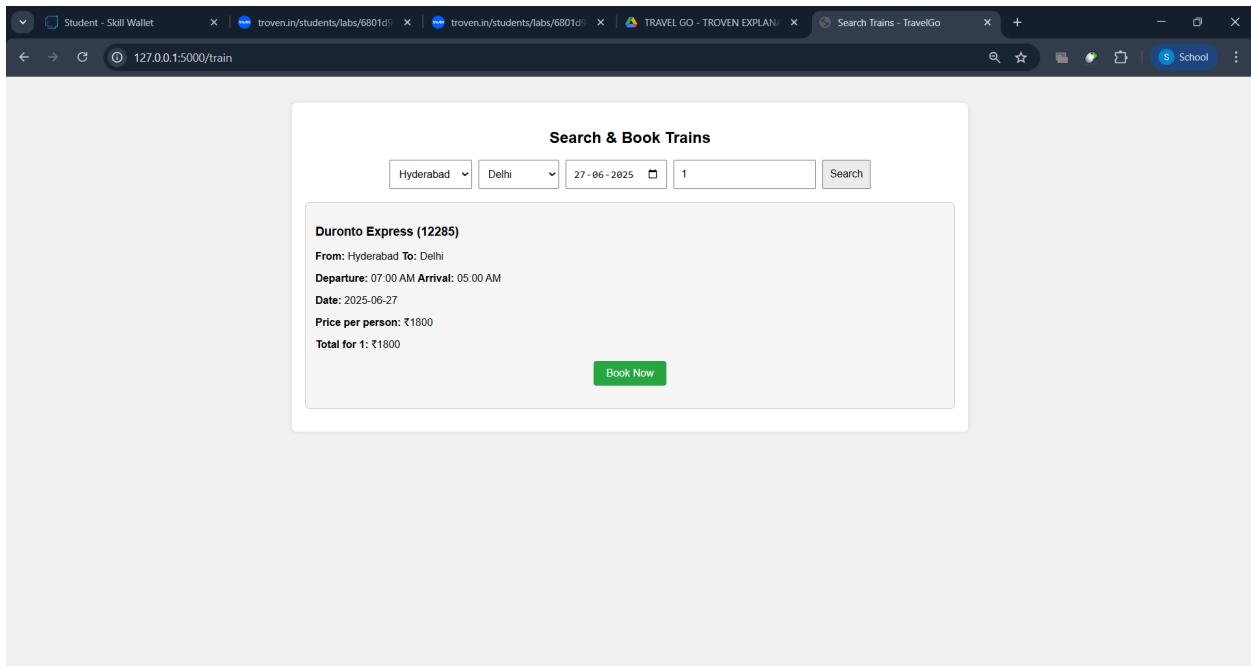
Bus:

The screenshot shows the 'Search & Book Buses' interface. At the top, there are dropdown menus for 'From' (Hyderabad) and 'To' (Vijayawada), a date input (30-06-2025), a passenger count (1), and a 'Search' button. Below these are filter checkboxes for 'AC' (checked), 'Non-AC', 'Sleeper', 'Semi-Sleeper', and 'Seater'. A 'Sort by Price' dropdown is set to 'None'. The results section displays three bus options: 'Orange Travels' (AC Sleeper at 08:00 AM for ₹800/person), 'Garuda Express' (Non-AC Seater at 11:00 AM for ₹400/person), and 'Greenline Travels' (AC Sleeper at 09:00 PM for ₹850/person). Each entry has a 'Book' button.

The screenshot shows the 'Select Your Seats' page. The title is 'Select Your Seats' and a sub-instruction says 'You can select 1 seat(s)'. Below is a grid of 40 numbered seats (S1 to S40) arranged in 5 rows and 8 columns. At the bottom is a green 'Confirm Booking' button.

S1	S2	S3	S4
S5	S6	S7	S8
S9	S10	S11	S12
S13	S14	S15	S16
S17	S18	S19	S20
S21	S22	S23	S24
S25	S26	S27	S28
S29	S30	S31	S32
S33	S34	S35	S36
S37	S38	S39	S40

Train:



Flight:

The screenshot shows a flight search results page. At the top, there are dropdown menus for 'From' (Hyderabad), 'To' (Mumbai), a date selector (20-10-2025), a passenger count (1), and a search button. Below this, a flight result is displayed for 'IndiGo 6E-234'. The details are: Route: Hyderabad - Mumbai, Date: 2025-10-20 | 08:00 - 09:30, Price: ₹3500 × 1 = ₹3500, and a green 'Book' button.

The screenshot shows a confirmation page titled 'Confirm Your Flight Booking'. It lists the following flight details: Airline: IndiGo (6E-234), Route: Hyderabad → Mumbai, Date: 2025-02-10, Departure: 08:00, Arrival: 09:30, Passengers: 1, Price/person: ₹3500, and Total Price: ₹3500. A green 'Confirm Booking' button is at the bottom.

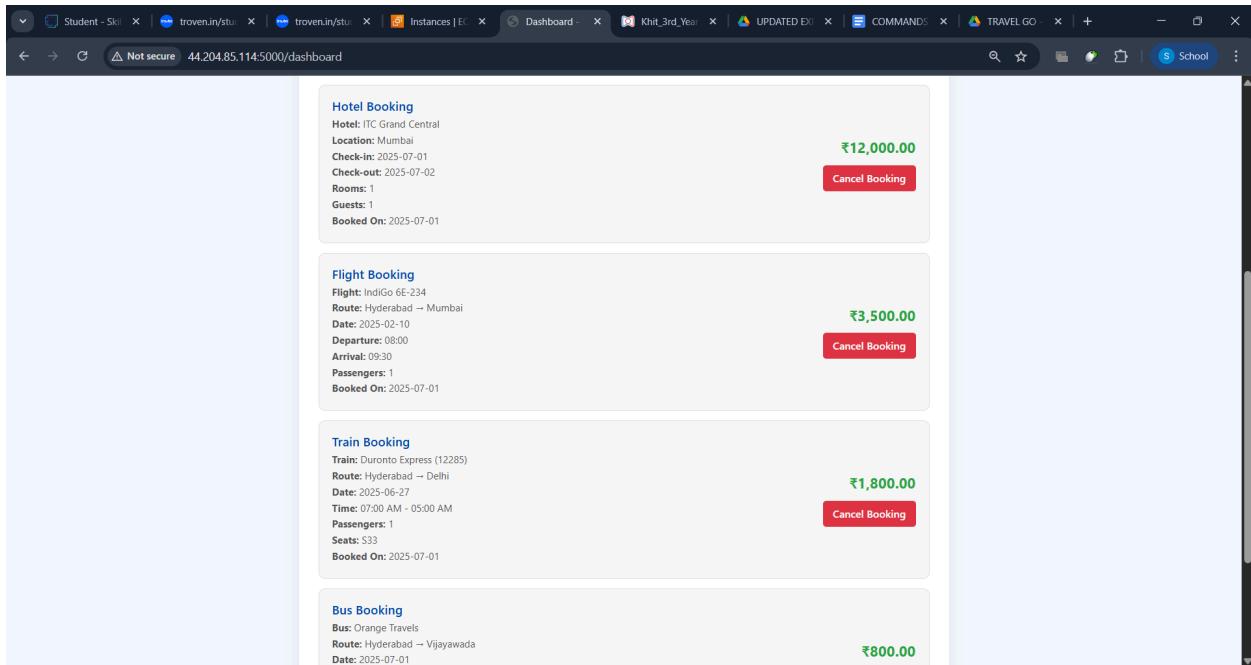
Hotel:

The screenshot shows a web browser window titled "TravelGo" with a search form for "Book Your Perfect Stay". The form includes fields for destination ("Mumbai"), check-in date ("30-06-2025"), check-out date ("01-07-2025"), number of rooms ("1"), and number of guests ("1"). Below the fields are several filter checkboxes: "5-Star" (checked), "4-Star", "3-Star", "WiFi", "Pool", and "Parking". A "Search" button is located to the right of the guest input field. Below the search bar is a dropdown menu labeled "Sort by: None". A result card for "ITC Grand Central" is displayed, showing it's a 5-star hotel in Mumbai at ₹12000/night, with amenities WiFi, Pool, Parking, and Spa. A green "Book" button is on the right side of the card.

The screenshot shows a web browser window titled "Confirm Hotel" with a confirmation dialog titled "Confirm Your Hotel Booking". The dialog lists the following details:
- 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

A large green "Confirm Booking" button is centered at the bottom of the dialog.

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.