

TravelGo – A Cloud-Based Travel Booking System

Project Description:

TravelGo is an integrated, full-stack travel booking web application that enables users to effortlessly plan and reserve transportation and accommodation across buses, trains, flights, and hotels through a single platform. Developed using Flask for backend operations and hosted on Amazon EC2, TravelGo ensures reliability, scalability, and responsiveness even under heavy traffic. Data management is handled through AWS DynamoDB, allowing for secure and efficient storage of user profiles and booking records.

The application supports secure user registration and login, interactive search for travel options, and seamless booking workflows. It also incorporates AWS Simple Notification Service (SNS) to send real-time email notifications for booking confirmations and cancellations, ensuring users remain updated throughout their travel planning.

The platform features a clean, responsive user interface that adapts to different devices. Notable features include dynamic seat selection for bus bookings, hotel filtering by preferences (such as budget or luxury), and a centralized dashboard to track and manage all bookings.

Scenario 1: Unified Booking Across All Travel Modes

TravelGo simplifies multi-modal travel planning by allowing users to book buses, trains, flights, and hotels all from one place. For example, a traveler booking a trip from Hyderabad to Bangalore can log in, explore all available transport modes, and confirm their preferred options. The Flask backend handles user queries and real-time data fetching, while AWS EC2 hosting ensures high availability and responsiveness during peak usage like weekends or holiday seasons.

Scenario 2: Instant Booking Confirmation via AWS SNS

Upon completing any booking—whether it's a hotel reservation or a flight ticket—TravelGo triggers an email notification using AWS SNS. For instance, if a user books a hotel in Chennai, they immediately receive an email confirmation with all relevant booking details. This is managed via Flask backend logic, which interacts with DynamoDB to store the booking and then calls SNS to notify the user. Admins or service providers can also receive alerts, facilitating real-time awareness and efficient customer support.

Scenario 3: Personalized Dashboard for Booking Management

TravelGo includes a personalized dashboard where users can view, manage, and cancel bookings. Bookings are categorized (bus, train, flight, hotel) and displayed with dates, prices, and status updates. Flask dynamically pulls this data from DynamoDB and renders it using responsive HTML/CSS templates. Users can easily track their travel history, upcoming plans, and process cancellations with a single click—ensuring a seamless and user-friendly experience on both desktop and mobile devices.

Project WorkFlow:

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

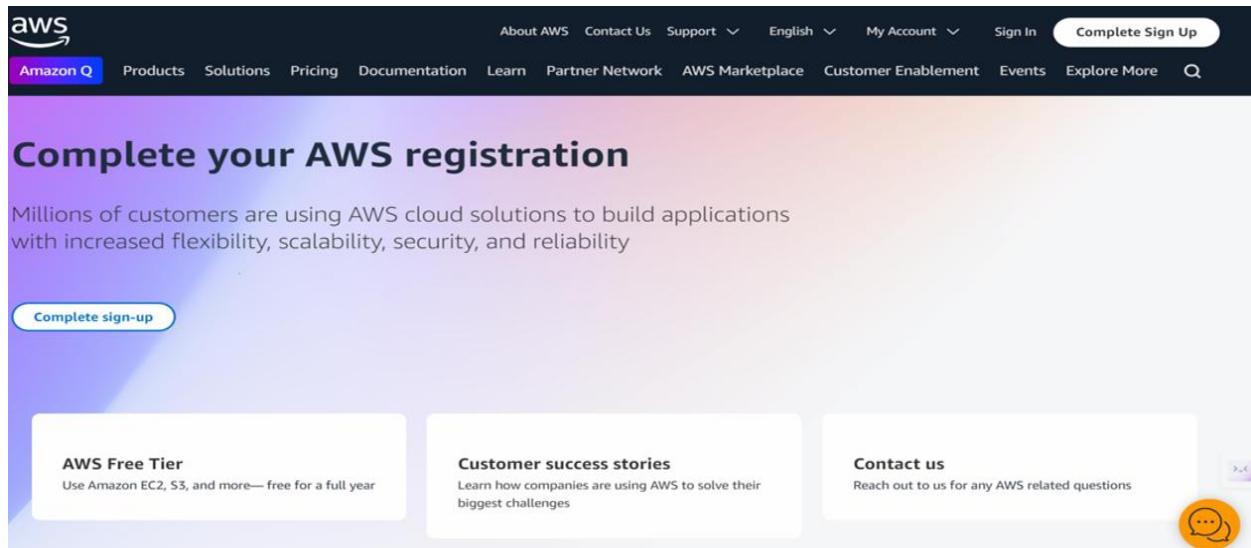
8. Testing and Deployment

Activity 8.1: Conduct functional testing to verify user registration, login, book requests, and notifications.

Milestone 1: AWS Account Setup and Login

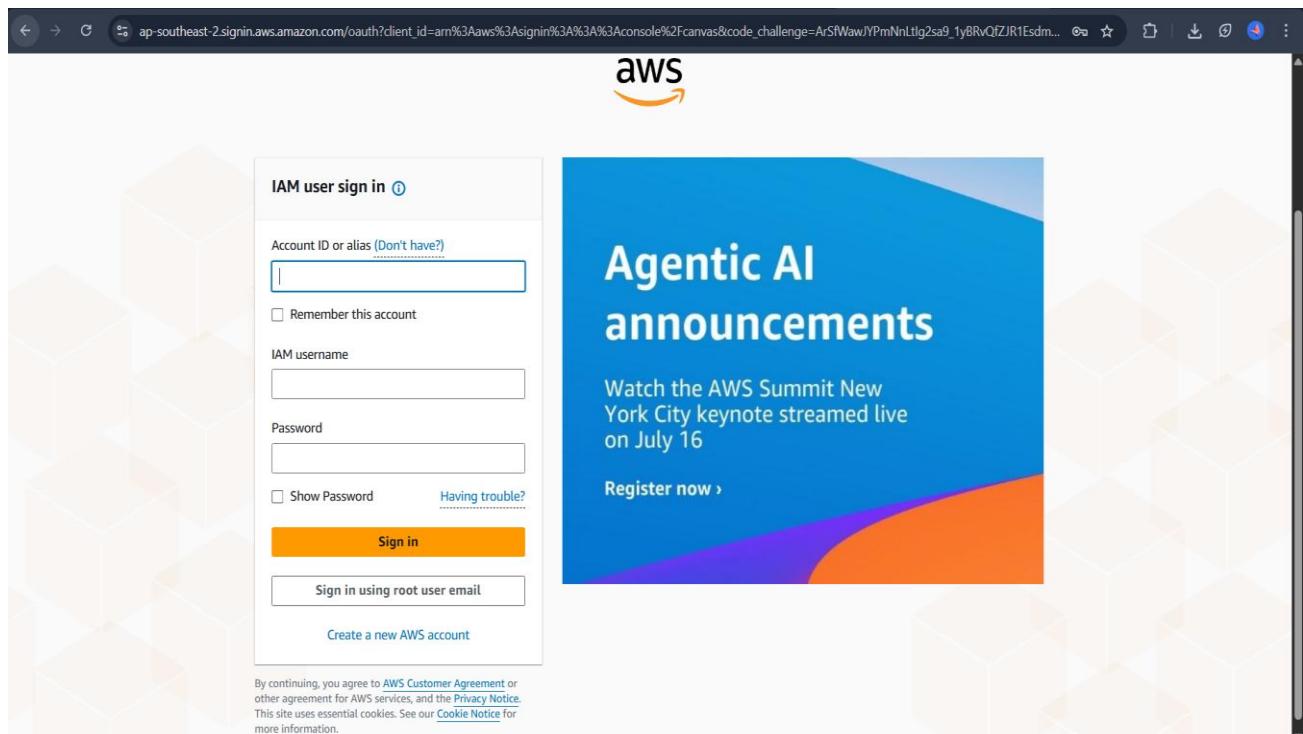
- Activity 1.1: Set up an AWS account if not already done.

- Sign up for an AWS account and configure billing settings.



- Activity 1.2: Log in to the AWS Management Console

- After setting up your account, log in to the [AWS Management Console](#)



Milestone 2: DynamoDB Database Creation and Setup

• Activity 2.1: Navigate to the DynamoDB

- In the AWS Console, navigate to DynamoDB and click on create tables.

The screenshot shows the AWS Console Home page. On the left, there's a sidebar with 'Recently visited' services: IAM, EC2, DynamoDB, Simple Notification Service, Billing and Cost Management, AWS Marketplace, Route 53, and S3. Below this is a 'Welcome to AWS' section with 'Getting started with AWS'. To the right, there's an 'Applications' section with a table header for Name, Description, Region, and Originator. A message says 'No applications' and 'Get started by creating an application.' with a 'Create application' button. At the top right, there are 'Reset to default layout' and '+ Add widgets' buttons. The bottom of the page has sections for 'AWS Health' and 'Cost and usage'.

The screenshot shows the Amazon DynamoDB service dashboard. The left sidebar has navigation links: Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations (New), Reserved capacity, Settings, DAX (Clusters, Subnet groups, Parameter groups, Events), and CloudShell. The main content area has a 'Database' title and a large banner for 'Amazon DynamoDB: A fast and flexible NoSQL database service for any scale'. It includes a 'Get started' callout with 'Create table' and a 'Pricing' callout with a link to learn more about pricing. At the bottom, there are links for Documentation, Feedback, and a footer with copyright information.

• Activity 2.2: Create a DynamoDB table for storing registration details and book requests.

- Create Users table with partition key "Email" with type String and click on create tables.

DynamoDB > Tables > Create table

Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.

Partition key
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.

<input type="text" value="user_email"/>	String
---	--------

1 to 255 characters and case sensitive.

Sort key - optional
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.

<input type="text" value="booking_date"/>	String
---	--------

1 to 255 characters and case sensitive.

Table settings

<input checked="" type="radio"/> Default settings The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these settings, choose Customize settings .	<input type="radio"/> Customize settings Use these advanced features to make DynamoDB work better for your needs.
---	--

Tags

Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

[Add new tag](#)

You can add 50 more tags.

[Cancel](#) [Create table](#)

- Follow the same steps to create a travelgo_users table with Email as the primary key.

DynamoDB > Tables > Create table

Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.

Partition key
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.

<input type="text" value="email"/>	String
------------------------------------	--------

1 to 255 characters and case sensitive.

Sort key - optional
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.

<input type="text" value="Enter the sort key name"/>	String
--	--------

1 to 255 characters and case sensitive.

Table settings

<input checked="" type="radio"/> Default settings The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these settings, choose Customize settings .	<input type="radio"/> Customize settings Use these advanced features to make DynamoDB work better for your needs.
---	--

Tags

Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

Add new tag

You can add 50 more tags.

Cancel **Create table**

- o Follow the same steps to create a trains table with train_number as the partition key.

DynamoDB > Tables > Create table

Create table

Table details Info
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.)

Partition key
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.

Sort key - optional
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.

Table settings

Default settings
The fastest way to create your table. You can modify most of these settings after your table has been created. To modify

Customize settings
Use these advanced features to make DynamoDB work better for your needs.

Tags

Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

Add new tag

You can add 50 more tags.

Cancel **Create table**

Milestone 3: SNS Notification Setup

- **Activity 3.1: Create SNS topics for sending email notifications to users and library staff.**

- o In the AWS Console, search for SNS and navigate to the SNS Dashboard.

The screenshot shows the AWS search interface for 'DynamoDB'. The left sidebar lists various AWS services and features under 'DynamoDB' and 'DAX'. The main search results are displayed in two sections: 'Services' and 'Features'. Under 'Services', there are cards for 'Simple Notification Service', 'Route 53 Resolver', and 'Route 53'. Under 'Features', there are cards for 'Events' (with 'ElastiCache feature') and 'SMS' (with 'AWS End User Messaging feature'). At the bottom, there's a section asking 'Were these results helpful?' with 'Yes' and 'No' buttons.

The screenshot shows the Amazon SNS service page. The left sidebar includes links for Dashboard, Topics, Subscriptions, Mobile, Push notifications, and Text messaging (SMS). The main content area features a 'New Feature' banner about FIFO topics. Below it is a large heading 'Amazon Simple Notification Service' with the subtext 'Pub/sub messaging for microservices and serverless applications.' A detailed description of Amazon SNS follows. To the right, there's a 'Create topic' form with a 'Topic name' input field containing 'MyTopic', a 'Next step' button, and a link to 'Start with an overview'. At the bottom right is a 'Pricing' link.

- Click on **Create Topic** and choose a name for the topic.

The screenshot shows the 'Topics' page in the Amazon SNS service. The left sidebar is identical to the previous screenshot. The main area displays a table with three entries, each representing a topic. The columns are 'Name', 'Type', and 'ARN'. The first topic is named 'MyTopic'. There are buttons for 'Edit', 'Delete', and 'Publish message' at the top of the table, along with navigation arrows and a search bar.

- Choose Standard type for general notification use cases and Click on Create Topic.

Amazon SNS > Topics > Create topic

Create topic

Details

Type | Info Topic type cannot be modified after topic is created

FIFO (first-in, first-out)

- Strictly-preserved message ordering
- Exactly-once message delivery
- Subscription protocols: SQS

Standard

- Best-effort message ordering
- At-least once message delivery
- Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints

Name: MyTopic Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).

Display name - optional | Info To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.

My Topic Maximum 100 characters.

- Configure the SNS topic and note down the Topic ARN.

Amazon SNS

Dashboard Topics Subscriptions

Mobile Push notifications Text messaging (SMS)

Topic Travlego2 created successfully. You can create subscriptions and send messages to them from this topic.

Travlego2

Details

Name: Travlego2	Display name:
ARN: arn:aws:sns:ap-south-1:353250843450:Travlego2	Topic owner: 353250843450
Type: Standard	

Subscriptions (0)

Access policy | Data protection policy | Delivery policy (HTTP/S) | Delivery status logging | Encryption | Tags | Int'l |

Create subscription

- Activity 3.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.

Amazon SNS > Subscriptions > Create subscription

Create subscription

Details

Topic ARN: Q. arn:aws:sns:ap-south-1:353250843450:Travlego2

Protocol: The type of endpoint to subscribe

Email

Endpoint: An email address that can receive notifications from Amazon SNS.

test@example.com

After your subscription is created, you must confirm it. Info

Subscription filter policy - optional Info This policy filters the messages that a subscriber receives.

Redrive policy (dead-letter queue) - optional Info Send undeliverable messages to a dead-letter queue.

- o After subscription request for the mail confirmation.
- o 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> 11:15 (0 minutes ago) ☆

to me ▾

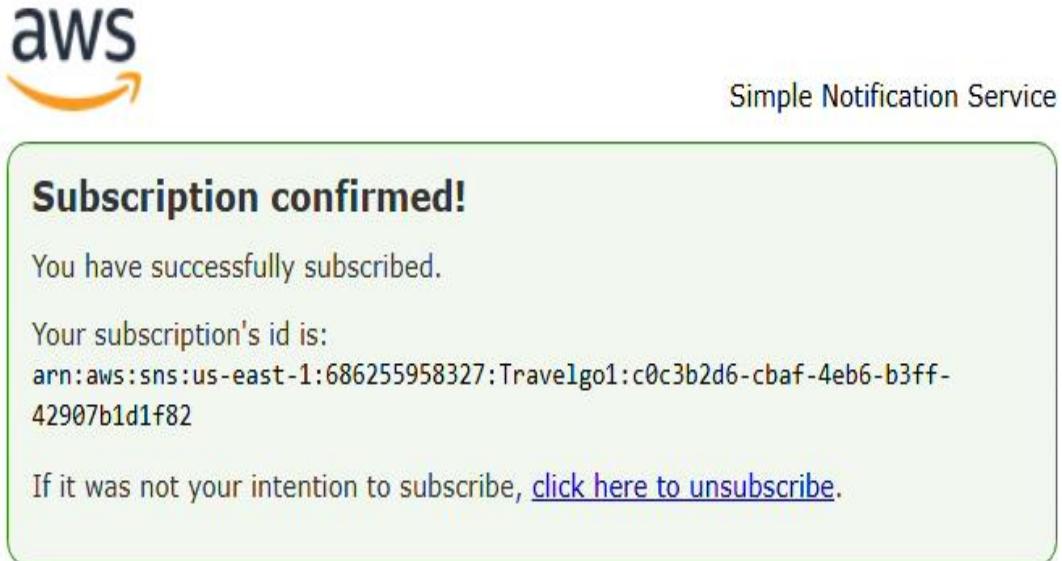
Why is this message in spam? This message is similar to messages that were identified as spam in the past.

[Report as not spam](#)

You have chosen to subscribe to the topic:
`arn:aws:sns:us-east-1:686255958327:Travelgo1`

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](#)

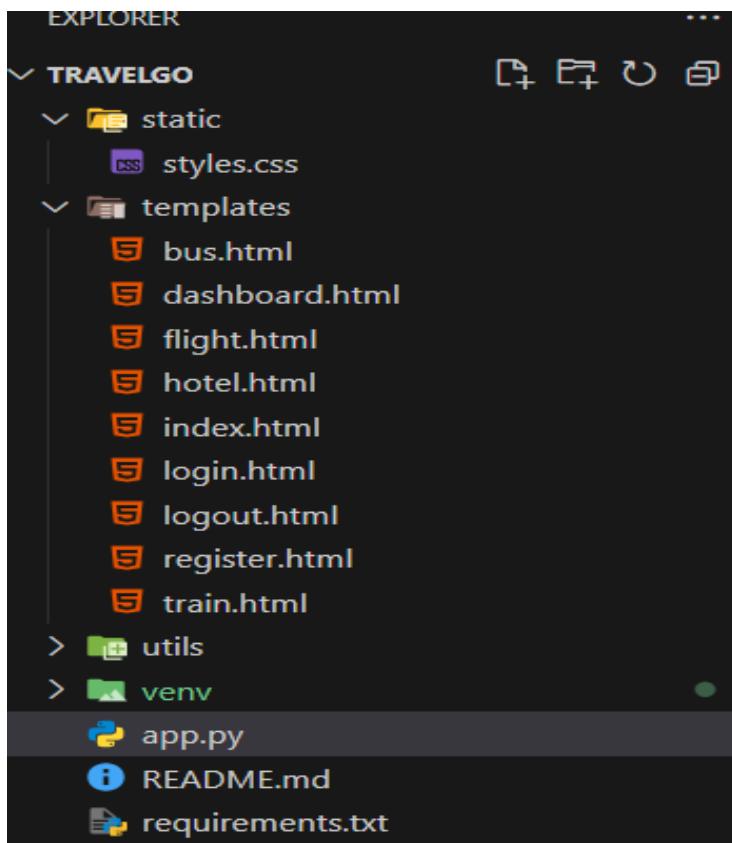


- o Successfully done with the SNS mail subscription and setup, now store the ARN link.

Milestone 4: Backend Development and Application Setup

- **Activity 4.1: Develop the backend using Flask**

- o File Explorer Structure



Description of the code :

- **Flask App Initialization**

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

Description:

Import essential libraries including Flask utilities for routing, template rendering, session handling, and request processing; Boto3 and DynamoDB conditions for performing AWS DynamoDB operations; Werkzeug security module for password hashing and verification; Datetime for handling timestamps; Decimal for precise number representation (required by DynamoDB); and UUID for generating unique booking and user identifiers.

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

Description: initialize the Flask application instance using `Flask(__name__)` to start building the web app.

- **Dynamodb Setup:**

```
# DynamoDB Tables
users_table = dynamodb.Table('travelgo_users')
bookings_table = dynamodb.Table('bookings')
```

Description:

Define references to DynamoDB tables used in the application.

- `users_table` connects to the `travelgo_users` table, which stores user registration data such as names, emails, and hashed passwords.
- `bookings_table` connects to the `bookings` table, which stores all travel booking records including buses, trains, flights, and hotels, along with user details, booking IDs, and timestamps.

- **SNS Connection**

```
SNS_TOPIC_ARN = 'arn:aws:sns:us-east-1:713881794827:travelgo:302dcf42-b7bc

# Function to send SNS notifications
def send_sns_notification(subject, message):
    try:
        sns_client.publish(
            TopicArn=SNS_TOPIC_ARN,
            Subject=subject,
            Message=message
        )
        print(f"[✓] SNS notification sent: {subject}")
    except Exception as e:
        print(f"SNS Error: Could not send notification - {e}")
```

Description:

Define the **SNS Topic ARN** used to publish notifications through **AWS Simple Notification Service (SNS)**.

The `send_sns_notification()` function is responsible for sending real-time **email alerts or notifications** (e.g., booking confirmations or cancellations).

It uses the `sns_client.publish()` method to send a message with a given subject and body to the specified SNS topic.

Error handling is included to catch and print any exceptions if the notification fails.

- **Routes for Web Pages**

- **Home Route:**

```
# Routes
@app.route('/')
def home():
    return render_template('index.html', logged_in='email' in session)
```

Description:

Define the home route (/) of the Flask application.

The home() function renders the index.html template. It also checks if the user is logged in by verifying if 'email' exists in the session. If so, it passes logged_in=True to the template, allowing the frontend to dynamically adjust the UI (e.g., show or hide login/register buttons).

- **Register Route:**

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

        # Check if user already exists
        existing = users_table.get_item(Key={'email': email})
        if 'Item' in existing:
            flash('User already exists!', 'error')
            return render_template('register.html')

        # Hash password and store user
        hashed_password = generate_password_hash(password)
        user_data = {
            'email': email,
            'name': name,
            'password': hashed_password,
            'logins': 0
        }

        users_table.put_item(Item=user_data)
        print(f"[✓] Registered new user: {email}")

        flash('Registration successful! Please log in.', 'success')
        return redirect(url_for('login'))
    return render_template('register.html')
```

Description:

Handles the /register route for user signup using GET and POST methods.
Checks if the user already exists in the travelgo_users table and prevents duplicates.
If new, hashes the password, stores the user in DynamoDB, and redirects to the login page.

• login Route:

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

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

        if 'Item' in user_response:
            user = user_response['Item']
            if check_password_hash(user['password'], password):
                session['email'] = email
                session['user_name'] = user.get('name', email)

                # Update login count
                users_table.update_item(
                    Key={'email': email},
                    UpdateExpression='ADD logins :val',
                    ExpressionAttributeValues={':val': 1}
                )

                flash('Login successful! Welcome back.', 'success')
                return redirect(url_for('dashboard'))

            flash('Invalid email or password. Please try again.', 'error')
            return render_template('login.html')

    return render_template('login.html')
```

Description:

Handles the /login route for user authentication via GET and POST methods.
On POST, it retrieves the user from travelgo_users by email and verifies the password using check_password_hash().
If valid, it stores user details in the session and updates the login count in DynamoDB.
On failure, it flashes an error message and re-renders the login page.

- **Dashboard Route:**

```
@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
    try:
        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
        for booking in bookings:
            for key, value in booking.items():
                if isinstance(value, Decimal):
                    booking[key] = float(value)

    except Exception as e:
        print(f"Error fetching bookings: {e}")
        bookings = []

    return render_template('dashboard.html', username=user_email, bookings=bookings)
```

Description:

Defines the /dashboard route to display the logged-in user's booking history.

Checks session for a valid login, then queries the bookings table using the user's email to fetch bookings in reverse chronological order.

Converts any Decimal values from DynamoDB to float for proper display in the HTML template.

Renders dashboard.html, passing the user's email and their bookings for display.

- **Booking Route:**

```
# Booking Routes
@app.route('/bus')
def bus_booking():
    if 'email' not in session:
        flash('Please login to book tickets.', 'error')
        return redirect(url_for('login'))
    return render_template('bus.html')

@app.route('/train')
def train_booking():
    if 'email' not in session:
        flash('Please login to book tickets.', 'error')
        return redirect(url_for('login'))
    return render_template('train.html')

@app.route('/flight')
def flight_booking():
    if 'email' not in session:
        flash('Please login to book tickets.', 'error')
        return redirect(url_for('login'))
    return render_template('flight.html')

@app.route('/hotel')
def hotel_booking():
    if 'email' not in session:
        flash('Please login to book tickets.', 'error')
        return redirect(url_for('login'))
    return render_template('hotel.html')
```

Description:

Defines individual booking routes for bus, train, flight, and hotel pages. Each route first checks if the user is logged in by verifying the session; if not, it redirects to the login page with an error message. If the user is authenticated, the corresponding booking template (bus.html, train.html, etc.) is rendered.

- **Deployment Code:**

```
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Description:

Starts the Flask application when the script is run directly. Runs the app in debug mode for development, allowing live code reloading and error display. Binds the server to all IP addresses (0.0.0.0) on port 5000, making it accessible on the local network or EC2 instance.

Milestone 5: IAM Role Setup

• Activity 5.1: Create IAM Role.

- In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.

The image consists of three vertically stacked screenshots from the AWS IAM console.

Screenshot 1: IAM Service Selection
Shows the search results for 'iam'. The 'IAM' service is selected, described as 'Manage access to AWS resources'.

Screenshot 2: IAM Dashboard
Shows the IAM dashboard with a blue banner about new access analyzers. It includes sections for Security recommendations (root user has MFA, no active access keys) and IAM resources (resources in this AWS Account). On the right, it shows account details: Account ID (353250843450), Account Alias (Create), and a sign-in URL (https://353250843450.signin.aws.amazon.com/console).

Screenshot 3: Create Role Wizard - Step 1
Shows the 'Create role' wizard. In the 'Commonly used services' dropdown, 'EC2' is selected. A tooltip indicates that EC2 can perform actions in the current account. Below the dropdown, there's a link to 'Choose a service or use case'.

• Activity 5.2: Attach Policies.

Attach the following policies to the role:

- **AmazonDynamoDBFullAccess:** Allows EC2 to perform read/write operations on DynamoDB.
- **AmazonSNSFullAccess:** Grants EC2 the ability to send notifications via SNS.
- **AmazonEC2FullAccess:** Provides full permissions to manage all EC2 resources, including instances, volumes, and networking.

The screenshots show the 'Add permissions' step of creating a new IAM role. The user is selecting three specific policies from a list of available AWS managed policies.

Screenshot 1: Selecting AmazonDynamoDBFullAccess

Policy name	Type	Description
AmazonDynamoDBFullAccess	AWS managed	Provides full access to Amazon DynamoDB
AmazonDynamoDBFullAccess_v2	AWS managed	Provides full access to Amazon DynamoDB
AmazonDynamoDBFullAccesswithDataPipeline	AWS managed	This policy is on a Data Pipeline
AmazonDynamoDBReadOnlyAccess	AWS managed	Provides read-only access to Amazon DynamoDB
AWSLambdaDynamoDBExecutionRole	AWS managed	Provides list and describe access to Amazon DynamoDB

Screenshot 2: Selecting AmazonSNSFullAccess

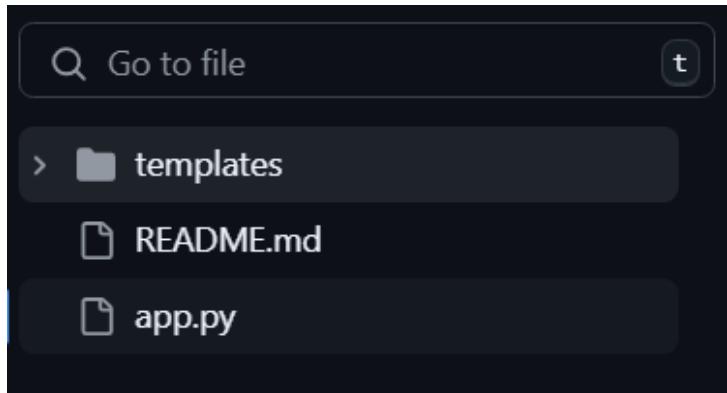
Policy name	Type	Description
AmazonSNSFullAccess	AWS managed	Provides full access to Amazon Simple Notification Service (SNS)
AmazonSNSReadOnlyAccess	AWS managed	Provides read-only access to Amazon Simple Notification Service (SNS)
AmazonSNSRole	AWS managed	Default policy for SNS roles
AWSElasticBeanstalkRoleSNS	AWS managed	(Elastic Beanstalk)
AWSIoTDeviceDefenderPublishFindingsToSNSMitigationLambdaFunctionRole	AWS managed	Provides message publishing access to Amazon SNS

Screenshot 3: Selecting AmazonEC2FullAccess

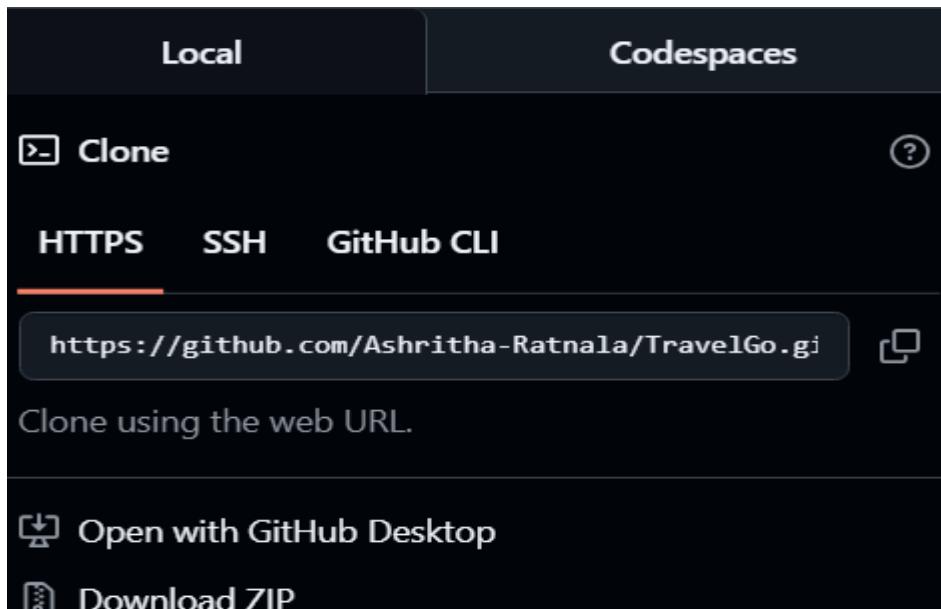
Policy name	Type	Description
AmazonEC2ContainerRegistryFullAccess	AWS managed	Provides admin access to Amazon Elastic Container Registry (ECR)
AmazonEC2ContainerRegistryPowerUser	AWS managed	Provides full access to Amazon ECR
AmazonEC2ContainerRegistryPullOnly	AWS managed	Provides access to pull images from Amazon ECR
AmazonEC2ContainerRegistryReadOnly	AWS managed	Provides read-only access to Amazon ECR
AmazonEC2ContainerServiceAutoscaleRole	AWS managed	Policy to enable Amazon EC2 Container Service (ECS) Auto Scaling
AmazonEC2ContainerServiceEventsRole	AWS managed	Policy to enable Amazon EC2 Container Service (ECS) Events
AmazonEC2ContainerServiceforEC2Role	AWS managed	Default policy for Amazon EC2 Container Service (ECS) for Amazon EC2
AmazonEC2ContainerServiceRole	AWS managed	Default policy for Amazon EC2 Container Service (ECS)
AmazonEC2FullAccess	AWS managed	Provides full access to Amazon EC2
AmazonEC2ReadOnlyAccess	AWS managed	Provides read-only access to Amazon EC2

Milestone 6: EC2 Instance Setup

- Note: Load your Flask app and Html files into GitHub repository.



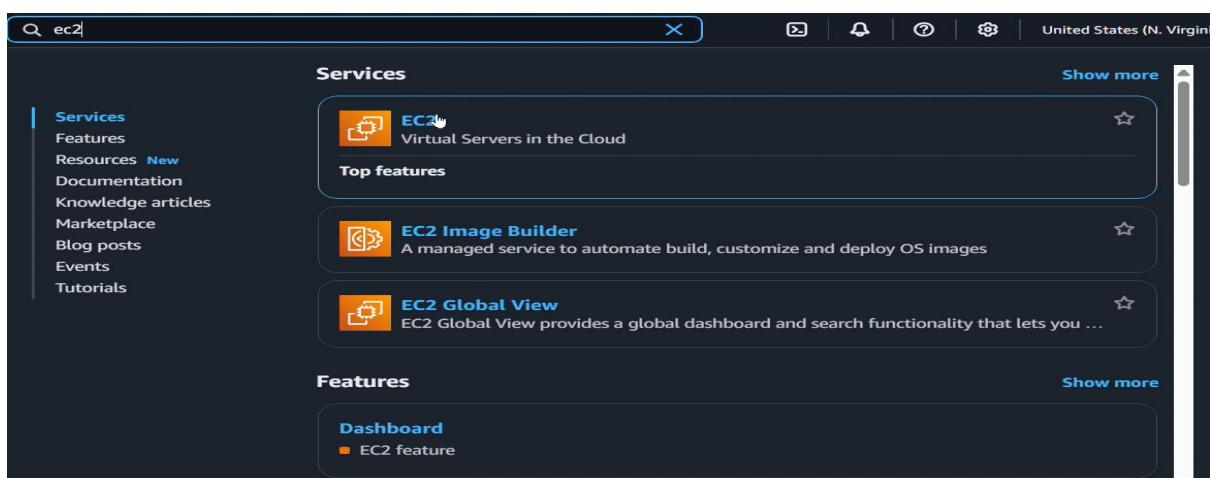
The screenshot shows a GitHub repository interface. At the top, there is a search bar labeled "Go to file" and a "t" icon. Below the search bar, the repository structure is shown: a "templates" folder containing "README.md" and "app.py".



The screenshot shows the "Clone" page for the "TravelGo" repository on GitHub. It features tabs for "Local" and "Codespaces". Under the "Local" tab, there are three cloning options: "HTTPS" (selected), "SSH", and "GitHub CLI". Below these options is a URL input field containing "https://github.com/Ashritha-Ratnala/TravelGo.git" with a copy icon to its right. A note below the URL says "Clone using the web URL.". Further down, there are links for "Open with GitHub Desktop" and "Download ZIP".

- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

- Launch EC2 Instance
 - In the AWS Console, navigate to EC2 and launch a new instance.



The screenshot shows the AWS EC2 service page. The top navigation bar includes a search bar with "ec2", a user icon, a bell icon, a help icon, and a gear icon, followed by "United States (N. Virginia)". The main content area is titled "Services" and features a "Top features" section with three items: "EC2" (Virtual Servers in the Cloud), "EC2 Image Builder" (A managed service to automate build, customize and deploy OS images), and "EC2 Global View" (EC2 Global View provides a global dashboard and search functionality that lets you ...). Below this is a "Features" section with a "Dashboard" item (labeled as an "EC2 feature"). On the left side, there is a sidebar with a "Services" heading and links to "Features", "Resources", "Documentation", "Knowledge articles", "Marketplace", "Blog posts", "Events", and "Tutorials".

- Click on Launch instance to launch EC2 instance.

The screenshot shows the AWS EC2 homepage. On the left, there's a sidebar with navigation links like Dashboard, EC2 Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, and Images. The main content area features a large heading "Amazon Elastic Compute Cloud (EC2)" and sub-headings "Create, manage, and monitor virtual servers in the cloud." Below this is a paragraph about the service's breadth and depth, mentioning over 600 instance types. To the right, there's a call-to-action box with a "Launch instance" button and a "View dashboard" link. At the bottom, there's a "Benefits and features" section and a "Get started" box with a "Take our walkthroughs to help you" link.

The screenshot shows the "Launch an instance" wizard. Step 1: Set instance details. It has a blue header bar with a message: "It seems like you may be new to launching instances in EC2. Take a walkthrough to learn about EC2, how to launch instances and about best practices". Below this are two buttons: "Do not show me this message again" and "Take a walkthrough". The main form has fields for "Name and tags" (with "Travelgoapplication1" entered) and "Application and OS Images (Amazon Machine Image)". On the right, there's a summary panel showing "Number of instances: 1", "Software Image (AMI): Amazon Linux 2023 AMI 2023.7.2...read more", and "Virtual server type (instance type): t2.micro".

- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).

The screenshot shows the "Amazon Machine Image (AMI)" catalog. At the top, there's a search bar with placeholder text "Search our full catalog including 1000s of application and OS images". Below it is a "Quick Start" section with icons for various AMIs: Amazon Linux, macOS, Ubuntu, Windows, Red Hat, SUSE Linux, and Debian. To the right, there's a "Browse" button and a "Including AWS, Mark the Con" link. The main content area shows a card for the "Amazon Linux 2023 kernel-6.1 AMI", which includes details like "ami-05ffe3c48a9991133 (64-bit (x86), uefi-preferred) / ami-022bbd2ccaf21691f (64-bit (Arm), uefi)", "Virtualization: hvm", "ENA enabled: true", "Root device type: ebs", and "Free tier".

- Create and download the key pair for Server access.

Create key pair

Key pair name
Key pairs allow you to connect to your instance securely.

travel-go-application-1

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

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](#)

- **Activity 6.2:Configure security groups for HTTP, and SSH access.**

Edit inbound rules:

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-04f74a082f34acd4e	Custom TCP	TCP	5000	Custom	0.0.0.0/0
sgr-0e91ec1a5eaa31f68	HTTPS	TCP	443	Custom	0.0.0.0/0
sgr-0a2fdcb6ce079d31	SSH	TCP	22	Custom	0.0.0.0/0

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.

- 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.

EC2 Instances

Instances (1/1) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
TravelGoproject	i-0b6de2ae9071ed0a7	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b	ec2-18-206...

i-0b6de2ae9071ed0a7 (TravelGoproject)

Security details

Details

Security group name	Security group ID	Description	VPC ID
launch-wizard-1	sg-074f829b17da4071a	launch-wizard-1 created 2025-07-01T05:45:46.696Z	vpc-0ed9be657d2390aa2
Owner	Inbound rules count	Outbound rules count	
686255958327	3 Permission entries	1 Permission entry	

Inbound rules

Name	Security group rule ID	IP version	Type	Protocol	Port range
-	sgr-04f74a082f34acd4e	IPv4	Custom TCP	TCP	5000
-	sgr-0e91ec1a5ea31f68	IPv4	HTTPS	TCP	443
-	sgr-0a2fdcb6ce079d31	IPv4	SSH	TCP	22

Modify IAM role

Select an IAM role to attach to your instance.

Instance ID: i-04e6fc7081801414 (travelgoapplication)

IAM role: studentuser

Create new IAM role

Update IAM role

● Now connect the EC2 with the files

EC2 Instances > i-0b6de2ae9071ed0a7 > Connect to instance

Connect

Connect to an instance using the browser-based client.

EC2 Instance Connect

⚠️ Unable to verify public subnet

You are not authorized to perform this operation. User: arn:aws:sts::686255958327:assumed-role/rsoaccount-new/6801da4369d20120be221457 is not authorized to perform: ec2:DescribeRouteTables because no identity-based policy allows the ec2:DescribeRouteTables action

Unable to verify if associated subnet [subnet-0661d905cf55c8217](#) is a public subnet.
To use EC2 Instance Connect, your instance must be in a public subnet. [To make the subnet a public subnet, add a route in the subnet route table to an internet gateway.](#)

Instance ID: i-0b6de2ae9071ed0a7 (TravelGoproject)

Connect using a Public IP: Connect using a public IPv4 or IPv6 address

Connect using a Private IP: Connect using a private IP address and a VPC endpoint

Public IPv4 address: 18.206.251.133

IPv6 address: -

Milestone 7: Deployment on EC2

Activity 7.1: Install Software on the EC2 Instance

Install Python3, Flask, and Git:

On Amazon Linux 2:

```
Sudo yum install git -y  
sudo yum install python3 -y  
sudo yum install python3-pip -y
```

Pip install flask

Pip install boto3

Activity 7.2: Clone Your Flask Project from GitHub

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

Run: '[git clone https://github.com/your-github-username/your-repository-name.git](https://github.com/your-github-username/your-repository-name.git)'

Note: change your-github-username and your-repository-name with your credentials

here: 'git clone https://github.com/AlekhyaPenubakula/InstantLibrary.git'

- This will download your project to the EC2 instance.

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

```
cd InstantLibrary
```

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

Run the Flask Application

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

Verify the Flask app is running:

<http://your-ec2-public-ip>

- Run the Flask app on the EC2 instance

```
*** 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://192.168.137.149:5000  
Press CTRL+C to quit  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 141-023-319
```

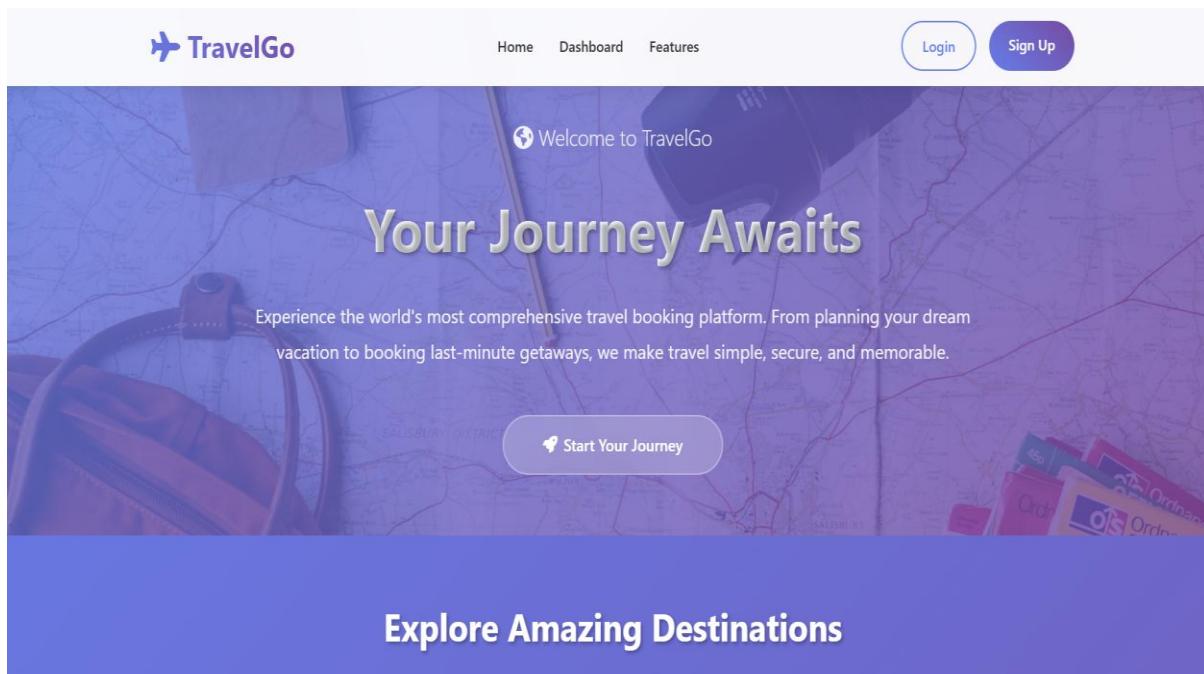
Access the website through:

PublicIPs: <http://54.205.69.13:5000/>

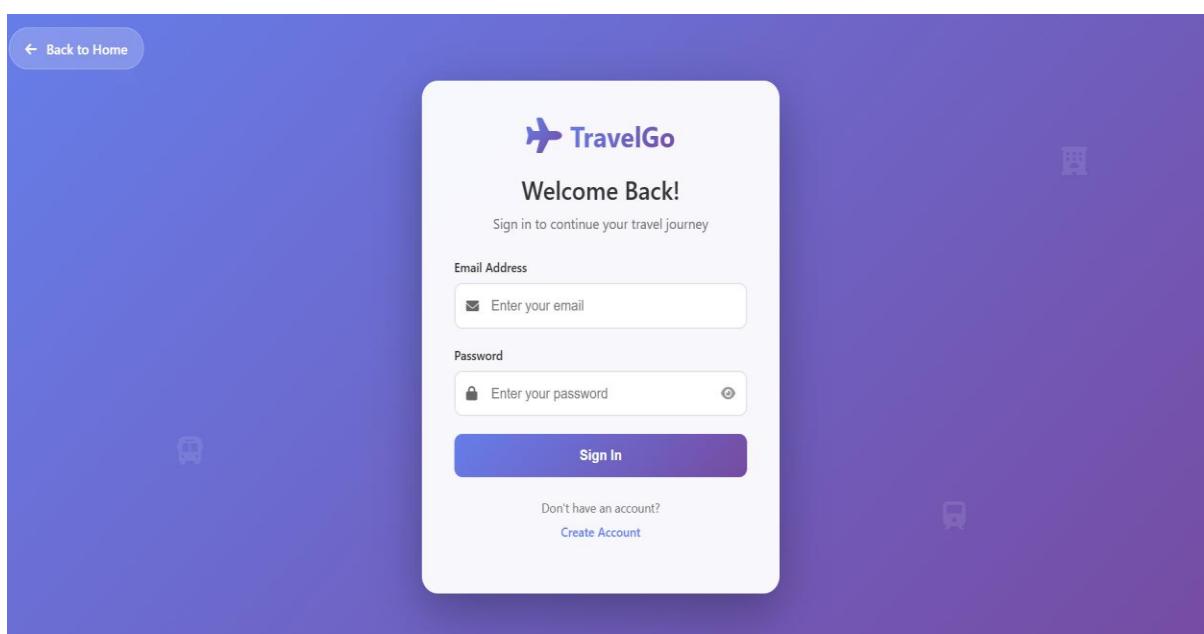
Milestone 8: Testing and Deployment

- **Activity 8.1: Conduct functional testing to verify user registration, login, book requests, and notifications.**

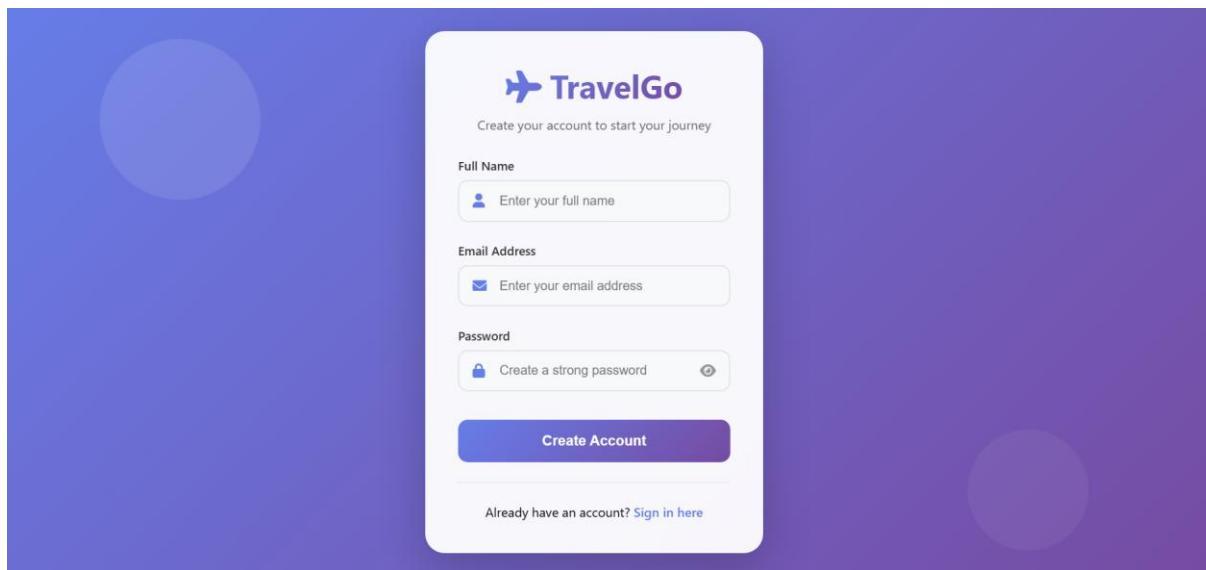
Index page:



Login page:



Register page:

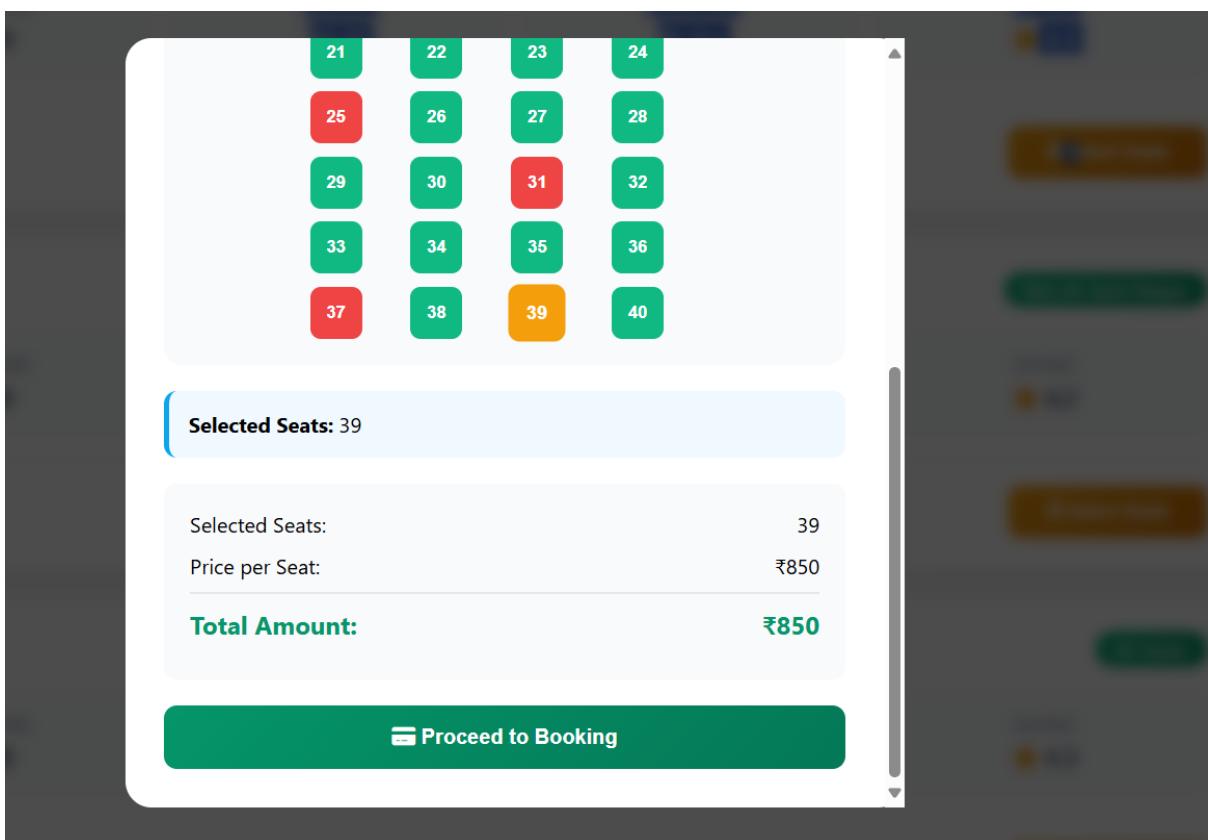
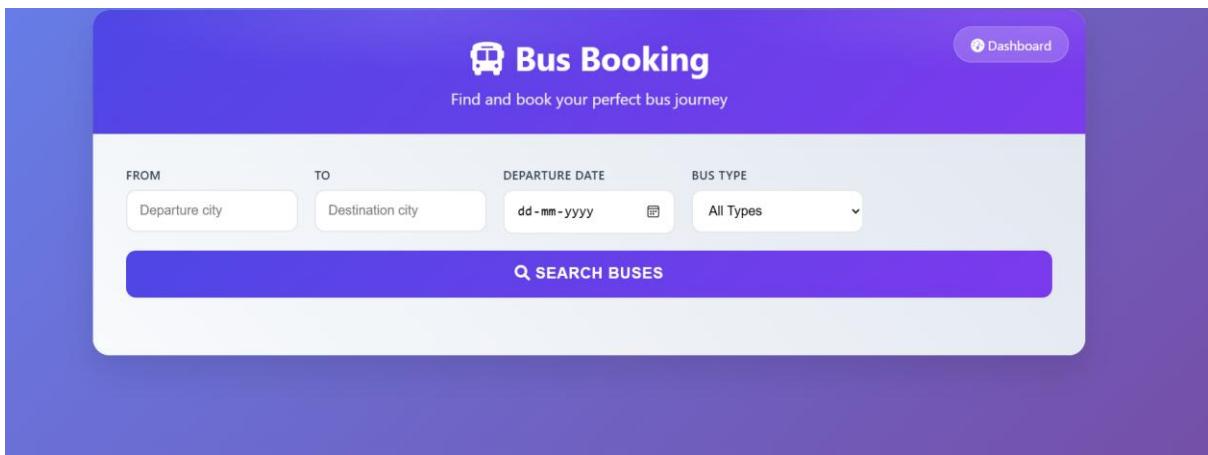


Dashboard page:

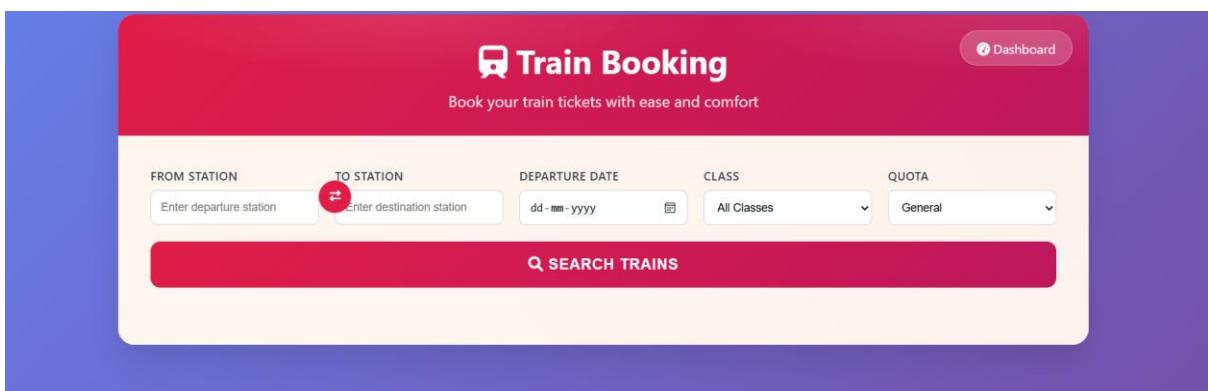
A screenshot of the TravelGo dashboard. At the top, it says 'Welcome back, Traveler!' and 'Ready for your next adventure? Let's plan something amazing together.' Below this are four stats: '0 Active Bookings', '0 Cities Visited', '0 Hours Traveled', and '5.0 Average Rating'. The dashboard has a purple header with the TravelGo logo and a red 'Logout' button.

A screenshot of the TravelGo dashboard showing service cards. There are four cards: 'Bus Tickets' (with an image of a bus, placeholder: Comfortable intercity bus travel at affordable prices, and a 'Book Now' button), 'Train Tickets' (with an image of a train, placeholder: Fast and reliable train journeys across the country, and a 'Book Now' button), 'Flight Tickets' (with an image of an airplane wing, placeholder: Domestic and international flights at best prices, and a 'Book Now' button), and 'Hotel Booking' (with an image of a resort, placeholder: Luxury and budget hotels for every traveler, and a 'Book Now' button). The dashboard has a purple header with the TravelGo logo and a red 'Logout' button.

Bus booking page:



Train booking page:



Found 3 trains

All Trains Superfast Express Mail Passenger

Rajdhani Express

12301 ON TIME

New Delhi 16:55 Howrah 10:10
NDLS HWH

17h 15m 1441 km

1st AC (1A) ₹3500 Available 2nd AC (2A) ₹2200 Available 3rd AC (3A) ₹1500 WL 8

Meal Blanket Wi-Fi Charging Point Book Now

Flight booking page:

Book Your Flight

Find the best flight deals and book your journey with ease

Round Trip One Way

From To Departure Date Return Date

Departure city Destination city dd-mm-yyyy dd-mm-yyyy

Passengers Class

1 Passenger Economy

Search Flights

Available Flights

Sort by: Price (Low to High)

Flight Details	Price	Action
22:30 Mumbai -> 01:15 Dubai 3h 45m Emirates EK512	₹18,500	Book Now

✓ Wi-Fi ✓ Meal ✓ Entertainment ✓ Lounge Access

Hotel booking page:

Find Your Perfect Stay

Discover amazing hotels and book your dream accommodation

Destination

Check-in Date

Check-out Date

Guests

Rooms

Search Hotels

Available Hotels

Sort by: Price (Low to High)



4★

Heritage Boutique Stay

Jaipur, Rajasthan

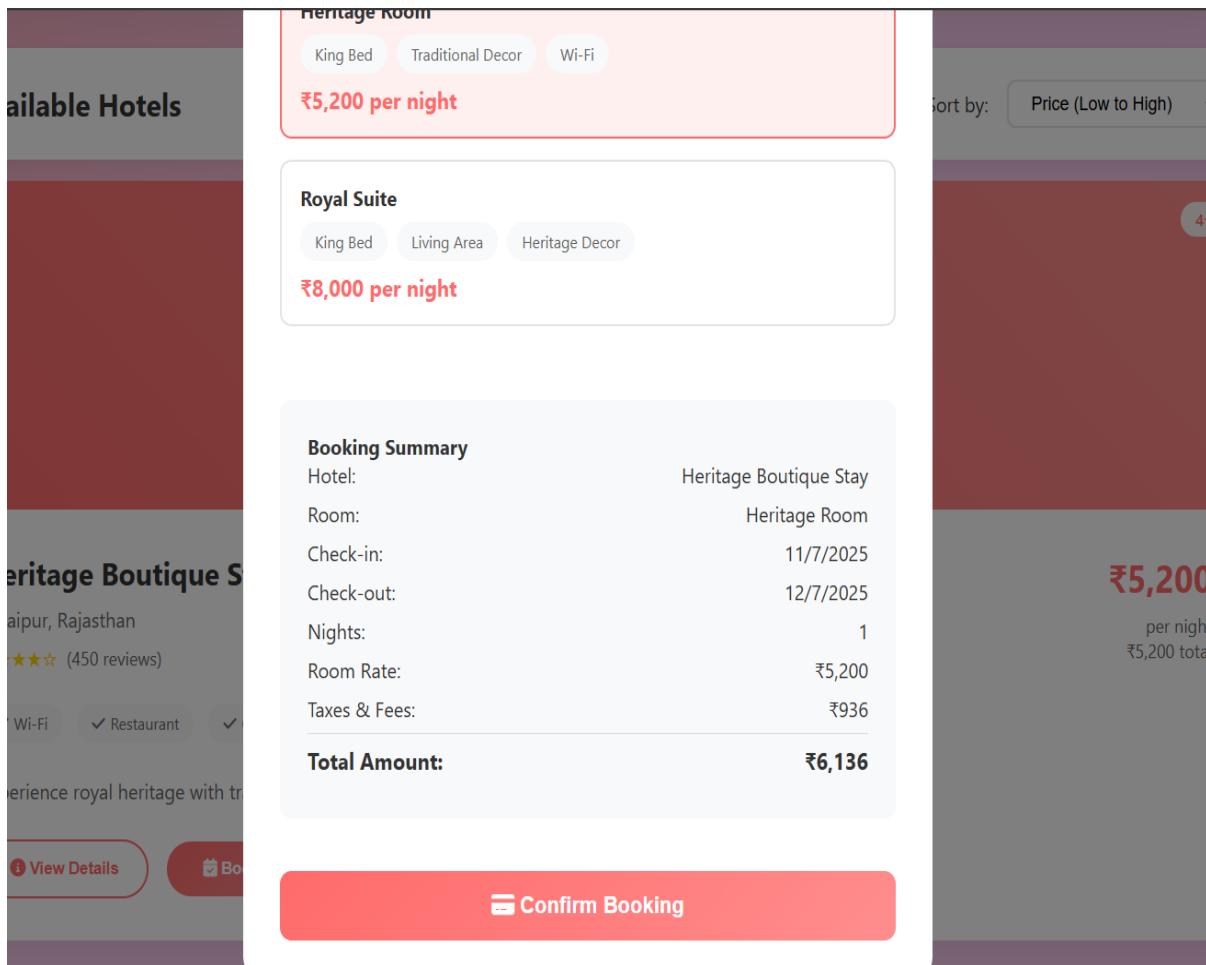
★★★★☆ (450 reviews)

✓ Wi-Fi ✓ Restaurant ✓ Cultural Tours ✓ Parking

₹5,200 per night ₹5,200 total

Experience royal heritage with traditional architecture, cultural programs, and authentic cuisine.

[View Details](#) [Book Now](#)



Conclusion:

TravelGo successfully integrates modern cloud technologies to deliver a seamless, user-friendly travel booking experience. By combining Flask, AWS EC2, DynamoDB, and SNS, the platform ensures real-time responsiveness, secure data handling, and instant user notifications. Its unified interface for booking buses, trains, flights, and hotels—along with dynamic dashboards and filtering options—makes it a scalable and reliable solution for modern travel needs. TravelGo not only enhances user convenience but also demonstrates the practical application of full-stack development and cloud services in solving real-world problems.