TravelGo: A Cloud-Powered Real-Time Travel Booking Platform Using AWS

Project Description:

TravelGo is a full-stack, cloud-based travel booking platform designed to simplify the process of reserving buses, trains, flights, and hotels through a unified interface. Built using Flask as the backend framework, the application is deployed on Amazon EC2 and leverages DynamoDB for efficient storage of user data and bookings. TravelGo allows users to register, log in, search for transportation and accommodation options, and book their travel with ease. Once a booking is confirmed or cancelled, users receive real-time email notifications powered by AWS Simple Notification Service (SNS), keeping them informed throughout their journey.

The platform's user-friendly interface supports dynamic seat selection for buses, hotel filtering based on preferences such as luxury or budget, and provides booking summaries along with centralized cancellation management. By combining cloud scalability, responsive design, and secure session handling, TravelGo delivers a seamless and real-time travel planning experience for users.

Scenarios:

Scenario 1: Hassle-Free Multi-Mode Travel Booking Experience

TravelGo offers users a unified platform to search and book buses, trains, flights, and hotels all in one place. For instance, a user planning a trip from Hyderabad to Bangalore can log in, select their preferred mode of transport, choose from available options, and proceed to booking. Flask manages the backend operations such as retrieving travel listings and processing user input in real-time. Hosted on AWS EC2, the platform remains responsive even during high-traffic hours like weekends or holiday seasons, allowing multiple users to browse and book without delay.

Scenario 2: Real-Time Booking Confirmation with AWS SNS

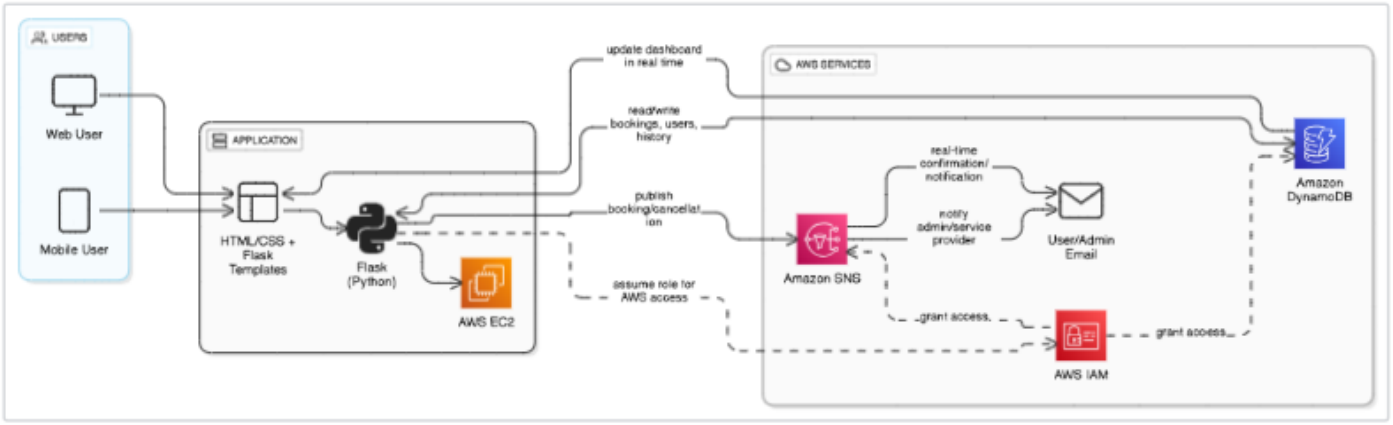
Once a booking is made—whether it's a train ticket or a hotel stay—TravelGo uses AWS SNS to instantly notify the user. For example, after a student books a hotel in Chennai, SNS sends a real-time email notification confirming the booking with all the relevant details. This notification is triggered from the Flask backend after the booking is successfully recorded in DynamoDB. Additionally, SNS can alert admin or service providers, ensuring transparency and real-time updates on every transaction.

Scenario 3: Dynamic Dashboard with Personal Travel History

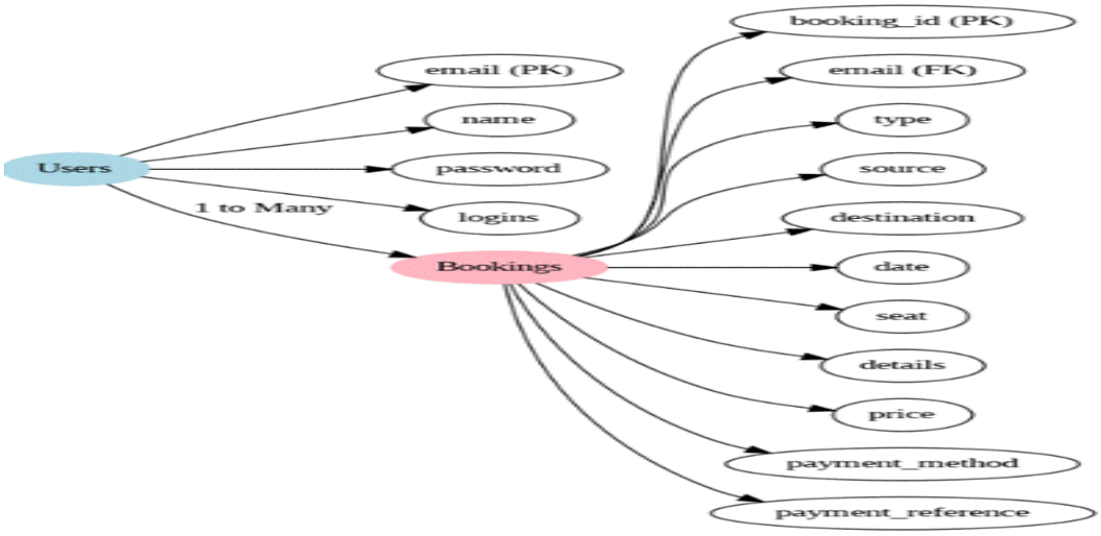
TravelGo features a dynamic user dashboard that displays all past and upcoming bookings for the logged-in user. For example, a user who has booked a flight and a hotel can view these bookings categorized by type, along with dates, price, and cancellation options. Flask fetches this data from AWS DynamoDB, which persistently stores all user bookings. The dashboard UI, powered by responsive HTML/CSS and Flask templates, ensures users can review or manage bookings anytime, from any device, with real-time updates and quick cancellation workflows supported.

AWS ARCHITECTURE

This AWS-based architecture powers a scalable and secure web application using Amazon EC2 for hosting the backend, with a lightweight framework like Flask handling core logic. Application data is stored in Amazon DynamoDB, ensuring fast, reliable access, while user access is managed through AWS IAM for secure authentication and control. Real-time alerts and system notifications are enabled via Amazon SNS, enhancing communication and user engagement



Entity Relationship (ER)Diagram:



Pre-requisites:

- AWS Account Setup : https://docs.aws.amazon.com/accounts/latest/reference/getting-started.html
- AWS IAM (Identity and Access Management) : https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html
- AWS EC2 (Elastic Compute Cloud) : https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html
- AWS DynamoDB : https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html
- Amazon SNS : https://docs.aws.amazon.com/sns/latest/dg/welcome.html
- Git Documentation : https://git-scm.com/doc
- VS Code Installation : (download the VS Code using the below link or you can get that in Microsoft store) https://code.visualstudio.com/download

Project WorkFlow:

Milestone 1. Backend Development and Application Setup
- Develop the Backend Using Flask.
- Integrate AWS Services Using boto3.

Milestone 2. AWS Account Setup and Login
- Set up an AWS account if not already done.
- Log in to the AWS Management Console

Milestone 3. DynamoDB Database Creation and Setup
- Create a DynamoDB Table.
- Configure Attributes for User Data and Book Requests**.**

Milestone 4. SNS Notification Setup
- Create SNS topics for book request notifications.
- Subscribe users and library staff to SNS email notifications**.**

Milestone 5. IAM Role Setup
- Create IAM Role
- Attach Policies

Milestone 6. EC2 Instance Setup

- Launch an EC2 instance to host the Flask application.
- Configure security groups for HTTP, and SSH access.

Milestone 7. Deployment on EC2

- Upload Flask Files
- Run the Flask App

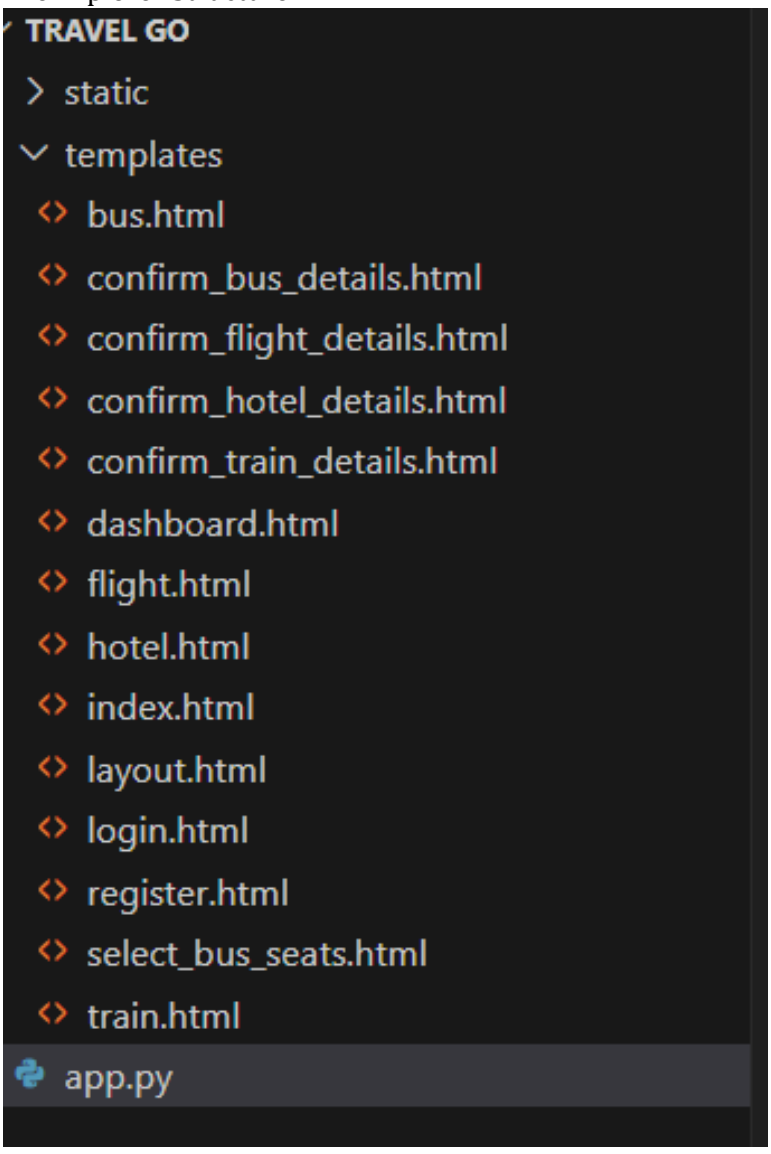
Milestone 8. Testing and Deployment

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

Milestone 1: Backend Development and Application SetUp

- **Activity 1.1:** Develop the Backend Using Flask

1. File Explorer Structure



Description: Organize the project with HTML templates for each feature (e.g., login, wishlist, quiz, checkout) under the templates folder and manage backend logic in app.py .

Description of the code :

? Flask App Initialization

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

- Import essential Flask modules for web handling, Boto3 for AWS integration, Werkzeug for password hashing, and datetime for timestamp management.

```python
app = Flask(__name__)
app.secret_key = 'your_secret_key_here' #
```

- Initialize the Flask application and set a secret key to securely manage user sessions and form data.

```python
# AWS Setup using IAM Role
REGION = 'us-east-1'  # Replace with your actual AWS region
dynamodb = boto3.resource('dynamodb', region_name=REGION)
sns_client = boto3.client('sns', region_name=REGION)

users_table = dynamodb.Table('travelgo_users')
trains_table = dynamodb.Table('trains') # Note: This table is declar
bookings_table = dynamodb.Table('bookings')
```

- Connect to DynamoDB using Boto3 and define references to the UserTable and WishlistTable for user and wishlist data operations.

Routes for Core Functionalities:

```python
# 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')
```

Create the home and registration routes, where the registration route securely hashes user passwords and stores user data in DynamoDB upon form submission.

Login route: Implement the user login route to validate credentials using DynamoDB and securely manage session data while updating the user's login count.

```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'], passwo
            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')
```

- **Dashboard Route**: Secure the user dashboard and implement a route to add items to the wishlist, storing them in DynamoDB with item details and a timestamp.

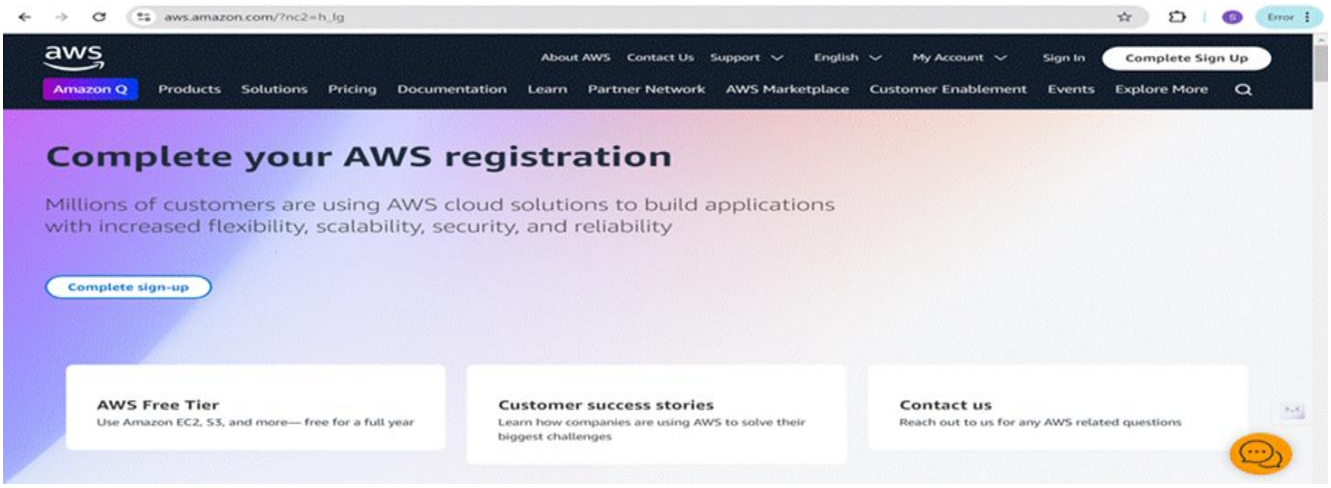
```python
@app.route('/dashboard')
def dashboard():
    if 'email' not in session:
        return redirect(url_for('login'))
    user_email = session['email']

    # 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
def final_confirm_bus_booking():
        flash(f"Failed to confirm bus booking due to database error: {e}", 'error')
        return redirect(url_for("bus"))

    send_sns_notification(
        subject="Bus Booking Confirmed",
        message=f"Dear {booking['user_email']},\nYour bus from {booking['source']} to {booking['destination']} on {booking['travel_da
    )

    flash('Bus booking confirmed successfully!', 'success')
    return redirect(url_for('dashboard'))

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

@app.route('/confirm_flight_details')
def confirm_flight_details():
    if 'email' not in session:
        return redirect(url_for('login')) # Ensure user is logged in

    booking = {
        'flight_id': request.args['flight_id'],
        'airline': request.args['airline'],
        'flight_number': request.args['flight_number'],
        'source': request.args['source'],
        'destination': request.args['destination'],
        'departure_time': request.args['departure'],
        'arrival_time': request.args['arrival'],
        'travel_date': request.args['date'],
        'num_persons': int(request.args['passengers']),
        'price_per_person': Decimal(request.args['price']), # Convert to Decimal for consistency
    }
    booking['total_price'] = booking['price_per_person'] * booking['num_persons']
    session['pending_booking'] = booking # Store for final confirmation
```

```python
def cancel_booking():
    booking_id = request.form.get('booking_id')
    user_email = session['email']
    booking_date = request.form.get('booking_date') # This is crucial as it's the sort key

    if not booking_id or not booking_date:
        flash("Error: Booking ID or Booking Date is missing for cancellation.", 'error')
        return redirect(url_for('dashboard'))

    try:
        # Delete item using the primary key (user_email and booking_date)
        # This does not use GSI, so it remains unchanged.
        bookings_table.delete_item(
            Key={'user_email': user_email, 'booking_date': booking_date}
        )
        flash(f"Booking {booking_id} cancelled successfully!", 'success')
    except Exception as e:
        flash(f"Failed to cancel booking {booking_id}: {str(e)}", 'error')

    return redirect(url_for('dashboard'))


if __name__ == '__main__':
    # IMPORTANT: In a production environment, disable debug mode and specify a production-ready host.
    app.run(debug=True, host='0.0.0.0')
```

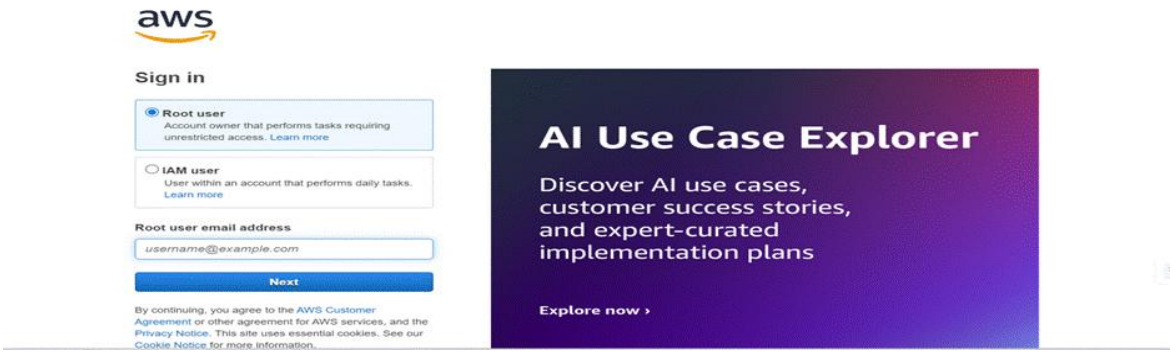
- **Milestone 2: AWS Account Setup and Login.**

    - **Activity 2.1:** Set up an AWS account if not already done.
        - Sign up for an AWS account and configure billing settings.



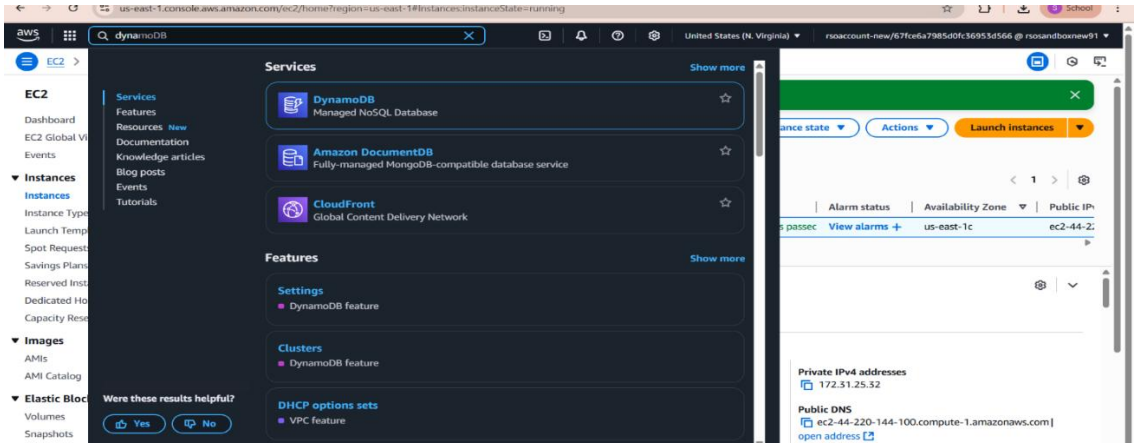
- Click on the "Create an AWS Account" button.
- Follow the prompts to enter your email address and choose a password.
- Provide the required account information, including your name, address, and phone number.
- Enter your payment information. (Note: While AWS offers a free tier, a credit card or debit card is required for verification.)
- Complete the identity verification process.
- Choose a support plan (the basic plan is free and sufficient for starting).
- Once verified, you can sign in to your new AWS accounts.

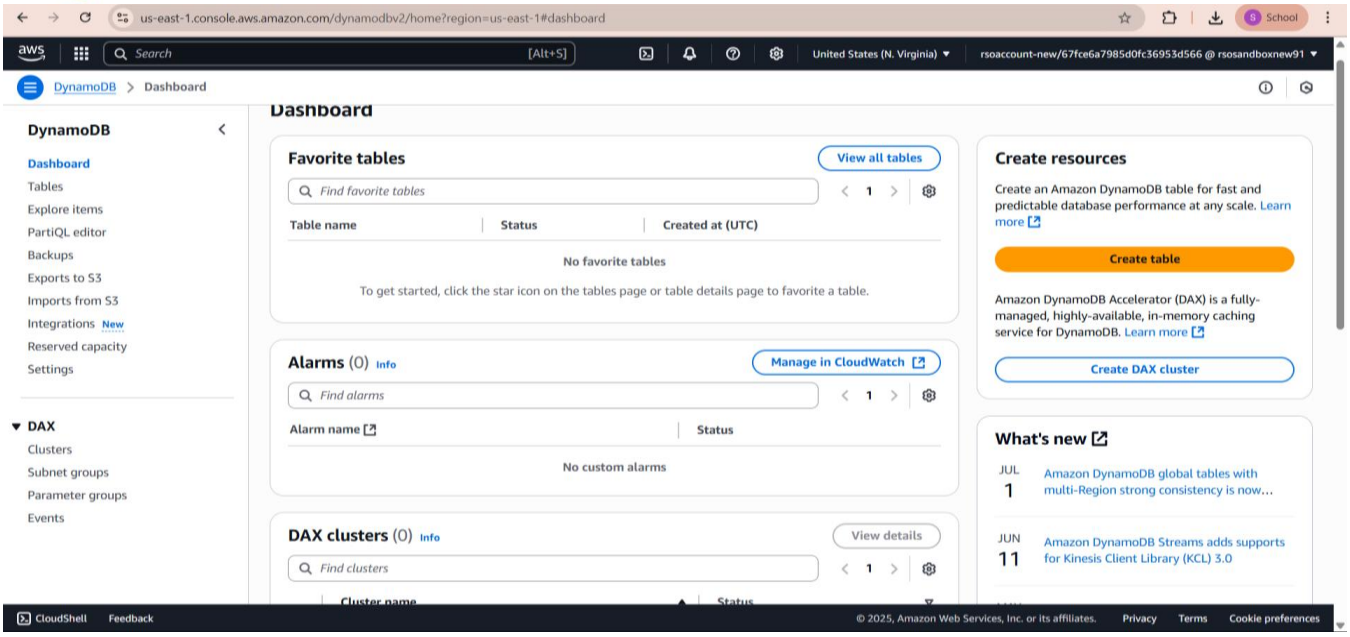
- **Activity2.2 :** Log in to the AWS Management Console
    - After setting up your account, log in to the [AWS Management Console](#).



Milestone 3: DynamoDB Database Creation and Setup

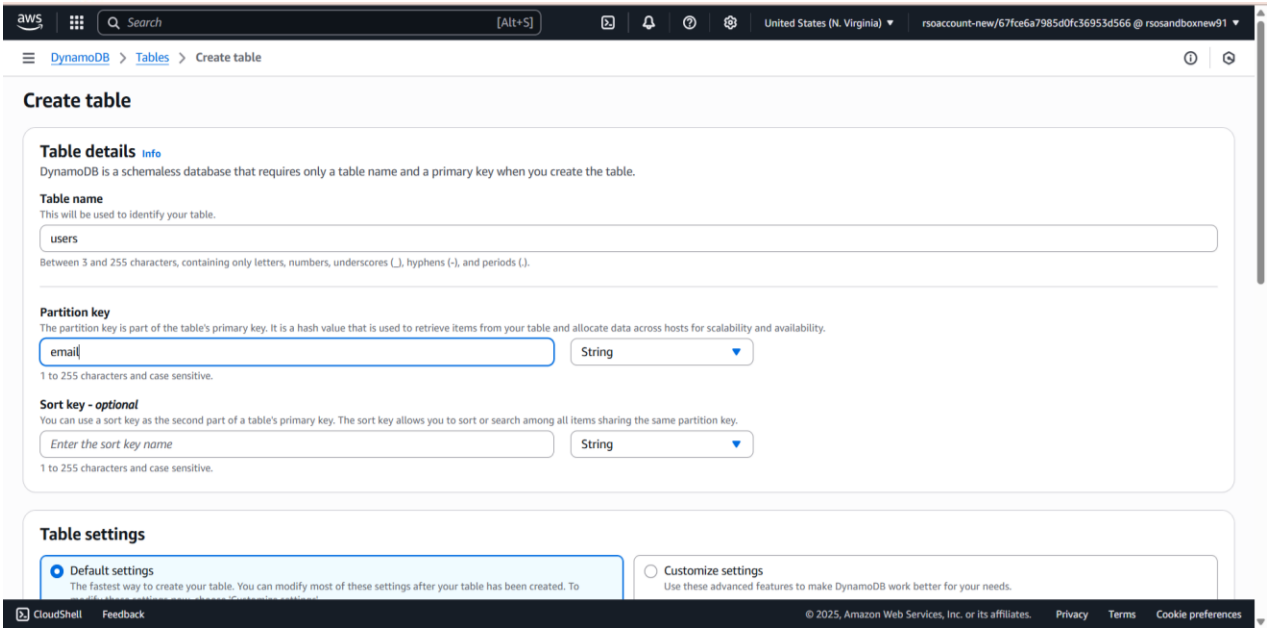
- **Activity 3.1:**Navigate to the DynamoDB
    - In the AWS Console, navigate to DynamoDB and click on create tables**.**

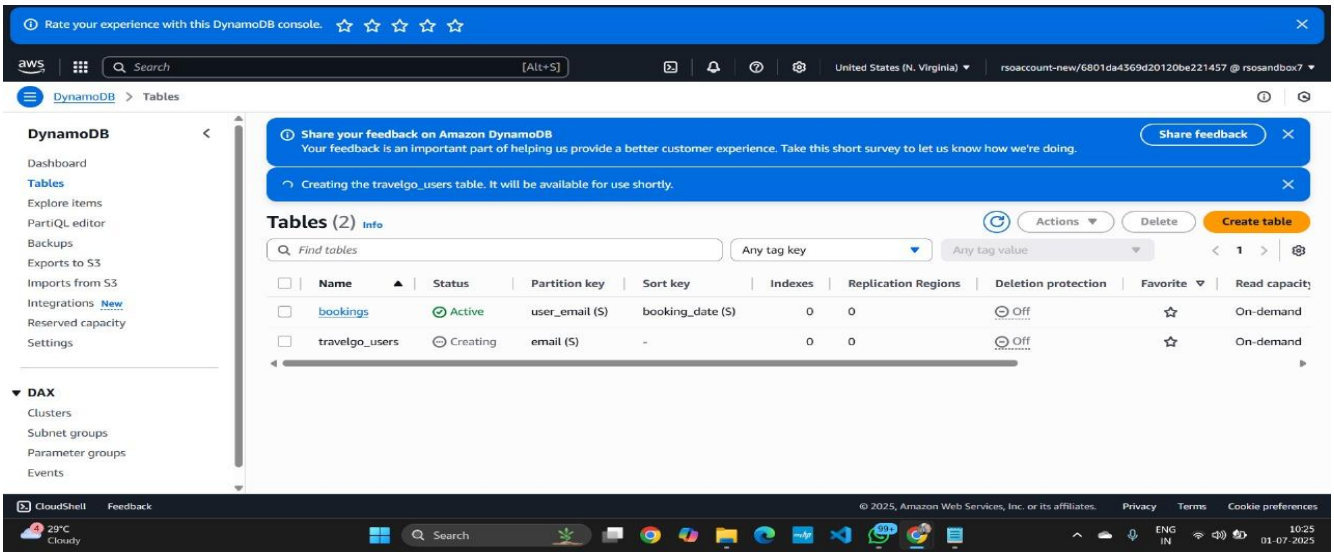




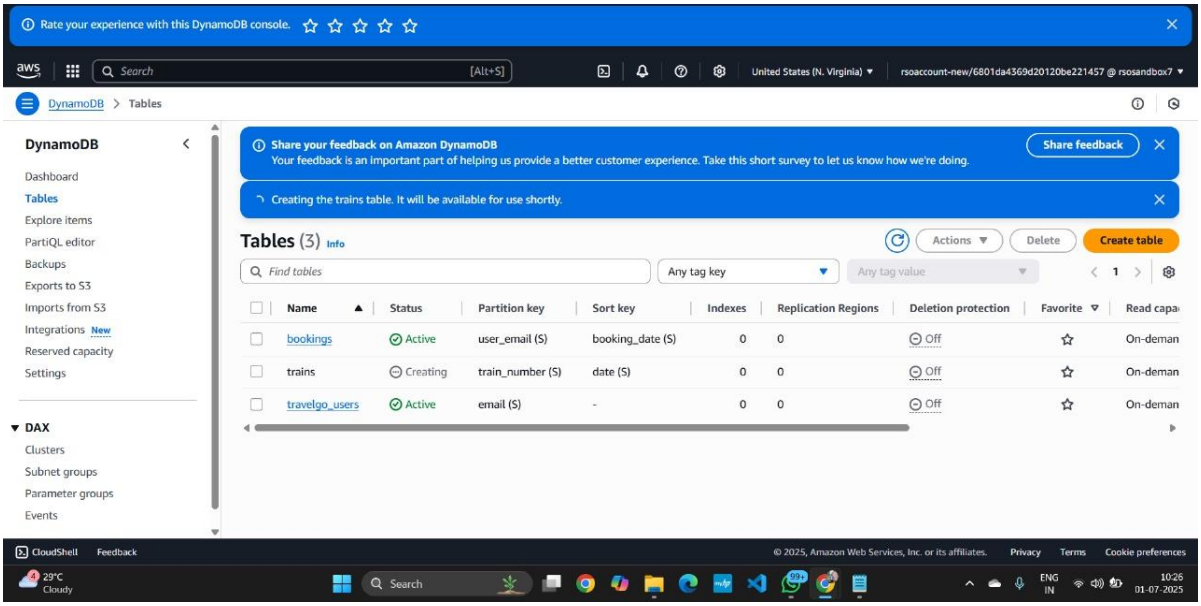
Activity 3.2: Create a DynamoDB table for storing registration details and book requests.

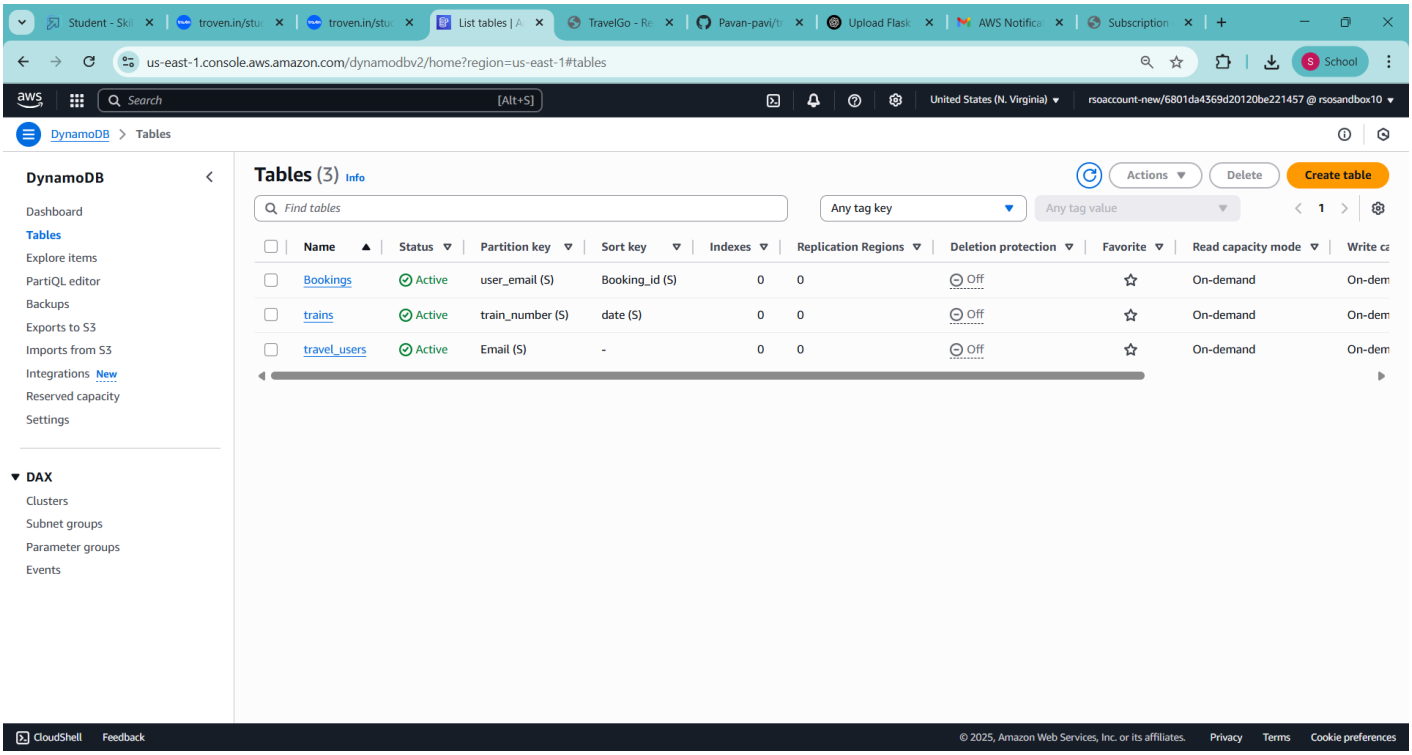
- Create Users table with partition key “Email ” with type String and click on create tables.


Search [Alt+S]
United States (N. Virginia)
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew91
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.
users
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.
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.
Enter the sort key name
String
1 to 255 characters and case sensitive.
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
Customize settings
Use these advanced features to make DynamoDB work better for your needs.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Rate your experience with this DynamoDB console.
Search [Alt+S]
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandbox7
DynamoDB > Tables
DynamoDB
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
Share your feedback on Amazon DynamoDB
Your feedback is an important part of helping us provide a better customer experience. Take this short survey to let us know how we're doing.
Share feedback
Creating the travelgo_users table. It will be available for use shortly.
Tables (2) Info
Actions
Delete
Create table
Find tables
Any tag key
Any tag value
Name
Status
Partition key
Sort key
Indexes
Replication Regions
Deletion protection
Favorite
Read capacity
bookings
Active
user_email (S)
booking_date (S)
0
0
Off
On-demand
travelgo_users
Creating
email (S)
-
0
0
Off
On-demand
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
29°C
Cloudy
Search
ENG
IN
10:25
01-07-2025

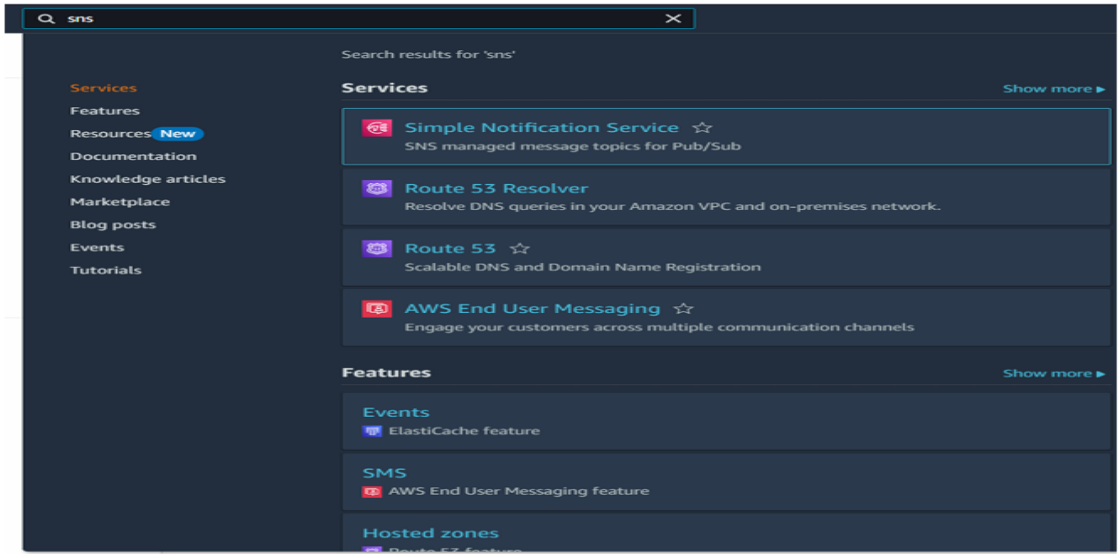
- Follow the same steps to create a Bookings for storing booking records with email as the partition key and booking_id as the sort key.



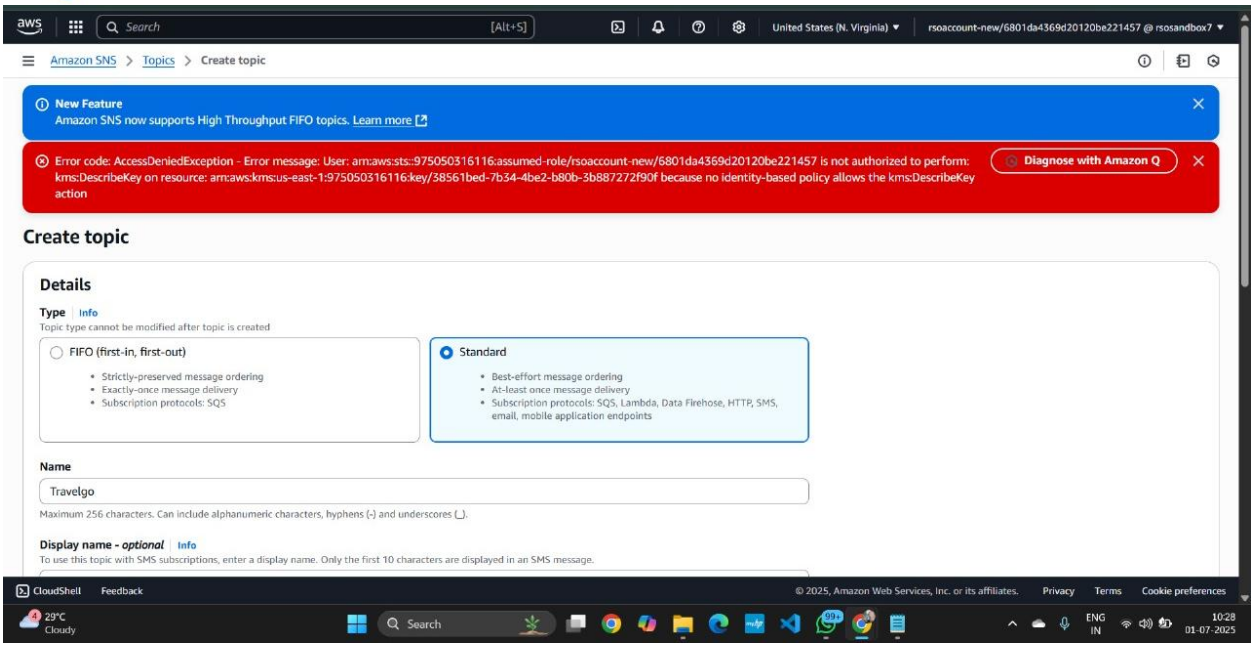


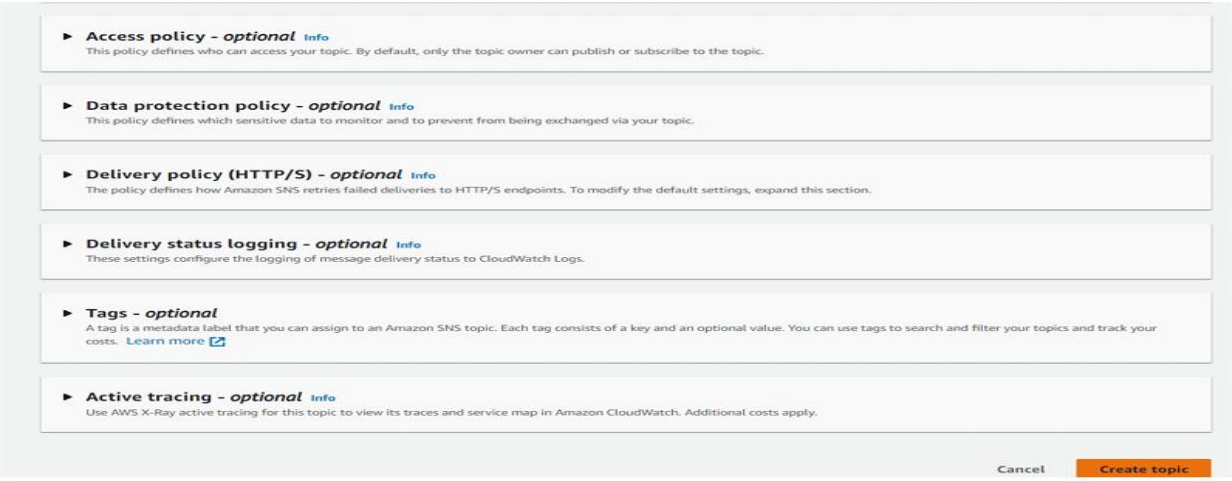
Milestone 4. SNS Notification Setup

- **Activiy4.1:** Create SNS topics for book request notifications**.**

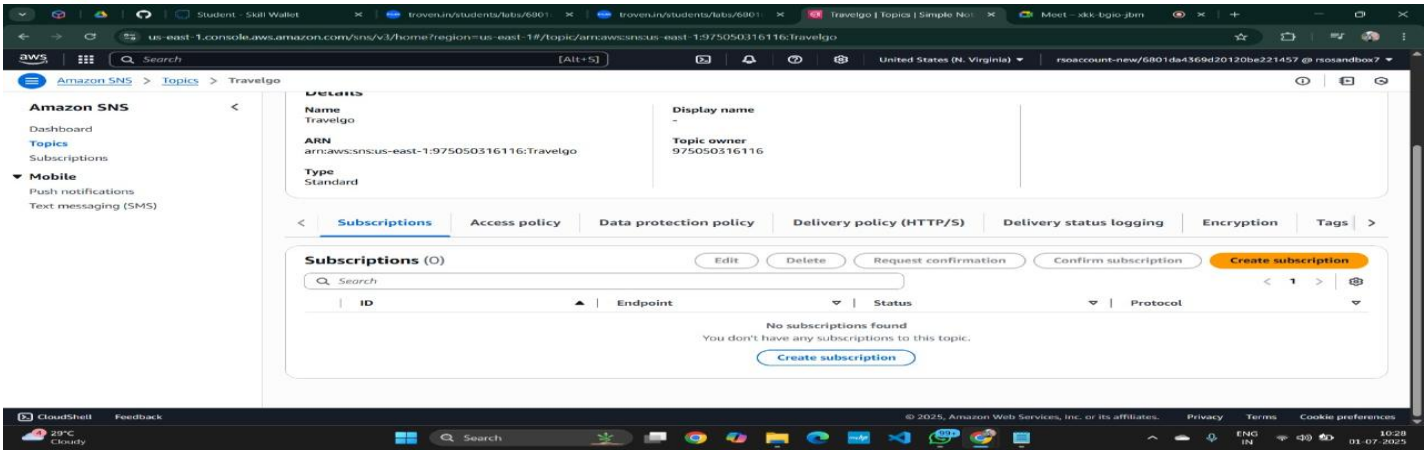


- Click on Create Topic and choose a name for the topic

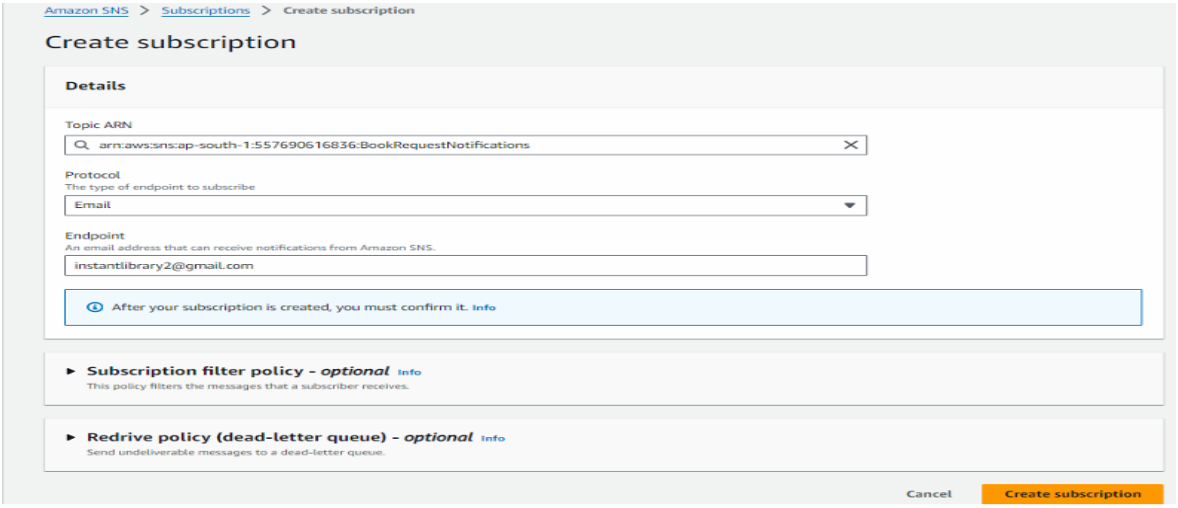


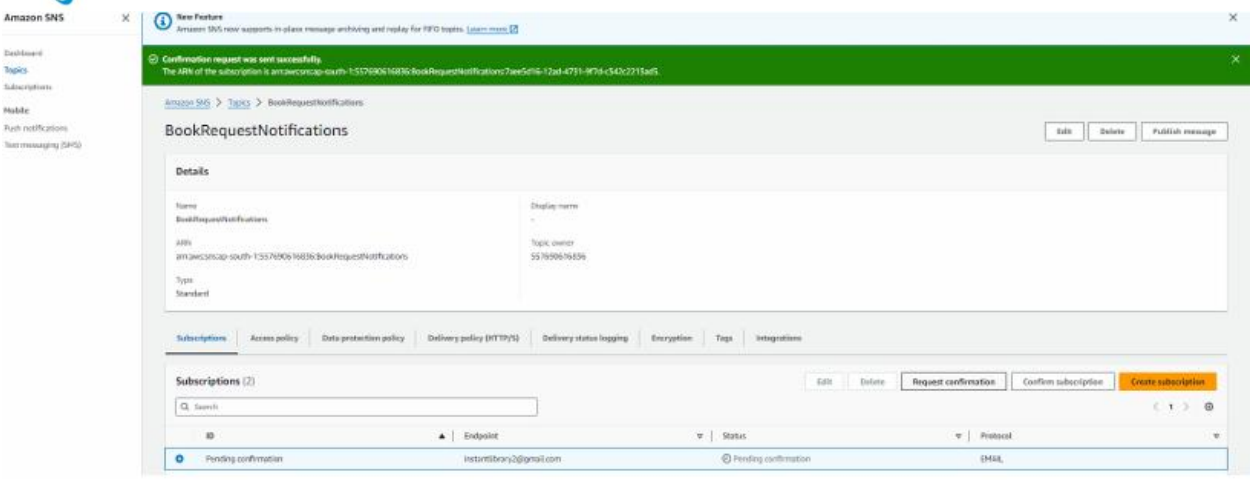


- Configure the SNS topic and note down the Topic ARN

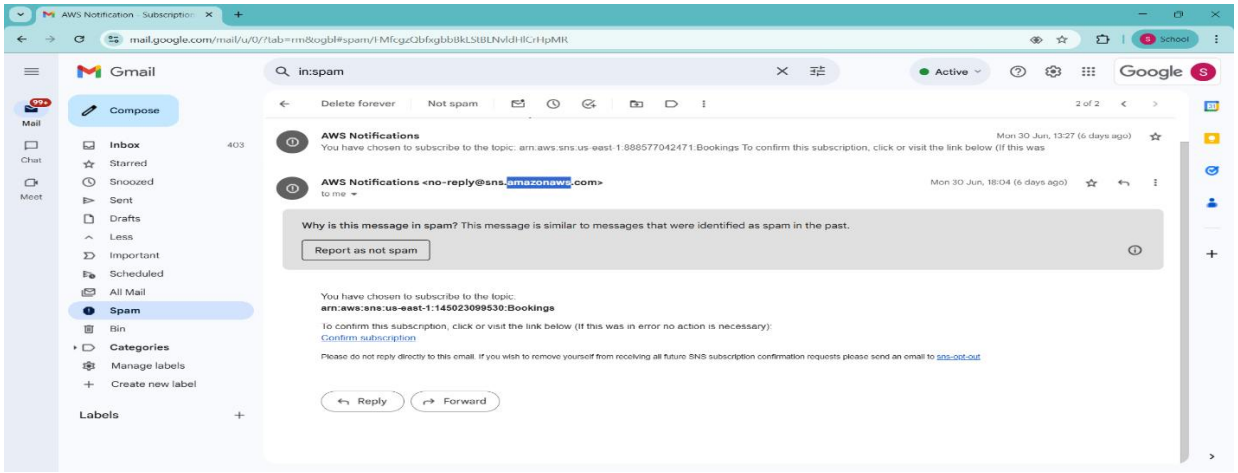


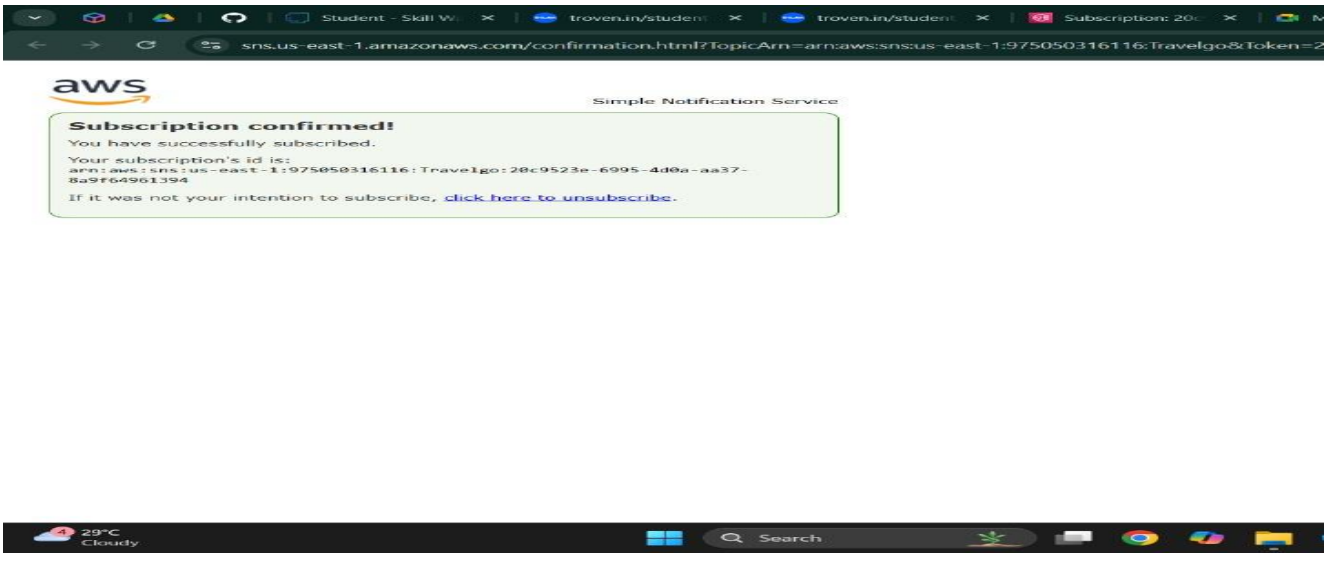
○ Subscribe users and library staff to SNS email notifications.





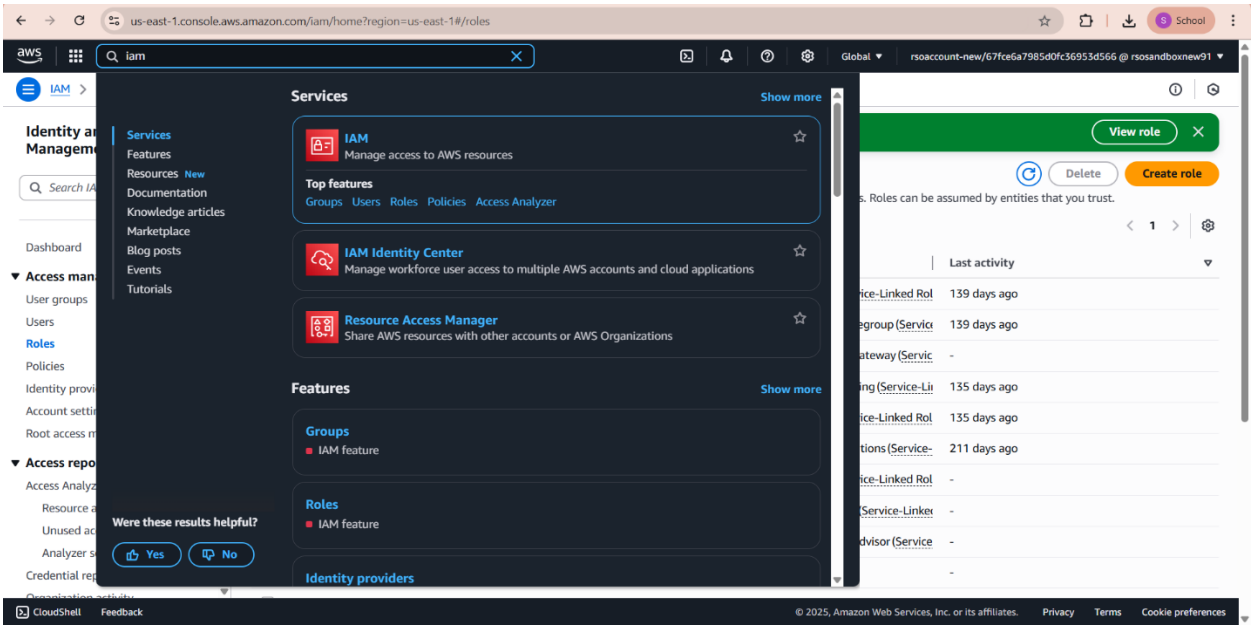
- After Comfirmation of Subscription going to Mail for comfirm Mail .

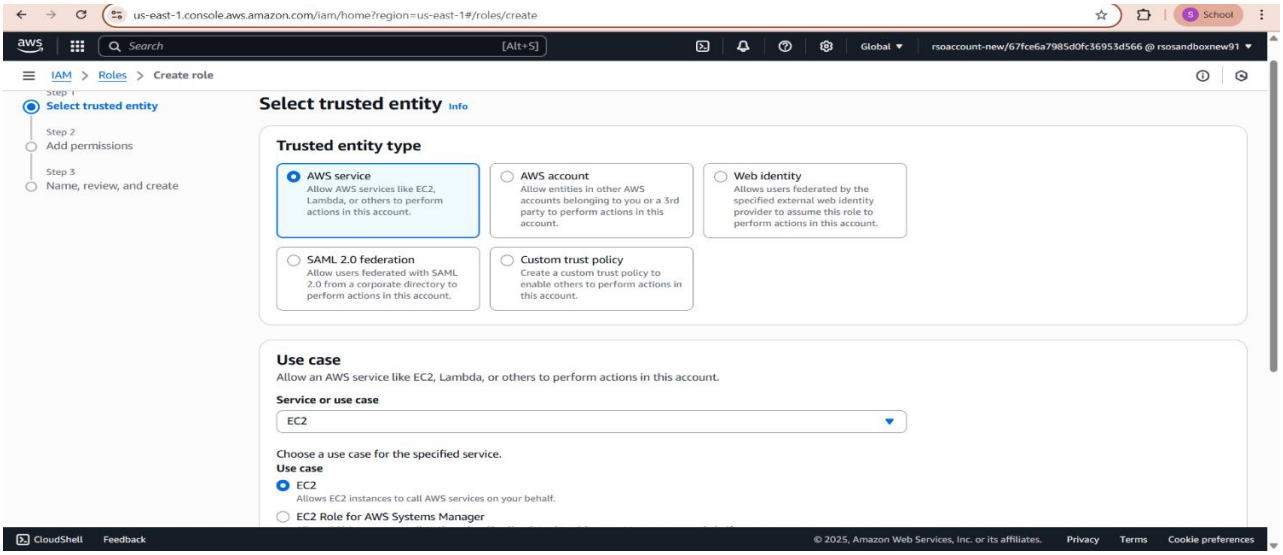




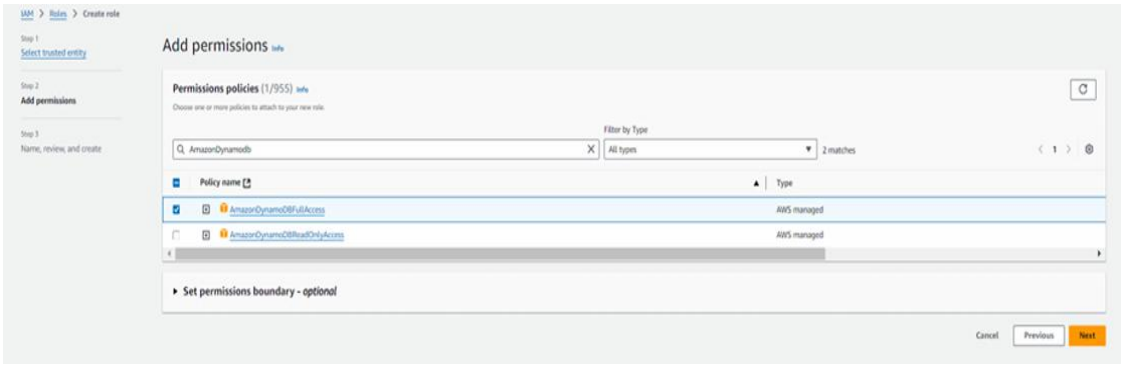
Milestone 5 : IAM Role Setup.

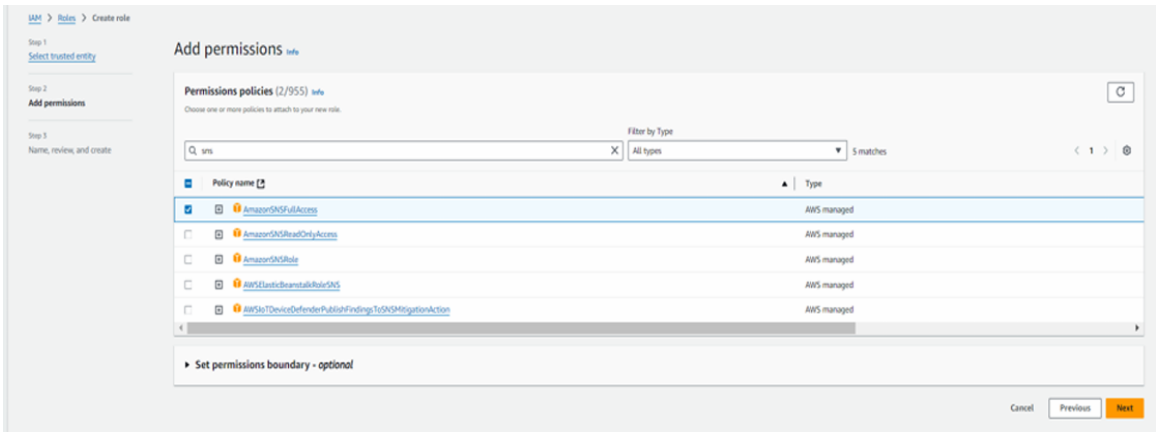
- **Activity 5.1 :** In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB .

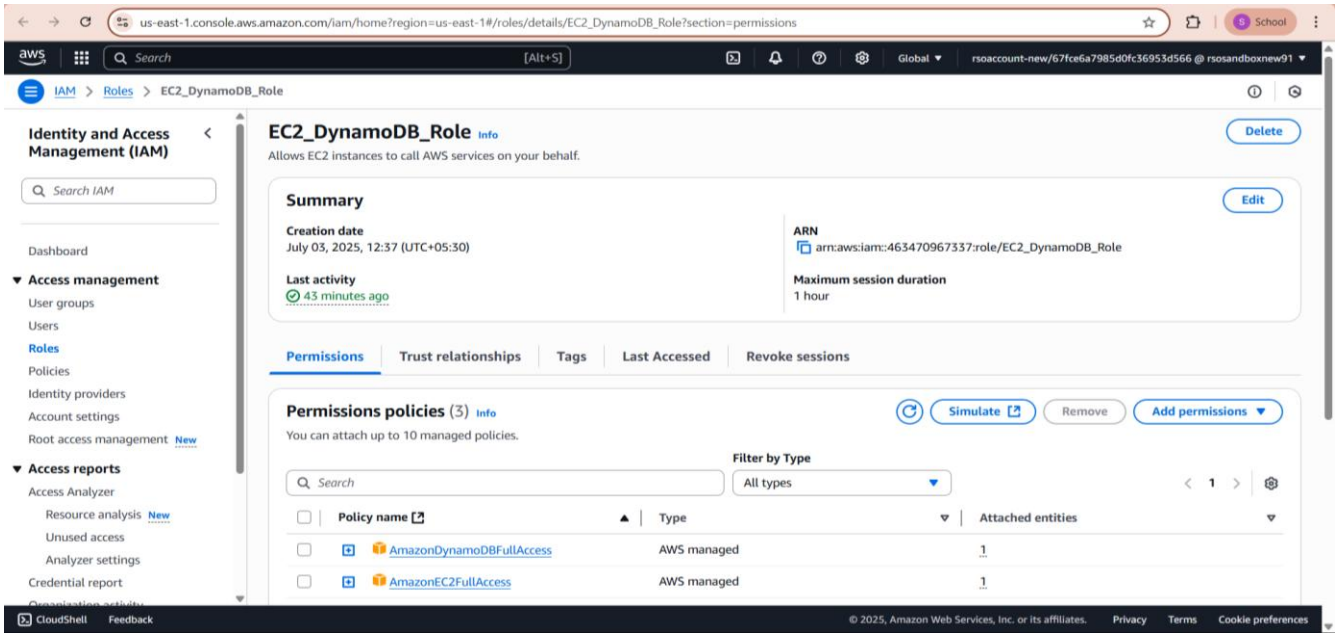




- **Activity 5.2 : Attach Policies**
- AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.
- AmazonSNSFullAccess: Grants EC2 the ability to send notifications via SNS.

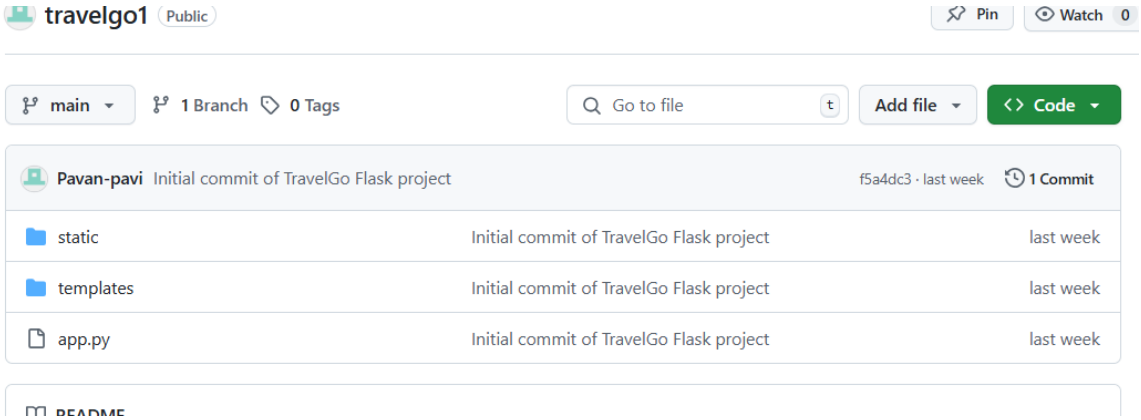

IAM > Roles > Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
Add permissions Info
Permissions policies (1/955) Info
Choose one or more policies to attach to your new role.
Filter by Type
AmazonDynamodb
All types
2 matches
Policy name
Type
AmazonDynamoDBFullAccess
AWS managed
AmazonDynamoDBReadOnlyAccess
AWS managed
Set permissions boundary - optional
Cancel
Previous
Next

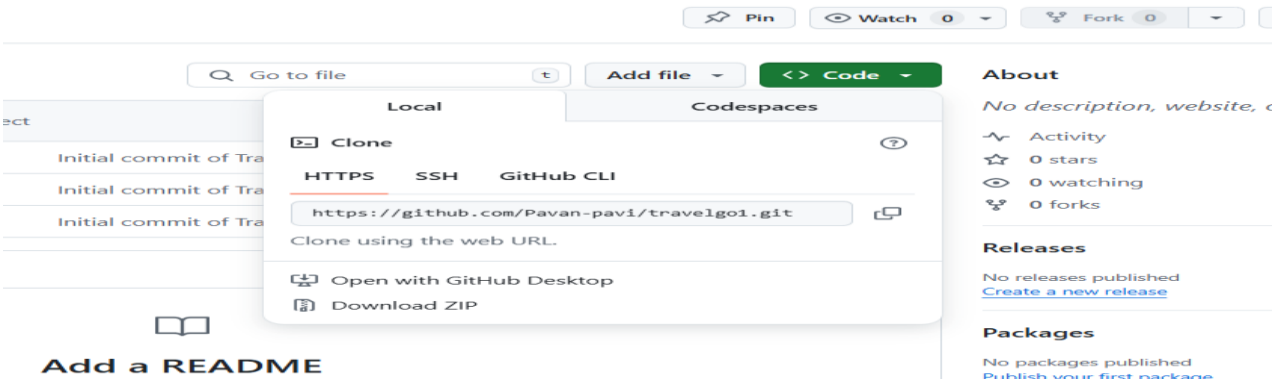

IAM > Roles > Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
Add permissions Info
Permissions policies (2/955) Info
Choose one or more policies to attach to your new role.
Filter by Type
sns
All types
5 matches
Policy name
Type
AmazonSNSFullAccess
AWS managed
AmazonSNSReadOnlyAccess
AWS managed
AmazonSNSRole
AWS managed
AWSElasticBeanstalkRoleSNS
AWS managed
AWSIoTDeviceDefenderPublishFindingsToSNSMitigationAction
AWS managed
Set permissions boundary - optional
Cancel
Previous
Next


us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles/details/EC2_DynamoDB_Role?section=permissions
Search
[Alt+S]
Global
rsoaccount-new/67fce6a7985d0fc36953d566 @ rsosandboxnew91
IAM > Roles > EC2_DynamoDB_Role
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management New
Access reports
Access Analyzer
Resource analysis New
Unused access
Analyzer settings
Credential report
EC2_DynamoDB_Role Info
Allows EC2 instances to call AWS services on your behalf.
Delete
Summary
Edit
Creation date
July 03, 2025, 12:37 (UTC+05:30)
ARN
arn:aws:iam::463470967337:role/EC2_DynamoDB_Role
Last activity
43 minutes ago
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.
Simulate
Remove
Add permissions
Filter by Type
Search
All types
Policy name
Type
Attached entities
AmazonDynamoDBFullAccess
AWS managed
1
AmazonEC2FullAccess
AWS managed
1
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Milestone 6. EC2 Instance Setup

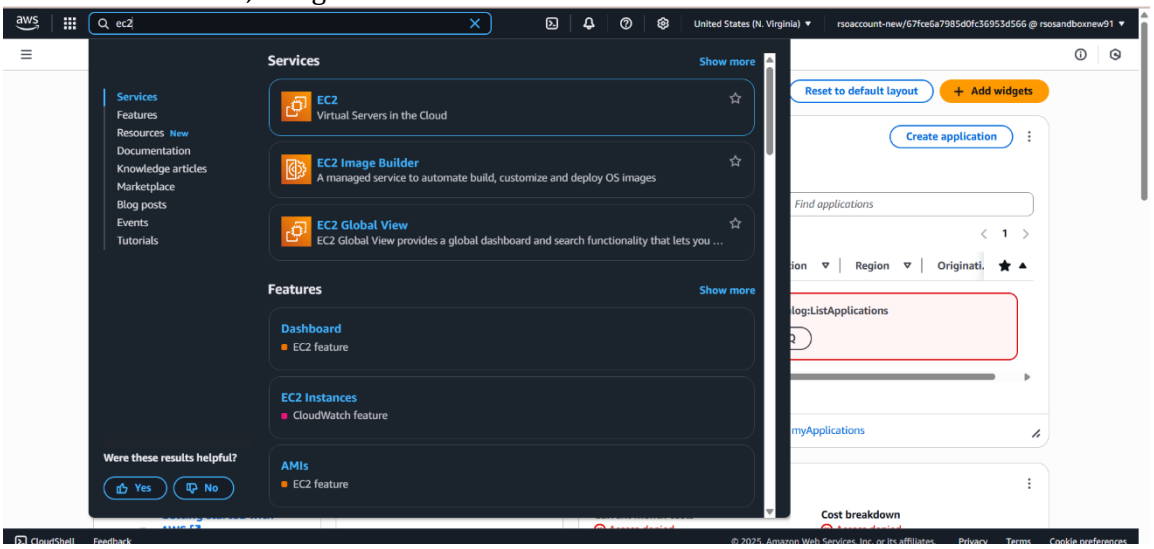
Note: Load your Flask app and Html files into GitHub repository .

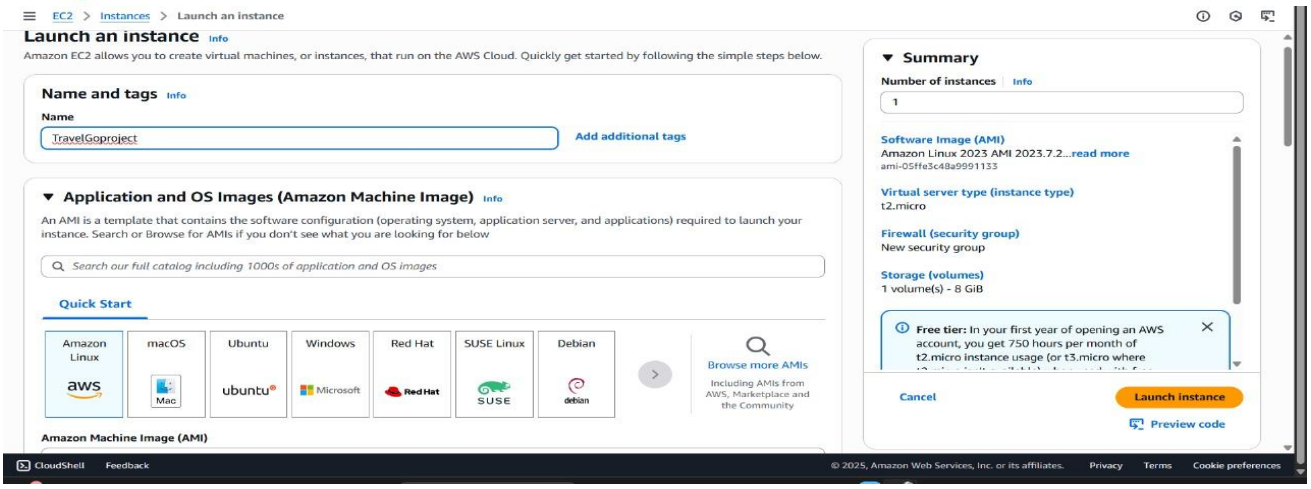


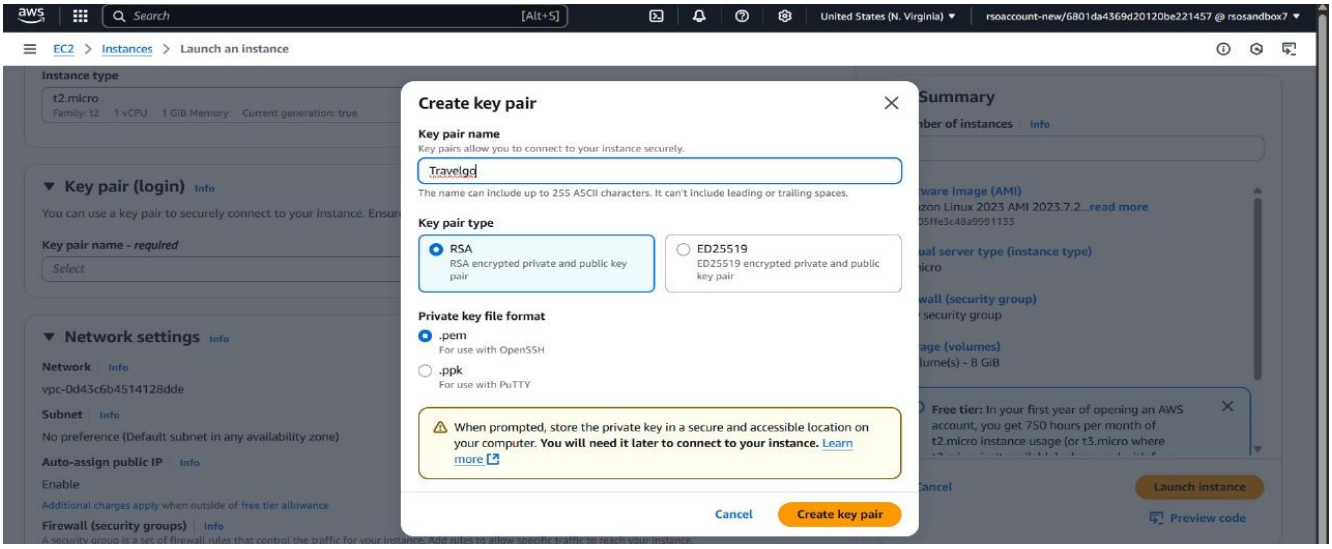


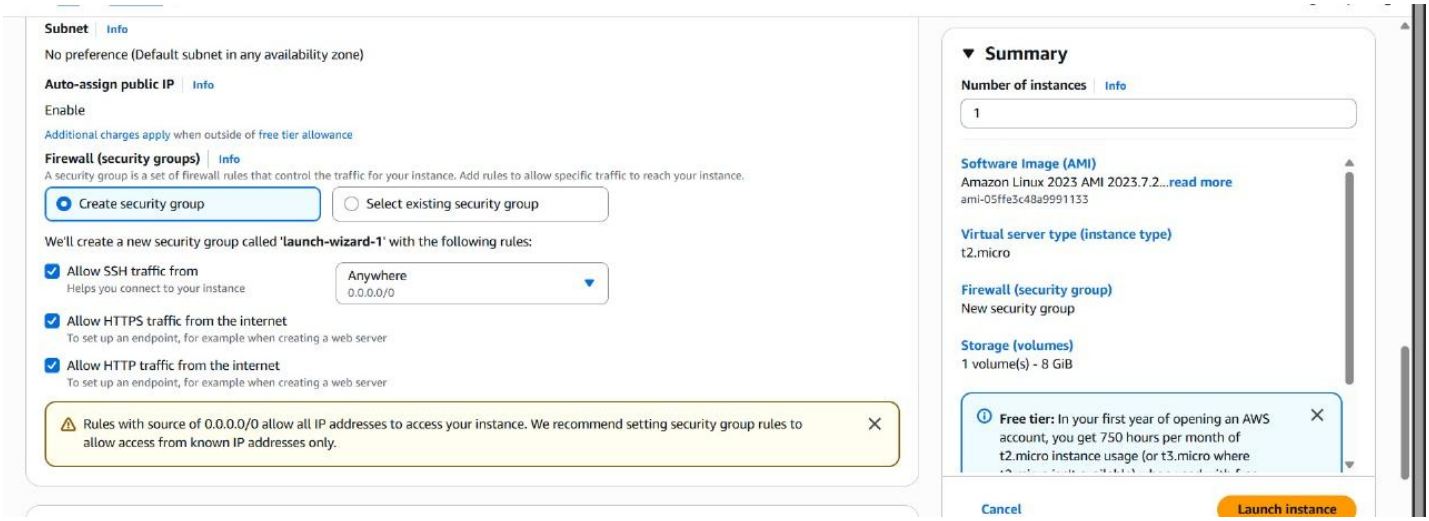
Activity 6.1: Launch an EC2 instance to host the Flask application.

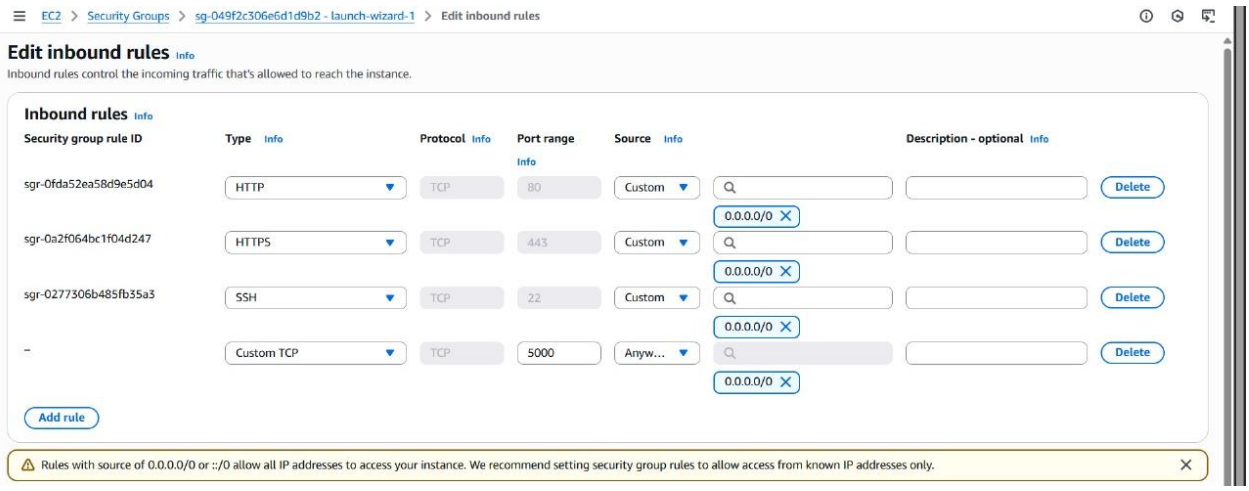
- In the AWS Console, navigate to EC2 and launch a new instance.

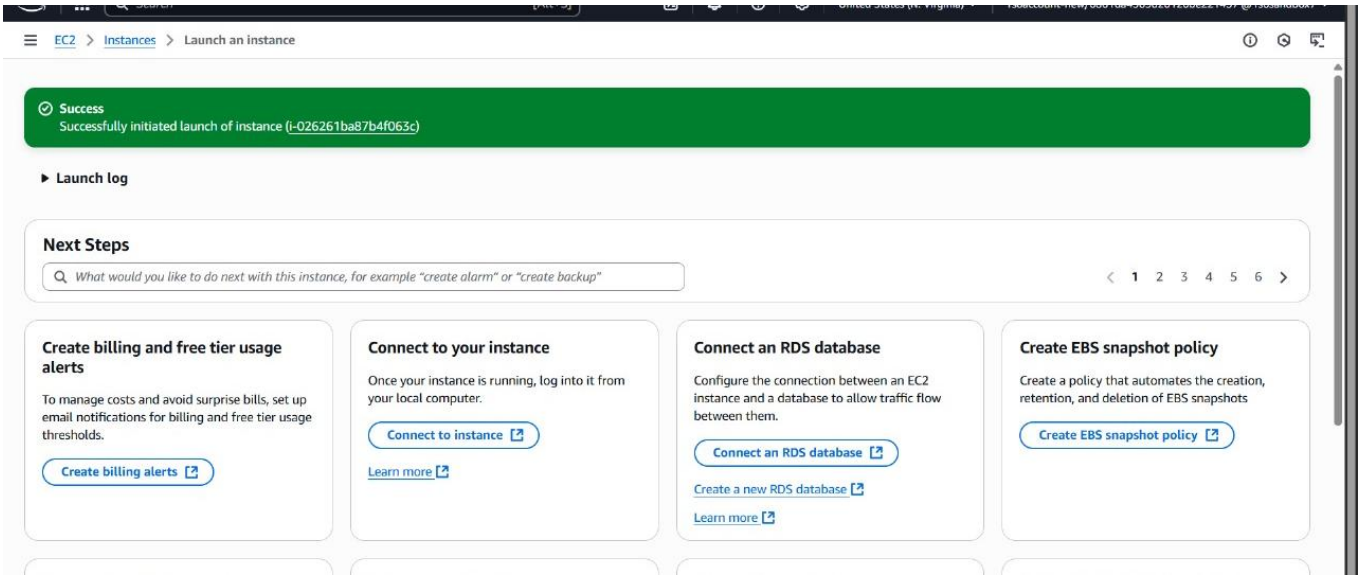


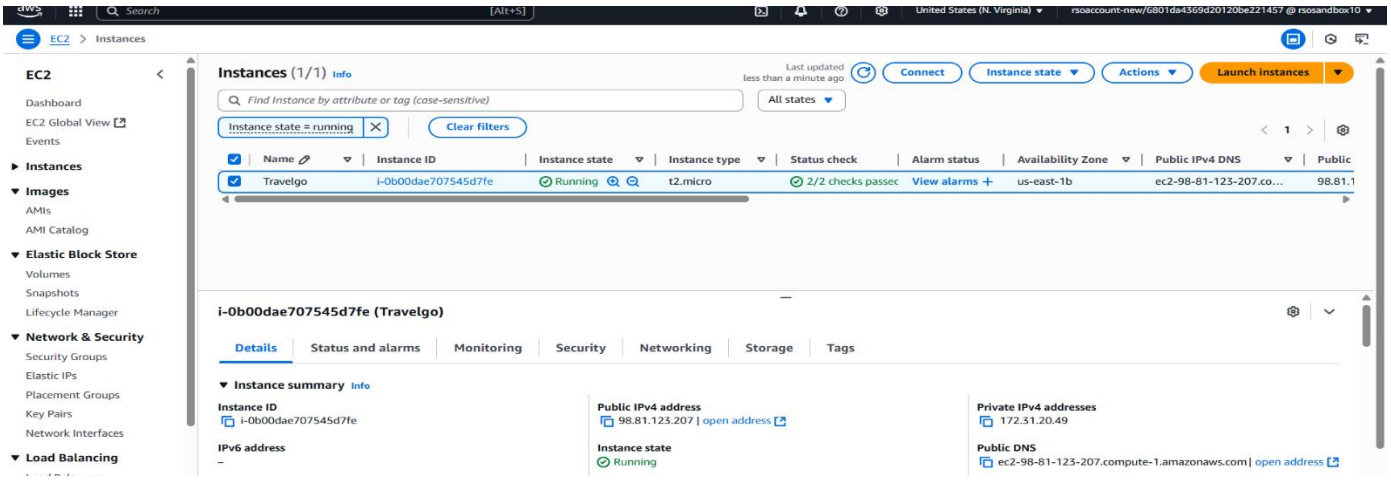

EC2 > Instances > Launch an instance
Launch an instance Info
Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.
Name and tags Info
Name
TravelGoproject
Add additional tags
Application and OS Images (Amazon Machine Image) Info
An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below
Search our full catalog including 1000s of application and OS images
Quick Start
Amazon Linux
macOS
Ubuntu
Windows
Red Hat
SUSE Linux
Debian
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community
Amazon Machine Image (AMI)
Summary
Number of instances Info
1
Software Image (AMI)
Amazon Linux 2023 AMI 2023.7.2...read more
ami-05ffe3c48a9991133
Virtual server type (instance type)
t2.micro
Firewall (security group)
New security group
Storage (volumes)
1 volume(s) - 8 GiB
Free tier: In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where
Cancel
Launch instance
Preview code
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

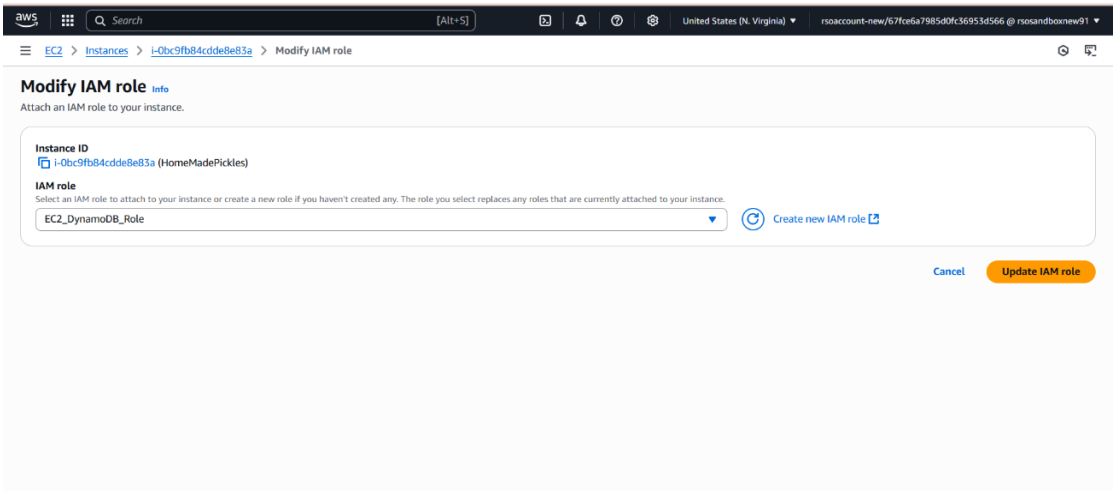

Search
[Alt+S]
United States (N. Virginia)
EC2 > Instances > Launch an instance
Instance type
t2.micro
Key pair (login) Info
Key pair name - required
Select
Network settings Info
Network Info
vpc-0d43c6b4514128dde
Subnet Info
No preference (Default subnet in any availability zone)
Auto-assign public IP Info
Enable
Additional charges apply when outside of free tier allowance
Firewall (security groups) Info
Create key pair
Key pair name
Key pairs allow you to connect to your instance securely.
Travelgc
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
Cancel
Create key pair
Summary
Launch instance
Preview code

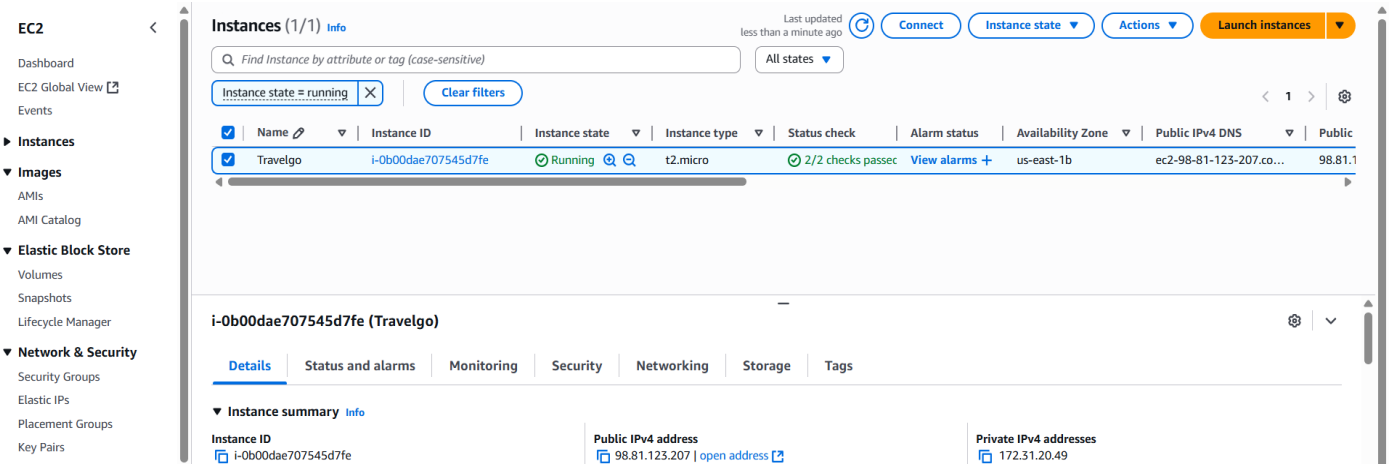

Subnet Info
No preference (Default subnet in any availability zone)
Auto-assign public IP Info
Enable
Additional charges apply when outside of free tier allowance
Firewall (security groups) Info
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.
Create security group
Select existing security group
We'll create a new security group called 'launch-wizard-1' with the following rules:
Allow SSH traffic from
Helps you connect to your instance
Anywhere
0.0.0.0/0
Allow HTTPS traffic from the internet
To set up an endpoint, for example when creating a web server
Allow HTTP traffic from the internet
To set up an endpoint, for example when creating a web server
Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.
Summary
Number of instances Info
1
Software Image (AMI)
Amazon Linux 2023 AMI 2023.7.2...read more
ami-05ffe3c48a9991133
Virtual server type (instance type)
t2.micro
Firewall (security group)
New security group
Storage (volumes)
1 volume(s) - 8 GiB
Free tier: In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where
Cancel
Launch instance

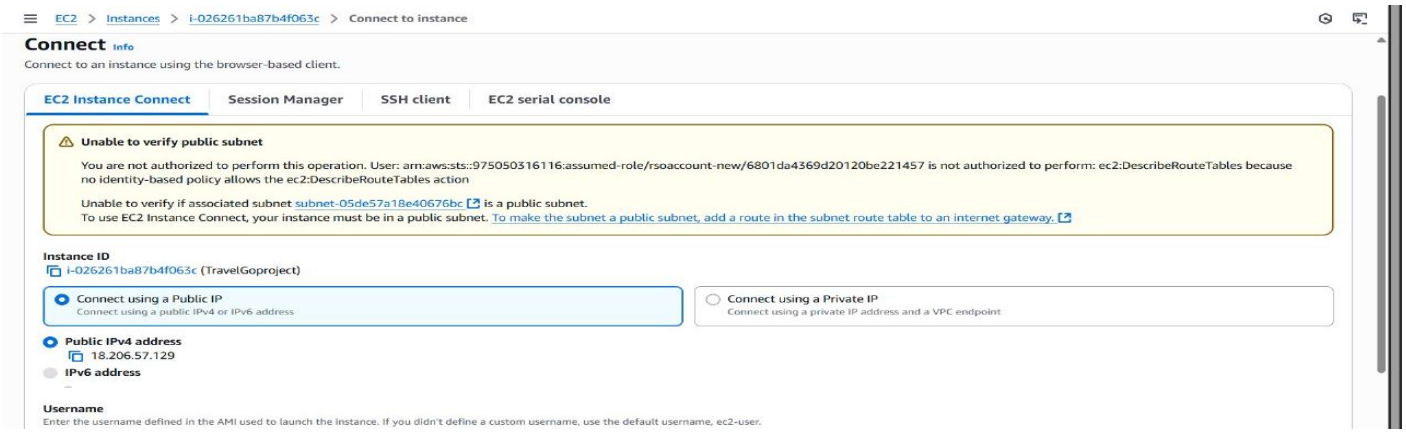

EC2 > Security Groups > sg-049f2c306e6d1d9b2 - launch-wizard-1 > Edit inbound rules
Edit inbound rules Info
Inbound rules control the incoming traffic that's allowed to reach the instance.
Inbound rules Info
Security group rule ID
Type Info
Protocol Info
Port range Info
Source Info
Description - optional Info
sgr-0fda52ea58d9e5d04
HTTP
TCP
80
Custom
0.0.0.0/0
Delete
sgr-0a2f064bc1f04d247
HTTPS
TCP
443
Custom
0.0.0.0/0
Delete
sgr-0277306b485fb35a3
SSH
TCP
22
Custom
0.0.0.0/0
Delete
–
Custom TCP
TCP
5000
Anyw...
0.0.0.0/0
Delete
Add rule
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.

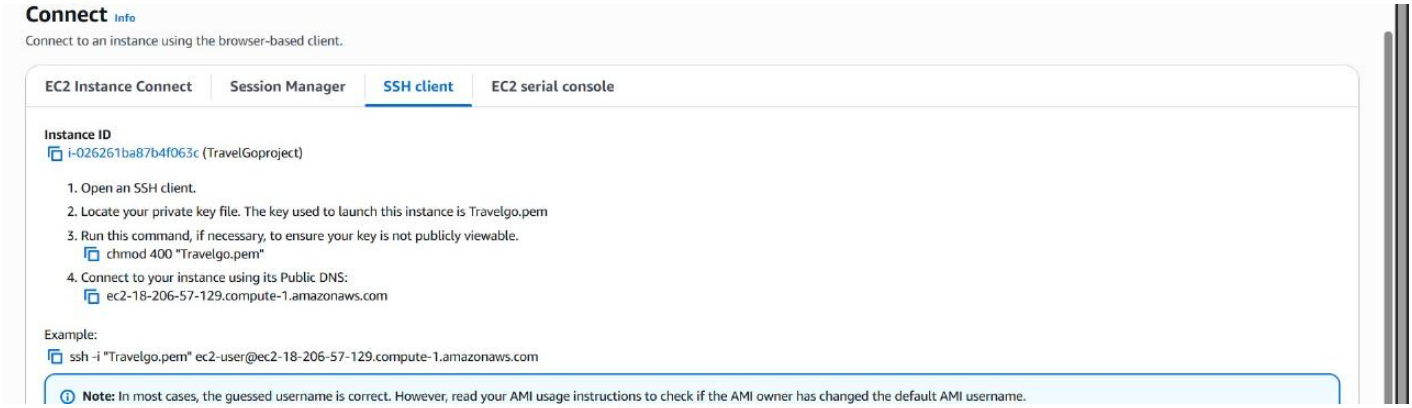

EC2 > Instances > Launch an instance
Success
Successfully initiated launch of instance (i-026261ba87b4f063c)
Launch log
Next Steps
What would you like to do next with this instance, for example "create alarm" or "create backup"
1 2 3 4 5 6
Create billing and free tier usage alerts
To manage costs and avoid surprise bills, set up email notifications for billing and free tier usage thresholds.
Create billing alerts
Connect to your instance
Once your instance is running, log into it from your local computer.
Connect to instance
Learn more
Connect an RDS database
Configure the connection between an EC2 instance and a database to allow traffic flow between them.
Connect an RDS database
Create a new RDS database
Learn more
Create EBS snapshot policy
Create a policy that automates the creation, retention, and deletion of EBS snapshots
Create EBS snapshot policy


EC2 > Instances
EC2
Dashboard
EC2 Global View
Events
Instances
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
Load Balancing
Instances (1/1) Info
Last updated less than a minute ago
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Instance state = running
Clear filters
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Public IPv4 DNS
Travelgo
i-0b00dae707545d7fe
Running
t2.micro
2/2 checks passed
View alarms
us-east-1b
ec2-98-81-123-207.co...
i-0b00dae707545d7fe (Travelgo)
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Instance summary Info
Instance ID
i-0b00dae707545d7fe
IPv6 address
–
Public IPv4 address
98.81.123.207 | open address
Instance state
Running
Private IPv4 addresses
172.31.20.49
Public DNS
ec2-98-81-123-207.compute-1.amazonaws.com | open address


Search [Alt+S]
United States (N. Virginia)
rsoaccount-new/67fce6a7985d0fc56953d566 @ rsosandboxnew91
EC2 > Instances > i-0bc9fb84cdde8e83a > Modify IAM role
Modify IAM role Info
Attach an IAM role to your instance.
Instance ID
i-0bc9fb84cdde8e83a (HomeMadePickles)
IAM role
Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.
EC2_DynamoDB_Role
Create new IAM role
Cancel
Update IAM role


EC2
Dashboard
EC2 Global View
Events
Instances
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Instances (1/1) Info
Last updated less than a minute ago
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Instance state = running
Clear filters
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Public IPv4 DNS
Travelgo
i-0b00dae707545d7fe
Running
t2.micro
2/2 checks passec
View alarms
us-east-1b
ec2-98-81-123-207.co...
i-0b00dae707545d7fe (Travelgo)
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Instance summary Info
Instance ID
i-0b00dae707545d7fe
Public IPv4 address
98.81.123.207 | open address
Private IPv4 addresses
172.31.20.49


EC2 > Instances > i-026261ba87b4f063c > Connect to instance
Connect Info
Connect to an instance using the browser-based client.
EC2 Instance Connect
Session Manager
SSH client
EC2 serial console
Unable to verify public subnet
You are not authorized to perform this operation. User: arn:aws:sts::975050316116: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-05de57a18e40676bc 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-026261ba87b4f063c (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
18.206.57.129
IPv6 address
Username
Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.



Milestone 7: Deployment on EC2

Activity 7.1: Install Software on the EC2 Instance

- **Install Python3, Flask, and Git:**
- **On Amazon Linux 2:**
- **sudo yum update -y**
- **sudo yum install python3 git**
- **sudo pip3 install flask boto3**
- **Verify Installations:**
- **flask --version**
- **git –version**

Activity 7.2 : Clone Your Flask Project from GitHub.

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

Run :’ git clone https://github.com/Pavan-pavi/travelgo1 ’ .

This will download the Project to Ec2 instance .

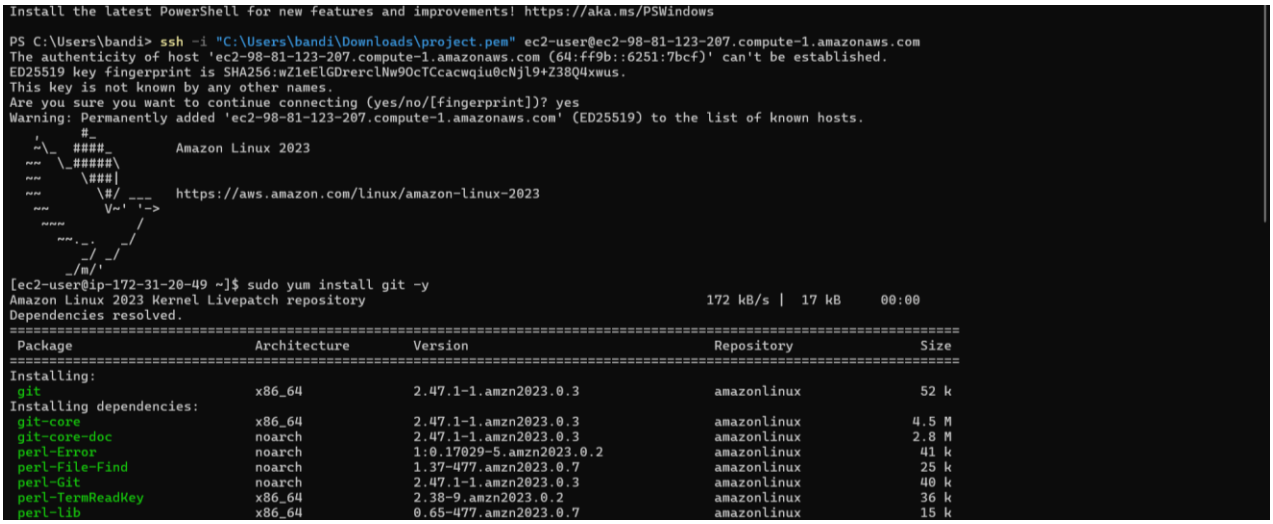
To navigate to the project directory, run the following command:

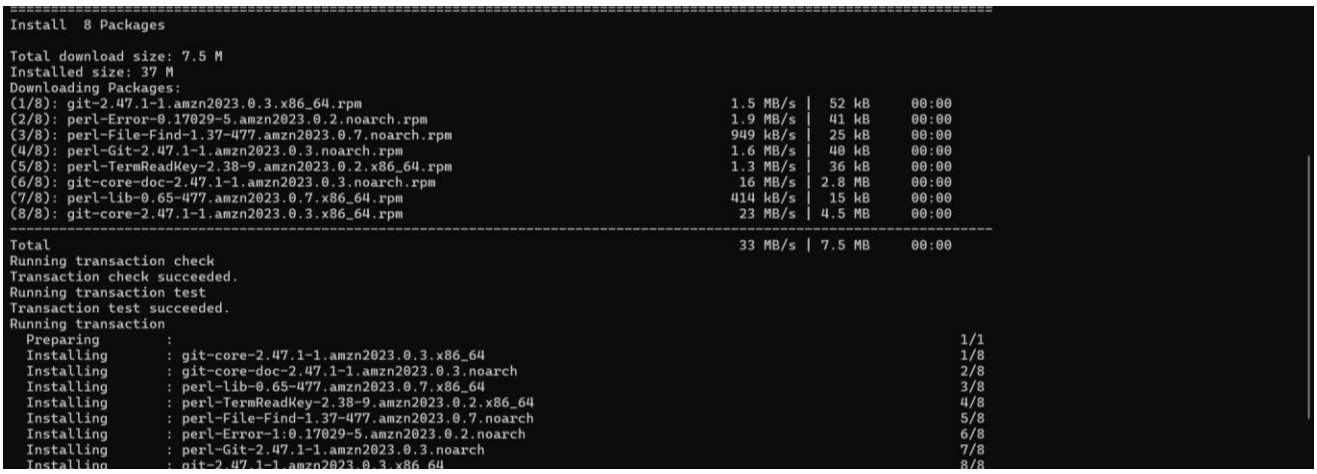
cd InstantLibrary

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





```
  Downloading MarkupSafe-3.0.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (20 kB)
Collecting itsdangerous>=2.2.0
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting jinja2>=3.1.2
  Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)
     |████████████████████████████████| 134 kB 79.0 MB/s
Collecting zipp>=3.20
  Downloading zipp-3.23.0-py3-none-any.whl (10 kB)
Installing collected packages: zipp, markupsafe, werkzeug, jinja2, itsdangerous, importlib-metadata, click, blinker, flask
Successfully installed blinker-1.9.0 click-8.1.8 flask-3.1.1 importlib-metadata-8.7.0 itsdangerous-2.2.0 jinja2-3.1.6 markupsafe-3.0.2 werkzeug-3.1.3 zipp-3
.23.0
Defaulting to user installation because normal site-packages is not writeable
Collecting boto3
  Downloading boto3-1.39.0-py3-none-any.whl (139 kB)
     |████████████████████████████████| 139 kB 13.9 MB/s
Collecting botocore<1.40.0,>=1.39.0
  Downloading botocore-1.39.0-py3-none-any.whl (13.8 MB)
     |████████████████████████████████| 13.8 MB 36.5 MB/s
Collecting s3transfer<0.14.0,>=0.13.0
  Downloading s3transfer-0.13.0-py3-none-any.whl (85 kB)
     |████████████████████████████████| 85 kB 8.4 MB/s
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (from boto3) (0.10.0)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.0->boto3) (2.8.1)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.40.0,>=1.39.0->boto3) (1.25.10)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.40.0,>=1.39.0->boto3) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed boto3-1.39.0 botocore-1.39.0 s3transfer-0.13.0
[ec2-user@ip-172-31-20-49 travelgo1]$ 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://172.31.20.49:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
```

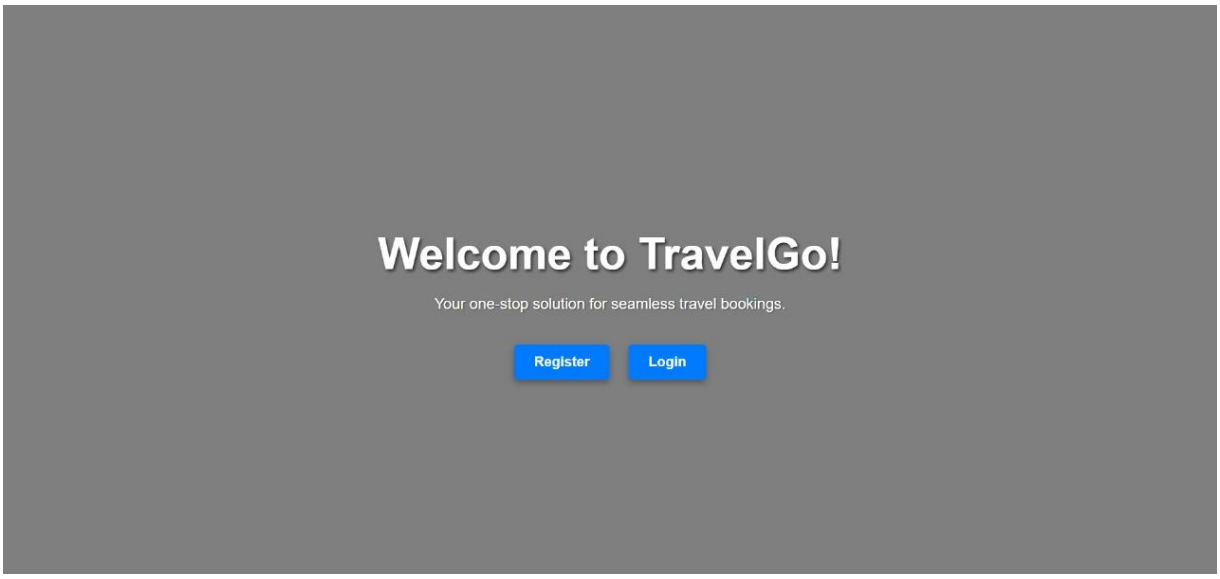
Access the website through:

Public IPs: http://98.81.123.207:5000/

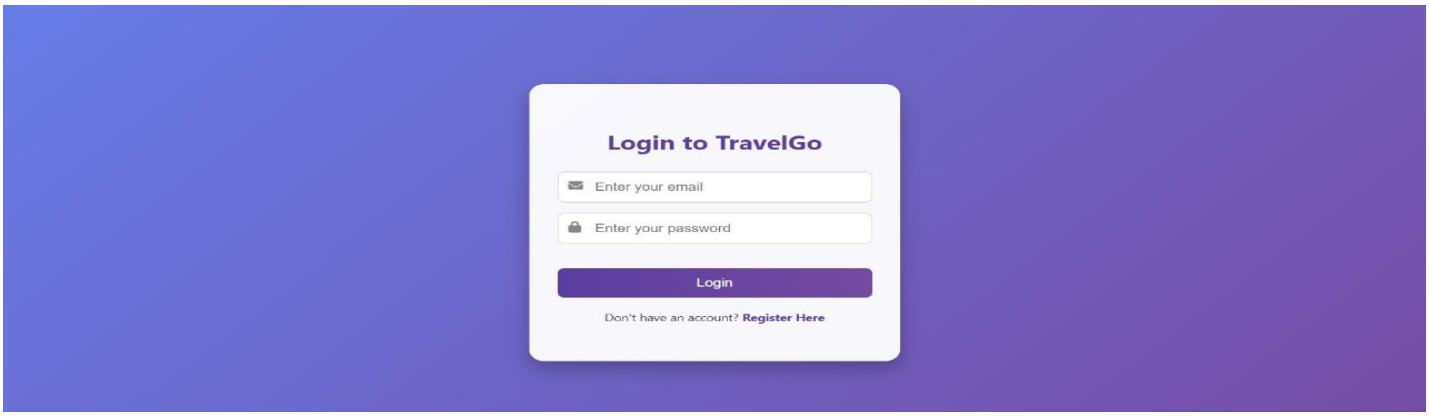
Milestone 8. Testing and Deployment

Activity 8.1 : Conduct functional testing to verify user signup, login, buy/sell stocks and notifications.

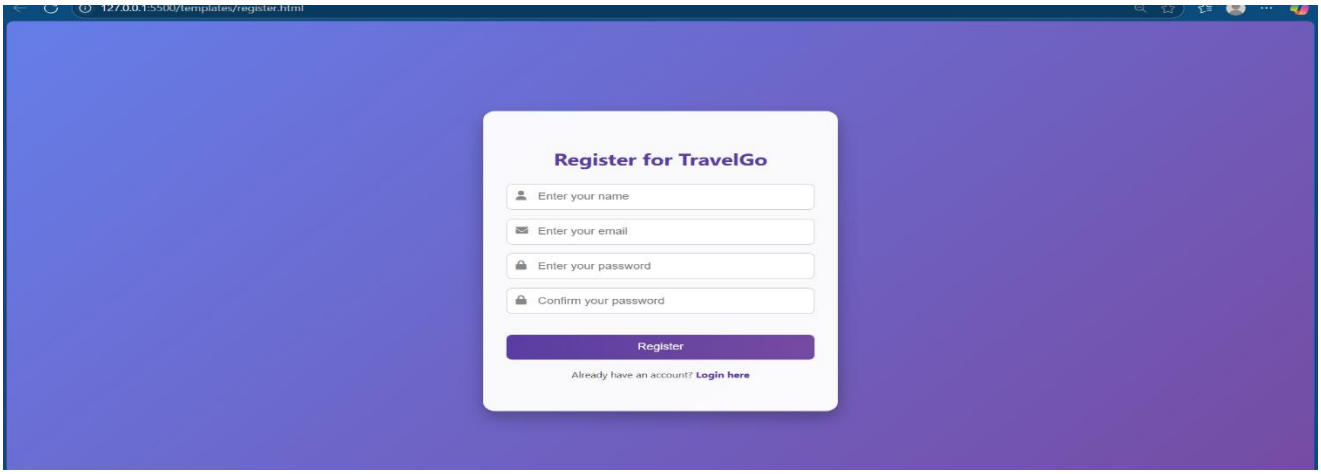
HOME PAGE:



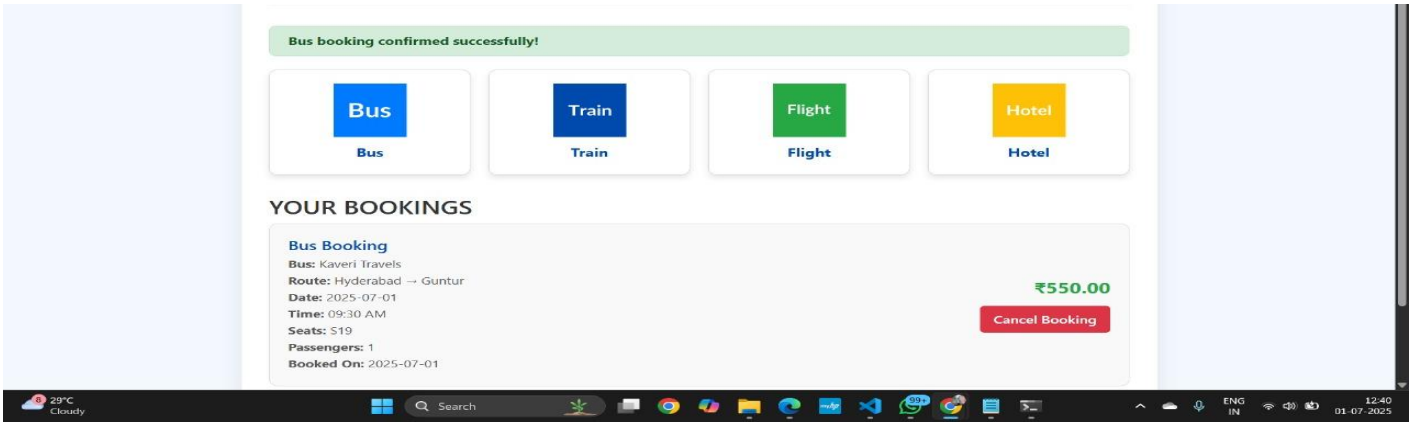
LOGIN PAGE :



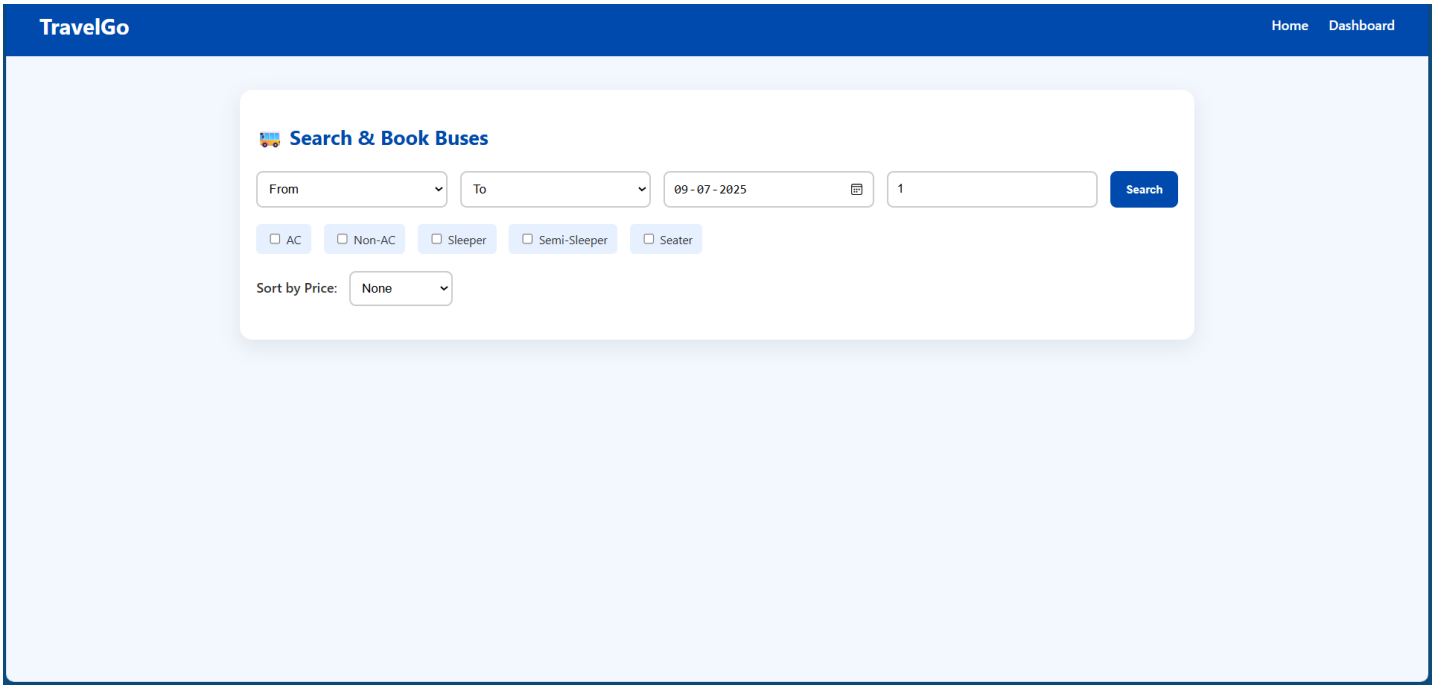
SIGNUP PAGE :



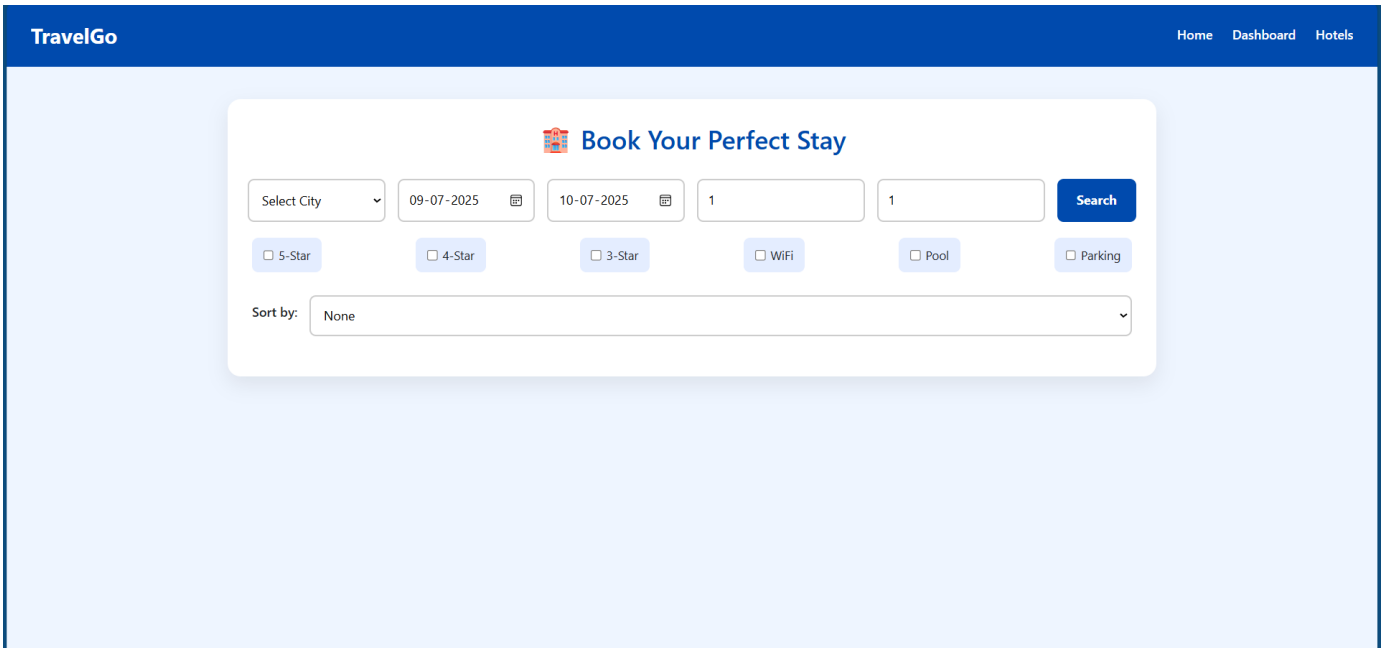
Dashboard:



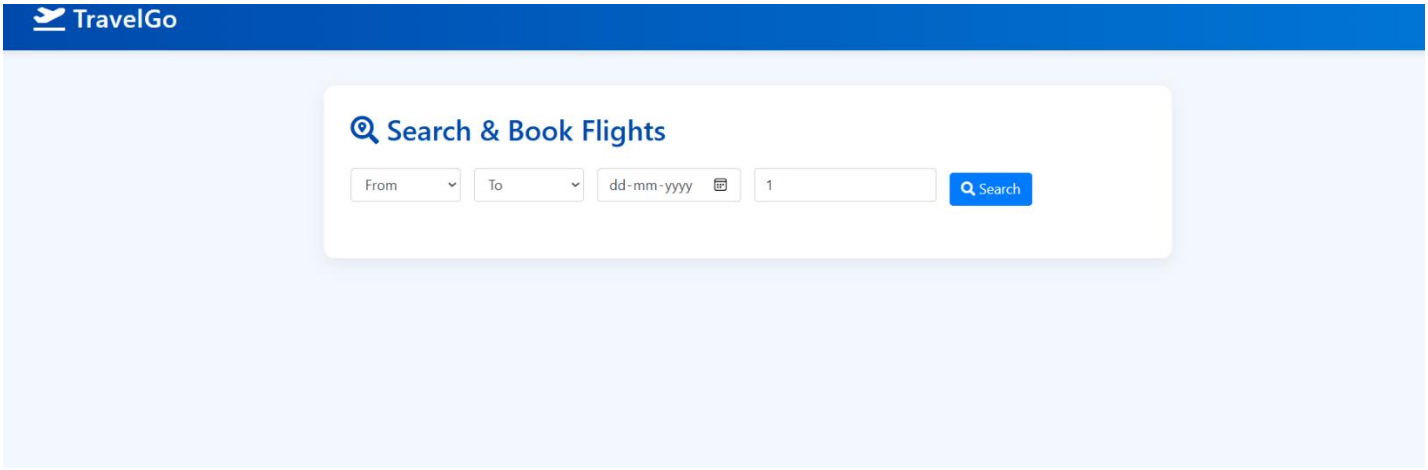
BUS PAGE :



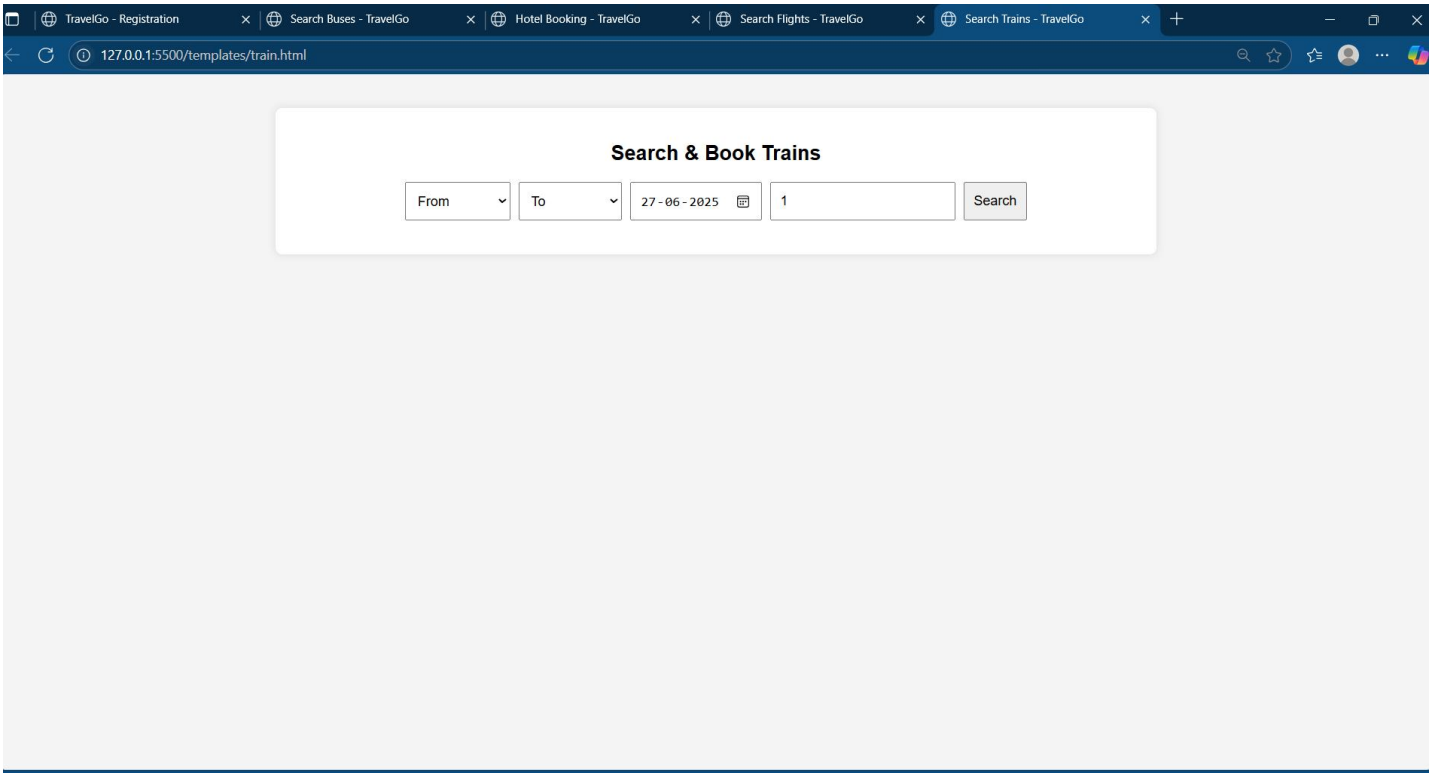
HOTEL PAGE :



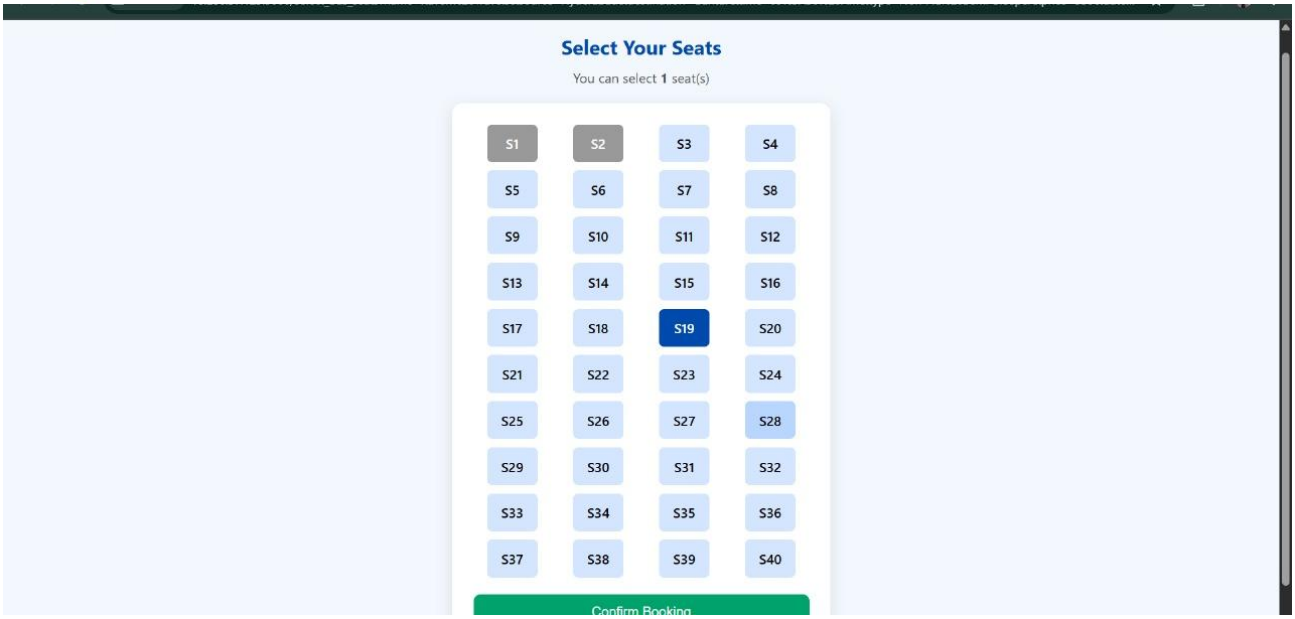
FLIGHT PAGE :



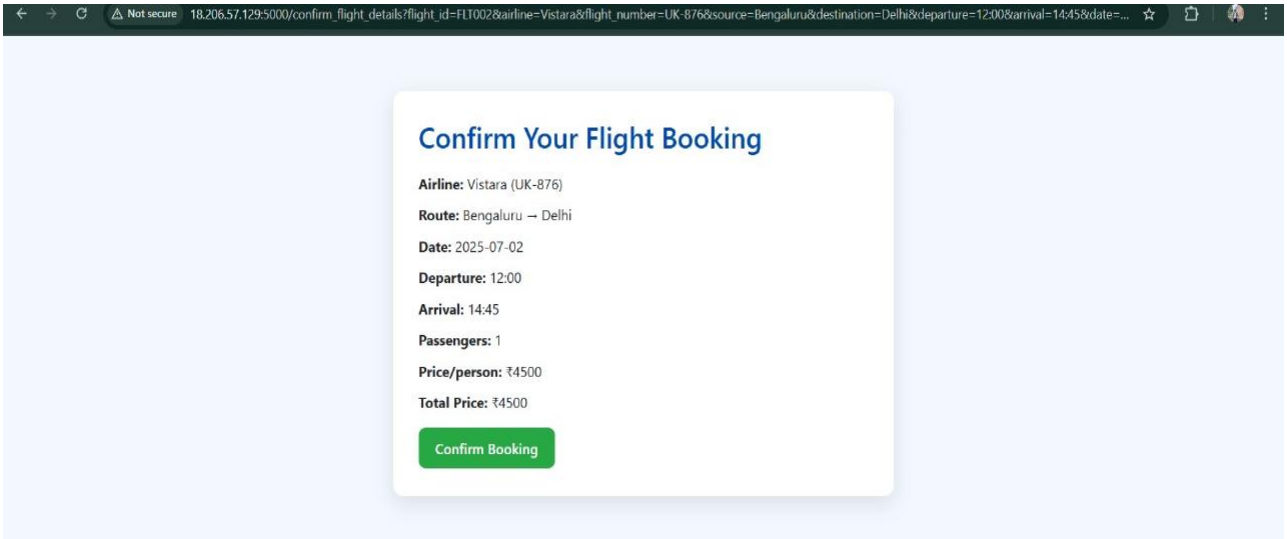
TRAIN PAGE :



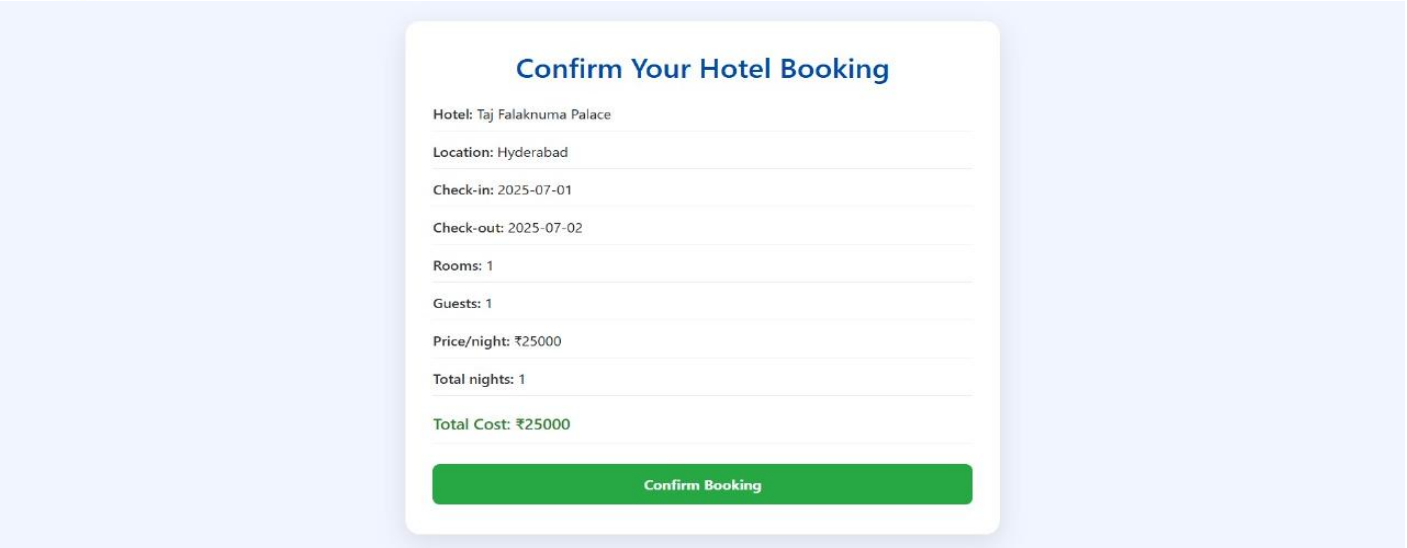
SEAT SELECT PAGE :



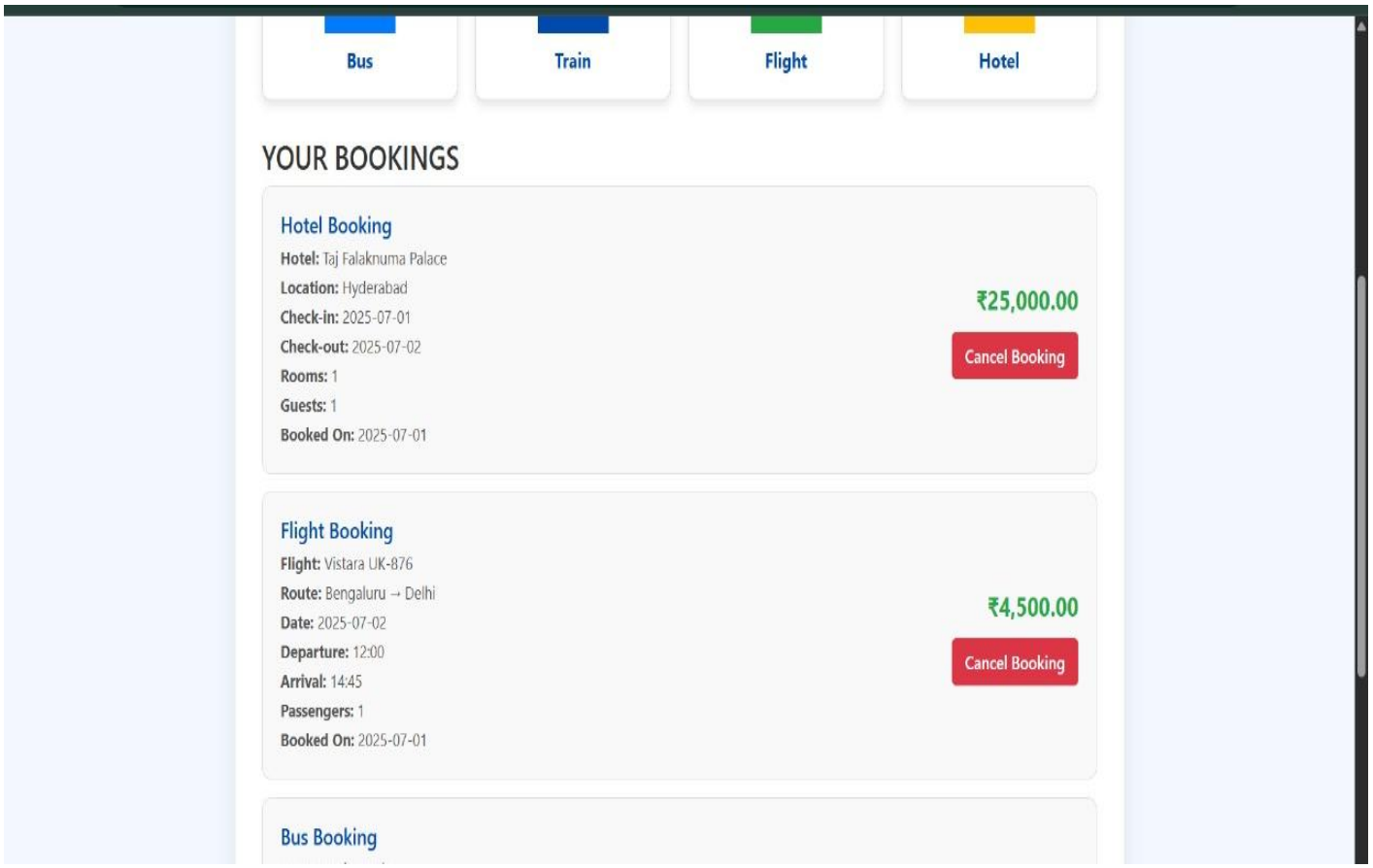
FLIGHT BOOKING PAGE :



HOTEL BOOKING PAGE :


Confirm Your Hotel Booking
Hotel: Taj Falaknuma Palace
Location: Hyderabad
Check-in: 2025-07-01
Check-out: 2025-07-02
Rooms: 1
Guests: 1
Price/night: ₹25000
Total nights: 1
Total Cost: ₹25000
Confirm Booking

BOOKINGS :


Bus
Train
Flight
Hotel
YOUR BOOKINGS
Hotel Booking
Hotel: Taj Falaknuma Palace
Location: Hyderabad
Check-in: 2025-07-01
Check-out: 2025-07-02
Rooms: 1
Guests: 1
Booked On: 2025-07-01
₹25,000.00
Cancel Booking
Flight Booking
Flight: Vistara UK-876
Route: Bengaluru → Delhi
Date: 2025-07-02
Departure: 12:00
Arrival: 14:45
Passengers: 1
Booked On: 2025-07-01
₹4,500.00
Cancel Booking
Bus Booking

Conclusion :

The TravelGo Website has been successfully developed and deployed using a scalable and cloud-native architecture. Leveraging AWS services such as EC2 for hosting, DynamoDB for real-time data management, and SNS for instant booking and cancellation notifications, the platform provides a seamless travel booking experience for users. TravelGo enables registered users to search and book buses, trains, flights, and hotels in a centralized, intuitive interface, eliminating the complexities of navigating multiple travel services.

The cloud infrastructure ensures high availability and smooth performance even during peak usage, while the Flask backend ensures efficient handling of user authentication, dynamic booking flows, and data transactions. Real-time notification integration via AWS SNS allows users to receive booking confirmations and cancellations immediately via email, improving communication and user engagement.

In summary, the TravelGo Website offers a modern, reliable, and user-friendly solution for managing travel and accommodation needs. It highlights the potential of cloud-based platforms in building unified travel systems, simplifying operations, and enhancing the overall user experience.

……..THE END……