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

Project Description:

TravelsGo is a comprehensive, 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 for backend development, 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 various 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 userfriendly interface supports dynamic seat selection for buses, hotel filtering based on preferences such as luxury or budget, and provides booking summaries along with centralized cancellation management. By combining cloud scalability, responsive design, and secure session handling, TravelGo delivers a seamless and real-time travel planning experience for users.

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

TravelGo offers a unified platform for booking buses, trains, flights, and hotels, allowing users to seamlessly plan entire trips. For instance, a user can log in, select their preferred transport, and then book accommodation, all within one interface. Flask efficiently manages real-time retrieval of diverse travel listings and processes user input Hosted on AWS EC2, the platform remains highly responsive even during peak traffic, ensuring multiple users can browse and book without any delays, providing a truly integrated travel planning experience

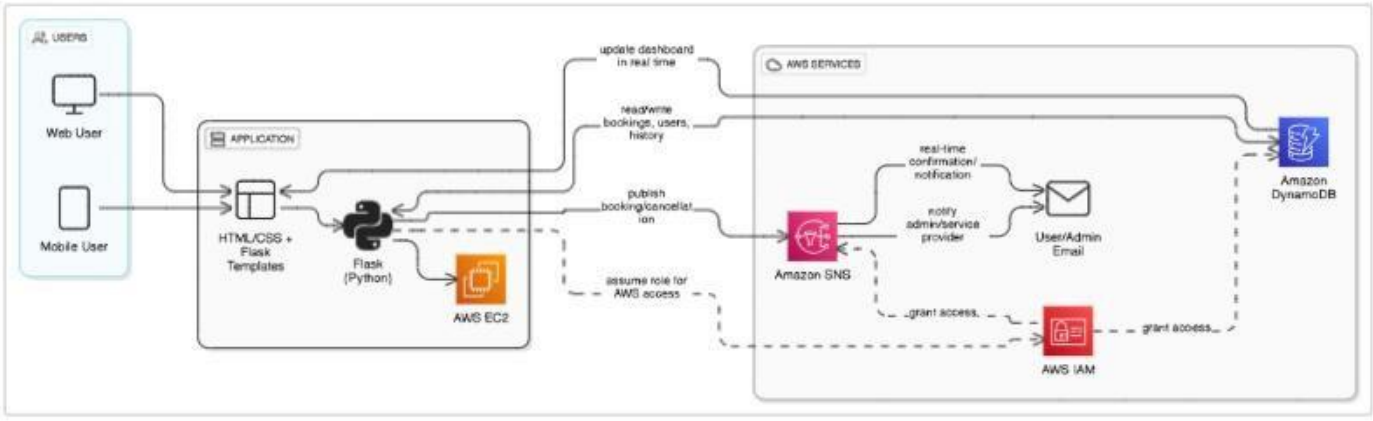
Scenario 2: Real-Time Booking Confirmation and Updates with AWS SNS

Upon successful booking, TravelGo instantly notifies the user using AWS SNS, sending real-time email confirmations with all relevant details. For example, after a student books a hotel, they immediately receive an email confirming their reservation. This notification is triggered from the Flask backend once the booking is securely recorded in DynamoDB. AWS SNS can also alert admin staff or service providers, ensuring transparency and real-time updates across all stakeholders for every transaction, from initial booking to any subsequent cancellations

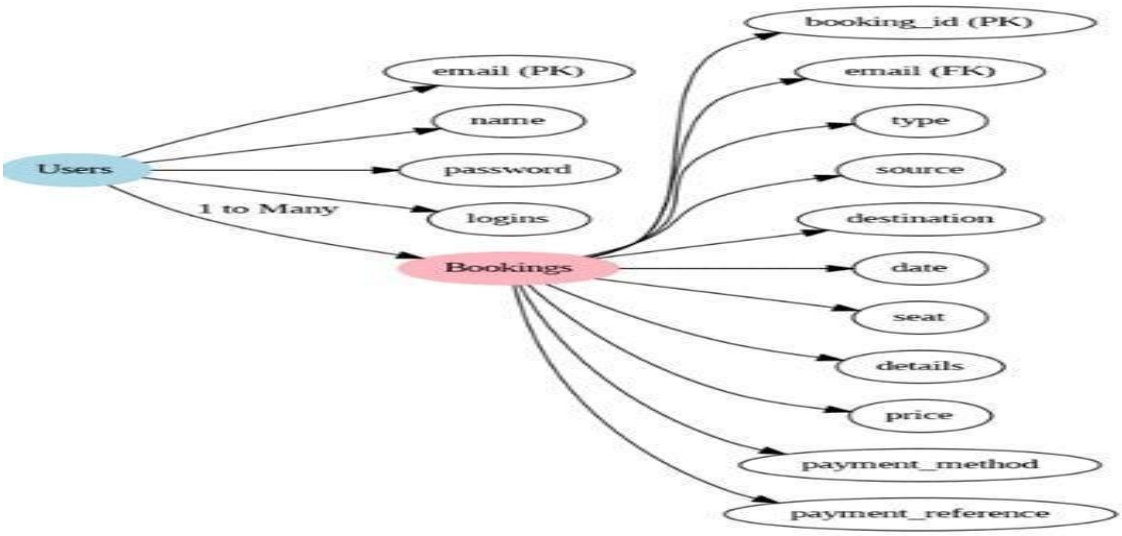
Scenario 3: Dynamic User Dashboard with Comprehensive Travel History & Management

TravelGo features a dynamic and personalized user dashboard that provides a comprehensive overview of all past and upcoming bookings. A logged-in user can easily view their flight and hotel reservations, categorized by type, along with essential details like dates and prices. Flask efficiently fetches this personalized data from AWS DynamoDB, which persistently stores all user bookings. The responsive UI, powered by HTML/CSS and Flask templates, ensures users can review, manage, and quickly cancel their bookings anytime, from any device, with real-time updates

AWS ARCHITECTURE



Entity Relationship (ER)Diagram:



Pre-requisites:

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

Project WorkFlow:

1. Backend Development and Application Setup

Activity 1.1: Develop the Backend Using Flask.

Activity 1.2: Integrate AWS Services Using boto3

2. . AWS Account Setup and Login

Activity 2.1: . AWS Account Setup and Login

Activity 2.2: Log in to the AWS Management Console.

3. DynamoDB Database Creation and Setup

Activity 3.1: Create a DynamoDB Table.

Activity 3.2: Configure Attributes for User Data and Book Requests.

4. SNS Notification Setup

Activity 4.1: Create SNS topics for book request notifications.

Activity 4.2: Subscribe users and library staff to SNS email notifications.

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

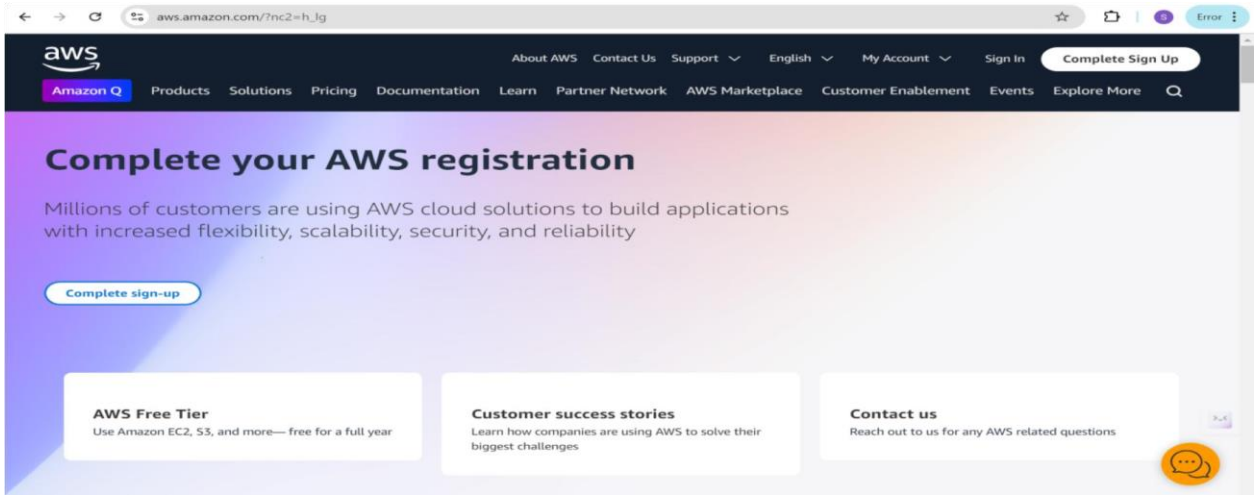
8. Testing and Deployment

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

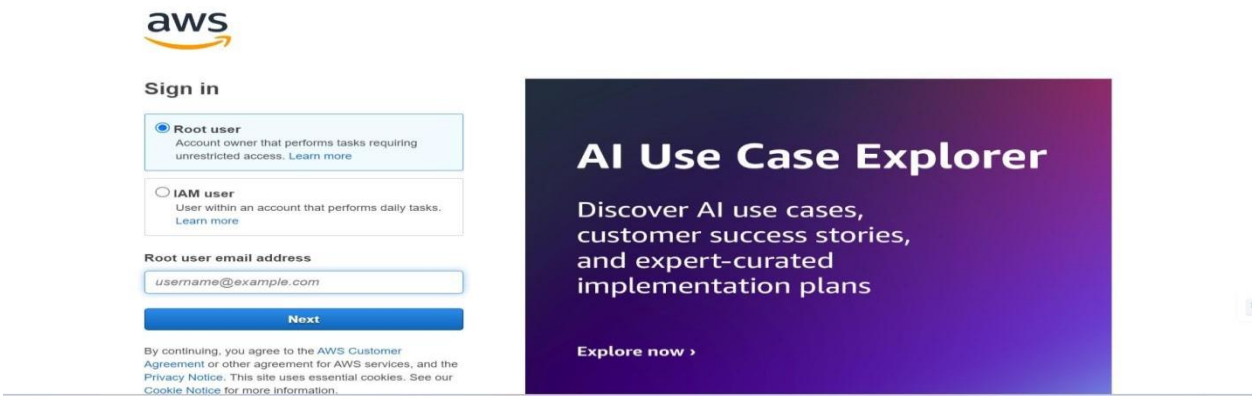
Milestone 1: AWS Account Setup and Login

This is for your understanding only, please refrain from creating an AWS account. A temporary account will be provided via Troven.

- Go to the AWS website (https://aws.amazon.com/).
- 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.



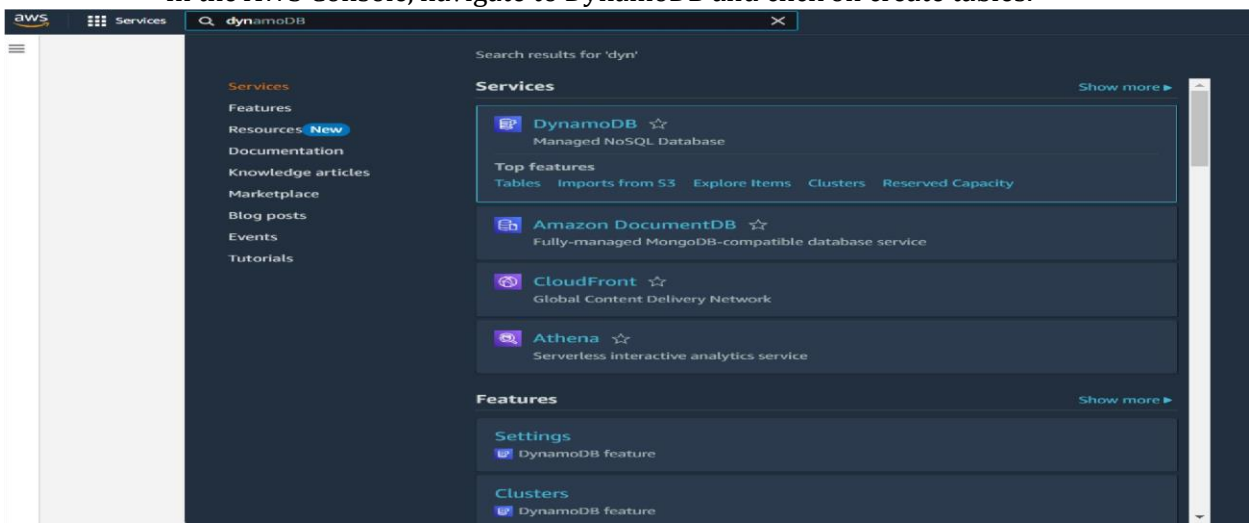
- **Activity 1: Log in to the AWS Management Console**
  - After setting up your account, log in to the AWS Management Console.

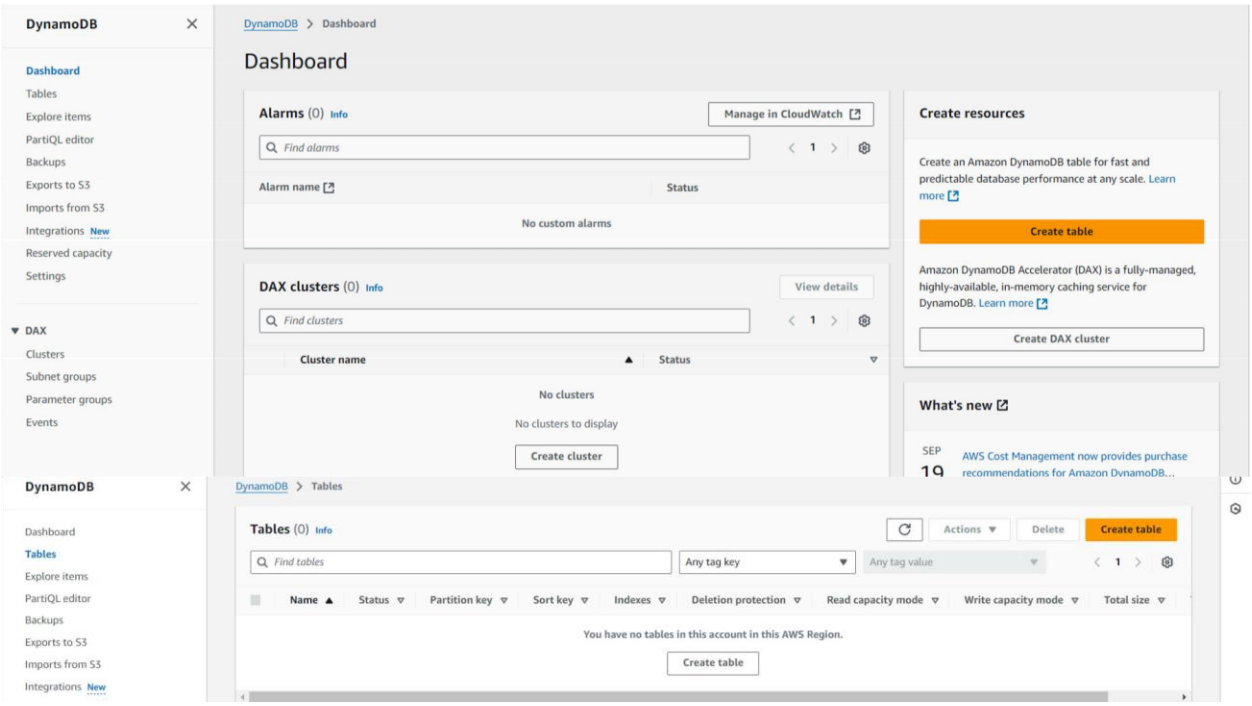


Milestone 2: DynamoDB Database Creation and Setup

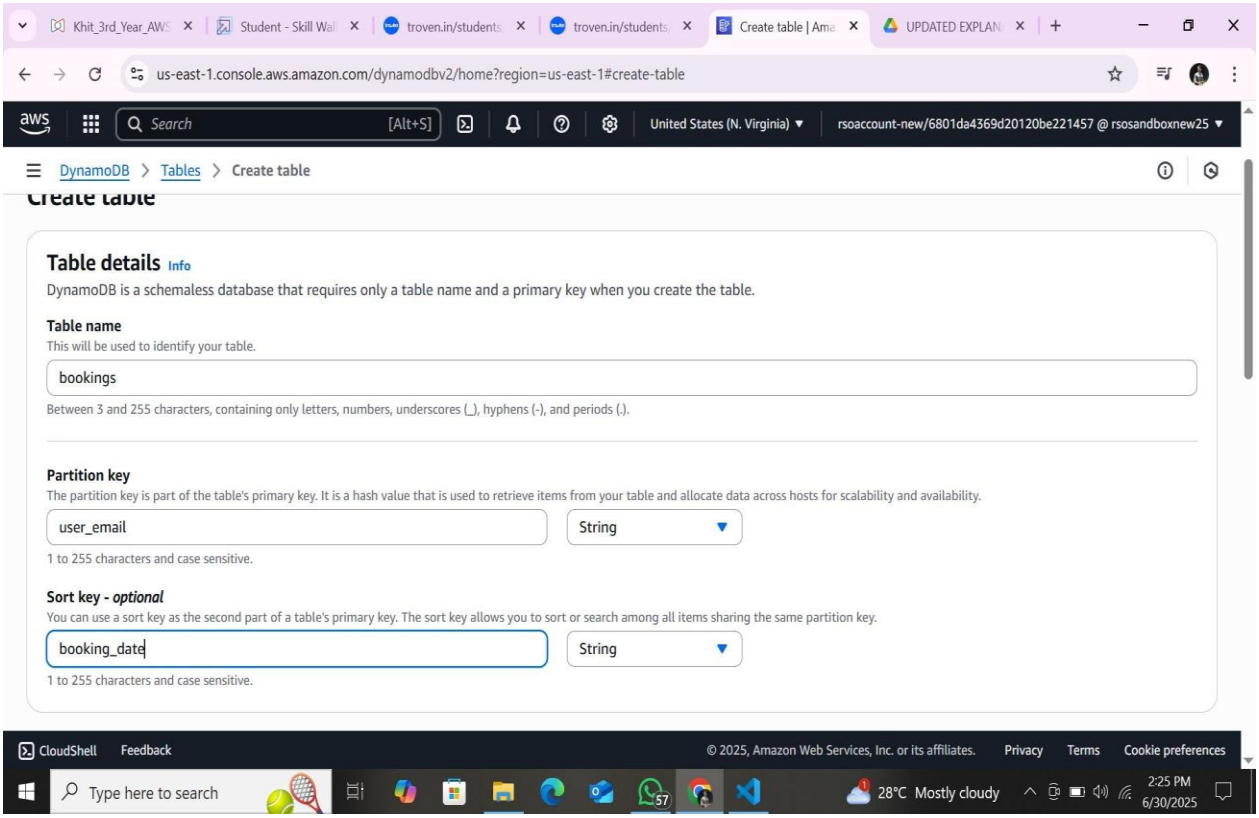
Database Creation and Setup involves initializing a cloud-based NoSQL database to store and manage application data efficiently. This step includes defining tables, setting primary keys, and configuring read/write capacities. It ensures scalable, high-performance data storage for seamless backend operations.

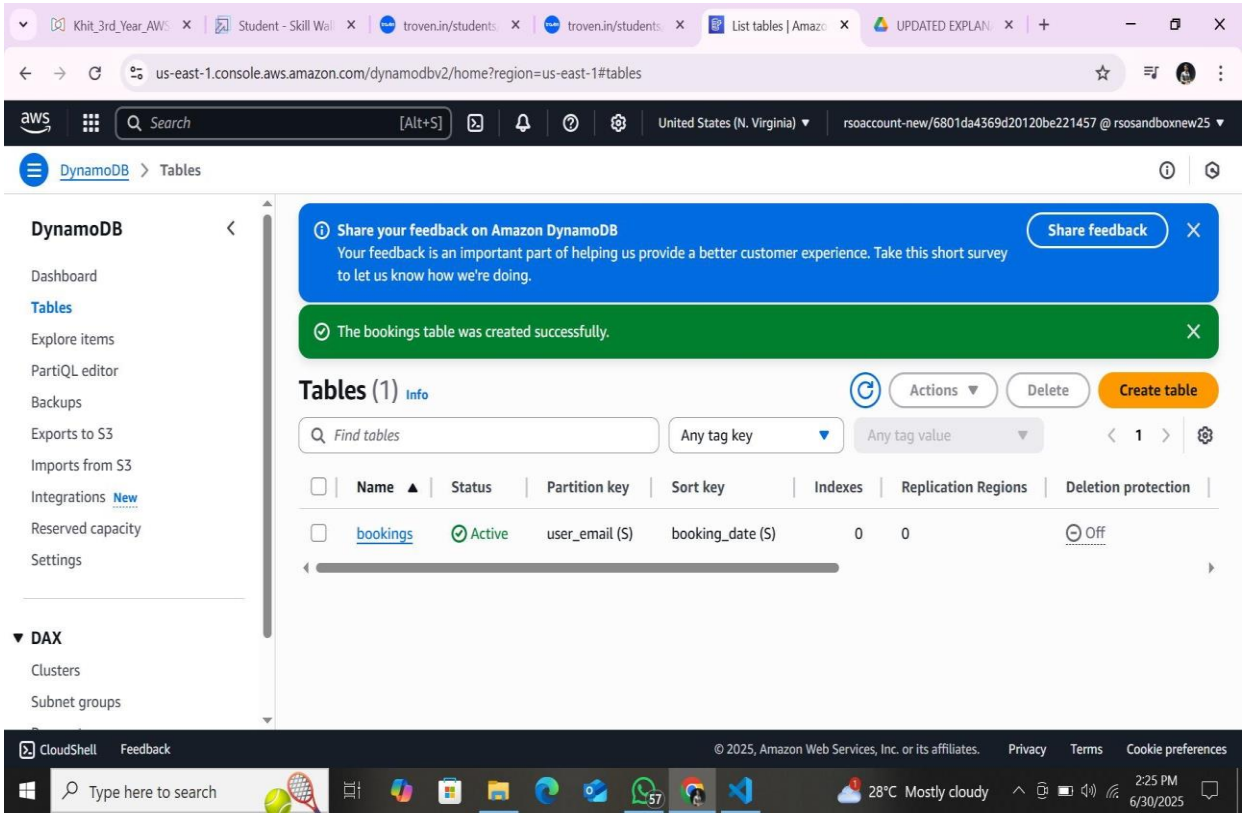
- **Activity 2.1: Navigate to the DynamoDB**
    - In the AWS Console, navigate to DynamoDB and click on create tables.

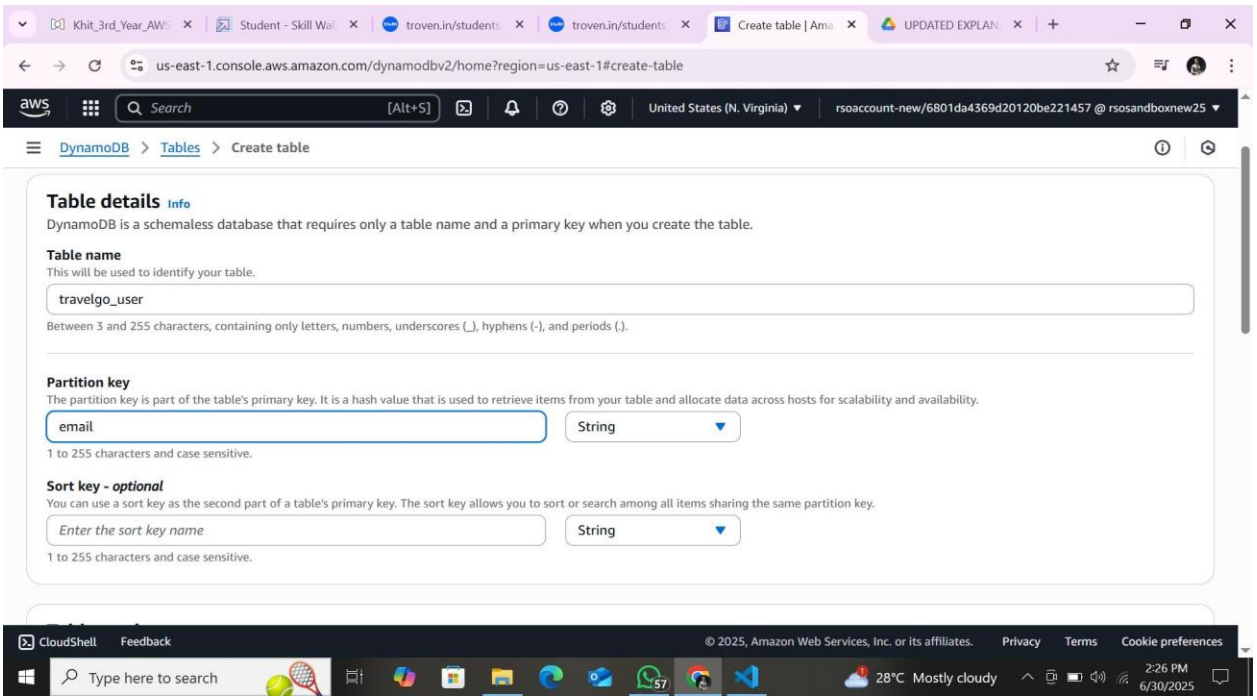


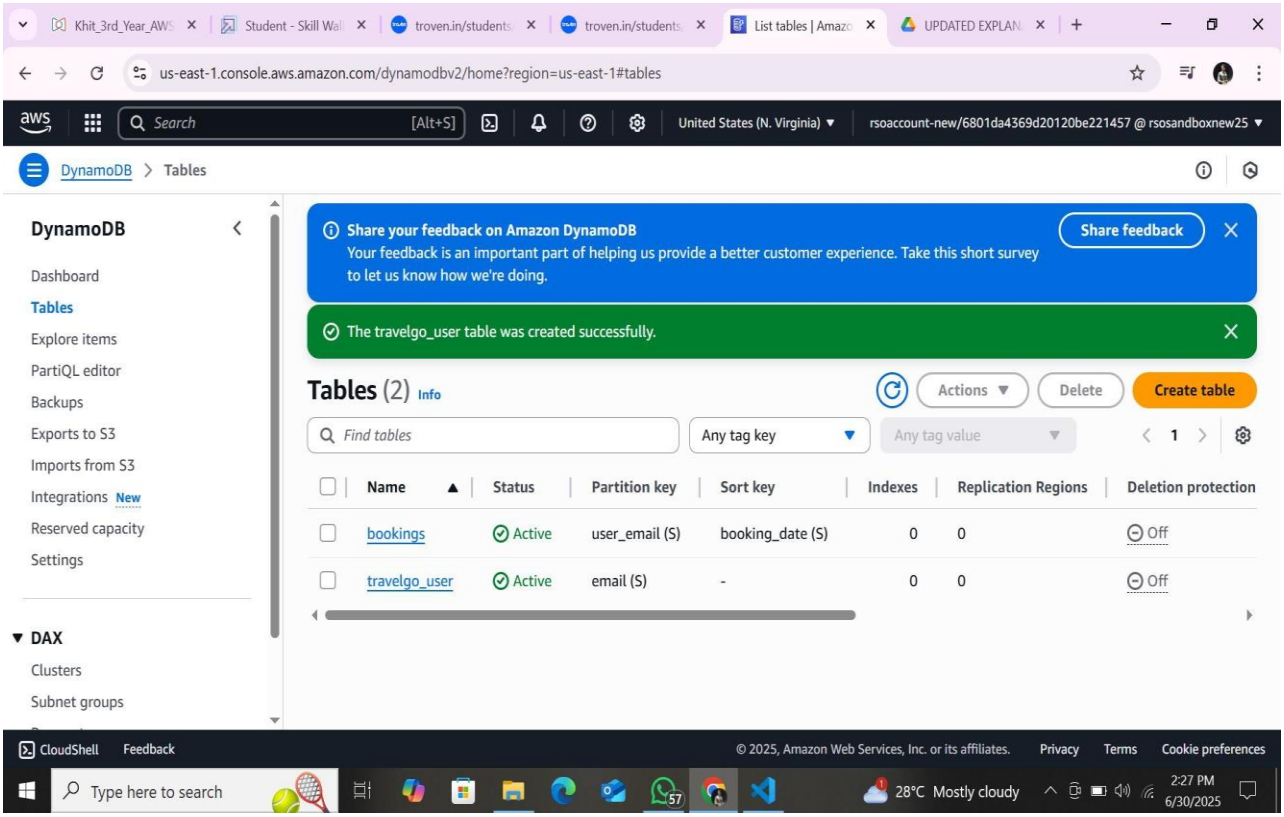


- **Activity 2.2: Create a DynamoDB table for storing registration details and book requests.**




us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#tables
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
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
The bookings table was created successfully.
Tables (1) Info
Actions
Delete
Create table
Find tables
Any tag key
Any tag value
Name
Status
Partition key
Sort key
Indexes
Replication Regions
Deletion protection
bookings
Active
user_email (S)
booking_date (S)
0
0
Off
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
28°C Mostly cloudy
2:25 PM
6/30/2025


us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#create-table
DynamoDB > Tables > 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.
travelgo_user
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.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
28°C Mostly cloudy
2:26 PM
6/30/2025

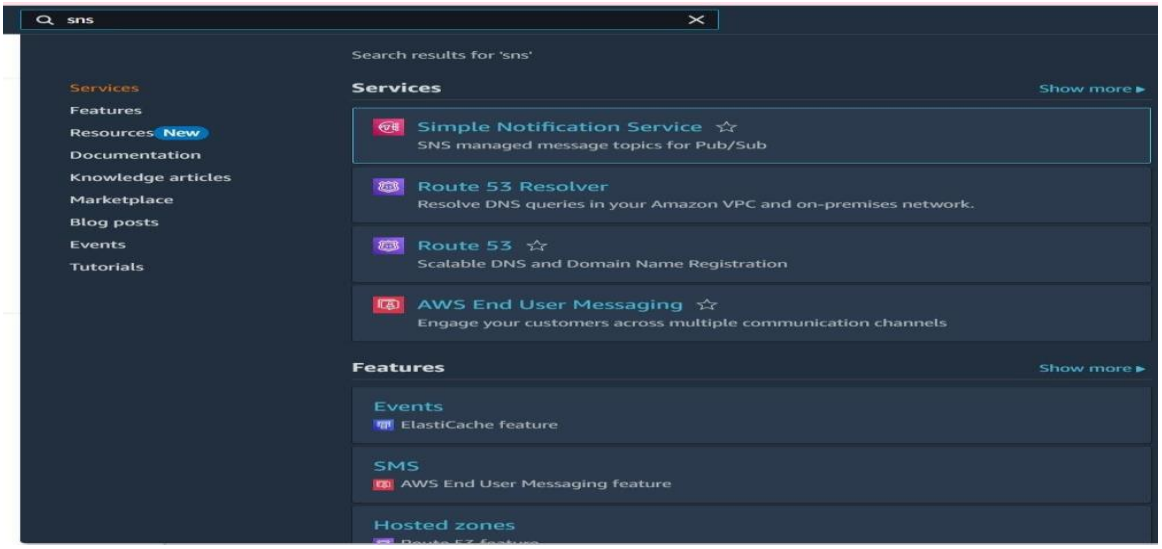


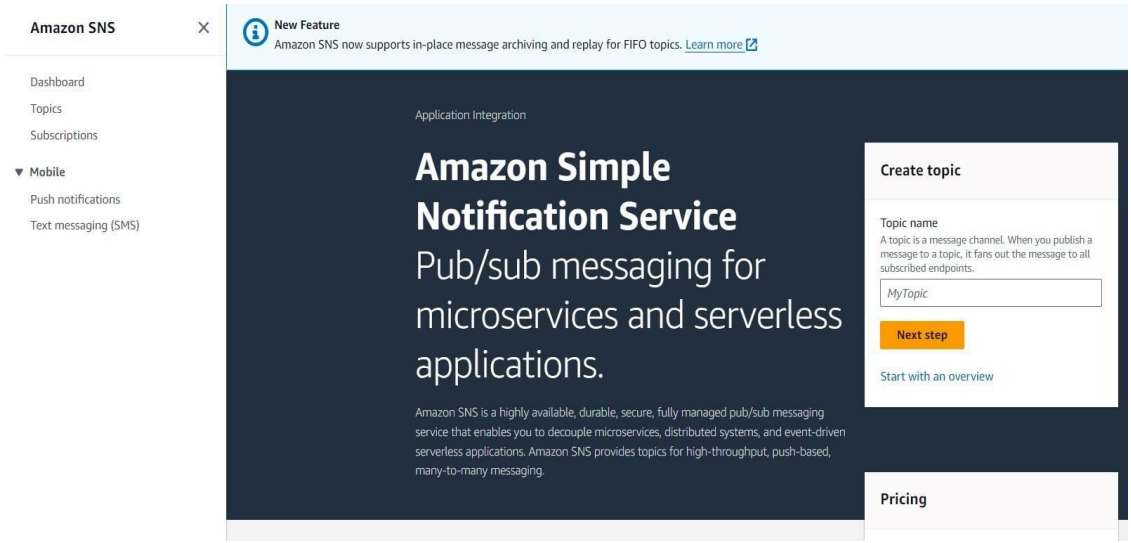
I created three dynamo Tables for my project they are bookings, trains and travelgo user .

Milestone 3: SNS Notification Setup

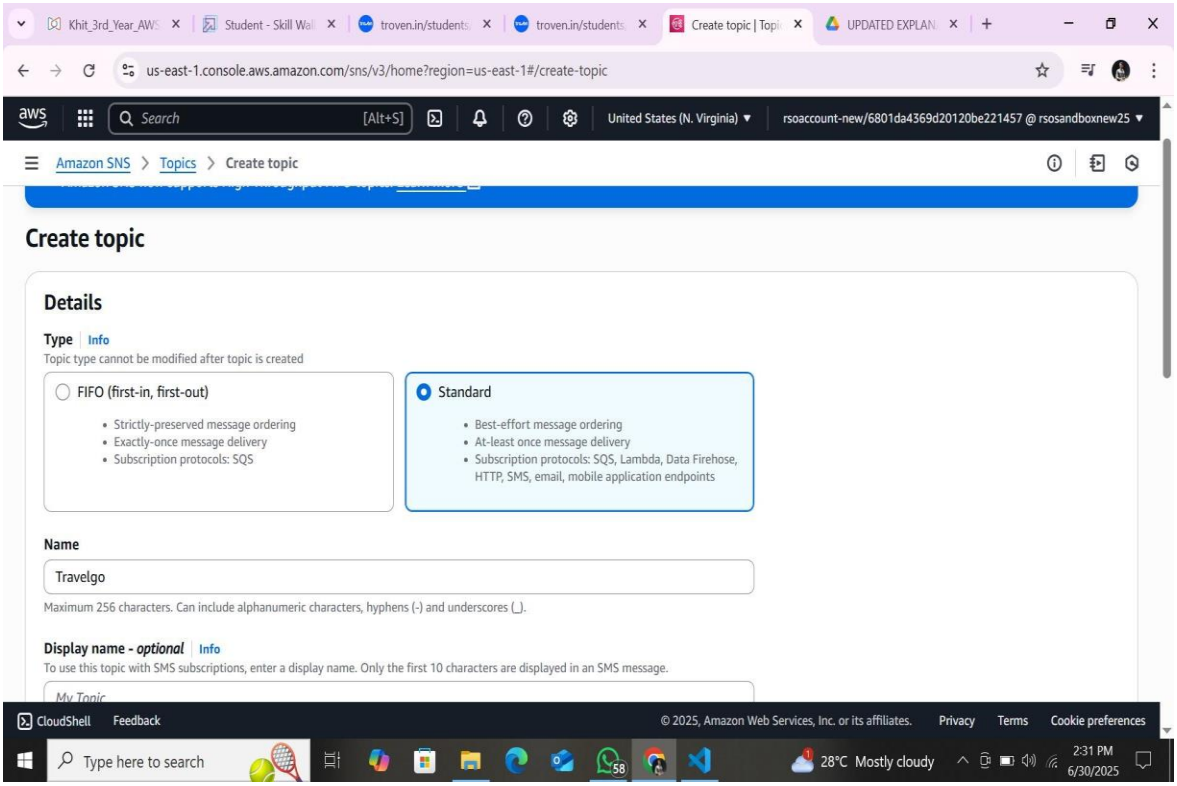
☐ Activity 3.1: Create SNS topics for sending email notifications to users and library staff.

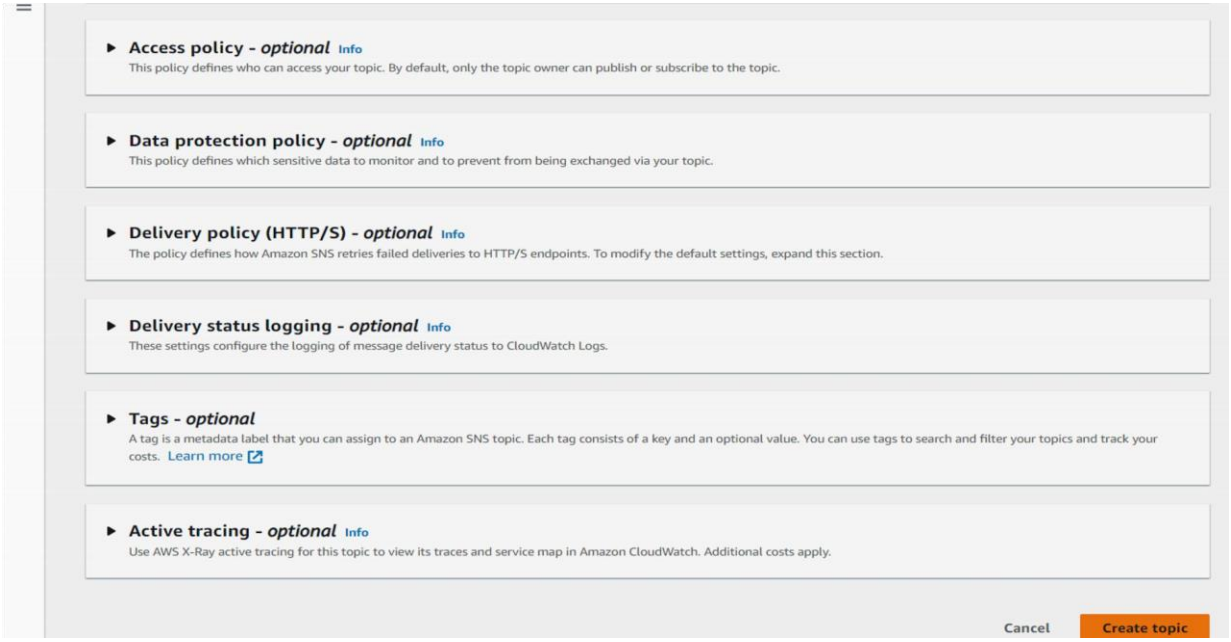
- In the AWS Console, search for SNS and navigate to the SNS Dashboard.



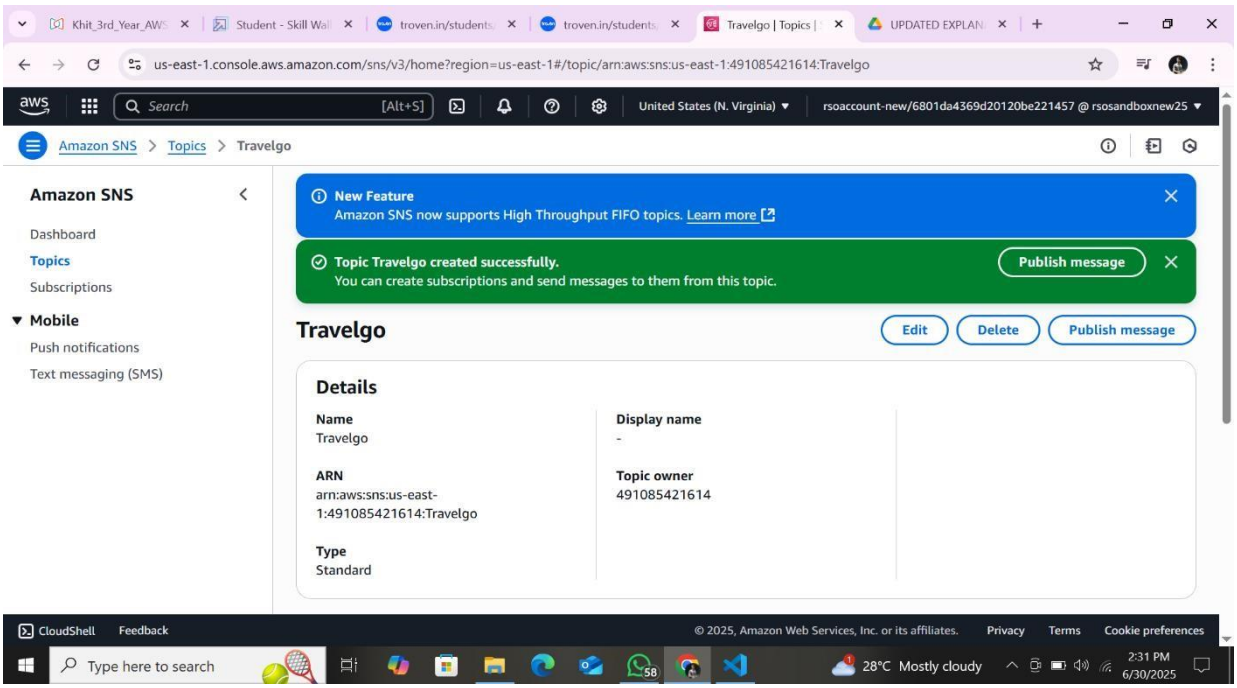


- Click on **Create Topic** and choose a name for the topic
- Choose Standard type for general notification use cases and Click on Create Topic.


us-east-1.console.aws.amazon.com/sns/v3/home?region=us-east-1#/create-topic
Search
[Alt+S]
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew25
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
Travelgo
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
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
28°C Mostly cloudy
2:31 PM
6/30/2025


Access policy - optional Info
This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.
Data protection policy - optional Info
This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.
Delivery policy (HTTP/S) - optional Info
The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.
Delivery status logging - optional Info
These settings configure the logging of message delivery status to CloudWatch Logs.
Tags - optional
A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. Learn more
Active tracing - optional Info
Use AWS X-Ray active tracing for this topic to view its traces and service map in Amazon CloudWatch. Additional costs apply.
Cancel
Create topic

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

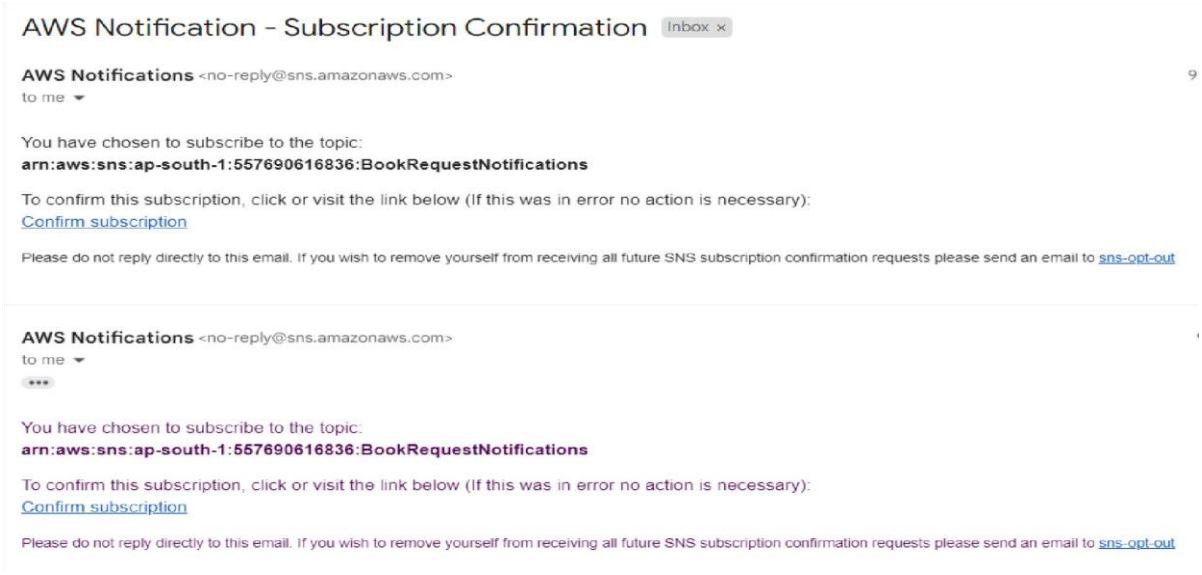


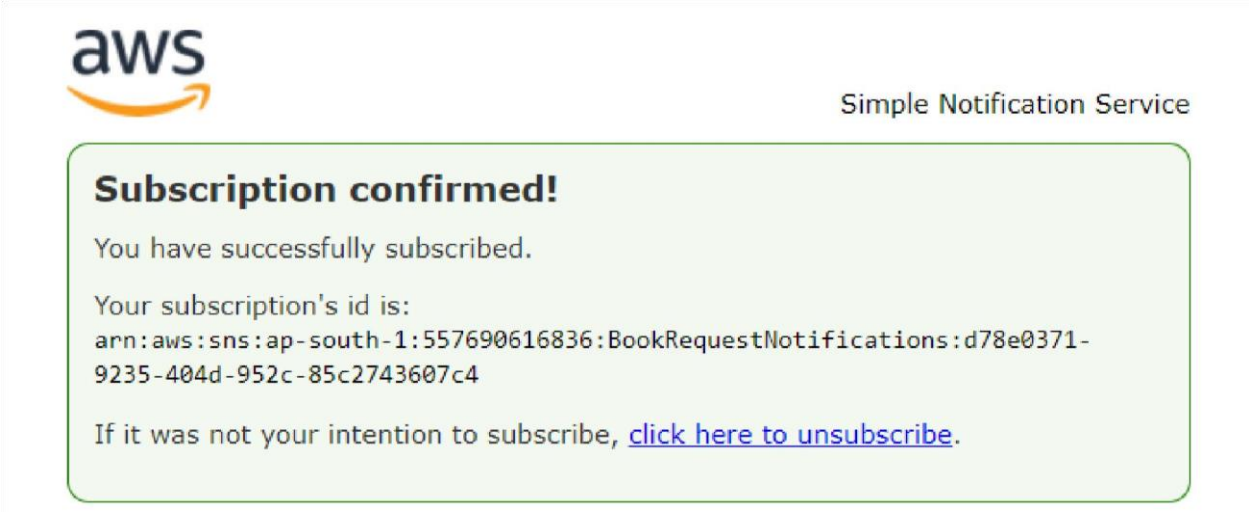
- **Activity 3.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.**

  - Subscribe users (or admin staff) to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails.

  - After subscription request for the mail confirmation

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

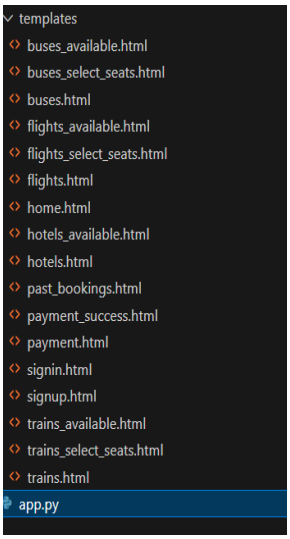




- Successfully done with the SNS mail subscription and setup, now store the ARN link.

Milestone 4:Backend Development and Application Setup

- **Activity 4.1: Develop the backend using Flask**
  - File Explorer Structure



Description: set up the TRAVEL GO project with an app.py file, a static/ folder for assets, and a templates/ directory containing all required HTML pages like home, login, register, subject-specific pages (e.g. buse.html, hotel.html, train.html, flight.html).

Description of the code :

- **Flask App Initialization**

```
from flask import Flask, render_template, request, redirect, url_for, session, flash
import random
import string
import datetime
import boto3
from botocore.exceptions import ClientError
```

Description: import essential libraries including Flask utilities for routing, Boto3 for DynamoDB operations, SMTP and email modules for sending mails, and Bcrypt for password hashing and verification

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

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

- Dynamodb Setup:

```
# AWS Configuration
AWS_REGION = 'us-west-2'  # Update to your desired region
DYNAMODB_TABLE_USERS = 'Users'
DYNAMODB_TABLE_BOOKINGS = 'Bookings'
SNS_TOPIC_ARN = 'arn:aws:sns:your-region:your-account-id:your-sns-topic'  # Update with your ARN
```

Description: Initialize the DynamoDB resource for the us-east-1 region and set up access to the Users and Requests tables for storing user details and book requests.

- **SNS Connection**

```
def send_notification(message):
    try:
        response = sns_client.publish(
            TopicArn=SNS_TOPIC_ARN,
            Message=message,
            Subject='Booking Notification'
        )
        return response
    except ClientError as e:
        print(f"Error sending notification: {e}")
        return None
```

Description: Configure **SNS** to send notifications when a book request is submitted. Paste your stored ARN link in the sns_topic_arn space, along with the region name where the SNS topic is created. Also, specify the chosen email service in SMTP_SERVER (e.g., Gmail, Yahoo, etc.) and enter the subscribed email in the SENDER_EMAIL section. Create an 'App password' for the email ID and store it in the SENDER_PASSWORD section. ● **Routes for Web Pages**

- **Home Route:**

```
@app.route('/home')
def home():
    if 'username' not in session:
        return redirect(url_for('signin'))
    return render_template('home.html', username=session['username'])
```

Description: define the home route / to automatically redirect users to the register page when they access the base URL.

- **Register Route:**

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

        if get_user(username):
            flash('Username already exists', 'error')
        elif add_user(username, password):
            flash('Account created! Please sign in', 'success')
            return redirect(url_for('signin'))
        else:
            flash('Error creating account', 'error')

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

Description: define /register route to validate registration form fields, hash the user password using Bcrypt, store the new user in DynamoDB with a login count, and send an SNS notification on successful registration

- **login Route (GET/POST)**:

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

        user = get_user(username)
        if user and user['password'] == password:
            session['username'] = username
            flash('Login successful!', 'success')
            return redirect(url_for('home'))
        else:
            flash('Invalid credentials', 'error')

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

Description: define /login route to validate user credentials against DynamoDB, check the password using Bcrypt, update the login count on successful authentication, and redirect users to the home page

- **Train , Bus and Hotel routes:**

```
mock_travel_options = {
    'flights': [
        {'id': 'flight1', 'flight_number': 'AC101', 'airline': 'AirConnect',
         'departure_time': '06:00 AM', 'arrival_time': '08:00 AM', 'price': 150.00, 'duration': '2h 00m'},
        {'id': 'flight2', 'flight_number': 'SH202', 'airline': 'SkyHigh',
         'departure_time': '09:30 AM', 'arrival_time': '11:30 AM', 'price': 180.00, 'duration': '2h 00m'},
    ],
    'buses': [
        {'id': 'bus1', 'name': 'Express Lines', 'departure_time': '08:00 AM',
         'arrival_time': '12:00 PM', 'price': 25.00, 'total_seats': 20},
    ],
    'trains': [
        {'id': 'train1', 'number': 'REX001', 'name': 'Rail Express',
         'departure_time': '07:00 AM', 'arrival_time': '01:00 PM', 'price': 60.00},
    ],
    'hotels': [
        {'id': 'hotel1', 'name': 'Grand Plaza Hotel', 'location': 'New York',
         'price_per_night': 120.00, 'rating': 4.5},
    ]
```

Description: define /home-page to render the main homepage, /ebook-buttons to handle subject selection and redirection, and /<subject>.html dynamic route to render subjectspecific pages like Mathematics or English.

- **Request Routes:**

```python
@app.route('/book/<transport_type>', methods=['GET', 'POST'])
def book_transport(transport_type):
    if 'username' not in session:
        return redirect(url_for('signin'))

    if request.method == 'POST':
        session['current_booking'] = {
            'type': transport_type,
            'source': request.form.get('source'),
            'destination': request.form.get('destination'),
            'date': request.form.get('date'),
        }
        return redirect(url_for(f'{transport_type}_available'))

    return render_template(f'{transport_type}.html')

@app.route('/book/<transport_type>/available')
def transport_available(transport_type):
    if 'username' not in session:
        return redirect(url_for('signin'))

    booking = session.get('current_booking')
    if not booking or booking['type'] != transport_type:
        return redirect(url_for(f'book_{transport_type}'))

    options = mock_travel_options.get(transport_type, [])
    return render_template(f'{transport_type}_available.html',
                           options=options,
                           booking=booking)

@app.route('/payment', methods=['GET', 'POST'])
def payment():
    if 'username' not in session:
        return redirect(url_for('signin'))

    booking = session.get('current_booking')
    if not booking:
        return redirect(url_for('home'))

    if request.method == 'POST':
        booking_id = generate_unique_id()
        booking_details = {
            'booking_id': booking_id,
            'username': session['username'],
            'type': booking['type'],
            'status': 'confirmed',
            'created_at': datetime.datetime.now().isoformat(),
            'details': booking
        }

        if add_booking(booking_details):
            send_notification(
                f"New booking: {booking_id}\n"
                f"Type: {booking['type']}\n"
                f"User: {session['username']}"
```

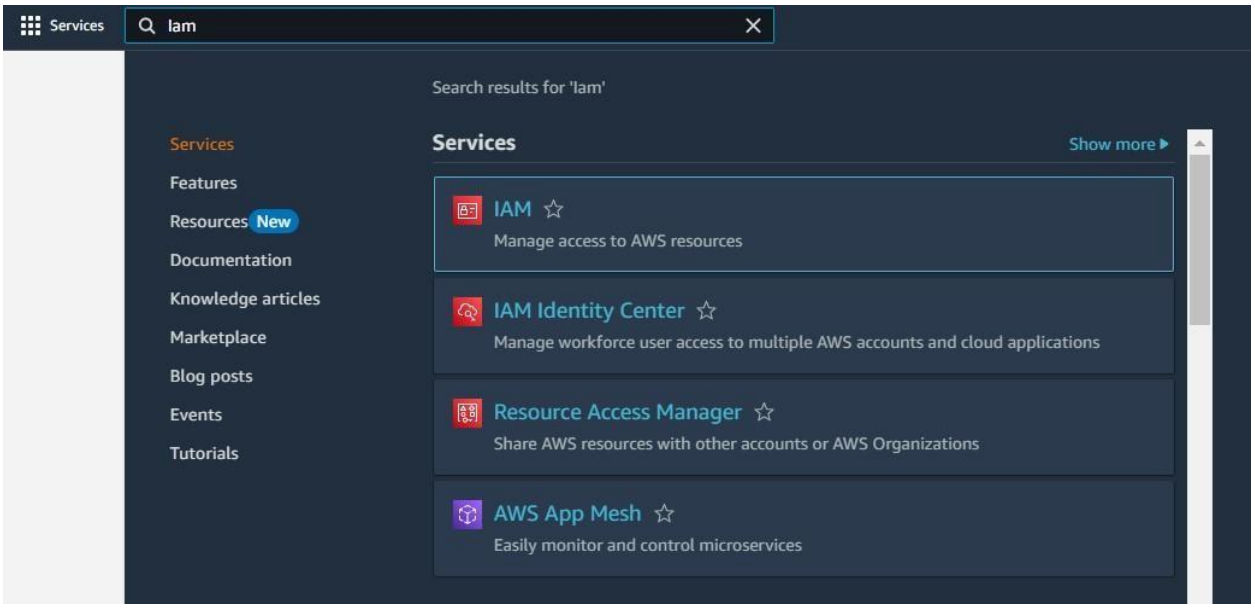
Deployment Code:

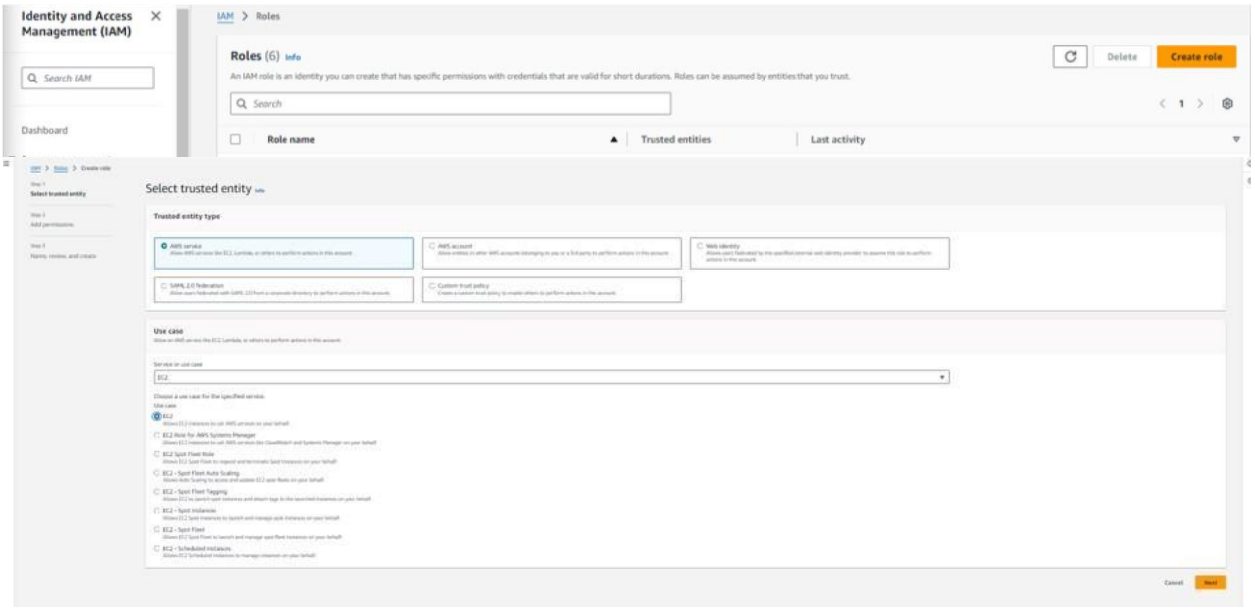
```python
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

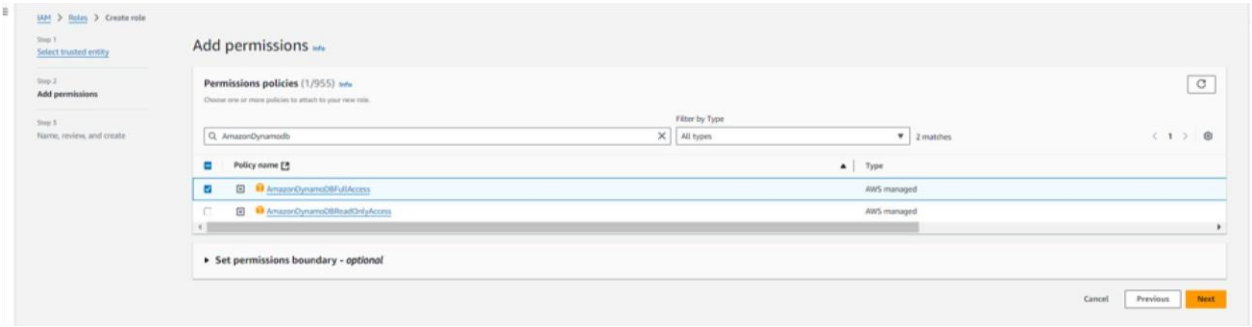
Description: start the Flask server to listen on all network interfaces (0.0.0.0) at port 80 with debug mode enabled for development and testing.

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.



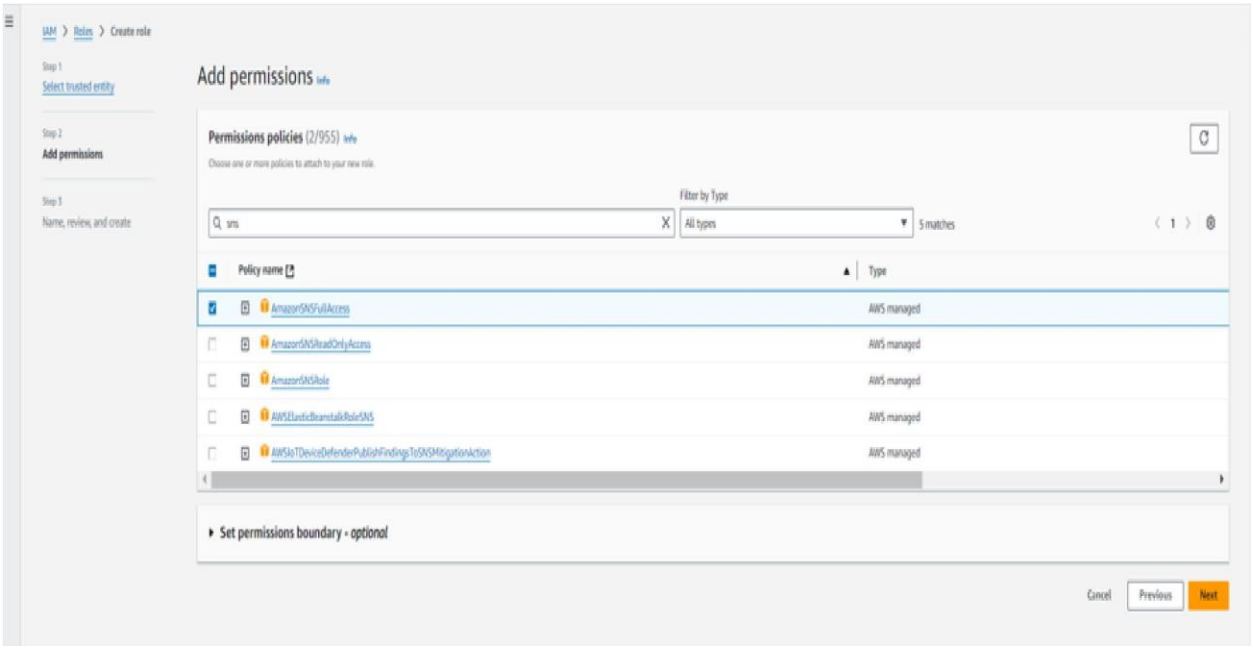


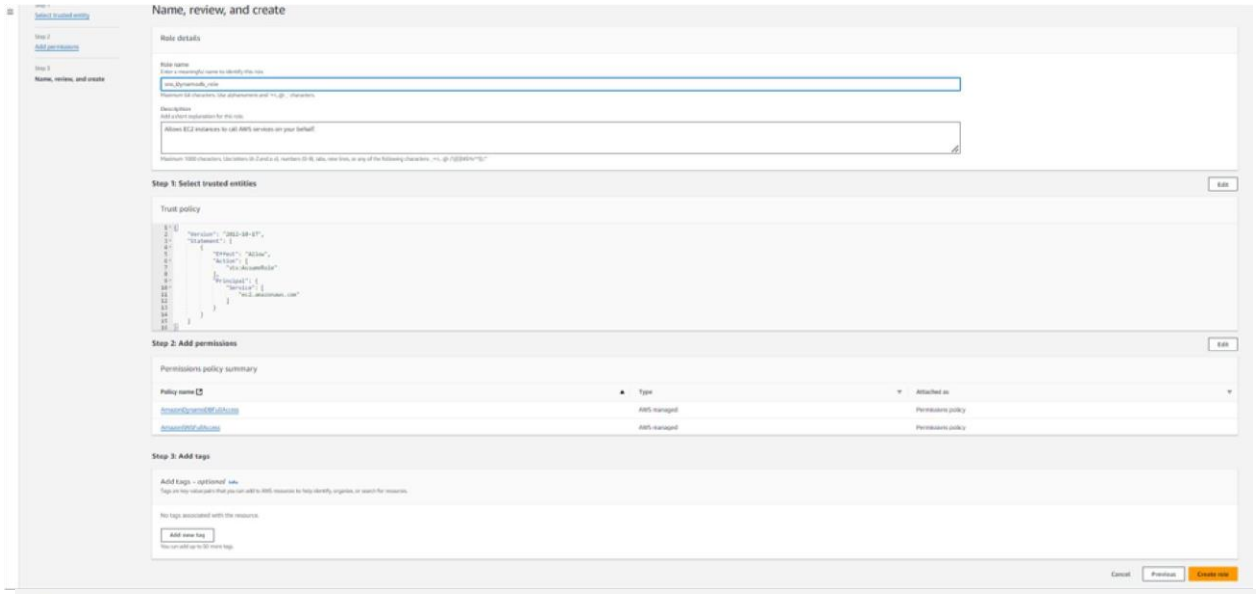


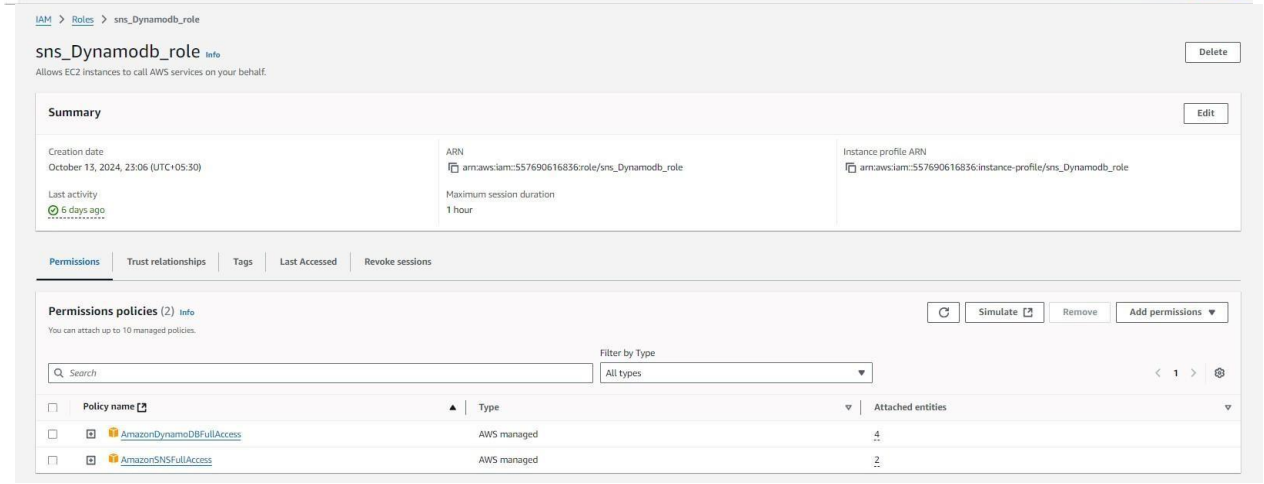
- **Activity 5.2: Attach Policies.**

  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.

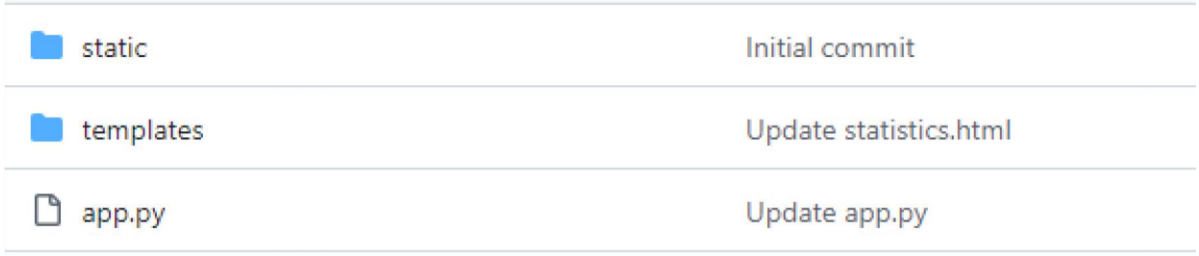


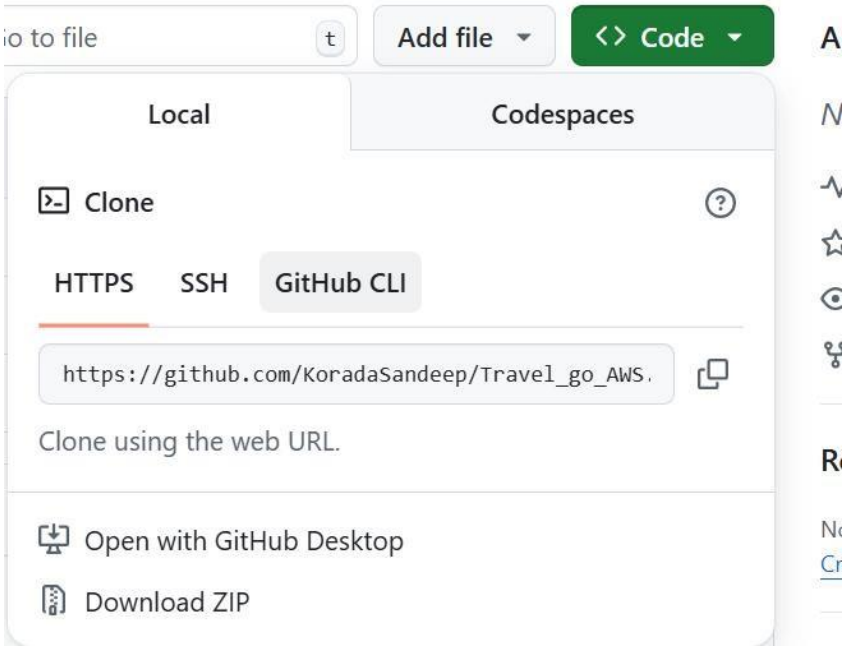

Name, review, and create
Role details
Step 1: Select trusted entities
Trust policy
Step 2: Add permissions
Permissions policy summary
Step 3: Add tags
Add tags - optional
Cancel
Previous
Create role


IAM > Roles > sns_Dynamodb_role
sns_Dynamodb_role Info
Allows EC2 instances to call AWS services on your behalf.
Delete
Summary
Edit
Creation date
October 13, 2024, 23:06 (UTC+05:30)
ARN
arn:aws:iam::557690616836:role/sns_Dynamodb_role
Instance profile ARN
arn:aws:iam::557690616836:instance-profile/sns_Dynamodb_role
Last activity
6 days ago
Maximum session duration
1 hour
Permissions
Trust relationships
Tags
Last Accessed
Revoke sessions
Permissions policies (2) Info
You can attach up to 10 managed policies.
Simulate
Remove
Add permissions
Search
Filter by Type
All types
Policy name
Type
Attached entities
AmazonDynamoDBFullAccess
AWS managed
4
AmazonSNSFullAccess
AWS managed
2

Milestone 6: EC2 Instance Setup

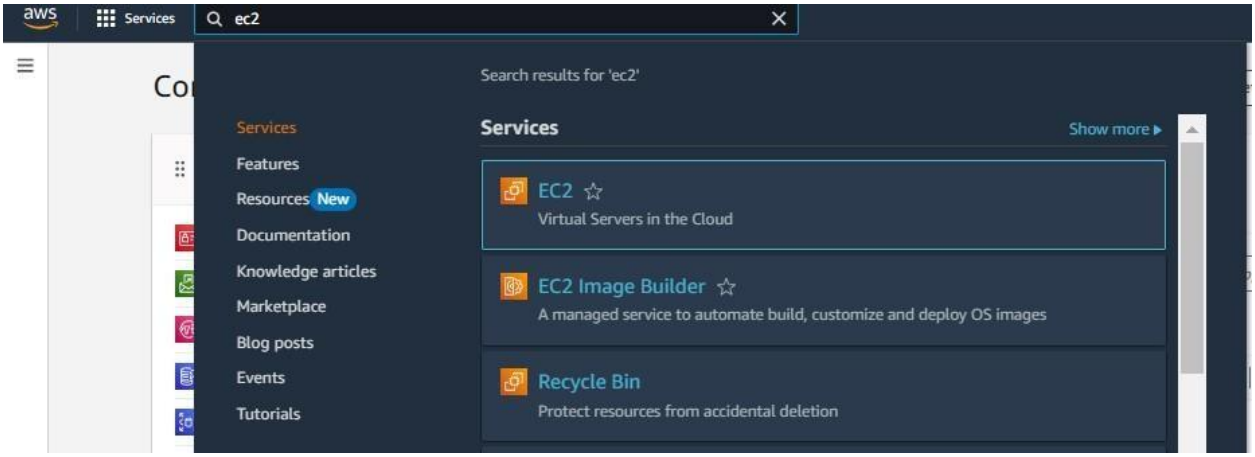
- **Note: Load your Flask app and Html files into GitHub repository.**



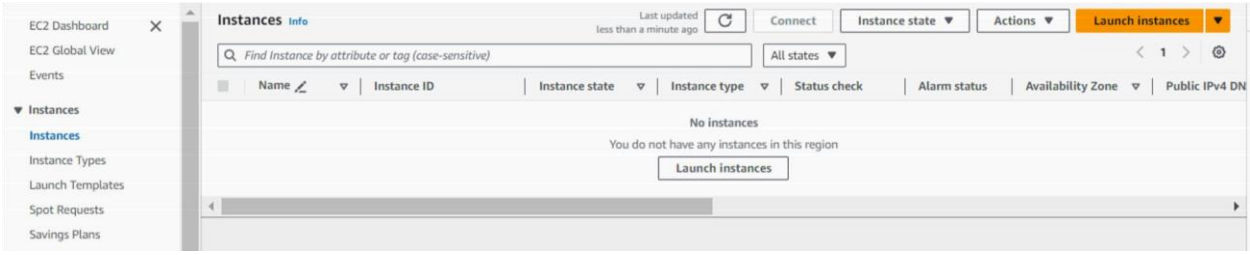


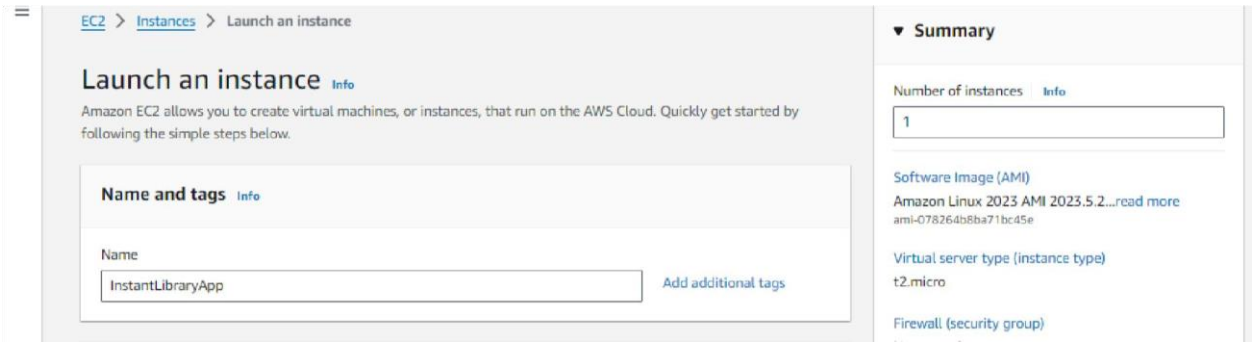
- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

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

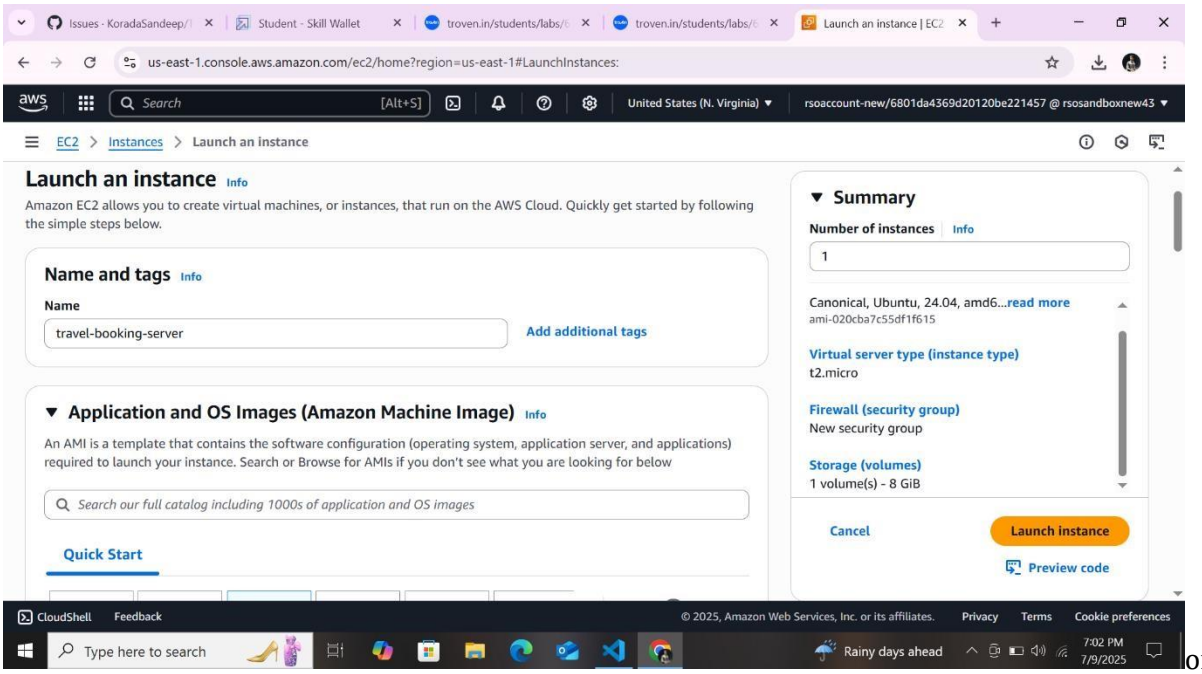


- Click on Launch instance to launch EC2 instance

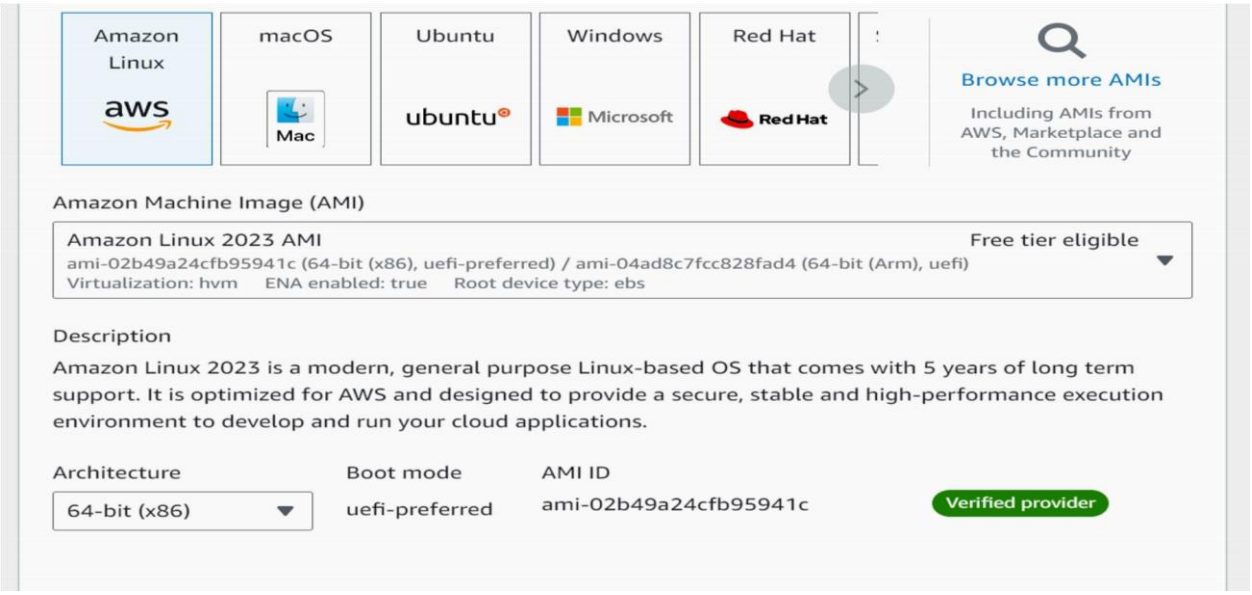




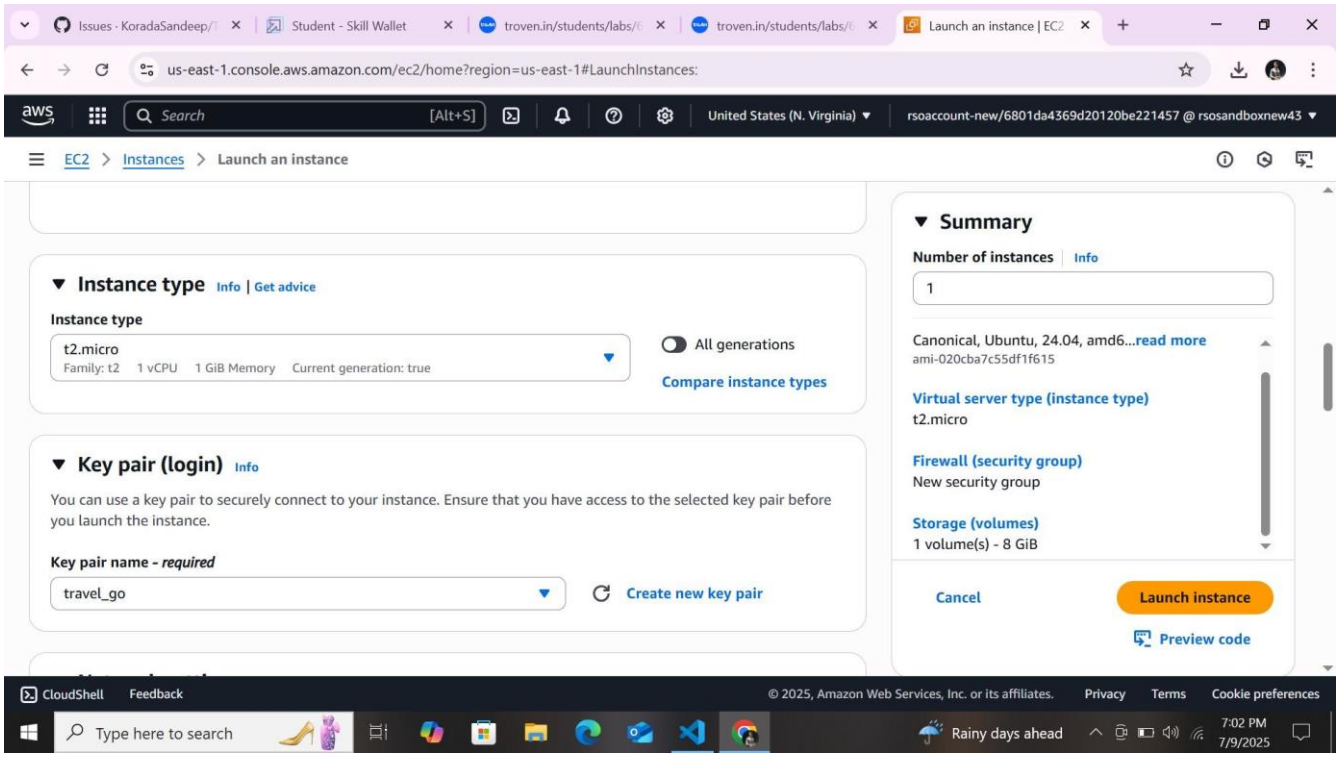
- Choose Amazon Linux 2

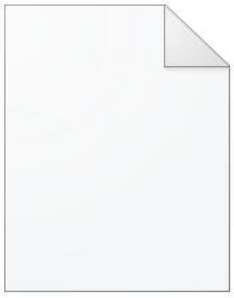


or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).



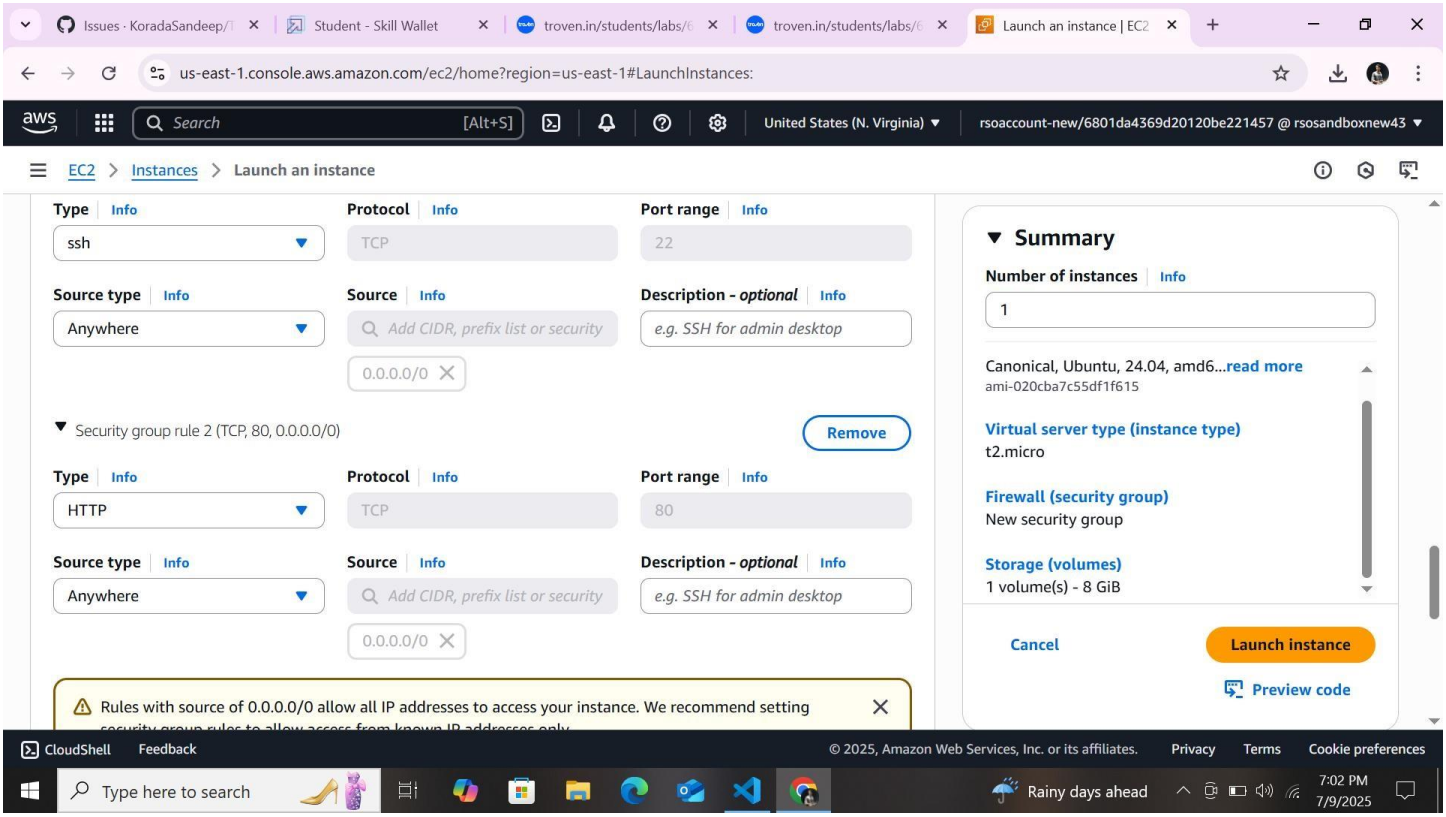
- Create and download the key pair for Server access.

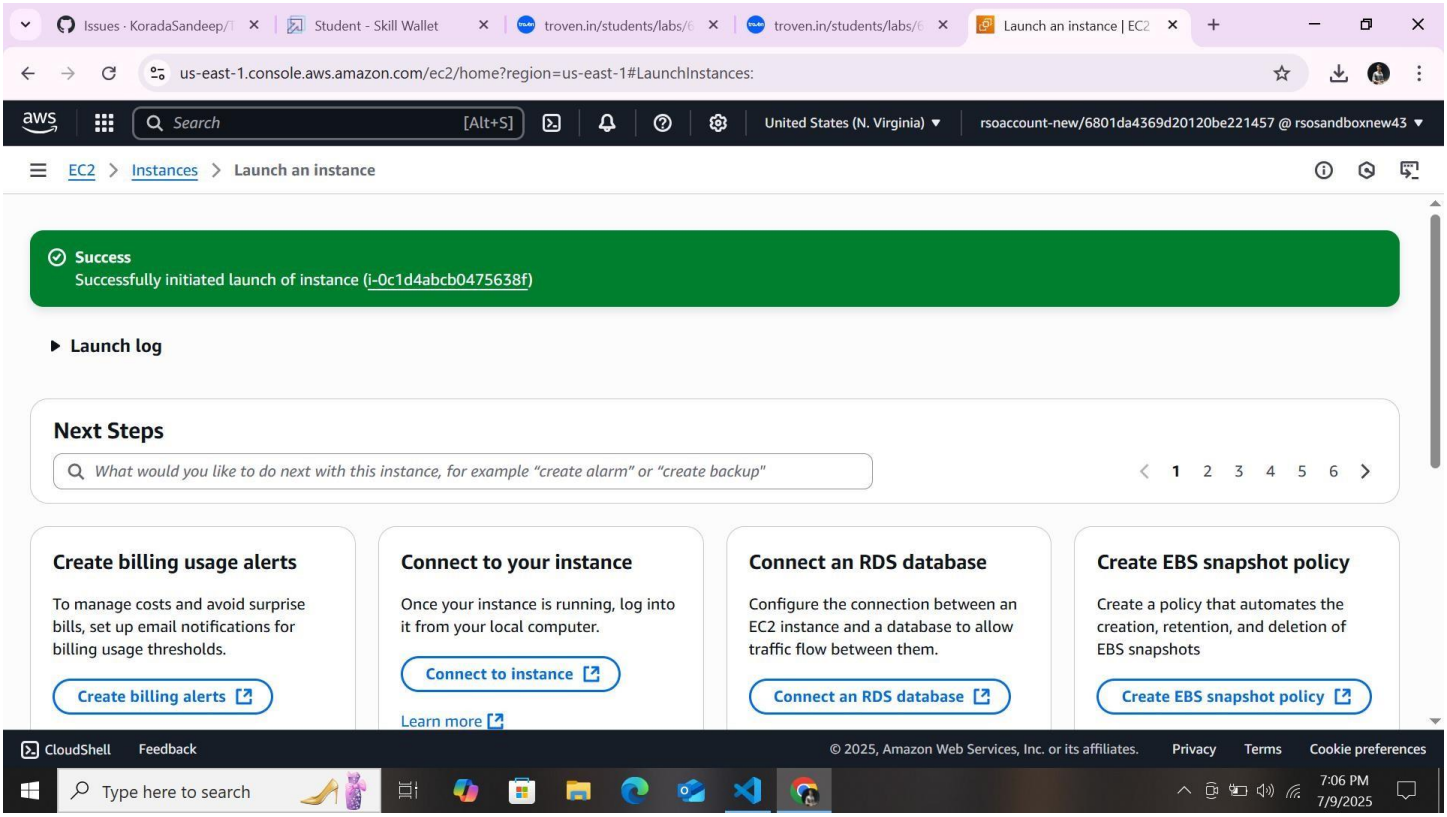




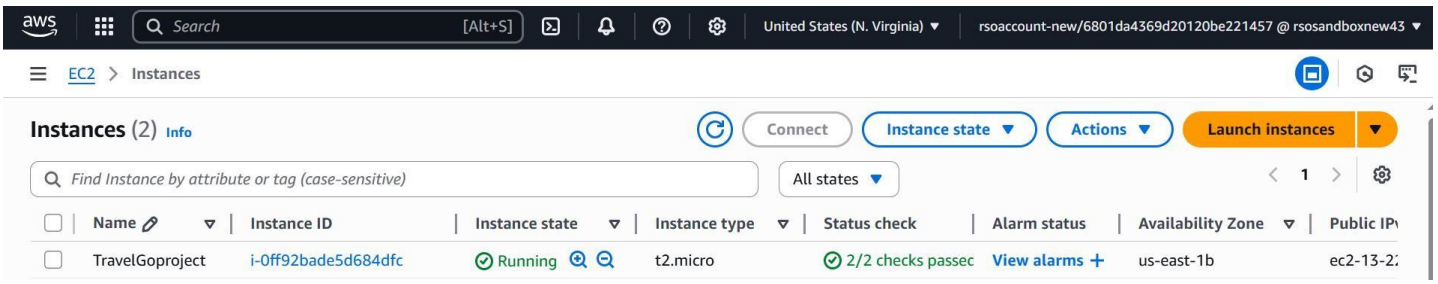
travel_go.pem

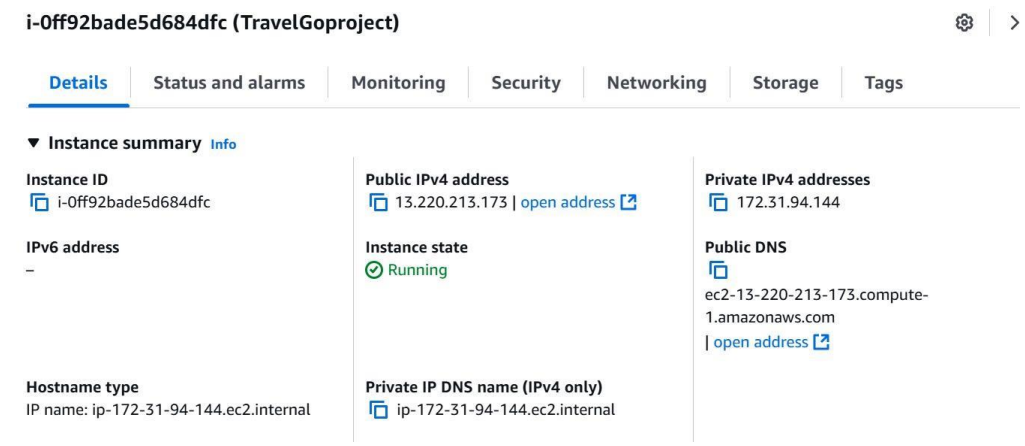
- **Activity 6.2:Configure security groups for HTTP, and SSH access.**

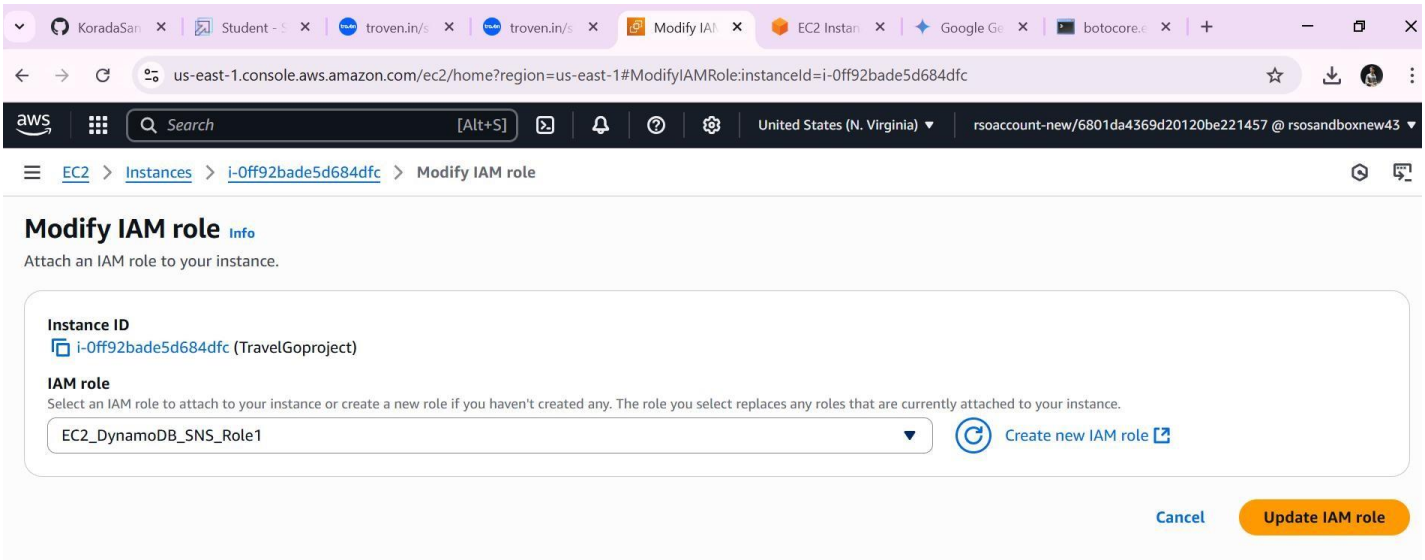

Launch an instance | EC2
us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#LaunchInstances:
Search [Alt+S]
United States (N. Virginia)
rsoaccount-new/6801da4369d20120be221457 @ rsosandboxnew43
EC2 > Instances > Launch an instance
Type Info
ssh
Protocol Info
TCP
Port range Info
22
Source type Info
Anywhere
Source Info
Add CIDR, prefix list or security
0.0.0.0/0
Description - optional Info
e.g. SSH for admin desktop
Security group rule 2 (TCP, 80, 0.0.0.0/0)
Remove
Type Info
HTTP
Protocol Info
TCP
Port range Info
80
Source type Info
Anywhere
Source Info
Add CIDR, prefix list or security
0.0.0.0/0
Description - optional Info
e.g. SSH for admin desktop
Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting
Summary
Number of instances Info
1
Canonical, Ubuntu, 24.04, amd6...read more
ami-020cba7c55df1f615
Virtual server type (instance type)
t2.micro
Firewall (security group)
New security group
Storage (volumes)
1 volume(s) - 8 GiB
Cancel
Launch instance
Preview code
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
Rainy days ahead
7:02 PM
7/9/2025



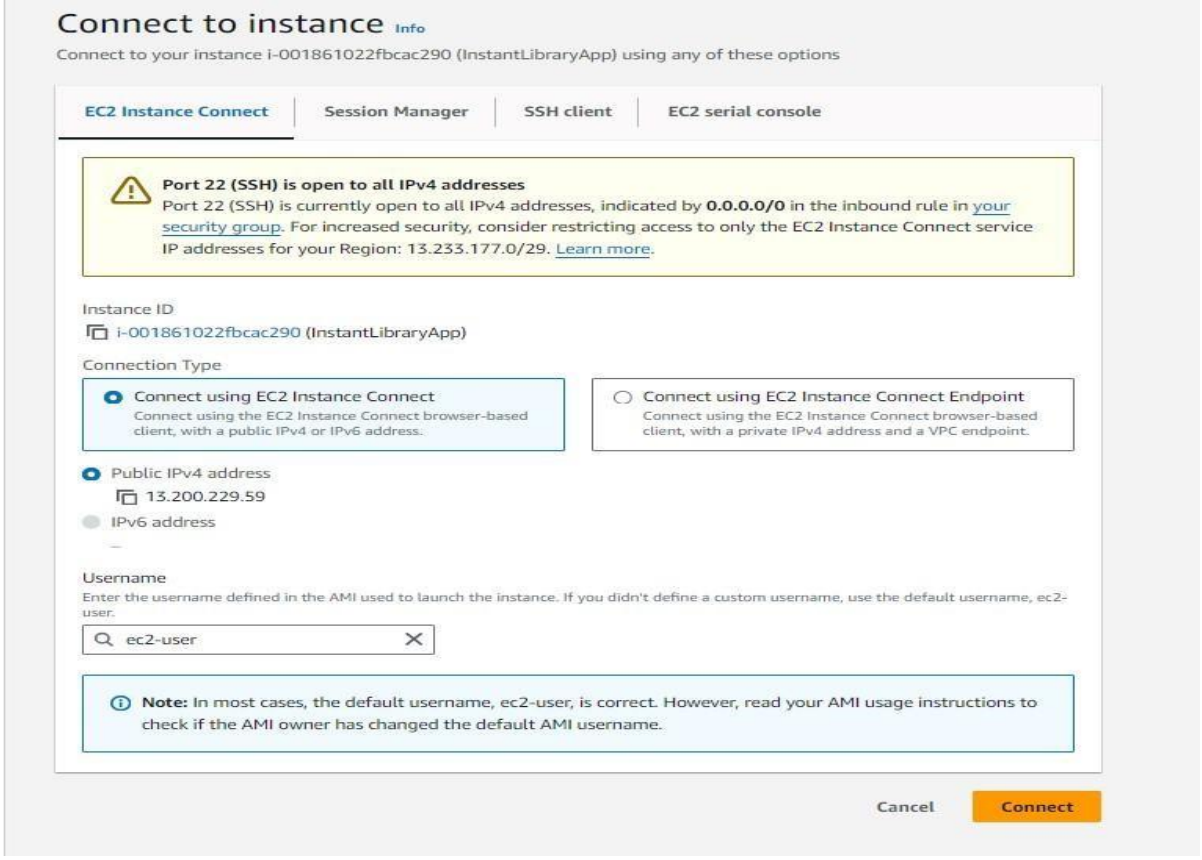
- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the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



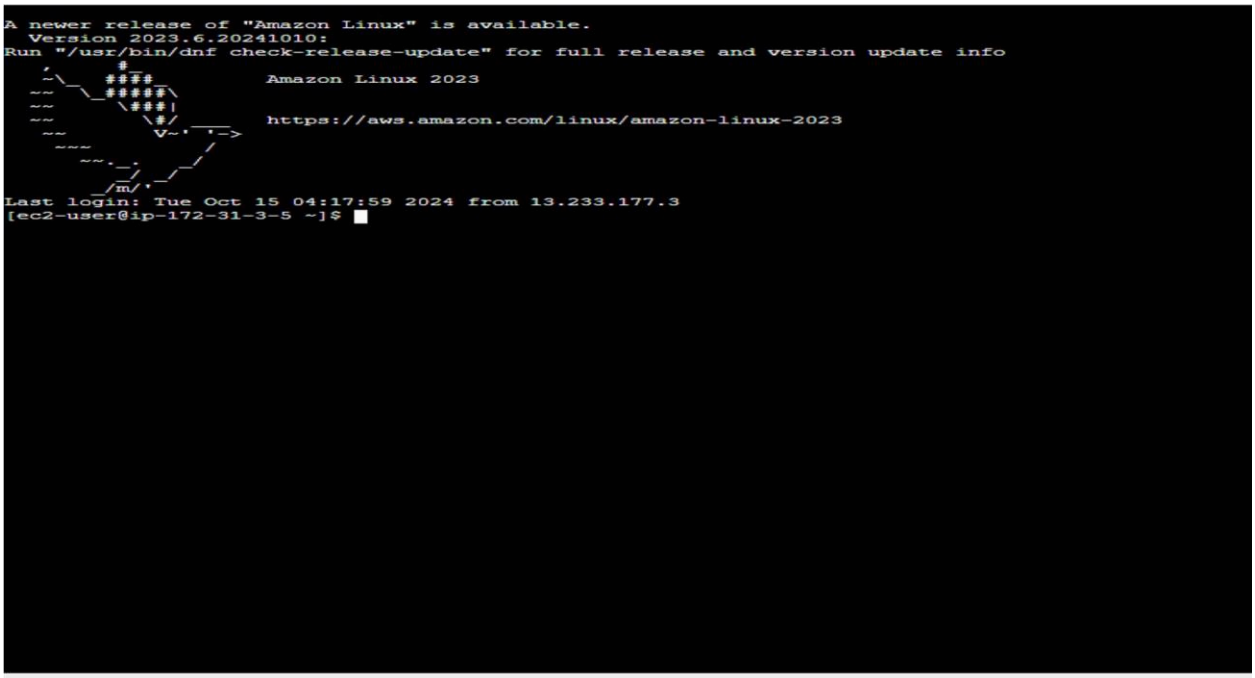




- Now connect the EC2 with the files



Smart Internz


A newer release of "Amazon Linux" is available.
Version 2023.6.20241010:
Run "/usr/bin/dnf check-release-update" for full release and version update info
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
Last login: Tue Oct 15 04:17:59 2024 from 13.233.177.3
[ec2-user@ip-172-31-3-5 ~]$
i-001861022fbcac290 (InstantLibraryApp)
PublicIPs: 13.201.74.42 PrivateIPs: 172.31.3.5

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.

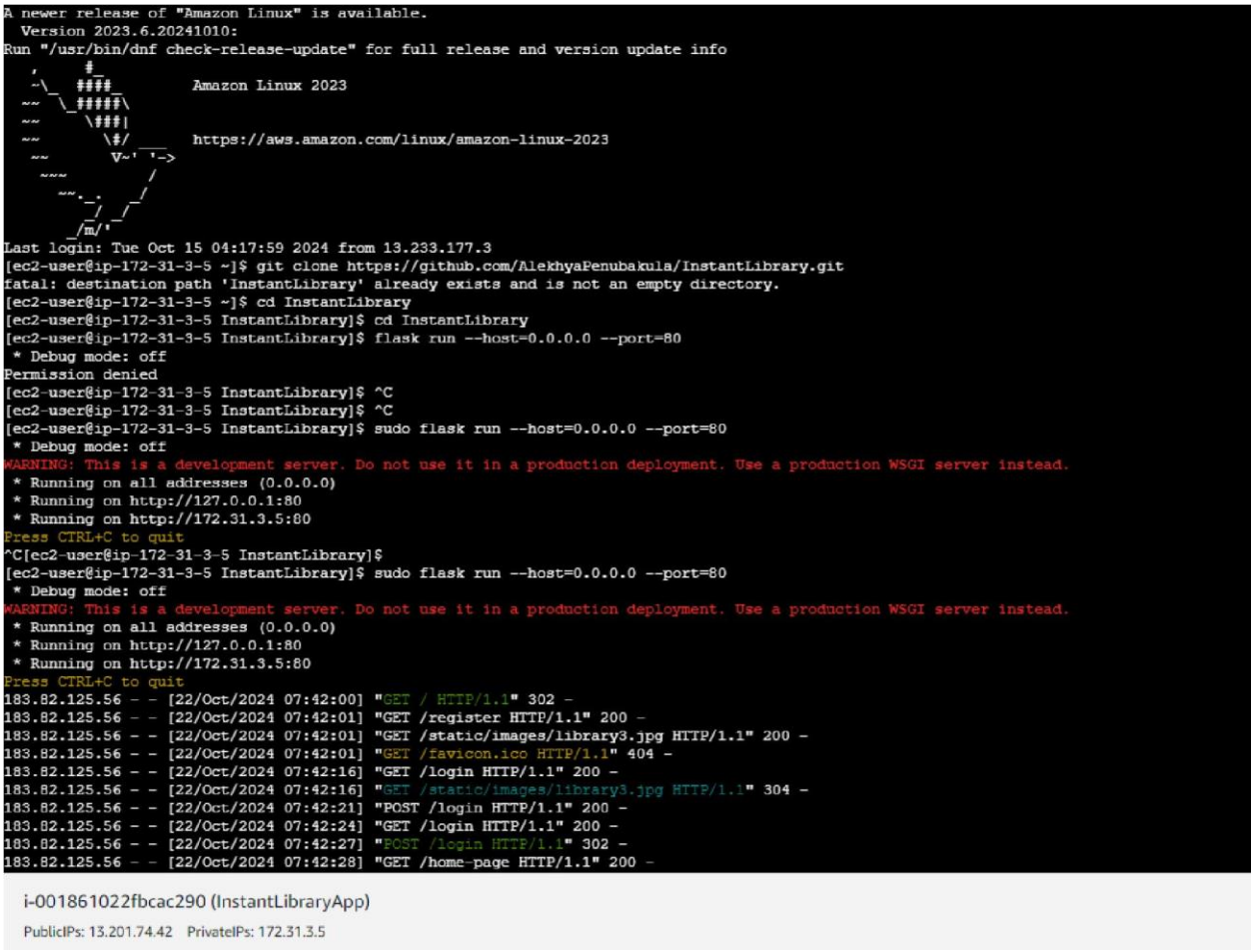
Run: 'git clone https://github.com/your-github-username/your-repository-name.git' Note: change your-github-username and your-repository-name with your credentials here: 'git clone https://github.com/AlekhyaPenubakula/InstantLibrary.git' ● This will download your project to the EC2 instance.

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-publicip

- Run the Flask app on the EC2 instance

```
[ec2-user@ip-172-31-3-5 InstantLibrary]$ sudo flask run --host=0.0.0.0 --port=80
 * Debug mode: off
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://127.0.0.1:80
 * Running on http://172.31.3.5:80
Press CTRL+C to quit
183.82.125.56 - - [22/Oct/2024 07:42:00] "GET / HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /register HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /static/images/library3.jpg HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /favicon.ico HTTP/1.1" 404 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /static/images/library3.jpg HTTP/1.1" 304 -
183.82.125.56 - - [22/Oct/2024 07:42:21] "POST /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:24] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:27] "POST /login HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:28] "GET /home-page HTTP/1.1" 200 -
```

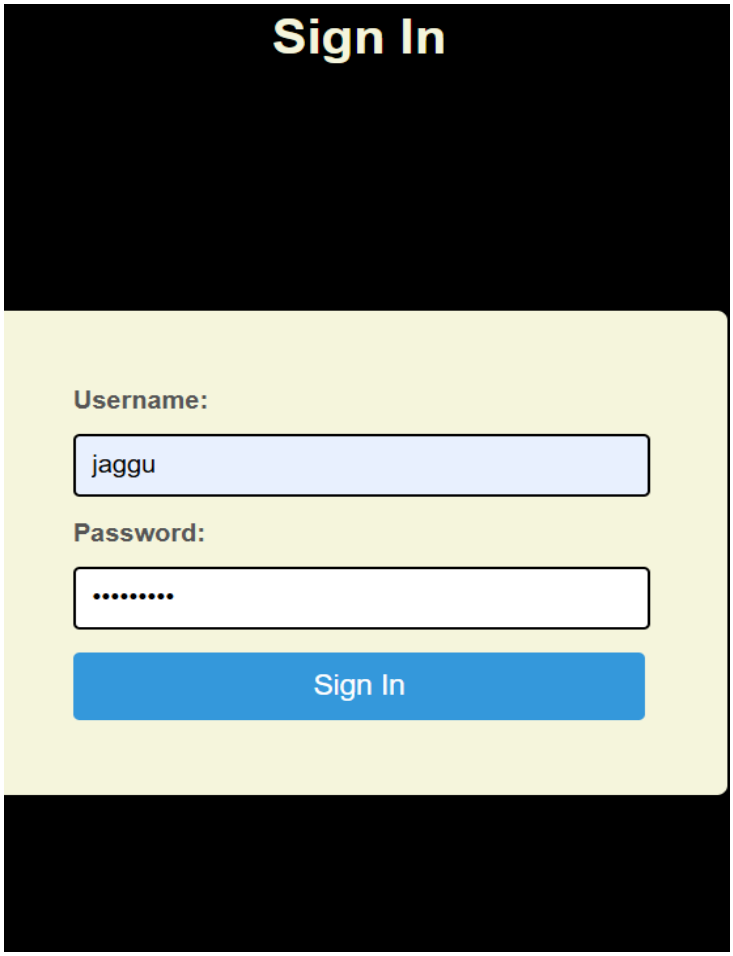
Access the website through:

PublicIPs: http:3.92.196.29:5000

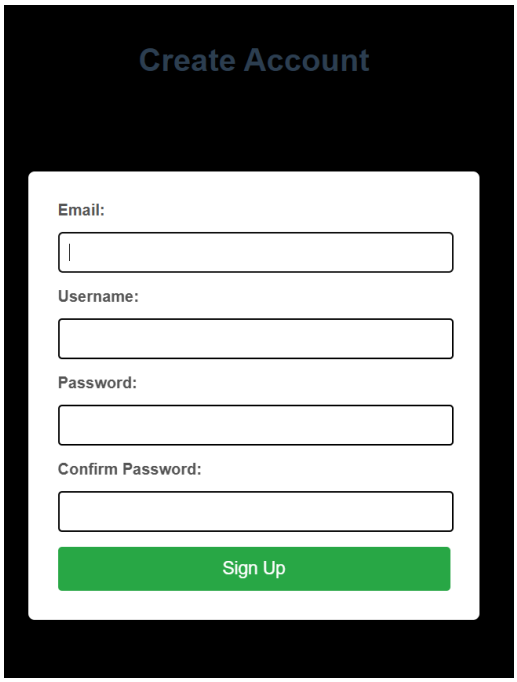
Milestone 8: Testing and Deployment

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

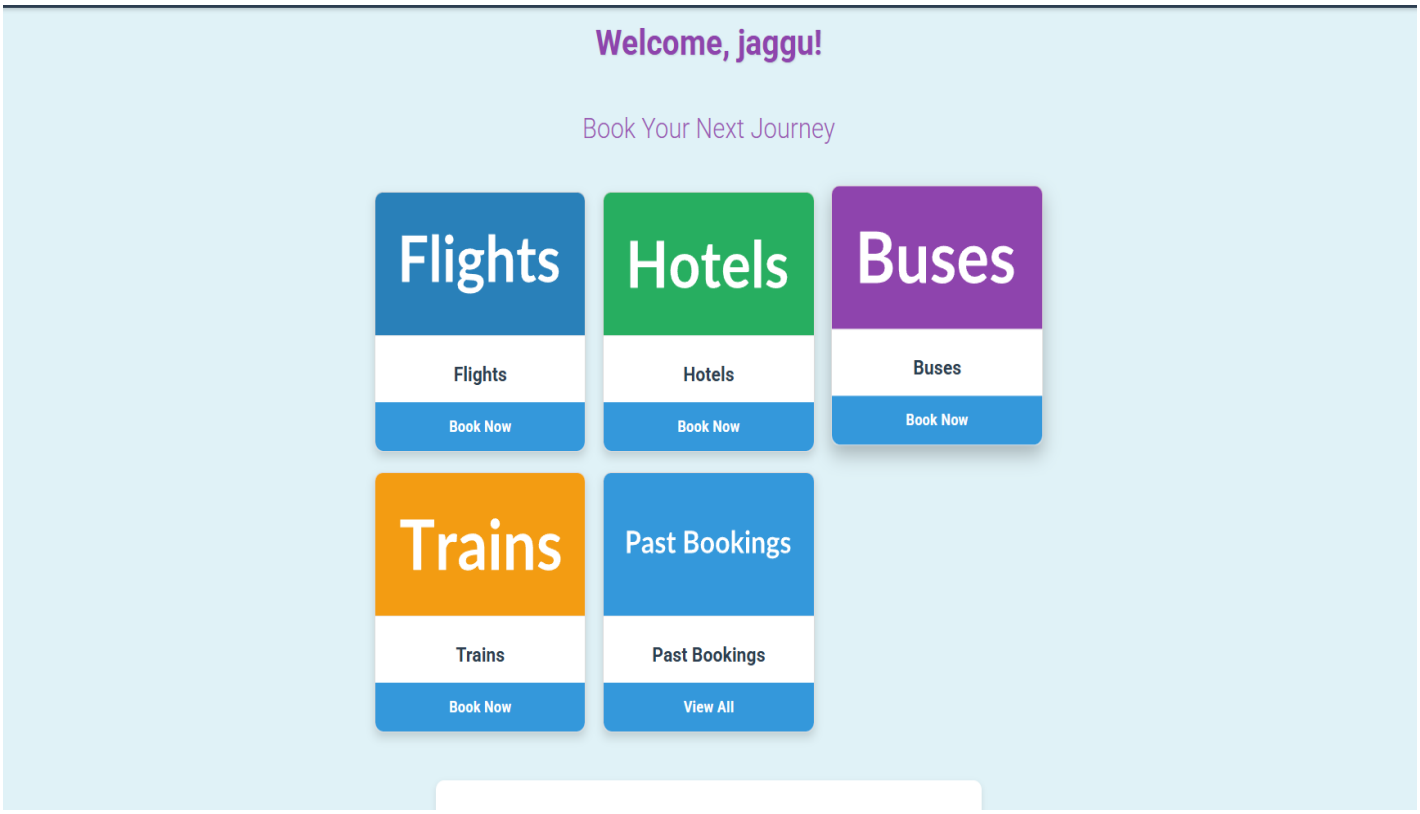
Login Page:



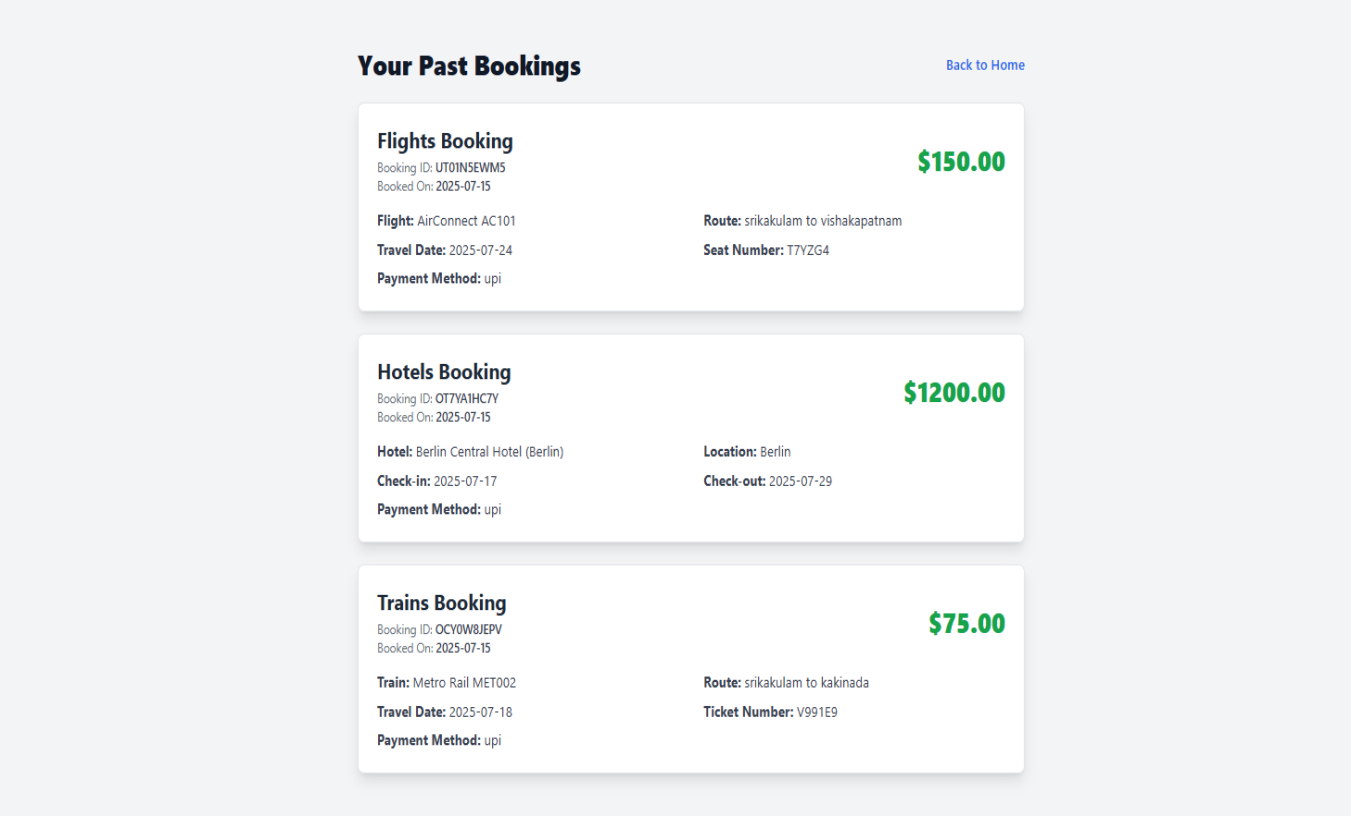
Signup page:


Create Account
Email:
Username:
Password:
Confirm Password:
Sign Up

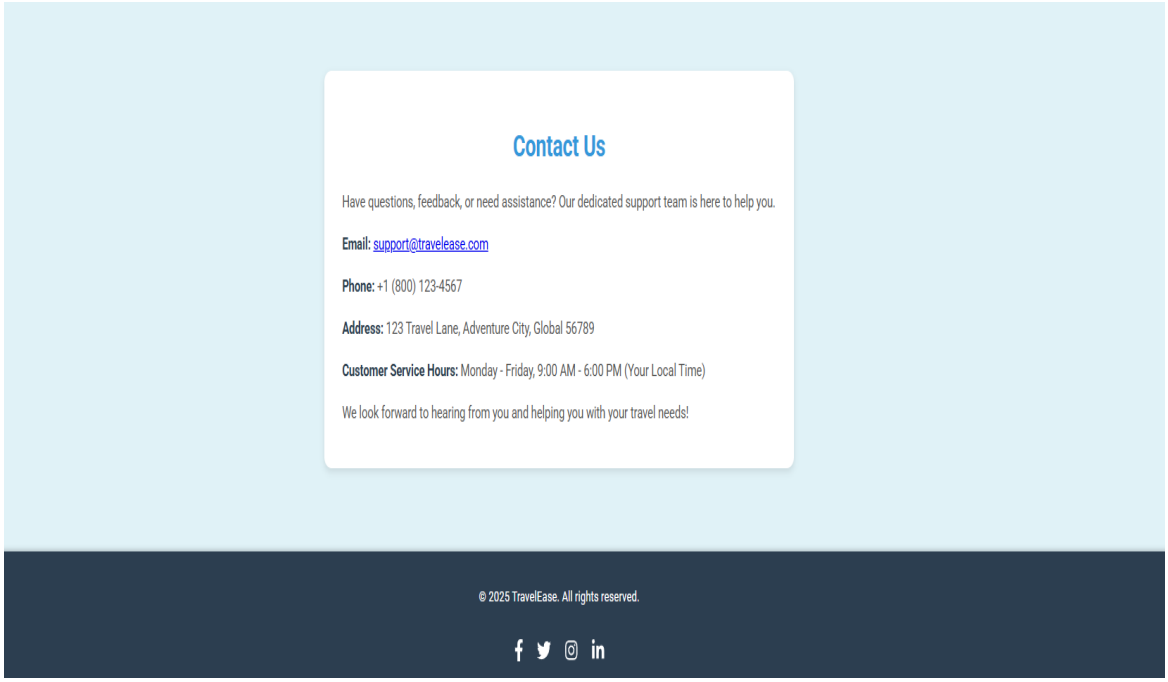
Home page:


Welcome, jaggu!
Book Your Next Journey
Flights
Flights
Book Now
Hotels
Hotels
Book Now
Buses
Buses
Book Now
Trains
Trains
Book Now
Past Bookings
Past Bookings
View All

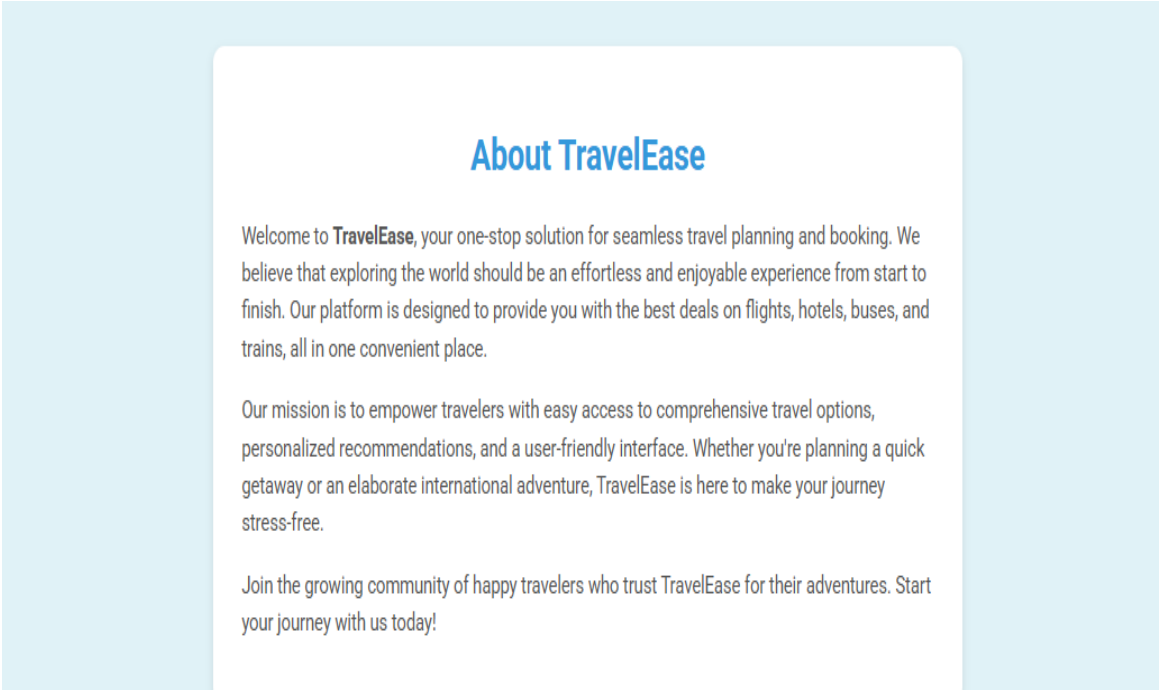
About Bookings page:



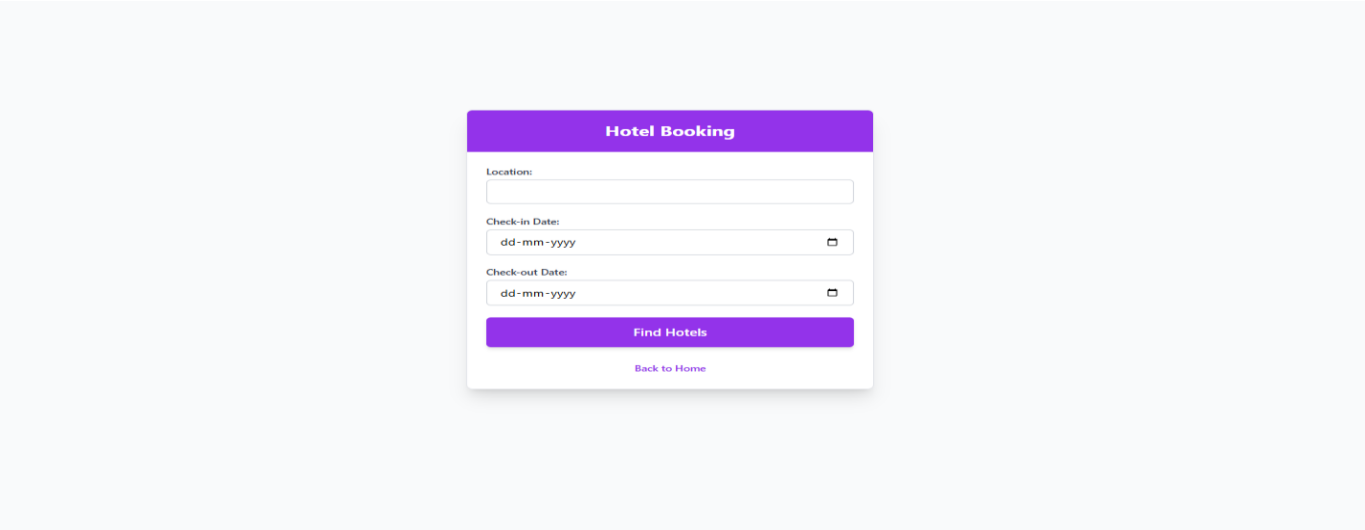
Contact Page:



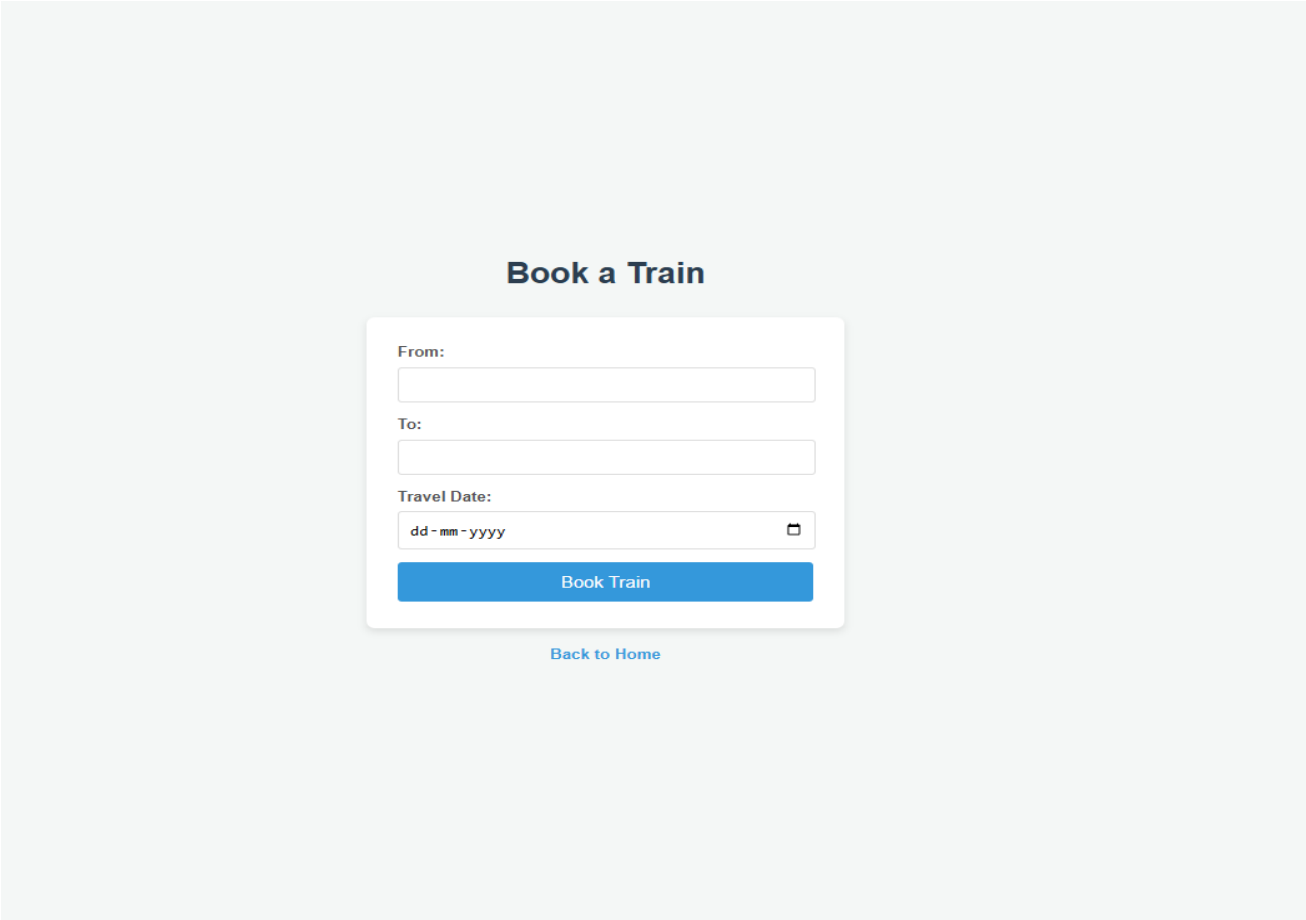
About page:



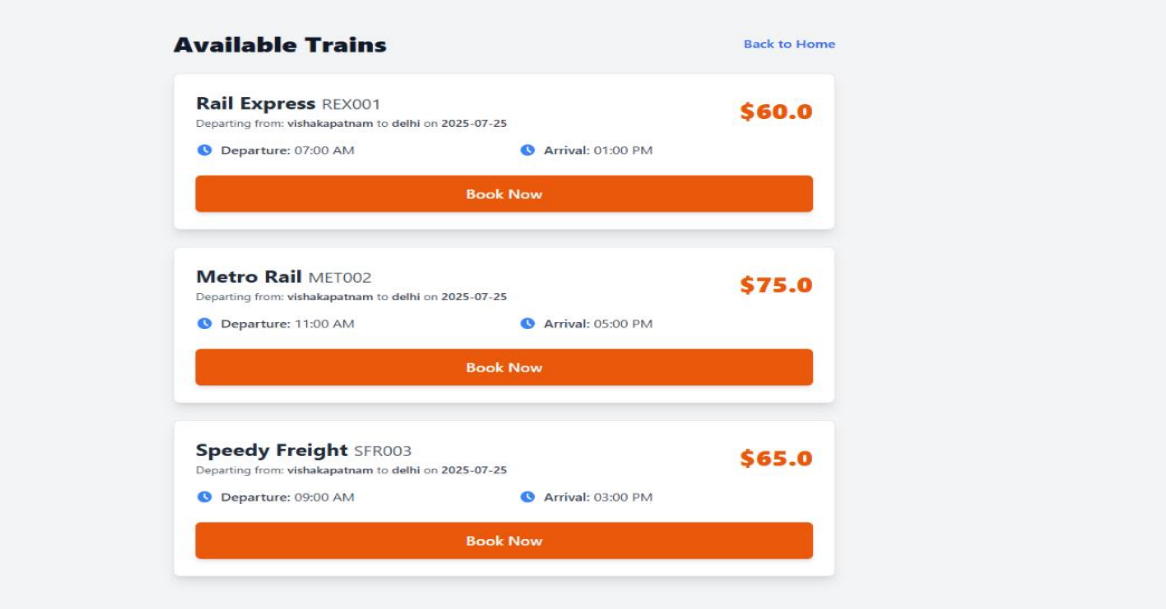
Hotels Booking :



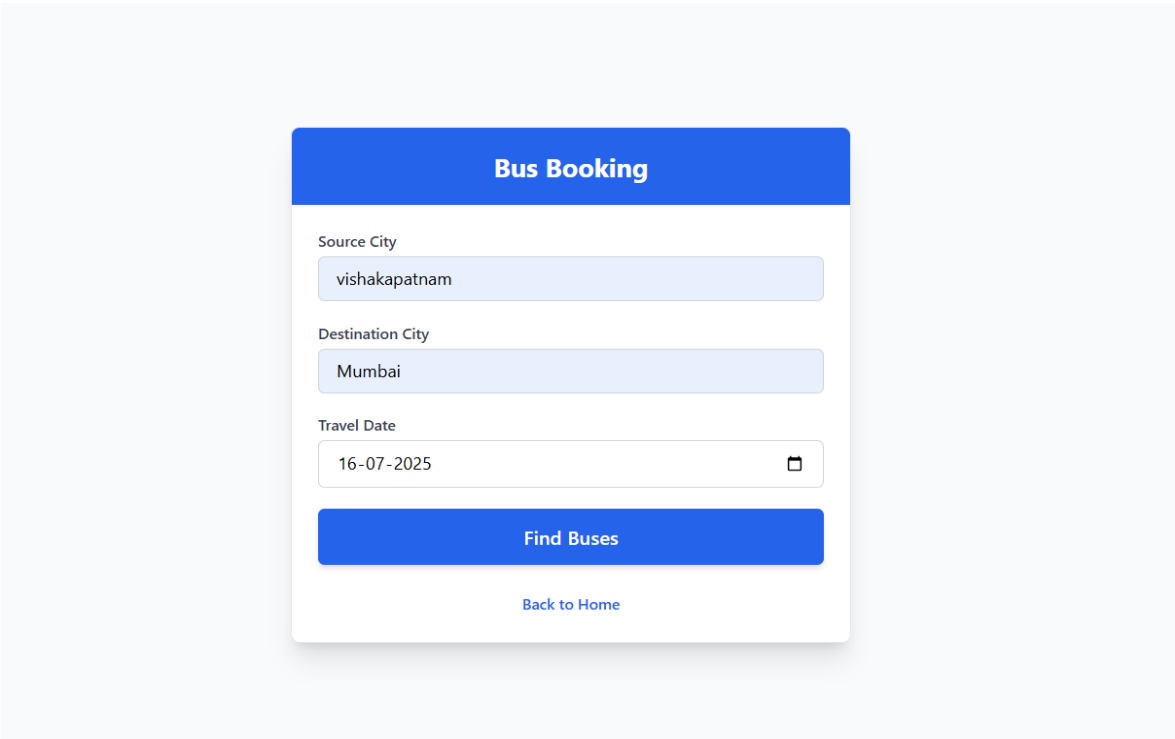
Trains Booking:

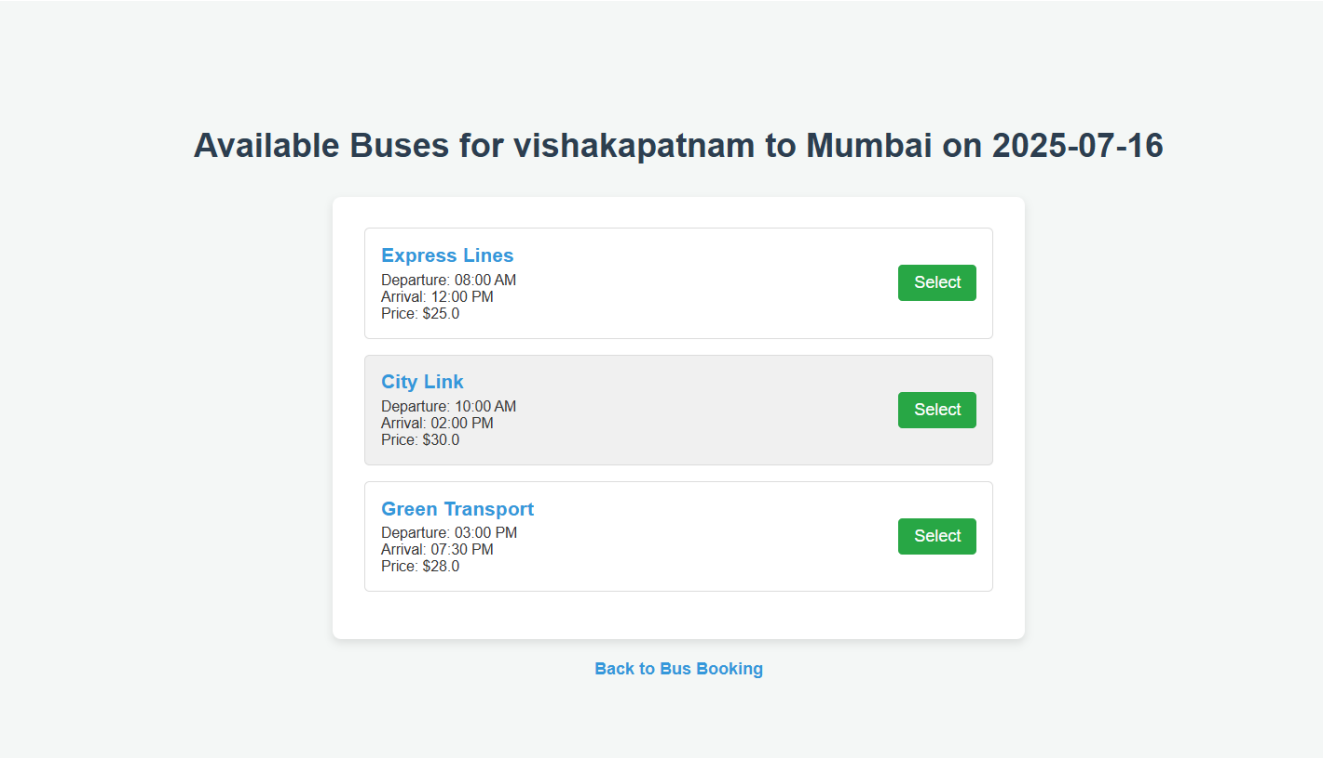


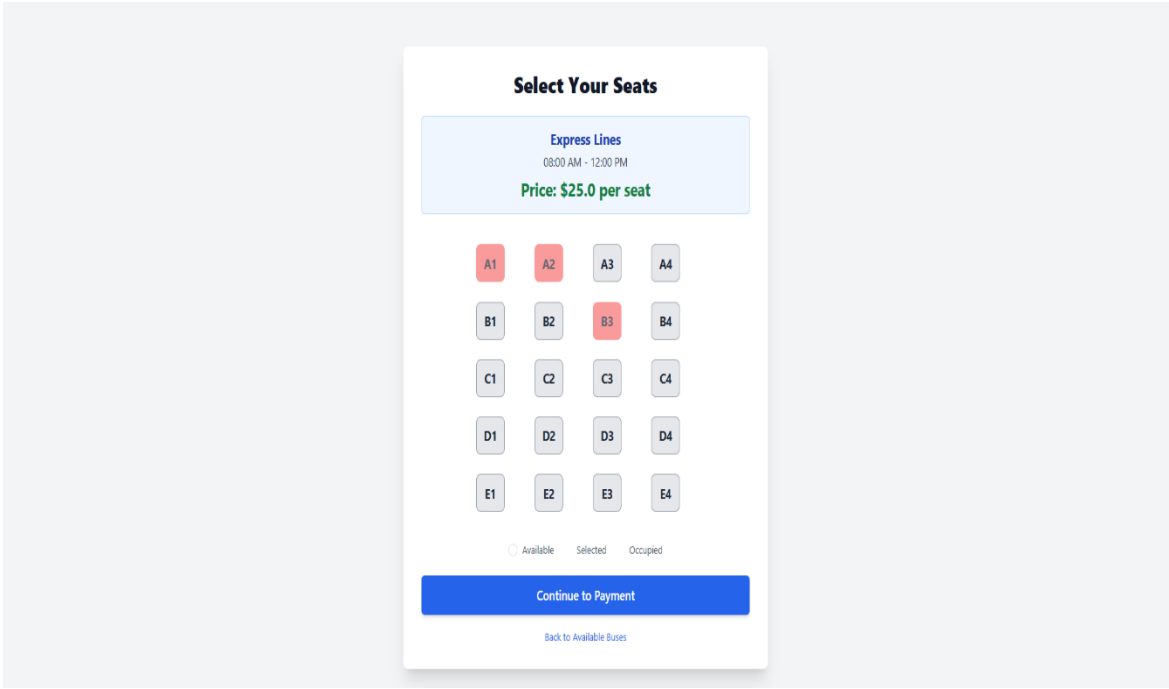
Available Trains:



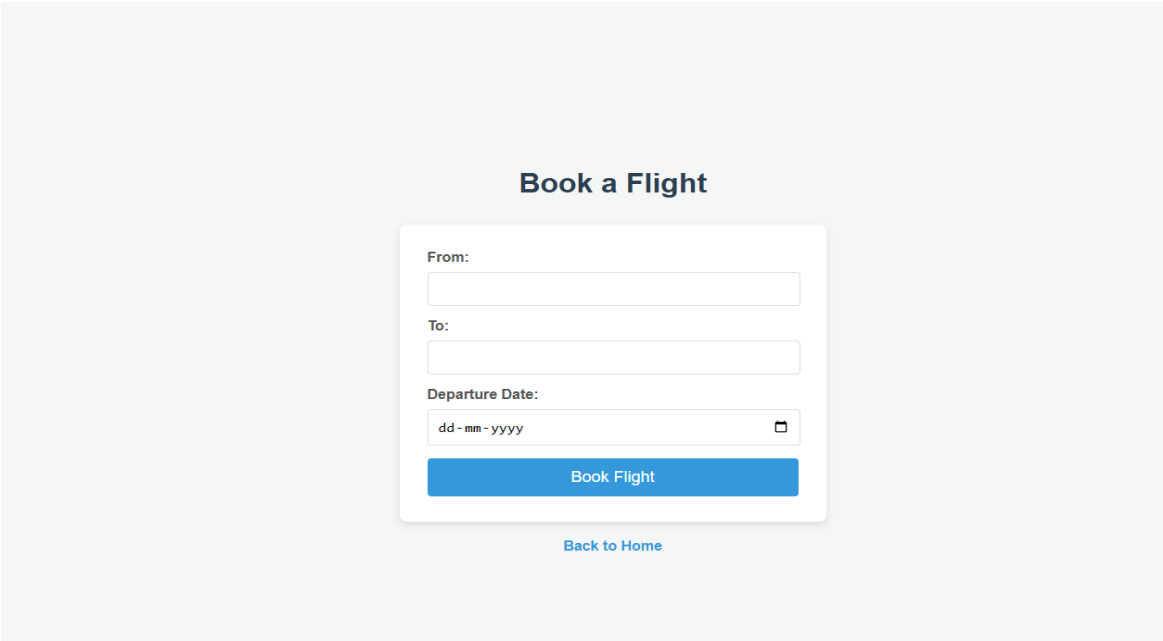
Buses Booking:

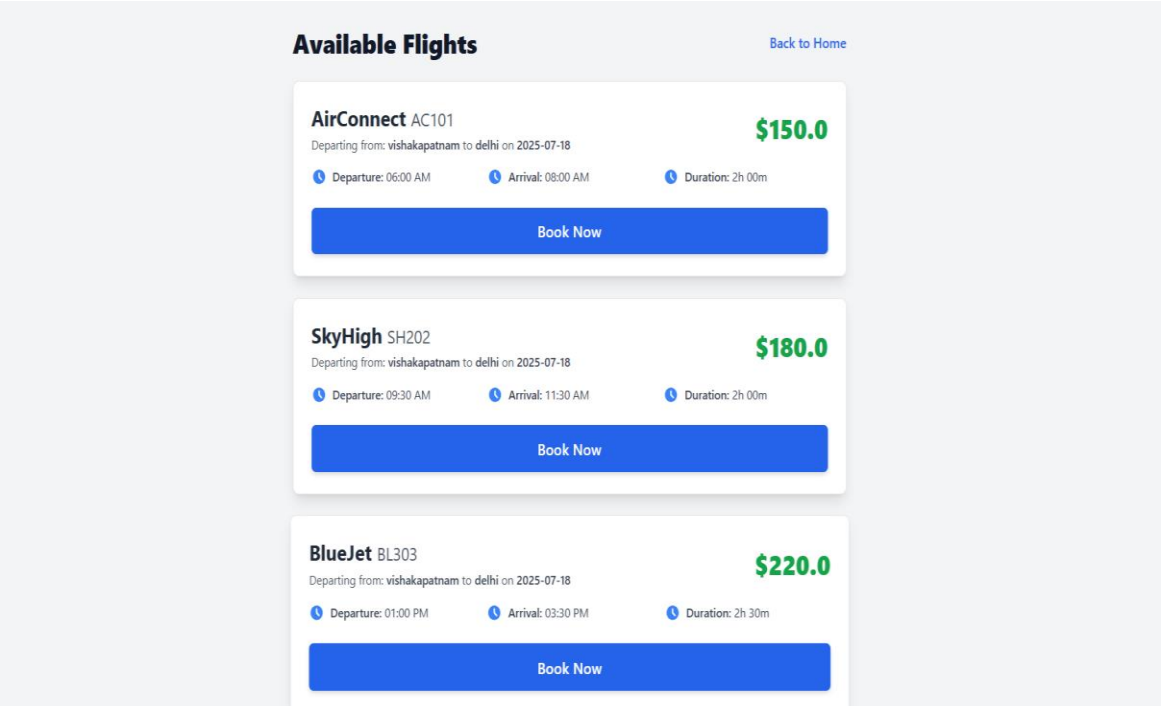



Available Buses for vishakapatnam to Mumbai on 2025-07-16
Express Lines
Departure: 08:00 AM
Arrival: 12:00 PM
Price: $25.0
Select
City Link
Departure: 10:00 AM
Arrival: 02:00 PM
Price: $30.0
Select
Green Transport
Departure: 03:00 PM
Arrival: 07:30 PM
Price: $28.0
Select
Back to Bus Booking


Select Your Seats
Express Lines
08:00 AM - 12:00 PM
Price: $25.0 per seat
A1
A2
A3
A4
B1
B2
B3
B4
C1
C2
C3
C4
D1
D2
D3
D4
E1
E2
E3
E4
Available
Selected
Occupied
Continue to Payment
Back to Available Buses

Flights Booking:





Complete Your Booking

Booking Details:

Type: flights

From: vishakapatnam

To: delhi

Date: 2025-07-18

Select Payment Method:

- ○ Credit Card
- ○ PayPal
- ○ Bank Transfer
- ◉ upi

Complete Payment

Cancel and Return Home

Conclusion:

The TravelGo Website has been successfully developed and deployed using a scalable and cloudnative 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.