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

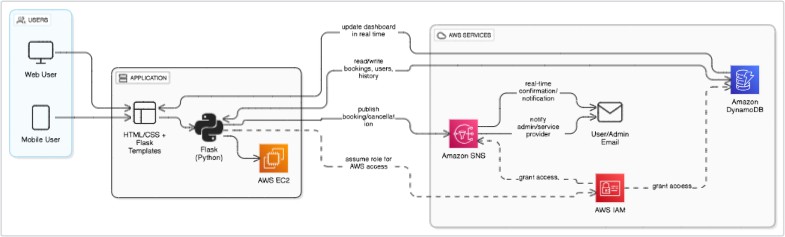
responsive HTML/CSS and Flask templates, ensures users can review or manage bookings anytime, from any device, with real-time updates and quick cancellation workflows supported.

## AWS ARCHITECTURE

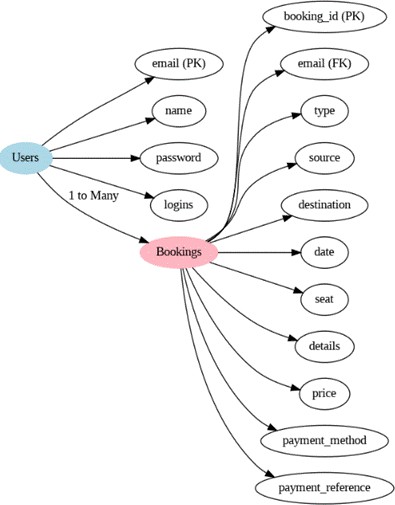
This AWS-based architecture powers a scalable and secure web application using Amazon EC2 for

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.ht](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html) [ml](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html)
* Amazon SNS : <https://docs.aws.amazon.com/sns/latest/dg/welcome.htm>l
* 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

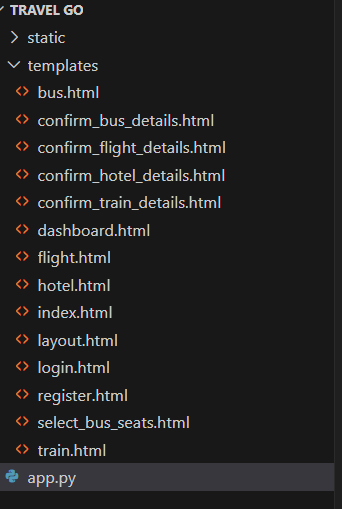
* Upload Flask Files
* Run the Flask App

**Milestone 8.** Testing and Deployment

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

# Milestone 1: Backend Development and Application SetUp

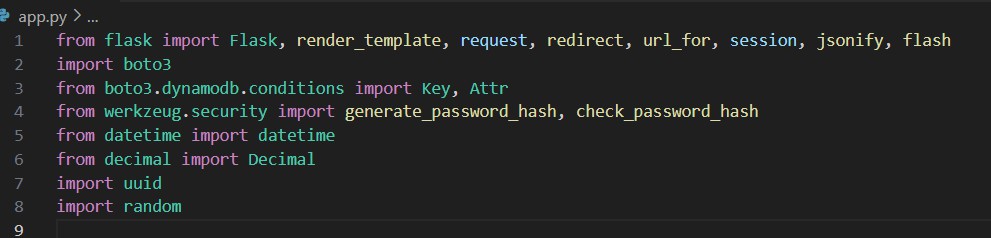
* **Activity 1.1:** Develop the Backend Using Flask

1. File Explorer Structure

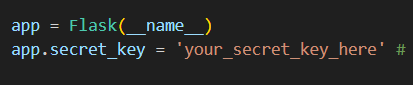
**Description:** Organize the project with HTML templates for each feature (e.g., login, wishlist, quiz, checkout) under the templates folder and manage backend logic in app.py .

### Description of the code :

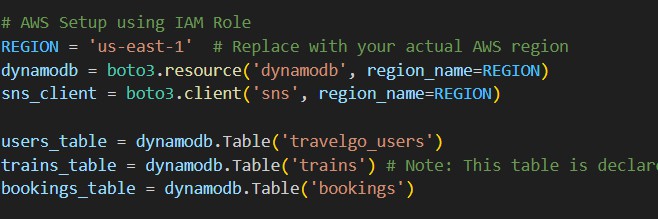
? Flask App Initialization



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



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

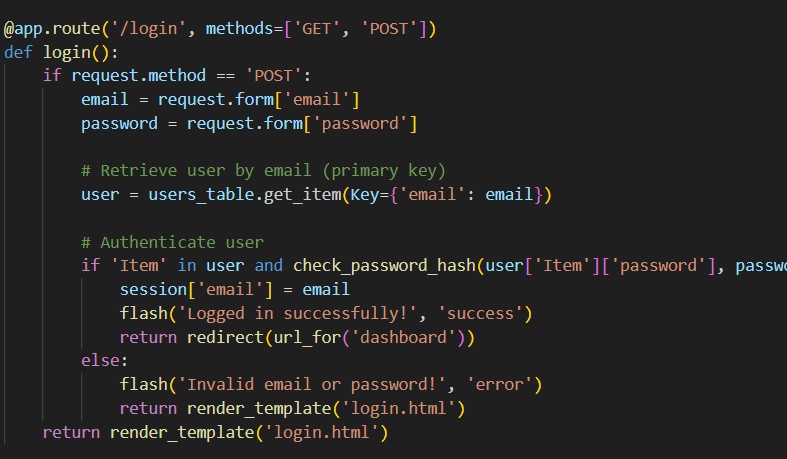


* + Connect to DynamoDB using Boto3 and define references to the UserTable and WishlistTable for user and wishlist data operations.

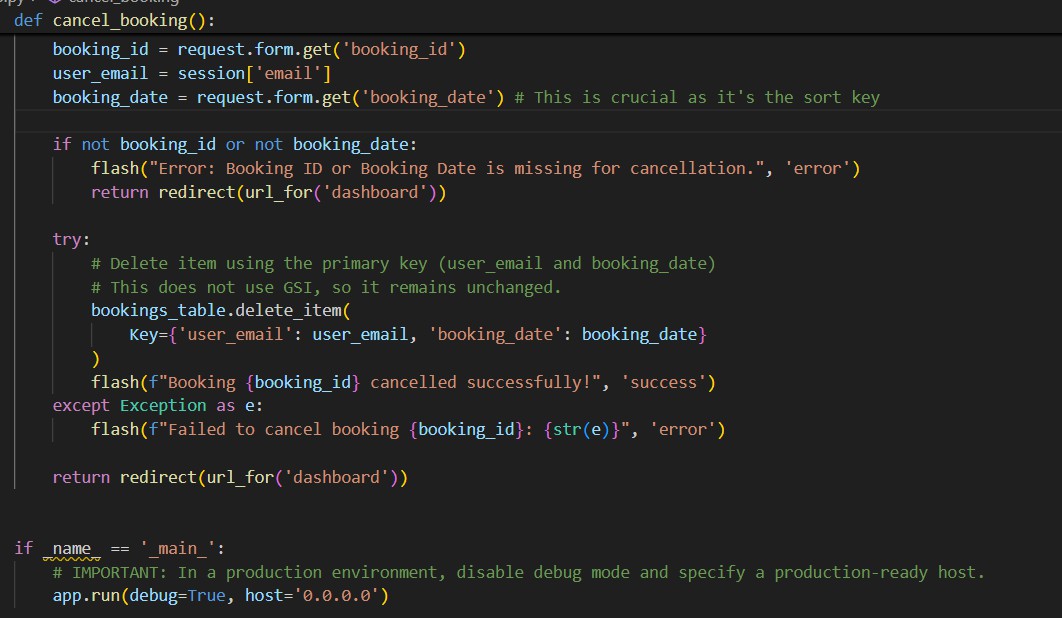
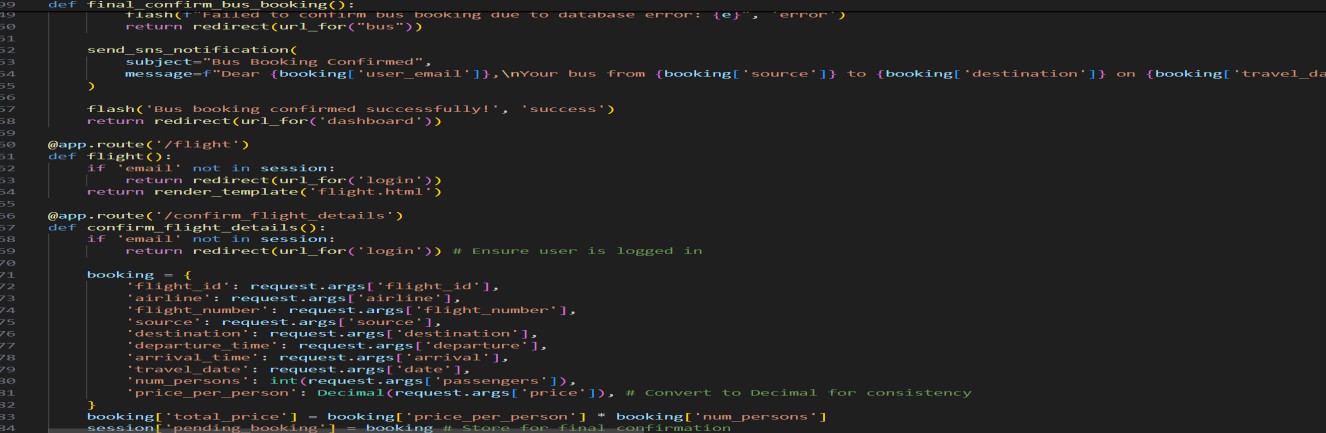
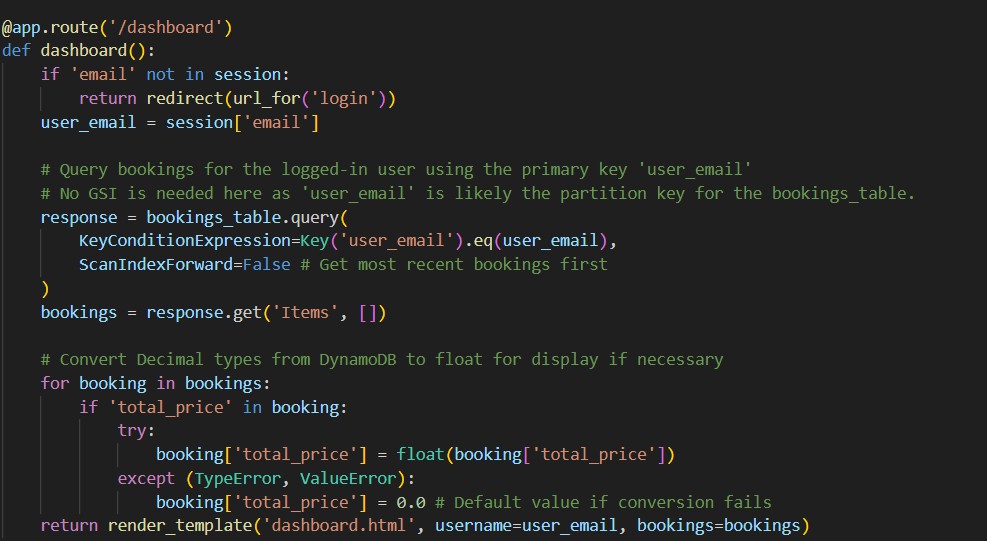
**Routes for Core Functionalities:**



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

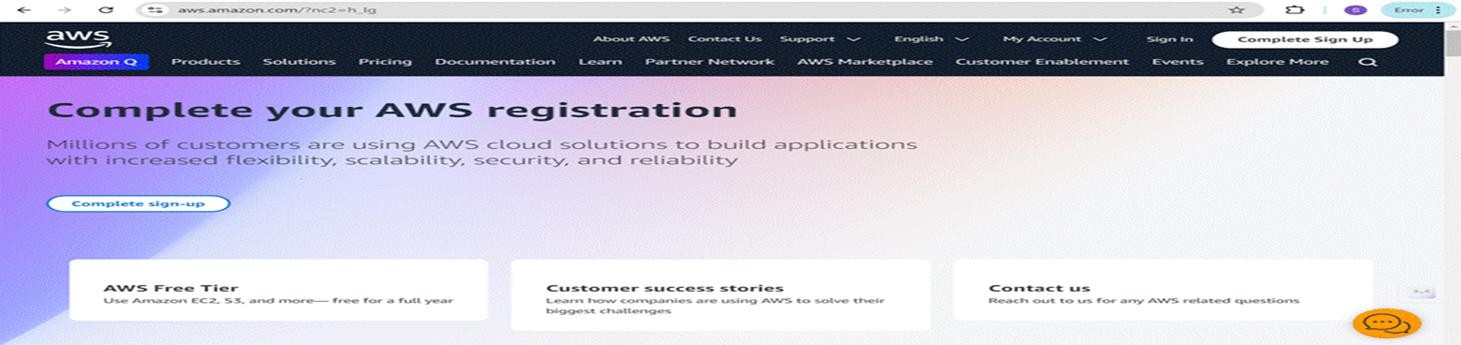
**Login route**: Implement the user login route to validate credentials using DynamoDB and securely manage session data while updating the user's login count.

* **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.



# Milestone 2: AWS Account Setup and Login.

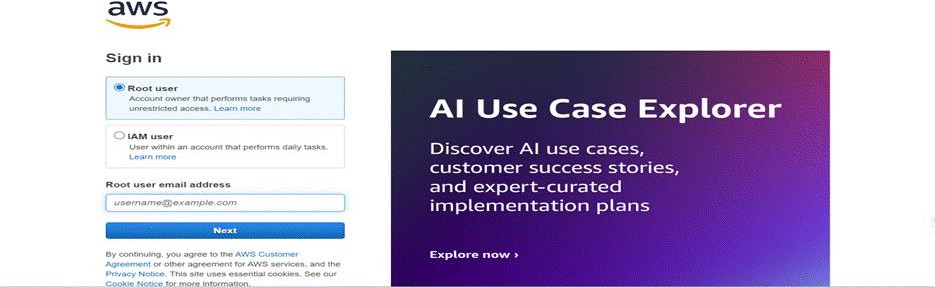
* + **Activity 2.1:** Set up an AWS account if not already done.
    - Sign up for an AWS account and configure billing settings.



* 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**.**

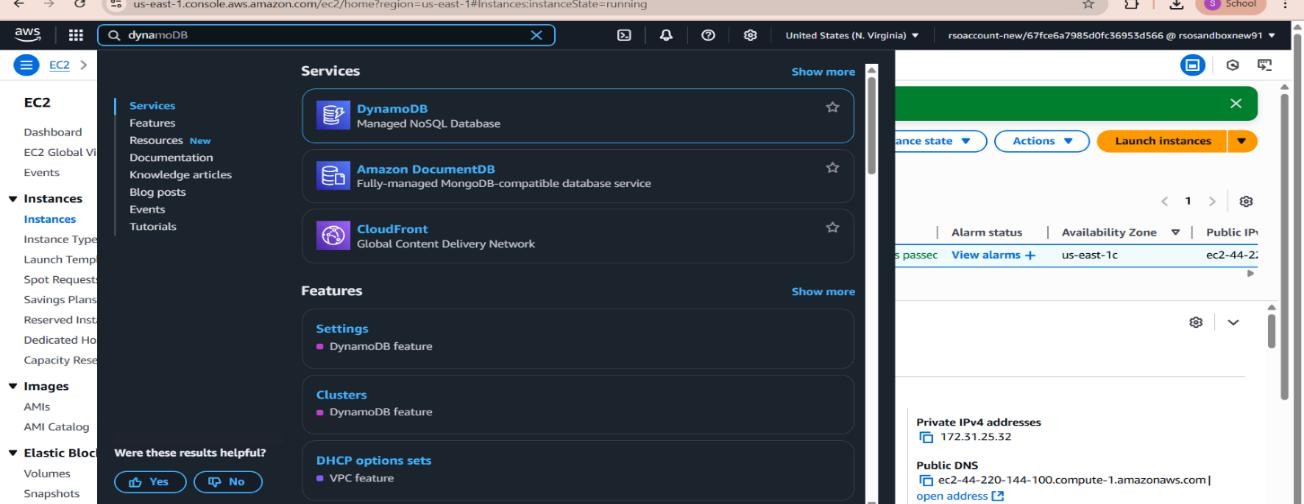


* **Activity2.2 :** Log in to the AWS Management Console
  + After setting up your account, log in to the [AWS Management Console](https://aws.amazon.com/console/)**.**

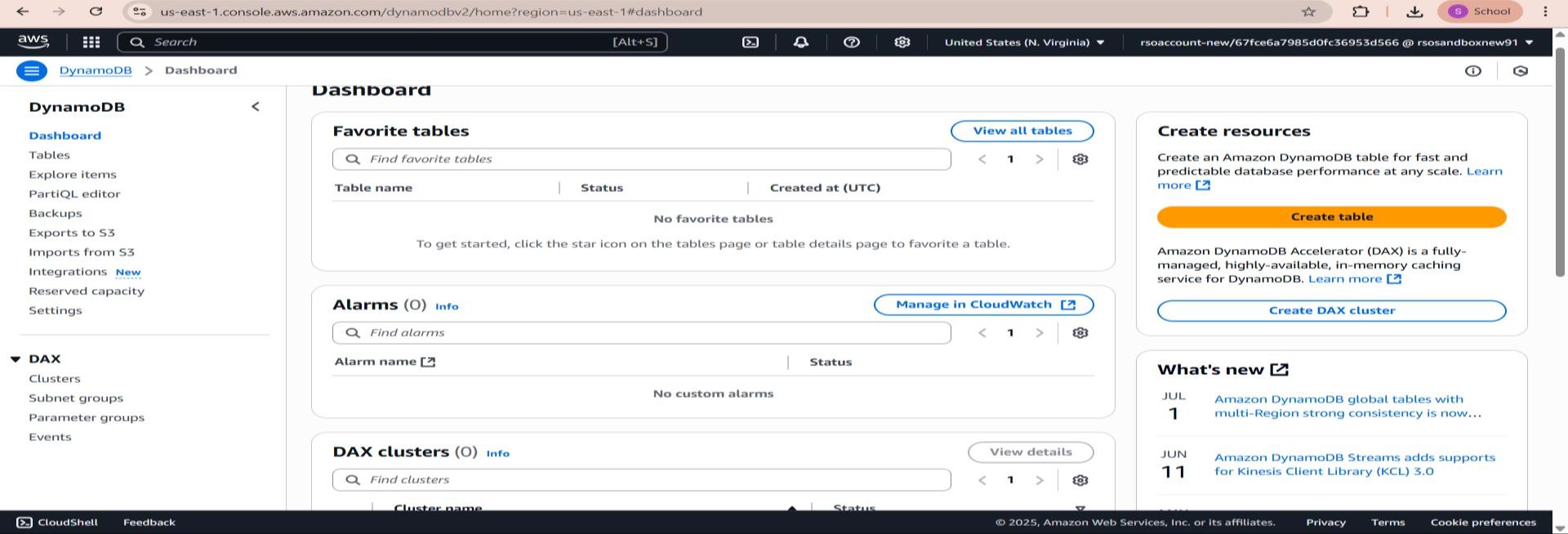
****

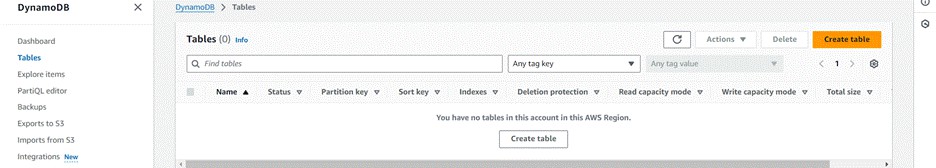
# Milestone 3: DynamoDB Database Creation and Setup

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

****



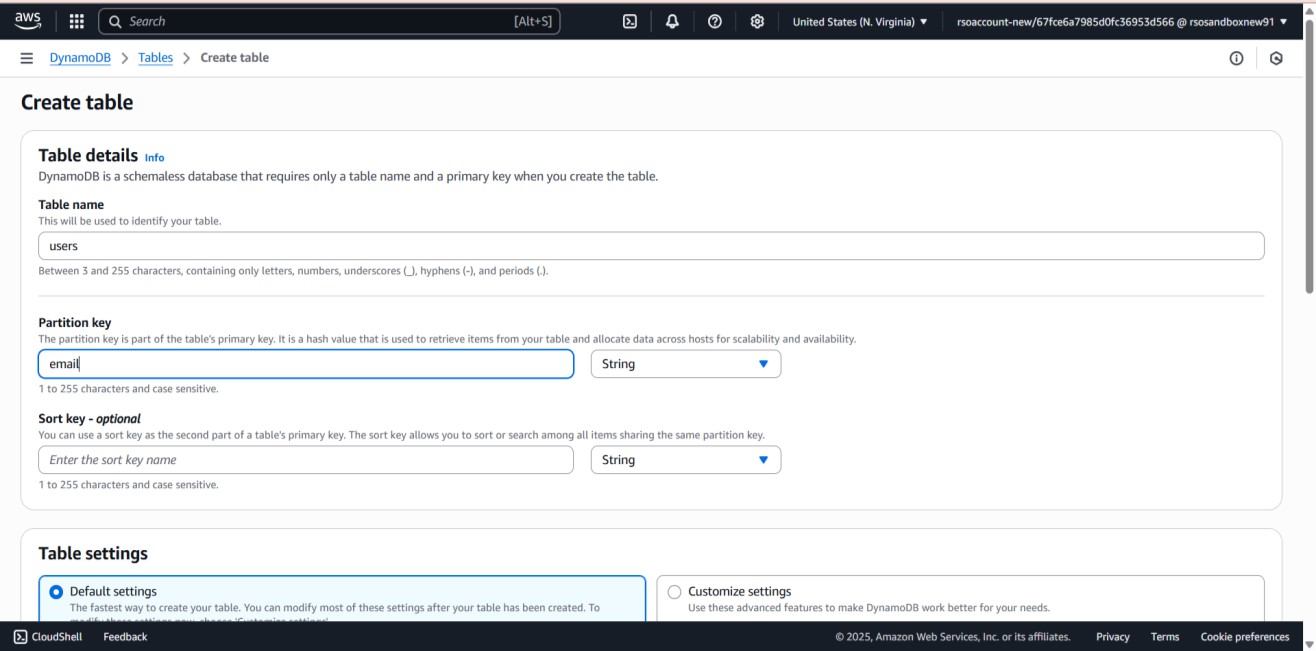


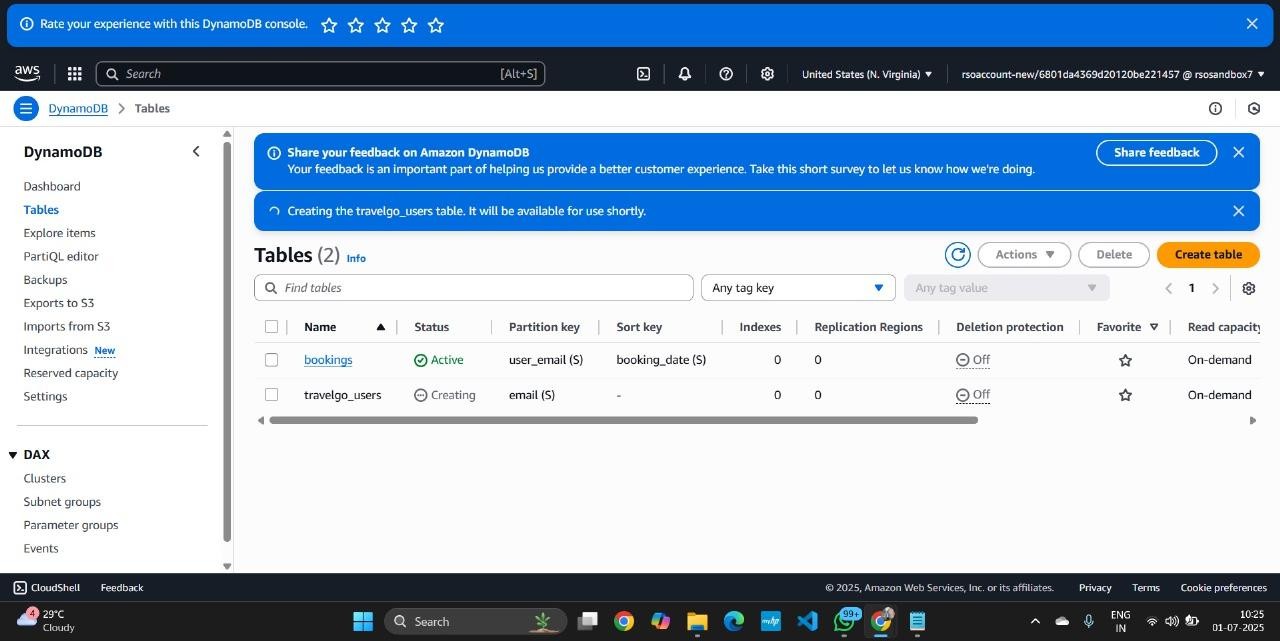
****

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

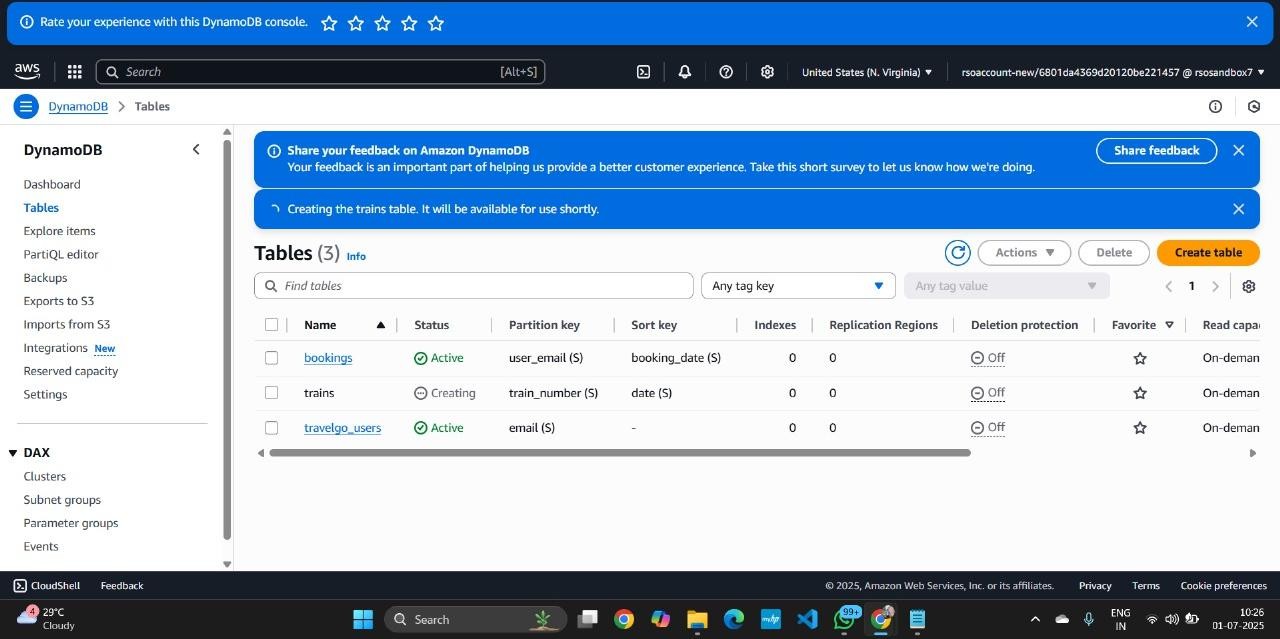
* Create Users table with partition key “Email ” with type String and click on create tables.

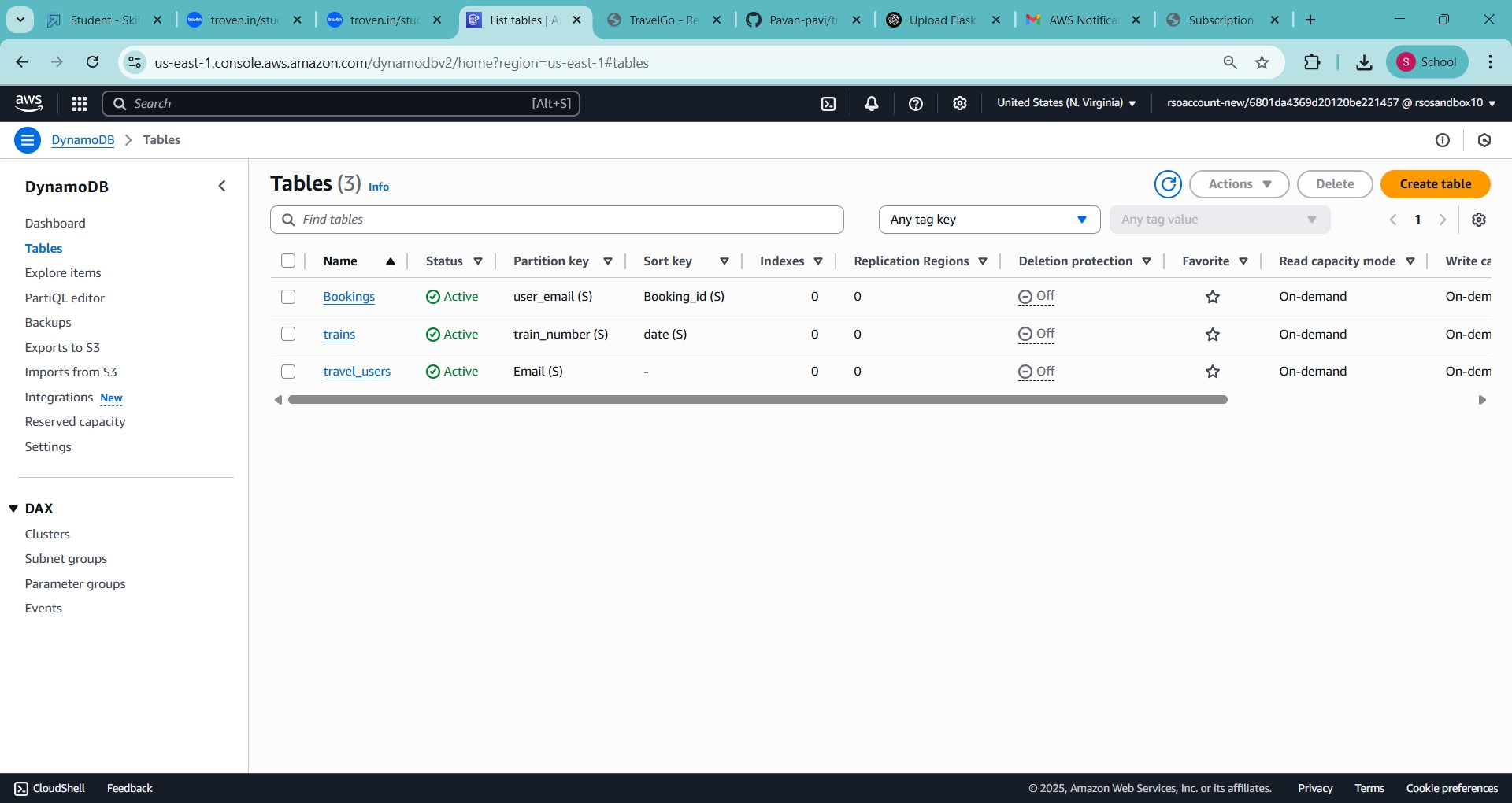






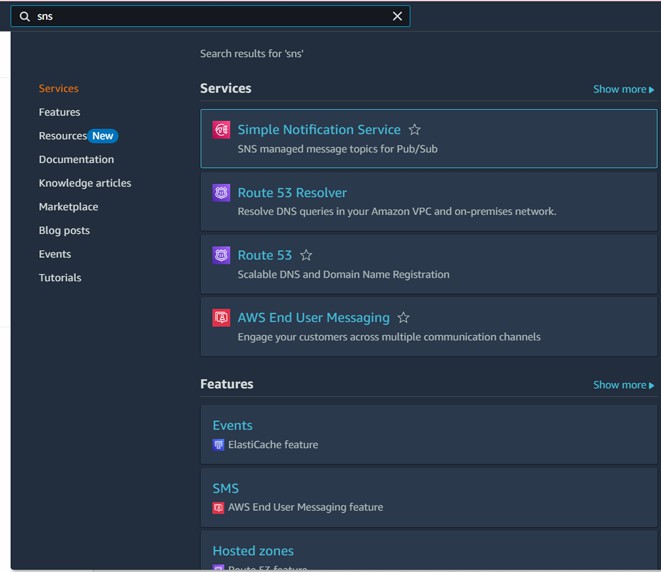
* Follow the same steps to create a Bookings for storing booking records with email as the partition key and booking\_id as the sort key.





# Milestone 4. SNS Notification Setup

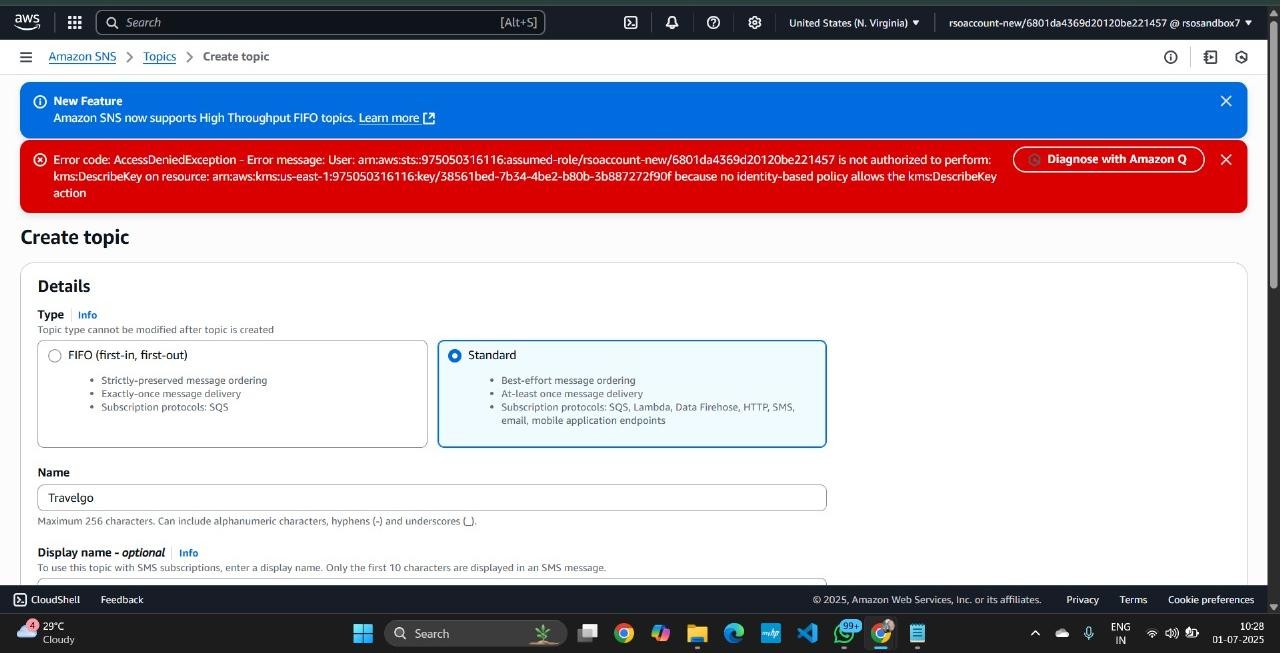
* **Activiy4.1:** Create SNS topics for book request notifications**.**

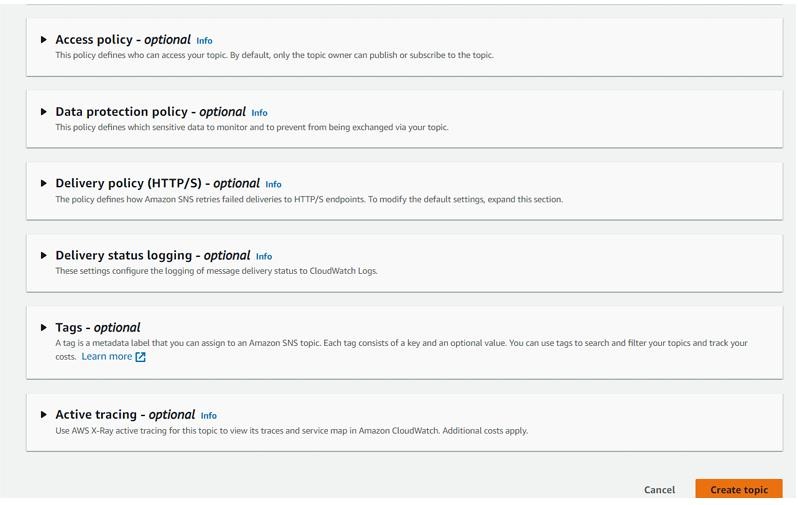
****

****

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

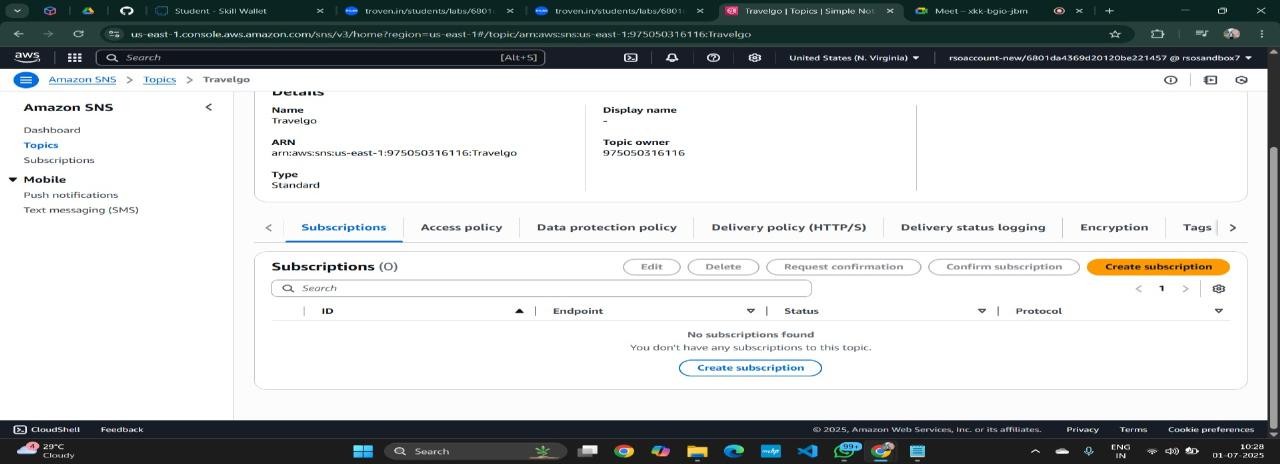




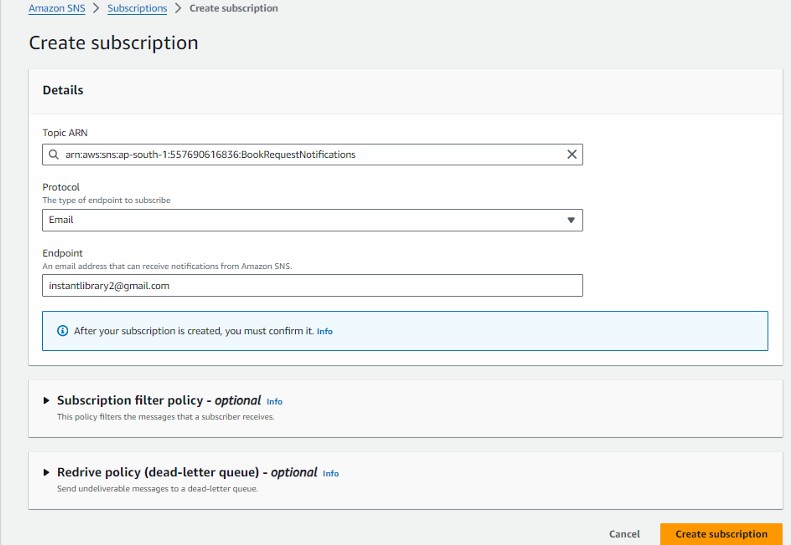


* Configure the SNS topic and note down the Topic ARN

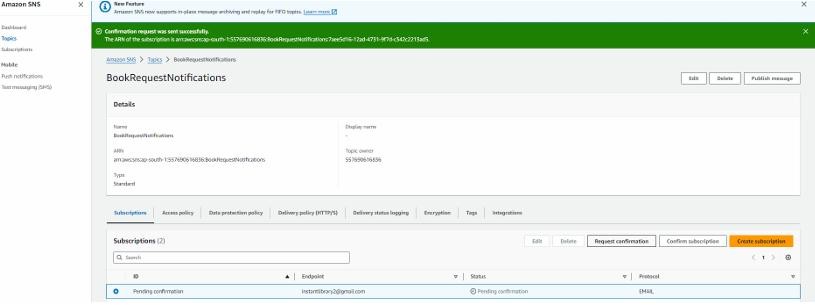




* Subscribe users and library staff to SNS email notifications.

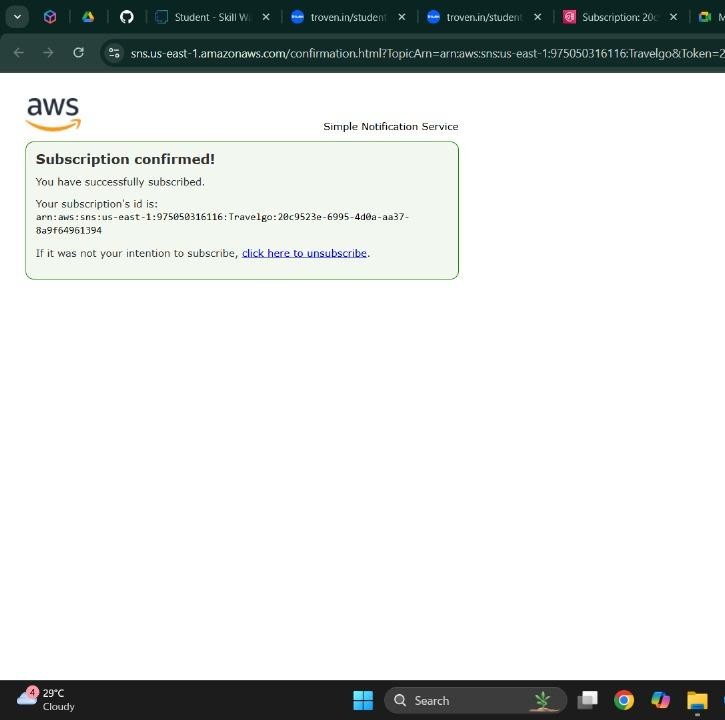






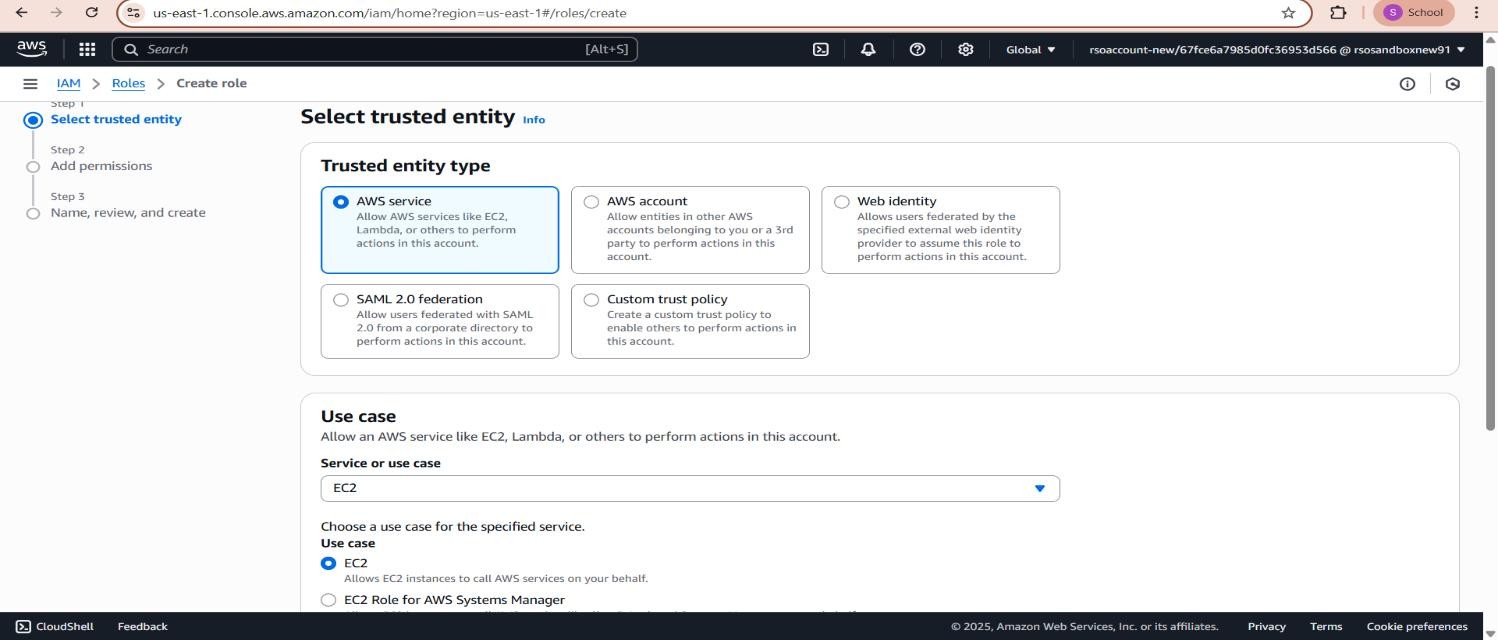
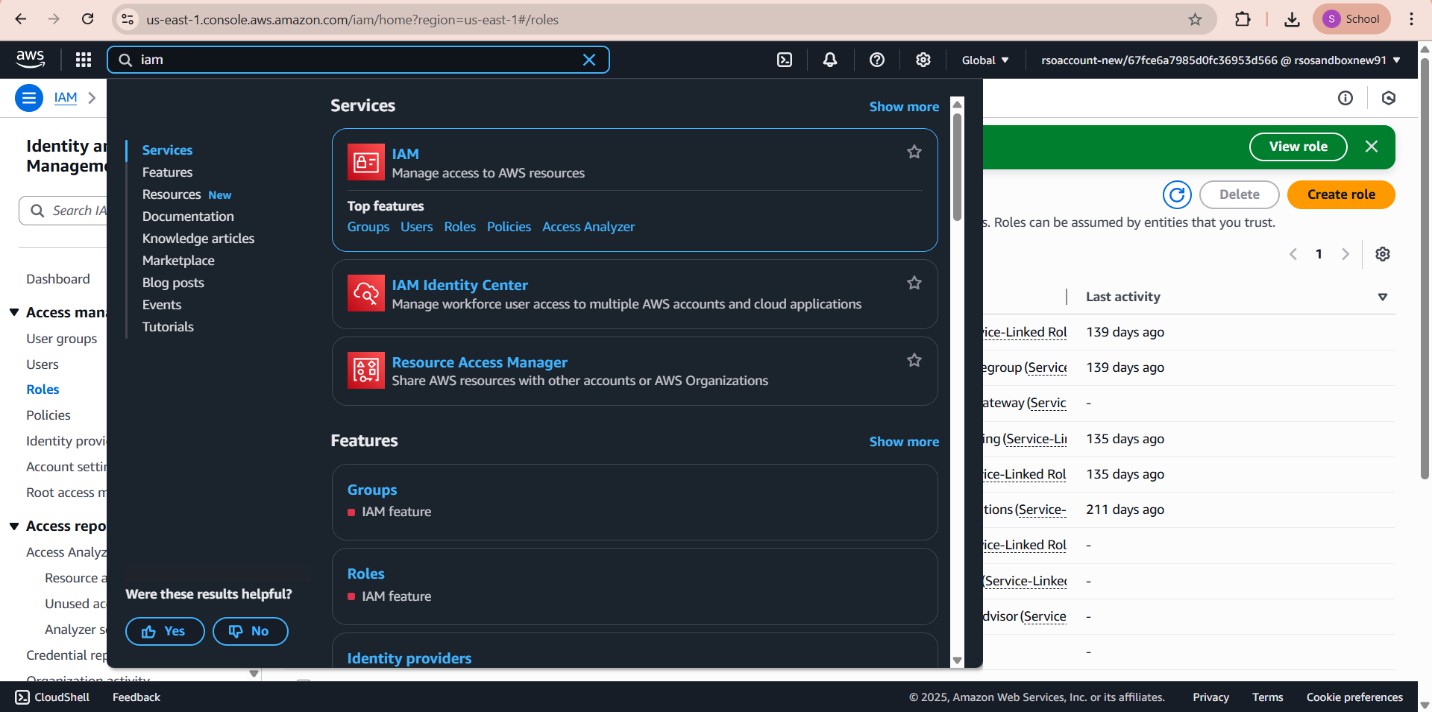
* + After Comfirmation of Subscription going to Mail for comfirm Mail .



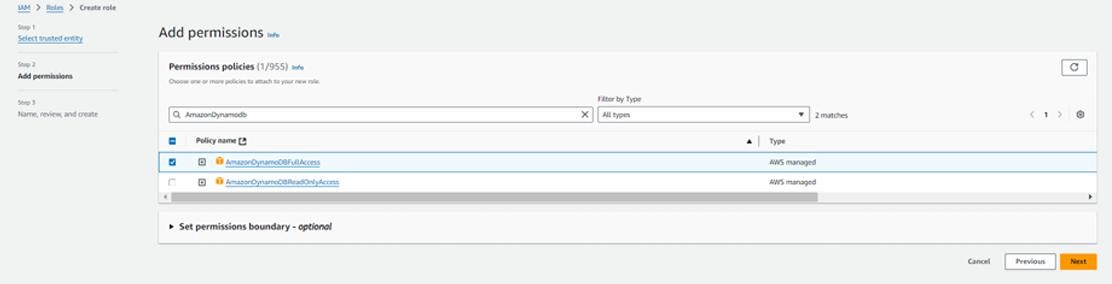


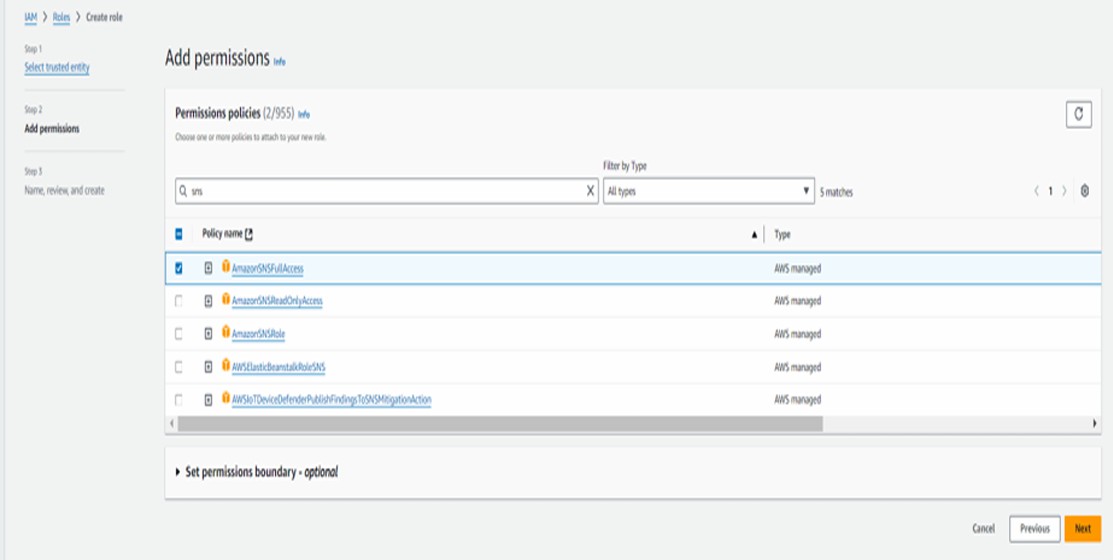
# Milestone 5 : IAM Role Setup.

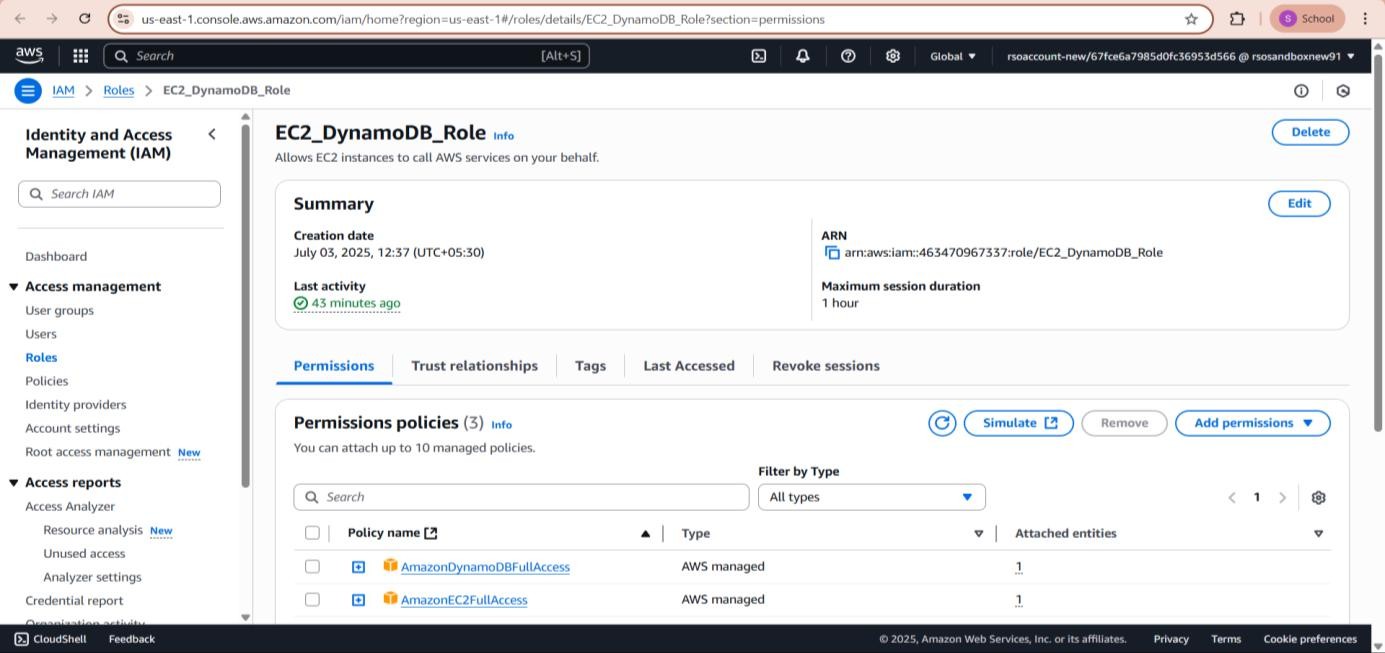
* + **Activity 5.1 :** In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB .



* + **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.

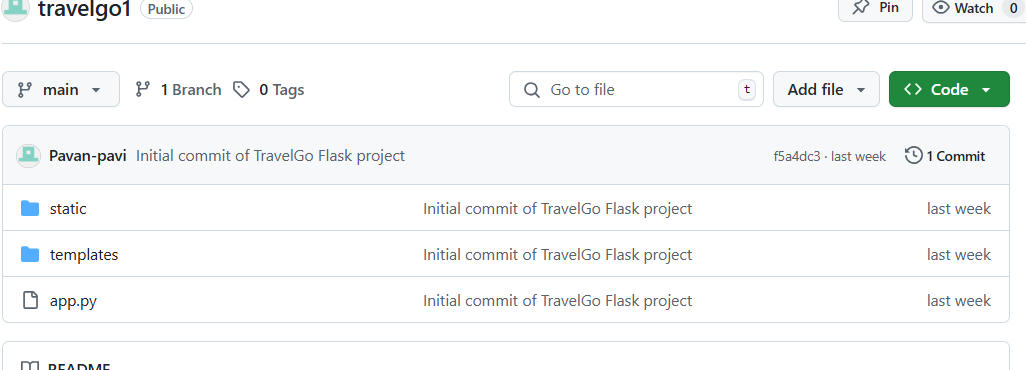


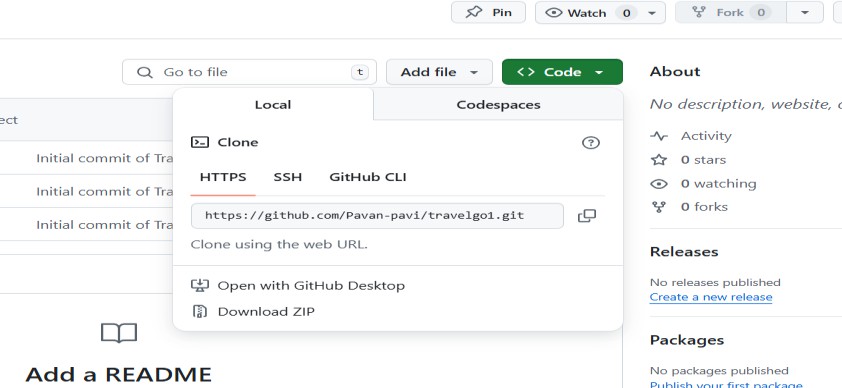




# Milestone 6. EC2 Instance Setup

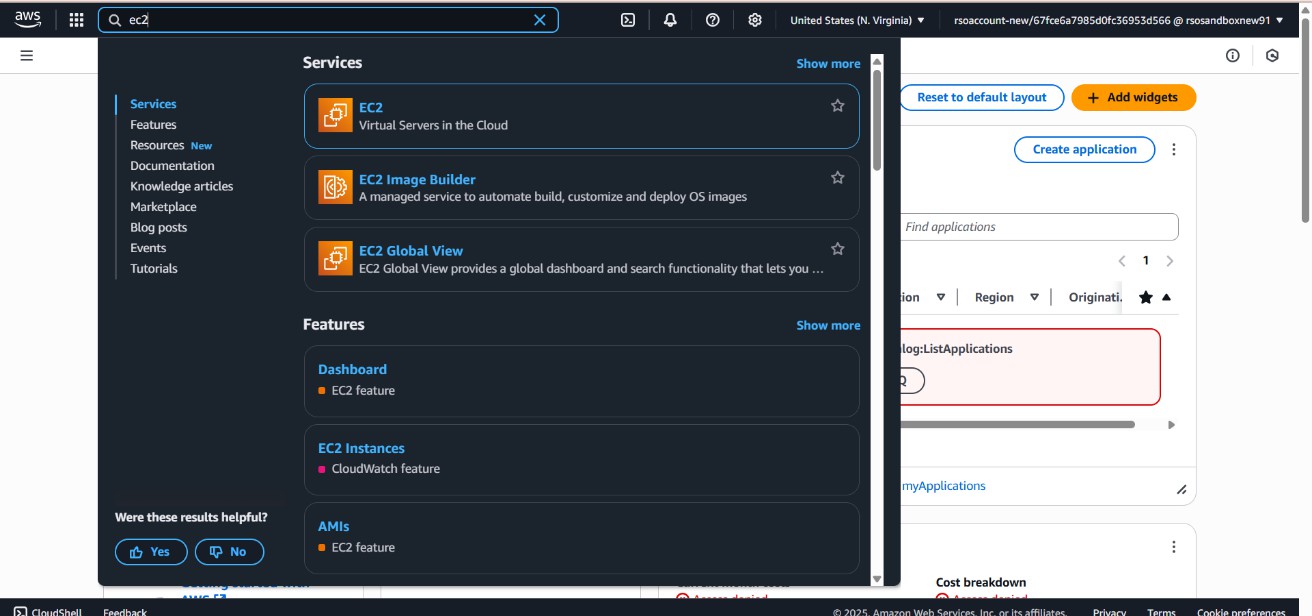
### Note: Load your Flask app and Html files into GitHub repository .

****

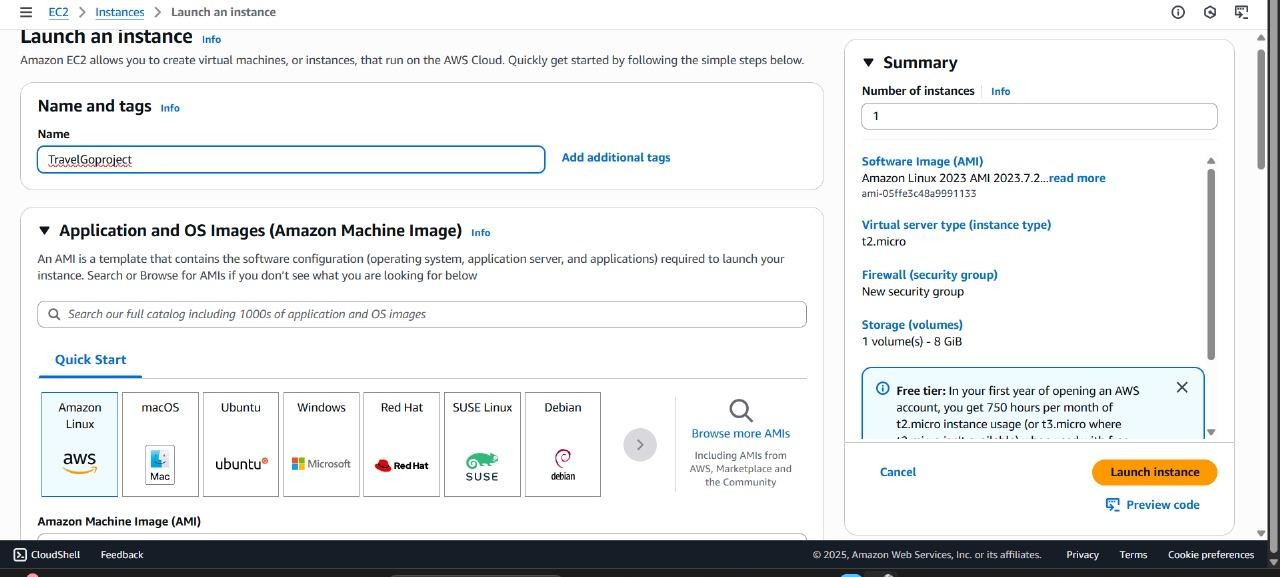
****

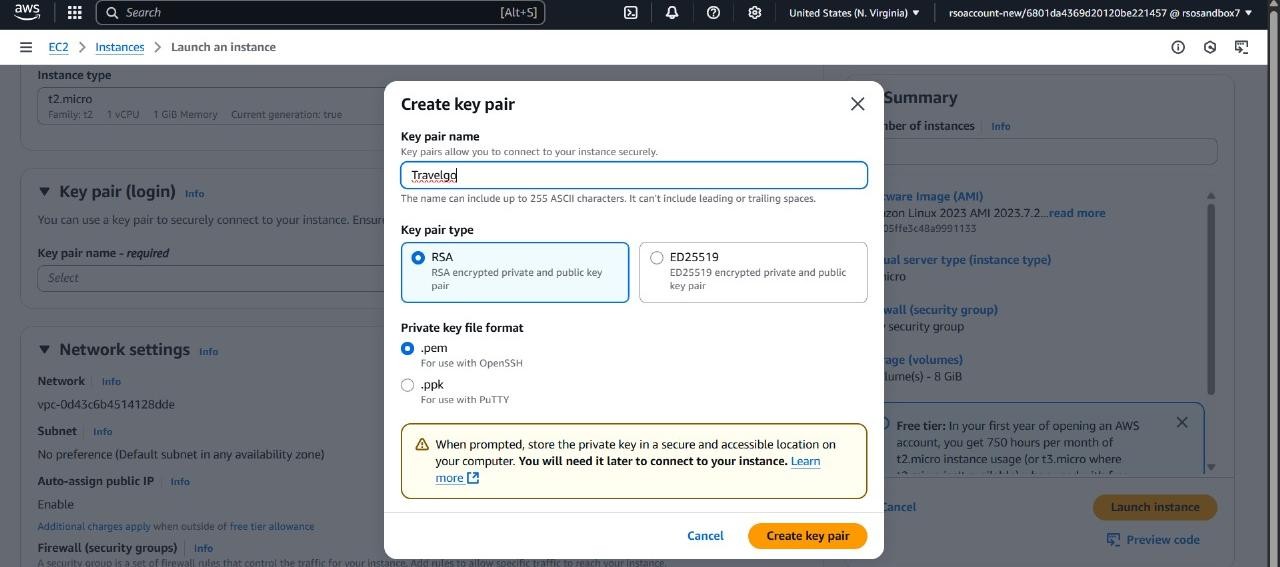
**Activity 6.1:** Launch an EC2 instance to host the Flask application.

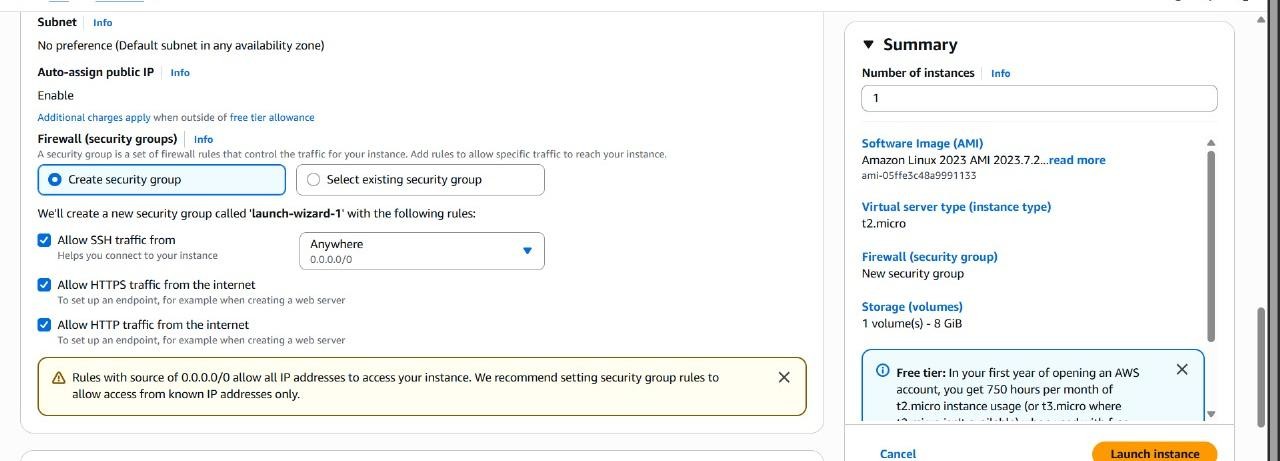
* In the AWS Console, navigate to EC2 and launch a new instance.



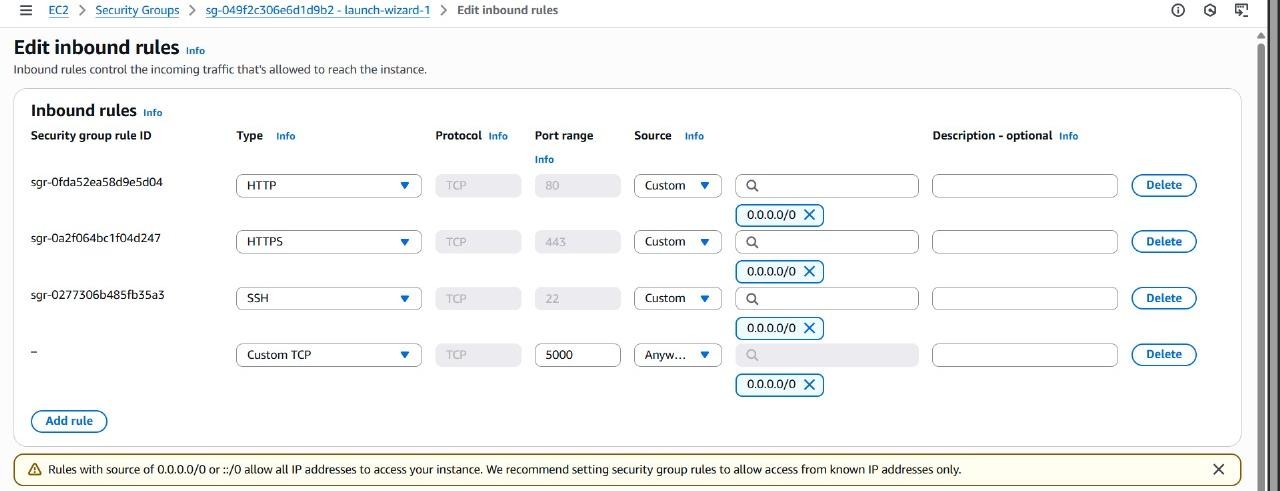


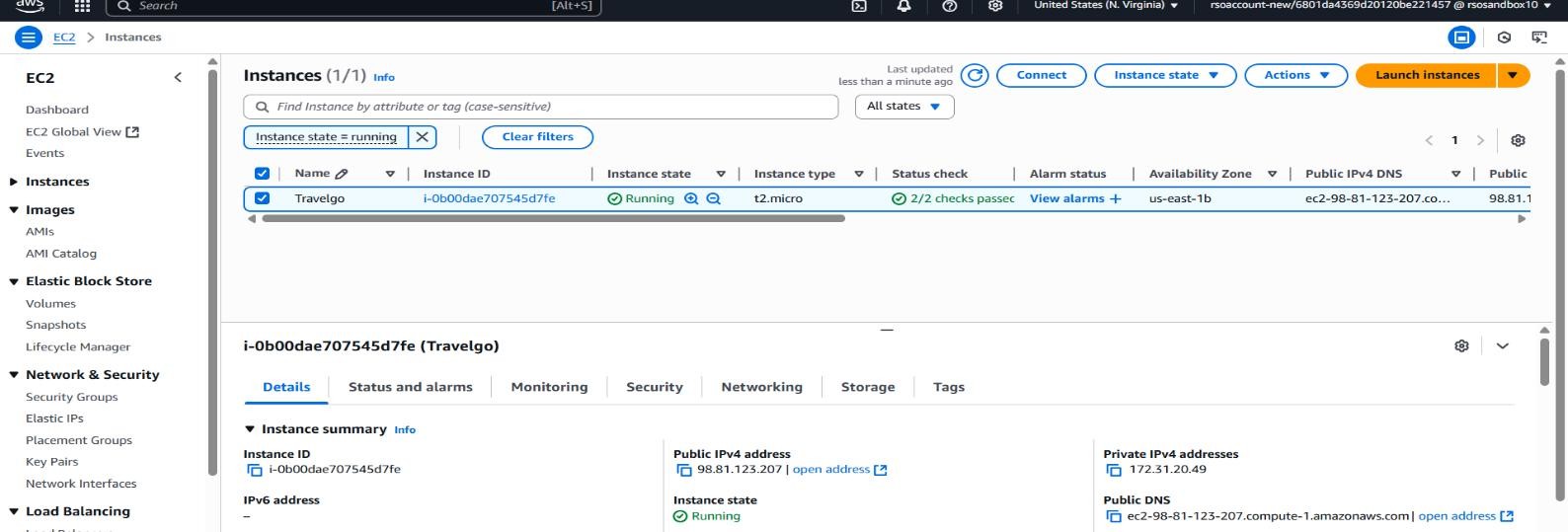
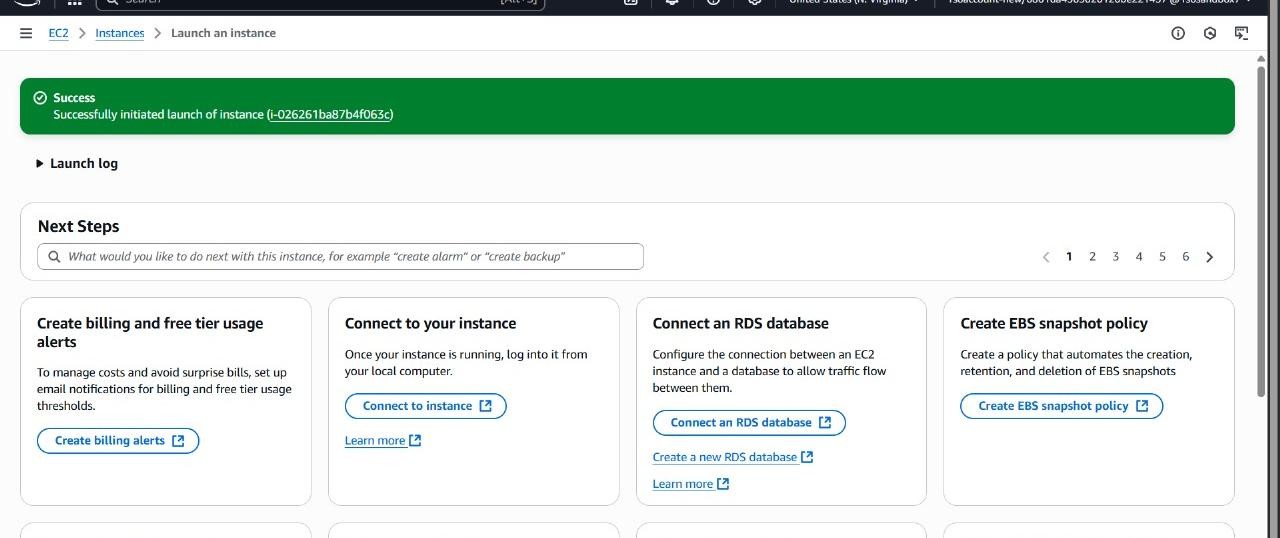




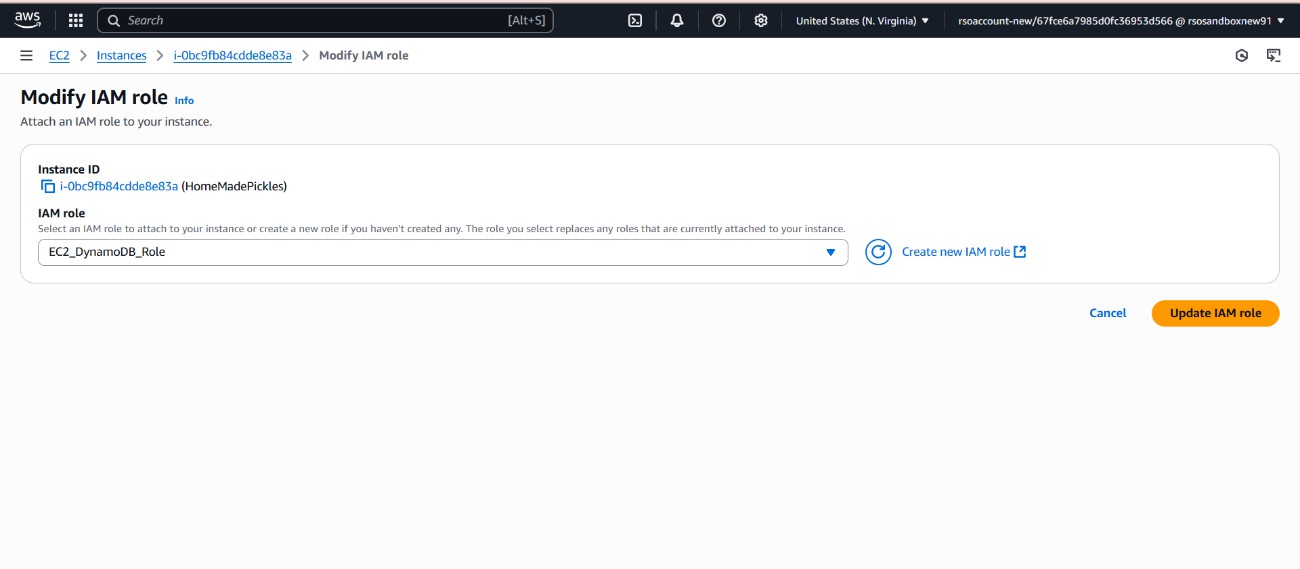


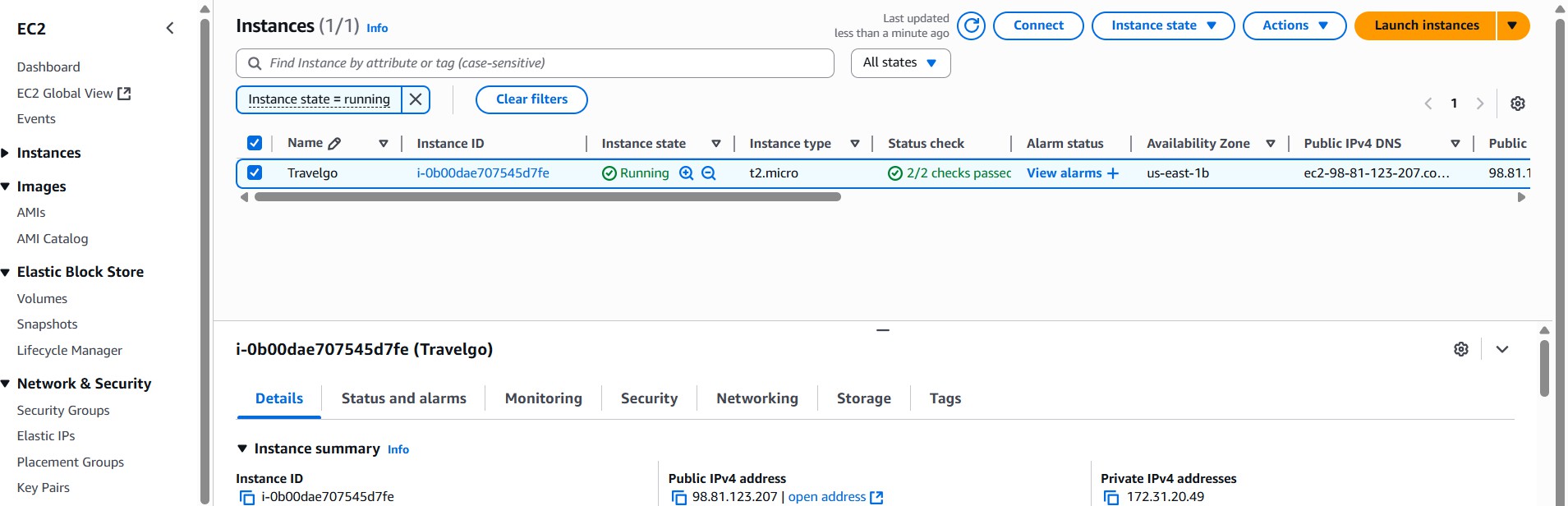


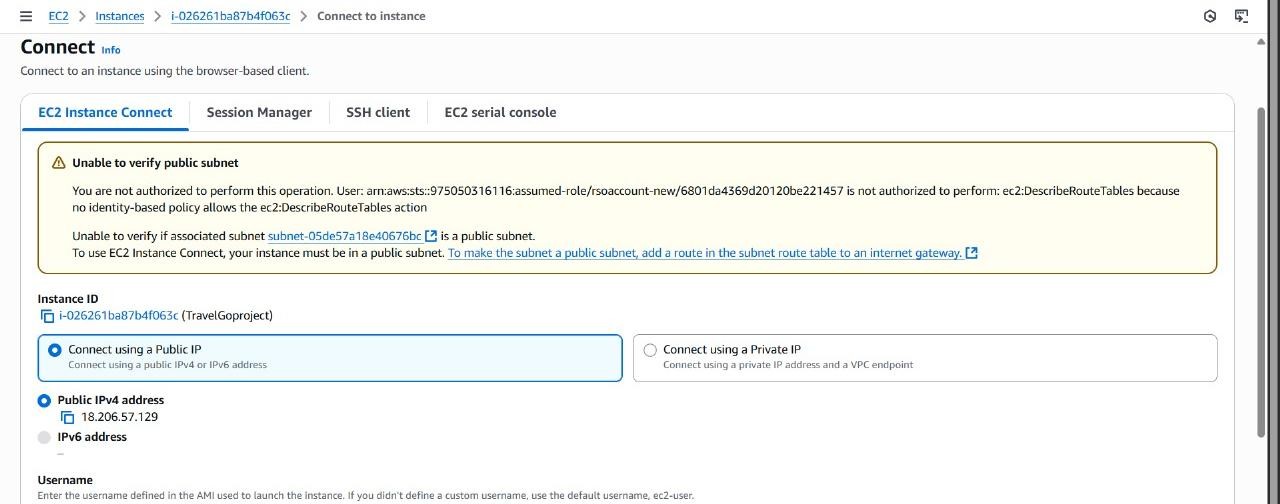




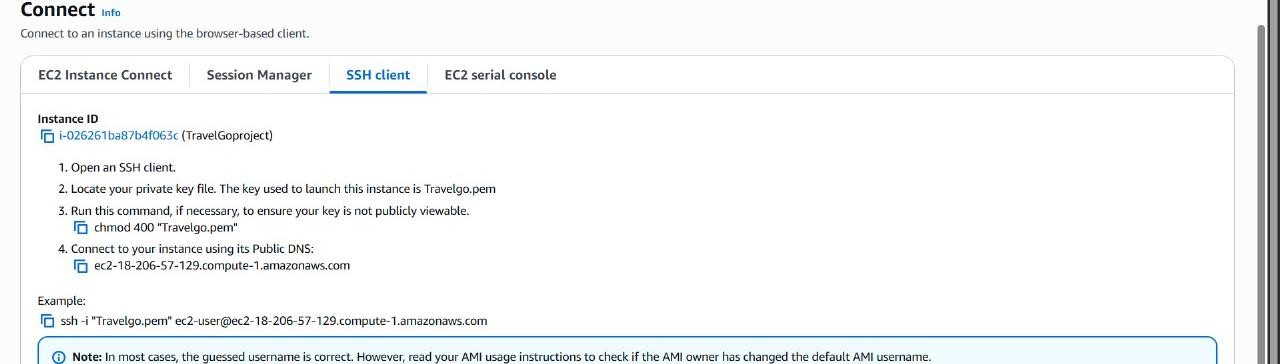












# Milestone 7: Deployment on EC2

**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/Sriram4207/Travelgo. This will download the Project to Ec2 instance .

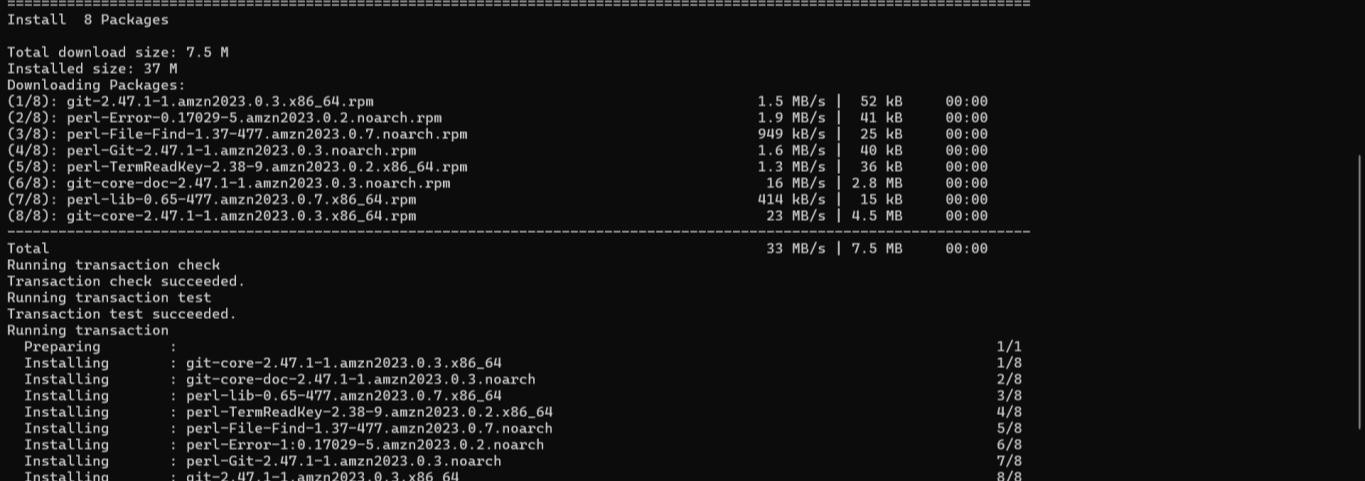
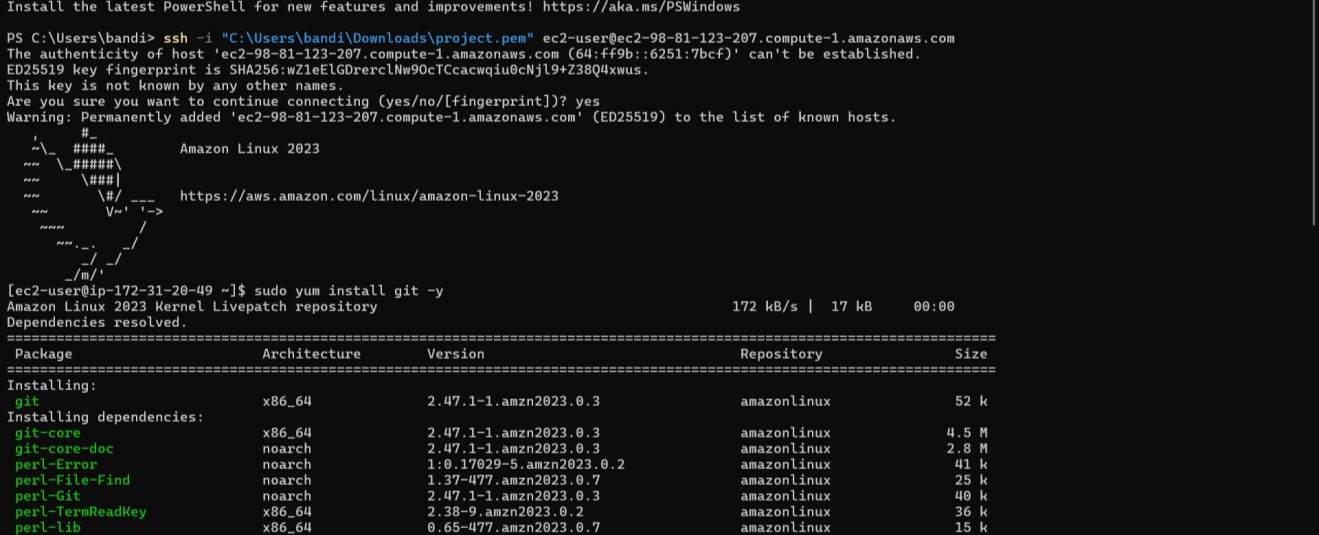
**To navigate to the project directory, run the following command:**

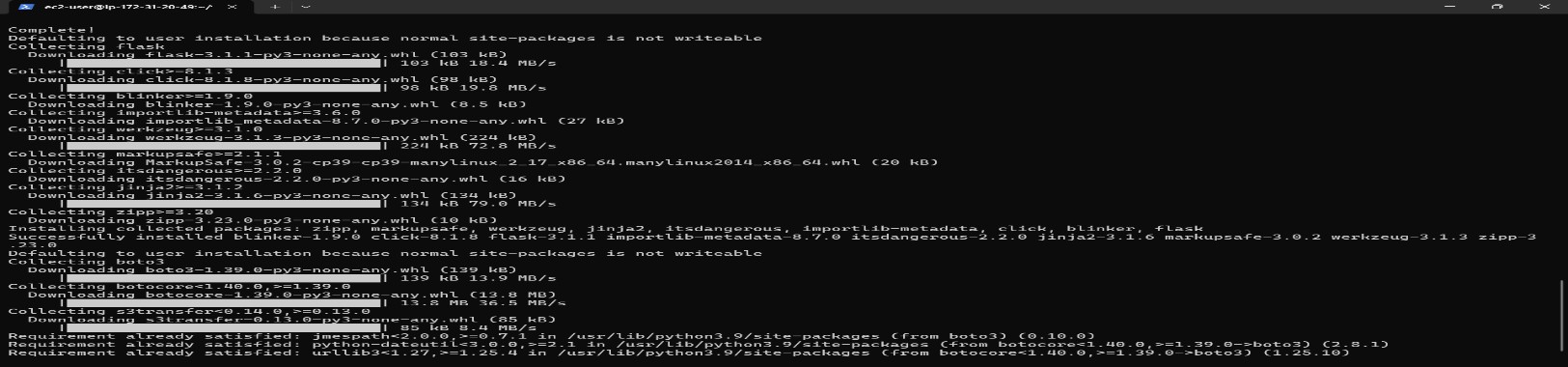
**cd InstantLibrary**

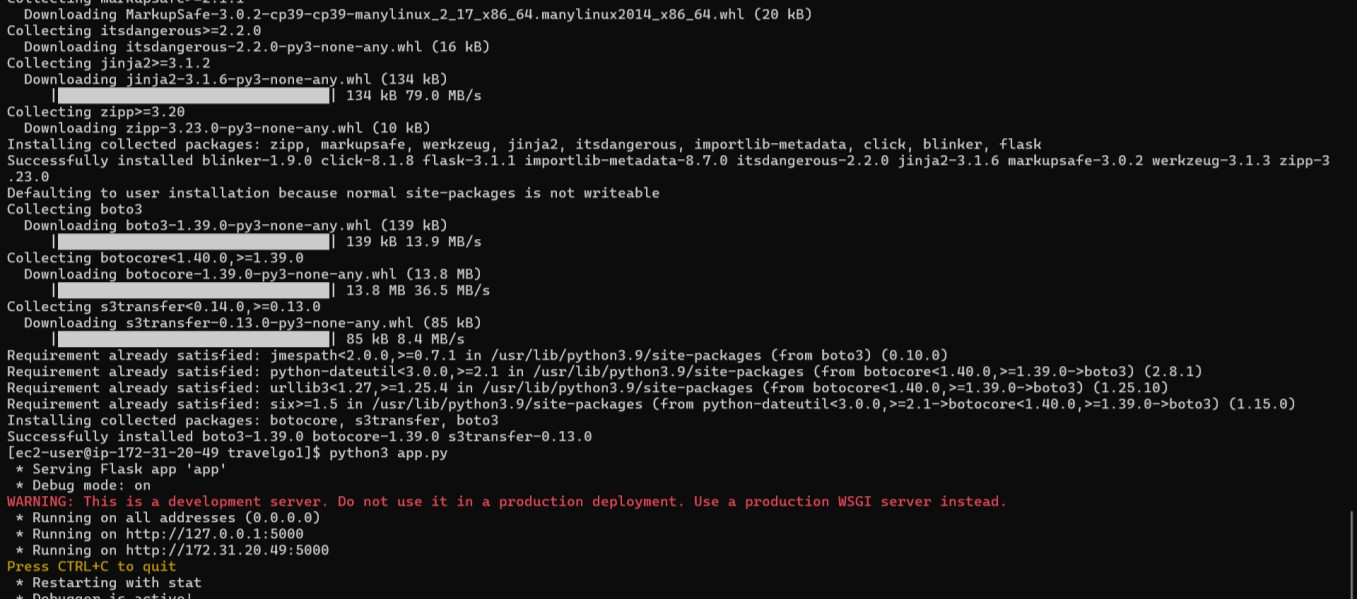
**Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:**

**Run the Flask Application**

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







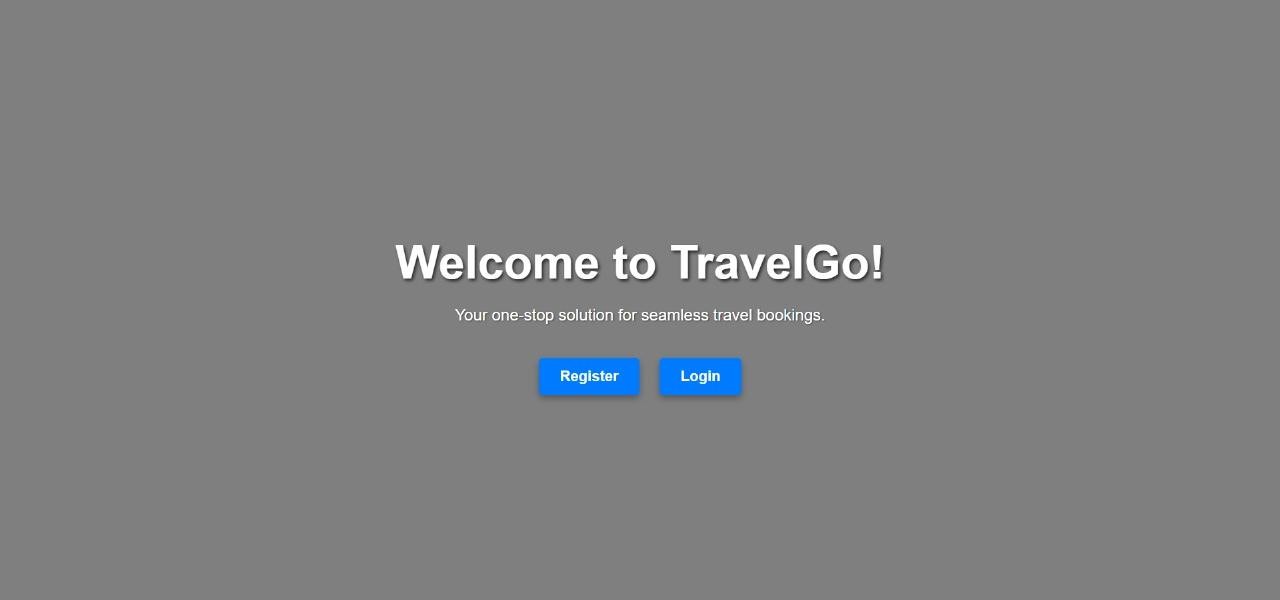
# Access the website through:

**Public IPs:** http://3.90.202.173:5000/

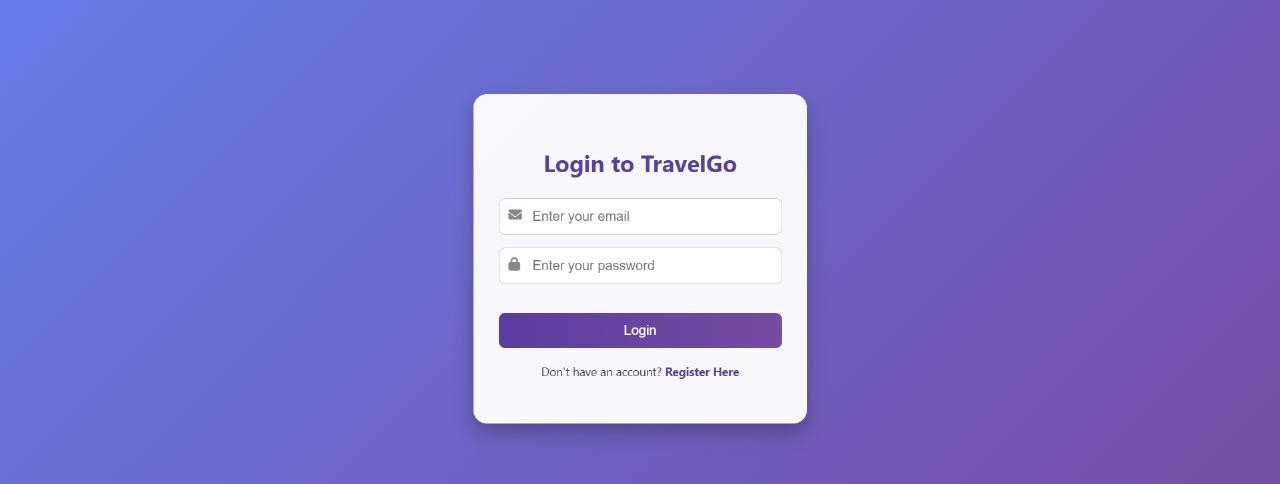
# Milestone 8. Testing and Deployment

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

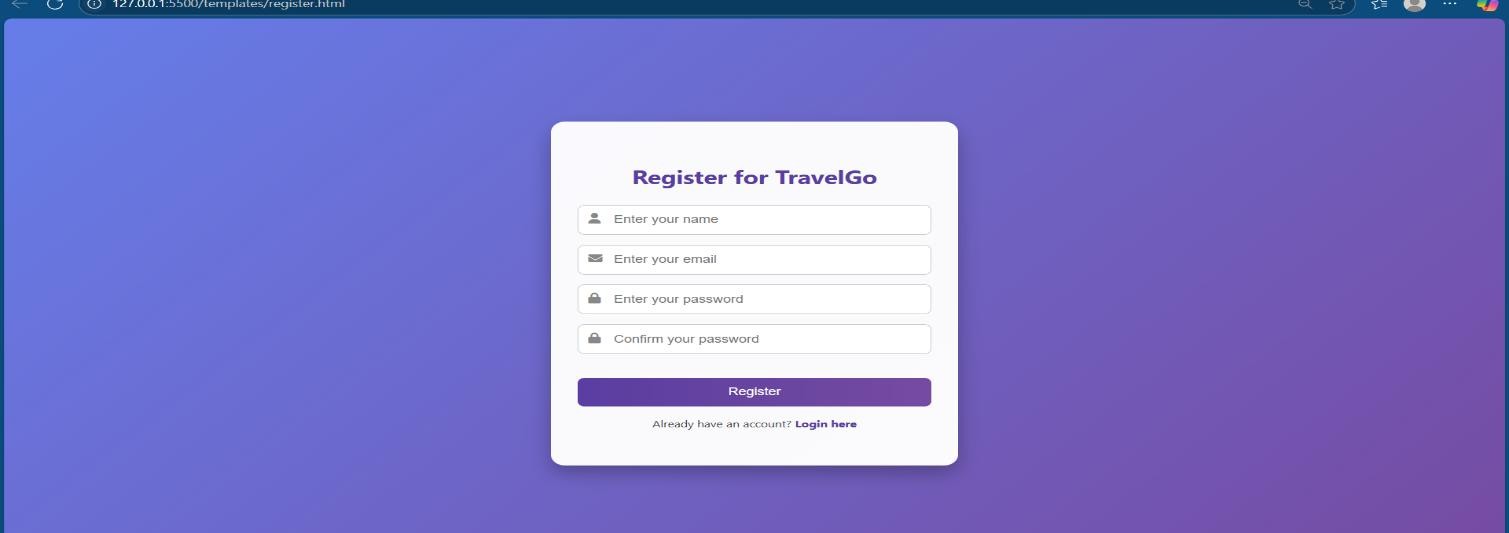
**HOME PAGE:**

****

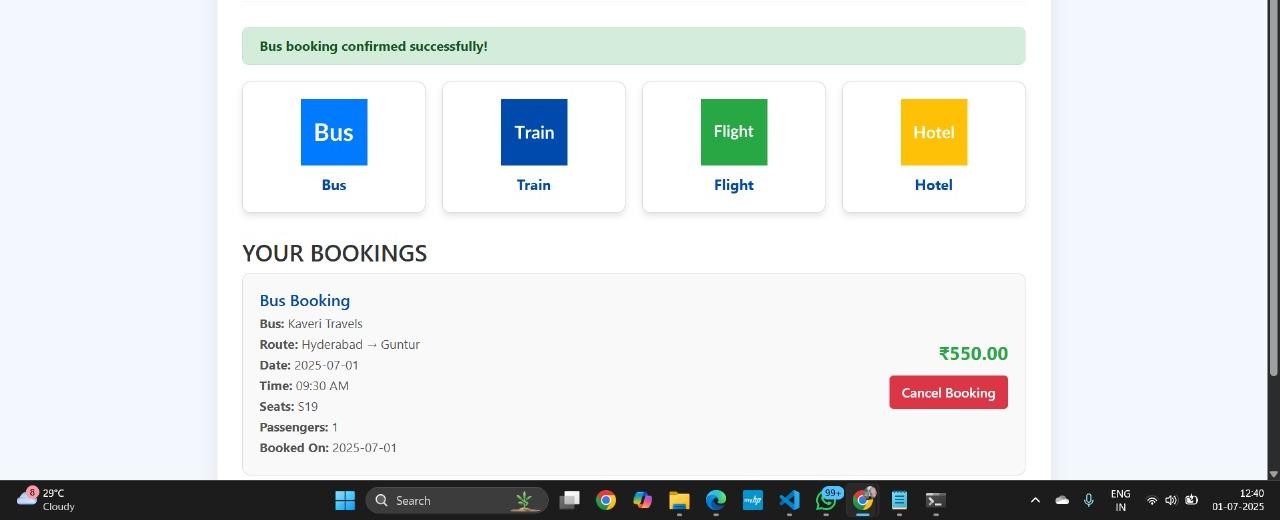
**LOGIN PAGE :**

****

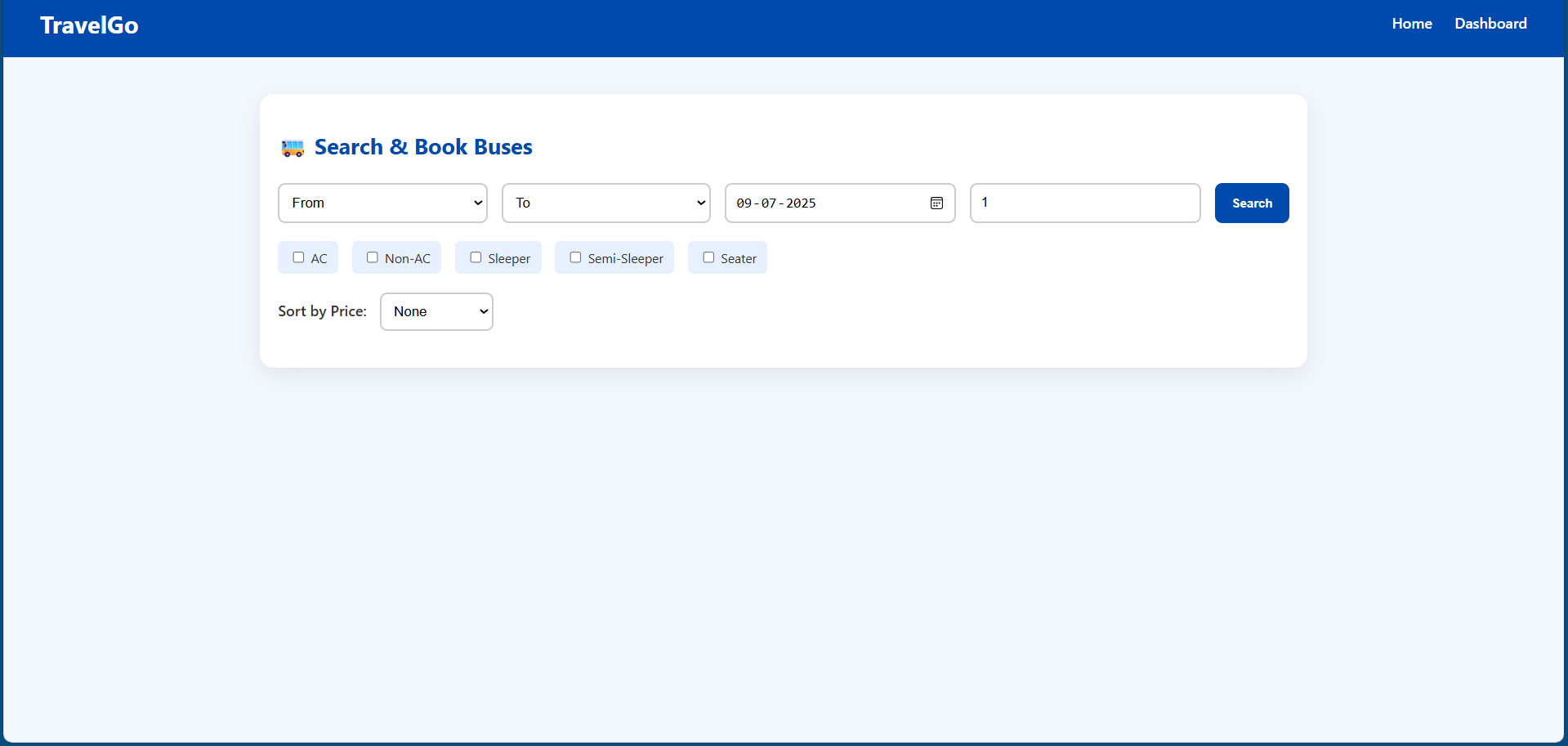
**SIGNUP PAGE :**

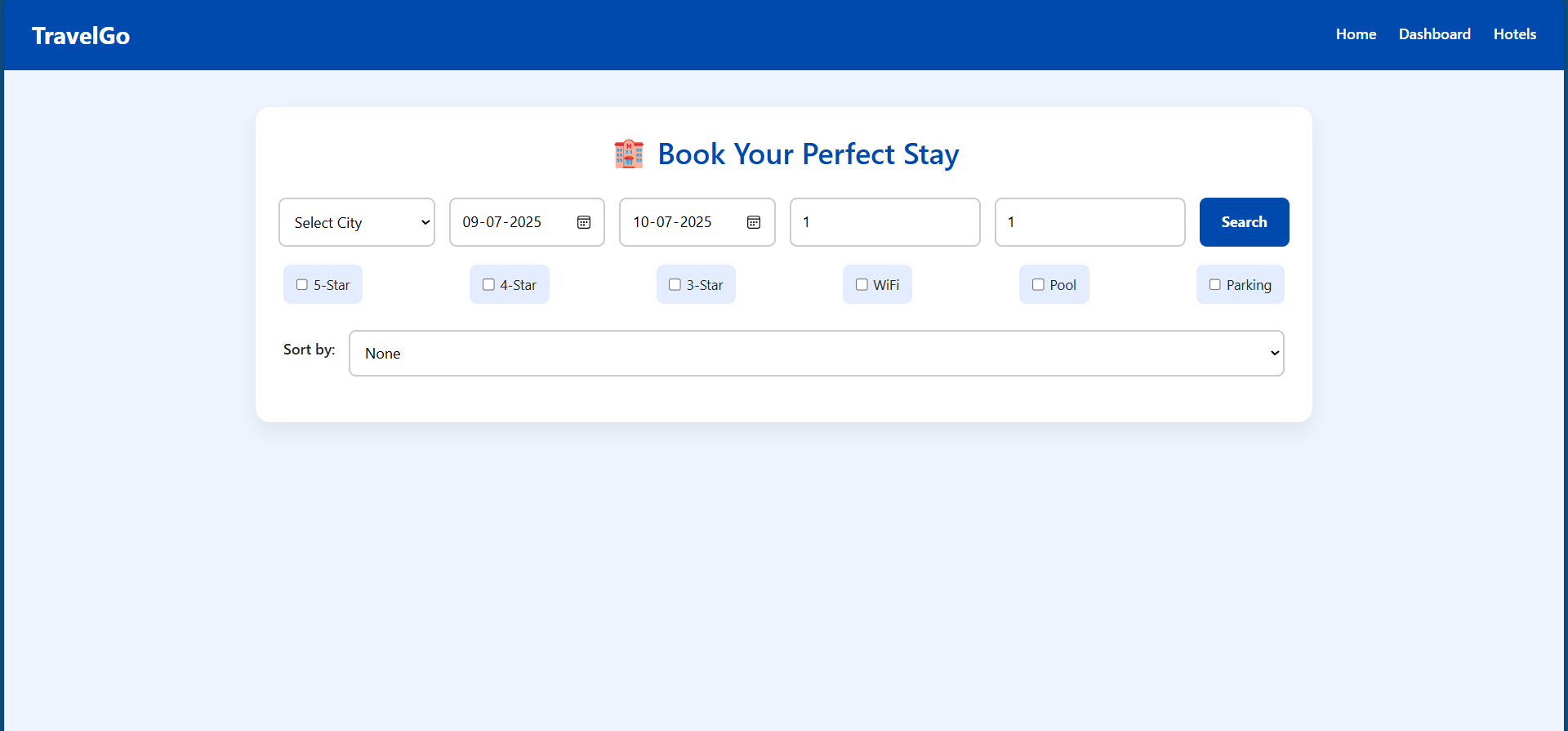
****

**Dashboard:**

****

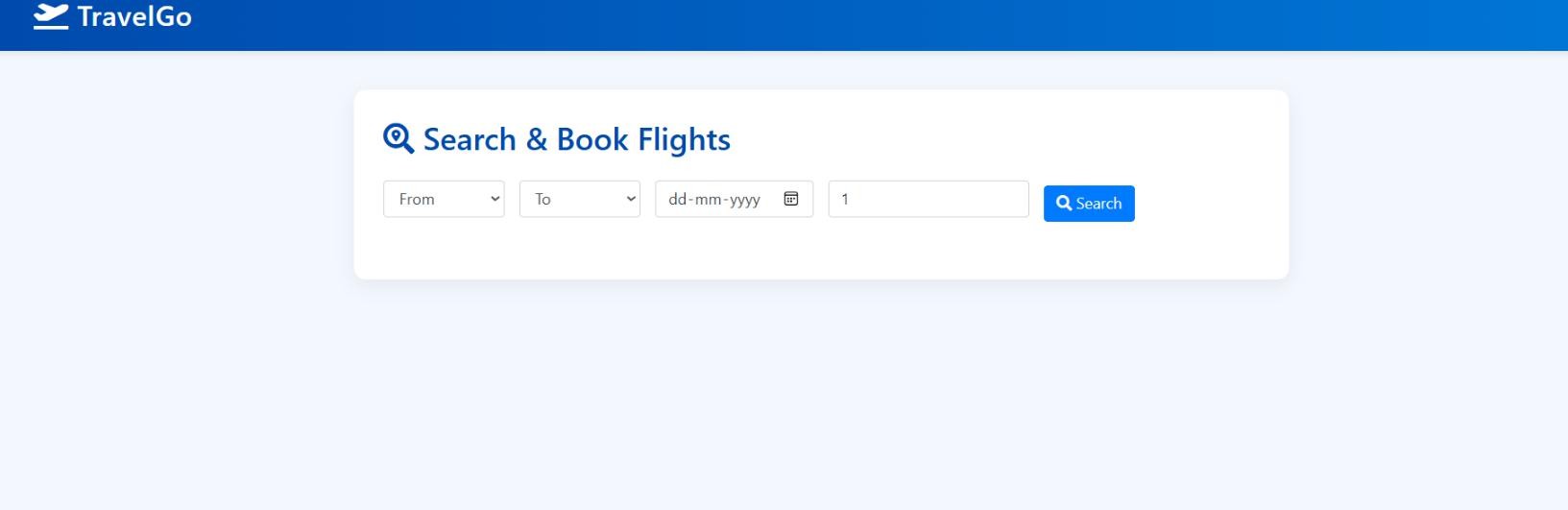
**BUS PAGE :**

****

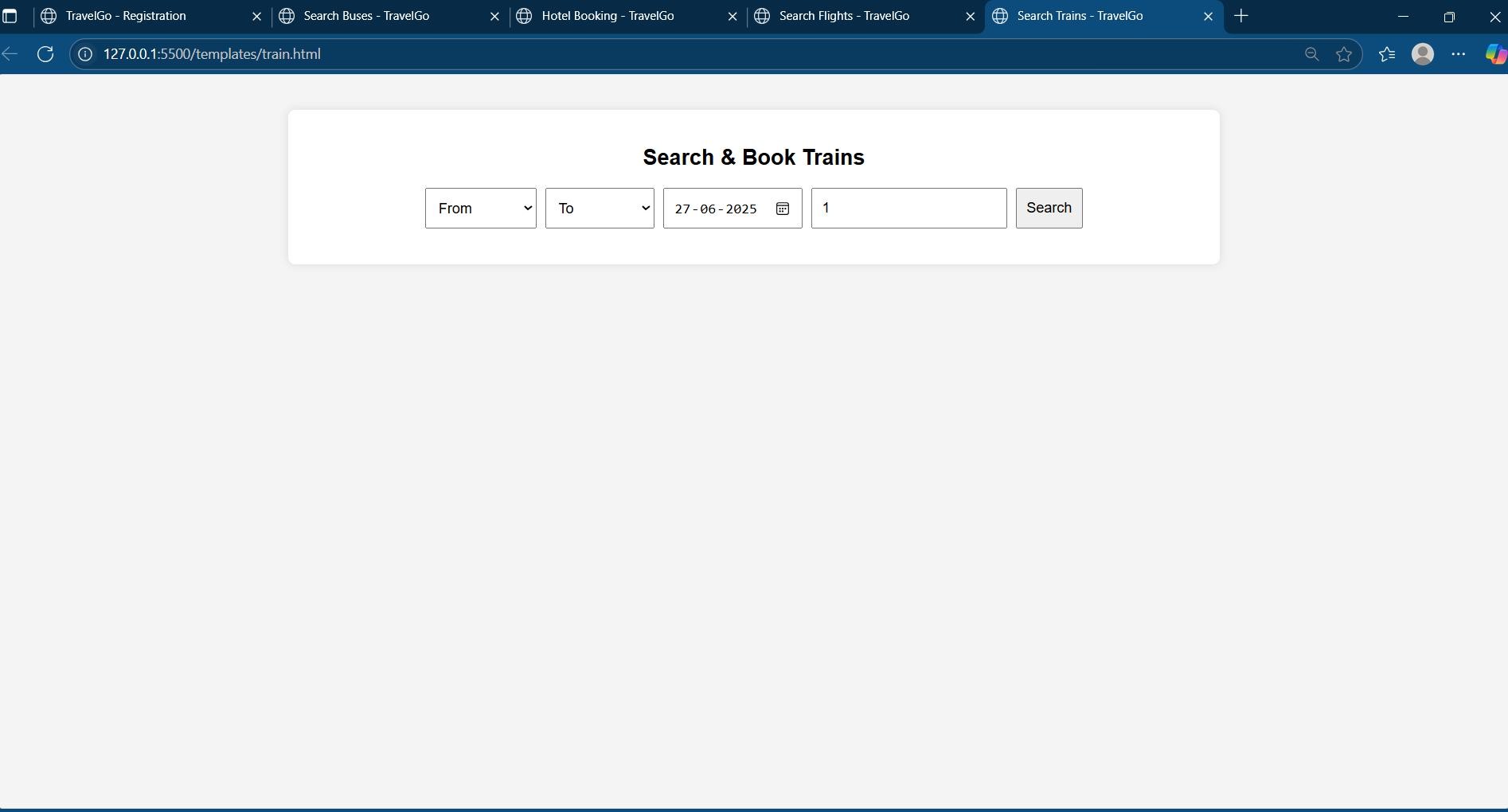
**HOTEL PAGE :**

**FLIGHT PAGE :**



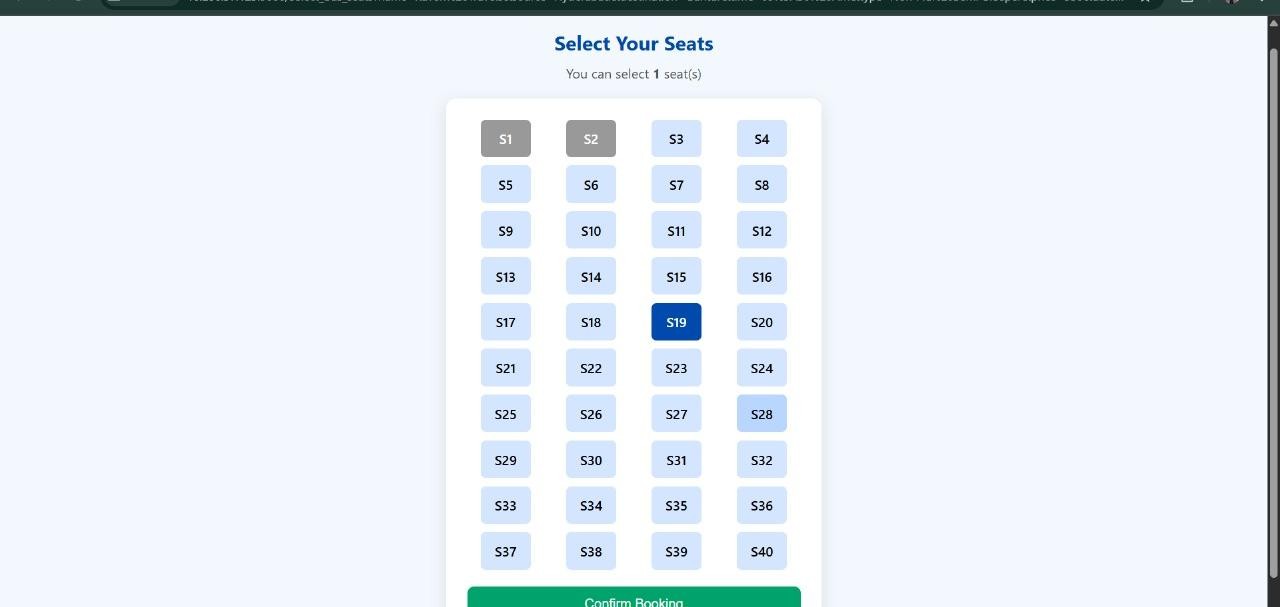


**TRAIN PAGE :**

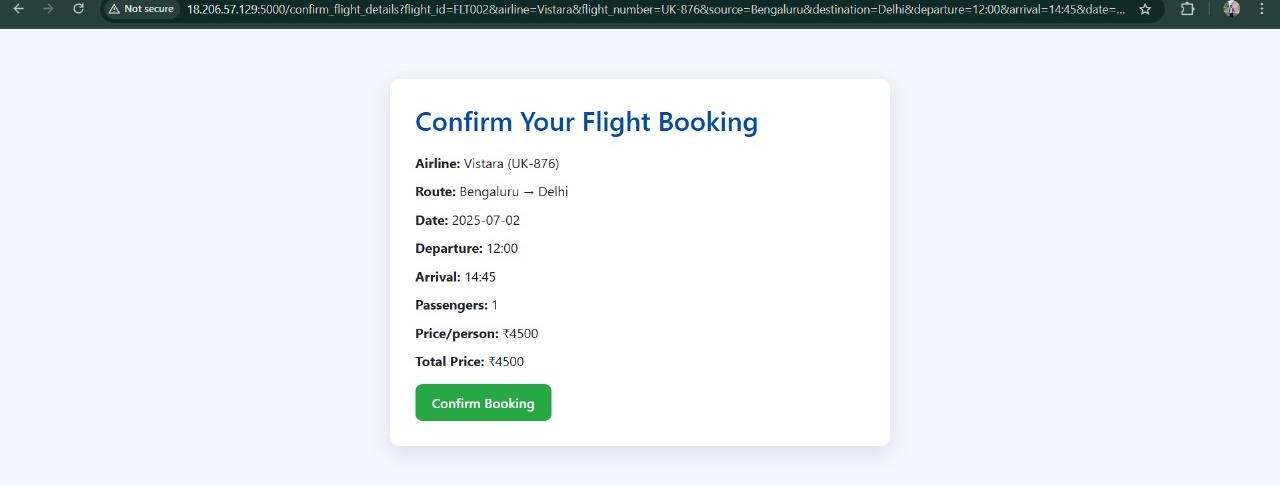
****

**SEAT SELECT PAGE :**



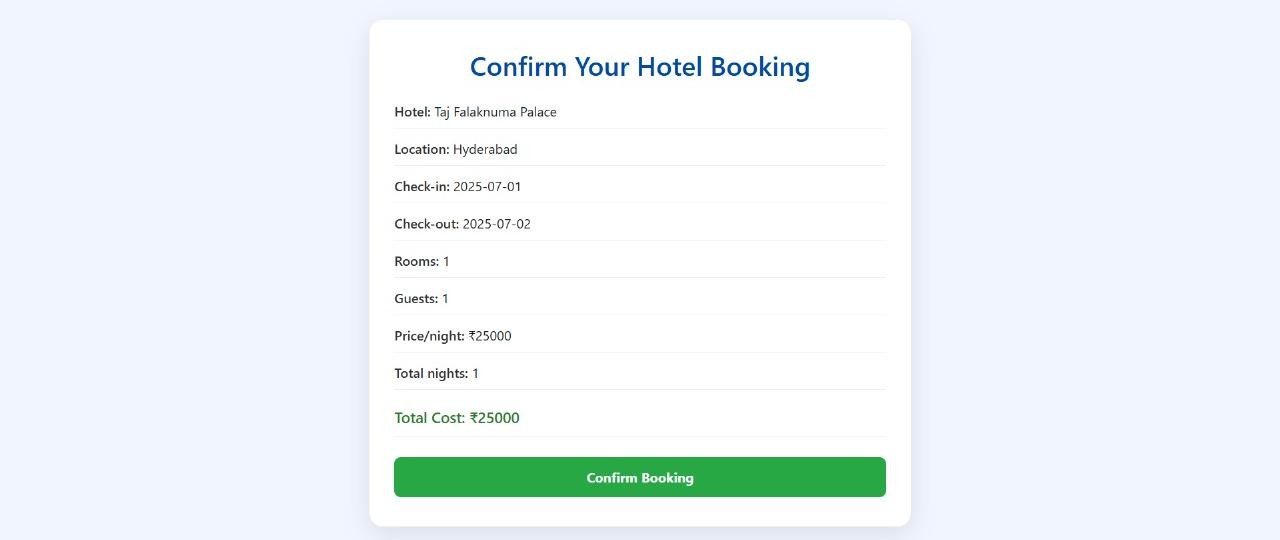


**FLIGHT BOOKING PAGE :**

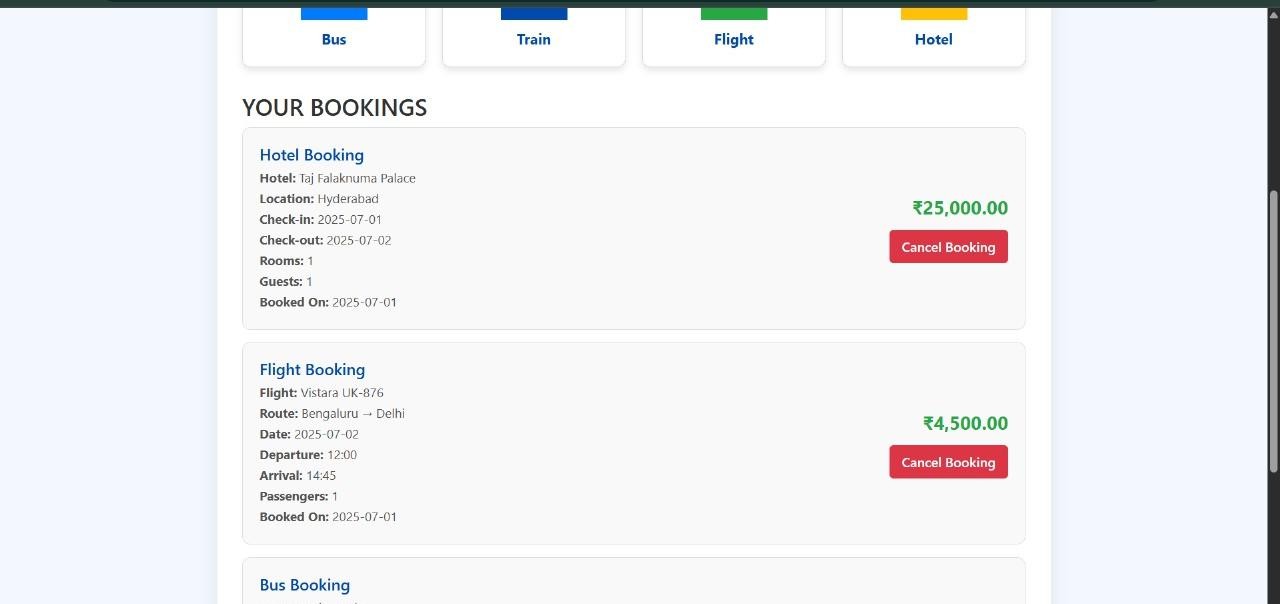
****

**HOTEL BOOKING PAGE :**





**BOOKINGS :**

****

**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………………………………………………