**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.

### 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.

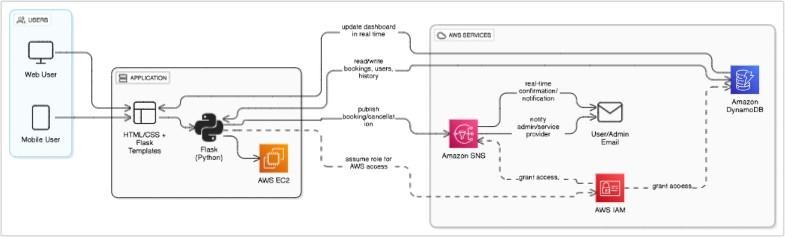
### Scenario 2: Real-Time Booking Confirmation with AWS SNS

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.

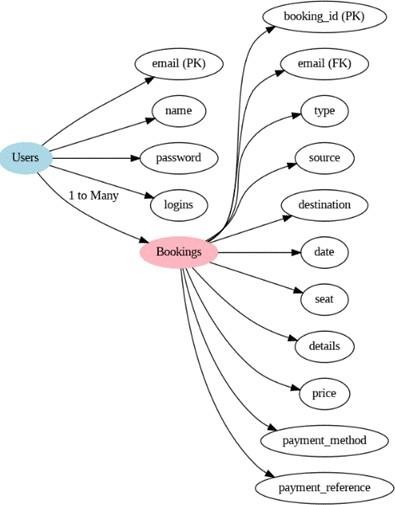
### Scenario 3: Dynamic Dashboard with Personal Travel History

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



Entity Relationship (ER)Diagram:



## Pre-requisites:

1. .**AWS Account Setup**: [AWS Account Setup](https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh)
2. **Understanding IAM**: [IAM Overview](https://youtu.be/gsgdAyGhV0o?si=3qg-bULgkD4LXNvR)
3. **Amazon EC2 Basics**: [EC2 Tutorial](https://youtu.be/8TlukLu11Yo?si=MUj0nEAOESRhHUIz)
4. **DynamoDB Basics**: [DynamoDB Introduction](https://docs.aws.amazon.com/dynamodb)
5. **SNS Overview**: [SNS Documentation](https://docs.aws.amazon.com/sns)
6. **Git Version Control**: [Git Documentation](https://git-scm.com/doc)

## Project WorkFlow:

### AWS Account Setup and Login

**Activity 1.1:** Set up an AWS account if not already done**. Activity 1.2:** Log in to the AWS Management Console

### DynamoDB Database Creation and Setup

**Activity 2.1**: Create a DynamoDB Table.

**Activity 2.2**: Configure Attributes for User Data and Book Requests.

### SNS Notification Setup

**Activity 3.1**: Create SNS topics for book request notifications.

**Activity 3.2**: Subscribe users and library staff to SNS email notifications.

### Backend Development and Application Setup

**Activity 4.1**:Develop the Backend Using Flask.

**Activity 4.2**: Integrate AWS Services Using boto3.

### IAM Role Setup

**Activity 5.1**: Create IAM Role

**Activity 5.2**: Attach Policies

### 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.

1. **Deployment on EC2 Activity 7.1**:Upload Flask Files **Activity 7.2**: Run the Flask App

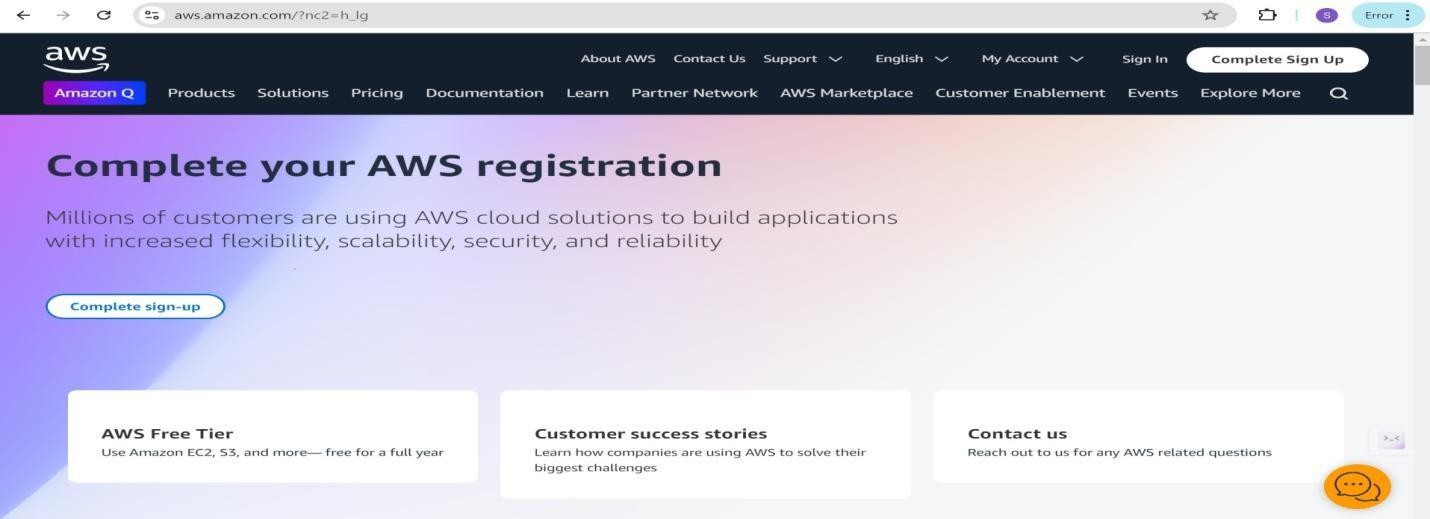
### Testing and Deployment

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

# Milestone 1: AWS Account Setup and Login

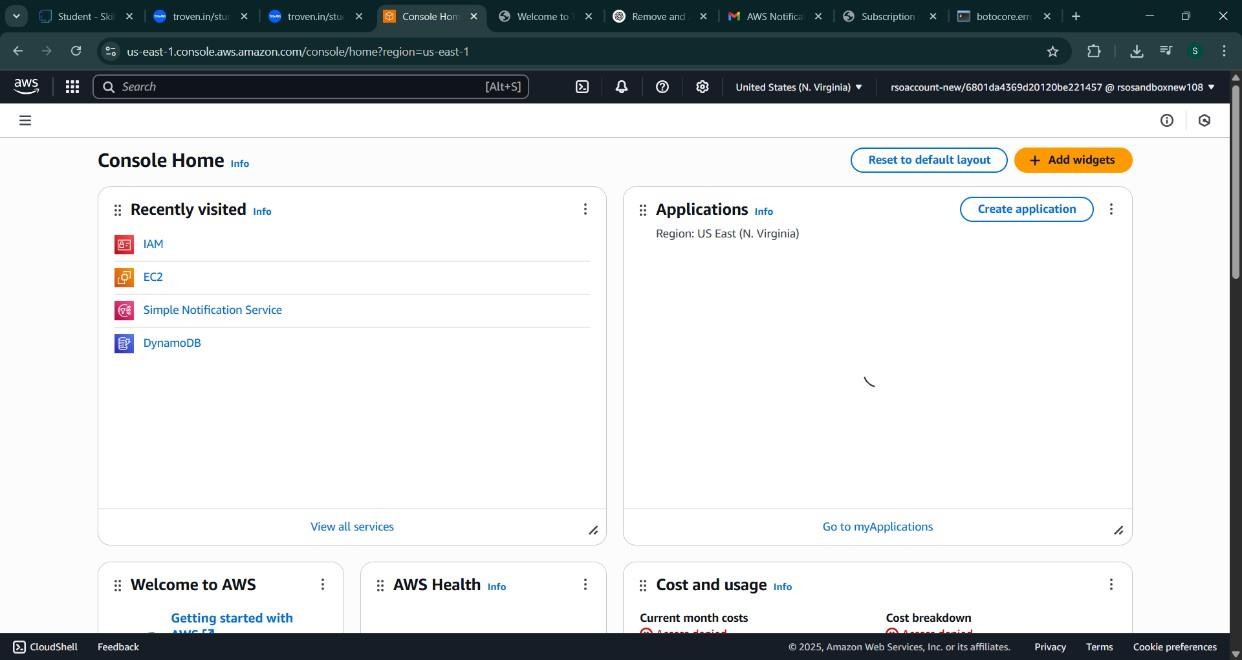
### Activity 1.1: Set up an AWS account if not already done.

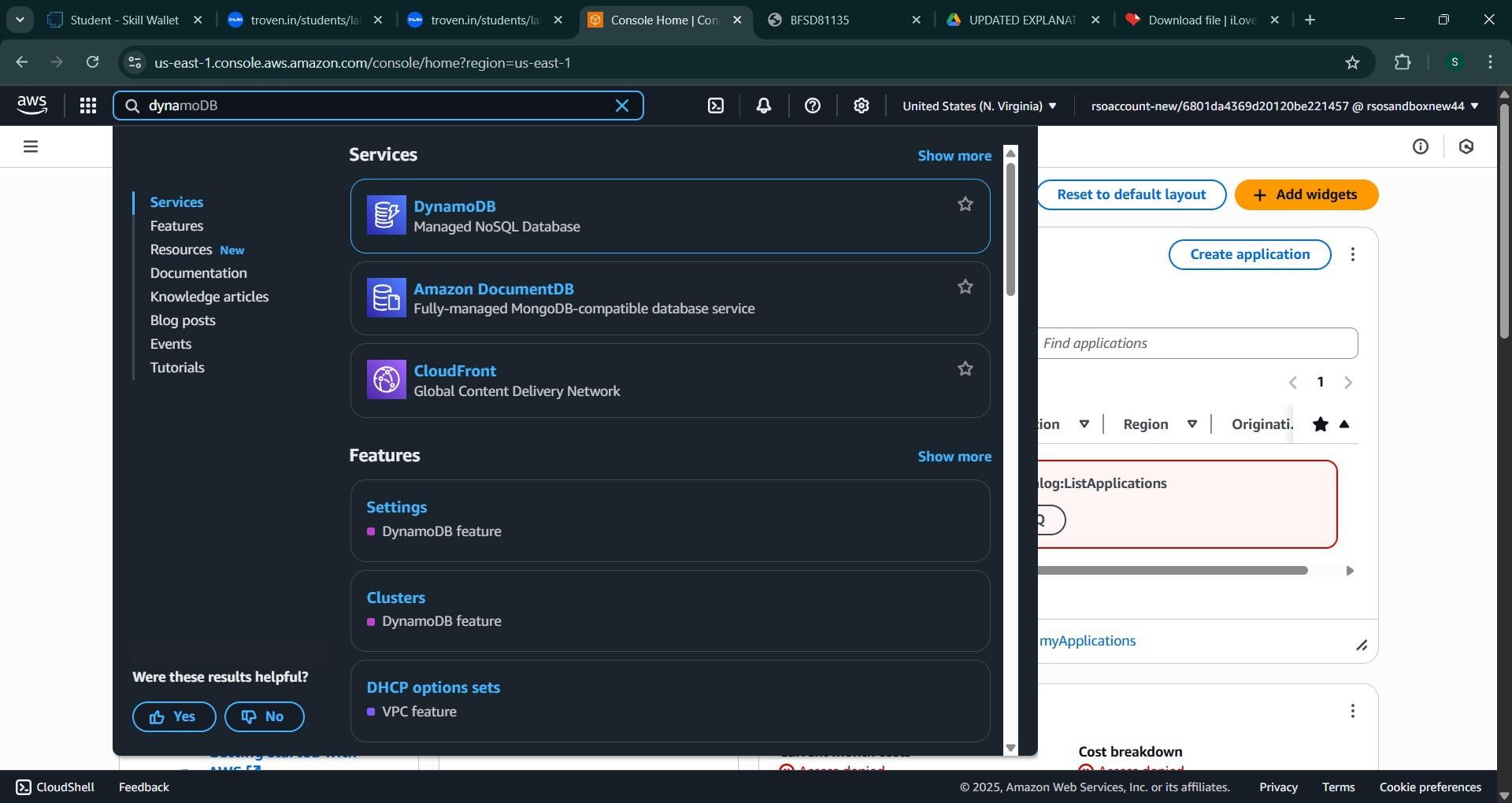
* + - Sign up for an AWS account and configure billing settings.



### Activity 1.2: Log in to the AWS Management Console

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

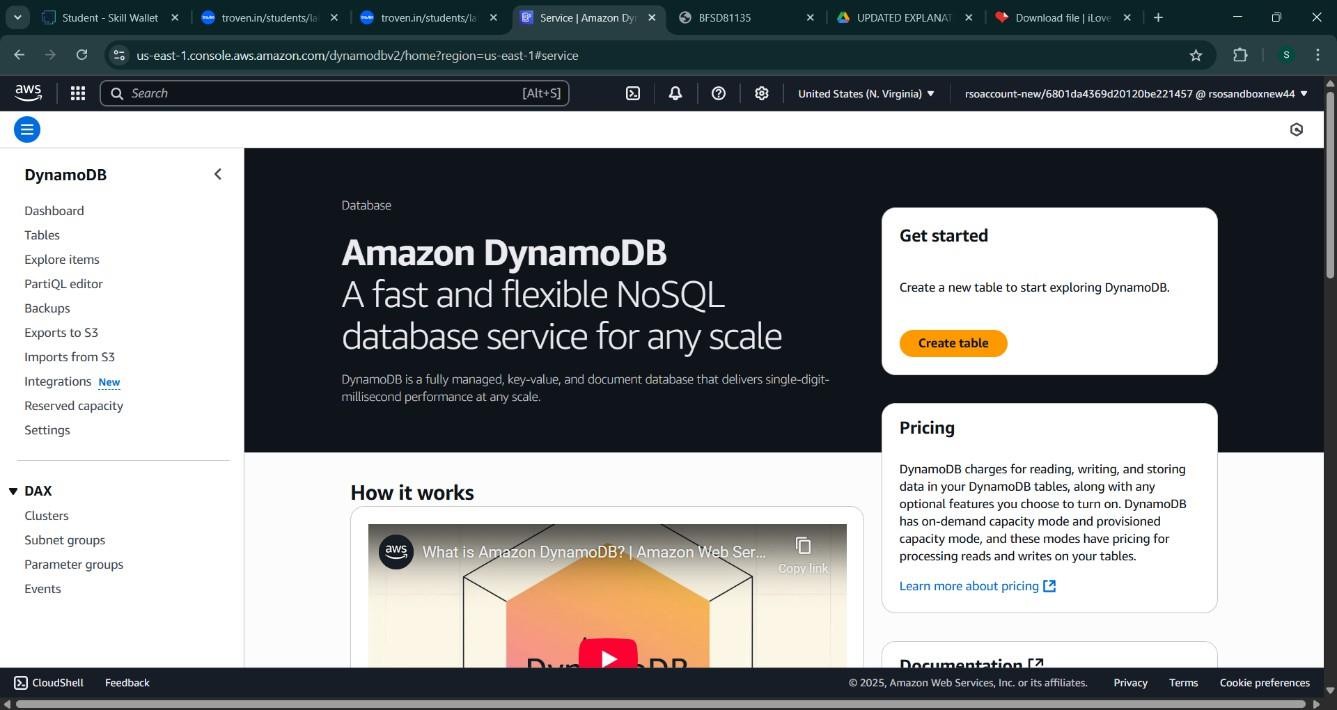




# Milestone 2: DynamoDB Database Creation and Setup

### Activity 2.1:Navigate to the DynamoDB

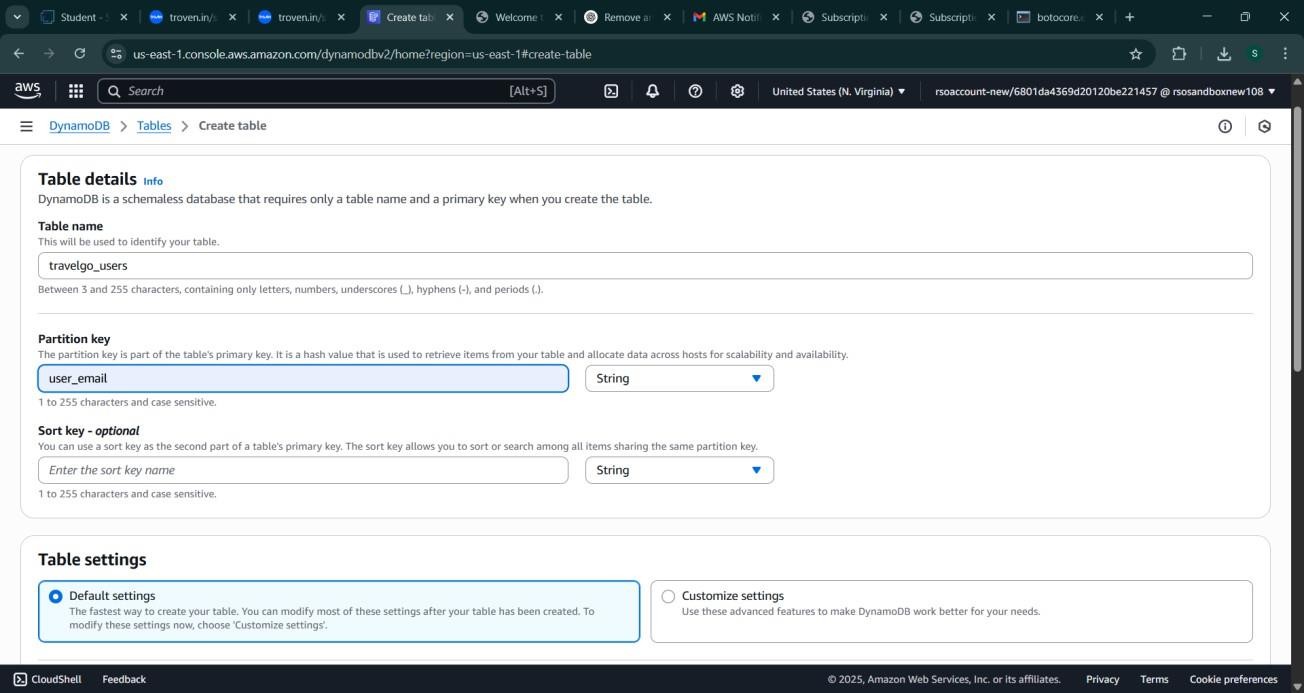
* + - In the AWS Console, navigate to DynamoDB and click on create tables.

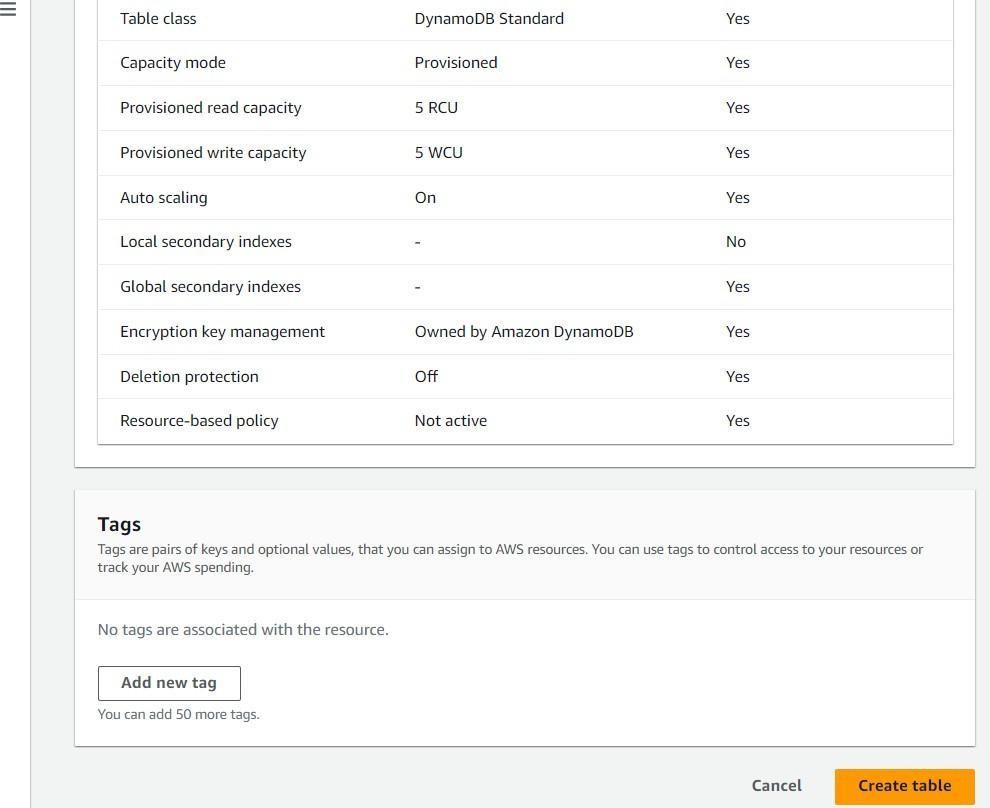
1. 

○

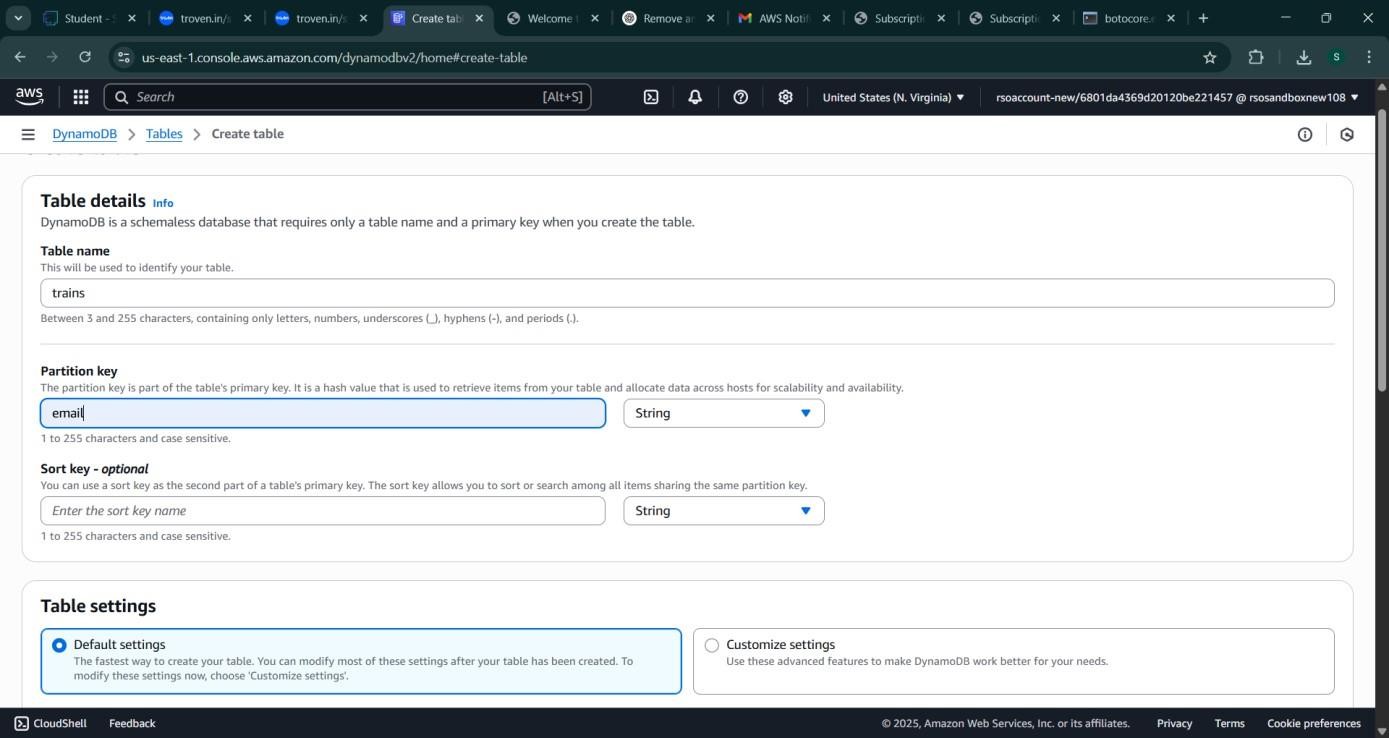
### Activity 2.2:Create a DynamoDB table for storing registration details and book requests.

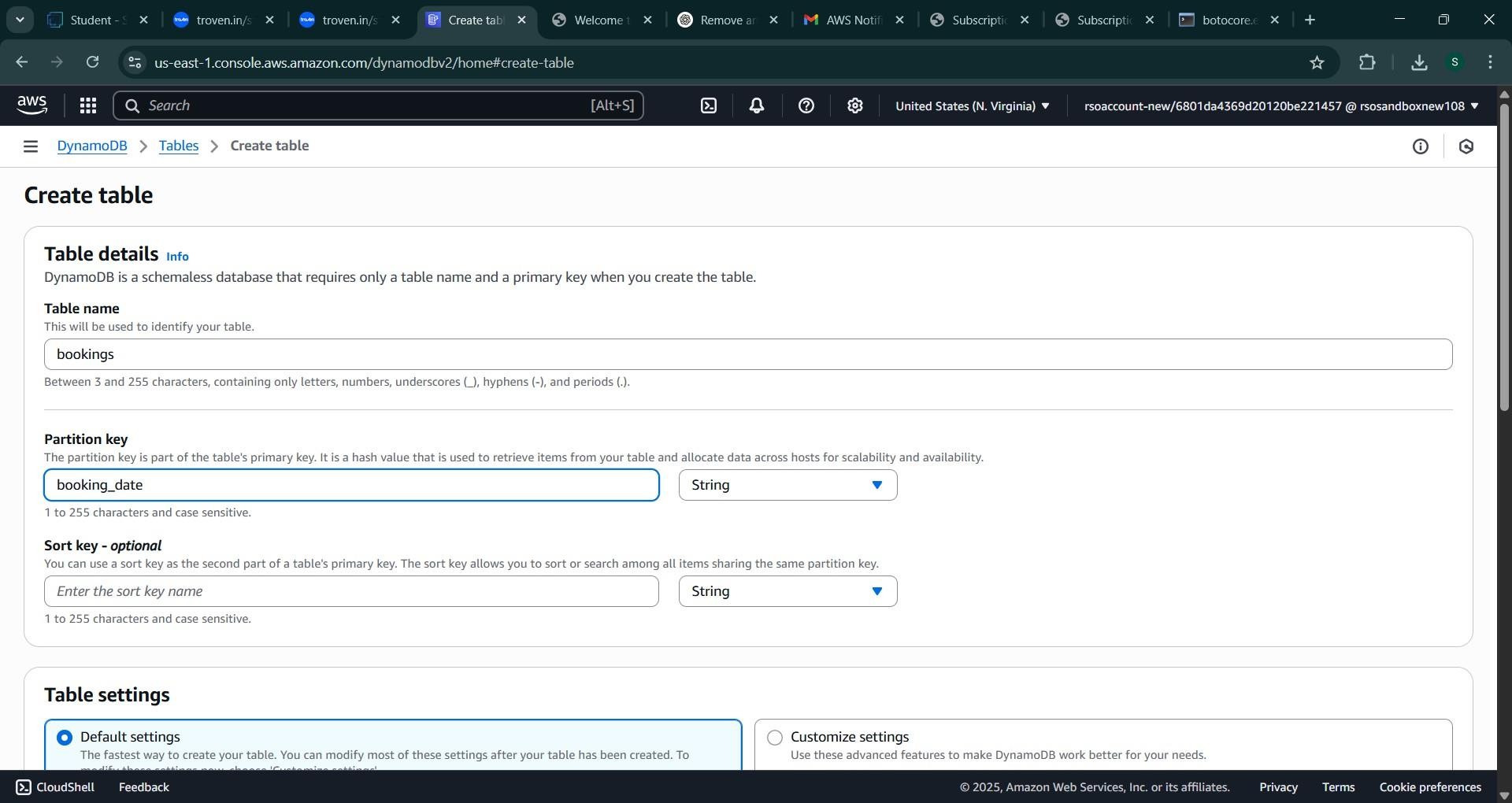
* + - Create Users table with partition key “user\_email” with type String and click on create tables.

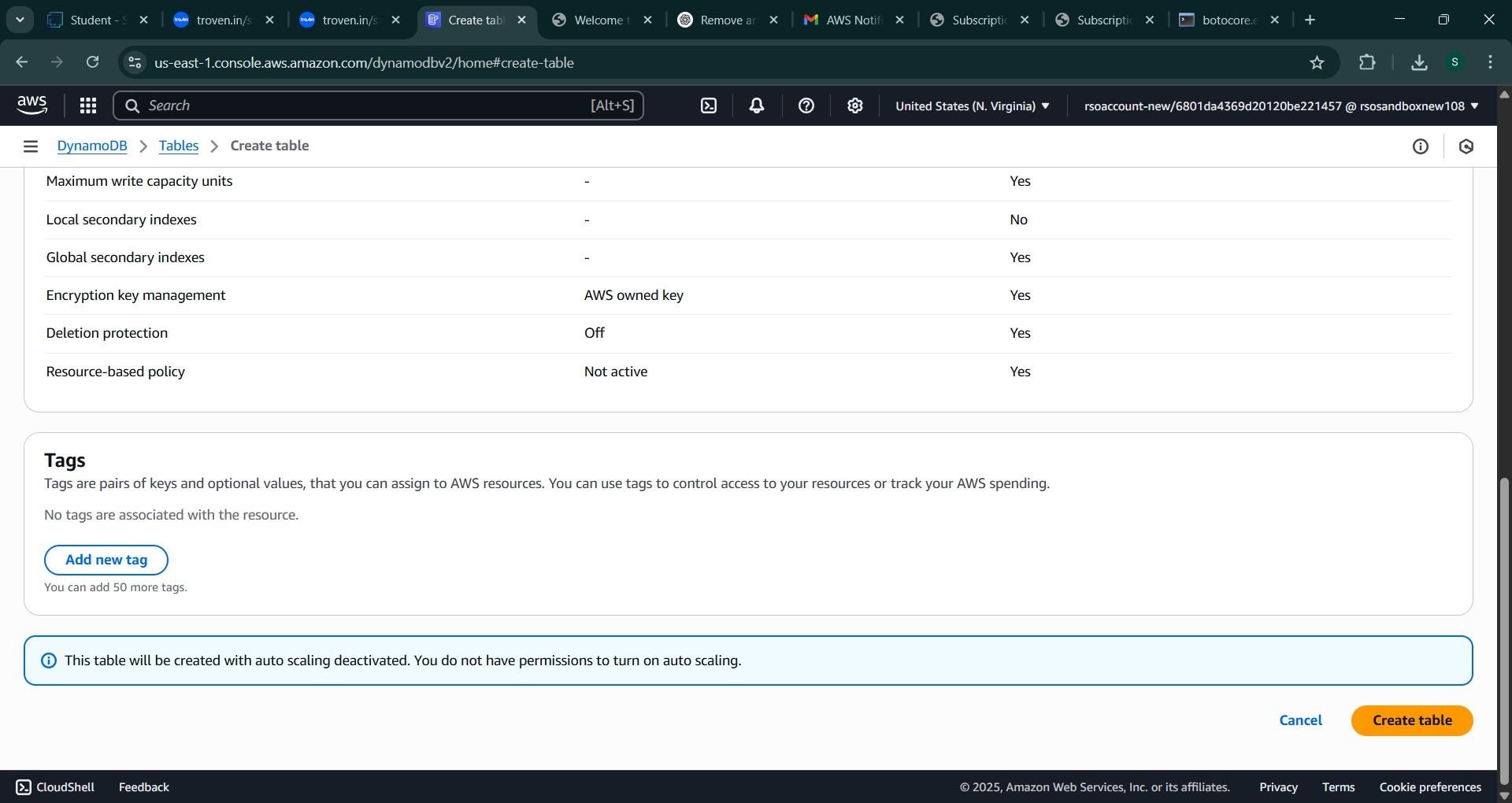
****



* Follow the same steps to create a requests table with user\_email as the primary key for book requests data.

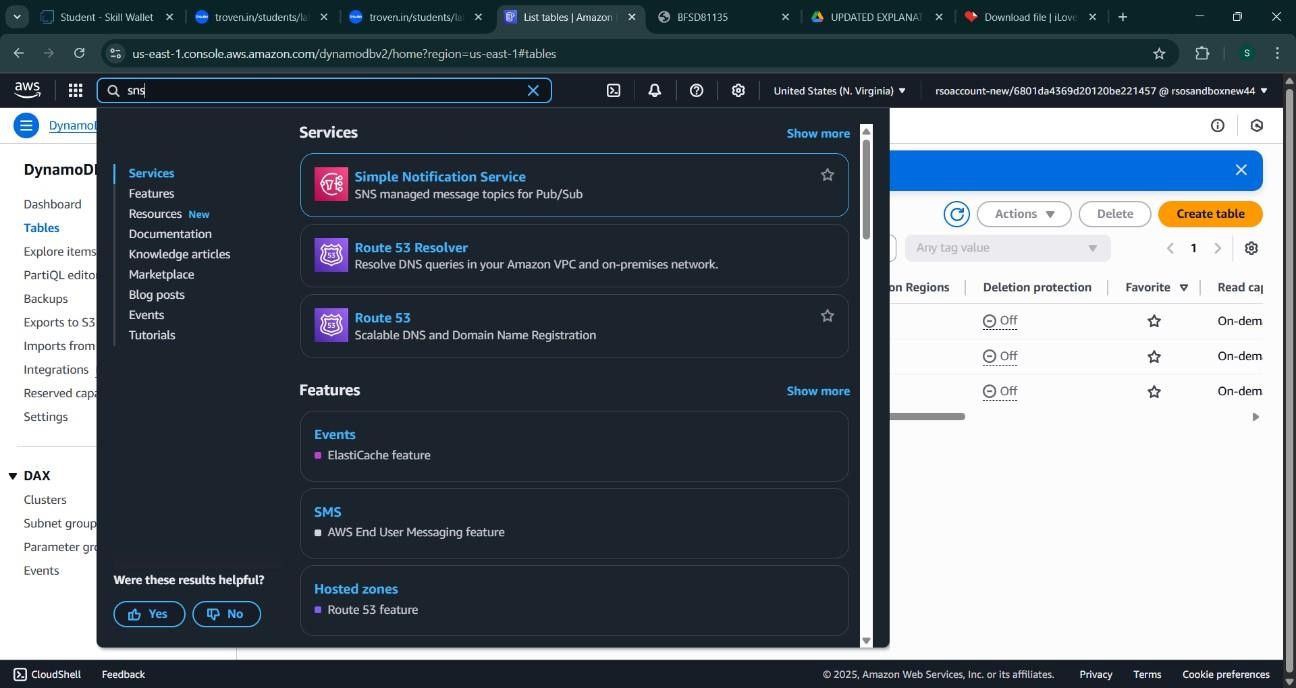


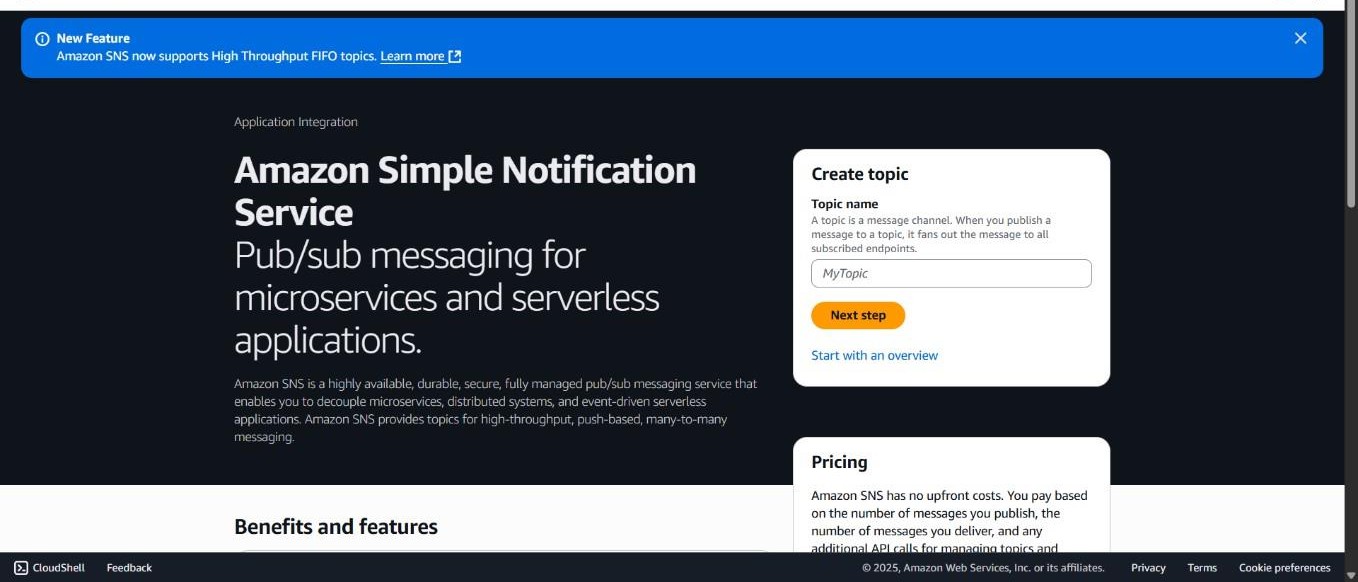


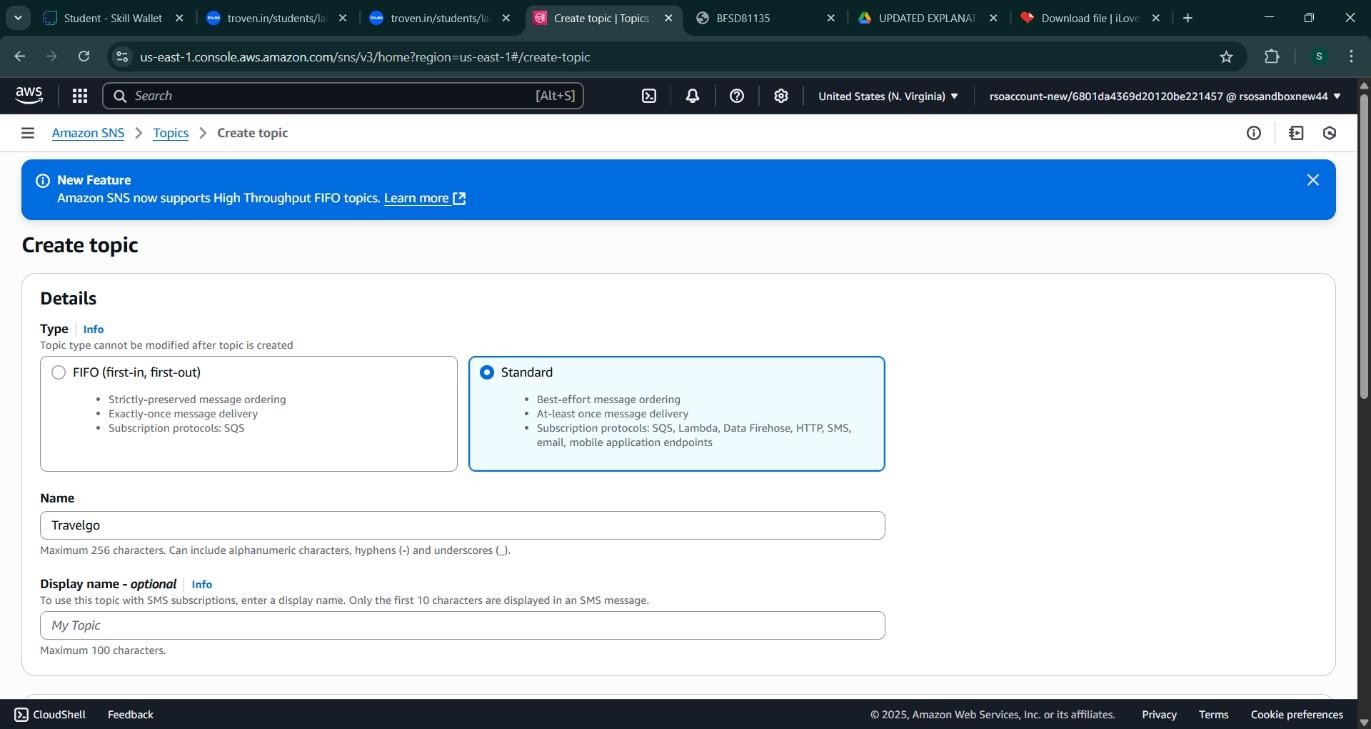


## Milestone 3: SNS Notification Setup

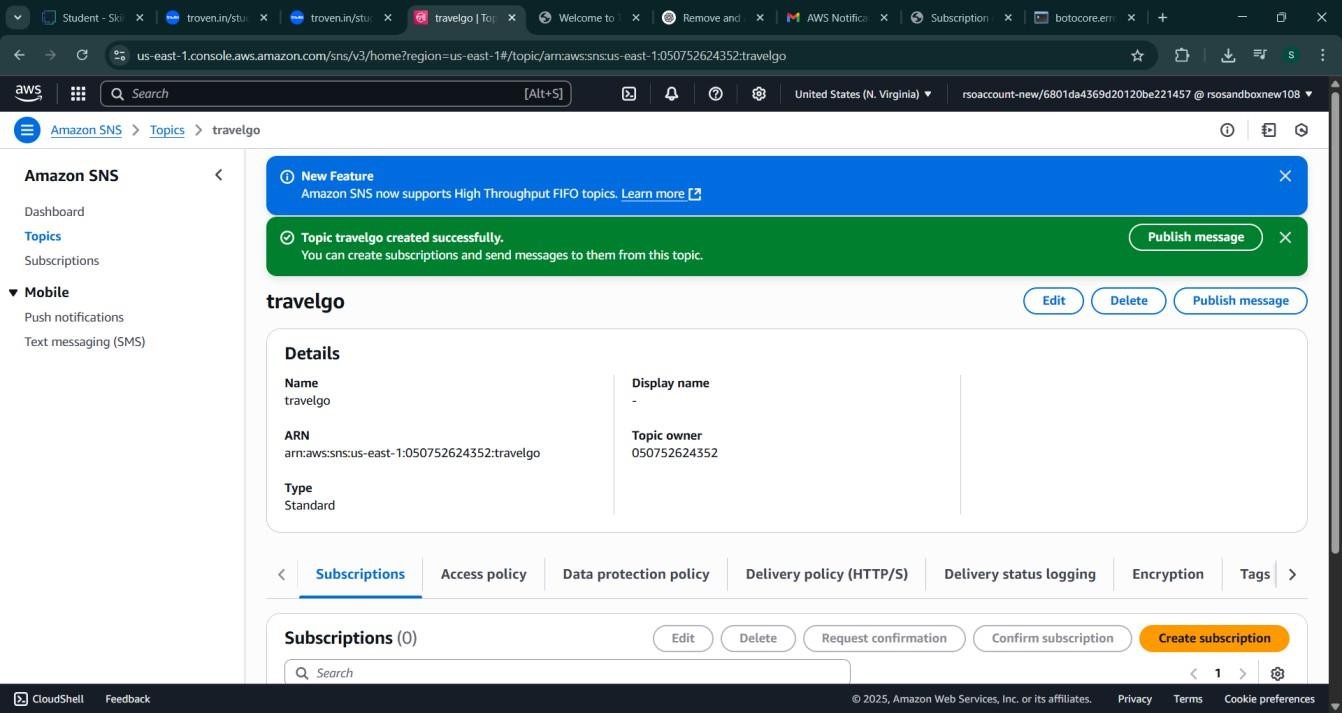
### Activity 3.1: Create SNS topics for sending email notifications to users

****

* + In the AWS Console, search for SNS and navigate to the SNS Dashboard.
* 
  + Click on **Create Topic** and choose a name for the topic.

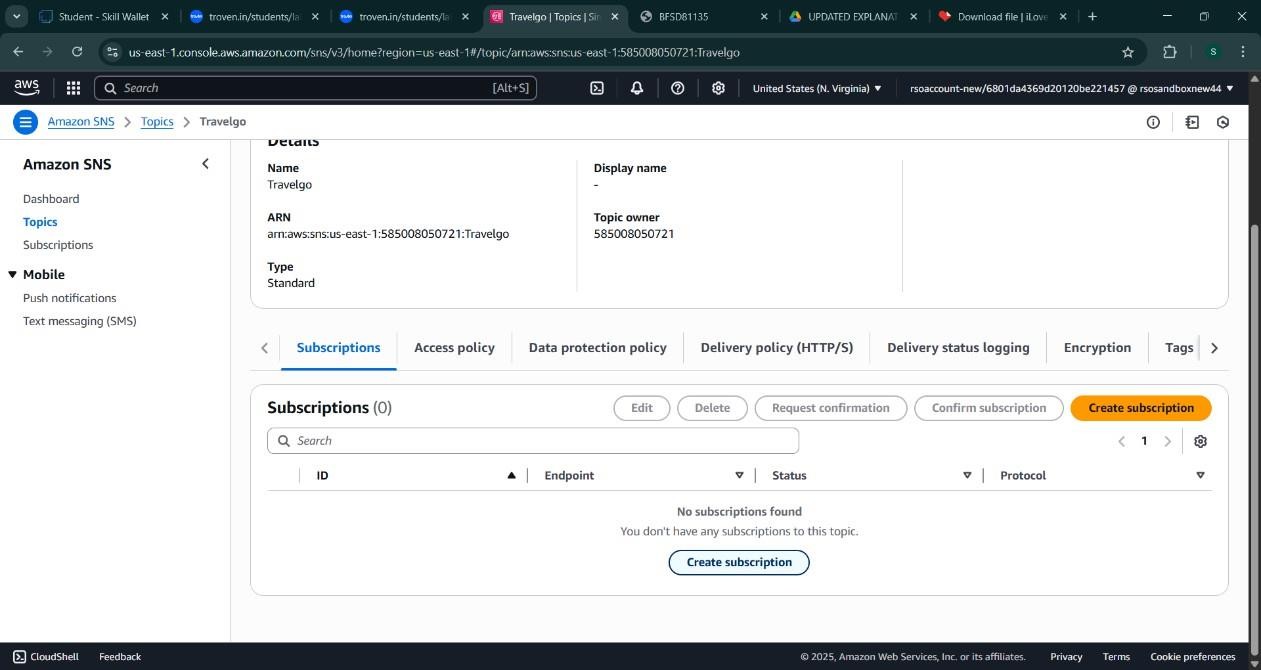


* + Choose Standard type for general notification use cases and Click on Create Topic.

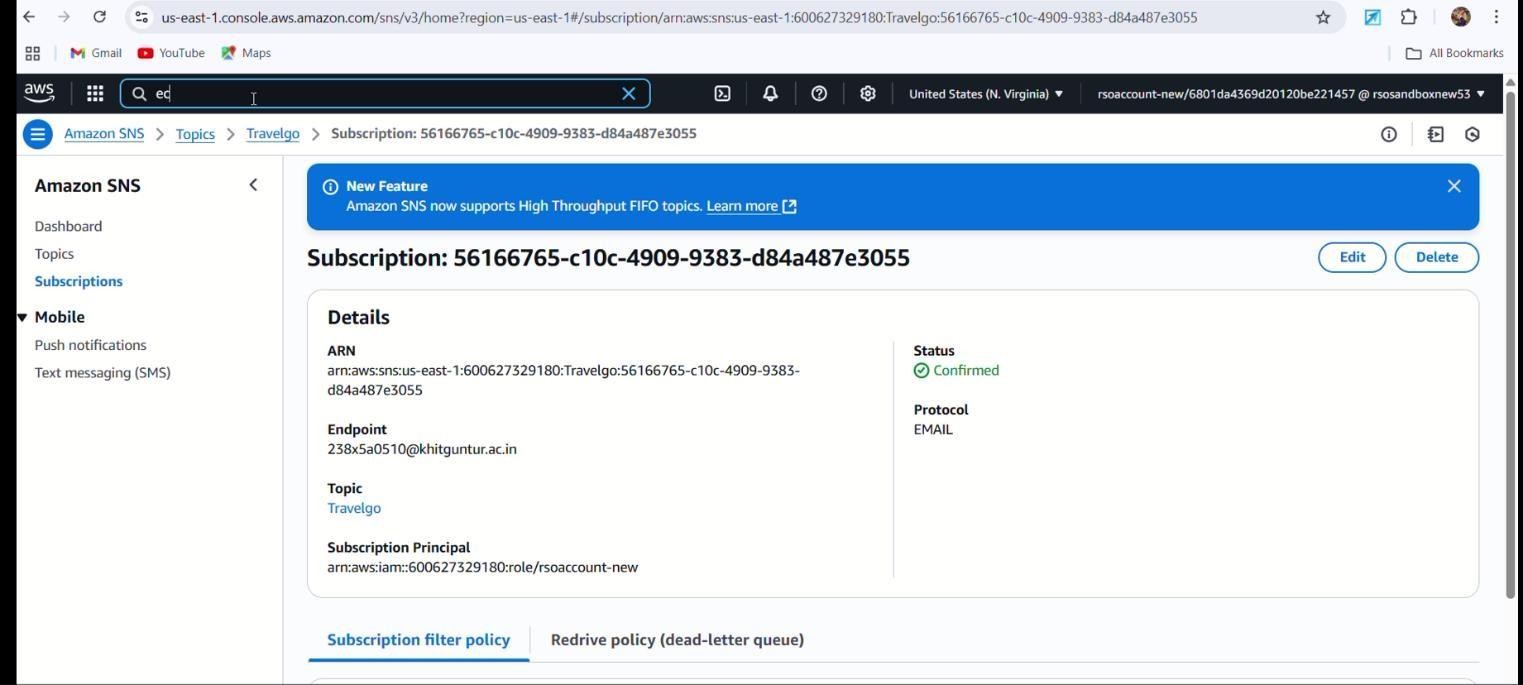


* + Configure the SNS topic and note down the **Topic ARN**.

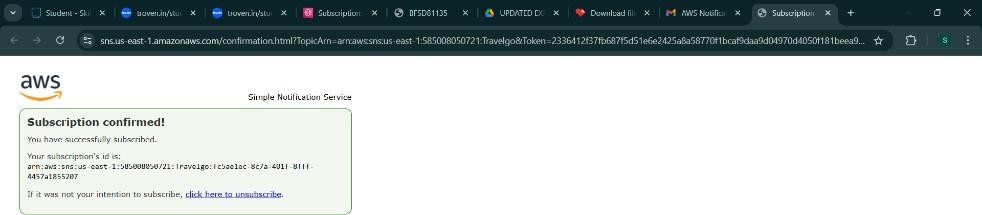
### Activity 3.2: Subscribe users relevant SNS topics to receive real-time notifications when a book request is made.

****

* + Subscribe users 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

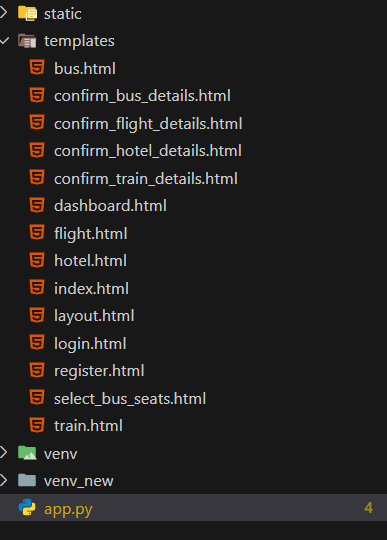


* + Successfully done with the SNS mail subscription and setup, now store the ARN link.

# Milestone 4:Backend Development and Application Setup

### Activity 4.1: Develop the backend using Flask

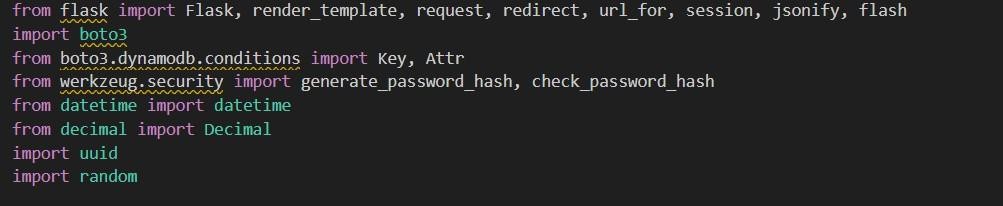
* + File Explorer Structure



**Description:** set up the **TravelGO** project with an app.py file, a static/ folder for assets, and a templates/ directory containing all required HTML pages like home, login, register, booking-specific pages (e.g., bus.html, train.html

## Description of the code :

### Flask App Initialization

****

**Description:** import essential libraries including Flask utilities for routing, Boto3 for DynamoDB operations,



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

### Dynamodb Setup:

****

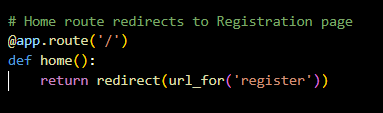
**Description:** initialize the DynamoDB resource for the us-east-1 region and set up access to the Users and Requests tables for storing user details and book requests.

### SNS Connection

**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.

## Routes for Web Pages

### Home Route:

****

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

### Register Route:

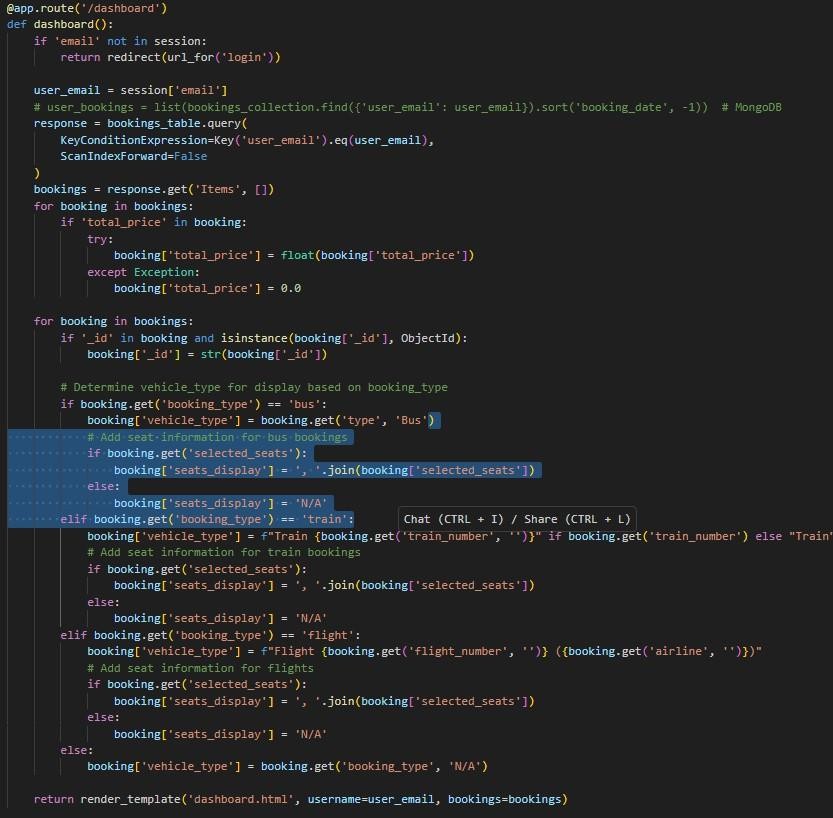
**Description:** define /register route to validate registration form fields, hash the user password using Bcrypt, store the new user in DynamoDB with a login count, and send an SNS notification on successful registration

### login Route (GET/POST):



**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

### Home,Bus,Train,Flight,Hotel Routes:

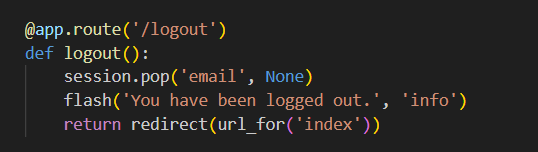


**Description:** define /dashboard-page to render the main homepage, to handle booking selection and redirection.

* + **confirmbooking Routes:**

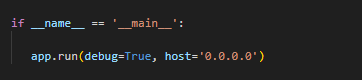
**Description:** define /request-form route to capture book request details from users, store the request in DynamoDB, send a thank-you email to the user, notify the admin, and confirm submission with a success message.

### Exit Route:



**Description:** define /logout route the index.html page to render when the user chooses to leave or close the application.

**Deployment Code:**

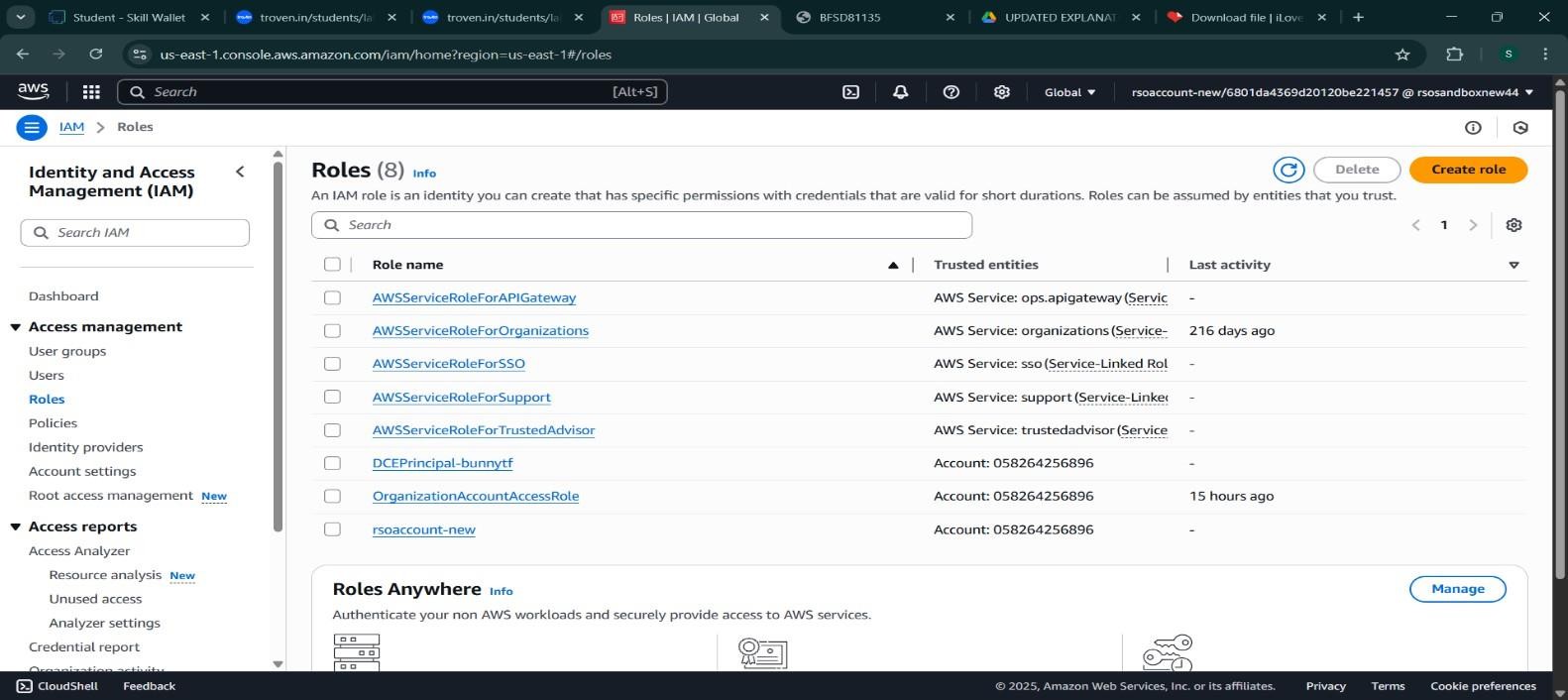


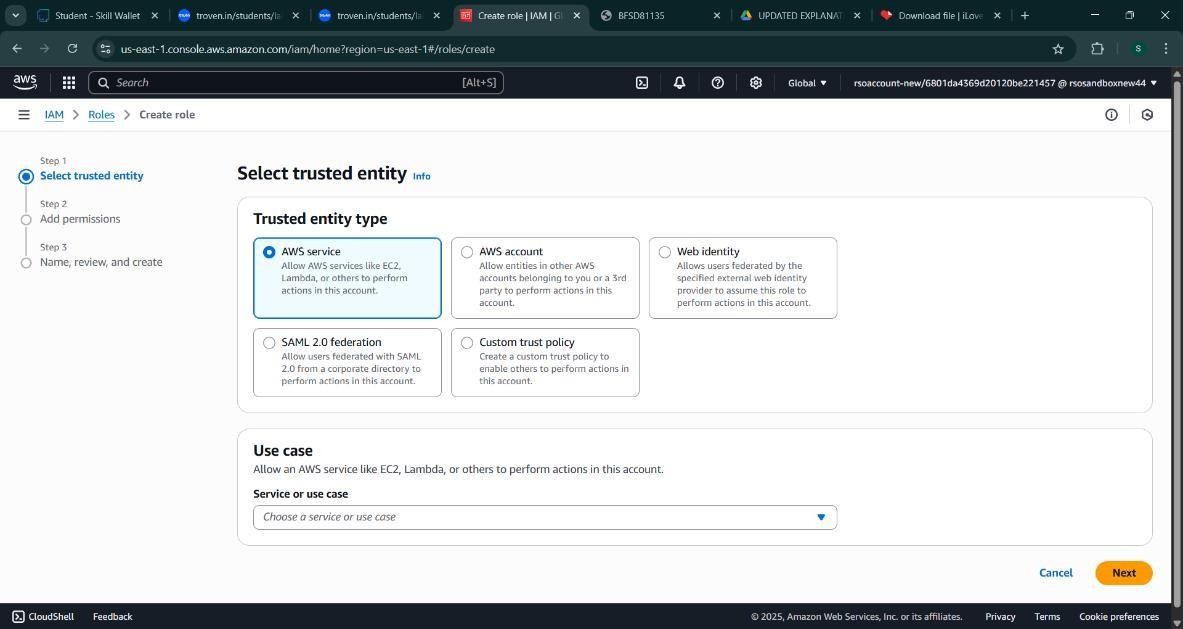
**Description:** start the Flask server to listen on all network interfaces (0.0.0.0) with debug mode enabled for development and testing.

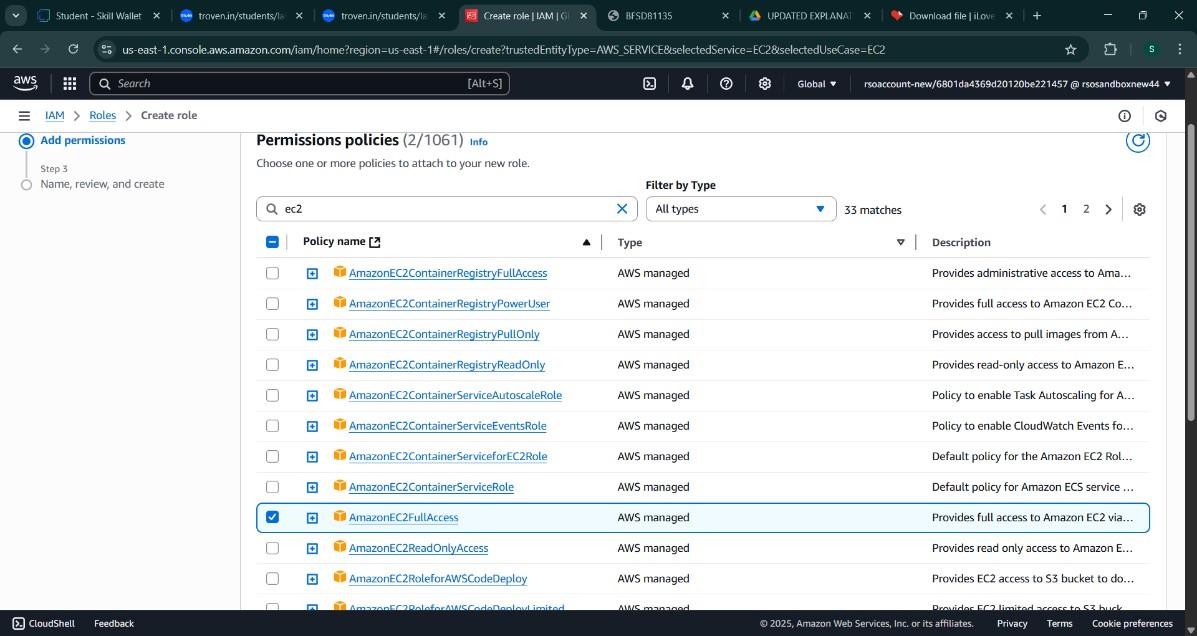
# Milestone 5: IAM Role Setup

### 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.



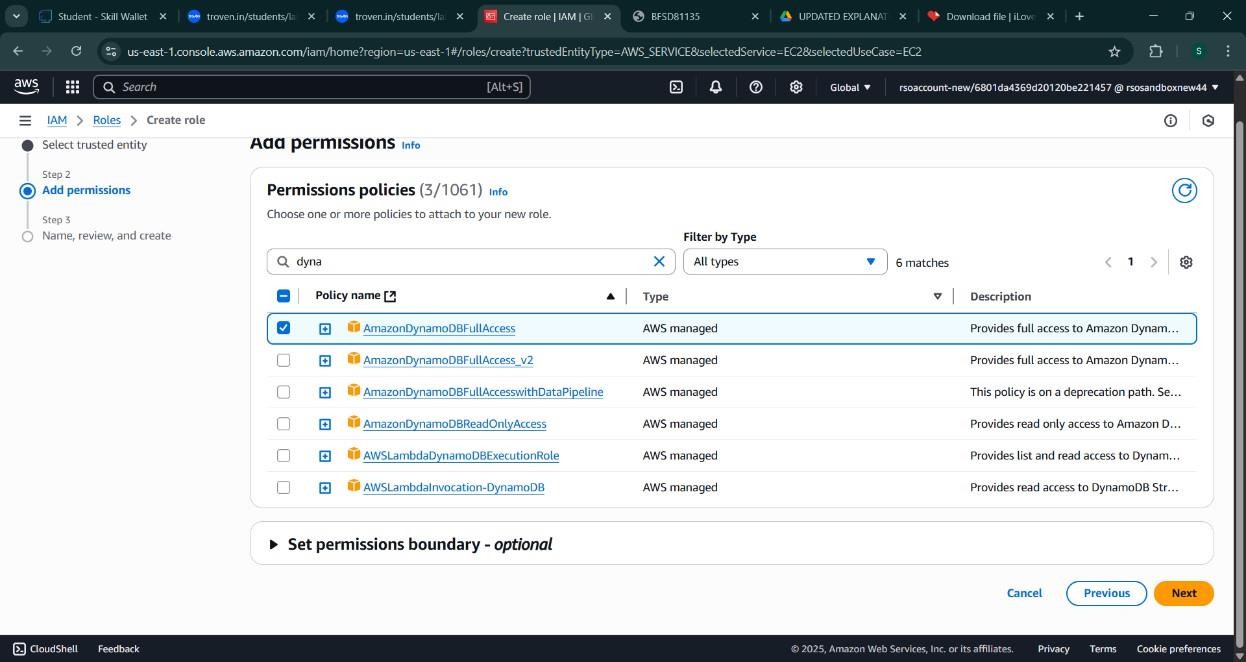


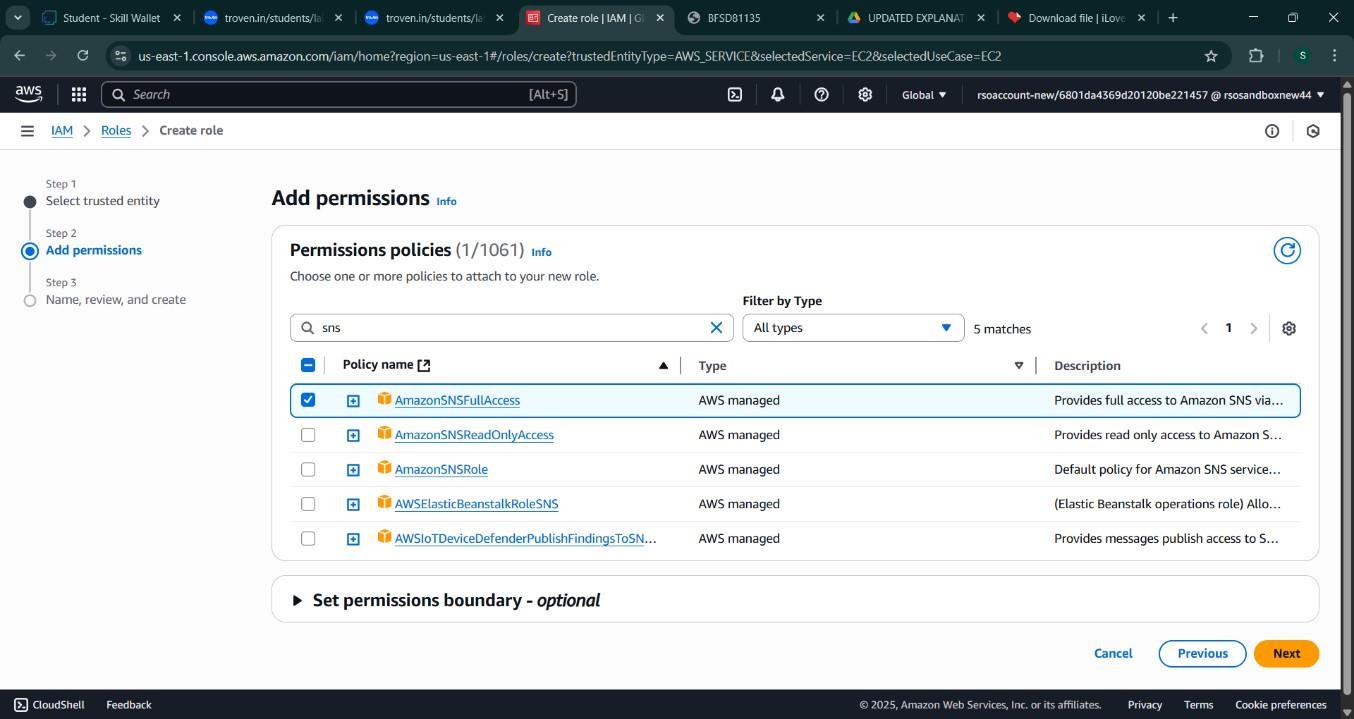


### Activity 5.2: Attach Policies.

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.





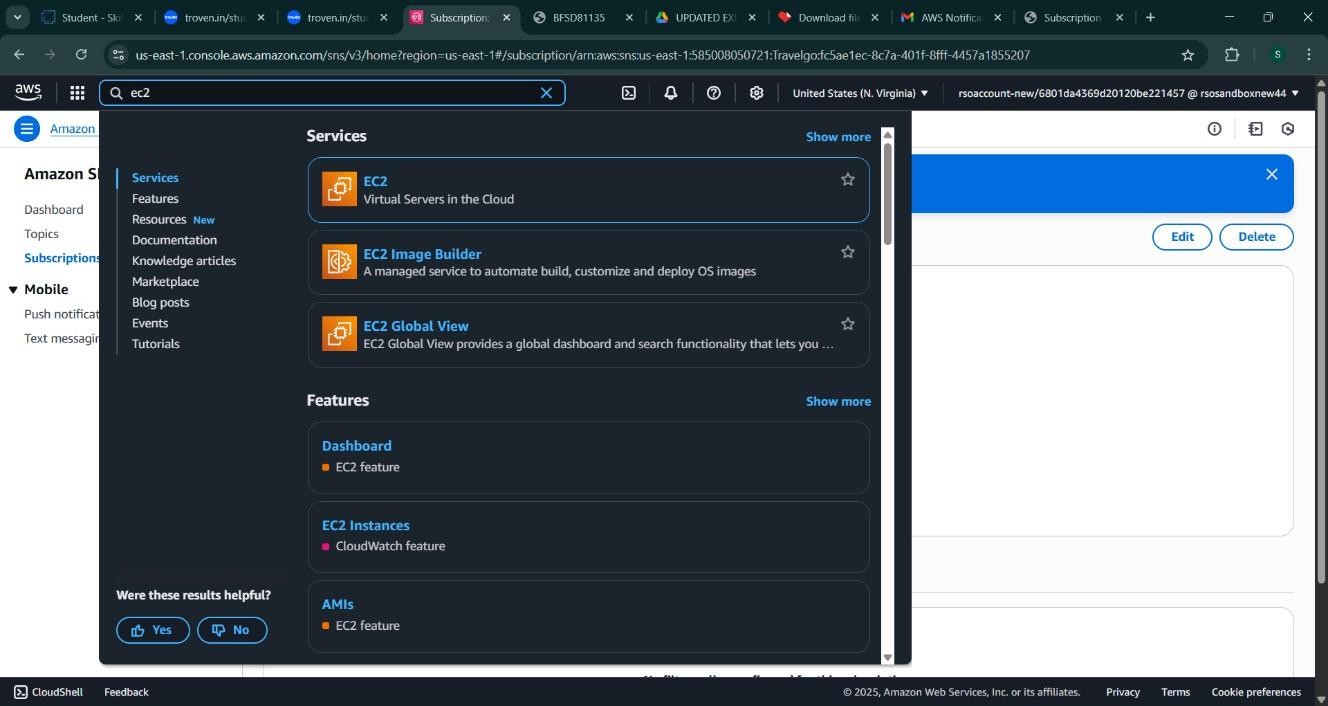
# Milestone 6: EC2 Instance Setup

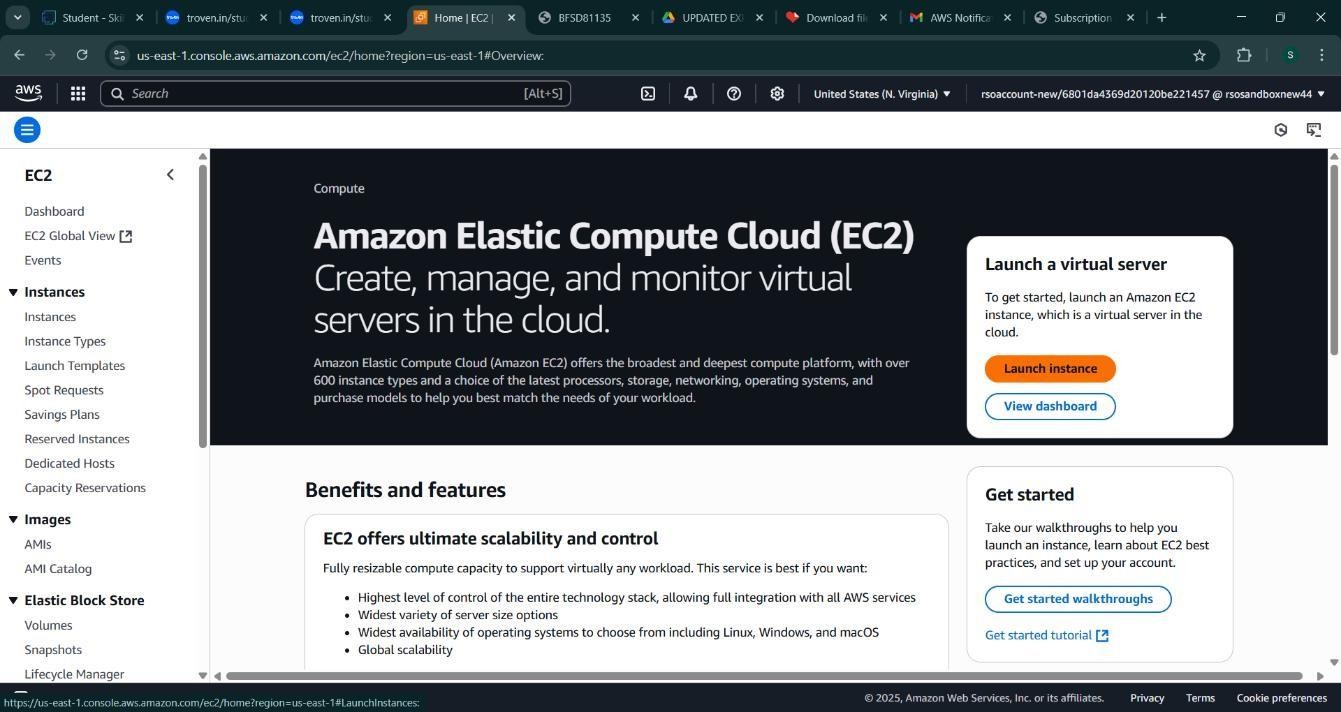
* **Note: Load your Flask app and Html files into GitHub repository.**

****

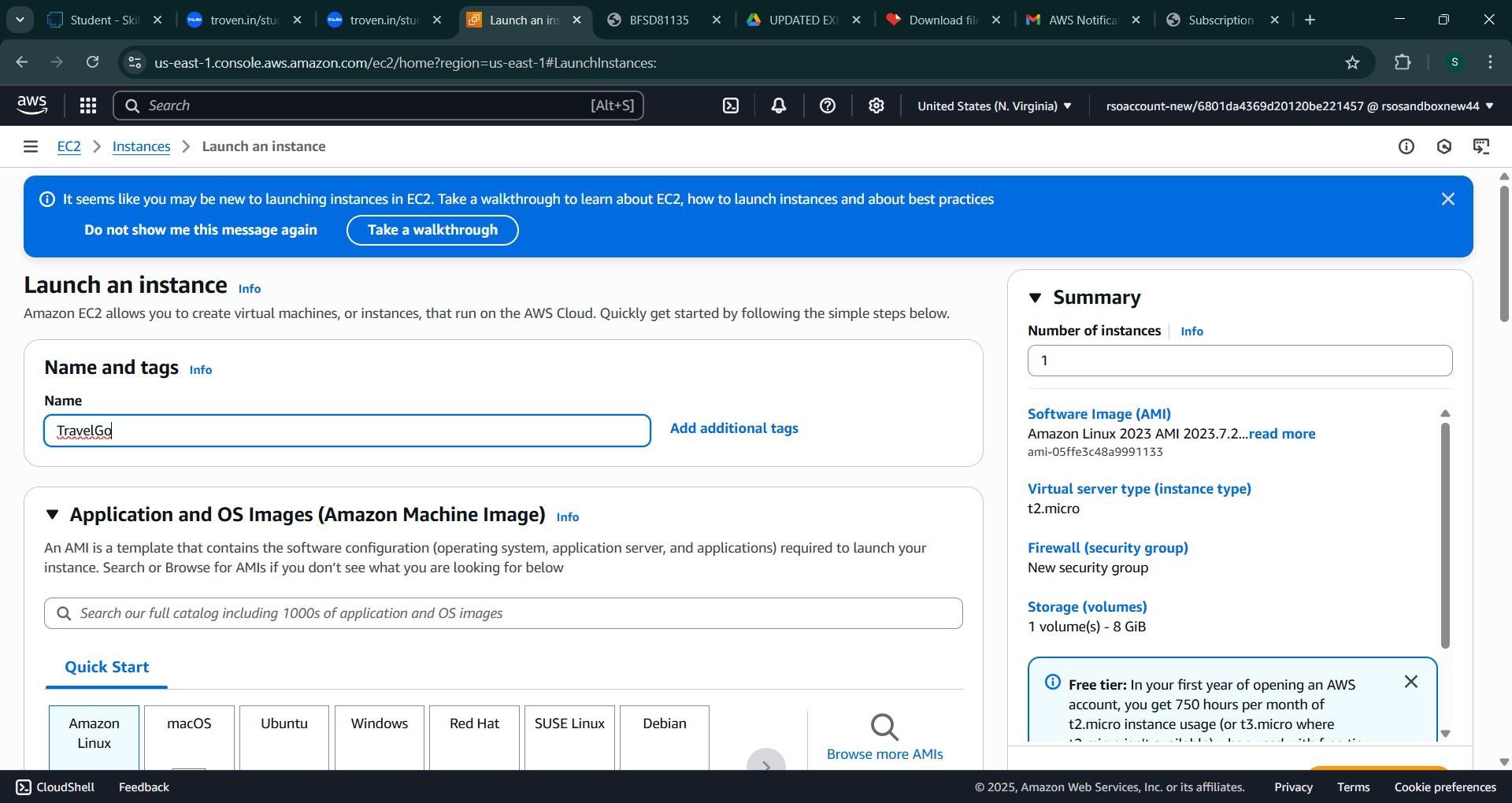
### Activity 6.1: Launch an EC2 instance to host the Flask application.

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



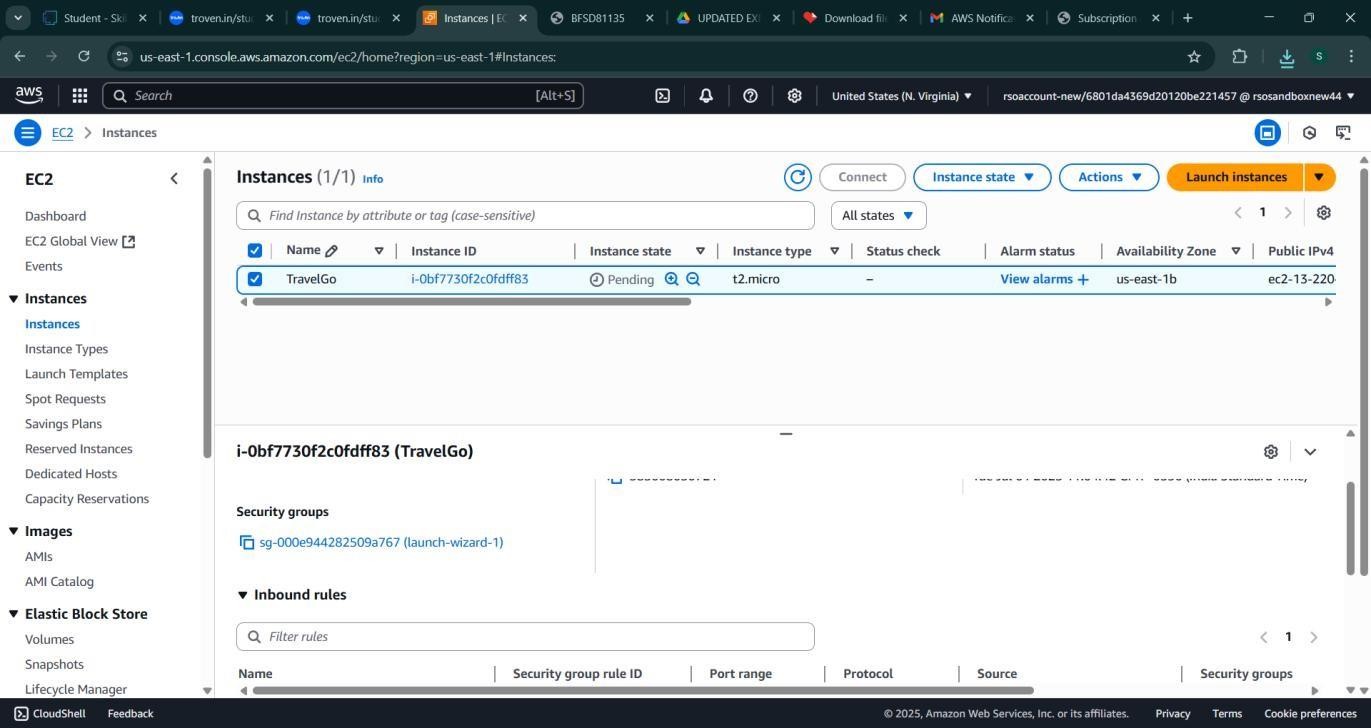


* Click on Launch instance to launch EC2 instance
  + Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).

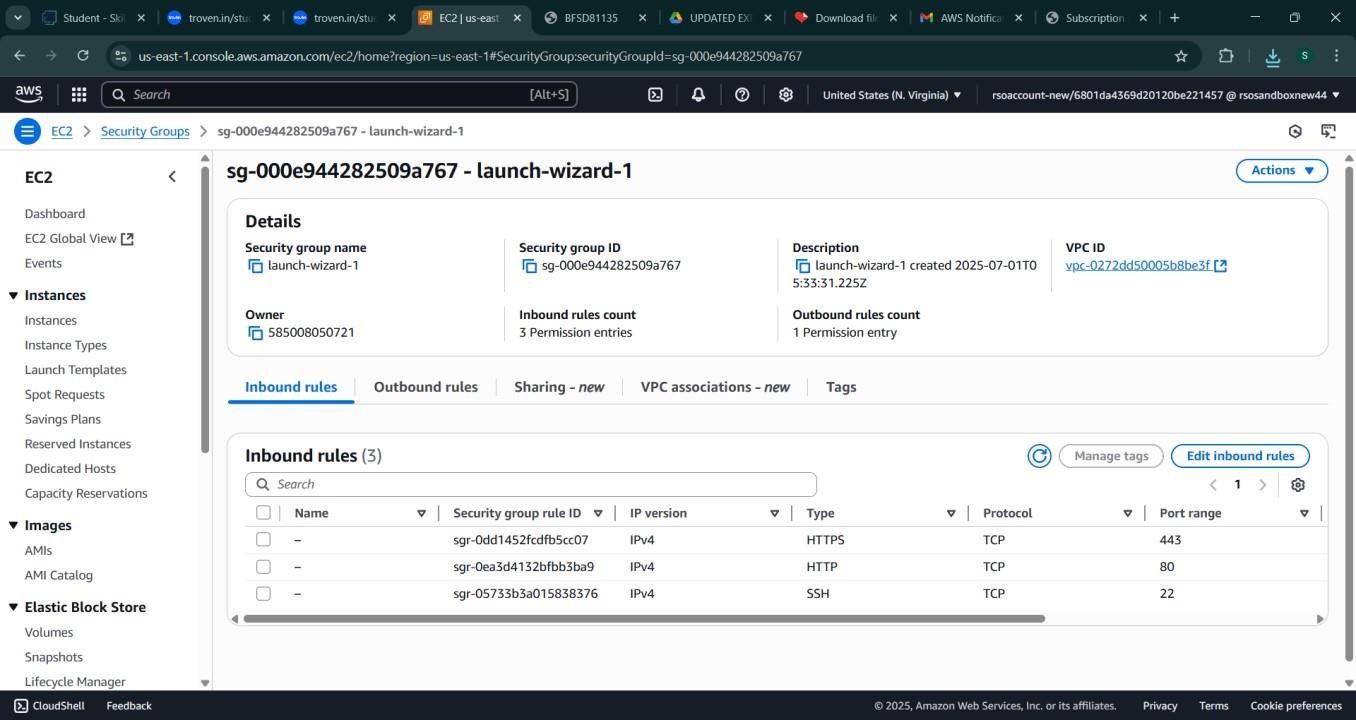


* Create and download the key pair for Server access.

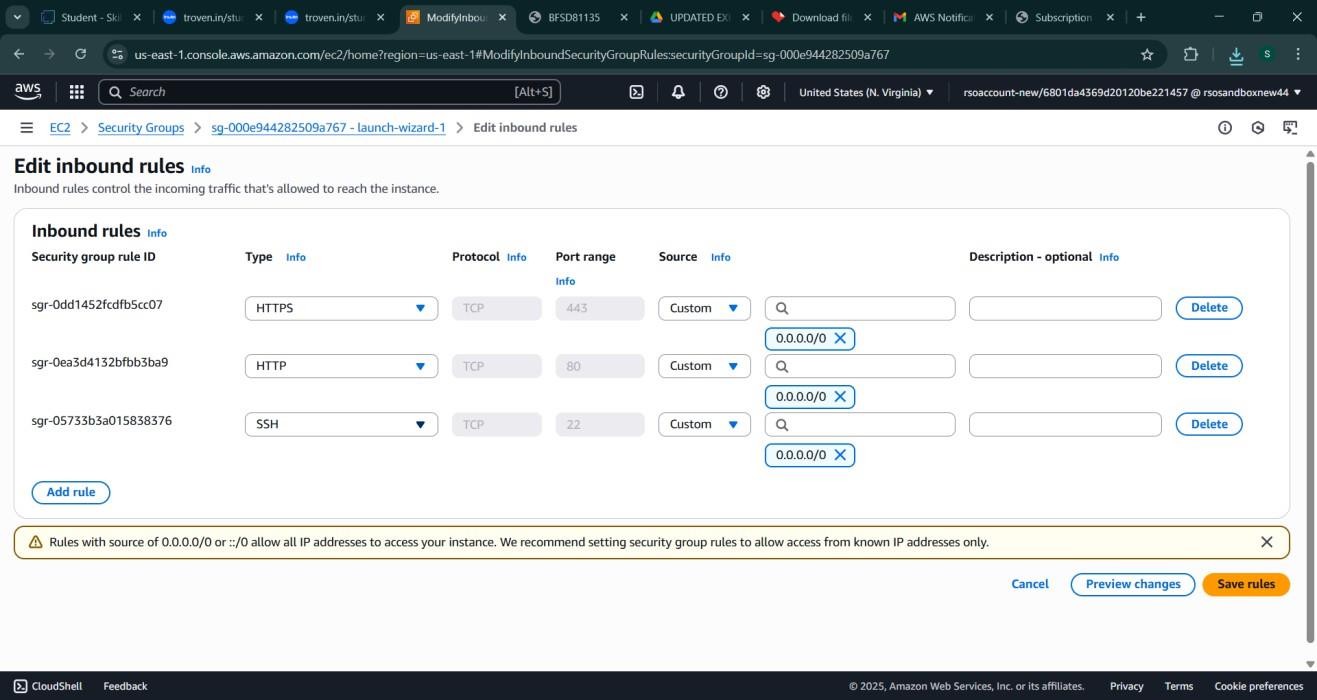


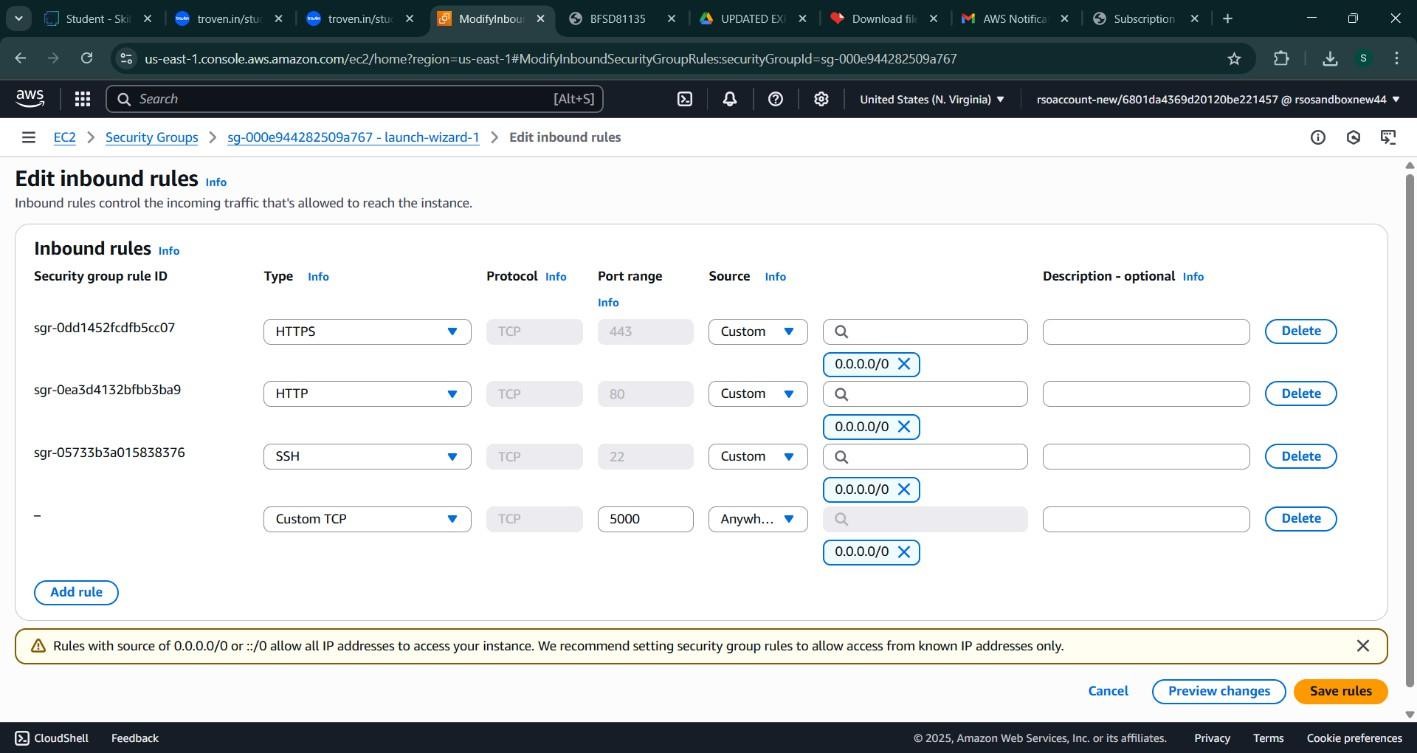


### Activity 6.2:Configure security groups for HTTP, and SSH access.

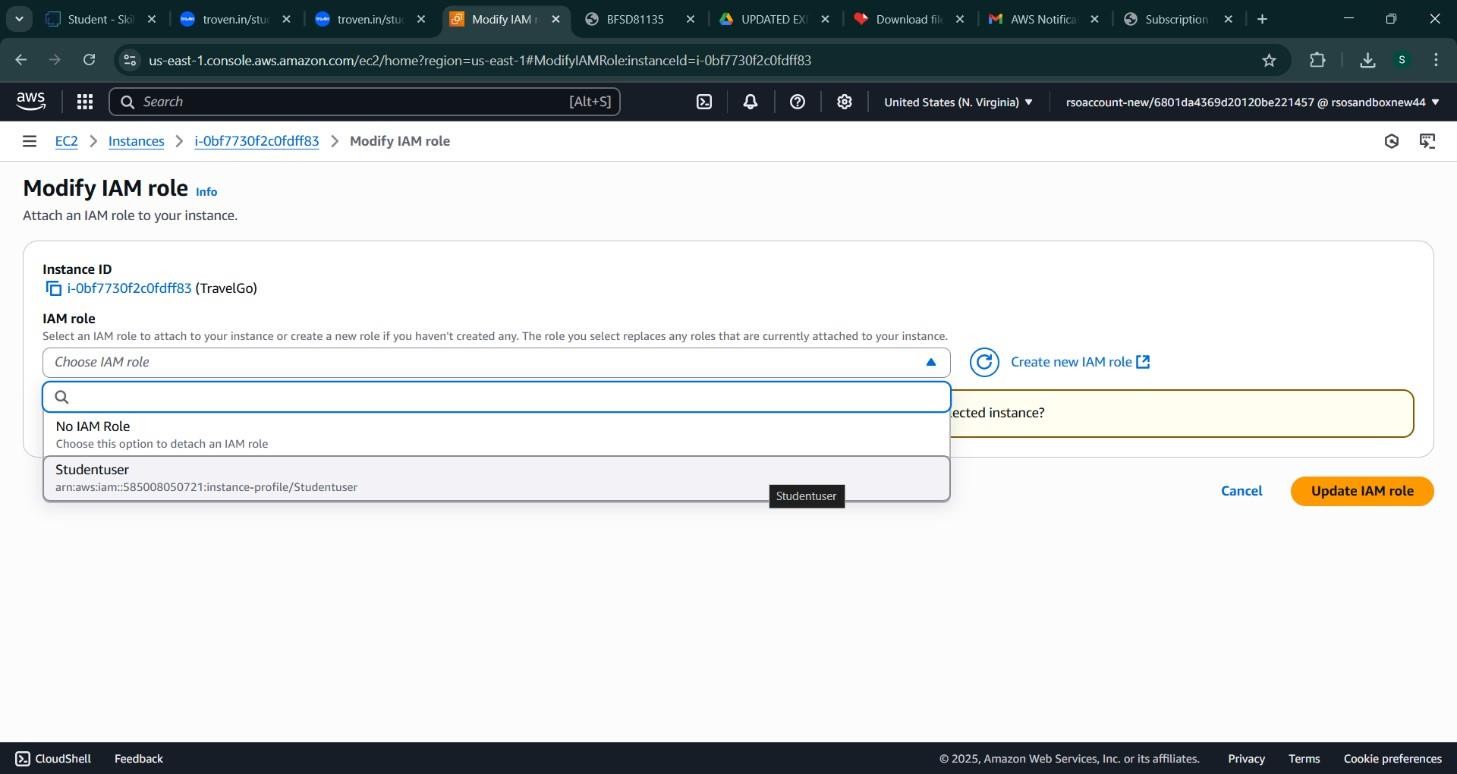
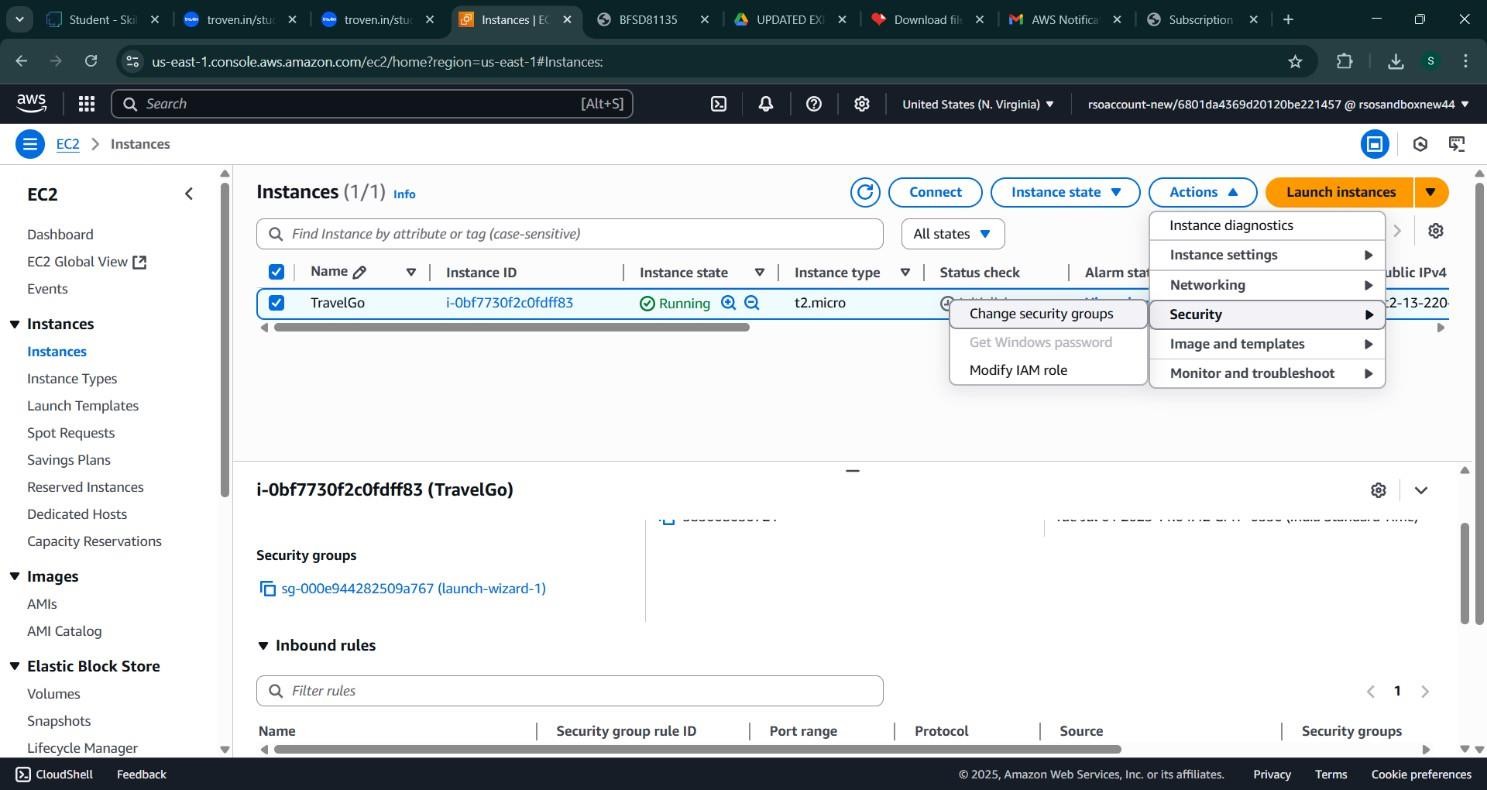
****

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

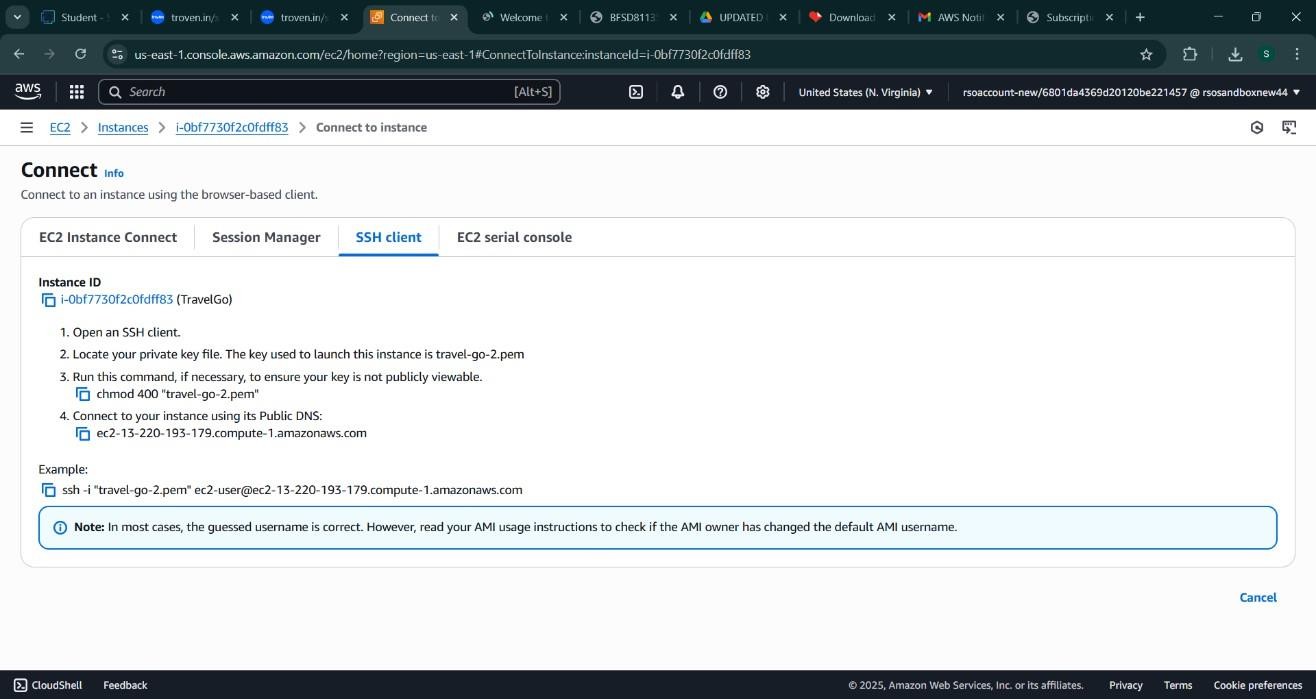
****

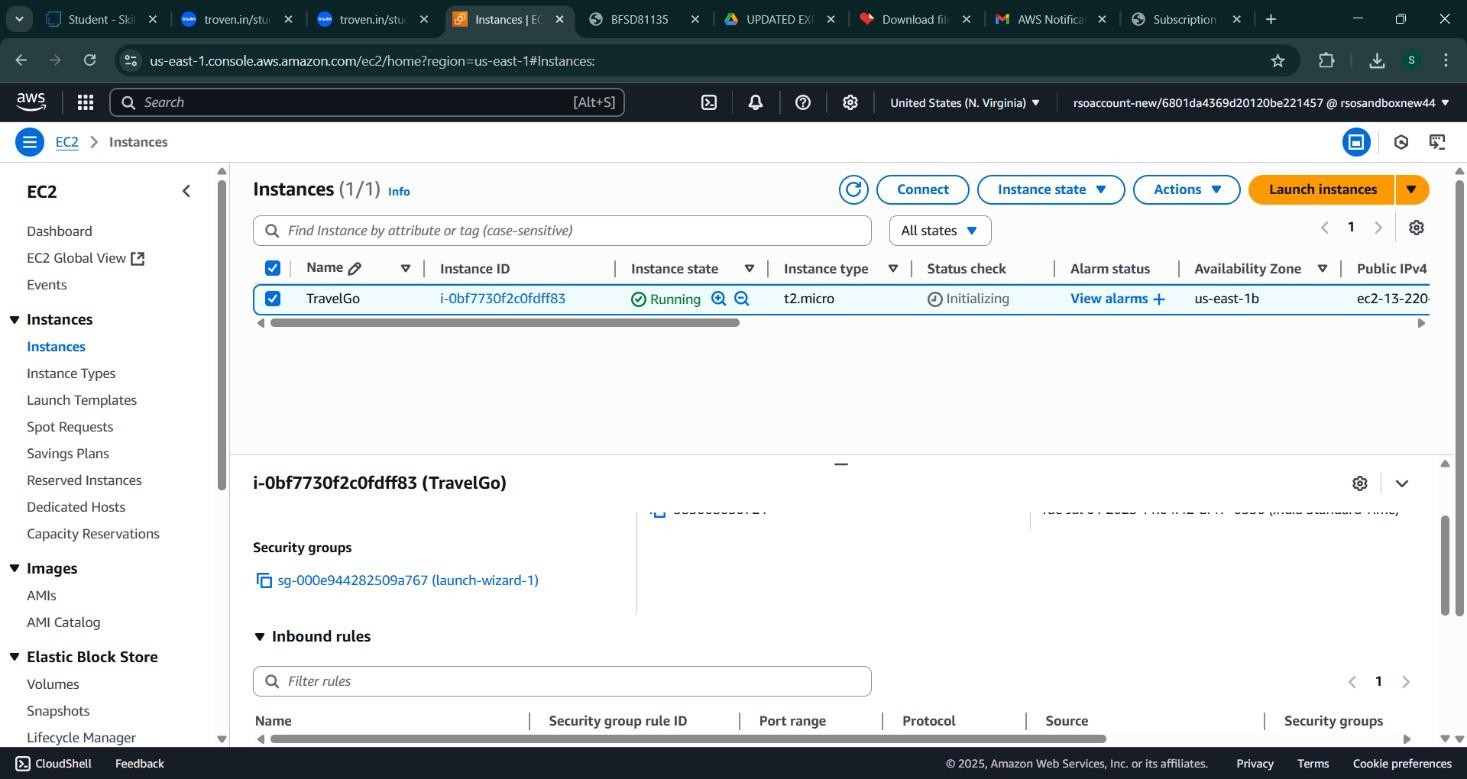
****

* 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





**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’](https://github.com/your-github-username/your-repository-name.git)

Note: change your-github-username and your-repository-name with your credentials here: ‘<https://github.com/Sriram9346834691/Travel-go->’

* This will download your project to the EC2 instance.

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

cd InstantLibrary

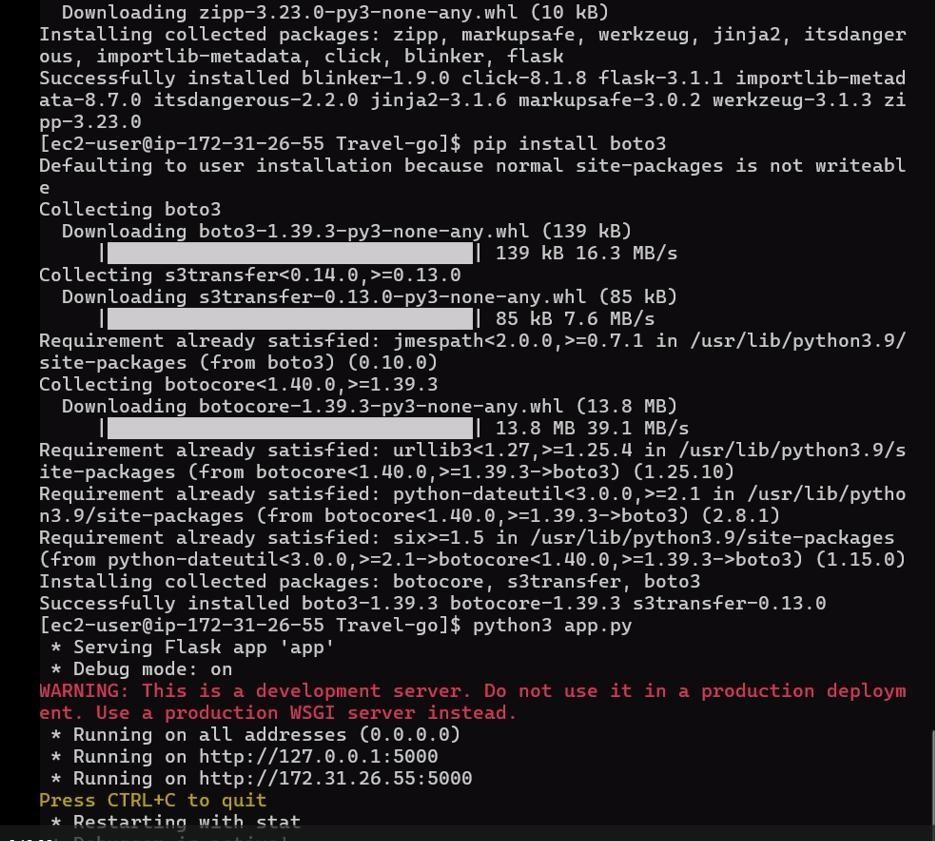
### Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:

**Run the Flask Application**

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

**Verify the Flask app is running**: [http://your-ec2-public-ip](http://your-ec2-public-ip/)

* Run the Flask app on the EC2 instance



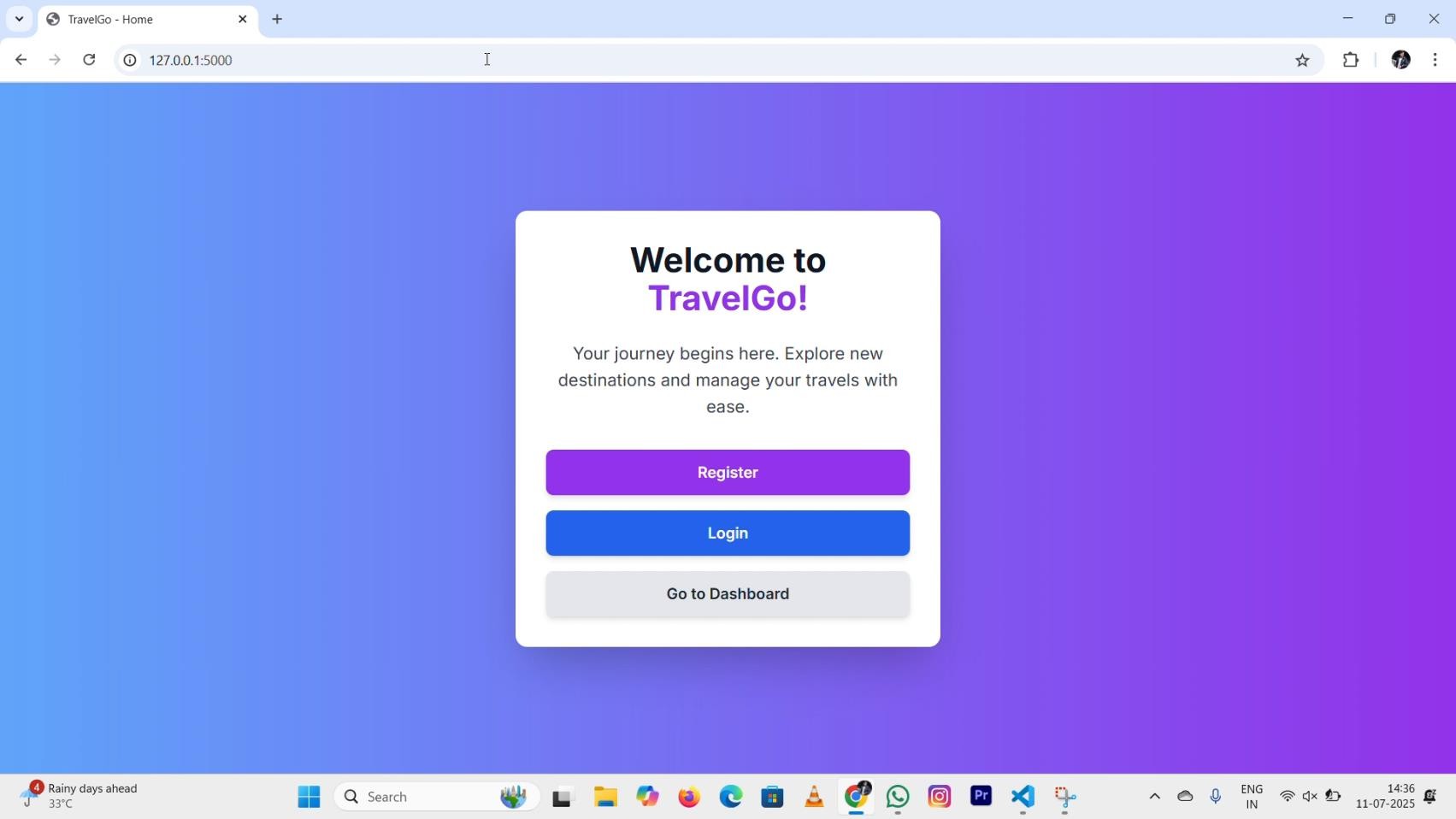
## Milestone 8: Testing and Deployment

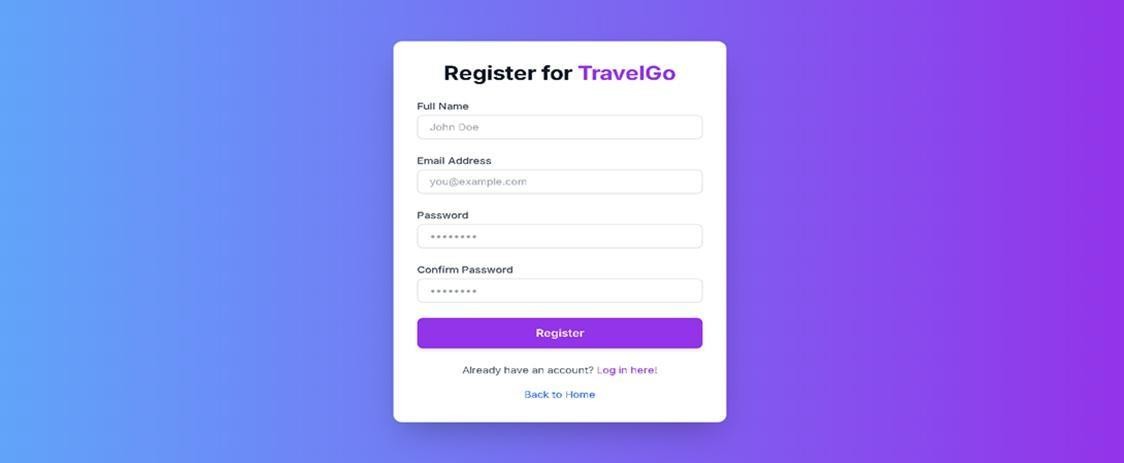
### Activity 8.1: Conduct functional testing to verify user registration, login, book requests, and notifications.

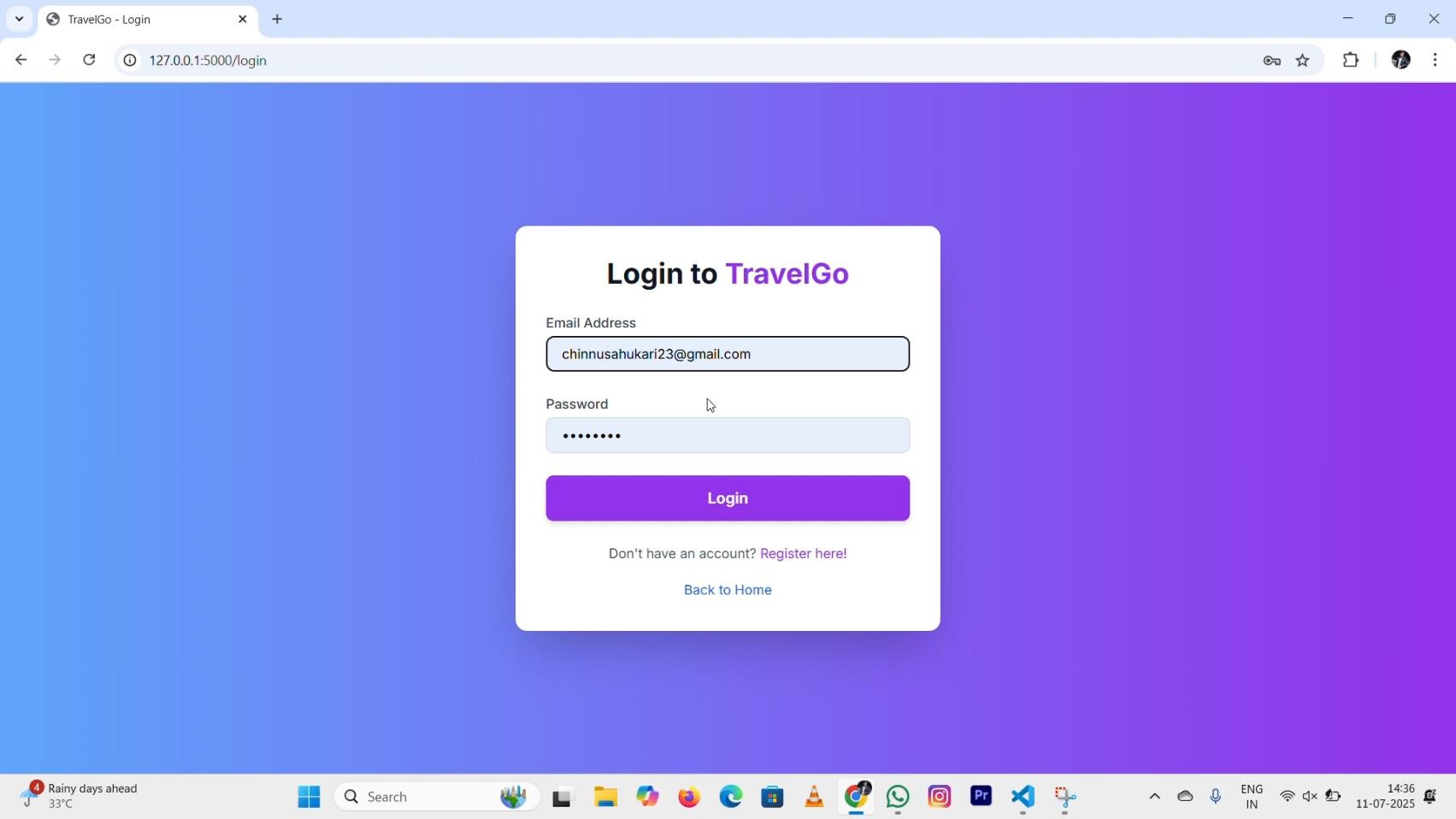
**Welcome Page: Register Page:**

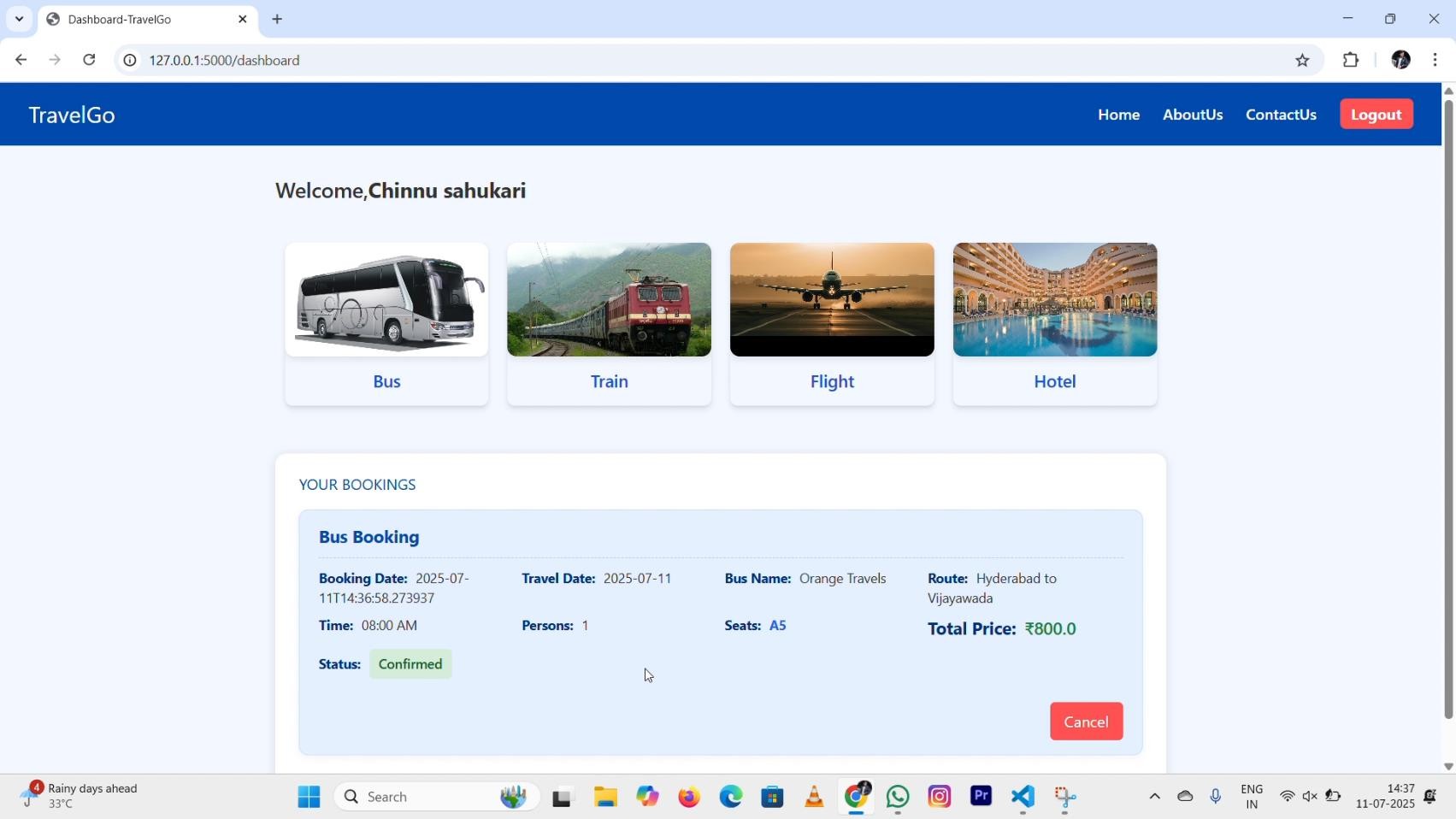
**Login Page: Bus Booking Page:**

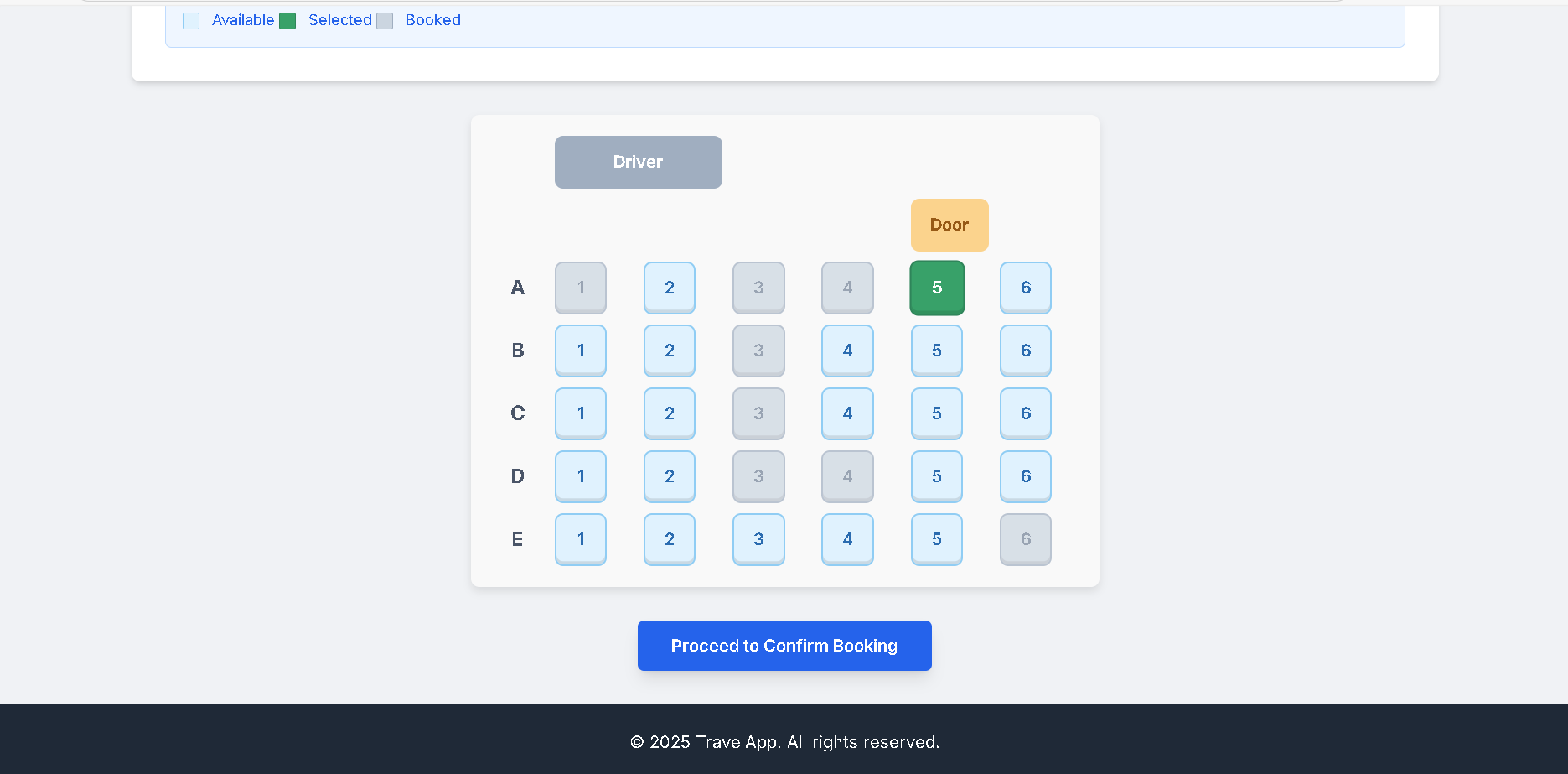
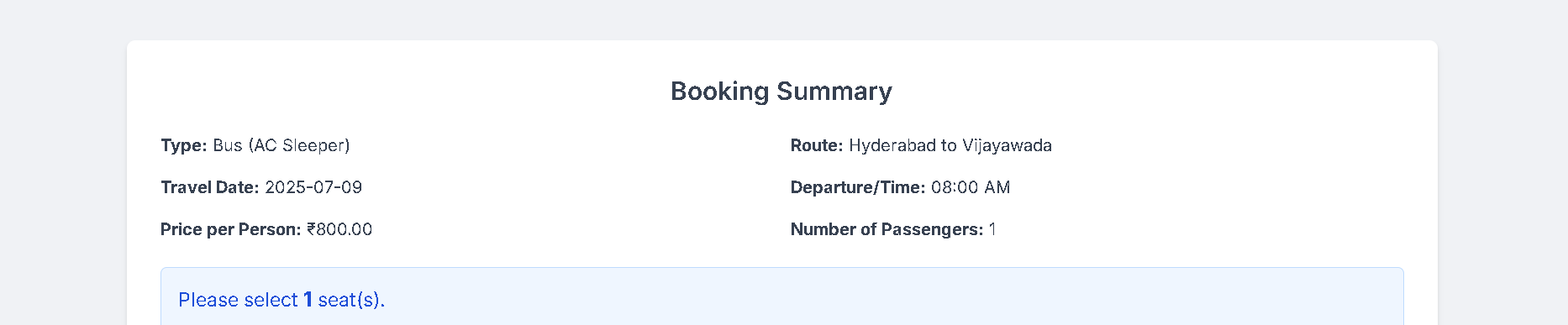
**Train Booking page:**

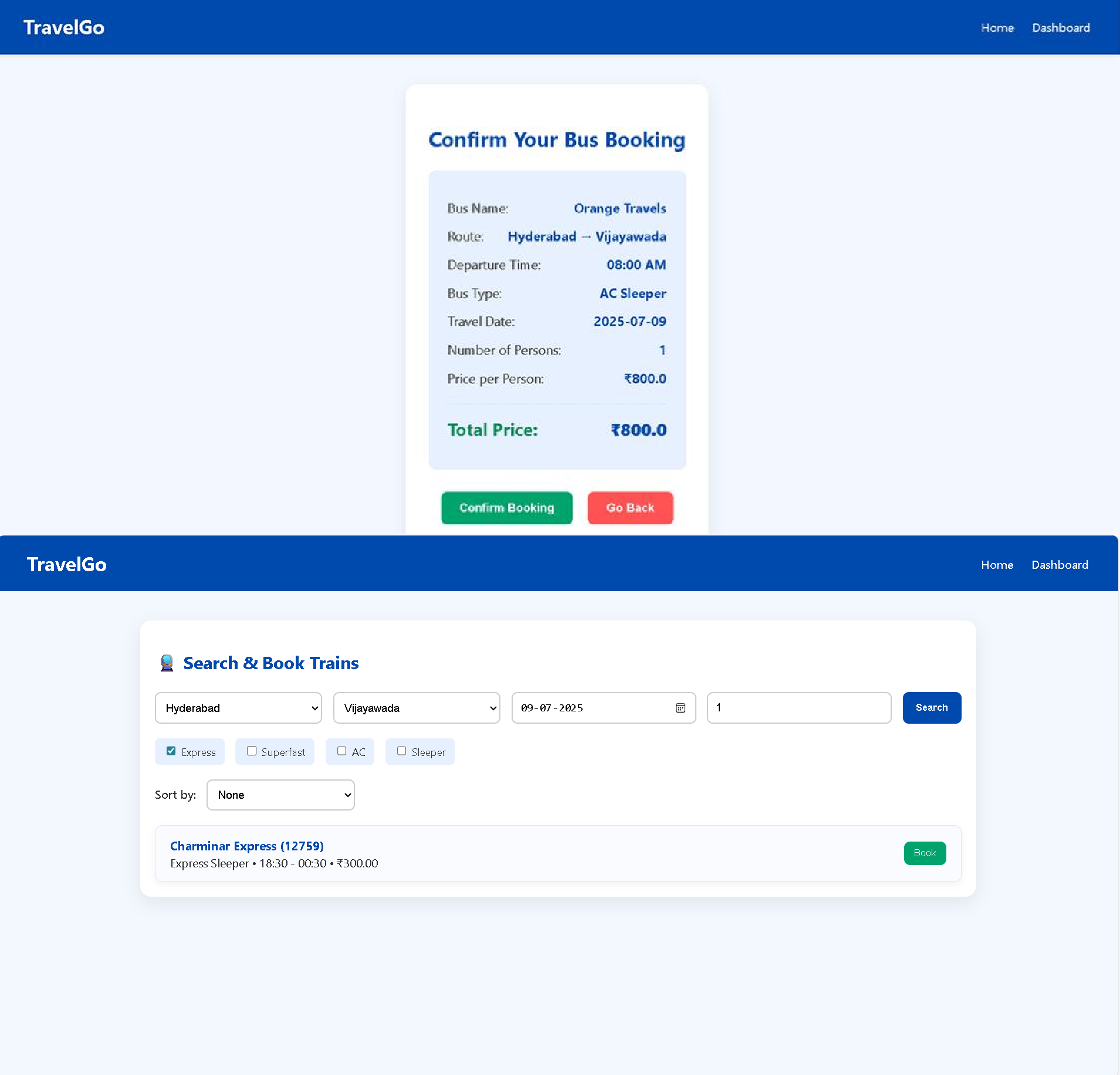
**Flight Booking Page: Hotel Booking Page:**

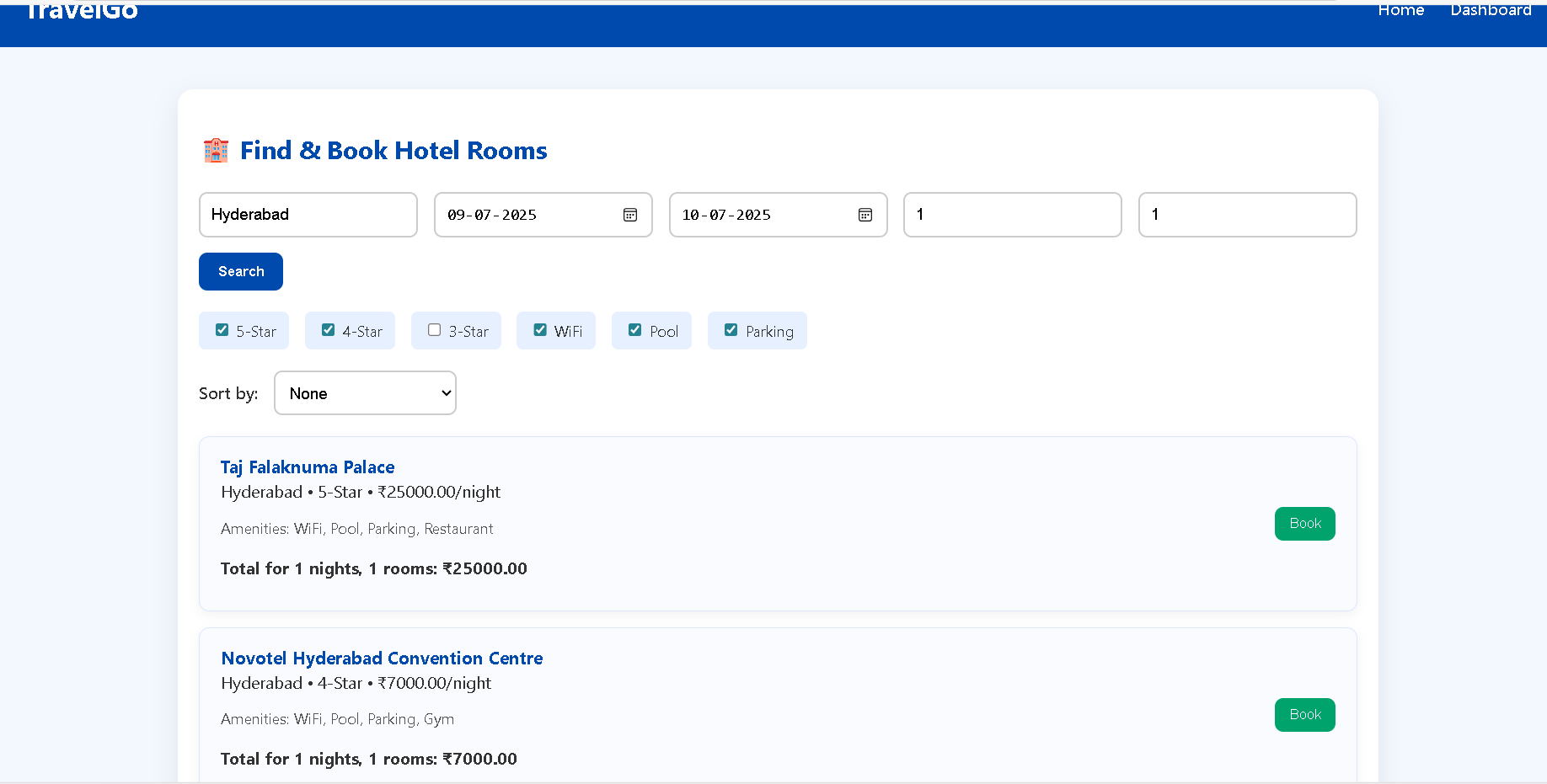
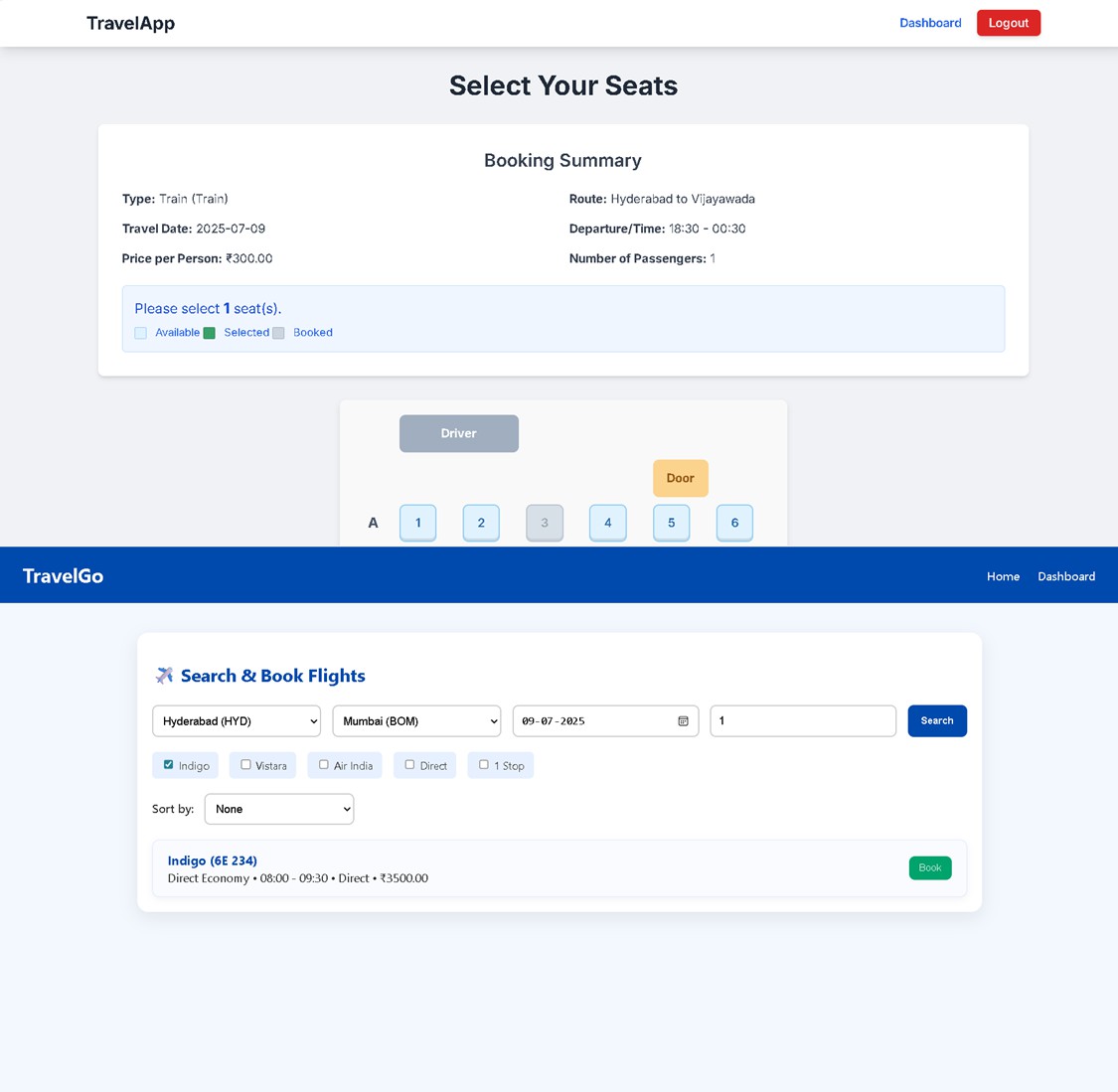
****







# 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.

**THANK YOU**