**Travel Booking System : Quick Access to Travel Services**

### Project Description:

At Greenfield University, students and faculty often face difficulties in managing travel bookings for academic trips, conferences, and internships. Manual booking processes are slow, unorganized, and prone to miscommunication.

To address this, the university's Cloud Solutions Department developed the Travel Booking System—a virtual platform that enables students and staff to book, manage, and track travel arrangements seamlessly. Built using Flask for backend logic, AWS EC2 for hosting, DynamoDB for storing booking data, and AWS SNS for sending real- time travel notifications, this system modernizes and streamlines the entire travel management process.

**Scenario 1: Streamlined Travel Booking for Students and Staff**

**With the Travel Booking System, users can register and log in securely. After logging in, students or faculty can easily access the travel booking interface to schedule transportation or accommodations for academic purposes. AWS EC2 ensures reliable performance, handling concurrent requests efficiently, even during peak usage. Flask manages user sessions and booking logic in real-time, while DynamoDB keeps track of all travel requests and bookings.**

**Scenario 2: Real-Time Travel Notifications**

**Whenever a new travel request is made or updated, AWS SNS instantly notifies the requester and the travel management team. For example, a faculty member books a flight for a conference—once submitted, Flask processes the booking and SNS sends confirmation emails to both the requester and the travel administrator. This ensures prompt communication and prevents delays or miscommunication.**

**Scenario 3: Easy Access to Travel Details**

**Users can log into the platform and view upcoming and past travel bookings. They can filter by date, destination, or status. The interface is intuitive, and backed by DynamoDB, which offers real-time data retrieval. AWS EC2 ensures the site remains available even under high demand, while Flask handles dynamic content rendering for each user.**

**.**

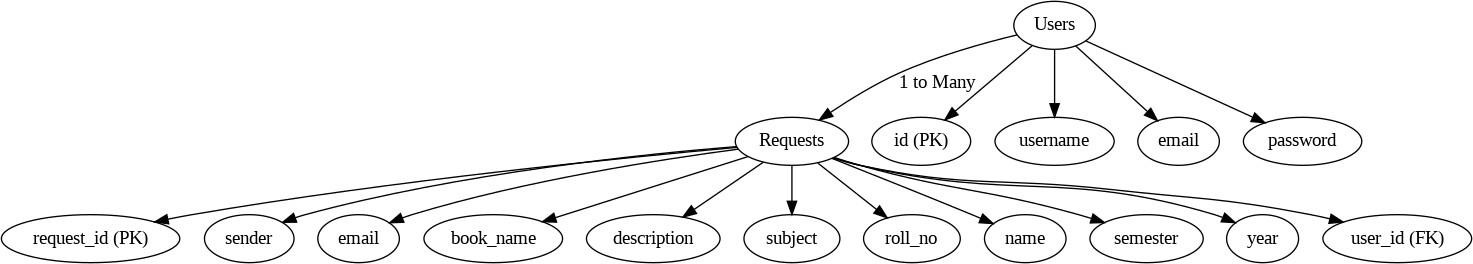
Cloud Architecture Overview :-

* **Frontend: HTML templates rendered by Flask (with routes for registration, login, booking, etc.)**
* **Backend: Flask (Python) application hosted on EC2**
* **Database: AWS DynamoDB (storing users and travel bookings)**
* **Notifications: AWS SNS (email alerts for booking confirmations)**
* **Deployment: Hosted on AWS EC2 Linux instance with Flask and Boto3 Cloud Architecture Overview :**
* **Frontend: HTML templates rendered by Flask (with routes for registration, login, booking, etc.)**
* **Backend: Flask (Python) application hosted on EC2**
* **Database: AWS DynamoDB (storing users and travel bookings)**
* **Notifications: AWS SNS (email alerts for booking confirmations)**
* **Deployment: Hosted on AWS EC2 Linux instance with Flask and Boto3**

### AWS ARCHITECTURE



**Entity Relationship (ER)Diagram:**



**Pre-Requisites :-**

1. **AWS Account**
2. **IAM Configuration**
3. **EC2 Instance Setup**
4. **DynamoDB Tables for Users and Bookings**
5. **SNS Topics for Travel Notifications**
6. **Flask App Code (uploaded via GitHub)**
7. **Git for version control**

### Project WorkFlow:

**Milestone 1: AWS Setup**

* + **Create AWS account**
  + **Configure IAM roles for EC2 instance**

**Milestone 2: DynamoDB Setup**

* + **Create tables: Users (with Email as partition key), Bookings (with BookingID or Email as**

**key)**

**Milestone 3: SNS Notifications**

* + **Create topic: travel-booking-alerts**
  + **Subscribe admin/staff and users via email**

**Milestone 4: Backend Development (Flask + Boto3)**

* **Flask routes: /register, /login, /book-travel, /view-bookings**
* **Store and retrieve data using DynamoDB**
* **Send SNS notifications on travel booking**

**Milestone 5: EC2 Hosting**

* + **Launch EC2 instance**
  + **Install Flask, Git, Boto3**
  + **Clone GitHub repo and run the Flask server**

**Milestone 6: Testing and Deployment**

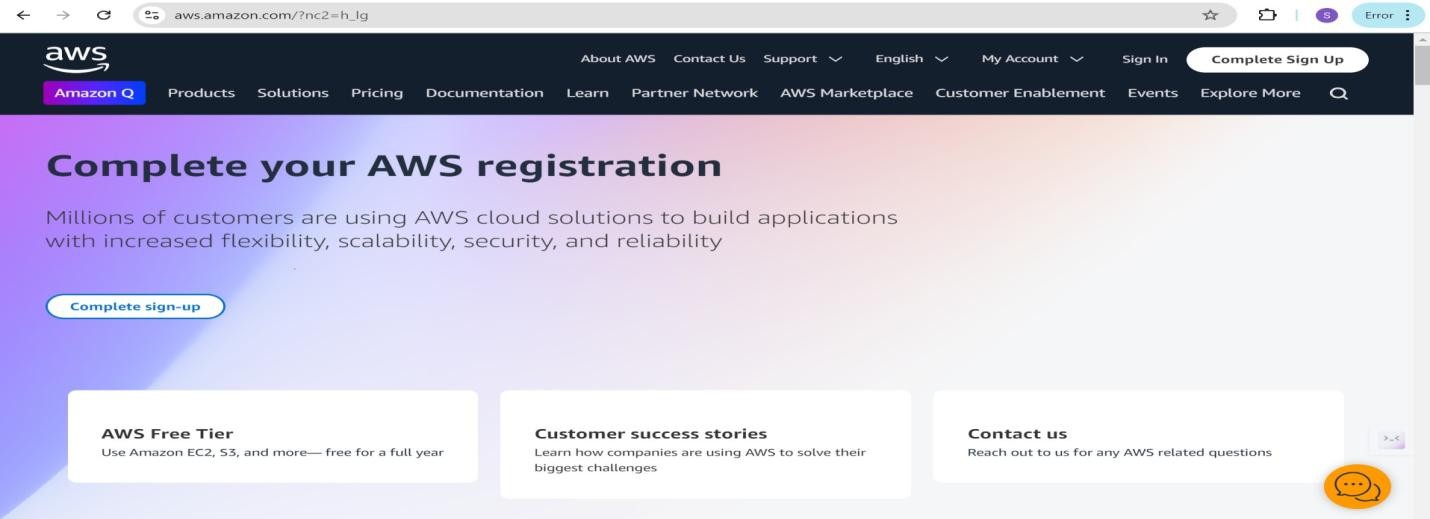
* + **Test user registration, login, booking functionality, and email alerts**

1. **Testing and Deployment**

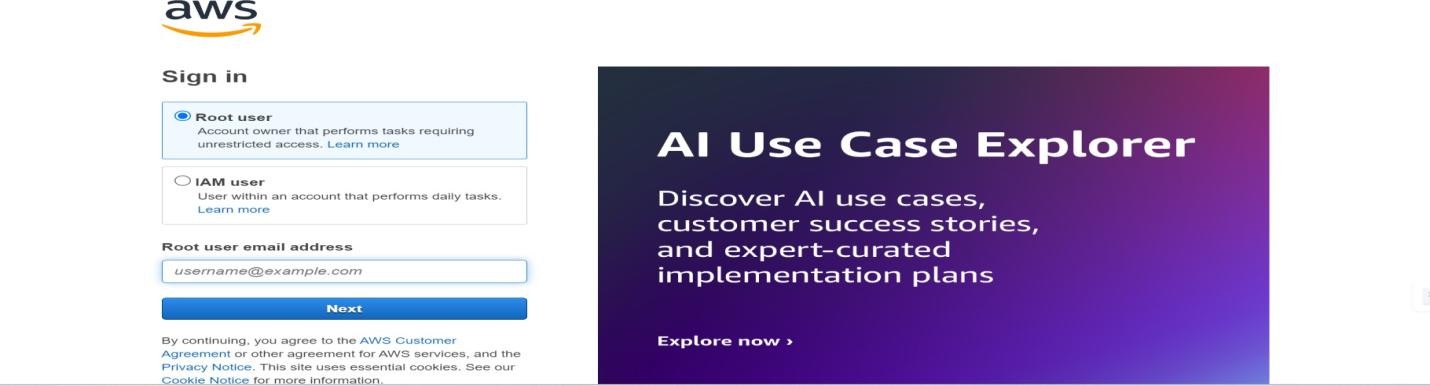
Activity 8.1: Conduct functional testing to verify user registration, login, book requests, and notifications.

## Milestone 1: AWS Account Setup and Login

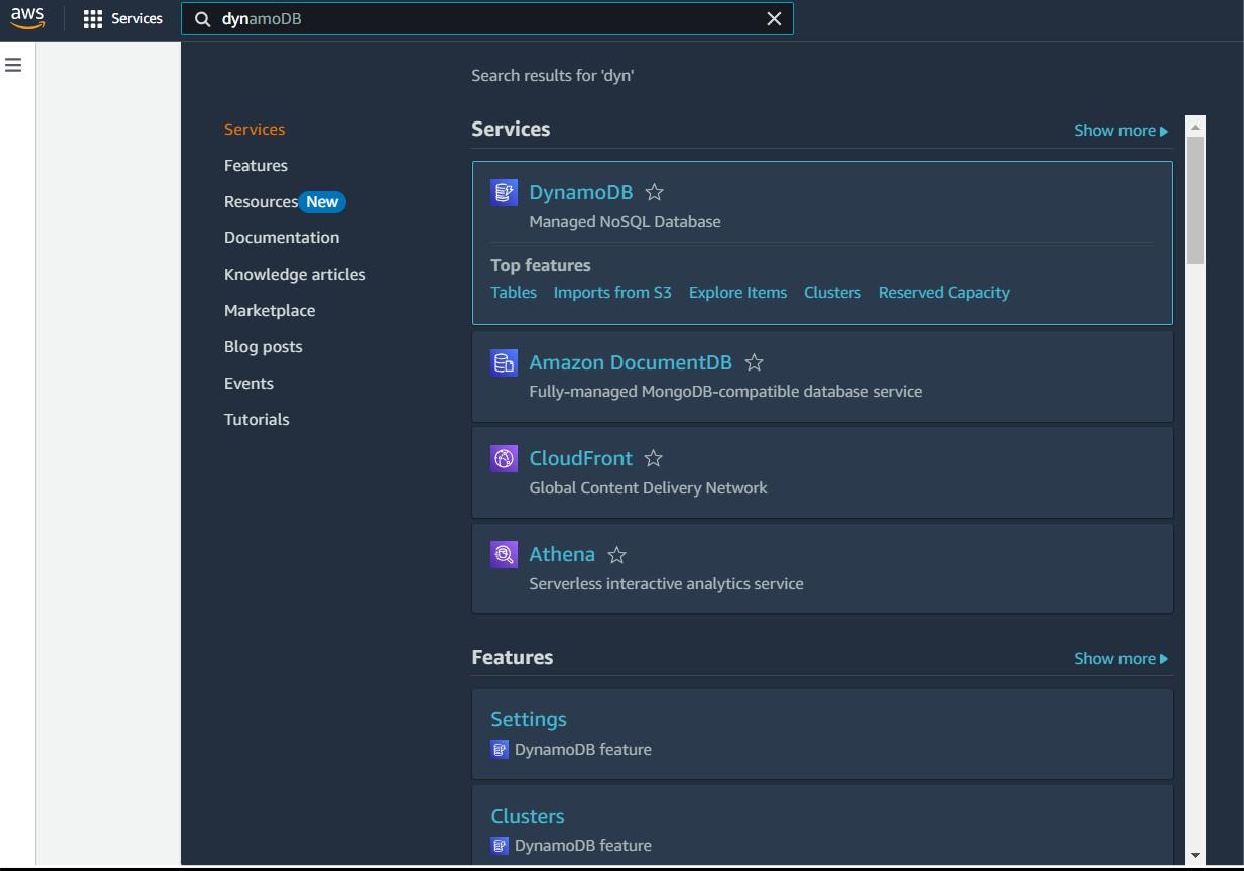
* + **Activity 1.1: Set up an AWS account if not already done.**
    - **Sign up for an AWS account and configure billing settings.**

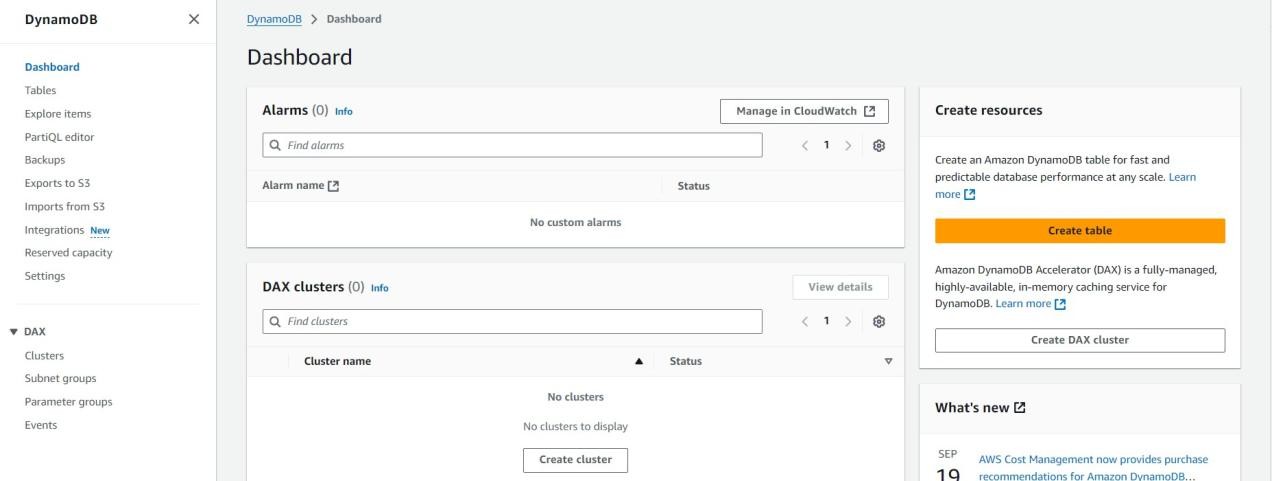


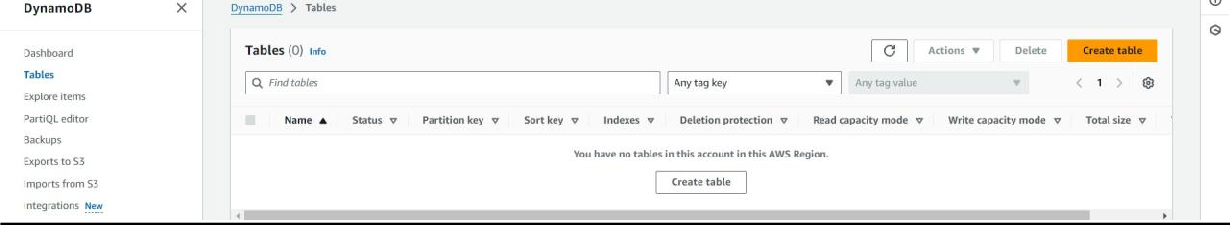
* + **Activity 1.2: Log in to the AWS Management Console**
    - **After setting up your account, log in to the** [**AWS Management Console.**](https://aws.amazon.com/console/)



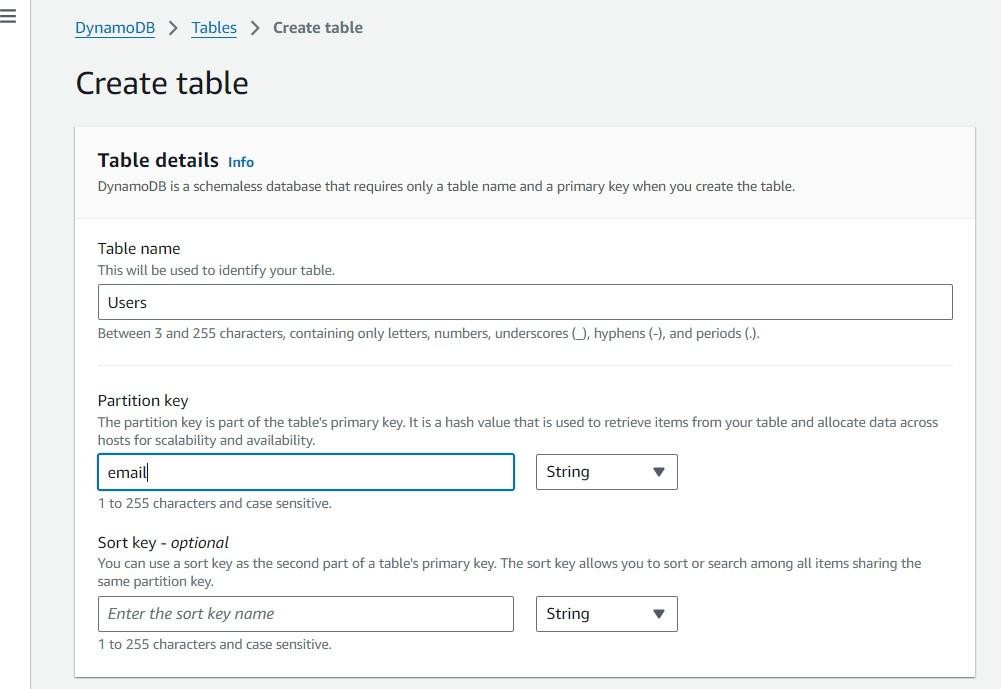
## Milestone 2: DynamoDB Database Creation and Setup

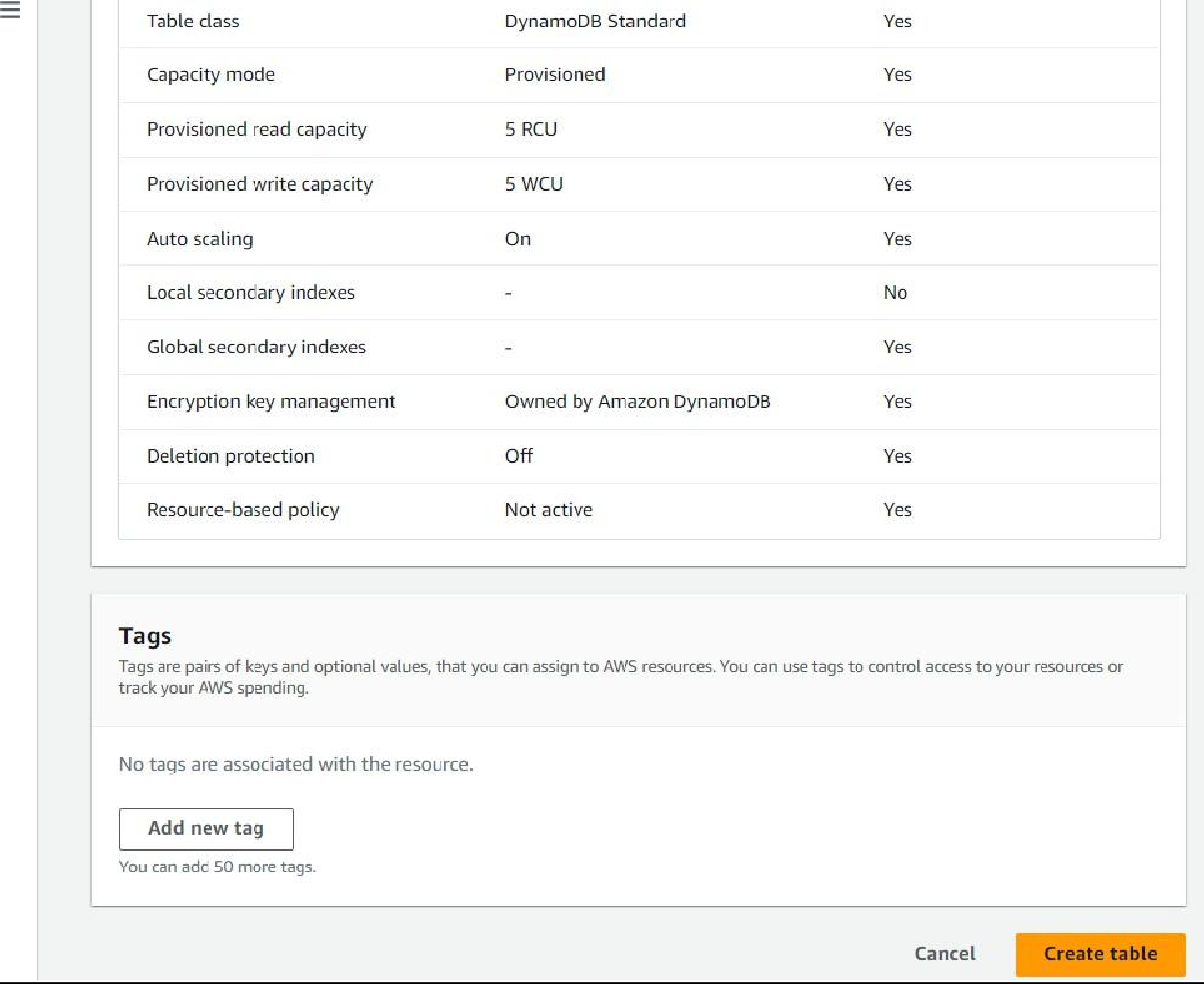
* + **Activity 2.1:Navigate to the DynamoDB**
    - **In the AWS Console, navigate to DynamoDB and click on create tables.**
    - 

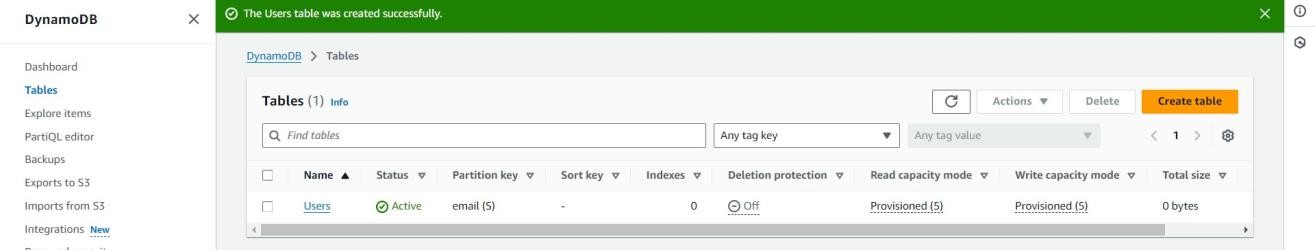




* + **Activity 2.2:Create a DynamoDB table for storing registration details and book requests.**
    - **Create Users table with partition key “Email” with type String and click on create tables.**

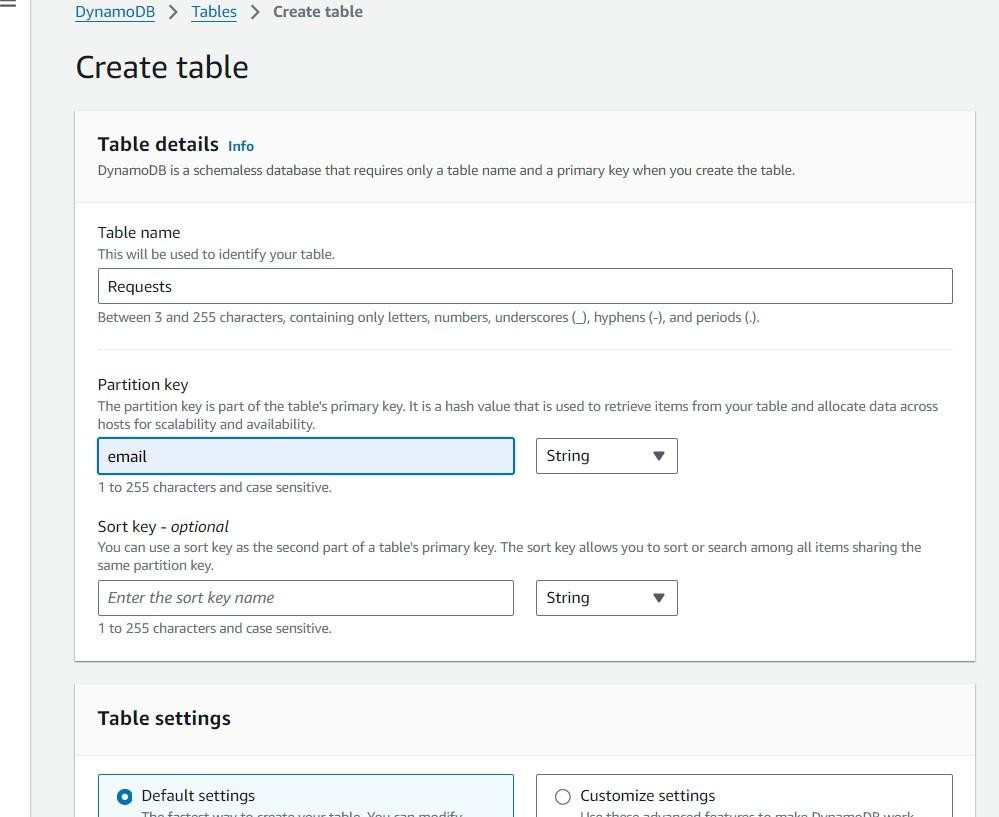


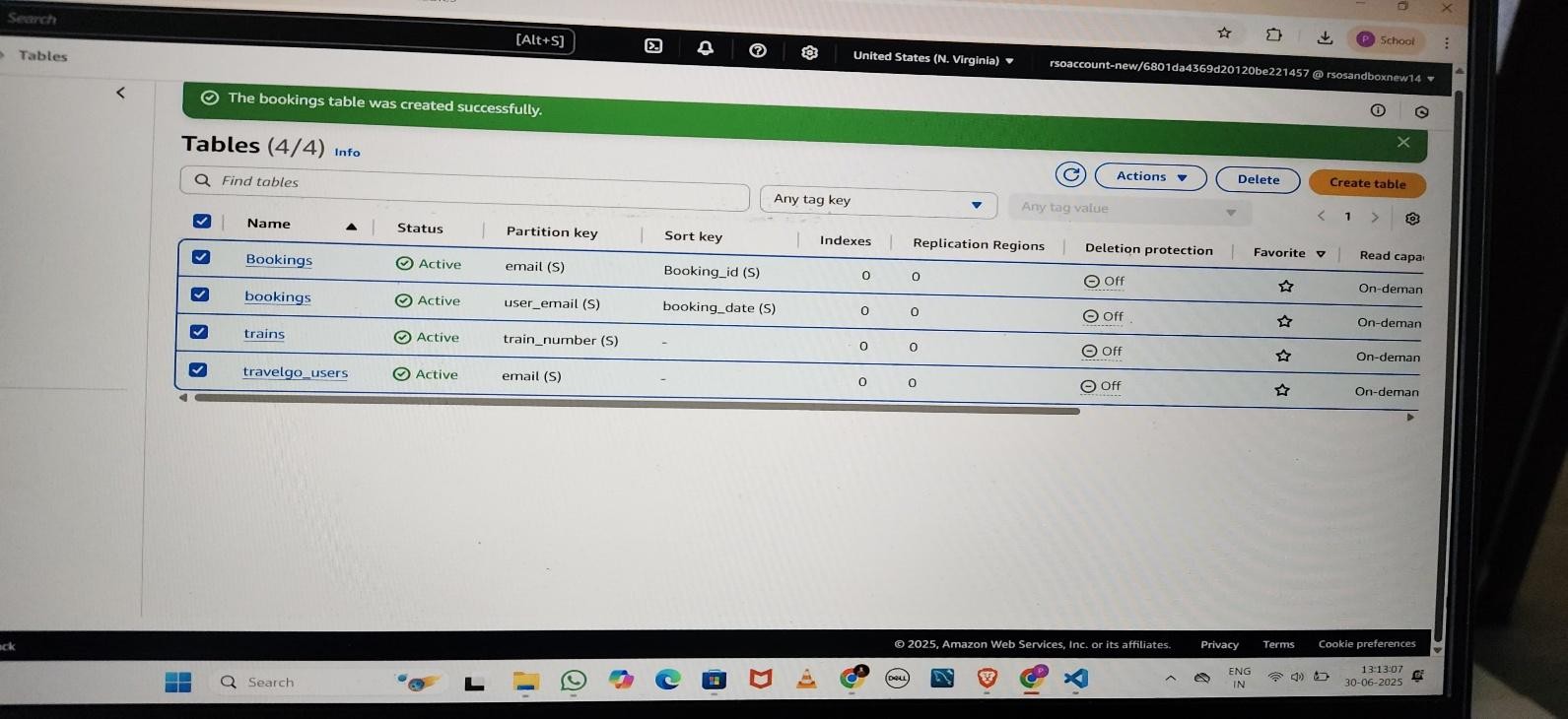


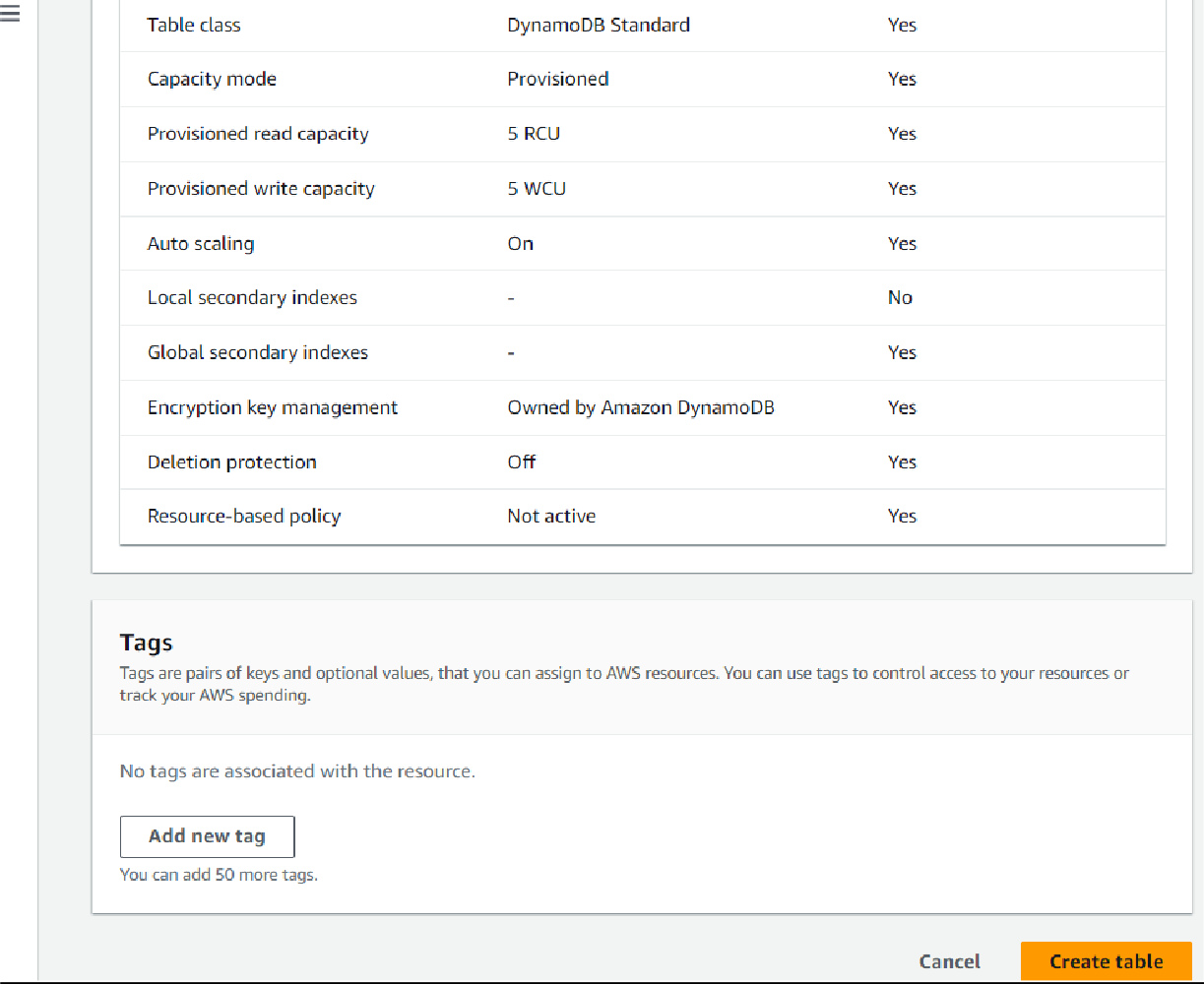


* + - **Follow the same steps to create a requests table with Email as the primary key for book requests data.**



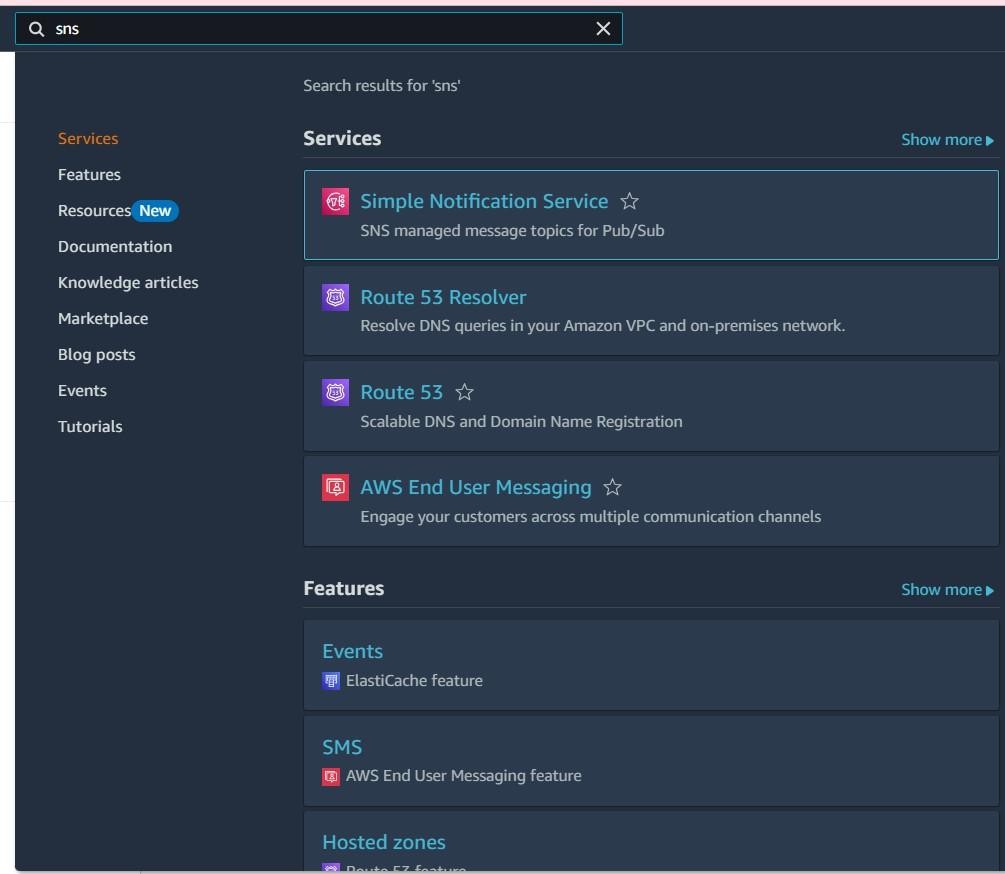


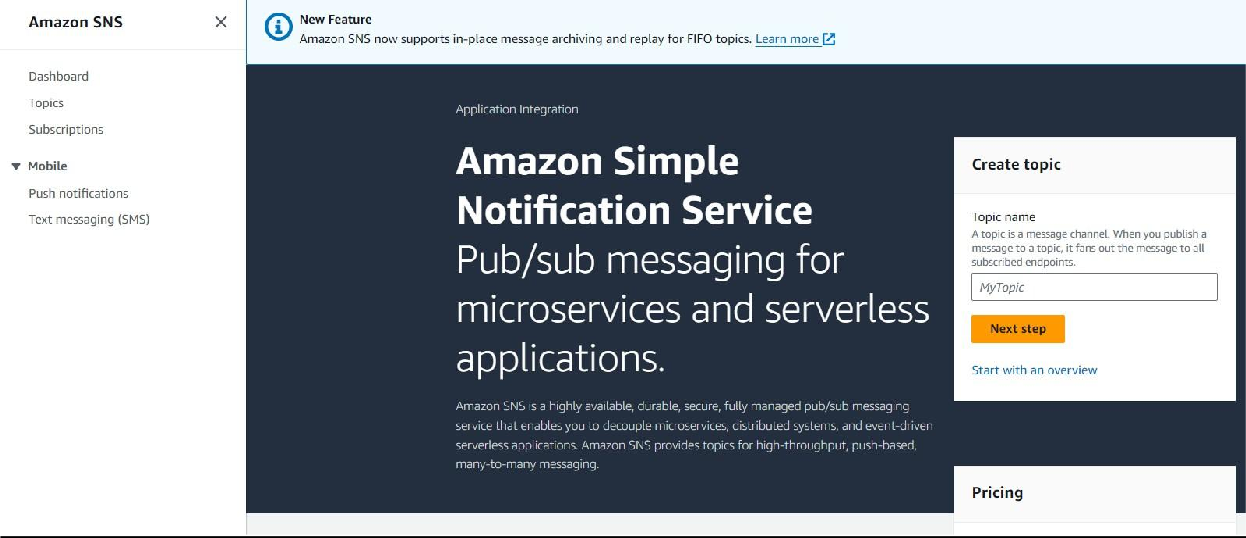




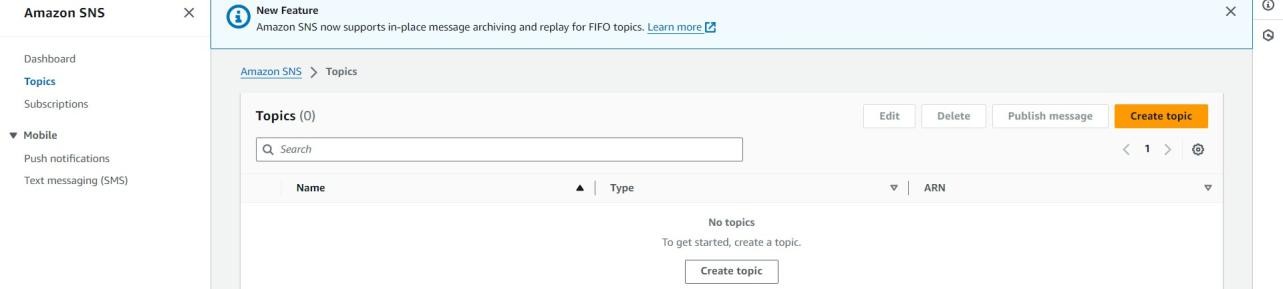
### Milestone 3: SNS Notification Setup

* **Activity 3.1: Create SNS topics for sending email notifications to users and library staff.**
  + **In the AWS Console, search for SNS and navigate to the SNS Dashboard.**

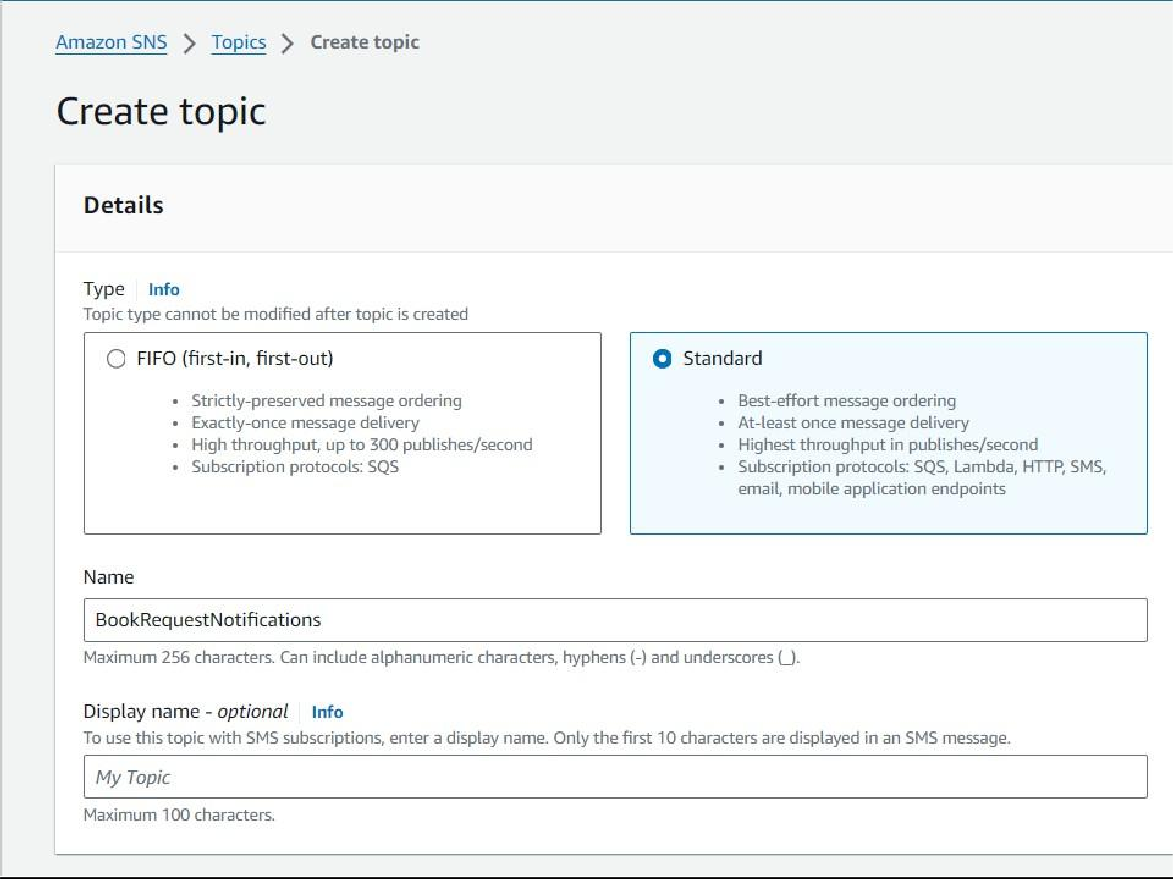


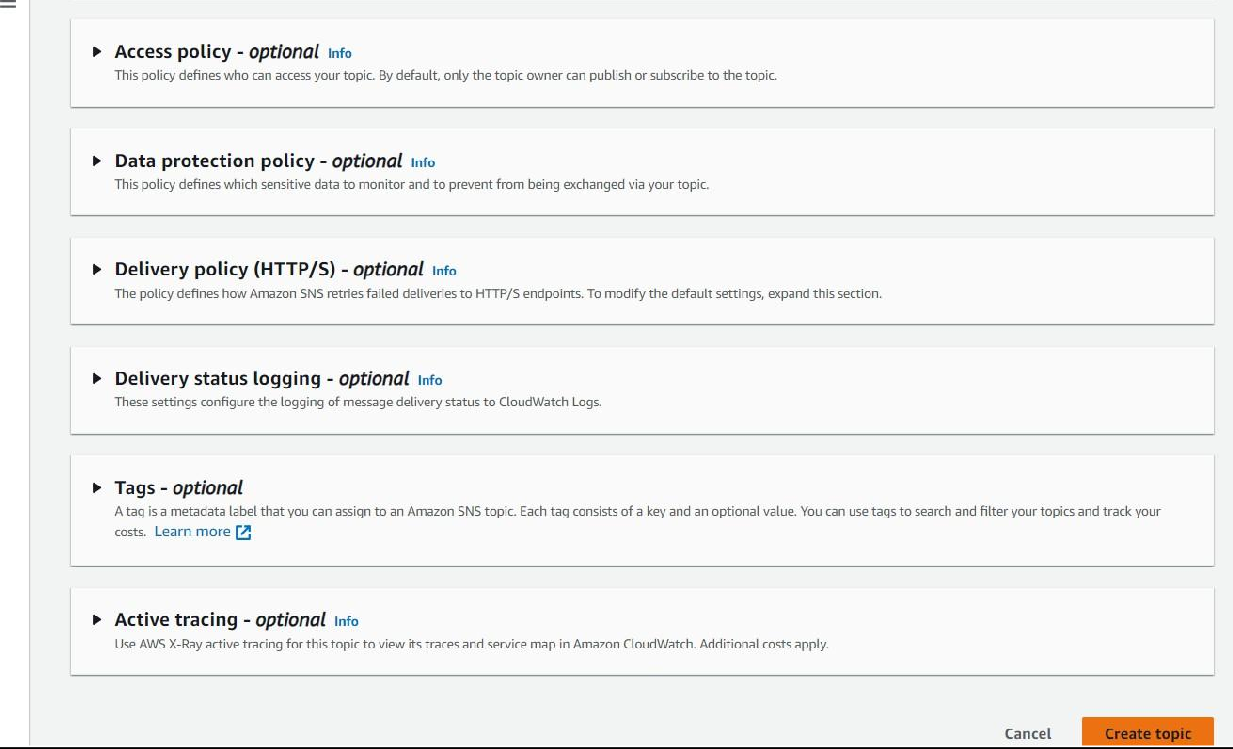


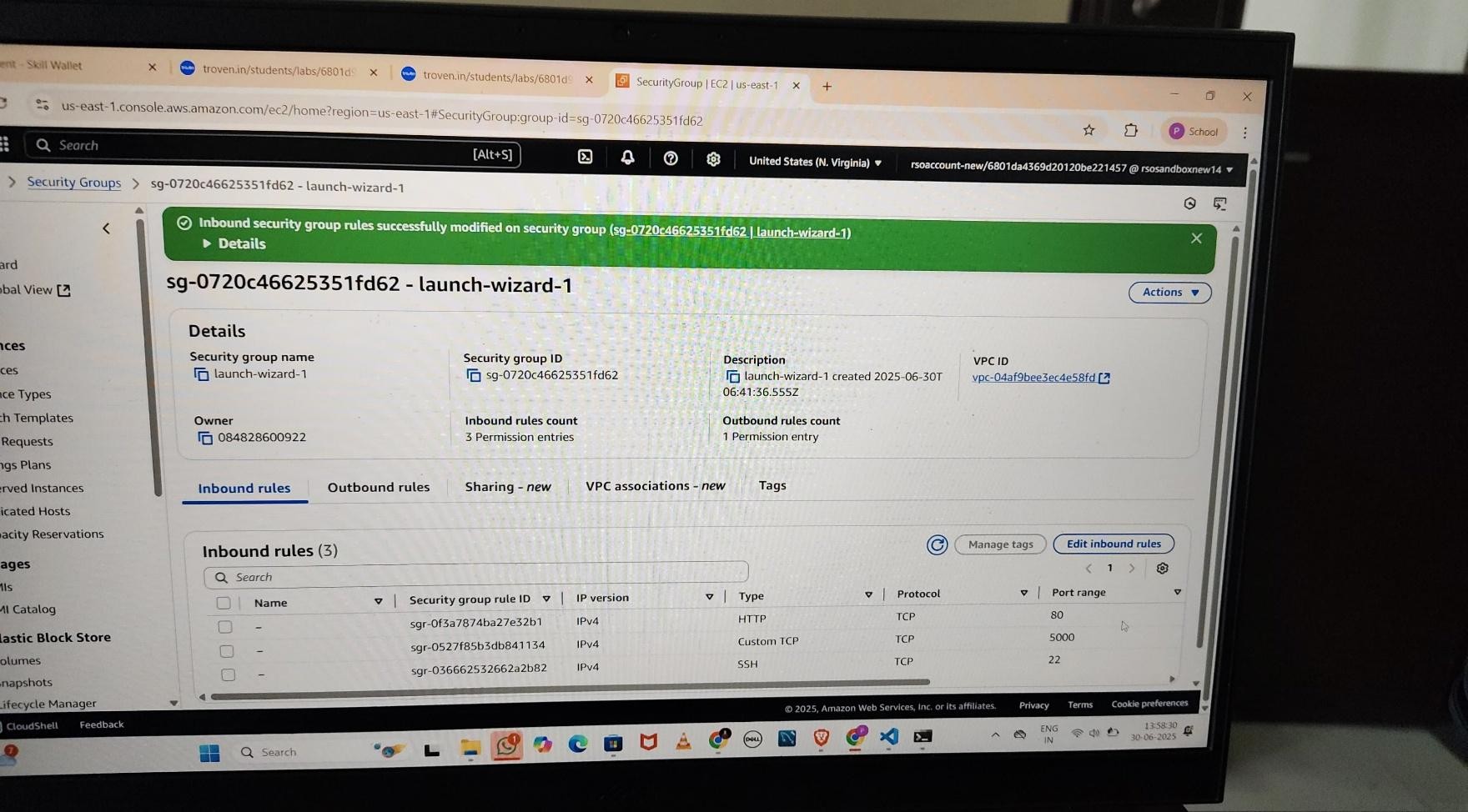
* + **Click on Create Topic and choose a name for the topic.**

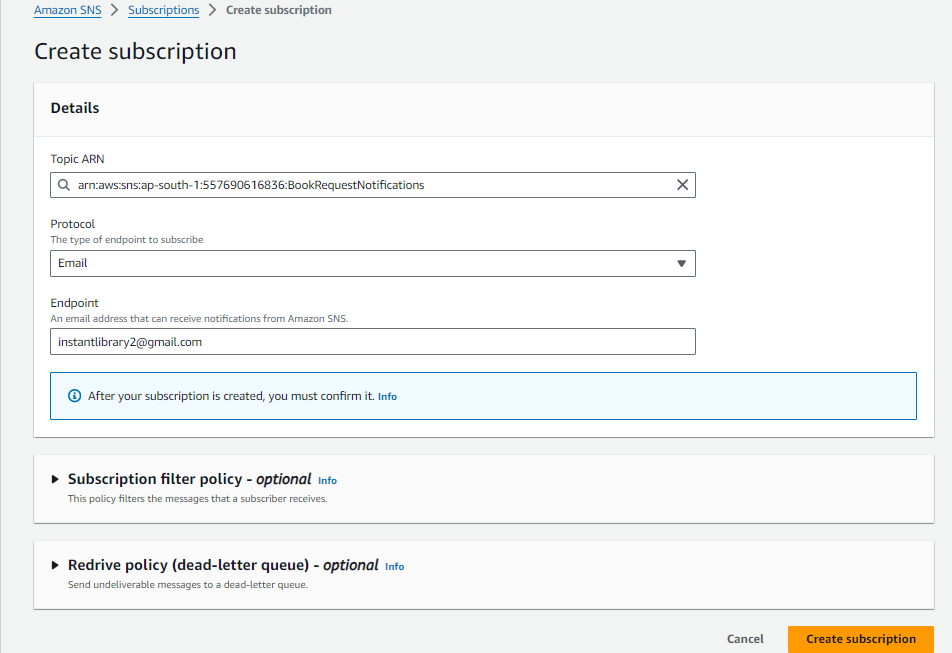


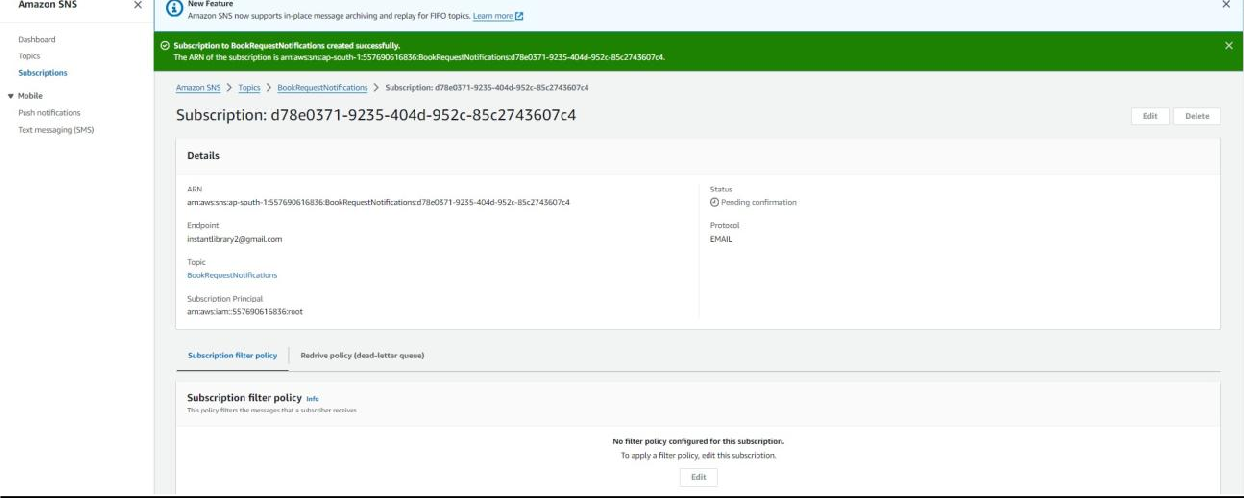
* + **Choose Standard type for general notification use cases and Click on Create Topic.**



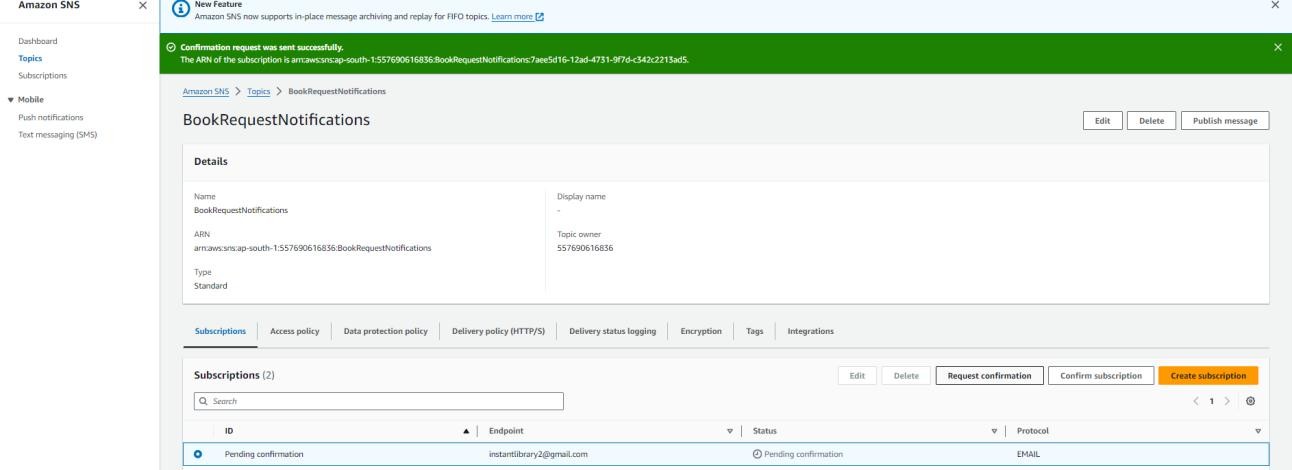


* + **Configure the SNS topic and note down the Topic ARN.**
* **Activity 3.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.**
  + **Subscribe users (or admin staff) to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails.**

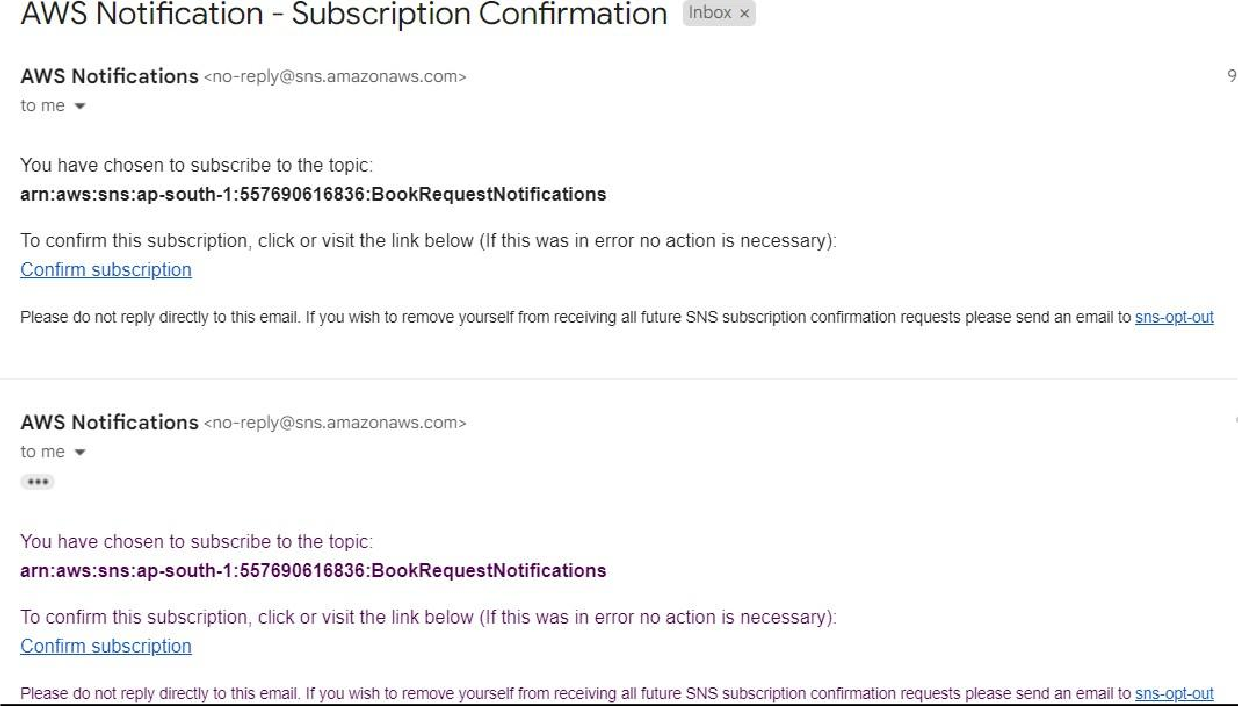


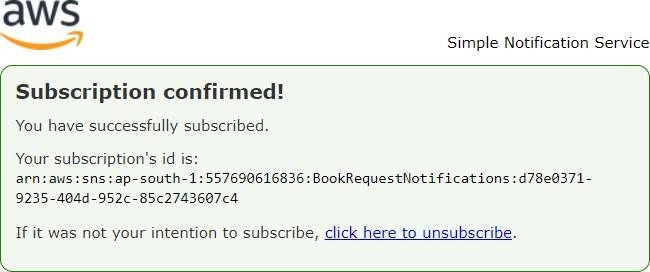


* + **After subscription request for the mail confirmation**

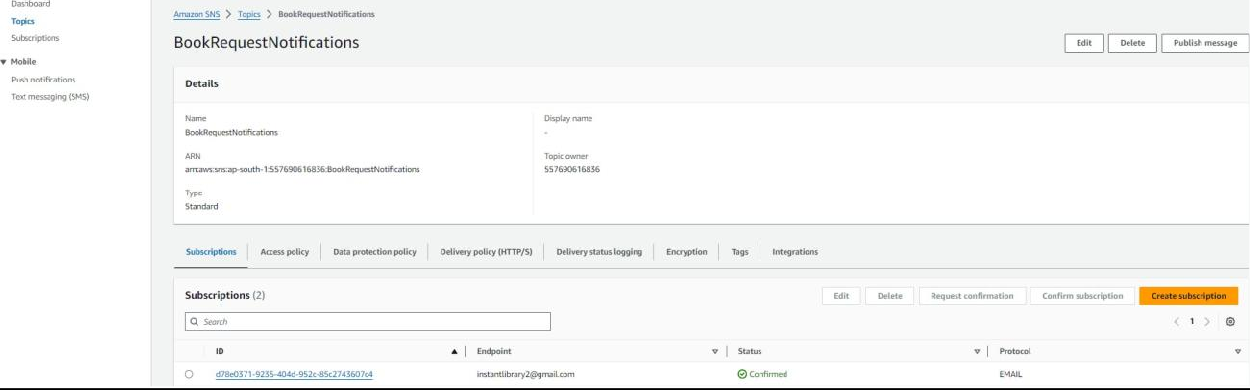


* + **Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.**





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



## Milestone 4:Backend Development and Application Setup

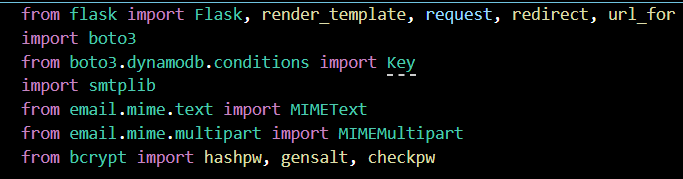
* **Activity 4.1: Develop the backend using Flask**
  + **File Explorer Structure**



Description: set up the INSTANT LIBRARY 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., computer\_science.html, data\_science.html), and utility pages (e.g., request- form.html, statistics.html).

### Description of the code :

* **Flask App Initialization**

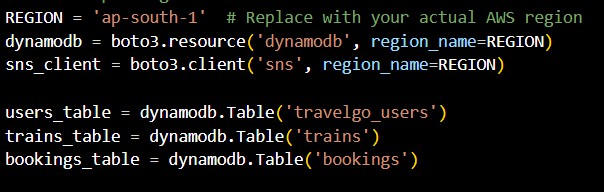


Description: import essential libraries including Flask utilities for routing, Boto3 for DynamoDB operations, SMTP and email modules for sending mails, and Bcrypt for password hashing and verification



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

* **Dynamodb Setup:**



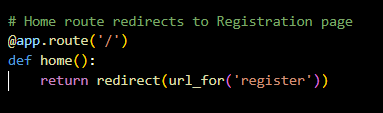
Description: initialize the DynamoDB resource for the ap-south-1 region and set up access to the Users and Requests tables for storing user details and book requests.

* **SNS Connection**

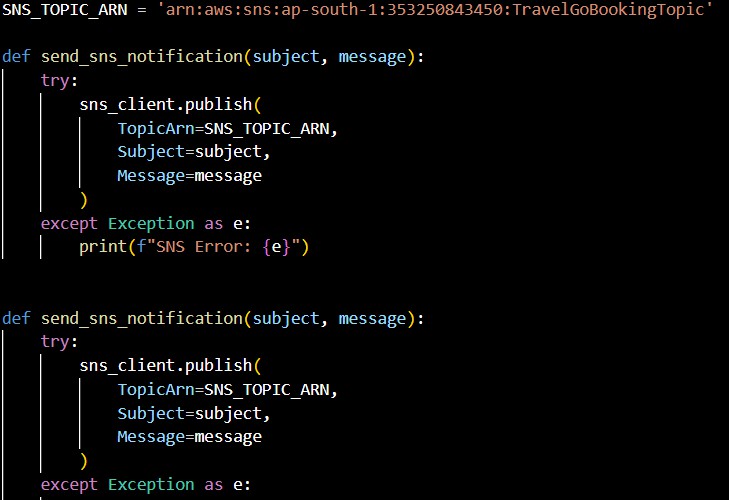
Description: Configure SNS to send notifications when a book request is submitted. Paste your stored ARN link in the sns\_topic\_arn space, along with the region\_name where the SNS topic is created. Also, specify the chosen email service in SMTP\_SERVER (e.g., Gmail, Yahoo, etc.) and enter the subscribed 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

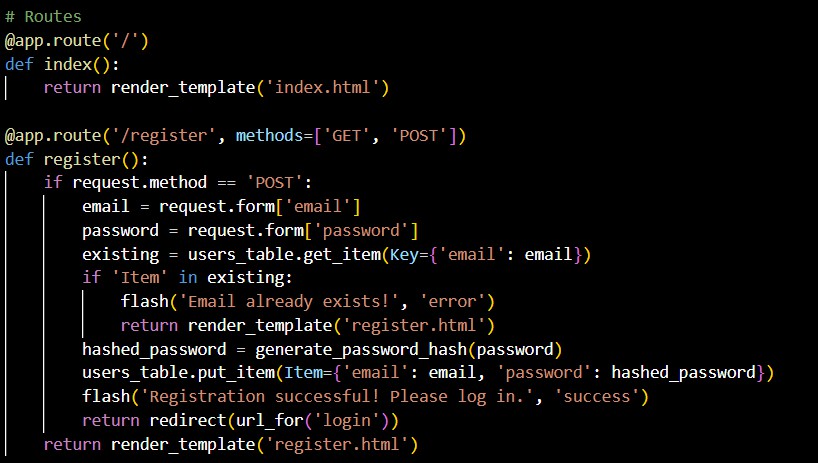
* + **Home Route:**



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

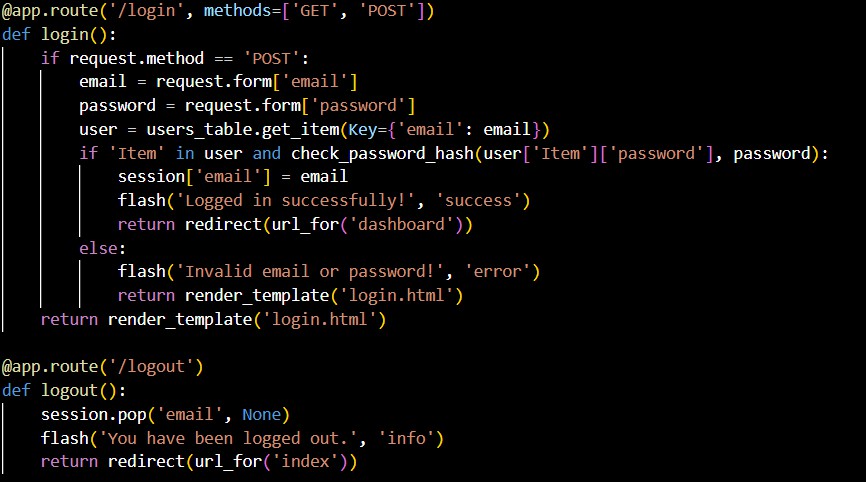
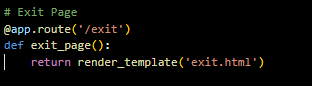


* + - **Register Route:**

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):**

**Exit Route:**

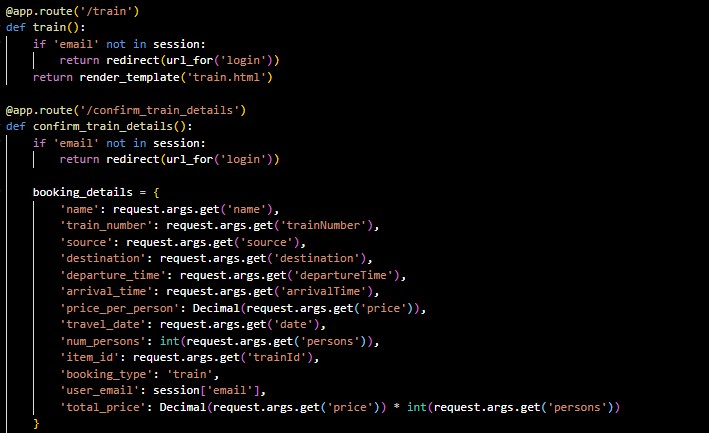


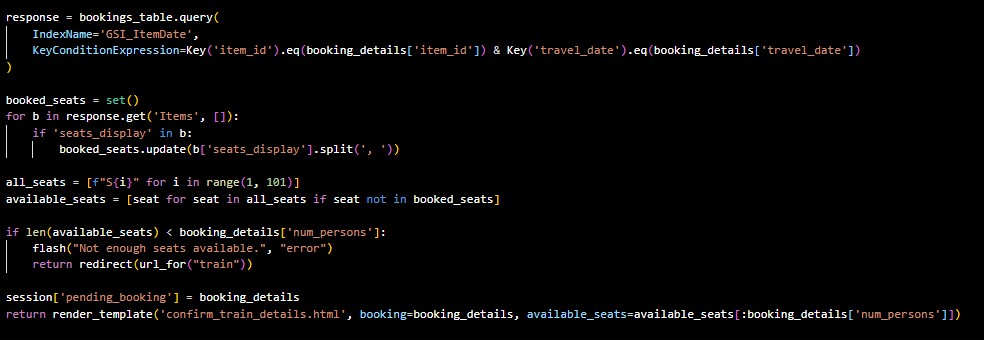
Description: define /exit route to render the exit.html page when the user chooses to leave or

Logout and Dashboard :-

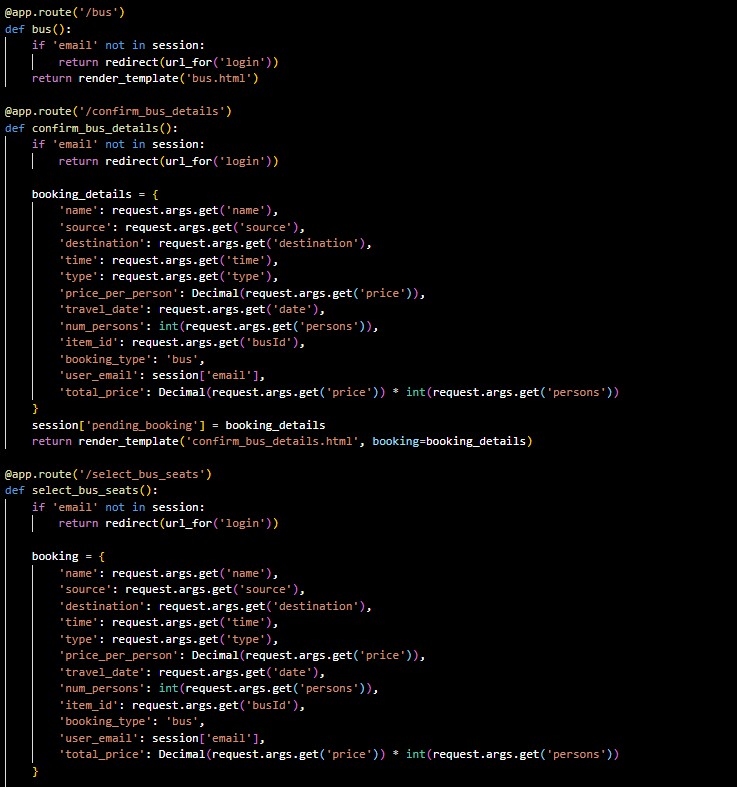


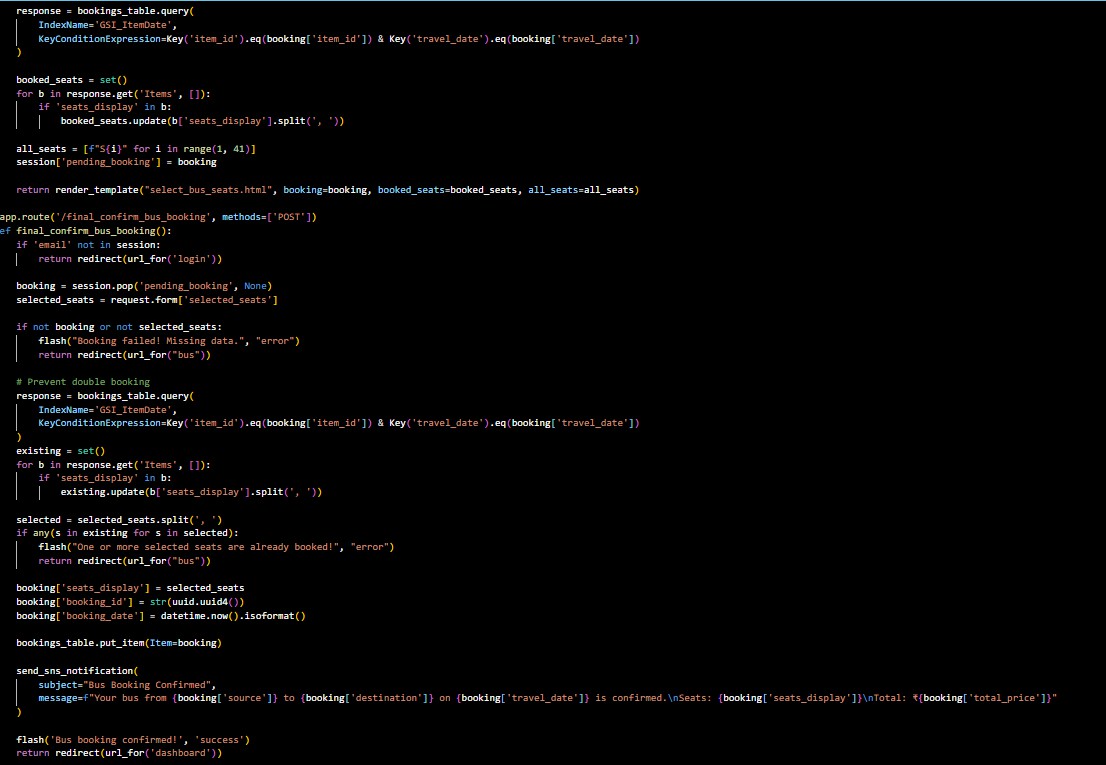
Train :-





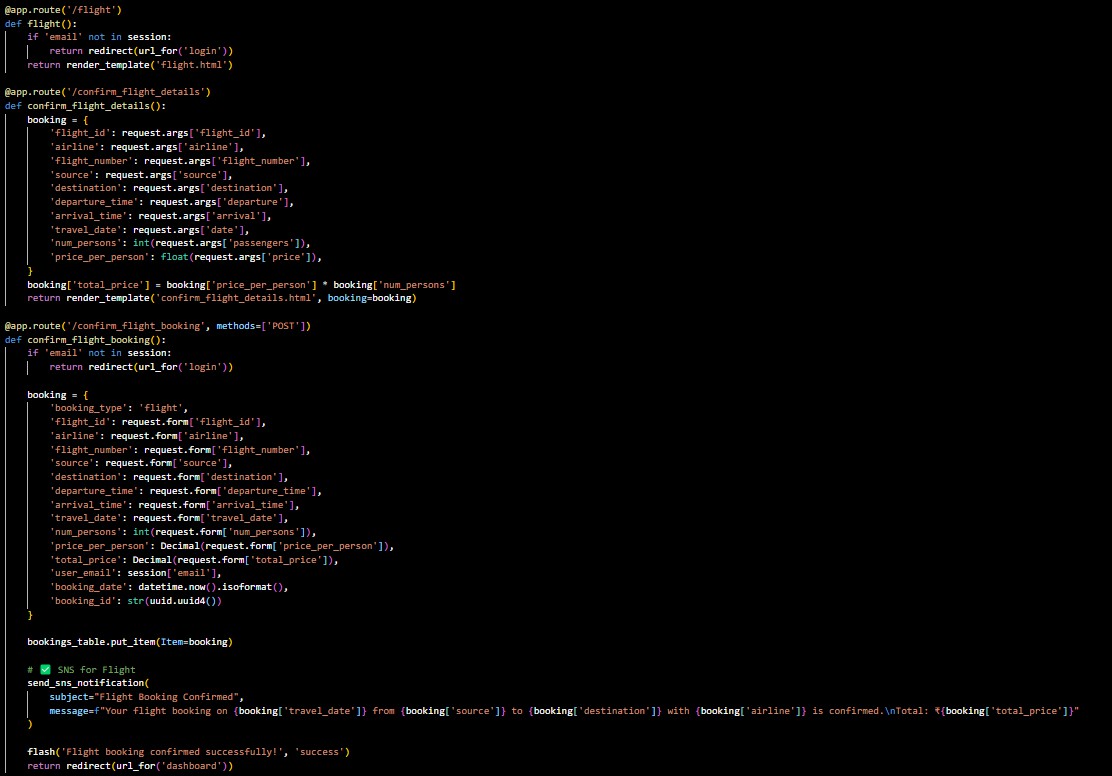
Buses:-



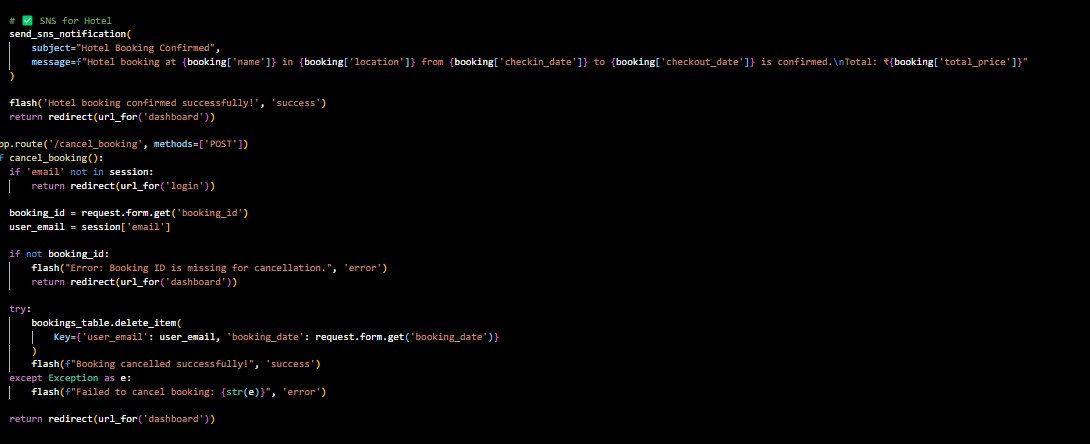
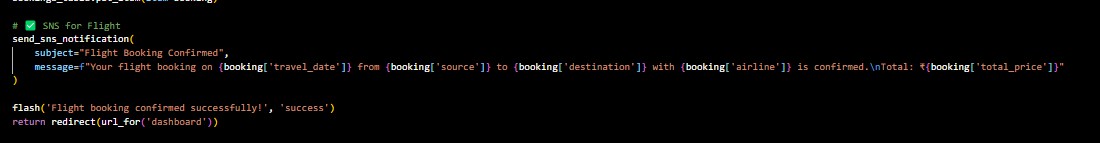


Flight :

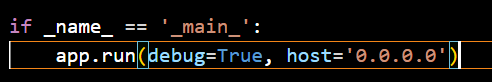




Response :-



Deployment Code:

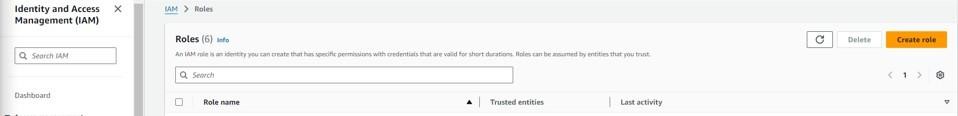
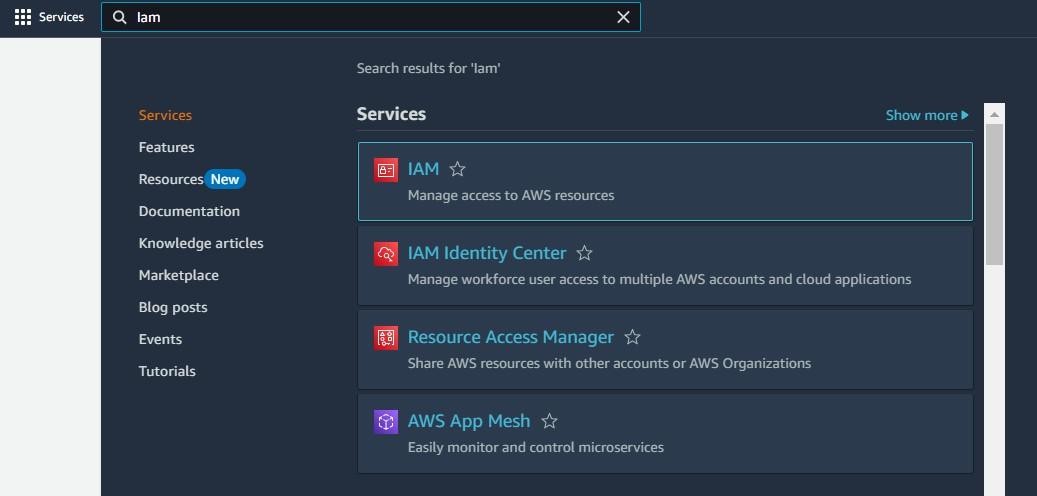


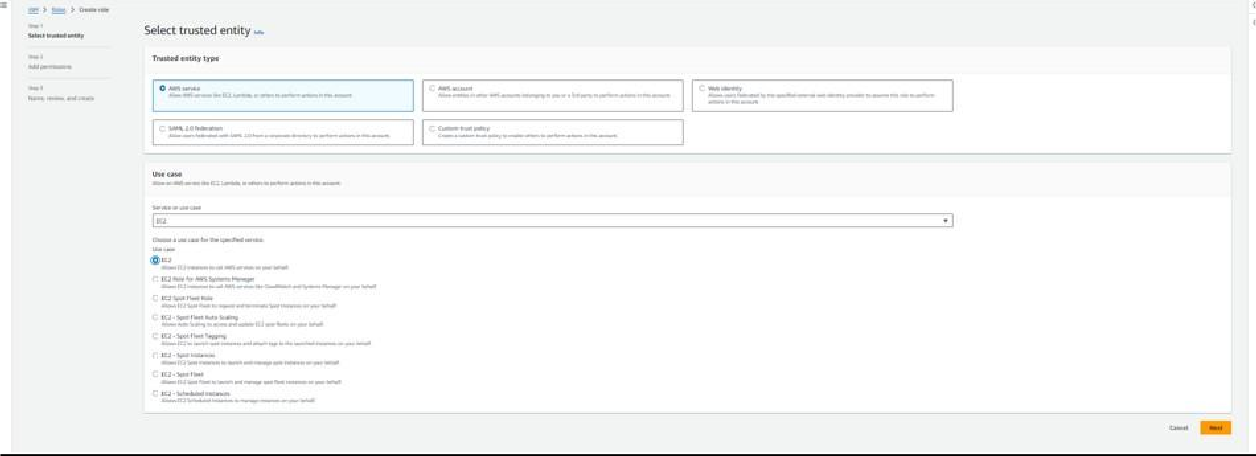
Description: start the Flask server to listen on all network interfaces (0.0.0.0) at port 80

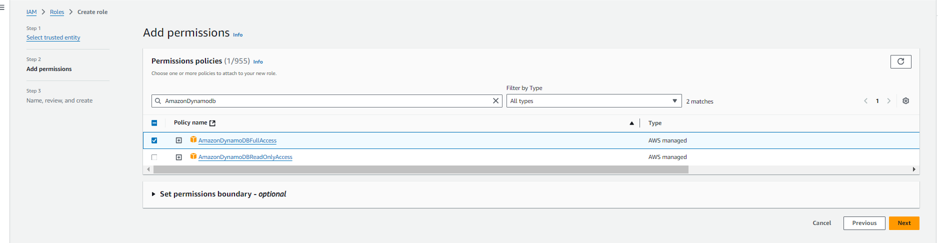
with debug mode enabled for development and testing.

## 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.**



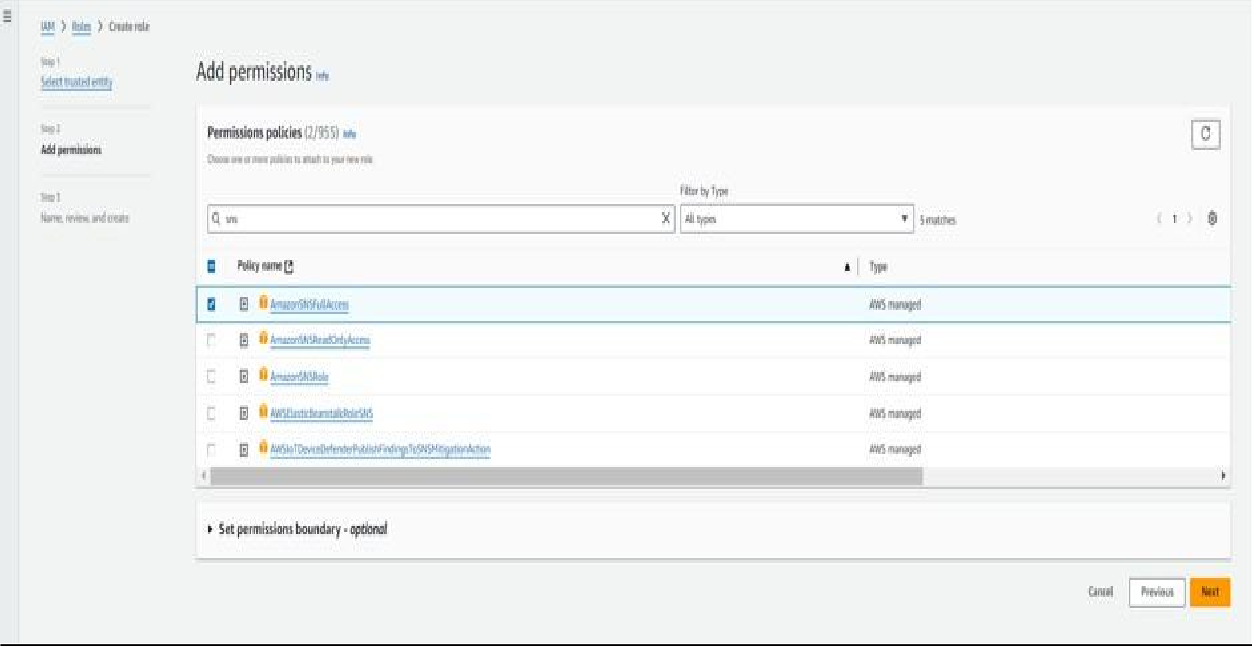


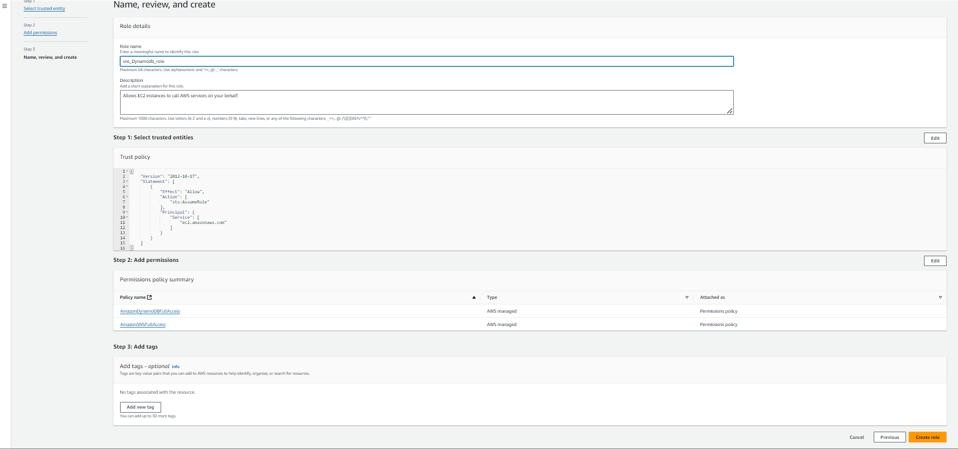


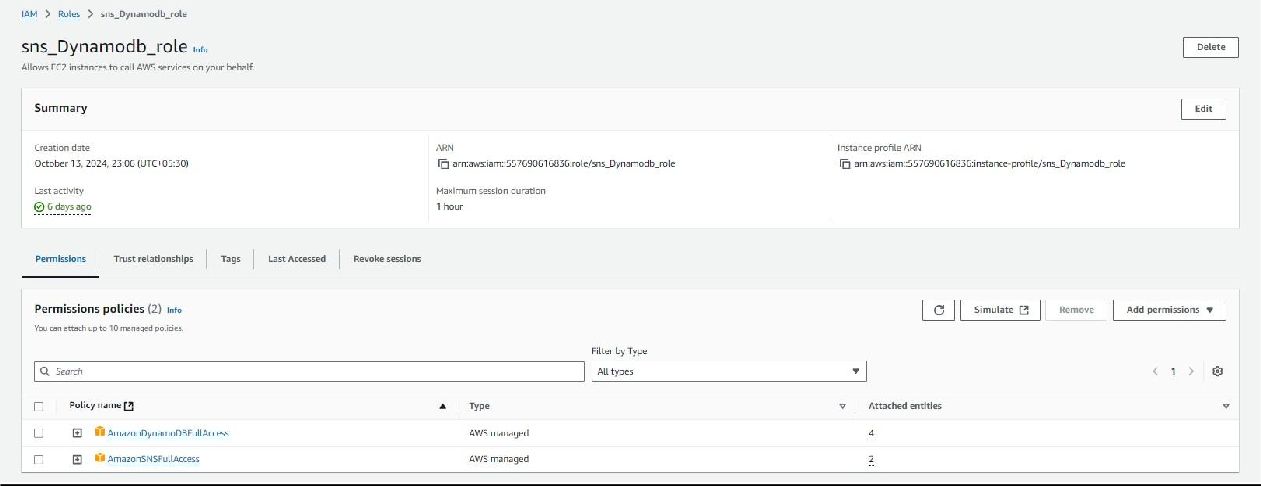
* **Activity 5.2: Attach Policies.**

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.**

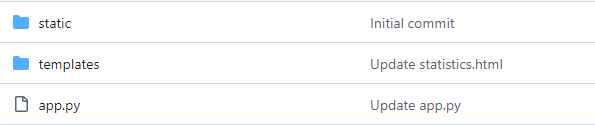


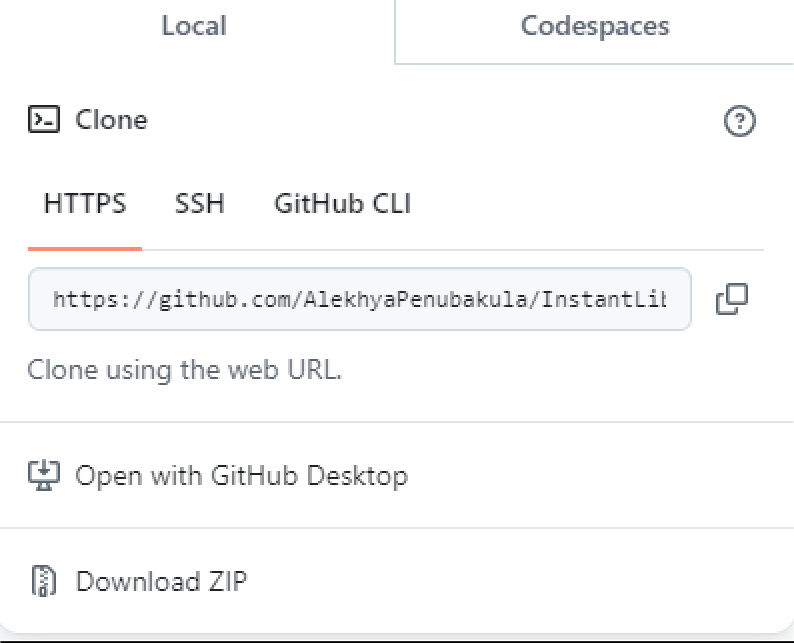




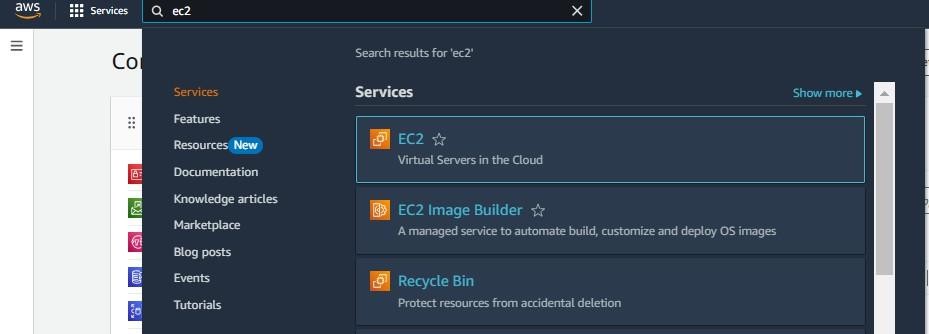
## Milestone 6: EC2 Instance Setup

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

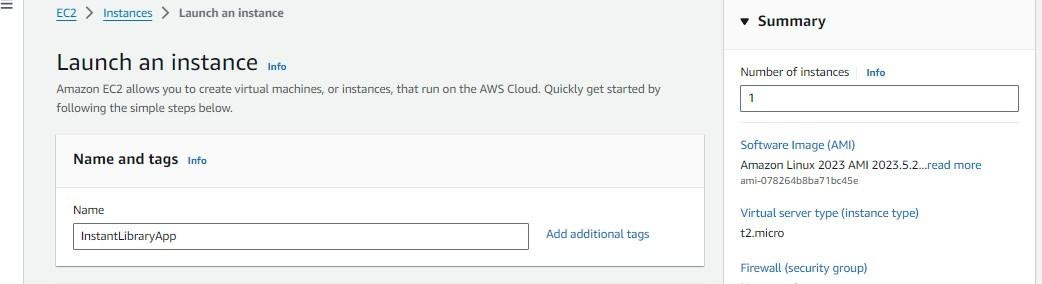
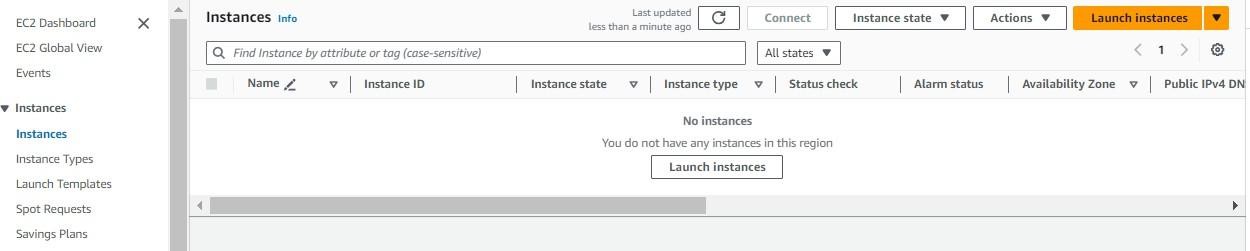




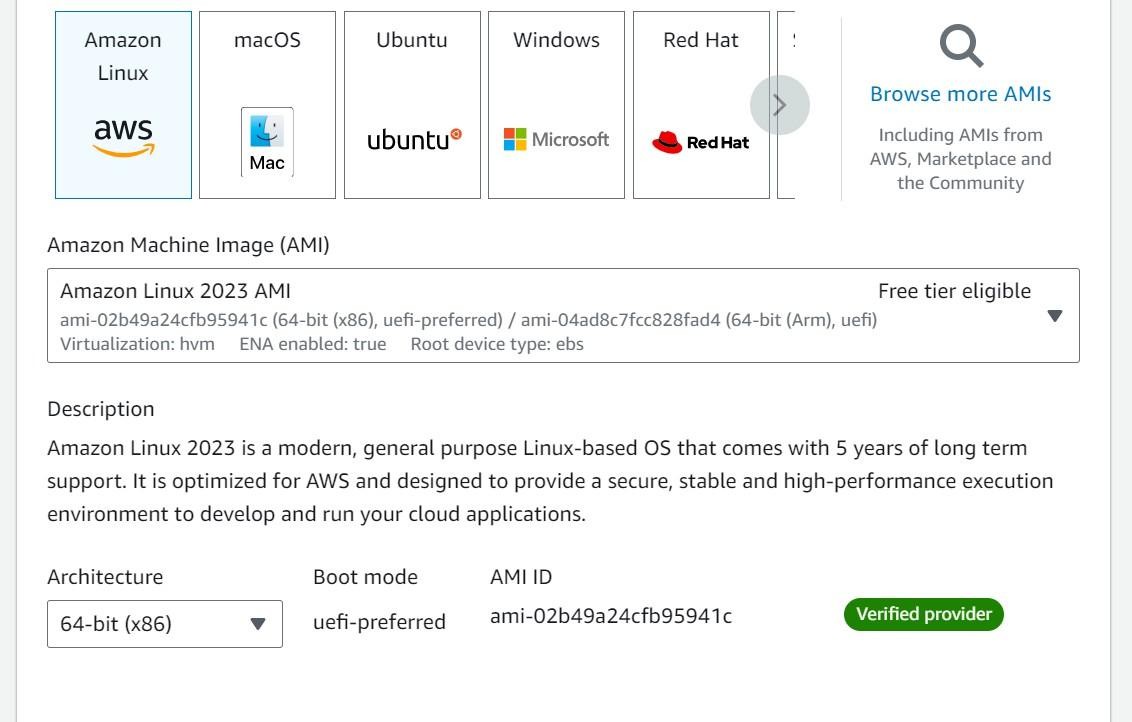
* **Activity 6.1: Launch an EC2 instance to host the Flask application.**
  + **Launch EC2 Instance**
    - **In the AWS Console, navigate to EC2 and launch a new instance.**



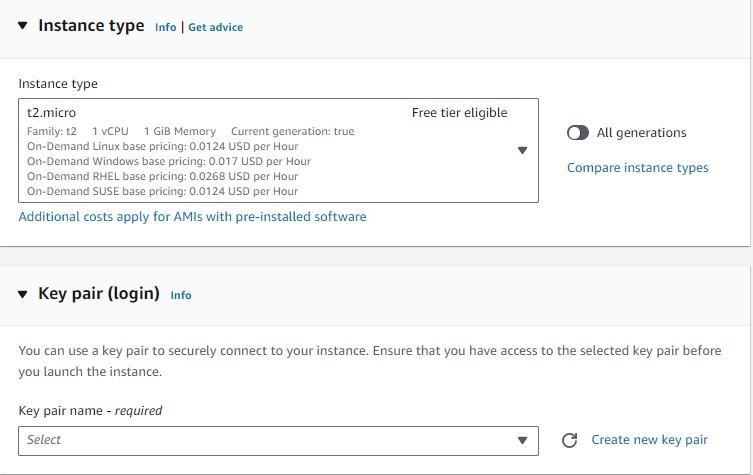
* **Click on Launch instance to launch EC2 instance**

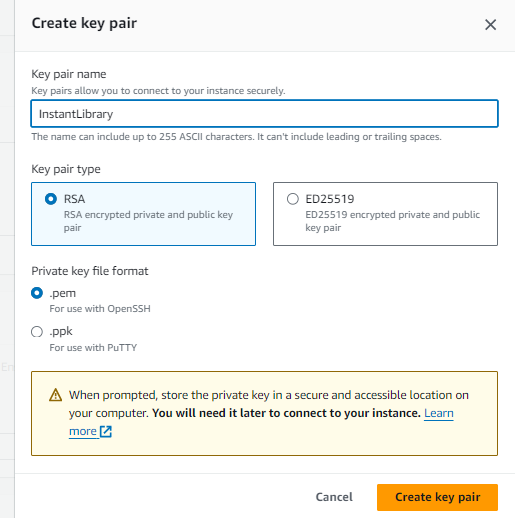


* + **Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).**

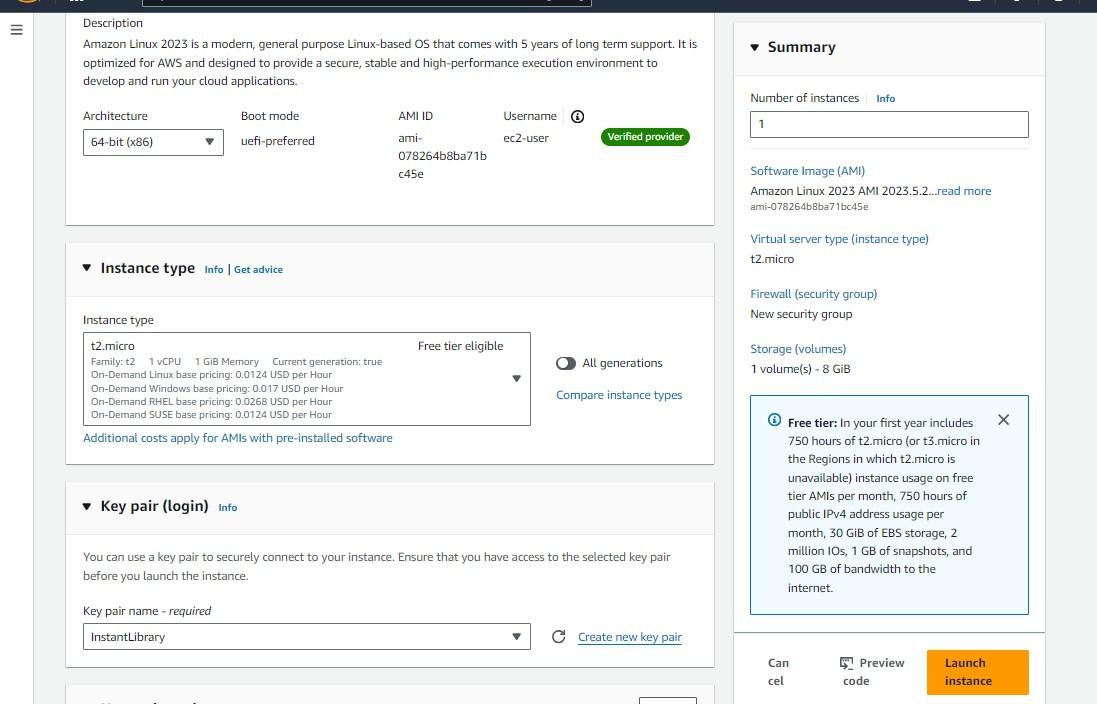


* **Create and download the key pair for Server access.**

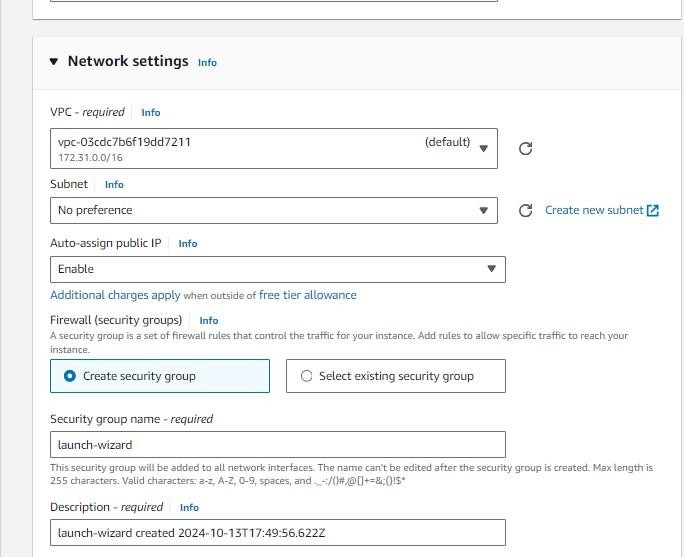


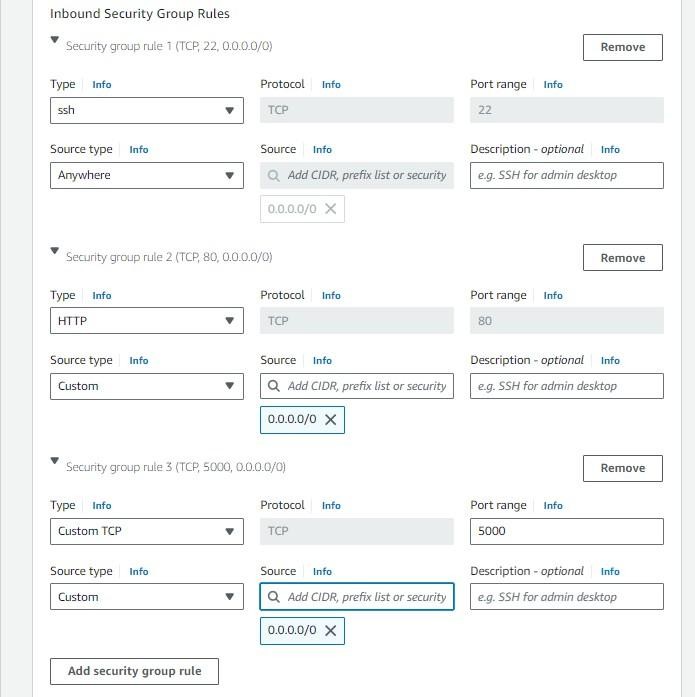


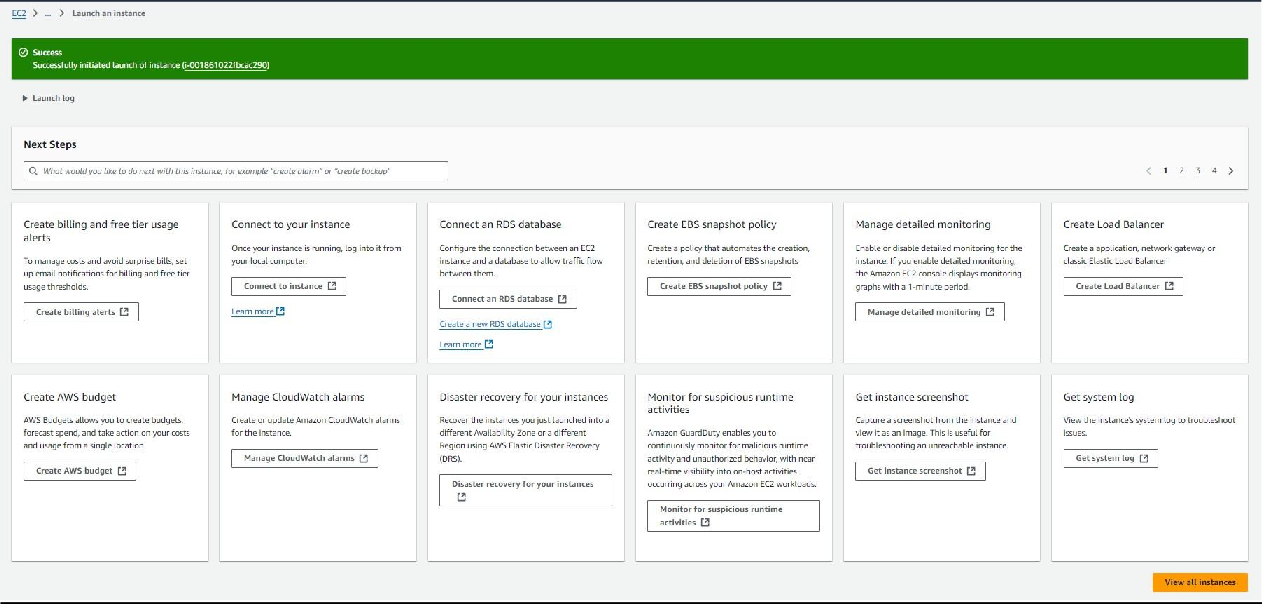




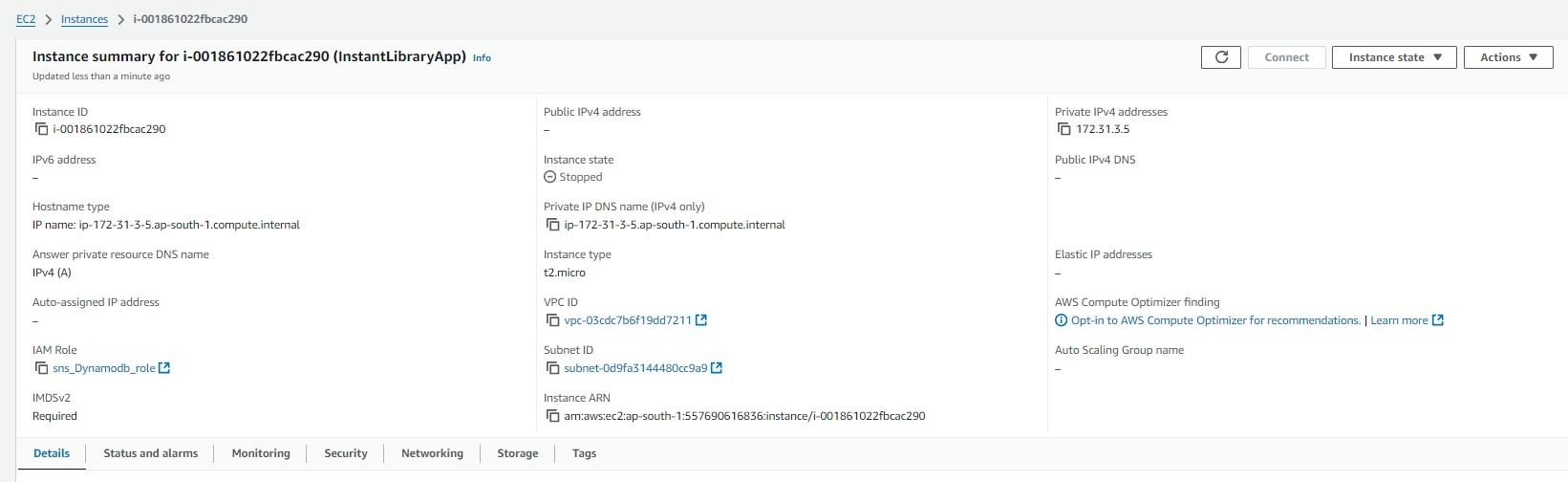
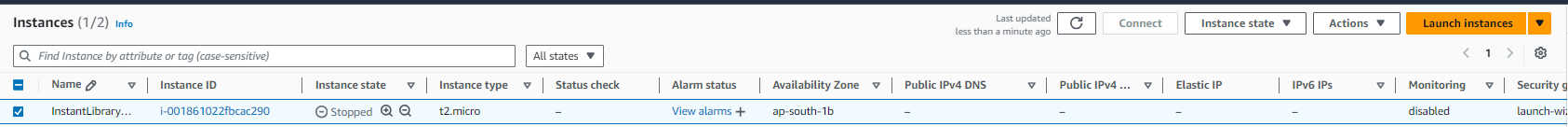
* **Activity 6.2:Configure security groups for HTTP, and SSH access.**

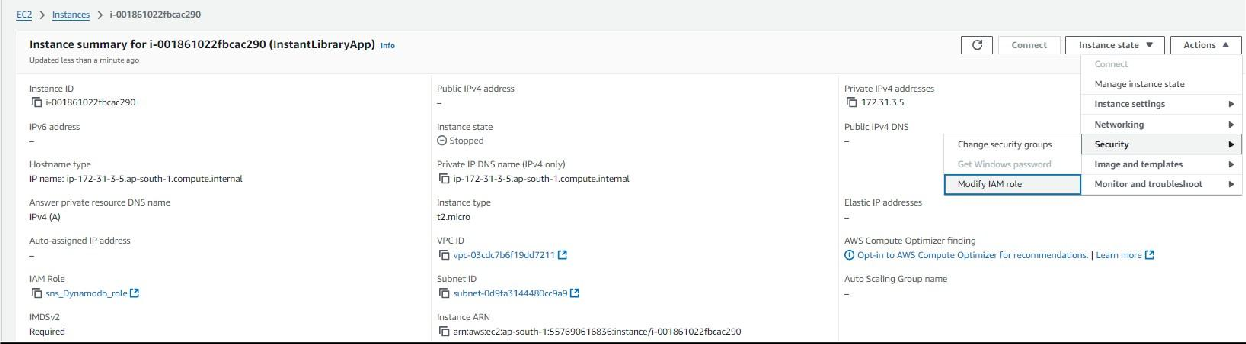


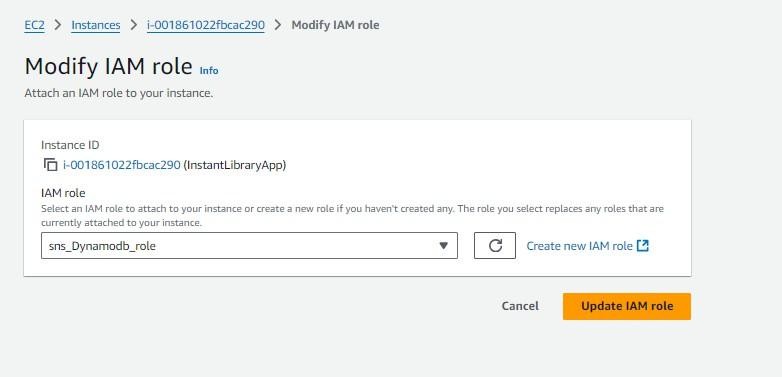




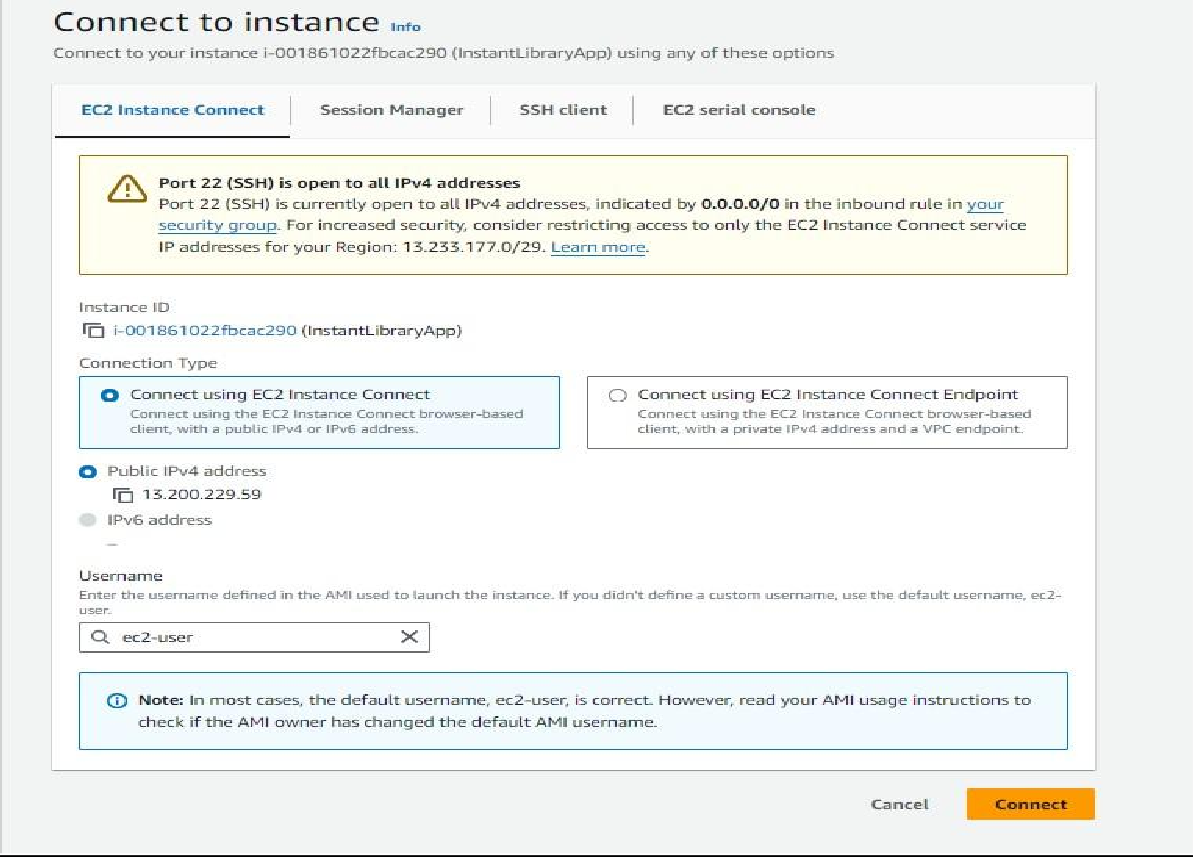
* **To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and selecting Modify IAM Role to attach the appropriate role. After the IAM role is connected,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

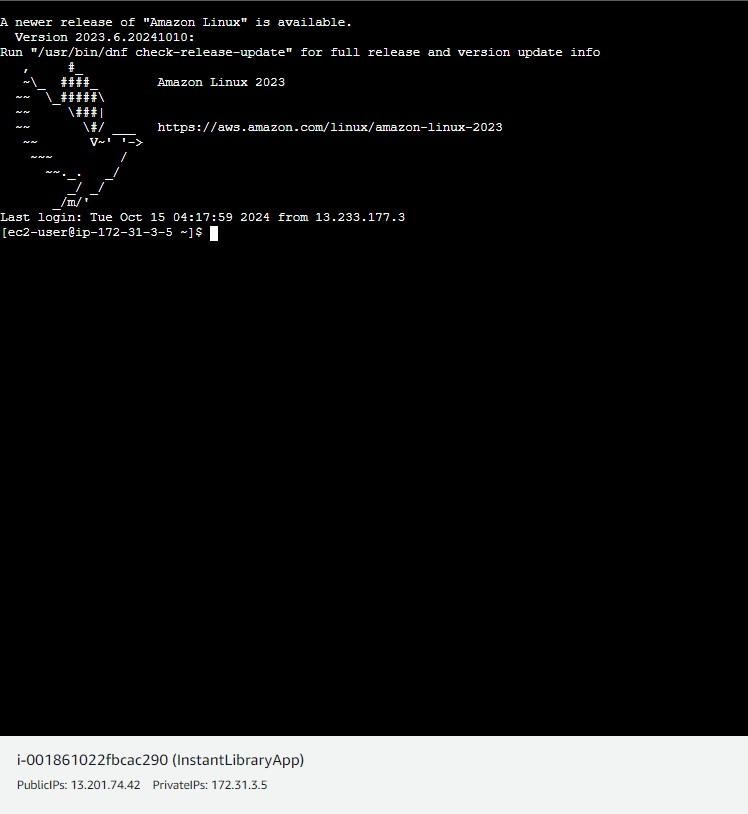






* **Now connect the EC2 with the files**





**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 clonehttps://github.com/vamsi200527/travelgoaws**

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

**credentials here:**

**‘git clone https://github.com/SoftwareKarthik/travelgoproject**

* **This will download your project to the EC2 instance.**

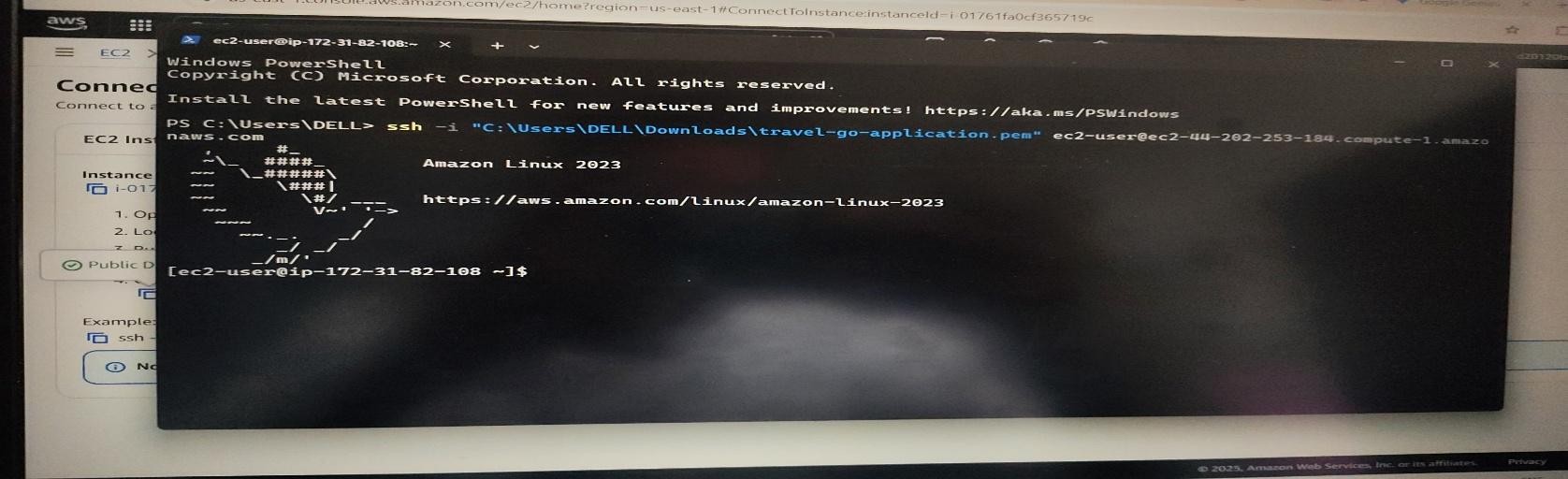
**To navigate to the project directory, run the following command:**

**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-**](http://your-ec2-public-ip/)[**ip**](http://your-ec2-public-ip/)

### Run the Flask app on the EC2 instance



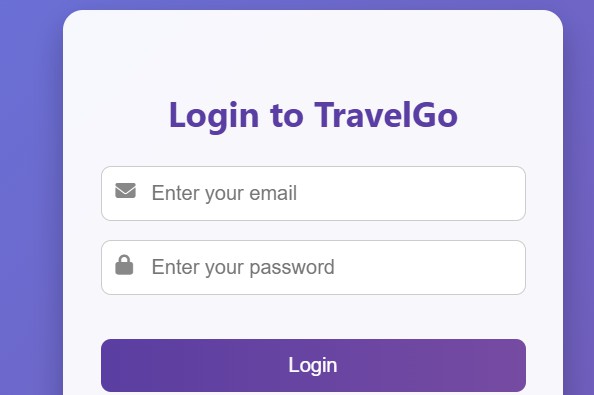
**Access the website through:**

<http://172.31.16.68:5000/>

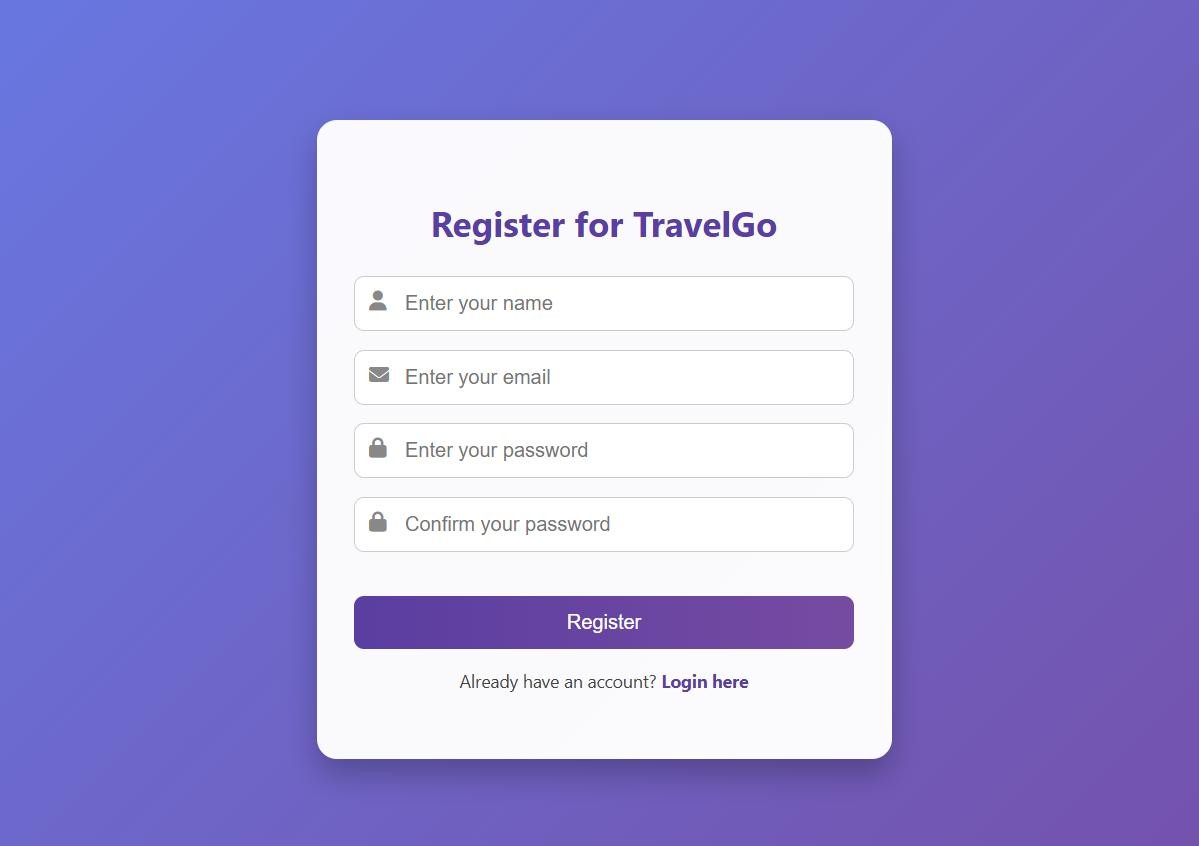
### Milestone 8: Testing and Deployment

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

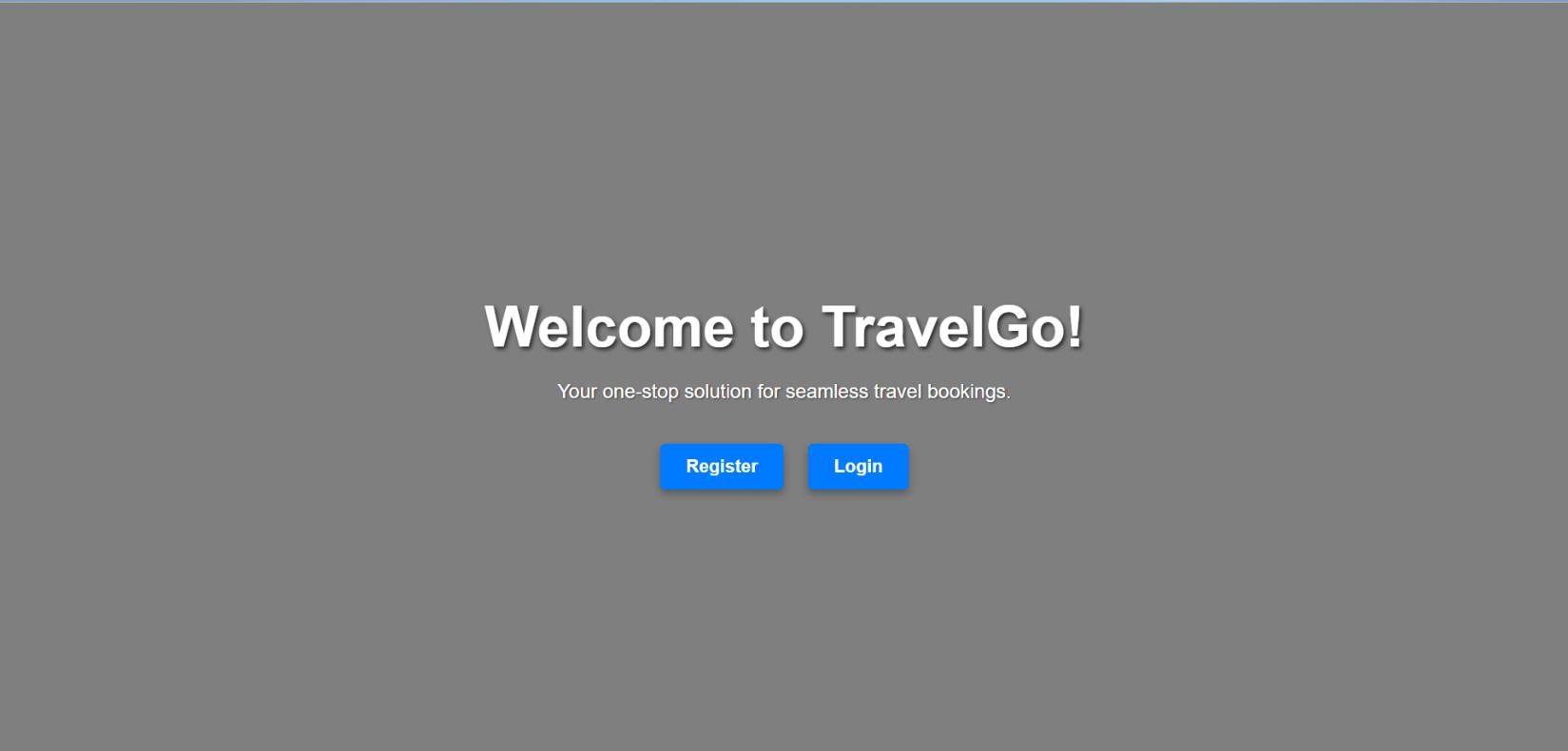
Login Page:



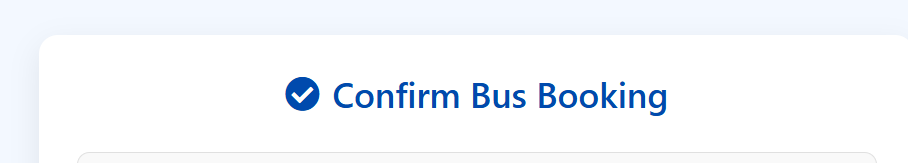
# Registration Page:



**Front Page:-**



# Confirm Booking Page:



## Conclusion:

The Greenfield University Travel Booking System has been successfully developed and deployed using a robust and scalable cloud architecture. By integrating Flask, AWS EC2, DynamoDB, and SNS, the platform offers a seamless experience for students and faculty to manage academic travel requests and bookings.

This cloud-based solution overcomes the limitations of traditional, manual travel management by automating booking processes, ensuring real-time communication, and enabling efficient tracking of travel data. The use of DynamoDB guarantees fast and secure storage, while AWS SNS keeps all stakeholders informed with instant email notifications.

The system’s responsive interface, backed by scalable AWS services, ensures smooth operation even under high usage. From registration and login to travel request submissions and status tracking, all functionalities have been thoroughly tested and optimized.

In summary, this platform significantly improves the efficiency and transparency of academic travel planning at Greenfield University, demonstrating the power of modern cloud technologies in solving real-world administrative challenges.

Experience Gained :-

Working on the Greenfield University Travel Booking System project provided valuable hands-on experience across multiple areas of cloud computing, backend development, and full-stack deployment. The following skills and insights were gained:

 Cloud Services Integration

Gained practical experience in setting up and integrating AWS services like **EC2**, **DynamoDB**, and **SNS**, learning how these services interact to support real-world web applications.

 Flask Web Development

Built a scalable backend using **Flask**, implementing routing, user authentication, and session management while connecting it to a cloud-based NoSQL database.

 DynamoDB Database Management

Designed and managed DynamoDB tables, learned how to perform CRUD operations using **boto3**, and understood best practices for data modeling in NoSQL environments.

 Real-time Notification System

Used **AWS SNS** to send email alerts, improving user interaction and system responsiveness. This reinforced the importance of real-time communication in modern applications.

 Deployment on AWS EC2

Learned how to launch, configure, and secure an **EC2 instance**, and successfully deployed a Flask app to a live production environment.

 IAM Roles and Security Best Practices

Understood the importance of **IAM policies and permissions** by configuring roles for EC2 to securely interact with other AWS services.

 Version Control and GitHub Integration

Managed project code using **Git**, enabling better team collaboration and version tracking through GitHub.

 Problem Solving and Debugging

Encountered and resolved various technical challenges during development and deployment, enhancing debugging and troubleshooting skills.

 Project Management and Documentation

Documented the entire workflow, from architecture to deployment, ensuring the project can be maintained, scaled, or enhanced in the future.