**MedTrack: AWS Cloud-Enabled Healthcare Management System**

**1. Project Description**

MedTrack is a full-stack, cloud-enabled healthcare management system using Flask for backend APIs, hosted on AWS EC2, with DynamoDB as its database. It provides a centralized platform for patients and doctors to register, book appointments, view medical history, and receive real-time notifications via AWS SNS. AWS IAM ensures secure, role-based access to system resources.

**Hardware Required:**

Processor: Intel i5 or equivalent (minimum). RAM: 4 GB (8 GB recommended for Full Stack MERN). Storage: 128 GB SSD or 128 GB HDD. Internet Connectivity: High-speed internet (minimum 10 Mbps per system). Additional: Audio-visual setup for interactive sessions (microphone, speakers, etc.).

**Software Required:**

Updated web browser (Google Chrome, Firefox, or Microsoft Edge). Visual Studio Code (or any preferred IDE). Git (latest version).

**System Required:**

Projector and Audio System for presentations in all labs/classrooms Classrooms/Labs are equipped with systems or provisions for students to join sessions with their own laptops.

**Description:**

In today’s fast-evolving healthcare landscape, efficient communication and coordination between doctors and patients are crucial. MedTrack is a cloud-based healthcare management system that streamlines patient doctor interactions by providing a centralized platform for booking appointments, managing medical histories, and enabling diagnosis submissions. To address these challenges, the project utilizes Flask for backend development, AWS EC2 for hosting, and DynamoDB for managing data. MedTrack allows patients to register, log in, book appointments, and submit diagnosis reports online. The system ensures real-time notifications, enhancing communication between doctors and patients regarding appointments and medical submissions. Additionally, AWS Identity and Access Management (IAM) is employed to ensure secure access control to AWS resources, allowing only authorized users to access sensitive data. This cloud-based solution improves accessibility and efficiency in healthcare services for all users.

**Scenarios:**

**Scenario 1: Efficient Appointment Booking System for Patients**

In the MedTrack system, AWS EC2 provides a reliable infrastructure to manage multiple patients accessing the platform simultaneously. For example, a patient can log in, navigate to the appointment booking page, and easily submit a request for an appointment. Flask handles backend operations, efficiently retrieving and processing user data in real-time. The cloud-based architecture allows the platform to handle a high volume of appointment requests during peak periods, ensuring smooth operation without delays.

**Scenario 2: Secure User Management with IAM**

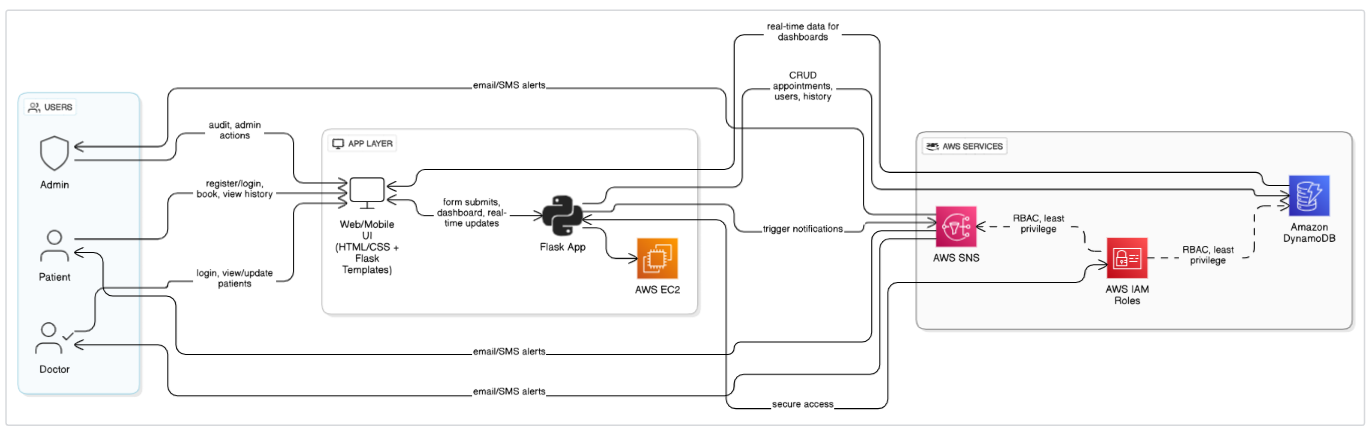
MedTrack utilizes AWS IAM to manage user permissions and ensure secure access to the system. For instance, when a new patient registers, an IAM user is created with specific roles and permissions to access only the features relevant to them. Doctors have their own IAM configurations, allowing them access to patient records and appointment details while maintaining strict security protocols. This setup ensures that sensitive data is accessible only to authorized users.

**Scenario 3: Easy Access to Medical History and Resources**

The MedTrack system provides doctors and patients with easy access to medical histories and relevant resources. For example, a doctor logs in to view a patient's medical history and upcoming appointments. They can quickly access, and update records as needed. Flask manages real-time data fetching from DynamoDB, while EC2 hosting ensures the platform performs seamlessly even when multiple users access it simultaneously, offering a smooth and uninterrupted user experience.

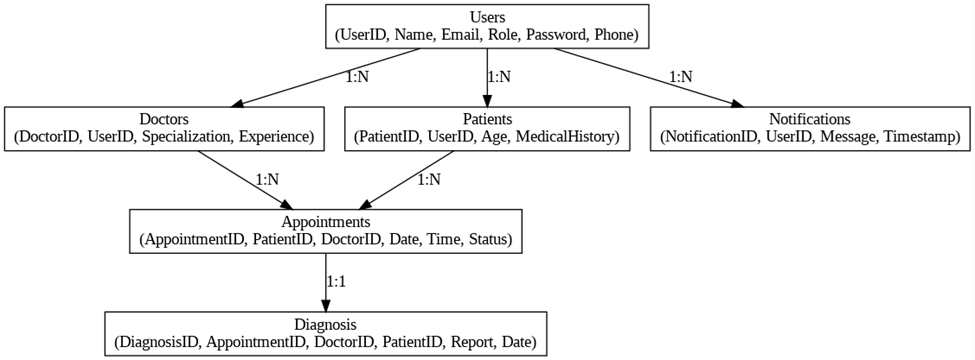
**Architecture**

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**

An ER (Entity-Relationship) diagram visually represents the logical structure of a database by defining entities, their attributes, and the relationships between them. It helps organize data efficiently by illustrating how different components of the system interact and relate. This structured approach supports effective database normalization, data integrity, and simplified query design.



**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/Introduction.html](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. Web Application Development and 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.
* Login to 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.

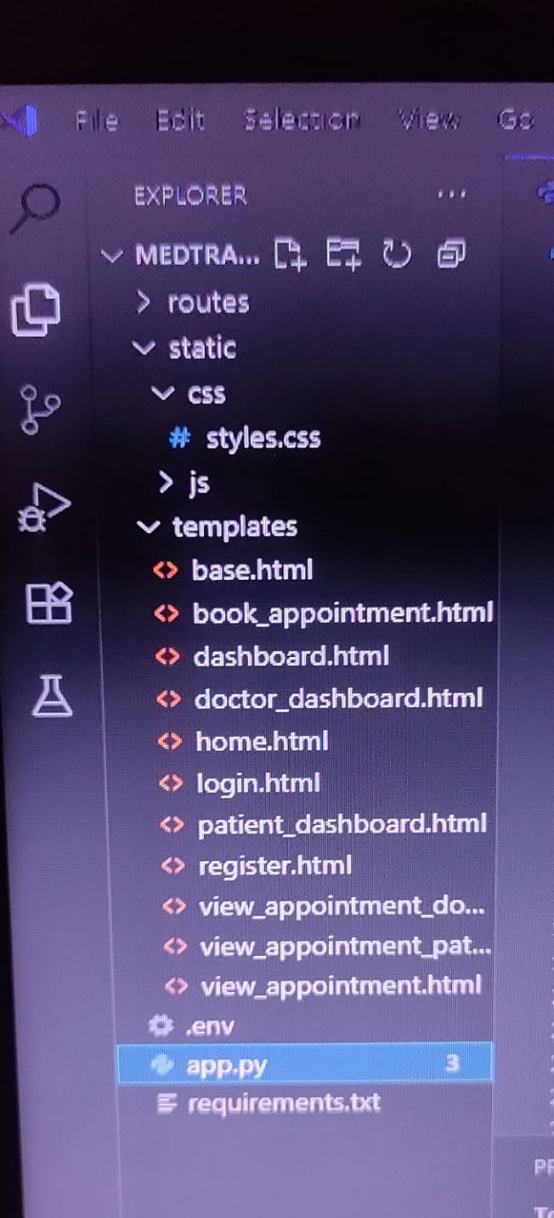
**Milestone 7. Deployment using EC2**

* Upload Flask Files
* Run the Flask App

**Milestone 8. Testing and Deployment**

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

**1.Develop the backend using Flask**



**Description of the code:**

**1. Imports:**

* Import necessary modules like Flask, render\_template, request, redirect, url\_for, session, and others as needed for your app.

**2. Flask App Initialization:**

* app = Flask(\_\_name\_\_): Starts the Flask web application.
* app.secret\_key: Used for securely signing the session cookie and enabling flash messages.

**3. Temporary In-Memory Storage:**

* users: A list or data structure to store registered user information temporarily.
* bookings: A list to store appointment or ticket bookings.
* booking\_counter: A counter to uniquely identify each booking.

**4. Authentication Routes:**

**4.1 Homepage /:**

* Displays the landing page or redirects users based on login status.

**4.2 Login /login:**

* **GET request**: Shows the login form.
* **POST request**:
  + Validates user credentials.
  + Checks if email exists and password matches.
  + On success, stores user info in session.
  + On failure, flashes an "Invalid login" message.

**4.3 Signup /signup:**

* **GET request**: Shows the signup form.
* **POST request**:
  + Collects new user details from the form.
  + Checks if the email is already registered.
  + Hashes the password and stores the user data.
  + Redirects to the login page on successful signup.

**4.4 Logout /logout:**

* Clears the user session to log out.
* Flashes a logout confirmation message.
* Redirects to the homepage.

**5. Main Application Pages:**

**5.1 Home after Login /home1, About /about, Contact /contact\_us:**

* Checks if the user is logged in.
* If logged in, renders the home1.html dashboard.
* If not logged in, redirects to /login.
* Serves static pages like About and Contact Us.

**6. Doctor and Patient Routes:**

* **Doctor Routes (doctor.py)**:  
  Handles doctor-specific views such as doctor dashboard, viewing appointments, and managing schedules.
* **Patient Routes (patient.py)**:  
  Manages patient-specific functionality like booking appointments and viewing their own appointments.

**7. Appointment Booking:**

* Page to book new appointments.
* Stores booking info in bookings.
* Associates bookings with users (patients) and doctors.
* Allows viewing and managing appointments based on user roles.

**8. Static Files:**

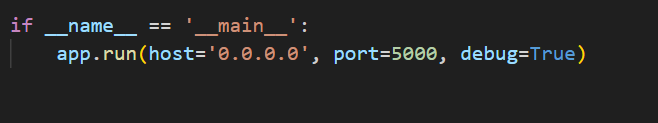
* CSS and JavaScript files stored under static/css and static/js for styling and interactive behavior.
* HTML templates under templates/ folder for rendering views.

**9. Environment Variables (.env):**

* Stores sensitive information like SECRET\_KEY and database credentials.
* Loaded into the app to configure secret keys and other settings.

**10. Running the Flask App:**

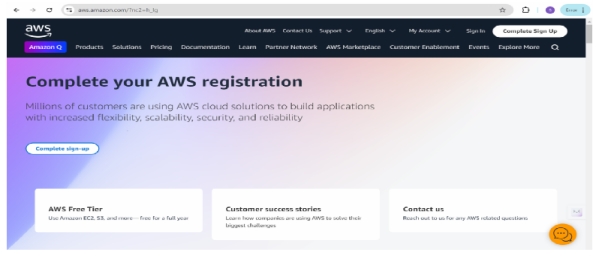
* The app is run using:



**2.** **AWS Account Setup and Login**

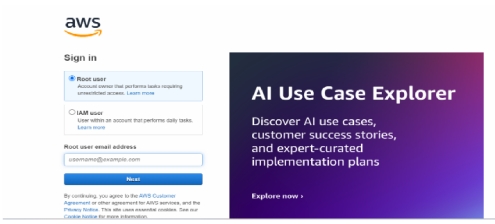
**● Activity 1.1: Set up an AWS account if not already done.**

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



**● Activity 1.2: Log in to the AWS Management Console**

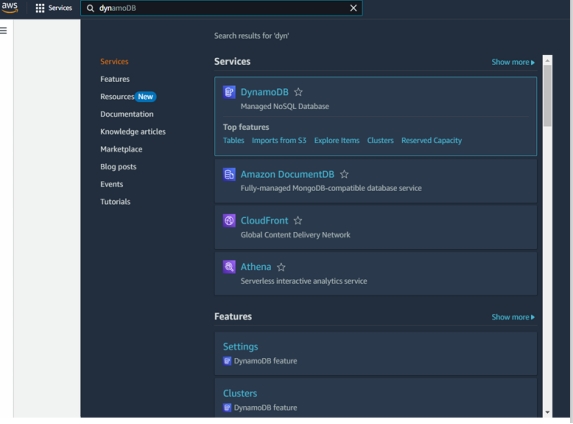
**○ After setting up your account, log in to the AWS Management Console.**

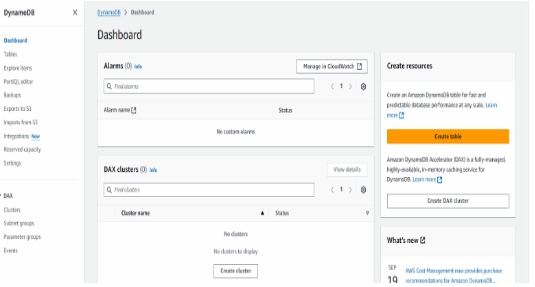


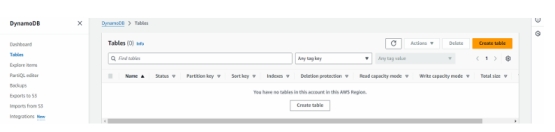
**2: DynamoDB Database Creation and Setup**

**● Activity 2.1:Navigate to the DynamoDB**

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

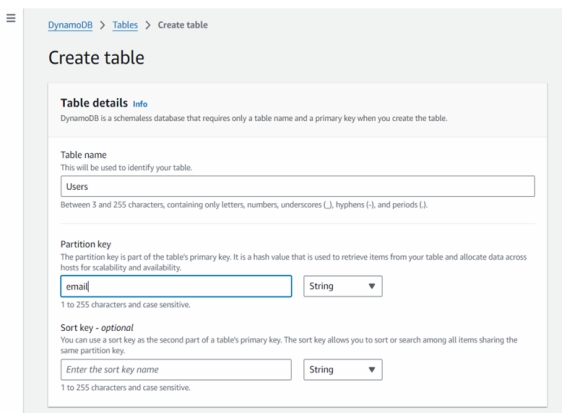


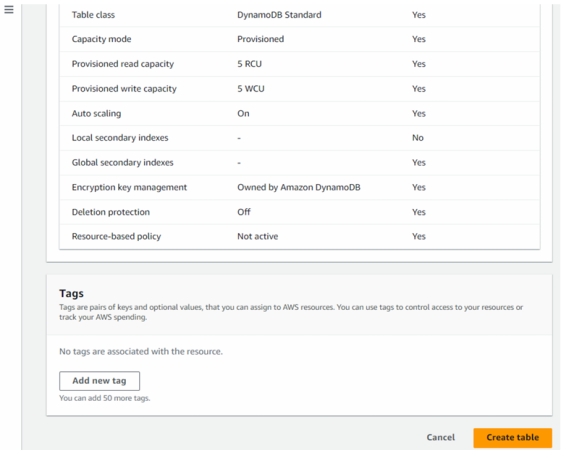


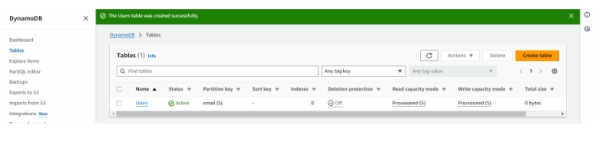


**● Activity 2.2:Create a DynamoDB table for storing registration details and book requests.**

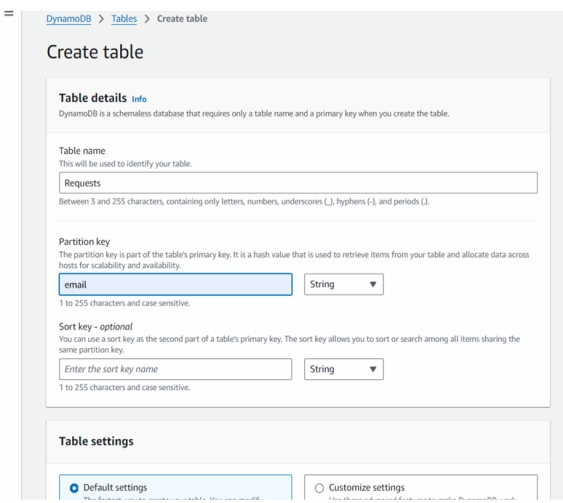
**○ Create Users table with partition key “Email” with type String and click on create tables.**

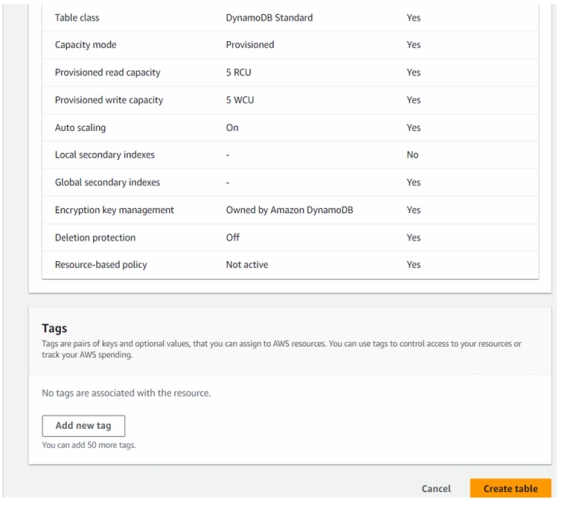


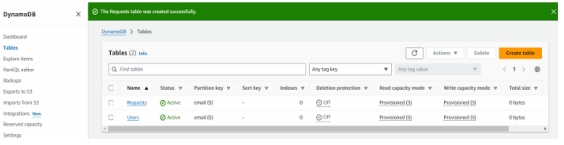




**○ Follow the same steps to create a requests table with Email as the primary key for book requests data.**



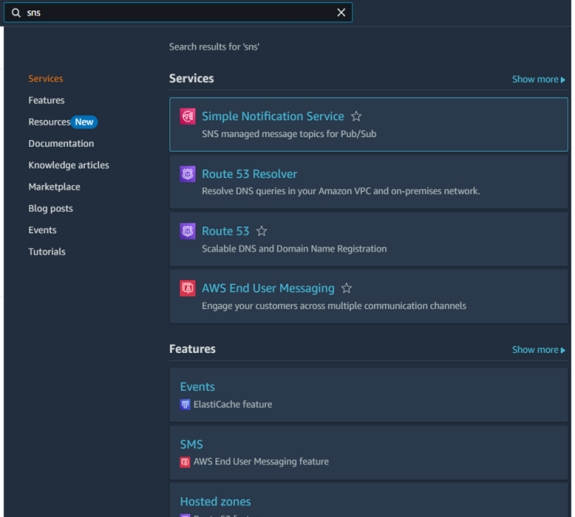


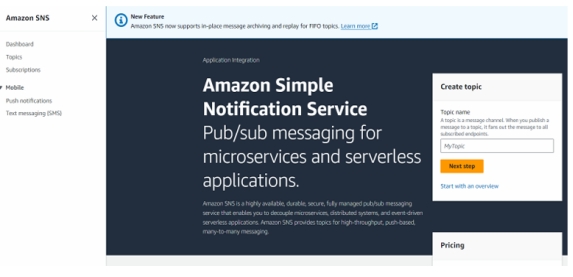


**3: SNS Notification Setup**

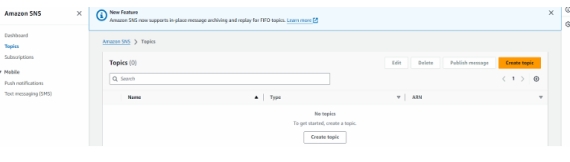
**● Activity 3.1: Create SNS topics for sending email notifications to users and library staff.**

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

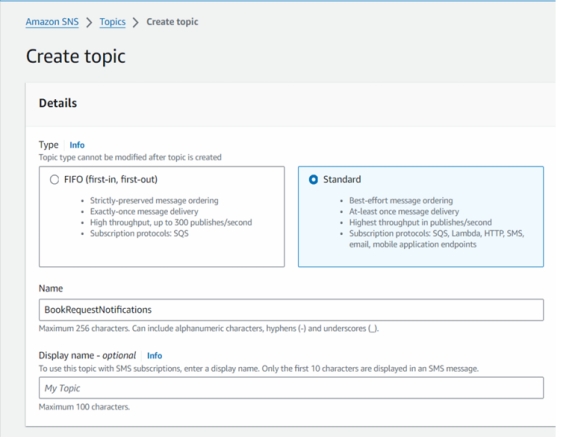


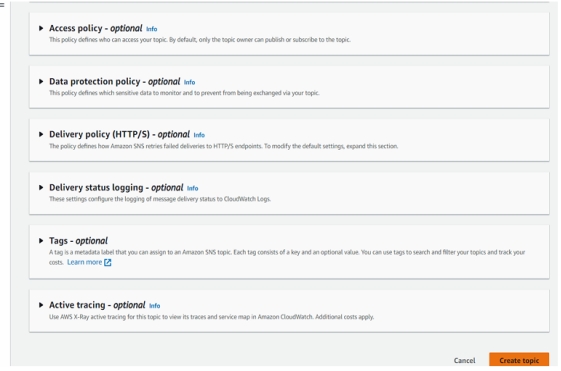


○ Click on **Create Topic** and choose a name for the topic**.**

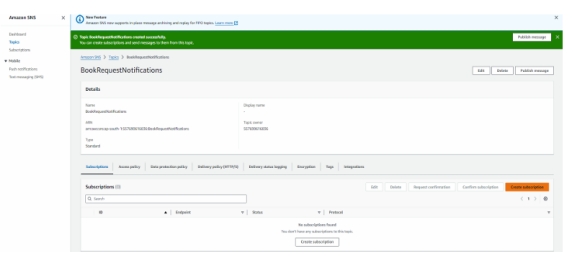


**○ Choose Standard type for general notification use cases and Click on Create Topic.**



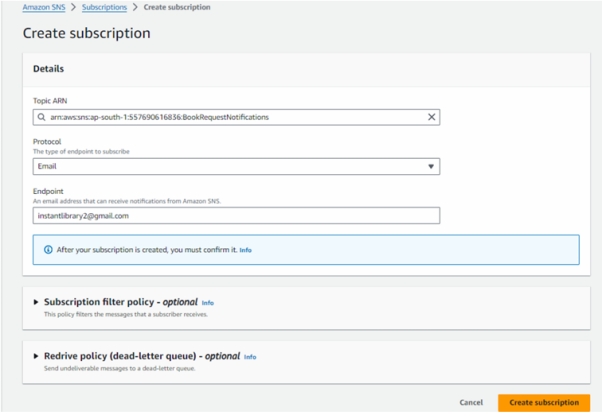


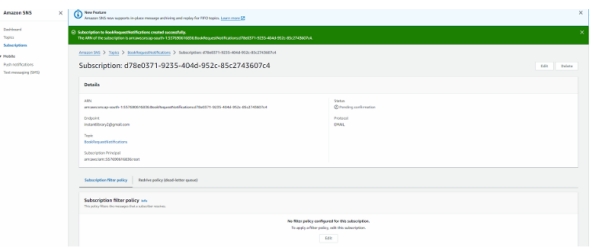
**○ Configure the SNS topic and note down the Topic ARN.**



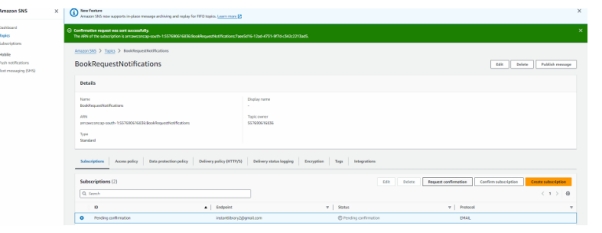
**● Activity 3.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.**

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

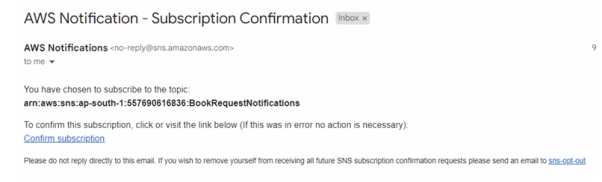


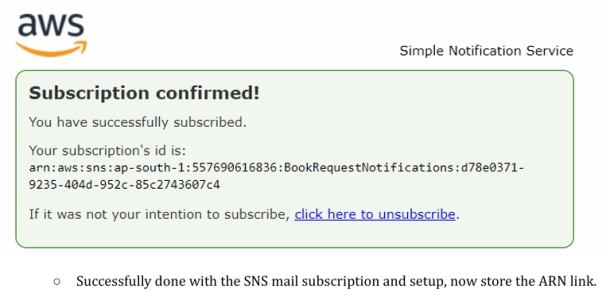


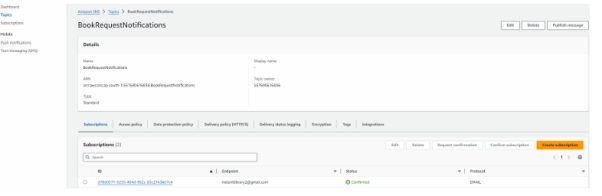
**○ After subscription request for the mail confirmation**



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



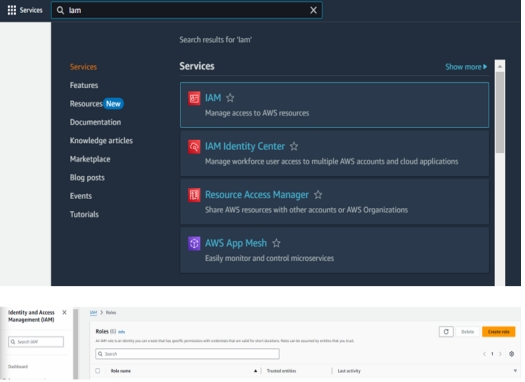


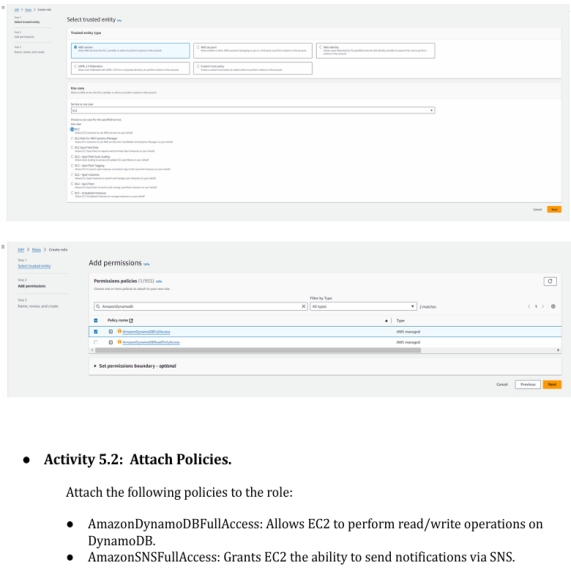


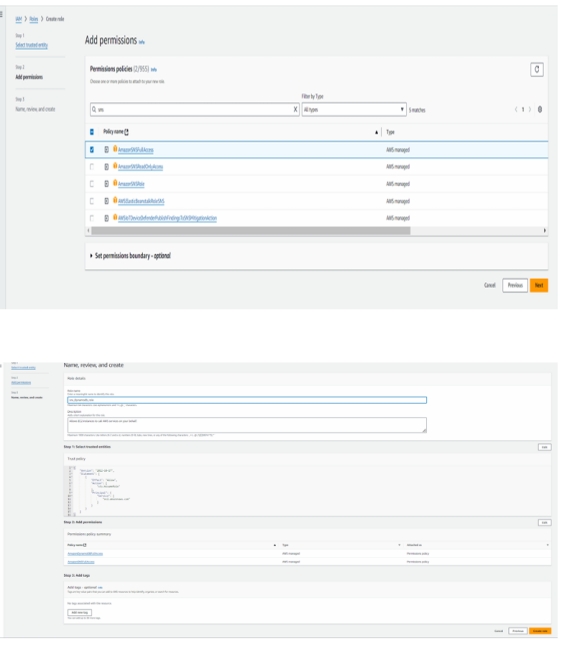
**5: IAM Role Setup**

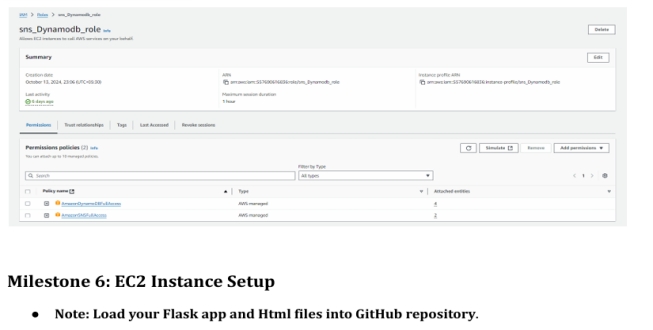
**● Activity 5.1:Create IAM Role.**

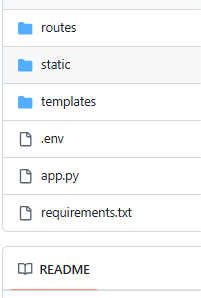
**○ In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.**

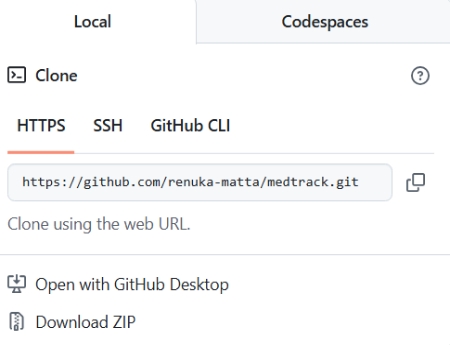


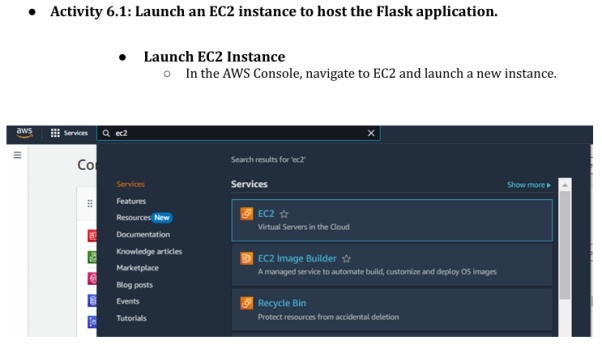


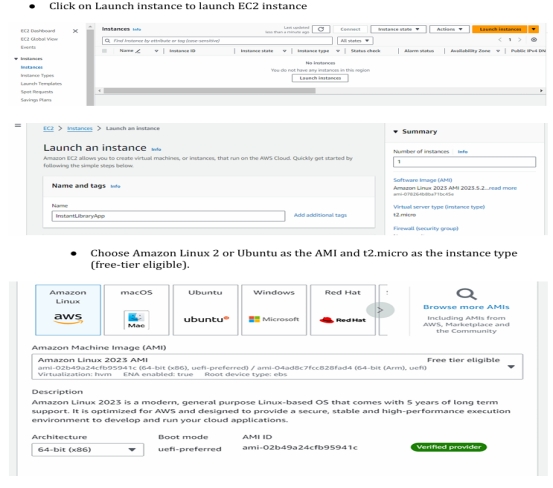


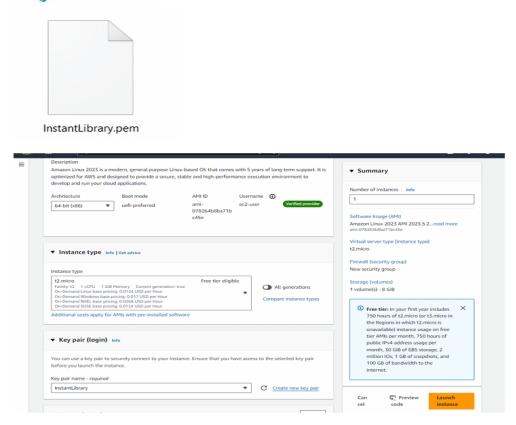


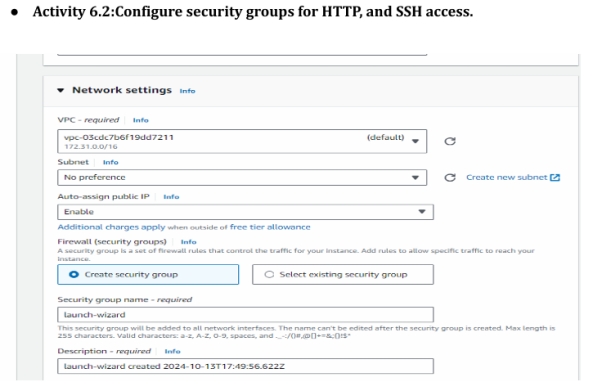


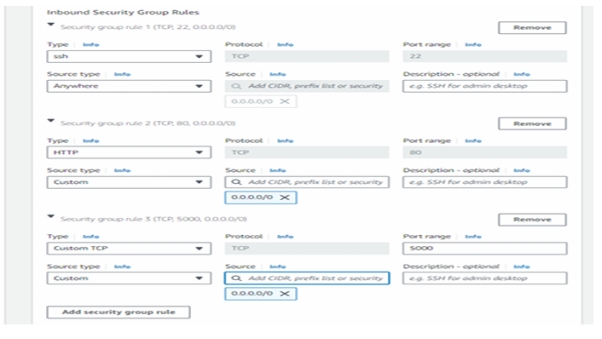


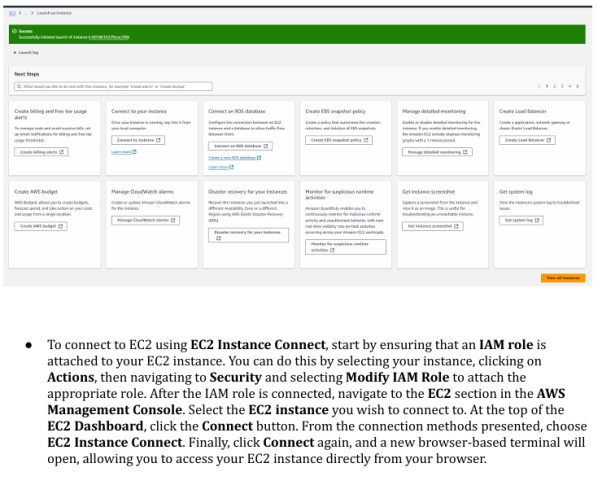


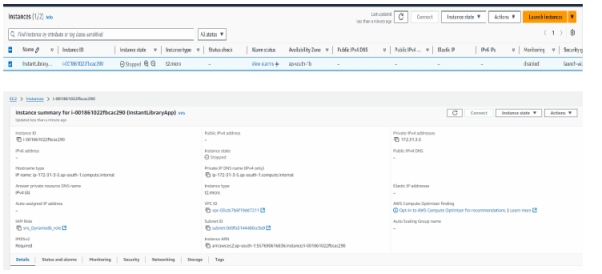


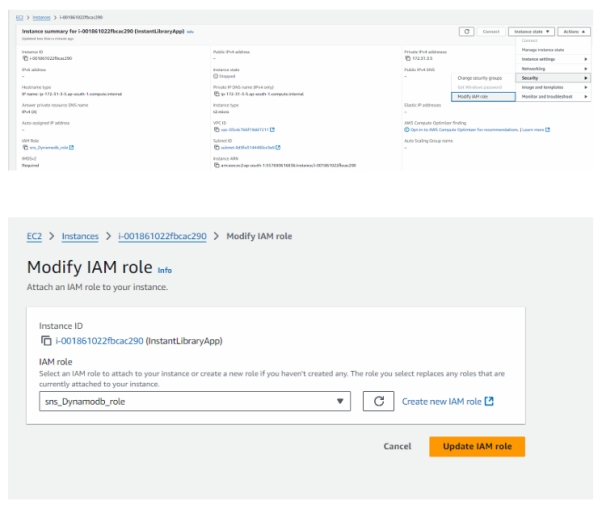




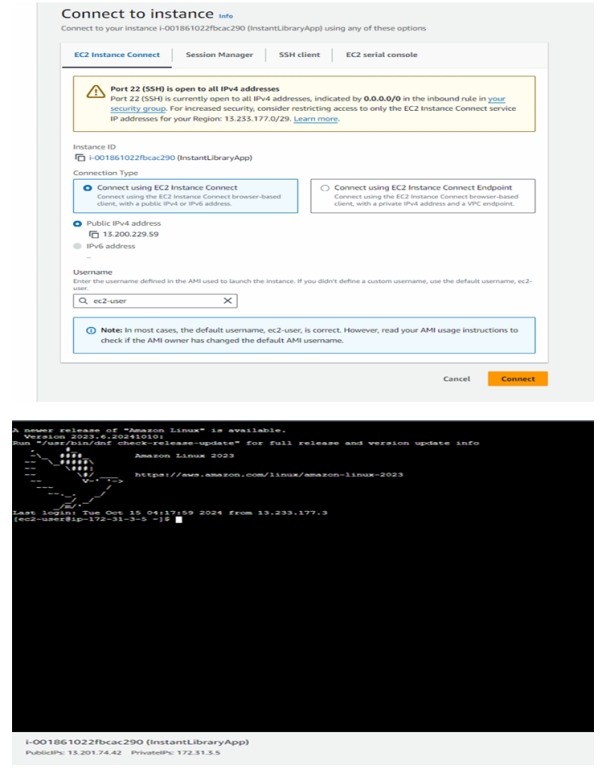


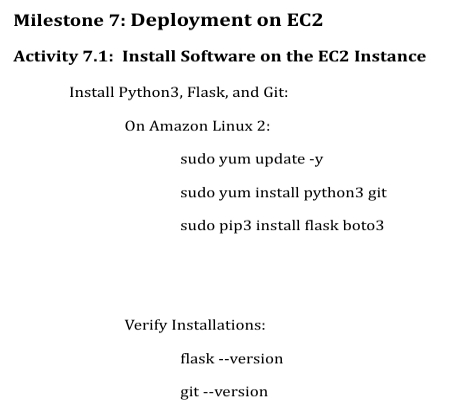


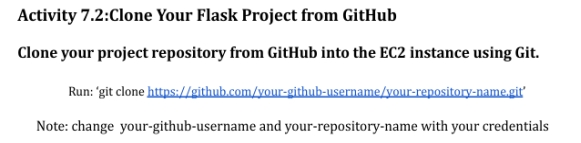




* **Now connect the EC2 with the files**



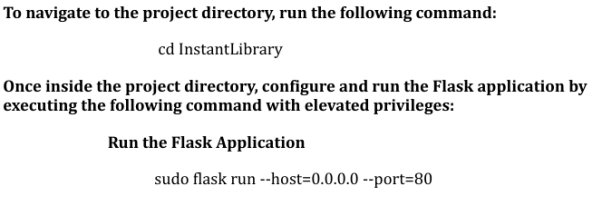


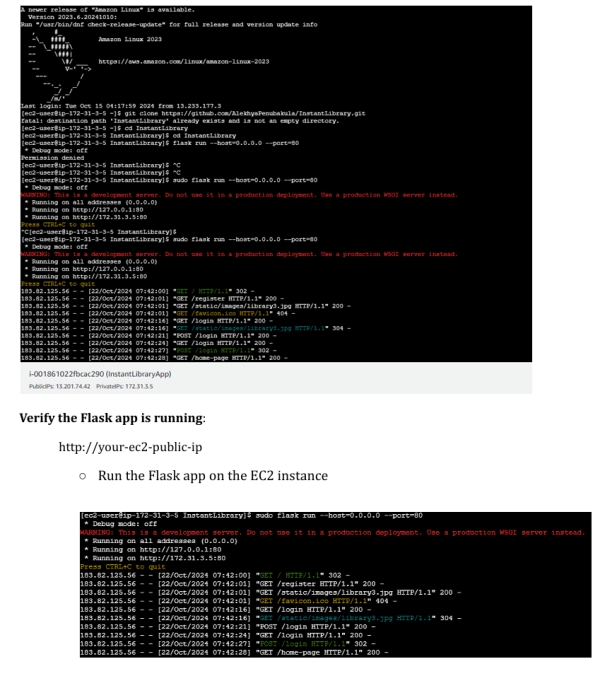


**Here: ‘git clone**

**<https://github.com/Navyasri14789/medtrack-project>**

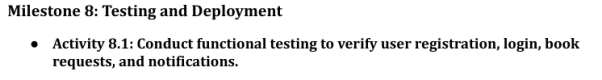
**This will download your project to the EC2 instance.**



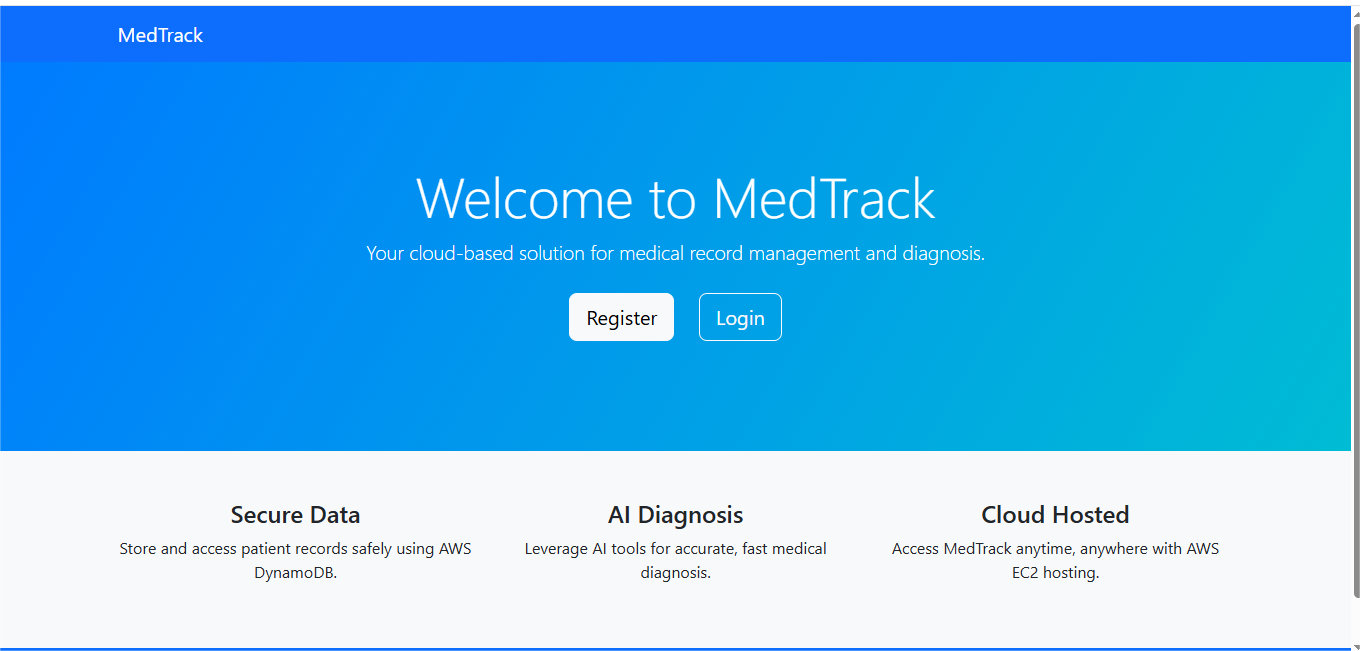


Access the Website Through

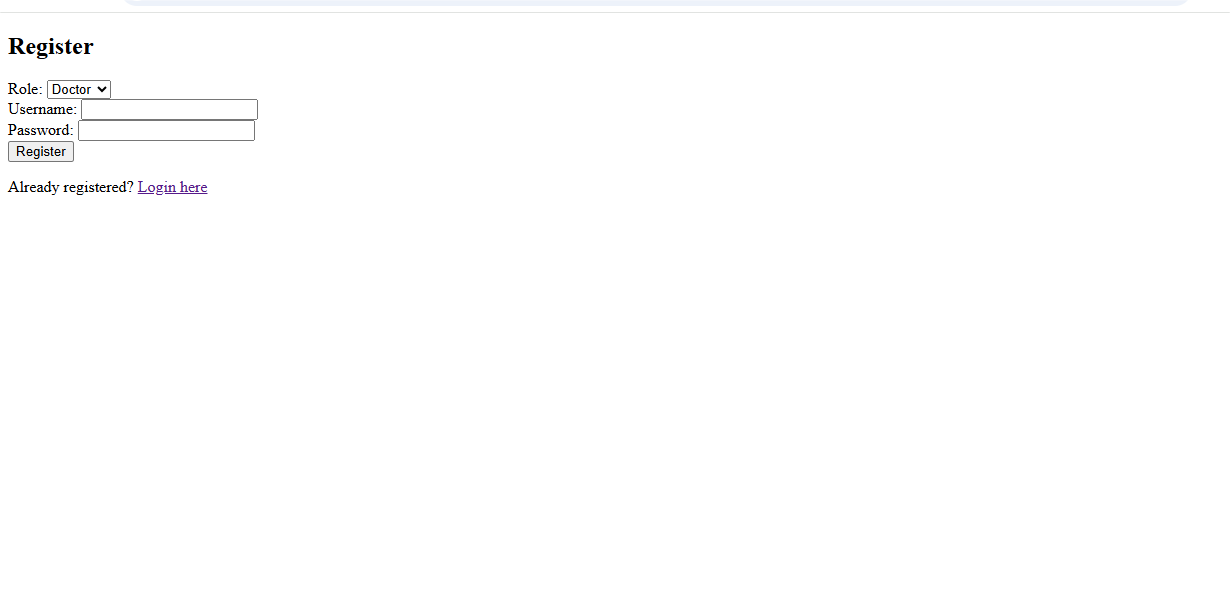
Running on http://127.0.0.1:5000



**Home Page:**



**Register page:**



**Login Page:**

