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

## Project 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.

### 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.

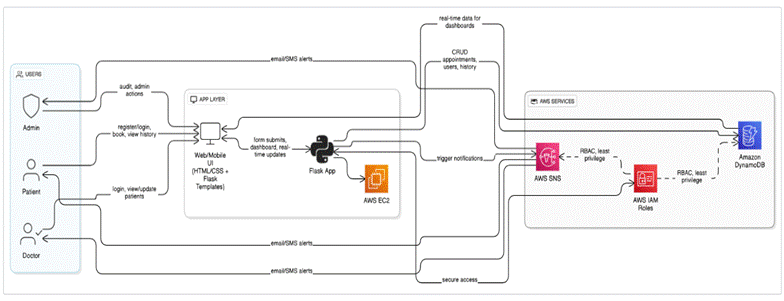
### Scenario 2: Secure User Management with IAM

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

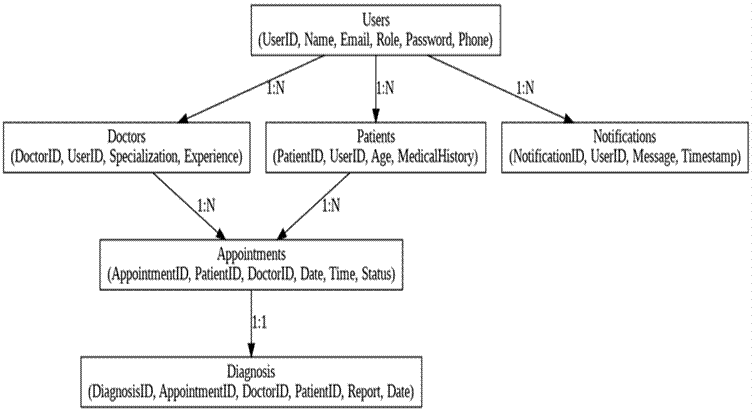
### Scenario 3: Easy Access to Medical History and Resources

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.

AWS ARCHITECTURE



Entity Relationship (ER)Diagram:



## Pre-requisites:

1. .**AWS Account Setup**: [AWSAccount Setup](https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh)
2. **Understanding IAM**: [IAMOverview](https://youtu.be/gsgdAyGhV0o?si=3qg-bULgkD4LXNvR)
3. **Amazon EC2 Basics**: [EC2Tutorial](https://youtu.be/8TlukLu11Yo?si=MUj0nEAOESRhHUIz)
4. **DynamoDB Basics**: [DynamoDB Introduction](https://docs.aws.amazon.com/dynamodb)
5. **SNS Overview**: [SNSDocumentation](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 alreadydone**. 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 requestnotifications.

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

### Backend Development and Application Setup

**Activity 4.1**:Develop the BackendUsing 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 instanceto host the Flask application.

**Activity 6.2**: Configure securitygroups 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 AccountSetup and Login

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

* + 1. Sign up for an AWS account and configure billingsettings.

|  |  |
| --- | --- |
|  |  |
|  |  |

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

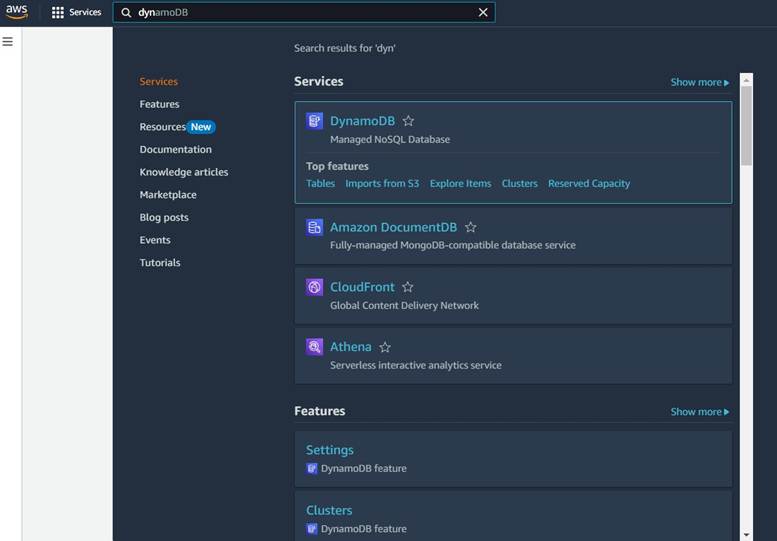
* + 1. After setting up your account,log in to the [AWS Management Console](https://aws.amazon.com/console/).

|  |  |
| --- | --- |
|  |  |
|  |  |

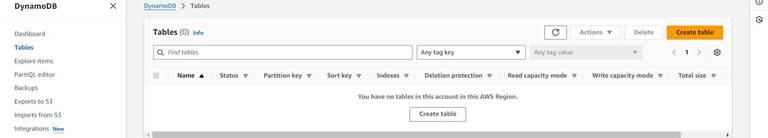
# Milestone 2: DynamoDBDatabase Creation and Setup

### Activity 2.1:Navigate to the DynamoDB

* + 1. In the AWS Console, navigateto DynamoDB and click on create tables.

* + 1. 

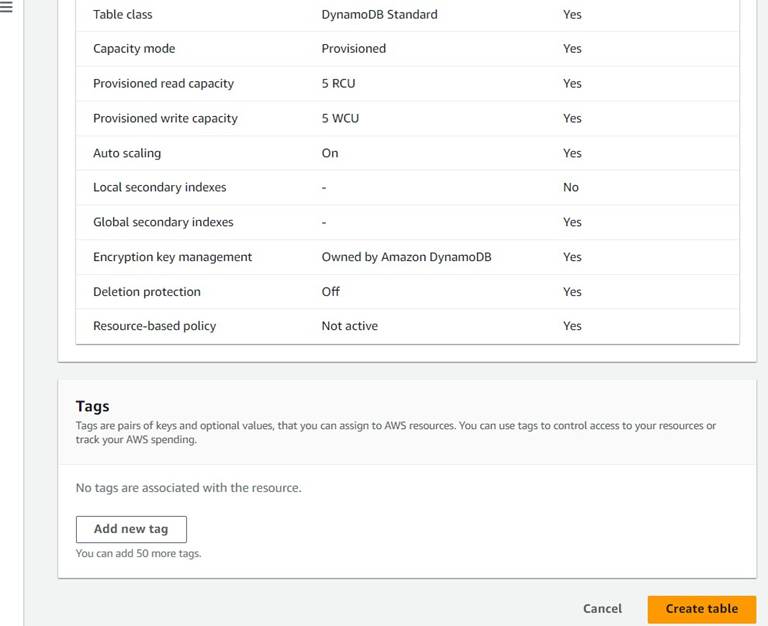
|  |  |
| --- | --- |
|  |  |
|  |  |



### Activity 2.2:Create a DynamoDBtable for storingregistration details and book requests.

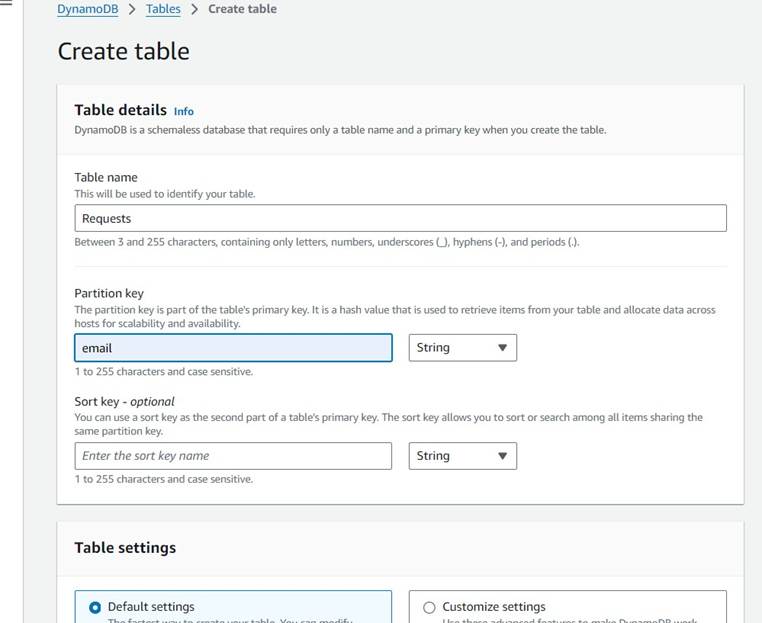
* + 1. Create Users table with partition key “Email” with type String and click on create tables.

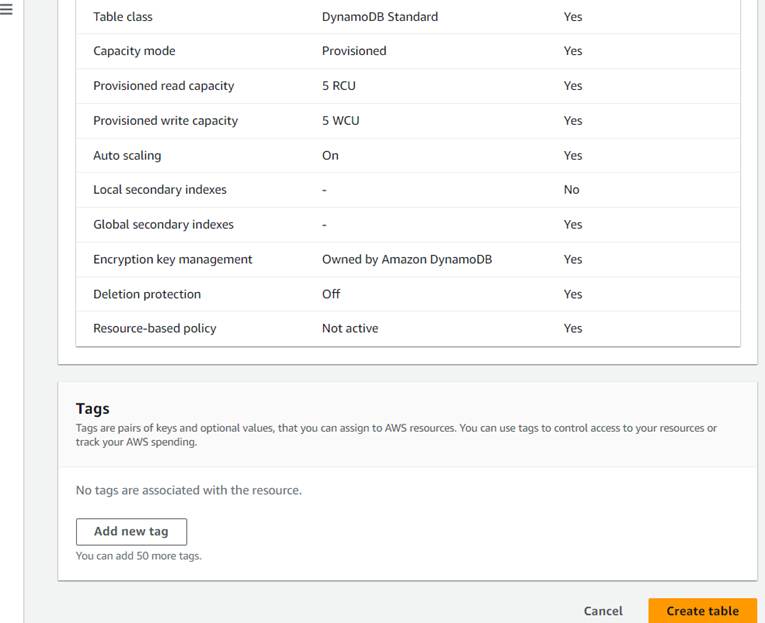
|  |  |
| --- | --- |
|  |  |
|  |  |



|  |  |
| --- | --- |
|  |  |
|  |  |

* + 1. Follow the same steps to create a requests tablewith Email as the primary key for book requests data.





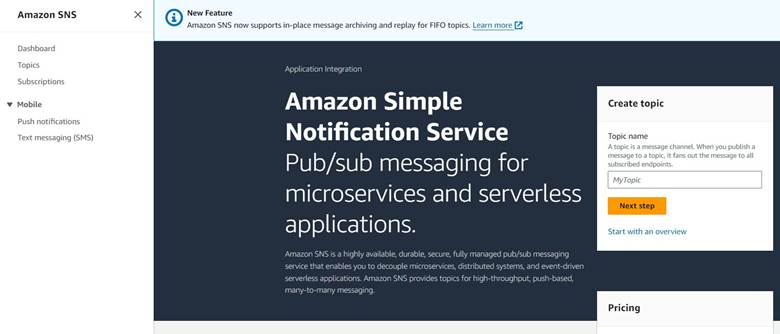
|  |  |
| --- | --- |
|  |  |
|  |  |

## Milestone 3: SNS Notification Setup

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

* 1. In the AWS Console,search for SNS and navigateto the SNS Dashboard.

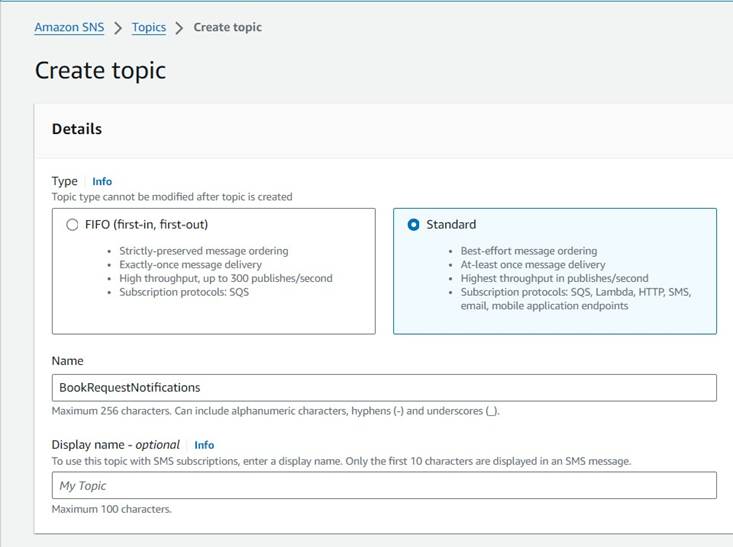
|  |  |
| --- | --- |
|  |  |
|  |  |

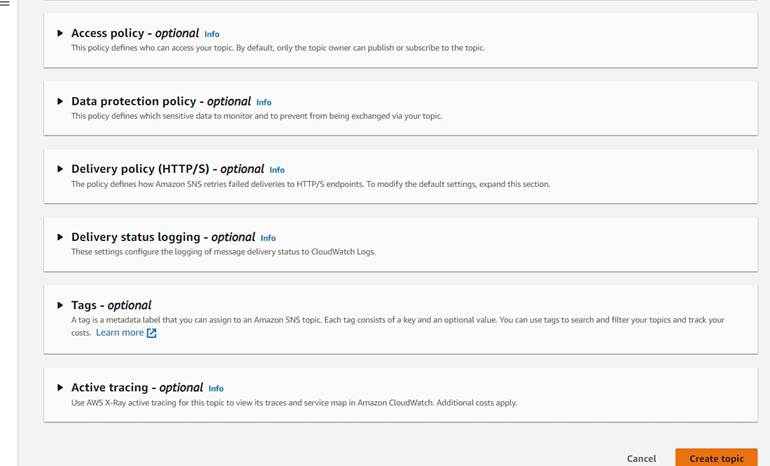


* 1. Click on **Create Topic** and choosea name for the topic.

|  |  |
| --- | --- |
|  |  |
|  |  |

* 1. Choose Standardtype for generalnotification use casesand Click on Create Topic.





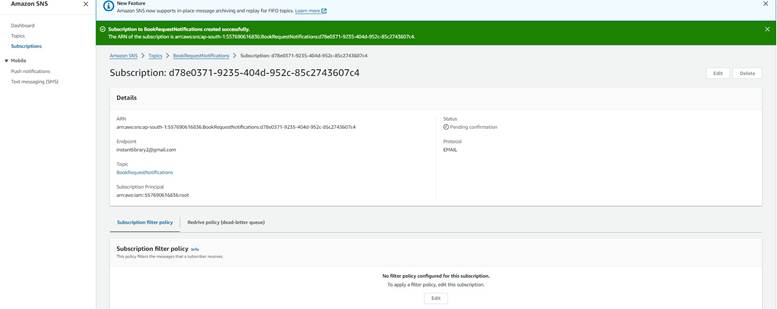
* 1. Configure the SNS topic and note down the **Topic ARN**.

|  |  |
| --- | --- |
|  |  |
|  |  |

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

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

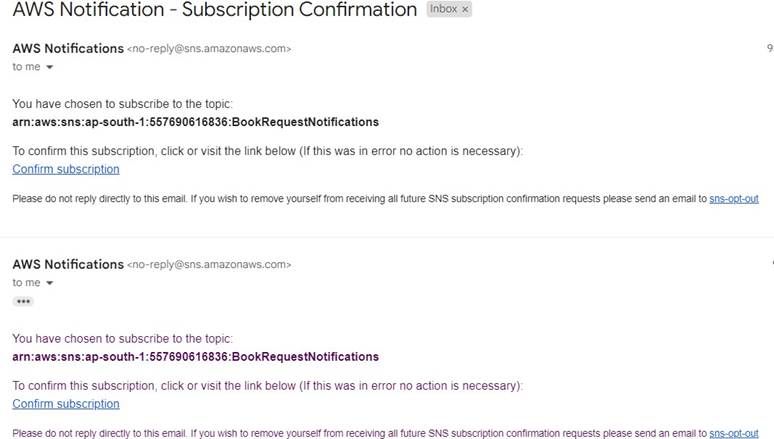
|  |  |
| --- | --- |
|  |  |
|  |  |



* 1. After subscription requestfor the mail confirmation

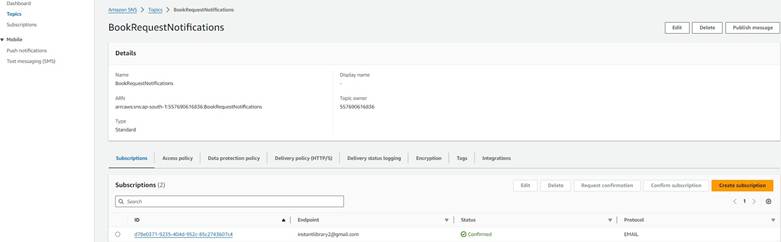
|  |  |
| --- | --- |
|  |  |
|  |  |

* 1. Navigate to the subscribed Emailaccount and Clickon the confirm subscription in the AWS Notification- Subscription Confirmation mail.



|  |  |
| --- | --- |
|  |  |
|  |  |

* 1. 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

* 1. 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 librariesincluding Flask utilitiesfor routing, Boto3 for DynamoDB operations, SMTP and email modules for sending mails, and Bcrypt for password hashing and verification

|  |  |
| --- | --- |
|  |  |
|  |  |

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

### Dynamodb Setup:

|  |  |
| --- | --- |
|  |  |
|  |  |

**Description:** initialize the DynamoDBresource for the ap-south-1 regionand 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):

|  |  |
| --- | --- |
|  |  |
|  |  |

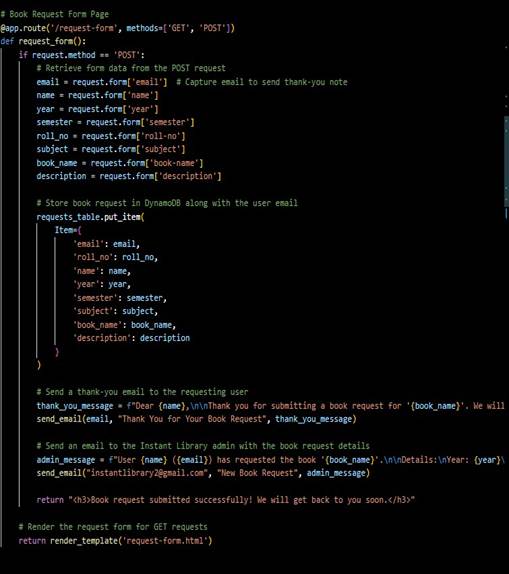
**Description:** define /login route to validate user credentials against DynamoDB, check the password usingBcrypt, update the login counton successful authentication, and redirect users to the home page

### Home, E- book buttons and subject routes:

|  |  |
| --- | --- |
|  |  |
|  |  |

**Description:** define /home-page to render the main homepage,/ebook-buttons to handle subject selection and redirection, and /<subject>.html dynamic route to render

subject-specific pages like Mathematics or English.

1. **Request Routes:**
2. 

**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 confirmsubmission with a success message.

### Exit Route:

|  |  |
| --- | --- |
|  |  |
|  |  |

**Description:** define /exit routeto render the exit.html page when the user choosesto leave or close the application.

**Deployment Code:**

|  |  |
| --- | --- |
|  |  |
|  |  |

**Description:** start the Flask serverto listen on all networkinterfaces (0.0.0.0) at port 80

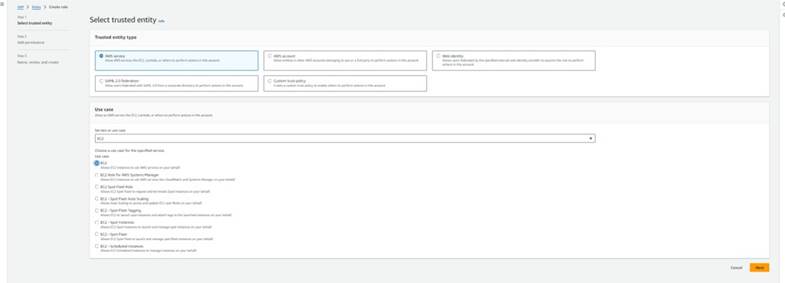
with debug mode enabled for development and testing.

# Milestone 5: IAM Role Setup

### Activity 5.1:Create IAM Role.

* 1. In the AWS Console,go to IAM and createa new IAM Role for EC2 to interact with DynamoDB and SNS.

|  |  |
| --- | --- |
|  |  |
|  |  |

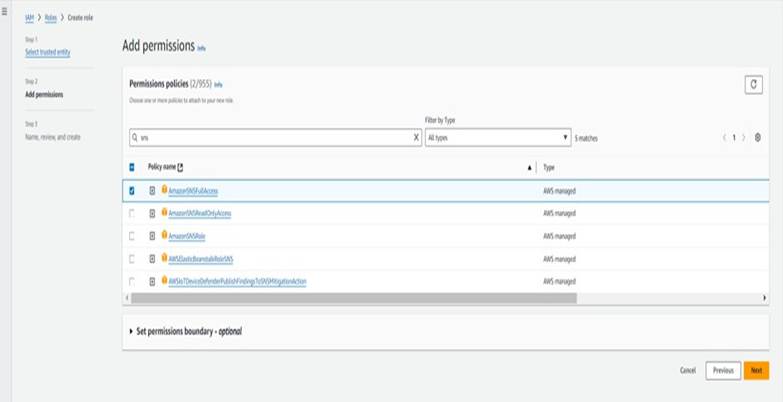


|  |  |
| --- | --- |
|  |  |
|  |  |

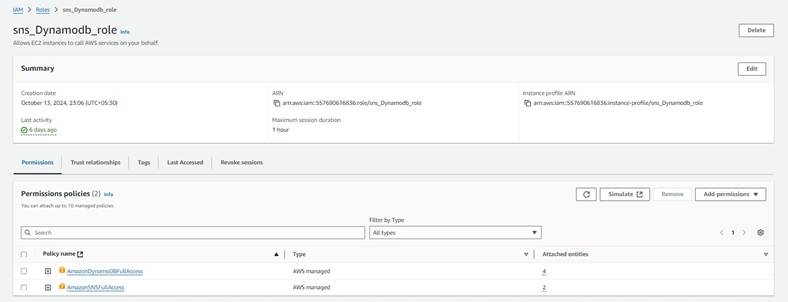
### Activity 5.2: Attach Policies.

Attach the following policiesto the role:

1. AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.
2. AmazonSNSFullAccess: Grants EC2 the ability to send notifications via SNS.



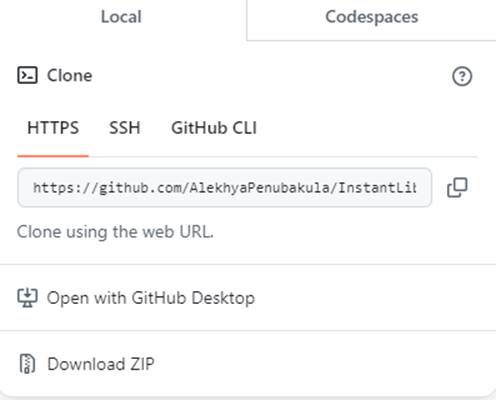
|  |  |
| --- | --- |
|  |  |
|  |  |



# Milestone 6: EC2 InstanceSetup

1. **Note: Load your Flask app and Html files into GitHubrepository.**

|  |  |
| --- | --- |
|  |  |
|  |  |



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

* 1. **Launch EC2 Instance**
     1. In the AWS Console,navigate to EC2 and launcha new instance.

|  |  |
| --- | --- |
|  |  |
|  |  |

1. Click on Launchinstance to launchEC2 instance

|  |  |
| --- | --- |
|  |  |
|  |  |

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

|  |  |
| --- | --- |
|  |  |
|  |  |

1. Create and downloadthe key pair for Serveraccess.

|  |  |
| --- | --- |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |

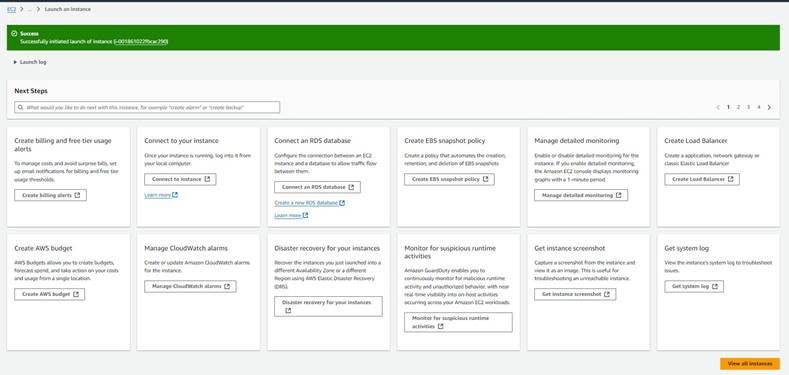


|  |  |
| --- | --- |
|  |  |
|  |  |

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

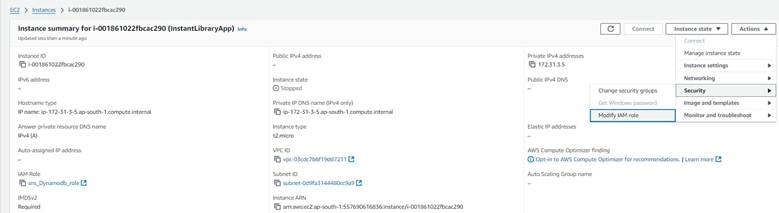
|  |  |
| --- | --- |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |



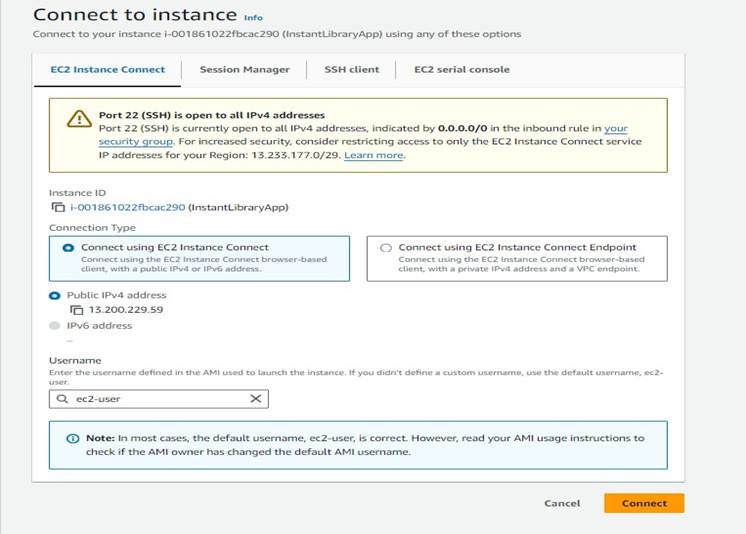
1. To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the 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 terminalwill open, allowing you to access your EC2 instance directly from your browser.

|  |  |
| --- | --- |
|  |  |
|  |  |



|  |  |
| --- | --- |
|  |  |
|  |  |

1. Now connect the EC2 with the files



|  |  |
| --- | --- |
|  |  |
|  |  |

**Milestone 7: Deployment on EC2**

### Activity 7.1: Install Softwareon the EC2 Instance

Install Python3, Flask, and Git: On Amazon Linux 2:

sudo yum update-y

sudo yum installpython3 git sudo pip3 install flaskboto3

Verify Installations:

flask --version git --version

### Activity 7.2:Clone Your Flask Projectfrom GitHub

**Clone your projectrepository from GitHubinto the EC2 instance using Git.**

Run: ‘git clone <https://github.com/your-github-username/your-repository-name.git>’

Note: change your-github-username and your-repository-name with your credentials here:  **‘git clone <https://github.com/renuka-matta/medtrack.git>’**

This will downloadyour project to the EC2 instance.

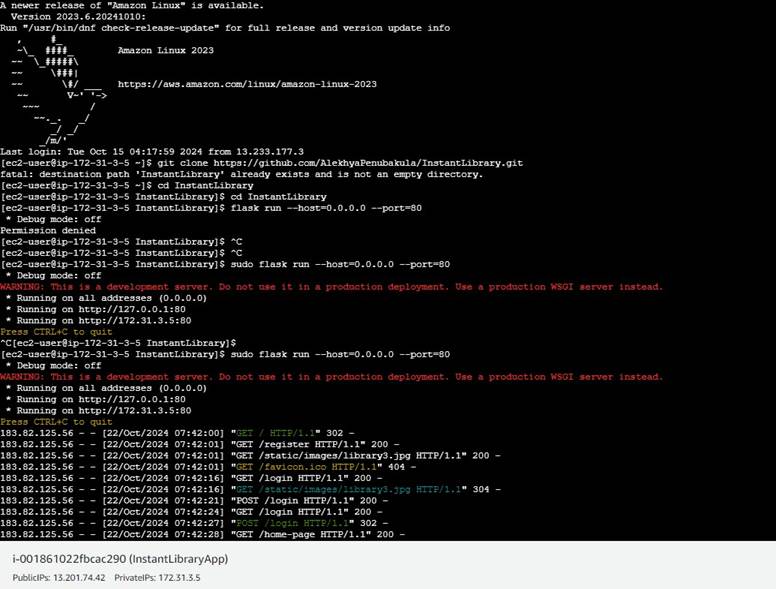
### To navigateto the project directory, run the following command:

cd InstantLibrary

### Once inside the projectdirectory, 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/)

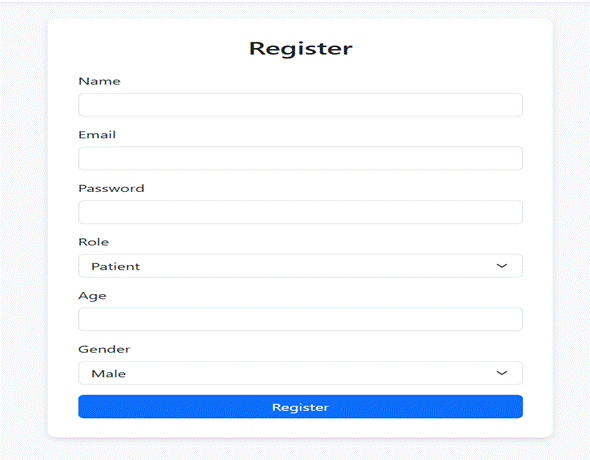
1. Run the Flask app on the EC2 instance

|  |  |
| --- | --- |
|  |  |
|  |  |

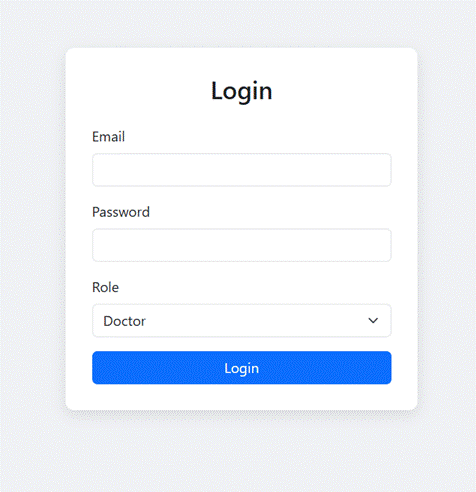
## Milestone 8: Testing and Deployment

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

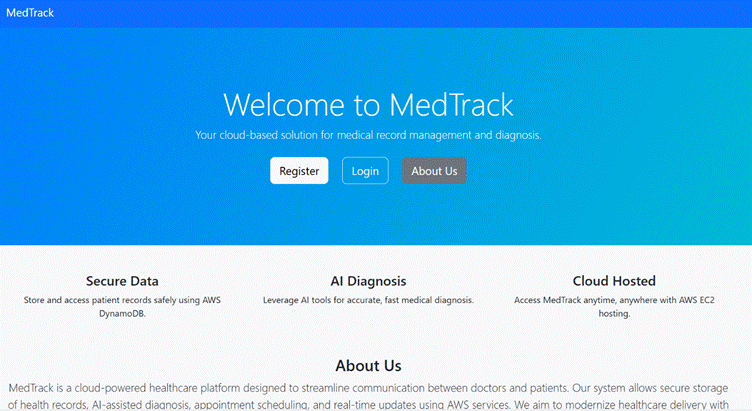
**Register Page:**



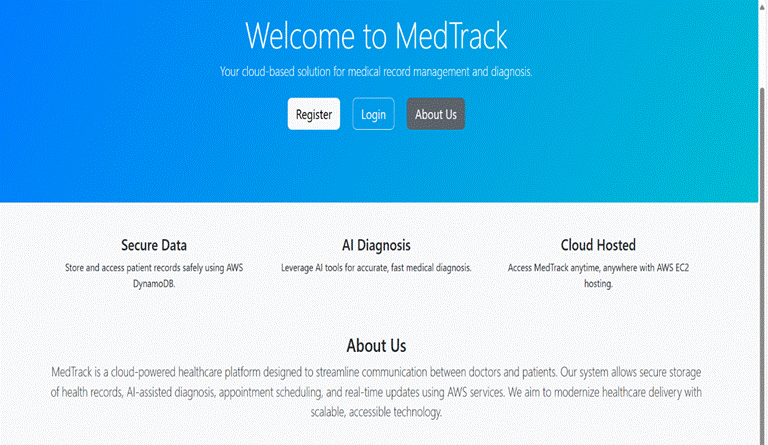
**Login Page:**



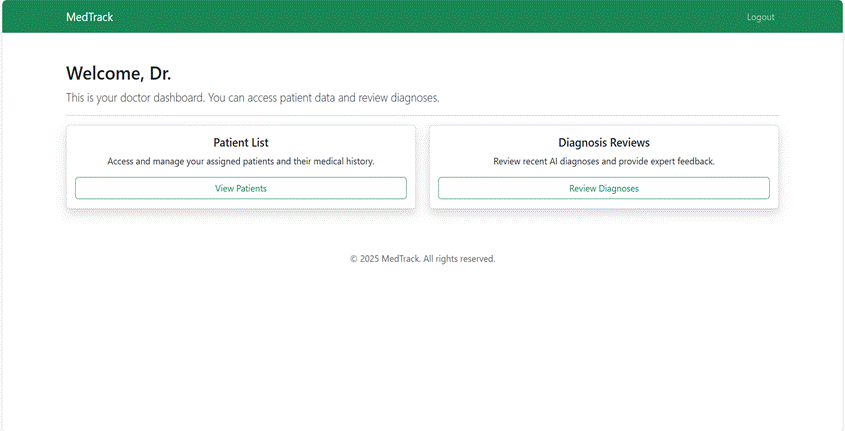
**Home page:**



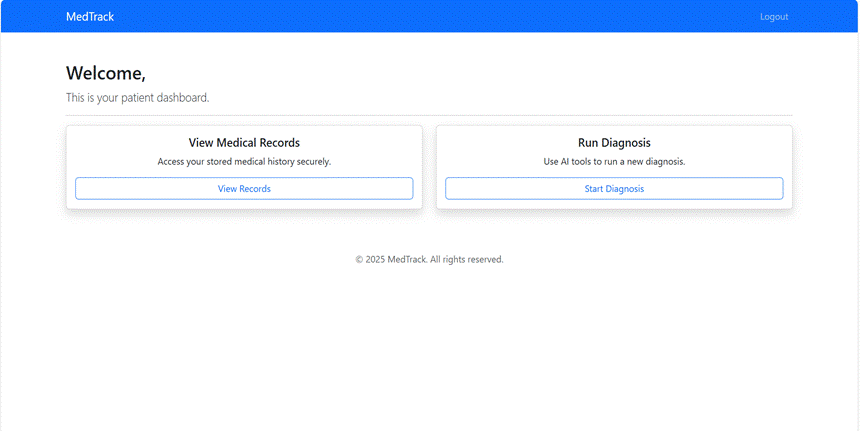
**About Us page:**



**Doctor DashBoard:**



**Patient DashBoard:**



**Exit:**

**Session Ended**

Please close this tab.

**Conclusion:**

**MedTrack** is a web-based healthcare management system built using Flask that facilitates interaction between doctors and patients. It allows users to register and log in as either a doctor or a patient. Patients can view available doctors, book appointments, and view their scheduled or past visits. Doctors can access their assigned appointments, update them with a diagnosis, treatment plan, and prescription, and mark them as completed. The system uses in-memory storage for users and appointments, making it suitable for demonstration or development purposes without requiring a database. Although email and AWS SNS notification features are present in the code, they are disabled by default. MedTrack is designed with a clear role-based workflow and is easily extendable for real-world deployment with additional features like persistent databases, email alerts, password encryption, and a modern UI.