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.

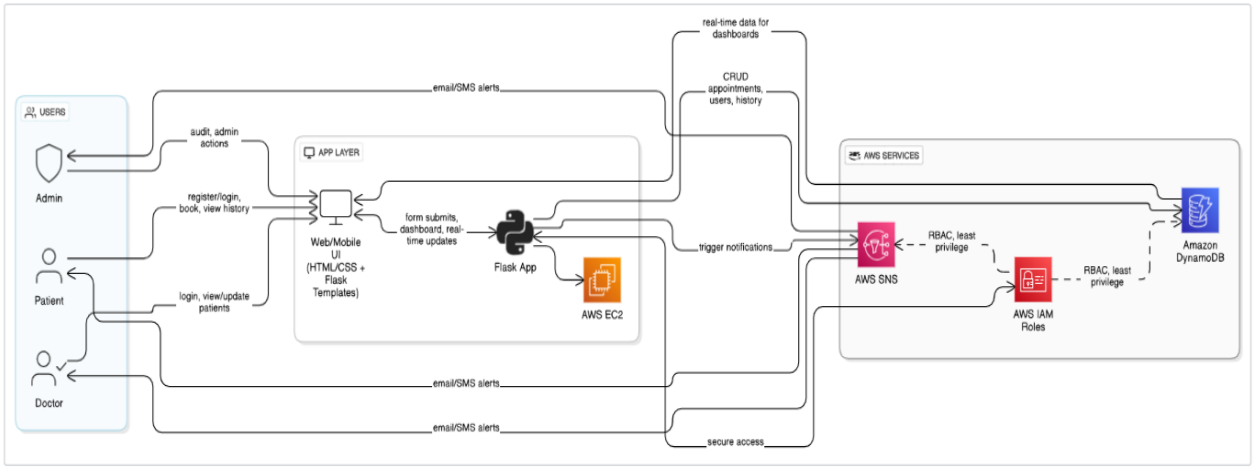
Scenario 2: Secure User Management with IAM

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.

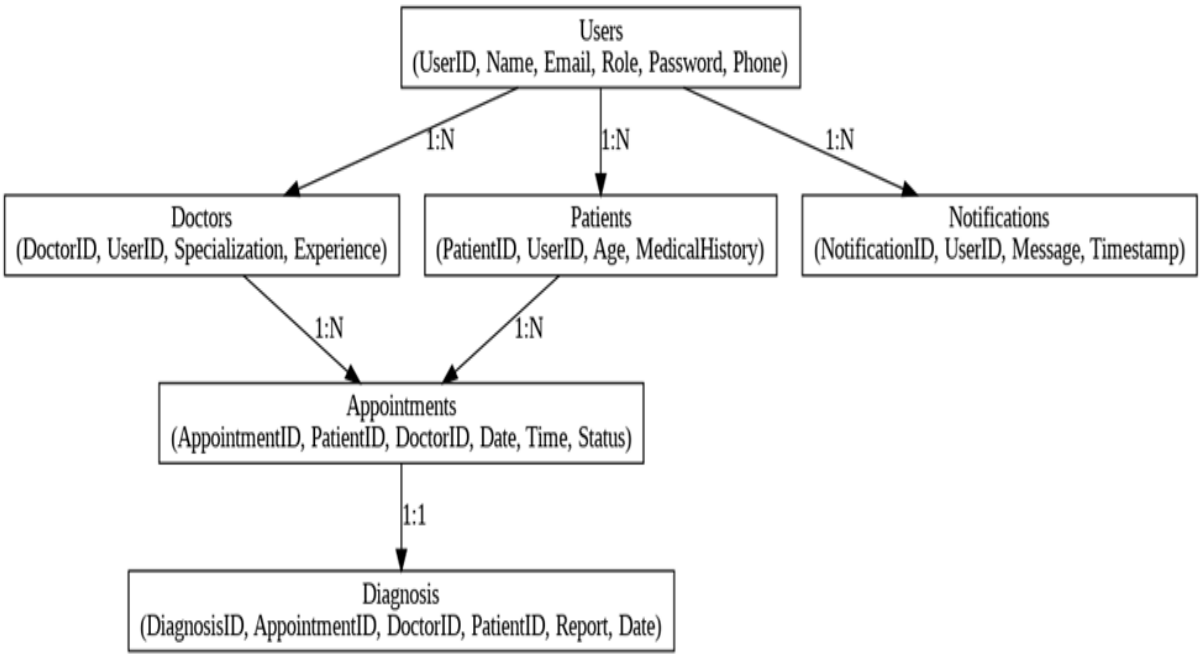
Scenario 3: Easy Access to Medical History and Resources

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:

- 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
- Amazon SNS: https://docs.aws.amazon.com/sns/latest/dg/welcome.html
- 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

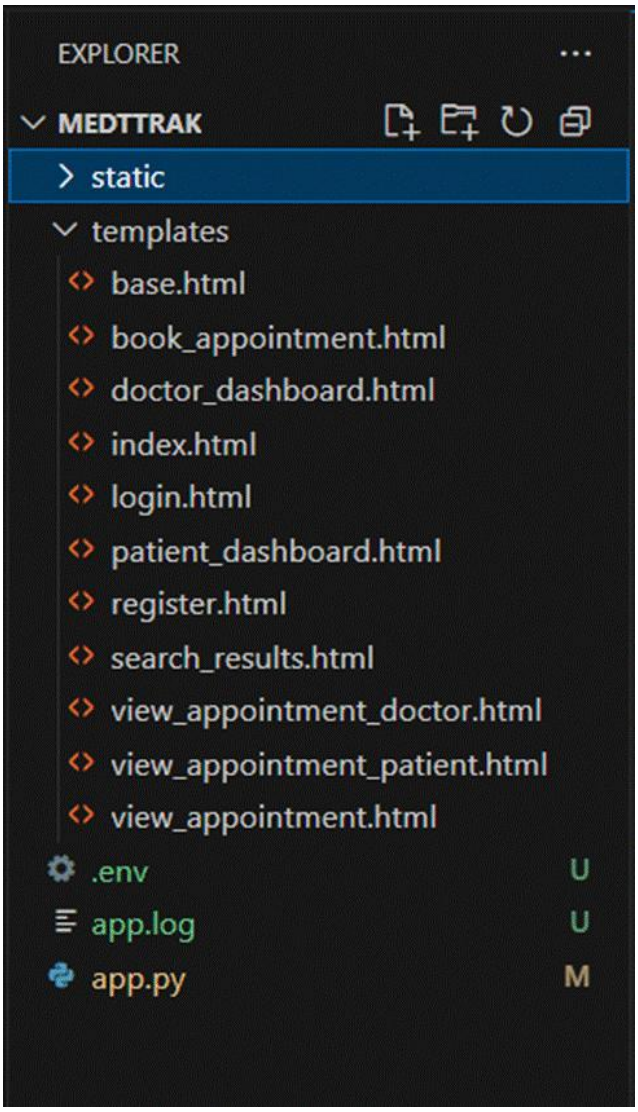
- Upload Flask Files
- Run the Flask App

Milestone 8. Testing and Deployment

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

1.Develop the backend using Flask

- File Explorer Structure



Description of the code :

1.Imports:

```
from flask import Flask, render_template, request, redirect, url_for, session, flash
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime
import json
```

2. Flask App Initialization:

```
app = Flask(__name__)
app.secret_key = 'your_secret_key_here'
```

- app = Flask(__name__): Starts the Flask application.
- app.secret_key: Required to safely use sessions and flashes.

3. Temporary In-Memory Storage:

```
# Temporary in-memory storage
users = []
bookings = []
booking_counter = 1
```

- users: List to store registered users.
  bookings: List to store all ticket bookings.
  booking_counter: A simple ID counter for bookings.

4. Authentication Routes:

4.1. Homepage /:

```
# Authentication Routes
@app.route('/')
def index():
    return render_template('index.html')
```

4.2. Login /login:

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        user = next((u for u in users if u['email'] == email), None)

        if user and check_password_hash(user['password'], password):
            session['user'] = {
                'id': user['id'],
                'name': user['name'],
                'email': user['email']
            }
            print("👤 Logged In User Session:", session['user'])
            return redirect(url_for('home1'))
        else:
            flash('Invalid email or password', 'danger')
    return render_template('login.html')
```

- If GET: Show the login page.
- If POST: Validate user login.
- Check if the email exists and password is correct.
- If success, save user info in session.
- If wrong, flash an "Invalid" message.

4.3. Signup /signup:

```python
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        password = generate_password_hash(request.form['password'])

        if any(u['email'] == email for u in users):
            flash('Email already registered!', 'danger')
            return redirect(url_for('signup'))

        new_user = {
            'id': len(users) + 1,
            'name': name,
            'email': email,
            'password': password
        }
        users.append(new_user)

        print("✅ Current Users List:", users)

        flash('Registration successful! Please login.', 'success')
        return redirect(url_for('login'))

    return render_template('signup.html')
```

- If GET: Show signup form.
- If POST:
- Collect user details.
- Check if email already exists.
- Save the user with hashed password into users.
- Redirect to login after successful registration.

4.4. Logout /logout:

```python
@app.route('/logout')
def logout():
    session.pop('user', None)
    flash('You have been logged out!', 'info')
    return redirect(url_for('index'))
```

- Remove the user from session (log out).
- Flash a message.
- Redirect to homepage.

5. Main Application Pages:

5.1. Home After Login /home1, About Page /about, Contact Page /contact_us

```python
# Application Routes
@app.route('/home1')
def home1():
    if 'user' not in session:
        return redirect(url_for('login'))

    print("👤 Current Session User Info:", session['user'])

    return render_template('home1.html')

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/contact_us')
def contact():
    return render_template('contact_us.html')
```

- Check if the user is logged in.
- If yes, show home1.html.
- If no, redirect to login page.
- Show About page.

- Show Contact Us page.

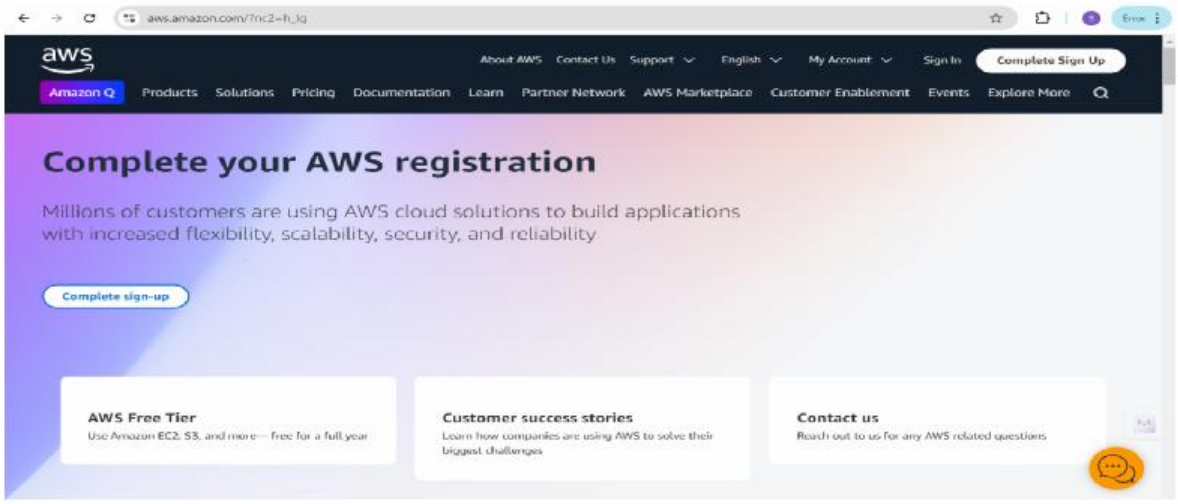
6.Running the Flask App:

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

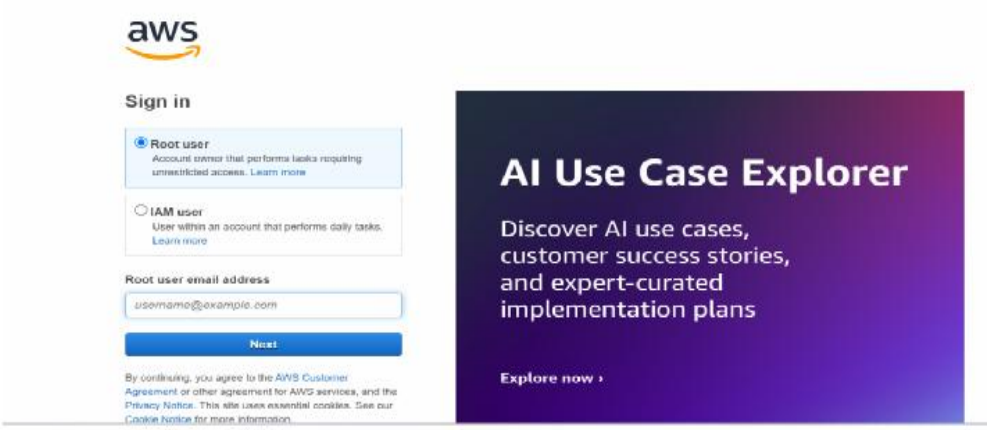
- Starts the app on your machine at http://localhost:5000.
- debug=True means it auto-reloads when you change the code.

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.

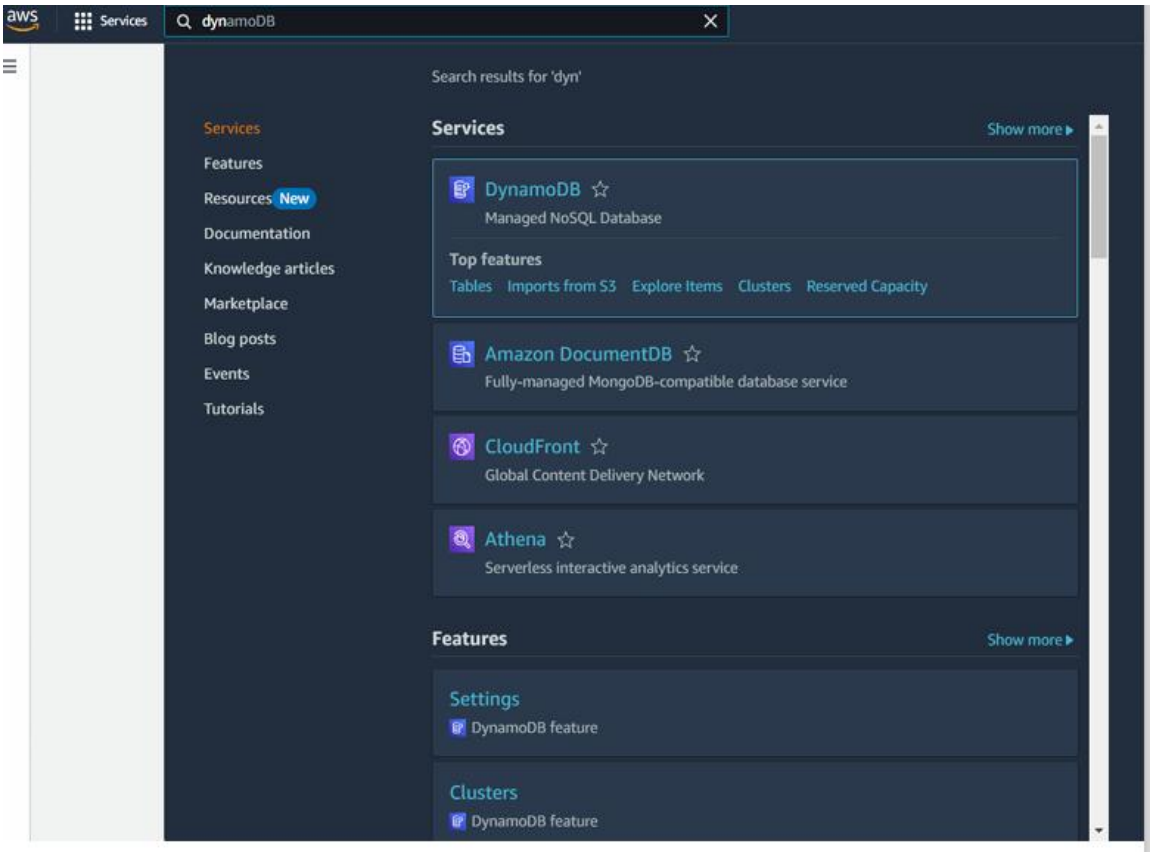


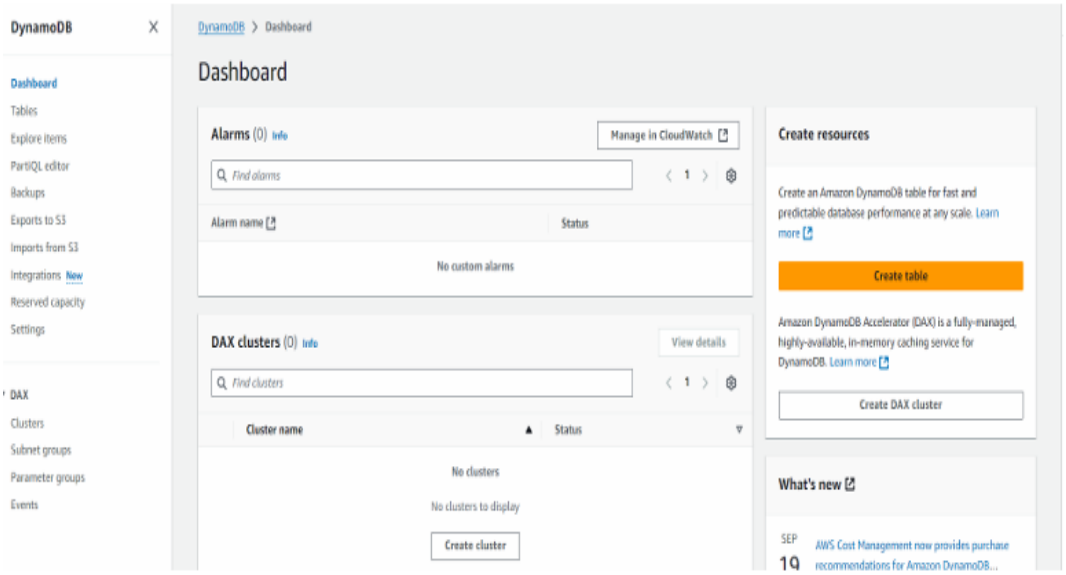
- **Activity 1.2: Log in to the AWS Management Console**
  - After setting up your account, log in to the AWS Management Console.

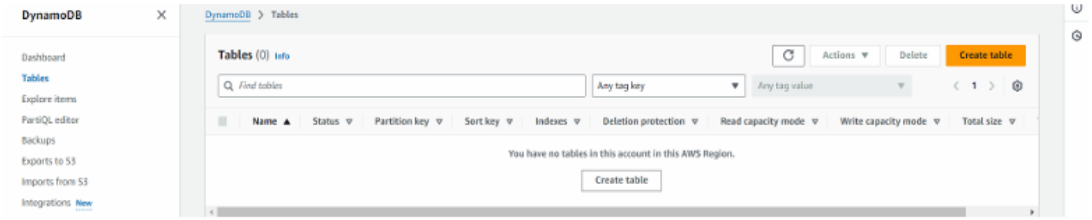


2: DynamoDB Database Creation and Setup

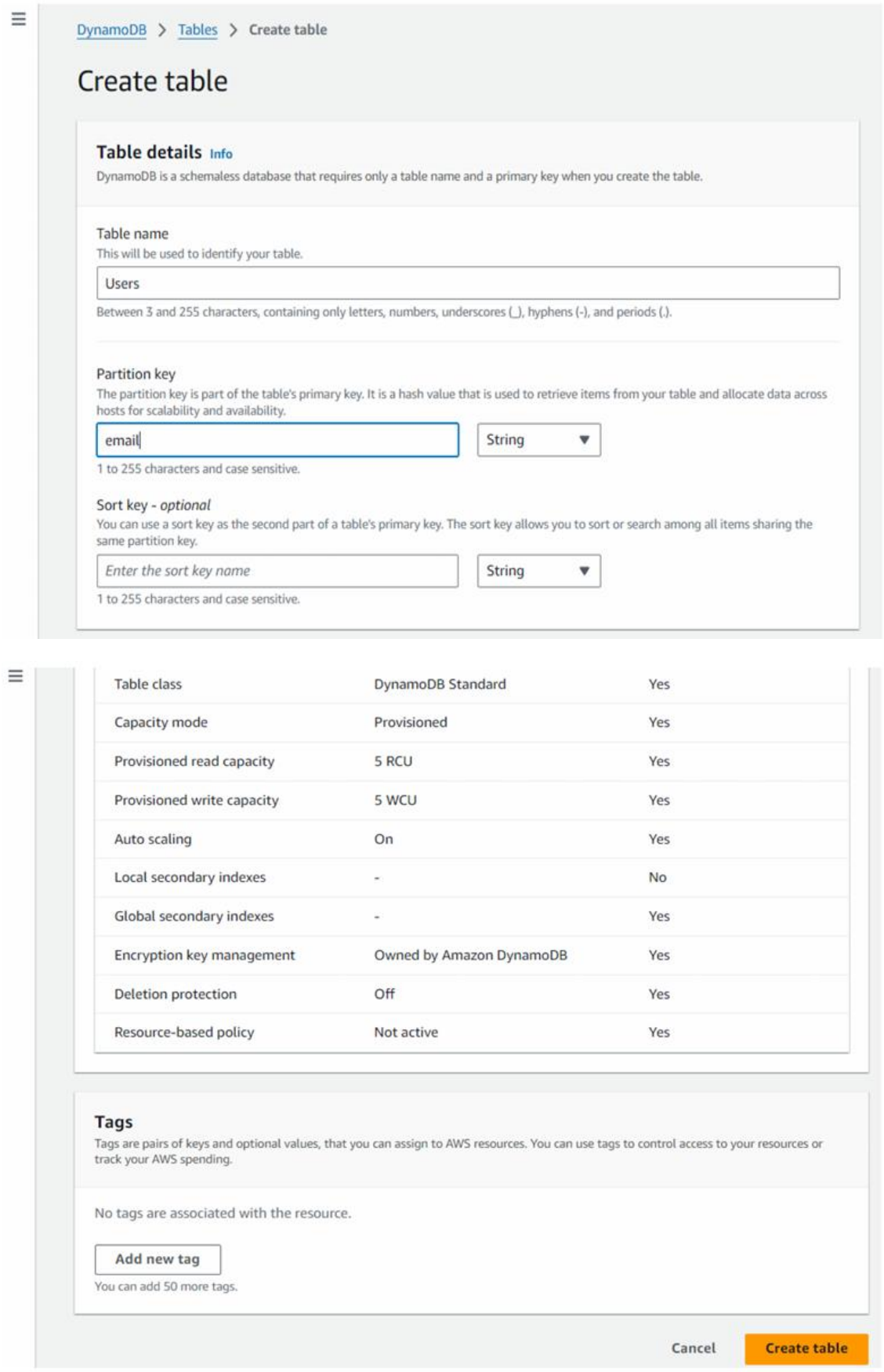
- **Activity 2.1:Navigate to the DynamoDB**
  - In the AWS Console, navigate to DynamoDB and click on create tables.

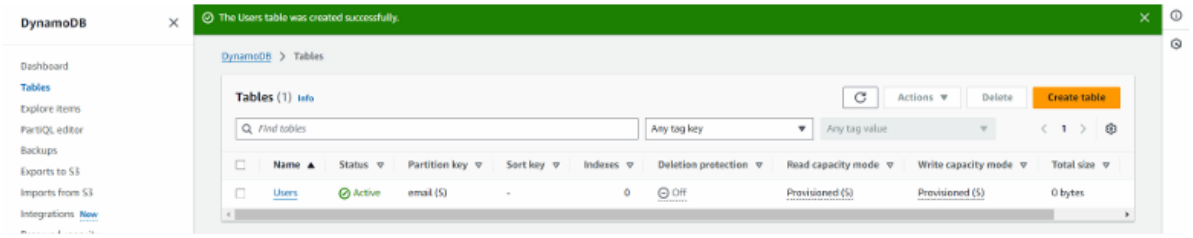




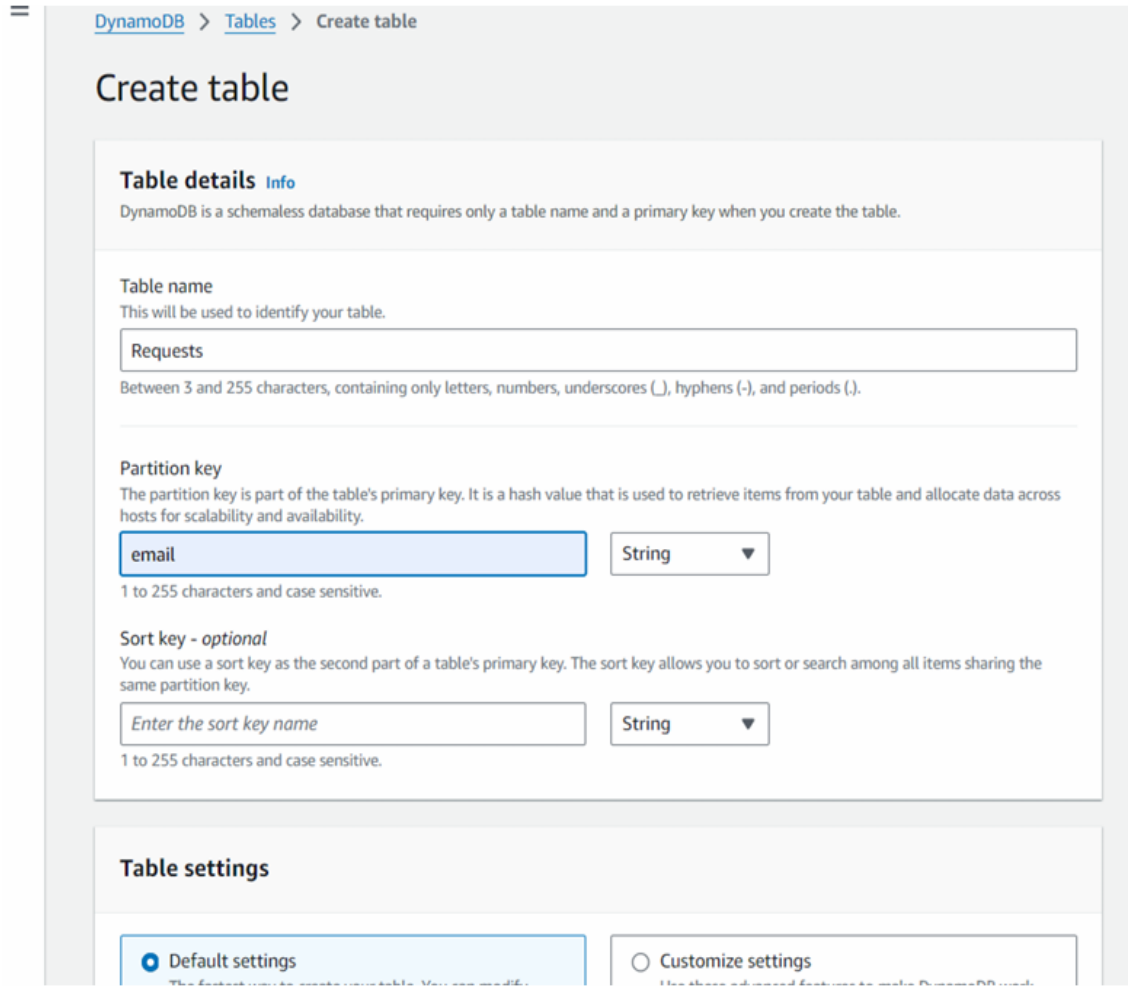


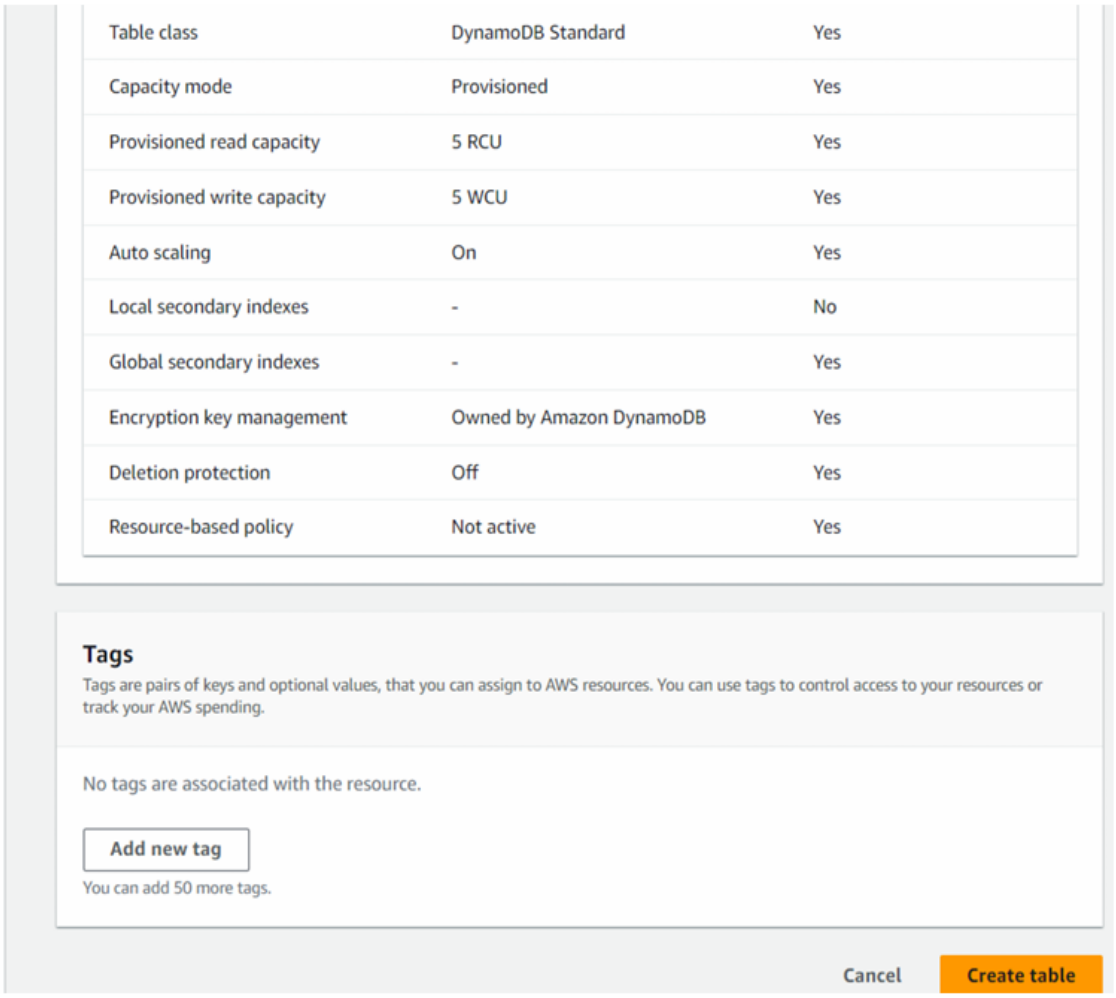
- **Activity 2.2:Create a DynamoDB table for storing registration details and book requests.**
    - Create Users table with partition key “Email” with type String and click on create tables.

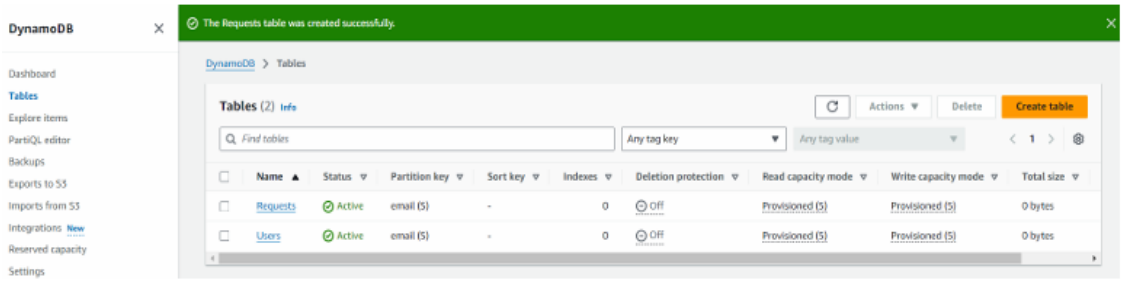

DynamoDB > Tables > Create table
Create table
Table details Info
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.
Table name
This will be used to identify your table.
Users
Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).
Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
email
String
1 to 255 characters and case sensitive.
Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
Enter the sort key name
String
1 to 255 characters and case sensitive.
Table class DynamoDB Standard Yes
Capacity mode Provisioned Yes
Provisioned read capacity 5 RCU Yes
Provisioned write capacity 5 WCU Yes
Auto scaling On Yes
Local secondary indexes - No
Global secondary indexes - Yes
Encryption key management Owned by Amazon DynamoDB Yes
Deletion protection Off Yes
Resource-based policy Not active Yes
Tags
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.
No tags are associated with the resource.
Add new tag
You can add 50 more tags.
Cancel
Create table



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



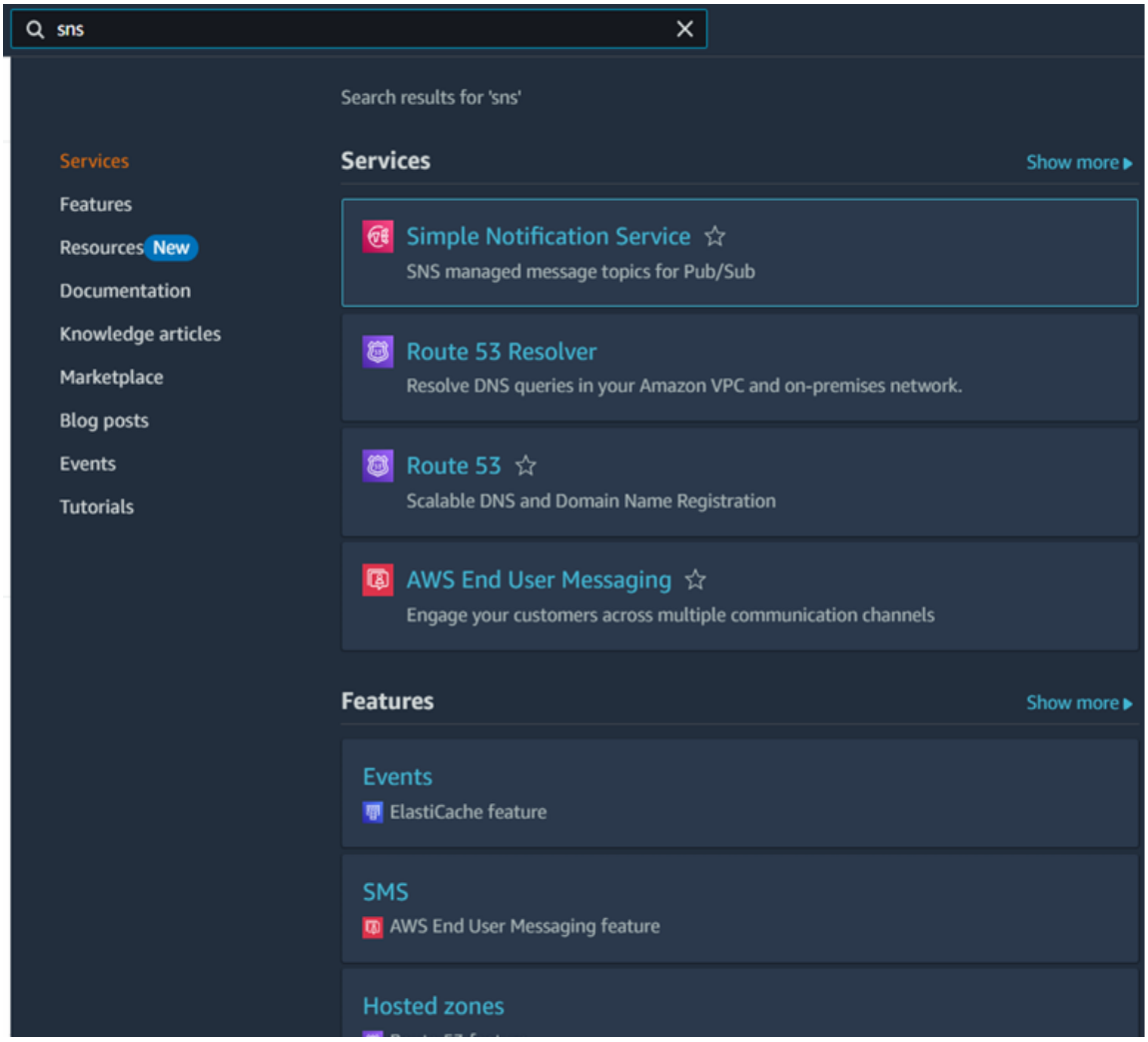


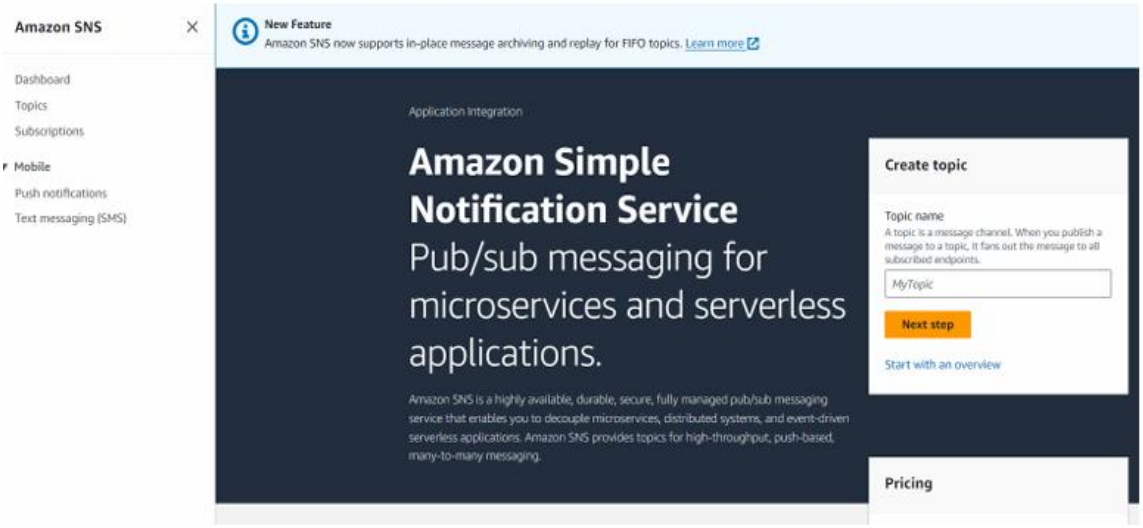


3: SNS Notification Setup

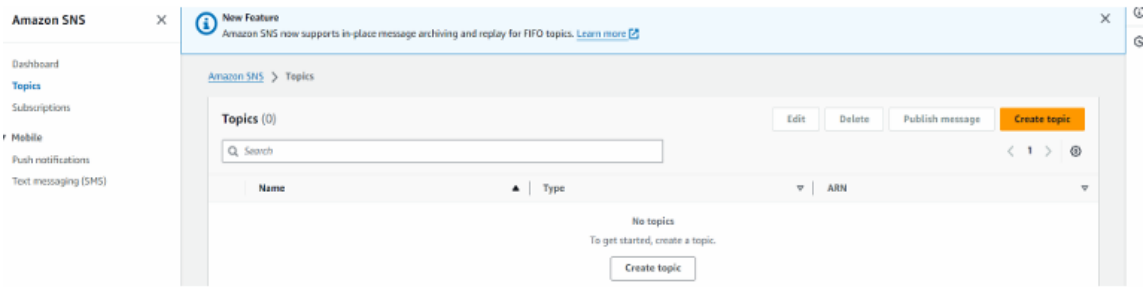
- **Activity 3.1: Create SNS topics for sending email notifications to users and library staff.**

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

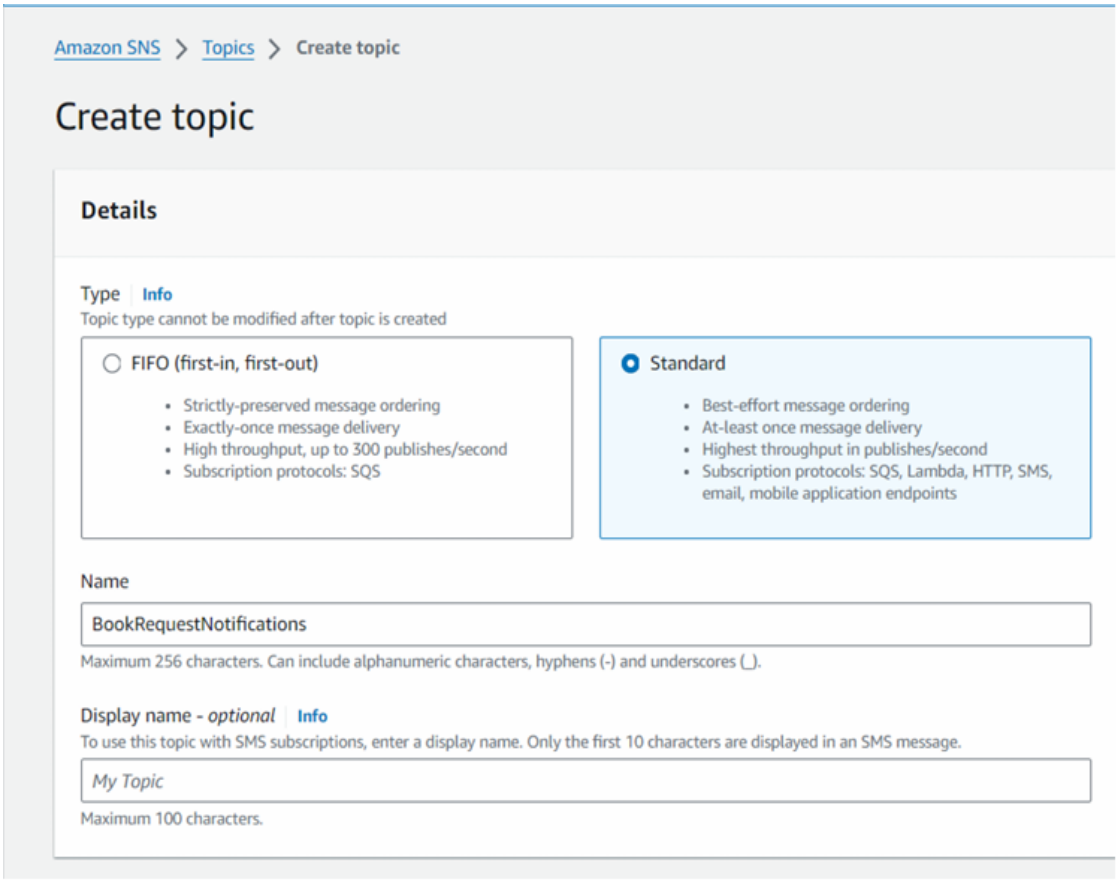


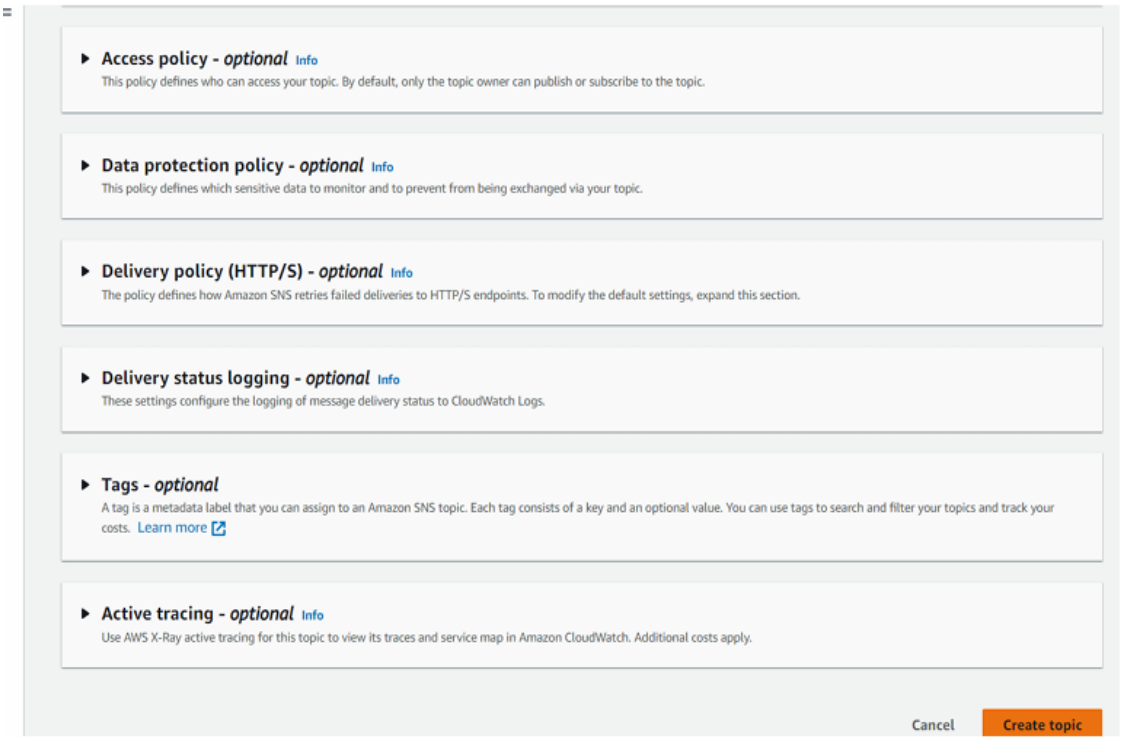


○ Click on **Create Topic** and choose a name for the topic.

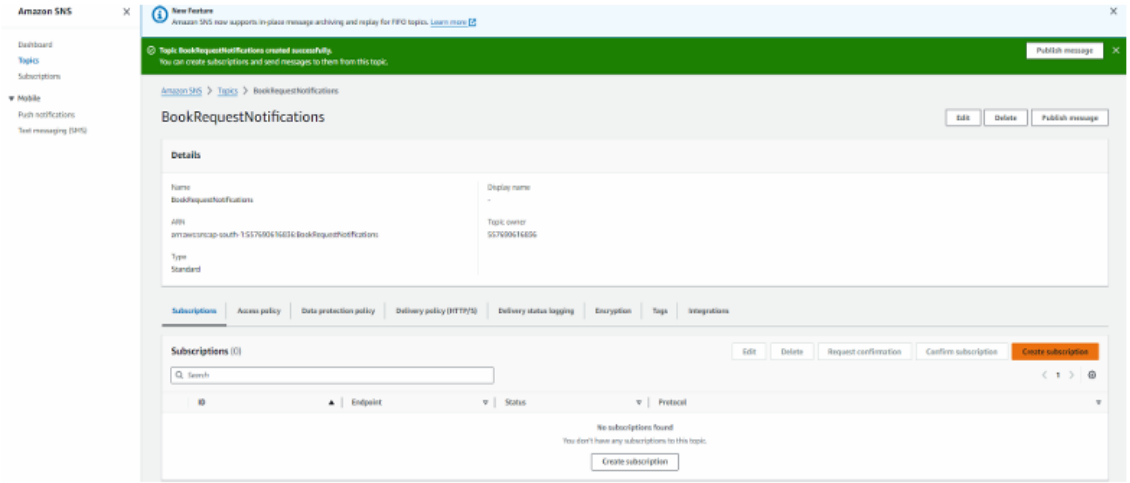


o Choose Standard type for general notification use cases and Click on Create Topic.

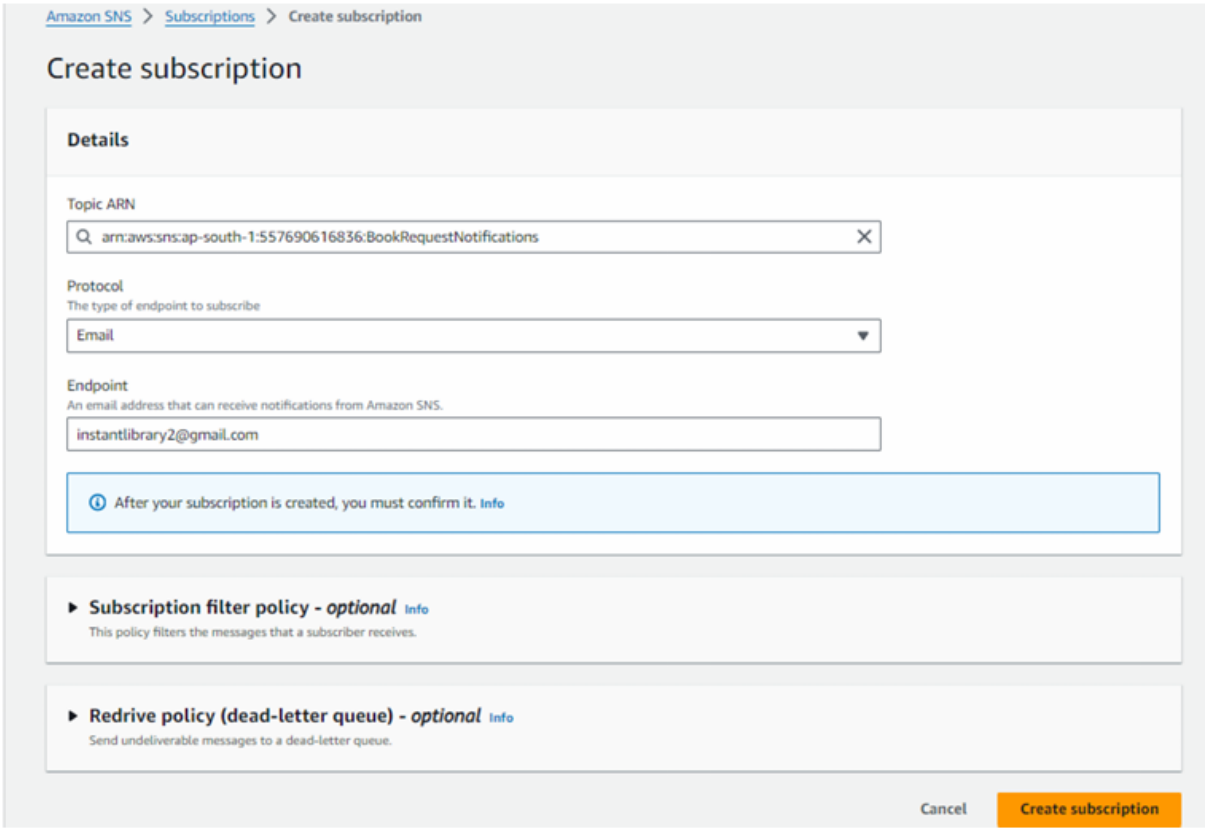


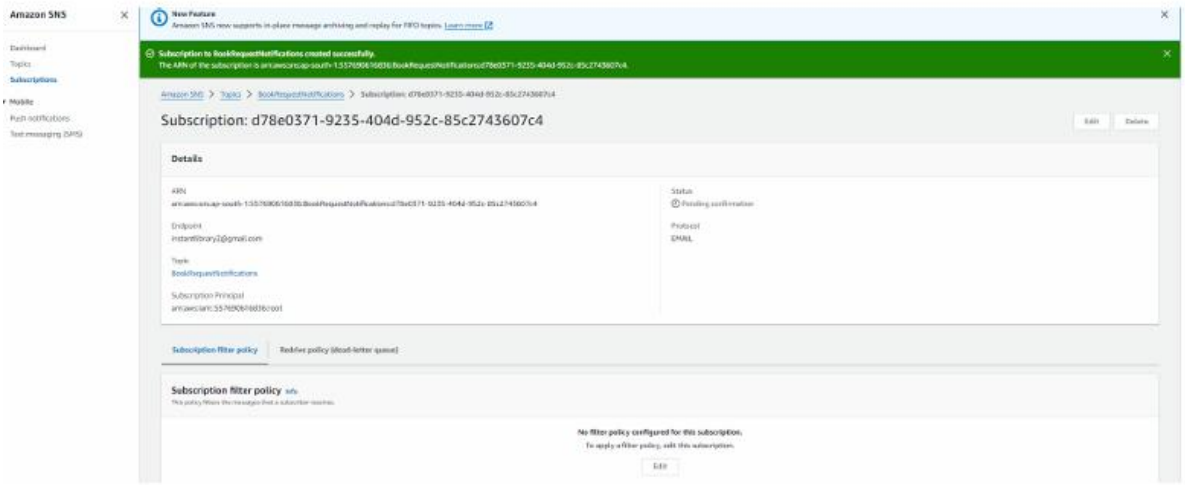


- Configure the SNS topic and note down the Topic ARN.

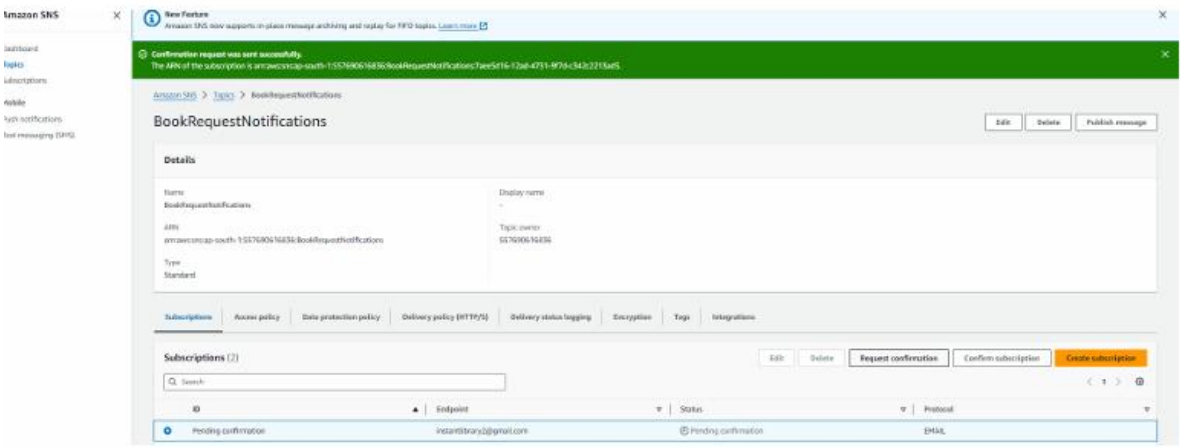


- **Activity 3.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.**
  - **Subscribe users (or admin staff) to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails.**

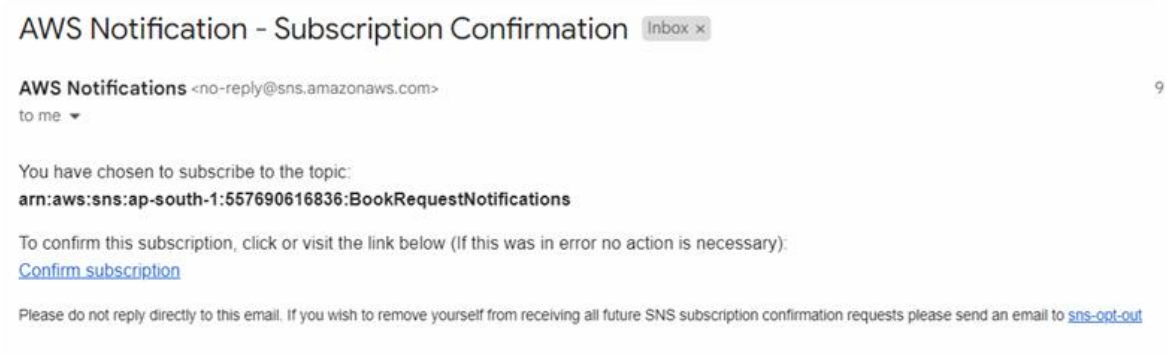


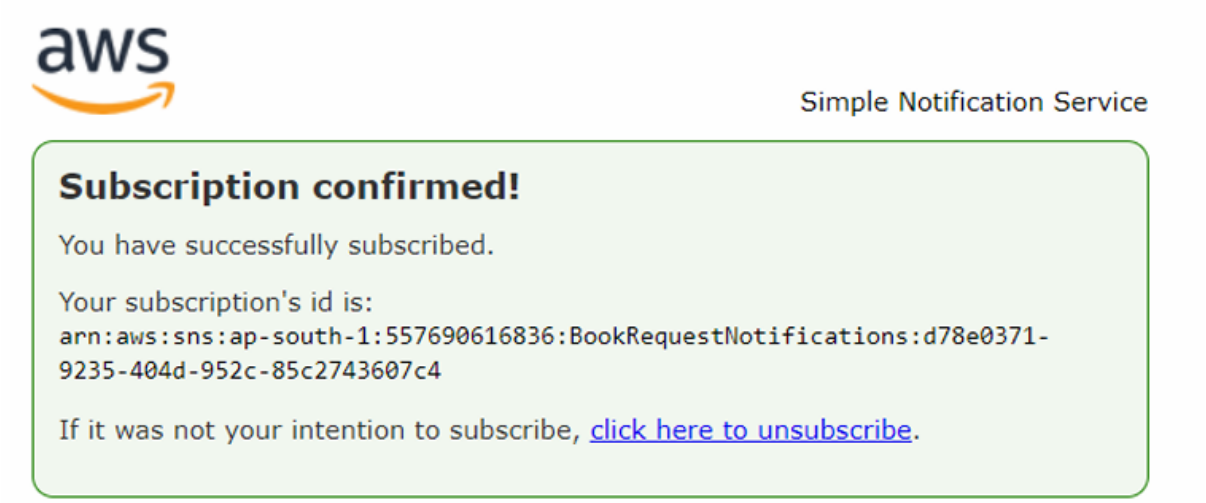


o After subscription request for the mail confirmation

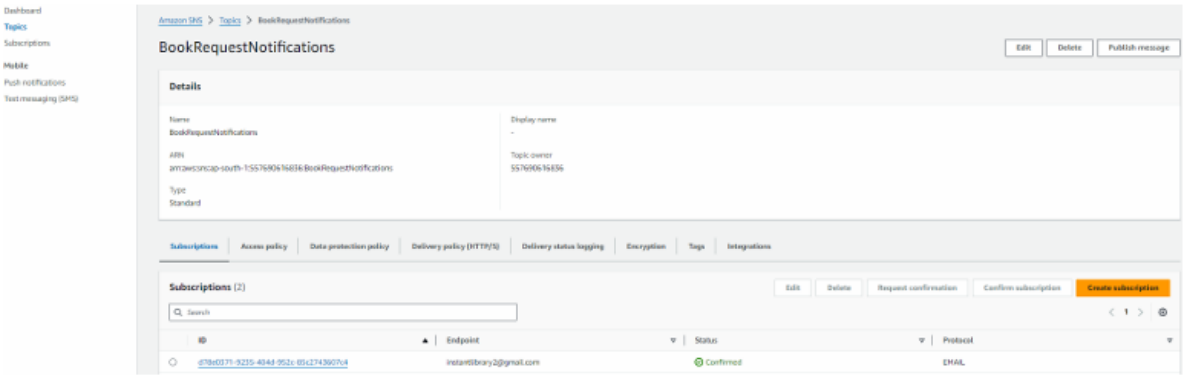


o Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.





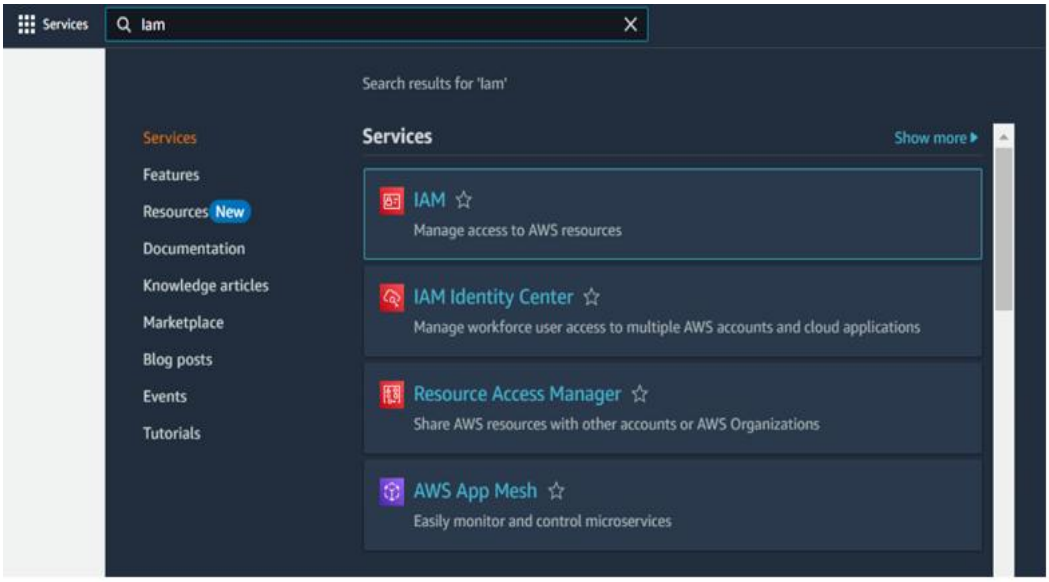
- Successfully done with the SNS mail subscription and setup, now store the ARN link.

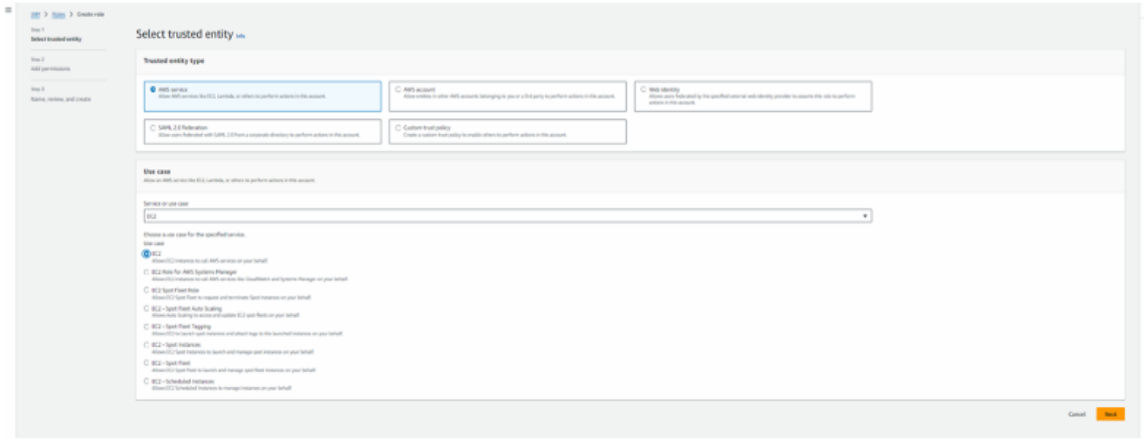


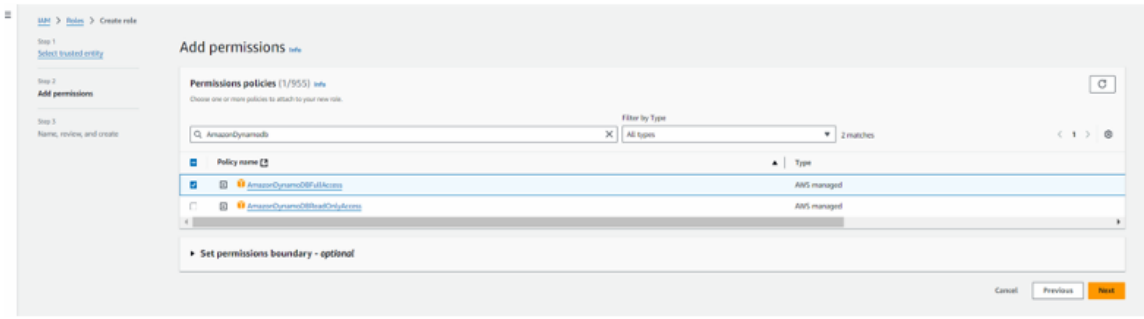
5: IAM Role Setup

- **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.



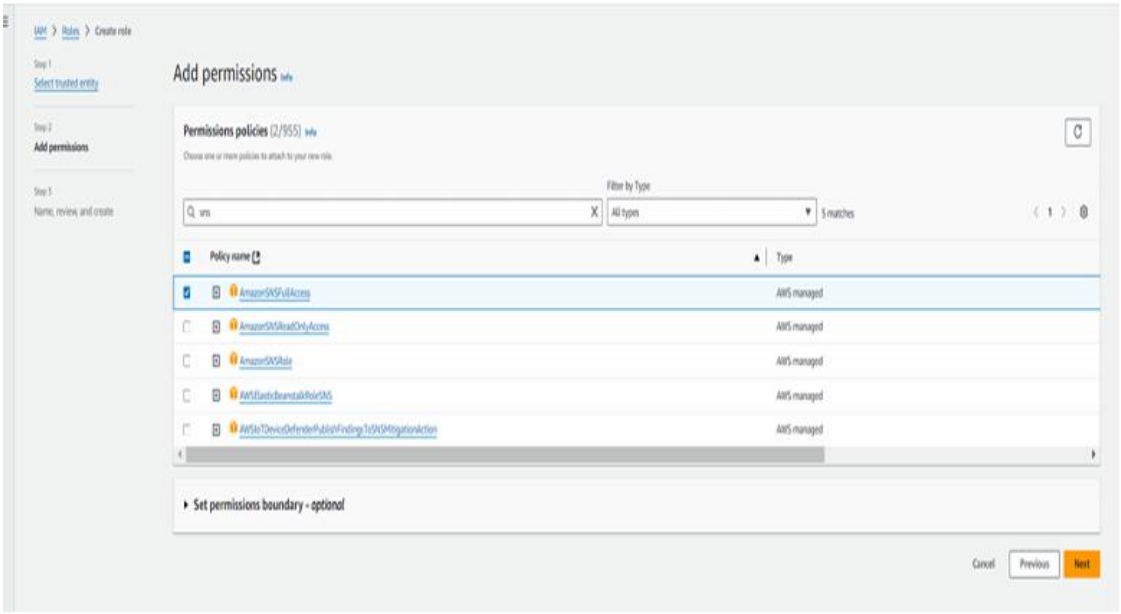


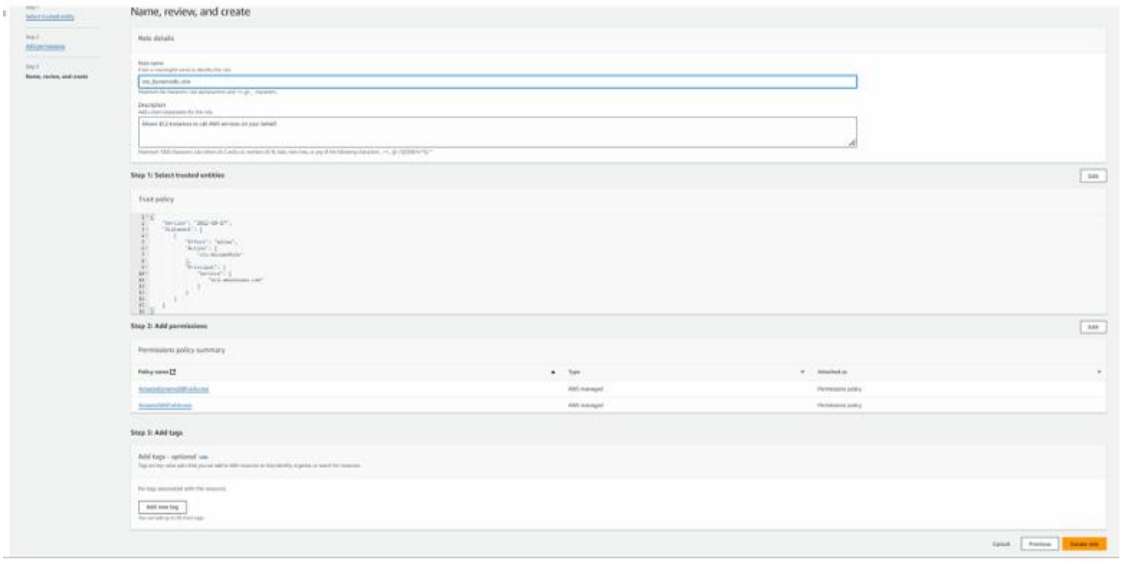


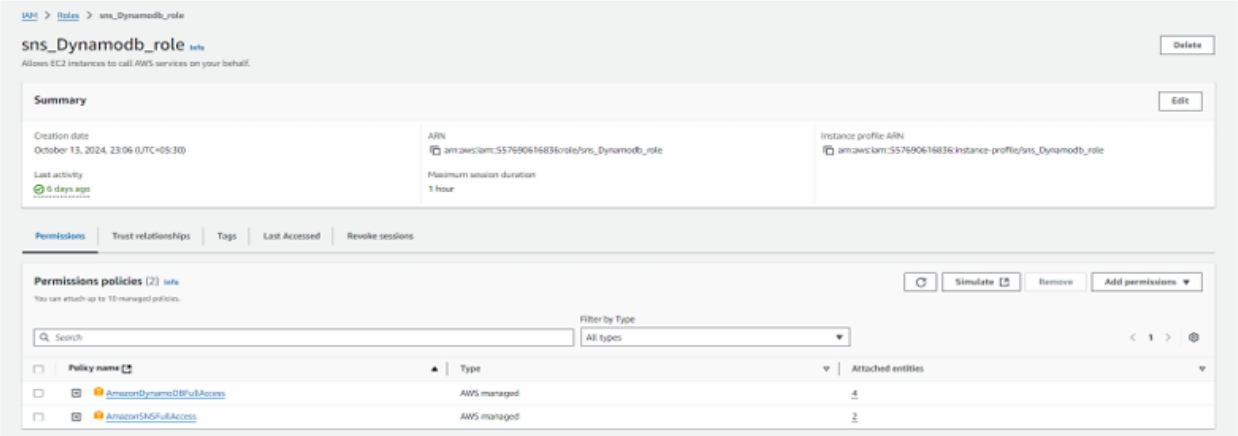
- **Activity 5.2: Attach Policies.**

  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.


IAM > Roles > Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
Add permissions Info
Permissions policies (2/955) Info
Choose one or more policies to attach to your new role.
Filter by Type
sns
All types
5 matches
Policy name
Type
AmazonSNSFullAccess
AWS managed
AmazonSNSReadOnlyAccess
AWS managed
AmazonSNSRole
AWS managed
AWSElasticBeanstalkRoleSNS
AWS managed
AWSIoTDeviceDefenderPublishFindingsToSNSMitigationAction
AWS managed
Set permissions boundary - optional
Cancel
Previous
Next

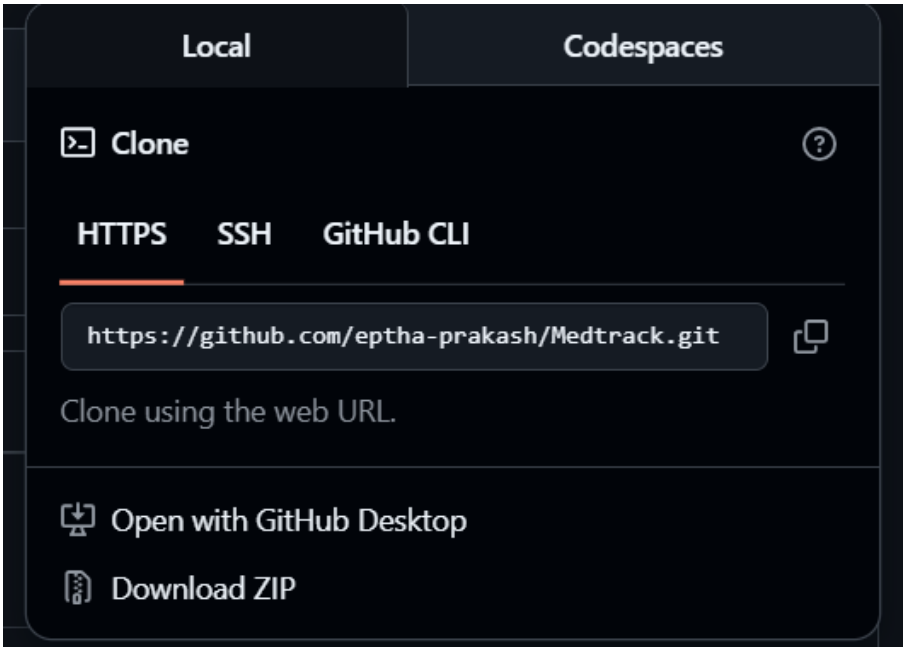

Name, review, and create
Role details
Step 1: Select trusted entities
Trust policy
Step 2: Add permissions
Permissions policy summary
Step 3: Add tags



Milestone 6: EC2 Instance Setup

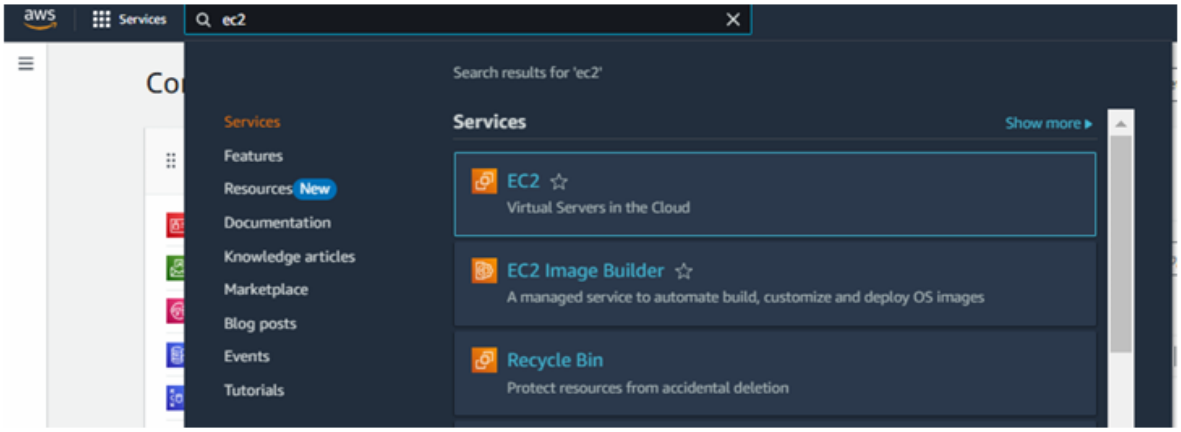
- **Note: Load your Flask app and Html files into GitHub repository.**

Name	Last commit message	Last commit date
..		
templates	Delete projectfiles/templates/Activities.html	last week
medtrack.py	Add files via upload	last week
readme.md	Create readme.md	last week

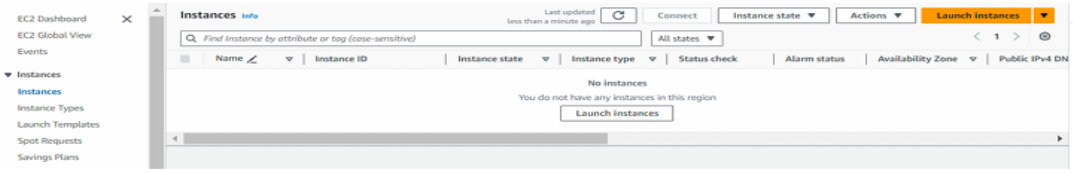


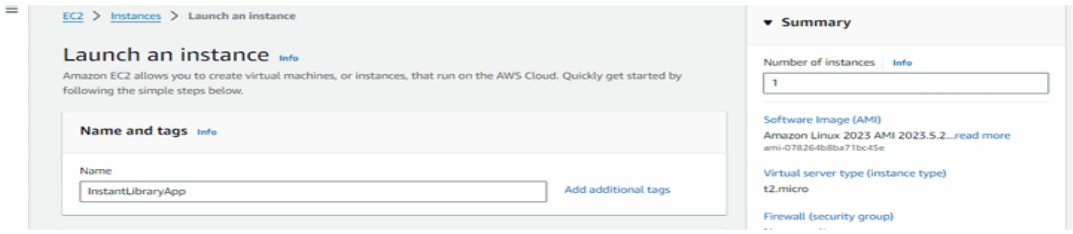
- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

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

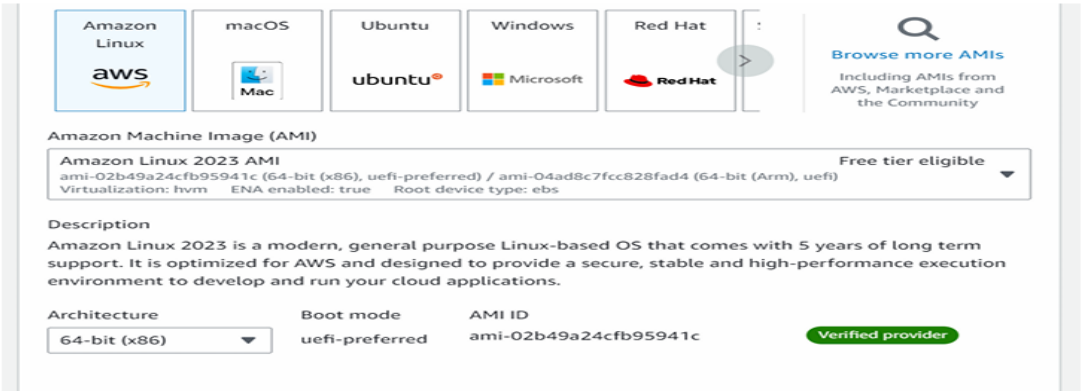


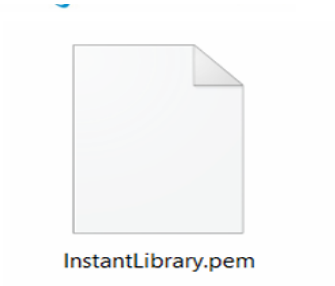
- Click on Launch instance to launch EC2 instance

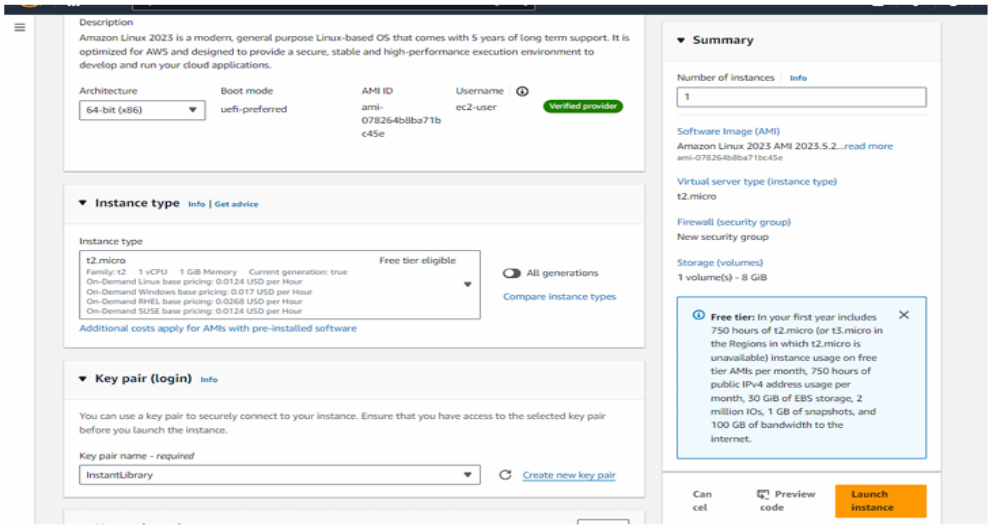




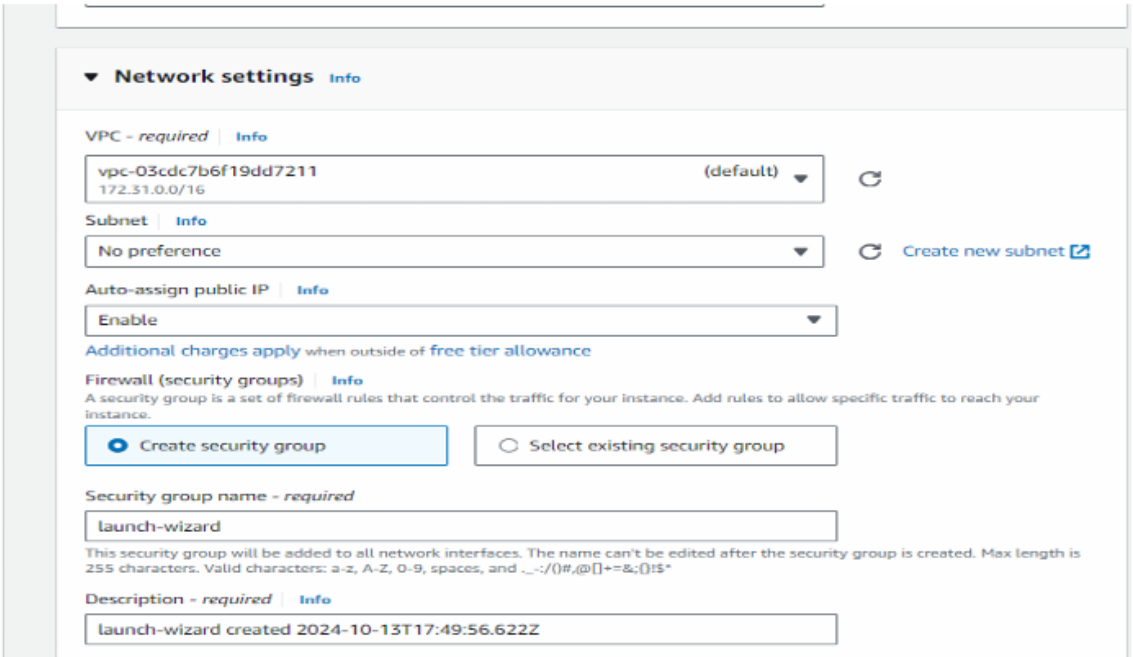
- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).

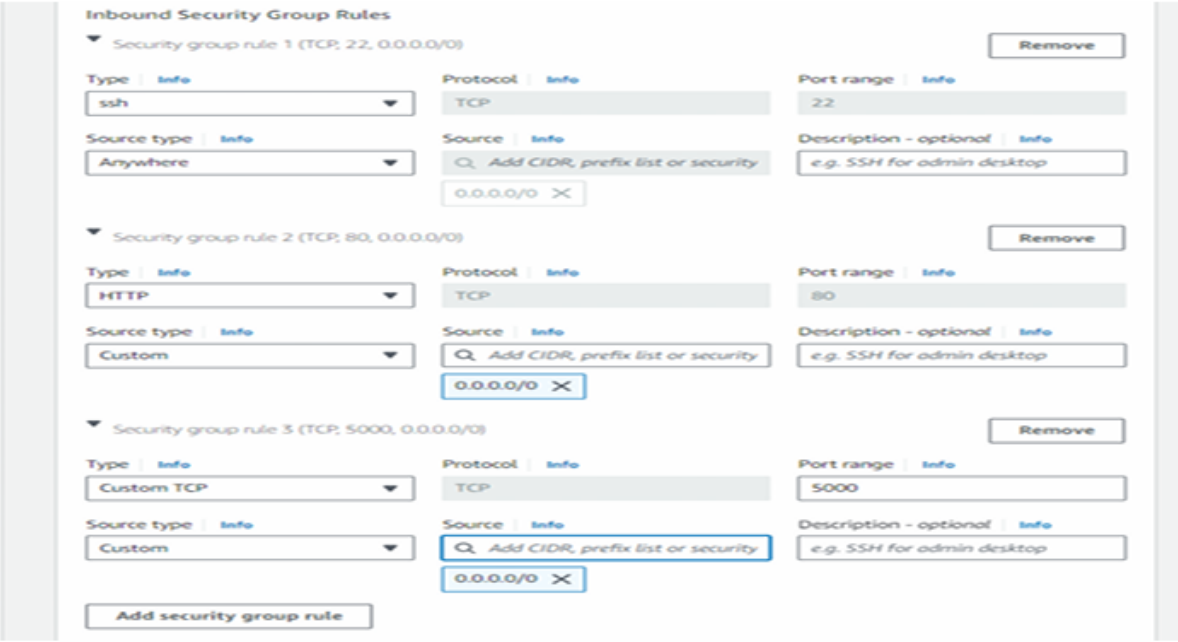


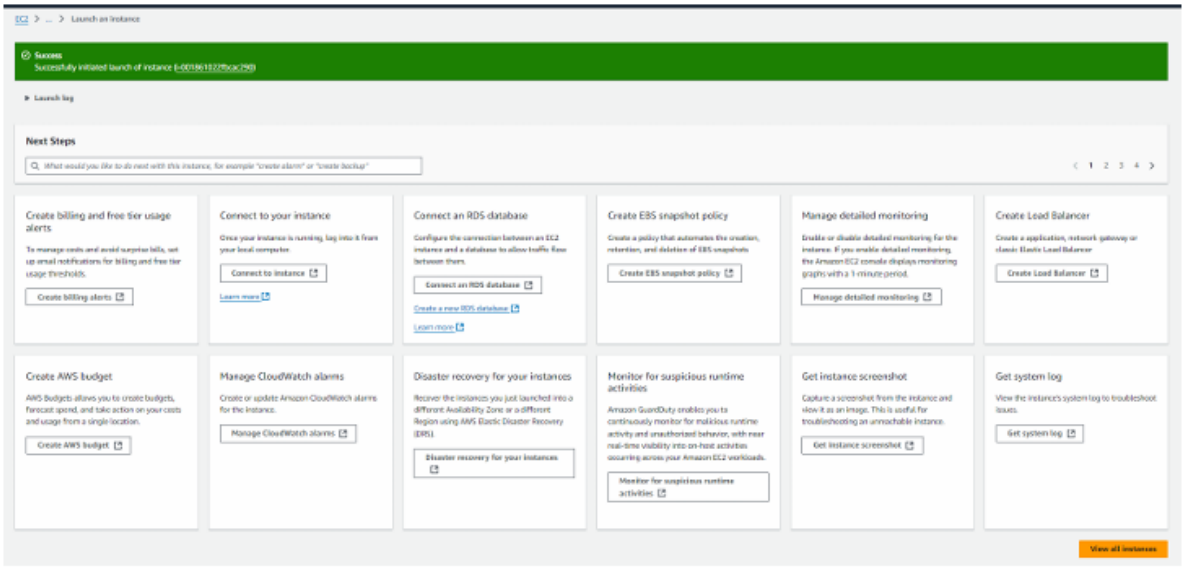




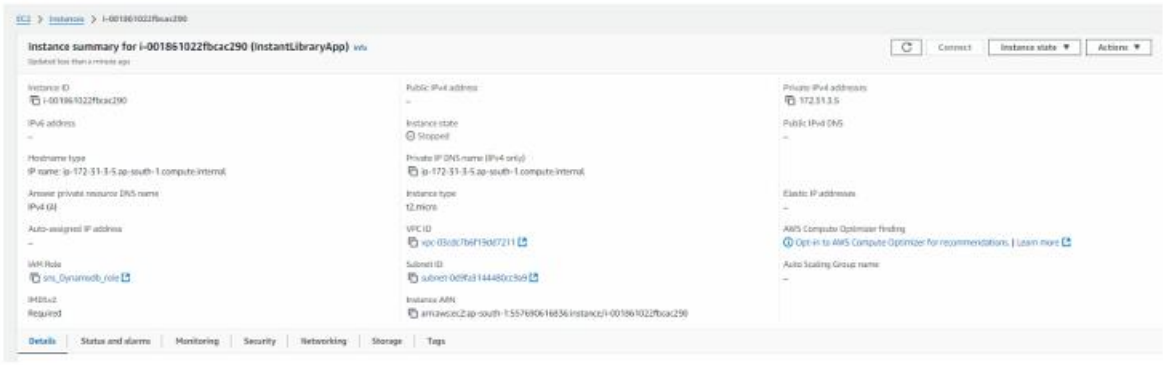
- **Activity 6.2:Configure security groups for HTTP, and SSH access.**

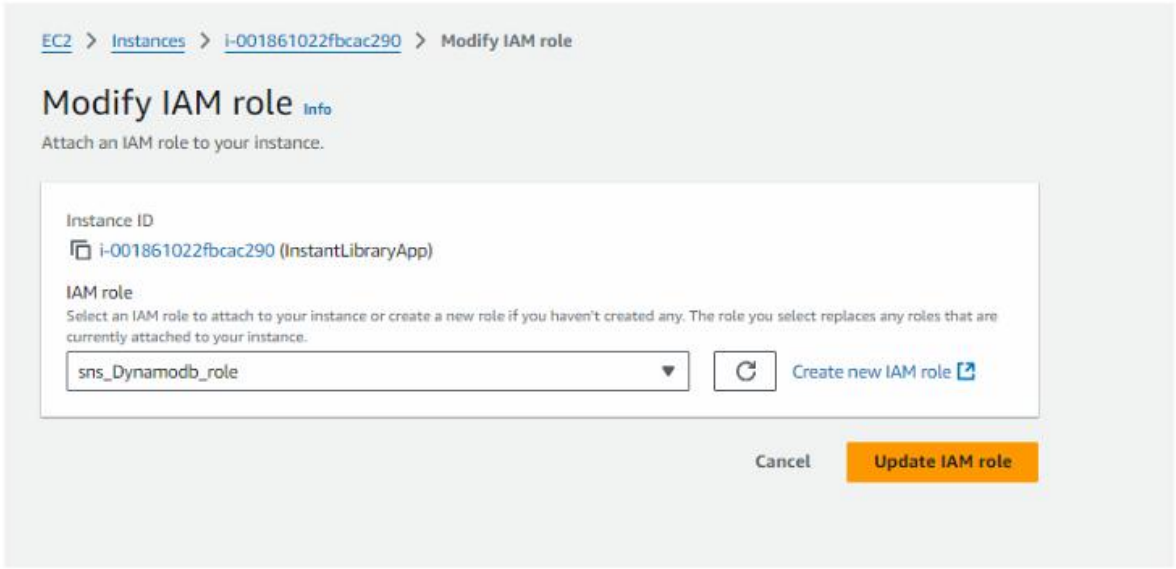




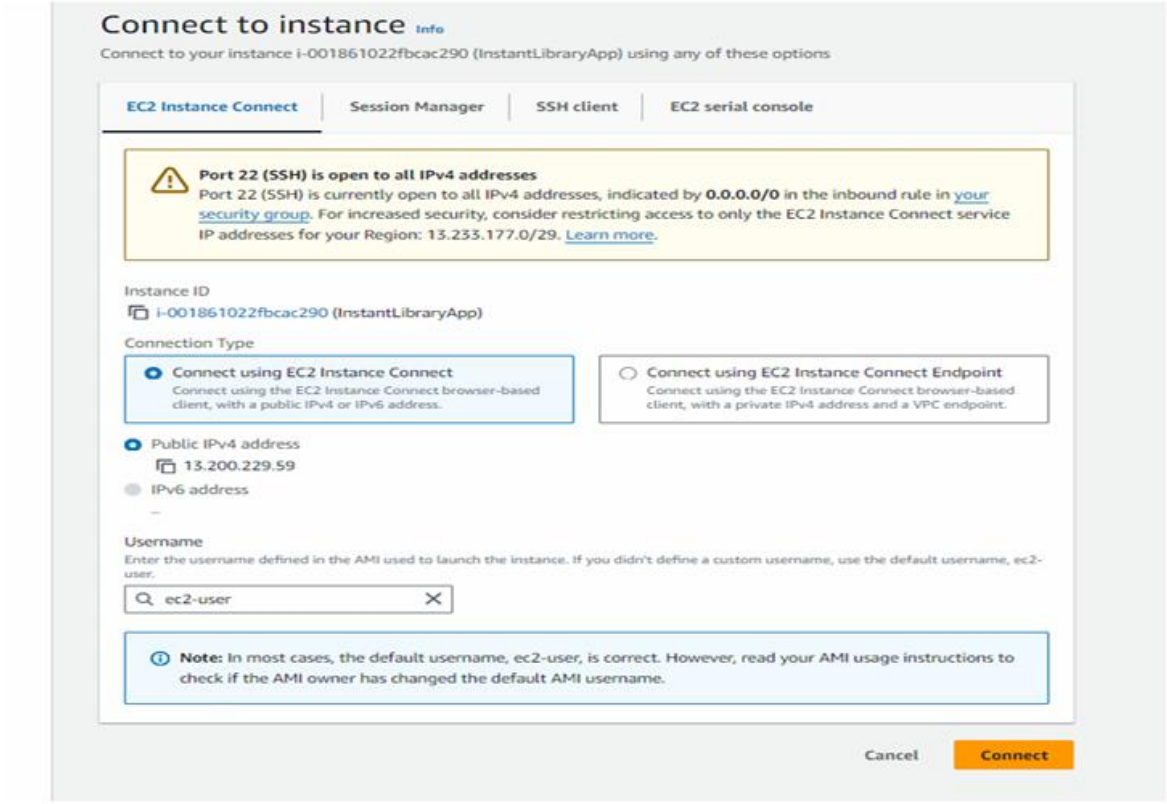


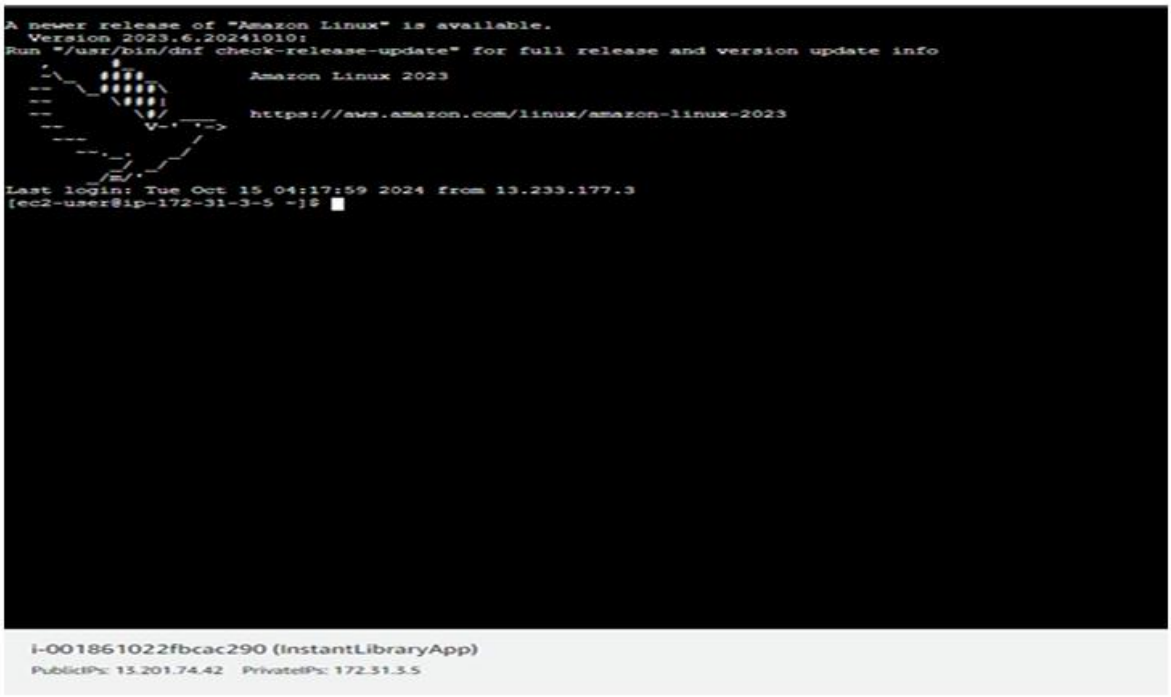
- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the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


Connect to instance Info
Connect to your instance i-001861022fbcac290 (InstantLibraryApp) using any of these options
EC2 Instance Connect
Session Manager
SSH client
EC2 serial console
Port 22 (SSH) is open to all IPv4 addresses
Port 22 (SSH) is currently open to all IPv4 addresses, indicated by 0.0.0.0/0 in the inbound rule in your security group. For increased security, consider restricting access to only the EC2 Instance Connect service IP addresses for your Region: 13.233.177.0/29. Learn more.
Instance ID
i-001861022fbcac290 (InstantLibraryApp)
Connection Type
Connect using EC2 Instance Connect
Connect using the EC2 Instance Connect browser-based client, with a public IPv4 or IPv6 address.
Connect using EC2 Instance Connect Endpoint
Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.
Public IPv4 address
13.200.229.59
IPv6 address
Username
Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.
ec2-user
Note: In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.
Cancel
Connect


A newer release of "Amazon Linux" is available.
Version 2023.6.20241010:
Run "/usr/bin/dnf check-release-update" for full release and version update info
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
Last login: Tue Oct 15 04:17:59 2024 from 13.233.177.3
[ec2-user@ip-172-31-3-5 ~]$
i-001861022fbcac290 (InstantLibraryApp)
PublicIPs: 13.201.74.42 PrivateIPs: 172.31.3.5

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'

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

Here: 'git clone https://github.com/eptha-prakash/Medtrack.git'

This will download your project to the EC2 instance.

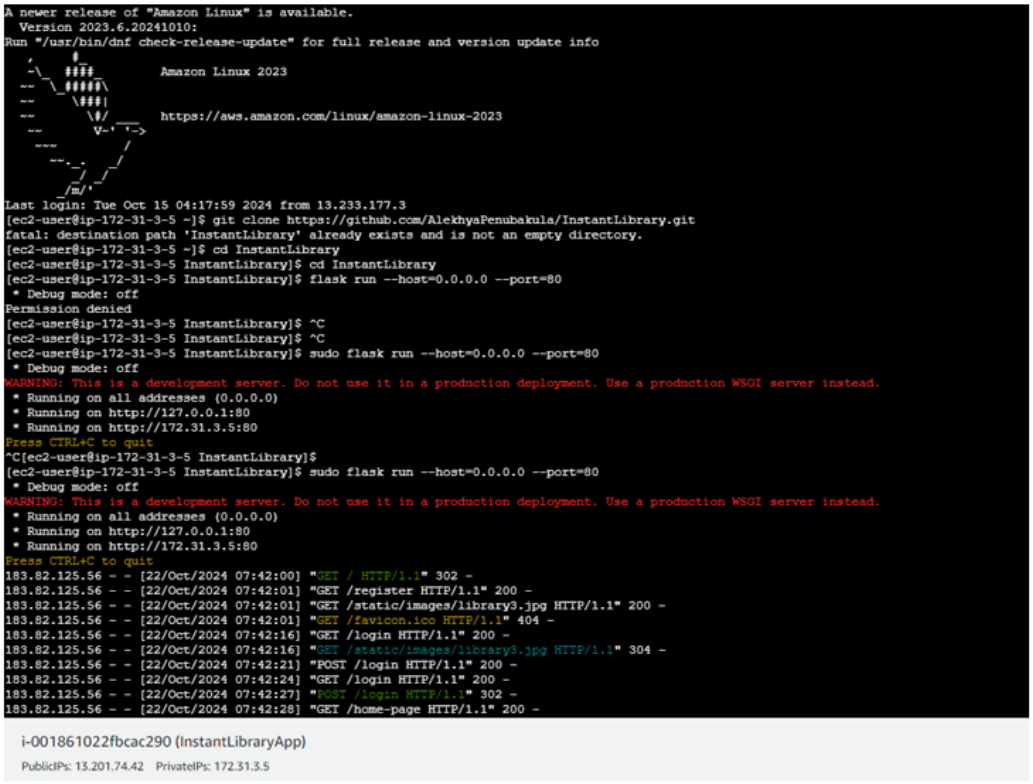
To navigate to the project directory, run the following command:

```
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

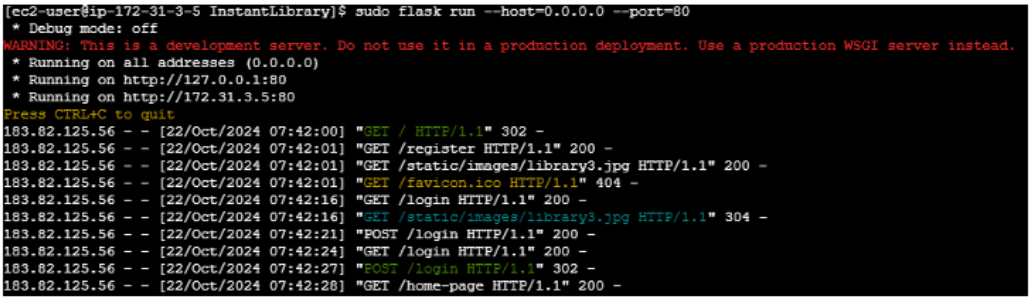
```
sudo flask run --host=0.0.0.0 --port=80
```



Verify the Flask app is running:

http://your-ec2-public-ip

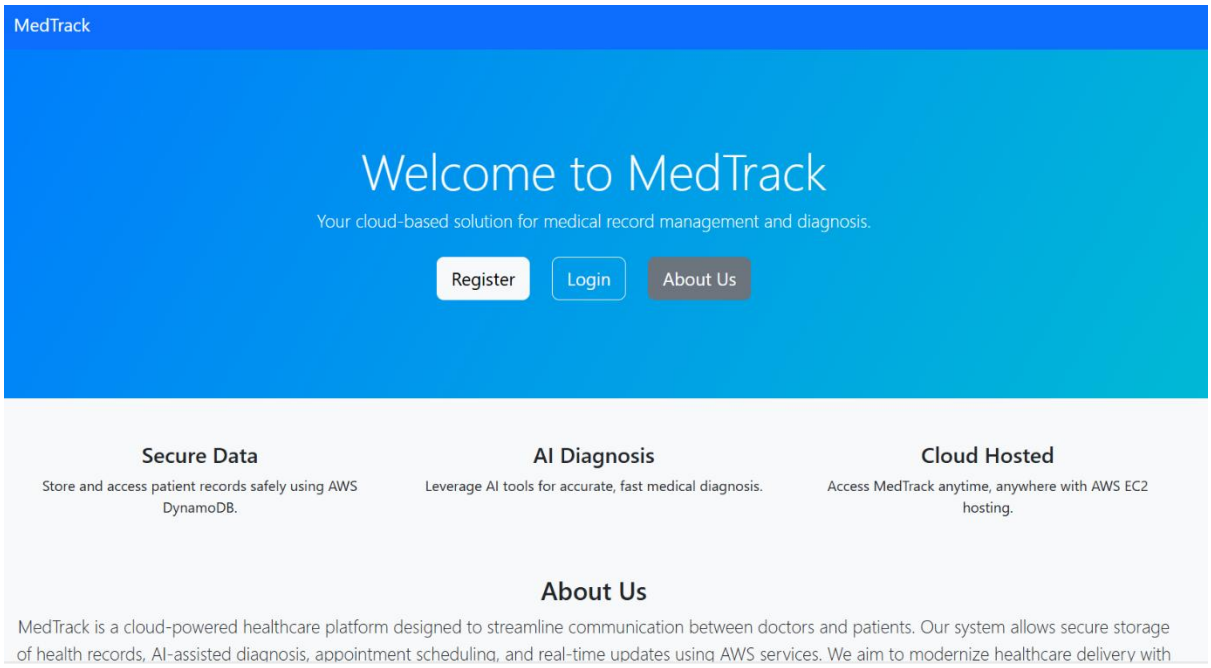
- Run the Flask app on the EC2 instance



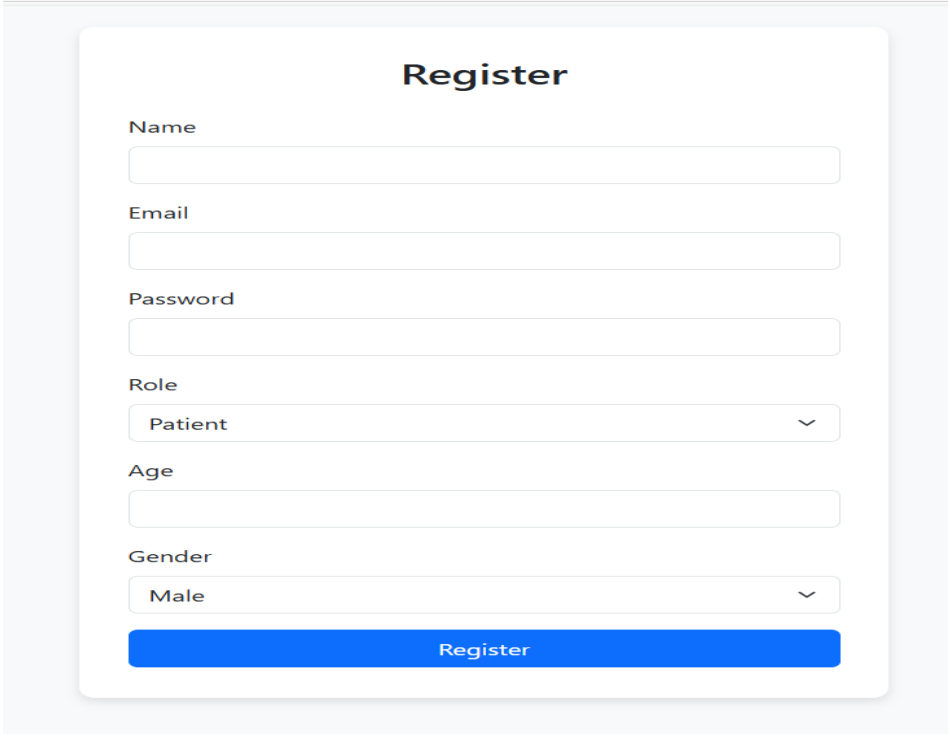
Milestone 8: Testing and Deployment

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

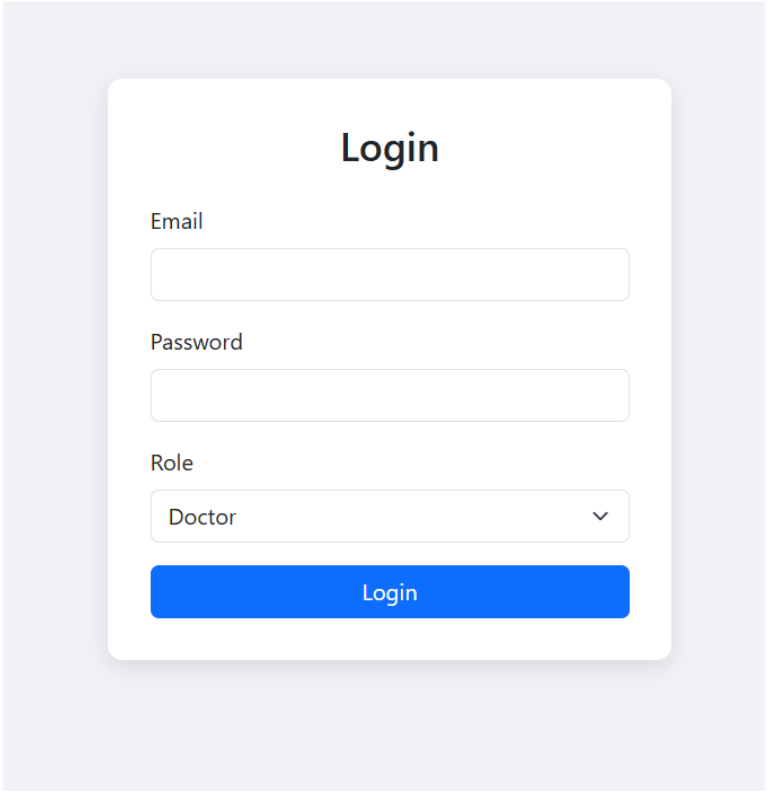
Home Page:



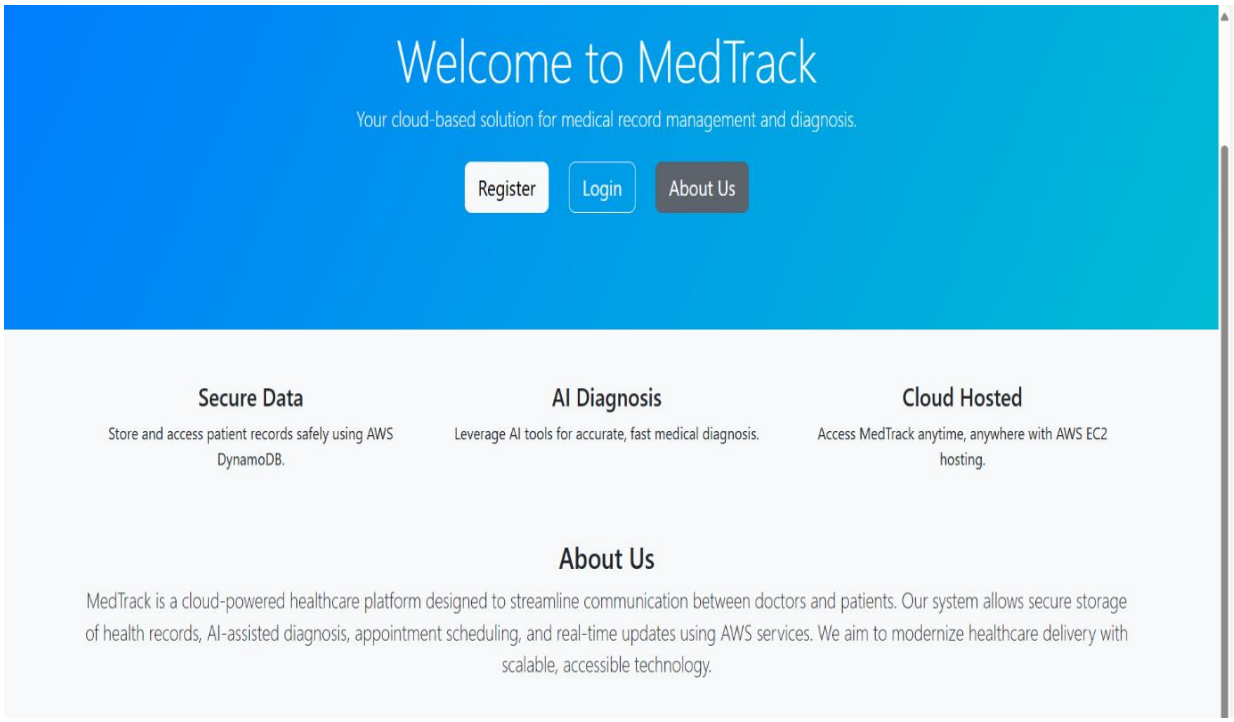
Register page:



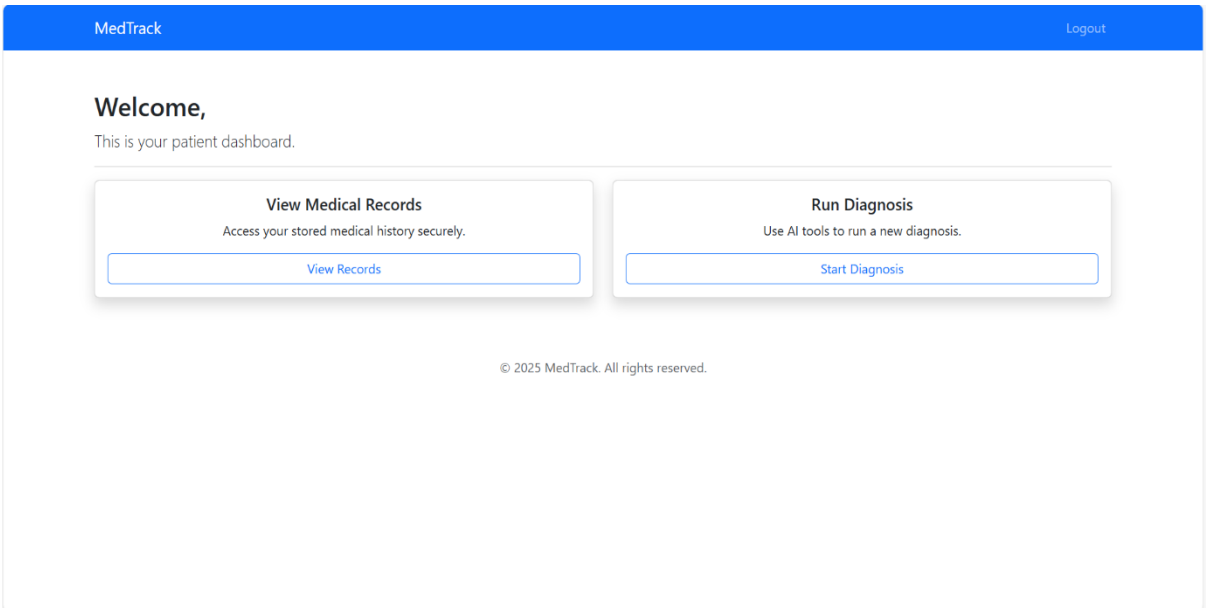
Login Page:



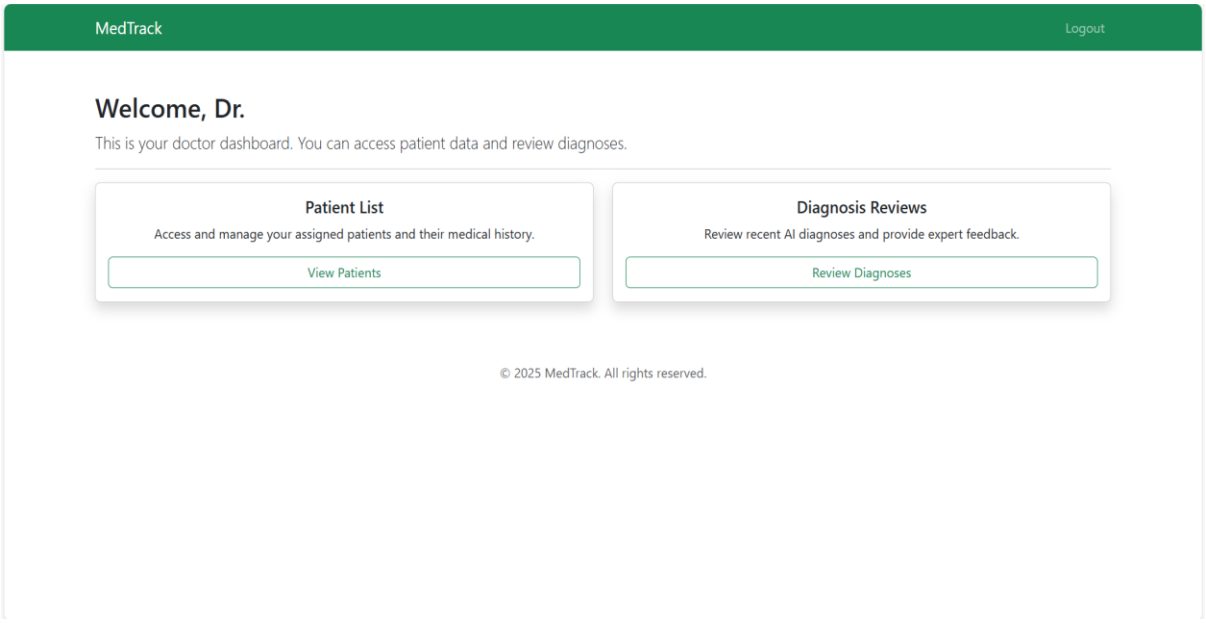
About Us Page:



Patient Dashboard:



Doctor 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.

Reference :

Github : https://github.com/eptha-prakash/Medtrack.git

Video demo : https://drive.google.com/drive/folders/1NC7-2I_dYx8HPYgeG4sDNl4O-tZZSZxo?usp=sharing