# MedTrack: AWS Cloud-Enabled Healthcare Management System

**Project Description**:

In today's fast-evolving healthcare landscape, ef icient 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 noti ications, 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 ef iciency in healthcare services for all users.

**Scenarios**:

**Scenario 1:** Ef icient 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, ef iciently 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.

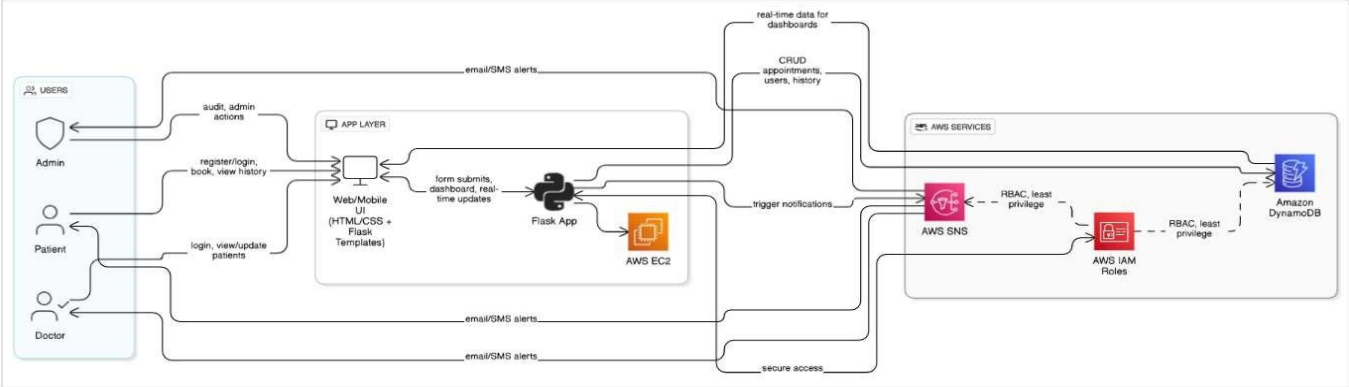
Scenario 2: Secure User Management with IAM

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 speci ic roles and permissions to access only the features relevant to them. Doctors have their own IAM con igurations, 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.

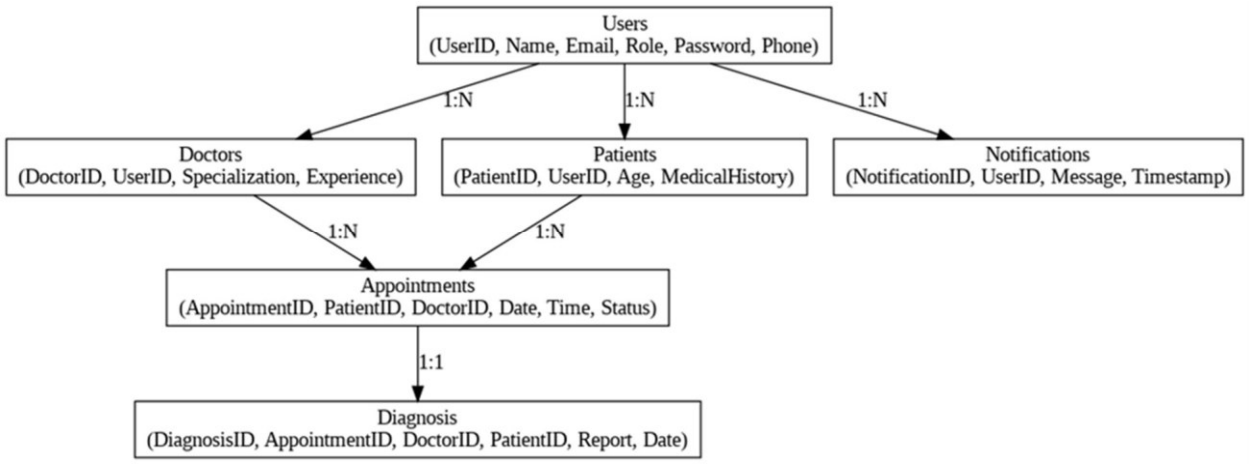
Scenario 3: Easy Access to Medical History and Resources

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: AWS Account Setup
2. Understanding IAM: IAM Overview
3. Amazon EC2 Basics: EC2 Tutorial
4. DynamoDB Basics: DynamoDB Introduction
5. SNS Overview: SNS Documentation

6. Git Version Control: Git Documentation

**Project WorkFlow**

**Milestone 1**. Web Application Development and SetupDevelop 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.
- Con igure Attributes for User Data and Book Requests.

**Milestone 4**. SNS Noti ication Setup

- Create SNS topics for book request noti ications.
- Subscribe users and library staff to SNS email noti ications.

**Milestone 5**. IAM Role Setup

- Create IAM Role
- Attach Policies

**Milestone 6**. EC2 Instance Setup

- Launch an EC2 instance to host the Flask application.
- Con igure security groups for HTTP, and SSH access.

**Milestone 7.** Deployment using EC2

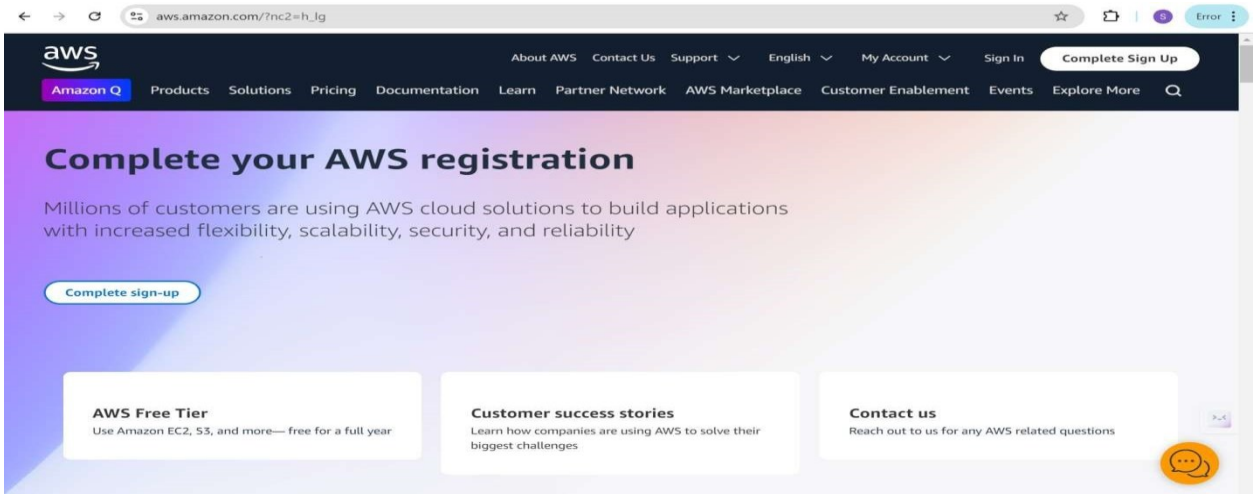
- Upload Flask Files
- Run the Flask App

**Milestone 8**. Testing and Deployment

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

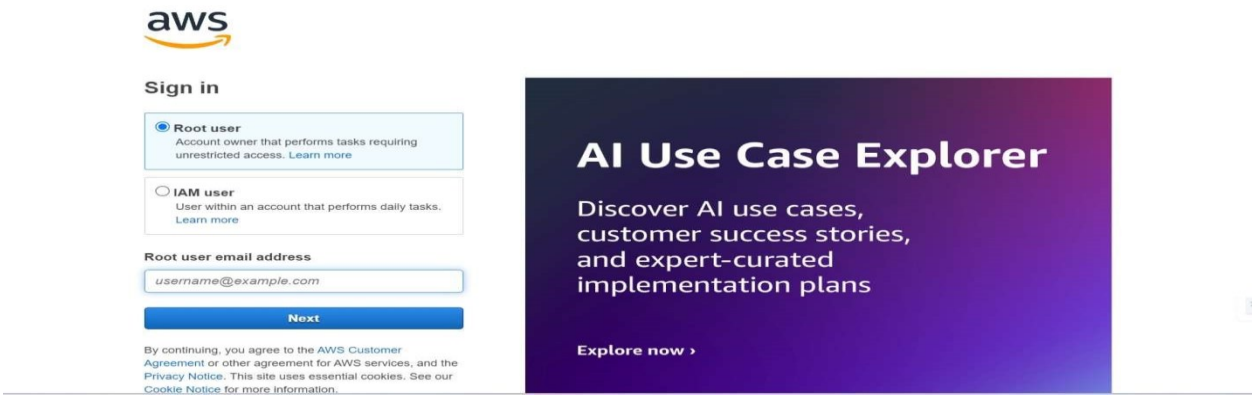
**Milestone 1:** Web Application Development and Setup

- Activity 1.1: Set up an AWS account if not already done.
    - Sign up for an AWS account and con igure billing settings.




- Activity 1.2: Log in to the AWS Management Console

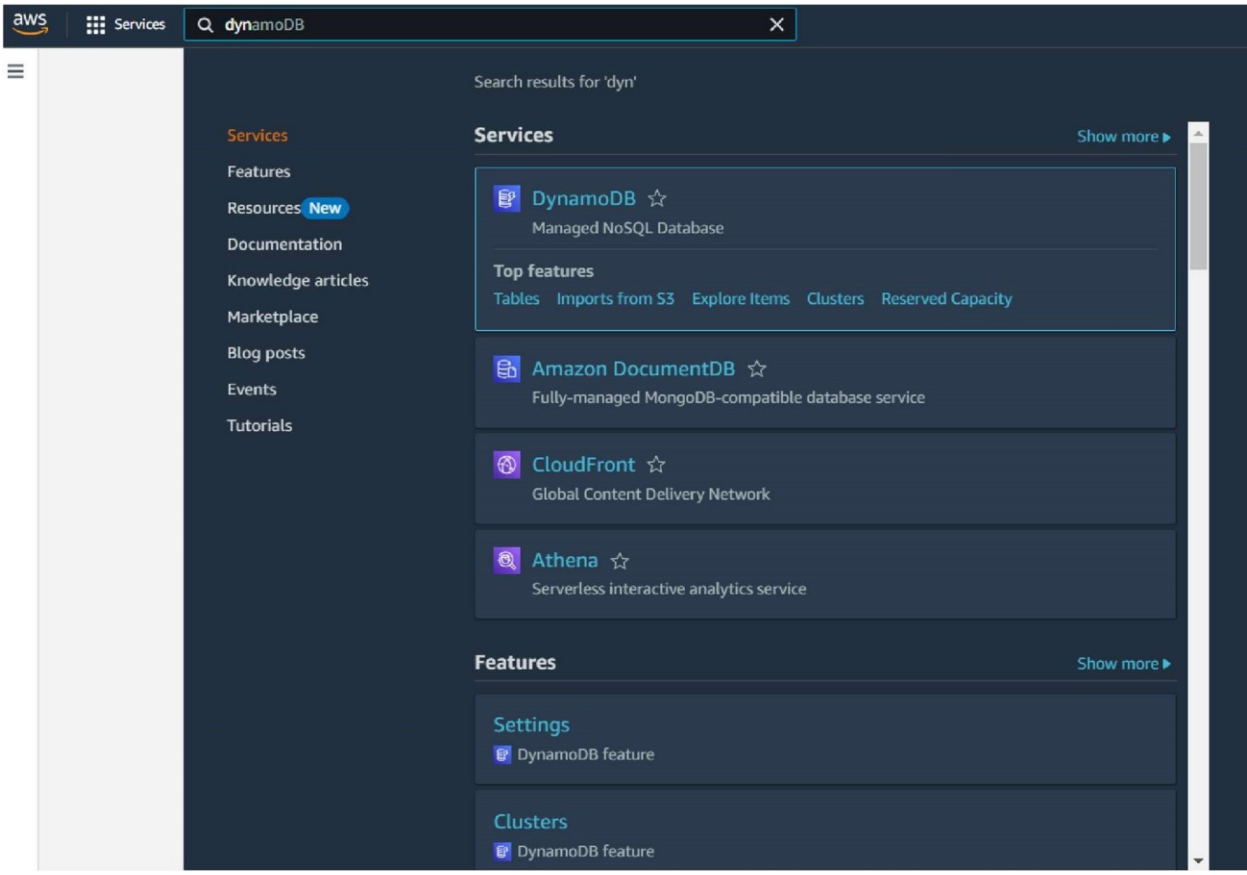
    - After setting up your account, log in to the [AWS Management Console](AWS Management Console).

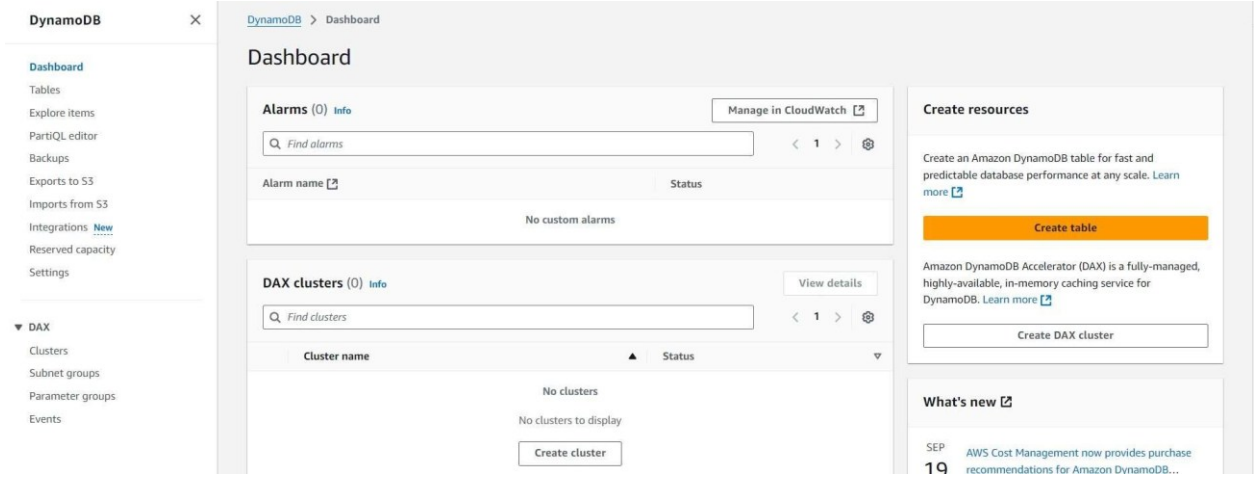


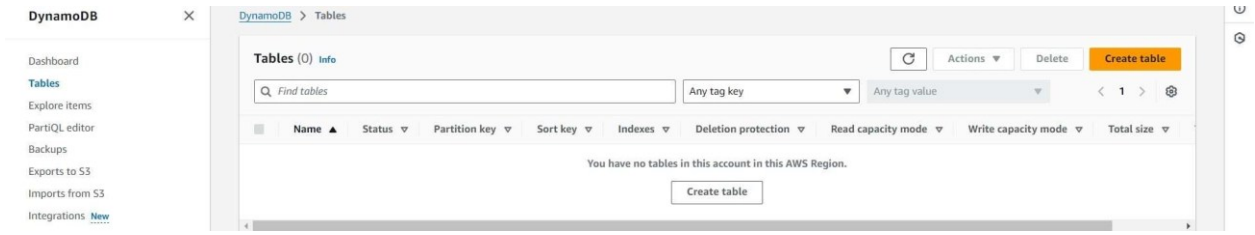
Milestone 2: DynamoDB Database Creation and Setup

- Activity 2.1:Navigate to the DynamoDB

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

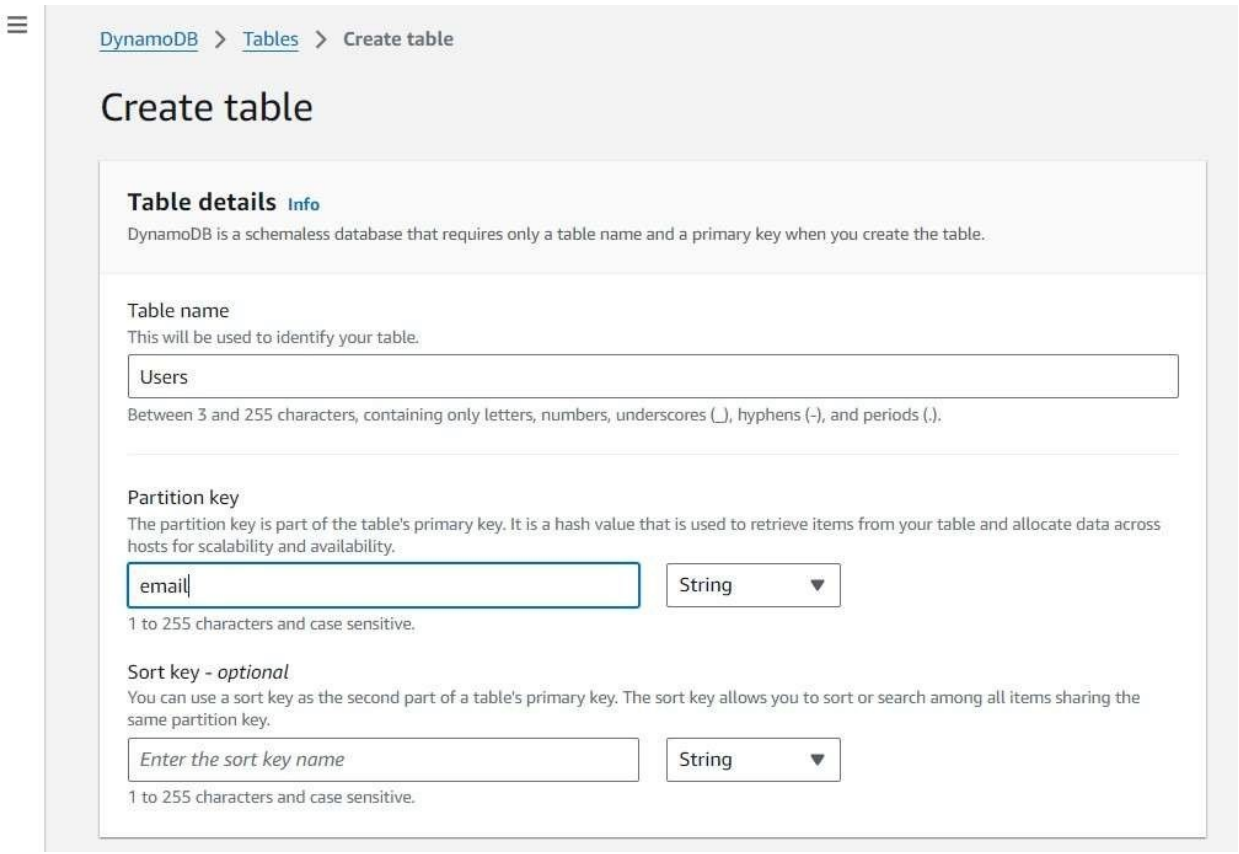





- Activity 2.2:Create a DynamoDB table for storing registration details and book requests.

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



| | | |
|---|---|---|
| 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**

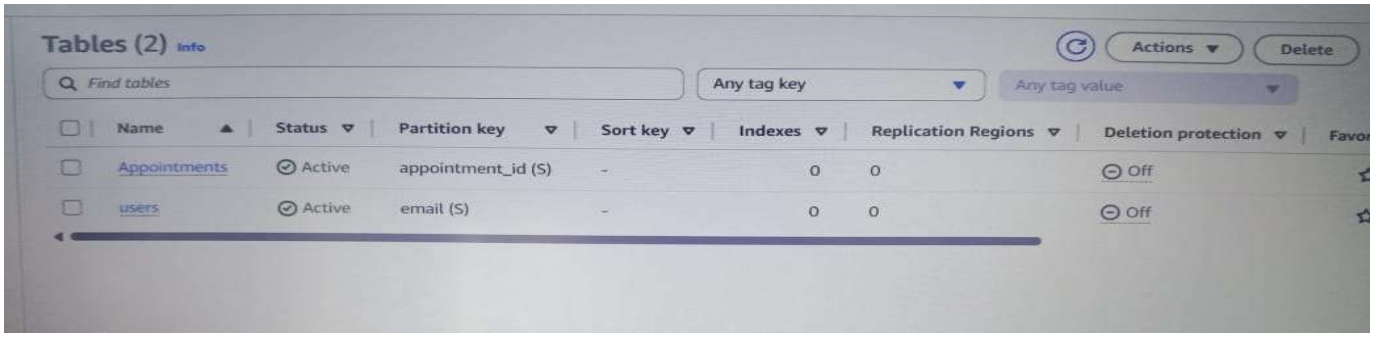
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

Add new tag

You can add 50 more tags.

Cancel Create table

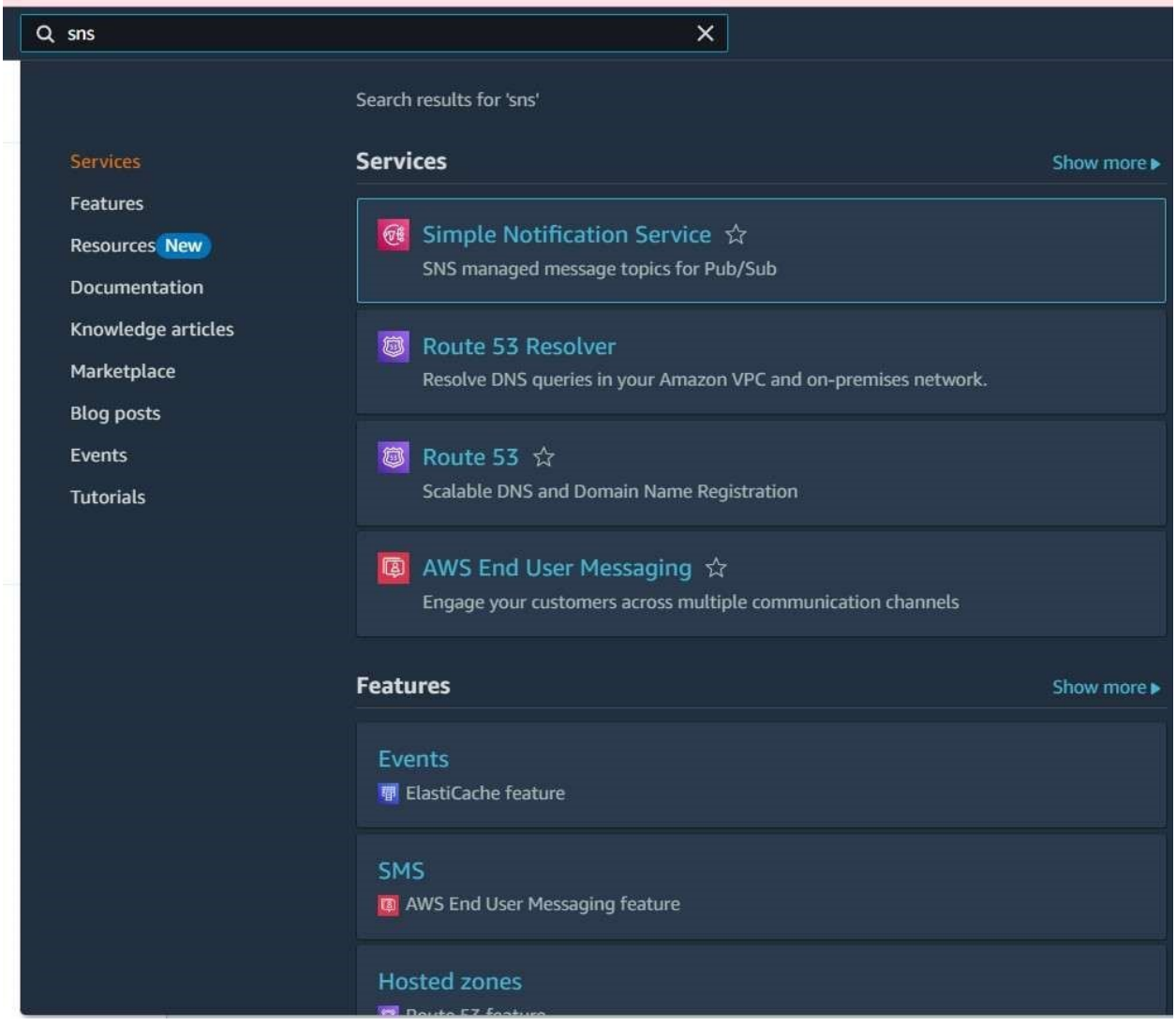


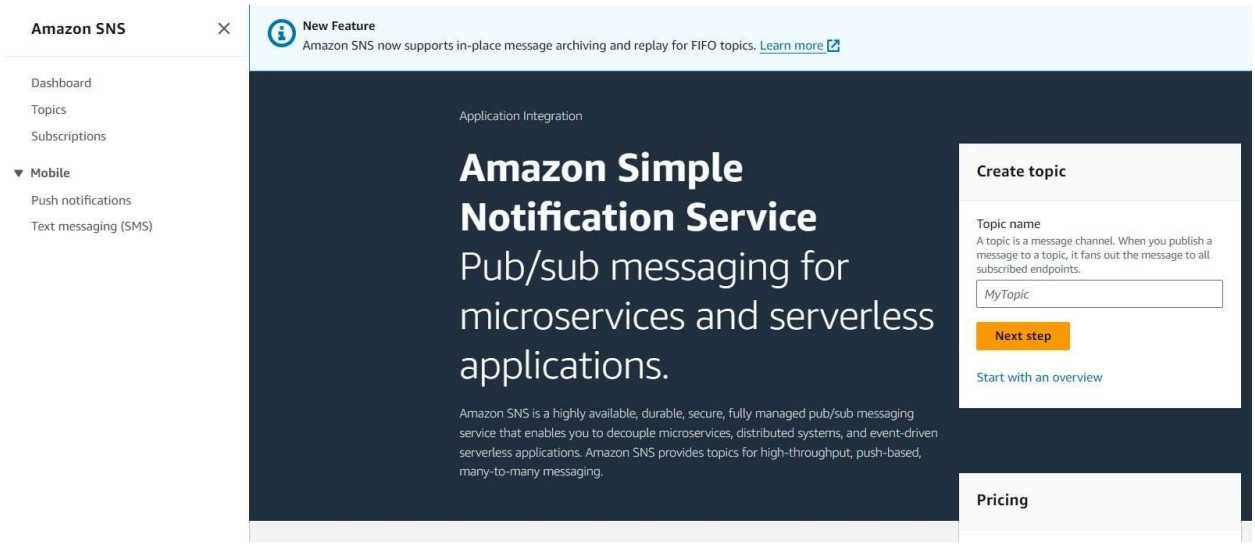

- Follow the same steps to create a requests table with Email as the primary key for book requests data.

Milestone 3: SNS Noti ication Setup

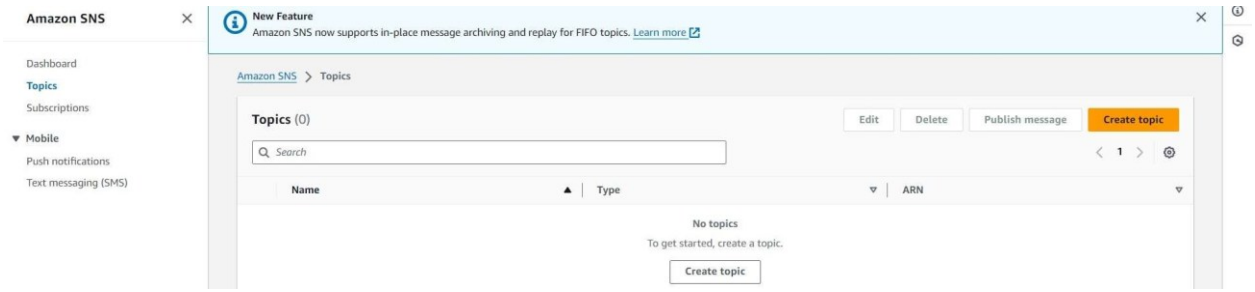
- Activity 3.1: Create SNS topics for sending email noti ications to users and library staff.

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

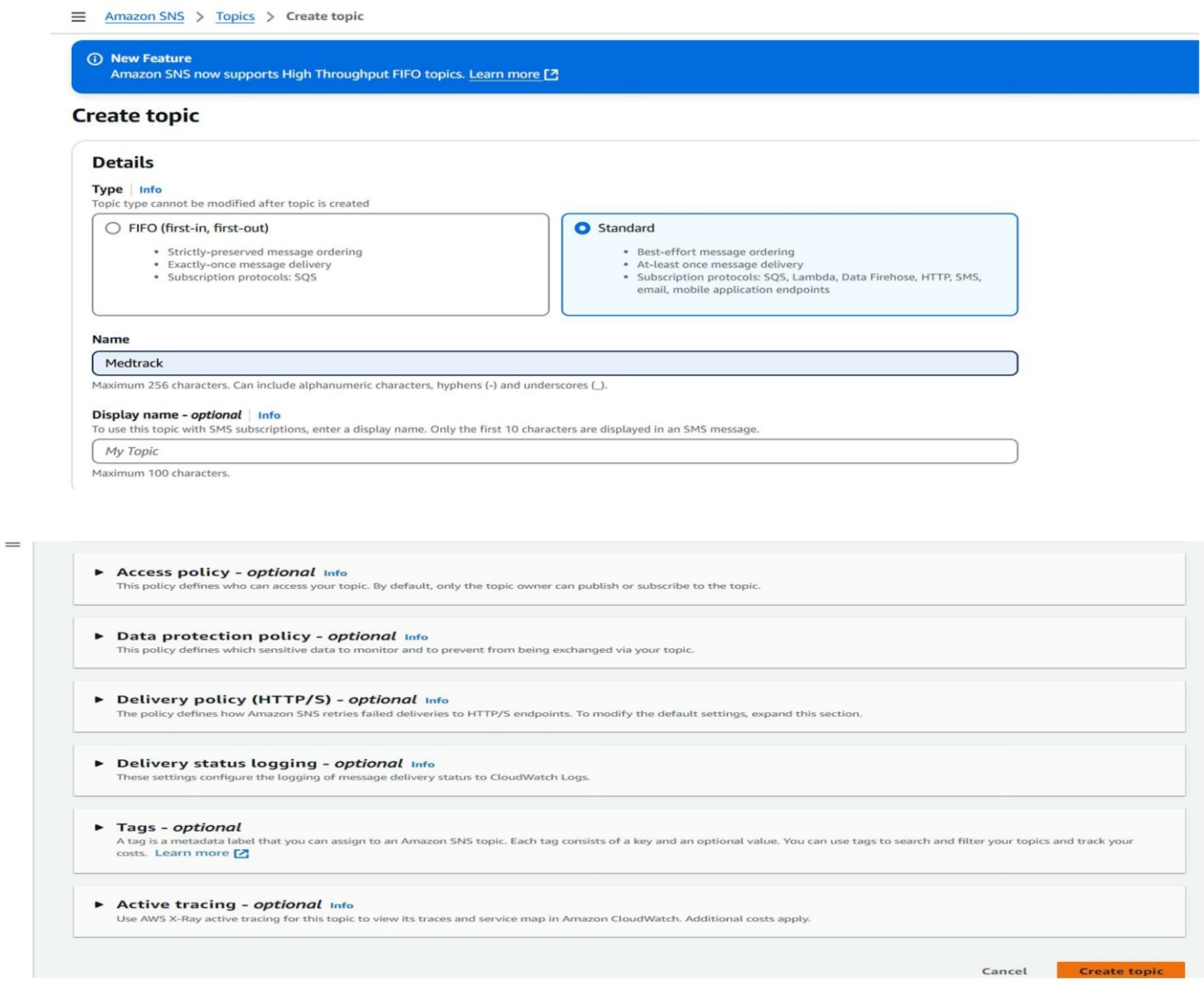




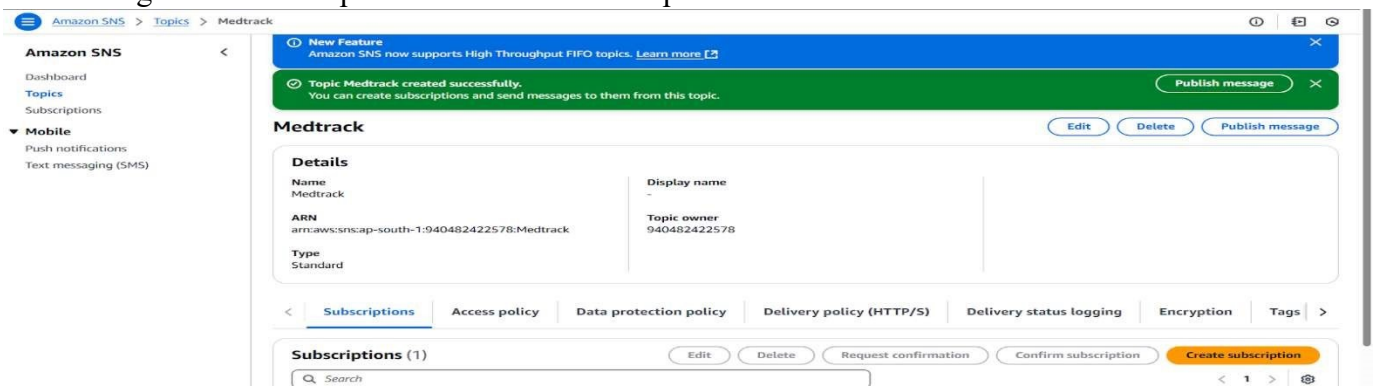

○ Click on Create Topic and choose a name for the topic.




○ Choose Standard type for general noti ication use cases and Click on Create Topic.



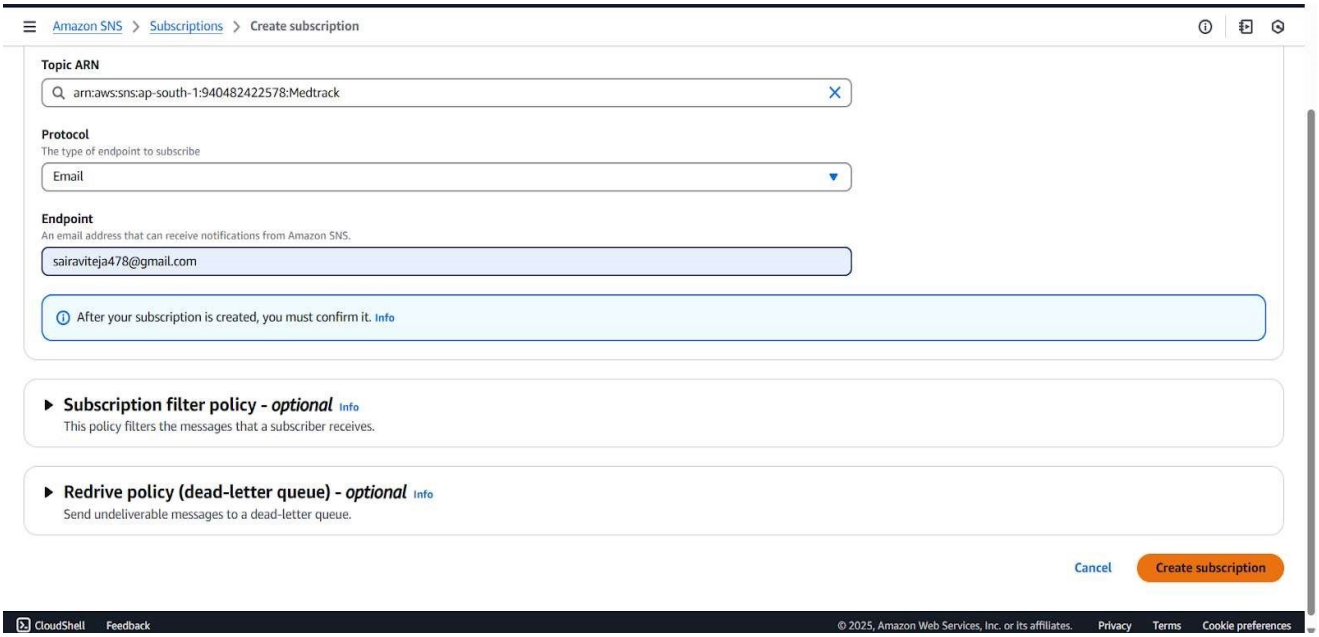

○ Con igure the SNS topic and note down the Topic ARN.



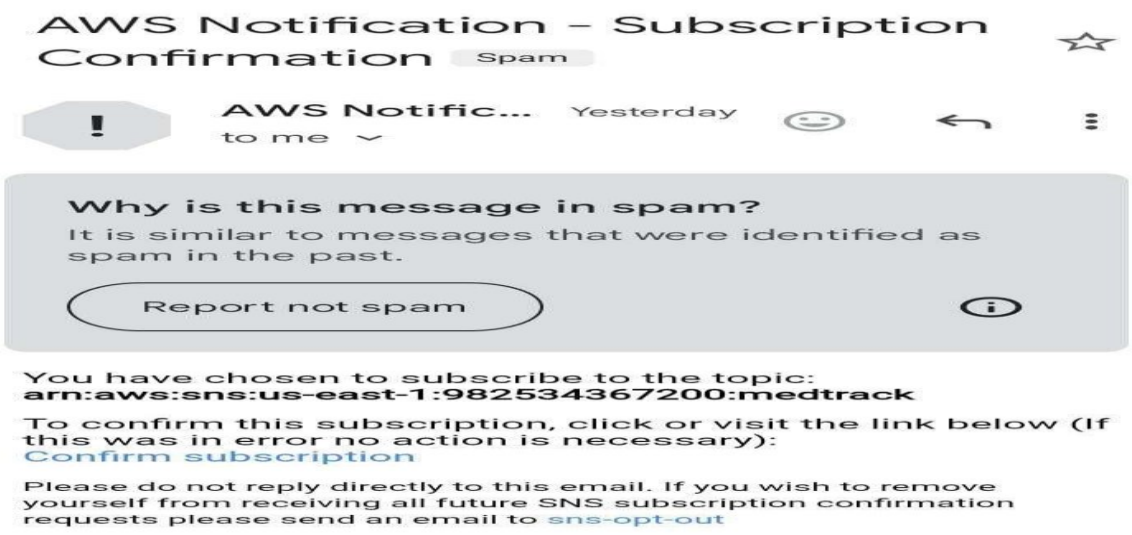

.Activity 3.2: Subscribe users and staff to relevant SNS

topics to receive real-time noti ications when a book request is made.

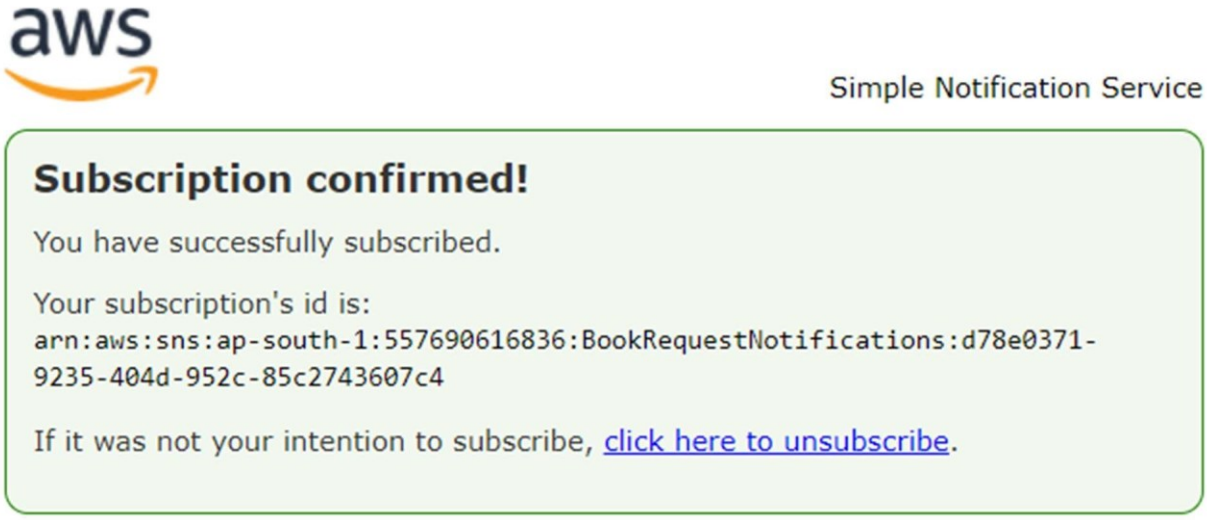
- Subscribe users (or admin staff) to this topic via Email. When a book request is made, noti ications will be sent to the subscribed emails.




○ After subscription request for the mail con irmation



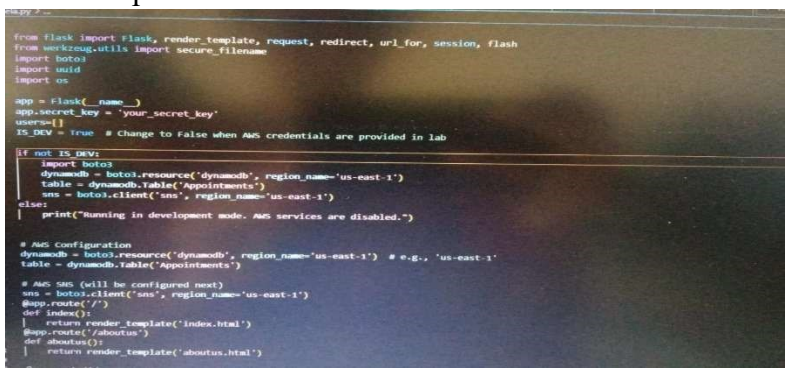
○ Navigate to the subscribed Email account and Click on the con irm subscription in the AWS Noti ication- Subscription Con irmation mail.

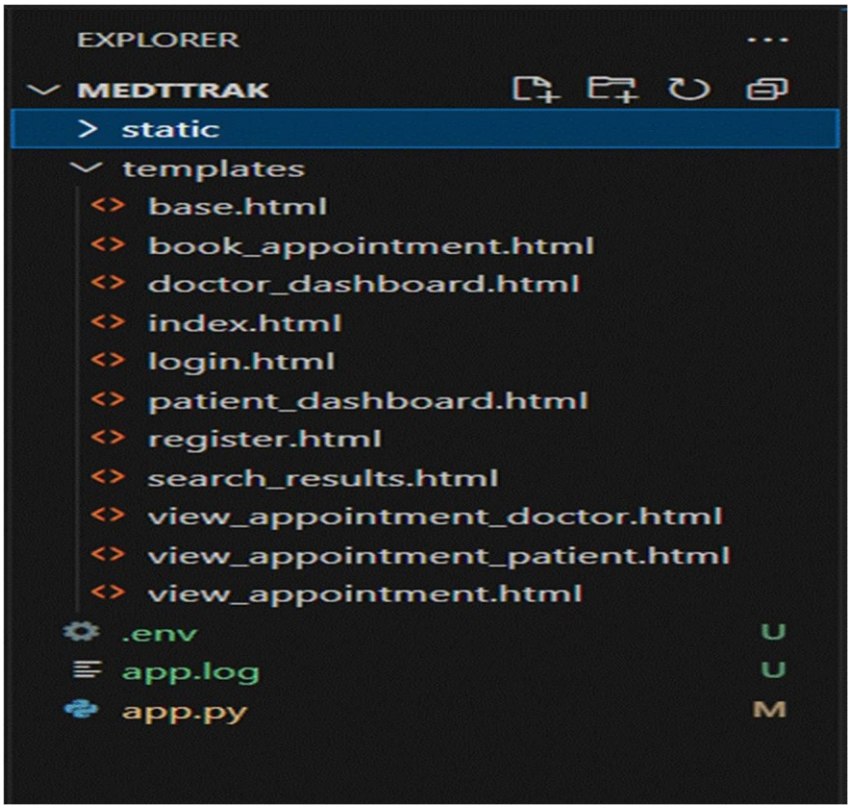



○ Successfully done with the SNS mail subscription and setup, now store the ARN link.

Milestone 4:Backend Development and Application Setup

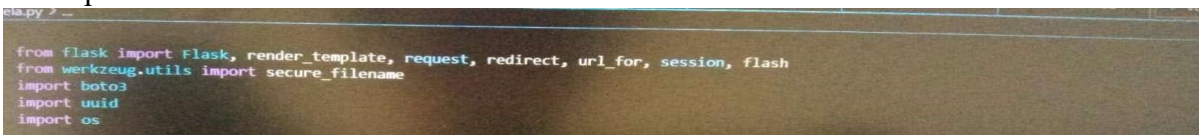
- Activity 4.1: Develop the backend using Flask
  - File Explorer Structure






Description: set up the INSTANT LIBRARY project with an app.py ile, a static/ folder for assets, and a templates/ directory containing all required HTML pages like home, login, register, subject-speci ic pages (e.g., computer_science.html, data_science.html), and utility pages (e.g., request-form.html, statistics.html).

Description of the code :



```
from flask import Flask, render_template, request, redirect, url_for, session, flash
from werkzeug.utils import secure_filename
import boto3
import uuid
import os
```

- 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 veri ication

```
app = Flask(__name__)
```

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

- Dynamodb Setup:

```
# AWS Configuration
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')  # e.g., 'us-east-1'
table = dynamodb.Table('Appointments')
```

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

```
# AWS SNS (will be configured next)
sns = boto3.client('sns', region_name='us-east-1')
```

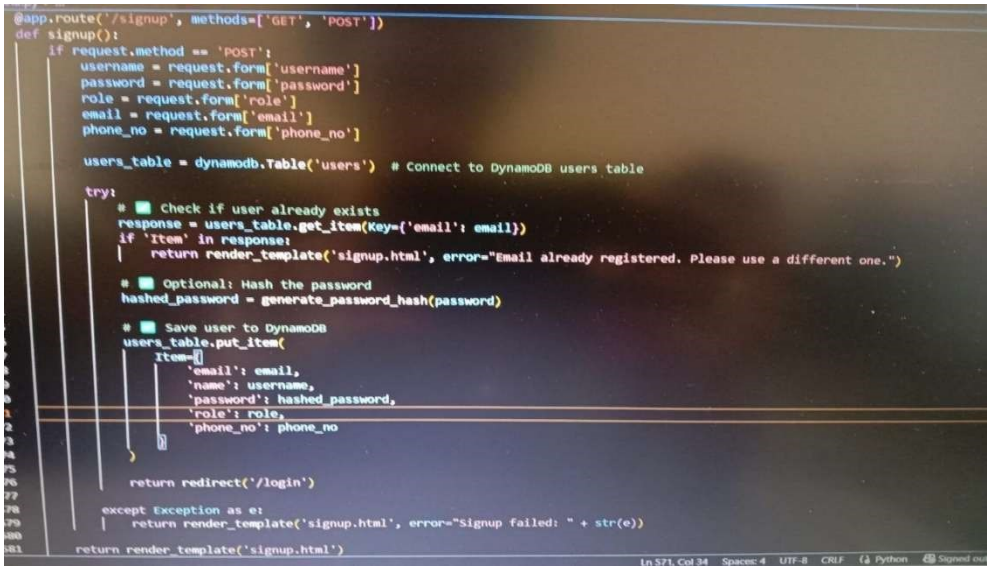
Description: Con igure SNS to send noti ications when a book request is submitted. Paste your stored ARN link in the sns_topic_arn space, along with the region name where the SNS topic is created. Also, specify the chosen email service in SMTP_SERVER (e.g., Gmail, Yahoo, etc.) and enter the subscribed 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
- index Route:

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

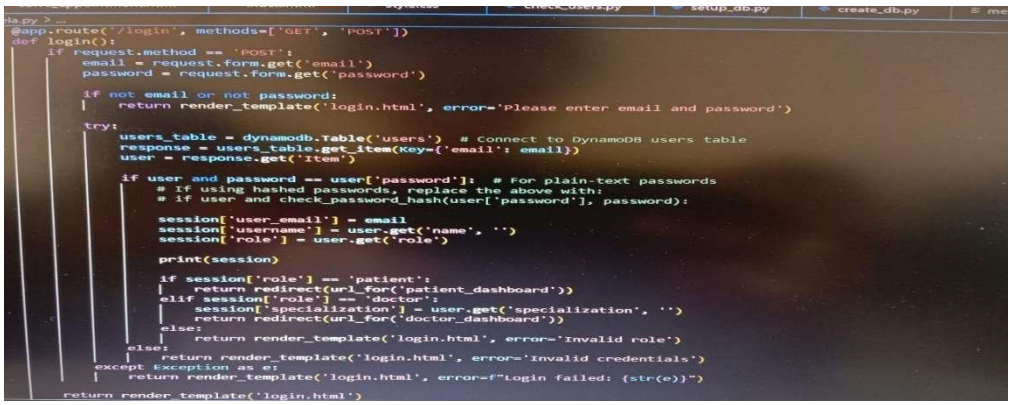
Description: de ine the index route / to automatically redirect users to the signup or login or aboutus or contactus page based on button we have chosen when they access the base URL.

- Sign up Route:




Description: de ine signup route to validate create account form ields, hash the user password using Bcrypt, store the new user in DynamoDB with a login count, and redirect to login page.

- login Route (GET/POST):




Description: de ine /login route to validate user credentials against DynamoDB, check the password using Bcrypt, update the login count on successful authentication, and redirect users to the patient dashboard or doctor dashboard based on button they have chosen.

Aboutus and contactus routes:

```python
@app.route('/aboutus')
def aboutus():
    return render_template('aboutus.html')

@app.route('/contactus')
def contactus():
    return render_template('contactus.html')
from werkzeug.security import generate_password_hash
```

Description: de ine /aboutus and contactus page to render from the index page based on buttons to handle redirection to aboutus page and contactus page and /<aboutus>.html and <contactus>.html dynamic route to render from/to speci ic pages.

- Request Routes:

```python
# Book Request Form Page
@app.route('/request-form', methods=['GET', 'POST'])
def request_form():
    if request.method == 'POST':
        # Retrieve form data from the POST request
        email = request.form['email']  # Capture email to send thank-you note
        name = request.form['name']
        year = request.form['year']
        semester = request.form['semester']
        roll_no = request.form['roll-no']
        subject = request.form['subject']
        book_name = request.form['book-name']
        description = request.form['description']

        # Store book request in DynamoDB along with the user email
        requests_table.put_item(
            Item={
                'email': email,
                'roll_no': roll_no,
                'name': name,
                'year': year,
                'semester': semester,
                'subject': subject,
                'book_name': book_name,
                'description': description
            }
        )

        # Send a thank-you email to the requesting user
        thank_you_message = f"Dear {name},\n\nThank you for submitting a book request for '{book_name}'. We will
        send_email(email, "Thank You for Your Book Request", thank_you_message)

        # Send an email to the Instant Library admin with the book request details
        admin_message = f"User {name} ({email}) has requested the book '{book_name}'.\n\nDetails:\nYear: {year}\
        send_email("instantlibrary2@gmail.com", "New Book Request", admin_message)

        return "<h3>Book request submitted successfully! We will get back to you soon.</h3>"

    # Render the request form for GET requests
    return render_template('request-form.html')
```

Description: de ine /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 con irm submission with a success message.

Logout Route:

```
@app.route('/logout')
def logout():
    session.clear()
    return redirect('/')
```

Description: de ine /exit route to render the exit.html page when the user chooses to leave or close the application.

Deployment Code:

```
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

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.

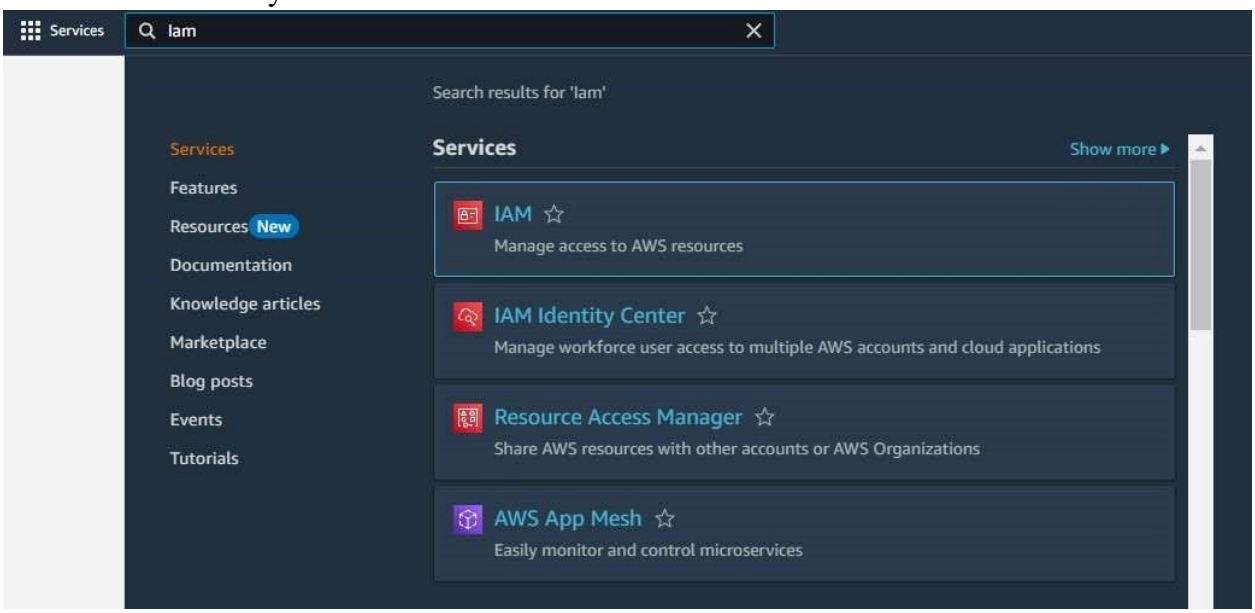
Book Appointment route:

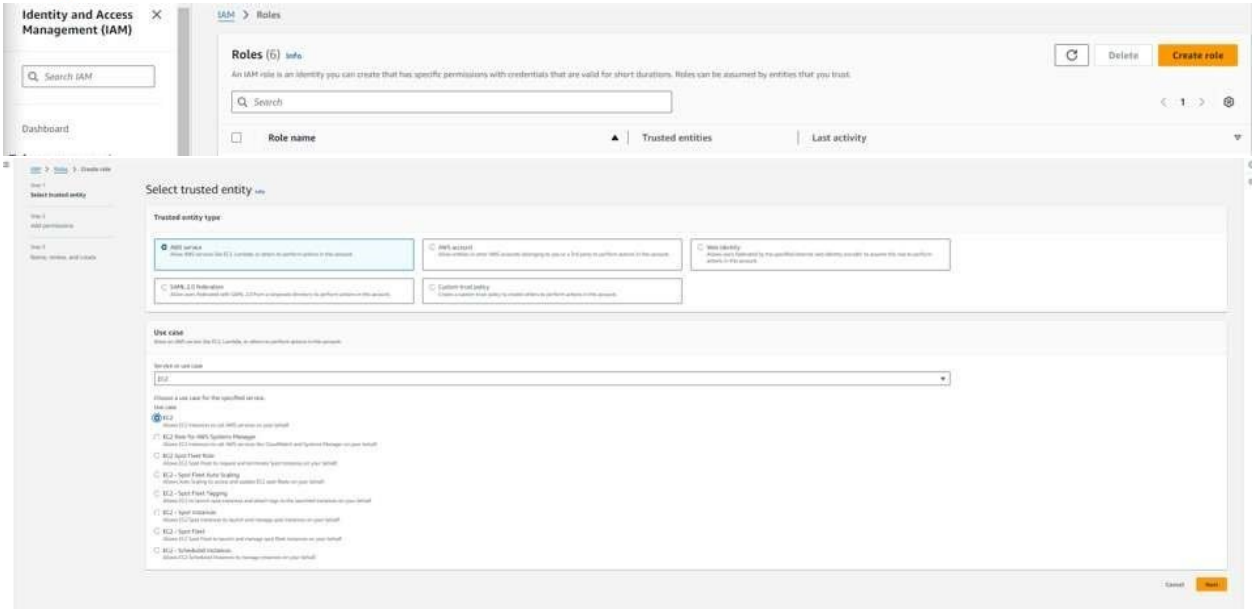
```
@app.route('/book_appointment', methods=['GET', 'POST'])
def appointment():
    if request.method == 'POST':
        # Extract form data
        name = request.form['name']
        email = request.form['email']
        date = request.form['date']
        time = request.form['time']
        age = request.form['age']
        gender = request.form['gender']
        problem = request.form['problem']
        specialization = request.form['specialization']
        filename = None

        # Handle file upload
        file = request.files.get('report')
        if file and file.filename != '':
            filename = secure_filename(file.filename)
            file.save(os.path.join("static/uploads", filename))

        appointment_id = str(uuid.uuid4())
```

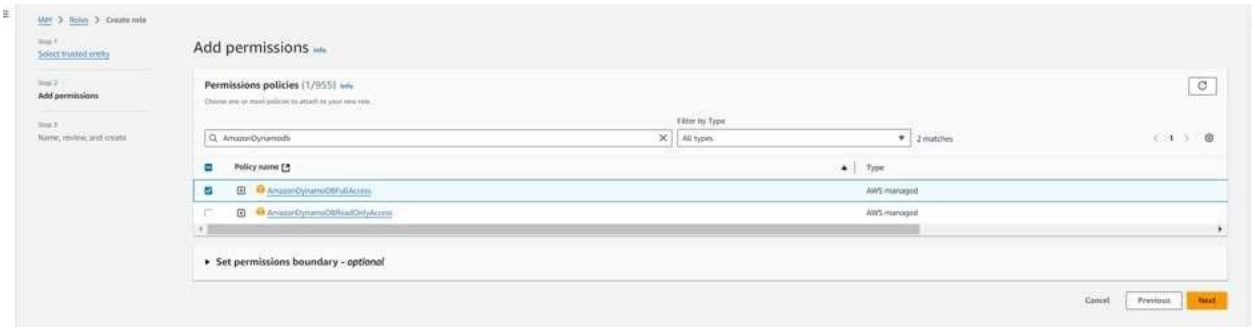
Milestone 5: IAM Role Setup

- 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.



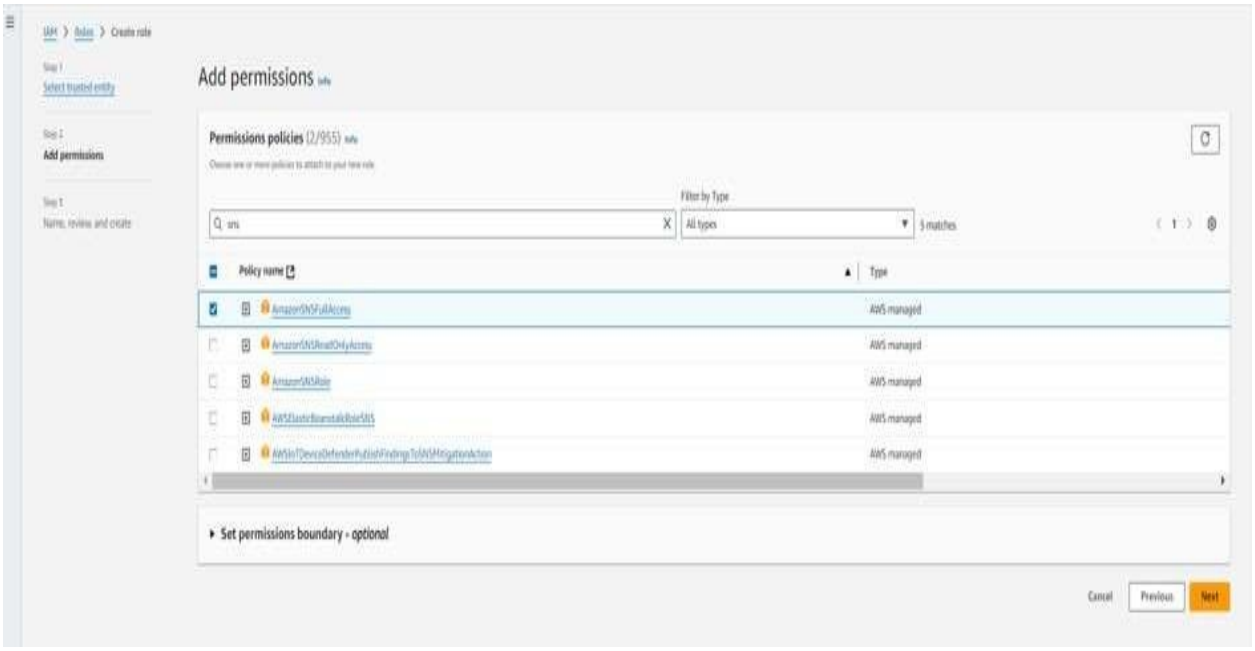


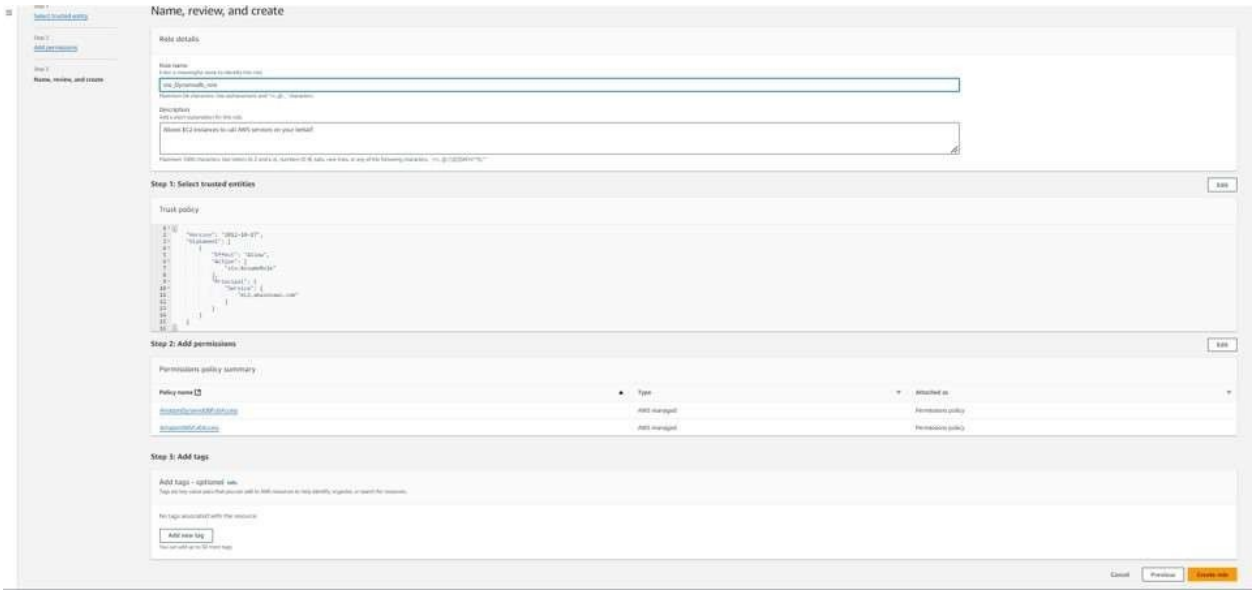



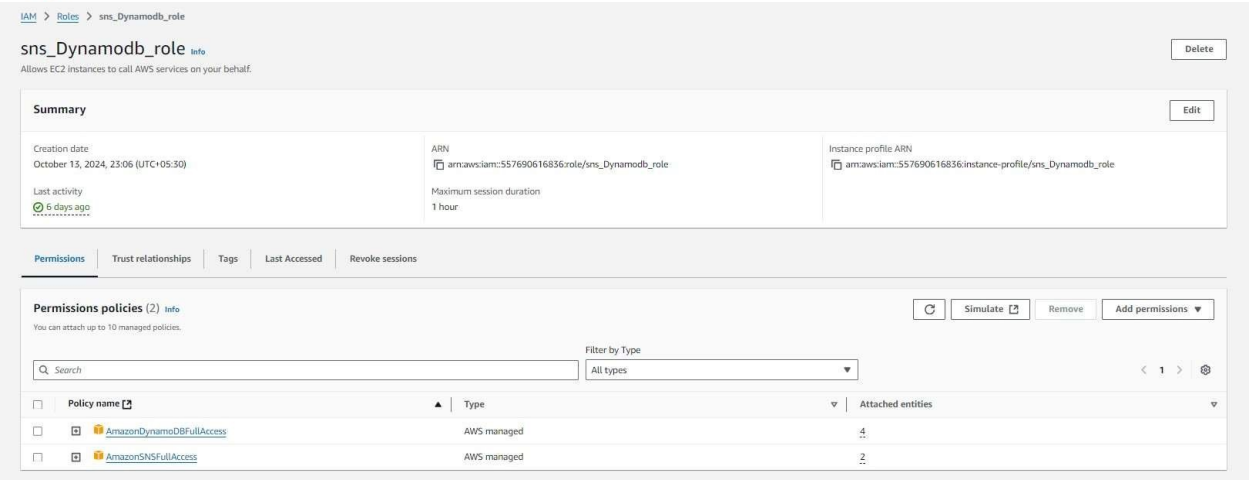

- Activity 5.2: Attach Policies.

  Attach the following policies to the role:

- AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.
- AmazonSNSFullAccess: Grants EC2 the ability to send noti ications via SNS.

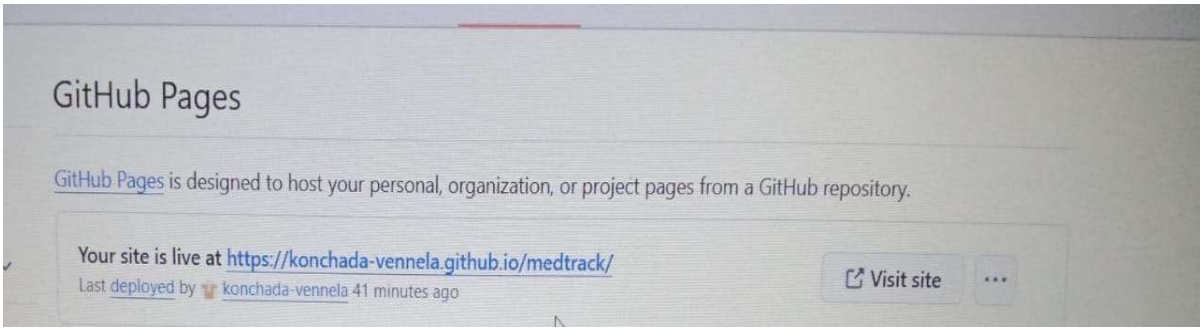

IAM > Roles > Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
Add permissions info
Permissions policies (2/955) info
Choose one or more policies to attach to your new role.
Filter by Type
sns
All types
5 matches
Policy name
Type
AmazonSNSFullAccess
AWS managed
AmazonSNSReadOnlyAccess
AWS managed
AmazonSNSRole
AWS managed
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
Policy name
Type
Attached as
Step 3: Add tags
Add tags - optional
Add new tag




Milestone 6: EC2 Instance Setup

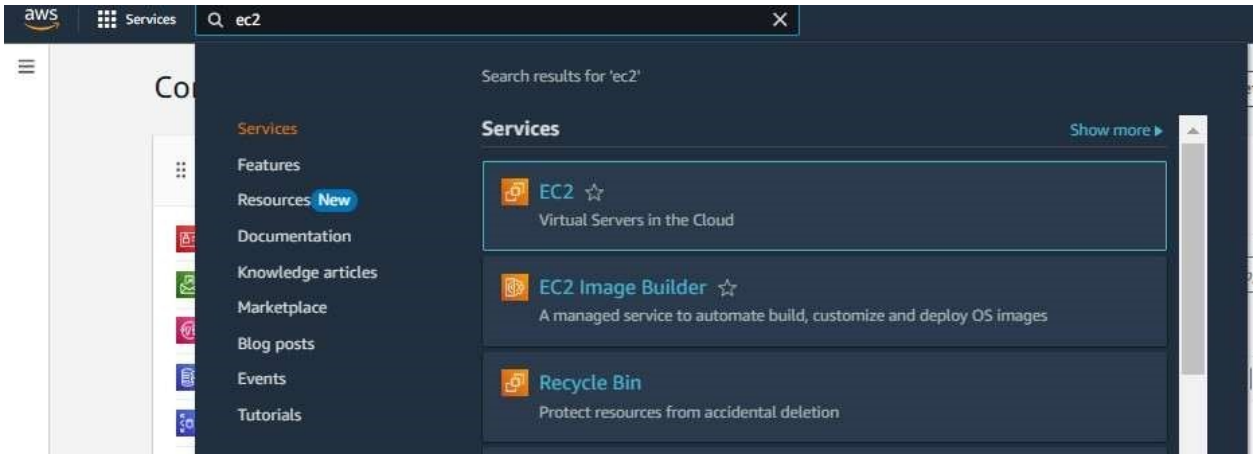
- Note: Load your Flask app and Html iles into GitHub repository.

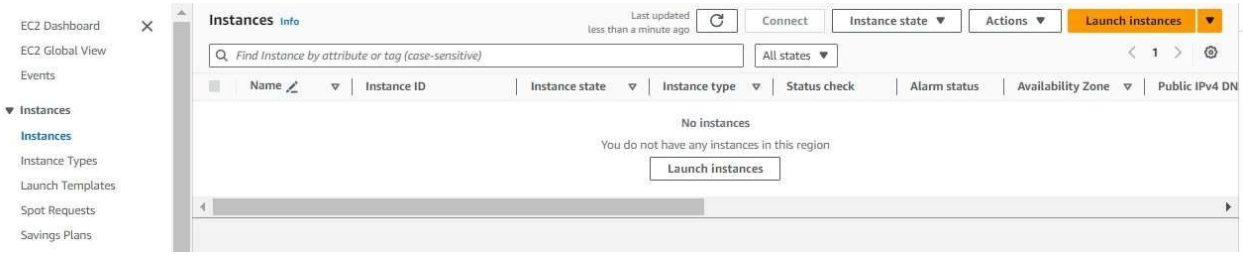

- Activity 6.1: Launch an EC2 instance to host the Flask application.

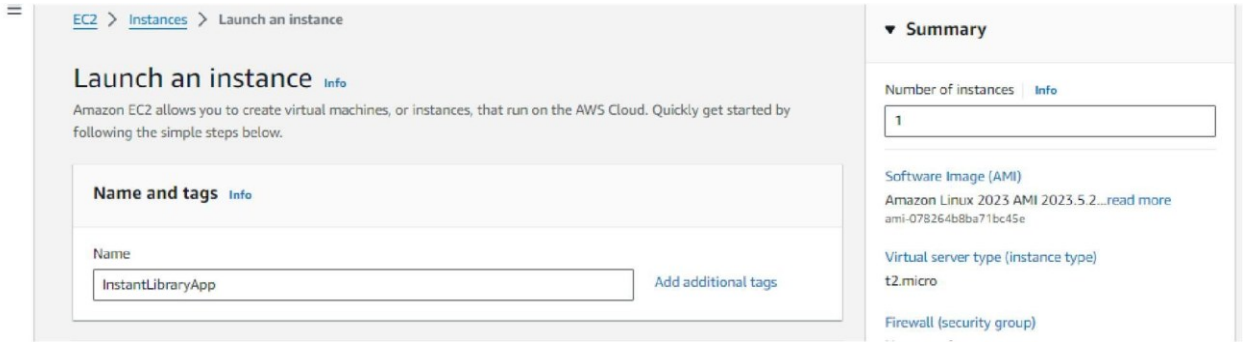
- Launch EC2 Instance

○ In the AWS Console, navigate to EC2 and launch a new instance.

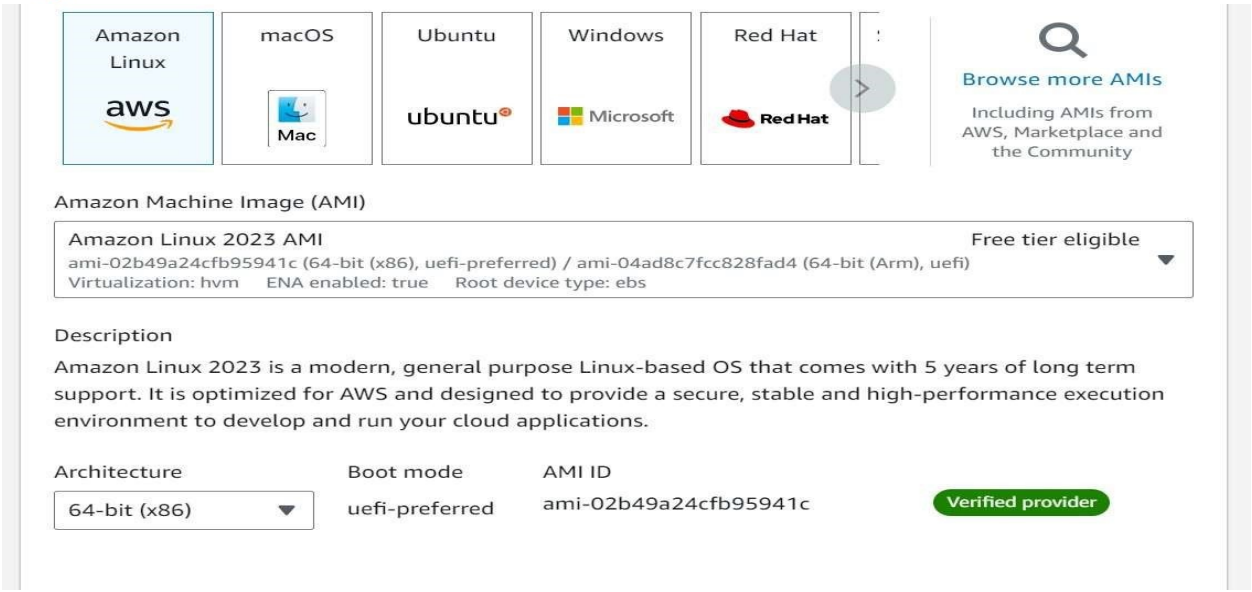



- Click on Launch instance to launch EC2 instance

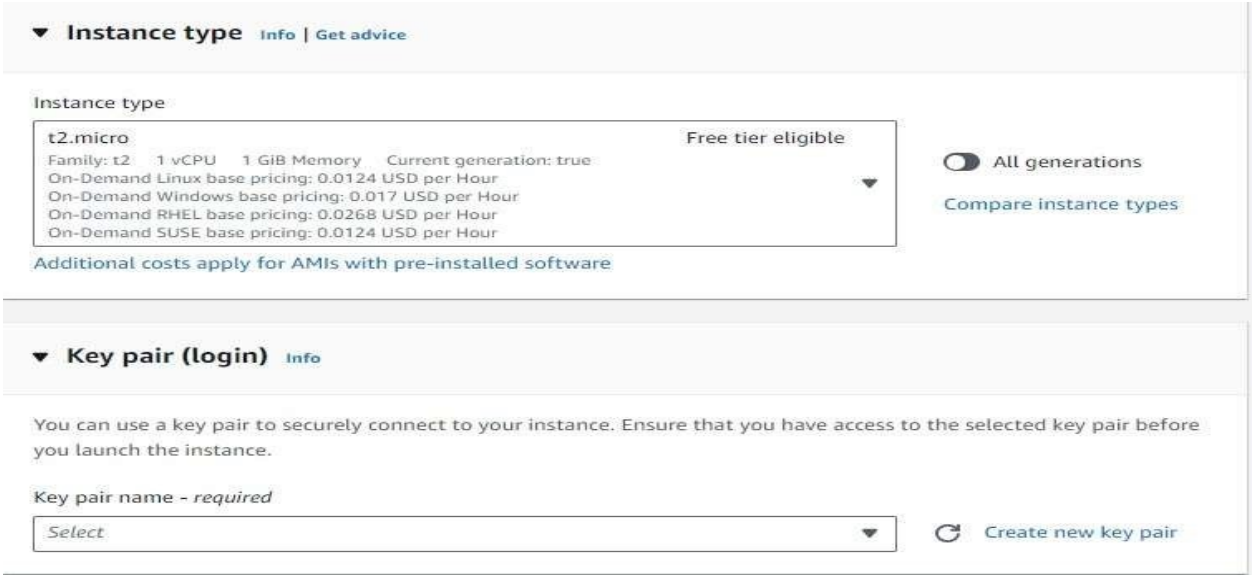


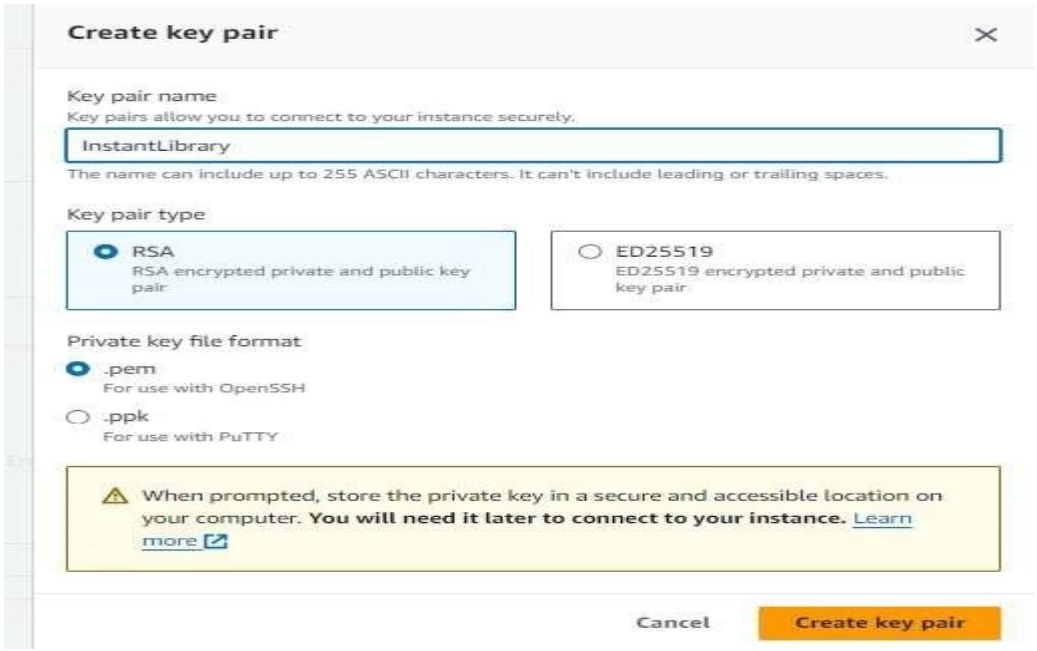



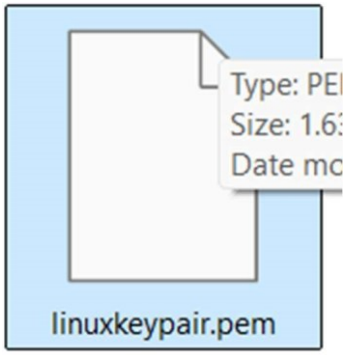

- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).

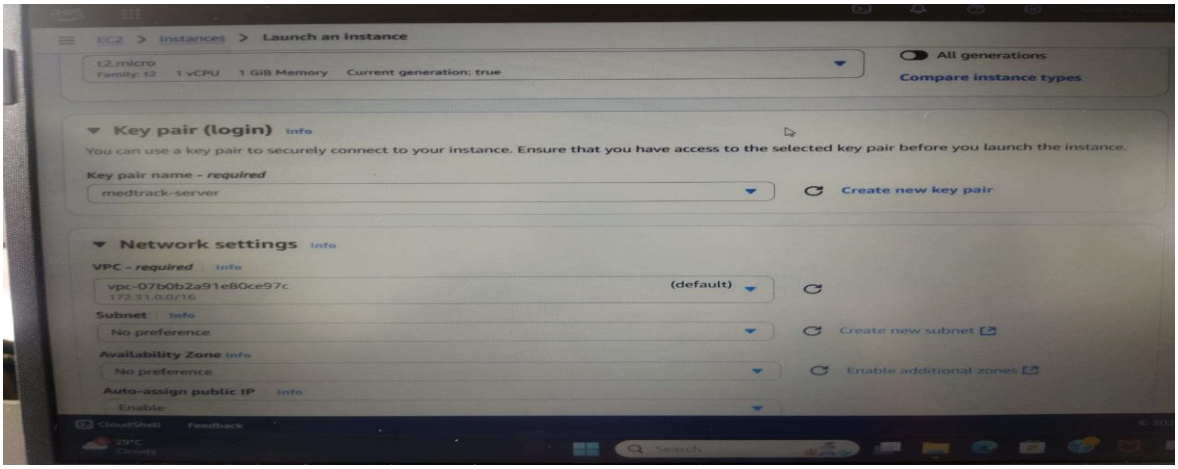



- Create and download the key pair for Server access.

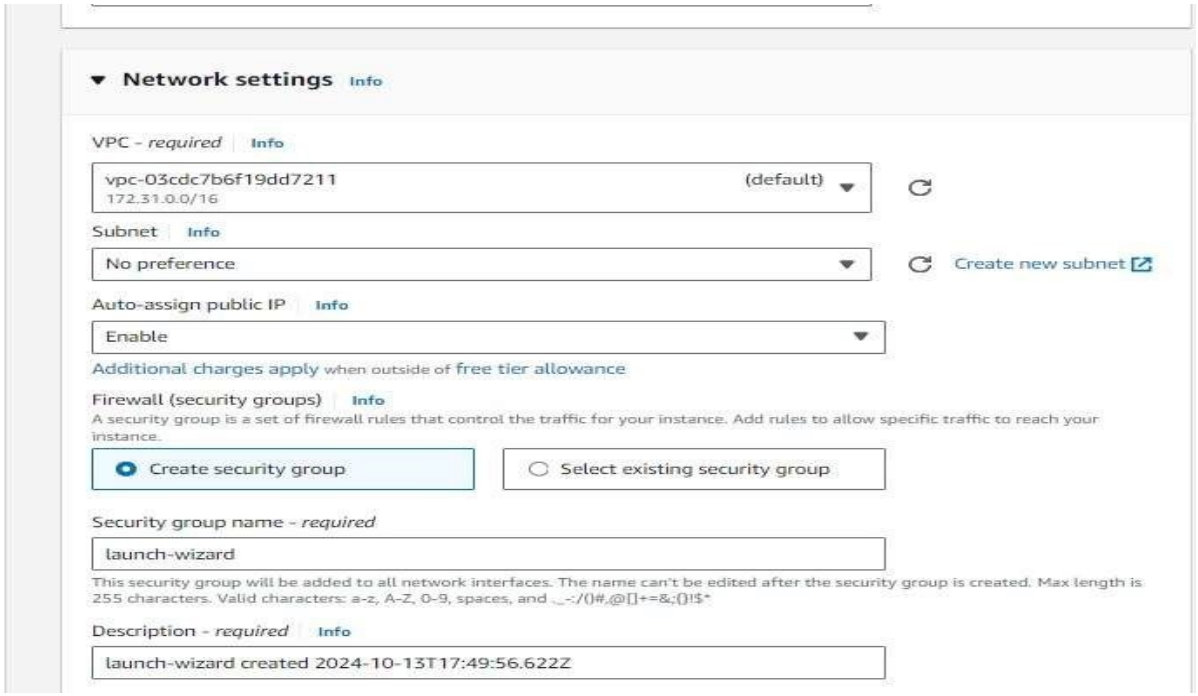


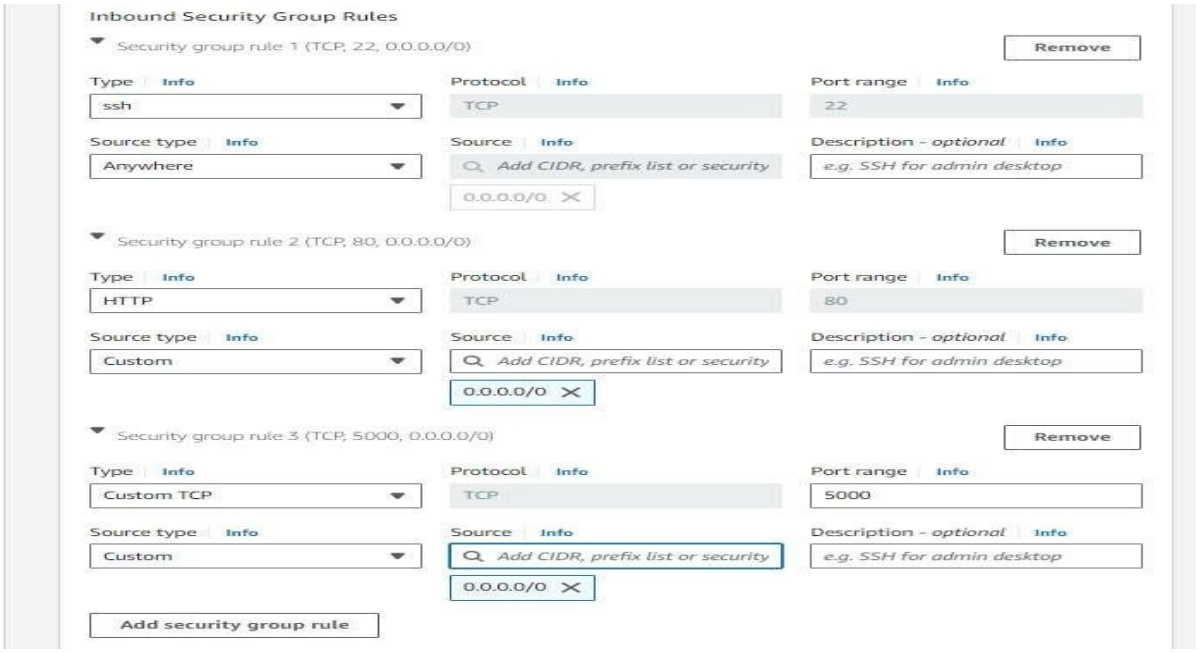

Create key pair
Key pair name
Key pairs allow you to connect to your instance securely.
InstantLibrary
The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.
Key pair type
RSA
RSA encrypted private and public key pair
ED25519
ED25519 encrypted private and public key pair
Private key file format
.pem
For use with OpenSSH
.ppk
For use with PuTTY
When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance. Learn more
Cancel
Create key pair

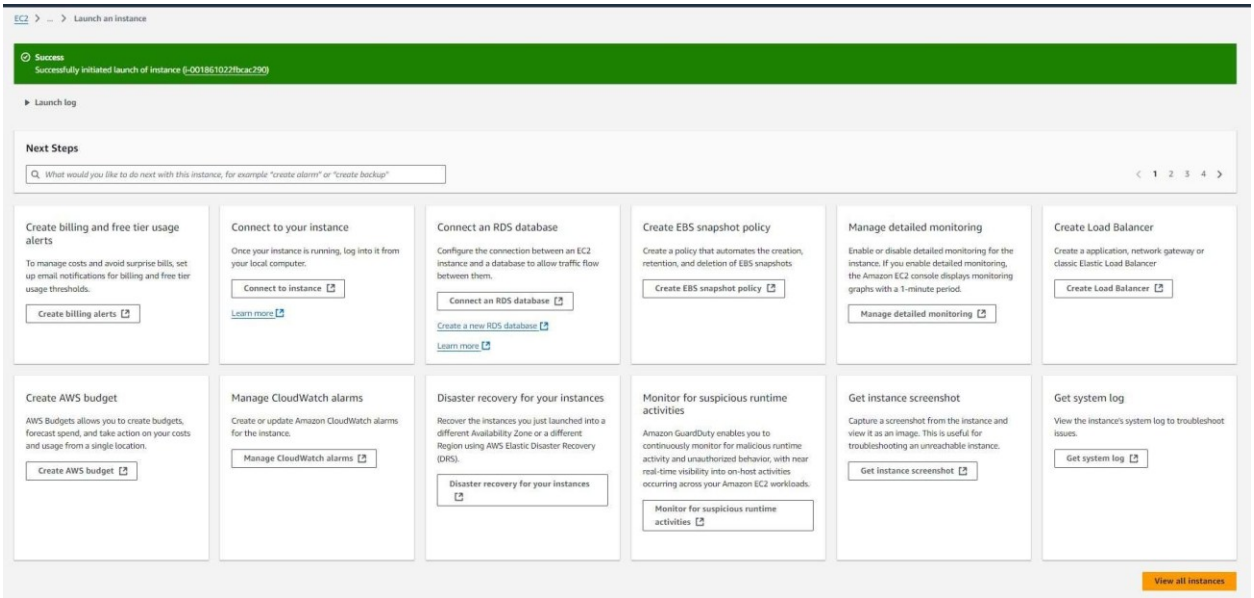

Type: PE
Size: 1.6
Date mc
linuxkeypair.pem


EC2 > Instances > Launch an instance
t2.micro
Family: t2 1 vCPU 1 GiB Memory Current generation: true
All generations
Compare instance types
Key pair (login) Info
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.
Key pair name - required
medtrack-server
Create new key pair
Network settings Info
VPC - required Info
vpc-07b0b2a91e80ce97c
172.31.0.0/16
(default)
Subnet Info
No preference
Create new subnet
Availability Zone Info
No preference
Enable additional zones
Auto-assign public IP Info
Enable

- Activity 6.2:Con igure security groups for HTTP, and SSH access.

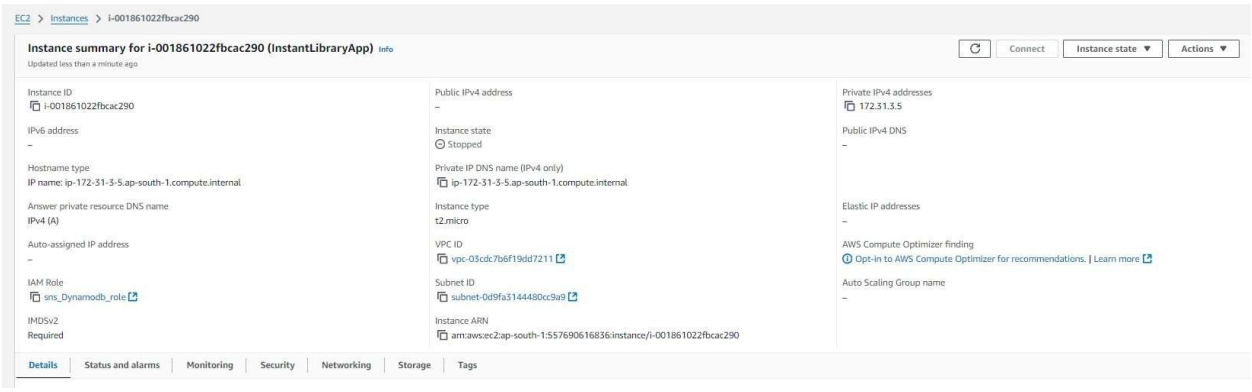


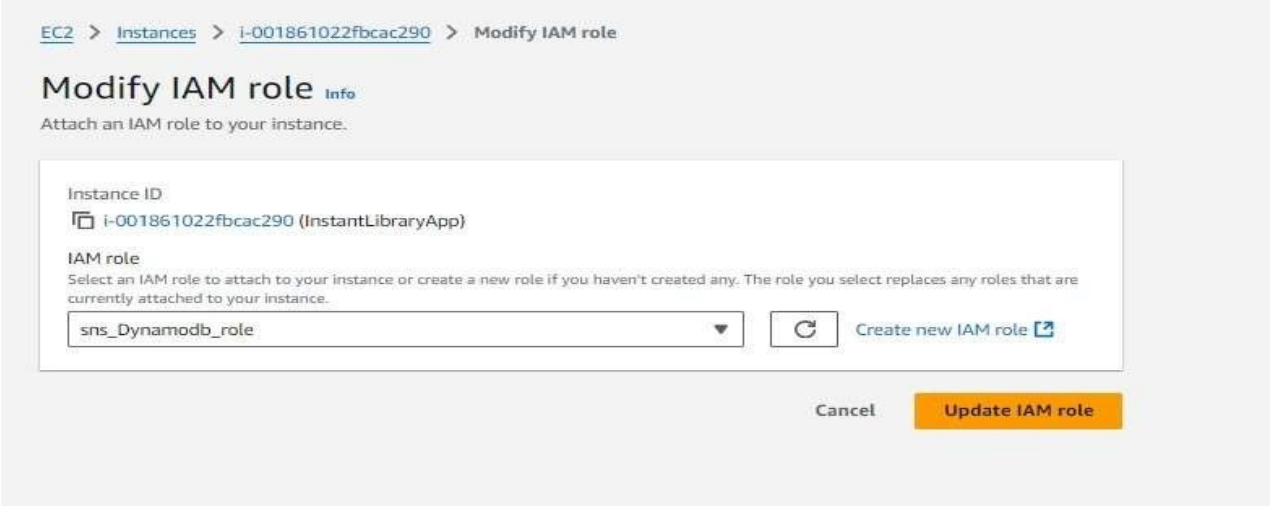



- To connect to EC2 using EC2 Instance Connect, start by ensuring that an IAM role is attached to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and selecting Modify IAM Role to attach the appropriate role. After the IAM role is connected, navigate to the EC2 section in the AWS Management Console. Select the EC2 instance you wish to connect to. At the top of the

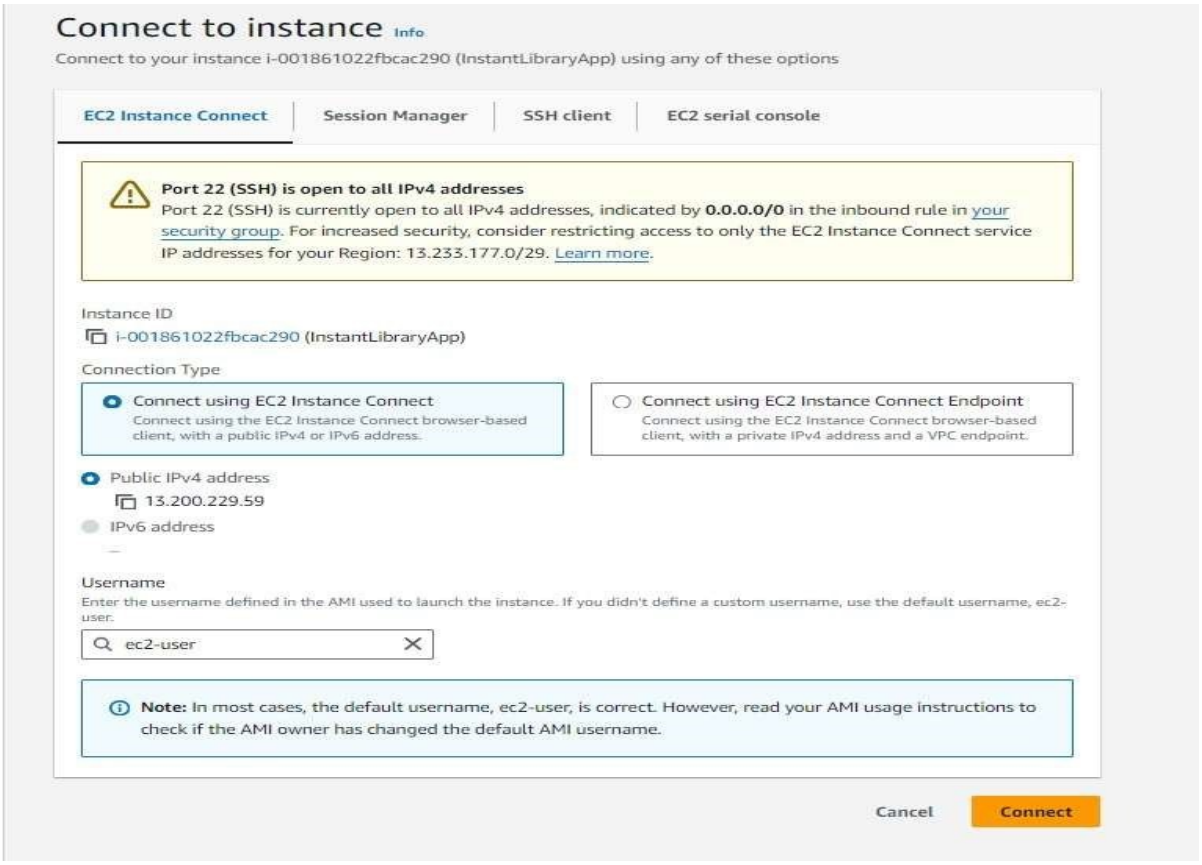
  EC2 Dashboard, click the Connect button. From the connection methods presented, choose EC2 Instance Connect. Finally, click Connect again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.

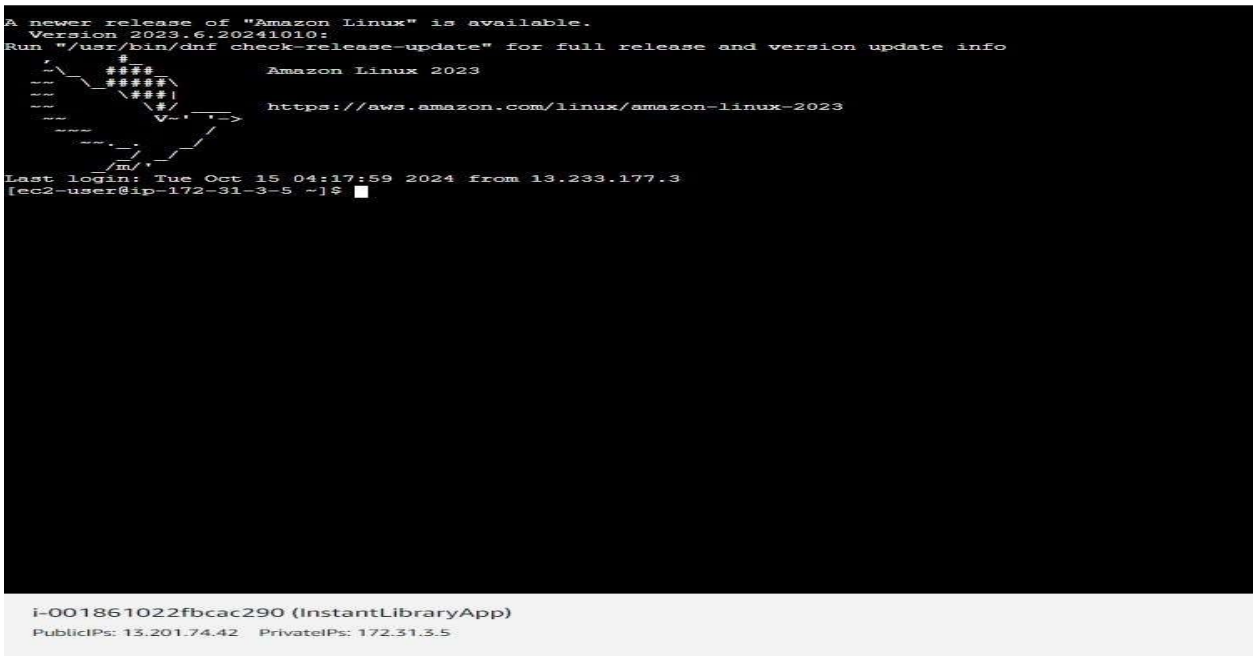







- Now connect the EC2 with the iles




Milestone 7: Deployment on EC2

Activity 7.1: Install Software on the EC2 Instance

Install Python3, Flask, and Git:

On Amazon Linux 2:

```
sudo yu update-y
sudo yum install python3 git
sudo pip3 install lask boto3
```

Verify Installations:

```
lask --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: ‘https://github.com/hemasundar2005/medtrack’ Note: change your-

githubusername and your-repository-name with your credentials here: ‘git clone

‘https://github.com/hemasundar2005/medtrack ‘

- 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, con igure and run the Flask application by executing the following command with elevated privileges:

Run the Flask Application

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

```
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
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Tue Oct 15 04:17:59 2024 from 13.233.177.3
[ec2-user@ip-172-31-3-5 ~]$ git clone https://github.com/AlekhyaPenubakula/InstantLibrary.git
fatal: destination path 'InstantLibrary' already exists and is not an empty directory.
[ec2-user@ip-172-31-3-5 ~]$ cd InstantLibrary
[ec2-user@ip-172-31-3-5 InstantLibrary]$ cd InstantLibrary
[ec2-user@ip-172-31-3-5 InstantLibrary]$ flask run --host=0.0.0.0 --port=80
 * Debug mode: off
Permission denied
[ec2-user@ip-172-31-3-5 InstantLibrary]$ ^C
[ec2-user@ip-172-31-3-5 InstantLibrary]$ ^C
[ec2-user@ip-172-31-3-5 InstantLibrary]$ sudo flask run --host=0.0.0.0 --port=80
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:80
 * Running on http://172.31.3.5:80
Press CTRL+C to quit
^C[ec2-user@ip-172-31-3-5 InstantLibrary]$
[ec2-user@ip-172-31-3-5 InstantLibrary]$ sudo flask run --host=0.0.0.0 --port=80
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:80
 * Running on http://172.31.3.5:80
Press CTRL+C to quit
183.82.125.56 - - [22/Oct/2024 07:42:00] "GET / HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /register HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /static/images/library3.jpg HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /favicon.ico HTTP/1.1" 404 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /static/images/library3.jpg HTTP/1.1" 304 -
183.82.125.56 - - [22/Oct/2024 07:42:21] "POST /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:24] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:27] "POST /login HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:28] "GET /home-page HTTP/1.1" 200 -
```

i-001861022fbcac290 (InstantLibraryApp)

PublicIPs: 13.201.74.42 PrivateIPs: 172.31.3.5

Verify the Flask app is running: http://your-ec2-public-ip

- Run the Flask app on the EC2 instance

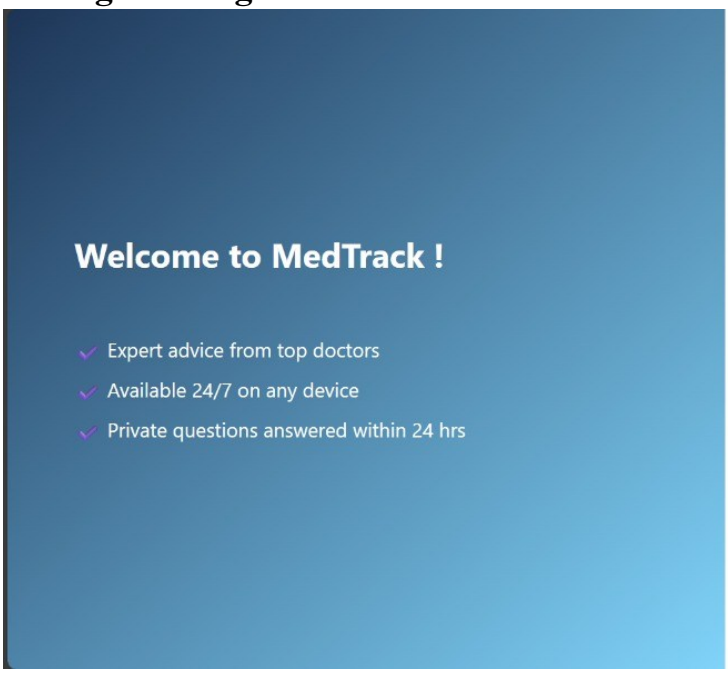
```
[ec2-user@ip-172-31-3-5 InstantLibrary]$ sudo flask run --host=0.0.0.0 --port=80
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:80
 * Running on http://172.31.3.5:80
Press CTRL+C to quit
183.82.125.56 - - [22/Oct/2024 07:42:00] "GET / HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /register HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /static/images/library3.jpg HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /favicon.ico HTTP/1.1" 404 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /static/images/library3.jpg HTTP/1.1" 304 -
183.82.125.56 - - [22/Oct/2024 07:42:21] "POST /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:24] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:27] "POST /login HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:28] "GET /home-page HTTP/1.1" 200 -
```

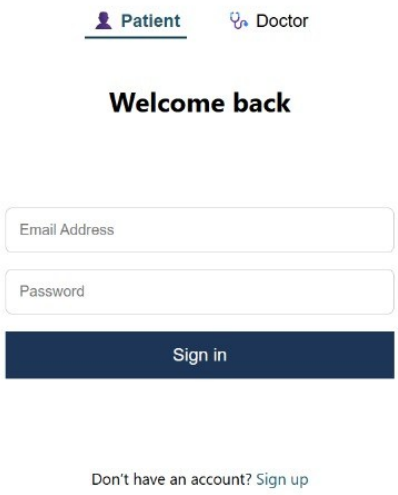
Access the website through:
PublicIPs: https:// 54.81.119.53 /

Milestone 8: Testing and Deployment

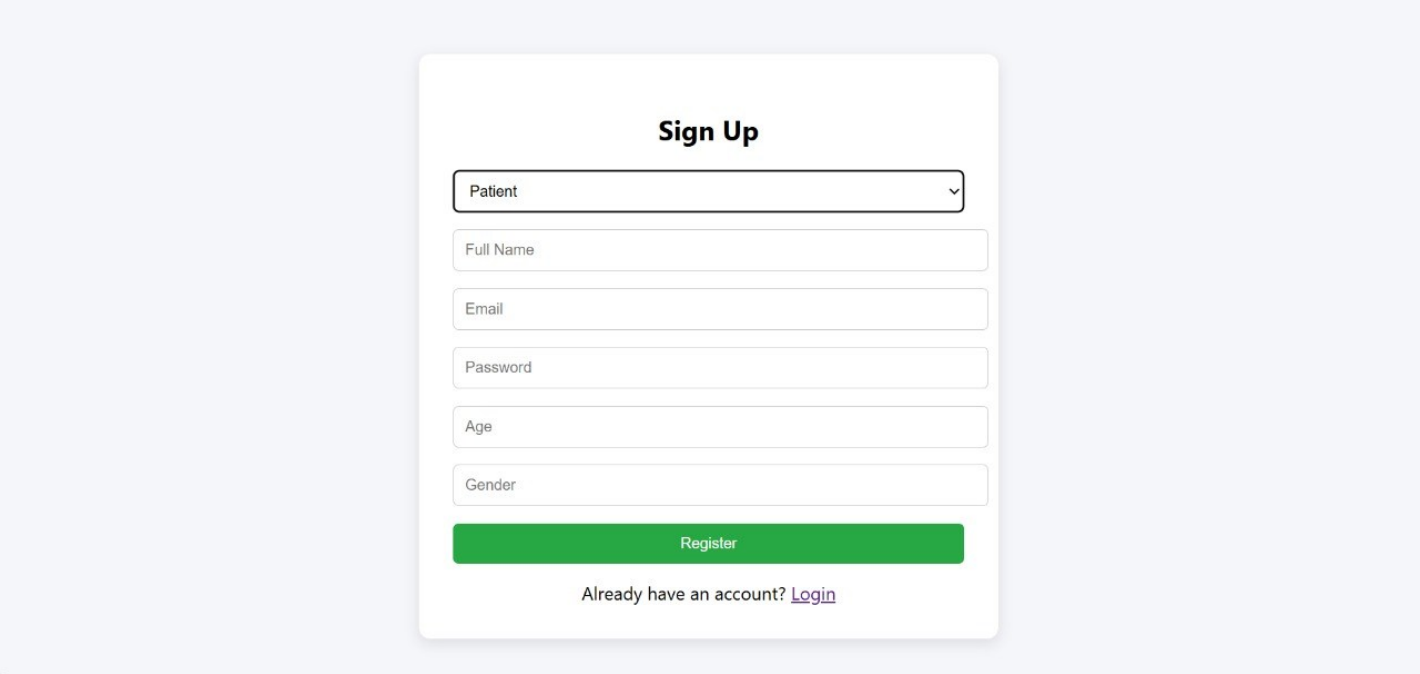
- Activity 8.1: Conduct functional testing to verify user registration, login, book requests, and notifications.

**Register Page:**

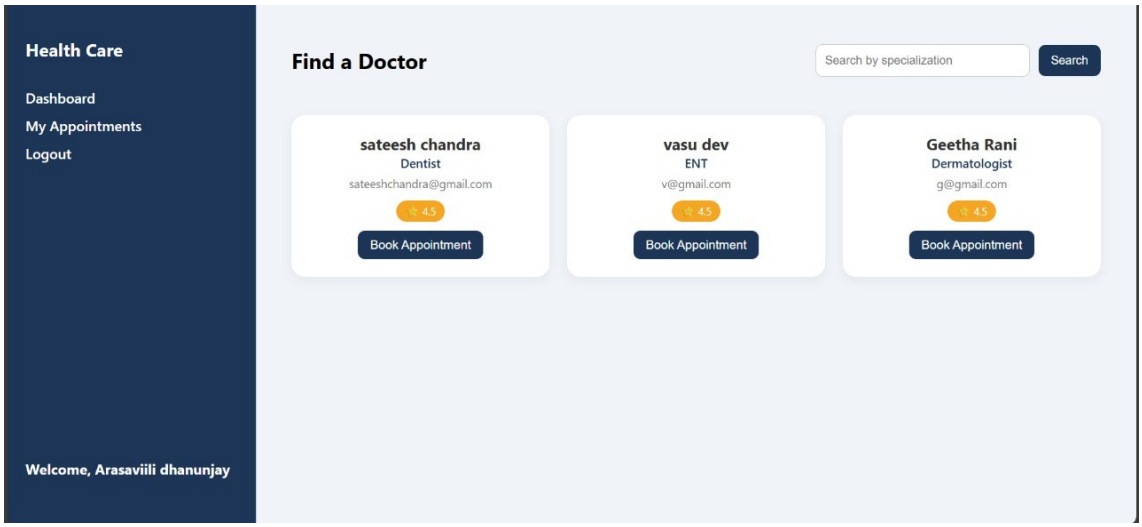


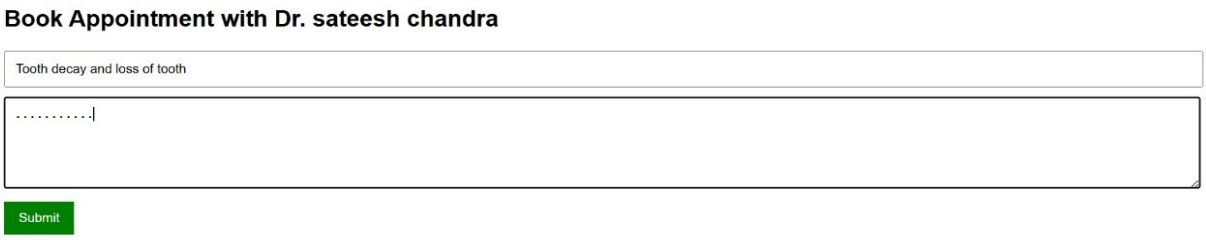

**Login Page:**

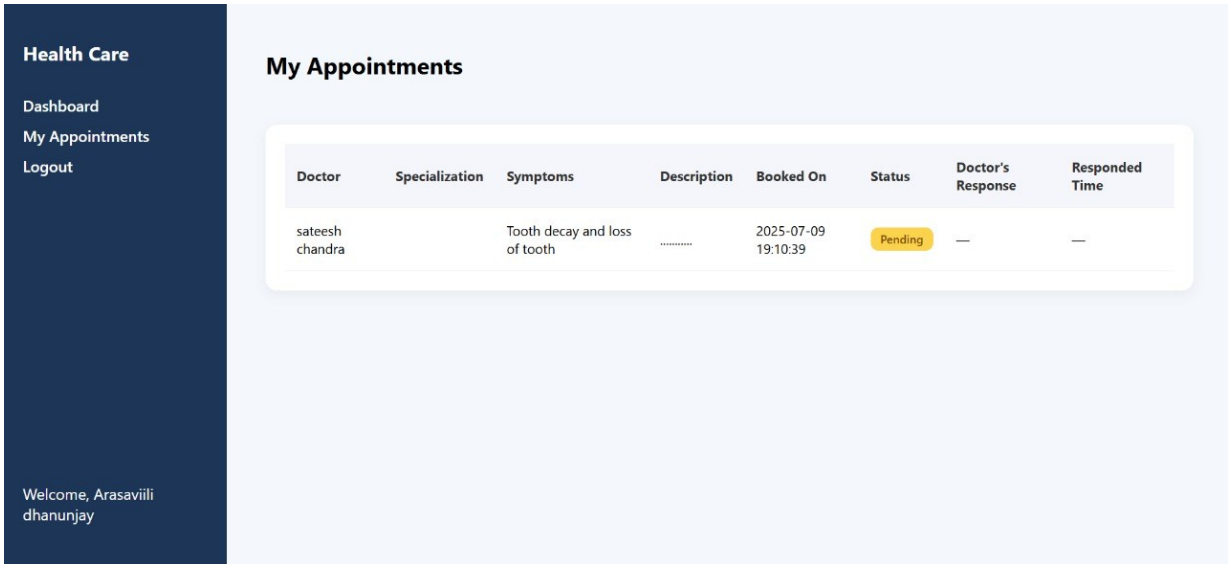



## Patient Dashboard page:

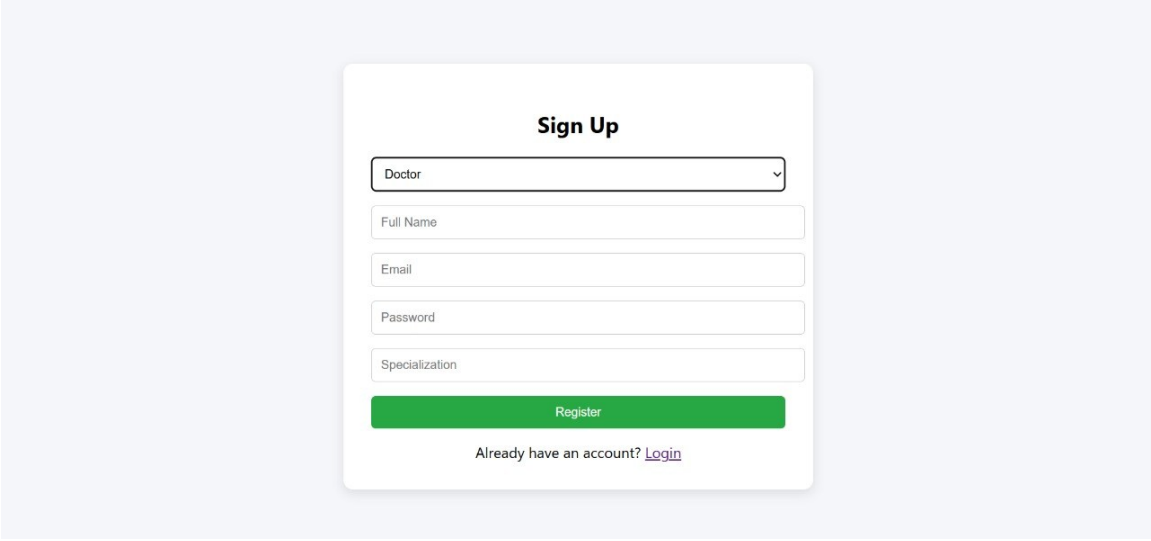


# Book Appointment page:

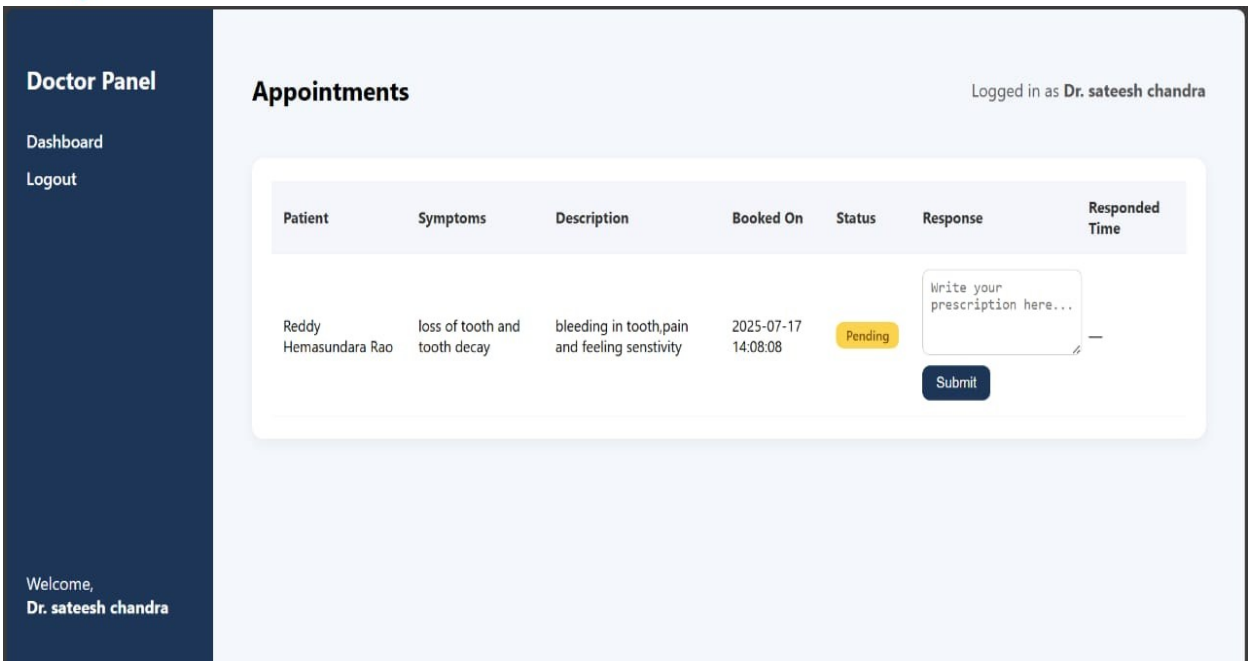





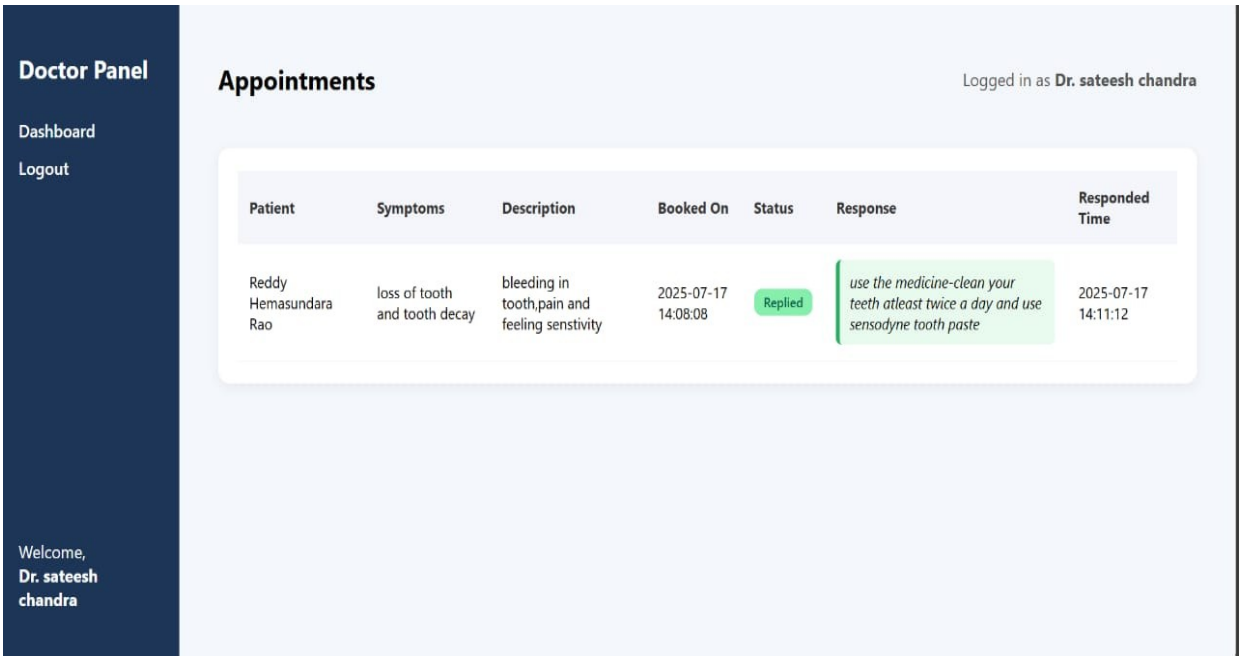

## Doctor login:



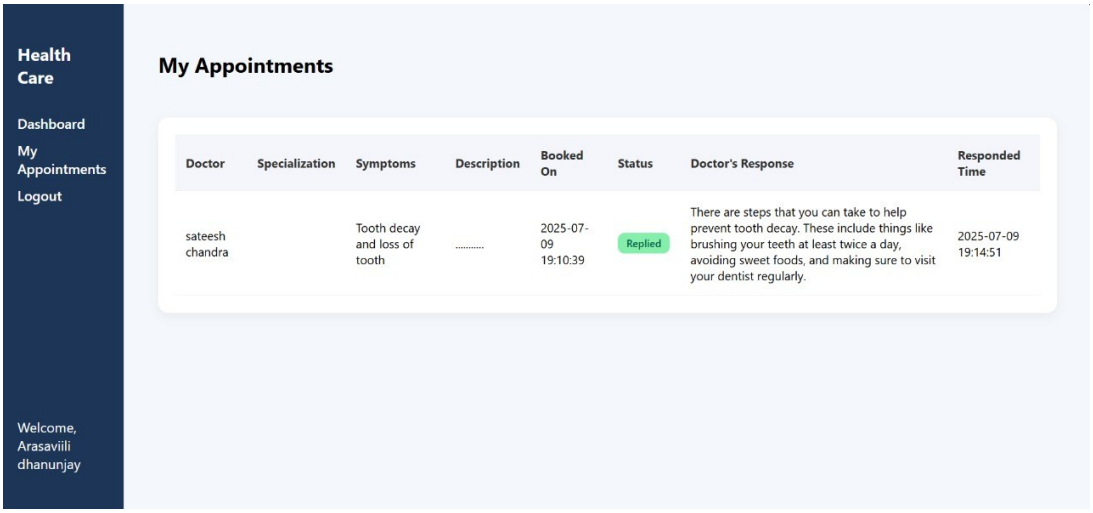

## Doctor Dashboard:




## Doctor Response:

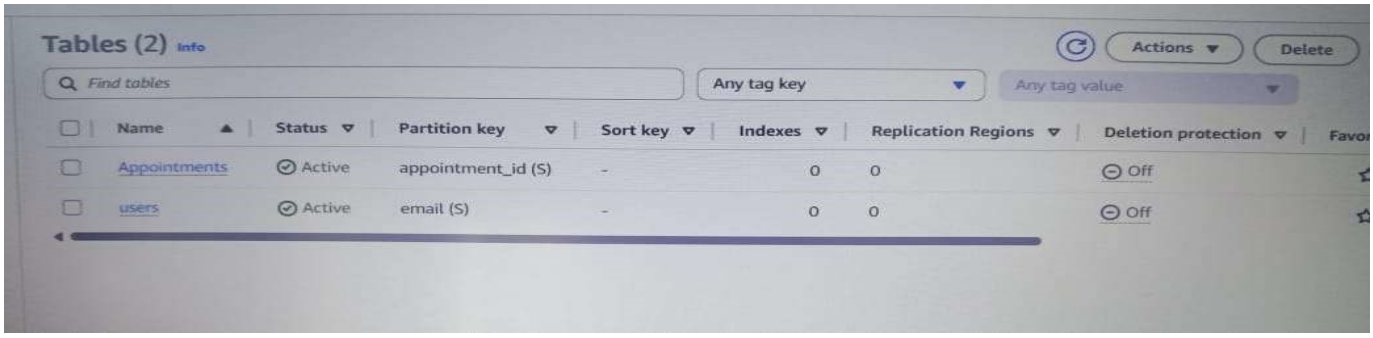



## Showing doctor response in patient dashboard:




Dynamodb Database Updations:

1. Users table and Appointments table :




## Conclusion:

The MedTrack application has been successfully developed and deployed using a robust cloud-based architecture tailored for modern healthcare environments. Leveraging AWS services such as EC2 for hosting, DynamoDB for secure and scalable patient data management, and SNS for real-time alerts, the platform ensures reliable and ef icient access to essential medical tracking services. This system addresses critical challenges in healthcare such as managing patient records, monitoring medication schedules, and ensuring timely communication between healthcare providers and patients.

The cloud-native approach enables seamless scalability, allowing MedTrack to support increasing numbers of users and data without compromising performance or reliability. The integration of

Flask with AWS ensures smooth backend operations, including patient registration, medication reminders, and health updates. Thorough testing has validated that all features—from user onboarding to alert noti ications—function reliably and securely.

In conclusion, the Med Track application delivers a smart, ef icient solution for modernizing healthcare management, improving patient care, and streamlining communication between medical staff and patients. This project highlights the transformative power of cloud-based technologies in solving real-world challenges in the healthcare sector.