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.

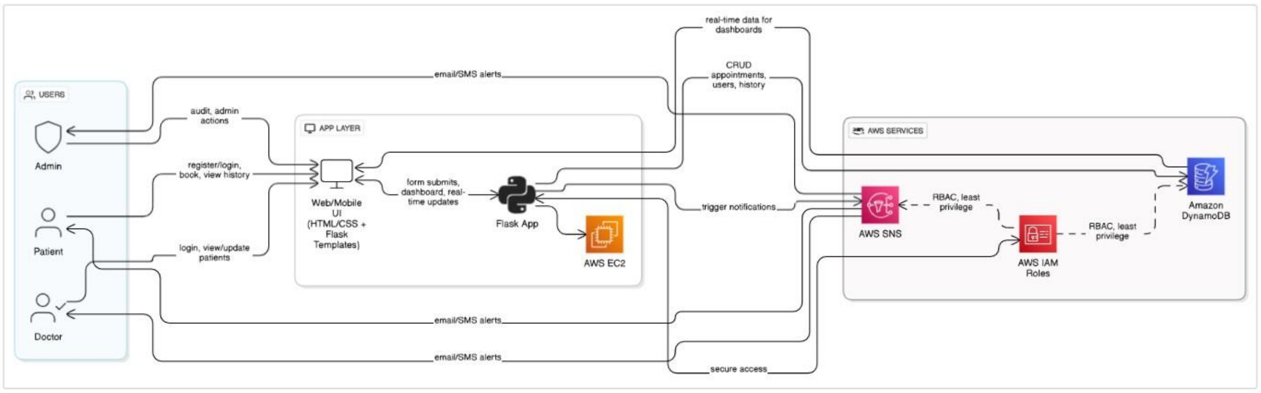
Scenario 2: Secure User Management with IAM

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.

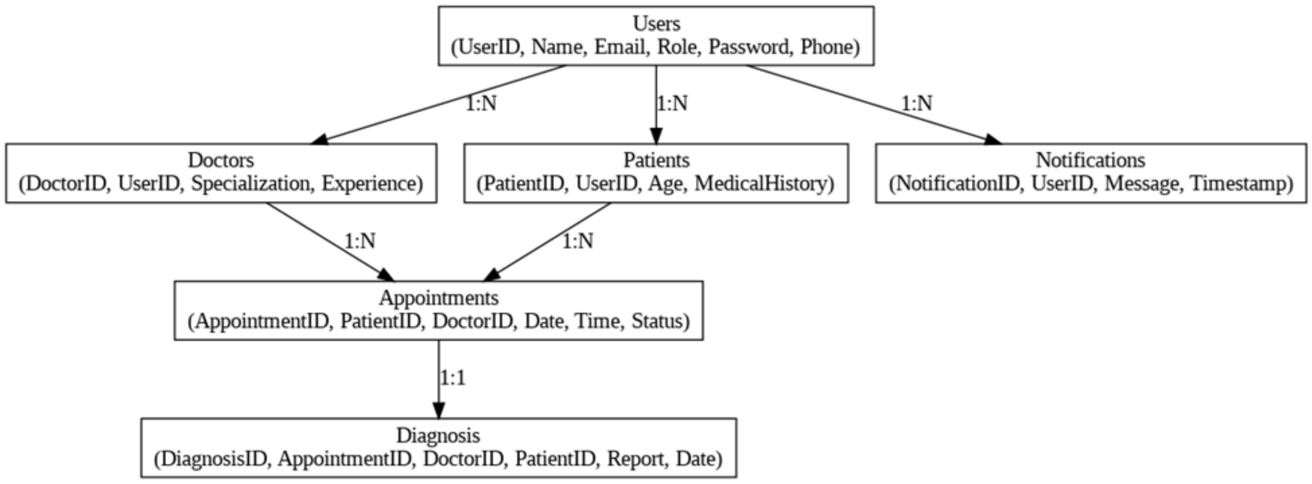
Scenario 3: Easy Access to Medical History and Resources

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:


Users
(UserID, Name, Email, Role, Password, Phone)
1:N
1:N
1:N
Doctors
(DoctorID, UserID, Specialization, Experience)
Patients
(PatientID, UserID, Age, MedicalHistory)
Notifications
(NotificationID, UserID, Message, Timestamp)
1:N
1:N
Appointments
(AppointmentID, PatientID, DoctorID, Date, Time, Status)
1:1
Diagnosis
(DiagnosisID, AppointmentID, DoctorID, PatientID, Report, Date)

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:

1. AWS Account Setup and Login

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

Activity 1.2: Log in to the AWS Management Console

2. DynamoDB Database Creation and Setup

Activity 2.1: Create a DynamoDB Table.

Activity 2.2: Configure Attributes for User Data and Book Requests.

3. SNS Notification Setup

Activity 3.1: Create SNS topics for book request notifications.

Activity 3.2: Subscribe users and library staff to SNS email notifications.

4. Backend Development and Application Setup

Activity 4.1:Develop the Backend Using Flask.

Activity 4.2: Integrate AWS Services Using boto3.

5. IAM Role Setup

Activity 5.1: Create IAM Role

Activity 5.2: Attach Policies

6. EC2 Instance Setup

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

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

7. Deployment on EC2

Activity 7.1:Upload Flask Files

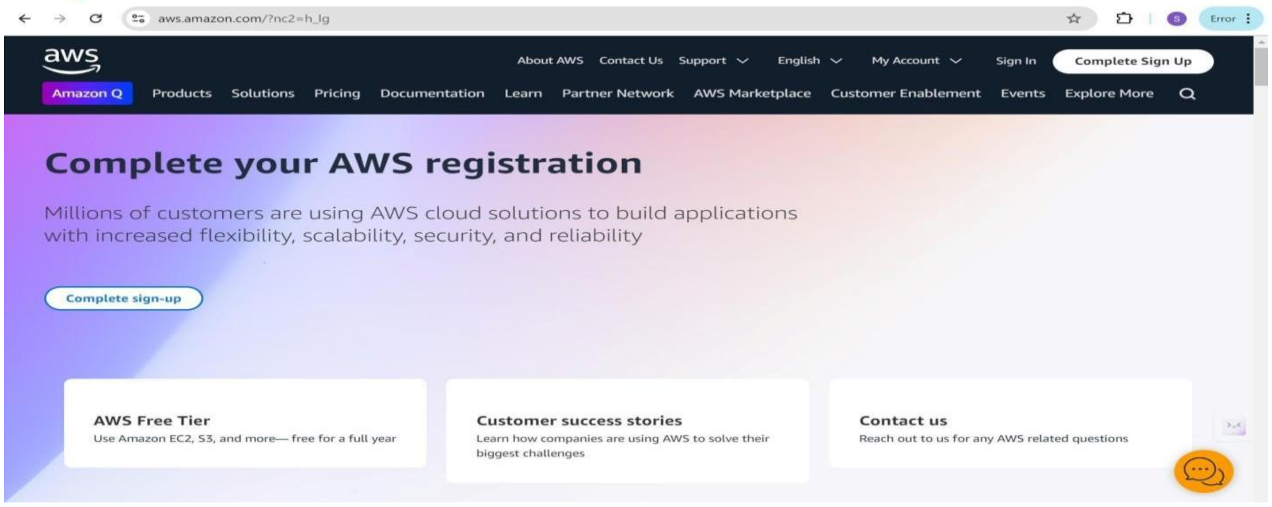
Activity 7.2: Run the Flask App

8. Testing and Deployment

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

Milestone 1: AWS Account Setup and Login

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



Complete your AWS registration

Millions of customers are using AWS cloud solutions to build applications with increased flexibility, scalability, security, and reliability

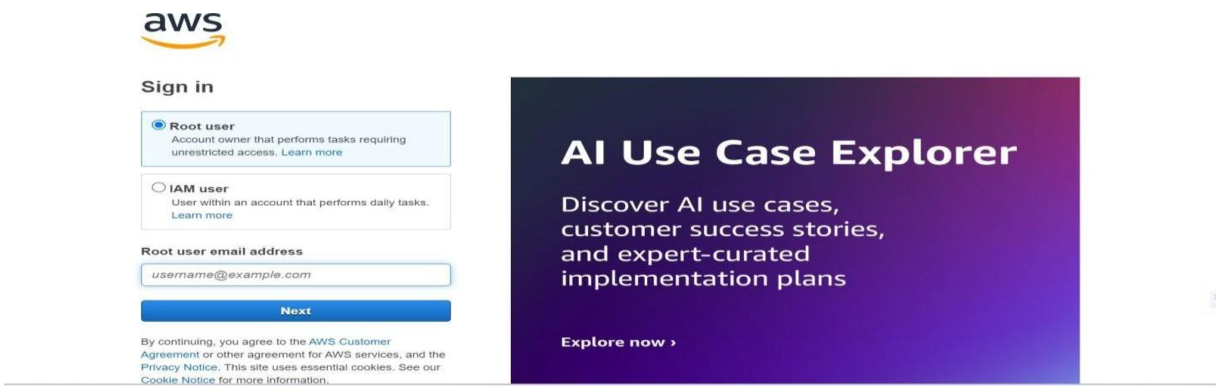
Complete sign-up

AWS Free Tier
Use Amazon EC2, S3, and more— free for a full year

Customer success stories
Learn how companies are using AWS to solve their biggest challenges

Contact us
Reach out to us for any AWS related questions

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



Sign in

Root user
Account owner that performs tasks requiring unrestricted access. Learn more

IAM user
User within an account that performs daily tasks. Learn more

Root user email address
username@example.com

Next

By continuing, you agree to the AWS Customer Agreement or other agreement for AWS services, and the Privacy Notice. This site uses essential cookies. See our Cookie Notice for more information.

AI Use Case Explorer

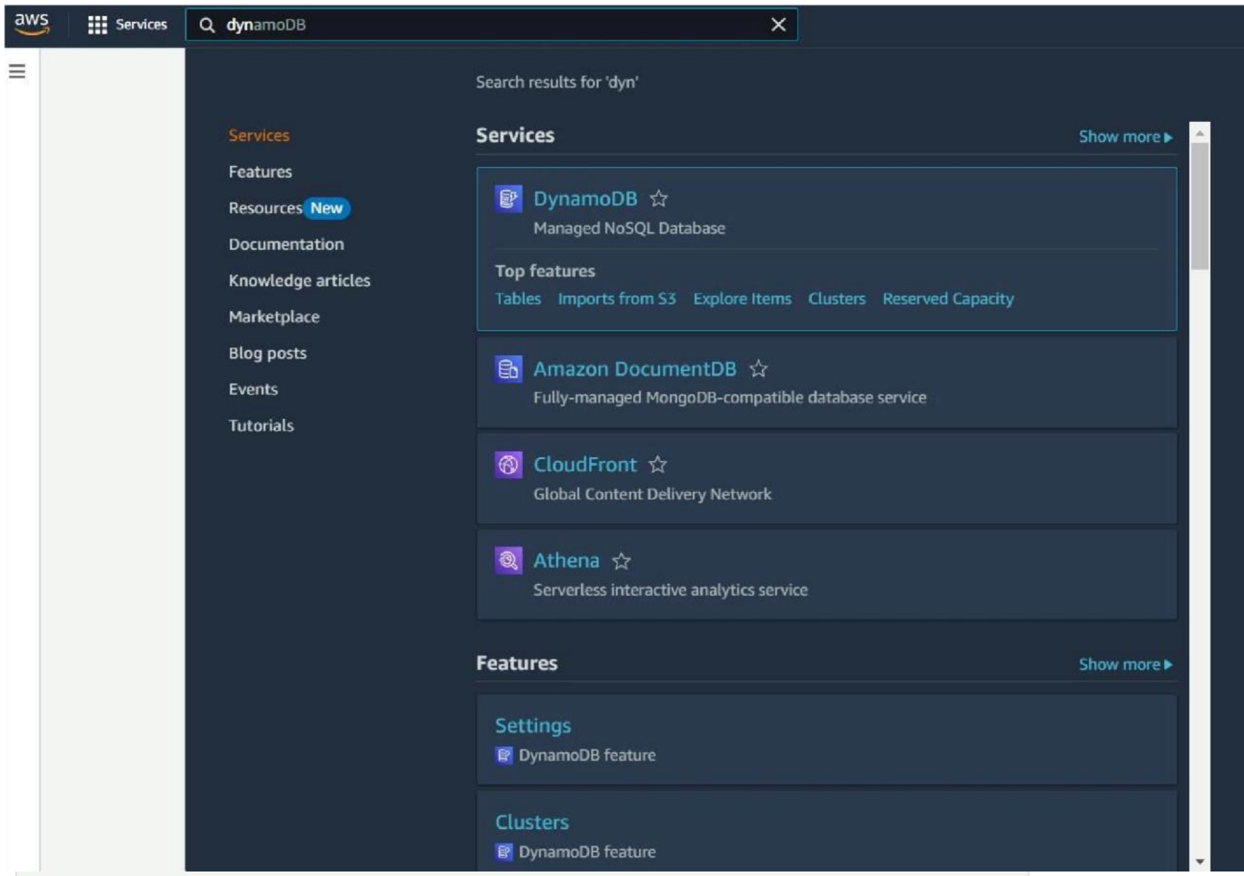
Discover AI use cases, customer success stories, and expert-curated implementation plans

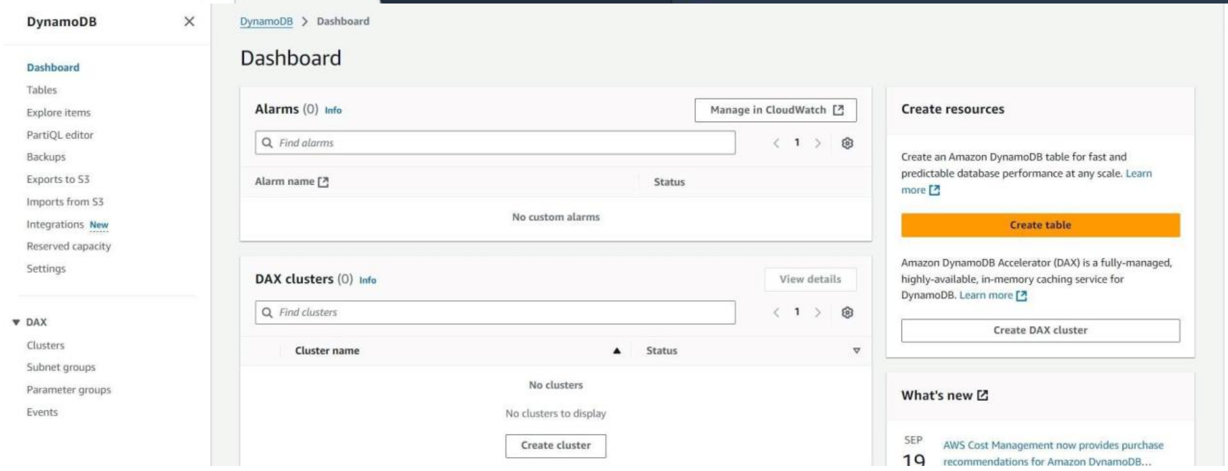
Explore now ›

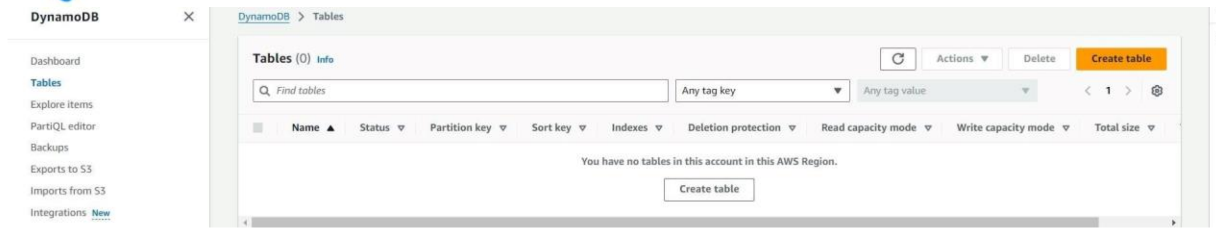
Milestone 2: DynamoDB Database Creation and Setup

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

○


aws
Services
dynamoDB
Search results for 'dyn'
Services
Features
Resources New
Documentation
Knowledge articles
Marketplace
Blog posts
Events
Tutorials
Services
Show more ▸
DynamoDB
Managed NoSQL Database
Top features
Tables
Imports from S3
Explore Items
Clusters
Reserved Capacity
Amazon DocumentDB
Fully-managed MongoDB-compatible database service
CloudFront
Global Content Delivery Network
Athena
Serverless interactive analytics service
Features
Show more ▸
Settings
DynamoDB feature
Clusters
DynamoDB feature


DynamoDB
Dashboard
Tables
Explore items
PartiQL editor
Backups
Exports to S3
Imports from S3
Integrations New
Reserved capacity
Settings
DAX
Clusters
Subnet groups
Parameter groups
Events
DynamoDB > Dashboard
Dashboard
Alarms (0) Info
Manage in CloudWatch
Find alarms
Alarm name
Status
No custom alarms
DAX clusters (0) Info
View details
Find clusters
Cluster name
Status
No clusters
No clusters to display
Create cluster
Create resources
Create an Amazon DynamoDB table for fast and predictable database performance at any scale. Learn more
Create table
Amazon DynamoDB Accelerator (DAX) is a fully-managed, highly-available, in-memory caching service for DynamoDB. Learn more
Create DAX cluster
What's new
SEP 19
AWS Cost Management now provides purchase recommendations for Amazon DynamoDB...

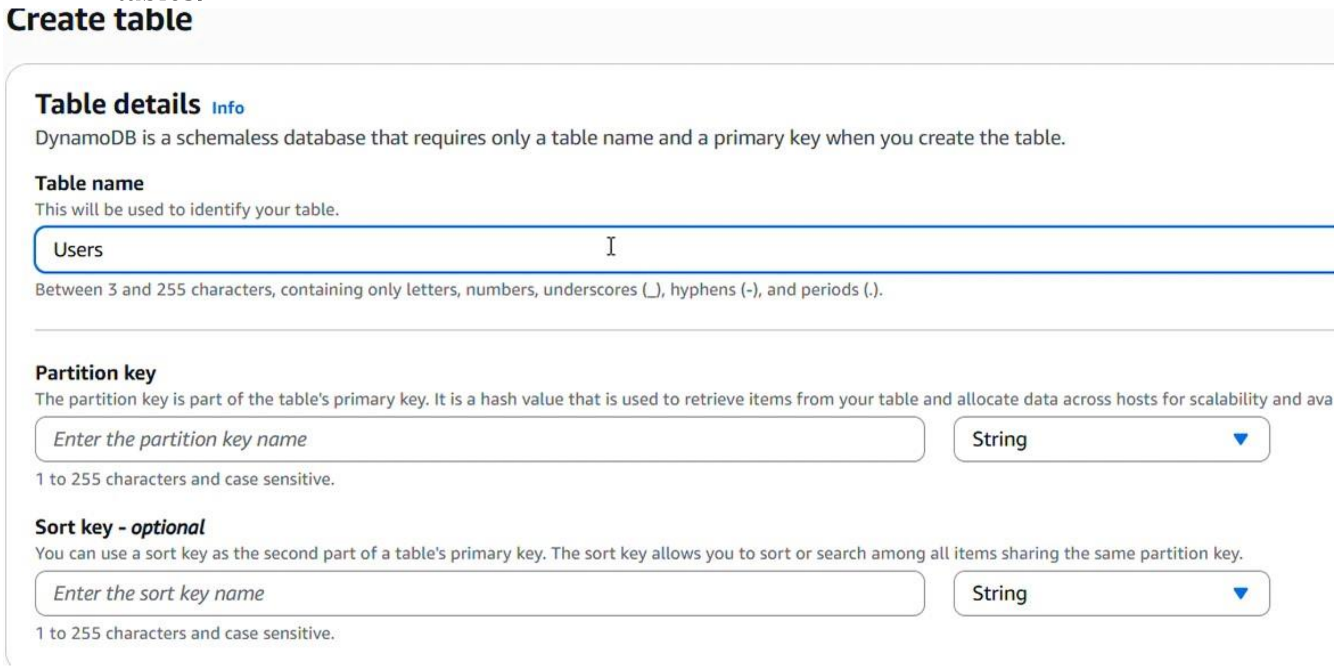


DynamoDB > Tables

Tables (0) Info

You have no tables in this account in this AWS Region.

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



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 ava

Enter the partition key name — String

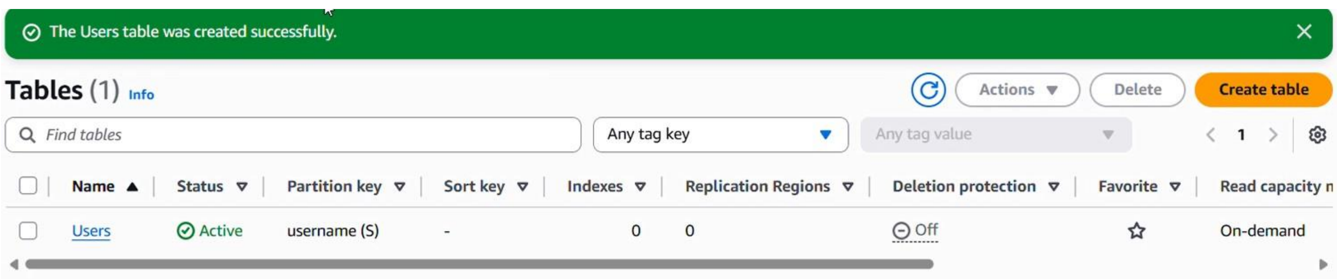
1 to 255 characters and case sensitive.

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.



The Users table was created successfully.

Tables (1) Info

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity n
Users	Active	username (S)	-	0	0	Off	☆	On-demand

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

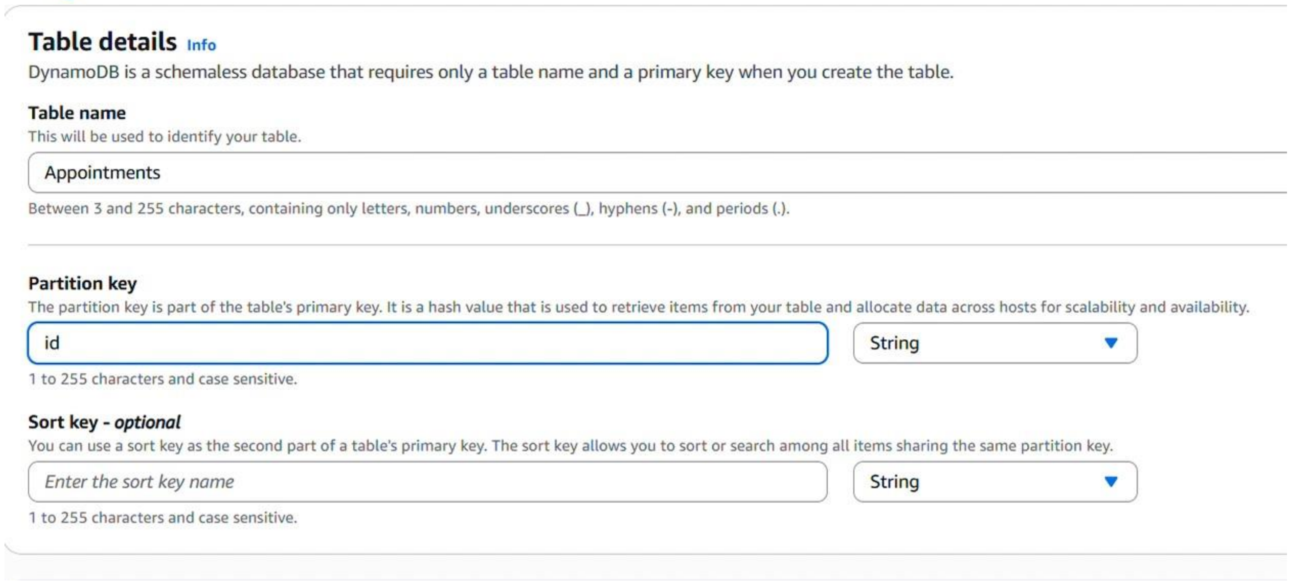


Table details Info

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.

Appointments

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.

id | 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.

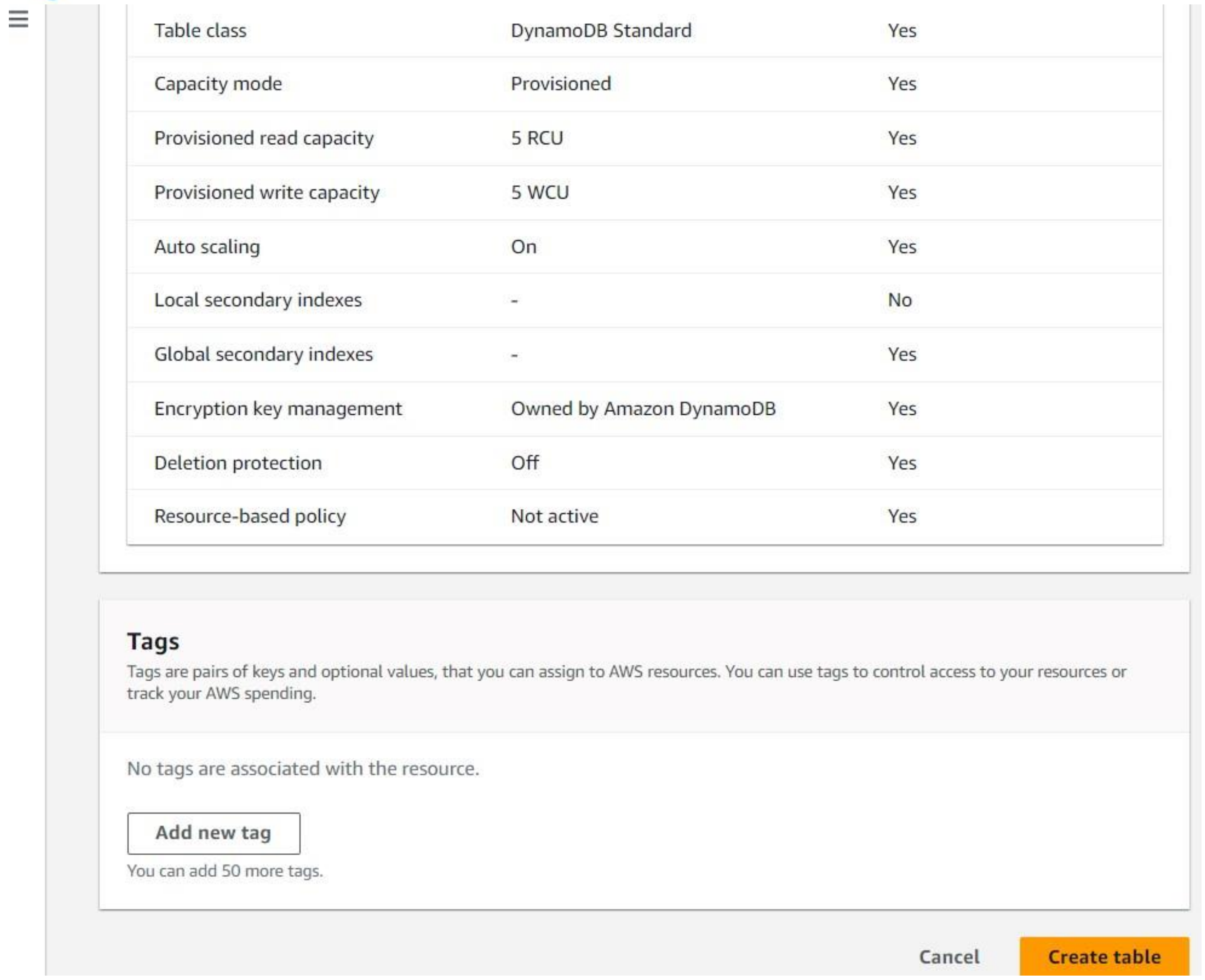


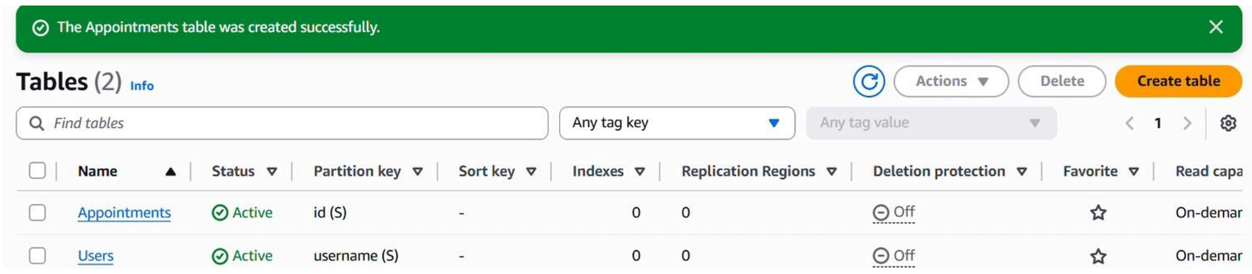
Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based 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.

You can add 50 more tags.



The Appointments table was created successfully.

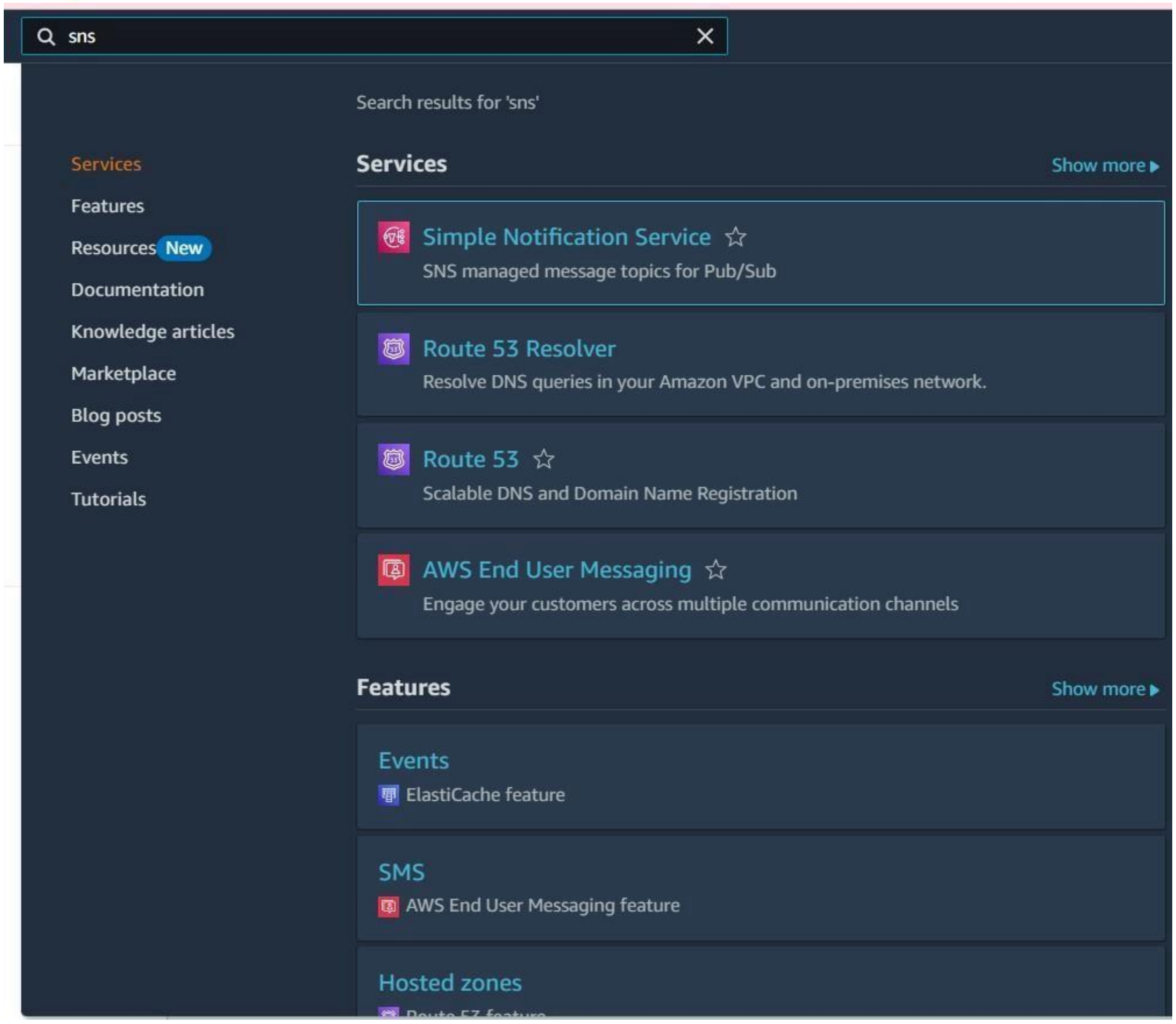
Tables (2) Info

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capa
Appointments	Active	id (S)	-	0	0	Off	☆	On-demar
Users	Active	username (S)	-	0	0	Off	☆	On-demar

Milestone 3: SNS Notification Setup

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

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


sns

Search results for 'sns'

Services

Features

Resources New

Documentation

Knowledge articles

Marketplace

Blog posts

Events

Tutorials

Services Show more

Simple Notification Service
SNS managed message topics for Pub/Sub

Route 53 Resolver
Resolve DNS queries in your Amazon VPC and on-premises network.

Route 53
Scalable DNS and Domain Name Registration

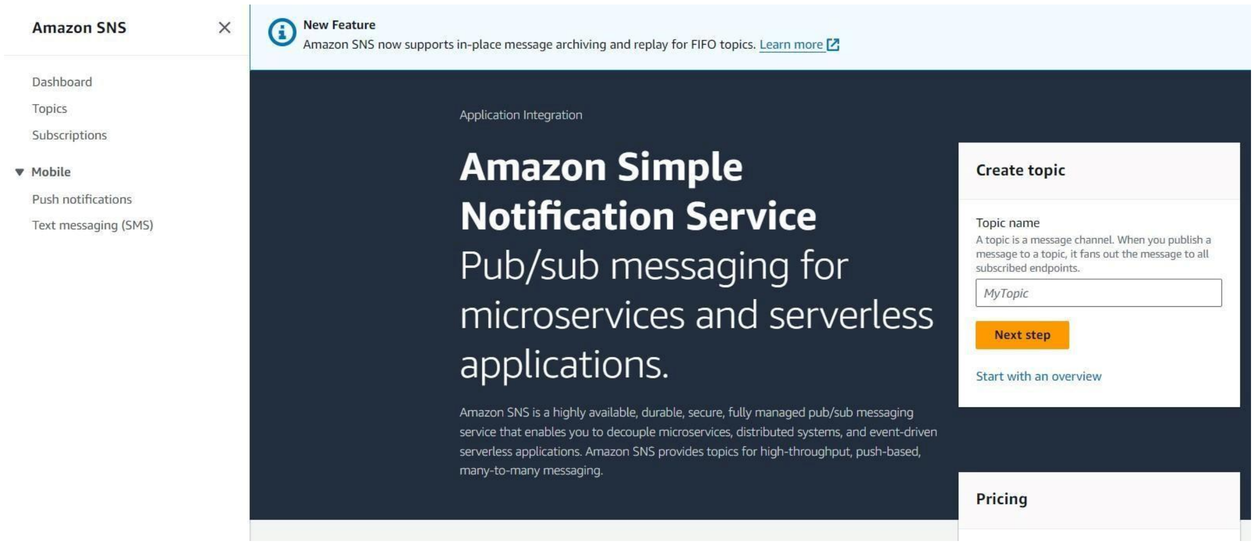
AWS End User Messaging
Engage your customers across multiple communication channels

Features Show more

Events
ElastiCache feature

SMS
AWS End User Messaging feature

Hosted zones


Amazon SNS

Dashboard
Topics
Subscriptions
Mobile
Push notifications
Text messaging (SMS)

New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics. Learn more

Application Integration

Amazon Simple Notification Service
Pub/sub messaging for microservices and serverless applications.

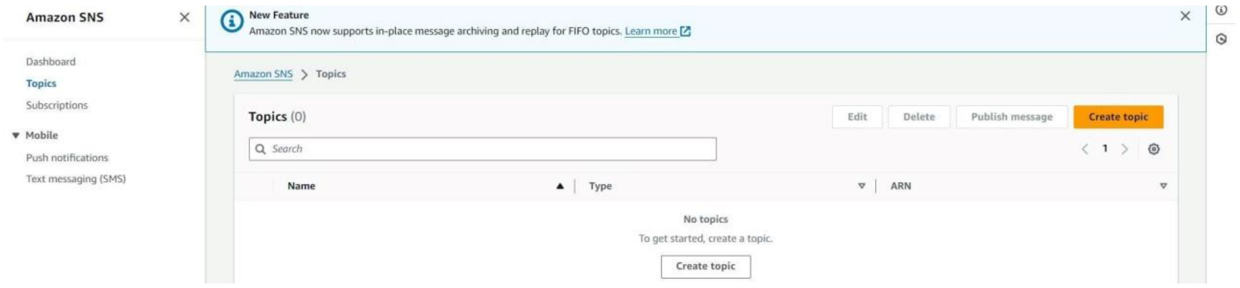
Amazon SNS is a highly available, durable, secure, fully managed pub/sub messaging service that enables you to decouple microservices, distributed systems, and event-driven serverless applications. Amazon SNS provides topics for high-throughput, push-based, many-to-many messaging.

Create topic

Topic name
A topic is a message channel. When you publish a message to a topic, it fans out the message to all subscribed endpoints.
MyTopic
Next step
Start with an overview

Pricing

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


Amazon SNS

Dashboard
Topics
Subscriptions
Mobile
Push notifications
Text messaging (SMS)

New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics. Learn more

Amazon SNS > Topics

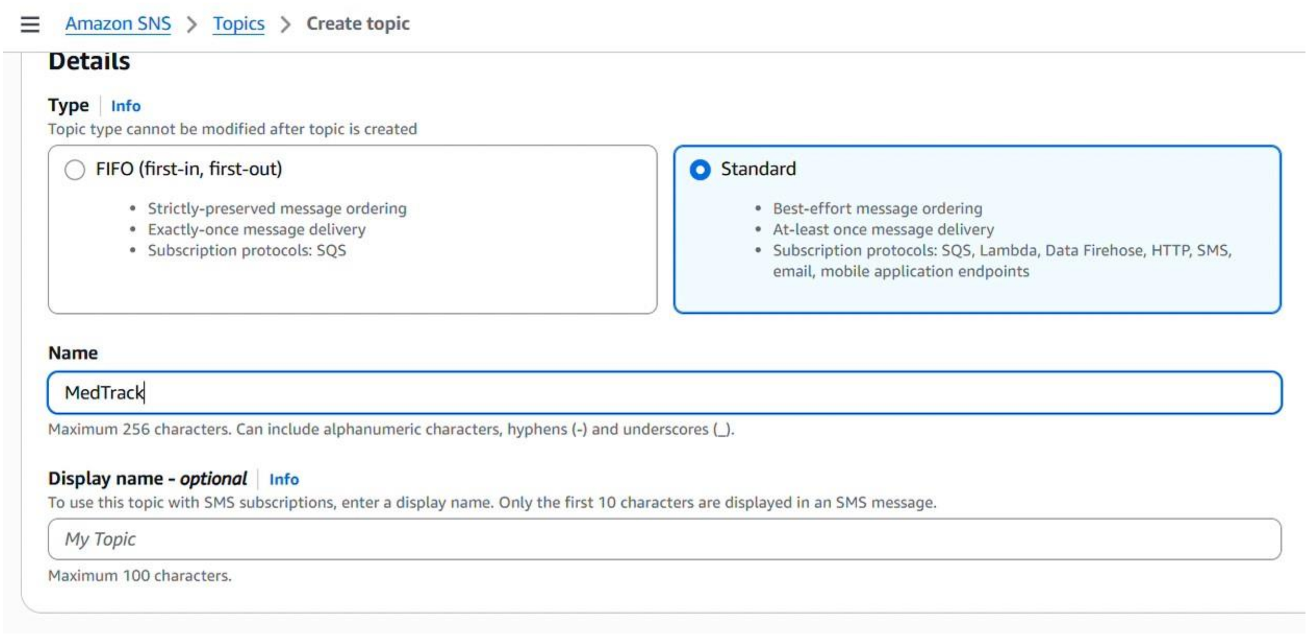
Topics (0)

Edit | Delete | Publish message | Create topic

Name	Type	ARN

No topics
To get started, create a topic.
Create topic

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


Amazon SNS > Topics > Create topic

Details

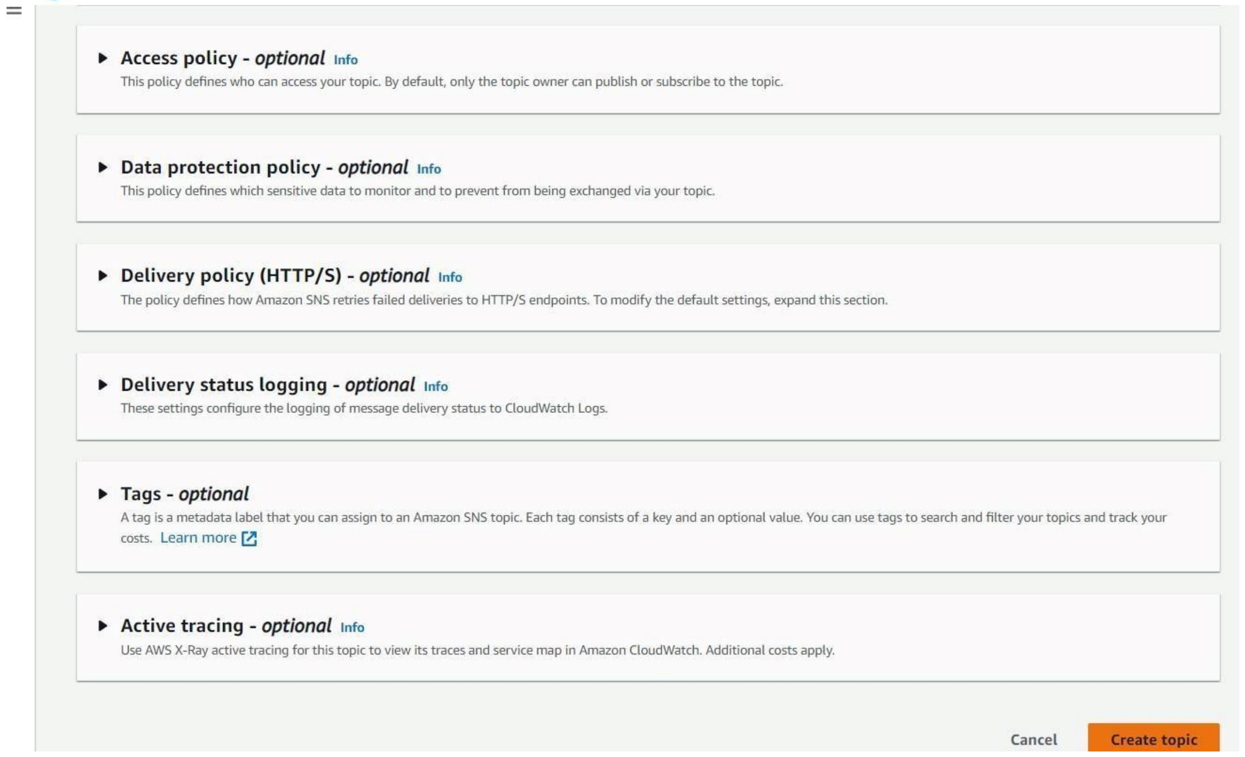
Type Info
Topic type cannot be modified after topic is created

( ) FIFO (first-in, first-out)
- Strictly-preserved message ordering
- Exactly-once message delivery
- Subscription protocols: SQS

[✓ Standard]
- Best-effort message ordering
- At-least once message delivery
- Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints

Name
MedTrack
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).

Display name - *optional* Info
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.
My Topic
Maximum 100 characters.



▸ Access policy - *optional* Info
This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.

▸ Data protection policy - *optional* Info
This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.

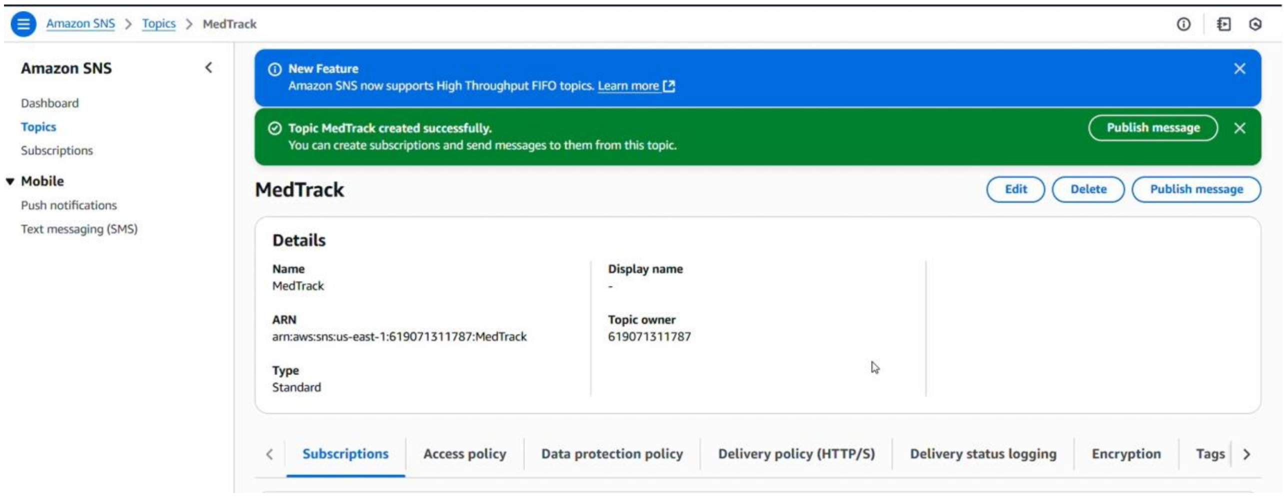
▸ Delivery policy (HTTP/S) - *optional* Info
The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.

▸ Delivery status logging - *optional* Info
These settings configure the logging of message delivery status to CloudWatch Logs.

▸ Tags - *optional*
A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. Learn more

▸ Active tracing - *optional* Info
Use AWS X-Ray active tracing for this topic to view its traces and service map in Amazon CloudWatch. Additional costs apply.

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



New Feature
Amazon SNS now supports High Throughput FIFO topics. Learn more

Topic MedTrack created successfully.
You can create subscriptions and send messages to them from this topic.

MedTrack

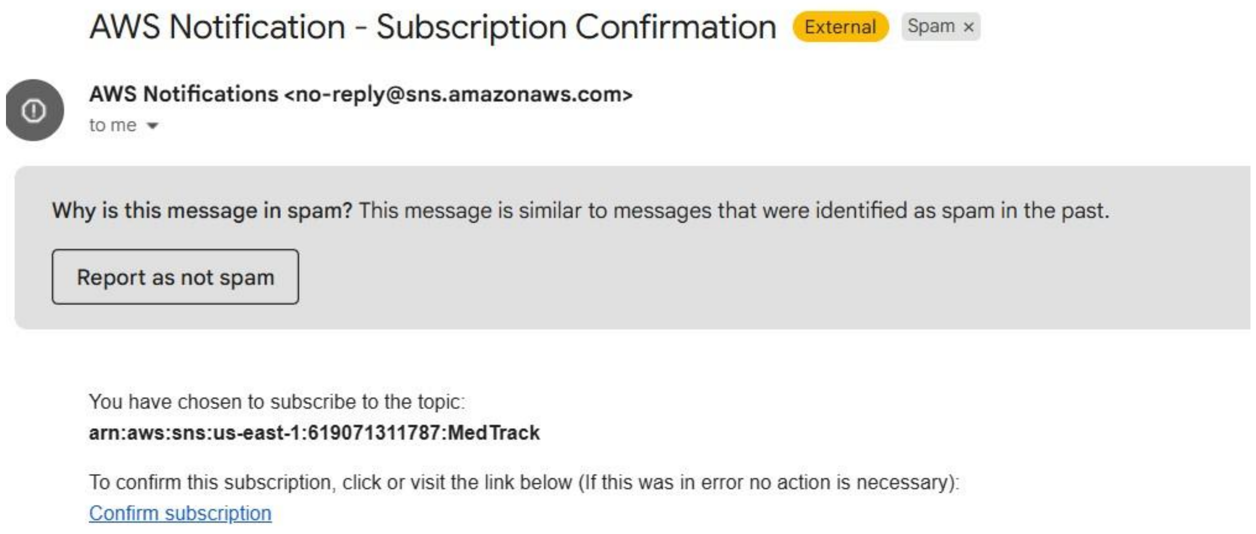
Details

Name MedTrack	Display name -
ARN arn:aws:sns:us-east-1:619071311787:MedTrack	Topic owner 619071311787
Type Standard	

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

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

- After subscription request for the mail confirmation



AWS Notification - Subscription Confirmation

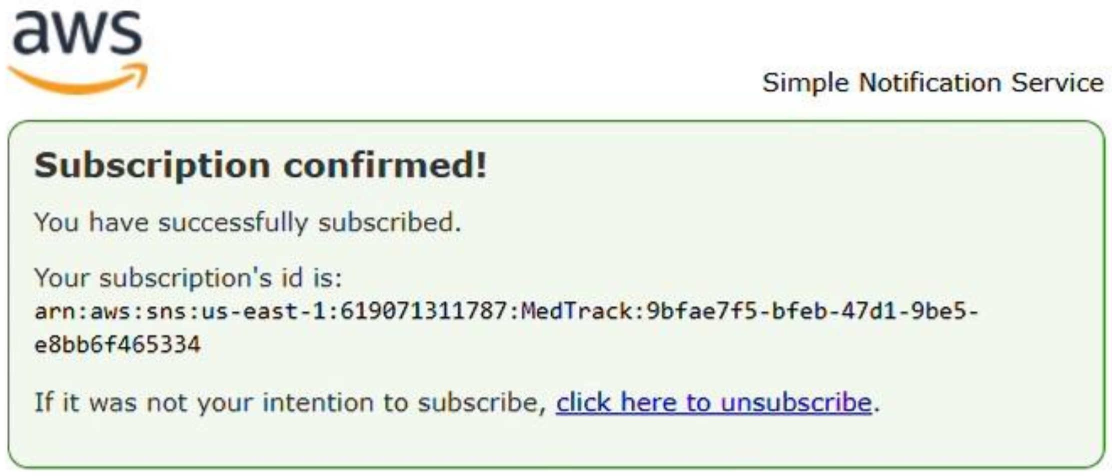
AWS Notifications <no-reply@sns.amazonaws.com>

Why is this message in spam? This message is similar to messages that were identified as spam in the past.

You have chosen to subscribe to the topic:
arn:aws:sns:us-east-1:619071311787:MedTrack

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):
Confirm subscription

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


Simple Notification Service

Subscription confirmed!

You have successfully subscribed.

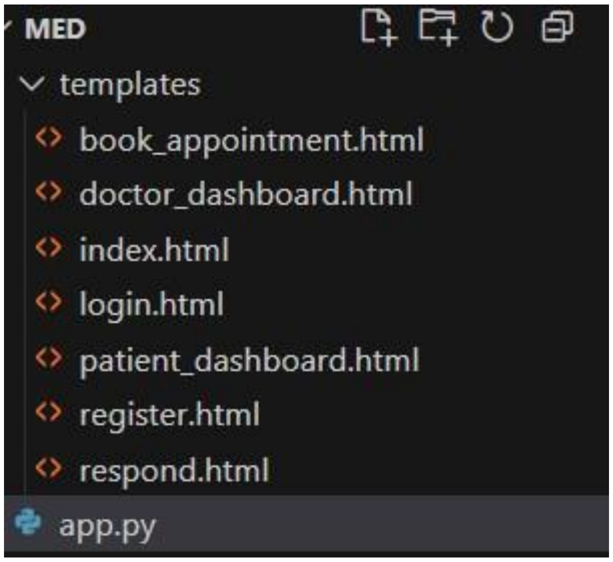
Your subscription's id is:
arn:aws:sns:us-east-1:619071311787:MedTrack:9bfae7f5-bfeb-47d1-9be5-e8bb6f465334

If it was not your intention to subscribe, click here to unsubscribe.

- ○ 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

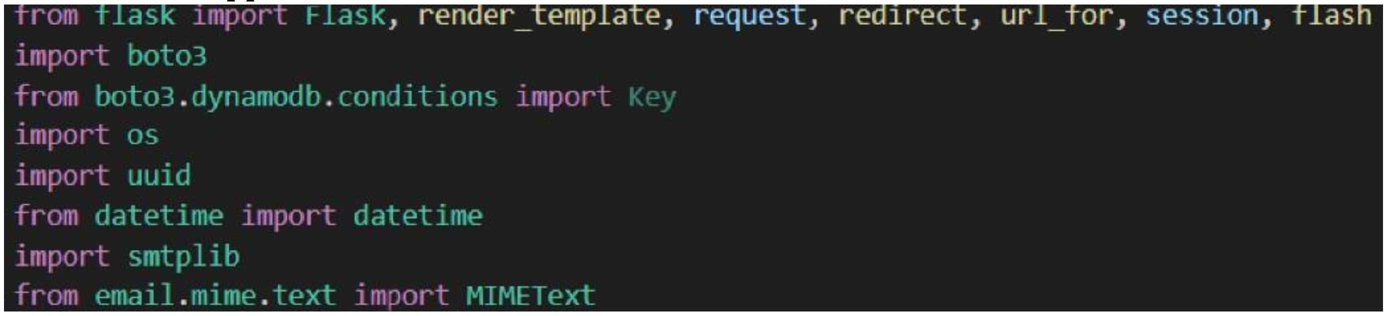


```
MED
  templates
    book_appointment.html
    doctor_dashboard.html
    index.html
    login.html
    patient_dashboard.html
    register.html
    respond.html
  app.py
```

Description: set up the INSTANT LIBRARY project with an app.py file and a templates/ directory containing all required HTML pages like home, login, register, subject-specific pages (e.g., index..html, login.html), and utility pages (e.g., respond.html).

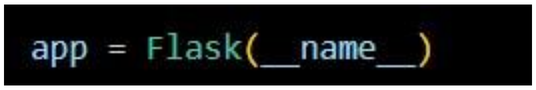
Description of the code :

- **Flask App Initialization**



```python
from flask import Flask, render_template, request, redirect, url_for, session, flash
import boto3
from boto3.dynamodb.conditions import Key
import os
import uuid
from datetime import datetime
import smtplib
from email.mime.text import MIMEText
```

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



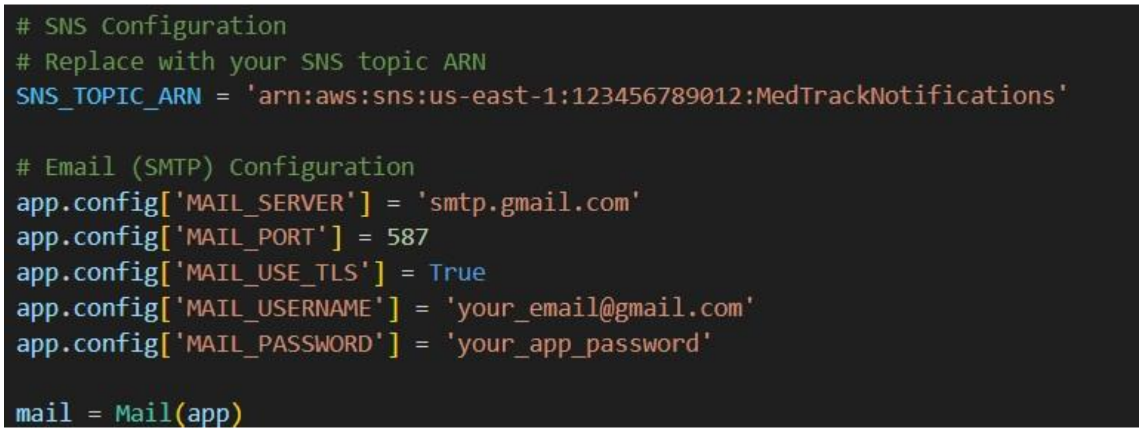
```python
app = Flask(__name__)
```

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

- **Dynamodb Setup:**

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

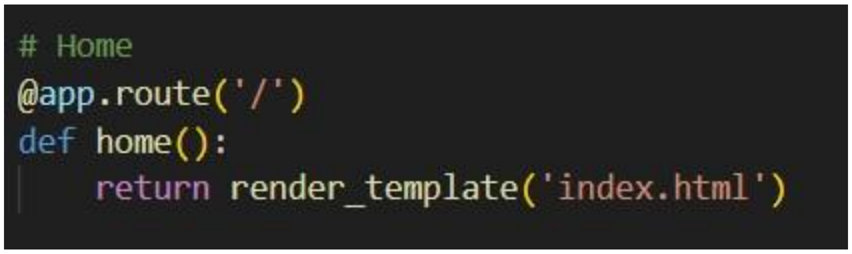
- **SNS Connection**



```python
# SNS Configuration
# Replace with your SNS topic ARN
SNS_TOPIC_ARN = 'arn:aws:sns:us-east-1:123456789012:MedTrackNotifications'

# Email (SMTP) Configuration
app.config['MAIL_SERVER'] = 'smtp.gmail.com'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USERNAME'] = 'your_email@gmail.com'
app.config['MAIL_PASSWORD'] = 'your_app_password'

mail = Mail(app)
```

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**

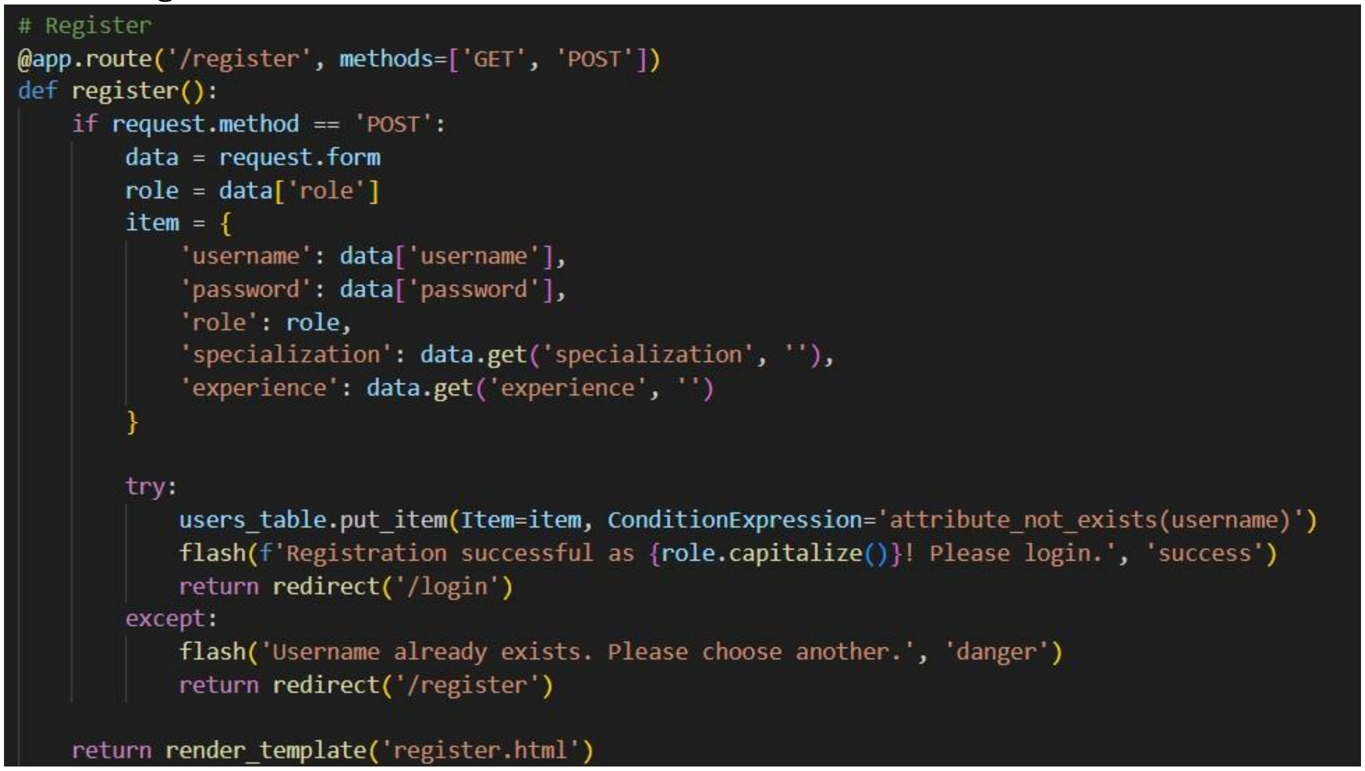
- **Home Route:**



```python
# Home
@app.route('/')
def home():
    return render_template('index.html')
```

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

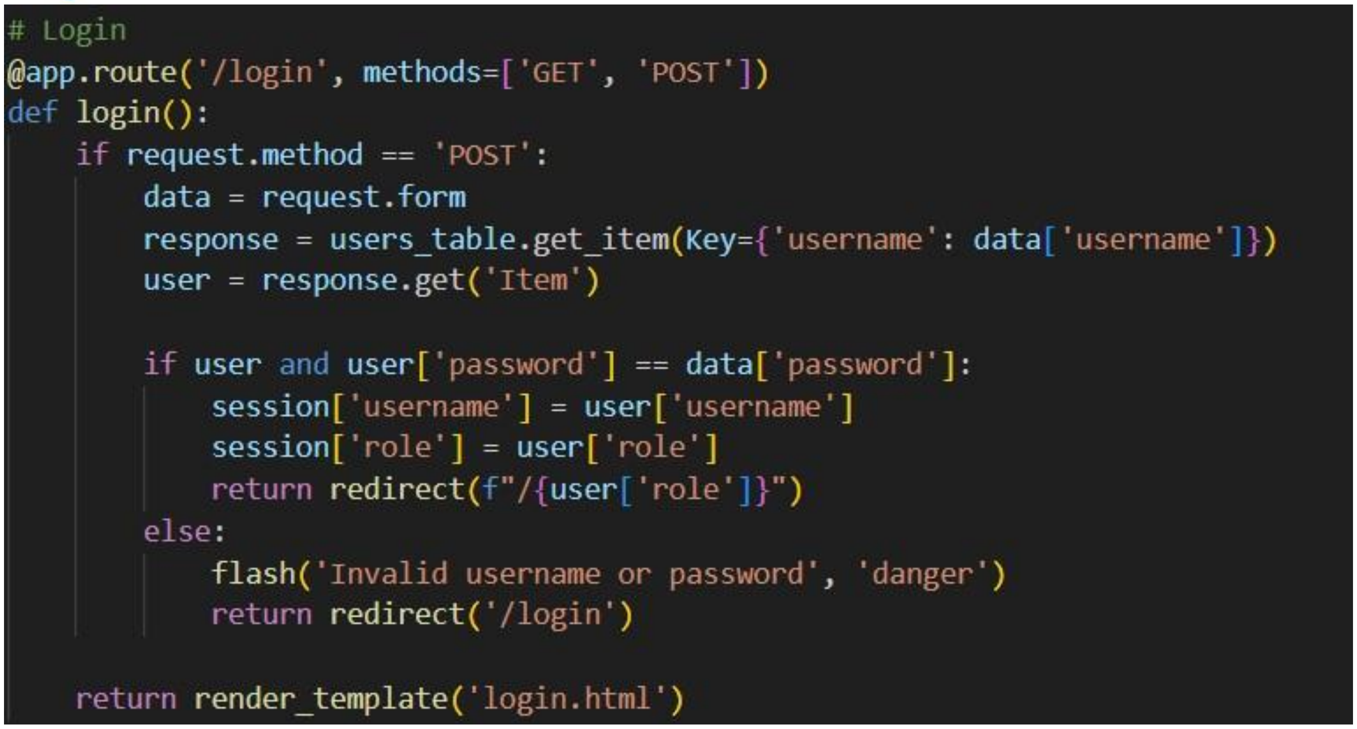
- **Register Route:**



```python
# Register
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        data = request.form
        role = data['role']
        item = {
            'username': data['username'],
            'password': data['password'],
            'role': role,
            'specialization': data.get('specialization', ''),
            'experience': data.get('experience', '')
        }

        try:
            users_table.put_item(Item=item, ConditionExpression='attribute_not_exists(username)')
            flash(f'Registration successful as {role.capitalize()}! Please login.', 'success')
            return redirect('/login')
        except:
            flash('Username already exists. Please choose another.', 'danger')
            return redirect('/register')

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

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.

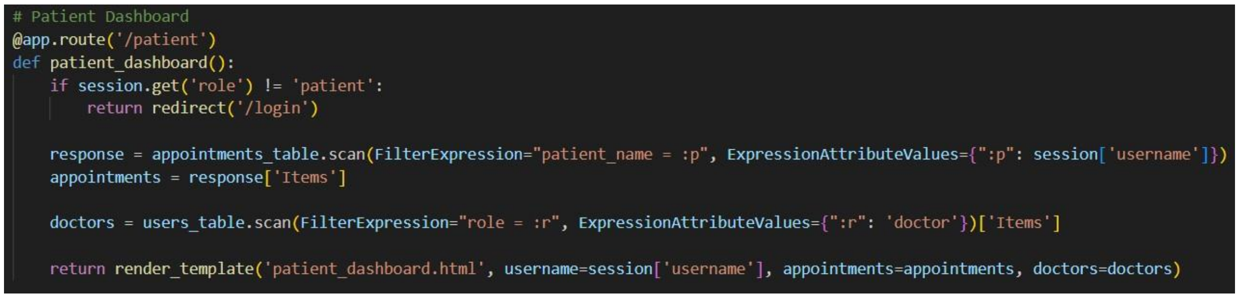
- **login Route (GET/POST)**:



```python
# Login
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        data = request.form
        response = users_table.get_item(Key={'username': data['username']})
        user = response.get('Item')

        if user and user['password'] == data['password']:
            session['username'] = user['username']
            session['role'] = user['role']
            return redirect(f"/{user['role']}")
        else:
            flash('Invalid username or password', 'danger')
            return redirect('/login')

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

Description: define /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 home page

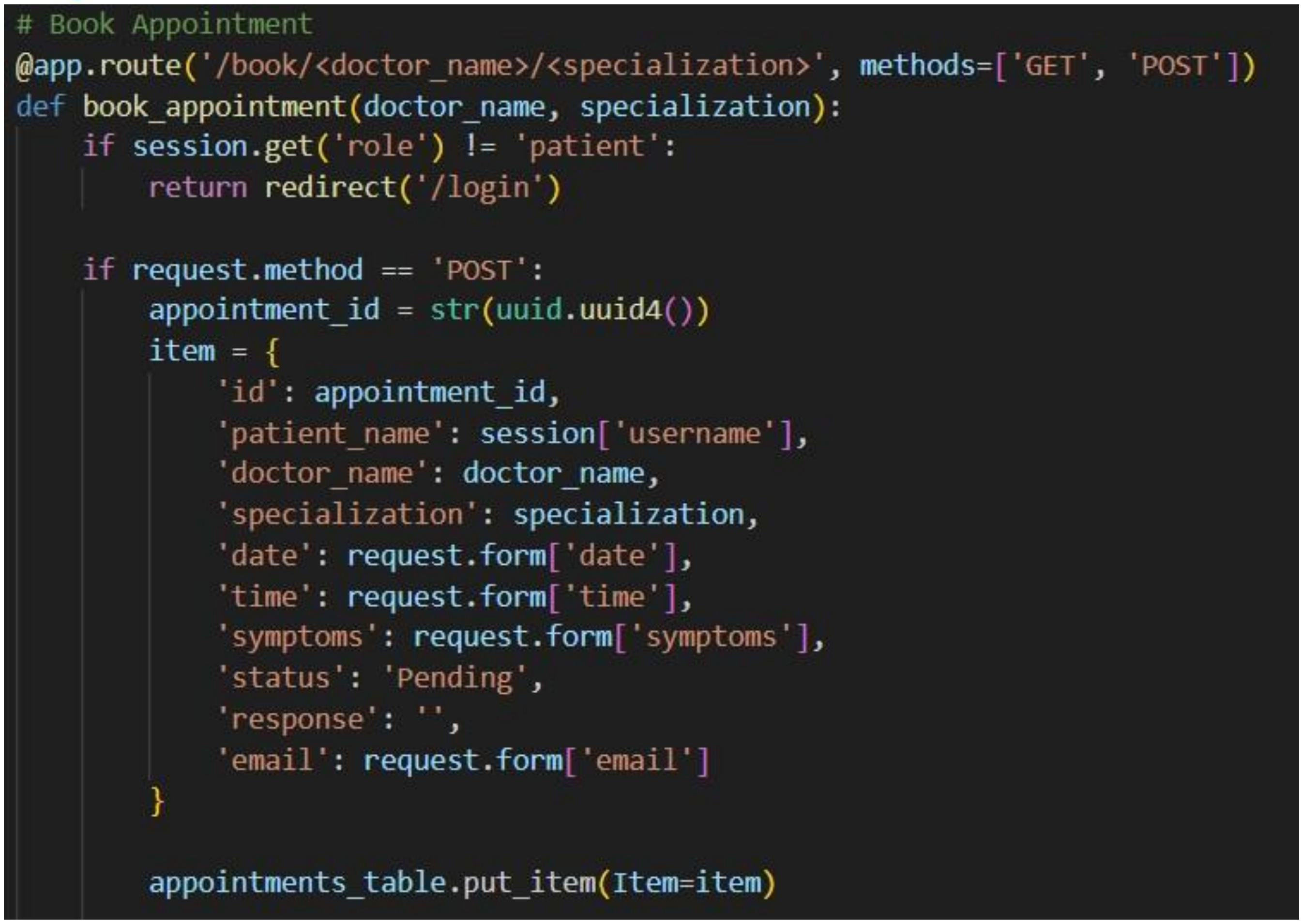
- **Patient Dashboard Route:**



```python
# Patient Dashboard
@app.route('/patient')
def patient_dashboard():
    if session.get('role') != 'patient':
        return redirect('/login')

    response = appointments_table.scan(FilterExpression="patient_name = :p", ExpressionAttributeValues={":p": session['username']})
    appointments = response['Items']

    doctors = users_table.scan(FilterExpression="role = :r", ExpressionAttributeValues={":r": 'doctor'})['Items']

    return render_template('patient_dashboard.html', username=session['username'], appointments=appointments, doctors=doctors)
```

Description: Loads the patient dashboard after login. Patients can view their upcoming appointments, cancel them, and browse available doctors to book new appointments.

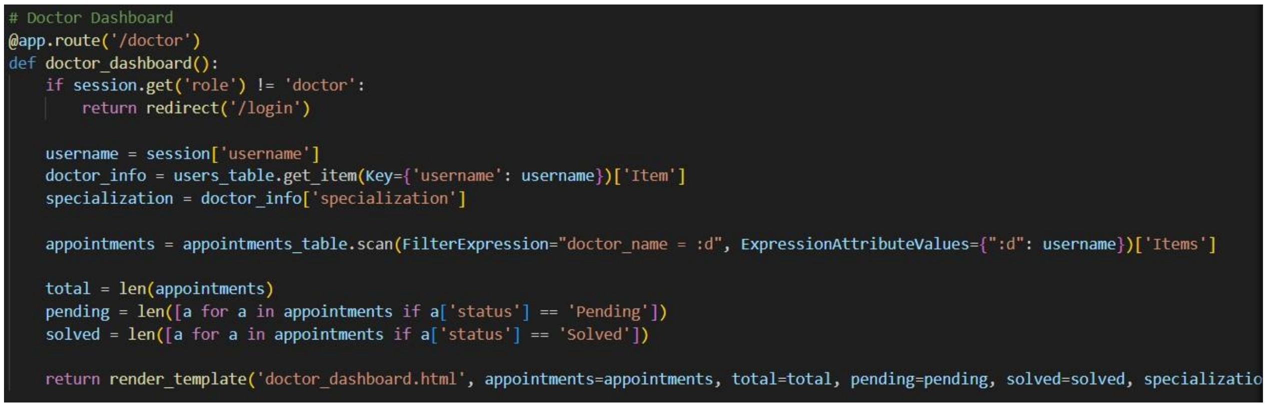
- **Book Appointment Route:**



```python
# Book Appointment
@app.route('/book/<doctor_name>/<specialization>', methods=['GET', 'POST'])
def book_appointment(doctor_name, specialization):
    if session.get('role') != 'patient':
        return redirect('/login')

    if request.method == 'POST':
        appointment_id = str(uuid.uuid4())
        item = {
            'id': appointment_id,
            'patient_name': session['username'],
            'doctor_name': doctor_name,
            'specialization': specialization,
            'date': request.form['date'],
            'time': request.form['time'],
            'symptoms': request.form['symptoms'],
            'status': 'Pending',
            'response': '',
            'email': request.form['email']
        }

        appointments_table.put_item(Item=item)
```

Description: Displays the appointment booking form for the selected doctor. Patients provide date, time, symptoms, and email to book an appointment and receive notification.

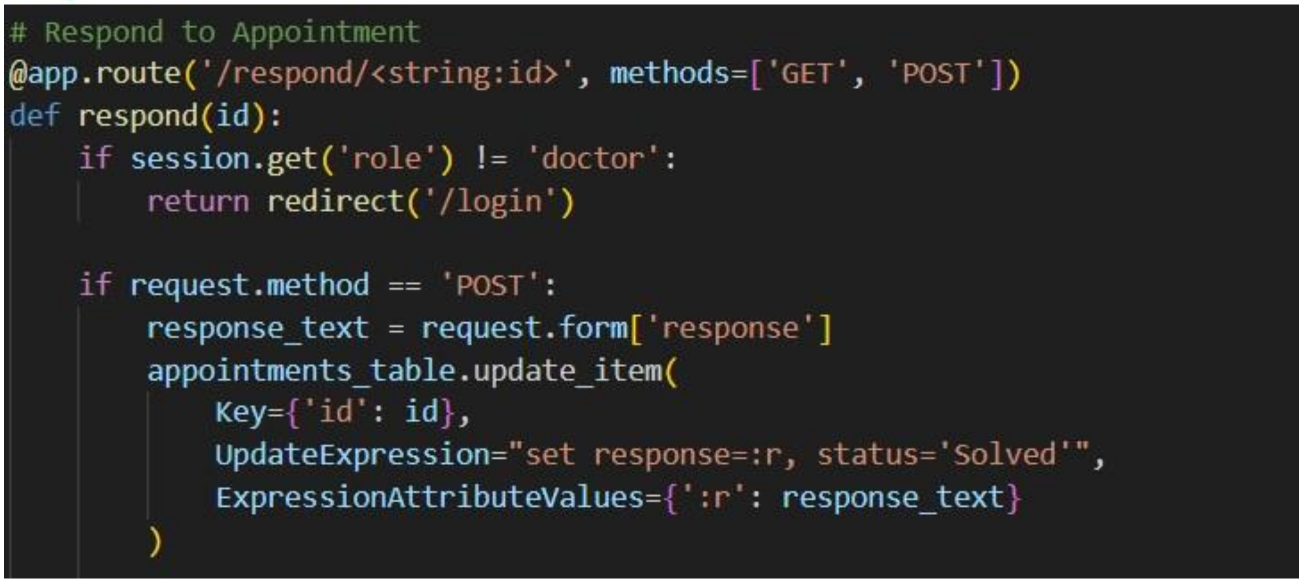
- **Doctor Dashboard Route:**



```python
# Doctor Dashboard
@app.route('/doctor')
def doctor_dashboard():
    if session.get('role') != 'doctor':
        return redirect('/login')

    username = session['username']
    doctor_info = users_table.get_item(Key={'username': username})['Item']
    specialization = doctor_info['specialization']

    appointments = appointments_table.scan(FilterExpression="doctor_name = :d", ExpressionAttributeValues={":d": username})['Items']

    total = len(appointments)
    pending = len([a for a in appointments if a['status'] == 'Pending'])
    solved = len([a for a in appointments if a['status'] == 'Solved'])

    return render_template('doctor_dashboard.html', appointments=appointments, total=total, pending=pending, solved=solved, specializatio
```

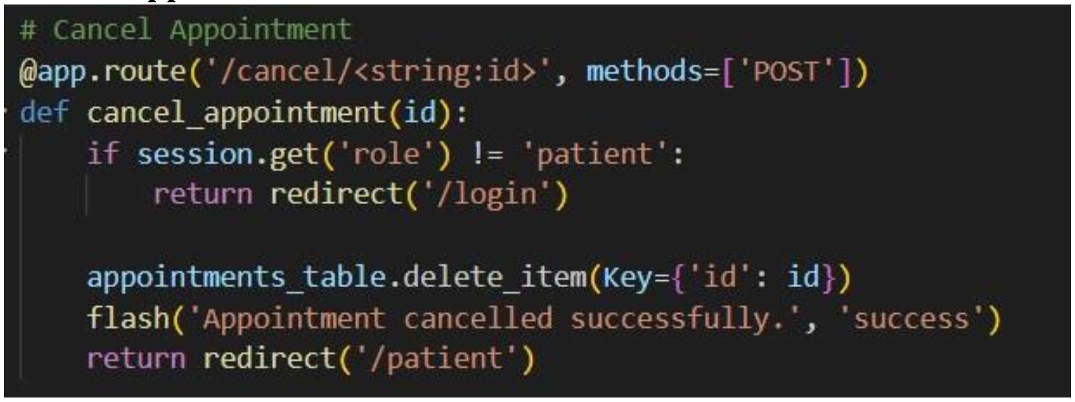
Description: Loads the doctor dashboard after login. Doctors can view pending and completed appointments based on their specialization, and respond to them with prescriptions.

- **Respond to Appointment Route:**


```python
# Respond to Appointment
@app.route('/respond/<string:id>', methods=['GET', 'POST'])
def respond(id):
    if session.get('role') != 'doctor':
        return redirect('/login')

    if request.method == 'POST':
        response_text = request.form['response']
        appointments_table.update_item(
            Key={'id': id},
            UpdateExpression="set response=:r, status='Solved'",
            ExpressionAttributeValues={':r': response_text}
        )
```

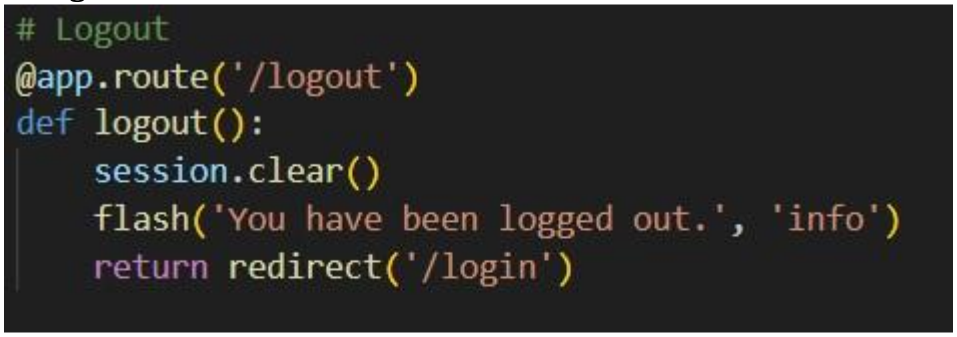
Description: Allows doctors to respond to a specific appointment by submitting their diagnosis or treatment notes. This also updates the appointment status to "Solved".

- **Cancel Appointment Route:**


```python
# Cancel Appointment
@app.route('/cancel/<string:id>', methods=['POST'])
def cancel_appointment(id):
    if session.get('role') != 'patient':
        return redirect('/login')

    appointments_table.delete_item(Key={'id': id})
    flash('Appointment cancelled successfully.', 'success')
    return redirect('/patient')
```

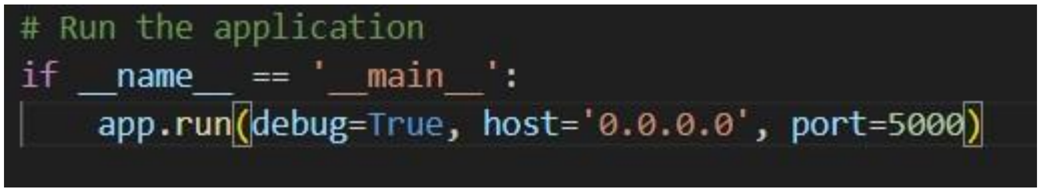
Description: Allows patients to cancel a specific appointment they previously booked. It removes the appointment record from the system.

- **Logout Route:**


```python
# Logout
@app.route('/logout')
def logout():
    session.clear()
    flash('You have been logged out.', 'info')
    return redirect('/login')
```

Description: Clears the current session and logs out the user, redirecting them to the login page with a confirmation flash message.

- **Deployment Code:**

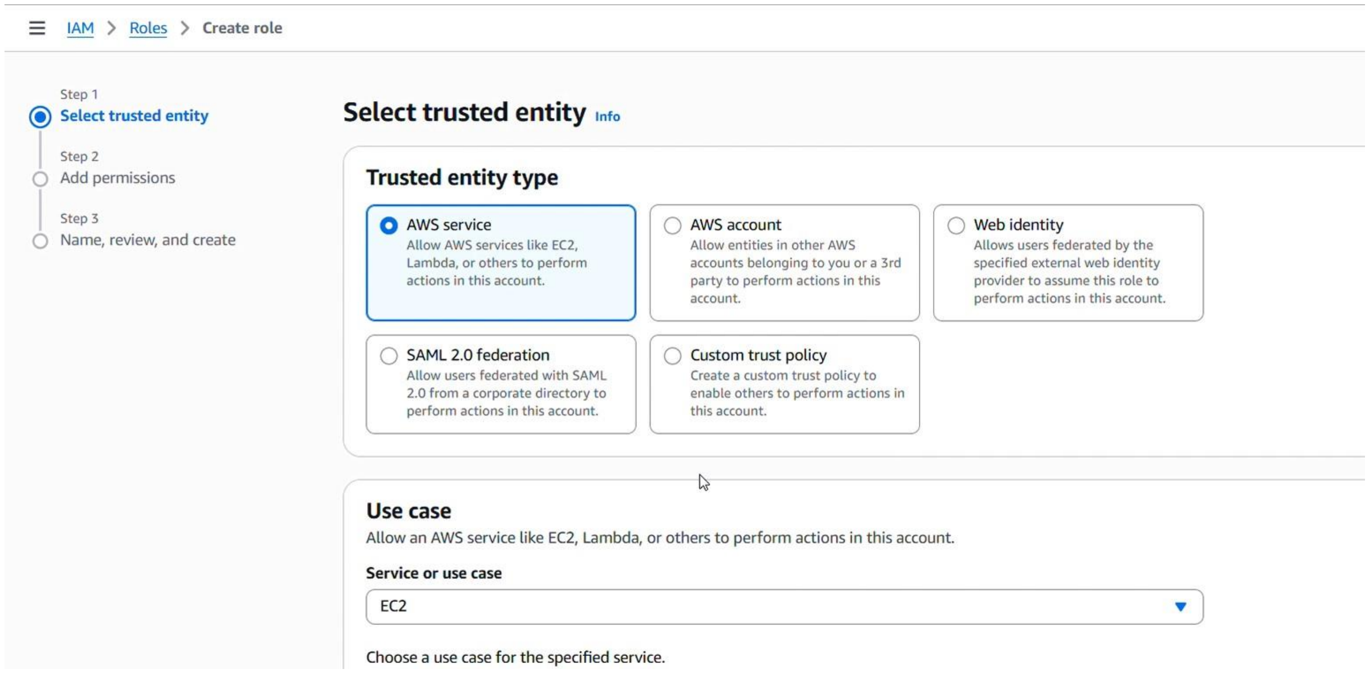


```
# Run the application
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Description: Starts the Flask app in debug mode on port 5000, accessible from any network interface (0.0.0.0). This setup is ideal for development on a remote server or local network.

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.



IAM > Roles > Create role

Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create

Select trusted entity Info

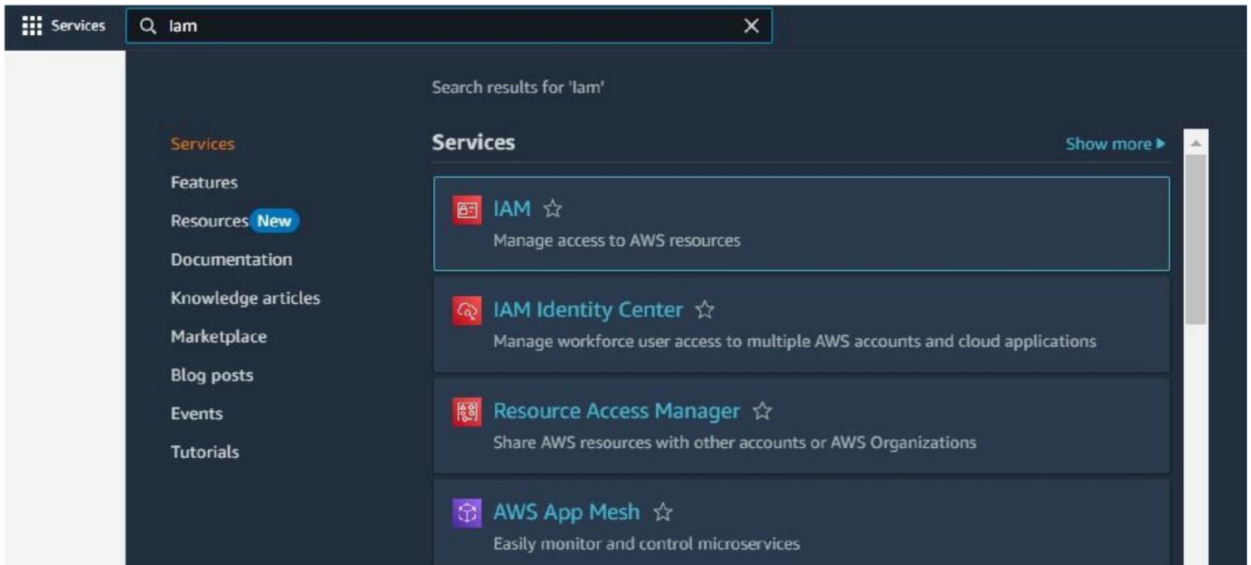
Trusted entity type

- ● AWS service — Allow AWS services like EC2, Lambda, or others to perform actions in this account.
- ○ AWS account — Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
- ○ Web identity — Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.
- ○ SAML 2.0 federation — Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.
- ○ Custom trust policy — Create a custom trust policy to enable others to perform actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case
EC2

Choose a use case for the specified service.



Services | Iam

Search results for 'Iam'

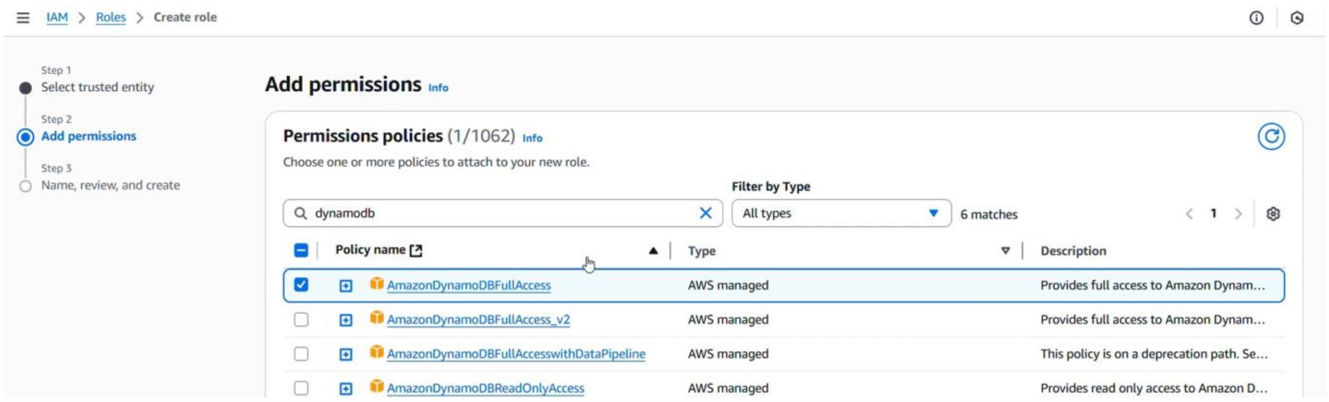
Services

- IAM — Manage access to AWS resources
- IAM Identity Center — Manage workforce user access to multiple AWS accounts and cloud applications
- Resource Access Manager — Share AWS resources with other accounts or AWS Organizations
- AWS App Mesh — Easily monitor and control microservices

- **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.



Add permissions Info

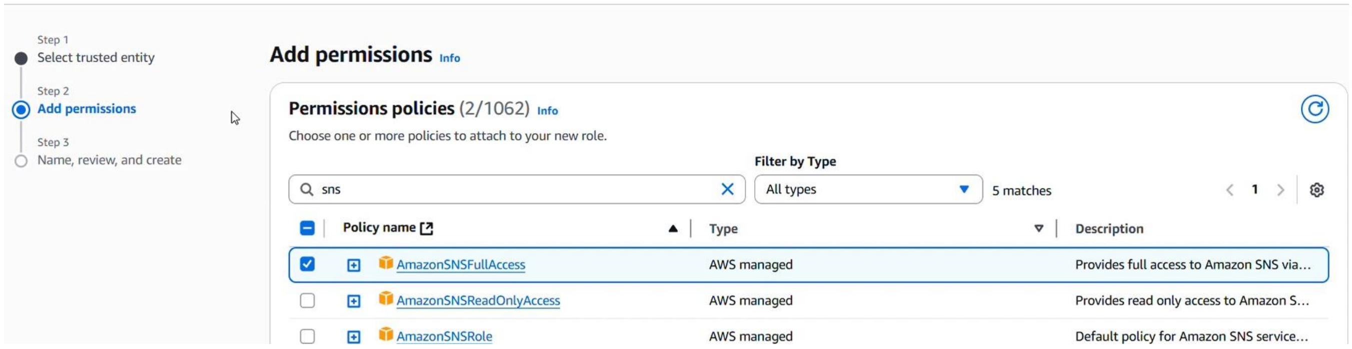
Permissions policies (1/1062) Info

Choose one or more policies to attach to your new role.

Filter by Type

dynamodb | All types | 6 matches

	Policy name	Type	Description
✓	AmazonDynamoDBFullAccess	AWS managed	Provides full access to Amazon Dynam...
	AmazonDynamoDBFullAccess_v2	AWS managed	Provides full access to Amazon Dynam...
	AmazonDynamoDBFullAccesswithDataPipeline	AWS managed	This policy is on a deprecation path. Se...
	AmazonDynamoDBReadOnlyAccess	AWS managed	Provides read only access to Amazon D...



Add permissions Info

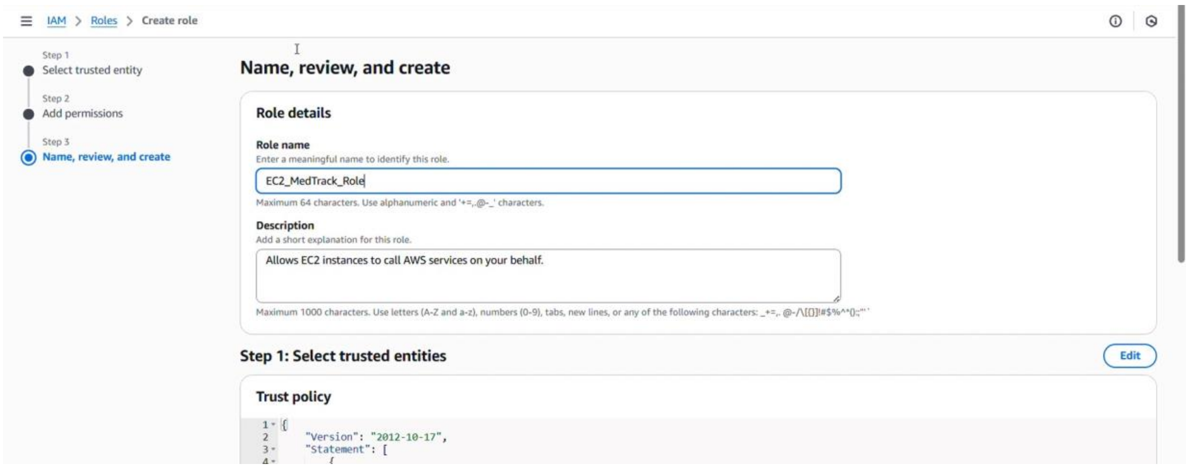
Permissions policies (2/1062) Info

Choose one or more policies to attach to your new role.

Filter by Type

sns | All types | 5 matches

	Policy name	Type	Description
✓	AmazonSNSFullAccess	AWS managed	Provides full access to Amazon SNS via...
	AmazonSNSReadOnlyAccess	AWS managed	Provides read only access to Amazon S...
	AmazonSNSRole	AWS managed	Default policy for Amazon SNS service...



Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.

EC2_MedTrack_Role

Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.

Description
Add a short explanation for this role.

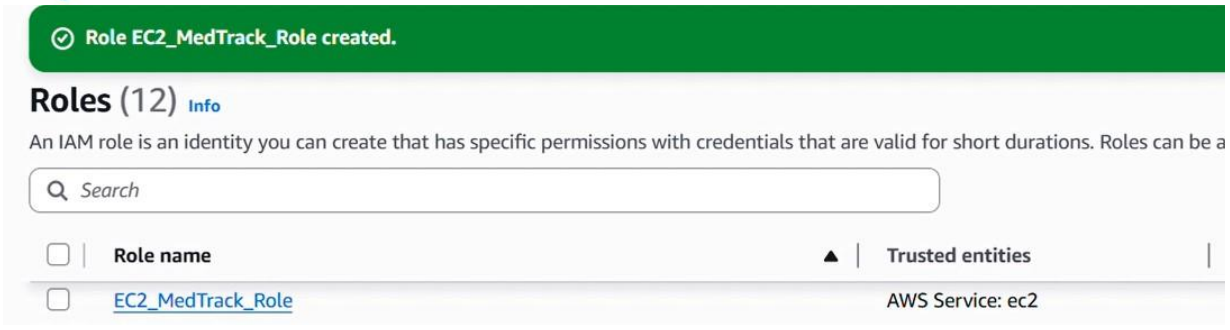
Allows EC2 instances to call AWS services on your behalf.

Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _+=,. @-/\[{}]!#$%^*();"'`

Step 1: Select trusted entities

Trust policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```


Role EC2_MedTrack_Role created.

Roles (12) Info

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be a

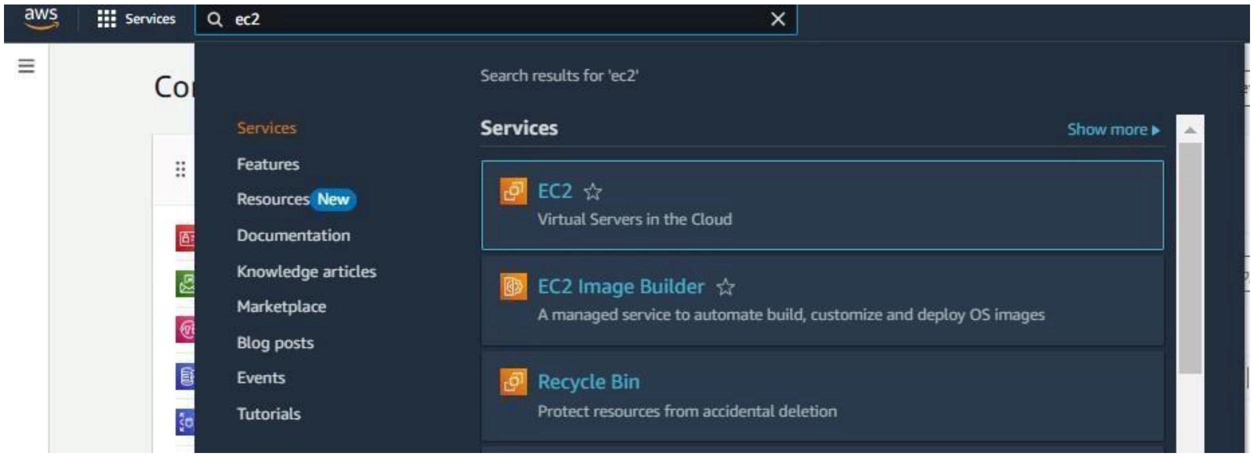
Role name	Trusted entities
EC2_MedTrack_Role	AWS Service: ec2

Milestone 6: EC2 Instance Setup

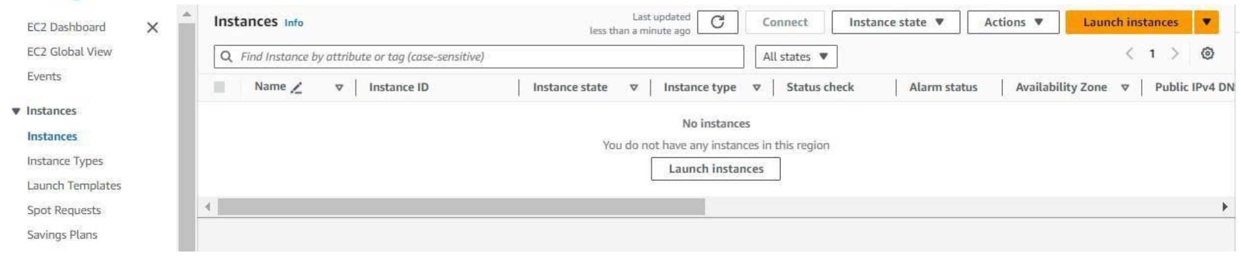
- **Note: Load your Flask app and Html files into GitHub repository**.

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

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



- Click on Launch instance to launch EC2 instance

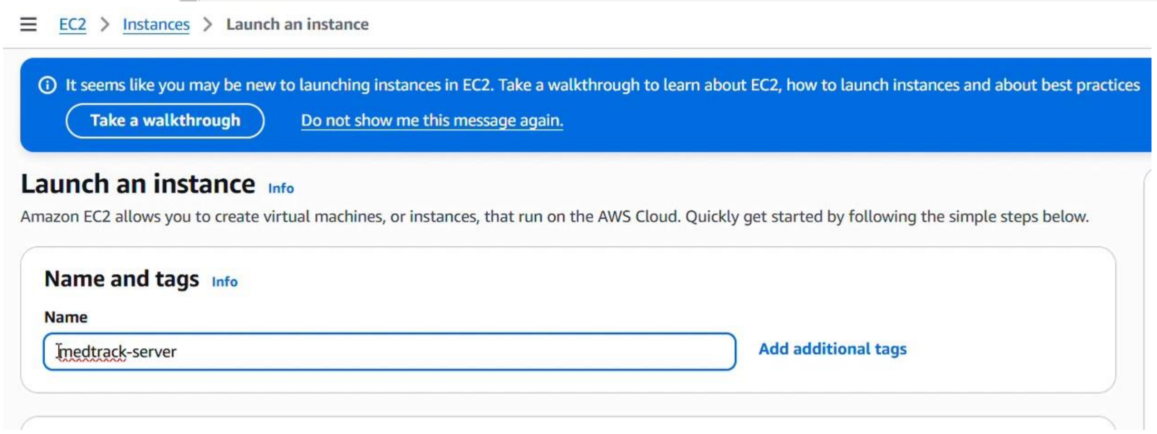


Instances Info

Last updated less than a minute ago

No instances

You do not have any instances in this region



EC2 > Instances > Launch an instance

It seems like you may be new to launching instances in EC2. Take a walkthrough to learn about EC2, how to launch instances and about best practices

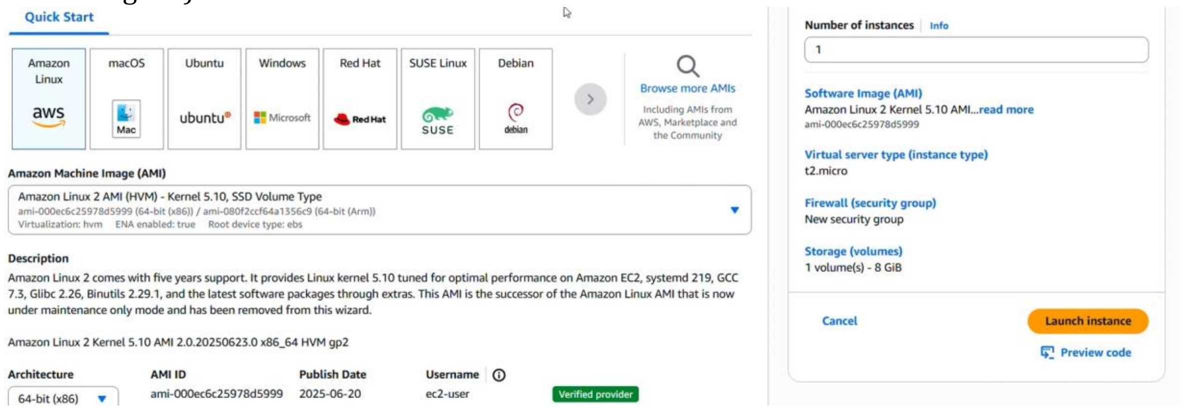
Launch an instance Info

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags Info

Name: medtrack-server

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



Amazon Machine Image (AMI)

Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type
ami-000ec6c25978d5999 (64-bit (x86)) / ami-080f2ccf64a1356c9 (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

Description

Amazon Linux 2 comes with five years support. It provides Linux kernel 5.10 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras. This AMI is the successor of the Amazon Linux AMI that is now under maintenance only mode and has been removed from this wizard.

Amazon Linux 2 Kernel 5.10 AMI 2.0.20250623.0 x86_64 HVM gp2

Architecture	AMI ID	Publish Date	Username
64-bit (x86)	ami-000ec6c25978d5999	2025-06-20	ec2-user

Number of instances Info: 1

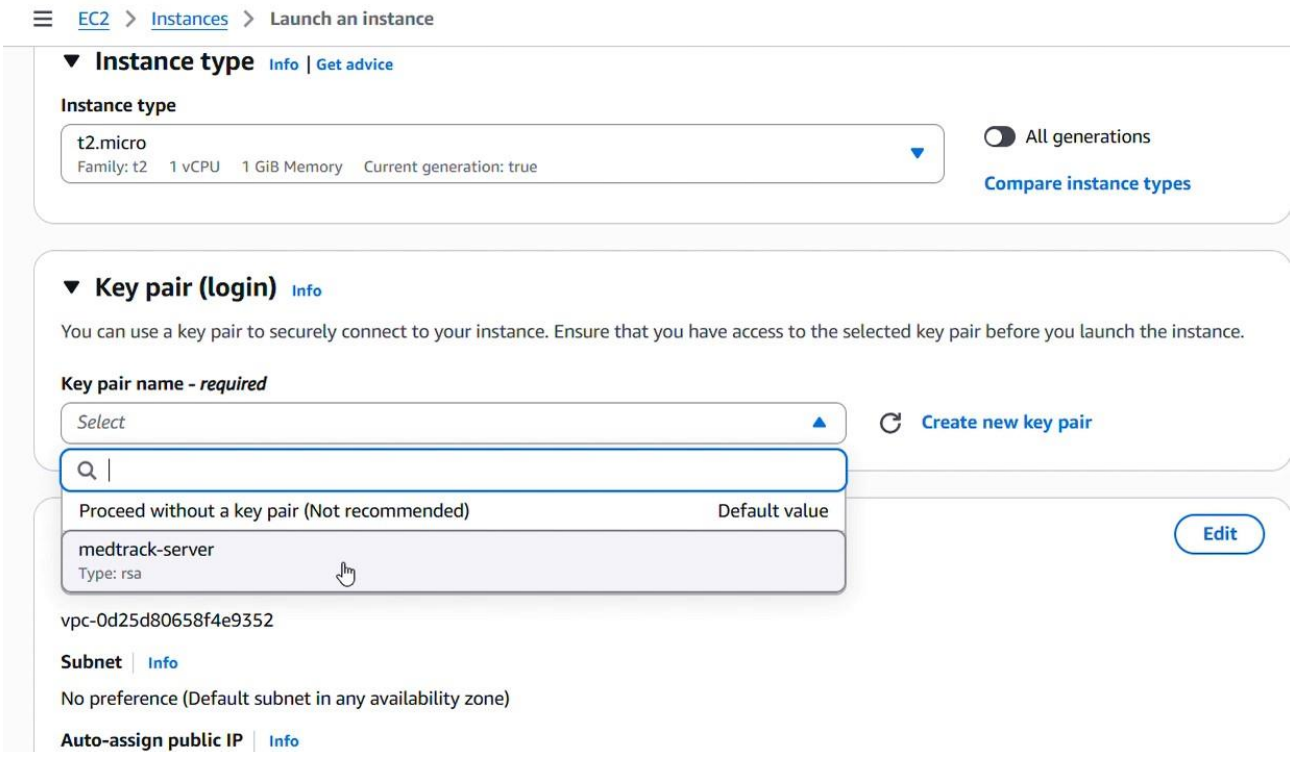
Software Image (AMI): Amazon Linux 2 Kernel 5.10 AMI...read more
ami-000ec6c25978d5999

Virtual server type (instance type): t2.micro

Firewall (security group): New security group

Storage (volumes): 1 volume(s) - 8 GiB

- Create and download the key pair for Server access.


EC2 > Instances > Launch an instance

▼ Instance type Info | Get advice

Instance type

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*

Select

Create new key pair

Proceed without a key pair (Not recommended) Default value

medtrack-server
Type: rsa

Edit

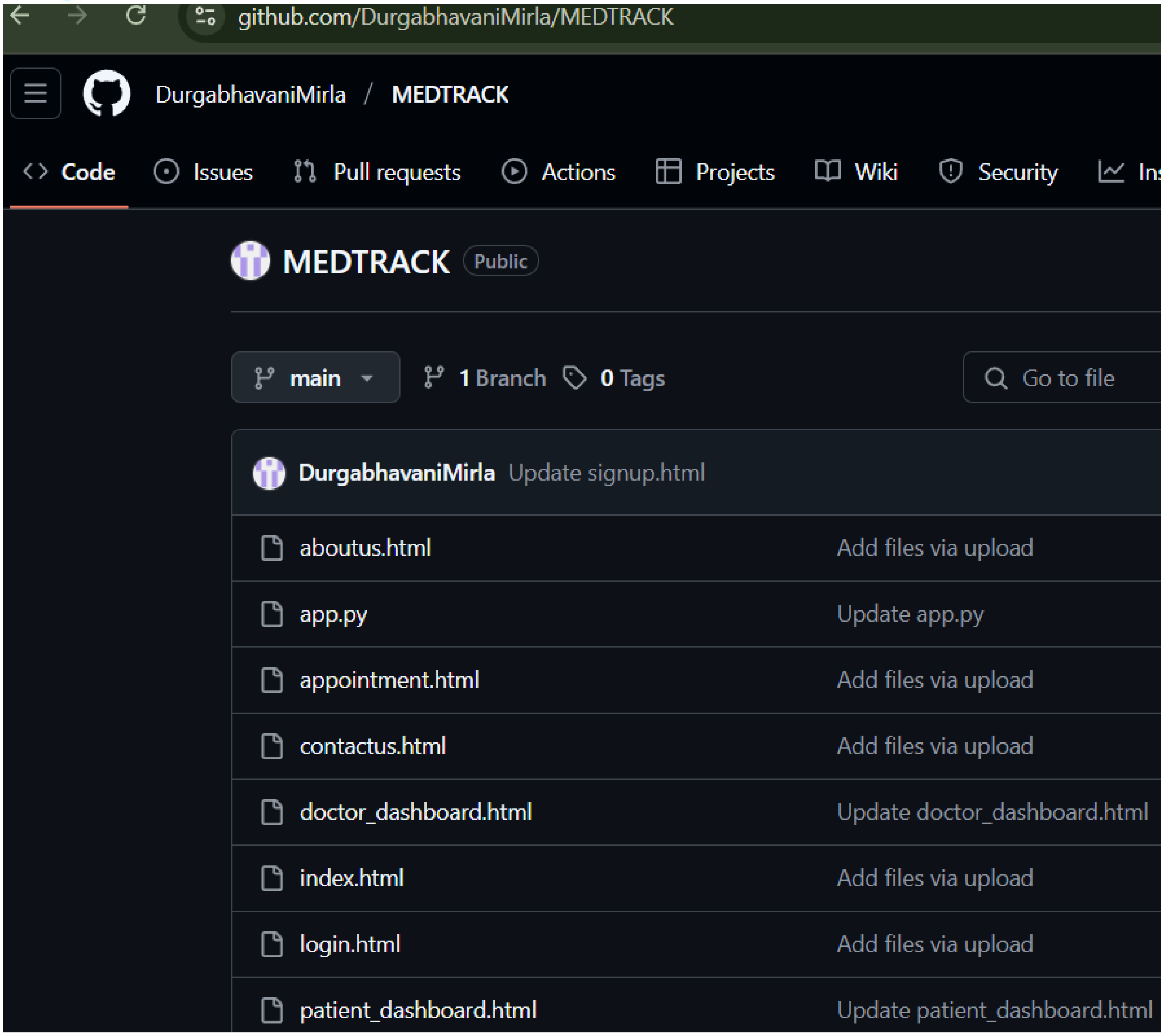
vpc-0d25d80658f4e9352

Subnet Info

No preference (Default subnet in any availability zone)

Auto-assign public IP Info

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

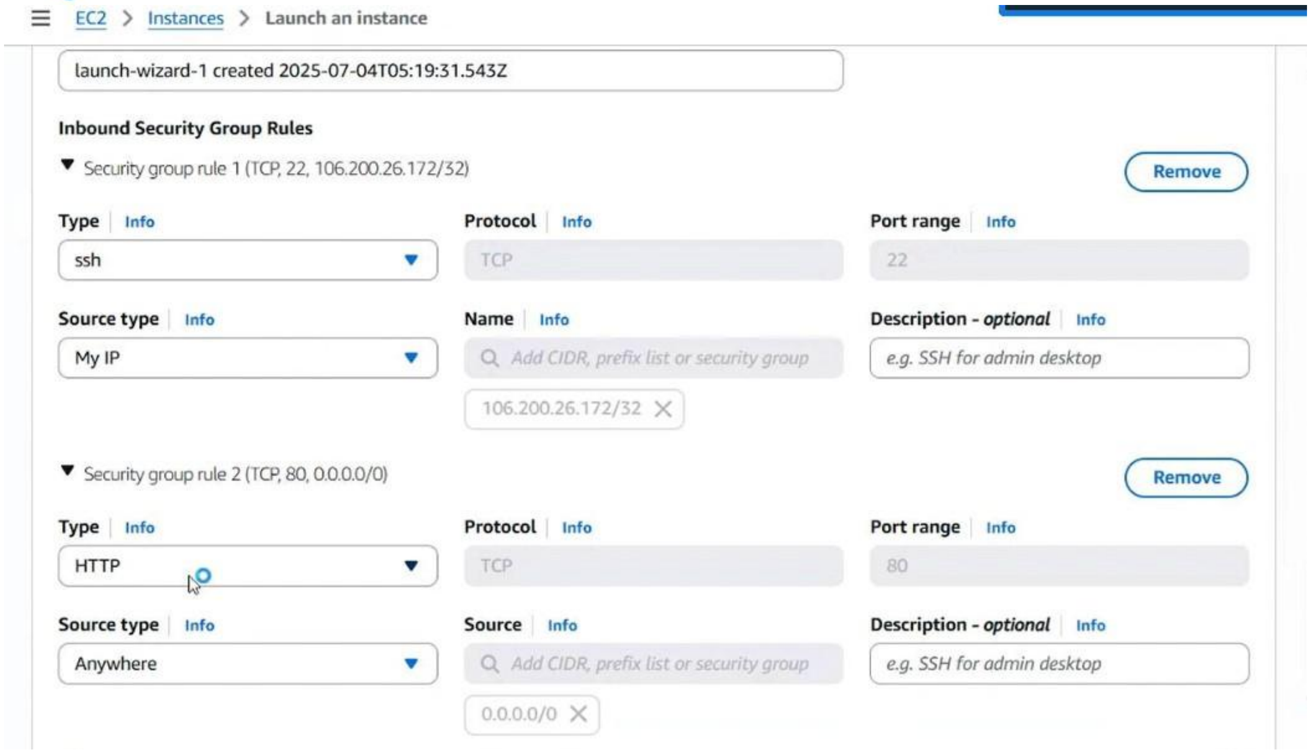

github.com/DurgabhavaniMirla/MEDTRACK

DurgabhavaniMirla / MEDTRACK

MEDTRACK Public

DurgabhavaniMirla Update signup.html

aboutus.html	Add files via upload
app.py	Update app.py
appointment.html	Add files via upload
contactus.html	Add files via upload
doctor_dashboard.html	Update doctor_dashboard.html
index.html	Add files via upload
login.html	Add files via upload
patient_dashboard.html	Update patient_dashboard.html



launch-wizard-1 created 2025-07-04T05:19:31.543Z

Inbound Security Group Rules

▼ Security group rule 1 (TCP, 22, 106.200.26.172/32)

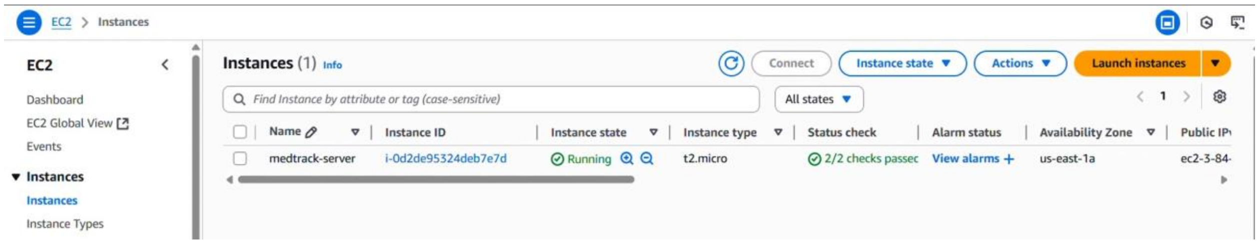
Type	Protocol	Port range
ssh	TCP	22

Source type	Name	Description - *optional*
My IP	106.200.26.172/32	*e.g. SSH for admin desktop*

▼ Security group rule 2 (TCP, 80, 0.0.0.0/0)

Type	Protocol	Port range
HTTP	TCP	80

Source type	Source	Description - *optional*
Anywhere	0.0.0.0/0	*e.g. SSH for admin desktop*

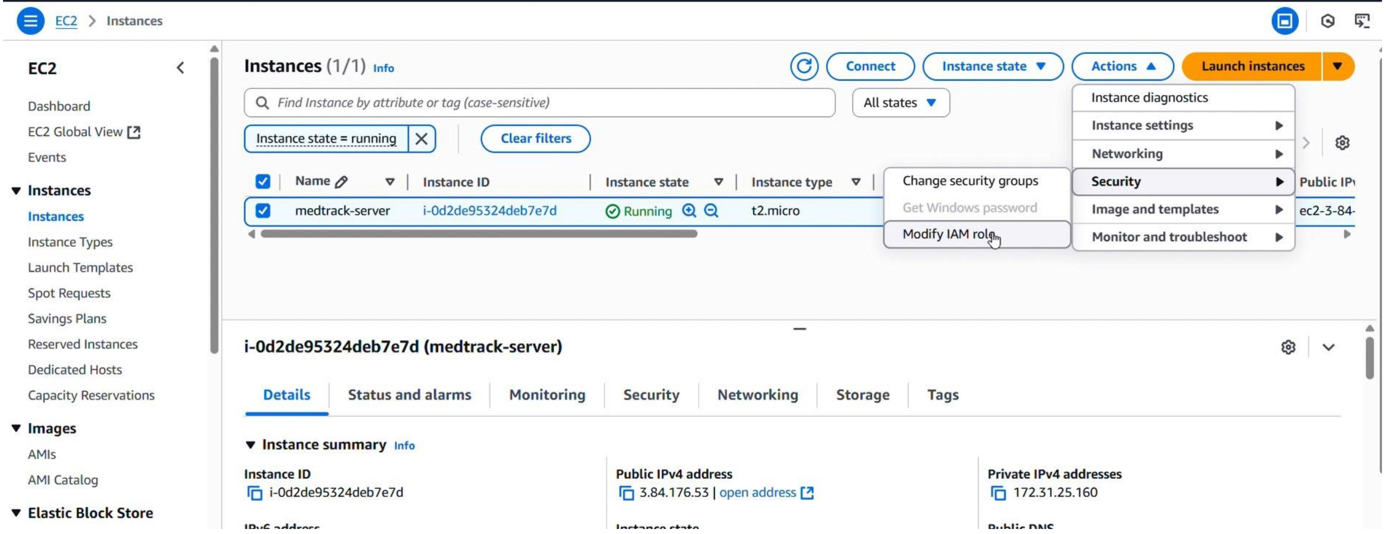


Instances (1)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IP
medtrack-server	i-0d2de95324deb7e7d	Running	t2.micro	2/2 checks passec	View alarms +	us-east-1a	ec2-3-84-

- 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.

Instance summary for i-001861022fbcac290 (InstantLibraryApp)
Updated less than a minute ago



Name	Instance ID	Instance state	Instance type	Public IPv
medtrack-server	i-0d2de95324deb7e7d	Running	t2.micro	ec2-3-84-

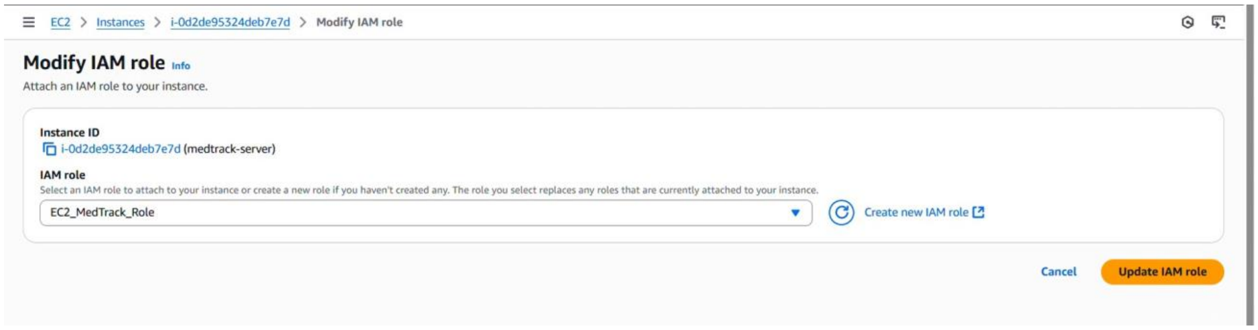
i-0d2de95324deb7e7d (medtrack-server)

Instance summary

Instance ID: i-0d2de95324deb7e7d

Public IPv4 address: 3.84.176.53 | open address

Private IPv4 addresses: 172.31.25.160

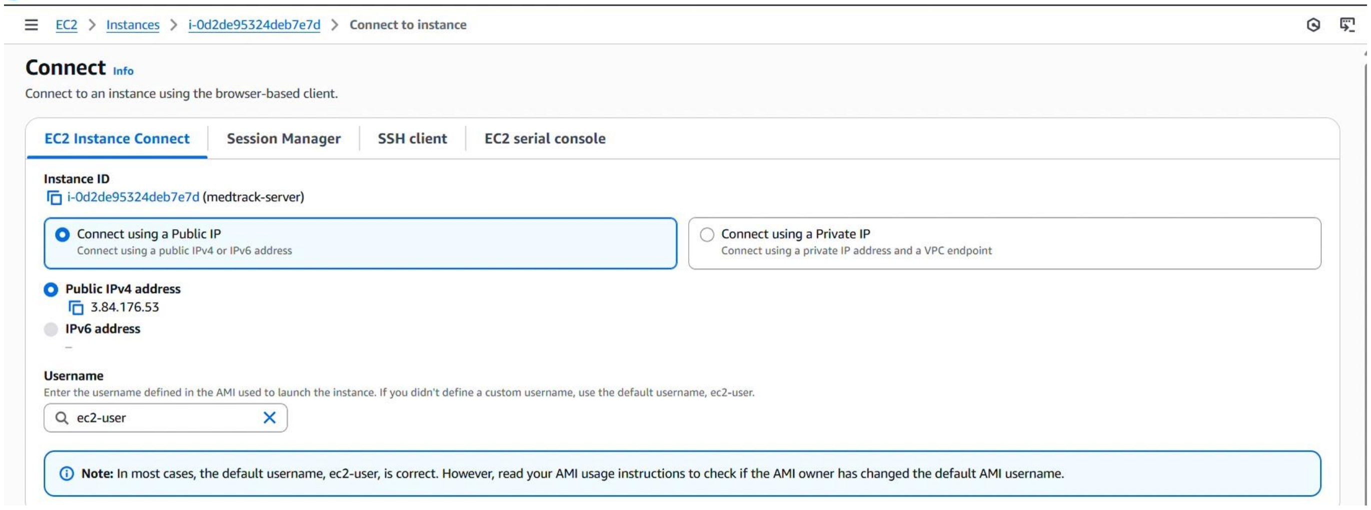

EC2 > Instances > i-0d2de95324deb7e7d > Modify IAM role
Modify IAM role Info
Attach an IAM role to your instance.
Create new IAM role
Cancel
Update IAM role

Instance ID: i-0d2de95324deb7e7d (medtrack-server)

IAM role: Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

EC2_MedTrack_Role

- Now connect the EC2 with the files



EC2 > Instances > i-0d2de95324deb7e7d > Connect to instance

Connect Info

Connect to an instance using the browser-based client.

EC2 Instance Connect | Session Manager | SSH client | EC2 serial console

Instance ID
i-0d2de95324deb7e7d (medtrack-server)

(●) Connect using a Public IP
Connect using a public IPv4 or IPv6 address

( ) Connect using a Private IP
Connect using a private IP address and a VPC endpoint

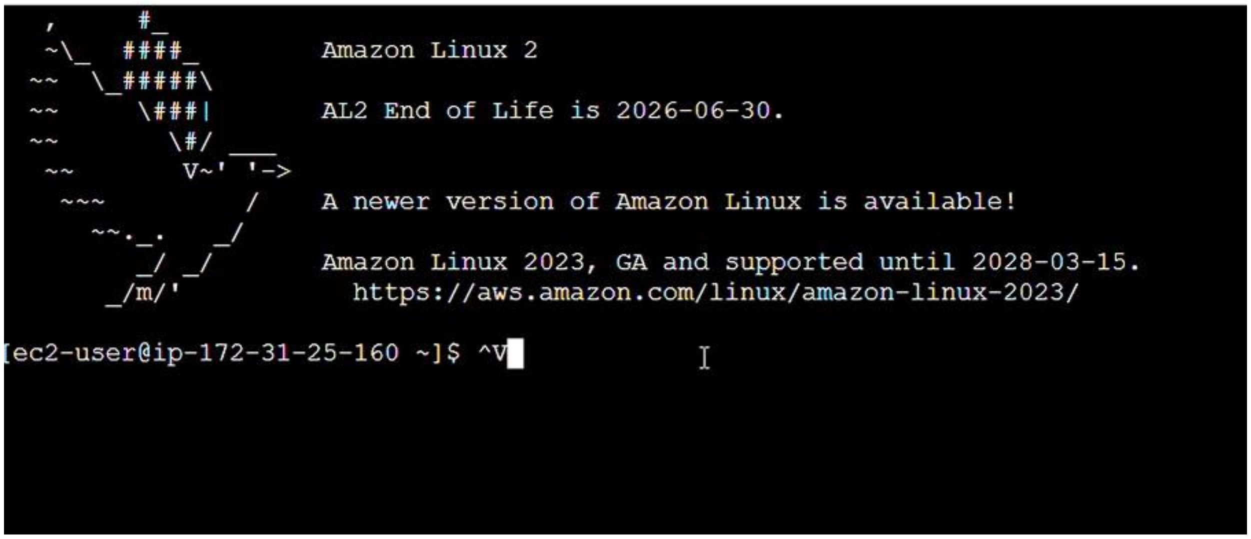
(●) **Public IPv4 address**
3.84.176.53

( ) **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.



```
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2026-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-172-31-25-160 ~]$ ^V
```

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.

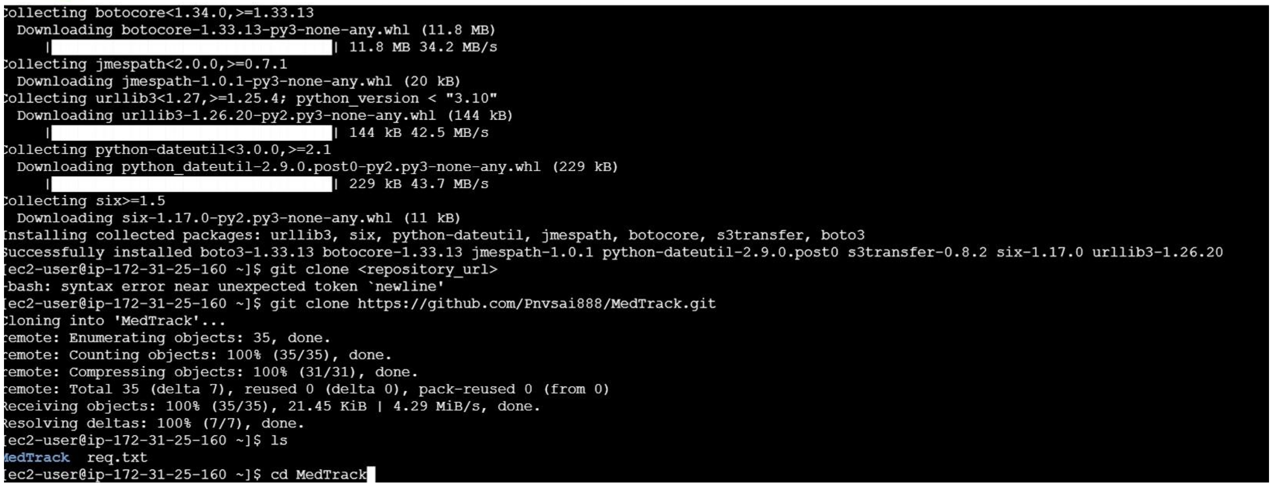
Run: 'git clone https://github.com/DurgabhavaniMirla/MEDTRACK.git'

- 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=5000



```
Collecting botocore<1.34.0,>=1.33.13
  Downloading botocore-1.33.13-py3-none-any.whl (11.8 MB)
     |████████████████████████████████| 11.8 MB 34.2 MB/s
Collecting jmespath<2.0.0,>=0.7.1
  Downloading jmespath-1.0.1-py3-none-any.whl (20 kB)
Collecting urllib3<1.27,>=1.25.4; python_version < "3.10"
  Downloading urllib3-1.26.20-py2.py3-none-any.whl (144 kB)
     |████████████████████████████████| 144 kB 42.5 MB/s
Collecting python-dateutil<3.0.0,>=2.1
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
     |████████████████████████████████| 229 kB 43.7 MB/s
Collecting six>=1.5
  Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: urllib3, six, python-dateutil, jmespath, botocore, s3transfer, boto3
Successfully installed boto3-1.33.13 botocore-1.33.13 jmespath-1.0.1 python-dateutil-2.9.0.post0 s3transfer-0.8.2 six-1.17.0 urllib3-1.26.20
[ec2-user@ip-172-31-25-160 ~]$ git clone <repository_url>
-bash: syntax error near unexpected token `newline'
[ec2-user@ip-172-31-25-160 ~]$ git clone https://github.com/Pnvsai888/MedTrack.git
Cloning into 'MedTrack'...
remote: Enumerating objects: 35, done.
remote: Counting objects: 100% (35/35), done.
remote: Compressing objects: 100% (31/31), done.
remote: Total 35 (delta 7), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (35/35), 21.45 KiB | 4.29 MiB/s, done.
Resolving deltas: 100% (7/7), done.
[ec2-user@ip-172-31-25-160 ~]$ ls
MedTrack  req.txt
[ec2-user@ip-172-31-25-160 ~]$ cd MedTrack
```

Verify the Flask app is running:

http://3.84.176.53:5000

○ Run the Flask app on the EC2 instance

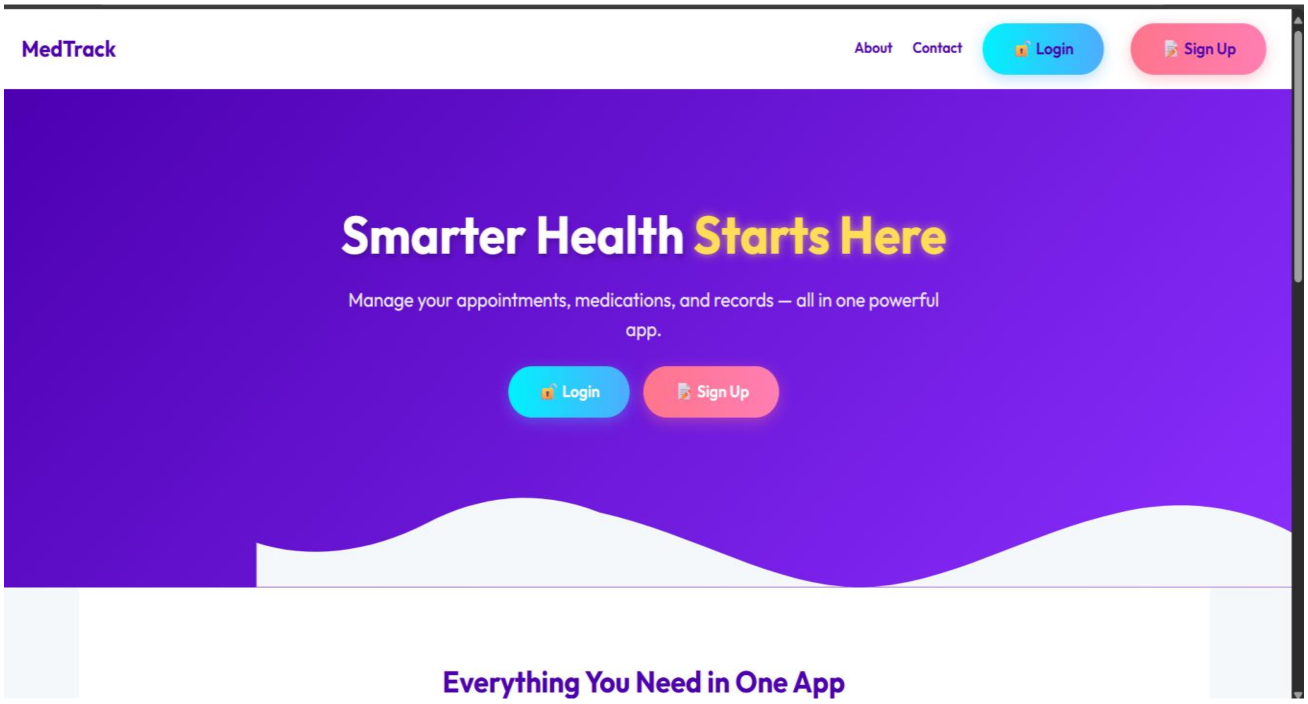
Access the website through:

PublicIPs: http://3.84.176.53:5000

Milestone 8: Testing and Deployment

- **Activity 8.1: Conduct functional testing to verify user sign-up, login, appointment booking, prescriptionand SNS notifications.**

Index Page:



MedTrack

About Contact Login Sign Up

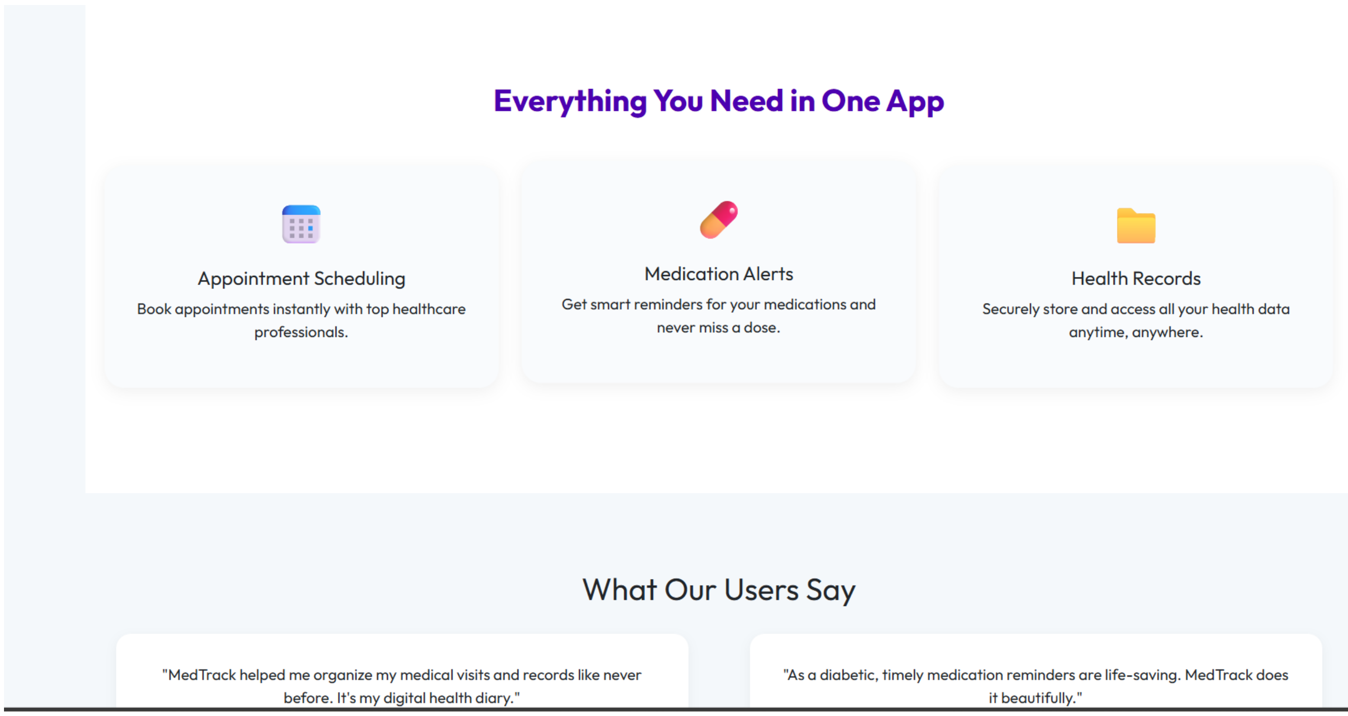
Smarter Health Starts Here

Manage your appointments, medications, and records — all in one powerful app.

Login Sign Up

Everything You Need in One App

Register Page:



Everything You Need in One App

Appointment Scheduling

Book appointments instantly with top healthcare professionals.

Medication Alerts

Get smart reminders for your medications and never miss a dose.

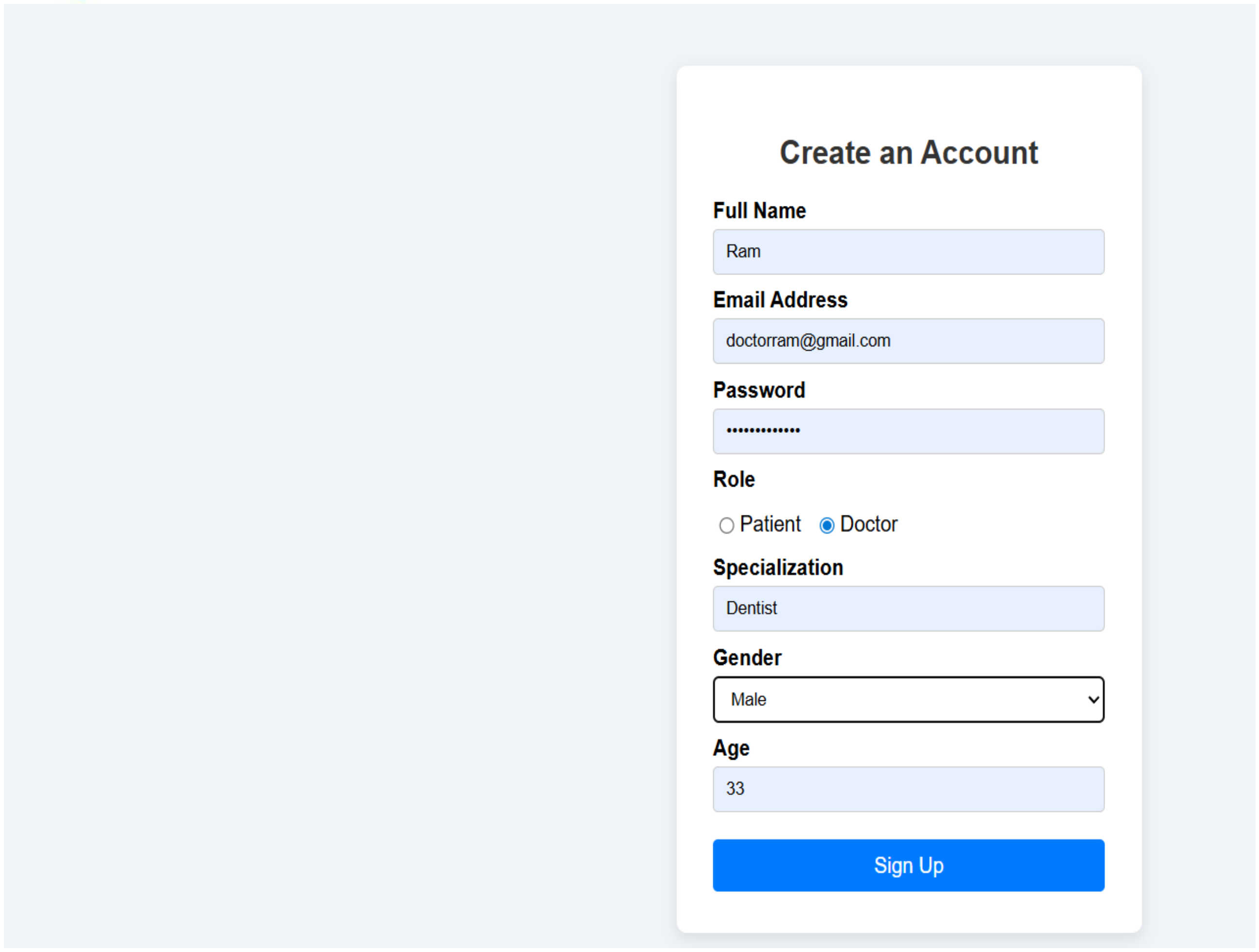
Health Records

Securely store and access all your health data anytime, anywhere.

What Our Users Say

"MedTrack helped me organize my medical visits and records like never before. It's my digital health diary."

"As a diabetic, timely medication reminders are life-saving. MedTrack does it beautifully."



Create an Account

Full Name

Ram

Email Address

doctorram@gmail.com

Password

•••••••••••••

Role

Patient [✓ Doctor]

Specialization

Dentist

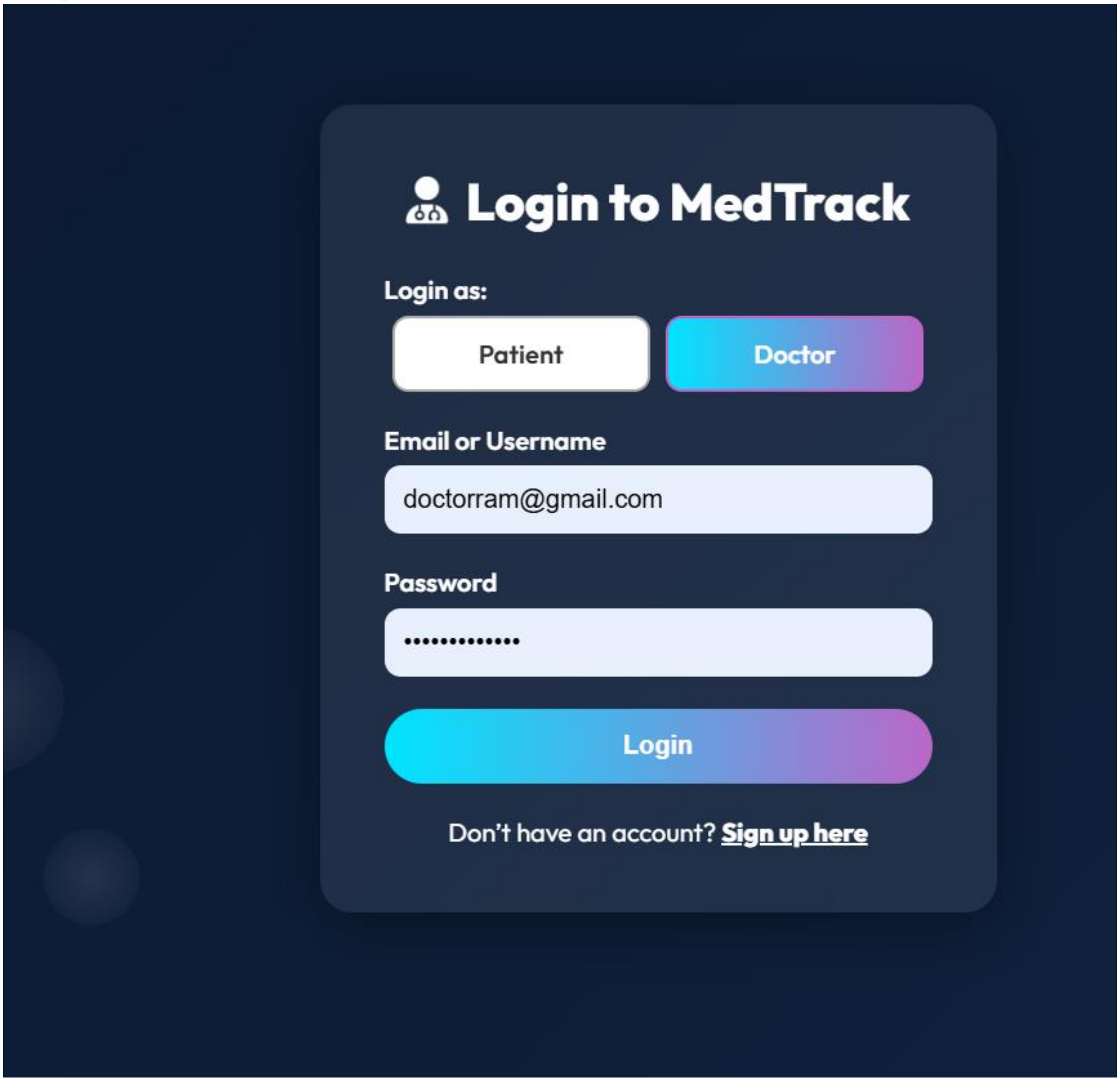
Gender

Male

Age

33

Login Page:

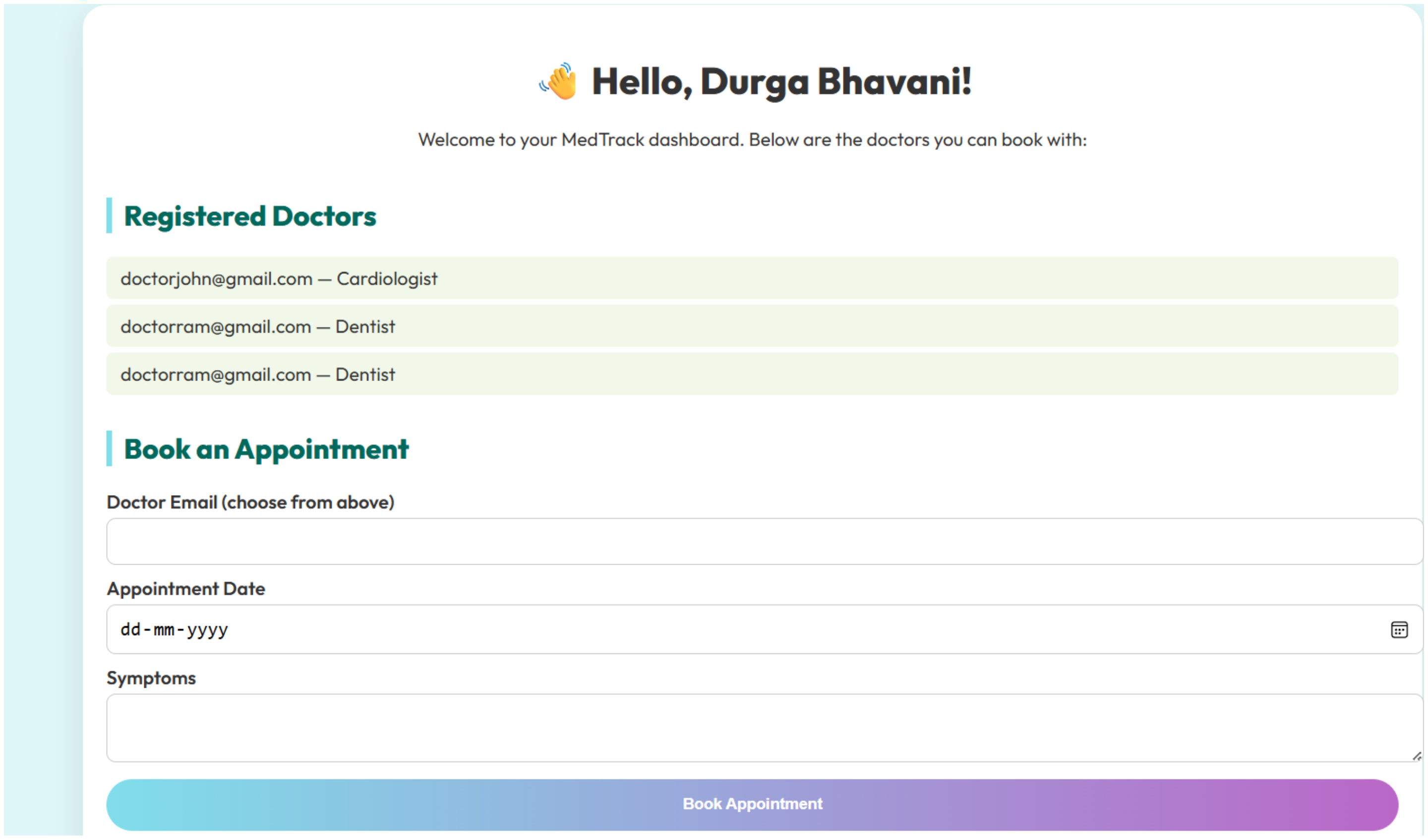

Login to MedTrack

Login as: Patient [✓ Doctor]

Email or Username: doctorram@gmail.com

Password: ••••••••••••••

Patient DashBoard:



👋 Hello, Durga Bhavani!

Welcome to your MedTrack dashboard. Below are the doctors you can book with:

Registered Doctors

doctorjohn@gmail.com — Cardiologist

doctorram@gmail.com — Dentist

doctorram@gmail.com — Dentist

Book an Appointment

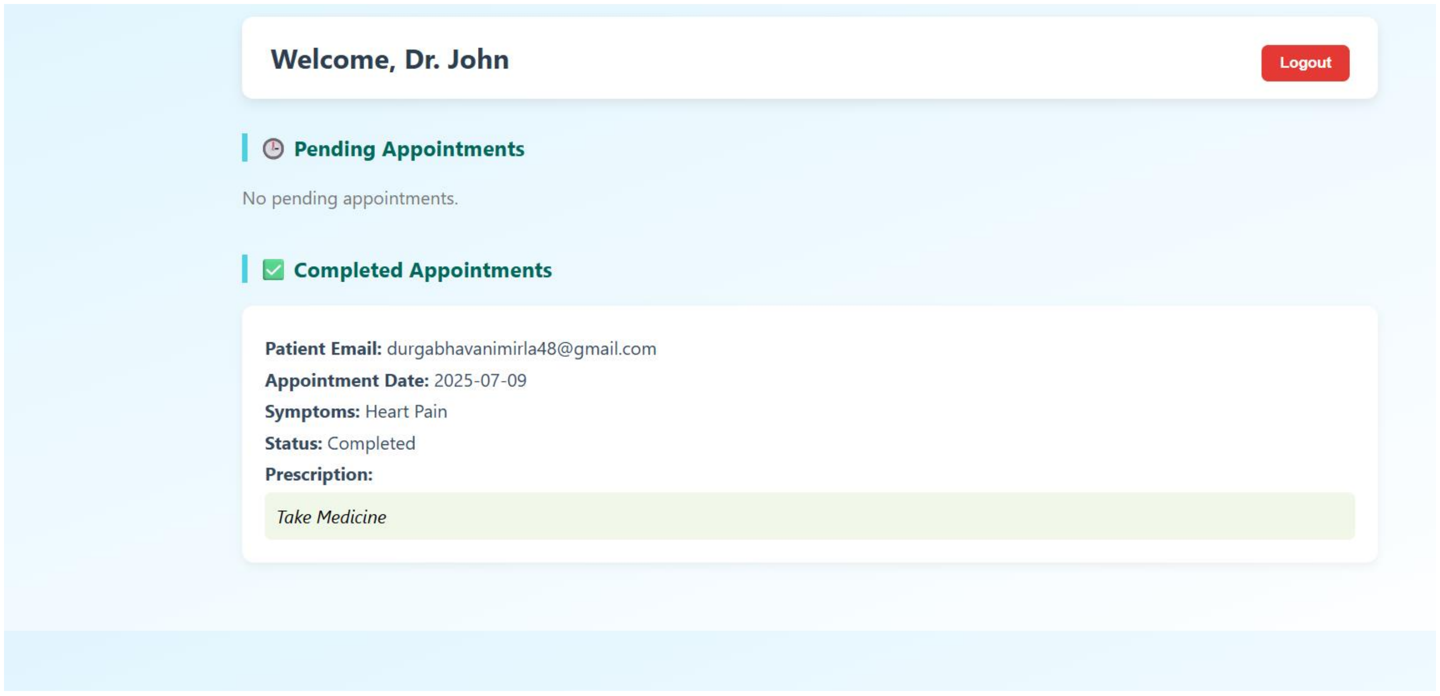
Doctor Email (choose from above)

Appointment Date

dd-mm-yyyy

Symptoms

Doctor DashBoard:



Welcome, Dr. John

Pending Appointments

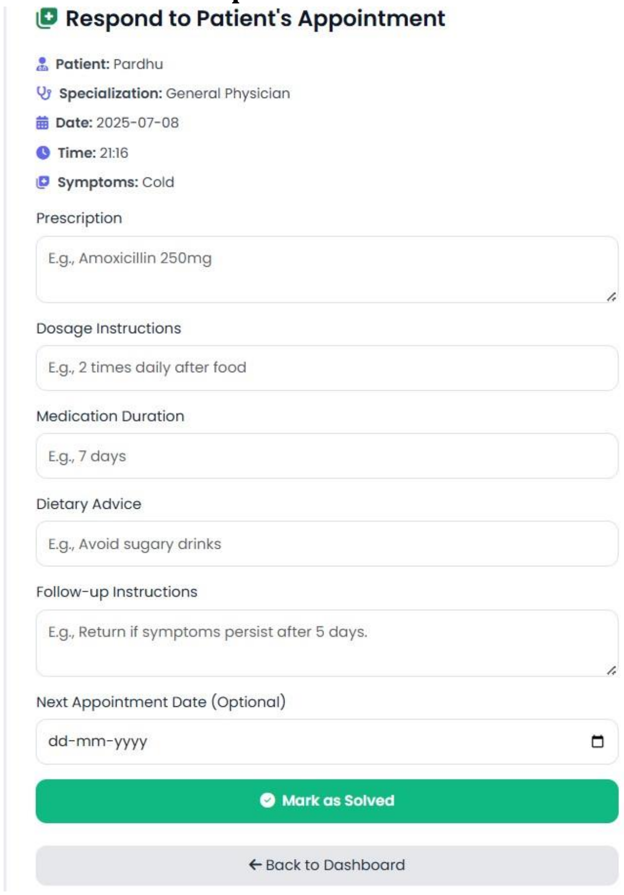
No pending appointments.

Completed Appointments

Patient Email: durgabhavanimirla48@gmail.com
Appointment Date: 2025-07-09
Symptoms: Heart Pain
Status: Completed
Prescription:

Take Medicine

Response:


Respond to Patient's Appointment

Patient: Pardhu

Specialization: General Physician

Date: 2025-07-08

Time: 21:16

Symptoms: Cold

Prescription

E.g., Amoxicillin 250mg

Dosage Instructions

E.g., 2 times daily after food

Medication Duration

E.g., 7 days

Dietary Advice

E.g., Avoid sugary drinks

Follow-up Instructions

E.g., Return if symptoms persist after 5 days.

Next Appointment Date (Optional)

dd-mm-yyyy

Conclusion:

MedTrack represents a significant step forward in modernizing healthcare management through a cloud-powered infrastructure. By integrating AWS services such as EC2, DynamoDB, SNS, and IAM with a Flask-based backend, the platform delivers a secure, scalable, and responsive environment for both patients and doctors.

The system successfully overcomes key healthcare challenges by enabling seamless appointment scheduling, prescription tracking, and timely medication reminders — all within a unified interface. Doctors benefit from centralized tools to manage appointments, issue prescriptions, and oversee patient progress, ultimately enhancing the quality of care.

Extensive testing has validated the platform's core features, ensuring robust performance in user authentication, data handling, and real-time notifications.

In essence, MedTrack stands as a powerful example of how cloud technologies can transform healthcare delivery. It not only streamlines doctor-patient interactions but also empowers users to take control of their health in a secure and efficient manner.