

MedTrack: AWS Cloud-Enabled Healthcare Management System

Project Description:

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

Scenario 1: Efficient Appointment Booking System for Patients

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

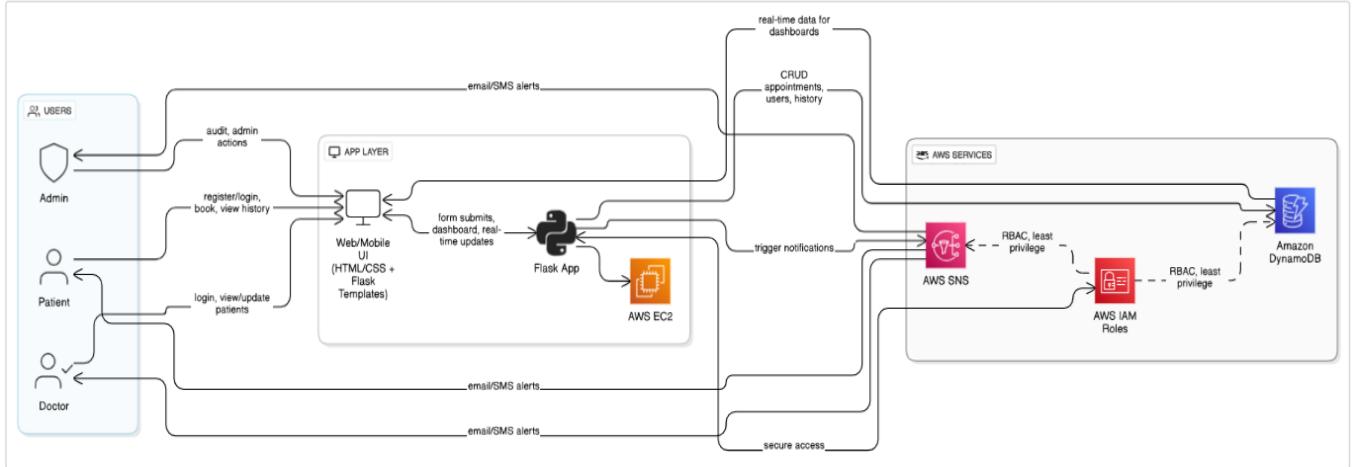
Scenario 2: Secure User Management with IAM

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

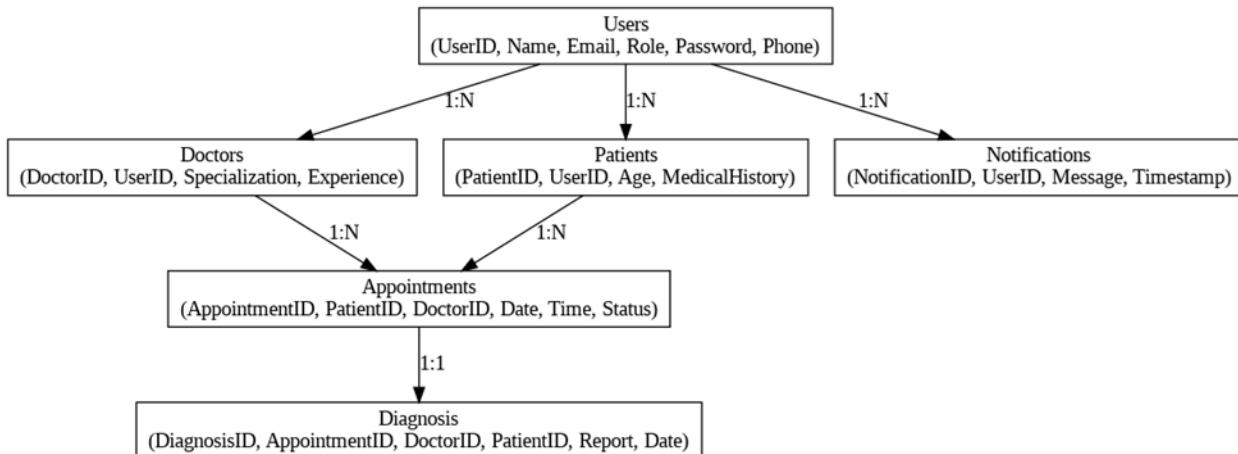
Scenario 3: Easy Access to Medical History and Resources

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

AWS ARCHITECTURE:



Entity Relationship (ER) Diagram:



Pre-requisites:

1. **AWS Account Setup:** [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

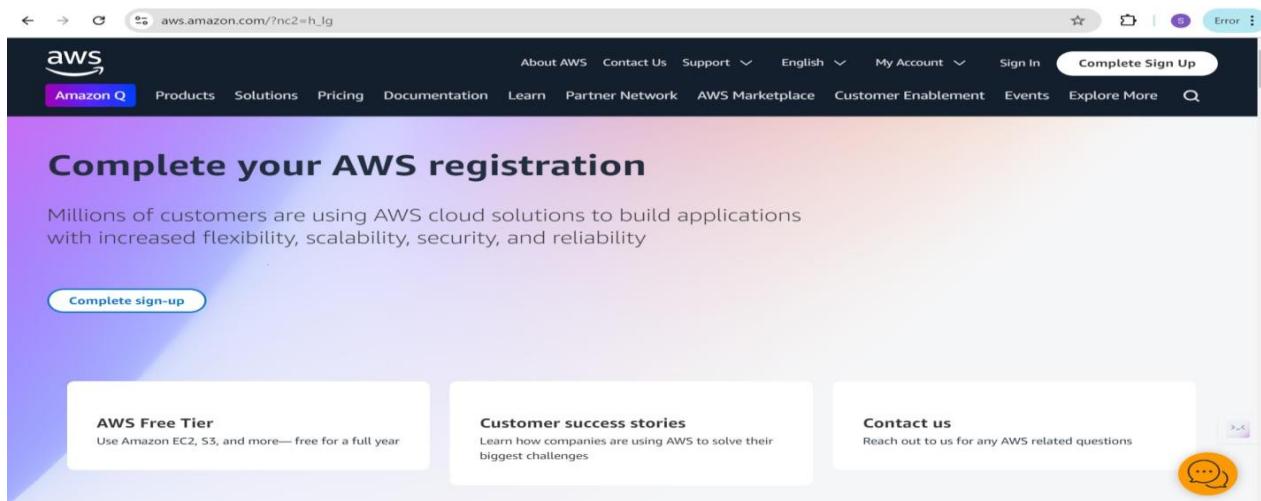
Activity 7.2: Run the Flask App

8. Testing and Deployment

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

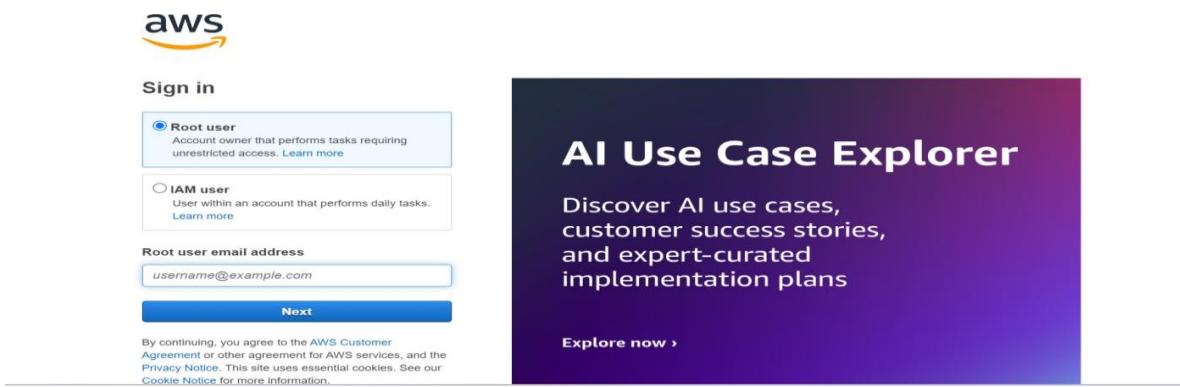
Milestone 1: AWS Account Setup and Login

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



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

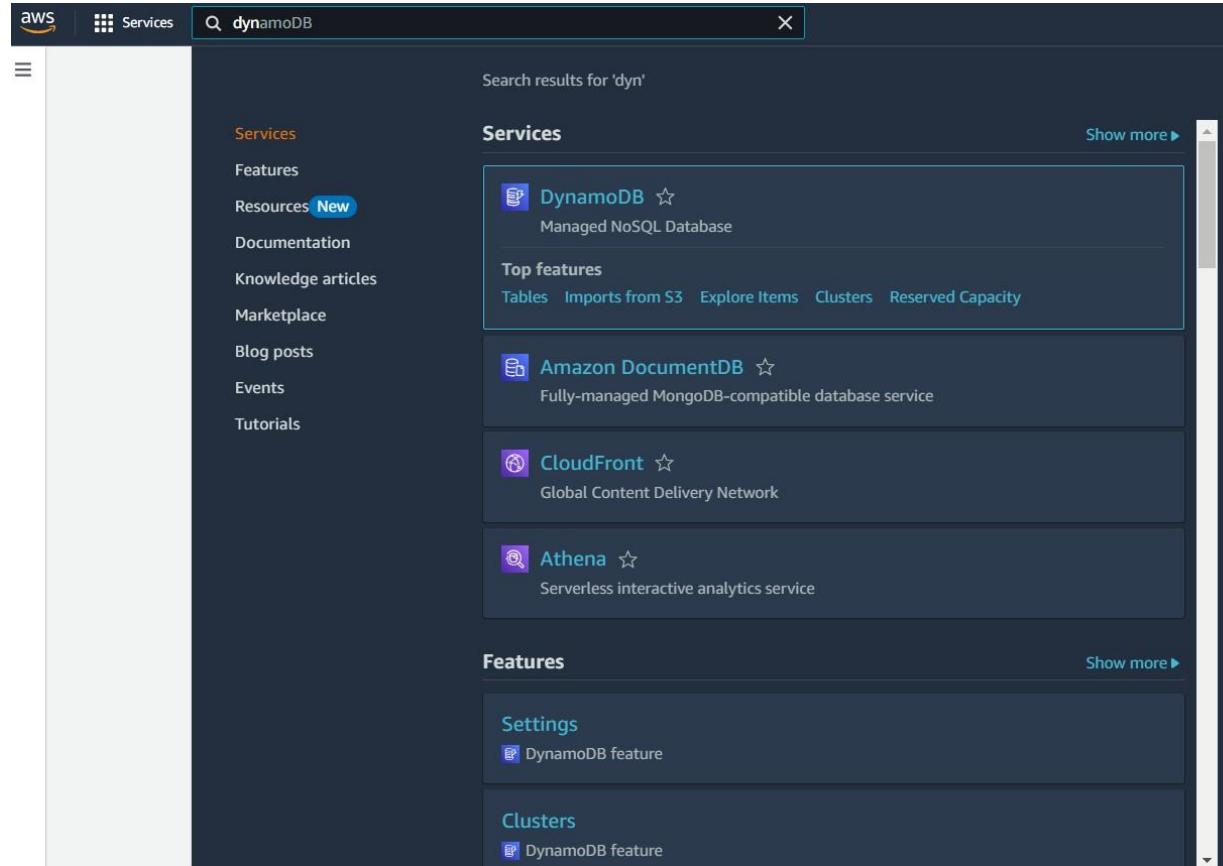
- After setting up your account, log in to the [AWS Management Console](#).



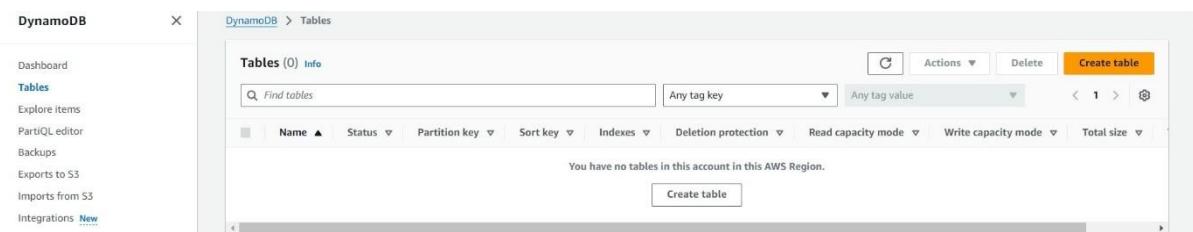
Milestone 2: DynamoDB Database Creation and Setup

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

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



The screenshot shows the AWS Services search results page. The search bar at the top contains 'dynamoDB'. On the left, there is a sidebar with links to Services, Features, Resources (New), Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials. The main content area displays a list of services under the heading 'Services'. The first item in the list is 'DynamoDB' (Managed NoSQL Database). Below it are 'Amazon DocumentDB' (Fully-managed MongoDB-compatible database service), 'CloudFront' (Global Content Delivery Network), and 'Athena' (Serverless interactive analytics service). At the bottom, there are sections for 'Features' (Settings, Clusters) and 'Show more ▶' buttons.



The screenshot shows the 'Tables' page within the DynamoDB service. The left sidebar has links for Dashboard, Tables (which is selected and highlighted in blue), Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, and Integrations (New). The main content area shows a table titled 'Tables (0) Info'. It includes a search bar ('Find tables'), filters ('Any tag key', 'Any tag value'), and sorting options ('Name', 'Status', 'Partition key', 'Sort key', 'Indexes', 'Deletion protection', 'Read capacity mode', 'Write capacity mode', 'Total size'). A message at the bottom states 'You have no tables in this account in this AWS Region.' There is a 'Create table' button at the bottom right.

● Activity 2.2: Create a DynamoDB table for storing data

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

☰ [DynamoDB](#) > [Tables](#) > [Create table](#)

Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

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.

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.

1 to 255 characters and case sensitive.

Table settings

Default settings

The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these

Customize settings

Use these advanced features to make DynamoDB work better for your needs.

☰ [DynamoDB](#) > [Tables](#) > [Create table](#)



Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	AWS owned key	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.

(i) This table will be created with auto scaling deactivated. You do not have permissions to turn on auto scaling.

Cancel

Create table

DynamoDB > Tables

Share your feedback on Amazon DynamoDB
 Your feedback is an important part of helping us provide a better customer experience. Take this short survey to let us know how we're doing.

The MedTrackUsers table was created successfully.

Tables (1) Info

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity mode
MedTrackUsers	Active	email (\$)	-	0	0	Off		On-demand

- Follow the same steps to create a remaining table with different partition key

DynamoDB > Tables

The MedTrackUsers1 table was created successfully.

Tables (7) Info

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity mode
MedTrackAppointments	Active	appointment_id (\$)	-	0	0	Off		On-demand
MedTrackDiagnosis	Active	diagnosis_id (\$)	-	0	0	Off		On-demand
MedTrackDoctors	Active	email (\$)	-	0	0	Off		On-demand
MedTrackNotifications	Active	notification_id (\$)	-	0	0	Off		On-demand
MedTrackPatients	Active	email (\$)	-	0	0	Off		On-demand
MedTrackUsers	Active	email (\$)	-	0	0	Off		On-demand

Milestone 3: SNS Notification Setup

- Activity 3.1: Create SNS topics to send email notifications to patients about their upcoming appointments.**

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

Q sns X

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
 Route 53 feature

Amazon SNS X

 **New Feature**
Amazon SNS now supports in-place message archiving and replay for FIFO topics. [Learn more](#) ↗

Dashboard

Topics

Subscriptions

▼ Mobile

Push notifications

Text messaging (SMS)

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

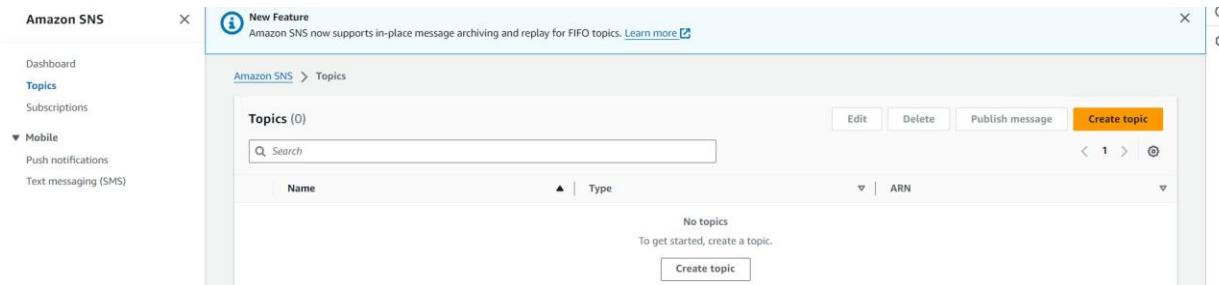
A topic is a message channel. When you publish a message to a topic, it fans out the message to all subscribed endpoints.

Next step

Start with an overview

Pricing

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



The screenshot shows the Amazon SNS Topics page. On the left, there is a sidebar with options like Dashboard, Topics, Subscriptions, Mobile, Push notifications, and Text messaging (SMS). The main area shows a table titled 'Topics (0)' with columns for Name, Type, and ARN. A search bar is at the top of the table. At the bottom right of the table, there is a 'Create topic' button. A blue banner at the top of the page says 'New Feature' and mentions that Amazon SNS now supports in-place message archiving and replay for FIFO topics.

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

☰ [Amazon SNS](#) > [Topics](#) > [Create topic](#)

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

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.

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**

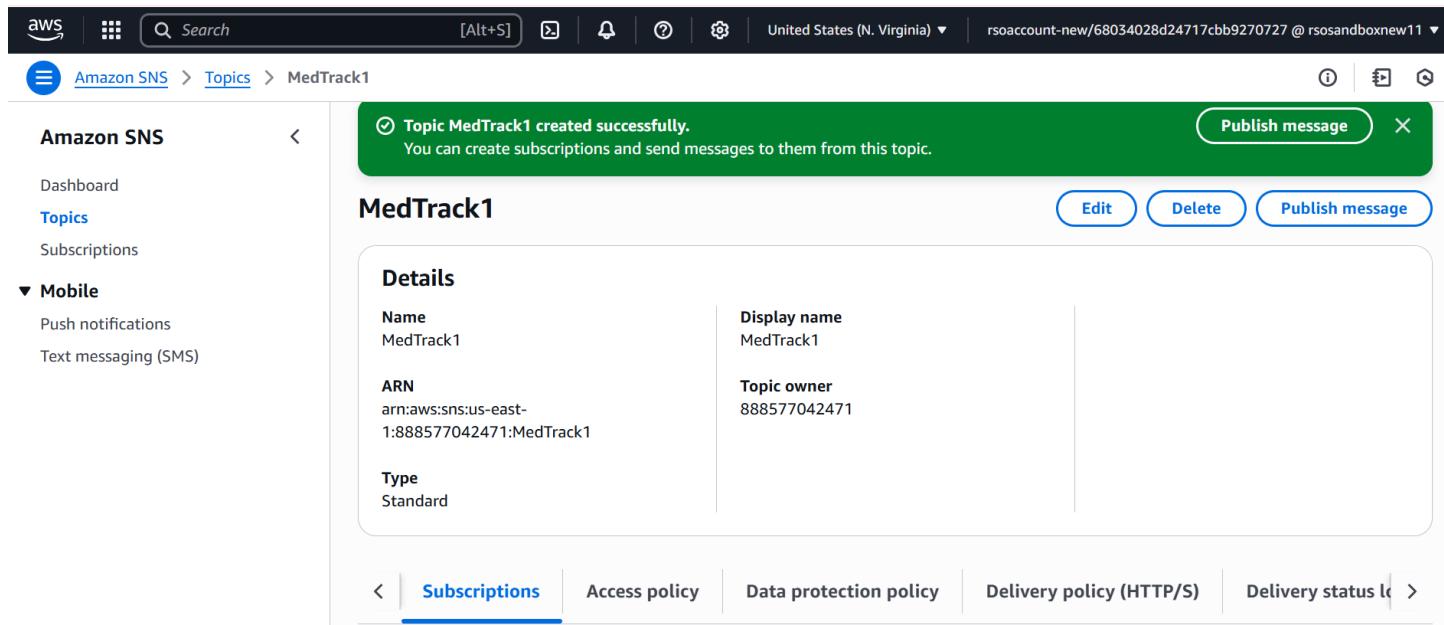
A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. [Learn more](#)

► **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.

[Cancel](#) [Create topic](#)

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



The screenshot shows the AWS SNS Topics page. A new topic, "MedTrack1", has been created successfully. The top navigation bar includes the AWS logo, a search bar, and various icons. The left sidebar shows links for Dashboard, Topics (which is selected), Subscriptions, and Mobile (Push notifications and Text messaging (SMS)). The main content area displays the "MedTrack1" topic details: Name (MedTrack1), Display name (MedTrack1), ARN (arn:aws:sns:us-east-1:888577042471:MedTrack1), Topic owner (888577042471), and Type (Standard). Below the details, there are tabs for Subscriptions, Access policy, Data protection policy, Delivery policy (HTTP/S), and Delivery status log. A green success message at the top states: "Topic MedTrack1 created successfully. You can create subscriptions and send messages to them from this topic." There are also "Edit", "Delete", and "Publish message" buttons.

- **Activity 3.2: Subscribe users (or admin staff) to this topic via email. When a book request is made, notifications will be sent to the subscribed emails.**
 - Subscribe users (or admin staff) to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails.

Amazon SNS > Subscriptions > Create subscription

Create subscription

Details

Topic ARN X

Protocol
The type of endpoint to subscribe

Select protocol ▼

i After your subscription is created, you must confirm it. [Info](#)

► **Subscription filter policy - optional** [Info](#)

This policy filters the messages that a subscriber receives.

Amazon SNS > Topics > MedTrack1 > Subscription: 950ce8bb-b681-40b8-81f4-1a171bb5fb3c

i **Subscription to MedTrack1 created successfully.**
 The ARN of the subscription is arn:aws:sns:us-east-1:888577042471:MedTrack1:950ce8bb-b681-40b8-81f4-1a171bb5fb3c.

Subscription: 950ce8bb-b681-40b8-81f4-1a171bb5fb3c Edit Delete

Details	
<p>ARN <code>arn:aws:sns:us-east-1:888577042471:MedTrack1:950ce8bb-b681-40b8-81f4-1a171bb5fb3c</code></p> <p>Endpoint <code>keerthianakapalli@gmail.com</code></p> <p>Topic MedTrack1</p> <p>Subscription Principal <code>arn:aws:iam::888577042471:role/rsoaccount-new</code></p>	<p>Status i Pending confirmation</p> <p>Protocol EMAIL</p>

- After subscription request for the mail confirmation

Subscription: 71eb046e-12a1-4016-828c-9cf3f0cf39c3	
Details	
ARN	Status
arn:aws:sns:us-east-1:888577042471:MedTrack2:71eb046e-12a1-4016-828c-9cf3f0cf39c3	 Confirmed
Endpoint	Protocol
keerthianakapalli@gmail.com	EMAIL

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

You have chosen to subscribe to the topic:

arn:aws:sns:us-east-1:888577042471:MedTrack1

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

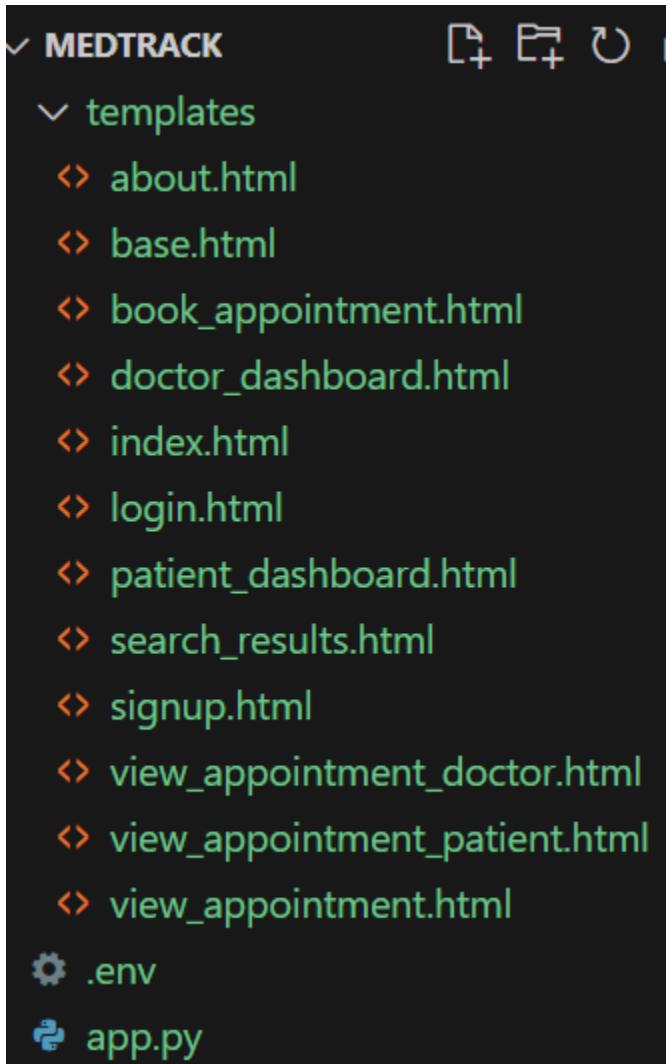
[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)

- 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



The screenshot shows a file explorer window with the following directory structure:

- ✓ MEDTRACK [copy] [move] [refresh] [close]
 - ✓ templates [copy] [move] [refresh] [close]
 - ↳ about.html
 - ↳ base.html
 - ↳ book_appointment.html
 - ↳ doctor_dashboard.html
 - ↳ index.html
 - ↳ login.html
 - ↳ patient_dashboard.html
 - ↳ search_results.html
 - ↳ signup.html
 - ↳ view_appointment_doctor.html
 - ↳ view_appointment_patient.html
 - ↳ view_appointment.html
 - ⚙ .env
 - 🐍 app.py

Description: set up the Med track project with an app.py file, a static/ folder for assets, and a templates/ directory containing all required HTML pages like base, login, signup, patient dashboard, doctor dashboard, book appointment etc.

Description of the code:

- **Flask App Initialization:**

In the Med Track project, the Flask app is initialized to establish the backend infrastructure, enabling it to handle multiple user interactions such as patient registration, appointment booking, and submission of medical reports. The Flask framework processes incoming requests, communicates with the DynamoDB database for storing user data, and integrates seamlessly with AWS services. Additionally, the routes and APIs are defined to manage different functionalities like secure login, appointment scheduling, and medical history retrieval. This initialization sets up the foundation for smooth, real-time communication between patients and doctors while ensuring the app is scalable and secure.

```
from flask import Flask, request, jsonify, session, render_template, redirect, url_for, flash
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime, timedelta
from dotenv import load_dotenv
import boto3
import logging
import os
import uuid
from functools import wraps
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

# -----
# Load Environment Variables
# -----
if not load_dotenv():
    print("Warning: .env file not found. Using default configurations.")

logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)
```

Description: Import essential libraries including Flask utilities for routing, session management, and template rendering; Boto3 for interacting with AWS services like DynamoDB and SNS; smtplib and email modules for sending email notifications; uuid for generating unique identifiers; dotenv for loading environment variables; and Werkzeug for secure password hashing and verification.

```
# Flask App Initialization
# -----
app = Flask(__name__)
```

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

- **Dynamodb Setup:**

```
USERS_TABLE_NAME = os.environ.get('USERS_TABLE_NAME', 'MedTrackUsers')
DOCTORS_TABLE_NAME = os.environ.get('DOCTORS_TABLE_NAME', 'MedTrackDoctors')
PATIENTS_TABLE_NAME = os.environ.get('PATIENTS_TABLE_NAME', 'MedTrackPatients')
APPOINTMENTS_TABLE_NAME = os.environ.get('APPOINTMENTS_TABLE_NAME', 'MedTrackAppointments')
DIAGNOSIS_TABLE_NAME = os.environ.get('DIAGNOSIS_TABLE_NAME', 'MedTrackDiagnosis')
NOTIFICATIONS_TABLE_NAME = os.environ.get('NOTIFICATIONS_TABLE_NAME', 'MedTrackNotifications')
```

Description: Initialize the DynamoDB resource in the ap-south-1 region and set up access to the Users, Doctors, Patients, Appointments, Diagnosis, and Notifications tables for storing and managing user details, doctor information, patient data, appointment bookings, medical diagnoses, and notification records in the MedTrack system

- **SNS Connection**

```
# Email Configuration
SMTP_SERVER = os.environ.get('SMTP_SERVER', 'smtp.gmail.com')
SMTP_PORT = int(os.environ.get('SMTP_PORT', 587))
SENDER_EMAIL = os.environ.get('SENDER_EMAIL')
SENDER_PASSWORD = os.environ.get('SENDER_PASSWORD')
ENABLE_EMAIL = os.environ.get('ENABLE_EMAIL', 'False').lower() == 'true'
```

```
# SNS Configuration
SNS_TOPIC_ARN = os.environ.get('SNS_TOPIC_ARN')
ENABLE_SNS = os.environ.get('ENABLE_SNS', 'False').lower() == 'true'
```

```
def send_email_notification(to_email, subject, body):
    if not ENABLE_EMAIL or not SENDER_EMAIL:
        logger.info(f"Email notification would be sent: {subject}")
        return True

    try:
        msg = MIME Multipart()
        msg[ 'From' ] = SENDER_EMAIL
        msg[ 'To' ] = to_email
        msg[ 'Subject' ] = subject
        msg.attach(MIMEText(body, 'plain'))

        server = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
        server.starttls()
        server.login(SENDER_EMAIL, SENDER_PASSWORD)
        server.send_message(msg)
        server.quit()

        logger.info(f"Email sent successfully to {to_email}")
        return True
    except Exception as e:
        logger.error(f"Failed to send email: {e}")
        return False
```

Description: Configure Amazon SNS to send notifications (e.g., for appointments or medication alerts). Paste your stored SNS Topic ARN in the sns_topic_arn variable and specify the region name where the topic is created (e.g., ap-south-1). Set the email service provider in SMTP_SERVER (e.g., Gmail or Outlook), and provide the sender's subscribed email address in SENDER_EMAIL. Generate an app-specific password for the email account and store it in SENDER_PASSWORD to enable secure authentication for sending emails via SMTP.

- **Routes for Web Pages**

- **Index Route:**

```
# Route: Select role
@app.route('/')
def index():
    return render_template('index.html')
```

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

- **Signup Route:**

```
@app.route('/signup/<role>', methods=['GET', 'POST'])
def signup(role):
    if role not in ('patient', 'doctor'):
        flash('Invalid role selected.', 'danger')
        return redirect(url_for('index'))

    if request.method == 'POST':
        name = request.form.get('name')
        email = request.form.get('email')
        password = request.form.get('password')

        if not name or not email or not password:
            flash('All fields are required.', 'warning')
            return render_template('signup.html', role=role)

        # Check if user already exists
        if dynamodb:
            user_table = get_user_table()
            response = user_table.get_item(Key={'email': email})
            if 'Item' in response:
                flash('User already exists.', 'danger')
                return render_template('signup.html', role=role)
```

```
user_id = str(uuid.uuid4())
hashed_password = generate_password_hash(password)
user_data = {
    'user_id': user_id,
    'name': name,
    'email': email,
    'password_hash': hashed_password,
    'role': role,
    'created_at': datetime.now().isoformat(),
    'is_active': True
}

if dynamodb:
    user_table.put_item(Item=user_data)
else:
    local_db['users'][email] = user_data

flash('Signup successful! Please log in.', 'success')
return redirect(url_for('login', role=role))

return render_template('signup.html', role=role)
```

Description: define the /signup/<role> route to handle registration for both patients and doctors. It validates user input, checks if the user already exists in the DynamoDB Users table (or local DB fallback), securely hashes the password using Werkzeug, generates a unique user ID, and stores the new user record along with metadata like creation time and role. On successful signup, it flashes a success message and redirects to the login page.

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

```

@app.route('/login/<role>', methods=['GET', 'POST'])
def login(role):
    if role not in ('patient', 'doctor'):
        flash("Invalid role.", "danger")
        return redirect(url_for('index'))

    if request.method == 'POST':
        email = request.form.get('email')
        password = request.form.get(['password'])

        if not email or not password:
            flash('Email and password are required.', 'warning')
            return render_template('login.html', role=role)

        # Rate-limiting check
        client_ip = request.remote_addr
        if client_ip in login_attempts:
            attempt_data = login_attempts[client_ip]
            if attempt_data['count'] >= 5:
                if datetime.now() - attempt_data['last_attempt'] < timedelta(minutes=15):
                    flash('Too many login attempts. Try again later.', 'danger')
                    return render_template('login.html', role=role)
            else:
                login_attempts[client_ip] = {'count': 0, 'last_attempt': datetime.now()}

```

```

# Fetch user data
user_data = None
if dynamodb:
    user_table = get_user_table()
    try:
        response = user_table.get_item(Key={'email': email})
        if 'Item' in response:
            user_data = response['Item']
    except Exception as e:
        logger.error(f"Error fetching user from DynamoDB: {e}")
else:
    user_data = local_db['users'].get(email)

# Validate user
if not user_data or user_data.get('role') != role:
    login_attempts.setdefault(client_ip, {'count': 0, 'last_attempt': datetime.now()})
    login_attempts[client_ip]['count'] += 1
    login_attempts[client_ip]['last_attempt'] = datetime.now()
    flash('Invalid email or role.', 'danger')
    return render_template('login.html', role=role)

if not check_password_hash(user_data.get('password_hash', ''), password):
    login_attempts.setdefault(client_ip, {'count': 0, 'last_attempt': datetime.now()})
    login_attempts[client_ip]['count'] += 1

```

```

        login_attempts[client_ip]['last_attempt'] = datetime.now()
        flash('Incorrect password.', 'danger')
        return render_template('login.html', role=role)
        # Reset rate-limiting counter
        login_attempts.pop(client_ip, None)

        session['user'] = user_data['email']    # This is what your @login_required checks!
        session['user_id'] = user_data['user_id']
        session['email'] = user_data['email']
        session['name'] = user_data['name']
        session['role'] = user_data['role']
        session.permanent = True

        logger.info(f"{role.capitalize()} logged in: {email}")
        flash('Login successful!', 'success')

        # Redirect to appropriate dashboard
        print("Redirecting to:", url_for(f"{role}_dashboard"))

        return redirect(url_for(f"{role}_dashboard"))

return render_template('login.html', role=role)
    
```

Description: define the /login/<role> route to handle login functionality for both patients and doctors. The route validates the user's email and password, checks for role mismatches, and verifies the password using Werkzeug. A rate-limiting mechanism restricts repeated failed login attempts based on IP address. On successful authentication, the user's session is initialized, login attempts are reset, and the user is redirected to their respective dashboard.

- Book appointment, patient dashboard and doctor dashboard routes:

```

@app.route('/patient_dashboard')
@login_required(role='patient')
def patient_dashboard():
    user_email = session['user']
    dashboard_data = get_patient_dashboard_data(user_email)
    doctors = [
        {'email': 'drsmith@example.com', 'name': 'Dr. Smith'},
        {'email': 'drjohnson@example.com', 'name': 'Dr. Johnson'},
        {'email': 'saikiran@gmail.com', 'name': 'Dr. Sai'},
        # ... more doctors ...
    ]
    return render_template('patient_dashboard.html', **dashboard_data, doctors=doctors)

@app.route('/doctor_dashboard')
@login_required(role='doctor')
def doctor_dashboard():
    user_email = session['user']
    dashboard_data = get_doctor_dashboard_data(user_email)
    return render_template('doctor_dashboard.html', **dashboard_data)
    
```

```

@app.route('/book_appointment', methods=['GET', 'POST'])
@login_required(role='patient')
def book_appointment():
    if request.method == 'POST':
        doctor_email = request.form['doctor']
        date = request.form['date']
        time = request.form['time']
        title = request.form.get('title', 'Consultation')
        location = request.form.get('location', 'Office 203')
        color = request.form.get('color', '#3498db')

        # Get doctor and patient names from local_db
        doctor_name = local_db['users'].get(doctor_email, {}).get('name', 'Doctor')
        patient_email = session['user']
        patient_name = local_db['users'].get(patient_email, {}).get('name', 'Patient')

```

```

        appointment = [
            'patient': patient_email,
            'patient_name': patient_name,
            'doctor': doctor_email,
            'doctor_name': doctor_name,
            'title': title,
            'date': date,
            'time': time,

```

```

            'location': location,
            'color': color
        ]
        local_db['appointments'].append(appointment)
        flash('Appointment booked successfully!', 'success')
        return redirect(url_for('patient_dashboard'))
    return render_template('book_appointment.html', user_name=session['name'])

```

Description: define the /book-appointment route to allow patients to schedule appointments with doctors by selecting from available options and submitting a form. The appointment details are stored in the Appointments DynamoDB table, and notifications may be triggered upon successful booking. The /patient-dashboard route displays personalized patient data including upcoming appointments, prescriptions, and medical history. The /doctor-dashboard route provides doctors with access to view and manage appointments, patient records, and diagnosis details.

Exit Route:

```
# Exit Page
@app.route('/exit')
def exit_page():
    return render_template('exit.html')
```

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

Deployment Code:

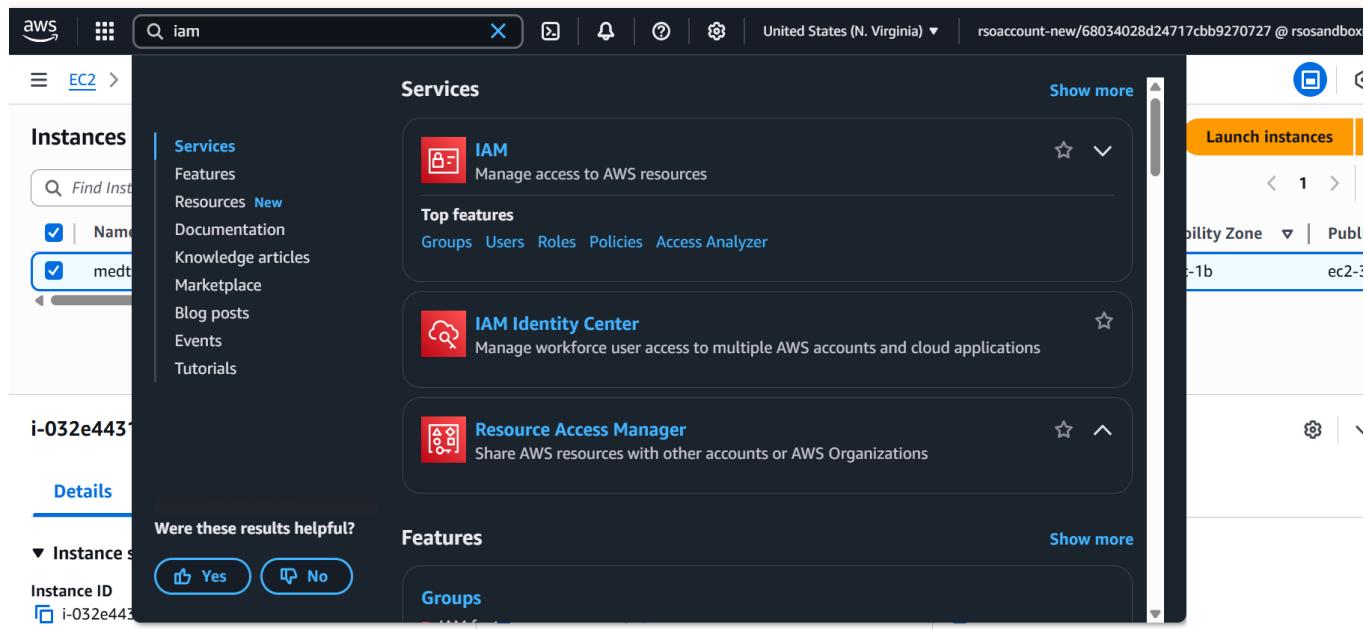
```
if __name__ == '__main__':
    app.run(debug=True)
```

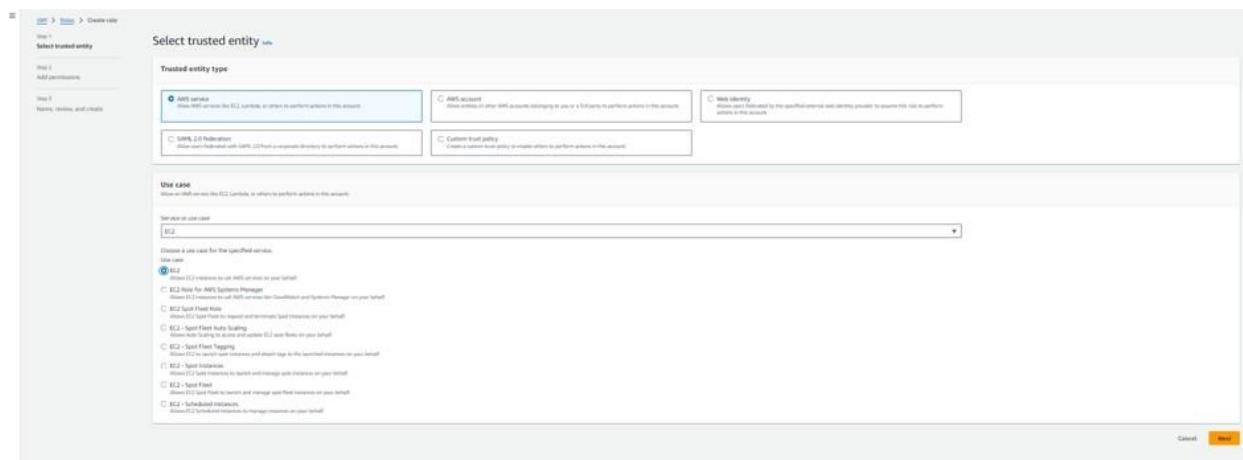
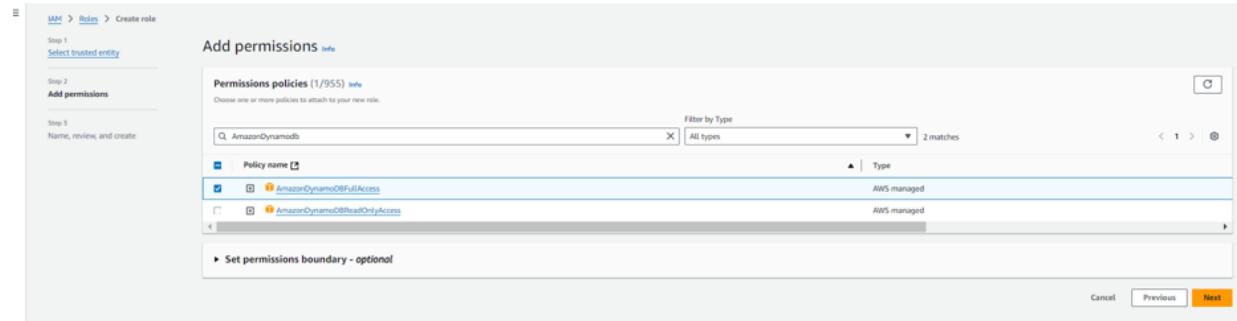
Description: This block checks if the script is being run directly. If so, it starts the Flask development server with debug=True, enabling live reload and detailed error messages for development and testing purposes.

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 notifications via SNS.

[IAM](#) > [Roles](#) > [Create role](#)
1 | 2

Step 3

[Name, review, and create](#)**Role name**

Enter a meaningful name to identify this role.

EC2_MedTrack1_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[Edit](#)**Trust policy**

1 ▾ [f]

3 | [IAM](#) > [Roles](#)
1 | 2
✓ Role EC2_MedTrack1_Role created.
[View role](#)

X

Roles (9) [Info](#)[Edit](#)[Delete](#)[Create role](#)

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

4 | Search

1 > 2 > 3

⚙

<input type="checkbox"/> Role name	▲ Trusted entities	Last activity
AWSServiceRoleForOrganizations	AWS Service: organizations (Service-Linked Role)	220 days ago
AWSServiceRoleForSSO	AWS Service: sso (Service-Linked Role)	-
AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Linked Role)	-
DCEPrincipal-bunnytf	Account: 058264256896	-
EC2_MedTrack_Role	AWS Service: ec2	-
EC2_MedTrack1_Role	AWS Service: ec2	-

Milestone 6: EC2 Instance Setup

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

 templates	Update index.html	5 days ago
 .env	Update .env	4 days ago
 app.py	Update app.py	5 days ago

Local **Codespace**

 **Clone**

[HTTPS](#) [SSH](#) [GitHub CLI](#)

<https://github.com/keerthianakapalli/Medtrack>

Clone using the web URL.

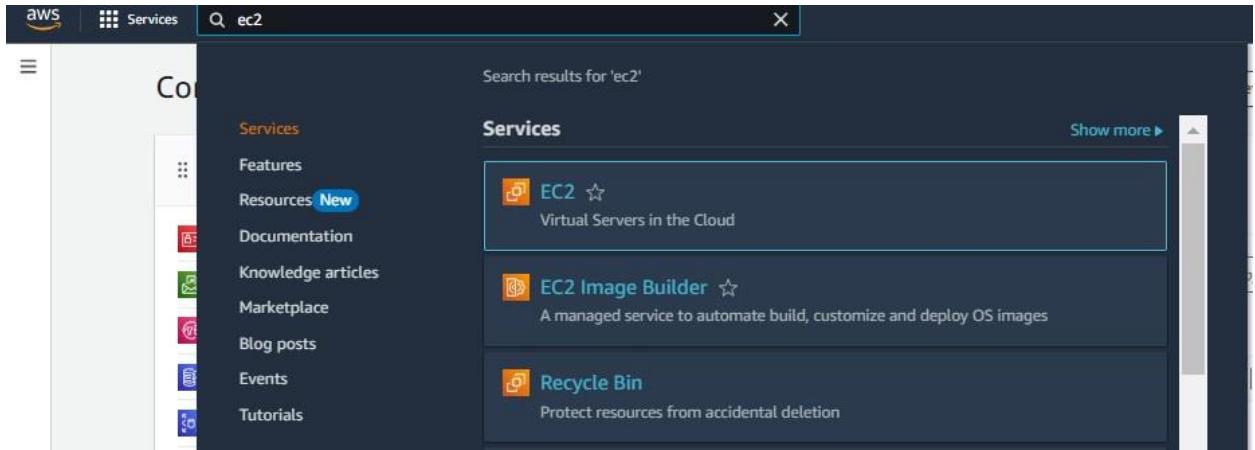
 [Open with GitHub Desktop](#)

 [Download ZIP](#)

- **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.



-
-
-
-
-

EC2 Dashboard X

EC2 Global View

Events

Instances

Instances

Instance Types

EC2 > Instances > Launch an instance

Instances Info

Last updated less than a minute ago

Find Instance by attribute or tag (case-sensitive)

All states

Name Instance ID Instance state Instance type Status check Alarm status Availability Zone Public IPv4 DN

No instances

You do not have any instances in this region

Launch instances

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

[Take a walkthrough](#) [Do not show me this message again.](#)

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

[Add additional tags](#)

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

EC2 > Instances > Launch an instance



Including AMIs from AWS, Marketplace and the Community

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

Create key pair

Key pair name
 Key pairs allow you to connect to your instance securely.

I

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

[Cancel](#) [Create key pair](#)


[medtrack-server.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

▼  [Create new key pair](#)

▼ **Network settings** Info

 [Edit](#)



☰ EC2 > Instances > Launch an instance

Type Info ssh	Protocol Info TCP	Port range Info 22
Source type Info My IP	Name Info Add CIDR, prefix list or security group	Description - optional Info e.g. SSH for admin desktop
▼ Security group rule 2 (TCP 80) 		
Type Info HTTP	Protocol Info TCP	Port range Info 80
Source type Info Custom	Source Info Add CIDR, prefix list or security group	Description - optional Info e.g. SSH for admin desktop
Add security group rule		

▼ Summary

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)

[Cancel](#) [Launch instance](#)

 [Preview code](#)

▼ Security group rule 2 (TCP, 80, 0.0.0.0/0)		Remove
Type Info	Protocol Info	Port range Info
HTTP ▼	TCP ▼	80
Source type Info	Source Info	Description - optional Info
Anywhere ▼	<input type="text"/> Add CIDR, prefix list or security group	e.g. SSH for admin desktop

☰ EC2 > Instances > Launch an instance

 **Success**
Successfully initiated launch of instance ([i-032e443157e075855](#))

▶ Launch log

Next Steps

Q What would you like to do next with this instance, for example "create alarm" or "create backup"

< 1 2 3 4 5 6 >

Create billing usage alerts

To manage costs and avoid surprise bills, set up email notifications for billing usage thresholds.

Create billing alerts

Connect to your instance

Once your instance is running, log into it from your local computer.

[Connect to instance](#)

Connect an RDS database

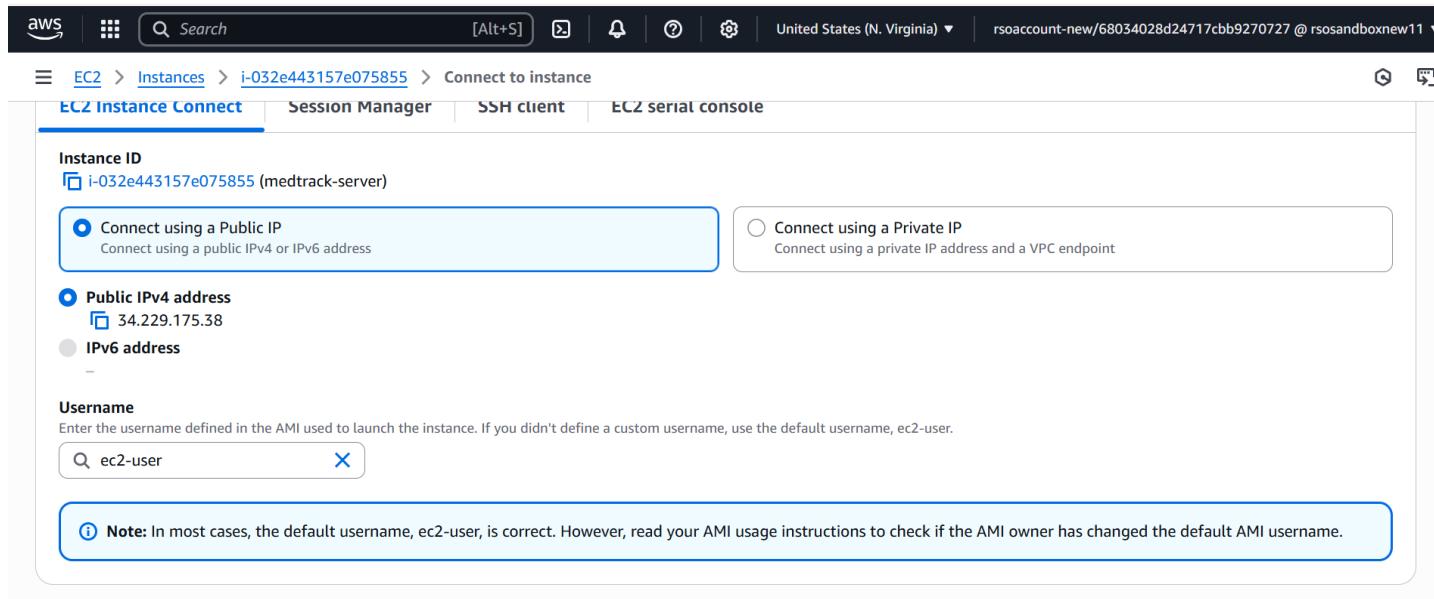
Configure the connection between an EC2 instance and a database to allow traffic flow between them.

[Connect an RDS database](#)

Create EBS snapshot policy

Create a policy that automates the creation, retention, and deletion of EBS snapshots

- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the IAM role is connected, navigate to the **EC2** section in the **AWS Management Console**. Select the **EC2 instance** you wish to connect to. At the top of the **EC2 Dashboard**, click the **Connect** button. From the connection methods presented, choose **EC2 Instance Connect**. Finally, click **Connect** again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.
- Now connect the EC2 with the files



Milestone 7: Deployment on EC2

Activity 7.1: Install Software on the EC2 Instance

Install Python3, Flask, and Git:

On Amazon Linux 2:

```
sudo yum update -y
sudo yum install python3 git
sudo pip3 install flask boto3
```

Verify Installations:

```
flask --version
git --version
```

Activity 7.2: Clone Your Flask Project from GitHub

Clone your project repository from GitHub into the EC2 instance using Git.

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

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

here: 'git clone https://github.com/keerthianakapalli/Medtrack.git'

- This will download your project to the EC2 instance.

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

```
cd Medtrack
```

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
```

```

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/AlekhyaPenuakula/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]$ ^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
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:

Public IPs: <https://34.229.175.38/>

Milestone 8: Testing and Deployment

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

Login Page:

MedTrack

Patient Login

Email:

Password:

[Login](#)

Don't have an account?

[Sign Up](#)

Signup Page:

MedTrack

Patient Signup

Name

Enter your name

Email

Enter your email

Password

Create a password

[Sign Up](#)

Home page:

 **MedTrack**
Home Features About Us Contact
Login patient
Login doctor

Welcome to MedTrack

A smart, secure platform to manage your medical records and health info—all in one place.

[Learn More](#)

About Us page:


Home Features About Us Contact
Login patient
Login doctor

About MedTrack



MedTrack was founded with a simple mission: to empower individuals to take control of their healthcare journey. Our platform bridges the gap between patients and healthcare providers, creating a seamless experience for managing medical information.

Developed by a team of healthcare professionals and technology experts, MediTrack combines medical expertise with cutting-edge technology to deliver a solution that addresses the real needs of patients and doctors.

We believe that better health management leads to better health outcomes. That's why we're dedicated to creating tools that make healthcare more accessible, organized, and personalized for everyone.

50,000+

Active Users

1,200+

Healthcare Providers

98%

User Satisfaction

Contact Page:

 **MedTrack**

- [Home](#)
- [Features](#)
- [About Us](#)
- [Contact](#)

[Login patient](#)
[Login doctor](#)

Contact Information

Feel free to reach out to us using any of the methods below. We're available Monday through Friday, 9am to 6pm EST.

 **Our Location**
123 Health Avenue, Boston, MA 02115

 **Phone Number**
+1 (800) 555-MED

 **Email Address**
support@medittrack.com

 **Working Hours**
Monday - Saturday: 9am - 6pm EST

Send us a Message

Your Name

Your Email

Subject

Your Message

[Send Message](#)

Features page:

 **MedTrack**

- [Home](#)
- [Features](#)
- [About Us](#)
- [Contact](#)

[Login patient](#)
[Login doctor](#)

Key Features

Discover the powerful tools that make MediTrack the best health companion for patients and healthcare providers.



Medication Management

Track your medications, get reminders for doses, and receive alerts when it's time for refills.

[Learn more →](#)

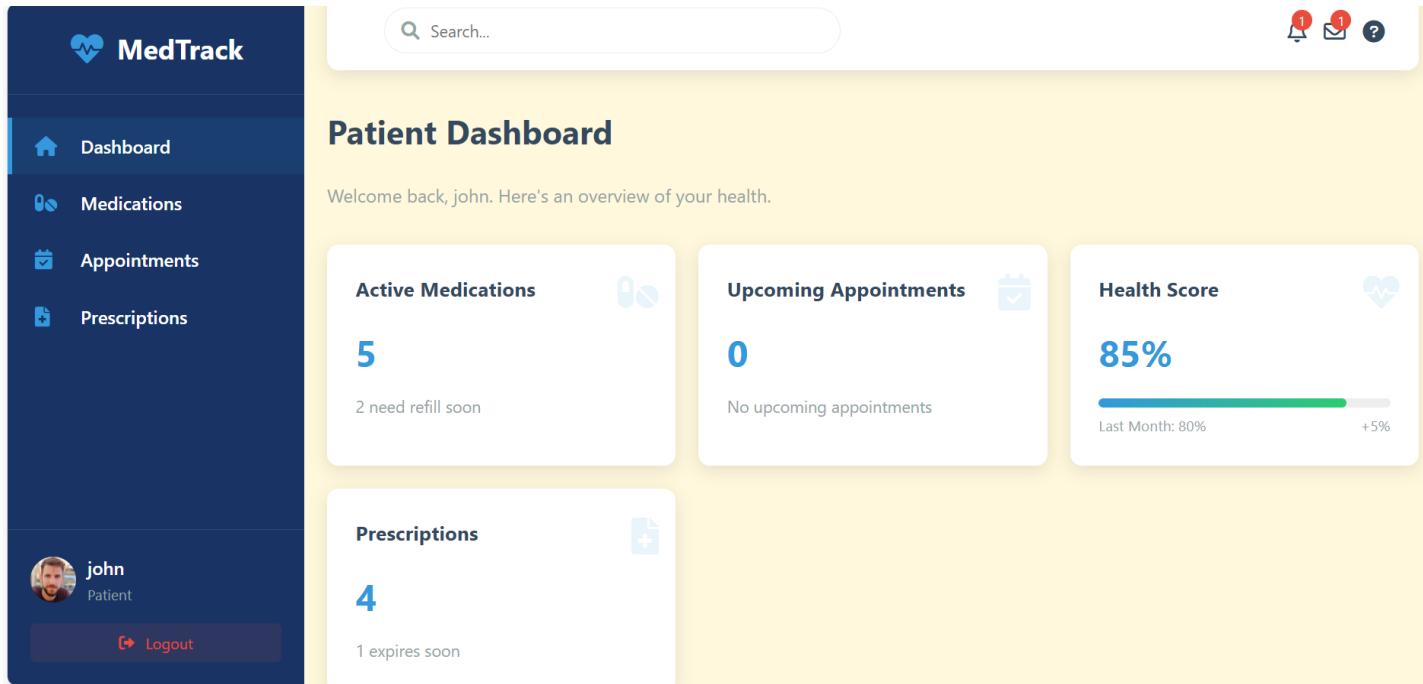


Appointment Scheduling

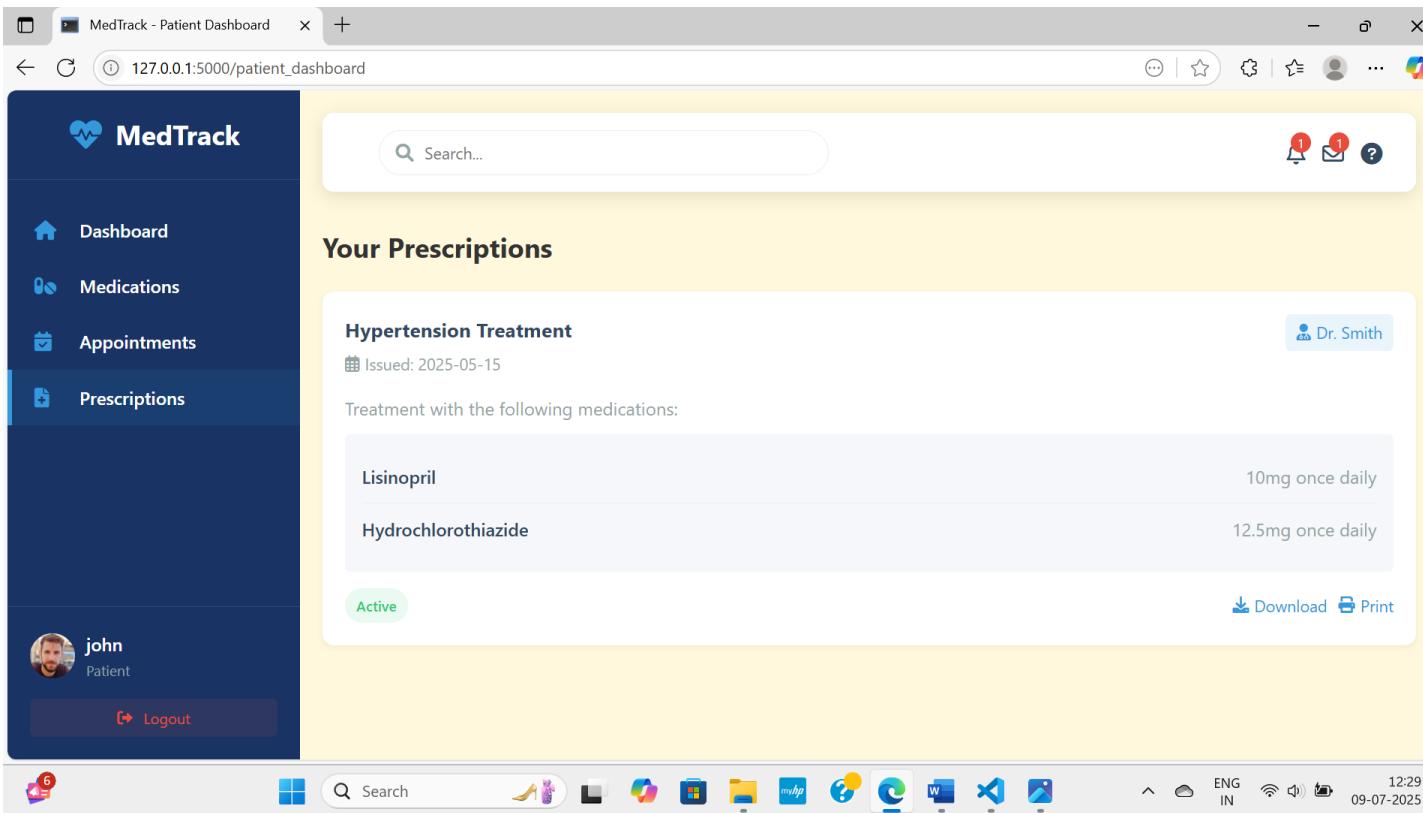


Privacy & Security

Patient dashboard page:

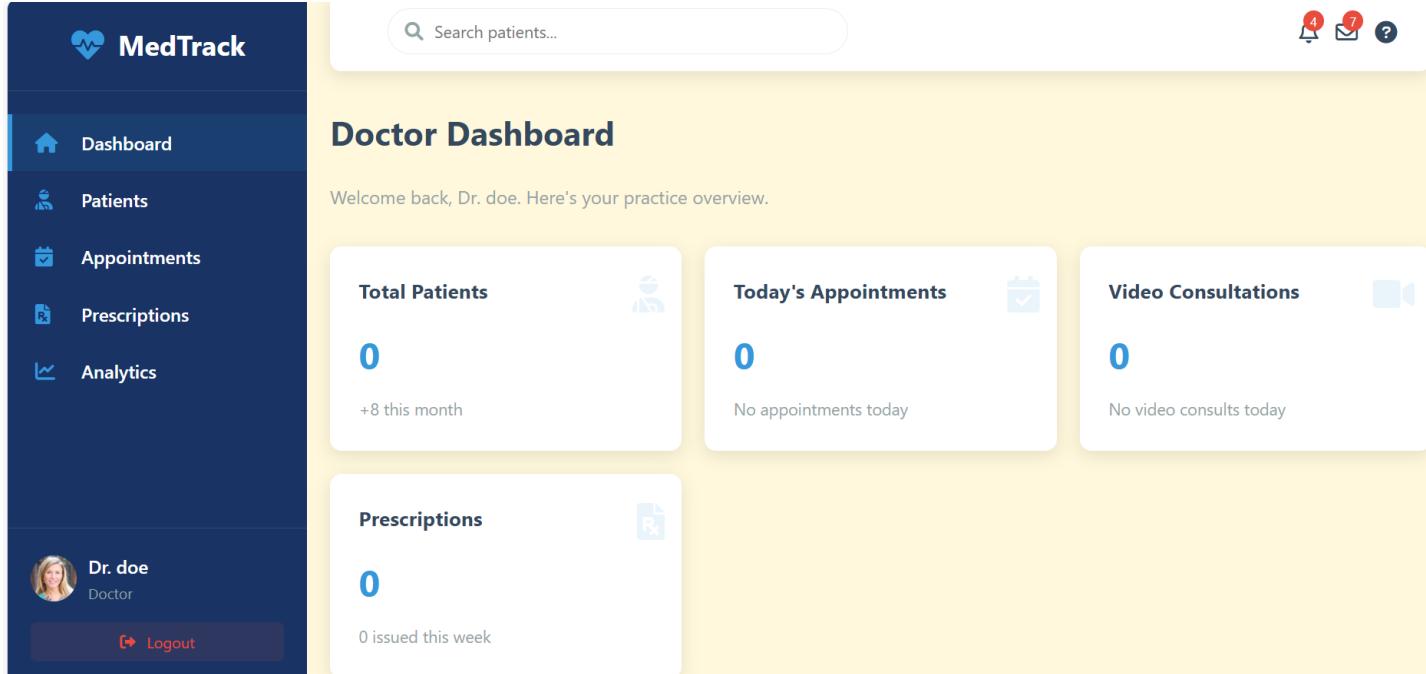


The screenshot shows the MedTrack Patient Dashboard. On the left sidebar, there are links for Dashboard, Medications, Appointments, Prescriptions, and Logout. The main area displays a welcome message: "Welcome back, john. Here's an overview of your health." Below this are four cards: "Active Medications" (5, 2 need refill soon), "Upcoming Appointments" (0, No upcoming appointments), "Health Score" (85%, Last Month: 80%, +5%), and "Prescriptions" (4, 1 expires soon).

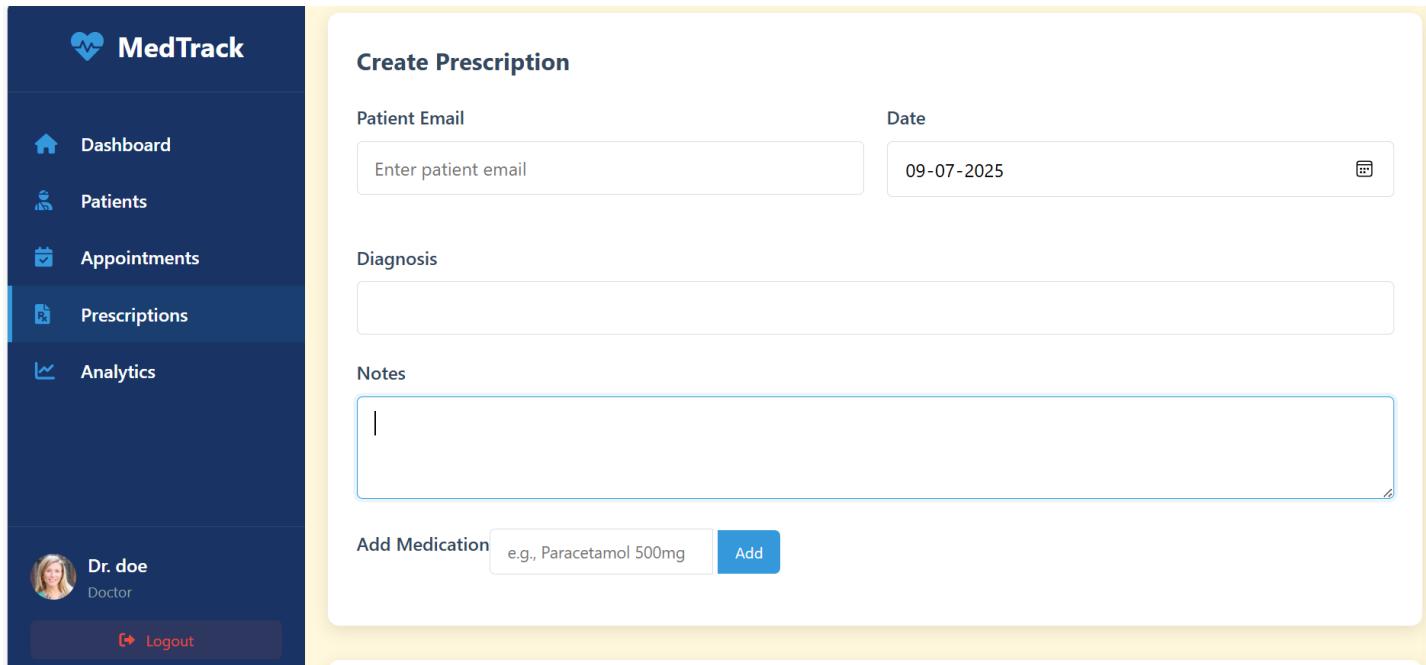


The screenshot shows the "Your Prescriptions" section of the dashboard. It lists a treatment for "Hypertension Treatment" issued on "2025-05-15" by "Dr. Smith". The treatment includes "Lisinopril 10mg once daily" and "Hydrochlorothiazide 12.5mg once daily". A green "Active" button is present. At the bottom right, there are "Download" and "Print" icons.

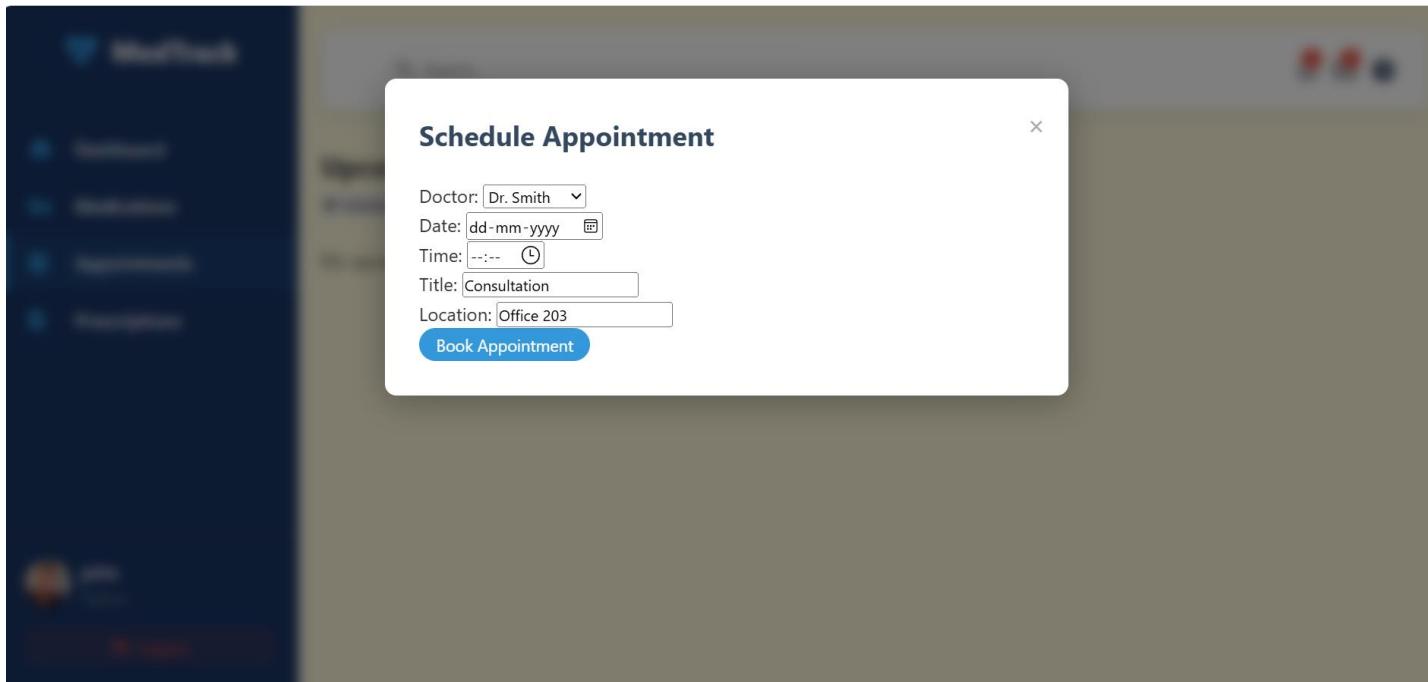
Doctor dashboard page:



The dashboard features a sidebar on the left with icons for Dashboard, Patients, Appointments, Prescriptions, and Analytics. It also shows a profile picture of Dr. doe and a Logout button. The main area has a search bar at the top right and a title "Doctor Dashboard". A welcome message says "Welcome back, Dr. doe. Here's your practice overview." Below are four cards: "Total Patients" (0, +8 this month), "Today's Appointments" (0, No appointments today), "Prescriptions" (0, 0 issued this week), and "Video Consultations" (0, No video consults today).



The page shows the "Create Prescription" form. It includes fields for "Patient Email" (with placeholder "Enter patient email"), "Date" (set to "09-07-2025"), "Diagnosis" (an empty text area), "Notes" (an empty text area), and a "Add Medication" field with placeholder "e.g., Paracetamol 500mg" and a "Add" button.

Book Appointment page:**Exit:****Session Ended**

Please close this tab.

Conclusion:

The MedTrack application has been successfully developed and deployed in a local environment to demonstrate core healthcare functionalities such as user registration, login, appointment booking, medication tracking, and patient-doctor interaction. By utilizing Flask for the backend, HTML/CSS for the frontend, and a local mock database for data handling, the system ensures smooth and responsive user interactions during development and testing.

This locally deployed setup allows for effective testing of essential workflows like patient dashboard access, doctor appointments, diagnosis entry, and reminder notifications, without the need for cloud infrastructure. Secure password management, session handling, and form validation have been implemented to ensure a realistic and secure simulation of a healthcare system.

In conclusion, the local deployment of Medtrack serves as a reliable foundation for development, enabling efficient prototyping, testing, and demonstration of all key features. It lays the groundwork for future migration to a scalable cloud-based environment while already providing value as an offline or intranet-based medical record management system.

