

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:

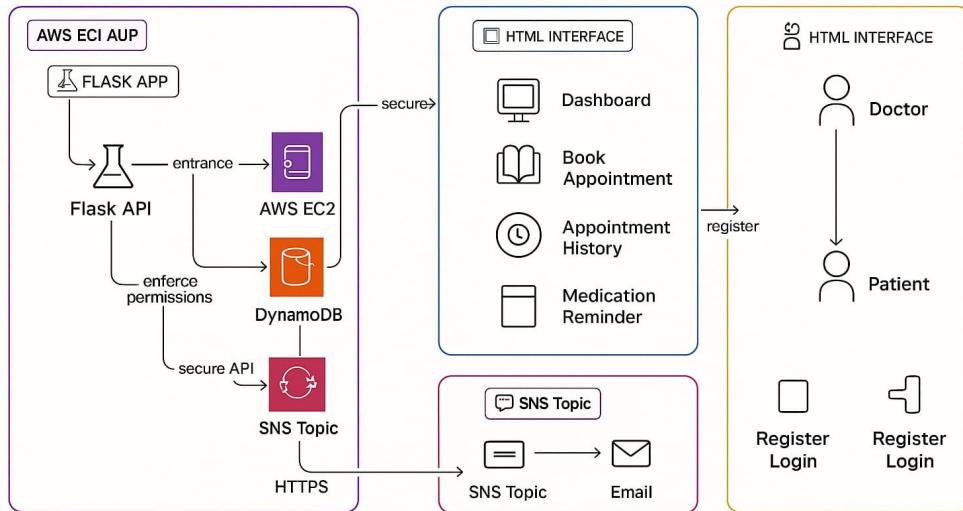
MedTrack utilizes AWS IAM to manage user permissions and ensure secure access to the system. For instance, when a new patient registers, an IAM user is created with specific roles and permissions to access only the features relevant to them. Doctors have their own IAM configurations, allowing them access 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.

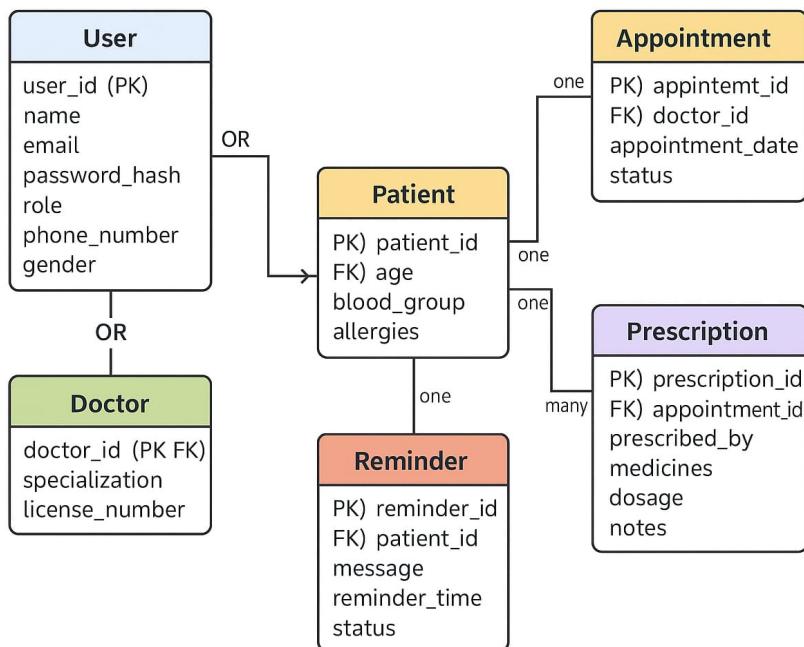
AWS ARCHITECTURE:

Medtack Infrastructure



Patients or doctors can create a new account or log in if they already have existing credentials to book appointments

ER DIAGRAM:



Pre-requisites:

1. **.AWS Account Setup:** [AWS Account Setup](#)
2. **Understanding IAM:** [IAM Overview](#)

3. **Amazon EC2 Basics:** [EC2Tutorial](#)
4. **DynamoDB Basics:** [DynamoDB Introduction](#)
5. **SNS Overview:** [SNSDocumentation](#)
6. **Git Version Control:** [Git Documentation](#)

PROJECT WORK FLOW

1.AWS Account Setup and Login:

Activity1.1: Set up an AWS account if not already done.

Activity1.2: Log into the AWS Management Console

2.DynamoDB Database Creation and Setup:

Activity2.1: Create a DynamoDB Table.

Activity2.2: Configure Attributes for User Data and Book Requests.

3.SNS Notification Setup:

Activity3.1: Create SNS topics for book request notifications.

Activity3.2: Subscribe users and library staff to SNS email notifications.

4.Backend Development and Application Setup:

Activity4.1: Develop the Backend Using Flask.

Activity4.2: Integrate AWS Services Using boto3.

5.IAM Role Setup:

Activity5.1: Create IAM Role

Activity5.2: Attach Policies

6.EC2 Instance Setup

Activity6.1:LaunchanEC2instancetohosttheFlaskapplication.

Activity6.2:ConfiguresecuritygroupsforHTTP, andSSHaccess.

7. Deployment on EC2

Activity7.1:UploadFlaskFiles

Activity7.2:RuntheFlaskApp

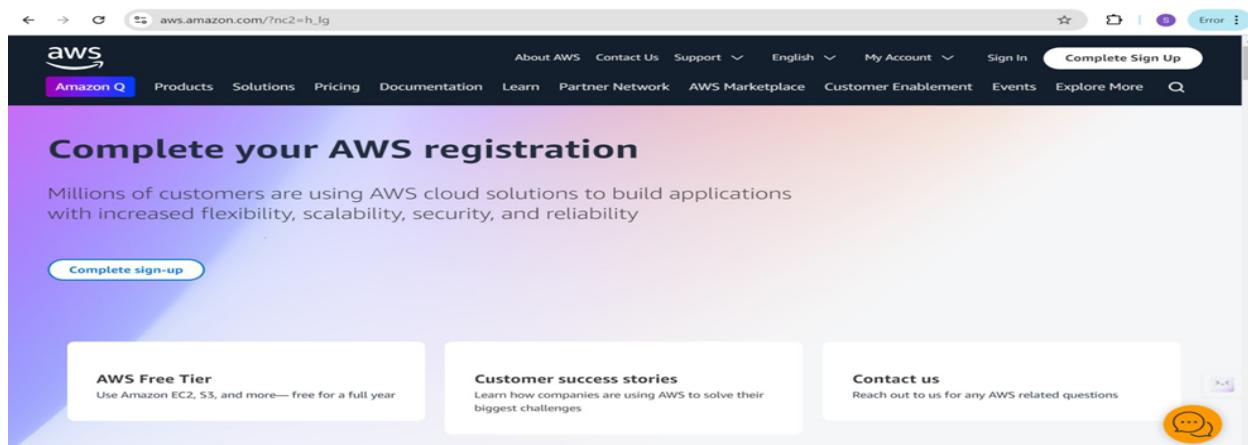
8. Testing and Deployment

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

Milestone 1: AWS Account Setup and Login

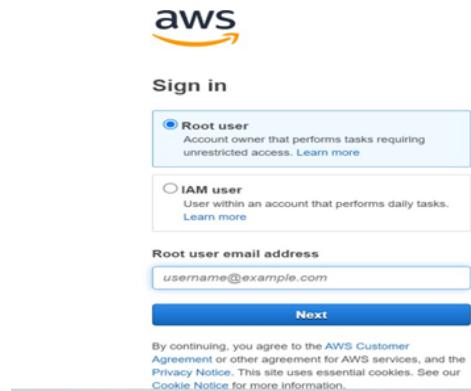
- **Activity 1.1: Set up an AWS account if not already done.**

- Sign up for an AWS account and configure billing settings.



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

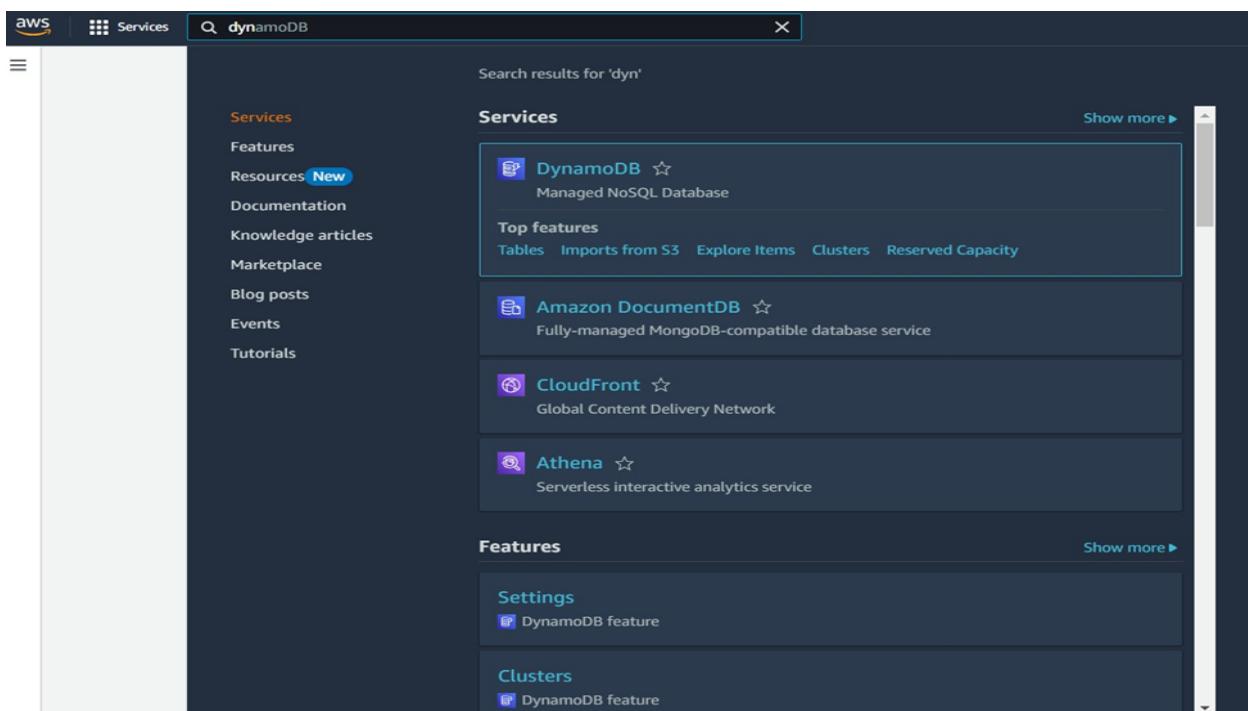
- After setting up your account, log in to the [AWS Management Console](#).



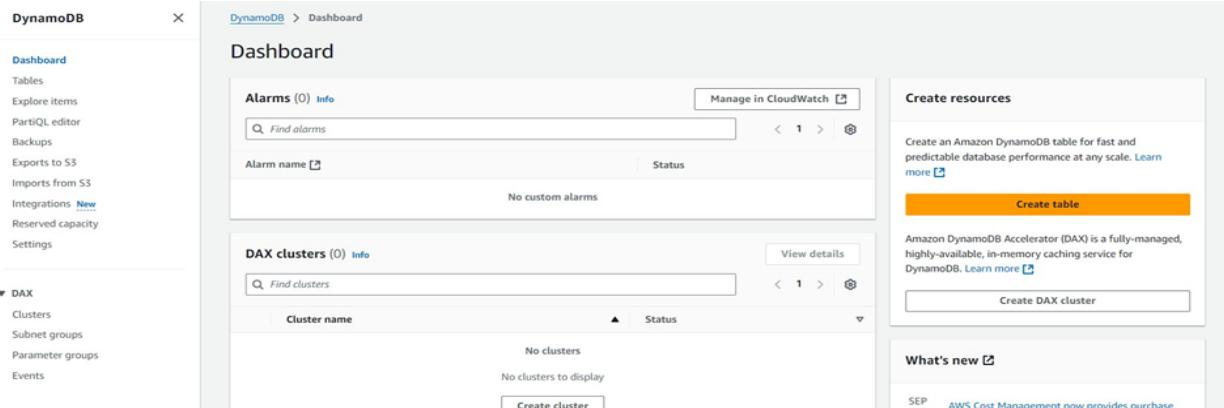
Milestone 2: DynamoDB Database Creation and Setup

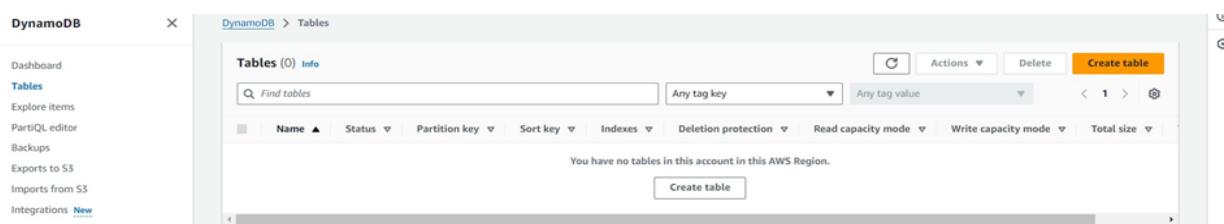
- **Activity 2.1: Navigate to the DynamoDB**

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



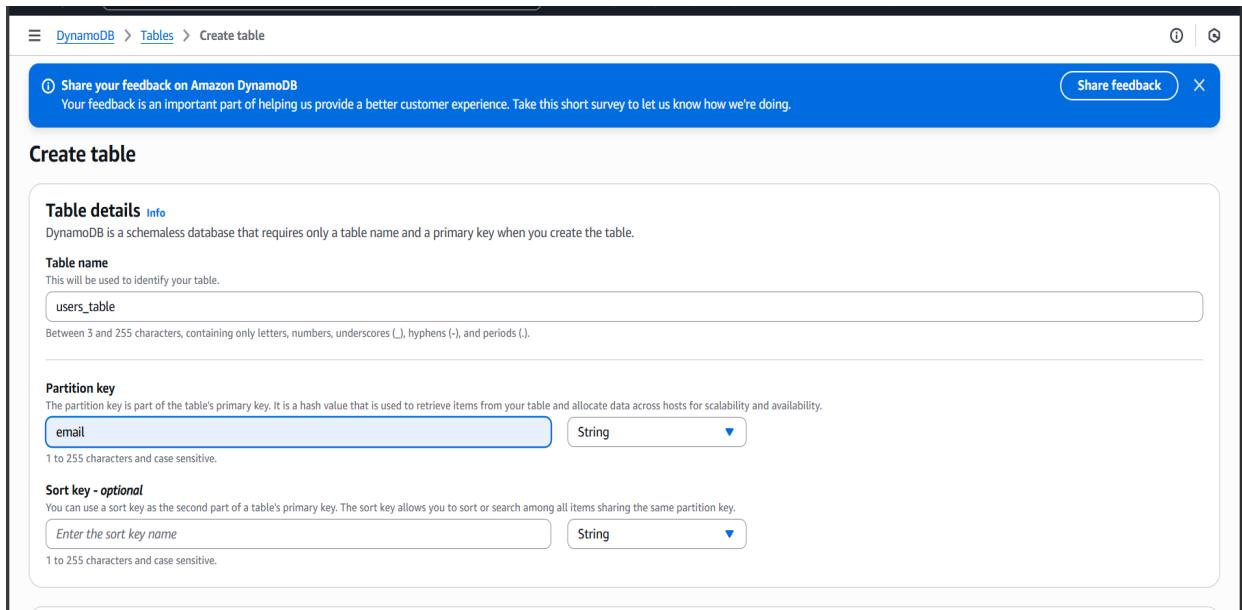
The screenshot shows the AWS Services search results for 'dynamodb'. The search bar at the top contains 'dynamodb'. On the left, there is a sidebar with links to Services, Features, Resources (New), Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials. The main area is titled 'Search results for "dyn"' and shows a list of services under 'Services' and 'Features'. Under 'Services', 'DynamoDB' is listed as a Managed NoSQL Database. Under 'Features', 'Amazon DocumentDB' is listed as a Fully-managed MongoDB-compatible database service, 'CloudFront' as a Global Content Delivery Network, and 'Athena' as a Serverless interactive analytics service. There are 'Show more' buttons for both sections.





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

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



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.

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 class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

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

No tags are associated with the resource.

[Add new tag](#)

You can add 50 more tags.

[Cancel](#) [Create table](#)

The screenshot shows the AWS DynamoDB 'Tables' page. On the left, there's a navigation sidebar with 'DynamoDB' selected. The main area displays a table with one item: 'Users'. The table has columns for Name, Status, Partition key, Sort key, Indexes, Replication Regions, Deletion protection, Favorite, and Read capacity mode. The 'Users' row shows 'Active' status, 'email (\$)' as the partition key, and 'On-demand' as the read capacity mode. Above the table, a success message says 'The Users table was created successfully.'

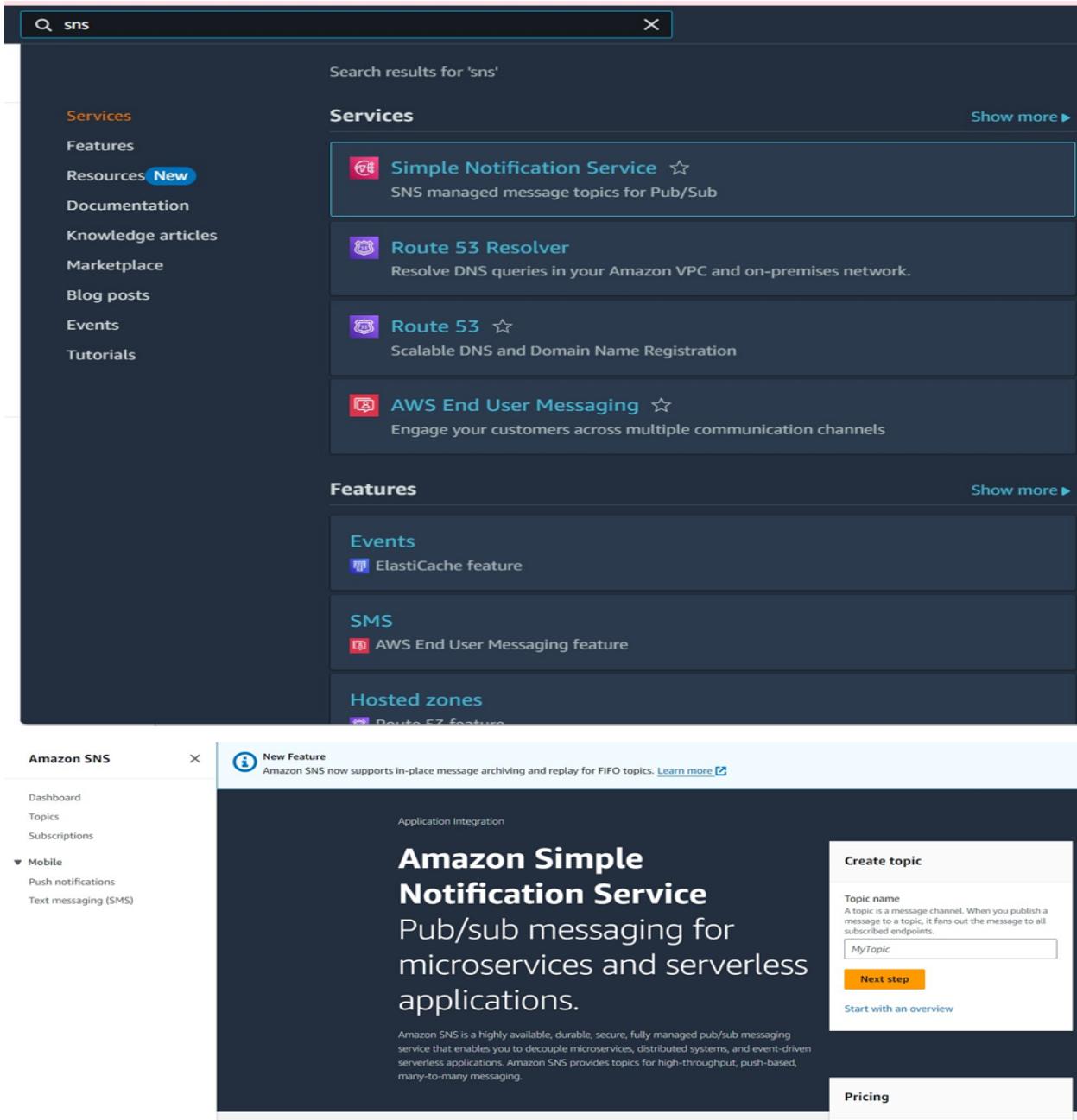
- Follow the same steps to create an Appointments,Prescriptions,Medications tablewith ID as the primary key for book requests data.

The screenshot shows the AWS DynamoDB 'Tables' page. The navigation sidebar is identical to the previous screenshot. The main area now displays four tables: 'Appointments', 'Medications', 'Prescriptions', and 'Users'. Each table has the same structure as the 'Users' table in the first screenshot. Above the table, a success message says 'The Medications table was created successfully.'

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.



The screenshot shows two main sections: the AWS Services search results and the Amazon SNS dashboard.

AWS Services Search Results:

- Search bar: sns
- Search results for 'sns':
 - Simple Notification Service** (New) - 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
- Services sidebar:
 - Services
 - Features
 - Resources **New**
 - Documentation
 - Knowledge articles
 - Marketplace
 - Blog posts
 - Events
 - Tutorials

Amazon SNS Dashboard:

- Left sidebar:
 - Amazon SNS
 - Dashboard
 - Topics
 - Subscriptions
 - Mobile
 - Push notifications
 - Text messaging (SMS)
- Top banner: New Feature - Amazon SNS now supports in-place message archiving and replay for FIFO topics. [Learn more](#)
- Main content area:
 - Application Integration
 - Amazon Simple Notification Service** - Pub/sub messaging for microservices and serverless applications.
 - Description: 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 form:
 - Topic name: MyTopic
 - Next step: [Next step](#)
 - Start with an overview: [Start with an overview](#)
 - Pricing: [Pricing](#)

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

The image shows the Smartbridge logo at the top left. To its right is the Amazon SNS Topics interface. A blue banner at the top of the interface reads "New Feature" followed by the text "Amazon SNS now supports in-place message archiving and replay for FIFO topics. Learn more". Below the banner, the interface shows a list titled "Topics (0)" with a search bar and a "Create topic" button.

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

The image shows the "Create topic" form in the Amazon SNS console. The "Type" section is expanded, showing two options: "FIFO (first-in, first-out)" and "Standard". The "Standard" option is selected and highlighted with a blue border. The "Name" field contains the value "MedTrackNotification". The "Display name - optional" field contains the value "My Topic". Below these fields, several sections are listed as expandable accordions:

- ▶ Access policy - optional Info
- ▶ Data protection policy - optional Info
- ▶ Delivery policy (HTTP/S) - optional Info
- ▶ Delivery status logging - optional Info
- ▶ Tags - optional
- ▶ Active tracing - optional Info

At the bottom right of the form, there are "Cancel" and "Create topic" buttons.

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



The screenshot shows the AWS SNS console with the following details:

- Topic:** MedTrackNotification
- Name:** MedTrackNotification
- ARN:** arn:aws:sns:us-east-1:241533142623:MedTrackNotification
- Type:** Standard
- Display name:** -
- Topic owner:** 241533142623

Below the topic details, there are tabs for Subscriptions, Access policy, Data protection policy, Delivery policy (HTTP/S), Delivery status logging, Encryption, and Tags. The Subscriptions tab is selected, showing 0 subscriptions. There are buttons for Edit, Delete, Request confirmation, Confirm subscription, and Create subscription.

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.

The screenshot shows the AWS SNS console with the following details:

Create subscription

Details

Topic ARN: arn:aws:sns:us-east-1:241533142623:MedTrackNotification

Protocol: Email

Endpoint: An email address that can receive notifications from Amazon SNS.

Success Message: Subscription to MedTrackNotification created successfully. The ARN of the subscription is arn:aws:sns:us-east-1:241533142623:MedTrackNotification:5b6ddd93-7912-437c-ba65-272244d944f5.

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



Gmail

Compose

Inbox 1,068

Starred

Snoozed

Sent

Drafts 6

Less

Important

Chats

Scheduled

All Mail

Spam 9

Trash

Categories

Manage subscriptions

Manage labels

Create new label

Labels +

in:spam

Delete forever Not spam

AWS Notification - Subscription Confirmation Spam

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

7:53PM (0 minutes ago)

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

Report not spam

You have chosen to subscribe to the topic:
arn:aws:sns:us-east-1:879381270777:MedTrackNotifications

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](#).

Reply Forward

Enable desktop notifications for Gmail. OK No thanks



Simple Notification Service

Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:

arn:aws:sns:us-east-1:879381270777:MedTrackNotifications:d997a23b-3620-49fb-8238-786366e5811d

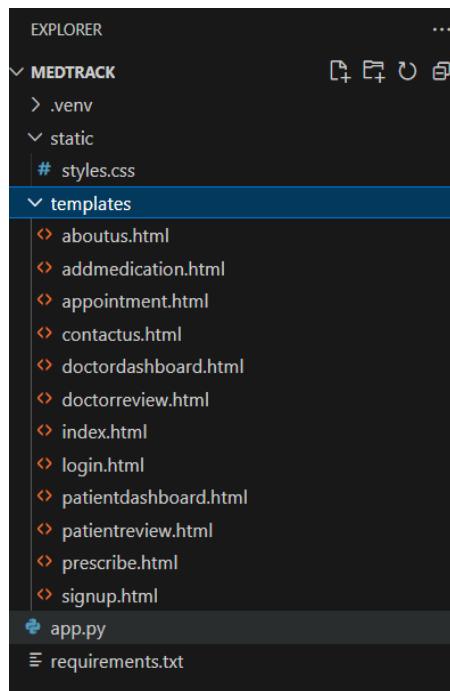
If it was not your intention to subscribe, [click here to unsubscribe](#).

- 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



Description: set up the INSTANT LIBRARY project with an app.py file, a static/ folder for assets, and a templates/ directory containing all required HTML pages like index, login, signup, subject-specific pages (e.g., Doctordashboard.html, patientdashboard.html), and utility pages.

Description of the code :

● Flask App Initialization

```
p.py > ...
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

app = Flask(__name__)
app.secret_key = 'randomkey'

USE_DYNAMODB = False # Set to True when deploying on AWS
AWS_REGION = 'us-east-1'
```

Description: This project begins by importing all essential libraries required for MedTrack. Flask utilities are used to handle routing, sessions, and template rendering. **Boto3** is used to interact with AWS **DynamoDB** for storing user, appointment, and prescription data. **SMTP** and the **email.mime** modules are utilized for sending custom email notifications to users. Additionally, libraries like **uuid** and **datetime** assist in generating unique IDs and managing date/time operations.

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

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

- **Dynamodb Setup:**

```
USE_DYNAMODB = True # Set to True when deploying on AWS
AWS_REGION = 'us-east-1'

if USE_DYNAMODB:
    dynamodb = boto3.resource('dynamodb', region_name=AWS_REGION)
    users_table = dynamodb.Table('Users')
    appointments_table = dynamodb.Table('Appointments')
    prescriptions_table = dynamodb.Table('Prescriptions')
    medications_table = dynamodb.Table('Medications')
```

Description: Initialize the **DynamoDB** resource for the **us-east-1** region and set up access to the Users, Appointments, Prescriptions, and Medications tables. These tables are used to securely store user information, appointment bookings, doctor-issued prescriptions, and patient medication details for the MedTrack healthcare application.

- **SNS Connection:**

```
p.py / ...
SNS_TOPIC_ARN = "arn:aws:sns:ap-south-1:123456789012:MedTrackNotifications"
sns = boto3.client('sns', region_name=AWS_REGION)
def send_sns_email(subject, message):
    try:
        sns.publish(
            TopicArn=SNS_TOPIC_ARN,
            Message=message,
            Subject=subject
        )
        print("✉ SNS email sent successfully.")
    except Exception as e:
        print(f"✗ Failed to send SNS email: {e}")

def send_local_sms(phone, message):
    print(f"[SIMULATED SMS to {phone}]: {message}")
db = {
    'users': {
        "dr.john@example.com": {
            "name": "Dr. John", "password": "doc123", "role": "doctor", "phone": "1234567890", "reminders": []
        },
        "patient.rita@example.com": {
            "name": "Rita", "password": "pat456", "role": "patient", "phone": "9876543210", "reminders": []
        }
    },
    'appointments': [],
    'prescriptions': [],
    'medications': []
}
```

Description:

Configure **Amazon SNS** with your Topic ARN and region (e.g., us-east-1) to send medical alerts. Set up **SMTP** (like Gmail) in **SENDER_EMAIL** and **SENDER_PASSWORD** using an app password to send local email notifications for reminders and signups.

● Routes for Web Pages

Home Route:

Description: Defines the home route / to render the **index page**, which contains the **Login** and **Sign Up** buttons for both patients and doctors.

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

● Sign Up Route:

Description: Defines /signup route to **collect user details** (name, email, password, phone, role, and reminder frequency), **store them in DynamoDB or local DB**, and **send an SNS Welcome email** upon successful registration.

```

@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        data = {k: request.form[k] for k in ['name', 'email', 'password', 'phone', 'role', 'reminder']}
        users = load_users()
        if data['email'] in users:
            flash("Email already exists", "error")
            return redirect(url_for('signup'))
        db['users'][data['email']] = data

        subject = "🌟 Welcome to MedTrack!"
        message = f"Hi {data['name']},\n\nYou have successfully registered in the MedTrack app.\n\nStay healthy!"
        send sns_email(subject, message)

        flash("Signup successful! Now login.", "success")
        return redirect(url_for('login'))
    return render_template('signup.html')

```

Login Route (GET/POST):

Description: Defines /login route to validate login credentials against the **user database**, initiate a session, and **redirect doctors to doctordashboard or patients to patientdashboard** based on their role.

```

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email, password = request.form['email'], request.form['password']
        user = db['users'].get(email)
        if user and user['password'] == password:
            session.update({"email": email, "role": user['role'], "name": user['name'], "phone": user.get('phone', '')})
            return redirect(url_for(f"[user['role']]dashboard"))
        flash("Invalid credentials", "error")
    return render_template('login.html')

```

- Doctor Dashboard Route:

Description: Defines /doctordashboard route where doctors can **view, approve, or reject appointments**, and **prescribe medications**. Also displays **alerts for missed medications** by patients.



```
@app.route('/doctordashboard', methods=['GET', 'POST'])
def doctordashboard():
    if session.get('role') != 'doctor':
        return redirect(url_for('login'))

    name = session['name']
    doctor_appts = [a for a in db['appointments'] if a['doctor'].strip().lower() == name.strip().lower()]

    now = datetime.now()
    for appt in doctor_appts:
        appt_time_str = f"{appt['date']} {appt['time']}"
        try:
            appt_time = datetime.strptime(appt_time_str, "%Y-%m-%d %H:%M")
        except ValueError:
            continue
        if appt['status'] == 'Pending' and now > appt_time:
            appt['status'] = 'Missed'

    alerts = []
    for m in db['medications']:
        if m['status'].lower() == 'missed':
            alerts.append(f"Patient {m['patient']} missed dose of {m['medicine']} at {m['time']}")

    if request.method == 'POST':
        decision = request.form['decision']
        appt_id = request.form['appointment_id']
        for a in db['appointments']:
            if a['id'] == appt_id:
                a['status'] = decision
                break
        flash(f"Appointment {decision.lower()}ed.", "info")

    return render_template('doctordashboard.html',
                           name=name,
                           email=session['email'],
                           appointments=doctor_appts,
```

Patient Dashboard Route:

Description: Defines /patientdashboard route where patients can **view appointments, prescriptions, and medication schedules**, and **mark doses as Taken, Missed, or Pending**

```

@app.route('/patientdashboard')
def patientdashboard():
    if session.get('role') != 'patient':
        return redirect(url_for('login'))

    name = session.get('name')
    email = session.get('email')

    user_appointments = [appt for appt in db['appointments'] if appt['patient'] == name]

    # Convert date + time to datetime object and check if it's passed
    now = datetime.now()
    for appt in user_appointments:
        appt_time_str = f"{appt['date']} {appt['time']}"
        try:
            appt_time = datetime.strptime(appt_time_str, "%Y-%m-%d %H:%M")
        except ValueError:
            appt_time = now # fallback if formatting is bad

        if appt['status'] == 'Pending' and now > appt_time:
            appt['status'] = 'Missed'

    user_medications = [med for med in db['medications'] if med['patient'] == name]
    user_prescriptions = [p for p in load_prescriptions() if p['patient'] == name]

    alerts = []
    for med in user_medications:
        med_time_str = f"{med.get('date', now.strftime('%Y-%m-%d'))} {med['time'].split()[0]}"

        try:
            med_time = datetime.strptime(med_time_str, "%Y-%m-%d %H:%M")
        except:
            continue
        if med['status'] == 'Pending' and now > med_time:
            med['status'] = 'Missed'
    
```

Appointment Booking Route:

Description: Defines /appointment route to **allow patients to book appointments** by submitting required details. Bookings are saved in **DynamoDB or local DB** and shown on dashboards

```

@app.route('/appointment', methods=['GET', 'POST'])
def appointment():
    if request.method == 'POST':
        f = request.form
        appt = {
            'id': str(uuid.uuid4()), 'patient': session.get('name'), 'name': f['name'], 'gender': f['gender'],
            'phone': f['phone'], 'age': f['age'], 'email': session.get('email'),
            'department': f['department'], 'problem': f['problem'], 'doctor': f['doctor'],
            'status': 'Pending', 'date': f['date'], 'time': f['time']
        }
        save_appointment(appt)
        flash("Appointment booked successfully!", "success")
        return redirect(url_for('appointment'))
    return render_template('appointment.html')
    
```

Prescription Route:

Description: Defines /prescribe/<appt_id> route for doctors to add prescriptions linked to a specific appointment. Medications are saved and reminders are scheduled.

```
3 @app.route('/prescribe/<appt_id>', methods=['GET', 'POST'])
4 def prescribe(appt_id):
5     if request.method == 'POST':
6         f = request.form
7         time = f["{f['time']} {f['ampm']}]" if 'ampm' in f else f['time']
8         presc = {
9             'id': str(uuid.uuid4()), 'appointment_id': appt_id, 'medicine': f['medicine'],
0             'dosage': f['dosage'], 'days_left': f['days_left'], 'time': time,
1             'status': 'Pending', 'patient': f['patient']
2         }
3         save_prescription(presc)
4         save_medication(presc)
5         flash("Prescription added & medication reminder set.", "success")
6         return redirect(url_for('doctordashboard'))
7     appt = next((a for a in db['appointments'] if a['id'] == appt_id), None)
8     return render_template('prescribe.html', appointment=appt)
```

Add Medication Route:

Description: Defines /addmedication route for patients to **manually add a medication schedule**, which is later used for reminders.

```
@app.route('/addmedication', methods=['GET', 'POST'])
def addmedication():
    if session.get('role') != 'patient': return redirect(url_for('login'))
    if request.method == 'POST':
        f = request.form
        med = {
            'id': str(uuid.uuid4()), 'patient': session['name'], 'medicine': f['medicine'], 'date': f['date'],
            'dosage': f['dosage'], 'time': f['time'], 'days_left': f['days_left'], 'status': f.get('status', 'Pending')
        }
        save_medication(med)
        flash("Medication added successfully.", "success")
        return redirect(url_for('patientdashboard'))
    return render_template('addmedication.html')
```

Notification Route:

Description:Defines /send_notifications route to **send SNS notifications** to patients for scheduled medications based on the current date and time.

```

@app.route('/send_notifications')
def send_notifications():
    now = datetime.now()
    current_time = now.strftime("%H:%M")
    today = now.strftime("%Y-%m-%d")
    reminders_sent = 0

    for med in db['medications']:
        med_time = datetime.strptime(med['time'], "%I:%M %p").strftime("%H:%M")
        med_date = med.get('date', today) # Optional date field
        if med['status'].lower() == 'pending' and med_time == current_time and med_date == today:
            # Send SMS email reminder
            subject = f'{MEDICINE} Medication Reminder: {med["medicine"]}'
            message = f'Dear {med["patient"]},\n\nThis is your MedTrack reminder to take {med["medicine"]} at {med["time"]} today.\n\nStay healthy!'
            send_sms_email(subject, message)

            med['status'] = 'Sent' # Prevent duplicate notifications
            reminders_sent += 1

    return f'{reminders_sent} reminder(s) sent.'

```

• Update Medication Status Route:

Description: Defines /update_medication_status route to **allow patients to update the status** of their medications—Taken, Missed, or Pending.

```

@app.route('/update_medication_status', methods=[ 'POST'])
def update_medication_status():

    if session.get('role') != 'patient':
        return redirect(url_for('login'))

    for med in db['medications']:
        form_key = f"status_{med['id']}"
        if form_key in request.form:
            new_status = request.form[form_key]
            current_time = datetime.now().strftime("%H:%M")
            if new_status == "Not Yet" and current_time > med['time']:
                med['status'] = "Missed"
            else:
                med['status'] = "Completed" if new_status == "Taken" else "Pending"

    flash("Medication status updated.", "success")
    return redirect(url_for('patientdashboard'))

```

Profile Update Route:

Description: Defines /update-profile route to let **users update their name, email, and phone**, which is reflected in the session and user table.

```

@app.route('/update-profile', methods=['POST'])
def update_profile():
    if 'email' not in session or 'role' not in session:
        return redirect(url_for('login'))

    email = session['email']
    role = session['role']

    # Support for local or DynamoDB users
    users = load_users()

    if email not in users:
        flash("User not found", "error")
        return redirect(url_for(f'{role}dashboard'))

    user = users[email]
    user['name'] = request.form['name']
    user['email'] = request.form['email']
    user['phone'] = request.form['phone']

    # Update session values
    session['name'] = user['name']
    session['email'] = user['email']
    session['phone'] = user['phone']

    if not USE_DYNAMODB:
        db['users'][email] = user
    else:
        users_table.put_item(Item=user)

    flash("Profile updated successfully.", "success")
    return redirect(url_for(f'{role}dashboard'))

```

Static Info Routes:

/aboutus – Displays About Us page.

/contactus – Sends a confirmation message after user submits a contact form.

```

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

@app.route('/contactus', methods=['GET', 'POST'])
def contactus():
    if request.method == 'POST':
        flash("Message sent successfully. Thank you for contacting MedTrack!", 'success')
        return redirect(url_for('contactus'))
    return render_template('contactus.html')

```

• Logout Route:

Description: Defines /logout to **clear session data** and safely redirect the user to the login page.

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

Deployment Code:

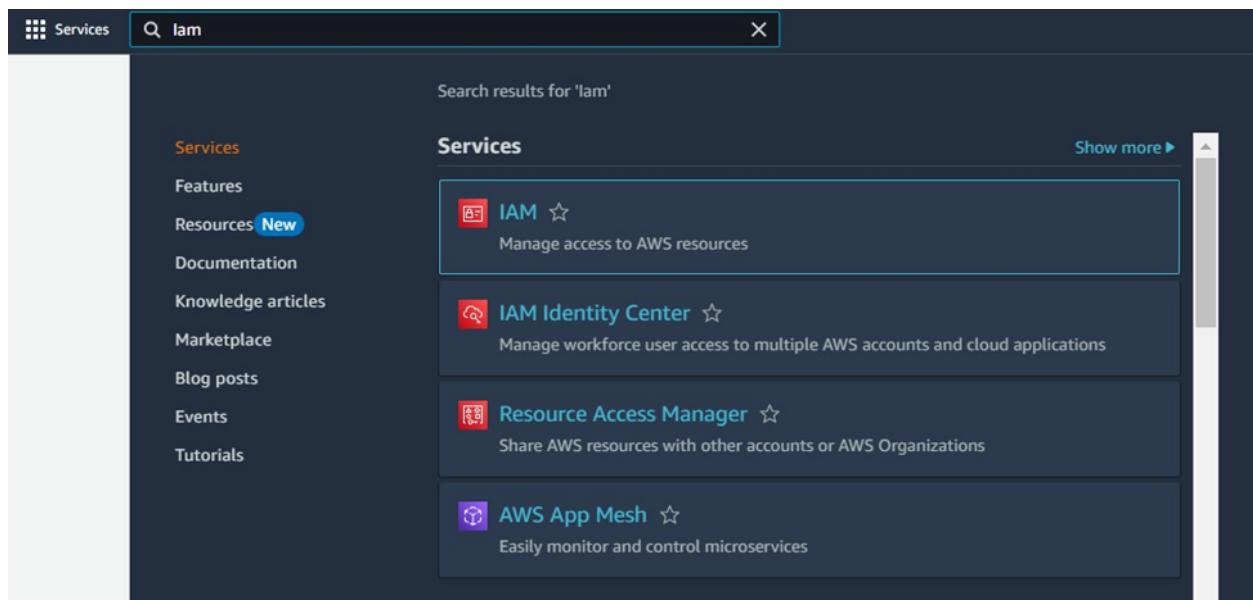
```
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

Description: start the Flask server to listen on all network interfaces (0.0.0.0) at port 80 with debug mode enabled for development and testing.

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.

Permissions policies (2/1063) [Info](#)

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

Filter by Type

Name, Policy name All types 5 matches

Policy name	Type	Description
<input checked="" type="checkbox"/> AmazonSNSFullAccess	AWS managed	Provides full access to Amazon SNS via...
<input type="checkbox"/> AmazonSNSReadOnlyAccess	AWS managed	Provides read only access to Amazon S...
<input type="checkbox"/> AmazonSNSRole	AWS managed	Default policy for Amazon SNS service...
<input type="checkbox"/> AWSElasticBeanstalkRoleSNS	AWS managed	(Elastic Beanstalk operations role) Allo...
<input type="checkbox"/> AWSSoTDeviceDefenderPublishFindingsToSN...	AWS managed	Provides messages publish access to S...

▶ Set permissions boundary - optional



aws | Search [Alt+S] Global rsoaccount-new/68034028d24717ccb9270727 @ rsosandboxnew60 ▾

IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Add permissions Info

Permissions policies (2/1063) Info

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

Filter by Type All types 6 matches

Policy name	Type	Description
<input checked="" type="checkbox"/> AmazonDynamoDBFullAccess	AWS managed	Provides full access to Amazon Dynam...
<input type="checkbox"/> AmazonDynamoDBFullAccess_v2	AWS managed	Provides full access to Amazon Dynam...
<input type="checkbox"/> AmazonDynamoDBFullAccesswithDataPipeline	AWS managed	This policy is on a deprecation path. Se...
<input type="checkbox"/> AmazonDynamoDBReadOnlyAccess	AWS managed	Provides read only access to Amazon D...
<input type="checkbox"/> AWSLambdaDynamoDBExecutionRole	AWS managed	Provides list and read access to Dynam...
<input type="checkbox"/> AWSLambdaInvocation-DynamoDB	AWS managed	Provides read access to DynamoDB Str...

▶ Set permissions boundary - optional

Create Previous Next

aws | Search [Alt+S] Global rsoaccount-new/68034028d24717ccb9270727 @ rsosandboxnew60 ▾

IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.

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

Description
Add a short explanation for this role.

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: {
2:     "Version": "2012-10-17",
3:     "Statement": [
4:         {
5:             "Effect": "Allow",
6:             "Action": [
7:                 "sts:AssumeRole"
8:             ]
9:         }
10:    ]
11: }
```



The screenshot shows the AWS IAM Roles page. A success message at the top says "Role EC2_MedTrack_Role created." Below it, the "Roles (9) Info" section lists two roles: "OrganizationAccountAccessRole" (Account: 058264256896, Last activity: 1 hour ago) and "rsoaccount-new" (Account: 058264256896, Last activity: 17 minutes ago). On the right, there are sections for "Access AWS from your non AWS workloads" (using X.509 Standard or AWS Certificate Manager Private Certificate Authority), and "Temporary credentials" (using AWS Lambda or AWS Lambda Function). A sidebar on the left includes sections for Access management, Access reports, and Credential report.

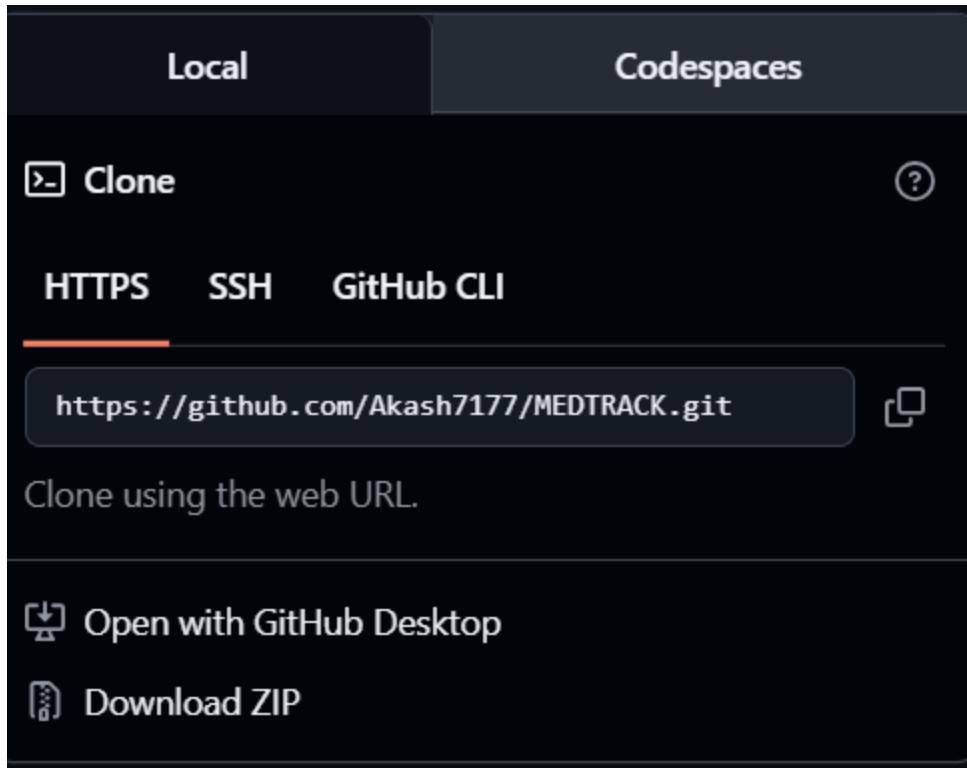
The screenshot shows the AWS EC2 Instances page. A success message at the top says "Successfully attached EC2_MedTrack_Role to instance i-0c1e4d16fde784183". The "Instances (1/2) Info" table shows one running instance named "medtrack-server" (Instance ID: i-0c1e4d16fde784183, Instance type: t2.micro, Status check: 2/2 checks passed, Public IP: 3.208.17.147). Another row for "medtrack-server" (Instance ID: i-01215ff2232615d2) is listed as pending. The instance details for "i-0c1e4d16fde784183" show its public and private IP addresses, instance state (Running), and public DNS (ec2-3-208-17-147.compute-1.amazonaws.com).

Milestone 6: EC2 Instance Setup

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

The screenshot shows a GitHub commit history. It includes three commits:

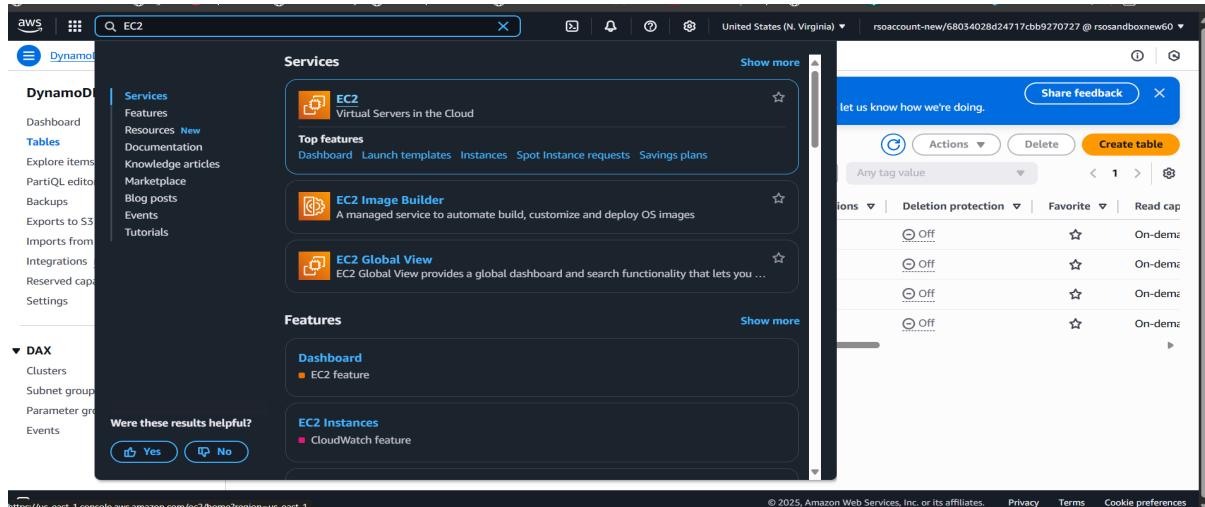
- A blue folder icon labeled "static" with the text "Initial commit" to its right.
- A blue folder icon labeled "templates" with the text "Update statistics.html" to its right.
- A blue file icon labeled "app.py" with the text "Update app.py" to its right.



- **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



EC2 Dashboard Instances

EC2 Global View Instances

Events

Instance Types

Launch Templates

Spot Requests

Savings Plans

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

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

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

Launch an instance

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

Name: medtrack-server Add additional tags

Application and OS Images (Amazon Machine Image)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below.

Search our full catalog including 1000s of application and OS images

Quick Start

Amazon Linux macOS Ubuntu Windows Red Hat SUSE Linux Debian

Summary

Number of instances: 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

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

- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as theinstancetype (free-tier eligible).
- Create and downloadthe key pair for Server access.

EC2 > Instances > Launch an instance

Instance type

Instance type: t2.micro Family: t2 1 vCPU 1 GiB Memory Current generation: true All generations Compare instance types

Key pair (login)

Key pair name - required: medtrack-server Create new key pair

Network settings

VPC - required: vpc-066d5c3fe7b1b104b 172.31.0.0/16 (default) Create new subnet

Subnet: No preference Create new subnet

Summary

Number of instances: 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

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

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



EC2 > Instances > Launch an instance

Description - required | Info
launch-wizard-1 created 2025-07-04T05:13:13.357Z

Inbound Security Group Rules

▼ Security group rule 1 (TCP, 22, 59.92.61.45/32)

Type Info ssh	Protocol Info TCP	Port range Info 22	Remove
Source type Info My IP	Name Info <input type="text"/>	Description - optional Info e.g. SSH for admin desktop	
59.92.61.45/32 X			

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

Type Info HTTP	Protocol Info TCP	Port range Info 80	Remove
Source type Info Anywhere	Source Info <input type="text"/>	Description - optional Info e.g. SSH for admin desktop	
0.0.0.0/0 X			

▼ Summary

Number of instances | Info
1

Software Image (AMI)
Amazon Linux 2 Kernel 5.10 AMI...[read more](#)
ami-000ec6c25978d599

Virtual server type (instance type)
t2.micro

Firewall (security group)
New security group

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

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

medtrack-server.pem

Success
Successfully initiated launch of instance (i-001861022fbac290)

[Launch log](#)

Next Steps

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

1 2 3 4 >

<p>Create billing and free tier usage alerts</p> <p>To manage costs and avoid surprise bills, set up email notifications for billing and free tier usage thresholds.</p> <p>Create billing alerts</p>	<p>Connect to your instance</p> <p>Once your instance is running, log into it from your local computer.</p> <p>Connect to instance</p> <p>Learn more</p>	<p>Connect an RDS database</p> <p>Configure the connection between an EC2 instance and a database to allow traffic flow between them.</p> <p>Connect an RDS database</p> <p>Create a new RDS database</p> <p>Learn more</p>	<p>Create EBS snapshot policy</p> <p>Create a policy that automates the creation, retention, and deletion of EBS snapshots</p> <p>Create EBS snapshot policy</p>	<p>Manage detailed monitoring</p> <p>Enable or disable detailed monitoring for the instance. If you enable detailed monitoring, the Amazon EC2 console displays monitoring graphs with a 1-minute period.</p> <p>Manage detailed monitoring</p>	<p>Create Load Balancer</p> <p>Create a application, network gateway or classic Elastic Load Balancer</p> <p>Create Load Balancer</p>
<p>Create AWS budget</p> <p>AWS Budgets allows you to create budgets, forecast spend, and take action on your costs and usage from a single location.</p> <p>Create AWS budget</p>	<p>Manage CloudWatch alarms</p> <p>Create or update Amazon CloudWatch alarms for the instance.</p> <p>Manage CloudWatch alarms</p>	<p>Disaster recovery for your instances</p> <p>Recover the instances you just launched into a different Availability Zone or a different Region using AWS Elastic Disaster Recovery (EDR).</p> <p>Disaster recovery for your instances</p>	<p>Monitor for suspicious runtime activities</p> <p>Amazon GuardDuty enables you to continuously monitor for malicious runtime activity and unauthorized behavior, with near real-time visibility into on-host activities occurring across your Amazon EC2 workloads.</p> <p>Monitor for suspicious runtime activities</p>	<p>Get instance screenshot</p> <p>Capture a screenshot from the instance and view it as an image. This is useful for troubleshooting an unresponsive instance.</p> <p>Get instance screenshot</p>	<p>Get system log</p> <p>View the instance's system log to troubleshoot issues.</p> <p>Get system log</p>

[View all instances](#)

- 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.

The image contains two screenshots of the AWS Management Console. The top screenshot shows the EC2 Instances page. The sidebar includes links for Dashboard, EC2 Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Images (AMIs, AMI Catalog), and Elastic Block Store (Volumes, Snapshots). The main area displays a table of instances with columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IP. Two instances are listed: 'medtrack-server' (i-0c1e4d16fde784183) and another 'medtrack-server' (i-01215fc2232615d2). The bottom screenshot shows the 'Connect to instance' page for the instance i-0c1e4d16fde784183. It features tabs for EC2 Instance Connect, Session Manager, SSH client, and EC2 serial console. The 'EC2 Instance Connect' tab is selected. It shows the instance ID and two connection options: 'Connect using a Public IP' (selected) and 'Connect using a Private IP'. Below these are fields for 'Public IPv4 address' (3.208.17.147), 'IPv6 address' (empty), 'Username' (set to 'ec2-user'), and a note about the default username. At the bottom are 'Cancel' and 'Connect' buttons.



Last login: Fri Jul 4 07:19:48 2025 from ec2-18-206-107-28.compute-1.amazonaws.com
Amazon Linux 2
AL2 End of Life is 2026-06-30.
A newer version of Amazon Linux is available!
Amazon Linux 2023, GA and supported until 2028-03-15.
<https://aws.amazon.com/linux/amazon-linux-2023/>

```
[ec2-user@ip-172-31-17-129 ~]$ sudo su
```

i-0c1e4d16fde784183 (medtrack-server)
Public IPs: 3.208.17.147 Private IPs: 172.31.17.129

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 installpython3  
git sudo pip3 install flaskboto3
```

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/ushasree26/MedTrack.git>'

- This will download your project to the EC2 instance.

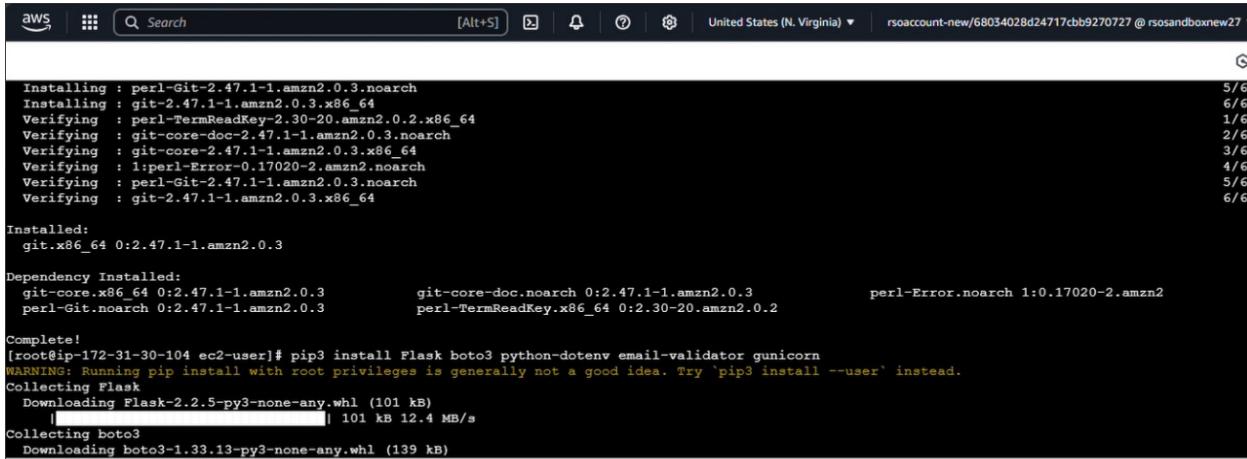
To navigate to the project directory, run the following command:

```
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



```

Installing : perl-Git-2.47.1-1.amzn2.0.3.noarch 5/6
Installing : git-2.47.1-1.amzn2.0.3.x86_64 6/6
Verifying  : perl-TermReadKey-2.30-20.amzn2.0.2.x86_64 1/6
Verifying  : git-core-doc-2.47.1-1.amzn2.0.3.noarch 2/6
Verifying  : git-core-2.47.1-1.amzn2.0.3.x86_64 3/6
Verifying  : perl-Error-0.17020-2.amzn2.noarch 4/6
Verifying  : perl-Git-2.47.1-1.amzn2.0.3.noarch 5/6
Verifying  : git-2.47.1-1.amzn2.0.3.x86_64 6/6

Installed:
git.x86_64 0:2.47.1-1.amzn2.0.3

Dependency Installed:
git-core.x86_64 0:2.47.1-1.amzn2.0.3          git-core-doc.noarch 0:2.47.1-1.amzn2.0.3            perl-Error.noarch 1:0.17020-2.amzn2
perl-Git.noarch 0:2.47.1-1.amzn2.0.3          perl-TermReadKey.x86_64 0:2.30-20.amzn2.0.2

Complete!
[root@ip-172-31-30-104 ec2-user]# pip3 install Flask boto3 python-dotenv email-validator gunicorn
WARNING: Running pip install with root privileges is generally not a good idea. Try 'pip3 install --user' instead.
Collecting Flask
  Downloading Flask-2.2.5-py3-none-any.whl (101 kB)
Collecting boto3
  Downloading boto3-1.33.13-py3-none-any.whl (139 kB)

```

Verify the Flask app is running:

<http://54.226.228.213:5000>

- Run the Flask app on the EC2 instance

Access the website through:

Public IPs: <http://54.226.228.213:5000>

Milestone 8: Testing and Deployment:

• Activity 8.1:

Conduct functional testing to verify user sign-up, login, appointment booking, prescription generation, medication reminders, and SNS notifications.

Index Page:

Akash7177/MED | Student - Skill Work | troven.in/student | trovenin/student | Instances | EC2 | EC2 Instance Com | MEDTRACK - Smart Internz | + | - | X

Not secure 54.226.228.213:5000

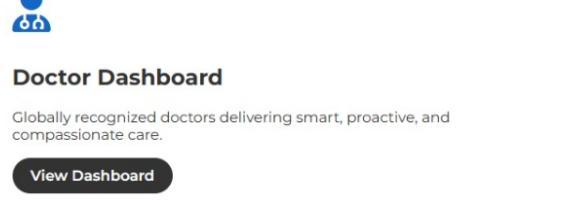
MEDTRACK

Contact Us About Us Book Appointment Login Sign Up



Doctor Dashboard
Globally recognized doctors delivering smart, proactive, and compassionate care.

[View Dashboard](#)



Patient Dashboard

Not secure 54.226.228.213:5000

Why Choose MEDTRACK?



Cloud Security

HIPAA-compliant, AWS-hosted, end-to-end encrypted data storage.



Smart Alerts

Timely reminders to help you stay on top of every dose and checkup.



Family Sharing

Give trusted access to doctors or caregivers — securely and easily.

Stay Informed

Get health tips, platform updates, and appointment alerts directly to your inbox.

Your email

[Subscribe](#)

© 2025 MEDTRACK. All rights reserved. | [Privacy Policy](#) | [Terms](#)

Signup Page:

A Not secure 54.226.228.213:5000/signup

Welcome to MedTrack

Problem: Many patients miss doses or struggle with medication routines, affecting recovery and health.

Solution: MedTrack ensures patients stay on schedule with reminders, secure tracking, and doctor communication.

- Secure user login & registration
- Medication tracking & dosage logs
- SMS/Email alerts (AWS SNS)
- Cloud storage via AWS DynamoDB
- Deployed with AWS EC2 & IAM

Email

 Password

 Phone Number

 Role
 Select
 Reminder Type
 Select
[Create Account](#)

Already have an account? [Login here](#)

login page:



Full Name

 Email Address

 Password
 
 Select your role
 Select

Forgot password?

Log In

 Google
 Facebook

Don't have an account? [Create one](#)

About Us page:



Not secure 54.226.228.213:5000/aboutus

Meet MedTrack

Your trusted partner for smarter health, digital care, and intelligent medication tracking.

MedTrack is an advanced, cloud-based health platform that transforms the way patients manage their medications. With built-in AI, smart reminders, and seamless doctor coordination, we're creating a smarter future in healthcare.

From prescription tracking to appointment scheduling and doctor-patient communication, MedTrack provides a complete digital care experience — accessible, secure, and easy to use.

We are built on cutting-edge AWS services like EC2 (scalable compute), DynamoDB (fast NoSQL database), SNS (notification system), and IAM (robust access control) to ensure privacy, scalability, and performance.

Our Core Team

Akash
Founder & AI Engineer

Dr. v kanaka durga
Chief Medical Advisor

Ramesh
Lead Backend Developer

ContactUs Page:

Not secure 54.226.228.213:5000/contactus

Talk to Us

We're here to help with anything related to MedTrack.

Book Appointment Page:

Book Your Appointment - MedTrack

Full Name

Gender

Phone Number

Age

Department / Specialization

Describe Your Problem

-- Select Department --

Describe Your Problem

Select Available Doctor

Preferred Date
 ...

Preferred Time
 ... AM

Book Appointment Book Appointment

(Doctor)Explore Dashboard Page:

Our Expert Medical Team

Certified. Compassionate. Committed to Your Health.

At MedTrack, we take pride in the team of elite professionals who bring passion and excellence to patient care. Each of our doctors is not only an expert in their field but also dedicated to building a better health experience. Discover the brilliant minds behind MedTrack and see why thousands trust us with their well-being.



(Patient)Explore Dashboard Page:

What Our Patients Say About MedTrack

Real patients. Real stories. Read what people across India say about how MedTrack helps them stay healthier every day.



Anjali arora
Age 42 · Bengaluru
★★★★★

"MedTrack's reminders are super helpful for my diabetic patients. Even my mother uses it now and never misses her insulin shot!"



Rahul
Age 29 · Pune
★★★★★

"I track my asthma meds daily with MedTrack. The UI is clean, and AWS SMS alerts are timely. It feels like I have a health assistant."



Sneha pinili
Age 34 · Hyderabad
★★★★★

"After surgery, my routine needed structure. MedTrack gave that with a friendly UI and daily reports. Worth recommending!"



Sandeep
Age 37 · Delhi
★★★★★

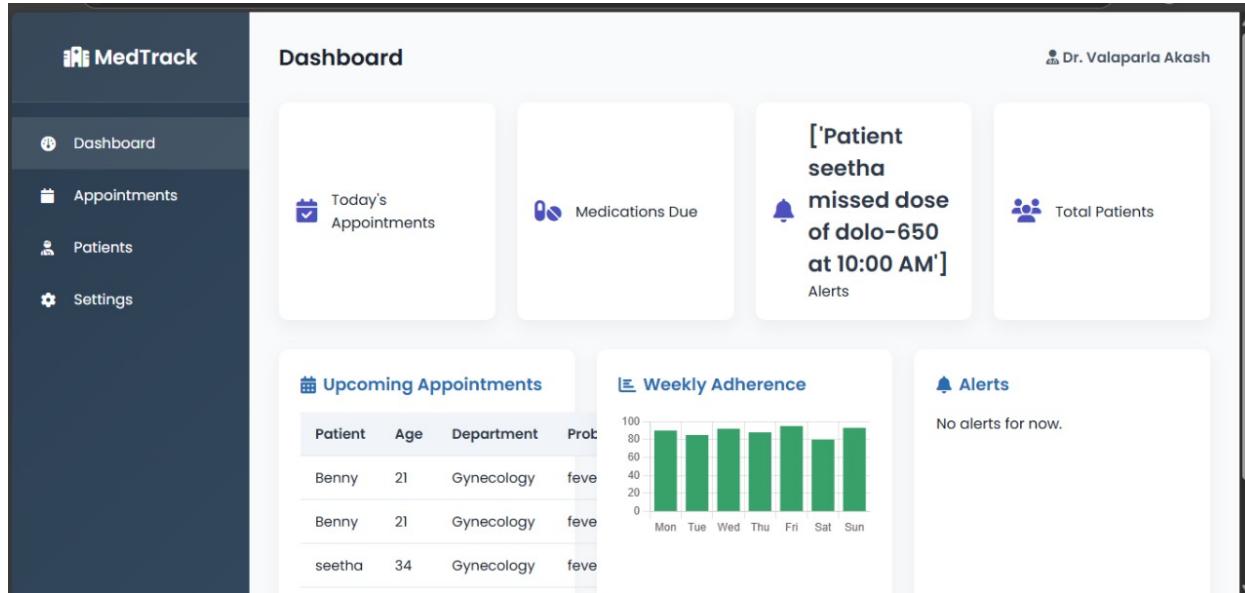
"The data security in MedTrack is top-notch. I use it for my daughter's allergy schedule too. Simple and reliable app."



Noor
Age 30 · Lucknow
★★★★★

"Using MedTrack is like having a digital nurse. It's perfect for my busy schedule and keeps my family organized."

Doctor Dashboard:



MedTrack

Dashboard

Dr. Valaparla Akash

Today's Appointments

Medications Due

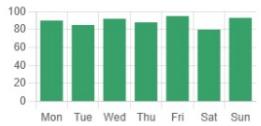
[Patient seetha missed dose of dolo-650 at 10:00 AM]

Total Patients

Upcoming Appointments

Patient	Age	Department	Prob.
Benny	21	Gynecology	feve
Benny	21	Gynecology	feve
seetha	34	Gynecology	feve

Weekly Adherence

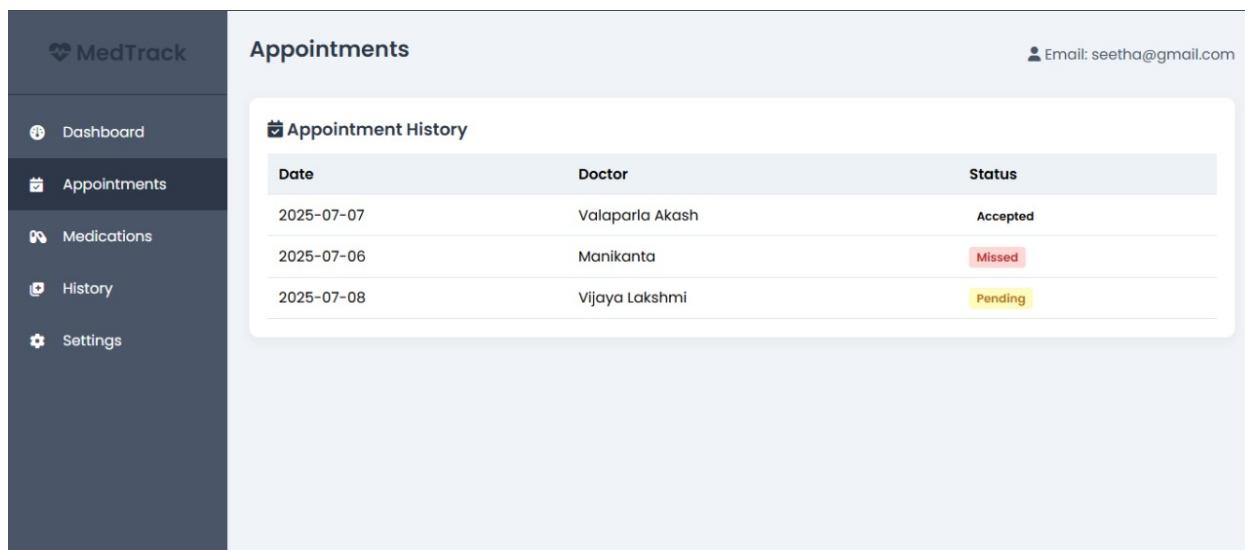


Day	Adherence
Mon	85
Tue	80
Wed	85
Thu	85
Fri	85
Sat	75
Sun	85

Alerts

No alerts for now.

Patient Dashboard:



MedTrack

Appointments

Email: seetha@gmail.com

Appointment History

Date	Doctor	Status
2025-07-07	Valaparla Akash	Accepted
2025-07-06	Manikanta	Missed
2025-07-08	Vijaya Lakshmi	Pending

Add Medication Page:

 MedTrack - Patient

Add Your Medication

Medicine Name

Dosage

Time of Day

Date

Number of Days

Status

[+ Add Medication](#)

[← Back to Dashboard](#)

Prescribe Medication:

 Prescribe Medication

Patient Name: chandana
Age: 24
Gender: Female
Problem: fever
Department: Gynecology

Medicine

Dosage

Days

Time

Patient

[Save Prescription](#)

Exit:

Session Ended

Please close this tab.

Conclusion:

The **MedTrack Healthcare Assistant** has been successfully developed and deployed using a robust cloud-based architecture. By leveraging **AWS services** such as **EC2 for hosting**, **DynamoDB for data storage**, **SNS for real-time email notifications**, and **IAM for secure access control**, the platform



ensures reliable, secure, and scalable access to essential healthcare services for both patients and doctors.

This system addresses common healthcare challenges by allowing **patients to book appointments, receive timely medication reminders**, and track their prescriptions — all from a unified platform. **Doctors can manage appointments, issue prescriptions, and monitor patient adherence**, improving the overall quality of care.

The **cloud-native approach** ensures seamless scalability and high availability, making it adaptable to increasing user demand. The integration of **Flask with AWS services** guarantees efficient backend operations, including secure user authentication and notification handling.

During testing, all core functionalities — from user registration and login to **SNS-powered reminders and real-time medication tracking** — were verified for reliability and accuracy.

In conclusion, **MedTrack offers a modern, cloud-powered solution** for improving patient-doctor engagement, simplifying health management tasks, and enhancing the overall healthcare experience. This project showcases how cloud technologies can effectively address real-world challenges in the medical field.