

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. The system allows users to manage appointments, medical records, and diagnosis reports in one centralized platform, improving the overall healthcare experience for both patients and doctors. All patient and doctor data, including appointment details and medical histories, are stored in AWS DynamoDB.

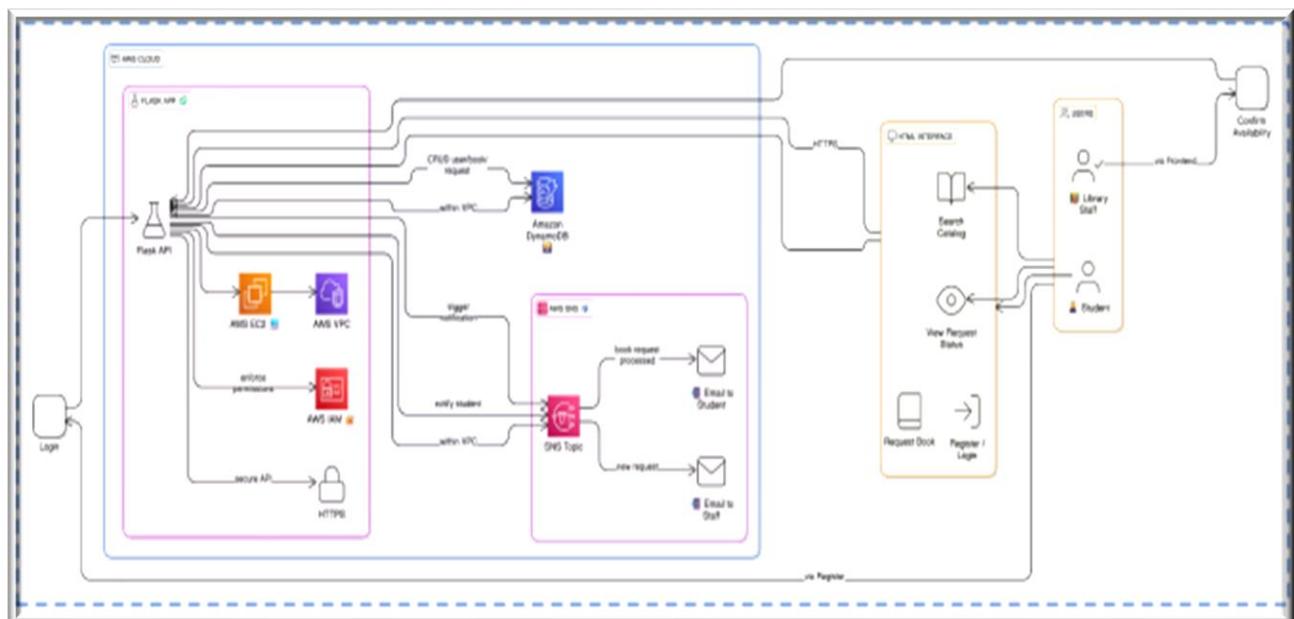
Scenario 2: Secure User Management with IAM

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.

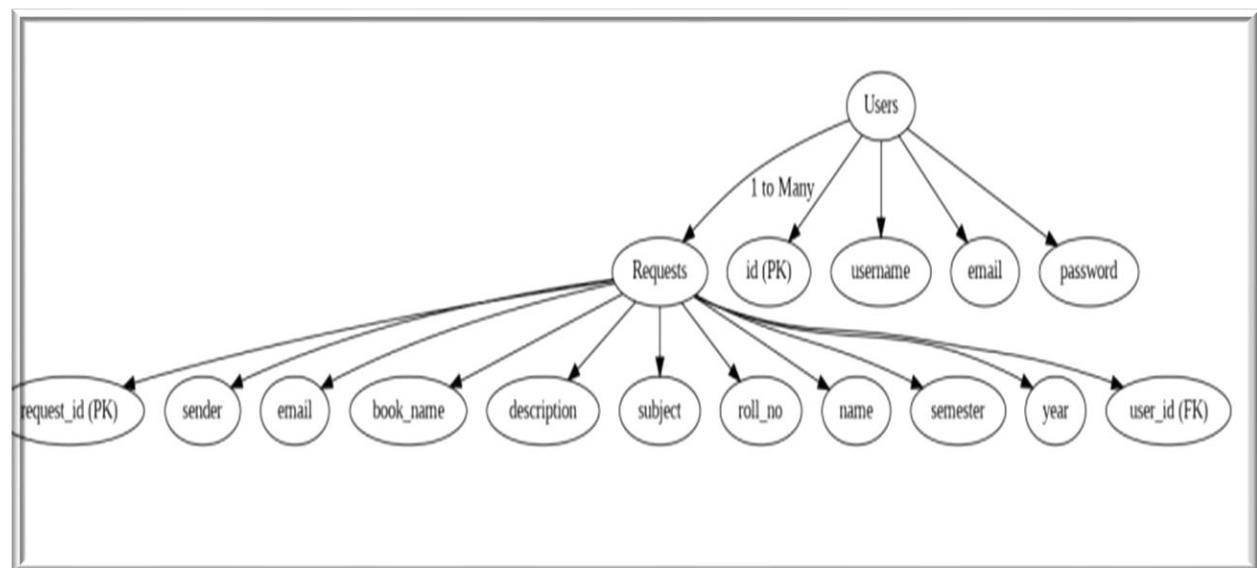
Scenario 3: Easy Access to Medical History and Resources

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



ER DIAGRAM



PROJECT WORK FLOW

1.AWSAccountSetupandLogin

Activity1.1:SetupanAWSaccountifnotalreadydone.

Activity1.2:LogintotheAWSManagementConsole

2.DynamoDBDatabaseCreationandSetup

Activity2.1:CreateaDynamoDBTable.

Activity2.2:ConfigureAttributesforUserDataandBookRequests.

3.SNSNotificationSetup

Activity3.1:CreateSNStopicsforbookrequestnotifications.

Activity3.2:SubscribeusersandlibrarystafftoSNSemailnotifications.

4.BackendDevelopmentandApplicationSetup

Activity4.1:DeveloptheBackendUsingFlask.

Activity4.2:IntegrateAWServicesUsingboto3.

5.IAMRoleSetup

Activity5.1:CreateIAMRole

Activity5.2:AttachPolicies

6.EC2InstanceSetup

Activity6.1:LaunchanEC2instancetohosttheFlaskapplication.

Activity6.2:ConfiguresecuritygroupsforHTTP, andSSHaccess.

7.DeploymentonEC2

Activity7.1:UploadFlaskFiles

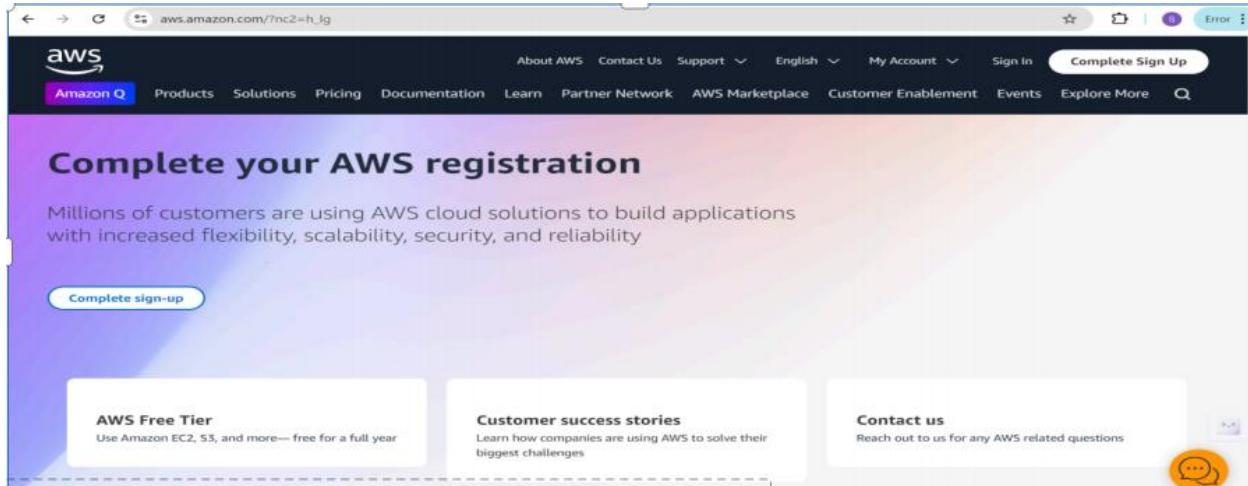
Activity7.2:RuntheFlaskApp

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

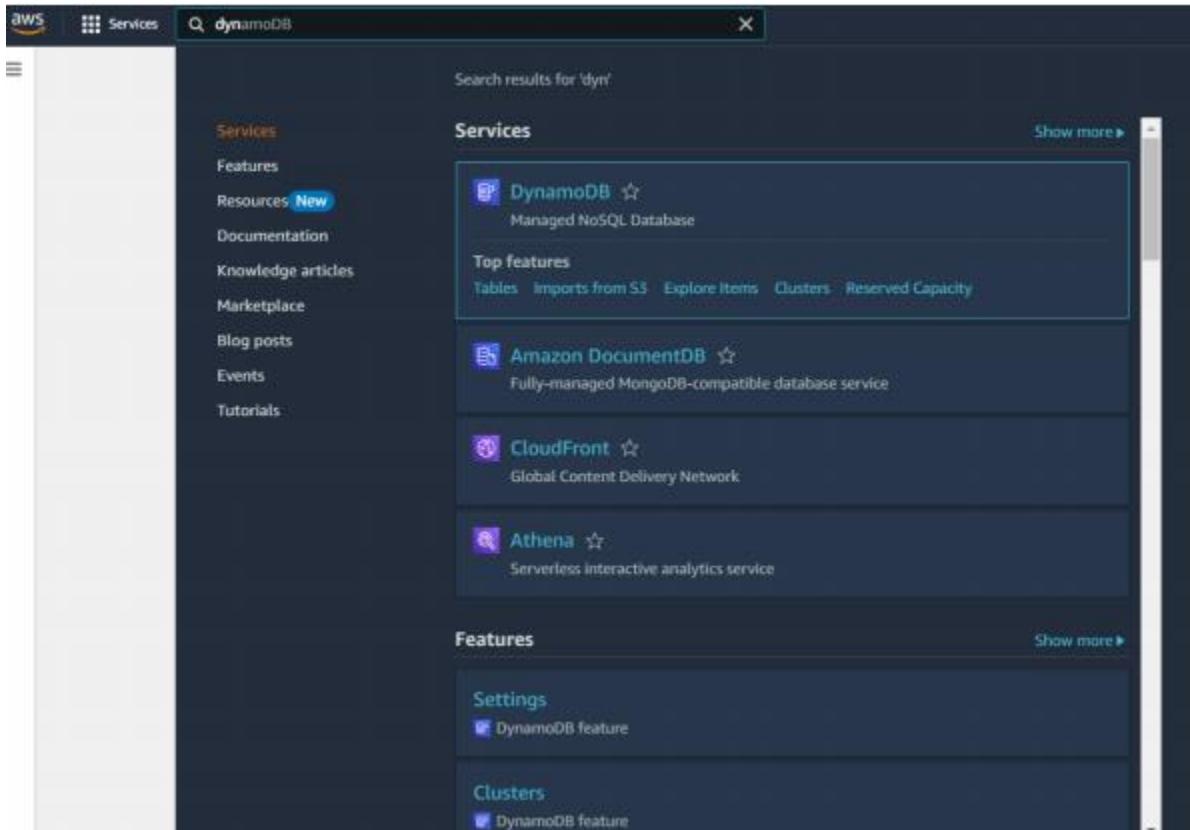
- After setting up your account, log in to the [AWS Management Console](#)

A screenshot showing two side-by-side sections. On the left is the "Sign in" page for AWS, featuring a "Root user" selection (selected) and an "IAM user" option. It also includes a "Root user email address" field containing "username@example.com" and a "Next" button. A small note at the bottom states: "By continuing, you agree to the AWS Customer Agreement or other agreement for AWS services, and the Privacy Notice. This site uses essential cookies. See our Cookies Notice for more information." On the right is a promotional section for the "AI Use Case Explorer". The title "AI Use Case Explorer" is in large white font, followed by the text "Discover AI use cases, customer success stories, and expert-curated implementation plans". A "Explore now >" button is at the bottom. A yellow "Ask a question" icon is located in the top right corner of this section.

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



A screenshot of the DynamoDB dashboard. The left sidebar has sections for 'Dashboard', 'Tables', 'Explore items', 'PartiQL editor', 'Backups', 'Exports to S3', 'Imports from S3', 'Integrations New', 'Reserved capacity', and 'Settings'. It also has a 'DAX' section with 'Clusters', 'Subnet groups', 'Parameter groups', and 'Events'. The main content area is titled 'Dashboard'. It contains two tables: 'Alarms (0) Info' and 'DAX clusters (0) Info'. The 'Create resources' section on the right includes a 'Create table' button and information about Amazon DynamoDB Accelerator (DAX). The 'What's new' section at the bottom right shows a recent update about AWS Cost Management providing purchase recommendations for Amazon DynamoDB.

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

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

Table 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.

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.

Notifications 0 0 1 1 0

Tables (4) Info

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection
MedTrackAppointments	Active	appointment_id (S)	-	0	0	Off
MedTrackDoctors	Active	email (S)	-	0	0	Off
MedTrackPatients	Active	email (S)	-	0	0	Off
MedTrackPrescriptions	Active	prescription_id (S)	-	0	0	Off

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 the AWS search interface with the query 'sns' entered. The search results page displays the 'Services' section, which includes the 'Simple Notification Service' (marked as new). Other services listed include Route 53 Resolver, Route 53, AWS End User Messaging, and AWS Lambda. Below the services, the 'Features' section lists Events, SMS, and Hosted zones, each with a corresponding icon and brief description.

The screenshot shows the Amazon SNS dashboard. On the left, there is a sidebar with links to Dashboard, Topics, Subscriptions, Mobile (Push notifications, Text messaging (SMS)), and Application Integration. The main content area features a large banner for 'Amazon Simple Notification Service' with the tagline 'Pub/sub messaging for microservices and serverless applications.' A 'New Feature' notice at the top right states that SNS now supports in-place message archiving and replay for FIFO topics. To the right of the banner, there is a 'Create topic' form. It includes fields for 'Topic name' (with placeholder 'MyTopic') and a 'Next step' button. Below the form, there is a link to 'Start with an overview'. At the bottom right, there is a 'Pricing' section.

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

The screenshot shows the Amazon SNS Topics page. A blue banner at the top right reads "New Feature" and says "Amazon SNS now supports in-place message archiving and replay for FIFO topics. Learn more". The main area has a table titled "Topics (0)" with columns for Name, Type, and ARN. There are buttons for Edit, Delete, Publish message, and Create topic. A search bar and a "Create topic" button are also present.

> Choose Standard type for general notification use cases and Click on Create

The screenshot shows the "Create topic" page under the "Topics" section. In the "Details" tab, the "Type" dropdown is set to "Standard". The "Standard" option is selected and highlighted with a blue border. It lists features like best-effort message ordering, at-least once delivery, highest throughput, and support for various protocols. Other options like "FIFO (first-in, first-out)" are shown with their own bullet points. Below the type selection, there are fields for "Name" (BookRequestNotifications) and "Display name - optional" (My Topic). The page also includes sections for Access policy, Data protection policy, Delivery policy (HTTP/S), Delivery status logging, Tags, and Active tracing, each with an "optional" note and a "Info" link. At the bottom right are "Cancel" and "Create topic" buttons.

> Configure the SNS topic and note down the **Topic ARN**

The screenshot shows the AWS SNS Topics page. In the top navigation bar, there are links for 'Amazon SNS', 'Topics', and 'project'. A blue banner at the top right indicates a 'New Feature' about High Throughput FIFO topics. Below it, a green banner says 'Topic project created successfully.' with the message 'You can create subscriptions and send messages to them from this topic.' There are three buttons: 'Edit', 'Delete', and 'Publish message'. On the left sidebar, under 'Mobile', there are links for 'Push notifications' and 'Text messaging (SMS)'. The main content area is titled 'project'.

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 'Create subscription' form. At the top, it says 'Amazon SNS > Subscriptions > Create subscription'. The form has several sections: 'Details' (Topic ARN: arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications, Protocol: Email, Endpoint: instantlibrary2@gmail.com), 'Subscription filter policy - optional' (info: This policy filters the messages that a subscriber receives.), and 'Redrive policy (dead-letter queue) - optional' (info: Send undeliverable messages to a dead-letter queue.). At the bottom right, there are 'Cancel' and 'Create subscription' buttons.

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

AWS Notification - Subscription Confirmation Spam x

medtrack <no-reply@sns.amazonaws.com>
to me ▾

Thu, Jul 3, 11:53 PM (2 days 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-1:851725243544:project`

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](#)



Simple Notification Service

Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:

`arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications:d78e0371-9235-404d-952c-85c2743607c4`

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

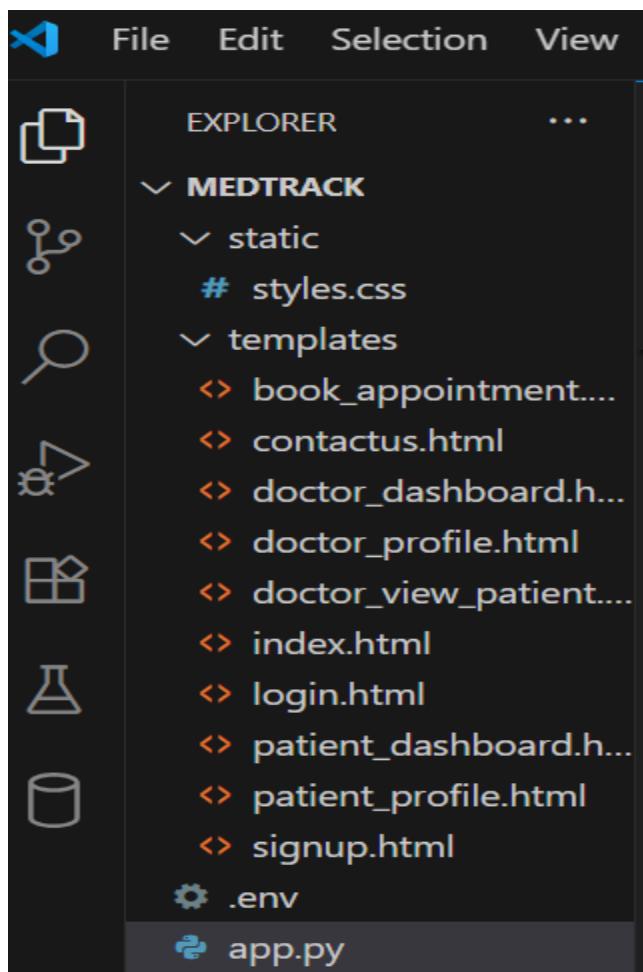
The screenshot shows the AWS SNS console with the following details:

- Subscription ID:** 40590805-9aeb-4094-8d6e-fbfe3ff6800f
- Topic:** project
- Endpoint:** salmaume567@gmail.com
- Status:** Confirmed
- Protocol:** EMAIL

A blue banner at the top indicates: "Amazon SNS now supports High Throughput FIFO topics. Learn more" with a link icon.

Milestone 4: Backend Development and Application Setup

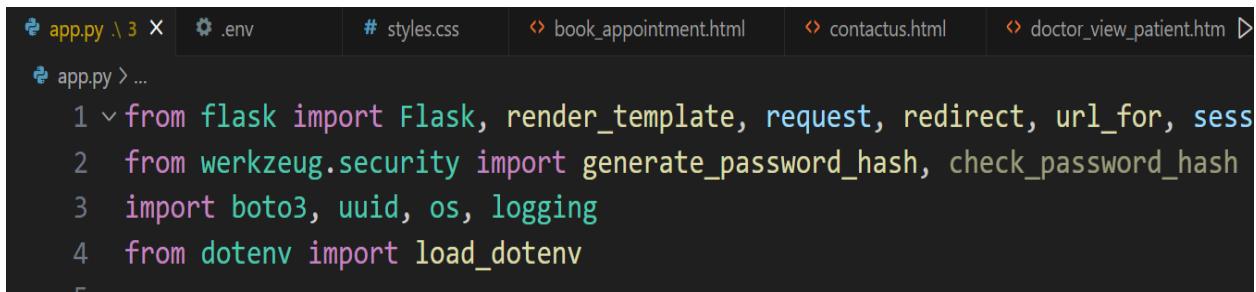
- **Activity 4.1: Develop the backend using Flask**
 - File Explorer Structure



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 home, login, register, subject-specific pages (e.g., computer_science.html, data_science.html), and utility pages (e.g., request-form.html, statistics.html)

Description of the code :

- Flask App Initialization



```
# app.py \ 3 X .env # styles.css book_appointment.html contactus.html doctor_view_patient.htm
app.py > ...
1 v from flask import Flask, render_template, request, redirect, url_for, sess
2 from werkzeug.security import generate_password_hash, check_password_hash
3 import boto3, uuid, os, logging
4 from dotenv import load_dotenv
```

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

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

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

```
# ----- AWS Setup -----
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
sns = boto3.client('sns', region_name='us-east-1')
SNS_TOPIC_ARN = os.getenv("SNS_TOPIC_ARN")

# DynamoDB Tables
doctor_table = dynamodb.Table("MedTrackDoctors")
patient_table = dynamodb.Table("MedTrackPatients")
appointment_table = dynamodb.Table("MedTrackAppointments")
prescription_table = dynamodb.Table("MedTrackPrescriptions")
```

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

- SNS Connection

```
# SNS Topic ARN (create the SNS topic in AWS and provide the ARN here)
sns = boto3.client('sns', region_name='ap-south-1')
sns_topic_arn = 'arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications'

# Email settings (for sending emails)
SMTP_SERVER = "smtp.gmail.com"
SMTP_PORT = 587
SENDER_EMAIL = "instantlibrary2@gmail.com"
SENDER_PASSWORD = "luut dsih nyvq dgzv" # Your app password

# Function to send email
def send_email(to_email, subject, body):
    msg = MIMEMultipart()
    msg['From'] = SENDER_EMAIL
    msg['To'] = to_email
    msg['Subject'] = subject
    msg.attach(MIMEText(body, 'plain'))

    try:
        server = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
        server.starttls()
        server.login(SENDER_EMAIL, SENDER_PASSWORD)
        text = msg.as_string()
        server.sendmail(SENDER_EMAIL, to_email, text)
        server.quit()
        print("Email sent successfully")
    except Exception as e:
        print(f"Failed to send email: {e}")
```

Description: Configure SNS to send notifications when a book request is submitted. Paste your stored ARN link in the sns_topic_arn space, along with the region_name where the SNS topic is created. Also, specify the chosen email service in SMTP_SERVER (e.g., Gmail, Yahoo, etc.) and enter the subscribed email in the SENDER_EMAIL section.

Routes for Web Pages

- Home Route

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

@app.route("/signup", methods=["GET", "POST"])
def signup():
    if request.method == "POST":
        role = request.form.get("role")
        name = request.form.get("name")
        email = request.form.get("email")
        phone = request.form.get("phone")
        gender = request.form.get("gender")
        password = request.form.get("password")

        table = doctor_table if role == "doctor" else patient_table
        if table.get_item(Key={"email": email}).get("Item"):
            flash("Email already registered.")
            return redirect(url_for("signup"))
```

Description: define /register route to validate registration form fields, hash the user password using Bcrypt, store the new user in DynamoDB with a login count, and send an SNS notification on successful registration

Login Route (GET/POST):

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

        table = doctor_table if role == "doctor" else patient_table
        user = table.get_item(Key={"email": email}).get("Item")

        if not user or user["password"] != password:
            flash("Invalid credentials.")
            return redirect(url_for("login"))

        session.update({"name": user["name"], "email": user["email"], "role": role})
        return redirect(url_for("doctor_dashboard" if role == "doctor" else patient_dashboard))
```

Doctor Dashboard route:

```
@app.route("/doctor_dashboard")
def doctor_dashboard():
    if session.get("role") != "doctor": return redirect(url_for("login"))

    name = session.get("name")
    write_mode = request.args.get("write_mode") == "yes"
    show_all = request.args.get("show_all") == "yes"

    upcoming, completed, patient_list = [], [], []
    total = 0

    items = appointment_table.scan().get("Items", [])
    for item in items:
        if item.get("doctor") != name: continue
        total += 1
        if item.get("prescription"):
```

Description: The doctor_dashboard route in app.py displays the doctor's main dashboard after login. It fetches appointment data from DynamoDB and categorizes

them as upcoming or completed. The route passes this data to doctor_dashboard.html, which shows a summary of appointments and patient interactions.

Patient dashboard Route

```
@app.route("/patient_dashboard", endpoint="patient_dash")
def patient_dashboard():
    if session.get("role") != "patient": return redirect(url_for("login"))
    name = session.get("name")

    show_all = request.args.get("show_all")
    prescription_success = request.args.get("prescription_success")
    (variable) appointments: Any
    appointments = appointment_table.scan()["Items"]
    prescriptions = prescription_table.scan()["Items"]

    upcoming = [a for a in appointments if a.get("patient") == name and a.
    completed = [a for a in appointments if a.get("patient") == name and a.
    patient_prescriptions = [p for p in prescriptions if p.get("patient")]

    return render_template("patient_dashboard.html", name=name,
```

Description: The patient_dashboard route displays the logged-in patient's dashboard with upcoming and completed appointments. It retrieves appointment and prescription data from DynamoDB, filtering it based on the patient's name.

Doctor view Patients route:

```
@app.route("/doctor_view_patients")
def doctor_view_patients():
    if session.get("role") != "doctor": return redirect(url_for("login"))
        (variable) session: SessionMixin
    name = session.get("name")
    items = appointment_table.scan().get("Items", [])
    patients = [item for item in items if item.get("doctor") == name and it

    return render_template("doctor_view_patients.html", name=name, patients

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

    doctor = session.get("name")
    patient = request.form["patient"]
    prescription = request.form["prescription"]
```

Doctor Profile Route:

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

    email = session.get("email")

    if request.method == "POST":
        doctor_table.update_item(
            Key={"email": email},
            UpdateExpression="SET #n = :name, phone = :phone, gender = :gender",
            ExpressionAttributeNames={"#n": "name"},
            ExpressionAttributeValues={
                ":name": request.form["name"],
                ":phone": request.form["phone"],
                ":gender": request.form["gender"],
                ":pwd": generate_password_hash(request.form["password"])
            })
        session["name"] = request.form["name"]
```

Patient Profile Route:

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

    email = session.get("email")
    if request.method == "POST":
        patient_table.put_item(Item={
            "email": email,
            "name": request.form["name"],
            "phone": request.form["phone"],
            "gender": request.form["gender"],
            "password": generate_password_hash(request.form["password"]),
            "role": "patient"
        })
        flash("Profile updated.")
        return redirect(url_for("patient_profile"))
```

Book Appointment Route:

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

    name = session.get("name")
    if request.method == "POST":
        appointment = {
            "id": str(uuid.uuid4()),
            "patient": name,
            "doctor": request.form["doctor"],
            "date": request.form["date"],
            "time": request.form["time"],
            "problem": request.form["problem"],
            "status": "accepted",
            "prescription": ""
        }
```

Contact Route:

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

Deployment code

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

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.

Services X

Search results for 'Iam'

Services

Features

Resources **New**

Documentation

Knowledge articles

Marketplace

Blog posts

Events

Tutorials

IAM ☆ Manage access to AWS resources

IAM Identity Center ☆ Manage workforce user access to multiple AWS accounts and cloud applications

Resource Access Manager ☆ Share AWS resources with other accounts or AWS Organizations

AWS App Mesh ☆ Easily monitor and control microservices

This screenshot shows the AWS Services search results for the term 'Iam'. On the left, there's a sidebar with links like Services, Features, Resources (New), Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials. The main area displays four services: IAM, IAM Identity Center, Resource Access Manager, and AWS App Mesh. Each service has a thumbnail icon, a name, a star rating, and a brief description.

Identity and Access Management (IAM) X

Dashboard

IAM > Roles

Roles (6) Info

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

Role name ▲ | Trusted entities | Last activity ▼

This screenshot shows the IAM Roles management page. It lists six roles and provides options to search, sort, and delete them. A prominent orange 'Create role' button is visible at the top right.

Step 1 Select trusted entity

Step 2 Add permissions

Step 3 Name, review, and create

Select trusted entity

Trusted entity type

AWS service Allows AWS services like S3, Lambda, or others to perform actions in this account.

IAM account Allows another AWS account belonging to you or a third party to perform actions in this account.

Web identity Allows users selected by the specified external web identity provider to assume this role to perform actions in this account.

Custom trust policy Creates a custom trust policy to enable others to perform actions in this account.

Use case

Service or use case: EC2

Choose a use case for the specified service.

Use case: EC2

EC2 - IAM roles for AWS services on your behalf

EC2 Role for AWS Systems Manager

EC2 Role for AWS Lambda

EC2 - Launch spot instances and attach tags to the launched instances on your behalf

EC2 - Spot Fleet Auto Scaling

EC2 - Spot Fleet Tagging

EC2 - Launch spot instances and attach tags to the launched instances on your behalf

EC2 - Spot Fleet

EC2 - Schedule instances to launch and manage spot fleet instances on your behalf

EC2 - Scheduled instances

This screenshot shows the 'Select trusted entity' step of the IAM Role creation wizard. It allows users to choose the type of trusted entity (AWS service, IAM account, Web identity, or Custom trust policy) and select a specific service or use case (e.g., EC2). The 'EC2' use case is selected.

The screenshot shows the 'Add permissions' step of creating a new IAM role. The left sidebar shows 'Step 1 Select trusted entity' and 'Step 2 Add permissions' (which is currently selected). The main area is titled 'Add permissions' with a 'Permissions policies (1/955)' link. A search bar contains 'AmazonDynamoDB'. The results show two AWS managed policies: 'AmazonDynamoDBFullAccess' and 'AmazonDynamoDBReadOnlyAccess', both of which are selected. Below the list is a 'Set permissions boundary - optional' section. At the bottom right are 'Cancel', 'Previous', and 'Next' buttons, with 'Next' being highlighted.

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.

The screenshot shows the 'Add permissions' step of creating a new IAM role. The left sidebar shows 'Step 1 Select trusted entity' and 'Step 2 Add permissions' (which is currently selected). The main area is titled 'Add permissions' with a 'Permissions policies (2/955)' link. A search bar contains 'sns'. The results show five AWS managed policies: 'AmazonSNSFullAccess' (selected), 'AmazonSNSReadOnlyAccess', 'AmazonSNSSecurityRole', 'AWSLambdaBasicExecutionRoleSNSS', and 'AWSIoTDeviceDefenderPublishFindingsToSNSMigrationAction'. Below the list is a 'Set permissions boundary - optional' section. At the bottom right are 'Cancel', 'Previous', and 'Next' buttons, with 'Next' being highlighted.

Screenshot of the AWS IAM Roles page showing 14 roles.

Identity and Access Management (IAM)

Roles (14) Info

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

Role name	Trusted entities	Last activity
AWSServiceRoleForAccessAnalyzer	AWS Service: access-analyzer (Service-Linked)	373 days ago
AWSServiceRoleForAPIGateway	AWS Service: ops.apigateway (Service-Linked)	-
AWSServiceRoleForApplicationAutoScaling_DynamoDBTable	AWS Service: dynamodb.application	373 days ago
AWSServiceRoleForAutoScaling	AWS Service: autoscaling (Service-Linked)	400 days ago
AWSServiceRoleForCloudWatchEvents	AWS Service: events (Service-Linked)	390 days ago
AWSServiceRoleForECS	AWS Service: ecs (Service-Linked Role)	176 days ago
AWSServiceRoleForElasticLoadBalancing	AWS Service: elasticloadbalancing (Service-Linked)	400 days ago

Screenshot of the AWS IAM Roles page showing 15 roles, with a success message for creating a new role.

Identity and Access Management (IAM)

Roles (1/15) Info

Role EC2_MedTrack_Role created.

Role EC2_MedTrack_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.

Role name	Trusted entities	Last activity
EC2_MedTrack_Role	AWS Service: ec2	-

Roles Anywhere Info

Authenticate your non AWS workloads and securely provide access to AWS services.

Access AWS from your non AWS

X.509 Standard

Temporary credentials

Milestone 6: EC2 Instance Setup

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

The screenshot shows a GitHub repository named "MEDTRACK" (Public). The repository contains the following files:

- static
- templates
- .env
- app.py

Commits listed on the page:

- Initial commit (for static)
- Update statistics.html (for templates)
- Update app.py (for app.py)

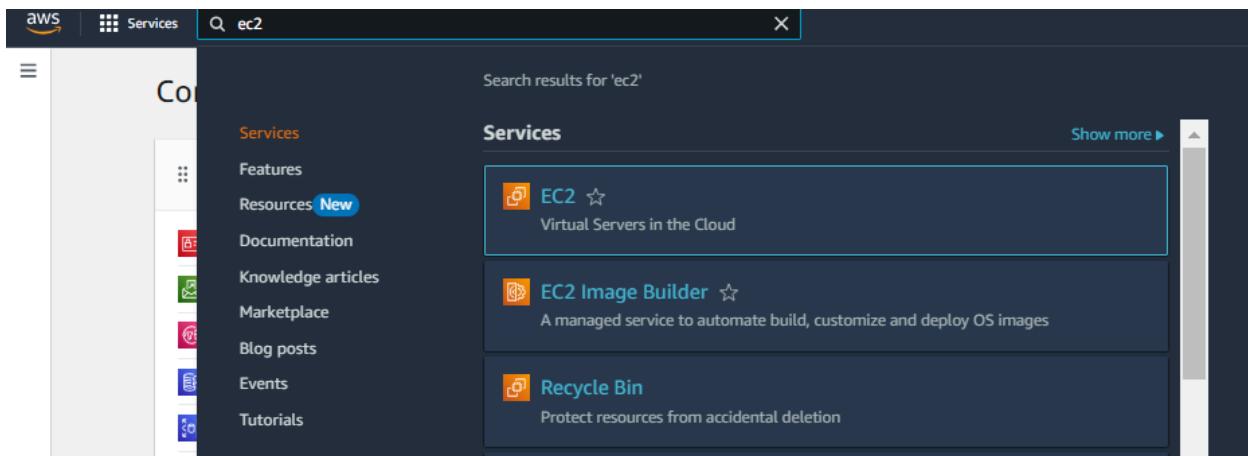
A modal dialog is open over the repository page, specifically for the "Clone" section. The dialog includes:

- Clone options: Local (selected), Codespaces.
- Protocol: HTTPS (selected), SSH, GitHub CLI.
- Clone URL: <https://github.com/Rohithbathini05/MEDTRACK.git>
- Actions: 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.



>Click on Launch instance to launch EC2 instance

A screenshot of the EC2 Dashboard. On the left, there is a sidebar with navigation links: EC2 Dashboard, EC2 Global View, Events, Instances (which is expanded to show Instances, Instance Types, Launch Templates, Spot Requests, and Savings Plans), and a search bar. The main content area is titled 'Instances Info' and shows a message: 'No instances' and 'You do not have any instances in this region'. A large orange 'Launch instances' button is prominently displayed. At the top of the main area, there are filters for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IPv4 DNS.

The screenshot shows the 'Launch an instance' wizard in the AWS Management Console. The left sidebar lists 'Name and tags', 'Application and OS Images (Amazon Machine Image)', and 'Quick Start'. The main area shows a configuration form with the following details:

- Name and tags**: Name is set to 'medtrack-server'.
- Software Image (AMI)**: Selected AMI is 'Amazon Linux 2 Kernel 5.10 AMI...' (ami-000ec6c25978d5999).
- Virtual server type (instance type)**: Selected instance type is 't2.micro'.
- Firewall (security group)**: New security group is selected.

At the bottom right are 'Cancel', 'Launch instance', and 'Preview code' buttons.

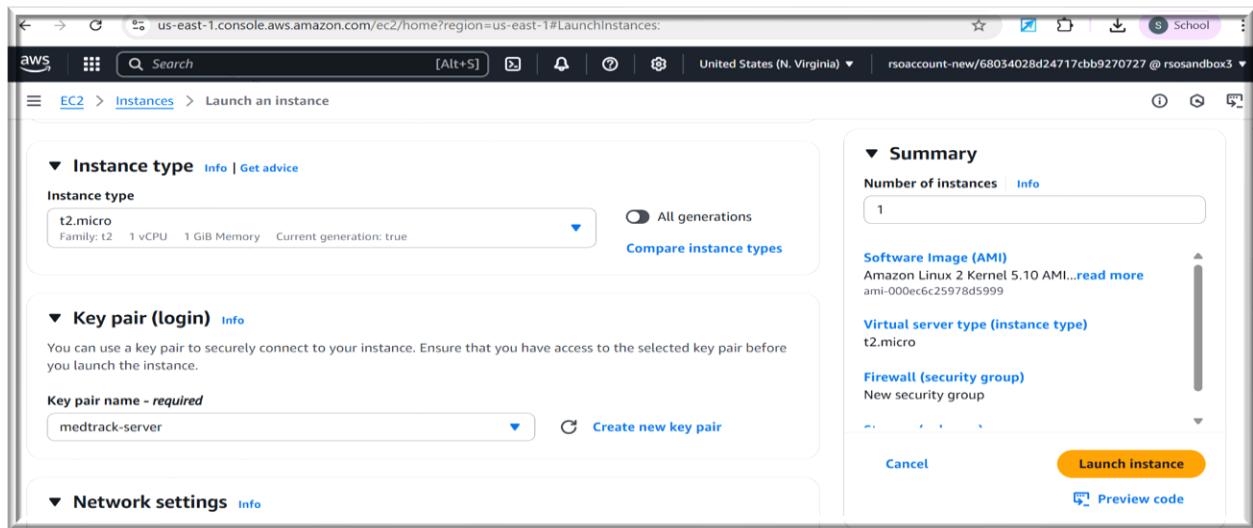
The screenshot shows the 'Amazon Machine Image (AMI)' search results page. The top navigation bar includes links for 'Amazon Linux', 'macOS', 'Ubuntu', 'Windows', 'Red Hat', and a search icon. A 'Browse more AMIs' button is also present.

The main content area displays the 'Amazon Linux 2023 AMI' entry, which includes the following details:

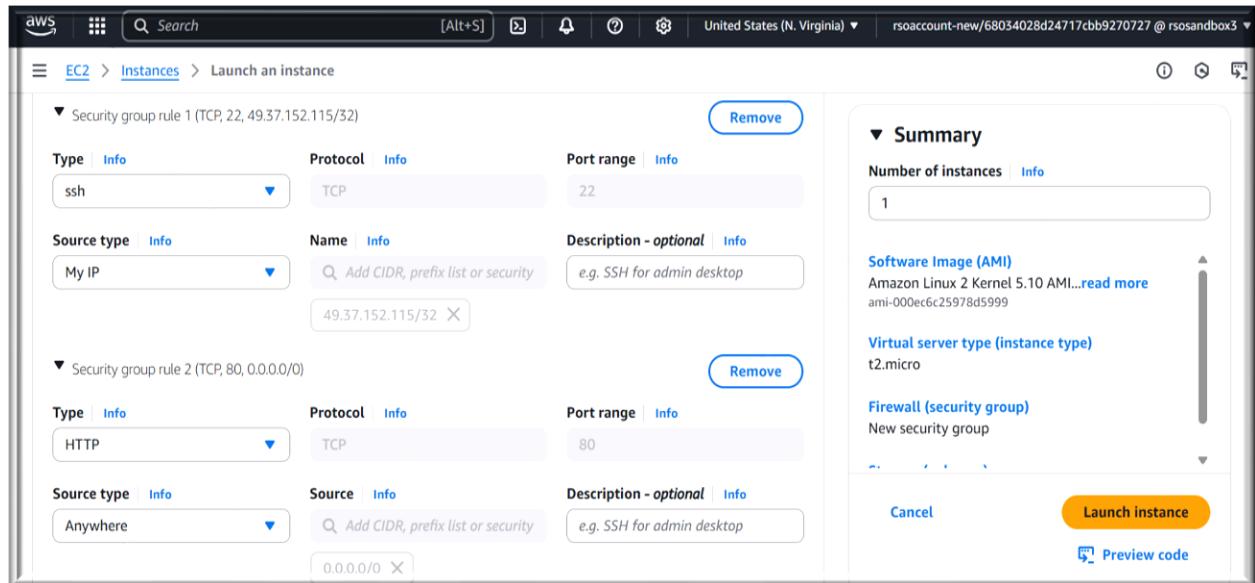
- AMI ID**: ami-02b49a24cfb95941c (64-bit (x86), uefi-preferred) / ami-04ad8c7fcc828fad4 (64-bit (Arm), uefi)
- Virtualization**: hvm
- ENAvanced**: true
- Root device type**: ebs
- Status**: Free tier eligible

The 'Description' section provides a brief overview of Amazon Linux 2023. Below it, the 'Architecture', 'Boot mode', and 'AMI ID' are listed as 64-bit (x86), uefi-preferred, and ami-02b49a24cfb95941c respectively. A 'Verified provider' badge is also present.

>Create and Download the keypair for server access:



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



The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with 'EC2' selected. Under 'Instances', there are links for 'Instances', 'Instance Types', 'Launch Templates', 'Spot Requests', 'Savings Plans', 'Reserved Instances', 'Dedicated Hosts', and 'Capacity Reservations'. Under 'Images', there are links for 'AMIs' and 'ALMs'. The main content area shows 'Instances (1/1) info' with a table. The table has columns for Name, Instance ID, Instance state, Instance type, Status check, and Alarm status. One row is selected for 'medtrack-server' with ID i-01e8a55830cb0f801, which is 'Running' on a 't2.micro' instance. Below the table, a detailed view for 'i-01e8a55830cb0f801 (medtrack-server)' is shown with tabs for Details, Status and alarms, Monitoring, Security, Networking, Storage, and Tags. The 'Details' tab is selected, showing 'Instance ID' i-01e8a55830cb0f801, 'Public IPv4 address' 98.80.11.234, and 'Private IPv4 addresses' 172.31.24.250.

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 screenshot shows the AWS EC2 Instance summary page for instance i-001861022fbcac290. The left pane displays various instance details: Instance ID (i-001861022fbcac290), IPv6 address (-), Hostname type (IP name: ip-172-31-3-5.ap-south-1.compute.internal), Answer private resource DNS name (IPv4 (A)), Auto-assigned IP address (-), IAM Role (sns_Dynamodb_role), and IMDSv2 (Required). The right pane shows the instance's configuration: Public IPv4 address (-), Instance state (Stopped), Private IPv4 addresses (172.31.3.5), Public IPv4 DNS (ip-172-31-3-5.ap-south-1.compute.internal), Instance type (t2.micro), VPC ID (vpc-03cdc7b6f19dd7211), Subnet ID (subnet-0d9fa3144480cc9a9), and Instance ARN (arn:aws:ec2:ap-south-1:557690616836:instance/i-001861022fbcac290). The 'Actions' menu is open, with 'Security' selected. Other options include 'Connect', 'Manage instance state', 'Instance settings', 'Networking', 'Get Windows password', 'Image and templates', 'Monitor and troubleshoot', and 'Auto Scaling Group name'.

The screenshot shows the AWS IAM Roles page. On the left, there's a sidebar with navigation links like 'Identity and Access Management (IAM)', 'Dashboard', 'Access management' (with 'Roles' selected), 'Policies', 'Identity providers', 'Account settings', and 'Root access management'. The main area displays a table titled 'Roles (14)'. The table has columns for 'Role name', 'Trusted entities', and 'Last activity'. The roles listed are: 'AWS ServiceRoleForAccessAnalyzer', 'AWS ServiceRoleForAPIGateway', 'AWS ServiceRoleForApplicationAutoScaling_DynamoDBTable', 'AWS ServiceRoleForAutoScaling', 'AWS ServiceRoleForCloudWatchEvents', 'AWS ServiceRoleForECS', and 'AWS ServiceRoleForElasticLoadBalancing'. Each role entry includes a link to its details and a timestamp indicating when it was last active.

Now connect the EC2 with the files

The screenshot shows the 'Connect to instance' page for an EC2 instance with ID i-001861022fbcac290. The instance is named 'InstantLibraryApp'. The 'EC2 Instance Connect' tab is selected. A warning message in a yellow box states: 'Port 22 (SSH) is open to all IPv4 addresses. Port 22 (SSH) is currently open to all IPv4 addresses, indicated by 0.0.0.0/0 in the inbound rule in your security group. For increased security, consider restricting access to only the EC2 Instance Connect service IP addresses for your Region: 13.233.177.0/29.' Below this, there are two connection options: 'Connect using EC2 Instance Connect' (selected) and 'Connect using EC2 Instance Connect Endpoint'. Under 'Connection Type', 'Public IPv4 address' is selected, showing the IP 13.200.229.59. There's also an option for 'IPv6 address'. The 'Username' field contains 'ec2-user'. A note at the bottom says: 'Note: In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.' At the bottom right are 'Cancel' and 'Connect' buttons.

```
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-172-31-24-250 ~] $
```

i-01e8a55830cb0f801 (medtrack-server)

Public IPs: 98.80.11.234 Private IPs: 172.31.24.250

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/Rohithbathini05/MEDTRACK.git>

This downloaded my project to the EC2 instance.

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

```
cd InstantLibrary
```

```
[root@ip-172-31-24-250 ec2-user]# pip install python-dotenv
DEPRECATION: Python 2.7 reached the end of its life on January 1st, 2020. Please upgrade your Python as Python 2.7 is no longer maintained.
pip 21.0 will drop support for Python 2.7 in January 2021. More details about Python 2 support in pip can be found at https://pip.pypa.io/en/latest/development/release-process/#python-2-support
WARNING: Running pip as root is generally not a good idea. Try 'pip install --user' instead.
Requirement already satisfied: python-dotenv in /usr/lib/python2.7/site-packages (from python-dotenv) (3.10.0.0)
Requirement already satisfied: typing; python_version < "3.5" in /usr/lib/python2.7/site-packages (from python-dotenv) (3.10.0.0)
[root@ip-172-31-24-250 ec2-user]# git clone https://github.com/ummesalma28/MEDTRACK.git
  Cloning into 'MEDTRACK'...
remote: Enumerating objects: 24, done.
remote: Counting objects: 100% (24/24), done.
remote: Compressing objects: 100% (23/23), done.
remote: Writing objects: 100% (24/24), reused (delta 0), pack-reused 0 (from 0)
   Receiving objects: 100% (24/24), 19.80 KiB | 6.60 MiB/s, done.
   Resolving deltas: 100% (5/5), done.
[root@ip-172-31-24-250 ec2-user]# ls
MEDTRACK req.txt
[root@ip-172-31-24-250 ec2-user]# cd MEDTRACK
[root@ip-172-31-24-250 MEDTRACK]# ls
app.py static templates
[root@ip-172-31-24-250 MEDTRACK]#
```

i-01e8a55830cb0f801 (medtrack-server)
PublicIPs: 98.80.11.234 PrivateIPs: 172.31.24.250

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Verify the Flask app is running

```
[root@ip-172-31-24-250 MEDTRACK]# pip3 install boto3
WARNING: Running pip install with root privileges is generally not a good idea. Try 'pip3 install --user' instead.
Collecting boto3
  Downloading boto3-1.33.13-py3-none-any.whl (139 kB)
Collecting jmespath<2.0.0,>=0.7.1
  Downloading jmespath-1.0.1-py3-none-any.whl (20 kB)
Collecting botocore<1.34.0,>=1.33.13
  Downloading botocore-1.33.13-py3-none-any.whl (11.8 MB)
Collecting s3transfer<0.9.0,>=0.8.2
  Downloading s3transfer-0.8.2-py3-none-any.whl (82 kB)
Collecting urllib3<1.27,>=1.25.4; python_version < "3.10"
  Downloading urllib3-1.26.20-py2.py3-none-any.whl (144 kB)
Collecting python-dateutil<3.0.0,>=2.1
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
Collecting six<1.5
  Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
```

i-01e8a55830cb0f801 (medtrack-server)
PublicIPs: 98.80.11.234 PrivateIPs: 172.31.24.250

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

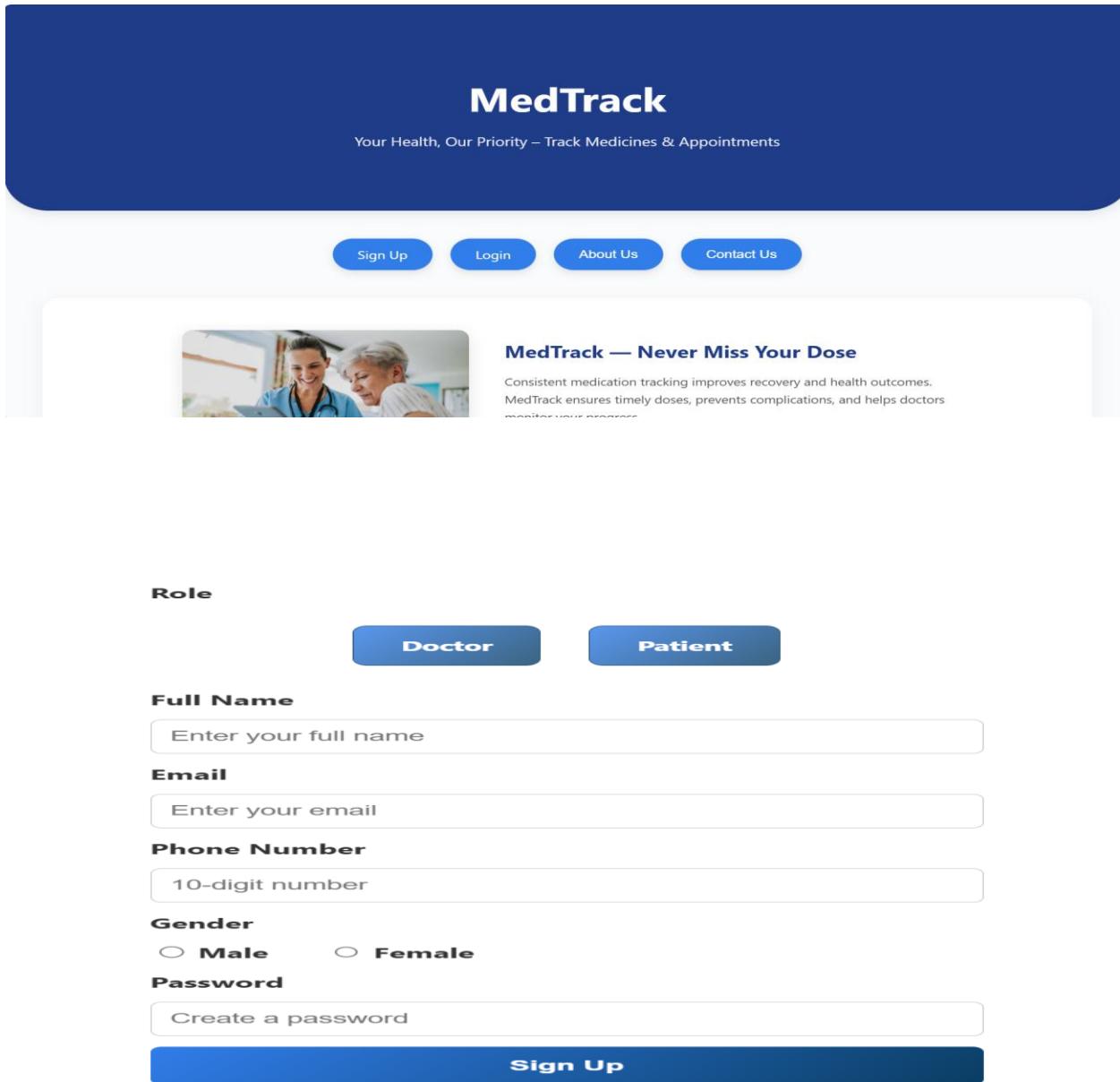
```
app.py MEDTRACK static templates
[root@ip-172-31-24-250 MEDTRACK]# python3 app.py
/usr/local/lib/python3.7/site-packages/boto3/compat.py:82: PythonDeprecationWarning: Boto3 will no longer support Python 3.7 starting December 13, 2023. To continue receiving service updates, bug fixes, and security updates please upgrade to Python 3.8 or later. More information can be found here: https://aws.amazon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-tools/
  warnings.warn(warning, PythonDeprecationWarning)
INFO:botocore.credentials:Found credentials in environment variables.
* Serving Flask app 'app'
* Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
/usr/local/lib/python3.7/site-packages/boto3/compat.py:82: PythonDeprecationWarning: Boto3 will no longer support Python 3.7 starting December 13, 2023. To continue receiving service updates, bug fixes, and security updates please upgrade to Python 3.8 or later. More information can be found here: https://aws.amazon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-tools/
  warnings.warn(warning, PythonDeprecationWarning)
INFO:botocore.credentials:Found credentials in environment variables.
WARNING:werkzeug: * Debugger is active!
INFO:werkzeug: * Debugger PIN: 507-351-819
https://github.com/ummesalma28/MEDTRACK.git
```

i-01e8a55830cb0f801 (medtrack-server)

Access the website through: <http://34.228.73.60:5000>

Milestone 8: Testing and Deployment

- Activity 8.1: Conduct functional testing to verify user index, login,signup,doctor dashboard, patient dashboard,book appointment , prescription,about us, contact us pages.



The screenshot shows the MedTrack homepage. At the top, there is a dark blue header with the MedTrack logo and tagline "Your Health, Our Priority – Track Medicines & Appointments". Below the header are four buttons: "Sign Up", "Login", "About Us", and "Contact Us". The main content area features a photograph of a doctor and a patient. To the right of the photo is a section titled "MedTrack — Never Miss Your Dose" with a subtext about consistent medication tracking improving recovery and health outcomes. Below this is a "Role" selection section with "Doctor" and "Patient" buttons. Further down are fields for "Full Name", "Email", and "Phone Number", each with a placeholder text box. There is also a "Gender" section with "Male" and "Female" radio buttons, and a "Password" field with a placeholder. A large blue "Sign Up" button is at the bottom.

MedTrack
Your Health, Our Priority – Track Medicines & Appointments

Sign Up Login About Us Contact Us



MedTrack — Never Miss Your Dose
Consistent medication tracking improves recovery and health outcomes.
MedTrack ensures timely doses, prevents complications, and helps doctors monitor your progress.

Role

Doctor **Patient**

Full Name
Enter your full name

Email
Enter your email

Phone Number
10-digit number

Gender
 Male Female

Password
Create a password

Sign Up

Login

Role

[Doctor](#) [Patient](#)

Email

Password

[Login](#)

[Create Account?](#)



MedTrack — Never Miss Your Dose

Consistent medication tracking improves recovery and health outcomes. MedTrack ensures timely doses, prevents complications, and helps doctors monitor your progress.

Whether for chronic care or daily wellness, MedTrack gives you peace of mind and control over your treatment.

About Us

MedTrack is a simple, reliable healthcare assistant that connects doctors and patients for smooth prescription management, medicine tracking, and appointment follow-ups. Our goal: smarter health, everyday.

[Dashboard](#) [Profile](#) [Logout](#)

Welcome, Shaik Umme Salma
Track your Appointments and Prescriptions with MedTrack

Patient Dashboard

[Book Appointment](#)

0
Upcoming Appointments

0
Completed Consultations

0
Prescriptions Available

Upcoming Appointments

No upcoming appointments.

Patient Profile Update

Full Name:

Shaik Umme Salma

Email:

salma@gmail.com

Phone:

6303918144

Gender:

Female

Password:

[Dashboard](#) [Profile](#) [Logout](#)

Welcome, Dr. Shaik Umme Salma

Manage your appointments with MedTrack

Doctor Dashboard

0

Pending Appointments

0

Completed Appointments

0

Total Appointments

Patients

[View Details](#)

Write Prescriptions

[Click to prescribe](#)

Write Prescription

Patient Name

Prescription

[Done](#)