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.

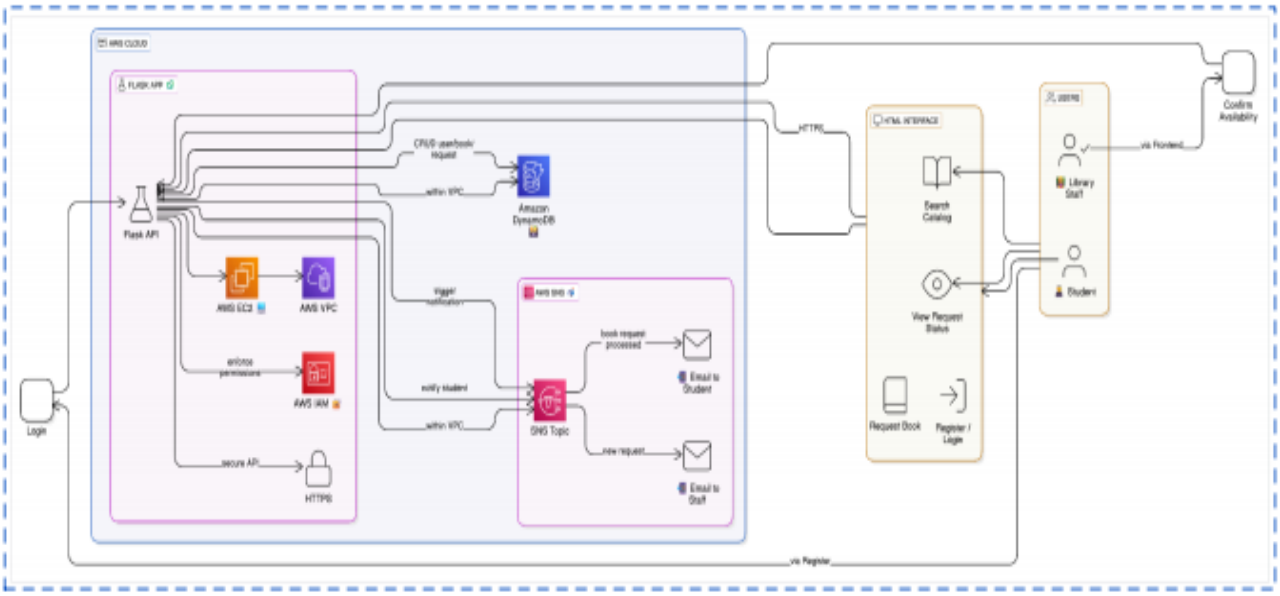
Scenario 2: Secure User Management with IAM

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.

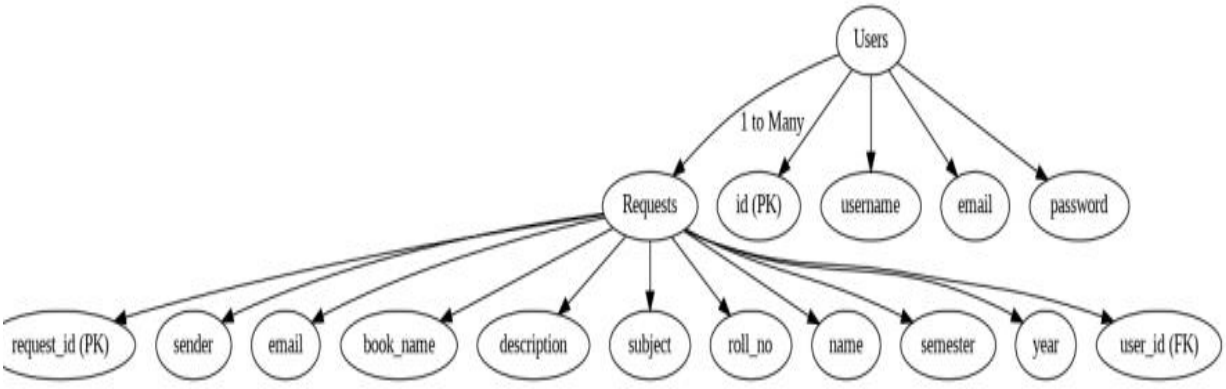
Scenario 3: Easy Access to Medical History and Resources

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:IntegrateAWSServicesUsingboto3.

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

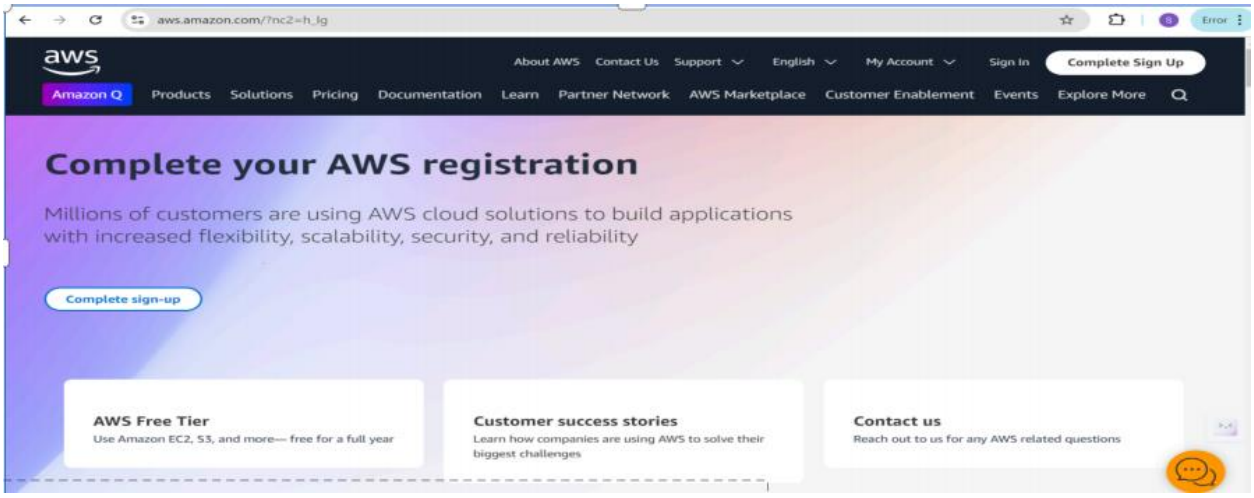
8. Testing and Deployment

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

Milestone 1: AWS Account Setup and Login

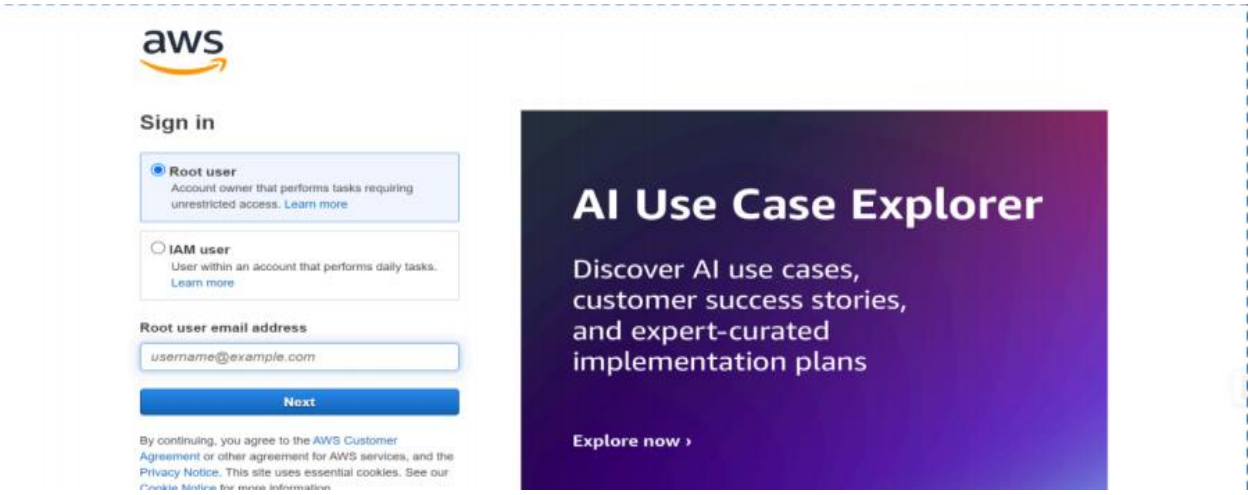
- **Activity 1.1: Set up an AWS account if not already done.**

○ Sign up for an AWS account and configure billing settings



Activity 1.2: Log in to the AWS Management Console

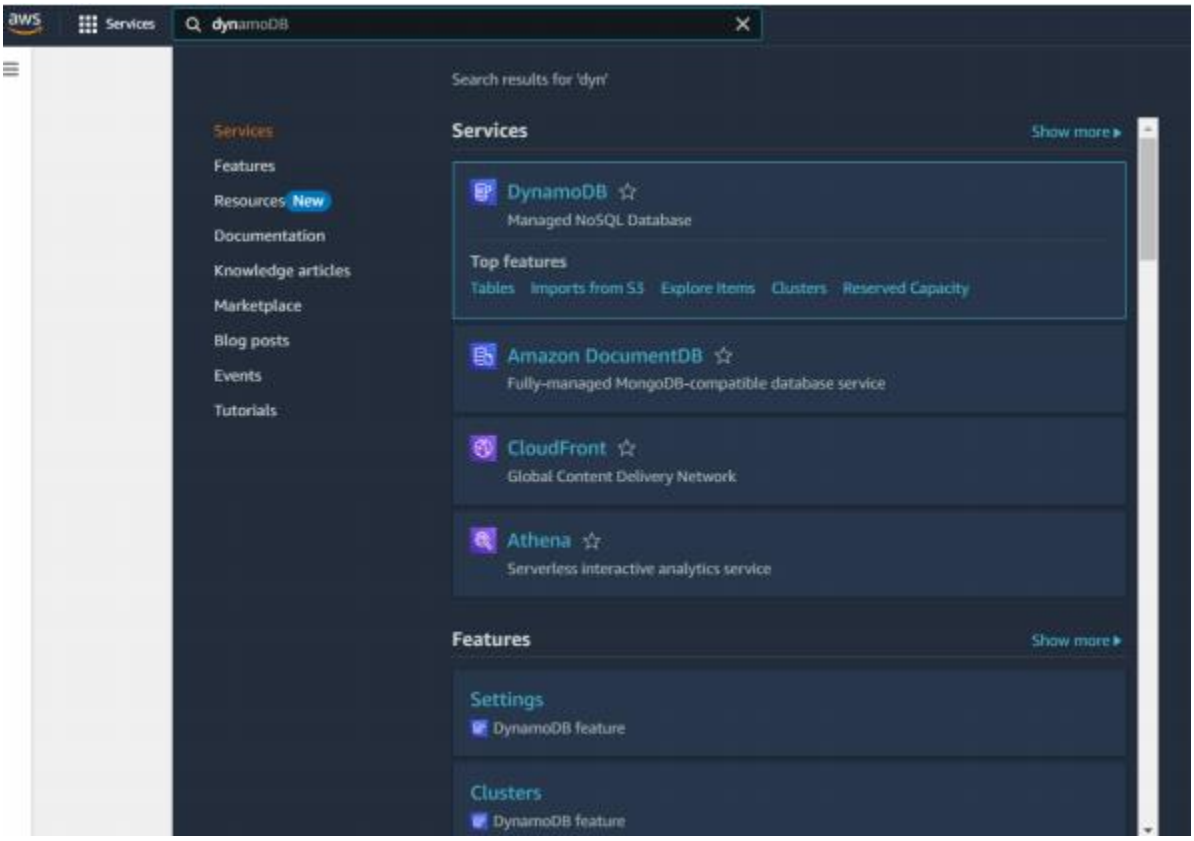
○ After setting up your account, log in to the AWS Management Console

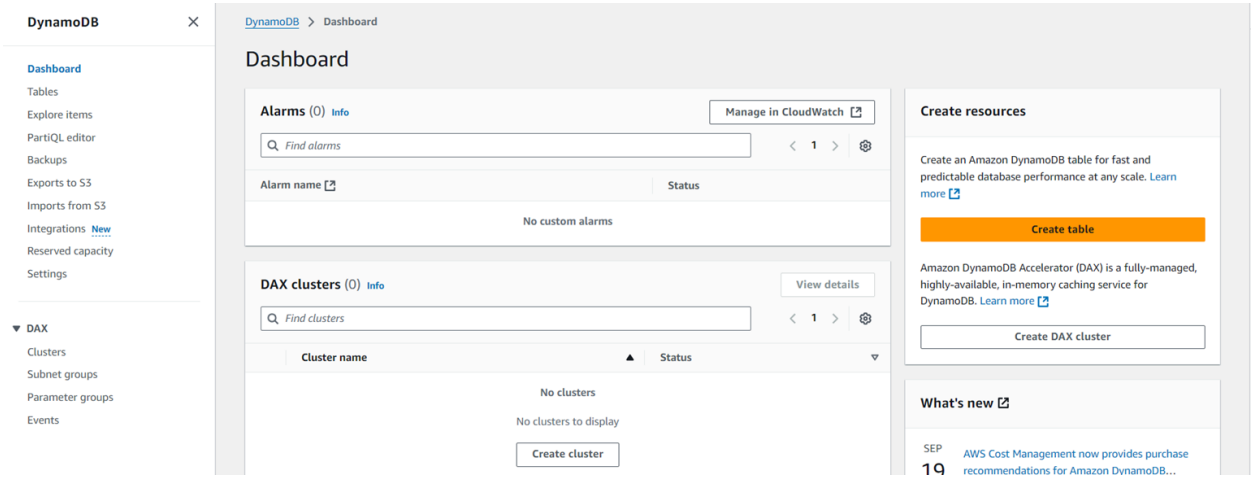


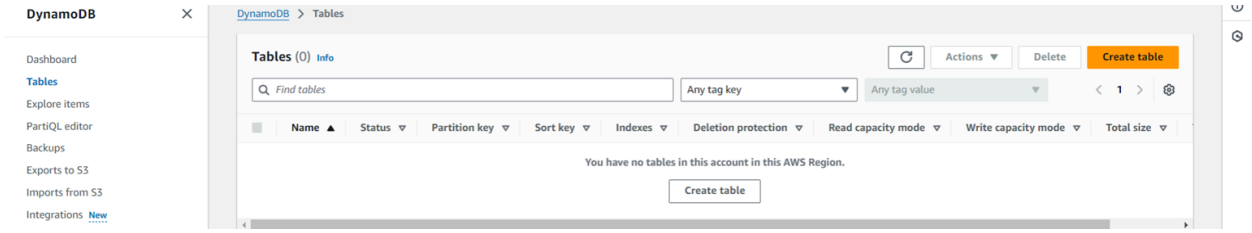
Milestone 2: DynamoDB Database Creation and Setup

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

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

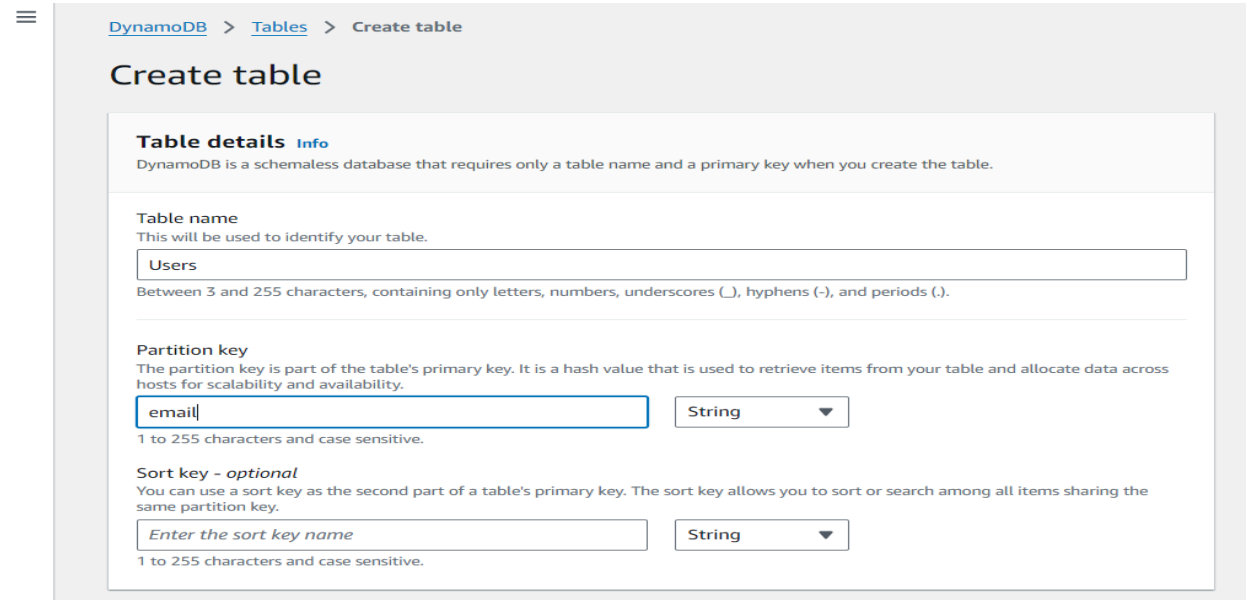


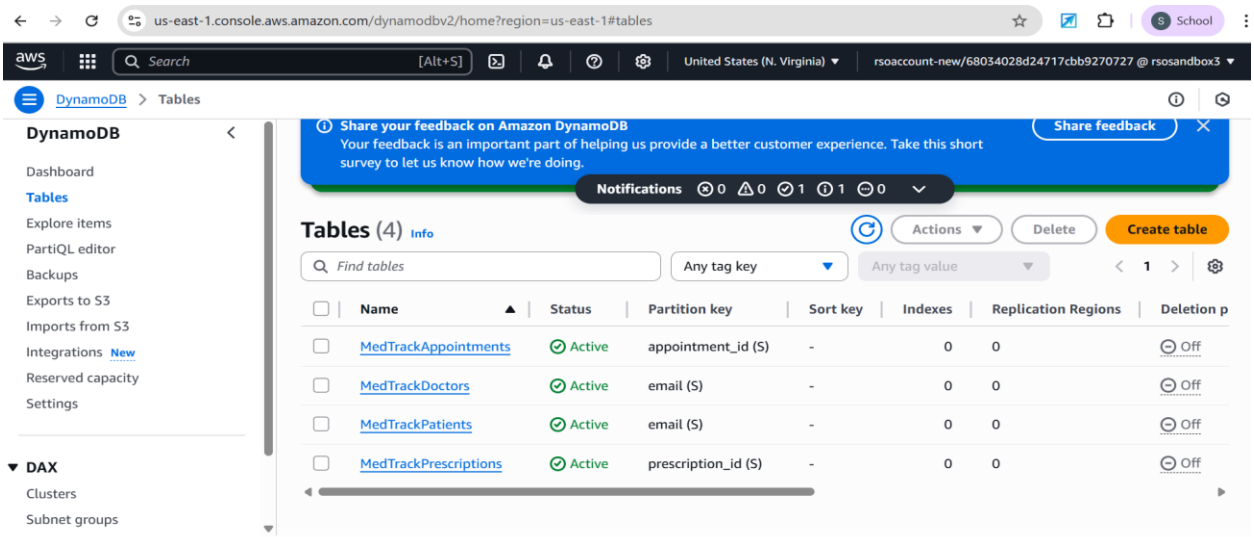




● **Activity 2.2:Create a DynamoDB table for storing registration details and book requests.**

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

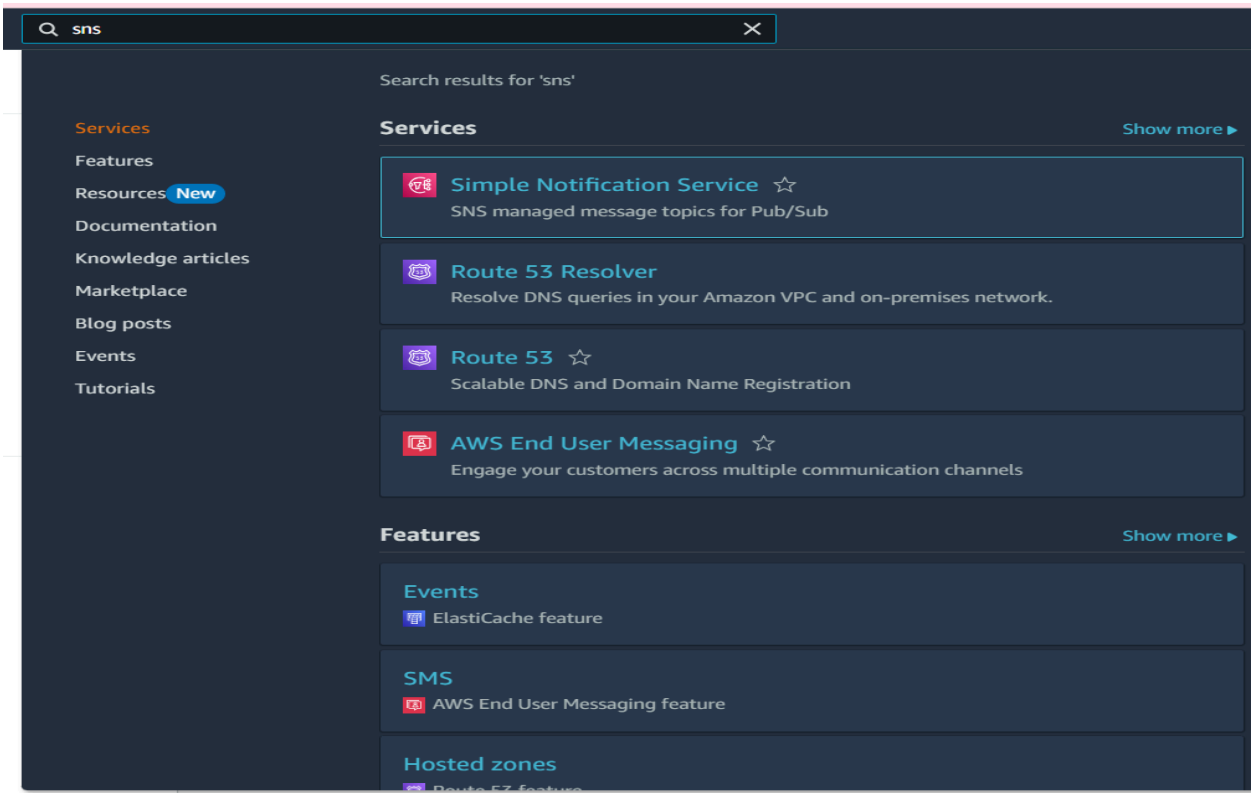


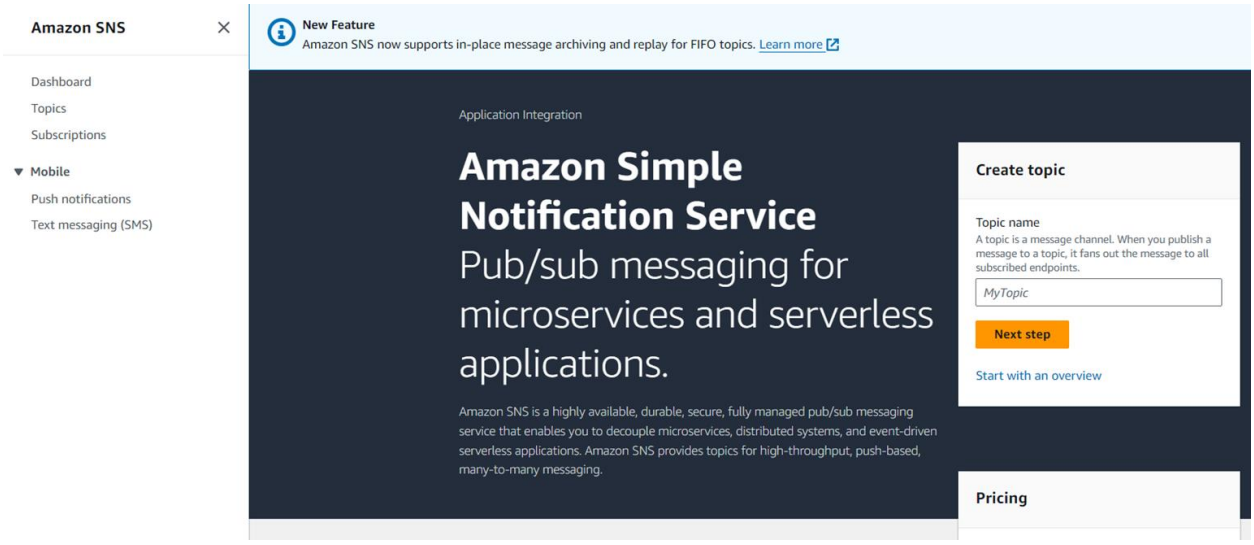


Milestone 3: SNS Notification Setup

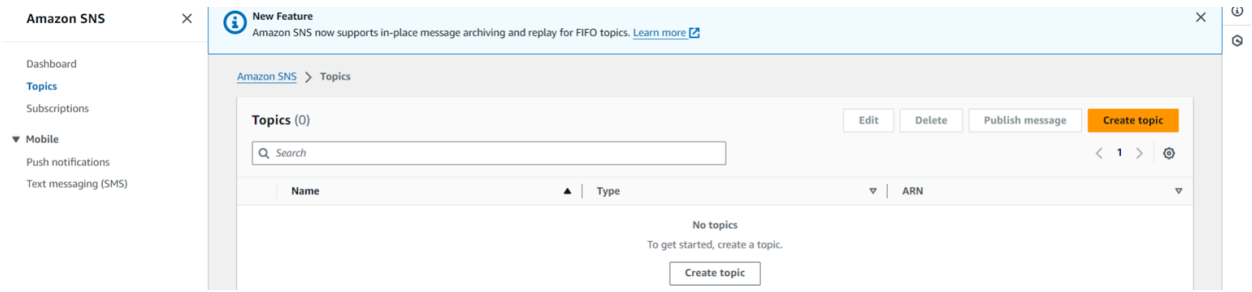
Activity 3.1

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.

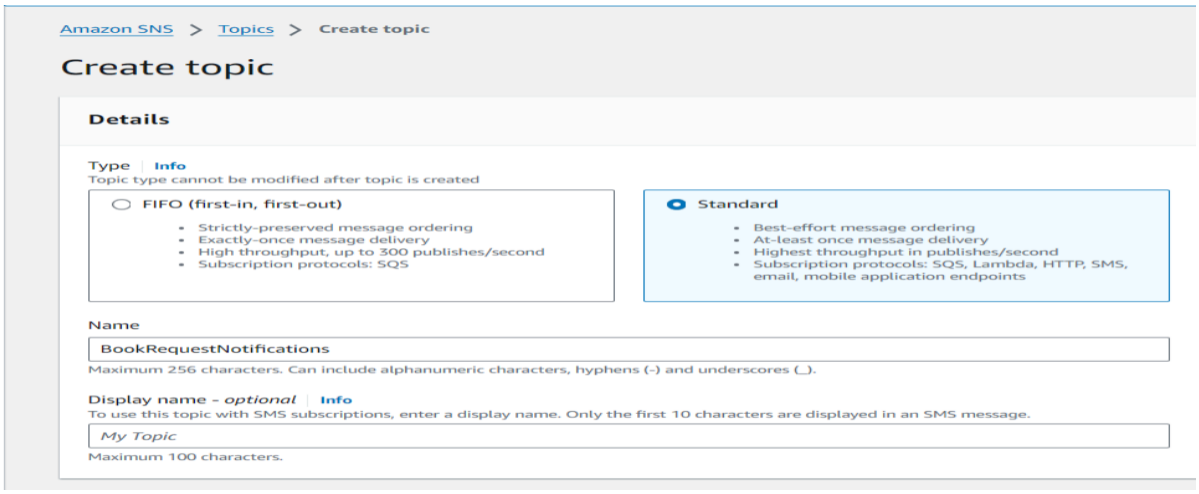


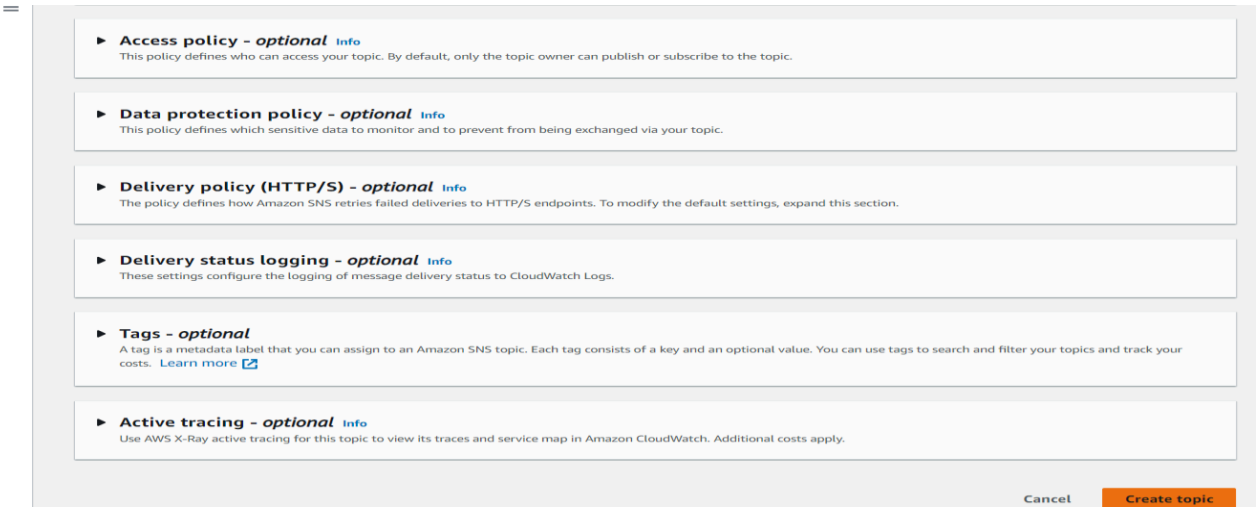


> Click on **Create Topic** and choose a name for the topic

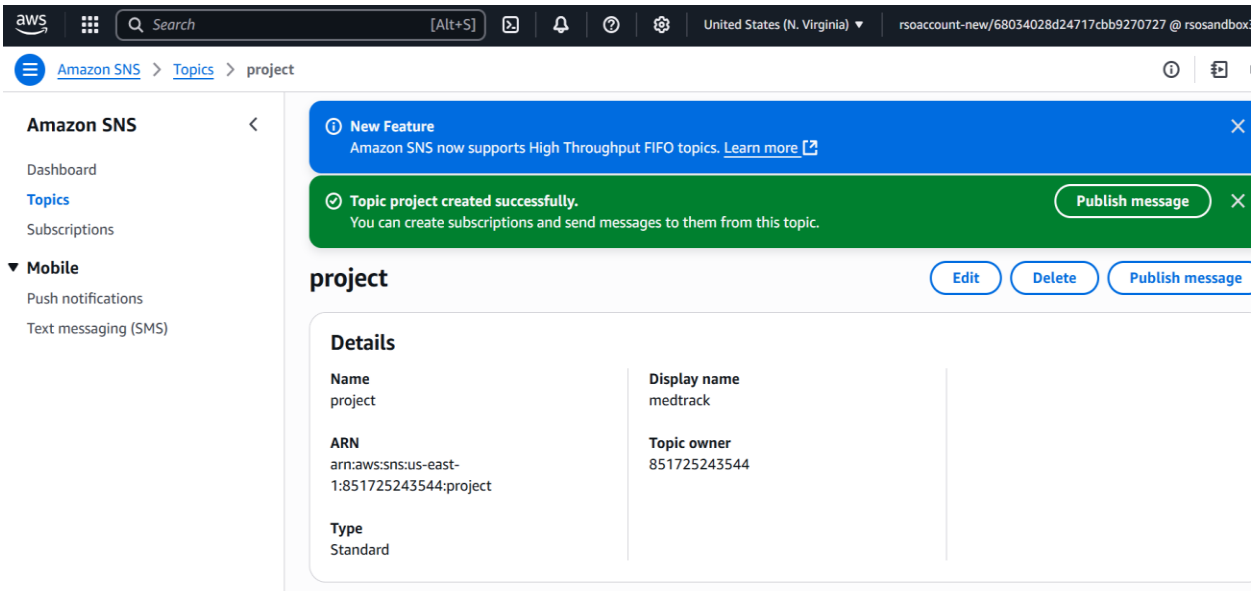


> Choose Standard type for general notification use cases and Click on Create



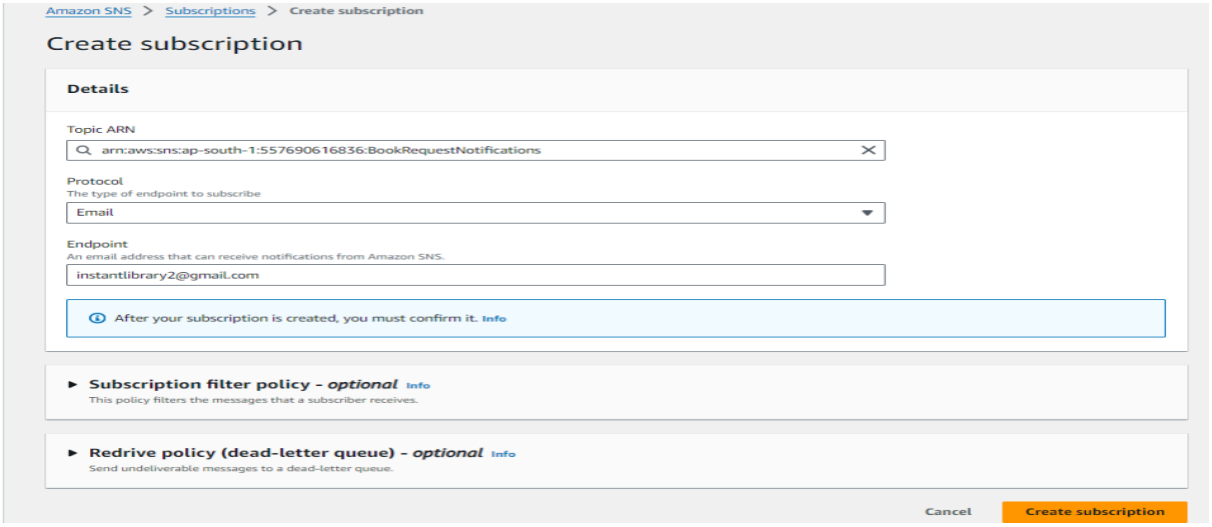


>Configure the SNS topic and note down the **Topic ARN**

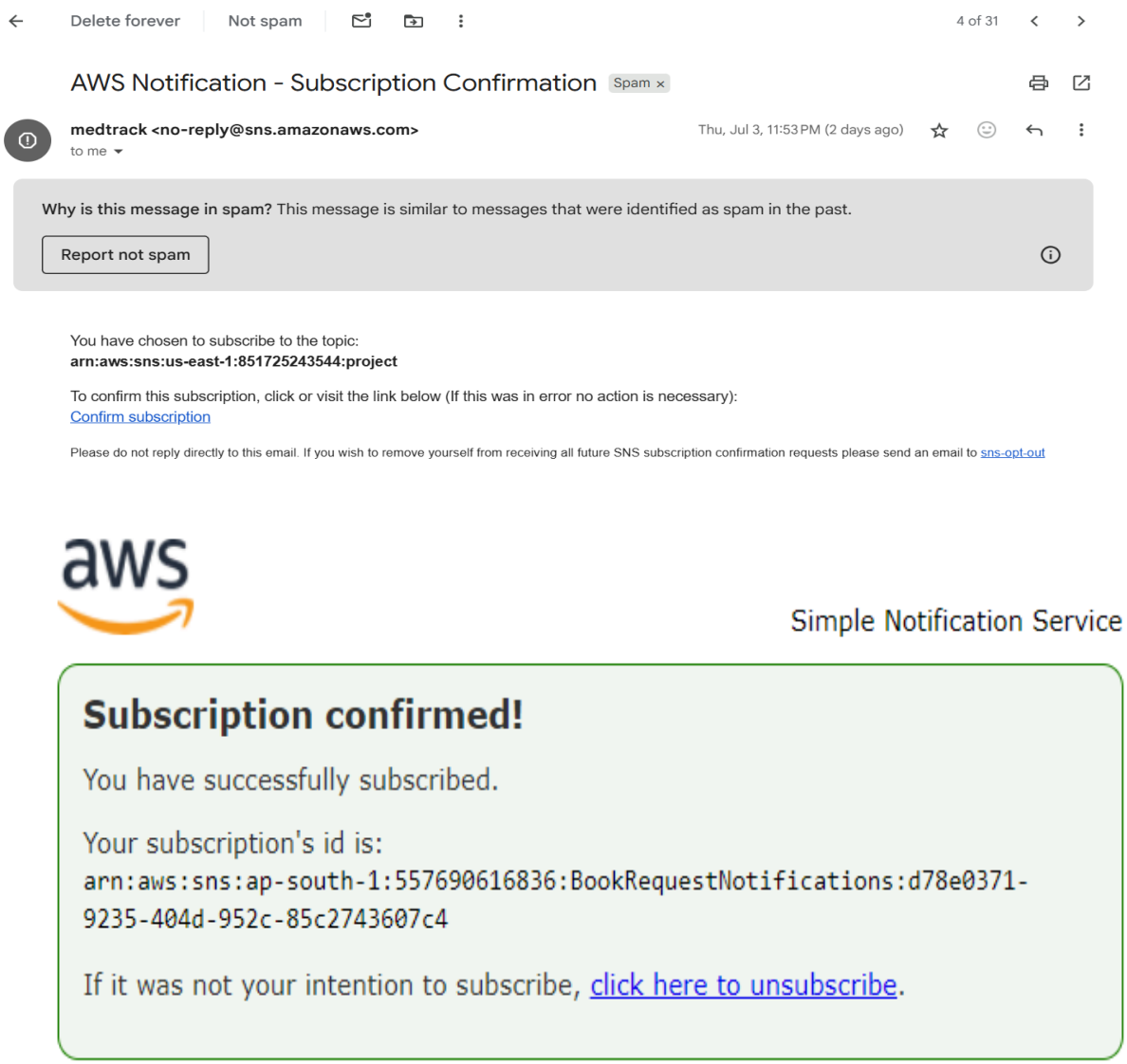


Activity 3.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.

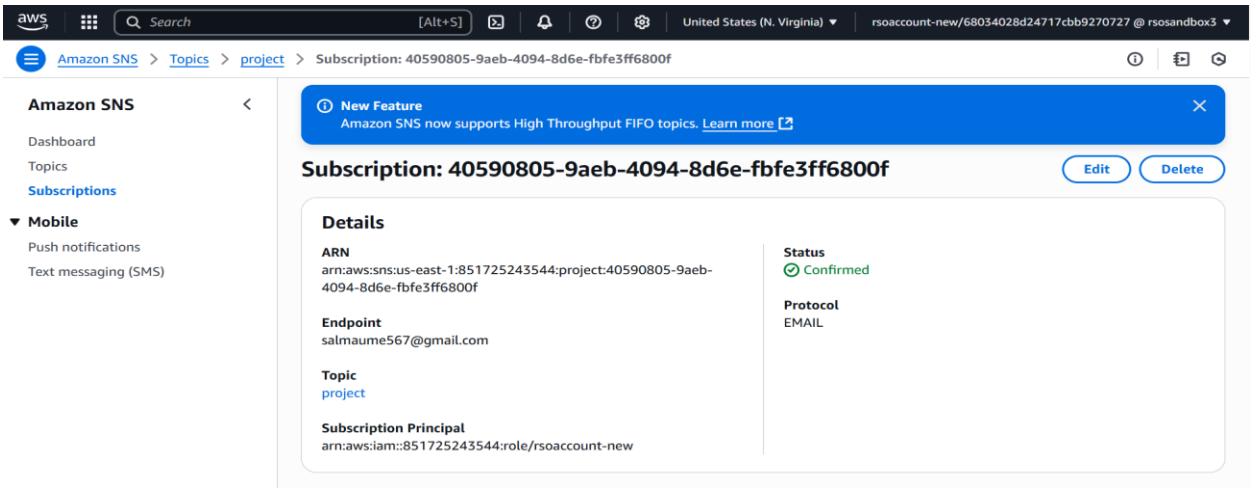
○ Subscribe users (or admin staff) to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails



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

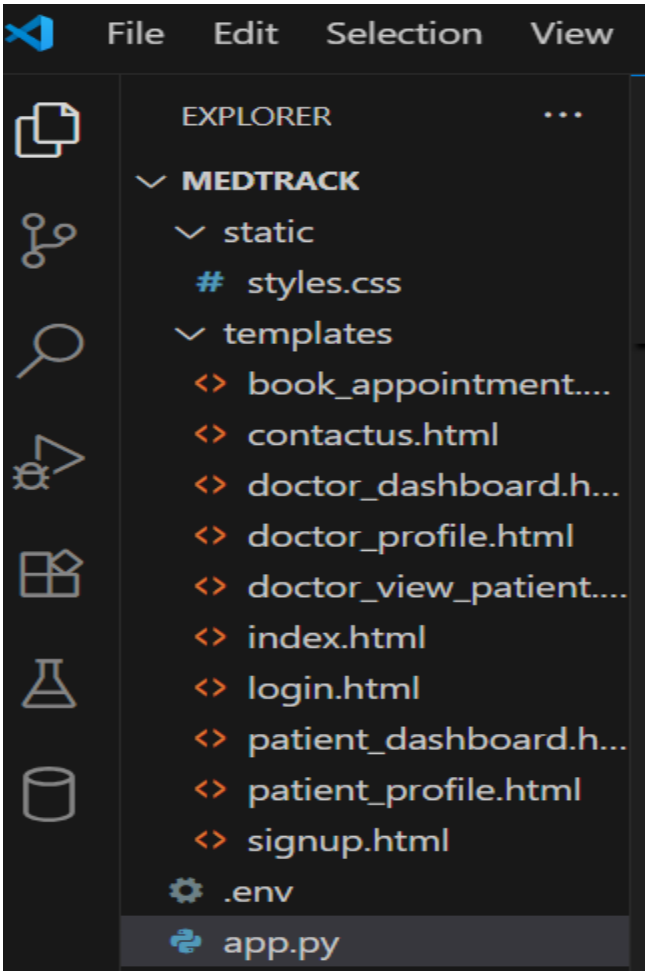


>Successfully done with the SNS mail subscription and setup, now store the ARN link



Milestone 4:Backend Development and Application Setup

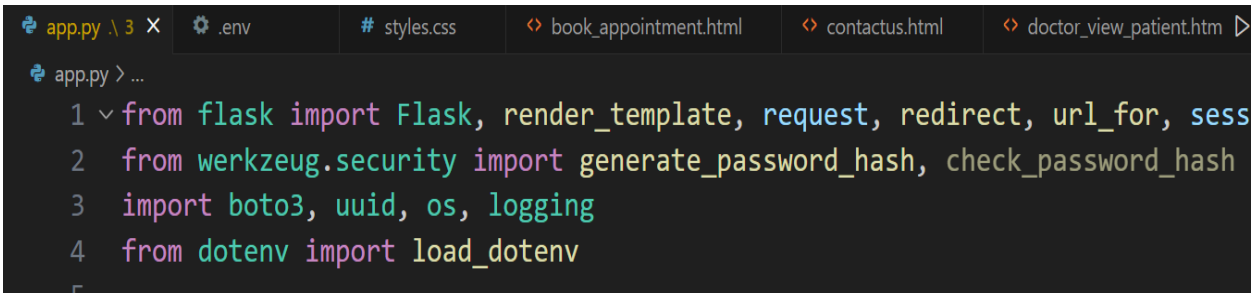
- **Activity 4.1: Develop the backend using Flask**
  - File Explorer Structure



Description: set up the INSTANT LIBRARY project with an app.py 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



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

```python
# 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

```python
# ---------------- 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):

```python
@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"], "rol
        return redirect(url_for("doctor_dashboard" if role == "doctor" els
```

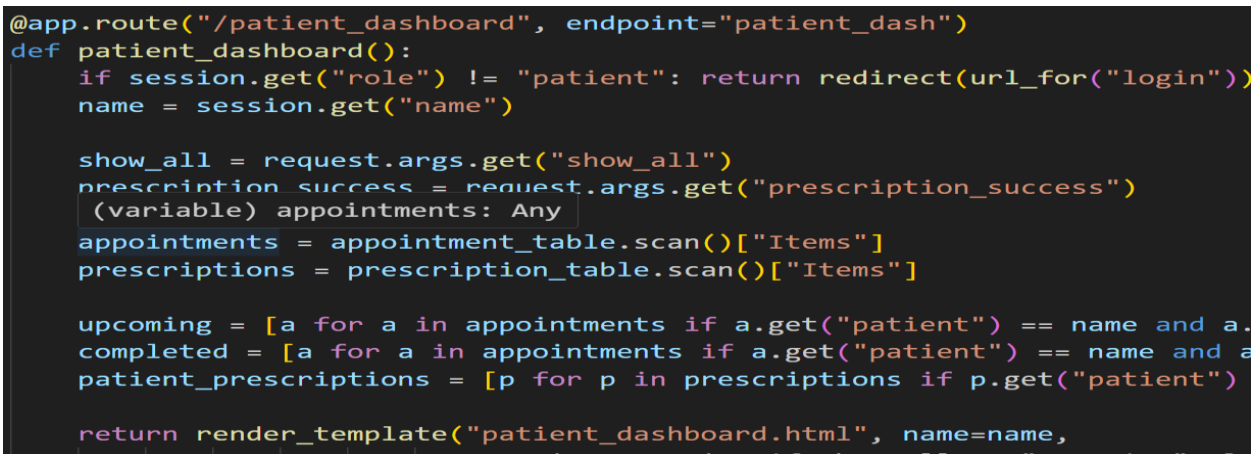
Doctor Dashboard route:

```python
@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

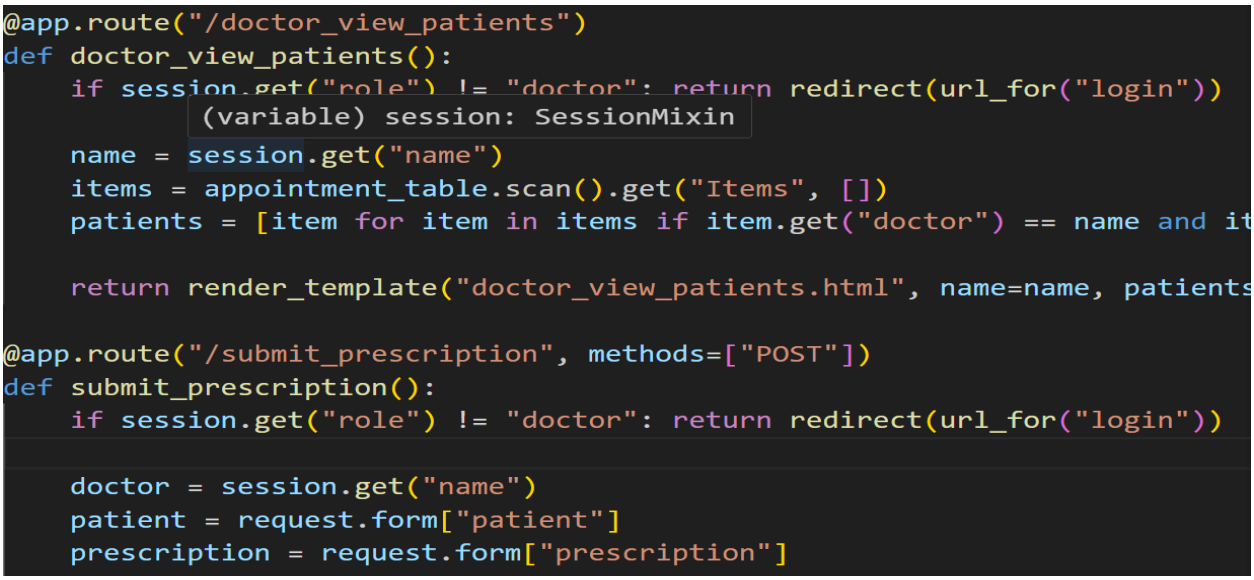
them as upcoming or completed.The route passes this data to doctor_dashboard.html, which shows a summary of appointments and patient interactions.

Patient dashboard Route



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:



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 = :g
            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")
```

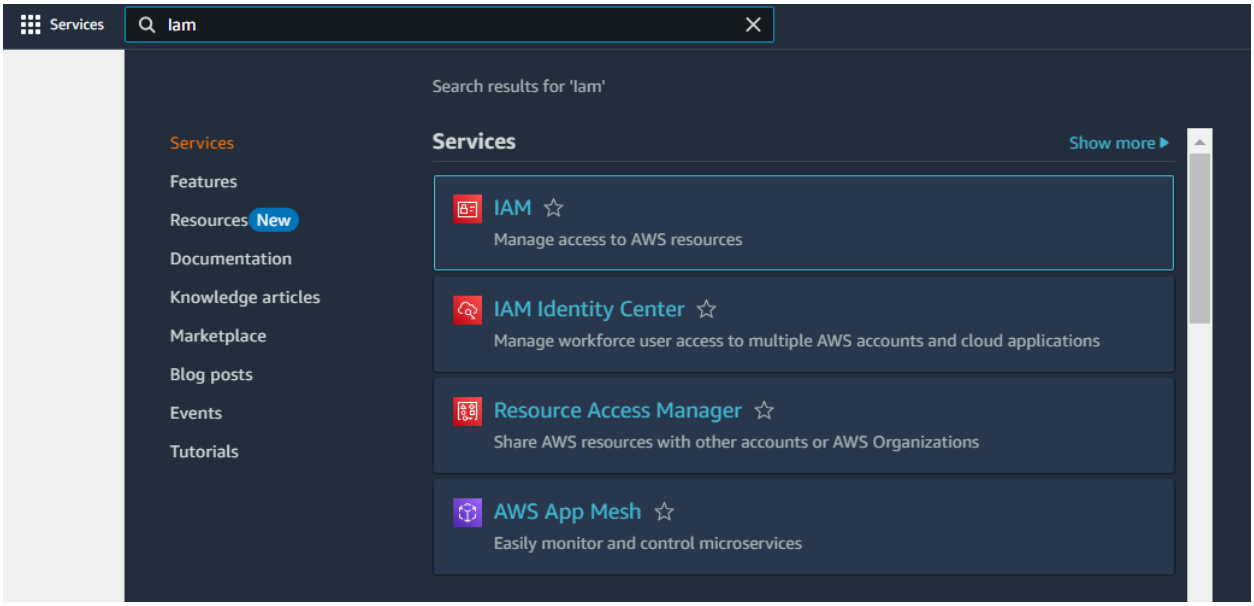
Deployement code

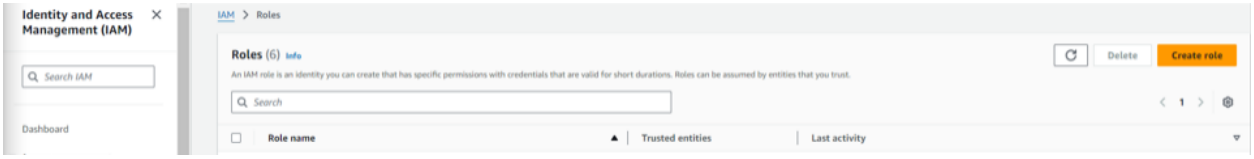
```
if __name__ == "__main__":
    app.run(debug=True)
```

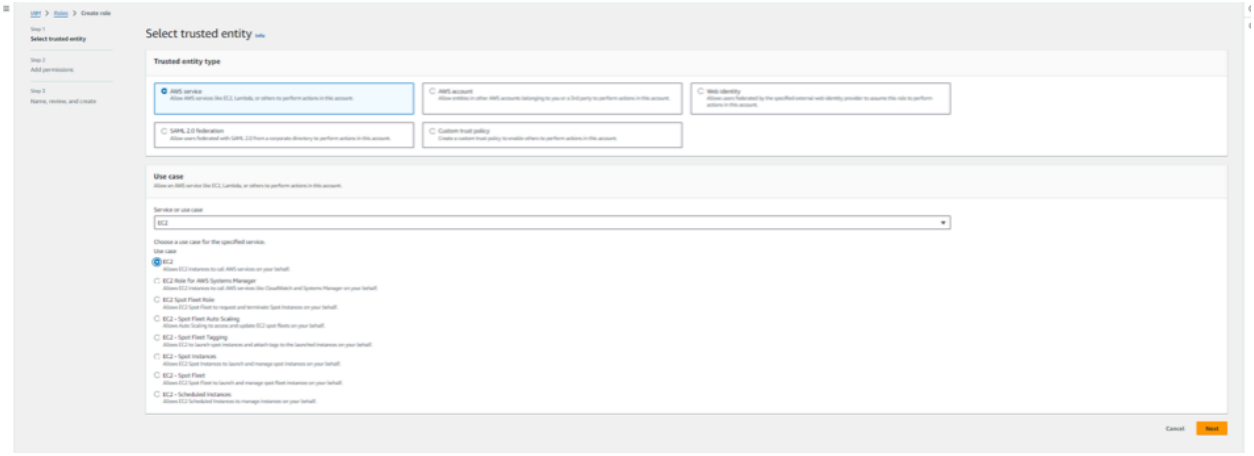
Milestone 5: IAM Role Setup

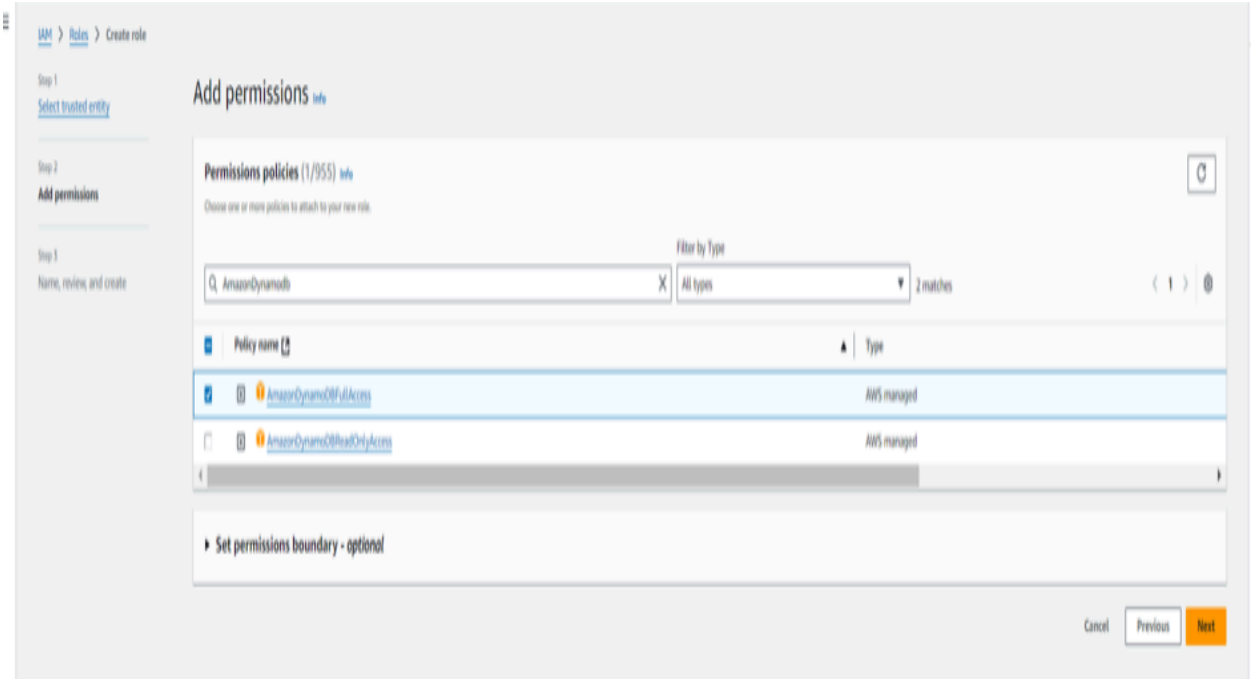
● Activity 5.1:Create IAM Role.

○ In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.


Services
Iam
Search results for 'Iam'
Services
Features
Resources New
Documentation
Knowledge articles
Marketplace
Blog posts
Events
Tutorials
Services
Show more
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


Identity and Access Management (IAM)
Search IAM
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.
Search
Delete
Create role
Role name
Trusted entities
Last activity


Select trusted entity
Trusted entity type
Use case

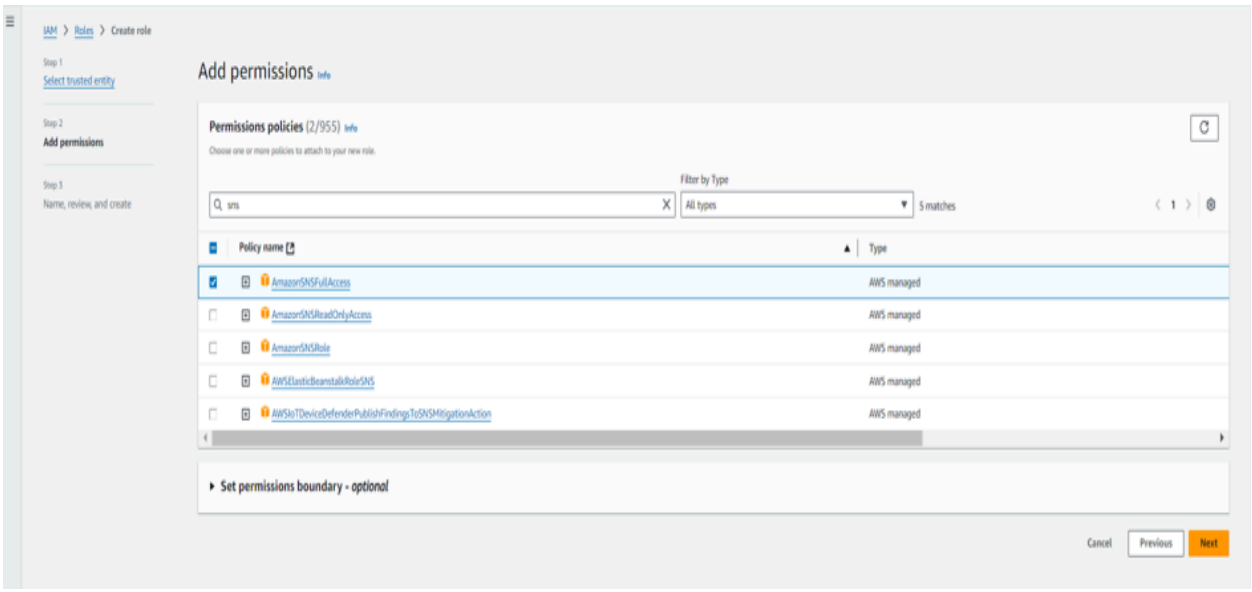


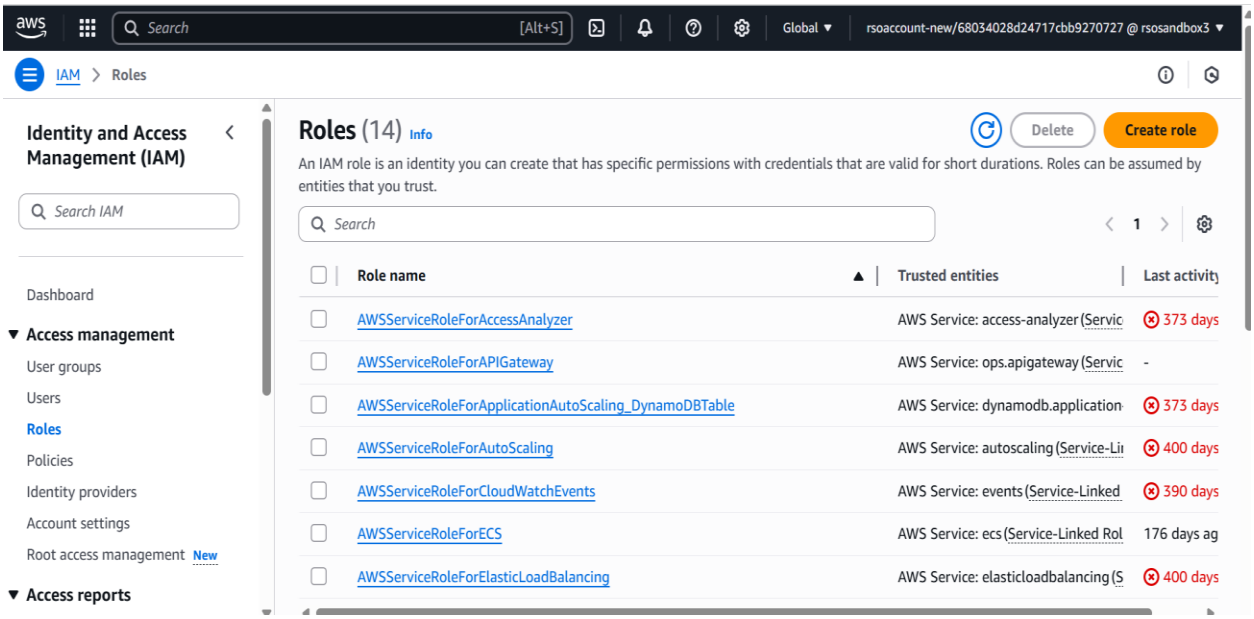
Activity 5.2: Attach Policies.

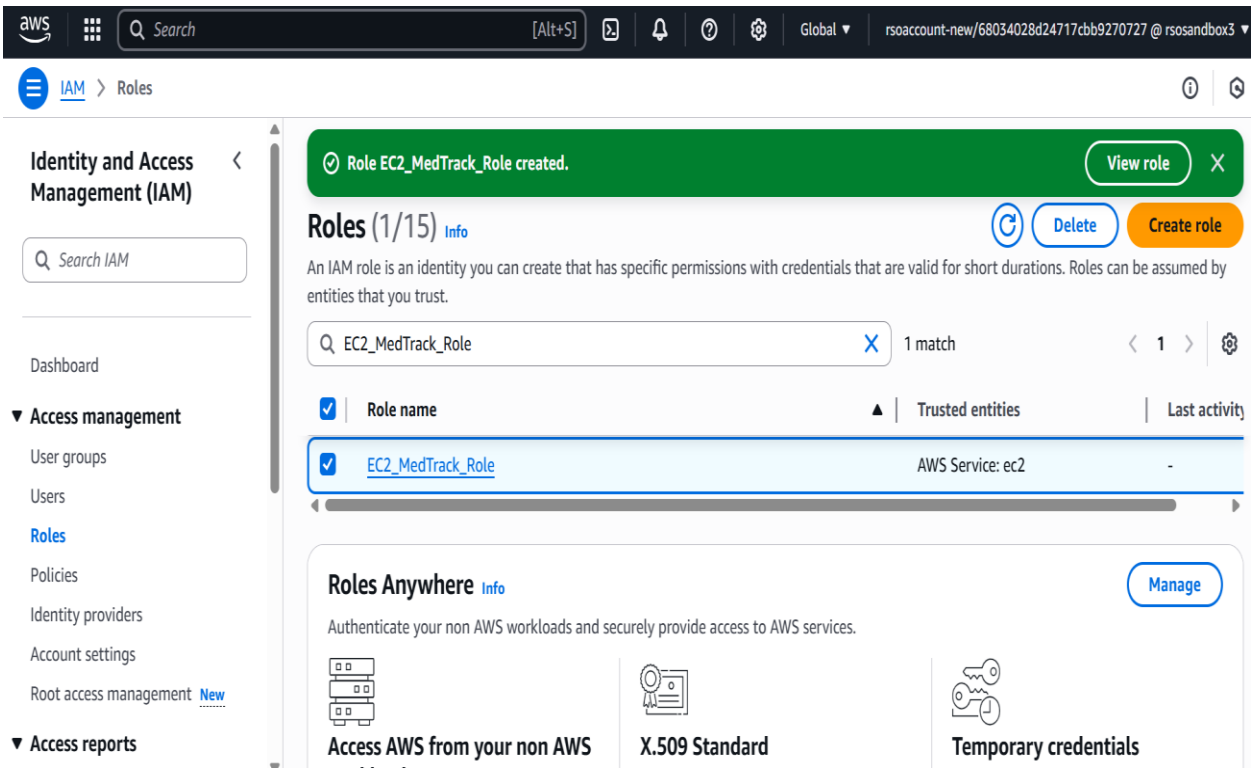
Attach the following policies to the role:

● AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.

● AmazonSNSFullAccess: Grants EC2 the ability to send notifications via SNS.

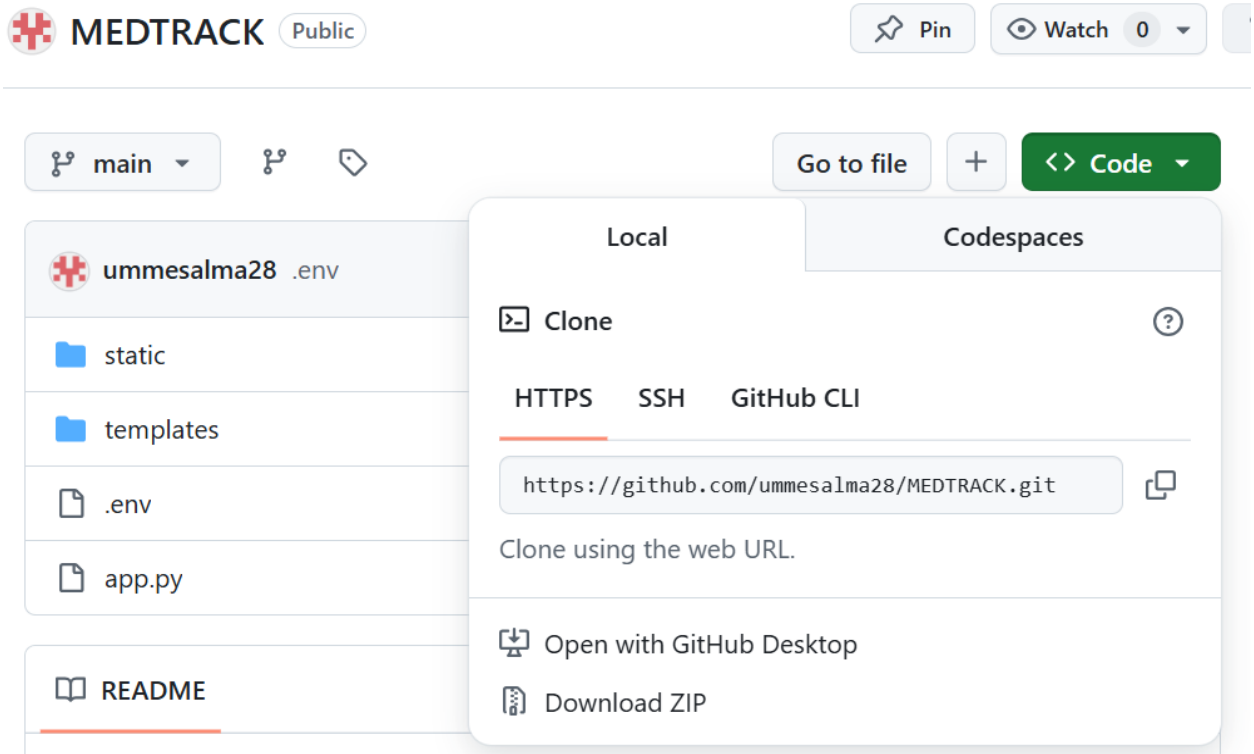



Search [Alt+S]
Global
rsoaccount-new/68034028d24717cbb9270727 @ rsosandbox3
IAM > Roles
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management New
Access reports
Roles (14) Info
An IAM role 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.
Delete
Create role
Search
Role name
Trusted entities
Last activity
AWSServiceRoleForAccessAnalyzer
AWS Service: access-analyzer (Servic
373 days
AWSServiceRoleForAPIGateway
AWS Service: ops.apigateway (Servic
-
AWSServiceRoleForApplicationAutoScaling_DynamoDBTable
AWS Service: dynamodb.application
373 days
AWSServiceRoleForAutoScaling
AWS Service: autoscaling (Service-Li
400 days
AWSServiceRoleForCloudWatchEvents
AWS Service: events (Service-Linked
390 days
AWSServiceRoleForECS
AWS Service: ecs (Service-Linked Rol
176 days ag
AWSServiceRoleForElasticLoadBalancing
AWS Service: elasticloadbalancing (S
400 days


Search [Alt+S]
Global
rsoaccount-new/68034028d24717cbb9270727 @ rsosandbox3
IAM > Roles
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management New
Access reports
Role EC2_MedTrack_Role created.
View role
Roles (1/15) Info
Delete
Create role
An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.
EC2_MedTrack_Role
1 match
Role name
Trusted entities
Last activity
EC2_MedTrack_Role
AWS Service: ec2
-
Roles Anywhere Info
Manage
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

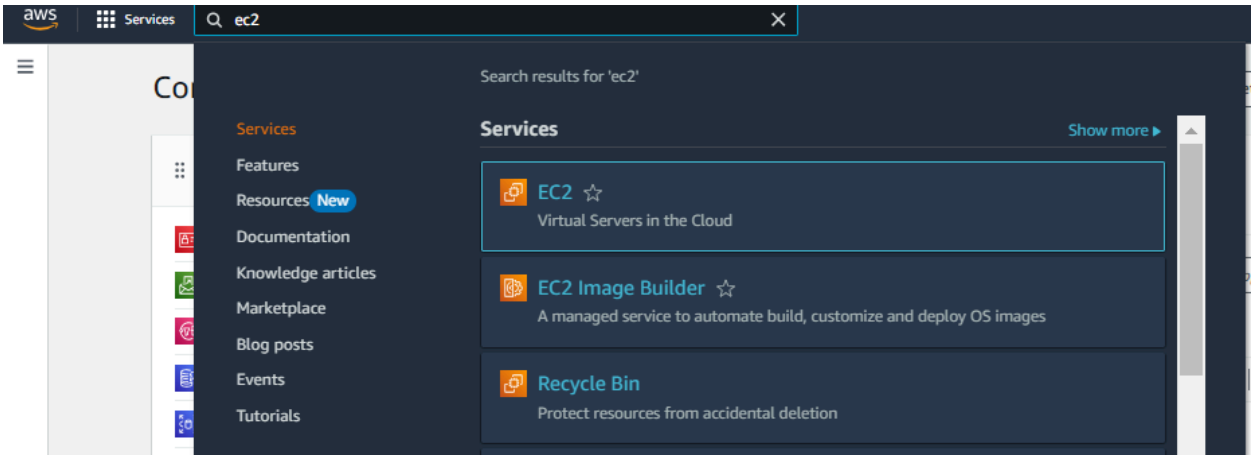
● Note: Load your Flask app and Html files into GitHub repository



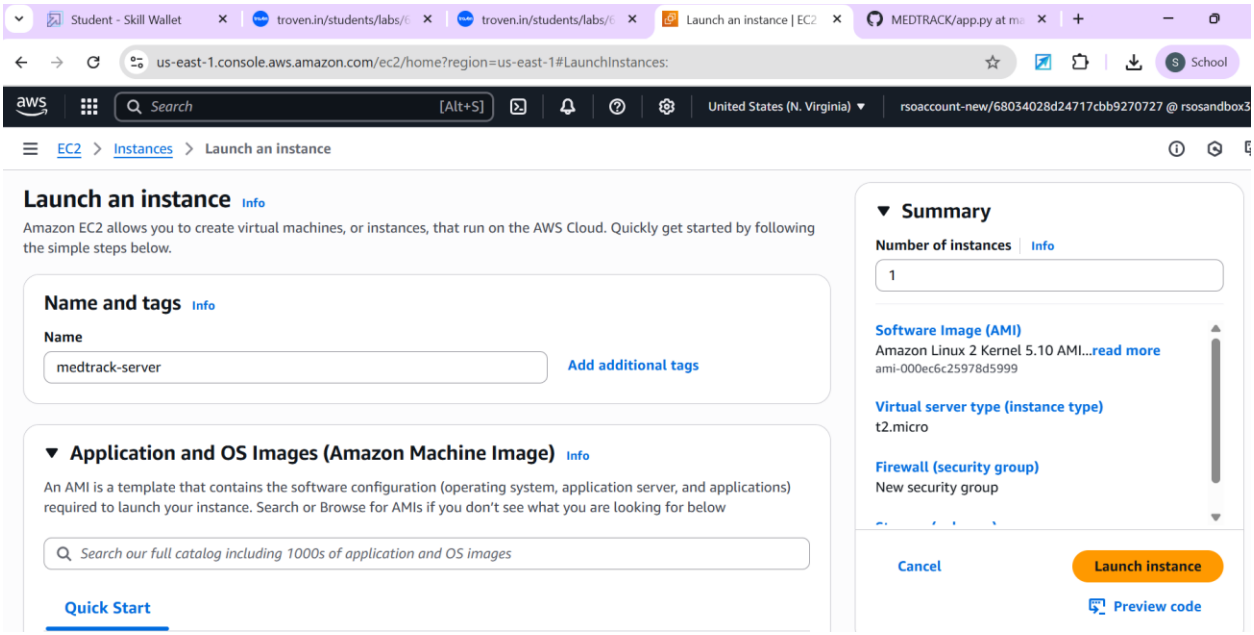
Activity 6.1: Launch an EC2 instance to host the Flask application.

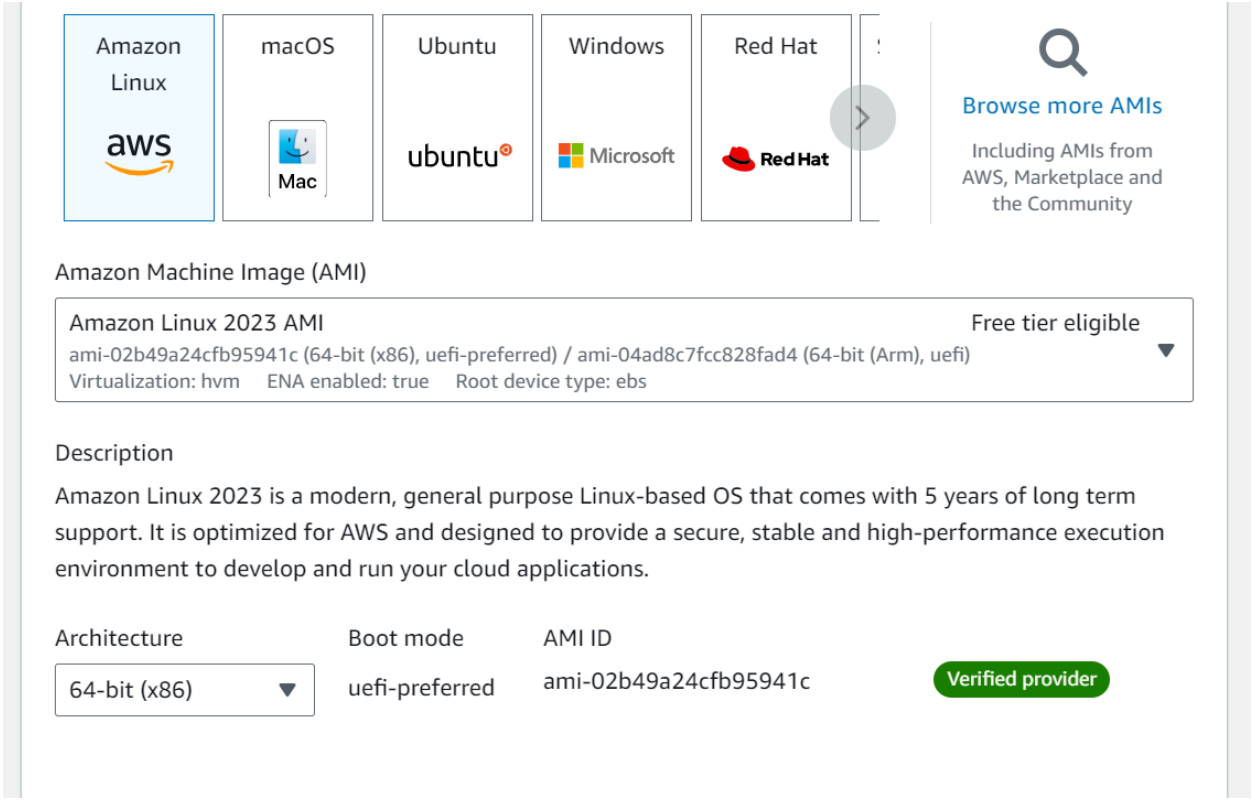
● Launch EC2 Instance

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

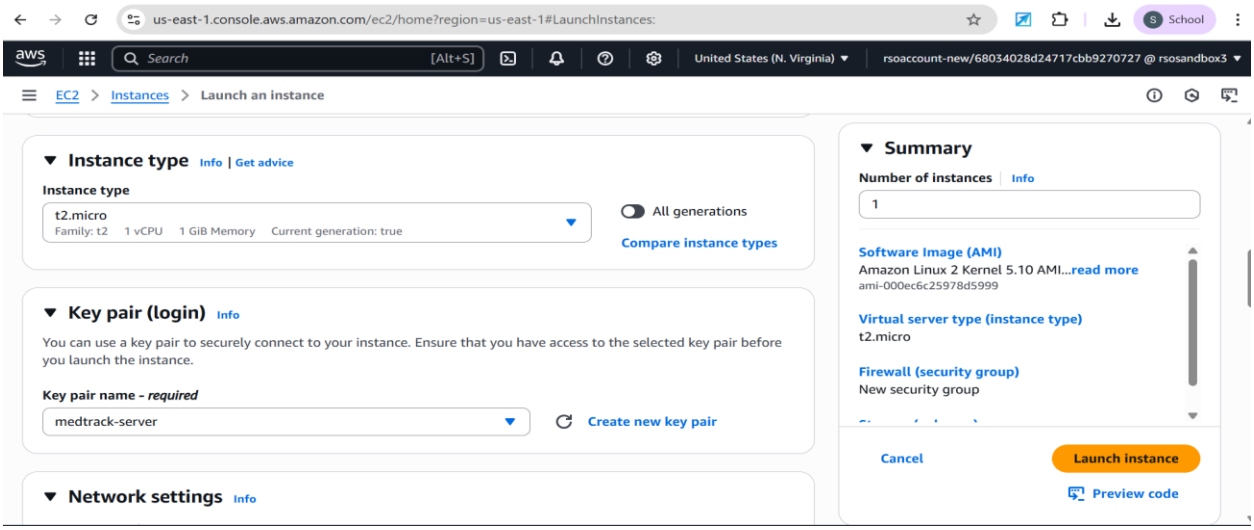


>Click on Launch instance to launch EC2 instance

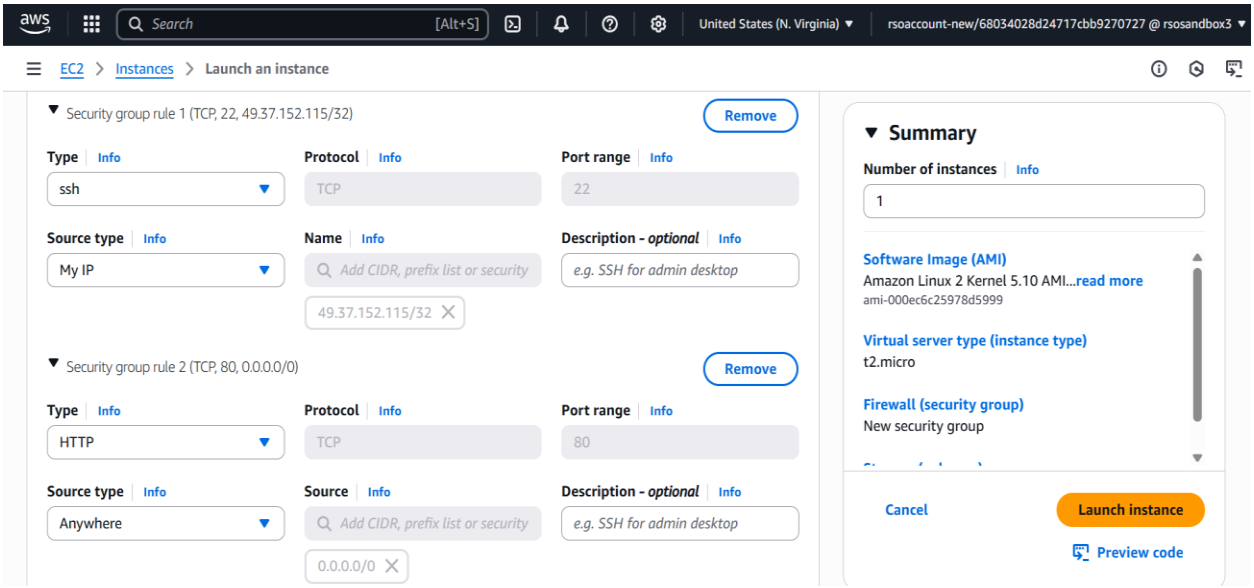

Launch an instance | EC2
us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#LaunchInstances:
Search [Alt+S]
United States (N. Virginia)
EC2 > Instances > Launch an instance
Launch an instance Info
Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.
Name and tags Info
Name
medtrack-server
Add additional tags
Application and OS Images (Amazon Machine Image) Info
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
Summary
Number of instances Info
1
Software Image (AMI)
Amazon Linux 2 Kernel 5.10 AMI...read more
ami-000ec6c25978d5999
Virtual server type (instance type)
t2.micro
Firewall (security group)
New security group
Cancel
Launch instance
Preview code

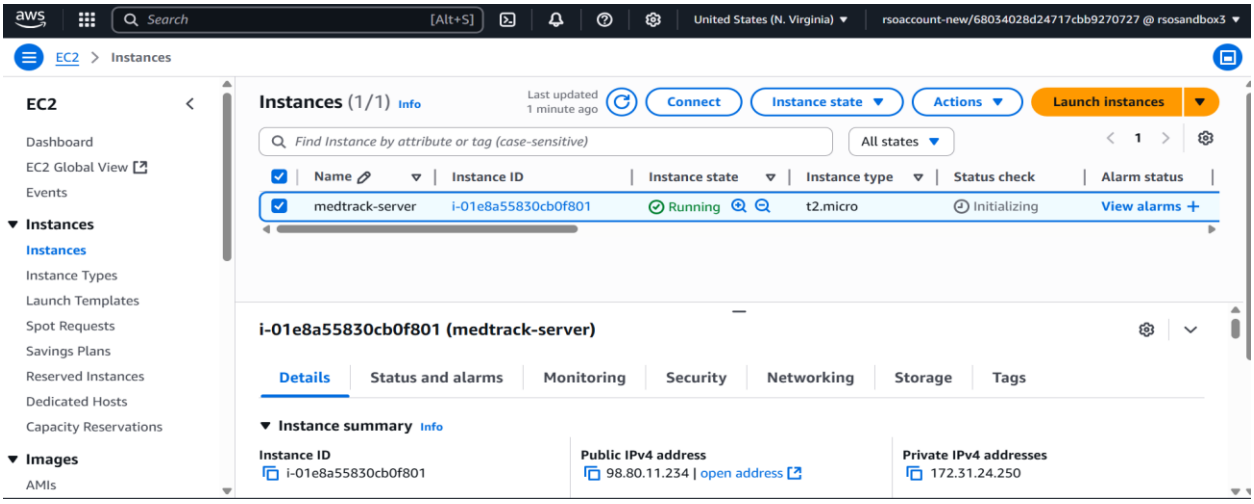

Amazon Linux
macOS
Mac
Ubuntu
ubuntu
Windows
Microsoft
Red Hat
Red Hat
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community
Amazon Machine Image (AMI)
Amazon Linux 2023 AMI
Free tier eligible
ami-02b49a24cfb95941c (64-bit (x86), uefi-preferred) / ami-04ad8c7fcc828fad4 (64-bit (Arm), uefi)
Virtualization: hvm ENA enabled: true Root device type: ebs
Description
Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.
Architecture
64-bit (x86)
Boot mode
uefi-preferred
AMI ID
ami-02b49a24cfb95941c
Verified provider

>Create and Download the keypair for server access:

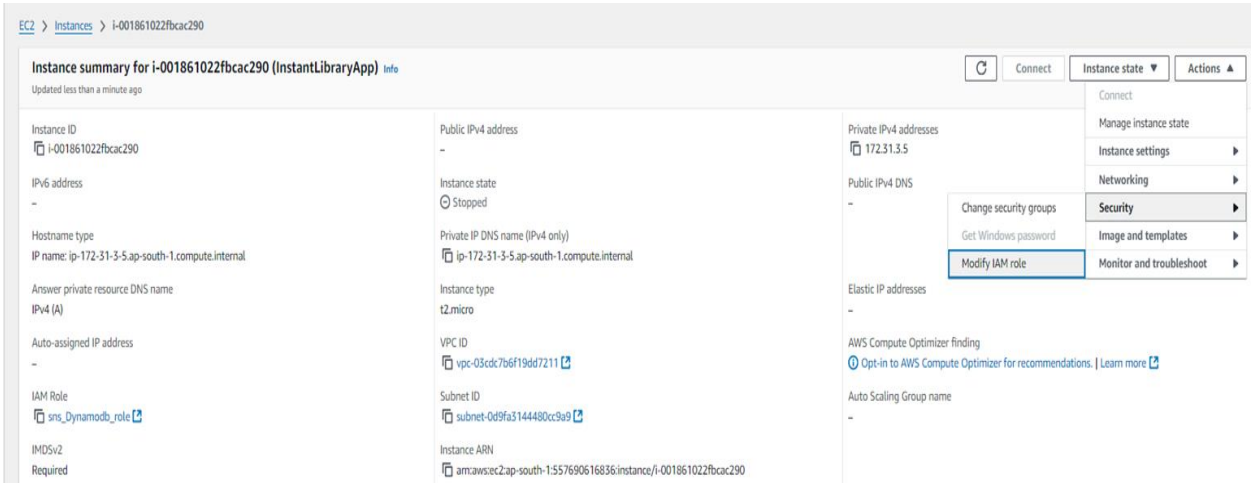


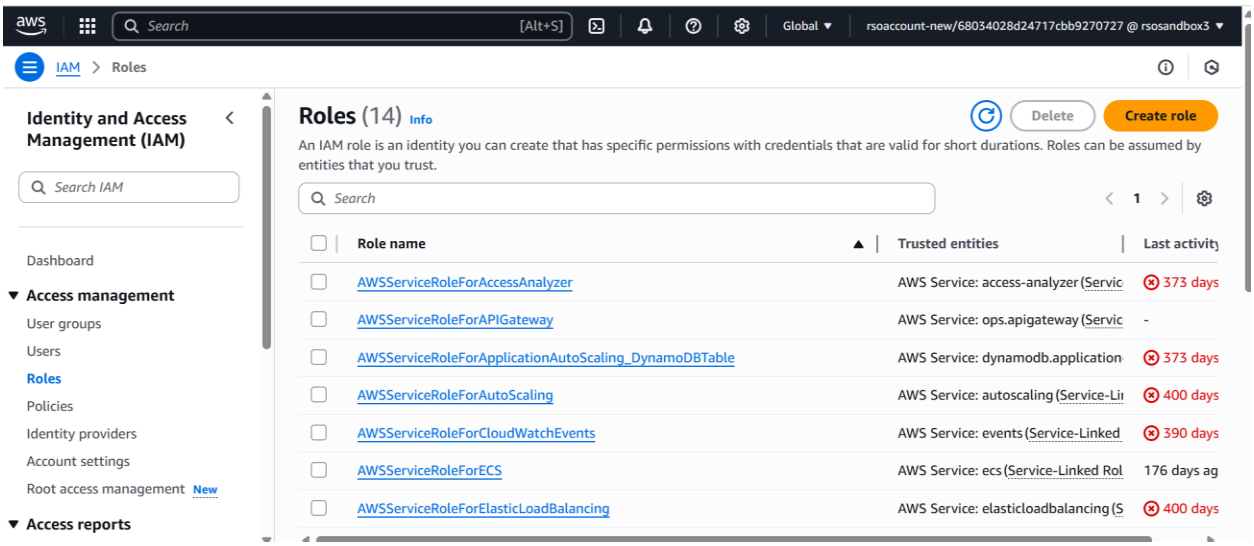
Activity 6.2:Configure security groups for HTTP, and SSH access.



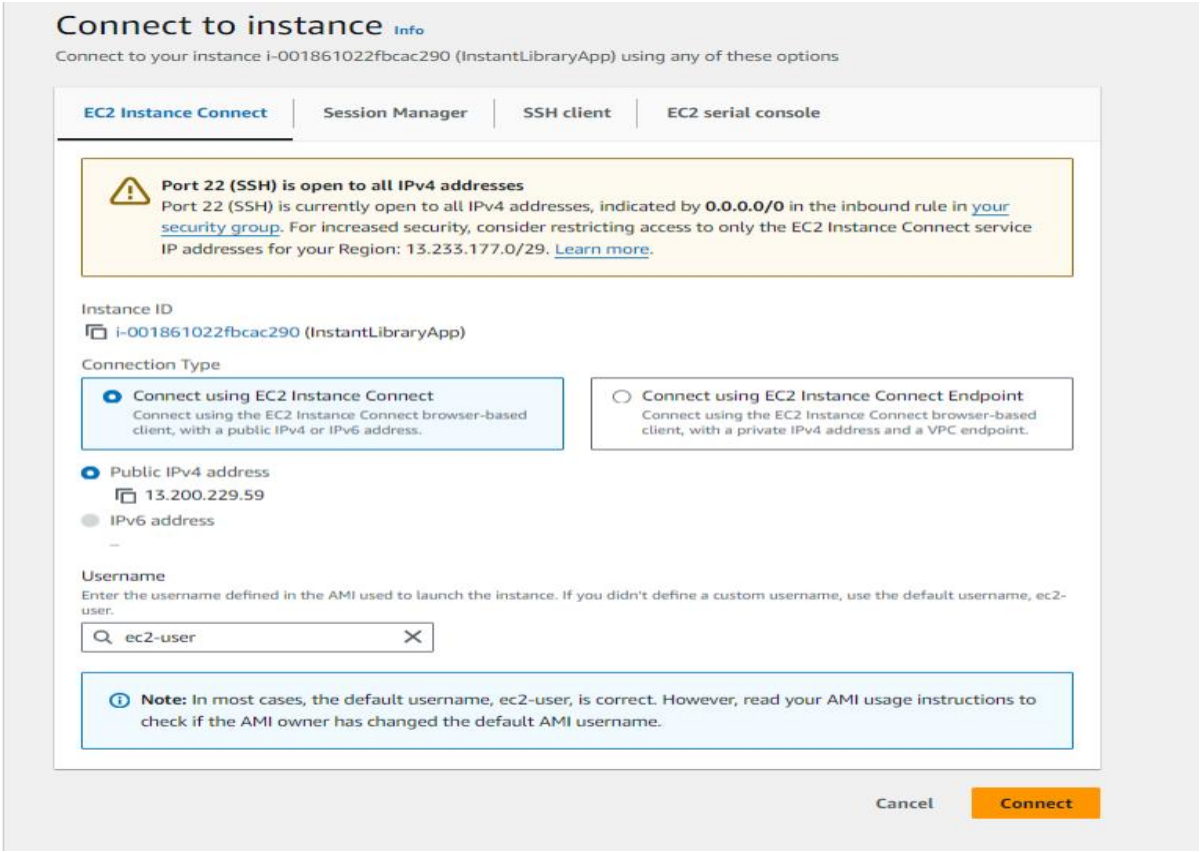


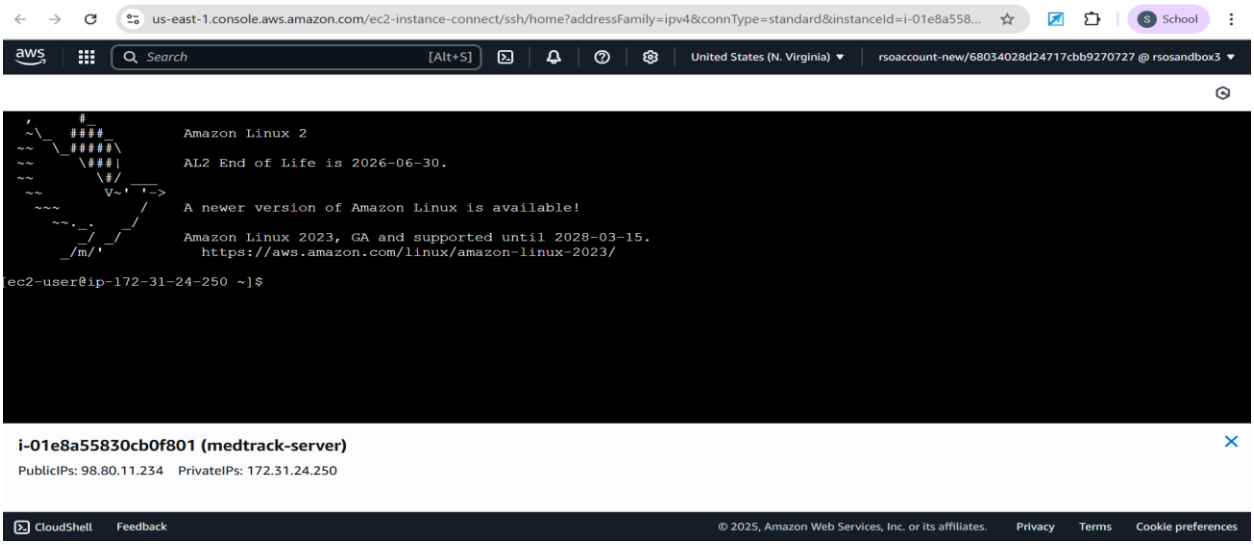
To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the IAM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





Now connect the EC2 with the files



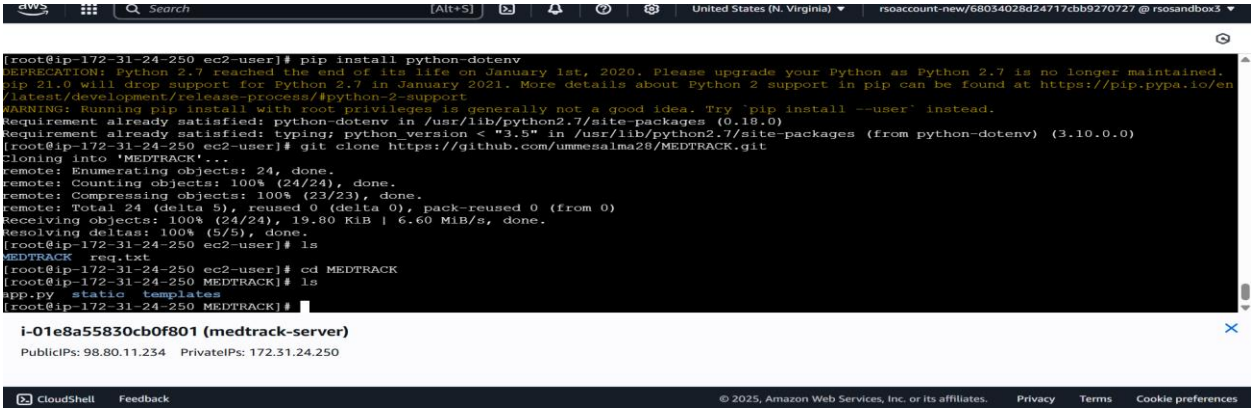


Milestone 7: Deployment on EC2
Activity 7.1: Install Software on the EC2 Instance

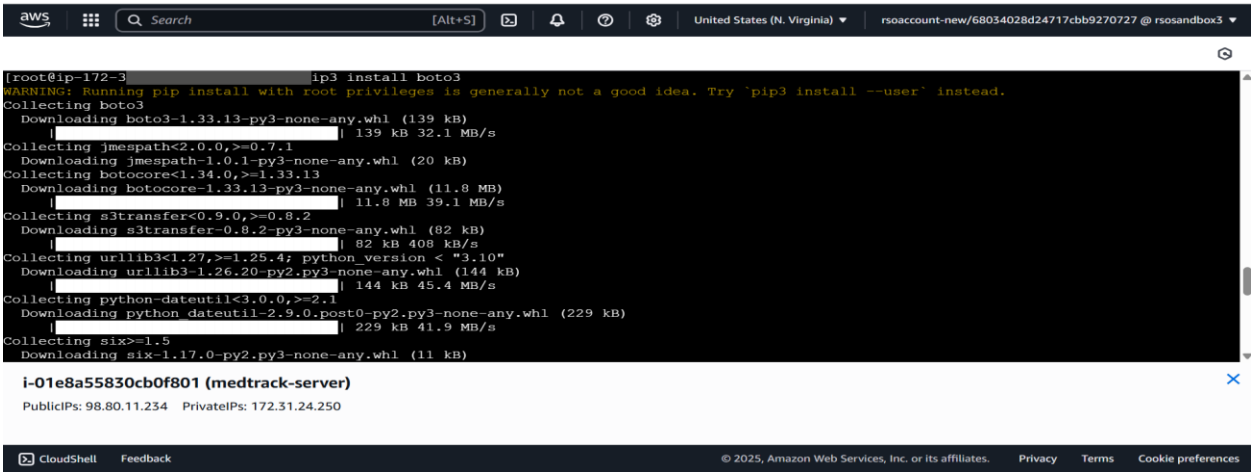
Install Python3, Flask, and Git:
On Amazon Linux 2:
sudo yum update -y
sudo yum install python3 git
sudo pip3 install flask boto3
Verify Installations:
flask --version
git --version

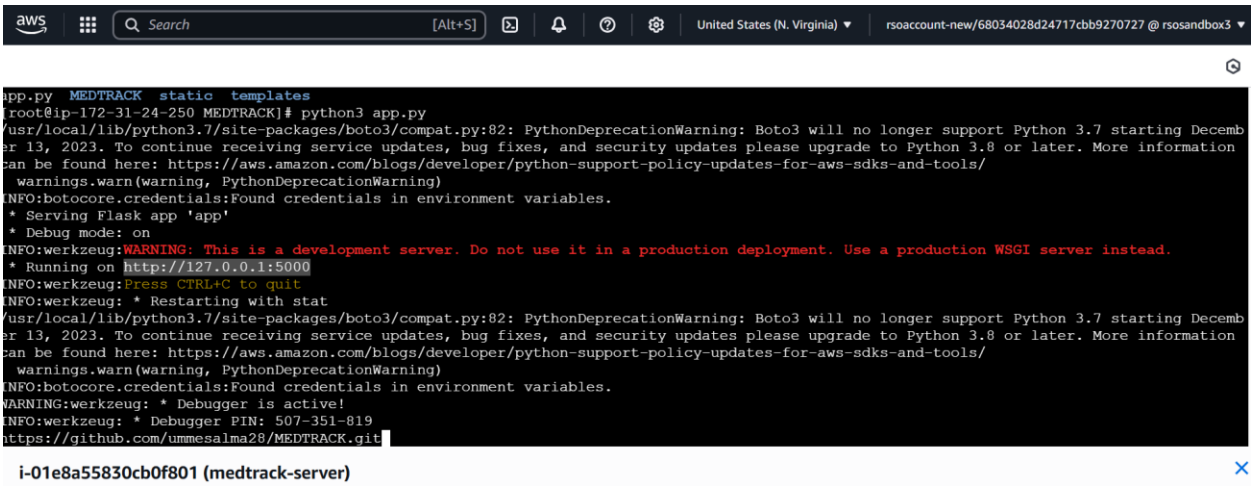
Activity 7.2:Clone Your Flask Project from GitHub
Clone your project repository from GitHub into the EC2 instance using Git

Run: git clone https://github.com/ummesalma28/MEDTRACK.git
This downloaded my project to the EC2 instance.
To navigate to the project directory, run the following command:
cd InstantLibrary



Verify the Flask app is running

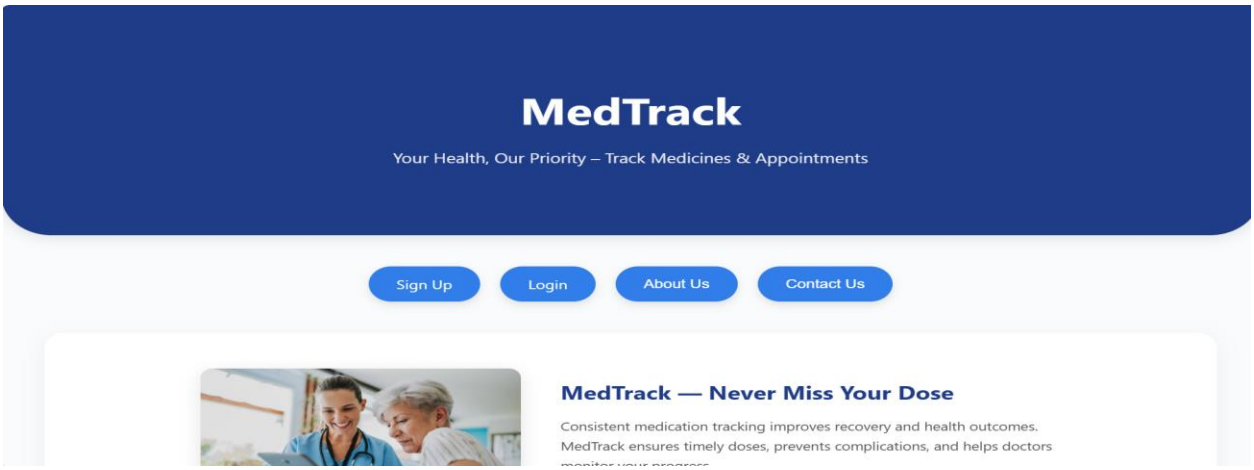


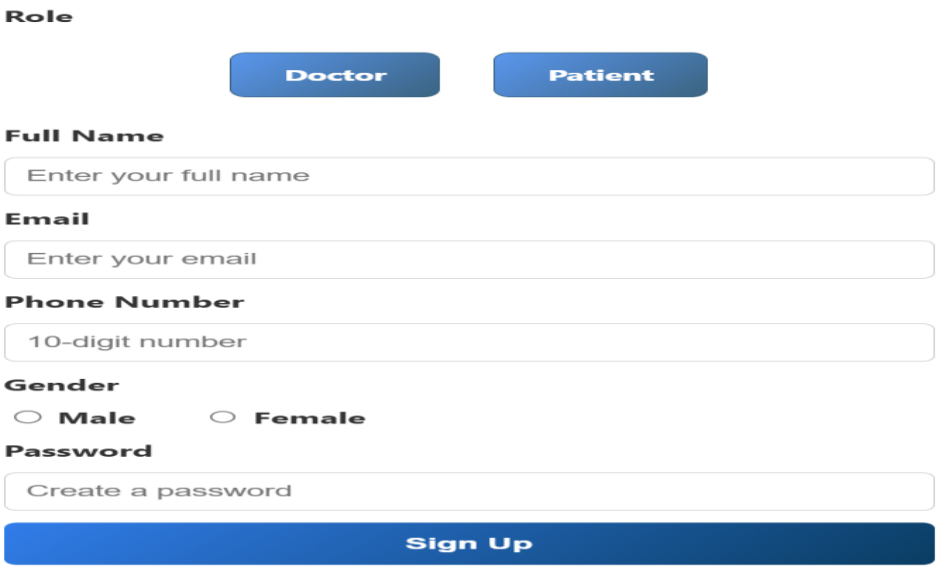


Access the website through: http://98.80.11.234:5000

Milestone 8: Testing and Deployment

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





Login

Role

Doctor | Patient

Email

228x1a42d6@khitguntur.ac.in

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

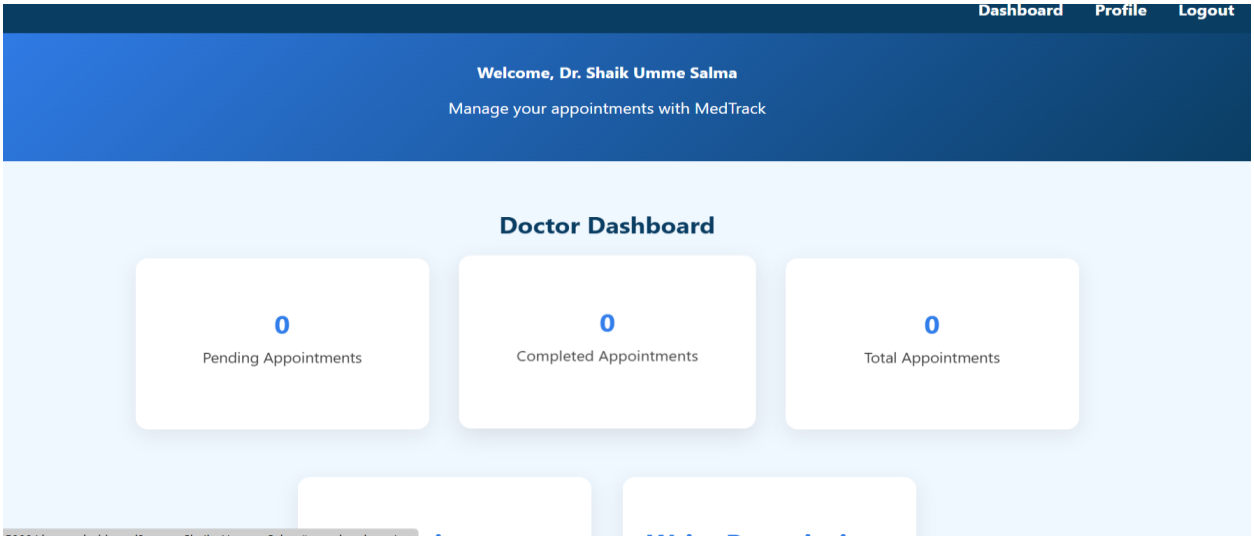
Book Appointment

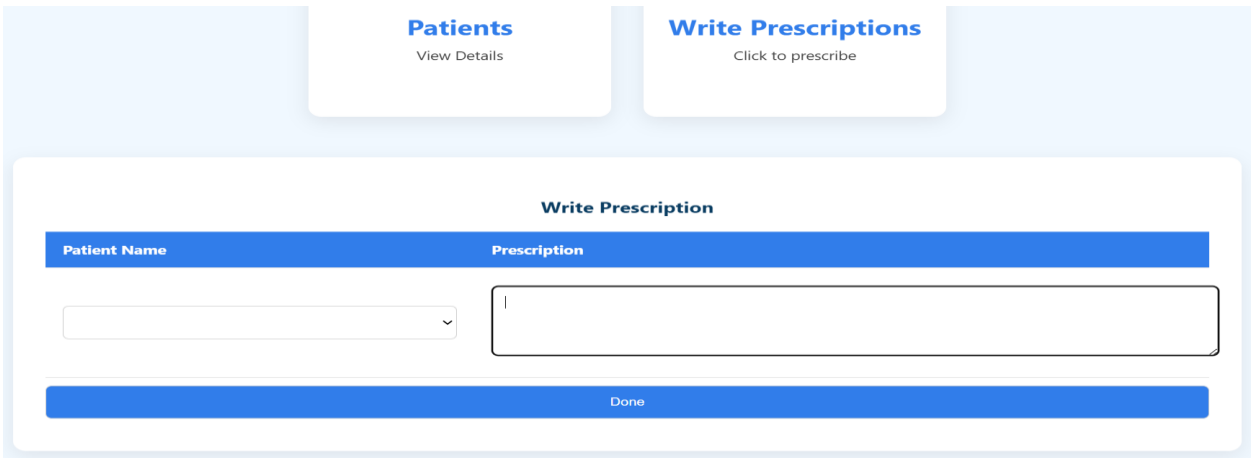
0	0	0
Upcoming Appointments	Completed Consultations	Prescriptions Available

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