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.

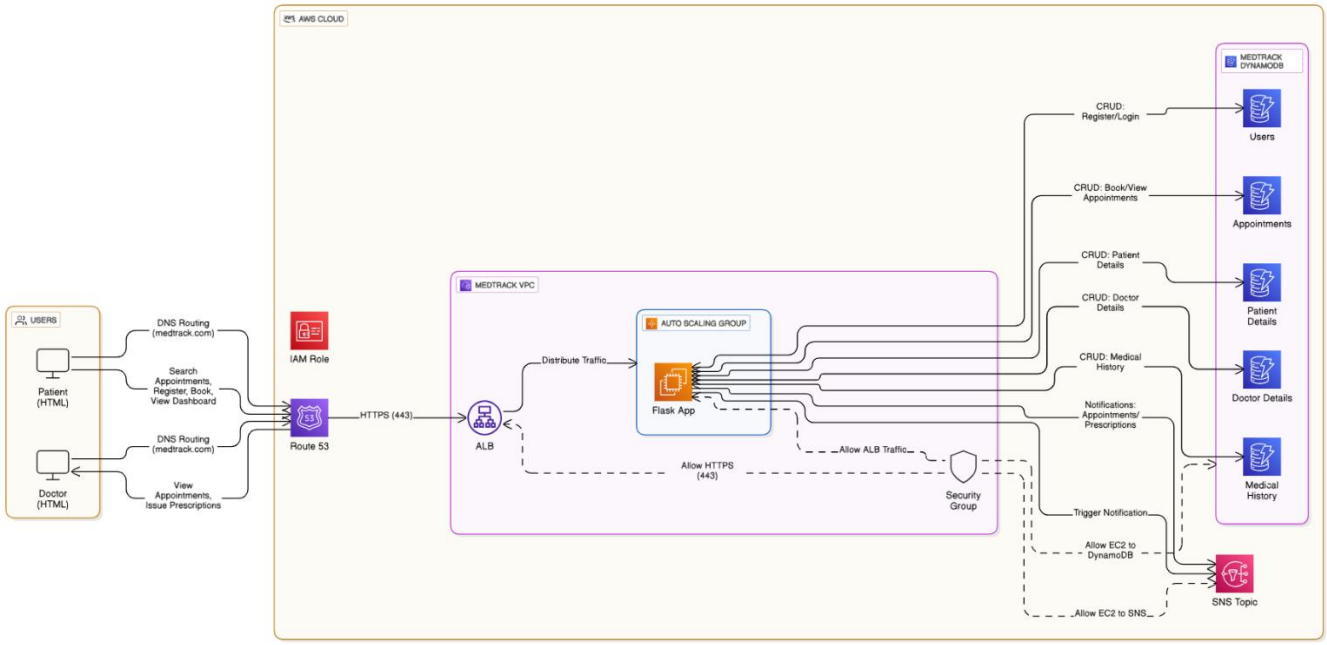
Scenario 2: Seamless Book Request Notifications for Library Staff

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.

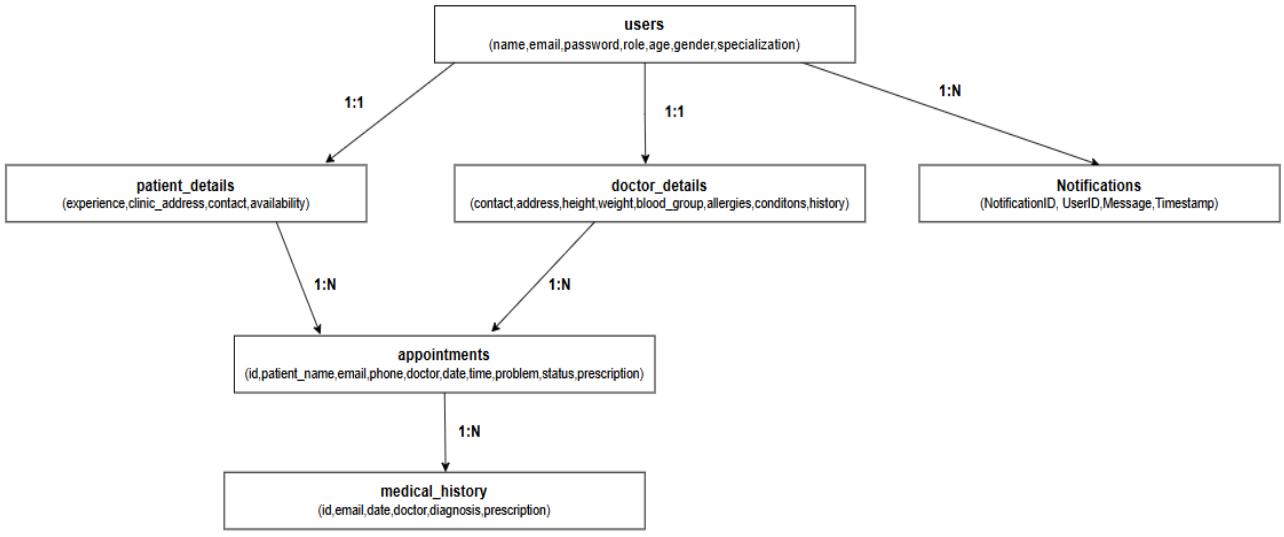
Scenario 3: Easy Access to Library Resources

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

AWS ARCHITECTURE



Entity Relationship (ER)Diagram:



Pre-requisites:

1. .**AWS Account Setup**:
   https://docs.aws.amazon.com/accounts/latest/reference/getting-started.html

2. **Understanding IAM**:
   https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

3. **Amazon EC2 Basics**:
   https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

4. **DynamoDB Basics**:
   https://docs.aws.amazon.com/amazondynamodb/Introduction.html

5. **SNS Overview**:
   https://docs.aws.amazon.com/sns/latest/dg/welcome.html

6. **Git Version Control**:
   https://git-scm.com/doc

Project Work Flow:

1.Backend Development and Application Setup

Activity 1.1:Develop the Backend Using Flask.

Activity 1.2: Integrate AWS Services Using boto3.

2.AWS Account Setup and Login

Activity 2.1: Set up an AWS account if not already done**.**

Activity 2.2: Log in to the AWS Management Console

3.DynamoDB Database Creation and Setup

Activity 3.1: Navigate to the DynamoDB

Activity 3.2: Create a DynamoDB table for storing data.

4.SNS Notification Setup

Activity 4.1: SNS topics for email notifications.

Activity 4.2: Subscribe users and Admin.

5.IAM Role Setup

Activity 5.1: Create IAM Role

Activity 5.2: Attach Policies

6.EC2 Instance Setup

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

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

7.Deployment on EC2

Activity 7.1: Install Software on the EC2 Instance.

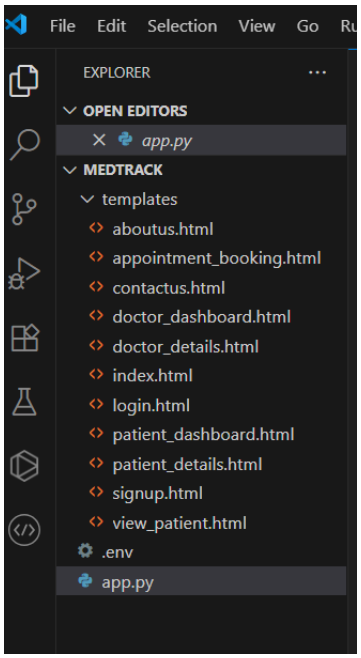
Activity 7.2: Clone Your Flask Project from GitHub

8.Testing and Deployment

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

Milestone 1:Backend Development and Application Setup

- **Activity 1.1: Develop the backend using Flask**
  - File Explorer Structure



Description of Codes:

- Imports:

```
from flask import Flask, render_template, request, redirect, session, url_for, flash, g
from datetime import datetime, timedelta
import os
import uuid
from functools import wraps
import boto3
from dotenv import load_dotenv
from werkzeug.security import generate_password_hash, check_password_hash
```

Description: Import core Flask utilities, Boto3 for DynamoDB access, SNS for notifications, environment loading via dotenv, and werkzeug.security for password hashing.

- Flask: routes & web framework
- render_template: to load HTML
- request: handle form submissions

- redirect, url_for: navigate between routes
- session: manage user login state
- flash: display alert
- generate_password_hash / check_password_hash: secure passwords
- boto3: AWS SDK for DynamoDB and SNS
- dotenv: read environment variables
- datetime, timedelta: timestamps
- uuid: unique IDs
- functools.wraps: decorators

- **app.py** initializes the Flask server with:

```
app = Flask(__name__)
app.secret_key = os.environ.get('SECRET_KEY', os.urandom(24))
app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(hours=1)
```

Description: initialize the Flask application instance using Flask(__name__) to start building the web app. Then initialize the Flask application with a random session secret key and configure the session lifetime.

- **DynamoDB Setup:**

```
AWS_REGION_NAME = os.environ.get('AWS_REGION_NAME', 'us-east-1')
USERS_TABLE_NAME = os.environ.get('USERS_TABLE_NAME', 'MedTrackUsers')
APPOINTMENTS_TABLE_NAME = os.environ.get('APPOINTMENTS_TABLE_NAME', 'MedTrackAppointments')
PATIENT_DETAILS_TABLE_NAME = os.environ.get('PATIENT_DETAILS_TABLE_NAME', 'MedTrackPatientDetails')
DOCTOR_DETAILS_TABLE_NAME = os.environ.get('DOCTOR_DETAILS_TABLE_NAME', 'MedTrackDoctorDetails')
MEDICAL_HISTORY_TABLE_NAME = os.environ.get('MEDICAL_HISTORY_TABLE_NAME', 'MedTrackMedicalHistory')

try:
    dynamodb = boto3.resource('dynamodb', region_name=AWS_REGION_NAME)
    sns = boto3.client('sns', region_name=AWS_REGION_NAME)
    SNS_TOPIC_ARN = os.environ.get('SNS_TOPIC_ARN')
    print("AWS credentials loaded using IAM Role or environment.")
except Exception as e:
    print(f"Error initializing DynamoDB or SNS: {e}")
    dynamodb = None
    sns = None
    SNS_TOPIC_ARN = None
```

Description:
Initializes DynamoDB resource for us-east-1 region and sets up references to MedTrack tables for storing users, appointments, doctor/patient details, and medical histories.

- **SNS Connection:**

```
def publish_to_sns(message, subject="MedTrack Notification"):
    if sns and SNS_TOPIC_ARN:
        try:
            sns.publish(
                TopicArn=SNS_TOPIC_ARN,
                Message=message,
                Subject=subject
            )
        except Exception as e:
            print(f"Error publishing to SNS: {e}")
    else:
        print("SNS not configured, skipping publish.")
```

Description:
Configures SNS notifications for sending alerts about user registration and appointments, by referencing the topic ARN from environment variables.

- **Helper Functions:**

```
def get_user_table():
    if dynamodb:
        return dynamodb.Table(USERS_TABLE_NAME)
    return None

Tabnine | Edit | Test | Explain | Document
def get_appointments_table():
    if dynamodb:
        return dynamodb.Table(APPOINTMENTS_TABLE_NAME)
    return None

Tabnine | Edit | Test | Explain | Document
def get_patient_details_table():
    if dynamodb:
        return dynamodb.Table(PATIENT_DETAILS_TABLE_NAME)
    return None

Tabnine | Edit | Test | Explain | Document
def get_doctor_details_table():
    if dynamodb:
        return dynamodb.Table(DOCTOR_DETAILS_TABLE_NAME)
    return None

Tabnine | Edit | Test | Explain | Document
def get_medical_history_table():
    if dynamodb:
        return dynamodb.Table(MEDICAL_HISTORY_TABLE_NAME)
    return None

Tabnine | Edit | Test | Explain | Document
def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'user_email' not in session:
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function
```

Description:
Defines helper methods to get specific DynamoDB tables and to wrap routes requiring login sessions.

- **Before Request Hook:**

```python
@app.before_request
def load_logged_in_user():
    user_email = session.get('user_email')
    g.user = None
    g.doctor_details = None
    if user_email:
        if dynamodb:
            user_table = get_user_table()
            try:
                response = user_table.get_item(Key={'email': user_email})
                if 'Item' in response:
                    g.user = response['Item']
                    if g.user['role'] == 'doctor':
                        doctor_table = get_doctor_details_table()
                        doc_resp = doctor_table.get_item(Key={'email': user_email})
                        g.doctor_details = doc_resp.get('Item')
            except Exception as e:
                print(f"Error loading user from DynamoDB: {e}")
```

Description:
Loads the current logged-in user into Flask's g object before each request, so the user session can be accessed throughout routes.

Routes:

- **Index Route (/):**

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

Description:
Renders the landing page index.html.

- **Signup Route (/signup):**

Description:
Handles GET to show the signup page, and POST to validate user inputs, hash passwords, save new users to DynamoDB, then redirect them to the relevant detail page based on role.

```python
@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')
        password = request.form.get('password')
        confirm_password = request.form.get('confirm_password')
        age = request.form.get('age')
        gender = request.form.get('gender')
        specialization = request.form.get('specialization') if role == 'doctor' else ''

        if password != confirm_password:
            return render_template('signup.html', password_error="Passwords do not match")

        hashed_password = generate_password_hash(password)

        user_data = {
            'email': email,
            'name': name,
            'password': hashed_password,
            'role': role,
            'age': age,
            'gender': gender,
            'specialization': specialization,
            'created_at': datetime.now().isoformat()
        }

        if dynamodb:
            user_table = get_user_table()
            try:
                user_table.put_item(Item=user_data)
                publish_to_sns(
                    f"Welcome {name}! Your {role} account has been created successfully.",
                    "New User Registration"
                )
            except Exception as e:
```

```python
    else:
        # local_db['users'][email] = user_data
        pass

    session.clear()
    session['user_email'] = email

    return redirect(url_for('doctor_details' if role == 'doctor' else 'patient_details'))

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

- **Login Route (/login):**

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = None

        if dynamodb:
            user_table = get_user_table()
            try:
                response = user_table.get_item(Key={'email': email})
                if 'Item' in response:
                    user = response['Item']
            except Exception as e:
                print(f"Error fetching user: {e}")
        else:
            # user = local_db['users'].get(email)
            pass

        if user and check_password_hash(user['password'], password):
            session.clear()
            session['user_email'] = email
            return redirect(url_for('doctor_dashboard' if user['role'] == 'doctor' else 'patient_dashboard'))
        else:
            flash("Invalid credentials")

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

Description:
Handles GET to render the login page, POST to verify credentials from DynamoDB, check password, and redirect to the role-based dashboard.

- **Logout Route (/logout):**

```python
@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('index'))
```

Description:
Clears the user session and redirects to the home page.

- **Patient Dashboard Route (/patient_dashboard):**

Description:
Displays patient dashboard with appointments, personal details, and medical history pulled from DynamoDB.

```python
@app.route('/patient_dashboard')
@login_required
def patient_dashboard():
    if g.user['role'] != 'patient':
        return redirect(url_for('login'))

    appointments = []
    history = []
    patient_details = None
    if dynamodb:
        appointments_table = get_appointments_table()
        details_table = get_patient_details_table()
        history_table = get_medical_history_table()
        try:
            a_resp = appointments_table.scan()
            appointments = [a for a in a_resp['Items'] if a['email'] == g.user['email']]
            d_resp = details_table.get_item(Key={'email': g.user['email']})
            patient_details = d_resp.get('Item')
            h_resp = history_table.scan()
            history = [h for h in h_resp['Items'] if h['email'] == g.user['email']]
        except Exception as e:
            print(f"Error loading patient dashboard: {e}")
```

- **Appointment Booking Route (/appointment_dashboard):**

```python
@app.route('/appointment_dashboard')
@login_required
def appointment_dashboard():
    if g.user['role'] != 'patient':
        return redirect(url_for('login'))

    doctors = []
    if dynamodb:
        user_table = get_user_table()
        doctor_details_table = get_doctor_details_table()
        try:
            response = user_table.scan()
            doctors = [u for u in response['Items'] if u['role'] == 'doctor']
            for doc in doctors:
                details_resp = doctor_details_table.get_item(Key={'email': doc['email']})
                details = details_resp.get('Item')
                doc['availability'] = details.get('availability', 'Not provided') if details else 'Not provided'
        except Exception as e:
            print(f"Error fetching doctors: {e}")
```

Description:
Lets patients view available doctors and schedule appointments.

- **Doctor Dashboard Route(/doctor_dashboard):**

Description:
Define /doctor_dashboard route to render the doctor's personal dashboard, showing their scheduled appointments (filtered by doctor name), along with their own stored profile details if available. This helps doctors manage appointments and see their patients.

```python
@app.route('/doctor_dashboard')
@login_required
def doctor_dashboard():
    if g.user['role'] != 'doctor':
        return redirect(url_for('login'))

    appointments = []
    if dynamodb:
        table = get_appointments_table()
        try:
            response = table.scan()
            appointments = [a for a in response['Items'] if a['doctor'] == g.user['name']]
        except Exception as e:
            print(f"Error fetching appointments: {e}")
    else:
        # appointments = [
        #     a for patient_appts in local_db['appointments'].values()
        #     for a in patient_appts if a['doctor'] == g.user['name']
        # ]
        pass

    for a in appointments:
        if 'status' not in a:
            a['status'] = 'Scheduled'

    return render_template('doctor_dashboard.html', user=g.user, appointments=appointments, doctor_details=g.doctor_deta
```

- **Patient Details Route(/patient_details):**

```python
@app.route('/patient_details')
@login_required
def patient_details():
    if g.user['role'] != 'patient':
        return redirect(url_for('login'))
    return render_template('patient_details.html', user=g.user)
```

Description:
Define /patient_details route to render the patient_details.html page, allowing a patient to complete their profile with personal medical details, like contact, address, allergies, and past conditions.

- **Doctor Details Route(/doctor_details):**

```python
@app.route('/doctor_details')
@login_required
def doctor_details():
    if g.user['role'] != 'doctor':
        return redirect(url_for('login'))
    return render_template('doctor_details.html')
```

Description:
Define /doctor_details route to render the doctor_details.html page, where a doctor can add or edit their own professional profile details (experience, contact, availability, etc.) right after signing up or while updating.

- **Exit & Static Info Routes:**

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

Tabnine | Edit | Test | Explain | Document
@app.route('/contactus')
def contactus():
    return render_template('contactus.html')
```

Description:
Simple routes for About Us and Contact Us pages.

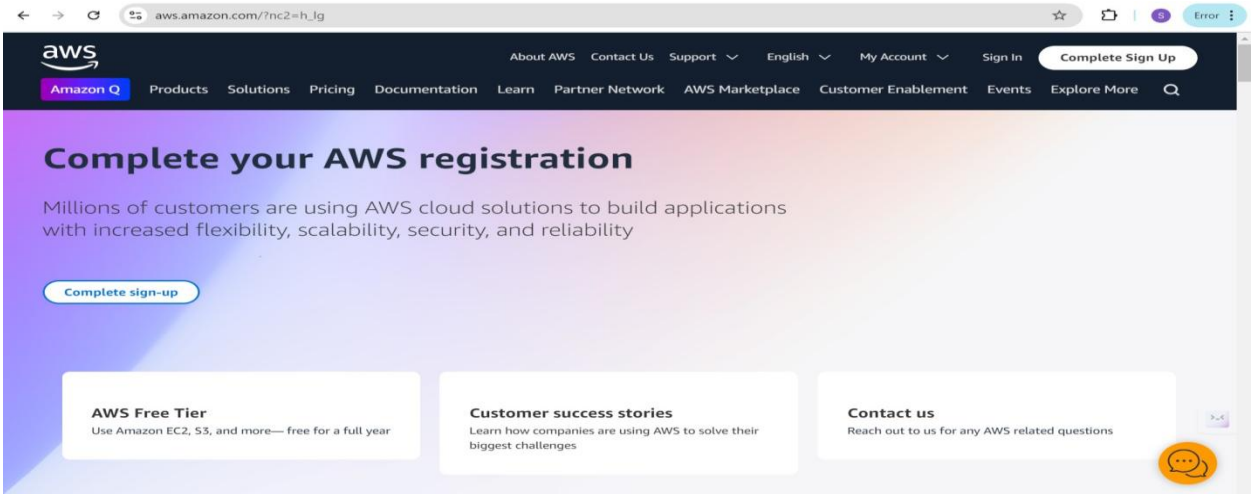
- **Deployment Code:**

```
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=5000, debug=True)
```

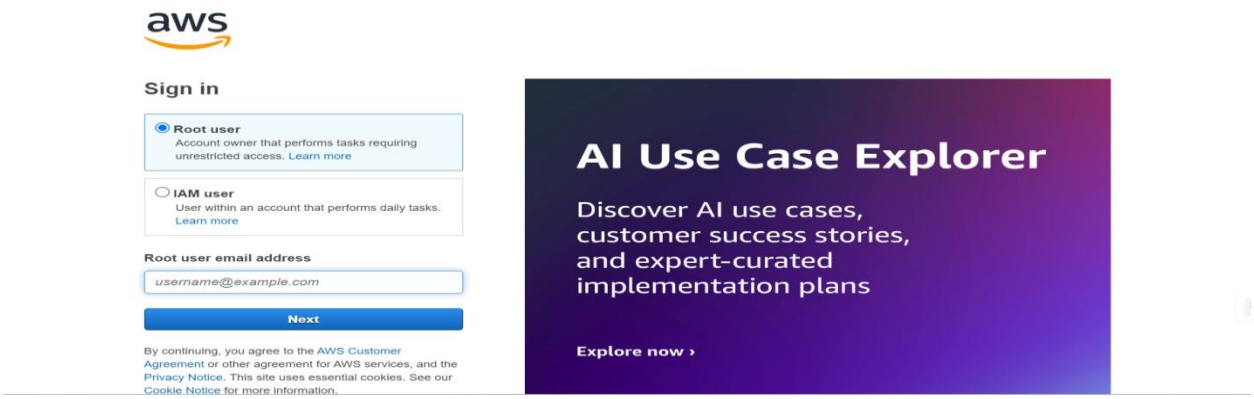
Description:
Starts the Flask server listening on 0.0.0.0 port 5000 with debug=True.

Milestone 2: AWS Account Setup and Login

- **Activity 2.1: Set up an AWS account if not already done.**
  - Sign up for an AWS account and configure billing settings.

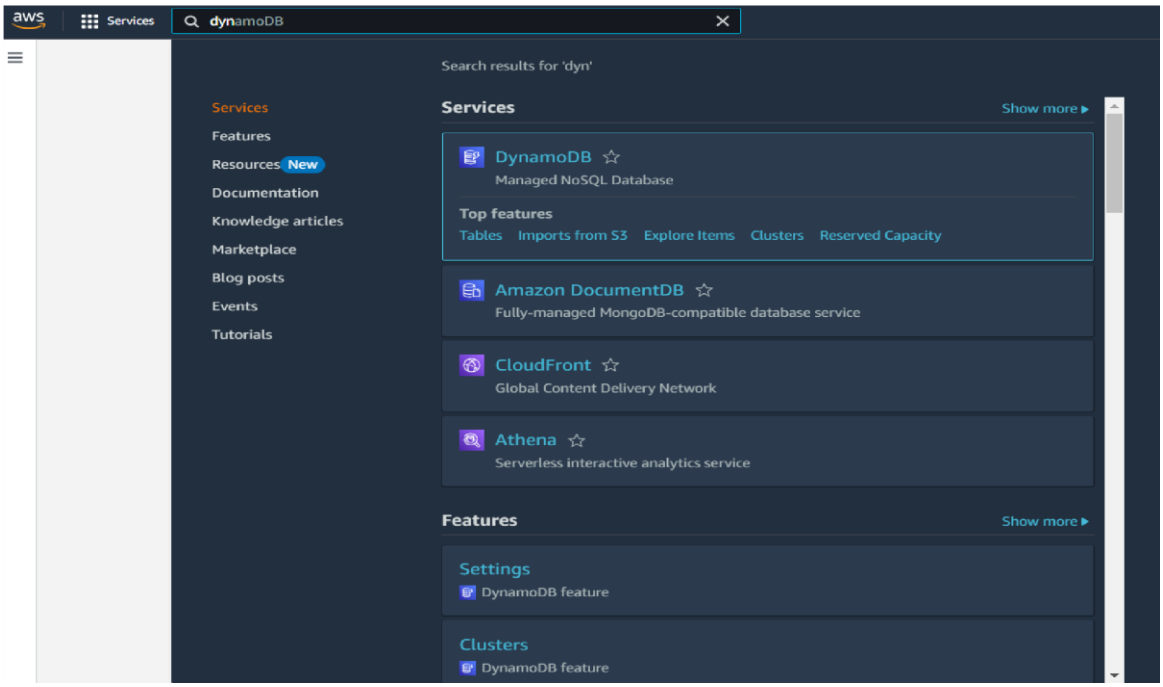


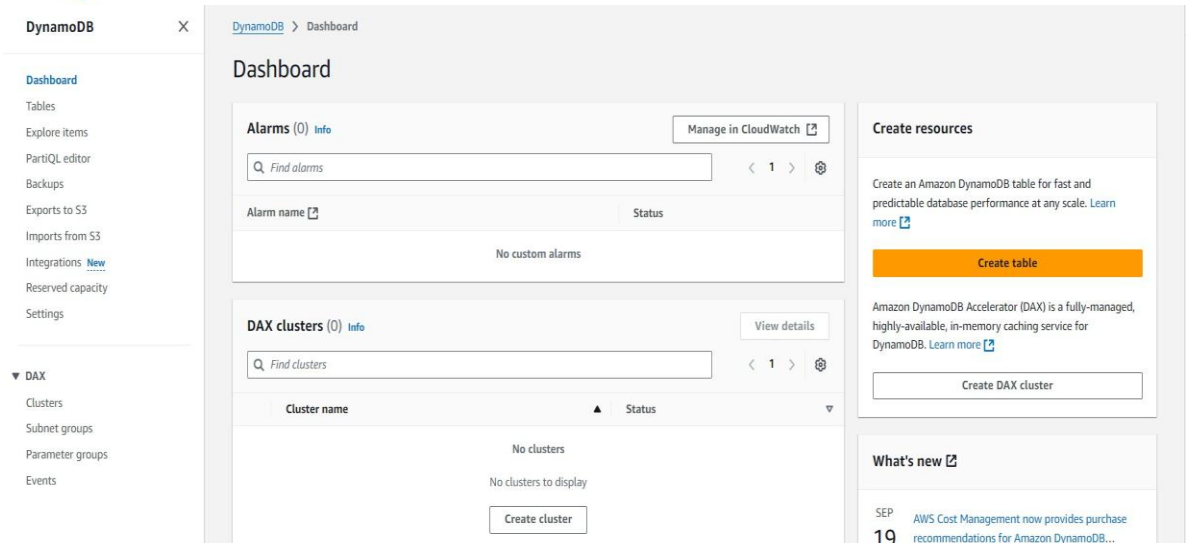
- **Activity 2.2: Log in to the AWS Management Console**
  - After setting up your account, log in to the AWS Management Console.

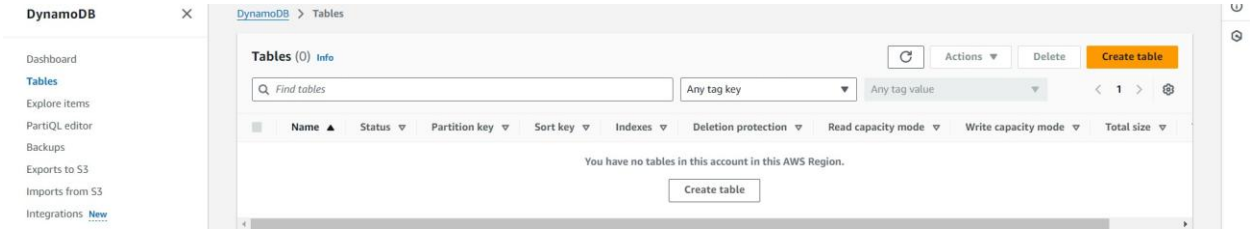


Milestone 3: DynamoDB Database Creation and Setup

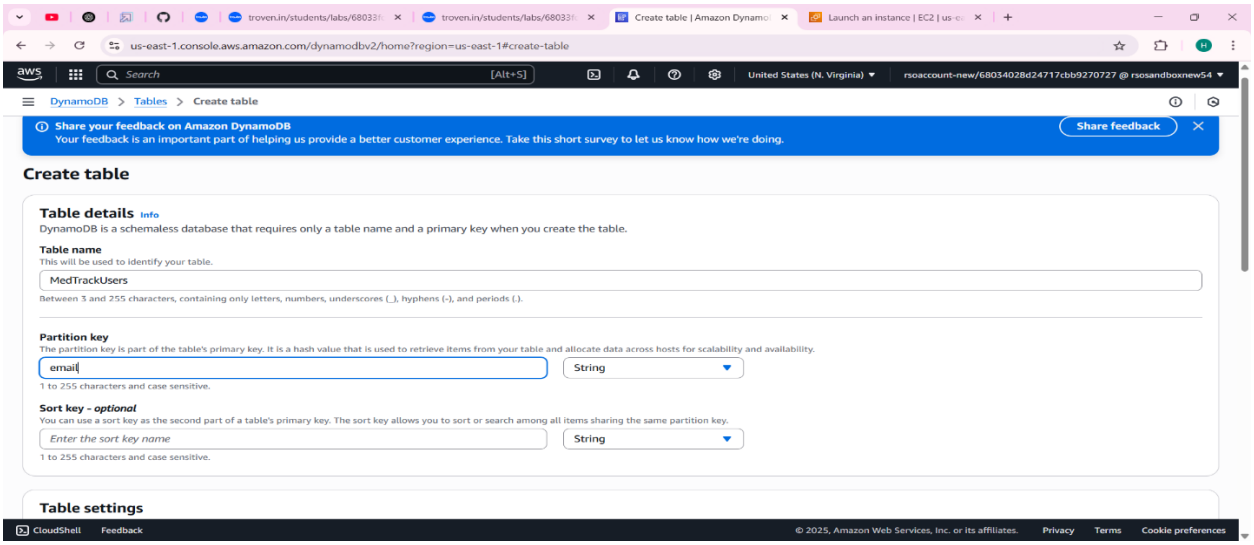
- **Activity 3.1:Navigate to the DynamoDB**
    - In the AWS Console, navigate to DynamoDB and click on create tables.







- **Activity 3.2:Create a DynamoDB table for storing registration details.**
    - Create MedTrackUsers table with partition key 'email' with type String and click on create tables for storing user data.



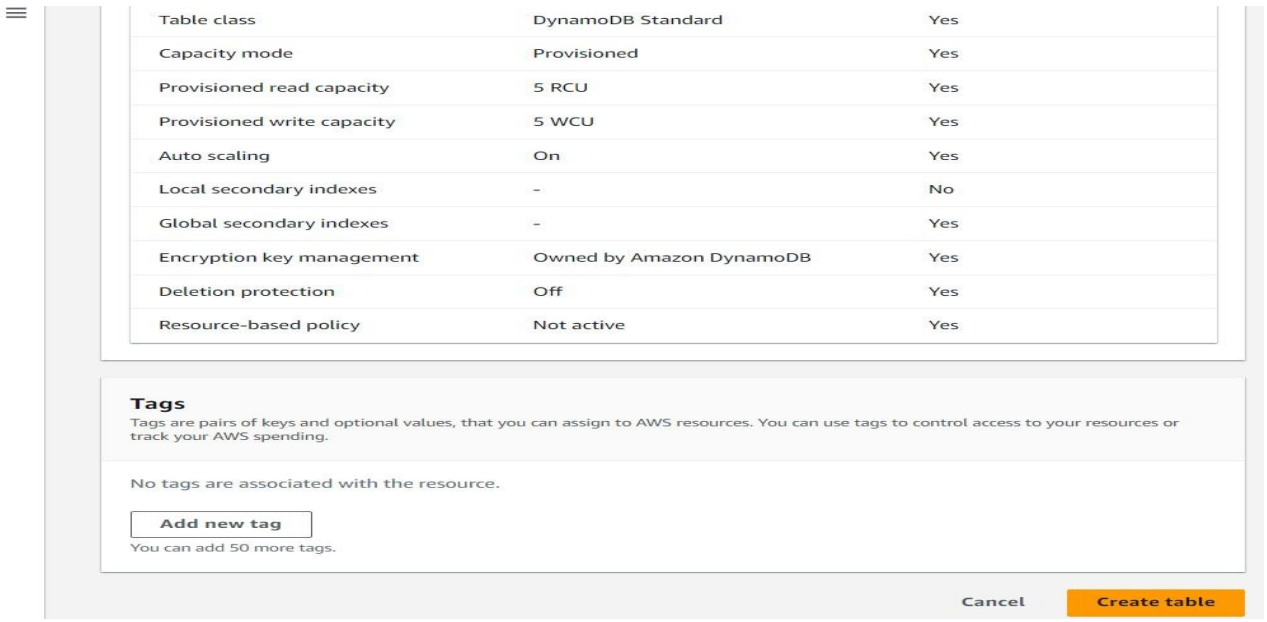


Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based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

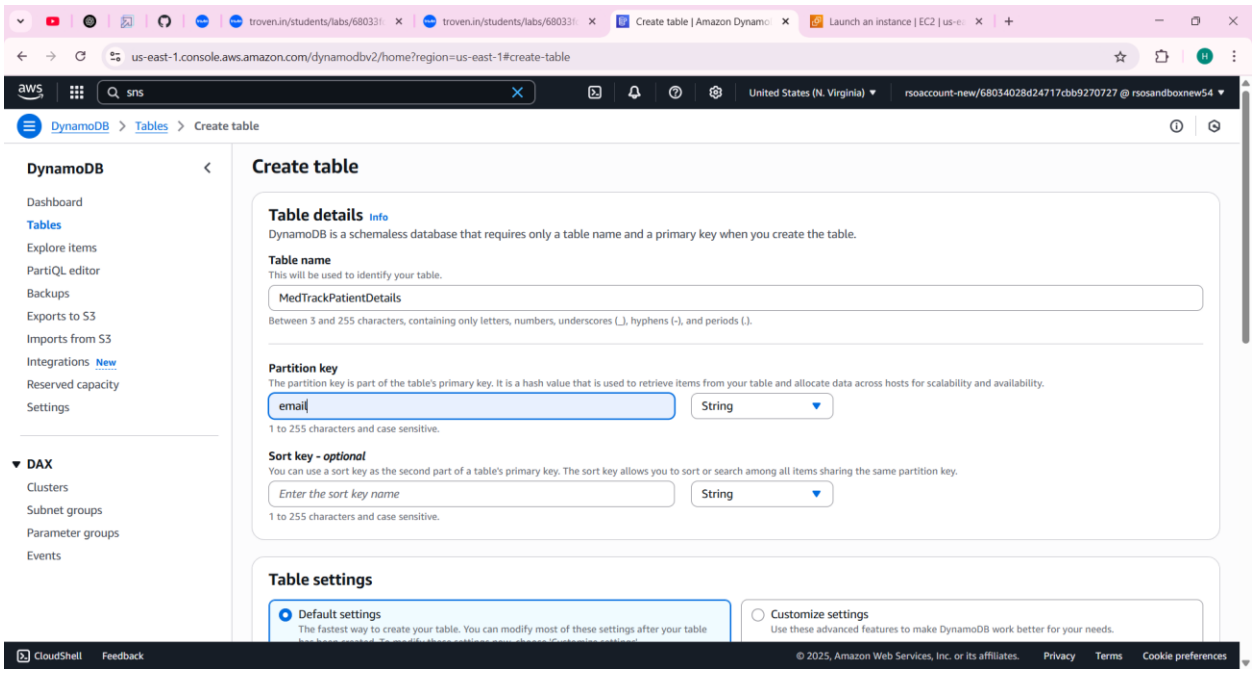
No tags are associated with the resource.

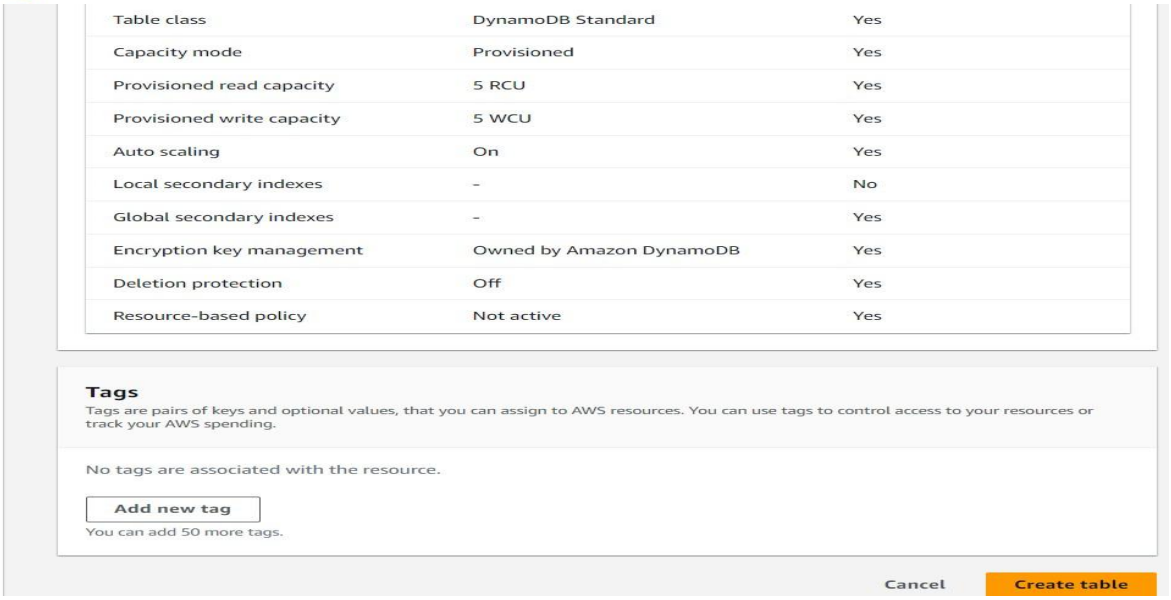
Add new tag

You can add 50 more tags.

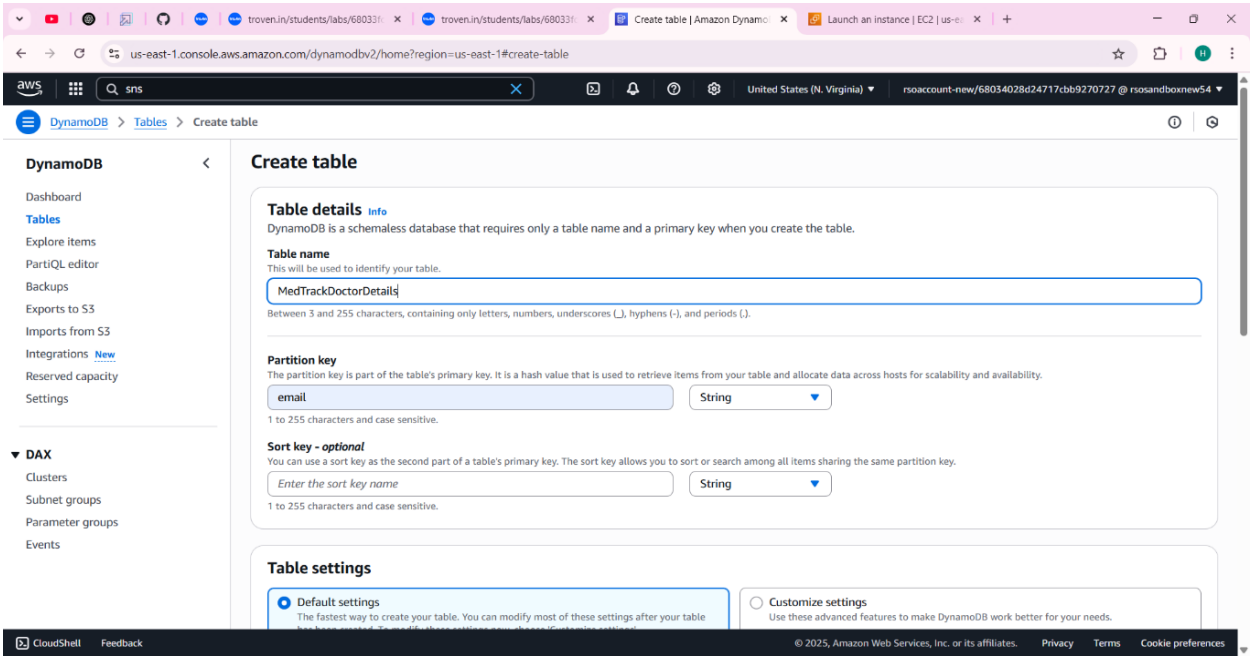
Cancel | Create table

- Follow the same steps to create a MedTrackPatientDetails table with 'email' as the partition key with type string to store patient details.

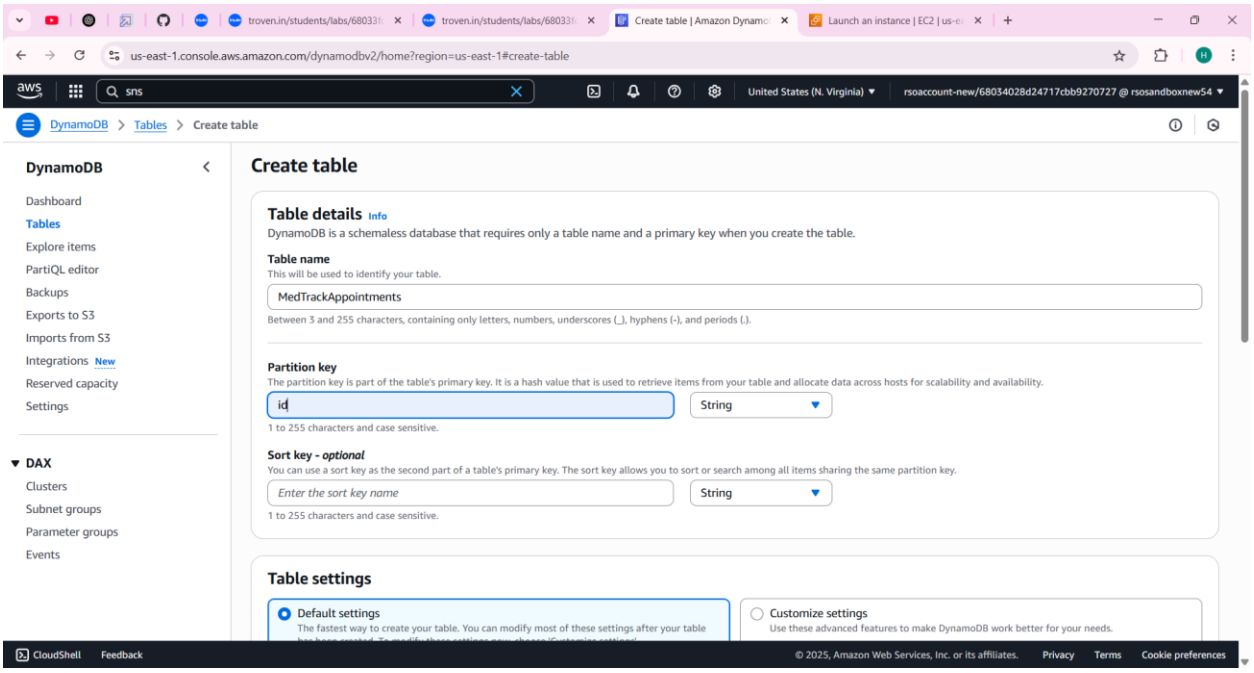




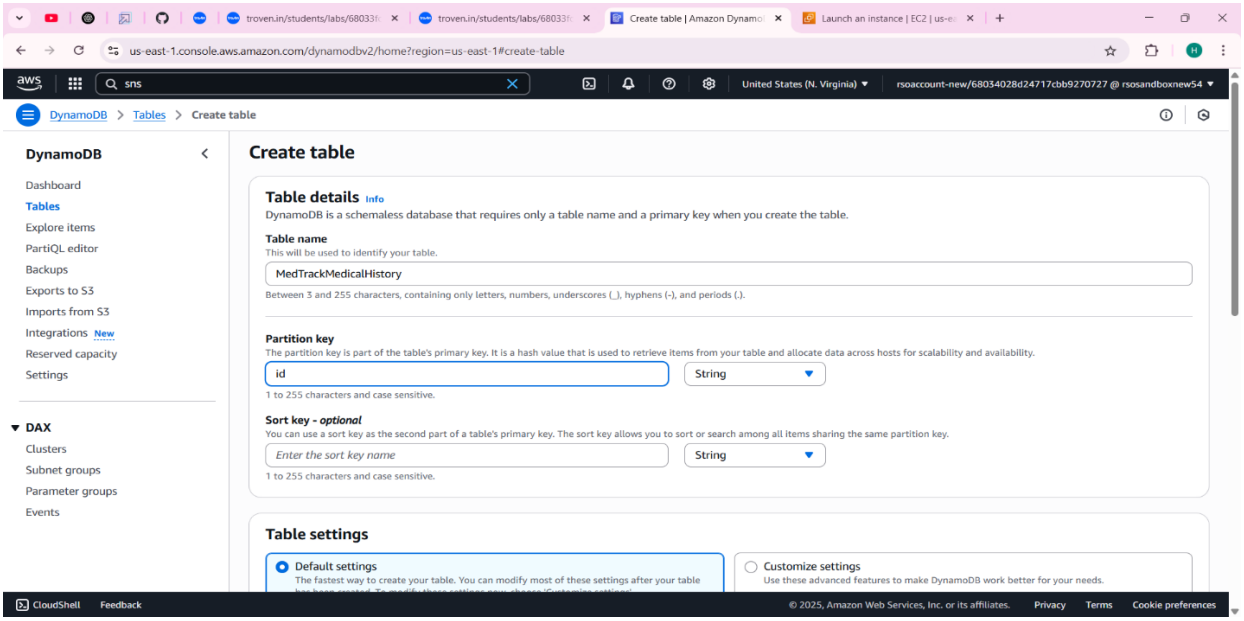
- Follow the same steps to create a MedTrackDoctorDetails table with 'email' as the partition key with type string to store patient details.



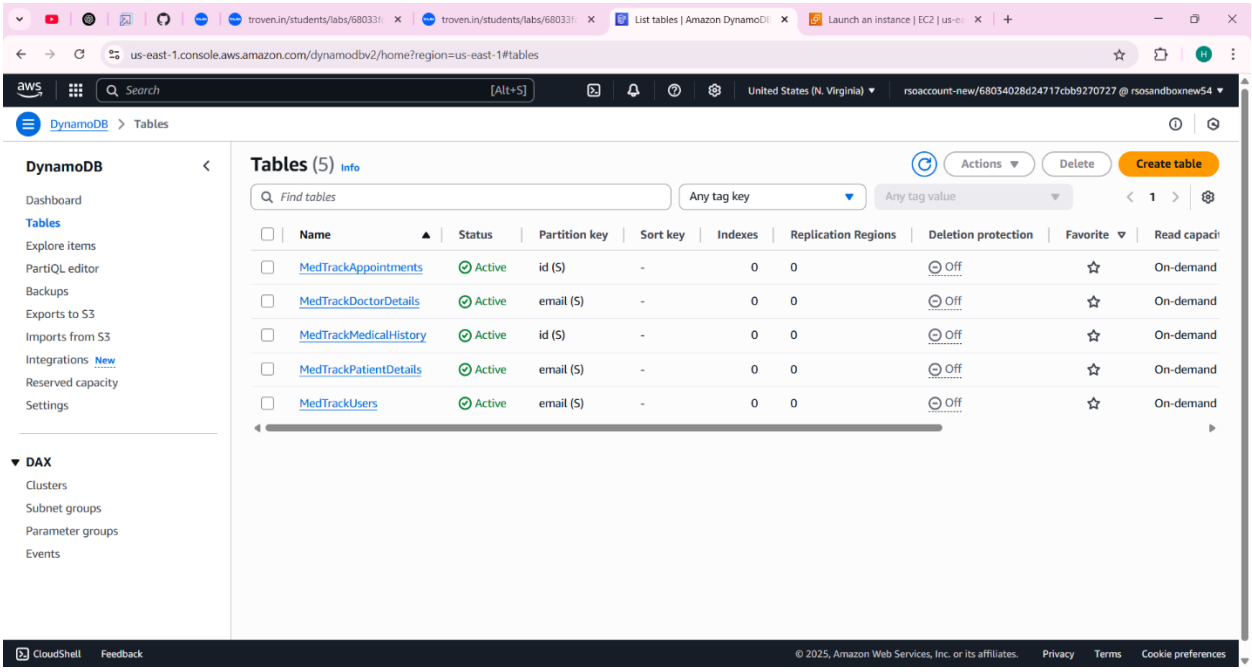
- Follow the same steps to create a MedTrackAppointments table with 'id' as the partition key with type string to store appointments details.



- Follow the same steps to create a MedTrackMedicalHistory table with 'id' as the partition key with type string to store details of medical history.

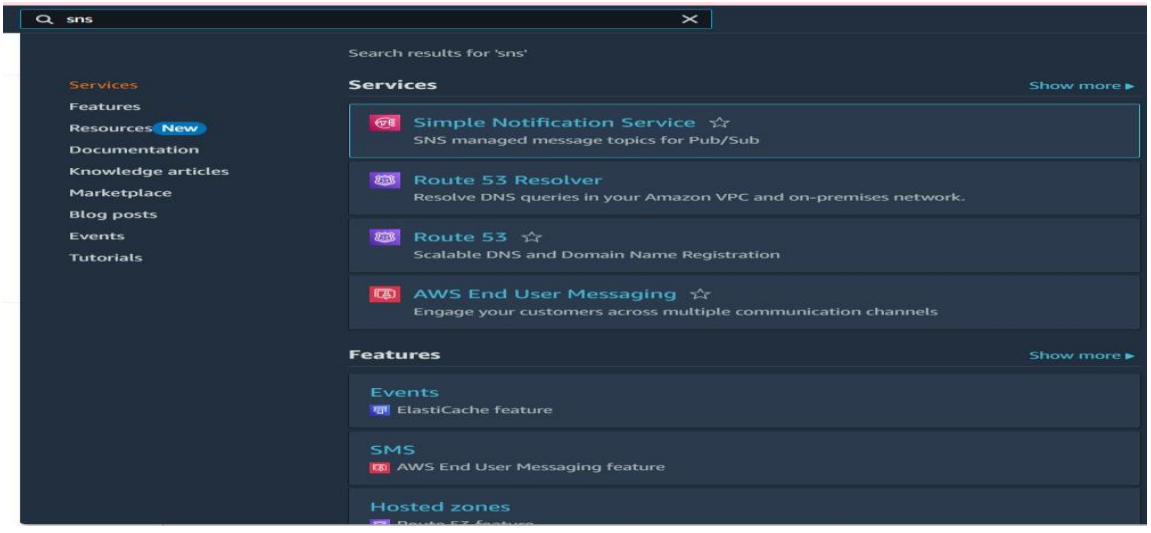


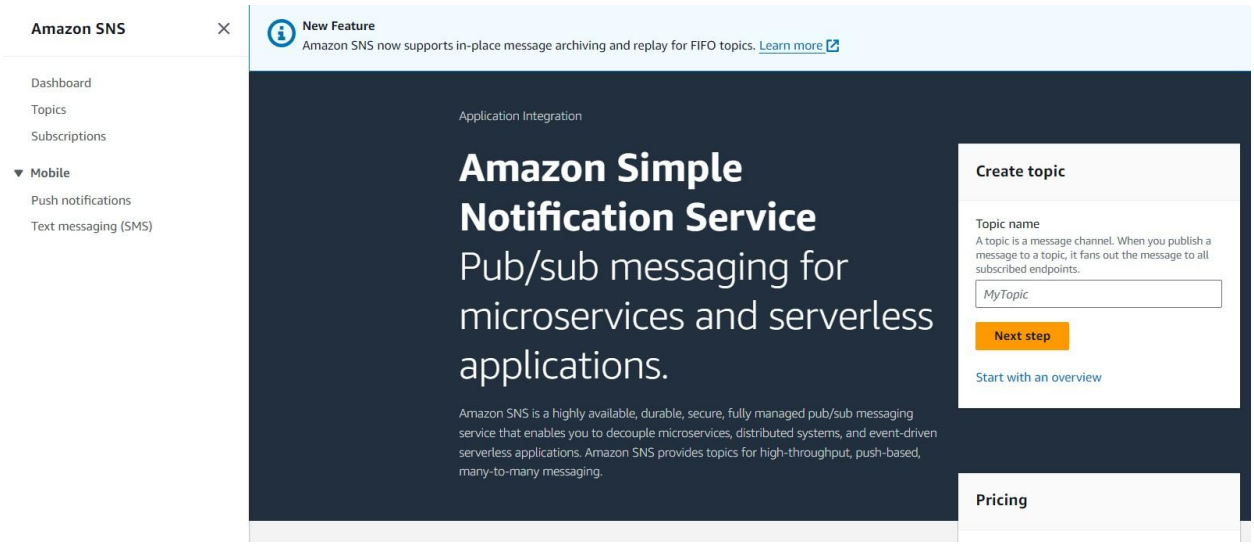
After creating all the tables as mentioned above the tables must be displayed like this.



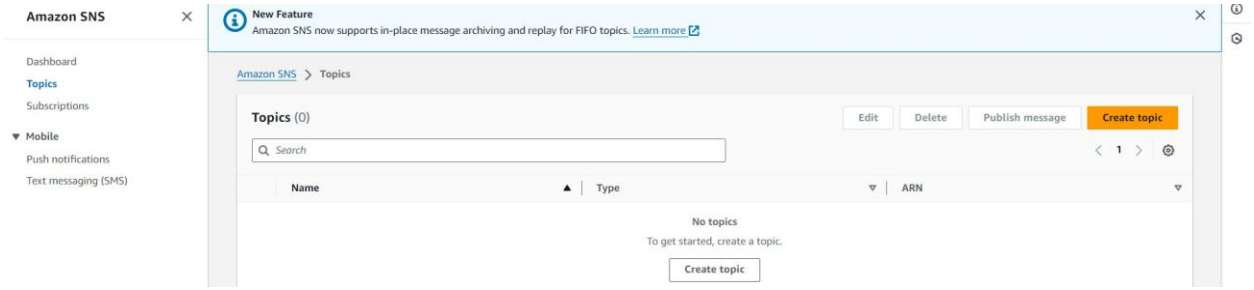
Milestone 4: SNS Notification Setup

- **Activity 4.1: Create SNS topics for sending notifications When a book request is made to the subscribed emails.**
  - In the AWS Console, search for SNS and navigate to the SNS Dashboard.

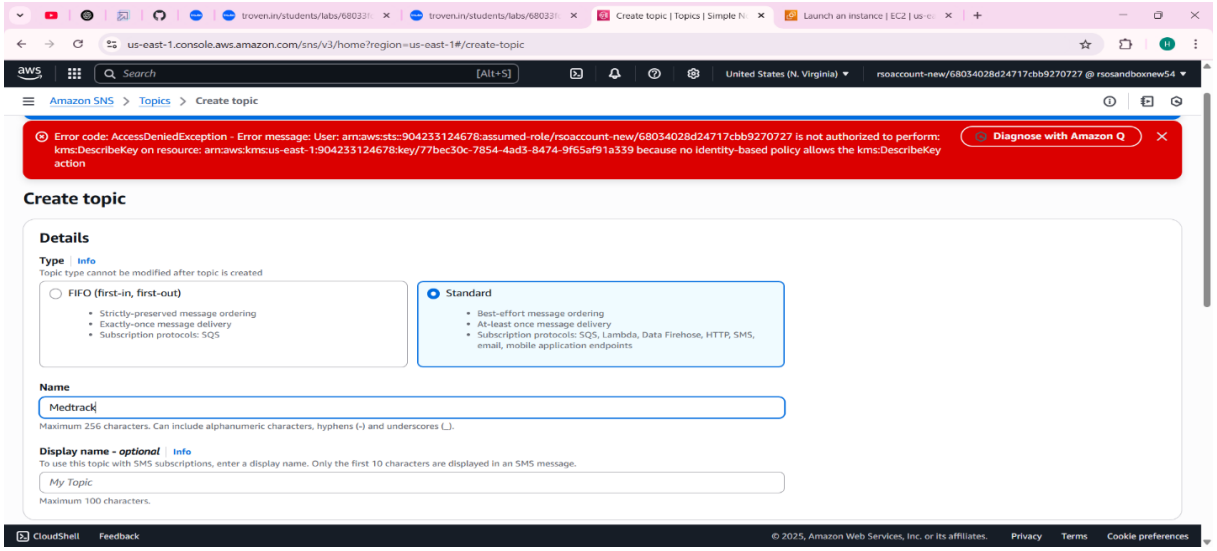


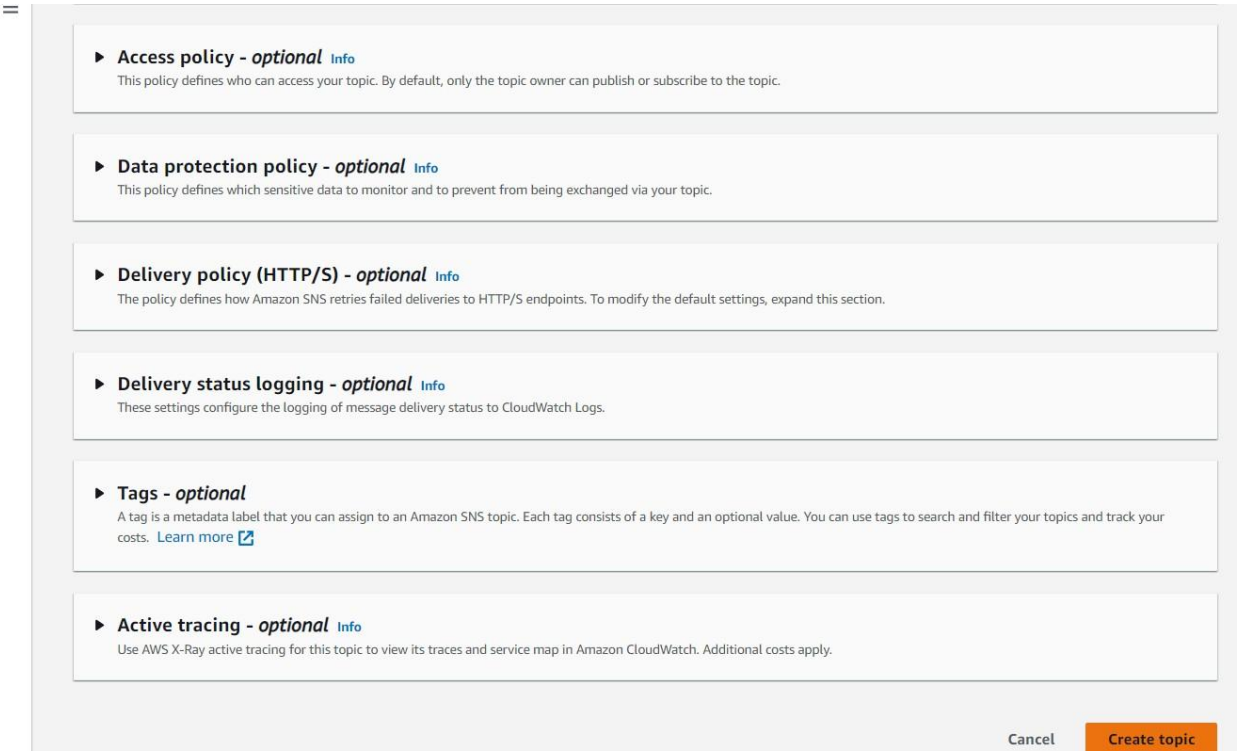


- Click on **Create Topic** and choose a name for the topic.

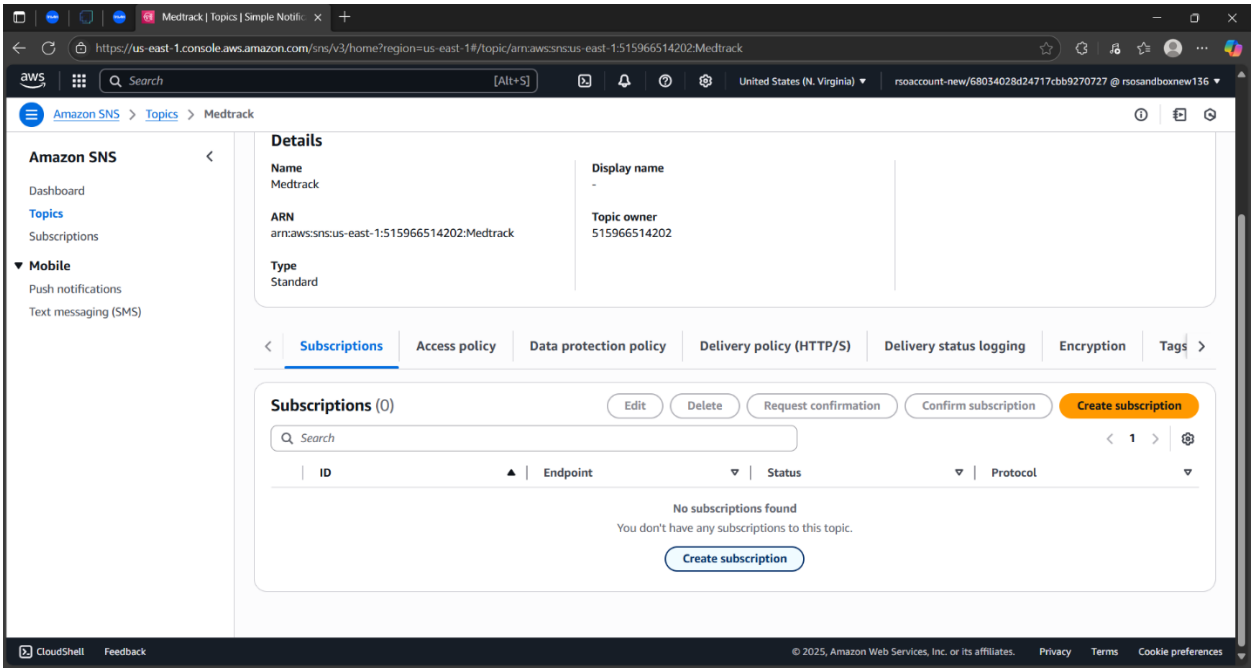


- Choose Standard type for general notification use cases and Click on Create Topic.

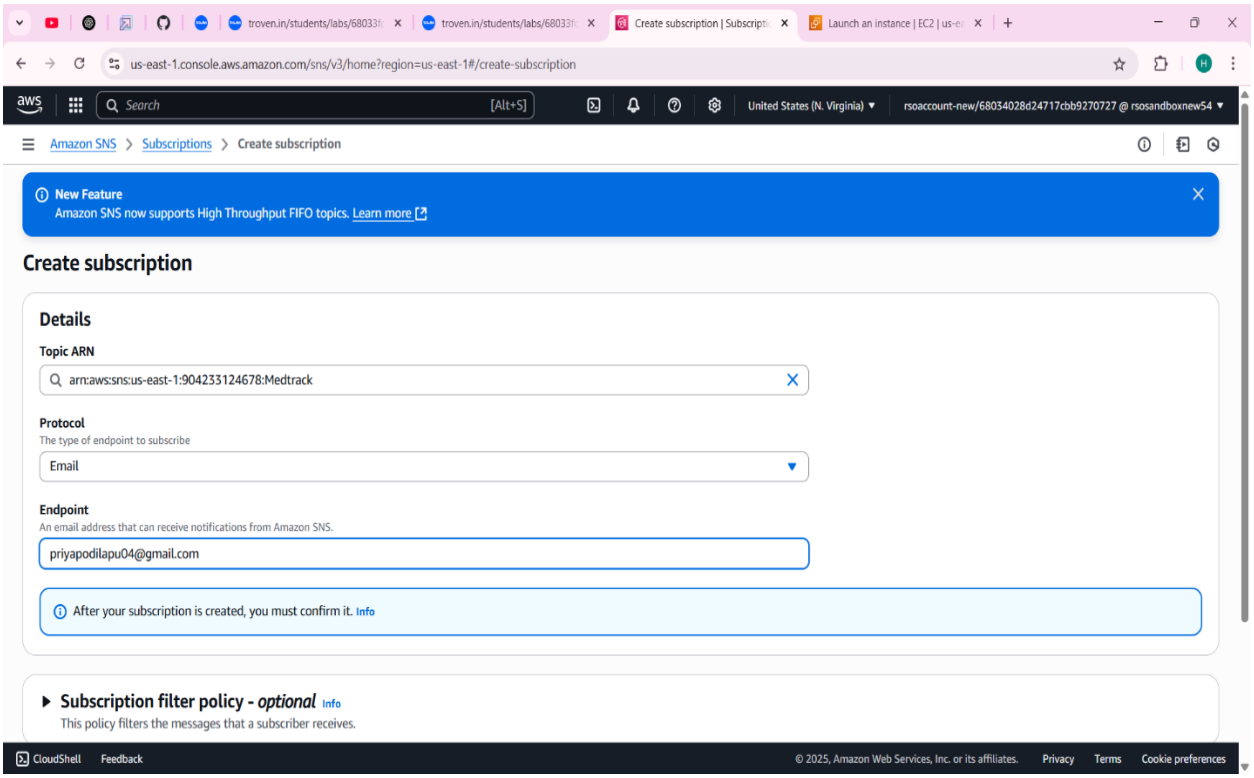




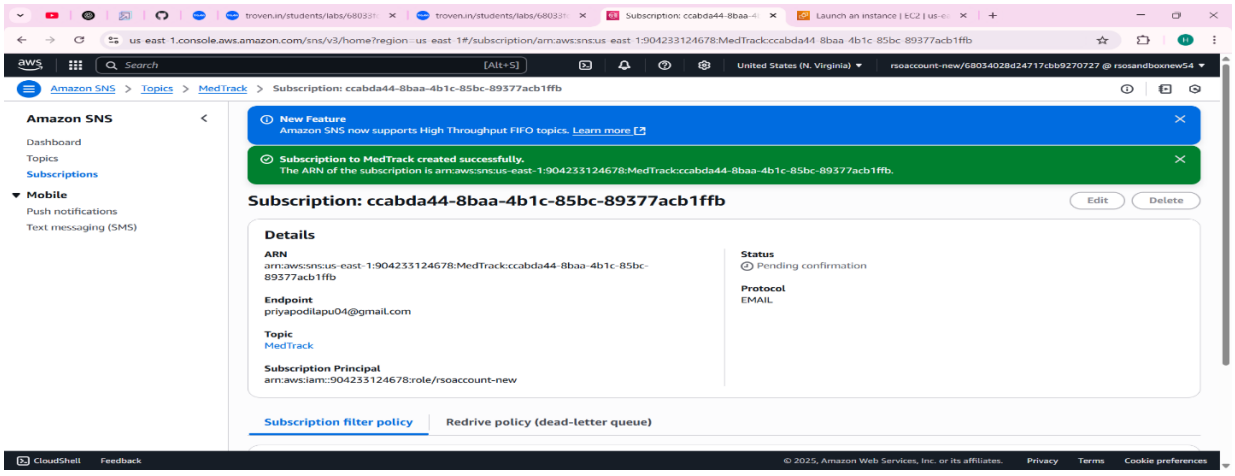
- Configure the SNS topic and note down the **Topic ARN**.



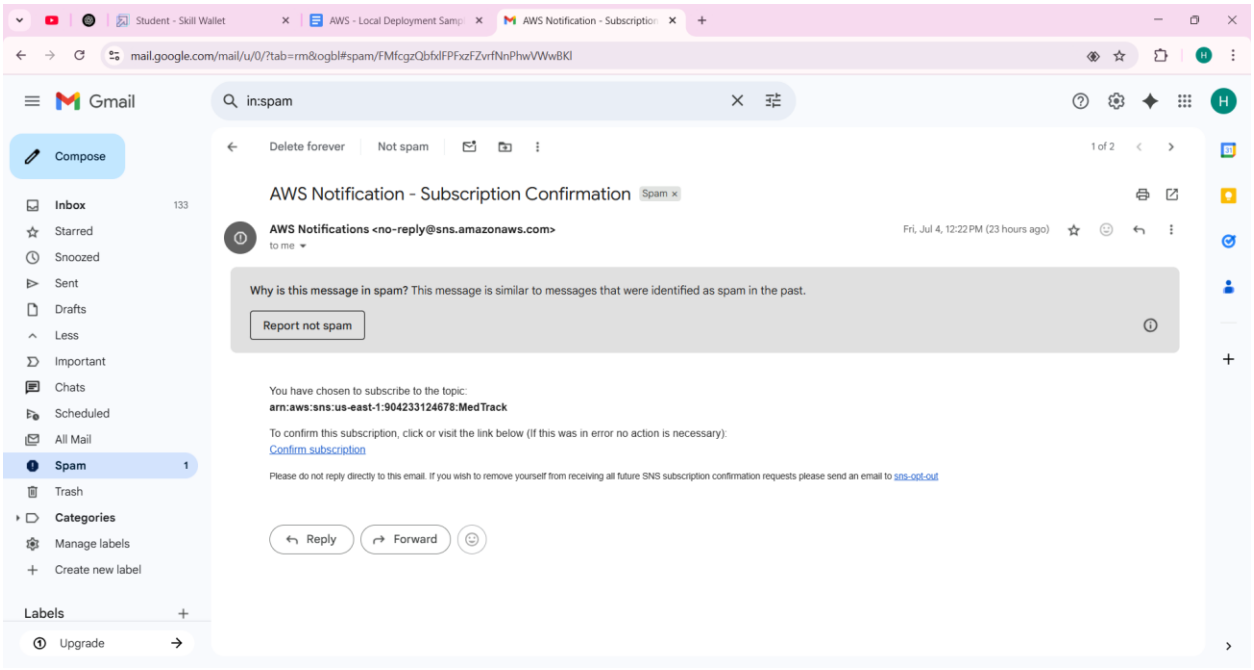
- **Activity 4.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.**
    - Subscribe users (or admin staff) to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails**.**

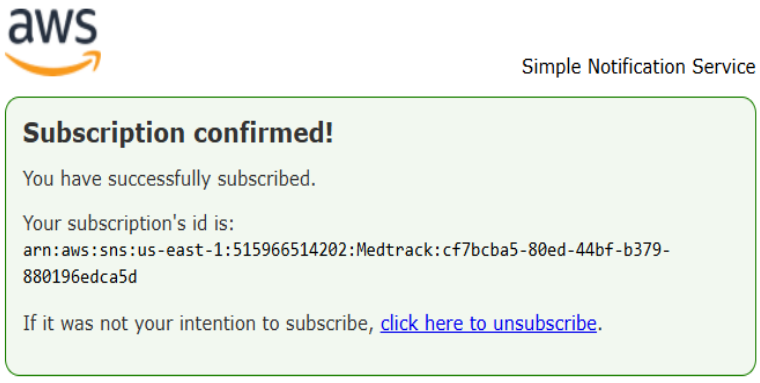


○ After subscription request for the mail confirmation

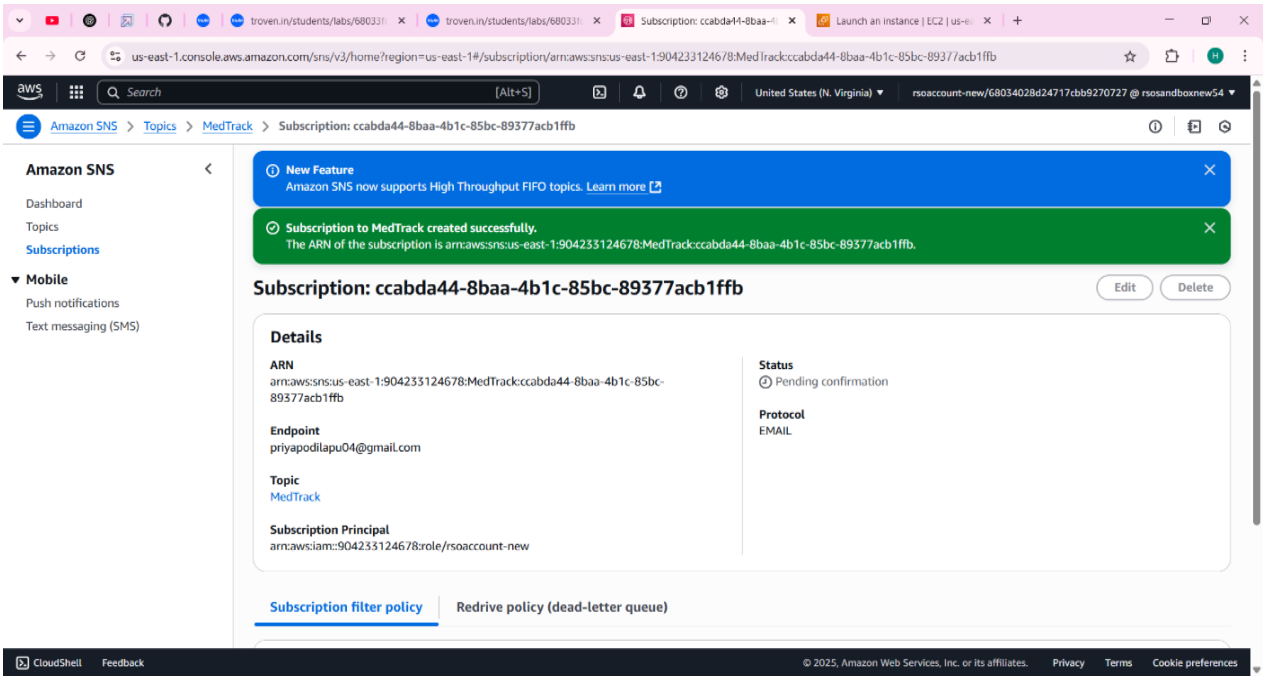


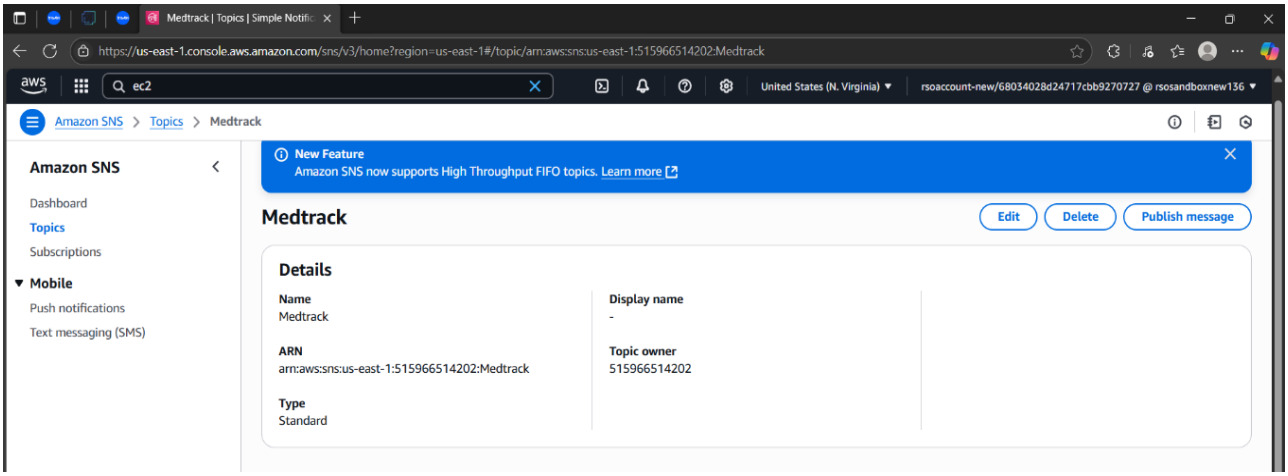
- Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.





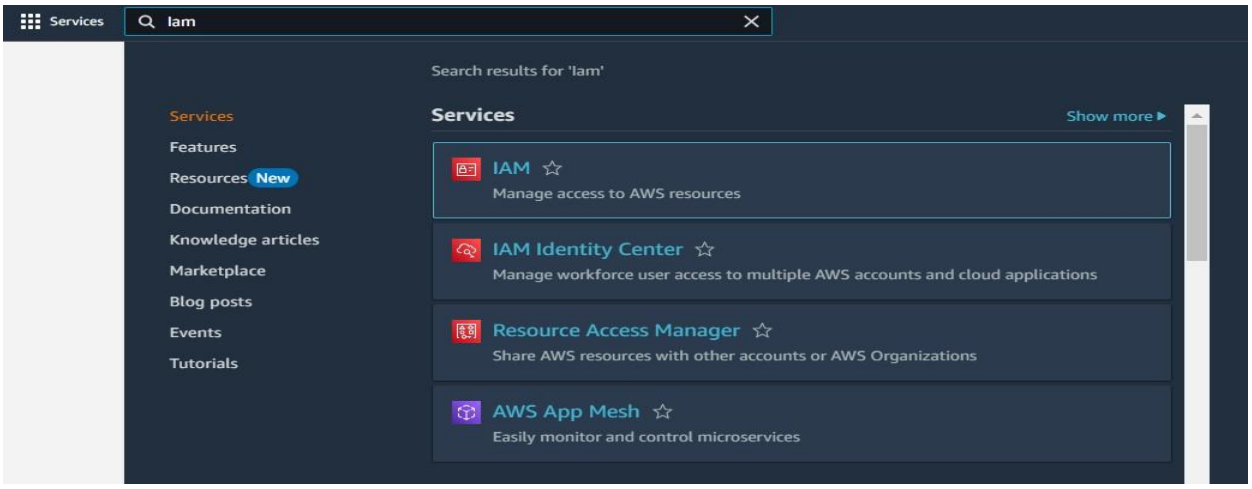
- Successfully done with the SNS mail subscription and setup, now store the ARN link.

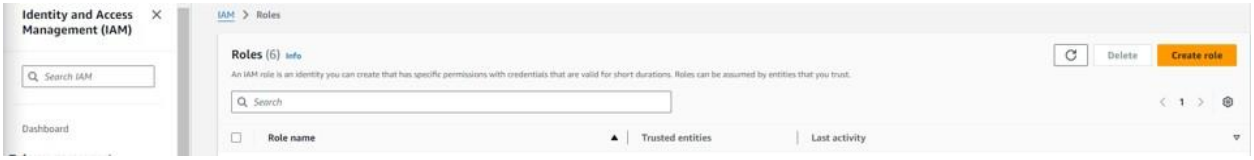


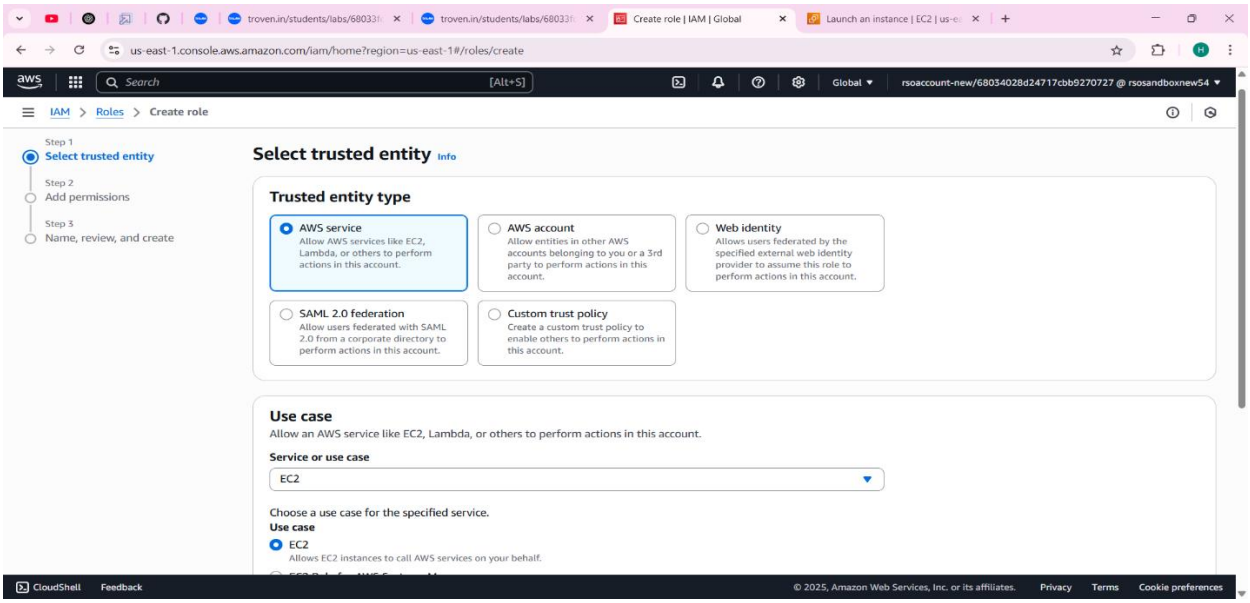


Milestone 5: IAM Role Setup

- **Activity 5.1:Create IAM Role.**
    - In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with Dynamo db and SNS.



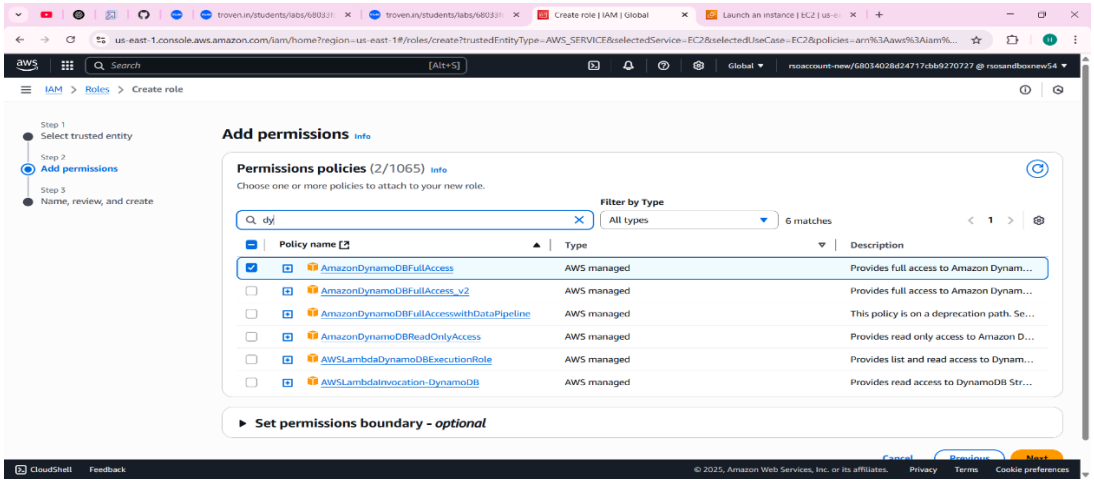


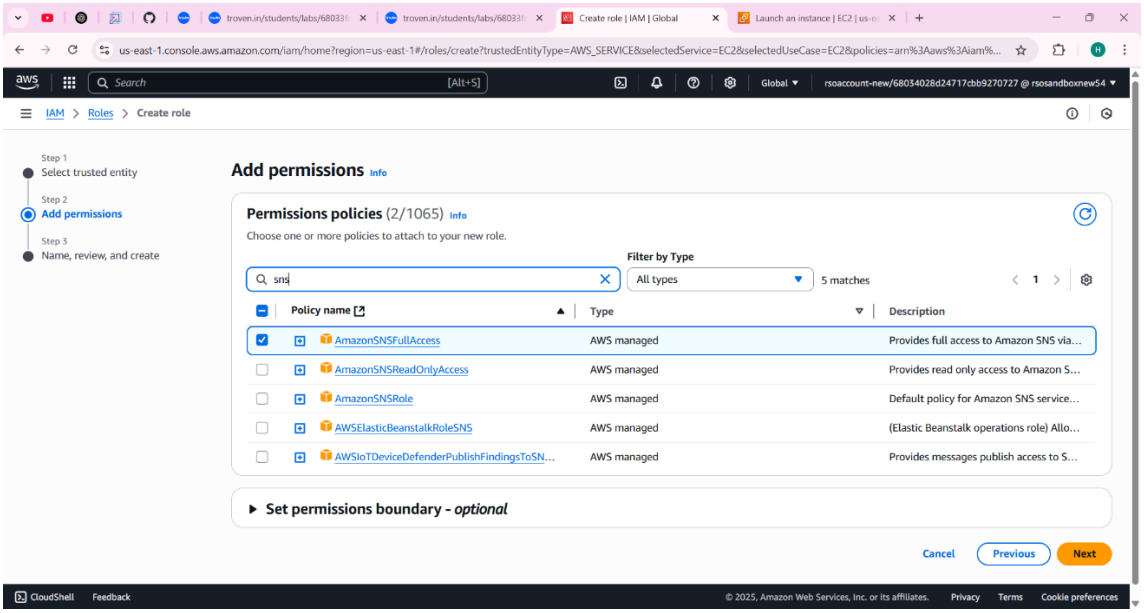


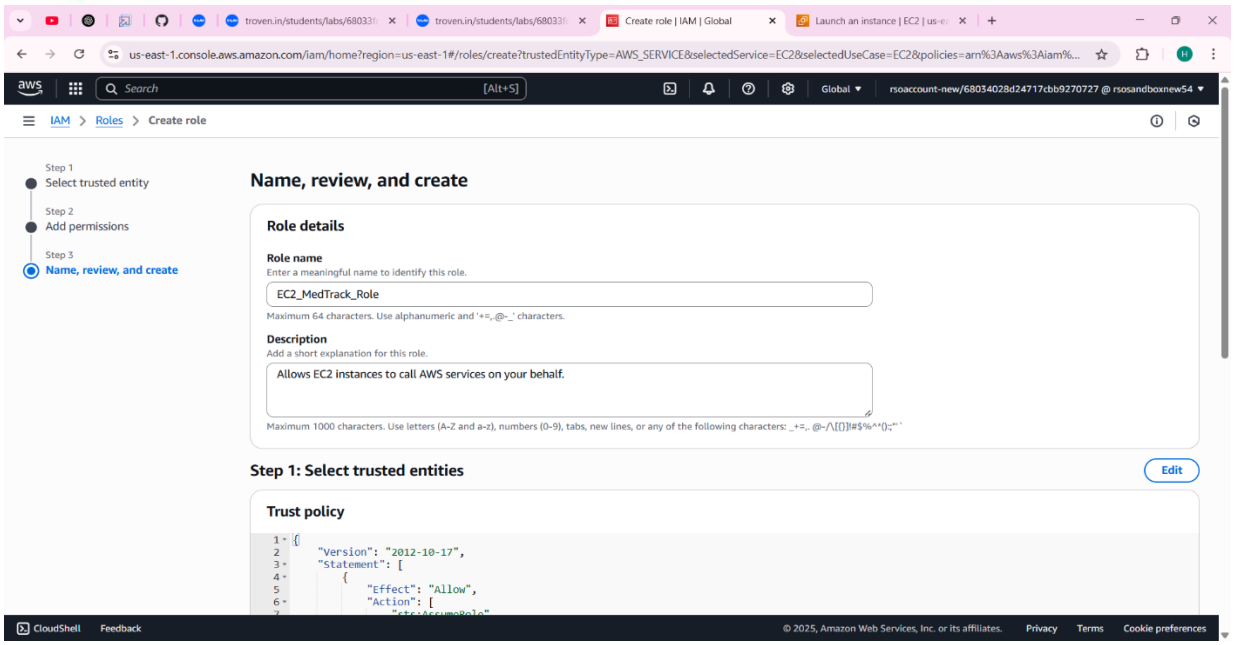
- **Activity 5.2: Attach Policies.**

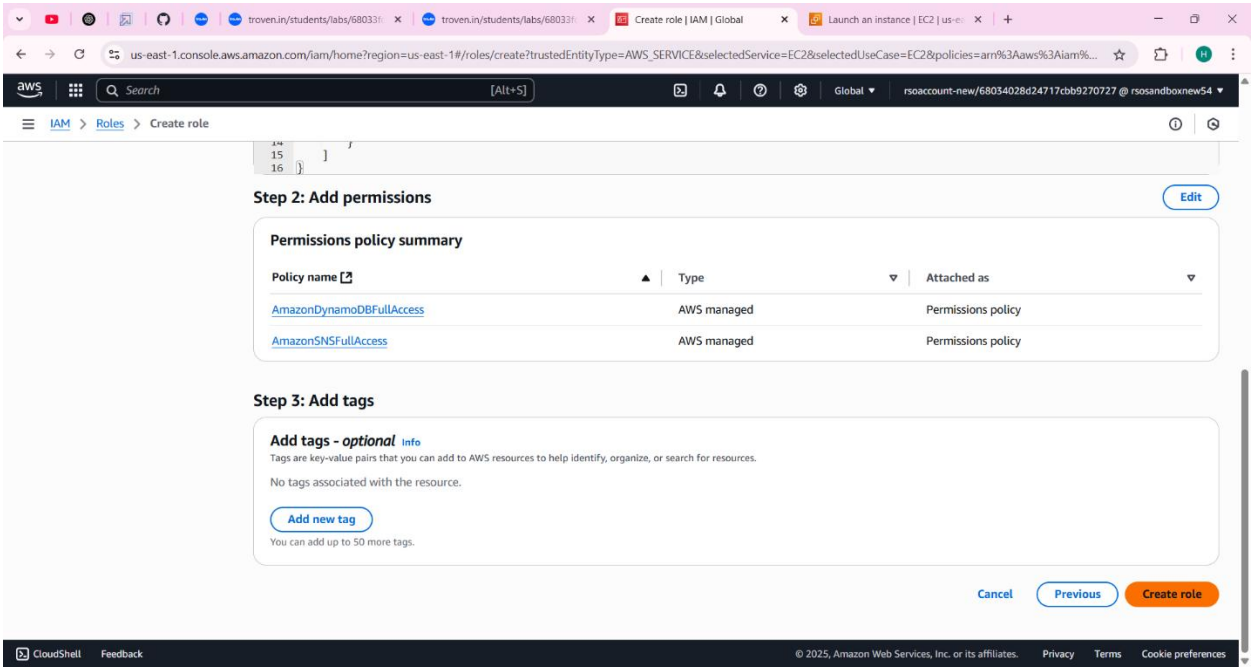
  Attach the following policies to the role:

  - AmazonDynamoDBFullAccess: Allows EC2 to perform read/write operations on DynamoDB.
  - AmazonSNSFullAccess: Grants EC2 the ability to send notifications via SNS.



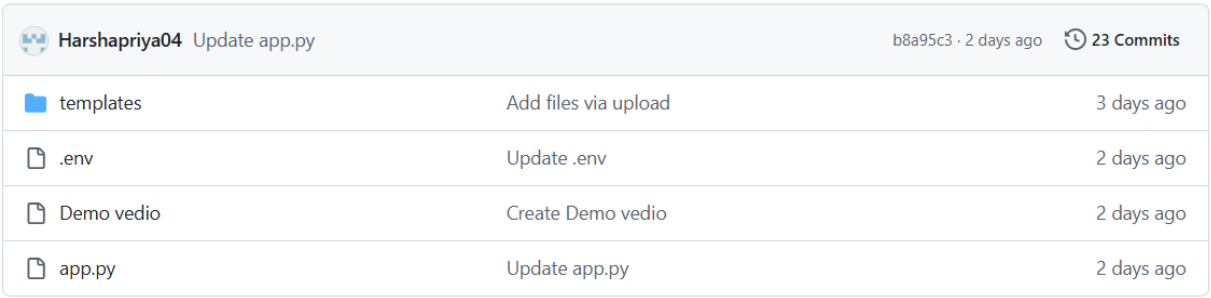


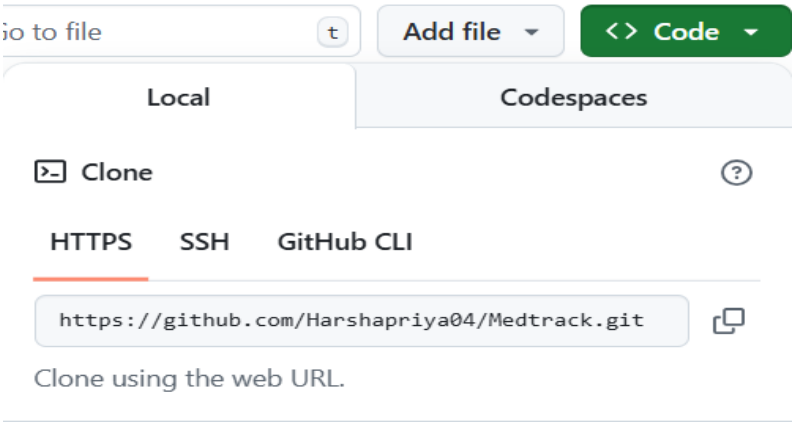

Create role | IAM | Global
us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles/create?trustedEntityType=AWS_SERVICE&selectedService=EC2&selectedUseCase=EC2&policies=arn%3Aaws%3Aiam%...
Search [Alt+S]
Global
rsoaccount-new/68034028d24717cbb9270727 @ rsosandboxnew54
IAM > Roles > Create role
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.
EC2_MedTrack_Role
Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.
Description
Add a short explanation for this role.
Allows EC2 instances to call AWS services on your behalf.
Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _+=,. @-/\[{}]!#$%^*();:"'
Step 1: Select trusted entities
Edit
Trust policy
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": [
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Create role | IAM | Global
us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles/create?trustedEntityType=AWS_SERVICE&selectedService=EC2&selectedUseCase=EC2&policies=arn%3Aaws%3Aiam%...
Search [Alt+S]
Global
rsoaccount-new/68034028d24717cbb9270727 @ rsosandboxnew54
IAM > Roles > Create role
]
}
Step 2: Add permissions
Edit
Permissions policy summary
Policy name
Type
Attached as
AmazonDynamoDBFullAccess
AWS managed
Permissions policy
AmazonSNSFullAccess
AWS managed
Permissions policy
Step 3: Add tags
Add tags - optional Info
Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.
No tags associated with the resource.
Add new tag
You can add up to 50 more tags.
Cancel
Previous
Create role
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Milestone 6: EC2 Instance Setup

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

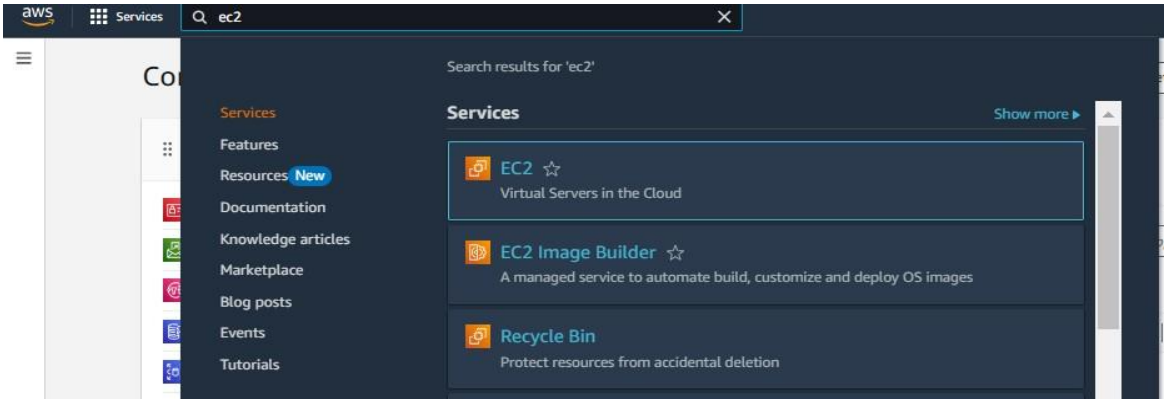




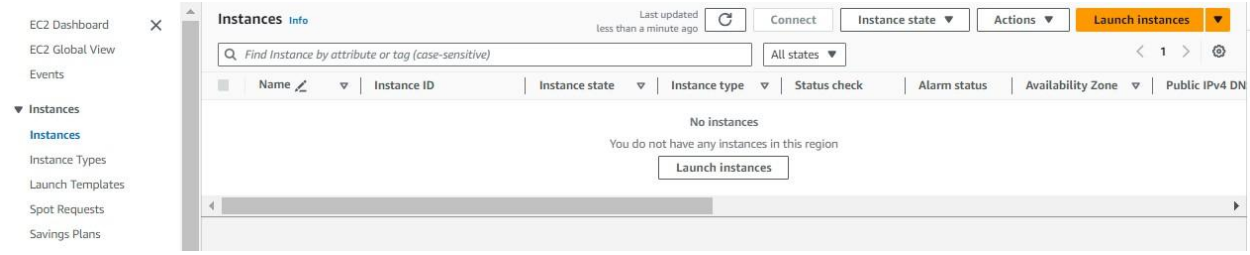
- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

  Launch EC2 Instance

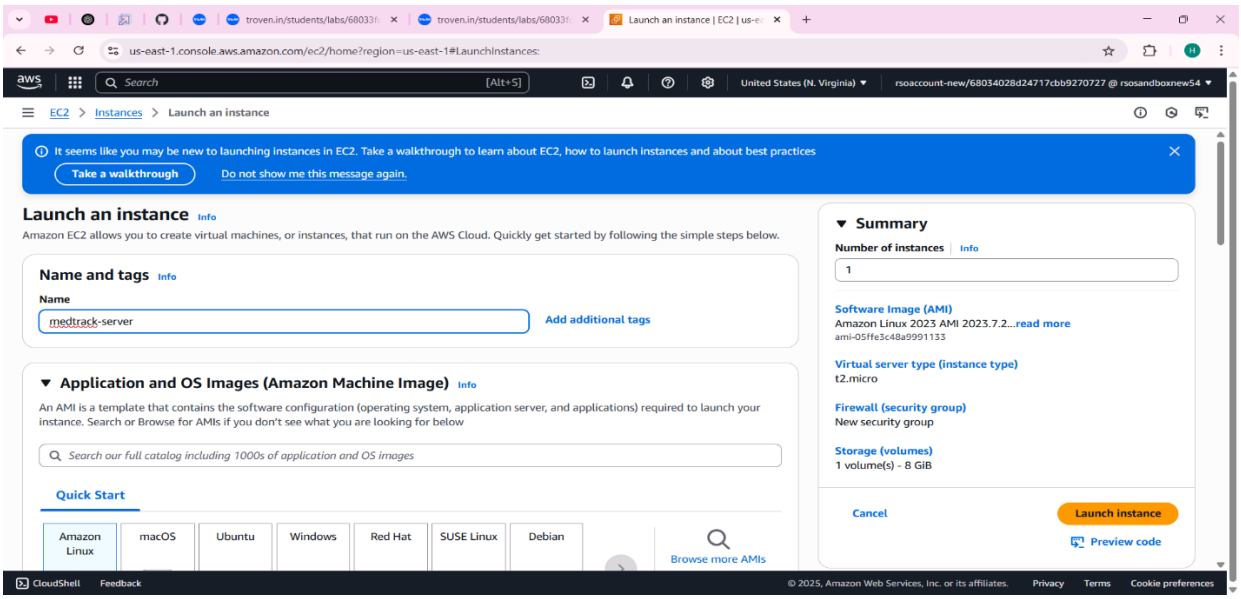
  - In the AWS Console, navigate to EC2 and launch a new instance.



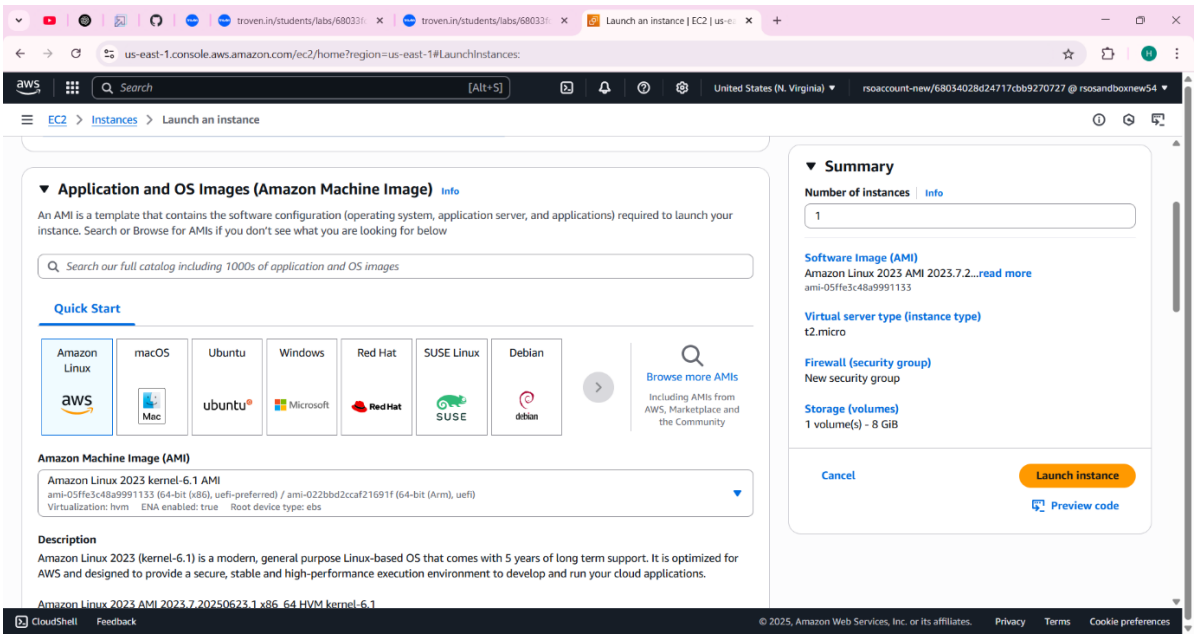
  - Click on Launch instance to launch EC2 instance



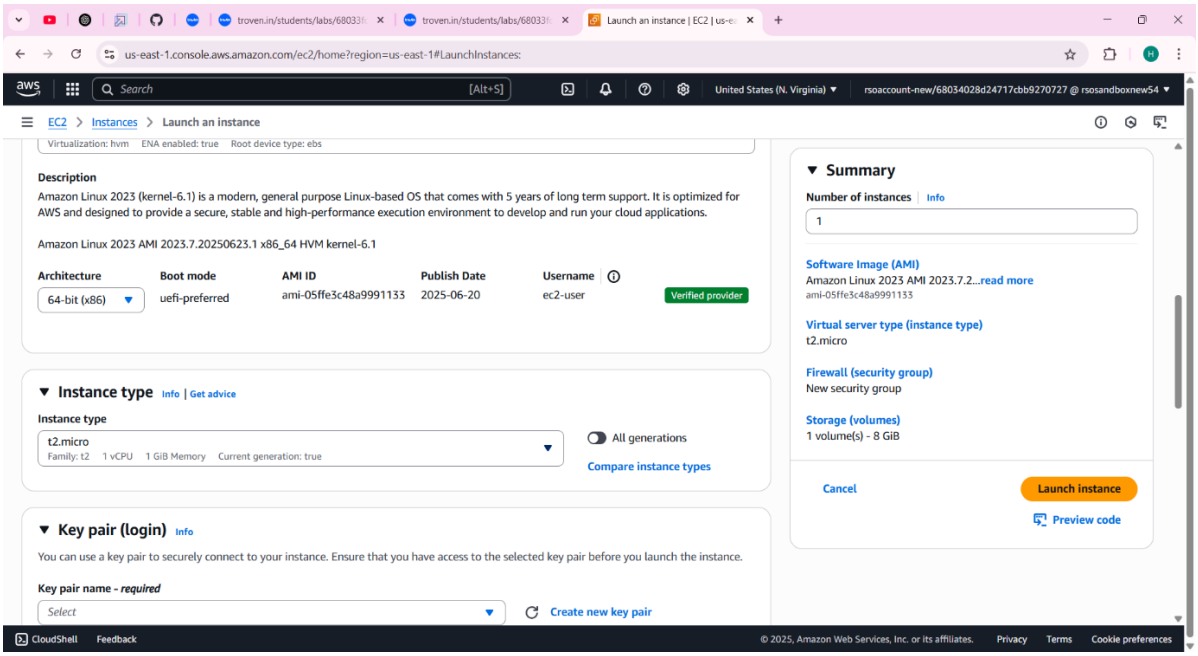
  - Create an instance with the name medtrack-server.



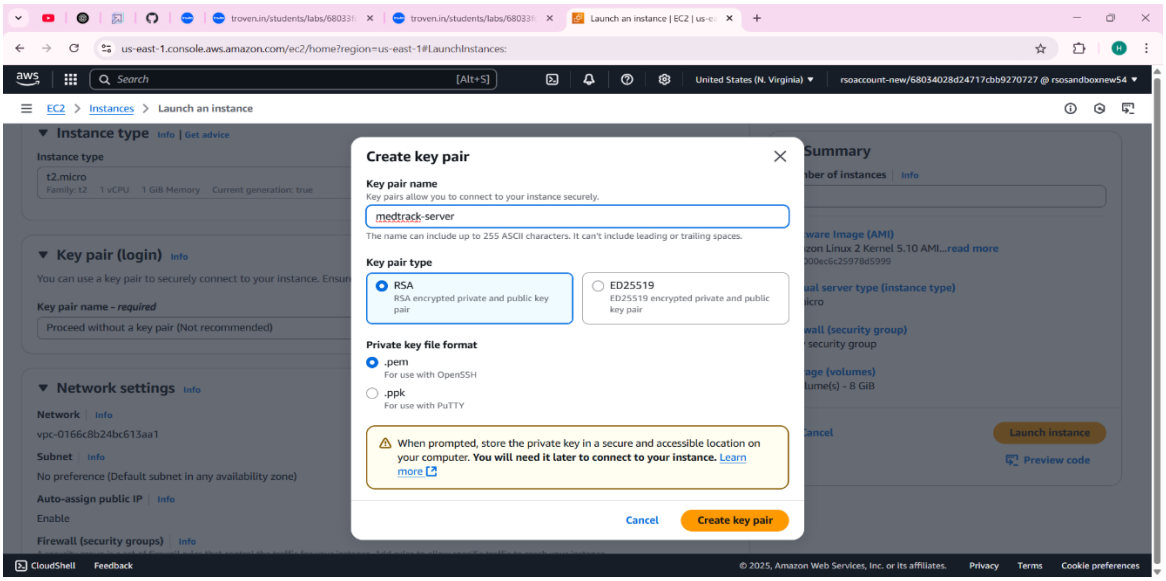
- Choose Amazon Linux 2 and choose Amazon Linux 2023 as the AMI and t2.micro as the instance type (free-tier eligible).

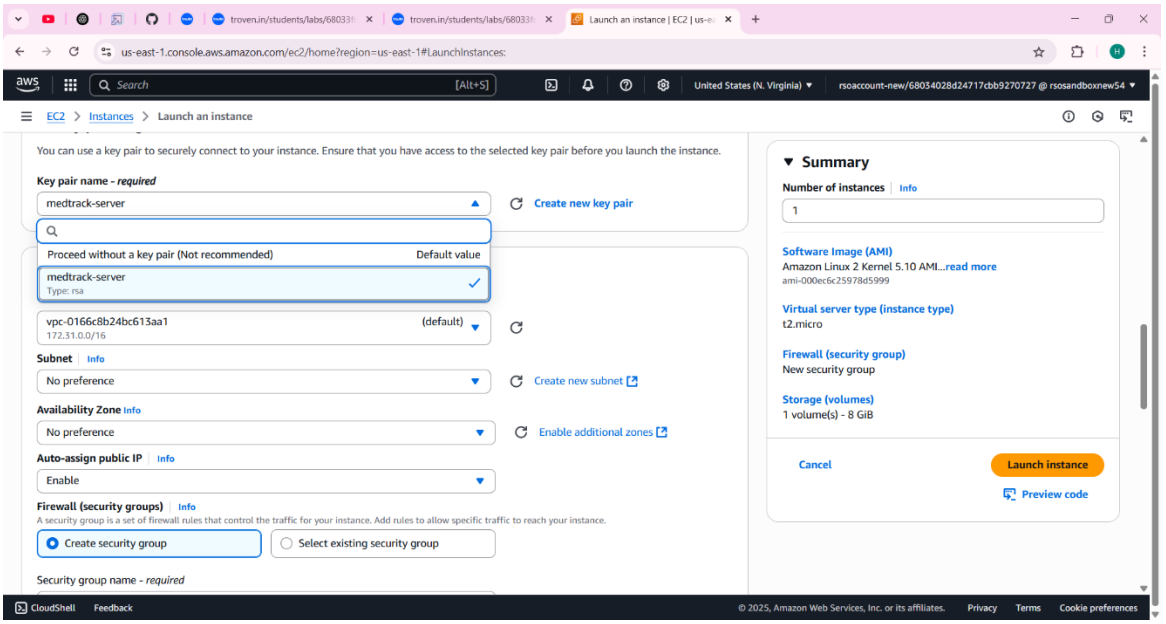


- Create and download the key pair for Server access.



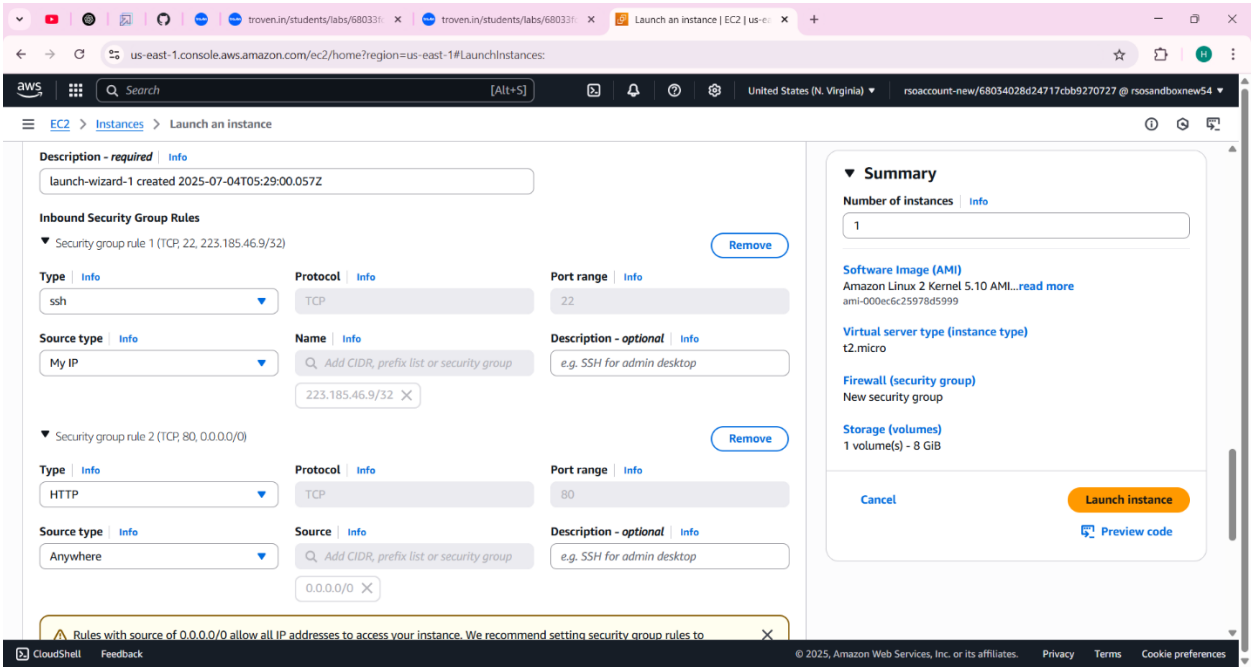
- Create a Key pair as RSA and only and select that Key pair.

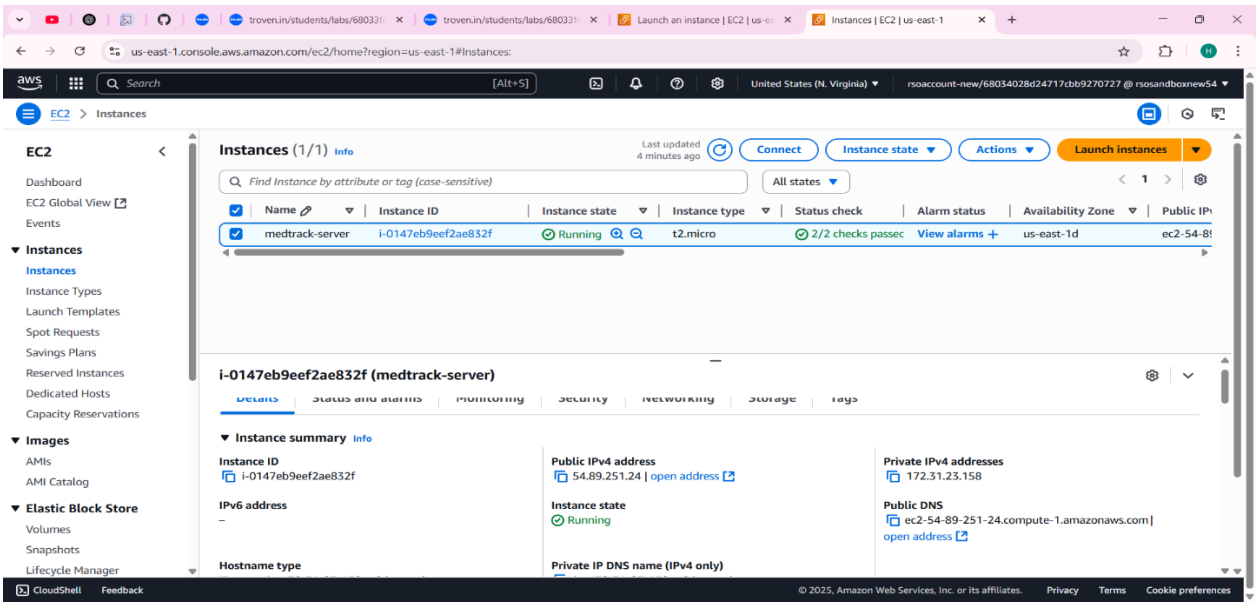




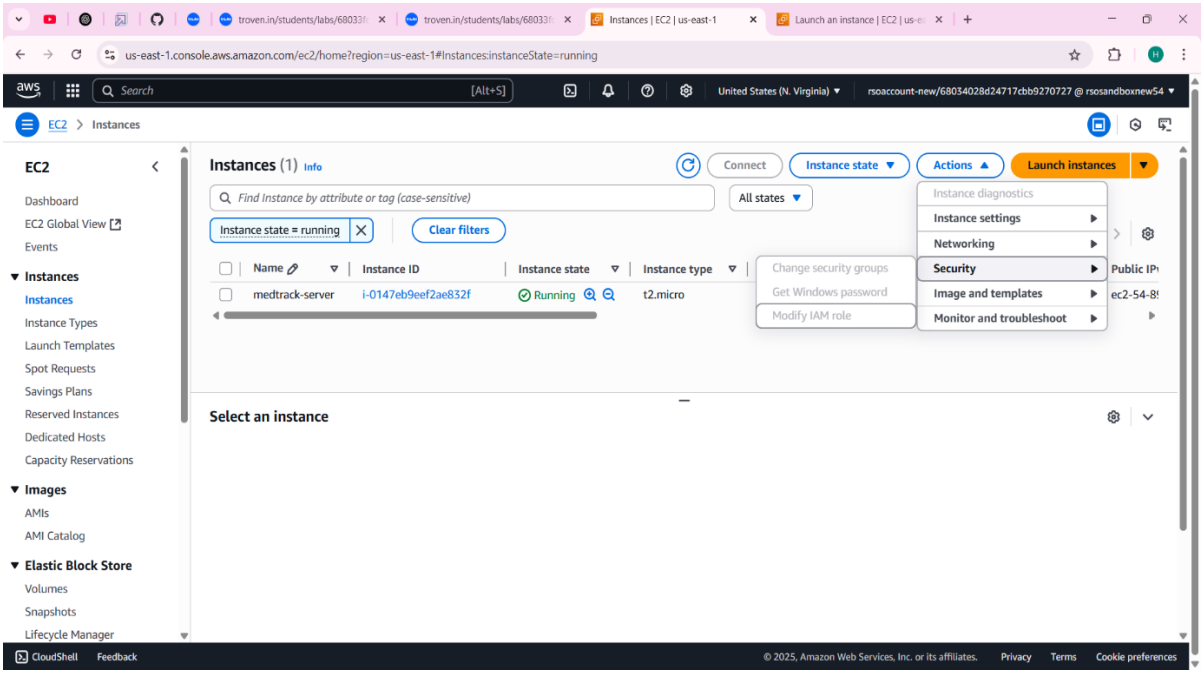
- **Activity 6.2:Configure security groups for HTTP, and SSH access.**

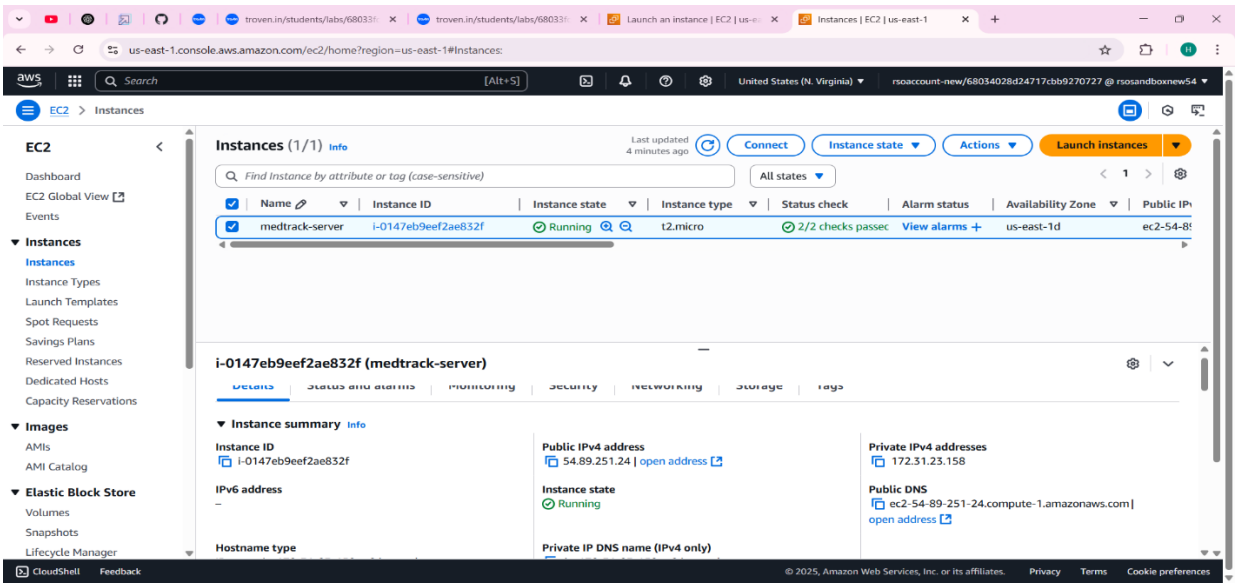
In network settings configure security groups for http – port range 80(type Anywhere) and ssh – port range 22 (type MY IP) and click on launch instance.



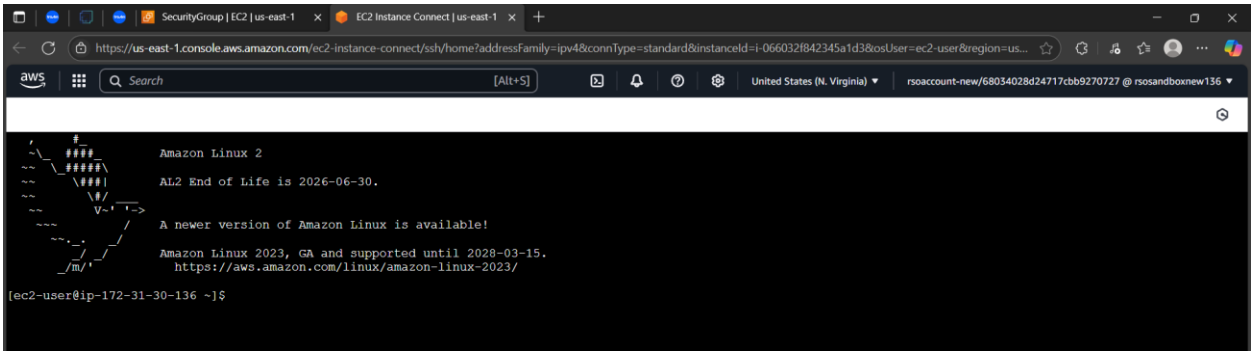


- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the 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 Linux2:

sudo yum update -y

sudo yum install python3 git

sudo pip3 install flask boto3

Verify Installations:

flask –version

git --version

Activity 7.2: Clone Your Flask Project from GitHub

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

Run: 'git clone https://github.com/your-github-username/your-repositname.git'

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

credentials

here: 'git clone https://github.com/Harshapriya04/Medtrack.git'

- This will download your project to the EC2 instance.

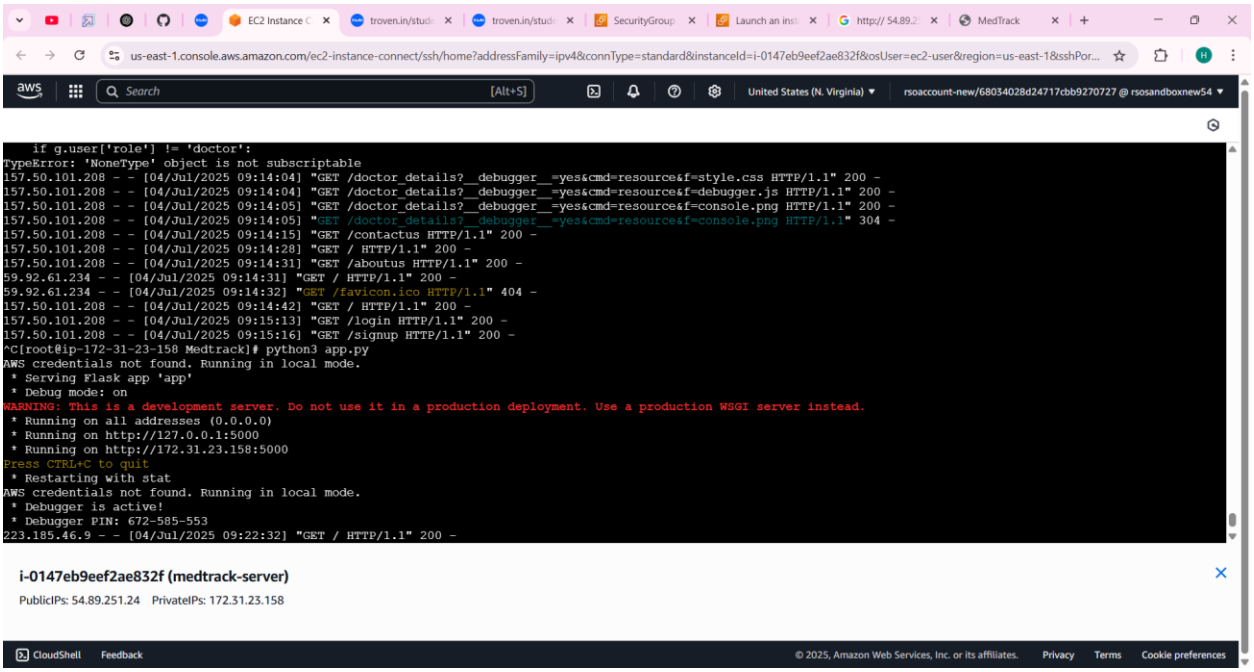
To navigate to the project directory, run the following command:

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

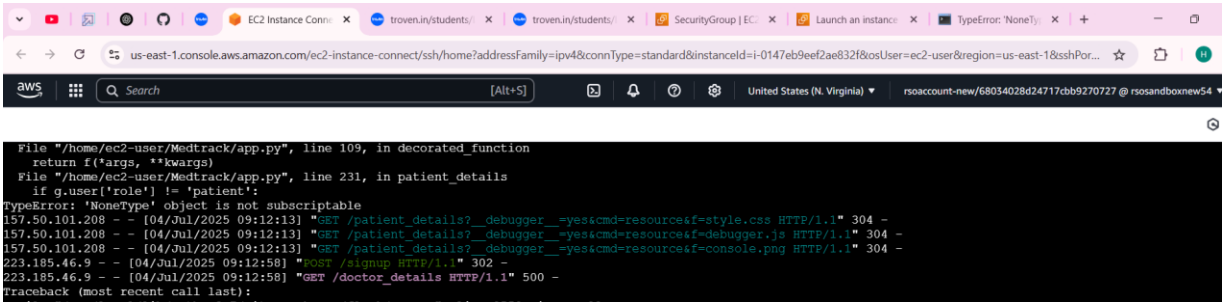
sudo flask run --host=0.0.0.0 --port=80



Verify the Flask app is running:

http://your-ec2-public-ip

- Run the Flask app on the EC2 instance



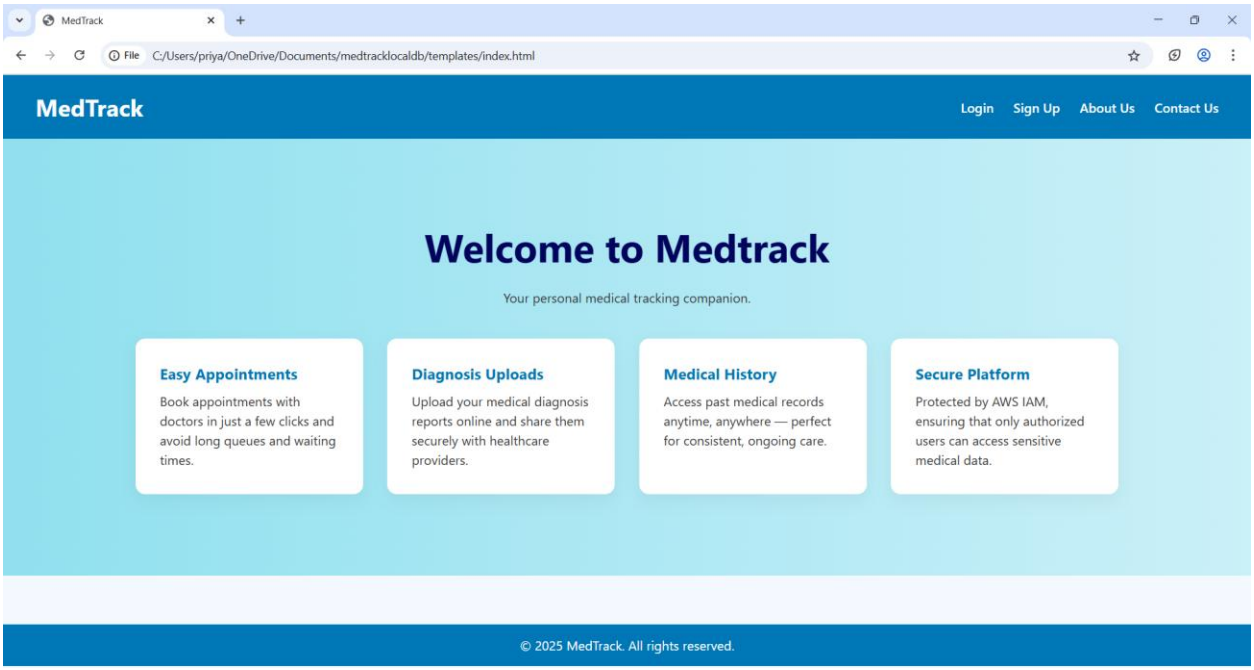
Access the website through:

PublicIPs: http://54.89.251.24:5000/

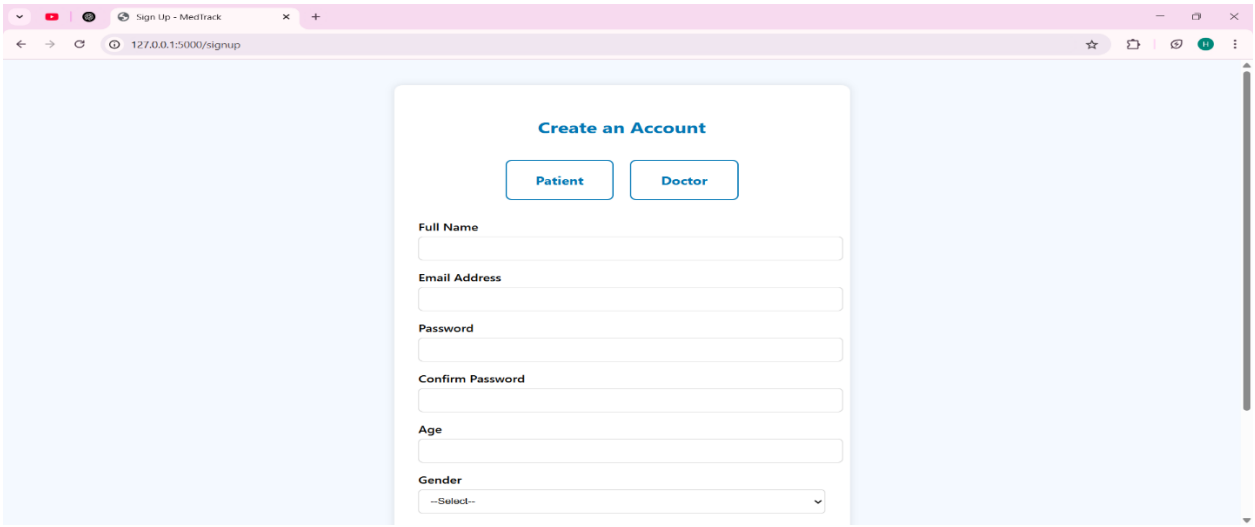
Milestone 8: Testing and Deployment

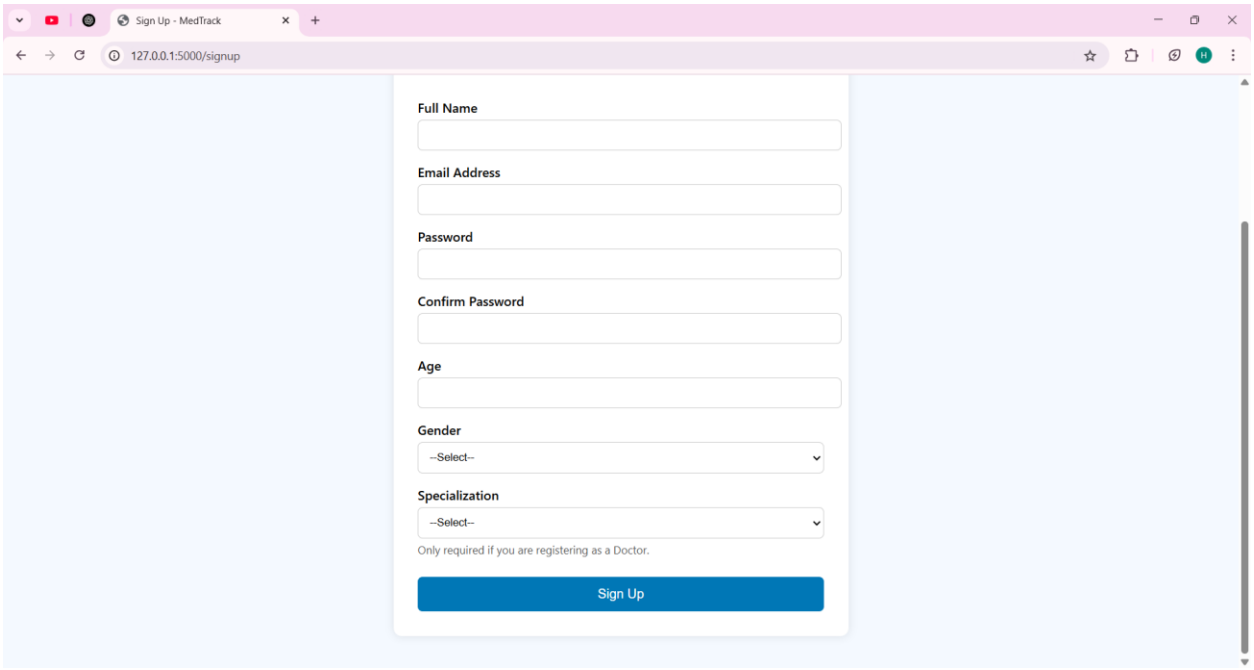
- **Activity 8.1: Conduct functional testing to verify user registration, login, book requests, and notifications.**

Index Page:

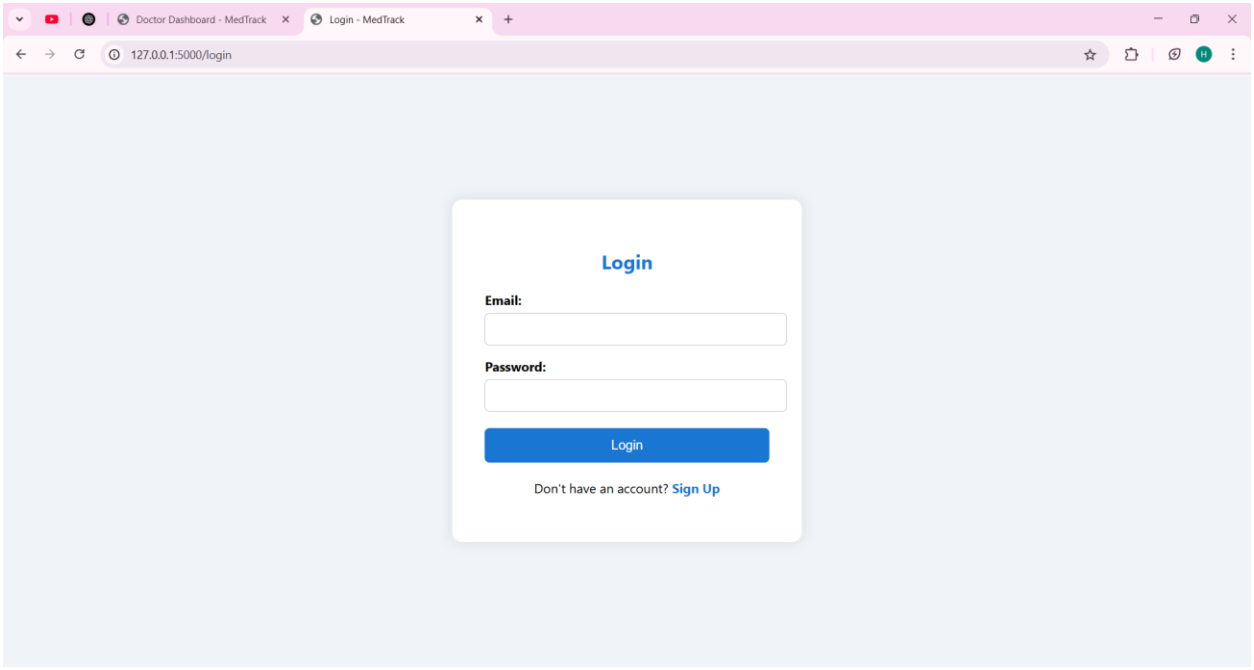


Signup Page:

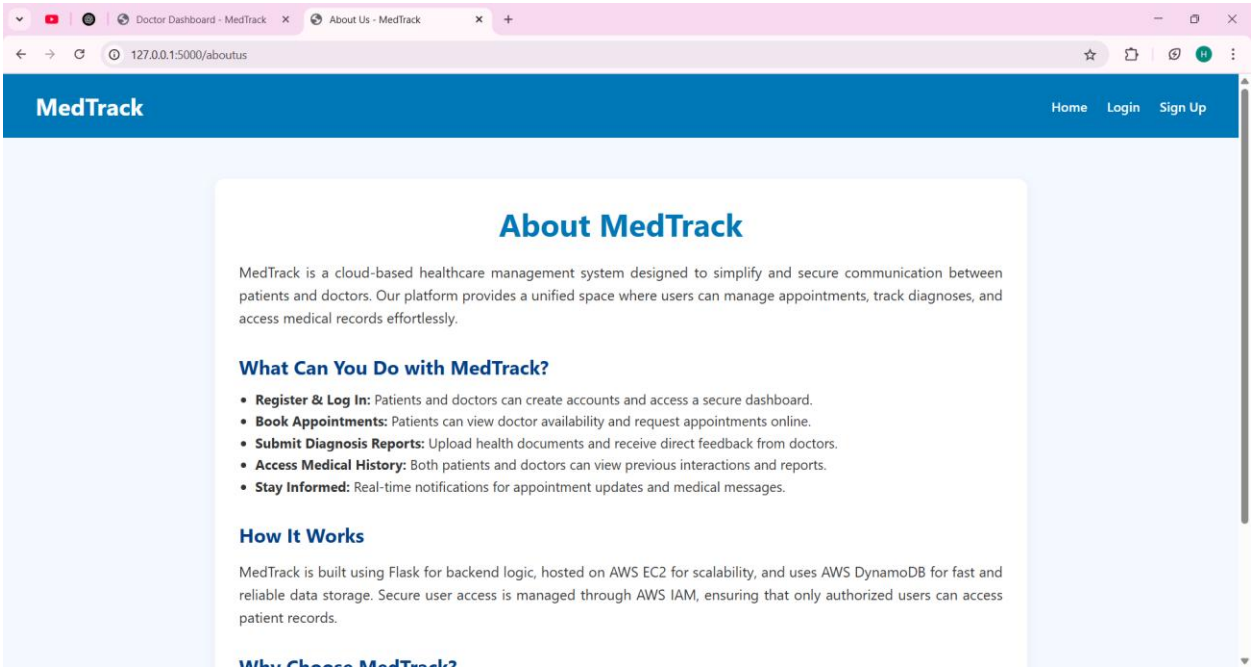


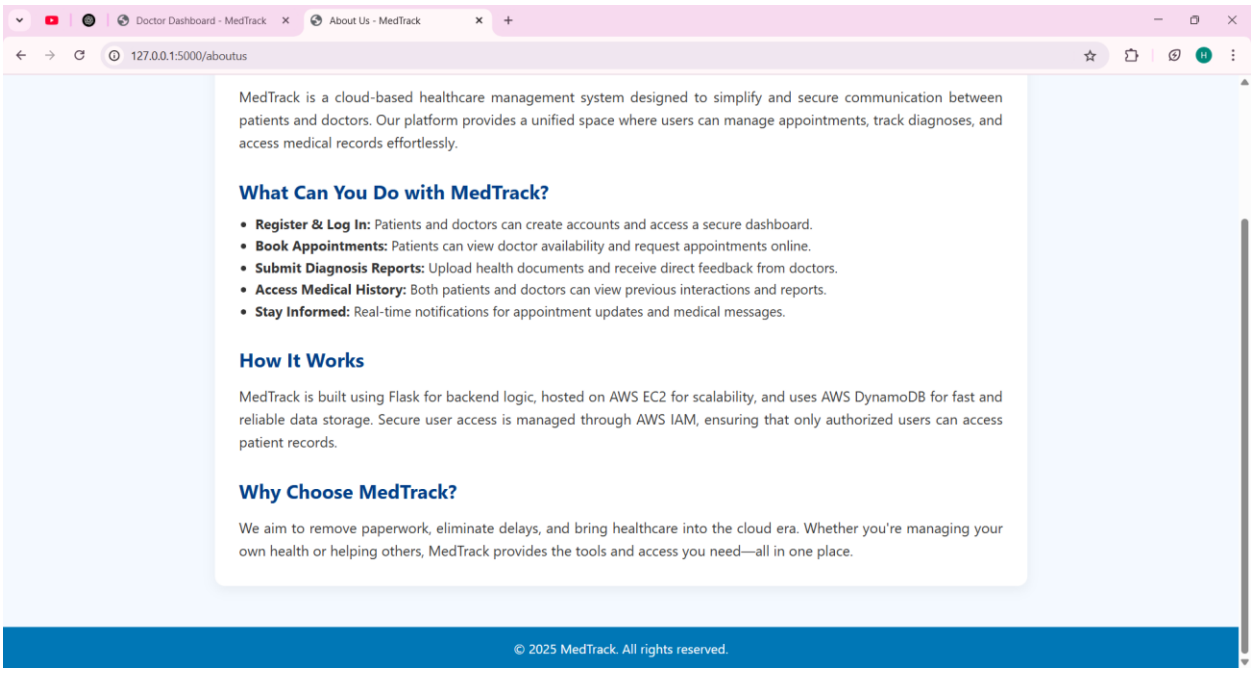

Sign Up - MedTrack
127.0.0.1:5000/signup
Full Name
Email Address
Password
Confirm Password
Age
Gender
--Select--
Specialization
--Select--
Only required if you are registering as a Doctor.
Sign Up

Login page:

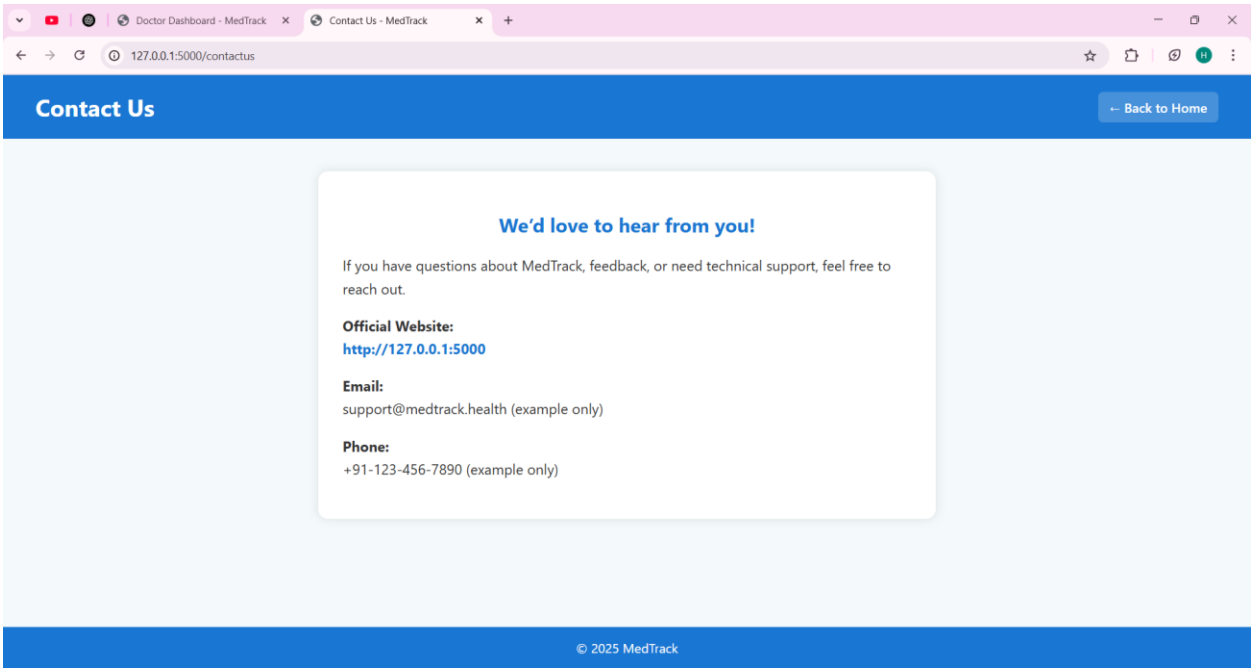

Doctor Dashboard - MedTrack
Login - MedTrack
127.0.0.1:5000/login
Login
Email:
Password:
Login
Don't have an account? Sign Up

About Us page:

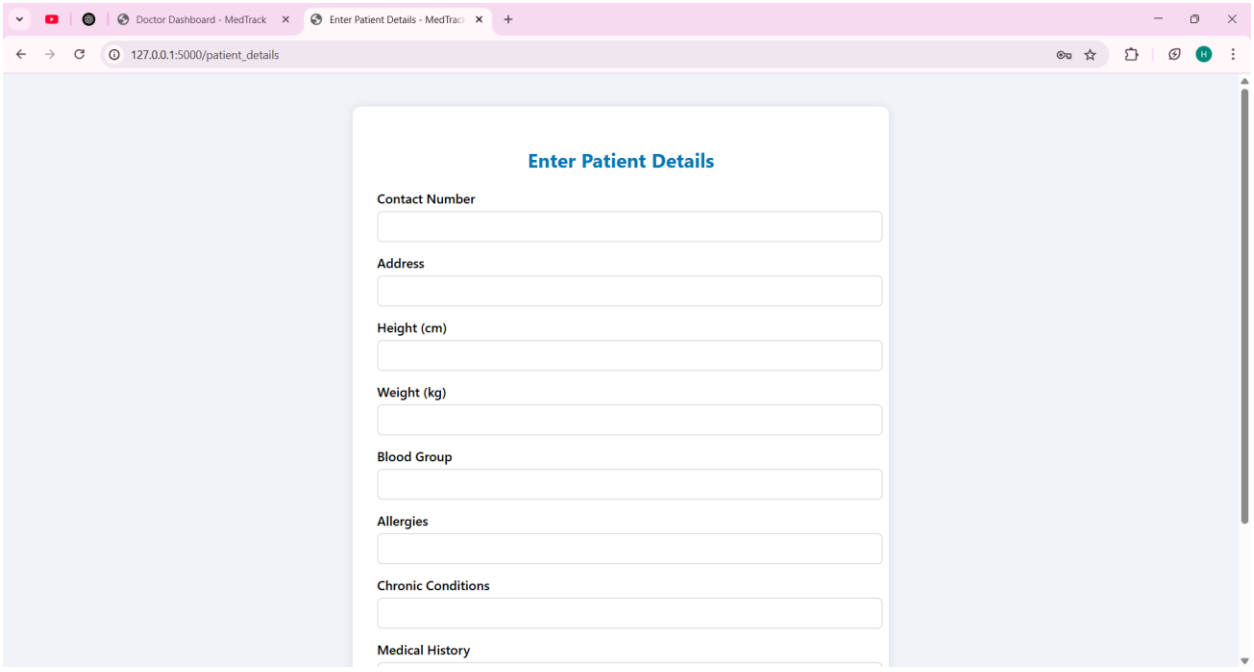


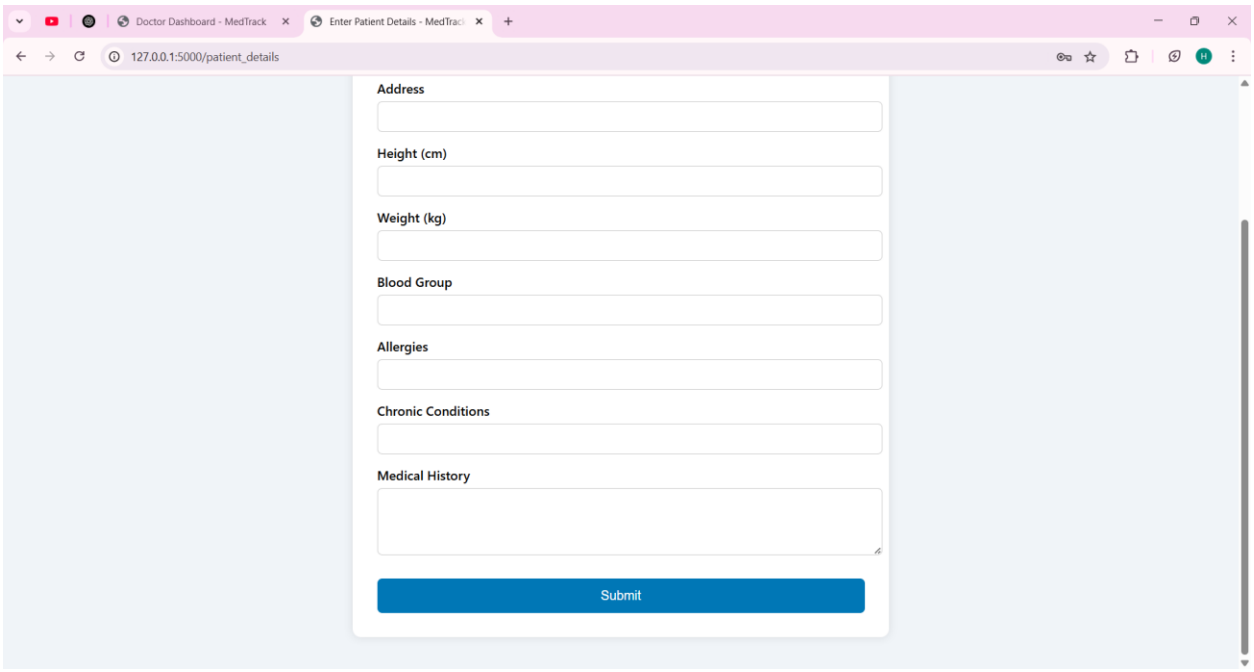


Contactus Page:

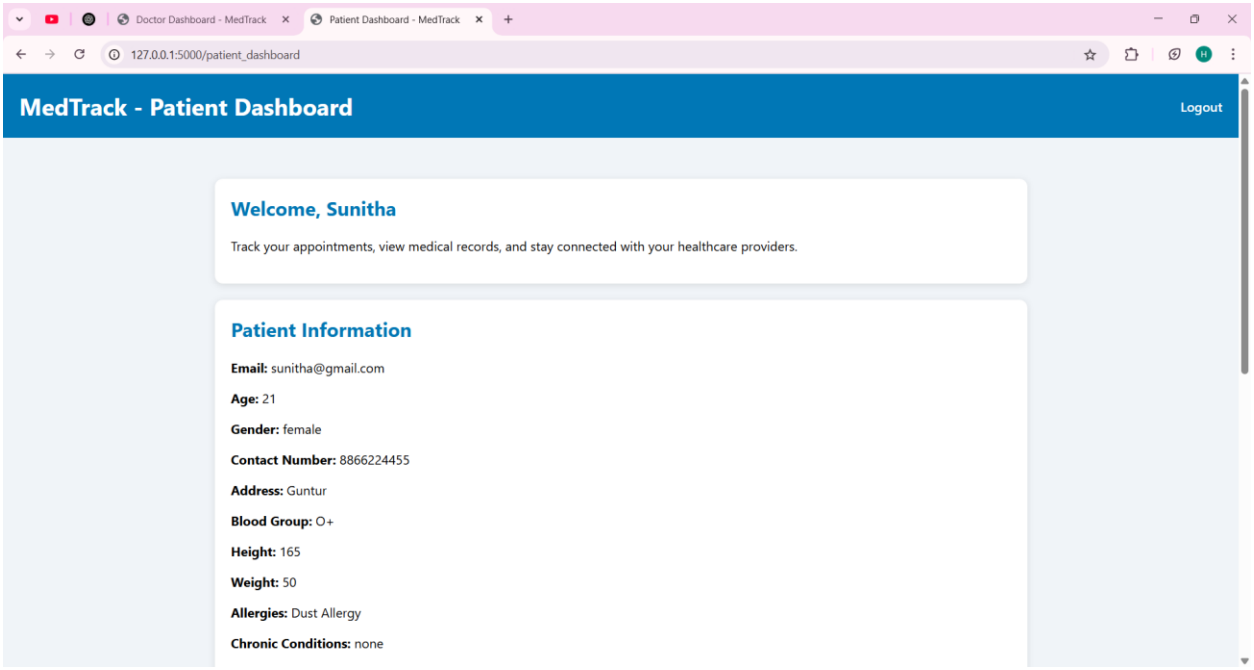


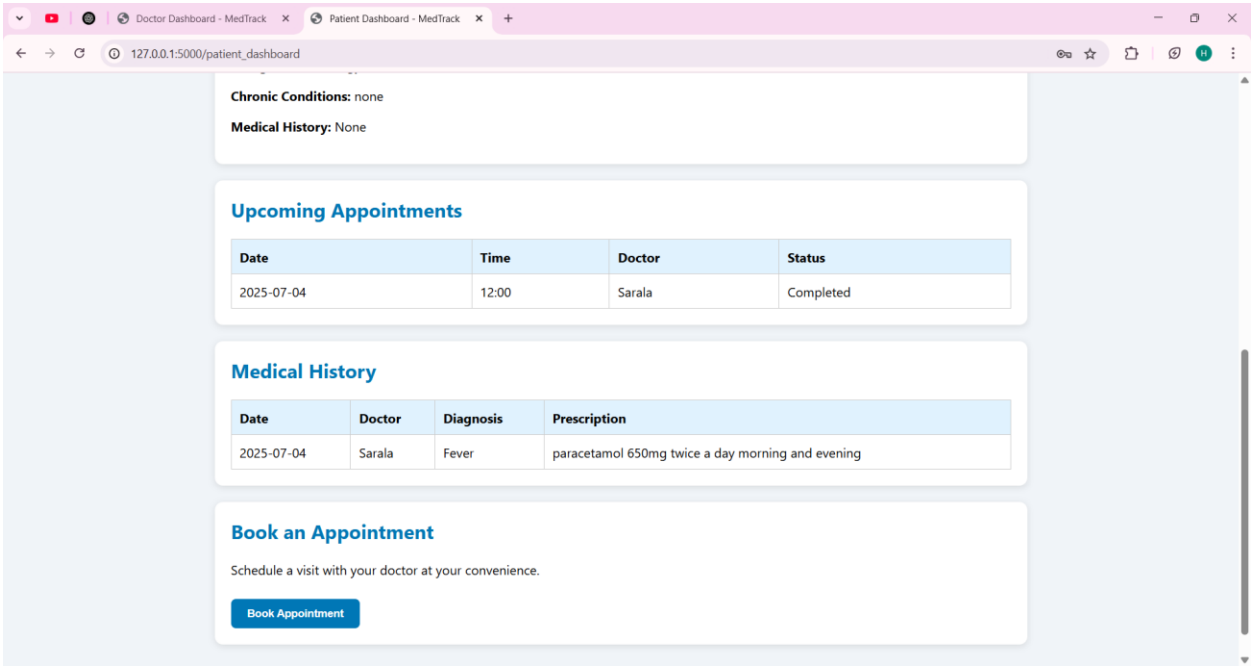
Patient Details page:



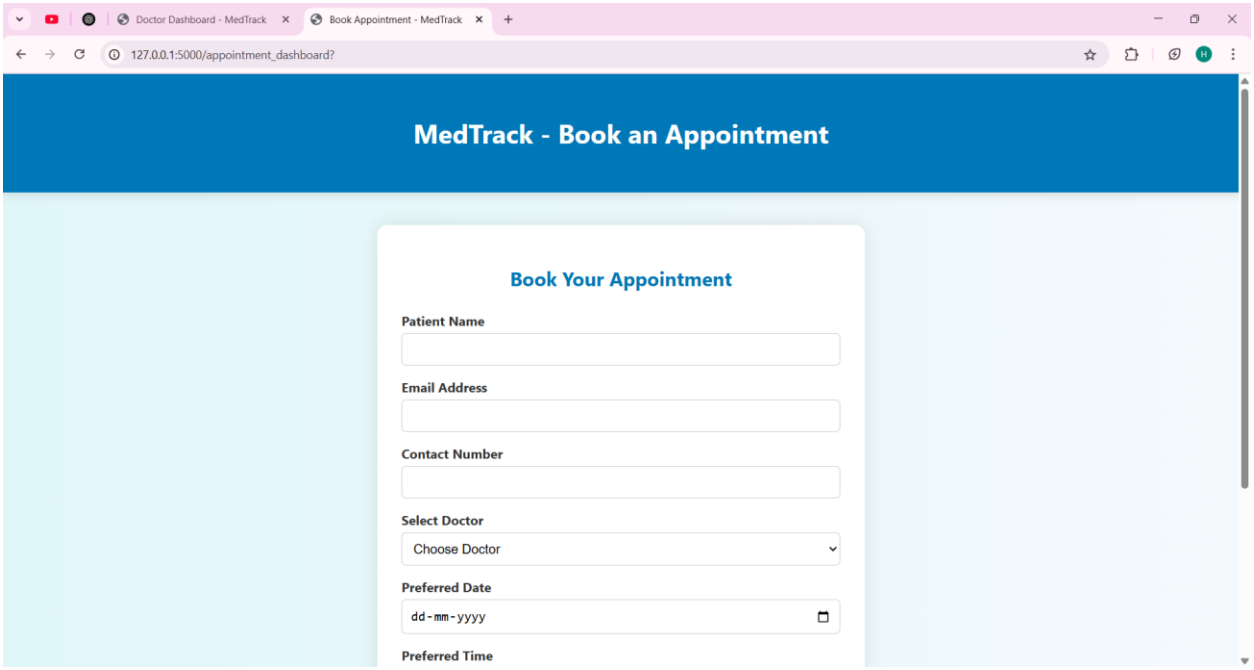


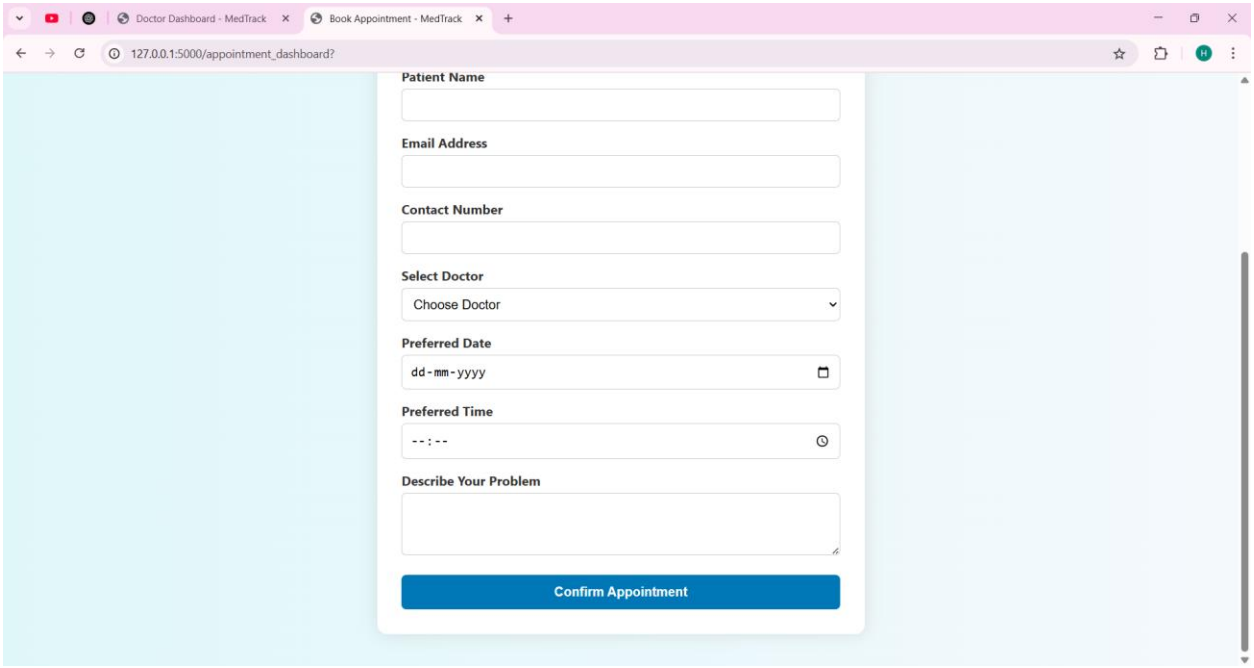
Patient Dashboard:



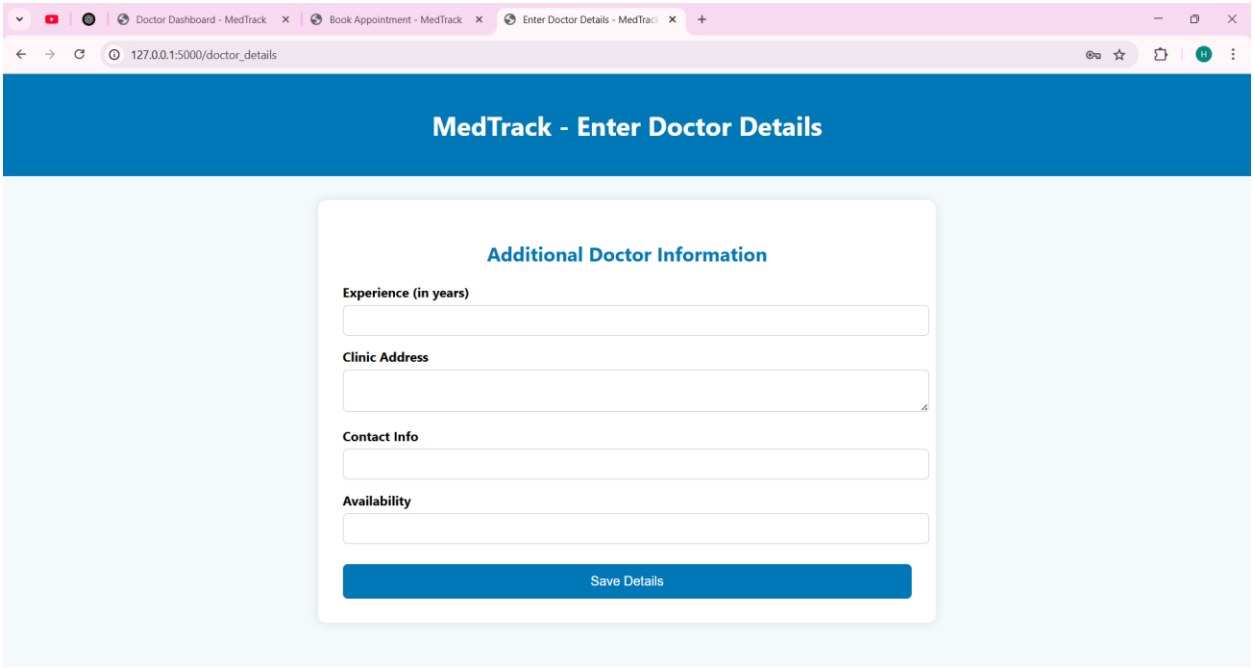


Appointment Booking Page:

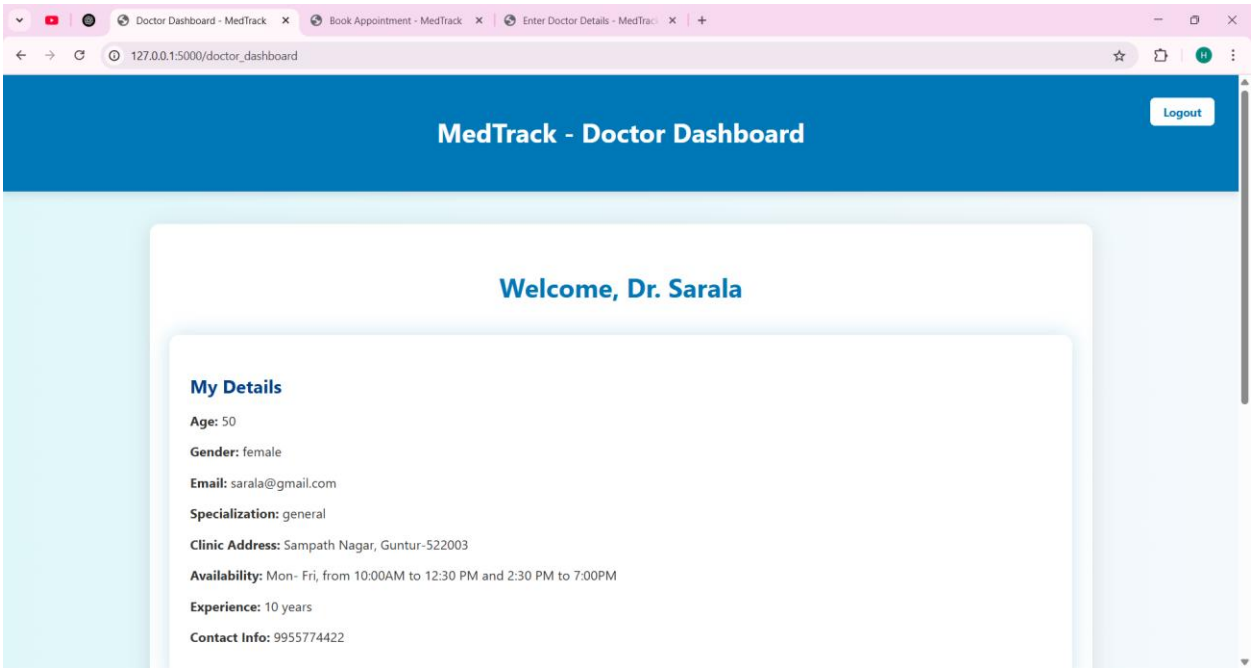


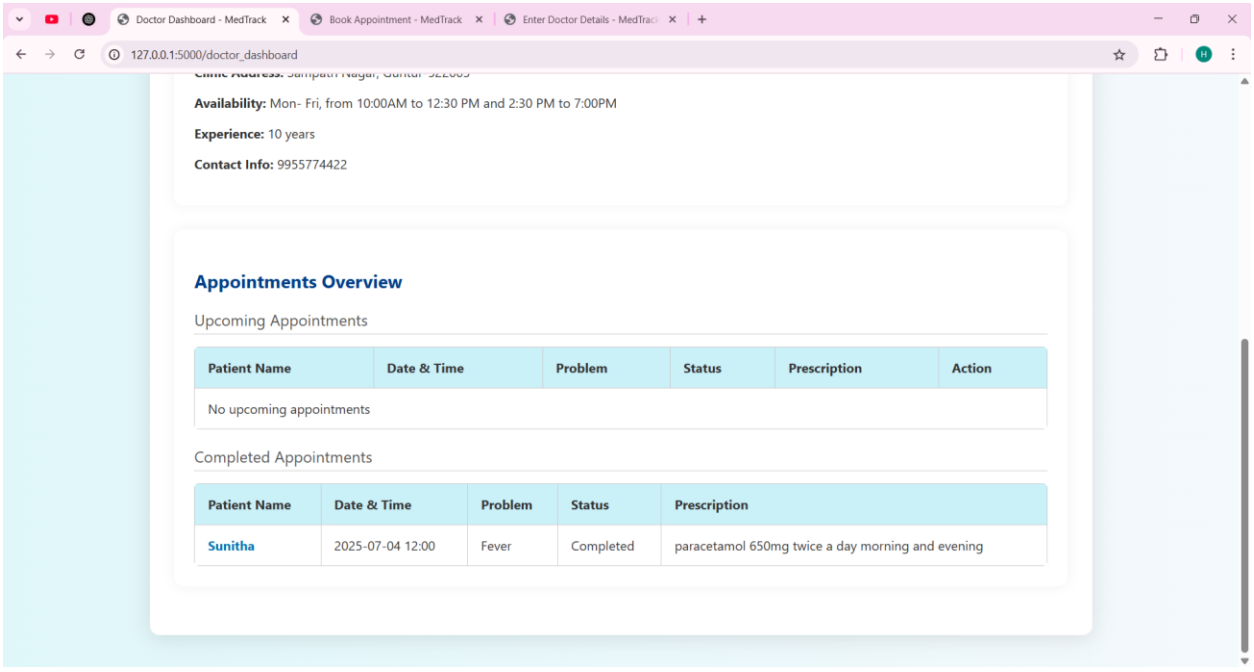


Doctor Details Page:

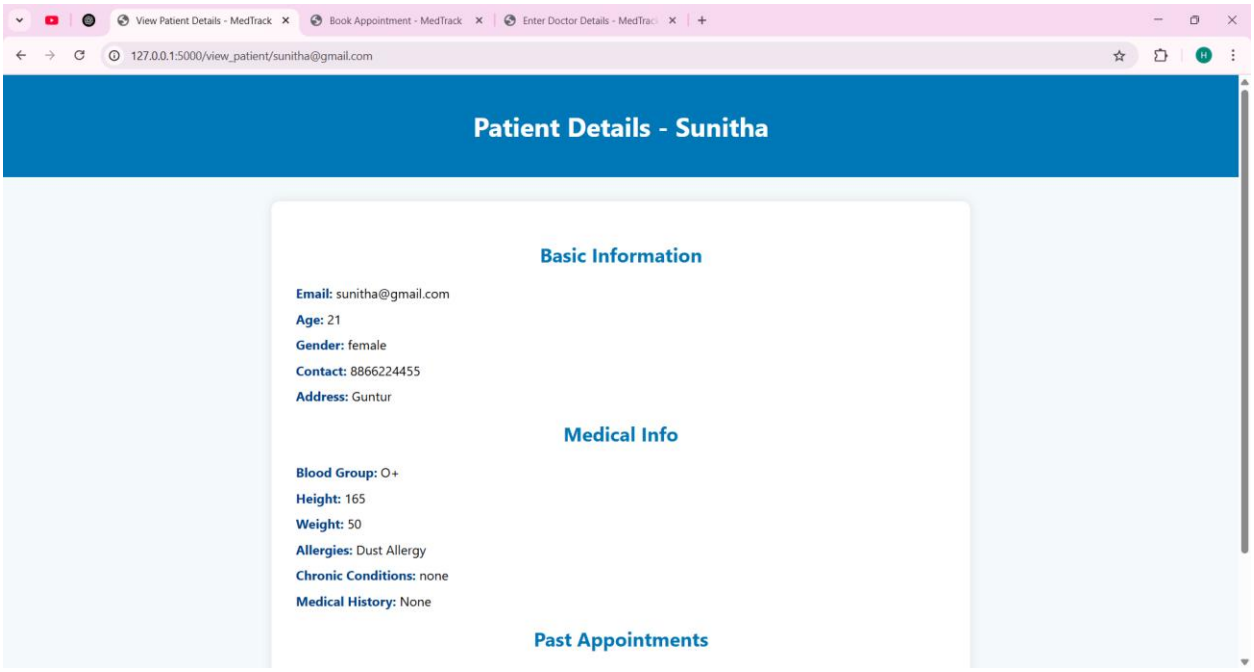


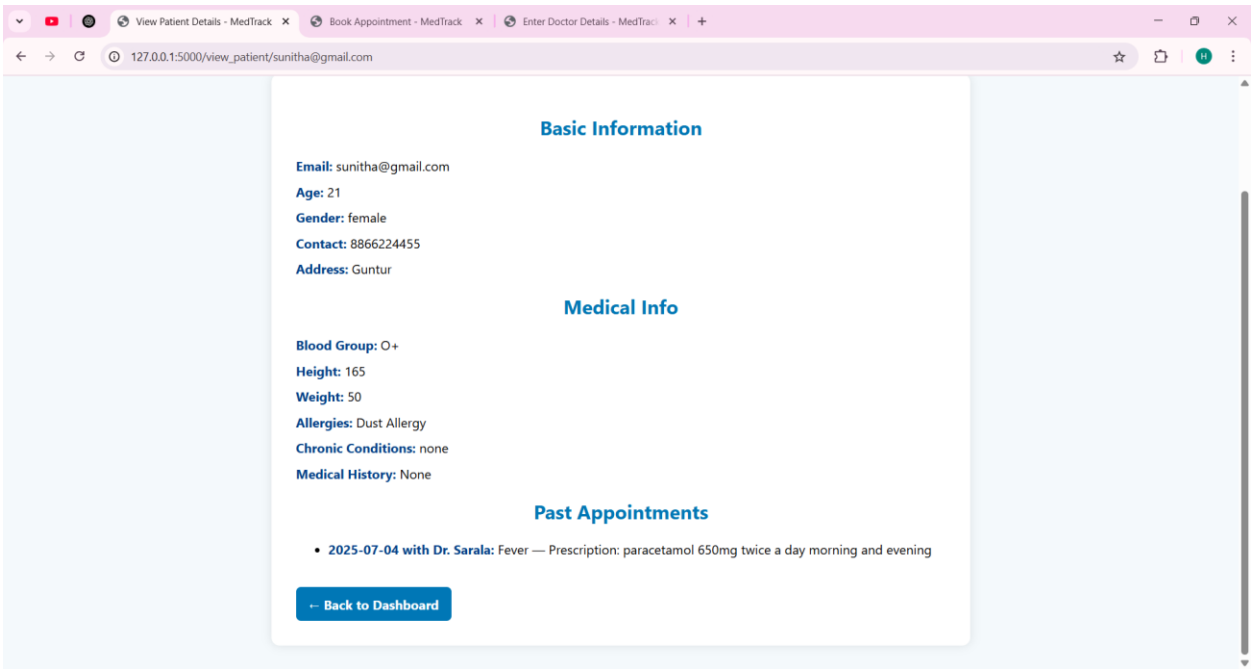
Doctor Dashboard Page:





View Patient Details Page:





Local Database updations :

1. **Users table :**

```
31 •  select * from user;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

id	role	name	email	password	age	gender	specialization
1	doctor	Sunitha	suni@gmail.com	suni1234	40	female	general
3	patient	Saketh	saketh@gmail.com	sak1234	15	male	
4	patient	Praneetha	praneetha@gmail.com	pra1234	21	female	
5	doctor	Ravindra	ravi@gmail.com	ravi1234	45	male	general
6	patient	Vamsi	vamsi@gmail.com	vamsi1234	11	male	
7	doctor	Suma Latha	suma@gmail.com	suma1234	38	female	gynecology
8	doctor	Ramesh	ramesh@gmail.com	ram1234	48	male	dental
9	patient	Sowmya	sowmya@gmail.com	sow1234	21	female	
10	doctor	Sarala	sarala@gmail.com	sarala1234	50	female	ent
11	patient	Siva	siva@gmail.com	siva1234	18	male	

2. **Doctor Details table :**

```
34 •  select * from doctor_detail;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	email	experience	clinic_address	contact	availability
2	suni@gmail.com	8	Laksmipuram, Guntur	9876543210	Mon-Fri 9am-5pm
3	ravi@gmail.com	10	Koritipadu , Guntur	9596442371	From 10:00AM to 12:30PM
4	suma@gmail.com	8	Inner Ring Road, Guntur	9955774422	Mon- Fri, from 10:00AM to 12:30PM
5	ramesh@gmail.com	9	Nallapadu, Guntur	8844662277	Mon- Fri, from 10:00AM to 12:30 PM ar
6	sarala@gmail.com	10	Kothapeta, Guntur	8321654972	Mon- Fri, from 10:00AM to 12:30PM
NULL	NULL	NULL	NULL	NULL	NULL

3. Patient Details table:

```
35 •  select * from patient_detail;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	email	contact	address	height	weight	blood_group	allergies	conditions	histor
1	saketh@gmail.com	7755331155	guntur	155	50	o+	none	none	none
2	praneetha@gmail.com	8529637413	guntur	160	55	o+	none	none	none
3	vamsi@gmail.com	8321654972	Guntur	145	40	o-	none	none	none
4	sowmya@gmail.com	8654723791	Guntur	165	50	o+	none	none	None
5	siva@gmail.com	9652387142	Guntur	175	50	o+	none	none	none
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

4. Appointment table:

```
32 •  select * from appointment;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	patient_name	email	phone	doctor	date	time	problem
1	Saketh	saketh@gmail.com	7755331155	Sunitha	2025-06-28	11:00	Fever
2	Praneetha	praneetha@gmail.com	8529637413	Sunitha	2025-06-27	11:00	Fever
3	Praneetha	praneetha@gmail.com	852147963	Ravindra	2025-06-27	11:00	Fever
4	Vamsi	vamsi@gmail.com	8321654972	Sunitha	2025-06-28	11:00	Vomitings and stomach
5	Vamsi	vamsi@gmail.com	8321654972	Ravindra	2025-06-27	12:00	Fever
6	Praneetha	praneetha@gmail.com	8529637413	Suma Latha	2025-06-27	12:15	Imbalance Menstrual c
7	Praneetha	praneetha@gmail.com	8529637413	Ravindra	2025-06-29	12:00	Vomitings and Fever
8	Sowmya	sowmya@gmail.com	8654723791	Ramesh	2025-06-28	03:00	Severe Tooth Pain
9	Vamsi	vamsi@gmail.com	8321654972	Ramesh	2025-06-28	03:30	Regular Tooth Checku
10	Siva	siva@gmail.com	9652387142	Sarala	2025-06-29	11:00	Severe ear pain
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

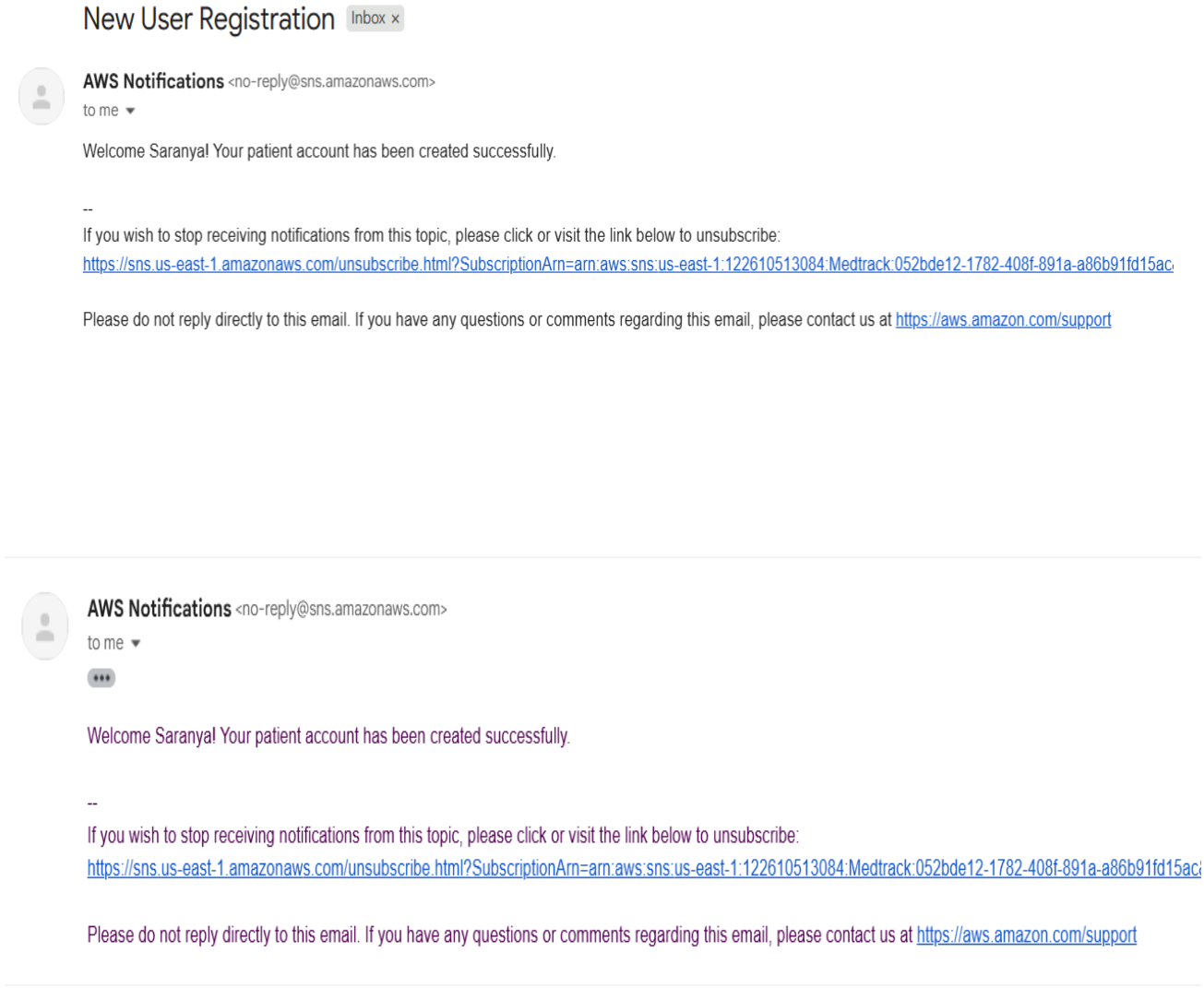
5. Medical History Table:

```
36 •  select * from medical_history;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	email	date	doctor	diagnosis	prescription
1	saketh@gmail.com	2025-06-28	Sunitha	Fever	paracetamol 50mg twice a day m
2	praneetha@gmail.com	2025-06-27	Sunitha	Fever	paracetamol 50mg twice a day m
3	praneetha@gmail.com	2025-06-27	Ravindra	Fever	paracetamol 50mg twice a day m
4	vamsi@gmail.com	2025-06-27	Ravindra	Fever	paracetamol 50mg twice a day m
5	vamsi@gmail.com	2025-06-28	Sunitha	Vomitings and stomach pain	use meftol medicine for per day a
6	praneetha@gmail.com	2025-06-27	Suma Latha	Imbalance Menstrual cycle	Eat and sleep well, eat papaya a
7	praneetha@gmail.com	2025-06-29	Ravindra	Vomitings and Fever	paracetamol 50mg twice a day m
8	sowmya@gmail.com	2025-06-28	Ramesh	Severe Tooth Pain	You are suffering due to cavity b
9	vamsi@gmail.com	2025-06-28	Ramesh	Regular Tooth Checkup	Your Teeth were all good just bru
10	siva@gmail.com	2025-06-29	Sarala	Severe ear pain	Use some wet cloth dipped in hot
NULL	NULL	NULL	NULL	NULL	NULL

Mail to the admin:



New User Registration Inbox ×

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

Welcome Saranya! Your patient account has been created successfully.

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:122610513084:Medtrack:052bde12-1782-408f-891a-a86b91fd15ac

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support

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

Welcome Saranya! Your patient account has been created successfully.

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:122610513084:Medtrack:052bde12-1782-408f-891a-a86b91fd15ac

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support

Conclusion:

The **MedTrack application** has been successfully developed and deployed using a robust cloud-based architecture tailored for modern healthcare environments. Leveraging AWS services such as EC2 for hosting, DynamoDB for secure and scalable patient data management, and SNS for real-time alerts, the platform ensures reliable and efficient access to essential medical tracking services. This system addresses critical challenges in healthcare such as managing patient records, monitoring medication schedules, and ensuring timely communication between healthcare providers and patients.

The cloud-native approach enables seamless scalability, allowing MedTrack to support increasing numbers of users and data without compromising performance or reliability. The integration of Flask with AWS ensures smooth backend operations, including patient registration, medication reminders, and health updates. Thorough testing has validated that all features—from user onboarding to alert notifications—function reliably and securely.

In conclusion, the MedTrack application delivers a smart, efficient solution for modernizing healthcare management, improving patient care, and streamlining communication between medical staff and patients. This project highlights the transformative power of cloud-based technologies in solving real-world challenges in the healthcare sector.