

MedTrack: AWS Cloud-Enabled Healthcare Management System

Project Description:

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

Scenario 1: Efficient Appointment Booking System for Patients

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

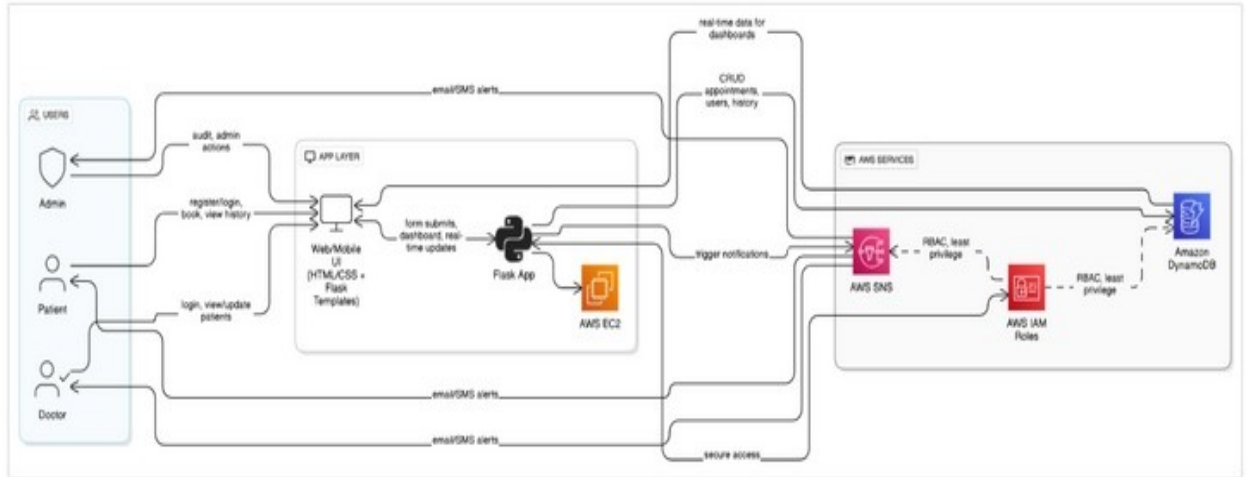
Scenario 2: Secure User Management with IAM

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

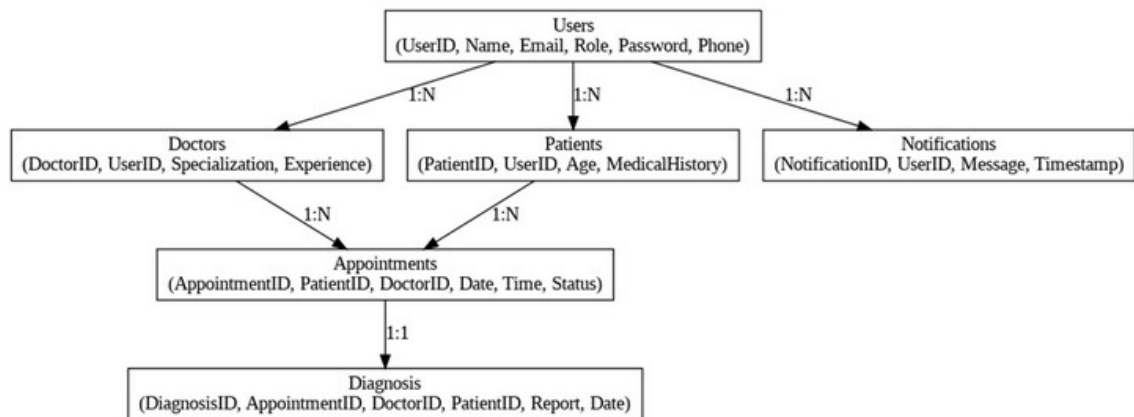
Scenario 3: Easy Access to Medical History and Resources

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

AWS ARCHITECTURE



Entity Relationship (ER)Diagram:



Pre-requisites:

- AWS Account Setup:
<https://docs.aws.amazon.com/accounts/latest/reference/getting-started.html>
- AWS IAM (Identity and Access Management):
<https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>
- AWS EC2 (Elastic Compute Cloud):
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>

- AWS DynamoDB:
<https://docs.aws.amazon.com/amazondynamodb/Introduction.html>
- Amazon SNS:
<https://docs.aws.amazon.com/sns/latest/dg/welcome.html>
- Git Documentation:
<https://git-scm.com/doc>
- VS Code Installation: (download the VS Code using the below link or you can get that in Microsoft store)
<https://code.visualstudio.com/download>

Project WorkFlow:

Milestone 1. Web Application Development and Setup

Activity 1.1: Develop the Backend Using Flask.

Activity 1.2: Integrate AWS Services Using boto3.

Milestone 2. AWS Account Setup and Login

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

Activity 2.2: Login to AWS Management Console.

Milestone 3. DynamoDB Database Creation and Setup

Activity 3.1: Create a DynamoDB Table.

Activity 3.2: Configure Attributes for User Data and Book Requests.

Milestone 4. SNS Notification Setup

Activity 4.1: Create SNS topics for book request notifications.

Activity 4.2: Subscribe users and library staff to SNS email notifications.

Milestone 5. IAM Role Setup

Activity 5.1: Create IAM Role

Activity 5.2: Attach Policies

Milestone 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.

Milestone 7. Deployment on EC2

Activity 7.1: Upload Flask Files

Activity 7.2: Run the Flask App

Milestone 8. Testing and Deployment

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

Milestone 1: Web Application Development and Setup

Backend Development and Application Setup focuses on establishing the core structure of the application. This includes configuring the backend framework, setting up routing, and integrating database connectivity. It lays the groundwork for handling user interactions, data management, and secure access.

Please refer to this sample as a guide for local deployment :

<https://docs.google.com/document/d/1sFF7-tJ6IgWtRbawWoA4W3PkkxEFrSJZhKzULgLsjxo/edit?usp=sharing>

Important Instructions:

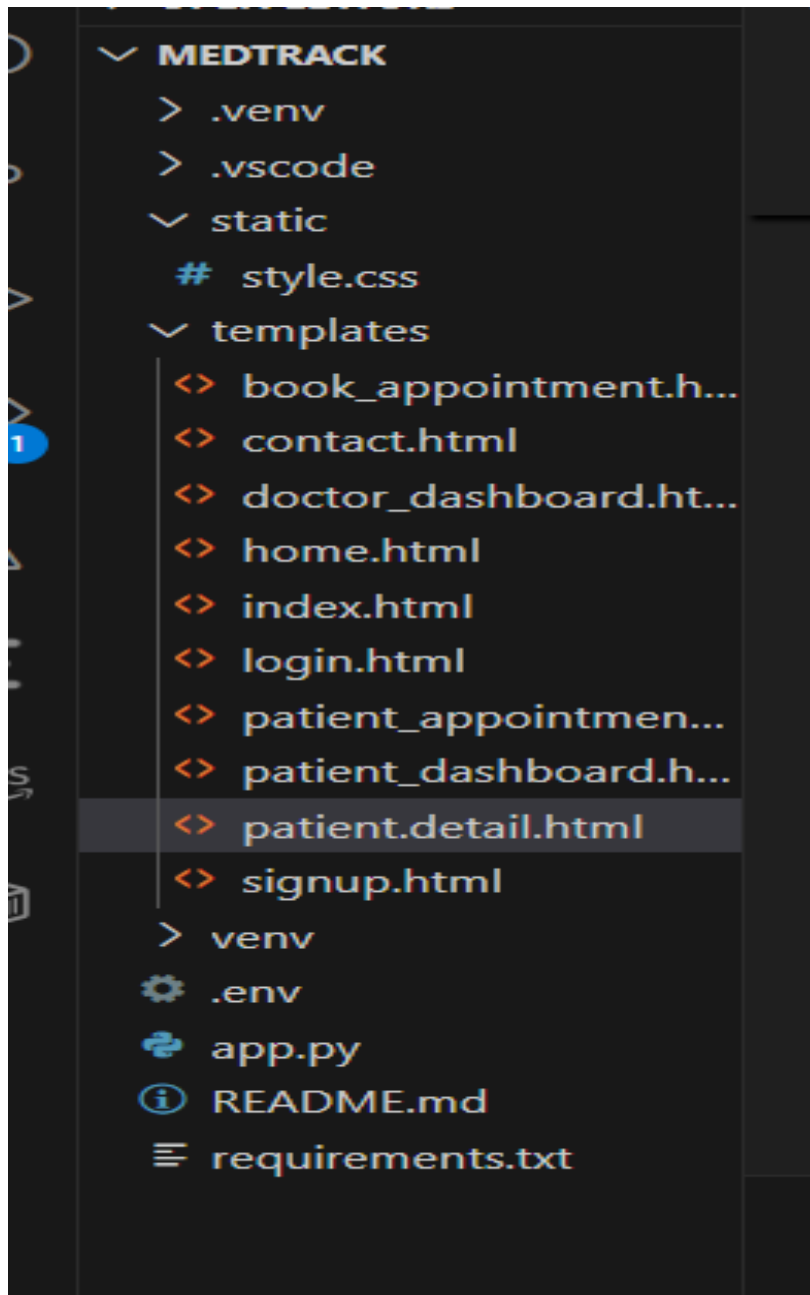
- Start by creating the necessary HTML pages and Flask routes (app.py) to build the core functionality of your application.
- During the initial development phase, store and retrieve data using Python dictionaries or lists locally. This will allow you to design, test, and validate your application logic without external database dependencies.
- Ensure your app runs smoothly with local data structures before integrating any cloud services.

Post Troven Access Activation:

- Once Troven Labs access is provided (valid for 3 hours), you must immediately proceed with Milestone 1 of your Guided Project instructions.
- At this point, modify your app.py and replace local dictionary/list operations with AWS services (such as DynamoDB, RDS, or others as per project requirements).
- Using the temporary credentials provided by Troven Labs, securely connect your application to AWS resources.
- Since the AWS configuration is lightweight and already instructed in the milestones, you should be able to complete the cloud integration efficiently within the allotted time.

FLASK DEPLOYMENT

- File Explorer Structure

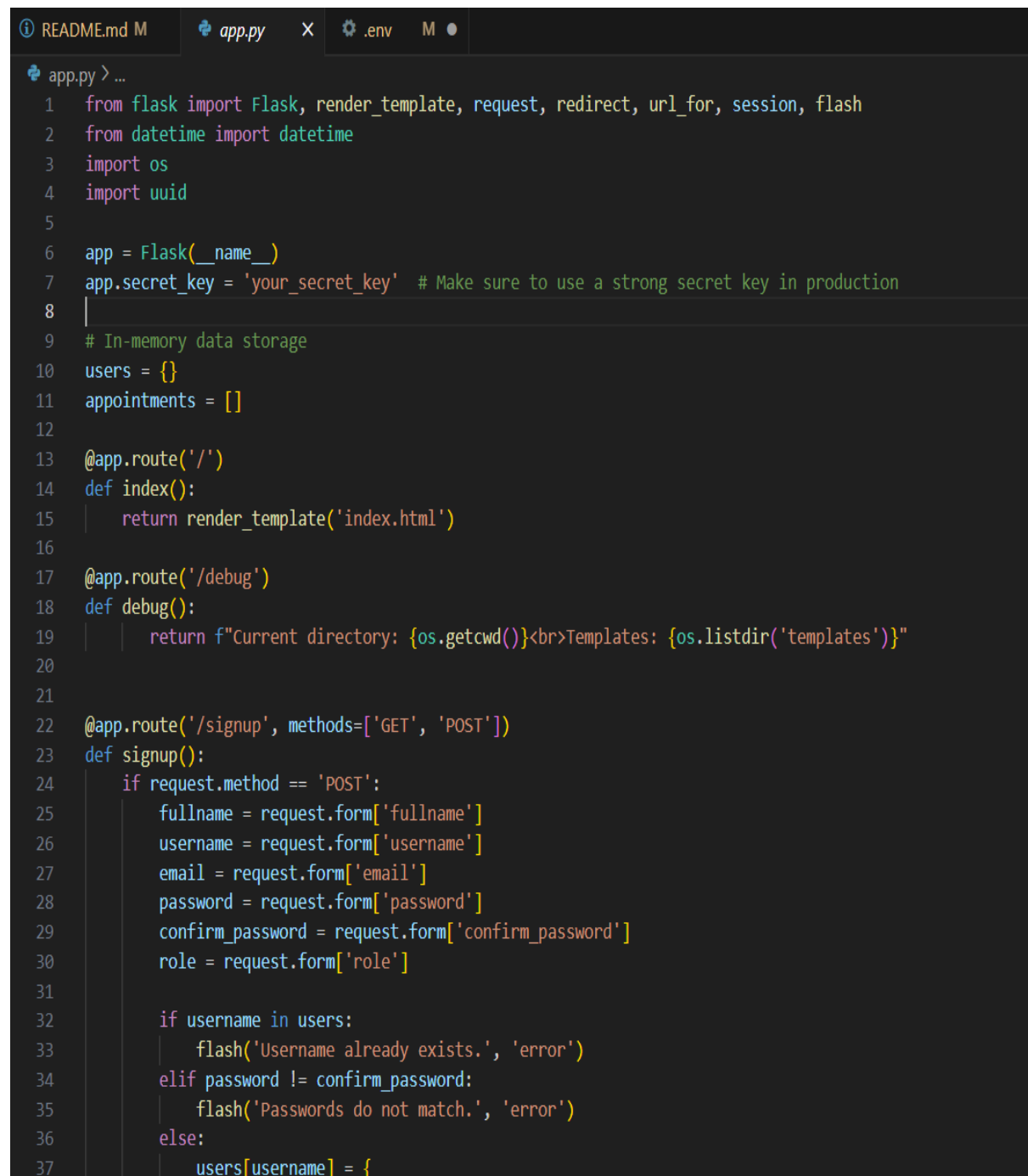


Description of the code :

Flask App Initialization:

In the MedTrack project, the Flask app is initialized to establish the backend infrastructure, enabling it to handle multiple user interactions such as patient registration, appointment booking, and submission of medical reports. The

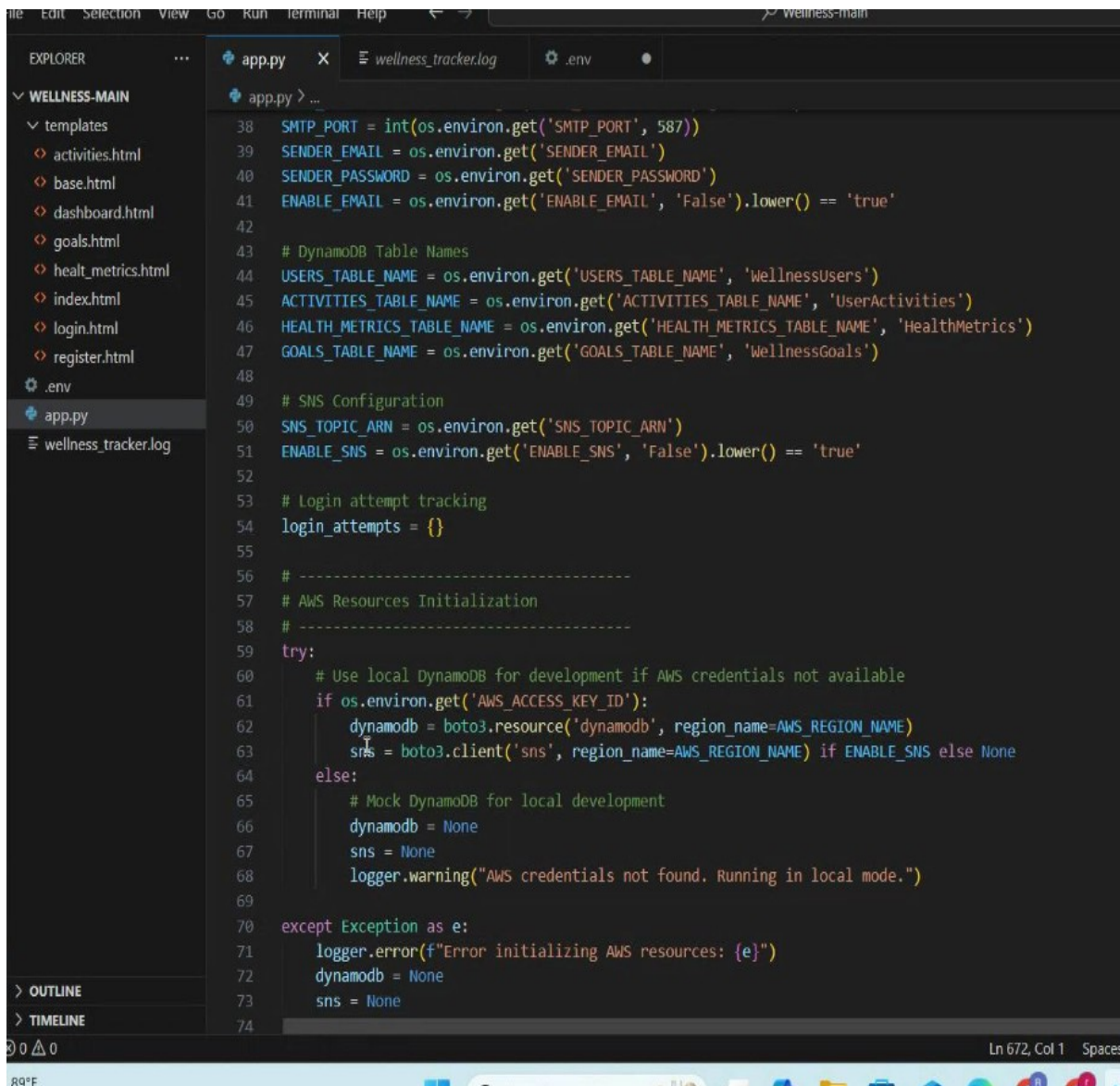
Flask framework processes incoming requests, communicates with the DynamoDB database for storing user data, and integrates seamlessly with AWS services. Additionally, the routes and APIs are defined to manage different functionalities like secure login, appointment scheduling, and medical history retrieval. This initialization sets up the foundation for smooth, real-time communication between patients and doctors while ensuring the app is scalable and secure.



```
app.py > ...
1  from flask import Flask, render_template, request, redirect, url_for, session, flash
2  from datetime import datetime
3  import os
4  import uuid
5
6  app = Flask(__name__)
7  app.secret_key = 'your_secret_key' # Make sure to use a strong secret key in production
8  |
9  # In-memory data storage
10 users = {}
11 appointments = []
12
13 @app.route('/')
14 def index():
15     return render_template('index.html')
16
17 @app.route('/debug')
18 def debug():
19     return f"Current directory: {os.getcwd()}<br>Templates: {os.listdir('templates')}"
20
21
22 @app.route('/signup', methods=['GET', 'POST'])
23 def signup():
24     if request.method == 'POST':
25         fullname = request.form['fullname']
26         username = request.form['username']
27         email = request.form['email']
28         password = request.form['password']
29         confirm_password = request.form['confirm_password']
30         role = request.form['role']
31
32         if username in users:
33             flash('Username already exists.', 'error')
34         elif password != confirm_password:
35             flash('Passwords do not match.', 'error')
36         else:
37             users[username] = {
```

SNS and Dynamodb initialization:

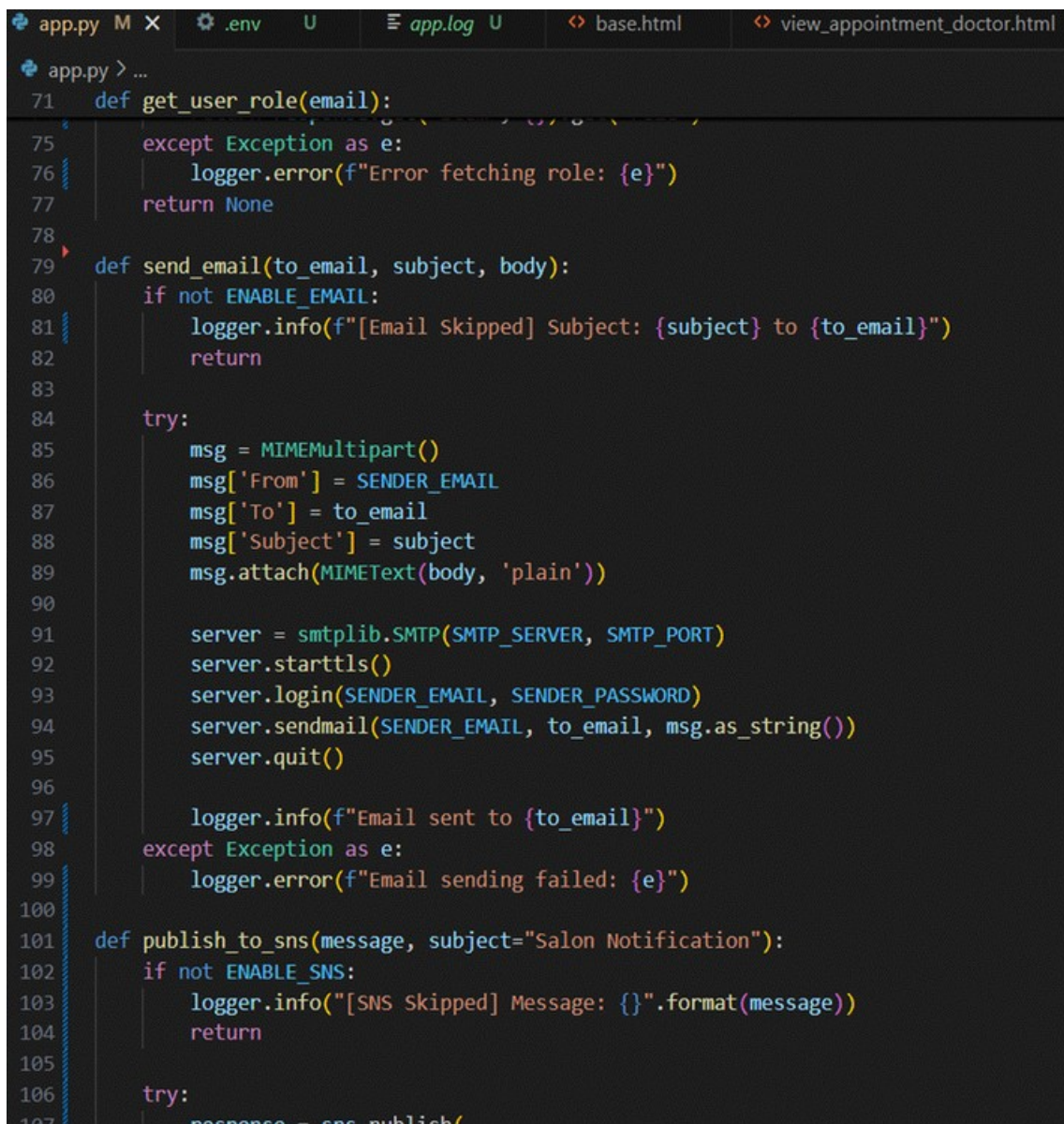
- In the MedTrack project, AWS SNS sends real-time notifications to patients and doctors about appointments and updates. DynamoDB stores user data, medical records, and appointments securely, offering fast, scalable access. Both services are integrated with Flask to ensure smooth communication and efficient data management.



```
38 SMTP_PORT = int(os.environ.get('SMTP_PORT', 587))
39 SENDER_EMAIL = os.environ.get('SENDER_EMAIL')
40 SENDER_PASSWORD = os.environ.get('SENDER_PASSWORD')
41 ENABLE_EMAIL = os.environ.get('ENABLE_EMAIL', 'False').lower() == 'true'
42
43 # DynamoDB Table Names
44 USERS_TABLE_NAME = os.environ.get('USERS_TABLE_NAME', 'WellnessUsers')
45 ACTIVITIES_TABLE_NAME = os.environ.get('ACTIVITIES_TABLE_NAME', 'UserActivities')
46 HEALTH_METRICS_TABLE_NAME = os.environ.get('HEALTH_METRICS_TABLE_NAME', 'HealthMetrics')
47 GOALS_TABLE_NAME = os.environ.get('GOALS_TABLE_NAME', 'WellnessGoals')
48
49 # SNS Configuration
50 SNS_TOPIC_ARN = os.environ.get('SNS_TOPIC_ARN')
51 ENABLE_SNS = os.environ.get('ENABLE_SNS', 'False').lower() == 'true'
52
53 # Login attempt tracking
54 login_attempts = {}
55
56 # -----
57 # AWS Resources Initialization
58 # -----
59 try:
60     # Use local DynamoDB for development if AWS credentials not available
61     if os.environ.get('AWS_ACCESS_KEY_ID'):
62         dynamodb = boto3.resource('dynamodb', region_name=AWS_REGION_NAME)
63         sns = boto3.client('sns', region_name=AWS_REGION_NAME) if ENABLE_SNS else None
64     else:
65         # Mock DynamoDB for local development
66         dynamodb = None
67         sns = None
68         logger.warning("AWS credentials not found. Running in local mode.")
69
70 except Exception as e:
71     logger.error(f"Error initializing AWS resources: {e}")
72     dynamodb = None
73     sns = None
74
```

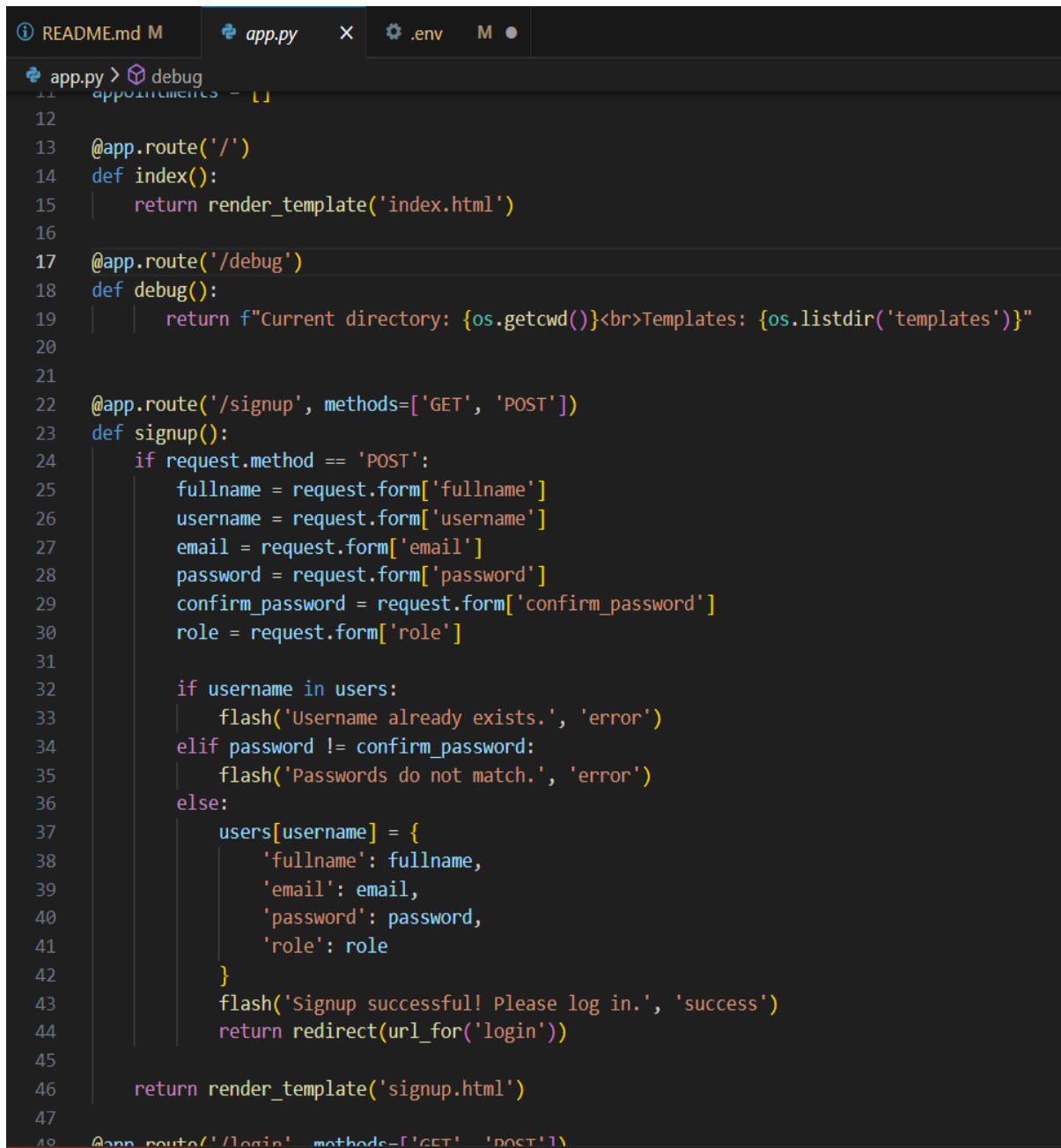

- **SNS Connection**


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. Create an 'App password' for the email ID and store it in the `SENDER_PASSWORD` section.

A screenshot of a code editor with a dark theme. The editor shows a Python file named `app.py` with line numbers from 71 to 107. The code defines two functions: `send_email` and `publish_to_sns`. The `send_email` function checks if email is enabled, constructs an email message using `MIMEMultipart`, and sends it via `smtplib`. The `publish_to_sns` function checks if SNS is enabled and publishes a message to an SNS topic. The code includes error handling with `try` and `except` blocks and uses `logger` for logging.

```
app.py > ...
71 def get_user_role(email):
72     ...
73     ...
74     ...
75 except Exception as e:
76     logger.error(f"Error fetching role: {e}")
77     return None
78
79 def send_email(to_email, subject, body):
80     if not ENABLE_EMAIL:
81         logger.info(f"[Email Skipped] Subject: {subject} to {to_email}")
82         return
83
84     try:
85         msg = MIMEMultipart()
86         msg['From'] = SENDER_EMAIL
87         msg['To'] = to_email
88         msg['Subject'] = subject
89         msg.attach(MIMEText(body, 'plain'))
90
91         server = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
92         server.starttls()
93         server.login(SENDER_EMAIL, SENDER_PASSWORD)
94         server.sendmail(SENDER_EMAIL, to_email, msg.as_string())
95         server.quit()
96
97         logger.info(f"Email sent to {to_email}")
98     except Exception as e:
99         logger.error(f"Email sending failed: {e}")
100
101 def publish_to_sns(message, subject="Salon Notification"):
102     if not ENABLE_SNS:
103         logger.info("[SNS Skipped] Message: {}".format(message))
104         return
105
106     try:
107         response = sns.publish(
```


- **Routes for Web Pages:**
Register Page



```
11 appointments -  debug
12
13 @app.route('/')
14 def index():
15     return render_template('index.html')
16
17 @app.route('/debug')
18 def debug():
19     return f"Current directory: {os.getcwd()}<br>Templates: {os.listdir('templates')}}"
20
21
22 @app.route('/signup', methods=['GET', 'POST'])
23 def signup():
24     if request.method == 'POST':
25         fullname = request.form['fullname']
26         username = request.form['username']
27         email = request.form['email']
28         password = request.form['password']
29         confirm_password = request.form['confirm_password']
30         role = request.form['role']
31
32         if username in users:
33             flash('Username already exists.', 'error')
34         elif password != confirm_password:
35             flash('Passwords do not match.', 'error')
36         else:
37             users[username] = {
38                 'fullname': fullname,
39                 'email': email,
40                 'password': password,
41                 'role': role
42             }
43             flash('Signup successful! Please log in.', 'success')
44             return redirect(url_for('login'))
45
46     return render_template('signup.html')
47
48 @app.route('/login', methods=['GET', 'POST'])
```

- The **login route** handles user authentication by verifying credentials stored in **DynamoDB**. Upon successful login, it increments the **login count** and redirects the user to their dashboard. This ensures secure access to the platform while maintaining user activity logs.

```
app.py x patient_details.html doctor_dashboard.html
app.py > ...
32
33 @app.route('/login', methods=['GET', 'POST'])
34 def login():
35     if request.method == 'POST':
36         username = request.form['username']
37         password = request.form['password']
38
39         user = users.get(username)
40         if user and user['password'] == password:
41             session['username'] = username
42             flash('Login successful.', 'success')
43             return redirect(url_for('home'))
44         else:
45             flash('Invalid credentials.', 'error')
46
47     return render_template('login.html')
48
```

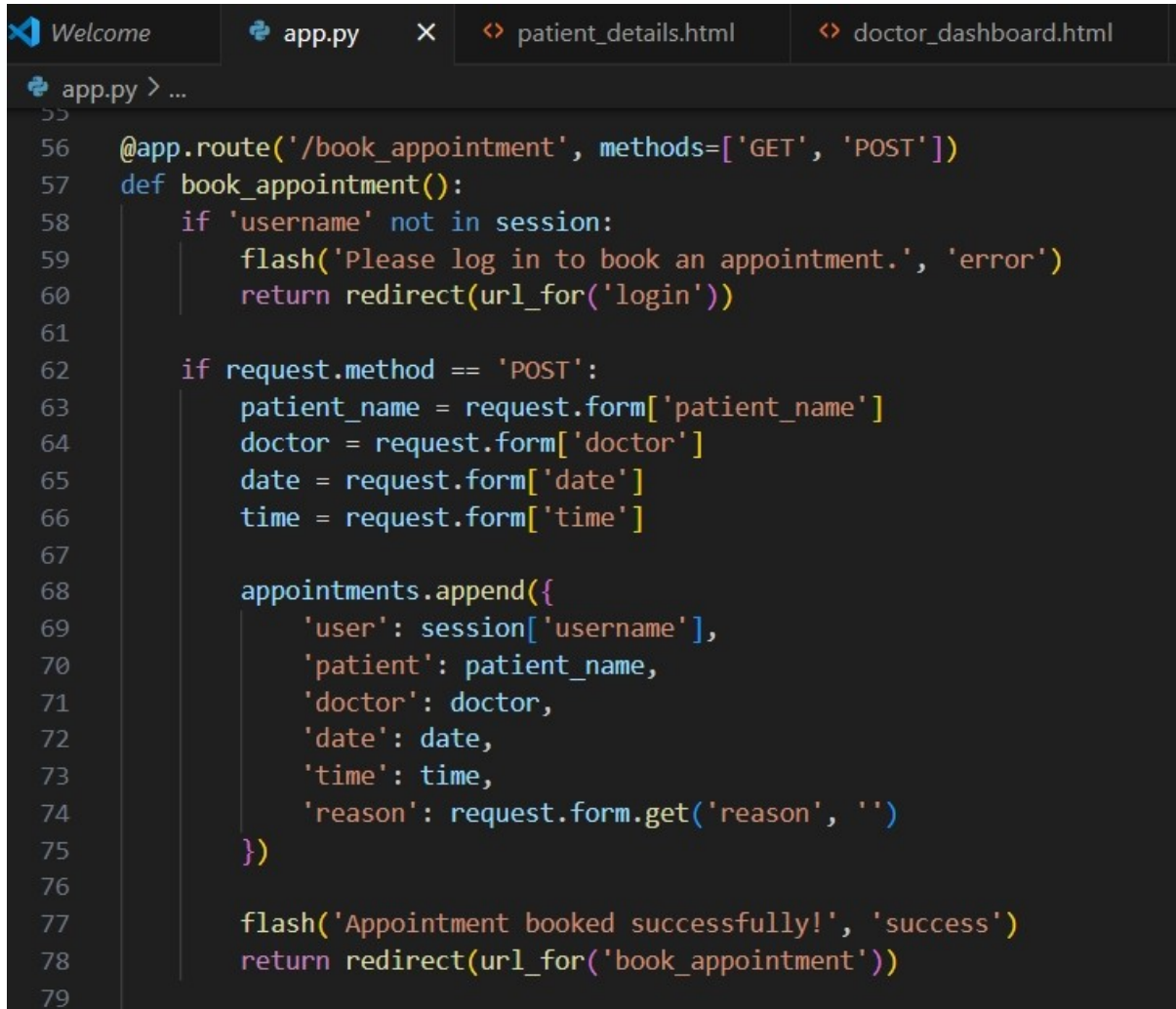
Logout Route:

The logout functionality allows users to securely end their session, clearing any session data and redirecting them to the login page. The dashboard provides users with an overview of their activities, such as upcoming appointments for patients or patient records for doctors, with relevant actions based on user roles.

```
welcome app.py x patient_details.html doctor_dashboard.html patient_dashboard.html
app.py > ...
63 def patient_details():
64
65     if not user:
66         flash('User not found.', 'error')
67         return redirect(url_for('login'))
68
69     return render_template('patient_details.html', username=username, email=user['email'])
70
71
72 @app.route('/logout')
73 def logout():
74     session.pop('username', None)
75     flash('Logged out successfully.', 'info')
76     return redirect(url_for('login'))
```

Book Appointment Route:

The book appointment route allows users to select a date, time, and doctor for their appointment. Upon submission, the system stores the appointment details in DynamoDB and sends a confirmation notification via SNS. This ensures smooth scheduling and timely updates for both patients and doctors.

A screenshot of a code editor with a dark theme. The editor has four tabs at the top: 'Welcome', 'app.py', 'patient_details.html', and 'doctor_dashboard.html'. The 'app.py' tab is active, showing Python code for a Flask application. The code defines a route for '/book_appointment' that handles both GET and POST requests. For GET requests, it checks if the user is logged in; if not, it flashes an error message and redirects to the login page. For POST requests, it extracts form data for patient name, doctor, date, and time, and appends this information to a list called 'appointments'. It also flashes a success message and redirects back to the booking page.

```
app.py > ...
56 @app.route('/book_appointment', methods=['GET', 'POST'])
57 def book_appointment():
58     if 'username' not in session:
59         flash('Please log in to book an appointment.', 'error')
60         return redirect(url_for('login'))
61
62     if request.method == 'POST':
63         patient_name = request.form['patient_name']
64         doctor = request.form['doctor']
65         date = request.form['date']
66         time = request.form['time']
67
68         appointments.append({
69             'user': session['username'],
70             'patient': patient_name,
71             'doctor': doctor,
72             'date': date,
73             'time': time,
74             'reason': request.form.get('reason', '')
75         })
76
77         flash('Appointment booked successfully!', 'success')
78         return redirect(url_for('book_appointment'))
79
```

Deployment Code:

The health routing feature in the MedTrack project checks the system's status by sending a request to a specific endpoint, ensuring the backend services are functioning properly. The `__name__ == '__main__'` block is used in the Flask app to ensure that the application runs only if the script is executed directly, not when imported as a module, enabling local

development or deployment on a server. This setup ensures that the app runs smoothly and is self-contained during execution.

```
Welcome  app.py  patient_details.html  doctor_dashboard.html  patient_dashboard.html
app.py > ...
144 def patient_appointments():
145     flash('Please log in.', 'error')
146     return redirect(url_for('login'))
147
148
149     user_appts = [a for a in appointments if a['user'] == session['username']]
150     return render_template('patient_appointments.html', appointments=user_appts)
151
152 @app.route('/patient_details')
153 def patient_details():
154     if 'username' not in session:
155         flash('Please log in to view your details.', 'error')
156         return redirect(url_for('login'))
157
158     username = session['username']
159     user = users.get(username)
160
161     if not user:
162         flash('User not found.', 'error')
163         return redirect(url_for('login'))
164
165     return render_template('patient_details.html', username=username, email=user['email'])
166
167
168
169 @app.route('/logout')
170 def logout():
171     session.pop('username', None)
172     flash('Logged out successfully.', 'info')
173     return redirect(url_for('login'))
174
175
176 if __name__ == '__main__':
177     app.run(debug=True)
```



AWS - Local Deployment Sample Code - Google Docs..

No description..

<https://docs.google.com/document/d/1sFF7-tJ6IgWtRbawWoA4W3PkxEFrSJZhKzULgLsjxo/edit?usp=sharing>

Milestone 2: AWS Account Setup

Important Notice: Use Troven Labs for AWS Access

Students are strictly advised not to create their own AWS accounts, as doing so may incur charges. Instead, we have set up a dedicated section called “Labs” on the Troven platform, which provides temporary and cost-free access to AWS services.

Once your website is locally deployed and fully functional, you must proceed with integrating AWS services only through the Troven Labs environment. This ensures secure, controlled access to AWS resources without any risk of personal billing.

All steps involving AWS (such as deploying to EC2, connecting to DynamoDB, or using SNS) must be carried out within the Troven Labs platform, as we've configured temporary credentials for each student.

Reminder: You must complete the Web Development task before gaining access to Troven. Once accessed, the AWS Console via Troven is available for only 3 hours—please plan your work accordingly.

Please follow the provided guidelines and access AWS exclusively through Troven to avoid unnecessary issues.

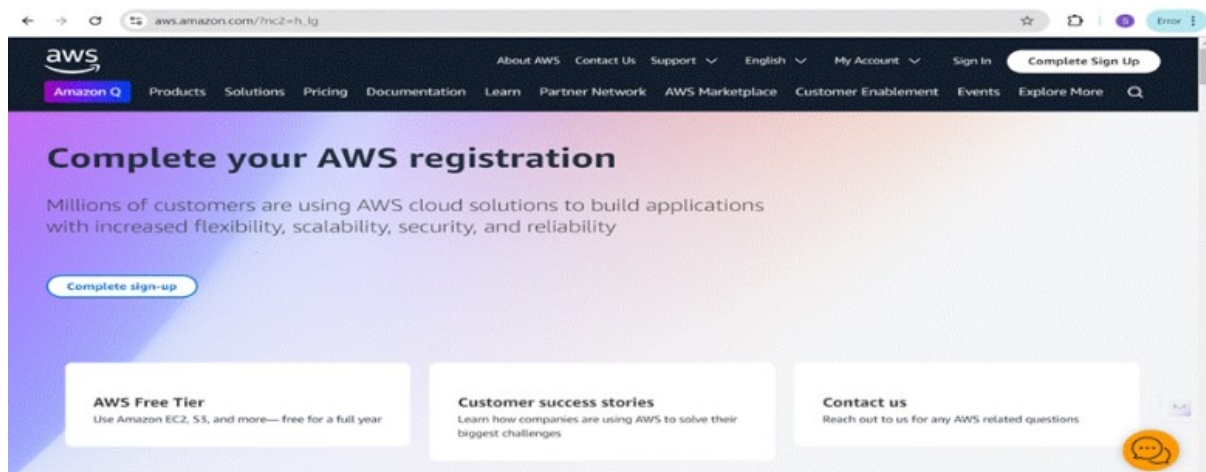
Please refer the below link -


<https://drive.google.com/file/d/1HzWc7AMJ2BrxhV-uaw5s0vWtcd-28qgl/view?usp=sharing>

AWS Account Setup and Login

This is for your understanding only, please refrain from creating an AWS account. A temporary account will be provided via Troven.


- Go to the AWS website (<https://aws.amazon.com/>).
- Click on the "Create an AWS Account" button.
- Follow the prompts to enter your email address and choose a password.
- Provide the required account information, including your name, address, and phone number.
- Enter your payment information. (Note: While AWS offers a free tier, a credit card or debit card is required for verification.)
- Complete the identity verification process.
- Choose a support plan (the basic plan is free and sufficient for starting).
- Once verified, you can sign in to your new AWS accounts.





Explore Free Tier products with a new AWS account.

To learn more, visit aws.amazon.com/free.



Sign up for AWS

Root user email address
Used for account recovery and as described in the [AWS Privacy Notice](#)


AWS account name
Choose a name for your account. You can change this name in your account settings after you sign up.

Verify email address

OR

Sign in to an existing AWS account

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



Sign in

☒ **Root user**
Account owner that performs tasks requiring unrestricted access. [Learn more](#)

☐ **IAM user**
User within an account that performs daily tasks. [Learn more](#)

Root user email address

Next

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 [Cookie Notice](#) for more information.

AI Use Case Explorer

Discover AI use cases,
customer success stories,
and expert-curated
implementation plans

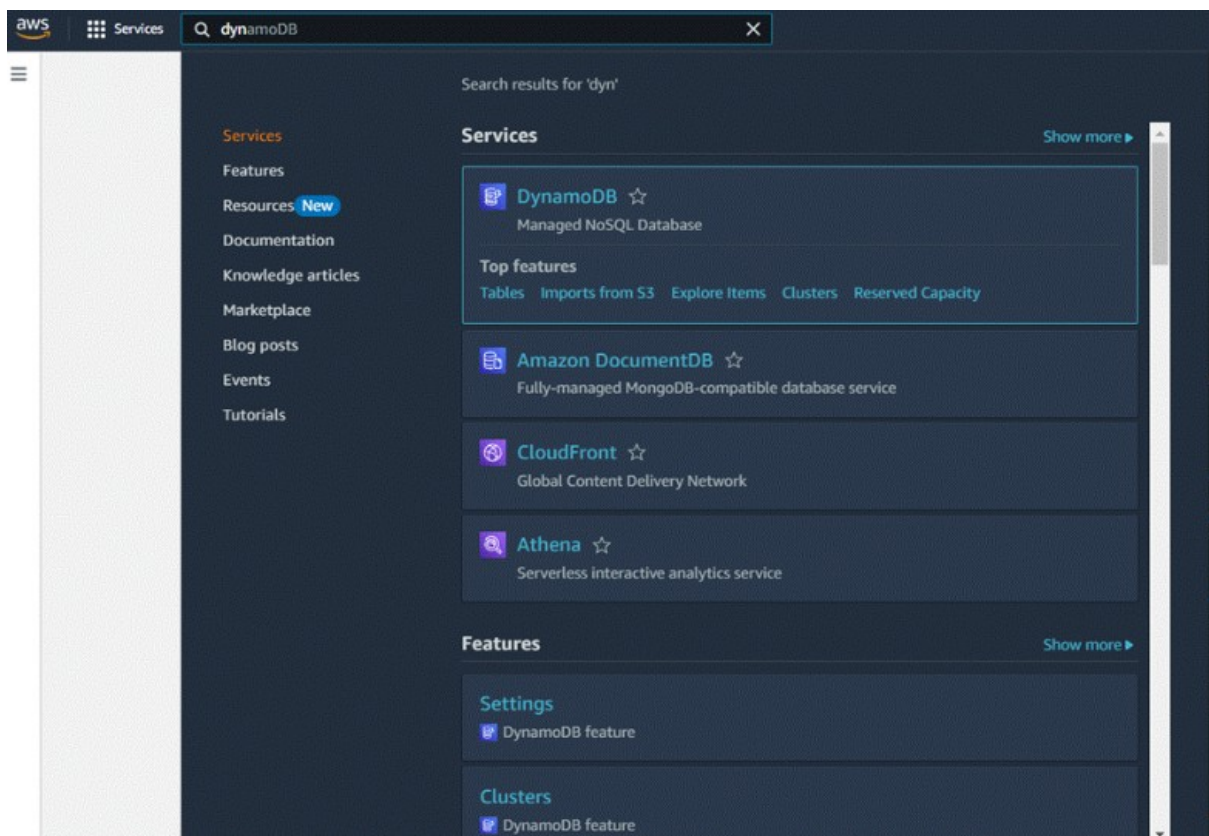
Explore now »

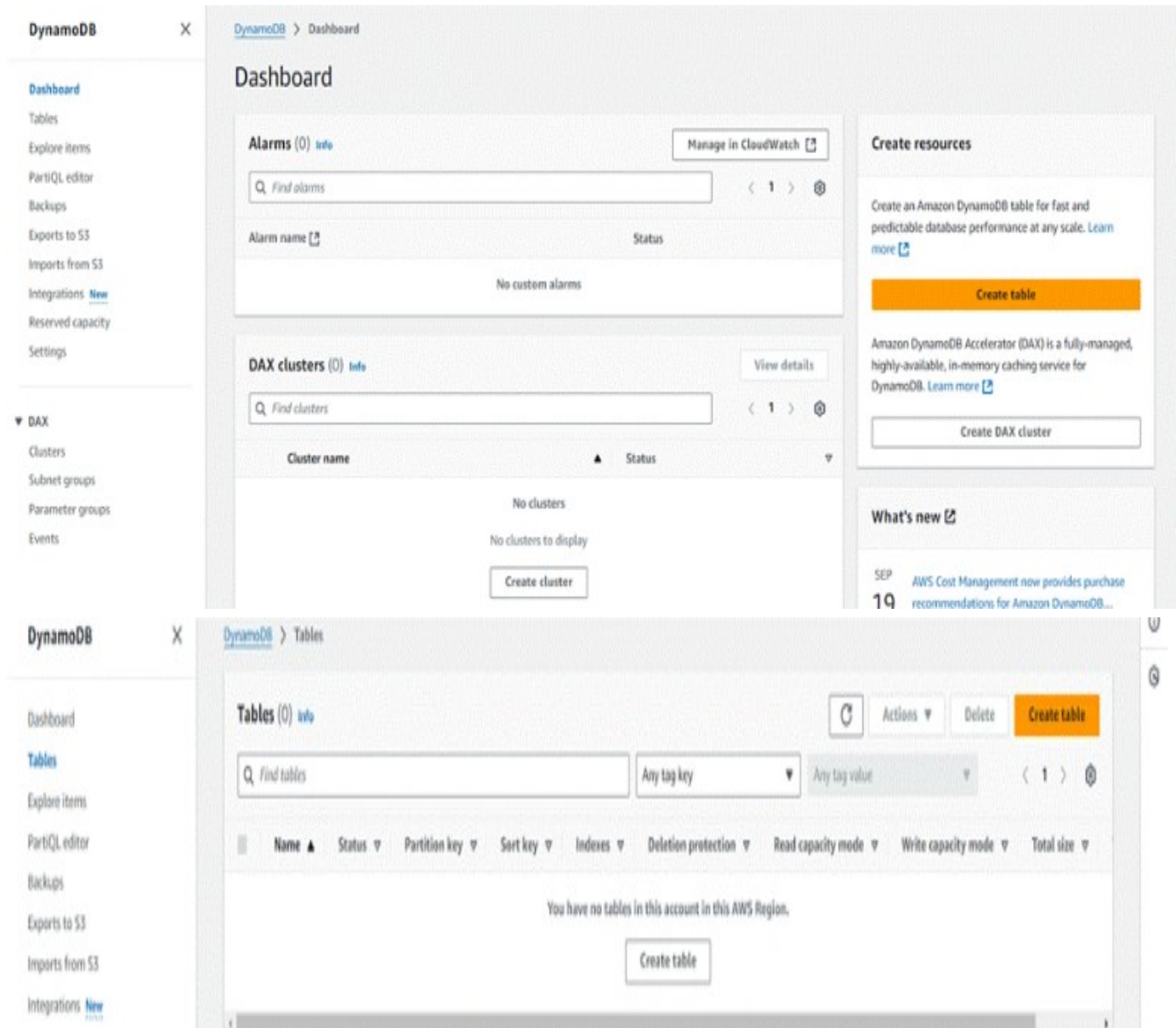
Milestone 3: DynamoDB Database Creation and Setup

Database Creation and Setup involves initializing a cloud-based NoSQL database to store and manage application data efficiently. This step includes defining tables, setting primary keys, and configuring read/write capacities. It ensures scalable, high-performance data storage for seamless backend operations.

Navigate to the DynamoDB

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





Create a DynamoDB table for storing data

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

[Alt+S]

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

Create table

Table details [Info](#)

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

Table name

This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

String ▼

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.

String ▼

1 to 255 characters and case sensitive.

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

Tags

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

No tags are associated with the resource.

You can add 50 more tags.

Cancel
Create table

- Create Appointments Table with partition key “appointment_id” with type String and click on create tables.

Create table

Table details [Info](#)

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

Table name

This will be used to identify your table.

AppointmentsTable

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.

appointment_id

String

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.

Enter the sort key name

String

1 to 255 characters and case sensitive.

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

Tags

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

No tags are associated with the resource.

Add new tag

You can add 50 more tags.

Cancel

Create table

Tables (2/10) Info

Q Status

Any tag key

Any tag value

< 1 >

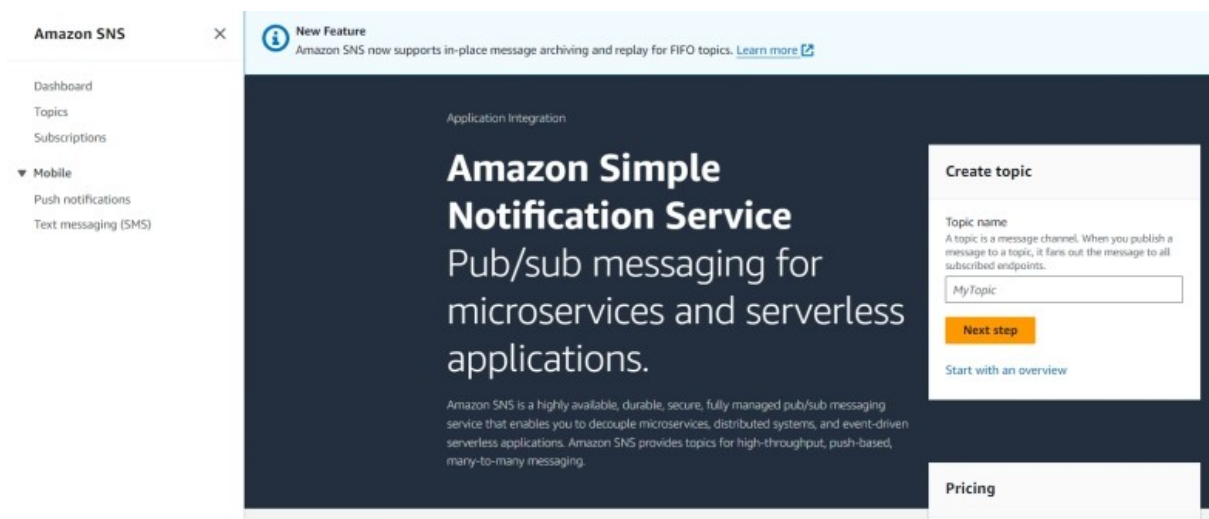
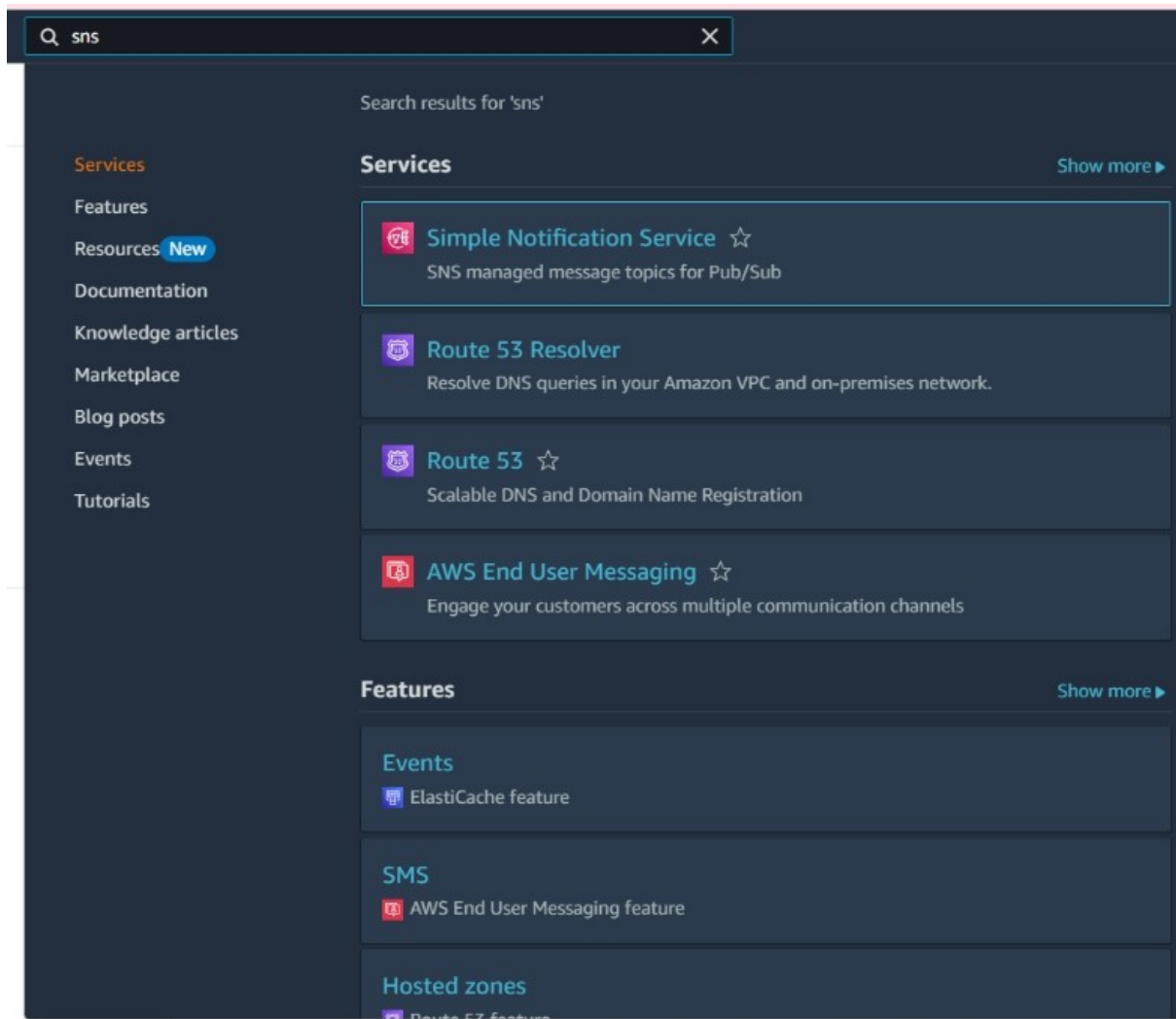
<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection
<input checked="" type="checkbox"/>	AppointmentsTable	Active	appointment_id (S)	-	0 0		Off
<input type="checkbox"/>	NextGenHospital_Appointments	Active	appointment_id (S)	-	0 0		Off
<input type="checkbox"/>	NextGenHospital_ContactMessages	Active	message_id (S)	-	0 0		Off
<input type="checkbox"/>	NextGenHospital_Doctors	Active	doctor_id (S)	-	0 0		Off
<input type="checkbox"/>	NextGenHospital_PatientRecords	Active	patient_id (S)	-	0 0		Off
<input type="checkbox"/>	NextGenHospital_Users	Active	email (S)	-	0 0		Off
<input type="checkbox"/>	SalonAppointments	Active	appointment_id (S)	user_email (S)	0 0		Off
<input type="checkbox"/>	SalonStylists	Active	stylist_id (S)	-	0 0		Off
<input type="checkbox"/>	SalonUsers	Active	email (S)	-	0 0		Off
<input checked="" type="checkbox"/>	UsersTable	Active	email (S)	-	0 0		Off

Milestone 4 : SNS Notification Setup

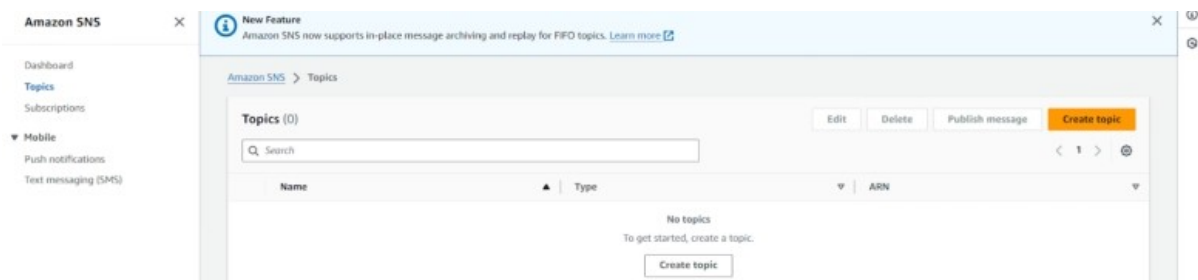
Amazon SNS is a fully managed messaging service that enables real-time notifications through channels like SMS, email, or app endpoints. You create topics, configure subscriptions, and integrate SNS into your app to send notifications based on specific events.

SNS topics for email notifications

- 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.

Amazon SNS > Topics > Create topic

New Feature
Amazon SNS now supports High Throughput FIFO topics. [Learn more](#)

Create topic

Details

Type | [Info](#)
Topic type cannot be modified after topic is created

☐ **FIFO (first-in, first-out)**

- Strictly-preserved message ordering
- Exactly-once message delivery
- Subscription protocols: SQS

☒ **Standard**

- Best-effort message ordering
- At-least once message delivery
- Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints

Name

Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).

Display name - optional | [Info](#)
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.

Maximum 100 characters.

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

Amazon SNS > Subscriptions > Create subscription

Topic ARN

Protocol
 The type of endpoint to subscribe

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

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

► **Subscription filter policy - optional** [Info](#)
 This policy filters the messages that a subscriber receives.

► **Redrive policy (dead-letter queue) - optional** [Info](#)
 Send undeliverable messages to a dead-letter queue.

[Cancel](#) [Create subscription](#)

- After subscription request for the mail confirmation

< **Subscriptions** Access policy Data protection policy Delivery policy (HTTP/S) Delivery status logging Encryption Tags >

Subscriptions (1) [Edit](#) [Delete](#) [Request confirmation](#) [Confirm subscription](#) [Create subscription](#)

ID	Endpoint	Status	Protocol
2c78944f-bb5d-4fb1-9982-74334...	sairaviteja478@gmail.com	Confirmed	EMAIL

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

AWS Notification - Subscription Confirmation Inbox x

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

9

to me ▼

You have chosen to subscribe to the topic:

arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications

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](#)

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

to me ▼

You have chosen to subscribe to the topic:

arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications

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!

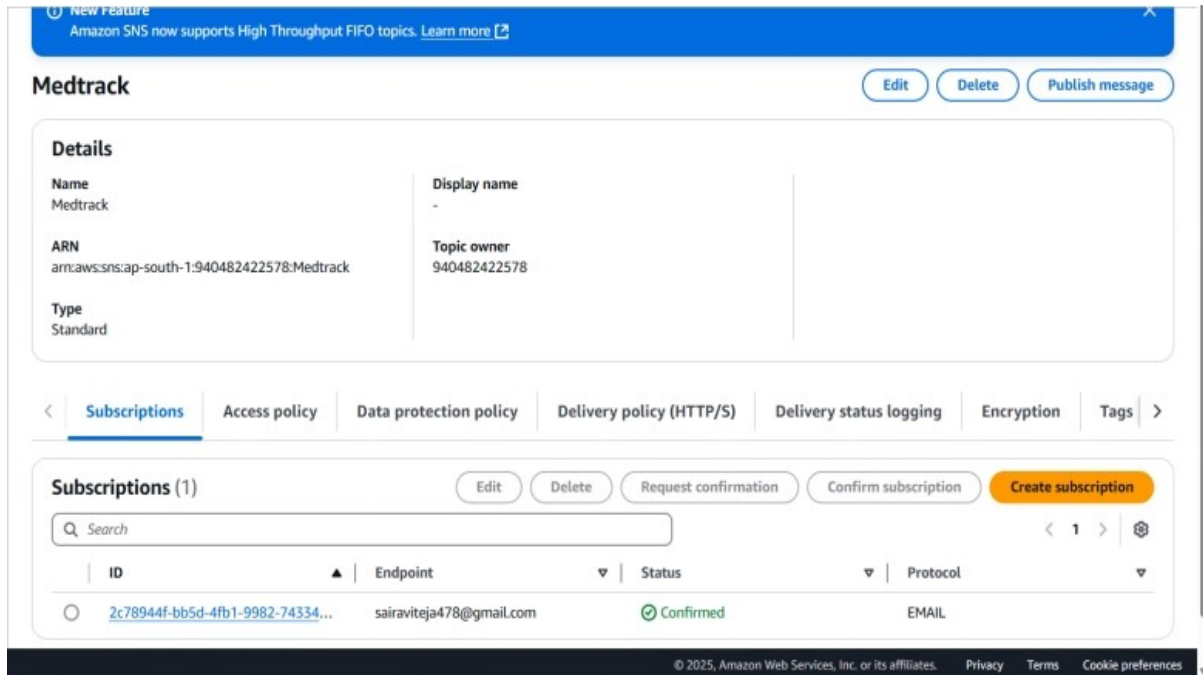
You have successfully subscribed.

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.

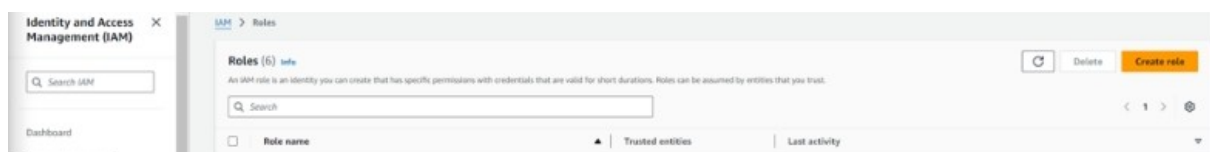
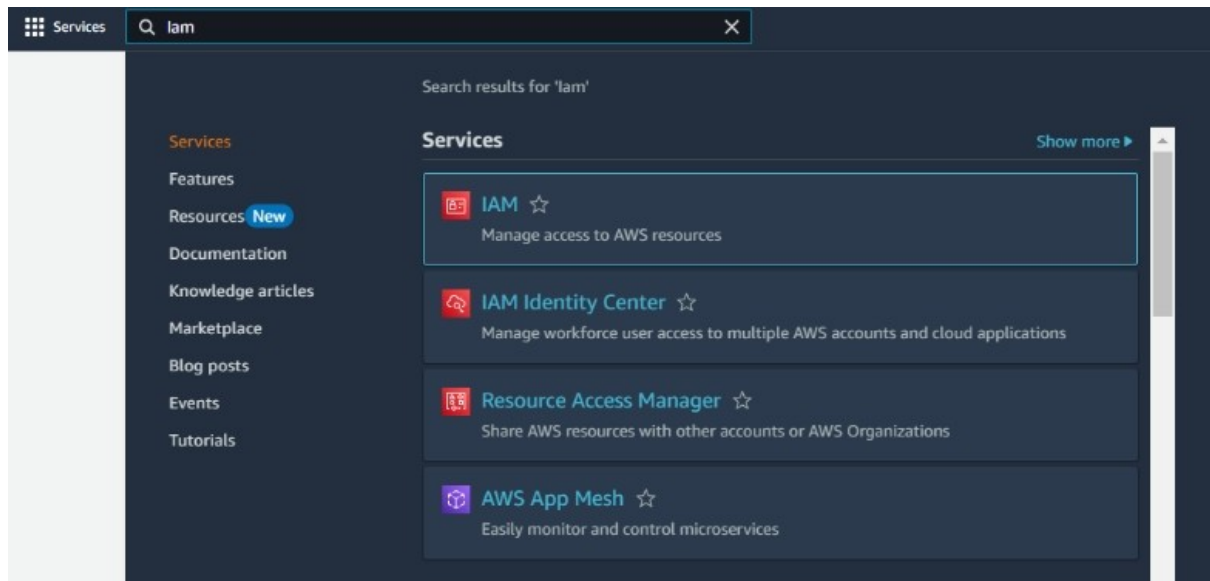


Milestone 5: IAM Role Setup

IAM (Identity and Access Management) role setup involves creating roles that define specific permissions for AWS services. To set it up, you create a role with the required policies, assign it to users or services, and ensure the role has appropriate access to resources like EC2, S3, or RDS. This allows controlled access and ensures security best practices in managing AWS resources.

Create IAM Role.

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



To create and select DynamoDBFullAccess and SNSFullAccess, go to the AWS IAM console, create a new role, and assign the respective policies. DynamoDBFullAccess allows full access to DynamoDB resources, while SNSFullAccess enables sending notifications via SNS. Attach the role to the relevant services to ensure proper integration with the project.

Select trusted entity

Trusted entity type

- ☒ **AWS account**
Allow AWS or non-AWS EC2 instance, or other to perform actions in this account.
- ☐ **AWS account**
Allow another AWS IAM account belonging to you or a federated party to perform actions in this account.
- ☐ **Web identity**
Allow an application by the specified external web identity provider to assume this role to perform actions in this account.
- ☐ **SAML 2.0 federation**
Allow an external identity provider (IdP) to perform actions in this account.
- ☐ **Custom trust policy**
Create a custom trust policy to enable others to perform actions in this account.

Use case
Allow an AWS or non-AWS EC2 instance, or other to perform actions in this account.

Service or use case:

Choose a use case for the specified service.

Use case:

- ☒ **EC2**
Allow EC2 instances to call AWS on their own behalf.
- ☐ **EC2 Role for AWS Systems Manager**
Allow EC2 instances to call AWS on their own behalf and Systems Manager on your behalf.
- ☐ **EC2 Spot Fleet Role**
Allow EC2 Spot Fleet to request and terminate spot instances on your behalf.
- ☐ **EC2 - Spot Fleet Auto Scaling**
Allow Spot Fleet to create and update EC2 Spot fleets on your behalf.
- ☐ **EC2 - Spot Fleet Tagging**
Allow EC2 to search your instances and attach tags to the specified instances on your behalf.
- ☐ **EC2 - Spot Instances**
Allow EC2 Spot instances to request and manage spot instances on your behalf.
- ☐ **EC2 - Spot Fleet**
Allow EC2 Spot Fleet to request and manage spot instances on your behalf.
- ☐ **EC2 - Spot Instance**
Allow EC2 Spot Instance to request and manage spot instances on your behalf.

Cancel **Next**

Add permissions

Permissions policies (1/955)

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

Filter by Type: 2 matches

Policy name	Type
<input checked="" type="checkbox"/> AmazonDynamoDBFullAccess	AWS managed
<input checked="" type="checkbox"/> AmazonDynamoDBReadOnlyAccess	AWS managed

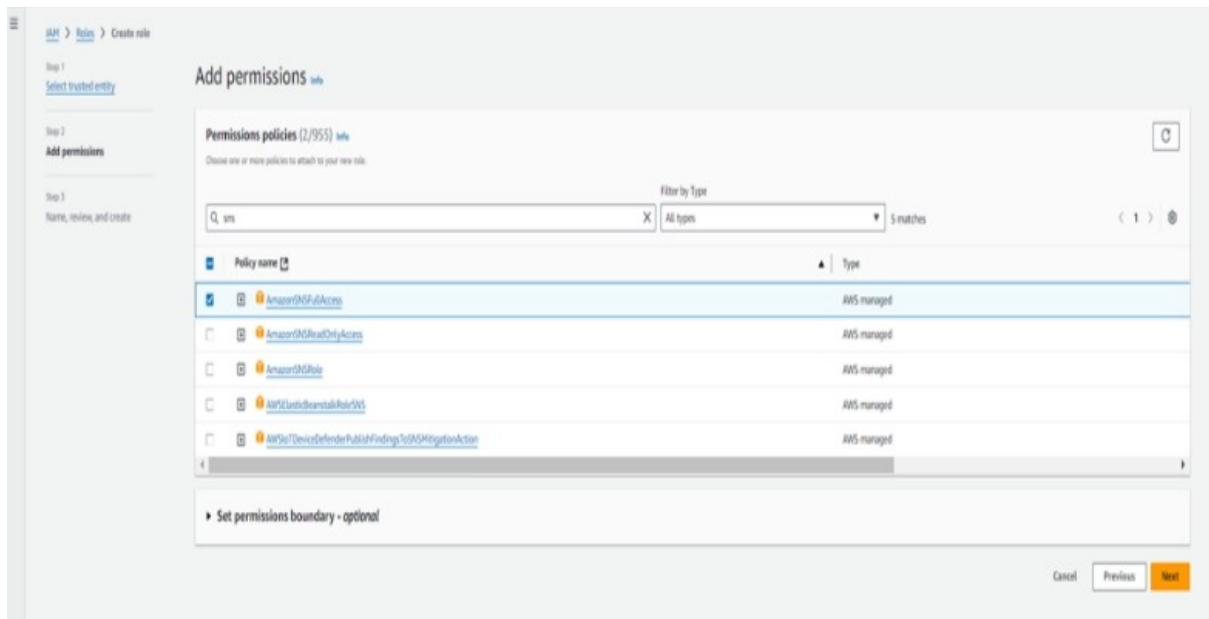
▶ Set permissions boundary - optional

Cancel Previous **Next**

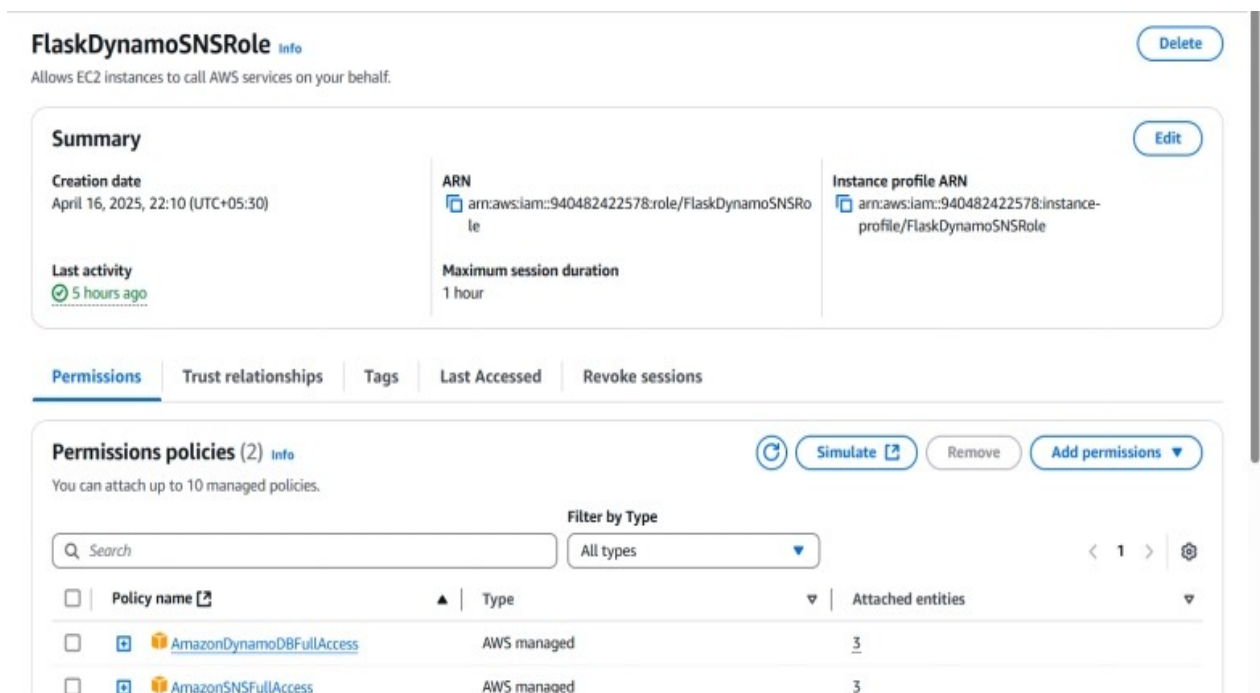
Attach Policies.

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.



To create a role named **flaskdynamodbsns**, go to the AWS IAM console, create a new role, and assign DynamoDBFullAccess and SNSFullAccess policies. Name the role flaskdynamodbsns and attach it to the necessary AWS services. This role will allow your Flask app to interact with both DynamoDB and SNS seamlessly.

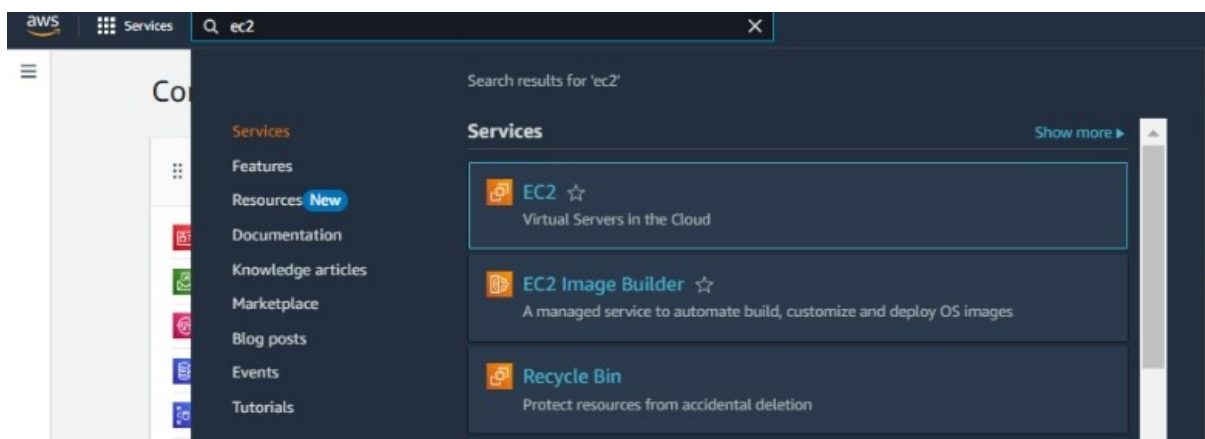


Milestone 6: EC2 Instance setup

To set up a public EC2 instance, choose an appropriate Amazon Machine Image (AMI) and instance type. Ensure the security group allows inbound traffic on necessary ports (e.g., HTTP/HTTPS for web applications). After launching the instance, associate it with an Elastic IP for consistent public access, and configure your application or services to be publicly accessible.

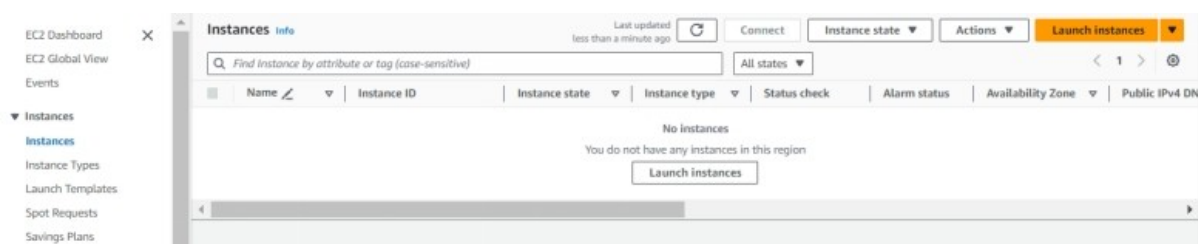
Launch an EC2 instance to host the Flask application.

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



To launch an EC2 instance with the name **flaskec2role**, follow these steps:

1. Go to the **AWS EC2 Dashboard** and click on **Launch Instance**.
2. Select your desired AMI, instance type, configure instance details, and under **IAM role**, choose the role **flaskec2role**. Finally, launch the 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

FlaskEC2Role

[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

Recents

Quick Start



[Browse more AMIs](#)

Including AMIs from
AWS, Marketplace and
the Community

To launch an EC2 instance with **Amazon Linux 2** or **Ubuntu** as the AMI and **t2.micro** as the instance type (free-tier eligible):

1. In the **Launch Instance** wizard, choose **Amazon Linux 2** or **Ubuntu** from the available AMIs.
2. Select **t2.micro** as the instance type, which is free-tier eligible, and continue with the configuration and launch steps.

Amazon Linux
aws

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

To create and download the key pair for server access:

1. In the **Launch Instance** wizard, under the **Key Pair** section, click **Create a new key pair**.
2. Name your key pair (e.g., **flaskkeypair**) and click **Download Key Pair**. This will download the .pem file to your system, which you will use to access the EC2 instance securely via SSH.

▼ Instance type Info | Get advice

Instance type

t2.micro

Family: t2 1 vCPU 1 GiB Memory Current generation: true

On-Demand Linux base pricing: 0.0124 USD per Hour

On-Demand Windows base pricing: 0.017 USD per Hour

On-Demand RHEL base pricing: 0.0268 USD per Hour

On-Demand SUSE base pricing: 0.0124 USD per Hour

Free tier eligible

☐ All generations

[Compare instance types](#)

Additional costs apply for AMIs with pre-installed software


▼ Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

Select

▼

 [Create new key pair](#)


▼ Key pair (login) Info

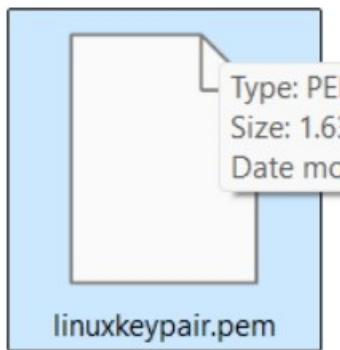
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

linuxkeypair

▼

 [Create new key pair](#)



Configure security groups for HTTP, and SSH access.

For network settings during EC2 instance launch:

1. In the **Network Settings** section, select the **VPC** and **Subnet** you wish to use (if unsure, the default VPC and subnet should work).
2. Ensure **Auto-assign Public IP** is enabled so your instance can be accessed from the internet.
3. In **Security Group**, either select an existing one or create a new one that allows SSH (port 22) access to your EC2 instance for remote login.

A screenshot of the AWS Management Console 'Network settings' section for an EC2 instance launch. The section is titled 'Network settings' with an 'Info' link. It contains several configuration options: 'VPC - required' with a dropdown menu showing 'vpc-03cdc7b6f19dd7211' and '(default)'; 'Subnet' with a dropdown menu showing 'No preference' and a 'Create new subnet' link; 'Auto-assign public IP' with a dropdown menu showing 'Enable'; 'Firewall (security groups)' with two radio buttons: 'Create security group' (selected) and 'Select existing security group'; 'Security group name - required' with a text input field containing 'launch-wizard'; and 'Description - required' with a text input field containing 'launch-wizard created 2024-10-13T17:49:56.622Z'. A note states: 'Additional charges apply when outside of free tier allowance'. Another note states: 'This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and _-./()#,@[]+=&:;!\$*'. The 'Info' link is also present next to the description field.

Inbound Security Group Rules

Security group rule 1 (TCP, 22, 0.0.0.0/0)
Remove

Type ssh

Source type Anywhere

Protocol TCP

Source 0.0.0.0/0

Port range 22

Description - optional e.g. SSH for admin desktop

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

Type HTTP

Source type Custom

Protocol TCP

Source 0.0.0.0/0

Port range 80

Description - optional e.g. SSH for admin desktop

Security group rule 3 (TCP, 5000, 0.0.0.0/0) Remove

Type Custom TCP

Source type Custom

Protocol TCP

Source 0.0.0.0/0

Port range 5000

Description - optional e.g. SSH for admin desktop

Add security group rule

[EC2](#) > [Launch an instance](#)

Success

Successfully initiated launch of instance i-001861028fwac290

[Launch log](#)

Next Steps

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

Create billing and free tier usage alerts

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

Create billing alerts

Connect to your instance

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

Connect to instance
Learn more

Connect an RDS database

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

Connect an RDS database
Create a new RDS database
Learn more

Create EBS snapshot policy

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

Create EBS snapshot policy

Manage detailed monitoring

Enable or disable detailed monitoring for the instance. If you enable detailed monitoring, the Amazon EC2 console displays monitoring graphs with a 1-minute period.

Manage detailed monitoring

Create Load Balancer

Create an application, network gateway or classic Elastic Load Balancer.

Create Load Balancer

Create AWS budget

AWS Budgets allow you to create budgets, forecast spend, and take action on your costs and usage from a single location.

Create AWS budget

Manage CloudWatch alarms

Create or update Amazon CloudWatch alarms for the instance.

Manage CloudWatch alarms

Disaster recovery for your instances

Recover the instances you just launched into a different Availability Zone or a different Region using AWS Elastic Disaster Recovery (EDR).

Disaster recovery for your instances

Monitor for suspicious runtime activities

Amazon GuardDuty enables you to continuously monitor for malicious runtime activity and unauthorized behaviors, with near real-time visibility into on-host activities occurring across your Amazon EC2 workload.

Monitor for suspicious runtime activities

Get instance screenshot

Capture a screenshot from the instance and view it as an image. This is useful for troubleshooting an unresponsive instance.

Get instance screenshot

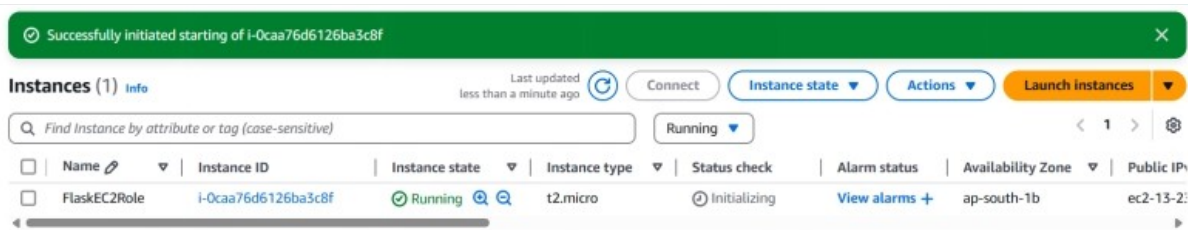
Get system log

View the instance's system log to troubleshoot issues.

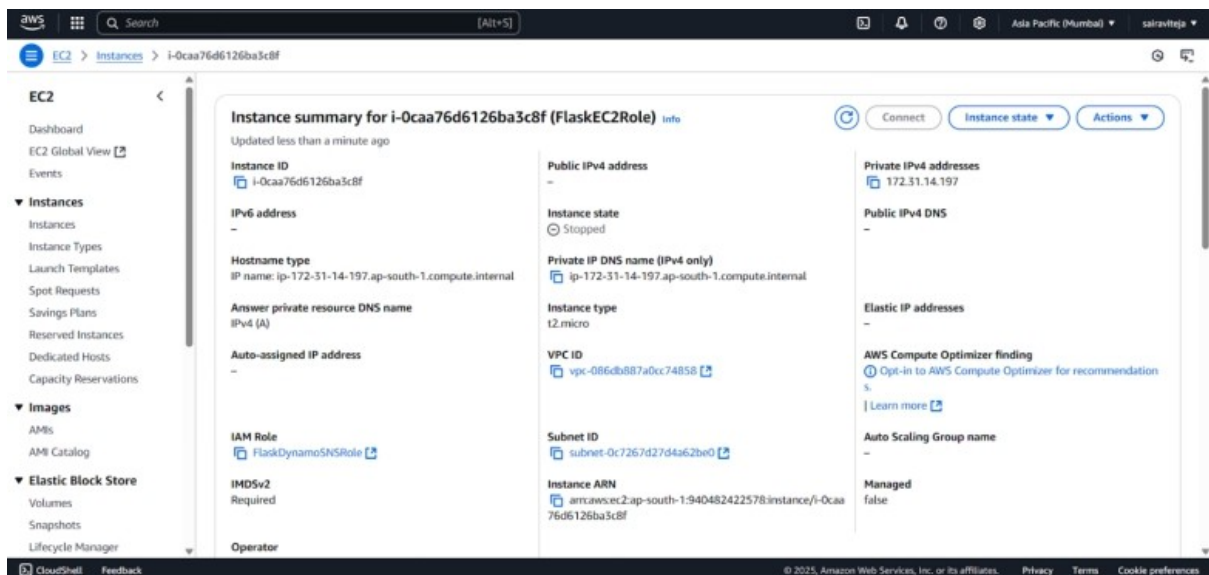
Get system log

[View all instances](#)

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

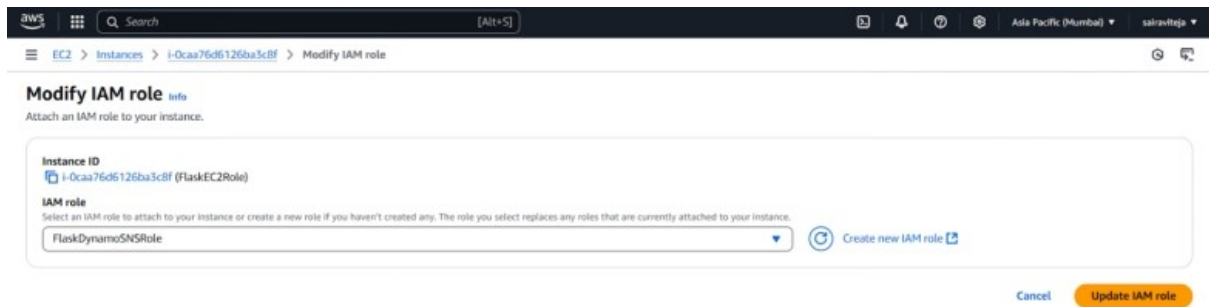


- The EC2 instance you are launching is configured with Amazon Linux 2 or Ubuntu as the AMI, t2.micro as the instance type (free-tier eligible), and flaskec2role IAM role for appropriate permissions. The flaskkeypair key pair is created for secure server access via SSH, and the instance is set to auto-assign a public IP for internet accessibility. The security group is configured to allow SSH (port 22) access for remote login.



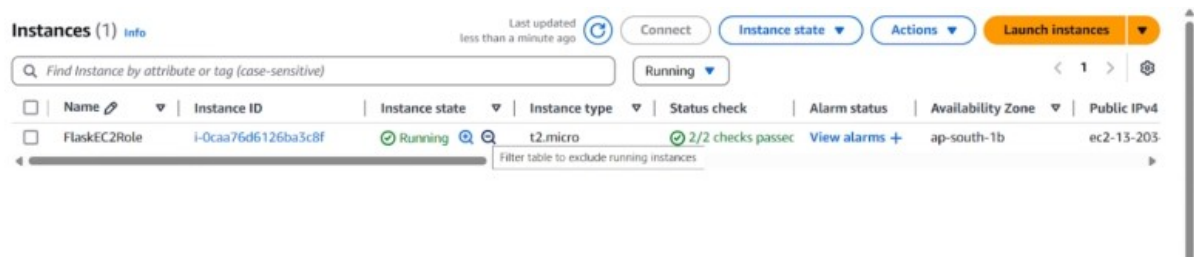
To modify the **IAM role** for your EC2 instance:

1. Go to the **AWS IAM Console**, select **Roles**, and find the **flaskec2role**.
2. Click **Attach Policies**, then choose the required policies (e.g., **DynamoDBFullAccess**, **SNSFullAccess**) and click **Attach Policy**.
3. If needed, update the instance to use this modified role by selecting the EC2 instance, clicking **Actions**, then **Security**, and **Modify IAM role** to select the updated role.

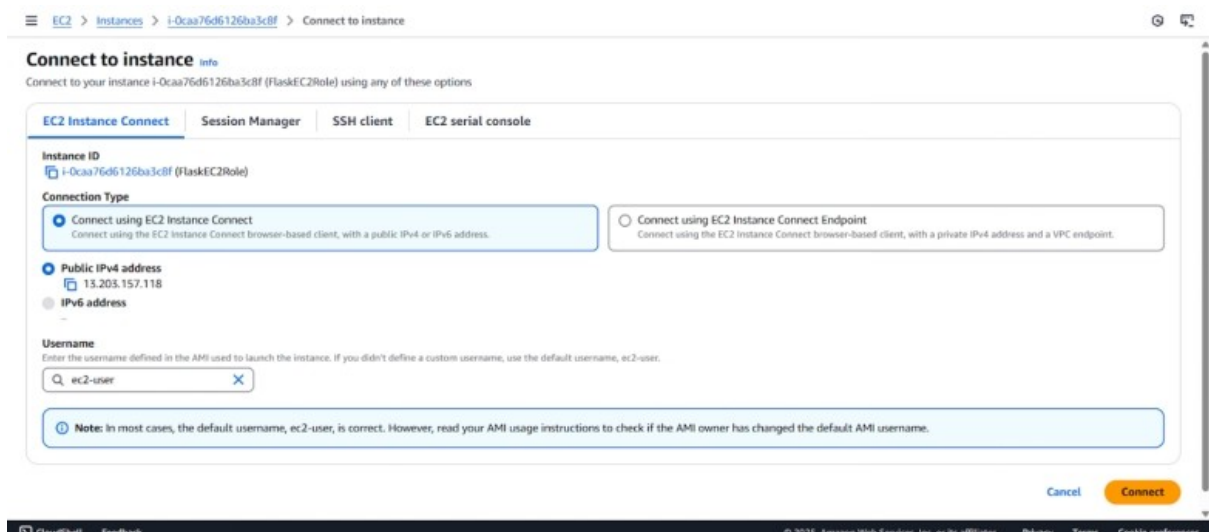


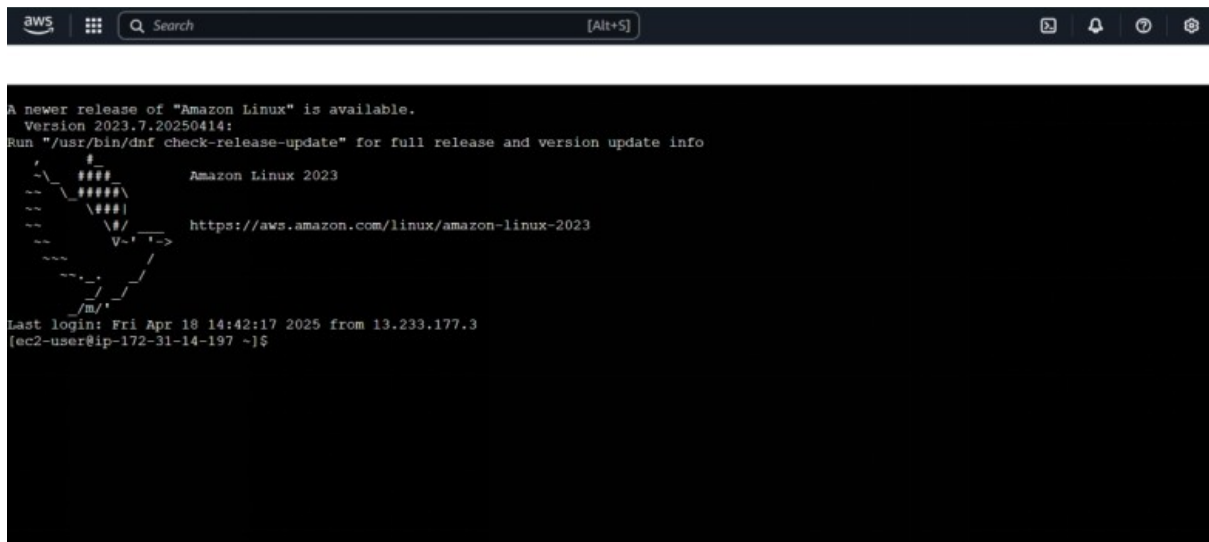
To connect to your EC2 instance:

1. Go to the **EC2 Dashboard**, select your running instance, and click **Connect**.
2. Follow the instructions provided in the **Connect To Your Instance** dialog, which will show the SSH command (e.g., `ssh -i flaskkeypair.pem ec2-user@<public-ip>`) to access your instance using the downloaded .pem key.



- Now connect the EC2 with the files



A screenshot of an AWS console terminal window. The terminal displays the Amazon Linux 2023 logo, which consists of a stylized 'A' made of hash symbols. Below the logo, it says 'Amazon Linux 2023' and provides a URL: 'https://aws.amazon.com/linux/amazon-linux-2023'. At the bottom of the terminal, it shows the last login time: 'Last login: Fri Apr 18 14:42:17 2025 from 13.233.177.3' and the prompt '(ec2-user@ip-172-31-14-197 ~)\$'. The terminal is running on an EC2 instance with the ID 'i-0caa76d6126ba3c8f' and role 'FlaskEC2Role'. The public IP is '13.203.157.118' and the private IP is '172.31.14.197'.

i-0caa76d6126ba3c8f (FlaskEC2Role)
PublicIPs: 13.203.157.118 PrivateIPs: 172.31.14.197

Milestone 7 : Deployment on EC2

Deployment on an EC2 instance involves launching a server, configuring security groups for public access, and uploading your application files. After setting up necessary dependencies and environment variables, start your application and ensure it's running on the correct port. Finally, bind your domain or use the public IP to make the application accessible online.

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
```

Clone Your Flask Project from GitHub

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

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

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

here: 'git clone https://github.com/Ravi-teja-777/medtrack_app.git

- This will download your project to the EC2 instance.

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

```
cd Medtrack
```

Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:

Run the Flask Application

```
sudo flask run --host=0.0.0.0 --port=5000
```

```
aws [Alt+S] Search [Alt+S] Asia Pacific (Mumbai) sairaviteja
2025-04-18 08:02:36,630 - _main_ - ERROR - Failed to fetch appointments: An error occurred (ValidationException) when calling the Query operation: The table does not have the specified index: DoctorEmailIndex
2025-04-18 08:02:36,637 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:02:36] "GET /dashboard HTTP/1.1" 200 -
2025-04-18 08:02:39,258 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:02:39] "GET /logout HTTP/1.1" 302 -
2025-04-18 08:02:39,289 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:02:39] "GET / HTTP/1.1" 200 -
2025-04-18 08:02:41,589 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:02:41] "GET /login HTTP/1.1" 200 -
2025-04-18 08:02:47,468 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:02:47] "POST /login HTTP/1.1" 302 -
2025-04-18 08:02:47,500 - _main_ - ERROR - Failed to query appointments: An error occurred (ValidationException) when calling the Query operation: The table does not have the specified index: PatientEmailIndex
2025-04-18 08:02:47,517 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:02:47] "GET /dashboard HTTP/1.1" 200 -
2025-04-18 08:02:49,506 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:02:49] "GET /book_appointment HTTP/1.1" 200 -
2025-04-18 08:03:25,936 - _main_ - INFO - Email sent to gtharunasri19@gmail.com
2025-04-18 08:03:28,795 - _main_ - INFO - Email sent to sairaviteja478@gmail.com
2025-04-18 08:03:28,796 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:03:28] "POST /book_appointment HTTP/1.1" 302 -
2025-04-18 08:03:28,852 - _main_ - ERROR - Failed to query appointments: An error occurred (ValidationException) when calling the Query operation: The table does not have the specified index: PatientEmailIndex
2025-04-18 08:03:28,868 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:03:28] "GET /dashboard HTTP/1.1" 200 -
2025-04-18 08:03:44,609 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:03:44] "GET /logout HTTP/1.1" 302 -
2025-04-18 08:03:44,654 - werkzeug - INFO - 110.235.236.48 - - [18/Apr/2025 08:03:44] "GET / HTTP/1.1" 200 -
* Serving Flask app 'app'
* Debug mode: off
2025-04-18 08:04:27,745 - werkzeug - INFO - WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.31.14.197:5000
2025-04-18 08:04:27,746 - werkzeug - INFO - Press CTRL+C to quit

i-0caa76d6126ba3c8f (FlaskEC2Role)
PublicIPs: 13.203.227.15 PrivateIPs: 172.31.14.197
```

Verify the Flask app is running:

<http://your-ec2-public-ip>

- Run the Flask app on the EC2 instance

```
[ec2-user@ip-172-31-3-5 InstantLibrary]$ sudo flask run --host=0.0.0.0 --port=80
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:80
* Running on http://172.31.3.5:80
Press CTRL+C to quit
183.82.125.56 - - [22/Oct/2024 07:42:00] "GET / HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /register HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /static/images/library3.jpg HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /favicon.ico HTTP/1.1" 404 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /static/images/library3.jpg HTTP/1.1" 304 -
183.82.125.56 - - [22/Oct/2024 07:42:21] "POST /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:24] "GET /login HTTP/1.1" 200 -
183.82.125.56 - - [22/Oct/2024 07:42:27] "POST /login HTTP/1.1" 302 -
183.82.125.56 - - [22/Oct/2024 07:42:28] "GET /home-page HTTP/1.1" 200 -
```

Access the website through:

PublicIPs: <https://13.201>

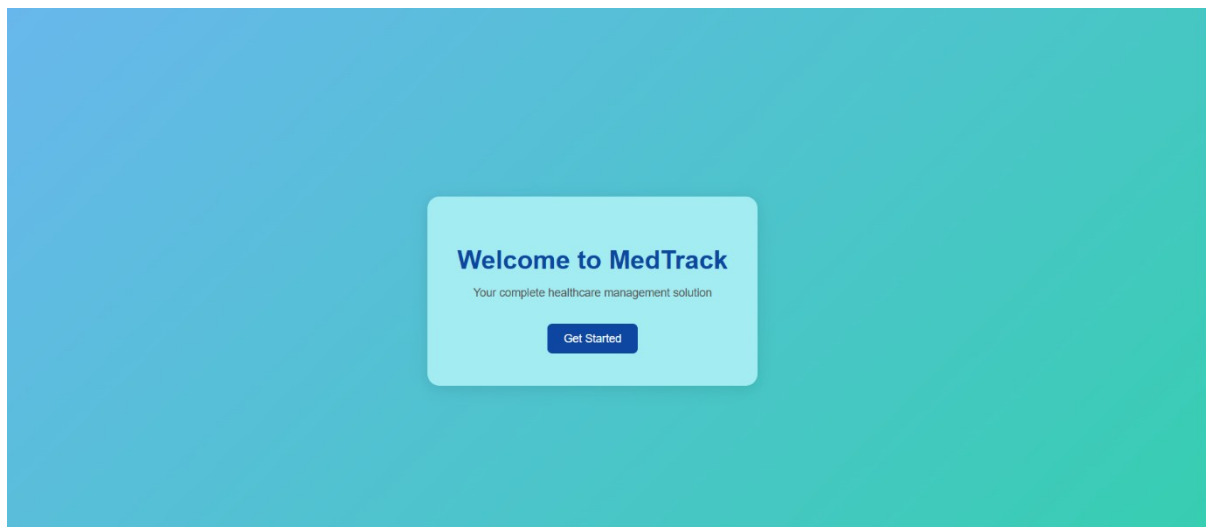
Milestone 8: Testing and Deployment

Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and perform a final round of live testing to confirm everything runs smoothly in the production environment.

Functional testing to verify the project

Index Page

The index page of the MedTrack: Healthcare Management System serves as the landing page for the application. It welcomes users and directs them to start using the system via a clean and modern interface.



Sign up page

The Sign Up page allows new users to create an account on the MedTrack healthcare management platform. It features a clean, professional design with a soft gradient background that matches the visual style of the home (index) page.

The image shows a 'Create Account' form centered on a page with a light blue background. The form is white with a thin grey border. It contains the following fields and elements:

- Create Account** (Section Header)
- Full Name:** Text input field
- Username:** Text input field
- Email:** Text input field
- Password:** Text input field
- Confirm Password:** Text input field
- Role:** Dropdown menu with the text '-- Select Role --' and a downward arrow.
- Sign Up** (Blue button)
- Below the button: [Already have an account? Login here](#) and [← Back to Home](#)

The bottom of the image shows a Windows taskbar with various application icons including the Start menu, Search, File Explorer, and several web browsers.

PATIENT LOGIN PAGE:

The Patient Login Page allow users to securely access their accounts on the platform. Each login page typically includes:

1. **Username and Password Fields:** Users enter their credentials (username and password) to authenticate their account.
2. **Login Button:** A button to submit login details and validate user access.

Once logged in, patients redirected to their respective dashboards to manage appointments, medical records, and other relevant tasks.

Login

Username:

Password:

Login

Don't have an account? [Sign Up](#)

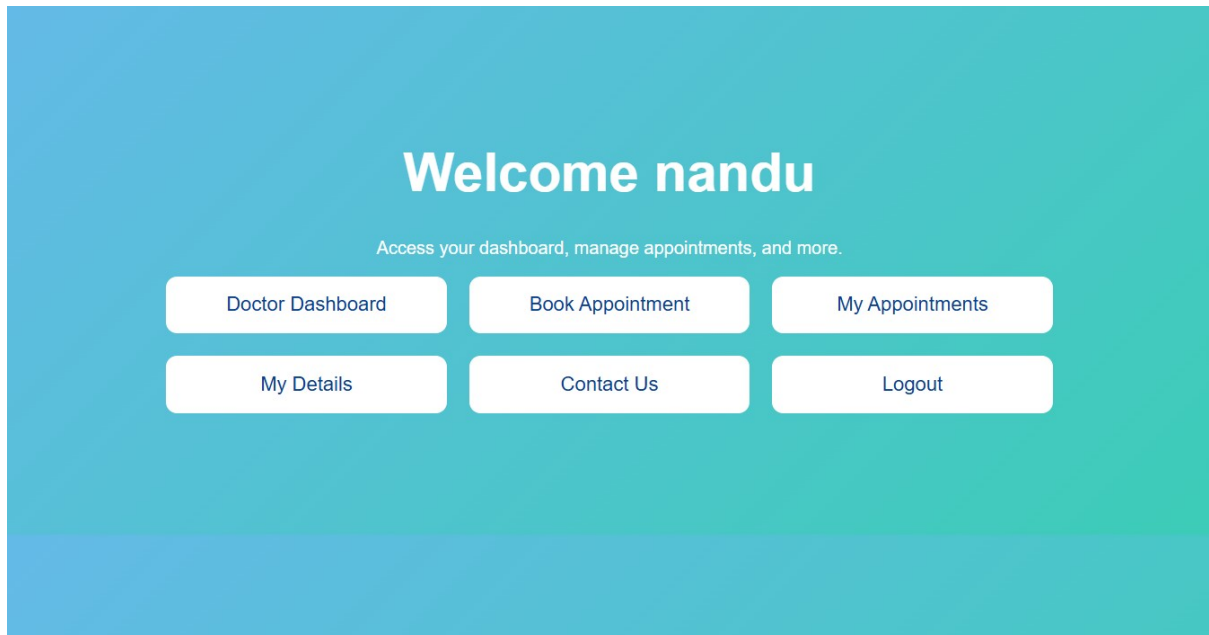
Home Page

The Home Page of your project is the main entry point for users, where they can interact with the system. It typically includes:

- **Input Fields:** For users to enter basic information like appointment requests, diagnosis submissions, or service bookings.
 1. **Navigation:** Links to other sections such as the login page, dashboard, or service options.

Responsive Design: Ensures the page is accessible across devices with a clean, user-friendly interface.

- The Home Page serves as the initial interface that directs users to the key functionalities of your web application
- provides an easy interface to manage appointments and track their status. It typically includes:



- Book Appointment Section: A form for selecting a doctor, choosing an appointment time, and submitting the request.
- Appointment Status: A section showing the current status of appointments (e.g., confirmed, pending, or completed) with options to view details or cancel.
- Upcoming Appointments: A list of future appointments with relevant details such as doctor name, date, and time.
- This dashboard helps patients book new appointments and keep track of their healthcare schedules.

Book Appointment

Login successful.

Patient Name:

Select Doctor:

--Choose Doctor--

Date:

dd-mm-yyyy

Time:

--:--

Reason for Visit:

Describe your issue...

Book Appointment

← Back to Home

My Appointments

Patient	Doctor	Date	Time
nandu	Dr. Johnson	2025-07-07	01:00

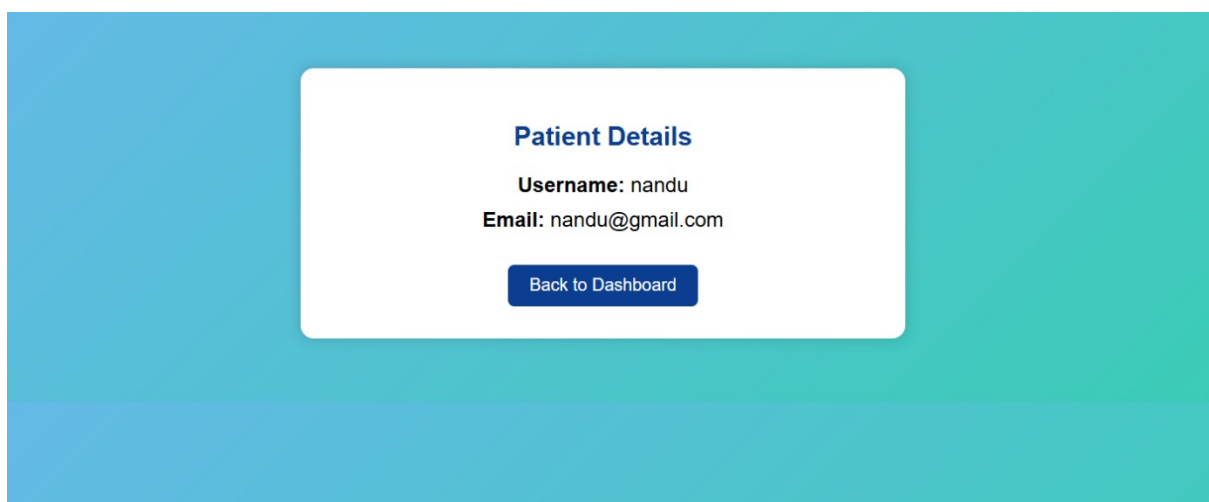
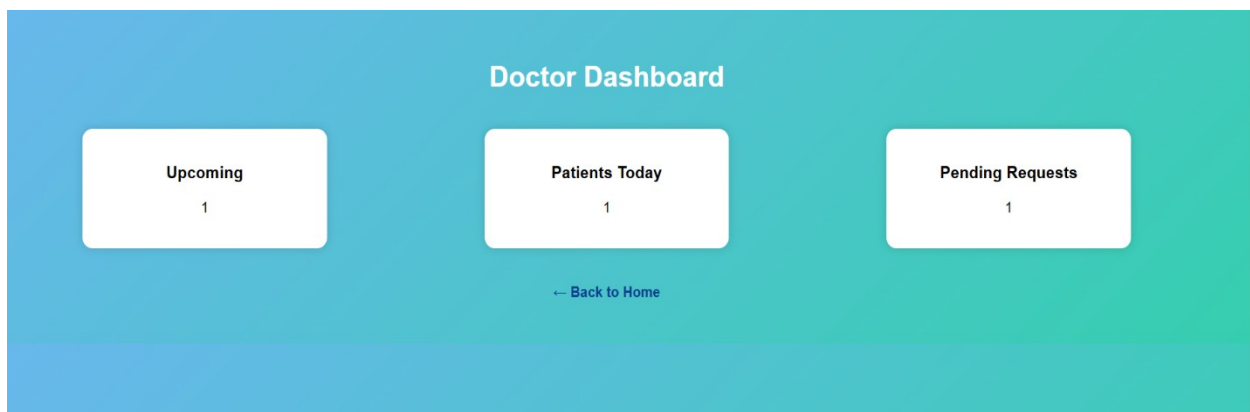
← Back to Home

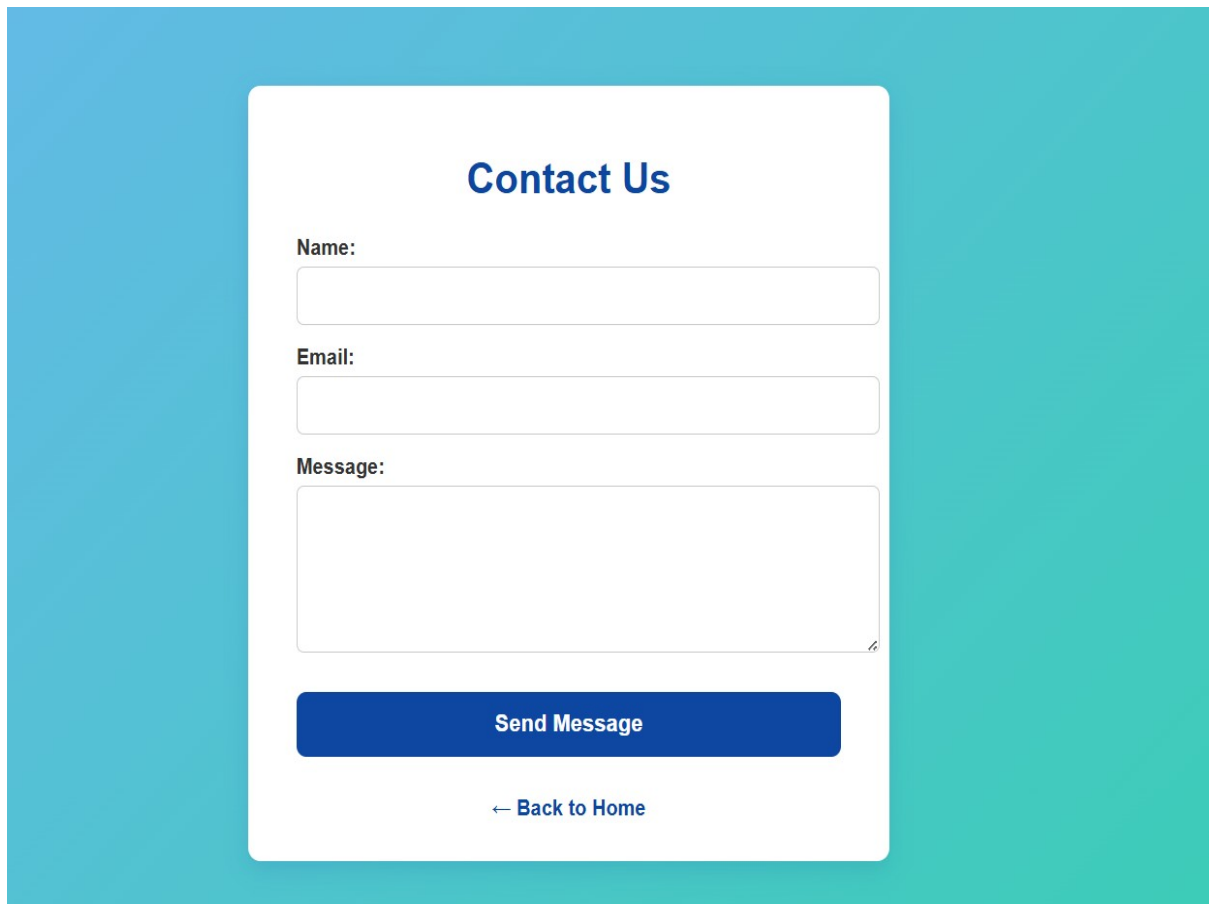
Doctor Dashboard:

The **Doctor Dashboard** provides doctors with a comprehensive view of their upcoming appointments and patient details. It typically includes:

- **Upcoming Appointments List:** A table or list showing patient names, appointment times, and appointment statuses (e.g., confirmed, pending).
- **Patient Details:** Quick access to each patient's medical history, contact information, and previous visit records.
- **Appointment Actions:** Options to view, confirm, reschedule, or cancel appointments, ensuring efficient management.

The dashboard serves as the main interface for doctors to manage their schedules, track patient interactions, and provide timely care.



A contact form titled "Contact Us" is centered on a teal background. The form is white with rounded corners and contains three input fields: "Name:", "Email:", and "Message:". Below these fields is a blue "Send Message" button and a link that says "← Back to Home".

Contact Us

Name:

Email:

Message:

Send Message

[← Back to Home](#)

DynamoDB Database updations :

1. Users table :

In the Users Table of DynamoDB, the data structure is designed to store user-related information for both patients and doctors. Typical updates include:

1. **Add New Users:** When a new patient or doctor registers, their details such as name, email, role (patient/doctor), contact info, and password hash are added to the table.
2. **Update User Info:** If a user updates their profile (e.g., changing contact details), the corresponding record in the table is modified.
3. **Status Tracking:** Track the status of user accounts (active, inactive) based on their activity or admin updates.

The Users Table serves as the central repository for all user data, enabling quick access and modification of details when necessary.

Table: UsersTable - Items returned (4) Actions Create item

Scan started on April 18, 2025, 19:27:34

<input type="checkbox"/>	email (String)	age	created_at	gender	login_count	name
<input type="checkbox"/>	sunder@thesmartbrid...	25	2025-04-18...	male	2	sunder
<input type="checkbox"/>	gtharunasri19@gmail...	21	2025-04-18...	female	1	tharuna
<input type="checkbox"/>	sairaviteja478@gmail...	21	2025-04-18...	male	2	Kambhan
<input type="checkbox"/>	siri@thesmartbridge.c...	24	2025-04-18...	female	1	siri

0. Appointment table :

In the Appointment Table of DynamoDB, the data structure stores information related to patient appointments. Typical updates include:

1. Add New Appointment: When a patient books an appointment, details such as patient ID, doctor ID, appointment date, time, and status (pending, confirmed, canceled) are stored.
2. Update Appointment Status: As appointments are confirmed, rescheduled, or canceled, the status field in the table is updated accordingly.
3. Appointment History: Historical data about completed appointments can also be stored to track past interactions between patients and doctors.

The Appointment Table allows for efficient management of appointments, ensuring accurate and up-to-date scheduling information for both doctors and patients.

Table: AppointmentsTable - Items returned (2)

Scan started on April 18, 2025, 19:27:09

Actions

Create item

< 1 >

<input type="checkbox"/>	appointment_id (String)	appointment_date	created_at	diagnosis	doctor_email
<input type="checkbox"/>	86d52cfe-cce6-46b0-941d-1...	2025-04-30	2025-04-18...		gtharunasri19...
<input type="checkbox"/>	42e13fef-fbe9-42d1-9905-f...	2025-04-21	2025-04-18...	make sure t...	siri@thesmart...

Appointment confirmation:

Book Appointment

Login successful.

Patient Name:

Select Doctor:

--Choose Doctor--

Date:

dd-mm-yyyy

Time:

--:--

Reason for Visit:

Describe your issue...

Book Appointment

← Back to Home

My Appointments

Patient	Doctor	Date	Time
nandu	Dr. Johnson	2025-07-07	01:00

← Back to Home

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.

