

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

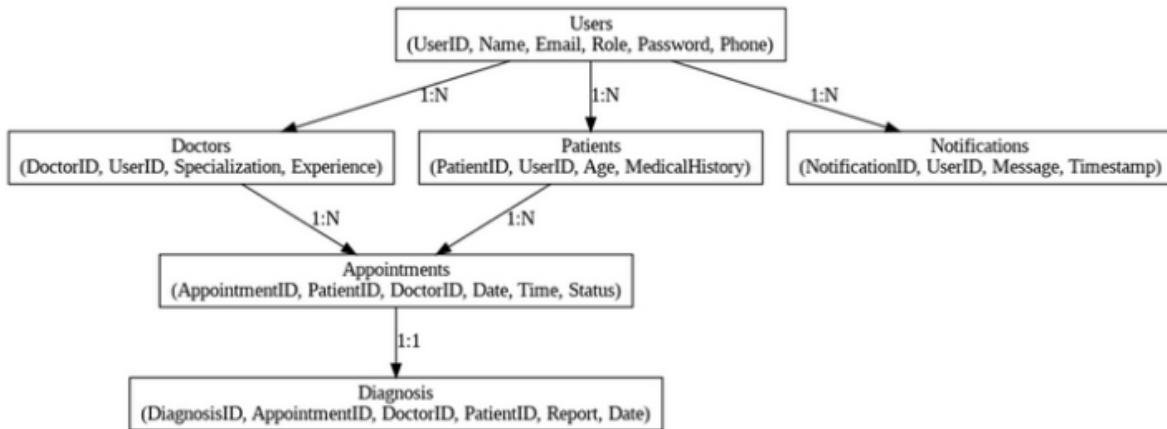
MedTrack utilizes AWS IAM to manage user permissions and ensure secure access to the system. For instance, when a new patient registers, an IAM user is created with specific roles and permissions to access only the features relevant to them. Doctors have their own IAM configurations, allowing them access 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-tJ6lgWtRbawWoA4W3PkxEFrSJZhKzULgLsjxo/edit?usp=sharing>

Important Instructions:

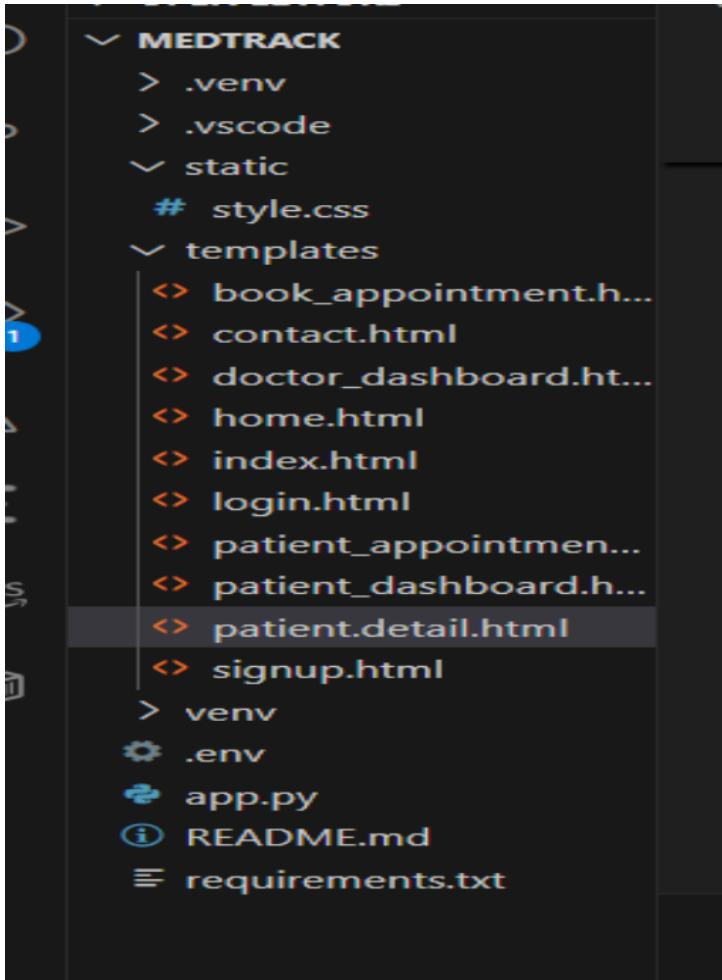
- Start by creating the necessary HTML pages and Flask routes (app.py) to build the core functionality of your application.
- During the initial development phase, store and retrieve data using Python dictionaries or lists locally. This will allow you to design, test, and validate your application logic 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

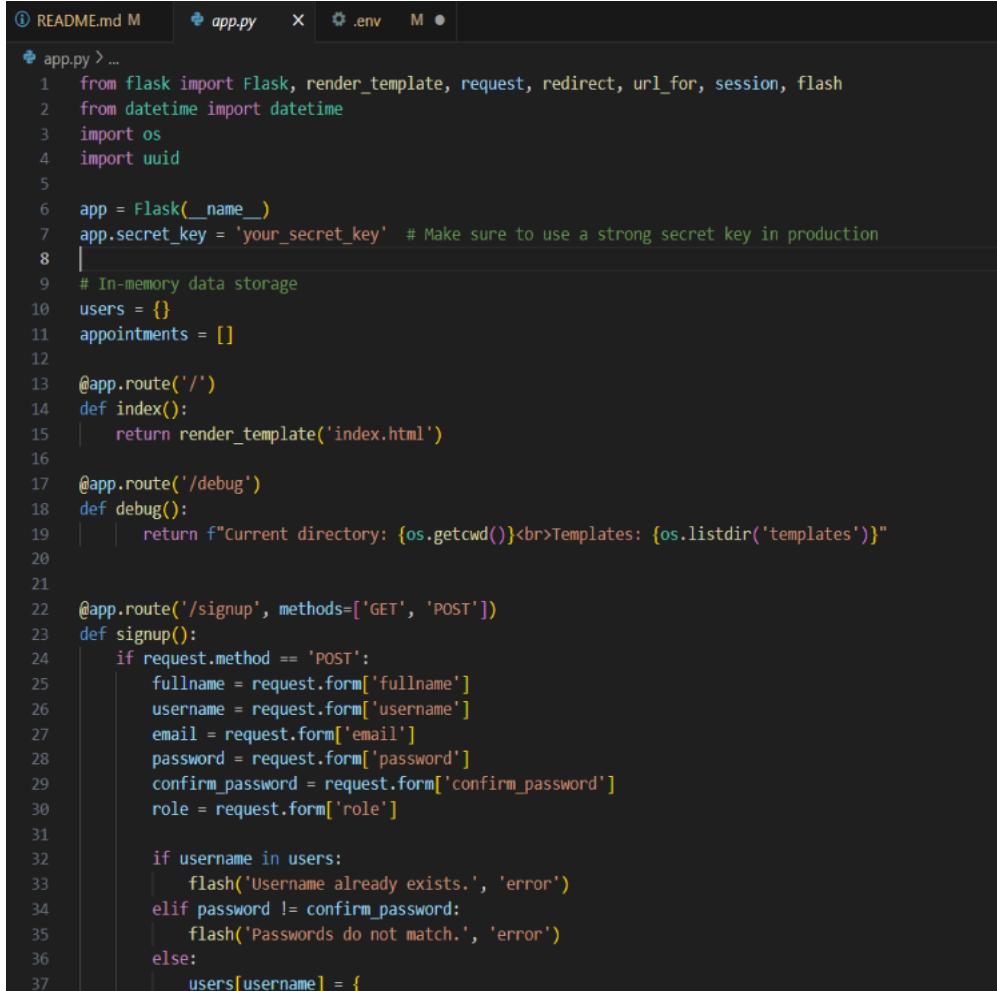


Description of the code :

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.



The screenshot shows a code editor with three tabs: README.md, app.py, and .env. The app.py tab is active and displays Python code for a Flask application. The code includes imports for Flask, render_template, request, redirect, url_for, session, and flash from the flask module. It also imports datetime from the datetime module, os from the os module, and uuid from the uuid module. The app is initialized with app = Flask(__name__). The secret key is set to 'your_secret_key'. A comment notes to use a strong secret key in production. An environment variable .env is mentioned. The code defines two lists: users and appointments. It contains routes for the root ('/'), a debug endpoint ('/debug') which returns the current directory and template list, and a signup endpoint ('/signup'). The signup logic handles POST requests by extracting form data for fullname, username, email, password, confirm_password, and role. It checks if the username already exists, ensures the password matches the confirmation, and then adds the new user to the users dictionary.

```
① README.md M ② app.py X ③ .env M ●

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()}  
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 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.

```
app.py M X .env U app.log U base.html view_appointment_doctor.html
app.py > ...
71     def get_user_role(email):
72         ...
73         except Exception as e:
74             logger.error(f"Error fetching role: {e}")
75             return None
76
77
78     def send_email(to_email, subject, body):
79         if not ENABLE_EMAIL:
80             logger.info(f"[Email Skipped] Subject: {subject} to {to_email}")
81             return
82
83
84         try:
85             msg = MIMEText(body)
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(
```

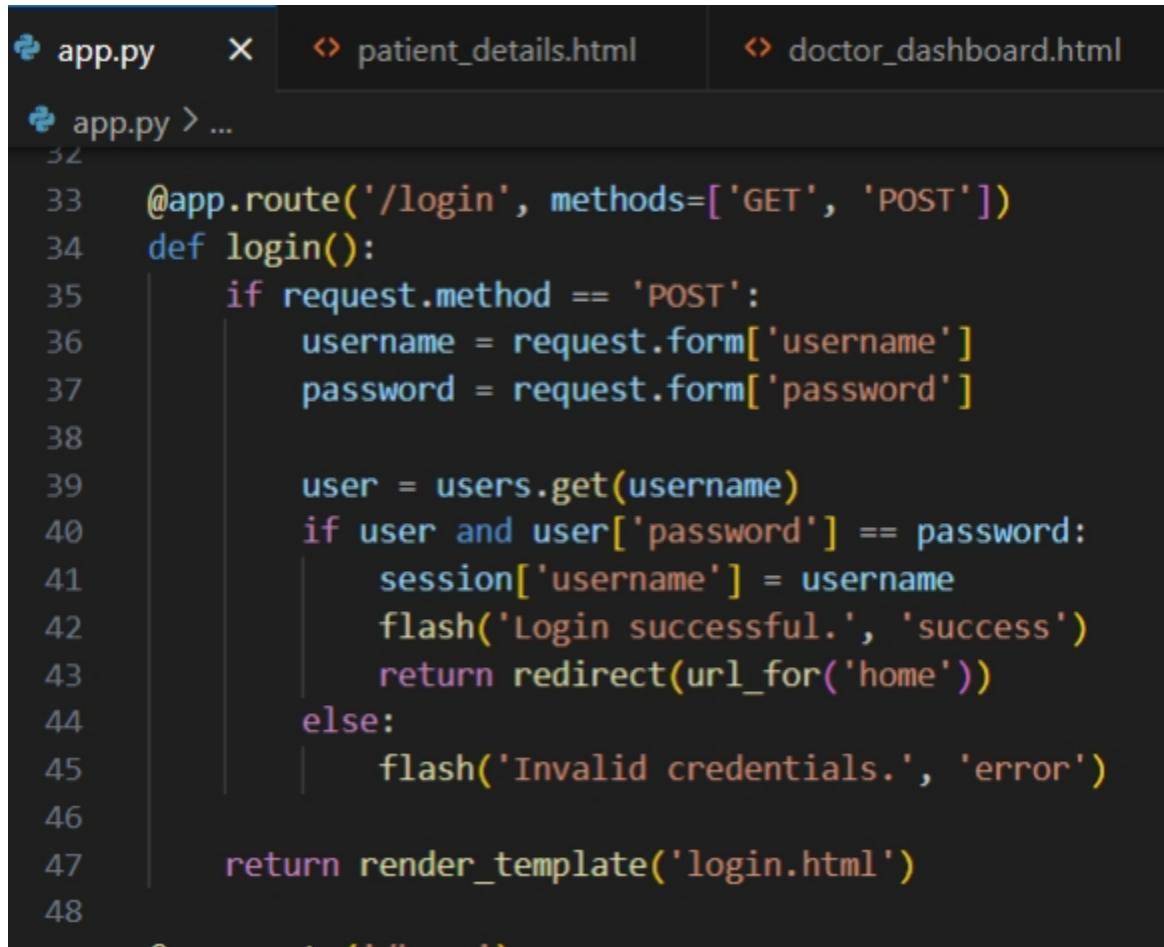
- **Routes for Web Pages:**

Register Page

The screenshot shows a code editor with several tabs at the top: README.md, app.py (which is the active tab), .env, and M. The code in the app.py tab is as follows:

```
app.py > debug
app: /home/username/PycharmProjects/ProjectName/app.py - L1
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'])
49
```

- 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.



The screenshot shows a code editor interface with three tabs. The active tab is 'app.py', which contains Python code for a login route. The other tabs are 'patient_details.html' and 'doctor_dashboard.html'. The code in 'app.py' is as follows:

```
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.

```
wwwelcome    app.py    patient_details.html    doctor_dashboard.html    patient_dashboard.html
app.py > ...
53     def patient_details():
54
55         if not user:
56             flash('User not found.', 'error')
57             return redirect(url_for('login'))
58
59     return render_template('patient_details.html', username=username, email=user['email'])
60
61
62
63
64
65
66
67
68
69     @app.route('/logout')
70     def logout():
71         session.pop('username', None)
72         flash('Logged out successfully.', 'info')
73         return redirect(url_for('login'))
74
```

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.

The screenshot shows a code editor with multiple tabs at the top: 'Welcome', 'app.py', 'patient_details.html', and 'doctor_dashboard.html'. The 'app.py' tab is active, displaying Python code for a Flask application. The code defines a route '/book_appointment' that handles both GET and POST requests. It checks if the user is logged in (session['username']). If not, it flashes a message and redirects to the login page. If the request is POST, it extracts form data for patient name, doctor, date, time, and reason, adds it to a list of appointments, and then flashes a success message before redirecting back to the booking page.

```
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.

```

❶ Welcome    ❁ app.py    ✎ patient_details.html    ❁ doctor_dashboard.html    ❁ patient_dashboard.html
❷ 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 if __name__ == '__main__':
176     app.run(debug=True)

```

AWS - Local Deployment Sample Code - Google Docs..

No description..

<https://docs.google.com/document/d/1sFF7-tJ6lgWtRbawWoA4W3PkxEFrSJZhKzULgLsjxo/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

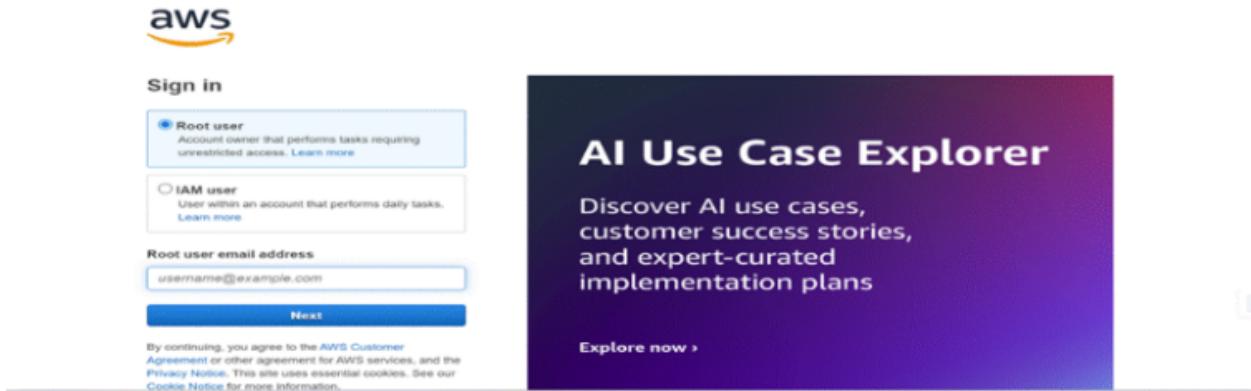
**This is for your understanding only, please refrain from creating an AWS account.
A temporary account will be provided via Troven.**

- 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.

The screenshot shows the AWS registration sign-up page. At the top, there's a navigation bar with links for About AWS, Contact Us, Support, English, My Account, Sign In, and Complete Sign Up. Below the navigation is a main heading "Complete your AWS registration". A sub-headline states: "Millions of customers are using AWS cloud solutions to build applications with increased flexibility, scalability, security, and reliability". There are three call-to-action buttons: "Complete sign-up", "AWS Free Tier" (with a sub-note about EC2, S3, etc.), and "Customer success stories" (with a sub-note about solving challenges). On the right, there's a "Contact us" section with a "Reach out to us for any AWS related questions" link and a yellow speech bubble icon.

The screenshot shows the AWS sign-up form. It features the AWS logo at the top. The form has two main sections: "Sign up for AWS" and "Explore Free Tier products with a new AWS account". The "Sign up for AWS" section includes fields for "Root user email address" (with a note about account recovery and a link to the AWS Privacy Notice) and "AWS account name" (with a note about choosing a name that can be changed later). Below these are buttons for "Verify email address" (highlighted in orange) and "Sign in to an existing AWS account". The "Explore Free Tier products with a new AWS account" section includes a note about using EC2, S3, and more for free for a year, and a link to learn more at aws.amazon.com/free. There are also decorative illustrations of clouds and storage units on either side of the form.

- 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

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.

AWS Services Search results for 'dyn'

Search results for 'dyn'

Services

Features

Resources New

Documentation

Knowledge articles

Marketplace

Blog posts

Events

Tutorials

DynamoDB Managed NoSQL Database

Amazon DocumentDB Fully-managed MongoDB-compatible database service

CloudFront Global Content Delivery Network

Athena Serverless interactive analytics service

Features

Show more ▶

Settings

DynamoDB feature

Clusters

DynamoDB feature

This screenshot shows the AWS Services search results for the query 'dyn'. The results are categorized into 'Services', 'Features', and 'Clusters'. Under 'Services', there are links to DynamoDB (Managed NoSQL Database), Amazon DocumentDB (Fully-managed MongoDB-compatible database service), CloudFront (Global Content Delivery Network), and Athena (Serverless interactive analytics service). Under 'Features', there is a link to 'DynamoDB feature'. Under 'Clusters', there is also a link to 'DynamoDB feature'. On the left sidebar, there is a navigation menu with links to 'Tables', 'Explore items', 'PartiQL editor', 'Backups', 'Exports to S3', 'Imports from S3', 'Integrations New', 'Reserved capacity', 'Settings', 'DAX Clusters', 'Subnet groups', 'Parameter groups', and 'Events'.

DynamoDB Dashboard

Dashboard

Tables

Explore items

PartiQL editor

Backups

Exports to S3

Imports from S3

Integrations New

Reserved capacity

Settings

DAX Clusters

Subnet groups

Parameter groups

Events

Alarms (0) [Info](#) Manage in CloudWatch [Logs](#)

Find alarms

Alarm name Status

No custom alarms

DAX clusters (0) [Info](#) View details

Find clusters

Cluster name Status

No clusters

No clusters to display

Create cluster

Create resources

Create table

Amazon DynamoDB Accelerator (DAX) is a fully-managed, highly-available, in-memory caching service for DynamoDB. [Learn more](#)

Create DAX cluster

What's new

SEP 10 AWS Cost Management now provides purchase recommendations for Amazon DynamoDB...

This screenshot shows the DynamoDB Dashboard. It includes sections for 'Alarms' (0) and 'DAX clusters' (0). The 'Alarms' section has a 'Manage in CloudWatch Logs' button and a search bar. The 'DAX clusters' section has a 'View details' button and a search bar. On the right side, there is a 'Create resources' sidebar with buttons for 'Create table' and 'Create DAX cluster'. Below the sidebar, there is a 'What's new' section with a message about AWS Cost Management providing purchase recommendations for Amazon DynamoDB.

The screenshot shows the AWS DynamoDB management console. On the left, there's a sidebar with links like Dashboard, Tables (selected), Explore items, PartQL editor, Backups, Exports to S3, Imports from S3, and Integrations. The main area is titled 'Tables (0) Info' and has a search bar, filters for 'Any tag key' and 'Any tag value', and a 'Create table' button. A message says 'You have no tables in this account in this AWS Region.'

Create a DynamoDB table for storing data

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

The screenshot shows the 'Create table' wizard. At the top, there's a navigation bar with the AWS logo, a search bar, and a 'Create table' button. Below it, the path is 'DynamoDB > Tables > Create table'. The main section is titled 'Create table' and contains 'Table details' with a note about schemaless databases. It has fields for 'Table name' (set to 'UsersTable'), 'Partition key' (set to 'email' with type 'String'), and 'Sort key - optional' (with a note about using it for sorting and a field for 'Enter the sort key name').

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](#)

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

The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The top navigation bar includes the AWS logo, a search bar, and a 'Tables' link. The main title is 'Create table'. Under 'Table details', the table name is set to 'AppointmentsTable'. The 'Partition key' section shows 'appointment_id' as the primary key of type 'String'. The 'Sort key - optional' section is empty. A note at the bottom states: '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.' The bottom of the page has a 'Next Step' button.

Table details [Info](#)
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.
 Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.)

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

1 to 255 characters and case sensitive.

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

1 to 255 characters and case sensitive.

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](#)

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

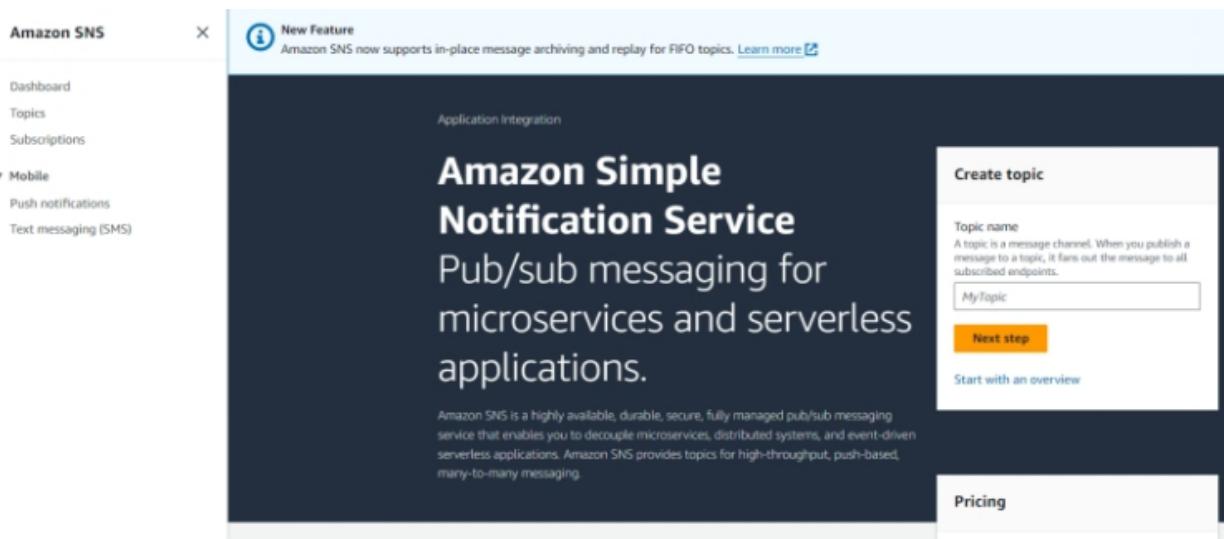
The screenshot shows the AWS search interface with the query 'sns' entered in the search bar. The results are categorized under 'Services' and 'Features'.

Services

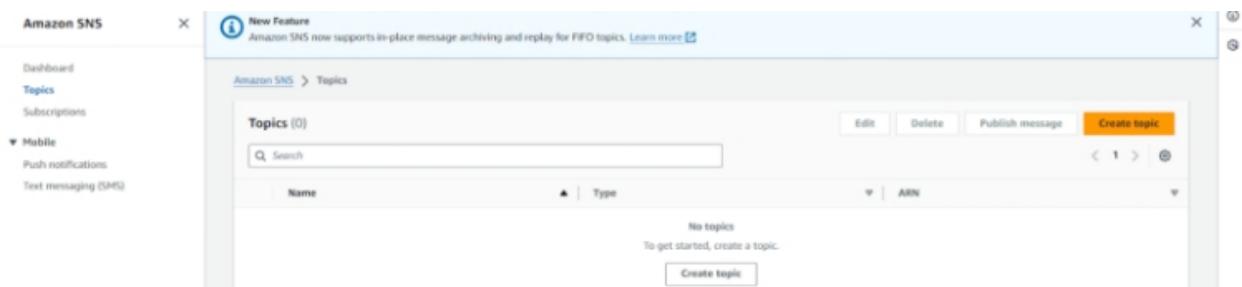
- Simple Notification Service** ☆
SNS managed message topics for Pub/Sub
- Route 53 Resolver**
Resolve DNS queries in your Amazon VPC and on-premises network.
- Route 53** ☆
Scalable DNS and Domain Name Registration
- AWS End User Messaging** ☆
Engage your customers across multiple communication channels

Features

- Events**
 - ElasticCache feature
- SMS**
 - AWS End User Messaging feature
- Hosted zones**
 - Route 53 features



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



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

Medtrack

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.

My Topic

Maximum 100 characters.

Access policy - optional [Info](#)

This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.

Data protection policy - optional [Info](#)

This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.

Delivery policy (HTTP/S) - optional [Info](#)

The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.

Delivery status logging - optional [Info](#)

These settings configure the logging of message delivery status to CloudWatch Logs.

Tags - optional

A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. [Learn more](#)

Active tracing - optional [Info](#)

Use AWS X-Ray active tracing for this topic to view its traces and service map in Amazon CloudWatch. Additional costs apply.

[Cancel](#)

[Create topic](#)

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

The screenshot shows the Amazon SNS Topics page. At the top, there's a blue banner with a 'New Feature' message about High Throughput FIFO topics. Below it, a green banner says 'Topic Medtrack created successfully.' The main area is titled 'Medtrack' and shows 'Details' for the topic. It includes fields for Name (Medtrack), ARN (arn:aws:sns:ap-south-1:940482422578:Medtrack), and Type (Standard). To the right, there's a 'Display name' field and a 'Topic owner' field showing the ARN. Below the details, there are tabs for Subscriptions, Access policy, Data protection policy, Delivery policy (HTTP/S), Delivery status logging, Encryption, and Tags. The 'Subscriptions' tab is active, showing one entry: 'Subscriptions (1)'. The entry shows the ARN of the subscription and includes buttons for Edit, Delete, Request confirmation, Confirm subscription, and Create subscription.

Subscribe users and Admin

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

The screenshot shows the 'Create subscription' wizard for the 'Medtrack' topic. The first step, 'Topic ARN', has the ARN 'arn:aws:sns:ap-south-1:940482422578:Medtrack' entered. The next step, 'Protocol', is set to 'Email'. The 'Endpoint' field contains the email address 'sara@teja47@gmail.com'. Below these fields, a note says 'After your subscription is created, you must confirm it.' A 'Subscription filter policy - optional' section is present with a note 'This policy filters the messages that a subscriber receives.' and a 'Redrive policy (dead-letter queue) - optional' section with a note 'Send undeliverable messages to a dead-letter queue.' At the bottom, there are 'Cancel' and 'Create subscription' buttons.

- 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	
	<input type="text"/> Search					

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

AWS Notification - Subscription Confirmation Inbox ×

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](#)



Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:

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

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

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

The screenshot shows the AWS SNS console for the 'Medtrack' topic. At the top, there is a blue banner with a 'New Feature' icon and the text 'Amazon SNS now supports High Throughput FIFO topics. Learn more'. Below the banner, the topic name 'Medtrack' is displayed, along with 'Edit', 'Delete', and 'Publish message' buttons. The 'Details' section shows the following information:

- Name: Medtrack
- Display name: -
- ARN: arn:aws:sns:ap-south-1:940482422578:Medtrack
- Topic owner: 940482422578
- Type: Standard

Below the details, there is a navigation bar with tabs: Subscriptions (which is selected), Access policy, Data protection policy, Delivery policy (HTTP/S), Delivery status logging, Encryption, and Tags. The 'Subscriptions' tab shows one subscription entry:

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

At the bottom of the page, there are links for 'Privacy', 'Terms', and 'Cookie preferences'.

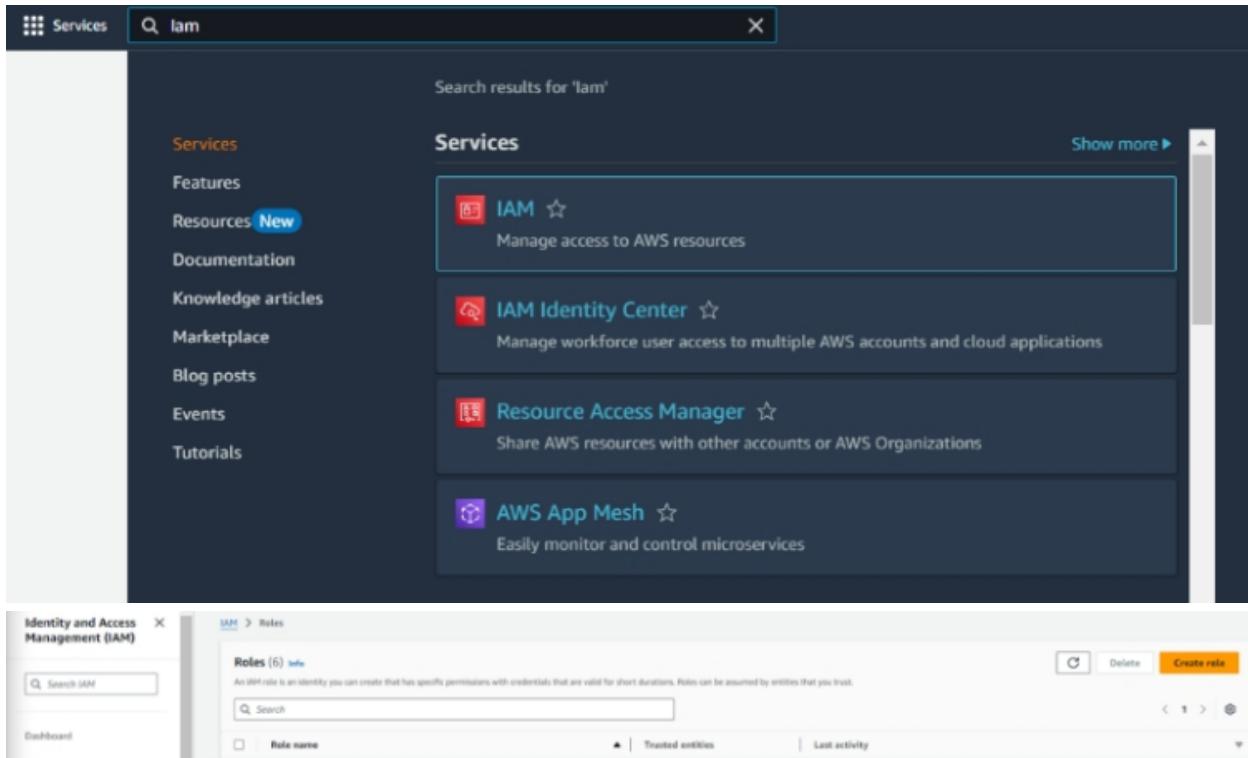
Milestone 5: IAM Role Setup

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.

The screenshots show the AWS IAM 'Create role' wizard at Step 2: Add permissions. In the first screenshot, under 'Select trusted entity', 'AmazonEC2FullAccess' is selected. Under 'Use case', 'AmazonEC2FullAccess' is also selected. In the second screenshot, the 'Permissions policies' section shows two policies attached: 'AmazonDynamoDBFullAccess' and 'AmazonSNSFullAccess'. Both are highlighted with orange circles.

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.

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Add permissions Info

Permissions policies (2/955) Info

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

Filter by Type All types

Policy name	Type
<input checked="" type="checkbox"/>  AmazonSNSFullAccess	AWS managed
<input type="checkbox"/>  AmazonSQSFullAccess	AWS managed
<input type="checkbox"/>  AmazonVPCRole	AWS managed
<input type="checkbox"/>  APIGatewayManageV2	AWS managed
<input type="checkbox"/>  AWSDeviceDefenderPublishFindingsToAWSLogs	AWS managed

Set permissions boundary - optional

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.

FlaskDynamoSNSRole		Info	Delete															
Allows EC2 instances to call AWS services on your behalf.																		
Summary			Edit															
Creation date	ARN	Instance profile ARN																
April 16, 2025, 22:10 (UTC+05:30)	arn:aws:iam::940482422578:role/FlaskDynamoSNSRole	arn:aws:iam::940482422578:instance-profile/FlaskDynamoSNSRole																
Last activity	Maximum session duration																	
5 hours ago	1 hour																	
Permissions	Trust relationships	Tags	Last Accessed	Revoke sessions														
Permissions policies (2) Info			C	Simulate	Remove	Add permissions												
You can attach up to 10 managed policies.				<	1	>												
<div style="display: flex; justify-content: space-between;"> Filter by Type Search All types ▼ </div> <table border="1"> <thead> <tr> <th><input type="checkbox"/></th> <th>Policy name</th> <th>Type</th> <th>Attached entities</th> <th>▼</th> </tr> </thead> <tbody> <tr> <td><input type="checkbox"/></td> <td>AmazonDynamoDBFullAccess</td> <td>AWS managed</td> <td>3</td> <td>▼</td> </tr> <tr> <td><input type="checkbox"/></td> <td>AmazonSNSFullAccess</td> <td>AWS managed</td> <td>3</td> <td>▼</td> </tr> </tbody> </table>				<input type="checkbox"/>	Policy name	Type	Attached entities	▼	<input type="checkbox"/>	AmazonDynamoDBFullAccess	AWS managed	3	▼	<input type="checkbox"/>	AmazonSNSFullAccess	AWS managed	3	▼
<input type="checkbox"/>	Policy name	Type	Attached entities	▼														
<input type="checkbox"/>	AmazonDynamoDBFullAccess	AWS managed	3	▼														
<input type="checkbox"/>	AmazonSNSFullAccess	AWS managed	3	▼														

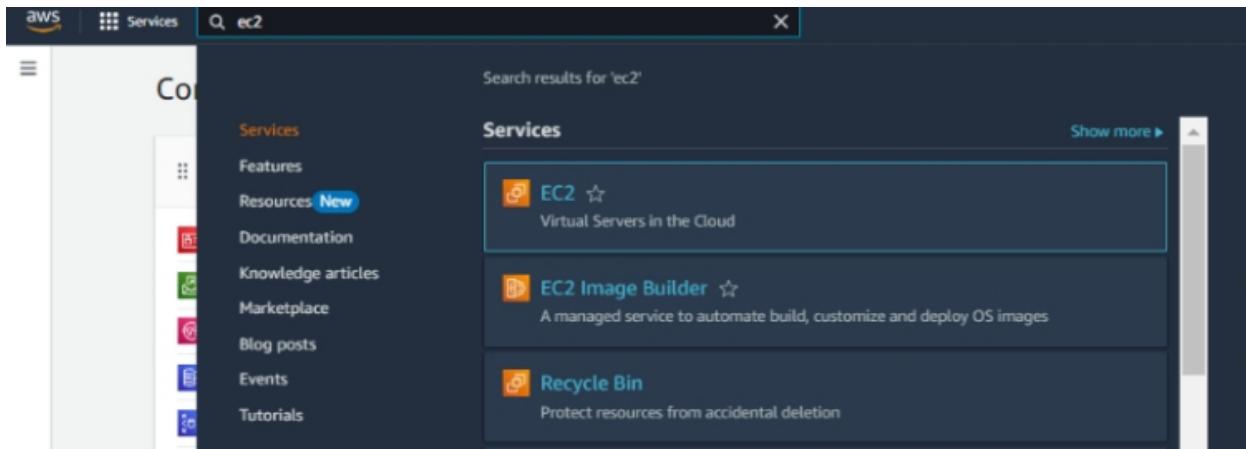
Milestone 6: EC2 Instance setup

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.

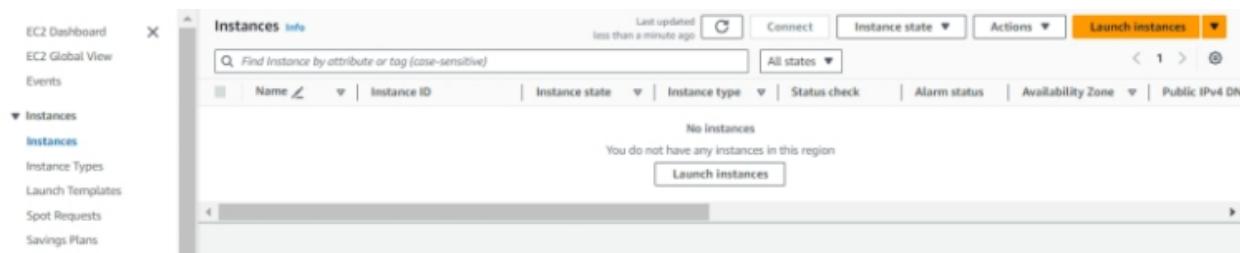
■ Launch EC2 Instance

- 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.

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

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 Machine Image (AMI)

Amazon Linux	macOS	Ubuntu	Windows	Red Hat	⋮

Amazon Linux 2023 AMI
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

t2.micro	Free tier eligible
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: 0.0117 USD per Hour On-Demand RHEL base pricing: 0.0268 USD per Hour On-Demand SUSE base pricing: 0.0124 USD per Hour	▼

All generations

Key pair (login)

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

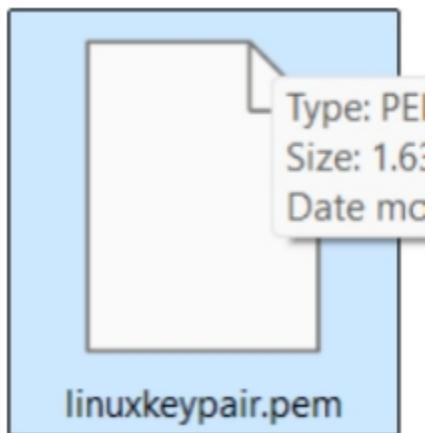
▼ Key pair (login) Info

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.

▼ Network settings [Info](#)

VPC - required [Info](#)

vpc-03cdc7b6f19dd7211 (default) [Edit](#)

Subnet [Info](#)

No preference [Edit](#) [Create new subnet](#)

Auto-assign public IP [Info](#)

Enable [Edit](#)

Additional charges apply when outside of free tier allowance

Firewall (security groups) [Info](#)

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Create security group [Edit](#)

Select existing security group [Edit](#)

Security group name - required

launch-wizard [Edit](#)

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 _-:/@#\$%^&_=!\$^*

Description - required [Info](#)

launch-wizard created 2024-10-13T17:49:56.622Z [Edit](#)

Inbound Security Group Rules

▼ Security group rule 1 (TCP, 22, 0.0.0.0/0) [Remove](#)

Type Info ssh Edit	Protocol Info TCP Edit	Port range Info 22 Edit
Source type Info Anywhere Edit	Source Info Add CIDR, prefix list or security Edit	Description - optional Info e.g. SSH for admin desktop Edit
0.0.0.0/0 Edit		

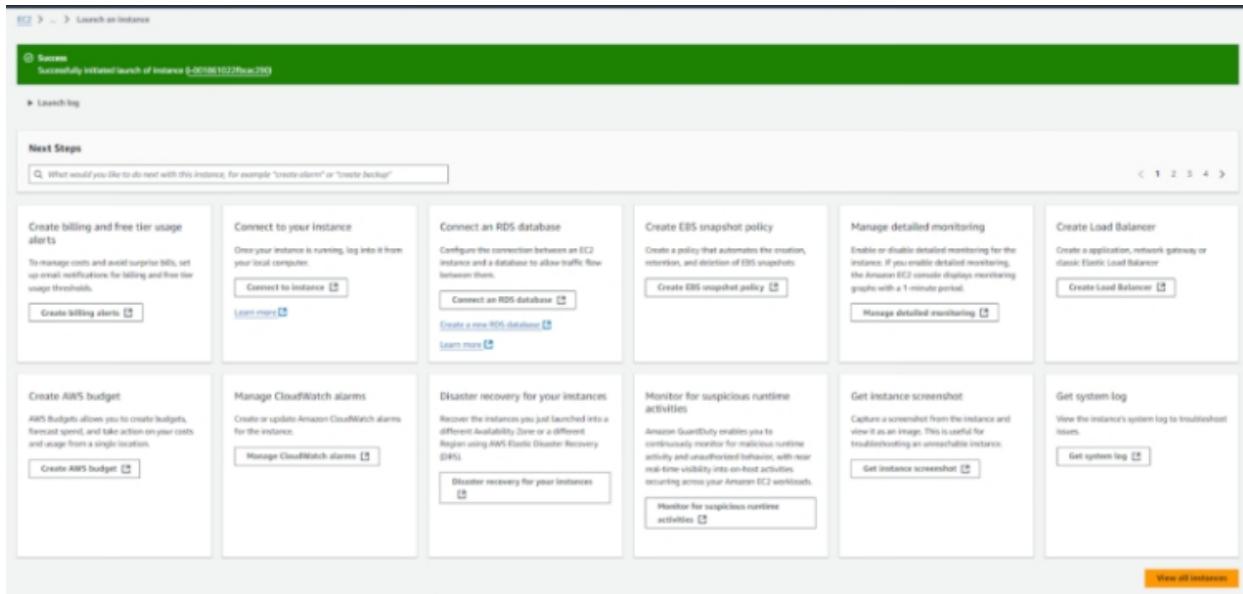
▼ Security group rule 2 (TCP, 80, 0.0.0.0/0) [Remove](#)

Type Info HTTP Edit	Protocol Info TCP Edit	Port range Info 80 Edit
Source type Info Custom Edit	Source Info Add CIDR, prefix list or security Edit	Description - optional Info e.g. SSH for admin desktop Edit
0.0.0.0/0 Edit		

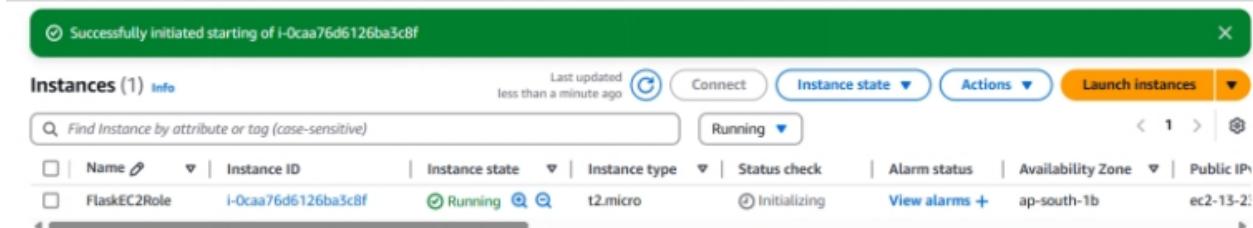
▼ Security group rule 3 (TCP, 5000, 0.0.0.0/0) [Remove](#)

Type Info Custom TCP Edit	Protocol Info TCP Edit	Port range Info 5000 Edit
Source type Info Custom Edit	Source Info Add CIDR, prefix list or security Edit	Description - optional Info e.g. SSH for admin desktop Edit
0.0.0.0/0 Edit		

[Add security group rule](#)



- 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.

The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with navigation links like Dashboard, EC2 Global View, Events, Instances, Images, Elastic Block Store, and CloudShell. The main area displays an instance summary for the instance ID i-0caa76d6126ba3c8f. The instance is currently stopped. Key details shown include:

- Public IPv4 address:** 172.31.14.197
- Instance state:** Stopped
- Private IP DNS name (IPv4 only):** ip-172-31-14-197.ap-south-1.compute.internal
- Instance type:** t2.micro
- VPC ID:** vpc-086cb887a0cc74850
- Subnet ID:** subnet-0c7267d27d4a62be0
- Instance ARN:** arnaws:ec2:ap-south-1:940482422578:instance/i-0caa76d6126ba3c8f

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.

The screenshot shows the 'Modify IAM role' dialog box. At the top, it says 'Modify IAM role' and 'Info'. Below that, it says 'Attach an IAM role to your instance.' The 'Instance ID' dropdown is set to 'i-0caa76d6126ba3c8f (FlaskEC2Role)'. Under 'IAM role', there's a dropdown menu with 'FlaskDynamoSNSRole' selected. To the right of the dropdown is a 'Create new IAM role' button. At the bottom right of the dialog are 'Cancel' and 'Update IAM role' buttons.

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.

The screenshot shows the AWS EC2 Instances dashboard. At the top, there are buttons for 'Last updated less than a minute ago', 'Connect', 'Instance state', 'Actions', and 'Launch instances'. A search bar says 'Find Instance by attribute or tag (case-sensitive)'. Below the search bar, there's a dropdown set to 'Running'. The main table has columns: Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IPv4. One row is selected, showing 'FlaskEC2Role' with Instance ID 'i-Ocaa76d6126ba3c8f', State 'Running', Type 't2.micro', Status '2/2 checks passed', Alarm status 'View alarms +', Availability Zone 'ap-south-1b', and Public IP 'ec2-13-203-'. A note at the bottom of the table says 'Filter table to exclude running instances'.

- Now connect the EC2 with the files

The screenshot shows the 'Connect to instance' dialog. At the top, it says 'EC2 > Instances > i-Ocaa76d6126ba3c8f > Connect to instance'. Below that, it says 'Connect to instance info' and 'Connect to your instance i-Ocaa76d6126ba3c8f (FlaskEC2Role) using any of these options'. There are four tabs: 'EC2 Instance Connect' (selected), 'Session Manager', 'SSH client', and 'EC2 serial console'. Under 'EC2 Instance Connect', there are two connection types: 'Connect using EC2 Instance Connect' (selected) and 'Connect using EC2 Instance Connect Endpoint'. The 'Connect using EC2 Instance Connect' section shows the instance ID 'i-Ocaa76d6126ba3c8f' and the public IPv4 address '13.205.157.118'. The 'Connect using EC2 Instance Connect Endpoint' section is collapsed. Below these, there's a 'Username' field containing 'ec2-user' and a note: 'Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.' At the bottom, there's a note: 'Note: In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.' On the right side, there are 'Cancel' and 'Connect' buttons. At the very bottom, there are links for 'CloudShell', 'Feedback', '© 2025, Amazon Web Services, Inc. or its affiliates.', 'Privacy', 'Terms', and 'Cookie preferences'.

A newer release of "Amazon Linux" is available.
Version 2023.7.20250414:
Run "/usr/bin/dnf check-release-update" for full release and version update info
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
Last login: Fri Apr 18 14:42:17 2025 from 13.233.177.3
[ec2-user@ip-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.

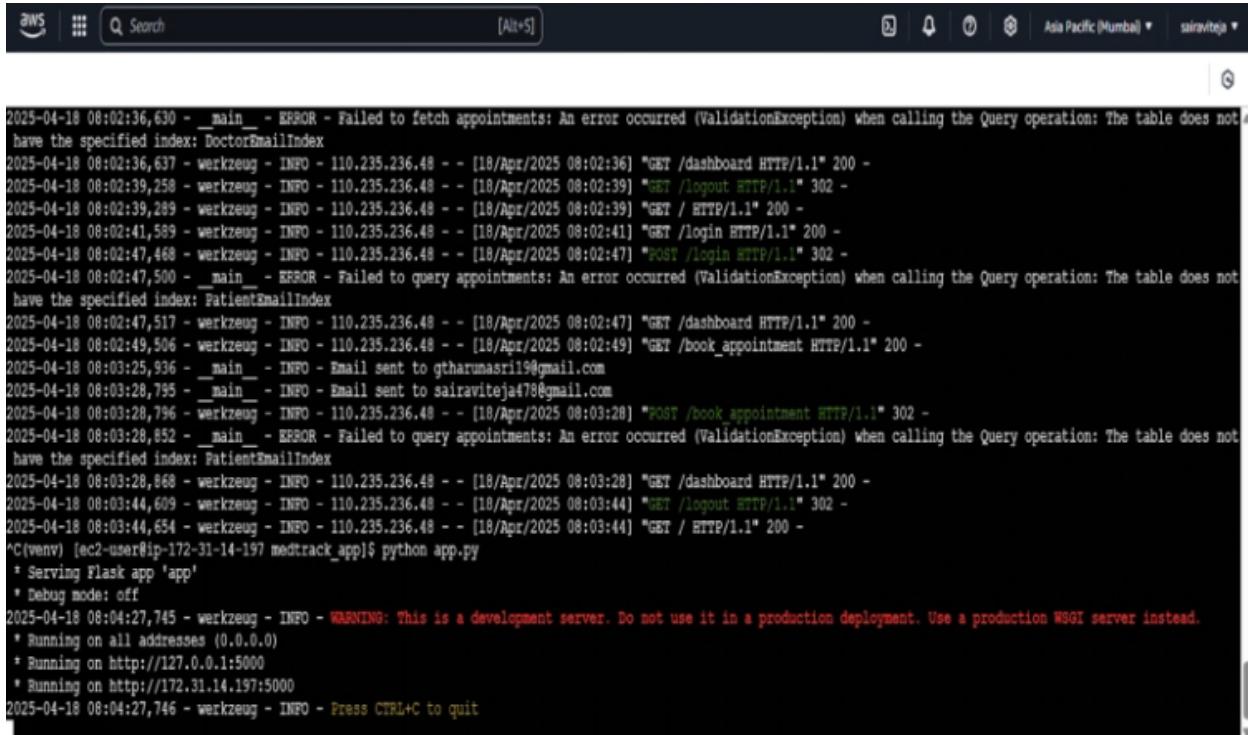
To navigate to the project directory, run the following command:

```
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
```



The screenshot shows a CloudWatch Log Stream for an EC2 instance. The logs are displayed in a terminal window with various log entries from the Flask application. Key log entries include:

- INFO: Failed to fetch appointments: An error occurred (ValidationException) when calling the Query operation: The table does not have the specified index: DoctorEmailIndex
- INFO: werkzeug - 110.235.236.48 - - [18/Apr/2025 08:02:36] "GET /dashboard HTTP/1.1" 200 -
- INFO: werkzeug - 110.235.236.48 - - [18/Apr/2025 08:02:39] "GET /logout HTTP/1.1" 302 -
- INFO: werkzeug - 110.235.236.48 - - [18/Apr/2025 08:02:39] "GET / HTTP/1.1" 200 -
- INFO: werkzeug - 110.235.236.48 - - [18/Apr/2025 08:02:41] "GET /login HTTP/1.1" 200 -
- INFO: werkzeug - 110.235.236.48 - - [18/Apr/2025 08:02:47] "POST /login HTTP/1.1" 302 -
- INFO: werkzeug - 110.235.236.48 - - [18/Apr/2025 08:02:47] "POST /login HTTP/1.1" 302 -
- INFO: werkzeug - 110.235.236.48 - - [18/Apr/2025 08:02:47] "GET /dashboard HTTP/1.1" 200 -
- INFO: werkzeug - 110.235.236.48 - - [18/Apr/2025 08:02:49] "GET /book_appointment HTTP/1.1" 200 -
- INFO: werkzeug - 110.235.236.48 - - [18/Apr/2025 08:03:25] "POST /book_appointment HTTP/1.1" 302 -
- INFO: werkzeug - 110.235.236.48 - - [18/Apr/2025 08:03:28] "POST /book_appointment HTTP/1.1" 302 -
- INFO: werkzeug - 110.235.236.48 - - [18/Apr/2025 08:03:28] "POST /book_appointment HTTP/1.1" 302 -
- INFO: werkzeug - 110.235.236.48 - - [18/Apr/2025 08:03:28] "GET /dashboard HTTP/1.1" 200 -
- INFO: werkzeug - 110.235.236.48 - - [18/Apr/2025 08:03:44] "GET /logout HTTP/1.1" 302 -
- INFO: werkzeug - 110.235.236.48 - - [18/Apr/2025 08:03:44] "GET / HTTP/1.1" 200 -
- * Serving Flask app 'app'
- * Debug mode: off
- INFO: werkzeug - 110.235.236.48 - - [18/Apr/2025 08:04:27] "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
- INFO: werkzeug - Press CTRL+C to quit

i-0caa76d6126ba3c8f (FlaskEC2Role)

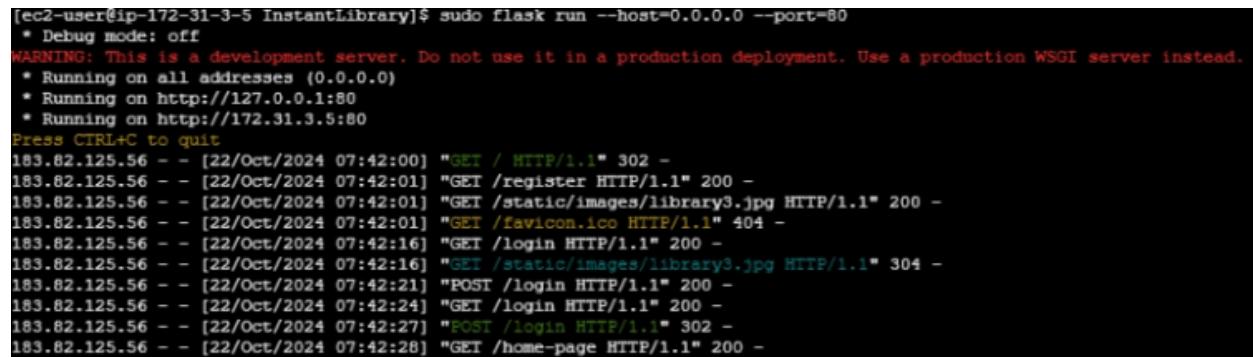
PublicIPs: 13.203.227.15 PrivateIPs: 172.31.14.197

X

Verify the Flask app is running:

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

Run the Flask app on the EC2 instance



The screenshot shows a CloudWatch Log Stream for an EC2 instance. The logs are displayed in a terminal window with various log entries from the Flask application. Key log entries include:

- INFO: werkzeug - 183.82.125.56 - - [22/Oct/2024 07:42:00] "GET / HTTP/1.1" 302 -
- INFO: werkzeug - 183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /register HTTP/1.1" 200 -
- INFO: werkzeug - 183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /static/images/library3.jpg HTTP/1.1" 200 -
- INFO: werkzeug - 183.82.125.56 - - [22/Oct/2024 07:42:01] "GET /favicon.ico HTTP/1.1" 404 -
- INFO: werkzeug - 183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /login HTTP/1.1" 200 -
- INFO: werkzeug - 183.82.125.56 - - [22/Oct/2024 07:42:16] "GET /static/images/library3.jpg HTTP/1.1" 304 -
- INFO: werkzeug - 183.82.125.56 - - [22/Oct/2024 07:42:21] "POST /login HTTP/1.1" 200 -
- INFO: werkzeug - 183.82.125.56 - - [22/Oct/2024 07:42:24] "GET /login HTTP/1.1" 200 -
- INFO: werkzeug - 183.82.125.56 - - [22/Oct/2024 07:42:27] "POST /login HTTP/1.1" 302 -
- INFO: werkzeug - 183.82.125.56 - - [22/Oct/2024 07:42:28] "GET /home-page HTTP/1.1" 200 -

Access the website through:

Public IPs: <https://13.201>

Milestone 8: Testing and Deployment

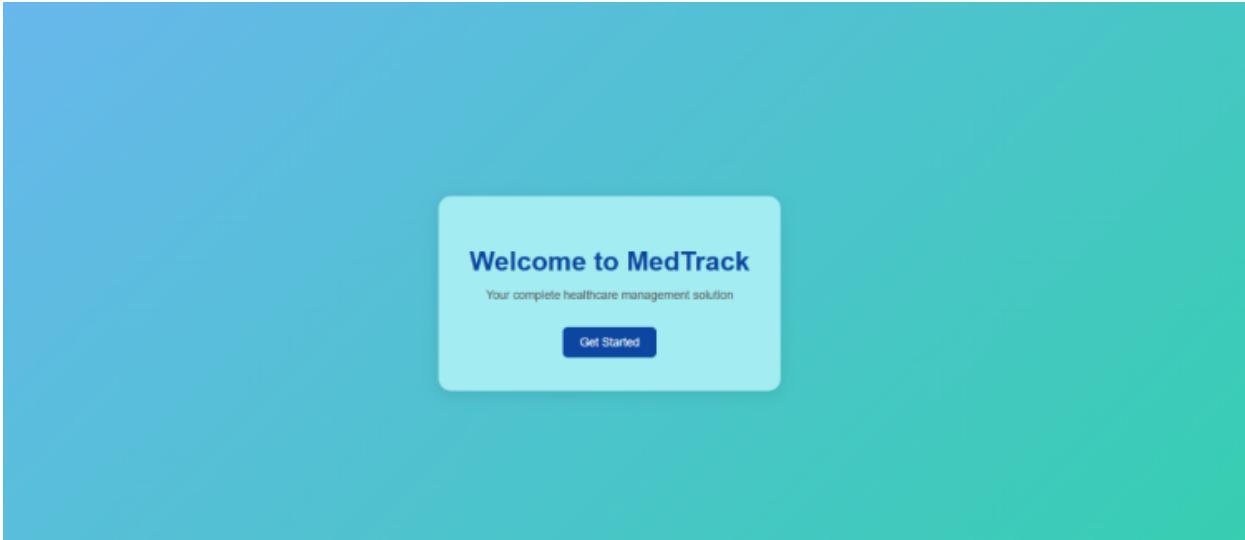
Testing and deployment involve verifying that your application works as expected before making it publicly accessible. Start by testing locally or on a staging environment to catch bugs and ensure functionality. Once tested, deploy the application to an EC2 instance, configure necessary services, and 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 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.
- 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 Index Page serves as the initial interface that directs users to the key functionalities of your web application

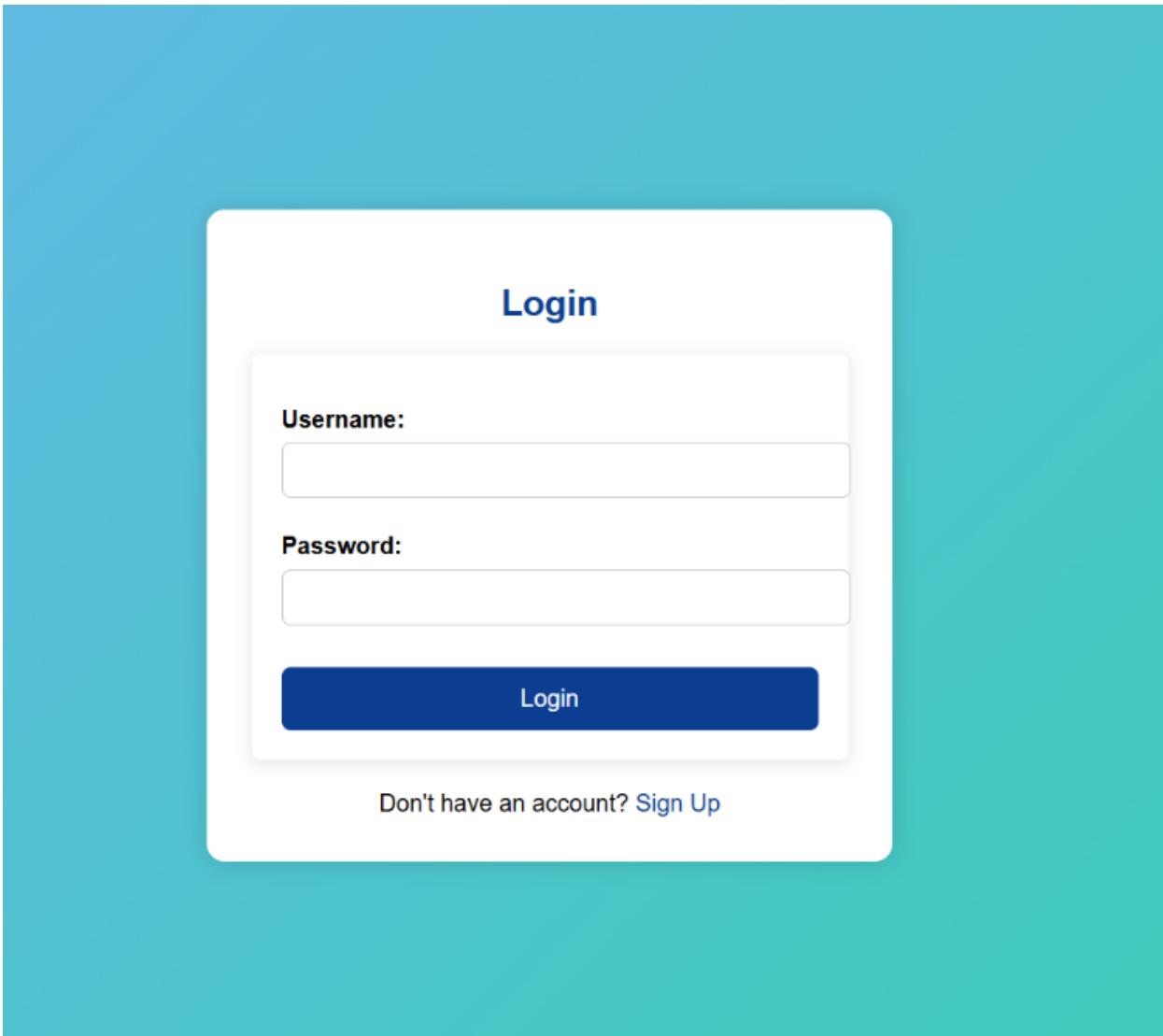


PATIENT LOGIN PAGE:

The Patient Login Page allows 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 are redirected to their respective dashboards to manage appointments, medical records, and other relevant tasks.



User Dashboard:

The User Dashboard (for patients) 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.

The screenshot shows a "Book Appointment" form on a teal-colored background. The form is contained within a white rounded rectangle. At the top center, it says "Book Appointment". Below that, a red message says "Login successful.". The form has several input fields:

- "Patient Name:" followed by a text input field.
- "Select Doctor:" followed by a dropdown menu with the placeholder "-Choose Doctor-".
- "Date:" followed by a date input field with the placeholder "dd-mm-yyyy".
- "Time:" followed by a time input field with the placeholder "... : ...".
- "Reason for Visit:" followed by a text area with the placeholder "Describe your issue...".

At the bottom of the form is a large blue button labeled "Book Appointment". Below the button is a link "← 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.

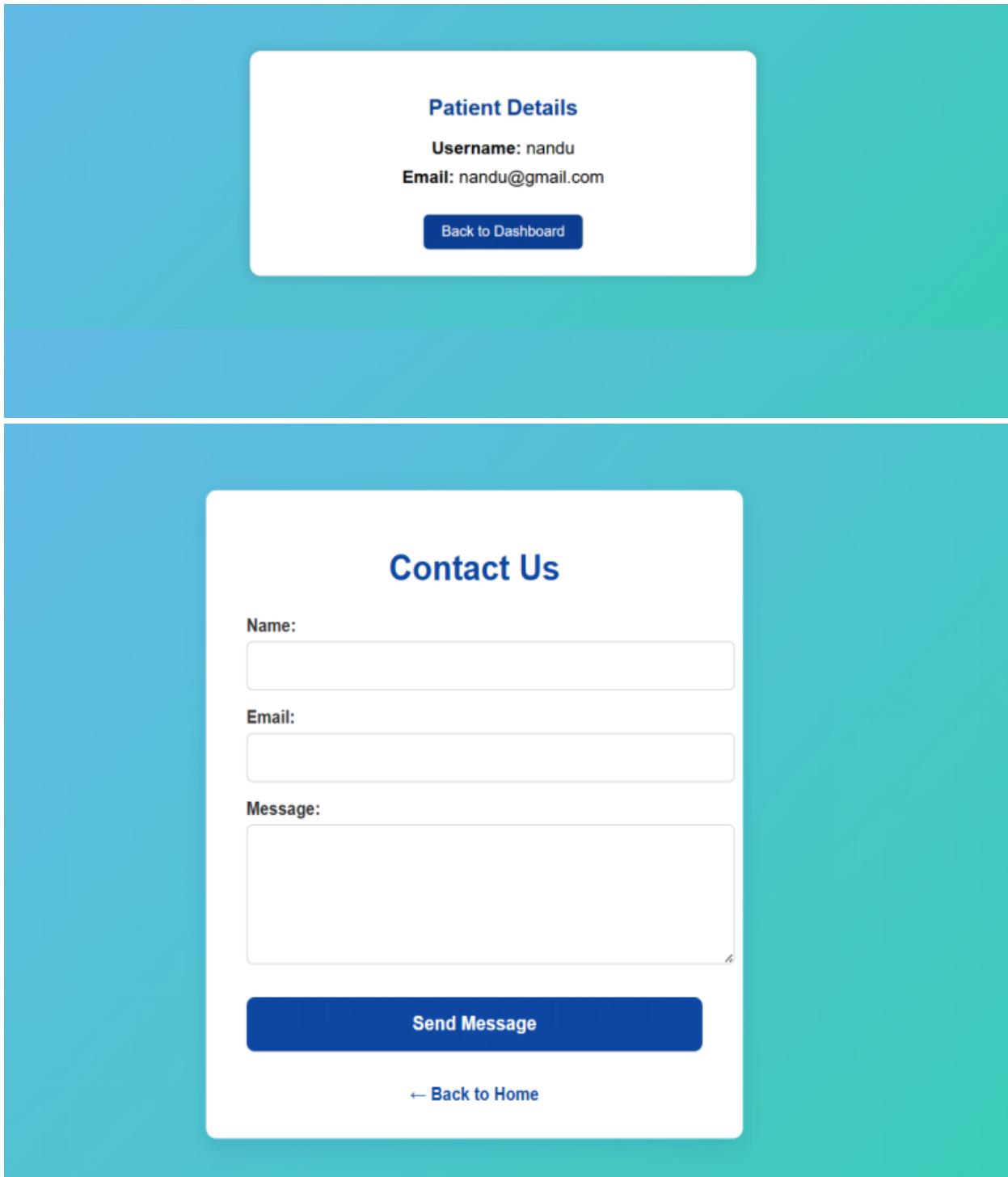
Doctor Dashboard

Upcoming
1

Patients Today
1

Pending Requests
1

[← Back to Home](#)



DynamoDB Database updati ons :

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)

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

	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

1. 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.

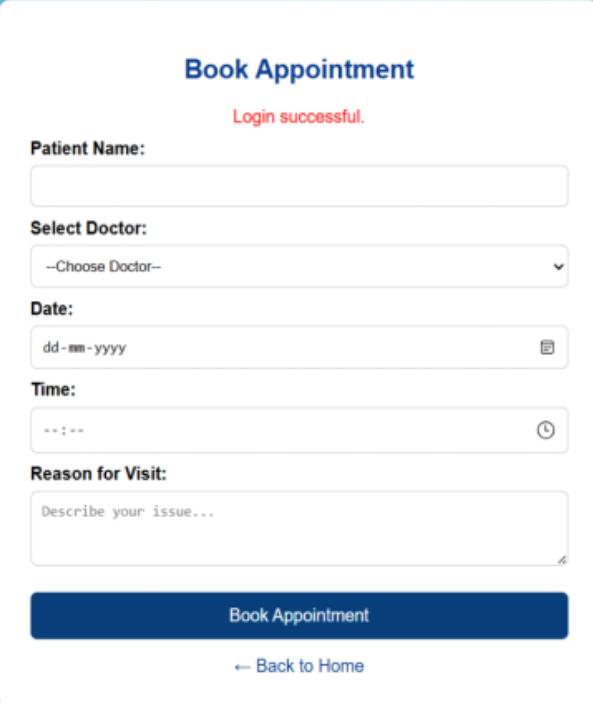
The Appointment Table allows for efficient management of appointments, ensuring accurate and up-to-date scheduling information for both doctors and patients.

Table: AppointmentsTable - Items returned (2)

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

	appointment_id (String)	appointment_date	created_at	diagnosis	doctor_email
<input type="checkbox"/>	86d52cfe-cce6-46b0-941d-1...	2025-04-30	2025-04-18...		gtharunasi19...
<input type="checkbox"/>	42e13fef-fbe9-42d1-9905-f...	2025-04-21	2025-04-18...	make sure t...	siri@thesmart...

Appointment confirmation:



The image shows a "Book Appointment" form. At the top, it says "Login successful." Below that, there are fields for "Patient Name" (with a placeholder input field), "Select Doctor" (with a dropdown menu showing "--Choose Doctor--"), "Date" (with a date input field showing "dd-mm-yyyy" and a calendar icon), "Time" (with a time input field showing ":: : ::" and a clock icon), and "Reason for Visit" (with a text area placeholder "Describe your issue..."). At the bottom of the form is a large blue "Book Appointment" button. Below the button, there is a link "← Back to Home".

The screenshot shows a mobile application interface titled "My Appointments". At the top, there is a header bar with the title "My Appointments". Below the header is a table with four columns: "Patient", "Doctor", "Date", and "Time". A single row of data is displayed: "nandu" under Patient, "Dr. Johnson" under Doctor, "2025-07-07" under Date, and "01:00" under Time. At the bottom of the screen, there is a navigation link labeled "← Back to Home".

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.