

MedTrack: A Cloud-Enabled Healthcare Management System

Project Description:

In today's fast-evolving healthcare landscape, efficient communication and coordination between doctors and patients are crucial. Med Track 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. Med Track 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.

Scenarios

Scenario 1: Efficient Appointment Booking System for Patients

In the Med Track 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.

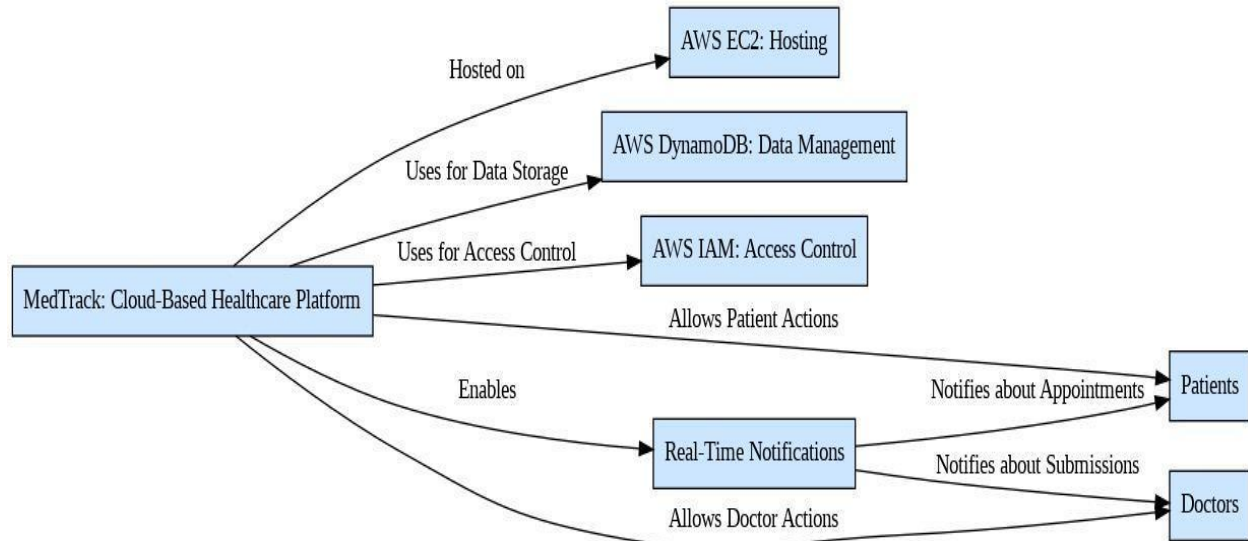
Scenario 2: Secure User Management with IAM

Med Track 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.

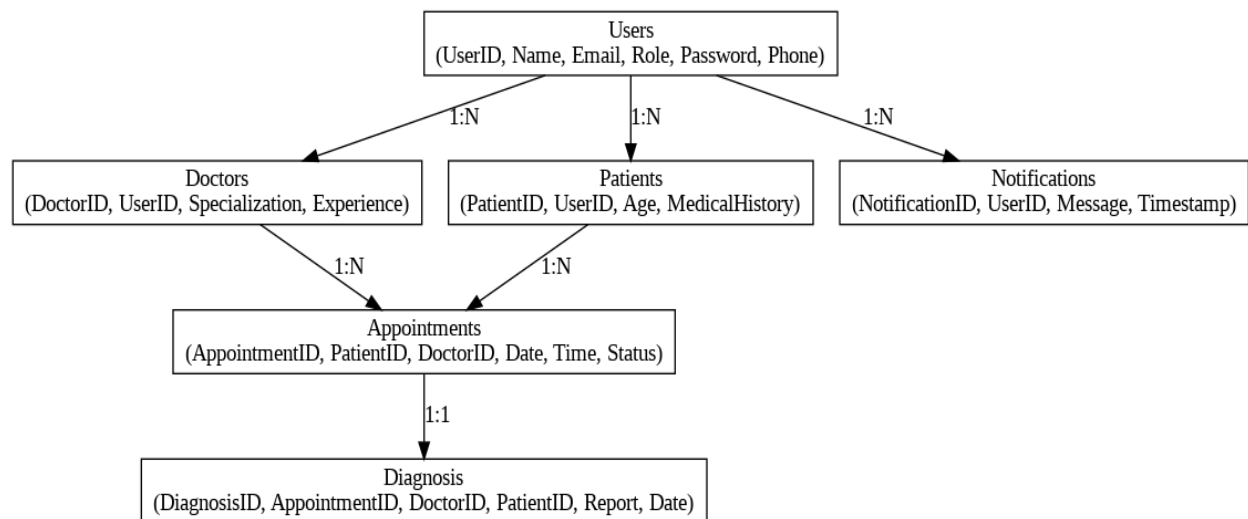
Scenario 3: Easy Access to Medical History and Resources

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

AWS ARCHITECTURE



Entity Relationship (ER)Diagram:



Pre-requisites:

1. **AWS Account Setup:** [AWS Account Setup](#)
2. **Understanding IAM:** [IAM Overview](#)
3. **Amazon EC2 Basics:** [EC2 Tutorial](#)
4. **DynamoDB Basics:** [DynamoDB Introduction](#)
5. **Git Version Control:** [Git Documentation](#)

Project Workflow: Med Track

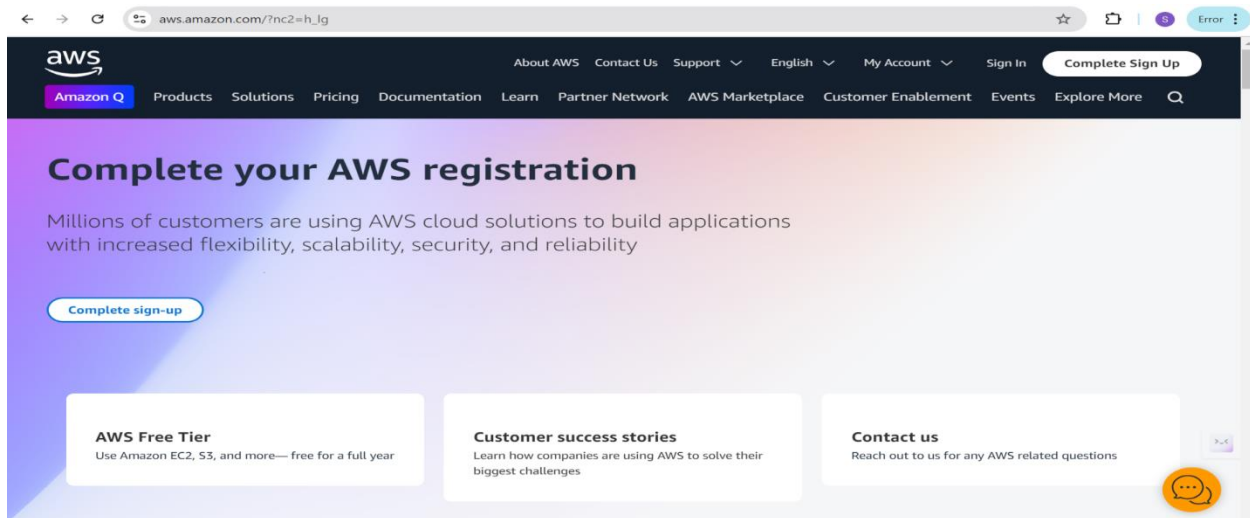
Project Flow

1. **AWS Account Setup and Login**
 - **Activity 1.1:** Set up an AWS account if not already done.
 - **Activity 1.2:** Log in to the AWS Management Console.
2. **DynamoDB Database Creation and Setup**
 - **Activity 2.1:** Create a DynamoDB Table.
 - **Activity 2.2:** Configure Attributes for User Data and Appointment Records.
3. **Backend Development and Application Setup**
 - **Activity 3.1:** Develop the Backend Using Flask.
 - **Activity 3.2:** Integrate AWS Services Using Boto3.
4. **IAM Role Setup**
 - **Activity 4.1:** Create IAM Roles for secure user access.
 - **Activity 4.2:** Attach Policies to ensure appropriate permissions.
5. **EC2 Instance Setup**
 - **Activity 5.1:** Launch an EC2 instance to host the Flask application.
 - **Activity 5.2:** Configure security groups for HTTP and SSH access.
6. **Testing and Deployment on EC2**
 - **Activity 6.1:** Upload Flask Files to the EC2 instance.
 - **Activity 6.2:** Run the Flask App to make it accessible.

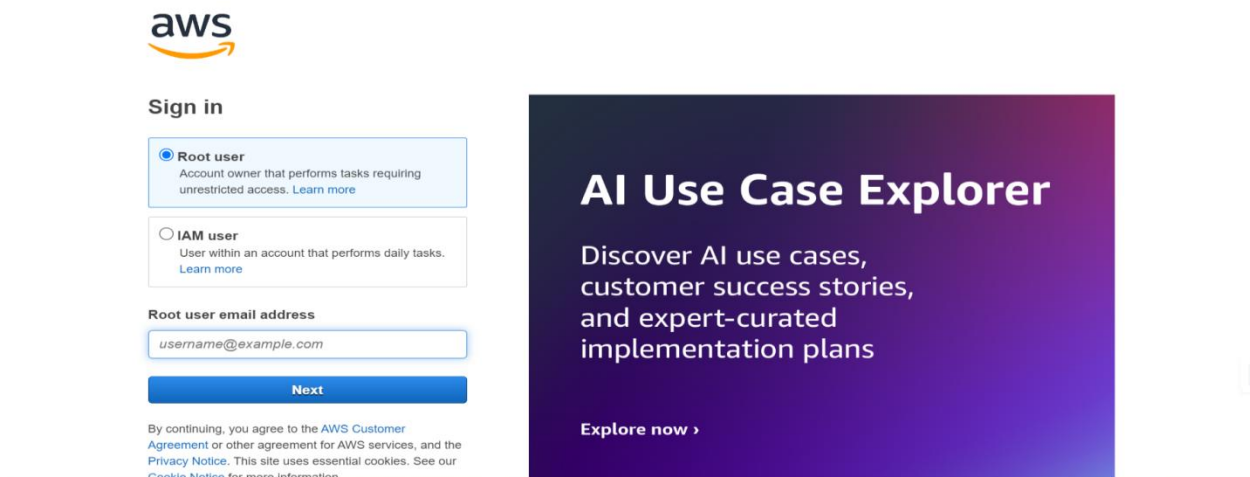
- **Activity 6.3:** Conduct functional testing to verify user registration, login, appointment booking, and data retrieval.

Milestone 1: AWS Account Setup and Login

- **Activity 1.1: Set up an AWS account if not already done.**
 - Sign up for an AWS account and configure billing settings.

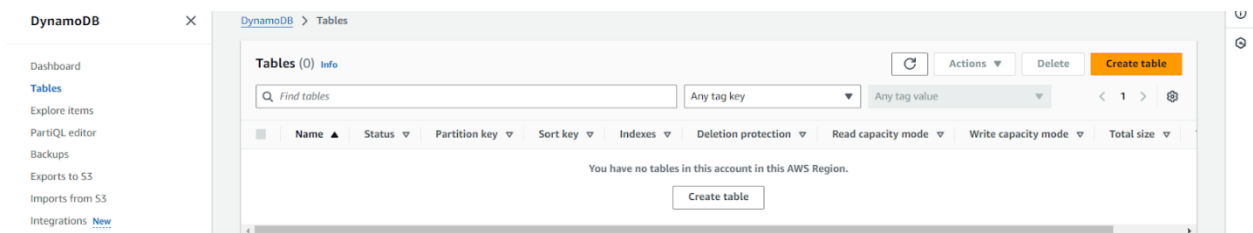
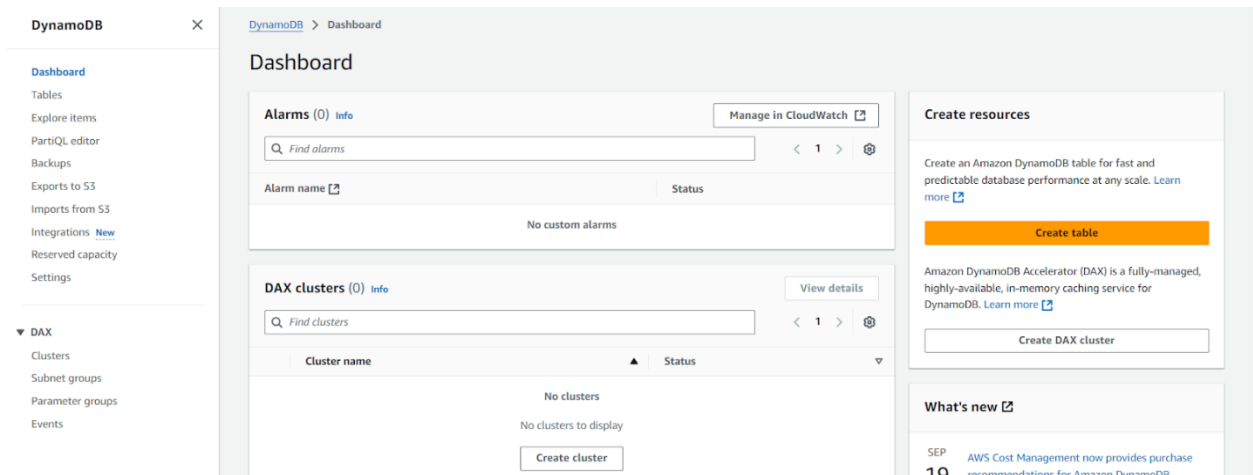
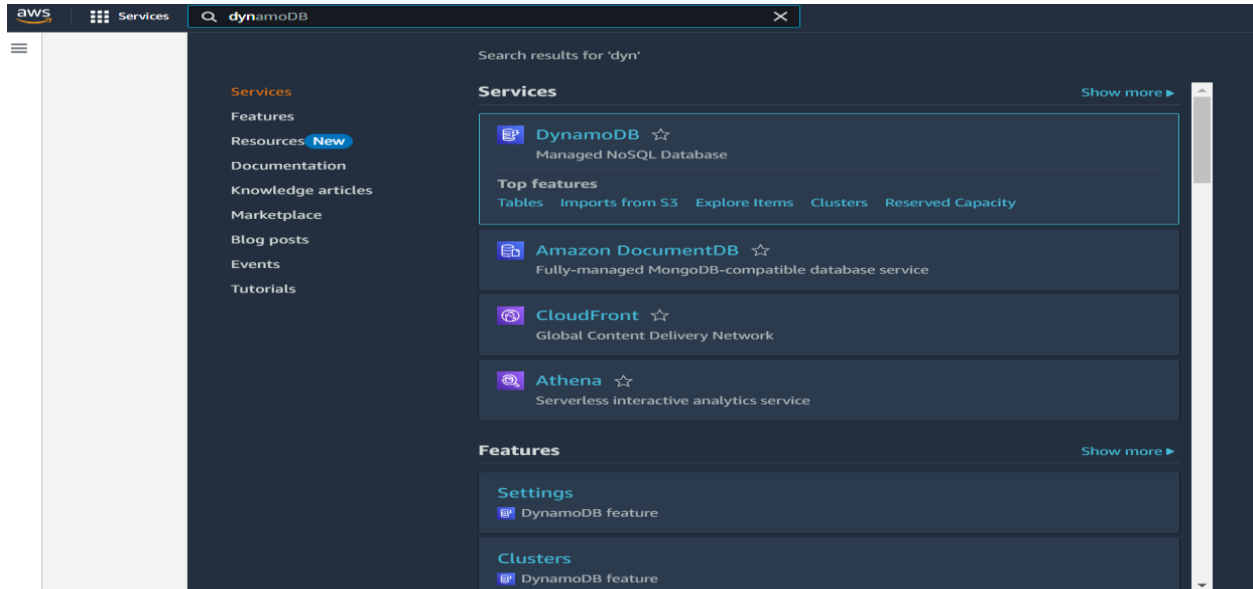


- **Activity 1.2: Log in to the AWS Management Console**
 - After setting up your account, log in to the [AWS Management Console](#).



Milestone 2: DynamoDB Database Creation and Setup

- **Activity 2.1: Navigate to the DynamoDB**
 - In the AWS Console, navigate to DynamoDB and click on create tables.



- **Activity 2.2: Create a DynamoDB table for storing registration details and book requests.**
 - Create Users Table with partition key “Email” with type String.

[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.

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.

Table settings

☒ **Default settings**
The fastest way to create your table. You can modify these settings now or after your table has been created.

☐ **Customize settings**
Use these advanced features to make DynamoDB work better for your needs.

Default table settings

These are the default settings for your new table. You can change some of these settings after creating the table.

Setting	Value	Editable after creation
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

Tags

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

No tags are associated with the resource.

Add new tag

You can add 50 more tags.

Cancel

Create table

- Follow the same steps to create Appointments Table with appointment_id as the primary key.

[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.

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

1 to 255 characters and case sensitive.

String ▼

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 ▼

Table settings



Default settings

The fastest way to create your table. You can modify these settings now or after your table has been created.



Customize settings

Use these advanced features to make DynamoDB work better for your needs.

Default table settings

These are the default settings for your new table. You can change some of these settings after creating the table.

Setting	Value	Editable after creation
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

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

DynamoDB > Tables

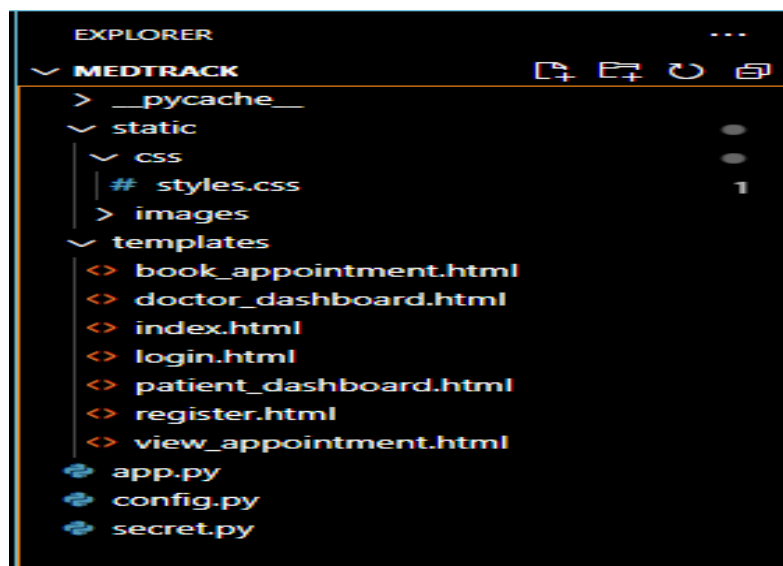
Tables (2) Info

Find tables Any tag key Any tag value < 1 > ⚙

<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Deletion protection	Favorite
<input type="checkbox"/>	AppointmentsTable	Active	appointment_id (S)	-	0	Off	☆
<input type="checkbox"/>	UsersTable	Active	email (S)	-	0	Off	☆

Milestone 3: Backend Development and Application Setup

- Activity 3.1: Develop the backend using Flask
 - File Explorer Structure



Description of the code:

- Flask App Initialization

```

1 from flask import Flask, request, session, redirect, url_for, render_template, flash
2 import boto3
3 from werkzeug.security import generate_password_hash, check_password_hash
4 from datetime import datetime
5 import uuid
6 from config import Config # Import Config class
7

```

```

9   app = Flask(__name__)
10  app.config.from_object(Config)
11  app.secret_key = app.config['SECRET_KEY']
12

```

- Use **boto3** to connect to **DynamoDB** for handling user registration, book requests database operations and also mention `region_name` where Dynamodb tables are created.

```

13  # Initialize DynamoDB resource
14  dynamodb = boto3.resource('dynamodb',
15  |         |         |         |         |         |         |         |
16  |         |         |         |         |         |         |         |
17  |         |         |         |         |         |         |         |
18  |         |         |         |         |         |         |         |
19  # Define DynamoDB tables
20  user_table = dynamodb.Table('UsersTable')
21  appointment_table = dynamodb.Table('AppointmentsTable')
22

```

Config.py:

```

config.py > ...
1  class Config:
2  |     AWS_ACCESS_KEY_ID = 'AKIA2#####OKDF4C'
3  |     AWS_SECRET_ACCESS_KEY = '1u+GQa#####/uXAS5zOZD'
4  |     AWS_REGION_NAME = 'ap-south-1'
5  |     SECRET_KEY = '24194554c#####61b6d87eb4de5'
6

```

Description: The Config class contains configuration settings for the Med Track application, including AWS access credentials and the secret key (Flask) for session management. These settings enable secure access to AWS services like DynamoDB and EC2 while ensuring the application's overall security.

Secret.py:

```

secret.py > ...
1  import os
2  import binascii
3
4  # Generate a random secret key
5  secret_key = binascii.hexlify(os.urandom(24)).decode()
6  print(secret_key)
7

```

Description: This code generates a random secret key for use in applications by creating a random sequence of 24 bytes using `os.Urandom(24)`. The bytes are then converted to a hexadecimal string, ensuring a secure key suitable for cryptographic operations.

- **Routes for Web Pages**

1. Home page Routes:

```
42 # Home Page
43 @app.route('/')
44 def index():
45     if is_logged_in():
46         return redirect(url_for('dashboard'))
47     return render_template('index.html')
48
```

Description: A route is defined for the home page (/) that checks if the user is logged in using the `is_logged_in()` function. If the user is authenticated, they are redirected to the dashboard; otherwise, the `index.html` template is rendered for unauthenticated users.

2. Register Route:

```
49 # Register User (Doctor/Patient)
50 @app.route('/register', methods=['GET', 'POST'])
51 def register():
52     if is_logged_in(): # Check if already logged in
53         return redirect(url_for('dashboard'))
54     if request.method == 'POST':
55         name = request.form['name']
56         email = request.form['email']
57         password = generate_password_hash(request.form['password']) # Hash password
58         age = request.form['age']
59         gender = request.form['gender']
60         role = request.form['role'] # 'doctor' or 'patient'
61
62         # Add user to DynamoDB
63         user_table.put_item(
64             Item={
65                 'email': email,
66                 'name': name,
67                 'password': password, # Store hashed password
68                 'age': age,
69                 'gender': gender,
70                 'role': role,
71                 'specialization': request.form['specialization'] if role == 'doctor' else None,
72             }
73         )
74         flash('Registration successful. Please log in.', 'success')
75         return redirect(url_for('login'))
76     return render_template('register.html')
```

Description: A route is defined for user registration ('/register') that handles both GET and POST requests. If the user is already logged in, they are redirected to the dashboard. For POST requests, the route collects form data (name, email, password, age, gender, role) and hashes the password for security. If the role is 'doctor,' a specialization is also collected. The user details are then stored in a DynamoDB table, and upon successful registration, the user is prompted to log in. For GET requests, the register.html template is rendered.

3.Register Route:

```
78 # Login User (Doctor/Patient)
79 @app.route('/login', methods=['GET', 'POST'])
80 def login():
81     if is_logged_in(): # If the user is already logged in, redirect to dashboard
82         return redirect(url_for('dashboard'))
83
84     if request.method == 'POST':
85         email = request.form['email']
86         password = request.form['password']
87         role = request.form['role'] # Get the selected role (doctor or patient)
88
89         # Validate user credentials
90         user = user_table.get_item(Key={'email': email}).get('Item')
91
92         if user:
93             # Check password and role
94             if check_password_hash(user['password'], password): # Use check_password_hash to verify has
95                 if user['role'] == role:
96                     session['email'] = email
97                     session['role'] = role # Store the role in the session
98                     flash('Login successful.', 'success')
99                     return redirect(url_for('dashboard'))
100                 else:
101                     flash('Invalid role selected.', 'danger')
102             else:
103                 flash('Invalid password.', 'danger')
104         else:
105             flash('Email not found.', 'danger')
106
107     return render_template('login.html')
```

Description: A route for user login ('/login') handles GET and POST requests. On POST, it validates the email, password, and role by checking credentials stored in DynamoDB, and logs the user in if successful. If already logged in, the user is redirected to the dashboard; otherwise, login.html is rendered.

4. Dashboard Route:

```
118 # Dashboard for both Doctors and Patients
119 @app.route('/dashboard')
120 def dashboard():
121     if not is_logged_in():
122         flash('Please log in to continue.', 'danger')
123         return redirect(url_for('login'))
124
125     role = session['role']
126     email = session['email']
127
128     if role == 'doctor':
129         # Show doctor dashboard with list of appointments
130         response = appointment_table.scan(
131             FilterExpression="#doctor_email = :email",
132             ExpressionAttributeNames={"#doctor_email": "doctor_email"},
133             ExpressionAttributeValues={"":email": email}
134         )
135         appointments = response['Items']
136         return render_template('doctor_dashboard.html', appointments=appointments)
137
138     elif role == 'patient':
139         # Show patient dashboard with list of their appointments
140         response = appointment_table.scan(
141             FilterExpression="#patient_email = :email",
142             ExpressionAttributeNames={"#patient_email": "patient_email"},
143             ExpressionAttributeValues={"":email": email}
144         )
145         appointments = response['Items']
146         return render_template('patient_dashboard.html', appointments=appointments)
147
```

Description: The dashboard route checks if the user is logged in and redirects them to the login page if not. Depending on the user's role (doctor or patient), it retrieves a list of appointments from DynamoDB using the user's email and renders either the `doctor_dashboard.html` or `patient_dashboard.html` template with the relevant appointment data.

5. Book Appointment Route

```
148 # Book an Appointment (Patient)
149 @app.route('/book_appointment', methods=['GET', 'POST'])
150 def book_appointment():
151     if not is_logged_in() or session['role'] != 'patient':
152         flash('Only patients can book appointments.', 'danger')
153         return redirect(url_for('login'))
154
155     if request.method == 'POST':
156         doctor_email = request.form['doctor_email']
157         symptoms = request.form['symptoms']
158         patient_email = session['email']
159
160         # Create a new appointment
161         appointment_id = str(uuid.uuid4())
162         appointment_table.put_item(
163             Item={
164                 'appointment_id': appointment_id,
165                 'doctor_email': doctor_email,
166                 'patient_email': patient_email,
167                 'symptoms': symptoms,
168                 'status': 'pending',
169                 'appointment_date': str(datetime.now()),
170             }
171         )
172         flash('Appointment booked successfully.', 'success')
173         return redirect(url_for('dashboard'))
174
```

```

175     # Get list of doctors for selection
176     response = user_table.scan(
177         FilterExpression="#role = :role",
178         ExpressionAttributeNames={"#role": "role"},
179         ExpressionAttributeValues={":role": 'doctor'})
180     )
181     doctors = response['Items']
182     return render_template('book_appointment.html', doctors=doctors)
183

```

Description: This route allows patients to book an appointment. If the user is logged in as a patient, they can submit a form with a doctor's email and symptoms. The appointment is then created with a unique ID and saved to DynamoDB. For GET requests, it retrieves a list of available doctors from DynamoDB and renders the book_appointment.html template for selection.

6.View_appointments Route:

```

184     # View Appointment (Doctor)
185     @app.route('/view_appointment/<appointment_id>', methods=['GET', 'POST'])
186     def view_appointment(appointment_id):
187         if not is_logged_in() or session['role'] != 'doctor':
188             flash('Only doctors can view appointments.', 'danger')
189             return redirect(url_for('login'))
190
191         # Fetch appointment details
192         response = appointment_table.get_item(Key={'appointment_id': appointment_id})
193         appointment = response.get('Item')
194
195         if request.method == 'POST':
196             diagnosis = request.form['diagnosis']
197             treatment_plan = request.form['treatment_plan']
198             prescription = request.form['prescription']
199

```

Description: This route allows doctors to view and update appointment details based on the provided appointment_id. If the user is logged in as a doctor, they can retrieve appointment information from DynamoDB and, via POST, submit a diagnosis, treatment plan, and prescription, updating the appointment status to "completed." For GET requests, the appointment data is displayed using the view_appointment.html template.

7.Submit_diagnosis Route:

```

219     # Submit Diagnosis (Doctor)
220     @app.route('/submit_diagnosis/<appointment_id>', methods=['POST'])
221     def submit_diagnosis(appointment_id):
222         diagnosis = request.form['diagnosis']
223         treatment_plan = request.form['treatment_plan']
224         prescription = request.form['prescription']
225
226         # Update the appointment in the database
227         appointment_table.update_item(
228             Key={'appointment_id': appointment_id},
229             UpdateExpression="SET diagnosis = :d, treatment_plan = :t, prescription = :p, #status = :s",
230             ExpressionAttributeNames={"#status": "status"},
231             ExpressionAttributeValues={
232                 ':d': diagnosis,
233                 ':t': treatment_plan,
234                 ':p': prescription,
235                 ':s': 'completed'
236             })
237
238         flash('Diagnosis submitted successfully.', 'success')
239         return redirect(url_for('dashboard'))
240
241

```

Description: This route allows doctors to submit a diagnosis, treatment plan, and prescription for a specific appointment via POST. The data is updated in DynamoDB by setting the diagnosis, treatment plan, prescription, and changing the appointment status to "completed." After the update, the doctor is redirected to the dashboard with a success message.

8. Logout Route

```
109 # Logout User
110 @app.route('/logout', methods=['GET', 'POST'])
111 def logout():
112     if request.method == 'POST': # Ensure it's a POST request for logout
113         session.pop('email', None)
114         session.pop('role', None)
115         flash('You have been logged out.', 'success')
116     return redirect(url_for('index'))
```

Description: This route handles user logout functionality by removing the user's email and role from the session upon receiving a POST request, effectively logging them out. Afterward, the user is redirected to the index page with a flash message indicating that they have been successfully logged out.

Deployment Code:

```
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

Description: This code snippet serves as the main entry point for the Flask application. When the script is executed directly, it starts the Flask development server in debug mode, allowing for live reloading and detailed error messages, which is useful for development and testing.

Milestone 4: IAM Role Setup

- **Activity 4.1: Create IAM Role.**
 - In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.
 - (Flask)

[IAM](#) > [Users](#) > Create user

Step 1
Specify user details

Step 2
Set permissions

Step 3
Review and create

Specify user details

User details

User name

med-user

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

☐ Provide user access to the AWS Management Console - optional
If you're providing console access to a person, it's a best practice [to](#) manage their access in IAM Identity Center.

?

If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. [Learn more](#)

Cancel

Next

Step 2: Add permissions

[Edit](#)

Permissions policy summary

Policy name ?	Type	Attached as
AmazonDynamoDBFullAccess	AWS managed	Permissions policy

Step 3: Add tags

Add tags - optional [Info](#)

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with the resource.

[Add new tag](#)

You can add up to 50 more tags.

[Cancel](#)[Previous](#)[Create role](#)

[IAM](#) > [Users](#)

Users (1/1) [Info](#)

Refresh

Delete

Create user

Search

< 1 > [Settings](#)

<input checked="" type="checkbox"/>	User name	Path	Group:	Last activity	MFA	Password age
<input checked="" type="checkbox"/>	med-user	/	0	10 hours ago	-	-

- **Activity 4.2: Attach Policies.**

Attach the following policies to the role:

- Amazon DynamoDB Full Access: Allows EC2 to perform read/write operations on DynamoDB.

IAM > Roles > Create role

Step 1
[Select trusted entity](#)

Step 2
Add permissions

Step 3
Name, review, and create

Add permissions [Info](#)

Permissions policies (1/955) [Info](#)

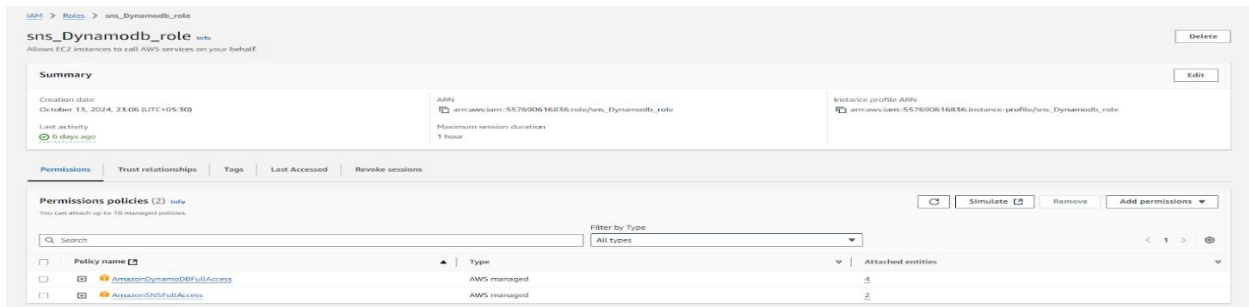
Choose one or more policies to attach to your new role.

Filter by Type
All types
4 matches

<input type="checkbox"/>	Policy name ?	Type	Description
<input checked="" type="checkbox"/>	AmazonDynamoDBFullAccess	AWS managed	Provides full access to Amazon Dynam...
<input type="checkbox"/>	AmazonDynamoDBReadOnlyAccess	AWS managed	Provides read only access to Amazon D...
<input type="checkbox"/>	AWSLambdaDynamoDBAccess	AWS managed	Provides list and read access to Dynam...
<input type="checkbox"/>	AWSLambdaInvocationDynamoAccess	AWS managed	Provides read access to DynamoDB Str...

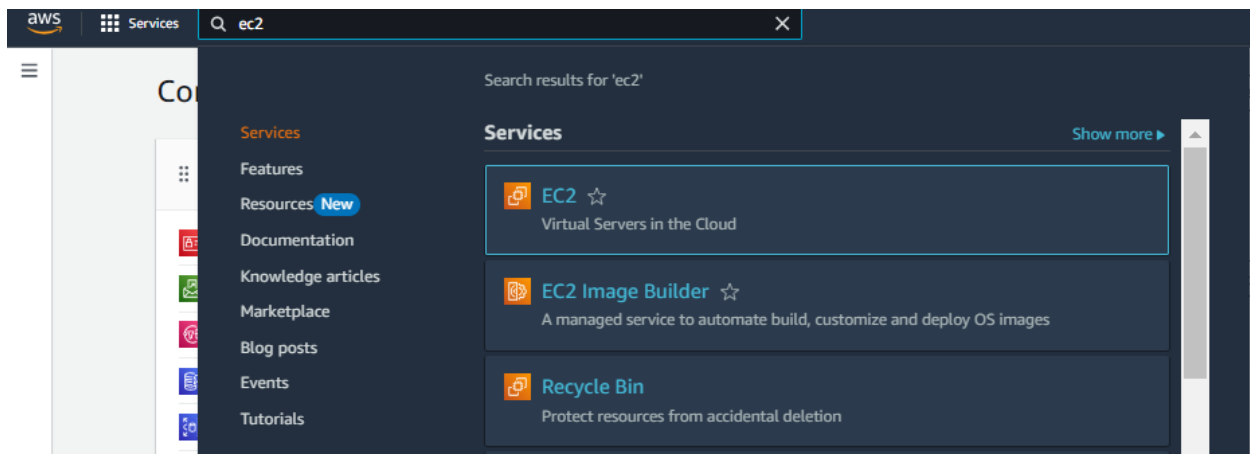
▶ Set permissions boundary - optional

[illegible]

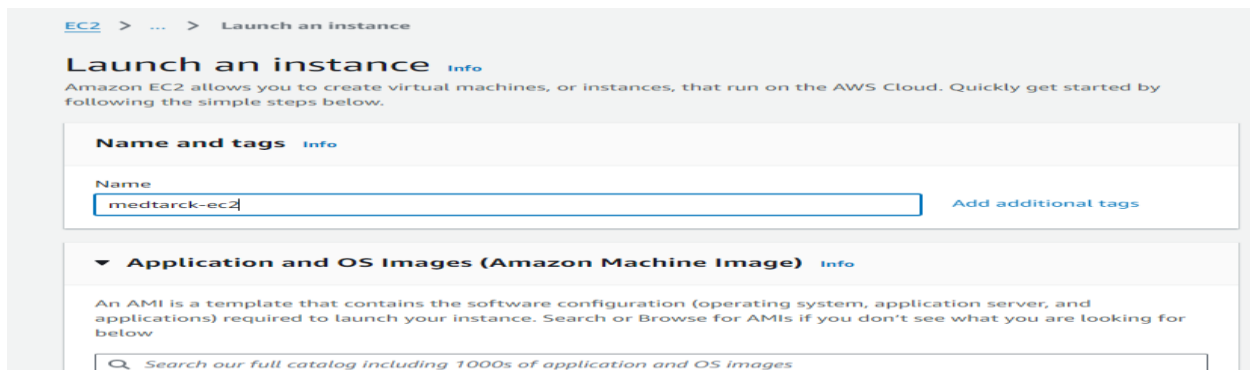


Milestone 5: EC2 Instance Setup

- Choose a Linux-based EC2 instance from the AWS Console to host the Med track application.
- **Activity 5.1: Launch an EC2 instance to host the Flask application.**
- **Launch EC2 Instance**
 - In the AWS Console, navigate to EC2 and launch a new instance.



- Click on Launch instance to launch EC2 instance



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

Recents

Quick Start

Amazon Linux
aws


macOS
Mac

Ubuntu
ubuntu

Windows
Microsoft

Red Hat
Red Hat

SUSE Li
SUS


Browse more AMIs
Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Amazon Linux 2023 AMI

ami-04a37924ffe27da53 (64-bit (x86), uefi-preferred) / ami-0846b753e2af0da6e (64-bit (Arm), uefi)
Virtualization: hvm ENA enabled: true Root device type: ebs

Free tier eligible

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-04a37924ffe27da53

Username

ec2-user

 Verified provider

- Create and download the key pair for Server access.

▼ 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

Create key pair

Key pair name

Key pairs allow you to connect to your instance securely.

med-key-pair

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

☒ RSA
RSA encrypted private and public key pair

☐ ED25519
ED25519 encrypted private and public key pair

Private key file format

☒ .pem
For use with OpenSSH

☐ .ppk
For use with PuTTY

⚠ When prompted, store the private key in a secure and accessible location on

Cancel

Create key pair

Activity 5.2: Configure security groups for HTTP, and SSH access.

▼ Network settings [Info](#)

Edit

Network [Info](#)

vpc-02ee710587549f123

Subnet [Info](#)

No preference (Default subnet in any availability zone)

Auto-assign public IP [Info](#)

Enable

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

☐ Select existing security group

We'll create a new security group called 'launch-wizard-2' with the following rules:

☒ Allow SSH traffic from
Helps you connect to your instance

Anywhere
0.0.0.0/0

☐ Allow HTTPS traffic from the internet
To set up an endpoint, for example when creating a web server

☐ Allow HTTP traffic from the internet
To set up an endpoint, for example when creating a web server

Instances (1/2) Info

Last updated less than a minute ago

Connect

Instance state

Actions

Launch instances

Find Instance by attribute or tag (case-sensitive)

All states

< 1 >

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability
<input type="checkbox"/>	med-ec2	i-05d97fea7203376de	Running	t2.micro	...	View alarms +	ap-south-
<input checked="" type="checkbox"/>	medtarck-ec2	i-0cbd6b6ccb2afc1f4	Running	t2.micro	Initializing	View alarms +	ap-south-

Setting up Inbound and Outbound rules

i-0cbd6b6ccb2afc1f4 (medtarck-ec2)

Details

Status and alarms

Monitoring

Security

Networking

Storage

Tags

▼ Security details

IAM Role

-

Owner ID

741448921864

Launch time

Wed Oct 23 2024 11:23:53 GMT+0530 (India Standard Time)

i-0f6f060a65171c78a (quiz-app)

▼ Inbound rules

Filter rules

< 1 >

Name	Security group rule ID	Port range	Protocol	Source
-	sgr-035f84939e1b2c81b	80	TCP	0.0.0.0/0
-	sgr-0a65c1024fe8d85fe	3306	TCP	0.0.0.0/0
-	sgr-0cde2a0a45ccc0352	22	TCP	0.0.0.0/0
-	sgr-0766684fdd95a7fce	443	TCP	0.0.0.0/0

- Add Type : HTTP > Source : Anywhere
- Add Type : HTTPS > Source : Anywhere

▼ Outbound rules

Filter rules

< 1 >

Name	Security group rule ID	Port range	Protocol	Destination
-	sgr-0aec66d8309fa06b1	All	All	0.0.0.0/0

Milestone 6: Testing and Deployment

- **Activity 6.1: Deploy to EC2**

1. Connect EC2 terminal.
2. Set up any necessary environment variables, including database connection strings.
3. Configure the web server to serve your application.
4. Start your application and ensure it's accessible via the EC2 instance's public IP or domain.
5. Run the below commands on ec2 terminal
6. `sudo yum update -y`
7. `sudo yum install python3 -y`
8. `sudo pip3 install virtualenv`
9. `python3 -m venv venv`
10. `source venv/bin/activate`
11. `pip install flask`

- **Functional Testing**

- Test the app.py application for functionality, including database interactions and frontend features.
- Run the Flask app **python3 app.py**
- It will give you the link

Access the website through:

Access the website through Public IPs:

[http:13.164.83.44:5000](http://13.164.83.44:5000)

- **Activity 6.2: Deployment**

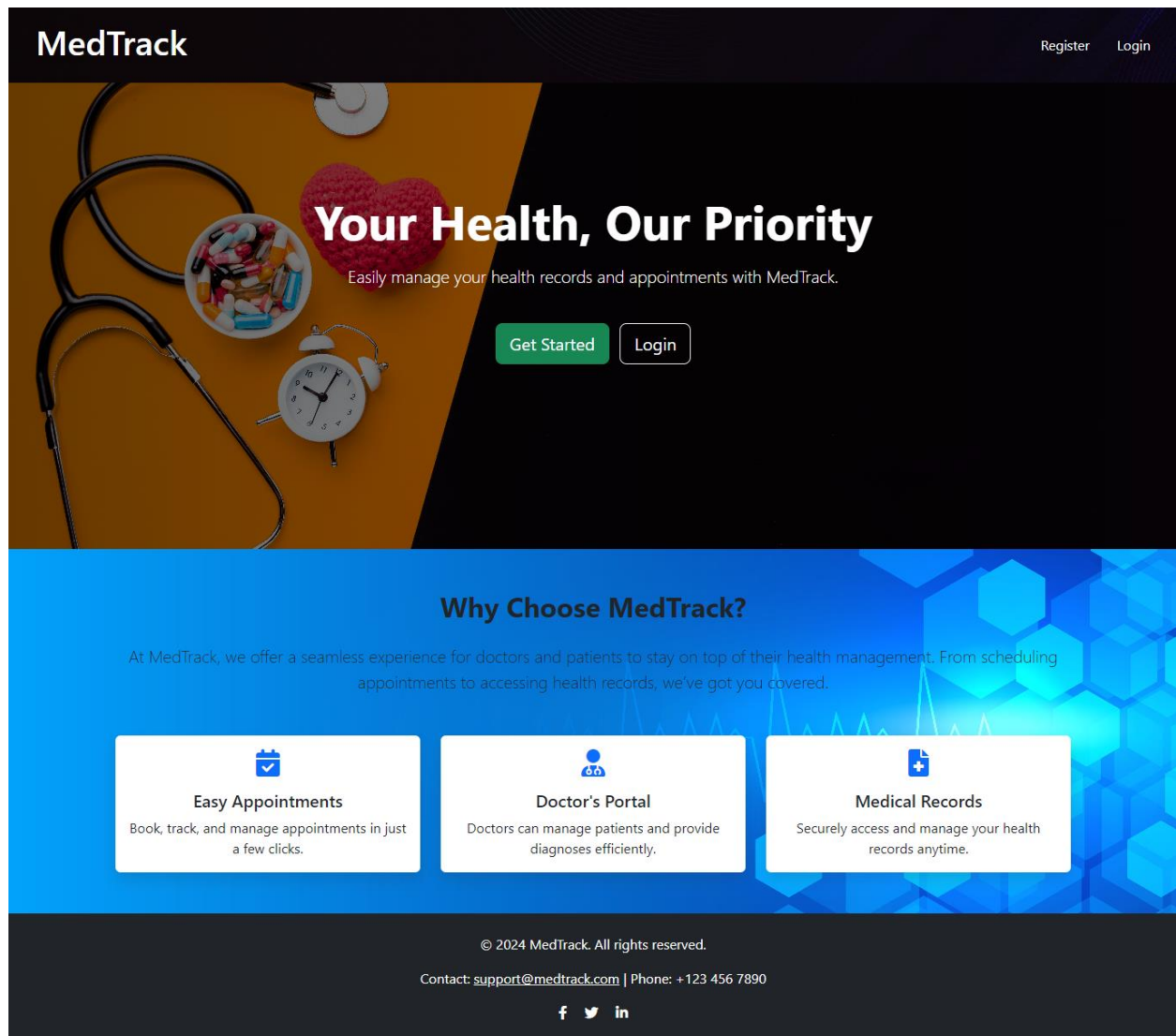
- Deploy the application in a production environment, ensuring high availability and performance.

Click on the link above and it will take you to the webpage:

Activity:6.3

- Conduct functional testing to verify user registration, login, (doctor, patient) dashboard, view appointments, book appointments,

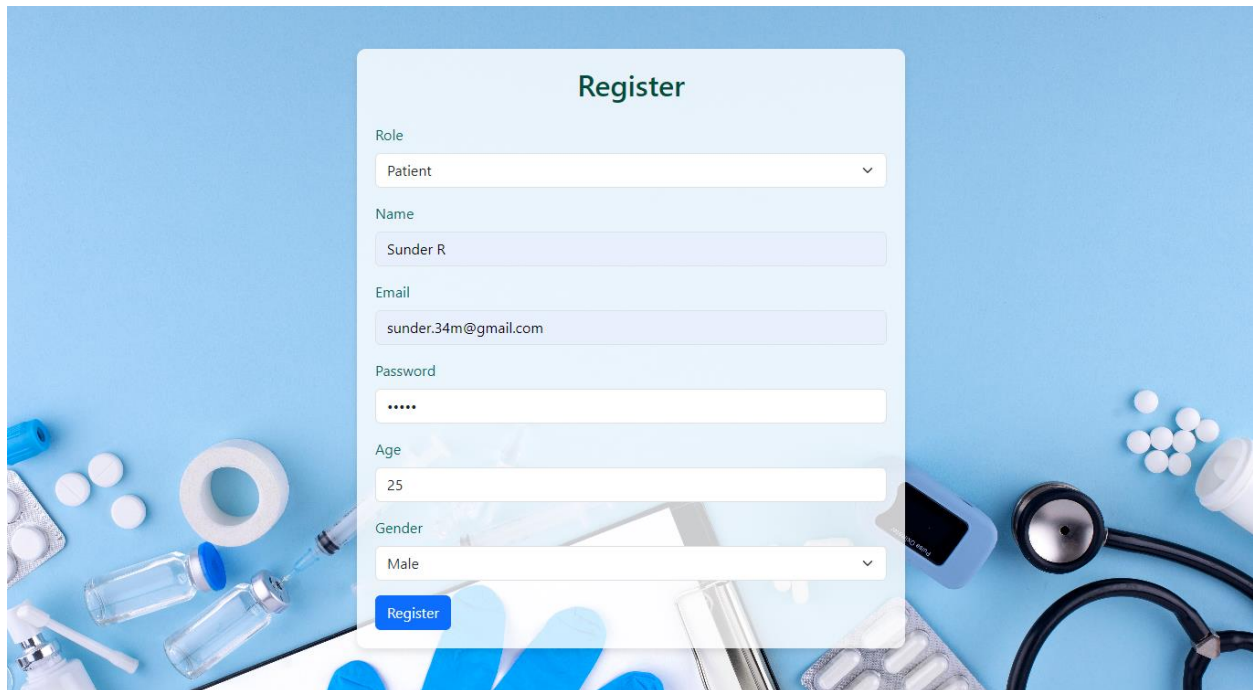
Home Page:



Description:

homepage for Med Track, featuring a navigation bar with options for user registration and login. The page includes a hero section promoting the platform's main purpose—helping users manage their health records and appointments—along with an about section detailing key features like easy appointments, a doctor’s portal, and secure medical records management.

Register Page (Patient):

The image shows a 'Register' form overlay on a light blue background with medical-themed items like pills, a stethoscope, and a syringe. The form is titled 'Register' in a bold, dark green font. It contains several input fields: a dropdown menu for 'Role' with 'Patient' selected, a text field for 'Name' containing 'Sunder R', a text field for 'Email' containing 'sunder.34m@gmail.com', a password field for 'Password' with masked characters '*****', a text field for 'Age' containing '25', and a dropdown menu for 'Gender' with 'Male' selected. A blue 'Register' button is positioned at the bottom of the form.

Register

Role
Patient

Name
Sunder R

Email
sunder.34m@gmail.com

Password

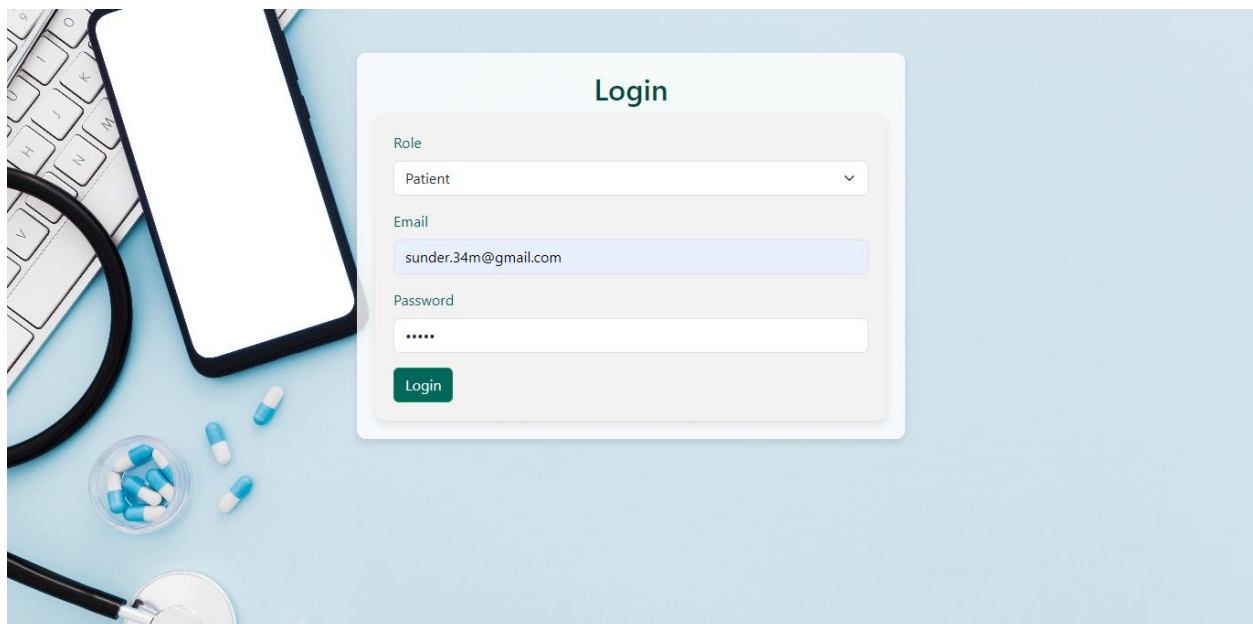
Age
25

Gender
Male

Register

Patient Registration: This registration page allows patients to sign up by entering personal details such as name, email, age, and gender. Once registered, patients can access Med Track to manage their appointments and medical records efficiently.

Login Page:

The image shows a 'Login' form overlay on a light blue background with medical-themed items like a keyboard, a stethoscope, and pills. The form is titled 'Login' in a bold, dark green font. It contains three input fields: a dropdown menu for 'Role' with 'Patient' selected, a text field for 'Email' containing 'sunder.34m@gmail.com', and a password field for 'Password' with masked characters '*****'. A green 'Login' button is positioned at the bottom of the form.

Login

Role
Patient

Email
sunder.34m@gmail.com

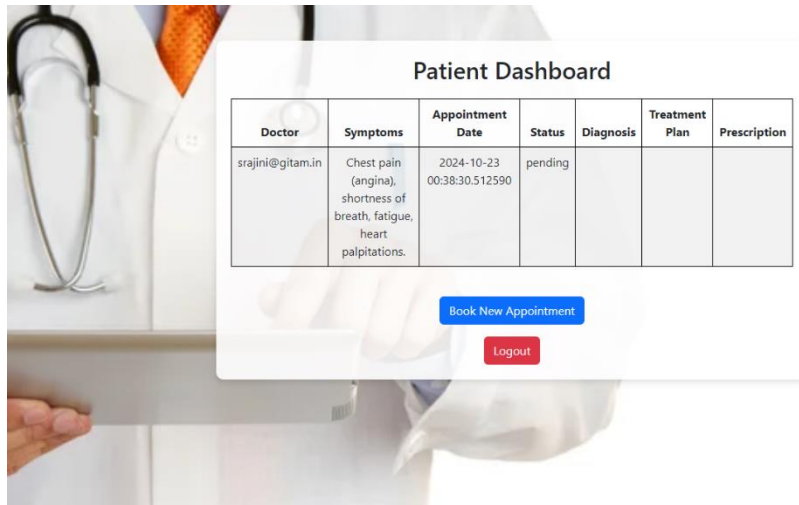
Password

Login

Patient Login:

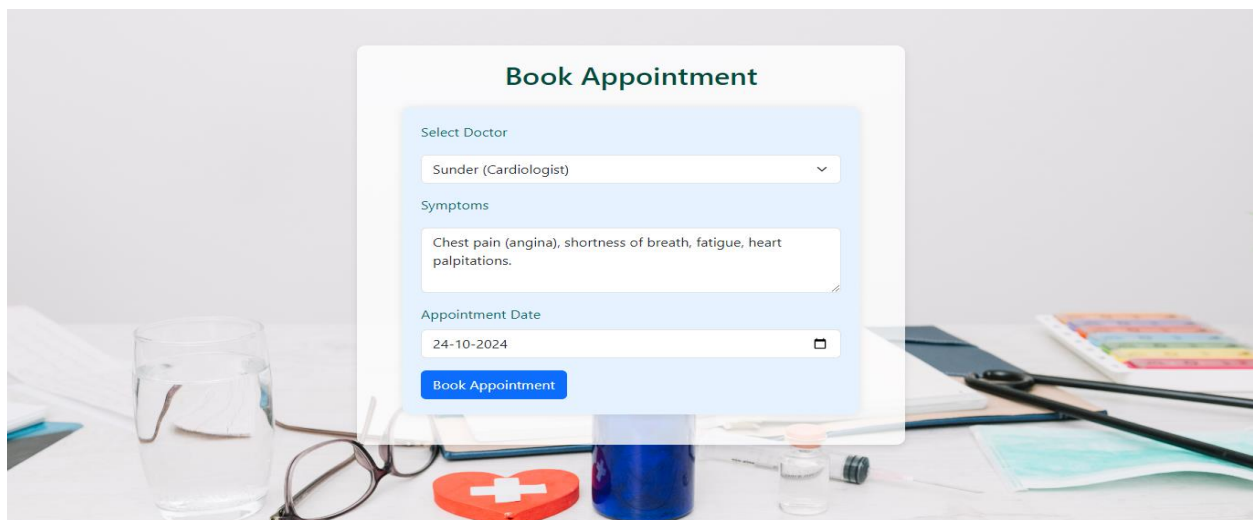
This login page allows patients to securely sign in by selecting their role and entering their email and password. Once authenticated, patients can access their personalized dashboard to manage appointments and view medical records.

Patient dashboard page:



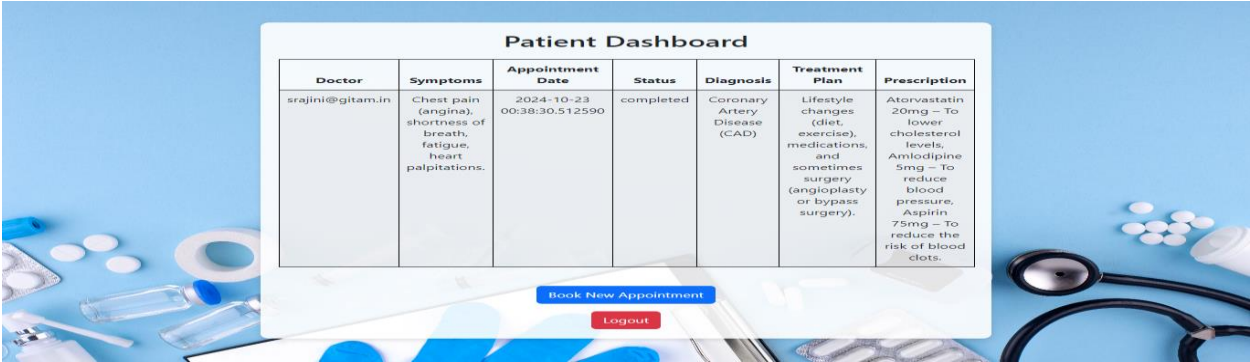
Patient Dashboard: (Before Appointment)

The Patient Dashboard displays detailed appointment information, including the doctor's name, symptoms, appointment date, and the current status of each appointment. Patients can also view diagnoses, treatment plans, and prescriptions, as well as book new appointments or log out from the system.



Book Appointment Page:

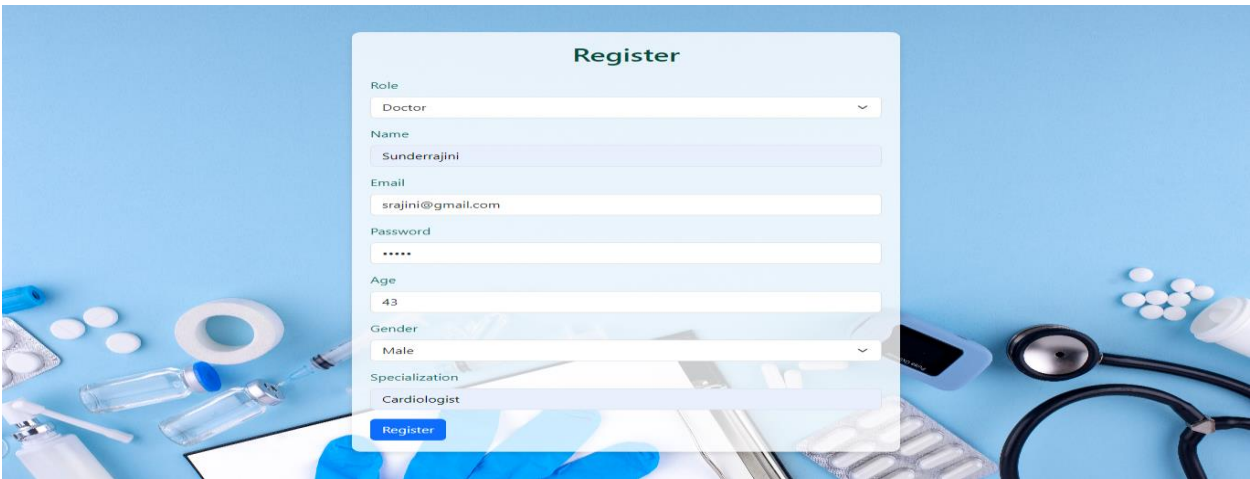
The Book Appointment page allows patients to schedule appointments by selecting a doctor, describing their symptoms, and choosing a preferred date. It provides an intuitive form to facilitate the booking process, ensuring that patients can quickly and efficiently manage their healthcare appointments.



Patient Dashboard: (After Appointment)

The Patient Dashboard displays detailed appointment information, including the doctor's name, symptoms, appointment date, and the current status of each appointment. Patients can also view diagnoses, treatment plans, and prescriptions, as well as book new appointments or log out from the system.

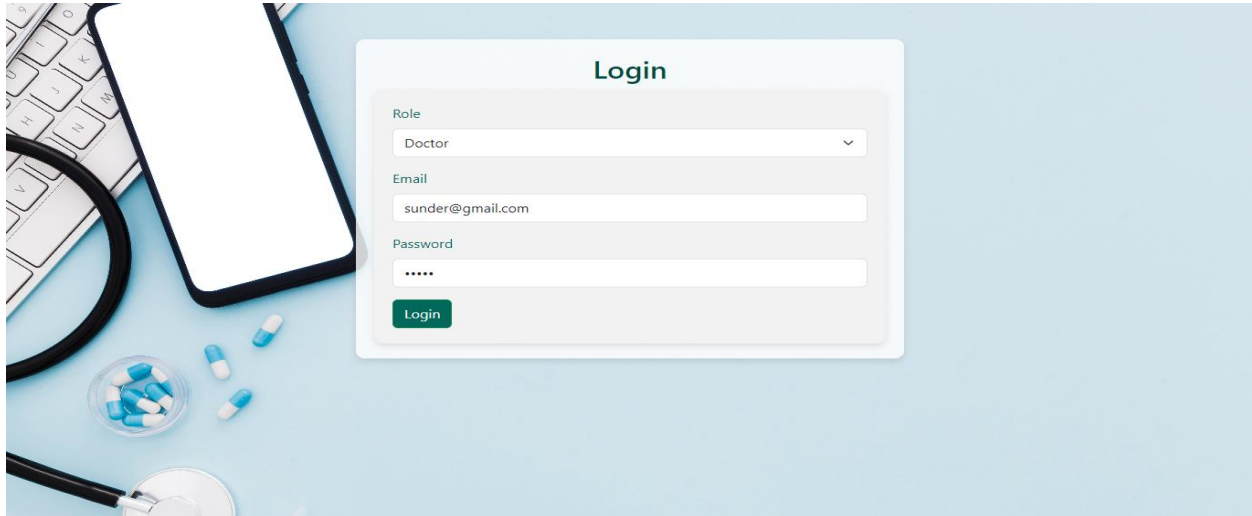
Register Page (Doctor):



Doctor Registration:

For doctors, the registration form includes an additional specialization field that appears when the doctor role is selected. Doctors can register to access patient management features and provide diagnoses through Med Track.

Login Page (Doctor):

A login form titled "Login" is displayed on a light blue background with medical-themed items like a stethoscope, keyboard, and pills. The form has three input fields: "Role" (a dropdown menu with "Doctor" selected), "Email" (containing "sunder@gmail.com"), and "Password" (with masked characters "*****"). A green "Login" button is at the bottom of the form.

Login

Role
Doctor

Email
sunder@gmail.com

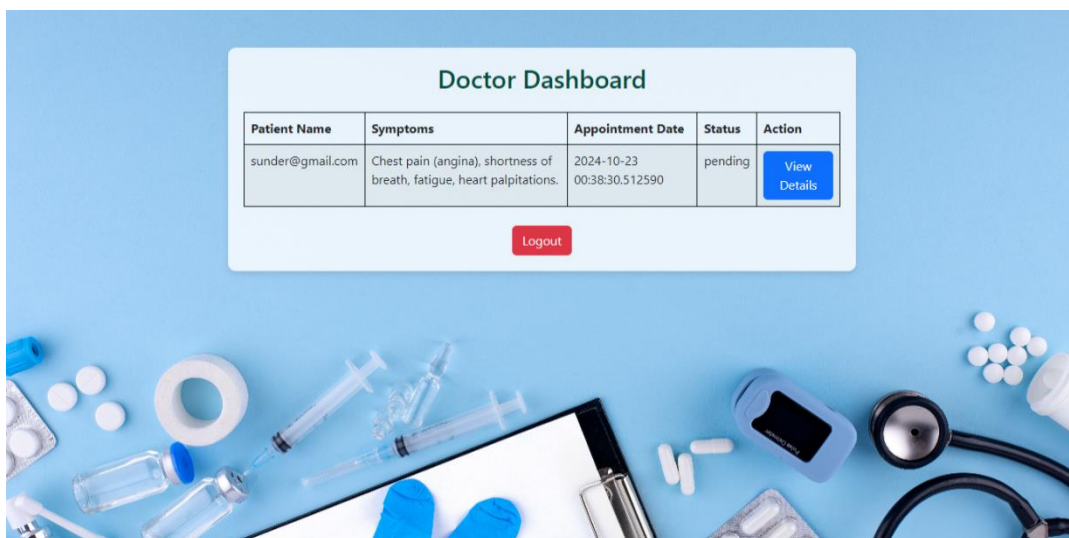
Password

Login

Doctor Login:

For doctors, the login form requires role selection, email, and password. Upon successful login, doctors can access the platform to manage patient records, submit diagnoses, and view appointments.

Doctor Dashboard Page:(Before Consulting)

A "Doctor Dashboard" interface is shown on a light blue background with medical-themed items. It features a table with patient records and a "Logout" button.

Doctor Dashboard

Patient Name	Symptoms	Appointment Date	Status	Action
sunder@gmail.com	Chest pain (angina), shortness of breath, fatigue, heart palpitations.	2024-10-23 00:38:30.512590	pending	View Details

Logout

Doctor Dashboard provides an organized view of upcoming appointments, displaying patient details, symptoms, and appointment status in a clean table format. Doctors can click "View Details" to update diagnosis, treatment plans, and prescriptions for each appointment. The interface is user-friendly, with a logout option for easy session management.

View Appointment

Doctor Email:	sunder.34m@gmail.com
Symptoms:	Rashes, Redness on the skin, Swelling, Itching Sensation
Appointment Date:	2024-10-23 00:19:16.434355
Status:	pending

Diagnosis

Seborrheic keratosis, or a benign skin lesion

Treatment Plan

5 days 2 times a day

Prescription

Cimzia, Cosentyx.

Submit Diagnosis

View Appointment Page:

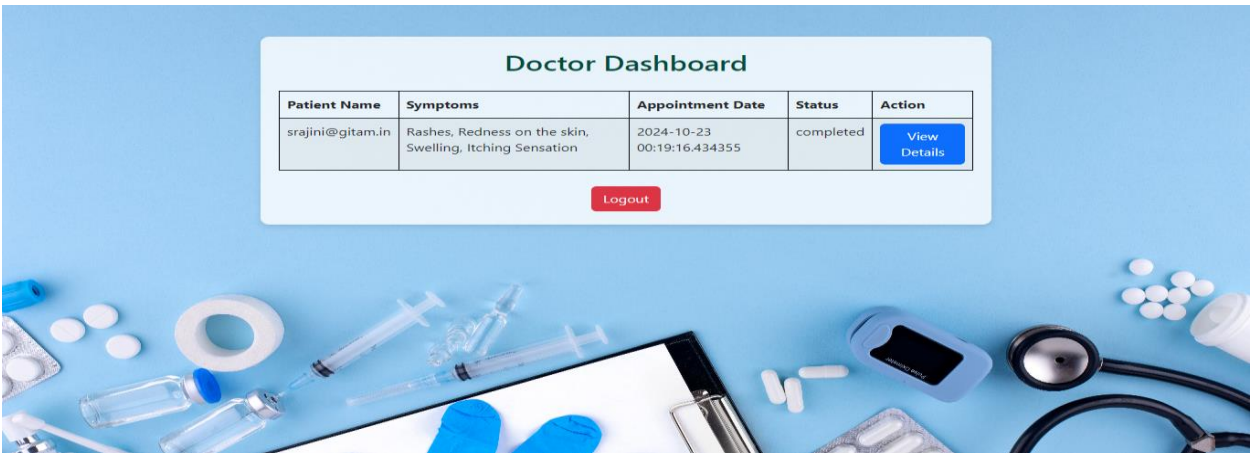
The View Appointment page provides a detailed summary of a patient’s scheduled appointment, including information about the doctor, symptoms, and the appointment status. It also includes a form for doctors to input the diagnosis, treatment plan, and prescription for the patient after the consultation. This page is designed to streamline the process of updating medical records and managing patient care in an organized and efficient way.

Doctor Dashboard Page:(After Consulting)

Doctor Dashboard

Patient Name	Symptoms	Appointment Date	Status	Action
srajini@gitam.in	Rashes, Redness on the skin, Swelling, Itching Sensation	2024-10-23 00:19:16.434355	completed	<div>View Details</div>

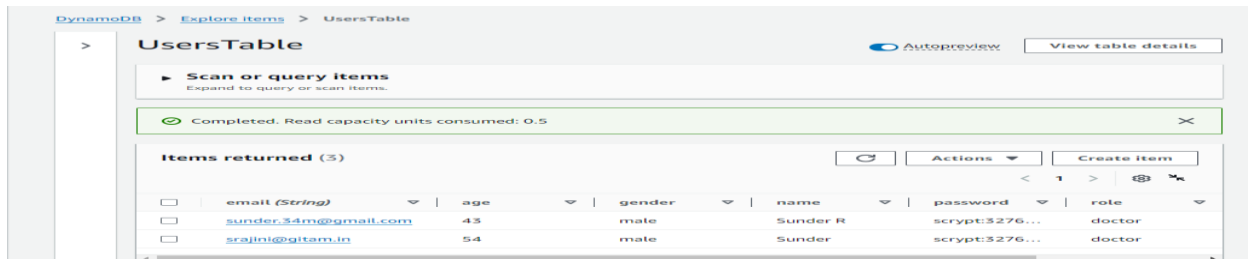
Logout



Doctor Dashboard provides an organized view of upcoming appointments, displaying patient details, symptoms, and appointment status in a clean table format. Doctors can click "View Details" to update diagnosis, treatment plans, and prescriptions for each appointment. The interface is user-friendly, with a logout option for easy session management.

DynamoDB up dations:

1. Users table:



DynamoDB > Explore items > UsersTable

UsersTable

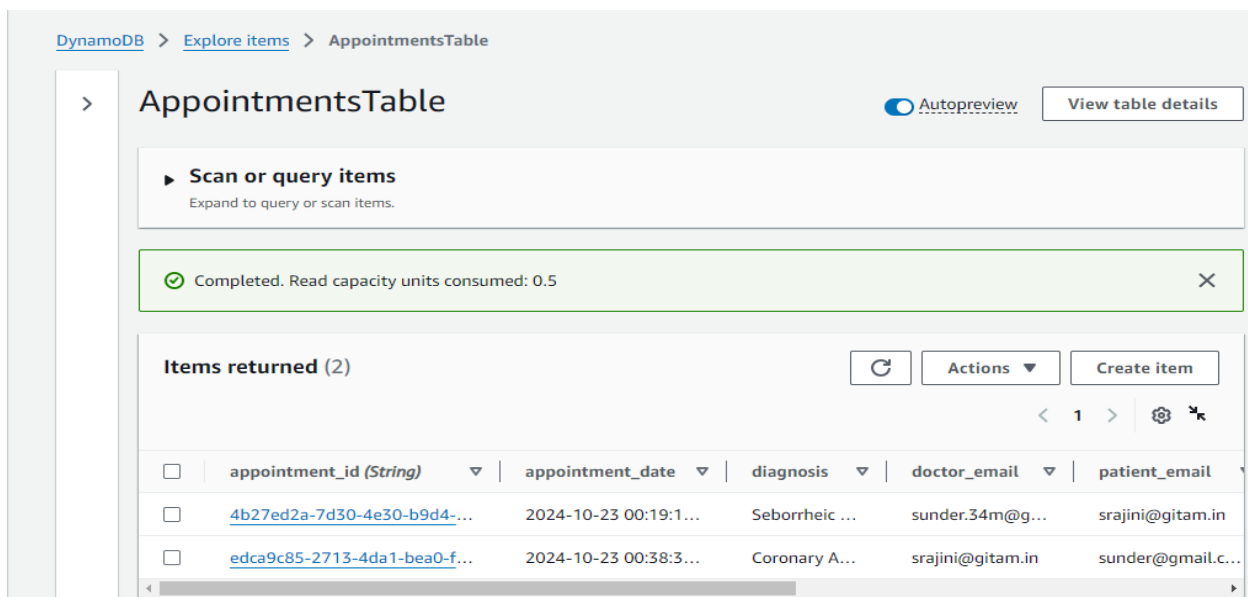
Scan or query items

Completed. Read capacity units consumed: 0.5

Items returned (3)

	email (String)	age	gender	name	password	role
<input type="checkbox"/>	sunder.34m@gmail.com	43	male	Sunder R	script:3276...	doctor
<input type="checkbox"/>	srajini@gitam.in	54	male	Sunder	script:3276...	doctor

2.Appointments table:



DynamoDB > Explore items > AppointmentsTable

AppointmentsTable

Scan or query items

Completed. Read capacity units consumed: 0.5

Items returned (2)

	appointment_id (String)	appointment_date	diagnosis	doctor_email	patient_email
<input type="checkbox"/>	4b27ed2a-7d30-4e30-b9d4-...	2024-10-23 00:19:1...	Seborrheic ...	sunder.34m@g...	srajini@gitam.in
<input type="checkbox"/>	edca9c85-2713-4da1-bea0-f...	2024-10-23 00:38:3...	Coronary A...	srajini@gitam.in	sunder@gmail.c...

Conclusion: The Med Track application has been successfully developed and deployed using a robust cloud-based architecture with AWS services such as EC2 for hosting, DynamoDB for data management, and IAM roles for secure access control. This platform enhances doctor-patient interactions by enabling patients to book appointments and submit diagnoses while allowing doctors to manage their schedules and access patient records efficiently. The cloud-native approach ensures seamless scalability, accommodating increasing user demand without compromising performance. Comprehensive testing has verified that all functionalities, from user registration to appointment management, operate smoothly. Ultimately, Med Track exemplifies the potential of cloud-based systems to address real-world challenges in the healthcare sector, improving communication and user experience.