**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.

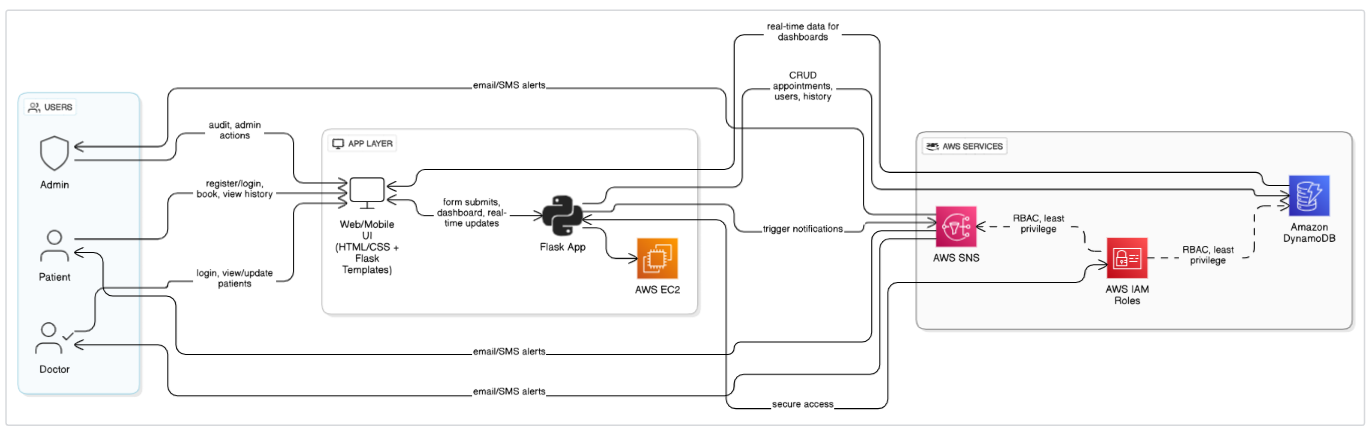
**Scenario 2: Secure User Management with IAM**

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.

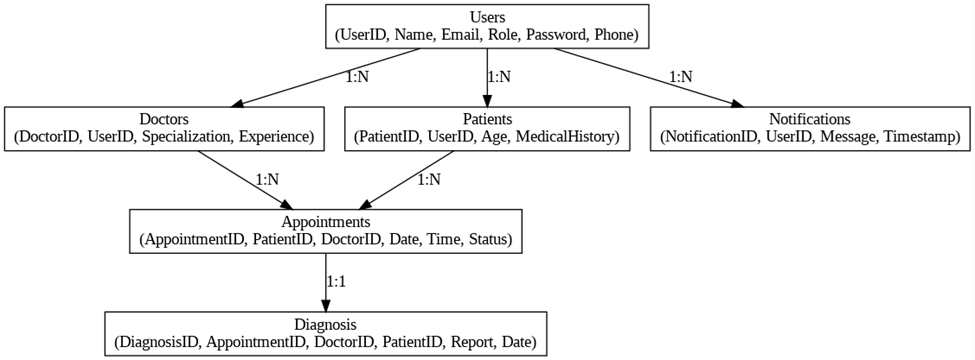
**Scenario 3: Easy Access to Medical History and Resources**

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**](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**](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**](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html)
* **AWS DynamoDB:**[**https://docs.aws.amazon.com/amazondynamodb/Introduction.html**](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html)
* **Amazon SNS:**[**https://docs.aws.amazon.com/sns/latest/dg/welcome.htm**](https://docs.aws.amazon.com/sns/latest/dg/welcome.htm)**l**
* **Git Documentation:**[**https://git-scm.com/doc**](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**](https://code.visualstudio.com/download)

**Project WorkFlow:**

**Milestone 1. Web Application Development and Setup**

* Develop the Backend Using Flask.
* Integrate AWS Services Using boto3.

**Milestone 2. AWS Account Setup and Login**

* Set up an AWS account if not already done.
* Login to AWS Management Console.

**Milestone 3. DynamoDB Database Creation and Setup**

* Create a DynamoDB Table.
* Configure Attributes for User Data and Book Requests.

**Milestone 4. SNS Notification Setup**

* Create SNS topics for book request notifications.
* Subscribe users and library staff to SNS email notifications.

**Milestone 5. IAM Role Setup**

* Create IAM Role
* Attach  Policies

**Milestone 6. EC2 Instance Setup**

* Launch an EC2 instance to host the Flask application.
* Configure security groups for HTTP, and SSH access.

**Milestone 7. Deployment using EC2**

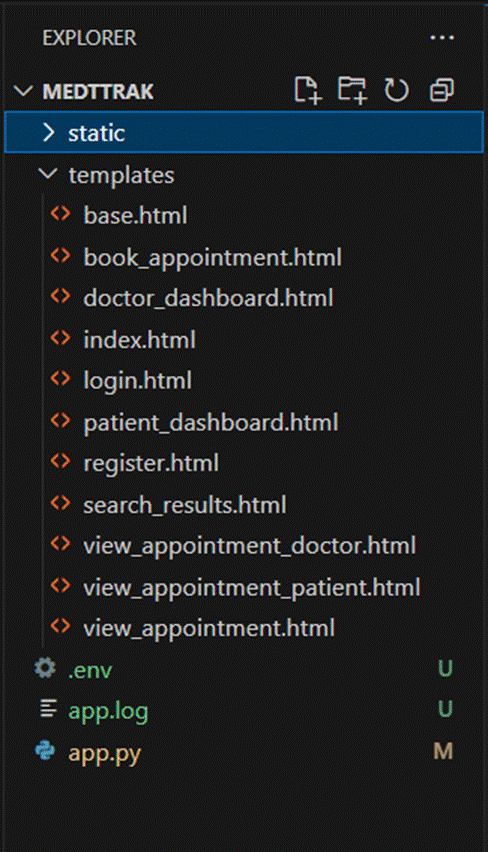
* Upload Flask Files
* Run the Flask App

**Milestone 8. Testing and Deployment**

* Conduct functional testing to verify user registration, login, book requests, and notifications.

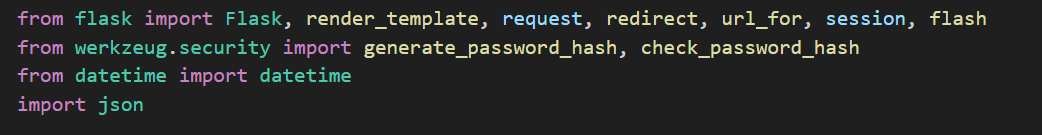
Develop the backend using Flask

* File Explorer Structure

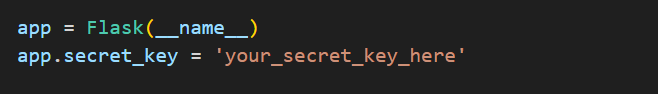


**Description of the code :**

**1.Imports:**

****

**2. Flask App Initialization:**

****

* **app = Flask(\_\_name\_\_): Starts the Flask application.**
* **app.secret\_key: Required to safely use sessions and flashes.**

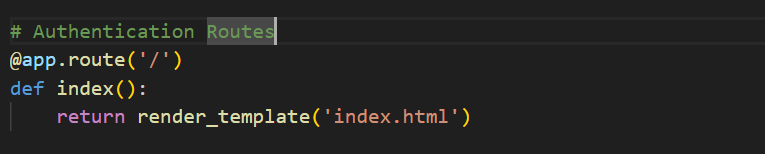
**3. Temporary In-Memory Storage:**

****

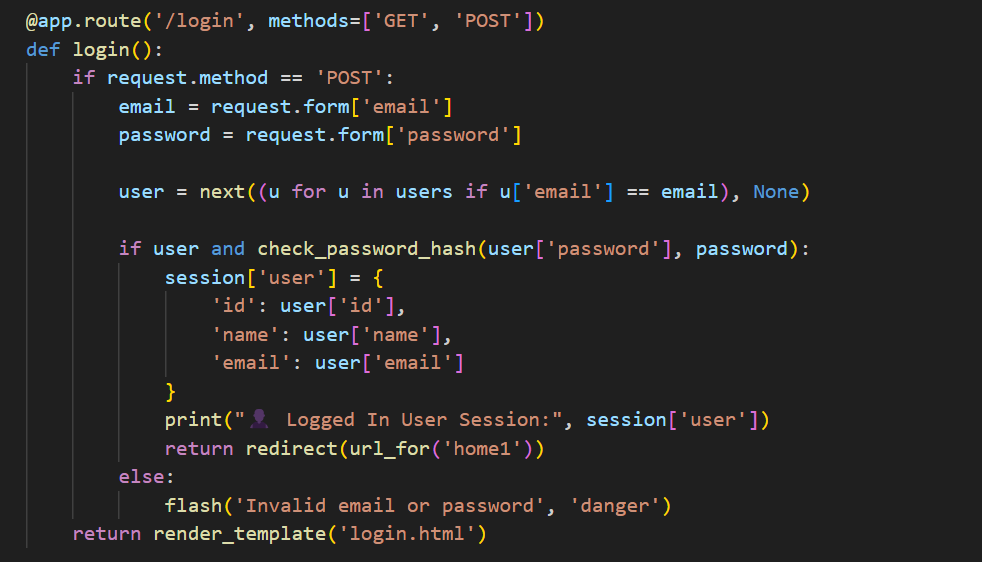
* **users: List to store registered users.  
  bookings: List to store all ticket bookings.  
  booking\_counter: A simple ID counter for bookings.**

**4. Authentication Routes:**

**4.1. Homepage /:**

****

**4.2. Login /login:**

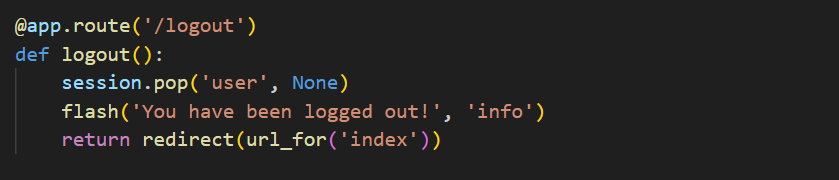
****

* **If GET: Show the login page.**
* **If POST: Validate user login.**
* **Check if the email exists and password is correct.**
* **If success, save user info in session.**
* **If wrong, flash an "Invalid" message.**

**4.3. Signup /signup:**

* **If GET: Show signup form.**
* **If POST:**
* **Collect user details.**
* **Check if email already exists.**
* **Save the user with hashed password into users.**
* **Redirect to login after successful registration.**

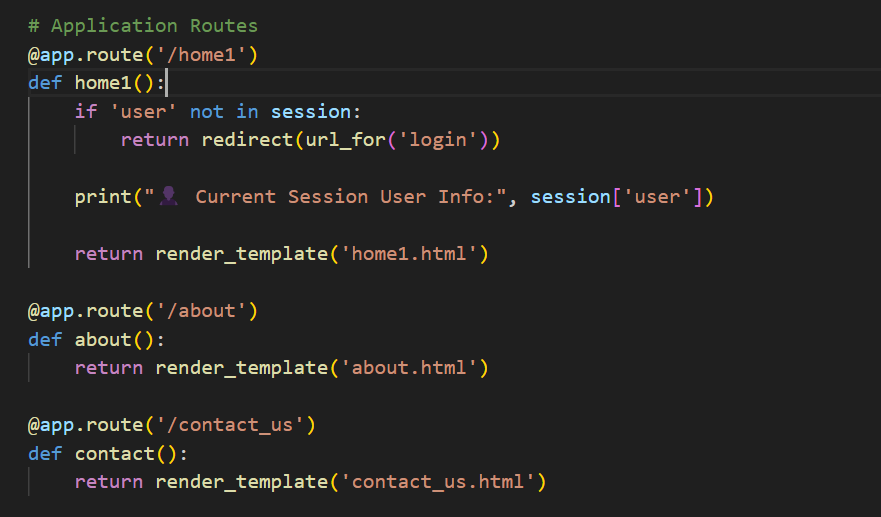
**4.4. Logout /logout:**

****

* **Remove the user from session (log out).**
* **Flash a message.**
* **Redirect to homepage.**

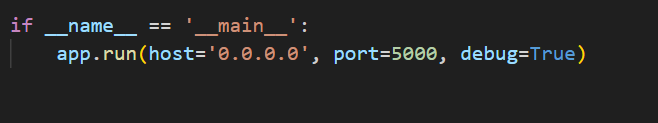
**5. Main Application Pages:**

**5.1. Home After Login /home1, About Page /about, Contact Page /contact\_us**

****

* **Check if the user is logged in.**
* **If yes, show home1.html.**
* **If no, redirect to login page.**
* **Show About page.**
* **Show Contact Us page.**

**6.Running the Flask App:**

****

* **Starts the app on your machine at http://localhost:5000.**
* **debug=True means it auto-reloads when you change the code.**