**Med Track: 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. 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.

**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.

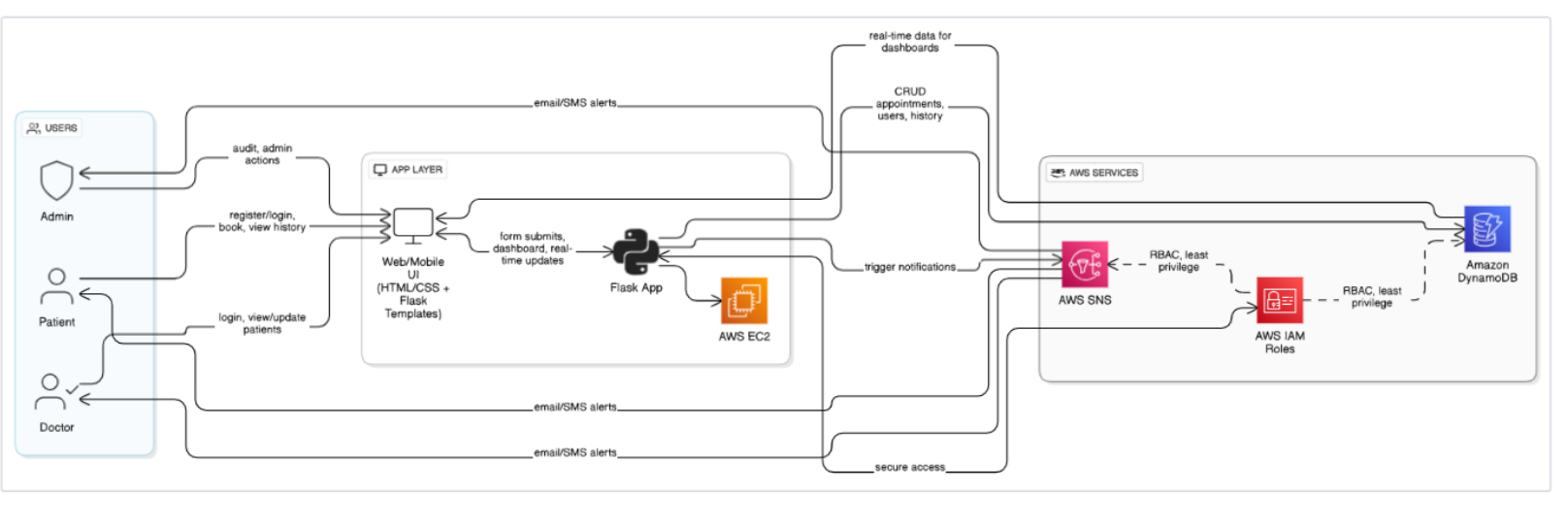
**Scenario 2**: **Secure User Management with IAM**

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.

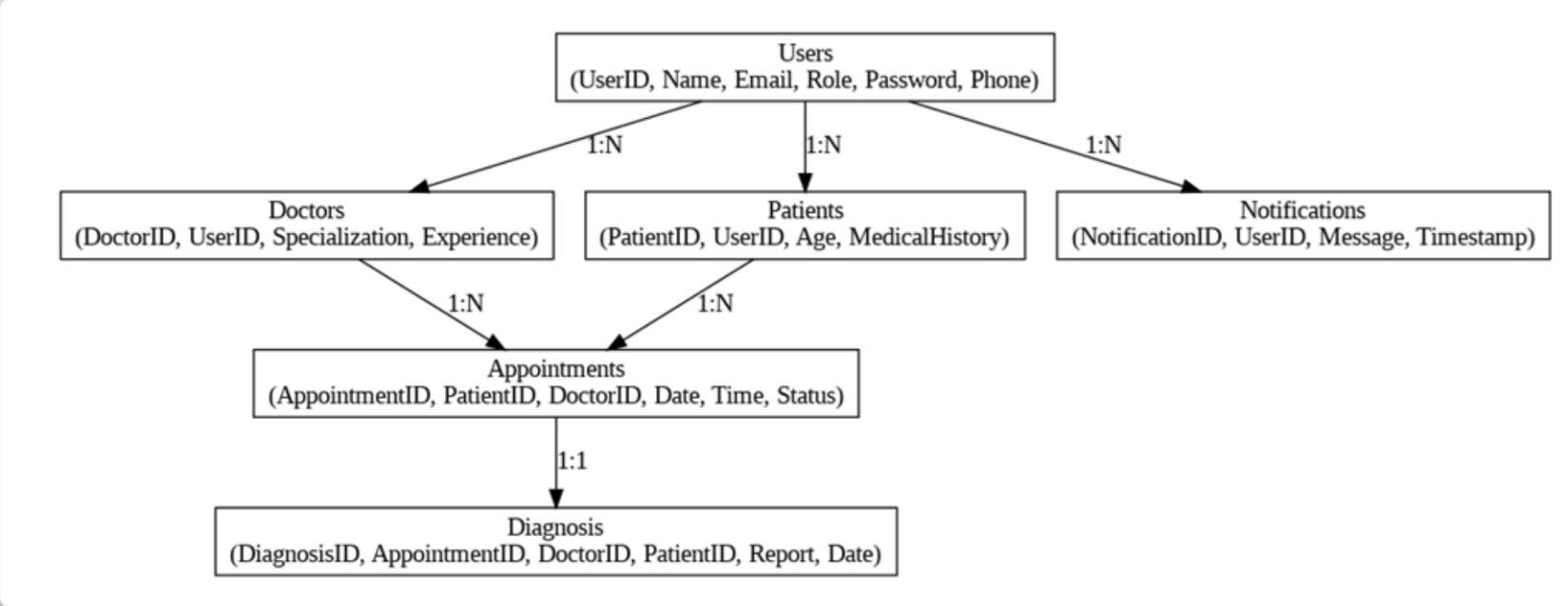
**Scenario 3**: **Easy Access to Medical History and Resources**

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:

## 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](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html)
* Amazon SNS:   
  <https://docs.aws.amazon.com/sns/latest/dg/welcome.htm>l
* 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 Work Flow**

**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**

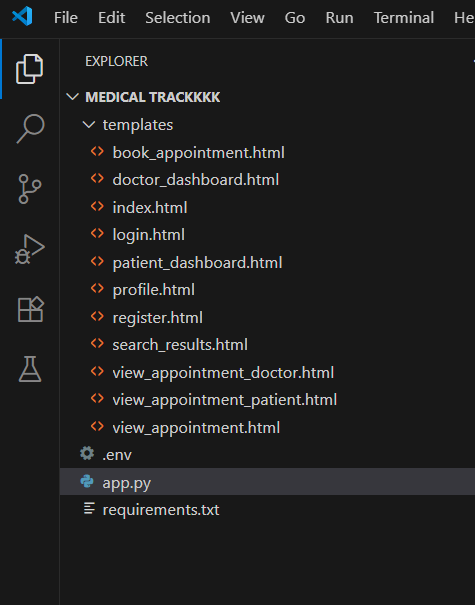
* Upload Flask Files
* Run the Flask App

**Milestone 8. Testing and Deployment**

* Conduct functional testing to verify user registration, login, book requests, and notifications.
  + - up for an AWS account and configure billing settings.

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

* **Develop the Backend Using Flask**.



Description:

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.

## Description of the code :

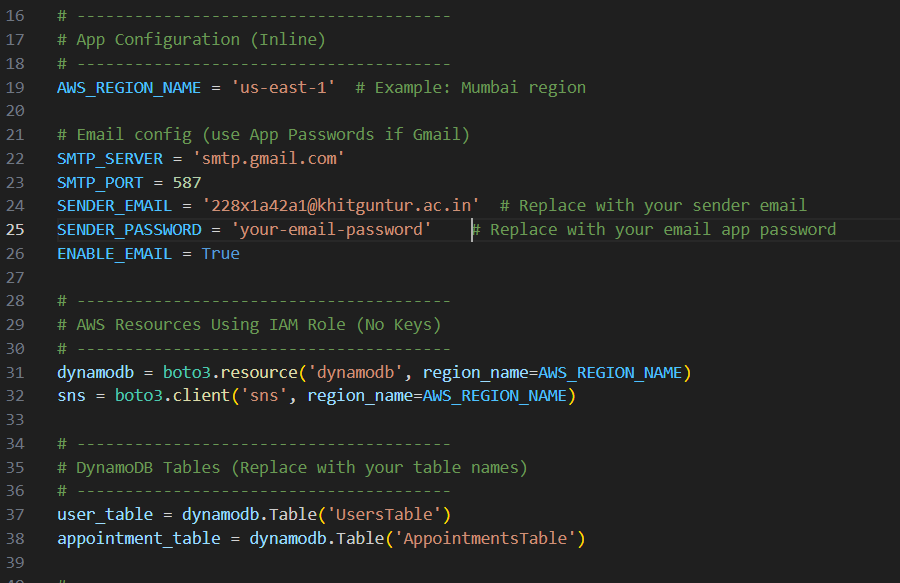
### Flask App Initialization

### 

**Description:** import essential libraries including Flask utilities for routing, Boto3 for DynamoDB Operations , SMTP and email modules for sending mails and other necessary modules.

**Description:** initialize the Flask application instance using Flask( name ) to start building the web app.

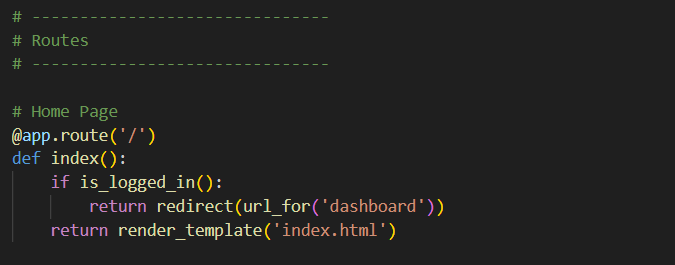
* **Integrate AWS Services Using boto3**



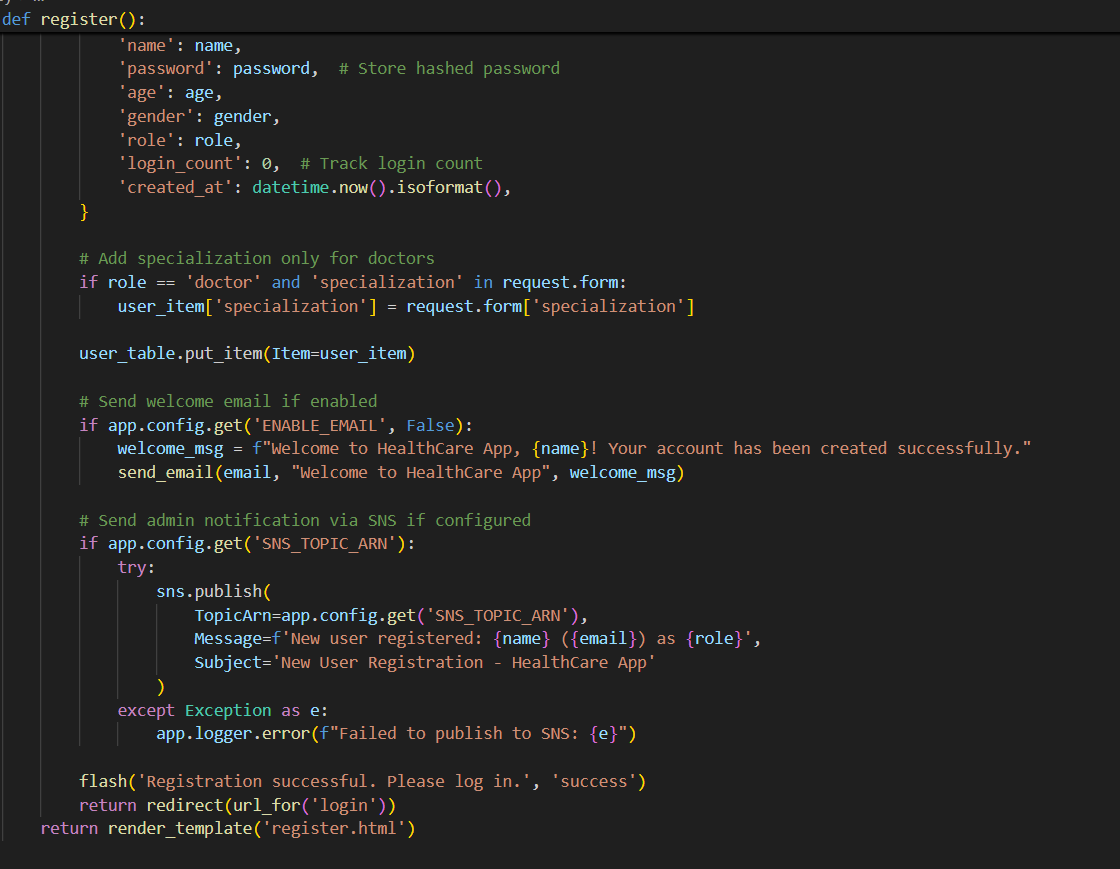
**Description:** initialize the DynamoDB resource for the us-east-1 region and set up access to the Users and Appointments tables for storing user details and appointments.

Configure **SNS** to send notifications. 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.

**Routes for Web Pages**

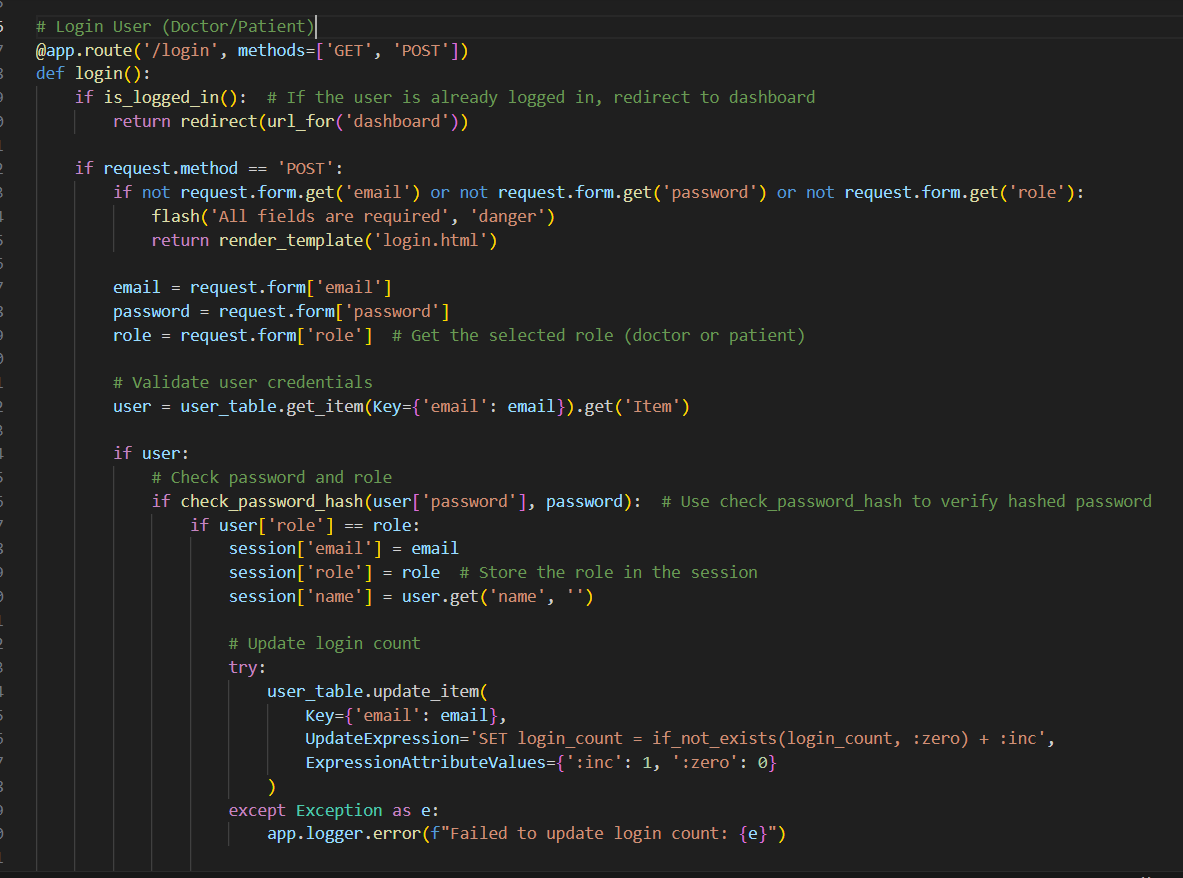
* **Home Page Routing:**
* 
* **Description:** define the home route / to automatically redirect users to the dashboard page when the user is logged in.
* **Register Route:**

****

****

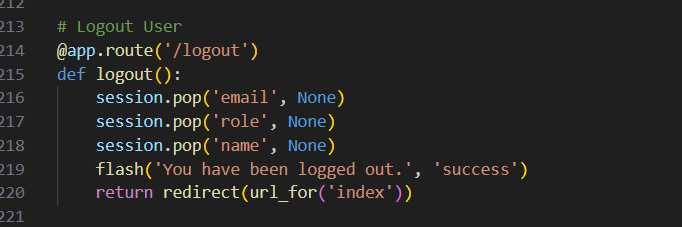
**Description:** This route handles user registration for doctors and patients. It validates form data, checks for duplicate emails, stores user info in DynamoDB, and sends optional email or SNS notifications. Passwords are securely hashed. Specialization is added for doctors. Flash messages provide feedback, and register.html is rendered for input or errors.

**Login Route:**

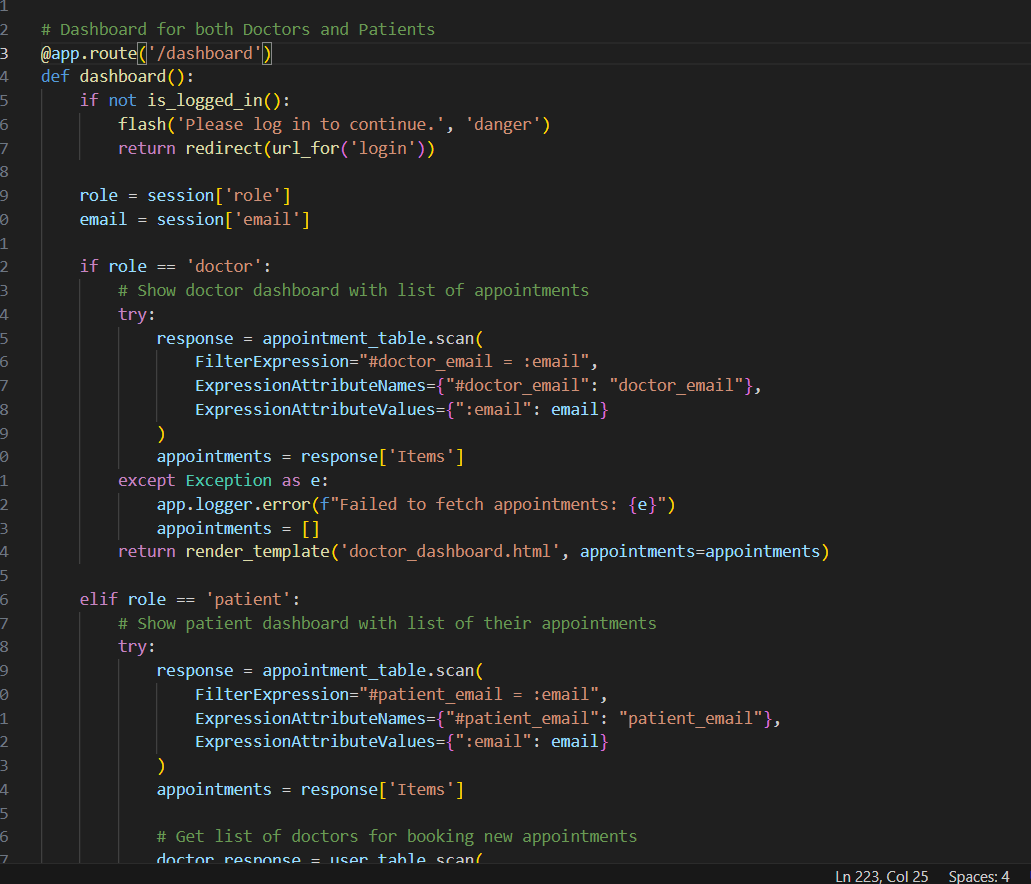


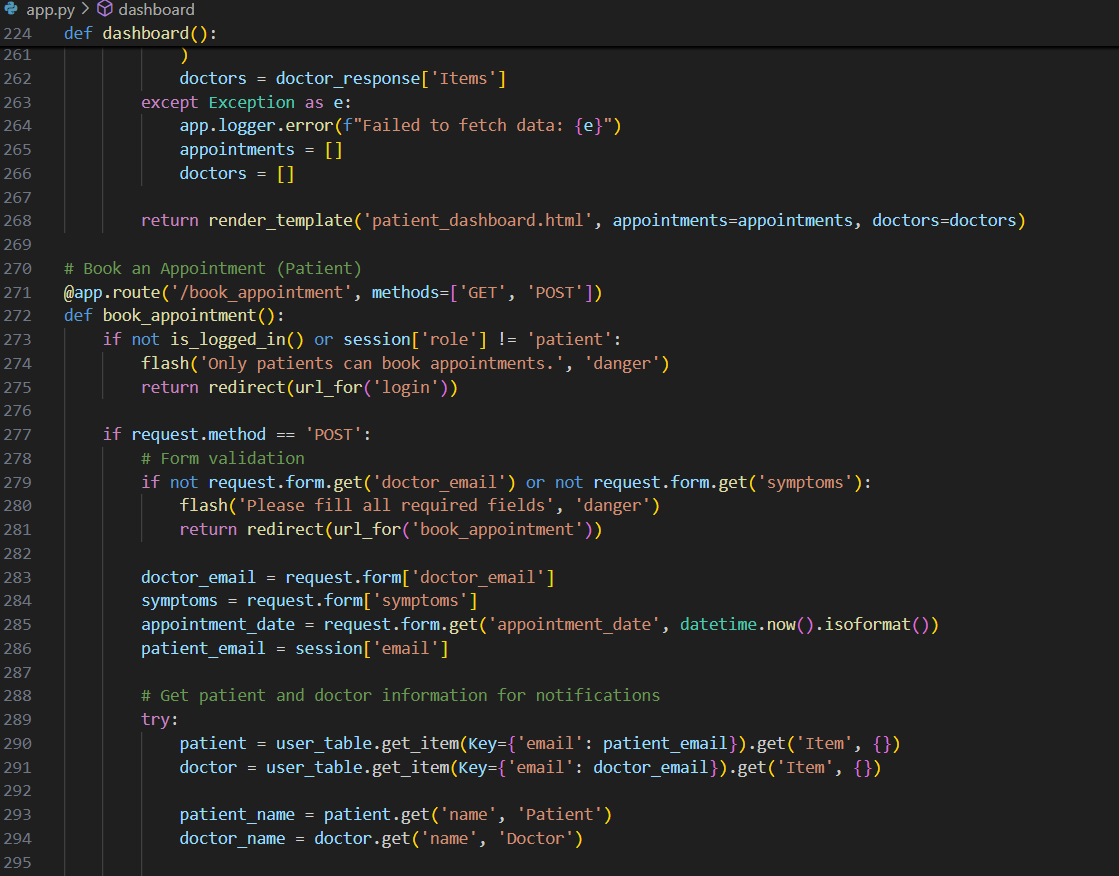


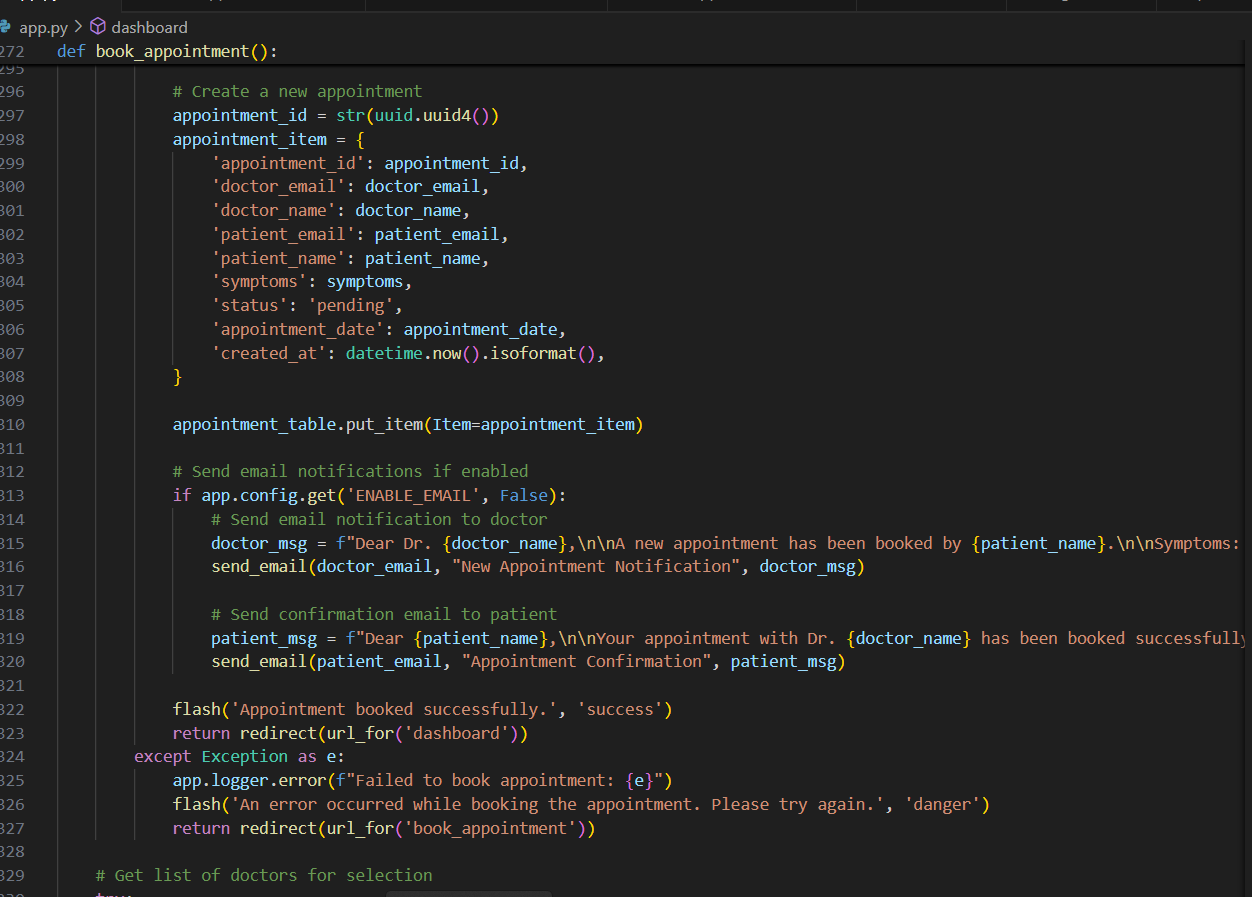
**Description:** This route handles login for doctors and patients. It validates email, password, and role, then verifies user credentials from DynamoDB. On success, it stores session data and updates login count. Feedback is provided via flash messages. Invalid inputs or mismatched roles trigger appropriate error prompts and return the login form.

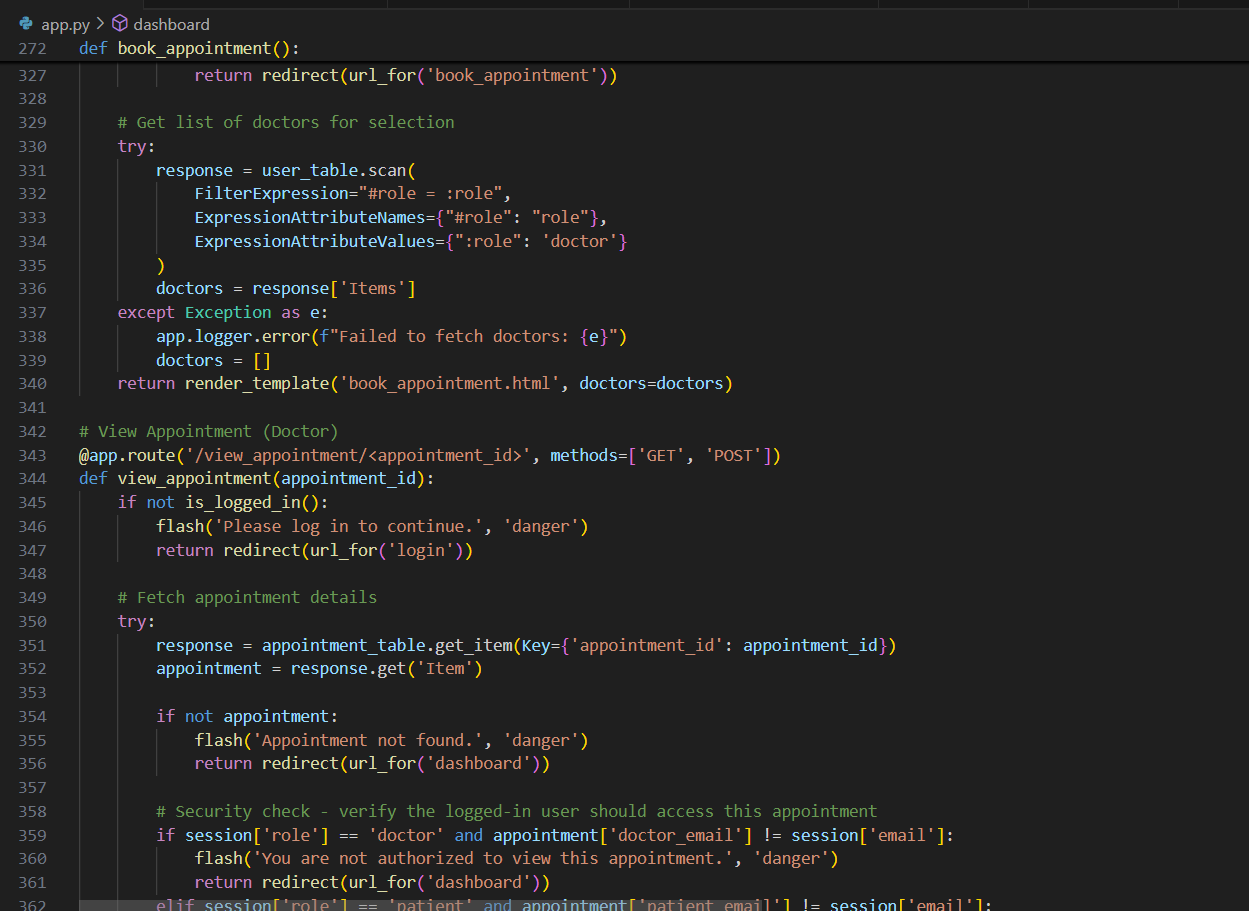


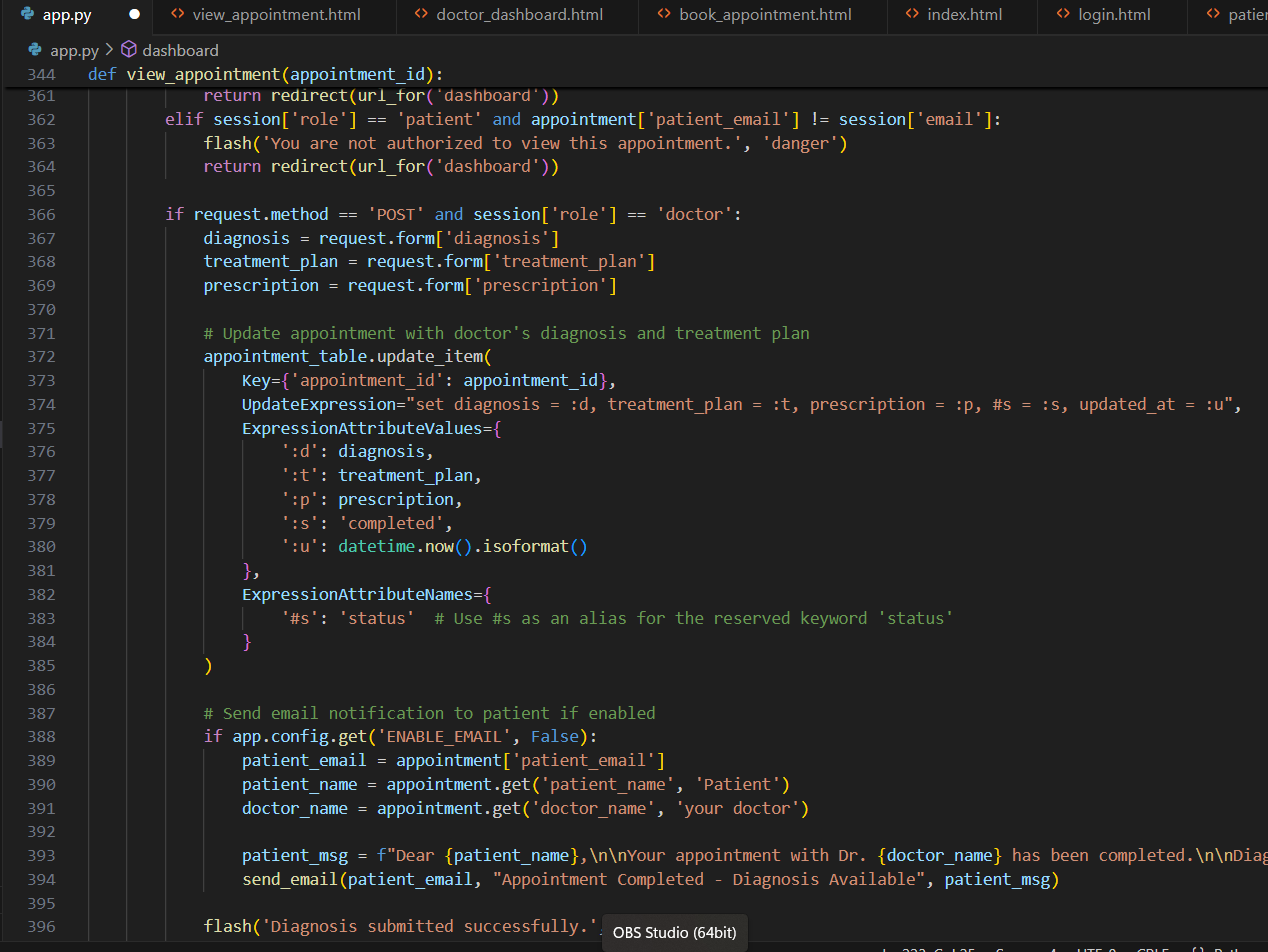
**Description:** This route securely logs out the current user by clearing session variables such as email, role, and name. It uses Flask's flash to display a confirmation message and redirects the user back to the homepage (index). This ensures proper session termination and improves user experience after logout.

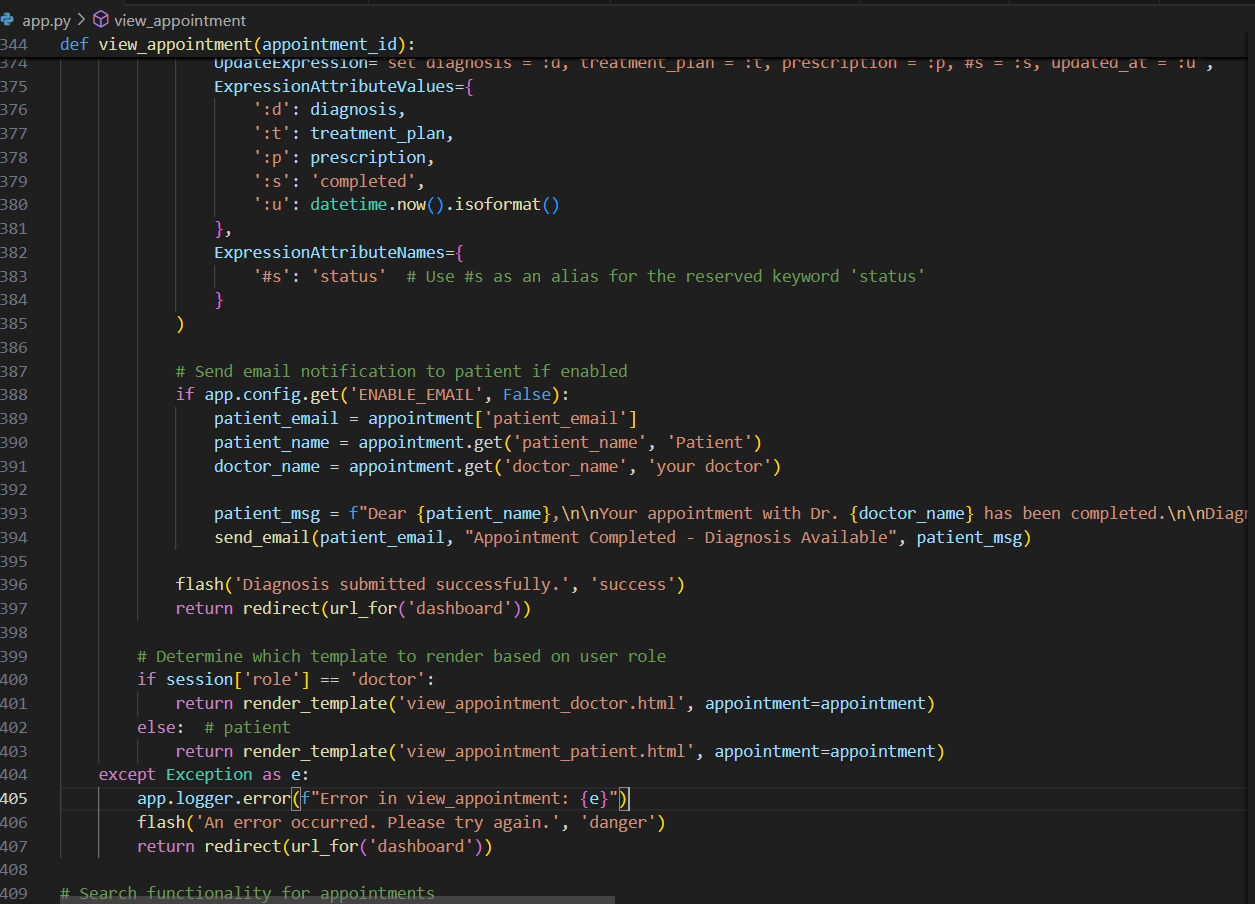






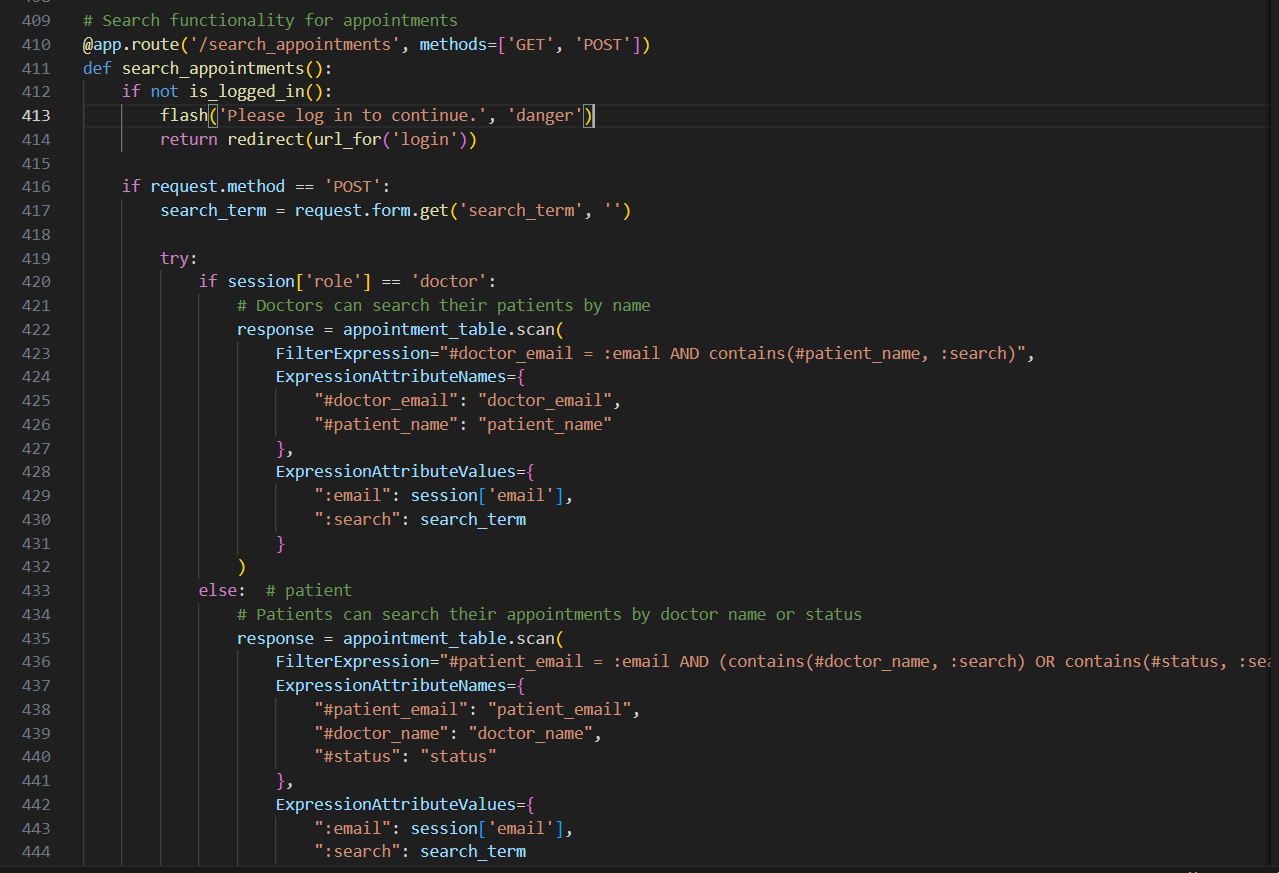






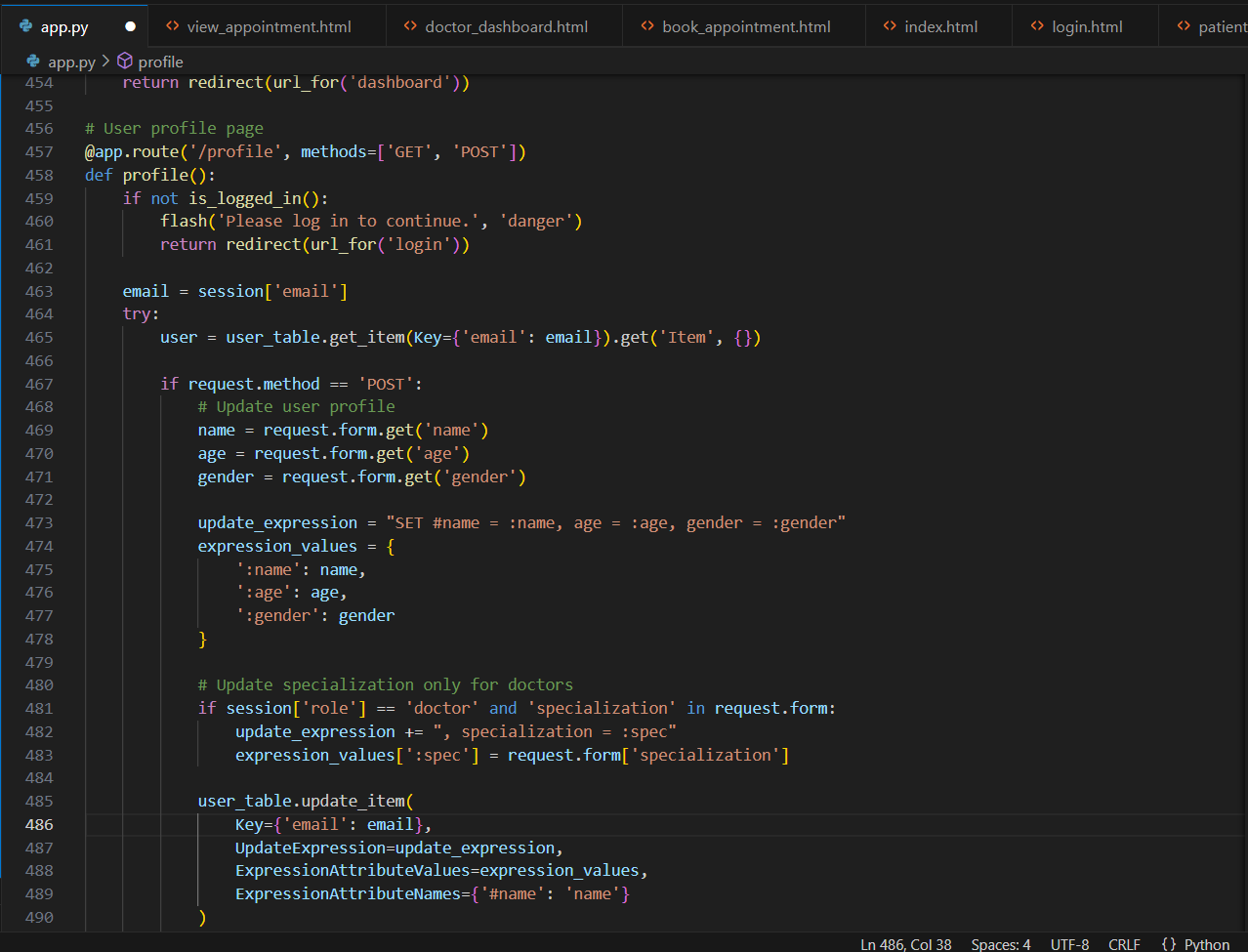
**Appointment Booking & Viewing**

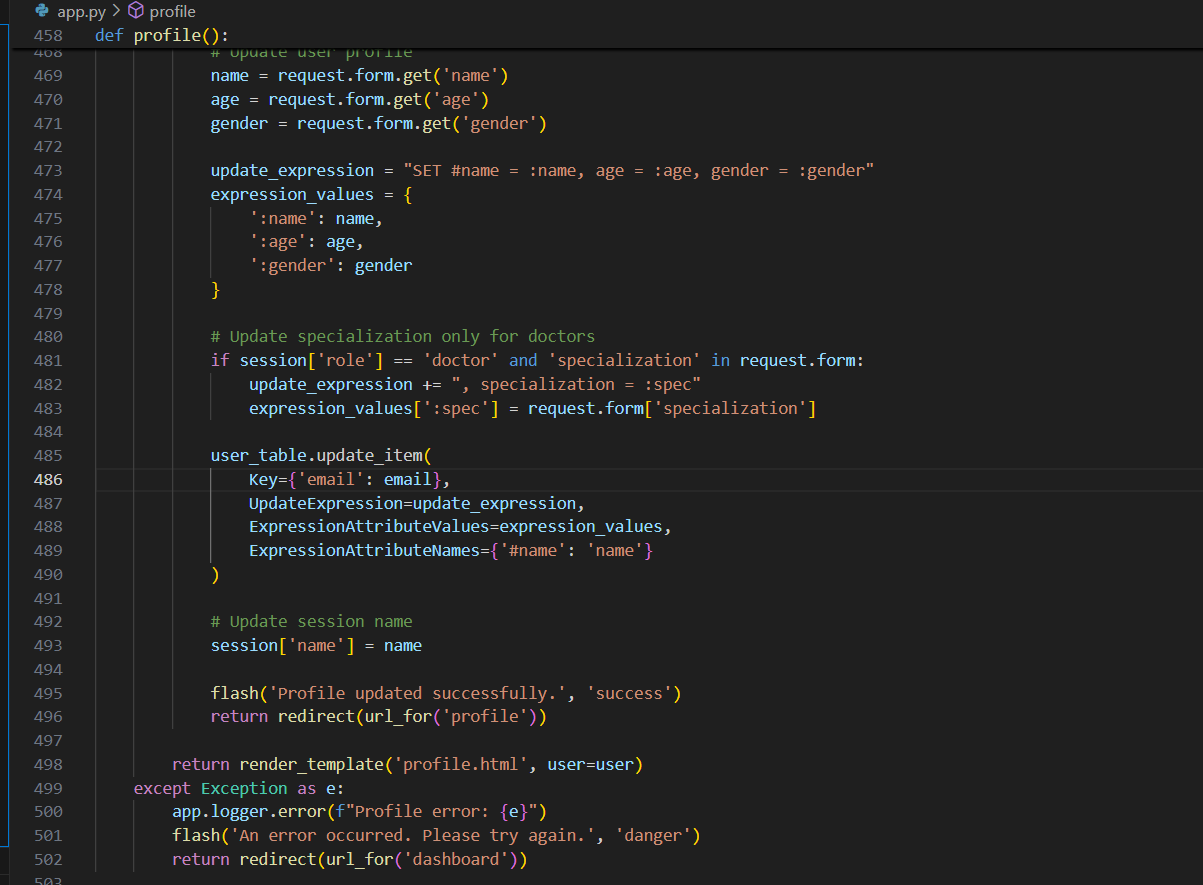
Patients can book appointments by selecting a doctor and submitting symptoms. Appointments are stored in DynamoDB and email confirmations are sent. Doctors and patients can view appointment details securely. Doctors may update status, diagnosis, and treatment plans. Notifications are sent on completion. Templates vary based on user roles.



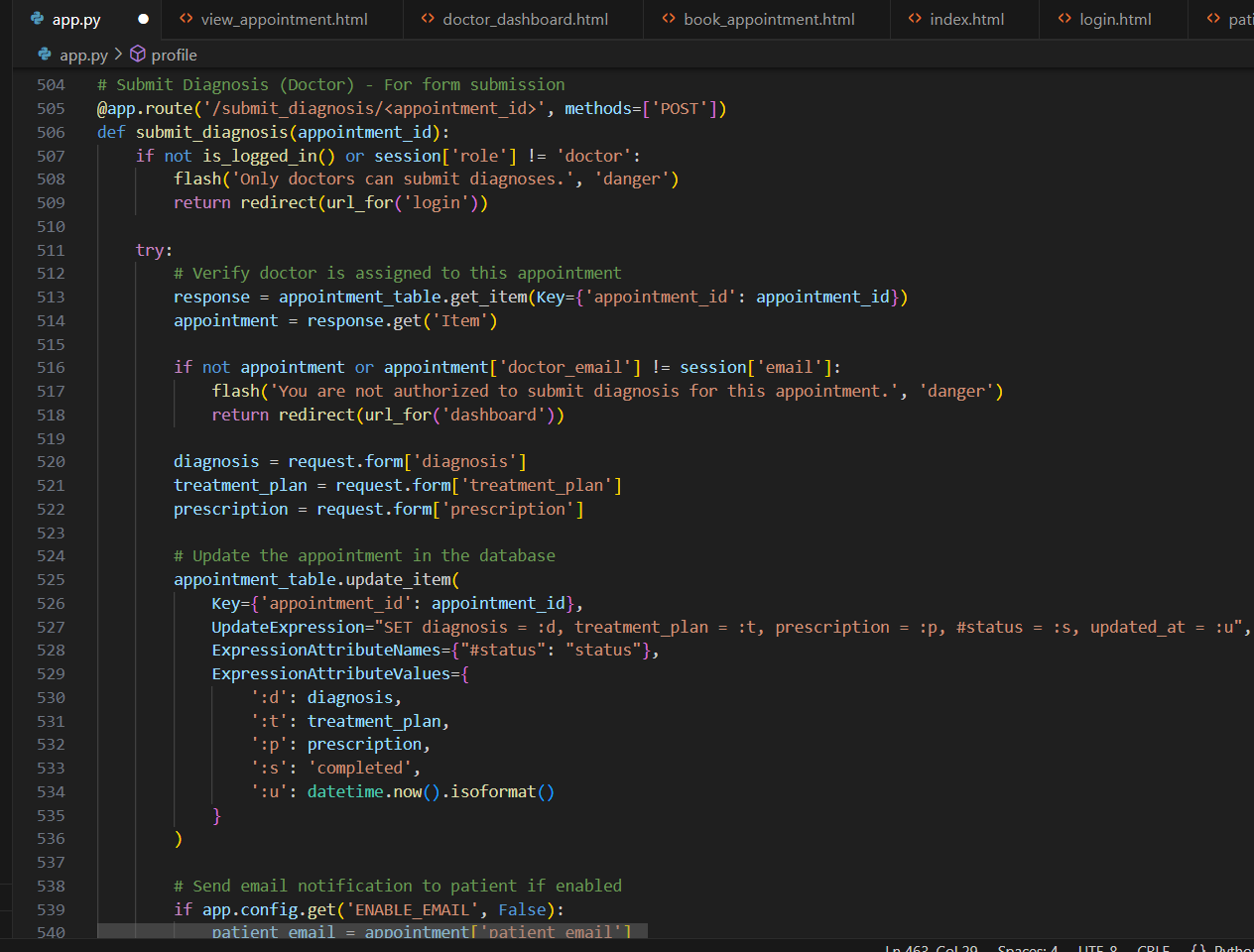


This route enables users to search appointments based on their role. Doctors can search by patient names, while patients can search by doctor names or appointment status. It uses DynamoDB filters and renders matched results in search\_results.html. If an error occurs, it logs the issue and notifies the user via flash.



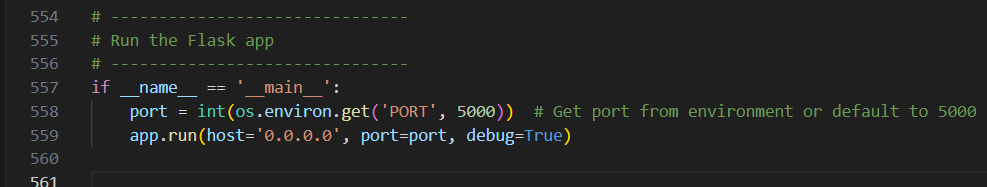


This route manages the user profile page. It retrieves the logged-in user's information from DynamoDB, displays it on the profile page, and allows updates to personal details such as name, age, gender, and (for doctors) specialization. Updates are reflected in both the database and the active session, with feedback via flash messages.





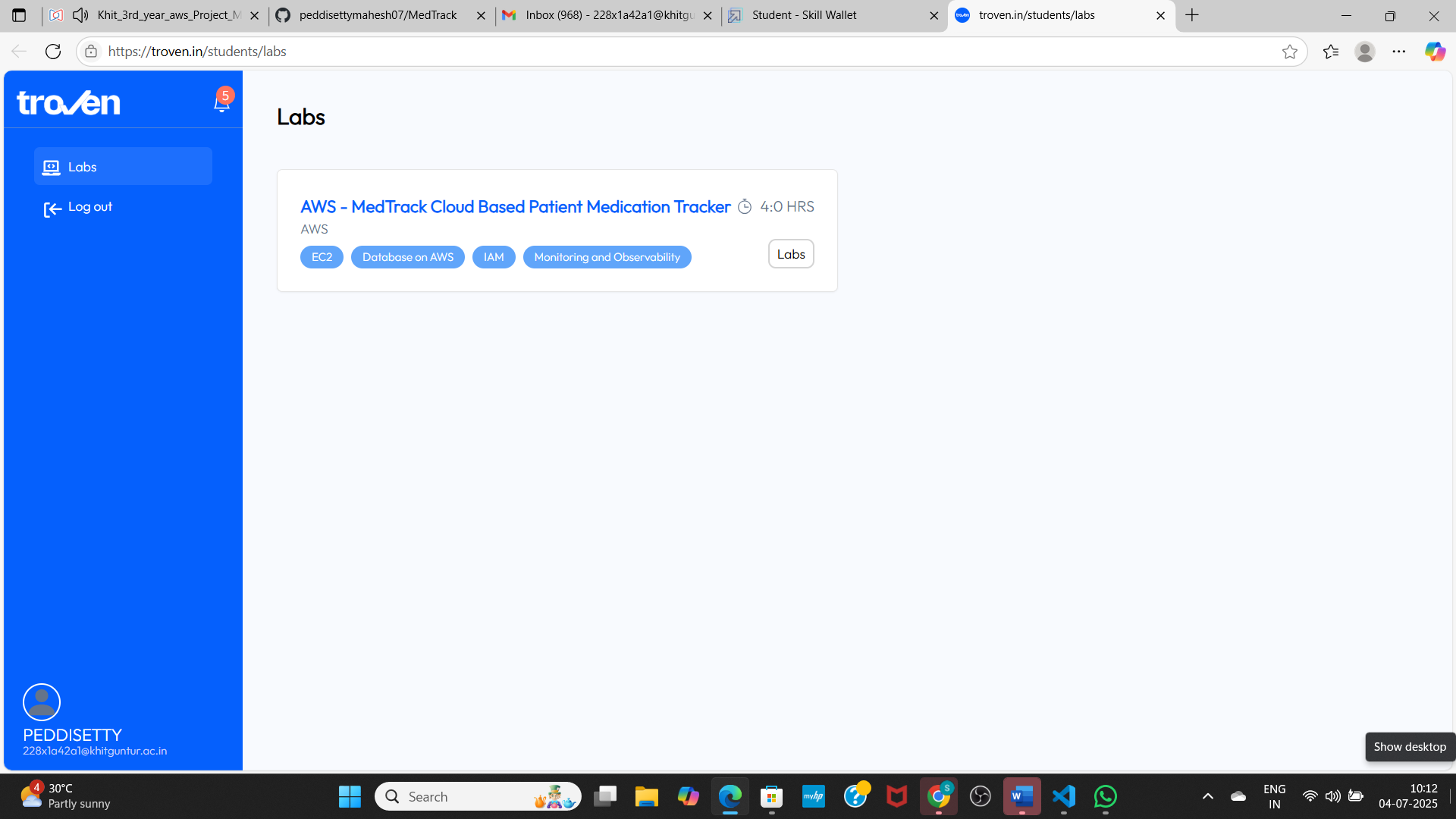
This route allows a doctor to submit a diagnosis for a specific appointment. It verifies the doctor's authorization, updates the appointment with diagnosis, treatment, and prescription details, and marks the appointment as completed. If enabled, it sends an email notification to the patient and logs the outcome using Flask flash messages.



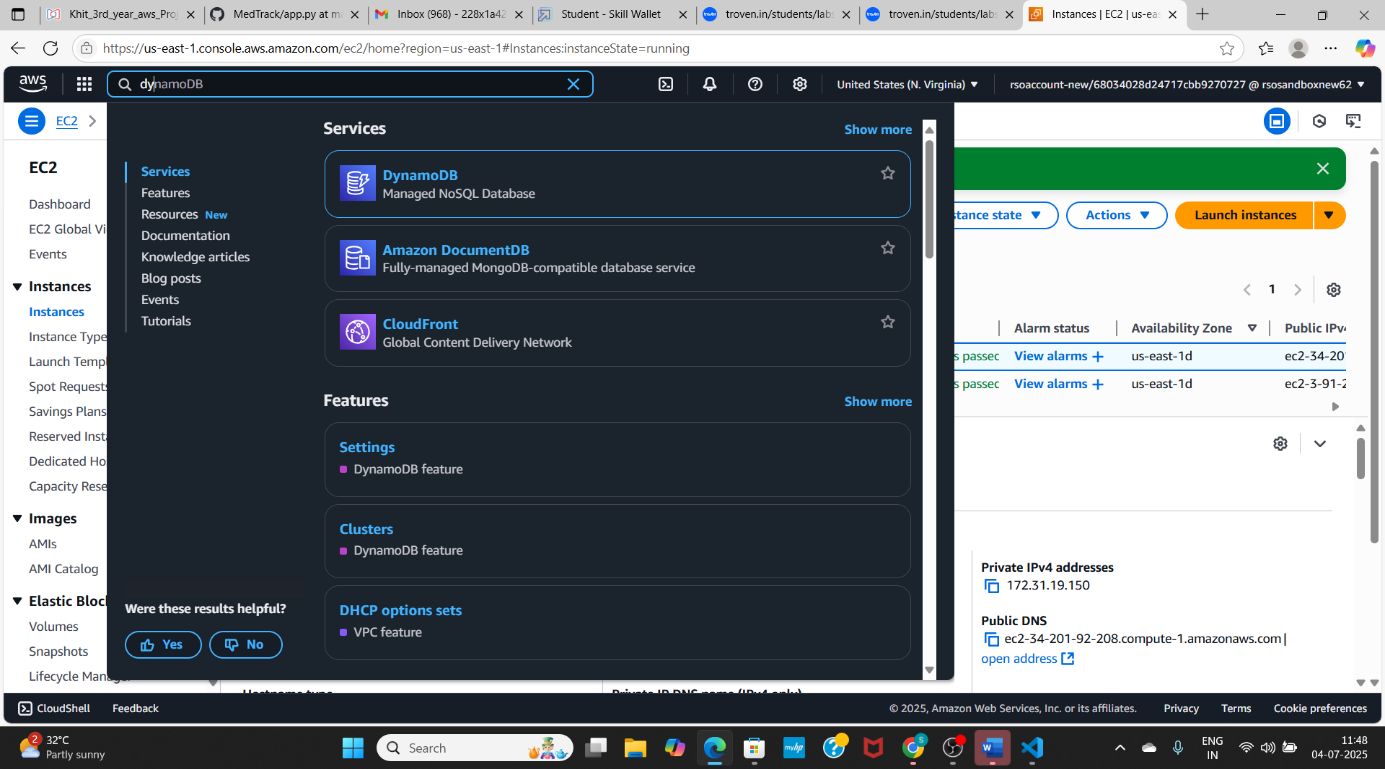
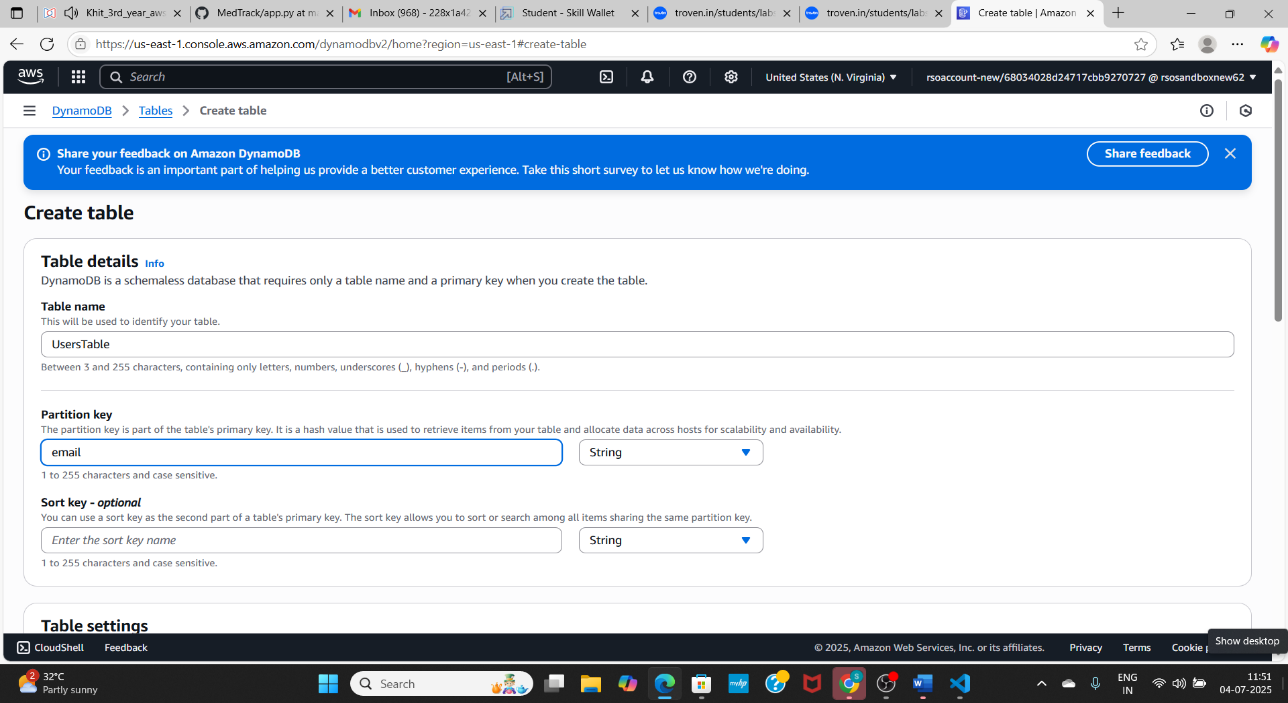
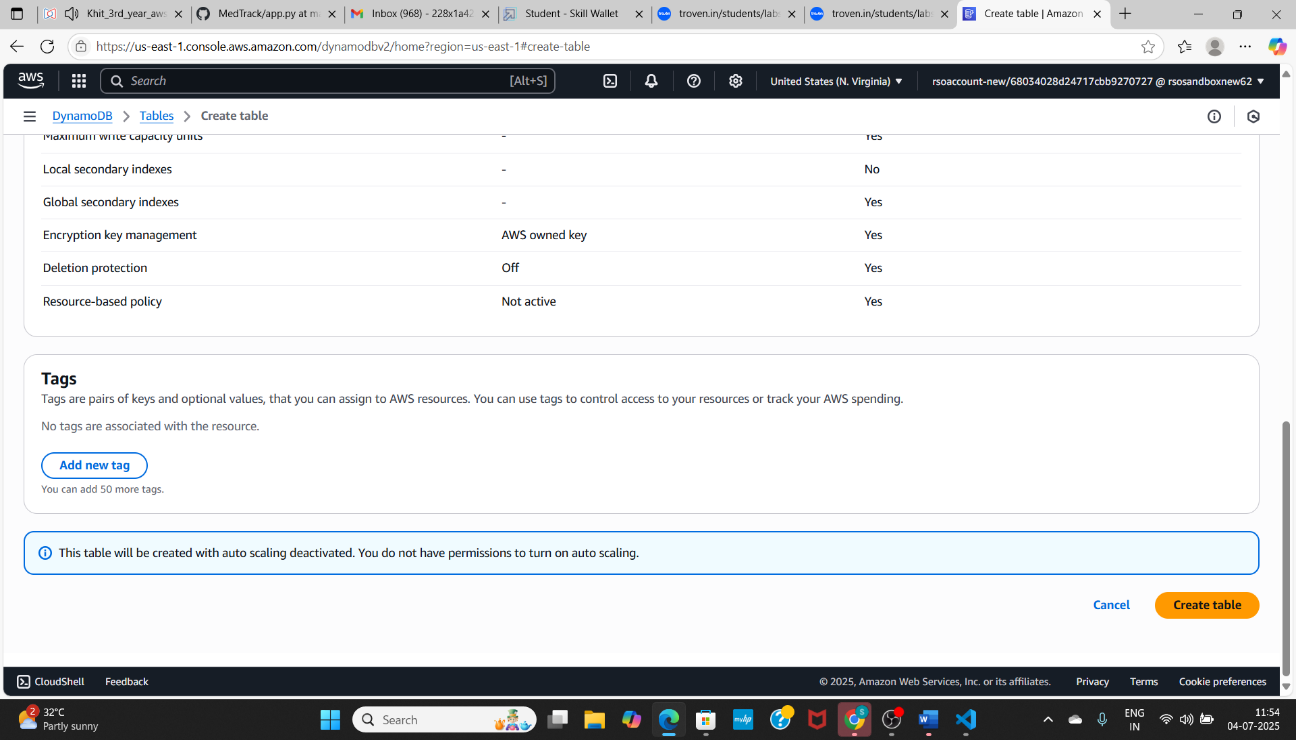
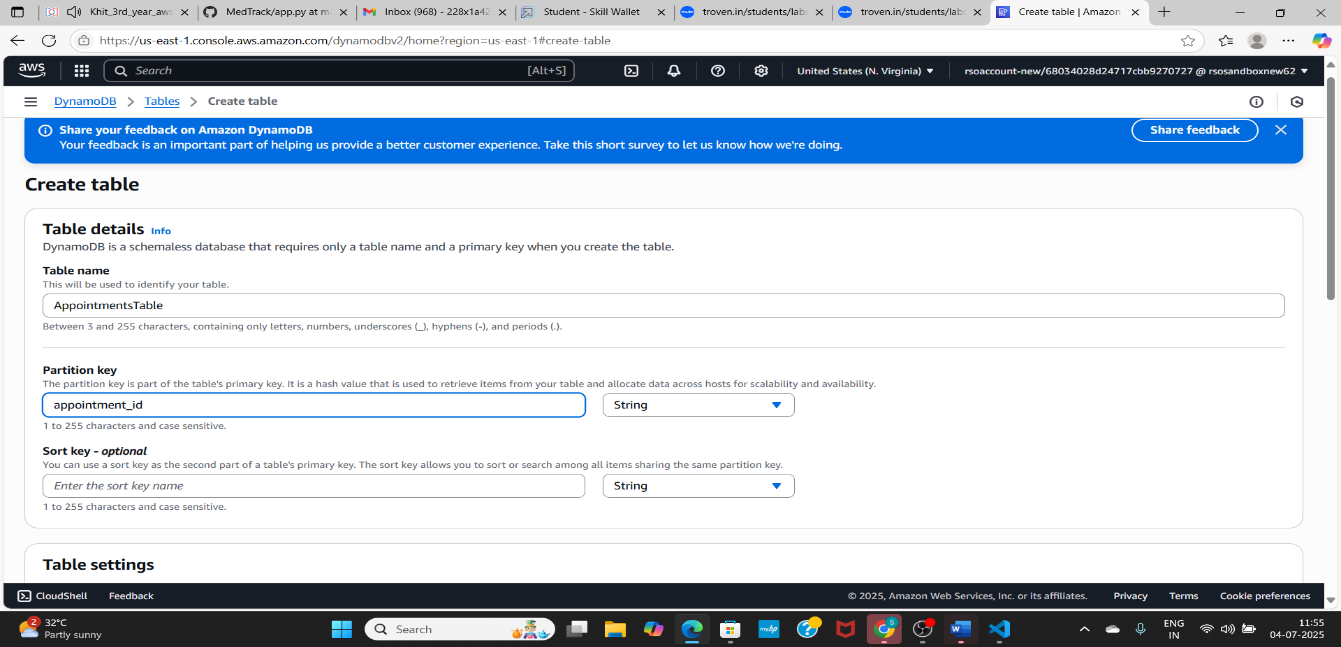
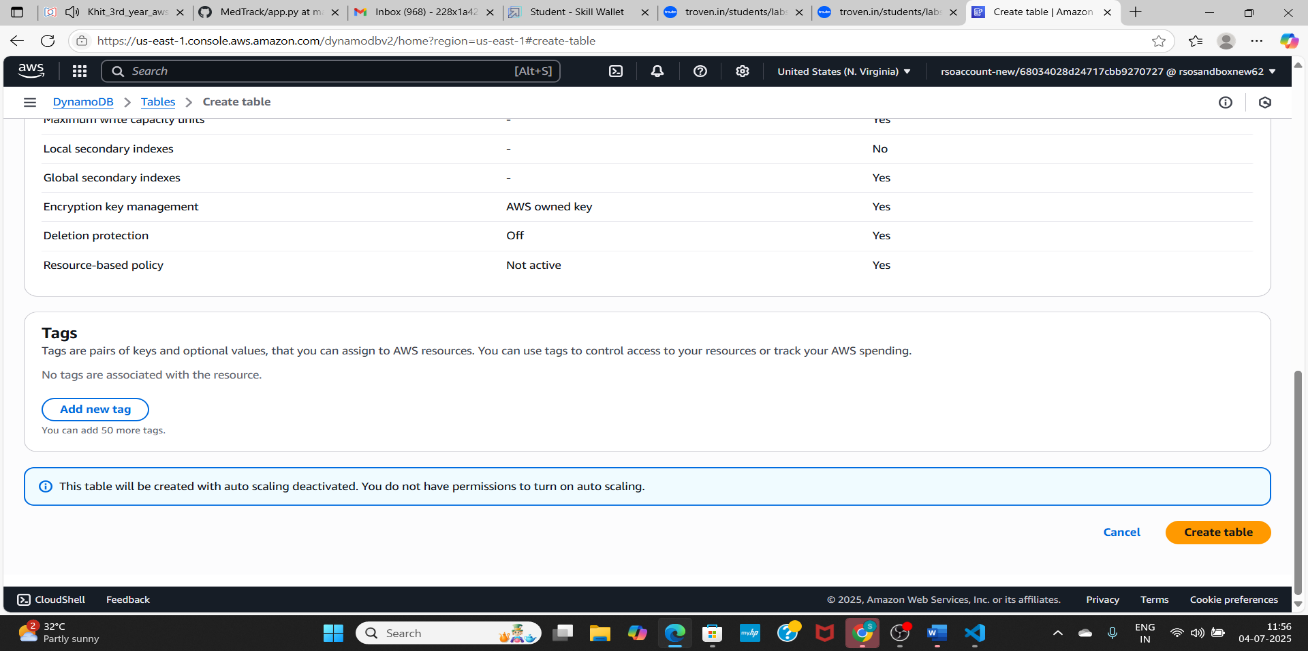
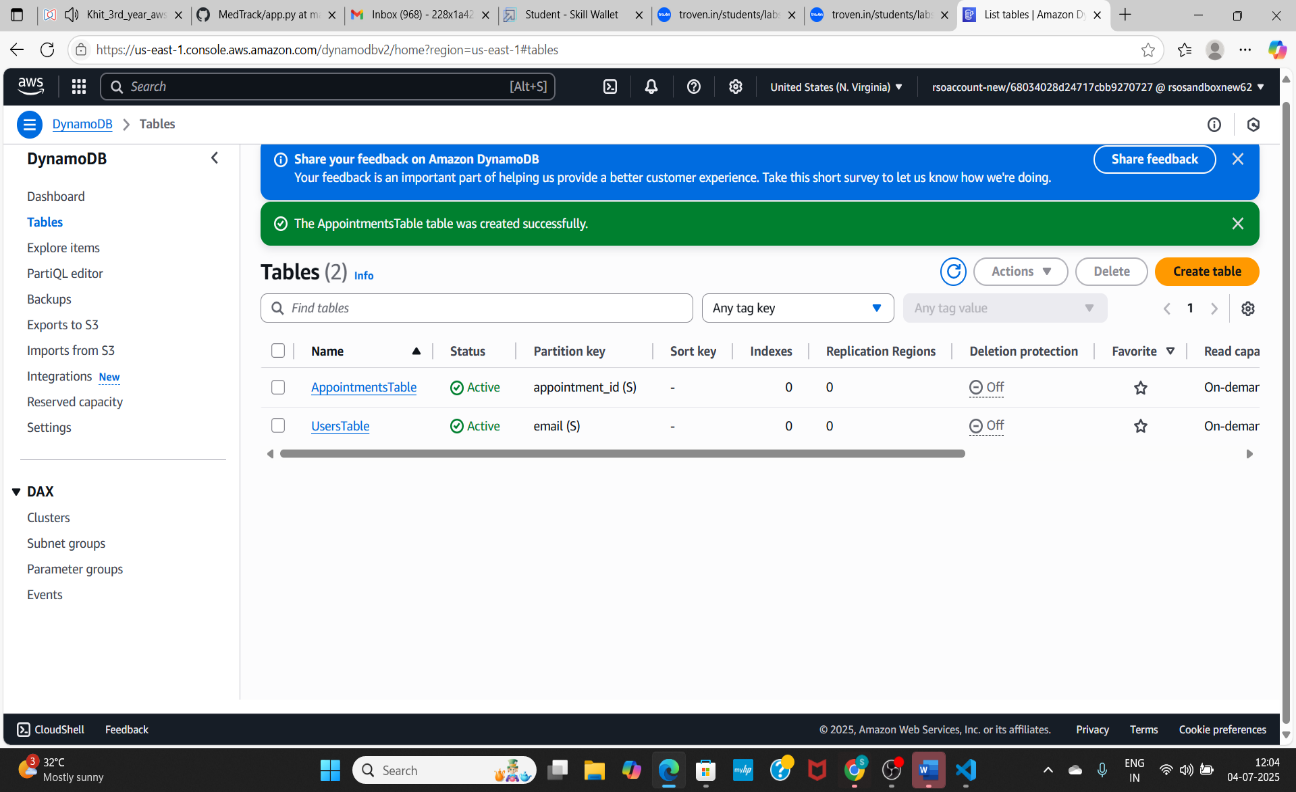
This final block runs the Flask app when the script is executed directly. It sets the port dynamically from an environment variable (PORT) or defaults to 5000, and uses 0.0.0.0 to allow external access—ideal for deployment (e.g. on EC2). Debug mode is enabled, which auto-reloads code changes and provides error tracebacks.

**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.
* 
* 
* 
* 
* 
* 

### Create a DynamoDB table for storing registration details and book requests.

* + - Create Users table with partition key “Email” with type String and click on create tables.
    - Create Appointments table with partition key “appointment\_id” with type String

and click on create tables.

**Milestone 4. SNS Notification Setup**

* Create SNS topics for notifications.

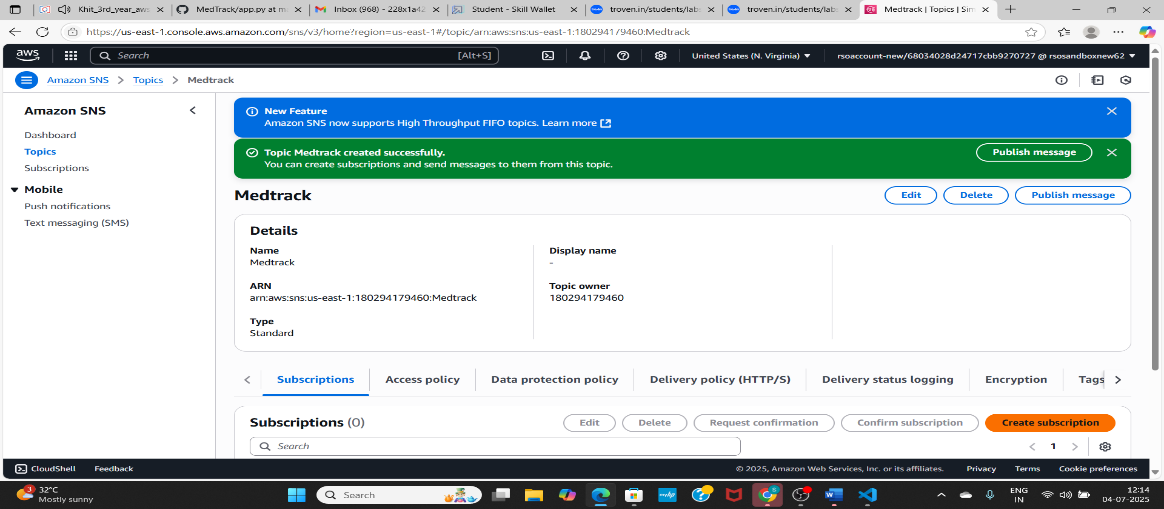
# 

# 

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

# 

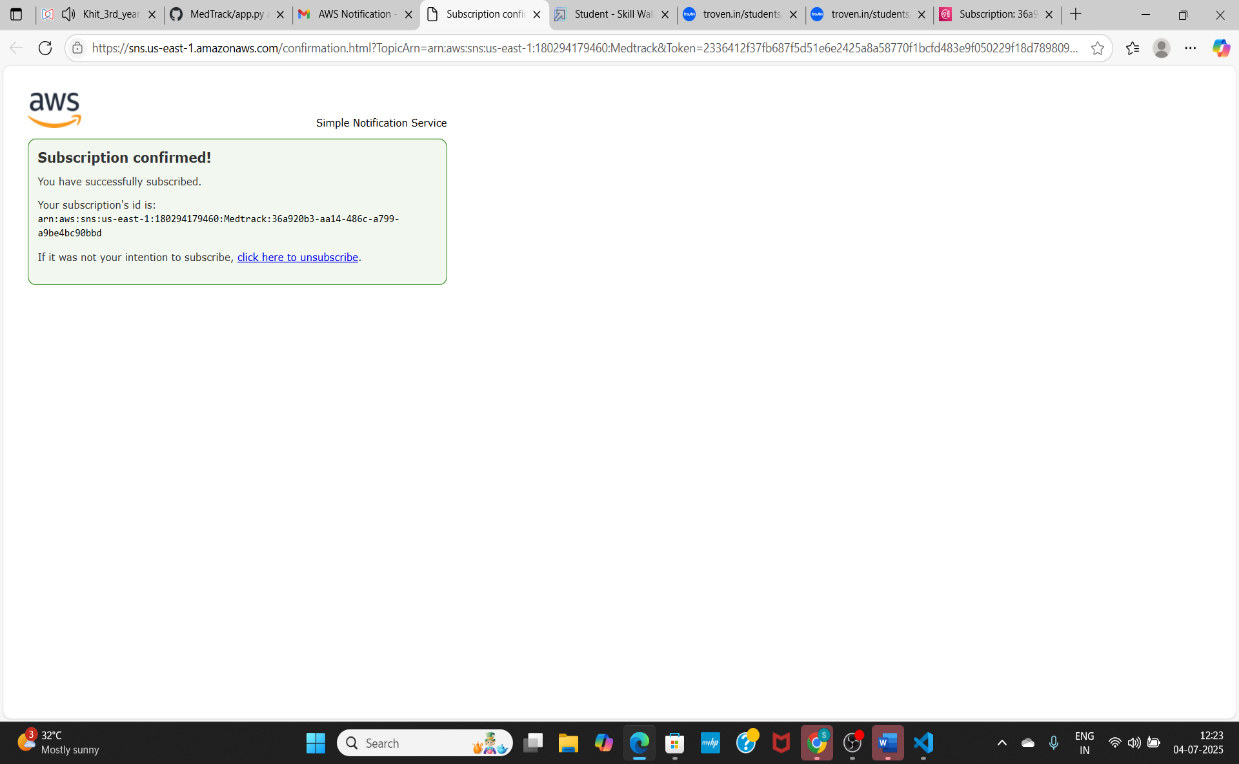
# 

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

# 

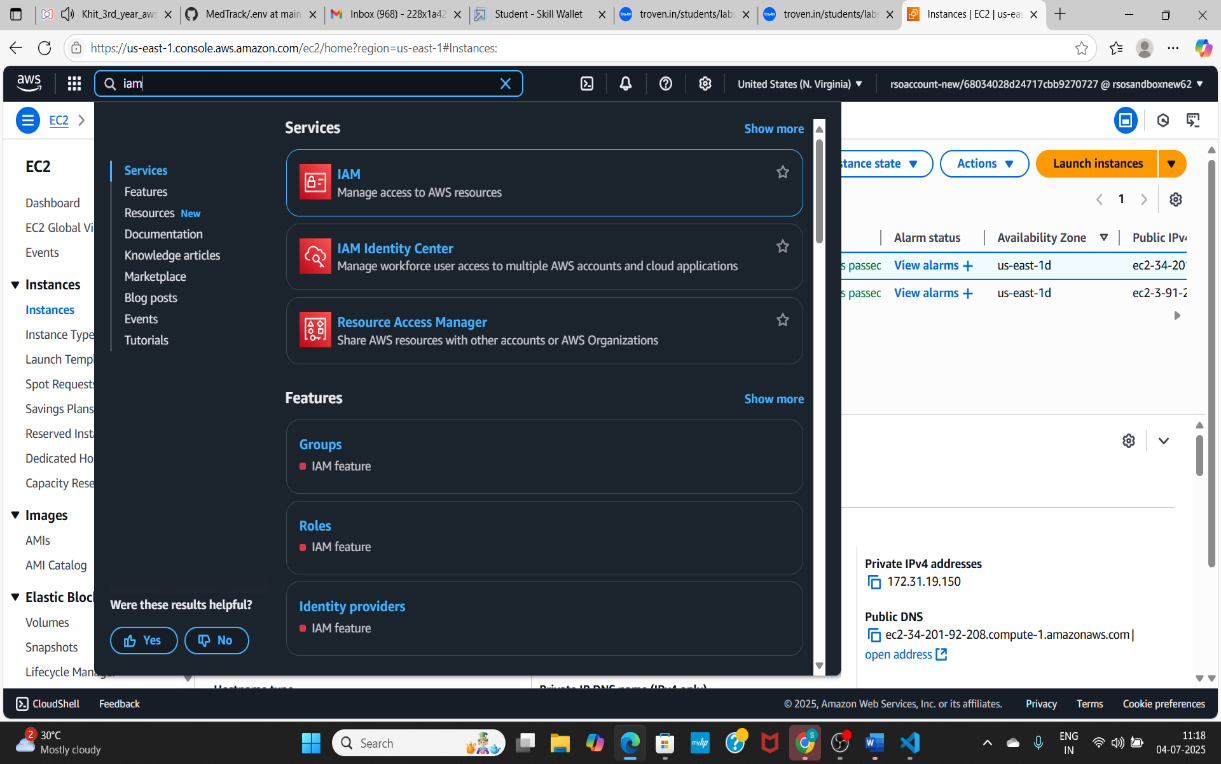
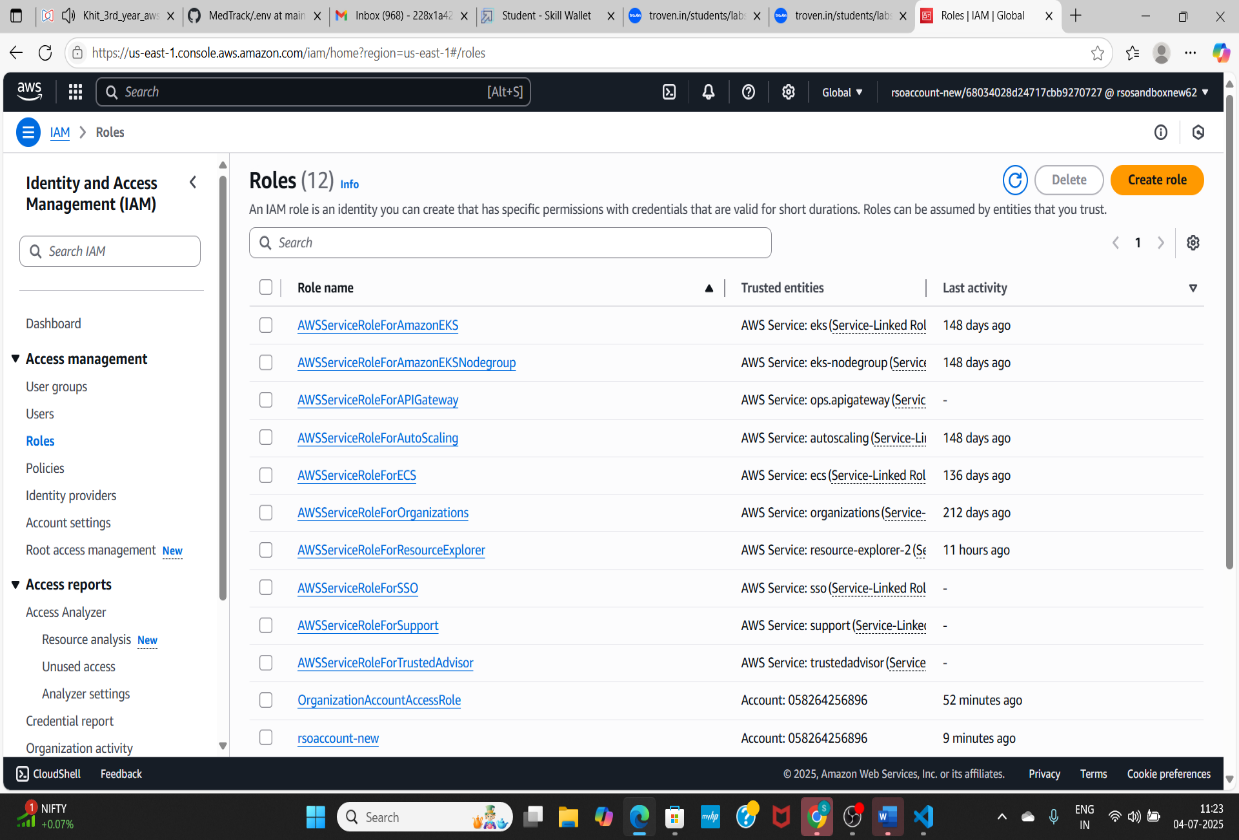
# 

# 

* + Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.
* 
  + Successfully done with the SNS mail subscription and setup, now store the ARN link.

# 

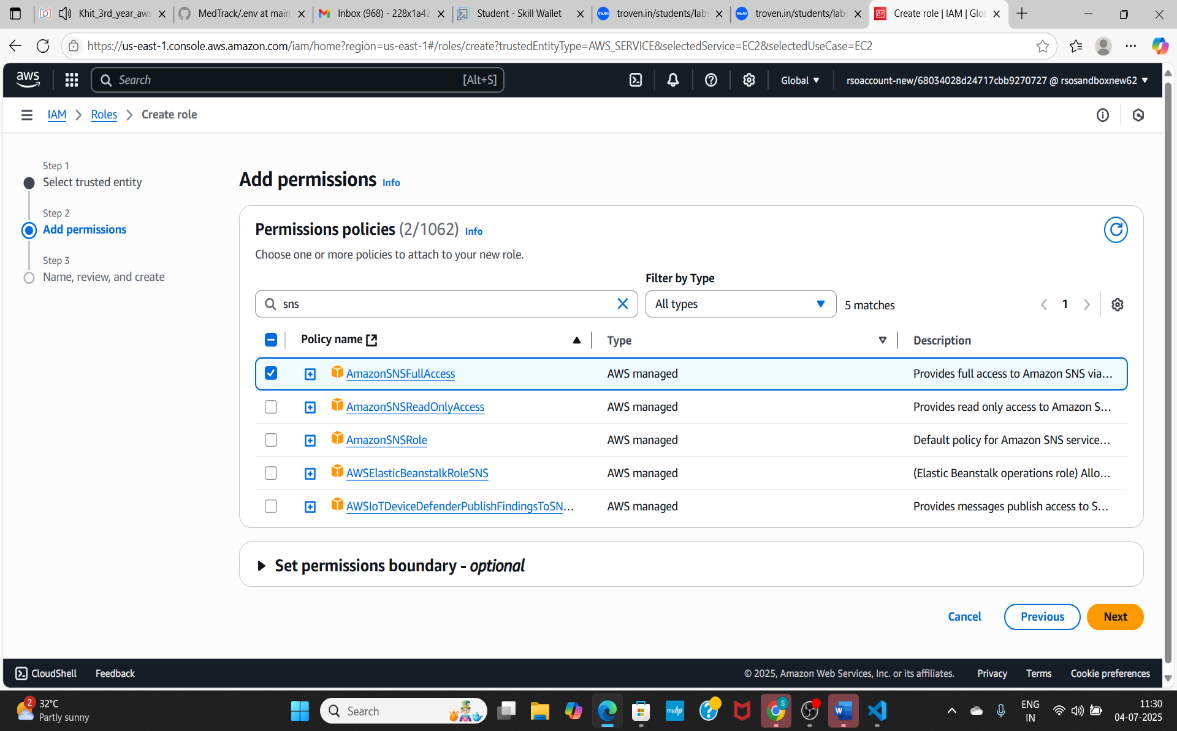
**Milestone 5. IAM Role Setup**

* Create IAM Role
* Attach Policies
* **Create IAM Role**
* 
* 
  + Click on create role

# 

# 

# Now click on next

* 
* 
* **Attach Policies**

# Add permission policies, give full access to AmazonDynamoDB and AmazonSNS.

# 

# 

# Now click on create role.

**Milestone 6. EC2 Instance Setup**

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

# 

# 

# 

# 

# 

# 

# 

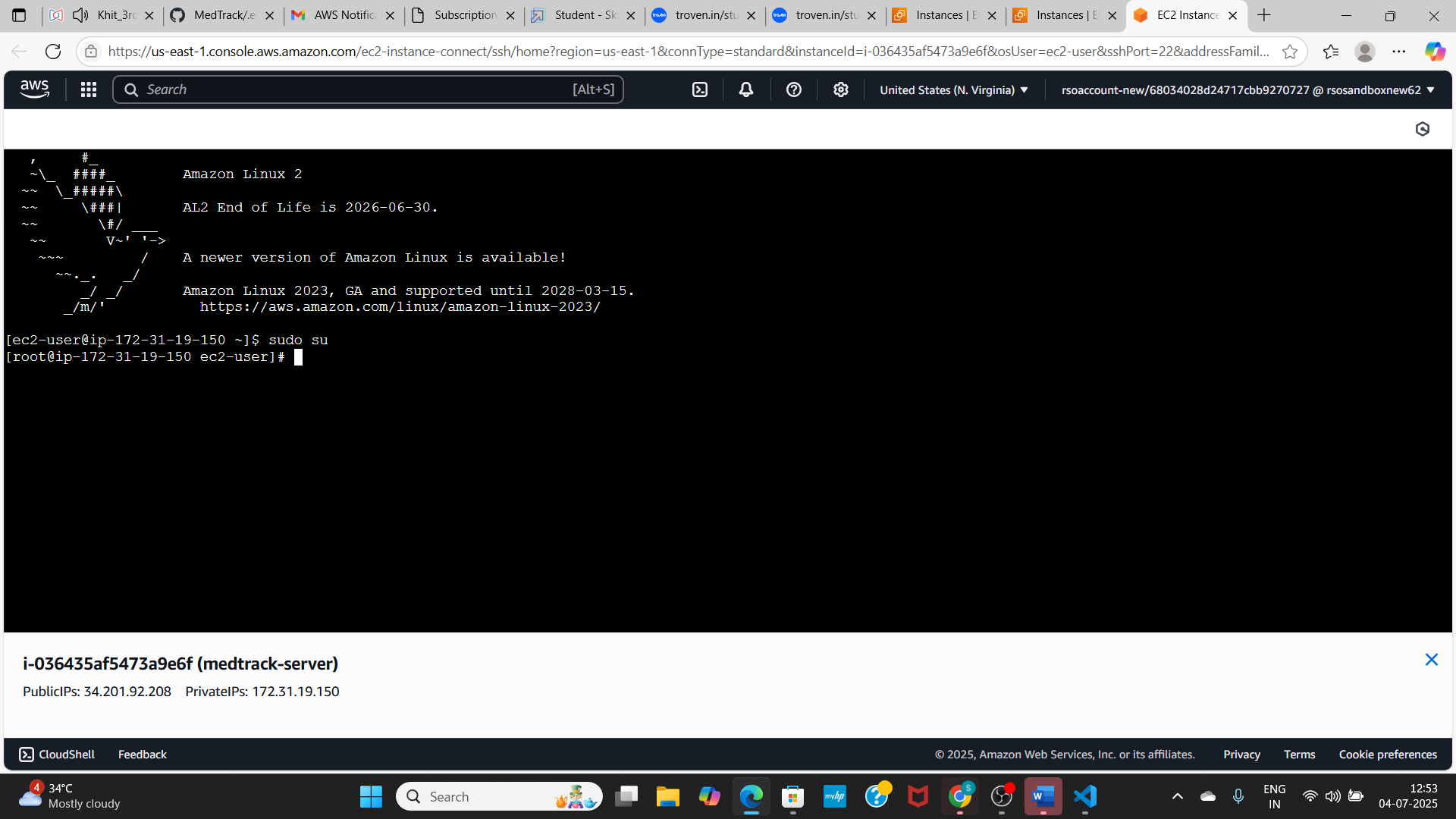
# 

* 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.

**Milestone 7. Deployment using EC2**

* Upload Flask Files

# Run the Flask App

* 

# 

### 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/peddisettymahesh07/MedTrack’

here: ‘git clone https://github.com/peddisettymahesh07/MedTrack.git’

* This will download your project to the EC2 instance.

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

cd MedTrack

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

**Run the Flask Application**

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

# 

**Verify the Flask app is running**: [http://your-ec2-public-ip](http://your-ec2-public-ip/)

* Run the Flask app on the EC2 instance
* Access the website through:
* **PublicIPs:** <https://34.201.92.208/>

Completed hosting on: <http://34.201.92.208:5000/>

**Milestone 8. Testing and Deployment**

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

# 

# 

# 

# 

# 

# 

# 

# 

**Conclusion**

The Med Track project successfully demonstrates the integration of cloud technologies with modern web development to deliver an efficient, secure, and user-friendly healthcare management system. By leveraging AWS services such as EC2, DynamoDB, IAM, and SNS, along with a Flask-based backend, the application streamlines critical healthcare processes including appointment booking, diagnosis management, and medical history access.

The system ensures scalability through cloud deployment, maintains strong data security through role-based IAM access, and enhances user experience with real-time email notifications and intuitive UI routing. Both doctors and patients benefit from the seamless interactions and accessibility provided by the platform.

Overall, Med Track not only meets its intended goals but also lays a solid foundation for future enhancements, offering a practical solution for digital transformation in healthcare communication and coordination.