TECHNICAL HUB


# Doctor Appointment System with Real-Time Queue

**Efficient, Smart, and Scalable Clinic Appointment Management**


**TEAM MEMBERS:-**

| | |
|---|---|
| 1. 23P31A1271 | BENHUR |
| 2. 23P31A05P3 | KARTHIk |
| 3. 23P31A1253 | JAGADEESH |
| 4. 23P31A04I5 | CHINNA |
| 5. 23P31A04I8 | VARSHITA |
| 6. 23A91A04G1 | HASINI |

## Project Overview

The **Doctor Appointment System with Real-Time Queue** is a modern, cloud-native solution developed to optimize and digitalize the appointment and queue management system in clinics and healthcare facilities. Traditional methods of managing patient appointments—such as manual registers or simple calendar tools—are often inefficient, leading to long wait times, disorganization, and unsatisfactory patient experiences.

This system addresses these issues by leveraging **serverless technologies offered by Amazon Web Services (AWS)**, such as **Cognito, API Gateway, Lambda, DynamoDB, SQS, and SNS**. It enables:

- Patients to **book appointments online** based on doctor availability.
- Real-time **tracking and management of patient queues**.
- **Instant notifications** to patients when their turn arrives.
- Secure authentication for both **doctors and patients**.
- A scalable, cost-effective architecture that can serve **multiple clinics or departments** without manual scaling or infrastructure management.

This system is designed with flexibility, security, and scalability in mind—providing healthcare facilities with an intelligent and automated way to manage their appointment lifecycle from start to finish.

# Problem Statement

In many healthcare facilities, appointment booking and patient queue management are still handled manually or through outdated software systems. This results in numerous operational inefficiencies such as:

- **Extended waiting times** for patients
- **Overlapping or missed appointments**
- **Disorganized patient flow** and scheduling conflicts
- **Inadequate communication** between clinic staff and patients

These challenges contribute to poor patient satisfaction, doctor fatigue, and loss of productivity. Most importantly, traditional systems lack the capability to manage queues dynamically or provide real-time updates to patients and staff.

There is a clear need for a **smart, automated system** that can efficiently manage the flow of patients through a clinic while improving the overall quality of care.

## Objectives

This project aims to address the above challenges with the following core objectives:

- **Allow patients to book appointments securely** via a user-friendly interface integrated with a reliable backend.
- **Provide real-time queue updates** so patients can monitor their position and estimated wait time.
- **Notify patients when it's their turn** through SMS or email alerts, reducing idle wait time in the clinic.
- **Enable doctors to manage appointments easily** from a centralized dashboard.
- **Build a scalable, low-maintenance system** using AWS serverless technologies to ensure cost-efficiency, reliability, and ease of deployment.

## Solution Architecture

### Core AWS Services Used

The Doctor Appointment System is built using a **serverless architecture**, which provides high scalability, minimal maintenance, and cost-effective operation. The system is entirely hosted on **Amazon Web Services (AWS)** and uses the following key services:

| Service | Purpose |
|---|---|
| **Amazon Cognito** | Handles **user authentication and authorization** for both do and patients. |
| **API Gateway** | Manages all **RESTful API endpoints**, routing requests to appropriate Lambda functions. |
| **AWS Lambda** | Executes **backend business logic** in response to API request service events. |
| **Amazon DynamoD** | Serves as the **NoSQL database** for storing user profiles, appointment data, and queue records. |
| **Amazon SQS** | Provides a **First-In-First-Out (FIFO) queue system** for manag real-time patient flow. |
| **Amazon EventBrid** | (Optional) Enables **event-driven workflows**, such as autom reminders or session triggers. |
| **Amazon SNS** | Sends **notifications to patients** when their appointment is r (via SMS or email). |

### High-Level Flow Overview

1. **Patients log in** using Cognito and book an appointment.
2. The booking request is handled through **API Gateway**, triggering a **Lambda function**.
3. Appointment details are stored in **DynamoDB**, and the patient is enqueued in **SQS**.
4. When the doctor starts their session and clicks “Next,” the **Lambda function dequeues** the next patient.
5. An **SNS notification** is sent to the patient informing them it's their turn.

# Authentication (Amazon Cognito)

## Overview

User authentication and identity management are critical to ensuring that only authorized users (doctors and patients) can access protected functionalities within the system. This project leverages **Amazon Cognito** to handle secure, scalable authentication without requiring custom authentication logic or infrastructure.

## Key Features

### User Pools

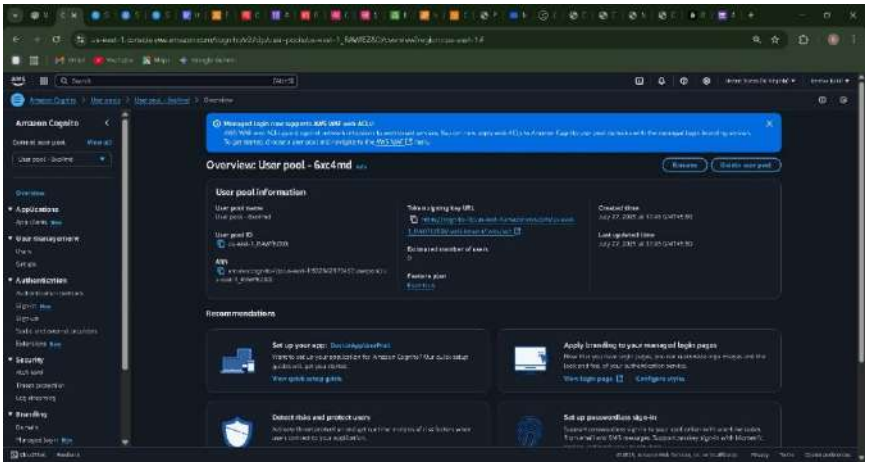
- Amazon Cognito **User Pools** are used to manage user registration and login.
- Separate login credentials are handled for **doctors** and **patients**.
- Each user is uniquely identified by their email or username.

### Groups

- Users are assigned to **Cognito groups** based on their role:
    - **Doctor Group** – can access session controls and view patient queues.
    - **Patient Group** – can book appointments and view their queue status.
- Group-based roles enable **fine-grained access control** to various API endpoints.

### Access Tokens

- Upon login, Cognito generates:
    - **ID Token** (user identity data)
    - **Access Token** (used for API authorization)
    - **Refresh Token** (to maintain session without re-login)
- The **Access Token** is passed with each request to **API Gateway**, which validates it before routing to backend services.



# User Roles & IAM Policies

**Proper access control is essential to ensure system security, data integrity, and the correct functioning of services. This system uses Amazon Cognito for user-level roles (doctors and patients) and AWS IAM for service-level permissions (Lambda, DynamoDB, SQS, etc.).**

## 1. Cognito User Roles

| Role | Description | Access |
|---|---|---|
| Doctor | Authenticated user who can start sessions, view queues, and serve patients | Read/write access to their queue, view appointments |
| Patient | Authenticated user who can book appointments and receive notifications | Limited to their own data, queue status, and notifications |

**Cognito Groups (Example):**

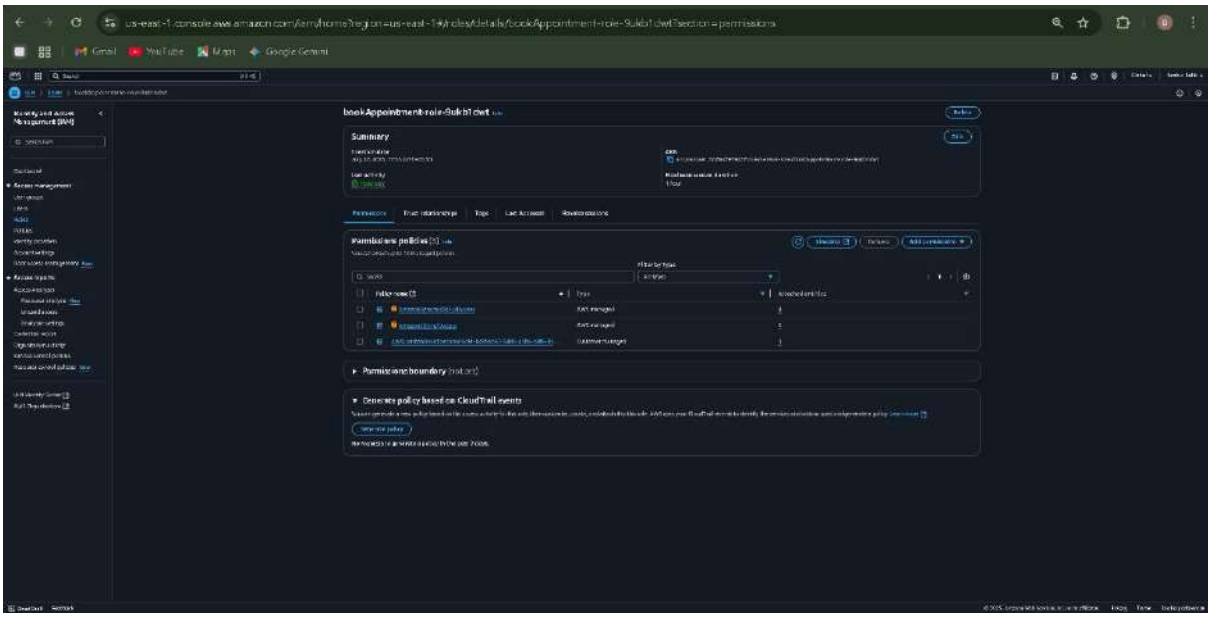
- **DoctorGroup**
- **PatientGroup**

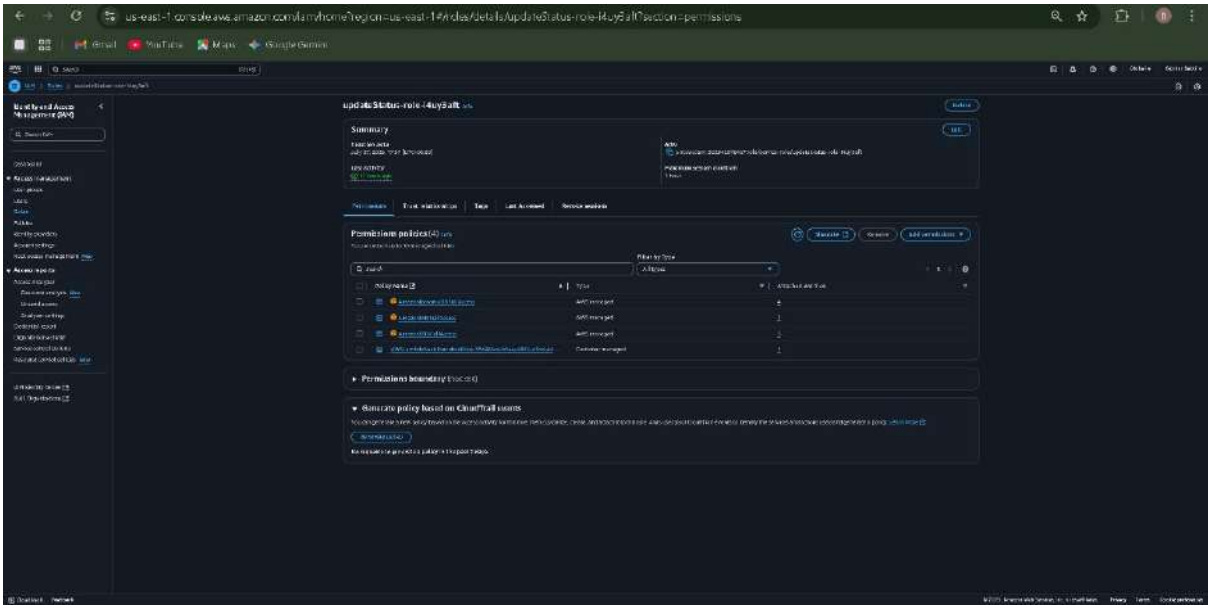
**Role-based access to specific APIs is enforced using Amazon Cognito User Pool Group Claims + API Gateway Authorizers.**

## 2. IAM Roles for AWS Services

| IAM Role | Attached To | Permissions (Summary) |
|---|---|---|
| LambdaExecutionRole | Lambda Functions | Read/write to DynamoDB, publish to SNS, access SQS queues |
| SQSServiceRole | Amazon SQS | Allows sending/receiving messages and triggering Lambda |
| SNSPublishRole | Amazon SNS | Allows Lambda to publish notifications to SNS topics |

| IAM Role | Attached To | Permissions (Summary) |
| --- | --- | --- |
| EventBridgeInvokeLambdaRole | Amazon EventBridge | Permission to trigger Lambda functions based on schedule or events |
| APIGatewayInvokeRole | API Gateway | Invokes Lambda functions via API Gateway |





# API + Lambda Architecture

## Overview

The backend logic of the Doctor Appointment System is built using **AWS Lambda**, which runs stateless, serverless functions in response to HTTP requests managed by **Amazon API Gateway**. This combination allows for fully scalable, event-driven workflows without managing traditional servers.

## Key API Endpoints

The following RESTful API endpoints are exposed via **API Gateway**, each invoking a corresponding **Lambda function**:

| Endpoint | Purpose |
|---|---|
| POST /book-appointment | Books a new appointment and enqueues the patier |
| GET /appointments/{id} | Fetches all appointments for a given user (doctor o patient) |
| POST /doctor/start-sessior | Starts the session and initializes the queue for a do |
| POST /queue/next | Moves the queue to the next patient and triggers notification |
| GET /queue/status | Retrieves the current queue status for a doctor |

## Lambda Functions

Each endpoint triggers a **dedicated AWS Lambda function** responsible for executing the required logic.

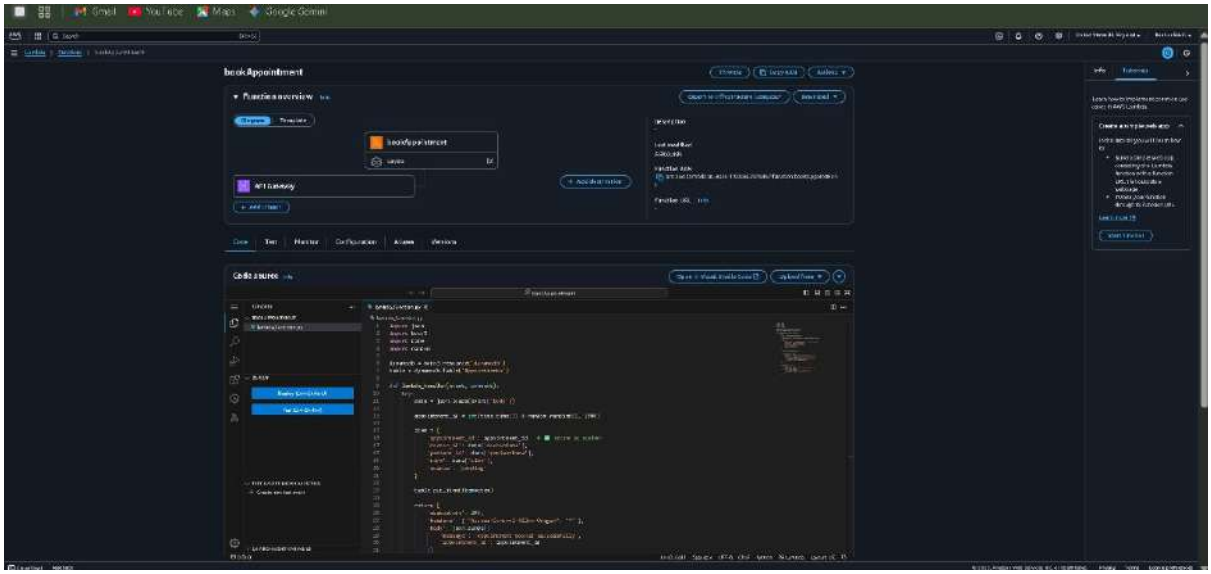
**Key Properties:**

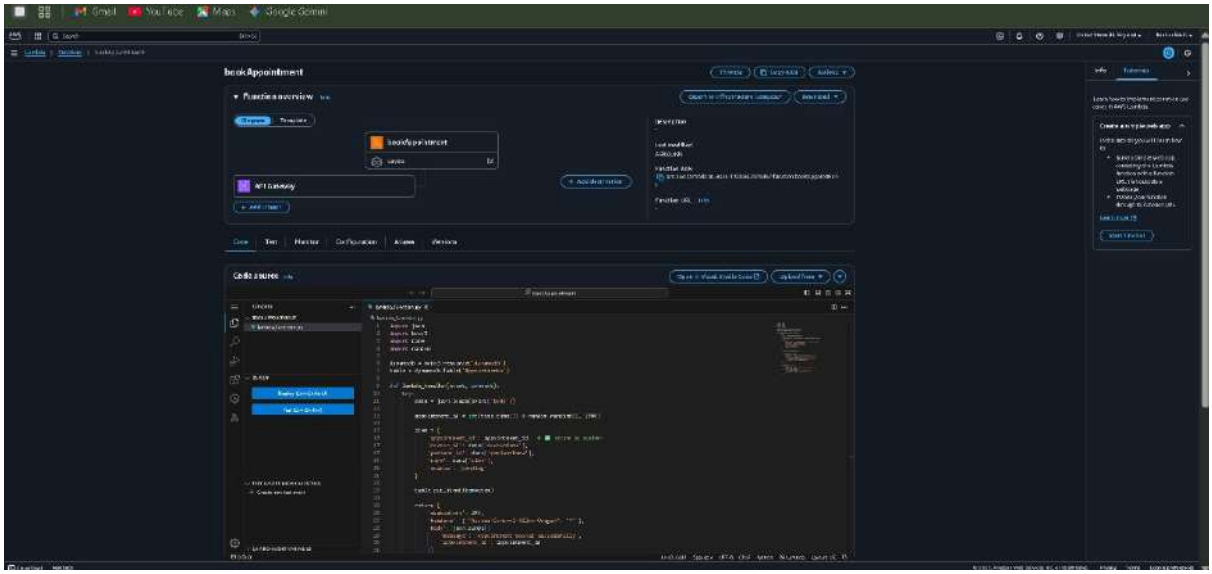
- **Stateless**: Functions do not maintain memory between invocations; all data is retrieved from and saved to **DynamoDB** or **SQS**.
- **Event-Driven**: Functions are triggered by HTTP requests, SQS messages, or EventBridge events.
- **Secure**: Lambda has scoped **IAM roles** that only allow access to specific AWS resources like DynamoDB, SQS, and SNS.

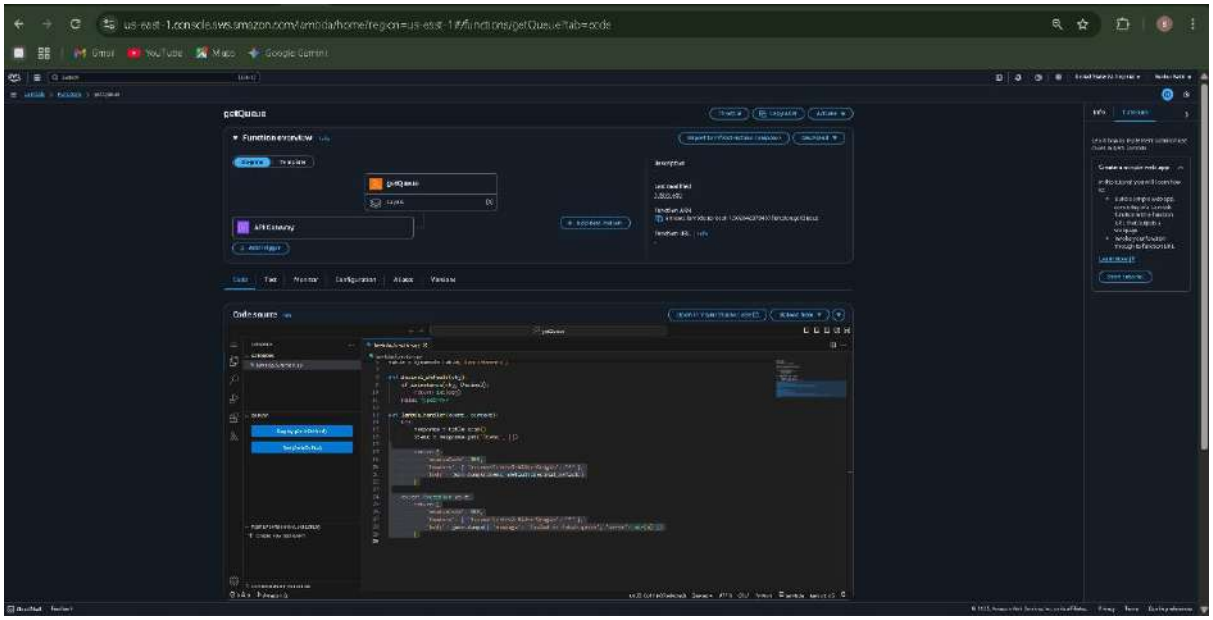
**Sample Functionality:**

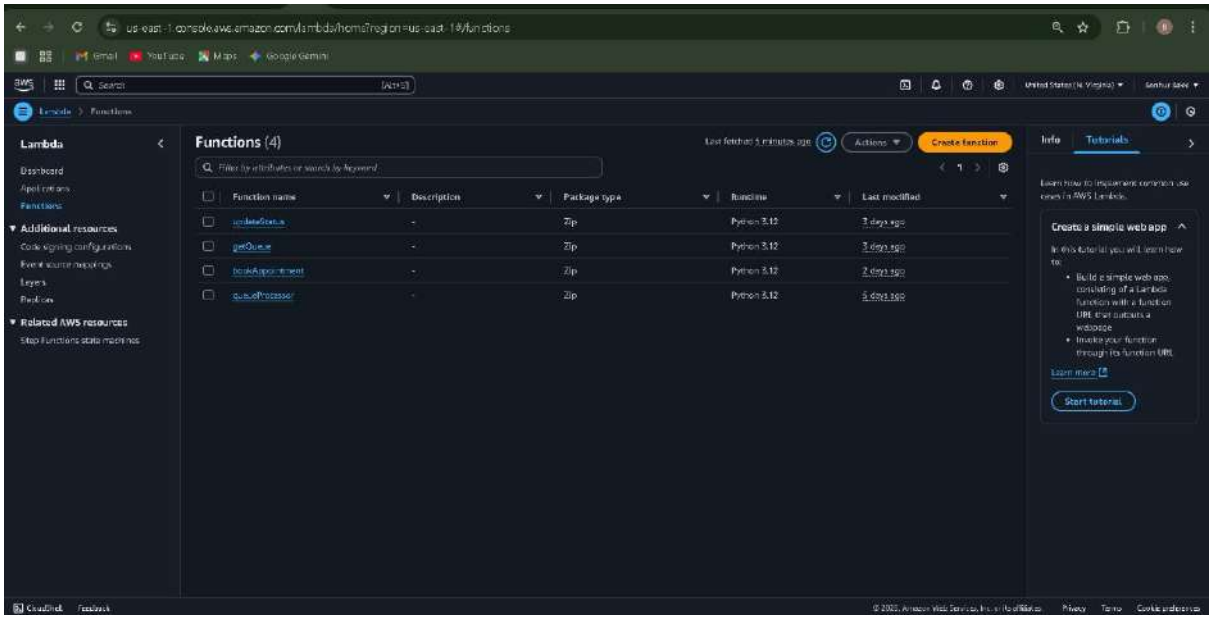
- **bookAppointment**: Validates request, stores booking data in DynamoDB, adds patient to SQS queue.
- **startSession**: Marks a doctor as “active” and initializes their session queue.
- **nextPatient**: Dequeues the next patient, updates appointment status, and sends an SNS notification.
- **getAppointments**: Queries DynamoDB based on doctor or patient ID.

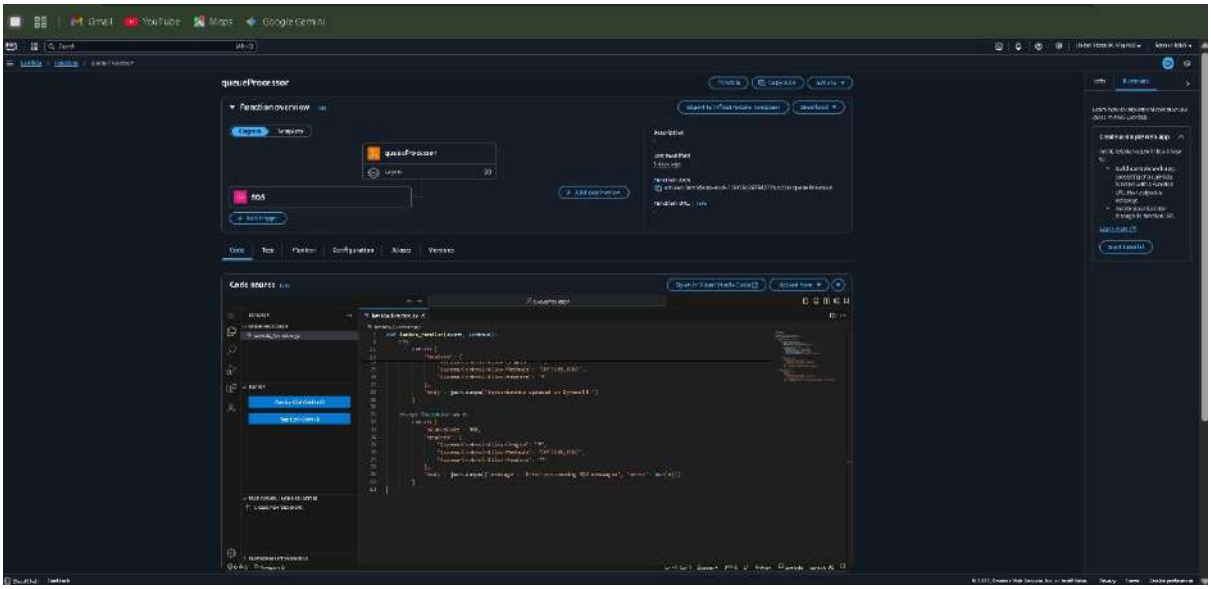
This architecture ensures **modularity**, **fault isolation**, and **cost-effectiveness**, as functions only execute when needed and are billed per invocation.








Functions (4)
Lambda
Dashboard
Applications
Functions
Additional resources
Code signing configurations
Event source mappings
Layers
Replicas
Related AWS resources
Step Functions state machines
Actions
Create function
Info
Tutorials
Function name
Description
Package type
Runtime
Last modified
updateStatus
getQueue
bookAppointment
queueProcessor
Zip
Python 3.12
3 days ago
2 days ago
5 days ago
Create a simple web app
Start tutorial



# Database Design (DynamoDB)

## Tables and Schema

The system uses **Amazon DynamoDB** as the primary data store to manage all appointment and queue-related records. Its NoSQL structure enables flexible, fast, and scalable data handling.

### 1. Doctors Table

- doctorId (Primary Key)
- name
- schedule (Available time slots)

### 2. Patients Table

- patientId (Primary Key)
- name
- contact (Phone or email)

### 3. Appointments Table

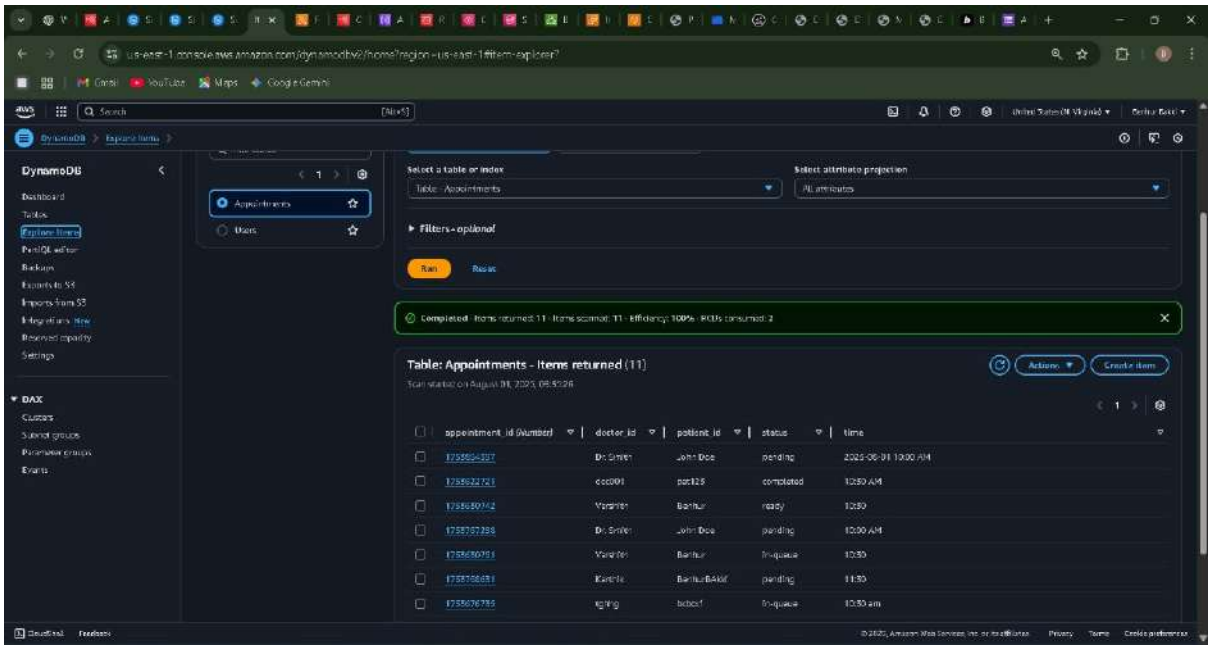
- appointmentId (Primary Key)
- doctorId (Foreign Key)
- patientId (Foreign Key)
- date
- status (e.g., booked, completed, cancelled)

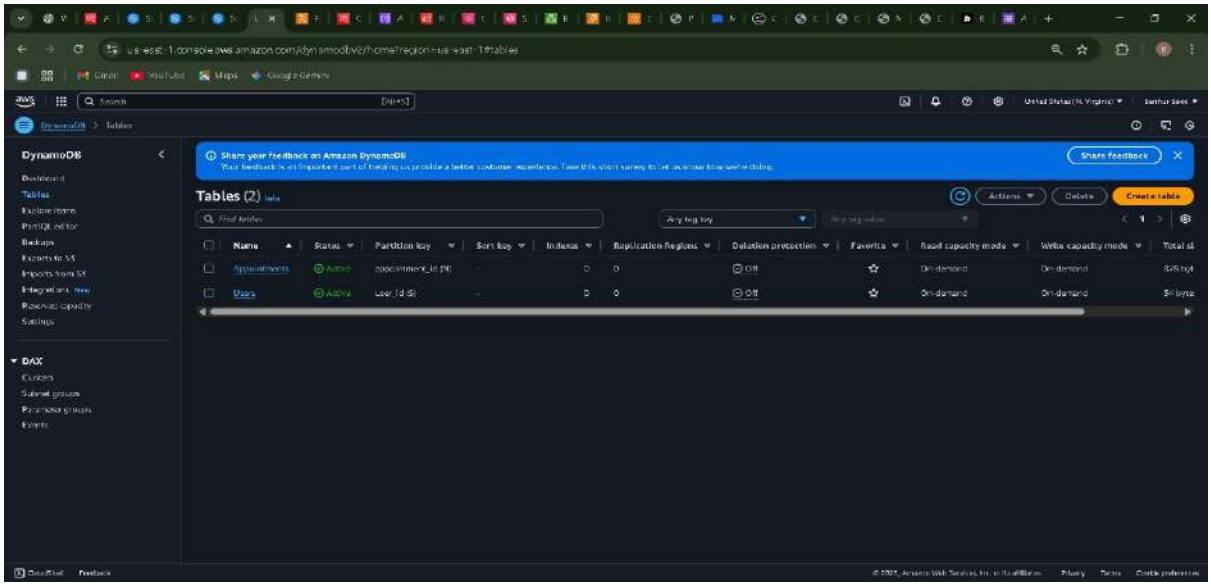
### 4. Queues Table

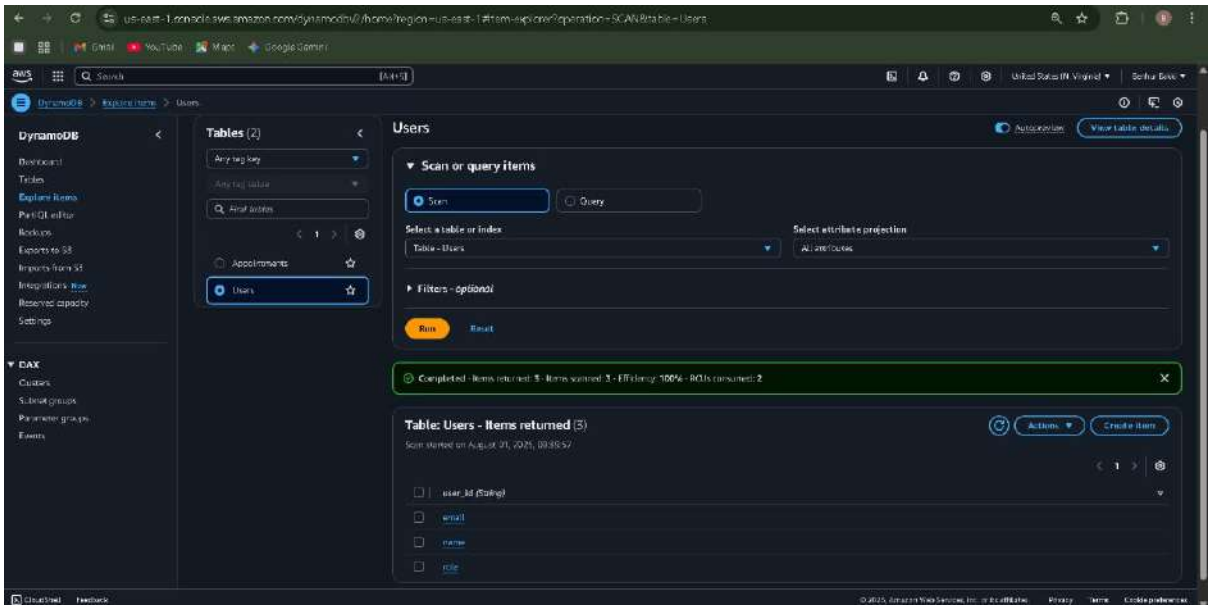
- doctorId (Primary Key)
- patientQueue (List or tokenized queue)

### Benefits

- **Low latency** reads and writes, optimized for real-time interactions.
- **Scalable** NoSQL structure handles thousands of concurrent users.
- **Indexed queries** support efficient filtering by doctor or patient.


DynamoDB
Explore items
Appointments
Users
Select a table or index
Table: Appointments
Select attribute projection
All attributes
Filters - optional
Run
Reset
Table: Appointments - items returned (11)
Actions
Create item
appointment_id (Number)
doctor_id
patient_id
status
time


DynamoDB
Tables
Share your feedback on Amazon DynamoDB
Share feedback
Tables (2)
Actions
Delete
Create table
Name
Status
Partition key
Sort key
Indexes
Replication Regions
Deletion protection
Favorite
Read capacity mode
Write capacity mode
Appointments
Users


DynamoDB
Explore items
Users
Tables (2)
Appointments
Users
Scan or query items
Scan
Query
Select a table or index
Table - Users
Select attribute projection
All attributes
Filters - optional
Run
Reset
Table: Users - items returned (3)
Actions
Create item
user_id (String)
email
name
role

## Amazon SQS (Simple Queue Service )

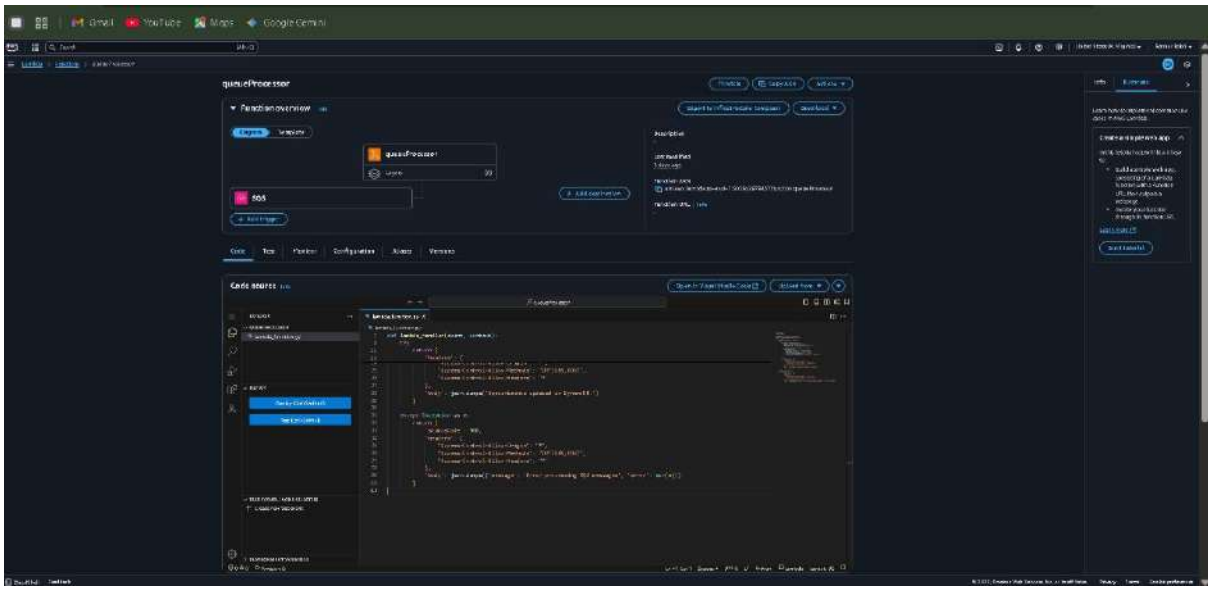
To maintain the **order of service**, each doctor is assigned a **dedicated FIFO (First-In-First-Out) queue**. This ensures that patients are handled **strictly in the order they booked**.

**How It Works:**

- **Queue Creation**: A FIFO queue is created per doctor at the time of registration or session start.
- **Enqueueing**: When a patient books an appointment, their details (or a reference token) are pushed into the corresponding doctor's queue.
- **Dequeueing**: When the doctor clicks the "Next" button in the interface, a Lambda function triggers:
    - Retrieves the next patient from the queue
    - Updates appointment status in DynamoDB
    - Sends a notification via SNS

**Benefits of Using FIFO SQS:**

- Guarantees **exact order of processing**
- Prevents **duplicate entries** through deduplication IDs
- Automatically scales with load
- Integrates seamlessly with Lambda and SNS



# Notification System (SNS)

## Overview

Timely communication with patients is a critical component of the Doctor Appointment System. To deliver **real-time notifications** when it's a patient's turn, the system uses **Amazon Simple Notification Service (SNS)** — a highly scalable and flexible messaging service that supports multiple communication protocols, including **SMS**, **email**, and **mobile push**.

## How It Works

- Each doctor is assigned a **dedicated SNS Topic** during session initialization.
    - Example: doctor-123-notify
- When a patient books an appointment, their contact details (email or phone number) are temporarily **subscribed** to the doctor's SNS Topic.
- When the doctor presses the **"Next"** button to move the queue, a **Lambda function**:

1. Dequeues the next patient (via SQS).
2. Sends a message to the doctor's SNS Topic.
3. The SNS Topic **broadcasts the message** to the subscribed patient.

## Sample Notification

Patients receive a personalized notification such as:

**"Your turn with Dr. Sharma. Please proceed to Room 2."**

This ensures patients are notified **instantly** and can act accordingly, minimizing idle waiting time and improving flow efficiency.

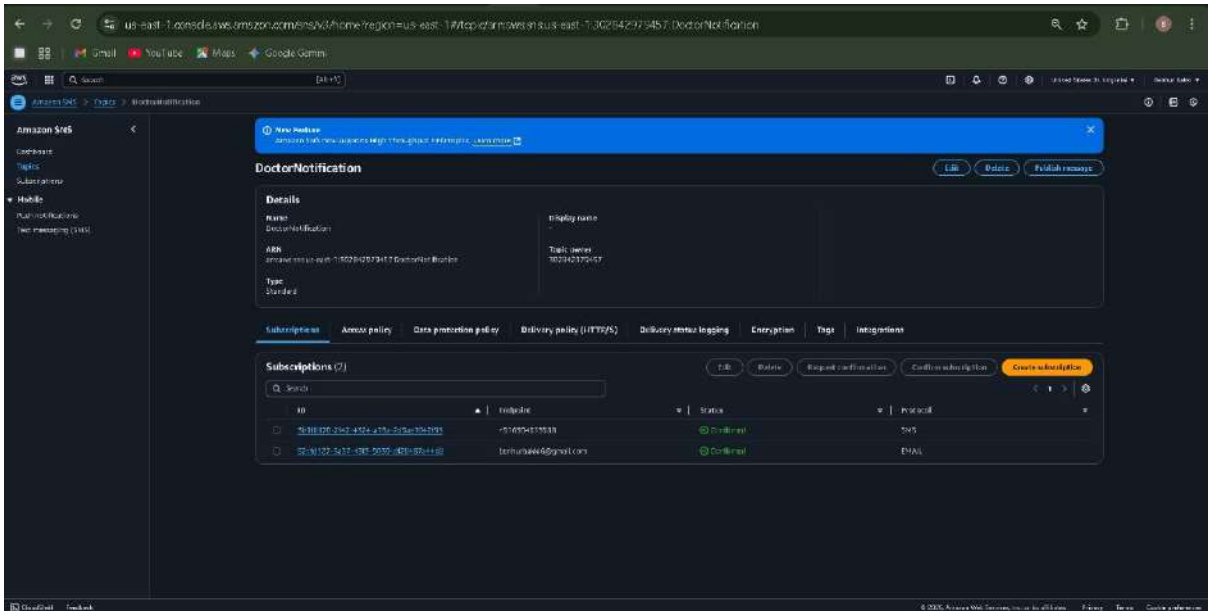
## Supported Delivery Channels

| **Channel** | **Usage** |
|---|---|
| **SMS** | Primary method for quick mobile alerts |
| **Email** | Secondary method or fallback if SMS is unavailable |
| **Mobile Push** *(Optional)* | Can be enabled using AWS Pinpoint for app-based notifications |

## Benefits of Using SNS

| Feature | Description |
|---|---|
| **Near real-time deliver** | Patients are notified almost instantly when it's their tu |
| **Scalable** | Can handle hundreds or thousands of messages per se |
| **Cost-effective** | Pay-per-message model makes it economical for clinics all sizes |
| **Multi-channel suppor** | Can expand to include push or app notifications if neec |
| **Serverless Integration** | Seamlessly integrates with Lambda, SQS, and EventBric |

Amazon SNS provides a **reliable, automated messaging layer** for patient engagement, ensuring efficient flow and improved clinic experience without requiring external messaging platforms.



# Workflow

Understanding the **end-to-end workflow** of the Doctor Appointment System helps clarify how various AWS services interact to provide a seamless, real-time experience for both doctors and patients. The process is divided into two key flows: **Booking Flow** (patient-driven) and **Queue Flow** (doctor-driven).

**Booking Flow**

1. **Patient Logs In**
    - Patient authenticates securely using **Amazon Cognito** credentials.
2. **Selects Doctor & Time**
    - Patient browses available doctors and selects a preferred **time slot** from the schedule.
3. **Appointment Saved in DynamoDB**
    - Appointment details (doctorId, patientId, date, status) are stored in the Appointments table in **Amazon DynamoDB**.
4. **Patient Added to SQS Queue**
    - The patient's ID or appointment token is **enqueued** into the corresponding **FIFO SQS queue** for the selected doctor.
5. **Confirmation Sent**
    - The patient receives a **confirmation message** via **SNS**, either by SMS or email, indicating that the appointment has been successfully booked.

**Queue Flow (Doctor-Side Interaction)**

1. **Doctor Logs In and Starts Session**
    - The doctor logs into the system using **Cognito**, and starts the session, which initializes the patient queue.
2. **Presses "Next" to Dequeue**
    - When ready, the doctor presses the **"Next"** button, which triggers a **Lambda function** to:
        - Dequeue the next patient from the SQS queue.
        - Retrieve the patient's contact information from DynamoDB.
3. **SNS Sends Notification**
    - **Amazon SNS** publishes a personalized notification to the dequeued patient, alerting them that it's their turn to visit the doctor.

## Summary:-

This workflow enables:

- **Real-time updates** for patient position in the queue
- **Seamless doctor-patient communication**
- **Minimal clinic staff intervention**
- **Efficient use of cloud-native AWS services**

Together, these flows ensure the system is **automated, accurate, and scalable**, minimizing wait times and streamlining clinical operations.

# Conclusion & Future Scope

## Conclusion

The **Doctor Appointment System with Real-Time Queue** presents a modern, cloud-native solution to a long-standing challenge in healthcare—efficient and reliable appointment and queue management. By utilizing a fully serverless architecture on **Amazon Web Services (AWS)**, this system achieves:

- **Automated booking and queuing** without the need for manual staff intervention
- **Real-time communication** with patients via SNS notifications
- **High scalability** to accommodate clinics of various sizes
- **Secure authentication** through Amazon Cognito for both doctors and patients
- **Operational efficiency**, reducing delays, overlaps, and patient dissatisfaction

This system is not only technically efficient but also **cost-effective** and **easily deployable**, making it a strong candidate for adoption across a wide range of medical institutions.

## Future Scope

The system architecture has been designed with extensibility in mind. The following enhancements can be implemented to further improve functionality and user experience:

### Real-Time Dashboard

- Live updates of queue positions for doctors and patients
- Enhanced transparency and communication

### Admin Control Panel

- Administrative access to manage doctors, appointments, and users
- Role-based access and detailed analytics

### Predictive Wait Times Using Machine Learning

- Forecast queue durations based on appointment history and doctor availability
- Improve patient planning and reduce idle time