

# CSCI 5411

## Advanced Cloud Architecting

### TERM ASSIGNMENT

Banner ID: B00960433 | Shrey Patel

GITHUB\_REPO:

<https://github.com/shrey-3155/Healthcare-Appointment-Schedule-on-AWS/tree/main>

## Table of Contents

<b>1 About my application:</b>	4
<b>1.1 Public Repository link:</b>	4
<b>1.2 Functionality:</b>	4
<b>2 Hosting of the application on AWS:</b>	5
<b>2.1 Elastic Ip link of Ec2 frontend:</b>	5
<b>2.2 Snippets of successful application hosting on AWS:</b>	5
<b>3 Justification of the chosen AWS services:</b>	11
<b>3.1 Services used:</b>	11
<b>3.2 Justification:</b>	11
<b>4 Implementation of the AWS Well-Architected Framework principles and best practices:</b>	13
<b>4.1 Architecture Diagram:</b>	13
<b>4.2 Explanation from 6 well architected framework pillars:</b>	13
<b>4.3 Why each service suits my application?</b>	16
<b>5 Infrastructure of Code:</b>	19
<b>5.1 Snippet of Successful resources creation:</b>	19
<b>5.2 Cloud Formation file explanation:</b>	21
<b>6 References:</b>	24

## Table of Figures

Figure 1 Application URL to be accessed after Cloud Formation .....	5
Figure 2 Landing Page .....	5
Figure 3 Patient Signup.....	6
Figure 4 Patient Login .....	6
Figure 5 Patient Profile with image from S3 .....	7
Figure 6 All available doctors .....	7
Figure 7 Selection of Date .....	7
Figure 8 Selection of Time Slot .....	8
Figure 9 Receipt.....	8
Figure 10 Stripe Payment.....	9
Figure 11 Previous Appointment .....	9
Figure 12 Patient feedback.....	10
Figure 13 Doctor Login .....	10
Figure 14 Doctor Profile .....	10
Figure 15 Previous Appointments of doctor .....	11
Figure 16 Architecture Diagram .....	13
Figure 17 Successful Stack Creation .....	19
Figure 18 Resources creation snippet 1 .....	19
Figure 19 Resources creation snippet 2 .....	20
Figure 20 Resources creation snippet 3 .....	20
Figure 21 Resources creation snippet 4 .....	21
Figure 22 Resources creation snippet 5 .....	21

## 1 About my application:

I have chosen a repository of **Healthcare Appointment scheduling** application in MERN stack with Stripe integration which allows users to have two roles. They could be either registered doctor or patient. Patients can see all the specialist doctors of different domains with their fees, availability and name. They can select to meet with them and do the payment. They will get a google meet link in which they can join and take session from doctor. They will also receive a email notification about their appointment. Similarly for the doctors they can see which patient has their appointment booked and they will also receive google meet and email notification. The backend is done in Node Js with MongoDB as database while the frontend is a react application.

### 1.1 Public Repository link:

<https://github.com/Project-Based-Learning-IT/healthcare-appointment-scheduling-app/tree/calendar>

### 1.2 Functionality:

The Patient-Doctor Appointment Booking Web Application offers the following functionalities:

1. **User Authentication:**
  - Patients can sign in seamlessly using their Google accounts.
  - Doctors are required to register in the system before logging in to ensure secure access and validation.
2. **Appointment Booking:**
  - Patients can view available slots and book appointments directly through the application.
  - Upon booking, a calendar event is automatically created, complete with a meeting link for virtual consultations.
3. **Feedback System:**
  - Patients can provide feedback about their consultation experience, helping to improve the quality of service.
4. **JWT Authentication:**
  - Secure communication between the client and server is ensured using JSON Web Tokens (JWT), providing a robust authentication mechanism.
5. **Payment Integration:**
  - The application includes a built-in payment feature, allowing patients to pay consultation fees directly through the platform.
6. **Tech Stack:**
  - **Frontend:** Built using **React.js** for a dynamic and user-friendly interface.
  - **Backend:** Powered by **Node.js 16** and **Express.js**, ensuring efficient server-side functionality with MongoDB database for document storage.

These features collectively create a seamless and efficient experience for both patients and doctors, streamlining appointment management and virtual consultations.

## 2 Hosting of the application on AWS:

### 2.1 Elastic Ip link of Ec2 frontend:

<http://34.192.38.100>

**Note:** For this Ip to work, my lab session needs to be started, because my frontend is on Ec2 instance which stops when I shut down my lab session. But I have provided the screenshots of working application in 2.2.

### 2.2 Snippets of successful application hosting on AWS:

Outputs (2)				
<input type="text" value="Search outputs"/>				
Key	Value	Description	Export name	
ApplicationURL	<a href="http://ec2-54-221-190-152.compute-1.amazonaws.com:3000">http://ec2-54-221-190-152.compute-1.amazonaws.com:3000</a>	The URL of the deployed application	-	
NATGatewayElasticIP	98.85.189.192	Public Elastic IP address associated with the NAT Gateway	NATGatewayElasticIP	

Figure 1 Application URL to be accessed after Cloud Formation

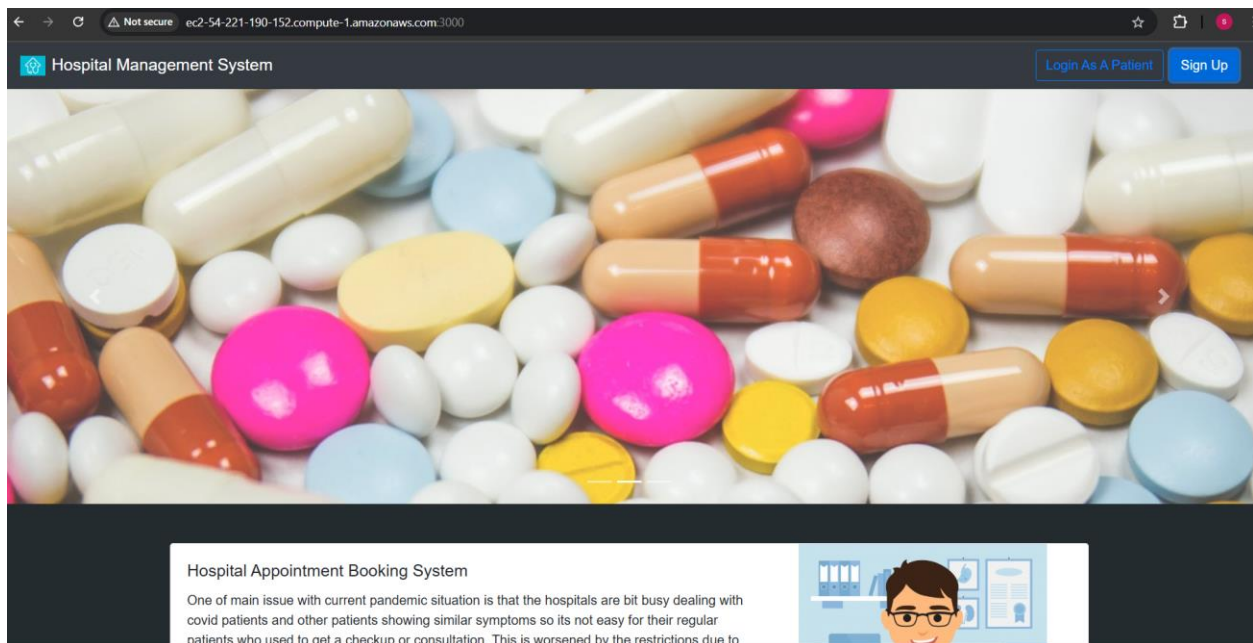
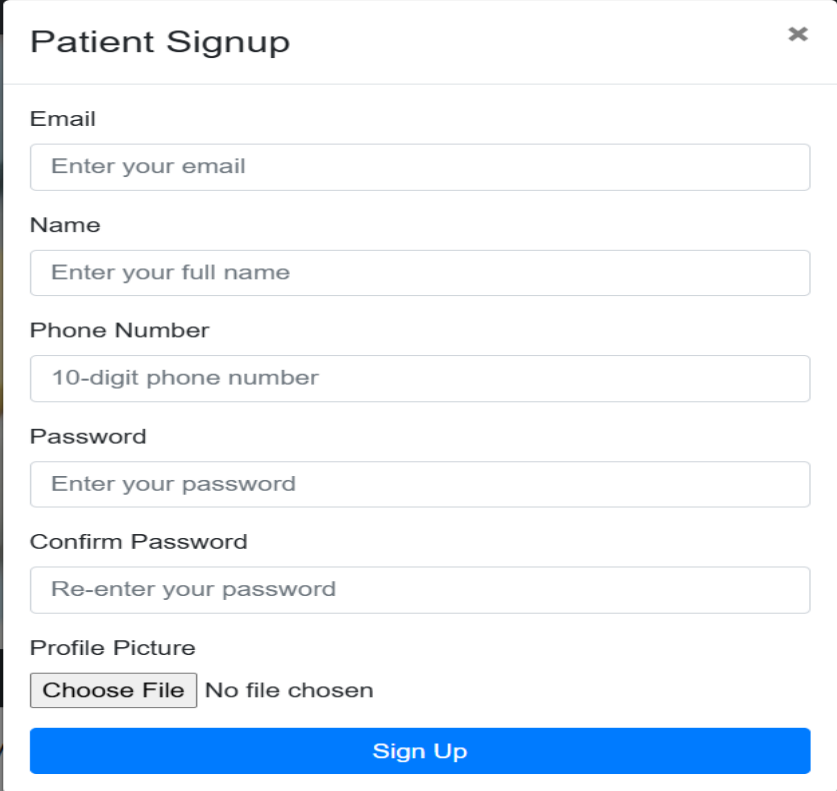


Figure 2 Landing Page



The image shows a 'Patient Signup' modal form overlaid on a background of various colored pills. The form includes fields for Email, Name, Phone Number, Password, and Confirm Password, along with a Profile Picture section and a 'Sign Up' button.

**Patient Signup** [X]

Email

Name

Phone Number

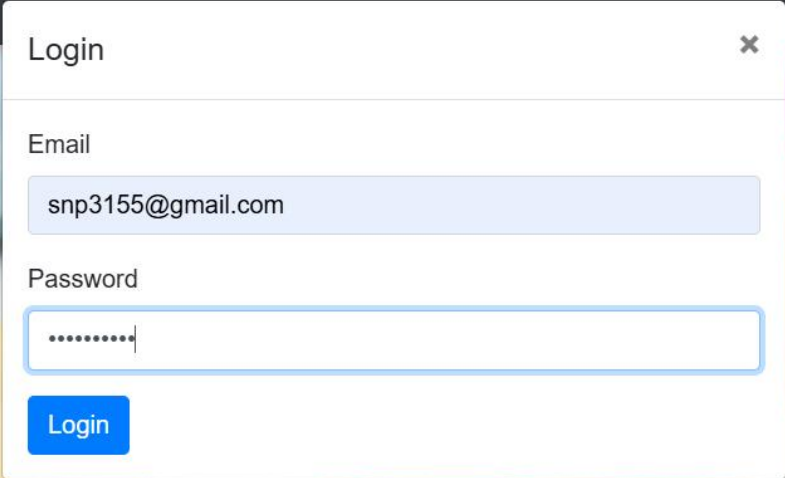
Password

Confirm Password

Profile Picture  
 No file chosen

nt Booking Sy  
current pandemic situation is that the hospitals are bit busy dealing with  
patients showing similar symptoms so its not easy for their regular

Figure 3 Patient Signup



The image shows a 'Login' modal form overlaid on a background of various colored pills. The form includes fields for Email and Password, and a 'Login' button.

**Login** [X]

Email

Password

..compute-1.amazonaws.com:3000

Figure 4 Patient Login

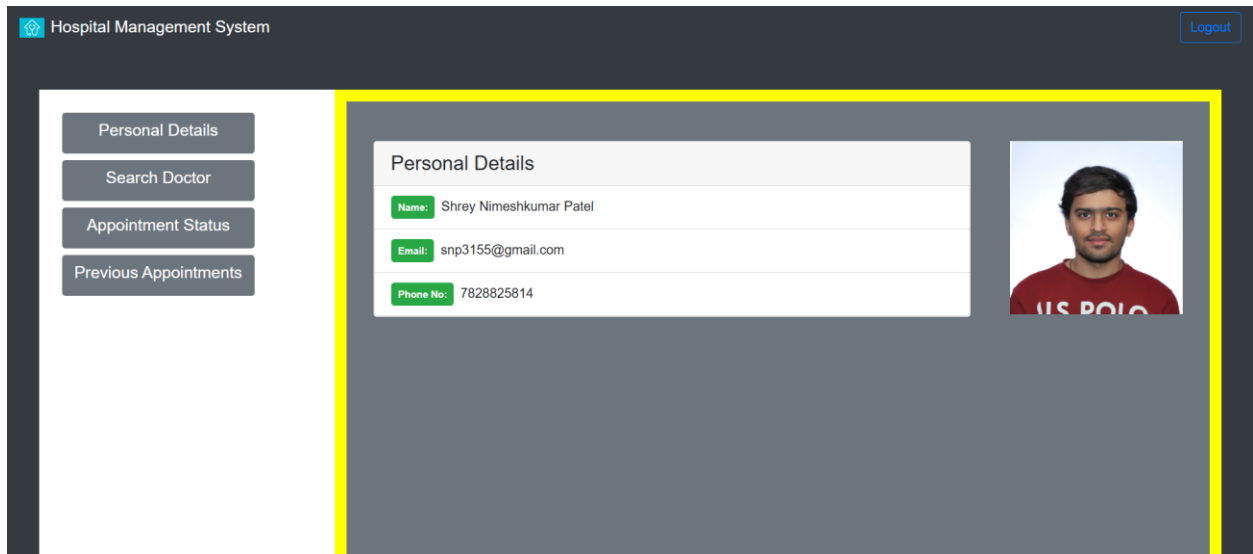


Figure 5 Patient Profile with image from S3

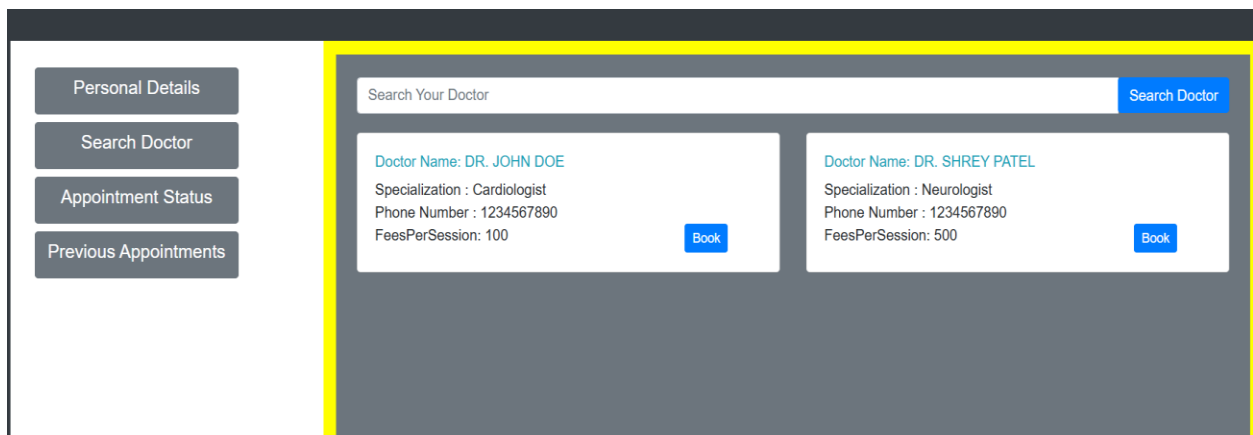


Figure 6 All available doctors

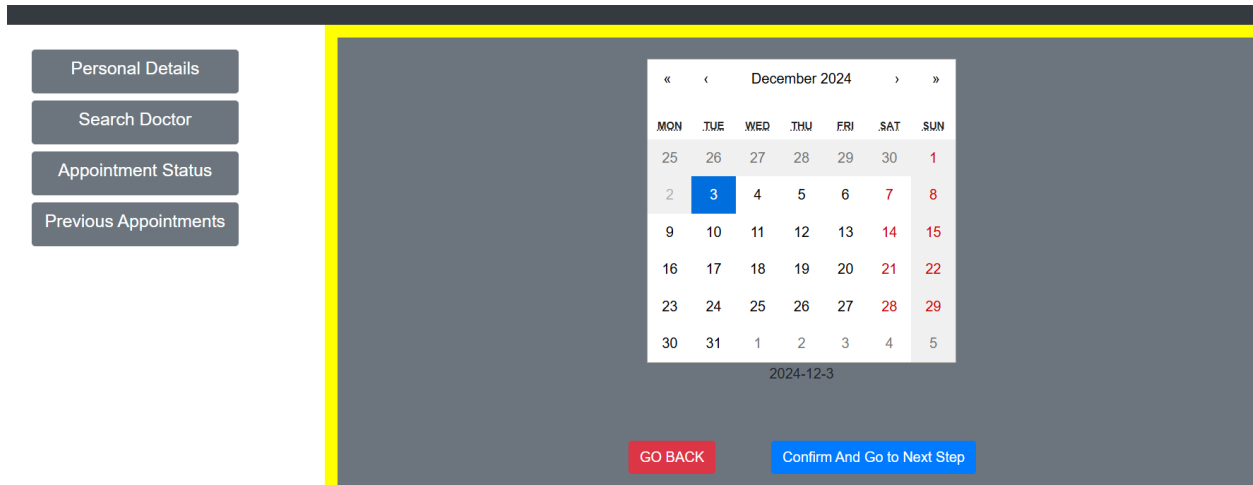


Figure 7 Selection of Date

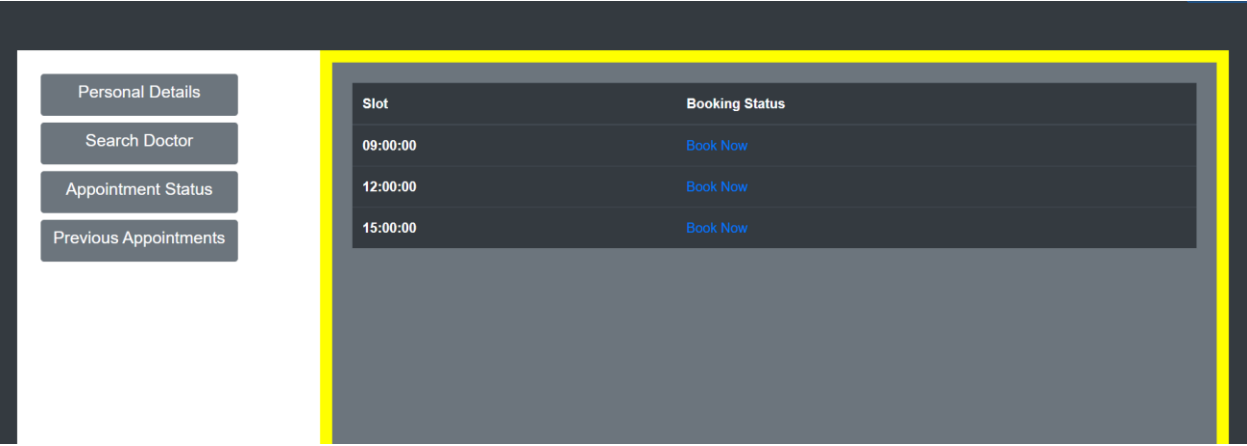


Figure 8 Selection of Time Slot

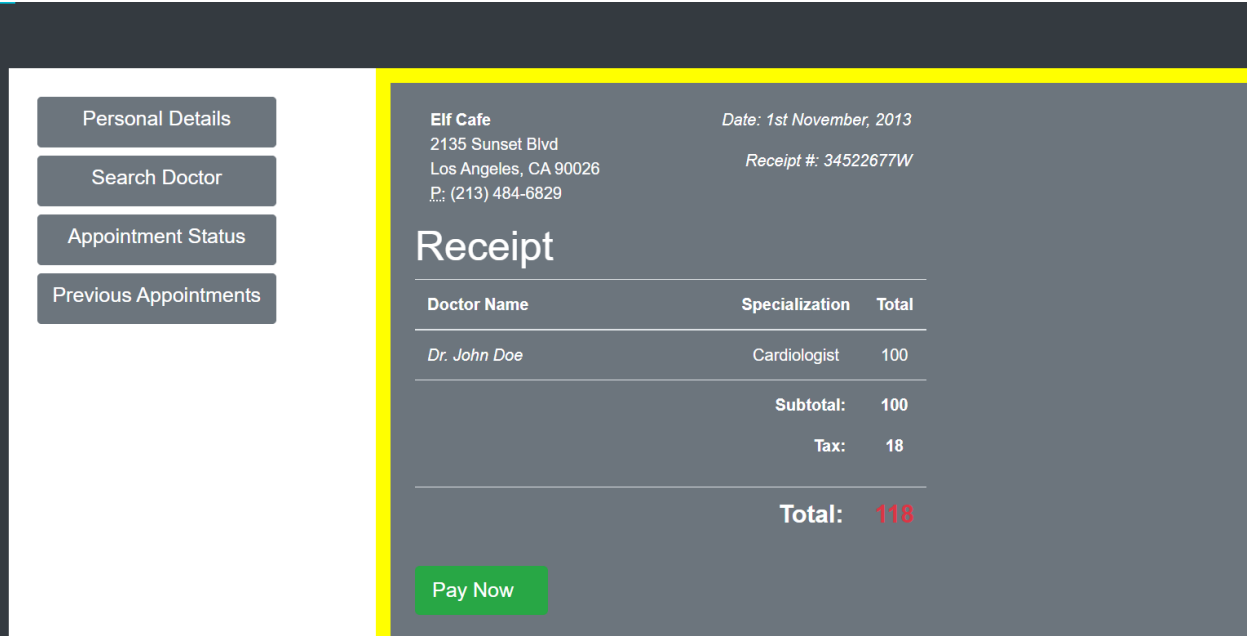


Figure 9 Receipt



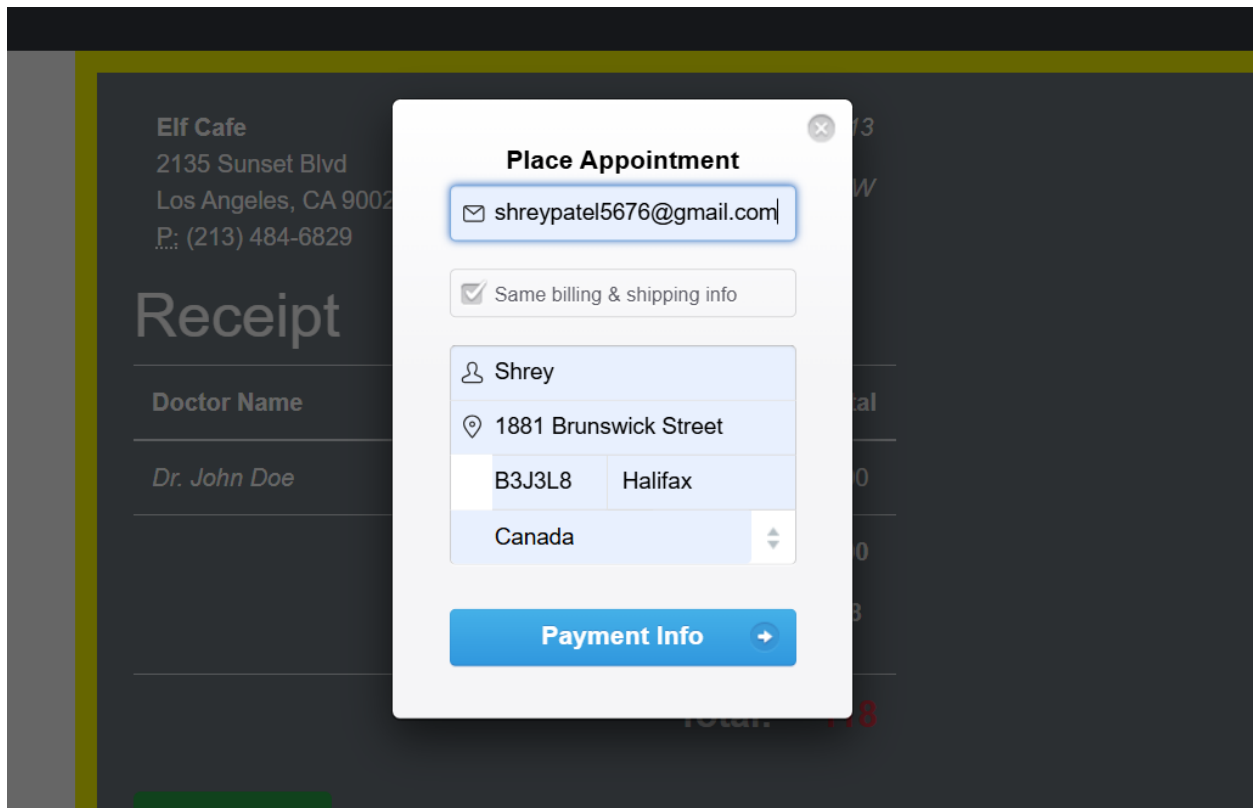


Figure 10 Stripe Payment



Figure 11 Previous Appointment

Hospital Management System Logout

It will be helpful if you share your Experience

★★★★★

Feel Free to Share your Experience

Provide Short Information

[GO BACK](#) [Submit](#)

Figure 12 Patient feedback

Not secure ec2-54-221-190-152.compute-1.amazonaws.com:3000/doctorlogin 🔍 ☆ 🔖

Hospital Management System Login As A Patient

Welcome back, Doc

Username

Password

[Sign In](#)

Figure 13 Doctor Login

Hospital Management System Logout

Today's Schedule

Personal Details

Previous Appointments

Personal Details

**NAME:** DR. SHREY PATEL

**SPECIALIZATION:** Neurologist

**PHONE NO:** 1234567890

**FEES PER SESSION:** 500

Figure 14 Doctor Profile

Today's Schedule	
Personal Details	
Previous Appointments	

Date	Time	Patient Name	Feedback
2024-11-30	12:00:00	Shrey Nimeshkumar Patel	-
2024-11-30	09:00:00	Shrey Patel	-
2024-11-28	12:00:00	Shrey Patel	-
2024-11-27	09:00:00	Shrey Patel	★★★★★ <a href="#">Details</a>

Figure 15 Previous Appointments of doctor

### 3 Justification of the chosen AWS services:

#### 3.1 Services used:

Services Category	AWS Service Name
Compute	Amazon EC2 Autoscaling, Amazon EC2, AWS Lambda
Storage	Amazon Simple Storage Service (S3)
Networking and Content Delivery	Amazon VPC, Elastic Load Balancing (ALB)
Application integration	Amazon Simple Notification Service
Management and Governance	AWS CloudFormation, Amazon CloudWatch
Security, identity, and compliance	AWS Secrets Manager

#### 3.2 Justification:

The architecture leverages a range of AWS services to ensure scalability, security, high availability, and optimal performance. Below is a detailed justification for each service:

##### 1. EC2 for Frontend Deployment (in Public Subnet):

- Amazon EC2 provides a flexible and scalable platform for hosting the frontend of the application. Deploying it in the public subnet ensures that users can easily access the application over the internet[1].
- This allows for high performance and quick response times when delivering static and dynamic content.

##### 2. Internet Gateway:

- The Internet Gateway is used to allow resources in the public subnet (like the frontend EC2 instance) to connect to the internet[2].
- It ensures secure and seamless communication between the public-facing resources and end users.

**3. Virtual Private Cloud (VPC):**

- A dedicated VPC provides isolated network environments for the application.
- It ensures better security, control, and segmentation of resources, with a clear separation between public and private subnets[3].

**4. EC2 Auto-Scaling for Backend (in Private Subnet):**

- Backend services hosted on EC2 instances are placed in the private subnet to enhance security and prevent unauthorized access[4].
- Auto-scaling ensures that the number of backend EC2 instances dynamically adjusts based on demand, ensuring high availability and cost optimization.

**5. Load Balancer for Backend EC2 Instances:**

- A load balancer is used to distribute incoming traffic evenly across multiple backend EC2 instances[5].
- This ensures optimal utilization of resources, avoids bottlenecks, and enhances fault tolerance.

**6. NAT Gateway:**

- The NAT Gateway allows backend EC2 instances in the private subnet to access the internet securely for tasks such as software updates or external API calls[6].
- It prevents direct inbound traffic from the internet to private resources.

**7. S3 Bucket for Storing User Images and Frontend HTML Images:**

- Amazon S3 is a highly durable and scalable storage service used to store user profile images and frontend assets like HTML images[7].
- It provides easy retrieval of static files with minimal latency and supports secure access mechanisms.

**8. Secrets Manager:**

- AWS Secrets Manager securely stores sensitive data, including the password salt, JWT secret key, and hashing algorithms[8].
- It automates secrets rotation, ensuring secure handling of credentials and reducing the risk of unauthorized access.

**9. Simple Notification Service (SNS):**

- SNS is used for sending notifications, such as emails for user activities like login, signup, and doctor appointment bookings[9].
- It provides a reliable and scalable messaging service to keep users informed, improving user engagement.

**10. AWS CloudFormation:**

- SNS is used for maintaining IaC. It is helpful in case of any downfall of the whole infrastructure. It ensures always backup of the whole resources pool and infrastructure. It ensures that the whole setup can be made again whenever and wherever needed[10].

These services collectively ensure the application is robust, highly available, secure, and scalable, meeting the demands of a modern, production-grade web application.

## 4 Implementation of the AWS Well-Architected Framework principles and best practices:

### 4.1 Architecture Diagram:

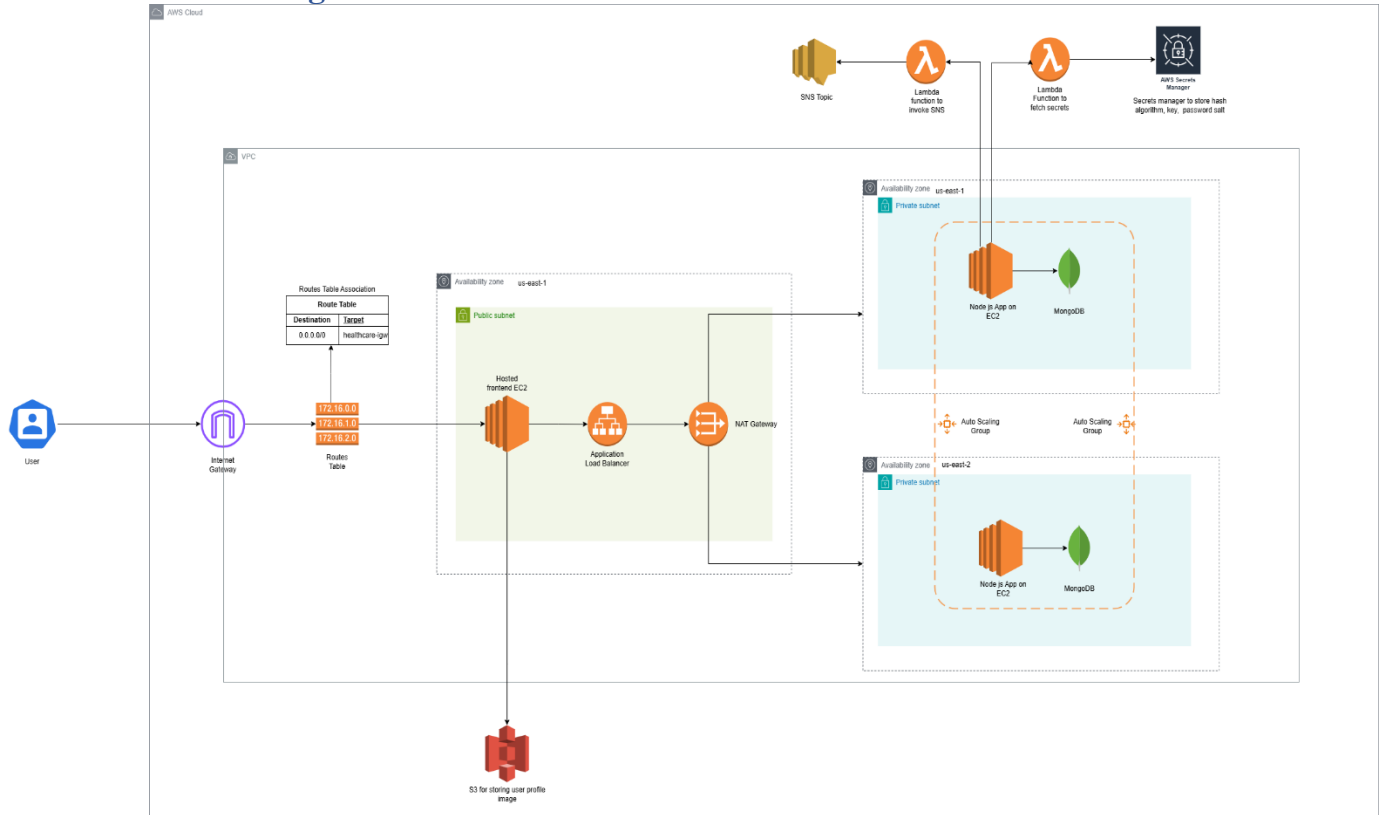


Figure 16 Architecture Diagram

### 4.2 Explanation from 6 well architected framework pillars:

The architecture of this application adheres to the **AWS Well-Architected Framework**, encompassing the six key pillars to ensure a robust, efficient, and scalable solution.

#### 1. Operational Excellence

Operational excellence focuses on the ability to monitor and improve systems to deliver business value efficiently.

- **Logging and Monitoring:**
  - The architecture employs Amazon CloudWatch to monitor the health and performance of EC2 instances (frontend and backend) and autoscaling activities.
  - Logs and metrics are used to identify issues proactively and maintain system reliability.
- **Automation:**
  - Auto-scaling groups for backend EC2 instances in **us-east-1** and **us-east-2** ensure automated scaling based on traffic. This reduces manual intervention[4].
  - Deployment pipelines for EC2 instances (frontend and backend) streamline code updates and patching.

- **Improvement Iterations:**
  - Regular reviews of monitoring data and logs allow for iterative improvements, ensuring optimal performance and adherence to changing requirements.

## 2. Security

Security is central to protecting information, systems, and assets.

- **Network Security:**
  - The backend EC2 instances are deployed in private subnets to restrict direct internet access. A **NAT Gateway** allows controlled outbound traffic for backend services[6].
  - A **VPC** isolates resources from external threats, providing a segmented network structure.
- **Data Protection:**
  - User credentials and sensitive keys (e.g., JWT secret key, password salt) are securely managed in **AWS Secrets Manager**, reducing the risk of exposure.
  - **S3 bucket policies** ensure that user images and other assets are accessible only through authenticated requests.
- **Access Control:**
  - IAM roles and policies enforce the principle of least privilege, ensuring resources and services are accessed only by authorized entities.
  - Multi-factor authentication (MFA) is enabled for administrators managing AWS resources.

## 3. Reliability

Reliability ensures the system can recover from failures and meet operational requirements.

- **Multi-AZ Deployment:**
  - Backend EC2 autoscaling groups span across **us-east-1** and **us-east-2**, ensuring high availability and fault tolerance. In the event of an AZ failure, traffic is routed to healthy instances in another AZ.
  - The **load balancer** distributes incoming requests across all available backend EC2 instances, reducing the impact of instance-level failures.
- **Backup and Recovery:**
  - **S3 versioning** is enabled for user images and assets, allowing recovery in case of accidental deletions.
  - Regular snapshots of backend EC2 instances and the database ensure data durability.
- **Scalability and Health Monitoring:**
  - Auto-scaling ensures backend instances scale dynamically based on demand, preventing downtime during traffic spikes.
  - Health checks for EC2 instances allow the load balancer to route traffic only to healthy resources.

## 4. Performance Efficiency

This pillar ensures efficient use of computing resources to meet demand.

- **Dynamic Scaling:**
  - Backend EC2 instances scale dynamically using auto-scaling groups, ensuring

resources are efficiently utilized during high traffic without incurring unnecessary costs during low usage.

- **Global Reach with Low Latency:**
  - The architecture deploys backend resources in multiple regions (**us-east-1** and **us-east-2**) to minimize latency for users in geographically diverse locations.
  - S3 is used for serving static assets, providing low-latency access globally through AWS edge locations.
- **Resource Optimization:**
  - EC2 instance types are chosen based on the workload requirements of frontend and backend services. Lightweight instances for the frontend and compute-optimized instances for the backend reduce cost and improve efficiency.

## 5. Cost Optimization

Cost optimization minimizes unnecessary expenses while delivering desired outcomes.

- **On-Demand Scaling:**
  - Backend autoscaling minimizes costs by scaling EC2 instances up or down based on real-time demand.
  - S3 storage is used for cost-effective storage of static assets, avoiding the need for large, expensive compute resources.
- **Resource Allocation:**
  - NAT Gateway and Internet Gateway are used efficiently, with the NAT Gateway serving private subnet resources only when necessary.
  - Secrets Manager reduces the overhead of custom key management systems.
- **Rightsizing Instances:**
  - The architecture employs EC2 instances with appropriate compute power and memory to avoid over-provisioning.
  - The use of reserved or spot instances for backend resources can further reduce costs.

## 6. Sustainability

Sustainability focuses on minimizing environmental impact by making efficient use of computing resources.

- **Efficient Resource Usage:**
  - Auto-scaling ensures instances are provisioned only when required, reducing wasteful resource usage.
  - S3 storage with lifecycle policies ensures old or unused files are archived or deleted automatically, saving storage space and reducing energy consumption.
- **Multi-AZ Optimization:**
  - By utilizing multiple AZs in **us-east-1** and **us-east-2**, the architecture spreads workloads effectively across regions, minimizing resource waste and maintaining system efficiency.
- **Serverless Notifications:**
  - Using **SNS** for email notifications offloads the need for additional compute resources to handle messaging, ensuring minimal environmental impact.

## 4.3 Why each service suits my application?

### 1. Amazon EC2 (Frontend Deployment and Backend in Autoscaling Group)

- **Why EC2?**  
EC2 was chosen for its flexibility in customizing the compute environment, ensuring the application's frontend and backend can operate efficiently under varying workloads[11].
- **Cost:**
  - EC2's on-demand and autoscaling features reduce costs by provisioning instances only when traffic increases.
  - The ability to mix instance types (e.g., spot instances for cost-sensitive workloads) ensures cost-effectiveness.
- **Performance:**
  - By deploying backend EC2 instances in an **Auto-Scaling Group** spanning **us-east-1** and **us-east-2**, the system ensures low latency for users across regions.
  - Optimized instance types are used to balance performance and cost for frontend and backend.
- **Security:**
  - Backend instances are in **private subnets**, isolated from direct internet access.
  - Security Groups restrict traffic, allowing only HTTPS traffic from the frontend to the backend.
- **Scalability:**
  - EC2 autoscaling dynamically adjusts the number of backend instances based on traffic, ensuring the application can handle peak loads seamlessly.

### 2. Application Load Balancer (ALB)

- **Why ALB?**  
The load balancer distributes incoming requests to healthy backend EC2 instances, ensuring high availability and fault tolerance[11].
- **Cost:**
  - ALB provides a pay-as-you-go pricing model, charging only for the resources actively used.
- **Performance:**
  - ALB ensures efficient traffic distribution, reducing latency and preventing overloading of backend instances.
  - Health checks detect unhealthy instances and redirect traffic to healthy ones.
- **Security:**
  - ALB supports HTTPS traffic, enabling secure communication between users and the application.
  - It also terminates SSL connections, offloading this heavy process from backend EC2 instances.
- **Scalability:**
  - ALB can handle traffic spikes by routing requests to auto-scaled backend instances across multiple AZs in **us-east-1** and **us-east-2**.

### 3. Amazon VPC

- **Why VPC?**  
VPC allows the creation of an isolated network for secure and efficient communication



between resources.

- **Cost:**
  - VPC is included at no additional cost, with expenses arising only from associated services (e.g., NAT Gateway).
- **Performance:**
  - A dedicated network reduces latency by isolating traffic and ensuring high-speed communication between frontend, backend, and database layers.
- **Security:**
  - Private subnets protect backend EC2 instances, while public subnets enable controlled internet access for the frontend.
  - Access Control Lists (ACLs) and security groups provide additional layers of protection.
- **Scalability:**
  - The VPC design supports growth by allowing easy addition of new subnets, services, or regions.

#### 4. NAT Gateway

- **Why NAT Gateway?**

The NAT Gateway enables backend EC2 instances in private subnets to securely access the internet for updates and external API calls.
- **Cost:**
  - Pay-as-you-go pricing ensures costs are only incurred for traffic processed through the NAT Gateway.
  - It avoids the need for maintaining additional EC2 instances for NAT.
- **Performance:**
  - Highly available and redundant by design, ensuring consistent connectivity for backend resources.
- **Security:**
  - Traffic is controlled, as only specific outbound traffic is allowed, reducing exposure to external threats.
- **Scalability:**
  - Handles large amounts of traffic without performance degradation, essential for applications with growing backend demands.

#### 5. Amazon S3

- **Why S3?**

S3 provides a scalable and durable storage solution for user-uploaded images and static assets (e.g., frontend HTML files, CSS).
- **Cost:**
  - S3's tiered pricing (Standard, Intelligent-Tiering) optimizes costs based on access frequency.
  - Storage is pay-per-use, and unused data can be archived using lifecycle policies to save costs.
- **Performance:**
  - High availability and low latency for delivering images and static assets globally.
  - Integration with **Amazon CloudFront** (optional) can further enhance

performance by caching content at edge locations.

- **Security:**
  - Bucket policies and **IAM roles** ensure only authorized users and services can access the content.
  - Server-side encryption protects data at rest.
- **Scalability:**
  - Unlimited storage capacity ensures the application can scale as the number of users and uploaded images grows.

## 6. Secrets Manager

- **Why Secrets Manager?**

Secrets Manager securely stores and retrieves sensitive information such as password salts, JWT secret keys, and hashing algorithms.
- **Cost:**
  - Secrets Manager is cost-efficient, avoiding the need for complex custom key management systems.
- **Performance:**
  - Seamless integration with AWS SDK ensures low-latency access to secrets when required by backend instances.
- **Security:**
  - Automatic key rotation reduces risks associated with stale credentials.
  - Secrets are encrypted and accessed securely via IAM policies.
- **Scalability:**
  - Secrets Manager scales effortlessly with the number of secrets and requests, ensuring smooth operations as the application grows.

## 7. Amazon SNS

- **Why SNS?**

SNS facilitates email notifications for critical events such as user login, signup, and appointment confirmations.
- **Cost:**
  - Pay-per-use pricing ensures cost efficiency, as charges are incurred only for published messages and delivered emails.
- **Performance:**
  - SNS can handle a high volume of notifications with low latency, ensuring timely communication with users.
- **Security:**
  - Messages are transmitted securely using encryption protocols, and IAM policies control who can publish or subscribe to topics.
- **Scalability:**
  - SNS is inherently scalable, supporting large volumes of notifications even as the user base expands.

# 5 Infrastructure of Code:

## 5.1 Snippet of Successful resources creation:

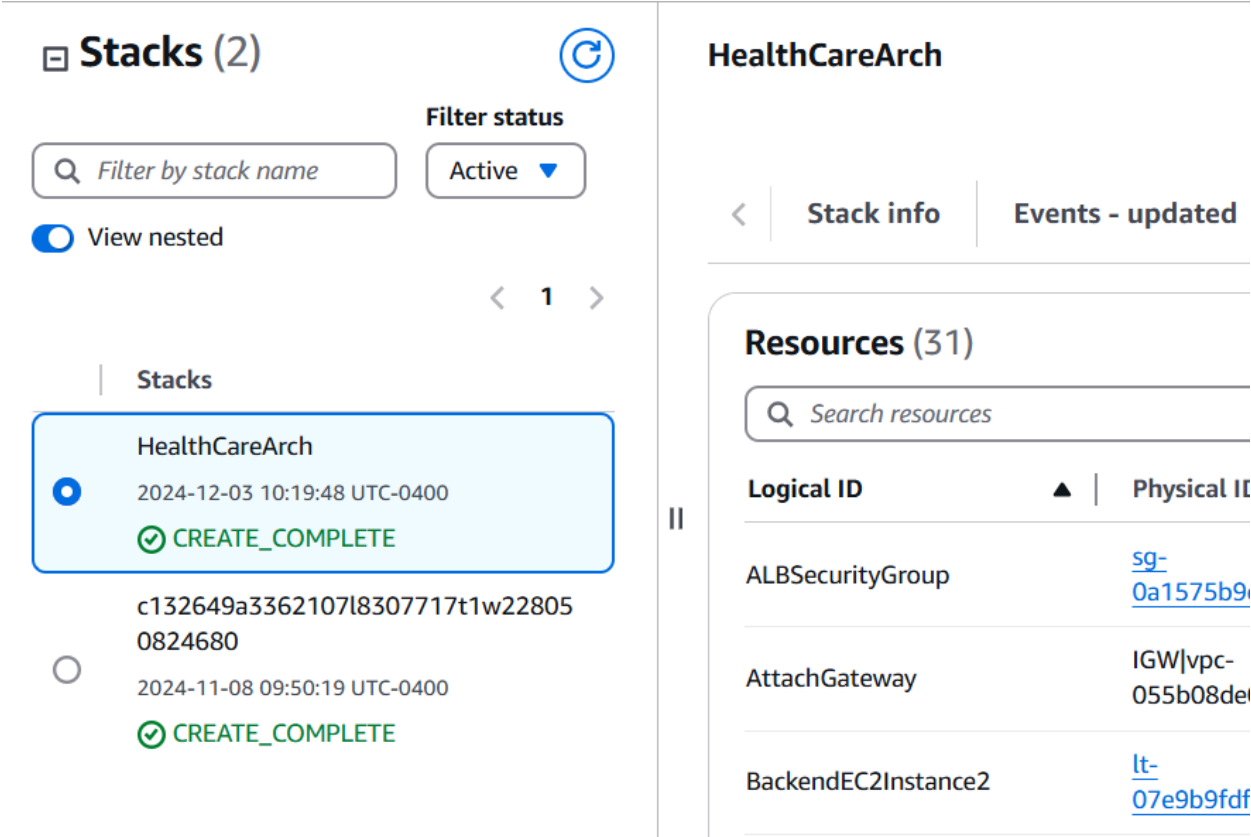


Figure 17 Successful Stack Creation

PublicRoute	rtb-0e74fd8fc37f9741a 0.0.0.0/0	AWS::EC2::Route	✔ CREATE_COMPLETE
PublicRouteTable	rtb-0e74fd8fc37f9741a	AWS::EC2::RouteTable	✔ CREATE_COMPLETE
PublicSubnet1	<a href="#">subnet-01d4b1bfcff63c536</a>	AWS::EC2::Subnet	✔ CREATE_COMPLETE
PublicSubnet1RouteTableAssociation	rtbassoc-05791eb63b39fe1b9	AWS::EC2::SubnetRouteTableAssociation	✔ CREATE_COMPLETE
PublicSubnet2	<a href="#">subnet-0de954d7af20b1d20</a>	AWS::EC2::Subnet	✔ CREATE_COMPLETE
PublicSubnet2RouteTableAssociation	rtbassoc-083aa3cf1de0d9b34	AWS::EC2::SubnetRouteTableAssociation	✔ CREATE_COMPLETE

Figure 18 Resources creation snippet 1

NATGateway	nat-040eb3551b23214e0	AWS::EC2::NatGateway	✔ CREATE_COMPLETE	-
PrivateRoute	rtb-0c2ad1aad6394e8c4 0.0.0.0/0	AWS::EC2::Route	✔ CREATE_COMPLETE	-
PrivateRouteTable	rtb-0c2ad1aad6394e8c4	AWS::EC2::RouteTable	✔ CREATE_COMPLETE	-
PrivateSubnet1	<a href="#">subnet-0c4e9bfa2558470a1</a> 	AWS::EC2::Subnet	✔ CREATE_COMPLETE	-
PrivateSubnet1RouteTableAssociation	rtbassoc-024dc47200278a9bc	AWS::EC2::SubnetRouteTableAssociation	✔ CREATE_COMPLETE	-
PrivateSubnet2	<a href="#">subnet-01ebc9148282bc5dc</a> 	AWS::EC2::Subnet	✔ CREATE_COMPLETE	-
PrivateSubnet2RouteTableAssociation	rtbassoc-0226a702abc2fff75	AWS::EC2::SubnetRouteTableAssociation	✔ CREATE_COMPLETE	-

Figure 19 Resources creation snippet 2



MyScaleOutPolicy	arn:aws:autoscaling:us-east-1:228050824680:scalingPolicy:e6cbf1e4-ac90-4228-9ebc-02ac3e866888:autoScalingGroupName/HealthCareArch-MyAutoScalingGroup-EsQu5Cb6HxD:policyName/HealthCareArch-MyScaleOutPolicy-fiMvMN1SajeX	AWS::AutoScaling::ScalingPolicy	✔ CREATE_COMPLETE	-
MySecret	<a href="#">arn:aws:secretsmanager:us-east-1:228050824680:secret:MySecret-BzWSjs</a> 	AWS::SecretsManager::Secret	✔ CREATE_COMPLETE	-
MyVPC	<a href="#">vpc-055b08de00be08f50</a> 	AWS::EC2::VPC	✔ CREATE_COMPLETE	-

Figure 20 Resources creation snippet 3

	<a href="#">07d6c8cd10a4c30d4</a>			
InternetGateway	<a href="#">igw-09d66b0365a3c8f33</a>	AWS::EC2::InternetGateway	CREATE_COMPLETE	
MyAutoScalingGroup	<a href="#">HealthCareArch-MyAutoScalingGroup-EsQu5Cb6HxhD</a>	AWS::AutoScaling::AutoScalingGroup	CREATE_COMPLETE	
MyScaleInAlarm	<a href="#">HealthCareArch-MyScaleInAlarm-a3bvk24KXVcD</a>	AWS::CloudWatch::Alarm	CREATE_COMPLETE	
MyScaleInPolicy	arn:aws:autoscaling:us-east-1:228050824680:scalingPolicy:81934fb8-b1b8-465b-9e38-0fc866b460ab:autoScalingGroupName/HealthCareArch-MyAutoScalingGroup-	AWS::AutoScaling::ScalingPolicy	CREATE_COMPLETE	

Figure 21 Resources creation snippet 4

ALBSecurityGroup	<a href="#">sg-0a1575b9d878094de</a>	AWS::EC2::SecurityGroup	CREATE_COMPLETE	-
AttachGateway	IGW vpc-055b08de00be08f50	AWS::EC2::VPCGatewayAttachment	CREATE_COMPLETE	-
BackendEC2Instance2	<a href="#">lt-07e9b9fd81001a02</a>	AWS::EC2::LaunchTemplate	CREATE_COMPLETE	-
BackendListener	<a href="#">arn:aws:elasticloadbalancing:us-east-1:228050824680:listener/app/BackendLoadBalancer/09657ec5df296da0/9745c73e9ea89c04</a>	AWS::ElasticLoadBalancingV2::Listener	CREATE_COMPLETE	-
	<a href="#">arn:aws:elasticloadbalancing:us-east-1:228050824680:listener/app/BackendLoadBalancer/09657ec5df296da0/9745c73e9ea89c04</a>	AWS::ElasticLoadBalancingV2::Listener	CREATE_COMPLETE	-

Figure 22 Resources creation snippet 5

## 5.2 Cloud Formation file explanation:

This CloudFormation template provisions infrastructure for a **Healthcare Appointment Scheduling Application**. It defines resources for networking, security, compute, and data storage in a structured and reusable manner. Below is the detailed explanation of the various sections and resources:

### 1. Parameters

Parameters allow dynamic configuration of the template:

- **Prefix:** Prepends a standard name to resource names.
- **emailSender:** Specifies the name of the Lambda function for email notifications.
- **BucketName:** Name of the S3 bucket for appointment scheduling.
- **ApiName:** Name for the API Gateway.
- **DynamoDBTableName:** Table for storing patient data.
- **InstanceType:** Configurable instance size for EC2 instances.
- **KeyName:** Key pair for EC2 SSH access.
- **GitHubLink:** Repository link to fetch application code.
- **VPCName:** Dynamic naming for the VPC.

## 2. Resources

### Networking

- **VPC (MyVPC):** Defines an isolated virtual network with DNS support.
- **Subnets:**
  - **PublicSubnet1 & PublicSubnet2:** For public-facing resources like EC2 frontend and NAT gateway.
  - **PrivateSubnet1 & PrivateSubnet2:** For backend EC2 and database instances.
- **Internet Gateway (InternetGateway):** Enables internet access for the VPC.
- **Route Tables:**
  - **PublicRouteTable:** Routes traffic from public subnets to the internet.
  - **PrivateRouteTable:** Routes private subnet traffic through a NAT gateway for secure internet access.
- **NAT Gateway:**
  - Provides internet access to private resources via **ElasticIP**.

### Secrets Manager

- **MySecret:** Stores sensitive configuration details for the frontend and backend, including API keys, client ID, secret keys, etc., for secure access management.

### Security Groups

Defines access rules for EC2 instances:

- **FrontendSecurityGroup:** Allows SSH and HTTP access for frontend servers (ports 22, 3000).
- **BackendSecurityGroup:** Permits traffic from frontend and ALB (ports 80, 5000).
- **ALBSecurityGroup:** Secures access to the Application Load Balancer.

### Compute Resources

- **FrontendEC2Instance:**
  - Launches an EC2 instance for the frontend application in a public subnet.
  - Installs Node.js and pulls code from the GitHub repository.
  - Starts the frontend React app and sets up environment variables.
- **BackendEC2Instance2:**
  - Uses a launch template for scalable backend instances.
  - Configures MongoDB, Node.js, and backend application setup.

### Storage

- **S3 Bucket** (specified via the parameter BucketName): Can store application-related files, logs, or artifacts.

### Auto Scaling

Auto Scaling ensures your application can handle varying levels of traffic while optimizing cost and performance by automatically adjusting the number of EC2 instances. Below is an explanation of the key components of the Auto Scaling setup:

1. **Auto Scaling Group (MyAutoScalingGroup):**

- **Purpose:** Automatically launches and terminates EC2 instances based on the defined scaling policies.
- **Key Properties:**
  - **MinSize / MaxSize / DesiredCapacity:** Defines the minimum, maximum, and initial number of instances.
  - **LaunchTemplate:** Specifies the template that defines how the instances should be configured.
  - **VPCZoneIdentifier:** Lists the subnets where instances will be deployed.
  - **HealthCheckType:** Configured to use Elastic Load Balancer (ELB) for health checks.
  - **Tags:** Tags the instances for identification and organization.
  - **TargetGroupARNs:** Associates the instances with a target group for load balancing.

2. **Scaling Policies:**

- **Scale Out Policy (MyScaleOutPolicy):**
  - Triggers an increase in capacity by one instance when certain conditions are met (e.g., high CPU usage).
- **Scale In Policy (MyScaleInPolicy):**
  - Reduces capacity by one instance when load decreases.

3. **CloudWatch Alarms:**

- Monitor CPU utilization to trigger scaling policies:
  - **MyScaleOutAlarm:** Activates the scale-out policy when average CPU utilization exceeds 70%.
  - **MyScaleInAlarm:** Activates the scale-in policy when CPU utilization drops below 30%.

## **Load Balancer**

A Load Balancer distributes incoming application traffic across multiple targets (e.g., EC2 instances), ensuring high availability and fault tolerance. Below is an explanation of the load balancer setup:

1. **Application Load Balancer (BackendLoadBalancer):**

- **Type:** Application Load Balancer (ALB), suitable for HTTP/HTTPS traffic.
- **Scheme:** internet-facing allows external users to access the backend services.
- **Subnets:** Configured for public subnets to allow access from the internet.
- **Security Groups:** Protects the load balancer by restricting traffic using inbound and outbound rules.

2. **Target Group (BackendTargetGroup):**

- Defines the EC2 instances that the load balancer will route requests to.
- **Key Properties:**
  - **Port and Protocol:** Specifies that traffic will be forwarded to port 5000 using HTTP.
  - **Health Check:** Regularly checks the health of targets using / as the health check path.

3. **Listener (BackendListener):**
  - Listens for HTTP requests on port 80 and forwards them to the target group (BackendTargetGroup).

## 6 References:

1. **Amazon EC2**, Amazon Web Services, "Amazon EC2: Scalable Cloud Hosting," 2024. [Online]. Available: <https://aws.amazon.com/ec2/>. [Accessed: Dec 2 2024].
2. **Internet Gateway**, Amazon Web Services, "Amazon VPC Internet Gateways," 2024. [Online]. Available: [https://docs.aws.amazon.com/vpc/latest/userguide/VPC\\_Internet\\_Gateway.html](https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Internet_Gateway.html). [Accessed: Dec 2 2024].
3. **Virtual Private Cloud (VPC)**, Amazon Web Services, "Amazon VPC: Isolated Cloud Resources," 2024. [Online]. Available: <https://aws.amazon.com/vpc/>. [Accessed: Dec 2 2024].
4. **EC2 Auto Scaling**, Amazon Web Services, "Amazon EC2 Auto Scaling: Scale Capacity Automatically," 2024. [Online]. Available: <https://aws.amazon.com/autoscaling/ec2/>. [Accessed: Dec 2 2024].
5. **Load Balancer**, Amazon Web Services, "Elastic Load Balancing: Automatically Distribute Traffic," 2024. [Online]. Available: <https://aws.amazon.com/elasticloadbalancing/>. [Accessed: Dec 2 2024].
6. **NAT Gateway**, Amazon Web Services, "Amazon VPC NAT Gateways," 2024. [Online]. Available: <https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html>. [Accessed: Dec 2 2024].
7. **Amazon S3**, Amazon Web Services, "Amazon S3: Object Storage Built to Retrieve Any Amount of Data," 2024. [Online]. Available: <https://aws.amazon.com/s3/>. [Accessed: Dec 2 2024].
8. **AWS Secrets Manager**, Amazon Web Services, "AWS Secrets Manager: Securely Manage Secrets," 2024. [Online]. Available: <https://aws.amazon.com/secrets-manager/>. [Accessed: Dec 2 2024].
9. **Amazon SNS**, Amazon Web Services, "Amazon Simple Notification Service (SNS)," 2024. [Online]. Available: <https://aws.amazon.com/sns/>. [Accessed: Dec 2 2024].
10. **AWS CloudFormation**, Amazon Web Services, "AWS CloudFormation: Infrastructure as Code," 2024. [Online]. Available: <https://aws.amazon.com/cloudformation/>. [Accessed: Dec 2 2024].
11. **C. Osuji**, "IaC: Create an Auto Scaling Group & Application Load Balancer with AWS CloudFormation," *Medium*, Mar. 5, 2021. [Online]. Available: <https://chinelosuji.medium.com/iac-create-an-auto-scaling-group-application-load-balancer-with-aws-cloudformation-6ff154afdec7>. [Accessed: Dec 2 2024].