Installation and setup

Setting up AWS

To connect to our project, we set up a VPC to define the security groups for connecting to our application.

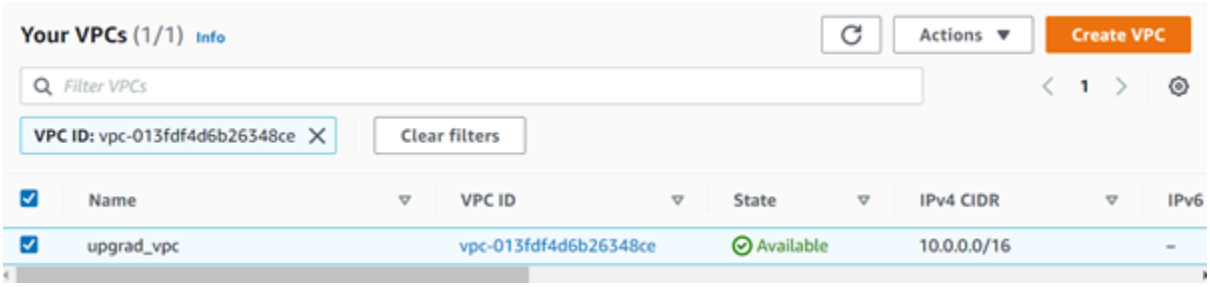


Fig 1.1: VPCs setup

To run our project, we are going to use two of AWS' EC2 instances. An EC2 instance to run the containerized application, and another to run the Kafka-based notification queue.

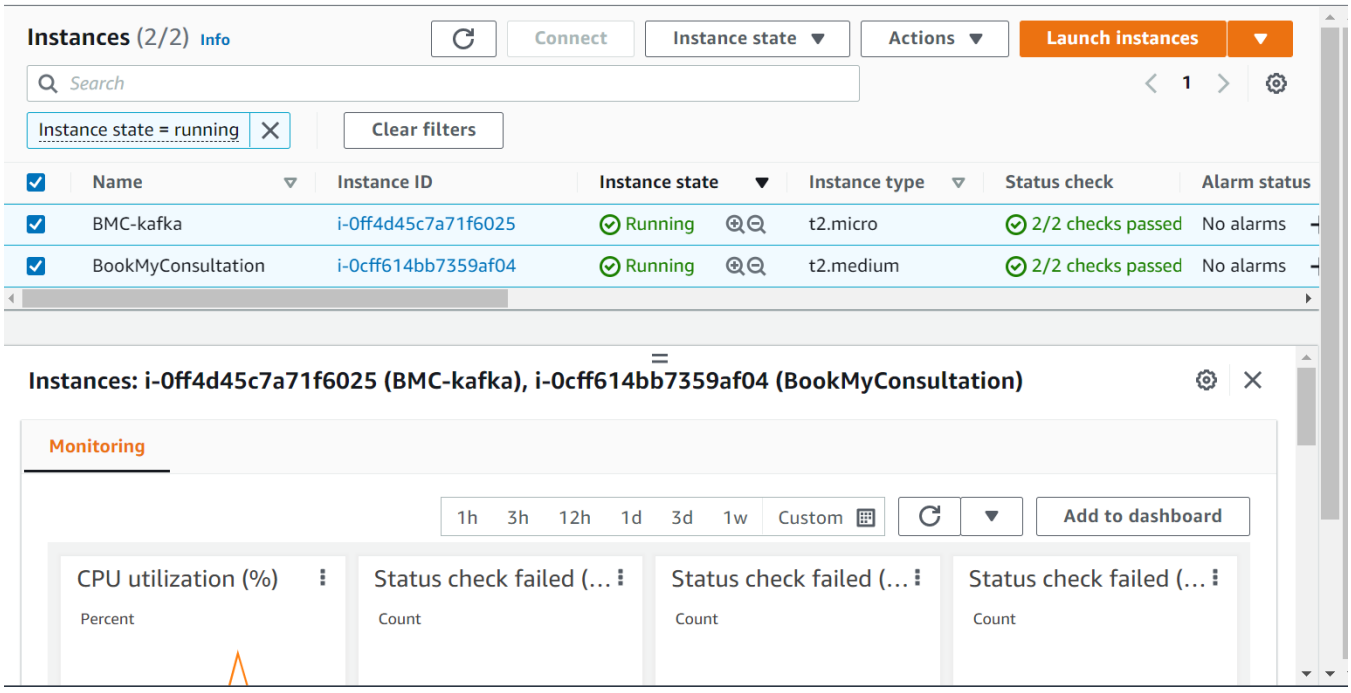


Fig 1.2: EC2 instances setup

We use an RDS instance to store our SQL databases, BookMyConsultation. An RDS instance was created in my AWS account as below.

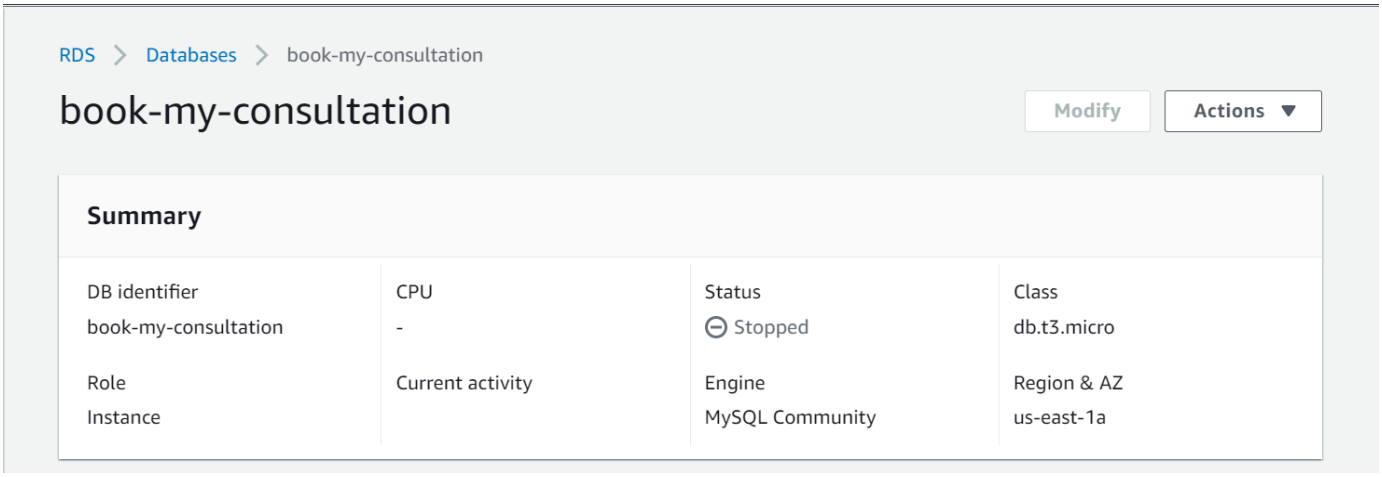


Fig 1.3: RDS instances setup

We also need to create a Mongo DB to store our NoSQL data of the Doctor, User, Prescriptions and Rating collections. I used the Atlas MongoDB for my collections as below.

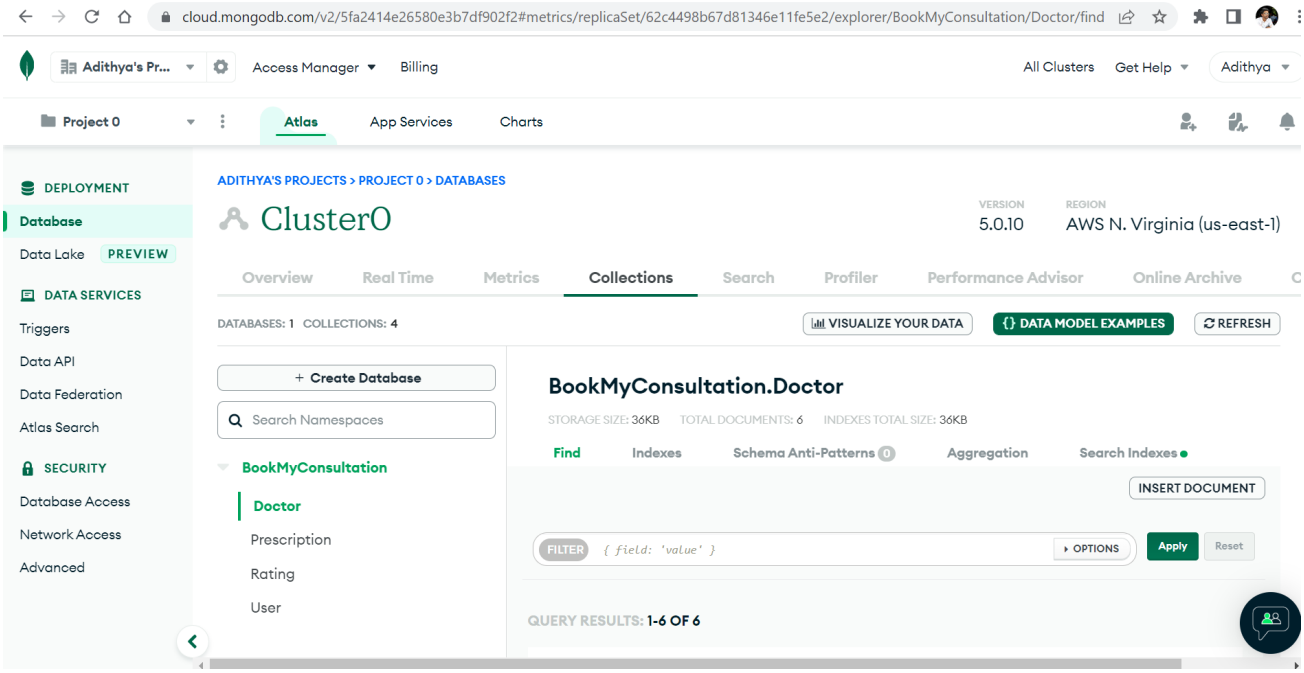


Fig 1.4: MongoDB setup

Setting up Application

1. Application Instance

To connect to our application instance, we ssh into our instance using the key-value pair generated during creation.

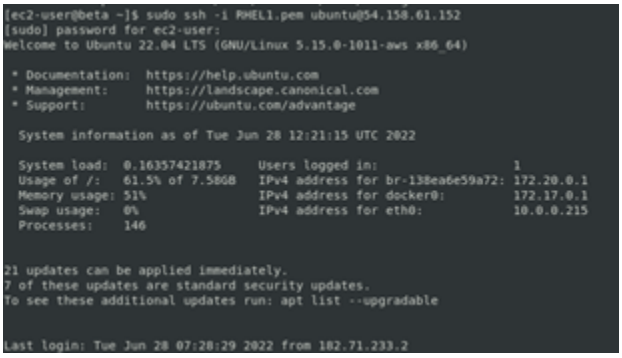


Fig 2.1a: Login to EC2 instance

Once inside, we install docker and docker-compose in the EC2 instance for running our application

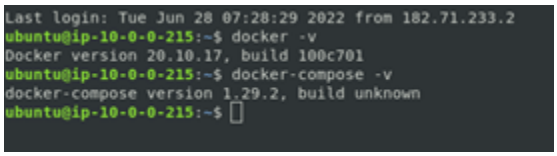


Fig 2.1b: Docker versions

Next, we move the application codebase into the ec2 instance using WinSCP

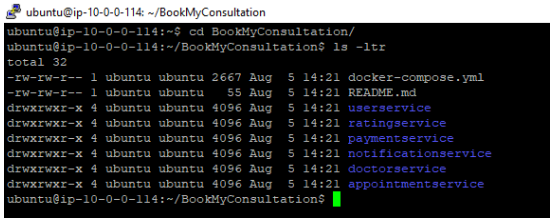


Fig 2.1c: Uploaded codebase

2. Kafka Instance

For the Kafka instance, we connect to the instance using ssh and the key-value pair generated/selected during creation. We download Java onto the instance and then follow the instructions given in the manual to download and set up Kafka and zookeeper.

Fig 2.2: Kafka server setup

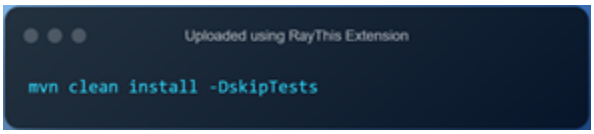
Implementation

Creating Dockerfiles

Once the code base is in the AWS instance, we create Dockerfiles for each application in order to containerize each application.

Each component needs to be compiled and pushed into a jar file, and that jar file needs to be uploaded into a java image for running.

To be compiled, must navigate into each service folder and run the following command to generate the jar file



Once the jar file is generated, we use the official java image to upload the jar file and set an entry point to run the jar file on startup

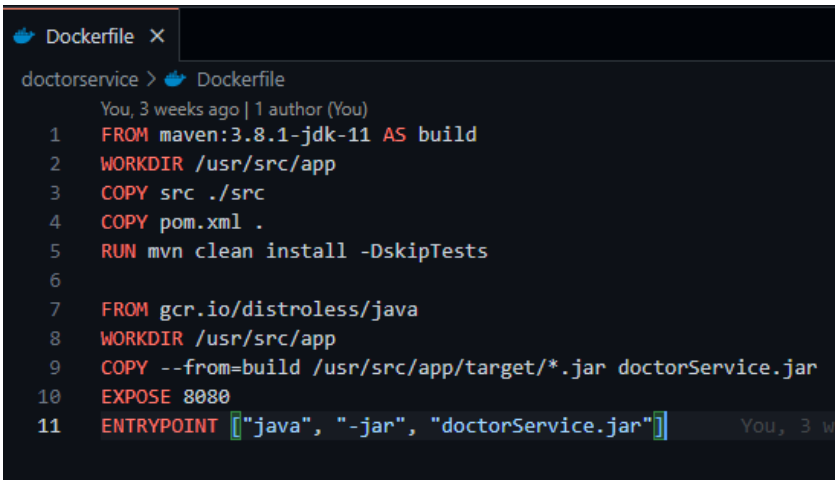


Fig 3.1: Dockerfile for DoctorService

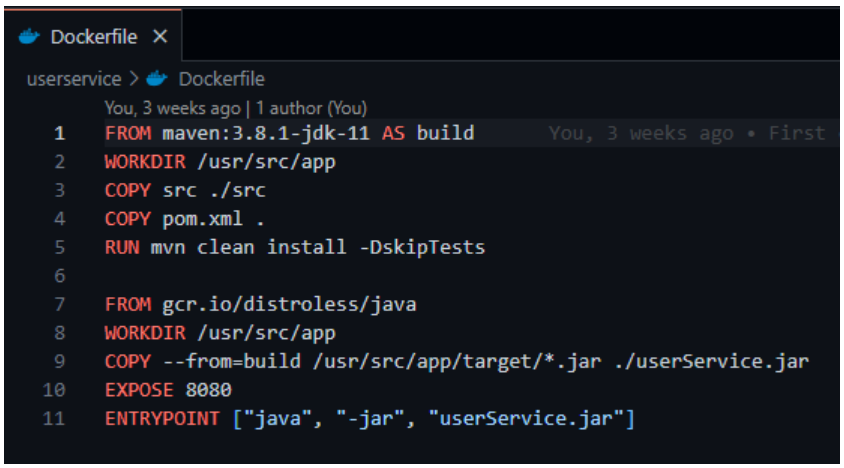


Fig 3.2: Dockerfile for UserService

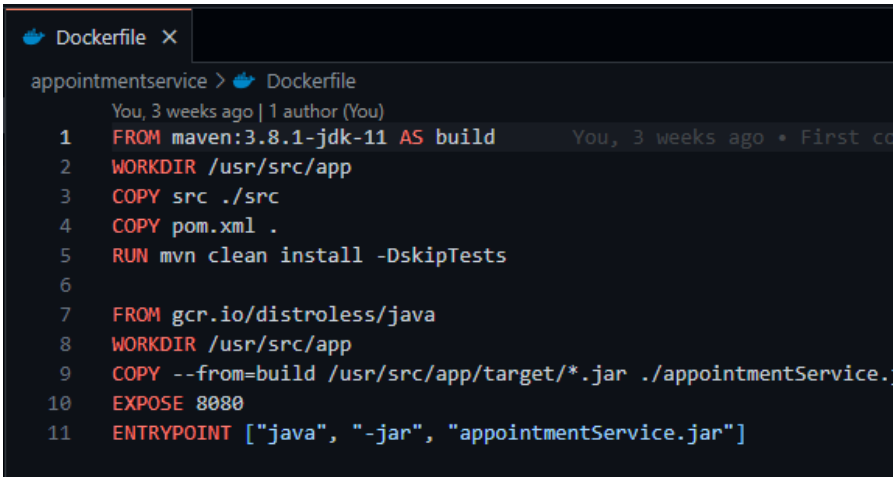


Fig 3.3: Dockerfile for AppointmentService

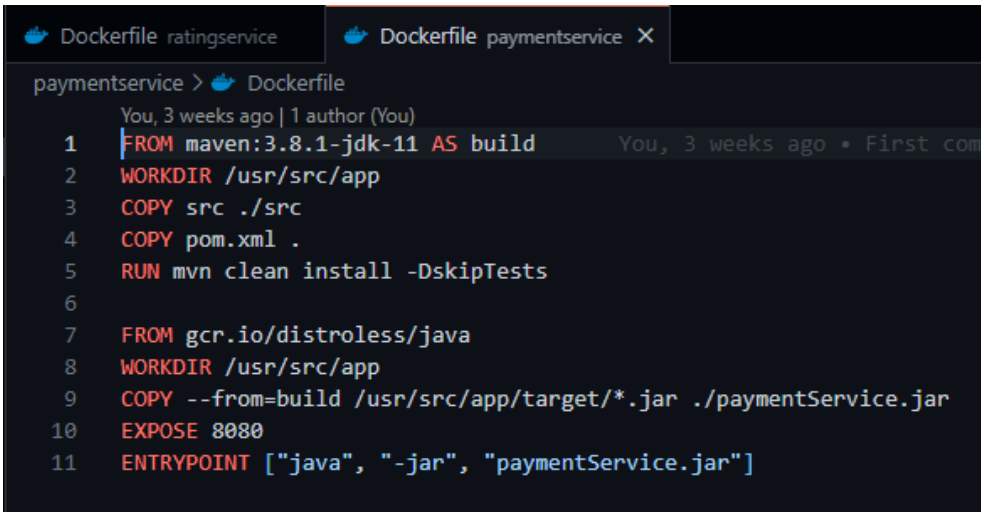


Fig 3.4: Dockerfile for PaymentService

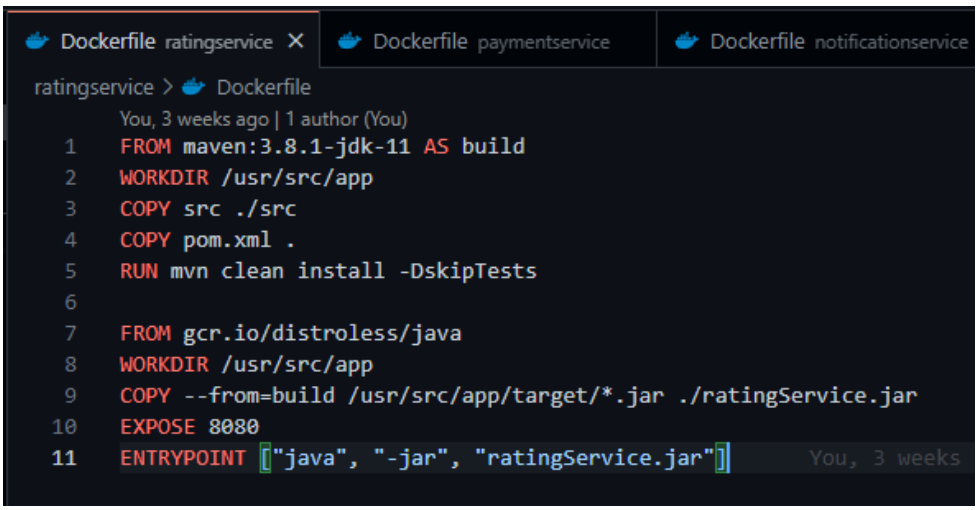


Fig 3.5: Dockerfile for ratingService

Fig 3.6: Dockerfile for NotificationService

Creating Docker-Compose File

We also set up the docker-compose file to build and deploy the services through the Dockerfiles.

Running the application

To begin, let's start the Kafka server in order for it to be ready when we start our service application.

To do that, we log into our Kafka instance and navigate to the kafka folder. Within the folder, we run the following commands.

Fig 5.1: start Kafka server

Before we start the services application, we need to build the images for each of the services we are deploying. To build the image for each service and tag them accordingly, we use

Fig 5.2: build service images

Once the build is done, we can start all the applications simultaneously using

Fig 5.3: start services

Once all the applications are up, the docker images and containers should be up and running.

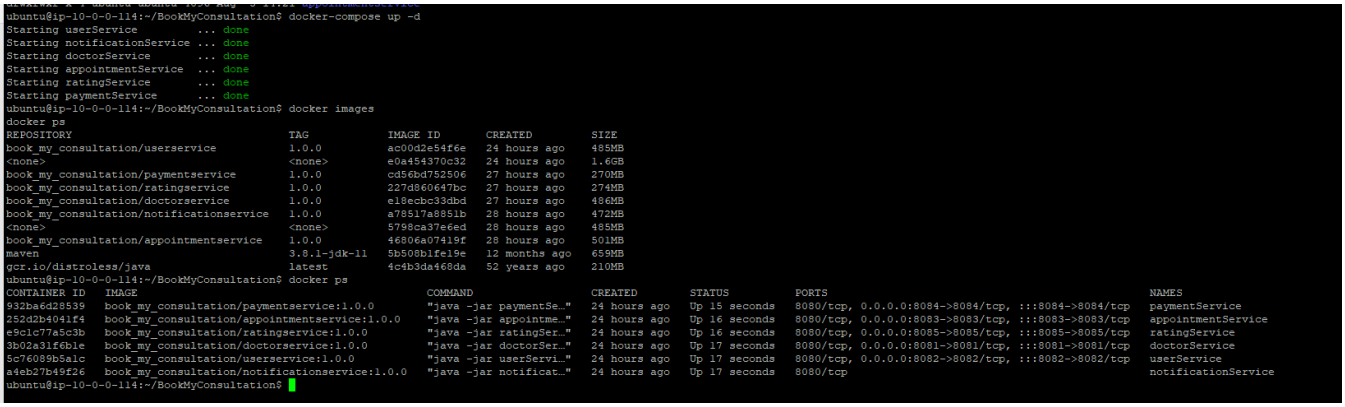


Fig 5.4: docker images and containers

Testing with APIs

DoctorService

1. CreateDoctor

For this API, we validate the details of the Doctor including the first name, last name, email Id and PAN ID. If any of the validations fail, we return an error elaborating on the same.

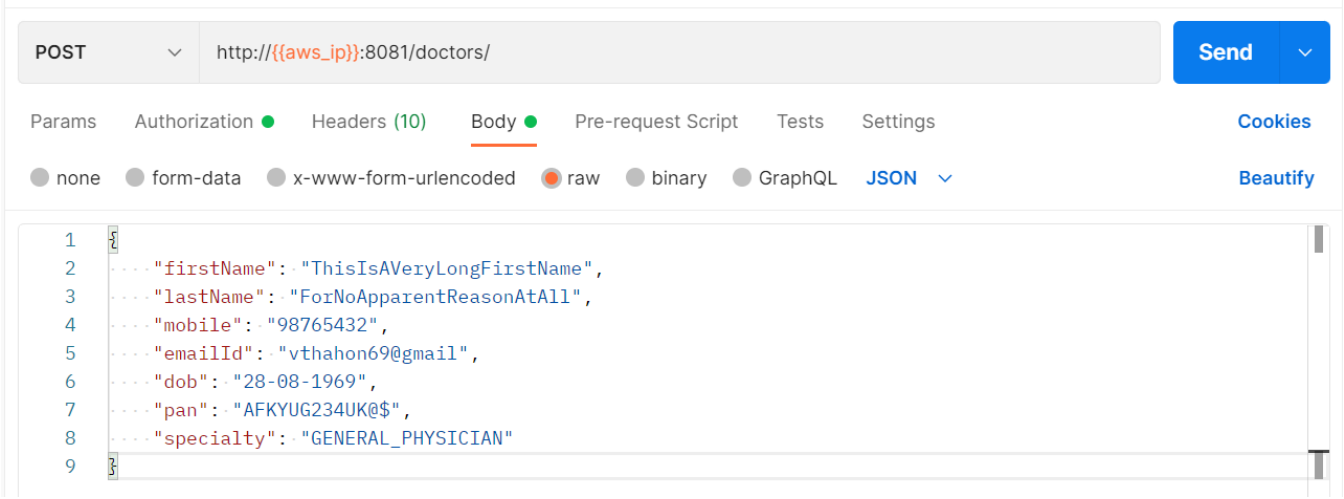


Fig 6.1a: Incorrect Details to create doctor

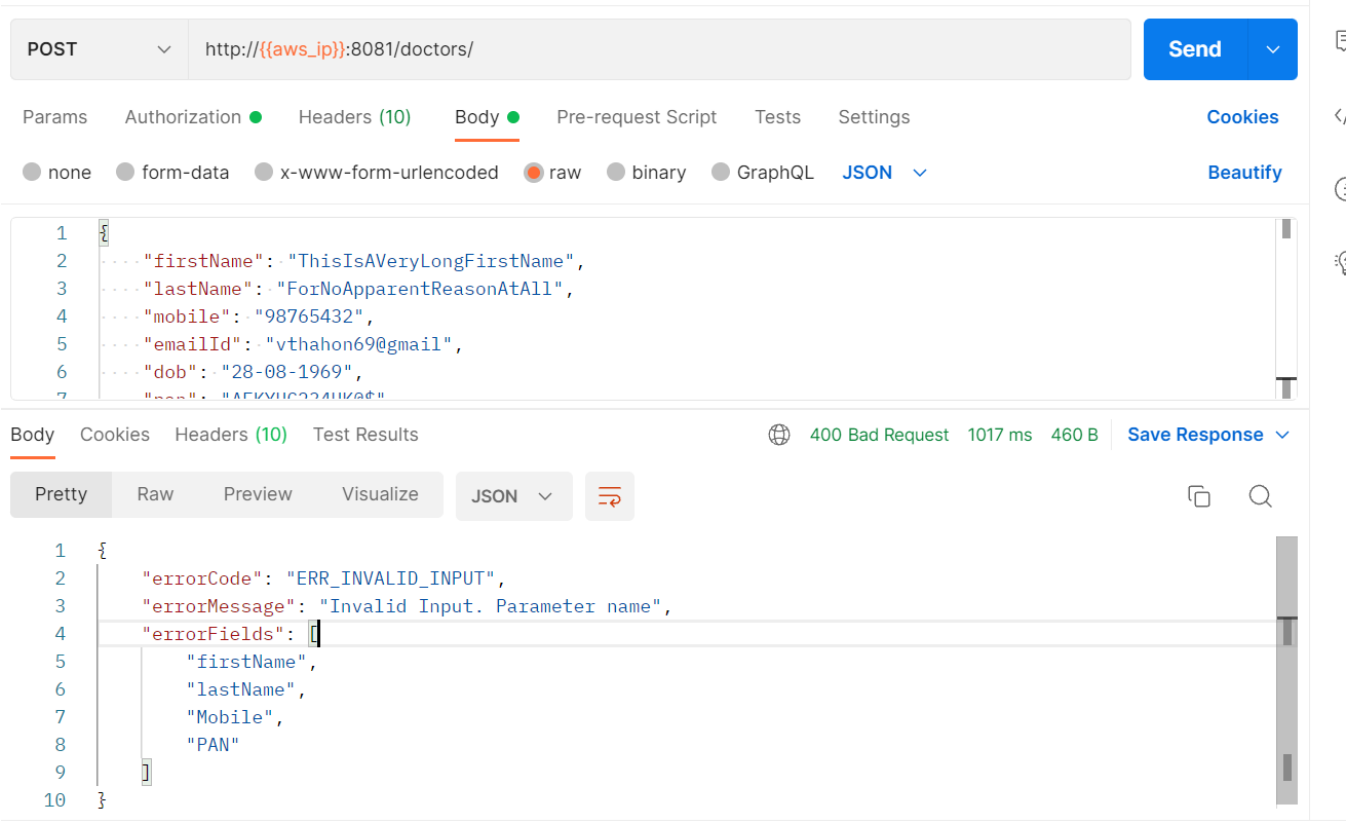


Fig 6.1b: Response for incorrect details to create Doctor

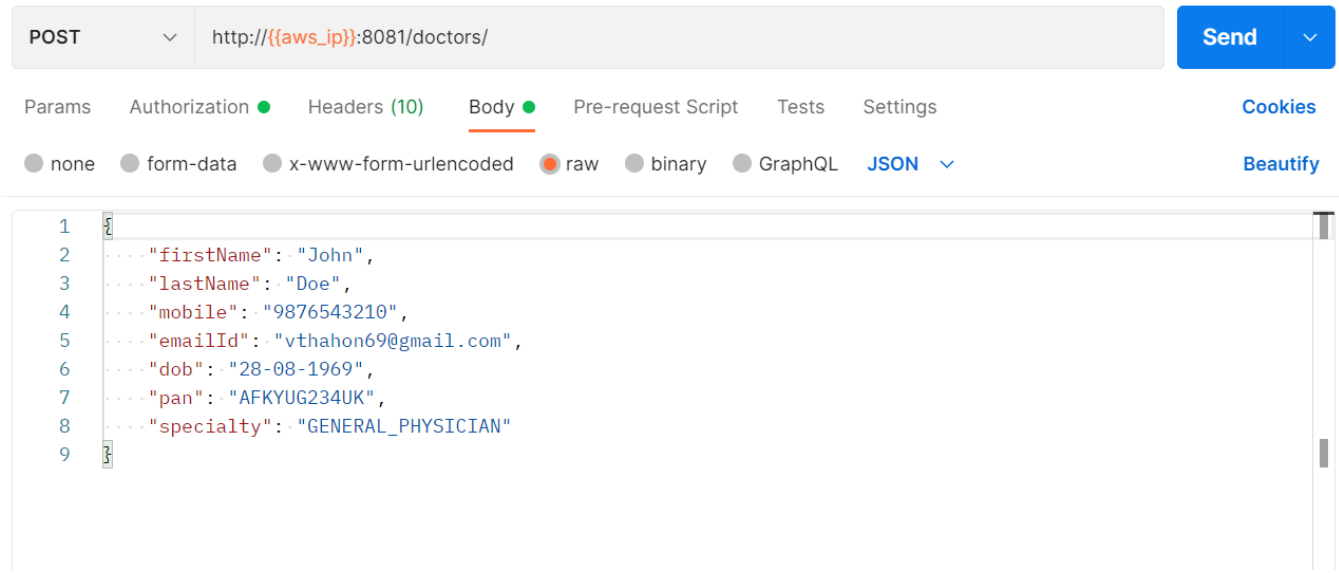


Fig 6.1c: Correct details to create doctor

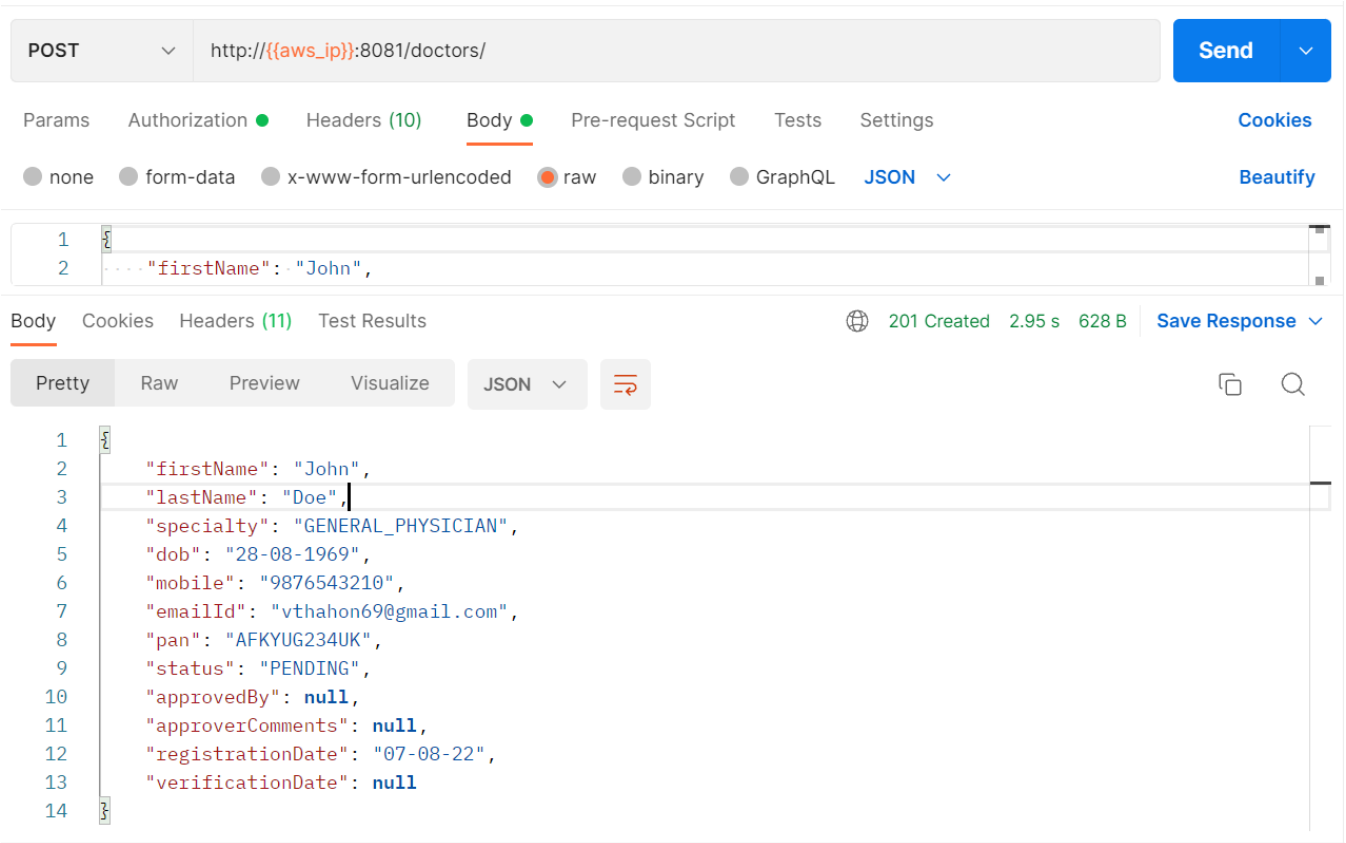


Fig 6.1d: Response for correct details to create doctor

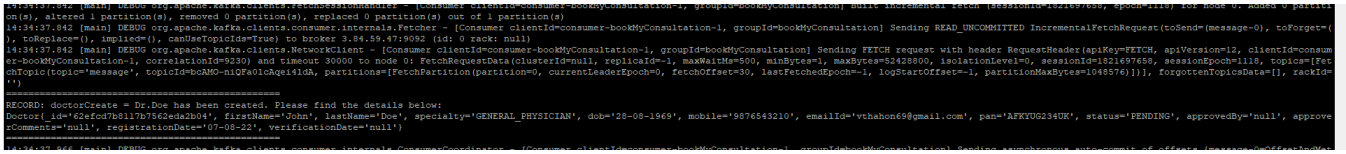


Fig 6.1e: Notification sent to Kafka server for Doctor creation

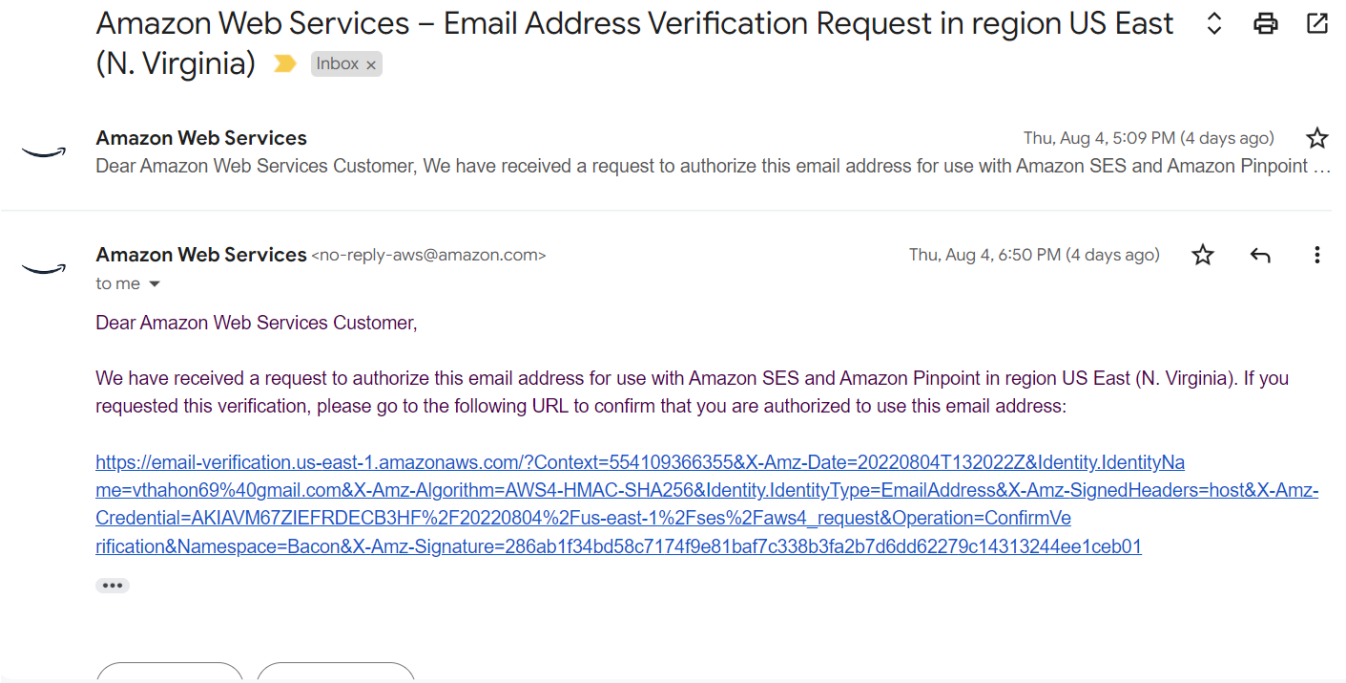


Fig 6.1f: Verification email sent to Doctor Email

2. Upload Documents to Doctor S3 bucket

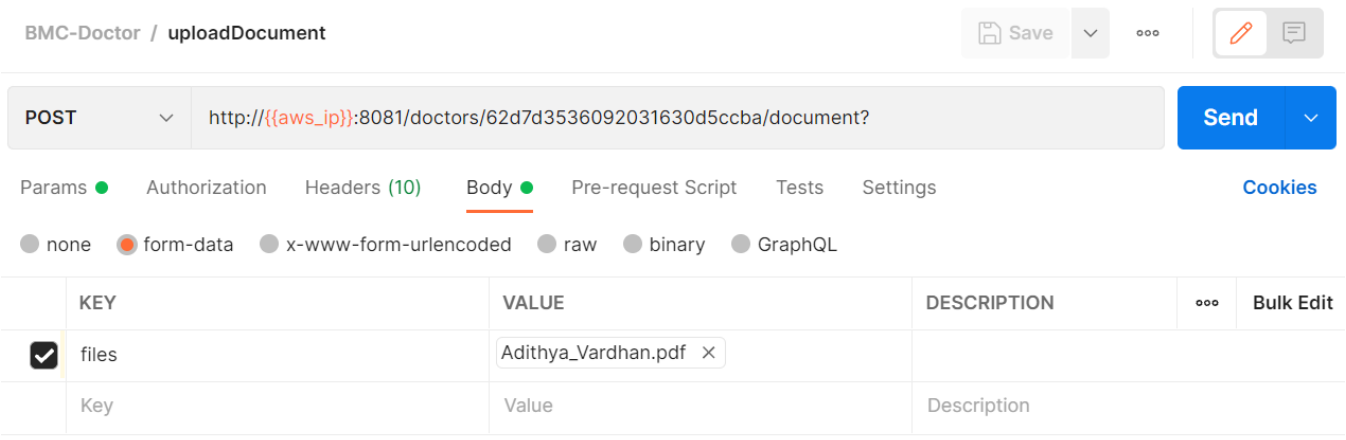


Fig 6.2a: Request to upload documents for Doctor

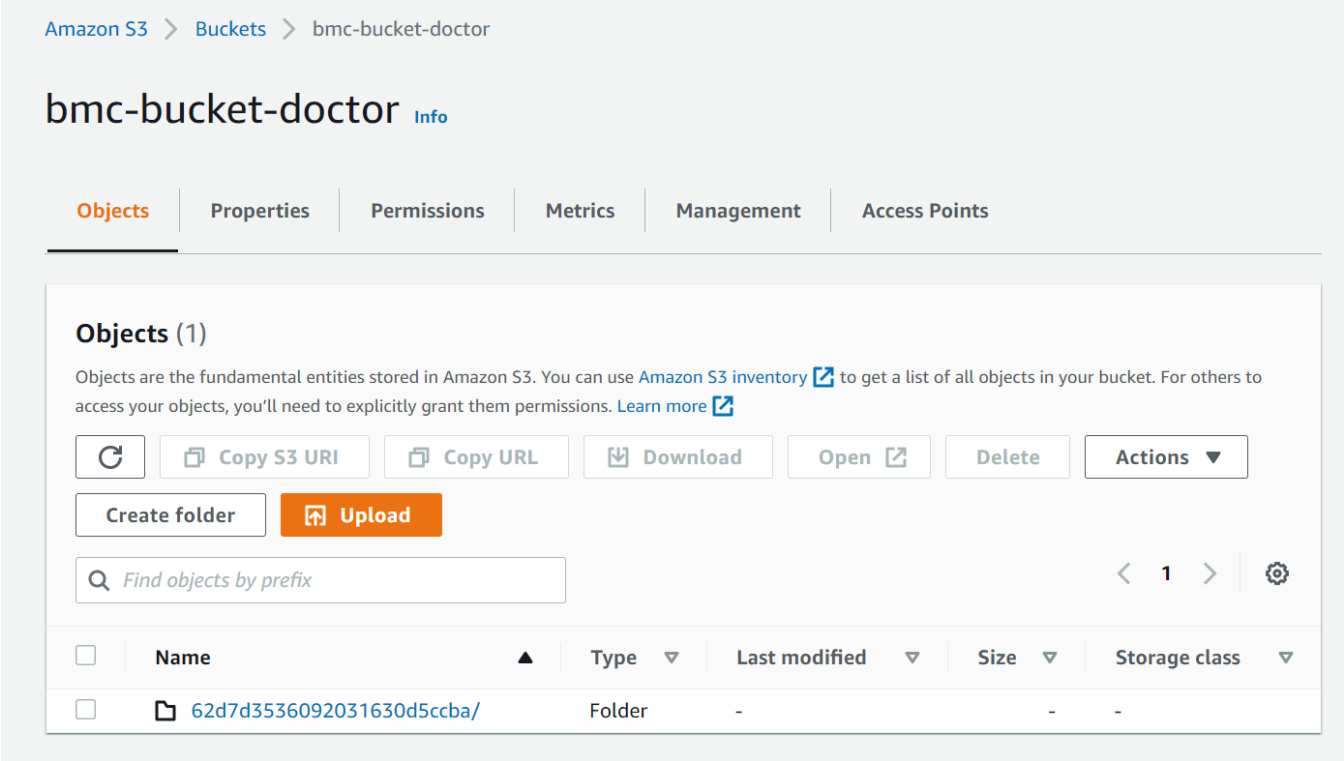


Fig 6.2b: Documents uploaded to Amazon S3

3. Approve Doctor

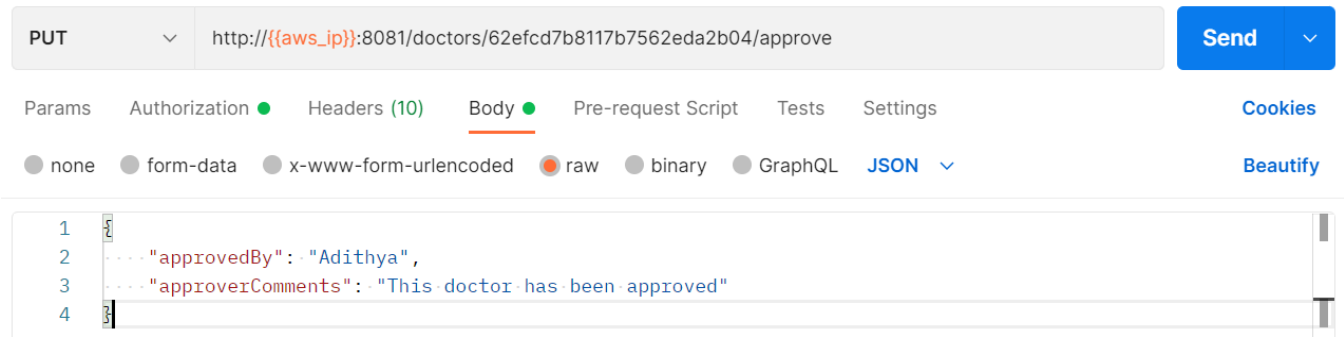


Fig 6.3a: Request to approve doctor

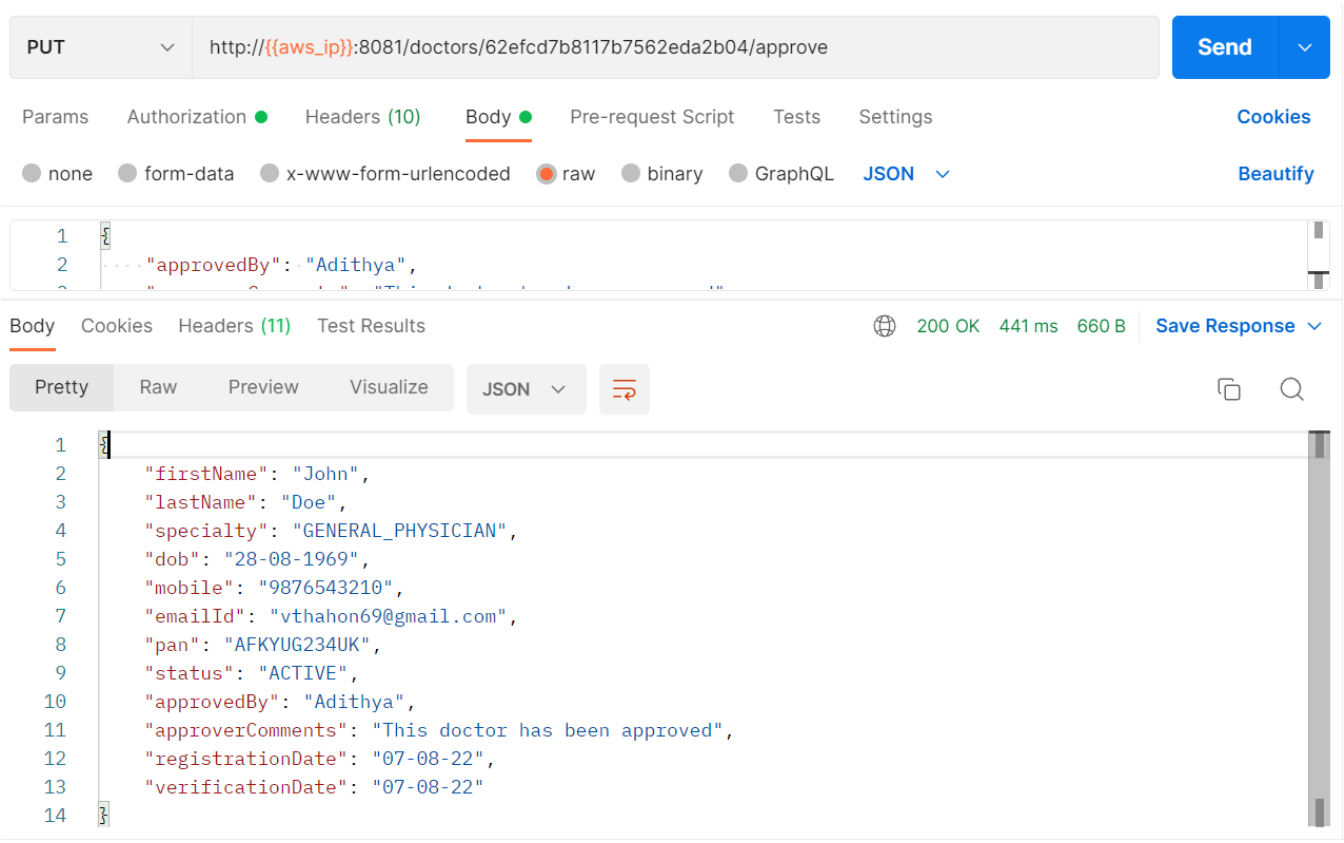


Fig 6.3b: Response for approving doctor

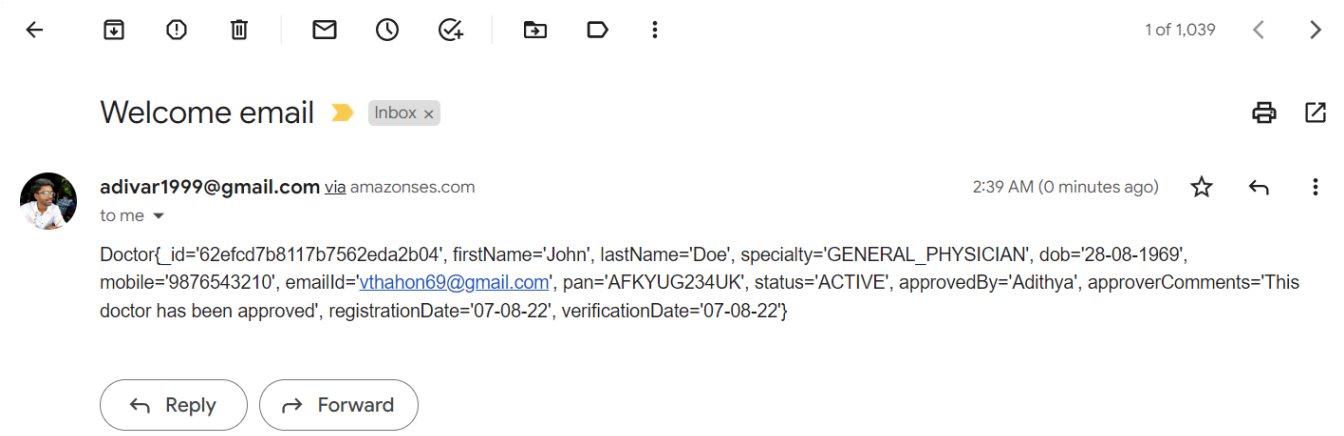


Fig 6.3c: Email for Doctor Approval

4. Reject Doctor

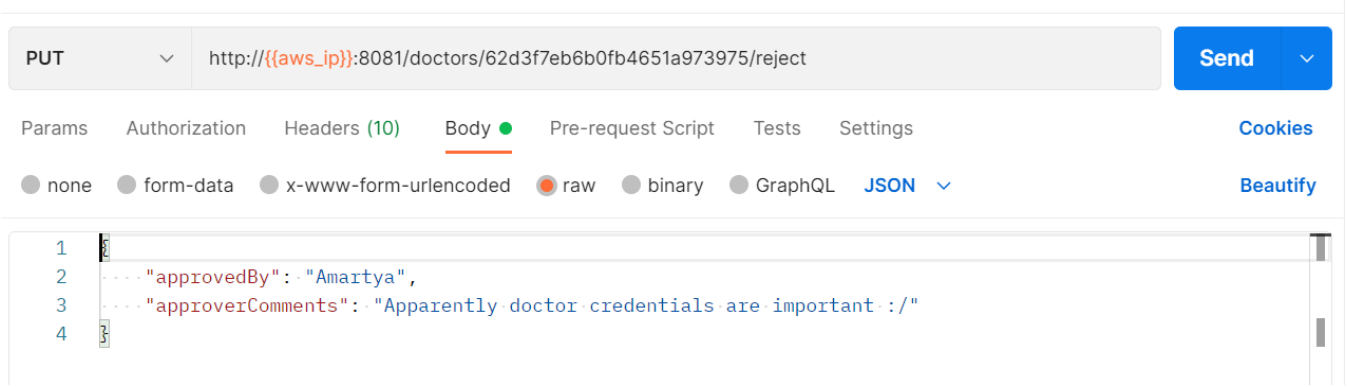


Fig 6.4a: Request to reject Doctor

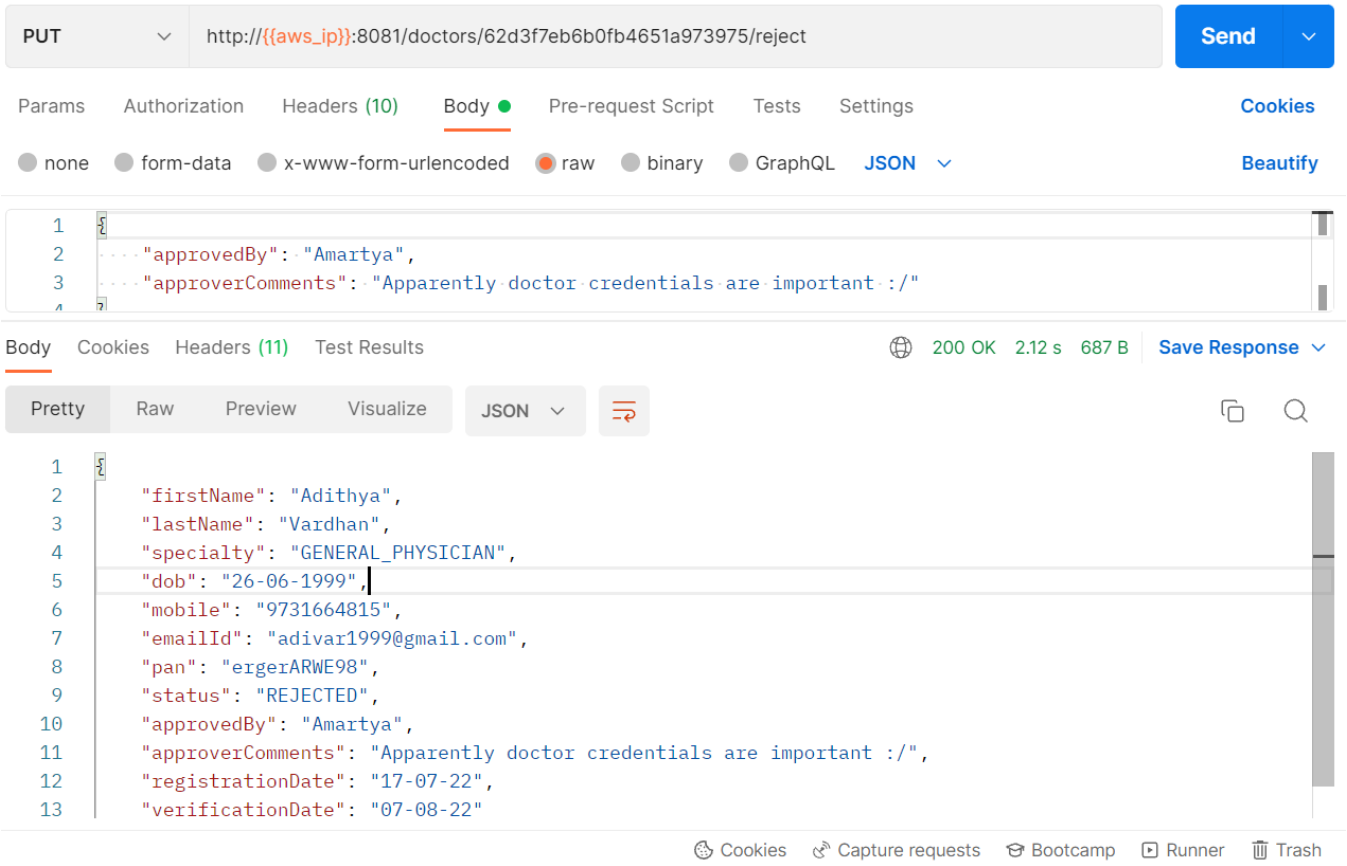


Fig 6.4b: Response to reject Doctor

5. Return a list of 20 Doctors sorted by rating

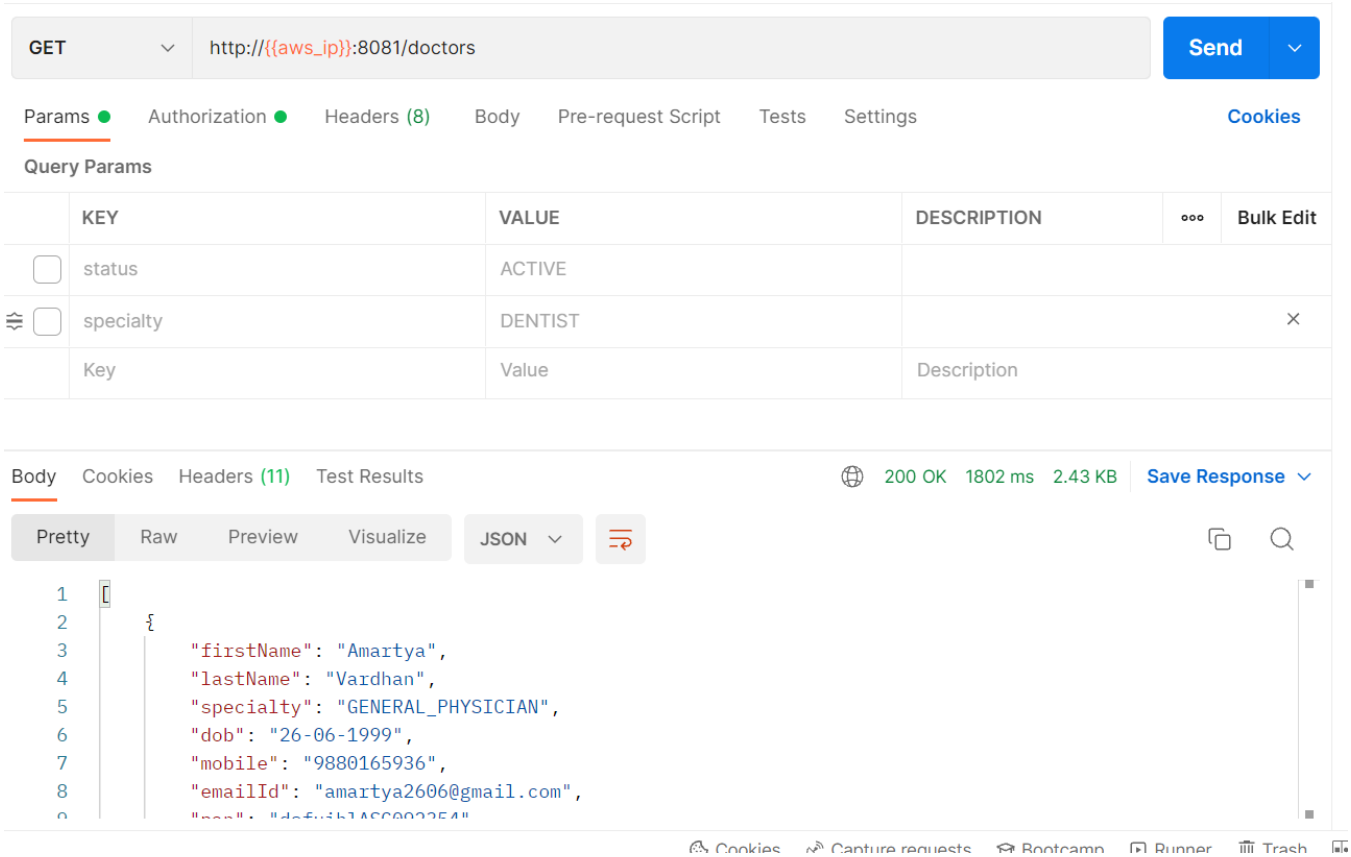


Fig 6.5a: Request and response for all doctors sorted by rating

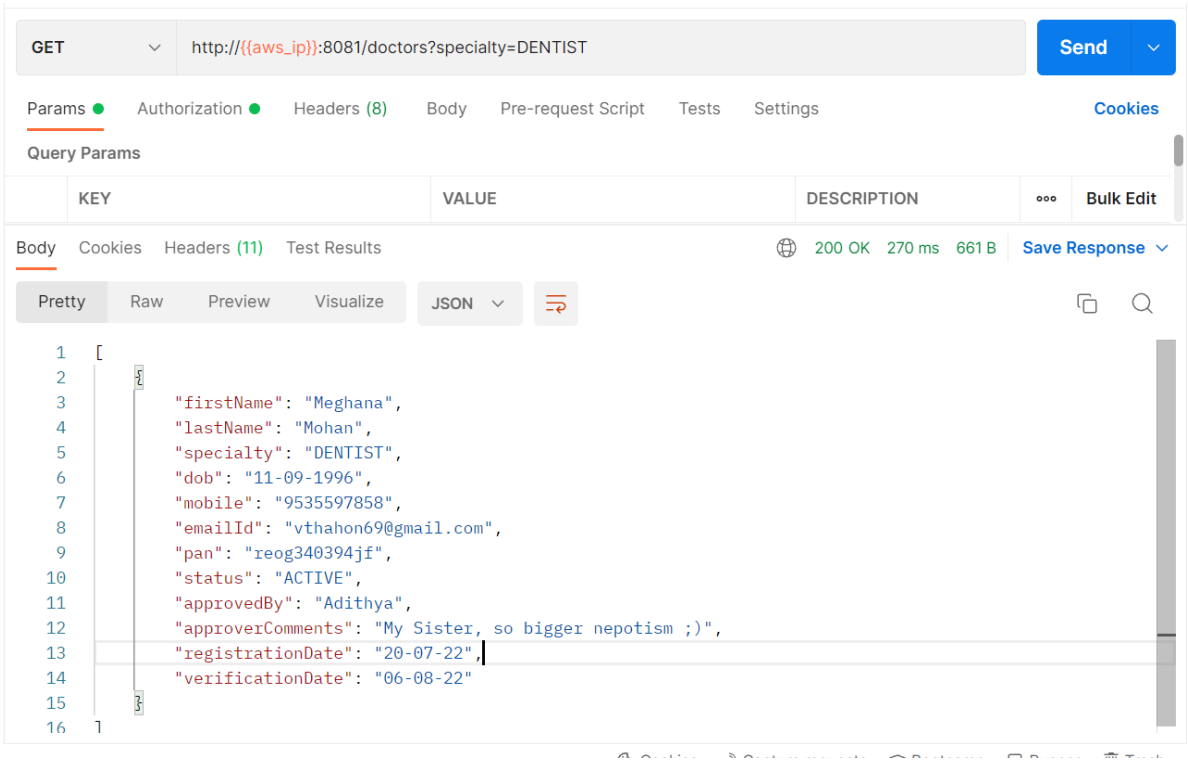


Fig 6.5b: Request and response for all doctors filtered by specialty

6. Return details of a Doctor by DoctorId

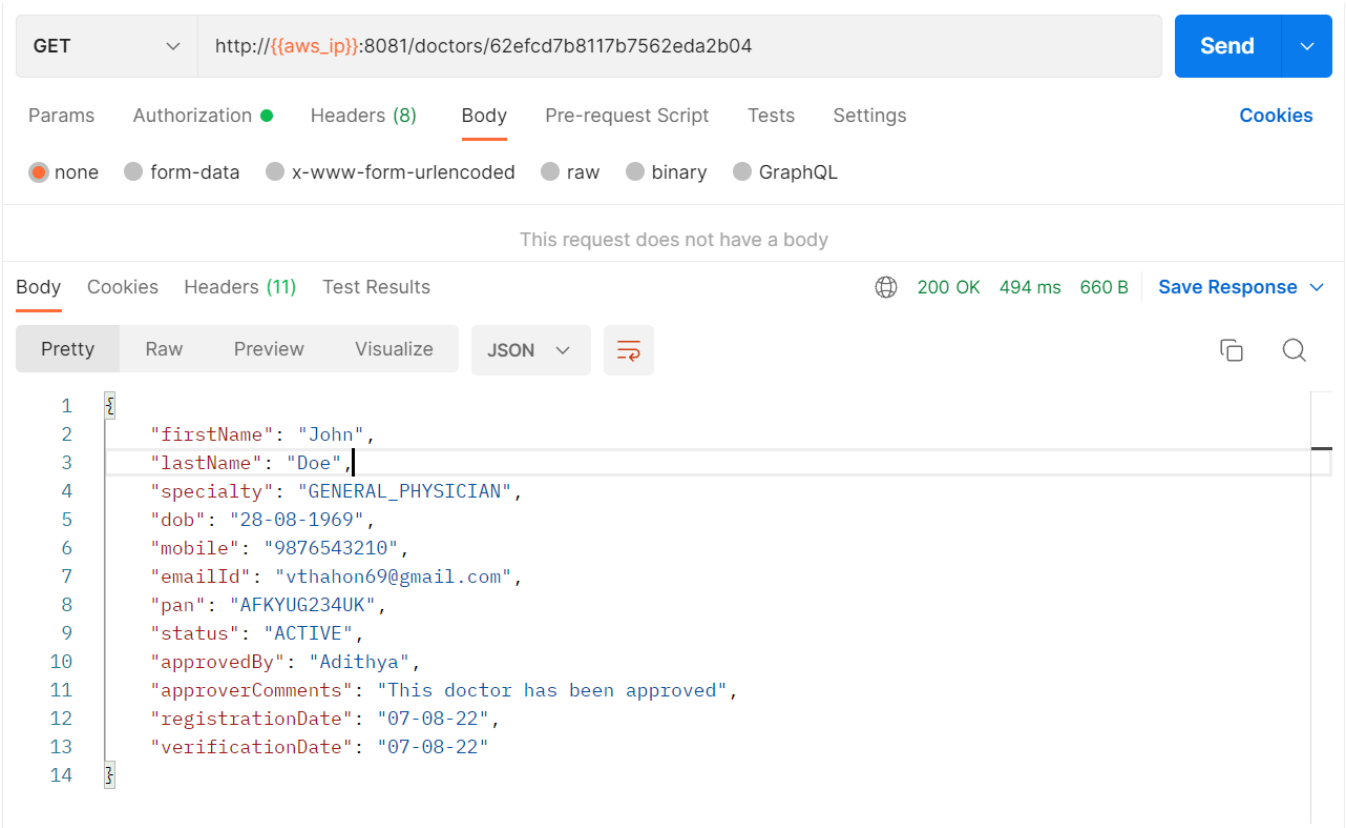


Fig 6.6: Request and response to get doctor details

UserService

1. Create User

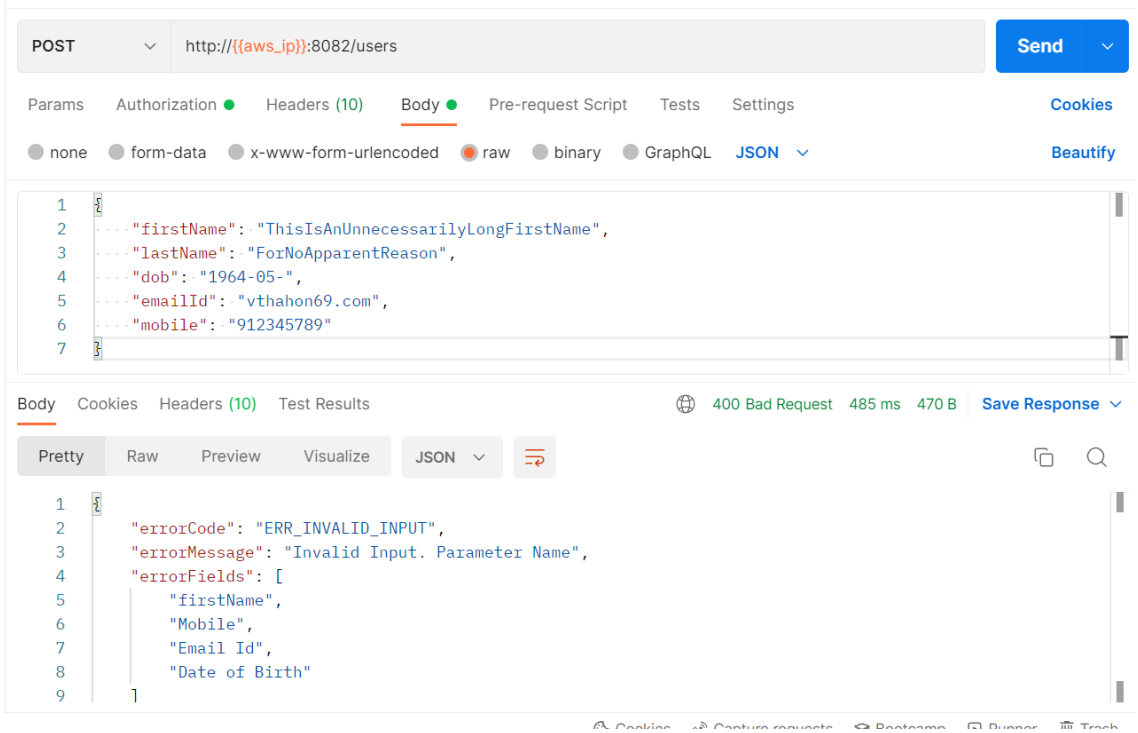


Fig 7.1a: Request and response for incorrect entries for Create User

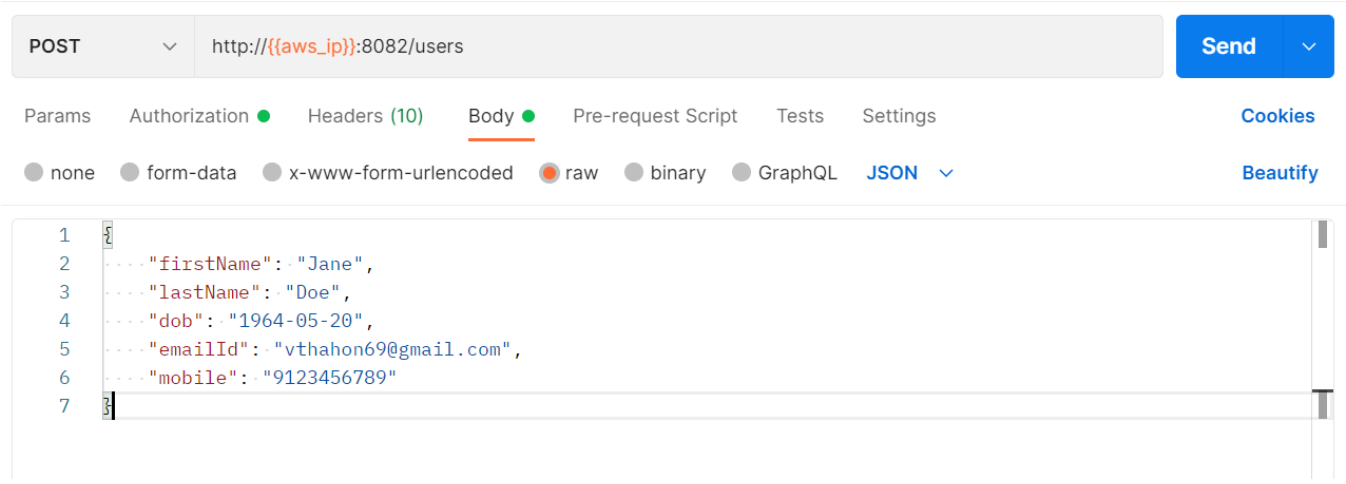


Fig 7.1b: Correct Request for Create User

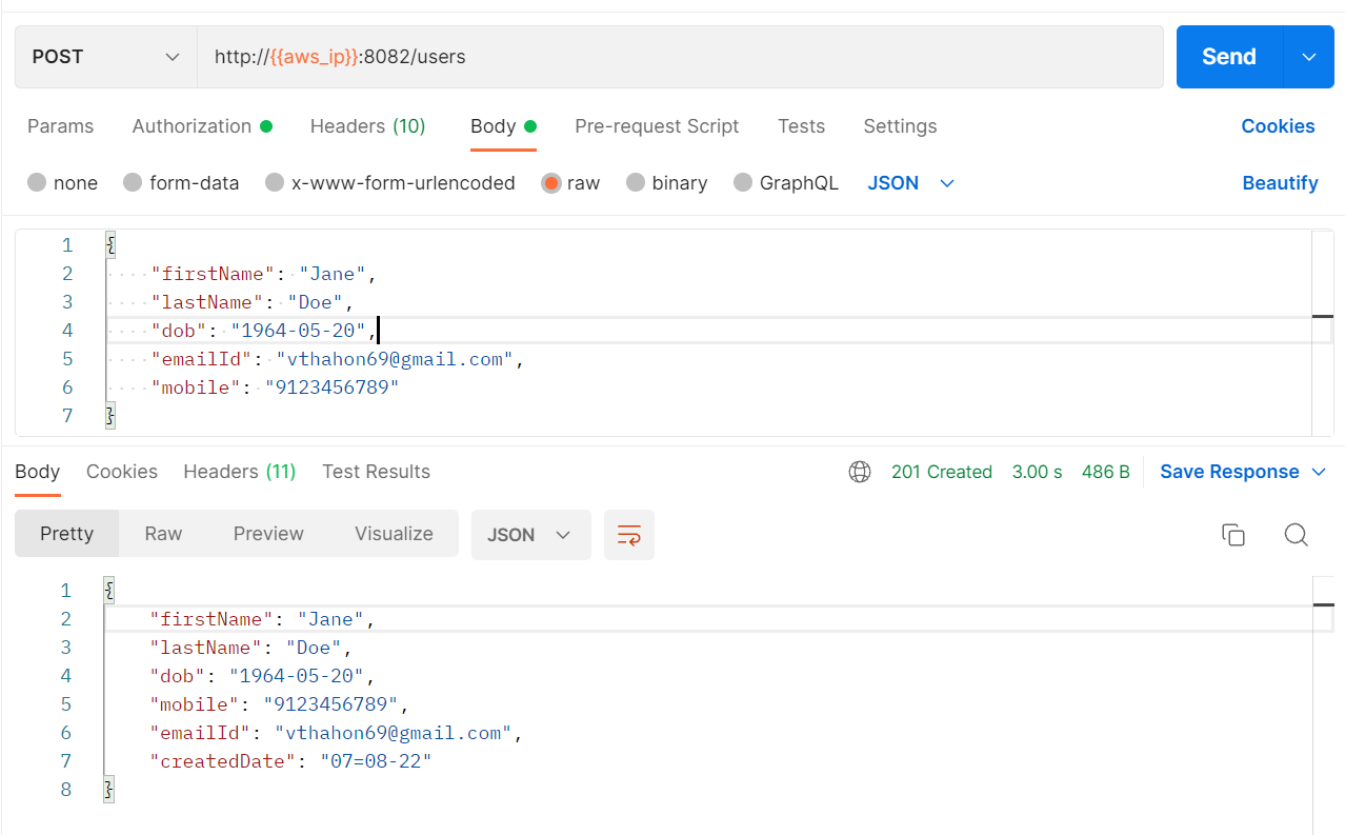


Fig 7.1c: Response for correct Create User

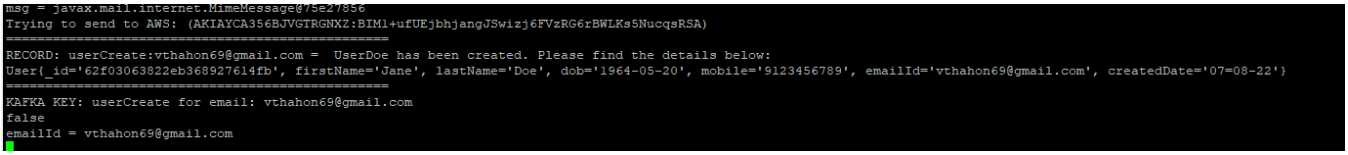
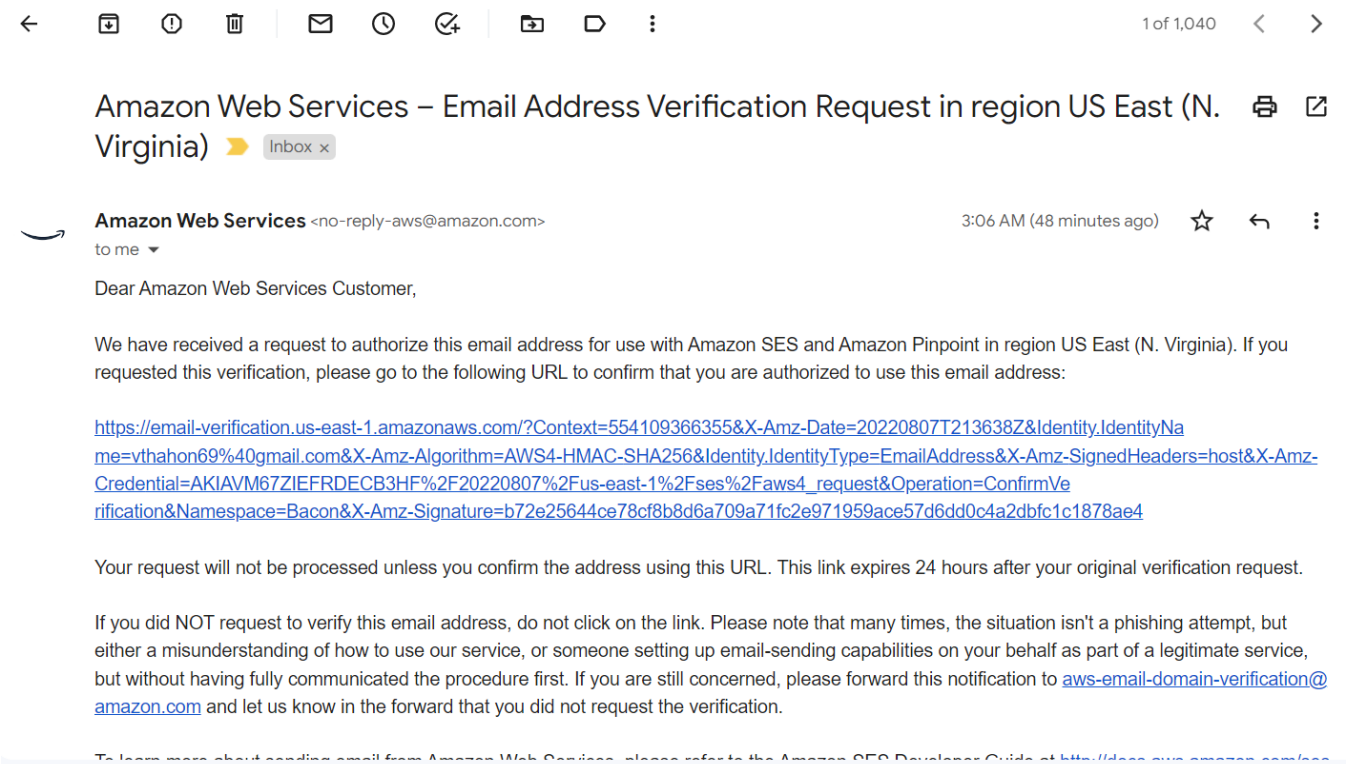


Fig7.1d: Notification in Kafka for creating user



Amazon Web Services – Email Address Verification Request in region US East (N. Virginia)

Inbox ×

Amazon Web Services <no-reply-aws@amazon.com> 3:06 AM (48 minutes ago)

to me

Dear Amazon Web Services Customer,

We have received a request to authorize this email address for use with Amazon SES and Amazon Pinpoint in region US East (N. Virginia). If you requested this verification, please go to the following URL to confirm that you are authorized to use this email address:

https://email-verification.us-east-1.amazonaws.com/?Context=554109366355&X-Amz-Date=20220807T213638Z&Identity.IdentityName=vthahon69%40gmail.com&X-Amz-Algorithm=AWS4-HMAC-SHA256&Identity.IdentityType=EmailAddress&X-Amz-SignedHeaders=host&X-Amz-Credential=AKIAVM67ZIEFRDECB3HF%2F20220807%2Fus-east-1%2Fses%2Faws4_request&Operation=ConfirmVerification&Namespace=Bacon&X-Amz-Signature=b72e25644ce78cf8b8d6a709a71fc2e971959ace57d6dd0c4a2dbfc1c1878ae4

Your request will not be processed unless you confirm the address using this URL. This link expires 24 hours after your original verification request.

If you did NOT request to verify this email address, do not click on the link. Please note that many times, the situation isn't a phishing attempt, but either a misunderstanding of how to use our service, or someone setting up email-sending capabilities on your behalf as part of a legitimate service, but without having fully communicated the procedure first. If you are still concerned, please forward this notification to aws-email-domain-verification@amazon.com and let us know in the forward that you did not request the verification.

Fig 7.1e: Verification Email for Create user

2. getUser

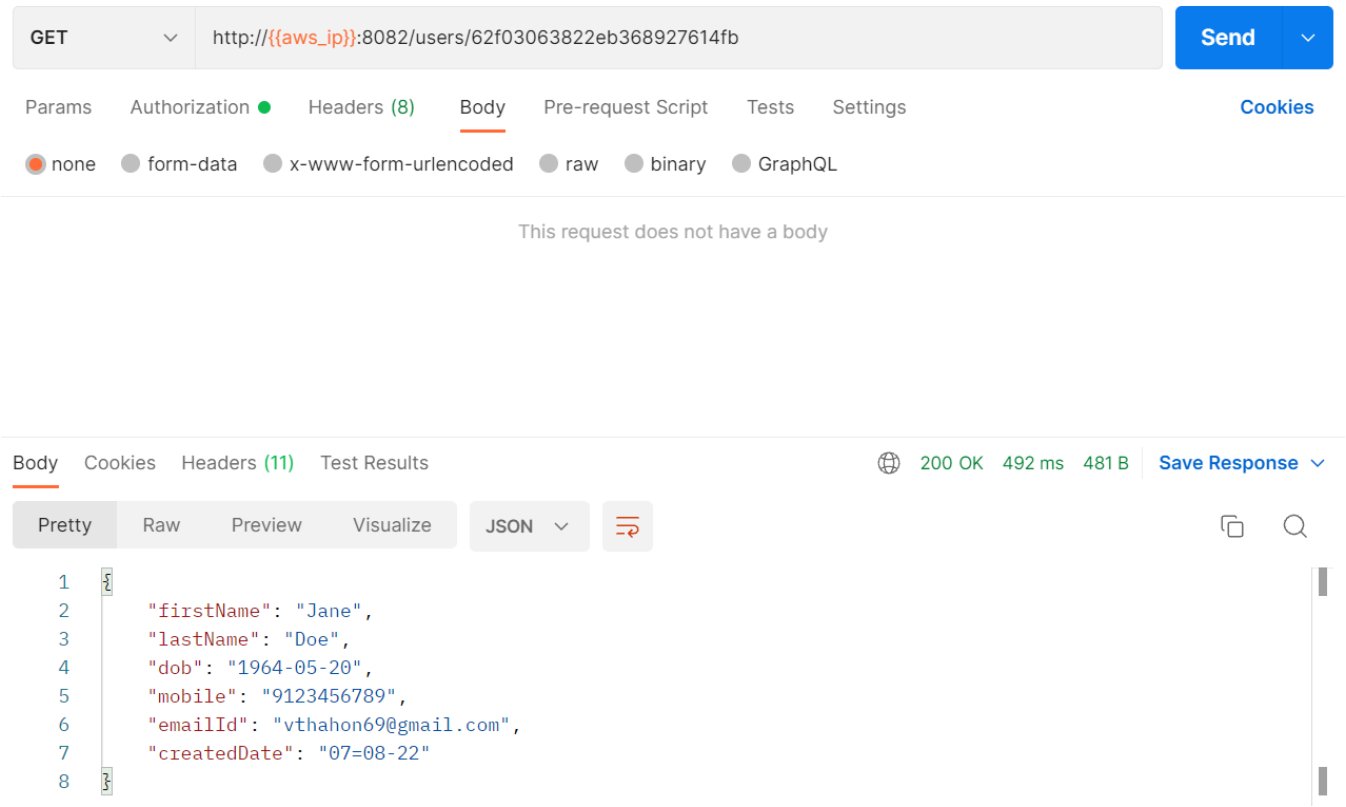


Fig 7.2: Request and response for getUser

3. Upload Documents for User

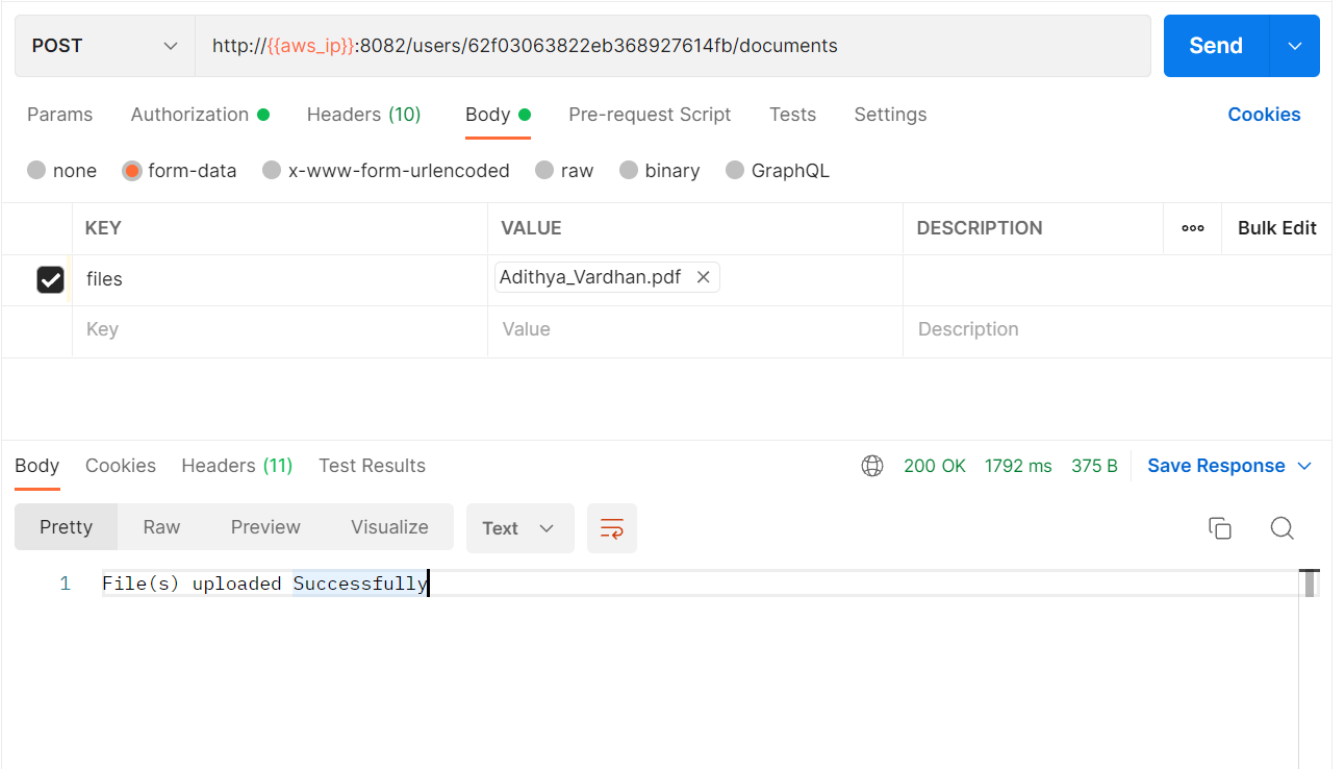


Fig 7.3a: Request and response for Upload Documents for User

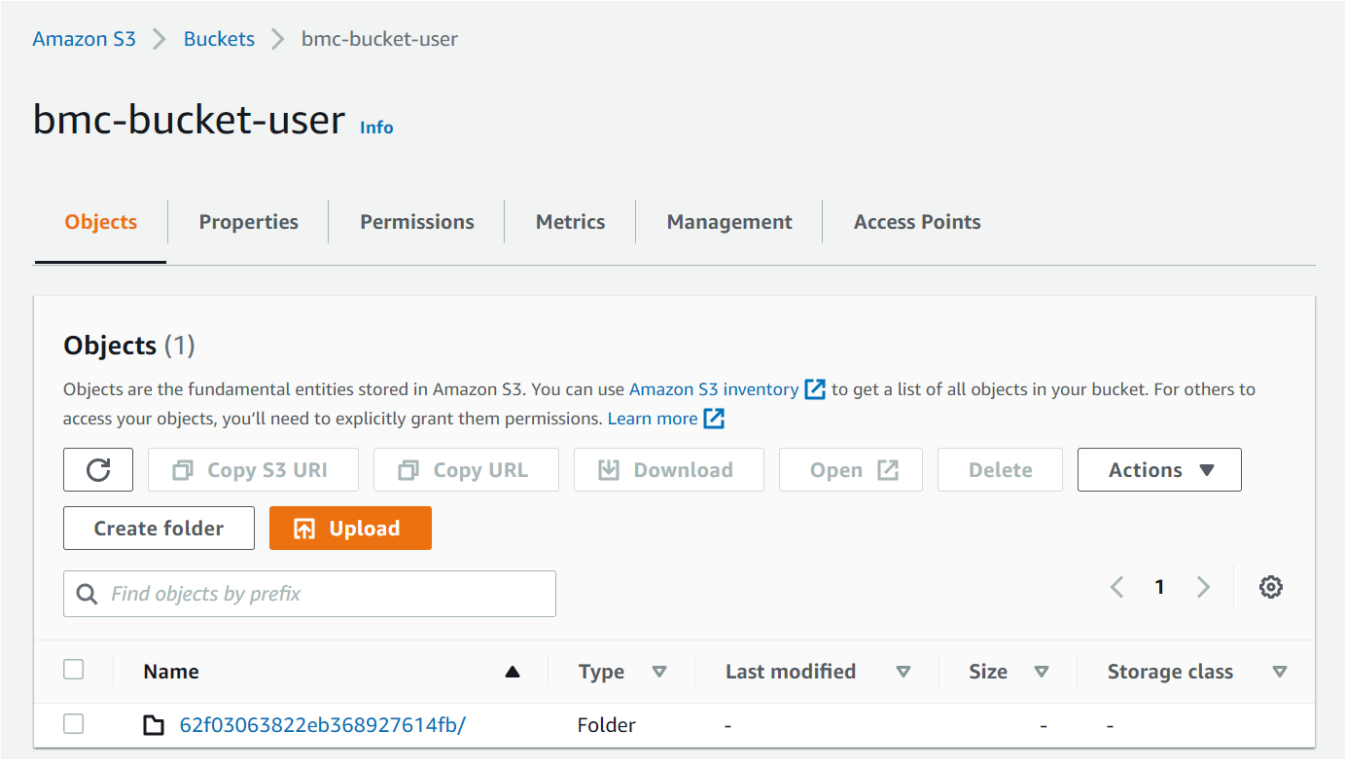


Fig 7.3b: File uploaded to Amazon S3

AppointmentService

1. createAvailability

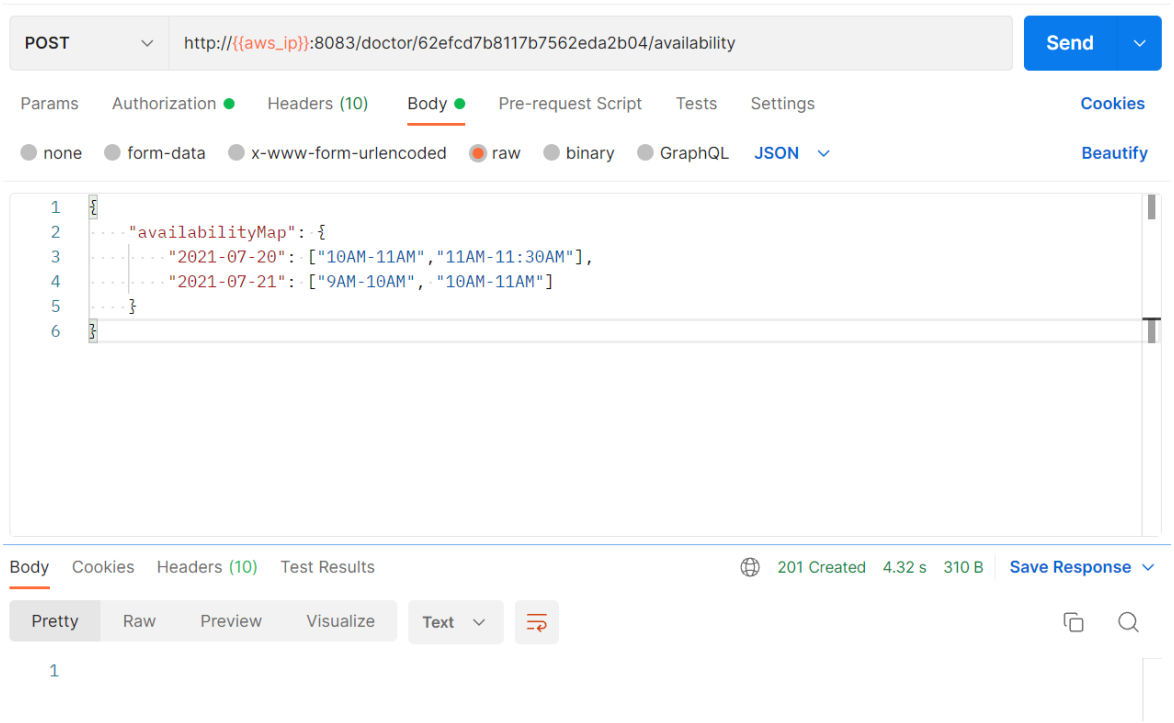


Fig 8.1: Request and response for setAvailability

2. getAvailability

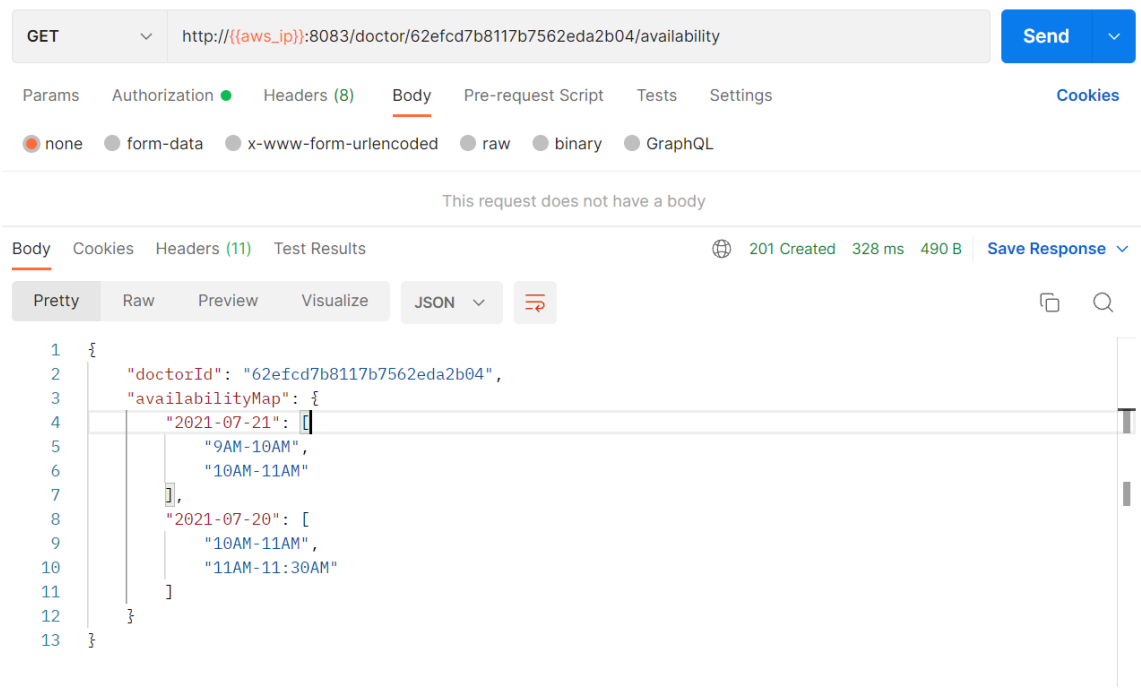


Fig 8.2: Request and response for getAvailability

3. createAppointment

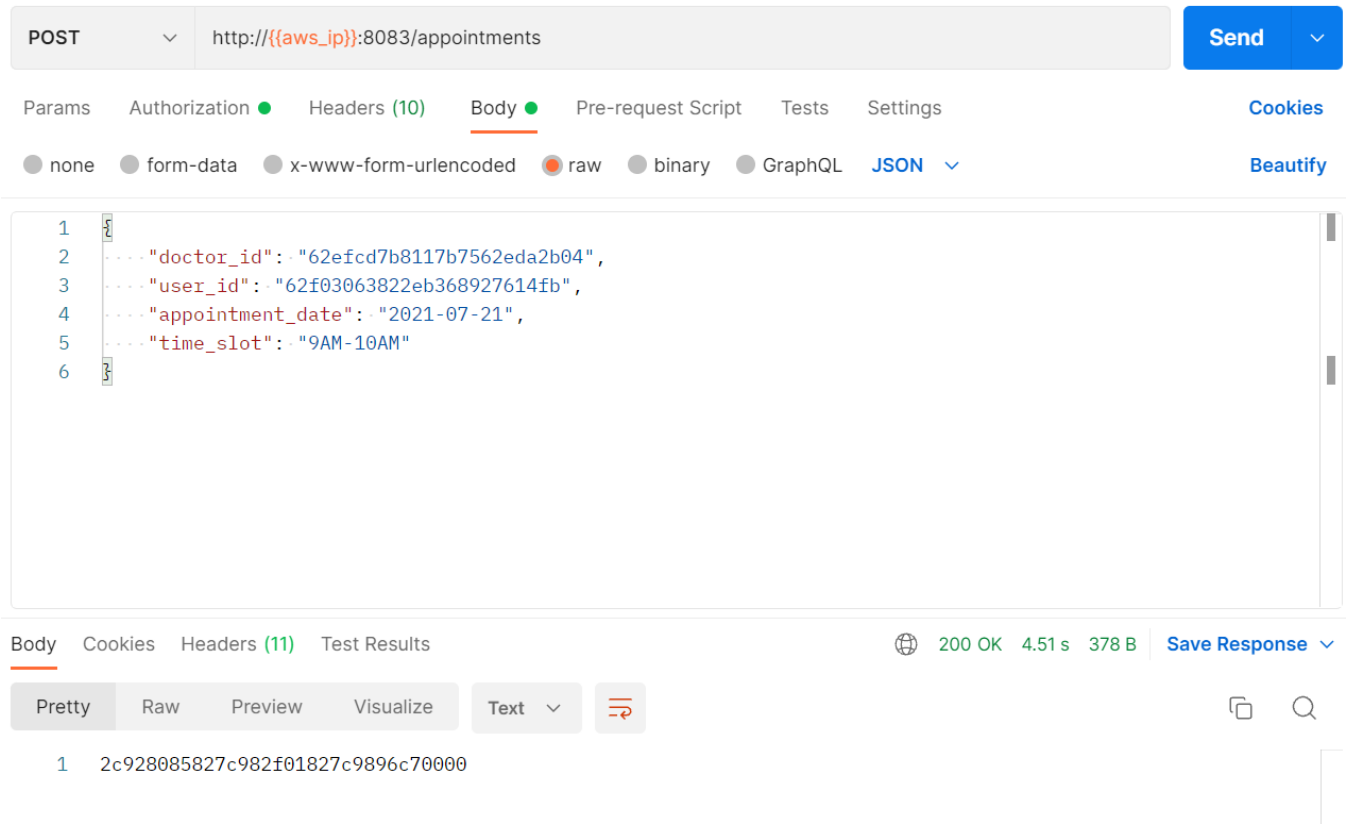


Fig 8.3a: Request and response for create Appointment

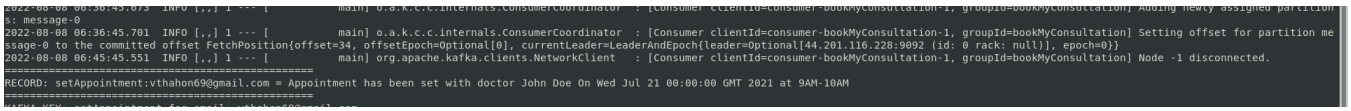


Fig 8.3b: Kafka Notification for create Appointment

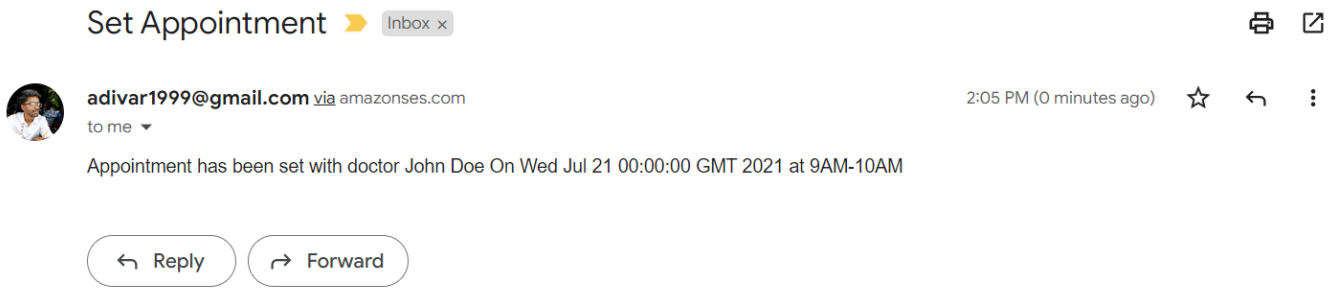


Fig 8.3c: Email confirmation for create Appointment

4. getAppointment

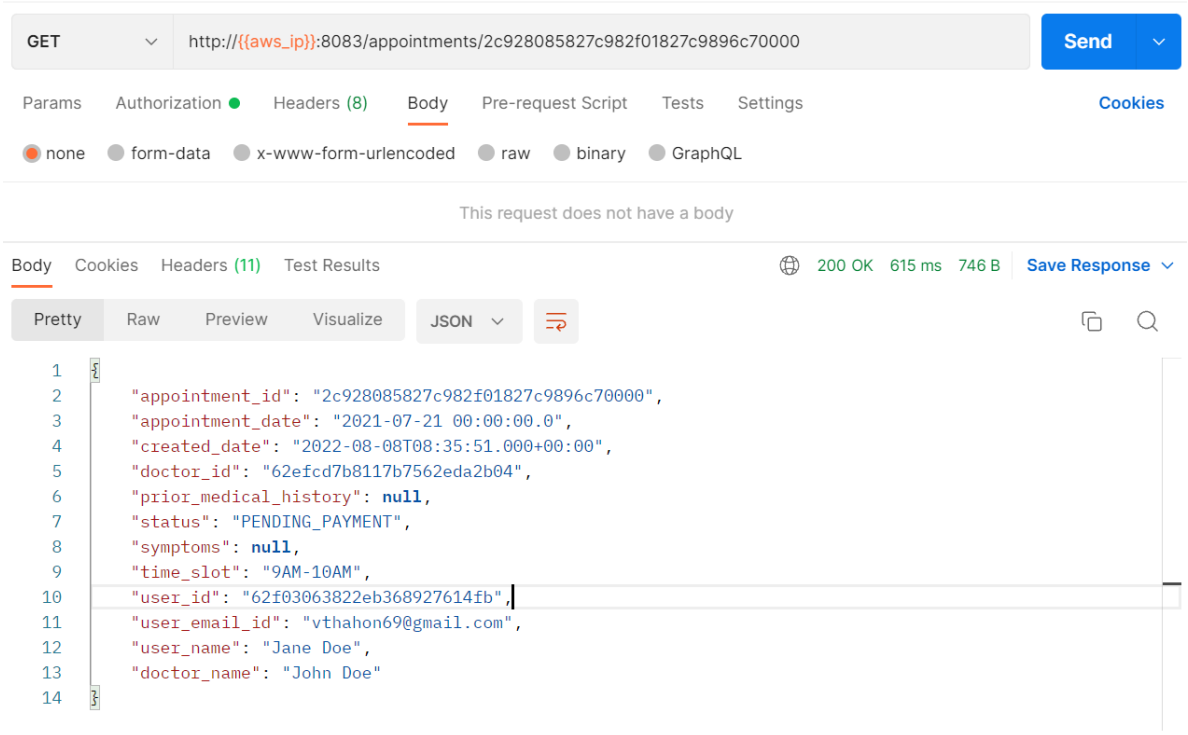


Fig 8.4: Request and response for get appointment

5. getAppointments

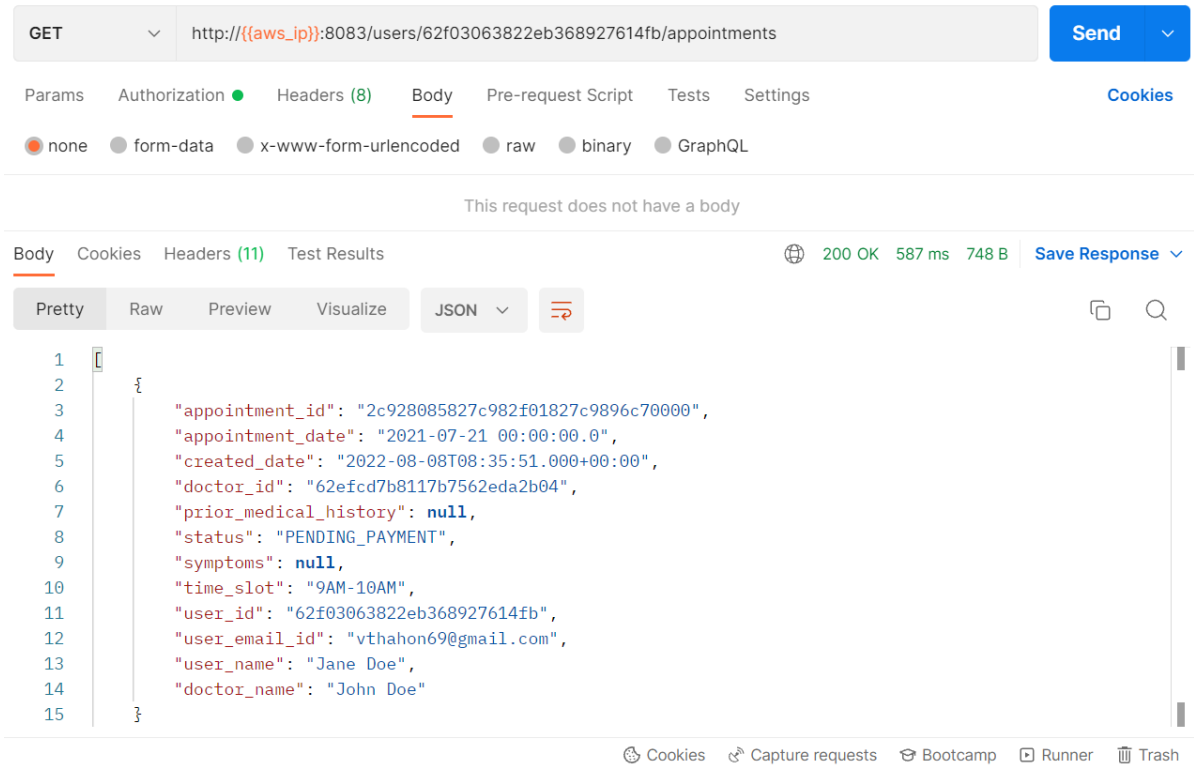


Fig 8.5: Request and response for get Appointments by user

6. createPrescription

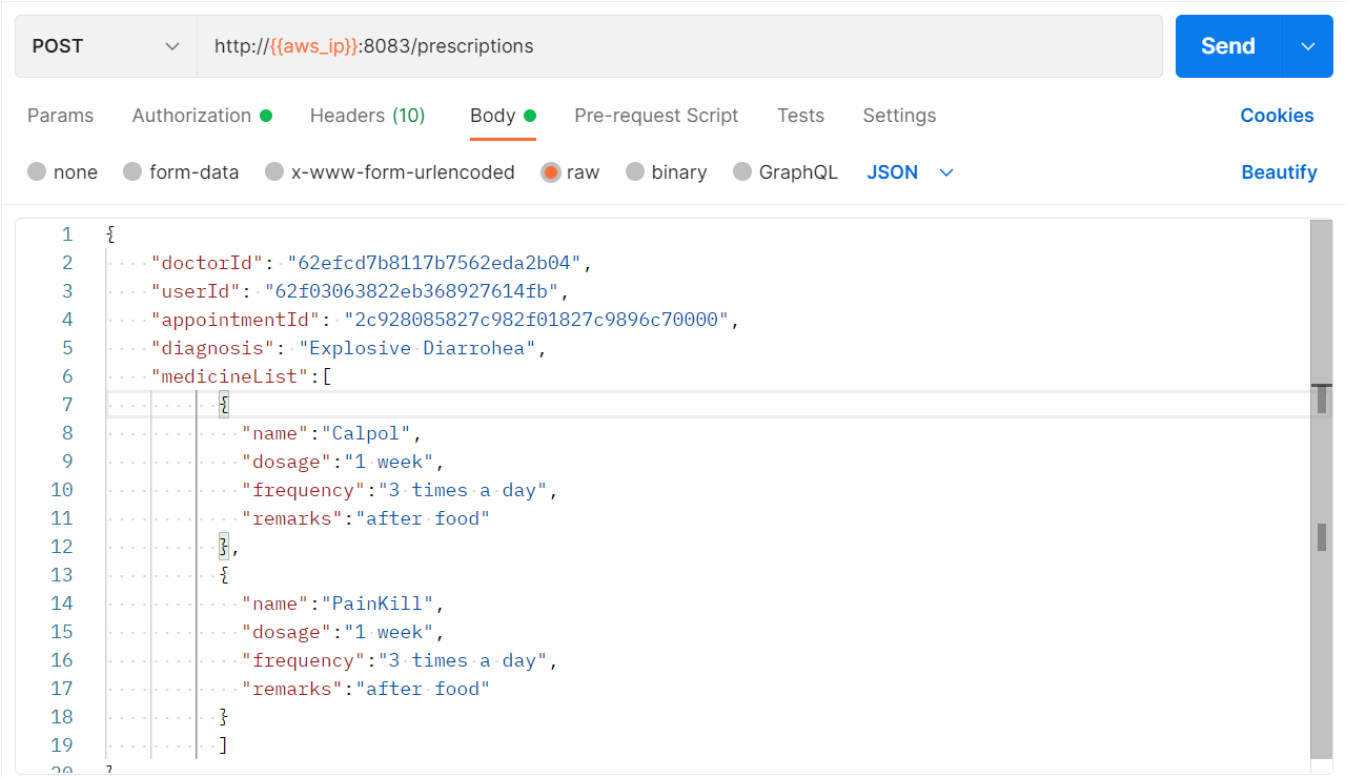


Fig 8.6a: Request for create Prescription

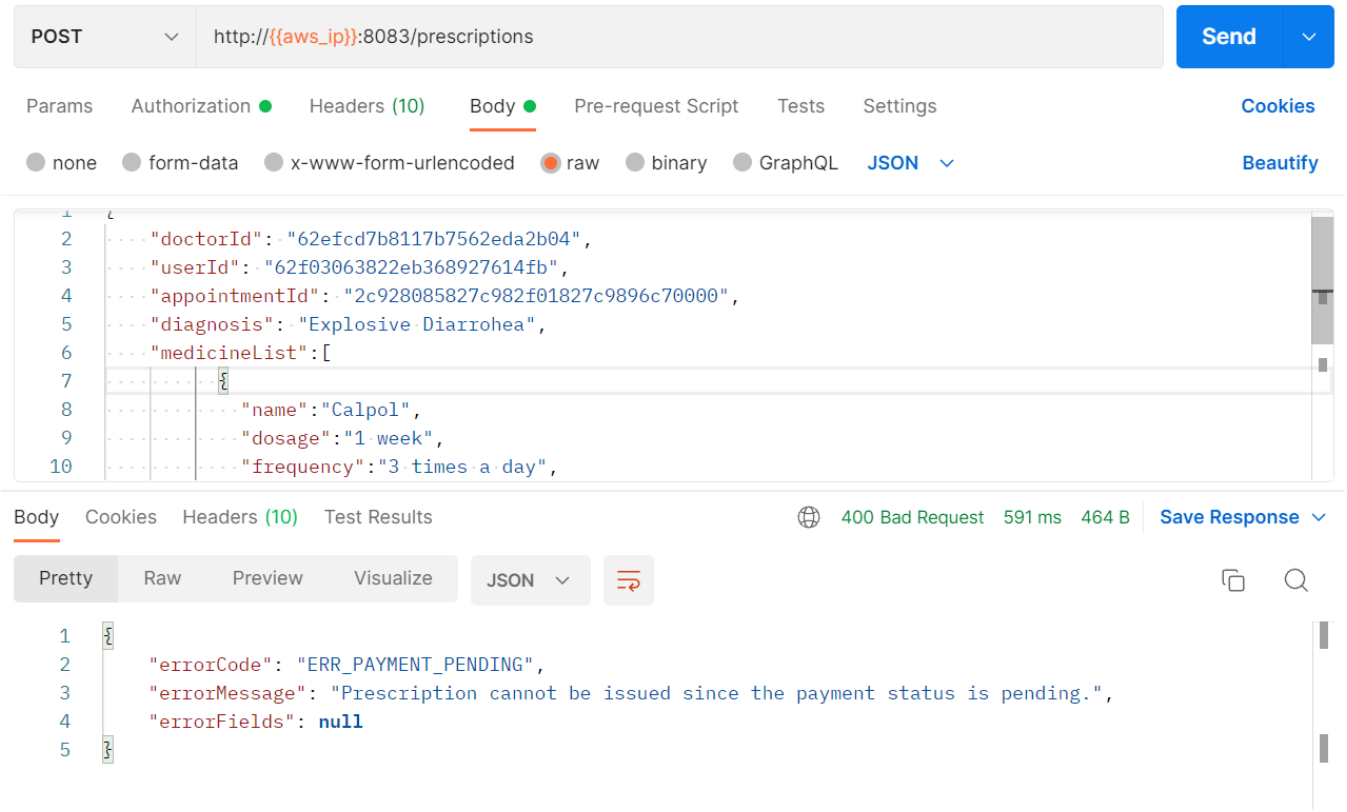


Fig 8.6b: Response when the Status of Appointment is PENDING

PaymentService

1. createPayment

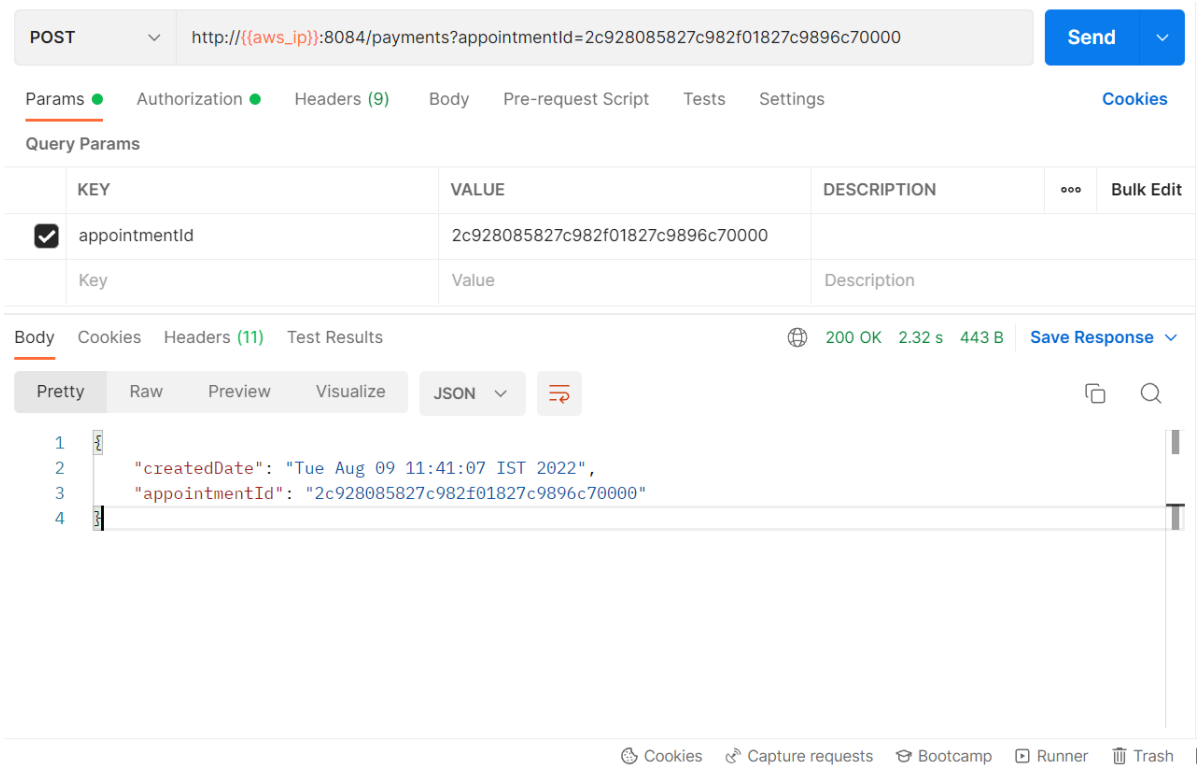


Fig 9.1: Request and response for create Payment

RatingService

1. createRating

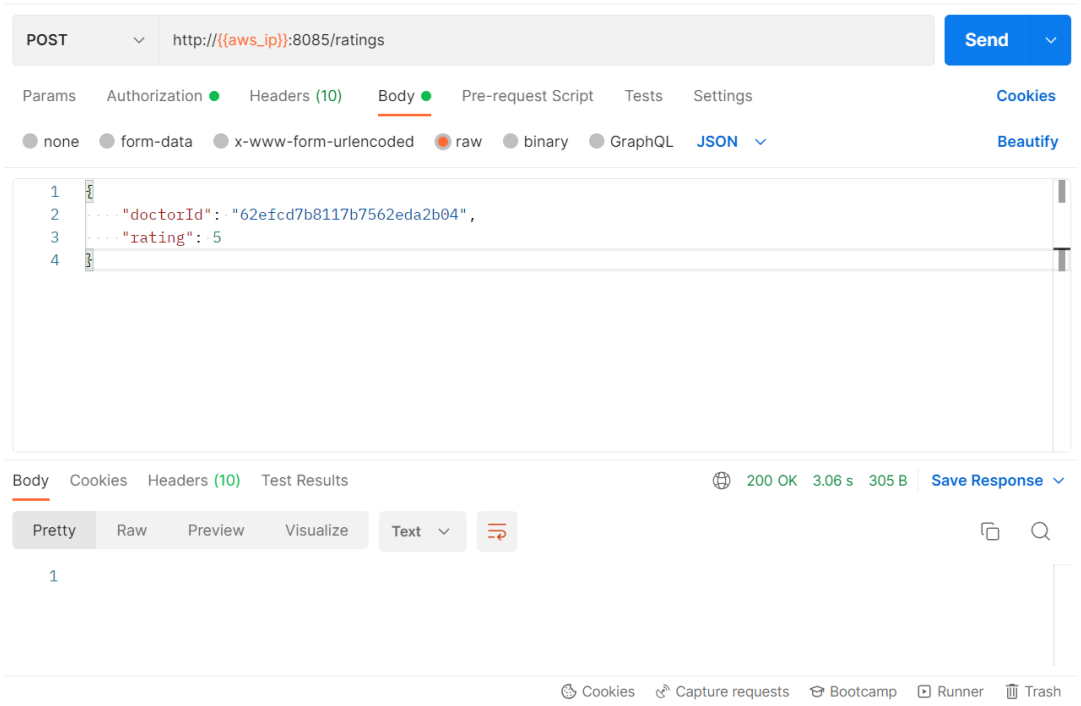


Fig 10.1: Request and response for creating Rating

NotificationService

The notification service uses Kafka to receive notifications sent from each service. For particular services, we receive confirmations to send emails using Amazon SES. The functionality has been implemented as shown in each of the required APIs.

The ***CreateDoctor*** and ***createUser*** API require SES to send a verification email to the respective email ID to confirm that the email exists.

The ***approveDoctor***, ***rejectDoctor***, ***setAppointment*** and ***setPrescription*** APIs request SES to send custom messages to the respective email ID of the Doctor or the User.

Other services send kafka notifications to print by themselves, as shown in each API

Authentication

Authentication in the form of JWT token validation has been implemented in each of the services that contain API endpoints. To generate the JWT tokens, I used a separate project developed during the Security module.

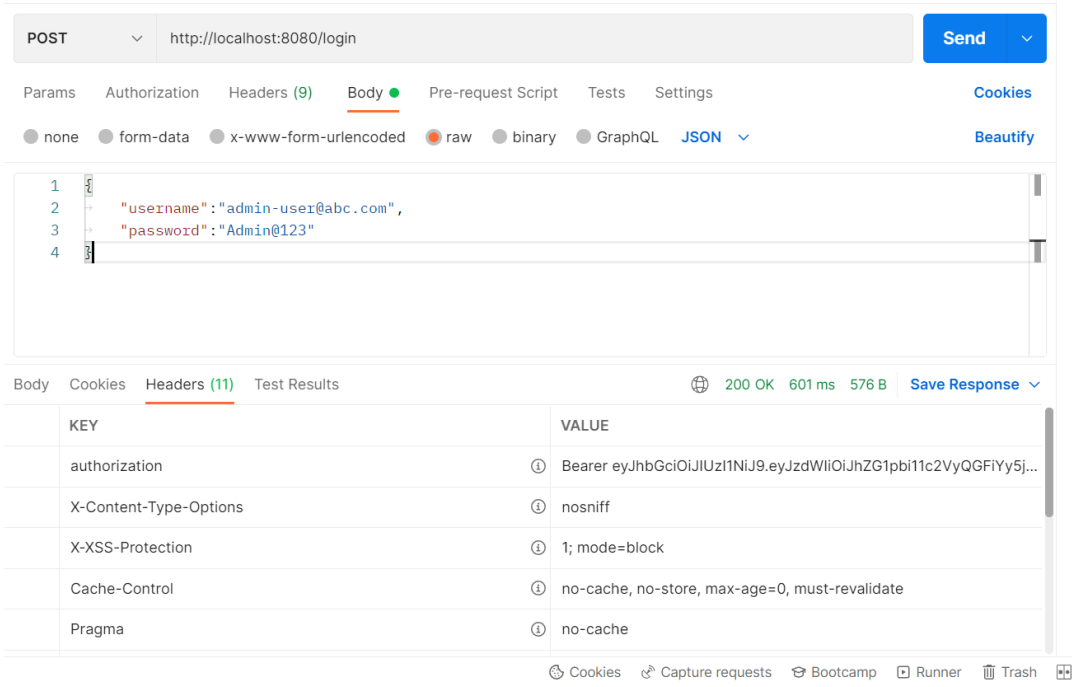


Fig 12.1: Request and response for Admin JWT token generation

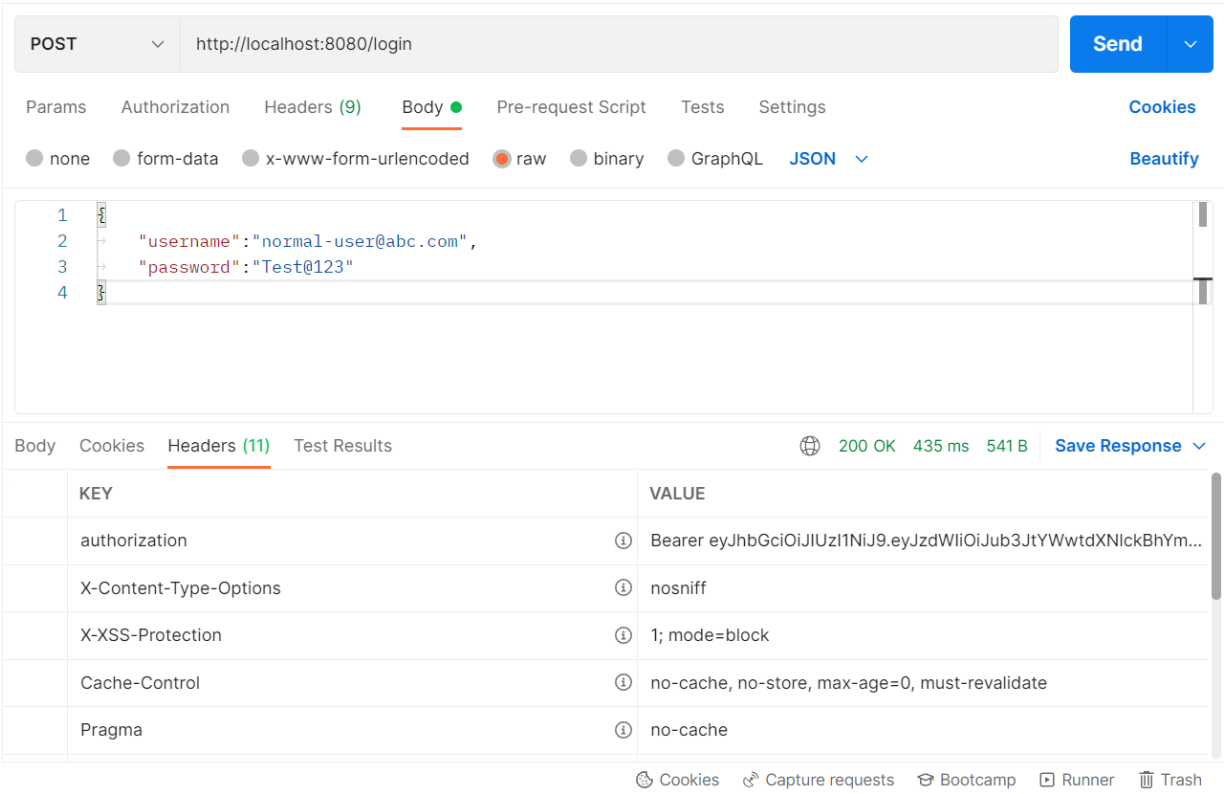


Fig 12.2: Request and response for User JWT token generation

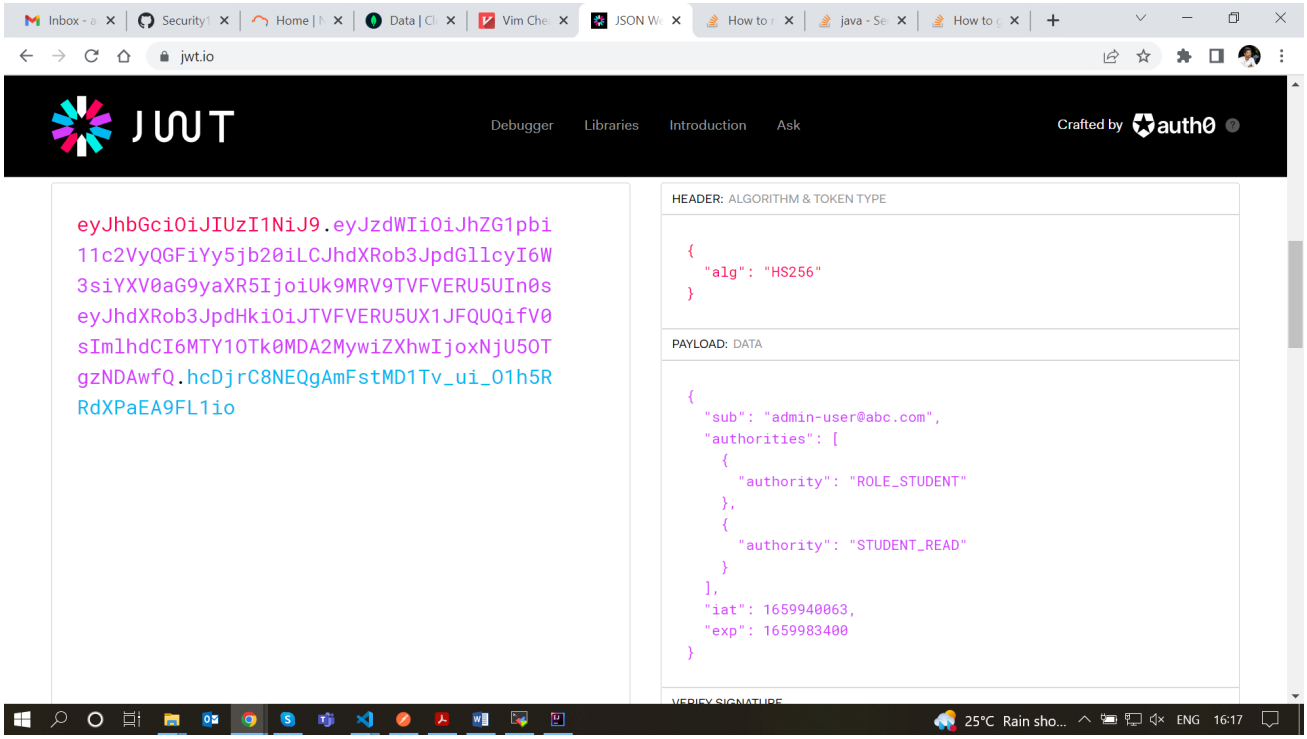


Fig 12.3: JWT token decrypted

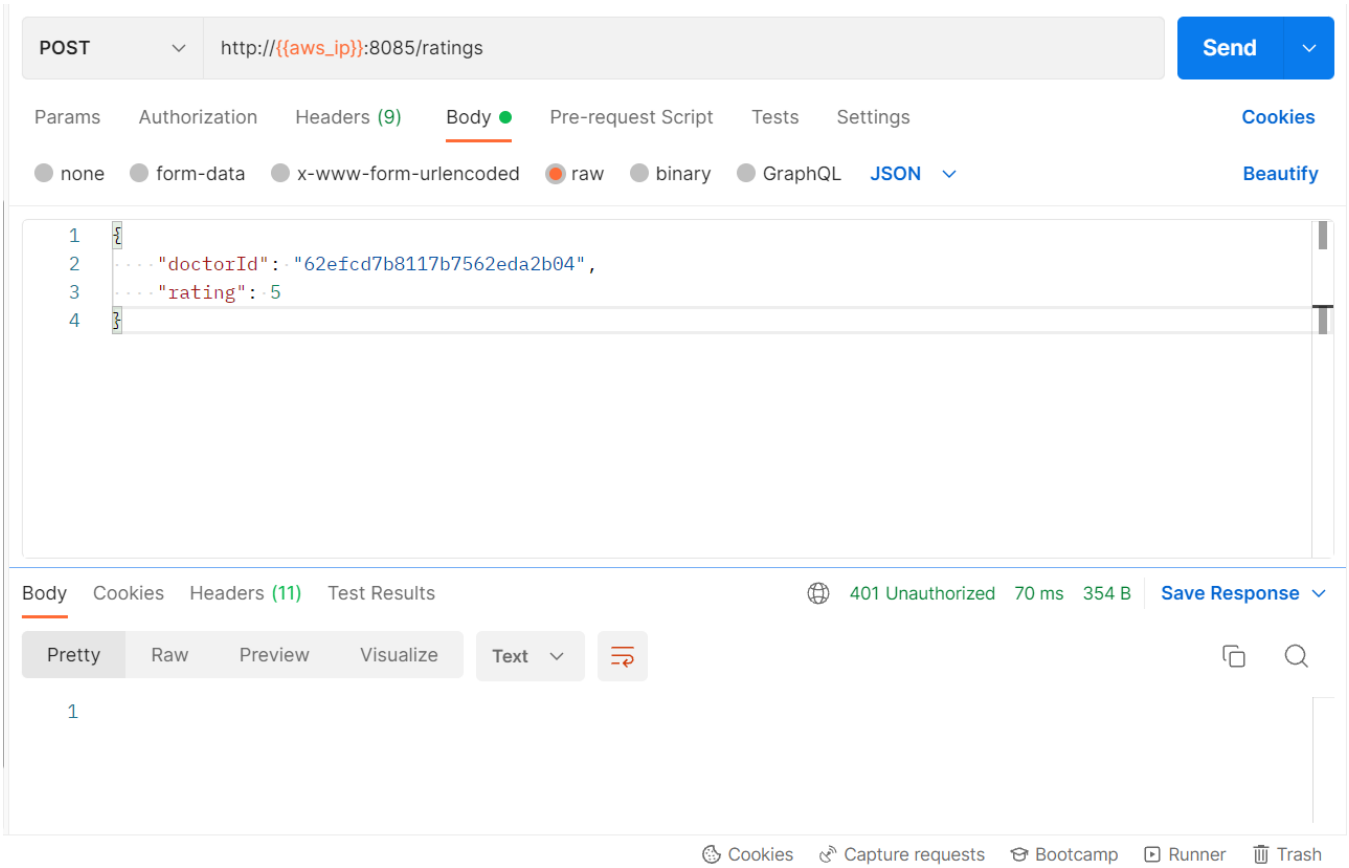


Fig 12.4: Response of an API without authorization

Closing the Project

At the end, once we have implemented all the use cases of the application, we need to close all parts of the project.

We start by stopping all running instances of the various services running in our project, by the command

Fig 9.1: Stop running instances

Then we close and terminate the RDS instance that we have running on the AWS cloud as well the EC2 instance running the services and the Kafka server.