BookMyConsultation
Upgrad Capstone Project

Initial Analysis:

Understood the overall design and architecture of book my consultation application and the services involved by going through the videos and documents provided.
I have made few changes in terms of response fields and values for few requests so that its more meaningful and also to handle the subsequent tasks easily.

The BookMyConsultation folder will contain the following services and files apart from this documentation file:

- doctor-service
- appointment-service
- user-service
- rating-service
- notification-service
- payment-service
- bmc-gateway
- docker-compose.yml

Note: I have decided to use externalized Kafka and MongoDB by setting it up on ec2 instance and created an RDS instance for MySQL DB in this capstone project.

I have also not used eureka-server for this project as the reason being that when we deploy these services on the docker and then eventually on k8s, they have their own DNS service discovery available as service component which will render eureka useless and additional overhead.

Hence, I have only used gateway service here to route requests to different services.

The BookMyConsultation application has many services and the logic and configuration changes for each service will be explained individually below and the Kafka, MongoDB, MySQL and ubuntu instance setup's will be available at the end of this documentation.

General Components (All Services):

I have implemented two components which are same across all services important for monitoring and securing all of them. They are **Aspect Oriented Logging** and **Security**.

Aspect Oriented Logging: I have added an spring aop starter and implemented Aspect Logging in all services to reduce the manual effort of logging in every class unless required, which combined with spring cloud sleuth helps in tracing.
General Log Format used for every service method:
While Entering: *log*.info("Entering CLASS_NAME:{} METHOD_NAME:{} with input parameters {}",className, methodName, inputArgs);
While Exiting: *log*.info("Exiting CLASS_NAME:{} METHOD_NAME:{} with return value [{}] and execution time of {}",className,methodName,
result,getTimeInMinutesAndSeconds(stopWatch.getTotalTimeMillis()));
This can be seen in logs of all services for any request made.

Security: I have used spring security to secure all the endpoints for all services and hence before accessing any service endpoint you need to get an auth jwt token from any service, and then this same token can be used for all requests to all services for a day as I have set the expiration date for the tokens post 24hours from creation for any user or admin.

For the purpose of this project, I have created two admin user's and 4 normal users (patient/doctor) as below:

You can get the user or admin token for hitting the individual endpoints based on roles allowed on the endpoint from below uri:
{{environment}}{{doctor-service-context-path}}/login
localhost:9191/doctor-service/login
Here environment refers to the host where your running, if its local you hit the gateway on localhost:9191 or if its on ec2 then ec2-url:9191, and for doctor-service-context-path, this needs to be added to every doctor-service uri so that bmc-gateway can route the requests to services correctly.

Note: I have added a /verify-email under doctor-service so that an admin can have his email verified and use that same email in notification-service for default from-email for all types of events.

BMC Gateway Logic:

Eg: In case of doctor-service endpoint you need to hit the endpoint on gateway with context path doctor-service before the actual doctor-service uri:
GET: localhost:9191/doctor-service/doctors/{doctorId}
Note: bmc-gateway has a filter in application.yml to strip the context path before routing the requests, that way I didn't have to add a context path to the individual services for the sole purpose of routing.

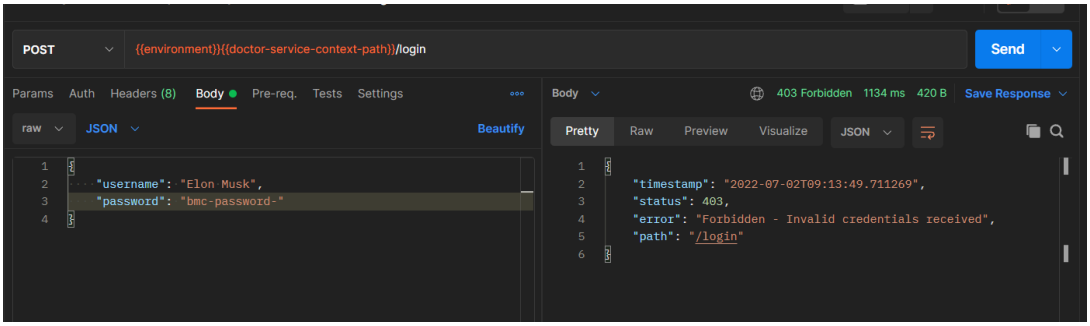
Context-paths for all services:
doctor-service: /doctor-service
appointment-service: /appointment-service
user-service: /user-service
rating-service: /rating-service
payment-service: /payment-service

Doctor Service (doctor-service):

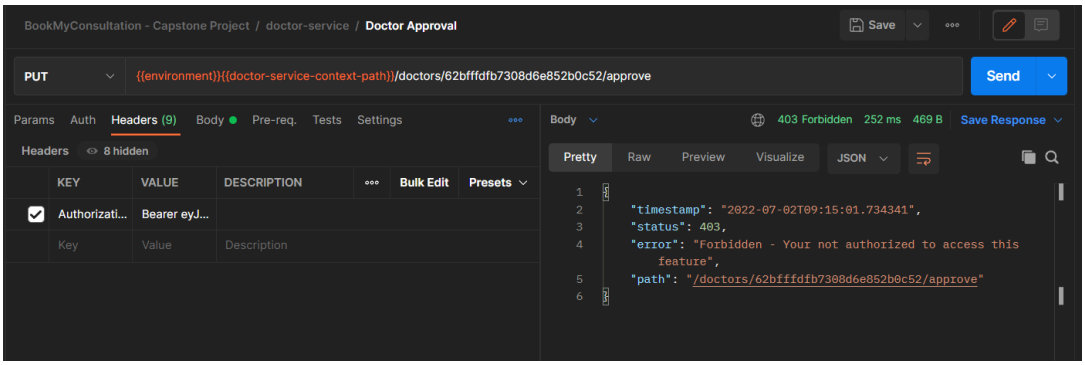
Logic and Design: In doctor-service we are storing the details of doctor in MongoDB and sending an email verification mail to doctor email Id provided using Amazon SES. We are also sending a message to notification-service for all doctor related events.

For doctor sorted list logic I have used required tag as false for speciality param so that we can either pass status, or both status and speciality and internally used a dynamic query with mongo template to get the result.

Changes on Responses: For the error responses I have attached the default messages along with the error fields so that user can understand the right format to send details. In case of forbidden errors, I have modified the response message to alert if it's an authorization issue or invalid credentials issue so that we can understand the issue. Please find the responses below:



Invalid password response



Using USER token for approve request

Once the tokens are received, they can be used for a day as its expiration is for current day + 1. The same token can be used for all service requests all of the logic and key to decode the token and allow access.

One important logic for rating I have implemented is that the rating service will calculate the average doctor rating and doctor-service will only receive the average rating from rating-service and update in its table, this way once its updated users can

see the doctors rating and also the sorting becomes easier at doctor side because of which I have added average rating field in doctor schema.
I used Json non-null property to not send any field which are null which handles the null being shown until the rating is received.

All other request responses are same as mentioned in the postman collection and service designs.

Configuration Change:
For the purpose of simplicity, I have added a comment Configure before run comment on every line where changes are to be made before run based on your setup.
You can find the screenshots for doctor-service config changes below:

```yaml
  data:
    mongodb:
      host: ec2-3-232-57-4.compute-1.amazonaws.com # Configure before run
      port: 27017
      database: doctor-service

server:
  port: 8081

kafka:
  producer:
    bootstrap-servers: ec2-44-207-79-237.compute-1.amazonaws.com:9092 # Configure before run
    topic: bmc-notification
  consumer:
    bootstrap-servers: ec2-44-207-79-237.compute-1.amazonaws.com:9092 # Configure before run
    topic: bmc-doctor
    group-id: doctor-service
```

application.yml

```java
public class S3Repository {
    private AmazonS3 s3Client;
    private String BUCKET_NAME = "ruhan.bmc.doctor.documents"; // Configure before run
    // This needs to be a unique bucket name across all the regions.

    ObjectMetadata metadata;

    @PostConstruct
    public void init(){
        // Update the amazon s3 admin access key and secret key here once generated for your account
        String accessKey = ""; // Configure before run
        String secretKey = ""; // Configure before run
        AWSCredentials credentials = new BasicAWSCredentials(accessKey,secretKey);
        s3Client = AmazonS3ClientBuilder
                .standard()
                .withCredentials(new AWSStaticCredentialsProvider(credentials))
                .withRegion(Regions.US_EAST_1)
                .build();
    }
```

S3Repository

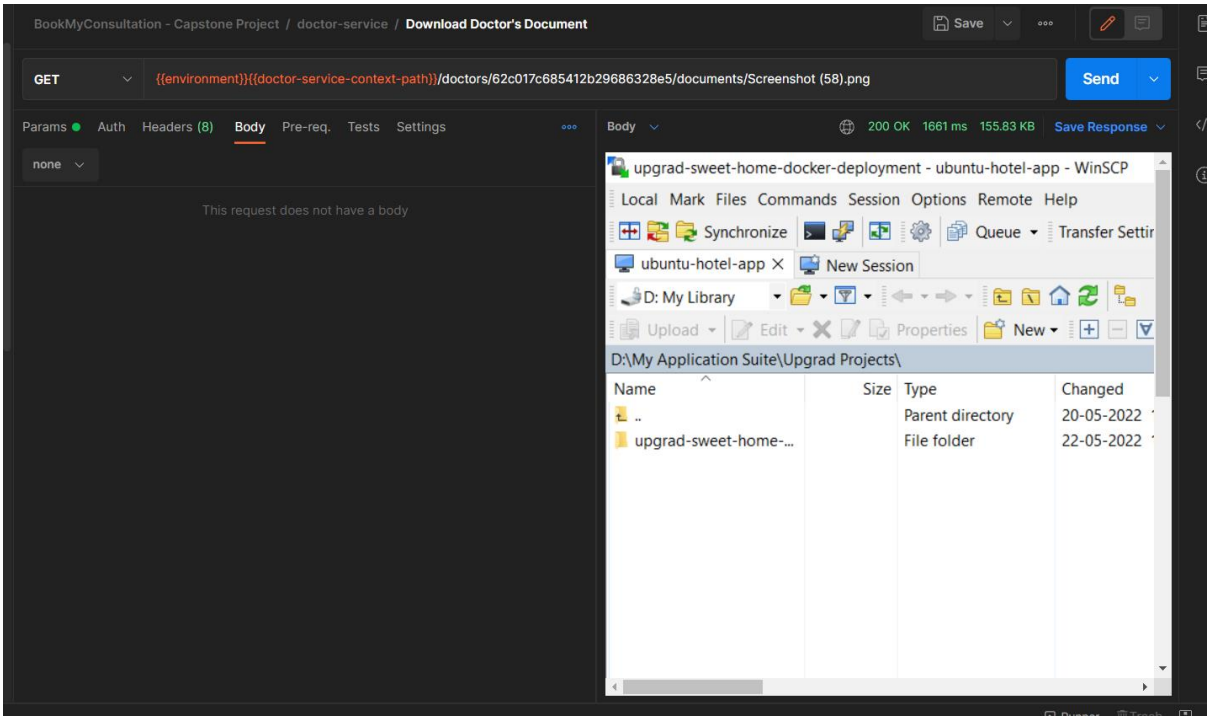
```java
    @PostConstruct
    public void init(){
        // When you hit the endpoint to verify the email this needs to be the ses access key for your AWS account
        String accessKey = ""; // Configure before run
        // When you hit the endpoint to verify the email this needs to be the ses secret key for your AWS account
        String secretKey = ""; // Configure before run

        //This is required for email verification
        StaticCredentialsProvider staticCredentialsProvider = StaticCredentialsProvider
                .create(AwsBasicCredentials.create(accessKey,secretKey));
        sesClient = SesClient.builder()
                .credentialsProvider(staticCredentialsProvider)
                .region(Region.US_EAST_1)
                .build();
    }
```

EmailService

I have also implemented the optional endpoints for getting the list of documents uploaded and also downloading a document based on its full name including the extension. I referred the online docs from amazon to understand how to handle such requests. Please enter the full name with extension (.png/.pdf/.txt) while retrieving the document.

Doctor service listens to bmc-doctor topic to update the rating of doctor received from rating-service.



For example: Here I have uploaded a screenshot of my previous assignment for demo.

User Service (user-service):

- **Logic and Design:** In case of user-service most of the logic and response are same as in doctor-service. It has all the operations such as register, upload, download and get individual user details.

  I have used comments wherever there is complex logic but usage of understandable variable names will make the steps easier to understand. A kafka producer also is used to send a user registration event to notification-service.

- **Configuration Changes:**

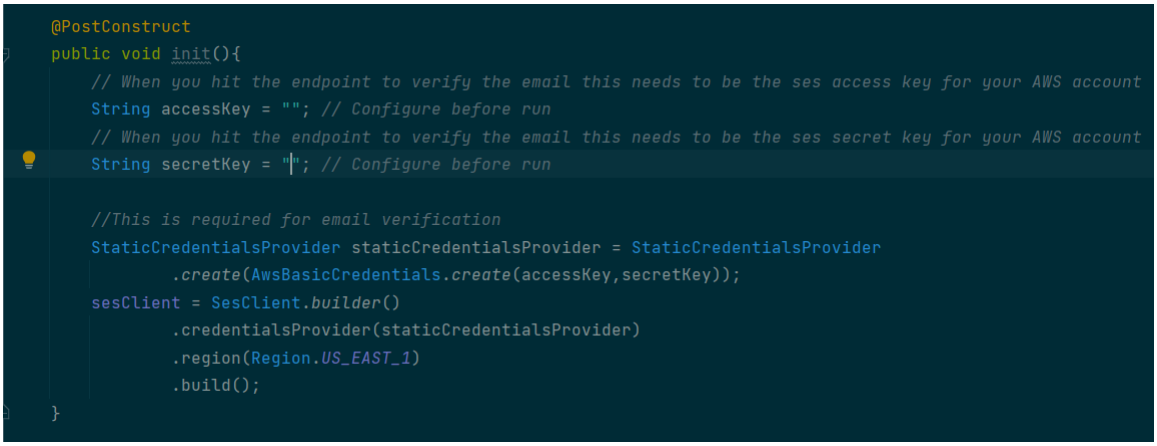
  For the purpose of simplicity, I have added a comment Configure before run on every line where changes are to be made before run based on your setup.
  You can find the screenshots for user-service config changes below:

```yaml
data:
  mongodb:
    host: ec2-3-232-57-4.compute-1.amazonaws.com # Configure before run
    port: 27017
    database: user-service

server:
  port: 8083

kafka:
  producer:
    bootstrap-servers: ec2-44-207-79-237.compute-1.amazonaws.com:9092 # Configure before run
    topic: bmc-notification
```

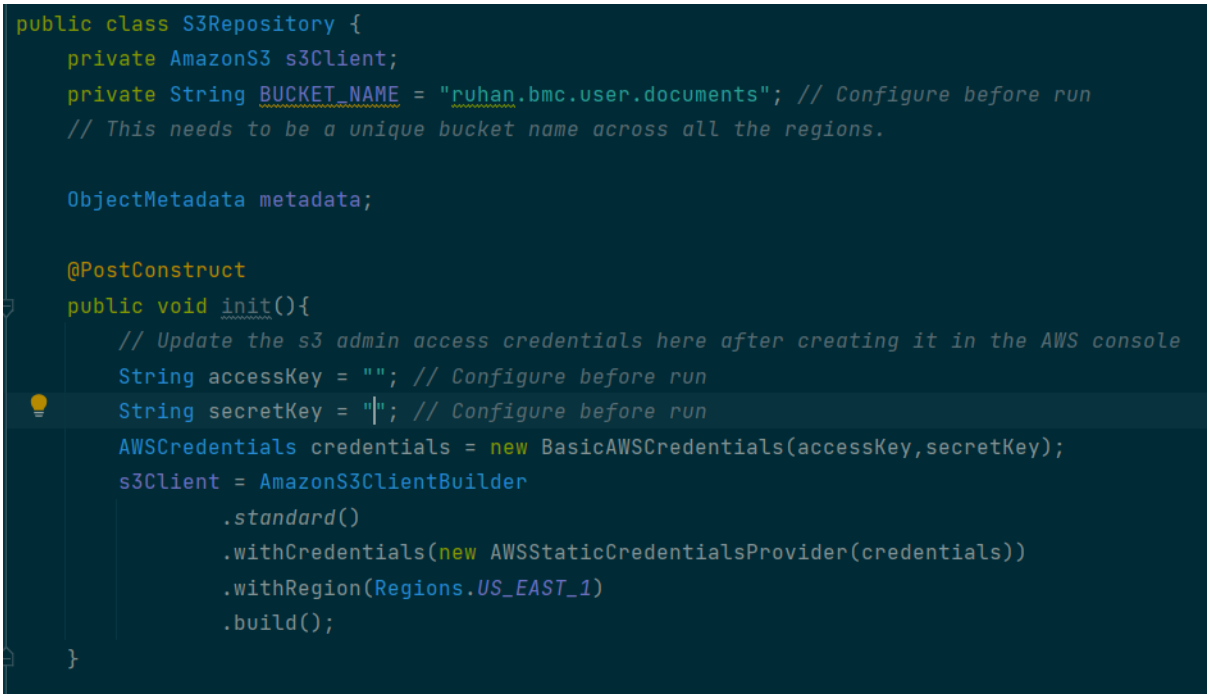
application.yml



```java
@PostConstruct
public void init(){
    // When you hit the endpoint to verify the email this needs to be the ses access key for your AWS account
    String accessKey = ""; // Configure before run
    // When you hit the endpoint to verify the email this needs to be the ses secret key for your AWS account
    String secretKey = ""; // Configure before run

    //This is required for email verification
    StaticCredentialsProvider staticCredentialsProvider = StaticCredentialsProvider
            .create(AwsBasicCredentials.create(accessKey,secretKey));
    sesClient = SesClient.builder()
            .credentialsProvider(staticCredentialsProvider)
            .region(Region.US_EAST_1)
            .build();
}
```

EmailService



S3Repository

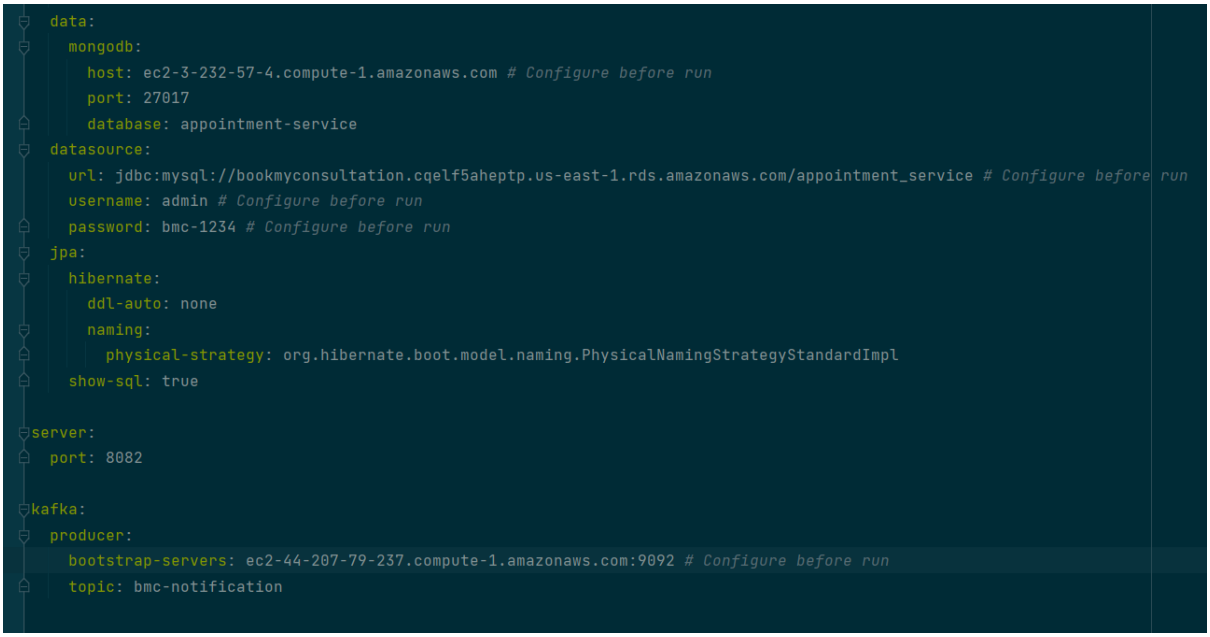
Appointment Service (appointment-service):

- **Logic and Design:** In appointment service I have used mapping method to take the input for doctor availability as map and then mapped them to Availability Entities as defined in schema.

  Appointment service has synchronous calls being made to doctor-service and user-service using open-feign during appointment booking call. I have added comments for the same in code. Considering we only receive the id's I make calls to these two services in order to fetch user and doctor details for notification-service and DB update.

  Request response for most of the endpoints are as expected apart from the few exception response changes mentioned in the doctor-service.

- **Configuration Change:**
  For the purpose of simplicity, I have added a comment Configure before run on every line where changes are to be made before run based on your setup.
  You can find the screenshots for appointment-service config changes below:



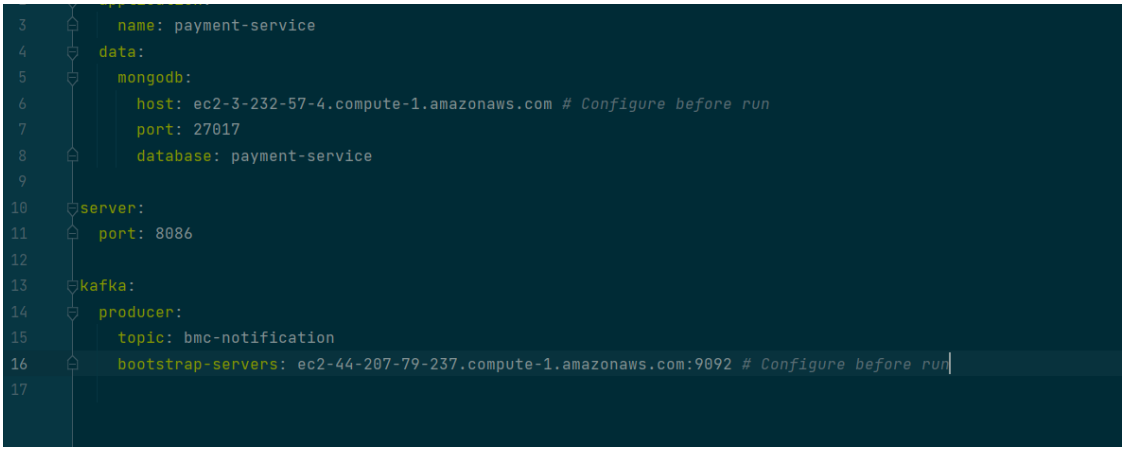
application.yml

Payment Service (payment-service):

- **Logic and Design:** In payment service we have only one endpoint for making a payment for appointment. Here I am doing a synchronous call to first update the payment status in appointment-service and then getting that same appointment object to send payment update notification to user by sending it to notification-service via kafka topic (Payment Event).

  Here I am also storing the transaction details at payment repo first in order to have a backup of transaction and also if there occurs an error, I have used transaction roll back to reverse any DB updates.

- **Configuration Changes:**
  For the purpose of simplicity, I have added a comment Configure before run on every line where changes are to be made before run based on your setup.
  You can find the screenshots for payment-service config changes below:



application.yml

Notification Service (notification-service):

- **Logic and Design:** In notification-service I have used a single topic bmc-notification to receive notifications from all services for all event types, but I have used the key of a record to identify the event type and process them accordingly.

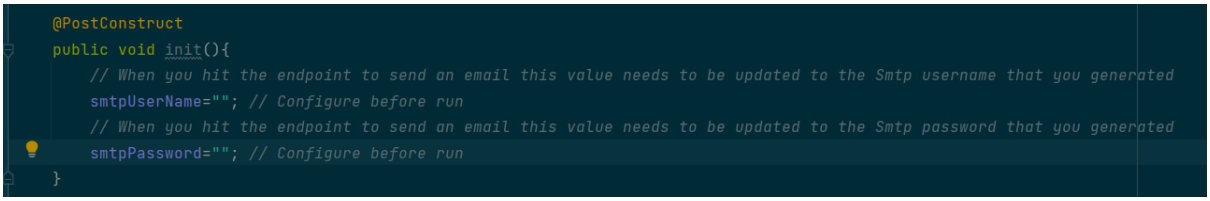
  The kafka consumer only consumes and forwards the record on to the event processor service where the actual logic for handling different events resides.

  Used user, doctor, appointment event types to process them separately before handing it over to email service for notification.

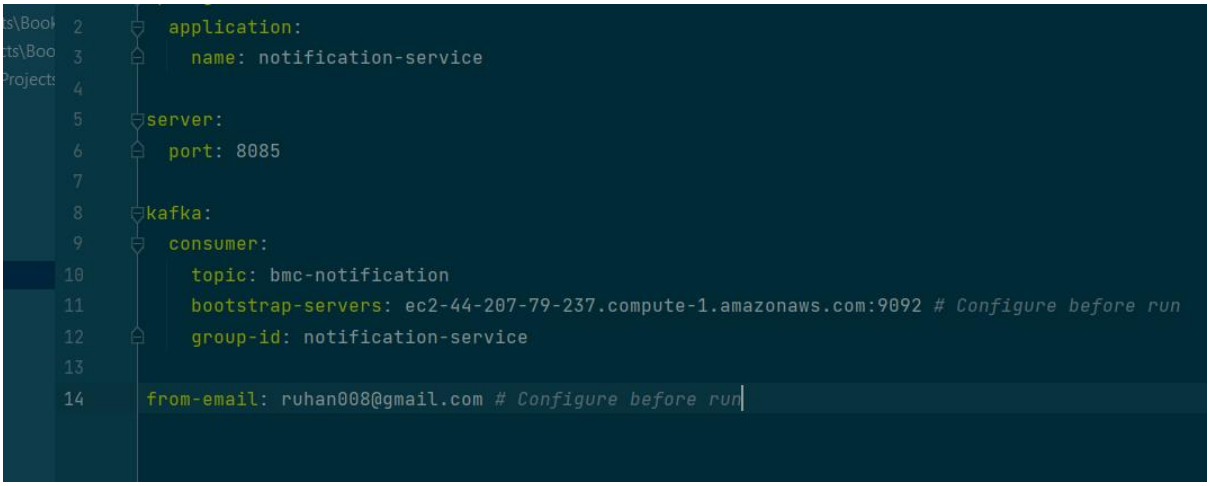
  In notification service it's important that you set a from-email in application.yml which is verified in ses to send emails to other verified user's else mail will not be sent.

  There are 5 email templates present under templates folder of src/main/resources. Each template is used for notification of user-appointment, doctor-approval, doctor-rejection, payment-confirmation and prescription-upload.

- **Configuration Change:**
  For the purpose of simplicity, I have added a comment Configure before run on every line where changes are to be made before run based on your setup.
  You can find the screenshots for notification-service config changes below:



EmailService



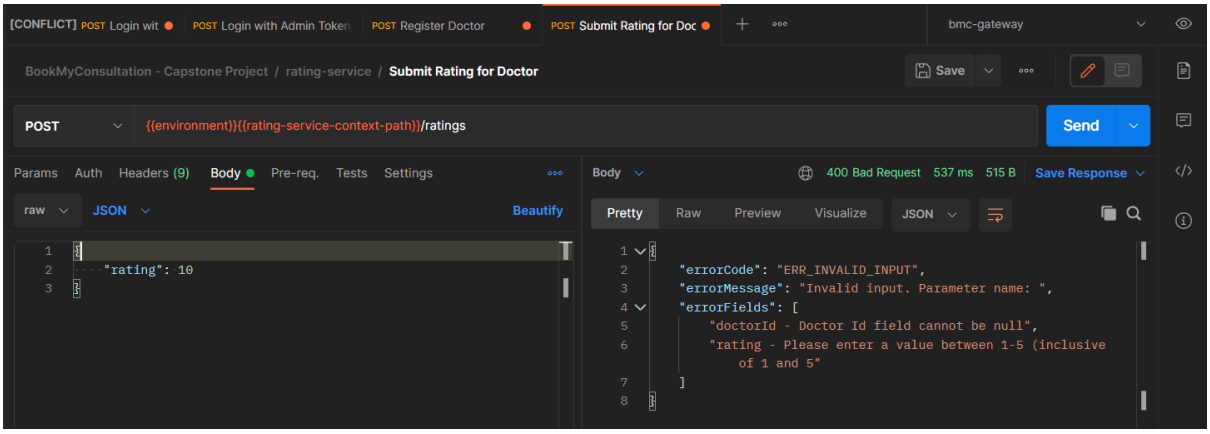
application.yml

Rating Service (rating-service):

- **Logic and Design:** In rating service we have a single endpoint to accept the rating for any doctor and I have modified the response to send a string confirming the acceptance of rating.

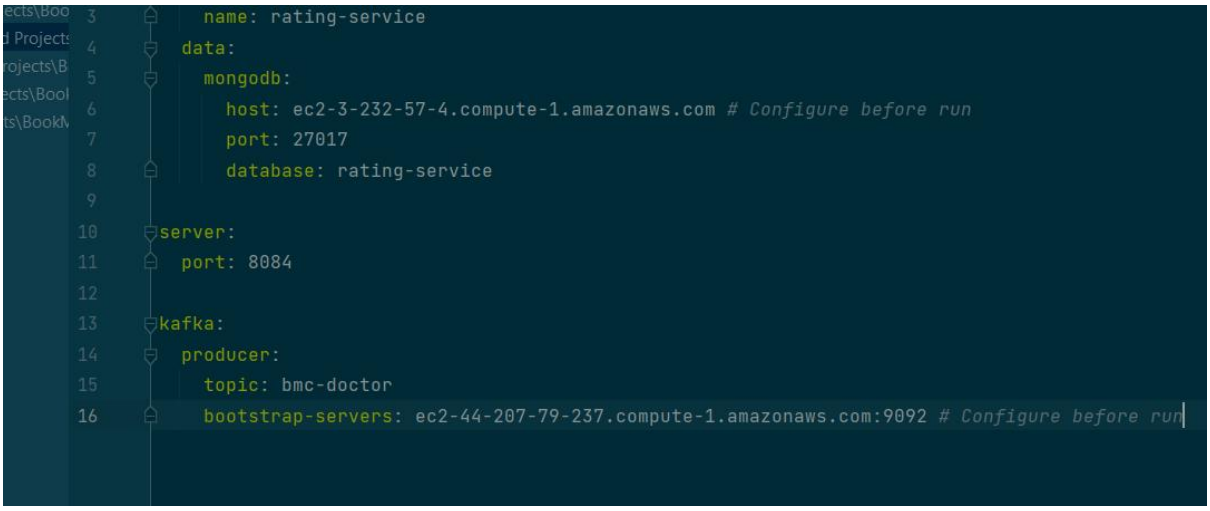
  I have added comments in the code for logic. What I am doing is that I have two separate tables, one for just storing the rating received (Rating) and other for storing the count with average doctor rating (DoctorRating). The logic adds the rating to individual ratings (1-5) counts and the calculates the average doctor rating for the doctor and updates the DoctorRating repository, which is the sent to kafka topic bmc-doctor which is being listened by doctor-service so that it can updates the doctors average rating in its repository whenever a rating event happens.

  I have also added validation on the rating received in case the user inputs a value for rating outside the range of 1-5 or if doctorId provided is null.



Error response for doctorId - null and rating - 10

- **Configuration Change:**
  For the purpose of simplicity, I have added a comment Configure before run on every line where changes are to be made before run based on your setup.
  You can find the screenshots for rating-service config changes below:



application.yml

BMC Gateway (bmc-gateway):

- **Logic and Design:** For bmc-gateway the routes are already configured to route the incoming requests to different services based on a predicate (context-path).

- **Configuration Change:**
  There is no configuration change required for bmc-gateway.

Docker Compose (docker-compose):

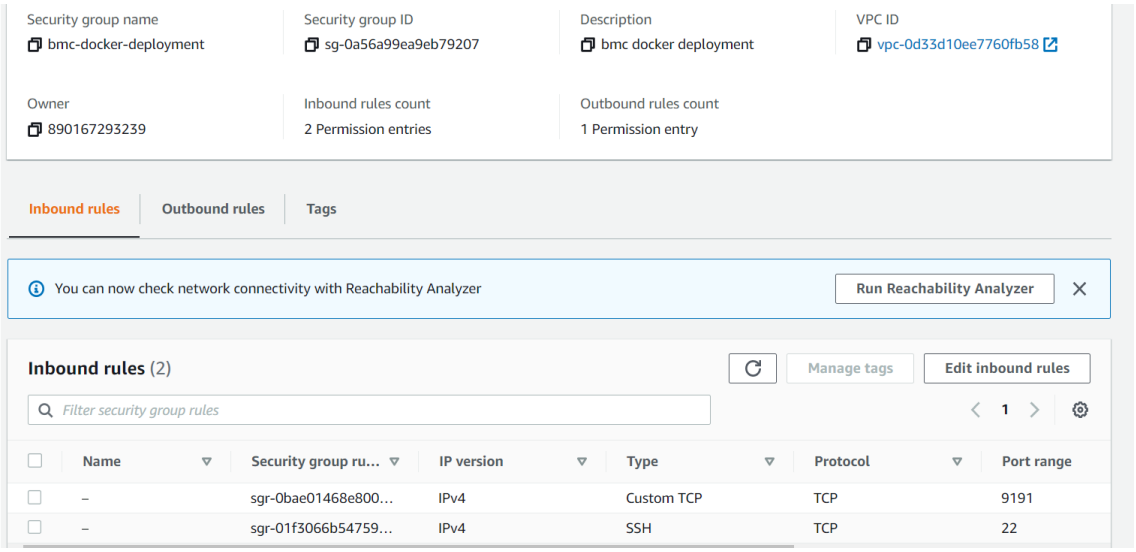
- The docker-compose.yml file is configured to build and run all the services once all the config changes are done as explained above and other DB and Kafka setup is also done.

  Command: docker-compose up or sudo docker-compose up can be used depending on whether you're running locally or on an ubuntu instance.

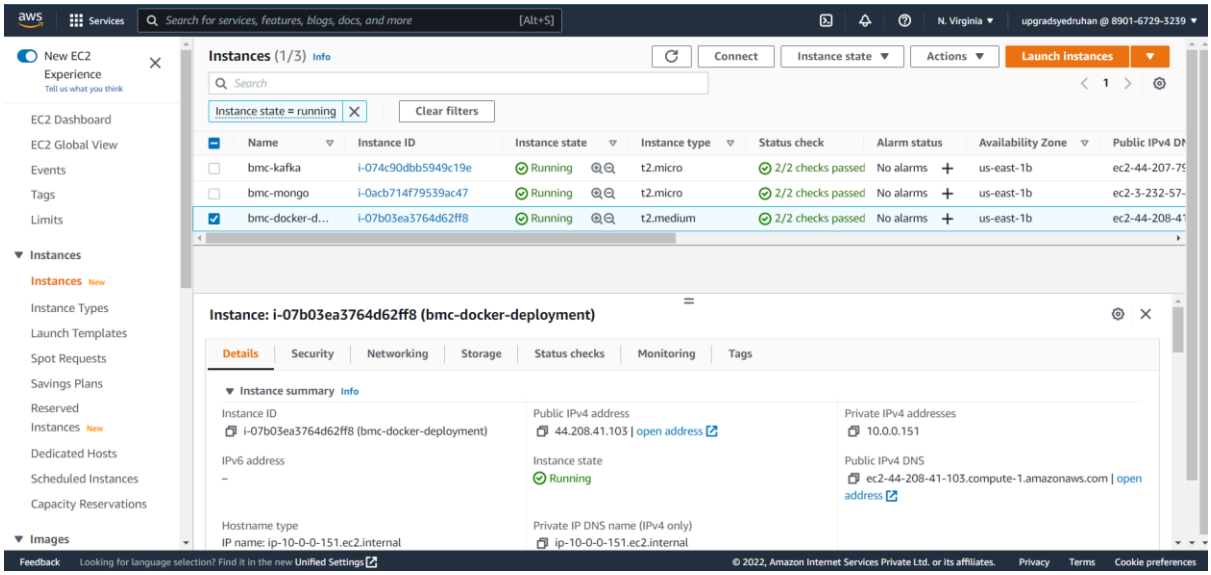
Note: All the service folders have Dockerfile's written for them.
Also, in case you're running it on aws ubuntu or ec2 instance and the build isn't able to find the folder then please replace build tag value under services with eg. ./notification-service with notification-service and try (remove ./ and try).

Screenshots for request and response on all services

I have deployed all services on an ubuntu instance using docker-compose and accessing all of them from local by only exposing bmc-gateway port 9191.

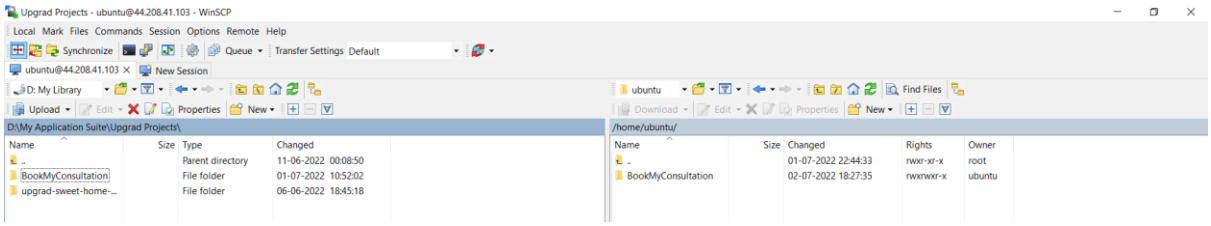


ubuntu instance security group

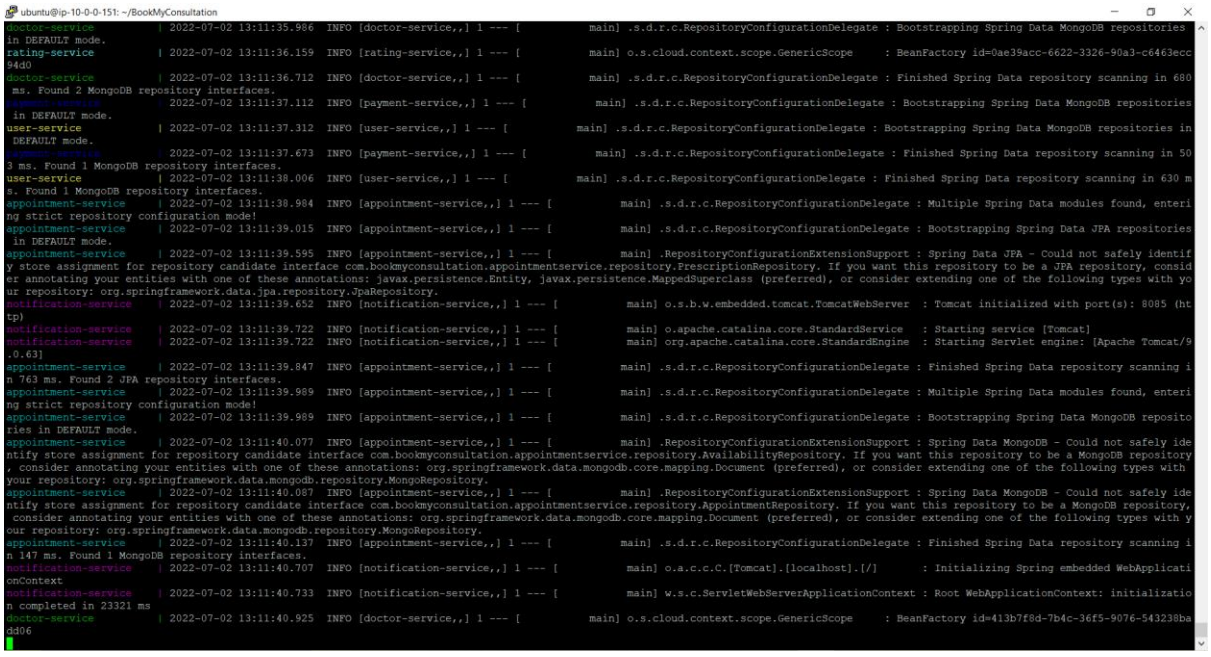


ubuntu ec2-instance

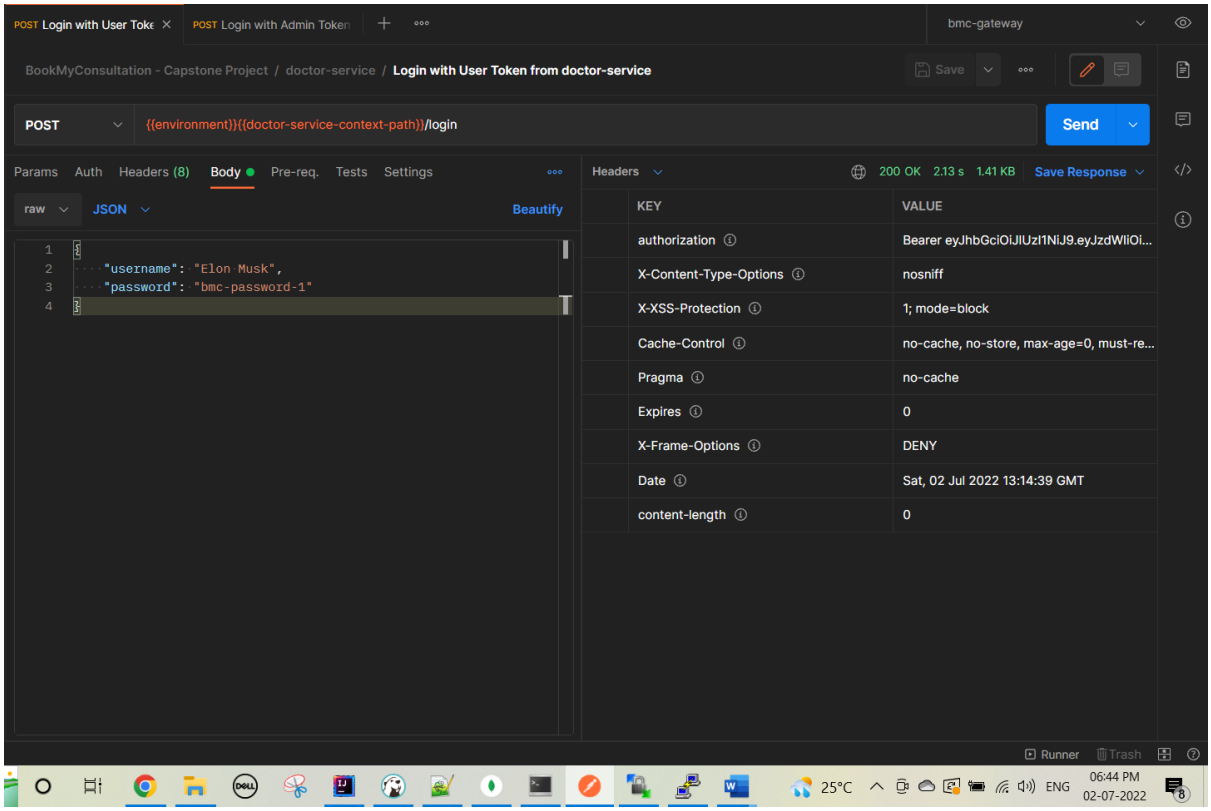
Postman environment setup: The environment variable has the ubuntu ec2 public ip with bmc-gateway port.



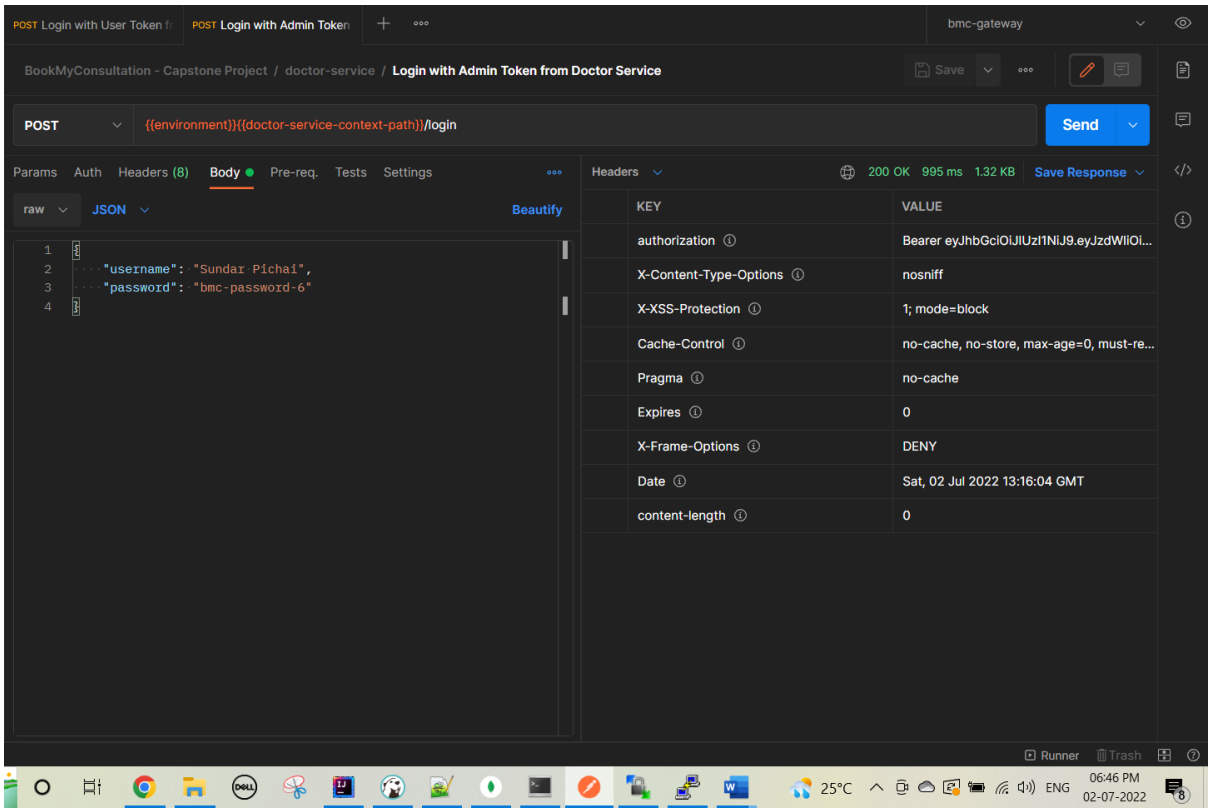
Uploaded via WinScp to ubuntu



Services Starting up

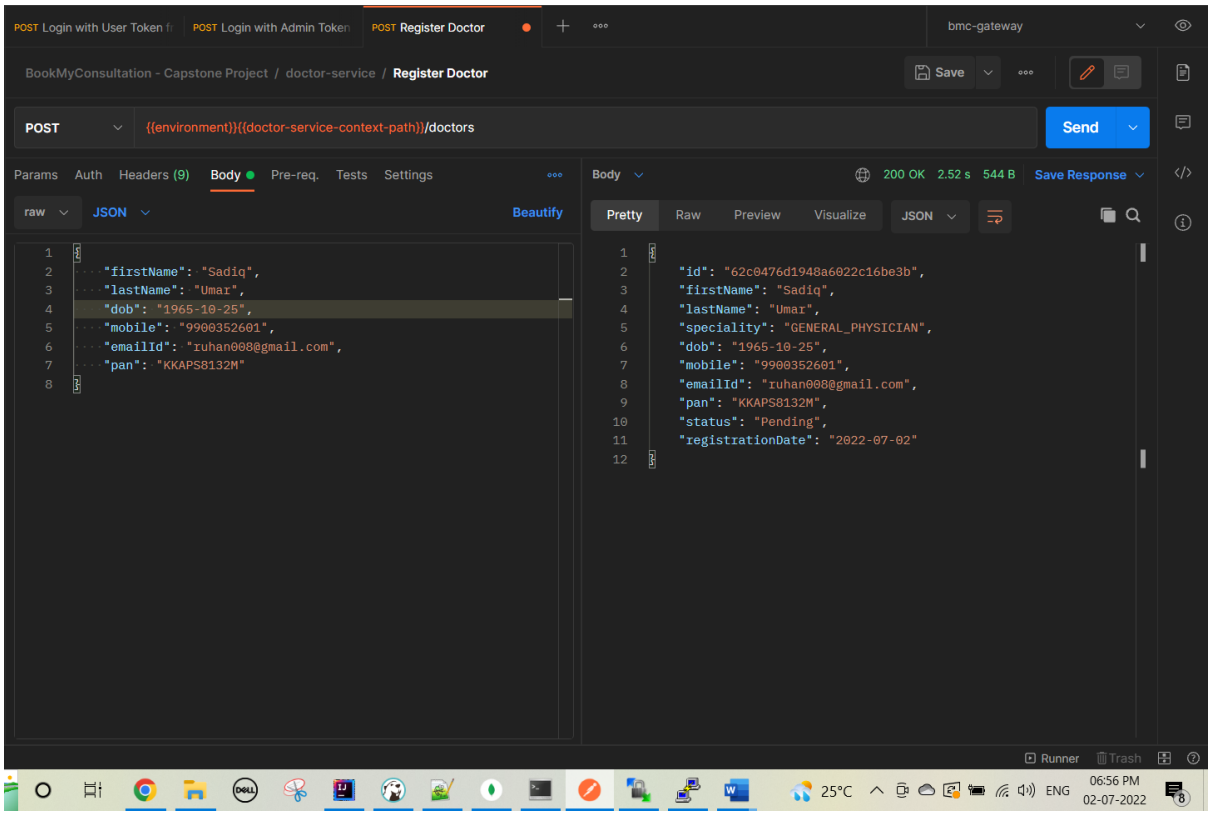


Getting user role token (Request & Response)

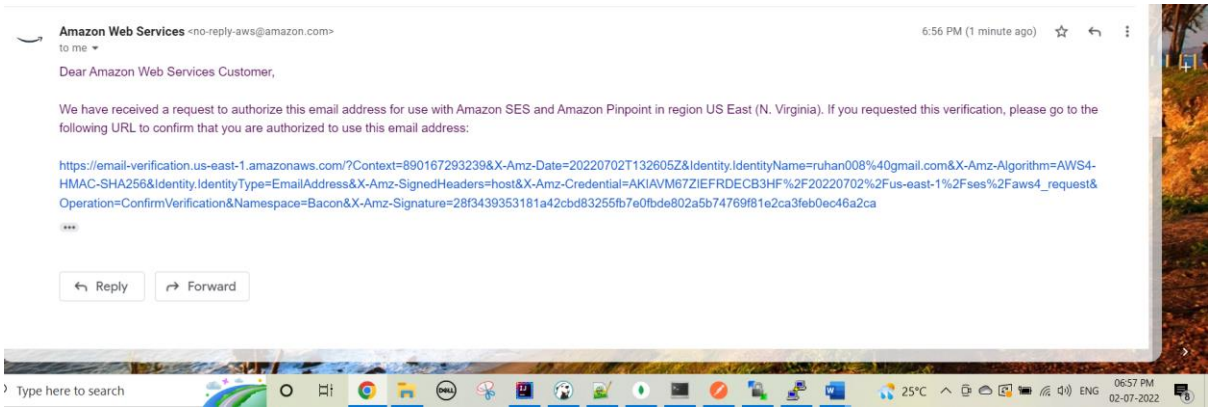


Getting admin role token (Request & Response)

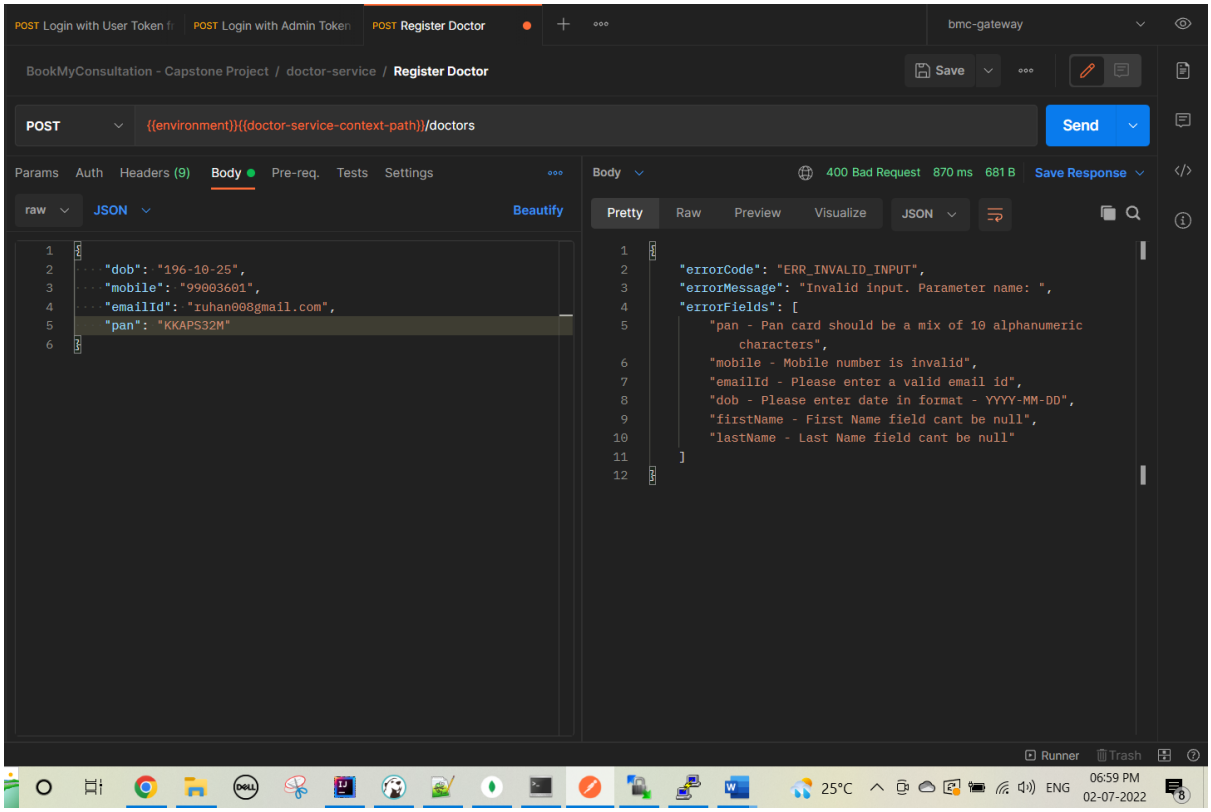
Doctor Service Request-Response:



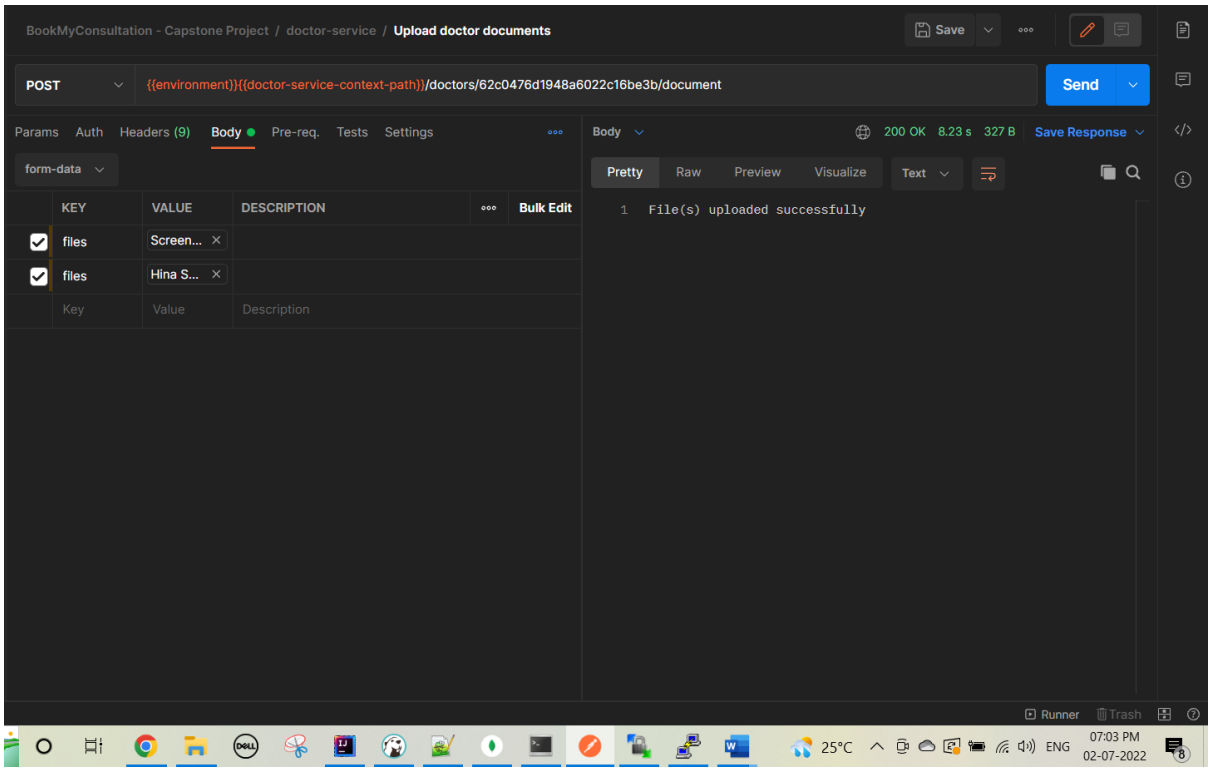
Register Doctor: Success (Request & Response)



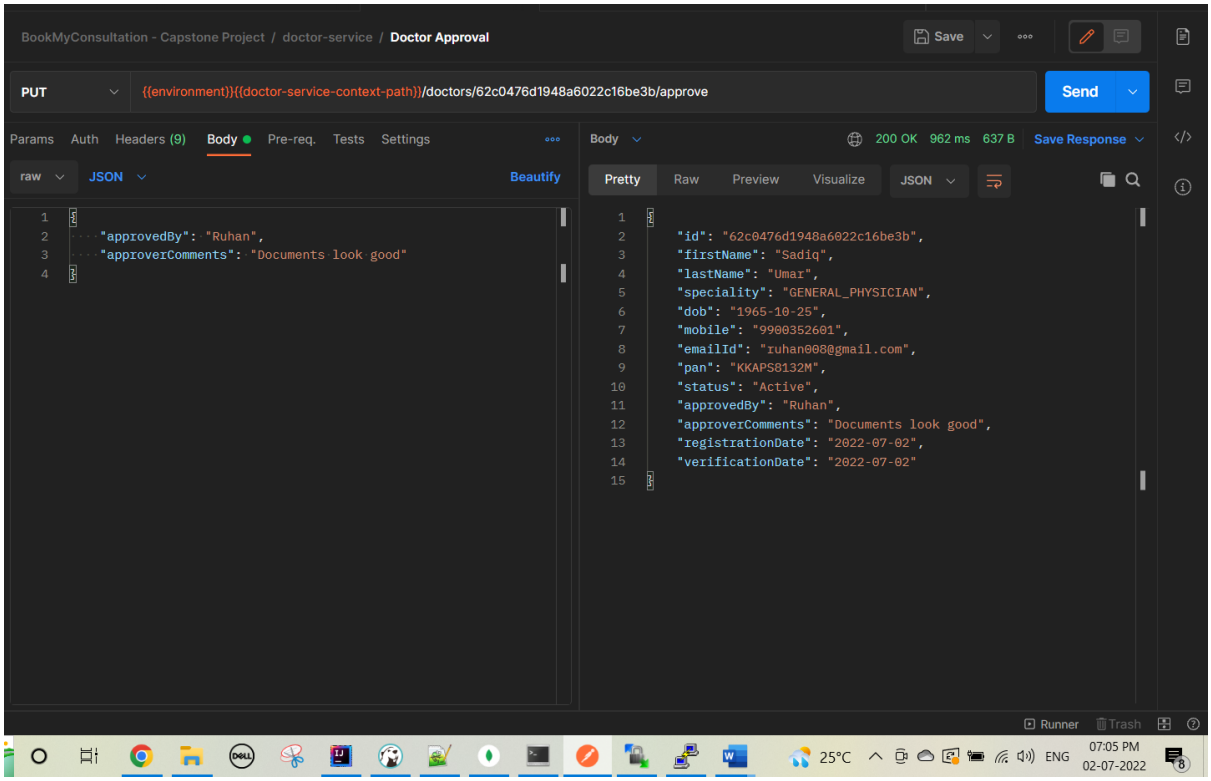
Email Verification



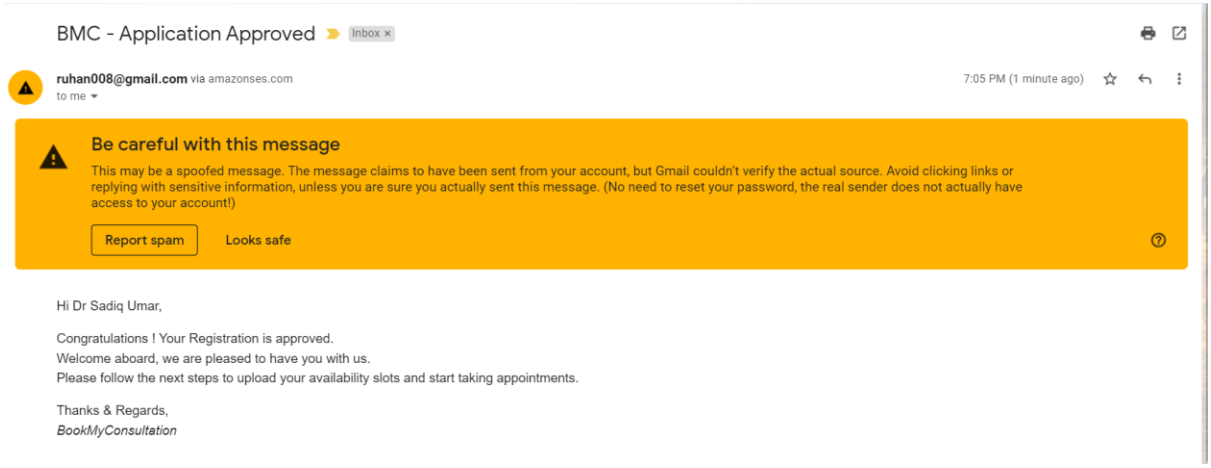
Doctor Registration: Error response for invalid field values.



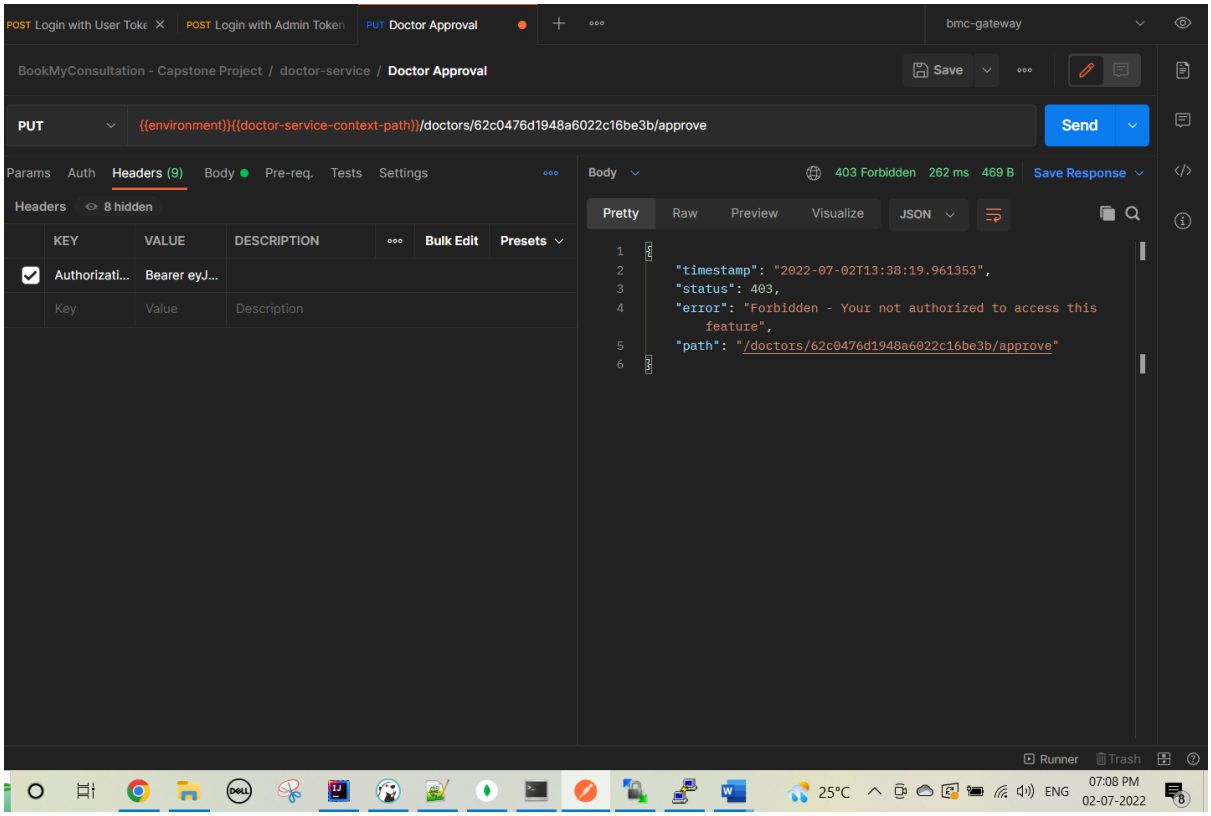
Upload doctor documents: Success



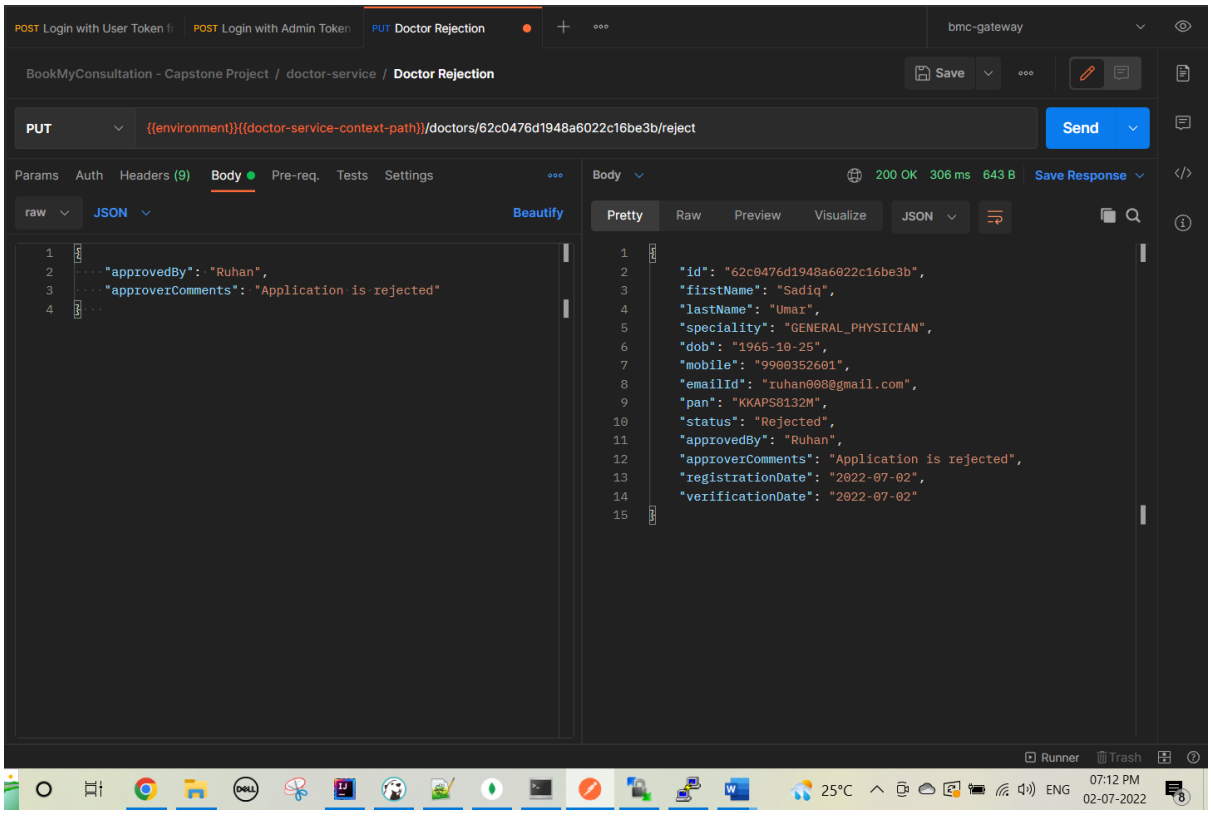
Approve Doctor Registration with Admin token in authorization header



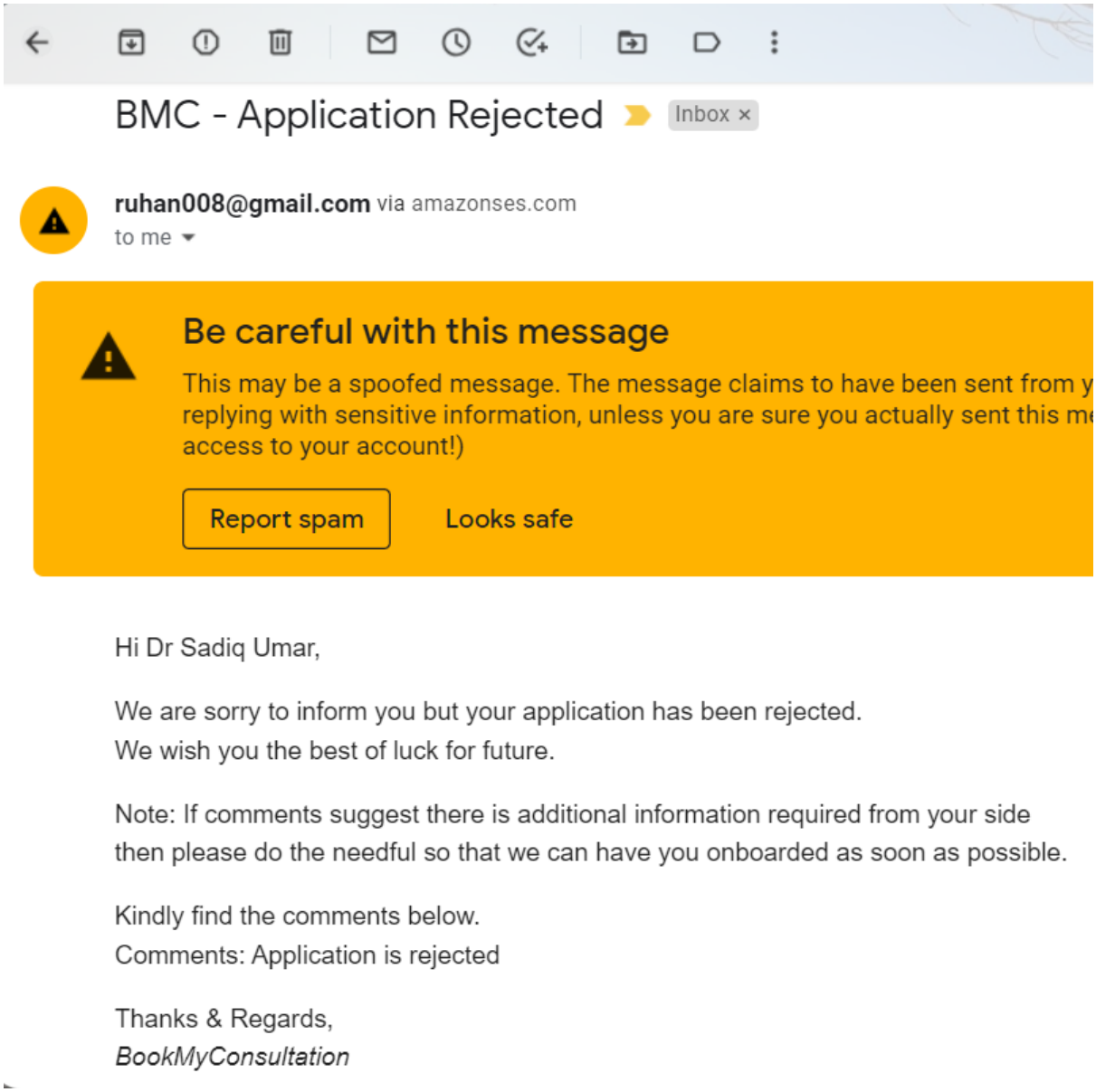
Approval Email Confirmation.



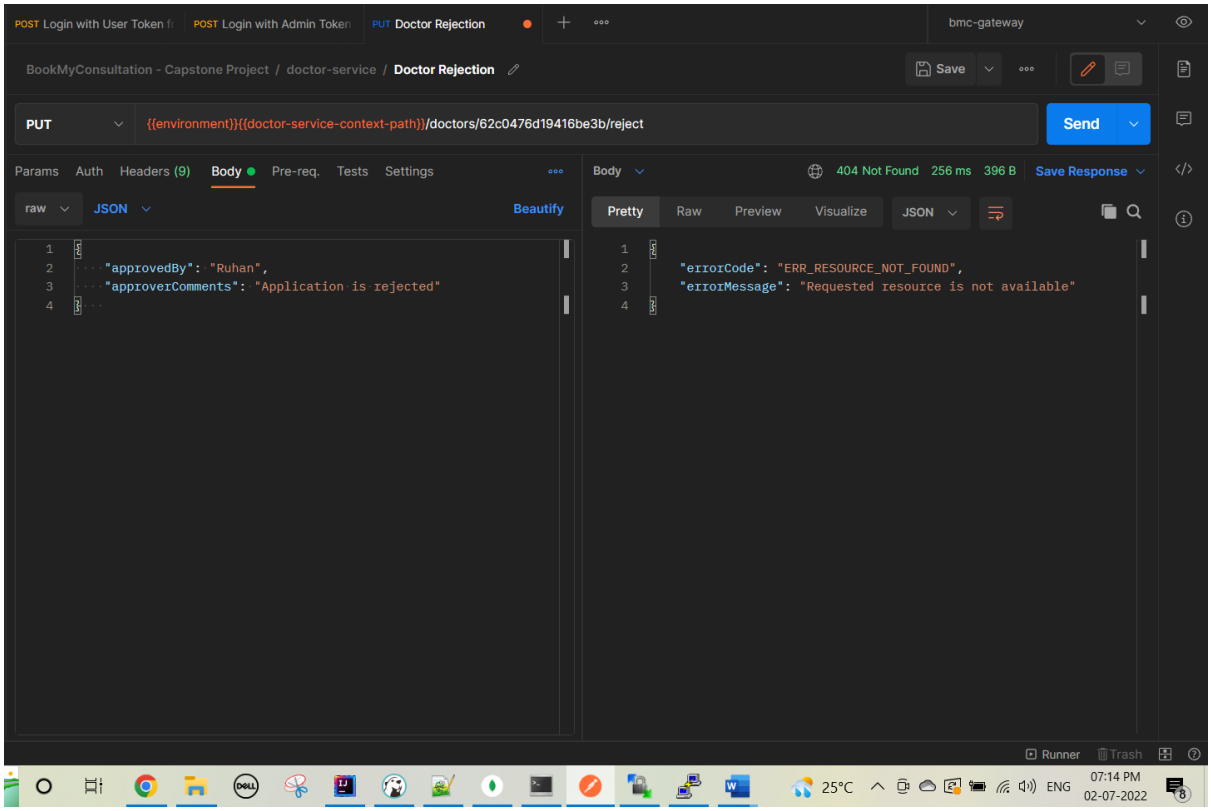
Forbidden error response for accessing approval with user token



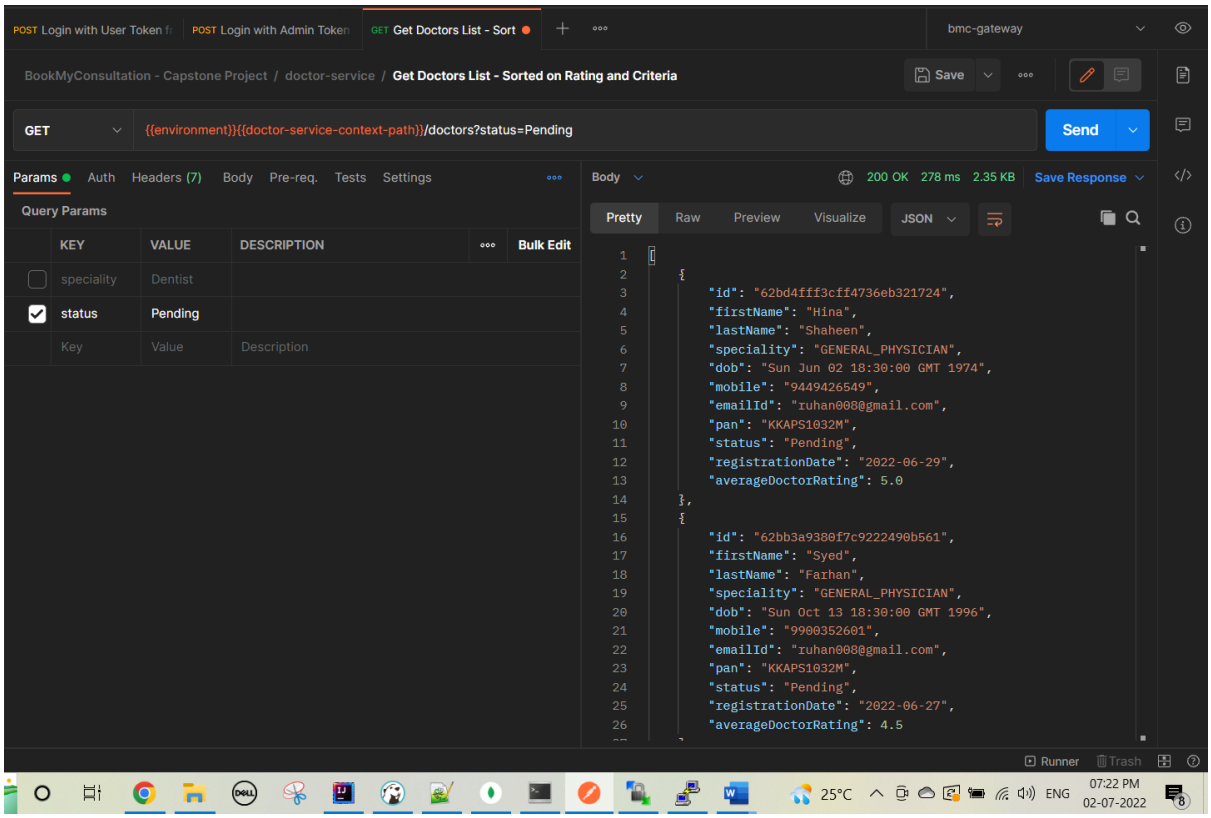
Doctor Rejection with admin token



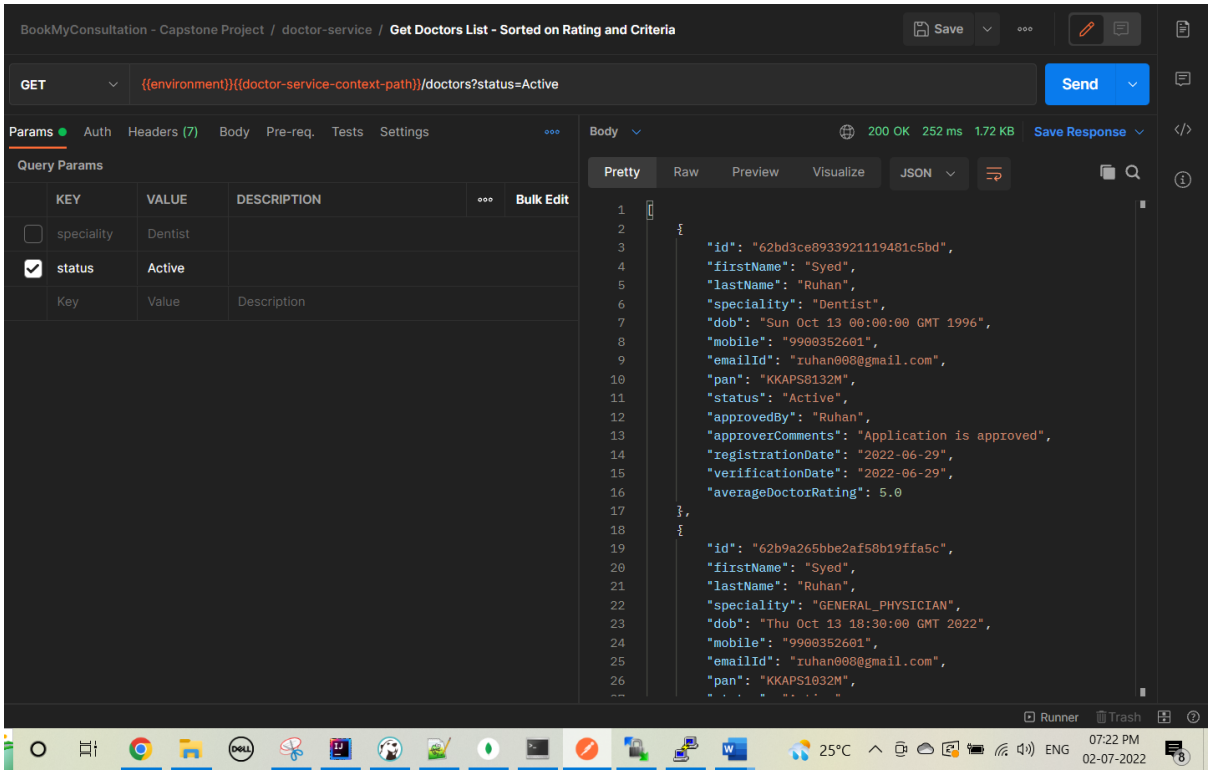
Doctor Rejection Email



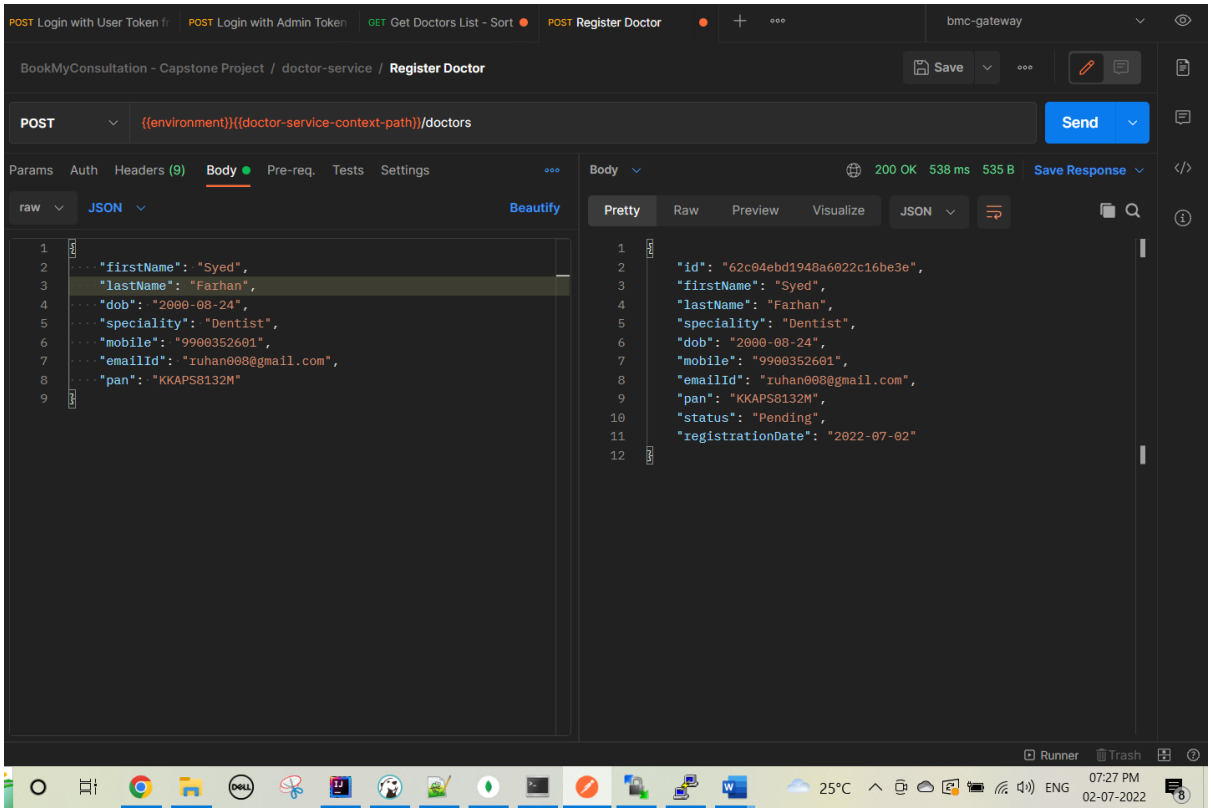
When invalid doctorId is sent. (Error fields is null and hence json property doesn't send it in response.



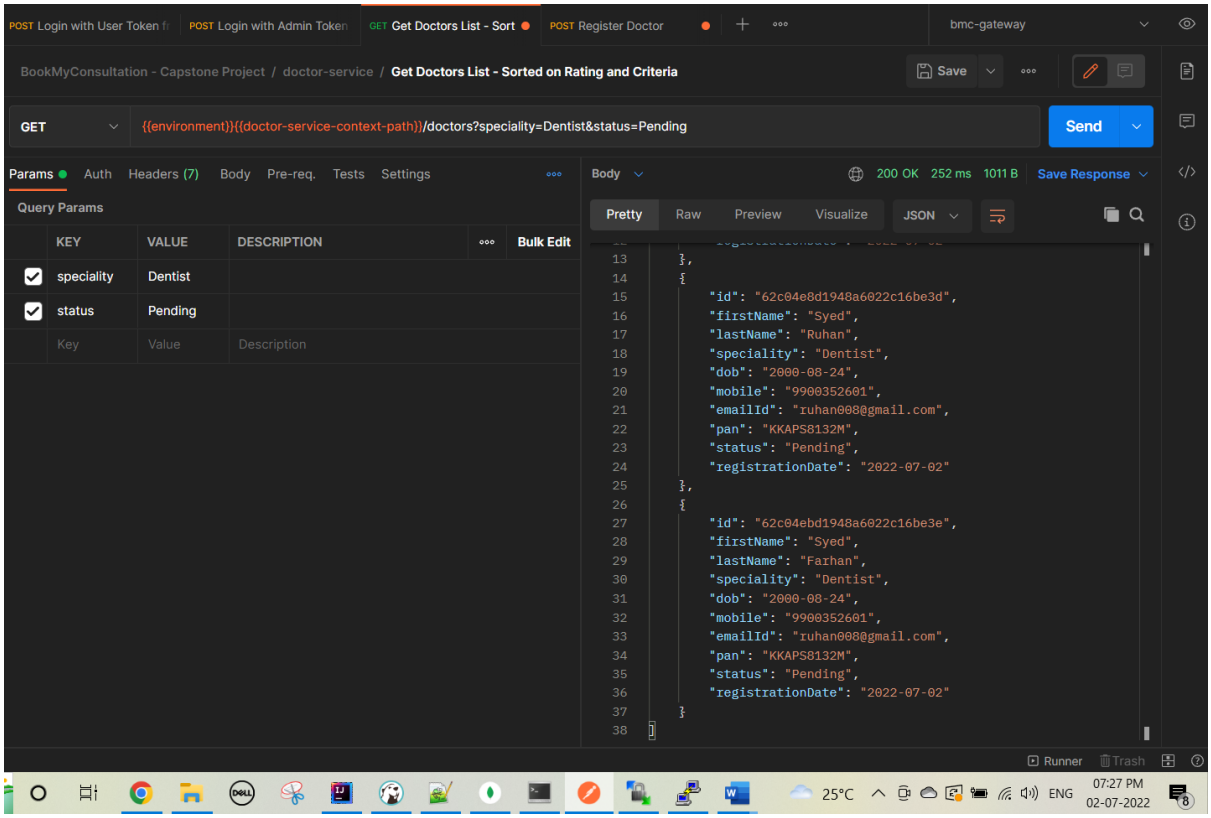
Get doctors list for status=Pending



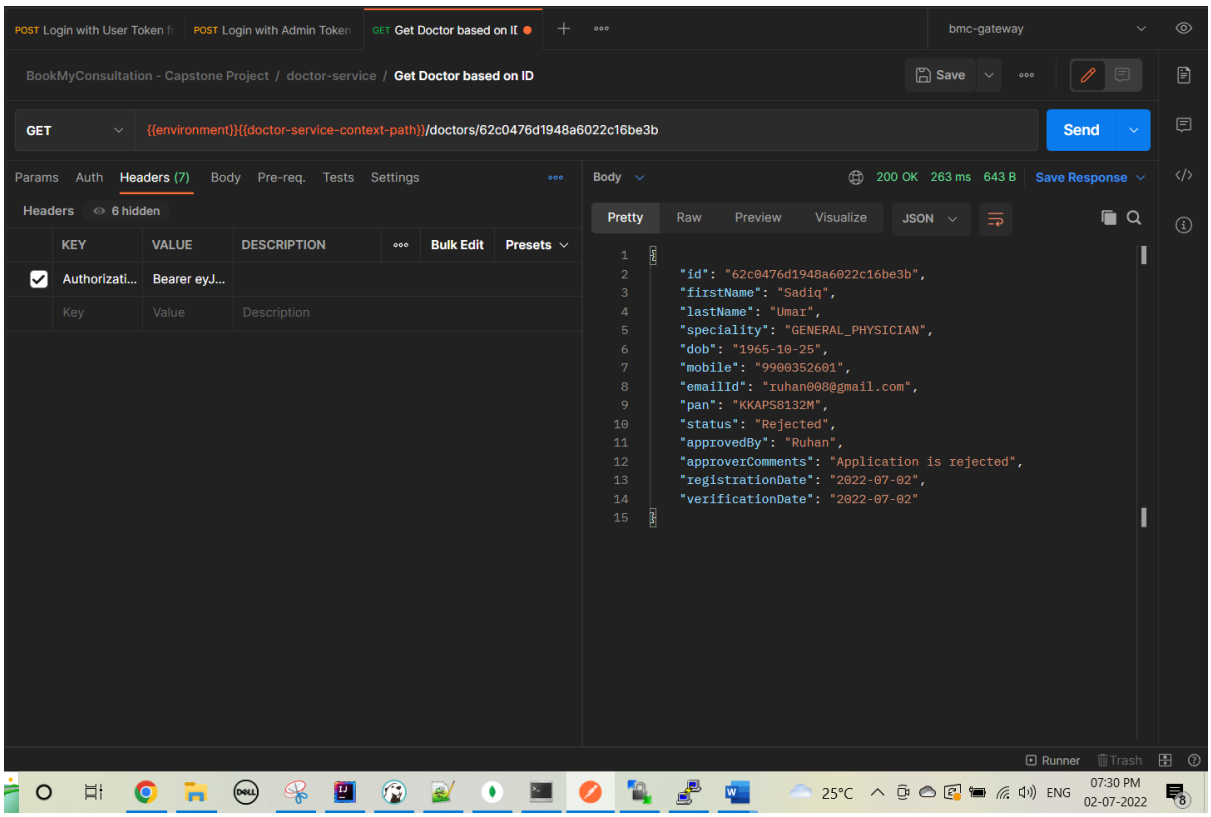
Get doctors list for status=Active



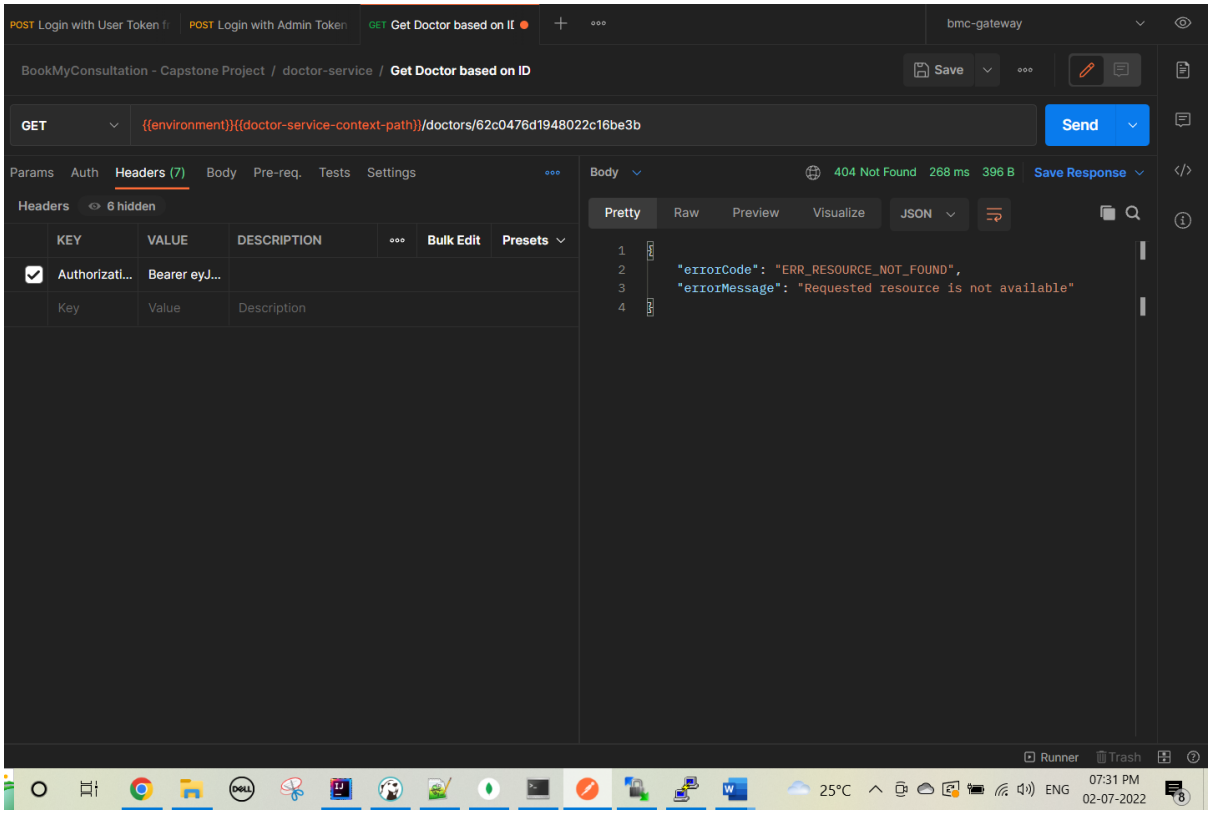
Upload a doctor with speciality Dentist



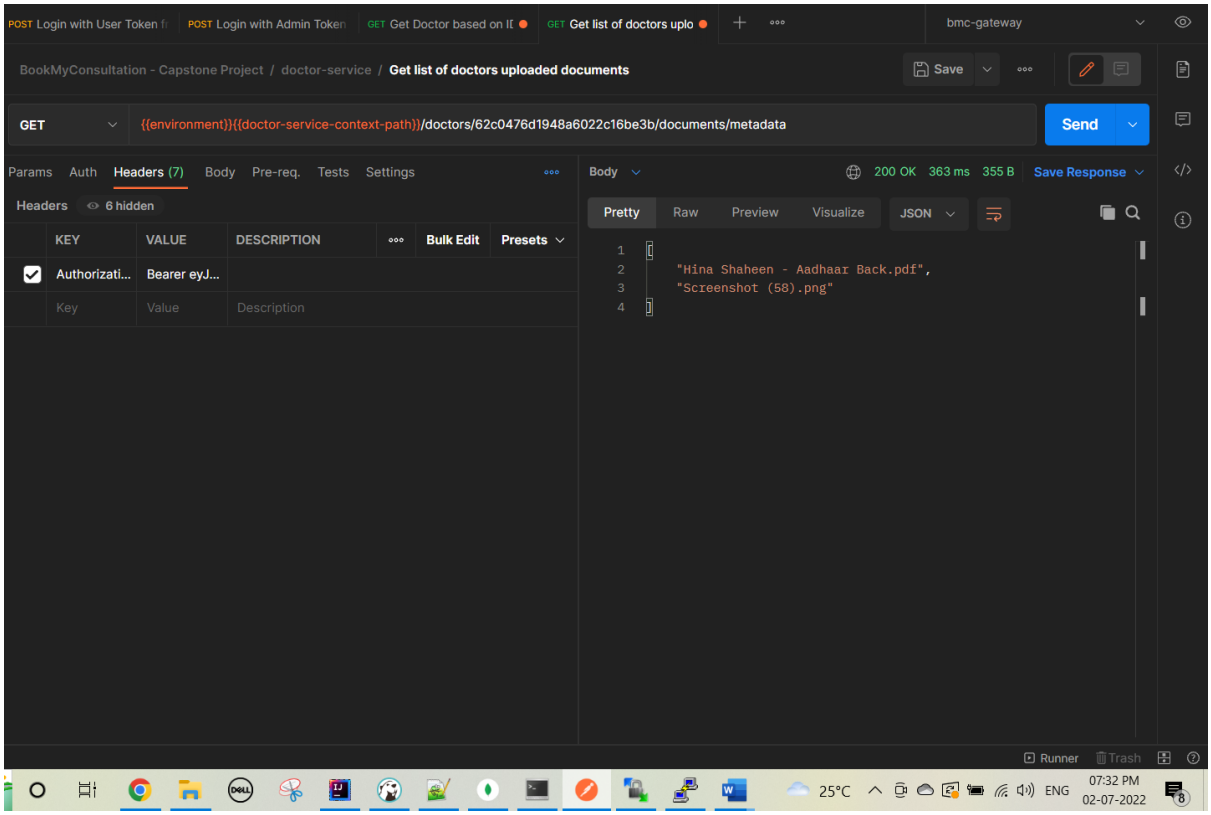
Get doctor with status=Pending and speciality=Dentist



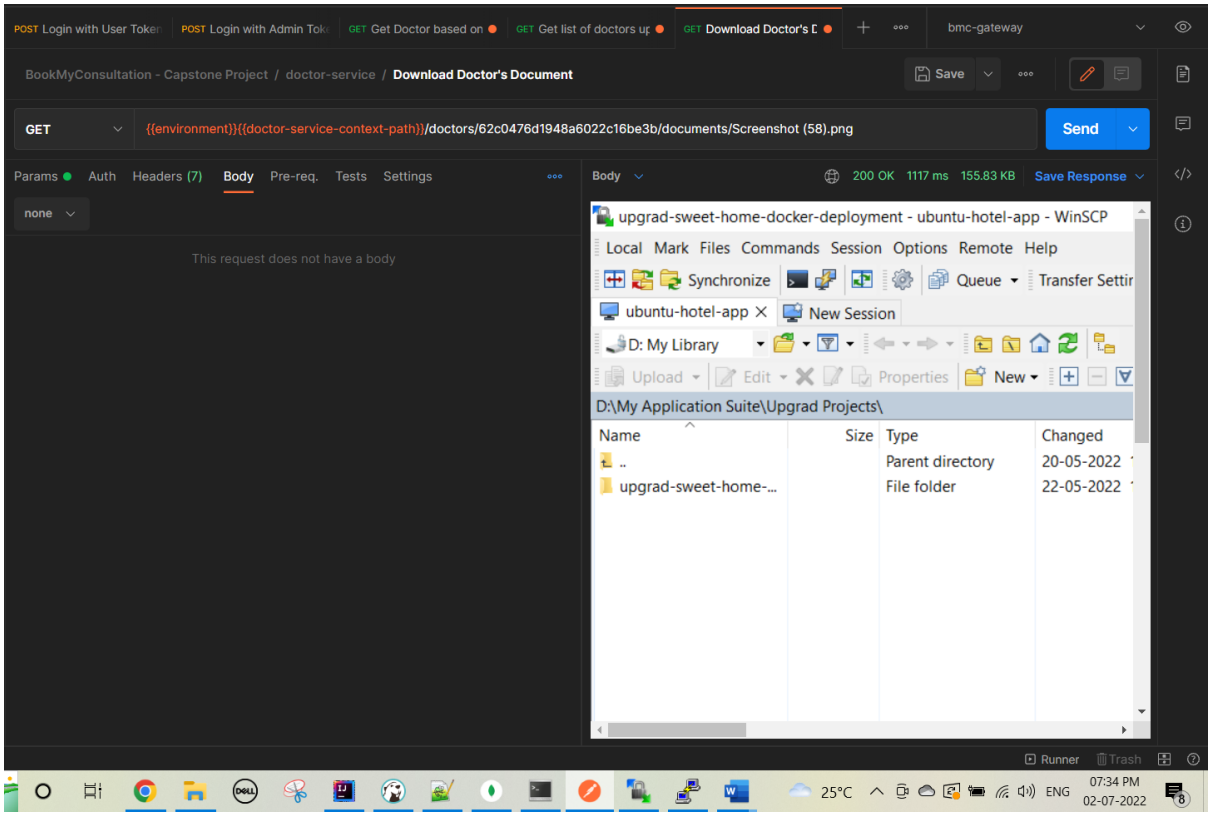
Get doctor by id



Invalid Doctor Id

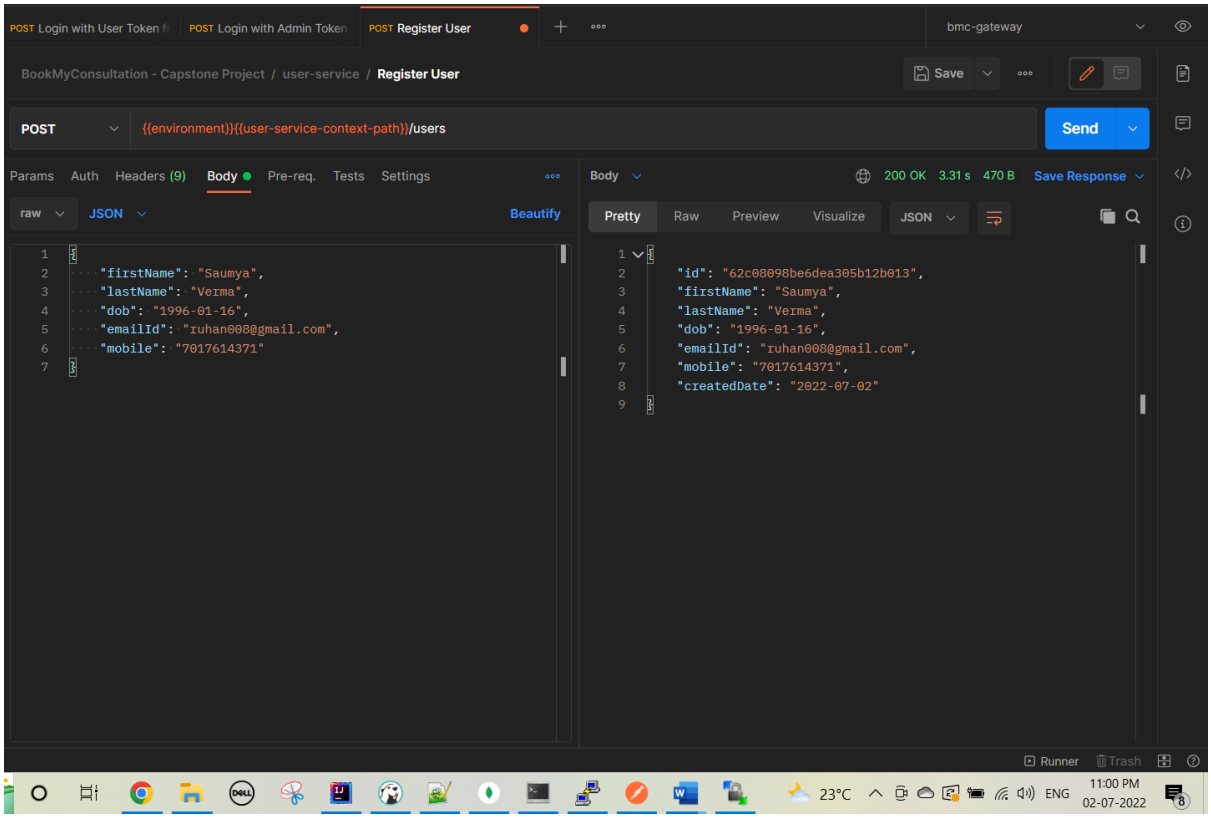


Get a list of uploaded documents by a doctor

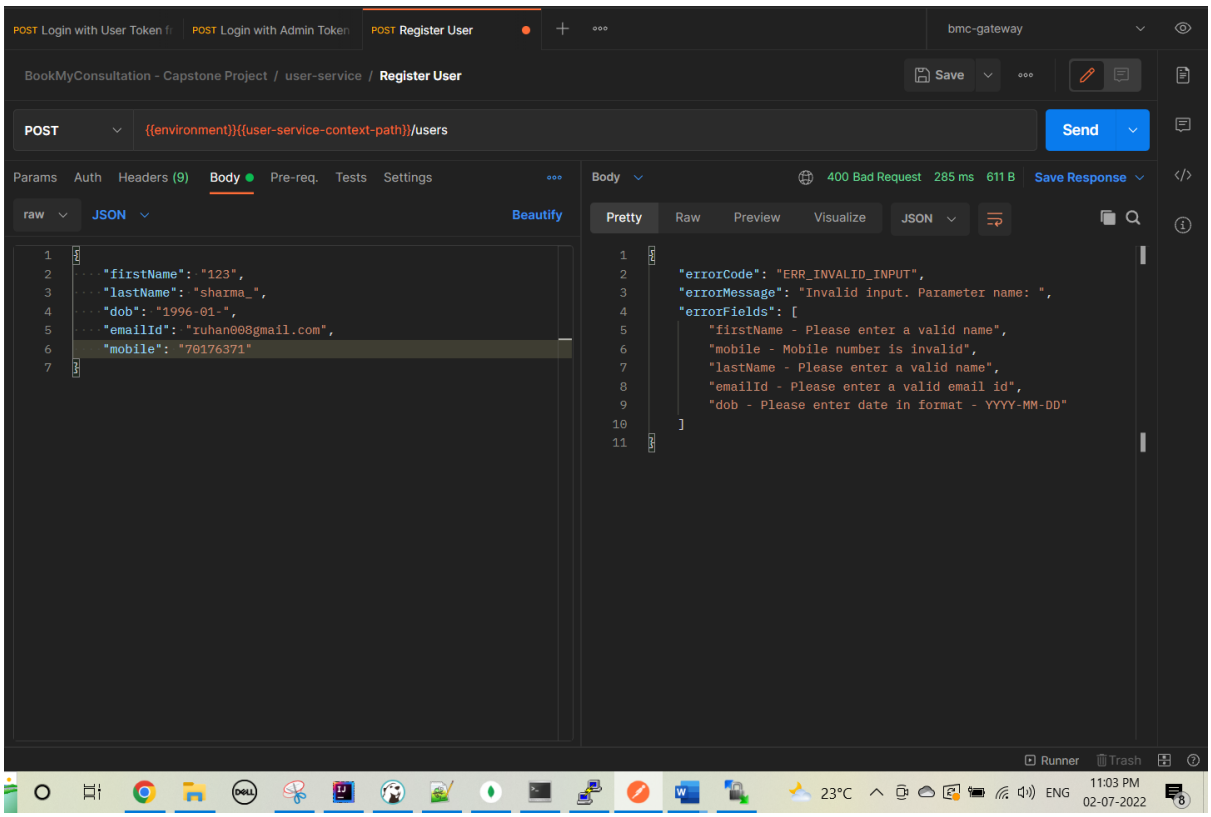


Download a document uploaded by a doctor.

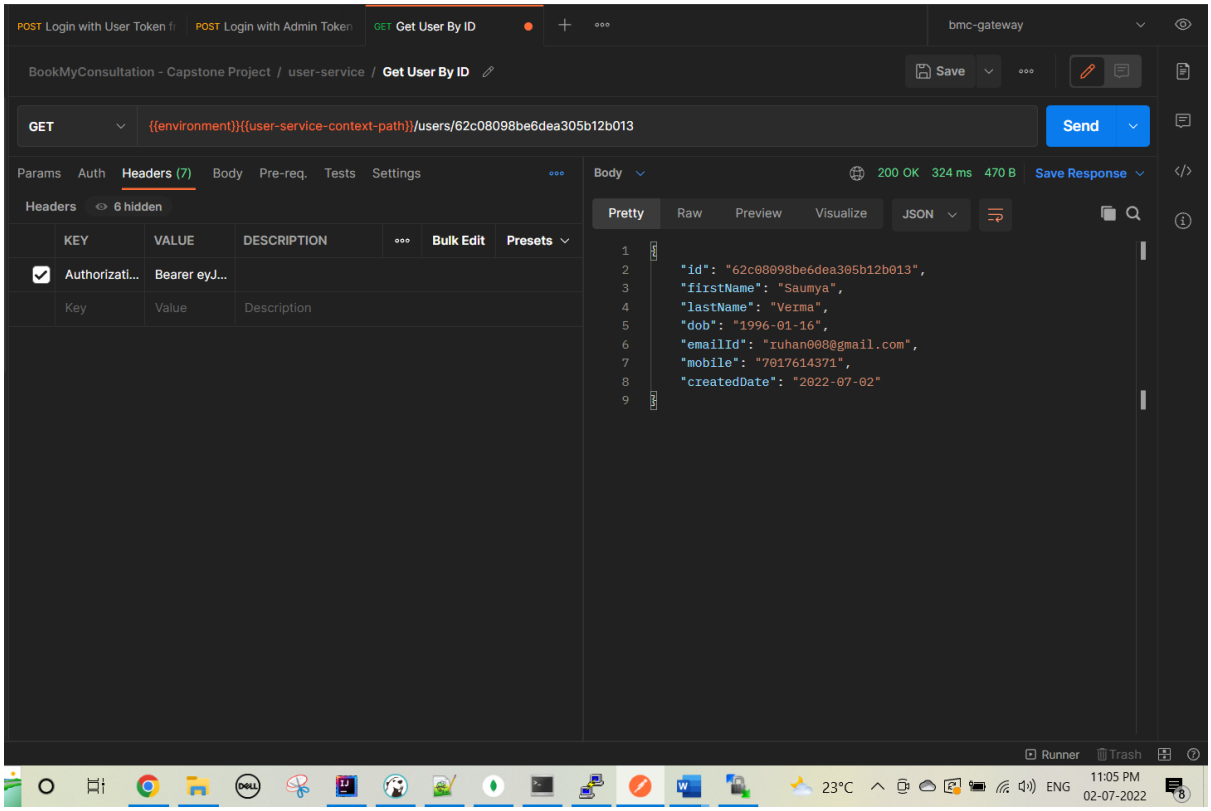
User Service Request-Response:



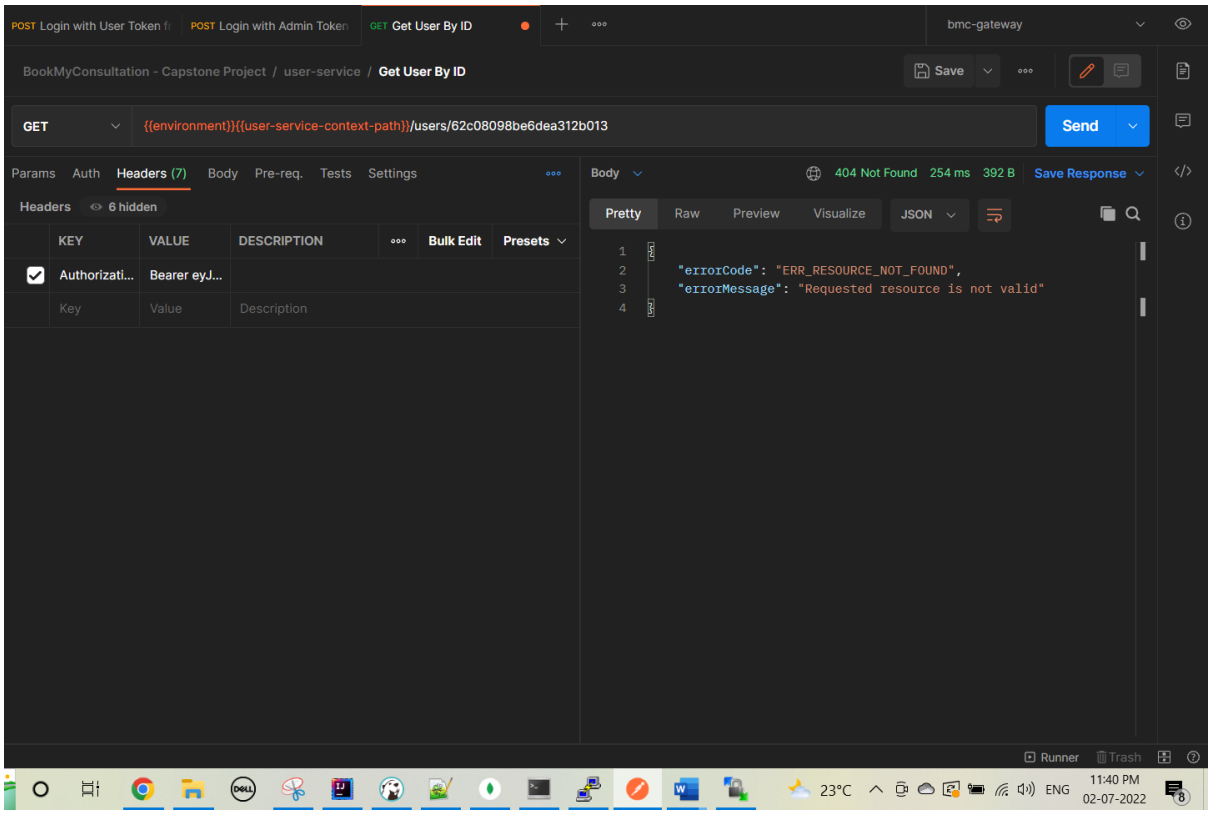
Register User: Success



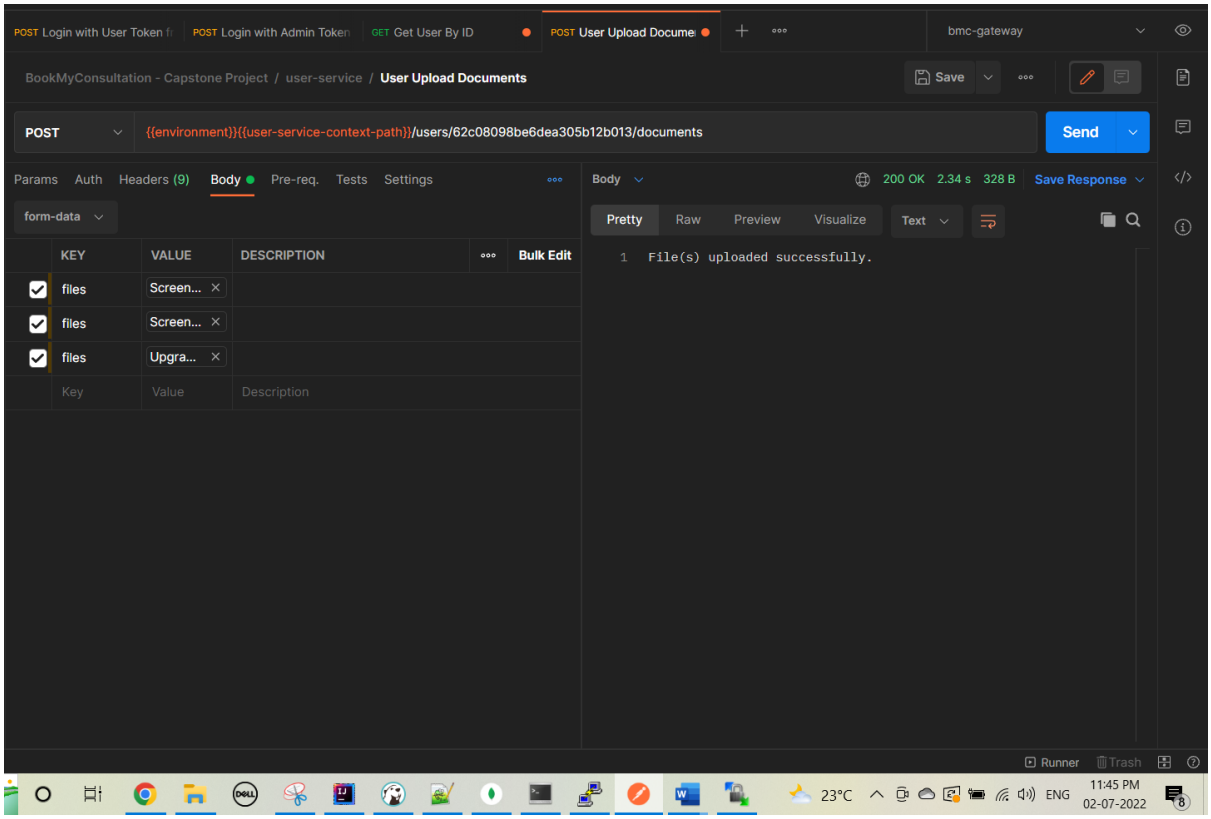
Register User: Invalid input error message



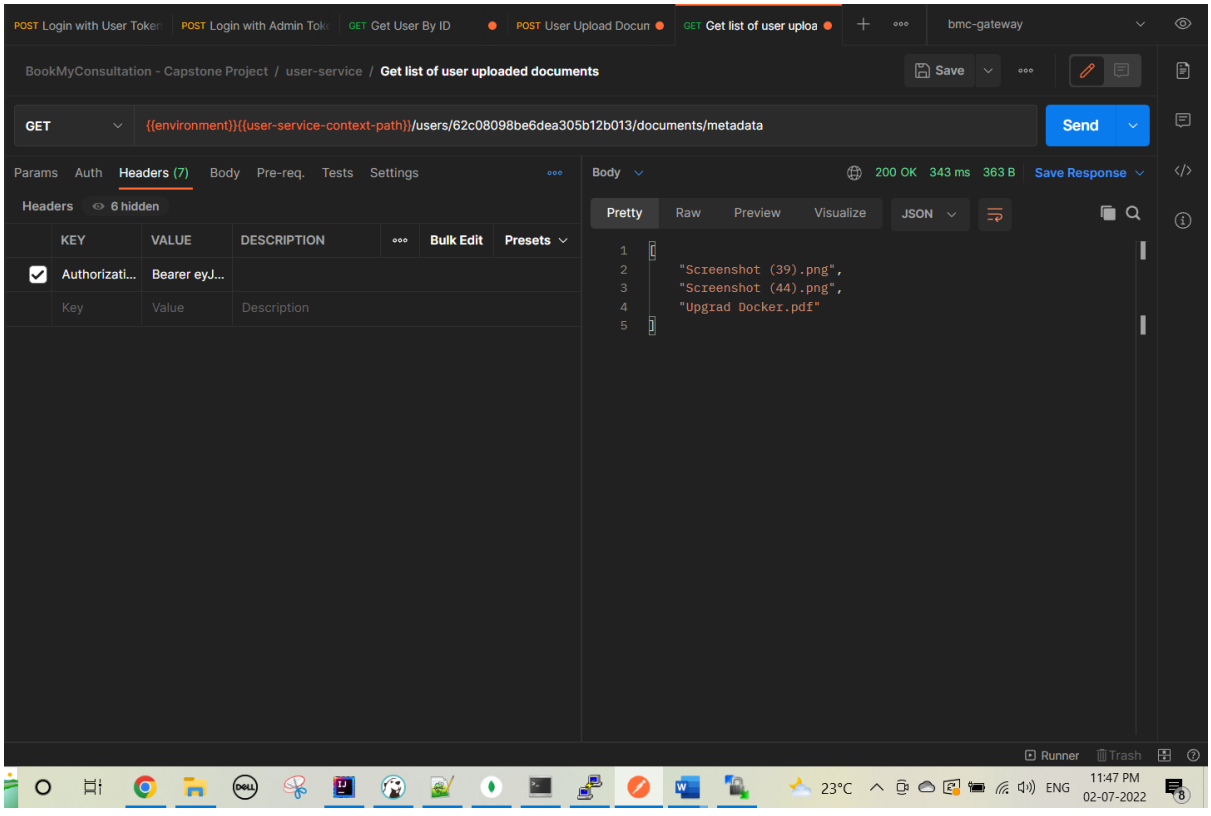
Get user details based on ID



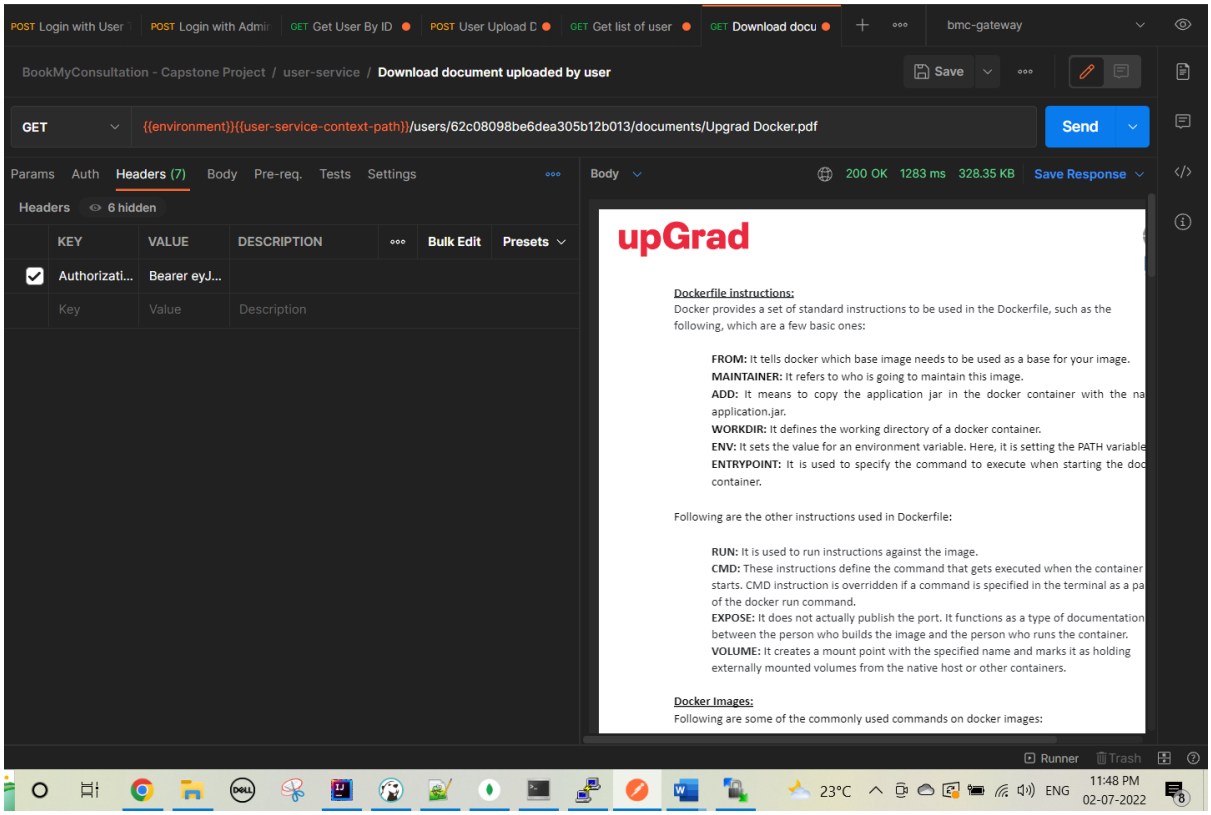
Invalid user id request



Upload user documents to s3: Success

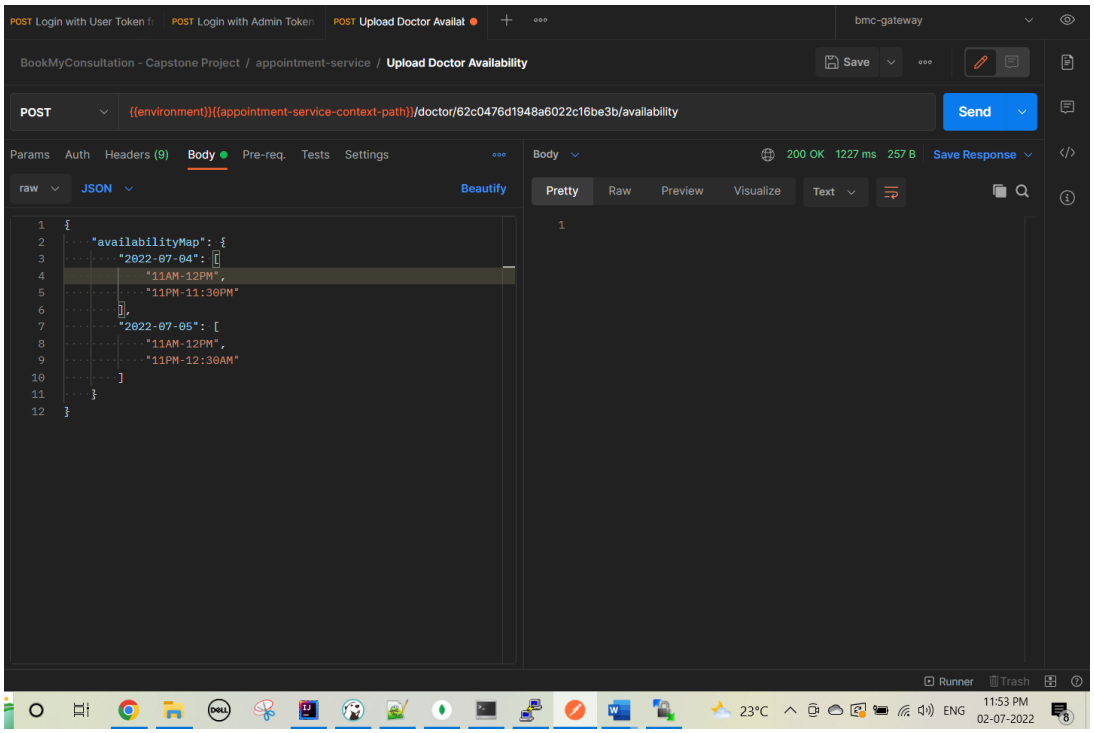


Get list of documents uploaded by user

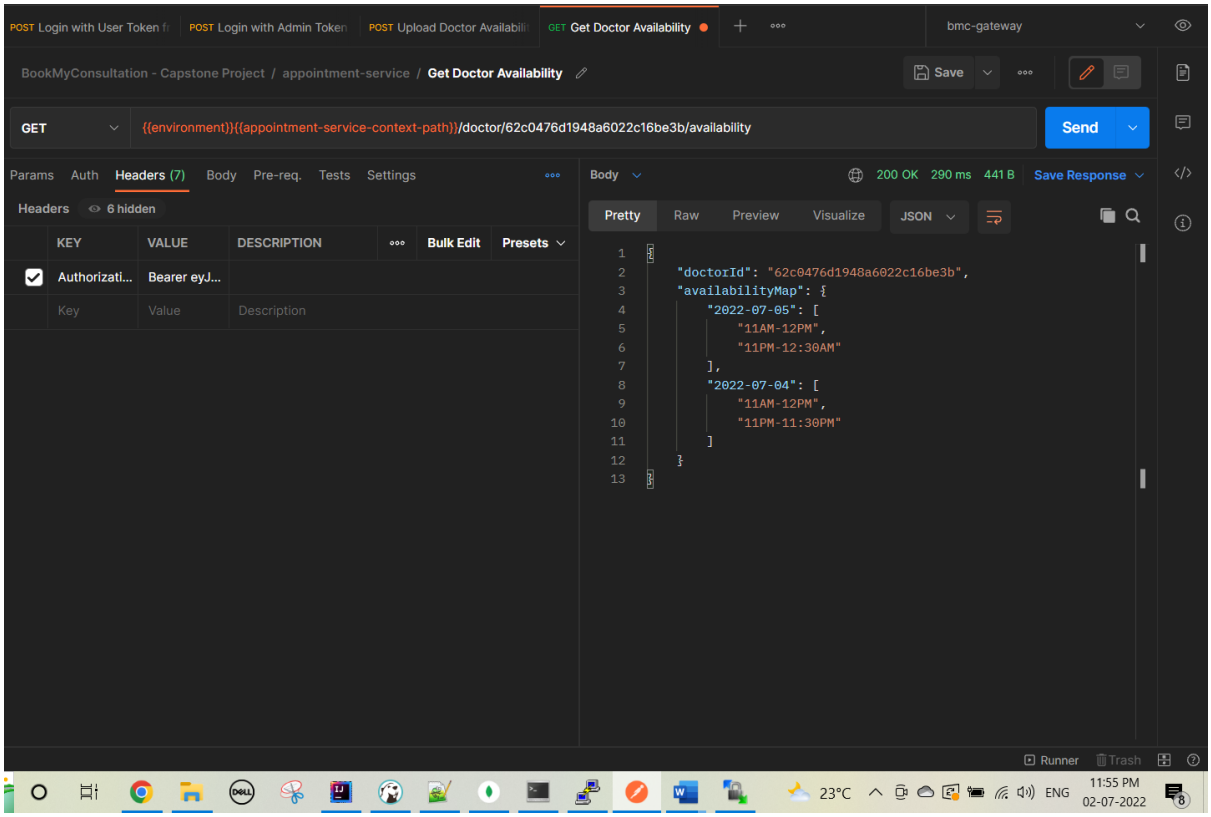


Download the document uploaded by user based on document name. (Make sure to give the full name including (.) extension.

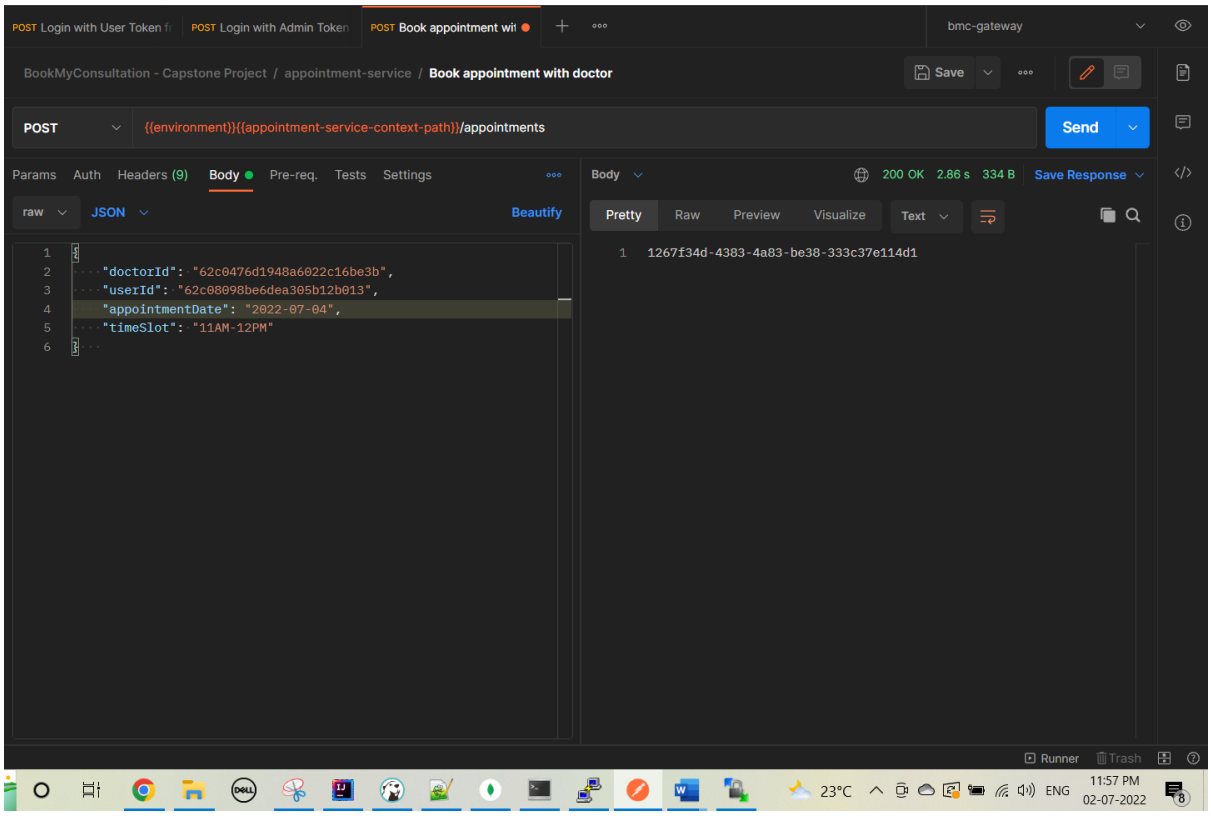
Appointment Service Request-Response:



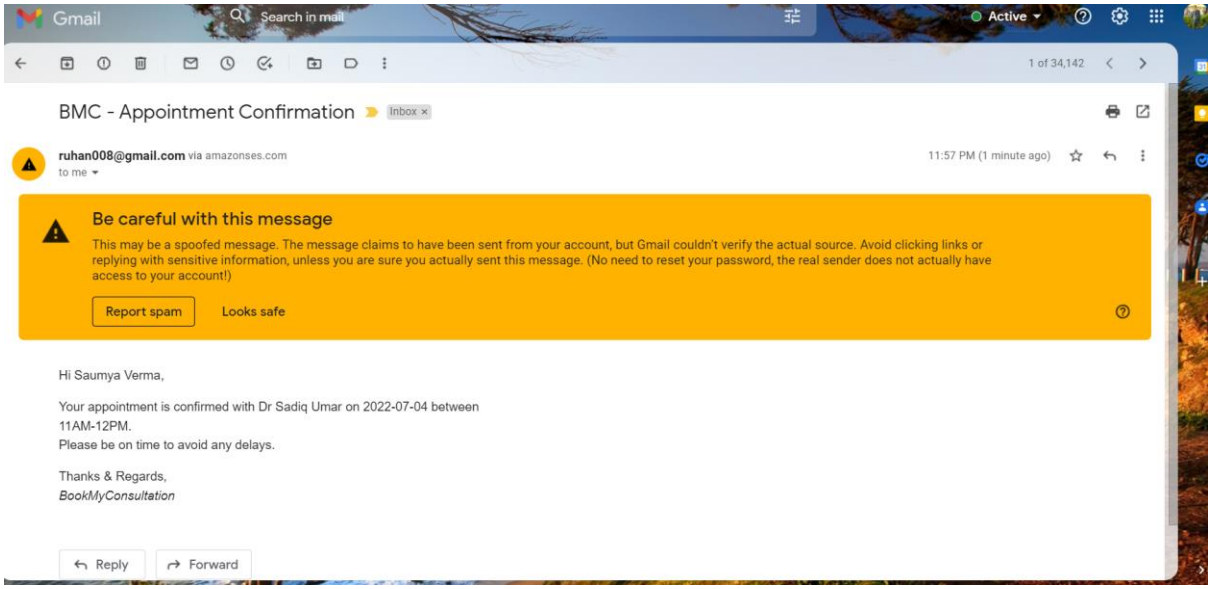
Update Doctor availability: Success



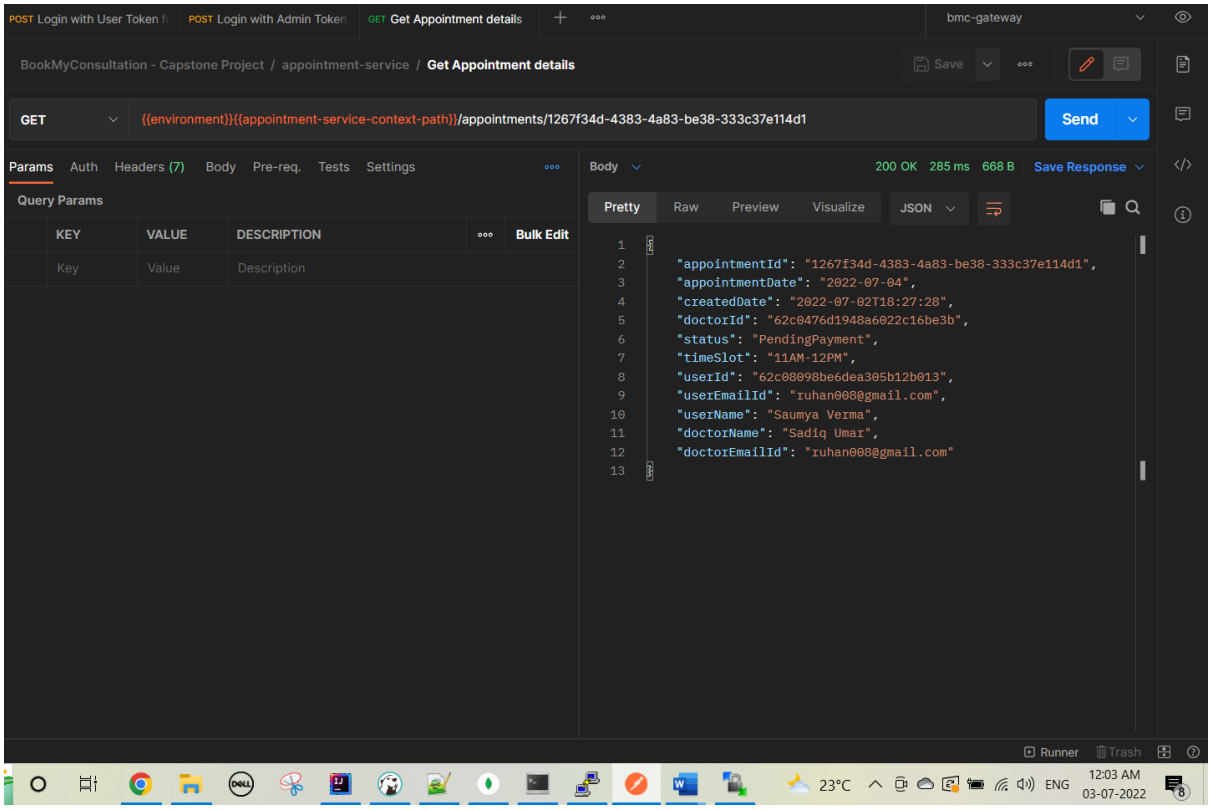
Get doctor availability based on ID



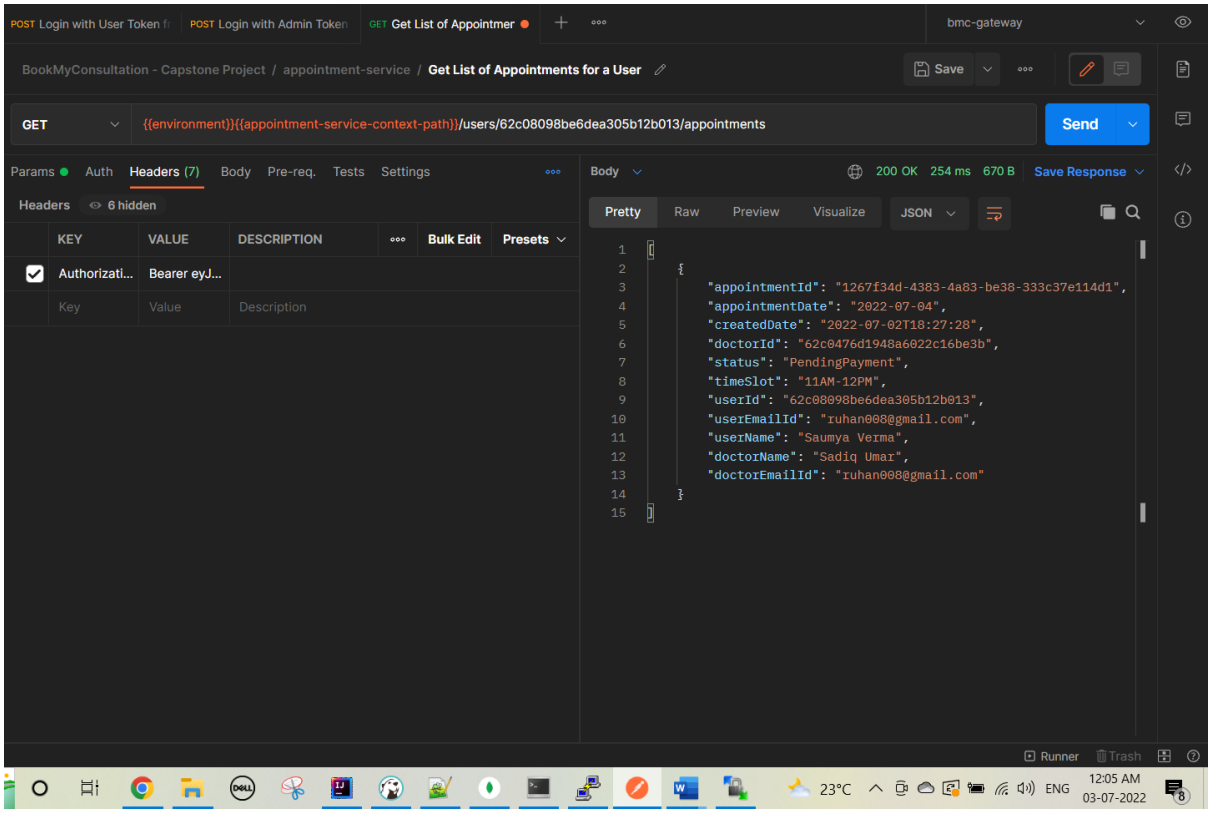
Book appointment with doctor: Success



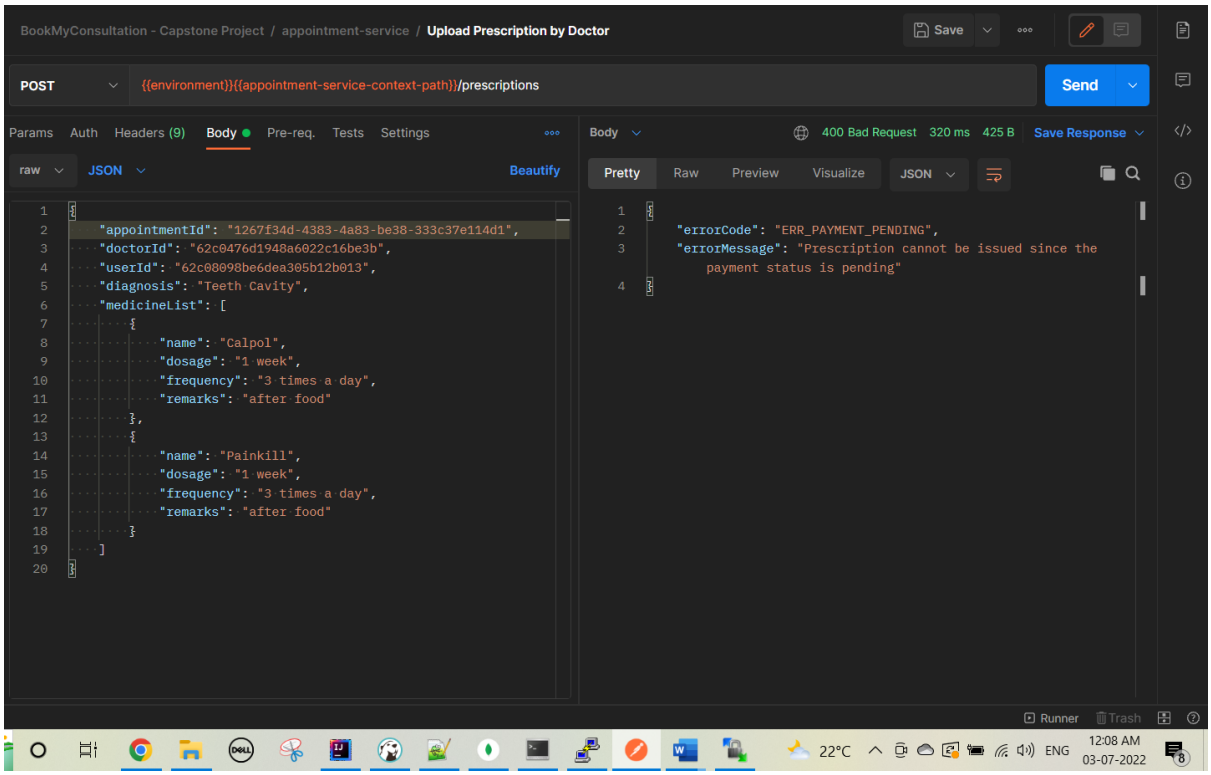
Email notification to user regarding appointment confirmation.



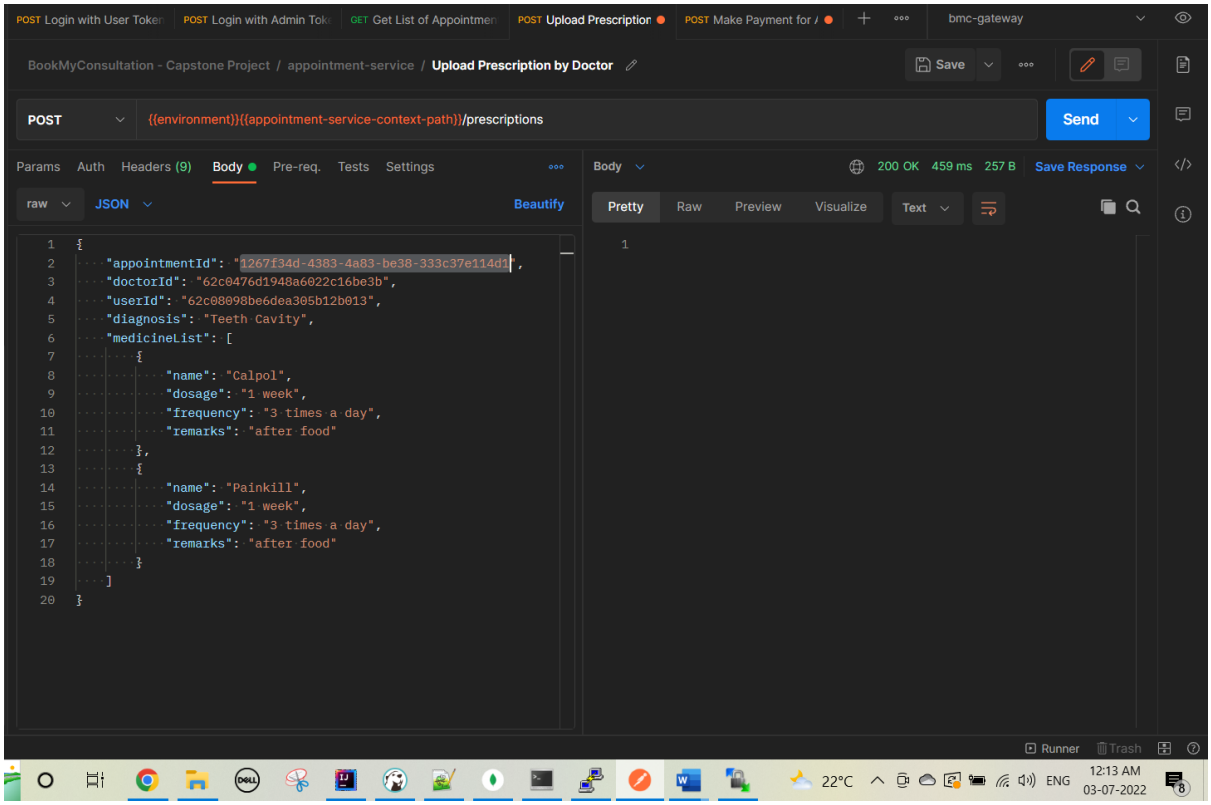
Get appointment details based on appointment id.



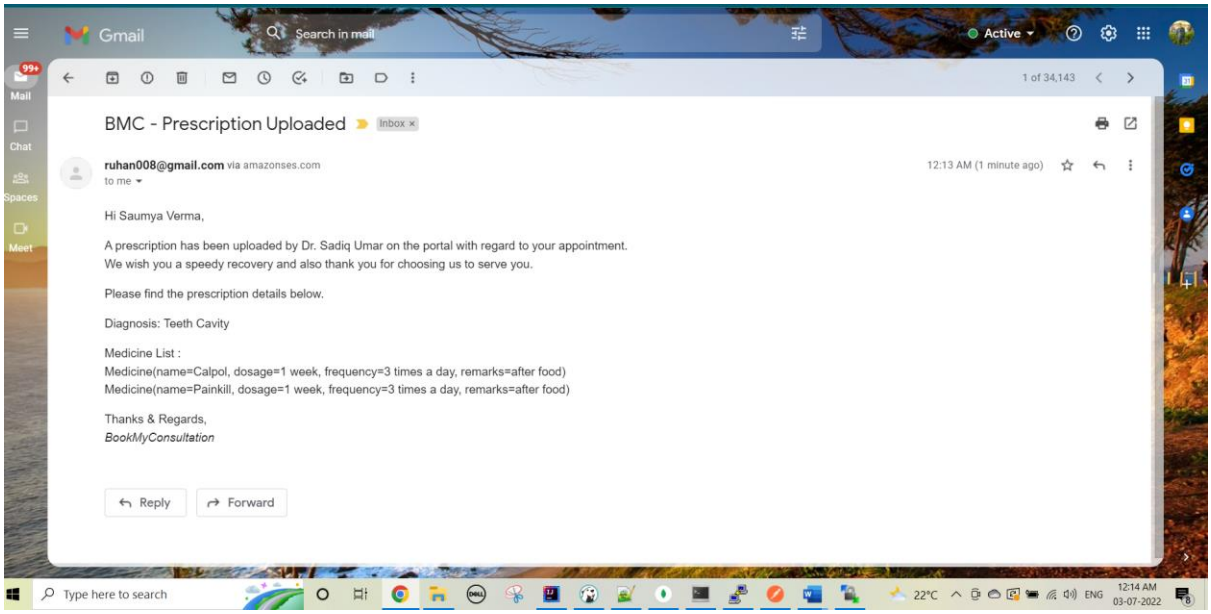
Get list of appointments booked by a user based on userId



Upload Prescription: Failure, error due to payment pending from patient side.

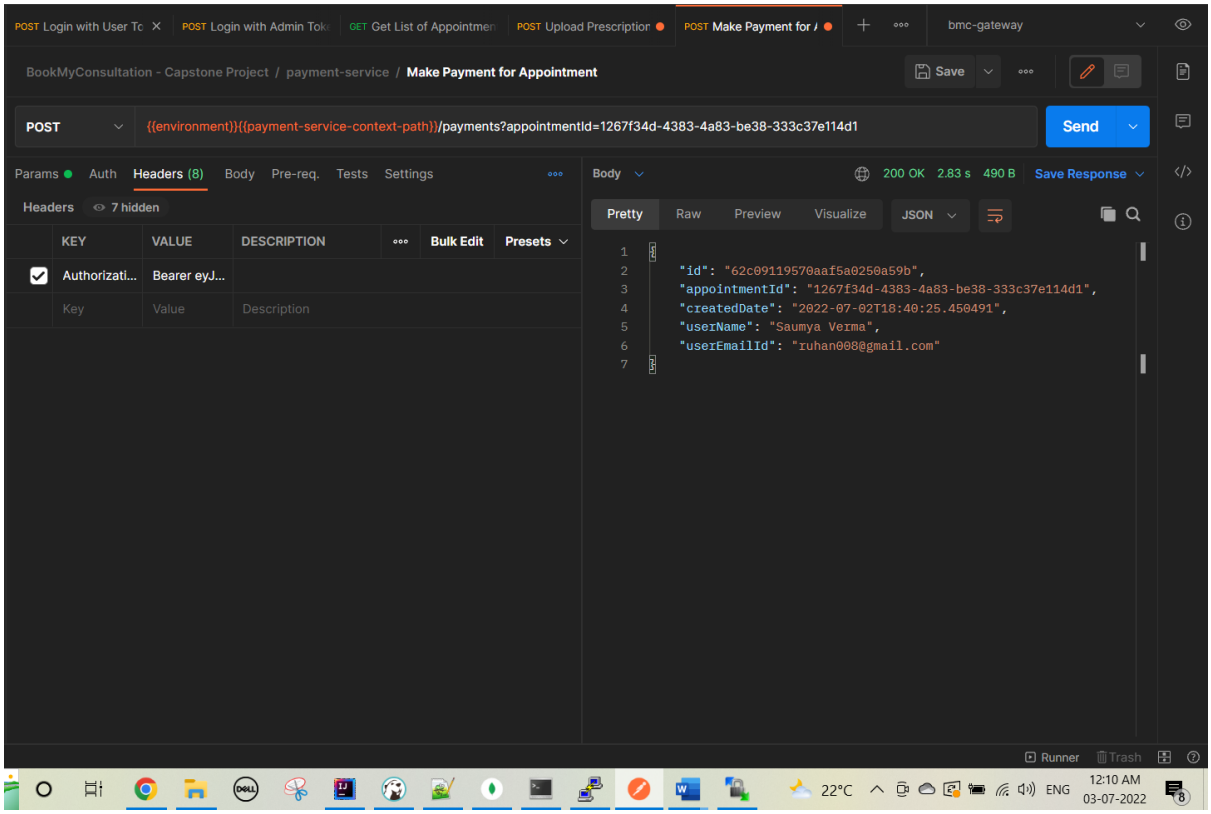


Upload prescription succeeds once user makes the payment and status is updated.

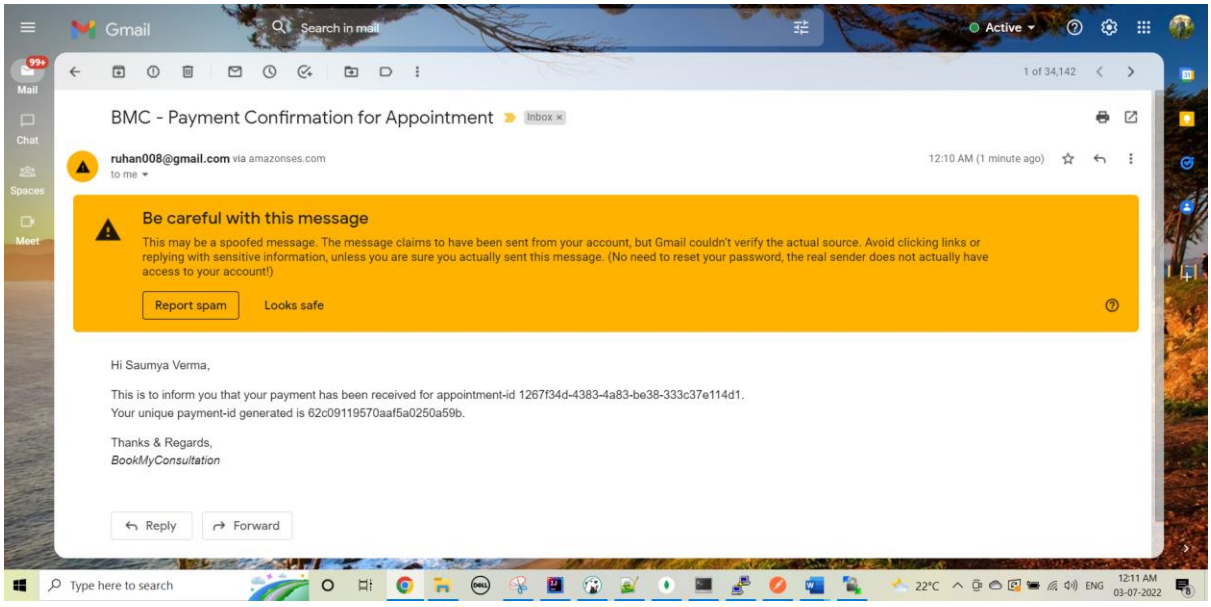


Prescription email to user after successful upload from doctor.

Payment Service Request-Response:

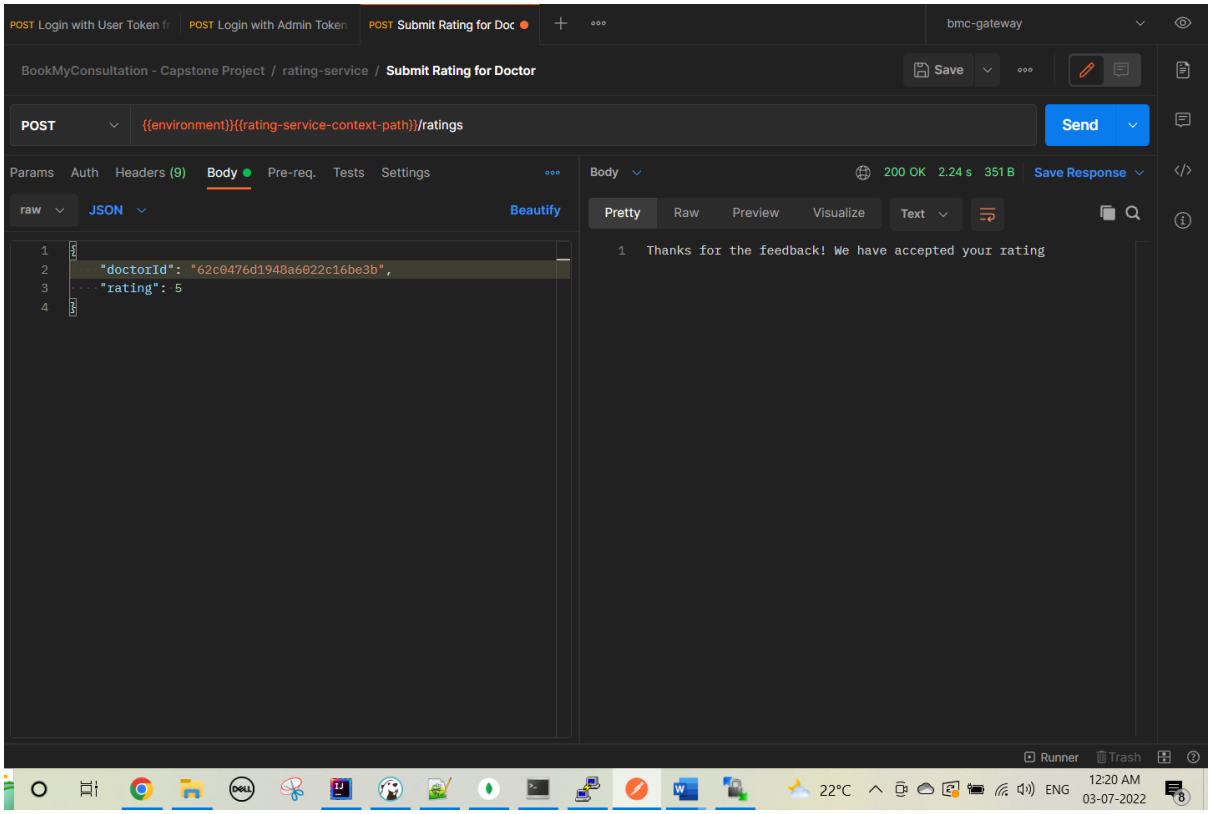


Payment for appointment – Success

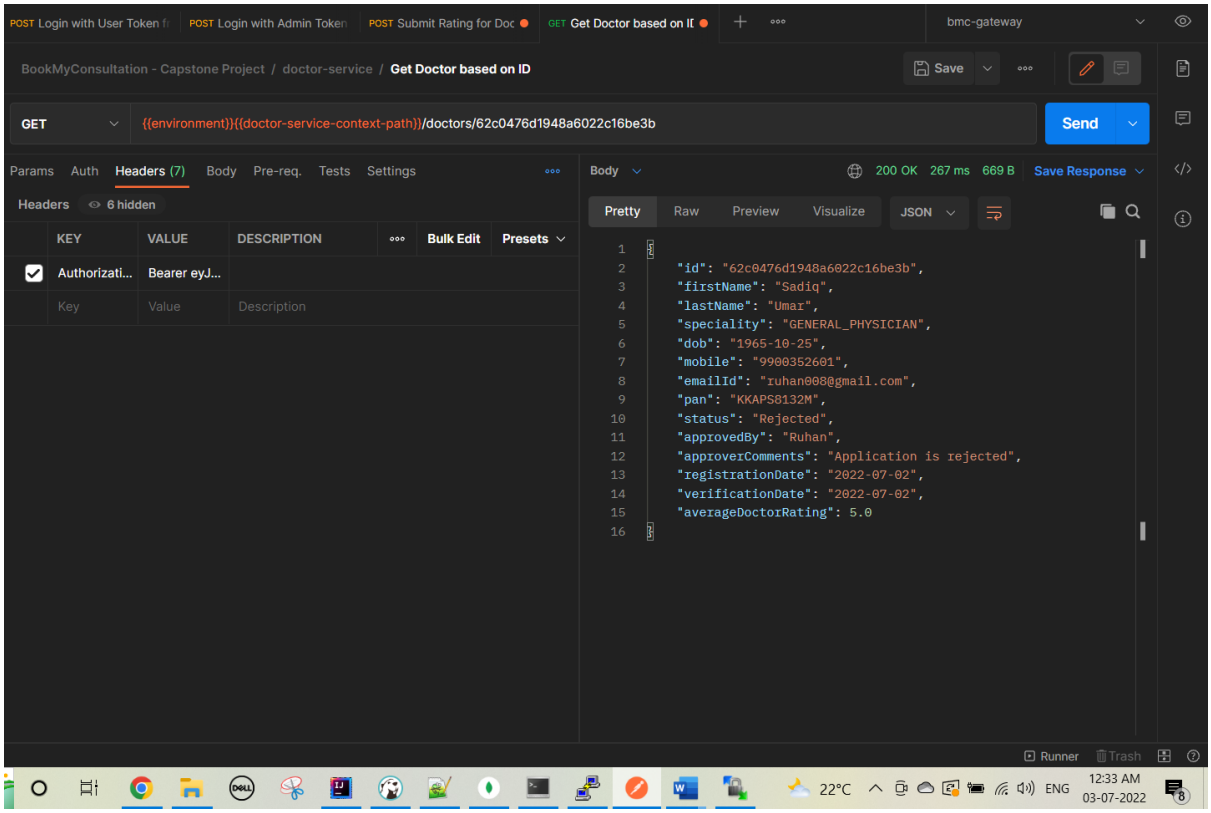


Payment email notification to user/patient

Rating Service Request-Response:

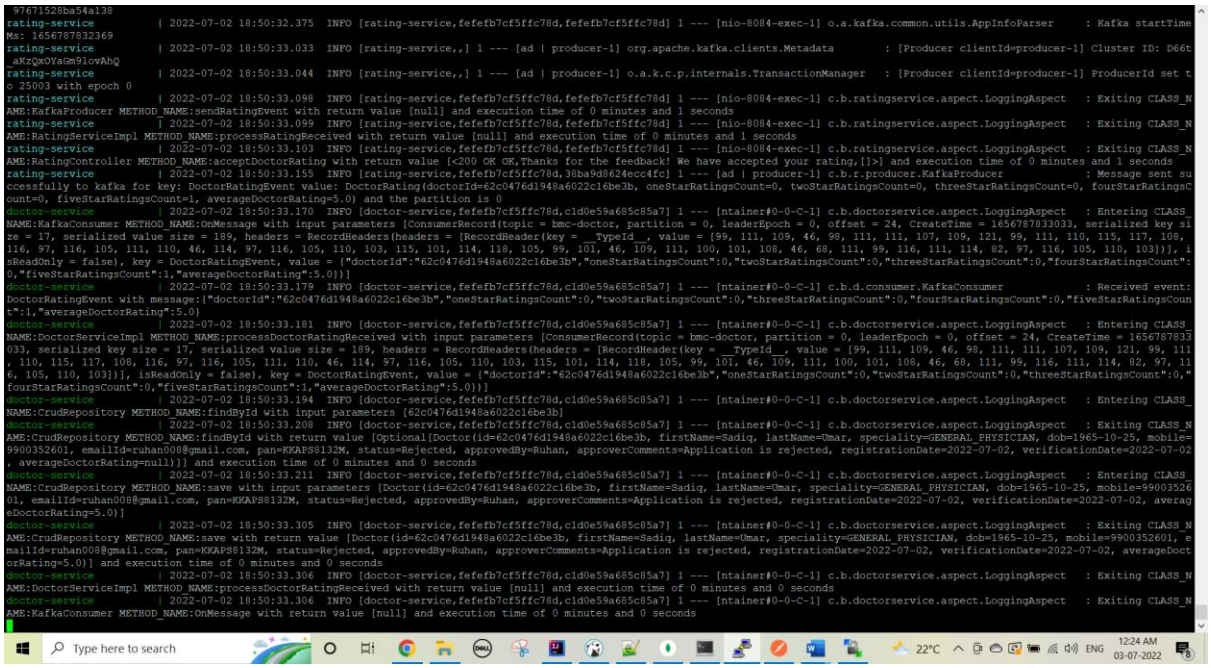


Submit rating for a doctor: Success



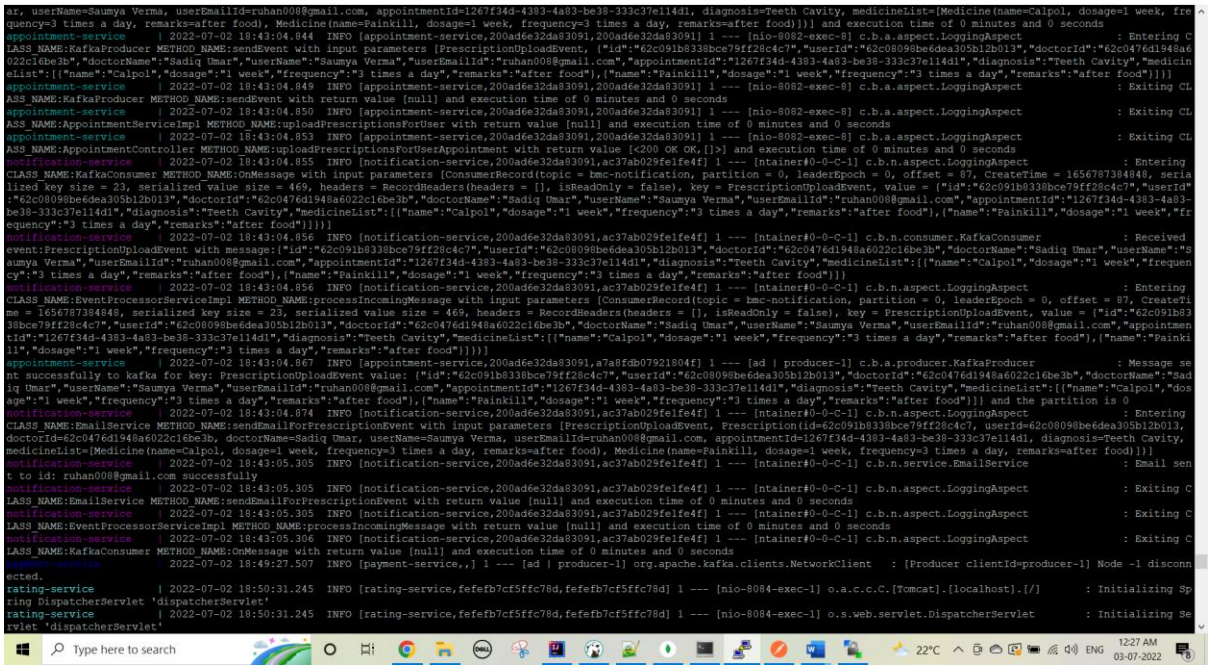
Doctor rating updated in response after processing the message received from bmc-doctor topic sent by rating-service.

Docker-Compose Logs:

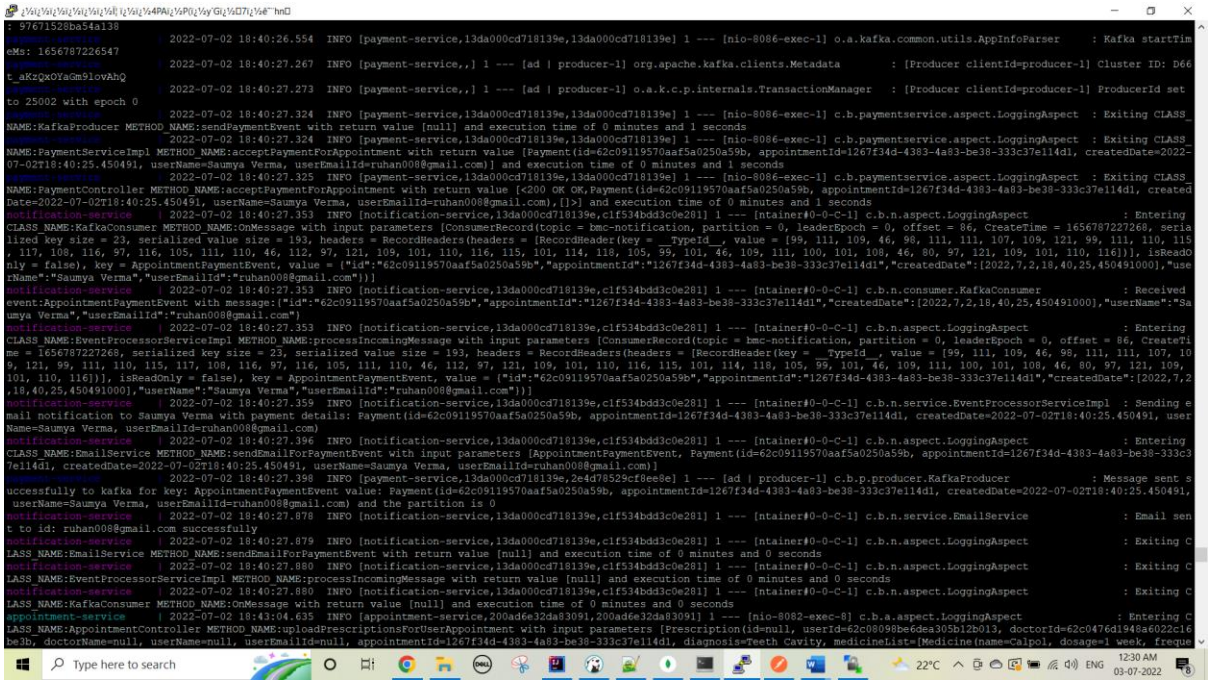


As you can see from the logs the doctor-service consumes the doctor rating message received from rating-service on bmc-doctor topic and updates the same in its repository.

Similar events happen with notification service when it receives different events for email notifications. Below are few log screenshots for the same:



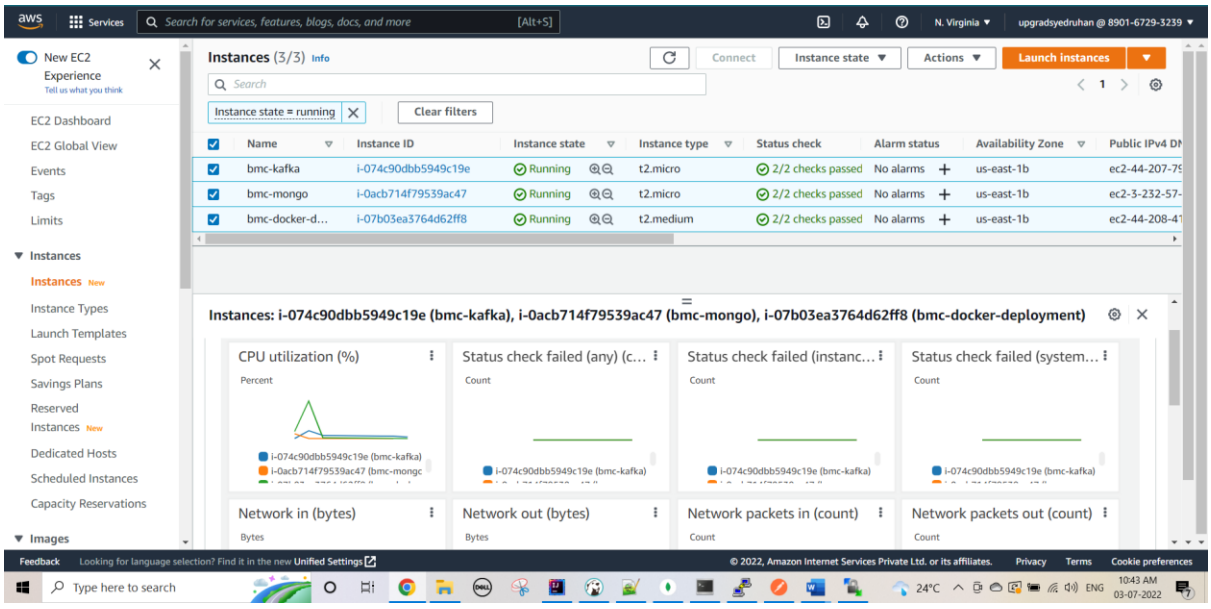
Prescription upload event



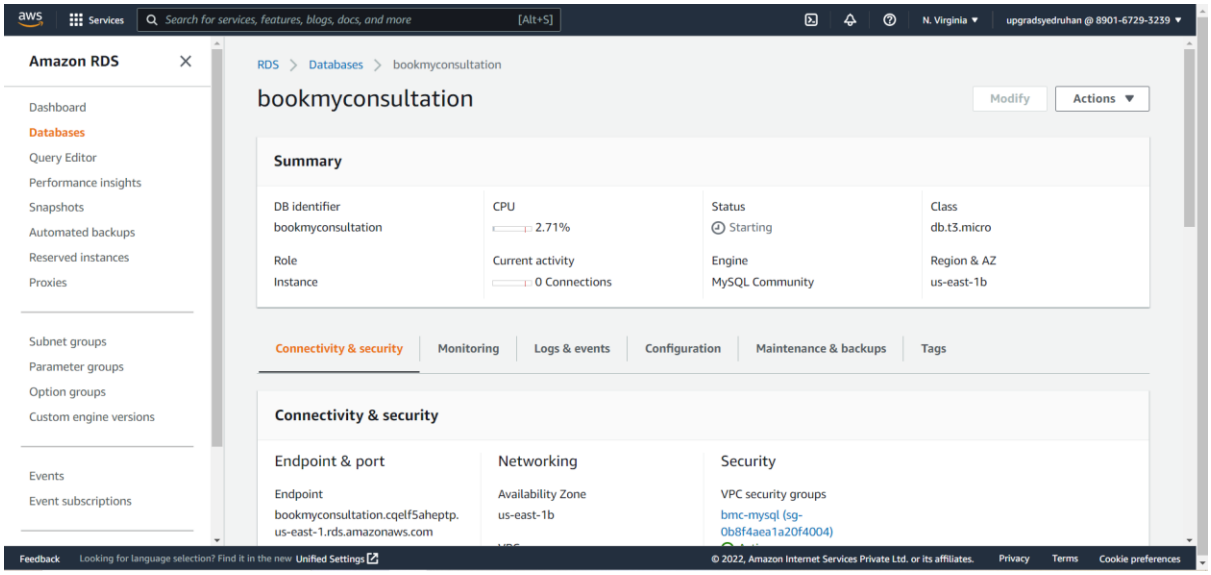
Payment Event

Kakfa, MongoDB, MySQL and Ubuntu Setup Screenshots:

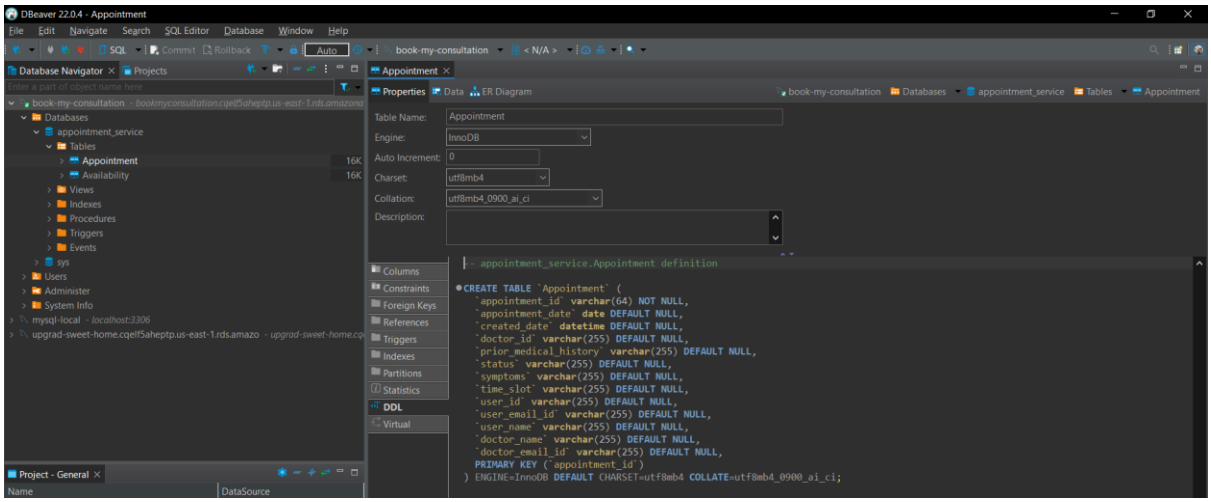
For Kafka, MongoDB and ubuntu I have created ec2 instances and installed kafka, MongoDB, Docker and Docker-Compose on them manually. As for MySQL I have used a RDS instance for the project. I did create a bookmyconsultation VPC to deploy all my resources under on VPC for project specifically. Please find the screenshots below:



EC-2 Instances bmc-kafka, bmc-mongo and bmc-docker-deployment



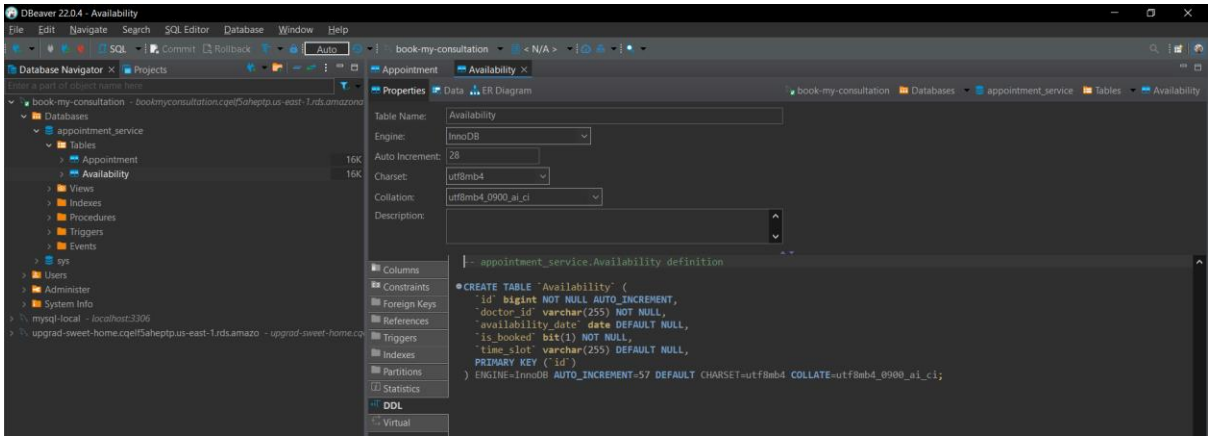
RDS MySQL Database instance



Appointment Table in MySQL

DDL:

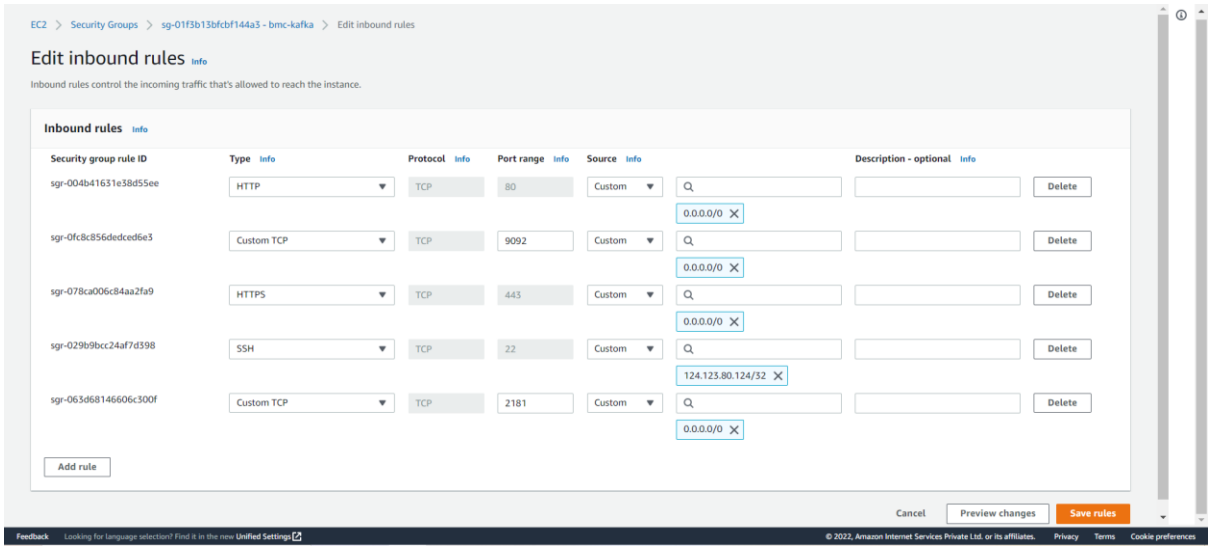
```
CREATE TABLE `Appointment` (
  `appointment_id` varchar(64) NOT NULL,
  `appointment_date` date DEFAULT NULL,
  `created_date` datetime DEFAULT NULL,
  `doctor_id` varchar(255) DEFAULT NULL,
  `prior_medical_history` varchar(255) DEFAULT NULL,
  `status` varchar(255) DEFAULT NULL,
  `symptoms` varchar(255) DEFAULT NULL,
  `time_slot` varchar(255) DEFAULT NULL,
  `user_id` varchar(255) DEFAULT NULL,
  `user_email_id` varchar(255) DEFAULT NULL,
  `user_name` varchar(255) DEFAULT NULL,
  `doctor_name` varchar(255) DEFAULT NULL,
  `doctor_email_id` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`appointment_id`)
)
```



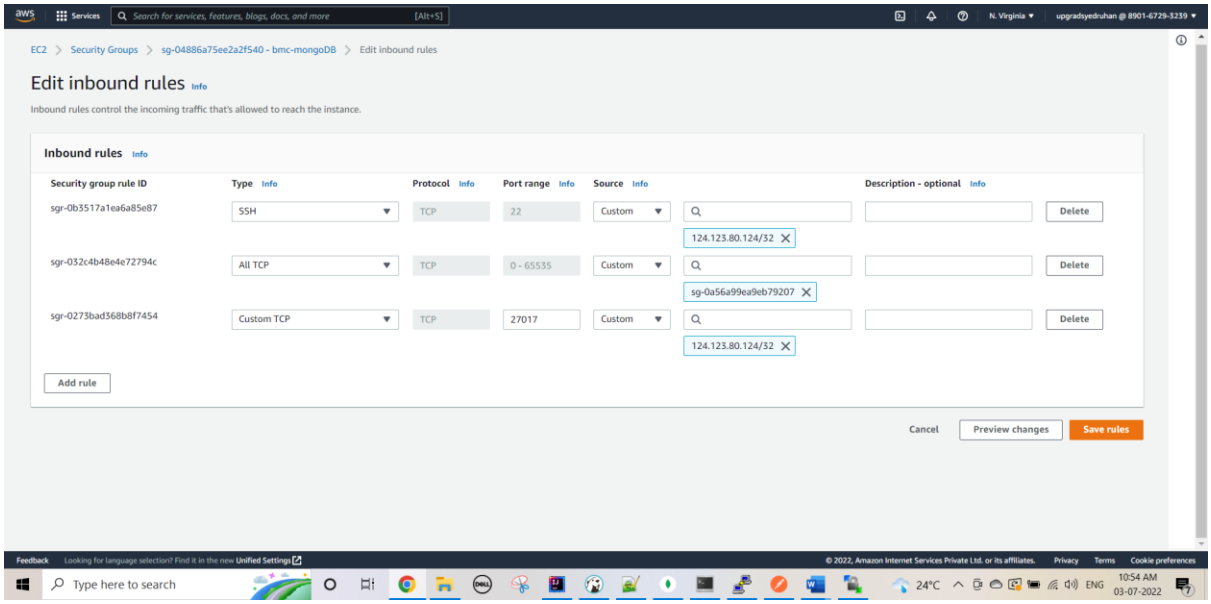
Availability Table in MySQL

DDL:

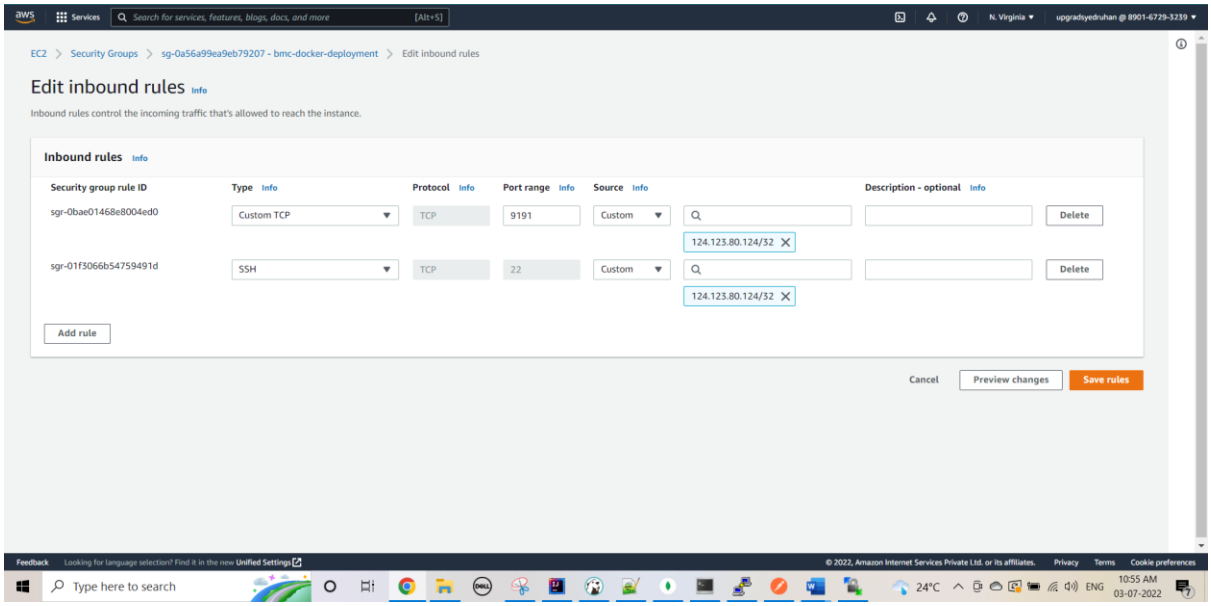
```
CREATE TABLE `Availability` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `doctor_id` varchar(255) NOT NULL,
  `availability_date` date DEFAULT NULL,
  `is_booked` bit(1) NOT NULL,
  `time_slot` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
   )
```



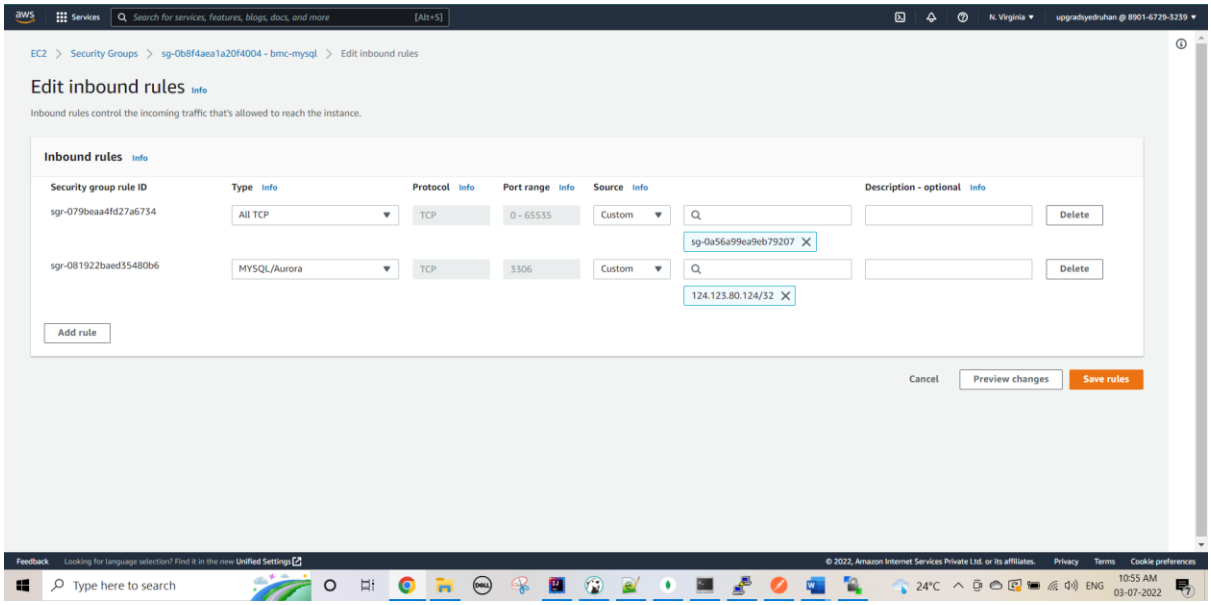
bmc-kafka security group



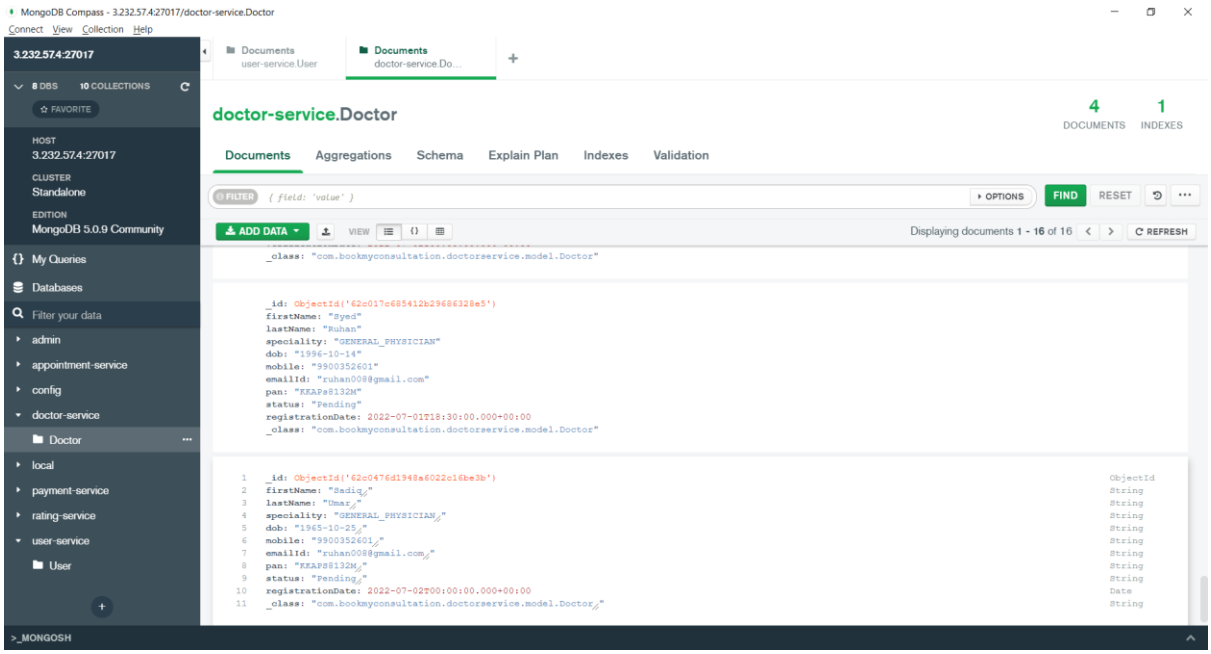
bmc-mongo security group



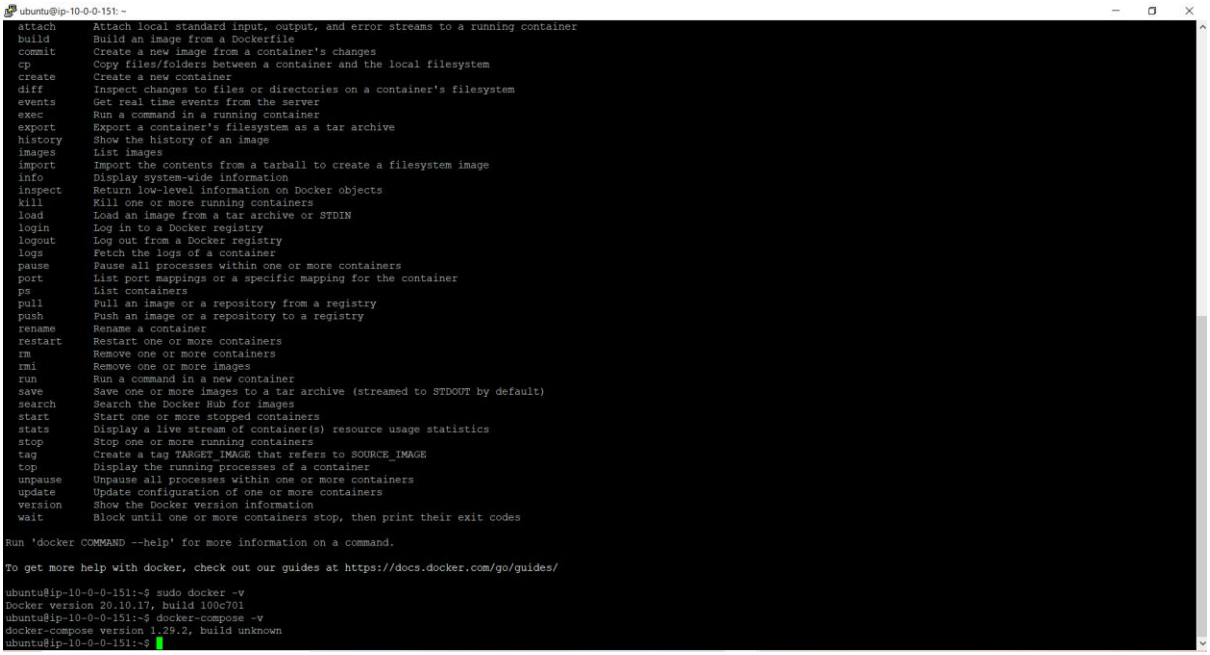
bmc-docker-deployment security group



bmc-mysql security group



MongoDB connected locally from MongoDBCompass



Docker and Docker-Compose setup on ubuntu

Future Enhancements: As part of future enhancements I have thought of the below implementation's for this project:

Monitoring Tools: As part of monitoring I will try to implement Prometheus, Grafana and Jaeger/Zipkin as part of the 3 pillars of monitoring, namely, Logging, Metrics Monitoring and distributed tracing.

Deployment: Deploy these services to on Kubernetes with all the previously mentioned monitoring tools.

Security: Implement Oauth2 as part of security to learn a new security implementation.

Final Note: I had a great time working on this project and learnt a lot of new things while working on these services and it was a fun journey to implement all the things I learnt through the course of this program.

I have tried to cover all the aspects of this project in terms of logic, design and config changes to be evaluated as mentioned in the evaluation guidelines, but in case of any issue or doubts while setting up the project please reach out to me on the below email ID or mobile number.

Email Id: ruhan008@gmail.com
Mobile: 9900352601

THANK YOU