Code Logic

This solution was approached using the **Externalized Kafka** and the **steps for code execution is present at the end of this document.**

1. Doctor Service

This micro service application is used

- to enroll new Doctor into BMC app.
- to approve/reject the registered Doctor.
- to send notification message to the Kafka message topic after the approval/rejection of the Registration.
- to upload/download the necessary documents by Doctor to support their registration.

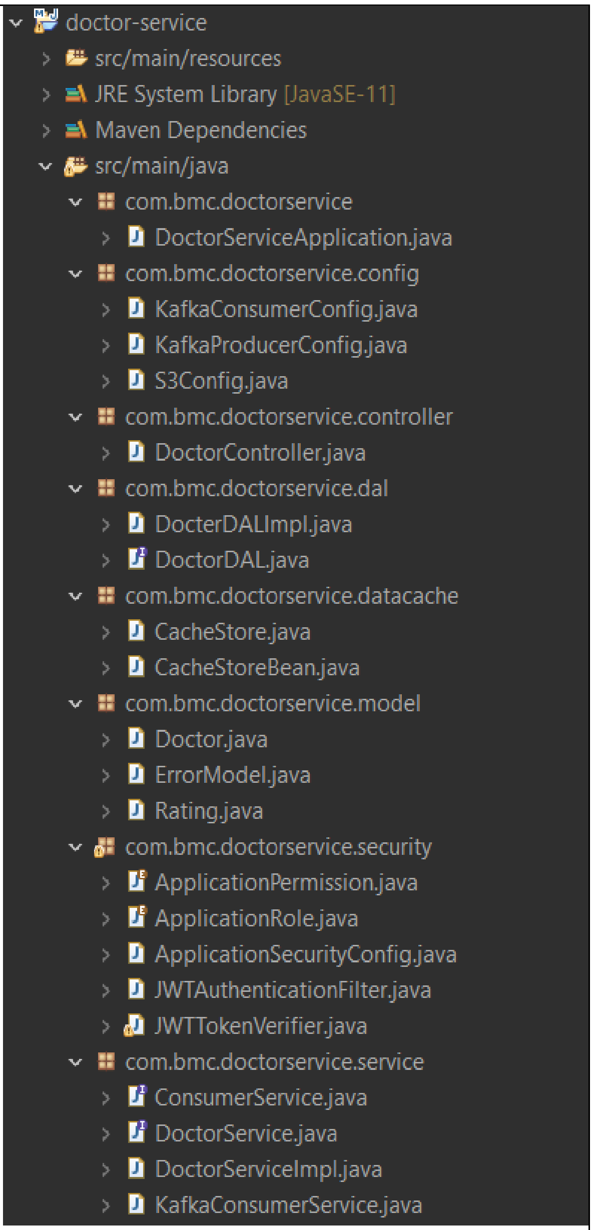
This Doctor micro service app has the following package structures.

- Security

- **ApplicationSecurityConfig** – Main class inside the security package which controls the configuration and security rules for different role (User & Admin) added to all end points supported by this doctor services.
- **JWTTokenVerifier –** This class checks if the Auth Token provided by the user is valid or not.
- **JWTAuthenticationFilter** – This class intercepts all incoming request and passes them through the **JWTTokenVerifier** & **ApplicationSecurityConfig** to verify Auth Token and check the security rule configured based on the roles respectively.
- **ApplicationRole** – This class defines two roles User & Admin for this microservice and grants the permissions and authorities for these two roles.

- Config

- **KafkaProducerConfig –** This class Creates the KafkaProducer object and configures the Kafka bootstrapServer to the KafkaProducer object, which is used to send asynchronous message to Kafka topic, when Doctor registration is approved/rejected.
- **KafkaConsumerConfig** – This class Creates the KafkaConsumer object and configures the Kafka bootstrapServer to the KafkaConsumer object, which is used to listen to Kafka topic incase any new Rating is provided to the Doctor via Rating service.
- **S3Config** – This class Creates the S3Client object and configures the AWS credentials to the



```
doctor-service
  src/main/resources
  JRE System Library [JavaSE-11]
  Maven Dependencies
  src/main/java
    com.bmc.doctorservice
      DoctorServiceApplication.java
    com.bmc.doctorservice.config
      KafkaConsumerConfig.java
      KafkaProducerConfig.java
      S3Config.java
    com.bmc.doctorservice.controller
      DoctorController.java
    com.bmc.doctorservice.dal
      DocterDALImpl.java
      DoctorDAL.java
    com.bmc.doctorservice.datacache
      CacheStore.java
      CacheStoreBean.java
    com.bmc.doctorservice.model
      Doctor.java
      ErrorModel.java
      Rating.java
    com.bmc.doctorservice.security
      ApplicationPermission.java
      ApplicationRole.java
      ApplicationSecurityConfig.java
      JWTAuthenticationFilter.java
      JWTTokenVerifier.java
    com.bmc.doctorservice.service
      ConsumerService.java
      DoctorService.java
      DoctorServiceImpl.java
      KafkaConsumerService.java
```

Code Logic

S3Client object, which is used to interact with AWS S3 services to upload and download the files.

- Controller

- **DoctorController –** Main controller class, which acts the entry point for the Doctor Service. It contains the methods to handle the incoming request and the methods for handling the exception when invalid input is provided by the user and reaches out to Service layer for further processing.

- Service

- **DoctorService & DoctorServiceImpl –** The Service package has one interface and its implementation class; this service reaches out to
  - ➢ DAL layer to access the objects from Mongo DB,
  - ➢ **Sends the asynchronous message** to the **Kafka topic** when the Doctor registration is either approved/rejected.
- **ConsumerService & KafkaConsumerService –** The Service package has one interface and its implementation class: this service class
  - ➢ **Listens to the Kafka topic** in case any new Rating is provided to the Doctor by a user via Rating service.

- DAL

- **DoctorDAL & DoctorDALImpl –** The Data Access Layer (DAL) package has one interface and its implementation class; this DAL classes uses the **MongoTemplate** class to perform the necessary CRUD operation on the Doctor object in MongoDB.

Model

- **Doctor** class acts as the entity to interact with Doctor collection in MongoDB.
- **ErrorModel** contains the body for the proper error message to be displayed to the user.
- **Rating** model for the rating object when it is captured from Kafka Topic.

- DataCache

- **CacheStore & CacheStoreBean –** These Classes creates the CacheStore to store the Doctor objects temporarily in RAM layer to access the objects faster.

Code Logic

2. <u>User Service</u>

This micro service application is used

- to enroll new User into BMC app.
- to send notification message to the Kafka message topic for User Verification email.
- to upload/download the necessary documents needed by User.

This User micro service app has the following package structures.

- Security

Same as in Doctor Services.

- Config

- **KafkaProducerConfig –** This class Creates the KafkaProducer object and configures the Kafka bootstrapServer to the KafkaProducer object, which is used to send asynchronous message to Kafka topic for User verification email.
- **S3Config –** This class Creates the S3Client object and configures the AWS credentials to the S3Client object, which is used to interact with AWS S3 services to upload and download the files.

- Controller

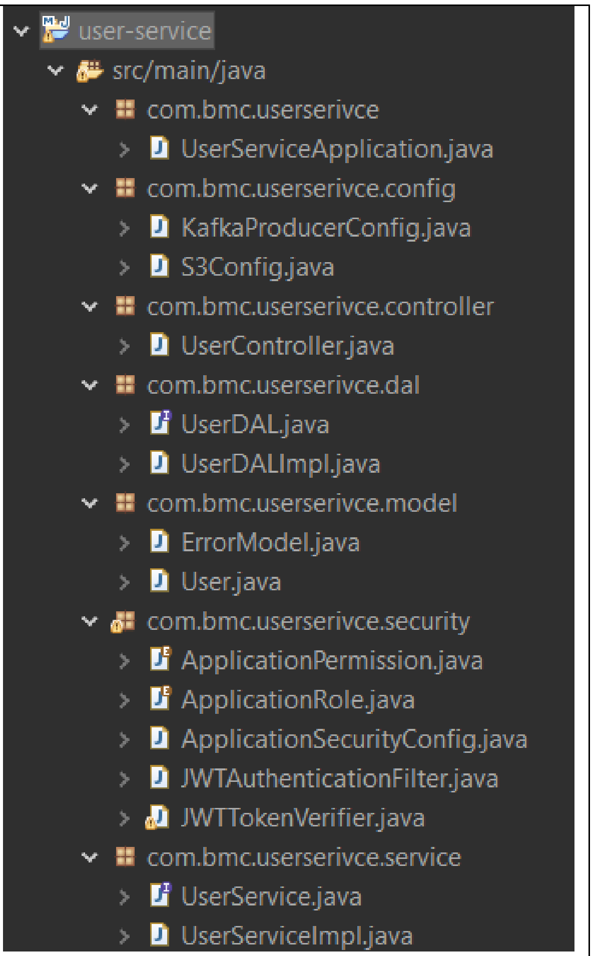
- **UserController –** Main controller class, which acts the entry point for the User Service. It contains the methods to handle the incoming request and the methods for handling the exception when invalid input is provided by the user.

- Service

- **UserService & UserServiceImpl –** The Service package has one interface and its implementation class; this service reaches out to
  - ➢ DAL layer to access the objects from Mongo DB,
  - ➢ **Sends the asynchronous message** to the **Kafka topic** to send the verification email to User.

- DAL

- **UserDAL & UserDALImpl –** The Data Access Layer (DAL) package has one interface and its implementation class; this DAL classes uses the **MongoTemplate** class to perform the necessary CRUD operation on the User object in MongoDB.



```
user-service
  src/main/java
    com.bmc.userserivce
      UserServiceApplication.java
    com.bmc.userserivce.config
      KafkaProducerConfig.java
      S3Config.java
    com.bmc.userserivce.controller
      UserController.java
    com.bmc.userserivce.dal
      UserDAL.java
      UserDALImpl.java
    com.bmc.userserivce.model
      ErrorModel.java
      User.java
    com.bmc.userserivce.security
      ApplicationPermission.java
      ApplicationRole.java
      ApplicationSecurityConfig.java
      JWTAuthenticationFilter.java
      JWTTokenVerifier.java
    com.bmc.userserivce.service
      UserService.java
      UserServiceImpl.java
```

Code Logic

Model

- **User** class acts as the entity to interact with User Collection in MongoDB.
- **ErrorModel** contains the body for the proper error message to be displayed to the user.

3. Appointment Service

This micro service application is used

- to publish the Availability of the Doctor.
- to book Appointments with Doctors by users
- to send Prescription
- to send message to the Kafka topic for the appointment confirmation & Prescription email.

This User micro service app has the following package structures.

- Security

Same as in Doctor Services.

- Config

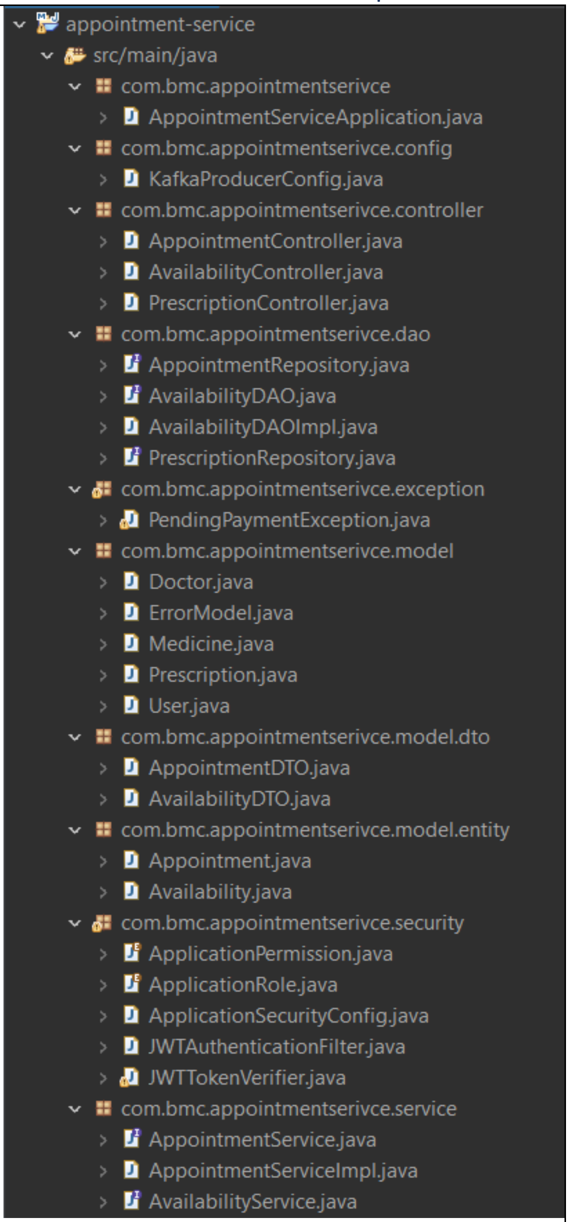
- **KafkaProducerConfig –** This class Creates the KafkaProducer object and configures the Kafka bootstrapServer to the KafkaProducer object, which is used to send asynchronous message to Kafka topic for User verification email.

- Controller

- **AvailabilityController –** Main controller class, which acts the entry point for the publishing the Availabilities of Doctor. It contains the methods to handle the API request and the methods for handling the exception when invalid input is provided by the user and reaches out to Service layer for further processing.

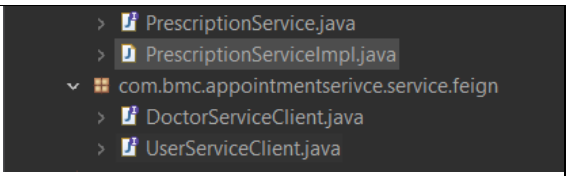
- **AppointmentController –** Controller class, which acts the entry point for the Appointment services. It contains the methods to handle the incoming request and reaches out to Service layer for further processing.

- **PrescriptionController –** Controller class, which acts the entry point for the Prescription services. It contains the methods to handle the incoming request and reaches out to Service layer for further processing.



```
appointment-service
  src/main/java
    com.bmc.appointmentserivce
      AppointmentServiceApplication.java
    com.bmc.appointmentserivce.config
      KafkaProducerConfig.java
    com.bmc.appointmentserivce.controller
      AppointmentController.java
      AvailabilityController.java
      PrescriptionController.java
    com.bmc.appointmentserivce.dao
      AppointmentRepository.java
      AvailabilityDAO.java
      AvailabilityDAOImpl.java
      PrescriptionRepository.java
    com.bmc.appointmentserivce.exception
      PendingPaymentException.java
    com.bmc.appointmentserivce.model
      Doctor.java
      ErrorModel.java
      Medicine.java
      Prescription.java
      User.java
    com.bmc.appointmentserivce.model.dto
      AppointmentDTO.java
      AvailabilityDTO.java
    com.bmc.appointmentserivce.model.entity
      Appointment.java
      Availability.java
    com.bmc.appointmentserivce.security
      ApplicationPermission.java
      ApplicationRole.java
      ApplicationSecurityConfig.java
      JWTAuthenticationFilter.java
      JWTTokenVerifier.java
    com.bmc.appointmentserivce.service
      AppointmentService.java
      AppointmentServiceImpl.java
      AvailabilityService.java
```

Code Logic

- Service • **AvailabilityService & AvailabilityServiceImpl –** The Availability Service package has one interface and its implementation class; this service reaches out to ➢ DAL layer to access the objects from MY_SQL Database. ➢ Reaches out to Doctor and User services to verify the correctness of the Doctor and User id provided as input. • **AppointmentService & AppointmentServiceImpl –** The Appointment Service package has one interface and its implementation class; this service reaches out to ➢ DAL layer to access the objects from MY_SQL Database. ➢ **Sends the asynchronous message** to the **Kafka topic** to send the appointment confirmation email to User. • **PrescriptionService & PrescriptionServiceImpl –** The Prescription Service package has one interface and its implementation class; this service reaches out to ➢ DAL layer to access the objects from MY_SQL Database. ➢ **Sends the asynchronous message** to the **Kafka topic** to send the prescription email to User. • **DoctorServiceClient & UserServiceClient –** These two interfaces annotated with ***@FeignClient*** which reaches to Doctor and User service respectively to check their ids and fetches their details like name and Email address. **- DAL** • **AvailabilityDAO & AvailabilityDAOImpl –** The Data Access Layer (DAL) package has one interface and its implementation class; this DAL classes uses the **EntityManager** class to perform the necessary CRUD operation on the Availability object in MY-SQL DB. • **AppointmentRepositry –** This interface in DAO layer extends the JPARepositry to perform the necessary CRUD operation on the Appointment object in in MY-SQL DB. • **PrescriptionRepositry –** This interface in DAO layer extends the MongoRepositry to perform the	 PrescriptionService.java PrescriptionServiceImpl.java com.bmc.appointmentserivce.service.feign DoctorServiceClient.java UserServiceClient.java

necessary CRUD operation on the Appointment object in in Mongo DB.

Model

- **Appointment** class acts the entity to interact with Appointment table in MY-SQL DB.
- **Availability** class acts the entity to interact with Availability table MY-SQL DB.
- **Prescription has list of Medicine class objects as one of its variables** which interacts with Prescription collection in MongoDB.

4. Payment Service

This micro service application is used

- to make Payment for the used appointment with Doctor.

This Payment micro service app has the following package structures.

- Security

Same as in Doctor Services.

- Config

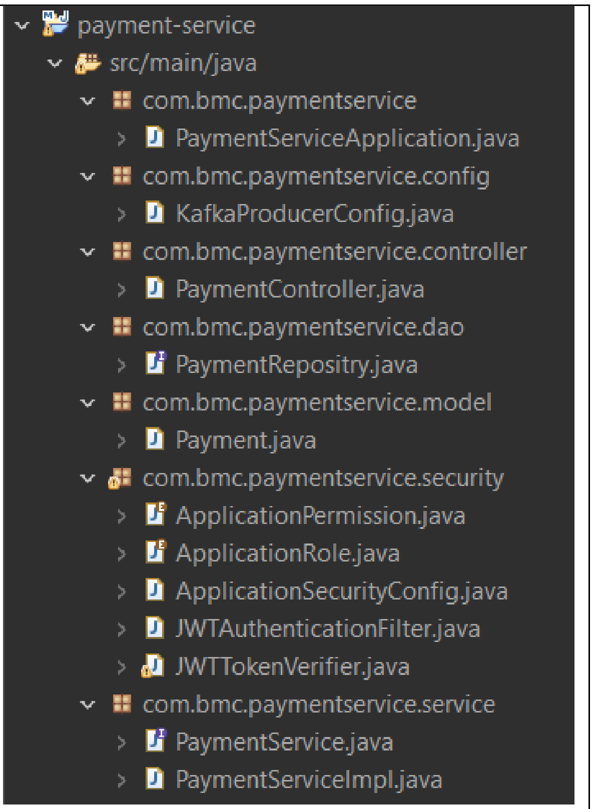
- **KafkaProducerConfig –** This class Creates the KafkaProducer object and configures the Kafka bootstrapServer to the KafkaProducer object, which is used to send asynchronous message to Kafka topic for User verification email.

- Controller

- **PaymentController –** Main controller class, which acts the entry point for the Payment Service. It contains the methods to handle the incoming request and the methods for handling the exception when invalid input is provided by the user.

- Service

- **PaymentService & PaymentServiceImpl –** The Service package has one interface and its implementation class; this service reaches out to
  - ➢ DAL layer to access the objects from Mongo DB.
  - ➢ Reaches out to Appointment services and update the Appointment status as Confirmed.
  - ➢ **Sends the asynchronous message** to the **Kafka topic** after the Payment is made.



```
payment-service
  src/main/java
    com.bmc.paymentservice
      PaymentServiceApplication.java
    com.bmc.paymentservice.config
      KafkaProducerConfig.java
    com.bmc.paymentservice.controller
      PaymentController.java
    com.bmc.paymentservice.dao
      PaymentRepositry.java
    com.bmc.paymentservice.model
      Payment.java
    com.bmc.paymentservice.security
      ApplicationPermission.java
      ApplicationRole.java
      ApplicationSecurityConfig.java
      JWTAuthenticationFilter.java
      JWTTokenVerifier.java
    com.bmc.paymentservice.service
      PaymentService.java
      PaymentServiceImpl.java
```

Code Logic

- DAO

- **PaymentRepositry –** This interface in DAO layer extends the **MongoRepositry** to perform the necessary CRUD operation on the Payment object in in Mongo DB.

Model

- **Payment** class acts as the entity to interact with Payment Collection in MongoDB.
- **ErrorModel** contains the body for the proper error message to be displayed to the user.

5. Rating Service

This micro service application is used

- to provide the Ratings for the Doctor.

This Rating micro service app has the following package structures.

- Security

Same as in Doctor Services.

- Config

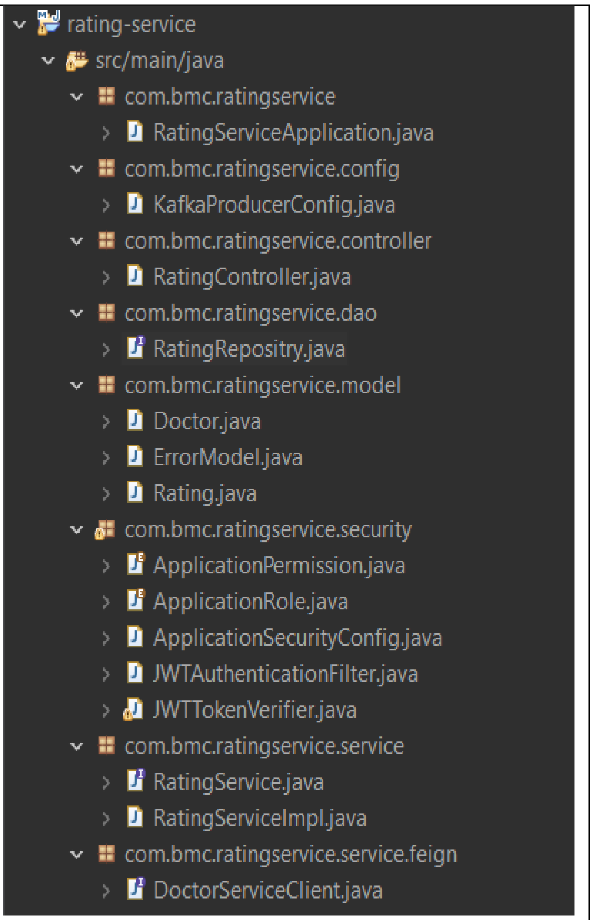
- **KafkaProducerConfig –** This class Creates the KafkaProducer object and configures the Kafka bootstrapServer to the KafkaProducer object, which is used to send asynchronous message to Kafka topic for User verification email.

- Controller

- **RatingController –** Main controller class, which acts the entry point for the Rating Service. It contains the methods to handle the incoming request and the methods for handling the exception when invalid input is provided by the user.

- Service

- **RatingService, RatingServiceImpl –** The Service package has one interface and its implementation class; this service reaches out to
  - ➢ DAL layer to access the objects from Mongo DB.
  - ➢ **Sends the asynchronous message** to the **Kafka topic** after the Rating is provided for the doctor.
- **DoctorServiceClient –** This interface annotated with ***@FeignClient*** and it reaches out to Doctor services and update the Ratings of the Doctor in the Doctor collection in MongoDB.



```
rating-service
  src/main/java
    com.bmc.ratingservice
      RatingServiceApplication.java
    com.bmc.ratingservice.config
      KafkaProducerConfig.java
    com.bmc.ratingservice.controller
      RatingController.java
    com.bmc.ratingservice.dao
      RatingRepositry.java
    com.bmc.ratingservice.model
      Doctor.java
      ErrorModel.java
      Rating.java
    com.bmc.ratingservice.security
      ApplicationPermission.java
      ApplicationRole.java
      ApplicationSecurityConfig.java
      JWTAuthenticationFilter.java
      JWTTokenVerifier.java
    com.bmc.ratingservice.service
      RatingService.java
      RatingServiceImpl.java
    com.bmc.ratingservice.service.feign
      DoctorServiceClient.java
```

Code Logic

- DAO

• **RatingRepositry**

– This interface in DAO layer extends the **MongoRepositry** to perform the necessary CRUD operation on the Rating object in in Mongo DB.

Model

• **Rating** class acts as the entity to interact with Rating Collection in MongoDB.

• **ErrorModel** contains the body for the proper error message to be displayed to the user.

6. Security Service

This micro service application is used

• to generate the JWT Token by providing the correct username and password.

This Security micro service app has the following package structures.

- Security

• **ApplicationPermission –** This class defines two App permission BMC_USER and BMC_ADMIN.

• **ApplicationRole –** This class defines two App roles USER and ADMIN and configures the permission BMC_USER and BMC_ADMIN to them respectively.

• **JWTAuthenticationFilter –** This class intercepts the incoming request and checks the user name and password and after successful authentication generates the JWT Token and adds it to the response header.

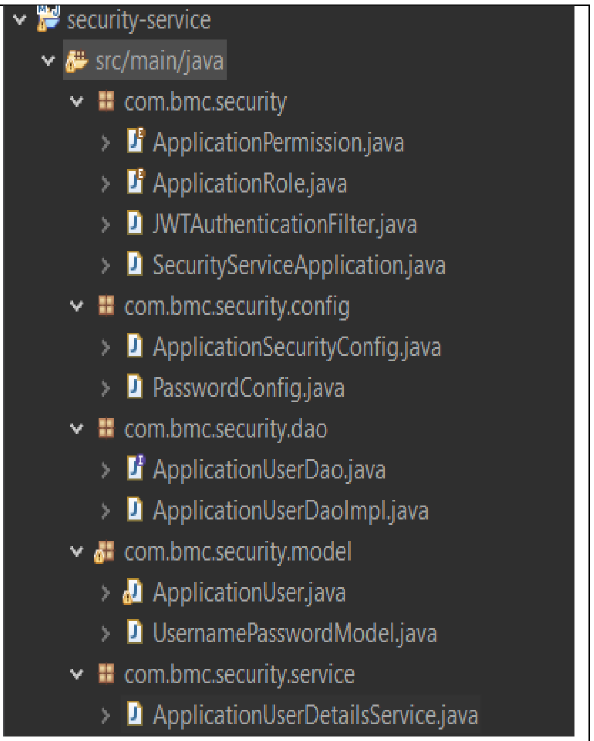
- Config

• **ApplicationSecurityConfig –** Main class inside the security package which controls the configuration and security rules for different role (User & Admin) added to all end points supported by this security services.

• **Password –** This class returns the password in the encrypted format.

- Service

• **ApplicationUserDetailsService –** This class in service layer implements the UserDetailsService interface and reaches out to DAO layer to fetch the username and password of the user.



```
security-service
  src/main/java
    com.bmc.security
      ApplicationPermission.java
      ApplicationRole.java
      JWTAuthenticationFilter.java
      SecurityServiceApplication.java
    com.bmc.security.config
      ApplicationSecurityConfig.java
      PasswordConfig.java
    com.bmc.security.dao
      ApplicationUserDao.java
      ApplicationUserDaoImpl.java
    com.bmc.security.model
      ApplicationUser.java
      UsernamePasswordModel.java
    com.bmc.security.service
      ApplicationUserDetailsService.java
```

- Dao

- **ApplicationUserDao & ApplicationUserDaoImpl –** This DAO layer has one interface and its implementation class where the logic to load the users with username and password is written.

- Model

- **ApplicationUser –** This class in model layer implements the UserDetails interface and sets the username, password, and the authorities.
- **UsernamePasswordModel –** This class acts as the DTO model to accepts the username and password in the Request body.

7. <u>Notification Service</u>

This micro service application is used

- to listen to the Kafka topic and receive the asynchronous message sent by various services to the Kafka topic.
- to send email to the Doctors and Users.

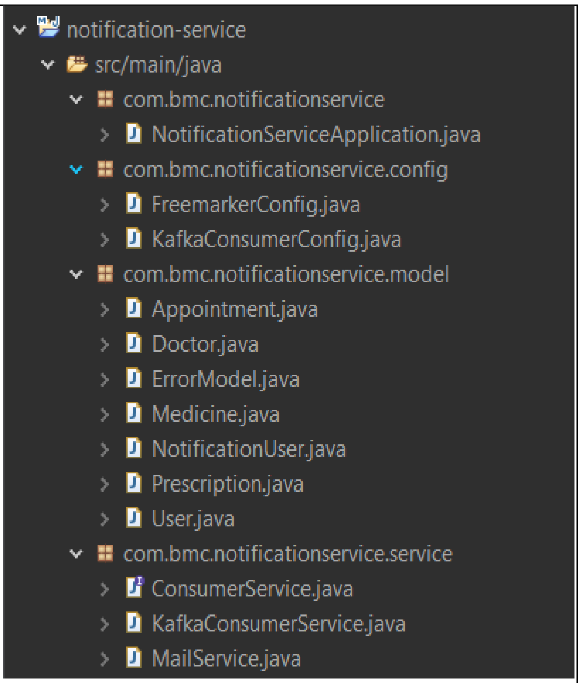
This Security micro service app has the following package structures.

- Config

- **KafkaConsumerConfig –** This class Creates the KafkaConsumer object and configures the Kafka bootstrapServer to the KafkaConsumer object, which is used to listen to the Kafka topic and receive the asynchronous message sent by various services to the Kafka topic.
- **FreemarkerConfig –** This class Creates the FreemarkerConfigurer object and it configures the freemarker template version to the FreemarkerConfigurer bean.

- Service

- **ConsumerService & KafkaConsumerService –** This service layer has interface ConsumerService and its implementation class KafkaConsumerService where the method listen is annotated with ***@KafkaListener*** annotation to listen to the Kafka topic and receive the asynchronous message sent by various services to the Kafka topic. Then these messages are casted to the appropriate bean object like Doctor, User, Rating and Payment and then these bean objects are sent to the MailService class to send email to the appropriate players.

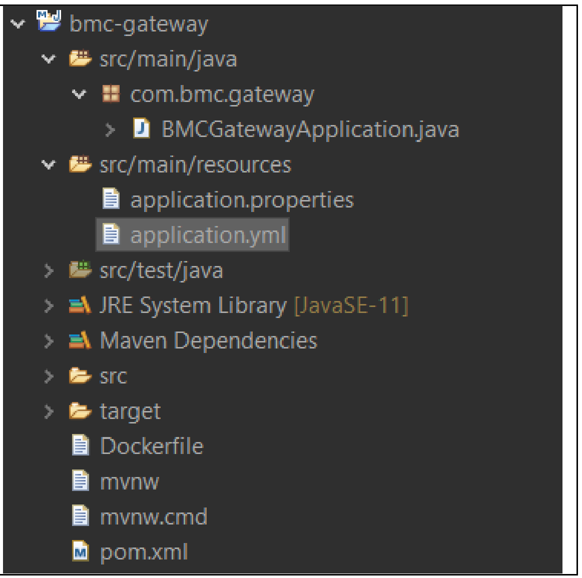


```
notification-service
  src/main/java
    com.bmc.notificationservice
      NotificationServiceApplication.java
    com.bmc.notificationservice.config
      FreemarkerConfig.java
      KafkaConsumerConfig.java
    com.bmc.notificationservice.model
      Appointment.java
      Doctor.java
      ErrorModel.java
      Medicine.java
      NotificationUser.java
      Prescription.java
      User.java
    com.bmc.notificationservice.service
      ConsumerService.java
      KafkaConsumerService.java
      MailService.java
```

• **MailService –** This class receives the request from KafkaConsumer class and applies the FTL templates and sends email to Doctor/User for the scenarios detailed in the requirements by using the ***AWS SES*** service. **- Model** • **Doctor, User, Prescription and Rating –** These are the different objects which will be received by the Kafkalistener.	

8. BMC Gateway

This micro service application is an Gateway application which uses Spring Cloud Starter Gateway package and directs the traffic to the appropriate microservices url.

• **Application.yml –** All the routes, uris, predicates and the path to different microservices are defined in this application.yml file.	

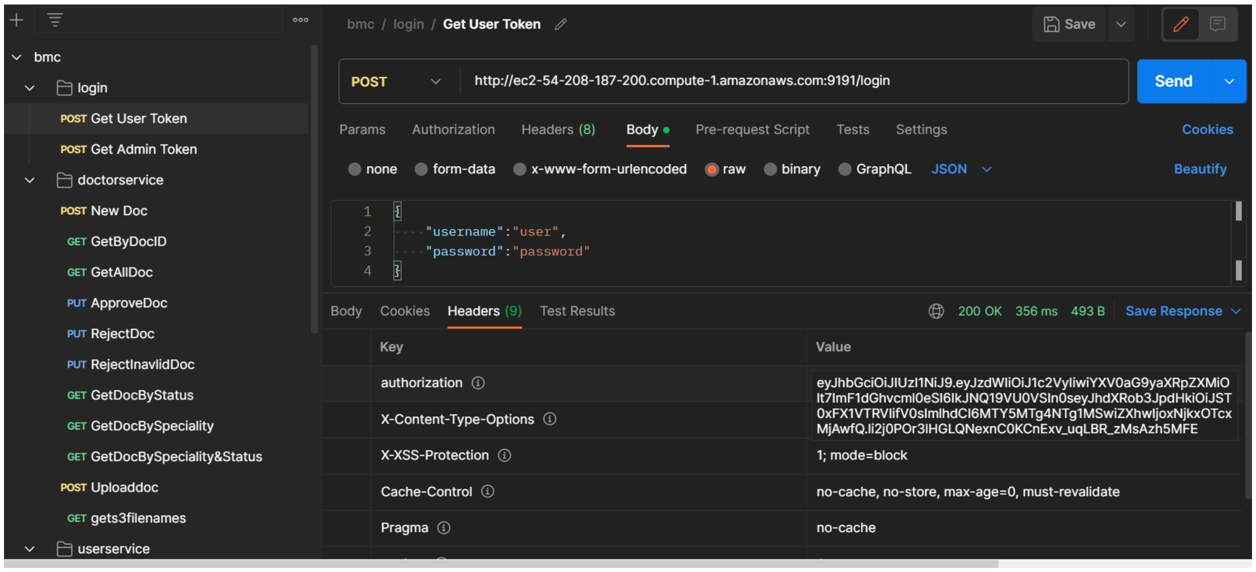
```
bmc-gateway
  src/main/java
    com.bmc.gateway
      BMCGatewayApplication.java
  src/main/resources
    application.properties
    application.yml
  src/test/java
  JRE System Library [JavaSE-11]
  Maven Dependencies
  src
  target
  Dockerfile
  mvnw
  mvnw.cmd
  pom.xml
```

Working Screenshots

1.) Security Services

a.) **Generating User Token**: In the Response Header section -> Authorization -> Token is generated.

Request body – username:**user**, password:**password**



POST http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/login

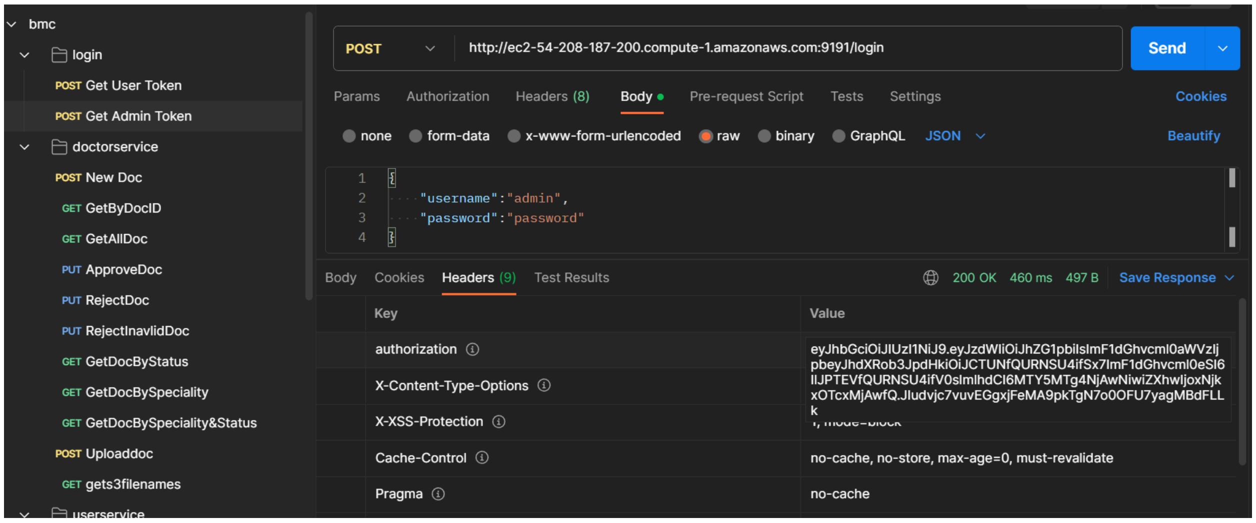
```
{
    "username":"user",
    "password":"password"
}
```

200 OK 356 ms 493 B

Key	Value
authorization	eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiYXV0aG9yaXRpZXMiOIt7ImF1dGhvcml0eSI6IkJNQ19VU0VSIn0seyJhdXRob3JpdHkiOiJST0xFX1VTRVIifV0sImlhdCI6MTY5MTg4NTg1MSwiZXhwIjoxNjkxOTcxMjAwfQ.Ii2j0POr3IHGLQNexnC0KCnExv_uqLBR_zMsAzh5MFE
X-Content-Type-Options	
X-XSS-Protection	1; mode=block
Cache-Control	no-cache, no-store, max-age=0, must-revalidate
Pragma	no-cache

b.) **Generating Admin Token**: In the Response Header section -> Authorization -> Token is generated.

Request body – username:**admin**, password:**password**



POST http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/login

```
{
    "username":"admin",
    "password":"password"
}
```

200 OK 460 ms 497 B

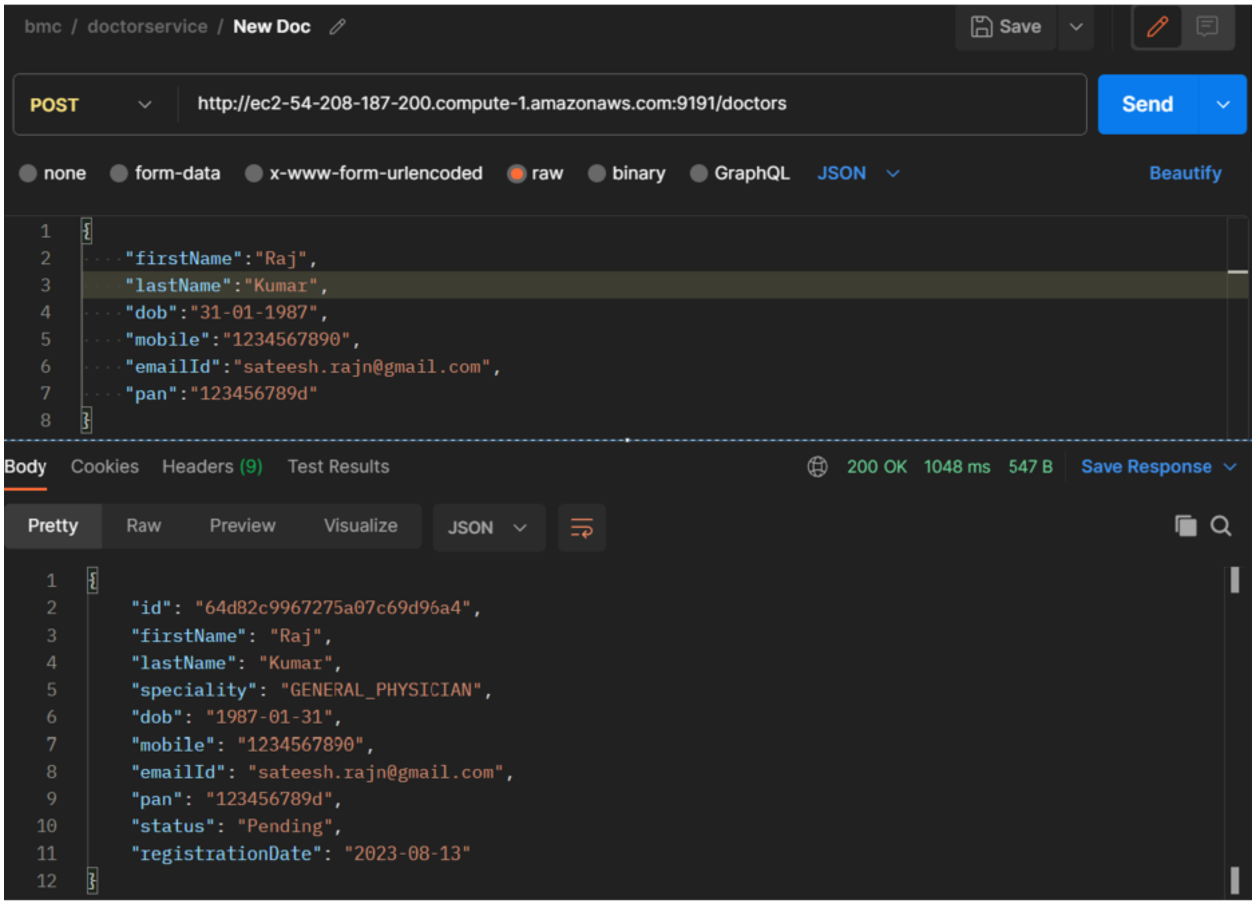
Key	Value
authorization	eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsImF1dGhvcml0aWVzIjpbeyJhdXRob3JpdHkiOiJCTUNfQURNSU4ifSx7ImF1dGhvcml0eSI6IlJPTEVfQURNSU4ifV0sImlhdCI6MTY5MTg4NjAwNiwiZXhwIjoxNjkxOTcxMjAwfQ.JIudvjc7vuvEGgxjFeMA9pkTgN7o0OFU7yagMBdFLLk
X-Content-Type-Options	
X-XSS-Protection	1; mode=block
Cache-Control	no-cache, no-store, max-age=0, must-revalidate
Pragma	no-cache

Working Screenshots

2.) Doctor Services

a.) URI: /doctors & HTTP method: POST & Correct Request

- This endpoint is responsible for collecting information about the doctor.



bmc / doctorservice / New Doc

POST http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/doctors

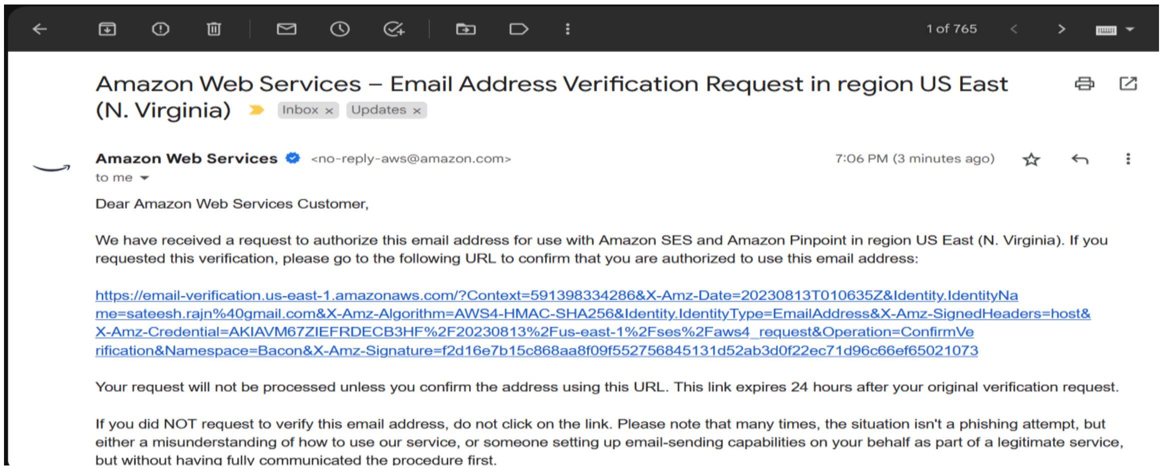
none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON

```
{
    "firstName":"Raj",
    "lastName":"Kumar",
    "dob":"31-01-1987",
    "mobile":"1234567890",
    "emailId":"sateesh.rajn@gmail.com",
    "pan":"123456789d"
}
```

Body | Cookies | Headers (9) | Test Results — 200 OK 1048 ms 547 B

```
{
    "id": "64d82c9967275a07c69d96a4",
    "firstName": "Raj",
    "lastName": "Kumar",
    "speciality": "GENERAL_PHYSICIAN",
    "dob": "1987-01-31",
    "mobile": "1234567890",
    "emailId": "sateesh.rajn@gmail.com",
    "pan": "123456789d",
    "status": "Pending",
    "registrationDate": "2023-08-13"
}
```

Verification email is triggered to Doctor's email id.



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

Amazon Web Services <no-reply-aws@amazon.com> 7:06 PM (3 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=591398334286&X-Amz-Date=20230813T010635Z&Identity.IdentityName=sateesh.rajn%40gmail.com&X-Amz-Algorithm=AWS4-HMAC-SHA256&Identity.IdentityType=EmailAddress&X-Amz-SignedHeaders=host&X-Amz-Credential=AKIAVM67ZIEFRDECB3HF%2F20230813%2Fus-east-1%2Fses%2Faws4_request&Operation=ConfirmVerification&Namespace=Bacon&X-Amz-Signature=f2d16e7b15c868aa8f09f552756845131d52ab3d0f22ec71d96c66ef65021073

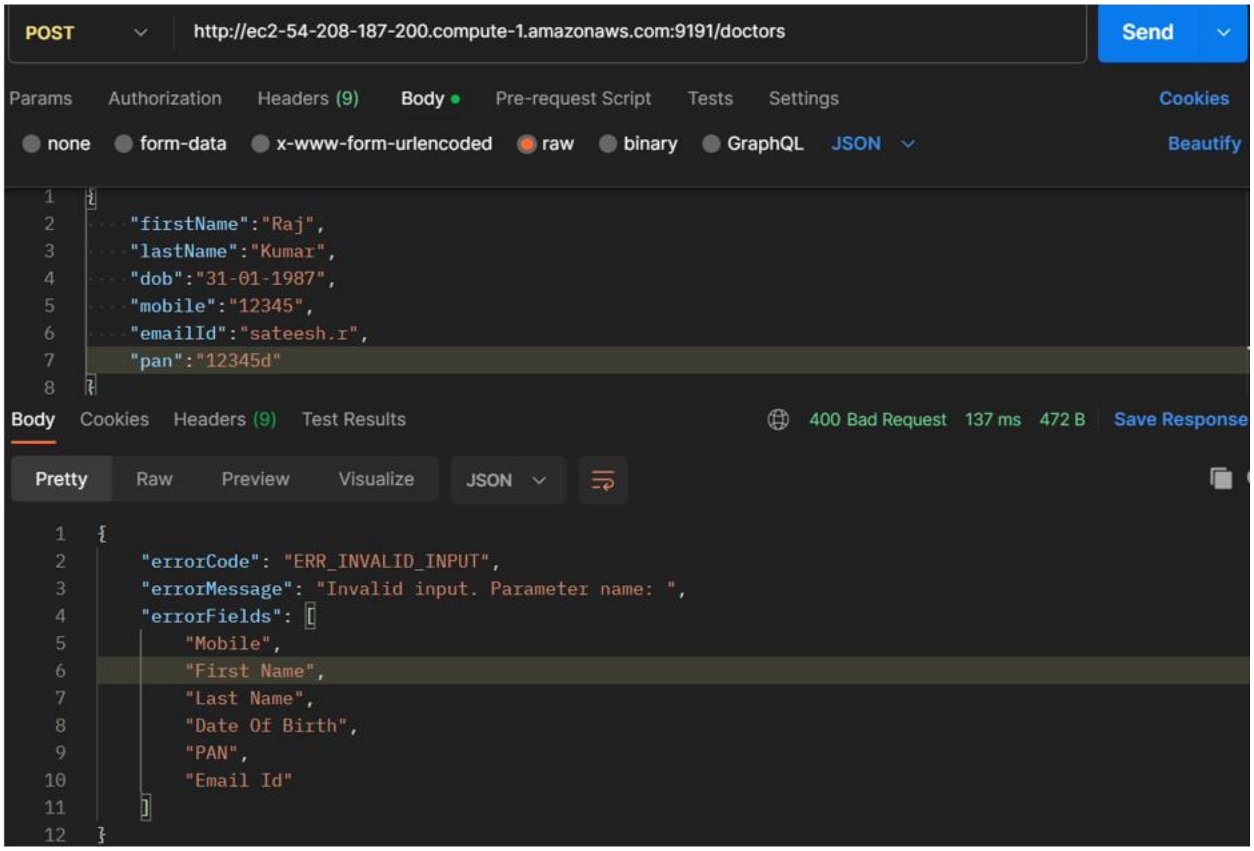
Your request will not be processed unless you confirm the address using this URL. This link expires 24 hours after your original verification request.

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.

Working Screenshots

b.) URI: /doctors & HTTP method: POST & Incorrect Request

- This endpoint is responsible for collecting information about the doctor.



POST http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/doctors

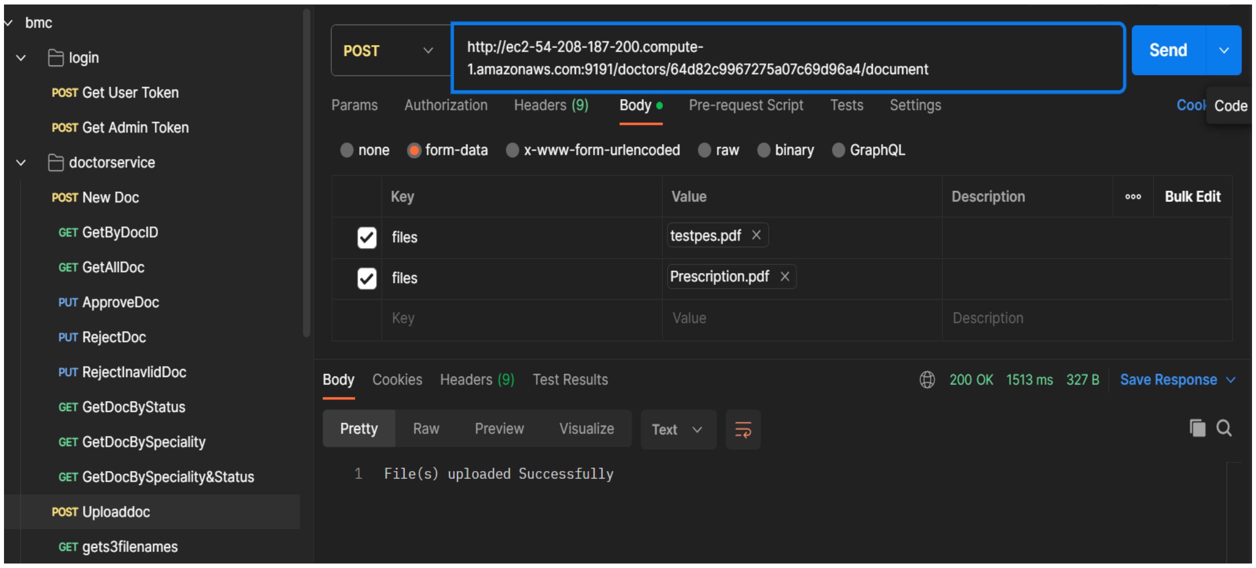
```
{
    "firstName":"Raj",
    "lastName":"Kumar",
    "dob":"31-01-1987",
    "mobile":"12345",
    "emailId":"sateesh.r",
    "pan":"12345d"
}
```

400 Bad Request 137 ms 472 B

```
{
    "errorCode": "ERR_INVALID_INPUT",
    "errorMessage": "Invalid input. Parameter name: ",
    "errorFields": [
        "Mobile",
        "First Name",
        "Last Name",
        "Date Of Birth",
        "PAN",
        "Email Id"
    ]
}
```

c.) URI: /doctors/{doctorId}/document & HTTP method: POST

- This endpoint is responsible for uploading the documents to an S3 bucket by the doctor.



bmc
- login
  - POST Get User Token
  - POST Get Admin Token
- doctorservice
  - POST New Doc
  - GET GetByDocID
  - GET GetAllDoc
  - PUT ApproveDoc
  - PUT RejectDoc
  - PUT RejectInavlidDoc
  - GET GetDocByStatus
  - GET GetDocBySpeciality
  - GET GetDocBySpeciality&Status
  - POST Uploaddoc
  - GET gets3filenames

POST http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/doctors/64d82c9967275a07c69d96a4/document

Body: form-data

	Key	Value	Description
✓	files	testpes.pdf	
✓	files	Prescription.pdf	

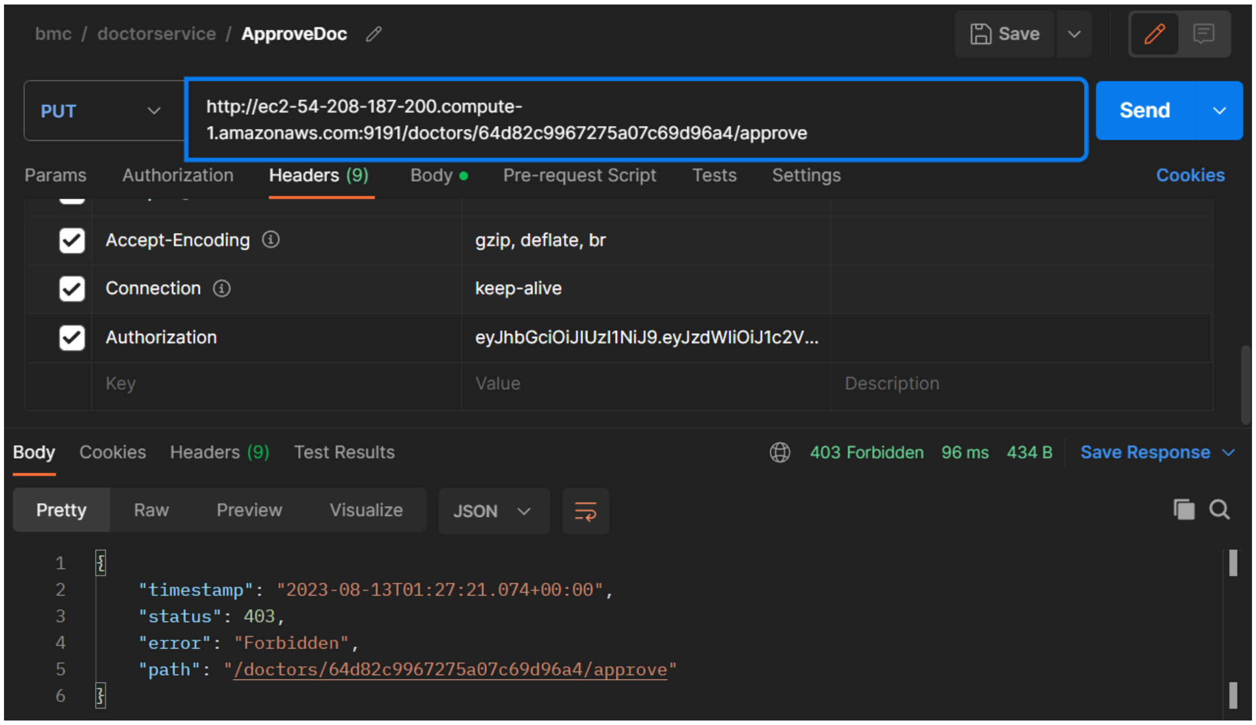
200 OK 1513 ms 327 B

```
File(s) uploaded Successfully
```

Working Screenshots

d.) URI: /doctors/{doctorId}/approve & HTTP method: PUT & With User Token

- This endpoint is responsible for approving the doctor's registration request.



bmc / doctorservice / ApproveDoc

PUT http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/doctors/64d82c9967275a07c69d96a4/approve

Headers (9)

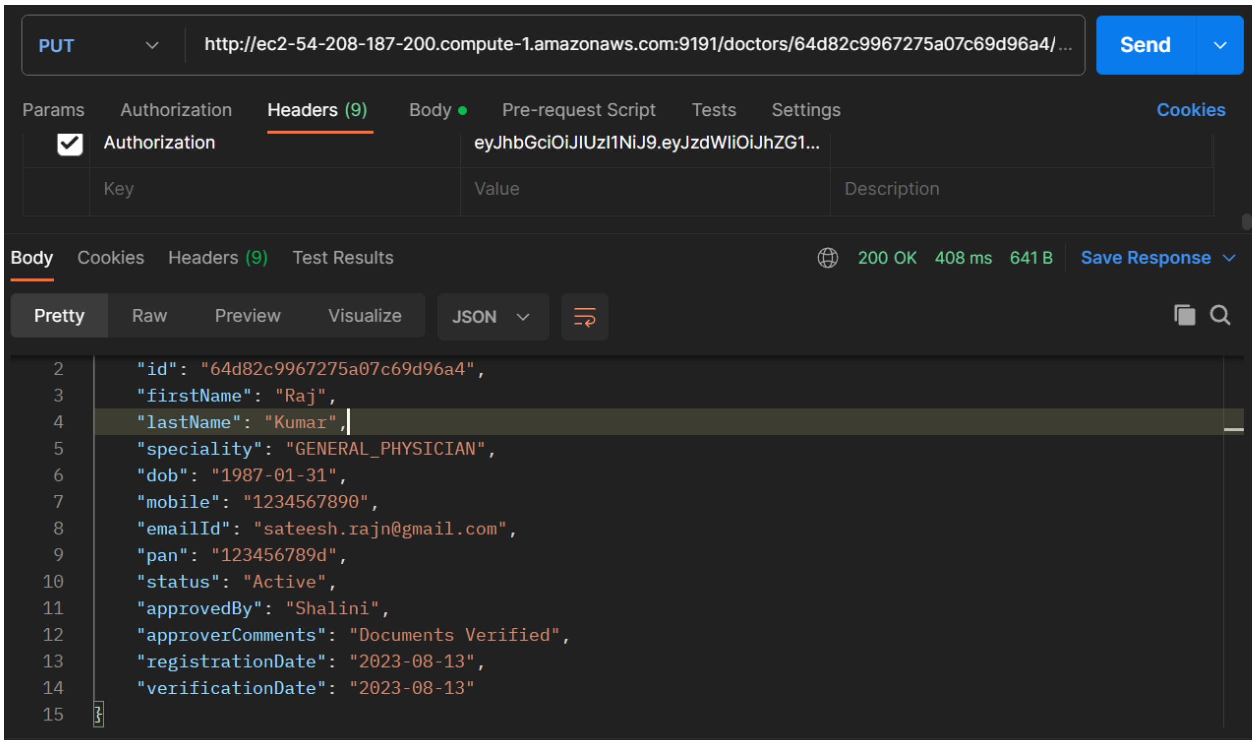
Key	Value	Description
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
Authorization	eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2V...	

403 Forbidden 96 ms 434 B

```
{
    "timestamp": "2023-08-13T01:27:21.074+00:00",
    "status": 403,
    "error": "Forbidden",
    "path": "/doctors/64d82c9967275a07c69d96a4/approve"
}
```

e.) URI: /doctors/{doctorId}/approve & HTTP method: PUT & With Admin Token

- This endpoint is responsible for approving the doctor's registration request.



PUT http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/doctors/64d82c9967275a07c69d96a4/...

Headers (9)

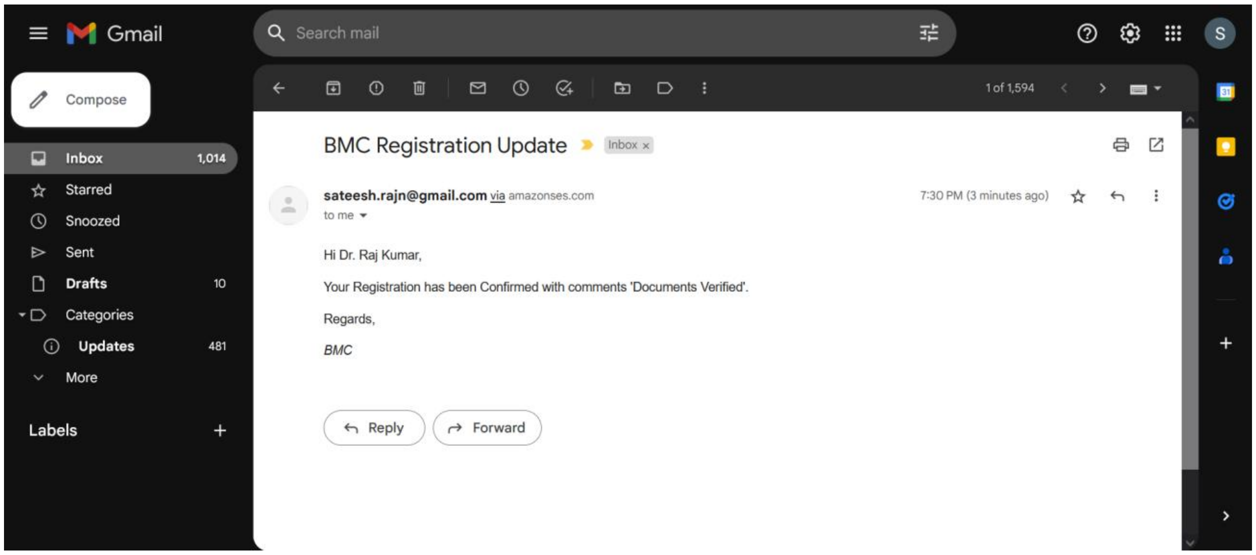
Key	Value	Description
Authorization	eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1...	

200 OK 408 ms 641 B

```
    "id": "64d82c9967275a07c69d96a4",
    "firstName": "Raj",
    "lastName": "Kumar",
    "speciality": "GENERAL_PHYSICIAN",
    "dob": "1987-01-31",
    "mobile": "1234567890",
    "emailId": "sateesh.rajn@gmail.com",
    "pan": "123456789d",
    "status": "Active",
    "approvedBy": "Shalini",
    "approverComments": "Documents Verified",
    "registrationDate": "2023-08-13",
    "verificationDate": "2023-08-13"
}
```

Working Screenshots

Registration Confirmation mail is sent to the doctor on Approval.



BMC Registration Update Inbox

sateesh.rajn@gmail.com via amazonses.com
to me
7:30 PM (3 minutes ago)

Hi Dr. Raj Kumar,

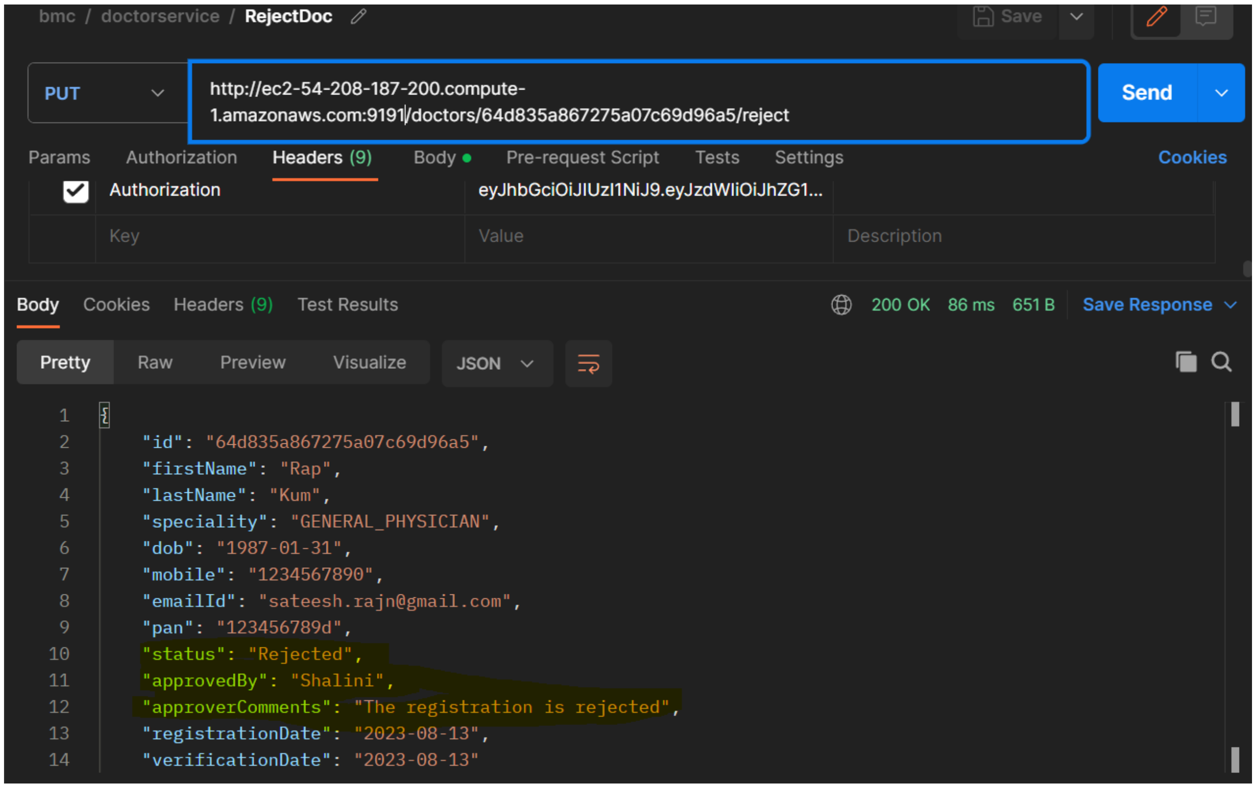
Your Registration has been Confirmed with comments 'Documents Verified'.

Regards,

BMC

f.) URI: /doctors/{doctorId}/reject & HTTP method: PUT

- This endpoint is responsible for rejecting the doctor's registration request.



PUT http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/doctors/64d835a867275a07c69d96a5/reject

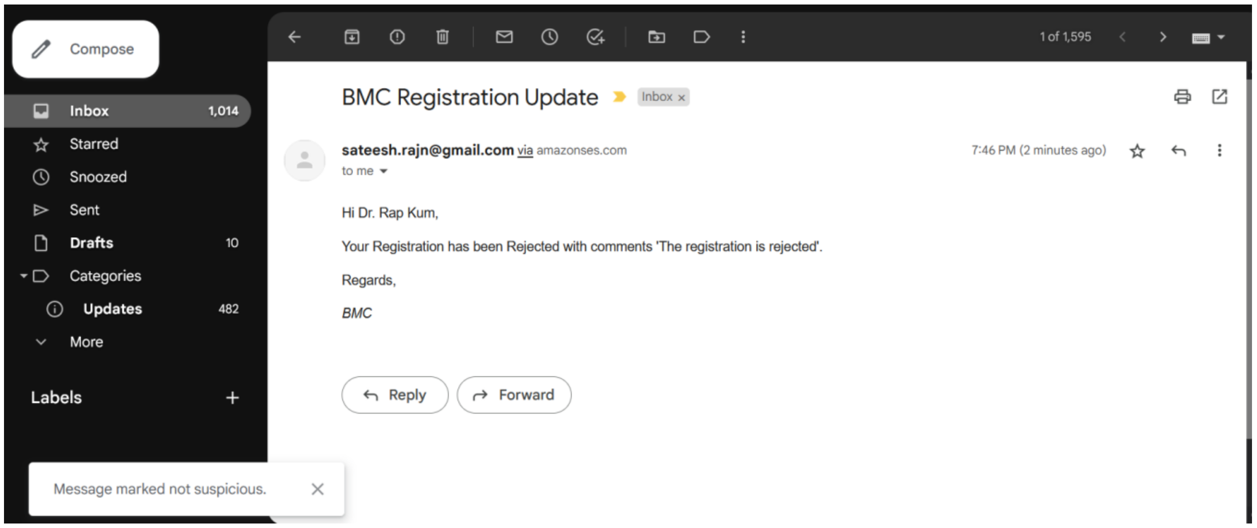
Key	Value	Description
✓ Authorization	eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1...	

200 OK 86 ms 651 B

```
{
    "id": "64d835a867275a07c69d96a5",
    "firstName": "Rap",
    "lastName": "Kum",
    "speciality": "GENERAL_PHYSICIAN",
    "dob": "1987-01-31",
    "mobile": "1234567890",
    "emailId": "sateesh.rajn@gmail.com",
    "pan": "123456789d",
    "status": "Rejected",
    "approvedBy": "Shalini",
    "approverComments": "The registration is rejected",
    "registrationDate": "2023-08-13",
    "verificationDate": "2023-08-13"
```

Working Screenshots

Registration Rejection mail is sent to the doctor on Rejection.



BMC Registration Update

sateesh.rajn@gmail.com via amazonses.com

to me

7:46 PM (2 minutes ago)

Hi Dr. Rap Kum,

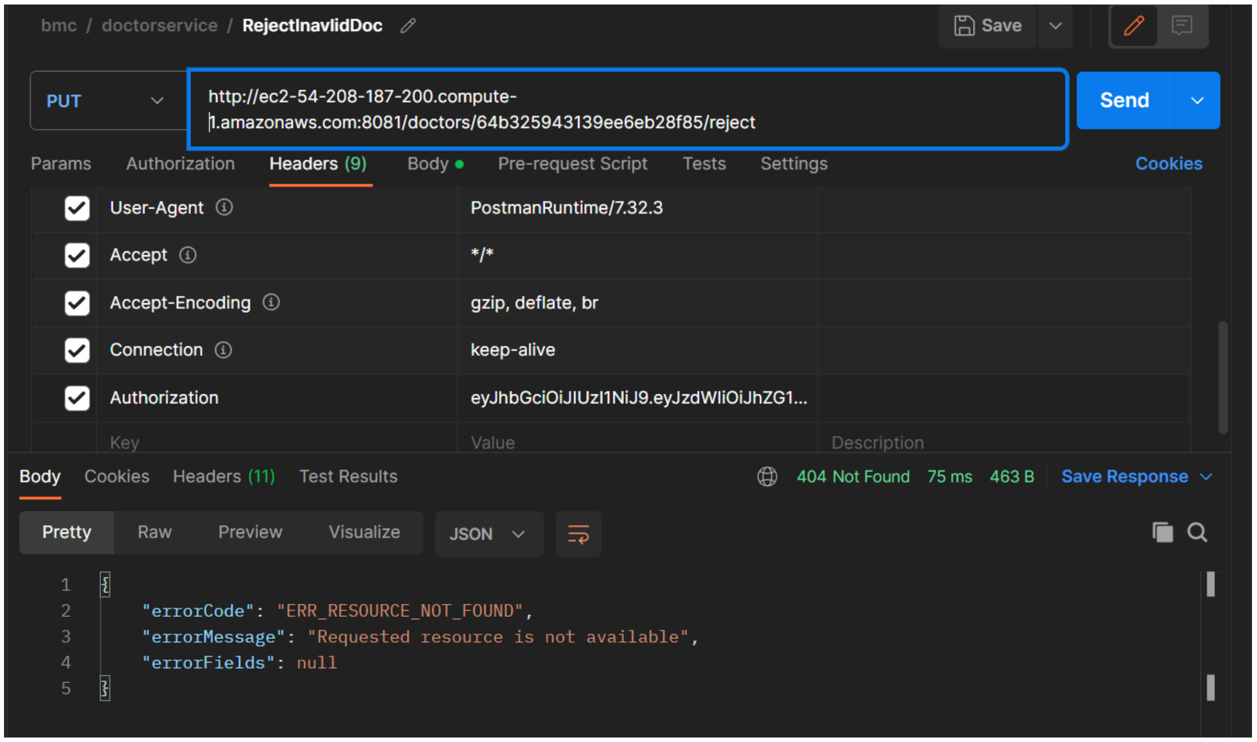
Your Registration has been Rejected with comments 'The registration is rejected'.

Regards,

BMC

Message marked not suspicious.

g.) **URI: /doctors/{doctorId}/reject & HTTP method: PUT & Invalid Doctor id is provided as input**



bmc / doctorservice / RejectInavlidDoc

PUT http://ec2-54-208-187-200.compute-1.amazonaws.com:8081/doctors/64b325943139ee6eb28f85/reject

	Key	Value	Description
✓	User-Agent	PostmanRuntime/7.32.3	
✓	Accept	*/*	
✓	Accept-Encoding	gzip, deflate, br	
✓	Connection	keep-alive	
✓	Authorization	eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1...	

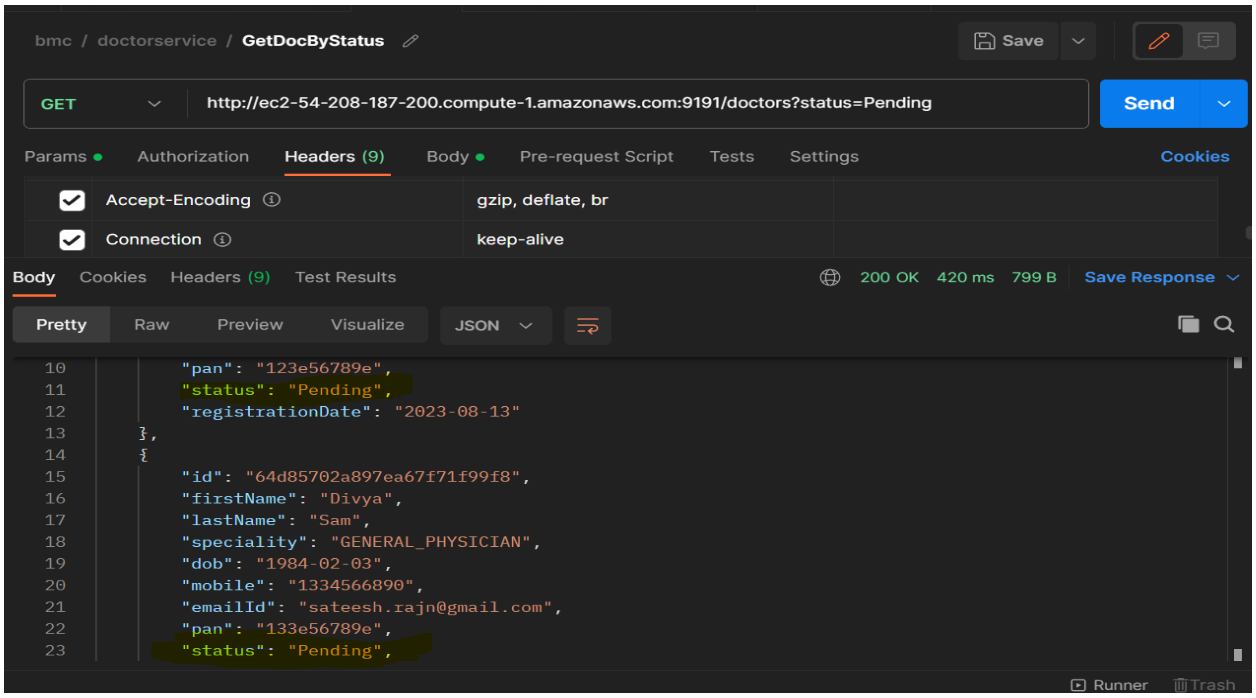
404 Not Found 75 ms 463 B

```
{
    "errorCode": "ERR_RESOURCE_NOT_FOUND",
    "errorMessage": "Requested resource is not available",
    "errorFields": null
}
```

Working Screenshots

h.) URI: /doctors?status=Pending & HTTP method: GET

- This endpoint is responsible for returning the Doctor in Pending state.



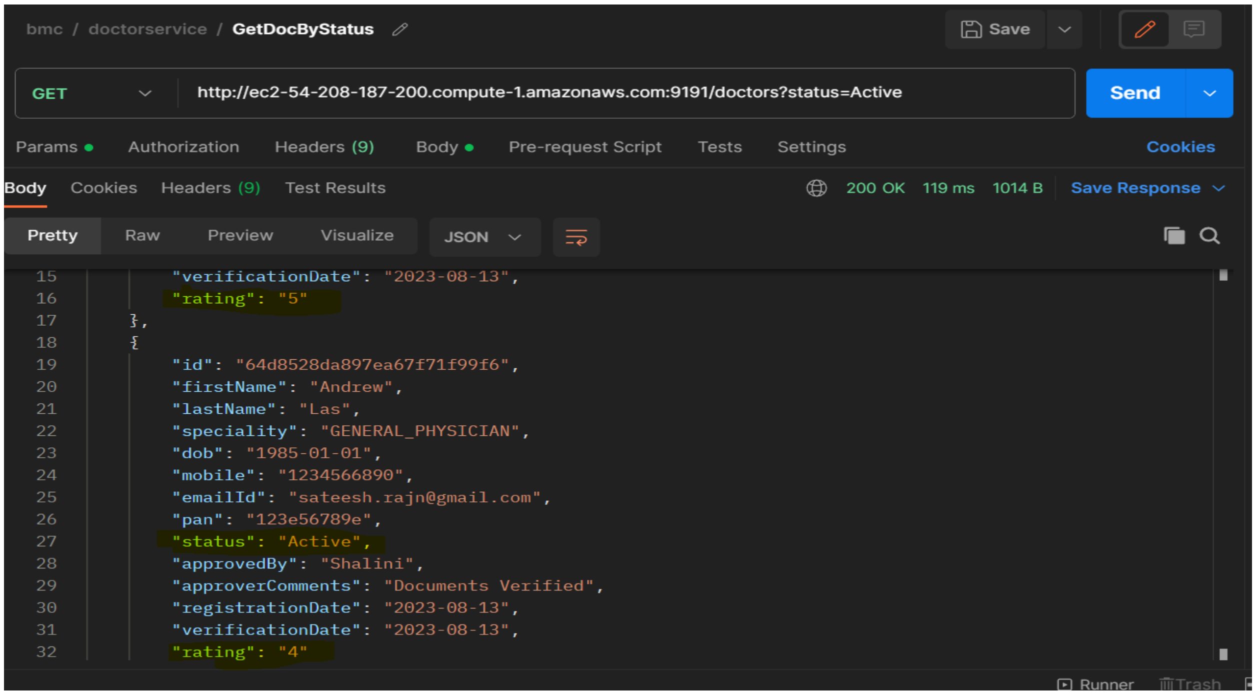
http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/doctors?status=Pending

Accept-Encoding	gzip, deflate, br
Connection	keep-alive

```
            "pan": "123e56789e",
            "status": "Pending",
            "registrationDate": "2023-08-13"
        },
        {
            "id": "64d85702a897ea67f71f99f8",
            "firstName": "Divya",
            "lastName": "Sam",
            "speciality": "GENERAL_PHYSICIAN",
            "dob": "1984-02-03",
            "mobile": "1334566890",
            "emailId": "sateesh.rajn@gmail.com",
            "pan": "133e56789e",
            "status": "Pending",
```

i.) URI: /doctors?status=Active & HTTP method: GET

- This endpoint is responsible for returning the Doctor in Active state and sorted by Rating in Descending order with Highest rated doctor coming first.



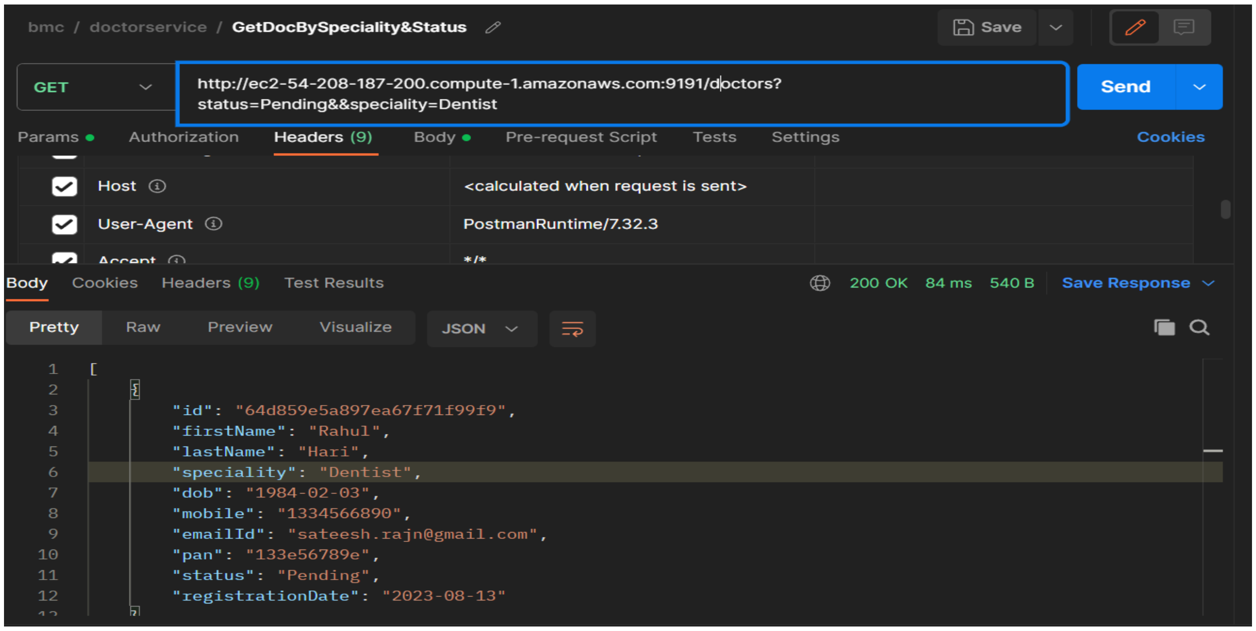
http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/doctors?status=Active

```
            "verificationDate": "2023-08-13",
            "rating": "5"
        },
        {
            "id": "64d8528da897ea67f71f99f6",
            "firstName": "Andrew",
            "lastName": "Las",
            "speciality": "GENERAL_PHYSICIAN",
            "dob": "1985-01-01",
            "mobile": "1234566890",
            "emailId": "sateesh.rajn@gmail.com",
            "pan": "123e56789e",
            "status": "Active",
            "approvedBy": "Shalini",
            "approverComments": "Documents Verified",
            "registrationDate": "2023-08-13",
            "verificationDate": "2023-08-13",
            "rating": "4"
```

Working Screenshots

j.) URI: /doctors?status=Pending&speciality=Dentist & HTTP method: GET

- This endpoint is responsible for returning the Doctor in Pending State with Specialty as Dentist.



bmc / doctorservice / GetDocBySpeciality&Status

GET http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/doctors?status=Pending&&speciality=Dentist

Headers (9)

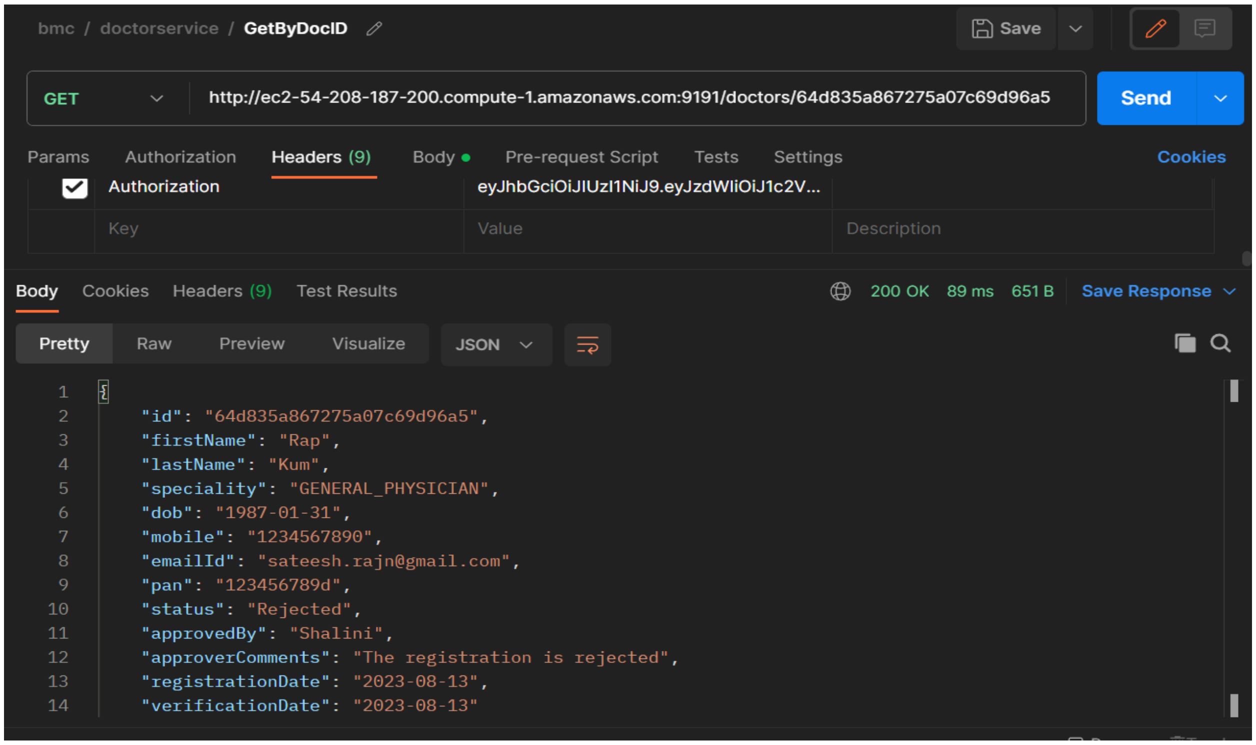
	Key	Value
✓	Host	<calculated when request is sent>
✓	User-Agent	PostmanRuntime/7.32.3
✓	Accept	*/*

200 OK 84 ms 540 B

```
[
    {
        "id": "64d859e5a897ea67f71f99f9",
        "firstName": "Rahul",
        "lastName": "Hari",
        "speciality": "Dentist",
        "dob": "1984-02-03",
        "mobile": "1334566890",
        "emailId": "sateesh.rajn@gmail.com",
        "pan": "133e56789e",
        "status": "Pending",
        "registrationDate": "2023-08-13"
    }
```

k.) URI: /doctors/{doctorId} & HTTP method: GET

- This endpoint is responsible for returning the details of the doctor based on the doctor ID.



bmc / doctorservice / GetByDocID

GET http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/doctors/64d835a867275a07c69d96a5

Headers (9)

	Key	Value	Description
✓	Authorization	eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2V...	

200 OK 89 ms 651 B

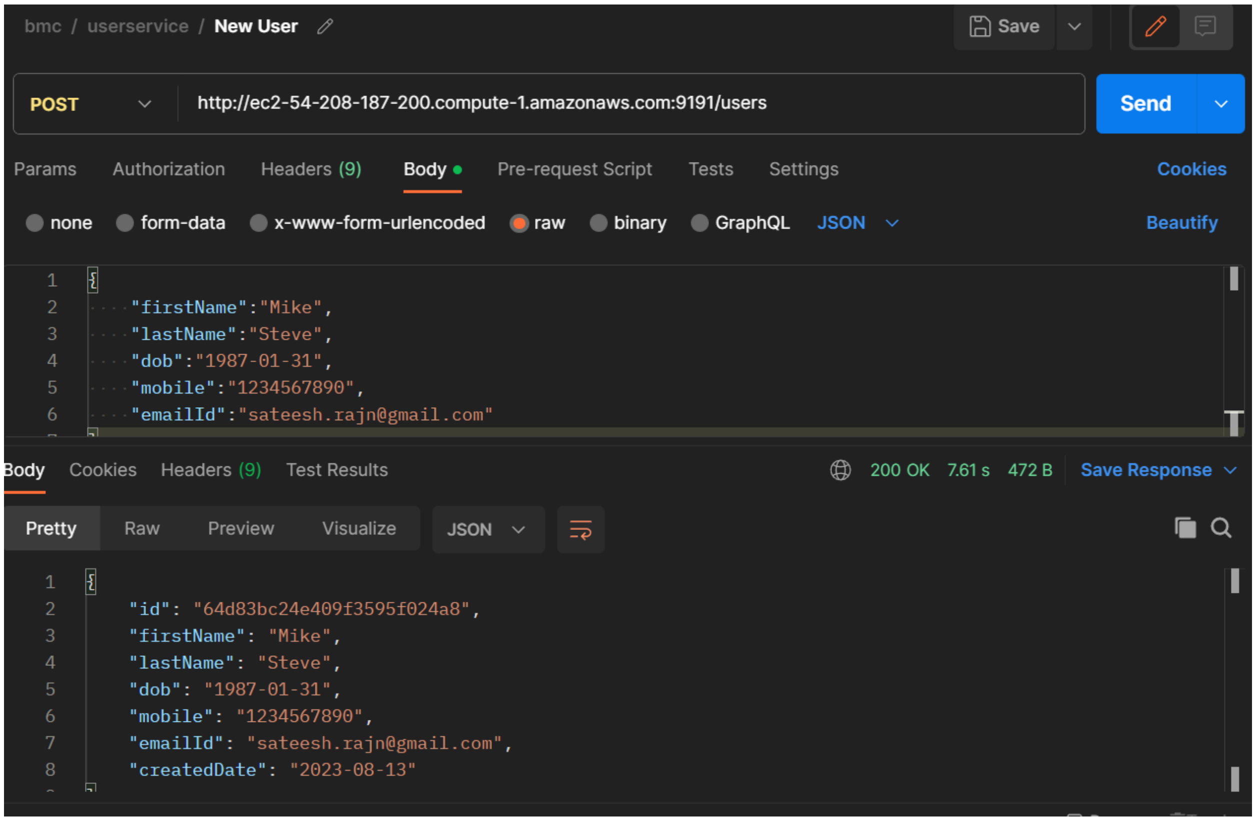
```
{
    "id": "64d835a867275a07c69d96a5",
    "firstName": "Rap",
    "lastName": "Kum",
    "speciality": "GENERAL_PHYSICIAN",
    "dob": "1987-01-31",
    "mobile": "1234567890",
    "emailId": "sateesh.rajn@gmail.com",
    "pan": "123456789d",
    "status": "Rejected",
    "approvedBy": "Shalini",
    "approverComments": "The registration is rejected",
    "registrationDate": "2023-08-13",
    "verificationDate": "2023-08-13"
```

Working Screenshots

3. User Services

a.) URI: /users & HTTP method: POST & Correct Request

- This endpoint is responsible for collecting information about the user.

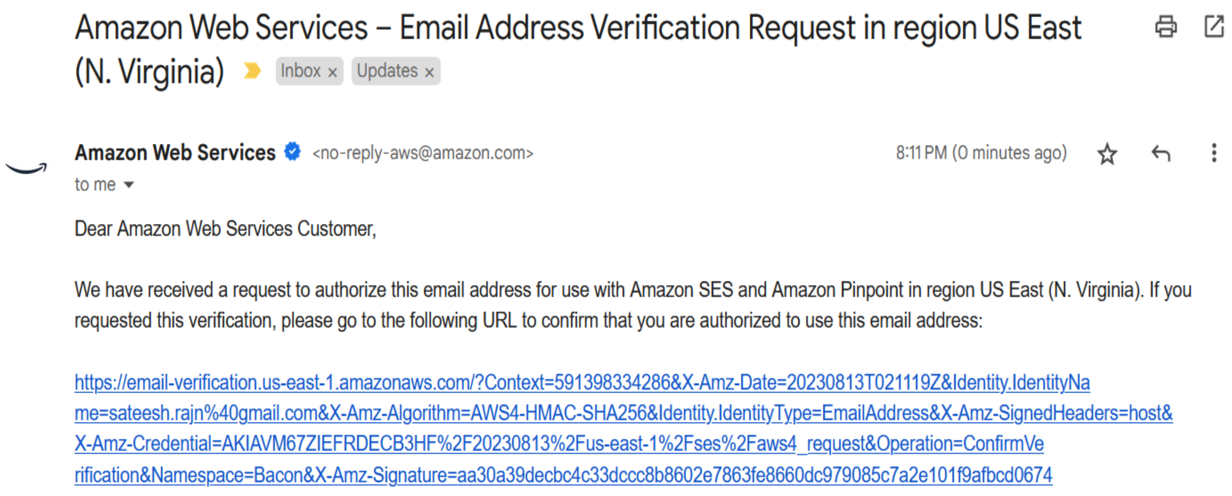


POST http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/users

```
{
    "firstName":"Mike",
    "lastName":"Steve",
    "dob":"1987-01-31",
    "mobile":"1234567890",
    "emailId":"sateesh.rajn@gmail.com"
```

```
{
    "id": "64d83bc24e409f3595f024a8",
    "firstName": "Mike",
    "lastName": "Steve",
    "dob": "1987-01-31",
    "mobile": "1234567890",
    "emailId": "sateesh.rajn@gmail.com",
    "createdDate": "2023-08-13"
```

Verification email is triggered to User's email id.



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

Amazon Web Services <no-reply-aws@amazon.com> 8:11 PM (0 minutes ago)
to me

Dear Amazon Web Services Customer,

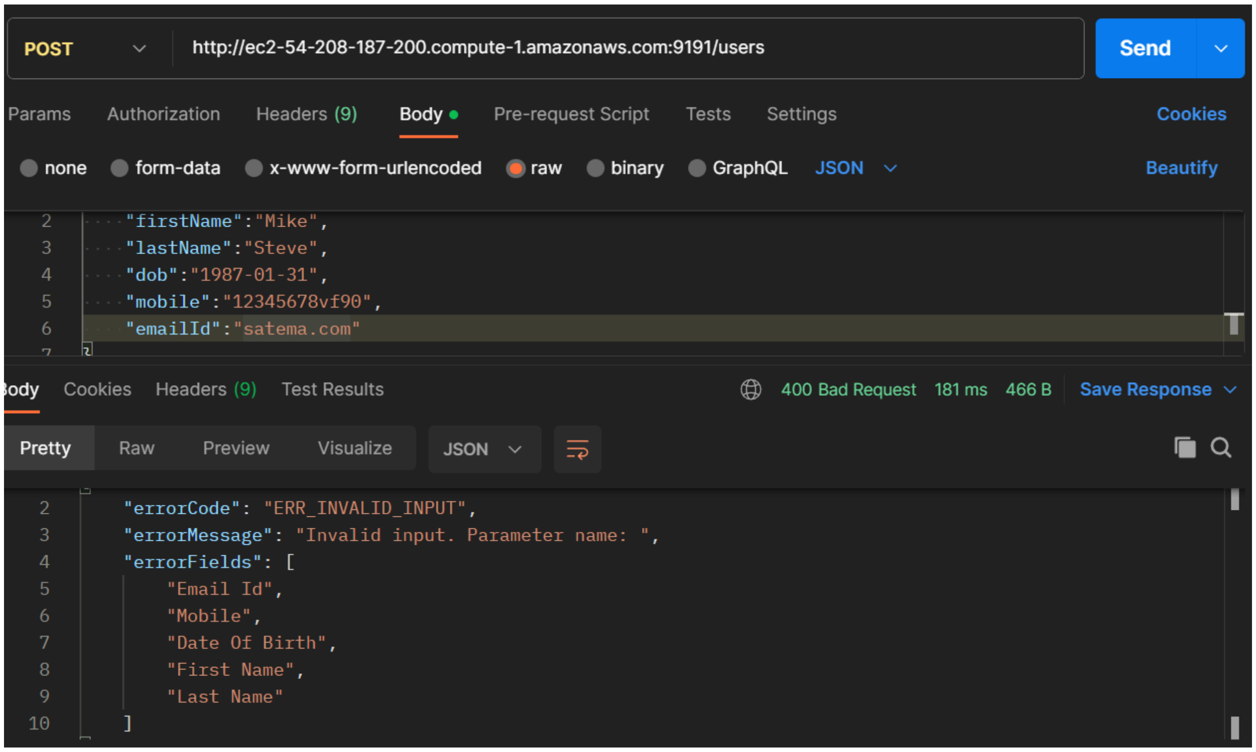
We have received a request to authorize this email address for use with Amazon SES and Amazon Pinpoint in region US East (N. Virginia). If you requested this verification, please go to the following URL to confirm that you are authorized to use this email address:

https://email-verification.us-east-1.amazonaws.com/?Context=591398334286&X-Amz-Date=20230813T021119Z&Identity.IdentityName=sateesh.rajn%40gmail.com&X-Amz-Algorithm=AWS4-HMAC-SHA256&Identity.IdentityType=EmailAddress&X-Amz-SignedHeaders=host&X-Amz-Credential=AKIAVM67ZIEFRDECB3HF%2F20230813%2Fus-east-1%2Fses%2Faws4_request&Operation=ConfirmVerification&Namespace=Bacon&X-Amz-Signature=aa30a39decbc4c33dccc8b8602e7863fe8660dc979085c7a2e101f9afbcd0674

Working Screenshots

b.) URI: /Users & HTTP method: POST & Incorrect Request

- This endpoint is responsible for collecting information about the doctor.



POST http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/users

Params | Authorization | Headers (9) | Body | Pre-request Script | Tests | Settings

raw JSON

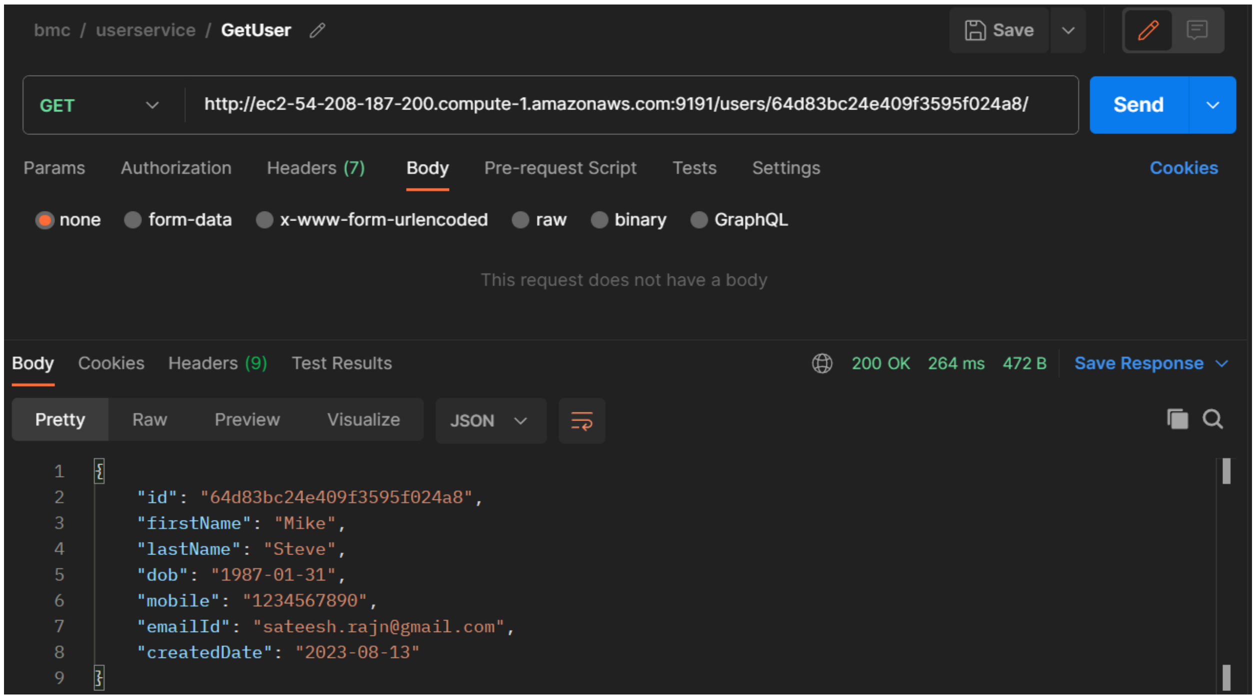
```
    "firstName":"Mike",
    "lastName":"Steve",
    "dob":"1987-01-31",
    "mobile":"12345678vf90",
    "emailId":"satema.com"
```

400 Bad Request 181 ms 466 B

```
    "errorCode": "ERR_INVALID_INPUT",
    "errorMessage": "Invalid input. Parameter name: ",
    "errorFields": [
        "Email Id",
        "Mobile",
        "Date Of Birth",
        "First Name",
        "Last Name"
    ]
```

c.) URI: /users/{userId} & HTTP method: GET

- This endpoint is responsible for returning the details of the user based on the userId.



bmc / userservice / GetUser

GET http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/users/64d83bc24e409f3595f024a8/

Params | Authorization | Headers (7) | Body | Pre-request Script | Tests | Settings

none

This request does not have a body

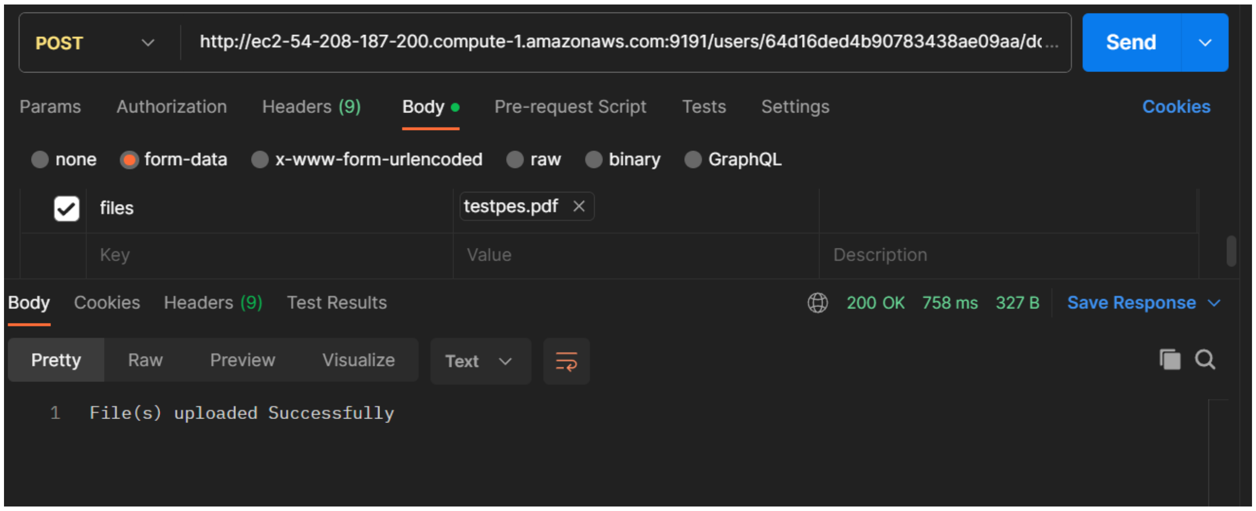
200 OK 264 ms 472 B

```
{
    "id": "64d83bc24e409f3595f024a8",
    "firstName": "Mike",
    "lastName": "Steve",
    "dob": "1987-01-31",
    "mobile": "1234567890",
    "emailId": "sateesh.rajn@gmail.com",
    "createdDate": "2023-08-13"
}
```

Working Screenshots

d.) URI: /users/{userId}/document & HTTP method: POST

- This endpoint is responsible for uploading the documents to an S3 bucket by the user.



POST http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/users/64d16ded4b90783438ae09aa/dc...

Body: form-data

Key	Value	Description
files	testpes.pdf	

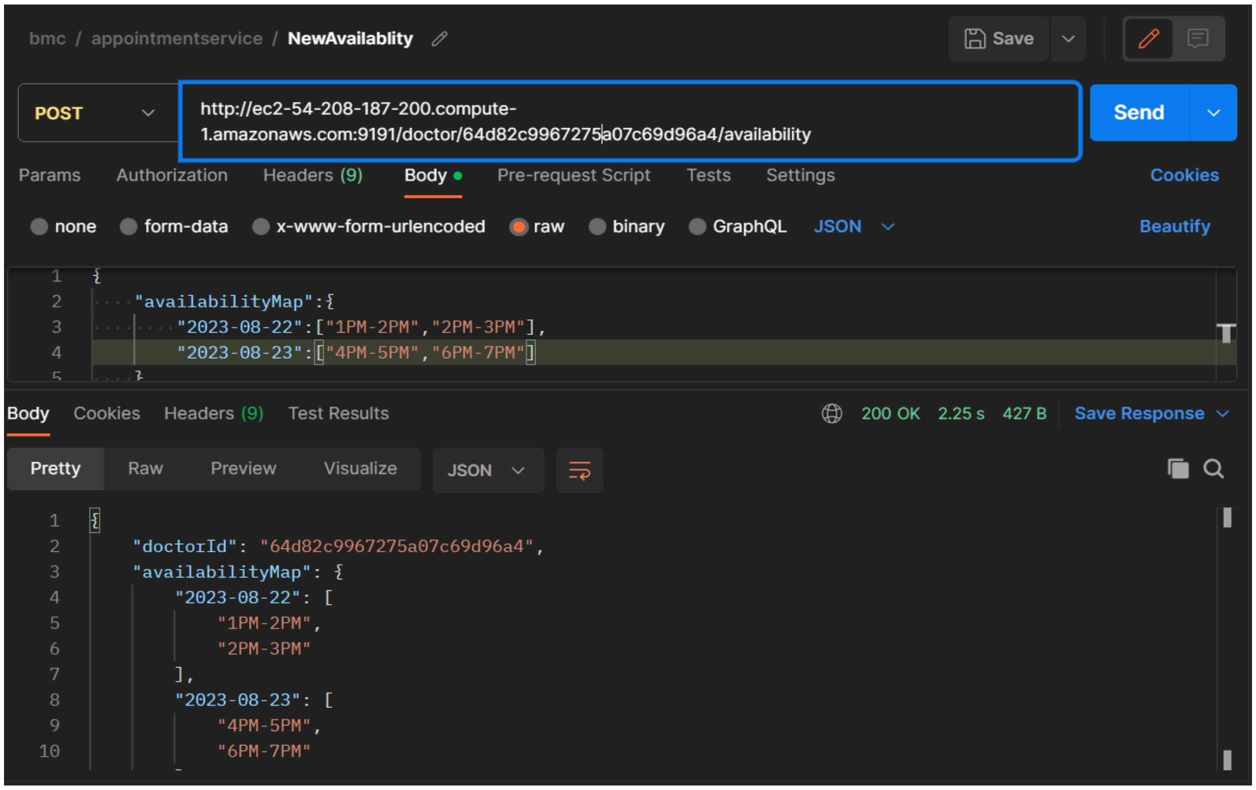
200 OK 758 ms 327 B

```
File(s) uploaded Successfully
```

4. Appointment Services

a.) URI: /doctor/{doctorId}/availability & HTTP method: POST

- This endpoint is responsible for updating the availability of the doctors.



bmc / appointmentservice / NewAvailablity

POST http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/doctor/64d82c9967275a07c69d96a4/availability

```
{
    "availabilityMap":{
        "2023-08-22":["1PM-2PM","2PM-3PM"],
        "2023-08-23":["4PM-5PM","6PM-7PM"]
    }
```

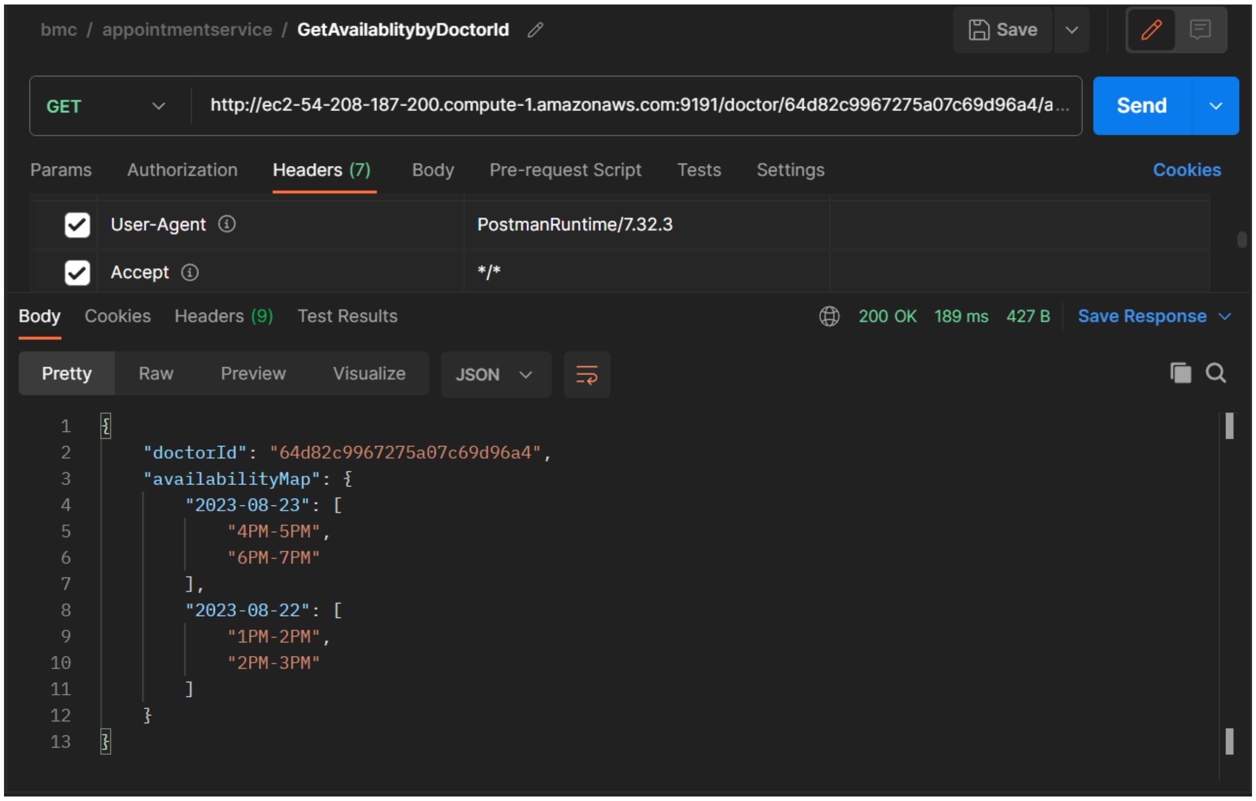
200 OK 2.25 s 427 B

```
{
    "doctorId": "64d82c9967275a07c69d96a4",
    "availabilityMap": {
        "2023-08-22": [
            "1PM-2PM",
            "2PM-3PM"
        ],
        "2023-08-23": [
            "4PM-5PM",
            "6PM-7PM"
```

Working Screenshots

b.) URI: /doctor/{doctorId}/availability & HTTP method: GET

- This endpoint is responsible for fetching the availability of the doctors.


bmc / appointmentservice / GetAvailablitybyDoctorId
Save
Send
Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies
Body Cookies Headers (9) Test Results
Save Response
Pretty Raw Preview Visualize JSON

GET http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/doctor/64d82c9967275a07c69d96a4/a...

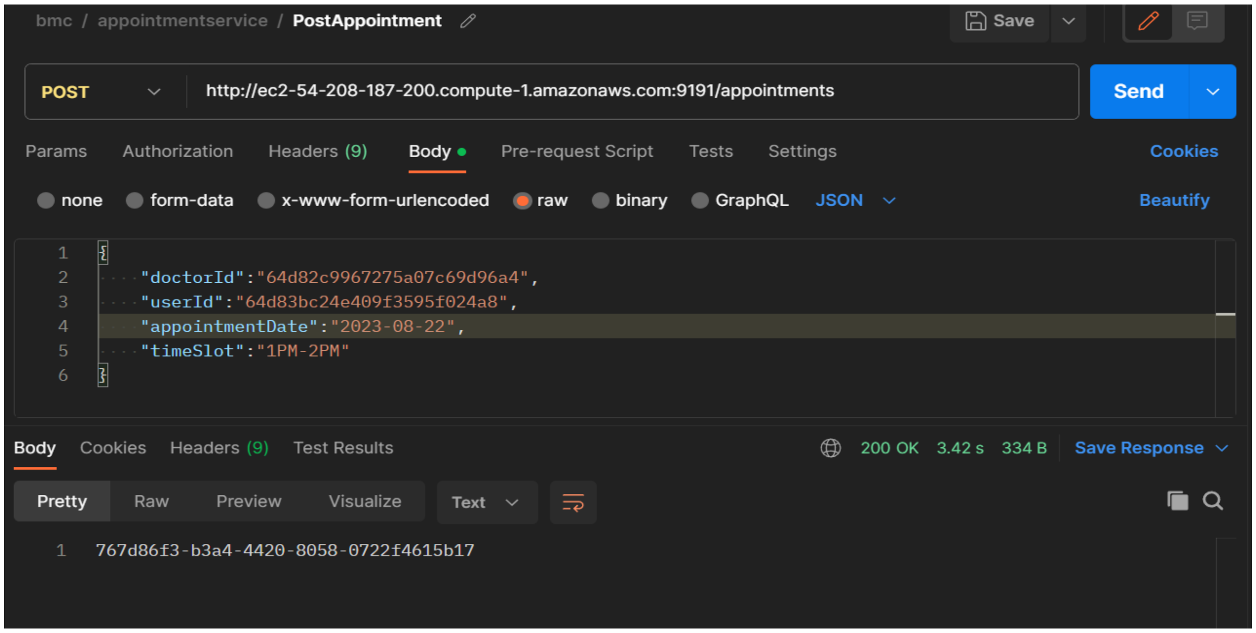
User-Agent	PostmanRuntime/7.32.3
Accept	*/*

200 OK 189 ms 427 B

```
{
    "doctorId": "64d82c9967275a07c69d96a4",
    "availabilityMap": {
        "2023-08-23": [
            "4PM-5PM",
            "6PM-7PM"
        ],
        "2023-08-22": [
            "1PM-2PM",
            "2PM-3PM"
        ]
    }
}
```

c.) URI: /appointments & HTTP method: POST

- This endpoint is responsible for booking an appointment.


bmc / appointmentservice / PostAppointment
Save
Send
Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify
Body Cookies Headers (9) Test Results
Save Response
Pretty Raw Preview Visualize Text

POST http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/appointments

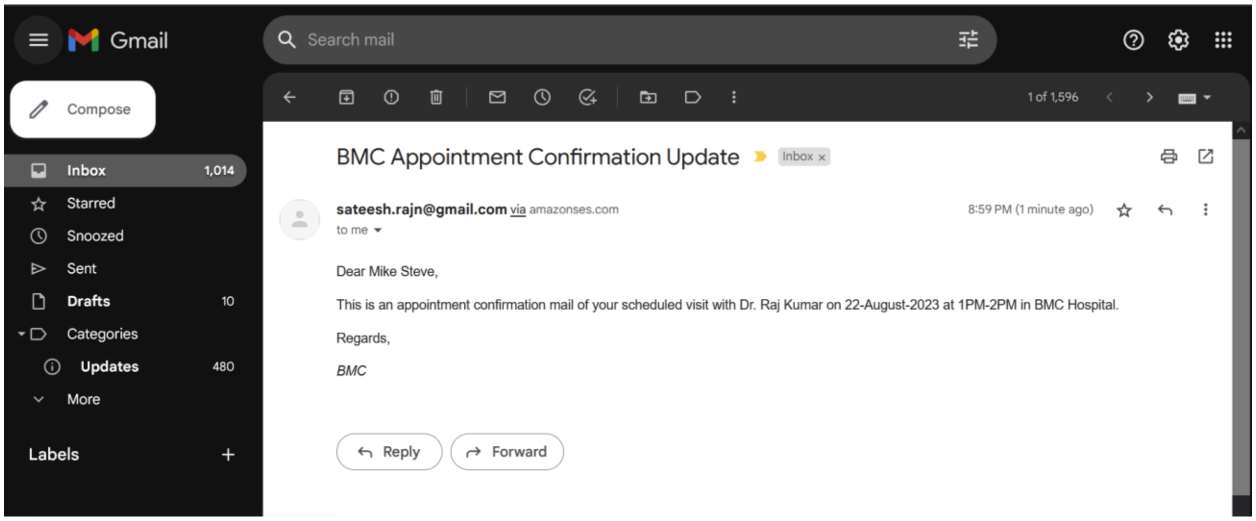
```
{
    "doctorId":"64d82c9967275a07c69d96a4",
    "userId":"64d83bc24e409f3595f024a8",
    "appointmentDate":"2023-08-22",
    "timeSlot":"1PM-2PM"
}
```

200 OK 3.42 s 334 B

```
767d86f3-b3a4-4420-8058-0722f4615b17
```

Working Screenshots

An appointment confirmation email is triggered to the user email id.



BMC Appointment Confirmation Update

sateesh.rajn@gmail.com via amazonses.com 8:59 PM (1 minute ago)
to me

Dear Mike Steve,

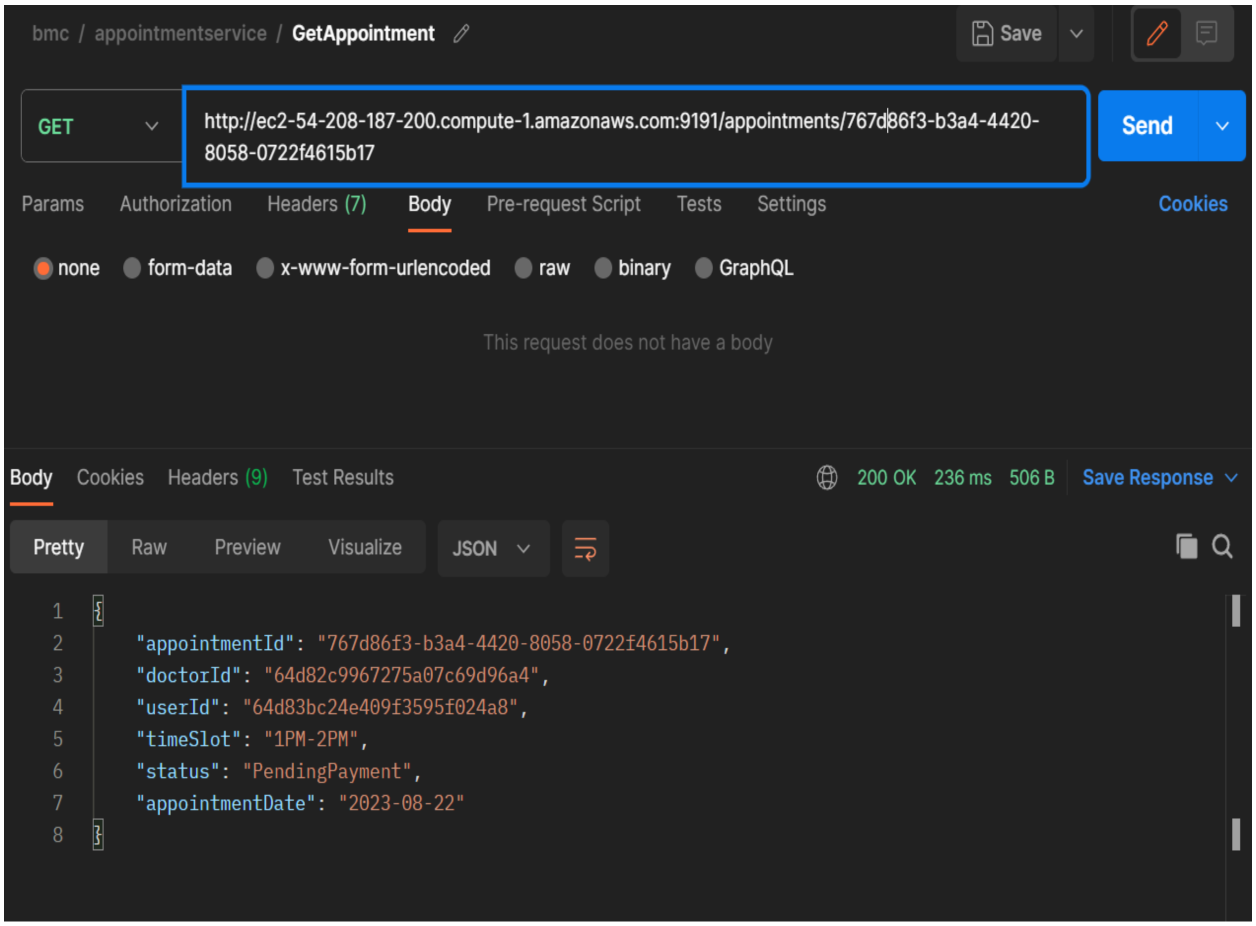
This is an appointment confirmation mail of your scheduled visit with Dr. Raj Kumar on 22-August-2023 at 1PM-2PM in BMC Hospital.

Regards,

BMC

d.) URI: /appointments/{appointmentId} & HTTP method: GET

- This endpoint is responsible for getting details of an appointment by its id.



GET http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/appointments/767d86f3-b3a4-4420-8058-0722f4615b17

Body: none

This request does not have a body

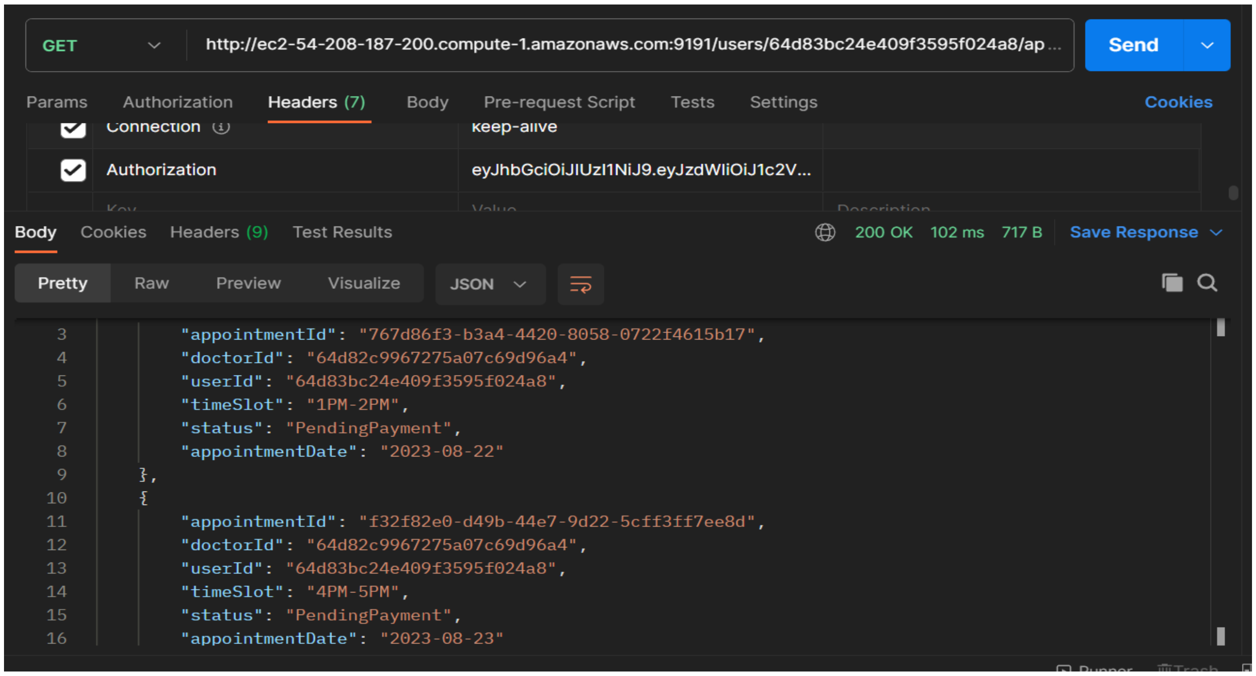
200 OK 236 ms 506 B

```json
{
    "appointmentId": "767d86f3-b3a4-4420-8058-0722f4615b17",
    "doctorId": "64d82c9967275a07c69d96a4",
    "userId": "64d83bc24e409f3595f024a8",
    "timeSlot": "1PM-2PM",
    "status": "PendingPayment",
    "appointmentDate": "2023-08-22"
}
```

Working Screenshots

e.) URI: /users/{userId}/appointments & HTTP method: GET

- This endpoint is responsible for retrieving the details of all the appointments corresponding to a userId.



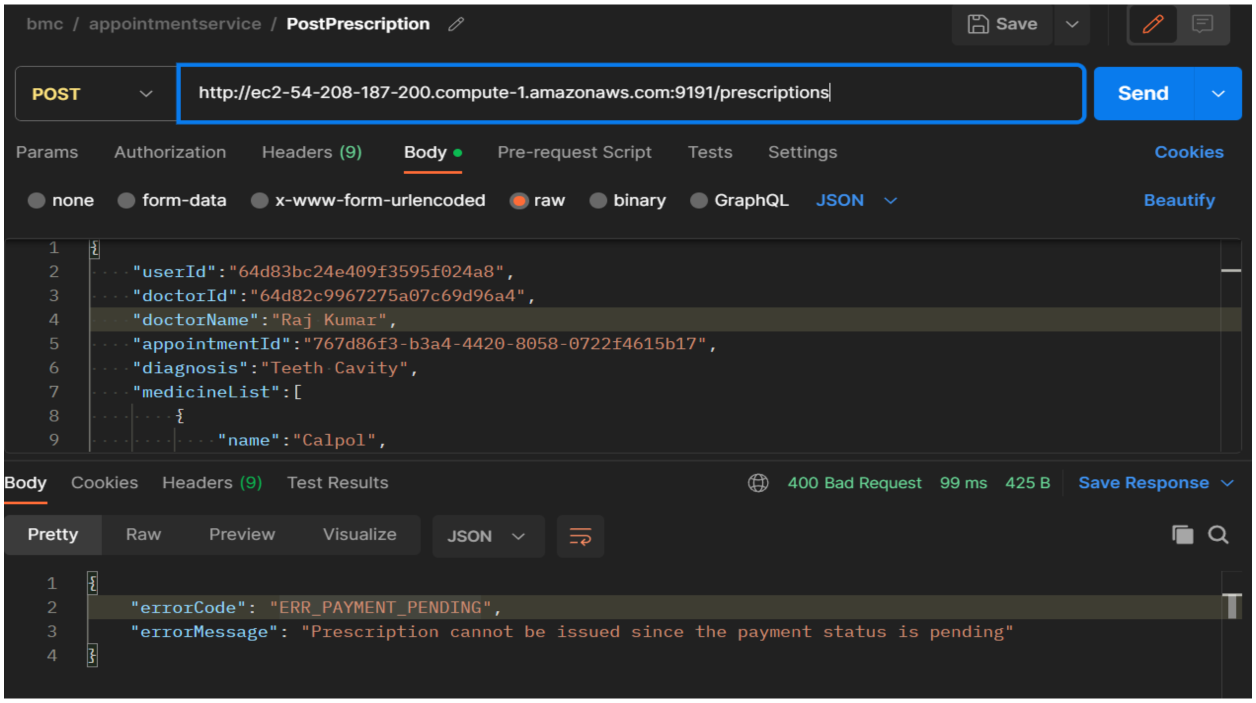
GET http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/users/64d83bc24e409f3595f024a8/ap...

Key	Value
Connection	keep-alive
Authorization	eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2V...

200 OK 102 ms 717 B

```
        "appointmentId": "767d86f3-b3a4-4420-8058-0722f4615b17",
        "doctorId": "64d82c9967275a07c69d96a4",
        "userId": "64d83bc24e409f3595f024a8",
        "timeSlot": "1PM-2PM",
        "status": "PendingPayment",
        "appointmentDate": "2023-08-22"
    },
    {
        "appointmentId": "f32f82e0-d49b-44e7-9d22-5cff3ff7ee8d",
        "doctorId": "64d82c9967275a07c69d96a4",
        "userId": "64d83bc24e409f3595f024a8",
        "timeSlot": "4PM-5PM",
        "status": "PendingPayment",
        "appointmentDate": "2023-08-23"
```

f.) URI: /prescriptions & HTTP method: POST & Pending Payment



bmc / appointmentservice / PostPrescription

POST http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/prescriptions

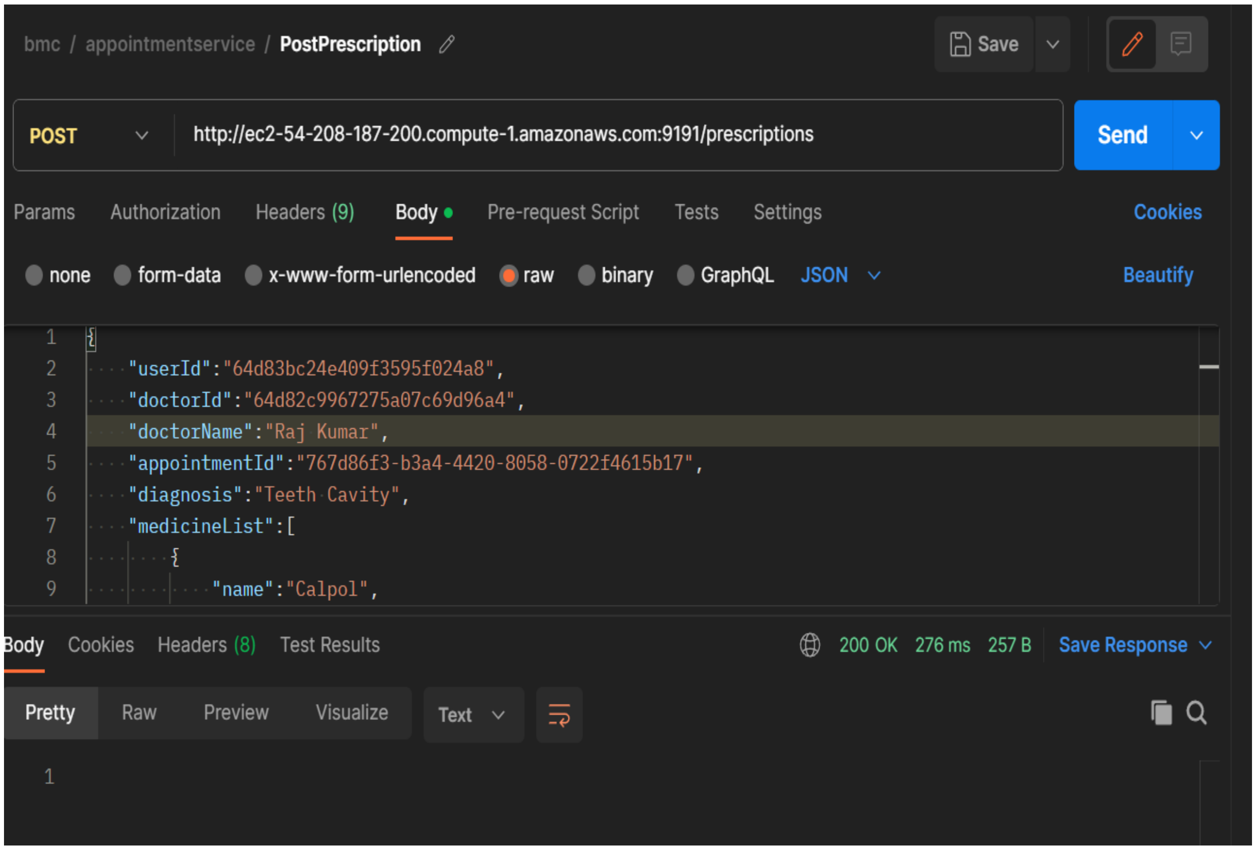
```
{
    "userId":"64d83bc24e409f3595f024a8",
    "doctorId":"64d82c9967275a07c69d96a4",
    "doctorName":"Raj Kumar",
    "appointmentId":"767d86f3-b3a4-4420-8058-0722f4615b17",
    "diagnosis":"Teeth Cavity",
    "medicineList":[
        {
            "name":"Calpol",
```

400 Bad Request 99 ms 425 B

```
{
    "errorCode": "ERR_PAYMENT_PENDING",
    "errorMessage": "Prescription cannot be issued since the payment status is pending"
}
```

Working Screenshots

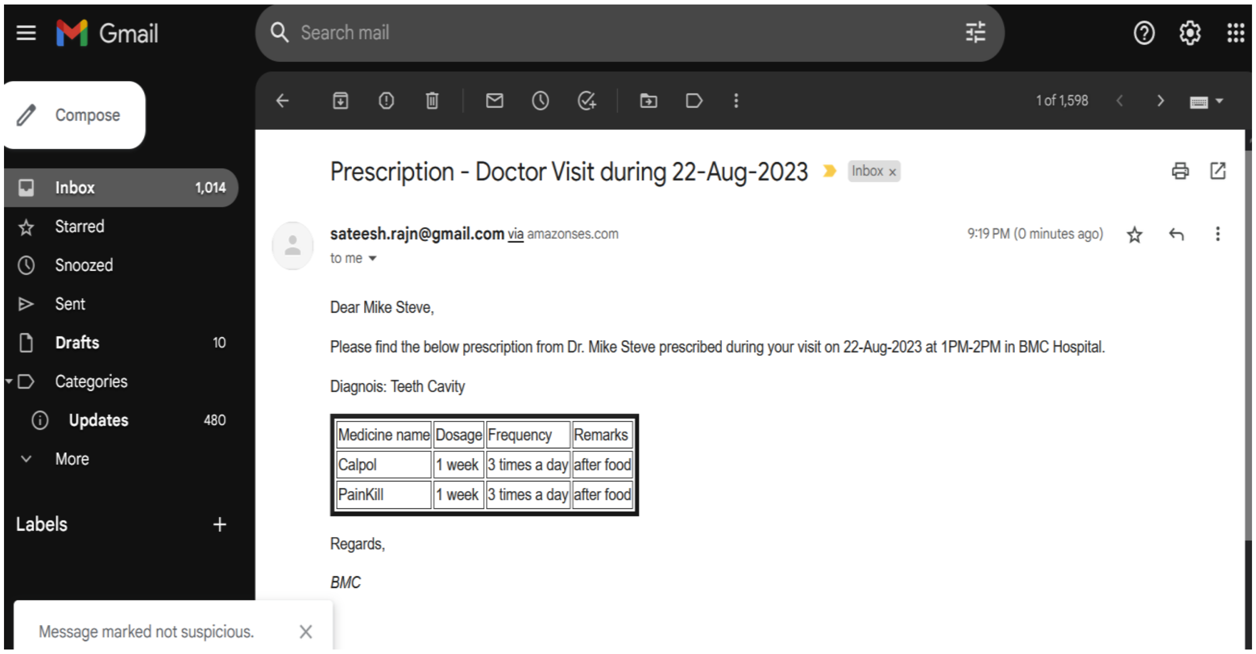
g.) URI: /prescriptions & HTTP method: POST & Confirmed Payment



POST http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/prescriptions

```
{
    "userId":"64d83bc24e409f3595f024a8",
    "doctorId":"64d82c9967275a07c69d96a4",
    "doctorName":"Raj Kumar",
    "appointmentId":"767d86f3-b3a4-4420-8058-0722f4615b17",
    "diagnosis":"Teeth Cavity",
    "medicineList":[
        {
            "name":"Calpol",
```

A prescription email is triggered to the user.



Prescription - Doctor Visit during 22-Aug-2023

sateesh.rajn@gmail.com via amazonses.com
to me
9:19 PM (0 minutes ago)

Dear Mike Steve,

Please find the below prescription from Dr. Mike Steve prescribed during your visit on 22-Aug-2023 at 1PM-2PM in BMC Hospital.

Diagnois: Teeth Cavity

Medicine name	Dosage	Frequency	Remarks
Calpol	1 week	3 times a day	after food
PainKill	1 week	3 times a day	after food

Regards,

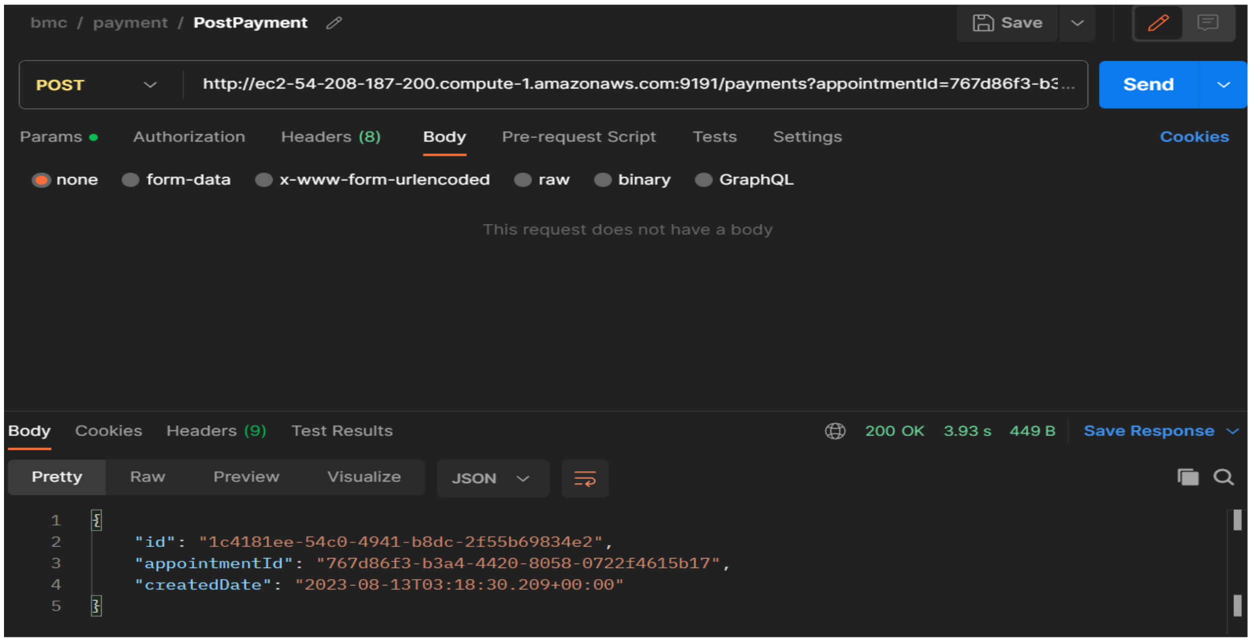
BMC

Working Screenshots

5. Payment Services

a.) URI: /payments?appointmentId=<the appointmentId for which you want to make a payment> & HTTP method: POST

- This endpoint is responsible for making payments.



bmc / payment / **PostPayment**

POST http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/payments?appointmentId=767d86f3-b3...

Body: none

This request does not have a body

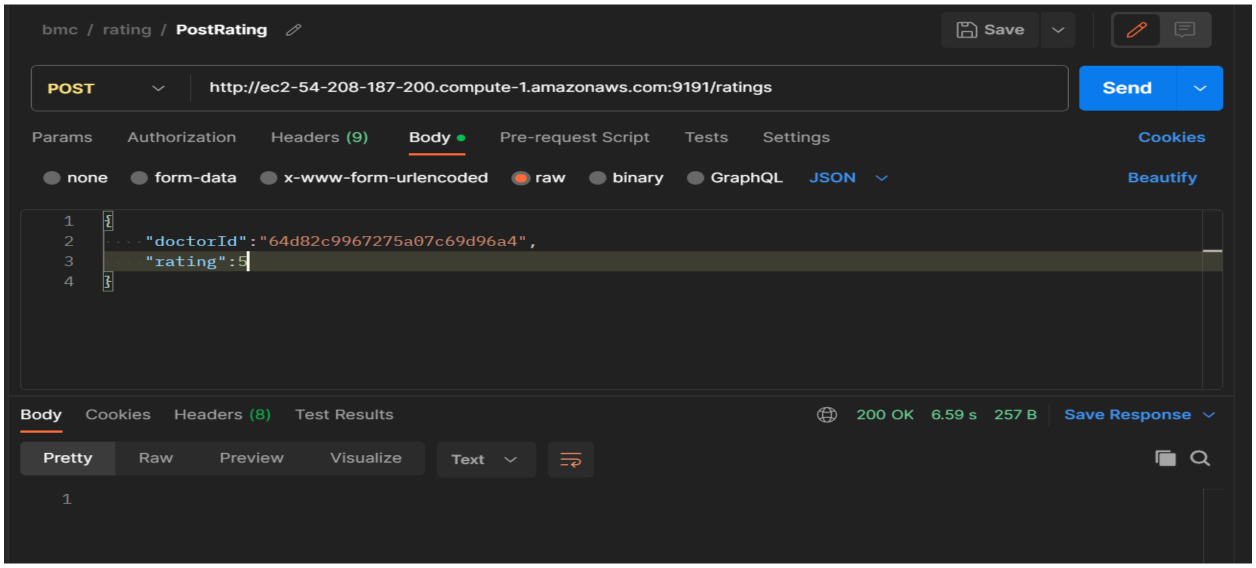
200 OK 3.93 s 449 B

```
{
    "id": "1c4181ee-54c0-4941-b8dc-2f55b69834e2",
    "appointmentId": "767d86f3-b3a4-4420-8058-0722f4615b17",
    "createdDate": "2023-08-13T03:18:30.209+00:00"
}
```

6.) Rating Services

a.) URI: /ratings & HTTP method: POST

- This endpoint is used by the users to submit the ratings of their experience with the doctor with whom they had an appointment.



bmc / rating / **PostRating**

POST http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/ratings

```
{
    "doctorId":"64d82c9967275a07c69d96a4",
    "rating":5
}
```

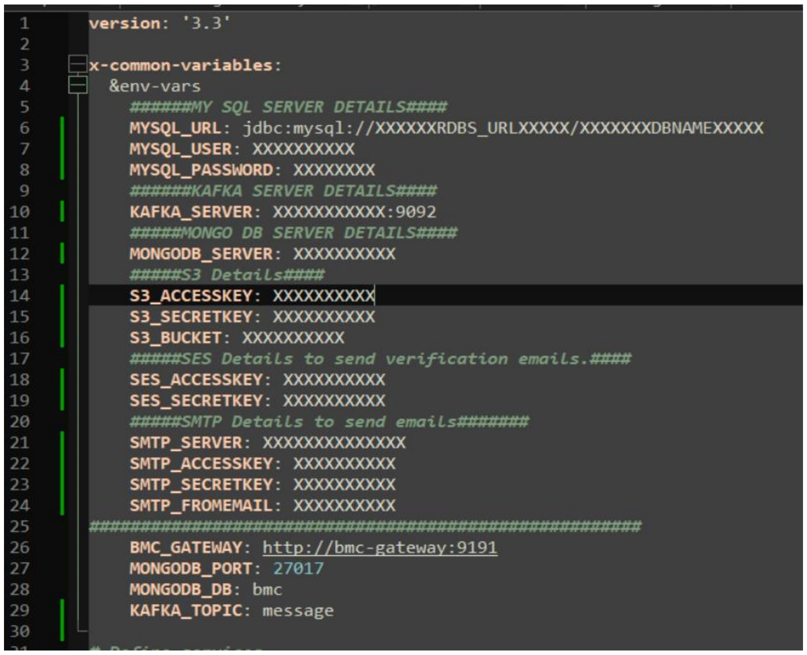
200 OK 6.59 s 257 B

Code Execution

1. Unzip the BookMyConsulation.zip

2. Open the docker-compose.yaml, from line 6 to 24, please provide the correct details of MongoDB server, Kafka server, RDS URL and DB Name, S3 AccessKey, S3 SecretKey, S3 Bucket name, SES AccessKey, S3 SecretKey, SMTP Server name, SMTP AccessKey, SMTP SecretKey, SMTP From email id.

   Note: No changes are needed in the application.properties of any micro services.



```
1   version: '3.3'
2
3   x-common-variables:
4     &env-vars
5       ######MY SQL SERVER DETAILS####
6       MYSQL_URL: jdbc:mysql://XXXXXXRDBS_URLXXXXX/XXXXXXXDBNAMEXXXXX
7       MYSQL_USER: XXXXXXXXX
8       MYSQL_PASSWORD: XXXXXXXX
9       ######KAFKA SERVER DETAILS####
10      KAFKA_SERVER: XXXXXXXXXX:9092
11      #####MONGO DB SERVER DETAILS####
12      MONGODB_SERVER: XXXXXXXXXX
13      #####S3 Details####
14      S3_ACCESSKEY: XXXXXXXXXX
15      S3_SECRETKEY: XXXXXXXXXX
16      S3_BUCKET: XXXXXXXXXX
17      #####SES Details to send verification emails.####
18      SES_ACCESSKEY: XXXXXXXXXX
19      SES_SECRETKEY: XXXXXXXXXX
20      #####SMTP Details to send emails#######
21      SMTP_SERVER: XXXXXXXXXXXXXX
22      SMTP_ACCESSKEY: XXXXXXXXXX
23      SMTP_SECRETKEY: XXXXXXXXXX
24      SMTP_FROMEMAIL: XXXXXXXXXX
25  ######################################################
26      BMC_GATEWAY: http://bmc-gateway:9191
27      MONGODB_PORT: 27017
28      MONGODB_DB: bmc
29      KAFKA_TOPIC: message
30
```

3. The jar files are removed from the solution zip file, so jar files should be generated again using the below steps.

4. Navigate to appointment-service micro service folder,

   Generate jar file using the below command.
   sudo mvn clean install spring-boot:repackage -DskipTests

   Note: -**DskipTests is very important in the above command** as the DB and Kafka server details are provided via docker-compose file, build generation may fail by executing the test scripts while generating the build.

5. Navigate to bmc-gateway micro service folder,

   Generate jar file using the below command.
   sudo mvn clean install spring-boot:repackage -DskipTests

Code Execution

Note: -**DskipTests is very important in the above command** as the DB and Kafka server details are provided via docker-compose file, build generation may fail by executing the test scripts while generating the build.

6. Navigate to doctor-service micro service folder,

   Generate jar file using the below command.

   ```
   sudo mvn clean install spring-boot:repackage -DskipTests
   ```

   Note: -**DskipTests is very important in the above command** as the DB and Kafka server details are provided via docker-compose file, build generation may fail by executing the test scripts while generating the build.

7. Navigate to notification-service micro service folder,

   Generate jar file using the below command.

   ```
   sudo mvn clean install spring-boot:repackage -DskipTests
   ```

   Note: -**DskipTests is very important in the above command** as the DB and Kafka server details are provided via docker-compose file, build generation may fail by executing the test scripts while generating the build.

8. Navigate to payment-service micro service folder,

   Generate jar file using the below command.

   ```
   sudo mvn clean install spring-boot:repackage -DskipTests
   ```

   Note: -**DskipTests is very important in the above command** as the DB and Kafka server details are provided via docker-compose file, build generation may fail by executing the test scripts while generating the build.

9. Navigate to rating-service micro service folder,

   Generate jar file using the below command.

   ```
   sudo mvn clean install spring-boot:repackage -DskipTests
   ```

   Note: -**DskipTests is very important in the above command** as the DB and Kafka server details are provided via docker-compose file, build generation may fail by executing the test scripts while generating the build.

Code Execution

10. Navigate to security-service micro service folder,
    Generate jar file using the below command.
    sudo mvn clean install spring-boot:repackage -DskipTests

    Note: -**DskipTests is very important in the above command** as the DB and Kafka server details are provided via docker-compose file, build generation may fail by executing the test scripts while generating the build.

11. Navigate to user-service micro service folder,
    Generate jar file using the below command.
    sudo mvn clean install spring-boot:repackage -DskipTests

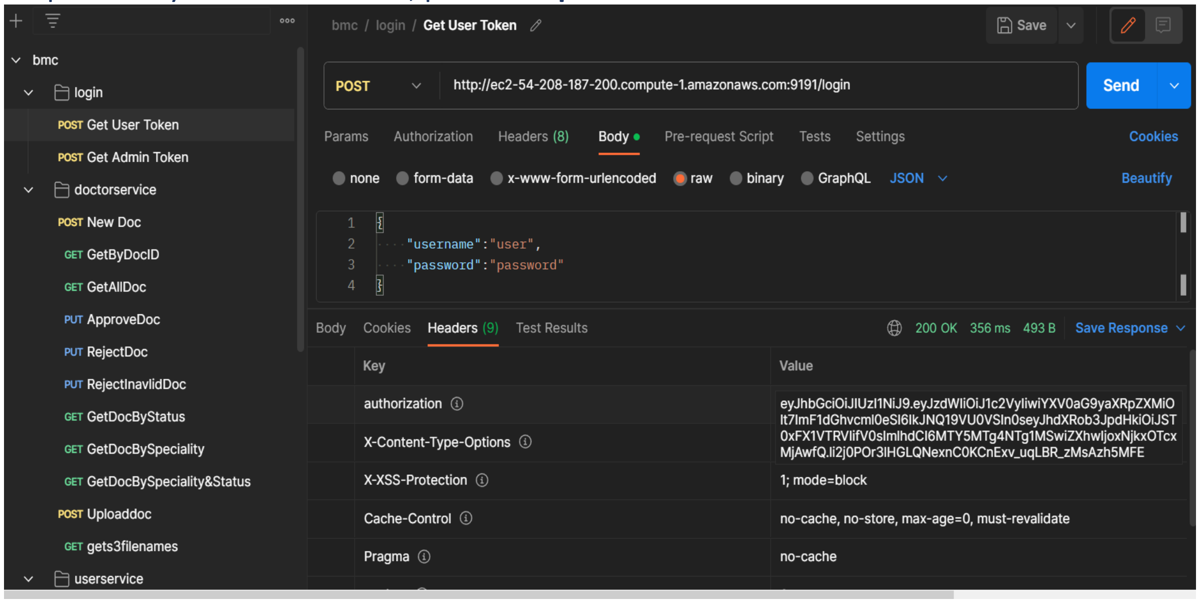
    Note: -**DskipTests is very important in the above command** as the DB and Kafka server details are provided via docker-compose file, build generation may fail by executing the test scripts while generating the build.

12. Navigate to user-service micro service folder, run below command to generate the images and start the container for the Microservices.
    sudo docker-compose up -d

13. Wait for 1-2 mins for all the services to come up.

14. All services will be routed via bmc-gateway services which is in port 9191.
    Sample url: http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/login

15. First generate the Security Token using the Security Services.
    Generating User Token: In the Response Header section -> Authorization -> Token is generated.
    Request body – username:**user**, password:**password**



POST http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/login

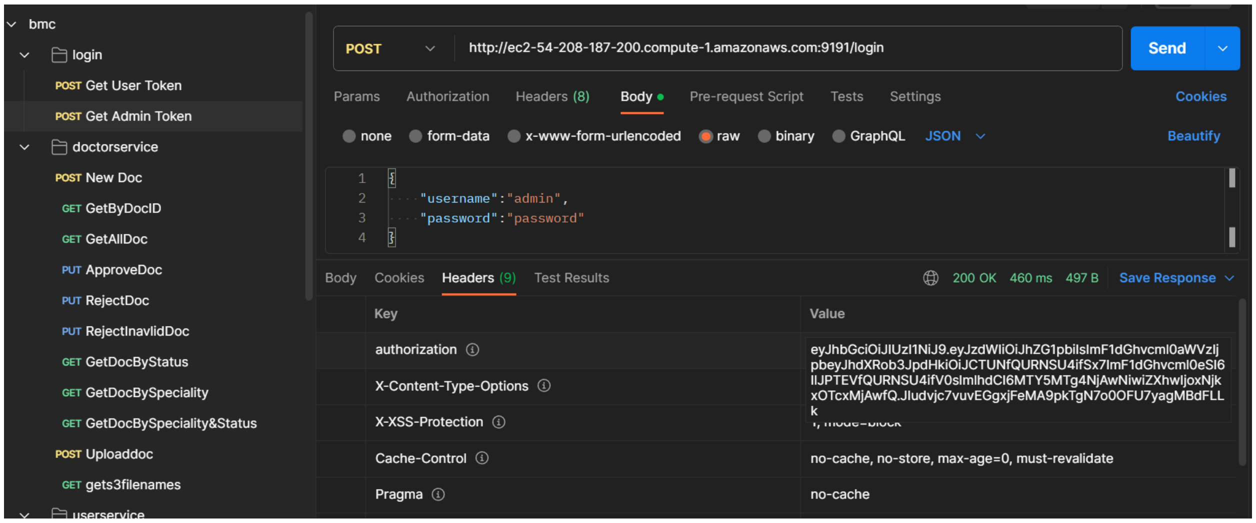
```
{
    "username":"user",
    "password":"password"
}
```

Key	Value
authorization	eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiYXV0aG9yaXRpZXMiOlt7ImF1dGhvcml0eSI6IkJNQ19VU0VSIn0seyJhdXRob3JpdHkiOiJST0xFX1VTRVIifV0sImlhdCI6MTY5MTg4NTg1MSwiZXhwIjoxNjkxOTcxMjAwfQ.Ii2j0POr3lHGLQNexnC0KCnExv_uqLBR_zMsAzh5MFE
X-Content-Type-Options	
X-XSS-Protection	1; mode=block
Cache-Control	no-cache, no-store, max-age=0, must-revalidate
Pragma	no-cache

Code Execution

Generating Admin Token: In the Response Header section -> Authorization -> Token is generated.

Request body – username:**admin**, password:**password**



```
POST  http://ec2-54-208-187-200.compute-1.amazonaws.com:9191/login
```

```
{
    "username":"admin",
    "password":"password"
}
```

200 OK 460 ms 497 B

Key	Value
authorization	eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsImF1dGhvcml0aWVzIjpbeyJhdXRob3JpdHkiOiJCTUNfQURNSU4ifSx7ImF1dGhvcml0eSI6IlJPTEVfQURNSU4ifV0sImlhdCI6MTY5MTg4NjAwNiwiZXhwIjoxNjkxOTcxMjAwfQ.Jludvjc7vuvEGgxjFeMA9pkTgN7o0OFU7yagMBdFLLk
X-Content-Type-Options	
X-XSS-Protection	1; mode=block
Cache-Control	no-cache, no-store, max-age=0, must-revalidate
Pragma	no-cache

16. After generating the User and Admin Tokens, pass this Token in the Header section of the request and start testing.

    While passing the token don't pass **Bearer** word before the token, this approach is adopted as per the screenshots provided in the problem statement.

Explanation of Logic:

1. Solution is approached by using an **External Kafka** server by installing it in an EC2 instance.
2. Separate My -SQL RDS is created AWS and appropriate Inbound rule are added to allow connection from the EC2 instance where Docker is installed.
3. Separate EC2 instance is created AWS and MongoDB software is installed in the server and appropriate Inbound rule are added to allow connection from the EC2 instance where Docker is installed.
4. Separate Ubuntu EC2 instance is created, and Docker software is installed in the server.
5. For each microservices, Dockerfile is written and the corresponding the Jar files are generated.
6. Finally the docker-compose.yaml is generated, where the configuration for each microservices are defined.
7. Once the services are started using the docker-compose up command, all the microservices are started successfully and works as expected.