

Project Explanation

So, we have created 9 Microservices for the Book My Consultation App.

1. Service Registry: service-registry
2. BMC Gateway: bmc-gateway
3. Notification Service: notification-service
4. Rating Service: rating-service
5. User Service: user-service
6. Doctor Service: doctor-service
7. Appointment Service: appointment-service
8. Payment Service: payment-service
9. Security Provider: security-provider

Let us first explain the Deployment Process and then later let us brief you with the API Testing Screenshots, and Code Explanation by Service

Deployment Process

We needed to compose and deploy the whole project. For the same, please find the steps (**with setup screenshots**) that we took for the same:

- First we created 3 elastic IP Addresses that we will use later for our 3 servers.
 - First for the Main Server
 - Second for the Kafka Server
 - Third for the Mongo Server

The screenshot shows the AWS EC2 dashboard with the 'Elastic IP addresses' section. There are three entries listed:

Name	Allocated IPv4 address
ElasticIpBmcServerApp	34.192.75.206
ElasticIpBmcServerKafkaConnect	34.226.187.86
ElasticIpBmcServerMongoConnect	52.73.212.58

- Now, we created a VPC and Security group respectively referring from this [resource](#). Please find the screenshots attached for the Inbound and Outbound rules that we set in

our security groups of our VPC. This also includes My IP Address from our local home desktop/server to connect to both of these instances.

VPC ID: vpc-025dc8c79bf17d00a

State: Available

Tenancy: Default

Default VPC: No

IPv4 CIDR: 10.0.0.0/16

Network Address Usage metrics: Disabled

DNS hostnames: Enabled

Main route table: rtb-071d5b020a41bcd6d

IPv6 pool: -

Owner ID: 609194545037

DNS resolution: Enabled

Main network ACL: acl-063fde5f2a11427ac

IPv6 CIDR (Network border group): -

Resource map:

- VPC:** Show details (Your AWS virtual network)
- Subnets (1):** Subnets within this VPC
- Route tables (2):** Route network traffic to resources
- Network connections (2):** Connections to other networks

Security group name: BmcServerGroup

Security group ID: sg-03f47b6ae82a5d7c9

Description: set inbound rules

VPC ID: vpc-025dc8c79bf17d00a

Owner: 609194545037

Inbound rules count: 30 Permission entries

Outbound rules count: 1 Permission entry

Outbound rules (1/1):

Name	Security group rule...	IP version	Type	Protocol	Port range	Destination
-	sgr-0210528281d7f23c	IPv4	All traffic	All	All	0.0.0.0/0

The screenshot shows the AWS VPC console with the 'Security Groups' section selected. A specific security group, 'sg-03f47b6ae82a5d7c9 - BmcServerGroup', is viewed. The 'Details' tab is active, displaying information such as the security group name ('BmcServerGroup'), ID ('sg-03f47b6ae82a5d7c9'), owner ('609194545037'), and various rule counts. The 'Inbound rules' tab is selected, showing a table of 30 entries. Each entry includes columns for Name, Security group rule, IP version, Type, Protocol, Port range, Source, and Description. The table lists various IP addresses and port ranges, mostly using TCP and IPv4.

- Then we launched 3 EC2 instances in our VPC referring from this [resource](#). Please find the configuration for the same.
 - First EC2 instance is for running the **Main server** and we choose Ubuntu operating system for the same server. We created a t2.large server for the same with 30 GB Magnetic standard storage.
 - Second EC2 instance is for running the **Kafka server** and we choose the Amazon AMI operating system for the same server. We created a t2.medium server for the same with 8 GB Magnetic standard storage.
 - Third EC2 instance is for running the **Mongo server** and we choose the Amazon AMI operating system for the same server. We created a t2.medium server for the same with 8 GB Magnetic standard storage.

The screenshot shows the AWS EC2 Instances page. On the left, a sidebar lists various AWS services and navigation links. The main content area displays a table of running instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
BmcServerKafkaConnect	i-001da872da53113d1	Running	t2.medium	2/2 checks passed	No alarms	us-east-1a
BmcServerApp	i-014b4cc40ff9c5d4	Running	t2.large	2/2 checks passed	No alarms	us-east-1a
BmcServerMongoConnect	i-05525956fd1a52cb4	Running	t2.medium	2/2 checks passed	No alarms	us-east-1a

At the bottom, a "Select an instance" dropdown is visible.

- Now we allocate the Elastic IP Addresses to the EC2 server per server so that we can use the same as pointers for our code.

The screenshot shows the AWS Elastic IP addresses page. The sidebar includes a link to the "Elastic IPs" section under "Network & Security". The main table lists the allocated IP addresses:

Name	Allocated IPv4 add...	Type	Allocation ID	Associated instance ID	Private IP address	Actions
ElasticIpBmcServerApp	34.192.75.206	Public IP	eipalloc-024c75b499ee76ee4	i-014b4cc40ff9c5d4	10.0.8.76	
ElasticIpBmcServerKafkaConnect	34.226.187.86	Public IP	eipalloc-089ef5320e687a4e2	i-001da872da53113d1	10.0.15.165	
ElasticIpBmcServerMongoConnect	52.73.212.58	Public IP	eipalloc-02b569439f19f4e2	i-05525956fd1a52cb4	10.0.8.162	

A note at the bottom encourages users to "View IP address usage and recommendations to release unused IPs with Public IP insights."

- We store the AWS key-pairs for the instances in one level up directory of the `BookMyConsultation` folder. In our case, it's `book-my-consultation-monorepo`. We store our key pairs in `book-my-consultation-monorepo/aws-keys` (that we won't be

sharing as not needed and concerns privacy), whereas the project lives at [book-my-consultation-monorepo/BookMyConsultation](#) folder.

- Now, we create a single RDS instance referring from this [resource](#). Please find the screenshot for the running Database server.

The screenshot shows the AWS RDS console for the 'bmc-db-instance' database. The left sidebar has 'Databases' selected. The main area shows the 'Summary' tab for the database, including its identifier, CPU usage (2.51%), status (Available), role (Instance), current activity (0 connections), engine (MySQL Community), class (db.t3.micro), and region (us-east-1b). Below the summary, there are tabs for 'Connectivity & security', 'Monitoring', 'Logs & events', 'Configuration', 'Maintenance & backups', and 'Tags'. The 'Connectivity & security' tab is active, displaying details about the endpoint, networking (VPC, subnet group, subnets), and security (VPC security groups, certificate authority). The VPC security group is set to 'default (sg-2fbefafa69)' and is marked as 'Active'. The certificate authority is 'rds-ca-2019'. The bottom of the page includes standard AWS footer links like CloudShell, Feedback, Language, Privacy, Terms, and Cookie preferences.

- The RDS instance were set with following specifications
 - User name: admin
 - Password: bmcpassword
- Now for the RDS Instance, we will use the default VPC and Security group that were given by AWS for the region. For this specific security group, please find the screenshots attached for the Inbound and Outbound rules that we set in this one. This not only includes My IP Address from our local home desktop/server to connect to the database instance but also includes the IP Address of the Main Server EC2 instance to connect to the database instance respectively.

The screenshot shows the AWS EC2 Security Groups page. A search bar at the top has "sg-2fbef69" entered. The main table lists one security group: "sg-2fbef69" (default) with VPC ID "vpc-2cda4e56". The "Outbound rules" tab is selected. Below it, a message says "You can now check network connectivity with Reachability Analyzer" and a "Run Reachability Analyzer" button.

Name	Security group ID	Security group name	VPC ID	Description	Owner	Inbound rules count
-	sg-2fbef69	default	vpc-2cda4e56	default VPC security gr...	609194545037	3 Permission entries

sg-2fbef69 - default

Details | Inbound rules | **Outbound rules** | Tags

You can now check network connectivity with Reachability Analyzer

Outbound rules (1/1)

Name	Security group rule...	IP version	Type	Protocol	Port range	Destination
-	sgr-0eeb417787656c4...	IPv4	All traffic	All	All	0.0.0.0/0

This screenshot is identical to the one above, except the "Inbound rules" tab is selected instead of "Outbound rules". It shows three inbound rules: one from "sg-2fbef69 / default" (All traffic, All, All), and two from "MySQL/Aurora" (TCP port 3306, TCP port 3306).

Name	Security group rule...	IP version	Type	Protocol	Port range	Source
-	sgr-0f45ea244798d8ff4	-	All traffic	All	All	sg-2fbef69 / default
-	sgr-055e7655cc6cb926d	IPv4	MySQL/Aurora	TCP	3306	49.36.187.35/32
-	sgr-001cba52e145630...	IPv4	MySQL/Aurora	TCP	3306	34.192.75.206/32

- Now we will login to Amazon Simple Email Service (SES) and generate SMTP credentials (IAM User) for email delivery referring from this [resource](#). We will also create Verified Identities (as seen in the below screenshot) for the same process.

The screenshot shows the AWS Amazon SES configuration interface. On the left, a sidebar lists various configuration options like Account dashboard, Reputation metrics, SMTP settings, Configuration sets, Dedicated IPs, Email templates, Suppression list, Cross-account notifications, and Email receiving. The 'Verified identities' option is selected. The main content area is titled 'Verified identities' and contains a message about identity status updates. Below this is a table titled 'Identities (4) Info' showing four entries:

Identity	Identity type	Identity status
harry@kibanu.com	Email address	Verified
harry+40423mohitgupta@kibanu.com	Email address	Verified
harry+36045sanjeevsharma@kibanu.com	Email address	Verified
harry+27232ayushgupta@kibanu.com	Email address	Verified

- Now it's time to create an IAM user which will help our app to upload files to S3 bucket.

The screenshot shows the AWS IAM User details interface. The user is named 's3-public-user'. The 'Summary' section provides basic information: ARN (arn:aws:iam:609194545037:user/s3-public-user), Console access (Disabled), Last console sign-in (Never used, 24 hours old), and Access key 1 (AKIAV3WTBGGQAYUW2D6 - Active). The 'Permissions' tab is selected, showing a single policy attached: 'AmazonS3FullAccess'. The JSON code for this policy is displayed:

```

1: {
2:   "Version": "2012-10-17",
3:   "Statement": [
4:     {
5:       "Effect": "Allow",
6:       "Action": [
7:         "s3:*",
8:         "s3-object-lambda-*"
9:       ],
10:      "Resource": "*"
11:    }
12:  ]
13: }

```

- Now it is time to log in and gain access to our created instances thus we now log in to our instances, referring from this [resource](#).
- First we connect to the **Mongo Server Instance**, we install MongoDB referring from this [resource](#).

- We now run `systemctl status mongod` to ensure that the **Mongo Server** is running correctly.

```
x ec2-user@ip-10-0-8-162:~ (ssh)
https://aws.amazon.com/amazon-linux-2/
-bash: warning: setlocale: LC_CTYPE: cannot change locale (UTF-8): No such file or directory
[ec2-user@ip-10-0-8-162 ~]$ 
[ec2-user@ip-10-0-8-162 ~]$ systemctl status mongod
● mongod.service - MongoDB Database Server
   Loaded: loaded (/usr/lib/systemd/system/mongod.service; enabled; vendor preset: disabled)
     Active: active (running) since Sun 2023-08-13 09:53:49 UTC; 2h 5min ago
       Docs: https://docs.mongodb.org/manual
 Main PID: 3175 (mongod)
    CGroup: /system.slice/mongod.service
           └─3175 /usr/bin/mongod -f /etc/mongod.conf

Aug 13 09:53:49 ip-10-0-8-162.ec2.internal systemd[1]: Started MongoDB Database Server.
Aug 13 09:53:57 ip-10-0-8-162.ec2.internal mongod[3175]: {"t":{"$date":"2023-08-13T09:53:57.678Z"},"s":"I", "c":"CONTROL", "id":7484500, "ctx": "-", "msg":"Env...o false"}
Hint: Some lines were ellipsized, use -l to show in full.
[ec2-user@ip-10-0-8-162 ~]$ 
```

- Now we connect to the **Kafka Server Instance** by opening a new terminal to setup the Notification Process. In the **Kafka server** EC2 instance, we install Kafka referring from this [resource](#). We will cd into the kafka directory.
 - Now we will run the command: `bin/zookeeper-server-start.sh config/zookeeper.properties` to start the **Zookeeper**

- Now will open a new terminal and cd into the kafka directory.
 - Now we will run the command: bin/kafka-server-start.sh config/server.properties to start the **Kafka Server**.

```

× ec2-user@ip-10-0-15-165:~/kafka_2.13-3.5.0 (ssh)
which 100 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2023-08-13 12:05:21,445] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-43 in 100 milliseconds for epoch 0, of which 100 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2023-08-13 12:05:21,445] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-13 in 100 milliseconds for epoch 0, of which 100 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2023-08-13 12:05:21,445] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-28 in 100 milliseconds for epoch 0, of which 100 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2023-08-13 12:05:21,445] INFO [GroupCoordinator 0]: Member consumer-notification-4-5e23f972-d7f6-4957-adfd-b7b150b4bf06 in group notification has failed, removing it from the group (kafka.coordinator.group.GroupCoordinator)
[2023-08-13 12:05:31,429] INFO [GroupCoordinator 0]: Preparing to rebalance group notification in state PreparingRebalance with old generation 46 (__consumer_offsets-3) (reason: removing member consumer-notification-4-5e23f972-d7f6-4957-adfd-b7b150b4bf06 on heartbeat expiration) (kafka.coordinator.group.GroupCoordinator)
[2023-08-13 12:05:31,431] INFO [GroupCoordinator 0]: Member consumer-notification-2-34998521-5bd5-4c07-8a6b-dcc7b5eefab0 in group notification has failed, removing it from the group (kafka.coordinator.group.GroupCoordinator)
[2023-08-13 12:05:31,431] INFO [GroupCoordinator 0]: Member consumer-notification-3-a415e281-7570-4a25-b548-ce56d7ca8a49 in group notification has failed, removing it from the group (kafka.coordinator.group.GroupCoordinator)
[2023-08-13 12:05:31,431] INFO [GroupCoordinator 0]: Member consumer-notification-1-06c18ce8-4d21-400b-bc85-590c67ca167f in group notification has failed, removing it from the group (kafka.coordinator.group.GroupCoordinator)
[2023-08-13 12:05:31,433] INFO [GroupCoordinator 0]: Group notification with generation 47 is now empty (__consumer_offsets-3) (kafka.coordinator.group.GroupCoordinator)
[2023-08-13 12:05:31,463] INFO [GroupCoordinator 0]: Member consumer-doctor-1-08b90436-e8e3-4df7-9448-eb5c1f2bcd4 in group doctor has failed, removing it from the group (kafka.coordinator.group.GroupCoordinator)
[2023-08-13 12:05:31,463] INFO [GroupCoordinator 0]: Preparing to rebalance group doctor in state PreparingRebalance with old generation 18 (__consumer_offsets-25) (reason: removing member consumer-doctor-1-08b90436-e8e3-4df7-9448-eb5c1f2bcd4 on heartbeat expiration) (kafka.coordinator.group.GroupCoordinator)
[2023-08-13 12:05:31,464] INFO [GroupCoordinator 0]: Group doctor with generation 19 is now empty (__consumer_offsets-25) (kafka.coordinator.group.GroupCoordinator)
[2023-08-13 12:05:31,465] INFO [GroupCoordinator 0]: Member consumer-appointment-1-a3a04b0b-0da3-400d-8ab0-94493bc011f1 in group appointment has failed, removing it from the group (kafka.coordinator.group.GroupCoordinator)
[2023-08-13 12:05:31,465] INFO [GroupCoordinator 0]: Preparing to rebalance group appointment in state PreparingRebalance with old generation 17 (__consumer_offsets-47) (reason: removing member consumer-appointment-1-a3a04b0b-0da3-400d-8ab0-94493bc011f1 on heartbeat expiration) (kafka.coordinator.group.GroupCoordinator)
[2023-08-13 12:05:31,465] INFO [GroupCoordinator 0]: Group appointment with generation 18 is now empty (__consumer_offsets-47) (kafka.coordinator.group.GroupCoordinator)
|

```

- Now will open a new terminal and cd into the kafka directory to ensure whether **Zookeeper** and **Kafka Server** has started or not by running `nc -vz localhost 2181` and `nc -vz localhost 9092` respectively.

```

× ec2-user@ip-10-0-15-165:~/kafka_2.13-3.5.0 (ssh)
[ec2-user@ip-10-0-15-165 kafka_2.13-3.5.0]$ nc -vz localhost 2181
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connected to 127.0.0.1:2181.
Ncat: 0 bytes sent, 0 bytes received in 0.01 seconds.
[ec2-user@ip-10-0-15-165 kafka_2.13-3.5.0]$ nc -vz localhost 9092
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connected to 127.0.0.1:9092.
Ncat: 0 bytes sent, 0 bytes received in 0.01 seconds.
[ec2-user@ip-10-0-15-165 kafka_2.13-3.5.0]$ |

```

- Now let's connect to the **Main Server** by opening a new terminal, we will now run `sudo docker compose up --build` to deploy our Multi container project.

```

ubuntu@ip-10-0-8-76:~/BookMyConsultation (ssh)
doctorService | 2023-08-13 12:11:38.220 INFO 1 --- [ntainer#0-0-C-1] o.s.k.l.KafkaMessageListenerContainer : doctor: partitions assigned: [doctor-0]
serviceregistry | 2023-08-13 12:11:38.265 INFO 1 --- [nio-8761-exec-7] c.n.e.registry.AbstractInstanceRegistry : Registered instance USER-SERVICE/171d543e9504:USER-SERVICE:8083 with status UP (replication=false)
userservice | 2023-08-13 12:11:38.268 INFO 1 --- [nioReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_USER-SERVICE/171d543e9504:USER-SERVICE:8083 - registration status: 204
bmcgateway | 2023-08-13 12:11:56.793 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Disable delta property : false
bmcgateway | 2023-08-13 12:11:56.794 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Single vip registry refresh property : null
bmcgateway | 2023-08-13 12:11:56.794 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Force full registry fetch : false
bmcgateway | 2023-08-13 12:11:56.794 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Application is null : false
bmcgateway | 2023-08-13 12:11:56.794 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Registered Applications size is zero : true
bmcgateway | 2023-08-13 12:11:56.794 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Application version is -1: false
bmcgateway | 2023-08-13 12:11:56.794 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Getting all instance registry info from the eureka server
bmcgateway | 2023-08-13 12:11:56.807 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : The response status is 200
serviceregistry | 2023-08-13 12:11:58.596 INFO 1 --- [a-EvictionTimer] c.n.e.registry.AbstractInstanceRegistry : Running the evict task with compensationTime 0ms
securityprovider | 2023-08-13 12:11:58.747 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Disable delta property : false
securityprovider | 2023-08-13 12:11:58.747 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Single vip registry refresh property : null
securityprovider | 2023-08-13 12:11:58.747 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Force full registry fetch : false
securityprovider | 2023-08-13 12:11:58.747 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Application is null : false
securityprovider | 2023-08-13 12:11:58.747 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Registered Applications size is zero : true
securityprovider | 2023-08-13 12:11:58.747 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Application version is -1: false
securityprovider | 2023-08-13 12:11:58.747 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Getting all instance registry info from the eureka server
securityprovider | 2023-08-13 12:11:58.762 INFO 1 --- [freshExecutor-0] com.netflix.discovery.DiscoveryClient : The response status is 200

```

- We can confirm that all servers by opening a new terminal and running the command `sudo docker ps -a` respectively.
- Finally our project has been deployed, the Images for all the services are created, and all the containers are running, we can confirm the same by visiting PORT 8761 from our **Main Server** EC2 instance's Elastic IP (see attached screenshot). In our case it was **34.192.75.206:8761** respectively.

The screenshot shows the Spring Eureka dashboard at the URL `34.192.75.206:8761`. The interface includes:

- System Status:** Shows Environment (N/A), Data center (N/A), Current time (2023-08-13T12:15:44+0000), Uptime (00:05), Lease expiration enabled (true), Renews threshold (13), and Renews (last min) (14).
- DS Replicas:** A table listing instances registered with Eureka across various services.
- Instances currently registered with Eureka:**

Application	AMIs	Availability Zones	Status
APPOINTMENT-SERVICE	n/a (1)	(1)	UP (1) - ea955cd7ccf7:APPOINTMENT-SERVICE:8082
BMC-GATEWAY	n/a (1)	(1)	UP (1) - 1a580fb702c5:BMC-GATEWAY:9191
DOCTOR-SERVICE	n/a (1)	(1)	UP (1) - b7cbfd1dcef4:DOCTOR-SERVICE:8081
PAYMENT-SERVICE	n/a (1)	(1)	UP (1) - 6afeb030b6d9:PAYMENT-SERVICE:8086
RATING-SERVICE	n/a (1)	(1)	UP (1) - c359714b73b6:RATING-SERVICE:8084
SECURITY-PROVIDER	n/a (1)	(1)	UP (1) - bb4f6c56cccd4:SECURITY-PROVIDER:8088
USER-SERVICE	n/a (1)	(1)	UP (1) - 171d543e9504:USER-SERVICE:8083
- General Info:** A table showing system metrics like total-available-memory (257mb), number-of-cpus (2), current-memory-usage (45mb (17%)), and server-uptime (00:05).

API Testing Screenshots

- Now it's time to test whether APIs are being hit correctly using postman or not.
- First let us generate a JWT token for the ADMIN respectively.

The screenshot shows the Postman application interface. In the left sidebar, under 'My Workspace', there is a collection named 'BookMyConsultation' which contains several sub-services like 'user-service', 'rating-service', 'appointment-service', and 'doctor-service'. The 'user-service' section has various API endpoints such as 'Fetch the details of the user...', 'POST Register the User', 'POST Upload the documents', etc. The main workspace shows a POST request to 'http://34.192.75.206:8088/foodDelivery/security/generate-token'. The 'Body' tab is selected, showing a JSON payload with 'username' set to 'admin-user@abc.com' and 'password' set to 'Admin@123'. Below the body, the response status is 200 OK with a response time of 1446 ms and a size of 684 B. The response body is a long JSON token.

- Now let us generate a JWT token for the USER respectively.

This screenshot is similar to the previous one but for a 'user'. It shows the same Postman interface with the 'user-service' section of the 'BookMyConsultation' collection. A POST request is made to the same endpoint 'http://34.192.75.206:8088/foodDelivery/security/generate-token'. The 'Body' tab shows a JSON payload with 'username' set to 'normal-user@abc.com' and 'password' set to 'Test@123'. The response status is 200 OK with a response time of 970 ms and a size of 684 B. The response body is a long JSON token.

- NOW we will need to send a request to `POST /doctors` respectively. The Request body will contain `firstName`, `lastName`, `dob`, `emailID`, `mobile`, and `PAN` respectively. If the token has the role of either `USER` or `ADMIN`, then the endpoint can be accessed.
- But before sending the same, let us send an invalid request by deliberately not sending `firstName` and `lastName` respectively. We will see what's been seen in the below screenshot, an error will be thrown by the API.

The screenshot shows the Postman interface with the following details:

- Collection:** My Workspace
- Request:**
 - Method:** POST
 - URL:** `http://34.192.75.206:8081/doctors`
 - Body:** JSON (Pretty)


```

1   "dob": "1986-01-18",
2   ...
3   ...
4   ...
5   ...
6   ...
7   ...
  
```
- Response:**
 - Status:** 400 Bad Request
 - Body:**

```

1   {
2     "errorCode": "ERR_INVALID_INPUT",
3     "errorMessage": "Invalid input. Parameter name: ",
4     "errorFields": [
5       "Last Name",
6       "First Name"
7     ]
8   }
  
```

- Now it's time to send a Valid API request to `POST /doctors` respectively.

The screenshot shows the Postman interface. On the left, the 'My Workspace' sidebar lists collections like 'BookMyConsultation' and 'user-service'. The main area shows a POST request to 'http://34.192.75.206:8081/doctors'. The 'Body' tab contains the following JSON:

```

1
2   ...
3     "firstName": "Mohit",
4     "lastName": "Gupta",
5     "dob": "1986-01-18",
6     "emailId": "harry+40423@mohitgupta@kibanu.com",
7     "mobile": "1234541175",
8     "pan": "BDAL41175K"
9
10
11
12

```

The response status is 201 Created. The 'Test Results' section shows the returned JSON document:

```

1
2   {
3     "id": "64d8ce1fcc47f56ca941f5a3",
4     "firstName": "Mohit",
5     "lastName": "Gupta",
6     "speciality": "GENERAL_PHYSICIAN",
7     "dob": "1986-01-18",
8     "mobile": "1234541175",
9     "emailId": "harry+40423@mohitgupta@kibanu.com",
10    "pan": "BDAL41175K",
11    "status": "Pending",
12    "registrationDate": "2023-08-13"
13
14

```

- We can check the Doctor's database to confirm that the doctor collection has been successfully added.

The screenshot shows the MongoDB shell interface. The command `db.getCollection("doctor").find({})` is run, and the results are displayed in the 'Raw shell output' and 'Find Query (line 1)' panes. The selected document is shown in the 'Tree View' pane:

Key	Value	Type
<code>_id</code>	<code>64d8ce1fcc47f56ca941f5a3</code>	<code>ObjectID</code>
<code>firstName</code>	Mohit	<code>String</code>
<code>lastName</code>	Gupta	<code>String</code>
<code>speciality</code>	GENERAL_PHYSICIAN	<code>String</code>
<code>dob</code>	1986-01-18	<code>String</code>
<code>mobile</code>	1234541175	<code>String</code>
<code>emailId</code>	harry+40423@mohitgupta@kibanu.com	<code>String</code>
<code>pan</code>	BDAL41175K	<code>String</code>
<code>status</code>	Pending	<code>String</code>
<code>registrationDate</code>	2023-08-13	<code>String</code>
<code>averageRating</code>	0.0	<code>Double</code>
<code>_class</code>	<code>bookmyconsultation.doctorservice.entity.DoctorEntity</code>	<code>String</code>

- Now let's check our email to confirm that the email has been received.



Welcome Email

Inbox ×



harry@kibantu.com via amazoneses.com
to harry+40423mohitgupta ▾

Hi Mohit

Congratulations !!! Your Registration is confirmed

Regards,

Upgrad

Reply

Forward

- NOW based on the id returned from the POST /doctors response, we will look to send a request to GET /doctors/{doctorId} respectively to make sure of the contents. If the token has the role of either USER or ADMIN, then the endpoint can be accessed.

- But before sending the same, let us send an invalid request by deliberately not sending the correct `doctorId`. We will see what's been seen in the below screenshot, an error will be thrown by the API.

The screenshot shows the Postman interface with a collection named "BookMyConsultation / doctor-service". A GET request is selected with the URL `http://34.192.75.206:8081/doctors/64d8d471cc47f56ca941f5a6`. The response status is 404 Not Found, and the JSON body contains:

```

1  {
2    "errorCode": "ERR_RESOURCE_NOT_FOUND",
3    "errorMessage": "Requested resource is not available",
4    "errorFields": null
5  }

```

- Now it's time to send a Valid API request to `GET /doctors/{doctorId}` respectively.

The screenshot shows the Postman interface with the same collection. A GET request is selected with the URL `http://34.192.75.206:8081/doctors/64d8ce1fcc47f56ca941f5a3`. The response status is 201 Created, and the JSON body contains:

```

1  {
2    "id": "64d8ce1fcc47f56ca941f5a3",
3    "firstName": "Mohit",
4    "lastName": "Gupta",
5    "specialty": "GENERAL_PHYSICIAN",
6    "dob": "1986-01-18",
7    "mobile": "1234567890",
8    "emailId": "harry40423mohitgupta@kibanu.com",
9    "pan": "B0AL41175K",
10   "status": "Pending",
11   "registrationDate": "2023-08-13"
12

```

- Now the doctor will be uploading the documents. Behind the scenes, it will be uploaded to an S3 bucket.
- For the same let's send a `POST /doctors/{doctorId}/document` respectively. For the Request body, we will select `formData` this time, and upload the respective files that need to be uploaded. If the token has the role of either `USER` or `ADMIN`, then the endpoint can be accessed.

The screenshot shows the Postman application interface. On the left, the sidebar displays various collections and environments. The main workspace shows a collection named "BookMyConsultation / doctor-service". Within this collection, there is a "Document Upload" request. The "Body" tab is selected, showing a "form-data" key "files" with the value "Doctor's document.pdf". The response section indicates a successful 200 OK status with the message "File(s) uploaded Successfully".

- We will see File(s) uploaded successfully as a response from the API. Now we can check the Amazon S3 console to confirm whether the document has been uploaded or not.

The screenshot shows the AWS S3 console interface. On the left, there's a sidebar with various options like Buckets, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3, Storage Lens, Dashboards, AWS Organizations settings, and Feature spotlight. The main area shows a bucket named 'bmc.doctors.bucket' containing one object, '64d8ce1fcc47f56ca941f5a3/'. The object is named 'Doctor's document.pdf' and is a PDF file from August 13, 2023, at 19:36:45 (UTC+05:30), with a size of 636.3 KB and a storage class of Standard. There are buttons for Actions (Copy S3 URI, Copy URL, Download, Open, Delete, Create folder, Upload), a search bar, and a table view.

- Now let's test the endpoint that is responsible for approving the doctor's registration request. For the same, we will look to send a `PUT doctors/{doctorId}/approve` respectively. For the Request body, we will send `approvedBy`, `approverComments` respectively. If the token has the role of ADMIN, then only the endpoint can be accessed.
- But before sending the same, let us send an invalid request by deliberately not sending the token with the role of USER. We will see what's been seen in the below screenshot, an error will be thrown by the API.

The screenshot shows the Postman application interface. The left sidebar lists collections, environments, and history. The main workspace shows a collection named 'My Workspace' with several endpoints. One endpoint is selected: `PUT Approve the doctor's registration`. The request URL is `http://34.192.75.206:8081/doctors/64d8ce1fcc47f56ca941f5a3/approve`. The Body tab shows a JSON payload with `"approvedBy": "HARRY"` and `"approverComments": "Verified"`. The response status is 401 Unauthorized, with the message: `"error": "Unauthorized"` and `"path": "/doctors/64d8ce1fcc47f56ca941f5a3/approve"`.

- Still we will be testing invalid requests more by sending requests with wrong id for doctorId at `PUT doctors/{doctorId}/approve` respectively. We will see what's been seen in the below screenshot, an error will be thrown by the API.

The screenshot shows the Postman interface with the following details:

- Request Method:** PUT
- Request URL:** `http://34.192.75.206.8081/doctors/64d88d47cc47f56ca941f56/approve`
- Body (Pretty):**

```

1 | {
2 |     "approvedBy": "Harry",
3 |     "approverComments": "Verified"
4 |
5 |

```
- Status:** 404 Not Found
- Headers:** (14) - includes Content-Type: application/json, Accept: */*, User-Agent: Postman/1.29.0, Host: 34.192.75.206.8081, Connection: keep-alive, Cache-Control: no-cache, Pragma: no-cache, Postman-Token: 64d88d47cc47f56ca941f56, Content-Length: 14
- Body (Raw):**

```

1 | {
2 |     "errorCode": "ERR_RESOURCE_NOT_FOUND",
3 |     "errorMessage": "Requested resource is not available",
4 |     "errorFields": null
5 |

```

- Now it's time to send a valid request and look into whether everything works well.

The screenshot shows the Postman interface with the following details:

- Request Method:** PUT
- Request URL:** `http://34.192.75.206.8081/doctors/64d8ce1fcc47f56ca941f5a3/approve`
- Body (Pretty):**

```

1 | {
2 |     "approvedBy": "Harry",
3 |     "approverComments": "Verified"
4 |
5 |

```
- Status:** 200 OK
- Headers:** (14) - includes Content-Type: application/json, Accept: */*, User-Agent: Postman/1.29.0, Host: 34.192.75.206.8081, Connection: keep-alive, Cache-Control: no-cache, Pragma: no-cache, Postman-Token: 64d8ce1fcc47f56ca941f5a3, Content-Length: 14
- Body (Raw):**

```

1 | {
2 |     "id": "64d8ce1fcc47f56ca941f5a3",
3 |     "firstName": "Mohit",
4 |     "lastName": "Gupta",
5 |     "speciality": "GENERAL_PHYSICIAN",
6 |     "dob": "1996-01-10",
7 |     "mobile": "1234567890",
8 |     "emailId": "harry+40423mohitgupta@kibana.com",
9 |     "pan": "BOA411175K",
10 |     "status": "Active",
11 |     "approvedBy": "Harry",
12 |     "approverComments": "Verified",
13 |     "registrationDate": "2023-08-13",
14 |     "verificationDate": "2023-08-13"
15 |

```

- Now we can check email to find out whether we received email as a doctor by the notification service for being approved as a doctor.

The screenshot shows an email inbox with the following details:

Verification Email (Inbox)

From: **harry@kibantu.com via amazoneses.com**

To: harry+40423mohitgupta ▾

Subject: Hi Mohit

Body:

Your Registration is Active by Harry.

Approver Comment : Verified

Regards,

Upgrad

- Now let's test whether the doctor collection has been updated to active status or not.

The screenshot shows the MongoDB shell with the following command and output:

```
IntelliShell: BMC Mongo* > IntelliShell: BMC Mongo
52.73.212.58:27017 > db.getCollection("doctor").find({})
```

Output:

```
1> db.getCollection("doctor").find({})
2>
```

Key	Value	Type
✓ (3) _id : 64dbad14112ac2337dfdc359	{ 12 fields }	Document
✓ (4) _id : 64db898e12ac2337dfdc35a	{ 12 fields }	Document
✓ (5) _id : 64d8c1ecddadd2460d57618	{ 12 fields }	Document
✓ (6) _id : 64dbcbf5e8c516ca0320be	{ 12 fields }	Document
✓ (7) _id : 64d8ceffcc47f56ca941f5a3	{ 15 fields }	Document
✓ _id	64d8ceffcc47f56ca941f5a3	ObjectId
✓ firstName	Mohit	String
✓ lastName	Gupta	String
✓ specialty	GENERAL_PHYSICIAN	String
✓ dob	1986-01-18	String
✓ mobile	1234541175	String
✓ email	harry+40423mohitgupta@kibantu.com	String
✓ pan	BDA41175K	String
✓ status	Active	String
✓ approvedBy	Harry	String
✓ approverComments	Verified	String
✓ registrationDate	2023-08-13	String
✓ verificationDate	2023-08-13	String
✓ averageRating	0.0	Double
✓ _class	bookmyconsultation.doctorservice.entity.DoctorEntity	String

> ✓ (8) _id : 64db8cfcc5cc47f56ca941f5a4
> ✓ (9) _id : 64d8cfcc6cc47f56ca941f5a5
> ✓ (10) _id : 64d8d471cc47f56ca941f5a6

1 item selected

- Now let's test the endpoint that is responsible for rejecting the doctor's registration request. For the same, we will look to send a `PUT doctors/{doctorId}/reject` respectively. For the Request body, we will send `approvedBy`, `approverComments` respectively. If the token has the role of ADMIN, then only the endpoint can be accessed.
- For testing the same, let's create a new doctor registration so that we can reject the same as an ADMIN.

The screenshot shows the Postman interface with the following details:

- Collection:** BookMyConsultation / doctor-service
- Request Type:** POST
- URL:** `http://34.192.75.206:8081/doctors`
- Body (JSON):**

```

1   {
2     "firstName": "Ayush",
3     "lastName": "Gupta",
4     "dob": "1986-01-18",
5     "emailId": "harry+2732ayushgupta@kibanu.com",
6     "mobile": "123452732",
7     "pan": "BDAL2732K"
8   }

```
- Response:**

```

1   {
2     "_id": "64d8ad112ac2337fdcc357",
3     "firstName": "Ayush",
4     "lastName": "Gupta",
5     "specialty": "GENERAL_PHYSICIAN",
6     "dob": "1986-01-18",
7     "mobile": "123452732",
8     "emailId": "harry+2732ayushgupta@kibanu.com",
9     "pan": "BDAL2732K",
10    "status": "Pending",
11    "registrationDate": "2023-08-13"
12  }

```

The screenshot shows the MongoDB shell with the following command and results:

```

1 db.getCollection("doctor").find({})
2

```

Result:

Key	Value	Type
<code>(1) _id : 64d8ad112ac2337fdcc357</code>	{ 15 fields }	Document
<code>(2) _id : 64d8ac3c112ac2337fdcc358</code>	{ 12 fields }	Document
<code>(3) _id : 64d8ad112ac2337fdcc359</code>	{ 12 fields }	Document
<code>(4) _id : 64d8b98e812ac2337fdcc35a</code>	{ 12 fields }	Document
<code>(5) _id : 64d8cfcddadd2460d57618</code>	{ 12 fields }	Document
<code>(6) _id : 64d8cbff8e812ac2337fdcc35b</code>	{ 12 fields }	Document
<code>(7) _id : 64d8ceffc47f56ca941f5a3</code>	{ 15 fields }	Document
<code>(8) _id : 64d8cfcc5c47f56ca941f5a4</code>	{ 12 fields }	Document
<code>(9) _id : 64d8cfcc6cc47f56ca941f5a5</code>	{ 12 fields }	Document
<code>(10) _id : 64d8d471cc47f56ca941f5a6</code>	{ 12 fields }	Document

Document Data:

```

_id: 64d8d471cc47f56ca941f5a6
firstName: Ayush
lastName: Gupta
specialty: GENERAL_PHYSICIAN
dob: 1986-01-18
mobile: 123452732
emailId: harry+2732ayushgupta@kibanu.com
pan: BDAL2732K
status: Pending
registrationDate: 2023-08-13
averageRating: 0.0
_class: bookmyconsultation.doctorservice.entity.DoctorEntity

```

- Now it's time to test our reject endpoint, but before sending a valid request, let us send an invalid request by deliberately not sending the token with the role of USER. We will see what's been seen in the below screenshot, an error will be thrown by the API.

The screenshot shows the Postman interface with the following details:

- Collection:** My Workspace
- Request:** PUT Reject the doctor registration
- URL:** http://34.192.75.206:8081/doctors/64d8d47cc47f56ca941f5a6/reject
- Body (JSON):**

```

1   "approvedBy": "Harry",
2   "approverComments": "The registration is rejected"
3
4
5
6
7
    
```
- Response Status:** 401 Unauthorized
- Response Body (Pretty):**

```

1   {
2     "timestamp": "2023-08-13T16:10:56.437+00:00",
3     "status": 401,
4     "error": "Unauthorized",
5     "message": "",
6     "path": "/doctors/64d8d47cc47f56ca941f5a6/reject"
7
    
```

- Still we will be testing invalid requests more by sending requests with wrong id for doctord at PUT doctors/{doctorId}/reject respectively. We will see what's been seen in the below screenshot, an error will be thrown by the API.

The screenshot shows the Postman interface with the following details:

- Collection:** My Workspace
- Request:** PUT Reject the doctor registration
- URL:** http://34.192.75.206:8081/doctors/64d8d47cc47f56ca941f5a6/reject
- Body (JSON):**

```

1   "approvedBy": "Harry",
2   "approverComments": "The registration is rejected"
3
4
5
    
```
- Response Status:** 404 Not Found
- Response Body (Pretty):**

```

1   {
2     "errorCode": "ERR_RESOURCE_NOT_FOUND",
3     "errorMessage": "Requested resource is not available",
4     "errorFields": null
5
    
```

- NOW it's time to send a valid request and confirm whether as an admin we can reject the doctor or not.

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various collections and environments. In the center, a specific collection named 'BookMyConsultation' is selected, and under it, a sub-collection 'doctor-service' contains several API endpoints. One endpoint, 'PUT Reject the doctors registration', is highlighted. The 'Body' tab of the request configuration window is active, showing a JSON payload:

```

PUT http://34.192.75.206:8081/doctors/64d8d471cc47f56ca941f5a6/reject
{
  "approvedBy": "Harry",
  "approverComments": "The registration is rejected"
}

```

Below the body, the response status is shown as 'Status: 200 OK' with a response time of '496 ms' and a size of '800 B'. The response body is also displayed in a pretty-printed JSON format, matching the payload sent.

- Now let's test whether the doctor collection has been updated to rejected status or not.

The screenshot shows the MongoDB shell interface. A query is run against the 'doctor' collection to find all documents:

```

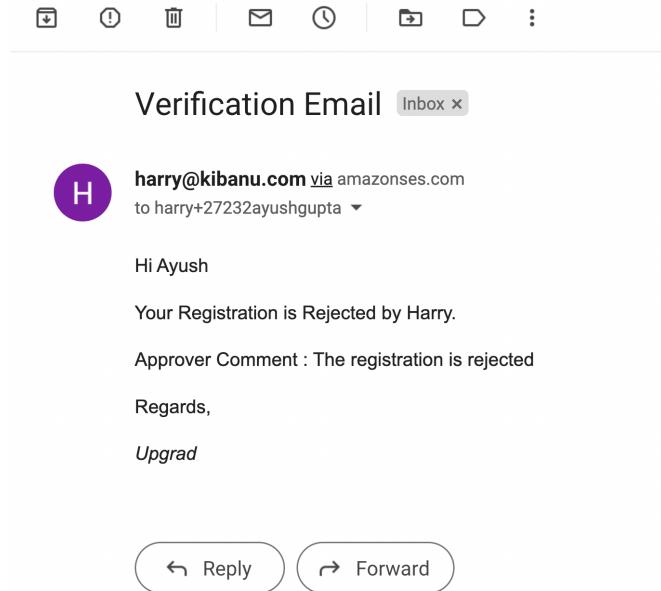
db.getCollection("doctor").find({})

```

The results are displayed in a table format. The table has three columns: 'Key', 'Value', and 'Type'. The 'Value' column shows the document structure, and the 'Type' column indicates the type of each field. The table shows 10 documents, each with fields like '_id', 'firstName', 'lastName', 'specialty', etc., with values corresponding to the JSON document sent via Postman.

Key	Value	Type
> (3) _id : 64d8ad14112ac2337dfdc359	{ 12 fields }	Document
> (4) _id : 64d8b98e112ac2337dfdc35a	{ 12 fields }	Document
> (5) _id : 64dc1ecddaddc2480d57618	{ 12 fields }	Document
> (6) _id : 64d8c8bf56e8516cadb320be	{ 12 fields }	Document
> (7) _id : 64dcce1fc47f56ca941f5a3	{ 15 fields }	Document
> (8) _id : 64d8cfcc5cc47f56ca941f5a4	{ 12 fields }	Document
> (9) _id : 64d8cfcc6cc47f56ca941f5a5	{ 12 fields }	Document
> (10) _id : 64d8d471cc47f56ca941f5a6	{ 15 fields }	Document
_id	64d8d471cc47f56ca941f5a6	ObjectID
firstName	Ayush	String
lastName	Gupta	String
specialty	GENERAL_PHYSICIAN	String
dob	1986-01-18	String
mobile	1234527232	String
emailId	harry+2723ayushgupta@kibanu.com	String
pan	BDAL27232K	String
status	Rejected	String
approvedBy	Harry	String
approverComments	The registration is rejected	String
registrationDate	2023-08-13	String
verificationDate	2023-08-13	String
averageRating	0.0	Double
__class	bookmyconsultation.doctorService.entity.DoctorEntity	String

- Now we can check email to find out whether we received email as a doctor by the notification service for being rejected as a doctor.



- Now let's test the `GET /doctors` respectively by various statuses to confirm whether the correct list is being returned. Role: USER, ADMIN
- First let's test with pending status by testing `GET /doctors?status=Pending` respectively.

The screenshot shows the Postman interface with a collection named 'My Workspace'. A specific request is selected: 'Fetch the list of doctors based on status and speciality'. The request method is 'GET' and the URL is `http://34.192.75.206:8081/doctors?status=Pending&speciality=GENERAL_PHYSICIAN`. The 'Params' tab shows two parameters: 'status' set to 'Pending' and 'speciality' set to 'GENERAL_PHYSICIAN'. The 'Body' tab shows the response body in JSON format:

```

1 > {
2   "id": "64d0b98e112ac2337dfdc35a",
3   "firstName": "Harry9856130254",
4   "lastName": "Doctor",
5   "speciality": "GENERAL_PHYSICIAN",
6   "dob": "1993-08-04",
7   "mobile": "1234554321",
8   "emailId": "Harry+9056130254@kibanu.com",
9   "pan": "BDALF4786K",
10  "status": "Pending",
11  "approvedBy": null,
12  "approverComments": null,
13  "registrationDate": "2023-08-13",
14  "verificationDate": null,
15}

```

The status bar at the bottom indicates 'Status: 200 OK'.

- Now let's test with rejected status by testing `GET /doctors?status=Rejected` respectively.

The screenshot shows the Postman interface with a collection named "BookMyConsultation / doctor-service". A specific request is selected: `GET Fetch : http://34.192.75.206:8081/doctors?status=Rejected&specialty=GENERAL_PHYSICIAN`. The "Params" tab is active, showing query parameters: `status: Rejected` and `specialty: GENERAL_PHYSICIAN`. The "Body" tab shows the response in JSON format:

```

1 > {
2   ...
3   "id": "64d8d471cc47f56ca941f5a6",
4   "firstName": "Avush",
5   "lastName": "Gupta",
6   "specialty": "GENERAL_PHYSICIAN",
7   "dob": "1986-01-18",
8   "mobile": "1234567232",
9   "emailId": "harry2732ayushgupta@kibanu.com",
10  "pan": "B0A2L7232K",
11  "status": "Rejected",
12  "approvedBy": "Harry",
13  "approverComments": "The registration is rejected",
14  "registrationDate": "2023-08-13",
15  "verificationDate": "2023-08-13",
16  "averageRating": 0.0
17 }

```

- Now let's test with active status by testing `GET /doctors?status=Active` respectively.

The screenshot shows the Postman interface with the same collection and request setup as the previous screenshot. The "Params" tab is active, showing query parameters: `status: Active` and `specialty: GENERAL_PHYSICIAN`. The "Body" tab shows the response in JSON format:

```

1 > {
2   ...
3   "id": "64d8ce1fc47f56ca941f5a3",
4   "firstName": "Mohit",
5   "lastName": "Gupta",
6   "specialty": "GENERAL_PHYSICIAN",
7   "dob": "1986-01-18",
8   "mobile": "1234543175",
9   "emailId": "harry40423mohitgupta@kibanu.com",
10  "pan": "B0A11175K",
11  "status": "Active",
12  "approvedBy": "Harry",
13  "approverComments": "Verified",
14  "registrationDate": "2023-08-13",
15  "verificationDate": "2023-08-13",
16  "averageRating": 6.0
17 }

```

- Now let's test `GET /doctors/{doctorId}/documents/metadata` that is responsible for returning the list of the documents uploaded by the doctor. Role: ADMIN,USER

The screenshot shows the Postman interface with a collection named "My Workspace". A specific request is selected: `GET /doctors/64d8ce1fc47f56ca941f5a3/documents/metadata`. The response body contains the following JSON:

```

[{"id": 1, "name": "Doctor's document.pdf"}]

```

- Now let's test `GET /doctors/{doctorId}/documents/{documentName}` that is responsible for downloading the documents uploaded by the doctor based on the name of the document. Role: ADMIN,USER

The screenshot shows the Postman interface with a collection named "My Workspace". A specific request is selected: `GET /doctors/64d8ce1fc47f56ca941f5a3/documents/Doctor's document.pdf`. The response body shows the PDF content of the document.

- Now let's test POST /users respectively. Request body: firstName, lastName, dob, emailId, mobile. If the token has the role of USER, then the endpoint can be accessed.

The screenshot shows the Postman application interface. On the left, the sidebar displays various collections and environments. In the center, a specific collection named "BMC API / user-service" is selected, and a request titled "POST Register the User" is being viewed. The request method is POST, and the URL is `http://34.192.75.206:8083/users`. The "Body" tab is active, showing a JSON payload:

```

1
2   {
3     "firstName": "Sanjeev",
4     "lastName": "Sharma",
5     "dob": "1986-01-19",
6     "emailId": "harry+36045sanjeevsharma@kibanu.com",
7     "mobile": "9912336045"
8   }

```

The response status is 201 Created, with a time of 807 ms and a size of 631 B. Below the response, the "Pretty" JSON output is shown:

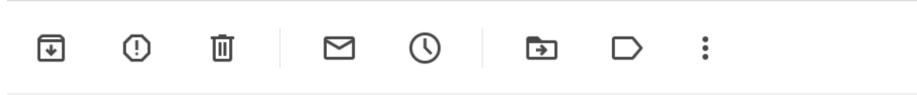
```

1
2   {
3     "id": "64db565c09a8c66d1245f76a",
4     "firstName": "Sanjeev",
5     "lastName": "Sharma",
6     "dob": "1986-01-19",
7     "mobile": "9912336045",
8     "emailId": "harry+36045sanjeevsharma@kibanu.com",
9     "createdDate": "2023-08-15"
}

```

The screenshot shows the MongoDB shell interface. The connection is to "IntelliShell: BMC Mongo" at port 27017. The command entered is `db.getCollection("user").find({})`. The results show three documents, each with fields: _id, firstName, lastName, dob, mobile, emailId, and createdDate. The data is displayed in a table format with columns for Key, Value, and Type.

Key	Value	Type
_id	64db565c09a8c66d1245f76a	Document
firstName	Sanjeev	String
lastName	Sharma	String
dob	1986-01-19	String
mobile	9912336045	String
emailId	harry+36045sanjeevsharma@kibanu.com	String
createdDate	2023-08-15	String
_class	bookmyconsultation.userservice.entity.UserEntity	String



Welcome Email Inbox



harry@kibantu.com via [amazonses.com](#)
to [harry+36045sanjeevsharma](#) ▾

Hi Sanjeev

Congratulations !!! Your Registration is confirmed

Regards,

Upgrad

⬅ Reply ➡ Forward

- In same `POST /users` endpoint, let's also test by providing an invalid input, we should expect an error in such case, and it's working (see screenshot)

The screenshot shows the Postman interface with the following details:

- Collection:** My Workspace
- Request:** POST Register the User
- URL:** `http://34.192.75.206:8083/users`
- Method:** POST
- Body:** JSON (selected)
Content:

```
1 {"lastName": "Sharma",
2 "dob": "1986-01-19",
3 "emailId": "harry+36045sanjeevsharma@kibantu.com",
4 "mobile": "9912336045"}  
5  
6  
7
```
- Status:** 400 Bad Request
- Response Body:**

```
1 {
2     "errorCode": "ERR_INVALID_INPUT",
3     "errorMessage": "Invalid input. Parameter name: ",
4     "errorFields": [
5         "First Name"
6     ]
7 }
```

- Now let's test `GET /users/{userID}` respectively. Role: USER, ADMIN

The screenshot shows the Postman interface with a successful API call. The URL is `http://34.192.75.206:8083/users/64db565c09a8c66d1245f76a`. The response status is 201 Created. The JSON body contains the following user data:

```

1  "id": "64db565c09a8c66d1245f76a",
2  "firstName": "Sanjeev",
3  "lastName": "Sharma",
4  "dob": "1986-01-19",
5  "mobile": "9912336045",
6  "emailId": "harry+36045sanjeevsharma@kibanu.com",
7  "createdDate": "2023-08-15"
8
9

```

- In same `GET /users/{userID}` endpoint, let's also test by providing an invalid input, we should expect an error in such case, and it's working (see screenshot)

The screenshot shows the Postman interface with an invalid input test. The URL is `http://34.192.75.206:8083/users`. The response status is 400 Bad Request. The JSON body contains the following error message:

```

1  {
2    "errorCode": "ERR_INVALID_INPUT",
3    "errorMessage": "Invalid input. Parameter name: ",
4    "errorFields": [
5      "First Name"
6    ]
7

```

- Now let's test the File upload endpoint: `POST /users/{id}/documents` respectively.

The screenshot shows the Postman interface with the following details:

- Collection:** BMC API
- Request:** POST Upload the documents
- URL:** http://34.192.75.206:8083/users/64db565c09a8c66d1245f77a/documents
- Body Type:** form-data
- Body Fields:**

Key	Value	Description
files	Patient document.pdf	
Key	Value	Description
- Response:**
 - Status: 200 OK
 - Time: 4.82 s
 - Size: 464 B
 - Body content: 1 File(s) uploaded Successfully

- We can now confirm the file upload by checking the Amazon S3 bucket.

The screenshot shows the AWS S3 console with the following details:

- Buckets:** bmc.users
- Objects:** Patient document.pdf
- Details:**

Name	Type	Last modified	Size	Storage class
Patient document.pdf	pdf	August 15, 2023, 16:23:04 (UTC+05:30)	636.3 KB	Standard

- Now let's test the File download endpoint: GET `/users/{id}/documents/{documentName}` respectively.

User books an appointment

Download the documents - My Workspace

POST Register the User POST JWT Token for ... GET Fetch the details ... POST Upload the doc ... GET Download the doc ... + *** No Environment

My Workspace

BMC API

user-service

GET Fetch the details of the user...
POST Register the User
POST Upload the documents
GET Download the documents
GET Metadata of the files uploaded...
POST User books an appointment ...
GET Fetch the details of an app...
GET Fetch all the appointments...
POST Prescription
GET Fetch the doctor's availability...
doctor-service

rating-service

appointment-service

GET Rate the doctor
POST Doctor updates his availability...
POST User books an appointment ...
GET Fetch the details of an app...
GET Fetch all the appointments...
POST Prescription
GET Fetch the doctor's availability...
doctor-service

GET Fetch the doctors details ...
GET Fetch the list of doctors ...
PUT Approve the doctors registration...
PUT Reject the doctors registration...
POST Doctor Registration
GET Document Download
POST Document Upload
GET Metadata of the files uploaded...

Body Cookies (1) Headers (15) Test Results

Pretty Raw Preview Visualize Text

Status: 200 OK Time: 4.82 s Size: 636.8 KB

- Now let's test the endpoint that returns the names of the file uploaded by a user: GET `/users/{id}/documents/metadata` respectively.

User books an appointment

Metadat of the files uploaded - My Workspace

POST Register the User POST JWT Token for ... GET Fetch the details ... POST Upload the doc ... GET Download the doc ... GET Metadata of ... + *** No Environment

My Workspace

BMC API

user-service

GET Fetch the details of the user...
POST Register the User
POST Upload the documents
GET Download the documents
GET Metadata of the files uploaded...
rating-service

appointment-service

GET Rate the doctor
POST Doctor updates his availability...
POST User books an appointment ...
GET Fetch the details of an app...
GET Fetch all the appointments...
POST Prescription
GET Fetch the doctor's availability...
doctor-service

GET Fetch the doctors details ...
GET Fetch the list of doctors ...
PUT Approve the doctors registration...
PUT Reject the doctors registration...
POST Doctor Registration
GET Document Download
POST Document Upload
GET Metadata of the files uploaded...

Body Cookies (1) Headers (14) Test Results

Pretty Raw Preview Visualize JSON

Status: 200 OK Time: 1085 ms Size: 459 B

- Now let's test the endpoint responsible for updating the availability of the doctors. `POST /doctor/{doctorId}/availability` respectively. Request body: Availability Map. Role: USER, ADMIN respectively.

The screenshot shows the Postman interface with the following details:

- Collection:** BMC API
- Request Type:** POST
- URL:** `http://34.192.75.206:8082/doctor/64d8ce1fcc47f56ca941f5a3/availability`
- Body (JSON):**

```

1 {
2   "availabilityMap": [
3     {
4       "date": "2021-07-18",
5       "times": ["10AM-11AM"]
6     },
7     {
8       "date": "2021-07-19",
9       "times": ["11AM-12PM"]
10    }
11  ]
12 }
```
- Status:** 200 OK
- Time:** 930 ms
- Size:** 469 B

- Now let's test the endpoint responsible for returning the availability of the doctors. `GET /doctor/{doctorId}/availability` respectively. Role: USER, ADMIN

The screenshot shows the Postman interface with the following details:

- Collection:** BMC API
- Request Type:** GET
- URL:** `http://34.192.75.206:8082/doctor/64d8ce1fcc47f56ca941f5a3/availability`
- Headers (9):**

Key	Value	Description
Key	Value	Description
- Body (Pretty):**

```

1 {
2   "doctorId": "64d8ce1fcc47f56ca941f5a3",
3   "availabilityMap": [
4     {
5       "date": "2021-07-18",
6       "times": ["10AM-11AM"]
7     },
8     {
9       "date": "2021-07-19",
10      "times": ["11AM-12PM"]
11    }
12  ]
13 }
```
- Status:** 200 OK
- Time:** 499 ms
- Size:** 548 B

- Now let's test the endpoint responsible for booking an appointment. POST /appointments respectively. Request body: doctorId, userId, appointmentDate, timeSlot. Role: USER, ADMIN

POST Register the User POST JWT Token for user POST User books an appointment GET Fetch the doctor's avail... + No Environment

POST <http://34.192.75.208:8082/appointments>

Params Authorization Headers (11) **Body** Pre-request Script Tests Settings Cookies Beautify

```

1
2   "doctorId": "6d80c01fc5d756ca941f5a3",
3   "doctorName": "Mohit Gupta"
4   "userId": "64d956c098c84dd1245f76a",
5   "timeSlot": "10AM-11AM",
6   "appointmentDate": "2021-07-18"
7

```

Status: 200 OK Time: 666 ms Size: 459 B Save as Example

Appointment Email Inbox x

H harry@kibanu.com via amazoneses.com
to harry+36045sanjeevsharma

Hi Sanjeev Sharma

Your Appointment is registered.

Details:

Doctor Name : Mohit Gupta
Appointment Date : 2021-07-18
Time : 10AM-11AM
Status : PendingPayment

Regards,

Upgrad

Reply Forward

- Now let's test the endpoint responsible for retrieving the details of an appointment. `GET /appointments/{appointmentId}` respectively. Role: USER, ADMIN.

The screenshot shows the Postman interface with a collection named "BMC API". A specific request is selected: `GET Fetch the details of an appointment based on appointmentId`. The URL is set to `http://34.192.75.206:8082/appointments/2c94808689f8c4c20189f914e11c0000`. The "Body" tab shows a JSON response with the following data:

```

1
2   "appointmentId": "2c94808689f8c4c20189f914e11c0000",
3   "doctorId": "64d8ce1fc47f5f5ca941f5a3",
4   "doctorName": "Mohit Gupta",
5   "userId": "64db565c09a8c66d1245f76a",
6   "timeSlot": "10AM-11AM",
7   "status": "PendingPayment",
8   "appointmentDate": "2021-07-18"
9

```

- Now let's test the endpoint responsible for retrieving the details of all the appointments corresponding to a userId. `GET /users/{userId}/appointments` respectively. Role: USER, ADMIN.

The screenshot shows the Postman interface with the same "BMC API" collection. A new request is selected: `GET Fetch all the appointments of a user based on userId`. The URL is set to `http://34.192.75.206:8082/users/64db565c09a8c66d1245f76a/appointments`. The "Body" tab shows a JSON response with the same data as the previous screenshot, indicating a single appointment record.

- Now let's test the endpoint responsible for sending the prescriptions for the appointment. As defined, the doctors will send the prescription to the users after the appointment. The prescription can be sent only if the status of the appointment is Confirmed. As we know that when the appointment is made, the status of the appointment is set to 'PendingPayment'. Once the payment is made, the status will change to Confirmed. The API endpoint is POST /prescriptions respectively. Request body: appointmentId, doctorId, userId, diagnosis, medicineList. Role: USER

The screenshot shows a POST request to `http://34.192.75.206:8082/prescriptions`. The request body is JSON:

```

1  {
2     "appointmentId": "2e94808689fc4c20189f914e1c9000",
3     "doctorId": "64dc8cc1fc47f56c94d15f3",
4     "doctorName": "Rohit Gupta",
5     "userId": "59359bc0c0d1245f76a",
6     "diagnosis": "High Chol",
7     "medicineList": [
8         {
9             "name": "Gliben",
10            "type": "tablet",
11            "dosage": "1 week",
12            "duration": "1 week",
13            "frequency": "3 times a day",
14            "remarks": "after food"
15        }
16    ]
17 }

```

The response status is 404 Not Found with the following JSON body:

```

1  {
2     "errorCode": "ERR_PAYMENT_PENDING",
3     "errorMessage": "Prescription cannot be issued since the payment status is pending"
4 }

```

- As we can see there is an error thrown to the POST /prescriptions API. This is good because it's expected.
- It's time to test the Payment Endpoint, POST /payments?appointmentId={appointmentId} respectively. Role: USER, no request body need to be sent.

The screenshot shows a POST request to `http://34.192.75.206:8086/payments?appointmentId=2c94808689fc4c20189f914e1c0000`. The query parameter is `appointmentId` with value `2c94808689fc4c20189f914e1c0000`.

The response status is 200 OK with the following JSON body:

```

1  {
2     "id": "144d66e151097f50d4ec45",
3     "appointmentId": "2c94808689fc4c20189f914e1c9000",
4     "creationDate": "2023-08-19T12:22:49.594659Z"
5 }

```

- Now as payment is done, let's test the POST /prescriptions again with same details as last time. This time we get 200 OK respectively and also get the email as expected.

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various collections and environments. The main workspace shows a POST request to 'BMC API / appointment-service / Prescription' with the URL `http://34.192.75.206:8082/prescriptions`. The 'Body' tab displays the JSON payload for the prescription:

```

1 {
2   "appointmentId": "2c94808689f8c4c20189f914e11c0000",
3   "doctorId": "64d8ce1fcc47f56ca941f5a3",
4   "doctorName": "Mohit Gupta",
5   "userId": "64d565c09a8c66d1245f76a",
6   "diagnosis": "Teeth Cavity",
7   "medicineList": []
8   [
9     {
10       "name": "Calpol",
11       "type": "Tablet",
12       "dosage": "1 week",
13       "duration": "1 week",
14       "frequency": "3 times a day",
15       "remarks": "after food"
16     }
]

```

The response status is 200 OK with a time of 833 ms and a size of 394 B. Below the body, there are tabs for Body, Cookies, Headers, Test Results, and a status bar indicating Status: 200 OK, Time: 833 ms, Size: 394 B.

The screenshot shows an email inbox with one message from 'harry@kibanu.com via amazoneses.com'. The subject is 'Your Prescription is Ready'. The message content is as follows:

H [harry@kibanu.com](#) via amazoneses.com
to harry+36045sanjeevsharma

17:53 (2 minutes ago) [Star](#) [Reply](#) [Forward](#)

Hi

Your Prescription is ready.

Details:

Doctor Id : 64d8ce1fcc47f56ca941f5a3
 Doctor Name : Mohit Gupta
 Appointment Id : 2c94808689f8c4c20189f914e11c0000
 Diagnosis : Teeth Cavity
 Medicines : name=Calpol | type=Tablet | dosage=1 week | duration=1 week | frequency=3 times a day | remarks=after food | name=PainKill | type=Syrup | dosage=1 week | duration=1 week | frequency=3 times a day | remarks=after food

Regards,

Upgrad

[Reply](#) [Forward](#)

- Now let's test the endpoint that is used by the users to submit the ratings of their experience with the doctor with whom they had an appointment. `POST /ratings` respectively. Request body: `doctorId`, `rating`. Role: `USER`. Here also we get `200 OK` respectively.

The screenshot shows the Postman interface with the following details:

- Collection:** My Workspace
- Request:**
 - Method: POST
 - URL: http://34.192.75.206:8084/ratings
 - Body tab selected
 - JSON content type
 - Body content:

```

1: {
2:   "doctorId": "64d8ce1fcc47f56ca941f5a3",
3:   "rating": "4",
4:   "comments": "Doctor is very knowledgeable"
5: }

```
- Response:**
 - Status: 200 OK
 - Time: 2.14 s
 - Size: 469 B

Code Explanation by Service

In the development of the BookMyConsultation Project, our team extensively crafted a robust architecture by employing a total of 9 microservices, each contributing uniquely to make the whole project work smoothly. These sets of microservices create a seamless and efficient healthcare consultation platform. Let's take a closer look at the building blocks of the BookMyConsultation Project.

- Service Registry: At the heart of the microservices-based application, lies the challenge of managing service instances dynamically as they come online or go offline. This dynamicity is efficiently managed by the Service Registry, akin to a sophisticated database that maintains a record of available instances of every microservice within the application. In our architectural design, the Eureka framework is adeptly employed to handle this critical task. With its agility in tracking service instances, Eureka ensures that the system remains responsive and fault-tolerant, effortlessly accommodating new services while elegantly handling service unavailability.

2. **BMC Gateway (API Gateway):** In the pursuit of a user-friendly interface and a streamlined API management process, the BMC Gateway is an indispensable component in our application. This intelligent API gateway takes charge of intercepting all incoming requests and skillfully routing them to their designated services. With its role as a mediator between the UI and underlying microservices, the BMC Gateway not only enhances security but also optimizes the user experience by offering a consolidated and organized interaction point.
3. **Notification Service:** The Notification Service serves as a vigilant listener, tuned in to the Kafka topics that serve as conduits for messages exchanged between services. This vigilant service ensures timely notifications, orchestrating the delivery of critical emails triggered by pivotal events such as Doctor's approval, Doctor's rejection, Appointment confirmation, and Prescription. By fulfilling this pivotal role, the Notification Service guarantees users and doctors stay informed and engaged throughout their healthcare journey.
4. **Rating Service:** The user experience should be the heart of any platform, and the Rating Service is used to enhance that same experience. This service empowers the users to rate their interactions and experiences with healthcare providers.
5. **User Service:** User onboarding is the foundational step in establishing a seamless connection between individuals and the healthcare ecosystem. The User Service microservice handles this critical aspect, offering a set of endpoints that facilitate user registration. Beyond the basics, users can also upload medical documents, ensuring that their past medical history is securely integrated into the platform.
6. **Doctor Service:** Empowering doctors to seamlessly integrate into the BookMyConsultation ecosystem makes a strong case for a sophisticated microservice that handles many facets of Doctor's onboarding journey. This Doctor Service steps into this role, providing designed endpoints for not only doctor registration but also embarking on the verification process overseen by BMC employees, ensuring the accuracy and authenticity of the doctor's details. Upon approval, the doctor's status transitions from Pending to Active, or in the case of rejection, to Rejected, managing the influx of healthcare providers into the platform.
7. **Appointment Service:** Scheduling appointments seamlessly, a cornerstone of the BookMyConsultation platform, is facilitated by the Appointment Service microservice. With a suite of defined endpoints, users can effortlessly check doctor availability and schedule appointments at their convenience. Similar to the User Service, the Appointment Service also accommodates document uploads, facilitating the exchange of crucial medical information and ensuring the continuity of care..

8. Payment Service: The endpoints defined within the payment service are used to make the payment, and while that happens, the status of the appointment will change from PendingPayment to Confirmed.
9. Security Provider: The Security Provider microservice fortifies the BookMyConsultation ecosystem with a robust security layer. At the heart of this service lies the generation and management of JSON Web Tokens (JWTs). While token validation happens in each microservice, the Security Provider stands as the authoritative source of truth. This centralization ensures consistency, accuracy, and a unified approach to security, offering users and doctors the peace of mind while doing healthcare interactions.

That's it, if you got here that means the project is running well properly and functionally.