Scalable Services – SEZG583

Group Assignment
Microservices-based Application Development and Exploring Deployment Tools

Healthcare Appointment System

Team Member	Contributed Segments	Contribution
Aikansh Boyal (2024TM93021)	Appointment Service, User Service, Inter-Communication Of Microservices, Deployment Of Microservices (Docker And Minikube), And Demonstration	20%
Ayush Sharma (2024TM93653)	Doctor Service	20%
Supriya P (2023TM93755)	Notification Service, Mail Trap Configurations, Architecture, Database Design, And Demonstration	20%
Amruthaa V (2024TM93152)	Segregation Of Information For The Documentation	20%
Naman Adlakha (2024TM93092)	Optimized and deployed microservice to enhance seamless communication and improve system efficiency across distributed components.	20%

Table of Contents

Quick Links

1. Video Demonstration Drive link: https://drive.google.com/drive/folders/1jjR9H_6dwLouoUk4Dx4oP8gNZYknnMlV?usp=sharing
2. User Service Repository: https://github.com/aikansh-bits/User-Services
3. Doctor Service Repository: https://github.com/AyushSharma-IN/DoctorAPIs.git
4. Appointment Service Repository: https://github.com/aikansh-bits/Appointment-Services
5. Notification Service: https://github.com/SupriyaPejavara/NotificationService.git

Introduction

The Healthcare Appointment System is a modern, web-based platform designed to provide an efficient experience for patients to book, manage, and receive reminders for appointments with doctors. Built using a microservices architecture, the system ensures flexibility, scalability, and maintainability by breaking down the application into independent, modular services.

This modular approach makes it easy to scale individual components, enhance specific features, and deploy updates without disrupting the entire system. The architecture leverages technologies like .NET 8, Node.js, and Docker, with each service deployed in a separate container to ensure environmental consistency and resource isolation.

Microservices Components

The system is composed of four independently deployed microservices:

- **User Service** – Handles user registration, authentication, and profile management.
- **Doctor Service** – Manages doctor profiles, specializations, and availability.
- **Appointment Service** – Facilitates appointment booking, rescheduling, and cancellations.
- **Notification Service** – Sends notifications via email/SMS about bookings, reminders, and cancellations.

Key Architectural Features

- **Microservices Architecture:**

  Each service is built and deployed independently, promoting scalability, maintainability, and independent updates without affecting the entire system.

- **Dockerized Deployment:**

  Every microservice runs in its own Docker container, ensuring environment consistency and enabling easy deployment and orchestration (optionally extendable to Kubernetes for scalability).

- **API Gateway:**

  A REST-based API Gateway serves as the single entry point for external clients. It simplifies inter-service communication and centralizes cross-cutting concerns such as:

  - Authentication and authorization
  - Rate limiting and request routing
  - Load balancing and logging

- **Database per Service Pattern:**

  Each microservice manages its own dedicated database, ensuring:

  - Data encapsulation – Services do not share databases, avoiding tight coupling.
  - Autonomy – Each service can choose the database type and schema best suited for its functionality.

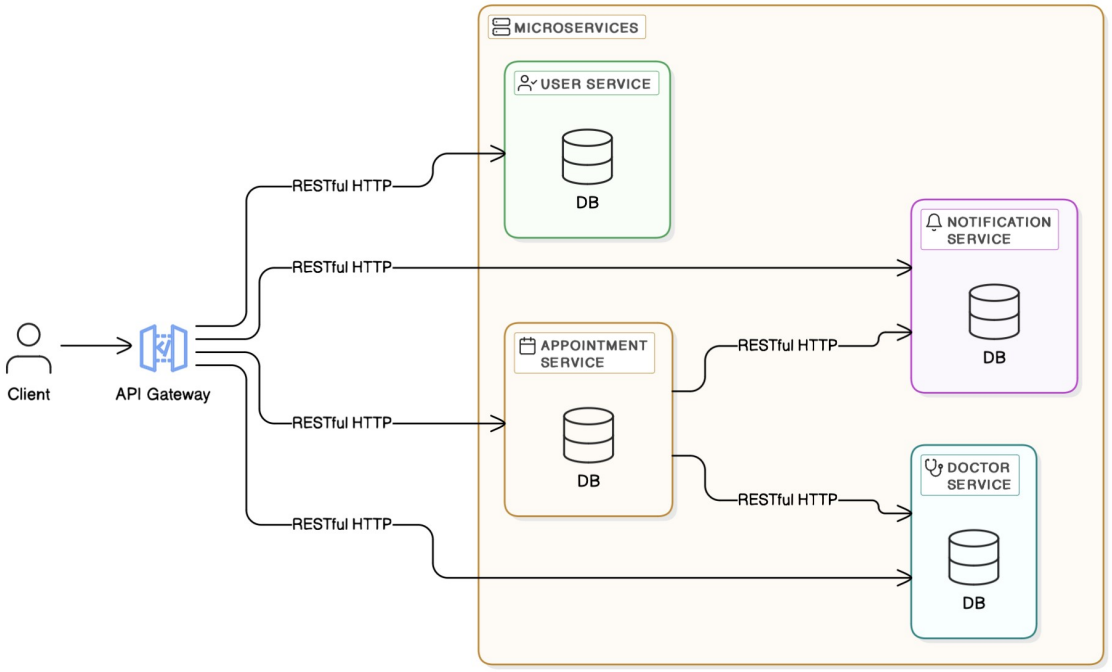
Prerequisites

Before starting the development of the healthcare appointment system, it is required to set up the necessary tools and dependencies. The application was built on Windows machines.

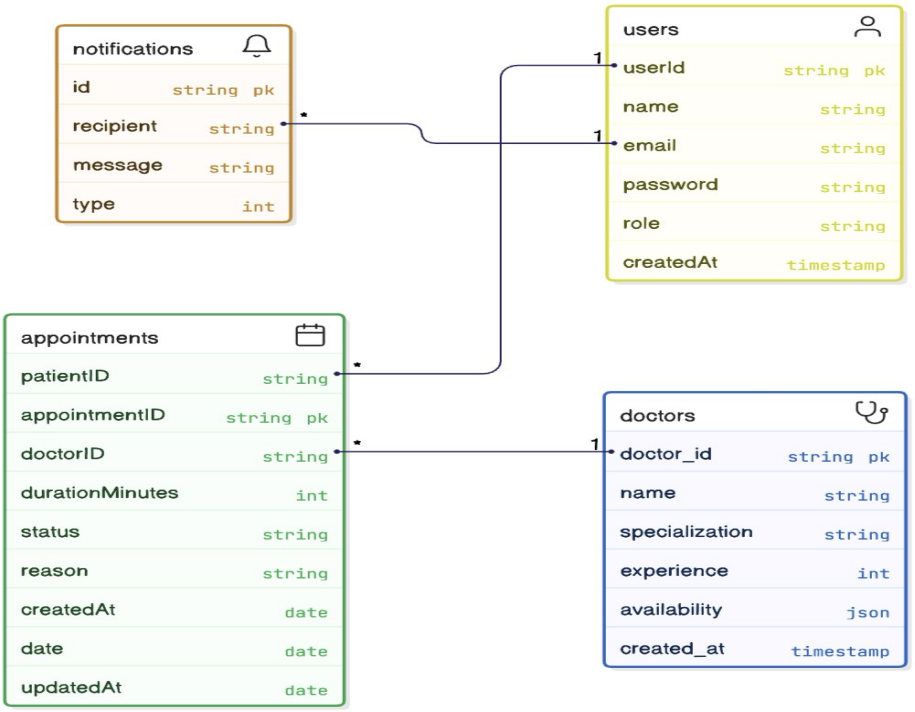
Below are the key tools and installations required for the development environment:

Technology	Purpose	Usage
.NET 8 SDK	Core framework for building backend APIs.	Developing the Notification Service API, implementing clean architecture, dependency injection, and integrating Entity Framework Core.
Node.js	JavaScript runtime for scalable applications.	Running frontend/backend scripts, enabling real-time features, and executing development tools.
Entity Framework Core 8	ORM for database interaction in .NET applications.	Managing data access logic, running migrations, querying, and storing data (e.g., in SQLite).
SQLite	Lightweight relational database engine.	Storing notification logs locally in .db files during development or lightweight deployments.
Azure Database	Scalable, cloud-hosted relational database platform.	Hosting production-ready databases with high availability and disaster recovery capabilities.
MongoDB (Atlas)	NoSQL document-oriented database.	Storing schema-less data such as logs, dynamic notification content, and metadata.
Mailtrap	Email testing tool for safe and controlled testing.	Simulating and capturing emails in test environmentswithout sending real messages to users.
Swagger UI	API documentation and testing interface.	Generating interactive and user-friendly documentation for APIs to streamline development and testing.
Postman	API development, testing, and debugging tool.	Testing, modifying, and documenting RESTful APIs across all services.

System Architecture



Database Design



Microservices overview

1. User Service

Overview

The user service offers a range of endpoints to facilitate user creation, login, and updating.

Database Schema

Column	Type	Description
userId	String	Primary Key (UUID)
Name	String	User full name
Email	String	Unique email address
password	String	Hashed password
Role	String	Role of the user: 'patient' or 'doctor'
createdAt	TIMESTAMP	Timestamp of the user

API Endpoints

This group of endpoints provides functionalities to manage user accounts, including creation, authentication, retrieval, and updating.

#	Endpoint	Method	Description
1	/users/signup	POST	Registers a new user (patient/doctor).
2	/users/login	POST	Authenticates the user and returns a token.
3	/getUser/{userId}	GET	Retrieves user profile details.
4	/updateUser/{userId}	PUT	Updates the user profile.

Code Repository

The complete source code for the User Service is available on GitHub. You can access and explore the code repository at the following link: https://github.com/aikansh-bits/User-Services

DockerFile

```
# Use the official Node.js image
FROM node:18-alpine

# Set the working directory
WORKDIR /app
```

```
# Copy package files and install dependencies
COPY package*.json ./
RUN npm install --production

# Copy application source code
COPY . .

# Copy environment variables
COPY .env .env

# Expose the port defined in .env (3000)
EXPOSE 3000

# Start the app
CMD ["npm", "start"]
```

2. Doctor Service

Overview

The Doctor Service offers a range of endpoints to facilitate doctor account management, including doctor creation, and the management of doctors.

Database Schema

Column	Type	Description
doctor_id	VARCHAR	Primary Key (UUID)
name	TEXT	Doctor's name
specialization	TEXT	Medical field
experience	INT	Years of experience
availability	JSON	Available days list
created_at	TIMESTAMP	The doctor added a timestamp

API Endpoints

This group of endpoints provides functionalities to manage doctor accounts, including creation, retrieval, updating, and deletion.

#	Endpoint	Method	Description
1	/api/Doctors	GET	Retrieves a list of all doctors.
2	/api/Doctors	POST	Updates the list of all doctors.
3	/api/Doctors/{doctorId}	GET	Retrieves detailed information about a doctor.
4	/api/Doctors/{doctorId}	PUT	Updates the detailed information of a doctor.
5	/api/Doctors/{doctorId}	DELETE	Deletes the information of a doctor.
6	/api/Doctors/{doctorId}/availability	PUT	Updates the availability of a doctor.

Code Repository

The complete source code for the Doctor Service is available on GitHub. You can access and explore the code repository at the following link:

https://github.com/AyushSharma-IN/DoctorAPIs.git

DockerFile

```
# Use the full .NET 8 ASP.NET runtime image (Debian-based, NOT Alpine)
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
WORKDIR /app
EXPOSE 8080
EXPOSE 8081

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src

COPY ["DoctorAPIs.csproj", "./"]
RUN dotnet restore "DoctorAPIs.csproj"

COPY . .
RUN dotnet build "DoctorAPIs.csproj" -c $BUILD_CONFIGURATION -o /app/build

FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "DoctorAPIs.csproj" -c $BUILD_CONFIGURATION -o /app/publish
/p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "DoctorAPIs.dll"]
```

3. Appointment Service

Overview

The Appointment Service offers a range of endpoints to facilitate appointment management. including appointment creation, notification, etc.

Database Schema

Column	Type	Description
appointmentID	int	Primary Key (UUID)
patientID	String	Foreign Key referencing the Users table
doctorID	String	Foreign Key referencing the Doctors table
Date	Date	Appointment date
durationMinutes	int	Appointment duration in minutes
Status	String	The current status of the appointment can be "scheduled", "completed", "cancelled", "no_show", "rescheduled", or by default to "scheduled"
Reason	String	The reason for the visit to the doctor
createdAt	Date	The appointment creation time
updatedAt	Date	The last time the appointment was updated

API Endpoints

This group of endpoints provides functionalities to manage doctor accounts, including creation,, retrieval, updating, and deletion.

Endpoint	Method	Description
/appointments/getAllAppointments	GET	Get all appointments.
/appointments/getAppointment/{appointmentID}	GET	Retrieves the appointment details.
/appointments/bookAppointment	POST	Book an appointment

/appointments/cancelAppointment/{appointmentID}	DELETE	Cancels an appointment.
/appointments/updateAppointment/{appointmentID}	PUT	Updates an appointment

Code Repository

The complete source code for the Appointment Service is available on GitHub. You can access and explore the code repository at the following link:
https://github.com/aikansh-bits/Appointment-Services

DockerFile

```
# Step 1: Use the official Node.js image
FROM node:18-alpine

# Step 2: Set the working directory inside the container
WORKDIR /app

# Step 3: Copy the package.json and package-lock.json (or npm-shrinkwrap.json) into the container
COPY package*.json ./

# Step 4: Install dependencies inside the container
RUN npm install --production

# Step 5: Copy the rest of your application code into the container
COPY . .

# Step 6: Expose port (5000 is the default for your app)
EXPOSE 5000

# Step 7: Define the command to run your app
CMD ["npm", "start"]
```

4. Notification Service

Overview

The Notification Service offers a range of endpoints to create notifications to specific users.

Database Schema

Column	Type	Description
recipient	VARCHAR	Recipient mail ID
message	VARCHAR	Message to be sent. Notification Text.
type	INT	0 – Email; 1 – SMS; 2 – Push notification

API Endpoint

This endpoint provides functionality to send notifications via email or SMS.

#	Endpoint	Method	Description
1	/notifications/send	POST	Send notifications based on type (email, sms, or push notifications)

Code Repository

The complete source code for the Notification Service is available on GitHub. You can access and explore the code repository at the following link:

https://github.com/SupriyaPejavara/NotificationService.git

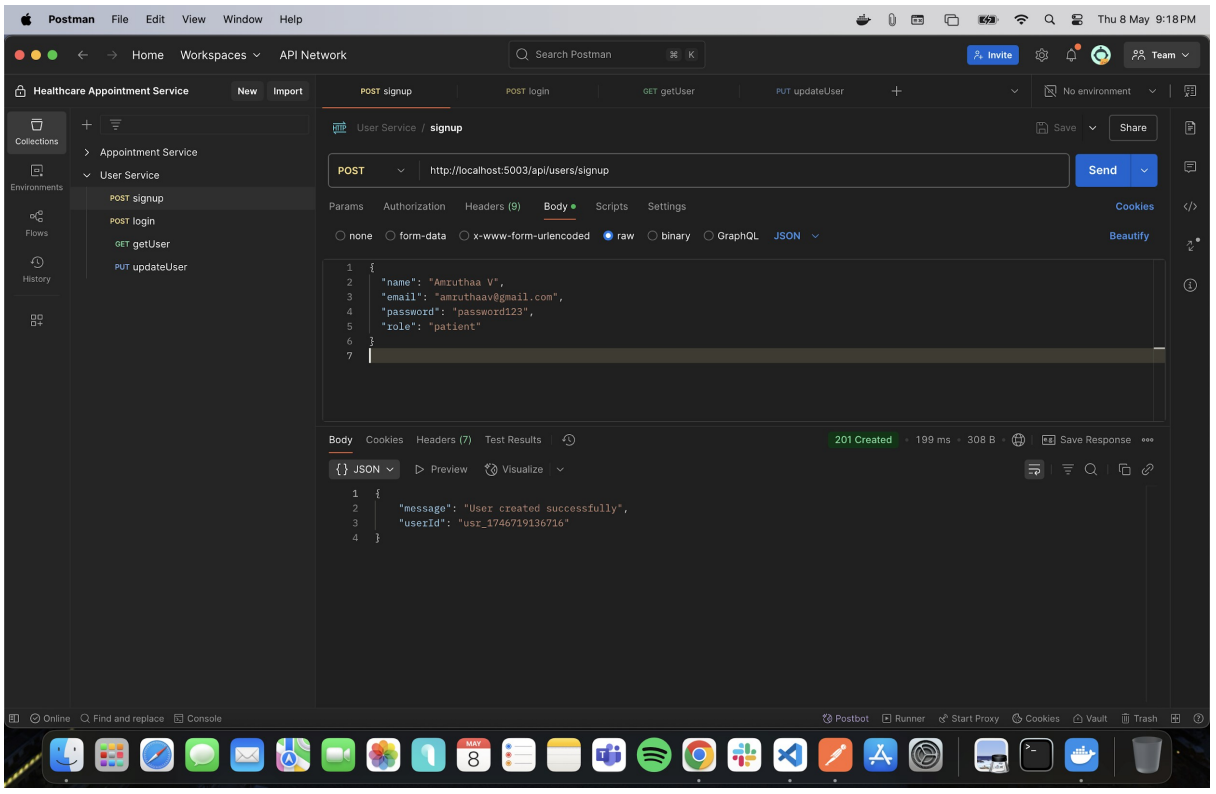
DockerFile

```
# Using Linux Alpine base image for .NET 8 runtime (compatible with Mac M1/M2, Linux)
FROM mcr.microsoft.com/dotnet/aspnet:8.0-alpine AS base
WORKDIR /app
EXPOSE 8080
EXPOSE 8081

# Build stage - SDK image to build the app
FROM mcr.microsoft.com/dotnet/sdk:8.0-alpine AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src

# Copy all the .csproj files and restore dependencies
COPY ["NotificationService.Api/NotificationService.Api.csproj", "NotificationService.Api/"]
COPY ["NotificationService.Application/NotificationService.Application.csproj",
"NotificationService.Application/"]
```

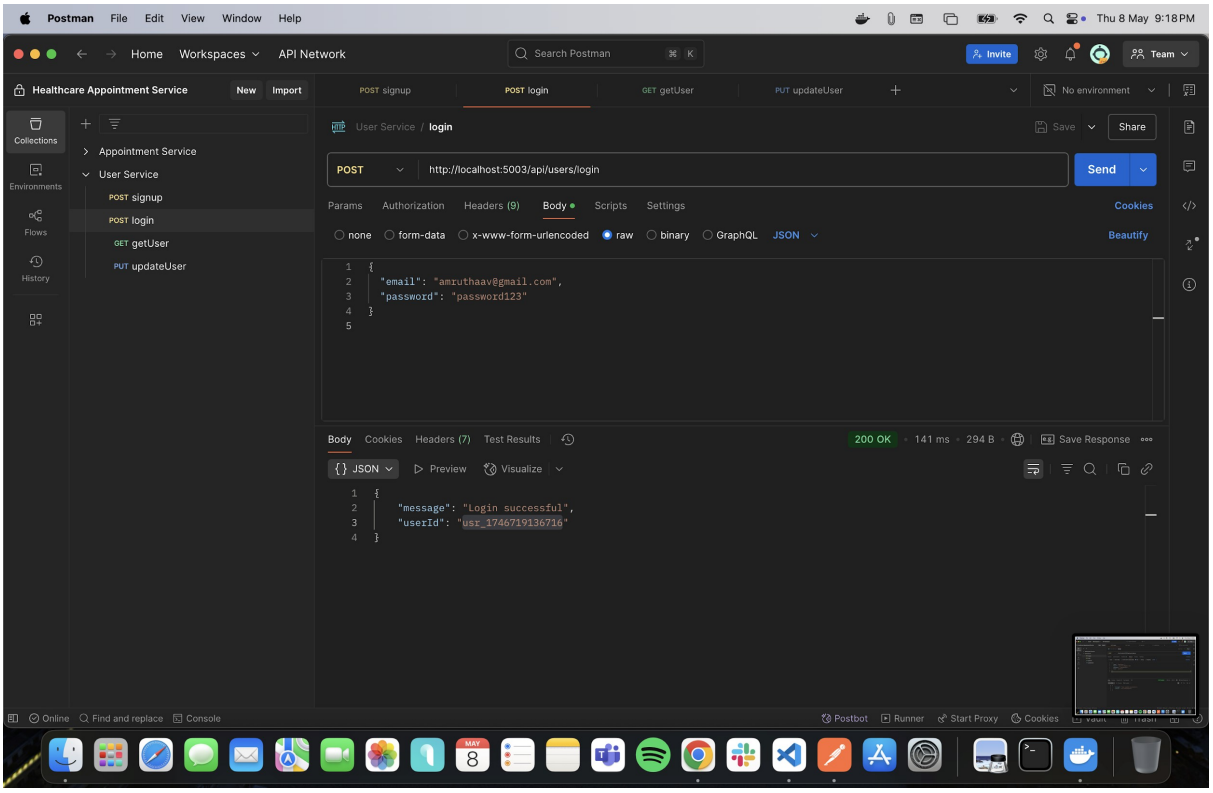
```
COPY ["NotificationService.Infrastructure/NotificationService.Infrastructure.csproj",
"NotificationService.Infrastructure/"]
COPY ["NotificationService.Domain/NotificationService.Domain.csproj",
"NotificationService.Domain/"]
RUN dotnet restore "NotificationService.Api/NotificationService.Api.csproj"


# Copy the entire source code
COPY . .

# Build the project
WORKDIR "/src/NotificationService.Api"
RUN dotnet build "NotificationService.Api.csproj" -c $BUILD_CONFIGURATION -o /app/build

# Publish the build to a folder
FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "NotificationService.Api.csproj" -c $BUILD_CONFIGURATION -o /app/publish
/p:UseAppHost=false

# Final runtime image
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "NotificationService.Api.dll"]
```

Screenshots of API testing

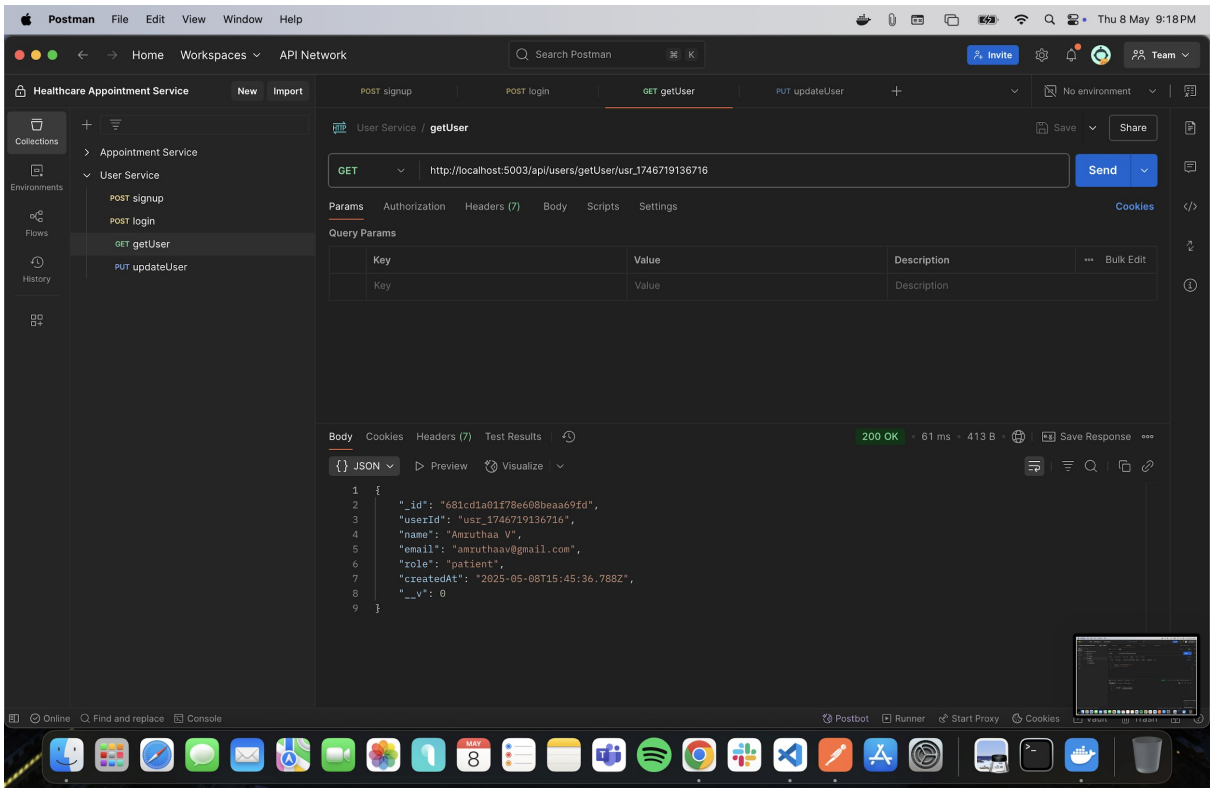
1. User Service



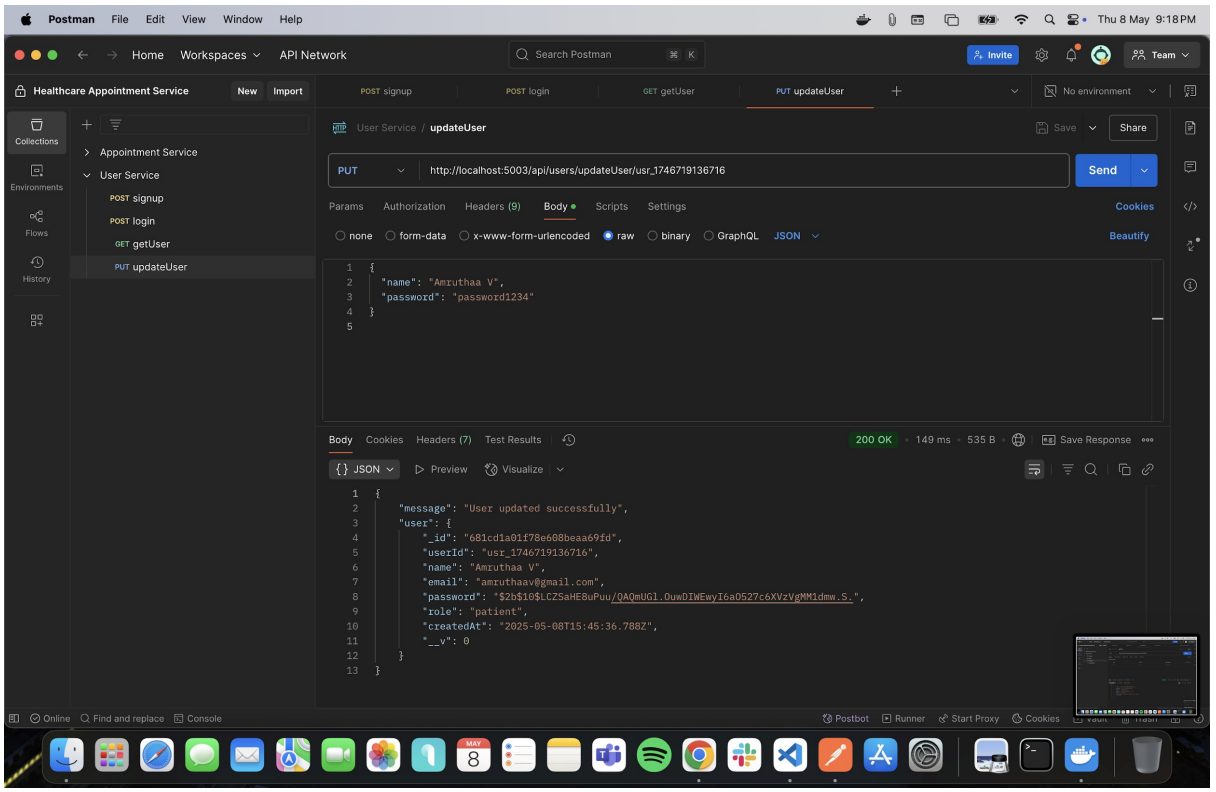
Detail: API response for user sign-up



Detail: API response for user log-in



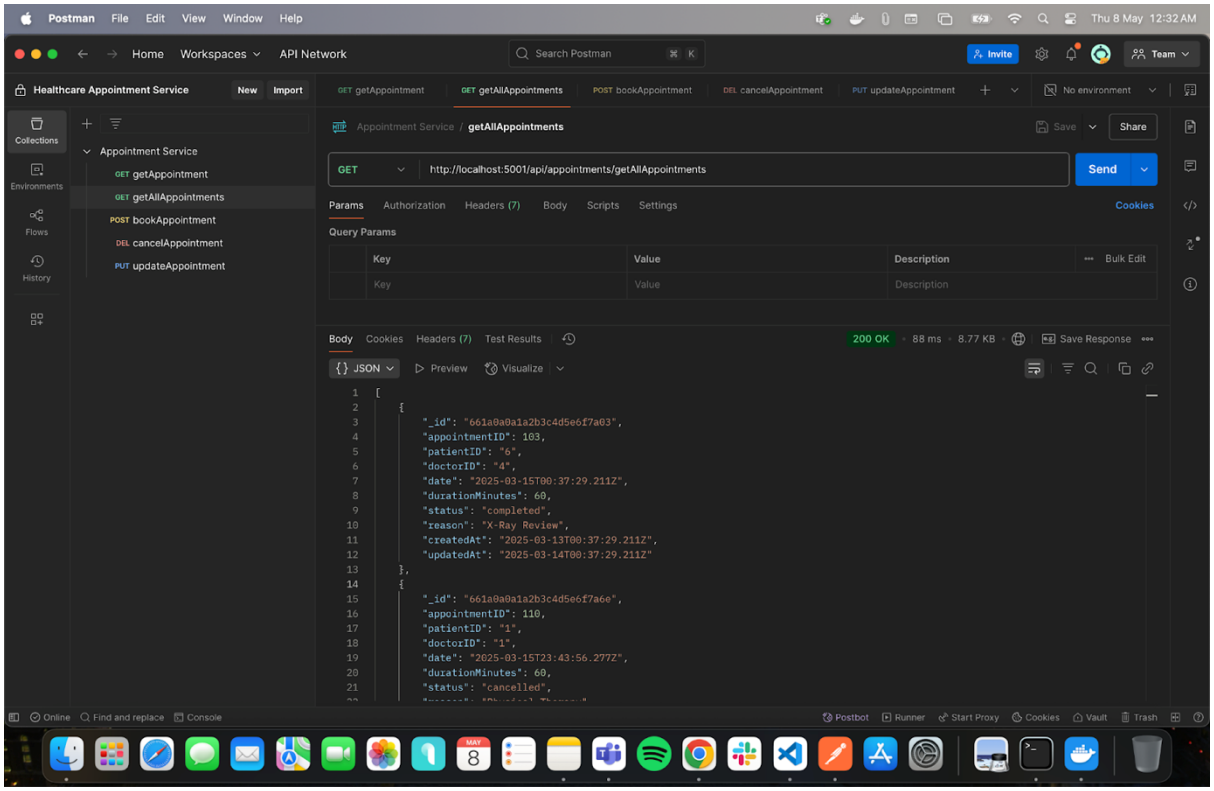
Detail: API response for getUser



Detail: API response for

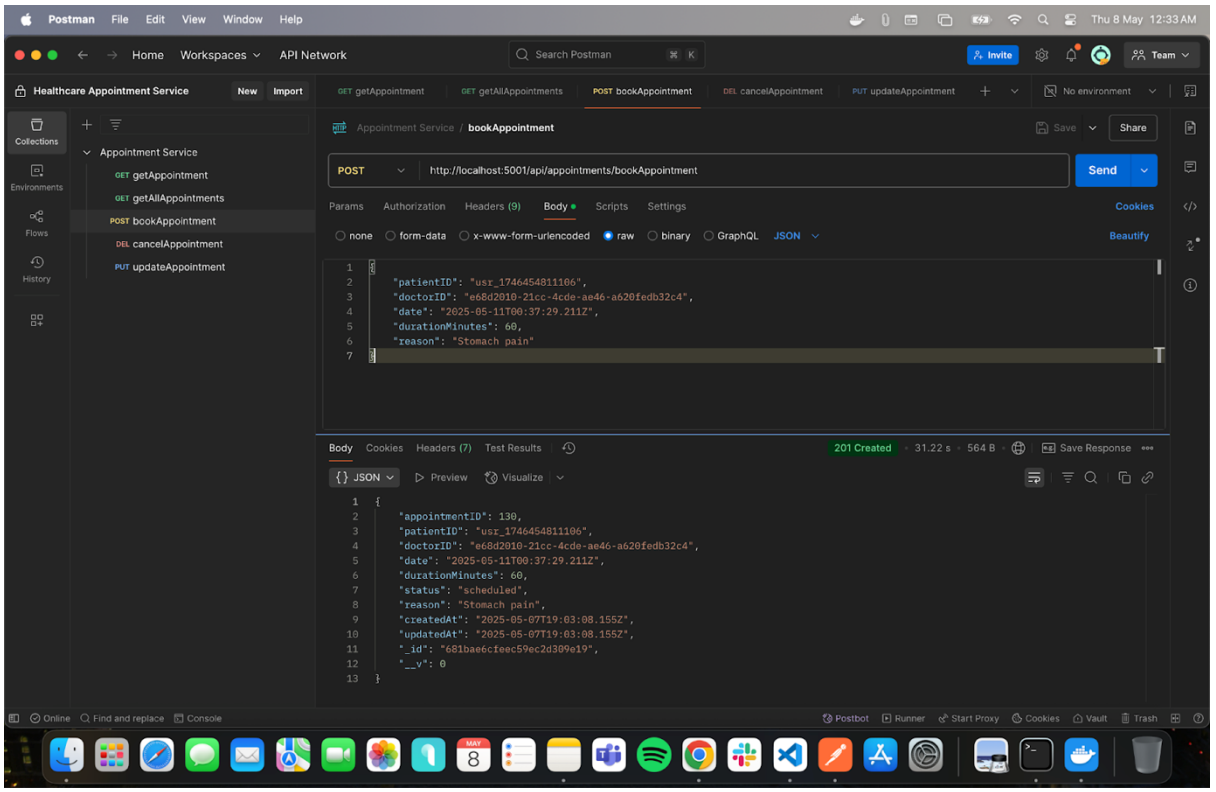
update user

2. Appointment Service

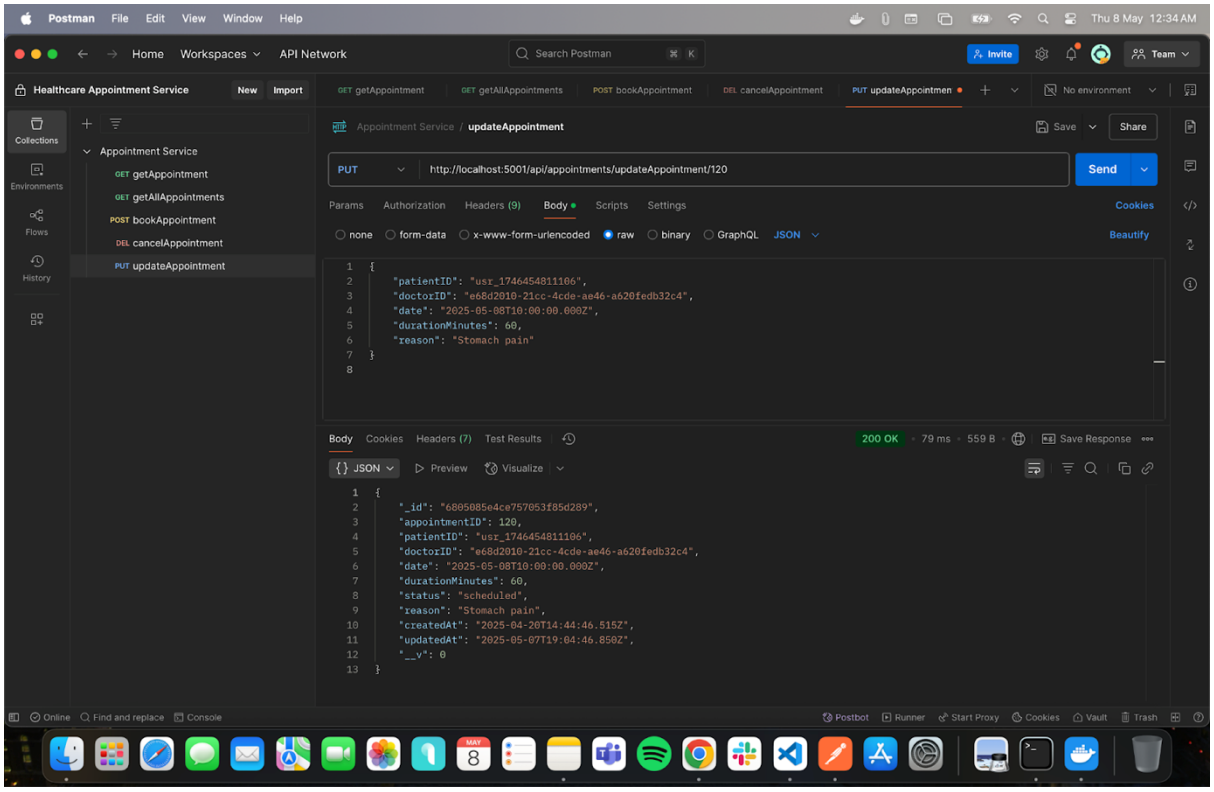


Details: API response for getting all appointments

Detail: API response for

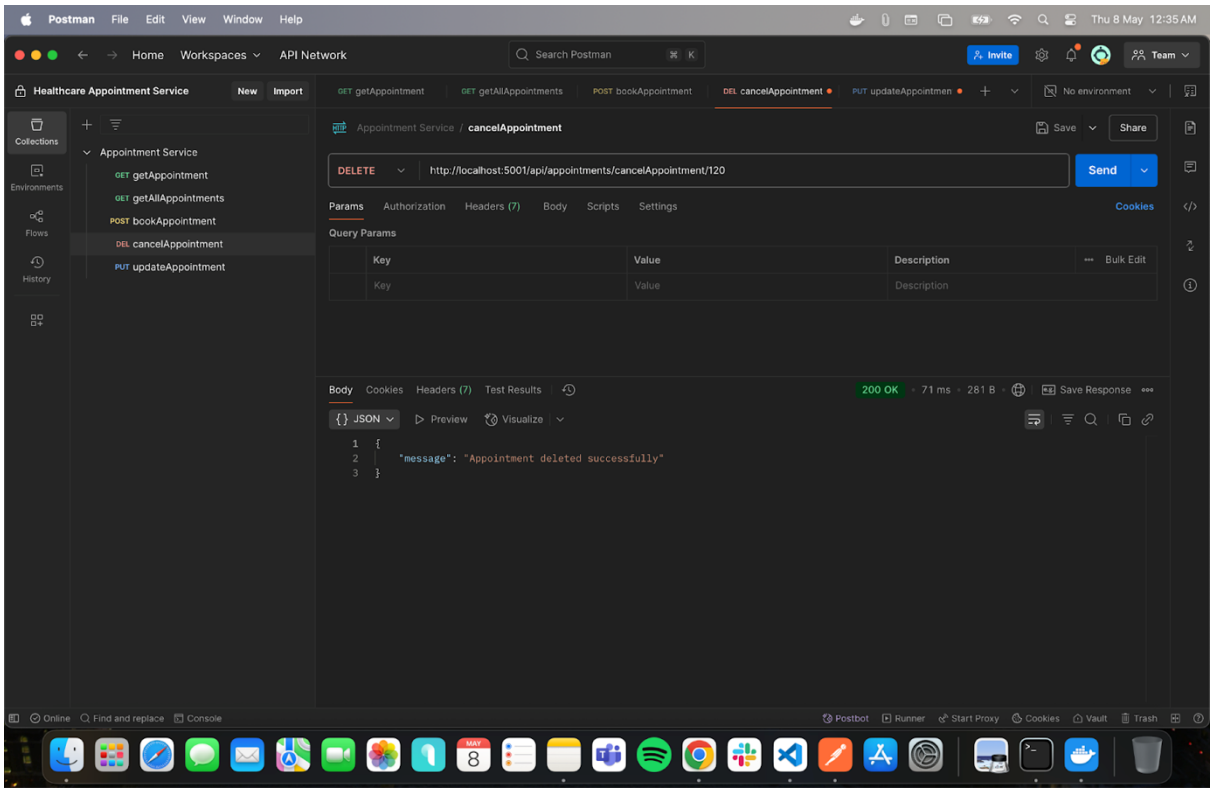


booking an appointment



Detail: API response for updating an appointment

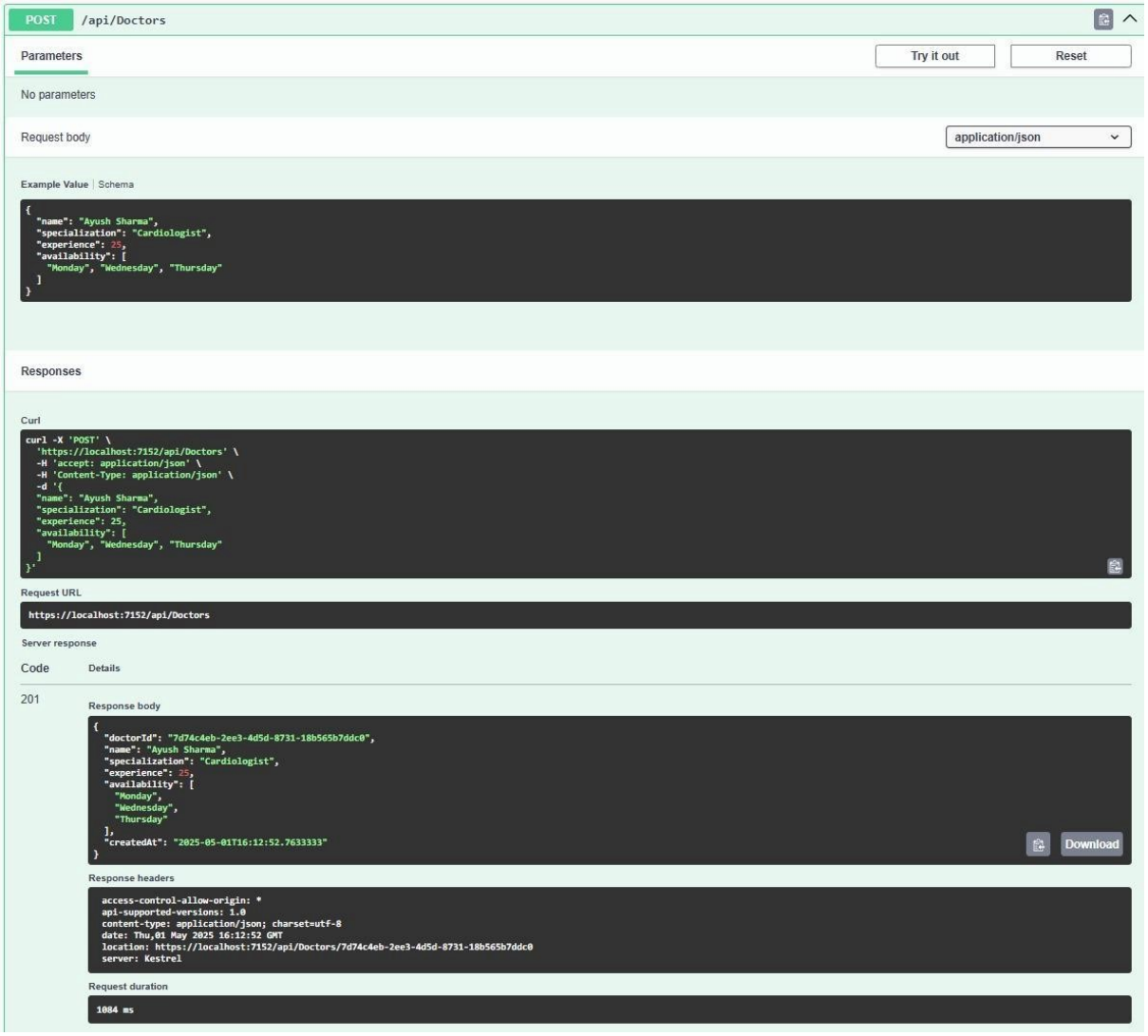
Detail: API response for



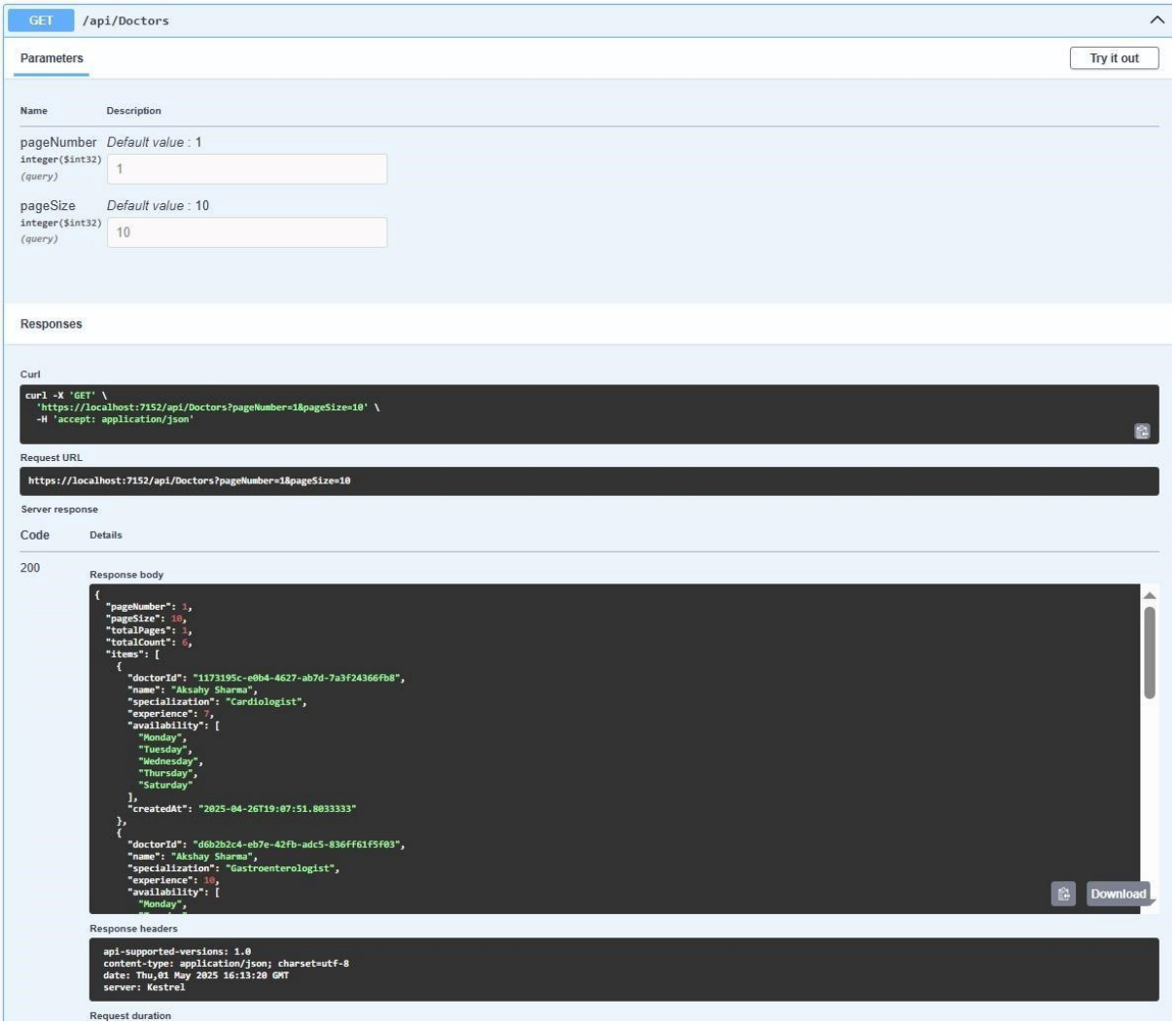
cancelling an appointment

Detail: API response for

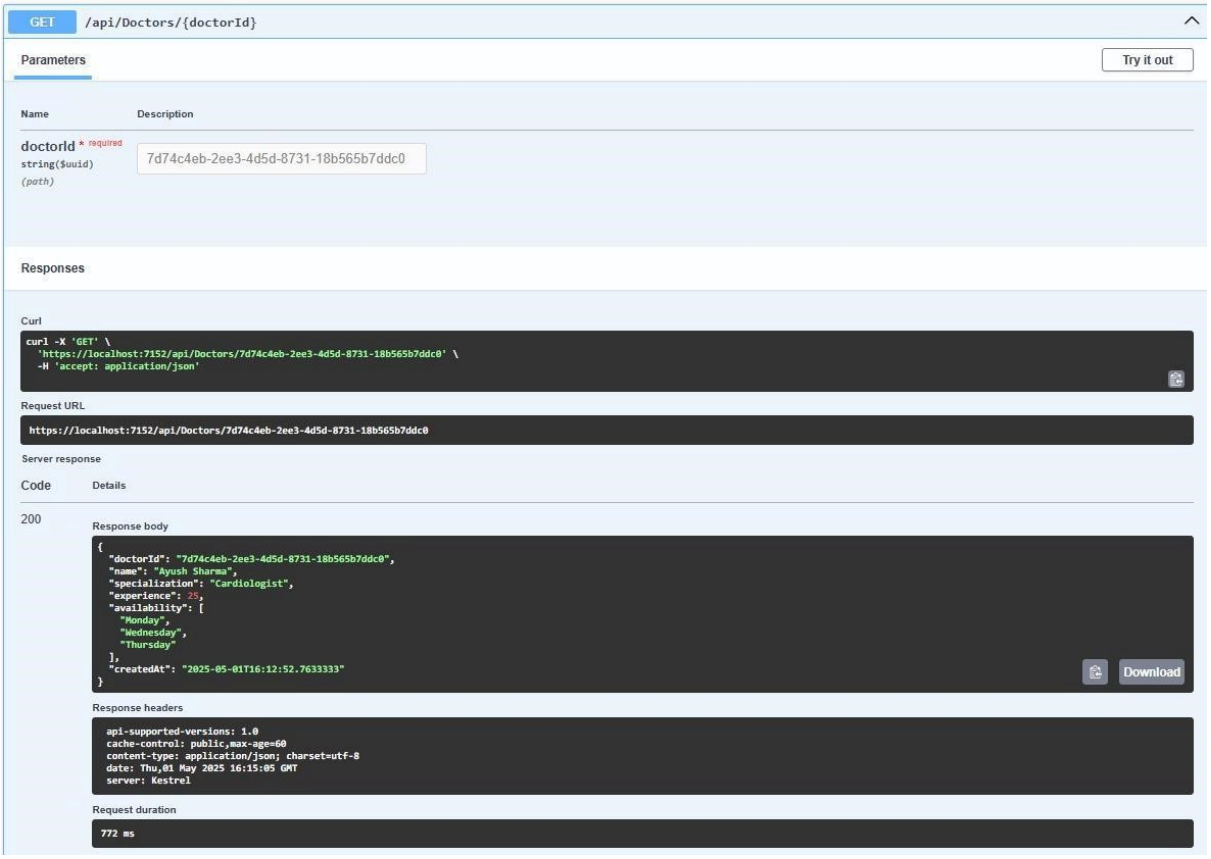
3. Doctor service



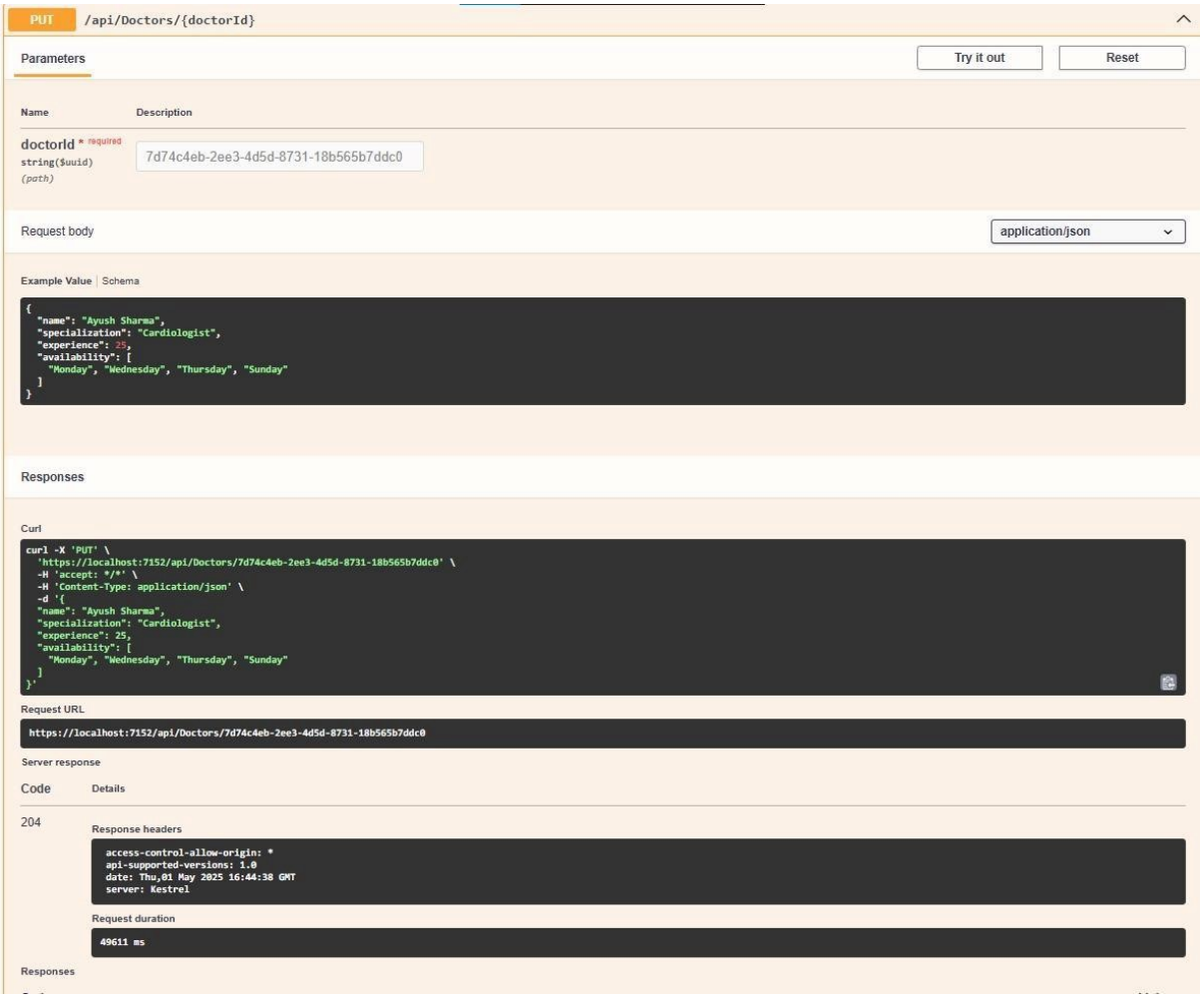
Detail: API response for creating a doctor



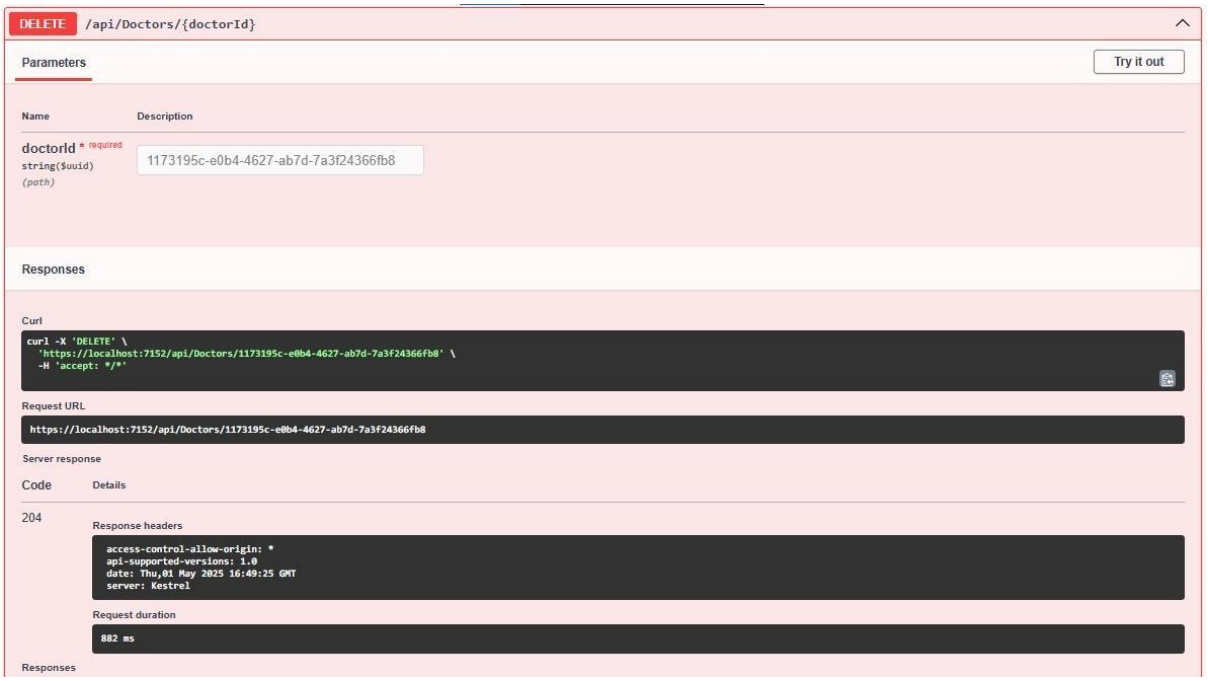
Detail: API response for getting all doctors



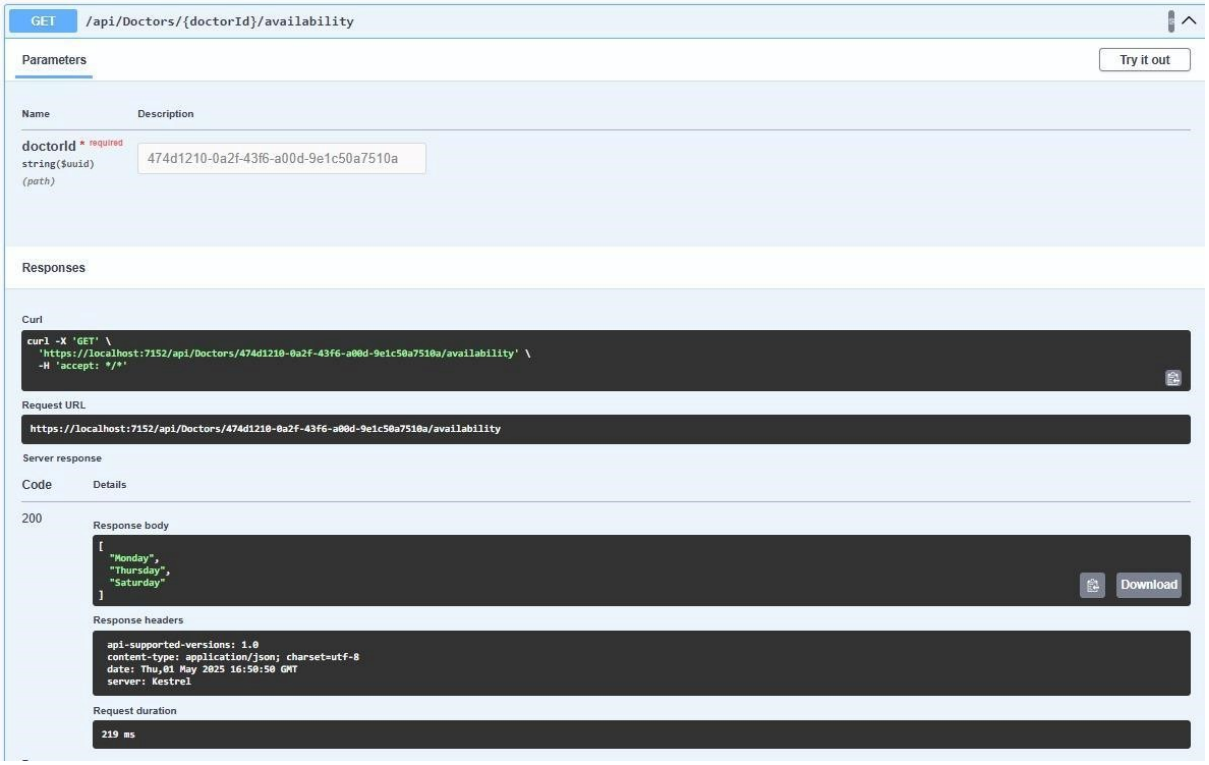
Detail: API response to get details of a particular doctor



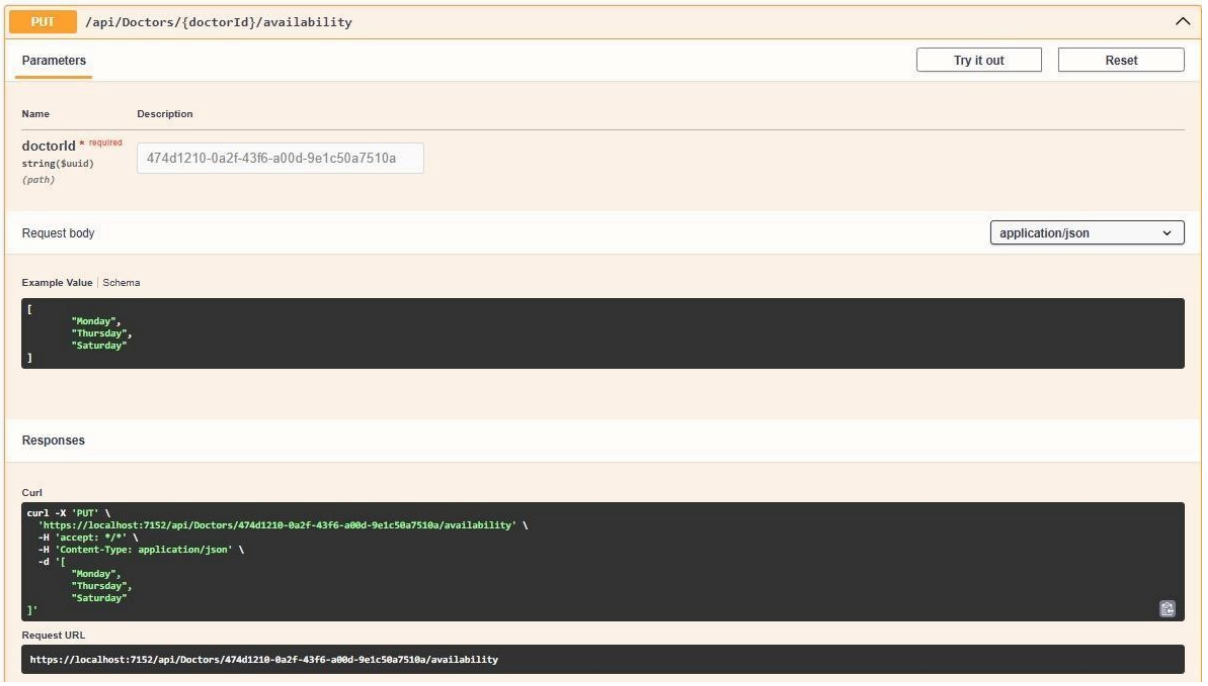
Detail: API response to update a doctor details



Detail: API response to delete a doctor information



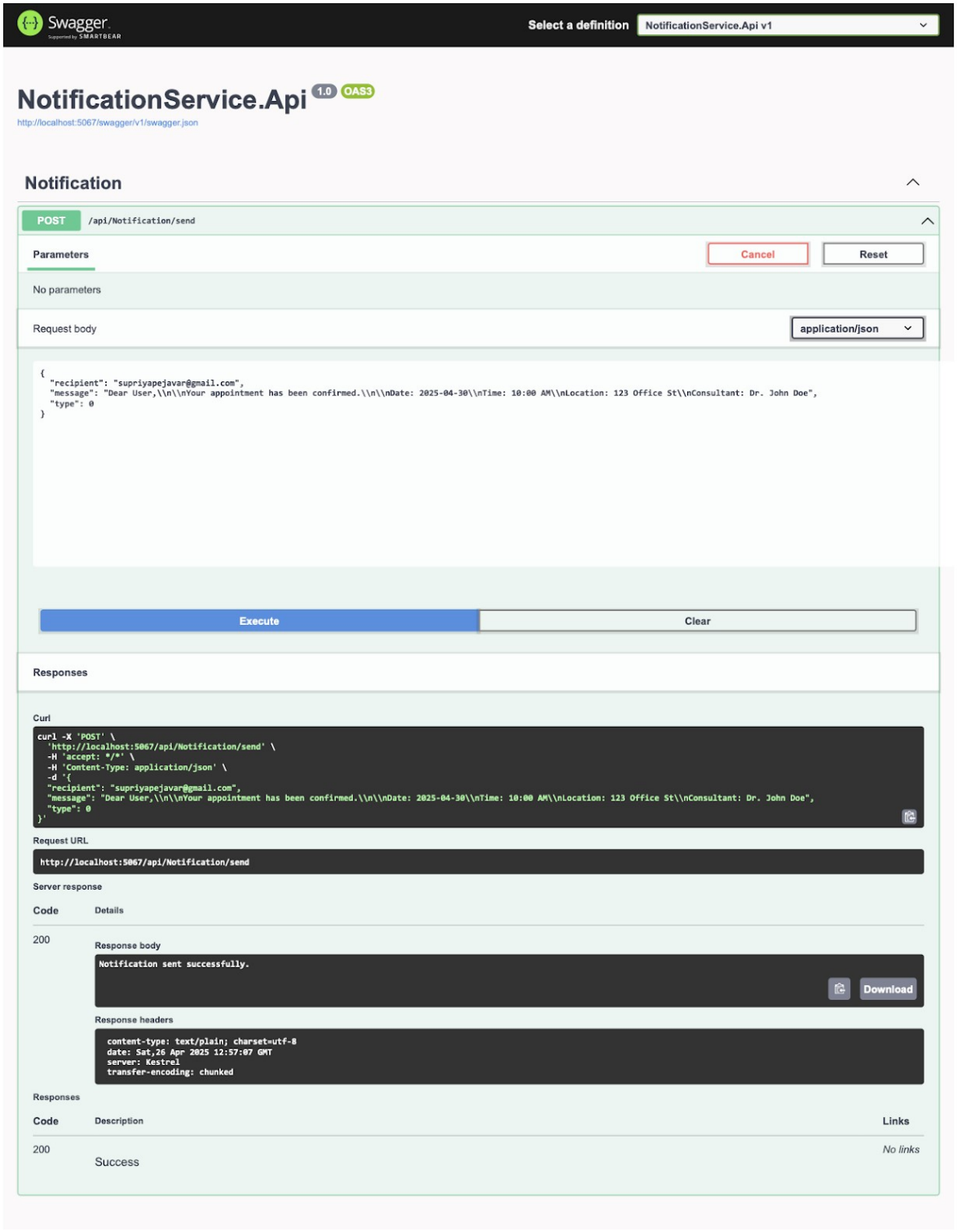
Detail: API response to get the availability of a doctor



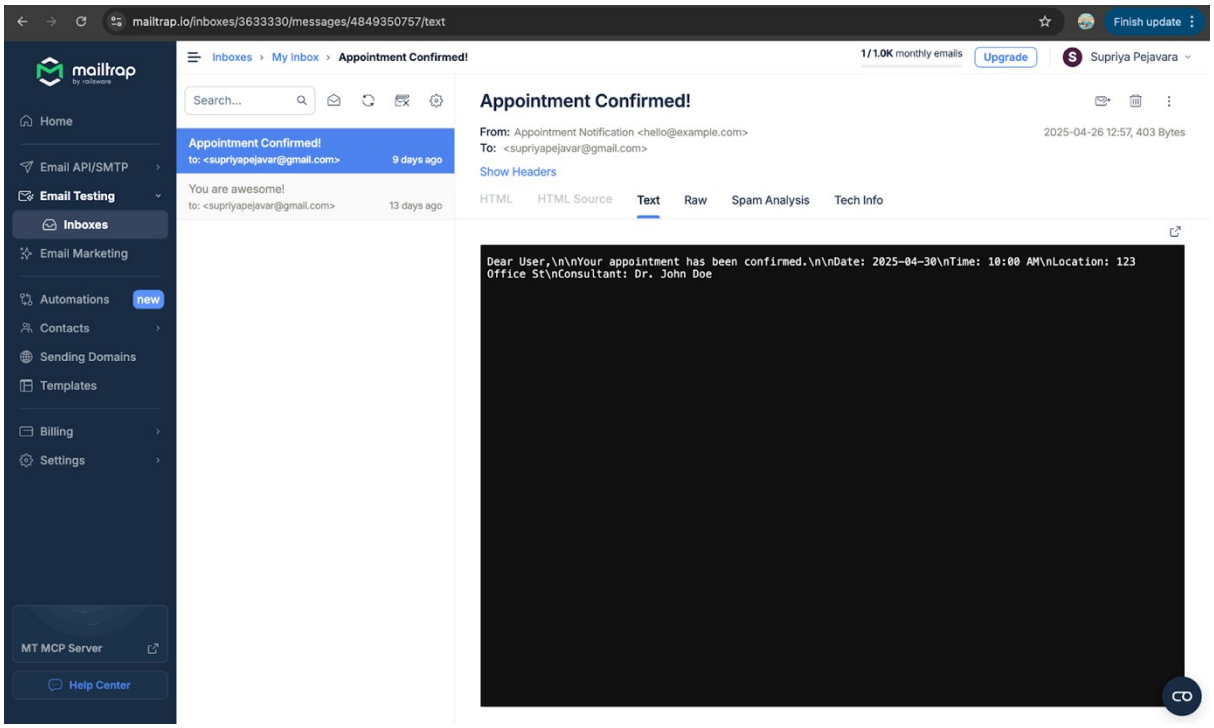
Detail: API response to update a doctor's availability.

4. Notification service

Detail: API response for user sign-up



Detail: API response to send a notification



Detail: Sample response of an email in Mailtrap.

Docker Deployment

What is Docker, and why are we using it?

Docker is primarily a platform for building, running, and managing containers. You can think of it as a way to package your application and all its dependencies into a single, lightweight unit that can run anywhere.

What Docker does well:

1. Packages your app into a container.
2. Runs containers locally.
3. Provides basic networking and volume support.
4. Useful for development, small-scale testing, or isolated environments.

Kubernetes – Container Orchestration System

Kubernetes (often abbreviated as K8s) is a system to manage multiple containers running across multiple machines, handling things like scaling, self-healing, networking, and deployment automation. Kubernetes builds on top of Docker (or any container runtime) to handle production-level needs.

Docker Commands

1. To stop a running container

```
docker stop service-name
```

Purpose: Stops a running container named service-name.
Why: You typically run this when you want to stop the service (e.g., for updates or rebuilding).

2. To remove a container

```
docker rm service-name
```

Purpose: Removes the container named service-name.
Why: You remove the container to free up resources or to replace it with a new one (often after stopping it).

⚠️ This doesn't remove the image, just the container.

3. To build a Docker image

```
docker build -t microservice-name .
```

Purpose: Builds a Docker image from the current directory (.)
-t microservice-name: Tags the image with the name microservice-name. **Why:**
You use this to create an image that you can run as a container.

4. To start a new container from the image

```
docker run --network healthcare-net \ --name service-name \ -p 5004:8080 \
-d \
microservice-name
```

-d: Runs the container in detached mode (in the background).
--network healthcare-net: Connects the container to a custom Docker network named healthcare-net, allowing it to communicate with other services on that network (like other microservices).
--name service-name: Names the container doctor-service for easier management.
-p 5004:8080: Maps port 8080 inside the container to port 5004 on your machine.
Requests to localhost: 5004 go to the container's app running on port 8080.
microservice-name: Specifies the image to run.

Kubernetes Commands:

1. **Switch to Minikube Docker**

```
eval $(minikube docker-env)
```

This command sets your terminal environment so that Docker commands are executed inside Minikube's Docker daemon rather than your local machine.
This is crucial because if you build the image locally (outside Minikube), Kubernetes won't be able to see it.

2. **Build the image inside Minikube**

```
docker build -t microservice-name:local .
```

docker build: Builds a Docker image.
-t microservice-name:local: Tags the image with a name and version (:local is a custom tag).
.: Tells Docker to use the current directory (where the Dockerfile is) as the build context.

3. **Deploy it to Kubernetes**

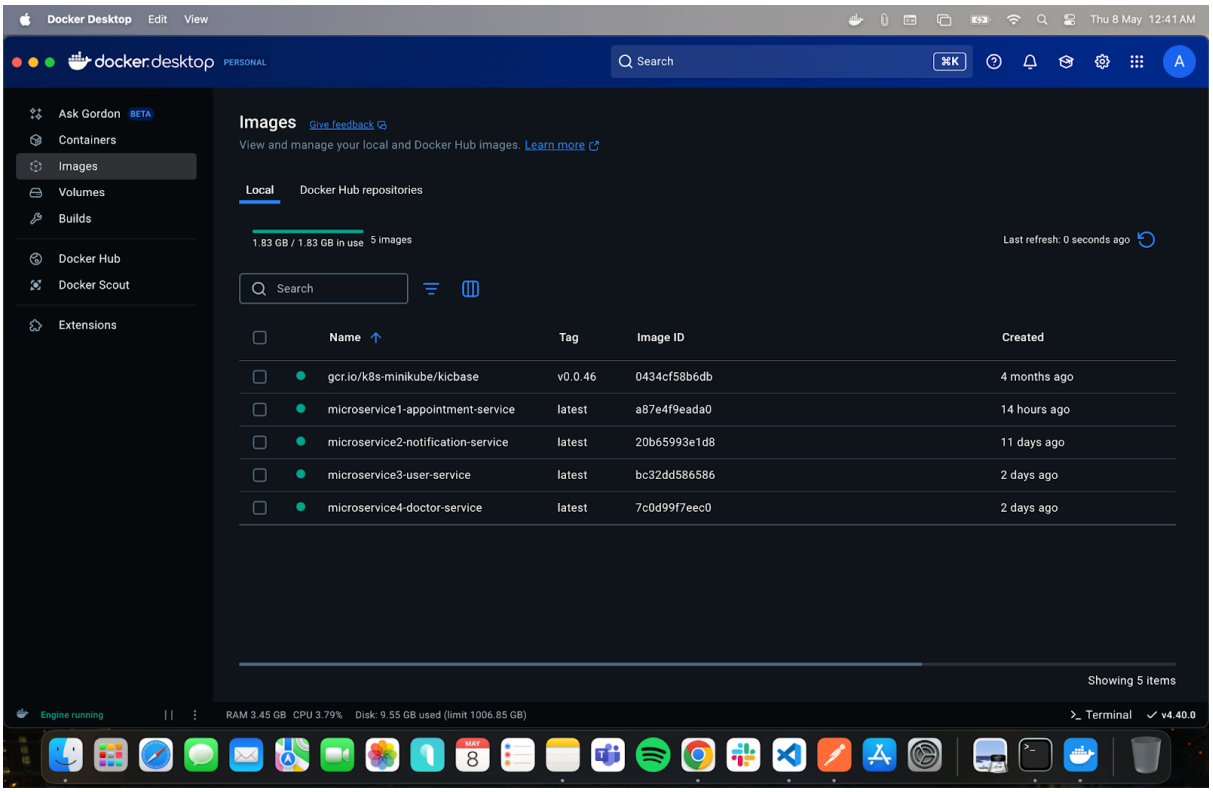
```
kubectl apply -f k8s/service-name.yaml
```

kubectl apply -f: Applies a configuration file to your Kubernetes cluster. **k8s/service-name.yaml:** The YAML file defines the Kubernetes resources (like Deployment, Service, etc.) for your microservice.

4. **Verify it's running**

```
kubectl get pods
```

It is used in Kubernetes to list all the pods running in the current namespace of your Kubernetes cluster. **kubectl:** The command-line tool for interacting with Kubernetes clusters.
get: Retrieves resources. **pods:** Specifies the type of resource you're querying (in this case, Pods).


Docker Desktop Edit View
Thu 8 May 12:41 AM
docker desktop PERSONAL
Search ⌘K
Ask Gordon BETA
Containers
Images
Volumes
Builds
Docker Hub
Docker Scout
Extensions
Images
Give feedback
View and manage your local and Docker Hub images. Learn more
Local
Docker Hub repositories
1.83 GB / 1.83 GB in use
5 images
Last refresh: 0 seconds ago
Search
Name
Tag
Image ID
Created
gcr.io/k8s-minikube/kicbase v0.0.46 0434cf58b6db 4 months ago
microservice1-appointment-service latest a87e4f9eada0 14 hours ago
microservice2-notification-service latest 20b65993e1d8 11 days ago
microservice3-user-service latest bc32dd586586 2 days ago
microservice4-doctor-service latest 7c0d99f7eec0 2 days ago
Showing 5 items
Engine running
RAM 3.45 GB CPU 3.79% Disk: 9.55 GB used (limit 1006.85 GB)
Terminal
v4.40.0

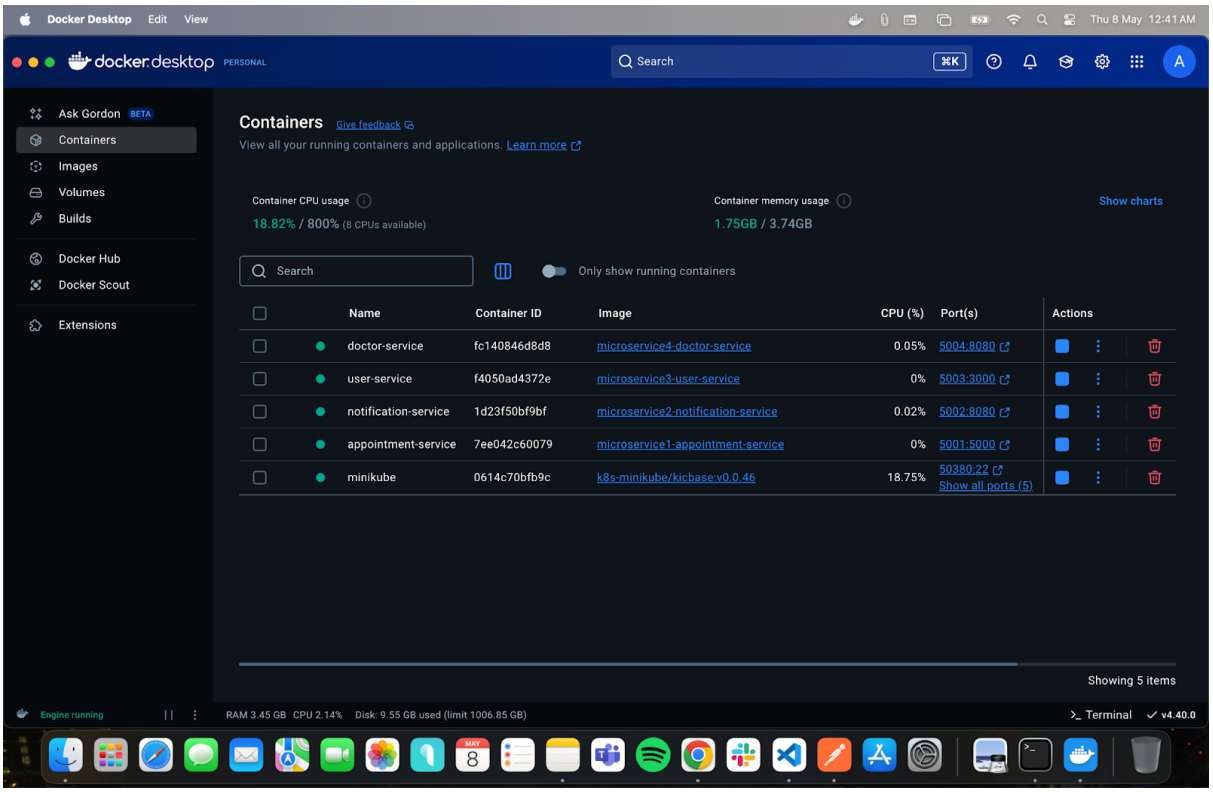

Docker Desktop Edit View
Thu 8 May 12:41 AM
docker desktop PERSONAL
Search ⌘K
Ask Gordon BETA
Containers
Images
Volumes
Builds
Docker Hub
Docker Scout
Extensions
Containers
Give feedback
View all your running containers and applications. Learn more
Container CPU usage
18.82% / 800% (8 CPUs available)
Container memory usage
1.75GB / 3.74GB
Show charts
Search
Only show running containers
Name
Container ID
Image
CPU (%)
Port(s)
Actions
doctor-service fc140846d8d8 microservice4-doctor-service 0.05% 5004:8080
user-service f4050ad4372e microservice3-user-service 0% 5003:3000
notification-service 1d23f50bf9bf microservice2-notification-service 0.02% 5002:8080
appointment-service 7ee042c60079 microservice1-appointment-service 0% 5001:5000
minikube 0614c70bfb9c k8s-minikube/kicbase:v0.0.46 18.75% 50380:22 Show all ports (5)
Showing 5 items
Engine running
RAM 3.45 GB CPU 2.14% Disk: 9.55 GB used (limit 1006.85 GB)
Terminal
v4.40.0

Table for ports

Microservice Name	Host Port	Container Port
user-service	5003	3000
doctor-service	5004	8080
appointment-service	5001	5000
notification-service	5002	8080

Kubernetes Deployment URLs

Microservice	Local URLs
user-service	http://127.0.0.1:50908/
doctor-service	http://127.0.0.1:51116/
appointment-service	http://127.0.0.1:51604/
notification-service	http://127.0.0.1:52187/