

Hospital Management System

Capstone Project

Table of Contents:

- 1) Introduction
- 2) Aim
- 3) Objective
- 4) Project Output
- 5) Architecture
- 6) Advantages

Introduction:

A hospital deals with thousands of patients daily and needs a reliable system to manage appointments, billing, doctors, medical records, and authentication. Manual processes often lead to errors, delays, and inefficiency.

This project implements a **Microservices-based Hospital Management System** to streamline these operations, ensuring security, scalability, and smooth communication between different modules.

Aim of the Project :

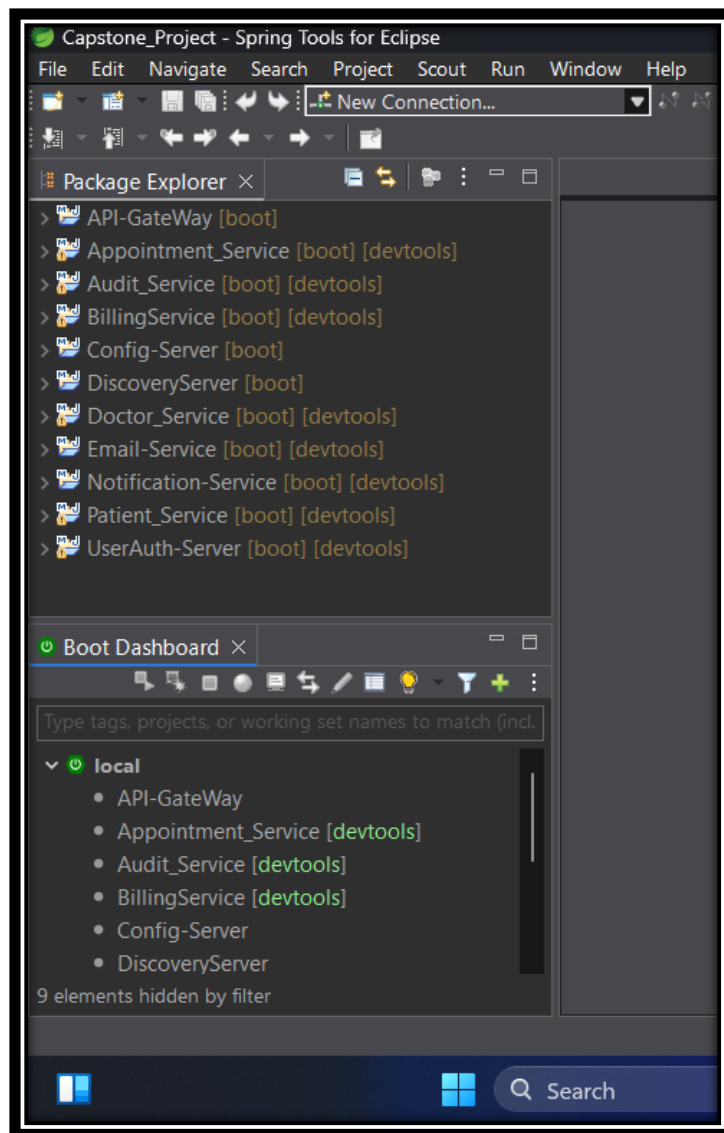
The main aim of the **Hospital Management System (HMS)** is to digitize and automate hospital operations by providing a secure, scalable, and user-friendly platform to manage patients, doctors, appointments, billing, and medical records.

Objective:

- Automate patient registration, doctor scheduling, billing, and medical history tracking.
- Provide secure login for patients, doctors, and admins.
- Enable online appointment booking and payment system.
- Store medical records securely with role-based access.
- Ensure event-driven communication using Kafka.
- Follow best practices like AOP logging, centralized config, and API Gateway for routing.

Hospital Management System OUTPUT

Project Structure:

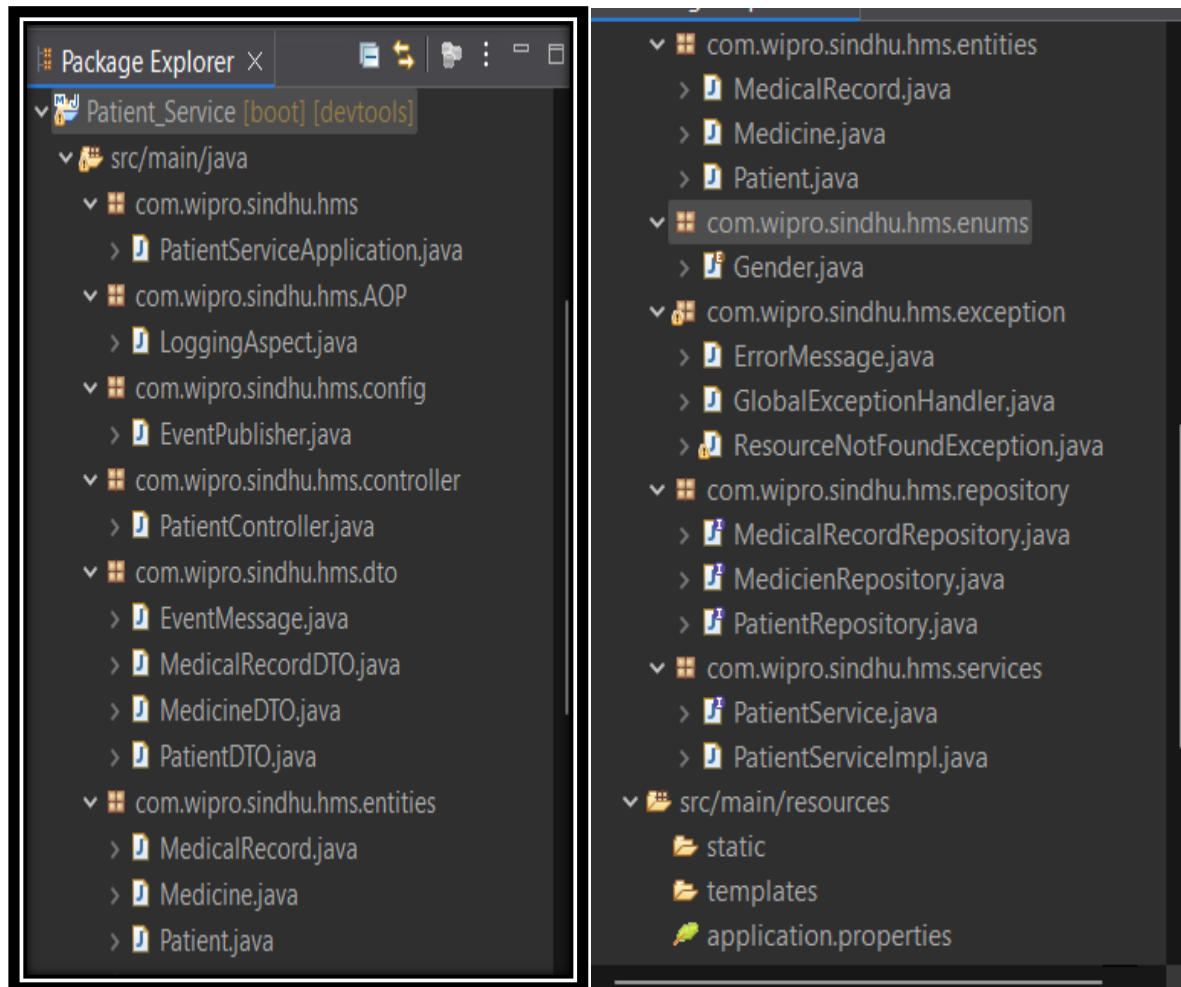


- In this project consist of Patient_Service, Doctor_Service, Appointment_Service, Billing_Service, Audit_Service, UserAuth_Service, Api-Gateway, Config-Server, Discovery-Server.
- And implementing AOP, Kafka, OpenFeign, Config for for Communication purpose.

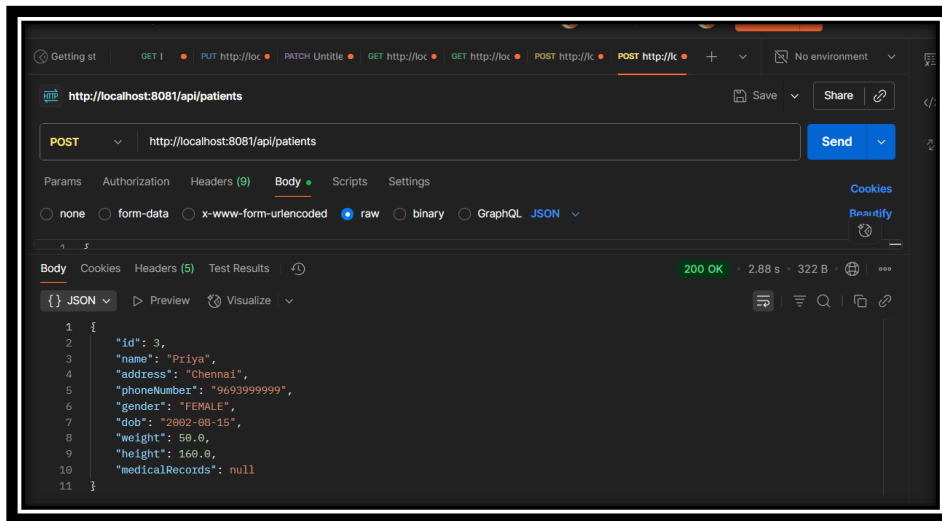
Patient_Service:

- Register new patients (CRUD operations).
- Maintain patient details (profile, history, prescriptions).
- Retrieve medical records securely.
- Communicate with Appointment and Billing services.

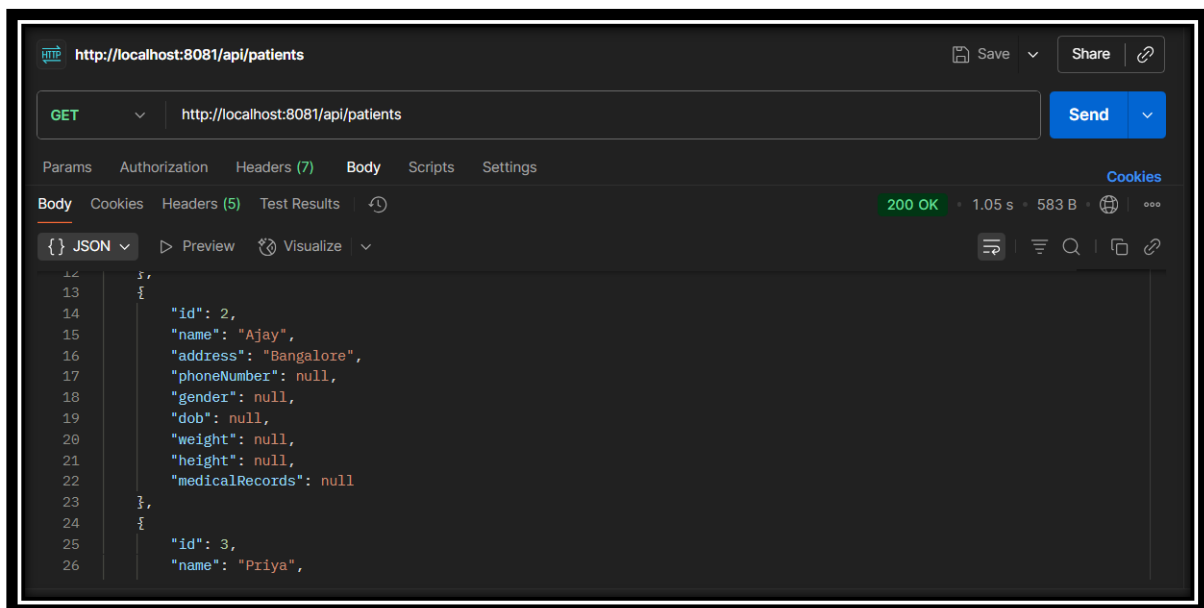
Patient_Service Structure

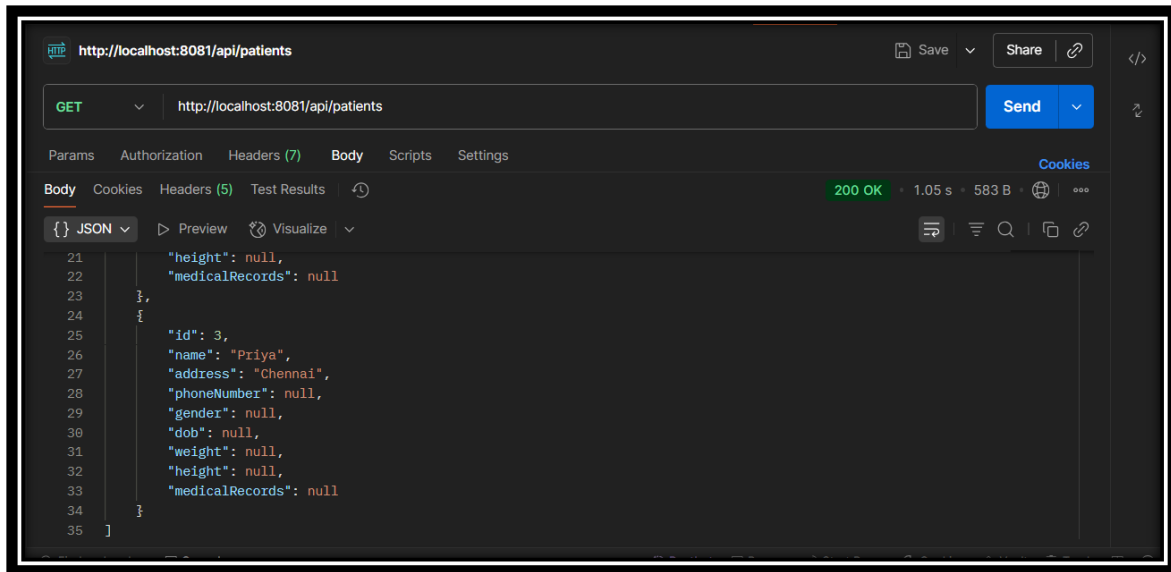


Patient (Create) :- Create a new resource on the server. And Used for inserting/adding data.

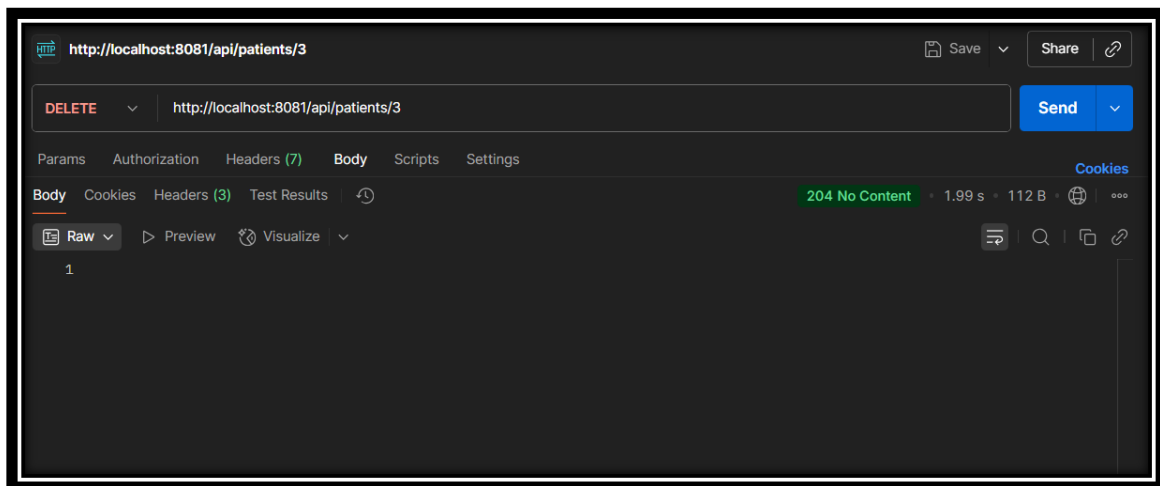


Patient GET :- Retrieve data from the server. And Does not change anything on the server (read-only).

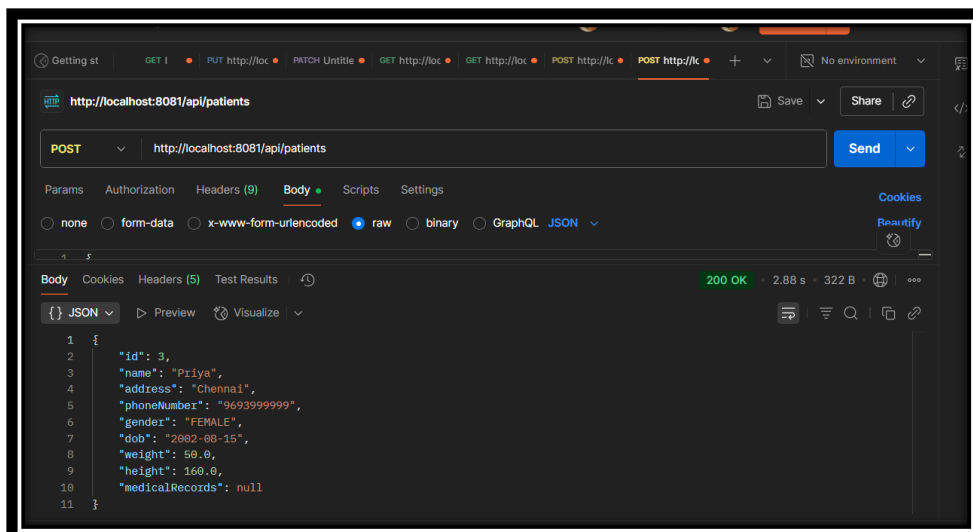




Patient (Delete):- Remove a resource from the server.



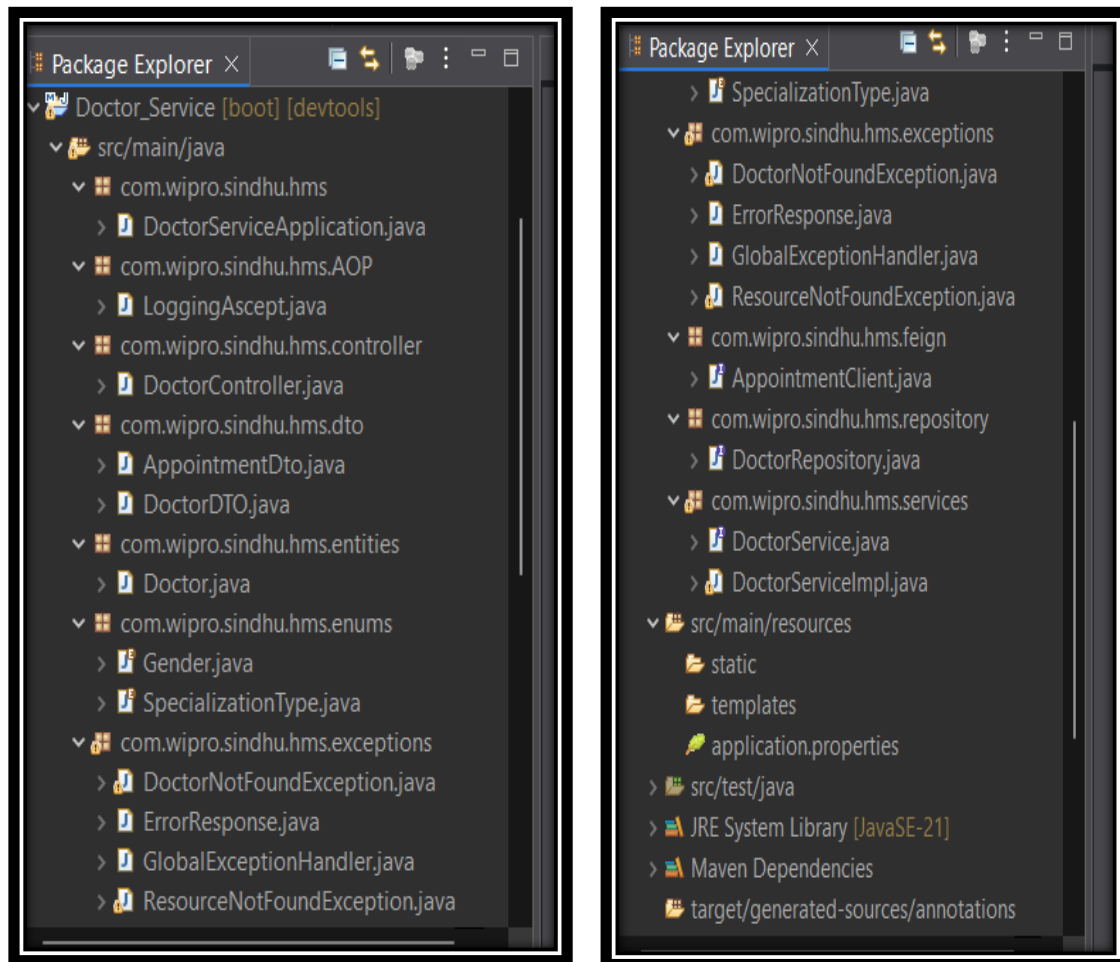
Patient (PUT):- Update an existing resource completely (replace old with new).



Doctor_Service:

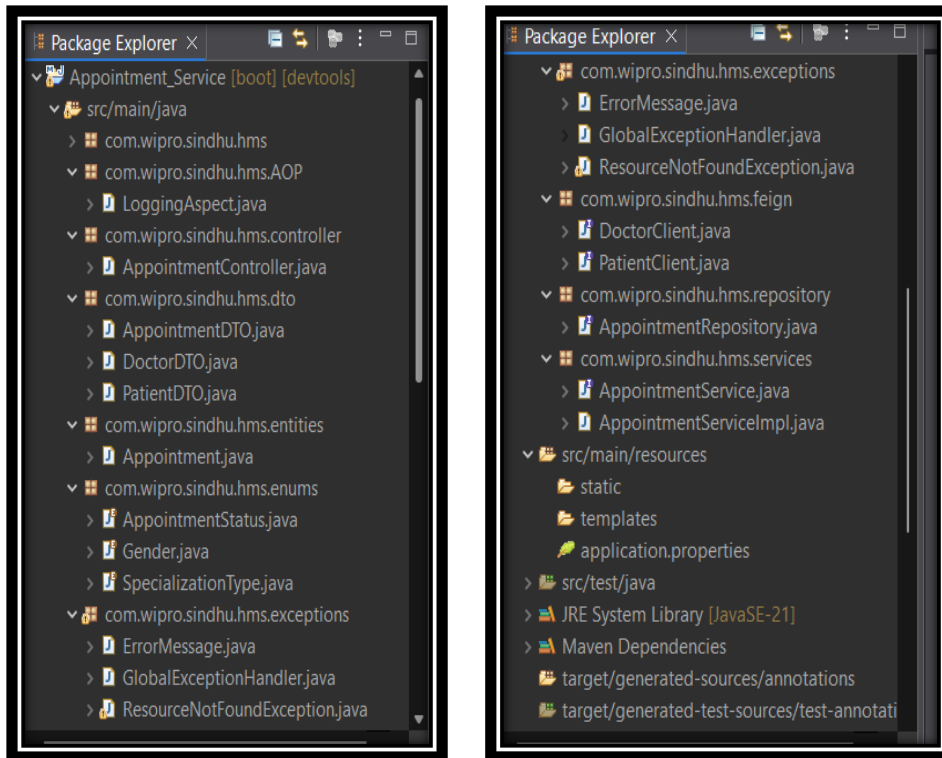
- Manage doctor profiles, departments, and availability.
- View and update patient diagnosis, prescriptions, and history.
- Integrate with Appointment_Service to fetch schedules.

Doctor_Service Project Structure:

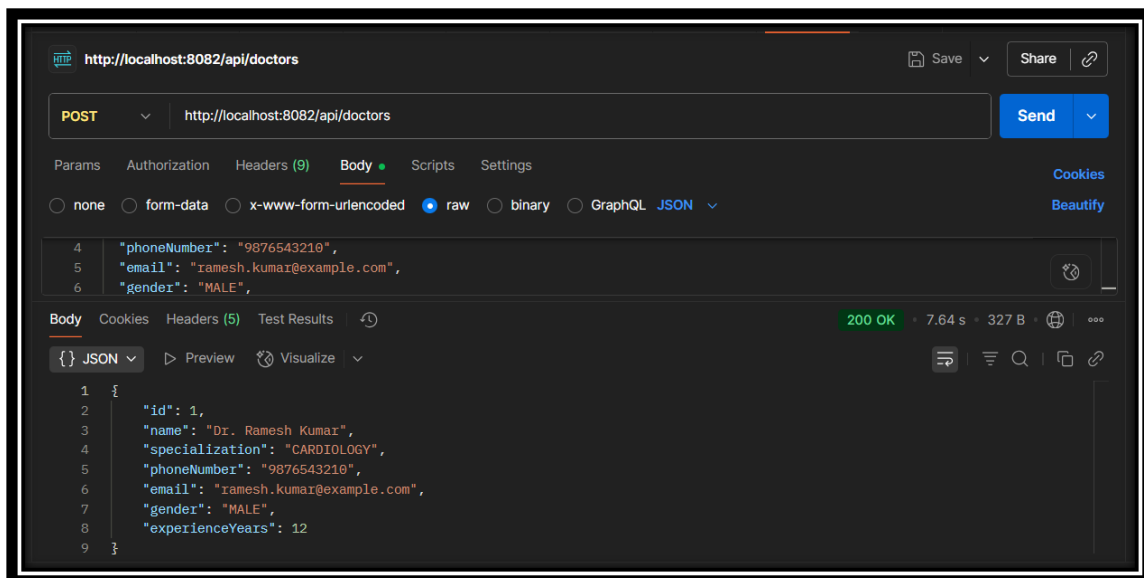


Appointment_Service :

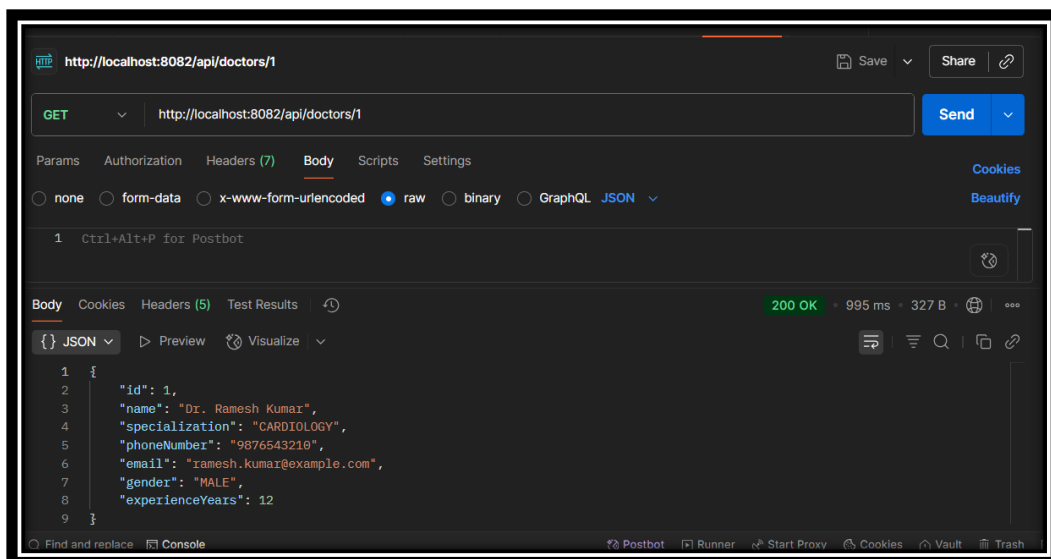
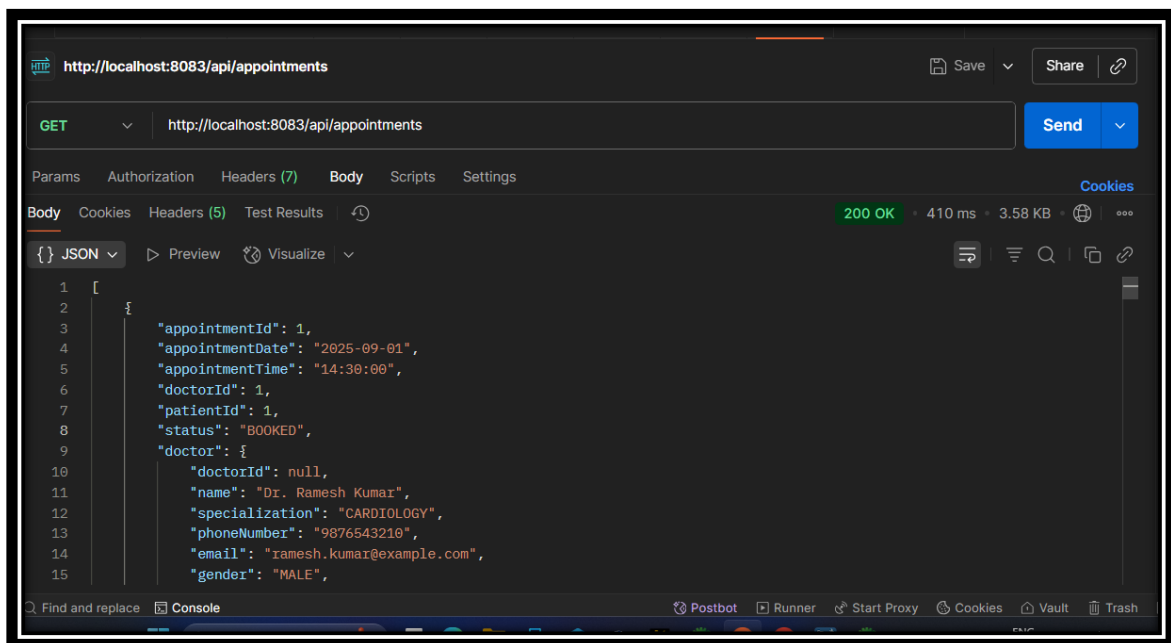
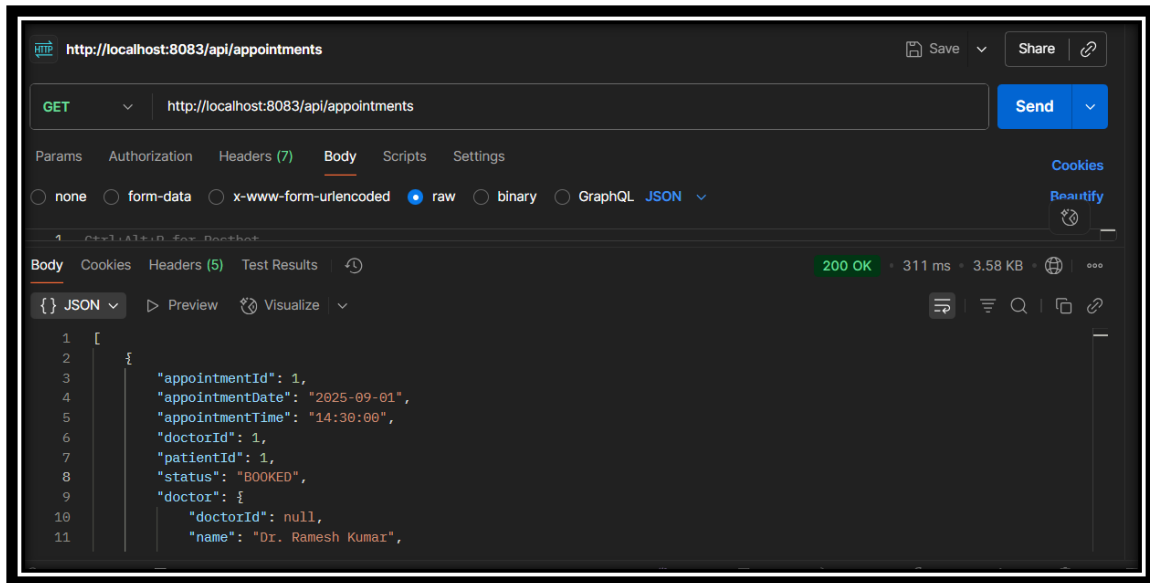
- Handle appointment booking, rescheduling, and cancellations.
- Ensure doctor availability before confirming booking.
- Notify patients & doctors via Notification/Kafka.
- Integrate with Billing for consultation fee payment.



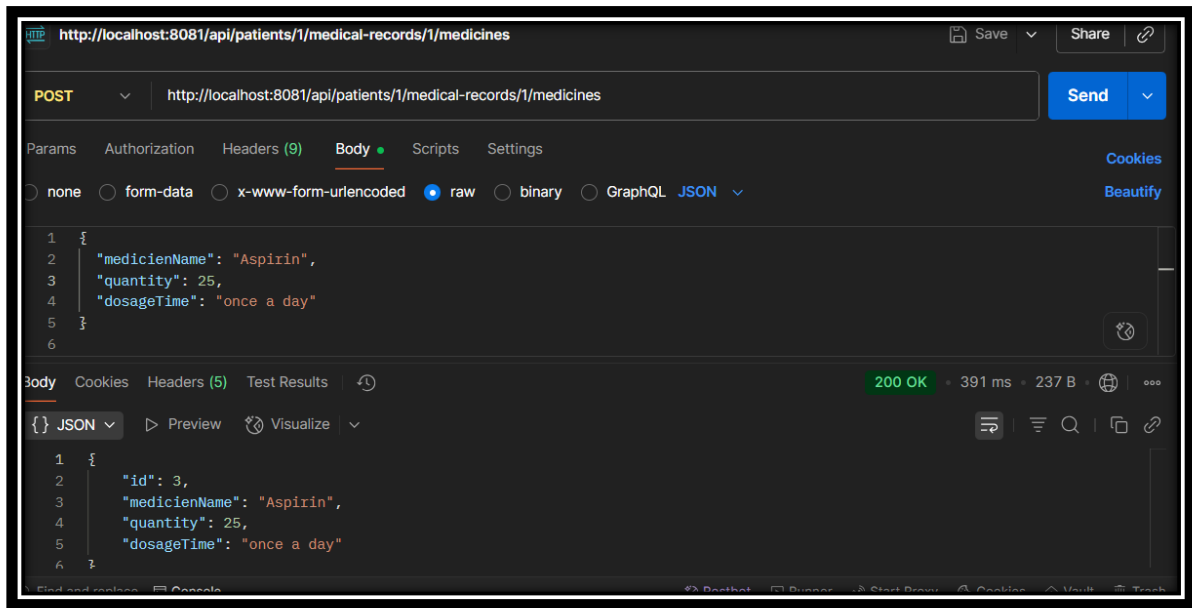
Doctors (Create Patient) :- Create a new resource on the server. And Used for inserting/adding data.



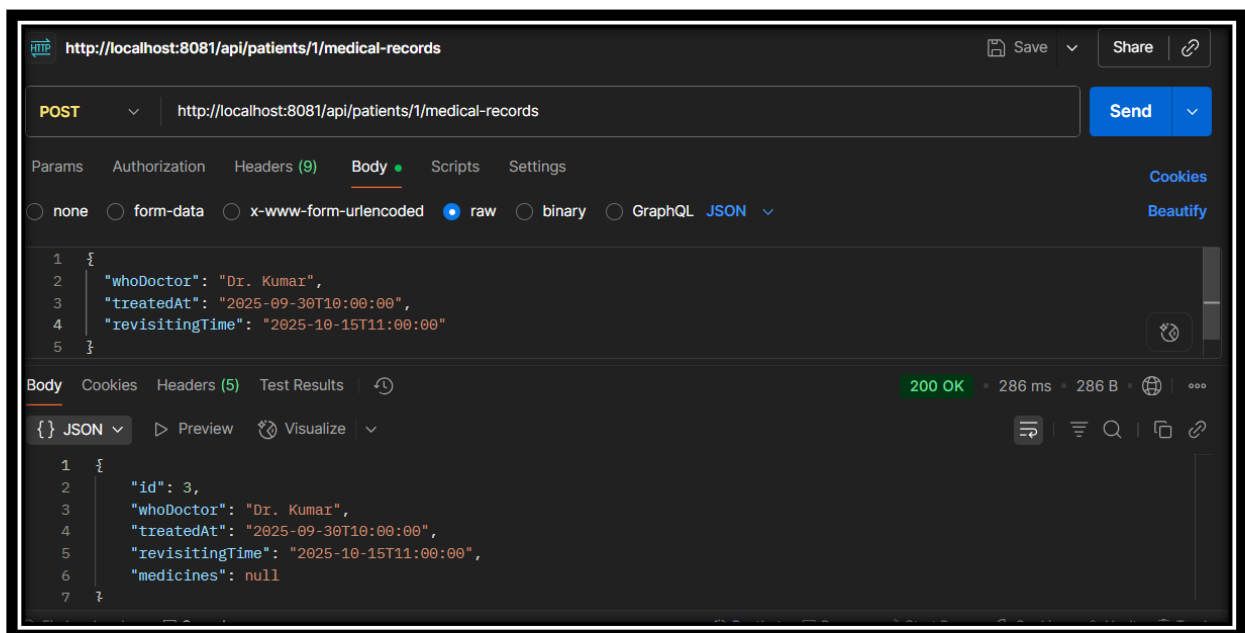
Appointment GET :- Retrieve data from the server. And Does not change anything on the server (read-only).



Medicine (Create) :



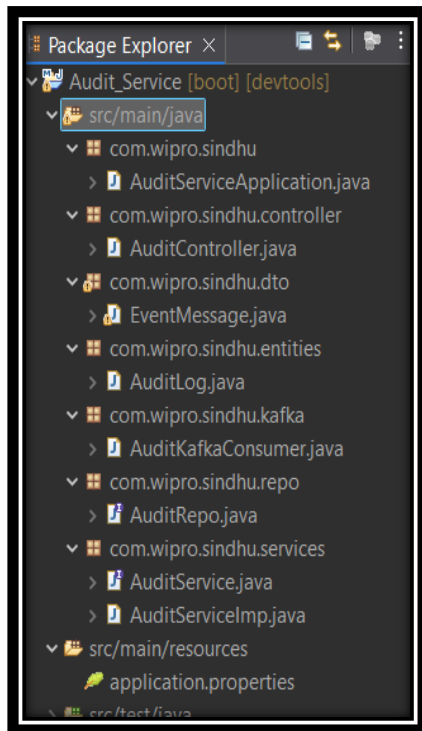
Medicine (Update):



Audit_Service:

- Logs all sensitive actions (login, billing, medical updates).
- Provides audit reports for compliance.
- Ensures traceability & accountability in hospital workflows.

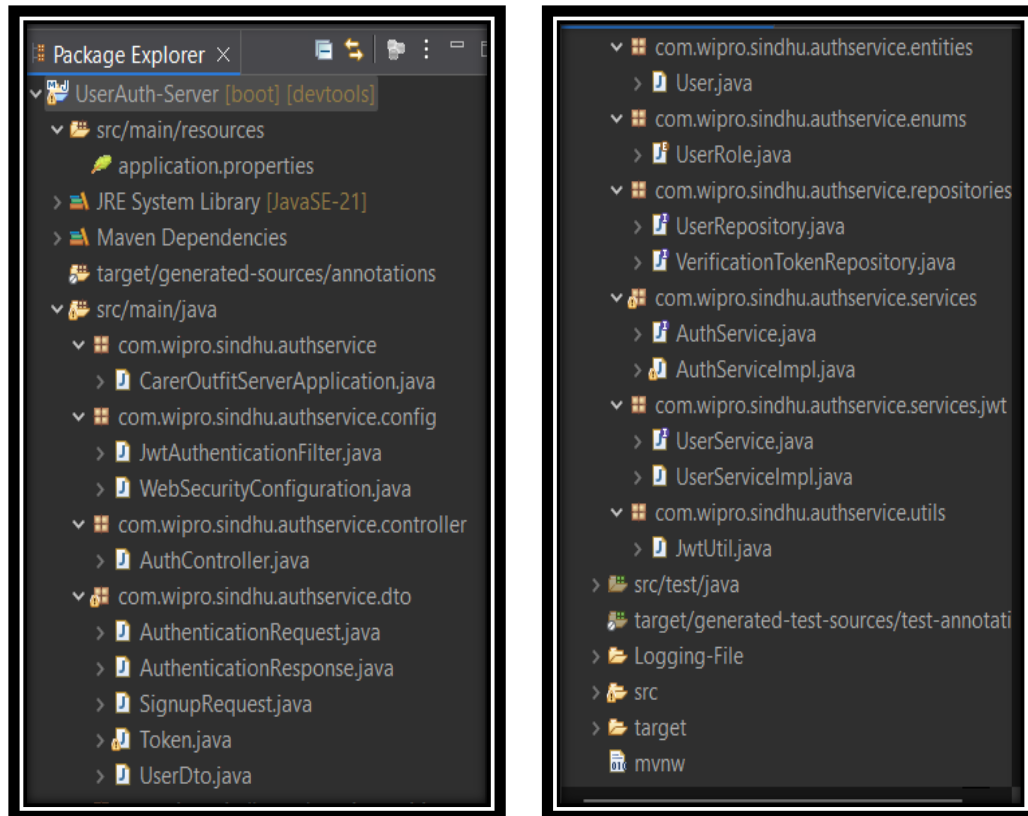
Audit_Service Project Structure:



UserAuth_Service:

- Manage authentication & authorization (JWT/OAuth2).
- Role-based access (Patient, Doctor, Admin).
- Issue tokens for secure communication across services.

UserAuth_Service Project Structure:



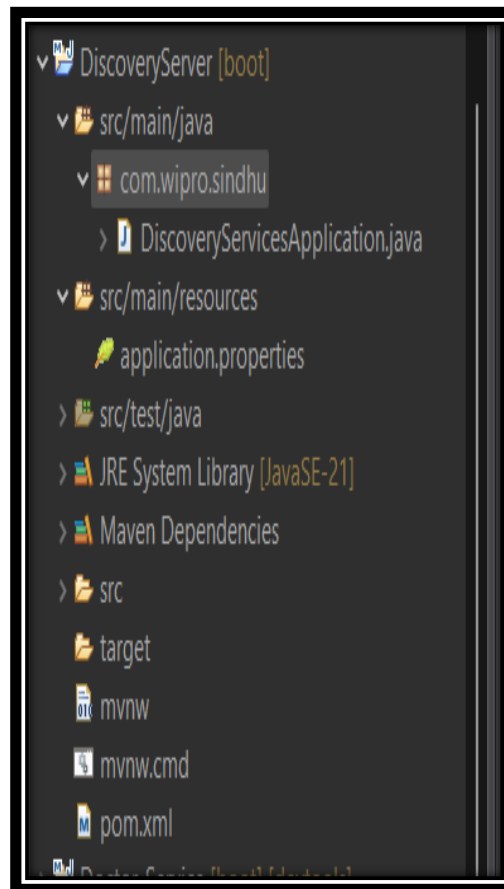
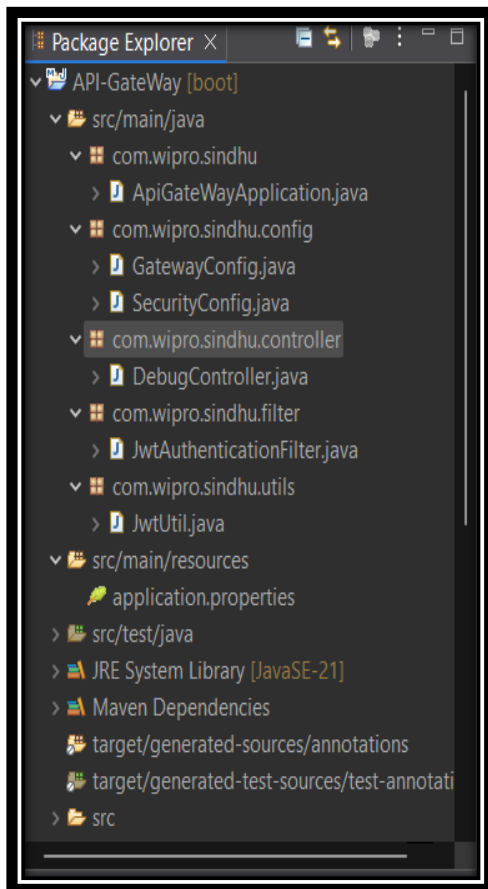
API-Gateway:

- Acts as a single entry point for all clients.
- Routes requests to respective microservices.
- Handles cross-cutting concerns like security, rate-limiting, logging.

Discovery_server:

- Service registry for all microservices.
- Ensures dynamic service discovery.
- Provides load balancing across service instances.

API-Gateway Project Structure: Discovery_Service Project Structure:



AOP (Aspect-Oriented Programming):

- Logging of sensitive actions (payments, prescriptions).
- Exception handling and monitoring.
- Helps in clean code by separating cross-cutting concerns.

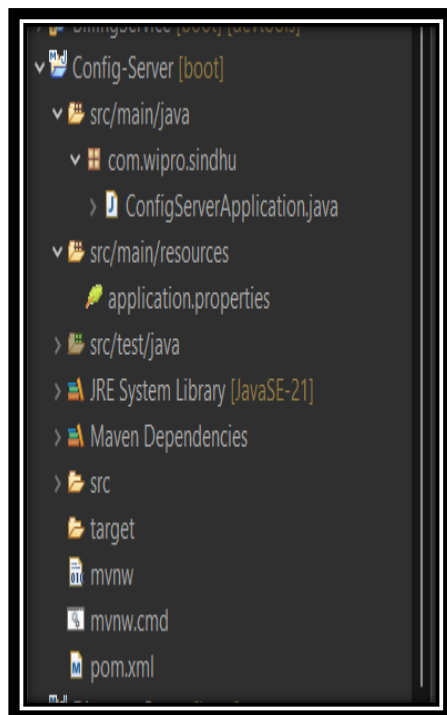
OpenFeign:

- Enables easy inter-service communication (e.g., Appointment_Service calling Doctor_Service).
- Reduces boilerplate REST Template code.

Config_Server:

- Centralized configuration management.
- Stores properties for all microservices.
- Supports dynamic refresh without restarting services.

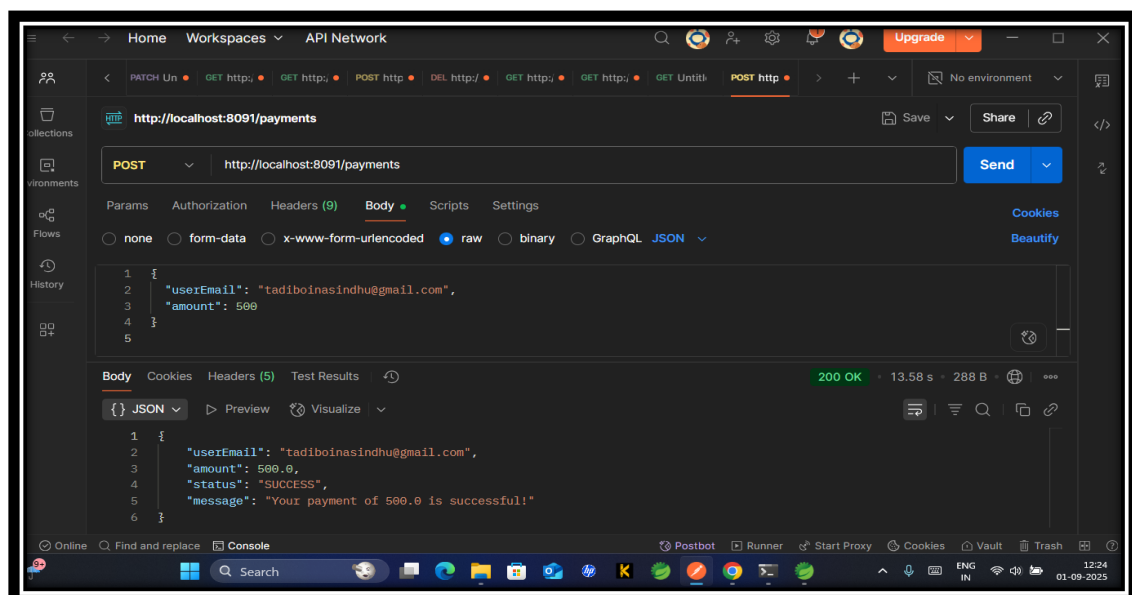
Confi_Server Project Structure:

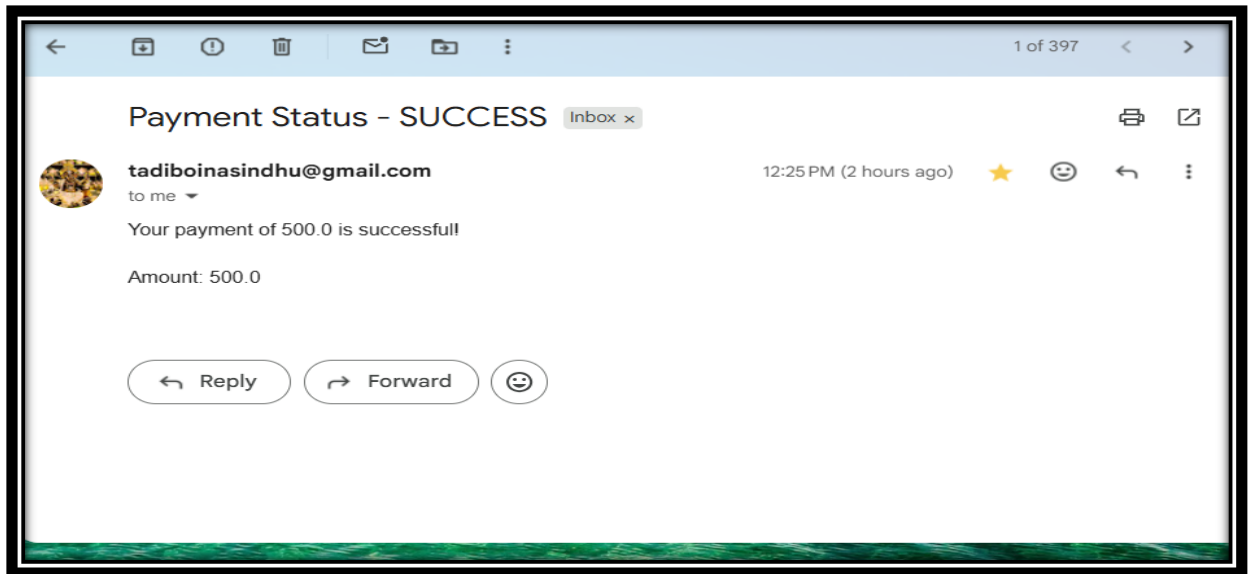


Kafka Integration :

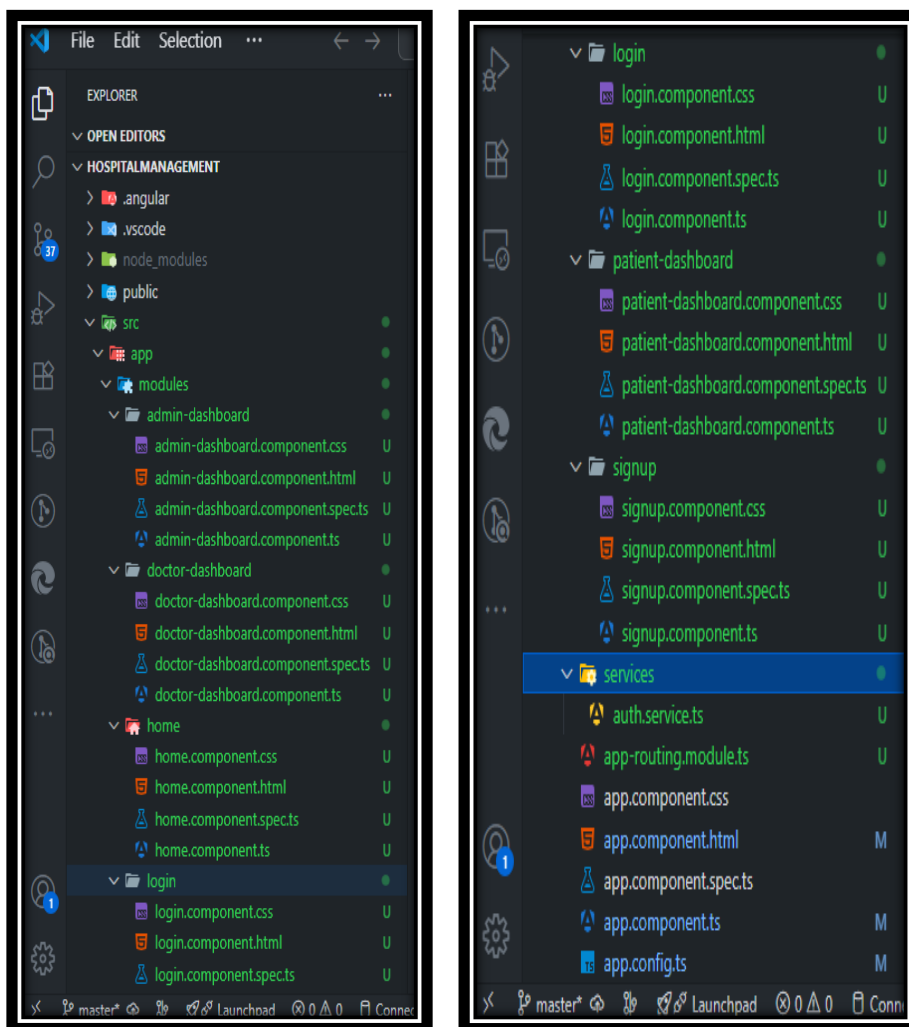
- Handles asynchronous communication between services.
- Sends real-time notifications (email/OTP) :
 - Appointment is booked/updated
 - Payment is successful/failed
 - Doctor prescription is uploaded

Kafka Output in Postman



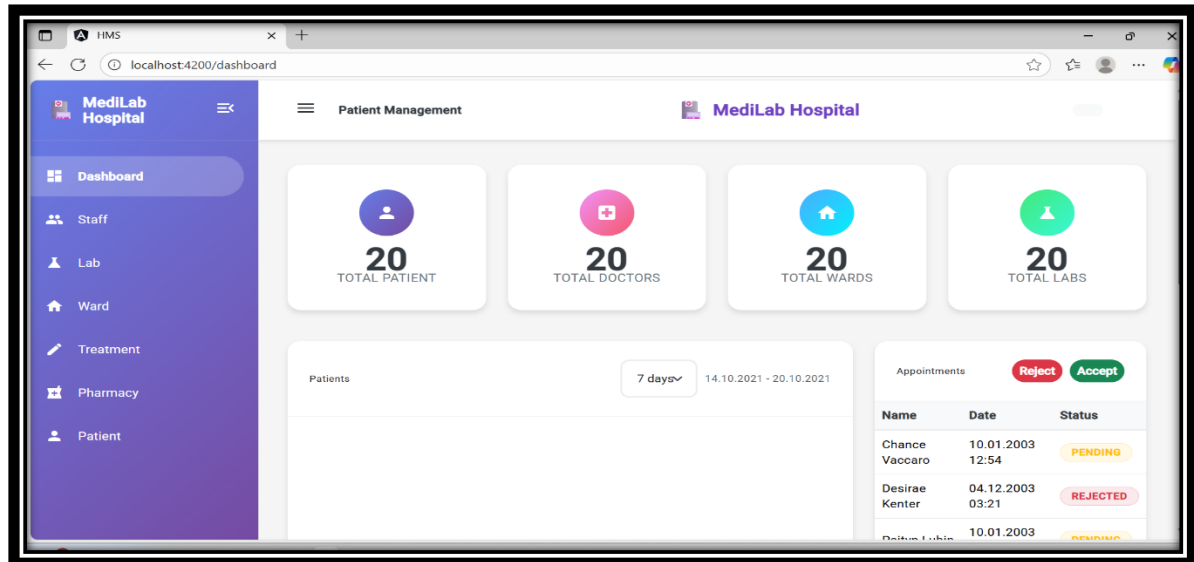


VS Code Frontend Project Structure:



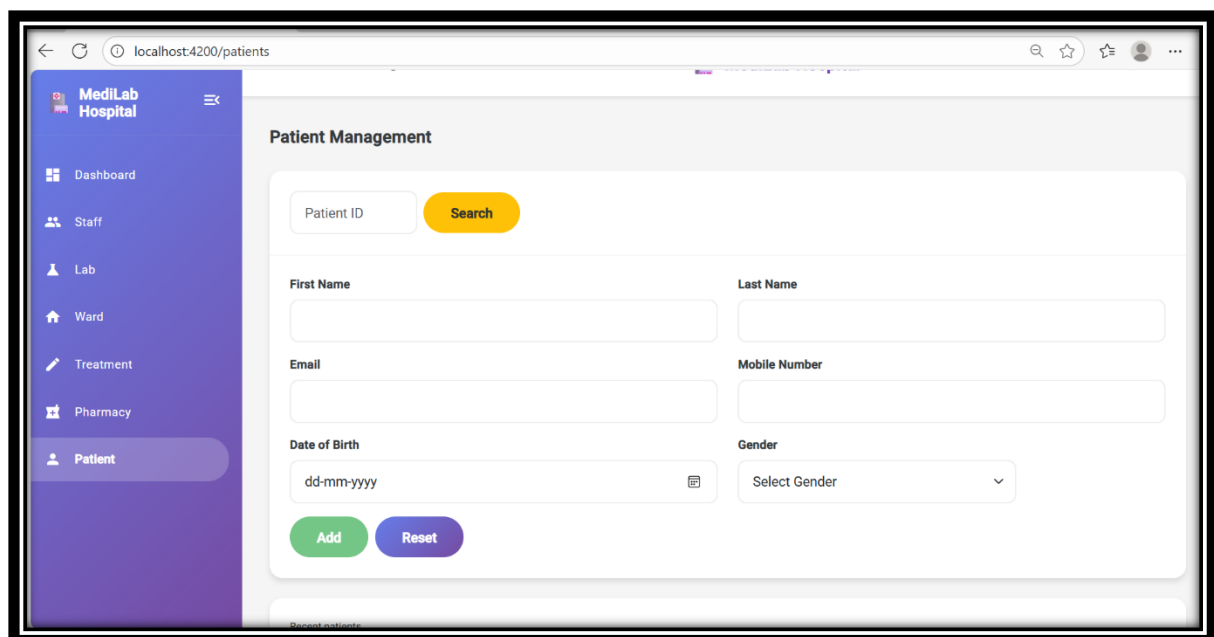
Frontend Output Hospital Management Dashboard Page:

The Hospital Management Dashboard acts as a centralized control panel for administrators, enabling them to monitor and manage the entire hospital ecosystem.



Patient Dashboard:

The Patient Dashboard allows patients to manage their personal info, appointments, payments, medical history, and notifications in one place



Doctor Dashboard:



The Doctor Dashboard helps doctors to manage their schedule, access patient records, prescribe treatments, and stay updated with notifications in a secure, user-friendly way.

The screenshot shows the 'Doctor Management' interface. On the left is a purple sidebar with the 'MediLab Hospital' logo and a menu containing 'Dashboard', 'Staff' (highlighted), 'Lab', 'Ward', 'Treatment', 'Pharmacy', and 'Patient'. The main content area has a title 'Doctor Management' and a search bar with 'Enter Doctor ID', a yellow 'Search' button, and a grey 'Reset' button. Below this is a white 'Add Doctor' form with fields for 'Name', 'Specialization', 'Experience (years)' (with '0' entered), 'Contact Number', and an 'Availability Status' dropdown. A green 'Add' button is at the bottom of the form.

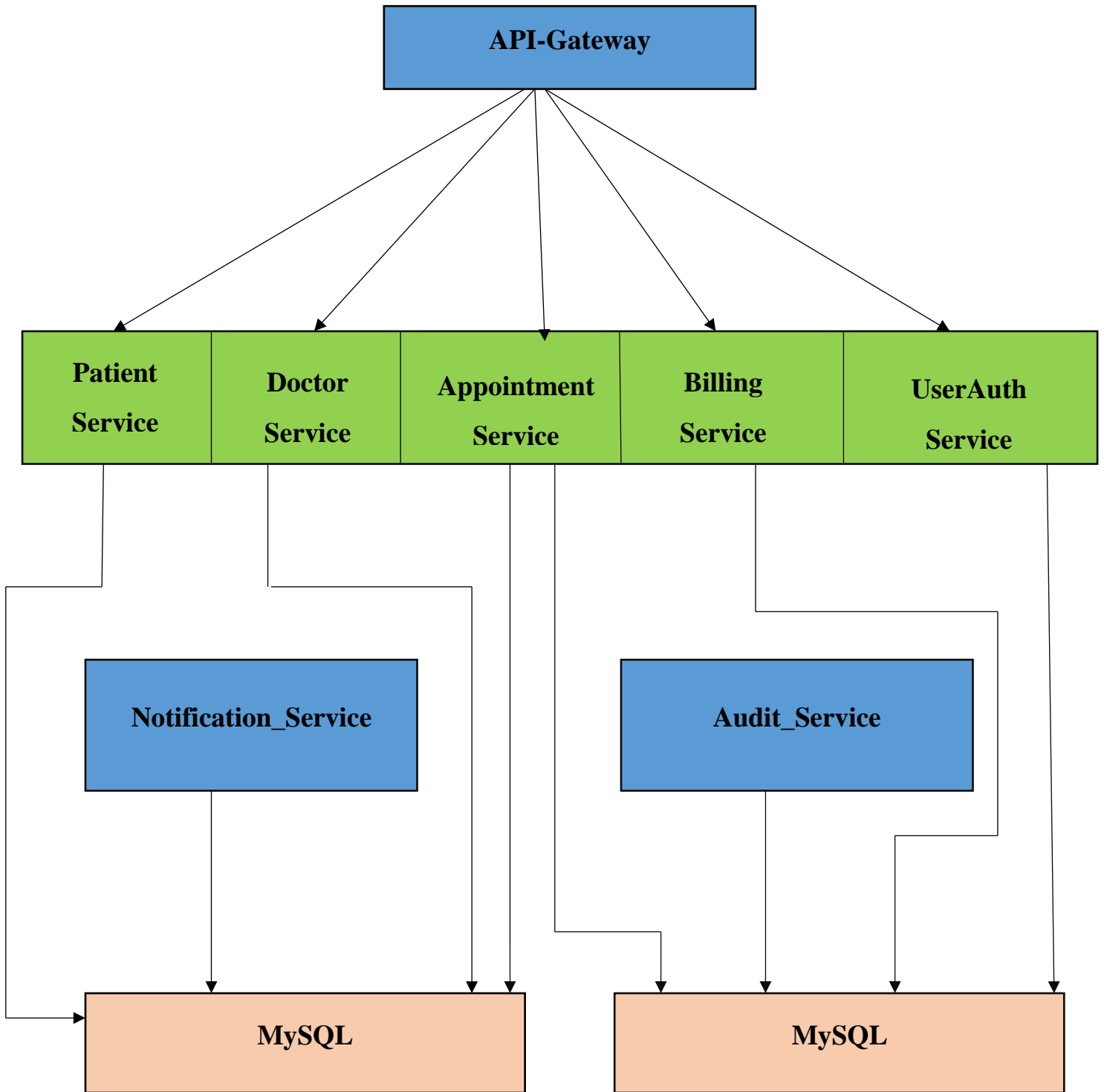
Billing Dashboard:

The Billing Dashboard is mainly used by the hospital's billing staff/admin (and patients for viewing their own bills).

The screenshot shows the 'Billing Management' interface. The left sidebar is identical to the previous dashboard. The main content area has a title 'Billing Management' and a 'Create Billing Record' form. This form includes fields for 'Patient ID', 'Appointment ID', 'Amount', and a 'Status' dropdown (currently showing 'PENDING'). There are green '+ Create' and purple 'Reset' buttons. Below the form is a 'Billing Records' table with the following data:

ID	Patient ID	Appointment ID	Amount	Status	Payment Date	Action
1	1	1	\$500.00	PAID	2025-08-16T02:33:11.72053	 

Hospital Management System Architecture



Advantage of HTTP Methods in REST APIs:

Using standard HTTP methods (GET, POST, PUT, PATCH, DELETE) makes the Hospital Management System RESTful, scalable, easy to maintain, and ensures clear communication between frontend and backend.

Advantages:

- **Reduces Paperwork** – Minimizes manual record-keeping and errors.
- **Improves Efficiency** – Speeds up patient registration, appointments, and billing.
- **Better Patient Care** – Provides quick access to medical history, prescriptions, and reports.
- **Data Security** – Ensures secure storage of medical records with role-based access.
- **Real-Time Communication** – Sends appointment and payment notifications instantly.

Conclusion:

The Hospital Management System successfully automates patient, doctor, appointment, and billing management, ensuring efficiency, data security, and better patient care. With microservices architecture, the system is scalable, reliable, and adaptable for future hospital needs.

