INFORMATICS INSTITUTE OF TECHNOLOGY

Computer Science

BSc (Hons) Computer Science

5COSC022W: Client-Server Architectures

Module Leader: Mr. Cassim Farook

Coursework Report

(2023/24)

Name: Mohamad Mahdi Sabry

UOW ID: w1954064

IIT ID: 20221253

Group: E

Table of Content

Table of Contents

List of Figures

Acknowledgement

I extend my heartfelt appreciation to the module leader, Mr. Cassim Farook, as well as to Mr. Kushan Bhareti, who served as both my lecturer and tutorial instructor for the duration of the semester. Their exceptional guidance and effective teaching methodologies have been instrumental in shaping my understanding of the subject matter. Mr. Bhareti's approachable demeanor and unwavering support have greatly facilitated our learning journey, making complex concepts seem more manageable. He consistently demonstrated a willingness to address any uncertainties we encountered, providing invaluable assistance every step of the way. Their dedication to our academic development has truly made a profound impact, for which I am sincerely grateful.

Introduction

The coursework specifications outline the criteria for creating a Health Management System API using Rest API architecture. This API will include data on patients, doctors, appointments, billing, and medical records.

The training utilizes Apache NetBeans IDE, Apache Tomcat Webserver, and Postman for request delivery and testing.
There are seven primary classes that declare instance variables and use OOP concepts like inheritance to improve code efficiency.
DAO classes perform CRUD actions on primary classes, which are then implemented in Resource classes.
The graphic below depicts a Web project with necessary packages and classes.

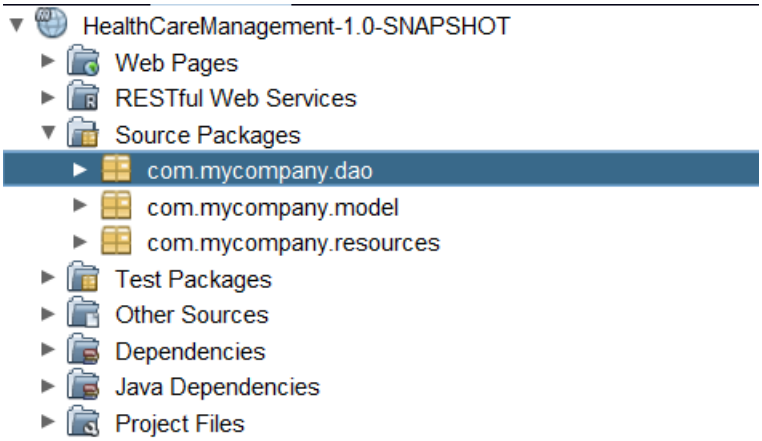


Figure 1 Setup

Project Overview

This project centers around the creation of a dynamic RESTful API leveraging JAX-RS, primarily aimed at establishing a robust healthcare management system API. Within this framework, seven pivotal entities are identified: Person, Patient, Doctor, Medical Records, Medical Bills, Appointments, and Prescriptions. The core objective is the comprehensive implementation of CRUD operations (GET, POST, PUT, and DELETE) for each entity. These functionalities facilitate seamless interaction with the API, fostering efficient management of healthcare data. By incorporating these features, the project endeavors to optimize healthcare processes, enhance data accessibility, and ensure the seamless integration of the API within healthcare systems.

```
<servlet-mapping>
    <servlet-name>HealthCareManagement</servlet-name>
    <url-pattern>/api/*</url-pattern>
</servlet-mapping>
```

Figure 2 URL

URL Example: - http://localhost:8080/HealthCareManagement/api/

System Design and Implementation

Entities

Person

The Person entity is the main entity based in the project which has variables which are extended to the classes like patient and doctor

Model implementation

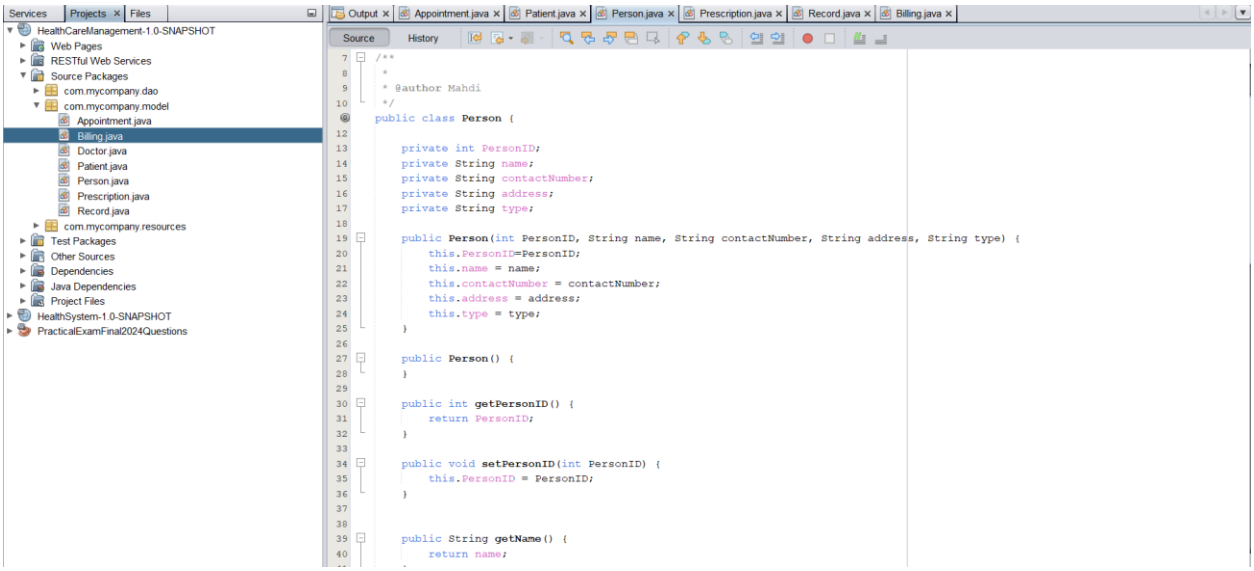


Figure 3 - Person Model

Dao implementation

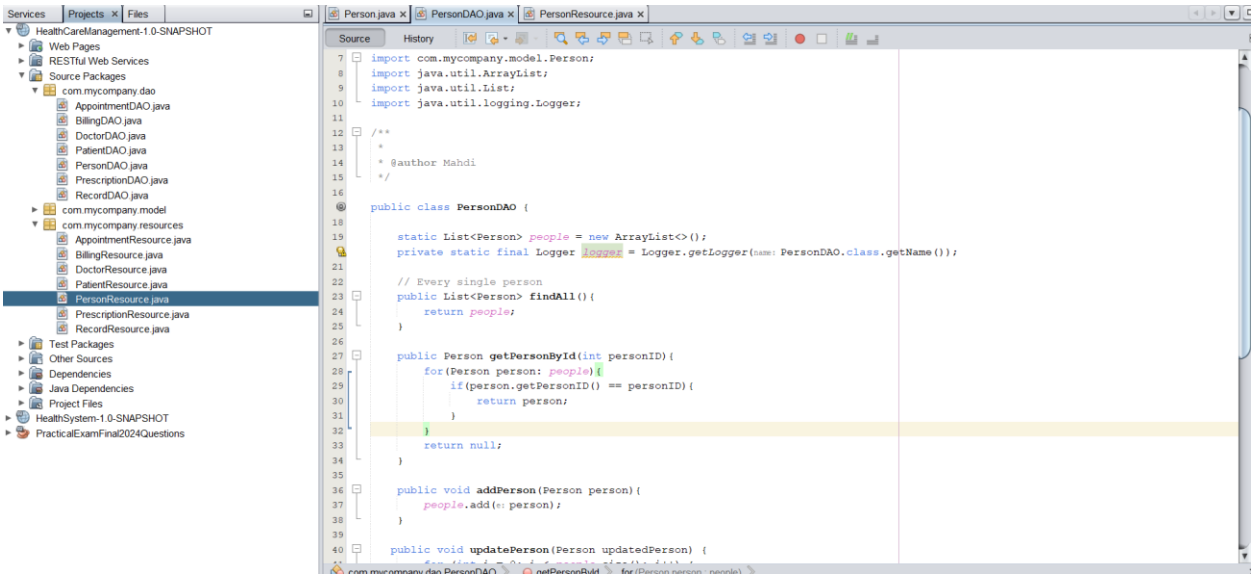
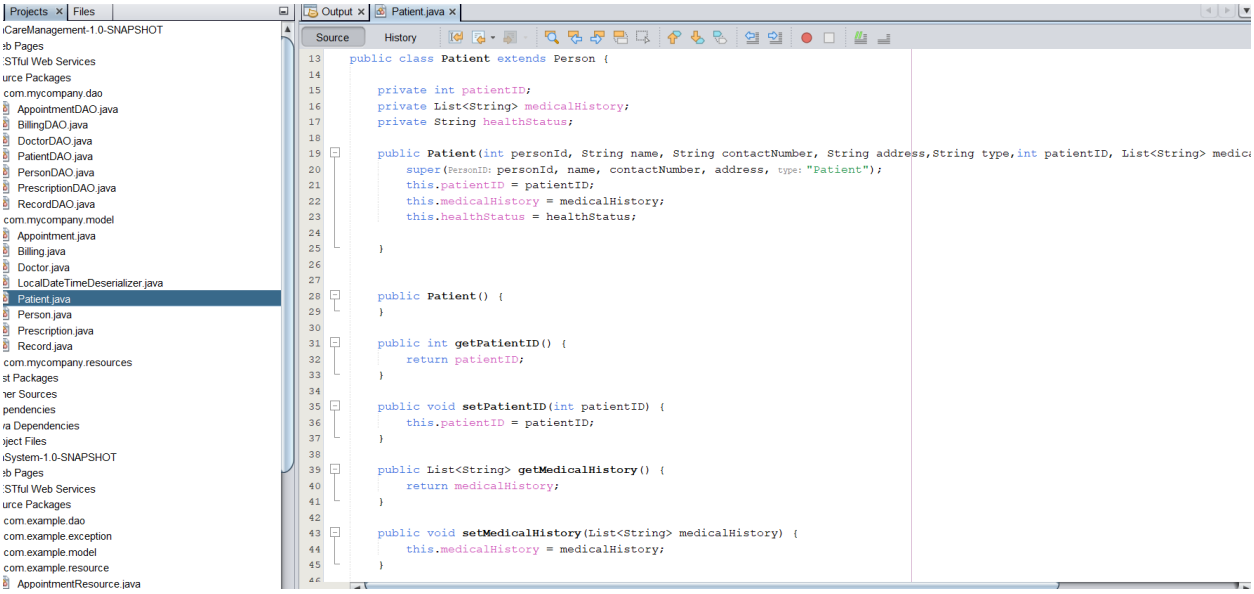


Figure 4 - Person DAO

Patient

The patient class inherits properties from persons and has its own instance variables to provide important patient information

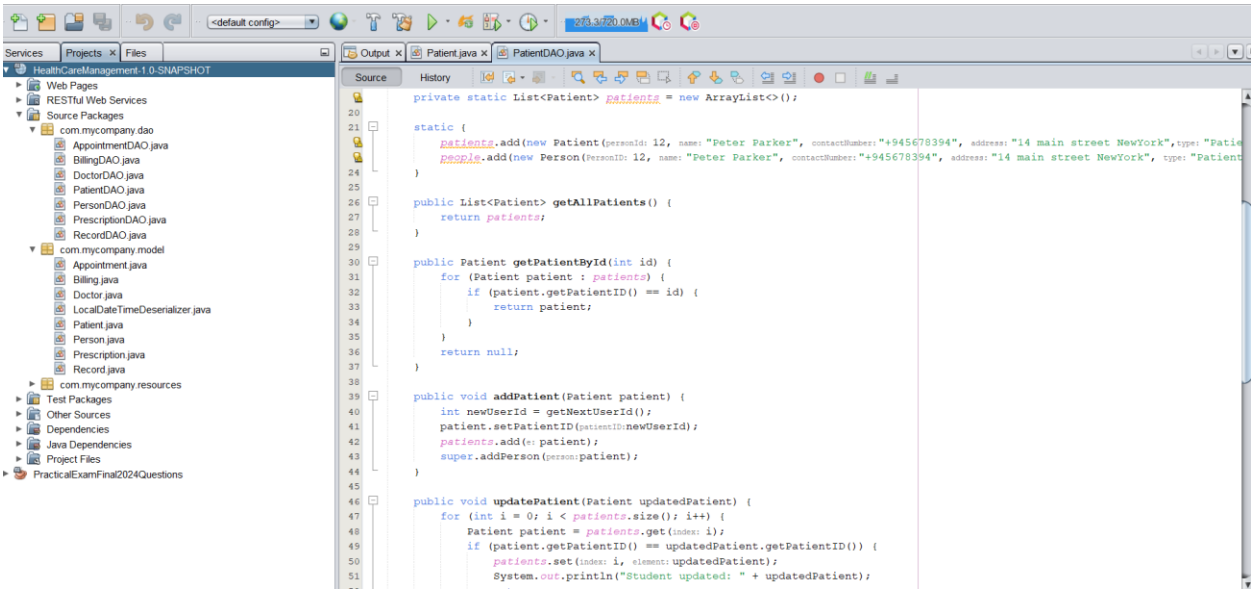
Model implementation



```java
public class Patient extends Person {

    private int patientID;
    private List<String> medicalHistory;
    private String healthStatus;

    public Patient(int personId, String name, String contactNumber, String address,String type,int patientID, List<String> medica
        super(PersonID: personId, name, contactNumber, address, type: "Patient");
        this.patientID = patientID;
        this.medicalHistory = medicalHistory;
        this.healthStatus = healthStatus;

    }


    public Patient() {
    }

    public int getPatientID() {
        return patientID;
    }

    public void setPatientID(int patientID) {
        this.patientID = patientID;
    }

    public List<String> getMedicalHistory() {
        return medicalHistory;
    }

    public void setMedicalHistory(List<String> medicalHistory) {
        this.medicalHistory = medicalHistory;
    }
```

Figure 5 - Patient Model

Dao implementation



```java
    private static List<Patient> patients = new ArrayList<>();

    static {
        patients.add(new Patient(personId: 12, name: "Peter Parker", contactNumber: "+945678394", address: "14 main street NewYork",type: "Patie
        people.add(new Person(PersonID: 12, name: "Peter Parker", contactNumber: "+945678394", address: "14 main street NewYork", type: "Patient
    }

    public List<Patient> getAllPatients() {
        return patients;
    }

    public Patient getPatientById(int id) {
        for (Patient patient : patients) {
            if (patient.getPatientID() == id) {
                return patient;
            }
        }
        return null;
    }

    public void addPatient(Patient patient) {
        int newUserId = getNextUserId();
        patient.setPatientID(patientID:newUserId);
        patients.add(e: patient);
        super.addPerson(person:patient);
    }

    public void updatePatient(Patient updatedPatient) {
        for (int i = 0; i < patients.size(); i++) {
            Patient patient = patients.get(index: i);
            if (patient.getPatientID() == updatedPatient.getPatientID()) {
                patients.set(index: i, element: updatedPatient);
                System.out.println("Student updated: " + updatedPatient);
```

Figure 6 - Patient DAO

Doctor

The Doctor class inherits properties from persons and has its own instance variables to provide important doctors information.

Model implementation

```java
/**
 *
 * @author Mahdi
 */
public class Doctor extends Person {

    private int DoctorID;
    private String specialization;

    public Doctor(int personId, String name, String contactNumber, String address, String type,String specialization,int DoctorI
    {
        super(PersonID: personId,name, contactNumber, address, type: "Doctor");
        this.specialization=specialization;
        this.DoctorID= DoctorID;
    }

    public Doctor() {
    }

    public int getDoctorID() {
        return DoctorID;
    }

    public void setDoctorID(int DoctorID) {
        this.DoctorID = DoctorID;
    }

    public String getSpecialization() {
        return specialization;
    }

    public void setSpecialization(String specialization) {
        this.specialization = specialization;
```

Figure 7 - Doctor Model

Dao implementation

```java
    private static List<Doctor> doctors = new ArrayList<>();

    static{
        doctors.add(new Doctor(personId: 111, name: "5 rupees doctor", contactNumber: "+987654345", address: "43 Chennai India",type: "Doctor",
        doctors.add(new Doctor(personId: 233, name: "Dr. Dolittle", contactNumber: "+943343242", address: "121 los angeles California",type: "Dc
        people.add(new Person(PersonID: 111, name: "5 rupees doctor", contactNumber: "+987654345", address: "43 Chennai India", type: "Doctor"))
        people.add(new Person(PersonID: 233, name: "Dr. Dolittle", contactNumber: "+943343242", address: "121 los angeles California", type: "Dc
    }

    public List<Doctor> getAllDoctors(){
        return doctors;
    }

    public Doctor getDoctorById(int id) {
        for (Doctor doctor : doctors) {
            if (doctor.getDoctorID() == id) {
                return doctor;
            }
        }
        return null;
    }

    public void addDoctor(Doctor doctor) {
        doctors.add(e: doctor);
        super.addPerson(person:doctor);
    }

    public void updateDoctor(Doctor updatedDoctor) {
        for (int i = 0; i < doctors.size(); i++) {
            Doctor doctor = doctors.get(index: i);
            if (doctor.getDoctorID() == updatedDoctor.getDoctorID()) {
                doctors.set(index: i, element: updatedDoctor);
                return;
```

Figure 8 - Doctor DAO

Appointment

Model implementation

```java
/**
 *
 * @author Mahdi
 */
public class Appointment {

    private int appointmentId;
    private Patient patient;
    private Doctor doctor;
    private String dateAndTime;

    public Appointment(int appointmentId, Patient patient, Doctor doctor, String dateAndTime) {
        this.appointmentId = appointmentId;
        this.patient = patient;
        this.doctor = doctor;
        this.dateAndTime = dateAndTime;
    }

    public Appointment(){
    }

    public int getAppointmentId() {
        return appointmentId;
    }

    public void setAppointmentId(int appointmentId) {
        this.appointmentId = appointmentId;
    }

    public Patient getPatient() {
        return patient;
    }

    public void setPatient(Patient patient) {
```

Figure 9 - Appointment Model

Dao implementation

```java
public class AppointmentDAO {

    private static final Logger logger = Logger.getLogger(name: AppointmentDAO.class.getName());
    static List<Appointment> appointments = new ArrayList<>();

    static {
        Date date = new Date();
        SimpleDateFormat dateFormat = new SimpleDateFormat(pattern: "yyyy-MM-dd");
        String formattedDate = dateFormat.format(date);
        PatientDAO patientDAO = new PatientDAO();
        DoctorDAO doctorDAO = new DoctorDAO();

        appointments.add(new Appointment(appointmentId: 1, patient: patientDAO.getPatientById(id: 143), doctor:doctorDAO.getDoctorById(id: 76
    }

    public List<Appointment> getAllAppointments() {
        return appointments;
    }

    public Appointment getAppointmentById(int appointmentId) {
        for (Appointment appointment : appointments) {
            if (appointment.getAppointmentId() == appointmentId) {
                return appointment;
            }
        }
        return null;
    }

    public void addAppointments(Appointment appointment) {
        int newAppointmentId = getAppointmentId();
        appointment.setAppointmentId(appointmentId: newAppointmentId);
```

Figure 10 - Appointment DAO

Prescription

Model implementation

```
company.dao
intmentDAO.java
gDAO.java
orDAO.java
ntDAO.java
onDAO.java
criptionDAO.java
rdDAO.java
company.model
intment.java
g.java
or.java
lDateTimeDeserializer.java
nt.java
on.java
cription.java
rd.java
company.resources
ges
ces
ies
ndencies
s
iFinal2024Questions
```

```java
    private int priscriptionId;
    private Patient patient;
    private String medication;
    private String dosage;
    private String instructions;
    private String duration;

    public Prescription(int priscriptionId, Patient patient, String medication, String dosage, String instructions, String dura
        this.priscriptionId = priscriptionId;
        this.patient = patient;
        this.medication = medication;
        this.dosage = dosage;
        this.instructions = instructions;
        this.duration = duration;
    }

    public Prescription(){}


    public int getPriscriptionId() {
        return priscriptionId;
    }

    public void setPriscriptionId(int priscriptionId) {
        this.priscriptionId = priscriptionId;
    }

    public Patient getPatient() {
        return patient;
```

Figure 11 - Prescription Model

Dao implementation

```java
 */
public class PrescriptionDAO {

    private static List<Prescription> prescriptions = new ArrayList<>();

    static{
        Patient patient1 = new Patient(personId: 12, name: "Peter Parker", contactNumber: "+945678394", address: "14 main street NewYork", type:

        prescriptions.add(new Prescription(priscriptionId: 001, patient: patient1, medication: "MedicineX", dosage: "2 Tablets", instructions: "Before
    }

    public List<Prescription> getAllPrescriptions(){
        return prescriptions;
    }

    public Prescription getPrescriptionById(int prescriptionId){
        for (Prescription prescription : prescriptions) {
            if (prescription.getPriscriptionId()== prescriptionId) {
                return prescription;
            }
        }
        return null;
    }

    // Add prescription
    public void addPrescription(Prescription prescription){
        int newPrescriptionId = getPrescriptionId();
        prescription.setPriscriptionId(priscriptionId: newPrescriptionId);
        prescriptions.add(e: prescription);
    }

    public void updatePrescription(Prescription updatedPrescription){
        for(int i = 0; i < prescriptions.size(); i++){
```

Figure 12 - Prescription DAO

Medical Record

Model implementation

```java
public class Record {

    private int recordID;
    private Patient patient;
    private List<String> diagnoses;
    private List<String> treatment;

    public Record(int recordID, Patient patient, List<String> diagnoses, List<String> treatment) {
        this.recordID = recordID;
        this.patient = patient;
        this.diagnoses = diagnoses;
        this.treatment = treatment;
    }

    public Record(){}

    public int getRecordID() {
        return recordID;
    }

    public void setRecordID(int recordID) {
        this.recordID = recordID;
    }

    public Patient getPatient() {
        return patient;
    }

    public void setPatient(Patient patient) {
        this.patient = patient;
    }
```

Figure 13 - Medical Record Model

Dao implementation

```java
    private static final Logger logger = Logger.getLogger(name: RecordDAO.class.getName());
    private static List<Record> records = new ArrayList<>();

    static{
        PatientDAO patientDAO = new PatientDAO();

        records.add(new Record(recordID: 1,patient: patientDAO.getPatientById(id: 143), diagnoses:Arrays.asList(a: "Spidersense", a: "Lost 1

    }
    public List<Record> getAllRecords(){
        logger.info(msg: "Getting Every Single medical records");
        return records;
    }

    public Record getRecordById(int id){
        logger.info(msg: "Get medical record by id");
        for(Record record : records){
            if (record.getRecordID() == id){
                return record;
            }
        }
        return null;
    }

    public void addRecord(Record record){
        records.add(e: record);
        logger.info(msg: "Added record successfully");
    }

    public void updateRecord(Record updatedRecord){
        for(int i = 0; i < records.size();i++){
            Record record = records.get(index: i);
```

Figure 14 - Medical Record DAO

Billing

Model implementation

```java
public class Billing {

    private Patient patient;
    private int billId;
    private double outStandingBalance;
    private double totalPayment;

    public Billing(Patient patient, int billId, double outStandingBalance, double totalPayment) {
        this.patient = patient;
        this.billId = billId;
        this.outStandingBalance = outStandingBalance;
        this.totalPayment = totalPayment;
    }

    public Billing(){}

    public Patient getPatient() {
        return patient;
    }

    public void setPatient(Patient patient) {
        this.patient = patient;
    }


    public int getBillId() {
        return billId;
    }

    public void setBillId(int buildId) {
        this.billId = buildId;
    }
```

Figure 15 - Billing Model

Dao implementation

```java
public class BillingDAO {

    private static final Logger logger = Logger.getLogger(name: BillingDAO.class.getName());
    // List to store Billing objects
    private static List<Billing> billings = new ArrayList<>();

    static {
        // Create PersonDAO object
        PatientDAO patientDAO = new PatientDAO();

        billings.add(new Billing(patient: patientDAO.getPatientById(id: 143), billId: 001, outStandingBalance: 10000, totalPayment: 50000));
    }

    // Method to get all Bills
    public List<Billing> getAllBills() {
        logger.log(level: Level.INFO, msg: "Getting all the bills");
        return billings;   // Return the list of bills
    }

    // Method to get a Bill data using Bill No
    public Billing getBillById(int id) {

        // Iterate through the bill list
        logger.log(level: Level.INFO, "Getting a bill by ID: " + id);
        for (Billing billing : billings) {
            if (billing.getBillId() == id) {
                return billing;    // return the relevant bill
            }
        }
        logger.log(level: Level.WARNING, "Bill is not found with ID: " + id);
        return null;

    }
```

Figure 16 - Billing DAO

Crud Implementation

RESTful Resources Implementation

The request methods Post, Get, Put, and Delete are referred to as the "crud operations" in the system since that is how the request-response model architecture is designed. Although the methods are defined in the dao classes, the resource classes carry out the crud activities.

Annotations offer valuable insight into this process. The path argument in resource classes provides the route and the url specifying for which element the crude action has to be done. Postman also provides the text answer because text was added to response codes during construction.
Additional web browser view outputs are available in the appendix section.

Person

GET

```java
@Path("/person")
public class PersonResource {

    private PersonDAO personDAO = new PersonDAO();
    private static final Logger logger = Logger.getLogger(name: PersonResource.class.getName());


    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Person> findAll() {
        try {
            return personDAO.findAll();
        } catch (Exception e) {
            // Handle the exception
            throw new WebApplicationException(message: "Error occurred while retrieving all persons.", cause: e);
        }
    }

```

Figure 17 - Get ALL

```java
@GET
@Path("/{PersonID}")
@Produces(MediaType.APPLICATION_JSON)
public Person getPersonById(@PathParam("PersonID") int PersonID) {
    try {
        Person person = personDAO.getPersonById(personID: PersonID);
        if (person != null) {
            return person;
        } else {
            throw new NotFoundException("Person with ID " + PersonID + " not found.");
        }
    } catch (Exception e) {
        // Handle the exception
        throw new WebApplicationException("Error occurred while retrieving person with ID " + PersonID, cause: e);
    }
}
```

Figure 18 - Get person by ID

POST

Post method for person has been omitted because there are two subclasses called Patient and Doctors. Creating is not allowed for person

PUT

```java
    @PUT
    @Path("/{PersonID}")
    @Consumes(MediaType.APPLICATION_JSON)
    public void updatePerson(@PathParam("PersonID") int PersonID, Person updatedPerson) {
        try {
            Person existingPatient = personDAO.getPersonById(personID: PersonID);

            if (existingPatient != null) {
                updatedPerson.setPersonID(PersonID);
                personDAO.updatePerson(updatedPerson);
            } else {
                throw new NotFoundException("Person with ID " + PersonID + " not found.");
            }
        } catch (Exception e) {
            // Handle the exception
            throw new WebApplicationException("Error occurred while updating person with ID " + PersonID, cause: e);
        }
    }
```

Figure 19 - Update Person

DELETE

```
    @DELETE
    @Path("/{PersonID}")
    public void deletePerson(@PathParam("PersonID") int PersonID) {
        try {
            personDAO.deletePerson(id: PersonID);
        } catch (Exception e) {
            // Handle the exception
            throw new WebApplicationException("Error occurred while deleting person with ID " + PersonID, cause: e);
        }
    }
}
```

Figure 20 - Delete person

Patients

GET

```
    private PatientDAO patientdao = new PatientDAO();
    private static final Logger logger = Logger.getLogger(name: PatientResource.class.getName());


    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Patient> getAllPatients() {
        return patientdao.getAllPatients();
    }
```

Figure 21 - Get Every Patient

```
    @GET
    @Path("/{patientID}")
    @Produces(MediaType.APPLICATION_JSON)
    public Patient getStudentById(@PathParam("patientID") int patientID) {
        return patientdao.getPatientById(id: patientID);
    }
```

Figure 22 - Get PatientByID

POST

```
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public void addPatient(Patient patient) {
        patientdao.addPatient(patient);
        logger.info(msg: "Patient added");
    }
```

Figure 23 - Create Patient

PUT

```
    @PUT
    @Path("/{patientID}")
    @Consumes(MediaType.APPLICATION_JSON)
    public void updatePatient(@PathParam("patientID") int patientID, Patient updatedPatient) {
        Patient existingPatient = patientdao.getPatientById(id: patientID);

        if (existingPatient != null) {
            updatedPatient.setPatientID(patientID);
            patientdao.updatePatient(updatedPatient);
        }
    }
```

Figure 24 - Update Patient

DELETE

```
    @DELETE
    @Path("/{patientID}")
    public void deletePatient(@PathParam("patientID") int patientID) {
        patientdao.deletePatient(id: patientID);
    }

}
```

Figure 25 - Delete Patient

Doctors

GET

```
@Path("/doctors")
public class DoctorResource {
    private DoctorDAO doctorDao = new DoctorDAO();

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Doctor> getAllDoctors() {
        return doctorDao.getAllDoctors();
    }

    @GET
    @Path("/{DoctorID}")
    @Produces(MediaType.APPLICATION_JSON)
    public Doctor getDoctorById(@PathParam("DoctorID") int DoctorID) {
        return doctorDao.getDoctorById(id: DoctorID);
    }
```

Figure 26 - Get Doctors

POST

```
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public void addDoctor(Doctor doctor) {
        doctorDao.addDoctor(doctor);
    }
```

Figure 27 - create Doctor

PUT

```
@PUT
@Path("/{DoctorID}")
@Consumes(MediaType.APPLICATION_JSON)
public void updateDoctor(@PathParam("DoctorID") int DoctorID, Doctor updatedDoctor) {
    Doctor existingDoctor = doctorDao.getDoctorById(id: DoctorID);

    if (existingDoctor != null) {
        updatedDoctor.setDoctorID(DoctorID);
        doctorDao.updateDoctor(updatedDoctor);
    }
}
```

Figure 28 - Update Doctor

DELETE

```
@DELETE
@Path("/{DoctorID}")
public void deleteDoctor(@PathParam("DoctorID") int DoctorID) {
    doctorDao.deleteDoctor(id: DoctorID);
}
}
```

Figure 29 - Delete Doctor

Appointments

GET

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public List<Appointment> getAllAppointments() {
    return appointmentDAO.getAllAppointments();
}
```

```
@GET
@Path("/{appointmentId}")
@Produces(MediaType.APPLICATION_JSON)
public Response getAppointmentById(@PathParam("appointmentId") int appointmentId) {
    // Retrieve the appointment by ID from your DAO or service layer
    Appointment appointment = appointmentDAO.getAppointmentById(appointmentId);

    // Check if the appointment exists
    if (appointment == null) {
        return Response.status(status:Response.Status.NOT_FOUND).entity("Appointment" + appointmentId).build();
    }
    // Return the response
    return Response.ok().entity(entity:appointment).build();
}
```

Figure 30 - get appointment

POST

```
@POST
@Consumes(MediaType.APPLICATION_JSON)
public void addAppointments(Appointment appointment) {
    appointmentDAO.addAppointments(appointment);
}
```

Figure 31 - Create appointment

PUT

```
    // Method to update an existing Appointment
    @PUT
    @Path("/{appointmentId}")
    @Consumes(MediaType.APPLICATION_JSON)
    public Response updateAppointment(@PathParam("appointmentId") int appointmentId, Appointment updatedAppointment) {
        try {
            Appointment existingAppointment = appointmentDAO.getAppointmentById(appointmentId);
            if (existingAppointment == null) {
                throw new NotFoundException("Appointment not found with ID: " + appointmentId);
            }
            updatedAppointment.setAppointmentId(appointmentId);
            appointmentDAO.updateAppointment(updatedAppointment);   // update the appointment
            // Return the success response
            return Response.status(status:Response.Status.OK).entity(entity:"Appointment updated successfully").build();
        } catch (Exception e) {
            // Throw WebApplicationException
            throw new WebApplicationException(message: "Error in updating the appointment", cause: e, status:500);
        }
    }
```

Figure 32 - Update appointment

DELETE

```
    // Method to delete an Appointment
    @DELETE
    @Path("/{appointmentId}")
    public Response deleteAppointment(@PathParam("appointmentId") int appointmentId) {
        try {    // Delete the appointment
            appointmentDAO.deleteAppointment(id: appointmentId);
            // Return the success response
            return Response.status(status:Response.Status.OK).entity(entity:"Appointment deleted successfully").build();
        } catch (Exception e) {
            // Throw WebApplicationException
            throw new WebApplicationException(message: "Error in deleting the appointment with ID", cause: e, status:500);
        }
    }
}
```

Figure 33 - Delete Appointment

Medical Records

GET

```
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Record> getAllRecords() {
        try {
            logger.info(msg:"Getting All medical records");
            return recordDAO.getAllRecords();
        } catch (Exception e) {
            logger.info(msg:"Error in getting record");
            throw new WebApplicationException(message: "Error in getting all the medical records", cause: e, status:500);

        }

    }

    @GET
    @Path("/{recordID}")
    @Produces(MediaType.APPLICATION_JSON)
    public Record getRecordById(@PathParam("recordID") int recordID) {
        try {
            logger.log(level: Level.INFO, msg:"Getting medical record for id number{0}", param1:recordID);
            return recordDAO.getRecordById(id: recordID);
        } catch (Exception e) {
            logger.log(level: Level.SEVERE, "Error in getting the medical record by ID: " + recordID + ", " + e.getMessage(), thrown
            // Throw WebApplicationException
            throw new WebApplicationException(message: "Error in getting the medical record by ID", cause: e, status:500);
        }
    }
```

Figure 34- Get Medical record

POST

```java
@POST
@Consumes(MediaType.APPLICATION_JSON)
public void addRecord(Record record) {
    try {
        recordDAO.addRecord(record);
        logger.info(msg: "Record created successfully");
    } catch (Exception e) {
        throw new RuntimeException(message: "Error adding medical record", cause: e);
    }
}
```

Figure 35 - create medical record

PUT

```java
@PUT
@Path("/{recordID}")
@Consumes(MediaType.APPLICATION_JSON)
public void updateRecord(Record updatedRecord, @PathParam("recordID") int recordID) {
    try {
        // Set the patient ID in the updated record
        updatedRecord.getPatient().setPatientID(patientID:recordID);
        // Call the DAO method to update the medical record
        recordDAO.updateRecord(updatedRecord);
        logger.info(msg: "Record updated successfully");
    } catch (Exception e) {
        // Handle the case where the medical record for the specified patient ID is not found
        throw e;
    }
}
```

Figure 36 - update medical record

DELETE

```java
@DELETE
@Path("/{recordID}")
public void deleteMedicalRecord(@PathParam("recordID") int recordID) {
    try {
        recordDAO.deleteRecord(id: recordID);
    } catch (Exception e) {
        throw e;
    }
}
}
```

Figure 37 - delete medical record

Medical Bills

GET

```java
@GET
@Produces(MediaType.APPLICATION_JSON)
public List<Billing> getAllBillings() {
    try {
        logger.log(level: Level.INFO, msg: "Getting all bills");
        return billingDAO.getAllBills();   // Return all bills
    } catch (Exception e) {
        logger.log(level: Level.SEVERE, "Error in getting all the bills: " + e.getMessage(), thrown: e);
        // Throw WebApplicationException
        throw new WebApplicationException(message: "Error in getting all the bills", cause: e, status: 500);
    }
}

@GET
@Path("/{billId}")
@Produces(MediaType.APPLICATION_JSON)
public Billing getBillingRecordByBillNo(@PathParam("billId") int billId) {
    try {
        return billingDAO.getBillById(id: billId);
    } catch (Exception e) {
        throw e;
    }
}
```

Figure 38 - get medical bills

POST

```java
@POST
@Consumes(MediaType.APPLICATION_JSON)
public void addBill(Billing billing) {
    try {
        billingDAO.addBill(billing);
    } catch (Exception e) {
        throw new RuntimeException(message: "Error adding billing", cause: e);
    }
}
```

Figure 39 - create medical bills

PUT

```java
@PUT
@Path("/{billId}")
@Consumes(MediaType.APPLICATION_JSON)
public void updateBill(@PathParam("billId") int billId, Billing updatedBill) {
    try {
        updatedBill.setBillId(buildId: billId); // Set the bill number from the path parameter
        billingDAO.updateBill(updatedBilling: updatedBill);
    } catch (Exception e) {
        throw e;
    }
}
```

Figure 40 - update bills

DELETE

```
@DELETE
@Path("/{billId}")
public void deleteBilling(@PathParam("billId") int billId) {
    try {
        billingDAO.deleteBill(id: billId);
    } catch (Exception e) {
        throw e;
    }
}
```

Figure 41 - delete bills

Prescriptions

GET

```
private PrescriptionDAO prescriptionDAO = new PrescriptionDAO();

@GET
@Produces(MediaType.APPLICATION_JSON)
public List<Prescription> getAllPrescriptions() {
    return prescriptionDAO.getAllPrescriptions();
}

@GET
@Path("{priscriptionId}")
@Produces(MediaType.APPLICATION_JSON)
public Prescription getPrescriptionById(@PathParam("priscriptionId") int priscriptionId) {
    return prescriptionDAO.getPrescriptionById(prescriptionId: priscriptionId);
}
```

Figure 42 - get Prescriptions

POST

```
@POST
@Consumes(MediaType.APPLICATION_JSON)
public void addPrescription(Prescription prescription) {
    prescriptionDAO.addPrescription(prescription);
}
```

Figure 43 - create prescriptions

PUT

```
@PUT
@Path("/{priscriptionId}")
@Consumes(MediaType.APPLICATION_JSON)
public void updatePrescription(@PathParam("prescriptionId") int priscriptionId, Prescription prescription) {
    // Ensuring the ID in the URL matches the ID in the object
    prescription.setPriscriptionId(priscriptionId);
    prescriptionDAO.updatePrescription(updatedPrescription:prescription);
}
```

Figure 44 - update prescriptions

DELETE

```
@DELETE
@Path("/{priscriptionId}")
public void deletePrescription(@PathParam("priscriptionId") int prescriptionId) {
    prescriptionDAO.deletePrescription(id: prescriptionId);
}
```

Figure 45 - delete prescriptions

Error Handling and Testing

Test Cases and Scenarios

Logger Classes

We can better grasp the request's state by using the logger class to identify the logs when the request is being processed.
To ensure clarity and know exactly what is being processed at any given time, a tag can be developed and used.

```java
private static final Logger logger = Logger.getLogger(name: BillingResource.class.getName());
private BillingDAO billingDAO = new BillingDAO();

@GET
@Produces(MediaType.APPLICATION_JSON)
public List<Billing> getAllBillings() {
    try {
        logger.log(level: Level.INFO, msg: "Getting all bills");
        return billingDAO.getAllBills();   // Return all bills
    } catch (Exception e) {
        logger.log(level: Level.SEVERE, "Error in getting all the bills: " + e.getMessage(), thrown: e);
        // Throw WebApplicationException
        throw new WebApplicationException(message: "Error in getting all the bills", cause: e, status: 500);
    }
}

@GET
@Path("/bills/{billId}")
@Produces(MediaType.APPLICATION_JSON)
public Billing getBillingRecordByBillNo(@PathParam("billId") int billId) {
    try {
        return billingDAO.getBillById(id: billId);
    } catch (Exception e) {
        throw e;
    }
}

@POST
@Consumes(MediaType.APPLICATION_JSON)
public void addBill(Billing billing) {
    try {
        billingDAO.addBill(billing);
    } catch (Exception e) {
        throw new RuntimeException(message: "Error adding billing", cause: e);
```

Figure 46 - logger

```java
private static final Logger logger = Logger.getLogger(name: RecordDAO.class.getName());
private static List<Record> records = new ArrayList<>();

static{
    PatientDAO patientDAO = new PatientDAO();

    records.add(new Record(recordID: 1,patient: patientDAO.getPatientById(id: 143), diagnoses:Arrays.asList(a: "Spidersense", a: "Lost his

}
public List<Record> getAllRecords(){
    logger.info(msg: "Getting Every Single medical records");
    return records;
}

public Record getRecordById(int id){
    logger.info(msg: "Get medical record by id");
    for(Record record : records){
        if (record.getRecordID() == id){
            return record;
        }
    }
    return null;
}

public void addRecord(Record record){
    records.add(e: record);
    logger.info(msg: "Added record successfully");
}
```

Error Handling

Encode handling of errors, exceptions, etc. is used for error handling. As a Web server with a request-response model architecture, the system provides status codes such as 500 internal server faults or 404 not found in response to requests. In addition to this, we frequently view exceptions when coding to overcome any faults and handle exceptions.

```java
    private static final Logger logger = Logger.getLogger(name: BillingResource.class.getName());
    private BillingDAO billingDAO = new BillingDAO();

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Billing> getAllBillings() {
        try {
            logger.log(level: Level.INFO, msg: "Getting all bills");
            return billingDAO.getAllBills();    // Return all bills
        } catch (Exception e) {
            logger.log(level: Level.SEVERE, "Error in getting all the bills: " + e.getMessage(), thrown: e);
            // Throw WebApplicationException
            throw new WebApplicationException(message: "Error in getting all the bills", cause: e, status: 500);
        }
    }

    @GET
    @Path("/bills/{billId}")
    @Produces(MediaType.APPLICATION_JSON)
    public Billing getBillingRecordByBillNo(@PathParam("billId") int billId) {
        try {
            return billingDAO.getBillById(id: billId);
        } catch (Exception e) {
            throw e;
        }
    }

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public void addBill(Billing billing) {
        try {
            billingDAO.addBill(billing);
        } catch (Exception e) {
            throw new RuntimeException(message: "Error adding billing", cause: e);
```

```java
            return Response.status(status:Response.Status.INTERNAL_SERVER_ERROR)
                    .entity("Error retrieving prescriptions: " + e.getMessage()).build();
        }
    }

    @GET
    @Path("/{prescriptionId}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response getPrescriptionById(@PathParam("prescriptionId") int prescriptionId) {
        try {
            Prescription prescription = prescriptionDAO.getPrescriptionById(prescriptionId);
            if (prescription == null) {
                return Response.status(status:Response.Status.NOT_FOUND)
                        .entity("Prescription with ID " + prescriptionId + " not found").build();
            }
            return Response.ok(entity:prescription).build();
        } catch (Exception e) {
            return Response.status(status:Response.Status.INTERNAL_SERVER_ERROR)
                    .entity("Error retrieving prescription: " + e.getMessage()).build();
        }
    }

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public Response addPrescription(Prescription prescription) {
        try {
            prescriptionDAO.addPrescription(prescription);
            return Response.status(status:Response.Status.CREATED).entity(entity:"Prescription added successfully").build();
        } catch (Exception e) {
            return Response.status(status:Response.Status.INTERNAL_SERVER_ERROR)
                    .entity("Error adding prescription: " + e.getMessage()).build();
        }
    }
```

Figure 47 - Error handling

Error Handling outputs

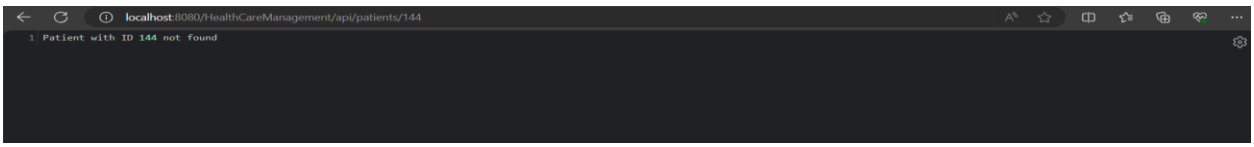


Figure 48 - Error handling output

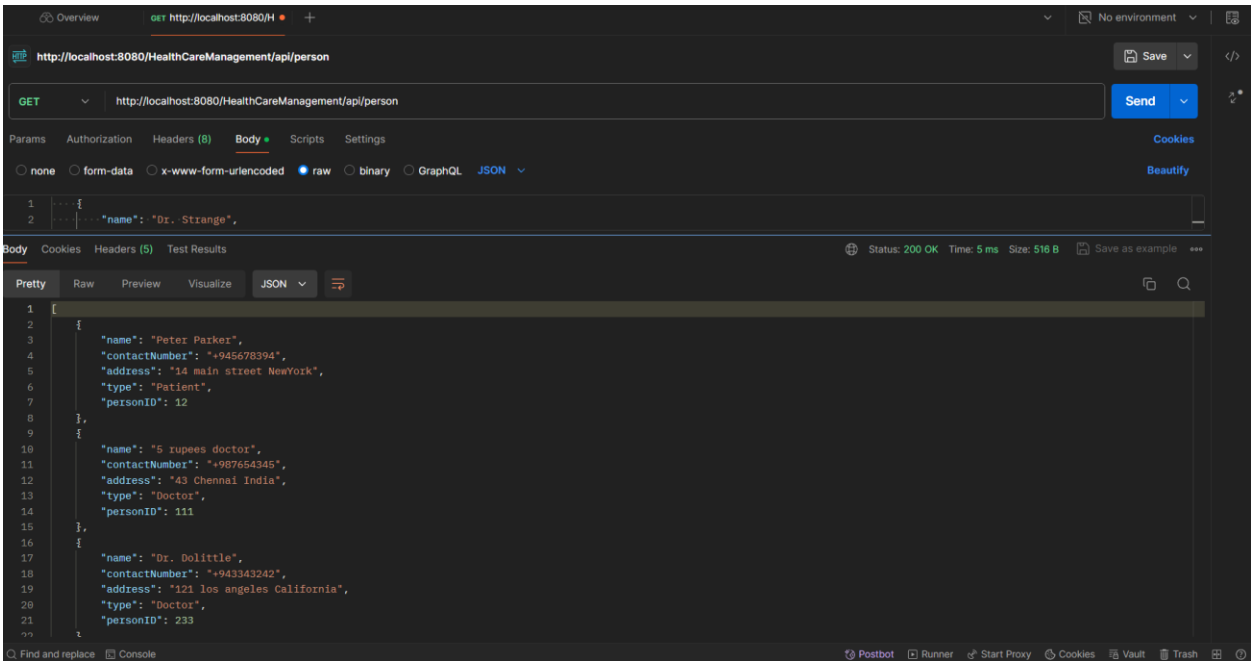
Postman Test Results

Below are the Postman test results for each CRUD operation:

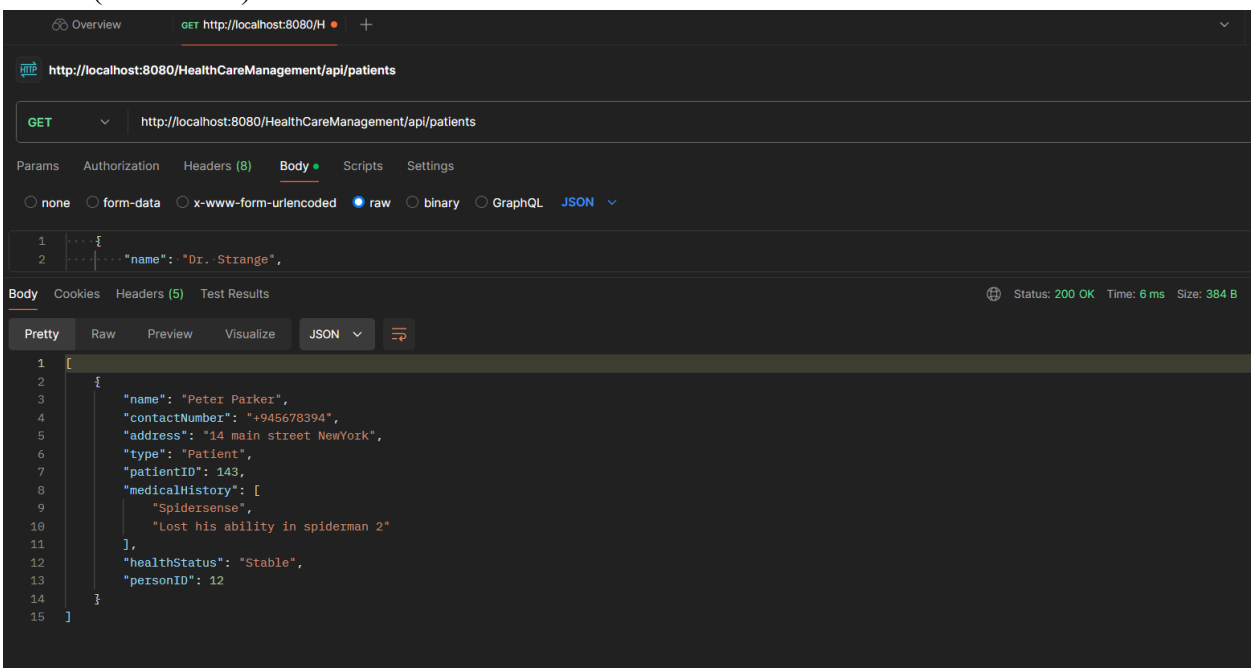
1. **Create (POST):** The POST request successfully created a new resource and returned a status code of 201 (Created). All mandatory fields were validated, and the response body contains the created resource's details.
2. **Read (GET):** The GET request fetched the resource(s) as expected, returning a status code of 200 (OK). The response body contains the requested data, and pagination, filtering, or sorting parameters were effectively applied if provided.
3. **Update (PUT/PATCH):** Both PUT and PATCH requests were tested for updating existing resources. PUT request resulted in a successful update with a status code of 200 (OK), replacing the entire resource with the provided data. PATCH request also successfully updated the resource, returning a status code of 200 (OK), and only modified fields were updated without affecting other parts of the resource.
4. **Delete (DELETE):** The DELETE request effectively removed the specified resource, returning a status code of 204 (No Content). Upon subsequent GET requests for the deleted resource, the server appropriately responded with a status code of 404 (Not Found), indicating that the resource no longer exists.

GET

Person(“/person”)



Patient(“/Patients”)



Doctor(“/doctors”)

http://localhost:8080/HealthCareManagement/api/doctors

GET http://localhost:8080/HealthCareManagement/api/doctors

Params Authorization Headers (8) Body Scripts Settings

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

```
{
    "name": "Dr. Strange",
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 6 ms Size: 484 B

Pretty Raw Preview Visualize JSON


```
[
    {
        "name": "5 rupees doctor",
        "contactNumber": "+987654345",
        "address": "43 Chennai India",
        "type": "Doctor",
        "specialization": "5 ruppes",
        "doctorID": 5,
        "personID": 111
    },
    {
        "name": "Dr. Dolittle",
        "contactNumber": "+943343242",
        "address": "121 los angeles California",
        "type": "Doctor",
        "specialization": "Veterinarian",
        "doctorID": 76,
        "personID": 233
    }
]
```


Appointment(“/appointments”)

Overview GET http://localhost:8080/H

http://localhost:8080/HealthCareManagement/api/appointments

GET http://localhost:8080/HealthCareManagement/api/appointments

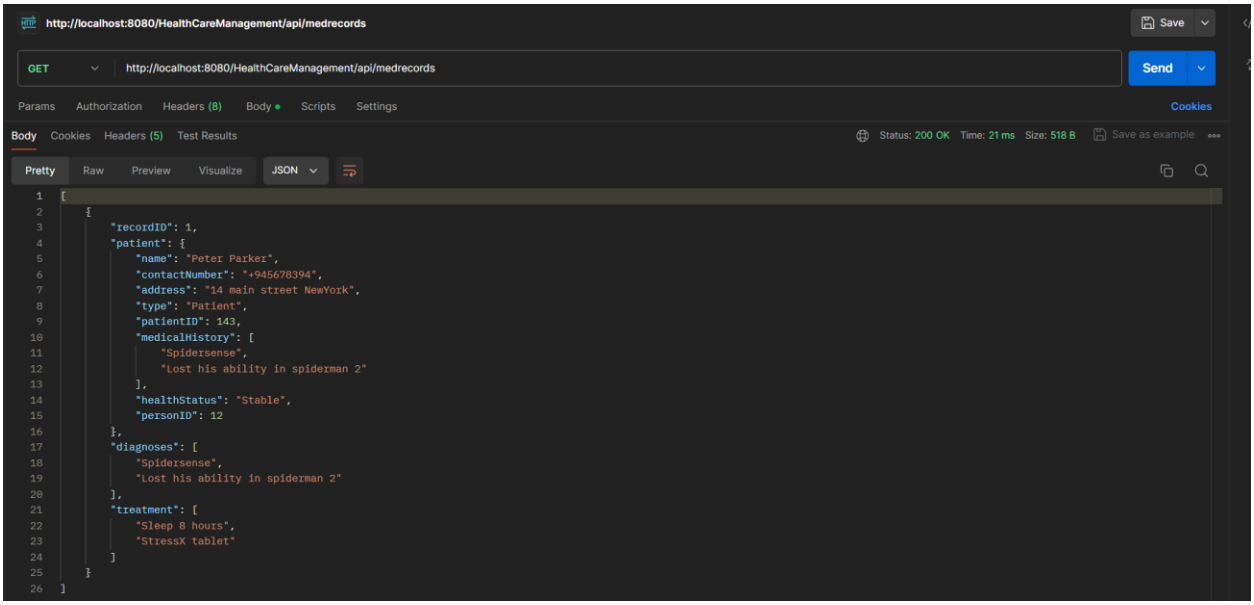
Params Authorization Headers (8) Body Scripts Settings

Body Cookies Headers (5) Test Results Status: 200 OK Time: 43 ms Size: 819 B

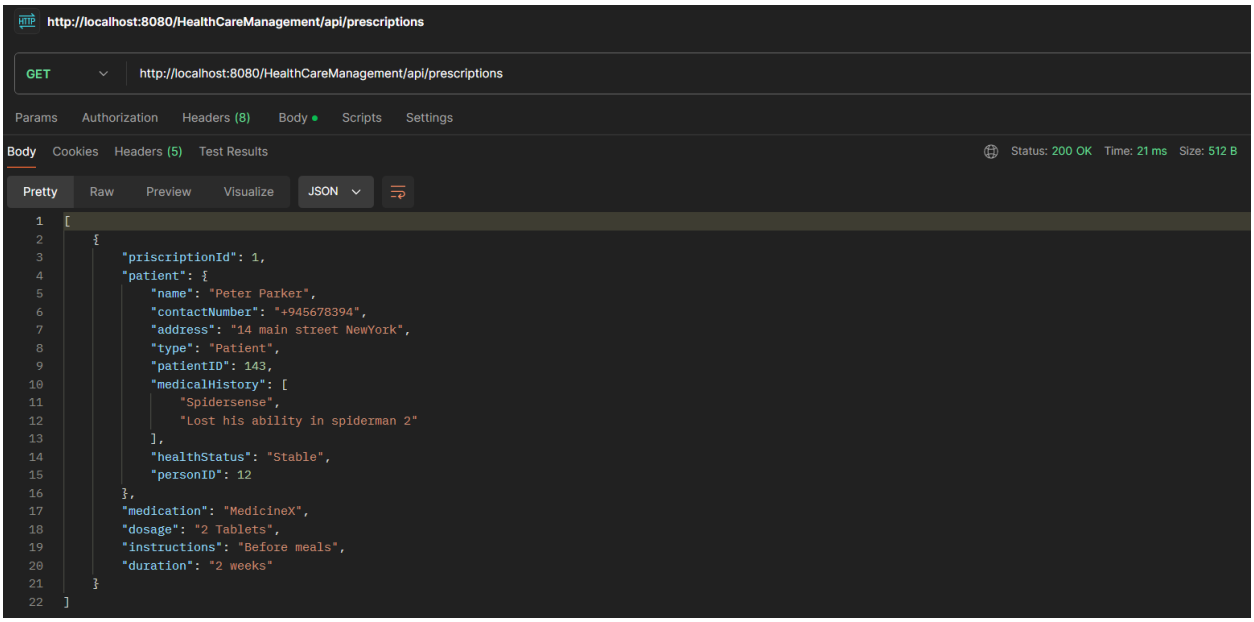
Pretty Raw Preview Visualize JSON


```
[
    {
        "appointmentId": 1,
        "patient": {
            "name": "Peter Parker",
            "contactNumber": "+945678394",
            "address": "14 main street NewYork",
            "type": "Patient",
            "patientID": 143,
            "medicalHistory": [
                "Spidersense",
                "Lost his ability in spiderman 2"
            ],
            "healthStatus": "Stable",
            "personID": 12
        },
        "doctor": {
            "name": "Dr. Dolittle",
            "contactNumber": "+943343242",
            "address": "121 los angeles California",
            "type": "Doctor",
            "specialization": "Veterinarian",
            "doctorID": 76,
            "personID": 233
        },
        "dateAndTime": {
```

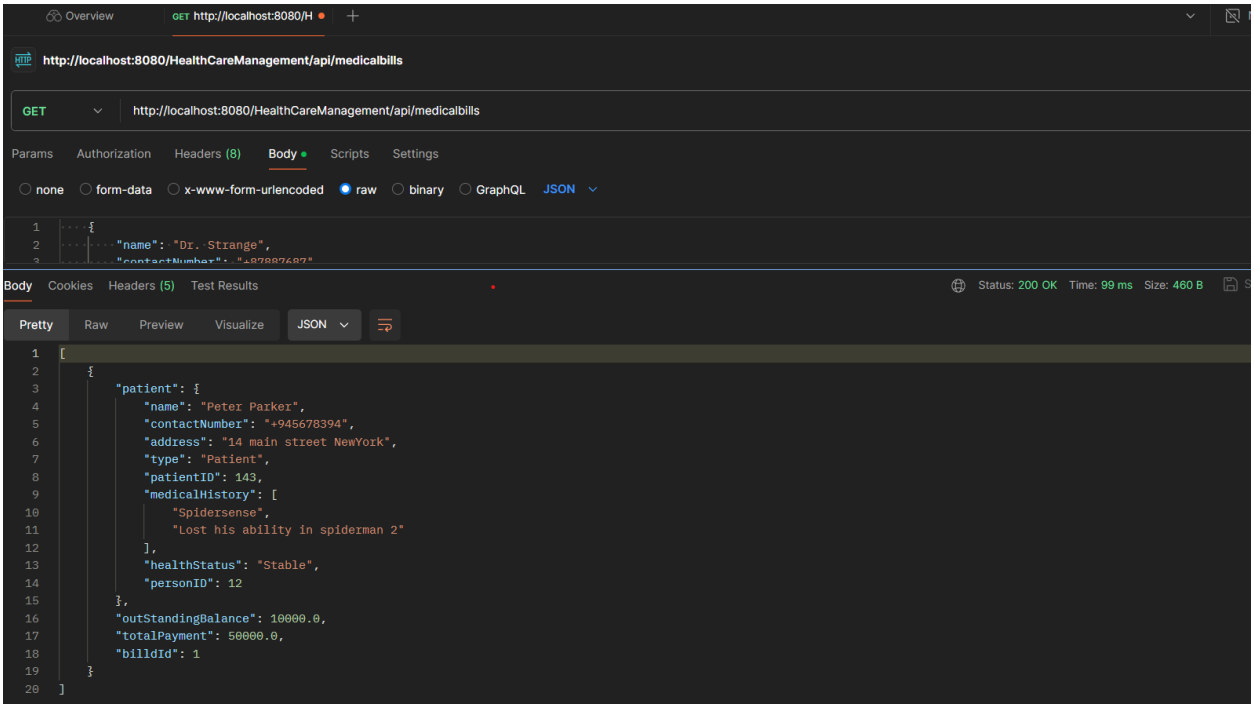
Medical-Record(“/medrecords”)



Prescriptions(“/prescriptions”)

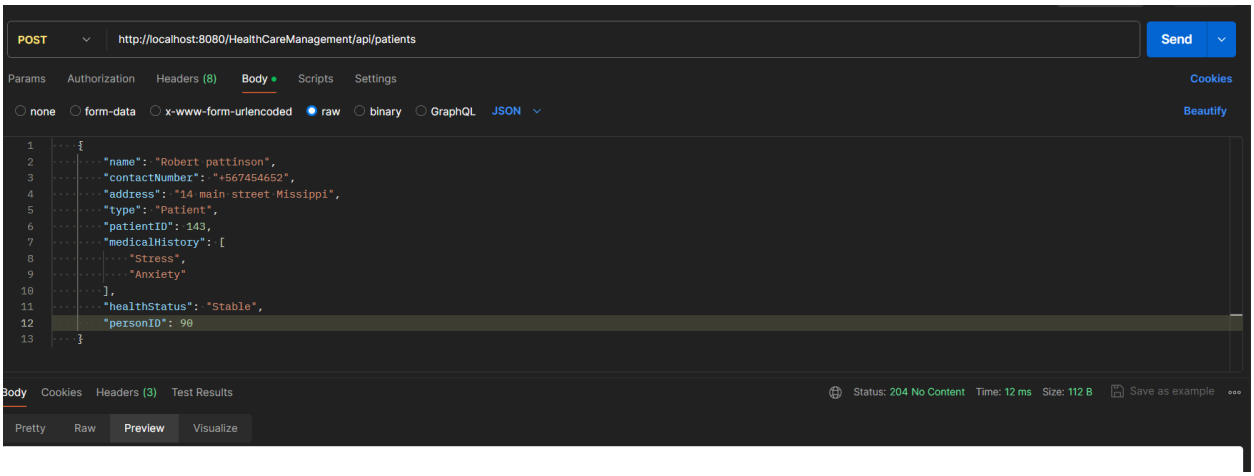


Billing("/medicalbills")

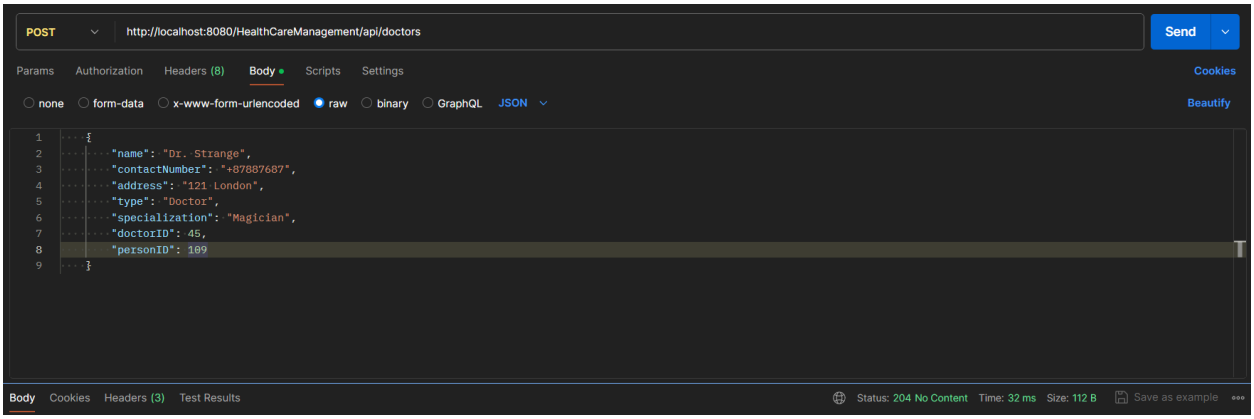


POST

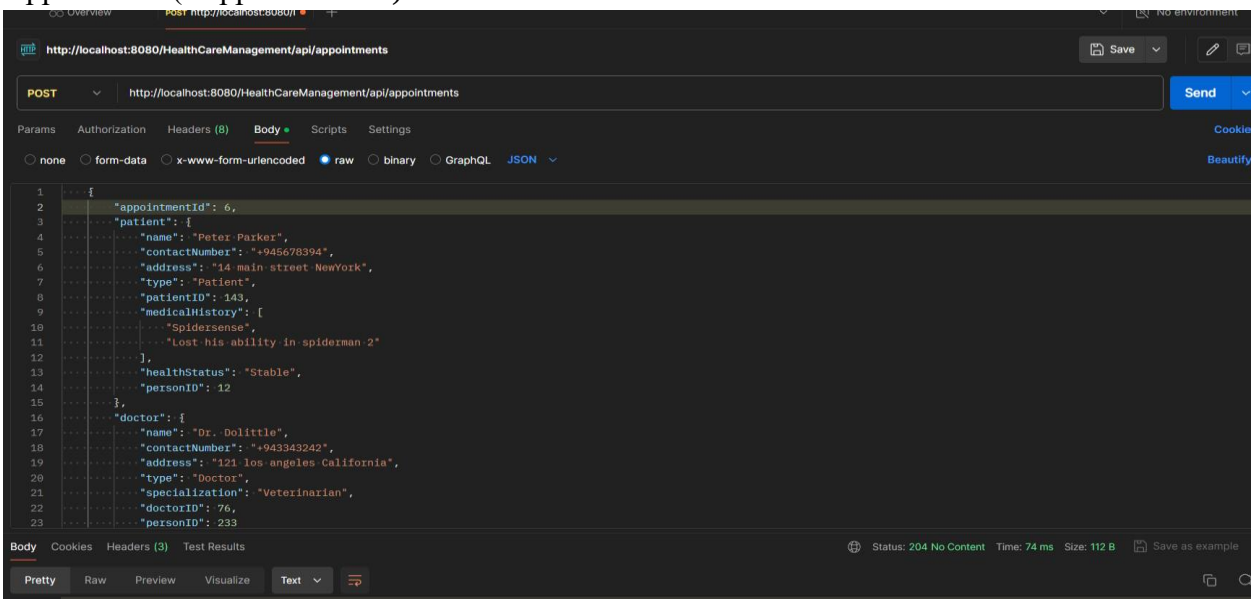
Patients("/patients")



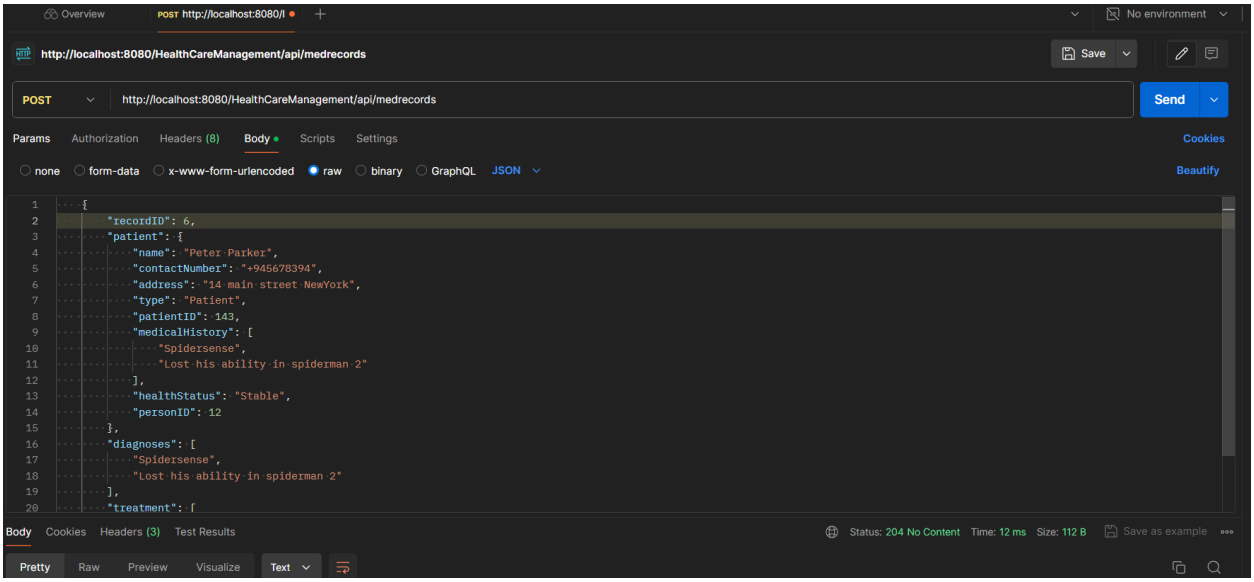
Doctor(“/doctors”)



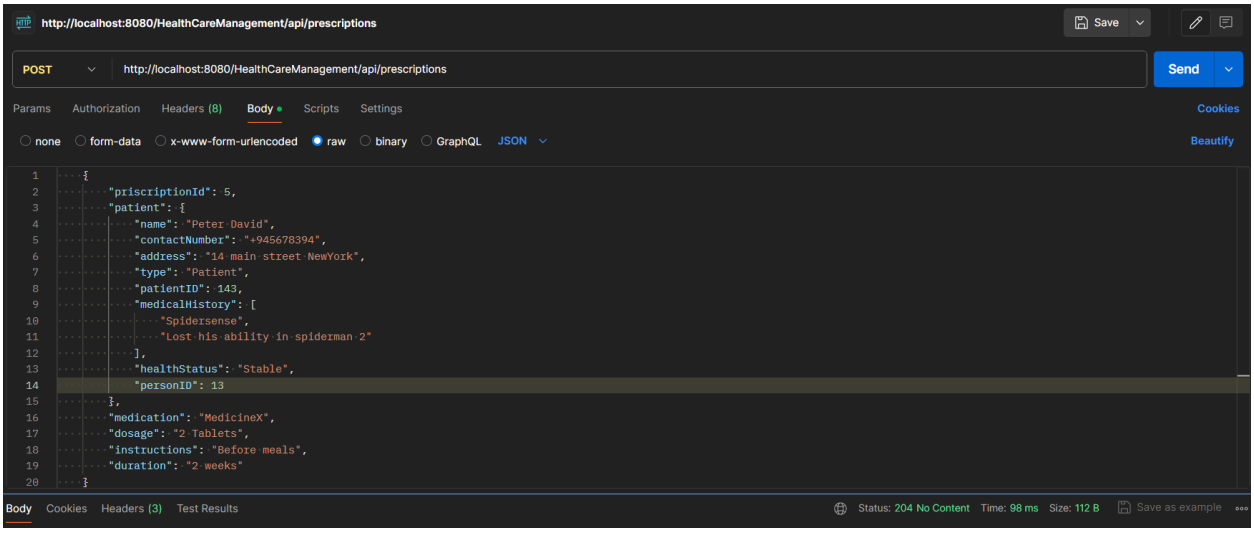
Appointment(“/appointments”)



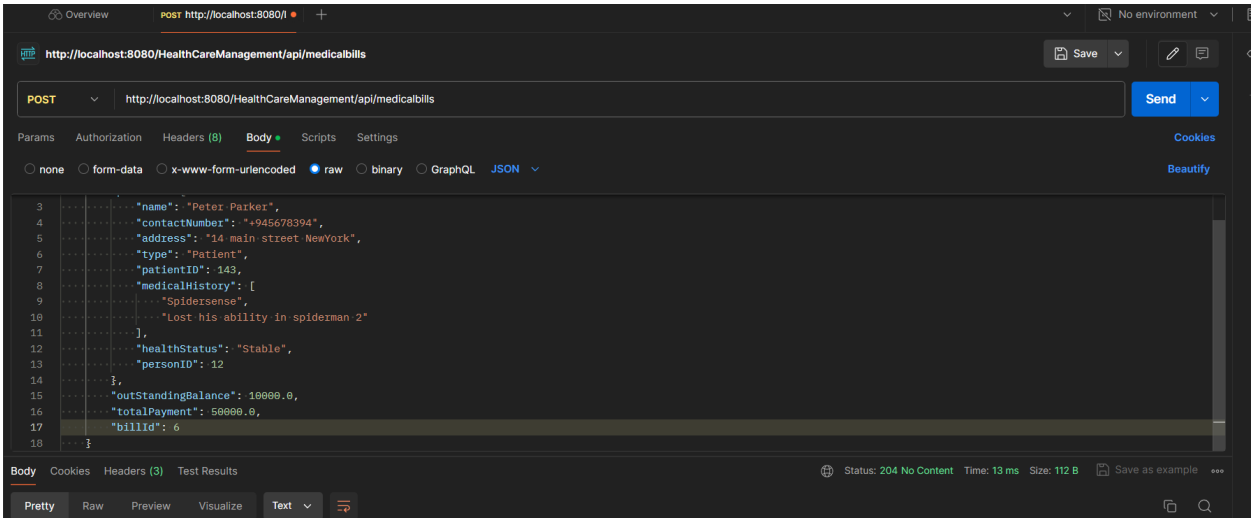
Medical-Record(“/medrecords”)



Prescriptions(“/prescriptions”)



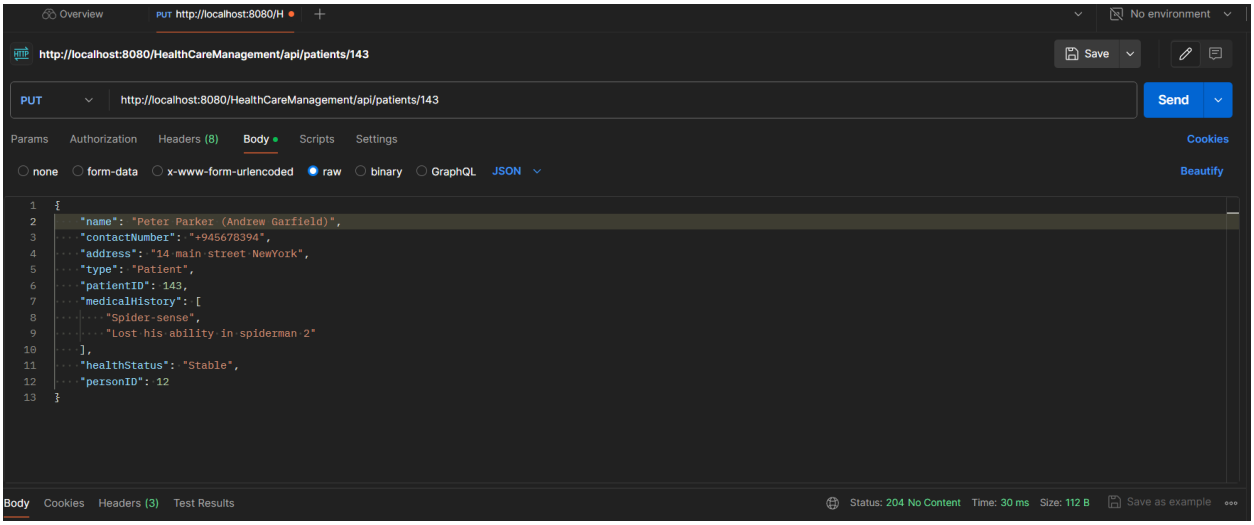
Billing(“/medicalbills”)



PUT

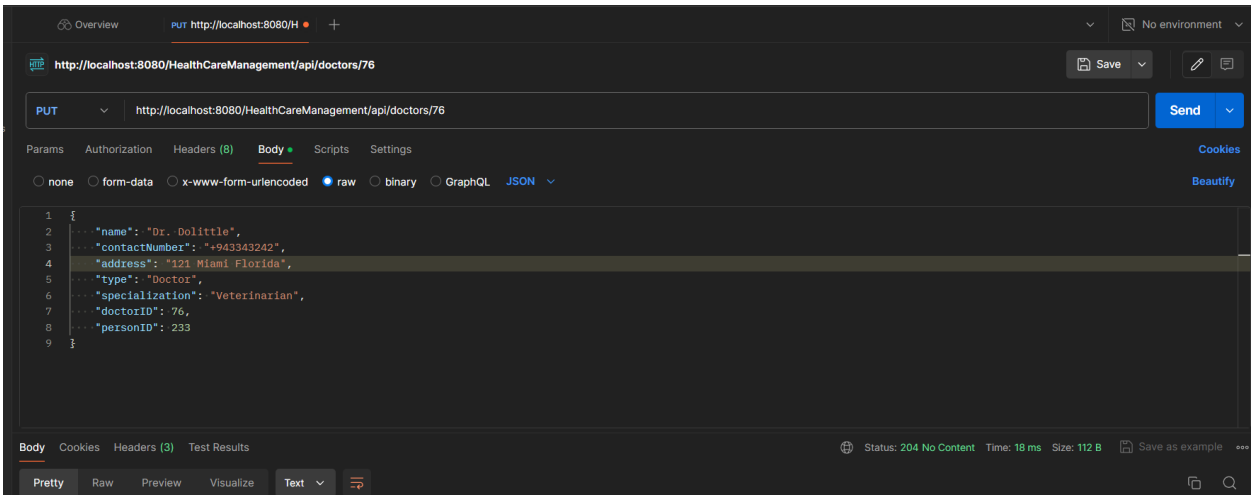
Patients(“/patients”)

Name has been updated

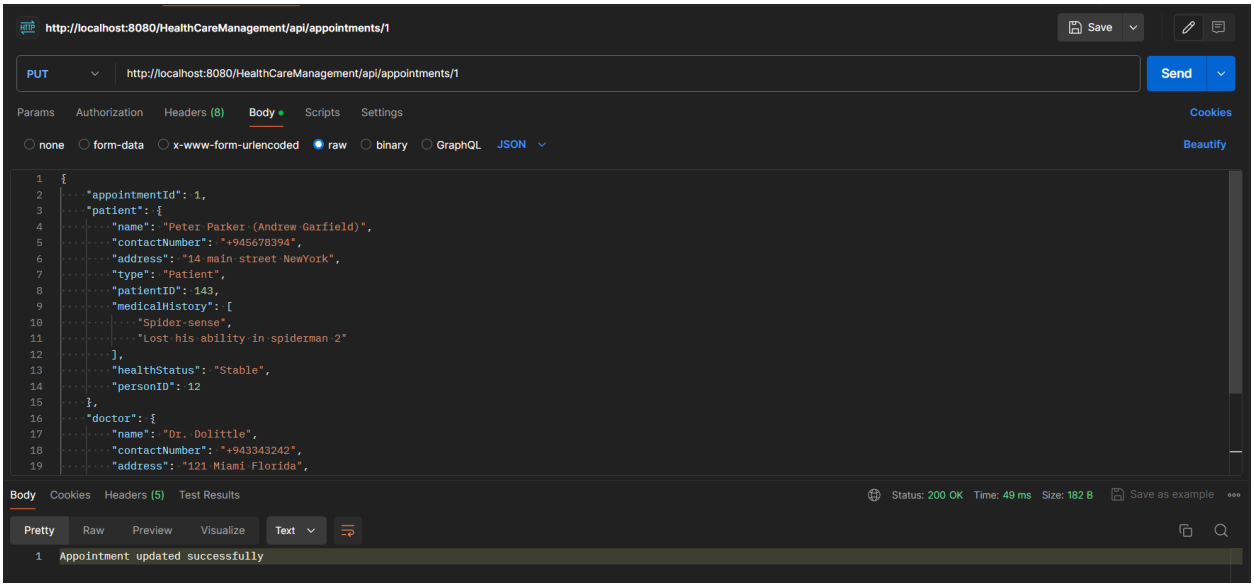


Doctor(“/doctors”)

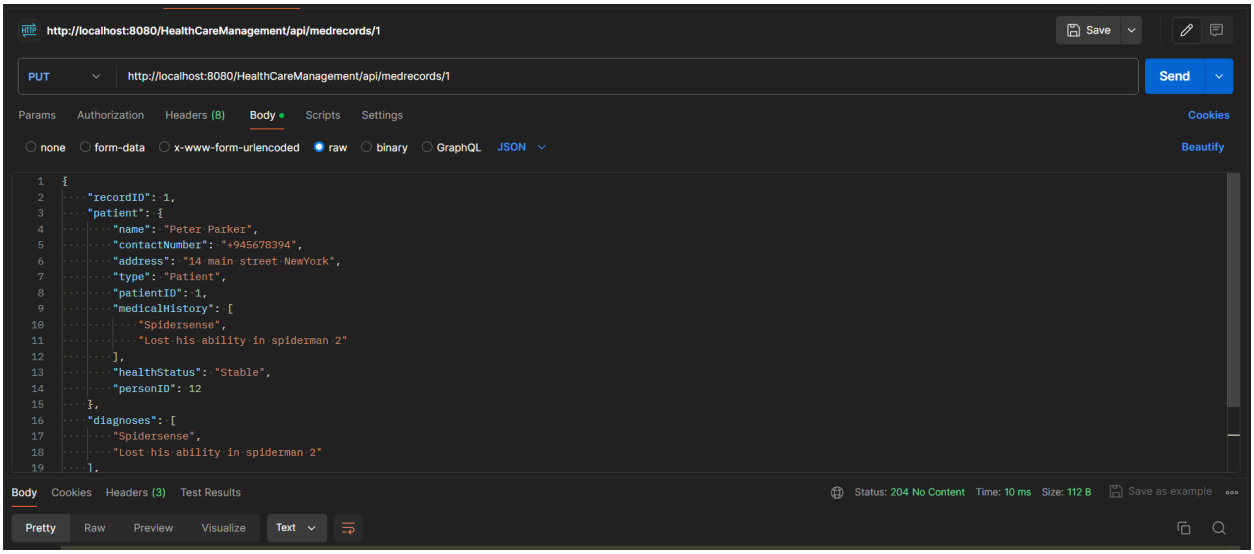
Updated the address



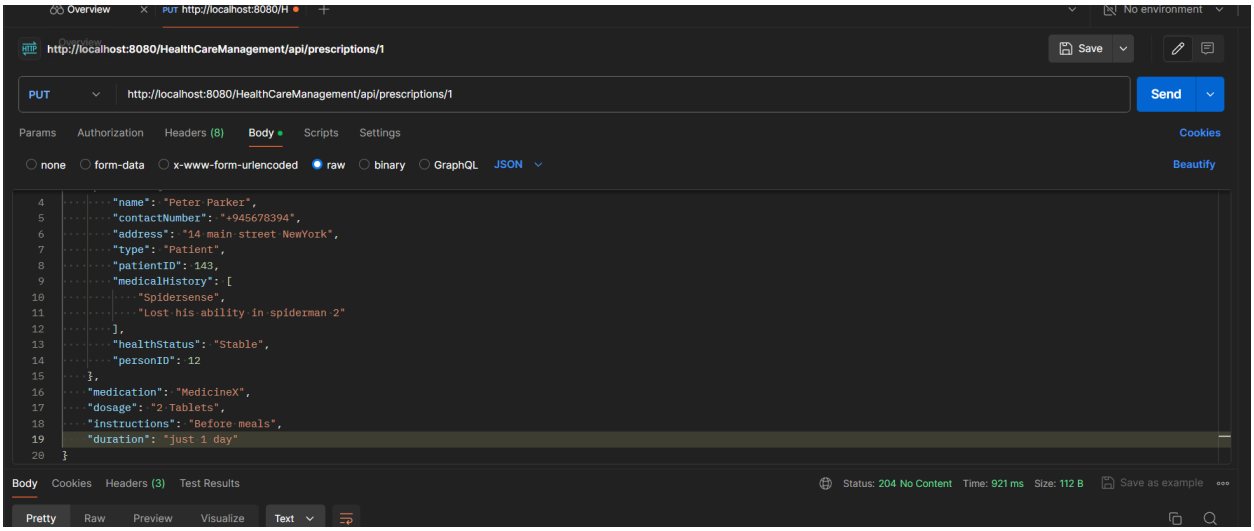
Appointment(“/appointments”)



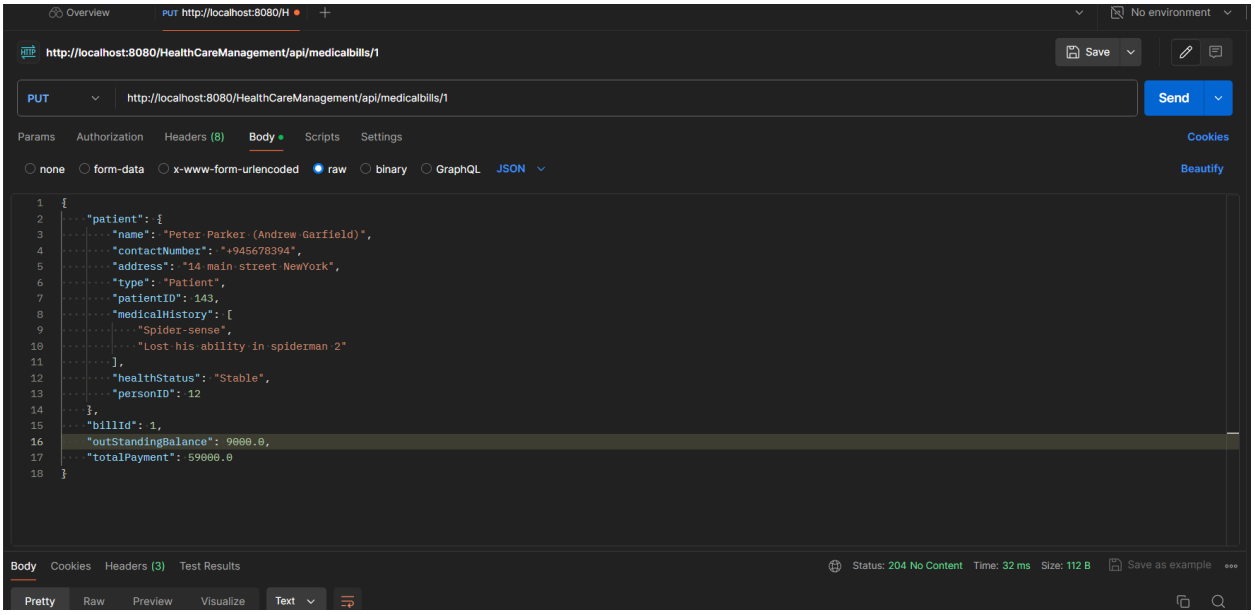
Medical-Record(“/medrecords”)



Prescriptions(“/prescriptions”)

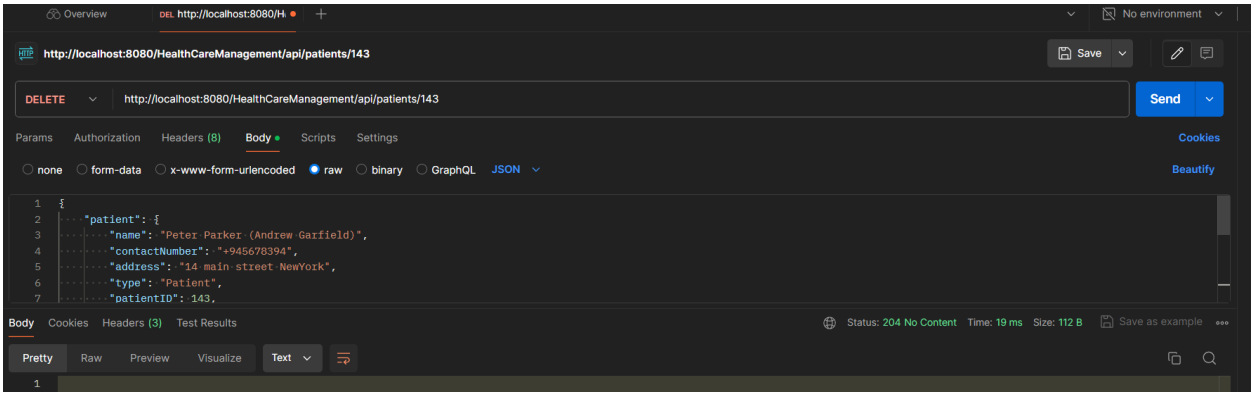


Billing(“/medicalbills”)

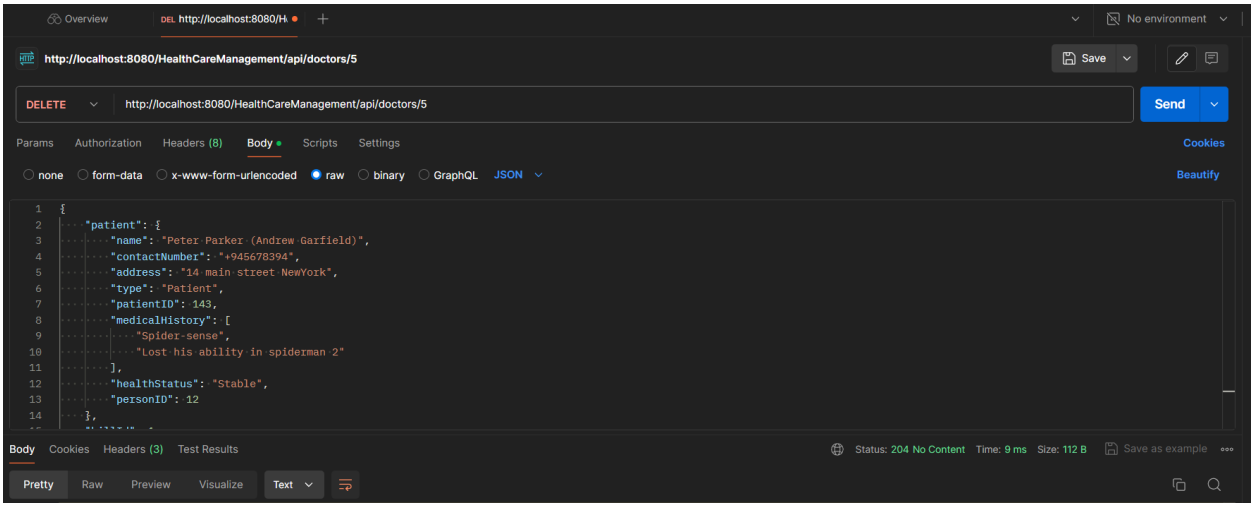


DELETE

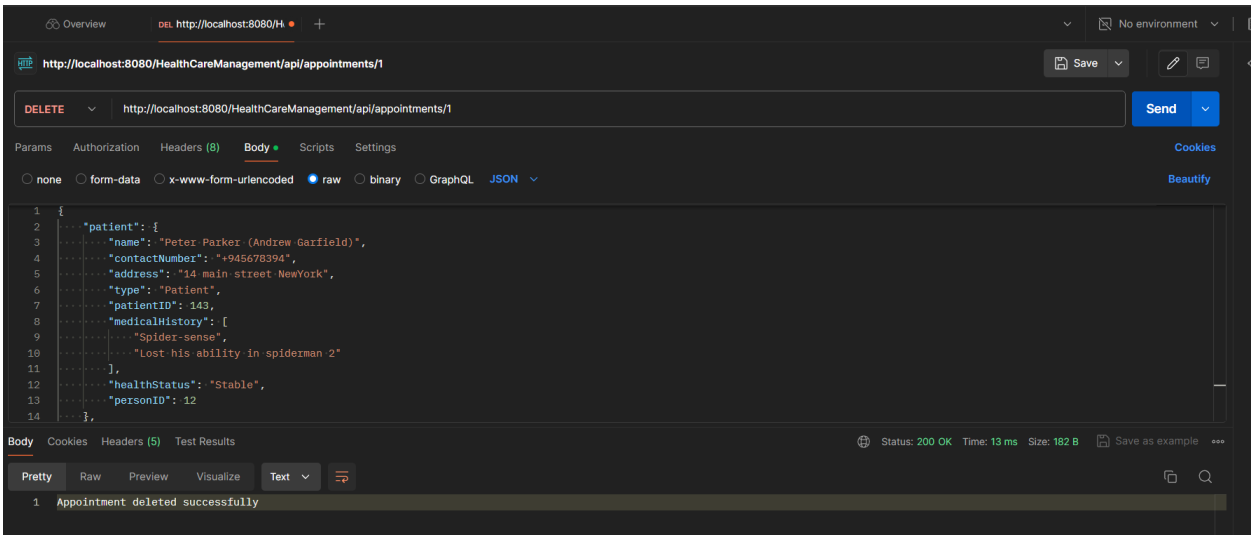
Patients(“/patients”)



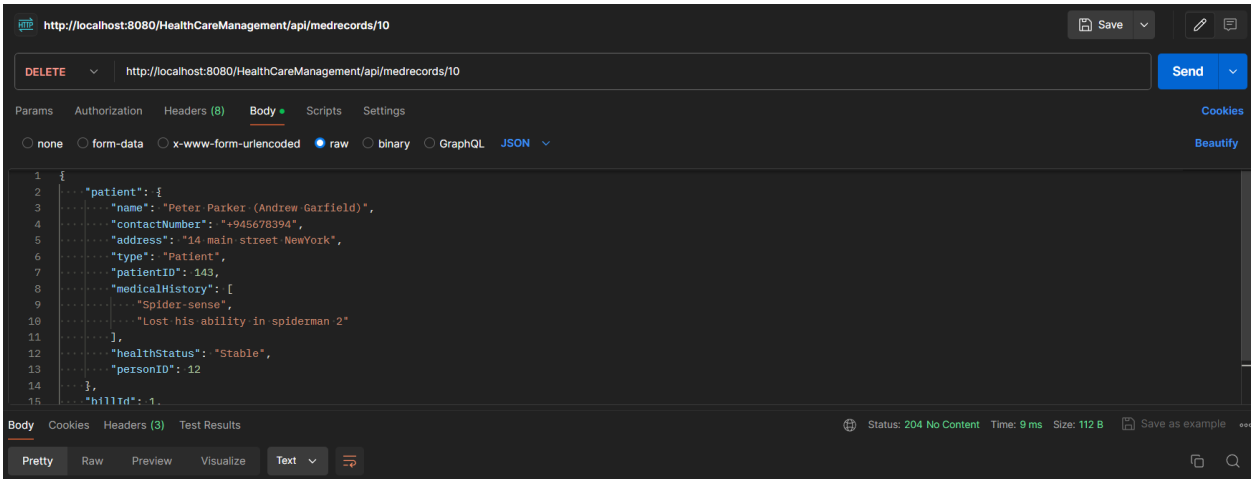
Doctor(“/doctors”)



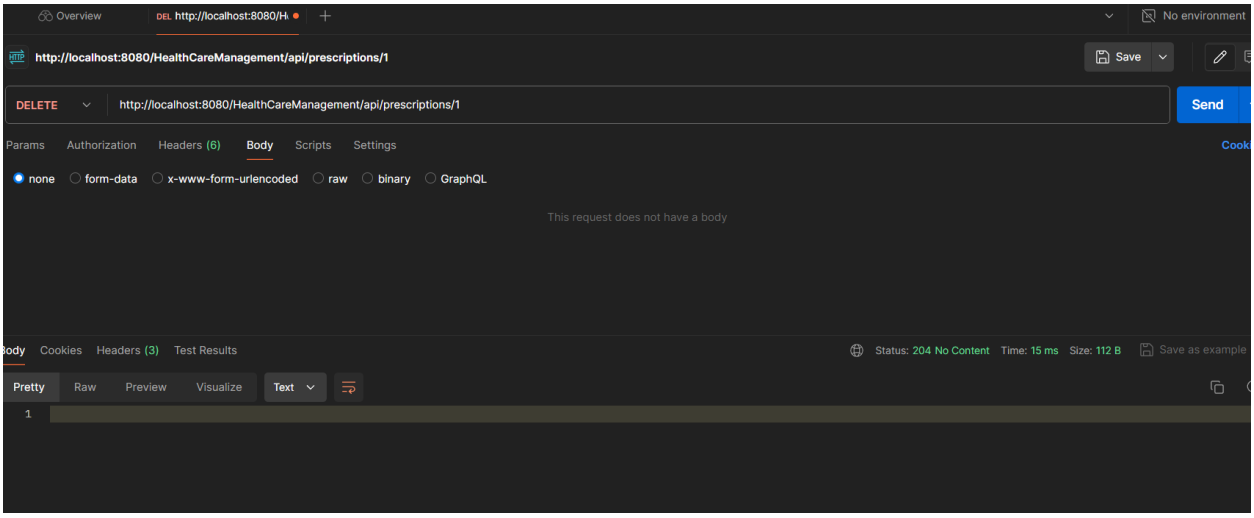
Appointment(“/appointments”)



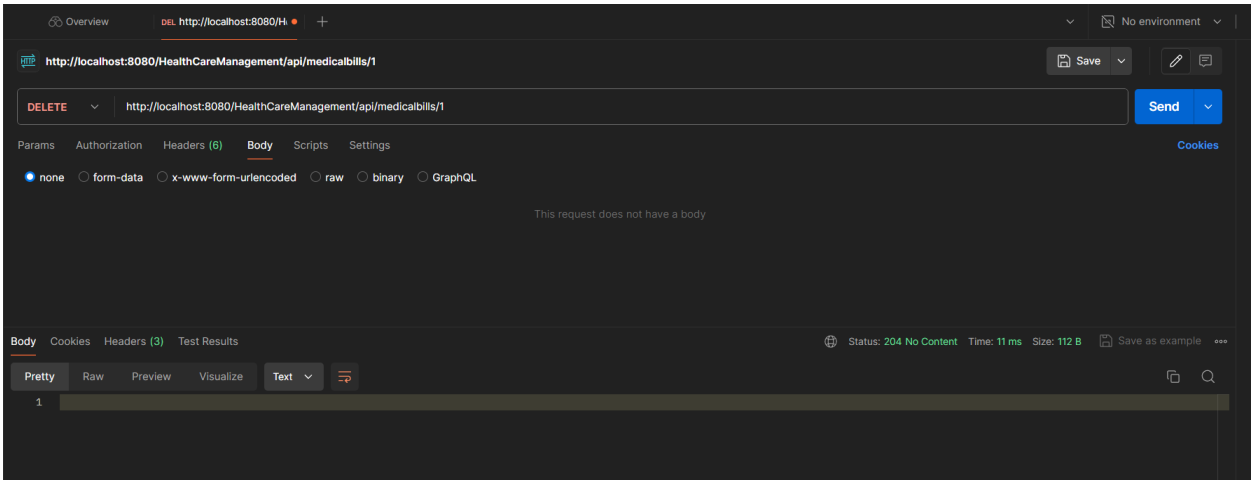
Medical-Record(“/medrecords”)



Prescriptions(“/prescriptions”)



Billing(“/medicalbills”)



Discussion

Challenges Faced

During the development process, several challenges were encountered, notably when dealing with Maven dependencies and utilizing NetBeans IDE. One significant difficulty arose when running Maven dependencies, leading to unexpected errors and disruptions in the build process. Additionally, there were instances where NetBeans experienced frequent crashes, hindering the development workflow, and causing delays in progress.

Solutions Implemented

To address these issues, various steps were taken. For Maven dependency-related problems, thorough troubleshooting was conducted, including reviewing and updating dependencies, resolving conflicts, and ensuring compatibility with the project requirements. This involved carefully examining the project's POM (Project Object Model) file, adjusting configurations, and seeking assistance from online resources and developer communities.

Conclusion

The project yielded a robust RESTful API using JAX-RS, catering specifically to healthcare management needs. With CRUD operations implemented for seven key entities—Person, Patient, Doctor, Medical Records, Medical Bills, Appointments, and Prescriptions—the system enables seamless interaction for efficient data management. Despite initial challenges with Maven dependencies and NetBeans IDE, diligent troubleshooting ensured successful completion.

By allowing for more interaction between the components, the project has room to grow. As of right now, each object functions alone, but future improvements might include creating smooth interactions between them. For example, the Patient entity might be connected to Appointments, Medical Bills, Medical Records, and Prescriptions, enabling data cross-referencing and automatic changes. This integration could improve overall healthcare management efficiency by streamlining procedures and improving data accuracy. The functionality and user experience of the system could be further enhanced by adding features like automated reminders for appointments or medication refills.

Appendix

Code Screenshots

```java
package com.mycompany.model;

/**
 *
 * @author Mahdi
 */
import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.databind.DeserializationContext;
import com.fasterxml.jackson.databind.JsonDeserializer;
import java.io.IOException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
public class LocalDateTimeDeserializer {
    private static final DateTimeFormatter FORMATTER = DateTimeFormatter.ISO_LOCAL_DATE_TIME;

    public LocalDateTime deserialize(JsonParser jsonParser, DeserializationContext deserializationContext)
            throws IOException {
        String dateTimeString = jsonParser.getValueAsString();
        return LocalDateTime.parse(text: dateTimeString, formatter:FORMATTER);
    }
}
```

```java
import com.mycompany.dao.PersonDAO;
import com.mycompany.model.Person;
import java.util.List;
import java.util.logging.Logger;
import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;

@Path("/person")
public class PersonResource {

    private PersonDAO personDAO = new PersonDAO();
    private static final Logger logger = Logger.getLogger(name: PersonResource.class.getName());


    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Person> findAll() {
        try {
            return personDAO.findAll();
        } catch (Exception e) {
            // Handle the exception
            throw new WebApplicationException(message: "Error occurred while retrieving all persons.", cause: e);
        }
    }

    @GET
    @Path("/{PersonID}")
    @Produces(MediaType.APPLICATION_JSON)
    public Person getPersonById(@PathParam("PersonID") int PersonID) {
```

```java
    @PUT
    @Path("/{PersonID}")
    @Consumes(MediaType.APPLICATION_JSON)
    public void updatePerson(@PathParam("PersonID") int PersonID, Person updatedPerson) {
        try {
            Person existingPatient = personDAO.getPersonById(personID: PersonID);

            if (existingPatient != null) {
                updatedPerson.setPersonID(PersonID);
                personDAO.updatePerson(updatedPerson);
            } else {
                throw new NotFoundException("Person with ID " + PersonID + " not found.");
            }
        } catch (Exception e) {
            // Handle the exception
            throw new WebApplicationException("Error occurred while updating person with ID " + PersonID, cause: e);
        }
    }

    @DELETE
    @Path("/{PersonID}")
    public void deletePerson(@PathParam("PersonID") int PersonID) {
        try {
            personDAO.deletePerson(id: PersonID);
        } catch (Exception e) {
            // Handle the exception
            throw new WebApplicationException("Error occurred while deleting person with ID " + PersonID, cause: e);
        }
    }
}
```

```java

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Prescription> getAllPrescriptions() {
        return prescriptionDAO.getAllPrescriptions();
    }

    @GET
    @Path("{priscriptionId}")
    @Produces(MediaType.APPLICATION_JSON)
    public Prescription getPrescriptionById(@PathParam("priscriptionId") int priscriptionId) {
        return prescriptionDAO.getPrescriptionById(prescriptionId: priscriptionId);
    }

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public void addPrescription(Prescription prescription) {
        prescriptionDAO.addPrescription(prescription);
    }

    @PUT
    @Path("/{priscriptionId}")
    @Consumes(MediaType.APPLICATION_JSON)
    public void updatePrescription(@PathParam("prescriptionId") int priscriptionId, Prescription prescription) {
        // Ensuring the ID in the URL matches the ID in the object
        prescription.setPriscriptionId(priscriptionId);
        prescriptionDAO.updatePrescription(updatedPrescription:prescription);
    }

    @DELETE
    @Path("/{priscriptionId}")
    public void deletePrescription(@PathParam("priscriptionId") int prescriptionId) {
        prescriptionDAO.deletePrescription(id: prescriptionId);
    }
```

```java
@Path("/appointments")
public class AppointmentResource {

    private AppointmentDAO appointmentDAO = new AppointmentDAO();


    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Appointment> getAllAppointments() {
        return appointmentDAO.getAllAppointments();
    }

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public void addAppointments(Appointment appointment) {
        appointmentDAO.addAppointments(appointment);
    }

    @GET
    @Path("/{appointmentId}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response getAppointmentById(@PathParam("appointmentId") int appointmentId) {
        // Retrieve the appointment by ID from your DAO or service layer
        Appointment appointment = appointmentDAO.getAppointmentById(appointmentId);

        // Check if the appointment exists
        if (appointment == null) {
            return Response.status(status:Response.Status.NOT_FOUND).entity("Appointment" + appointmentId).build();
        }
        // Return the response
        return Response.ok().entity(entity:appointment).build();
    }
```

```java

    // Method to update an existing Appointment
    @PUT
    @Path("/{appointmentId}")
    @Consumes(MediaType.APPLICATION_JSON)
    public Response updateAppointment(@PathParam("appointmentId") int appointmentId, Appointment updatedAppointment) {
        try {
            Appointment existingAppointment = appointmentDAO.getAppointmentById(appointmentId);
            if (existingAppointment == null) {
                throw new NotFoundException("Appointment not found with ID: " + appointmentId);
            }
            updatedAppointment.setAppointmentId(appointmentId);
            appointmentDAO.updateAppointment(updatedAppointment);   // update the appointment
            // Return the success response
            return Response.status(status:Response.Status.OK).entity(entity:"Appointment updated successfully").build();
        } catch (Exception e) {
            // Throw WebApplicationException
            throw new WebApplicationException(message: "Error in updating the appointment", cause: e, status:500);
        }
    }

    // Method to delete an Appointment
    @DELETE
    @Path("/{appointmentId}")
    public Response deleteAppointment(@PathParam("appointmentId") int appointmentId) {
        try {    // Delete the appointment
            appointmentDAO.deleteAppointment(id: appointmentId);
            // Return the success response
            return Response.status(status:Response.Status.OK).entity(entity:"Appointment deleted successfully").build();
        } catch (Exception e) {
            // Throw WebApplicationException
            throw new WebApplicationException(message: "Error in deleting the appointment with ID", cause: e, status:500);
        }
    }
```

Output × | DoctorResource.java × | PrescriptionResource.java × | RecordResource.java × | BillingResource.java ×

Source | History

```java
        try {
            List<Doctor> doctors = doctorDao.getAllDoctors();
            return Response.ok(entity:doctors).build();
        } catch (Exception e) {
            return Response.status(status:Response.Status.INTERNAL_SERVER_ERROR)
                    .entity("Error retrieving doctors: " + e.getMessage()).build();
        }
    }

    @GET
    @Path("/{DoctorID}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response getDoctorById(@PathParam("DoctorID") int DoctorID) {
        try {
            Doctor doctor = doctorDao.getDoctorById(id: DoctorID);
            if (doctor == null) {
                return Response.status(status:Response.Status.NOT_FOUND)
                        .entity("Doctor with ID " + DoctorID + " not found").build();
            }
            return Response.ok(entity:doctor).build();
        } catch (Exception e) {
            return Response.status(status:Response.Status.INTERNAL_SERVER_ERROR)
                    .entity("Error retrieving doctor: " + e.getMessage()).build();
        }
    }

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public Response addDoctor(Doctor doctor) {
        try {
            doctorDao.addDoctor(doctor);
            return Response.status(status:Response.Status.CREATED).entity(entity:"Doctor added successfully").build();
        } catch (Exception e) {
            return Response.status(status:Response.Status.INTERNAL_SERVER_ERROR)
```

com.mycompany.resources.DoctorResource > getAllDoctors > try > catch Exception e

```java

    // Method to update an existing Appointment
    @PUT
    @Path("/{appointmentId}")
    @Consumes(MediaType.APPLICATION_JSON)
    public Response updateAppointment(@PathParam("appointmentId") int appointmentId, Appointment updatedAppointment) {
        try {
            Appointment existingAppointment = appointmentDAO.getAppointmentById(appointmentId);
            if (existingAppointment == null) {
                throw new NotFoundException("Appointment not found with ID: " + appointmentId);
            }
            updatedAppointment.setAppointmentId(appointmentId);
            appointmentDAO.updateAppointment(updatedAppointment);   // update the appointment
            // Return the success response
            return Response.status(status:Response.Status.OK).entity(entity:"Appointment updated successfully").build();
        } catch (Exception e) {
            // Throw WebApplicationException
            throw new WebApplicationException(message: "Error in updating the appointment", cause: e, status:500);
        }
    }

    // Method to delete an Appointment
    @DELETE
    @Path("/{appointmentId}")
    public Response deleteAppointment(@PathParam("appointmentId") int appointmentId) {
        try {    // Delete the appointment
            appointmentDAO.deleteAppointment(id: appointmentId);
            // Return the success response
            return Response.status(status:Response.Status.OK).entity(entity:"Appointment deleted successfully").build();
        } catch (Exception e) {
            // Throw WebApplicationException
            throw new WebApplicationException(message: "Error in deleting the appointment with ID", cause: e, status:500);
        }
    }
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>

    <servlet>
        <servlet-name>HealthCareManagement</servlet-name>
        <servlet-class>
            org.glassfish.jersey.servlet.ServletContainer
        </servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>com.mycompany.resources</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>HealthCareManagement</servlet-name>
        <url-pattern>/api/*</url-pattern>
    </servlet-mapping>
</web-app>
```