UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

University of Westminster

Informatics Institute of Technology

Department of Computing
(2023/24) 5COSC022C Client-Server Architectures

Module	: 5COSC022C Client-Server Architectures
Module Leader	: Mr. Cassim Farook
Date of submission	: 07/05/2024
Student First Name	: Thantrige Sahan Vimukthi Dharmarathne
Tutorial	: Group E
IIT ID	: 20222165
UOW ID	: W1985549

Demo Video Link:
https://drive.google.com/file/d/1r6Y_caVIt_bHZCSb8QT0voIZrmGIqjYF/view?usp=sharing

Table of Contents

Introduction

A technique of communication between two or more computer programs or components is an application programming interface or API. This report discusses a health system application programming interface (Health System API). This health system API, which aims to simplify complicated tasks, maximize resource allocation, and optimize patient care, is a fantastic tool for contemporary healthcare management. Because it provides necessary features including patient administration, scheduling appointments, healthcare record-keeping, medication management, and invoicing functions. All these health system API's functionalities are more effective. Interoperability, scalability, and adaptability are guaranteed by RESTful principles, which are in line with modern software development methodologies. The Postman API tests validate the API's accuracy and performance. Postman is a collaboration platform for development of various type of API s. Postman is a collaboration platform that helps a lot in developing different types of APIs.

Overview of the project

The objective of this project is to use JAX-RS with the understanding of REST API principles and development to create functionalities for patient management, appointment scheduling medical record-keeping, prescription administration, and payment management. Establishing system entities with distinct features, such as Person, Patient, Doctor, Appointment, Medical Record, Prescription, and Billing, is one of the key tasks. In addition, the project results in the Data Access Object (DAO) classes and Resource classes, together with CRUD functions, logging, and exception handling, being implemented for every model class.

API endpoints and HTTP methods with paths

Person Entity

GET	/20222165_W1985549_CSA_CW/rest/persons/allPersons And /20222165_W1985549_CSA_CW/rest/persons/{personId} Both URI paths will give the person informations(datasets). The difference is the 'allPersons' in the first path will gives all the existing persons details and the second path will give the details about the relevant person's details according to the 'personId'.
POST	/20222165_W1985549_CSA_CW/rest/persons/create/{personId} This path will give the ability to create a new person with relevant details(dataset) using a new 'personId'.
PUT	/20222165_W1985549_CSA_CW/rest/persons/update/{personId} This path will update the information(dataset) of an existing person using the relevant existing 'personId'.
DELETE	/20222165_W1985549_CSA_CW/rest/persons/delete/{personId} This path will delete the existing person with details(dataset) from relevant array list using 'personId'.

Patient Entity

GET	/20222165_W1985549_CSA_CW/rest/patients/allPatient And /20222165_W1985549_CSA_CW/ rest/patients/{patientId} Both URI paths will give the patient informations(datasets). The difference is the 'allPatients' in the first path will gives all the existing patients details and the second path will give the details about the relevant patients's details according to the 'patientId'.
POST	/20222165_W1985549_CSA_CW/rest/patients /create/{patientId} This path will give the ability to create a new patient with relevant details(dataset) using a new 'patientId'.
PUT	/20222165_W1985549_CSA_CW/rest/patients/ update/{patientId}

	This path will update the information(dataset) of an existing patient using the relevant existing ‘patientId’.
DELETE	/20222165_W1985549_CSA_CW/rest/patients/ delete/{patientId} This path will delete the existing patient with details(dataset) from relevant array list using ‘patientId’.

Doctor Entity

GET	/20222165_W1985549_CSA_CW/rest/doctors/allDoctors And /20222165_W1985549_CSA_CW/rest/doctors/{doctorId} Both URI paths will give the doctor informations(datasets). The difference is the ‘allDoctors’ in the first path will gives all the existing doctors details and the second path will give the details about the relevant doctor’s details according to the ‘doctorId’.
POST	/20222165_W1985549_CSA_CW/rest/doctors/create/{doctorId} This path will give the ability to create a new doctor with relevant details(dataset) using a new ‘doctorId’.
PUT	/20222165_W1985549_CSA_CW/rest/doctors/update/{doctorId} This path will update the information(dataset) of an existing doctor using the relevant existing ‘doctorId’.
DELETE	/20222165_W1985549_CSA_CW/rest/doctors/delete/{doctorId} This path will delete the existing doctor with details(dataset) from relevant array list using ‘doctorId’.

Appointment Entity

GET	/20222165_W1985549_CSA_CW/rest/appointments/allAppointments And /20222165_W1985549_CSA_CW/rest/appointments/{appointId} Both URI paths will give the appointment informations(datasets). The difference is the ‘allAppointments’ in the first path will gives all the existing appointments details and the second path will give the details about the relevant appointment’s details according to the ‘appointId’.
POST	/20222165_W1985549_CSA_CW/rest/appointments/create/{appointId} This path will give the ability to create a new appointment with relevant details(dataset) using a new ‘appointId’.
PUT	/20222165_W1985549_CSA_CW/rest/appointments/update/{appointId}

	This path will update the information(dataset) of an existing appointment using the relevant existing ‘appointId’.
DELETE	/20222165_W1985549_CSA_CW/rest/appointments/delete/{appointId} This path will delete the existing appointment with details(dataset) from relevant array list using ‘appointID’.

Medical Record Entity

GET	/20222165_W1985549_CSA_CW/rest/medRecords/allMedRecords And /20222165_W1985549_CSA_CW/rest/medRecords/{medRecId} Both URI paths will give the medical record informations(datasets). The difference is the ‘allMedRecords’ in the first path will gives all the existing medical record details and the second path will give the details about the relevant medical record’s details according to the ‘medRecId’.
POST	/20222165_W1985549_CSA_CW/rest/medRecords/create/{medRecId} This path will give the ability to create a new medical record with relevant details(dataset) using a new ‘medRecId’.
PUT	/20222165_W1985549_CSA_CW/rest/medRecords/update/{medRecId} This path will update the information(dataset) of an existing medical record using the relevant existing ‘medRecId’.
DELETE	/20222165_W1985549_CSA_CW/rest/medRecords/delete/{medRecId} This path will delete the existing medical record with details(dataset) from relevant array list using ‘medRecId’.

Prescription Entity

GET	/20222165_W1985549_CSA_CW/rest/prescriptions/allPrescripts And /20222165_W1985549_CSA_CW/rest/prescriptions/{prescriptId} Both URI paths will give the prescription information(datasets). The difference is the ‘allPrescripts in the first path will gives all the existing prescription details and the second

	path will give the details about the relevant prescription's details according to the 'prescriptId.
POST	/20222165_W1985549_CSA_CW/rest/prescriptions/create/{prescriptId} This path will give the ability to create a new prescription with relevant details(dataset) using a new 'prescriptId.
PUT	/20222165_W1985549_CSA_CW/rest/prescriptions/update/{prescriptId} This path will update the information(dataset) of an existing prescription using the relevant existing 'prescriptId.
DELETE	/20222165_W1985549_CSA_CW/rest/prescriptions/delete/{prescriptId} This path will delete the existing prescription with details(dataset) from relevant array list using 'prescriptId.

Billing Entity

GET	/20222165_W1985549_CSA_CW/rest/billResource/ allBills And /20222165_W1985549_CSA_CW/rest/billResource/{billId} Both URI paths will give the bill informations(datasets). The difference is the 'allBills in the first path will gives all the existing bills details and the second path will give the details about the relevant bills's details according to the 'billId'.
POST	/20222165_W1985549_CSA_CW/rest/billResource/create/{billId} This path will give the ability to create a new bill with relevant details(dataset) using a new 'billId'.
PUT	/20222165_W1985549_CSA_CW/rest/billResource/update/{billId} This path will update the information(dataset) of an existing bill using the relevant existing 'billId'.
DELETE	/20222165_W1985549_CSA_CW/rest/billResource/delete/{billId} This path will delete the existing bill with details(dataset) from relevant array list using 'billId'.

Implementation

Code Organization

The project has four main packages.

1. com.company.model: Created to allocate all the model classes of the project. Such as Person, Patient, Doctor, Appointment, MedicalRecord, Prescription, and Billing.

2. com.company.dao: Created to allocate all the Data Access Object(DAO) classes of the project. Such as PersonDAO, PatientDAO, DoctorDAO, AppointmentDAO, MedicalRecordDAO, PrescriptionDAO, and BillingDAO.

3. com.company.resource: Created to allocate all the resource classes of the project. . Such as PersonResource, PatientResource, DoctorResource, AppointmentResource, MedicalRecordResource, PrescriptionResource, and BillingResource.

4. com.company.exception: Created to allocate the ResourceNotFoundException class and ResourceNotFoundExceptionMapper class to handle exceptions using logger and logger factory.

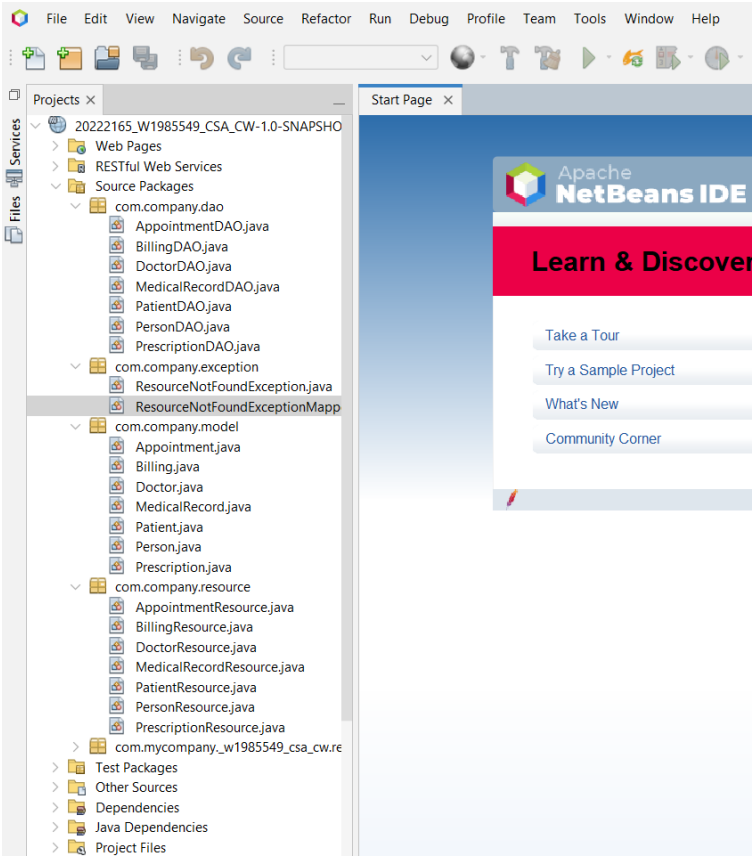


Figure 1: Code Organization

Implementation of model classes, DAOs, and RESTful Resources classes

Person, PersonDAO, and PersonResource

Private attributes related to Person class is initialized in the Person model class. This person model class is super class.It inherits two child classes called Patient and Doctor.Also all the relevant getters and setters are defined in the Person class. The PersonDAO class is included with all the CRUD operations which is Create, Read, Update, and Delete. In sequence they are createPerson(), getAllPersons, getPersonById(), updatePerson(), deletePerson(). Apart from that the PersonResource class is filled with relevant RESTful HTTP methods. Such as @GET, @POST, @PUT, @DELETE.

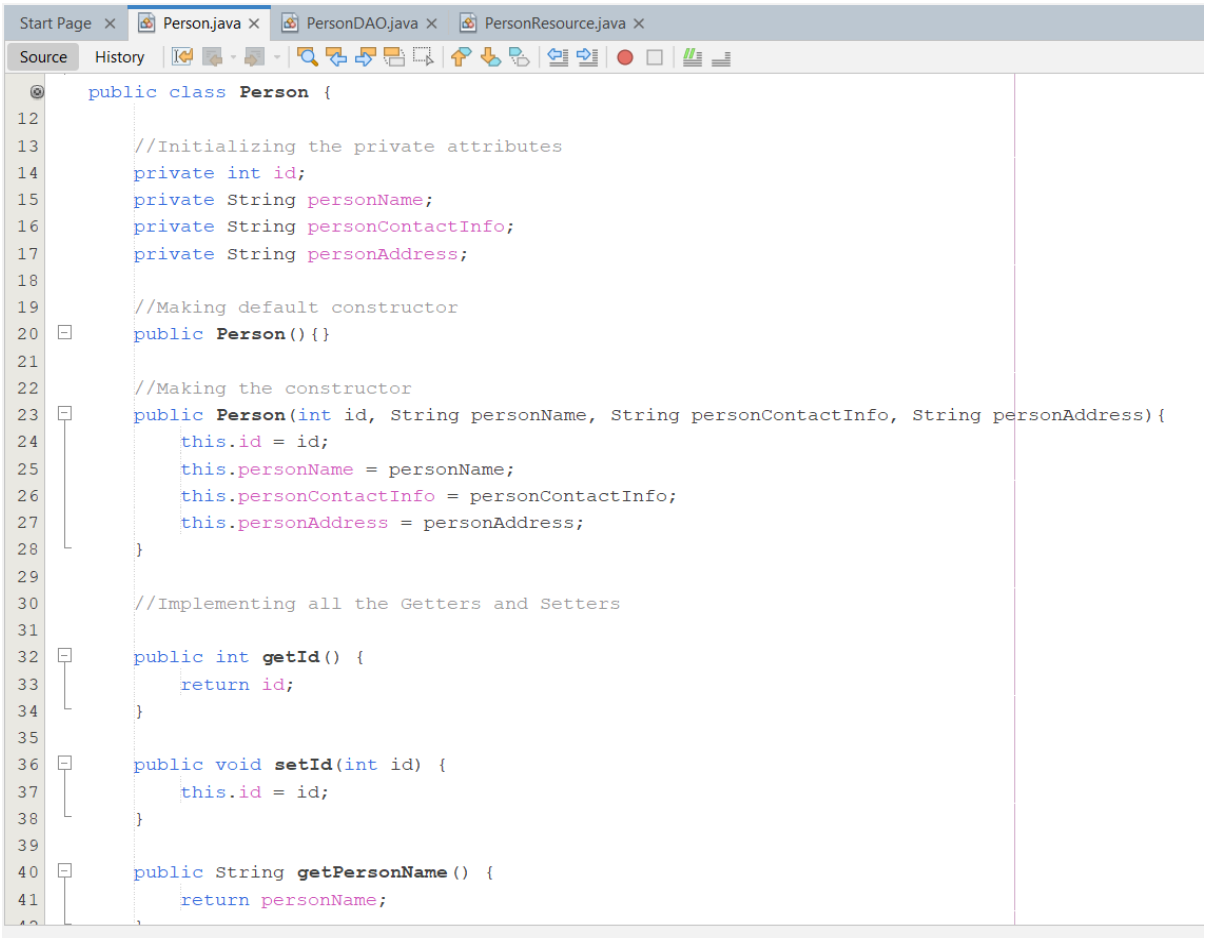


Figure 2:Code snippet for Person model class

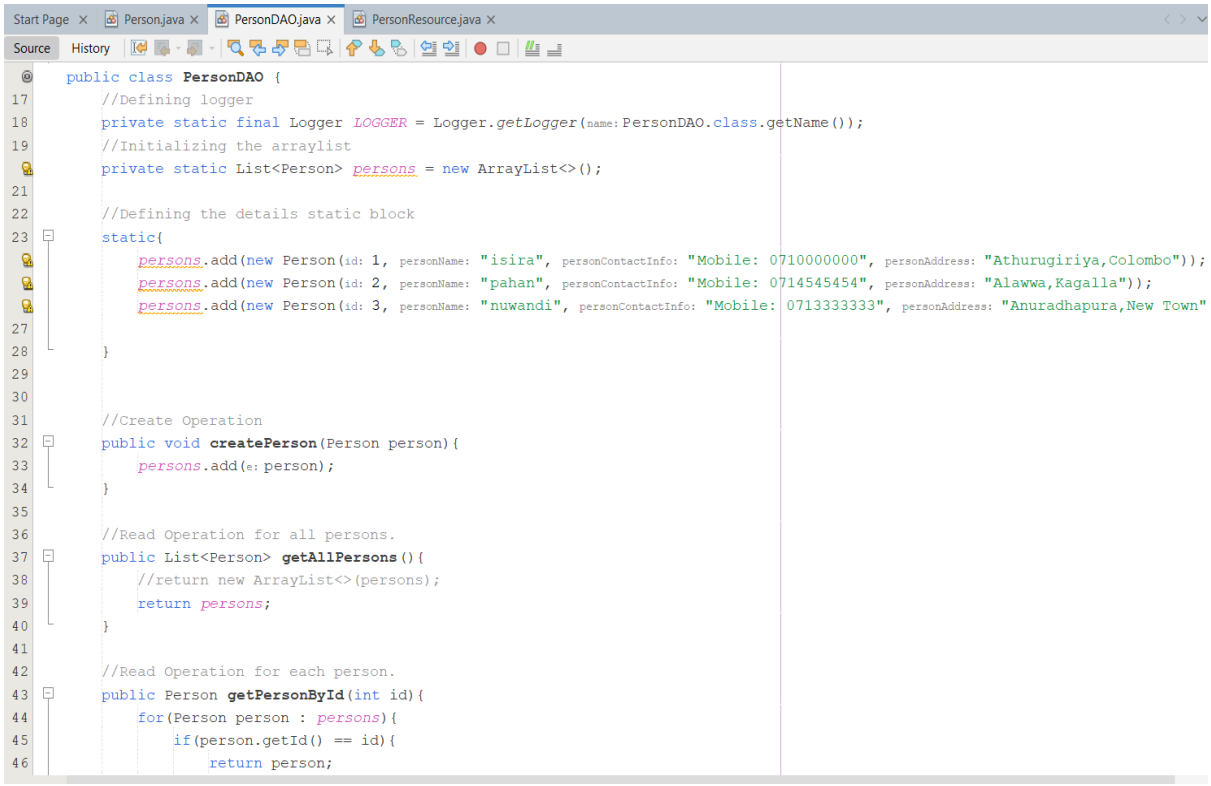


Figure 3: Code snippet for PersonDAO class

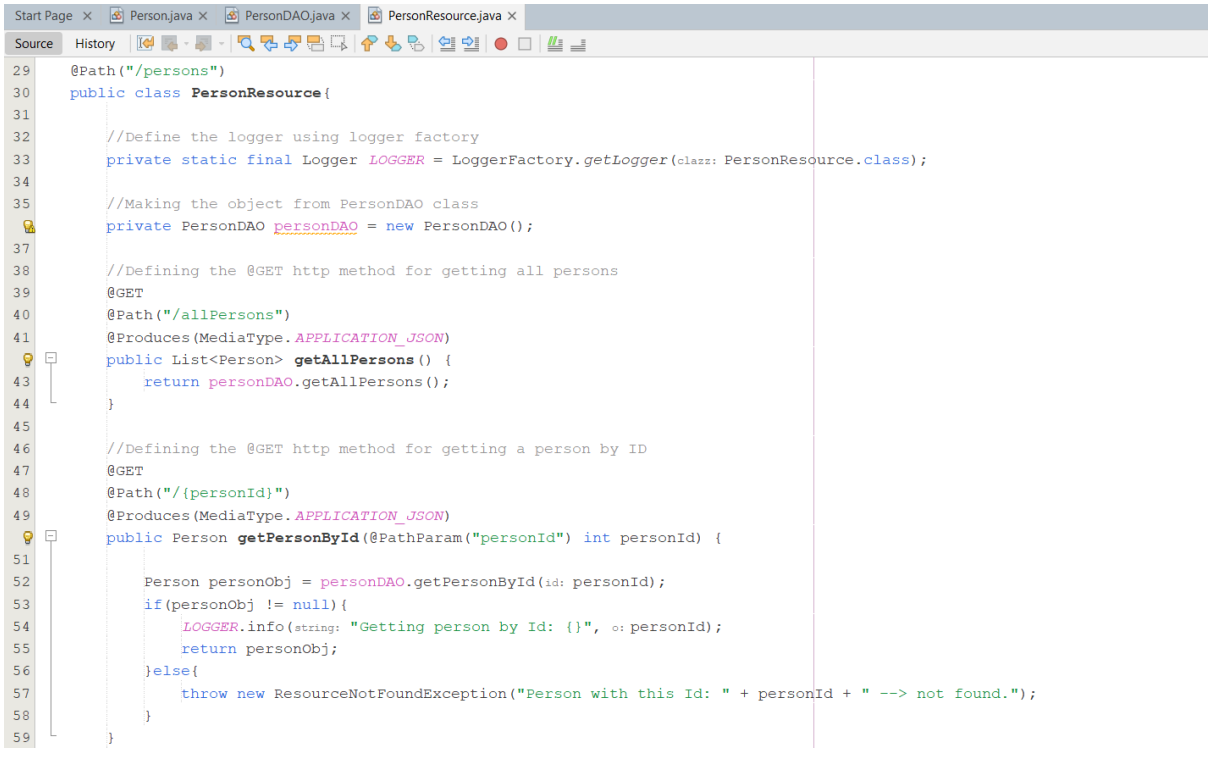


Figure 4: Code snippet for PersonResource class with @GET method

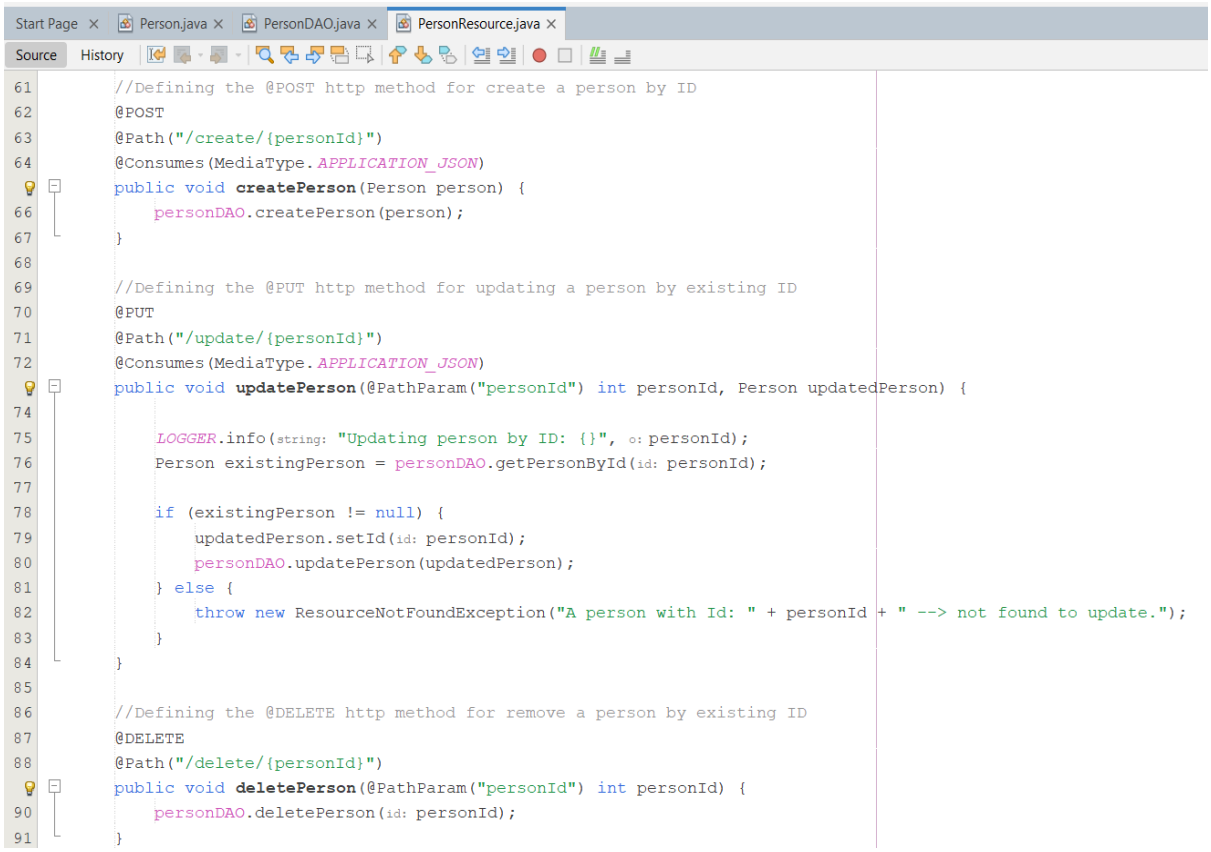
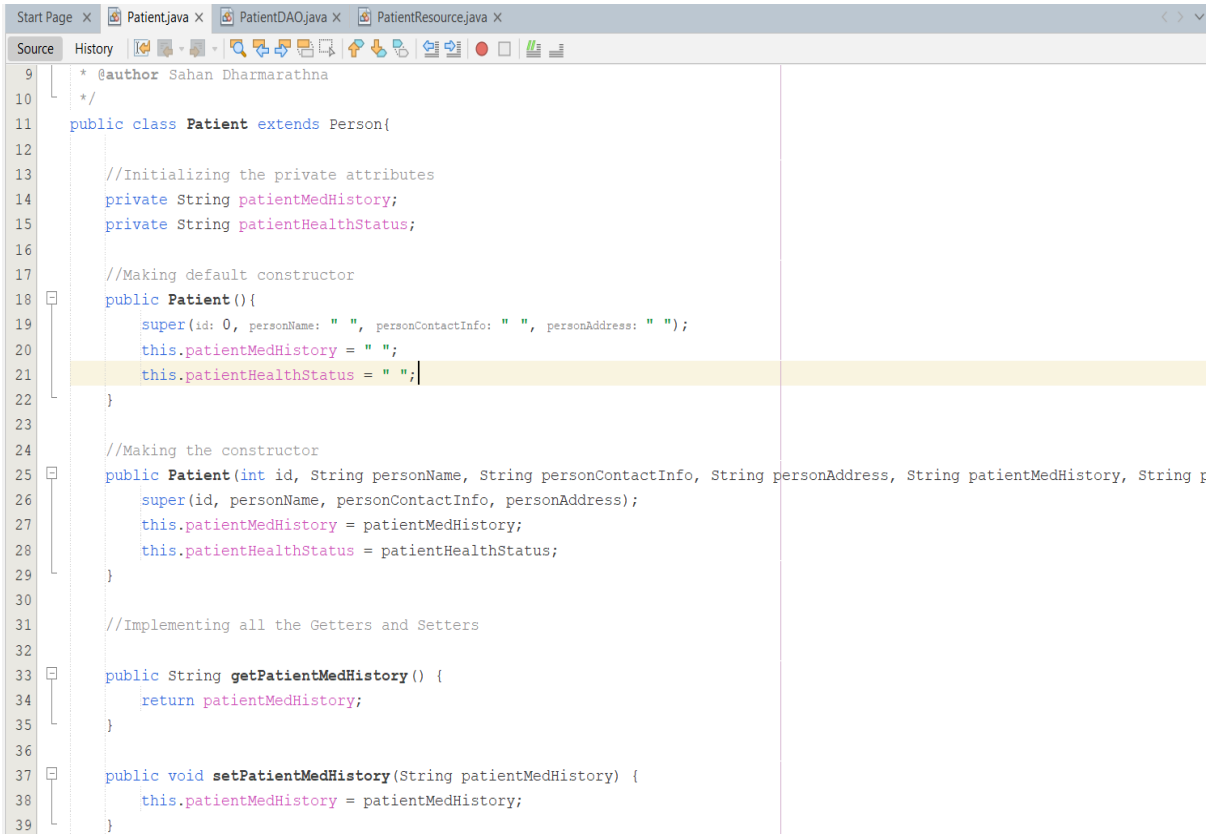


Figure 5: Code snippet for PersonResource class with @POST, @PUT & @DELETE methods

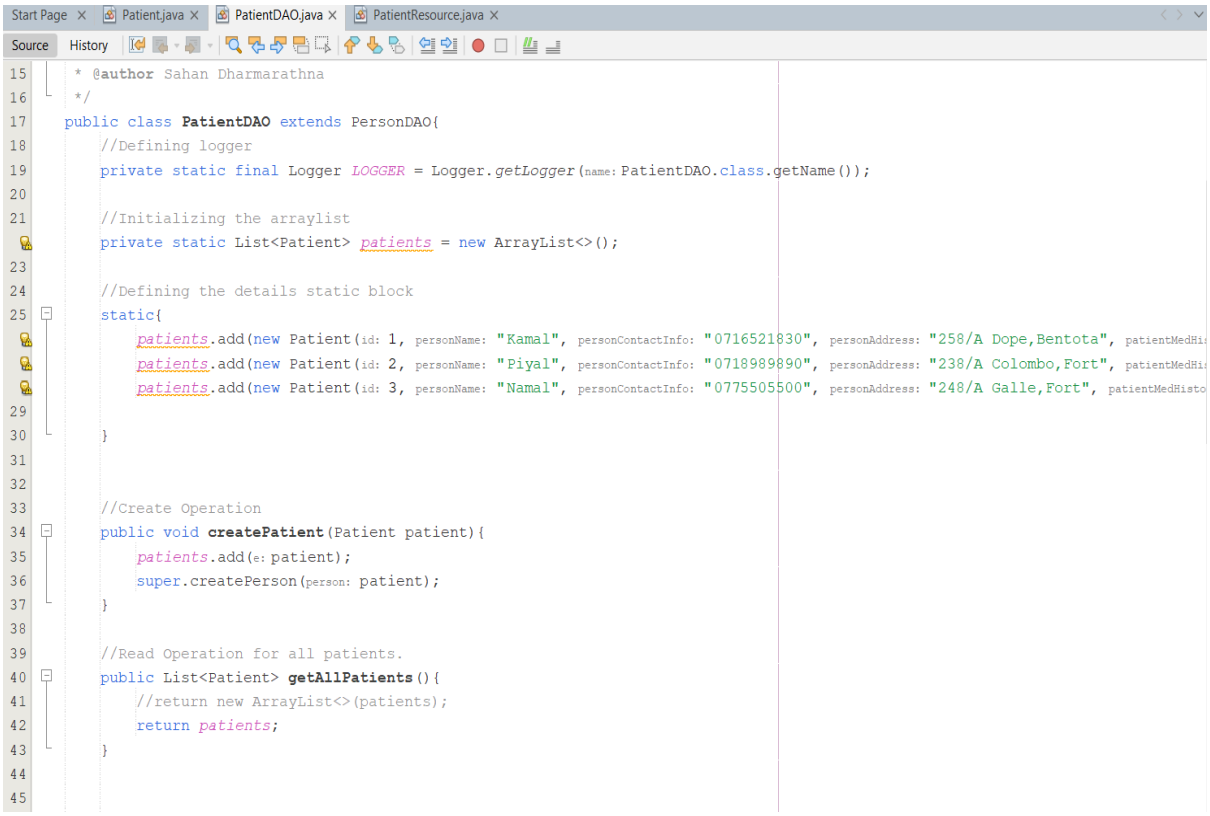
Patient, PatientDAO, and PatientResource

Patient class is child class which inherited from the Person model class.This Patient model class also included with private attributes which are related to Patient class. According to that attributes, all the getters and setters were initialized in the Patient model class. Apart from that all the CRUD methods are done in the PatientDAO class by making createPatient(), getAllPatients(), getPatientById(), updatePeteint(), and deletePatient() methods. And the RESTful HTTP methods which @GET, @POST, @PUT, @DELETE are implemented in the PatientResource class.



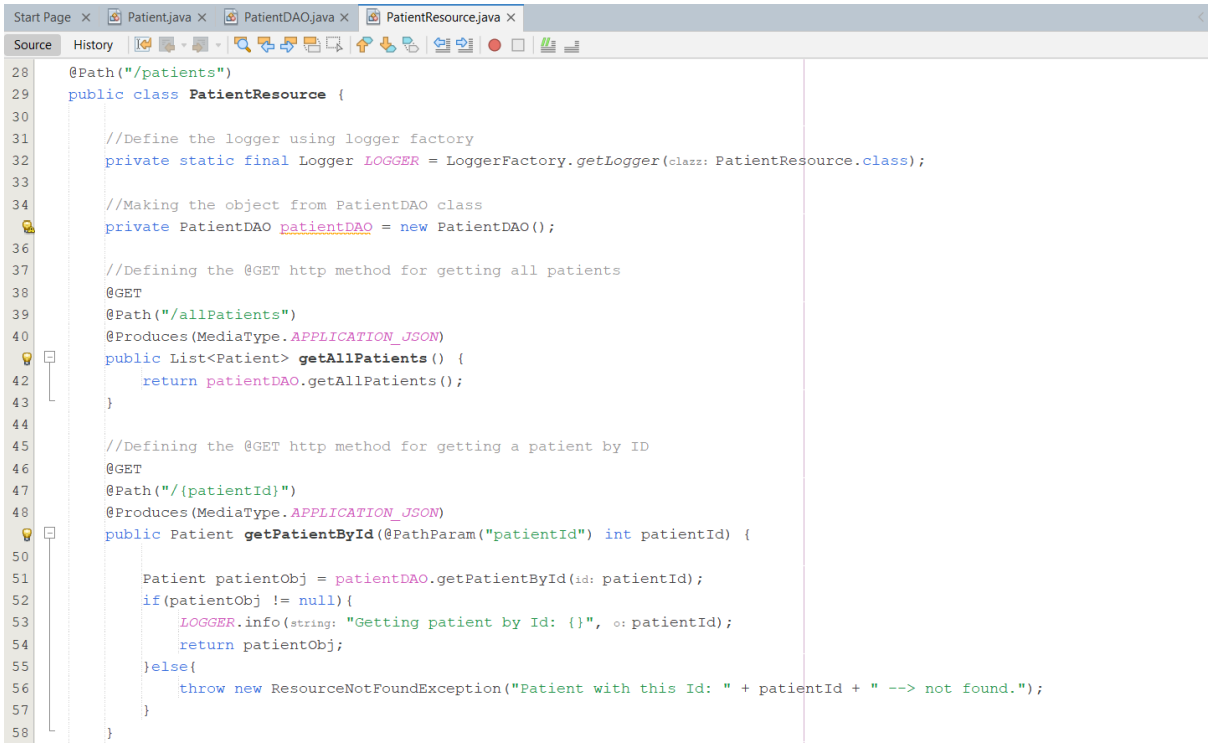
```java
/**
 * @author Sahan Dharmarathna
 */
public class Patient extends Person{

    //Initializing the private attributes
    private String patientMedHistory;
    private String patientHealthStatus;

    //Making default constructor
    public Patient(){
        super(id: 0, personName: " ", personContactInfo: " ", personAddress: " ");
        this.patientMedHistory = " ";
        this.patientHealthStatus = " ";
    }

    //Making the constructor
    public Patient(int id, String personName, String personContactInfo, String personAddress, String patientMedHistory, String p
        super(id, personName, personContactInfo, personAddress);
        this.patientMedHistory = patientMedHistory;
        this.patientHealthStatus = patientHealthStatus;
    }

    //Implementing all the Getters and Setters

    public String getPatientMedHistory() {
        return patientMedHistory;
    }

    public void setPatientMedHistory(String patientMedHistory) {
        this.patientMedHistory = patientMedHistory;
    }
```

Figure 6: Code snippet for Patient model class



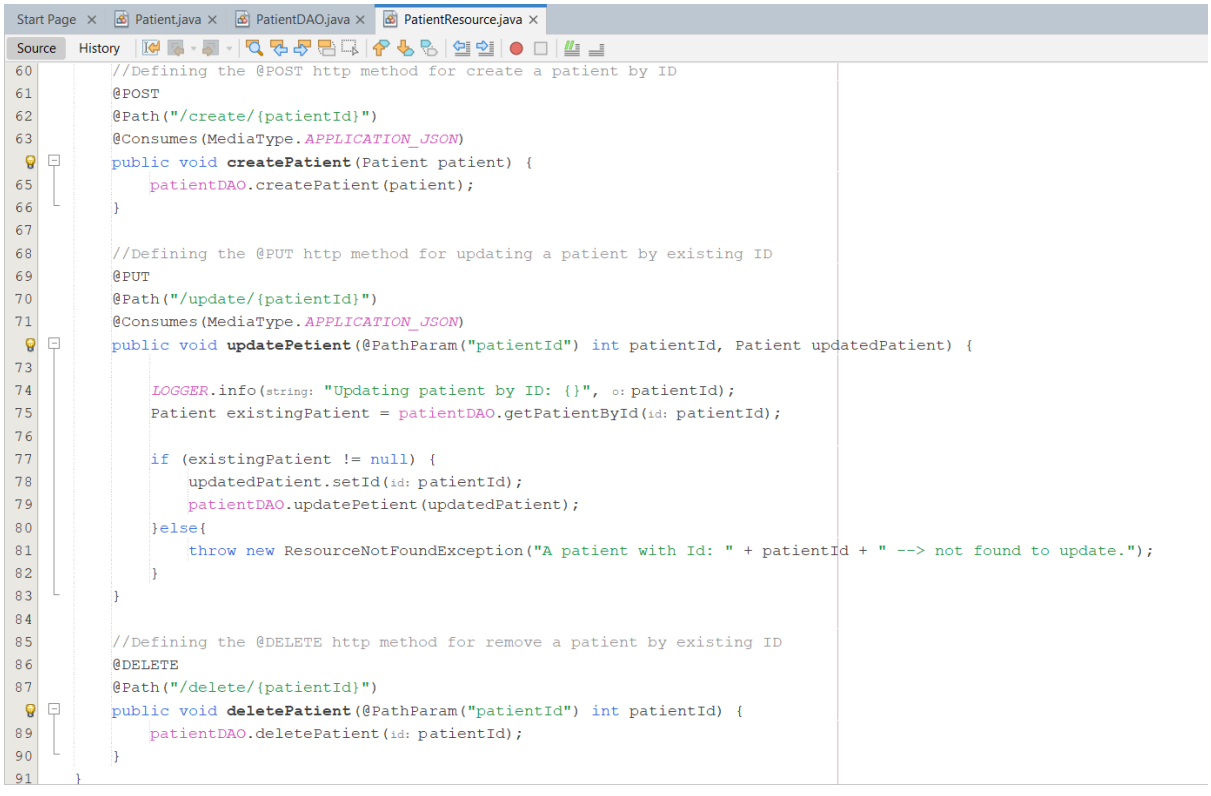
```java
/**
 * @author Sahan Dharmarathna
 */
public class PatientDAO extends PersonDAO{
    //Defining logger
    private static final Logger LOGGER = Logger.getLogger(name: PatientDAO.class.getName());

    //Initializing the arraylist
    private static List<Patient> patients = new ArrayList<>();

    //Defining the details static block
    static{
        patients.add(new Patient(id: 1, personName: "Kamal", personContactInfo: "0716521830", personAddress: "258/A Dope,Bentota",
        patients.add(new Patient(id: 2, personName: "Piyal", personContactInfo: "0718989890", personAddress: "238/A Colombo,Fort",
        patients.add(new Patient(id: 3, personName: "Namal", personContactInfo: "0775505500", personAddress: "248/A Galle,Fort",

    }


    //Create Operation
    public void createPatient(Patient patient){
        patients.add(e: patient);
        super.createPerson(person: patient);
    }

    //Read Operation for all patients.
    public List<Patient> getAllPatients(){
        //return new ArrayList<>(patients);
        return patients;
    }
```

Figure 7: Code snippet for PatientDAO class



```java
@Path("/patients")
public class PatientResource {

    //Define the logger using logger factory
    private static final Logger LOGGER = LoggerFactory.getLogger(clazz: PatientResource.class);

    //Making the object from PatientDAO class
    private PatientDAO patientDAO = new PatientDAO();

    //Defining the @GET http method for getting all patients
    @GET
    @Path("/allPatients")
    @Produces(MediaType.APPLICATION_JSON)
    public List<Patient> getAllPatients() {
        return patientDAO.getAllPatients();
    }

    //Defining the @GET http method for getting a patient by ID
    @GET
    @Path("/{patientId}")
    @Produces(MediaType.APPLICATION_JSON)
    public Patient getPatientById(@PathParam("patientId") int patientId) {

        Patient patientObj = patientDAO.getPatientById(id: patientId);
        if(patientObj != null){
            LOGGER.info(string: "Getting patient by Id: {}", o: patientId);
            return patientObj;
        }else{
            throw new ResourceNotFoundException("Patient with this Id: " + patientId + " --> not found.");
        }
    }
```

Figure 8: Code snippet for PatientResource class with @GET method



```java
    //Defining the @POST http method for create a patient by ID
    @POST
    @Path("/create/{patientId}")
    @Consumes(MediaType.APPLICATION_JSON)
    public void createPatient(Patient patient) {
        patientDAO.createPatient(patient);
    }

    //Defining the @PUT http method for updating a patient by existing ID
    @PUT
    @Path("/update/{patientId}")
    @Consumes(MediaType.APPLICATION_JSON)
    public void updatePetient(@PathParam("patientId") int patientId, Patient updatedPatient) {

        LOGGER.info(string: "Updating patient by ID: {}", o: patientId);
        Patient existingPatient = patientDAO.getPatientById(id: patientId);

        if (existingPatient != null) {
            updatedPatient.setId(id: patientId);
            patientDAO.updatePetient(updatedPatient);
        }else{
            throw new ResourceNotFoundException("A patient with Id: " + patientId + " --> not found to update.");
        }
    }

    //Defining the @DELETE http method for remove a patient by existing ID
    @DELETE
    @Path("/delete/{patientId}")
    public void deletePatient(@PathParam("patientId") int patientId) {
        patientDAO.deletePatient(id: patientId);
    }
}
```

Figure 9: Code snippet for PatientResource class with@POST, @PUT & @DELETE methods

Doctor, DoctorDAO, and DoctorResource

The Doctor class is also an inherited child class from the Person model class. This also included with all the related private attributes and the relevant getters and setters methods. As well as createDoctor(), getAllDoctors, getDoctorById(), updateDoctor(), and deleteDoctor() methods are belong to the DoctorDAO class. And that methods are doing the CRUD operations perfectly. Apart from that all the RESTful HTTP methods are done in the DoctorResource class (@GET, @POST, @PUT, @DELETE).

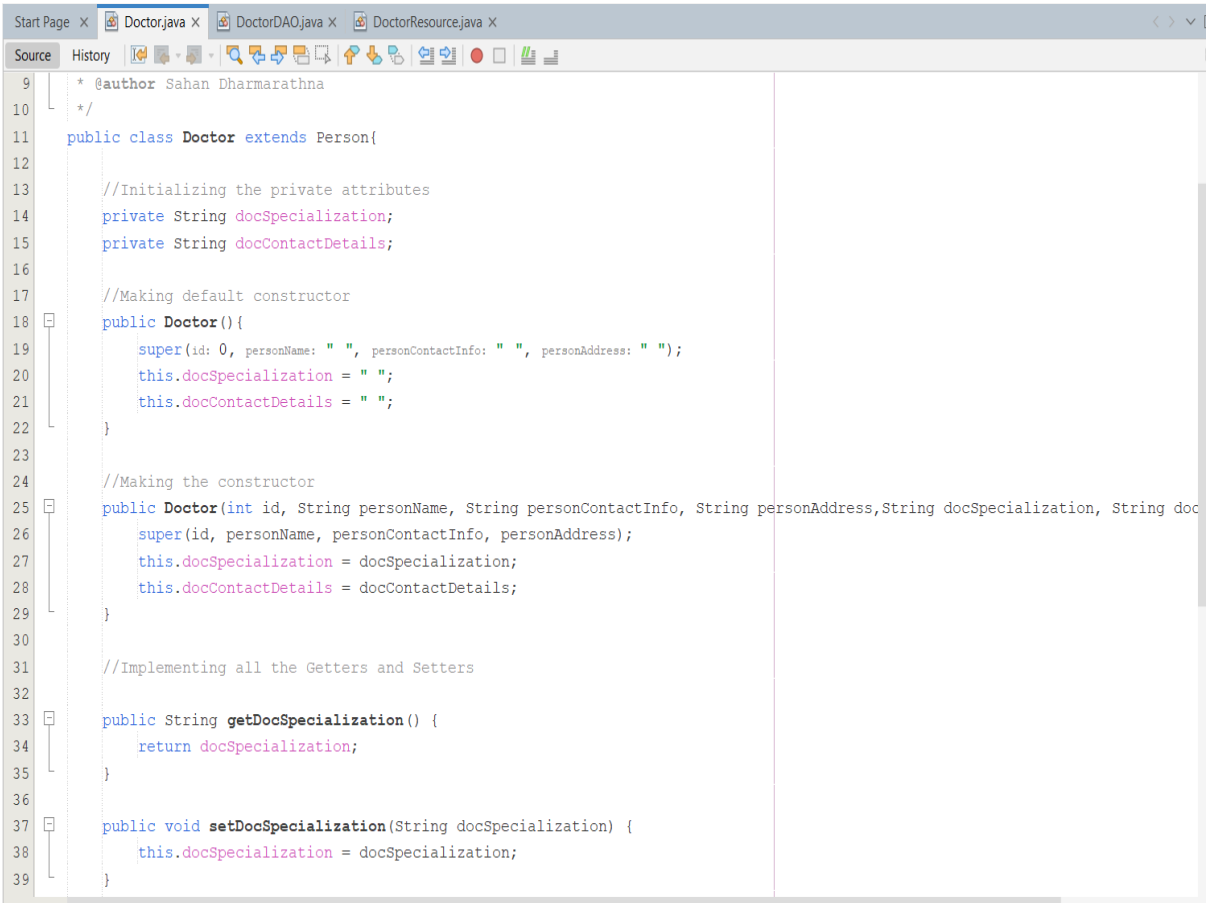
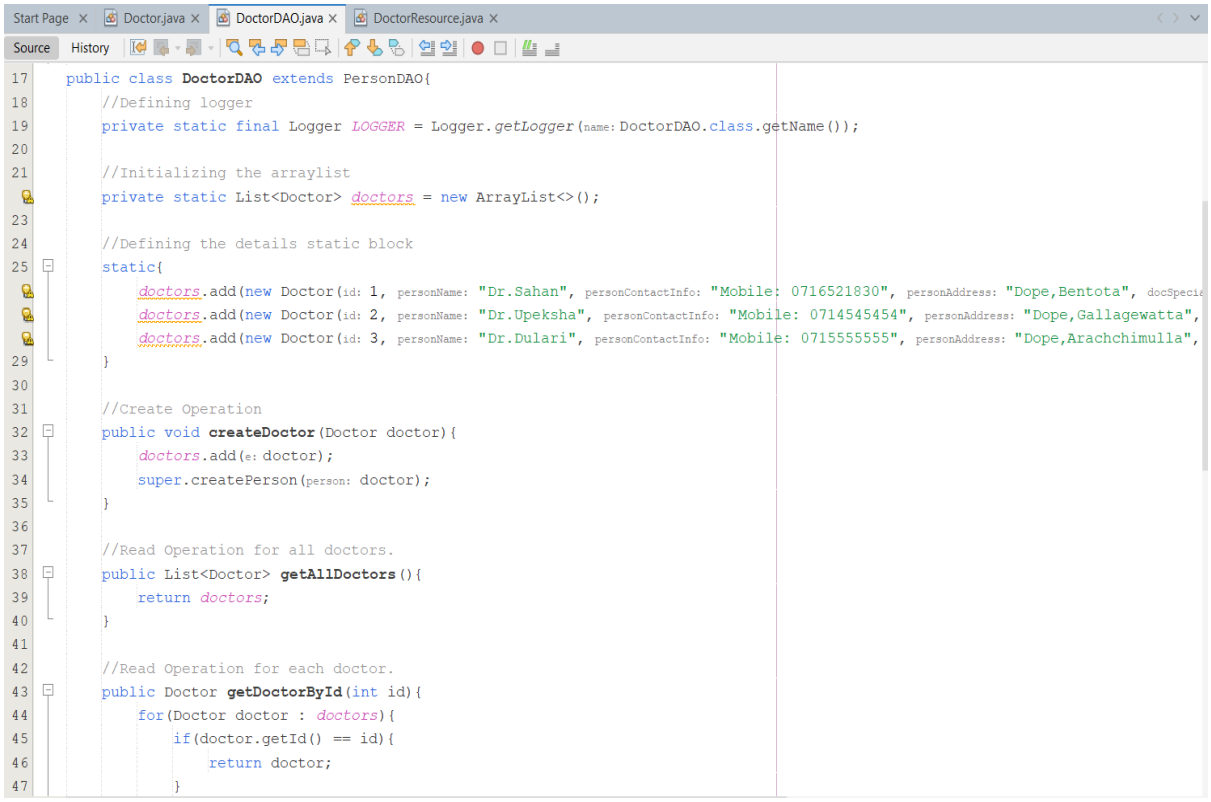
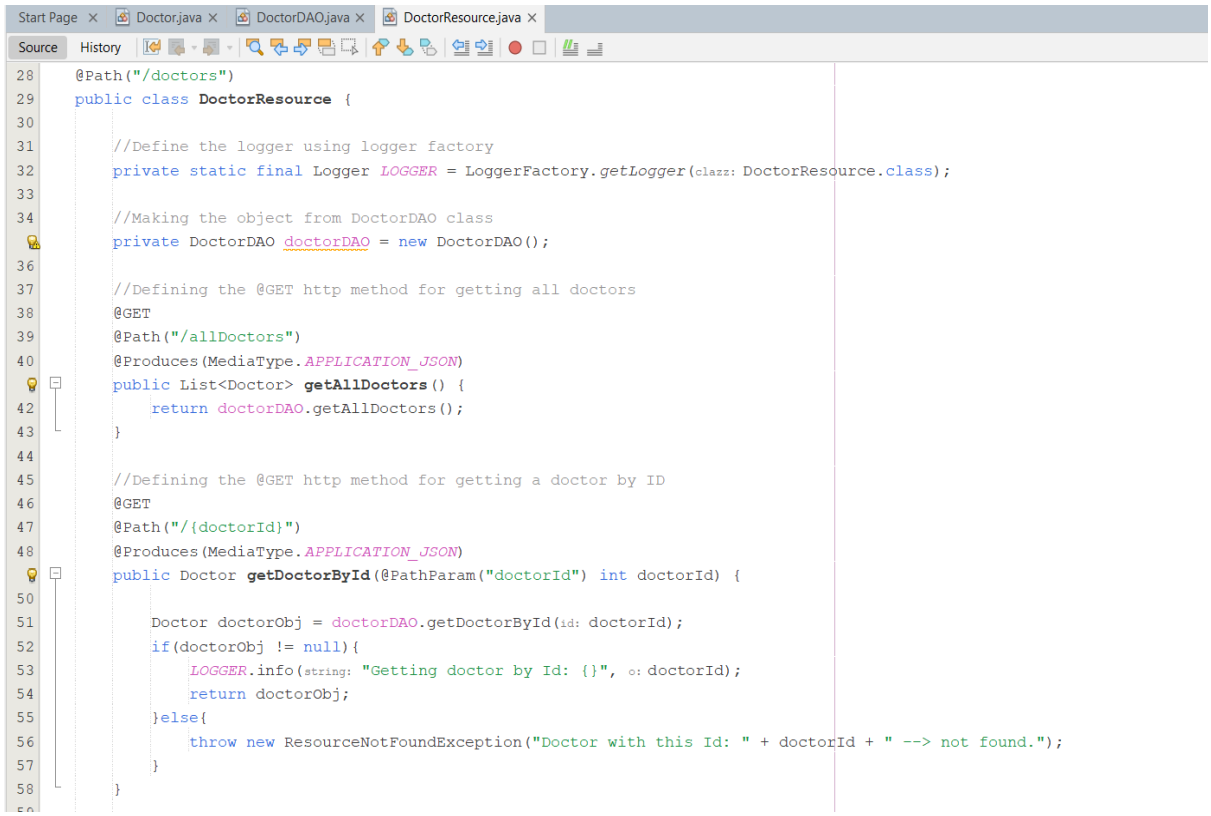


Figure 10: Code snippet for Doctor model class



```
Start Page | Doctor.java | DoctorDAO.java | DoctorResource.java
Source | History

17  public class DoctorDAO extends PersonDAO{
18      //Defining logger
19      private static final Logger LOGGER = Logger.getLogger(name: DoctorDAO.class.getName());
20
21      //Initializing the arraylist
22      private static List<Doctor> doctors = new ArrayList<>();
23
24      //Defining the details static block
25      static{
            doctors.add(new Doctor(id: 1, personName: "Dr.Sahan", personContactInfo: "Mobile: 0716521830", personAddress: "Dope,Bentota", docSpecia
            doctors.add(new Doctor(id: 2, personName: "Dr.Upeksha", personContactInfo: "Mobile: 0714545454", personAddress: "Dope,Gallagewatta",
            doctors.add(new Doctor(id: 3, personName: "Dr.Dulari", personContactInfo: "Mobile: 0715555555", personAddress: "Dope,Arachchimulla",
29      }
30
31      //Create Operation
32      public void createDoctor(Doctor doctor){
33          doctors.add(e: doctor);
34          super.createPerson(person: doctor);
35      }
36
37      //Read Operation for all doctors.
38      public List<Doctor> getAllDoctors(){
39          return doctors;
40      }
41
42      //Read Operation for each doctor.
43      public Doctor getDoctorById(int id){
44          for(Doctor doctor : doctors){
45              if(doctor.getId() == id){
46                  return doctor;
47              }
```

Figure 11: Code snippet for DoctorDAO class



```
Start Page | Doctor.java | DoctorDAO.java | DoctorResource.java
Source | History

28  @Path("/doctors")
29  public class DoctorResource {
30
31      //Define the logger using logger factory
32      private static final Logger LOGGER = LoggerFactory.getLogger(clazz: DoctorResource.class);
33
34      //Making the object from DoctorDAO class
35      private DoctorDAO doctorDAO = new DoctorDAO();
36
37      //Defining the @GET http method for getting all doctors
38      @GET
39      @Path("/allDoctors")
40      @Produces(MediaType.APPLICATION_JSON)
41      public List<Doctor> getAllDoctors() {
42          return doctorDAO.getAllDoctors();
43      }
44
45      //Defining the @GET http method for getting a doctor by ID
46      @GET
47      @Path("/{doctorId}")
48      @Produces(MediaType.APPLICATION_JSON)
49      public Doctor getDoctorById(@PathParam("doctorId") int doctorId) {
50
51          Doctor doctorObj = doctorDAO.getDoctorById(id: doctorId);
52          if(doctorObj != null){
53              LOGGER.info(string: "Getting doctor by Id: {}", o: doctorId);
54              return doctorObj;
55          }else{
56              throw new ResourceNotFoundException("Doctor with this Id: " + doctorId + " --> not found.");
57          }
58      }
```

Figure 12: Code snippet for DoctorResource class with @GET method

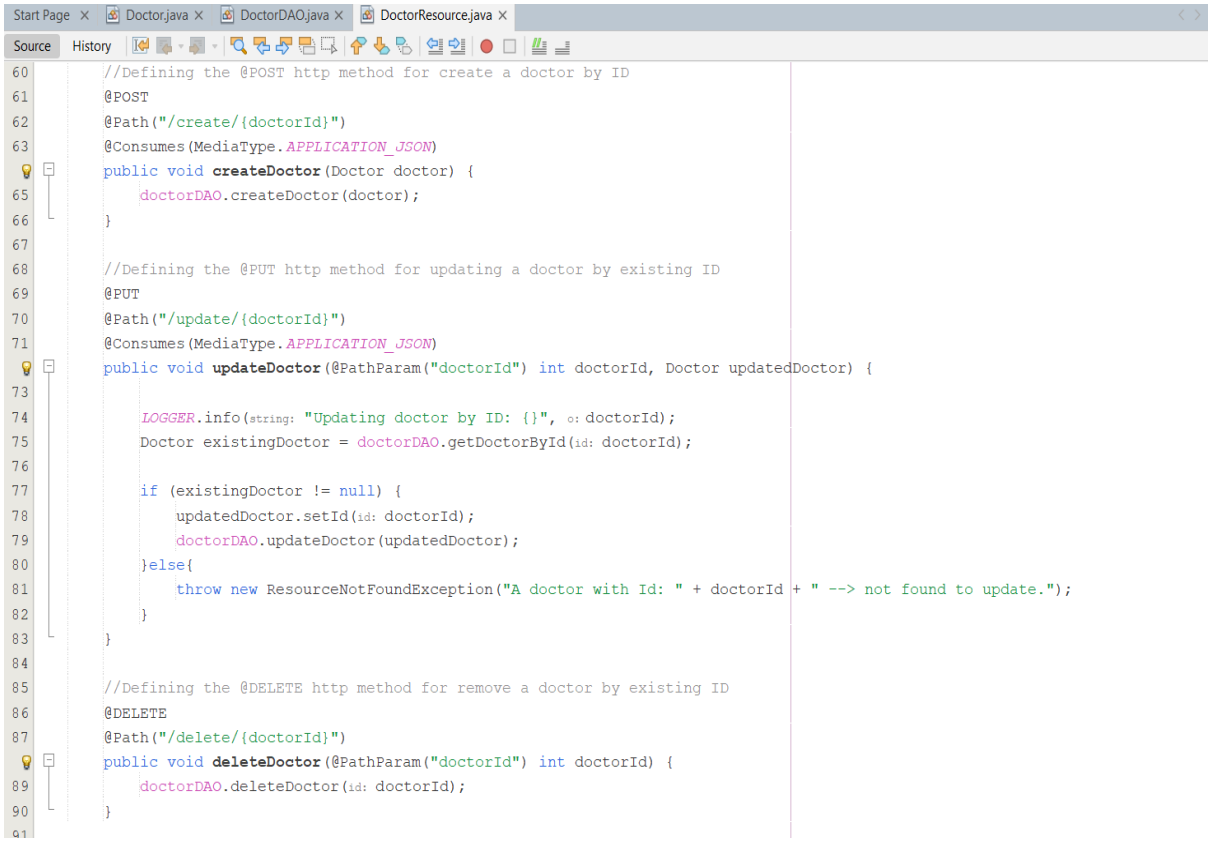


Figure 13: Code snippet for DoctorResource class with @POST, @PUT, & @DELETE methods

Appointment, AppointmentDAO, AppointmentResource

The appointment class is a model class which combines with both doctor and patient classes. In that class there are private attributes initialized and all the getters and setter methods are defined. Also the CRUD operations are done in the AppointmentDAO class in sequence createAppointment(), getAllAppointments(), getAppointmentById(), updateAppointment, and deleteAppointment(). Apart from that RESTful HTTP methods are don the AppointmentResource class(@GET, @POST, @PUT, @DELETE).

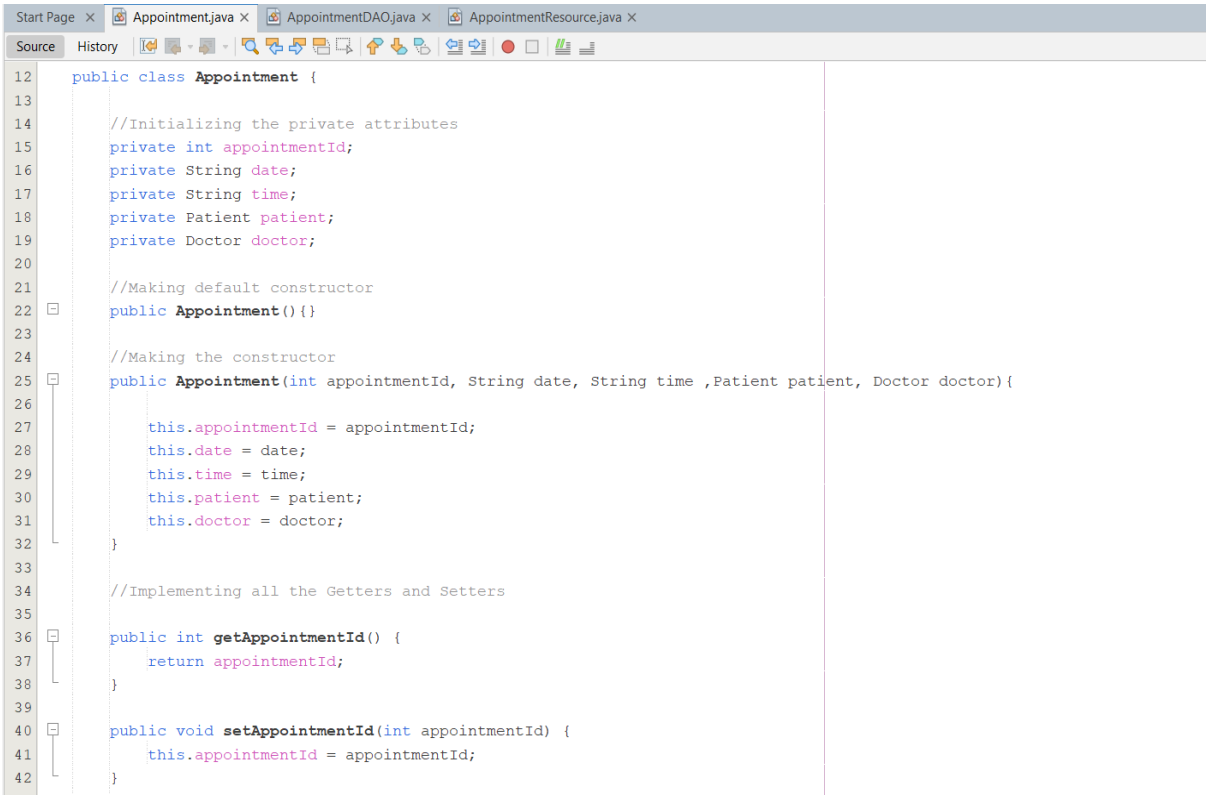


Figure 14: Code snippet for Appointment model class

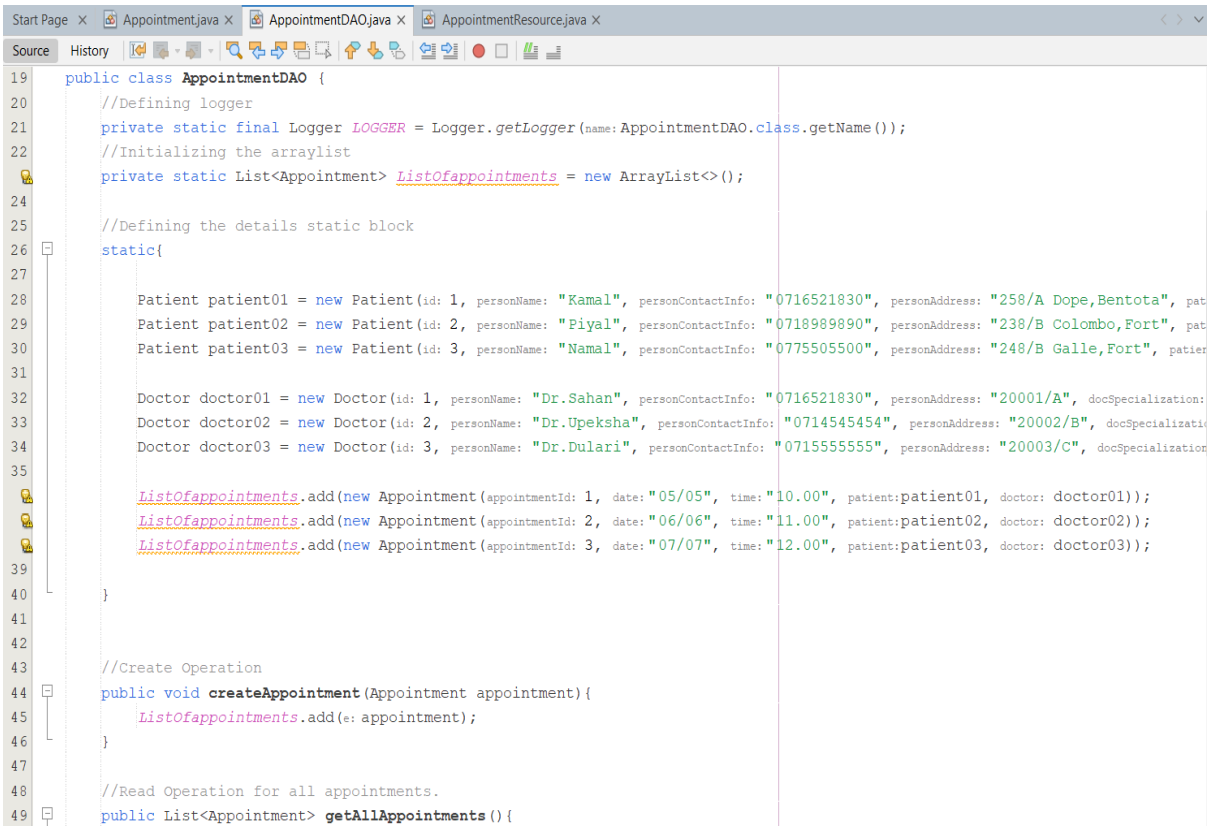
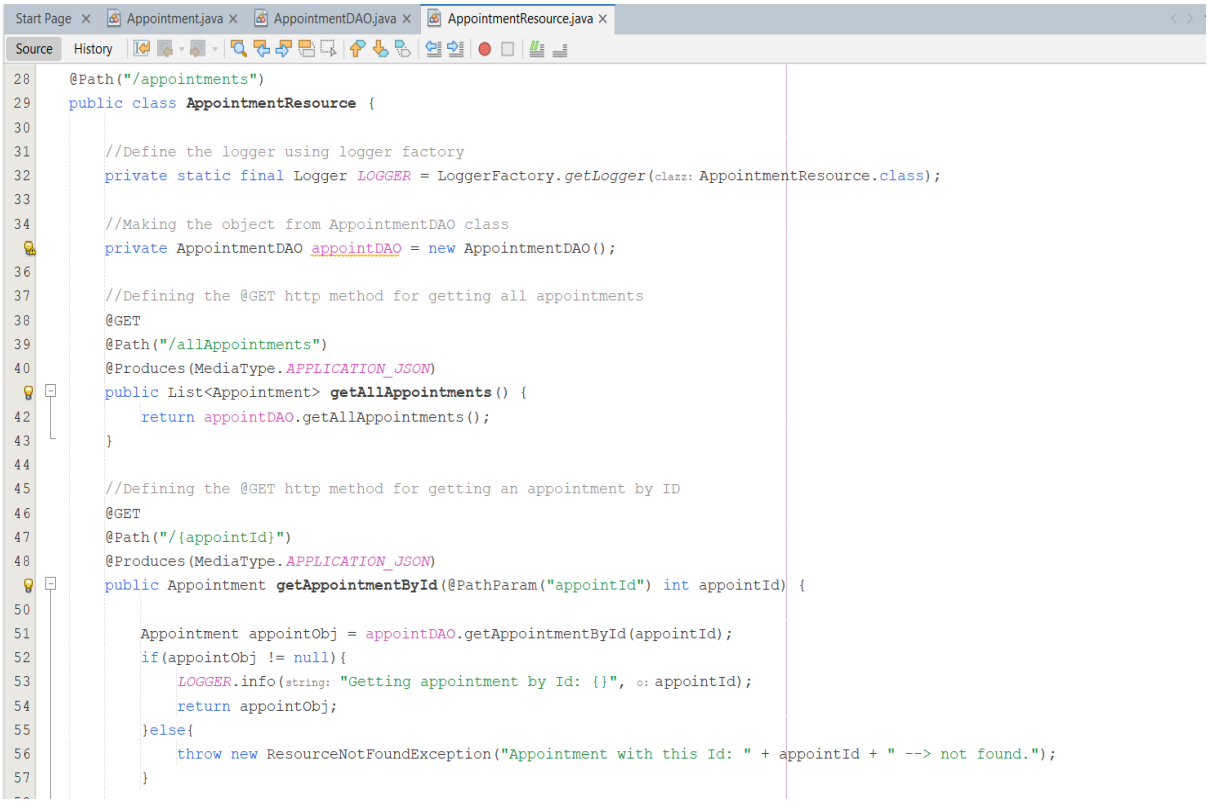
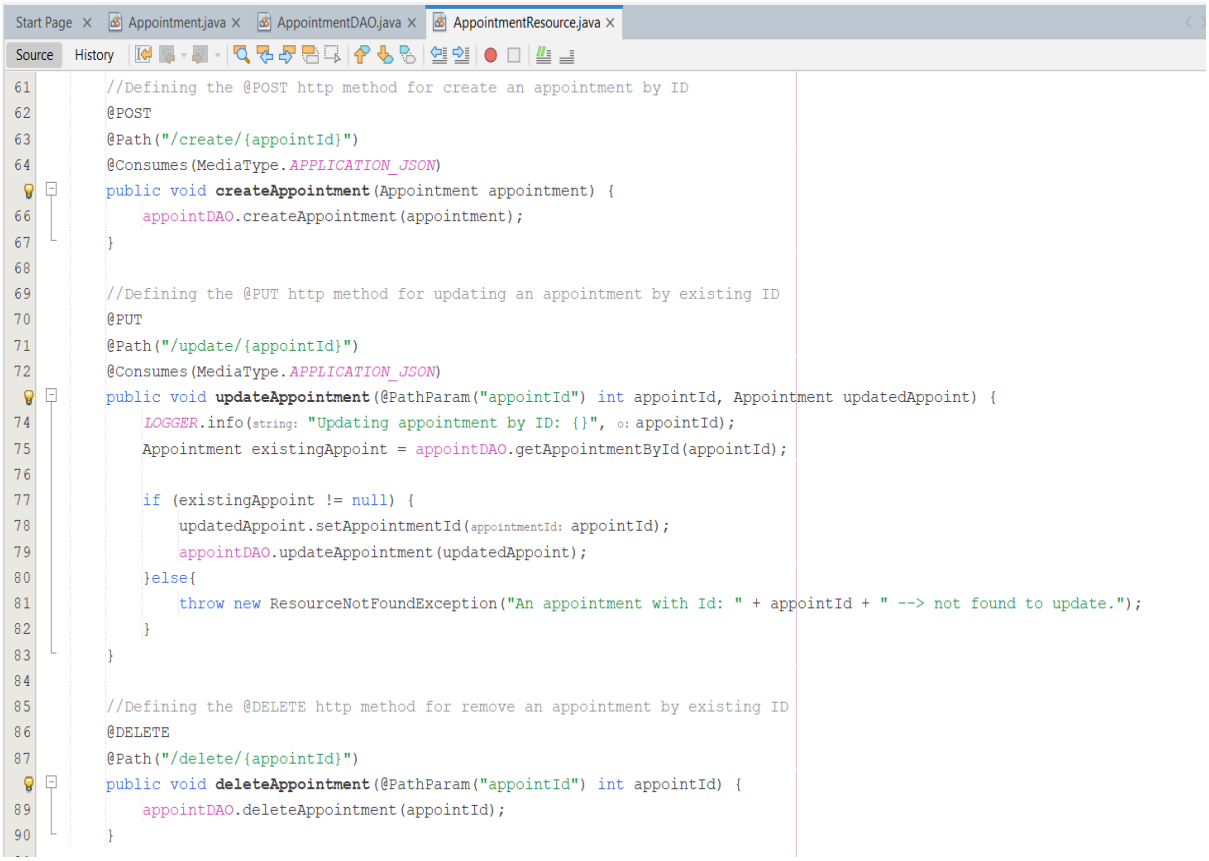


Figure 15: Code snippet for AppointmentDAO class



```
Start Page | Appointment.java | AppointmentDAO.java | AppointmentResource.java
Source | History
28  @Path("/appointments")
29  public class AppointmentResource {
30
31      //Define the logger using logger factory
32      private static final Logger LOGGER = LoggerFactory.getLogger(clazz: AppointmentResource.class);
33
34      //Making the object from AppointmentDAO class
35      private AppointmentDAO appointDAO = new AppointmentDAO();
36
37      //Defining the @GET http method for getting all appointments
38      @GET
39      @Path("/allAppointments")
40      @Produces(MediaType.APPLICATION_JSON)
41      public List<Appointment> getAllAppointments() {
42          return appointDAO.getAllAppointments();
43      }
44
45      //Defining the @GET http method for getting an appointment by ID
46      @GET
47      @Path("/{appointId}")
48      @Produces(MediaType.APPLICATION_JSON)
49      public Appointment getAppointmentById(@PathParam("appointId") int appointId) {
50
51          Appointment appointObj = appointDAO.getAppointmentById(appointId);
52          if(appointObj != null){
53              LOGGER.info(string: "Getting appointment by Id: {}", o: appointId);
54              return appointObj;
55          }else{
56              throw new ResourceNotFoundException("Appointment with this Id: " + appointId + " --> not found.");
57          }
```

Figure 16: Code snippet for AppointmentResource class with @GET method



```
Start Page | Appointment.java | AppointmentDAO.java | AppointmentResource.java
Source | History
61      //Defining the @POST http method for create an appointment by ID
62      @POST
63      @Path("/create/{appointId}")
64      @Consumes(MediaType.APPLICATION_JSON)
65      public void createAppointment(Appointment appointment) {
66          appointDAO.createAppointment(appointment);
67      }
68
69      //Defining the @PUT http method for updating an appointment by existing ID
70      @PUT
71      @Path("/update/{appointId}")
72      @Consumes(MediaType.APPLICATION_JSON)
73      public void updateAppointment(@PathParam("appointId") int appointId, Appointment updatedAppoint) {
74          LOGGER.info(string: "Updating appointment by ID: {}", o: appointId);
75          Appointment existingAppoint = appointDAO.getAppointmentById(appointId);
76
77          if (existingAppoint != null) {
78              updatedAppoint.setAppointmentId(appointmentId: appointId);
79              appointDAO.updateAppointment(updatedAppoint);
80          }else{
81              throw new ResourceNotFoundException("An appointment with Id: " + appointId + " --> not found to update.");
82          }
83      }
84
85      //Defining the @DELETE http method for remove an appointment by existing ID
86      @DELETE
87      @Path("/delete/{appointId}")
88      public void deleteAppointment(@PathParam("appointId") int appointId) {
89          appointDAO.deleteAppointment(appointId);
90      }
```

Figure 17: Code snippet for AppointmentResource class with @POST, @PUT, & @DELETE methods

MedicalRecords, MedicalRecordsDAO, MedicalRecordsResource

Private attributes related to MedicalRecords class is initialized in the MedicalRecords model class. Also all the relevant getters and setters are defined in the MedicalRecords class. The MedicalRecordsDAO class is included with all the CRUD operations which is Create, Read, Update, and Delete. In sequence they are createMedicalRecord(), getAllMedicalRecords(), getMedicalRecordById(), updateMedicalRecord(), and deleteMedicalRecord(). Apart from that the PersonResource class is filled with relevant RESTful HTTP methods. Such as @GET, @POST, @PUT, @DELETE.

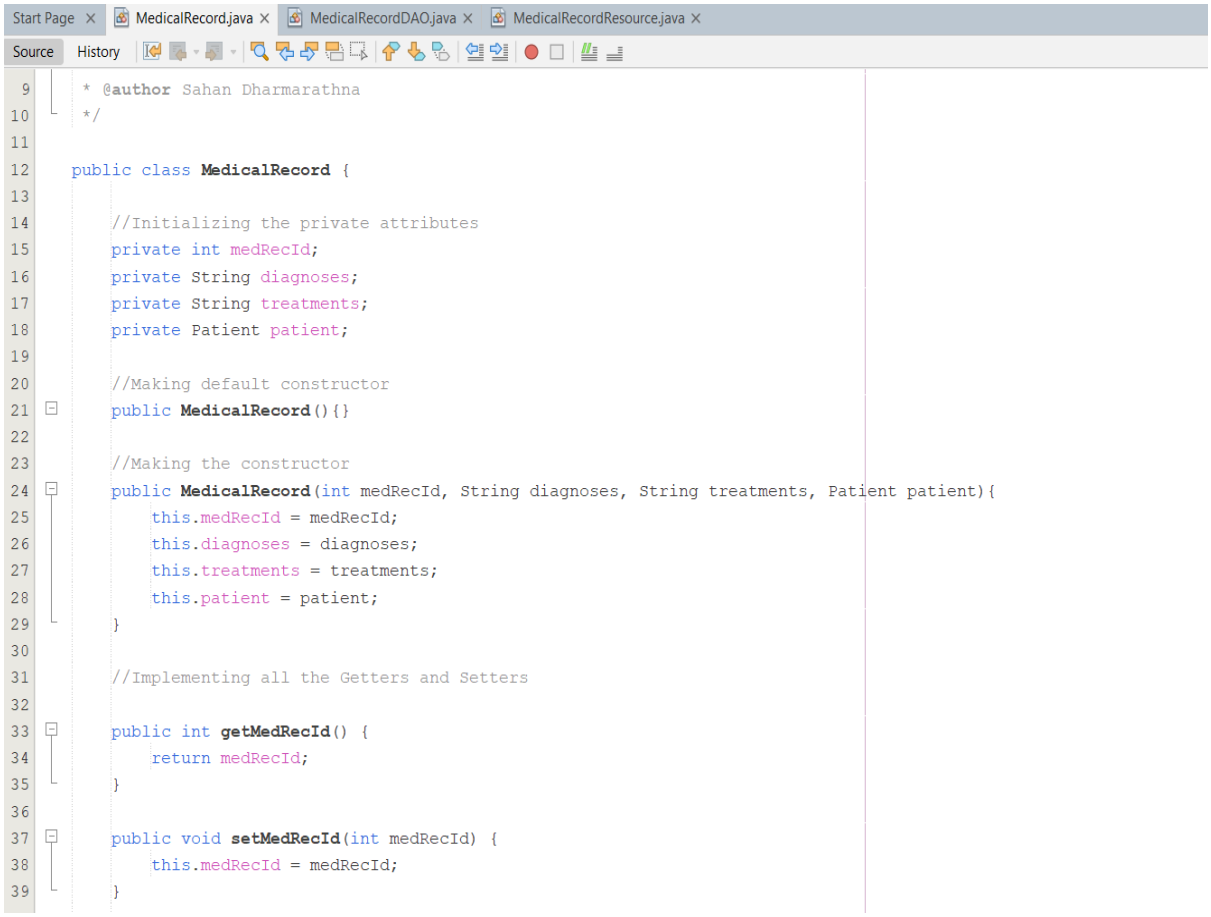
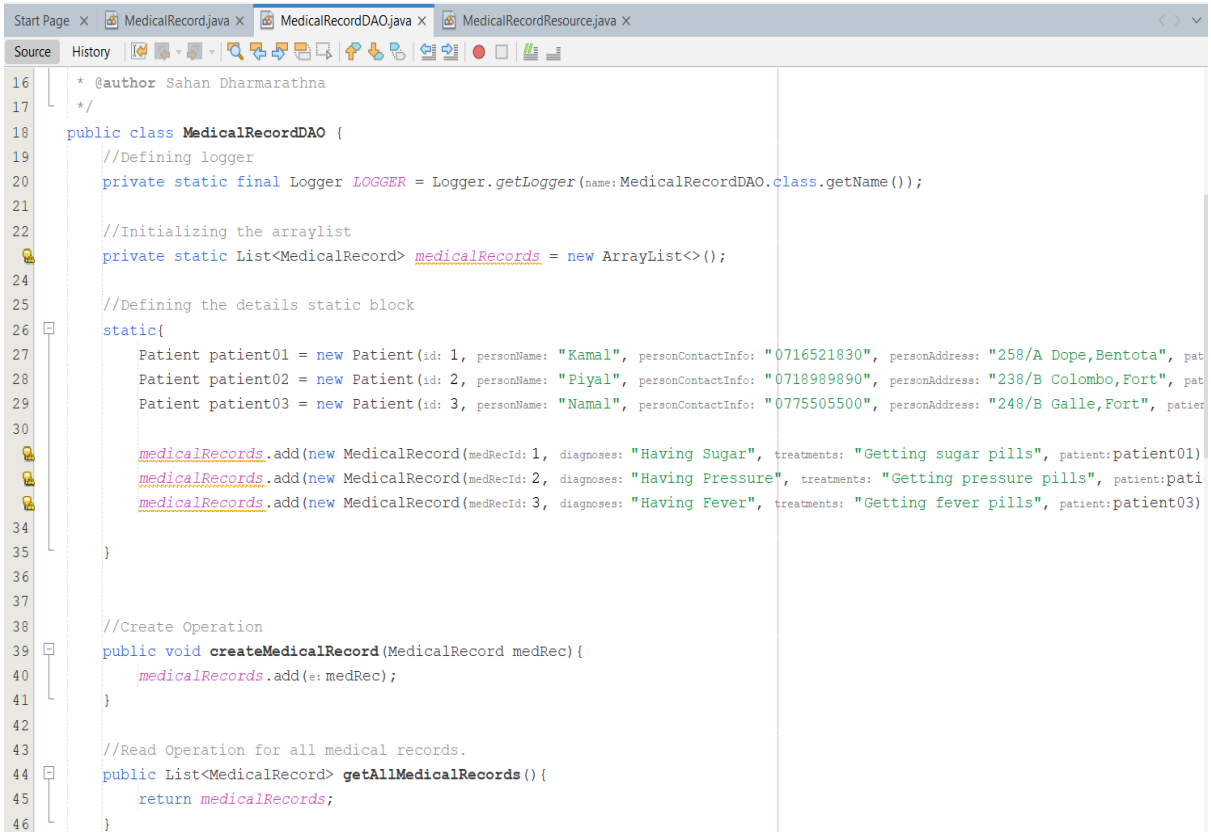
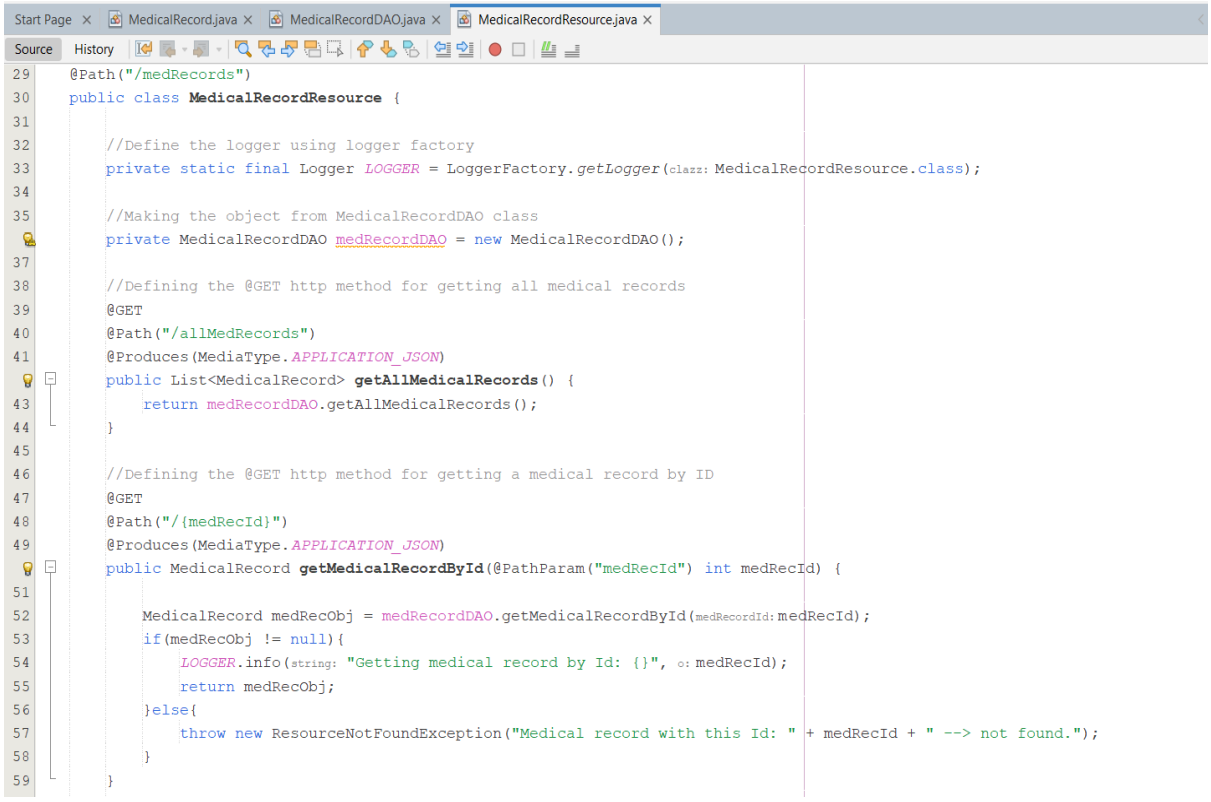


Figure 18: Code snippet for MedicalRecord model class



```
Start Page | MedicalRecord.java | MedicalRecordDAO.java | MedicalRecordResource.java
Source | History

 * @author Sahan Dharmarathna
 */
public class MedicalRecordDAO {
    //Defining logger
    private static final Logger LOGGER = Logger.getLogger(name: MedicalRecordDAO.class.getName());

    //Initializing the arraylist
    private static List<MedicalRecord> medicalRecords = new ArrayList<>();

    //Defining the details static block
    static{
        Patient patient01 = new Patient(id: 1, personName: "Kamal", personContactInfo: "0716521830", personAddress: "258/A Dope,Bentota", pat
        Patient patient02 = new Patient(id: 2, personName: "Piyal", personContactInfo: "0718989890", personAddress: "238/B Colombo,Fort", pat
        Patient patient03 = new Patient(id: 3, personName: "Namal", personContactInfo: "0775505500", personAddress: "248/B Galle,Fort", patier

        medicalRecords.add(new MedicalRecord(medRecId: 1, diagnoses: "Having Sugar", treatments: "Getting sugar pills", patient:patient01)
        medicalRecords.add(new MedicalRecord(medRecId: 2, diagnoses: "Having Pressure", treatments: "Getting pressure pills", patient:pati
        medicalRecords.add(new MedicalRecord(medRecId: 3, diagnoses: "Having Fever", treatments: "Getting fever pills", patient:patient03)

    }


    //Create Operation
    public void createMedicalRecord(MedicalRecord medRec){
        medicalRecords.add(e: medRec);
    }

    //Read Operation for all medical records.
    public List<MedicalRecord> getAllMedicalRecords(){
        return medicalRecords;
    }
```

Figure 19: Code snippet for MedicalRecordDAO class



```
Start Page | MedicalRecord.java | MedicalRecordDAO.java | MedicalRecordResource.java
Source | History

@Path("/medRecords")
public class MedicalRecordResource {

    //Define the logger using logger factory
    private static final Logger LOGGER = LoggerFactory.getLogger(clazz: MedicalRecordResource.class);

    //Making the object from MedicalRecordDAO class
    private MedicalRecordDAO medRecordDAO = new MedicalRecordDAO();

    //Defining the @GET http method for getting all medical records
    @GET
    @Path("/allMedRecords")
    @Produces(MediaType.APPLICATION_JSON)
    public List<MedicalRecord> getAllMedicalRecords() {
        return medRecordDAO.getAllMedicalRecords();
    }

    //Defining the @GET http method for getting a medical record by ID
    @GET
    @Path("/{medRecId}")
    @Produces(MediaType.APPLICATION_JSON)
    public MedicalRecord getMedicalRecordById(@PathParam("medRecId") int medRecId) {

        MedicalRecord medRecObj = medRecordDAO.getMedicalRecordById(medRecordId: medRecId);
        if(medRecObj != null){
            LOGGER.info(string: "Getting medical record by Id: {}", o: medRecId);
            return medRecObj;
        }else{
            throw new ResourceNotFoundException("Medical record with this Id: " + medRecId + " --> not found.");
        }
    }
```

Figure 20: Code snippet for MedicalRecordResource class with @GET method

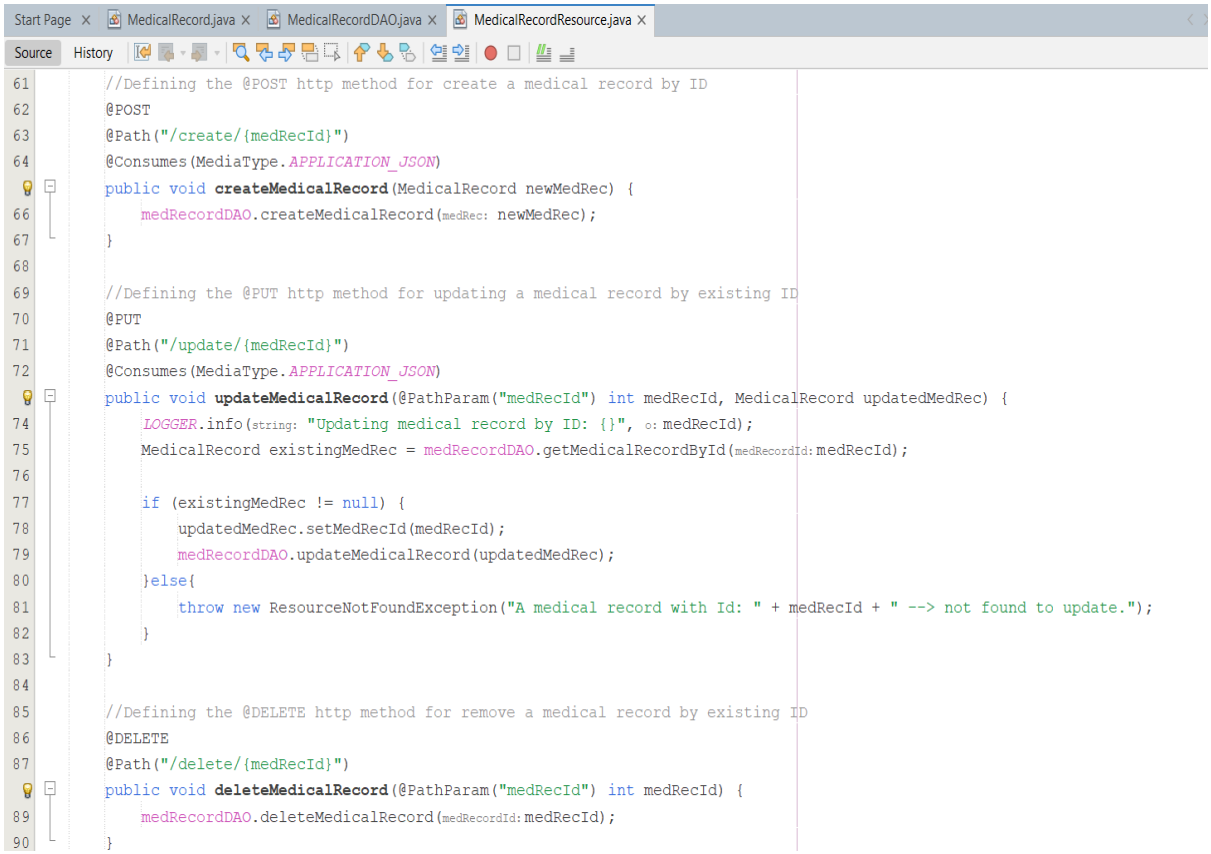


Figure 21: Code snippet for MedicalRecordResource class with @POST, @PUT, & @DELETE methods

Prescription, PrescriptionDAO, PrescriptionResource

This Prescription model class also included with private attributes which are related to Prescription class. According to that attributes, all the getters and setters were initialized in the Prescription model class. Apart from that all the CRUD methods are done in the PrescriptionDAO class by making createPrescription(), getAllPrescriptions(), getPrescriptionById(), updatePrescription(), and deletePrescription() methods. And the RESTful HTTP methods which @GET, @POST, @PUT, @DELETE are implemented in the PrescriptionResource class.

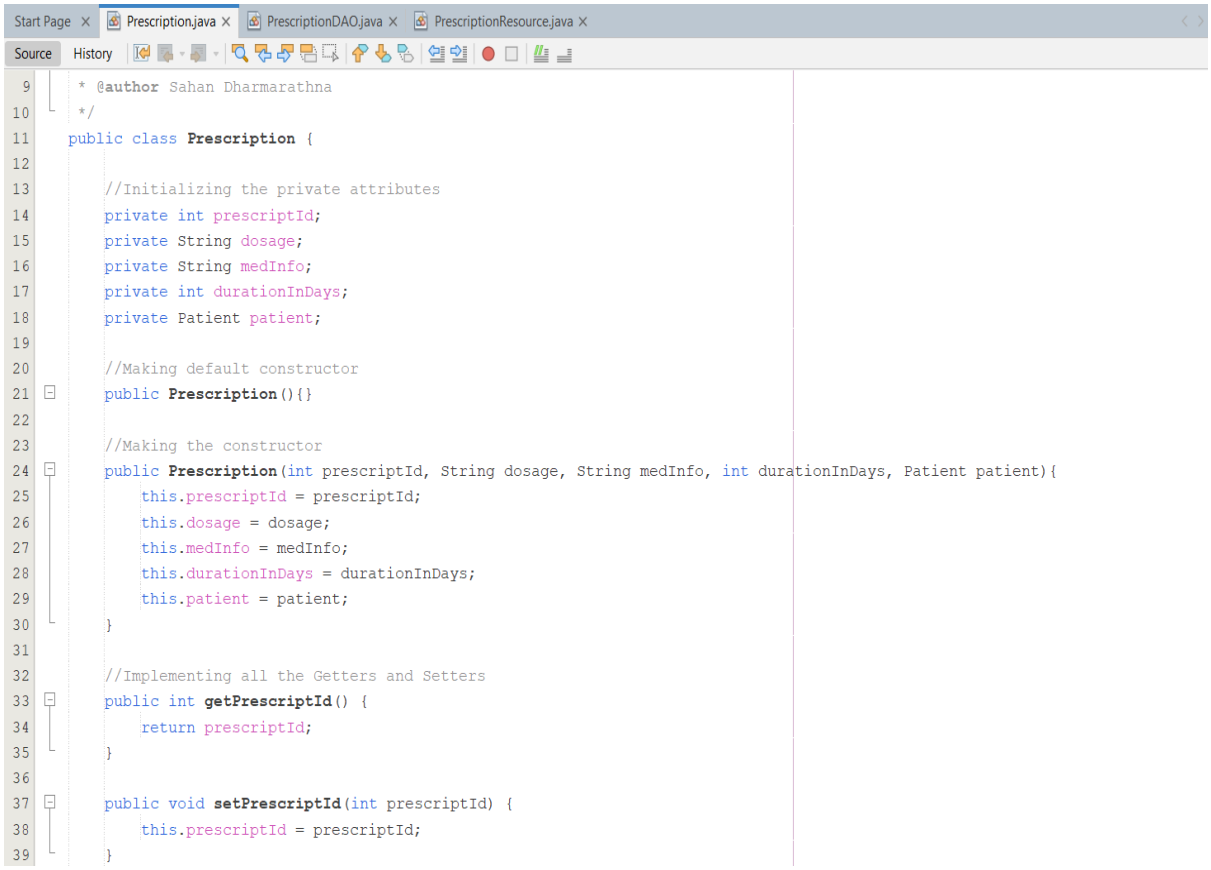


Figure 22: Code snippet for Prescription model class

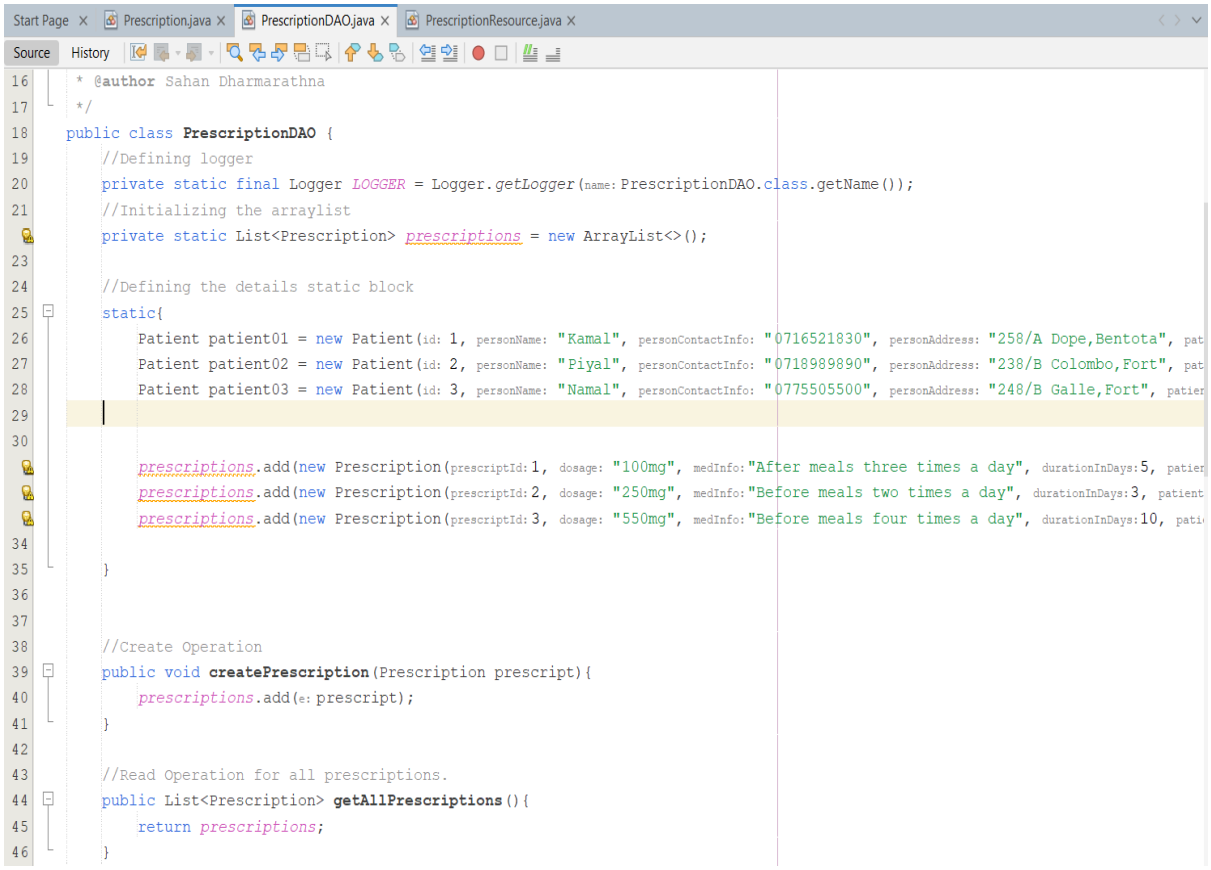
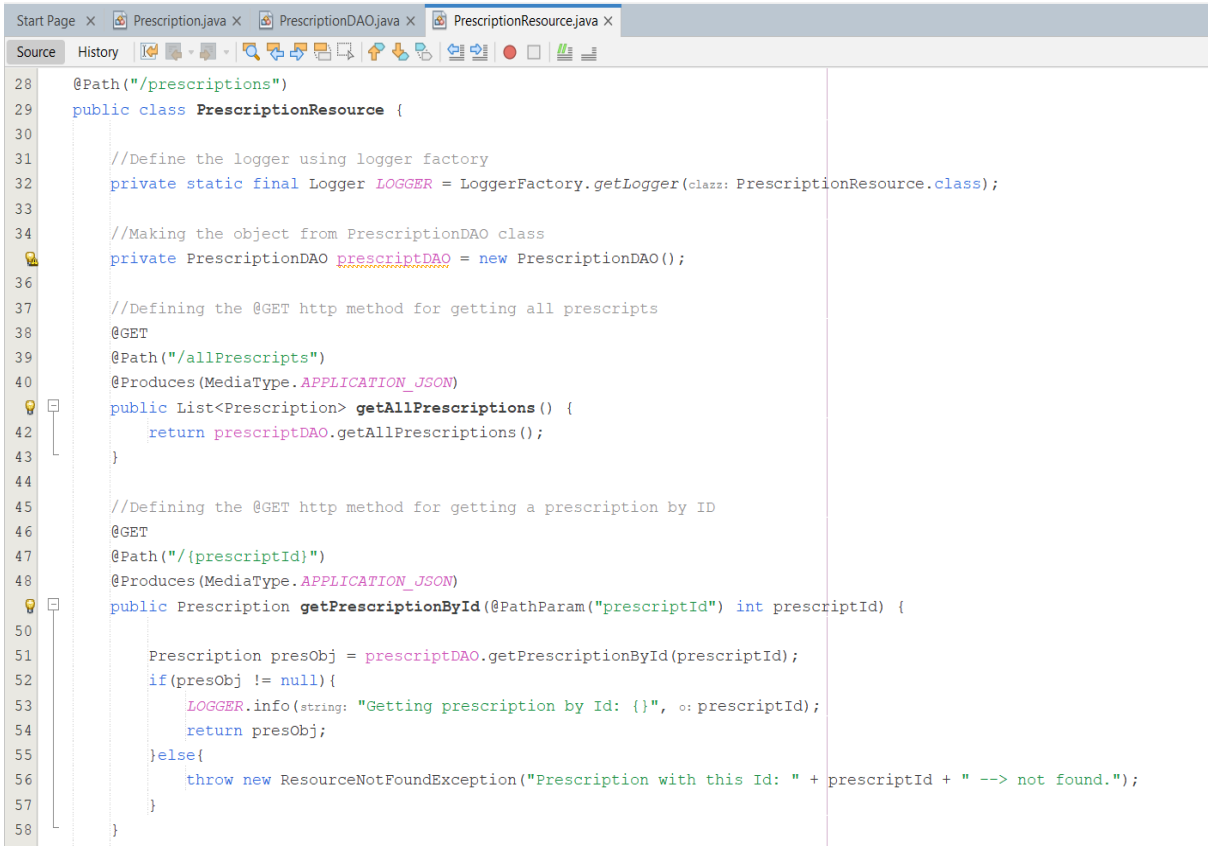
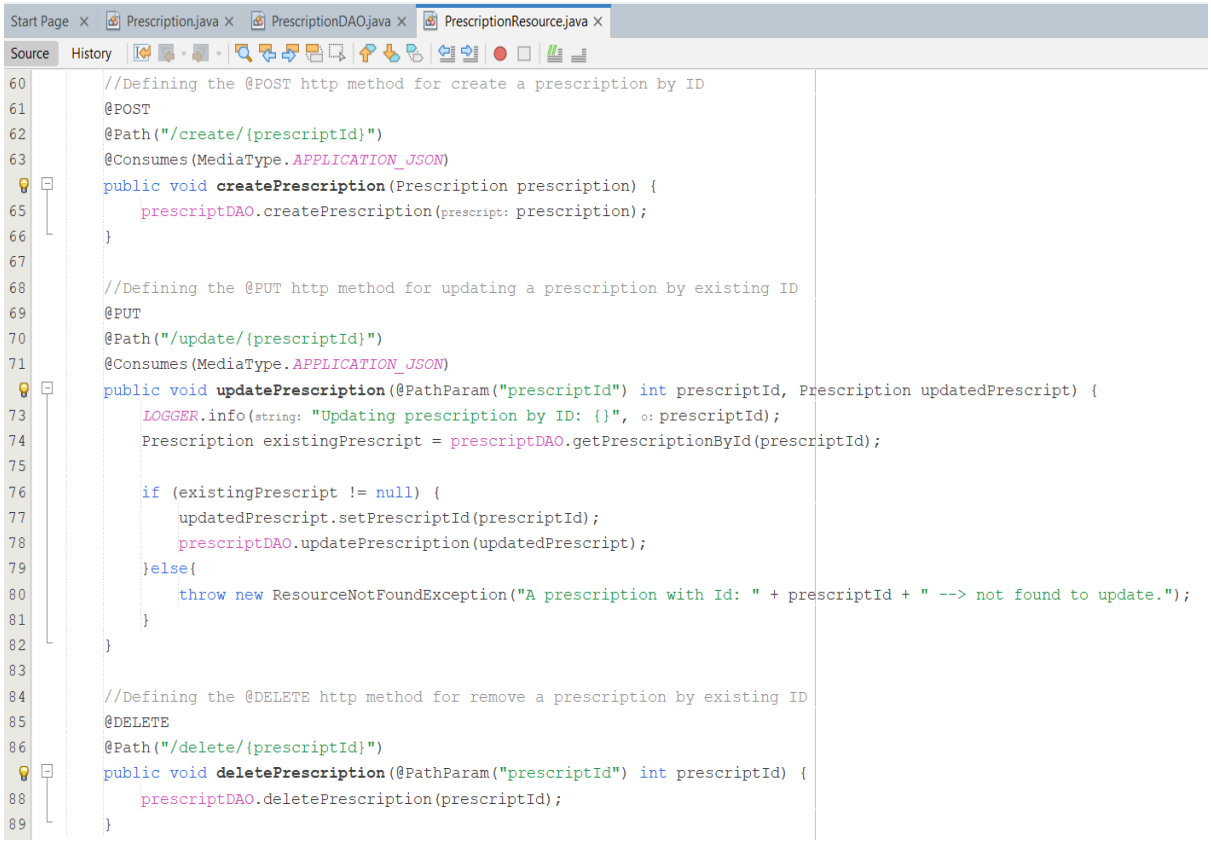


Figure 23: Code snippet for PrescriptionDAO class



```java
@Path("/prescriptions")
public class PrescriptionResource {

    //Define the logger using logger factory
    private static final Logger LOGGER = LoggerFactory.getLogger(clazz: PrescriptionResource.class);

    //Making the object from PrescriptionDAO class
    private PrescriptionDAO prescriptDAO = new PrescriptionDAO();

    //Defining the @GET http method for getting all prescripts
    @GET
    @Path("/allPrescripts")
    @Produces(MediaType.APPLICATION_JSON)
    public List<Prescription> getAllPrescriptions() {
        return prescriptDAO.getAllPrescriptions();
    }

    //Defining the @GET http method for getting a prescription by ID
    @GET
    @Path("/{prescriptId}")
    @Produces(MediaType.APPLICATION_JSON)
    public Prescription getPrescriptionById(@PathParam("prescriptId") int prescriptId) {

        Prescription presObj = prescriptDAO.getPrescriptionById(prescriptId);
        if(presObj != null){
            LOGGER.info(string: "Getting prescription by Id: {}", o: prescriptId);
            return presObj;
        }else{
            throw new ResourceNotFoundException("Prescription with this Id: " + prescriptId + " --> not found.");
        }
    }
```

Figure 24: Code snippet for PrescriptionResource class with @GET method



```java
    //Defining the @POST http method for create a prescription by ID
    @POST
    @Path("/create/{prescriptId}")
    @Consumes(MediaType.APPLICATION_JSON)
    public void createPrescription(Prescription prescription) {
        prescriptDAO.createPrescription(prescript: prescription);
    }

    //Defining the @PUT http method for updating a prescription by existing ID
    @PUT
    @Path("/update/{prescriptId}")
    @Consumes(MediaType.APPLICATION_JSON)
    public void updatePrescription(@PathParam("prescriptId") int prescriptId, Prescription updatedPrescript) {
        LOGGER.info(string: "Updating prescription by ID: {}", o: prescriptId);
        Prescription existingPrescript = prescriptDAO.getPrescriptionById(prescriptId);

        if (existingPrescript != null) {
            updatedPrescript.setPrescriptId(prescriptId);
            prescriptDAO.updatePrescription(updatedPrescript);
        }else{
            throw new ResourceNotFoundException("A prescription with Id: " + prescriptId + " --> not found to update.");
        }
    }

    //Defining the @DELETE http method for remove a prescription by existing ID
    @DELETE
    @Path("/delete/{prescriptId}")
    public void deletePrescription(@PathParam("prescriptId") int prescriptId) {
        prescriptDAO.deletePrescription(prescriptId);
    }
```

Figure 25: Code snippet for PrescriptionResource class with @POST, @PUT, & @DELETE methods

Billing, BillingDAO, BillingResource

This also included with all the related private attributes and the relevant getters and setters methods for Billing model class. As well as createBill(), getAllBills, getBillById(), updateBill(), and deleteBill() methods are belong to the BillingDAO class. And that methods are doing the CRUD operations perfectly. Apart from that all the RESTful HTTP methods are done in the BillingResource class (@GET, @POST, @PUT, @DELETE).

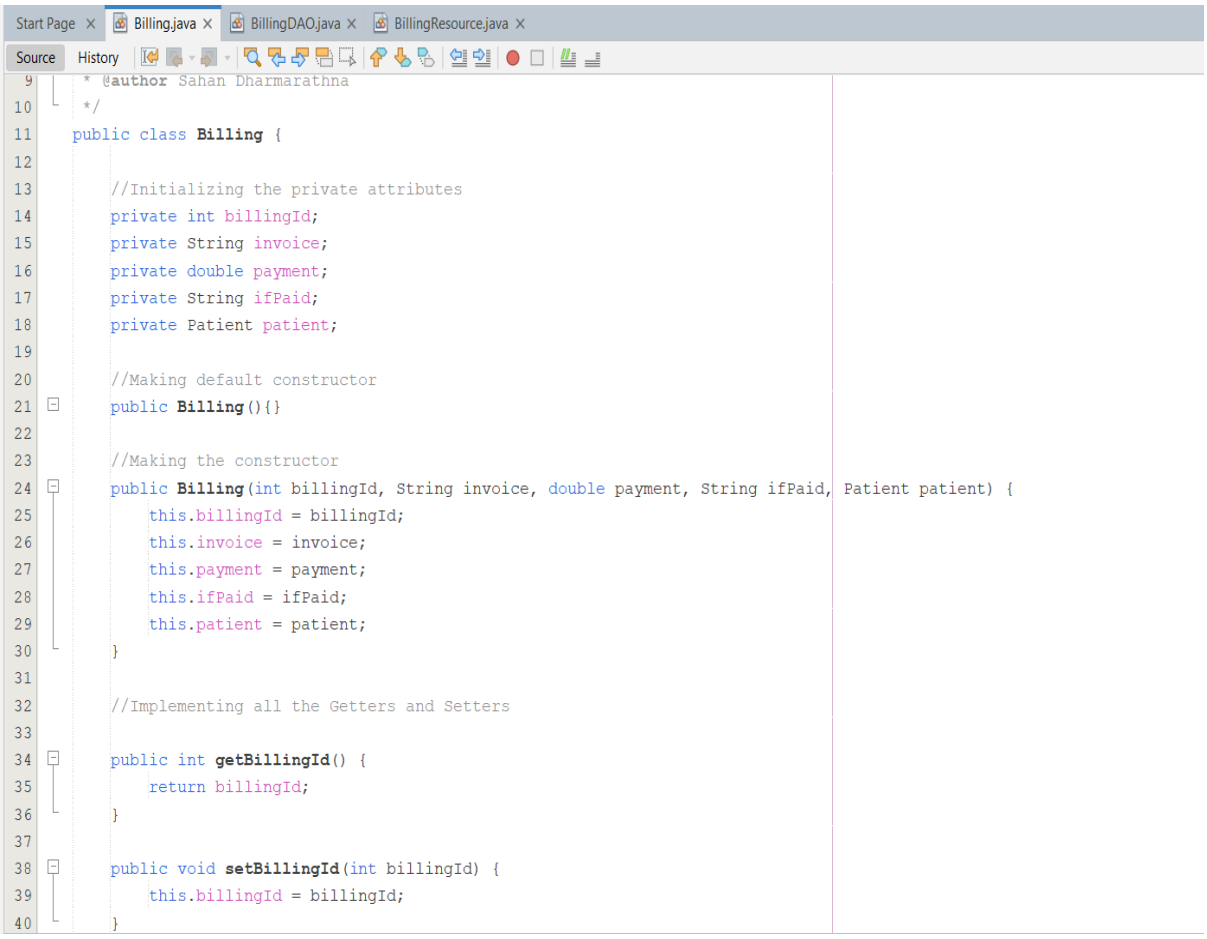
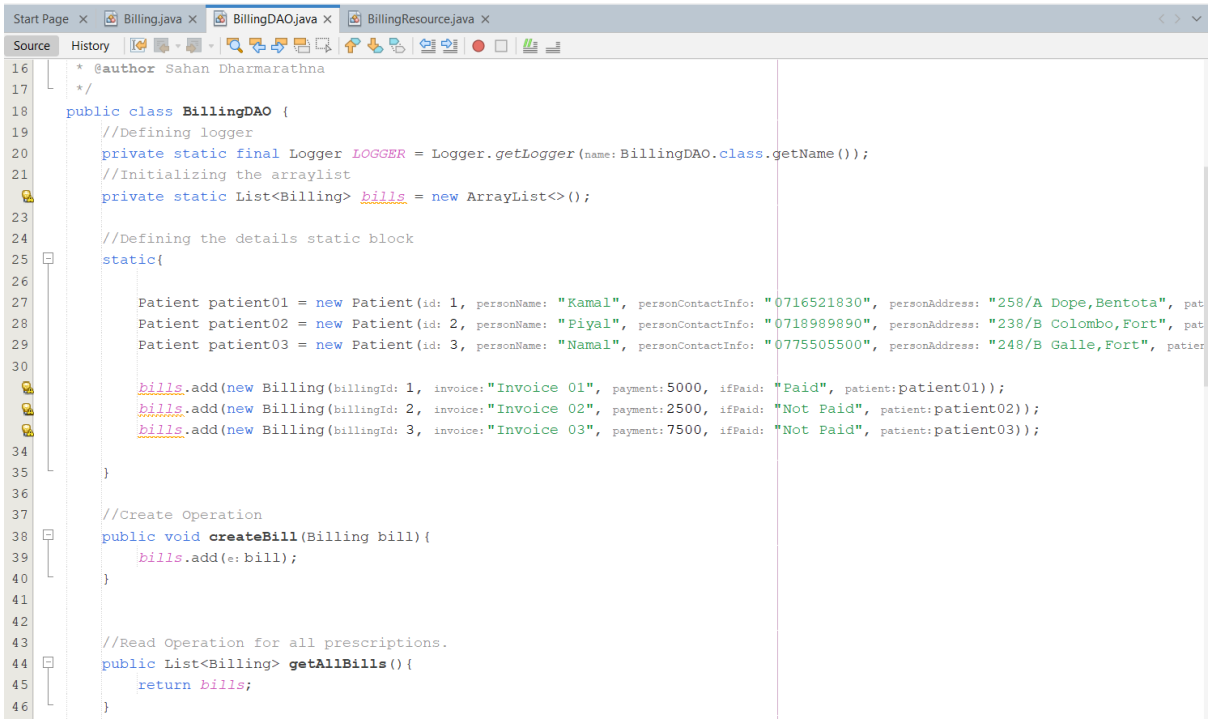
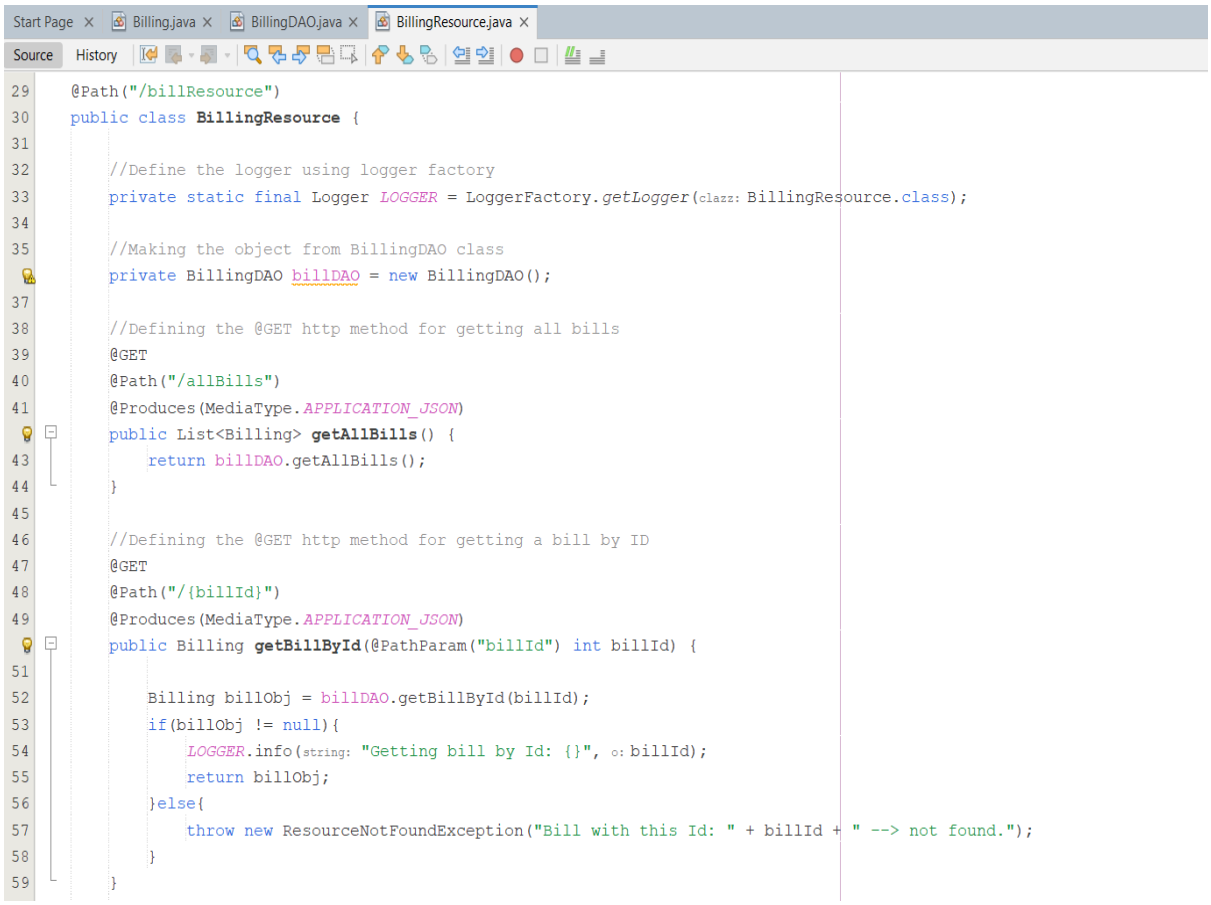


Figure 26: Code snippet for Billing model class



```java
 * @author Sahan Dharmarathna
 */
public class BillingDAO {
    //Defining logger
    private static final Logger LOGGER = Logger.getLogger(name: BillingDAO.class.getName());
    //Initializing the arraylist
    private static List<Billing> bills = new ArrayList<>();

    //Defining the details static block
    static{

        Patient patient01 = new Patient(id: 1, personName: "Kamal", personContactInfo: "0716521830", personAddress: "258/A Dope,Bentota", pat
        Patient patient02 = new Patient(id: 2, personName: "Piyal", personContactInfo: "0718989890", personAddress: "238/B Colombo,Fort", pat
        Patient patient03 = new Patient(id: 3, personName: "Namal", personContactInfo: "0775505500", personAddress: "248/B Galle,Fort", patien

        bills.add(new Billing(billingId: 1, invoice:"Invoice 01", payment:5000, ifPaid: "Paid", patient:patient01));
        bills.add(new Billing(billingId: 2, invoice:"Invoice 02", payment:2500, ifPaid: "Not Paid", patient:patient02));
        bills.add(new Billing(billingId: 3, invoice:"Invoice 03", payment:7500, ifPaid: "Not Paid", patient:patient03));

    }

    //Create Operation
    public void createBill(Billing bill){
        bills.add(e: bill);
    }


    //Read Operation for all prescriptions.
    public List<Billing> getAllBills(){
        return bills;
    }
}
```

Figure 27: Code snippet for BillingDAO class



```java
@Path("/billResource")
public class BillingResource {

    //Define the logger using logger factory
    private static final Logger LOGGER = LoggerFactory.getLogger(clazz: BillingResource.class);

    //Making the object from BillingDAO class
    private BillingDAO billDAO = new BillingDAO();

    //Defining the @GET http method for getting all bills
    @GET
    @Path("/allBills")
    @Produces(MediaType.APPLICATION_JSON)
    public List<Billing> getAllBills() {
        return billDAO.getAllBills();
    }

    //Defining the @GET http method for getting a bill by ID
    @GET
    @Path("/{billId}")
    @Produces(MediaType.APPLICATION_JSON)
    public Billing getBillById(@PathParam("billId") int billId) {

        Billing billObj = billDAO.getBillById(billId);
        if(billObj != null){
            LOGGER.info(string: "Getting bill by Id: {}", o: billId);
            return billObj;
        }else{
            throw new ResourceNotFoundException("Bill with this Id: " + billId + " --> not found.");
        }
    }
```

Figure 28: Code snippet for BillingResource class with @GET method

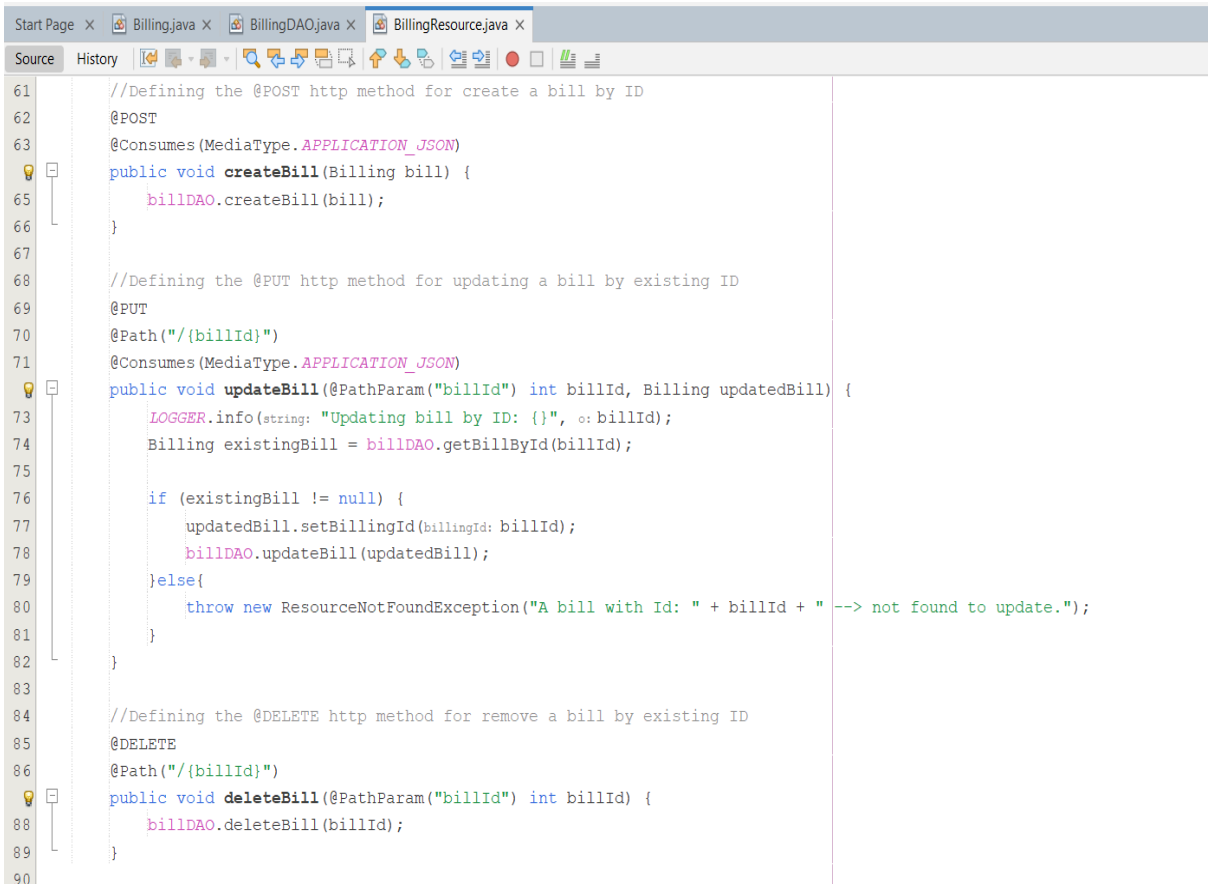


Figure 29: Code snippet for BillingResource class with @POST, @PUT, & @DELETE methods

- Commonly all the Data Access Object(DAO) classes are included with a static block where the each relevant data set is available. All these above mentioned model classes, DAO classes, and Resource classes are included with loggers and exception handling with error messages using HTTP response codes.

Testing Results from Postman

Person Entity

- GET

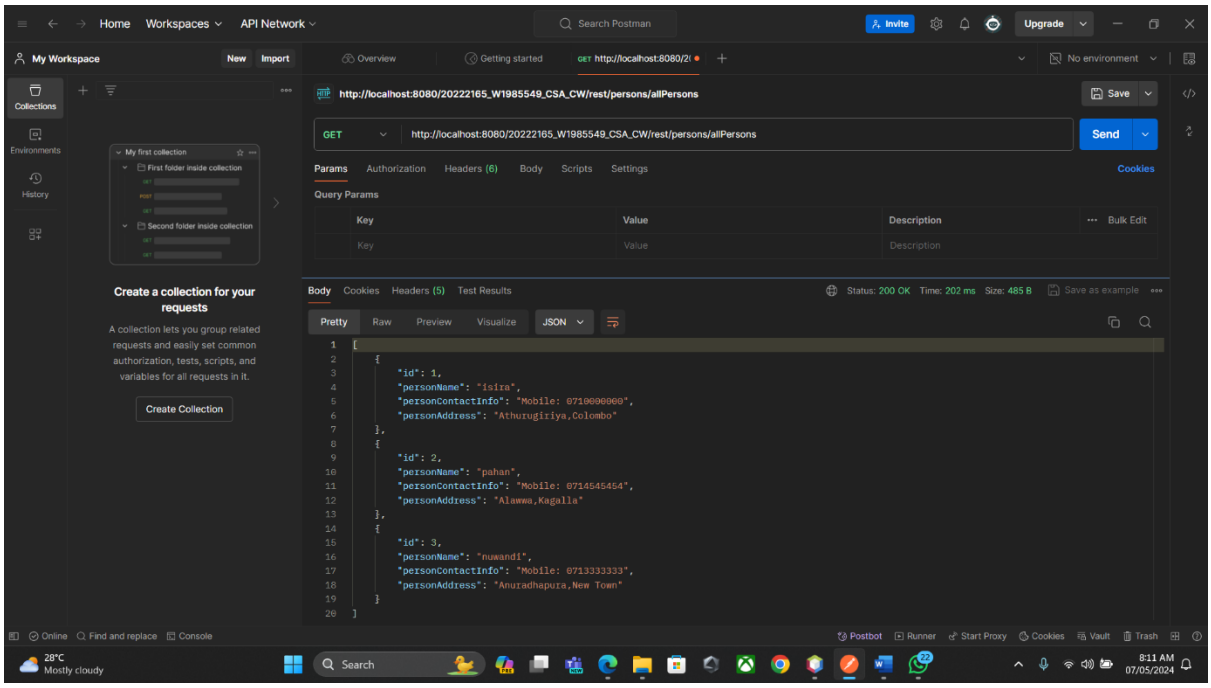


Figure 30: Postman Get method with all persons

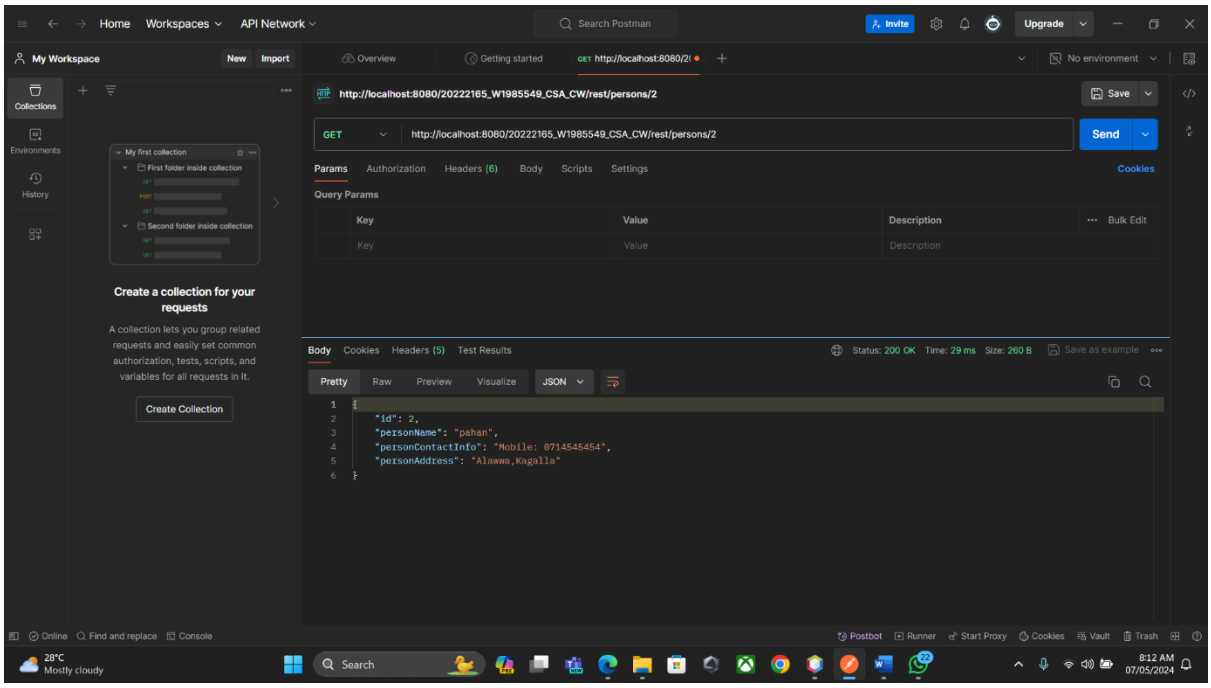


Figure 31: Postman Get method with persons id

- POST

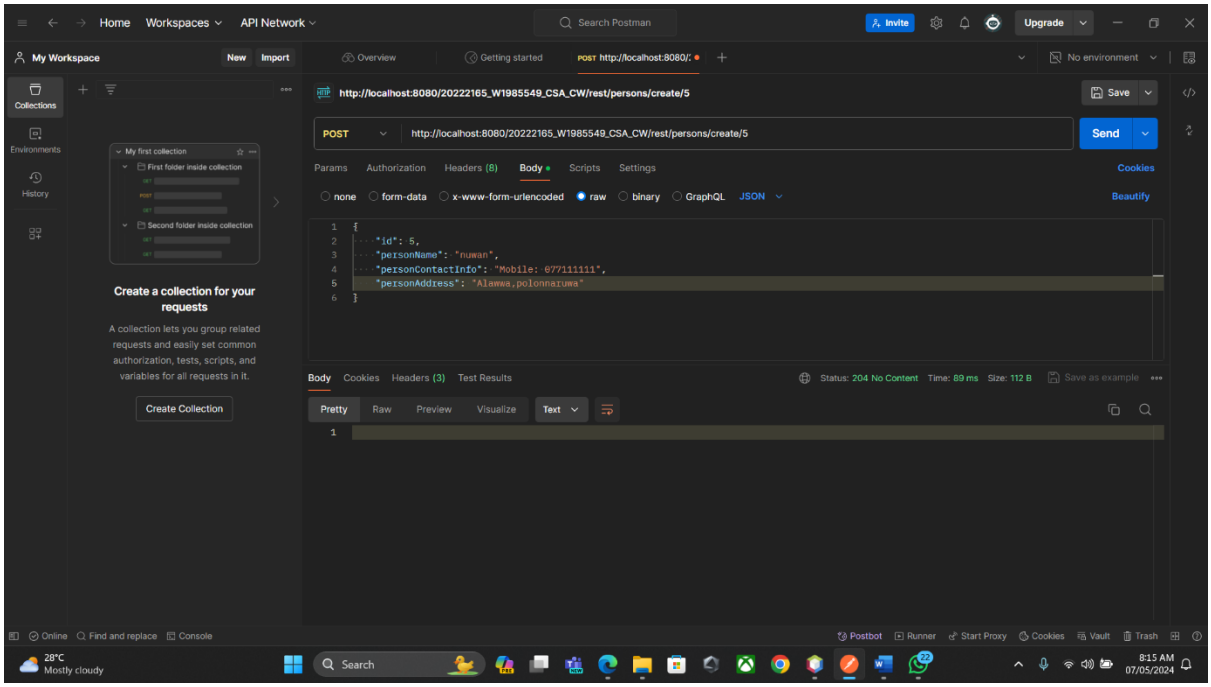


Figure 32: Postman POST method adding new person

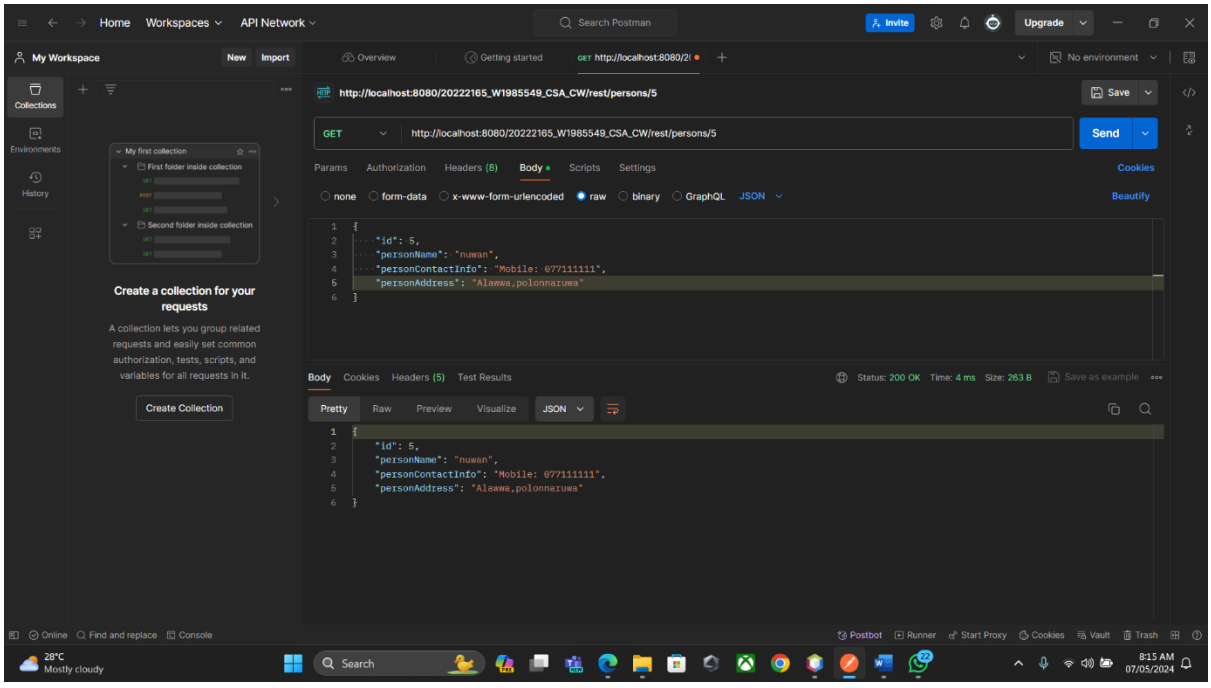


Figure 33: Postman POST method adding new person result

- PUT

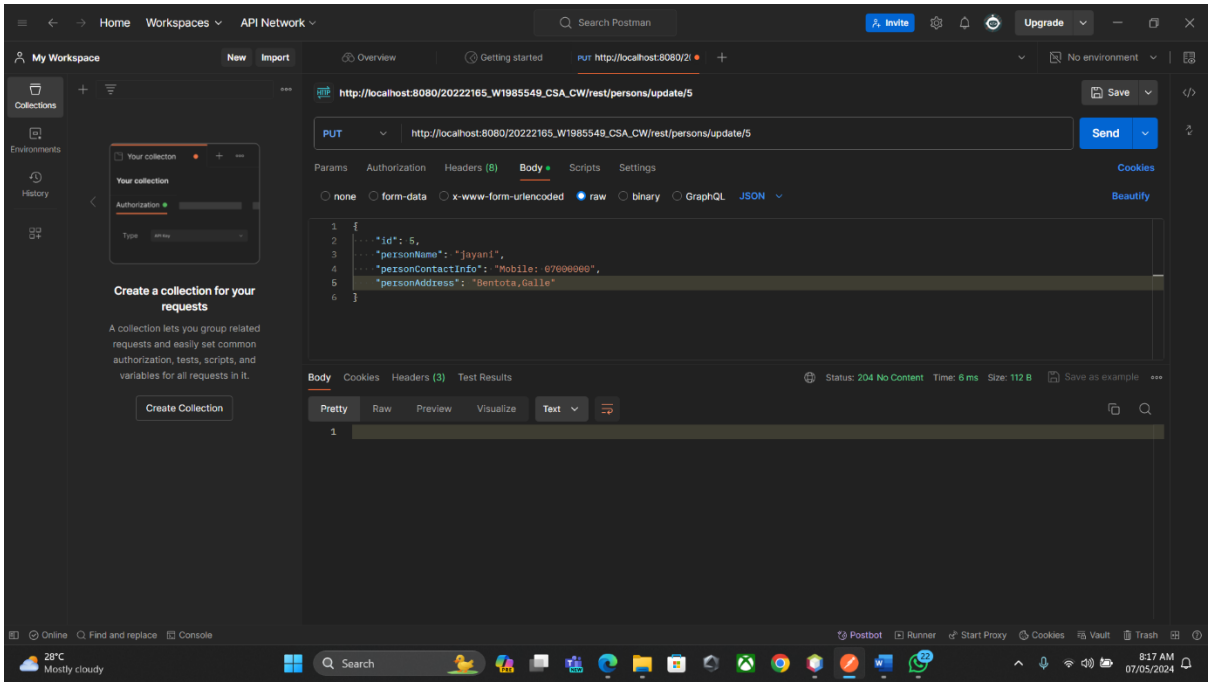


Figure 34: Postman PUT method updating existing person

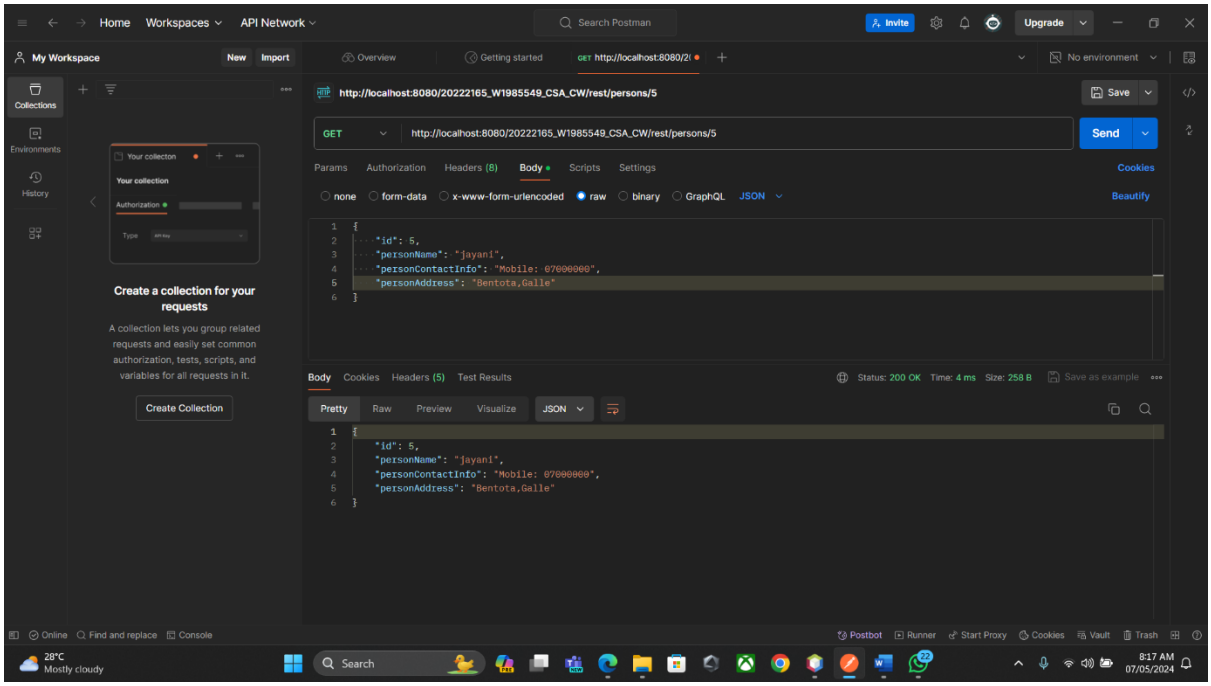


Figure 35: Postman PUT method updating existing person result

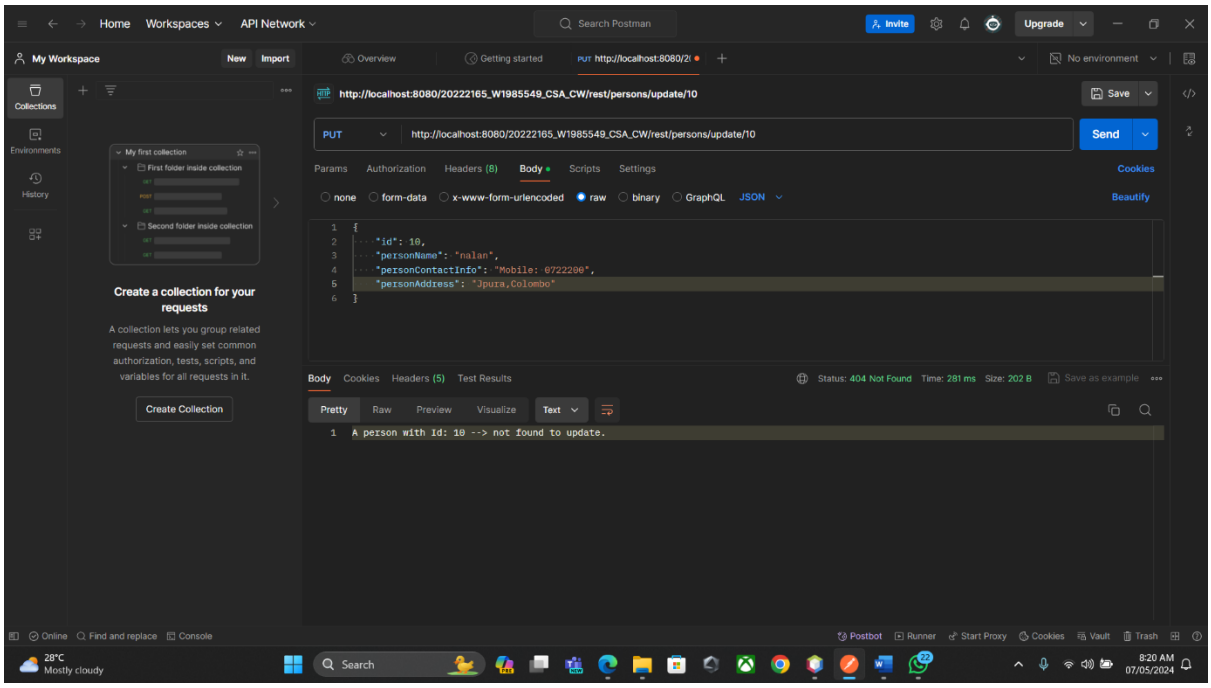


Figure 36: Postman PUT method updating non-existing person and giving error message

- DELETE

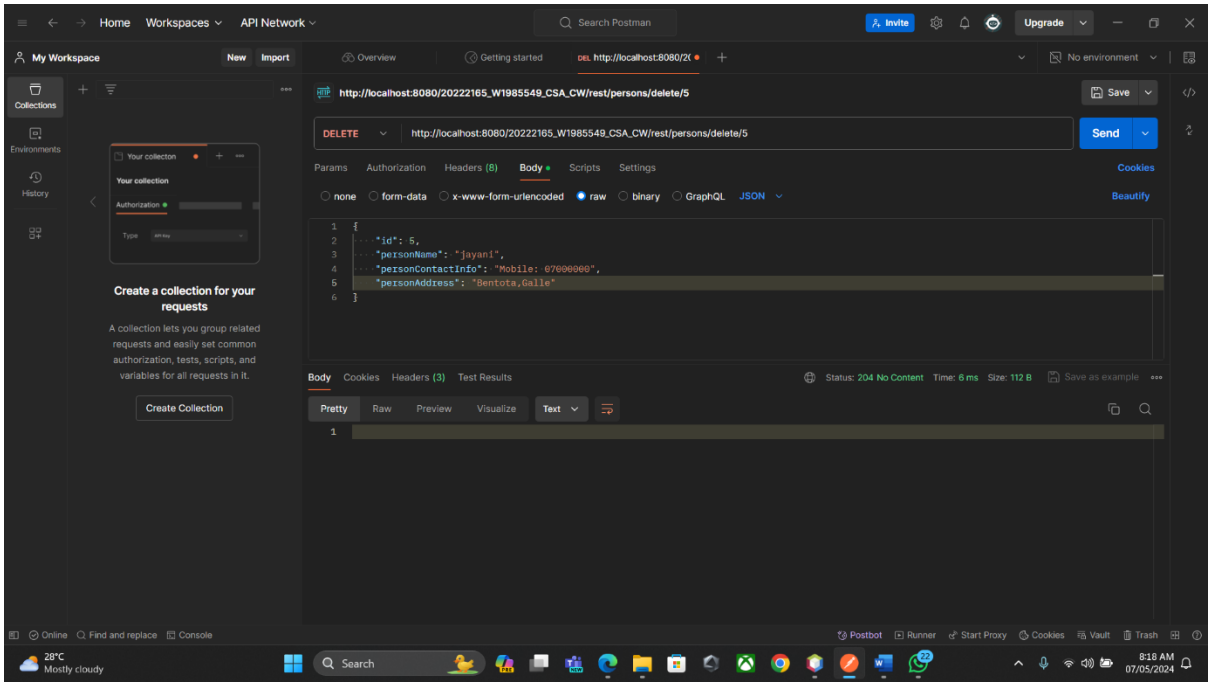


Figure 37: Postman DELETE method

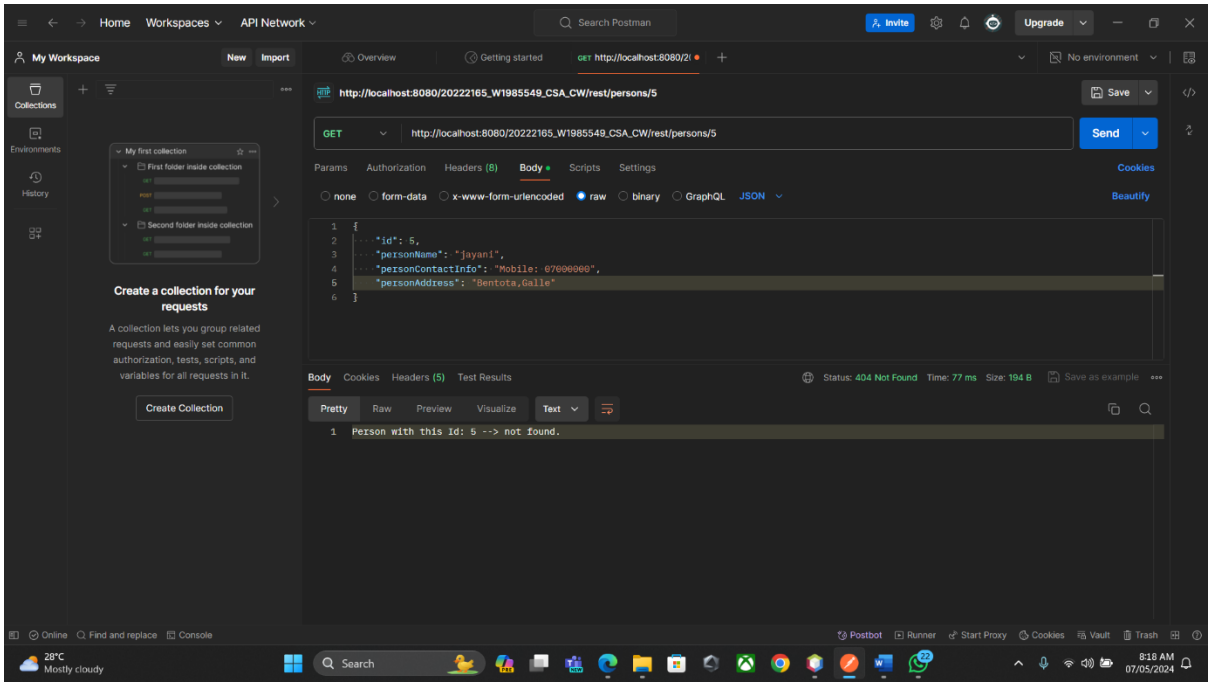


Figure 38: Postman DELETE method results

Patient Entity

- GET

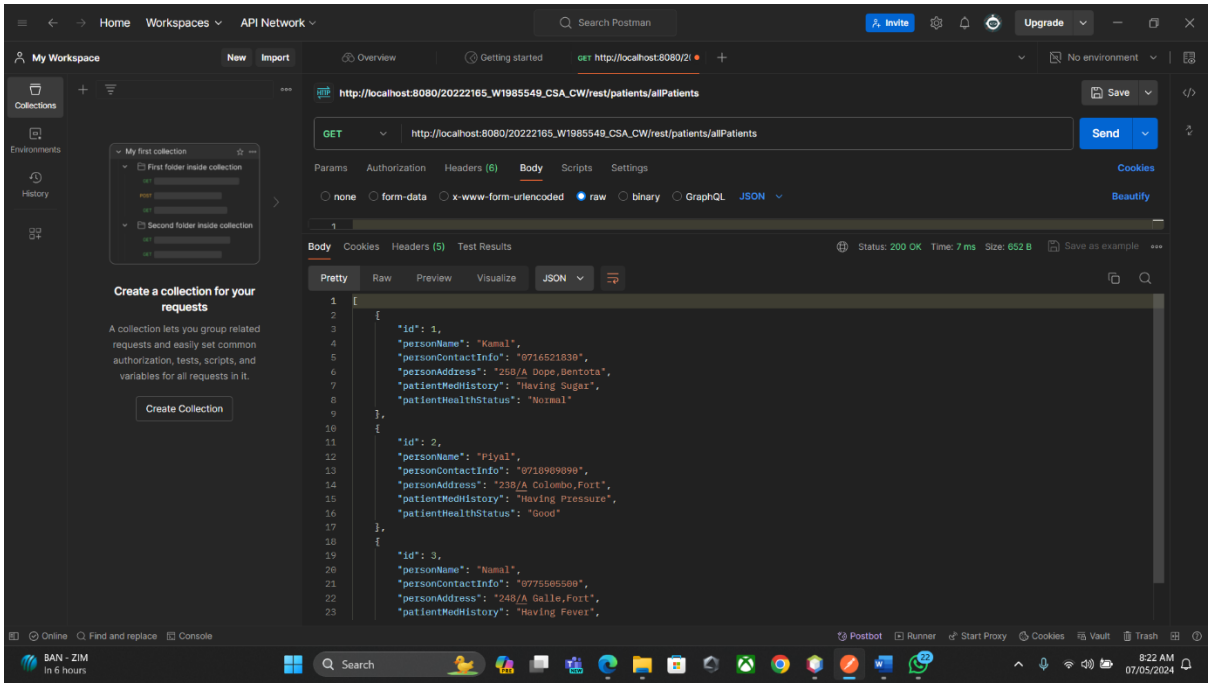


Figure 39: Postman GET method with all patients

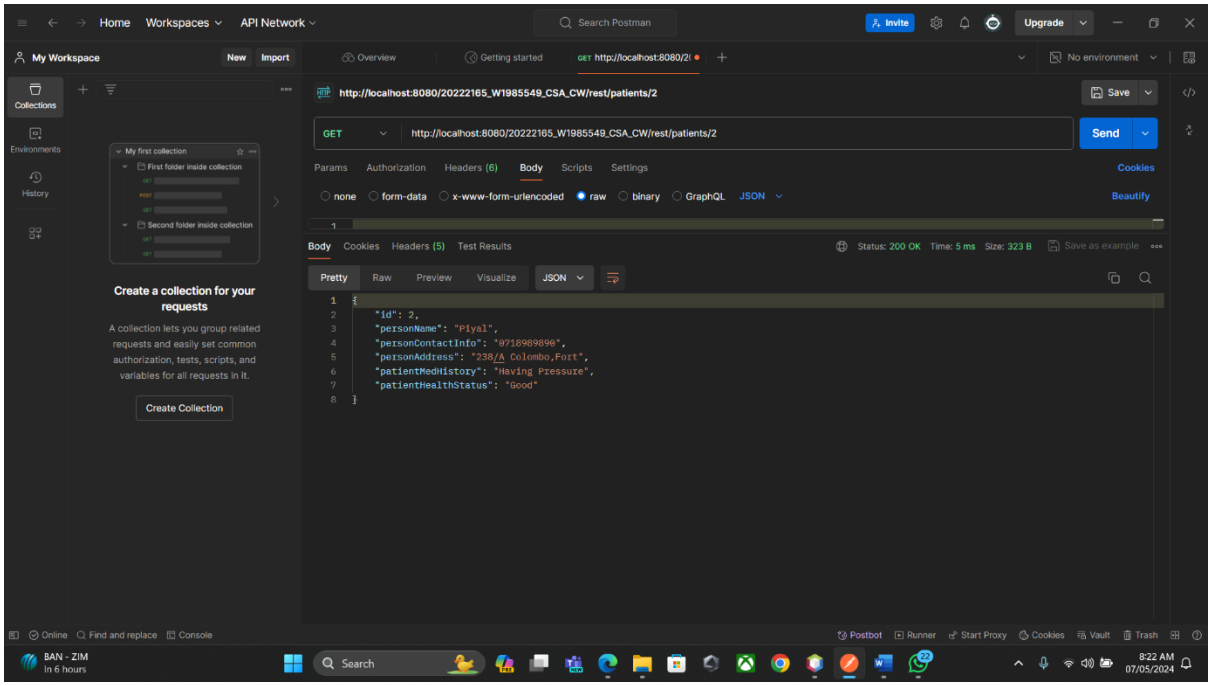


Figure 40:Postman GET method with patient with id

- POST

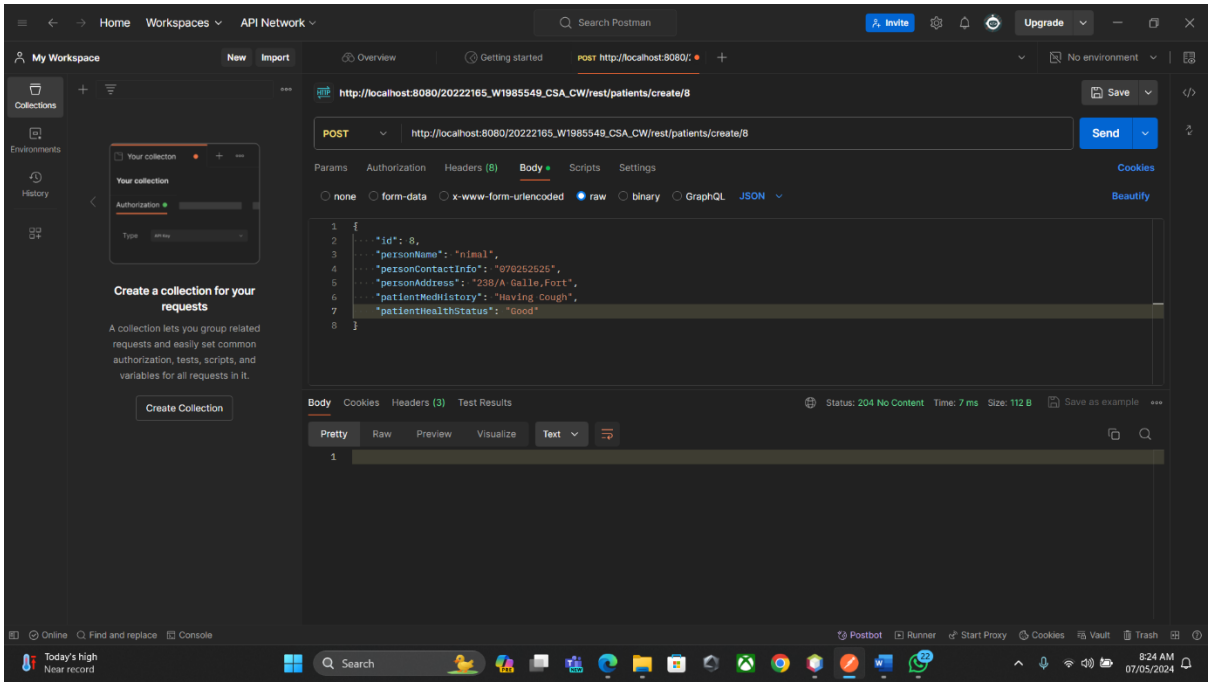


Figure 41:Postman POST method adding patient

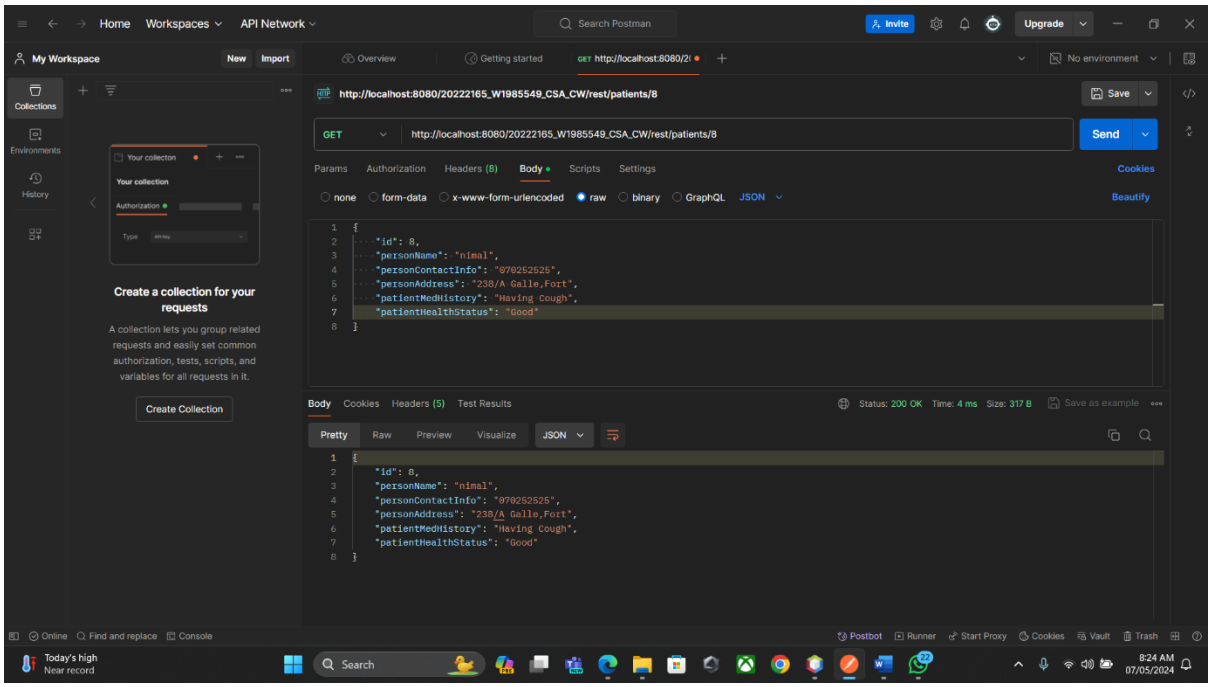


Figure 42:Postman POST method adding patient result

- PUT

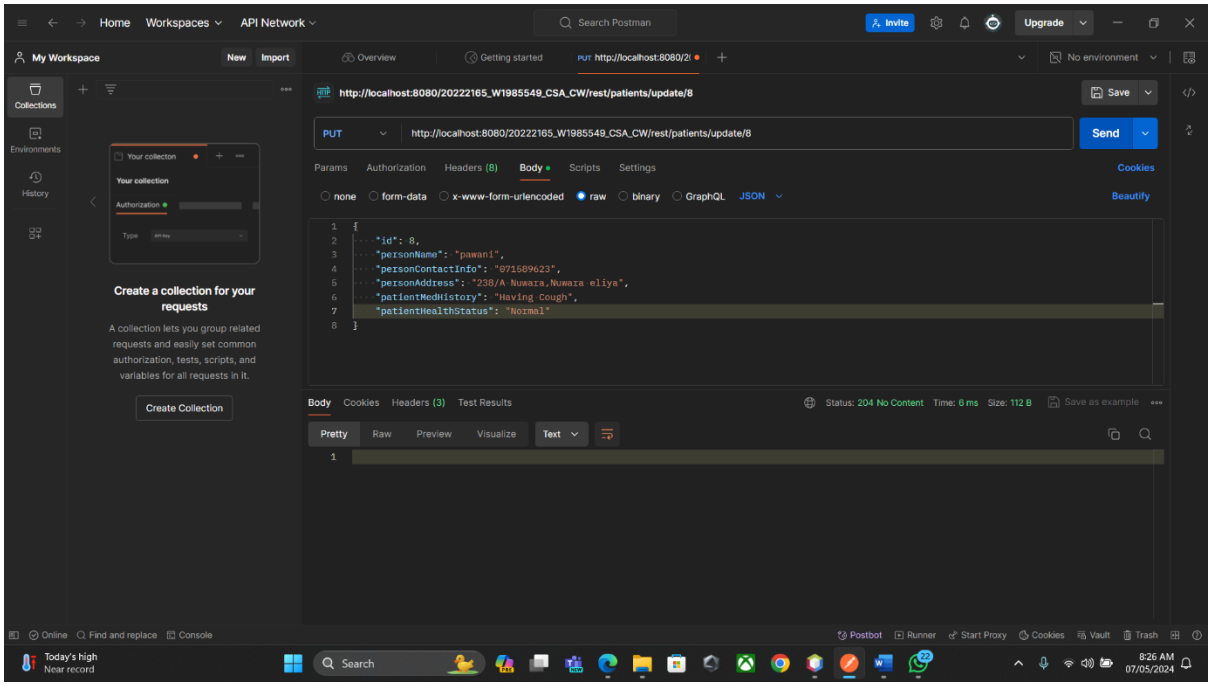


Figure 43:Postman PUT method updating existing patient

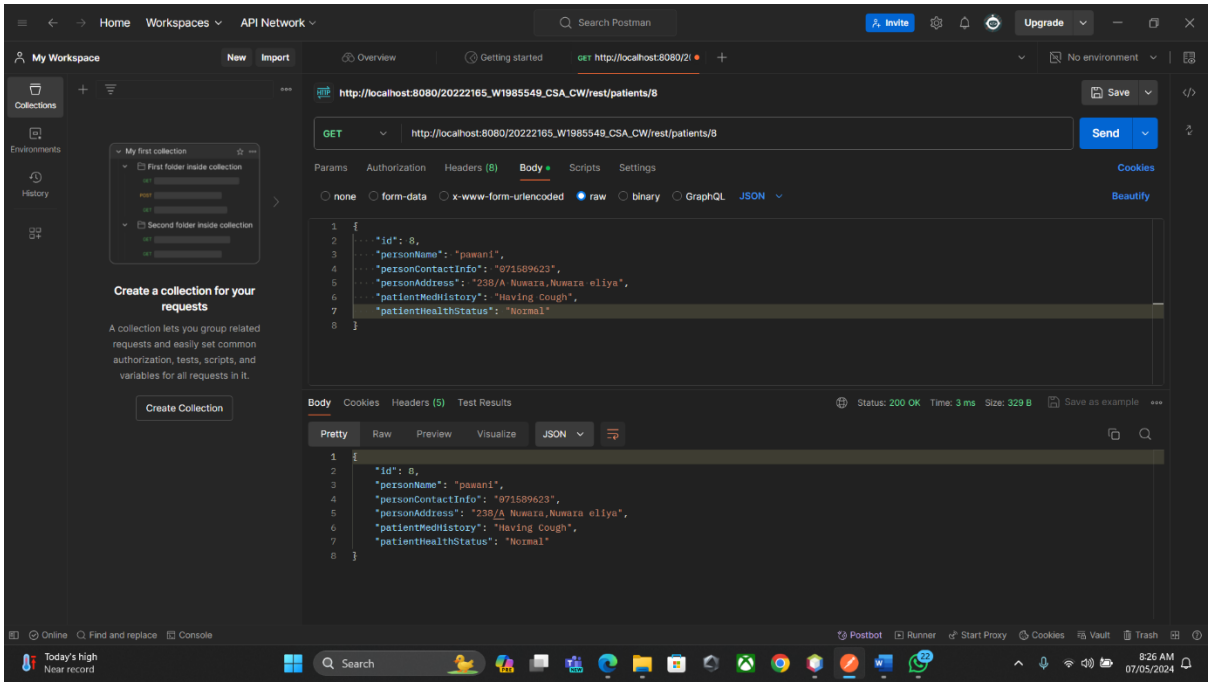


Figure 44:Postman PUT method updating existing patient result

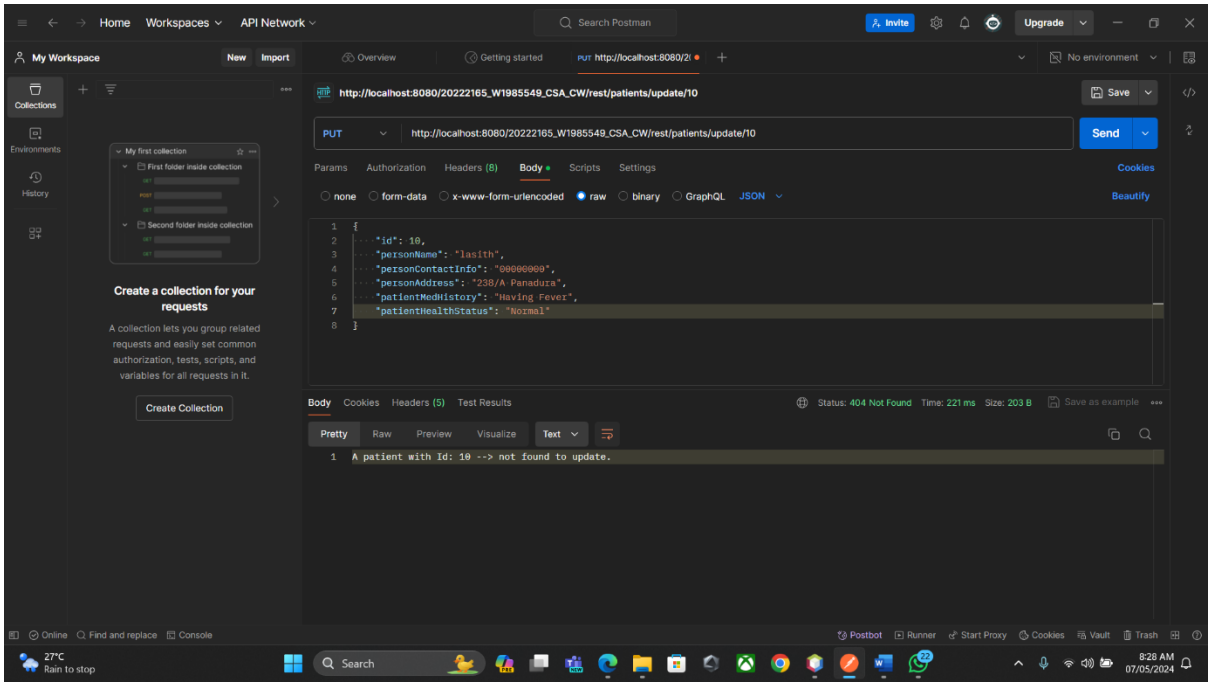


Figure 45:Postman PUT method updating non-existing patient and giving error message

- DELETE

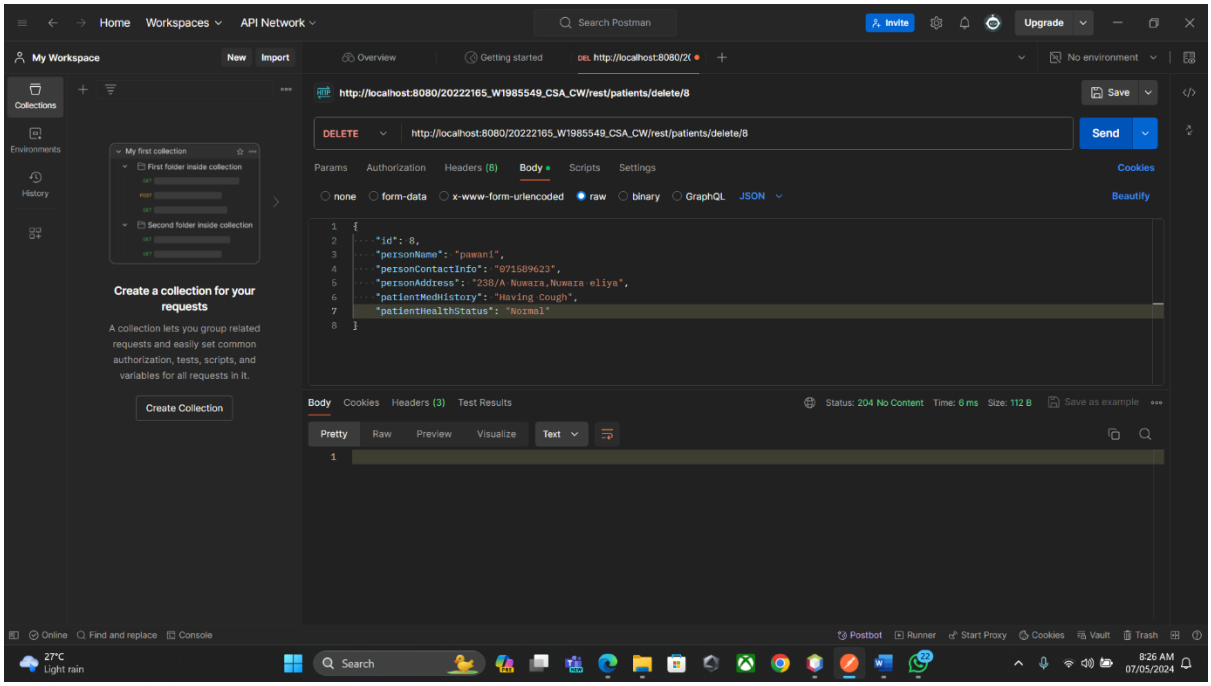


Figure 46:Postman DELETE method

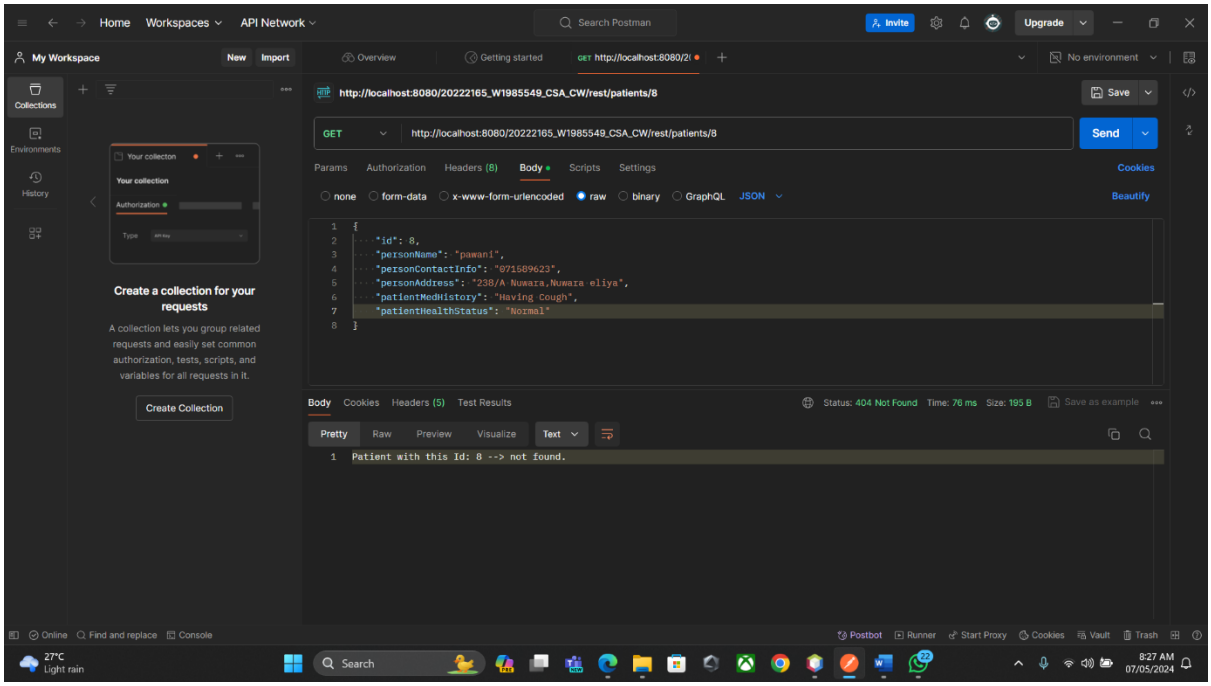


Figure 47:Postman DELETE method results by giving error message

Doctor Entity

- GET

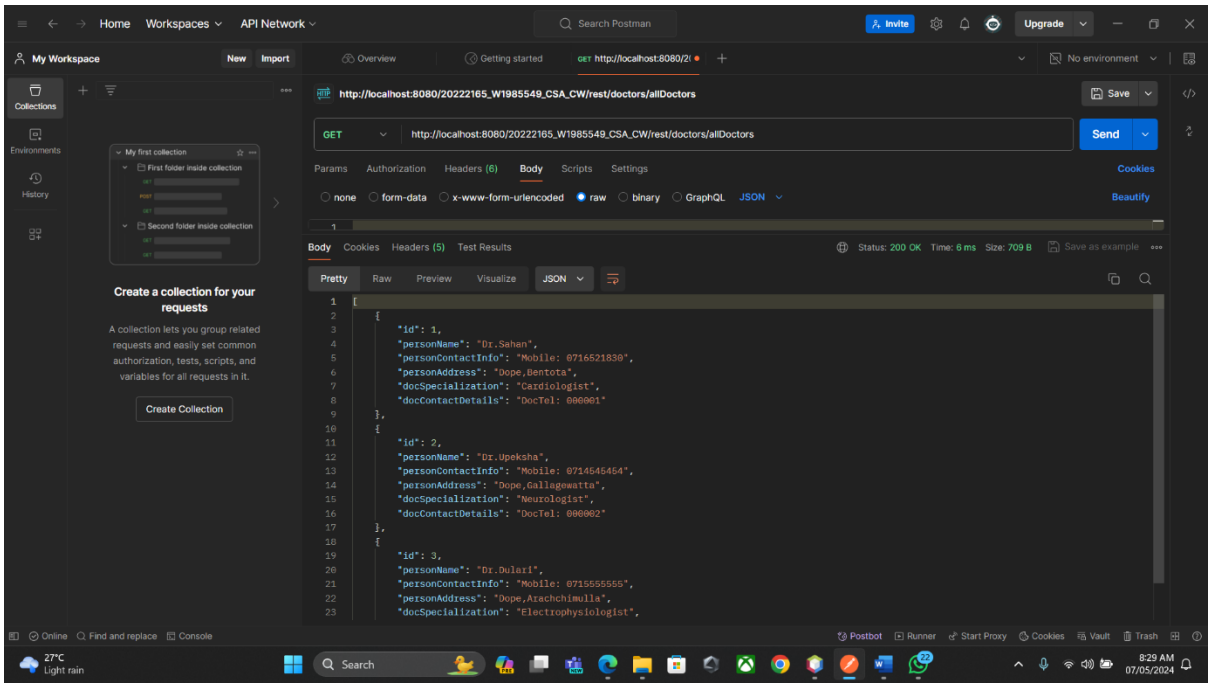


Figure 48:Postman GET method with all doctors

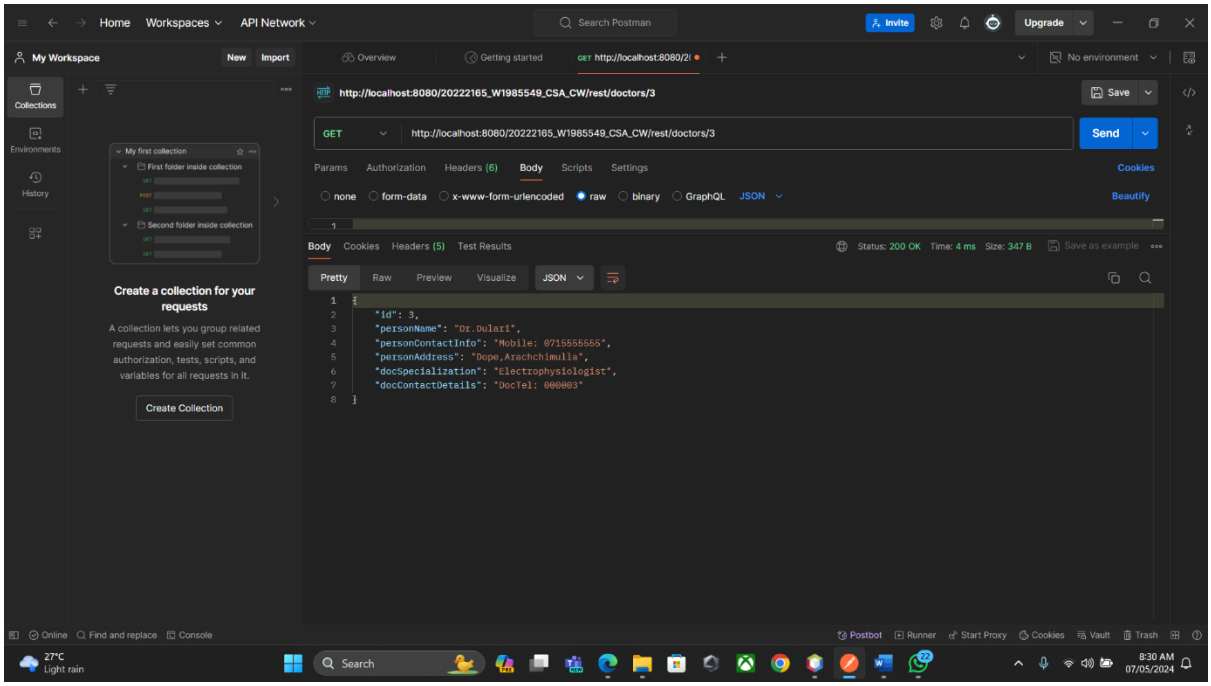


Figure 49:Postman GET method with doctor by id

- POST

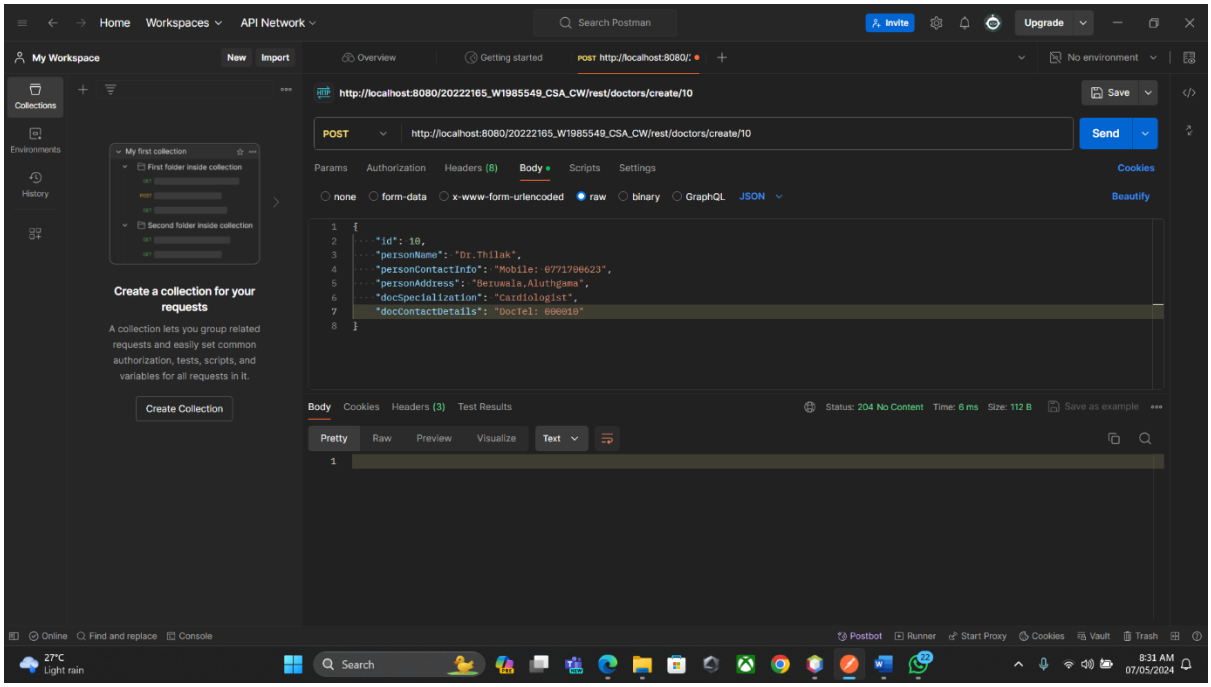


Figure 50:Postman POST method adding doctors

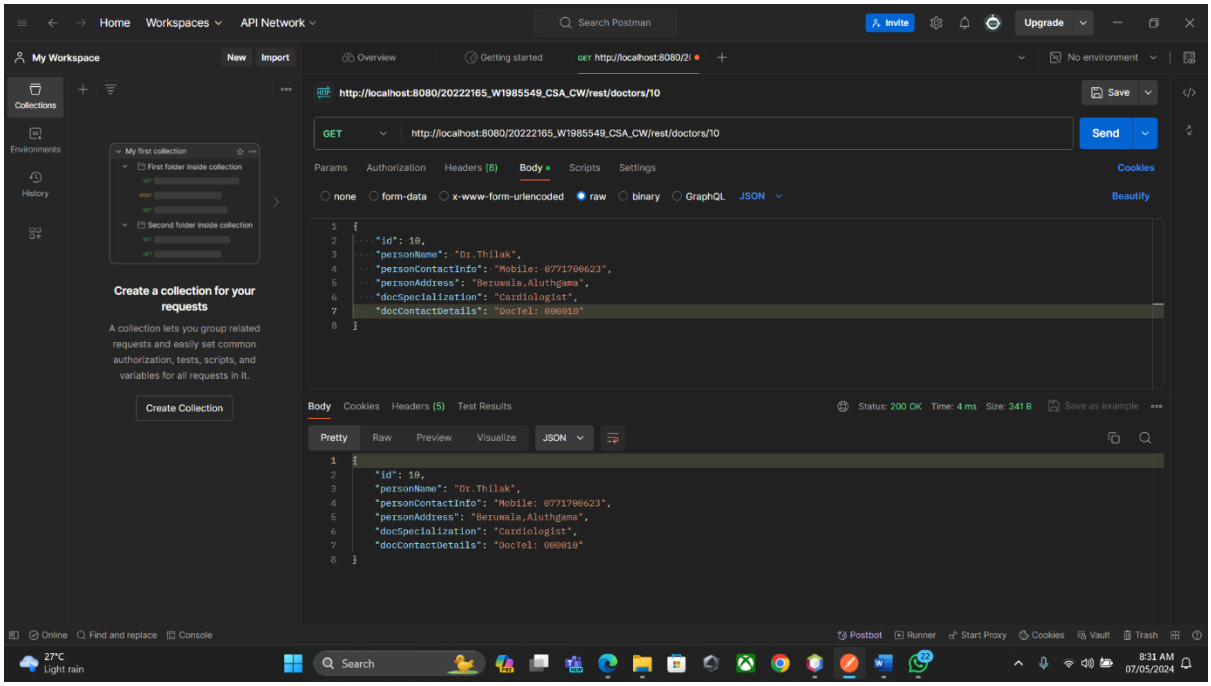


Figure 51:Postman POST method adding doctors results

- PUT

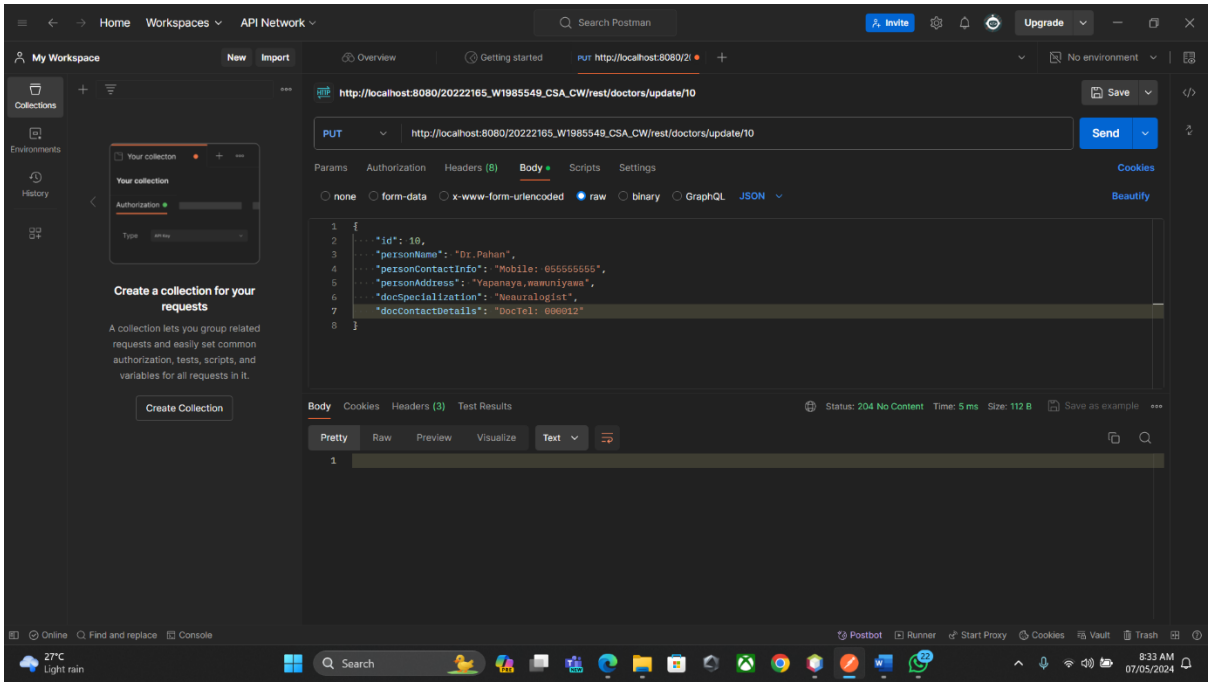


Figure 52:Postman PUT method updating existing doctors

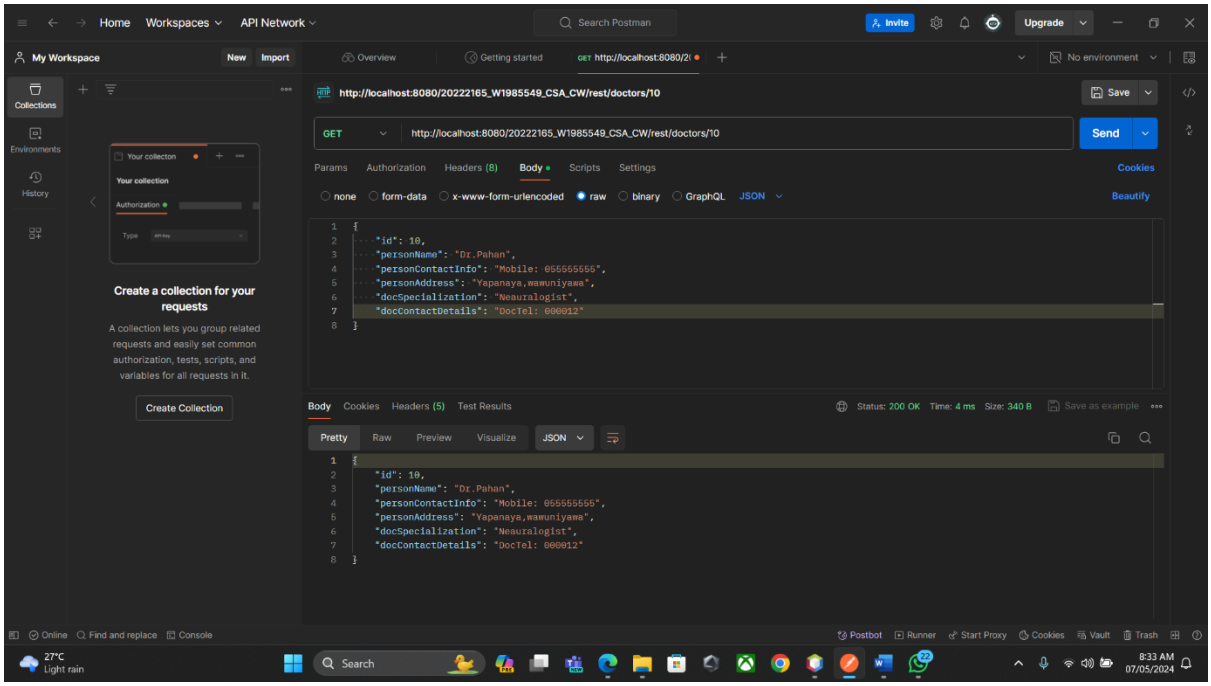


Figure 53:Postman PUT method updating existing doctors results

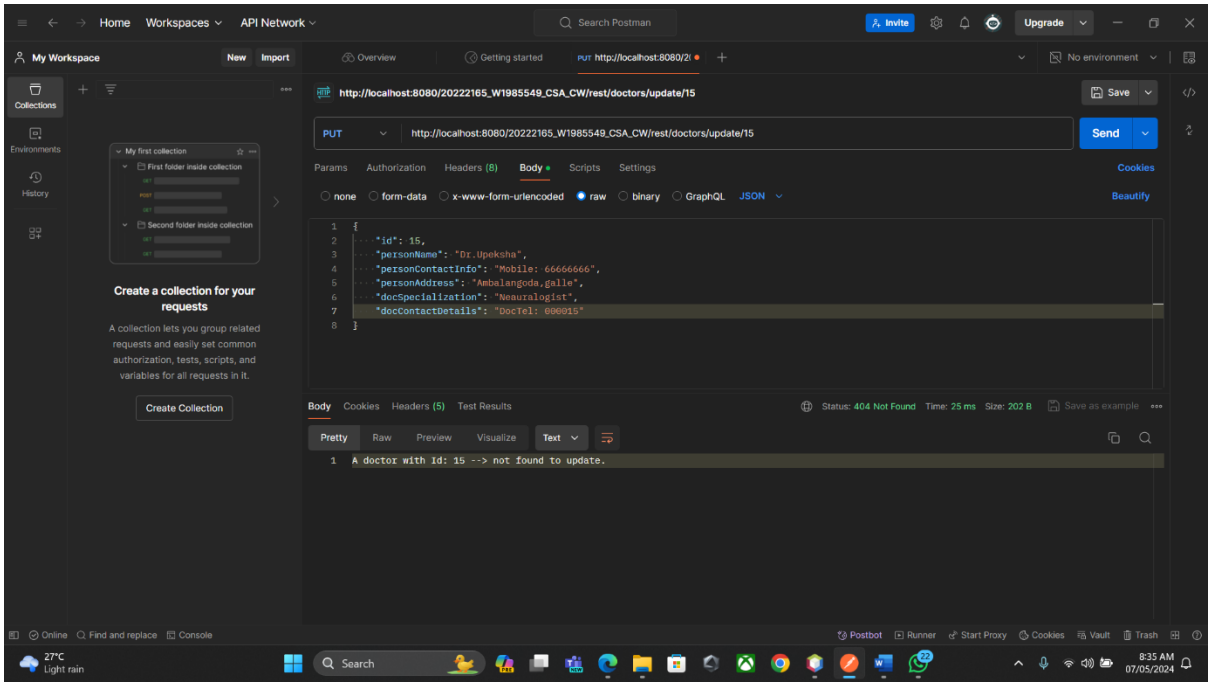


Figure 54:Postman PUT method updating non-existing doctors and giving errors

- DELETE

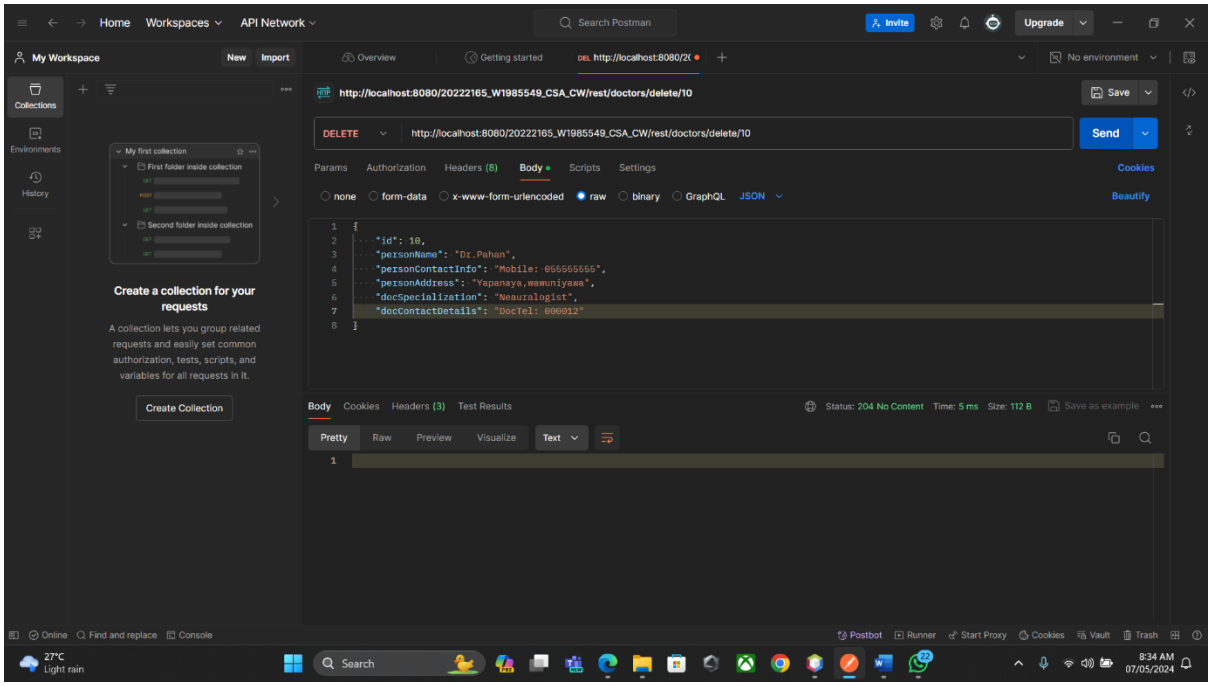


Figure 55:Postman DELETE method

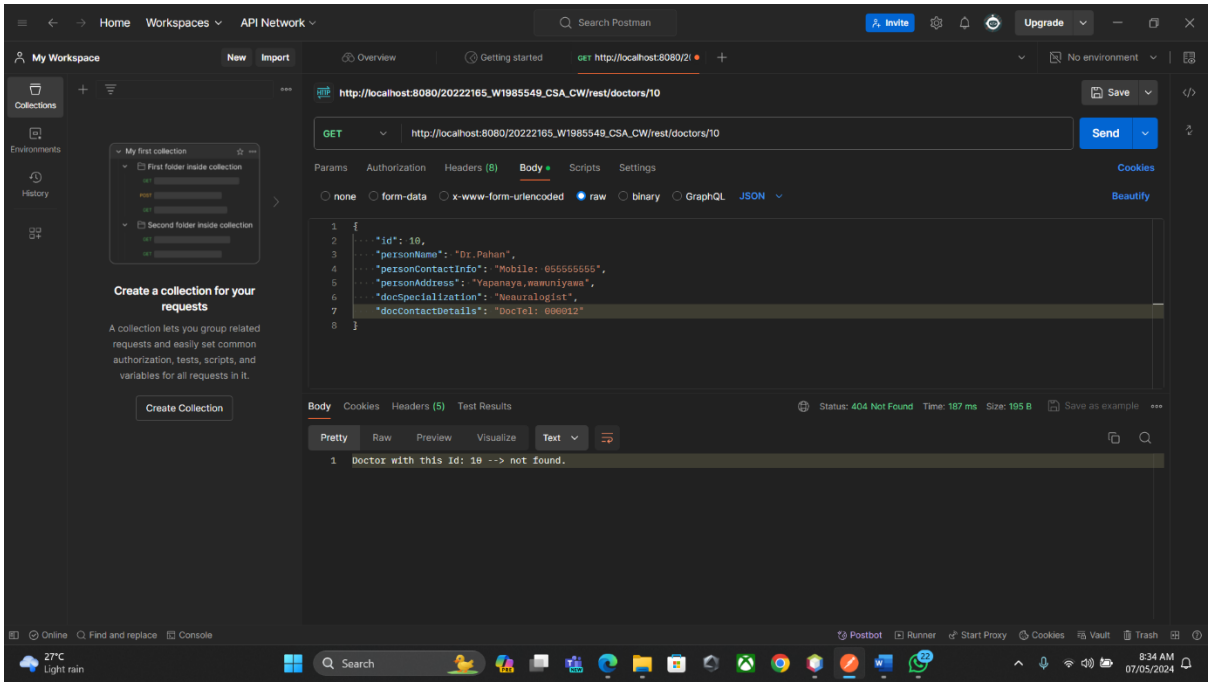


Figure 56:Postman DELETE method results with giving error message

Appointment Entity

- GET

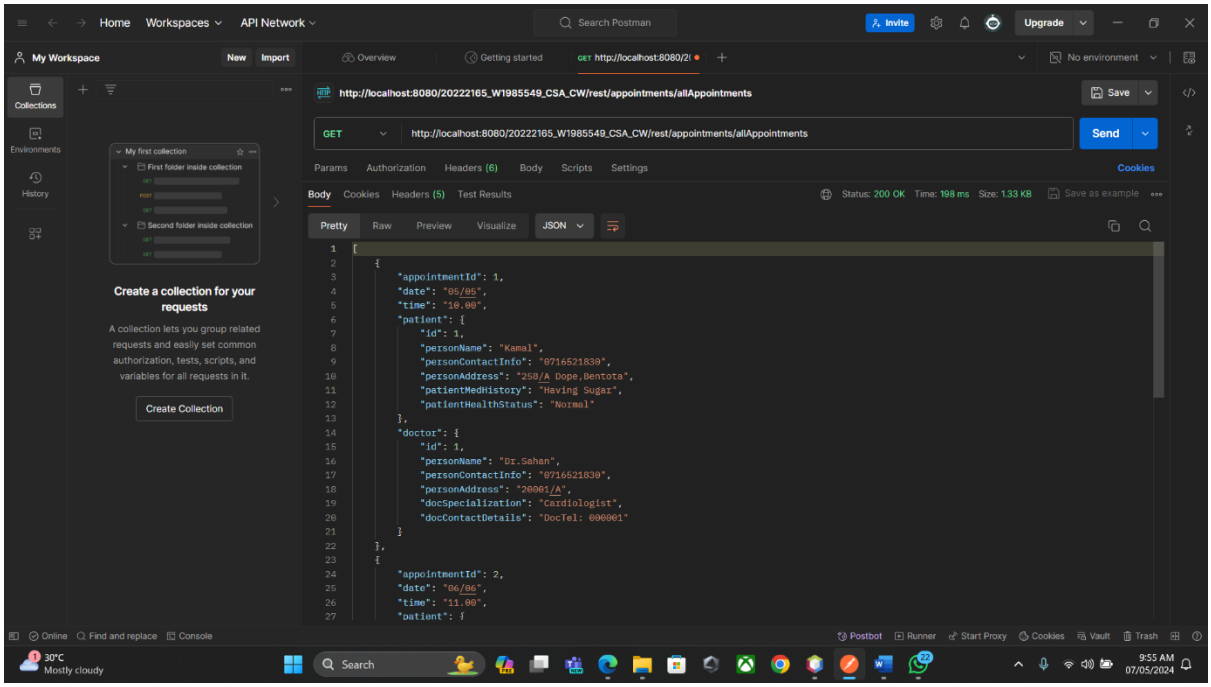


Figure 57:Postman GET method with all appointments

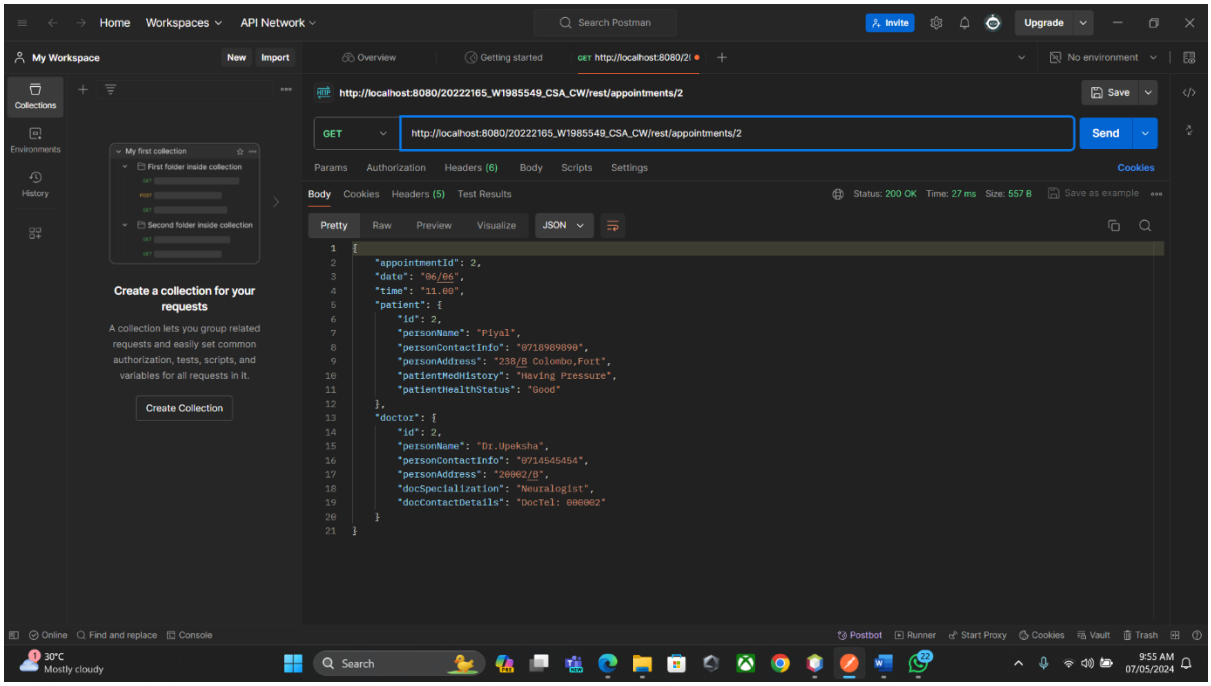


Figure 58:Postmat GET method with appointment by id

- POST

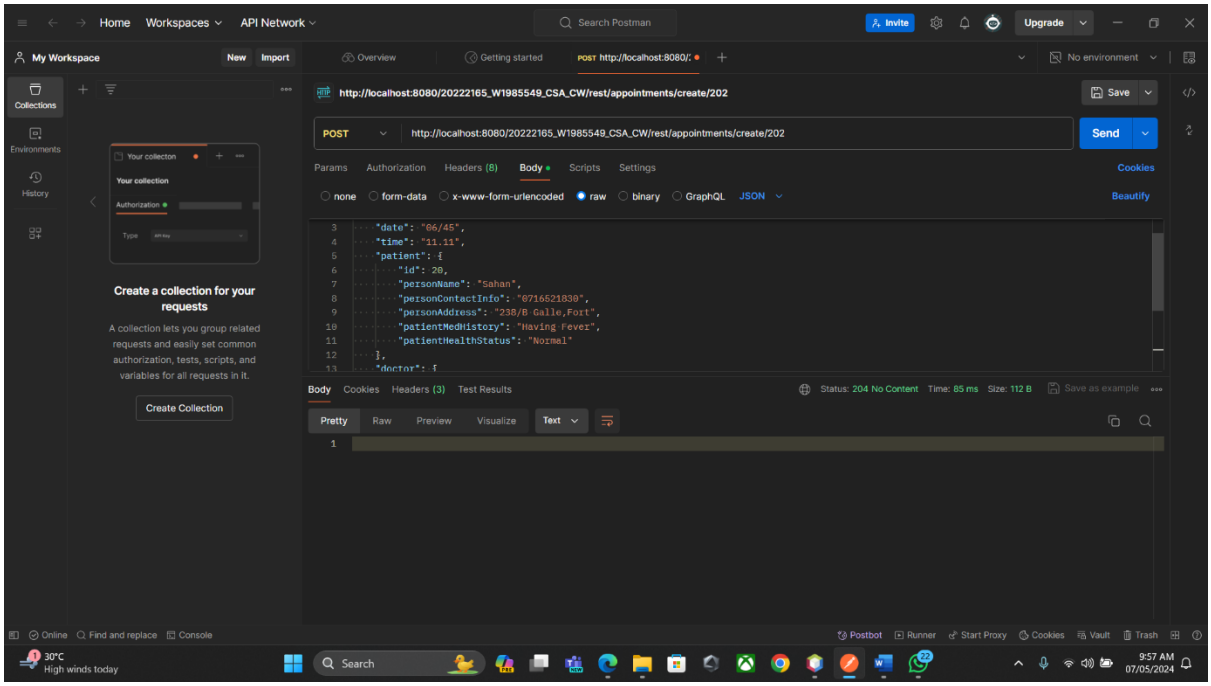


Figure 59: Postman POST method adding appoinments

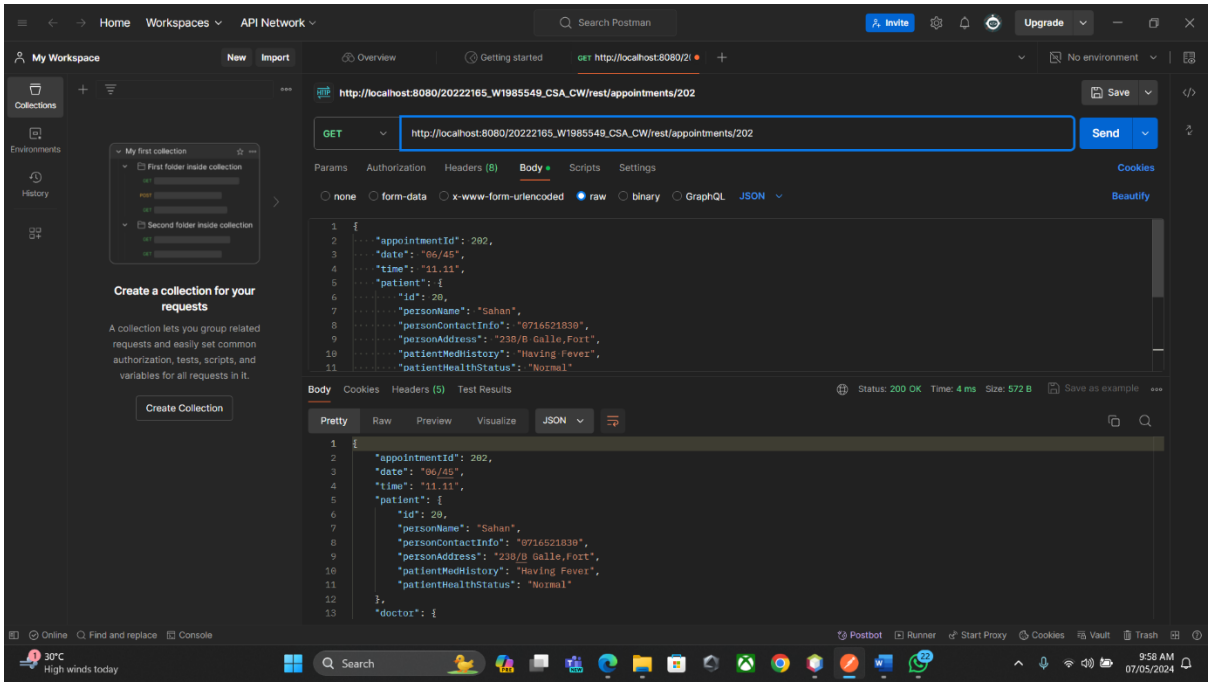


Figure 60: Postman POST method adding appoinments results

- PUT

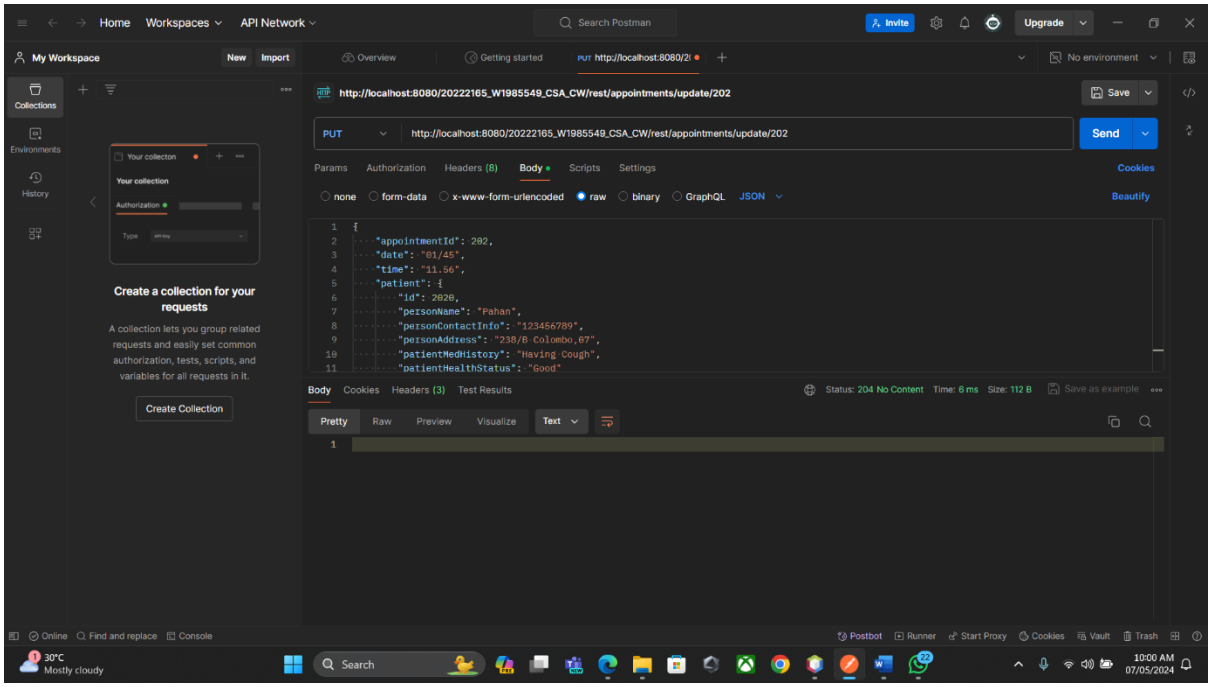


Figure 61: Postman PUT method Updating existing appoinments

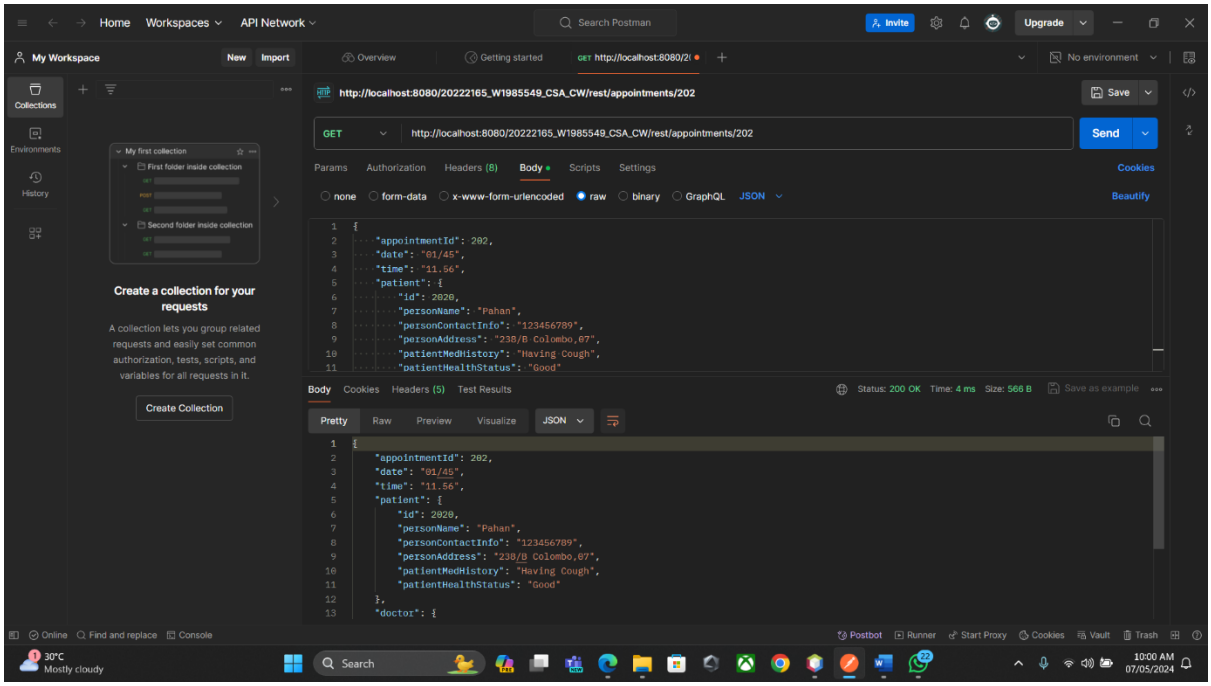


Figure 62: Postman PUT method Updating existing appoinments results

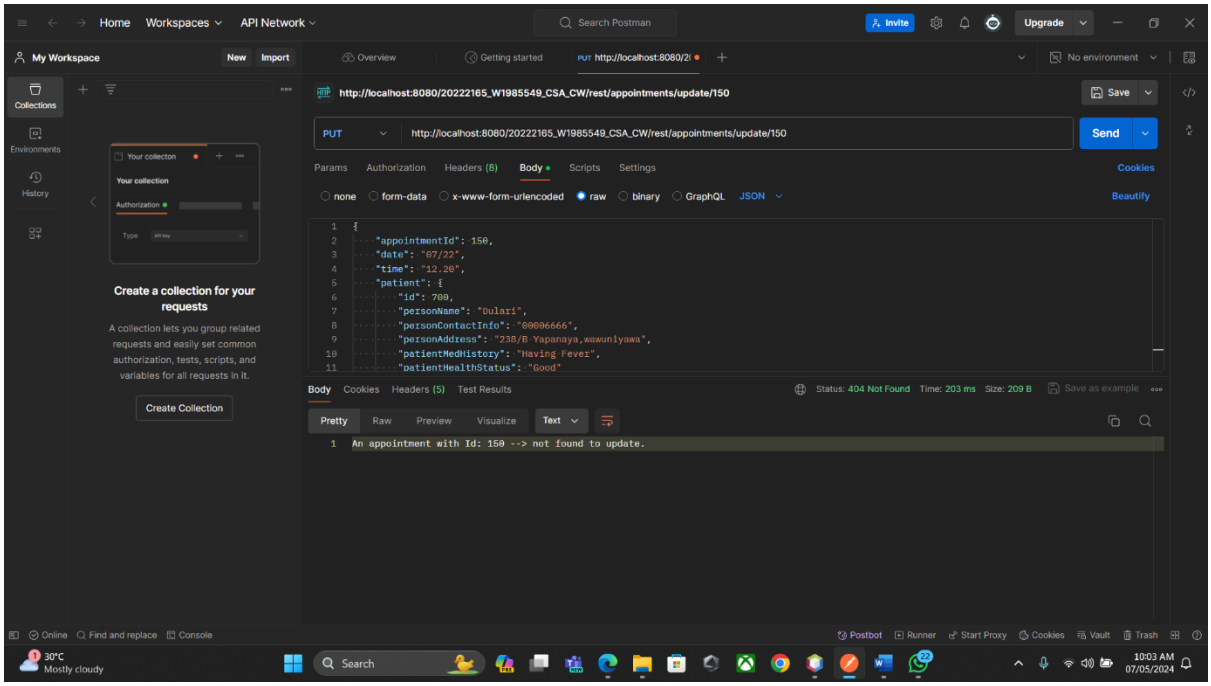


Figure 63: Postman PUT method Updating non- existing appoinments and giving errors

- DELETE

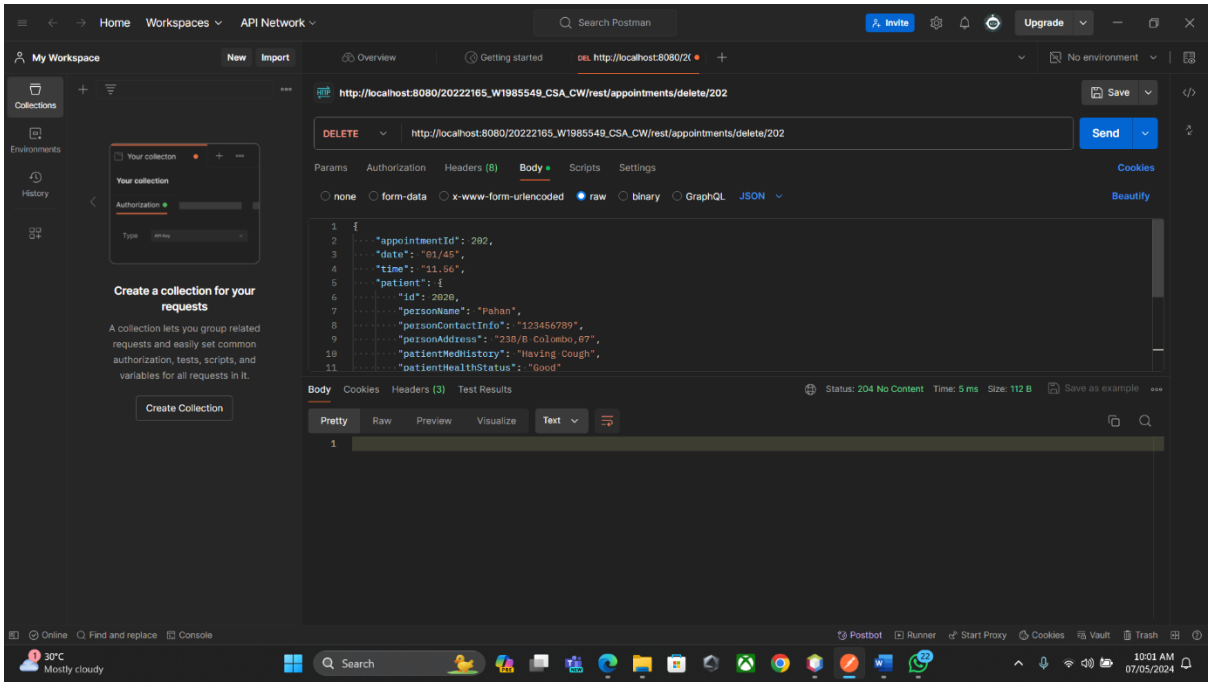


Figure 64: Postman DELETE method

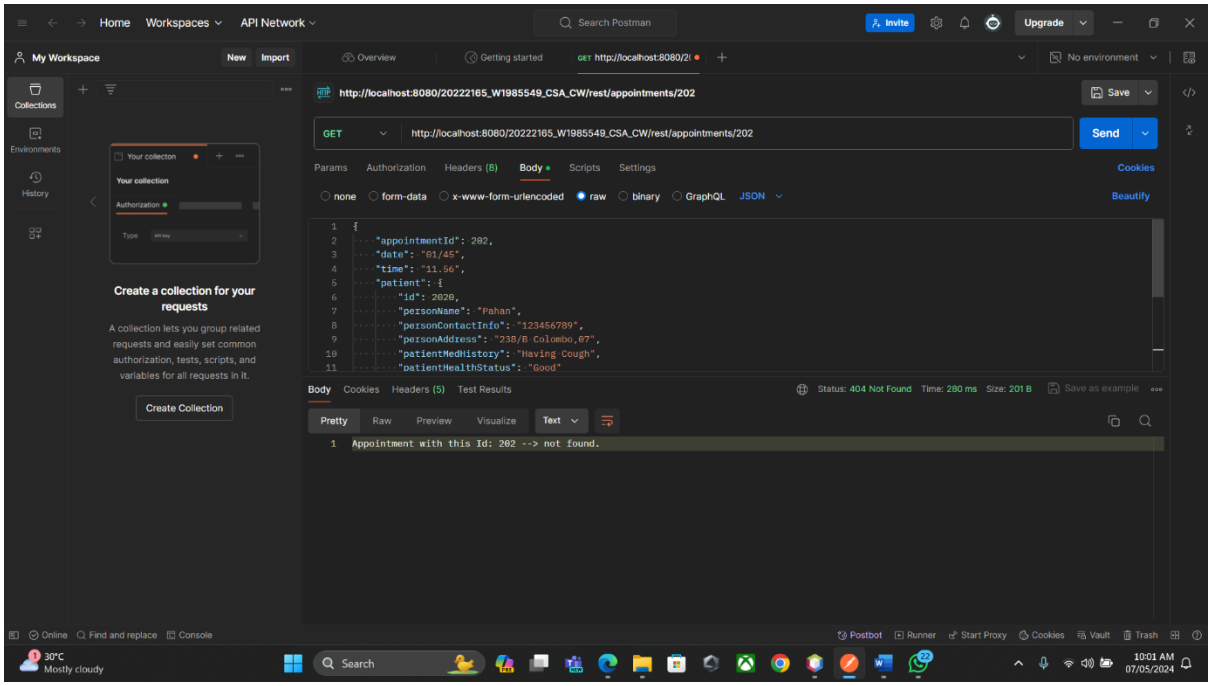


Figure 65: Postman DELETE method results with error printing

Medical Record Entity

- GET

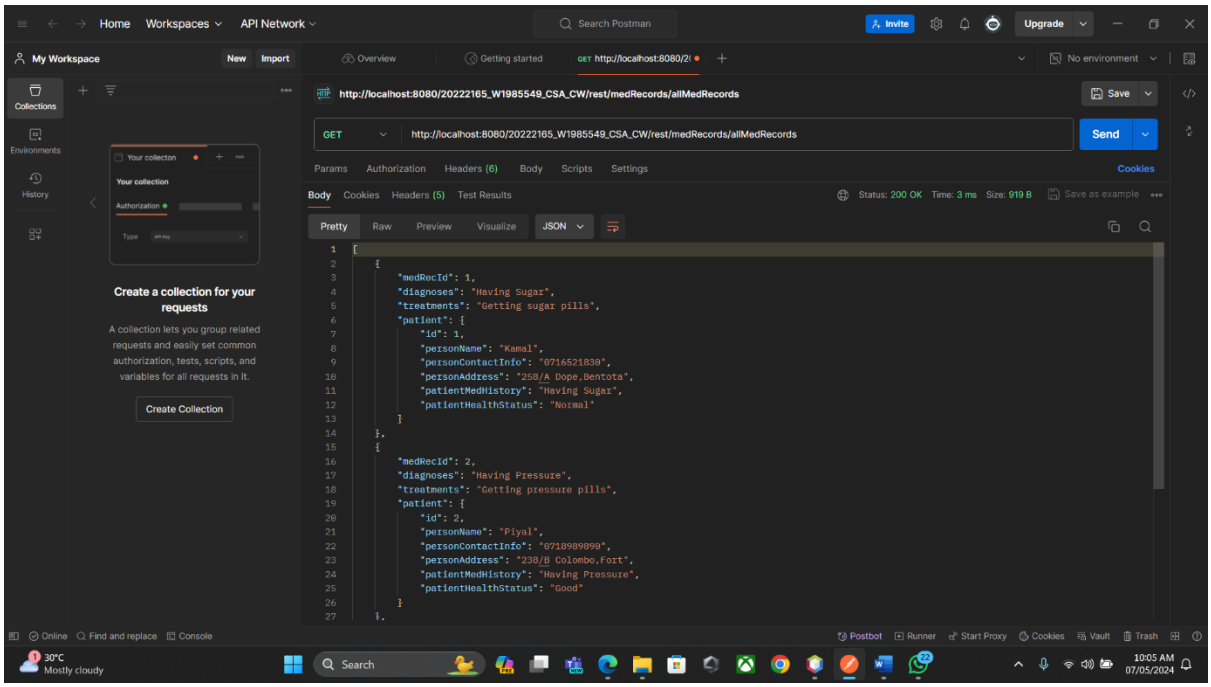


Figure 66: Postman GET method with all Medical Records

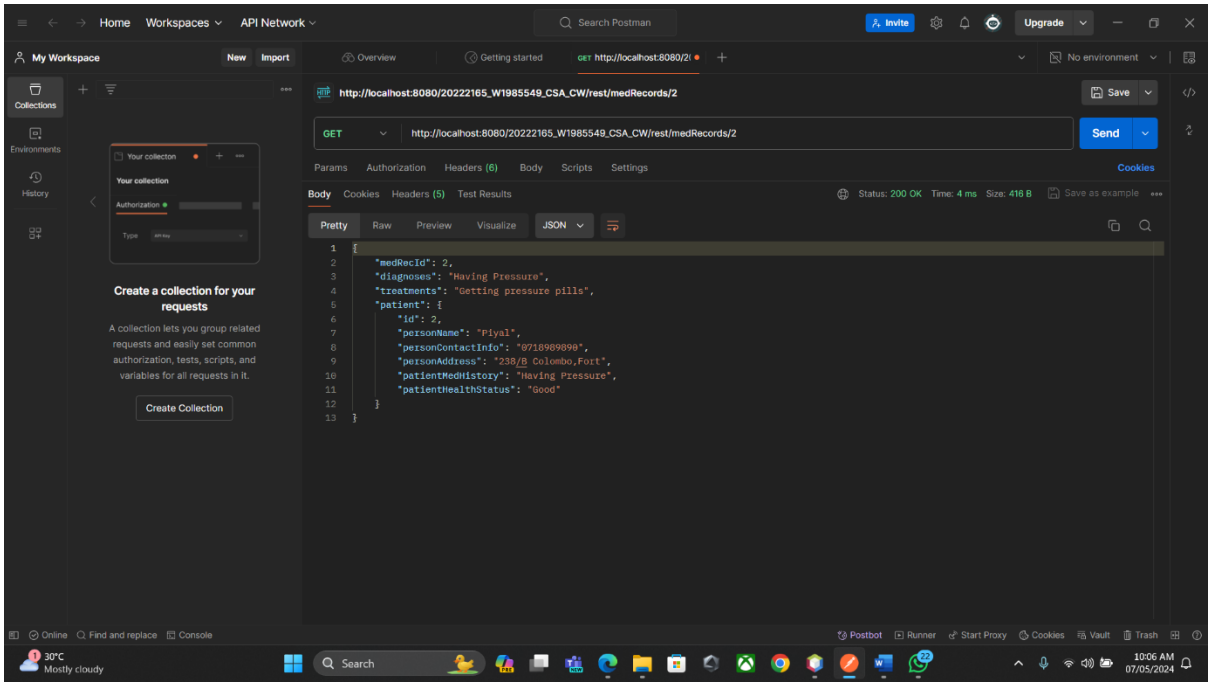


Figure 67: Postman GET method with Medical Record by id

- POST

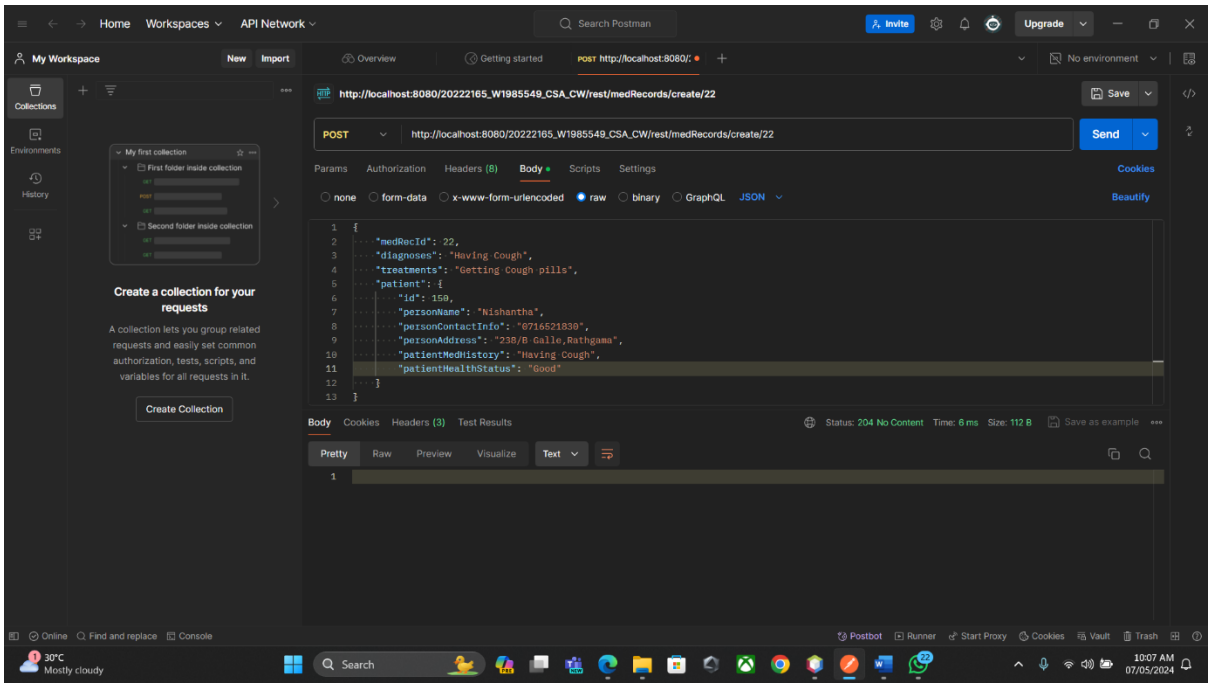


Figure 68: Postman POST method with adding Medical Records

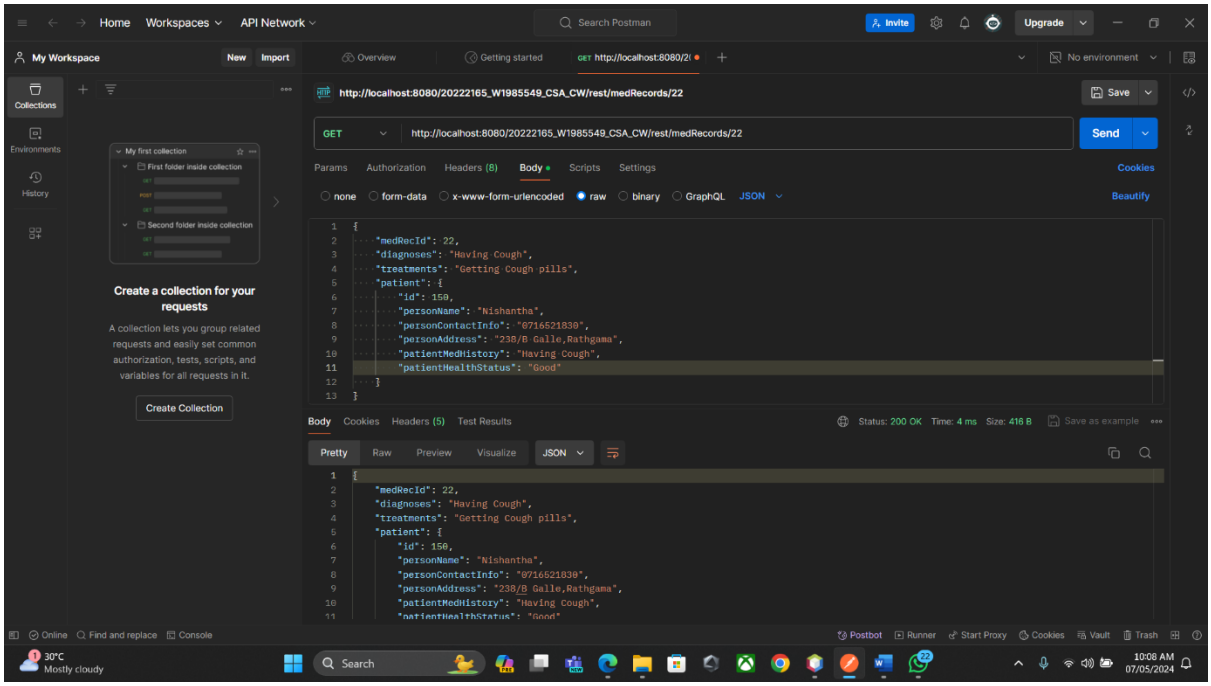


Figure 69: Postman POST method with adding Medical Records results

- PUT

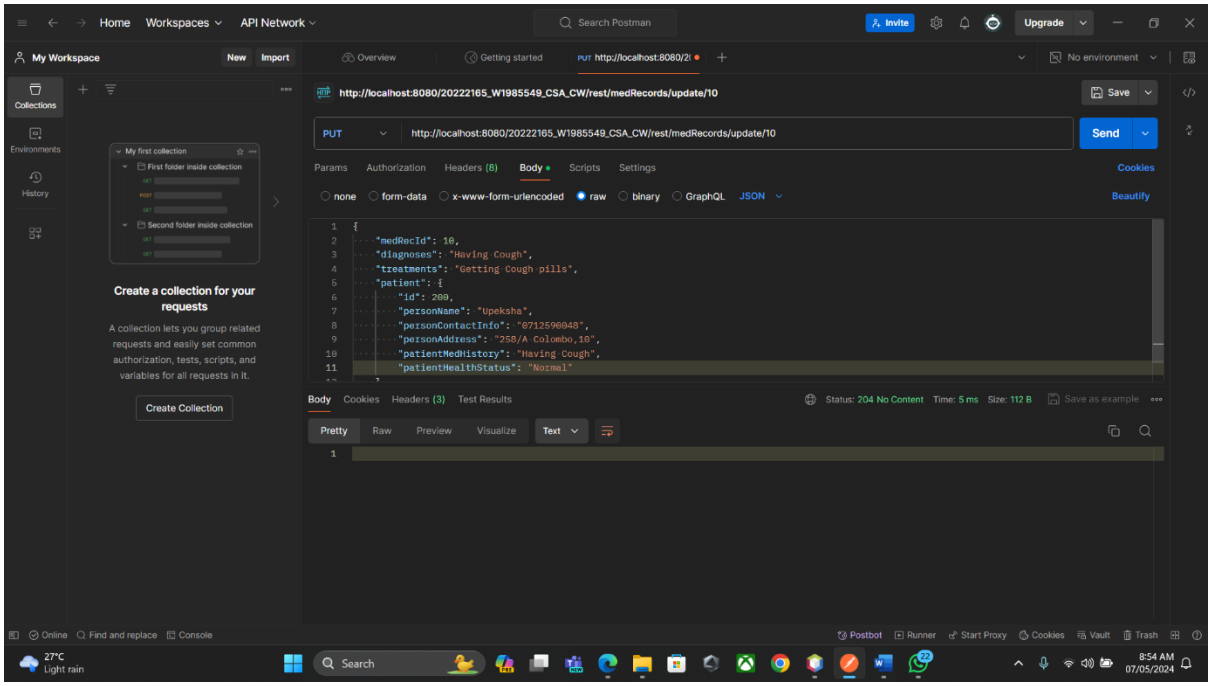


Figure 70: Postman PUT method with updating existing Medical Records

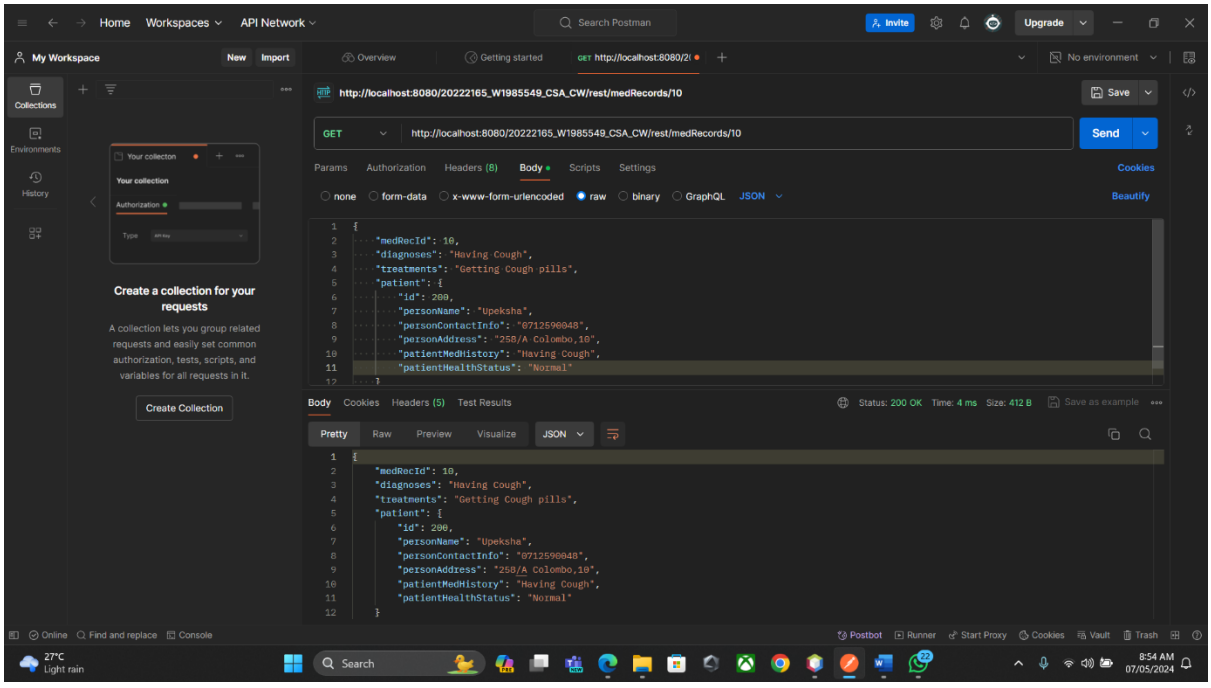


Figure 71: Postman PUT method with updating existing Medical Records results

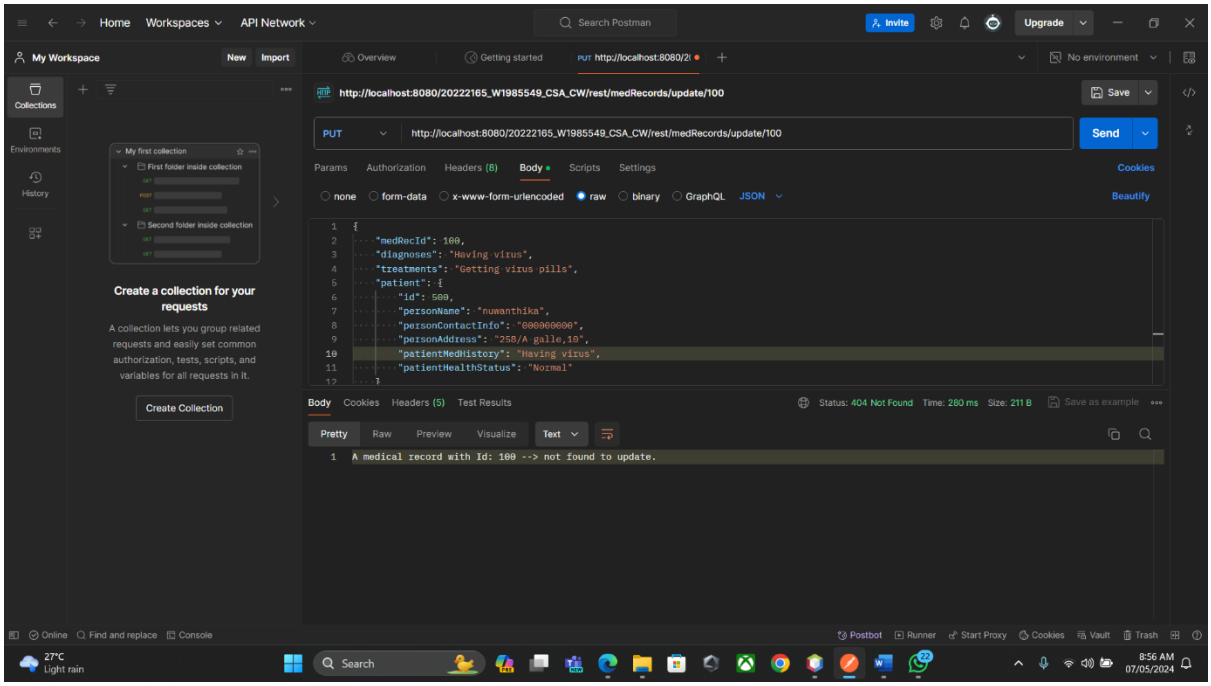


Figure 72: Postman PUT method with updating non-existing Medical Records with error printing

- DELETE

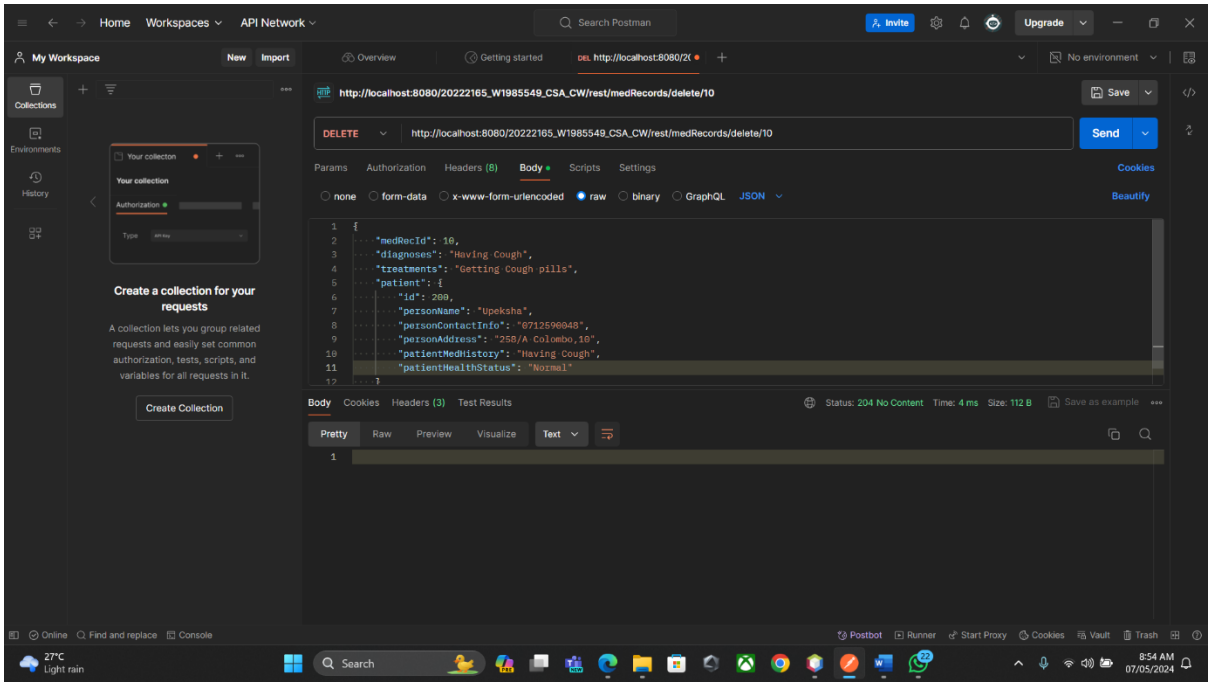


Figure 73: Postman DELETE method for medical records

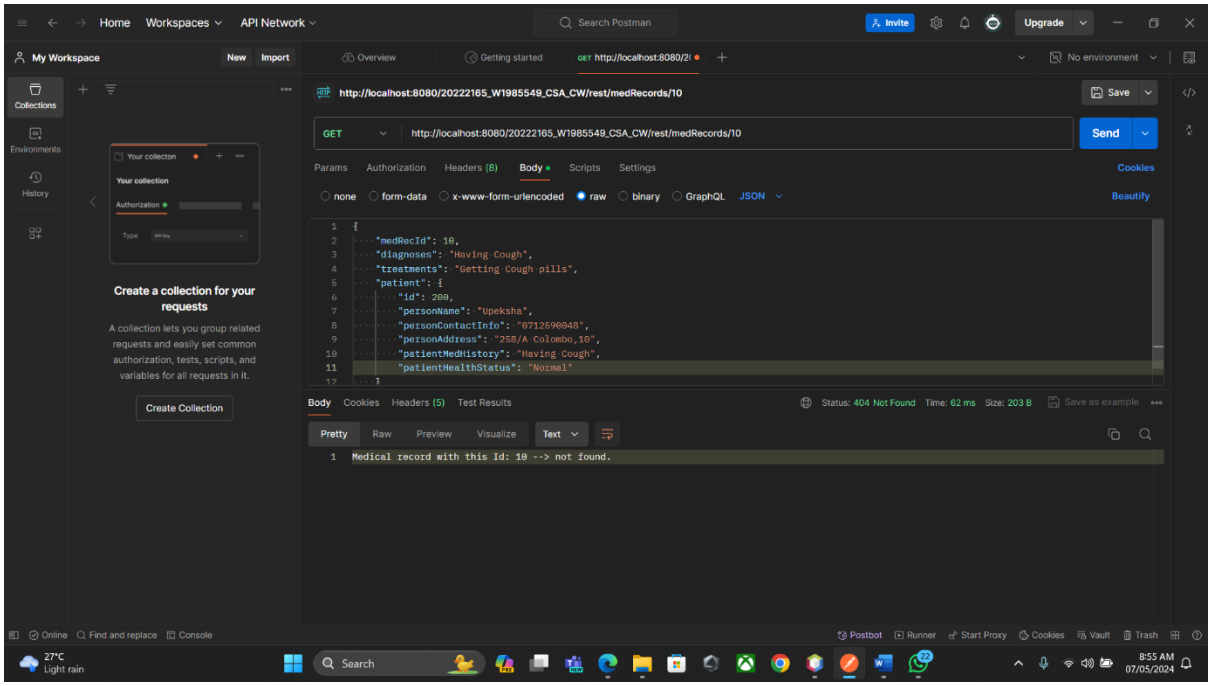


Figure 74: Postman DELETE method for medical records results with error printing

Prescription Entity

- GET

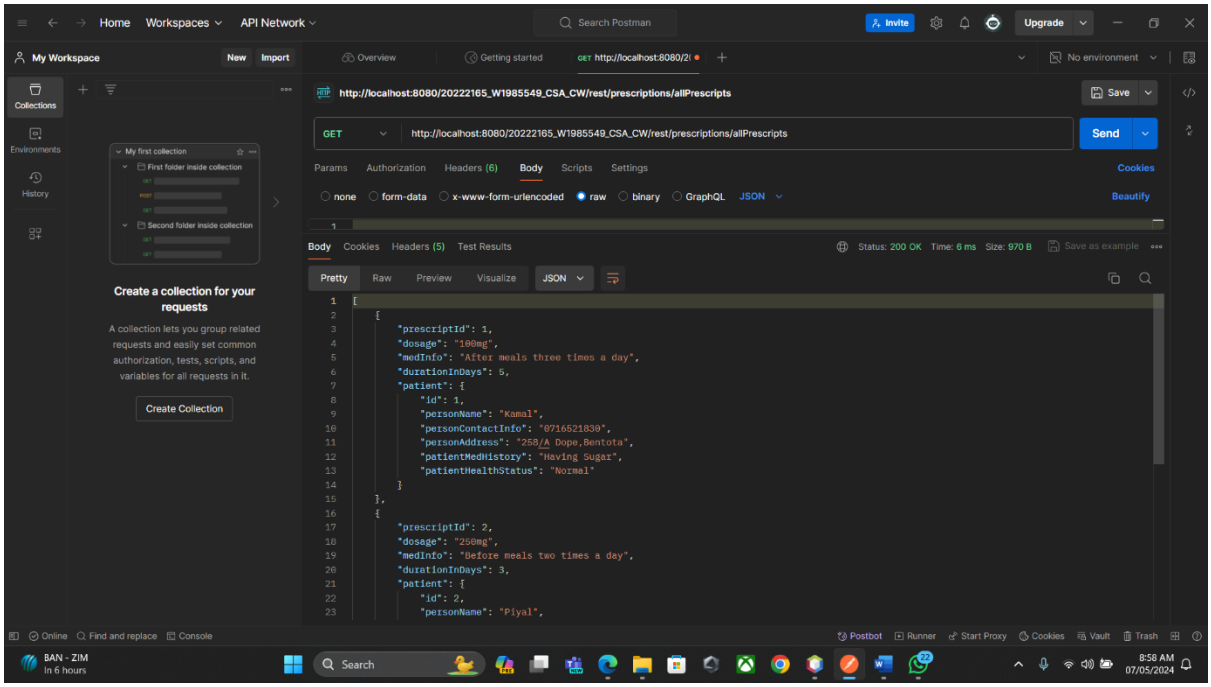


Figure 75: Postman GET method by all prescriptions

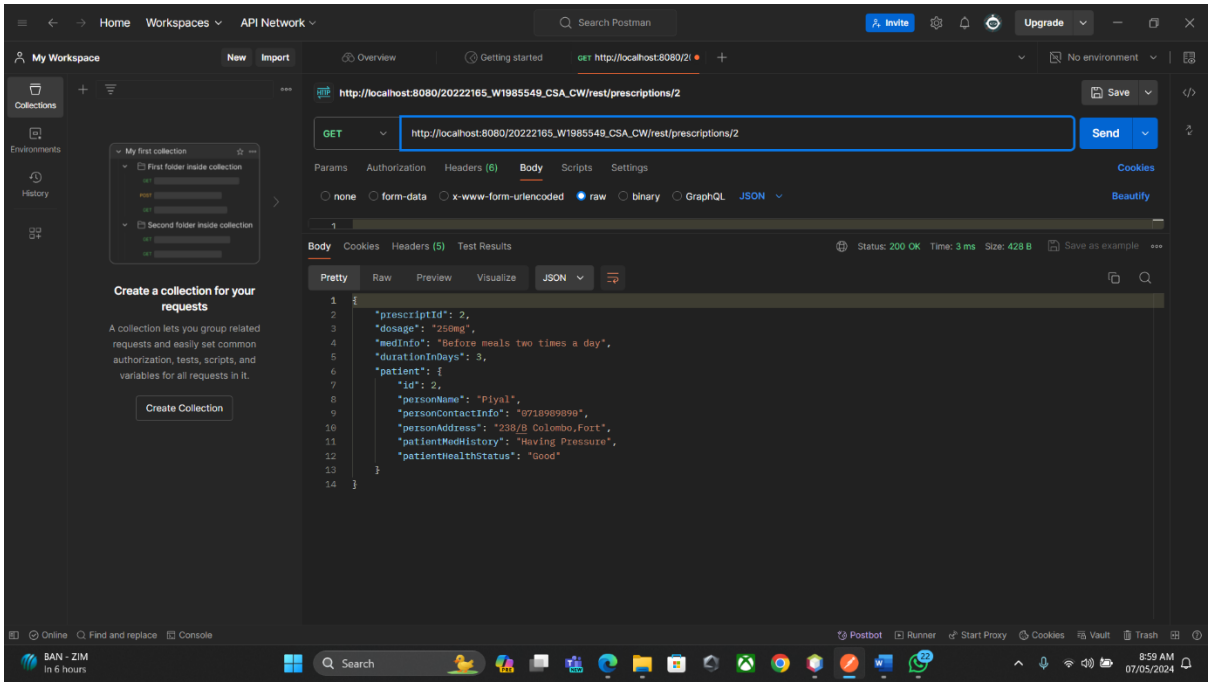


Figure 76: Postman GET method prescriptions by id

- POST

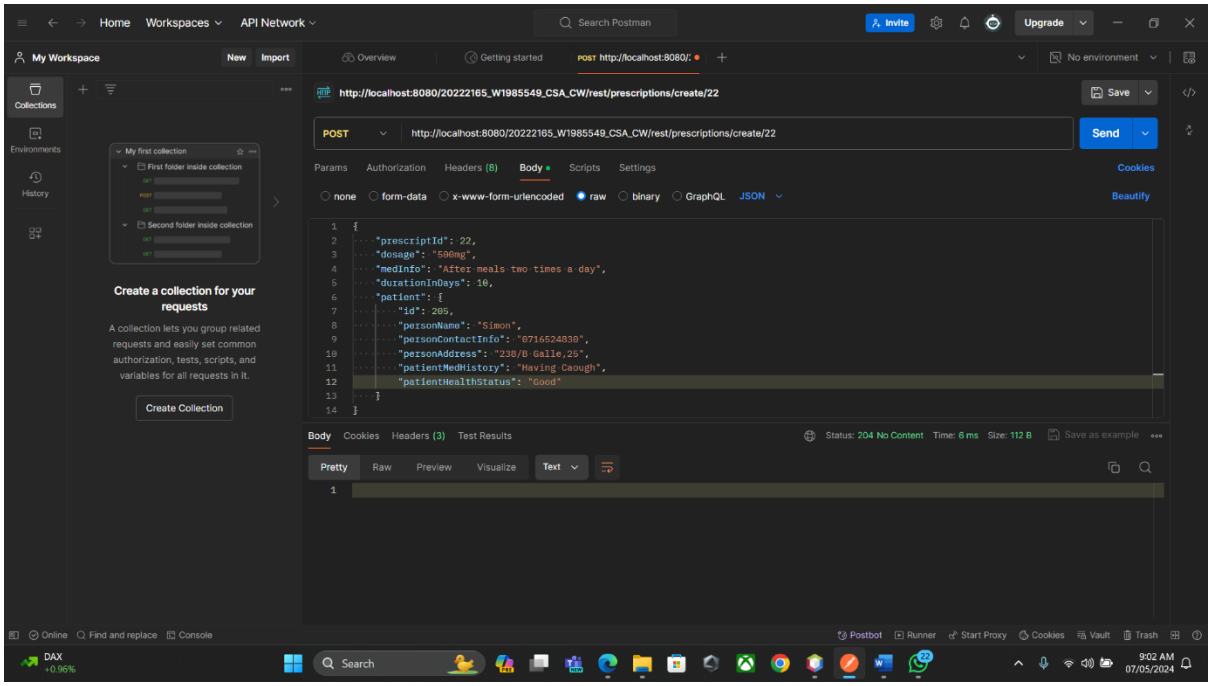


Figure 77: Postman POST method by adding prescriptions

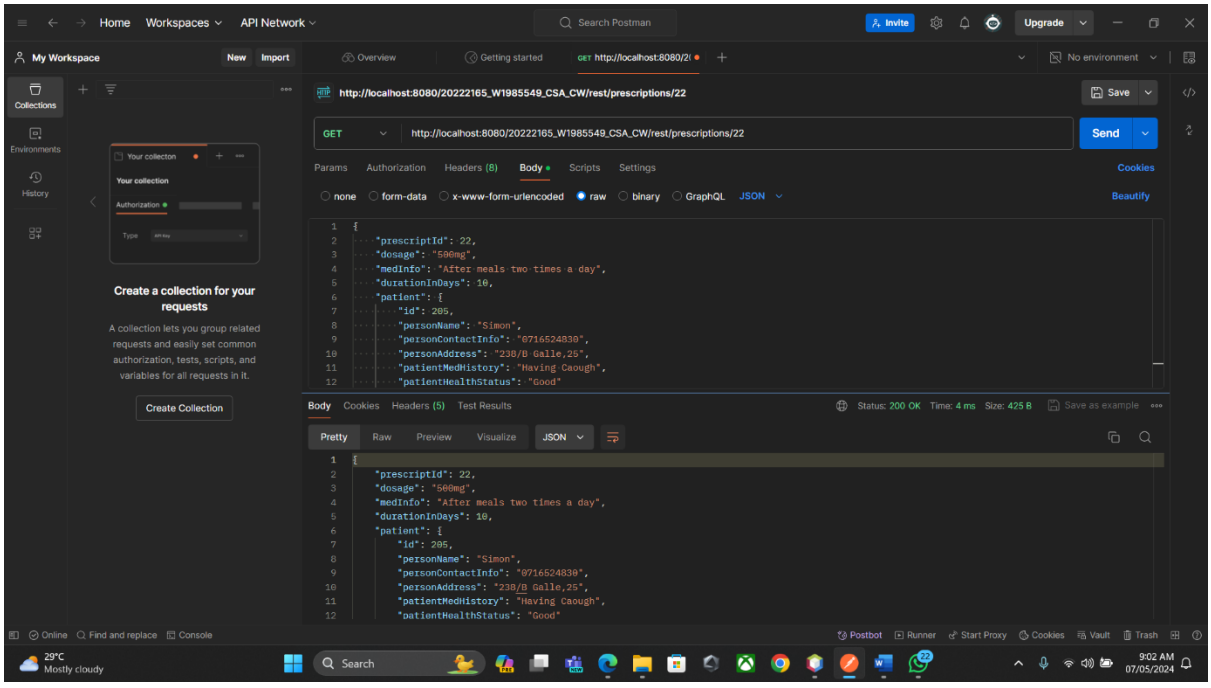


Figure 78: Postman POST method by adding prescriptions results

- PUT

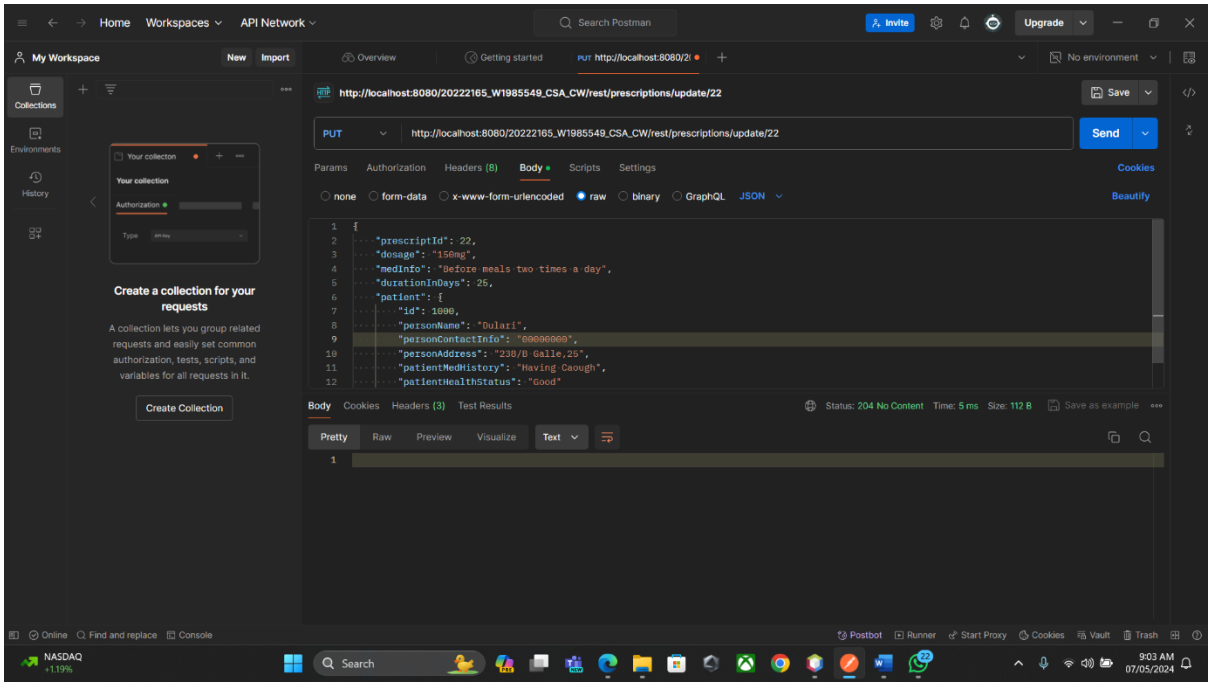


Figure 79: Postman PUT method by updating existing prescriptions

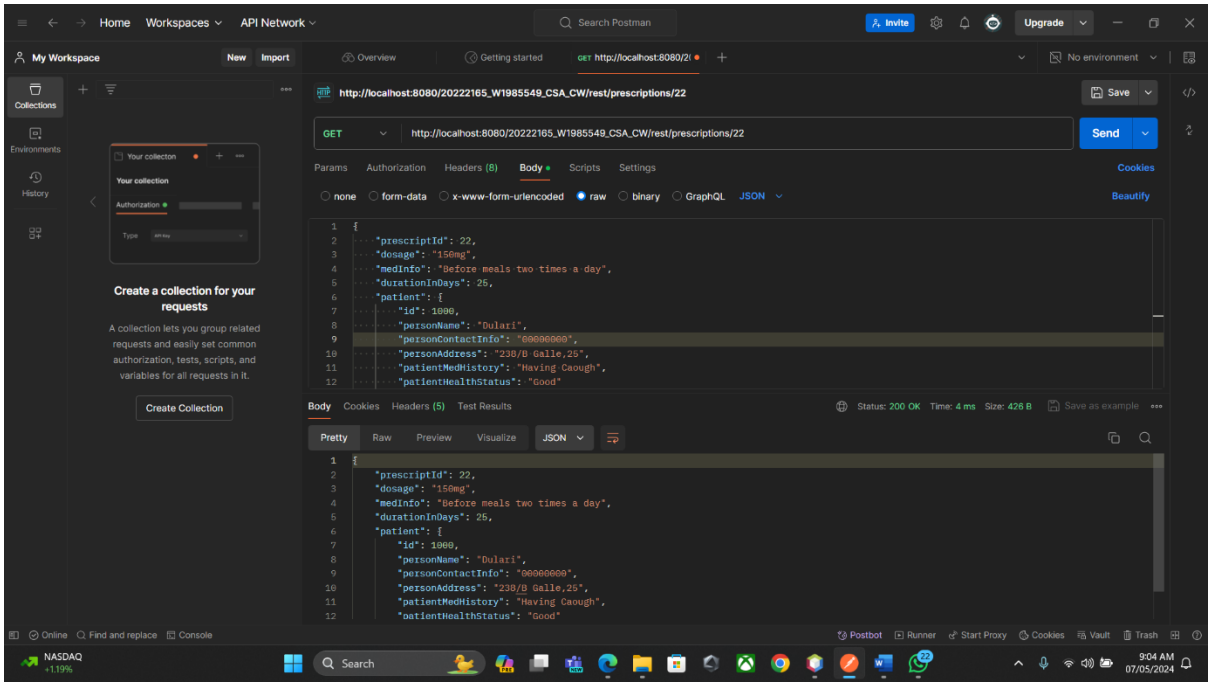


Figure 80: Postman PUT method by updating existing prescriptions results

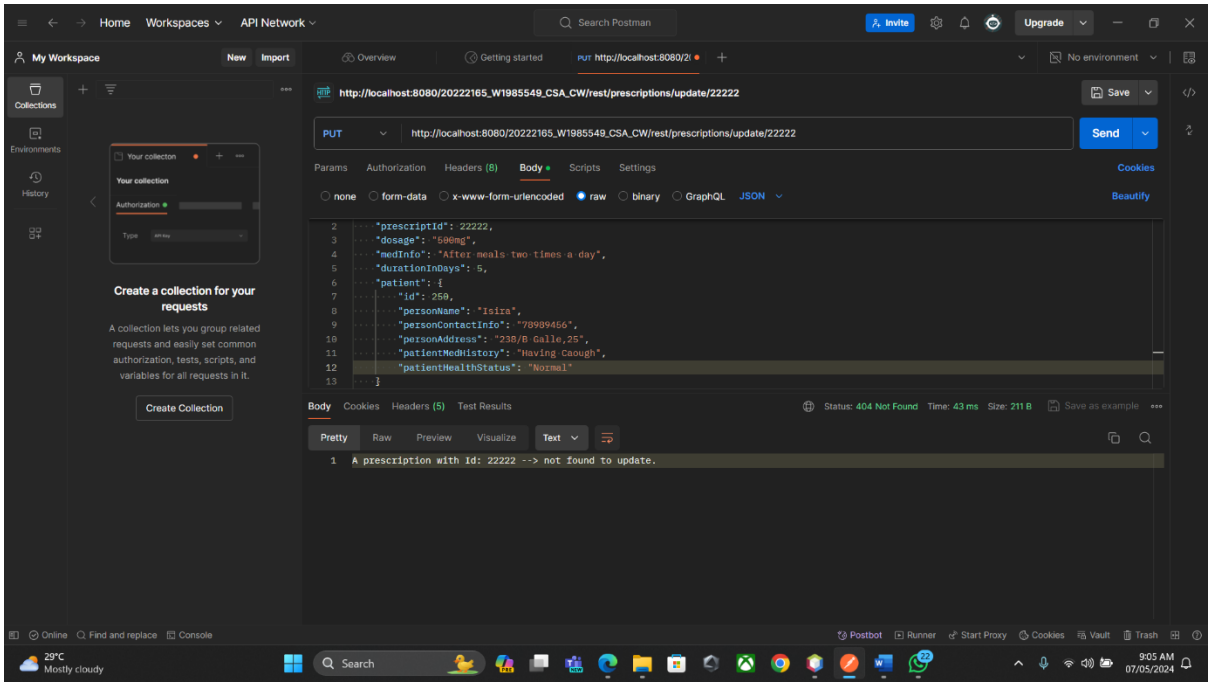


Figure 81: Postman PUT method by updating non-existing prescriptions and printin errors

- DELETE

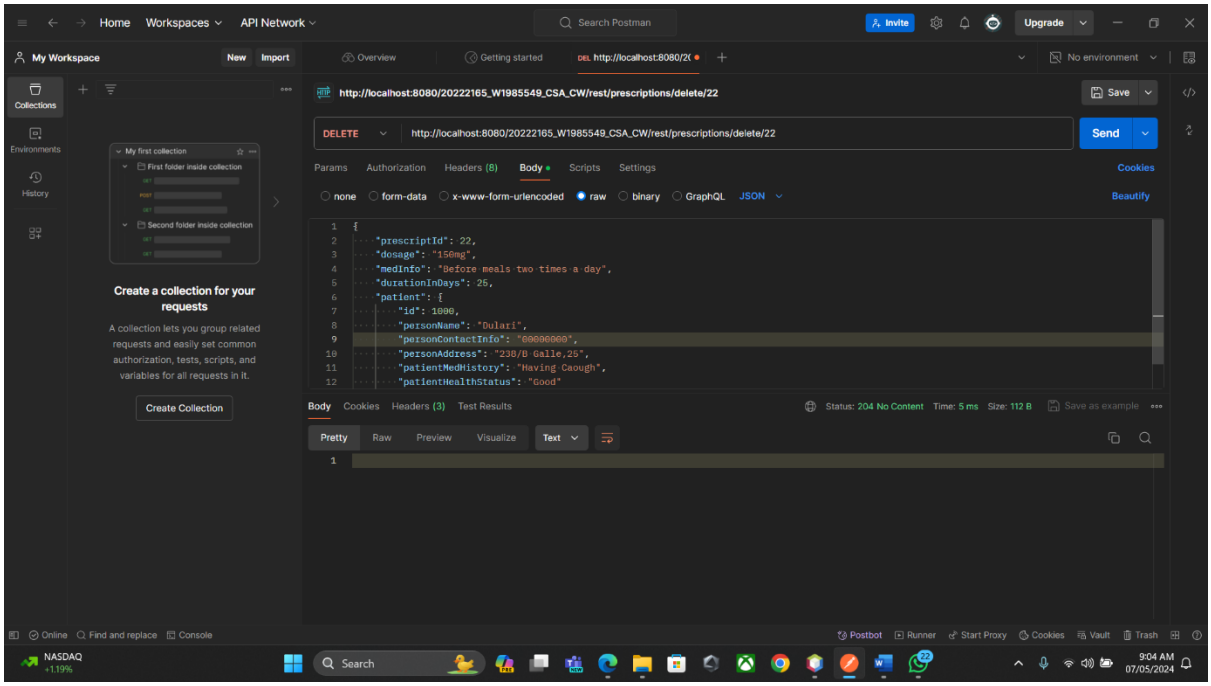


Figure 82: Postman DELETE method

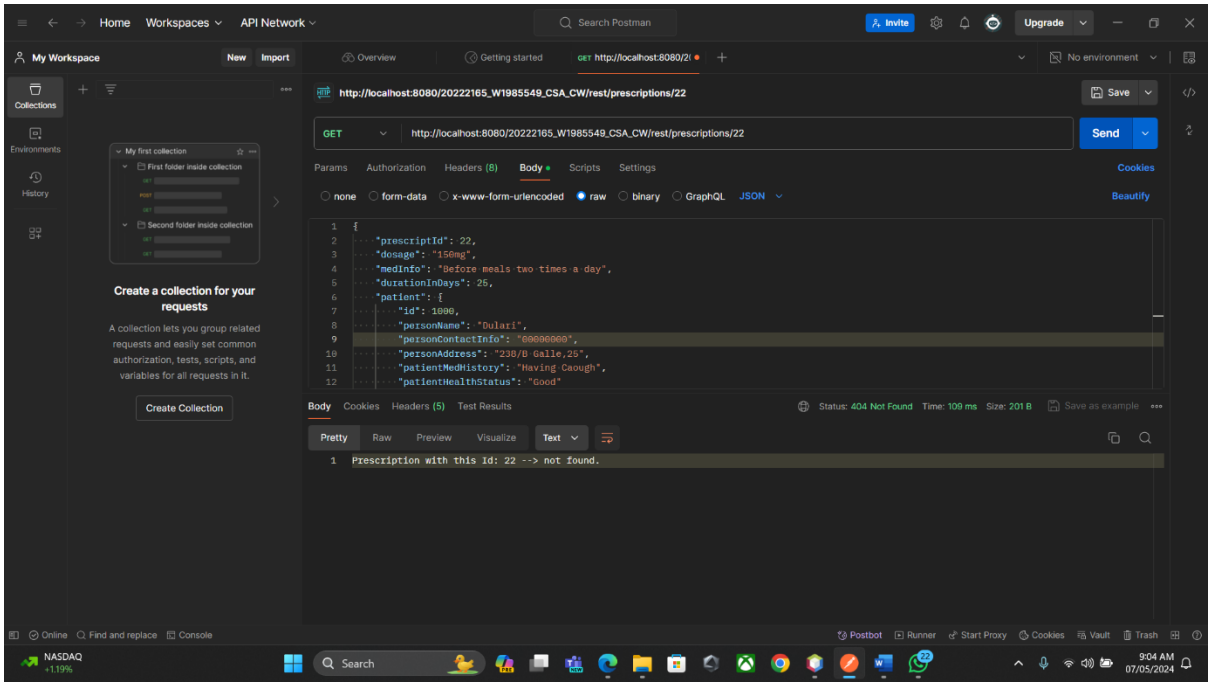


Figure 83: Postman DELETE method results and error printing

Billing Entity

- GET

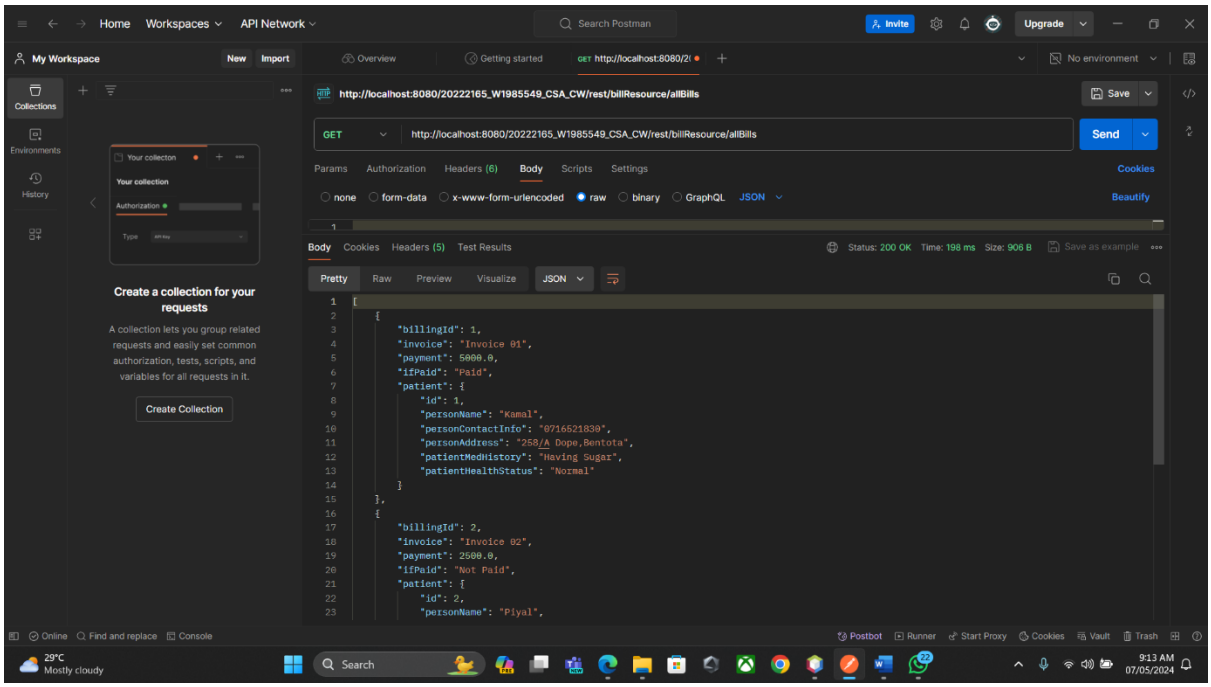


Figure 84: Postman GET method with all bills

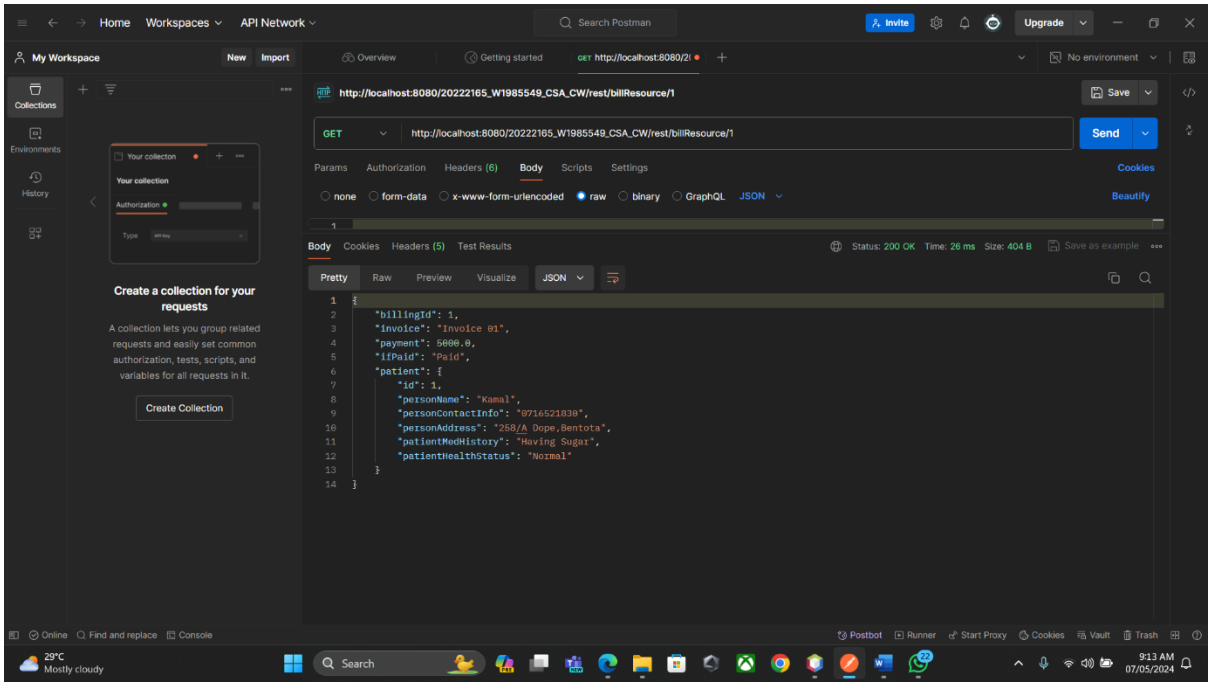


Figure 85: Postman GET method with bill id

- POST

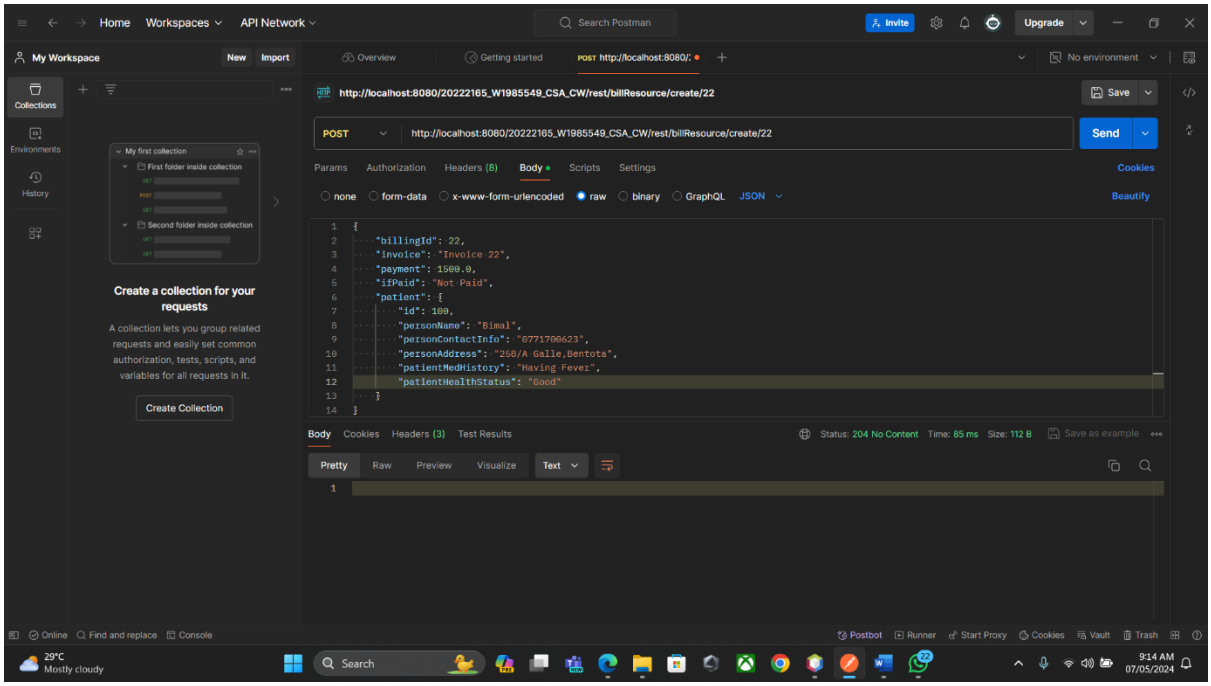


Figure 86: Postman POST method with adding bills

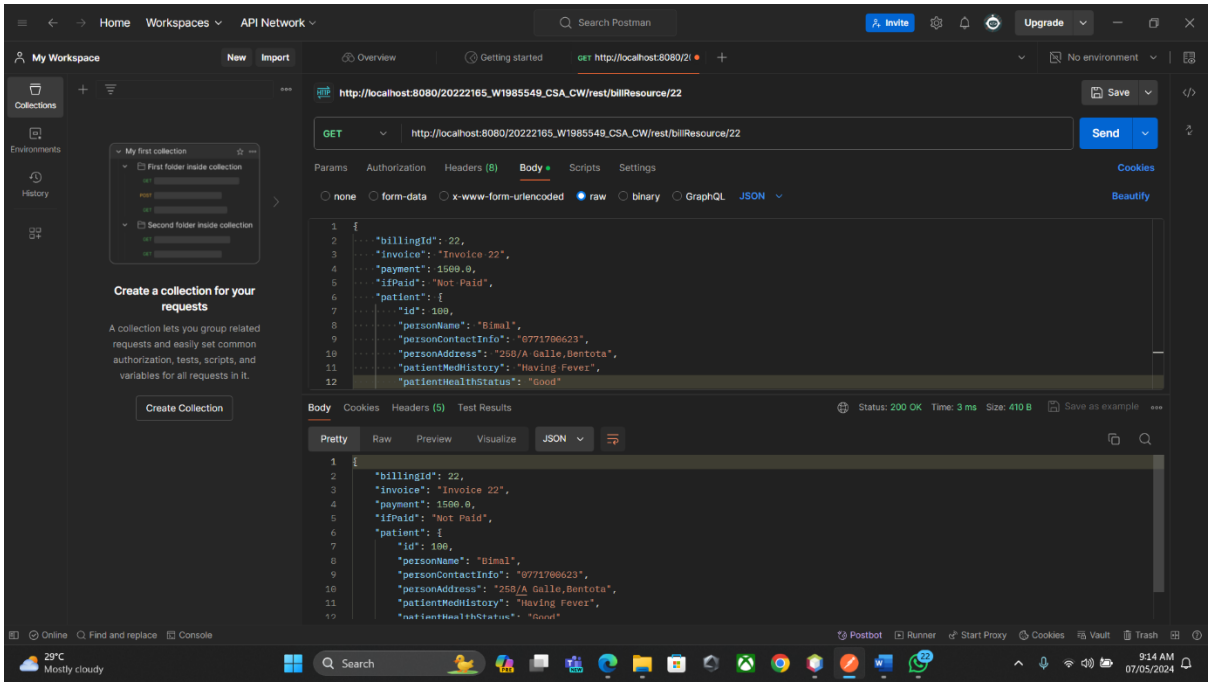


Figure 87: Postman POST method with adding bills results

- PUT

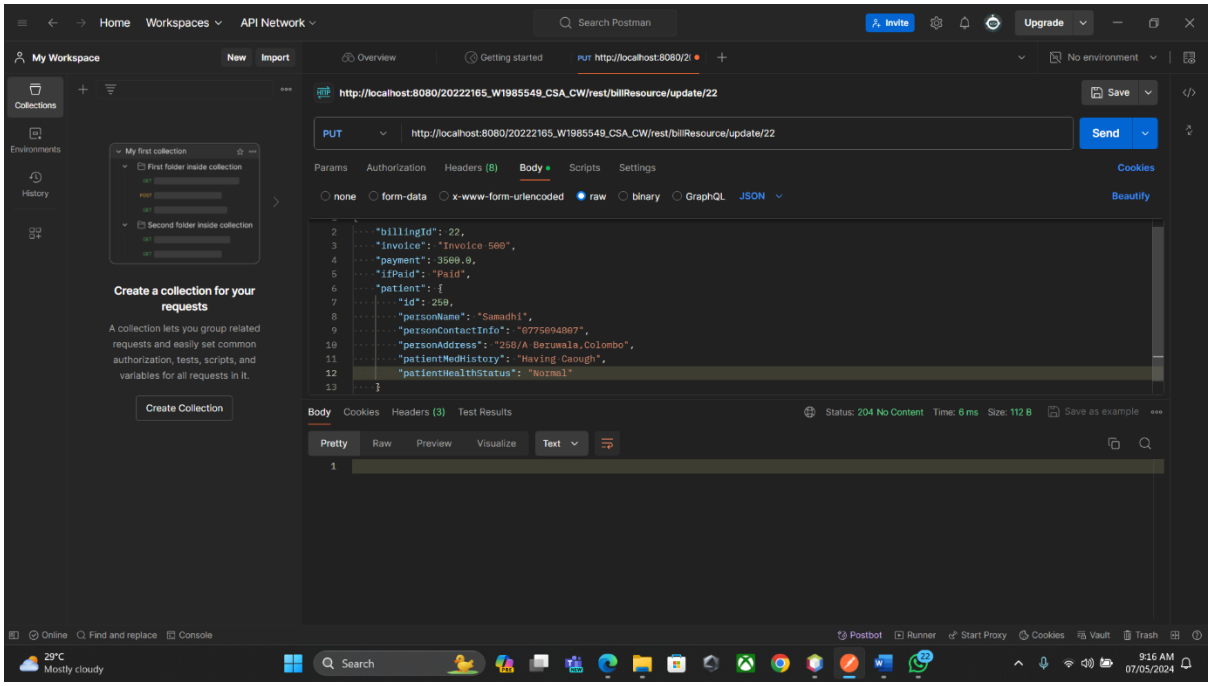


Figure 88: Postman PUT method with updating existing bill

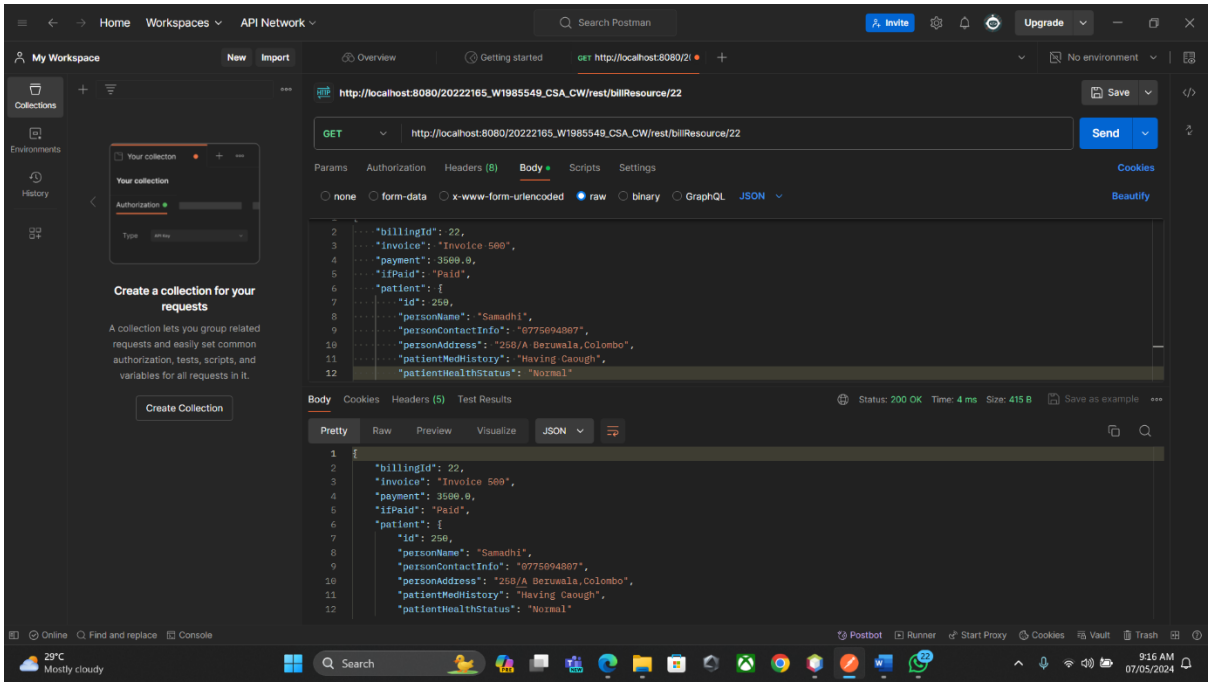


Figure 89: Postman PUT method with updating existing bill results

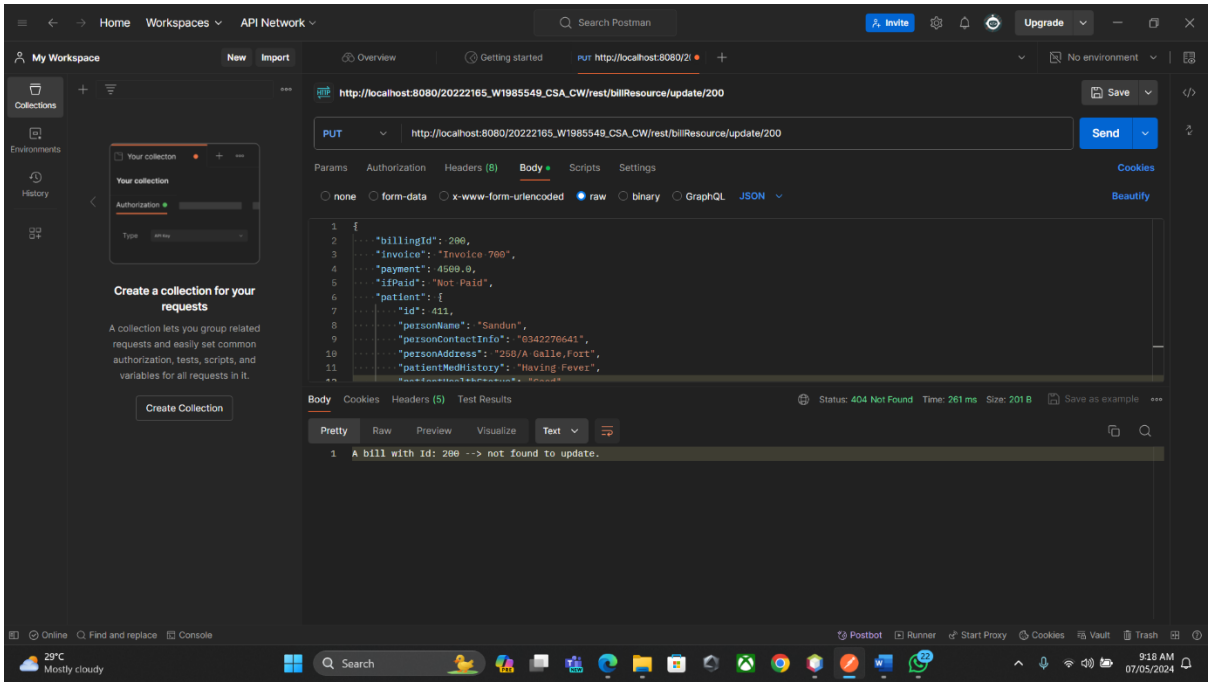


Figure 90: Postman PUT method with updating non-existing bill and error printing

- DELETE

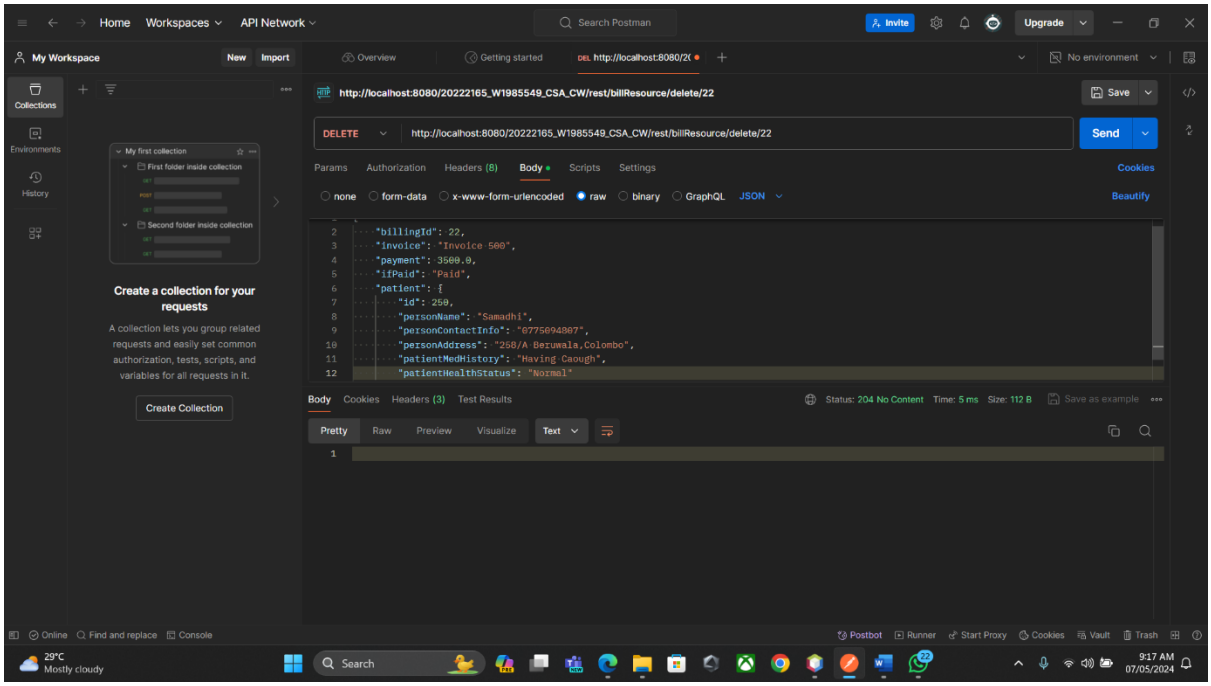


Figure 91: Postman DELETE method

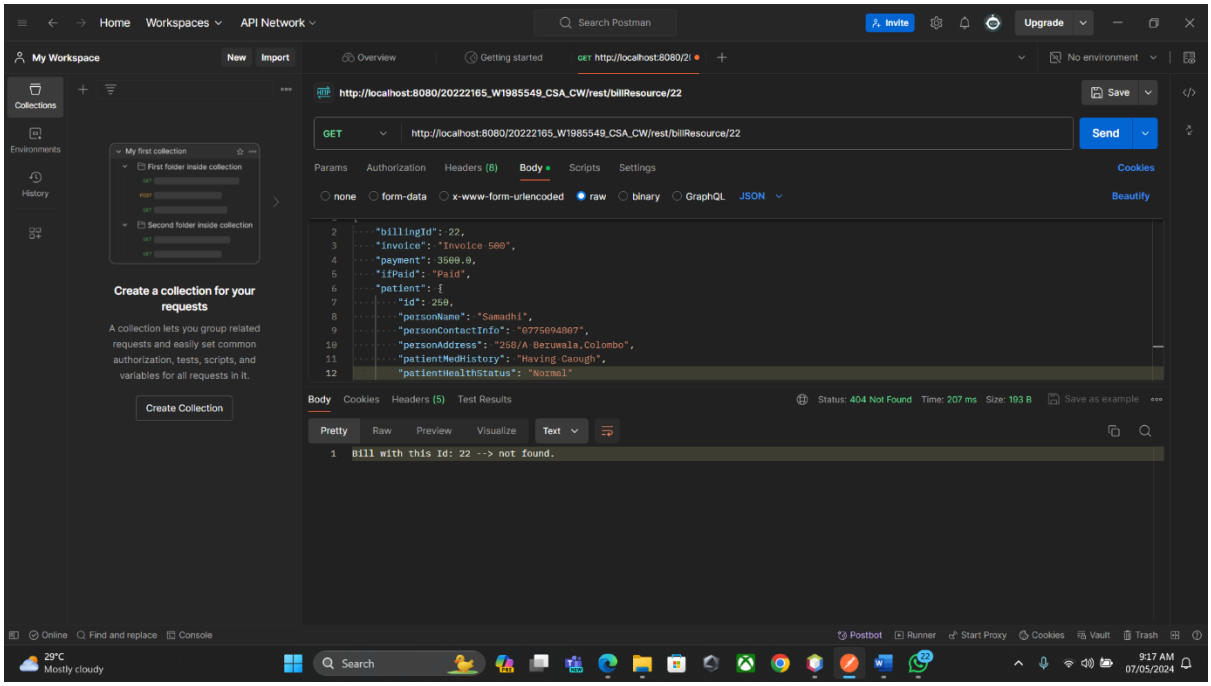


Figure 92: Postman DELETE method results with printing error

Conclusion

In this Health System API all the methods in model classes and DAO classes are worked properly. Not only that all the RESTful HTTP methods are functioning as expected and they are handling all the error exceptions perfectly. Therefore, we can conclude that, this Health API is well implemented and working as expected.