

# 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](https://drive.google.com/file/d/1r6Y_caVIt_bHZCSb8QT0voIZrmGIqjYF/view?usp=sharing)

## Table of Contents

Introduction .....	2
Overview of the project .....	2
API endpoints and HTTP methods with paths .....	3
Implementation .....	7
Code Organization .....	7
Implementation of model classes, DAOs, and RESTful Resources classes .....	8
Person, PersonDAO, and PersonResource .....	8
Patient, PatientDAO, and PatientResource .....	11
Doctor, DoctorDAO, and DoctorResource .....	14
Appointment, AppointmentDAO, AppointmentResource .....	16
MedicalRecords, MedicalRecordsDAO, MedicalRecordsResource .....	19
Prescription, PrescriptionDAO, PrescriptionResource .....	21
Billing, BillingDAO, BillingResource .....	24
Testing Results from Postman .....	27
Conclusion .....	61

## 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/{appId}  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 ‘appId’.
POST	/20222165_W1985549_CSA_CW/rest/appointments/create/{appId}  This path will give the ability to create a new appointment with relevant details(dataset) using a new ‘appId’.
PUT	/20222165_W1985549_CSA_CW/rest/appointments/update/{appId}

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

## 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.

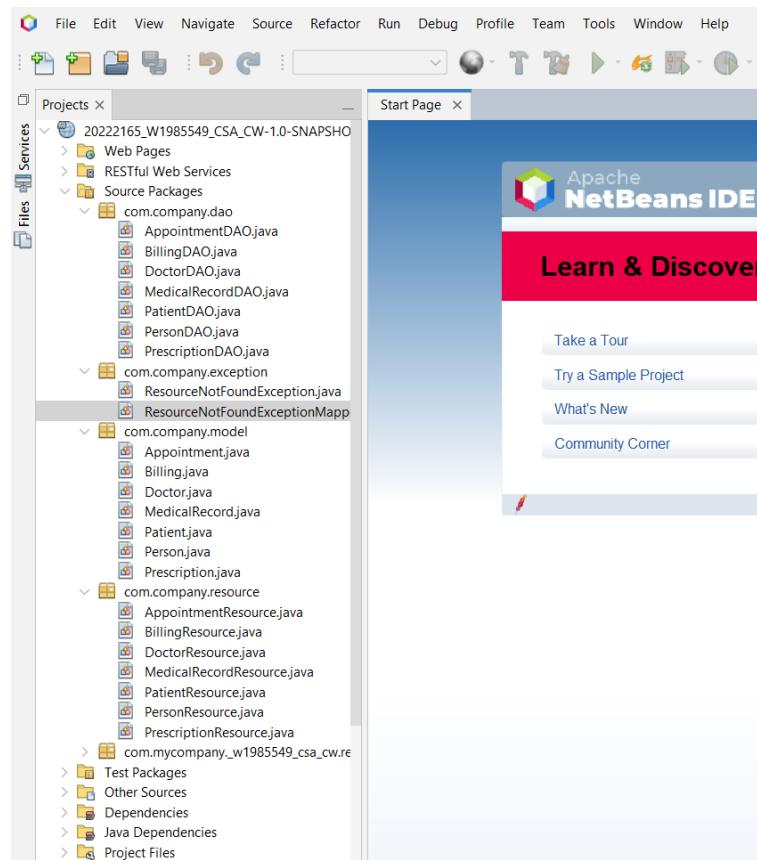


Figure 1: Code Organization

## 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.

The screenshot shows a Java code editor interface with the following details:

- Tab Bar:** Shows "Start Page X", "Person.java X", "PersonDAO.java X", and "PersonResource.java X".
- Toolbar:** Includes icons for Source, History, and various file operations like Open, Save, Find, and Copy.
- Code Area:** Displays the following Java code for the `Person` class:

```
public class Person {
    //Initializing the private attributes
    private int id;
    private String personName;
    private String personContactInfo;
    private String personAddress;

    //Making default constructor
    public Person(){}
    //Making the constructor
    public Person(int id, String personName, String personContactInfo, String personAddress){
        this.id = id;
        this.personName = personName;
        this.personContactInfo = personContactInfo;
        this.personAddress = personAddress;
    }
    //Implementing all the Getters and Setters
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getPersonName() {
        return personName;
    }
}
```

Figure 2:Code snippet for Person model class

The screenshot shows a Java code editor with the following code snippet for the PersonDAO class:

```
@  
public class PersonDAO {  
    //Defining logger  
    private static final Logger LOGGER = Logger.getLogger(PersonDAO.class.getName());  
    //Initializing the arraylist  
    private static List<Person> persons = new ArrayList<>();  
    //Defining the details static block  
    static{  
        persons.add(new Person(id: 1, personName: "isira", personContactInfo: "Mobile: 0710000000", personAddress: "Athurugiriya,Colombo"));  
        persons.add(new Person(id: 2, personName: "pahan", personContactInfo: "Mobile: 0714545454", personAddress: "Alawwa,Kagalla"));  
        persons.add(new Person(id: 3, personName: "nuwandi", personContactInfo: "Mobile: 0713333333", personAddress: "Anuradhapura,New Town")  
    }  
    //Create Operation  
    public void createPerson(Person person){  
        persons.add(person);  
    }  
    //Read Operation for all persons.  
    public List<Person> getAllPersons(){  
        //return new ArrayList<>(persons);  
        return persons;  
    }  
    //Read Operation for each person.  
    public Person getPersonById(int id){  
        for(Person person : persons){  
            if(person.getId() == id){  
                return person;  
            }  
        }  
    }  
}
```

Figure 3: Code snippet for PersonDAO class

The screenshot shows a Java code editor with the following code snippet for the PersonResource class:

```
@  
@Path("/persons")  
public class PersonResource {  
    //Define the logger using logger factory  
    private static final Logger LOGGER = LoggerFactory.getLogger(PersonResource.class);  
    //Making the object from PersonDAO class  
    private PersonDAO personDAO = new PersonDAO();  
    //Defining the @GET http method for getting all persons  
    @GET  
    @Path("/allPersons")  
    @Produces(MediaType.APPLICATION_JSON)  
    public List<Person> getAllPersons() {  
        return personDAO.getAllPersons();  
    }  
    //Defining the @GET http method for getting a person by ID  
    @GET  
    @Path("/{personId}")  
    @Produces(MediaType.APPLICATION_JSON)  
    public Person getPersonById(@PathParam("personId") int personId) {  
        Person personObj = personDAO.getPersonById(id: personId);  
        if(personObj != null){  
            LOGGER.info(string: "Getting person by Id: {}", o: personId);  
            return personObj;  
        }else{  
            throw new ResourceNotFoundException("Person with this Id: " + personId + " --> not found.");  
        }  
    }  
}
```

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

```

61     //Defining the @POST http method for create a person by ID
62     @POST
63     @Path("/create/{personId}")
64     @Consumes(MediaType.APPLICATION_JSON)
65     public void createPerson(Person person) {
66         personDAO.createPerson(person);
67     }
68
69     //Defining the @PUT http method for updating a person by existing ID
70     @PUT
71     @Path("/update/{personId}")
72     @Consumes(MediaType.APPLICATION_JSON)
73     public void updatePerson(@PathParam("personId") int personId, Person updatedPerson) {
74
75         LOGGER.info(string: "Updating person by ID: {}", o: personId);
76         Person existingPerson = personDAO.getPersonById(id: personId);
77
78         if (existingPerson != null) {
79             updatedPerson.setId(id: personId);
80             personDAO.updatePerson(updatedPerson);
81         } else {
82             throw new ResourceNotFoundException("A person with Id: " + personId + " --> not found to update.");
83         }
84     }
85
86     //Defining the @DELETE http method for remove a person by existing ID
87     @DELETE
88     @Path("/delete/{personId}")
89     public void deletePerson(@PathParam("personId") int personId) {
90         personDAO.deletePerson(id: personId);
91     }

```

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(), updatePatient(), and deletePatient() methods. And the RESTful HTTP methods which @GET, @POST, @PUT, @DELETE are implemented in the PatientResource class.

```

9  * @author Sahan Dharmarathna
10 */
11 public class Patient extends Person{
12
13     //Initializing the private attributes
14     private String patientMedHistory;
15     private String patientHealthStatus;
16
17     //Making default constructor
18     public Patient(){
19         super(id: 0, personName: " ", personContactInfo: " ", personAddress: " ");
20         this.patientMedHistory = " ";
21         this.patientHealthStatus = " ";
22     }
23
24     //Making the constructor
25     public Patient(int id, String personName, String personContactInfo, String personAddress, String patientMedHistory, String patientHealthStatus) {
26         super(id, personName, personContactInfo, personAddress);
27         this.patientMedHistory = patientMedHistory;
28         this.patientHealthStatus = patientHealthStatus;
29     }
30
31     //Implementing all the Getters and Setters
32
33     public String getPatientMedHistory() {
34         return patientMedHistory;
35     }
36
37     public void setPatientMedHistory(String patientMedHistory) {
38         this.patientMedHistory = patientMedHistory;
39     }

```

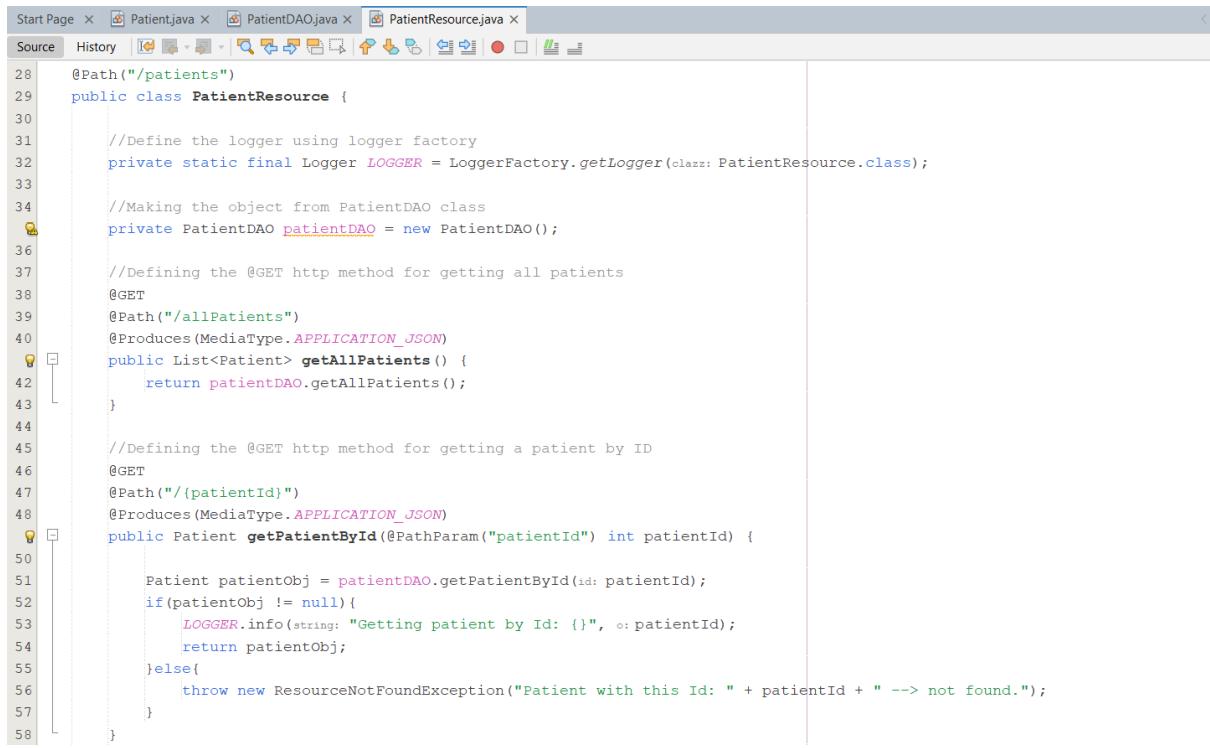
Figure 6: Code snippet for Patient model class

```

15  * @author Sahan Dharmarathna
16 */
17 public class PatientDAO extends PersonDAO{
18     //Defining logger
19     private static final Logger LOGGER = Logger.getLogger(name: PatientDAO.class.getName());
20
21     //Initializing the arraylist
22     private static List<Patient> patients = new ArrayList<>();
23
24     //Defining the details static block
25     static{
26         patients.add(new Patient(id: 1, personName: "Kamal", personContactInfo: "0716521830", personAddress: "258/A Dope,Bentota", patientMedHistory: " "));
27         patients.add(new Patient(id: 2, personName: "Piyal", personContactInfo: "0718989890", personAddress: "238/A Colombo,Port", patientMedHistory: " "));
28         patients.add(new Patient(id: 3, personName: "Namal", personContactInfo: "0775505500", personAddress: "248/A Galle,Port", patientMedHistory: " "));
29     }
30
31
32     //Create Operation
33     public void createPatient(Patient patient){
34         patients.add(patient);
35         super.createPerson(person: patient);
36     }
37
38     //Read Operation for all patients.
39     public List<Patient> getAllPatients(){
40         //return new ArrayList<>(patients);
41         return patients;
42     }
43
44
45

```

Figure 7: Code snippet for PatientDAO class

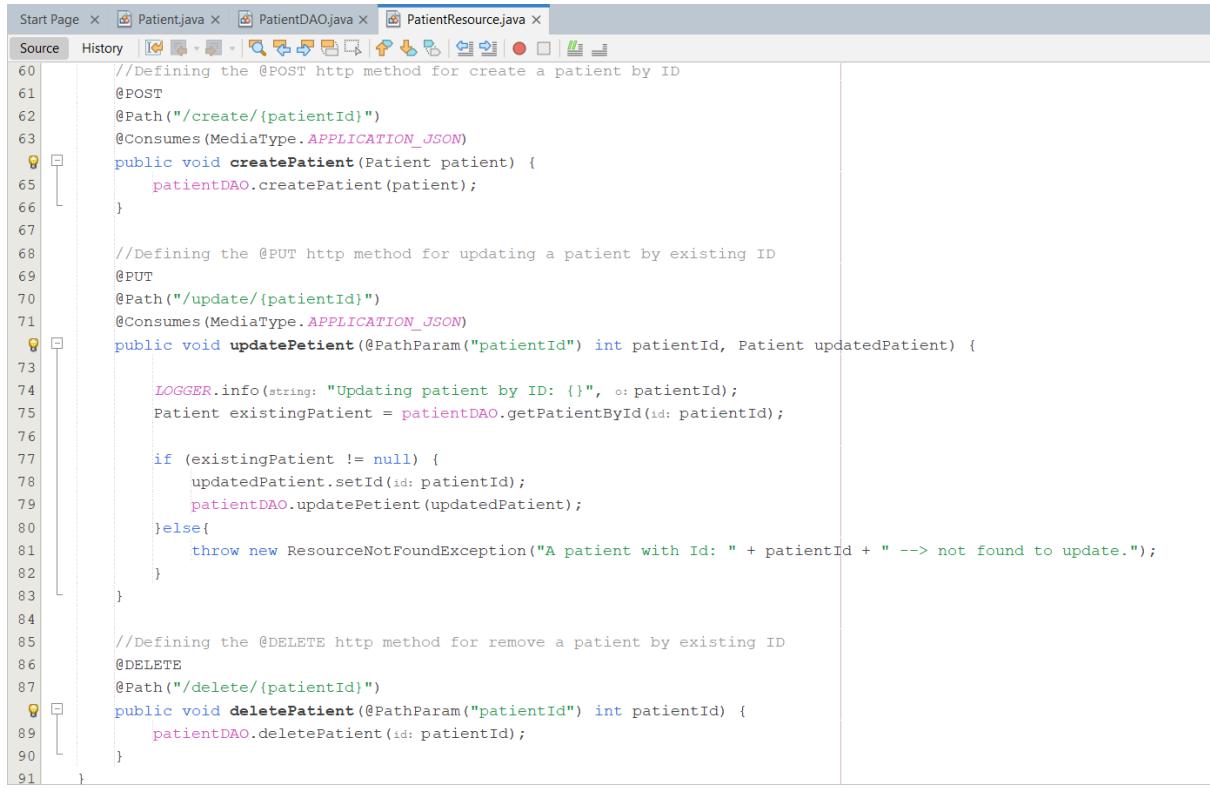


```

28  @Path("/patients")
29  public class PatientResource {
30
31      //Define the logger using logger factory
32      private static final Logger LOGGER = LoggerFactory.getLogger(PatientResource.class);
33
34      //Making the object from PatientDAO class
35      private PatientDAO patientDAO = new PatientDAO();
36
37      //Defining the @GET http method for getting all patients
38      @GET
39      @Path("/allPatients")
40      @Produces(MediaType.APPLICATION_JSON)
41      public List<Patient> getAllPatients() {
42          return patientDAO.getAllPatients();
43      }
44
45      //Defining the @GET http method for getting a patient by ID
46      @GET
47      @Path("/{patientId}")
48      @Produces(MediaType.APPLICATION_JSON)
49      public Patient getPatientById(@PathParam("patientId") int playerId) {
50
51          Patient patientObj = patientDAO.getPatientById(id: playerId);
52          if(patientObj != null){
53              LOGGER.info(string: "Getting patient by Id: {}", o: playerId);
54              return patientObj;
55          }else{
56              throw new ResourceNotFoundException("Patient with this Id: " + playerId + " --> not found.");
57          }
58      }

```

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



```

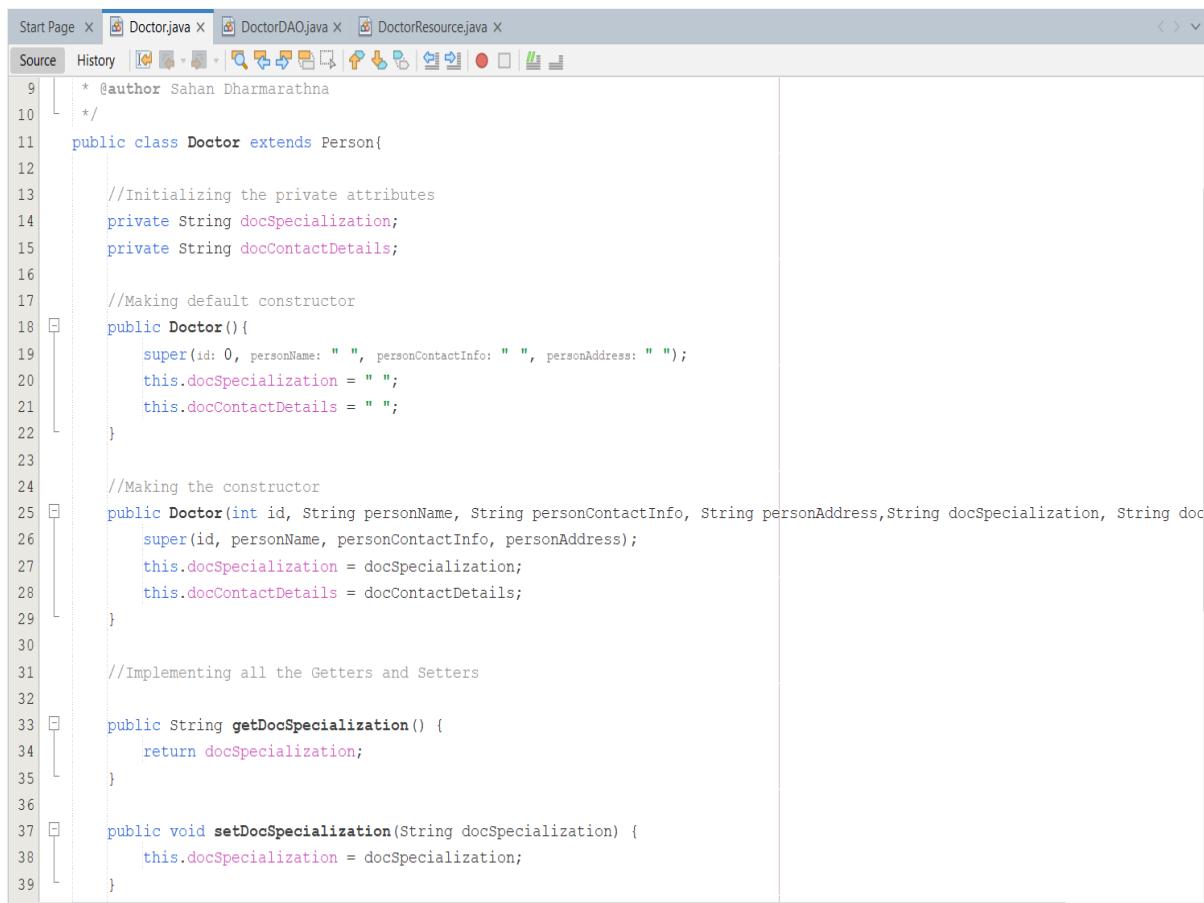
60  //Defining the @POST http method for create a patient by ID
61  @POST
62  @Path("/create/{patientId}")
63  @Consumes(MediaType.APPLICATION_JSON)
64  public void createPatient(Patient patient) {
65      patientDAO.createPatient(patient);
66  }
67
68  //Defining the @PUT http method for updating a patient by existing ID
69  @PUT
70  @Path("/update/{patientId}")
71  @Consumes(MediaType.APPLICATION_JSON)
72  public void updatePatient(@PathParam("patientId") int playerId, Patient updatedPatient) {
73
74      LOGGER.info(string: "Updating patient by ID: {}", o: playerId);
75      Patient existingPatient = patientDAO.getPatientById(id: playerId);
76
77      if (existingPatient != null) {
78          updatedPatient.setId(id: playerId);
79          patientDAO.updatePatient(updatedPatient);
80      }else{
81          throw new ResourceNotFoundException("A patient with Id: " + playerId + " --> not found to update.");
82      }
83  }
84
85  //Defining the @DELETE http method for remove a patient by existing ID
86  @DELETE
87  @Path("/delete/{patientId}")
88  public void deletePatient(@PathParam("patientId") int playerId) {
89      patientDAO.deletePatient(id: playerId);
90  }

```

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).



The screenshot shows a Java IDE interface with the Doctor.java file open. The code defines a Doctor class that extends the Person class. It includes two constructors: a default constructor and a parameterized constructor. Both constructors call the super constructor of Person and initialize private attributes docSpecialization and docContactDetails. The code also implements two getter and setter methods for docSpecialization.

```
9  * @author Sahan Dharmarathna
10 */
11 public class Doctor extends Person{
12
13     //Initializing the private attributes
14     private String docSpecialization;
15     private String docContactDetails;
16
17     //Making default constructor
18     public Doctor(){
19         super(id: 0, personName: " ", personContactInfo: " ", personAddress: " ");
20         this.docSpecialization = " ";
21         this.docContactDetails = " ";
22     }
23
24     //Making the constructor
25     public Doctor(int id, String personName, String personContactInfo, String personAddress, String docSpecialization, String docContactDetails) {
26         super(id, personName, personContactInfo, personAddress);
27         this.docSpecialization = docSpecialization;
28         this.docContactDetails = docContactDetails;
29     }
30
31     //Implementing all the Getters and Setters
32
33     public String getDocSpecialization() {
34         return docSpecialization;
35     }
36
37     public void setDocSpecialization(String docSpecialization) {
38         this.docSpecialization = docSpecialization;
39     }
}
```

Figure 10: Code snippet for Doctor model class

```

17 public class DoctorDAO extends PersonDAO{
18     //Defining logger
19     private static final Logger LOGGER = Logger.getLogger(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{
26         doctors.add(new Doctor(id: 1, personName: "Dr.Sahan", personContactInfo: "Mobile: 0716521830", personAddress: "Dope,Bentota", docSpeciality: "General Practitioner"));
27         doctors.add(new Doctor(id: 2, personName: "Dr.Upeksha", personContactInfo: "Mobile: 0714545454", personAddress: "Dope,Gallagewatta", docSpeciality: "General Practitioner"));
28         doctors.add(new Doctor(id: 3, personName: "Dr.Dulari", personContactInfo: "Mobile: 0715555555", personAddress: "Dope,Arachchimulla", docSpeciality: "General Practitioner"));
29     }
30
31     //Create Operation
32     public void createDoctor(Doctor doctor){
33         doctors.add(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             }
48         }
49     }

```

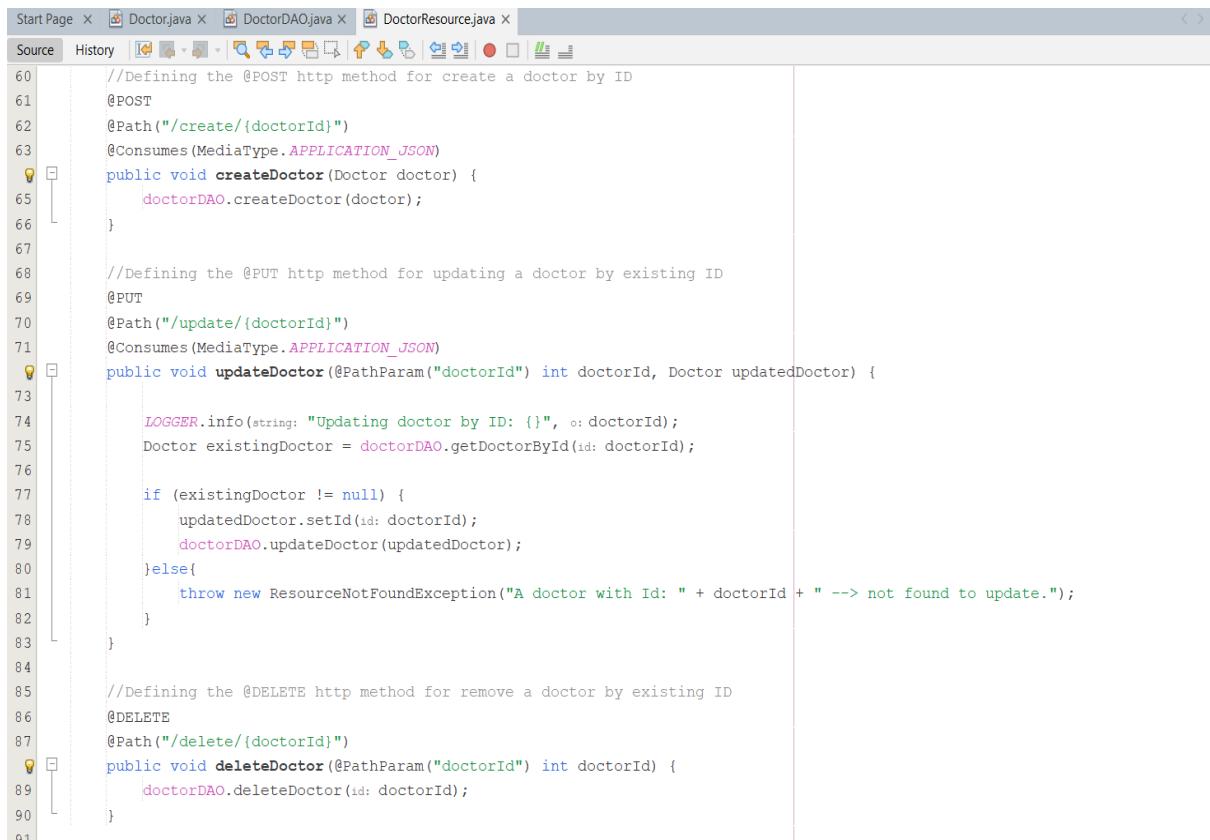
Figure 11: Code snippet for DoctorDAO class

```

28 @Path("/doctors")
29 public class DoctorResource {
30
31     //Define the logger using logger factory
32     private static final Logger LOGGER = LoggerFactory.getLogger(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



The screenshot shows a Java code editor with the DoctorResource.java file open. The code defines three RESTful methods: @POST for creating a doctor by ID, @PUT for updating a doctor by existing ID, and @DELETE for removing a doctor by existing ID. The code uses annotations from javax.ws.rs and javax.inject, and imports Doctor, DoctorDAO, and DoctorResource classes. It also includes a logger (LOGGER) and a ResourceNotFoundException.

```
60 //Defining the @POST http method for create a doctor by ID
61 @POST
62 @Path("/create/{doctorId}")
63 @Consumes(MediaType.APPLICATION_JSON)
64 public void createDoctor(Doctor doctor) {
65     doctorDAO.createDoctor(doctor);
66 }
67
68 //Defining the @PUT http method for updating a doctor by existing ID
69 @PUT
70 @Path("/update/{doctorId}")
71 @Consumes(MediaType.APPLICATION_JSON)
72 public void updateDoctor(@PathParam("doctorId") int doctorId, Doctor updatedDoctor) {
73
74     LOGGER.info(string: "Updating doctor by ID: {}", o: doctorId);
75     Doctor existingDoctor = doctorDAO.getDoctorById(id: doctorId);
76
77     if (existingDoctor != null) {
78         updatedDoctor.setId(id: doctorId);
79         doctorDAO.updateDoctor(updatedDoctor);
80     }else{
81         throw new ResourceNotFoundException("A doctor with Id: " + doctorId + " --> not found to update.");
82     }
83 }
84
85 //Defining the @DELETE http method for remove a doctor by existing ID
86 @DELETE
87 @Path("/delete/{doctorId}")
88 public void deleteDoctor(@PathParam("doctorId") int doctorId) {
89     doctorDAO.deleteDoctor(id: doctorId);
90 }
```

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 done in the AppointmentResource class(@GET, @POST, @PUT, @DELETE).

```

12 public class Appointment {
13
14     //Initializing the private attributes
15     private int appointmentId;
16     private String date;
17     private String time;
18     private Patient patient;
19     private Doctor doctor;
20
21     //Making default constructor
22     public Appointment(){}
23
24     //Making the constructor
25     public Appointment(int appointmentId, String date, String time ,Patient patient, Doctor doctor){
26
27         this.appointmentId = appointmentId;
28         this.date = date;
29         this.time = time;
30         this.patient = patient;
31         this.doctor = doctor;
32     }
33
34     //Implementing all the Getters and Setters
35
36     public int getAppointmentId() {
37         return appointmentId;
38     }
39
40     public void setAppointmentId(int appointmentId) {
41         this.appointmentId = appointmentId;
42     }

```

Figure 14: Code snippet for Appointment model class

```

19 public class AppointmentDAO {
20
21     //Defining logger
22     private static final Logger LOGGER = Logger.getLogger(AppointmentDAO.class.getName());
23
24     //Initializing the arraylist
25     private static List<Appointment> ListOfAppointments = new ArrayList<>();
26
27
28     //Defining the details static block
29     static{
30
31         Patient patient01 = new Patient(id: 1, personName: "Kamal", personContactInfo: "0716521830", personAddress: "258/A Dope,Bentota", pat
32         Patient patient02 = new Patient(id: 2, personName: "Piyal", personContactInfo: "0718989890", personAddress: "238/B Colombo,Fort", pat
33         Patient patient03 = new Patient(id: 3, personName: "Namal", personContactInfo: "0775505500", personAddress: "248/B Galle,Fort", pat
34
35         Doctor doctor01 = new Doctor(id: 1, personName: "Dr.Sahan", personContactInfo: "0716521830", personAddress: "20001/A", docSpecialization:
36         Doctor doctor02 = new Doctor(id: 2, personName: "Dr.Upeksha", personContactInfo: "0714545454", personAddress: "20002/B", docSpecializati
37         Doctor doctor03 = new Doctor(id: 3, personName: "Dr.Dulari", personContactInfo: "0715555555", personAddress: "20003/C", docSpecialization:
38
39         ListOfAppointments.add(new Appointment(appointmentId: 1, date:"05/05", time:"10.00", patient:patient01, doctor: doctor01));
40         ListOfAppointments.add(new Appointment(appointmentId: 2, date:"06/06", time:"11.00", patient:patient02, doctor: doctor02));
41         ListOfAppointments.add(new Appointment(appointmentId: 3, date:"07/07", time:"12.00", patient:patient03, doctor: doctor03));
42
43
44     //Create Operation
45     public void createAppointment(Appointment appointment){
46         ListOfAppointments.add(e: appointment);
47     }
48
49     //Read Operation for all appointments.
50     public List<Appointment> getAllAppointments () {

```

Figure 15: Code snippet for AppointmentDAO class

```

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("/{appointmentId}")
48         @Produces(MediaType.APPLICATION_JSON)
49         public Appointment getAppointmentById(@PathParam("appointmentId") int appointmentId) {
50
51             Appointment appointObj = appointDAO.getAppointmentById(appointmentId);
52             if(appointObj != null){
53                 LOGGER.info(string: "Getting appointment by Id: {}", o: appointmentId);
54                 return appointObj;
55             }else{
56                 throw new ResourceNotFoundException("Appointment with this Id: " + appointmentId + " --> not found.");
57             }
58         }

```

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

```

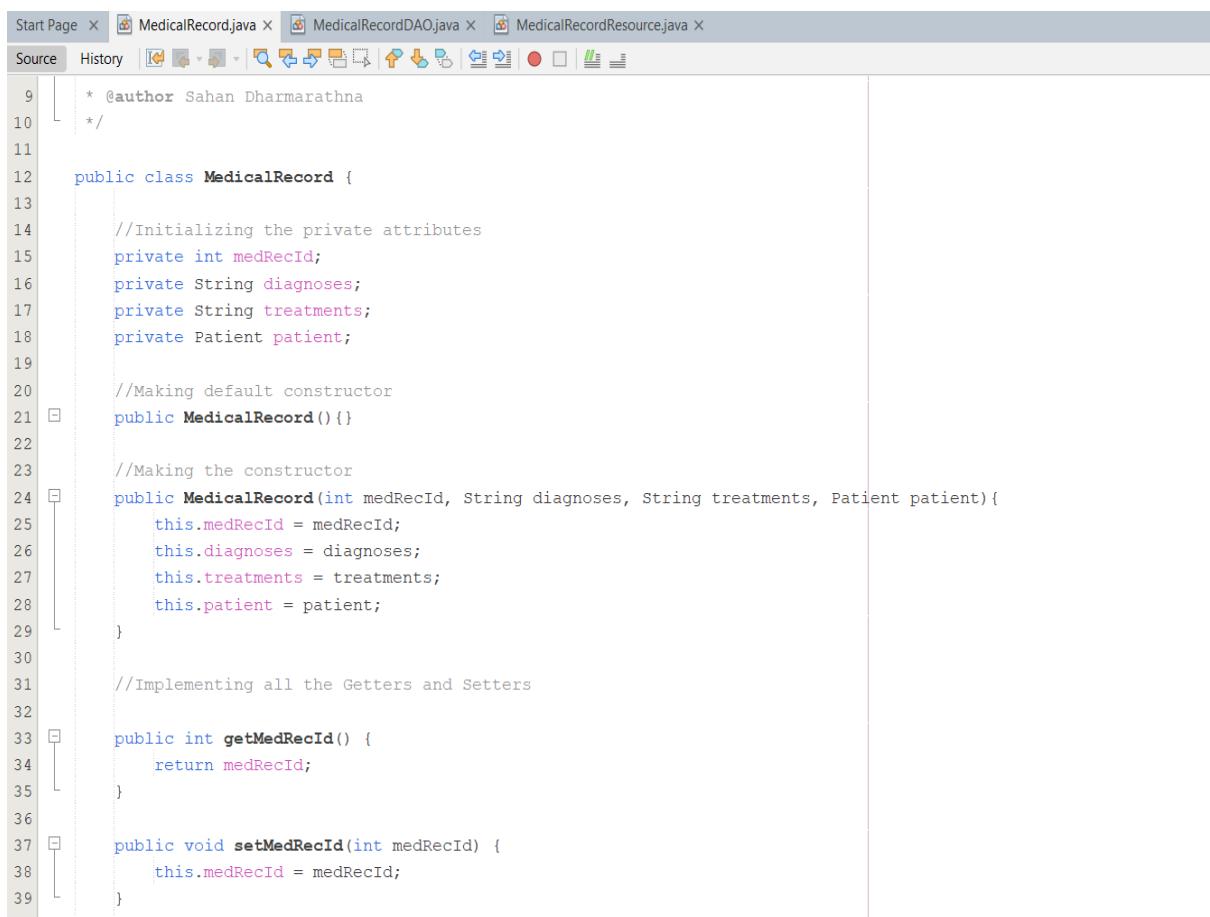
61         //Defining the @POST http method for create an appointment by ID
62         @POST
63         @Path("/create/{appointmentId}")
64         @Consumes(MediaType.APPLICATION_JSON)
65         public void createAppointment(Appointment appointment) {
66             appointDAO.createAppointment(appointment);
67         }
68
69         //Defining the @PUT http method for updating an appointment by existing ID
70         @PUT
71         @Path("/update/{appointmentId}")
72         @Consumes(MediaType.APPLICATION_JSON)
73         public void updateAppointment(@PathParam("appointmentId") int appointmentId, Appointment updatedAppointment) {
74             LOGGER.info(string: "Updating appointment by ID: {}", o: appointmentId);
75             Appointment existingAppoint = appointDAO.getAppointmentById(appointmentId);
76
77             if (existingAppoint != null) {
78                 updatedAppointment.setAppointmentId(appointmentId: appointmentId);
79                 appointDAO.updateAppointment(updatedAppointment);
80             }else{
81                 throw new ResourceNotFoundException("An appointment with Id: " + appointmentId + " --> 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/{appointmentId}")
88         public void deleteAppointment(@PathParam("appointmentId") int appointmentId) {
89             appointDAO.deleteAppointment(appointmentId);
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.



The screenshot shows a Java code editor with the following details:

- Project Structure:** The tabs at the top show "Start Page X", "MedicalRecord.java X", "MedicalRecordDAO.java X", and "MedicalRecordResource.java X".
- Toolbar:** Below the tabs is a toolbar with various icons for file operations like new, open, save, cut, copy, paste, etc.
- Code Editor:** The main area contains the Java code for the `MedicalRecord` class. The code includes:
  - A Javadoc comment block starting with `/* @author Sahan Dharmarathna */`.
  - A constructor `public MedicalRecord()`.
  - A parameterized constructor `public MedicalRecord(int medRecId, String diagnoses, String treatments, Patient patient)` with four parameters: `medRecId`, `diagnoses`, `treatments`, and `patient`. It initializes each attribute with its corresponding parameter.
  - Implementation of the `getMedRecId()` and `setMedRecId(int medRecId)` methods.

Figure 18: Code snippet for MedicalRecord model class

```

16  * @author Sahan Dharmarathna
17  */
18  public class MedicalRecordDAO {
19      //Defining logger
20      private static final Logger LOGGER = Logger.getLogger(MedicalRecordDAO.class.getName());
21
22      //Initializing the arraylist
23      private static List<MedicalRecord> medicalRecords = new ArrayList<>();
24
25      //Defining the details static block
26      static{
27          Patient patient01 = new Patient(id: 1, personName: "Kamal", personContactInfo: "0716521830", personAddress: "258/A Dope,Bentota", patientId: 1);
28          Patient patient02 = new Patient(id: 2, personName: "Piyal", personContactInfo: "0718989890", personAddress: "238/B Colombo,Fort", patientId: 2);
29          Patient patient03 = new Patient(id: 3, personName: "Namal", personContactInfo: "0775505500", personAddress: "248/B Galle,Fort", patientId: 3);
30
31          medicalRecords.add(new MedicalRecord(medRecId: 1, diagnoses: "Having Sugar", treatments: "Getting sugar pills", patient:patient01));
32          medicalRecords.add(new MedicalRecord(medRecId: 2, diagnoses: "Having Pressure", treatments: "Getting pressure pills", patient:patient02));
33          medicalRecords.add(new MedicalRecord(medRecId: 3, diagnoses: "Having Fever", treatments: "Getting fever pills", patient:patient03));
34      }
35
36
37      //Create Operation
38      public void createMedicalRecord(MedicalRecord medRec){
39          medicalRecords.add(medRec);
40      }
41
42
43      //Read Operation for all medical records.
44      public List<MedicalRecord> getAllMedicalRecords(){
45          return medicalRecords;
46      }

```

Figure 19: Code snippet for MedicalRecordDAO class

```

29  @Path("/medRecords")
30  public class MedicalRecordResource {
31
32      //Define the logger using logger factory
33      private static final Logger LOGGER = LoggerFactory.getLogger(MedicalRecordResource.class);
34
35      //Making the object from MedicalRecordDAO class
36      private MedicalRecordDAO medRecordDAO = new MedicalRecordDAO();
37
38      //Defining the @GET http method for getting all medical records
39      @GET
40      @Path("/allMedRecords")
41      @Produces(MediaType.APPLICATION_JSON)
42      public List<MedicalRecord> getAllMedicalRecords() {
43          return medRecordDAO.getAllMedicalRecords();
44      }
45
46      //Defining the @GET http method for getting a medical record by ID
47      @GET
48      @Path("/{medRecId}")
49      @Produces(MediaType.APPLICATION_JSON)
50      public MedicalRecord getMedicalRecordById(@PathParam("medRecId") int medRecId) {
51
52          MedicalRecord medRecObj = medRecordDAO.getMedicalRecordById(medRecId);
53          if(medRecObj != null){
54              LOGGER.info(string: "Getting medical record by Id: {}", o: medRecId);
55              return medRecObj;
56          }else{
57              throw new ResourceNotFoundException("Medical record with this Id: " + medRecId + " --> not found.");
58          }
59      }

```

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

```

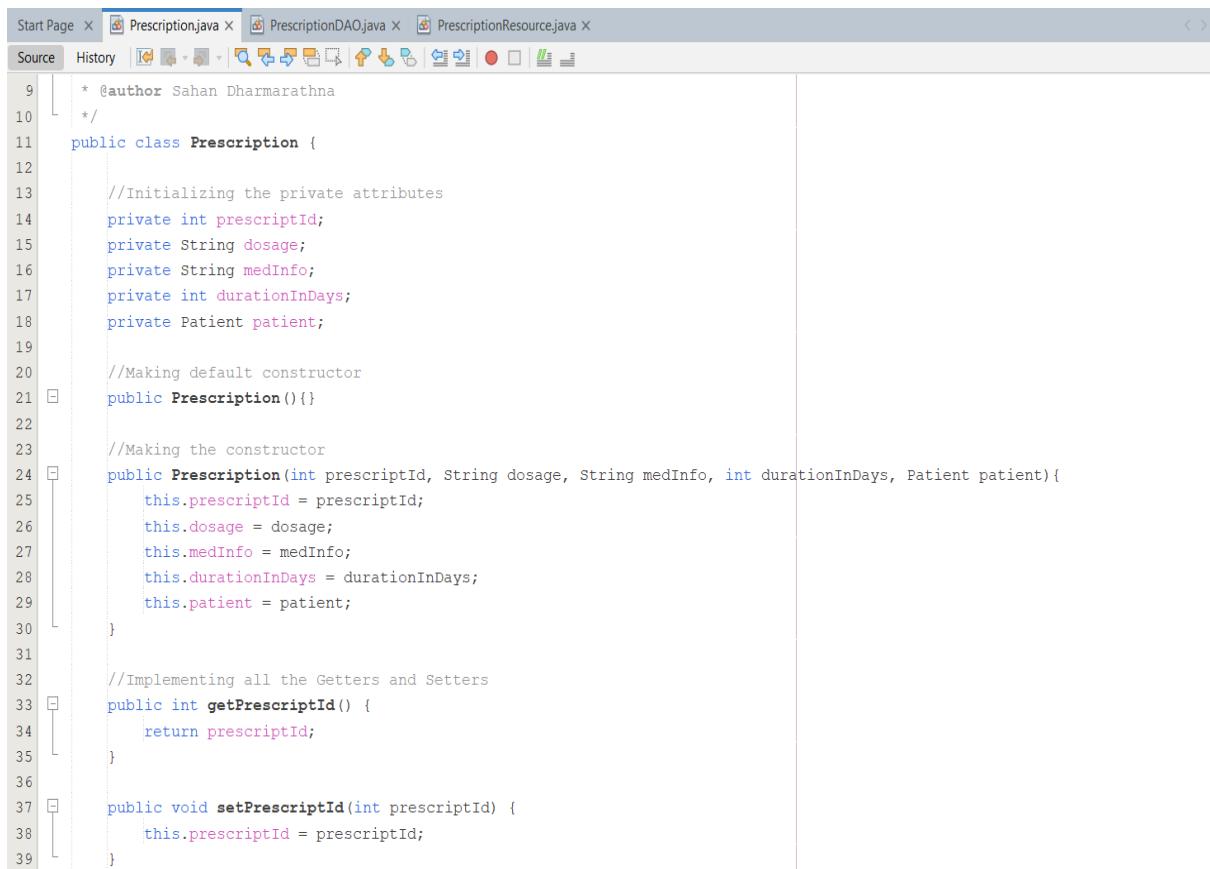
61 //Defining the @POST http method for create a medical record by ID
62 @POST
63 @Path("/create/{medRecId}")
64 @Consumes(MediaType.APPLICATION_JSON)
65 public void createMedicalRecord(MedicalRecord newMedRec) {
66     medRecordDAO.createMedicalRecord(newMedRec);
67 }
68
69 //Defining the @PUT http method for updating a medical record by existing ID
70 @PUT
71 @Path("/update/{medRecId}")
72 @Consumes(MediaType.APPLICATION_JSON)
73 public void updateMedicalRecord(@PathParam("medRecId") int medRecId, MedicalRecord updatedMedRec) {
74     LOGGER.info(string: "Updating medical record by ID: {}", o: medRecId);
75     MedicalRecord existingMedRec = medRecordDAO.getMedicalRecordById(medRecordId:medRecId);
76
77     if (existingMedRec != null) {
78         updatedMedRec.setMedRecId(medRecId);
79         medRecordDAO.updateMedicalRecord(updatedMedRec);
80     } else{
81         throw new ResourceNotFoundException("A medical record with Id: " + medRecId + " --> not found to update.");
82     }
83 }
84
85 //Defining the @DELETE http method for remove a medical record by existing ID
86 @DELETE
87 @Path("/delete/{medRecId}")
88 public void deleteMedicalRecord(@PathParam("medRecId") int medRecId) {
89     medRecordDAO.deleteMedicalRecord(medRecordId:medRecId);
90 }

```

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.

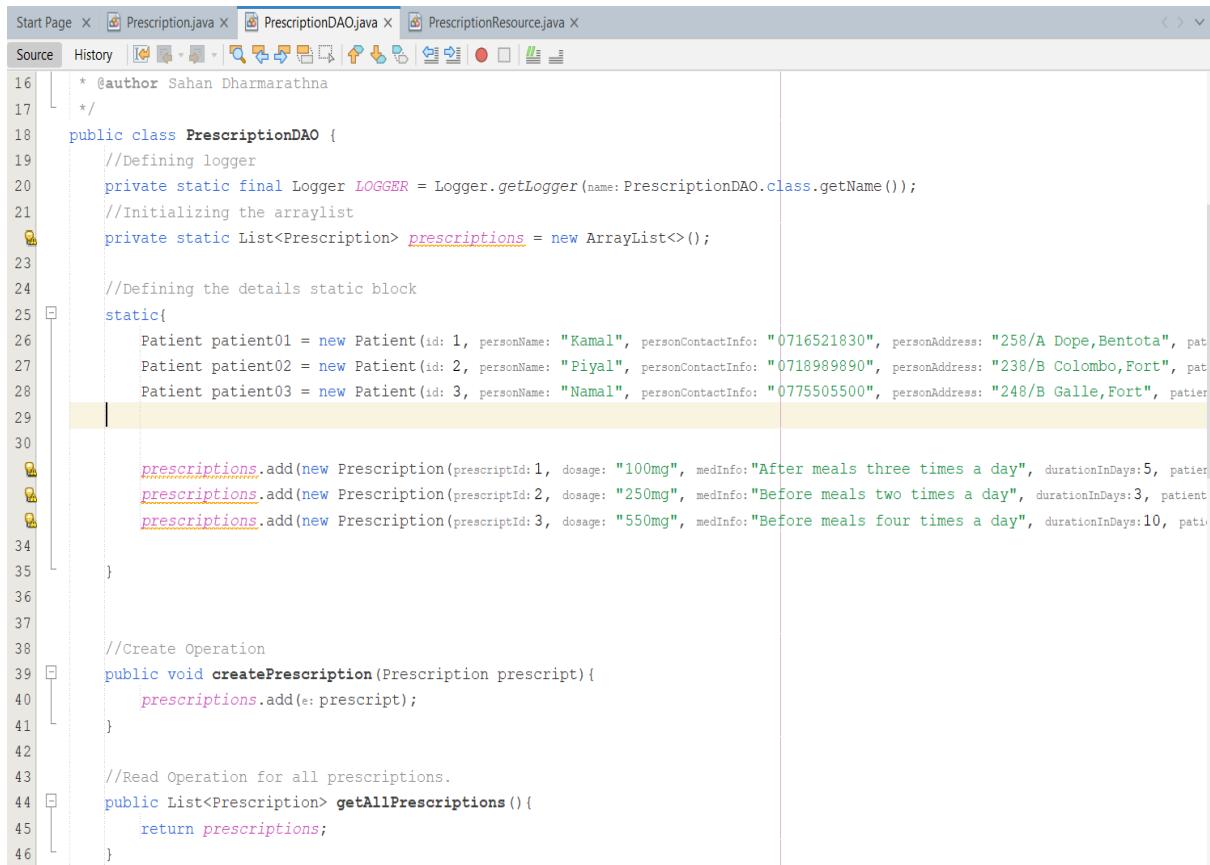


```

9  * @author Sahan Dharmarathna
10 */
11 public class Prescription {
12
13     //Initializing the private attributes
14     private int prescriptId;
15     private String dosage;
16     private String medInfo;
17     private int durationInDays;
18     private Patient patient;
19
20     //Making default constructor
21     public Prescription(){}
22
23     //Making the constructor
24     public Prescription(int prescriptId, String dosage, String medInfo, int durationInDays, Patient patient){
25         this.prescriptId = prescriptId;
26         this.dosage = dosage;
27         this.medInfo = medInfo;
28         this.durationInDays = durationInDays;
29         this.patient = patient;
30     }
31
32     //Implementing all the Getters and Setters
33     public int getPrescriptId() {
34         return prescriptId;
35     }
36
37     public void setPrescriptId(int prescriptId) {
38         this.prescriptId = prescriptId;
39     }

```

Figure 22: Code snippet for Prescription model class



```

16  * @author Sahan Dharmarathna
17 */
18 public class PrescriptionDAO {
19
20     //Defining logger
21     private static final Logger LOGGER = Logger.getLogger(name:PrescriptionDAO.class.getName());
22
23     //Initializing the arraylist
24     private static List<Prescription> prescriptions = new ArrayList<>();
25
26     //Defining the details static block
27     static{
28         Patient patient01 = new Patient(id: 1, personName: "Kamal", personContactInfo: "0716521830", personAddress: "258/A Dope,Bentota", patientId: 1);
29         Patient patient02 = new Patient(id: 2, personName: "Piyal", personContactInfo: "0718989890", personAddress: "238/B Colombo,Fort", patientId: 2);
30         Patient patient03 = new Patient(id: 3, personName: "Namal", personContactInfo: "0775505500", personAddress: "248/B Galle,Fort", patientId: 3);
31
32         prescriptions.add(new Prescription(prescriptId:1, dosage: "100mg", medInfo:"After meals three times a day", durationInDays:5, patientId: 1));
33         prescriptions.add(new Prescription(prescriptId:2, dosage: "250mg", medInfo:"Before meals two times a day", durationInDays:3, patientId: 2));
34         prescriptions.add(new Prescription(prescriptId:3, dosage: "550mg", medInfo:"Before meals four times a day", durationInDays:10, patientId: 3));
35     }
36
37
38     //Create Operation
39     public void createPrescription(Prescription prescript) {
40         prescriptions.add(e: prescript);
41     }
42
43     //Read Operation for all prescriptions.
44     public List<Prescription> getAllPrescriptions () {
45         return prescriptions;
46     }

```

Figure 23: Code snippet for PrescriptionDAO class

```

28  @Path("/prescriptions")
29  public class PrescriptionResource {
30
31      //Define the logger using logger factory
32      private static final Logger LOGGER = LoggerFactory.getLogger(PrescriptionResource.class);
33
34      //Making the object from PrescriptionDAO class
35      private PrescriptionDAO prescriptDAO = new PrescriptionDAO();
36
37      //Defining the @GET http method for getting all prescriptions
38      @GET
39      @Path("/allPrescripts")
40      @Produces(MediaType.APPLICATION_JSON)
41      public List<Prescription> getAllPrescriptions() {
42          return prescriptDAO.getAllPrescriptions();
43      }
44
45      //Defining the @GET http method for getting a prescription by ID
46      @GET
47      @Path("/{prescriptId}")
48      @Produces(MediaType.APPLICATION_JSON)
49      public Prescription getPrescriptionById(@PathParam("prescriptId") int prescriptId) {
50
51          Prescription presObj = prescriptDAO.getPrescriptionById(prescriptId);
52          if(presObj != null){
53              LOGGER.info(string: "Getting prescription by Id: {}", o: prescriptId);
54              return presObj;
55          }else{
56              throw new ResourceNotFoundException("Prescription with this Id: " + prescriptId + " --> not found.");
57          }
58      }

```

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

```

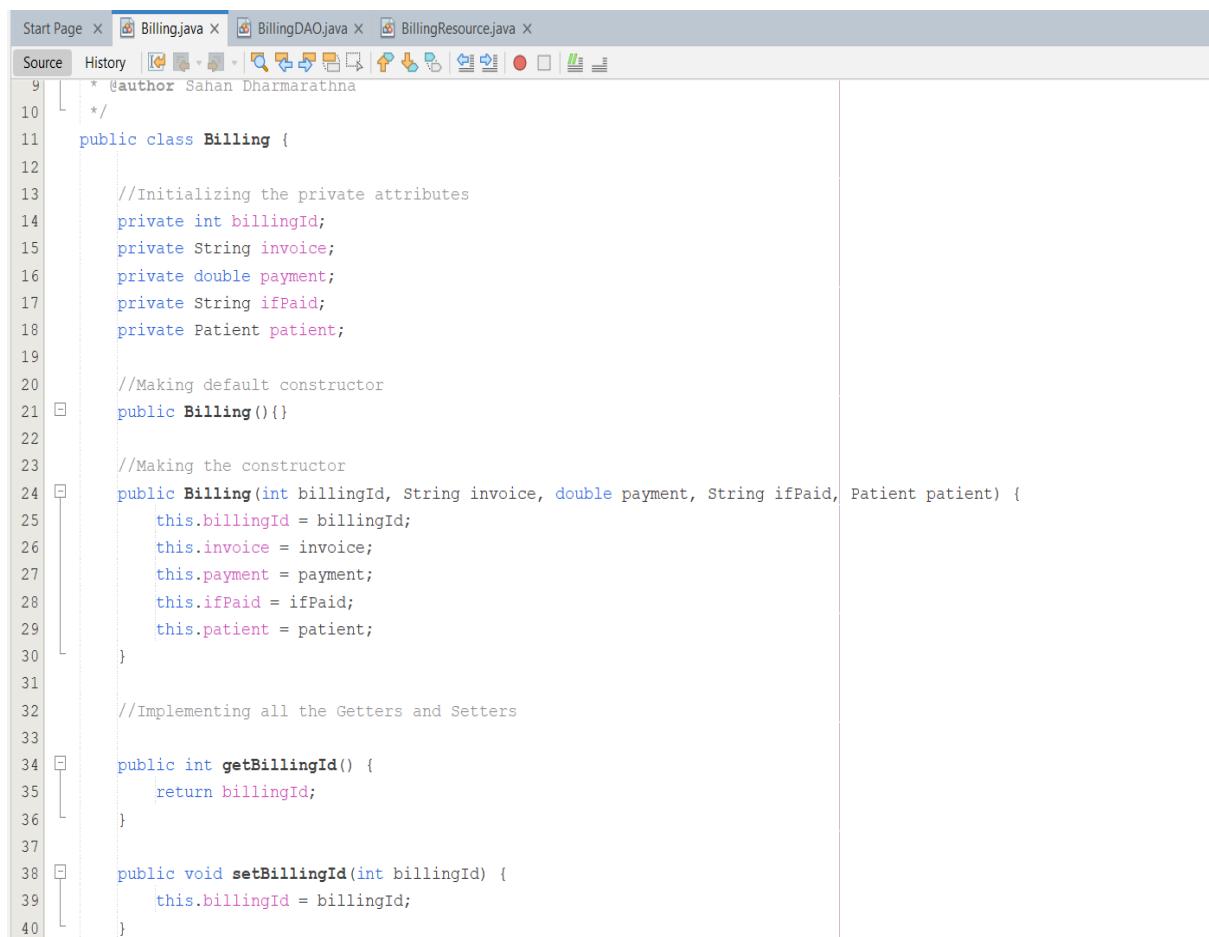
60      //Defining the @POST http method for create a prescription by ID
61      @POST
62      @Path("/create/{prescriptId}")
63      @Consumes(MediaType.APPLICATION_JSON)
64      public void createPrescription(Prescription prescription) {
65          prescriptDAO.createPrescription(prescript: prescription);
66      }
67
68      //Defining the @PUT http method for updating a prescription by existing ID
69      @PUT
70      @Path("/update/{prescriptId}")
71      @Consumes(MediaType.APPLICATION_JSON)
72      public void updatePrescription(@PathParam("prescriptId") int prescriptId, Prescription updatedPrescript) {
73          LOGGER.info(string: "Updating prescription by ID: {}", o: prescriptId);
74          Prescription existingPrescript = prescriptDAO.getPrescriptionById(prescriptId);
75
76          if (existingPrescript != null) {
77              updatedPrescript.setPrescriptId(prescriptId);
78              prescriptDAO.updatePrescription(updatedPrescript);
79          }else{
80              throw new ResourceNotFoundException("A prescription with Id: " + prescriptId + " --> not found to update.");
81          }
82      }
83
84      //Defining the @DELETE http method for remove a prescription by existing ID
85      @DELETE
86      @Path("/delete/{prescriptId}")
87      public void deletePrescription(@PathParam("prescriptId") int prescriptId) {
88          prescriptDAO.deletePrescription(prescriptId);
89      }

```

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).



The screenshot shows a Java code editor with the file 'Billing.java' open. The code defines a class 'Billing' with private attributes for billingId, invoice, payment, ifPaid, and patient. It includes a default constructor and a parameterized constructor. Getters and setters for the billingId attribute are implemented. The code is annotated with a copyright notice and author information.

```
9  * Copyright © 2023 Sahan Dharmarathna
10 */
11 public class Billing {
12
13     //Initializing the private attributes
14     private int billingId;
15     private String invoice;
16     private double payment;
17     private String ifPaid;
18     private Patient patient;
19
20     //Making default constructor
21     public Billing(){}
22
23     //Making the constructor
24     public Billing(int billingId, String invoice, double payment, String ifPaid, Patient patient) {
25         this.billingId = billingId;
26         this.invoice = invoice;
27         this.payment = payment;
28         this.ifPaid = ifPaid;
29         this.patient = patient;
30     }
31
32     //Implementing all the Getters and Setters
33
34     public int getBillingId() {
35         return billingId;
36     }
37
38     public void setBillingId(int billingId) {
39         this.billingId = billingId;
40     }
}
```

Figure 26: Code snippet for Billing model class

```

16     * @author Sahan Dharmarathna
17     */
18    public class BillingDAO {
19        //Defining logger
20        private static final Logger LOGGER = Logger.getLogger(name: BillingDAO.class.getName());
21        //Initializing the arraylist
22        private static List<Billing> bills = new ArrayList<>();
23
24        //Defining the details static block
25        static{
26
27            Patient patient01 = new Patient(id: 1, personName: "Kamal", personContactInfo: "0716521830", personAddress: "258/A Dope,Bentota", patientId: 1);
28            Patient patient02 = new Patient(id: 2, personName: "Piyal", personContactInfo: "0718989890", personAddress: "238/B Colombo,Fort", patientId: 2);
29            Patient patient03 = new Patient(id: 3, personName: "Namal", personContactInfo: "0775505500", personAddress: "248/B Galle,Fort", patientId: 3);
30
31            bills.add(new Billing(billingId: 1, invoice:"Invoice 01", payment:5000, ifPaid: "Paid", patient:patient01));
32            bills.add(new Billing(billingId: 2, invoice:"Invoice 02", payment:2500, ifPaid: "Not Paid", patient:patient02));
33            bills.add(new Billing(billingId: 3, invoice:"Invoice 03", payment:7500, ifPaid: "Not Paid", patient:patient03));
34        }
35
36        //Create Operation
37        public void createBill(Billing bill){
38            bills.add(e:bill);
39        }
40
41
42        //Read Operation for all prescriptions.
43        public List<Billing> getAllBills(){
44            return bills;
45        }
46

```

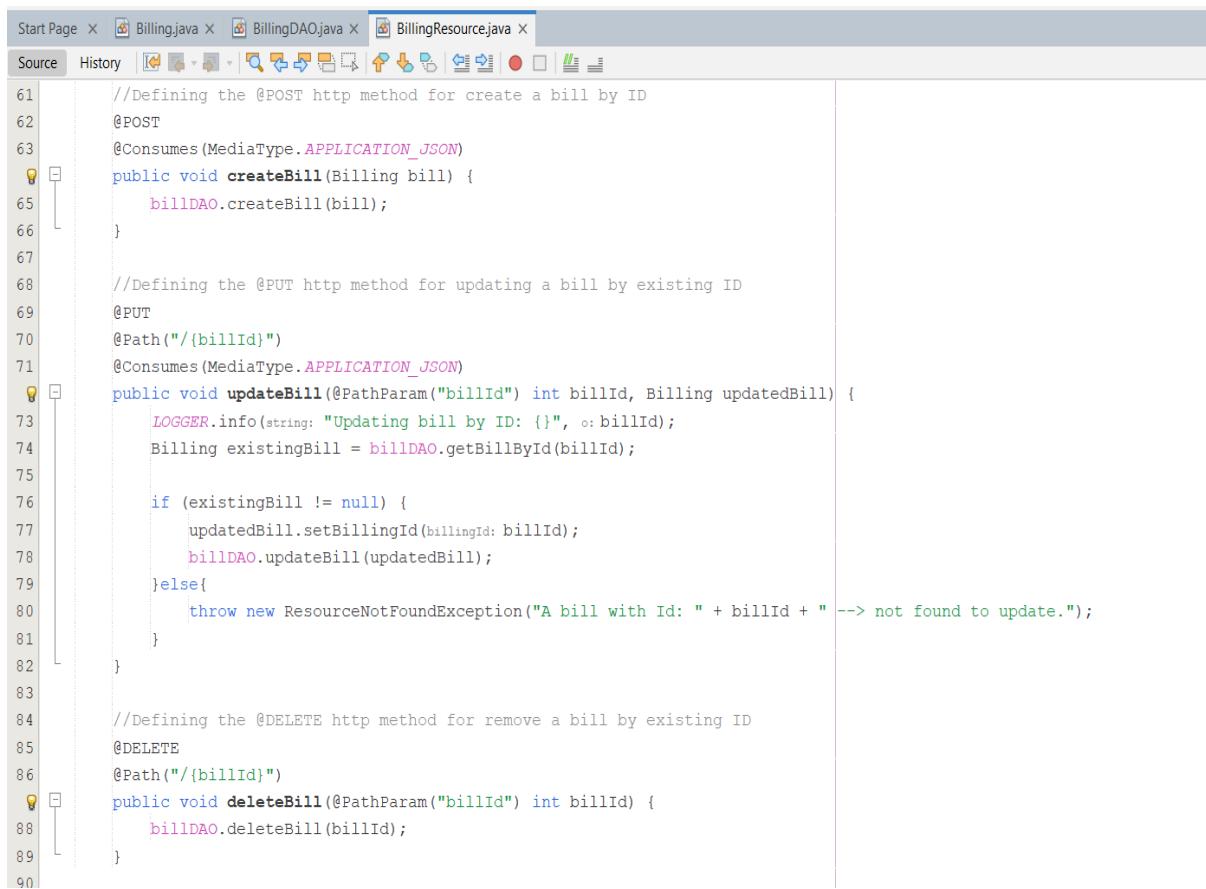
Figure 27: Code snippet for BillingDAO class

```

29     @Path("/billResource")
30     public class BillingResource {
31
32         //Define the logger using logger factory
33         private static final Logger LOGGER = LoggerFactory.getLogger(clazz: BillingResource.class);
34
35         //Making the object from BillingDAO class
36         private BillingDAO billDAO = new BillingDAO();
37
38         //Defining the @GET http method for getting all bills
39         @GET
40         @Path("/allBills")
41         @Produces(MediaType.APPLICATION_JSON)
42         public List<Billing> getAllBills() {
43             return billDAO.getAllBills();
44         }
45
46         //Defining the @GET http method for getting a bill by ID
47         @GET
48         @Path("/{billId}")
49         @Produces(MediaType.APPLICATION_JSON)
50         public Billing getBillById(@PathParam("billId") int billId) {
51
52             Billing billObj = billDAO.getBillById(billId);
53             if(billObj != null){
54                 LOGGER.info(string: "Getting bill by Id: {}", o:billId);
55                 return billObj;
56             }else{
57                 throw new ResourceNotFoundException("Bill with this Id: " + billId + " --> not found.");
58             }
59         }

```

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



The screenshot shows a Java code editor with the BillingResource.java file open. The code defines three methods: createBill, updateBill, and deleteBill. The createBill method is a POST request to create a bill by ID. The updateBill method is a PUT request to update an existing bill by ID, using a logger to info the update and handling a ResourceNotFoundException if the bill is not found. The deleteBill method is a DELETE request to remove a bill by ID.

```
61 //Defining the @POST http method for create a bill by ID
62 @POST
63 @Consumes(MediaType.APPLICATION_JSON)
64 public void createBill(Billing bill) {
65     billDAO.createBill(bill);
66 }
67
68 //Defining the @PUT http method for updating a bill by existing ID
69 @PUT
70 @Path("/{billId}")
71 @Consumes(MediaType.APPLICATION_JSON)
72 public void updateBill(@PathParam("billId") int billId, Billing updatedBill) {
73     LOGGER.info(string: "Updating bill by ID: {}", o:billId);
74     Billing existingBill = billDAO.getBillById(billId);
75
76     if (existingBill != null) {
77         updatedBill.setBillingId(billingId: billId);
78         billDAO.updateBill(updatedBill);
79     }else{
80         throw new ResourceNotFoundException("A bill with Id: " + billId + " --> not found to update.");
81     }
82 }
83
84 //Defining the @DELETE http method for remove a bill by existing ID
85 @DELETE
86 @Path("/{billId}")
87 public void deleteBill(@PathParam("billId") int billId) {
88     billDAO.deleteBill(billId);
89 }
```

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

The screenshot shows the Postman application interface. The left sidebar displays 'My Workspace' with a collection named 'My first collection'. The main area shows a GET request for 'http://localhost:8080/2022165\_W1985549\_CSA\_CW/rest/persons/allPersons'. The response status is 200 OK, and the response body is a JSON array of three person objects:

```
[{"id": 1, "personName": "isira", "personContactInfo": "Mobile: 0710699960", "personAddress": "Athurugiriya,Colombo"}, {"id": 2, "personName": "pahan", "personContactInfo": "Mobile: 0714545454", "personAddress": "Alawwa,Kagalla"}, {"id": 3, "personName": "nuwandi", "personContactInfo": "Mobile: 0713333333", "personAddress": "Anuradhapura,New Town"}]
```

Figure 30: Postman Get method with all persons

The screenshot shows the Postman application interface. The left sidebar displays 'My Workspace' with a collection named 'My first collection'. The main area shows a GET request for 'http://localhost:8080/2022165\_W1985549\_CSA\_CW/rest/persons/2'. The response status is 200 OK, and the response body is a single person object:

```
{"id": 2, "personName": "pahan", "personContactInfo": "Mobile: 0714545454", "personAddress": "Alawwa,Kagalla"}
```

Figure 31: Postman Get method with persons id

- POST

The screenshot shows the Postman application interface. In the top navigation bar, the URL is set to `POST http://localhost:8080/2022165_W1985549_CSA_CW/rest/persons/create/5`. The 'Body' tab is selected, showing a JSON payload:

```

1 {
2   ...
3   "id": 5,
4   ...
5   "personName": "nuwan",
6   ...
7   "personContactInfo": "Mobile: 077111111",
8   ...
9   "personAddress": "Alawwa,polonnaruwa"
10 }

```

The response status is 204 No Content, Time: 69 ms, Size: 112 B.

Figure 32: Postman POST method adding new person

The screenshot shows the Postman application interface. In the top navigation bar, the URL is set to `GET http://localhost:8080/2022165_W1985549_CSA_CW/rest/persons/5`. The 'Body' tab is selected, showing the JSON response from the previous POST request:

```

1 {
2   ...
3   "id": 5,
4   ...
5   "personName": "nuwan",
6   ...
7   "personContactInfo": "Mobile: 077111111",
8   ...
9   "personAddress": "Alawwa,polonnaruwa"
10 }

```

The response status is 200 OK, Time: 4 ms, Size: 263 B.

Figure 33: Postman POST method adding new person result

## ● PUT

The screenshot shows the Postman application interface. In the top navigation bar, the URL is set to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/persons/update/5`. The request method is selected as `PUT`. The `Body` tab is active, displaying the following JSON payload:

```

1 {
2   ... "id": 5,
3   ... "personName": "jayani",
4   ... "personContactInfo": "Mobile: 07000000",
5   ... "personAddress": "Bentota,Galle"
6 }

```

Below the body, the response status is shown as `204 No Content` with a time of `6 ms` and a size of `112 B`.

Figure 34: Postman PUT method updating existing person

This screenshot shows the same Postman interface after the PUT request has been sent. The URL remains the same: `http://localhost:8080/2022165_W1985549_CSA_CW/rest/persons/5`. The request method is now `GET`. The `Body` tab shows the updated person object returned by the server:

```

1 {
2   ... "id": 5,
3   ... "personName": "jayani",
4   ... "personContactInfo": "Mobile: 07000000",
5   ... "personAddress": "Bentota,Galle"
6 }

```

The response status is `200 OK` with a time of `4 ms` and a size of `256 B`.

Figure 35: Postman PUT method updating existing person result

The screenshot shows the Postman interface with a PUT request to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/persons/update/10`. The Body tab contains the following JSON payload:

```

1 {
2   ...
3   "id": 10,
4   ...
5   "personName": "nalan",
6   ...
7   "personContactInfo": "Mobile: 0722200",
8   ...
9   "personAddress": "Jpura, Colombo"
10 }

```

The response status is 404 Not Found, with the message: "A person with Id: 10 --> not found to update."

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

- **DELETE**

The screenshot shows the Postman interface with a DELETE request to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/persons/delete/5`. The Body tab contains the following JSON payload:

```

1 {
2   ...
3   "id": 5,
4   ...
5   "personName": "jayani",
6   ...
7   "personContactInfo": "Mobile: 07000000",
8   ...
9   "personAddress": "Bentota, Galle"
10 }

```

The response status is 204 No Content.

Figure 37: Postman DELETE method

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar displays 'My first collection' with two items: 'First folder inside collection' and 'Second folder inside collection'. Below this, there is a note: 'Create a collection for your requests' and a 'Create Collection' button. The main workspace shows a 'GET' request to 'http://localhost:8080/2022165\_W1985549\_CSA\_CW/rest/persons/5'. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   ... "id": 5,  
3   ... "personName": "jayani",  
4   ... "personContactInfo": "Mobile: 07000000",  
5   ... "personAddress": "Bentota,Galle"  
6 }
```

The 'Test Results' tab shows the response: 'Person with this id: 5 --> not found.' The status bar at the bottom indicates 'Status: 404 Not Found'.

Figure 38: Postman DELETE method results

# Patient Entity

- GET

The screenshot shows the Postman application interface. In the top navigation bar, the URL is set to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/patients/allPatients`. The request method is set to `GET`. The `Body` tab is selected, showing the raw JSON response. The response body is a list of three patient objects:

```
[{"id": 1, "personName": "Kamal", "personContactInfo": "0716521838", "personAddress": "258/A Dope,Bentota", "patientMedHistory": "Having Sugei", "patientHealthStatus": "Normal"}, {"id": 2, "personName": "Piyal", "personContactInfo": "0718969898", "personAddress": "238/A Colombo,Fort", "patientMedHistory": "Having Pressure", "patientHealthStatus": "Good"}, {"id": 3, "personName": "Namal", "personContactInfo": "0775565500", "personAddress": "248/A Galle,Fort", "patientMedHistory": "Having Fever"}]
```

Figure 39: Postman GET method with all patients

The screenshot shows the Postman application interface. In the top navigation bar, the URL is set to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/patients/2`. The request method is set to `GET`. The `Body` tab is selected, showing the raw JSON response. The response body is a single patient object:

```
{"id": 2, "personName": "Piyal", "personContactInfo": "0718969898", "personAddress": "238/A Colombo,Fort", "patientMedHistory": "Having Pressure", "patientHealthStatus": "Good"}
```

Figure 40: Postman GET method with patient with id

- POST

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar is visible with a collection named 'Your collector'. The main area displays a POST request to 'http://localhost:8080/2022165\_W1985549\_CSA\_CW/rest/patients/create/8'. The 'Body' tab is selected, showing a raw JSON payload:

```

1 {
2   "id": 8,
3   "personName": "nimal",
4   "personContactInfo": "970252525",
5   "personAddress": "238/A Galle,Fort",
6   "patientMedHistory": "Having Cough",
7   "patientHealthStatus": "Good"
8 }

```

The 'Test Results' section shows a response status of 204 No Content with a time of 7 ms and a size of 112 B. The bottom status bar indicates the current date and time as 07/05/2024 at 8:24 AM.

Figure 41:Postman POST method adding patient

This screenshot is identical to Figure 41, showing the same POST request to add a patient. The JSON payload is the same:

```

1 {
2   "id": 8,
3   "personName": "nimal",
4   "personContactInfo": "970252525",
5   "personAddress": "238/A Galle,Fort",
6   "patientMedHistory": "Having Cough",
7   "patientHealthStatus": "Good"
8 }

```

The response status is 204 No Content with a time of 4 ms and a size of 317 B. The bottom status bar indicates the current date and time as 07/05/2024 at 8:24 AM.

Figure 42:Postman POST method adding patient result

## ● PUT

The screenshot shows the Postman application interface. In the top navigation bar, the URL is set to `http://localhost:8080/2022165_W1985549_CSA.CW/rest/patients/update/8`. The request method is selected as `PUT`. The `Body` tab is active, showing a JSON payload:

```

1 {
2   ... "id": 8,
3   ... "personName": "pawan1",
4   ... "personContactInfo": "971589623",
5   ... "personAddress": "238/A Nuwara,Nuwara eliya",
6   ... "patientMediHistory": "Having Cough",
7   ... "patientHealthStatus": "Normal"
8 }

```

Below the body, the response status is shown as `204 No Content`. The bottom status bar indicates the operation was completed at 8:26 AM on 07/05/2024.

Figure 43:Postman PUT method updating existing patient

This screenshot shows Postman with a more complex collection structure. The left sidebar displays a collection named "My first collection" containing two nested folders: "First folder inside collection" and "Second folder inside collection", each with its own requests. The main request details are identical to Figure 43, targeting the same endpoint and using the same JSON payload. The response status is `200 OK`. The bottom status bar shows the operation was completed at 8:26 AM on 07/05/2024.

Figure 44:Postman PUT method updating existing patient result

The screenshot shows the Postman application interface. In the top navigation bar, 'Home' and 'Workspaces' are selected. The main workspace is titled 'My Workspace'. A collection named 'My first collection' is expanded, showing two folders: 'First folder inside collection' and 'Second folder inside collection', each containing a single item labeled 'item'. Below the collection tree, there is a note: 'Create a collection for your requests' and a 'Create Collection' button.

The central area displays a PUT request to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/patients/update/10`. The 'Body' tab is selected, showing a JSON payload:

```

1 {
2     "id": 10,
3     "personName": "lasith",
4     "personContactInfo": "0900000000",
5     "personAddress": "238/A Panadura",
6     "patientMedHistory": "Having Fever",
7     "patientHealthStatus": "Normal"
8 }

```

The 'Test Results' tab shows the response: Status: 404 Not Found, Time: 221 ms, Size: 203 B. The message in the results pane is: 'A patient with Id: 10 -> not found to update.'

The bottom status bar shows the date and time: 8:28 AM 07/05/2024.

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

- **DELETE**

The screenshot shows the Postman application interface. The top navigation bar has 'Home' and 'Workspaces' selected. The workspace is 'My Workspace'.

The central area displays a DELETE request to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/patients/delete/8`. The 'Body' tab is selected, showing a JSON payload:

```

1 {
2     "id": 8,
3     "personName": "pawani",
4     "personContactInfo": "071569623",
5     "personAddress": "238/A Nuwara,Nuwara eliya",
6     "patientMedHistory": "Having Cough",
7     "patientHealthStatus": "Normal"
8 }

```

The 'Test Results' tab shows the response: Status: 204 No Content, Time: 6 ms, Size: 112 B.

The bottom status bar shows the date and time: 8:26 AM 07/05/2024.

Figure 46:Postman DELETE method

The screenshot shows the Postman application interface. In the top navigation bar, there are links for Home, Workspaces, and API Network. A search bar is present, along with an Invite button, Upgrade button, and a minimize/maximize window control.

The main workspace is titled "My Workspace". It features a sidebar with sections for Collections, Environments, and History. A "Create a collection for your requests" button is visible.

A central panel displays a POST request to `http://localhost:8080/2022165_W1985549_CSA.CW/rest/patients/8`. The "Body" tab is selected, showing a JSON payload:

```
1 {  
2   ... "id": 8,  
3   ... "personName": "pawani",  
4   ... "personContactInfo": "071569623",  
5   ... "personAddress": "238/A Nukara.Numara eliya",  
6   ... "patientMedHistory": "Having Cough",  
7   ... "patientHealthStatus": "Normal"  
8 }
```

The response section shows a status of 404 Not Found, with a time of 76 ms and a size of 195 B. The message "Patient with this Id: 8 --> not found." is displayed.

The bottom of the screen shows the Windows taskbar with various pinned icons, including Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and others. The system tray indicates it's 8:27 AM on 07/05/2024, with a weather icon showing 27°C and light rain.

Figure 47:Postman DELETE method results by giving error message

## Doctor Entity

- GET

The screenshot shows the Postman application interface. In the left sidebar, there's a 'My Workspace' section with a 'Collections' tab selected. A 'My first collection' folder is expanded, containing two sub-folders: 'First folder inside collection' and 'Second folder inside collection'. Each folder contains a single item named 'doc'. Below the collection tree, there's a note about collections and a 'Create Collection' button.

In the main workspace, a request is being made to the URL `http://localhost:8080/2022165_W1985549_CSA_CW/rest/doctors/allDoctors`. The method is set to 'GET'. The 'Body' tab is selected, showing the raw JSON response. The response is a list of three doctors:

```
1 [
2   {
3     "id": 1,
4     "personName": "Dr.Sahan",
5     "personContactInfo": "Mobile: 0716521830",
6     "personAddress": "Dope,Bentota",
7     "docSpecialization": "Cardiologist",
8     "docContactDetails": "DocTel: 000001"
9   },
10   {
11     "id": 2,
12     "personName": "Dr.Upeksha",
13     "personContactInfo": "Mobile: 0714546464",
14     "personAddress": "Dope,Gallegawatta",
15     "docSpecialization": "Neurologist",
16     "docContactDetails": "DocTel: 000002"
17   },
18   {
19     "id": 3,
20     "personName": "Dr.Dulari",
21     "personContactInfo": "Mobile: 0715555555",
22     "personAddress": "Dope,Arachchimulla",
23     "docSpecialization": "Electrophysiologist"
24   }
]
```

The status bar at the bottom indicates the response was successful with a status of 200 OK, time 6 ms, and size 709 B. The system clock shows 8:29 AM on 07/05/2024.

Figure 48:Postman GET method with all doctors

This screenshot shows the same Postman interface as Figure 48, but with a different URL: `http://localhost:8080/2022165_W1985549_CSA_CW/rest/doctors/3`. The 'Body' tab is still selected, showing the raw JSON response for doctor ID 3:

```
1 [
2   {
3     "id": 3,
4     "personName": "Dr.Dulari",
5     "personContactInfo": "Mobile: 0715555555",
6     "personAddress": "Dope,Arachchimulla",
7     "docSpecialization": "Electrophysiologist",
8     "docContactDetails": "DocTel: 000003"
9   }
]
```

The status bar at the bottom indicates the response was successful with a status of 200 OK, time 4 ms, and size 347 B. The system clock shows 8:30 AM on 07/05/2024.

Figure 49:Postman GET method with doctor by id

- POST

The screenshot shows the Postman application interface. In the top navigation bar, the URL is set to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/doctors/create/10`. The 'Body' tab is selected, showing a JSON payload:

```

1 {
2   ...
3   "id": 10,
4   ...
5   "personName": "Dr.Thilak",
6   ...
7   "personContactInfo": "Mobile: 0771706623",
8   ...
9   "personAddress": "Beruwala,Aluthgama",
10  ...
11  "docSpecialization": "Cardiologist",
12  ...
13  "docContactDetails": "DocTel: 000010"
14 }

```

Below the body, the response status is shown as 204 No Content with a time of 6 ms. The bottom status bar indicates it's 8:31 AM on 07/05/2024.

Figure 50:Postman POST method adding doctors

The screenshot shows the Postman application interface. In the top navigation bar, the URL is set to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/doctors/10`. The 'Body' tab is selected, showing a JSON response:

```

1 {
2   ...
3   "id": 10,
4   ...
5   "personName": "Dr.Thilak",
6   ...
7   "personContactInfo": "Mobile: 0771706623",
8   ...
9   "personAddress": "Beruwala,Aluthgama",
10  ...
11  "docSpecialization": "Cardiologist",
12  ...
13  "docContactDetails": "DocTel: 000010"
14 }

```

Below the body, the response status is shown as 200 OK with a time of 4 ms. The bottom status bar indicates it's 8:31 AM on 07/05/2024.

Figure 51:Postman POST method adding doctors results

- PUT

The screenshot shows the Postman application interface. In the top navigation bar, the URL is set to `http://localhost:8080/20222165_W1985549_CSA.CW/rest/doctors/update/10`. The request method is selected as `PUT`. The `Body` tab is active, displaying a JSON payload:

```

1 {
2   "id": 10,
3   "personName": "Dr.Pahan",
4   "personContactInfo": "Mobile: 0655555556",
5   "personAddress": "Yapanaya,wamuniyawa",
6   "docSpecialization": "Neuralogist",
7   "docContactDetails": "DocTel: 000012"
8 }

```

Below the body, the response status is shown as `204 No Content` with a time of `5 ms` and a size of `112 B`.

Figure 52:Postman PUT method updating existing doctors

The screenshot shows the Postman application interface. In the top navigation bar, the URL is set to `http://localhost:8080/20222165_W1985549_CSA.CW/rest/doctors/10`. The request method is selected as `GET`. The `Body` tab is active, displaying a JSON payload:

```

1 {
2   "id": 10,
3   "personName": "Dr.Pahan",
4   "personContactInfo": "Mobile: 0655555556",
5   "personAddress": "Yapanaya,wamuniyawa",
6   "docSpecialization": "Neuralogist",
7   "docContactDetails": "DocTel: 000012"
8 }

```

Below the body, the response status is shown as `200 OK` with a time of `4 ms` and a size of `340 B`.

Figure 53:Postman PUT method updating existing doctors results

The screenshot shows the Postman application interface. In the top navigation bar, the URL is set to `PUT http://localhost:8080/2022165_W1985549_CSA_CW/rest/doctors/update/15`. The request body contains the following JSON:

```

1 {
2   ...
3   "id": 15,
4   ...
5   "personName": "Dr.Upeksha",
6   ...
7   "personContactInfo": "Mobile: 666666666",
8   ...
9   "personAddress": "Ambalangoda,galle",
10  ...
11  "docSpecialization": "Neurologist",
12  ...
13  "docContactDetails": "DocTel: 00001"
14 }

```

The response status is 404 Not Found, with the message: "A doctor with Id: 15 --> not found to update."

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

- **DELETE**

The screenshot shows the Postman application interface. In the top navigation bar, the URL is set to `DEL http://localhost:8080/2022165_W1985549_CSA_CW/rest/doctors/delete/10`. The request body contains the following JSON:

```

1 {
2   ...
3   "id": 10,
4   ...
5   "personName": "Dr.Palan",
6   ...
7   "personContactInfo": "Mobile: 0555555556",
8   ...
9   "personAddress": "Yepanaya,mmuniyama",
10  ...
11  "docSpecialization": "Neurologist",
12  ...
13  "docContactDetails": "DocTel: 000012"
14 }

```

The response status is 204 No Content.

Figure 55:Postman DELETE method

The screenshot shows the Postman application interface. In the left sidebar, under 'My Workspace', there is a 'Collections' section with 'My first collection' expanded, showing two folders: 'First folder inside collection' and 'Second folder inside collection', each containing one item. Below this, a note says: 'Create a collection for your requests. A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.' There is a 'Create Collection' button.

In the main workspace, a request is being made to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/doctors/10`. The method is set to 'DELETE'. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   ... "id": 10,  
3   ... "personName": "Dr.Pahan",  
4   ... "personContactInfo": "Mobile: 0555555555",  
5   ... "personAddress": "Yapanaya,wamuiyawa",  
6   ... "docSpecialization": "Neurologist",  
7   ... "docContactDetails": "DocTel: 000012"  
8 }
```

The 'Test Results' tab shows the response: Status: 404 Not Found, Time: 187 ms, Size: 105 B. The response body is: 'Doctor with this Id: 10 --> not found.'

The bottom of the screen shows the Windows taskbar with various pinned icons, including Postman, and the system tray indicating the date and time (07/05/2024, 8:34 AM).

Figure 56:Postman DELETE method results with giving error message

## Appointment Entity

- GET

The screenshot shows the Postman interface with a GET request to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/appointments/allAppointments`. The response status is 200 OK, and the response body is a JSON array of appointment objects:

```

1 [
2   {
3     "appointmentId": 1,
4     "date": "05/05",
5     "time": "10.00",
6     "patient": {
7       "id": 1,
8       "personName": "Kamal",
9       "personContactInfo": "0716521839",
10      "personAddress": "258/A Dope,Bentota",
11      "patientMedHistory": "Having Sugar",
12      "patientHealthStatus": "Normal"
13    },
14    "doctor": {
15      "id": 1,
16      "personName": "Dr.Gahan",
17      "personContactInfo": "0716521839",
18      "personAddress": "20001/LA",
19      "docSpecialization": "Cardiologist",
20      "docContactDetails": "DocTel: 000001"
21    }
22  },
23  {
24    "appointmentId": 2,
25    "date": "06/06",
26    "time": "11.00",
27    "patient": {}
28  }
29 ]

```

Figure 57:Postman GET method with all appointments

The screenshot shows the Postman interface with a GET request to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/appointments/2`. The response status is 200 OK, and the response body is a JSON object representing the second appointment:

```

1 {
2   "appointmentId": 2,
3   "date": "06/06",
4   "time": "11.00",
5   "patient": {
6     "id": 2,
7     "personName": "Piyal",
8     "personContactInfo": "0718909099",
9     "personAddress": "238/B Colombo,Fort",
10    "patientMedHistory": "Having Pressure",
11    "patientHealthStatus": "Good"
12  },
13  "doctor": {
14    "id": 2,
15    "personName": "Dr.Upeksha",
16    "personContactInfo": "9714545454",
17    "personAddress": "20002/B",
18    "docSpecialization": "Neurologist",
19    "docContactDetails": "DocTel: 000002"
20  }
21 }

```

Figure 58:Postman GET method with appointment by id

## ● POST

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar displays a collection named 'Your collection'. The main workspace shows a POST request to 'http://localhost:8080/2022165\_W1985549\_CSA\_CW/rest/appointments/create/202'. The 'Body' tab is selected, showing a JSON payload:

```

3 "date": "06/45",
4 "time": "11.11",
5 "patient": {
6     "id": 29,
7     "personName": "Sahan",
8     "personContactInfo": "6716521838",
9     "personAddress": "238/B Galle,Fort",
10    "patientMedHistory": "Having Fever",
11    "patientHealthStatus": "Normal"
12 },
13 "doctor": {}

```

The response status is 204 No Content, time 85 ms, size 112 B. The bottom status bar shows the date as 07/05/2024 and the time as 9:57 AM.

Figure 59: Postman POST method adding appointments

This screenshot shows Postman with a more complex collection structure. The 'My Workspace' sidebar shows a collection named 'My first collection' containing two folders: 'First folder inside collection' and 'Second folder inside collection', each with its own POST request. The main workspace shows a POST request to 'http://localhost:8080/2022165\_W1985549\_CSA\_CW/rest/appointments/202'. The 'Body' tab displays the same JSON payload as in Figure 59. The response status is 200 OK, time 4 ms, size 572 B. The bottom status bar shows the date as 07/05/2024 and the time as 9:58 AM.

Figure 60: Postman POST method adding appointments results

- PUT

The screenshot shows the Postman application interface. In the top navigation bar, the URL is set to `PUT http://localhost:8080/20222165_W1985549_CSA_CW/rest/appointments/update/202`. The 'Body' tab is selected, showing a JSON payload:

```

1 {
2   "appointmentId": 282,
3   "date": "01/45",
4   "time": "11.56",
5   "patient": {
6     "id": 2928,
7     "personName": "Pahan",
8     "personContactInfo": "123456789",
9     "personAddress": "238/B Colombo,67",
10    "patientMedHistory": "Having Cough",
11    "patientHealthStatus": "Good"
}

```

The status bar at the bottom indicates a successful response: Status: 204 No Content, Time: 6 ms, Size: 112 B.

Figure 61: Postman PUT method Updating existing appoinments

The screenshot shows the Postman application interface after a successful PUT request. The URL is now `GET http://localhost:8080/20222165_W1985549_CSA_CW/rest/appointments/202`. The 'Body' tab displays the updated appointment details, identical to the payload in Figure 61. The status bar at the bottom indicates a successful response: Status: 200 OK, Time: 4 ms, Size: 566 B.

Figure 62: Postman PUT method Updating existing appoinments results

The screenshot shows the Postman interface with a PUT request to `http://localhost:8080/20222165_W1985549_CSA_CW/rest/appointments/update/150`. The request body is a JSON object representing an appointment with ID 150. The response status is 404 Not Found, indicating the appointment was not found.

```

1 {
2     "appointmentId": 150,
3     "date": "07/22",
4     "time": "12:20",
5     "patient": {
6         "id": 700,
7         "personName": "Dulari",
8         "personContactInfo": "00006666",
9         "personAddress": "238/B Yapanaya, wawuniyawa",
10        "patientMedHistory": "Having Fever",
11        "patientHealthStatus": "Good"
}

```

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

## • DELETE

The screenshot shows the Postman interface with a DELETE request to `http://localhost:8080/20222165_W1985549_CSA_CW/rest/appointments/delete/202`. The request body is a JSON object representing an appointment with ID 202. The response status is 204 No Content, indicating the appointment was successfully deleted.

```

1 {
2     "appointmentId": 202,
3     "date": "01/45",
4     "time": "11:56",
5     "patient": {
6         "id": 2020,
7         "personName": "Pahan",
8         "personContactInfo": "123456789",
9         "personAddress": "238/B Colombo, 07",
10        "patientMedHistory": "Having cough",
11        "patientHealthStatus": "Good"
}

```

Figure 64: Postman DELETE method

The screenshot shows the Postman application interface. In the top navigation bar, there are links for Home, Workspaces, and API Network. A search bar is present, along with an Invite button and an Upgrade link. The main workspace is titled "My Workspace". On the left sidebar, there are sections for Collections, Environments, and History. A "Create a collection for your requests" section is visible. The central area displays a POST request to `http://localhost:8080/2022165_W1985549_CSA.CW/rest/appointments/202`. The "Body" tab is selected, showing a JSON payload:

```
1 {  
2     "appointmentId": 202,  
3     "date": "01/45",  
4     "time": "11.56",  
5     "patient": {  
6         "id": 2020,  
7         "personName": "Pahan",  
8         "personContactInfo": "123456789",  
9         "personAddress": "238/B Colombo, 07",  
10        "patientMedHistory": "Having cough",  
11        "patientHealthStatus": "Good"  
12    }  
13}
```

The "Test Results" tab shows the response: "Appointment with this Id: 202 --> not found." The status bar at the bottom indicates "Status: 404 Not Found Time: 280 ms Size: 201 B". The system tray shows the date and time as 07/05/2024 10:01 AM.

Figure 65: Postman *DELETE* method results with error printing

# Medical Record Entity

- GET

The screenshot shows the Postman application interface. In the top bar, there are tabs for Home, Workspaces, and API Network. The main area displays a collection named "Your collection". A specific request is selected, showing a GET method at the URL `http://localhost:8080/2022165_W1985549_CSA_CW/rest/medRecords/allMedRecords`. The response status is 200 OK, time is 3 ms, and size is 919 B. The response body is displayed in JSON format, showing two medical records. The first record has a medRecId of 1, diagnoses of "Having Sugaz", treatments of "Getting sugar pills", and a patient with id 1, name Kamal, contact info 0716521830, address 258/A Dope,Bentota, history of "Having Sugaz", and health status Normal. The second record has a medRecId of 2, diagnoses of "Having Pressure", treatments of "Getting pressure pills", and a patient with id 2, name Piyal, contact info 0718989999, address 238/B Colombo,Fort, history of "Having Pressure", and health status Good.

```
[{"medRecId": 1, "diagnoses": "Having Sugaz", "treatments": "Getting sugar pills", "patient": {"id": 1, "personName": "Kamal", "personContactInfo": "0716521830", "personAddress": "258/A Dope,Bentota", "patientMedHistory": "Having Sugaz", "patientHealthStatus": "Normal"}, "medRecId": 2, "diagnoses": "Having Pressure", "treatments": "Getting pressure pills", "patient": {"id": 2, "personName": "Piyal", "personContactInfo": "0718989999", "personAddress": "238/B Colombo,Fort", "patientMedHistory": "Having Pressure", "patientHealthStatus": "Good"}]
```

Figure 66: Postman GET method with all Medical Records

The screenshot shows the Postman application interface. In the top bar, there are tabs for Home, Workspaces, and API Network. The main area displays a collection named "Your collection". A specific request is selected, showing a GET method at the URL `http://localhost:8080/2022165_W1985549_CSA_CW/rest/medRecords/2`. The response status is 200 OK, time is 4 ms, and size is 416 B. The response body is displayed in JSON format, showing a single medical record with a medRecId of 2, diagnoses of "Having Pressure", treatments of "Getting pressure pills", and a patient with id 2, name Piyal, contact info 0718989999, address 238/B Colombo,Fort, history of "Having Pressure", and health status Good.

```
{"medRecId": 2, "diagnoses": "Having Pressure", "treatments": "Getting pressure pills", "patient": {"id": 2, "personName": "Piyal", "personContactInfo": "0718989999", "personAddress": "238/B Colombo,Fort", "patientMedHistory": "Having Pressure", "patientHealthStatus": "Good"}}
```

Figure 67: Postman GET method with Medical Record by id

- POST

The screenshot shows the Postman application interface. In the left sidebar, there's a 'My Workspace' section with a 'Collections' tab selected, displaying two collections: 'First folder inside collection' and 'Second folder inside collection'. Below this, there's a 'Create a collection for your requests' section with a 'Create Collection' button.

The main workspace shows a POST request to [http://localhost:8080/2022165\\_W1985549\\_CSA\\_CW/rest/medRecords/create/22](http://localhost:8080/2022165_W1985549_CSA_CW/rest/medRecords/create/22). The 'Body' tab is selected, showing a JSON payload:

```

1 {
2     "medRecId": 22,
3     "diagnoses": "Having Cough",
4     "treatments": "Getting Cough pills",
5     "patient": {
6         "id": 159,
7         "personName": "Nishantha",
8         "personContactInfo": "0716521838",
9         "personAddress": "238/B Galle,Rathgama",
10        "patientMedHistory": "Having Cough",
11        "patientHealthStatus": "Good"
12    }
13

```

The 'Test Results' panel at the bottom shows a status of 204 No Content, time 6 ms, and size 112 B. The Postman toolbar at the bottom includes icons for Postbot, Runner, Start Proxy, Cookies, Vault, and Trash.

Figure 68: Postman POST method with adding Medical Records

This screenshot is identical to Figure 68, showing the same POST request setup and JSON payload. The difference is in the 'Test Results' panel, which now displays a successful response with status 200 OK, time 4 ms, and size 416 B. The JSON response body is identical to the one sent in the request.

Figure 69: Postman POST method with adding Medical Records results

## ● PUT

The screenshot shows the Postman application interface. In the left sidebar, under 'My Workspace', there is a collection named 'My first collection' which contains two folders: 'First folder inside collection' and 'Second folder inside collection'. Below this, there is a section for creating a collection with the message: 'Create a collection for your requests'. A note states: 'A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.' At the bottom of the sidebar, there are buttons for 'Create Collection' and 'Collections'.

In the main workspace, a PUT request is being made to the URL `http://localhost:8080/20222165_W1985549_CSA.CW/rest/medRecords/update/10`. The 'Body' tab is selected, showing a JSON payload:

```

1 {
2     "medRecId": 10,
3     "diagnoses": "Having Cough",
4     "treatments": "Getting Cough pills",
5     "patient": [
6         {
7             "id": 298,
8             "personName": "Upesha",
9             "personContactInfo": "6712596048",
10            "personAddress": "258/A Colombo,10",
11            "patientMedHistory": "Having Cough",
12            "patientHealthStatus": "Normal"
13        }
14    ]
15 }

```

Below the body, the response status is shown as 'Status: 204 No Content Time: 5 ms Size: 112 B'. The bottom of the screen shows the Windows taskbar with various pinned icons.

Figure 70: Postman PUT method with updating existing Medical Records

This screenshot is identical to Figure 70, showing the same Postman interface and the same PUT request to update medical records. The JSON payload is the same, and the response status is 'Status: 204 No Content Time: 5 ms Size: 112 B'. The bottom of the screen shows the Windows taskbar with various pinned icons.

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

The screenshot shows the Postman interface with a PUT request to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/medRecords/update/100`. The request body is a JSON object:

```

1 {
2     "medRecId": 100,
3     "diagnoses": "Having virus",
4     "treatments": "Getting virus pills",
5     "patient": {
6         "id": 500,
7         "personName": "nuwanthika",
8         "personContactInfo": "0000000000",
9         "personAddress": "258/A galle,10",
10        "patientMedHistory": "Having virus",
11        "patientHealthStatus": "Normal"
12    }

```

The response status is 404 Not Found, indicating the record was not found.

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

## • DELETE

The screenshot shows the Postman interface with a DELETE request to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/medRecords/delete/10`. The request body is a JSON object:

```

1 {
2     "medRecId": 10,
3     "diagnoses": "Having Cough",
4     "treatments": "Getting Cough pills",
5     "patient": {
6         "id": 200,
7         "personName": "Upeksha",
8         "personContactInfo": "6712590048",
9         "personAddress": "258/A Colombo,10",
10        "patientMedHistory": "Having Cough",
11        "patientHealthStatus": "Normal"
12    }

```

The response status is 204 No Content, indicating the record was deleted.

Figure 73: Postman DELETE method for medical records

The screenshot shows the Postman application interface. In the top navigation bar, 'Home' is selected. The main workspace is titled 'My Workspace'. A collection named 'Your collector' is open, containing a single item named 'Your collection'. The collection has an 'Authorization' step set to 'API Key'. Below the collection, there is a note about creating a collection for requests and a 'Create Collection' button.

In the center, a request is being made to the URL `http://localhost:8080/2022165_W1985549_CSA_CW/rest/medRecords/10`. The method is set to 'GET'. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   "medRecId": 16,  
3   "diagnoses": "Having Cough",  
4   "treatments": "Getting Cough pills",  
5   "patient": {  
6     "id": 209,  
7     "personName": "Upeksha",  
8     "personContactInfo": "6712596048",  
9     "personAddress": "258/A Colombo,10",  
10    "patientMedHistory": "Having Cough",  
11    "patientHealthStatus": "Normal"  
12  }  
... ?
```

The response status is 404 Not Found, with a time of 62 ms and a size of 203 B. The response body is: 'Medical record with this Id: 10 --> not found.'

The bottom of the screen shows the Windows taskbar with various pinned icons, including Postbot, Runner, Start Proxy, Cookies, Vault, and Trash. The system tray shows the date and time as 07/05/2024 at 8:55 AM.

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

# Prescription Entity

- GET

The screenshot shows the Postman application interface. The left sidebar displays 'My Workspace' with a collection named 'My first collection'. The main workspace shows a GET request for 'http://localhost:8080/2022165\_W1985549\_CSA\_CW/rest/prescriptions/allPrescripts'. The 'Body' tab is selected, showing the raw JSON response. The response is a list of prescriptions, each containing details like prescription ID, dosage, medication info, duration in days, and patient information. The status bar at the bottom indicates 'Status: 200 OK'.

```
1 [
2   {
3     "prescriptId": 1,
4     "dosage": "100mg",
5     "medInfo": "After meals three times a day",
6     "durationInDays": 5,
7     "patient": {
8       "id": 1,
9       "personName": "Kamal",
10      "personContactInfo": "0716521839",
11      "personAddress": "258/A Dope,Bentota",
12      "patientMedHistory": "Having Sugar",
13      "patientHealthStatus": "Normal"
14    }
15  },
16  {
17    "prescriptId": 2,
18    "dosage": "250mg",
19    "medInfo": "Before meals two times a day",
20    "durationInDays": 3,
21    "patient": {
22      "id": 2,
23      "personName": "Piyal",
24    }
25  }
]
```

Figure 75: Postman GET method by all prescriptions

The screenshot shows the Postman application interface. The left sidebar displays 'My Workspace' with a collection named 'My first collection'. The main workspace shows a GET request for 'http://localhost:8080/2022165\_W1985549\_CSA\_CW/rest/prescriptions/2'. The 'Body' tab is selected, showing the raw JSON response. The response is a single prescription with details like prescription ID, dosage, medication info, duration in days, and patient information. The status bar at the bottom indicates 'Status: 200 OK'.

```
1 {
2   "prescriptId": 2,
3   "dosage": "250mg",
4   "medInfo": "Before meals two times a day",
5   "durationInDays": 3,
6   "patient": {
7     "id": 2,
8     "personName": "Piyal",
9     "personContactInfo": "0718989899",
10    "personAddress": "238/B Colombo,Fort",
11    "patientMedHistory": "Having Pressure",
12    "patientHealthStatus": "Good"
13  }
14 }
```

Figure 76: Postman GET method prescriptions by id

## ● POST

The screenshot shows the Postman application interface. In the left sidebar, there's a 'My Workspace' section with a 'Collections' tab selected. A collection named 'My first collection' is expanded, showing two folders: 'First folder inside collection' and 'Second folder inside collection', each containing several requests. Below this, there's a 'Create a collection for your requests' section with a 'Create Collection' button.

The main workspace shows a POST request to `http://localhost:8080/20222165_W1985549_CSA.CW/rest/prescriptions/create/22`. The 'Body' tab is selected, displaying a JSON payload:

```

1 {
2     "prescriptId": 22,
3     "dosage": "500mg",
4     "medInfo": "After meals two times a day",
5     "durationInDays": 10,
6     "patient": [
7         {
8             "id": 205,
9             "personName": "Simon",
10            "personContactInfo": "6716524830",
11            "personAddress": "238/B Galle,25",
12            "patientMedHistory": "Having Cough",
13            "patientHealthStatus": "Good"
14        }
15    ]
16}

```

The 'Test Results' tab at the bottom shows a successful response with status 204 No Content, time 6 ms, and size 112 B.

Figure 77: Postman POST method by adding prescriptions

This screenshot is similar to Figure 77 but shows the results of the POST request. The 'Body' tab still displays the same JSON payload, but the 'Test Results' tab now shows a successful response with status 200 OK, time 4 ms, and size 425 B.

The 'Headers' tab shows the following response headers:

```

Content-Type: application/json
Content-Length: 425
Date: Mon, 07 Jul 2024 09:02:43 GMT
Connection: keep-alive
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, PUT, DELETE, PATCH, HEAD, OPTIONS
Access-Control-Allow-Headers: Content-Type, Authorization, X-CSRF-Token
Access-Control-Max-Age: 3600
Access-Control-Allow-Credentials: true

```

Figure 78: Postman POST method by adding prescriptions results

## ● PUT

The screenshot shows the Postman application interface. The left sidebar displays 'My Workspace' with a collection named 'Your collection'. The main workspace shows a PUT request to `http://localhost:8080/2022165_W1985549_CSA.CW/rest/prescriptions/update/22`. The 'Body' tab is selected, showing a JSON payload:

```

1 {
2     "prescriptId": 22,
3     "dosage": "15mg",
4     "medInfo": "Before meals two times a day",
5     "durationInDays": 25,
6     "patient": [
7         {"id": 1096,
8             "personName": "Dulari",
9             "personContactInfo": "00000000",
10            "personAddress": "23/B Galle,25",
11            "patientMedHistory": "Having Cough",
12            "patientHealthStatus": "Good"

```

The response status is 204 No Content. The bottom navigation bar shows various icons including NASDAQ, Search, and browser tabs.

Figure 79: Postman PUT method by updating existing prescriptions

The screenshot shows the Postman application interface, similar to Figure 79 but with a GET request instead. The main workspace shows a GET request to `http://localhost:8080/2022165_W1985549_CSA.CW/rest/prescriptions/22`. The 'Body' tab is selected, showing the same JSON payload as in Figure 79. The response status is 200 OK, with a response body identical to the request body:

```

1 {
2     "prescriptId": 22,
3     "dosage": "15mg",
4     "medInfo": "Before meals two times a day",
5     "durationInDays": 25,
6     "patient": [
7         {"id": 1096,
8             "personName": "Dulari",
9             "personContactInfo": "00000000",
10            "personAddress": "23/B Galle,25",
11            "patientMedHistory": "Having Cough",
12            "patientHealthStatus": "Good"

```

The bottom navigation bar shows various icons including NASDAQ, Search, and browser tabs.

Figure 80: Postman PUT method by updating existing prescriptions results

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar displays 'Your collection' with an 'Authorization' tab selected. The main workspace shows a PUT request to `http://localhost:8080/2022165_W1985549_CSA.CW/rest/prescriptions/update/22222`. The 'Body' tab is selected, showing a JSON payload:

```
2 ... "prescriptionId": 22222,
3 ... "dosage": "50mg",
4 ... "medInfo": "After meals two times a day",
5 ... "durationInDays": 5,
6 ... "patient": [
7 ...     {
8 ...         "id": 259,
9 ...         "personName": "Isira",
10 ...         "personContactInfo": "78989456",
11 ...         "personAddress": "238/B Galle,25",
12 ...         "patientMedHistory": "Having Cough",
13 ...         "patientHealthStatus": "Normal"
14 ...     }
15 ... ]
```

The 'Test Results' tab shows a 404 Not Found error message: 'A prescription with id: 22222 --> not found to update.'

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

## • DELETE

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar displays 'My first collection' with a 'First folder inside collection' containing 'test' and 'test1' files, and a 'Second folder inside collection' containing 'test' and 'test1' files. The main workspace shows a DELETE request to `http://localhost:8080/2022165_W1985549_CSA.CW/rest/prescriptions/delete/22`. The 'Body' tab is selected, showing a JSON payload:

```
1 ...
2 ... "prescriptionId": 22,
3 ... "dosage": "15mg",
4 ... "medInfo": "Before meals two times a day",
5 ... "durationInDays": 26,
6 ... "patient": [
7 ...     {
8 ...         "id": 1996,
9 ...         "personName": "Dulari",
10 ...         "personContactInfo": "60000000",
11 ...         "personAddress": "238/B Galle,25",
12 ...         "patientMedHistory": "Having Cough",
13 ...         "patientHealthStatus": "Good"
14 ...     }
15 ... ]
```

The 'Test Results' tab shows a 204 No Content status message.

Figure 82: Postman DELETE method

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar displays 'My first collection' with two items: 'First folder inside collection' and 'Second folder inside collection'. Below this, there is a note about collections and a 'Create Collection' button. The main workspace shows a 'GET' request to 'http://localhost:8080/20222165\_W1985549\_CSA\_CW/rest/prescriptions/22'. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2     "prescriptionId": 22,  
3     "dosage": "15mg",  
4     "medInfo": "Before meals two times a day",  
5     "durationInDays": 25,  
6     "patient": [  
7         {"id": 1096,  
8             "personName": "Dulazi",  
9             "personContactInfo": "60000000",  
10            "personAddress": "238/0 Galle,25",  
11            "patientMedHistory": "Having Cough",  
12            "patientHealthStatus": "Good"}]
```

The 'Test Results' tab shows the response: 'Prescription with this Id: 22 -> not found.' The status bar at the bottom indicates 'Status: 404 Not Found'.

Figure 83: Postman DELETE method results and error printing

# Billing Entity

- GET

The screenshot shows the Postman application interface. In the top navigation bar, 'Home' and 'Workspaces' are visible. The main area displays a collection named 'Your collector' with one item named 'Your collection'. A sub-section titled 'Create a collection for your requests' is present. On the right, a request is being made to the URL `http://localhost:8080/2022165_W1985549_CSA_CW/rest/billResource/allBills`. The method is set to 'GET'. The 'Body' tab is selected, showing a raw JSON response. The response contains two bill objects:

```
1 [
2   {
3     "billingId": 1,
4     "invoice": "Invoice 01",
5     "payment": 5000.0,
6     "ifPaid": "Paid",
7     "patient": [
8       {
9         "id": 1,
10        "personName": "Kamal",
11        "personContactInfo": "0716521839",
12        "personAddress": "258/A Dope,Bentota",
13        "patientMedHistory": "Having Sugar",
14        "patientHealthStatus": "Normal"
15      }
16    ],
17   {
18     "billingId": 2,
19     "invoice": "Invoice 02",
20     "payment": 2500.0,
21     "ifPaid": "Not Paid",
22     "patient": [
23       {
24         "id": 2,
25         "personName": "Piyal",
26       }
27     ]
28   }
29 ]
```

The status bar at the bottom indicates '9:13 AM 07/05/2024'.

Figure 84: Postman GET method with all bills

The screenshot shows the Postman application interface. The left sidebar shows 'My Workspace' with 'Collections' and 'Environments'. A collection named 'Your collector' is selected, containing one item named 'Your collection'. A sub-section titled 'Create a collection for your requests' is present. On the right, a request is being made to the URL `http://localhost:8080/2022165_W1985549_CSA_CW/rest/billResource/1`. The method is set to 'GET'. The 'Body' tab is selected, showing a raw JSON response. The response contains a single bill object:

```
1 [
2   {
3     "billingId": 1,
4     "invoice": "Invoice 01",
5     "payment": 5000.0,
6     "ifPaid": "Paid",
7     "patient": [
8       {
9         "id": 1,
10        "personName": "Kamal",
11        "personContactInfo": "0716521839",
12        "personAddress": "258/A Dope,Bentota",
13        "patientMedHistory": "Having Sugar",
14        "patientHealthStatus": "Normal"
15      }
16    ],
17   {
18     "billingId": 2,
19     "invoice": "Invoice 02",
20     "payment": 2500.0,
21     "ifPaid": "Not Paid",
22     "patient": [
23       {
24         "id": 2,
25         "personName": "Piyal",
26       }
27     ]
28   }
29 ]
```

The status bar at the bottom indicates '9:13 AM 07/05/2024'.

Figure 85: Postman GET method with bill id

## ● POST

The screenshot shows the Postman application interface. In the left sidebar, there's a 'My Workspace' section with a 'Collections' tab selected. A 'Create a collection for your requests' note is present. The main workspace shows a POST request to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/billResource/create/22`. The 'Body' tab is active, displaying a JSON payload:

```

1 {
2     "billingId": 22,
3     "Invoice": "Invoice 22",
4     "payment": 1500.0,
5     "ifPaid": "Not Paid",
6     "patient": [
7         {
8             "id": 100,
9             "personName": "Bimal",
10            "personContactInfo": "0771700623",
11            "personAddress": "258/A Galle,Bentota",
12            "patientMedHistory": "Having Fever",
13            "patientHealthStatus": "Good"
14        }
15    ]
16}

```

The status bar at the bottom indicates: Status: 204 No Content, Time: 65 ms, Size: 112 B.

Figure 86: Postman POST method with adding bills

This screenshot is identical to Figure 86, showing the same POST request setup and JSON payload. The difference is in the response. The status bar now shows: Status: 200 OK, Time: 3 ms, Size: 410 B.

Figure 87: Postman POST method with adding bills results

## ● PUT

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar displays 'My first collection' and 'Second folder inside collection'. The main workspace shows a 'PUT' request to 'http://localhost:8080/2022165\_W1985549\_CSA\_CW/rest/billResource/update/22'. The 'Body' tab is selected, showing a JSON payload:

```

1   {
2     "billingId": 22,
3     "invoice": "Invoice_500",
4     "payment": 3500.0,
5     "ifPaid": "Paid",
6     "patient": [
7       {
8         "id": 259,
9         "personName": "Samadhi",
10        "personContactInfo": "9775694987",
11        "personAddress": "258/A Beruwala,Colombo",
12        "patientMedHistory": "Having Caugh",
13        "patientHealthStatus": "Normal"
14      }
15    ]
16  }

```

The 'Test Results' tab shows a successful response: Status: 204 No Content, Time: 6 ms, Size: 112 B.

Figure 88: Postman PUT method with updating existing bill

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar displays 'Your collector' and 'Your collection'. The main workspace shows a 'GET' request to 'http://localhost:8080/2022165\_W1985549\_CSA\_CW/rest/billResource/22'. The 'Body' tab is selected, showing a JSON payload identical to Figure 88:

```

1   {
2     "billingId": 22,
3     "invoice": "Invoice_500",
4     "payment": 3500.0,
5     "ifPaid": "Paid",
6     "patient": [
7       {
8         "id": 259,
9         "personName": "Samadhi",
10        "personContactInfo": "9775694987",
11        "personAddress": "258/A Beruwala,Colombo",
12        "patientMedHistory": "Having Caugh",
13        "patientHealthStatus": "Normal"
14      }
15    ]
16  }

```

The 'Test Results' tab shows a successful response: Status: 200 OK, Time: 4 ms, Size: 415 B.

Figure 89: Postman PUT method with updating existing bill results

The screenshot shows the Postman interface with a PUT request to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/billResource/update/200`. The request body is a JSON object representing a bill with various fields like billingId, invoice, payment, and patient. The response status is 404 Not Found, indicating the resource was not found.

```

1 {
2     "billingId": 200,
3     "invoice": "Invoice_700",
4     "payment": 4500.0,
5     "ifPaid": "Not Paid",
6     "patient": [
7         {
8             "id": 411,
9             "personName": "Sandun",
10            "personContactInfo": "6342270641",
11            "personAddress": "258/A Galle,Fort",
12            "patientMedHistory": "Having Fever",
13            "patientHealthStatus": "Normal"
14        }
15    ]
16}

```

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

## • DELETE

The screenshot shows the Postman interface with a DELETE request to `http://localhost:8080/2022165_W1985549_CSA_CW/rest/billResource/delete/22`. The request body is a JSON object representing a bill with various fields. The response status is 204 No Content, indicating the resource was successfully deleted.

```

1 {
2     "billingId": 22,
3     "invoice": "Invoice_500",
4     "payment": 3500.0,
5     "ifPaid": "Paid",
6     "patient": [
7         {
8             "id": 259,
9             "personName": "Samadhi",
10            "personContactInfo": "0775694987",
11            "personAddress": "258/A Beruwala,Colombo",
12            "patientMedHistory": "Having Cough",
13            "patientHealthStatus": "Normal"
14        }
15    ]
16}

```

Figure 91: Postman DELETE method

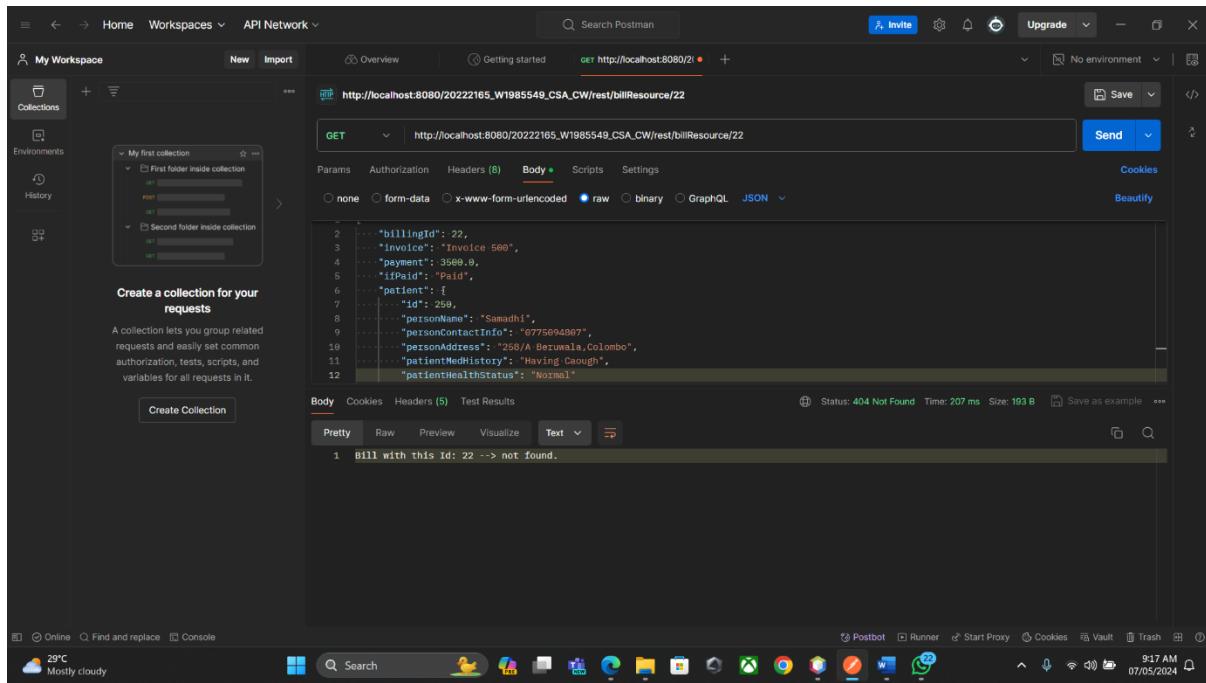


Figure 92: Postman DELETE method results with printing error

## 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.