**Project4: Medical Diagnostic Laboratory**

**Back End Development**

|  |  |
| --- | --- |
| \ |  |

**FUNCTIONAL SPECIFICATION**

|  |  |
| --- | --- |
| **ProjectCode:** |  |
| **ProjectName:** | Medical Diagnostic Laboratory |

**FUNCTIONAL SPECIFICATION**

**Table of Contents**

1. INTRODUCTION 3

2. SYSTEM OVERVIEW 3

3. SUB-SYSTEM DETAILS 4

4. DATA ORGANIZATION 5

5. REST APIs to be Built 6

6. ASSUMPTIONS 9

7. EXPECTATIONS 9

8. ACCEPTANCE CRITERIA 9

9. TRACEABILITY TO REQUIREMENTS 9



**FUNCTIONAL SPECIFICATION**

**1 Introduction**

Health Technologies Ltd is a company which builds a software system which is responsible for analyzing testreports of a patient.

Health Technologies Ltd plans to develop "Medical Diagnostic Laboratory " - web application [J2EE Batches - Web Application], where users can register, login, tests are carried out on clinical specimens to obtain information about the health of a patient.

**Scope and Overview:**

The scope of the “Medical Diagnostic Laboratory” will be to provide the functionality as described below. The system will be developed on a Windows operating system using Java/J2EE, Hibernate, Spring.

**2 System Overview**

The “Medical Diagnostic Laboratory” should support basic functionalities (explained in section 2.1) for all below listed users.

* Administrator (A)
* Patient(P)

***2.1 Authentication & Authorization***

***2.1.1 Authentication****:*

Any end-user should be authenticated using a unique patientid and password.

***2.1.2*** ***Authorization***

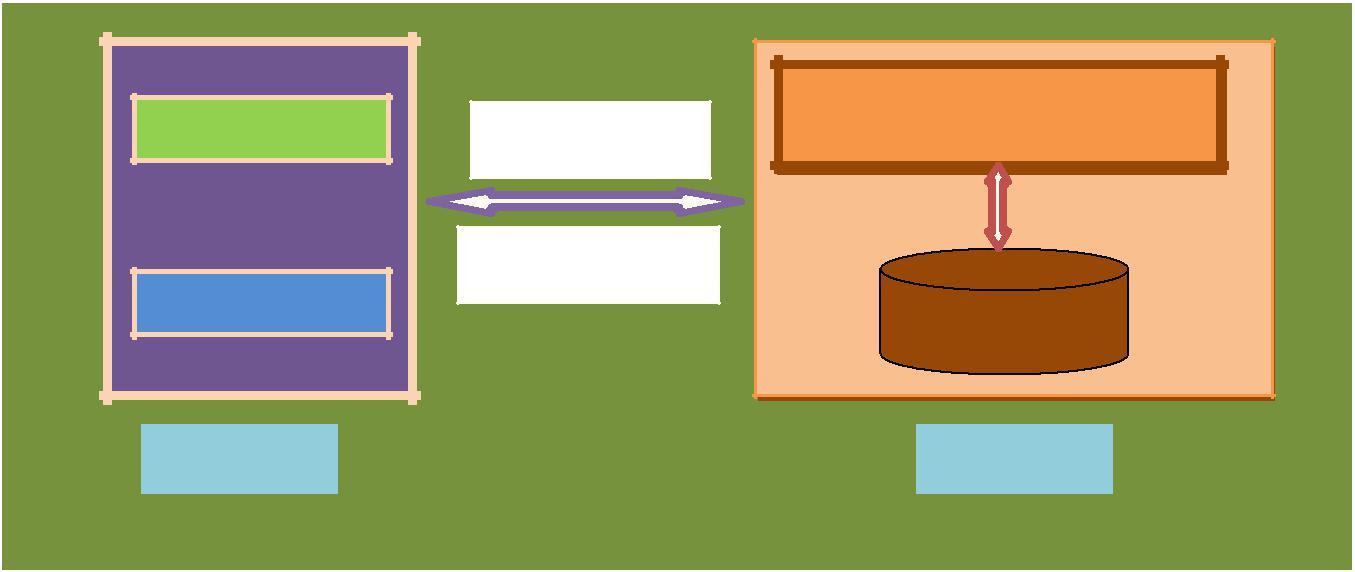
The operations supported and allowed would be based on the patient type. For example, Administrator has the rights to add patient information and view patient details. He can also view testreport and price details of a test.

Whereas Patient has a right to Add, Remove and Clear all the testdetails from testreport.

***2.2 Functional Flow***

The functional flow of the messages across different application components is shown below.

Ex. - Web Application.



|  |  |  |
| --- | --- | --- |
| **Administrator** | **MDL Application** |  |
| HTTP/GET |  |
|  | HTTP/Response |  |
| **Patient** | Database |  |
|  |  |
| Client GUI | Server |  |
|  | **Medical Diagnostic Laboratory** |  |
|  |  |  |
| ***2.3 Environment*** |  |  |

The system will be developed on any Windows OSmachine using J2EE, Hibernate and Spring.

* Intel hardware machine (PC P4-2.26 GHz, 512 MB RAM, 40 GB HDD)
* Server – Apache Tomcat 9
* Database – Oracle 18g
* JRE 8
* Spring Tool Suite



**FUNCTIONAL SPECIFICATION**

**3 Sub-system Details**

Medical Diagnostic Laboratory is defined, wherein all users need to login successfully before performing any of their respective operations.

Find below (section 3.1 & 3.2) tables that provides functionality descriptions for each type of patient / sub-system. Against each requirement, indicative data is listed in column ‘Data to include’. Further, suggested to add/modify more details wherever required with an approval from customer/faculty.

***3.1 Administrator***

The administrator as a user is defined to perform below listed operations after successful login.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Objects | Operations | Data to include | Remarks |
| AD-001  To  AD-004 | Doctor | Add  View  Modify  Delete | doctorId, doctorFirstName,  doctorLastName, doctorPhoneNumber, doctorEmail, qualification, profilePicture |  |
| AD-005  To  AD-0010 | Patient | Add  View  Modify  Delete | patientId, patientFirstName,  patientLastName, age, gender, phoneNumber, email |  |
| AD – 0011  to  AD - 0013 | Technician | Add  View  Modify  Delete | technicianId, technicianFirstName, technicianLastName, technicianEmailId, technicianPhoneNumber |  |

***3.2 Patient***

The patient as a user is defined to perform below listed operations after successful login.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Objects | Operations | Data to include |  | Remarks | | | | |
| US-001 | Patient | Register  Login | patientId, patientFirstName,  patientLastName, age, gender, phoneNumber |  |  |  |  |  |  |
| US-002 | testreport | View | testreportId, billingPrice, result, patient, medicalTest, technician |  |  |  |  |  |  |
| US-003 | Checkout | ViewDetails  and Price | doctor, patientId and Total Price |  |  |  |  |  |  |

**FUNCTIONAL SPECIFICATION**

***3.3 Login | Logout***

**[Web Application - J2EE, Hibernate, Spring]**

* Go to Registration screen when you click on Register link
* Go to Success screen when you login successfully after entering valid username & password fetched from the database.
* Redirect back to login screen if username & password are not matching
* Implement Session tracking for all logged in users before allowing access to application features. Anonymous users should be checked, unless explicitly mentioned.

**4 Data Organization**

This section explains the data storage requirements of the Laboratory Management System and **indicative** data description along with suggested table (database) structure. The following section explains few of the tables (fields) with description. However, in similar approach need to be considered for all other tables.

***4.1 Table: Patient\_Registration\_Details***

The user specific details such as username, email, phone etc. Authentication, and authorization / privileges should be kept in one or more tables, as necessary and applicable.

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| *PatientID* | PatientID is auto generated after registration and it is used as |
|  | LoginID. |
| *PatientFirstName* | FirstName of the Patient |
| *PatientLastName* | LastName of the Patient |
| *Age* | Patient’s Age |
| *Gender* | Gender of a Patient |
| *PhoneNumber* | 10 digit contact number |
| *Email* | EmailId of patient |
| *Password* | Password of patient |

***4.2 Table: Test\_Details***

This table contains information related to a product

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| *Test Id* | Unique Test Id, Here Test Id will be Primary Key |
| *Test Name* | Name of the Test e.g. blood test |
| *Test Category* |  |
| *Test Price* | Price of the Test |



***4.3 Table: TestReport\_Details***

This table contains information related to cart details

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| *TestReportId* | Unique TestReport ID Auto Generated |
| *PatientId* | PatientId corresponding to logged in Patient |
| *TestId* | TestId corresponding to logged in patient |
| *BillingPrice* | Price of the test |
| *Result* | Outcome of testreport |

***4.4 Table: Appointment\_Details***

This table contains information related to final checkout details

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| *AppointmentId* | Unique Appointment Id, this field must be primary key |
| *PatientId* | Registered Patient Id, this field should be foreign key |
| *DoctorId* | Unique DoctorId.this field must be foreign key |
| *TestId* | Unique TestId this field must be foreign key |
| *Date* | Unique date for unique PatientId |
| *TimeSlot* | Unique TimeSlot.for unique patientId |

***4.4 Table: Doctor\_Details***

This table contains information related to final checkout details

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| *DoctorId* | Unique Doctor Id, this field must be primary key |
| *DoctorFirstName* | FirstName of a doctor |
| *DoctorLastName* | LastName of a doctor |
| *Qualification* | Qualification of doctor |
| *DoctorPhoneNumber* | Phone number of doctor |
| *DoctorEmail* | Email of the doctor |
| *DoctorprofilePicture* | Picture of the doctor |

***4.4 Table: technician\_Details***

This table contains information related to final checkout details

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| *TechnicianId* | Unique TechnicianId, this field must be primary key |
| *TechnicianFirstName* | FristName of the Technician |
| *TechnicianLastName* | LastName of the Technician |
| *TechnicianPhoneNumber* | PhoneNumber of the Technician |
| *TechincianEmail* | Email of the Technician |

**FUNCTIONAL SPECIFICATION**

1. **REST APIsto be Built**

Create following REST resources which are required in the application,

1. Creating **User** Entity: Create Spring Boot with Microservices Application with Spring Data JPA

**Technology stack:**

* Spring Boot
* Spring REST
* Spring Data JPA

Here will have multiple layers into the application:

1. Create an Entity: Patient
2. Create a PatientRepository interface and will make use of Spring Data JPA
3. Will have findPatientById method
4. Add the Patient details
5. Create a PatientService class and will expose all these services
6. Finally, create a PatientController will have the following Uri’s:

|  |  |  |  |
| --- | --- | --- | --- |
| URI | METHODS | Description | Format |
| /patient/ | GET | Give a multiple patients description | JSON |
| /patient/patientId | GET | Give a single patient description searched based on patientId | String |
| /patient/ | POST | Add the patient details | JSON |
| /patient/ | PUT | Update the patient details | JSON |
| /patient/patientId/ | DELETE | Delete patient by id | String |

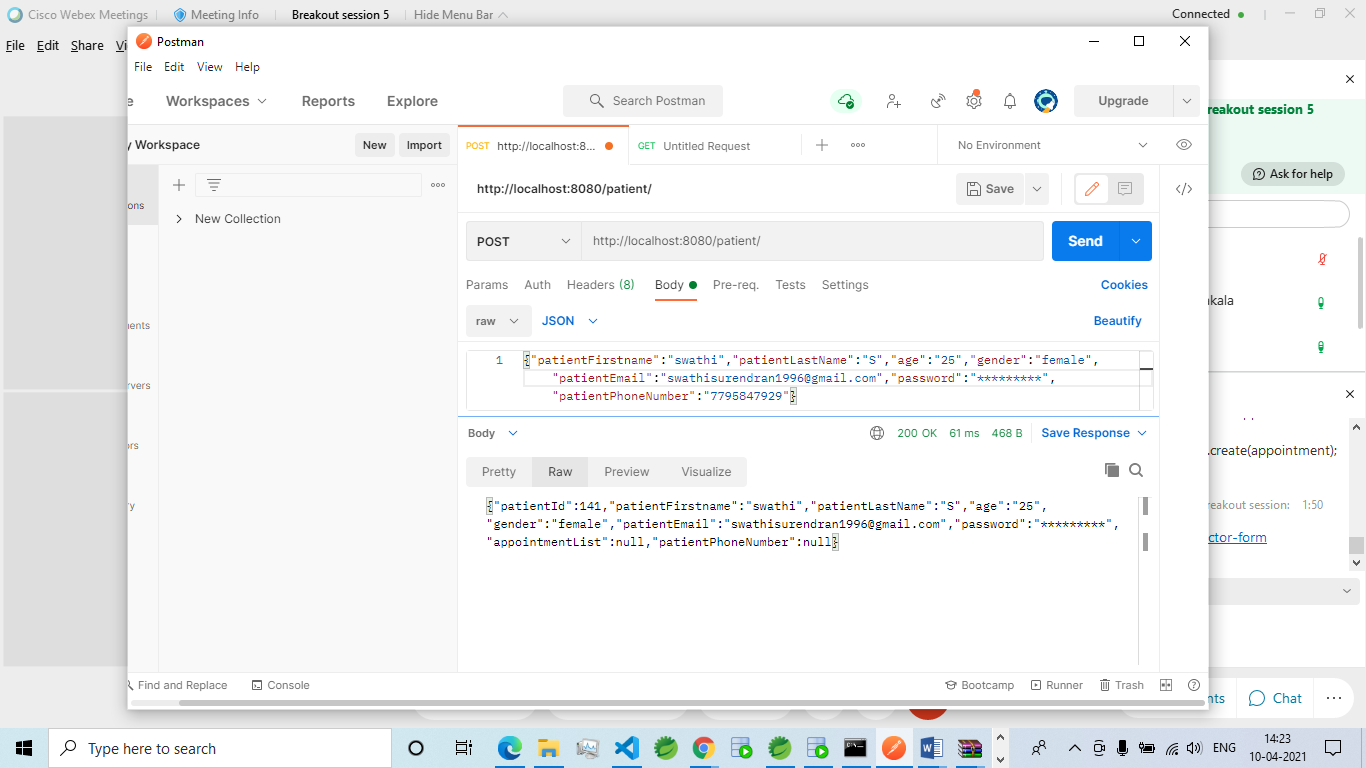
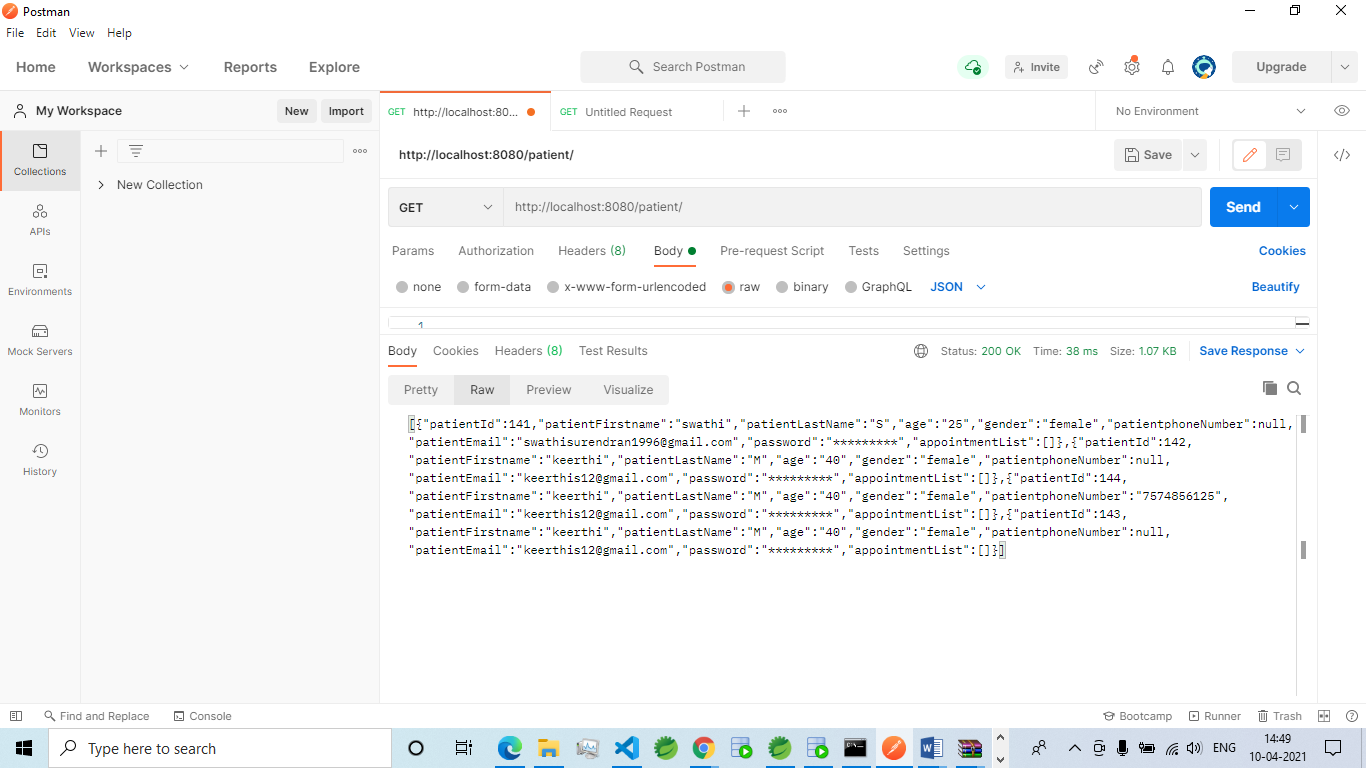
**Patient:**

**Review of POST Method using POSTMAN**

Creation of Patient object using POST Method in POSTMAN

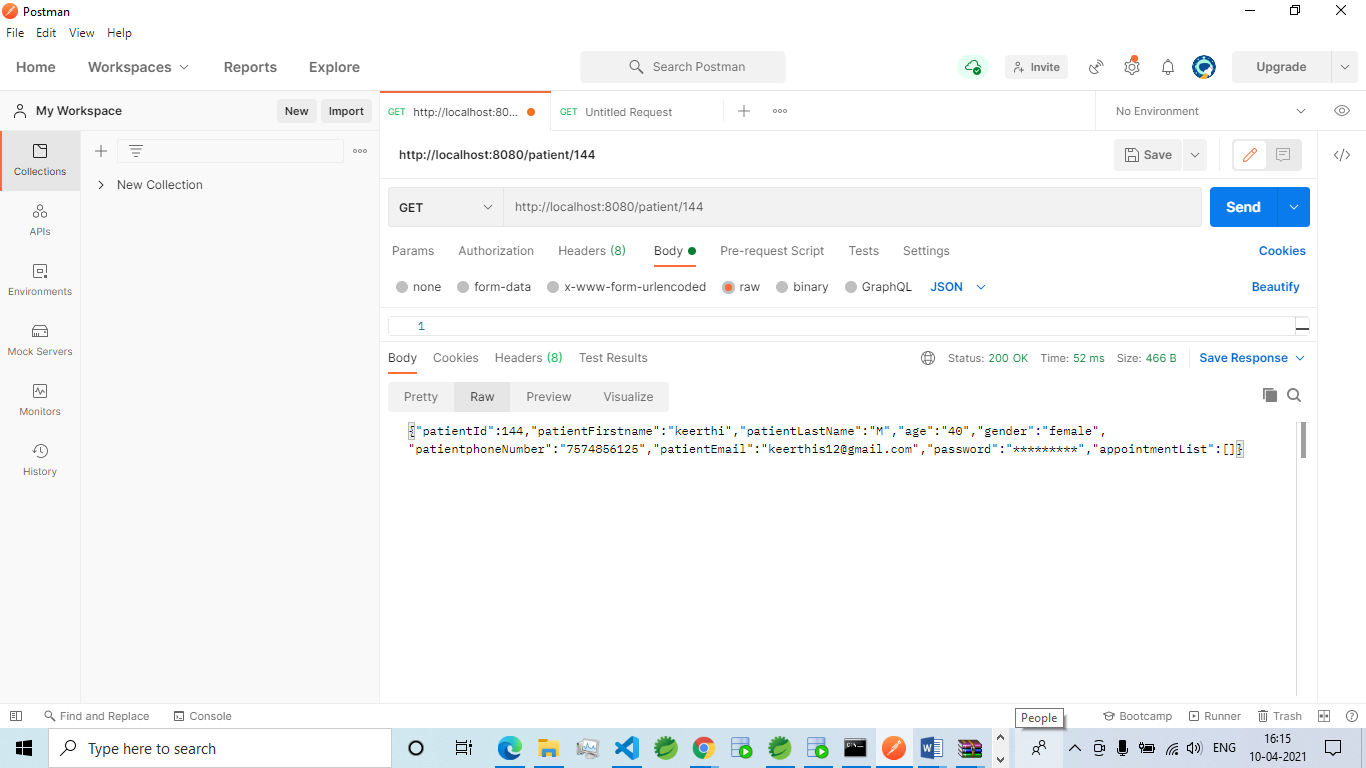
**Review of GET Method using POSTMAN**

Retrieval of all Patient objects using POST Method in POSTMAN



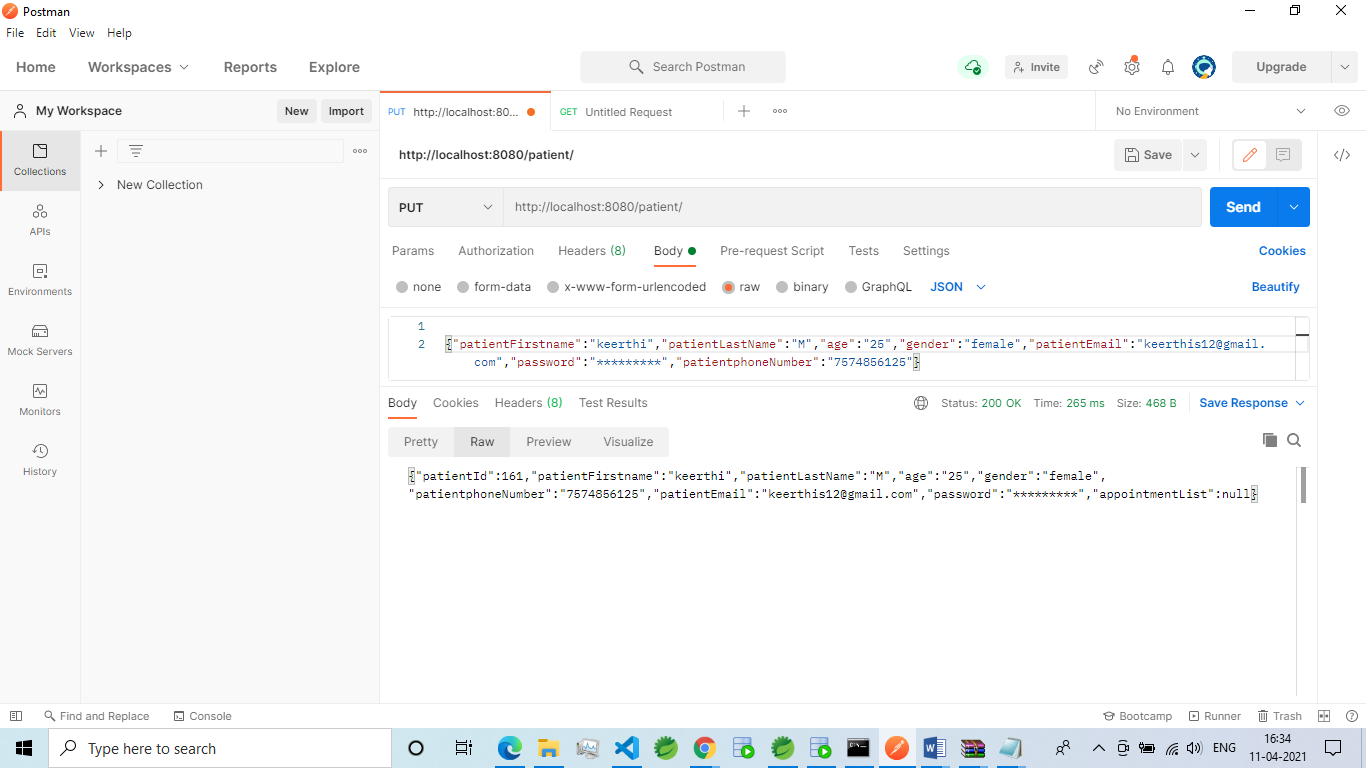
**Review of GET Method by patientId using POSTMAN**

Retrieval of a Patient object by patientId using POST Method in POSTMAN



**Review of PUT Method using POSTMAN**

Updation of a Patient object field using PUT Method in POSTMAN



1. Creating **Doctor** Entity:

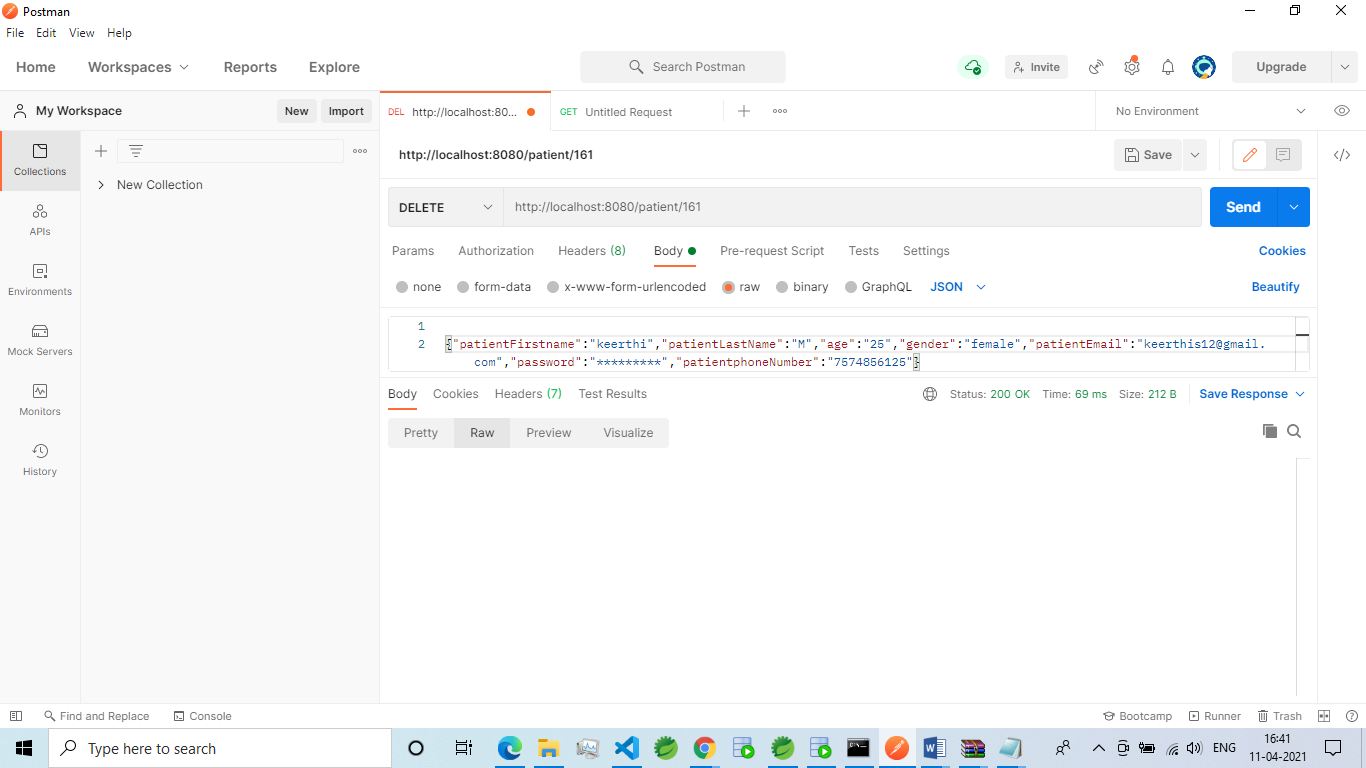
Build a RESTful resource for **Doctor** manipulations, where CRUD operations to be carried out. Here will have multiple layers into the application:

1. Create an Entity: Doctor
2. Create a DoctorRepository interface and will make use of Spring Data JPA
3. Will have findByDoctorId method
4. Add the Doctor details method
5. Will have deleteDoctorById method
6. Will have findAllDoctorsmethod
7. Create a DoctorService class and will expose all these services
8. Finally, create a DoctorController will have the following Uri’s:

|  |  |  |  |
| --- | --- | --- | --- |
| URI | METHODS | Description | Format |
| /doctor/ | GET | Get all the doctor details | JSON |
| /doctor/doctored | GET | Give a single doctor description searched based on doctored | JSON |
| /doctor/ | POST | Add the doctor details | JSON |
| /doctor/ | PUT | Update doctor details | JSON |
| /doctor/doctored | DELETE | Delete a Doctor based on doctored | JSON |

**Review of DELETE Method by Id using POSTMAN**

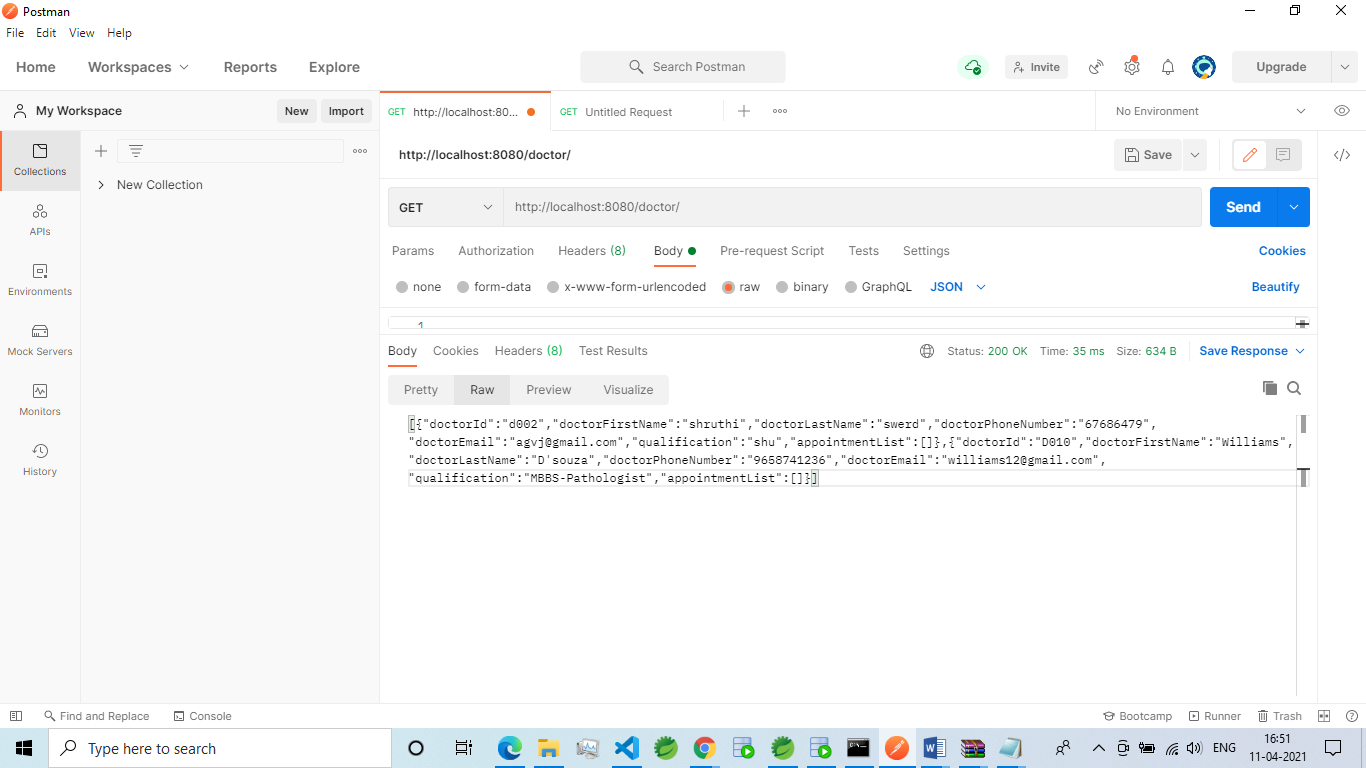
Deletion of a Doctor object by Id using DELETE Method in POSTMAN



DOCTOR:

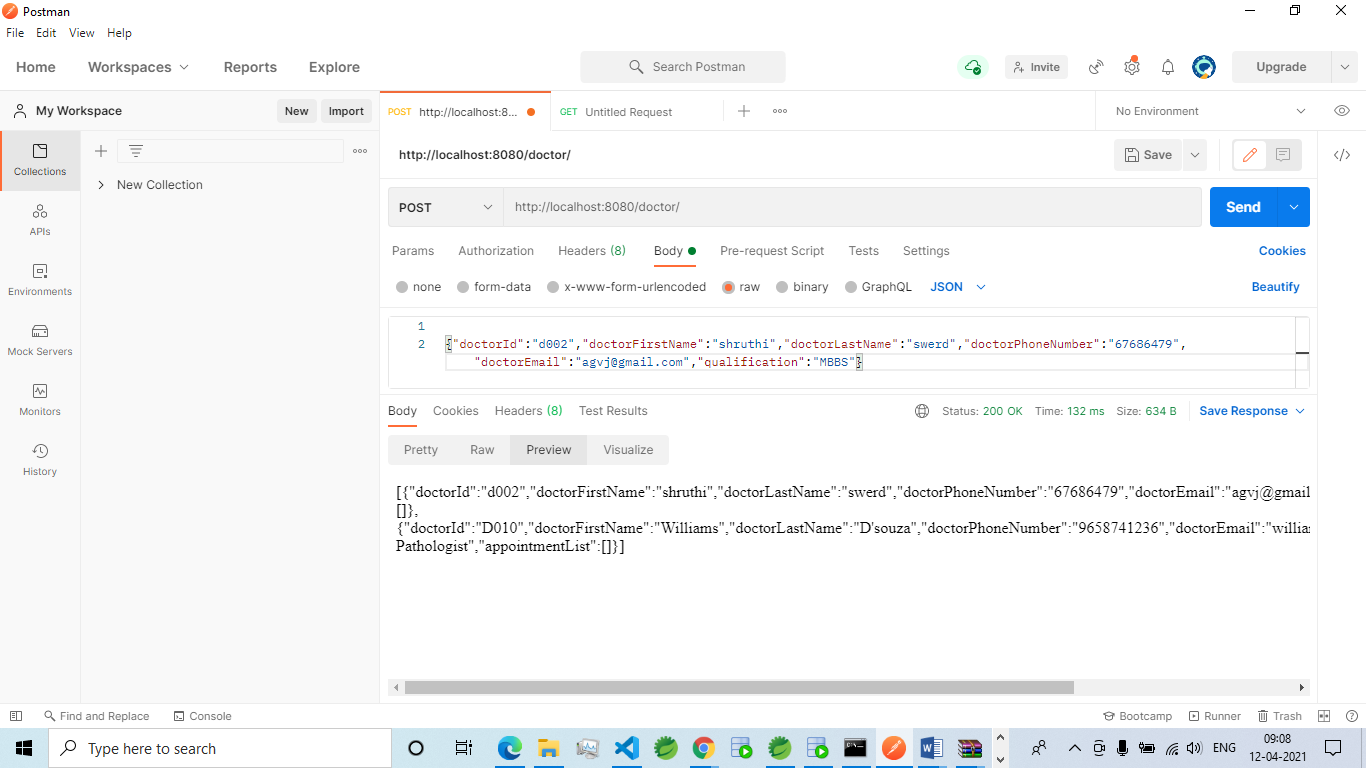
**Review of GET Method using POSTMAN**

Retrieval of a Doctor object using GET Method in POSTMAN



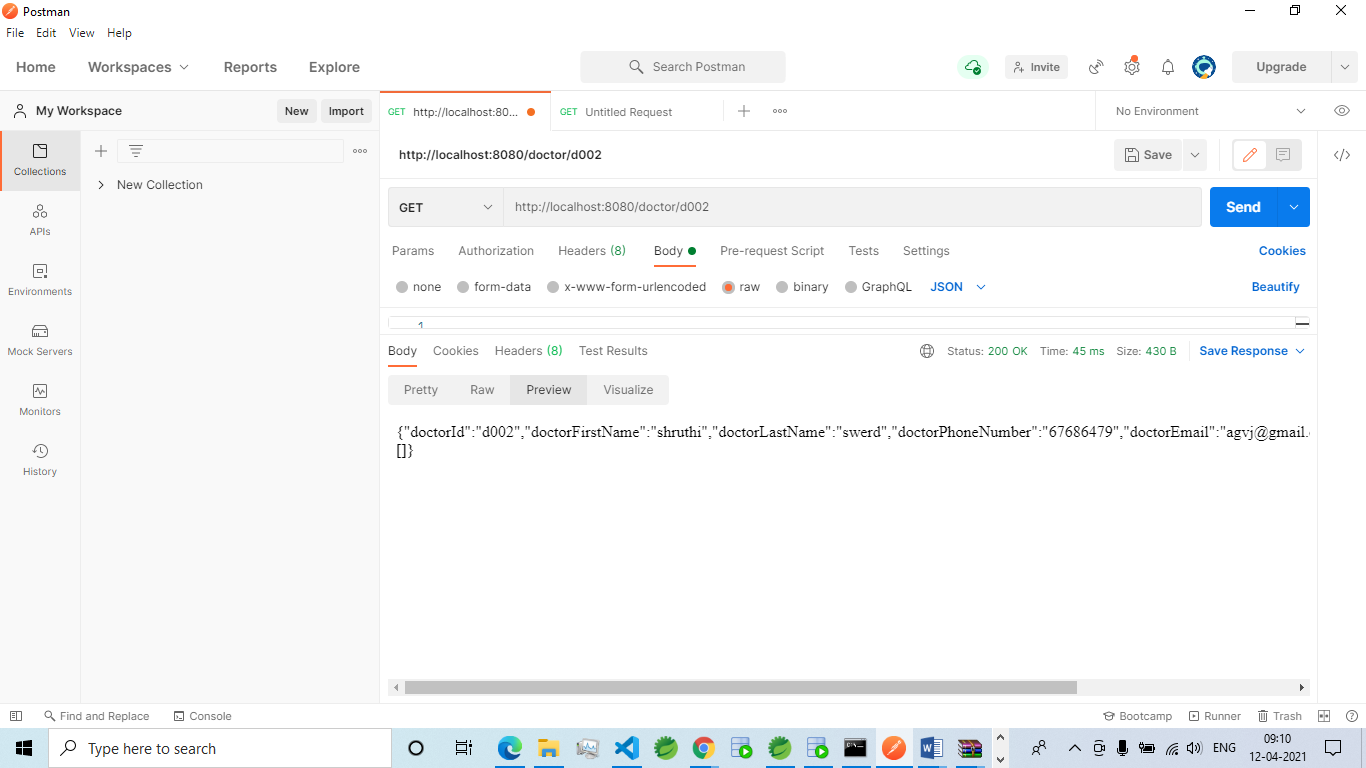
**Review of POST Method using POSTMAN**

Adding of Doctor object using POST Method in POSTMAN



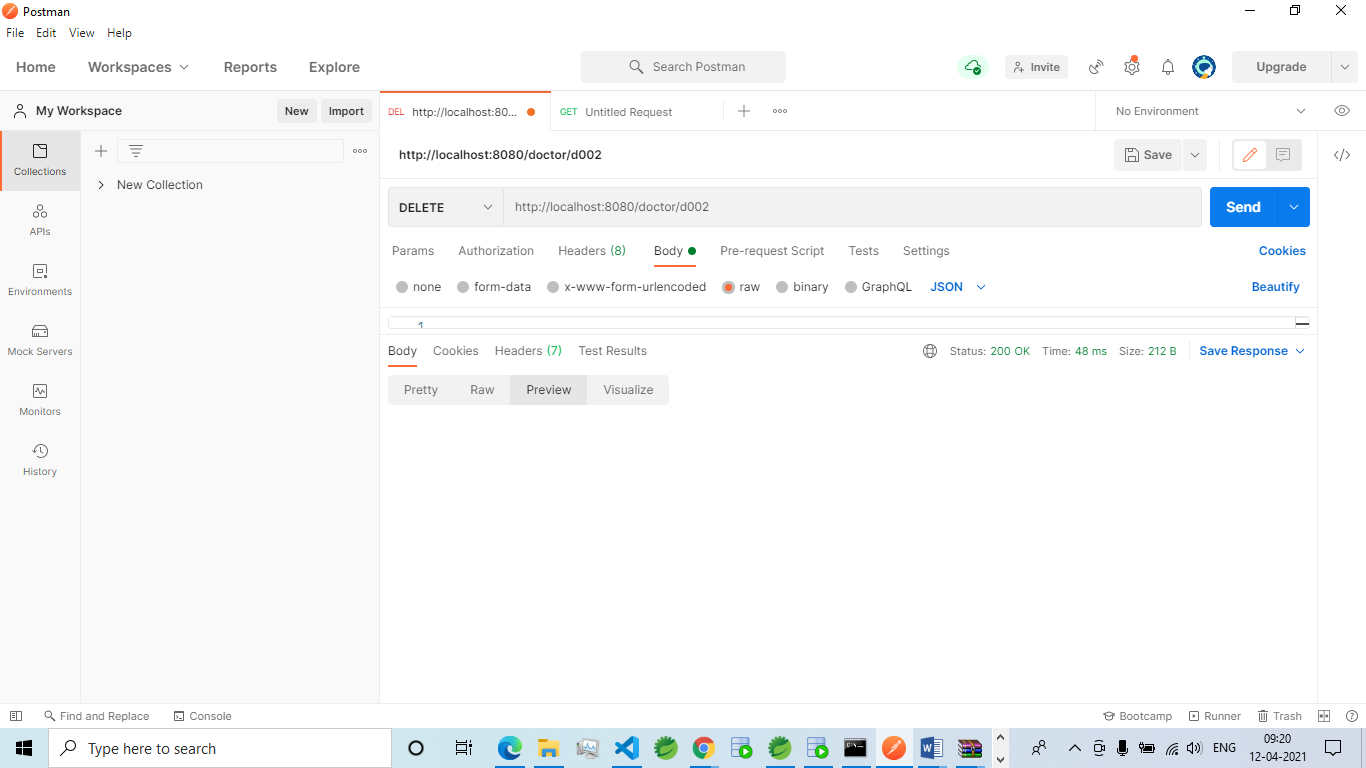
**Review of GET Method by Id using POSTMAN**

Retrieval of a Doctor object by Id using POST Method in POSTMAN



**Review of DELETE Method by Id using POSTMAN**

Deletion of a Doctor object by Id using DELETE Method in POSTMAN



1. Creating **Technician** Entity:

Build a RESTful resource for **Technician** manipulations, where following operations to be carried out. Here will have multiple layers into the application:

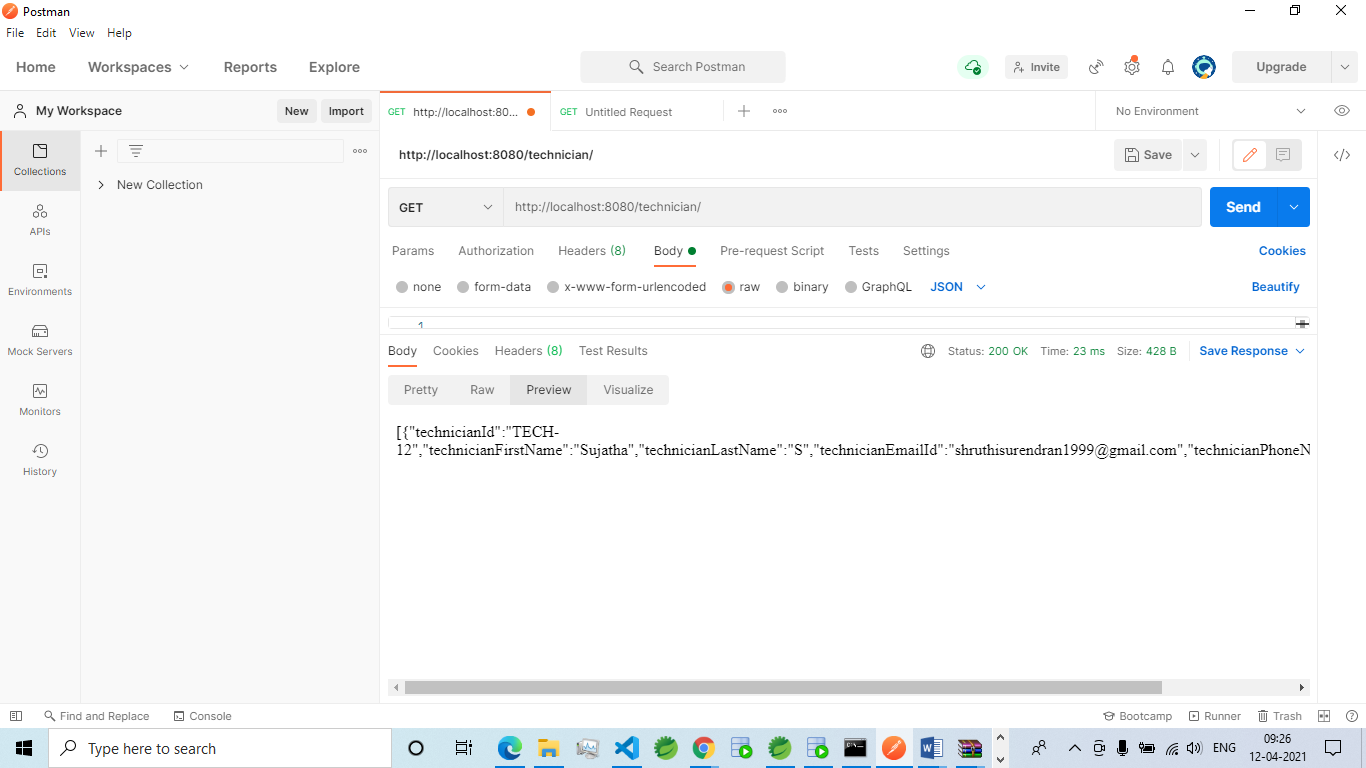
1. Create an Entity: Technician
2. Create a TechnicianRepository interface and will make use of Spring Data JPA
   1. Add the Technician details
   2. Will have deleteTechnicianById method to remove with specific technician Id.
   3. Will have updateTechnicianDetails
3. Create a TechnicianService class and will expose all these services
4. Finally, create a TechnicianController will have the following Uri’s:

|  |  |  |  |
| --- | --- | --- | --- |
| URI | METHODS | Description | Format |
| /technician/ | GET | Add the technician details with updated status. | JSON |
| /technician/technicianId | GET | Give a single technician description searched based on technicianId | JSON |
| /technician/ | POST | Add technician details | JSON |
| /technician/ | PUT | Update all technicians details | JSON |
| /technician/technicianId | DELETE | Delete a technician based on technicianId | JSON |

TECHNICIAN:

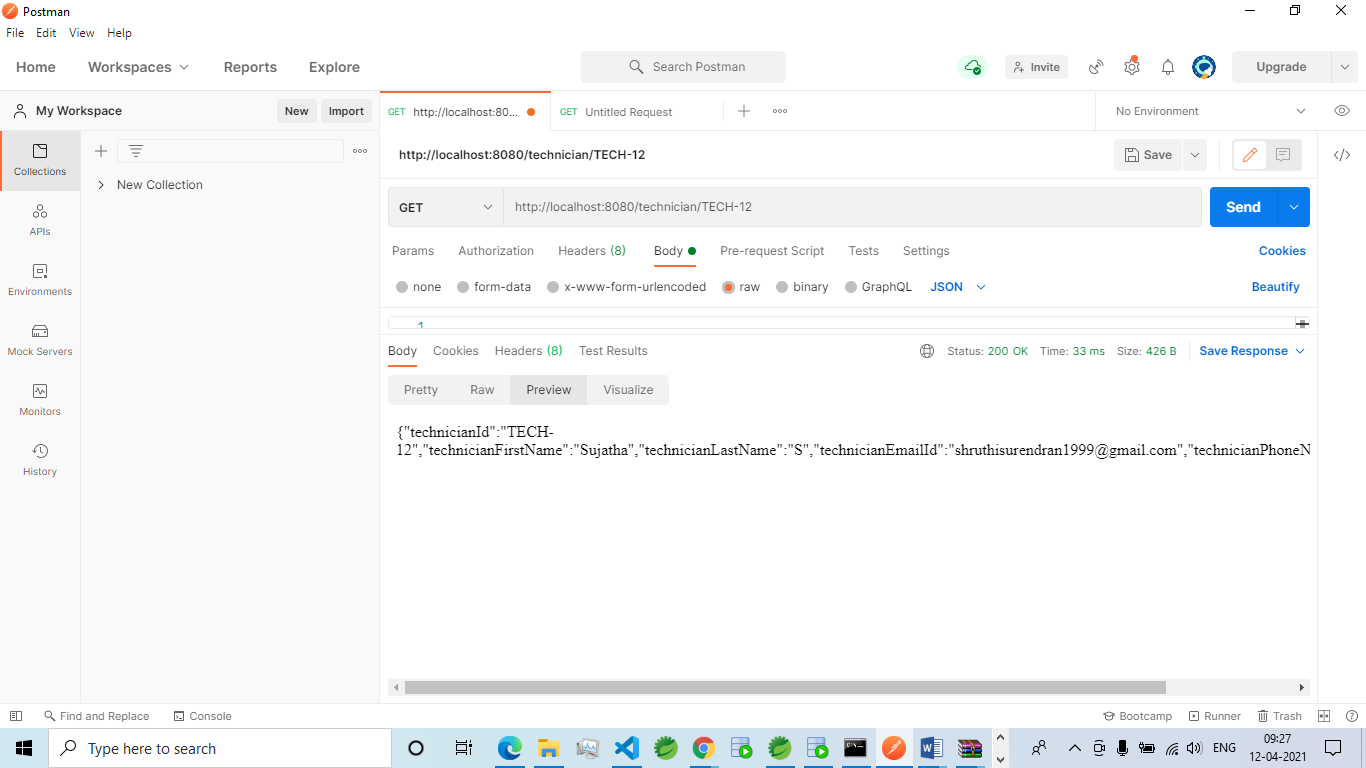
**Review of GET Method using POSTMAN**

Retrieval of a Technician object using GET Method in POSTMAN



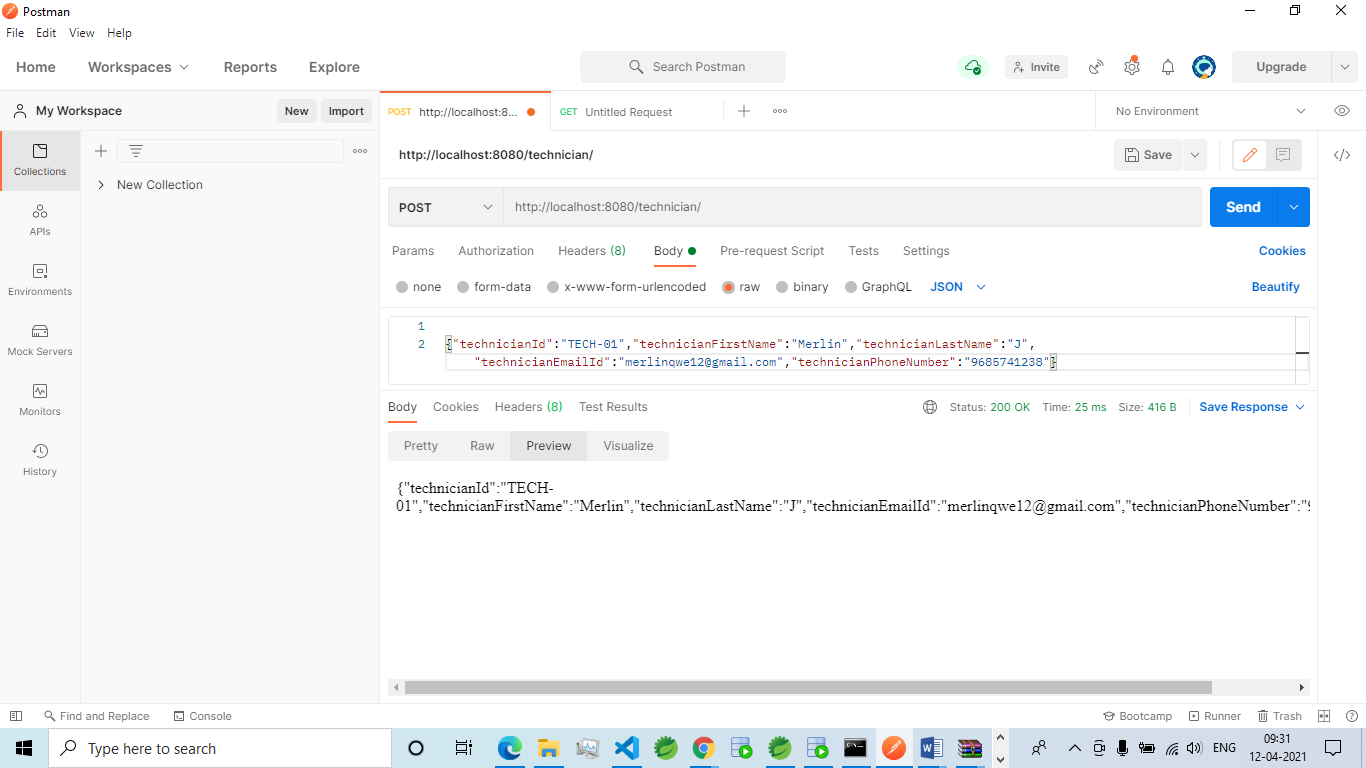
**Review of GET Method by Id using POSTMAN**

Retrieval of a Technician object using GET By Id Method in POSTMAN



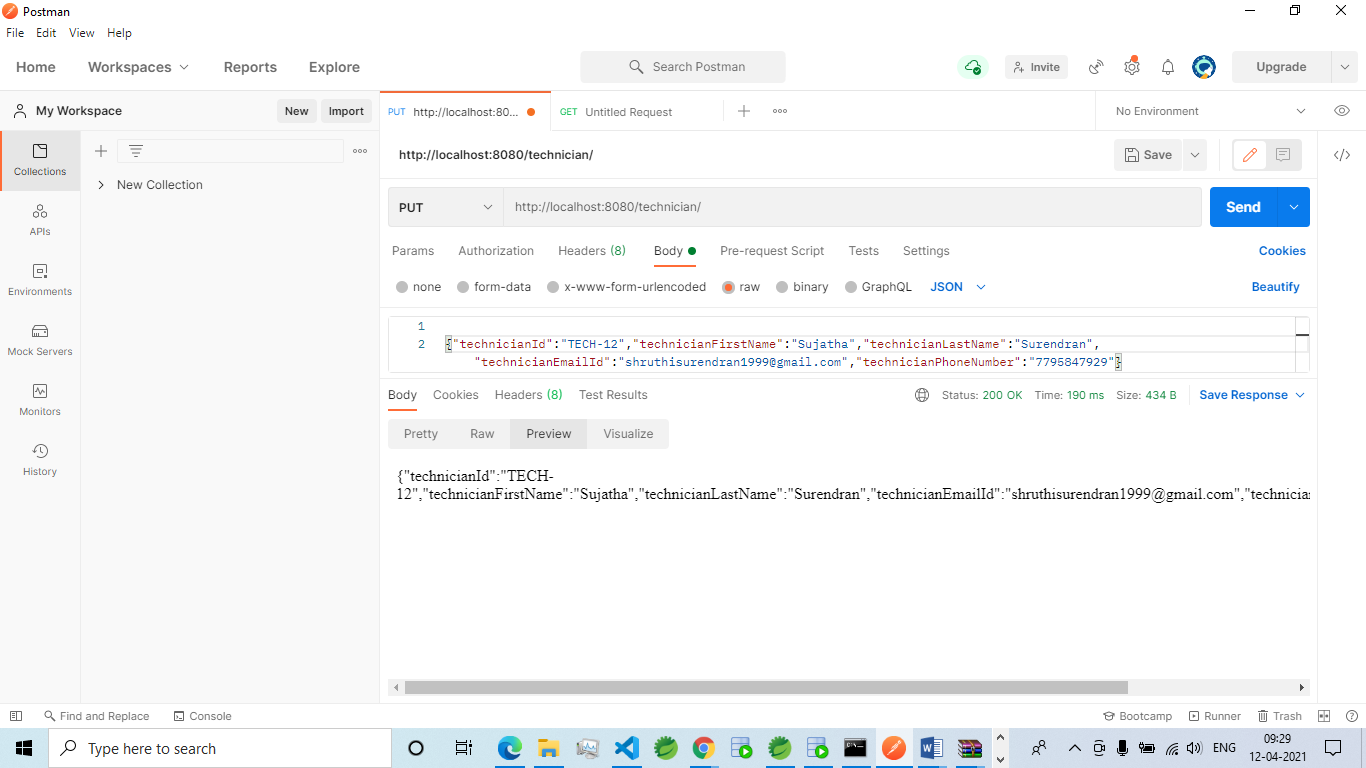
**Review of POST Method using POSTMAN**

Creation of a Technician object using POST Method in POSTMAN



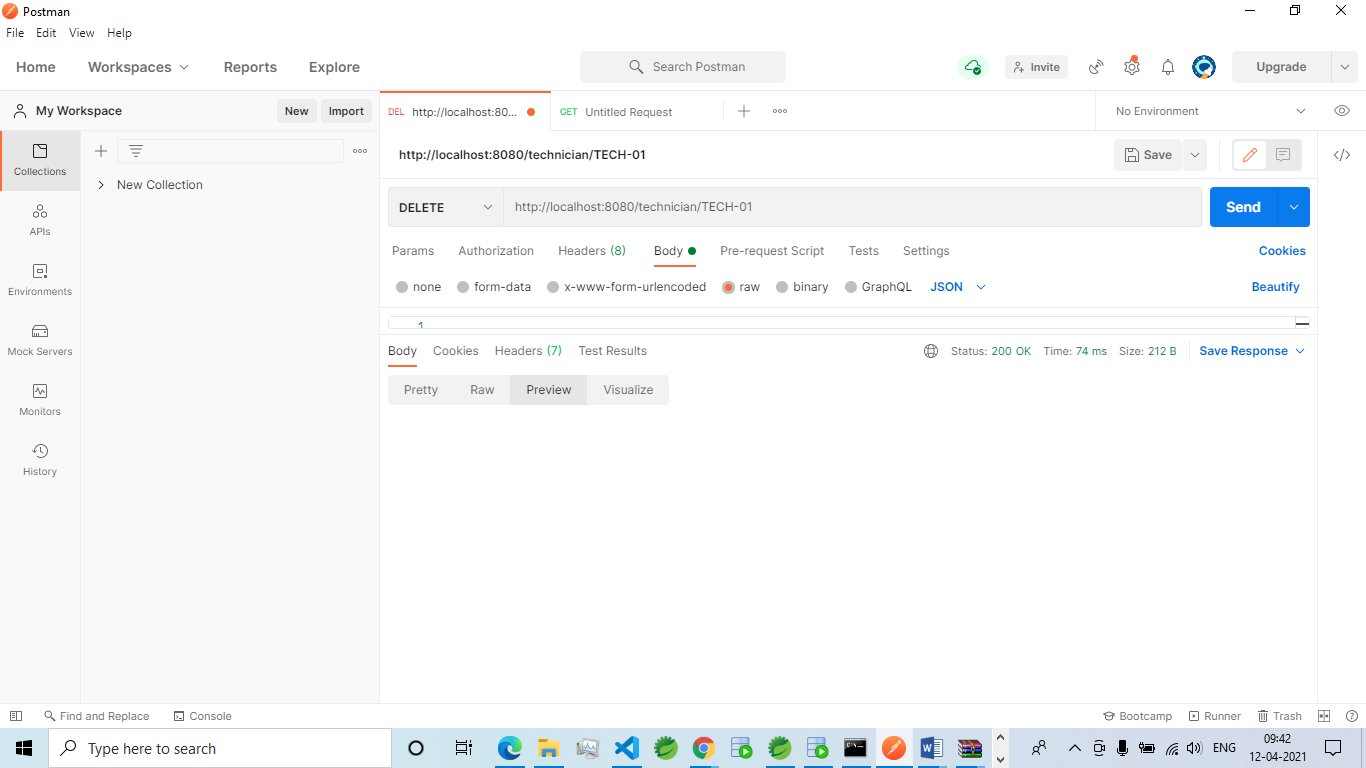
**Review of PUT Method using POSTMAN**

Updation of a Technician object using PUT Method in POSTMAN



**Review of DELETE Method using POSTMAN**

Deletion of a Technician object using DELETE Method in POSTMAN



1. Creating **MedicalTest**  Entity:

Build a RESTful resource for **MedicalTest** manipulations, where following operations to be carried out. Here will have multiple layers into the application:

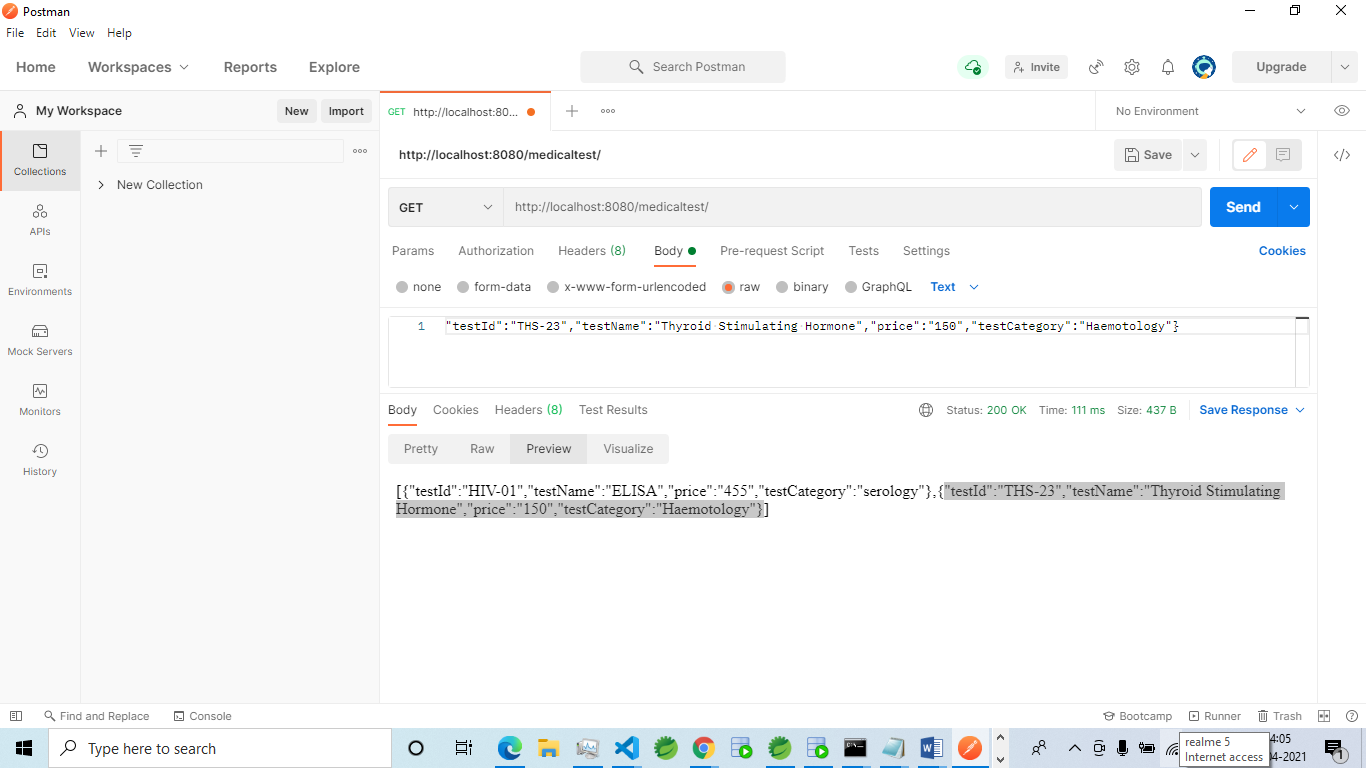
1. Create an Entity: MedicalTest
2. Create a MedicalTestRepository interface and will make use of Spring Data JPA
   1. Add the MedicalTest details
3. Create a MedicalTestService class and will expose all these services
4. Finally, create a MedicalTestController will have the following Uri’s:

|  |  |  |  |
| --- | --- | --- | --- |
| URI | METHODS | Description | Format |
| /medicalTest/ | GET | Add the medicalTest details with updated status. | JSON |
| /medicalTest/medicalTestId | GET | Give a single medicalTest description searched based on medicalTestId | JSON |
| /medicalTest/ | POST | Add medicalTest details | JSON |
| /medicalTest/ | PUT | Update all medicalTest details | JSON |
| /medicalTest/medicalTestId | DELETE | Delete a medicalTest based on medicalTestId | JSON |

MEDICALTEST:

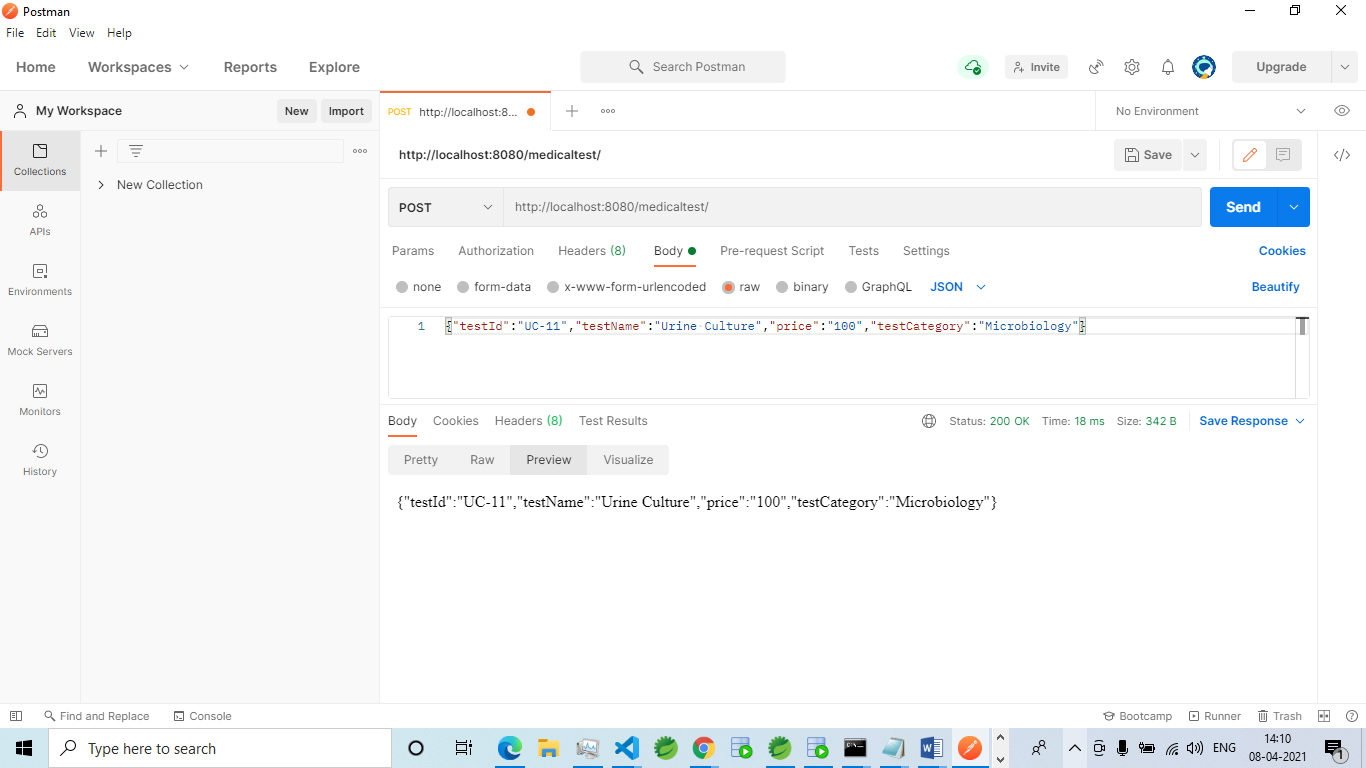
**Review of GET Method using POSTMAN**

Retrieval of all MedicalTest objects is done using GET Method in POSTMAN



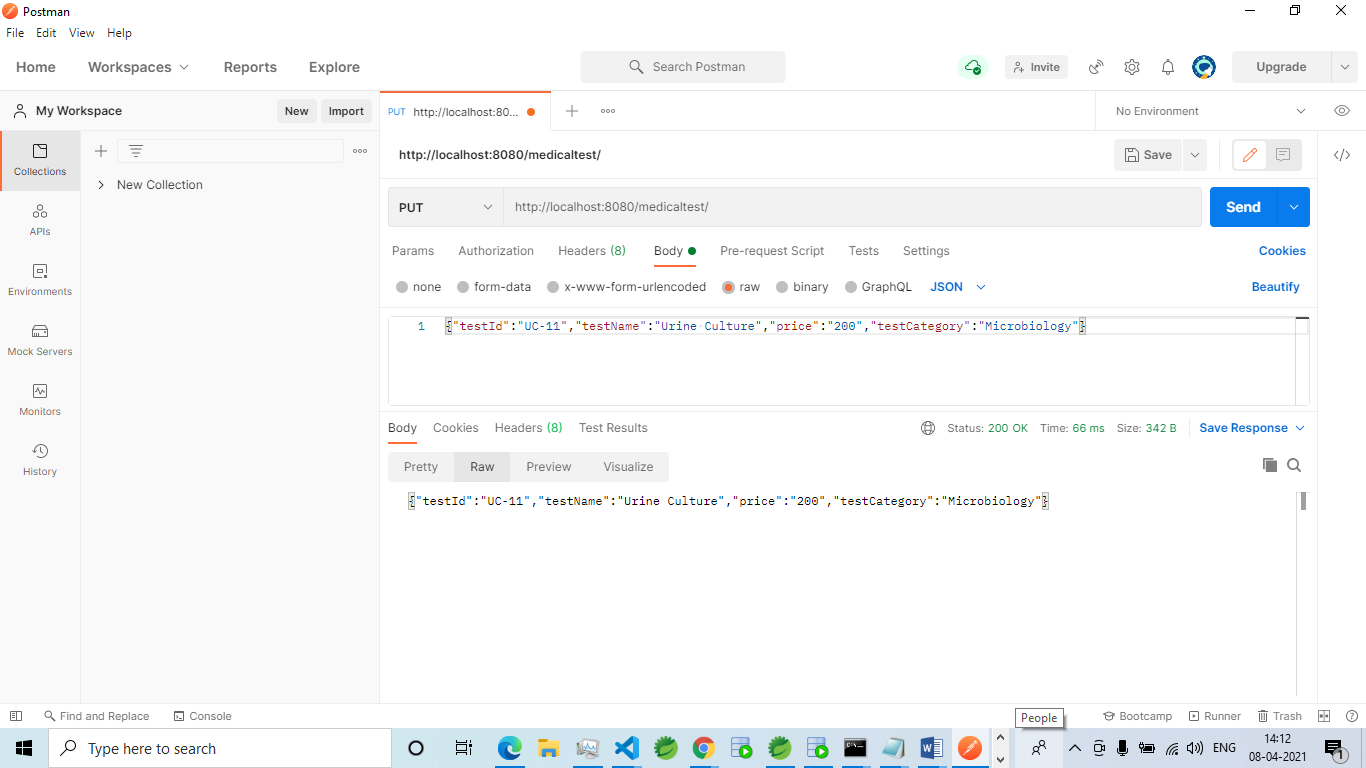
**Review of POST Method using POSTMAN**

Creation of a new object MedicalTest class is done using POST Method in POSTMAN



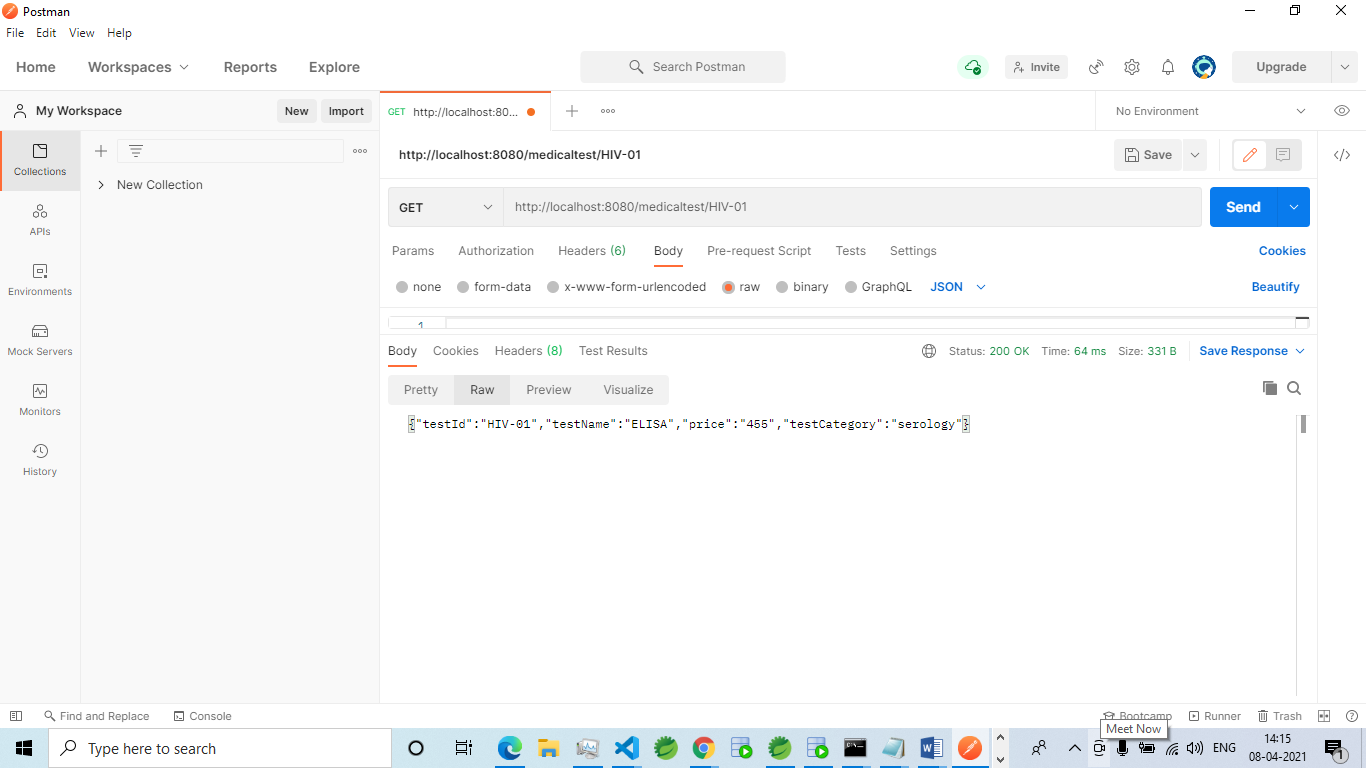
**Review of PUT Method using POSTMAN**

Updation of an object of MedicalTest class is done using PUT Method in POSTMAN



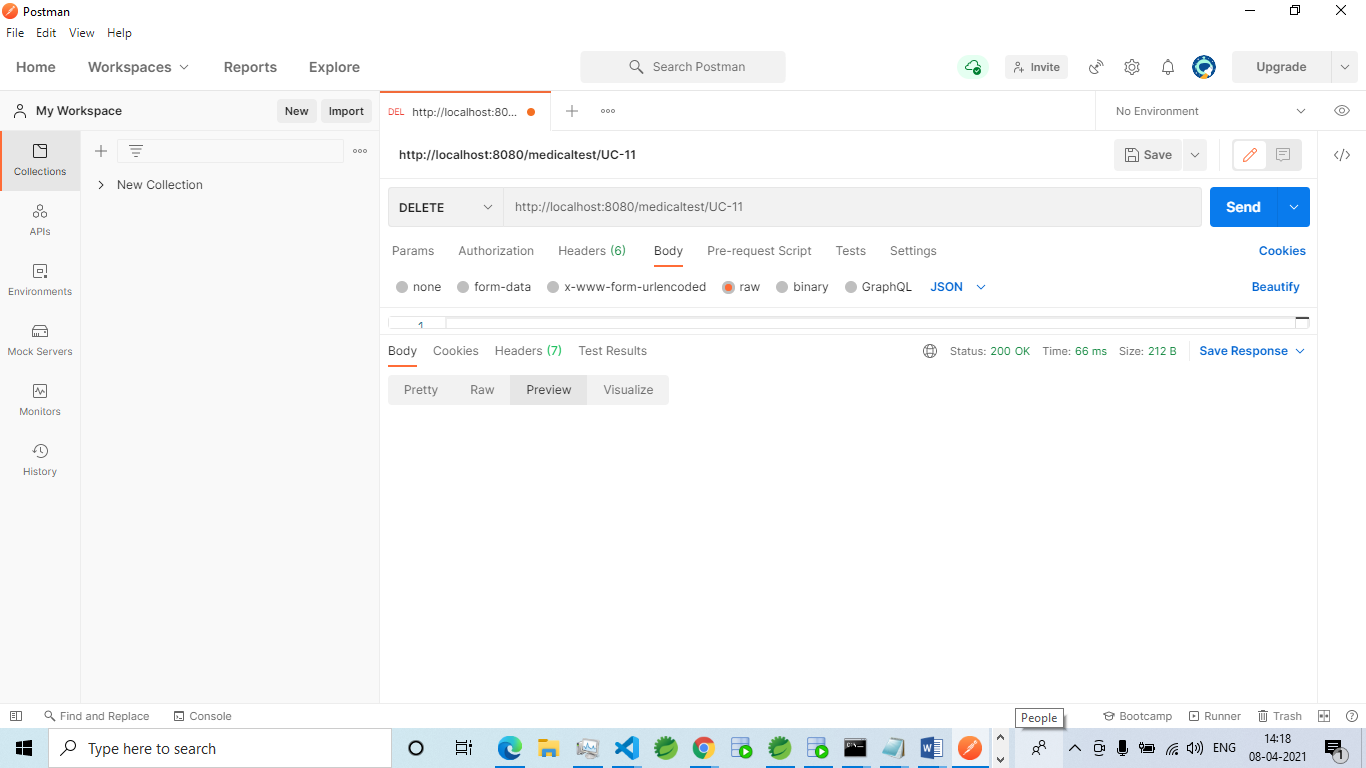
**Review of GET Method by testID using POSTMAN**

Retrieving an object of MedicalTest class by Id is done using GET Method by testID in POSTMAN



**Review of DELETE Method using POSTMAN**

Deleting of an object of MedicalTest class is done using DELETE Method by testID in POSTMAN



1. Creating **Appointment** Entity:

Build a RESTful resource for **Appointment**  manipulations, where following operations to be carried out. Here will have multiple layers into the application:

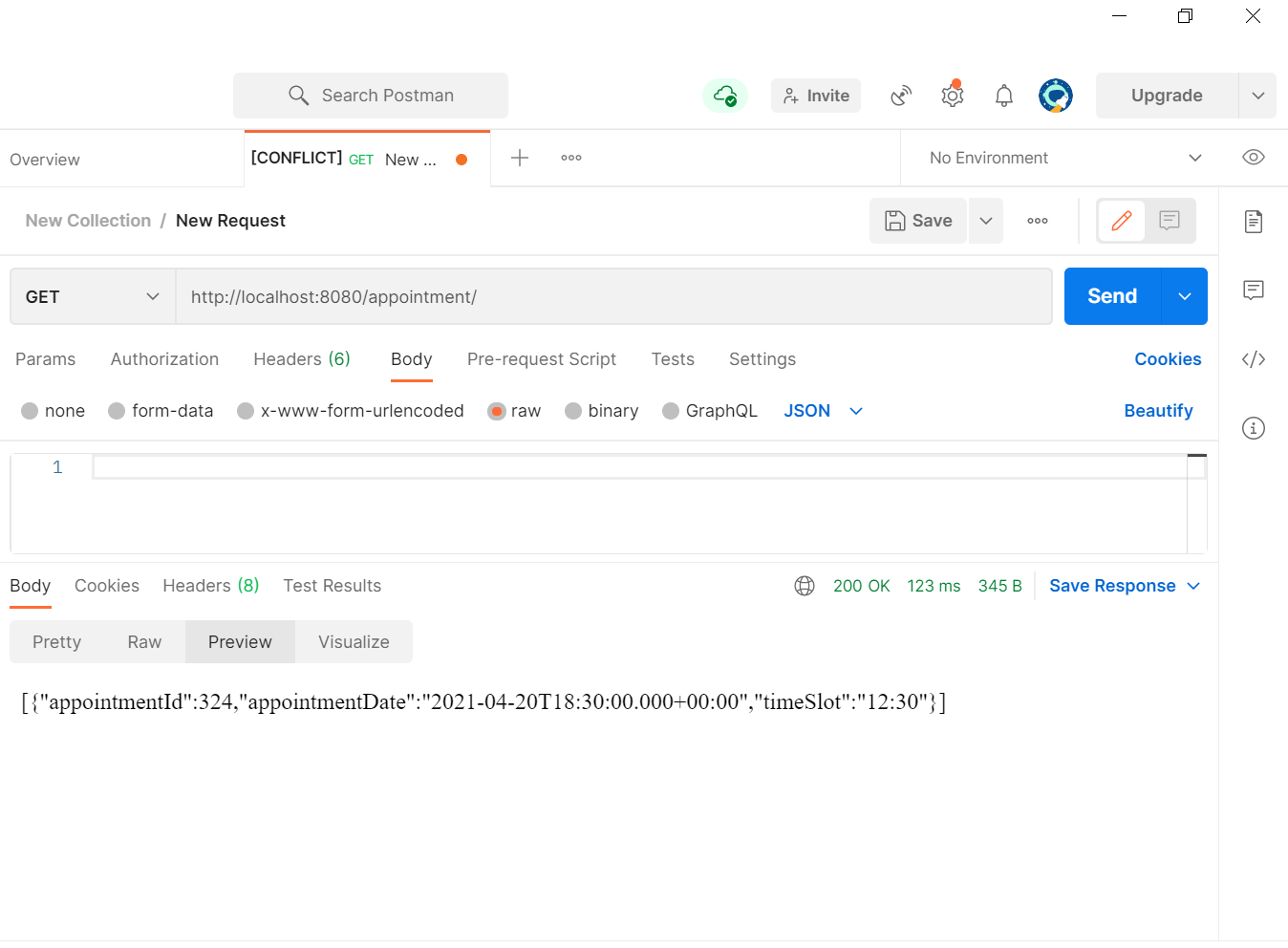
1. Create an Entity: Appointment
2. Create a AppointmentRepository interface and will make use of Spring Data JPA
   1. Add the Appointment details
   2. Will have deleteAppointmentById method to remove with specific technician Id.
   3. Will have updateAppointmentDetails
3. Create a AppointmentService class and will expose all these services
4. Finally, create a AppointmentController will have the following Uri’s:

|  |  |  |  |
| --- | --- | --- | --- |
| URI | METHODS | Description | Format |
| /appointment/ | GET | Add the appointment details with updated status. | JSON |
| /appointment/appointmentId | GET | Give a single appontment description searched based on appointmentId | JSON |
| /appointment/ | POST | Add appointment details | JSON |
| /appointment/ | PUT | Update all appointment details | JSON |
| /appointment/appointmentId | DELETE | Delete a appointment based on appointmentId | JSON |

APPOINTMENT

**Review of GET Method using POSTMAN**

Retrieval of all Appointment objects is done using GET Method in POSTMAN

****

1. Creating **TestReport** Entity:

Build a RESTful resource for **TestReport**  manipulations, where following operations to be carried out. Here will have multiple layers into the application:

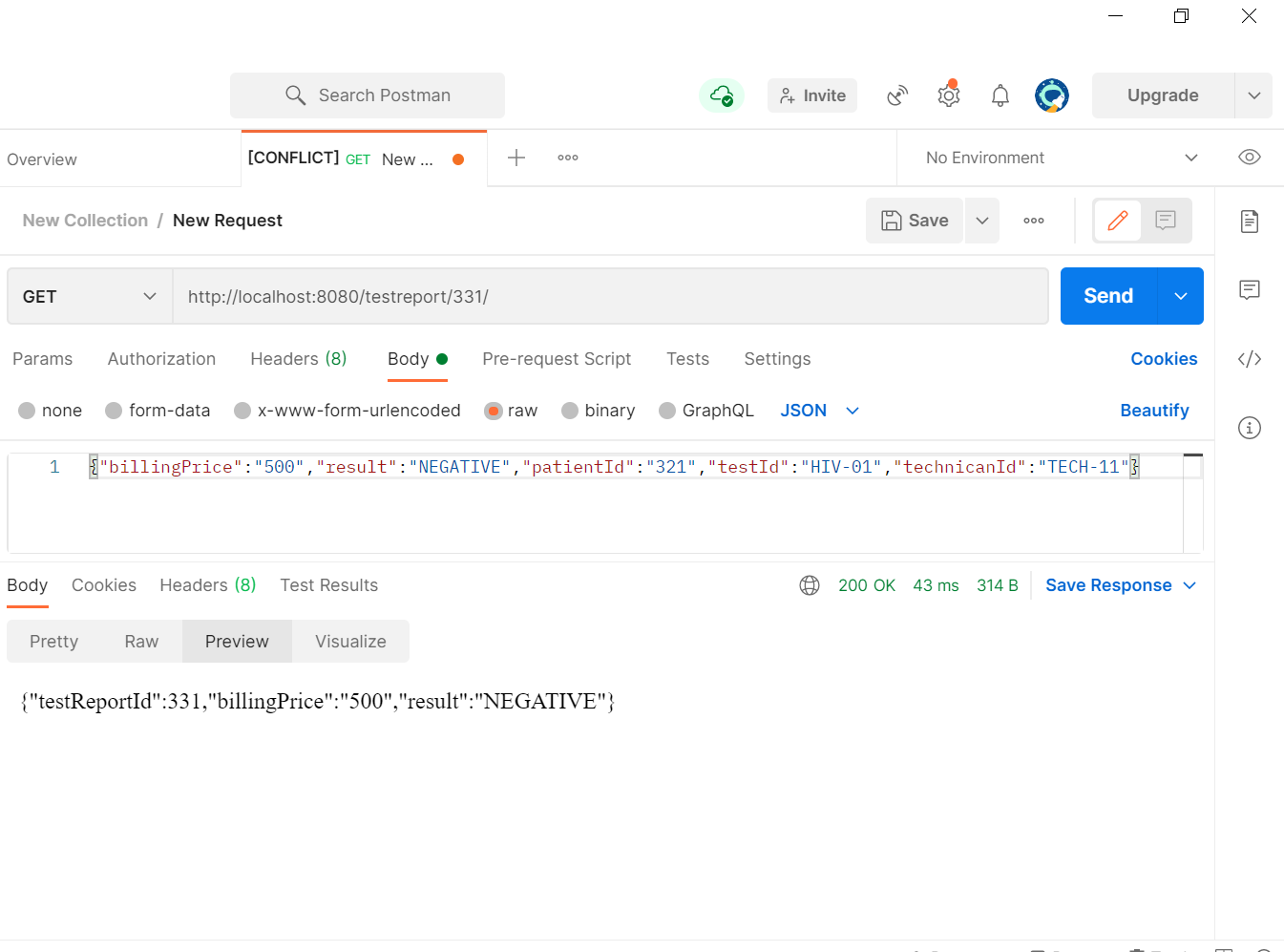
1. Create an Entity: TestReport
2. Create a TestReportRepository interface and will make use of Spring Data JPA
   1. Add the TestReport details
   2. Will have deleteTestReportById method to remove with specific technician Id.
   3. Will have updateTestReportDetails
3. Create a TestReportService class and will expose all these services
4. Finally, create a TestReportController will have the following Uri’s:

|  |  |  |  |
| --- | --- | --- | --- |
| URI | METHODS | Description | Format |
| /testreport/ | GET | Add the testreport details with updated status. | JSON |
| /testreport/testreportId | GET | Give a single testreport description searched based on testreportId | JSON |
| /testreport/ | POST | Add testreport details | JSON |
| /testreport/ | PUT | Update all testreport details | JSON |
| /testreport/testreportId | DELETE | Delete a testreport based on testreportId | JSON |

TESTREPORT:

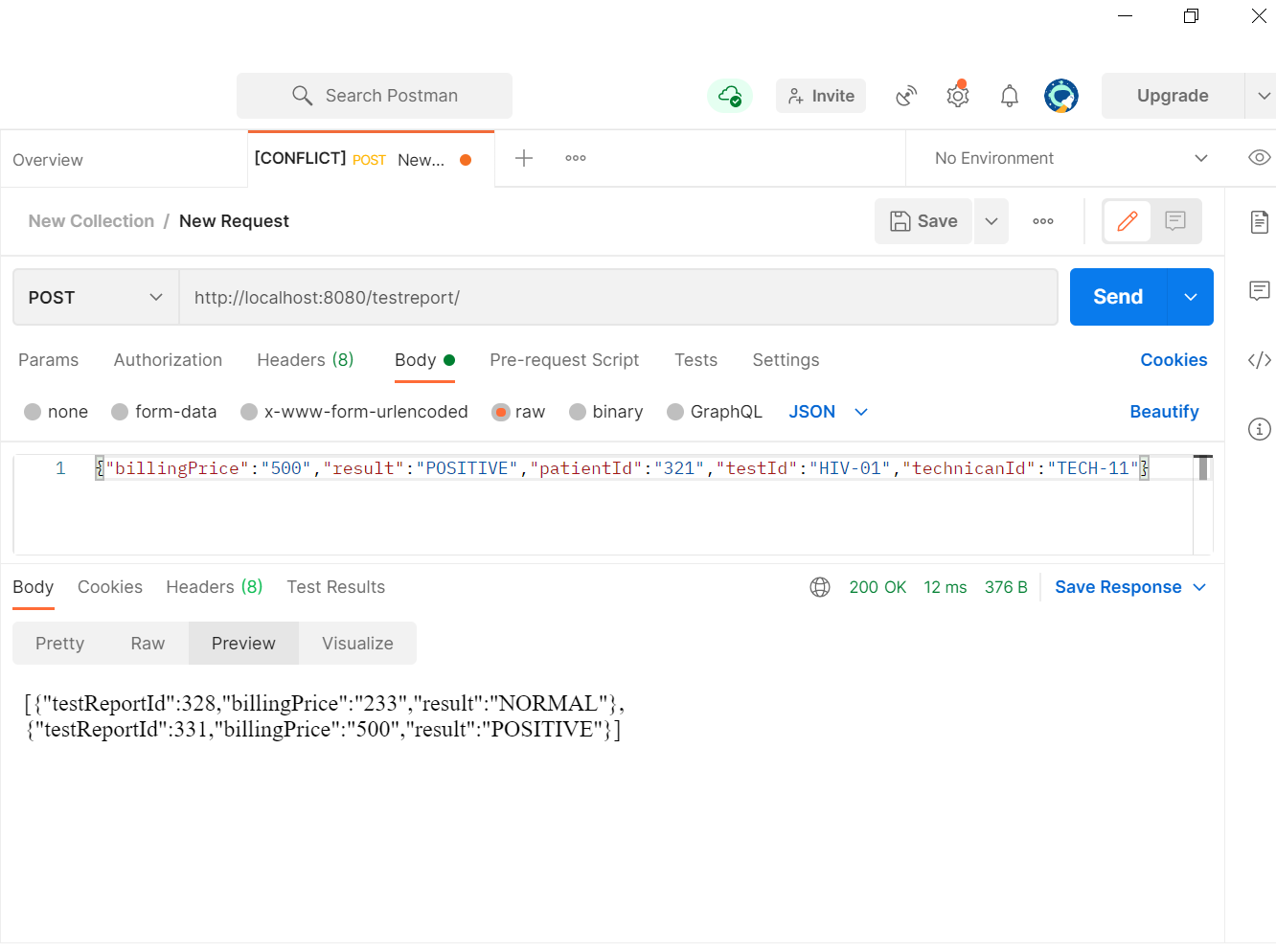
**Review of GET Method using POSTMAN**

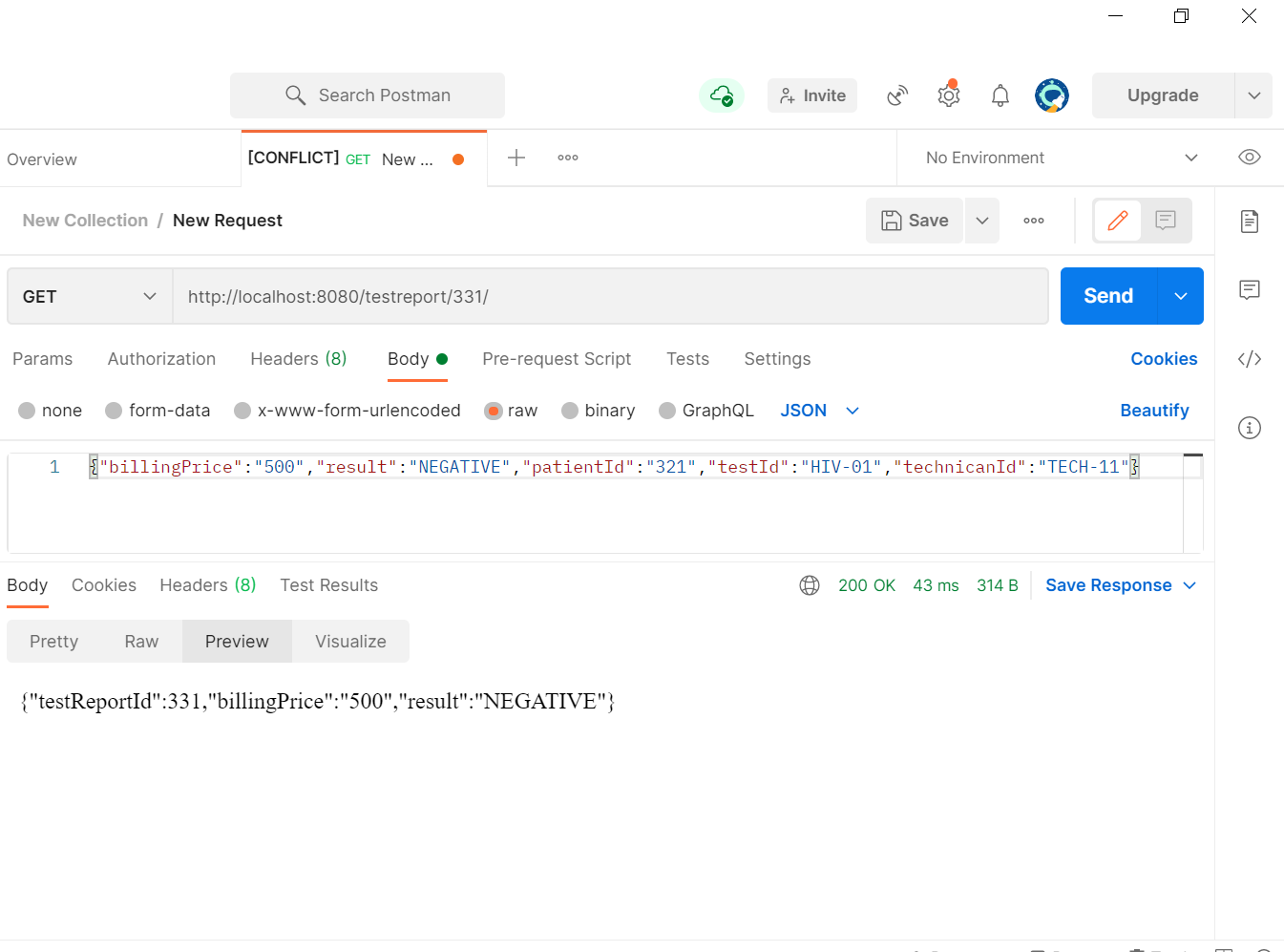
Retrieval of testreport objects is done using GET Method in POSTMAN



**Review of POST Method using POSTMAN**

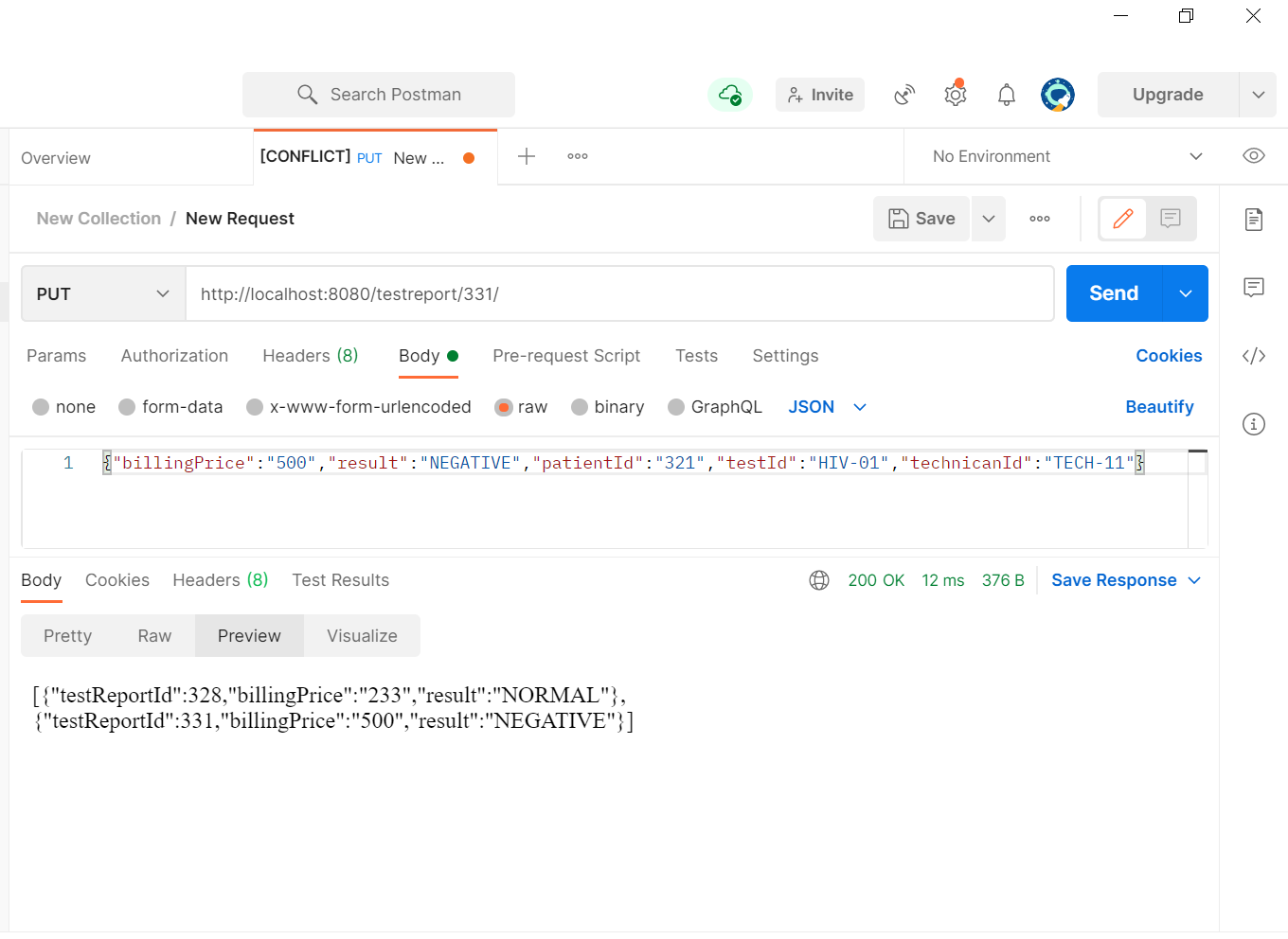
Creation of a new object testreport class is done using POST Method in POSTMAN





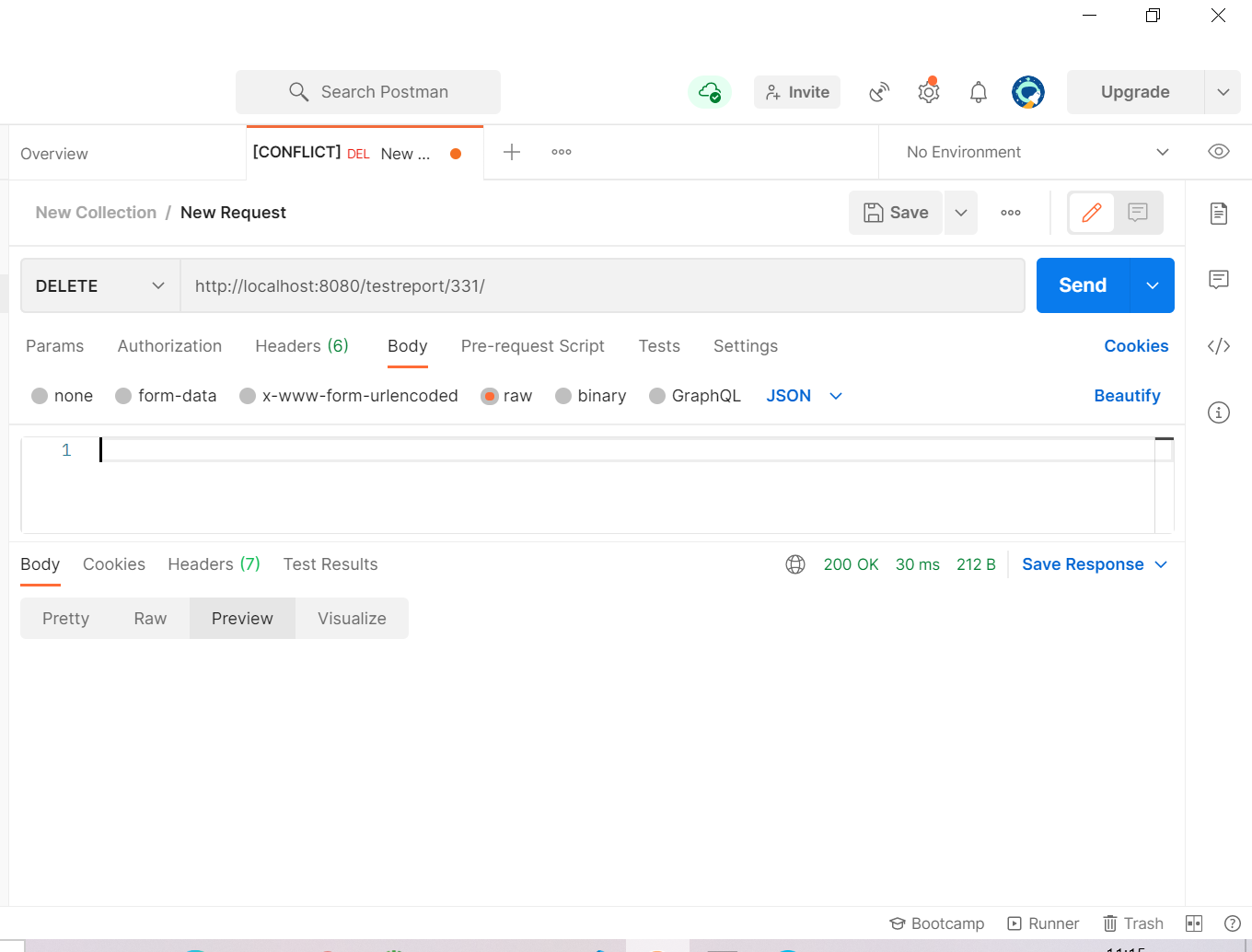
**Review of PUT Method using POSTMAN**

Updation of an object of testreport class is done using PUT Method in POSTMAN



**Review of DELETE Method using POSTMAN**

Deleting of an object of testreport class is done using DELETE Method by testID in POSTMAN



**FUNCTIONAL SPECIFICATION**

1. **Assumptions**

* User Interface: The type of client interface (front-end) to be supported - Angular based
* The administrator can add and remove tests into the database on a weekly basis.
* You must not allow patient to add same test twice.
* When you add test into cart the No. of tests selected will be incremented.
* If you remove the test from the cart, the counter will be decremented.
* The clear will remove all the tests so that the No. of tests will be zero
* The total amount will be calculated based on the test, accordingly, change the test counter & total amount.

1. **General Expectations**

* Participants must create the **Class Diagram, Sequence Diagram and ER Diagram**.
* Participants must do **Unit testing and Functional Testing using POSTMAN tool**
* Integration of Angular and Spring Boot with Microservices should be done, referring**Project2 -Frond End Development Project.**
  + The server should be a concurrent server servicing multiple client.
  + Database can be implemented using Oracle 11g or above.
  + To begin with, the application should support at least 1 admin and 2 customers.
  + Compilation and Build should be done using Eclipse IDE or STS
  + Source-code and all documents must be maintained (checked-in) in configuration management system (subversion)
  + Coding standards (for Java) should be followed

NOTE:

1. **Validation of user Data**

* Spring MVC using JSR-303 annotations
* AJAX validation without forcing the page to reload (Wherever applicable)
* JavaScript validation (if necessary)

1. **UI Design – (for Web Application) Use DIV/CSS or Semantic Elements to control the style and layout**
2. **Create at least one SQL DML-statement inside PL/SQL blocks**



**8 Acceptance Criteria**

All P1 requirements must be mandatorily implemented

**9 Traceability to Requirements**

Appropriate requirements from RS and FS are mapped here.



**Document Reference ID & Description: (Doc ID from which this document is derived)**



|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Reference document: RS** | **Current document: FS** |
|  | **Requirement/Feature (Section ID/Name)** | **Location (Section ID/Name)** |

1.



2.



**10 Acronyms and Glossary**

Acronym and glossary for this document mentioned in the below table.

|  |  |
| --- | --- |
| **Abbreviation** | **Remark** |
| MDL | Medical Diagnostic Laboratory |
| RS | Requirement Specification |
| FS | Functional Specification |

1. Validations should be performed at all levels of application appropriately.