Spring boot notes

**STS( Spring Tool Suite)**

LabTest : JDBC project

convert into spring framework

ORM : object relational Mapping

| | |

OOPS RDBMS java---MySQL:connection

|

MySQL

|

sql

---------------------------------------

spring framework - application framework - web application

loosly couple

tightly couple

12:22 06-10-2024

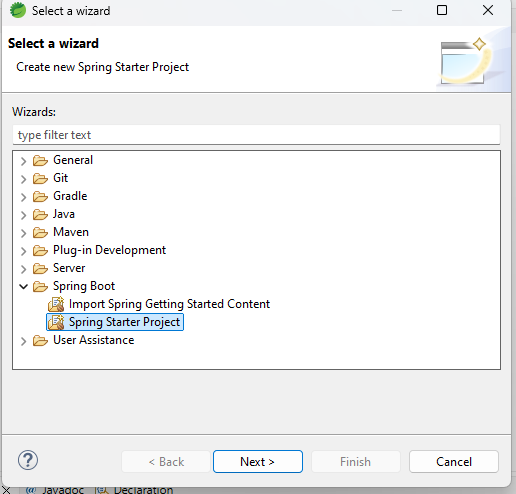
https://spring.io/tools

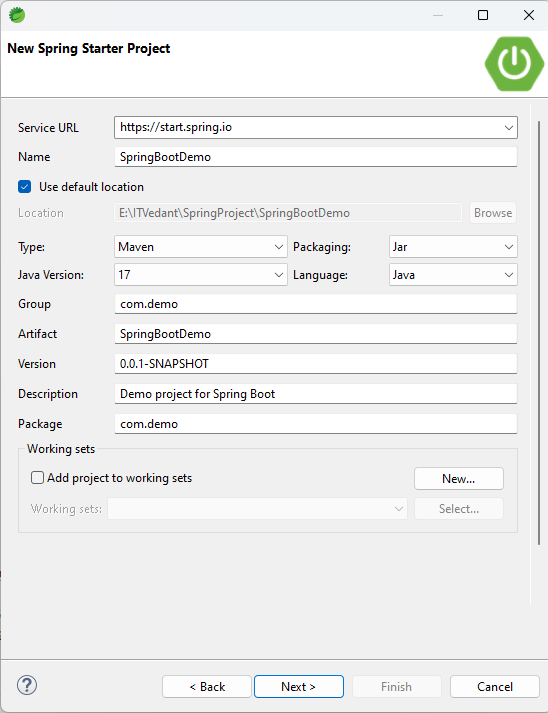
download - 4.25.0 windows x86\_64

extract file inside E drive ITVedant folder - open stringToolSuite4

pre request – we need

STS, MySQL workbench & Lombok

steps1 : spring start project

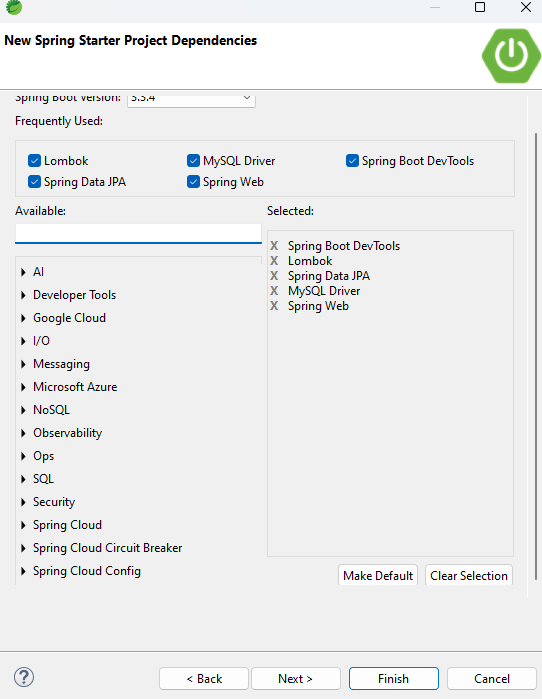


visible inside Pom.xml

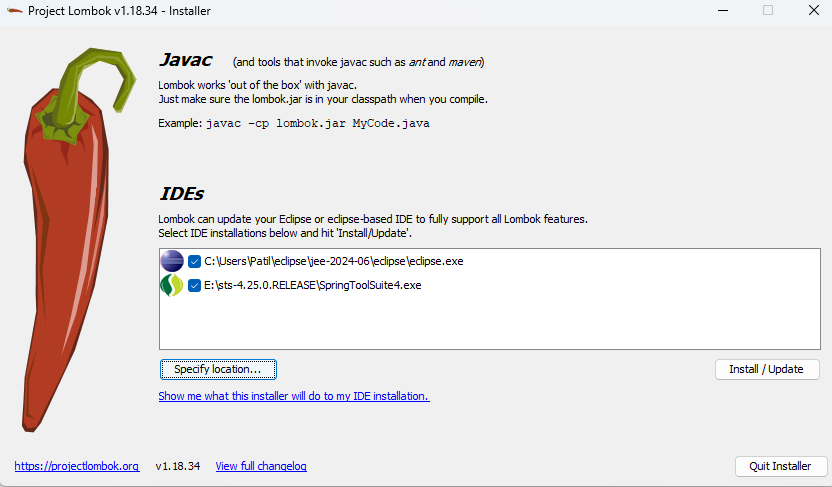
step2 add dependency web, mysql(for db), jpa(for generating table),

Lombok: we use Lombok - to reduce code - **loosely couple** – do with getter setter or contructors - to skip the multiple getter setter

devtools : act as live server(to run project automatic)



<https://projectlombok.org/download> - download this file – to reduce code



Click on install update – install successfull

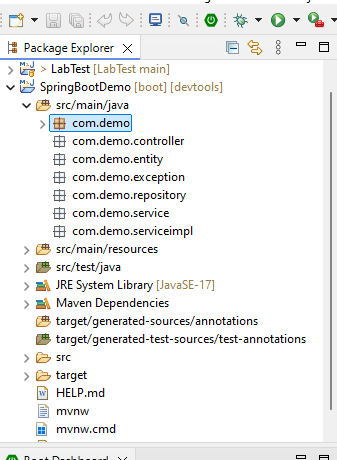
step3 : create DB(database)

workbench

CREATE DATABASE Anitavlab;

use Anitavlab;

step 4 create structure in STS



step 5

Entity : patient , appointment , lab , doctor , payment , report

Table will generate automatically we only have to write structure of the table

3 classes – Appointment, LabTest, Patient

**package** com.demo.entity;

**import** java.sql.Date;

**import** jakarta.persistence.Column;

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.GeneratedValue;

**import** jakarta.persistence.GenerationType;

**import** jakarta.persistence.Id;

**import** jakarta.persistence.Table;

**import** lombok.Getter;

**import** lombok.Setter;

//@ means annotation

@Entity

@Table(name="PatientDetails") // optional (name of table in database)

@Getter //generate all get methods

@Setter //generate all set methods

**public** **class** Patient {

@Id //to generate and create a primary key

@GeneratedValue(strategy = GenerationType.***IDENTITY*** ) // for auto increment

**private** **int** patientId; //primary key

//@Column : used for properties - unique , null , or what else we what for that purpose we use this

@Column(length = 25, nullable = **false**) // by default true

**private** String firstName;

@Column(length = 35)

**private** String lastName;

@Column(length = 10, nullable = **false**)

**private** Date dateofbirth;

@Column(length = 11, nullable = **false**, unique=**true**)

**private** String contactNumber;

@Column(length = 20, nullable = **false**, unique=**true**)

**private** String emailID;

@Column(length = 50, nullable = **false**)

**private** String address;

@Column(length = 5)

**private** String boodGroup ;

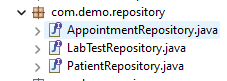
**private** **int** weight;

**private** **int** height;

}

Steps 6

Inside –



**Repository** – help to do all action line CRUD operation – give all logic for CRUD operation. Ex – save(), findAll(), getById(), **for all this method we extends JpaRepository**

delete from table where id=1; instead of this query we write only - deleteAll() method

create 3 interface – AppointmentRepository, LabTestRepository , PatientRepository

Example

**package** com.demo.repository;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** com.demo.entity.Patient;

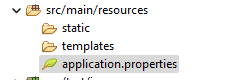
//extends JpaRepository for inbuilt method

**public** **interface** PatientRepository <Patient ,Integer >{

}

Step 7

For database connection here we – application.properties file



spring.application.name=SpringBootDemo

spring.jpa.hibernate.ddl-auto=update

spring.datasource.url=jdbc:mysql://localhost:3306/Anitavlab //database name

spring.datasource.username=root

spring.datasource.password=anita@123

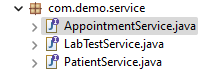
spring.datasource.driver-class-name =com.mysql.cj.jdbc.Driver

server.error.include-message=always

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect

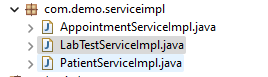
step 8

create interfaces for entities (declare CRUD() ) – AppointmentService , LabTestService, PatientService



step 9

create classes for serviceimp interfaces( logic of CRUD() ) – AppointmentServiceImpl , LabTestServiceImpl, PatientServiceImpl



Step 10

Add custom queries in respective repositories If needed

E:\ITVedant\_spring\SpringProject\SpringBootDemo\src\main\java\com\demo\repository

For ex :

//Custom query - to fetch patient detail based on contact number

Patient findByPhone(String contactNumber);

09-11-24

Step 11 :

JPQL: Java persistant query language - used for writing custom query – like groupby, inbetween

Mysql – “select \* from PatientDetails where contactnumber = ?”

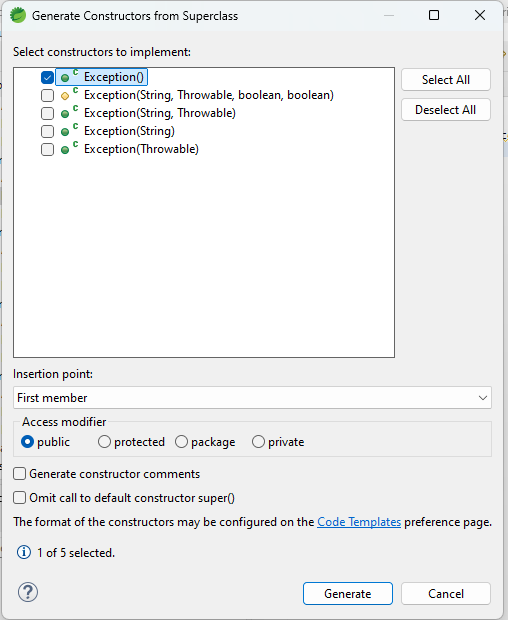
For java we use custom query becz java don’t understand mysql language

JPQL - Custom query – “select pd from PatientDetails pd where pd.contactnumber = ?”

Step 12 :

Create 3 exception :

LabTestNotFoundException, AppointmentNotFoundException, PatientNotFoundException



Step13 : pillers

Entity

Repository : custom query

Service – logic handle

Exception : handle error exception

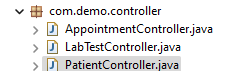
Controller : handle incoming request and share response based on request

Get : fetch data from application (status code :200)

Post – update, insert data into application (status code :201)

Delete – used to remove data from application (status code :204)

Put : used to modify existing data from application (status code :200)



10-11-24

Step 14

Exception : if entered data is correct , which is already entered

Validation : if user enter correct data or not

Step15 add validation

Add pom.xml dependency

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

We added this annotation in patient and labtest to ensure

NotBlank : used for string

NotNull : used for number

Step 16 write code controller

**public** **class** PatientController {

@Autowired //Marks a constructor, field, setter method

PatientService ps;

//@valid - to check validation while data insertion

//@RequestBody : body to http request

//post mapping:insert patient details with httpstatus created and httpstatuscode 201

@PostMapping("/registerPatient")

**public** ResponseEntity<Patient> registerPatient(@Valid @RequestBody Patient patient){

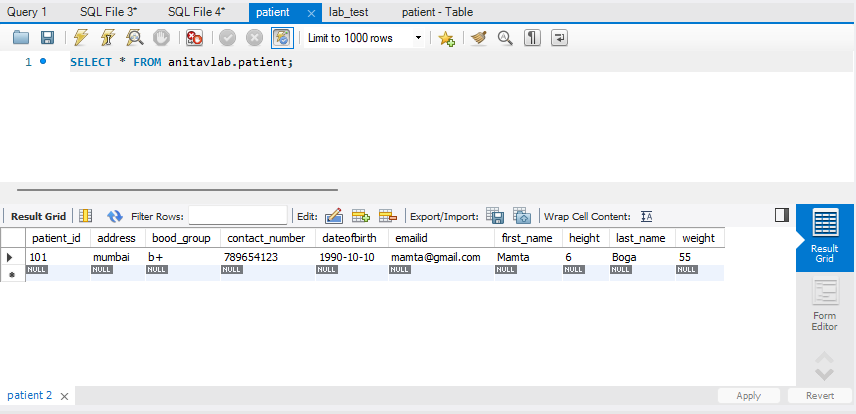
**return** **new** ResponseEntity<>(ps.registerPatient(patient),HttpStatusCode.*valueOf*(201)); //201 is post status code

}

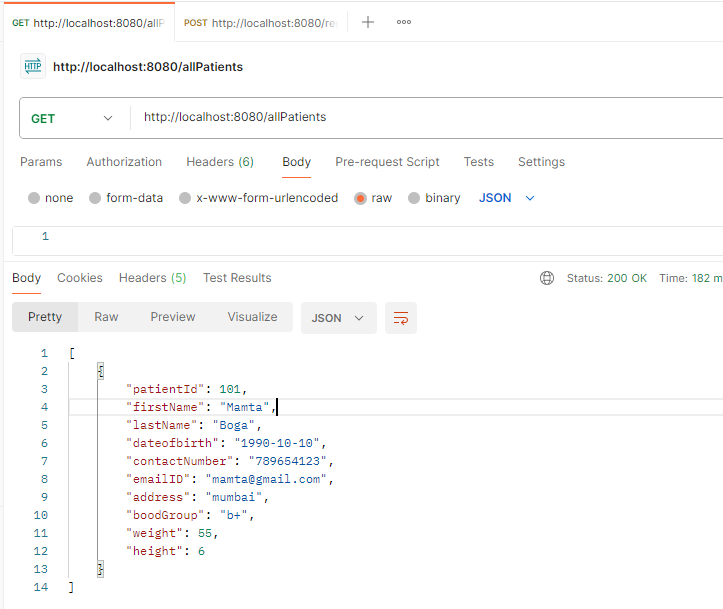
}

16-11-24

Enter data In mysql workbench first to check format



download postman - install



http://localhost:8080/allPatients

allPatients - from patientcontroller

fetch all patient data

it only except data in the form of json

 {

        "patientId": 102,

        "firstName": "Anita",

        "lastName": "Patil",

        "dateofbirth": "2001-10-05",

        "contactNumber": "908213404",

        "emailID": "anita@gmail.com",

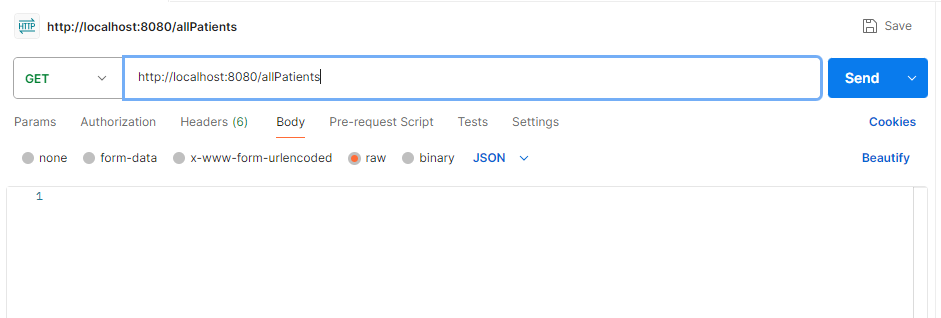
        "address": "nerul",

        "boodGroup": "o+",

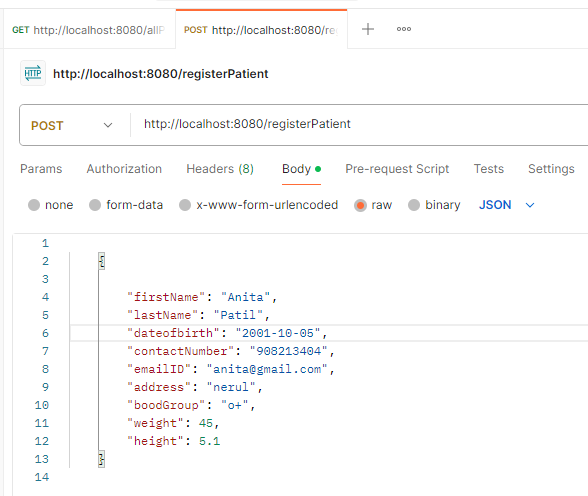
        "weight": 45,

        "height": 5

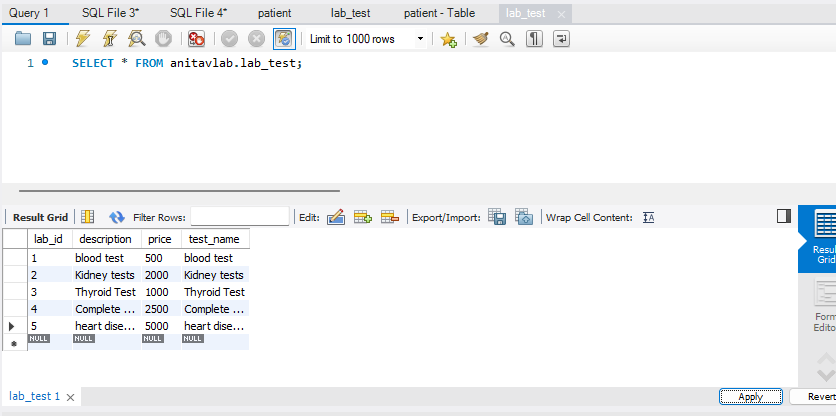
    },

enter data in database manually

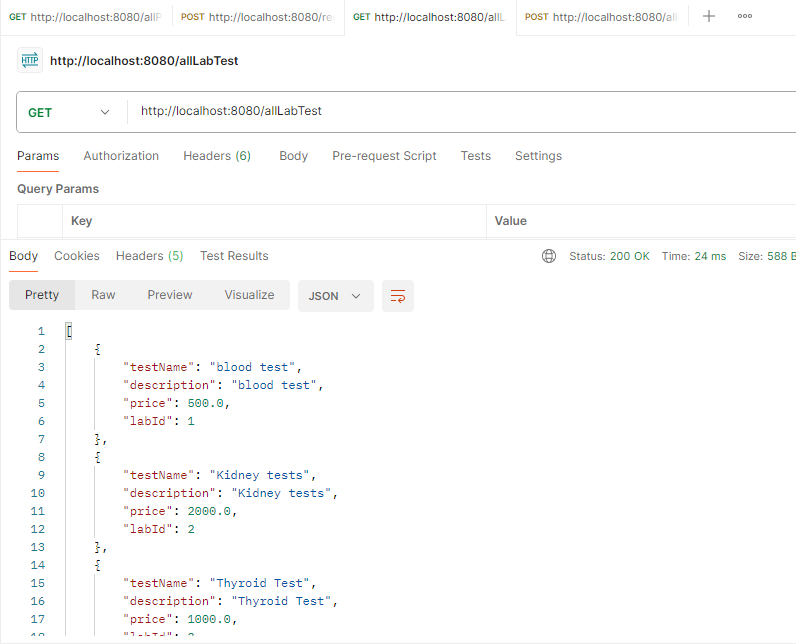
Remove[] bracket at the time of entering data



**Enter lab test detail in database**

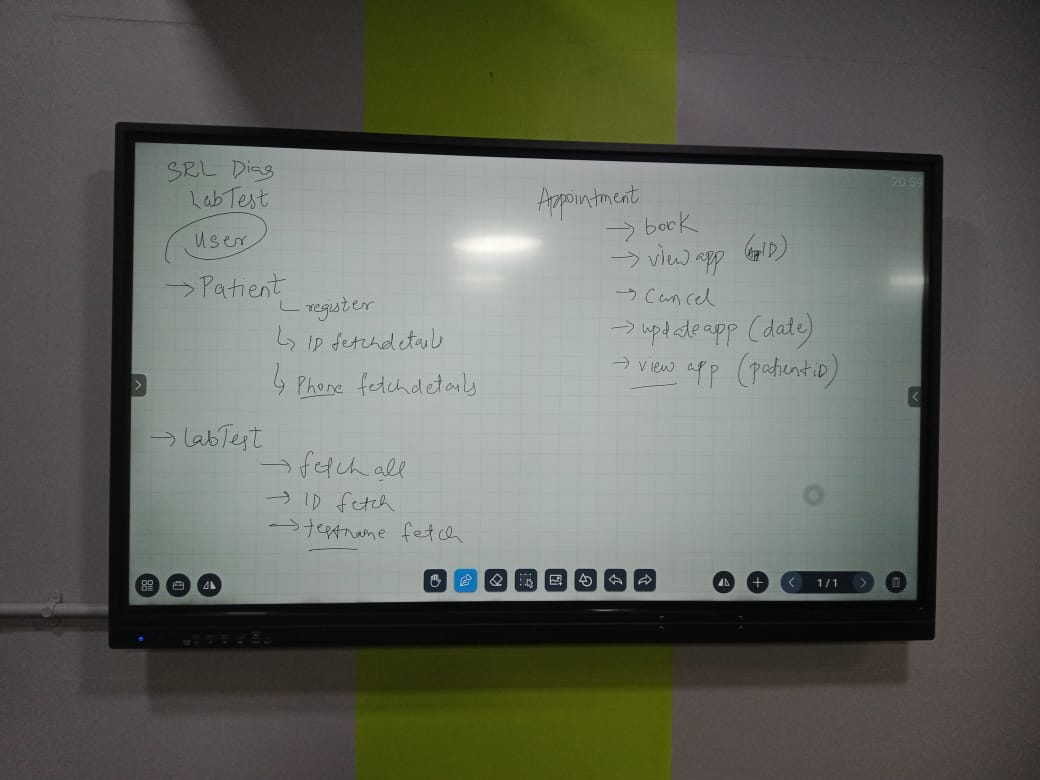


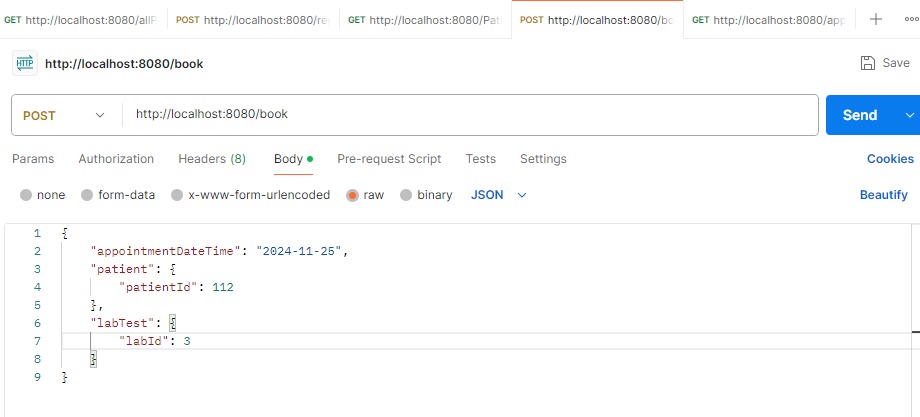
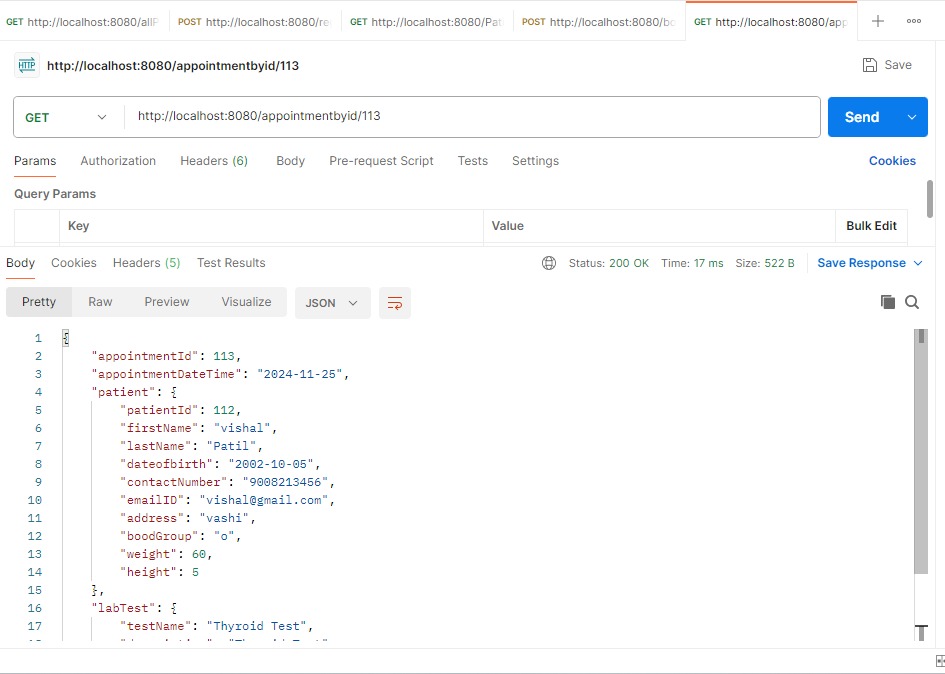
We cannot add data manually as patient we can only see the tests

 <http://localhost:8080/Patientsbyid/101>

<http://localhost:8080/registerPatient>

<http://localhost:8080/appointmentbyid/111>

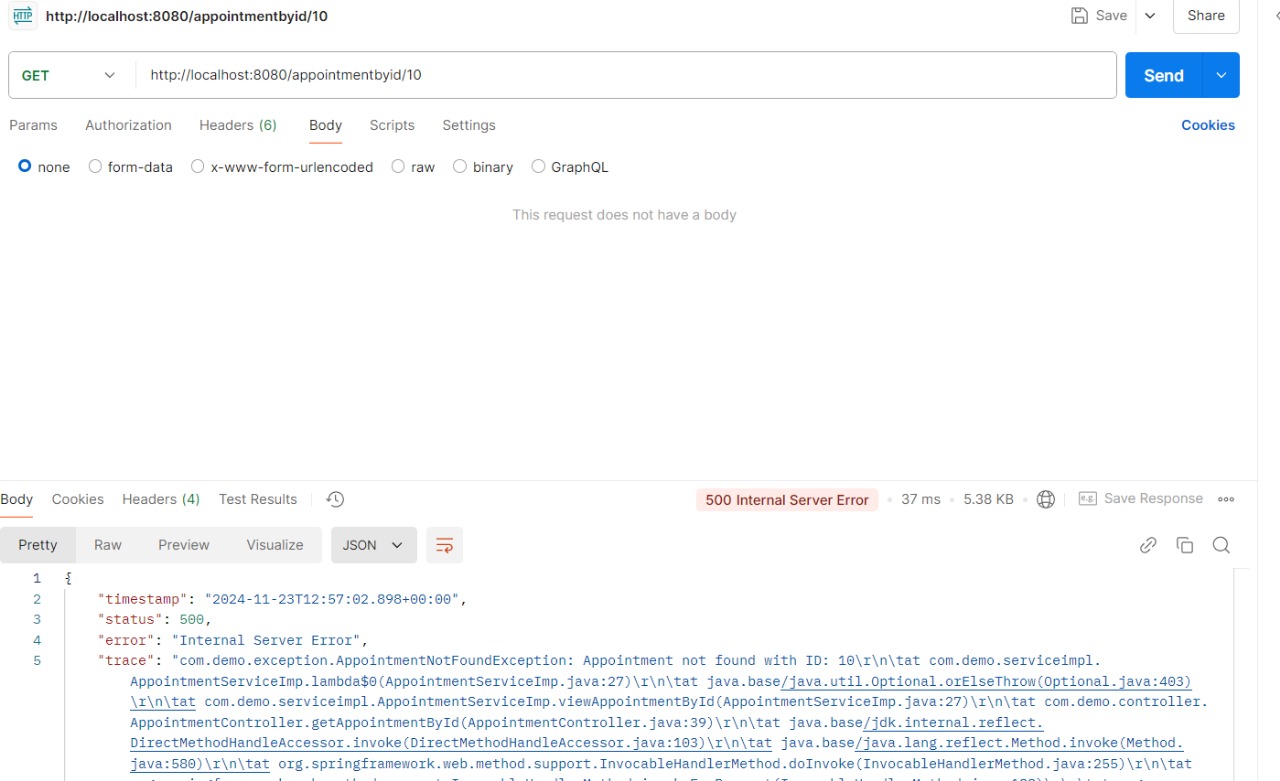




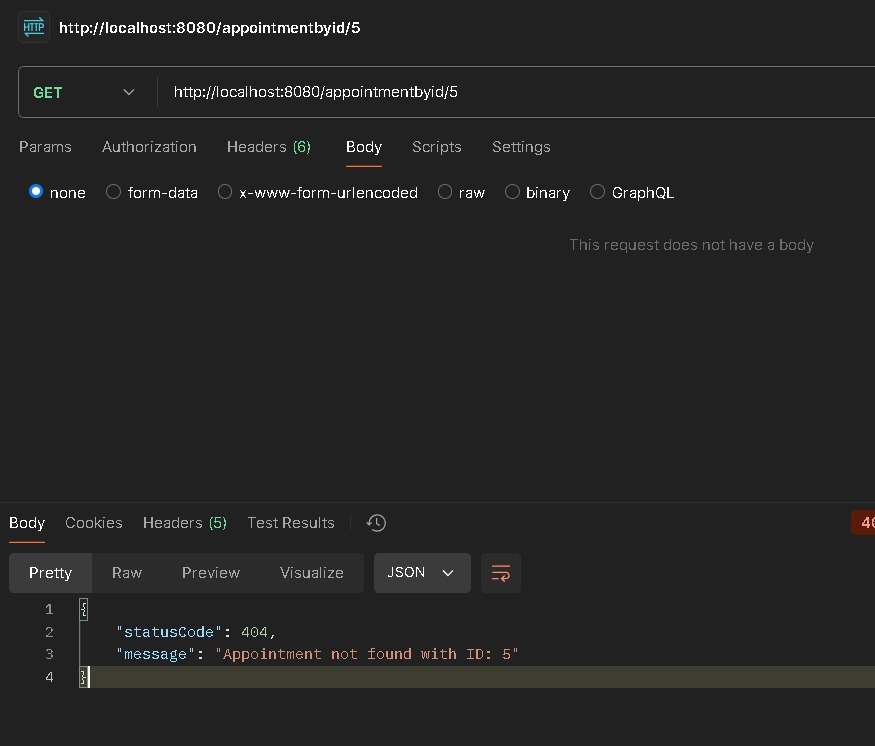
23-11-24

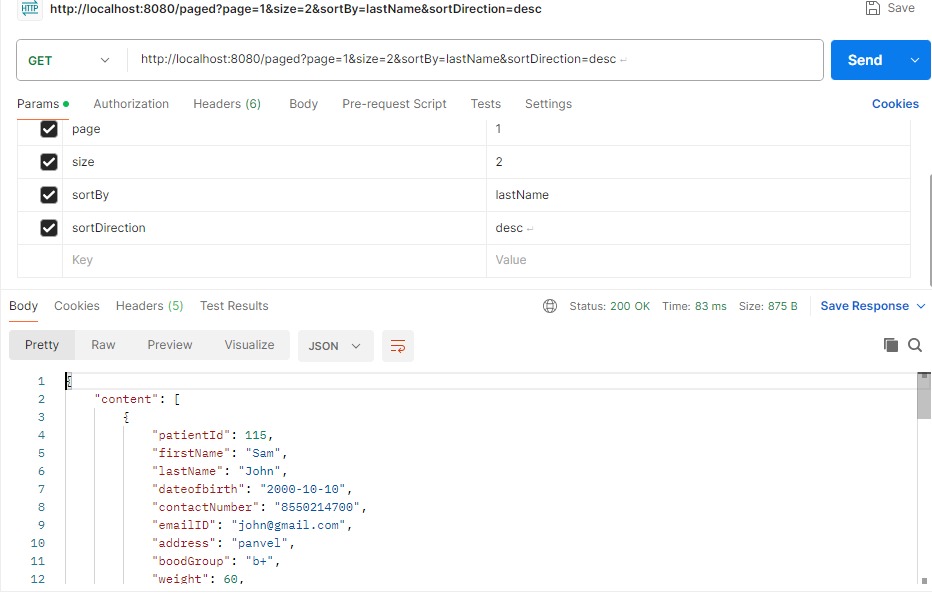
Before Without global exception handler

Add GlobalExceptionHandler for each entity pf exception class



After With global exception handler





**Phone number validation**



**Child class**

package com.demo.validation;

import jakarta.validation.Constraint;

import jakarta.validation.Payload;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Constraint(validatedBy = PhoneNumberValidator.class)

@Target({ ElementType.METHOD, ElementType.FIELD, ElementType.ANNOTATION\_TYPE,

ElementType.PARAMETER })

@Retention(RetentionPolicy.RUNTIME)

public @interface ValidPhoneNumber {

String message() default "Invalid phone number"; // Default error message

Class<?>[] groups() default {}; // Groups for validation

Class<? extends Payload>[] payload() default {}; // Additional data for validation

}

**Main class (implement child class)**

package com.demo.validation;

import jakarta.validation.ConstraintValidator;

import jakarta.validation.ConstraintValidatorContext;

public class PhoneNumberValidator implements ConstraintValidator<ValidPhoneNumber, String>{

@Override

public void initialize(ValidPhoneNumber constraintAnnotation) {

// Initialization logic, if needed

}

@Override

public boolean isValid(String value, ConstraintValidatorContext context) {

if (value == null || value.isEmpty()) {

return false; // If the value is null or empty, it's considered invalid

}

// Regex to validate a phone number (basic example, adjust as needed)

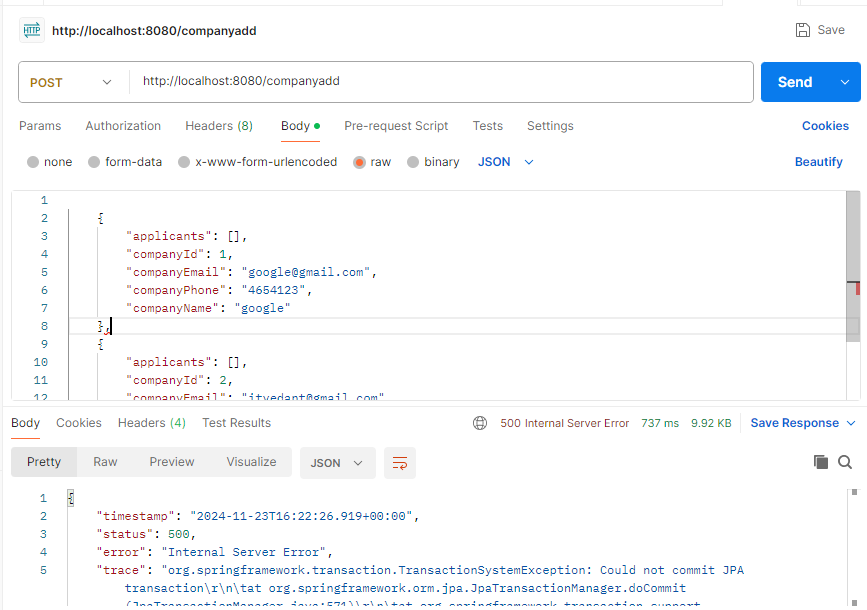
String phoneRegex = "^[6789]\\d{9}$"; // ^ it means it start with , For a 10-digit phone number

return value.matches(phoneRegex);

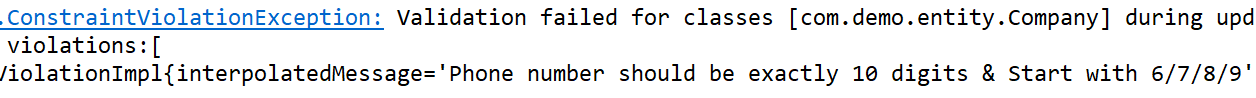
}

}

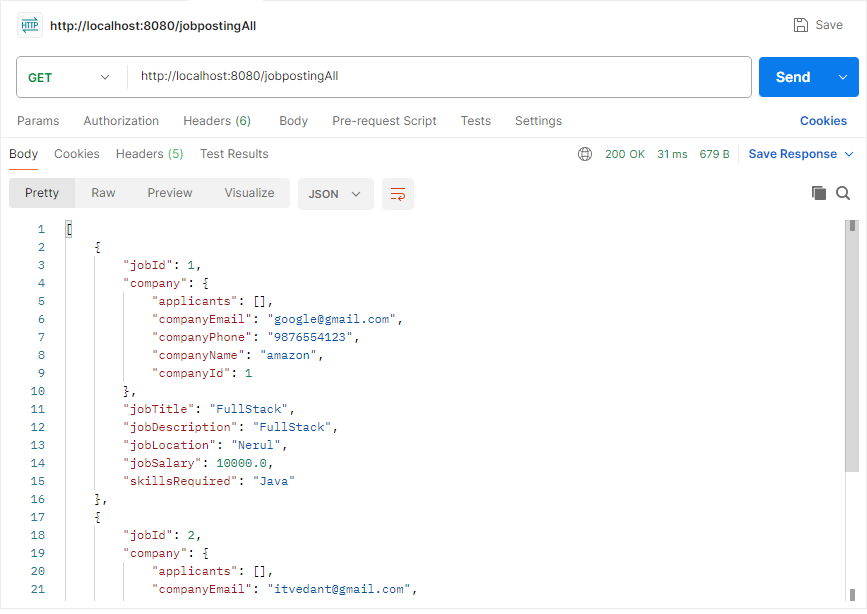
**Display error**



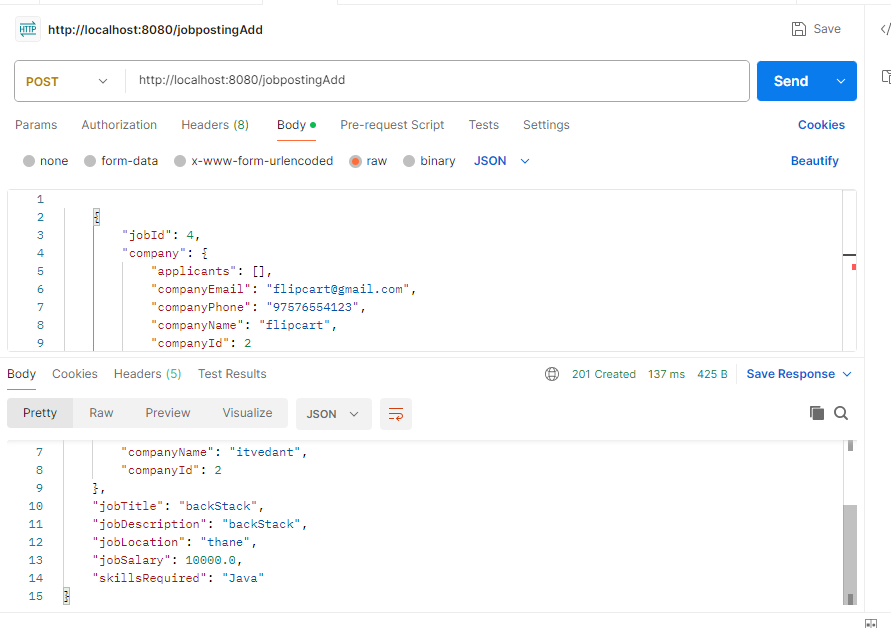
**In project console**



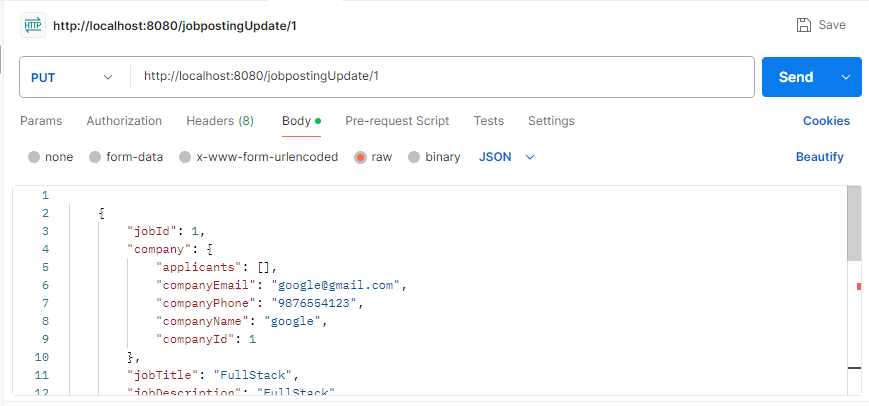
To get the data



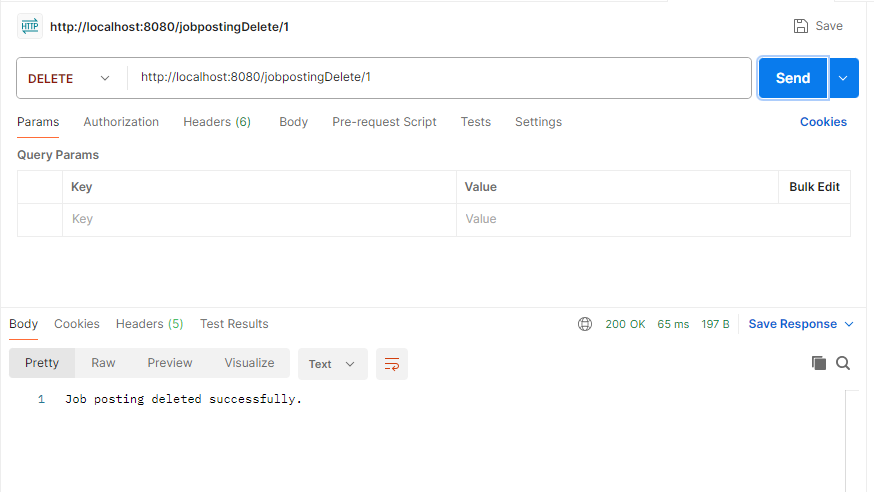
To add data



To update data



To delete data



Paging

