HOSPITAL MANAGEMENT

( SPRING BOOT PROJECT )

**INTRODUCTION:**

Spring Boot is an open source Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications. This chapter will give you an introduction to Spring Boot and familiarizes you with its basic concepts.

**SYSTEM SOFTWARE USED:**

Spring Boot Project:

There are multiple approaches to create Spring Boot project. We can use any of the following approach to create application.

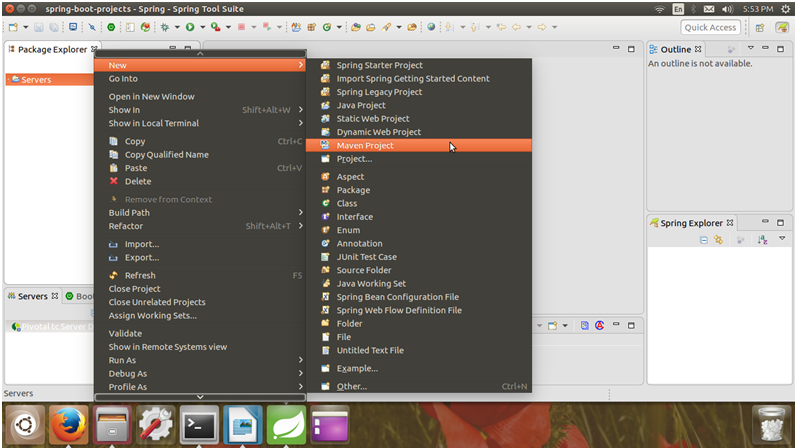
* Spring Maven Project
* Spring Starter Project Wizard
* Spring Initializer
* Spring Boot CLI

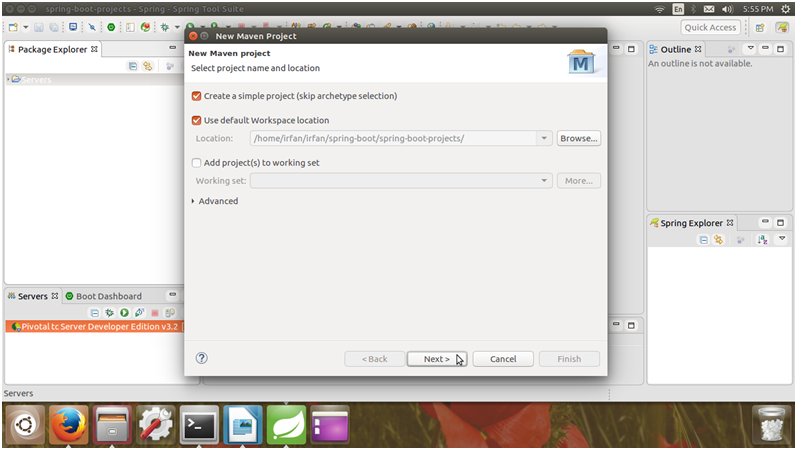
Here for all the example, we are using STS (Spring Toll Suite) IDE to create project. You can download this IDE from official site of Spring framework.

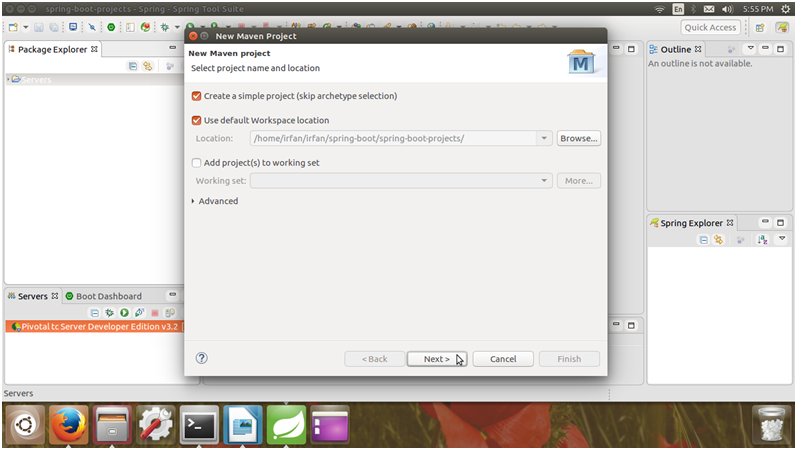
Spring Boot Maven Project

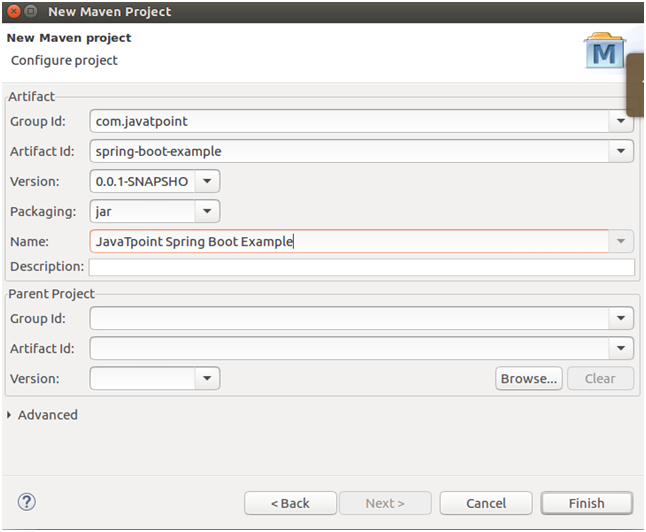
Creating Spring Boot project by creating maven project. It includes the following steps.

How to create an mavan project using spring tool suite with a following steps ,steps have to be followed while creating mavan project ,









This Maven project has a pom.xml file which contains the following default configuration.

// pom.xml

1. **<project** xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
2. **<modelVersion>**4.0.0**</modelVersion>**
3. **<groupId>**com.javatpoint**</groupId>**
4. **<artifactId>**spring-boot-example**</artifactId>**
5. **<version>**0.0.1-SNAPSHOT**</version>**
6. **<name>**JavaTpoint Spring Boot Example**</name>**
7. **</project>**

**Spring Boot Annotations:**

AUTOWIRED:

The @Autowired annotation provides more fine-grained control over where and how autowiring should be accomplished. The @Autowired annotation can be used to autowire bean on the setter method just like @Required annotation, constructor, a property or methods with arbitrary names and/or multiple arguments.

@Autowired on Setter Methods

You can use @Autowired annotation on setter methods to get rid of the <property> element in XML configuration file. When Spring finds an @Autowired annotation used with setter methods, it tries to perform byType autowiring on the method.

Example

@Component

public class Customer

{

private Person person;

@Autowired

public Customer(Person person)

{

this.person=person;

}

}

POSTMAPPING:

The @PostMapping is specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.POST).

The @PostMapping annotated methods in the @Controller annotated classes handle the HTTP POST requests matched with given URI expression.

@PostMapping Example:

@PostMapping(path = "users",

consumes = MediaType.APPLICATION\_JSON\_VALUE,

produces = MediaType.APPLICATION\_JSON\_VALUE)

public ResponseEntity<User> create(@RequestBody User newUser) {

User user = userService.save(newUser);

if (user == null) {

throw new ServerException();

} else {

return new ResponseEntity<>(user, HttpStatus.CREATED);

}

}

GETMAPPING:

The @GetMapping annotation is a specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET).

The @GetMapping annotated methods in the @Controller annotated classes handle the HTTP GET requests matched with given URI expression.

@GetMapping Example:

@RestController

public class UserController {

@Autowired

UserService userService;

@GetMapping("users")

public ResponseEntity<List<User>> getAll() {

return new ResponseEntity<>(userService.getAll(), HttpStatus.OK);

}

@GetMapping("users/{id}")

public ResponseEntity<User> getById(@PathVariable long id) {

Optional<User> user = userService.getById(id);

if (user.isPresent()) {

return new ResponseEntity<>(user.get(), HttpStatus.OK);

} else {

throw new RecordNotFoundException();

}

DELETEMAPPING:

The DELETE HTTP method is used to delete the resource and @DeleteMapping annotation for mapping HTTP DELETE requests onto specific handler methods.

Specifically, @DeleteMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.DELETE).

Example:

@DeleteMapping("/employees/{id}")

public Map<String, Boolean> deleteEmployee(@PathVariable(value = "id") Long employeeId)

throws ResourceNotFoundException {

Employee employee = employeeRepository.findById(employeeId).orElseThrow(() -> ResourceNotFoundException("Employee not found for this id :: " + employeeId));

employeeRepository.delete(employee);

Map<String, Boolean> response = new HashMap<>();

response.put("deleted", Boolean.TRUE);

return response;

}

PUTMAPPING

@PutMapping: It maps the HTTP PUT requests on the specific handler method. It is used to create a web service endpoint that creates or updates It is used instead of using: @RequestMapping(method = RequestMethod.PUT)

Example:

@PutMapping("/employees/{id}")

public Map<String, Boolean> deleteEmployee(@PathVariable(value = "id") Long employeeId)

throws ResourceNotFoundException {

Employee employee = employeeRepository.findById(employeeId)

.orElseThrow(() -> new ResourceNotFoundException("Employee not found for this id :: " + employeeId));

employeeRepository.delete(employee);

Map<String, Boolean> response = new HashMap<>();

response.put("deleted", Boolean.TRUE);

return response;

OVERRIDE:

The @Override annotation is a standard Java annotation that was first introduced in Java 1.5. The @Override annotation denotes that the child class method overrides the base class method.

For two reasons, the @Override annotation is useful.

If the annotated method does not actually override anything, the compiler issues a warning.

It can help to make the source code more readable.

id auto generated

Why we use @Override annotation:

Because of the following two advantages, using the @Override annotation when overriding a method is considered a best practice for coding in Java:

1) You’ll get a compile-time error if the programmer makes a mistake while overriding, such as using the wrong method name or parameter types. Because you are informing the compiler that you are overriding this method by using this annotation. If you don’t use the annotation, the sub-class method will be treated as a new method in the subclass (rather than the overriding method).

SERVICE: @Service annotation is used in your service layer and annotates classes that perform service tasks, often you don't use it but in many case you use this annotation to represent a best practice. For example, you could directly call a DAO class to persist an object to your database but this is horrible.

**USED METHODOLOGY:**

**Get the patient details using GET METHOD in postman**

GET Request in Postman

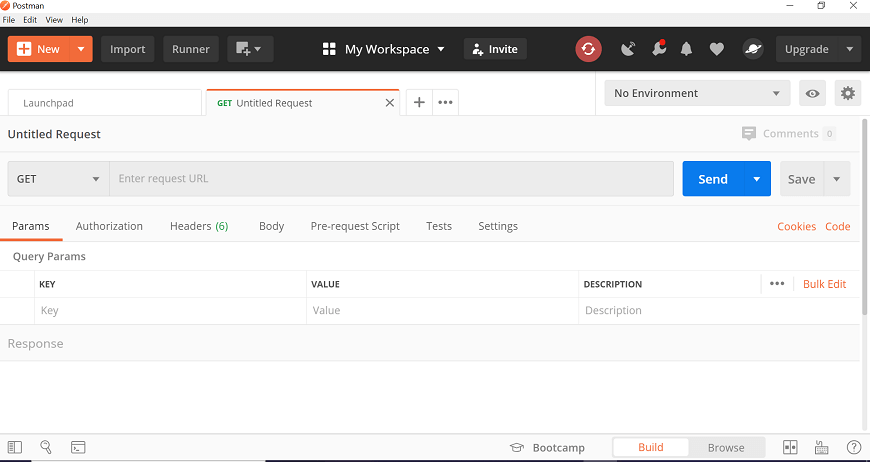
Since now we know how to create the request in Postman, it's time to work on GET request.  A GET request gets the information from the server. When you make the GET request on the server, then the server responds to the request.

GET request will not affect any data on the server. Means, there is no creation, updation, addition, or deletion of data on the server when you are making a GET request.

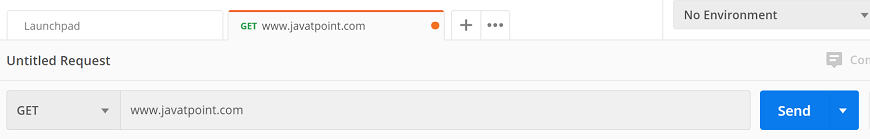
GET request contains all information inside the URL, and because of that, some people do not prefer to use GET request while they are sending some confidential data such as passwords. For example, if you search anything on Google, you actually using a GET request because there is no sensitive information, and you are just requesting the page. You can try to search for something on Google; you will get the same search string in the URL.

To create the first GET request in Postman, follow the following steps:

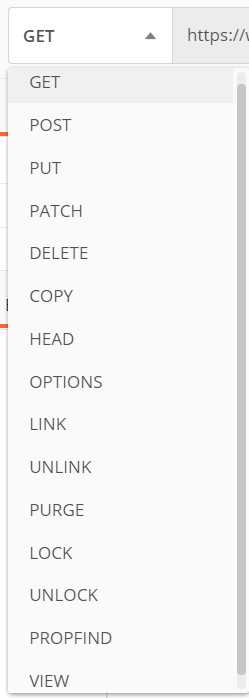
**Step 1:** Create a request. To create a new request, open a new tab, click from the **+** plus button.



**Step 2:** Enter the URL in the address bar. We will use ,



**Step 3:** Now, select the GET request from the list of request methods.



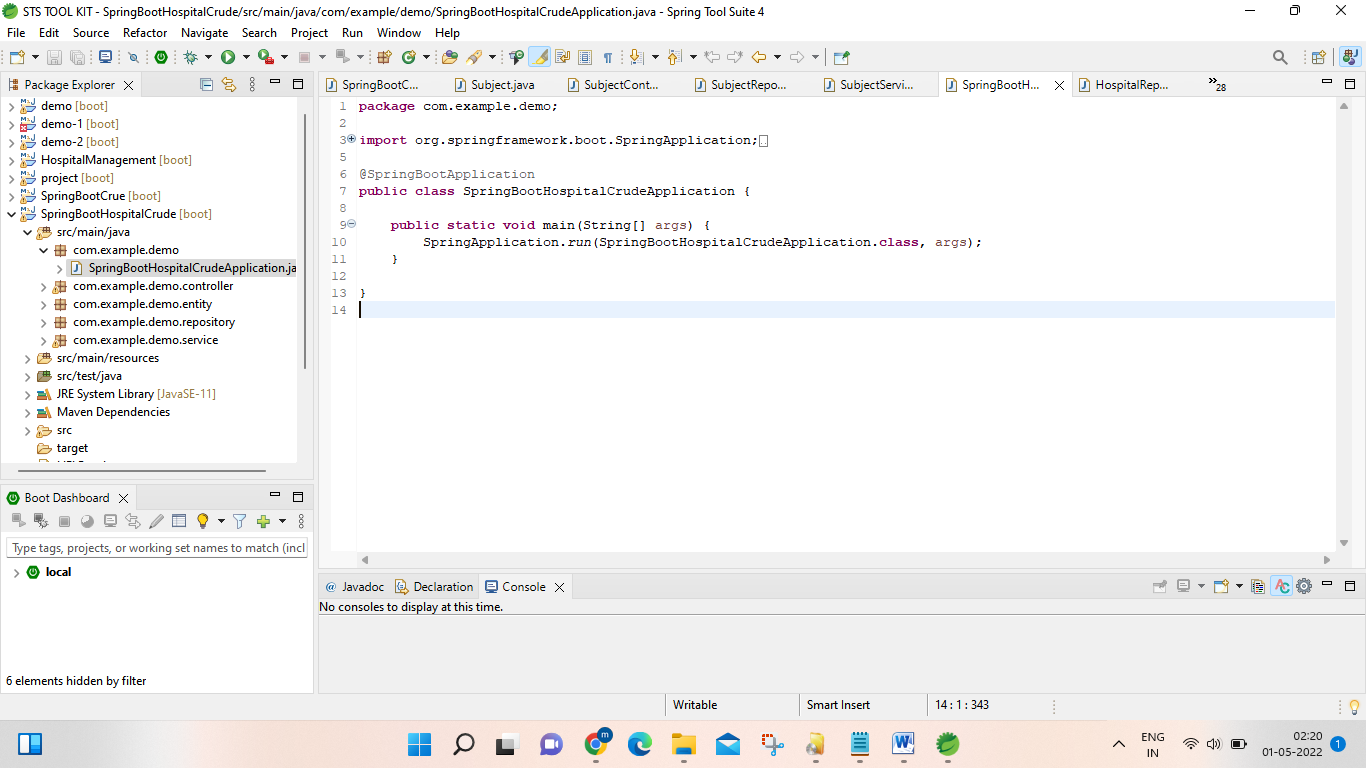
**Step 4:** Now press send.

GET Request in Postman

**Step 5:** Once you press the send button, you will get the response from the server. Make sure you have a proper internet connection; otherwise, you will not get a response.



SPRING PROJECT CLASSES WITH PACKAGE ALONG WITH SCREENSHOTS



Main class with package demo ,

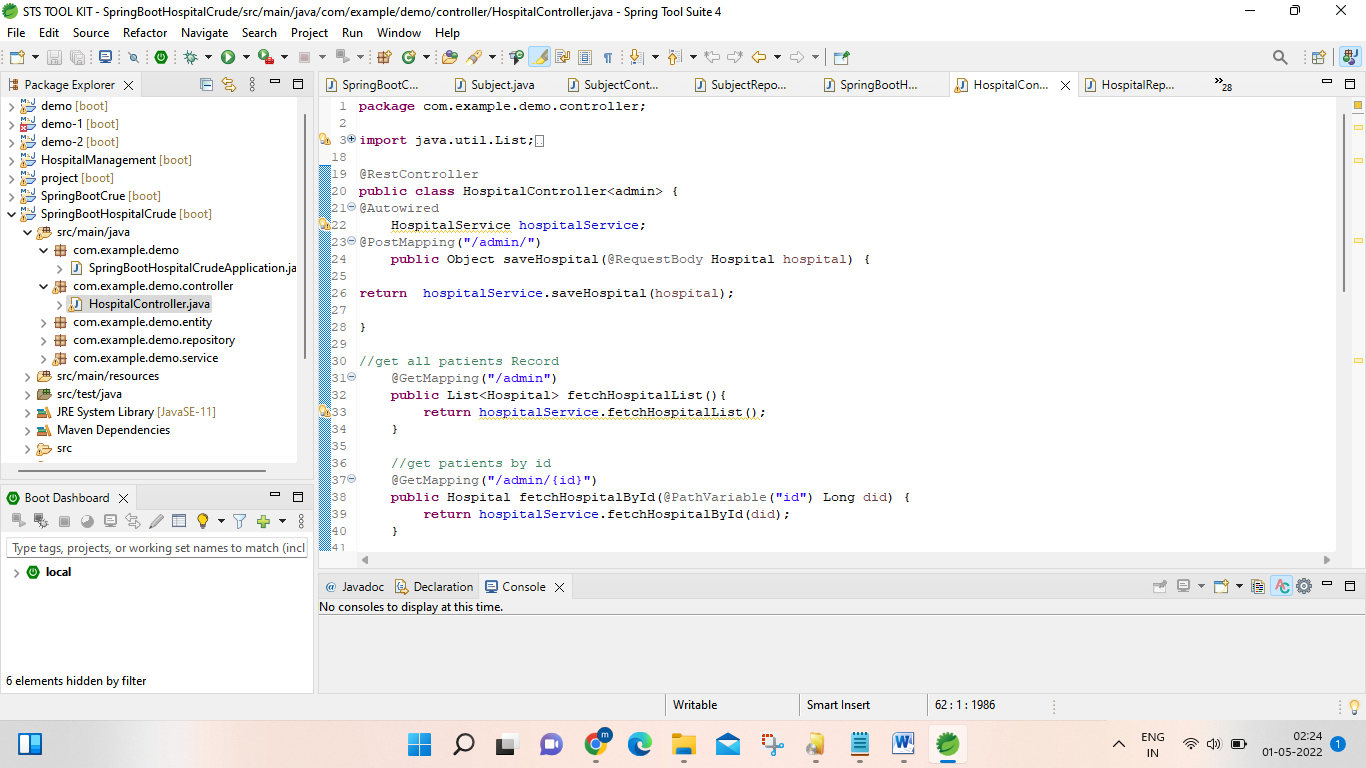
Java Main Method

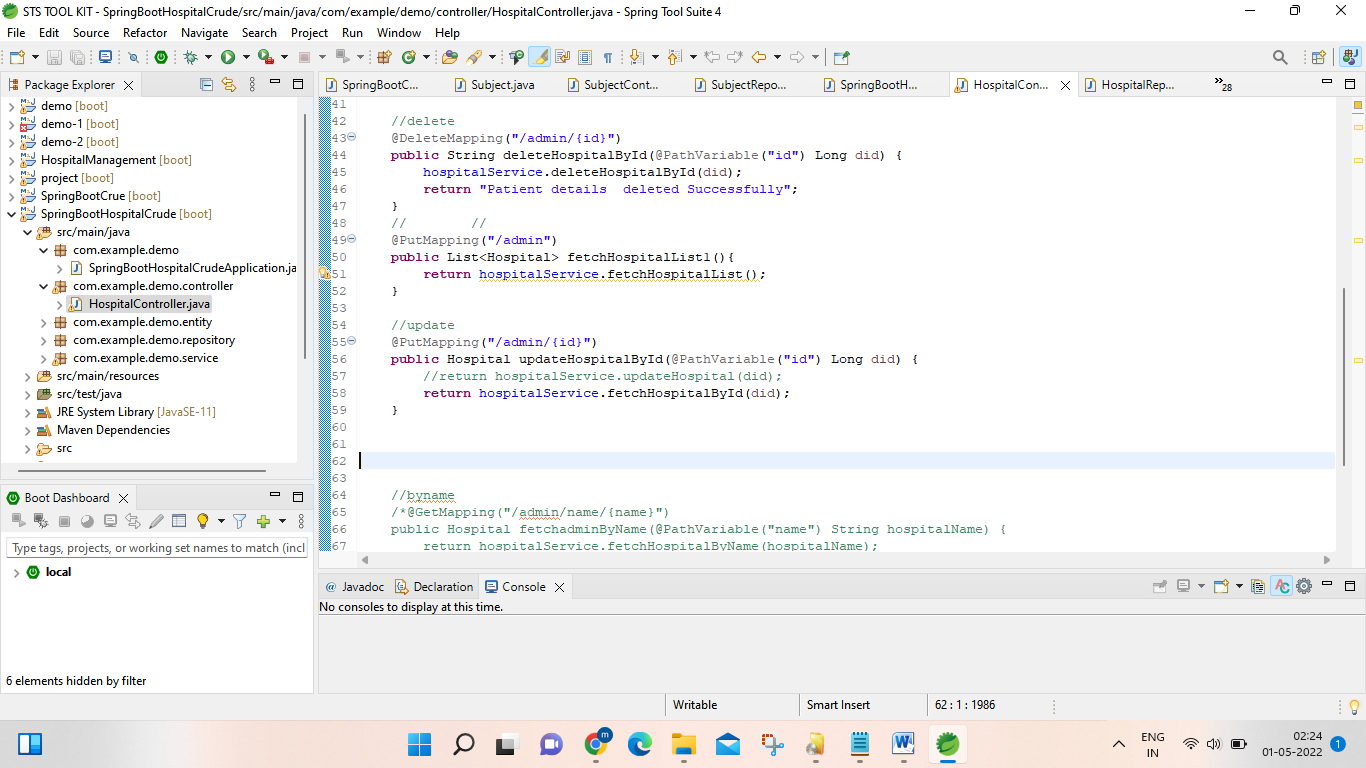
public: It is an access specifier. We should use a public keyword before the main() method so that JVM can identify the execution point of the program. If we use private, protected, and default before the main() method, it will not be visible to JVM.

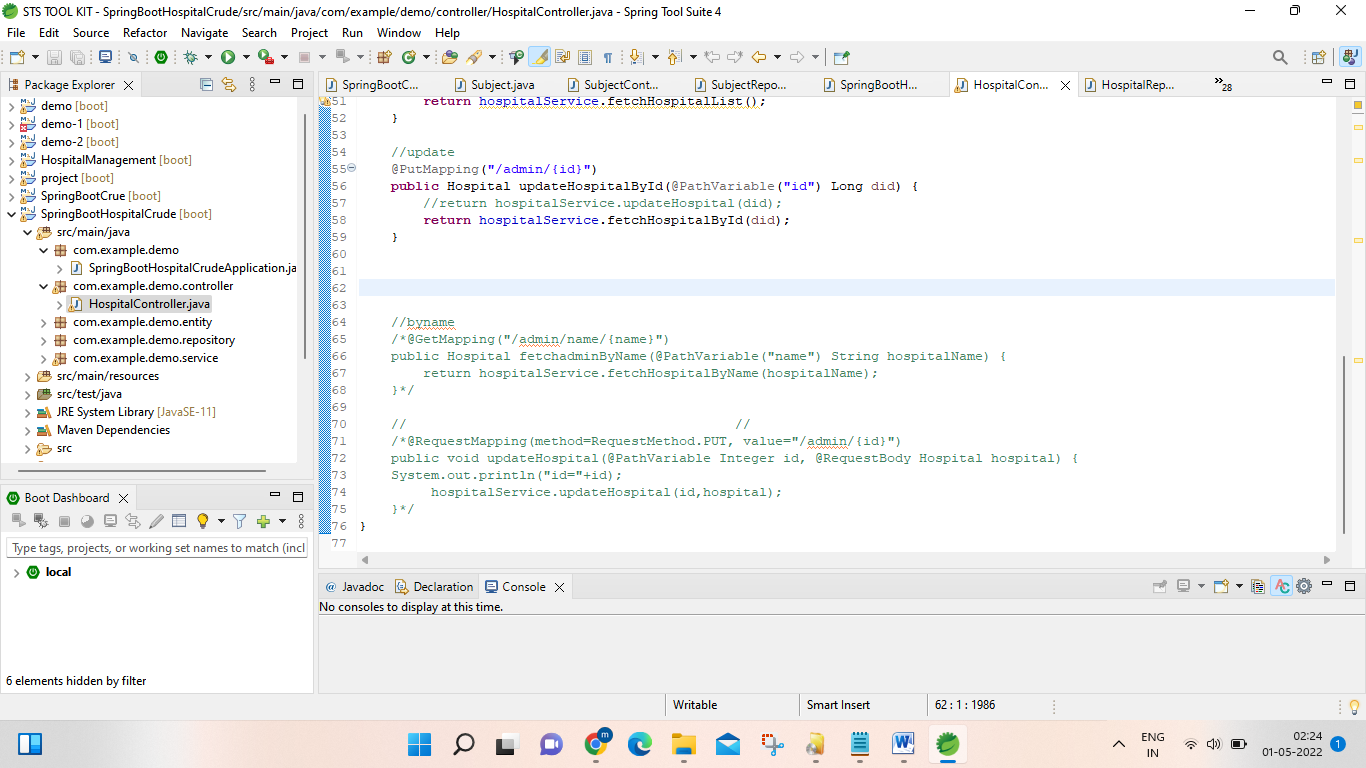
static: You can make a method static by using the keyword static. We should call the main() method without creating an object. Static methods are the method which invokes without creating the objects, so we do not need any object to call the main() method.

void: In Java, every method has the return type. Void keyword acknowledges the compiler that main() method does not return any value.

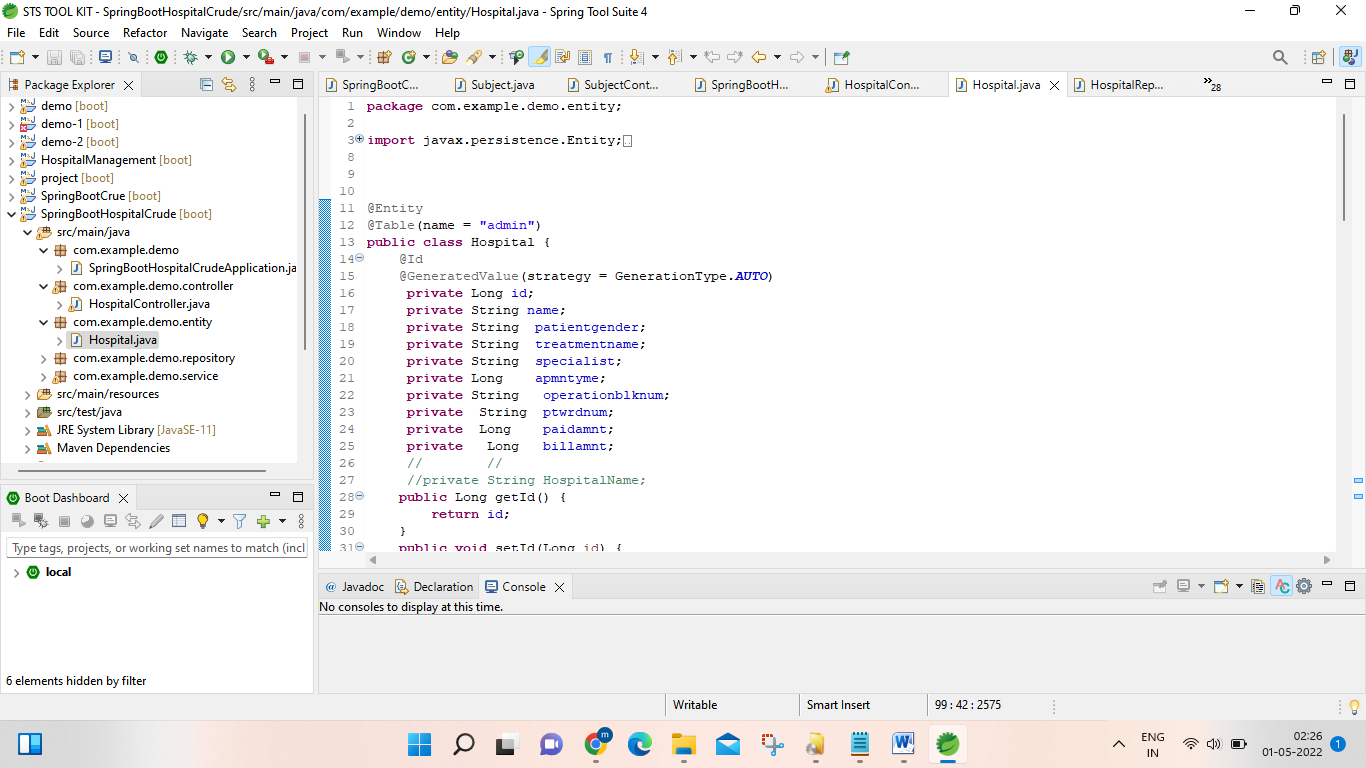
**controller with class name HospitalController**

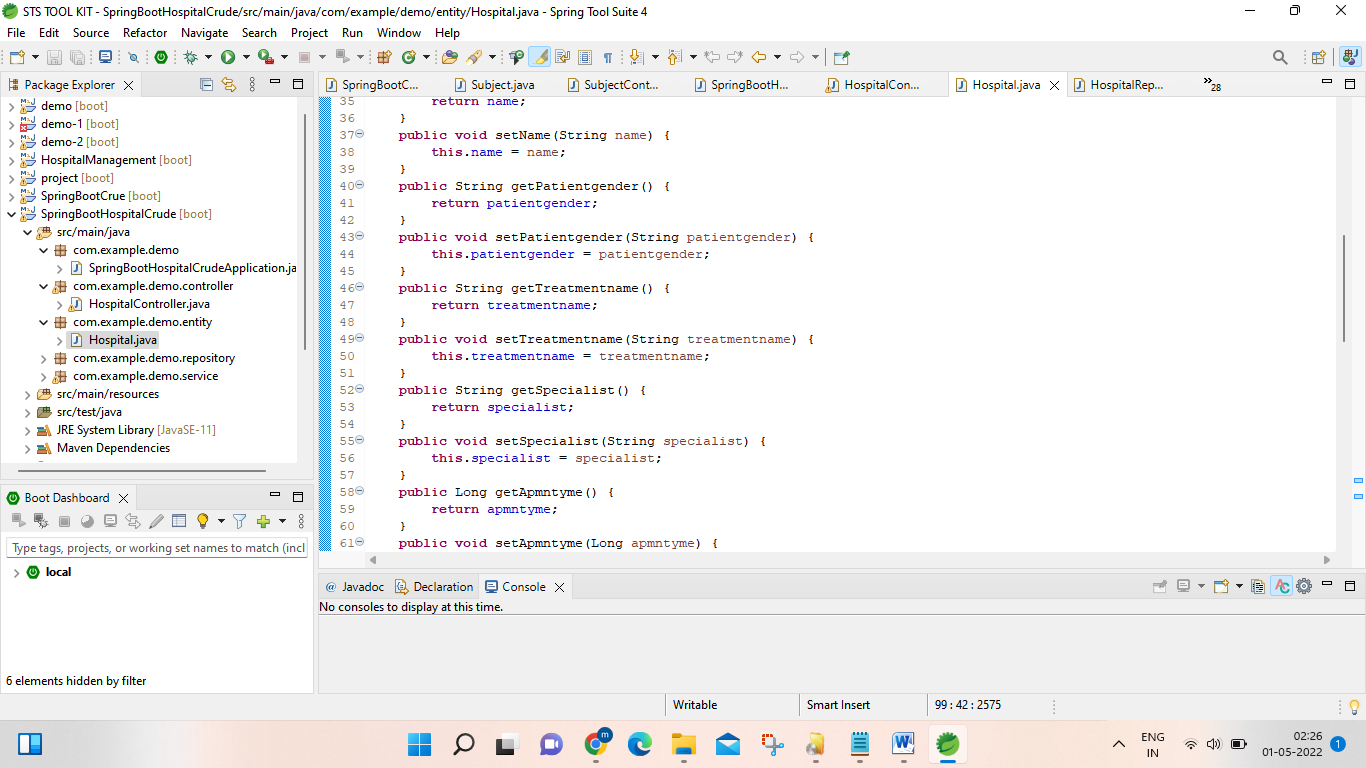
****

****

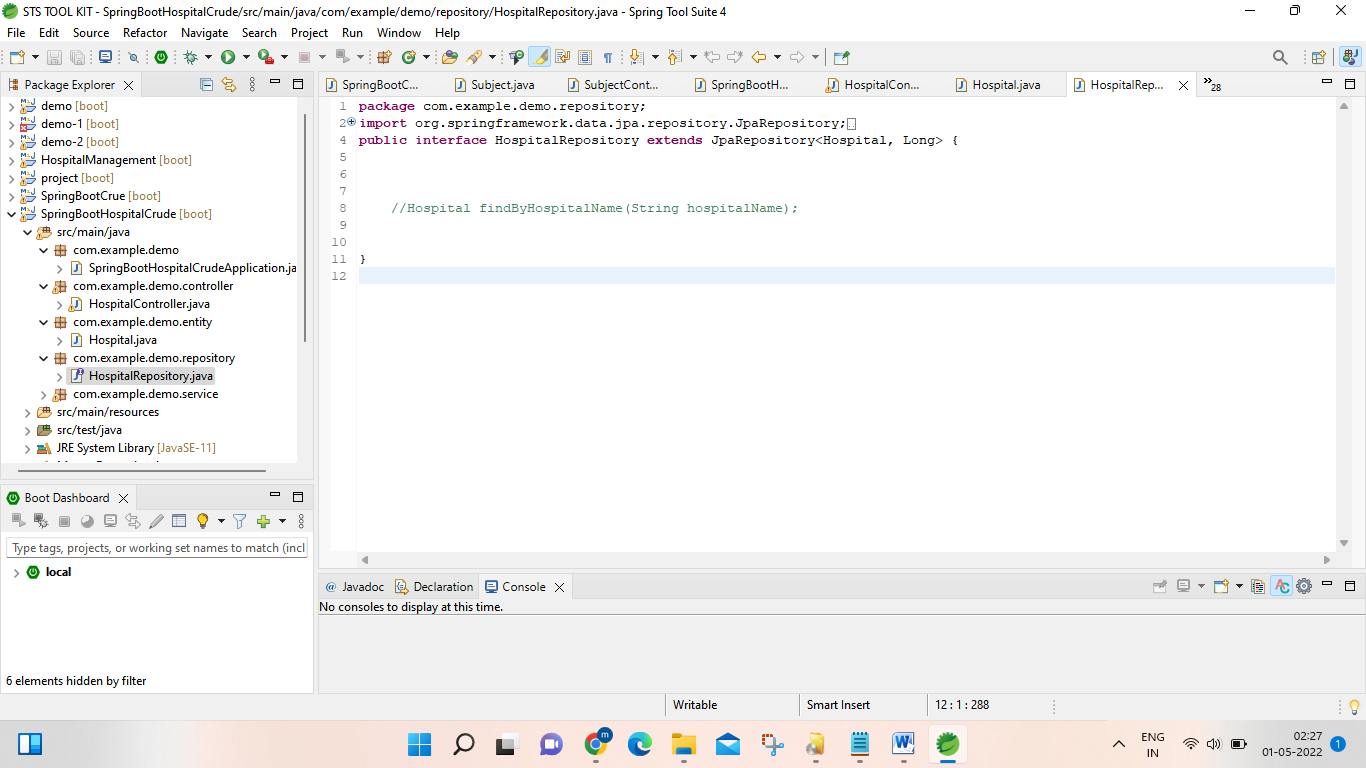
****

**entity with class name Hospital:**

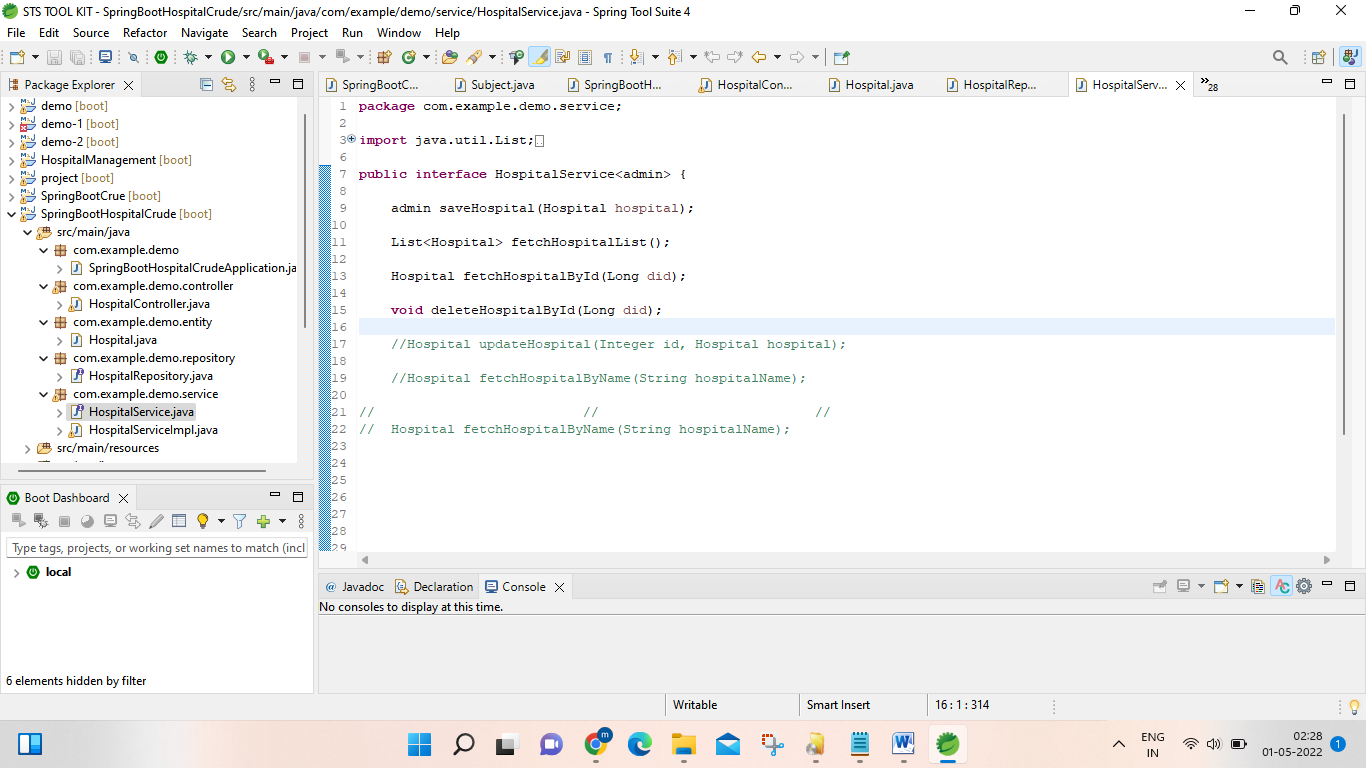
****

****

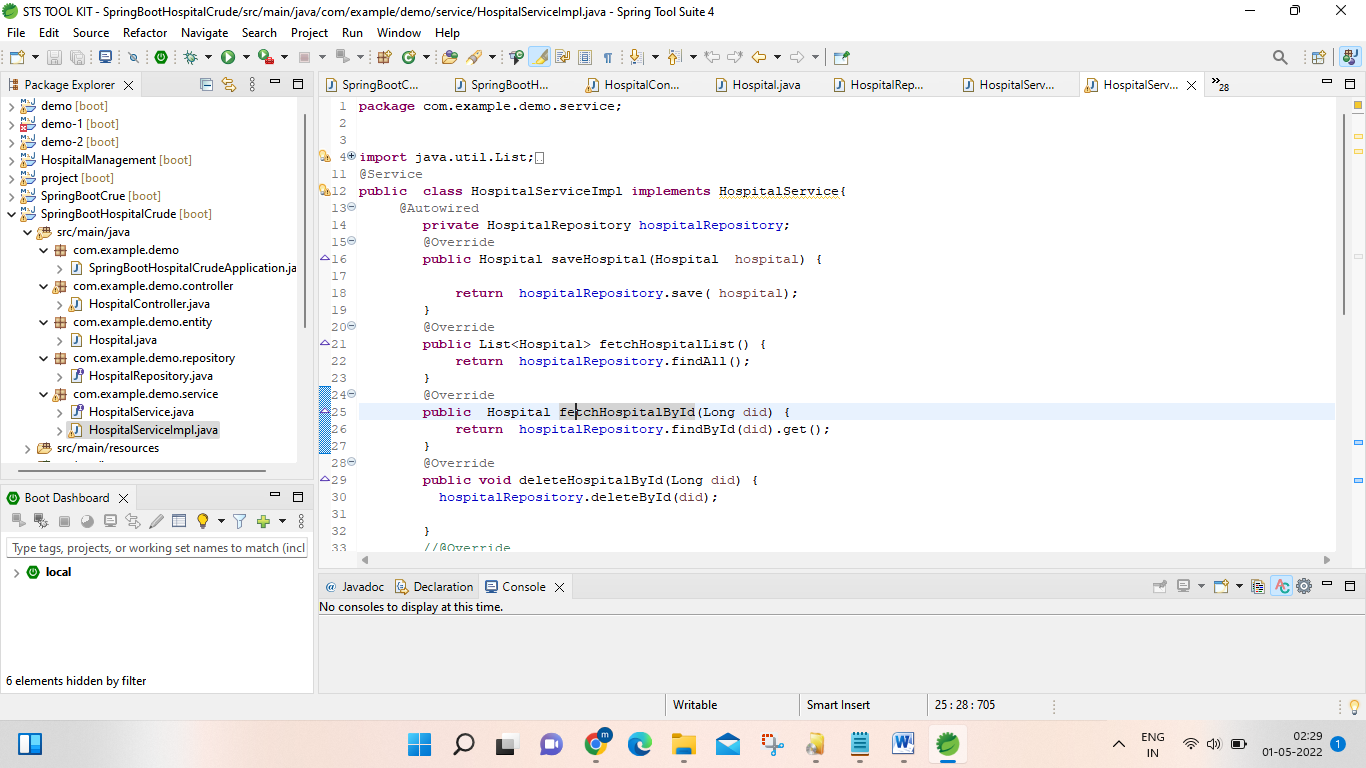
**repository with class name HospitalRepository**

****

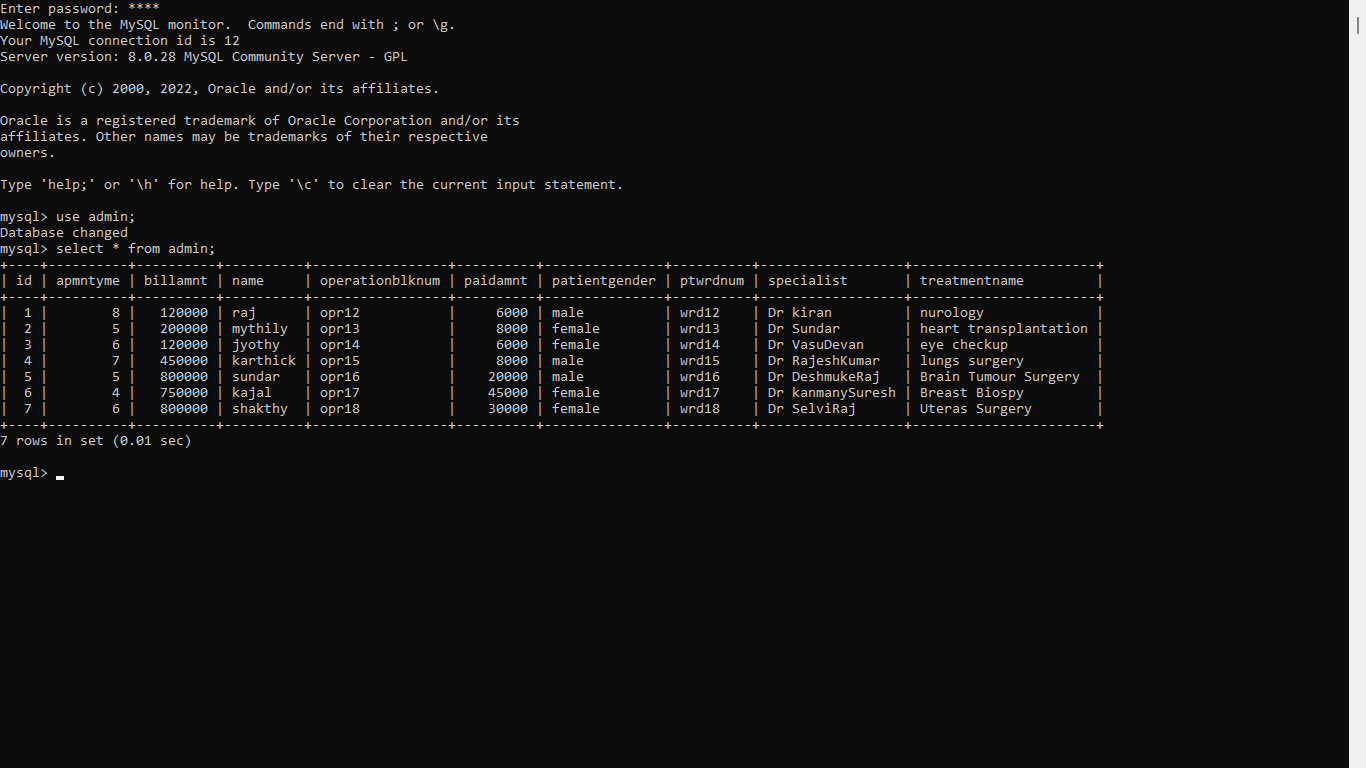
**service with interface name HospitalService**

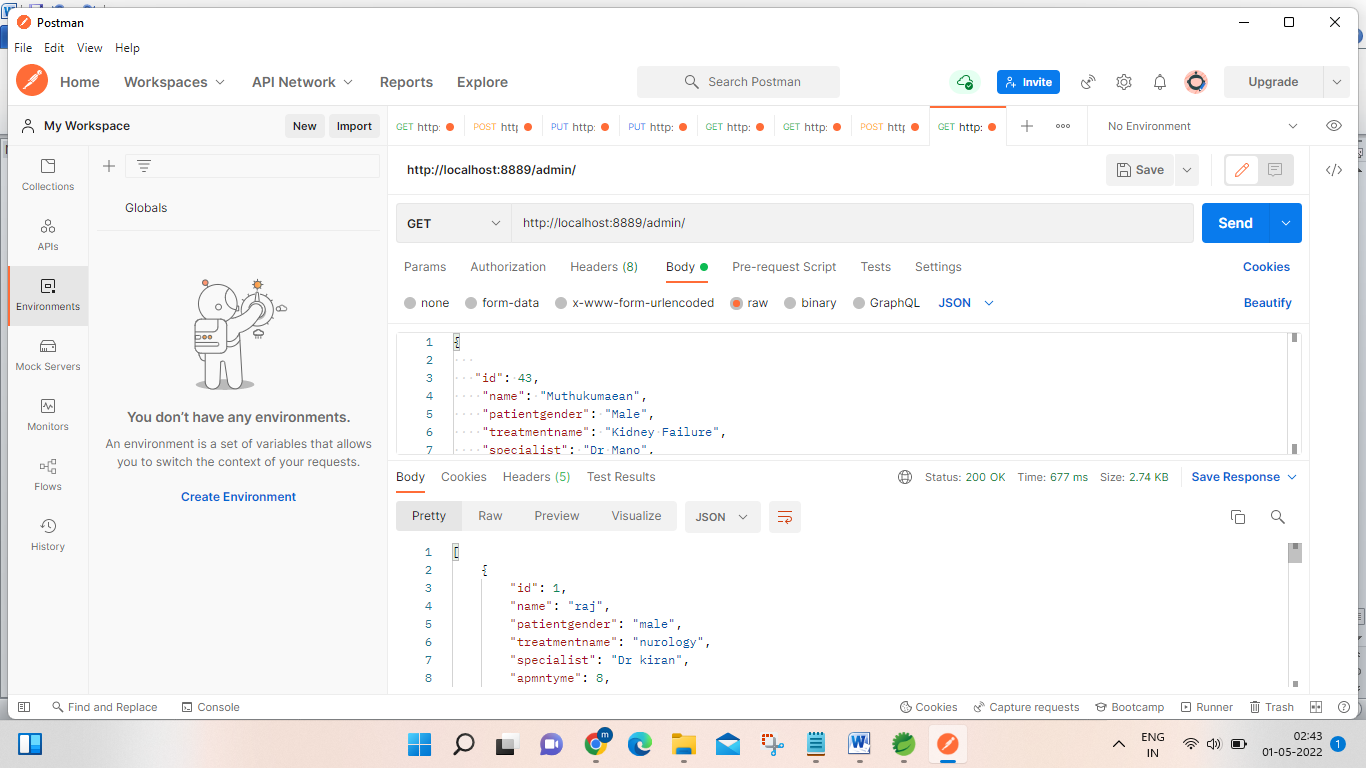
****

**service with class name HospitalServiceImpl**

****

**FRONT END AND BACK END SUUPORTER**

****

****

**Conclusion:**

**Registeration of patient details and patient treatment details using sts code run at application properties ,and show the details about patient using database saver in sql ,and enter the informations about patients details using postman app ,not only enter the details and also we can perform various operations like update the patient details,delete the patient details if the patient had discharged means .we can able to delete all of those history about patients.we can save and recover all those patients datas with hospital servicing management services in postman app.**