Arpit Singh

19BCG10069

Kaiburr – Task 1

Technical Task

(Placement)

GitHub Link: **https://github.com/TSM-ArpitSG/Kaiburr/tree/main/Kaibur_Tasks/Task_1**

Task 1:

Java REST API Example:

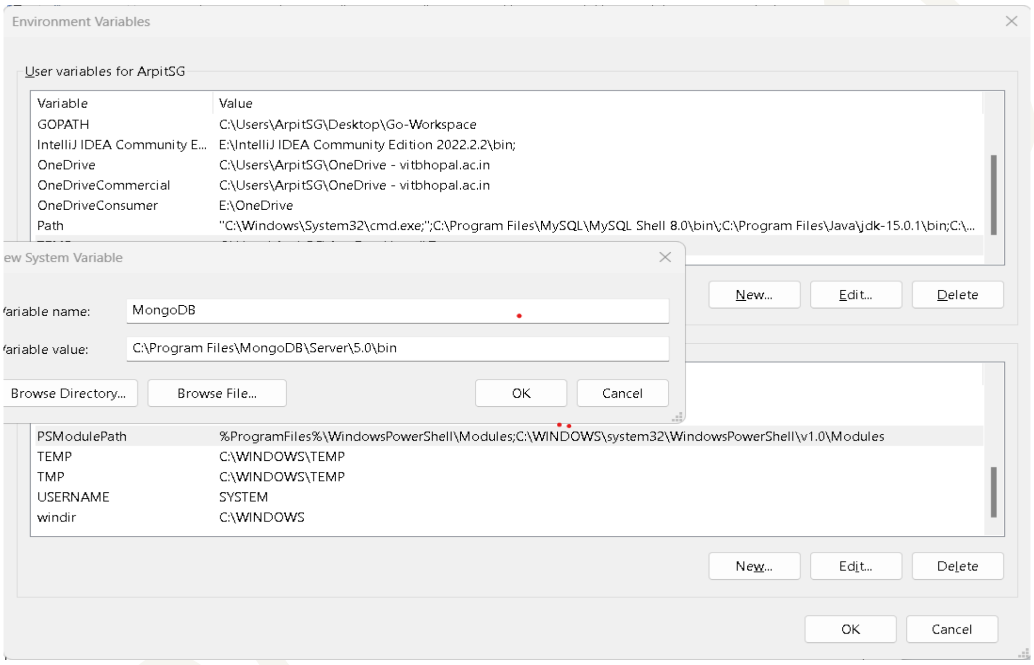
- Implement an application in java which provides a REST API with endpoints for searching, creating and deleting “server” objects:
- GET servers. Should return all the servers if no parameters are passed. When server id is passed as a parameter - return a single server or 404 if there’s no such a server.
- PUT a server. The server object is passed as a Json-encoded message body. Here’s an example:

```
{
  "id": "1",
  "name": "my centos",
  "language": "Java",
  "framework": "Spring Boot"
}
```

- DELETE a server. The parameter is a server ID.
- GET (find) servers by name. The parameter is a string. Must check if a server name contains this string and return one or more servers found. Return 404 if nothing is found. “Server” objects should be stored in MongoDB database. Be sure that you can show how your application responds to requests using post-man, curl or any other HTTP client.

Steps:

1. **Installing MongoDB and Setting up the Environment**:
   a. Install (Setup for windows) MongoDB from MongoDB.com (Community server). - To install MongoDB, you can follow the instructions on the official MongoDB website: https://docs.mongodb.com/manual/administration/install-community/
   b. Setup the System (Environment) variable as per the desired directory
   c. For Example: *C:\Program Files\MongoDB\Server\5.0\bin.*



Environment Variables

User variables for ArpitSG

Variable	Value
GOPATH	C:\Users\ArpitSG\Desktop\Go-Workspace
IntelliJ IDEA Community E...	E:\IntelliJ IDEA Community Edition 2022.2.2\bin;
OneDrive	C:\Users\ArpitSG\OneDrive - vitbhopal.ac.in
OneDriveCommercial	C:\Users\ArpitSG\OneDrive - vitbhopal.ac.in
OneDriveConsumer	E:\OneDrive
Path	"C:\Windows\System32\cmd.exe;";C:\Program Files\MySQL\MySQL Shell 8.0\bin\;C:\Program Files\Java\jdk-15.0.1\bin;C:\...

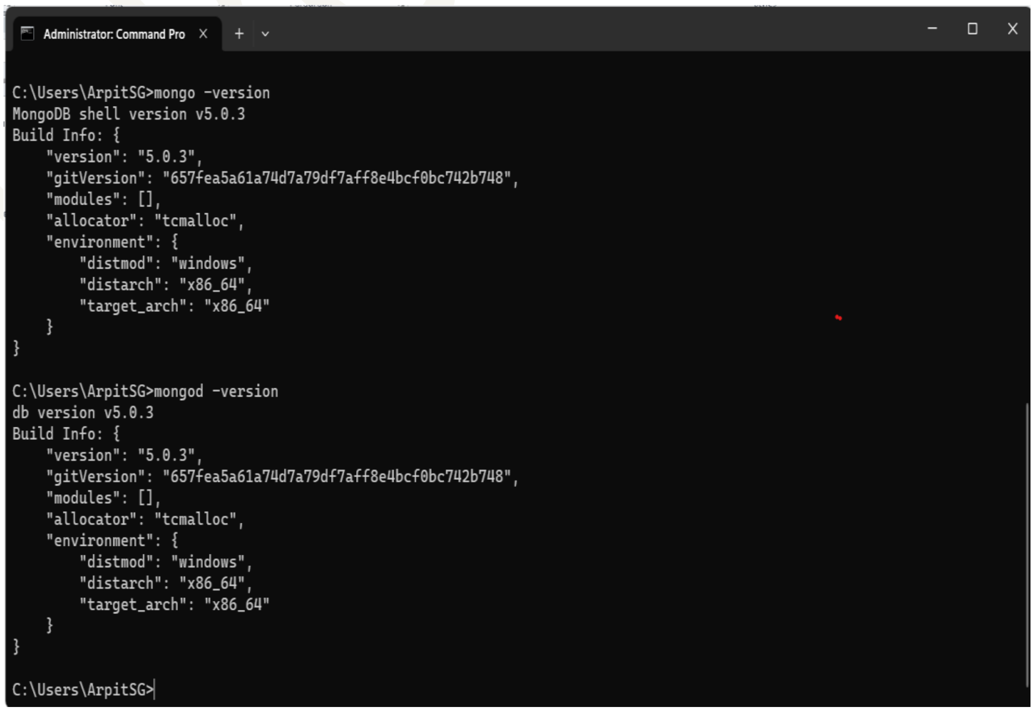
New System Variable

Variable name: MongoDB

Variable value: C:\Program Files\MongoDB\Server\5.0\bin

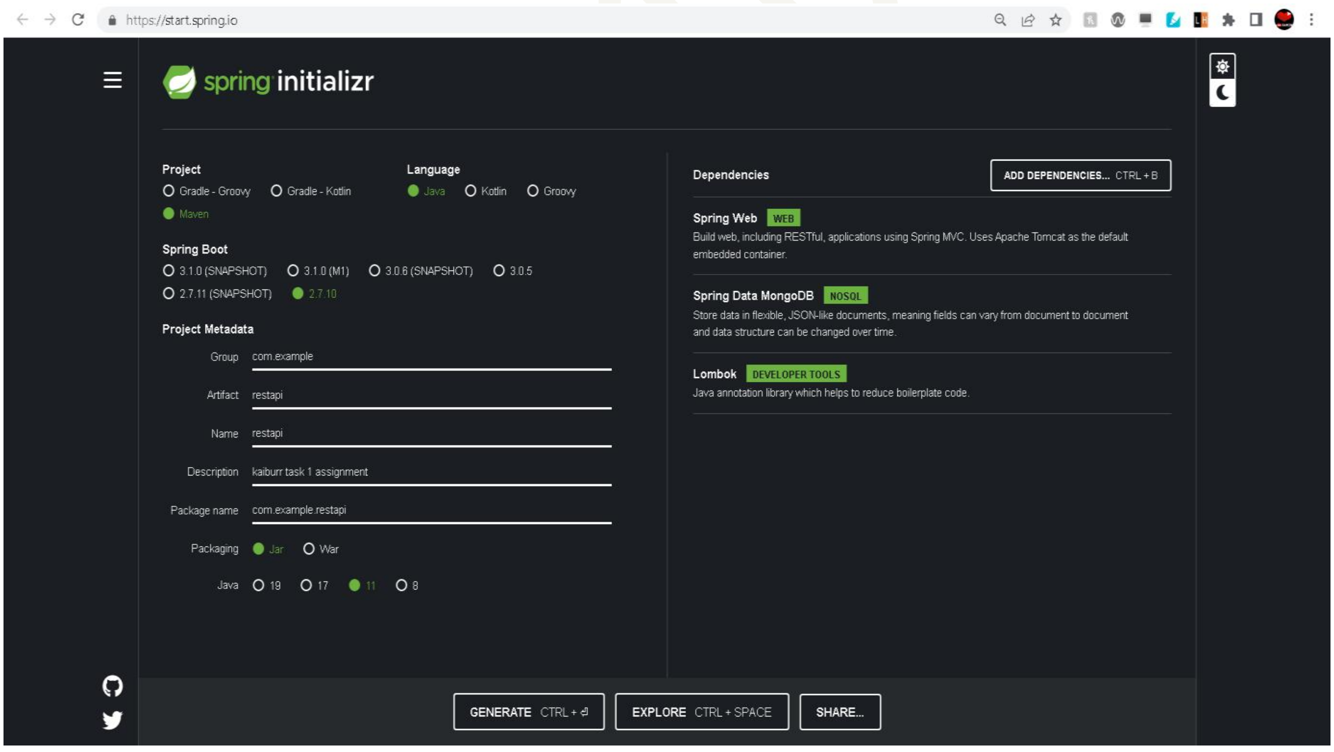
PSModulePath	%ProgramFiles%\WindowsPowerShell\Modules;C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules
TEMP	C:\WINDOWS\TEMP
TMP	C:\WINDOWS\TEMP
USERNAME	SYSTEM
windir	C:\WINDOWS

❖ Successful Installation can be checked as follows:



```
C:\Users\ArpitSG>mongo -version
MongoDB shell version v5.0.3
Build Info: {
    "version": "5.0.3",
    "gitVersion": "657fea5a61a74d7a79df7aff8e4bcf0bc742b748",
    "modules": [],
    "allocator": "tcmalloc",
    "environment": {
        "distmod": "windows",
        "distarch": "x86_64",
        "target_arch": "x86_64"
    }
}

C:\Users\ArpitSG>mongod -version
db version v5.0.3
Build Info: {
    "version": "5.0.3",
    "gitVersion": "657fea5a61a74d7a79df7aff8e4bcf0bc742b748",
    "modules": [],
    "allocator": "tcmalloc",
    "environment": {
        "distmod": "windows",
        "distarch": "x86_64",
        "target_arch": "x86_64"
    }
}

C:\Users\ArpitSG>
```

2. **Create a new Spring Boot Project Import dependencies:**
   a. To create a new Spring Boot project, you can use Spring Initializr: ***https://start.spring.io/*.**
   b. Choose the following options:
      i. ***Project*: Maven Project**
      ii. ***Language***: Java
      iii. ***Spring Boo**t*: 2.7.0 (as per preference).
      iv. ***Group***: com.example (Default).
      v. ***Artifact***: restapi
      vi. ***Packaging***: Jar
      vii. ***Java***: 11
   c. Click on the "**Add dependencies**" button and add the following depend-encies:
      i. Spring Web
      ii. Spring Data MongoDB
      iii. Lombok



Project
Gradle - Groovy Gradle - Kotlin [✓ Maven]

Language
[✓ Java] Kotlin Groovy

Spring Boot
3.1.0 (SNAPSHOT) 3.1.0 (M1) 3.0.6 (SNAPSHOT) 3.0.5
2.7.11 (SNAPSHOT) [✓ 2.7.10]

Project Metadata

Group	com.example
Artifact	restapi
Name	restapi
Description	kaiburr task 1 assignment
Package name	com.example.restapi
Packaging	[✓ Jar] War
Java	19 17 [✓ 11] 8

Dependencies

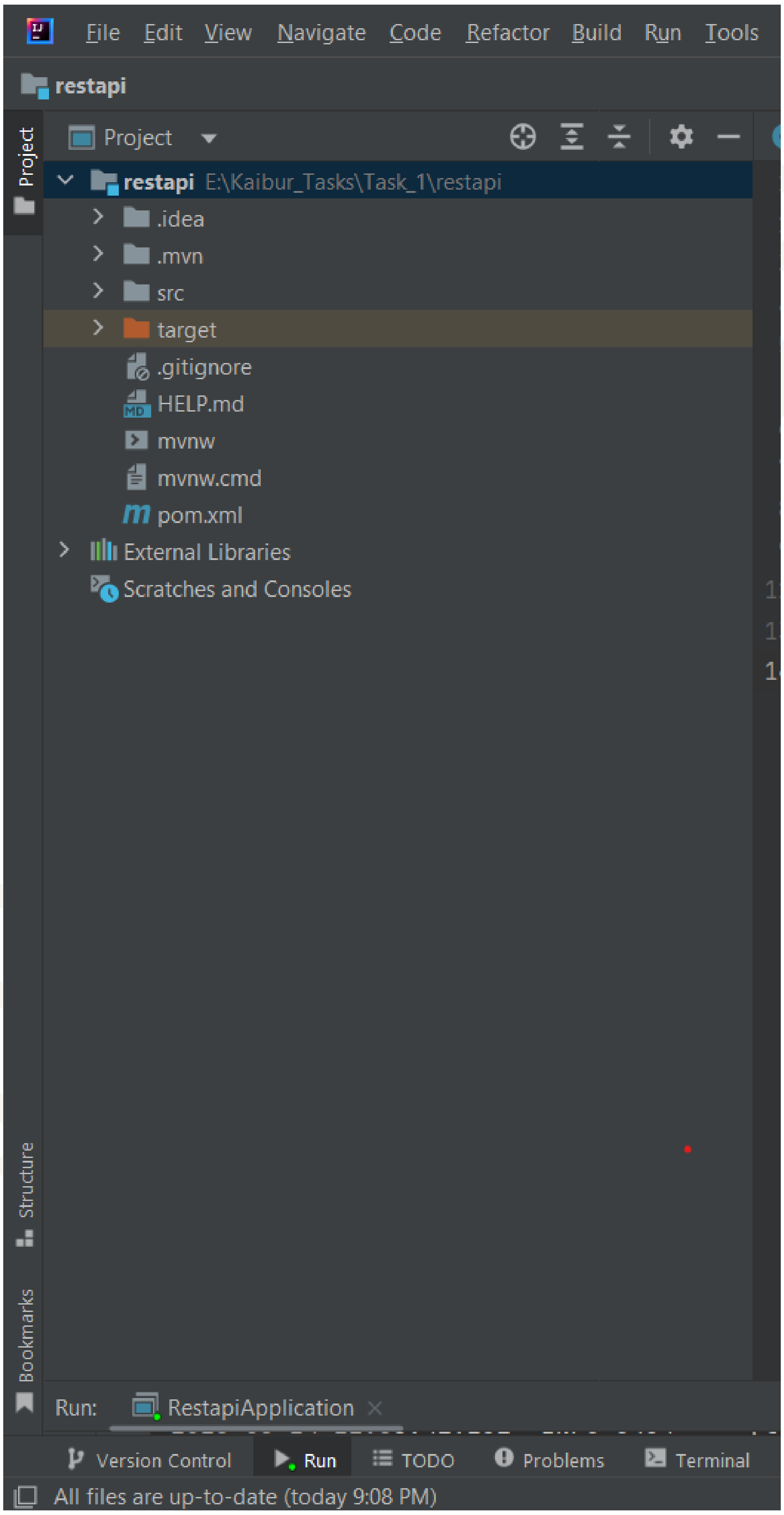
Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data MongoDB NOSQL
Store data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time.

Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

   d. Generate the project and open it in your ***preferred IDE*** (***IntelliJ IDEA in my case***).
      i. After Clicking generate we get a file named : "***restapi.zip***"

ii. Unzip it to a specified location on your system. ***(E:\Kaibur_Tasks\Task_1)***.
iii. Open this Directory in your IDE as a new project. (We get a file structure as show below)



restapi E:\Kaibur_Tasks\Task_1\restapi
.idea
.mvn
src
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles

Run: RestapiApplication

All files are up-to-date (today 9:08 PM)

Explanation on Tools/Technology(s) Used:

❖ **MongoDB**:

- ➢ ***In this project***, MongoDB is used as the database to store and retrieve data related to the servers. MongoDB is a popular NoSQL document database that is known for its scalability, flexibility, and performance. It stores data in a document format, which is similar to JSON, and can handle large volumes of unstructured data.

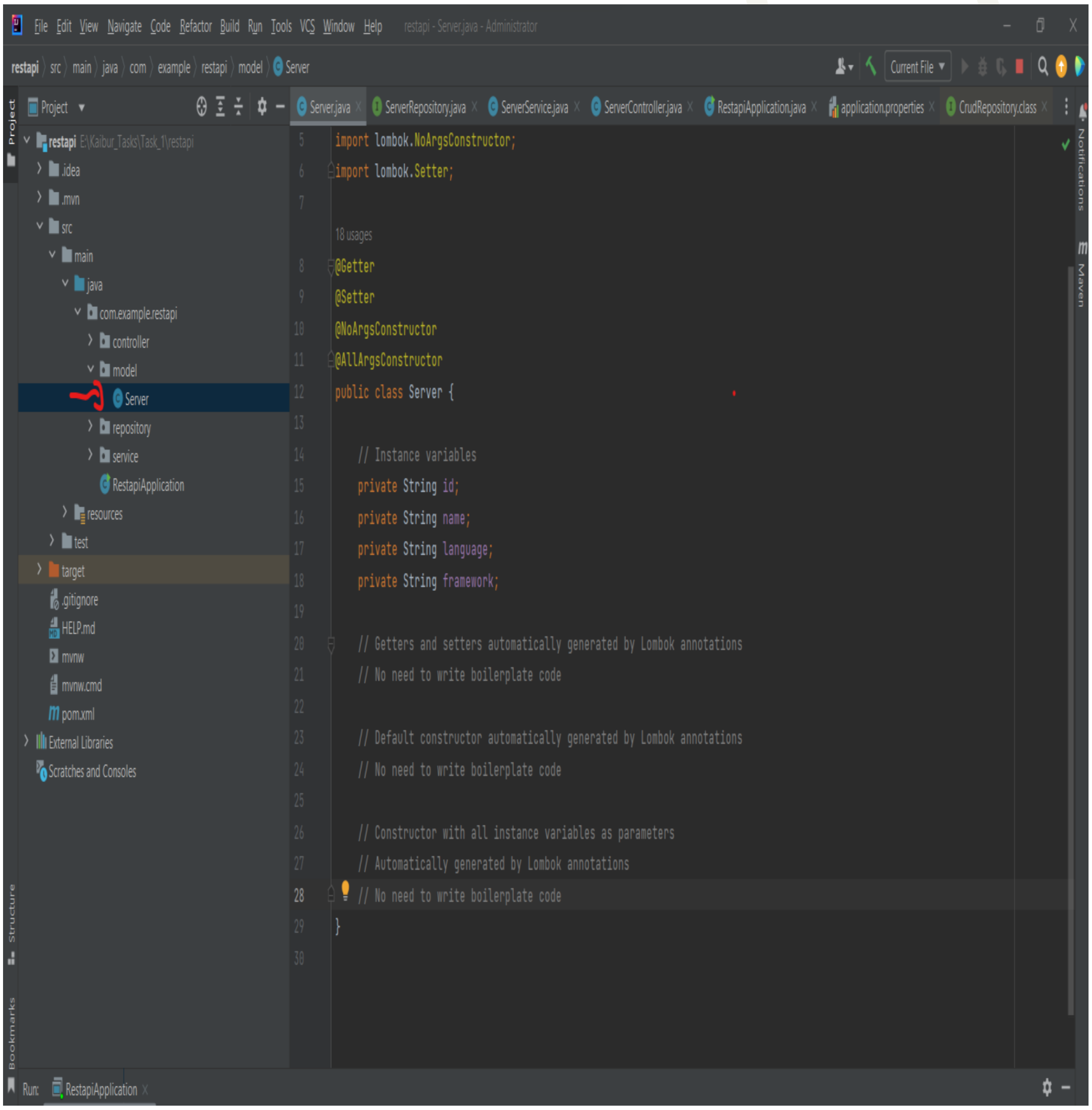
- ➢ Using MongoDB with Spring Boot allows for seamless integration of the database with the application, and provides features like automatic mapping of Java objects to MongoDB documents, support for querying, and the ability to easily scale the database by adding new nodes to the cluster. In this project, MongoDB is used to persist server data, and Spring Boot provides a simple API to interact with the database through the Repository and Service layers.

❖ **Spring.io (Spring boot Project)**:

- ➢ Spring Boot is a popular Java-based framework used to create standalone, production-grade applications. It provides an efficient way to develop a variety of applications, including web, mobile, and micro services.

- ➢ ***In this project***, we used Spring Boot to create a RESTful web service. Spring Boot's extensive integration with MongoDB allowed us to easily interact with the database without having to write a lot of boilerplate code. Additionally, Spring Boot's built-in support for various testing frameworks makes it easy to write and run tests for our application. Overall, Spring Boot provides a powerful and efficient way to develop web services and other applications in Java.

3. **Define the Server model**:
   a. Create a new package called "***model***" and define a new class called "***Server***". This class should have the following attributes:
      i. ***String id***
      ii. ***String name***
      iii. ***String language***
      iv. ***String framework***
   b. Add "***Lombok annotations***" to the class to generate getters, setters, and constructors automatically.



```java
import lombok.NoArgsConstructor;
import lombok.Setter;

18 usages
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Server {

    // Instance variables
    private String id;
    private String name;
    private String language;
    private String framework;

    // Getters and setters automatically generated by Lombok annotations
    // No need to write boilerplate code

    // Default constructor automatically generated by Lombok annotations
    // No need to write boilerplate code

    // Constructor with all instance variables as parameters
    // Automatically generated by Lombok annotations
    // No need to write boilerplate code
}
```

Explanation (Code):

1. Create a new package called "***model***" inside the "***src/main/java/com/example/restapi***" package.
2. Inside the "***model***" package, create a new Java class called "***Server***" with the above code:

❖ Here, we have used the following '***Lombok annotations***':

- @***Getter***: This generates getter methods for all attributes of the class.
- @***Setter***: This generates setter methods for all attributes of the class.
- @***NoArgsConstructor***: This generates a no-argument constructor for the class.
- @***AllArgsConstructor***: This generates a constructor that accepts arguments for all attributes of the class.

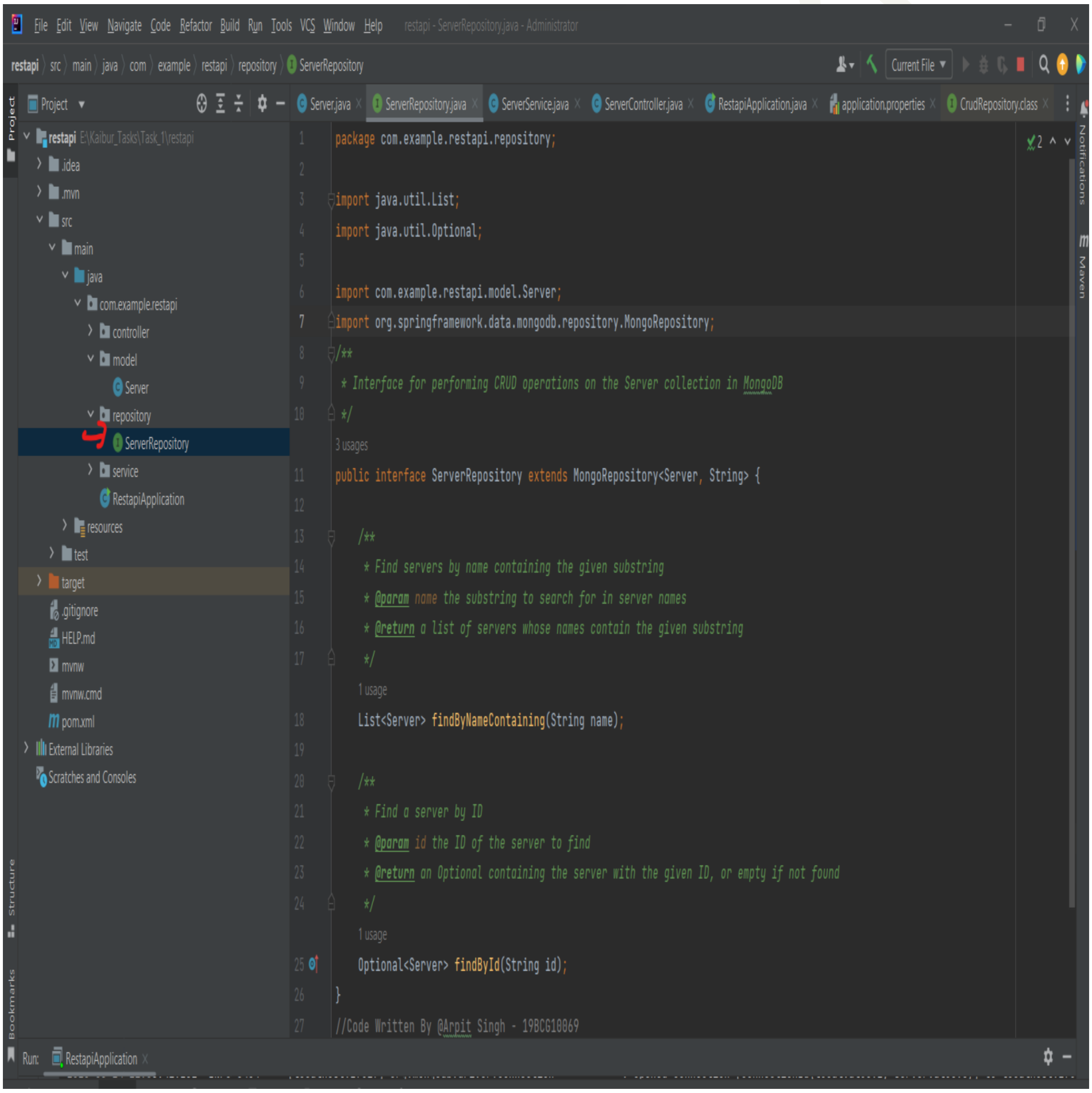
With these annotations, you don't need to write the boilerplate code for getter, setter, and constructor methods manually.

❖ The ***Server*** class has four attributes of type ***String***:

- ***id***: This represents the unique ID of the server.
- ***name***: This represents the name of the server.
- ***language***: This represents the programming language used by the server.
- ***framework***: This represents the web framework used by the server.

Now that we have defined the ***Server*** model class, we can proceed with the next steps to create the REST API endpoints.

4. **Define the Server repository**:
    a. Create a new package called "***repository***" and define a new interface called '***ServerRepository***'.
    b. This interface should extend '***MongoRepository<Server, String>***' and define the following methods:
        i. ***List<Server> findByNameContaining(String name)***
        ii. ***Optional<Server> findById(String id)***



```java
package com.example.restapi.repository;

import java.util.List;
import java.util.Optional;

import com.example.restapi.model.Server;
import org.springframework.data.mongodb.repository.MongoRepository;
/**
 * Interface for performing CRUD operations on the Server collection in MongoDB
 */
3 usages
public interface ServerRepository extends MongoRepository<Server, String> {

    /**
     * Find servers by name containing the given substring
     * @param name the substring to search for in server names
     * @return a list of servers whose names contain the given substring
     */
    1 usage
    List<Server> findByNameContaining(String name);

    /**
     * Find a server by ID
     * @param id the ID of the server to find
     * @return an Optional containing the server with the given ID, or empty if not found
     */
    1 usage
    Optional<Server> findById(String id);
}
//Code Written By @Arpit Singh - 19BCG10069
```

Explanation (Code):

3. Create a new package called "***repository***" inside the "***src/main/java/com/example/restapi***" package.
4. Inside the "***repository***" package, create a new Java Interface called "***ServerRepository***" with the above code:

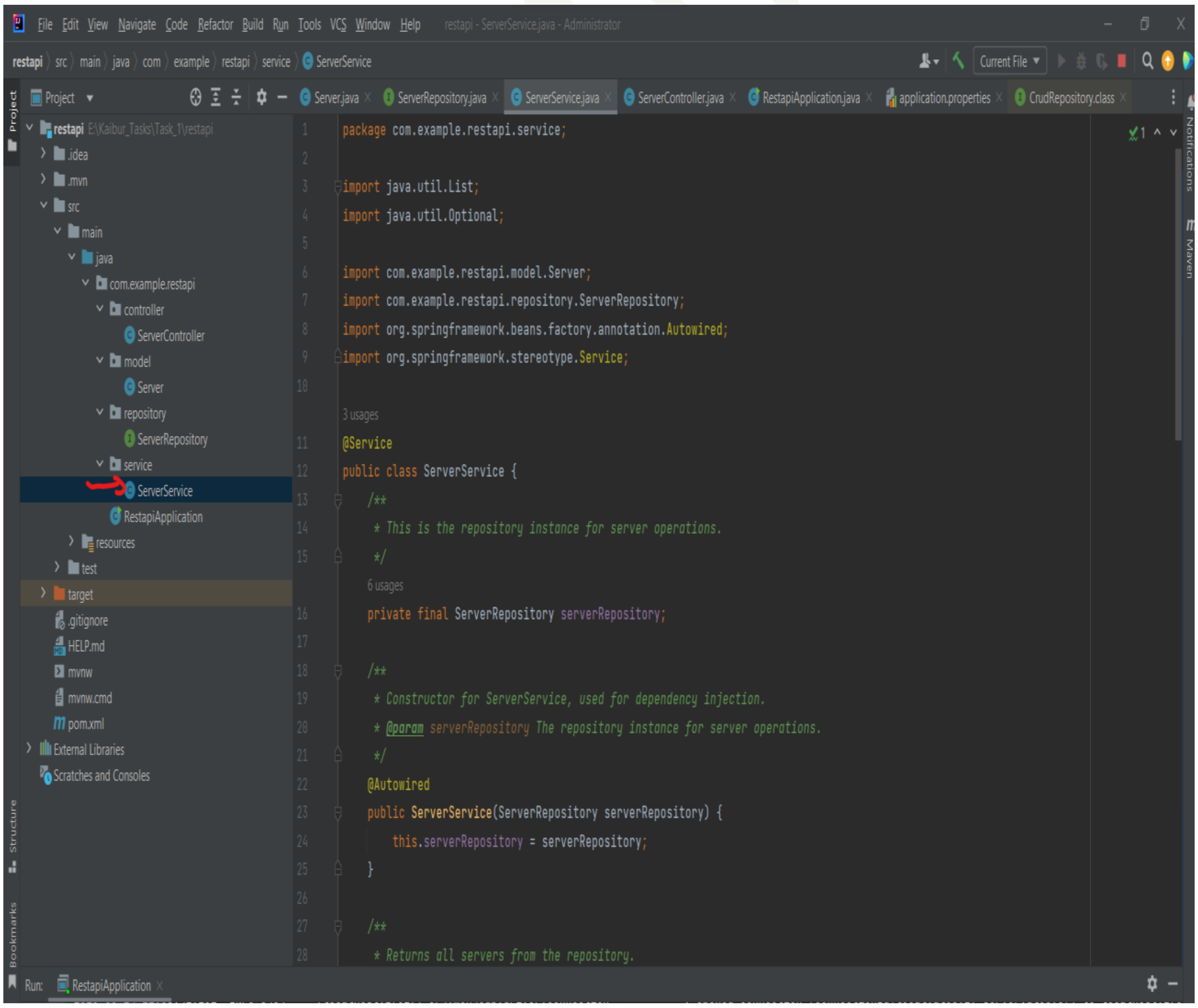
❖ Here, we have extended the ***MongoRepository*** interface provided by Spring Data MongoDB and added two additional methods:

- ***findByNameContaining(String name):*** This method searches for servers by name and returns a list of all servers whose names contain the given string.
- ***findById(String id):*** This method searches for a server by its unique ID and returns an ***Optional*** object that may or may not contain the server.

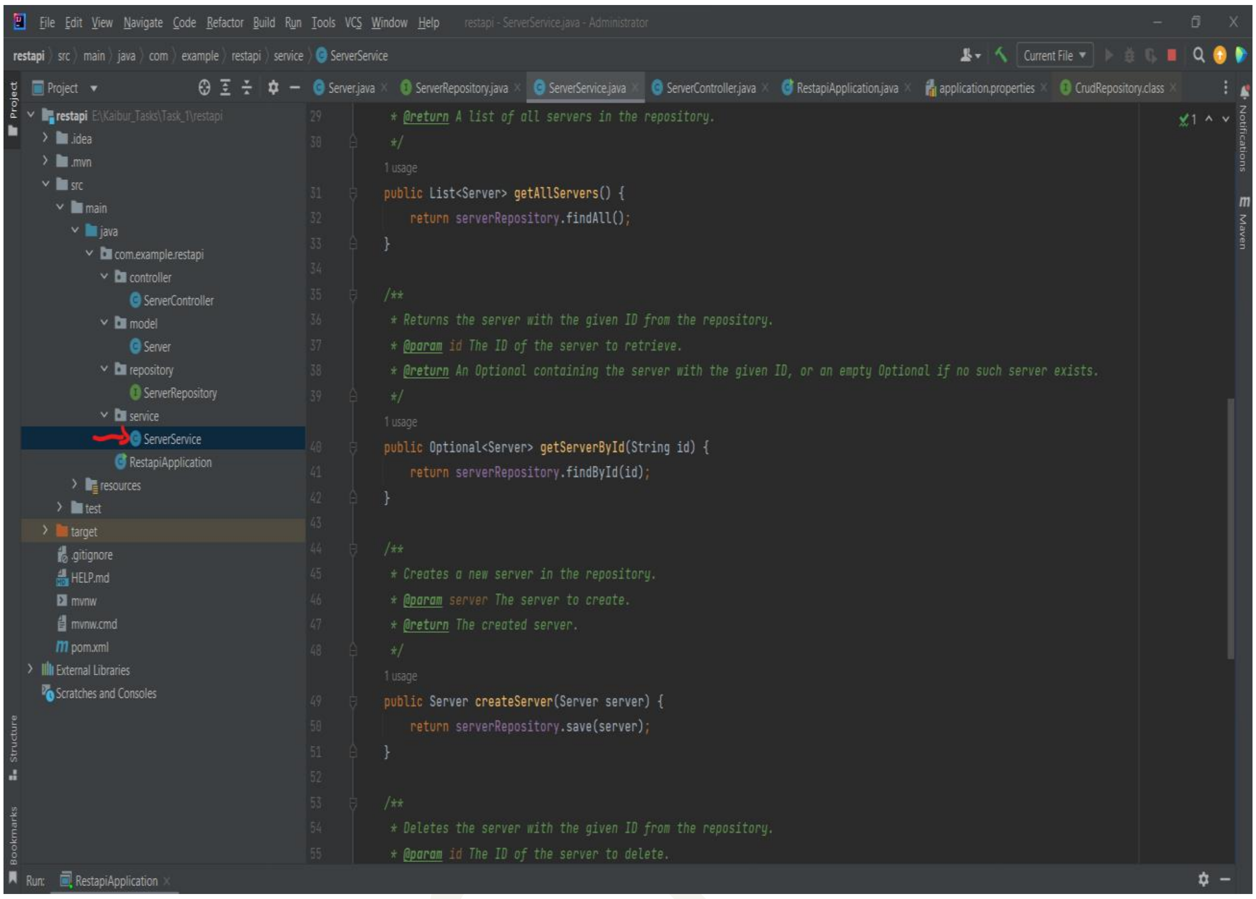
With these methods defined in the ***ServerRepository*** interface, you can now use them to perform CRUD operations on the ***servers*** collection in the MongoDB database. I will be using ***POSTMAN*** to make requests to the Sever (discussed later).

Now that we have defined the ***ServerRepository*** Interface, we can proceed with the next steps to create the Actual Functions to handle our requests.

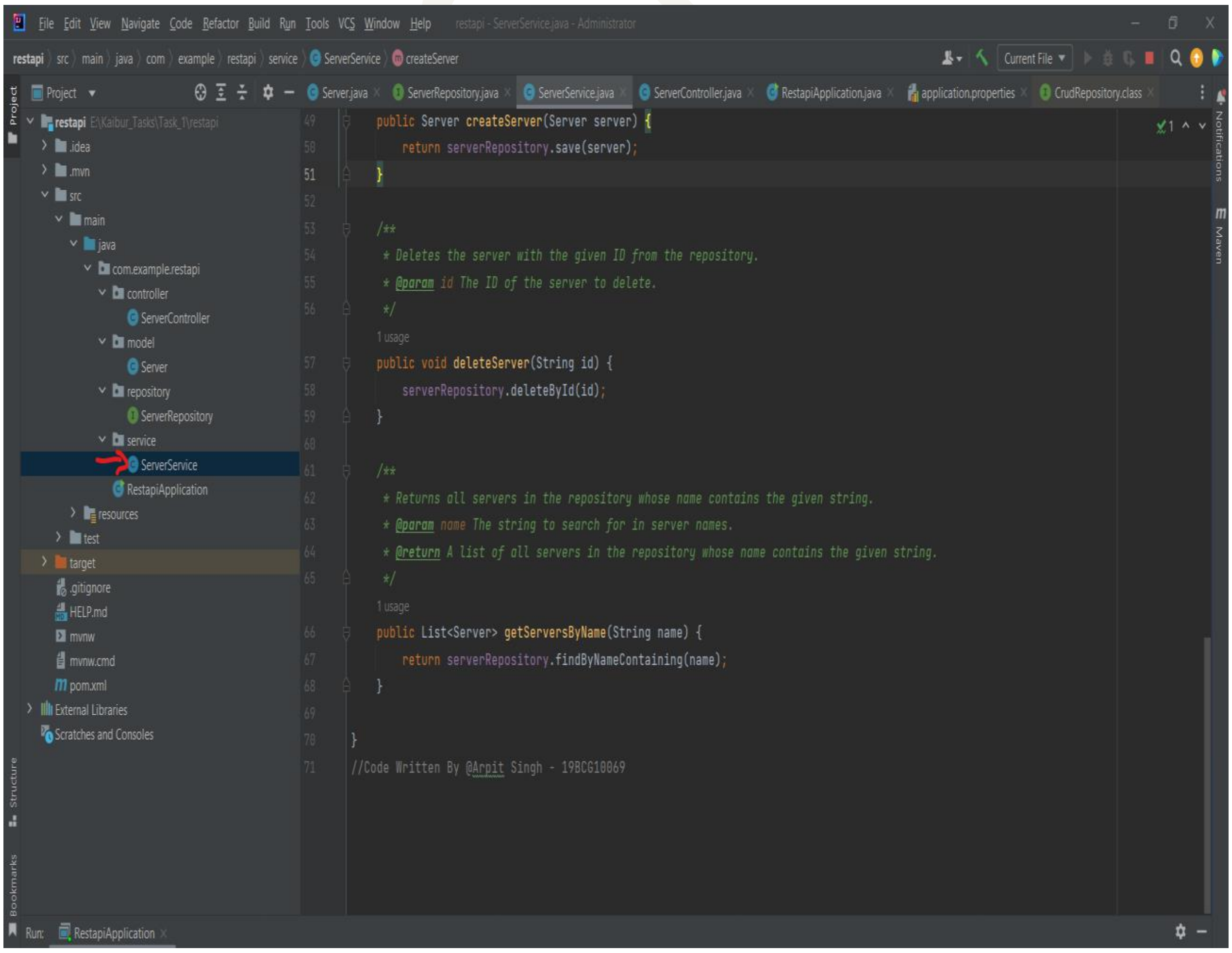
5. **Define the Server service**:
   a. Create a new package called “***service***” and define a new class called “***ServerService***”.
   b. This class should have a constructor that takes a ‘***ServerService’*** object as a parameter.
   c. Define the following methods:
      i. List<Server> getAllServers()
      ii. ResponseEntity<Server> getServerById(String id)
      iii. ResponseEntity<Server> createServer(@RequestBody Server server)
      iv. void deleteServer(String id)
      v. List<Server> getServersByName(@RequestParam("name") String name)
   d. In each method, call the corresponding method of the ‘***ServerRepository’ object*** and return the result**.**



```
package com.example.restapi.service;

import java.util.List;
import java.util.Optional;

import com.example.restapi.model.Server;
import com.example.restapi.repository.ServerRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

3 usages
@Service
public class ServerService {
    /**
     * This is the repository instance for server operations.
     */
    6 usages
    private final ServerRepository serverRepository;

    /**
     * Constructor for ServerService, used for dependency injection.
     * @param serverRepository The repository instance for server operations.
     */
    @Autowired
    public ServerService(ServerRepository serverRepository) {
        this.serverRepository = serverRepository;
    }

    /**
     * Returns all servers from the repository.
```



```
     * @return A list of all servers in the repository.
     */
    1 usage
    public List<Server> getAllServers() {
        return serverRepository.findAll();
    }

    /**
     * Returns the server with the given ID from the repository.
     * @param id The ID of the server to retrieve.
     * @return An Optional containing the server with the given ID, or an empty Optional if no such server exists.
     */
    1 usage
    public Optional<Server> getServerById(String id) {
        return serverRepository.findById(id);
    }

    /**
     * Creates a new server in the repository.
     * @param server The server to create.
     * @return The created server.
     */
    1 usage
    public Server createServer(Server server) {
        return serverRepository.save(server);
    }

    /**
     * Deletes the server with the given ID from the repository.
     * @param id The ID of the server to delete.
```



```
    public Server createServer(Server server) {
        return serverRepository.save(server);
    }

    /**
     * Deletes the server with the given ID from the repository.
     * @param id The ID of the server to delete.
     */
    1 usage
    public void deleteServer(String id) {
        serverRepository.deleteById(id);
    }

    /**
     * Returns all servers in the repository whose name contains the given string.
     * @param name The string to search for in server names.
     * @return A list of all servers in the repository whose name contains the given string.
     */
    1 usage
    public List<Server> getServersByName(String name) {
        return serverRepository.findByNameContaining(name);
    }

}
//Code Written By @Arpit Singh - 19BCG10069
```

Explanation (Code):

5. Create a new package called "***service***" inside the "***src/main/java/com/example/restapi***" package.

6. Inside the "***service***" package, create a new Java class called "***ServerService***" with the above code:

❖ Here, we have defined a ServerService class with the following methods:

- ***getAllServers()***: This method calls the ***findAll()*** method of the ***ServerRepository*** object to retrieve a list of all servers from the MongoDB database.

- ***getServerById(String id):*** This method calls the ***findById(String id)*** method of the ***ServerRepository*** object to retrieve a server with the given ID from the MongoDB database. If the server is found, it is returned as an Optional object; otherwise, an empty Optional object is returned.

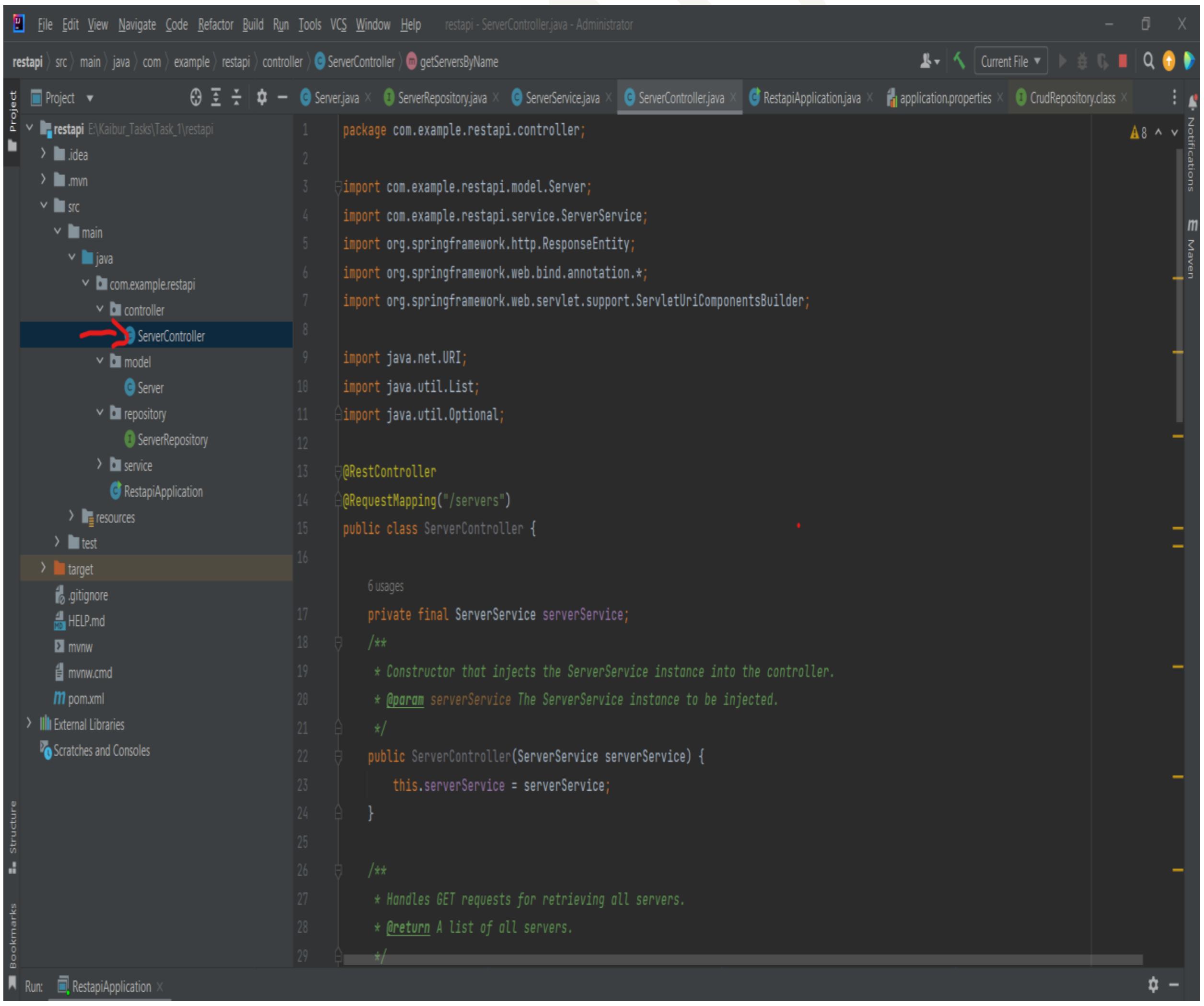
- ***createServer(Server server):*** This method calls the **save(Server server)** method of the ***ServerRepository*** object to add a new server to the MongoDB database.

- ***deleteServer(String id):*** This method calls the ***deleteById(String id)*** method of the ServerRepository object to delete a server with the given ID from the MongoDB database.

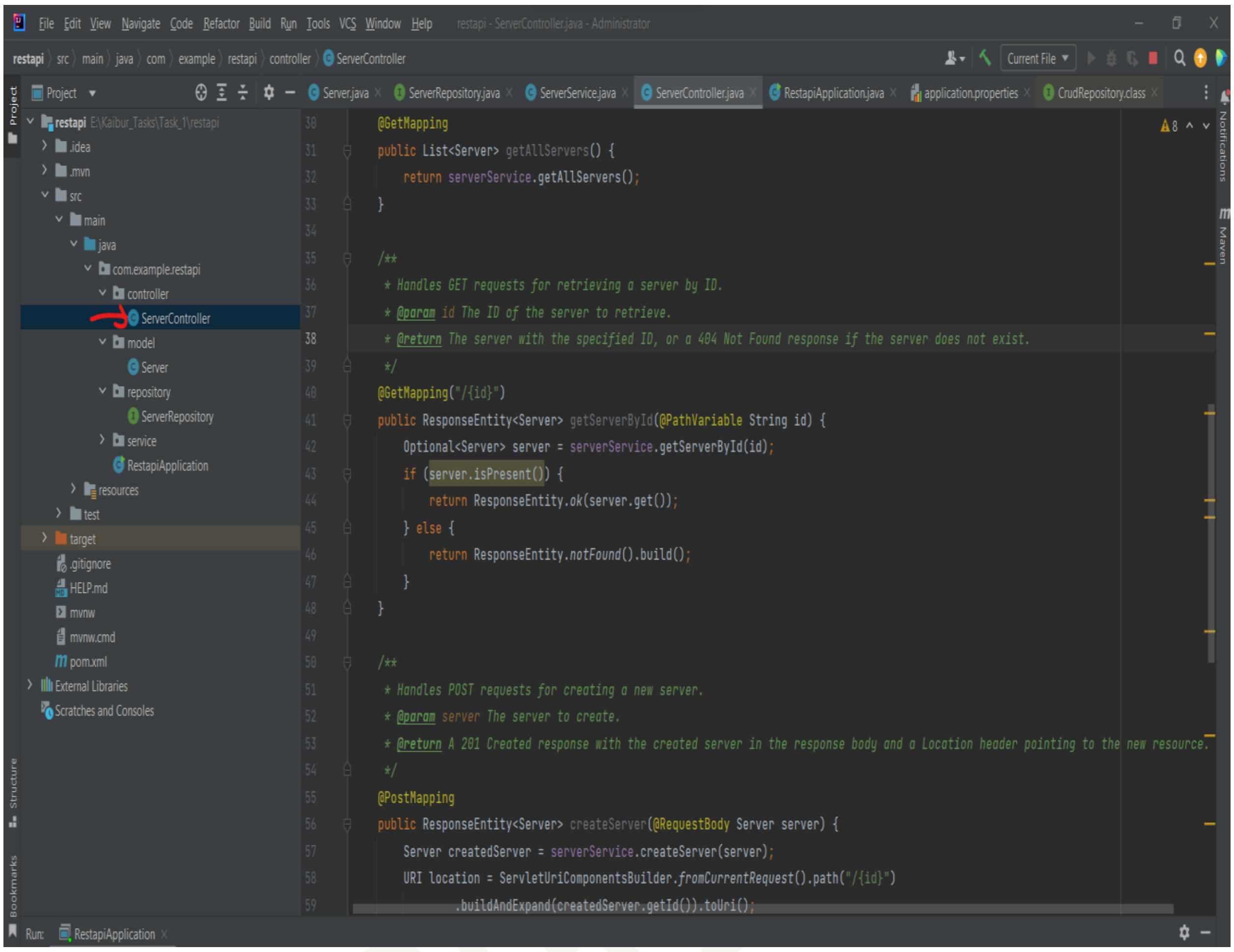
- ***getServersByName(String name):*** This method calls the ***findByNameContaining(String name)*** method of the ***ServerRepository*** object to retrieve a list of all servers whose names contain the given string from the MongoDB database.

Note that the @Service annotation is used to mark the ***ServerService*** class as a Spring service component, which can be injected into other Spring components using dependency injection. The @***Autowired*** annotation is used to inject the ServerRepository object into the ServerService constructor.

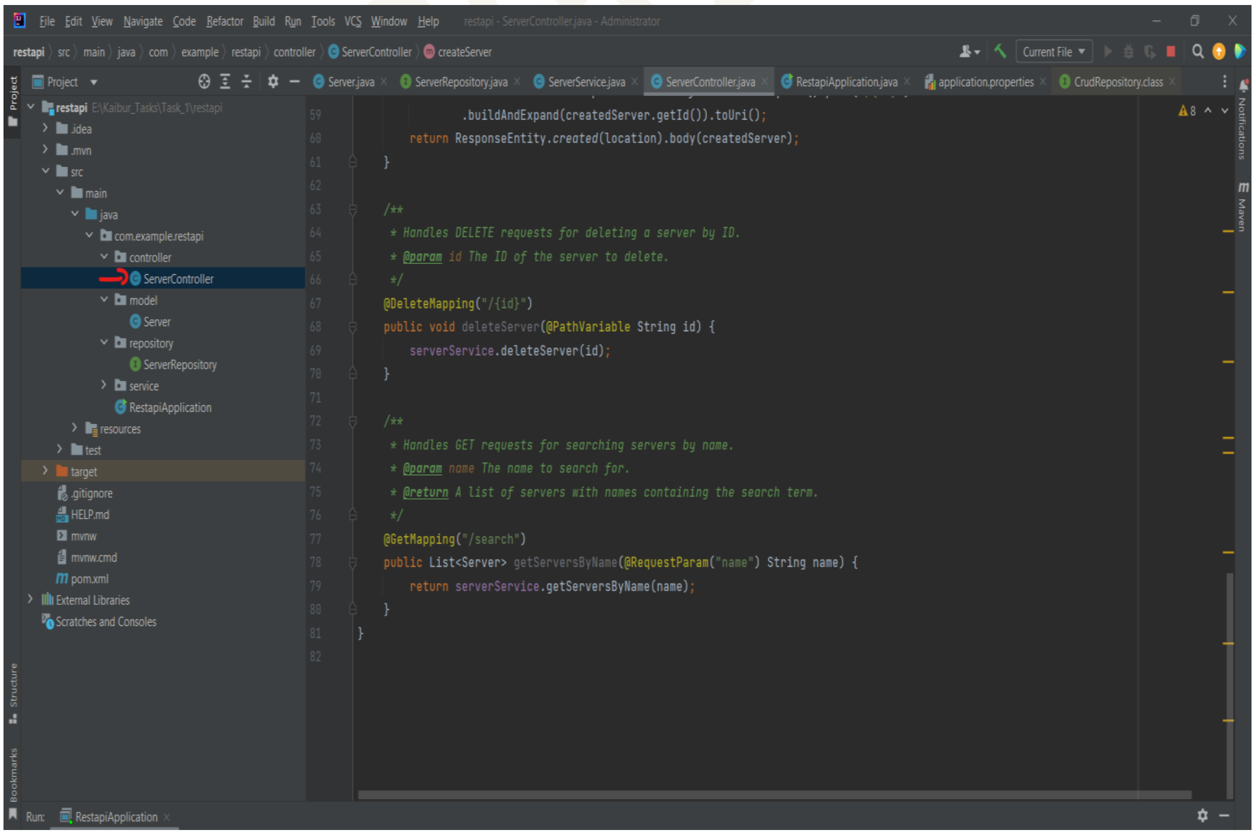
6. **Define the Server controller:**
   a. Create a new package called "***controller***" and define a new class called "***ServerController***".
   b. This class should have a constructor that takes a '***ServerService'*** object as a parameter.
   c. Define the following methods:
      i. ***List<Server> getAllServers()***
      ii. ***ResponseEntity<Server> getServerById(String id)***
      iii. ***ResponseEntity<Server> createServer(@RequestBody Server server)***
      iv. ***void deleteServer(String id)***
      v. ***List<Server> getServersByName(@RequestParam("name") String name)***
   d. In each method, call the corresponding method of the '***ServerService' object*** and return the result**.**



```java
package com.example.restapi.controller;

import com.example.restapi.model.Server;
import com.example.restapi.service.ServerService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

import java.net.URI;
import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/servers")
public class ServerController {

    6 usages
    private final ServerService serverService;
    /**
     * Constructor that injects the ServerService instance into the controller.
     * @param serverService The ServerService instance to be injected.
     */
    public ServerController(ServerService serverService) {
        this.serverService = serverService;
    }

    /**
     * Handles GET requests for retrieving all servers.
     * @return A list of all servers.
     */
```



```java
    @GetMapping
    public List<Server> getAllServers() {
        return serverService.getAllServers();
    }

    /**
     * Handles GET requests for retrieving a server by ID.
     * @param id The ID of the server to retrieve.
     * @return The server with the specified ID, or a 404 Not Found response if the server does not exist.
     */
    @GetMapping("/{id}")
    public ResponseEntity<Server> getServerById(@PathVariable String id) {
        Optional<Server> server = serverService.getServerById(id);
        if (server.isPresent()) {
            return ResponseEntity.ok(server.get());
        } else {
            return ResponseEntity.notFound().build();
        }
    }

    /**
     * Handles POST requests for creating a new server.
     * @param server The server to create.
     * @return A 201 Created response with the created server in the response body and a Location header pointing to the new resource.
     */
    @PostMapping
    public ResponseEntity<Server> createServer(@RequestBody Server server) {
        Server createdServer = serverService.createServer(server);
        URI location = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}")
                .buildAndExpand(createdServer.getId()).toUri();
```



```java
                .buildAndExpand(createdServer.getId()).toUri();
        return ResponseEntity.created(location).body(createdServer);
    }

    /**
     * Handles DELETE requests for deleting a server by ID.
     * @param id The ID of the server to delete.
     */
    @DeleteMapping("/{id}")
    public void deleteServer(@PathVariable String id) {
        serverService.deleteServer(id);
    }

    /**
     * Handles GET requests for searching servers by name.
     * @param name The name to search for.
     * @return A list of servers with names containing the search term.
     */
    @GetMapping("/search")
    public List<Server> getServersByName(@RequestParam("name") String name) {
        return serverService.getServersByName(name);
    }
}
```

Explanation (Code):

7. Create a new package called "***controller***" inside the "***src/main/java/com/example/restapi***" package.

8. Inside the "***controller***" package, create a new Java class called "***ServerController***" with the above code:

❖ ***Explanation***:

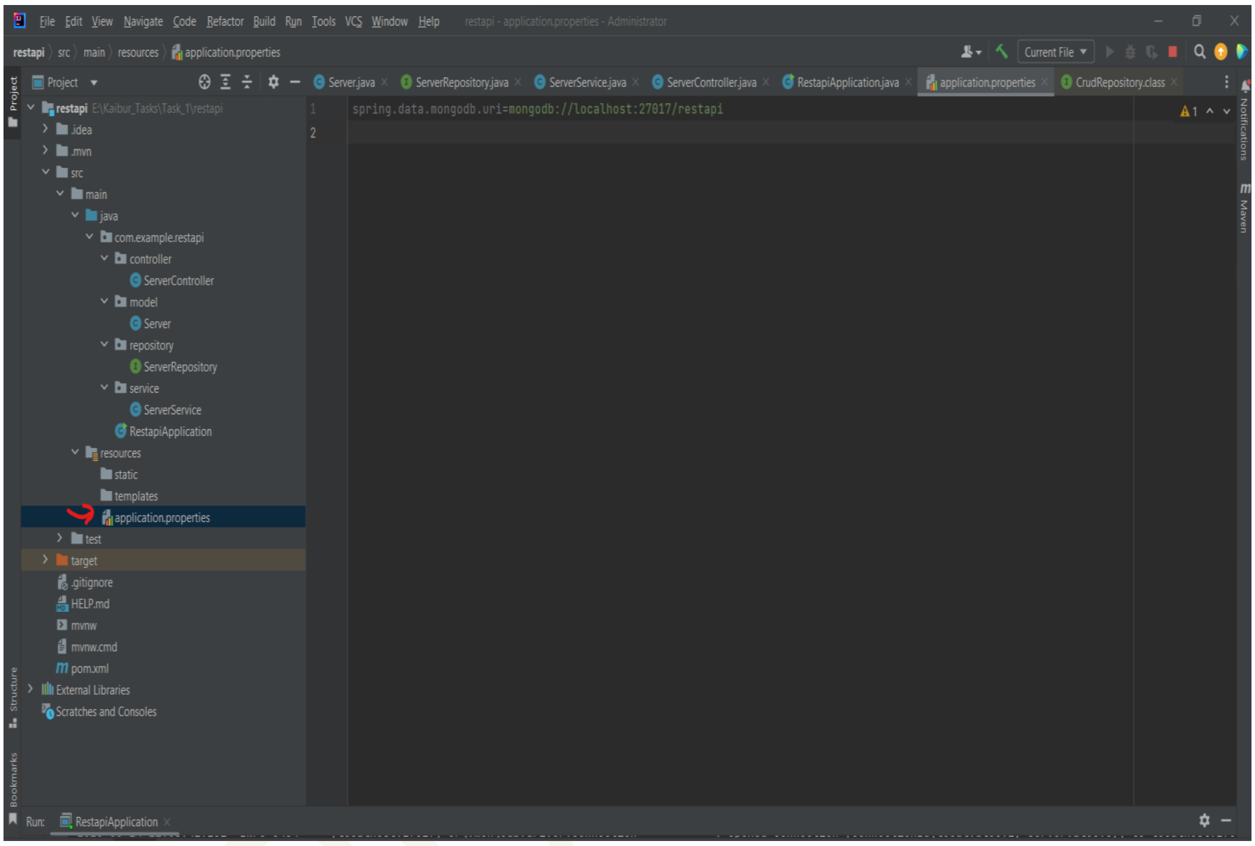
- The @***RestController*** annotation indicates that this class is a REST controller.
- The @***RequestMapping***("/servers") annotation sets the base URI for all methods in this class.
- The ServerService object is injected into the constructor using the @***Autowired*** annotation.
- The ***getAllServers***() method calls the ***getAllServers***() method of ServerService.
- The ***getServerById***() method calls the ***getServerById***() method of ServerService. If the server exists, it returns a **200 OK** response with the server object. Otherwise, it returns a **404 Not** Found response.
- The ***createServer***() method calls the ***createServer***() method of ServerService. It returns a **201 Created** response with the location of the newly created server object.
- The ***deleteServer***() method calls the ***deleteServer***() method of ServerService.
- The ***getServersByName***() method calls the ***getServersByName***() method of ***ServerService***. It accepts a query parameter name and returns a list of server objects that contain the given name.

7. **Configure MongoDB**:

   a. Open the "***application.properties***" file located in the "***src/main/resources***" directory.

   b. In the "***application.properties***" file, add the following configuration:

      i. ***'spring.data.mongodb.uri=mongodb://localhost:27017/restapi'***



```
spring.data.mongodb.uri=mongodb://localhost:27017/restapi
```

Explanation (Code):

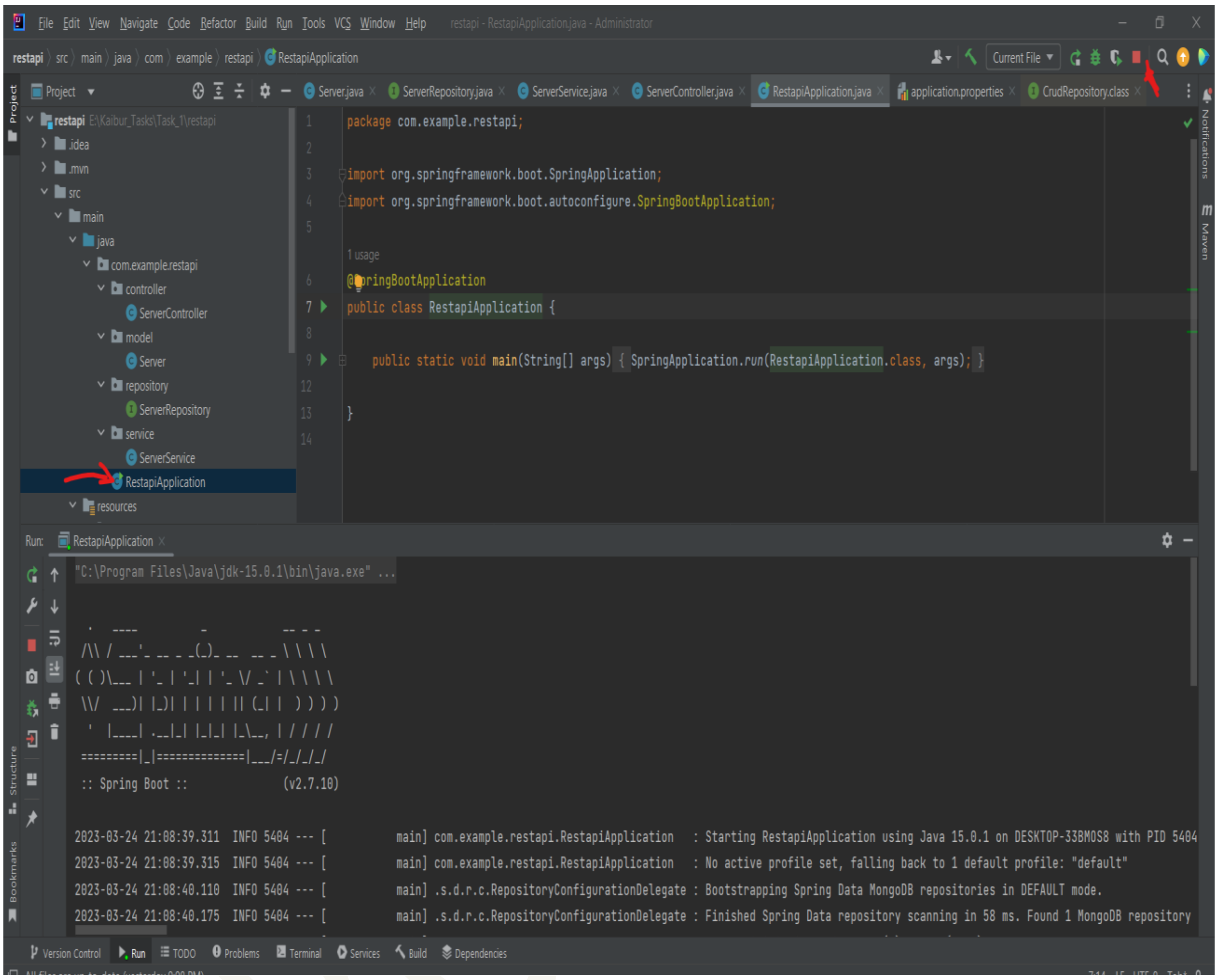
9. Open the "***application.properties***" file located in the src/main/resources directory.

10. Add the following "***configuration***" to the file:

    - `'spring.data.mongodb.uri=mongodb://localhost:27017/restapi'`
    - This configuration *sets the MongoDB URI* to "***mongodb://localhost:27017/restapi***", where '***localhost:27017***' is the default MongoDB ***host and port***, and ***restapi*** is the name of the ***database*** we'll be using for our application.
    - ***Save the "application.properties" file***.

8. **Run the application**:

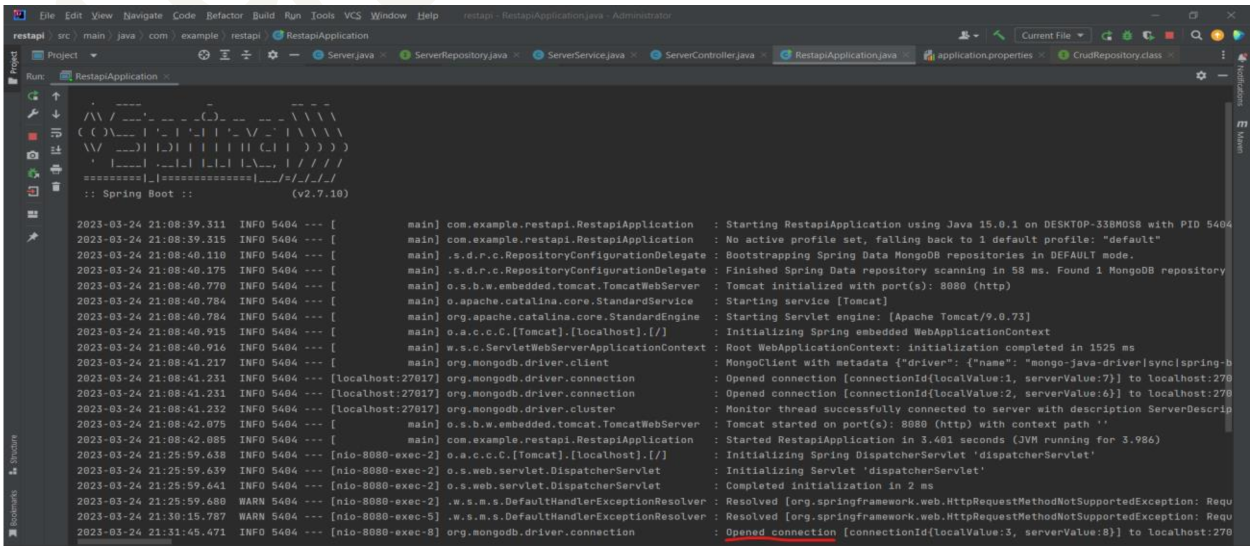
   a. Run the application by executing the "***RestApiApplication***" class.



```java
package com.example.restapi;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

1 usage
@SpringBootApplication
public class RestapiApplication {

    public static void main(String[] args) { SpringApplication.run(RestapiApplication.class, args); }

}
```

```
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe" ...

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::               (v2.7.10)

2023-03-24 21:08:39.311  INFO 5404 --- [           main] com.example.restapi.RestapiApplication   : Starting RestapiApplication using Java 15.0.1 on DESKTOP-33BMOS8 with PID 5404
2023-03-24 21:08:39.315  INFO 5404 --- [           main] com.example.restapi.RestapiApplication   : No active profile set, falling back to 1 default profile: "default"
2023-03-24 21:08:40.110  INFO 5404 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data MongoDB repositories in DEFAULT mode.
2023-03-24 21:08:40.175  INFO 5404 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 58 ms. Found 1 MongoDB repository
```

➢ The ***console log(s)*** will finish with below means the server has successfully started.



```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::               (v2.7.10)

2023-03-24 21:08:39.311  INFO 5404 --- [           main] com.example.restapi.RestapiApplication   : Starting RestapiApplication using Java 15.0.1 on DESKTOP-33BMOS8 with PID 5404
2023-03-24 21:08:39.315  INFO 5404 --- [           main] com.example.restapi.RestapiApplication   : No active profile set, falling back to 1 default profile: "default"
2023-03-24 21:08:40.110  INFO 5404 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data MongoDB repositories in DEFAULT mode.
2023-03-24 21:08:40.175  INFO 5404 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 58 ms. Found 1 MongoDB repository
2023-03-24 21:08:40.770  INFO 5404 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2023-03-24 21:08:40.784  INFO 5404 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2023-03-24 21:08:40.784  INFO 5404 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.73]
2023-03-24 21:08:40.915  INFO 5404 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2023-03-24 21:08:40.916  INFO 5404 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1525 ms
2023-03-24 21:08:41.217  INFO 5404 --- [           main] org.mongodb.driver.client                : MongoClient with metadata {"driver": {"name": "mongo-java-driver|sync|spring-b
2023-03-24 21:08:41.231  INFO 5404 --- [localhost:27017] org.mongodb.driver.connection            : Opened connection [connectionId{localValue:1, serverValue:7}] to localhost:270
2023-03-24 21:08:41.231  INFO 5404 --- [localhost:27017] org.mongodb.driver.connection            : Opened connection [connectionId{localValue:2, serverValue:6}] to localhost:270
2023-03-24 21:08:41.232  INFO 5404 --- [localhost:27017] org.mongodb.driver.cluster               : Monitor thread successfully connected to server with description ServerDescrip
2023-03-24 21:08:42.075  INFO 5404 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2023-03-24 21:08:42.085  INFO 5404 --- [           main] com.example.restapi.RestapiApplication   : Started RestapiApplication in 3.401 seconds (JVM running for 3.986)
2023-03-24 21:25:59.638  INFO 5404 --- [nio-8080-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-03-24 21:25:59.639  INFO 5404 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2023-03-24 21:25:59.641  INFO 5404 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet        : Completed initialization in 2 ms
2023-03-24 21:25:59.680  WARN 5404 --- [nio-8080-exec-2] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved [org.springframework.web.HttpRequestMethodNotSupportedException: Requ
2023-03-24 21:30:15.787  WARN 5404 --- [nio-8080-exec-5] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved [org.springframework.web.HttpRequestMethodNotSupportedException: Requ
2023-03-24 21:31:45.471  INFO 5404 --- [nio-8080-exec-8] org.mongodb.driver.connection            : Opened connection [connectionId{localValue:3, serverValue:8}] to localhost:270
```

Explanation (Code):

❖ ***To run the application***, you can follow the below steps:

1. Open the '***RestApiApplication'*** class.

2. Right-click on the RestApiApplication class and click on "***Run As***" and select "***Java Application***" option.

3. Wait for the application to start. You should see some ***logs in the console window*** indicating that the application has started successfully.

4. Open a "***web browser or a tool like Postman***", and enter the following '**URL**' to test the **API endpoints**:
   - ***To get all servers: http://localhost:8080/servers***
   - ***To get a single server by ID: http://localhost:8080/servers/{id}***
   - ***To create a new server: POST http://localhost:8080/servers***
   - ***To delete a server by ID: DELETE http://localhost:8080/servers/{id}***
   - ***To search for servers by name: http://localhost:8080/servers/search?name={name}***

5. Make sure to replace ***{id}*** and ***{name}*** with the actual ***search values***.

❖ **That's it! You have successfully implemented a Java REST API with endpoints for searching, creating, and deleting server objects using Spring Boot and MongoDB.**

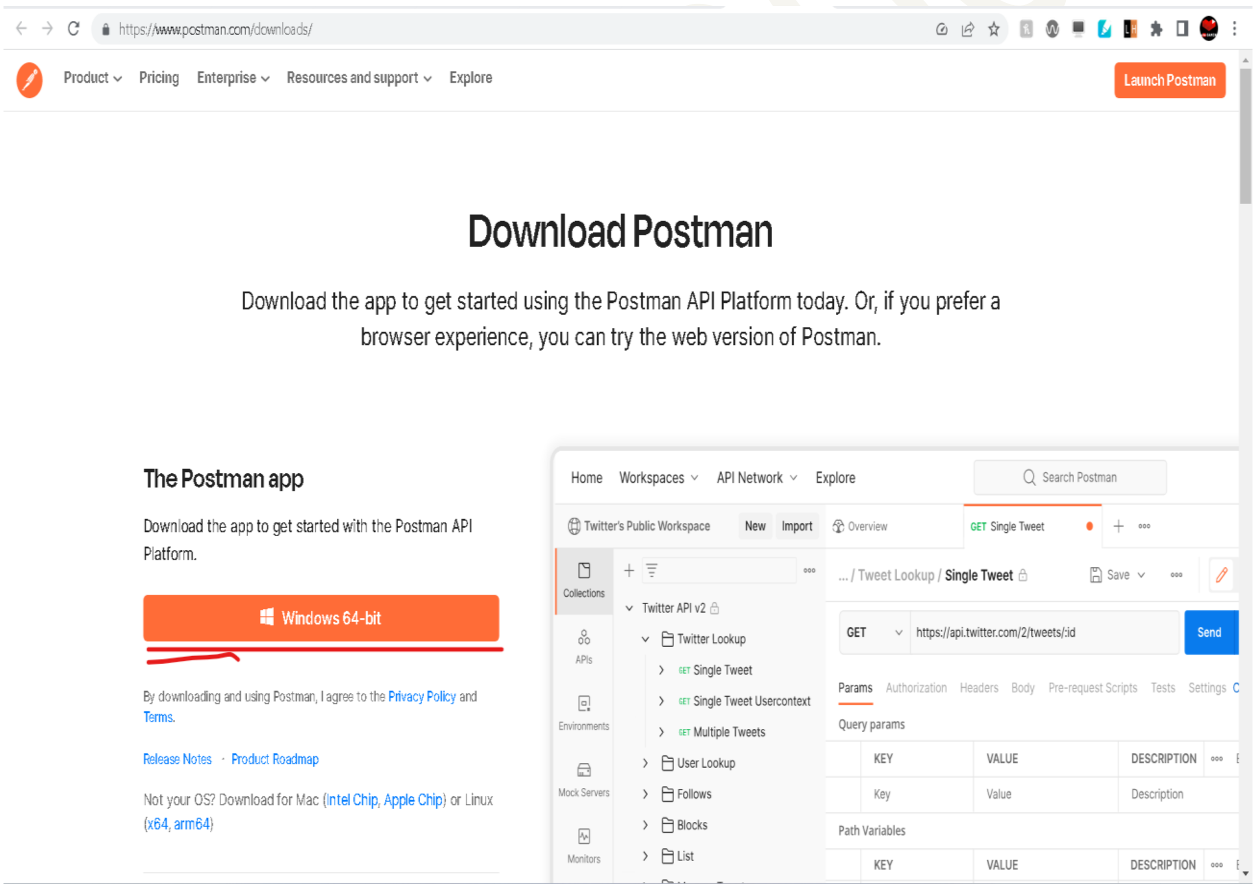
API Testing/Output:

- Here, I have used "***POSTMAN***" to make 'search/create/delete' request to the API.
- Also I have shown the output on a "***browser***". (Google Chrome)

❖ How To Use POSTMAN:

- ***Postman*** is a popular ***HTTP client*** that allows you to send HTTP requests and test APIs.

- Here are the steps to use Postman to test the API:
  1. Download and install Postman from **https://www.postman.com/downloads/**.



Download Postman

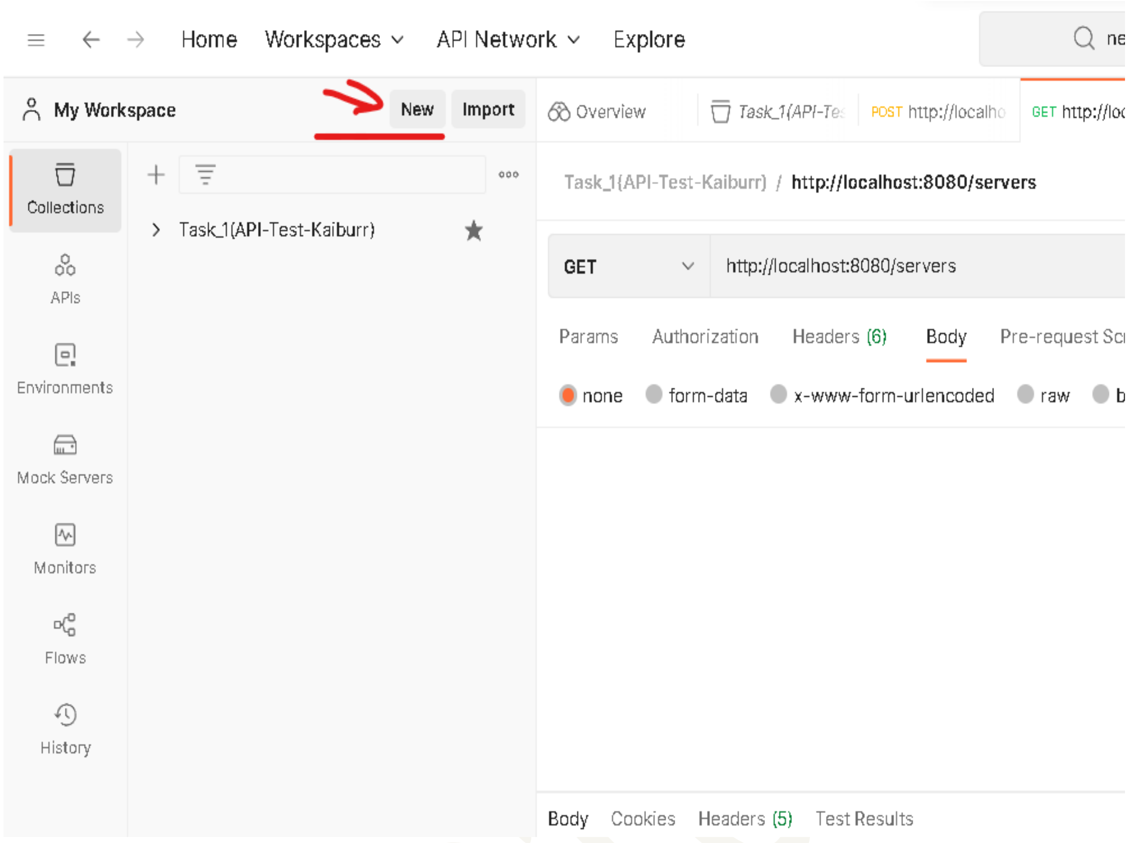
Download the app to get started using the Postman API Platform today. Or, if you prefer a browser experience, you can try the web version of Postman.

Download the app to get started with the Postman API Platform.

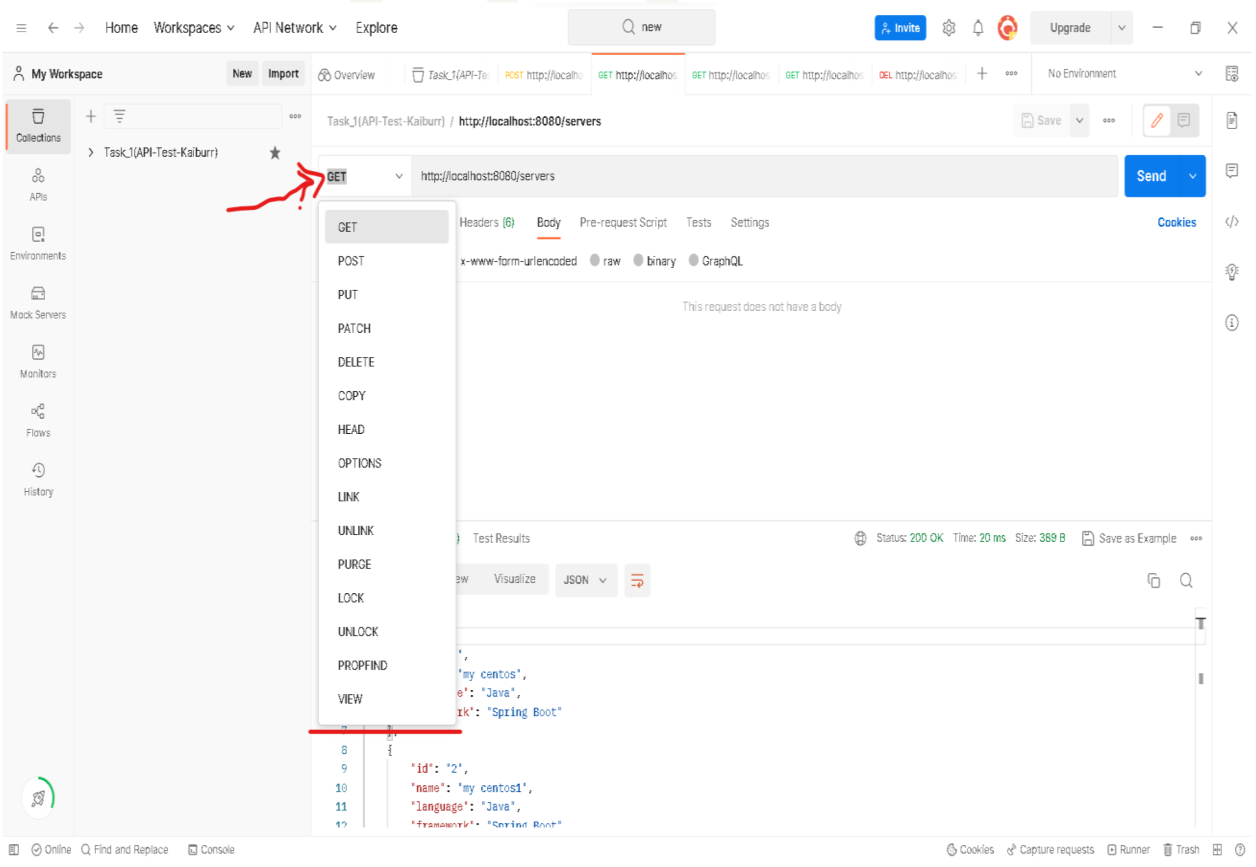
By downloading and using Postman, I agree to the Privacy Policy and Terms.

Not your OS? Download for Mac (Intel Chip, Apple Chip) or Linux (x64, arm64)

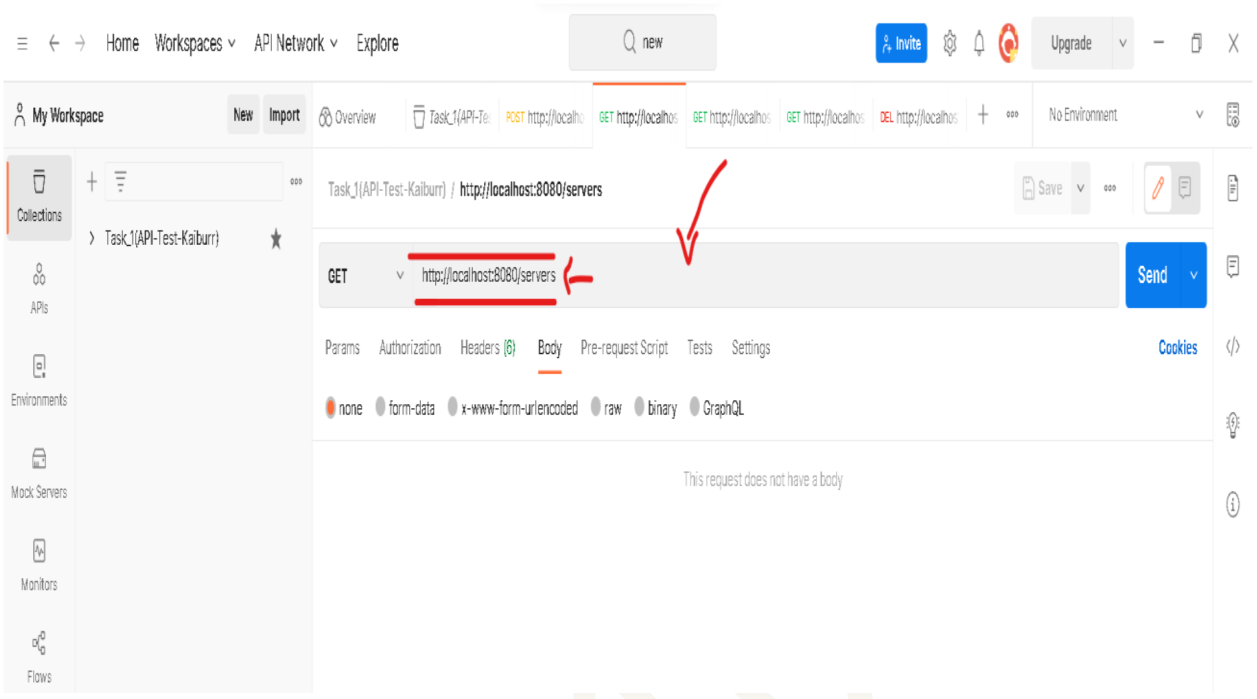
  2. Open Postman and click on the "***New***" button to create a new request.



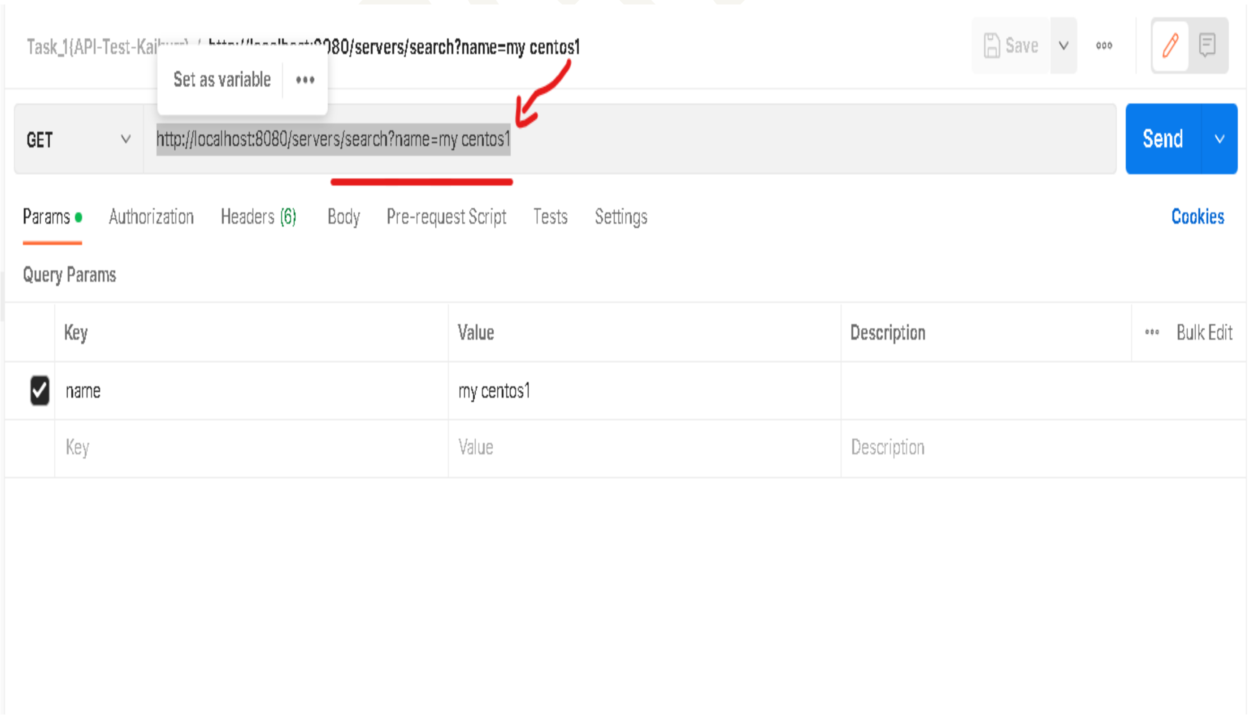
3. Select the ***HTTP method*** you want to use (GET, PUT, DELETE) from the dropdown list.



4. Enter the ***URL of the API endpoint*** you want to test (e.g. ***http://localhost:8080/servers***).

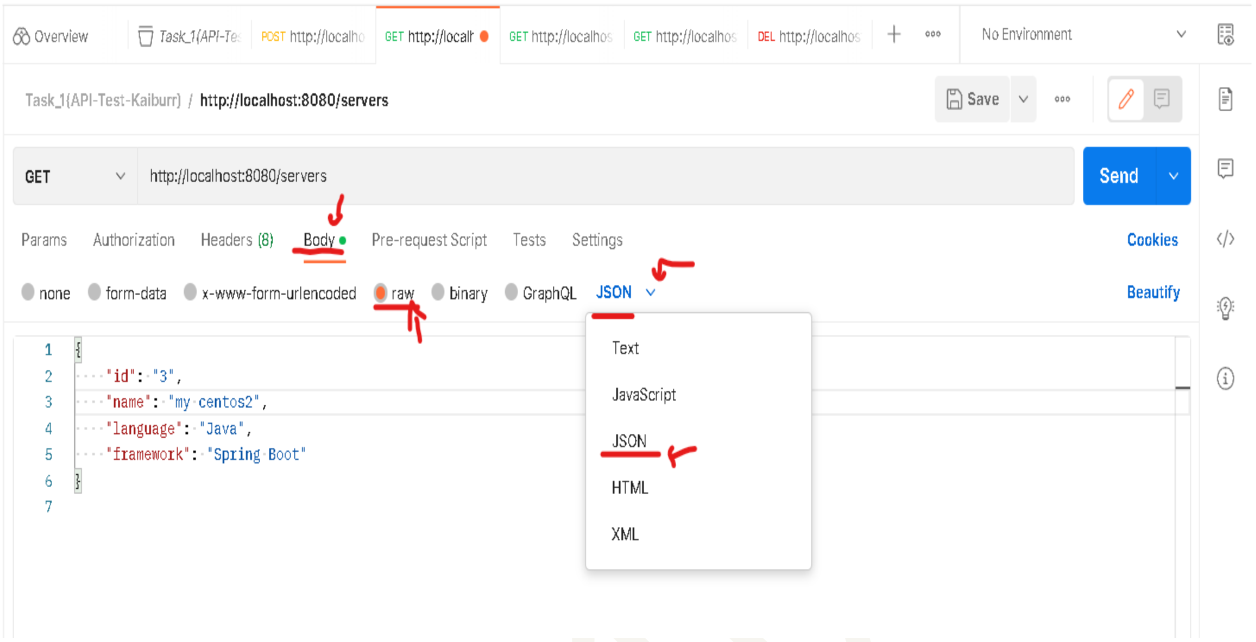


5. If you need ***to include parameters*** in your request (e.g. a server ID), you can ***add them to the URL*** or in the request body.


Task_1(API-Test-Kaiburr) / http://localhost:8080/servers/search?name=my centos1
Set as variable
GET
http://localhost:8080/servers/search?name=my centos1
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params

	Key	Value	Description
✓	name	my centos1	
	Key	Value	Description

6. If you need to include a ***request body*** (e.g. for creating a new server), click on the "***Body***" tab and select "***raw***". Then enter the "***JSON data***" in the text area.



Task_1(API-Test-Kaiburr) / http://localhost:8080/servers

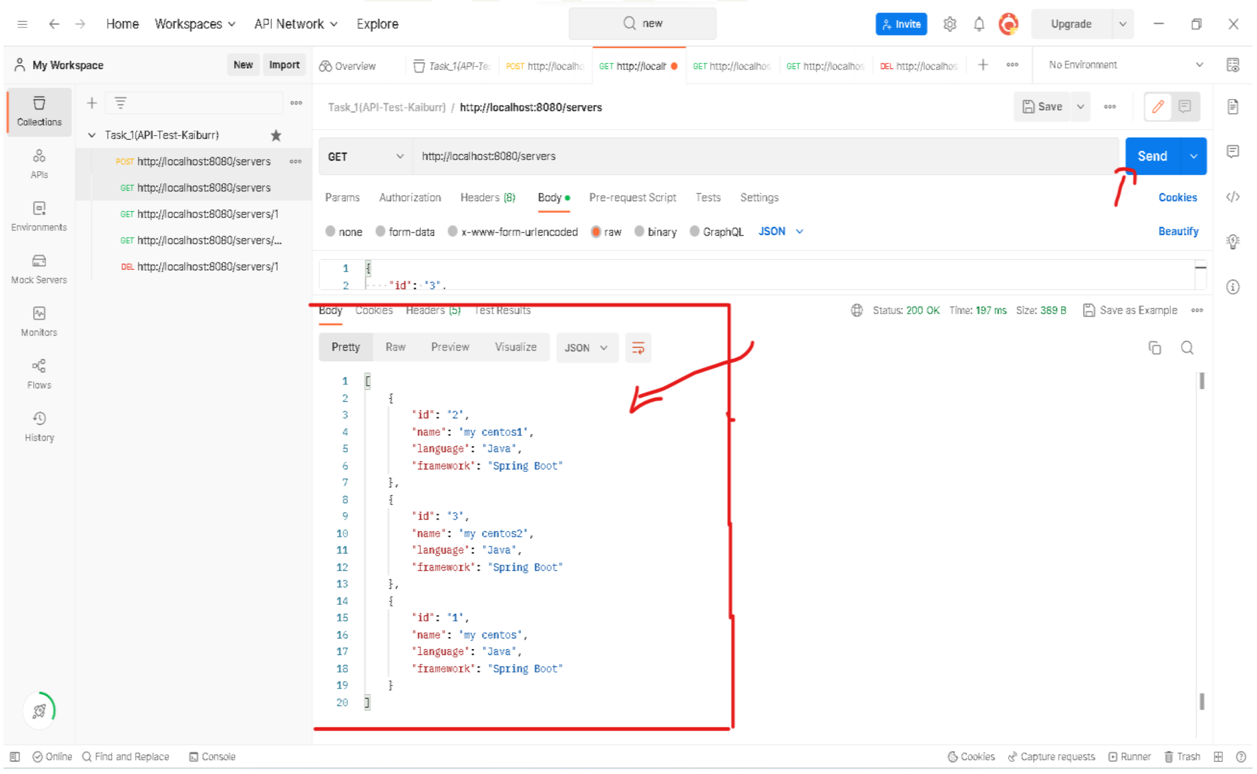
GET http://localhost:8080/servers

Params · Authorization · Headers (8) · Body · Pre-request Script · Tests · Settings

none · form-data · x-www-form-urlencoded · [✓ raw] · binary · GraphQL · JSON

```
{
    "id": "3",
    "name": "my centos2",
    "language": "Java",
    "framework": "Spring Boot"
}
```

7. Click the "***Send***" button to send the request.
8. You will see the response in the "***Response***" tab.



My Workspace · Task_1(API-Test-Kaiburr)

- POST http://localhost:8080/servers
- GET http://localhost:8080/servers
- GET http://localhost:8080/servers/1
- GET http://localhost:8080/servers/...
- DEL http://localhost:8080/servers/1

GET http://localhost:8080/servers

Status: 200 OK Time: 197 ms Size: 389 B

```
[
    {
        "id": "2",
        "name": "my centos1",
        "language": "Java",
        "framework": "Spring Boot"
    },
    {
        "id": "3",
        "name": "my centos2",
        "language": "Java",
        "framework": "Spring Boot"
    },
    {
        "id": "1",
        "name": "my centos",
        "language": "Java",
        "framework": "Spring Boot"
    }
]
```

➢ ***Repeat these steps*** for each API endpoint you want to test.

❖ API Testing/Output:

- To get a ***single server by ID***, you can send a ***GET*** request to ***http://localhost:8080/servers/{id}***, where {id} is the ID of the server you want to retrieve. Ex: ***http://localhost:8080/servers/1.***

- To ***get all servers***, you can send a ***GET*** request to ***http://localhost:8080/servers.***

- To ***delete a server by ID***, you can send a ***DELETE*** request to ***http://localhost:8080/servers/{id}***, where {id} is the ID of the server you want to delete. Ex: ***http://localhost:8080/servers/2.***

- To ***search servers by name***, you can send a ***GET*** request to ***http://localhost:8080/servers/search?name={name}***, where {name} is the name of the server you want to search for. Ex: ***http://localhost:8080/servers/search?name=my centos1.***

❖ Note that ***these "endpoints" correspond to the methods*** you defined in the ***ServerController*** class:

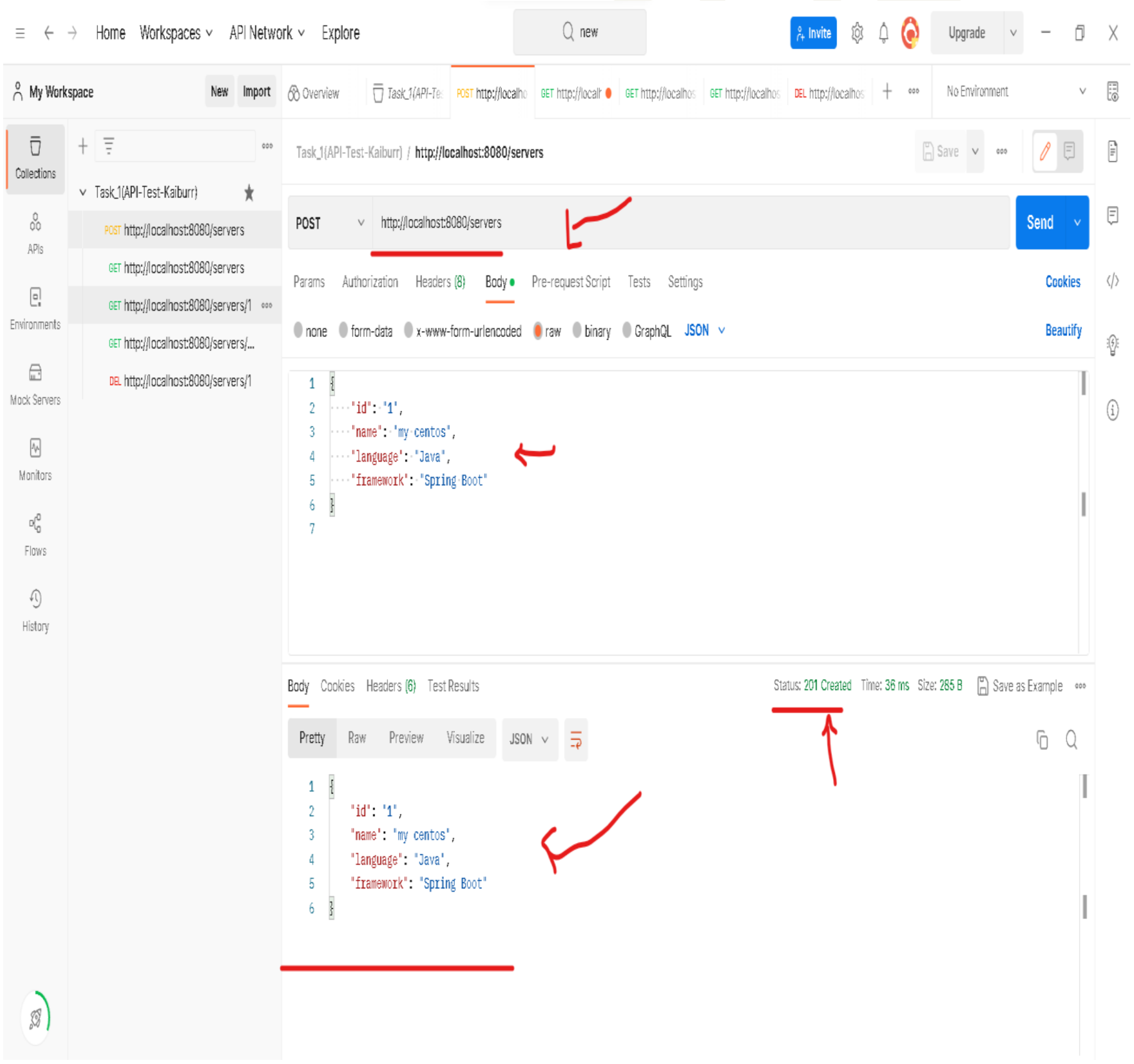
- ***getAllServers()*** corresponds to ***the GET request to /servers***
- ***getServerById(String id)*** corresponds to the ***GET request to /servers/{id}***
- ***createServer(Server server)*** corresponds to the ***PUT request to /servers***
- ***deleteServer(String id)*** corresponds to the ***DELETE request to /servers/{id}***
- ***getServersByName(String name)*** corresponds to the ***GET request to /servers/search?name={name}.***

❖ **Output:**

➢ Create/Add(POST) - http://localhost:8080/servers

- ***BODY:***

```
{
"id": "1",
"name": "my centos",
"language": "Java",
"framework": "Spring Boot"
}
```



Task_1(API-Test-Kaiburr) / http://localhost:8080/servers

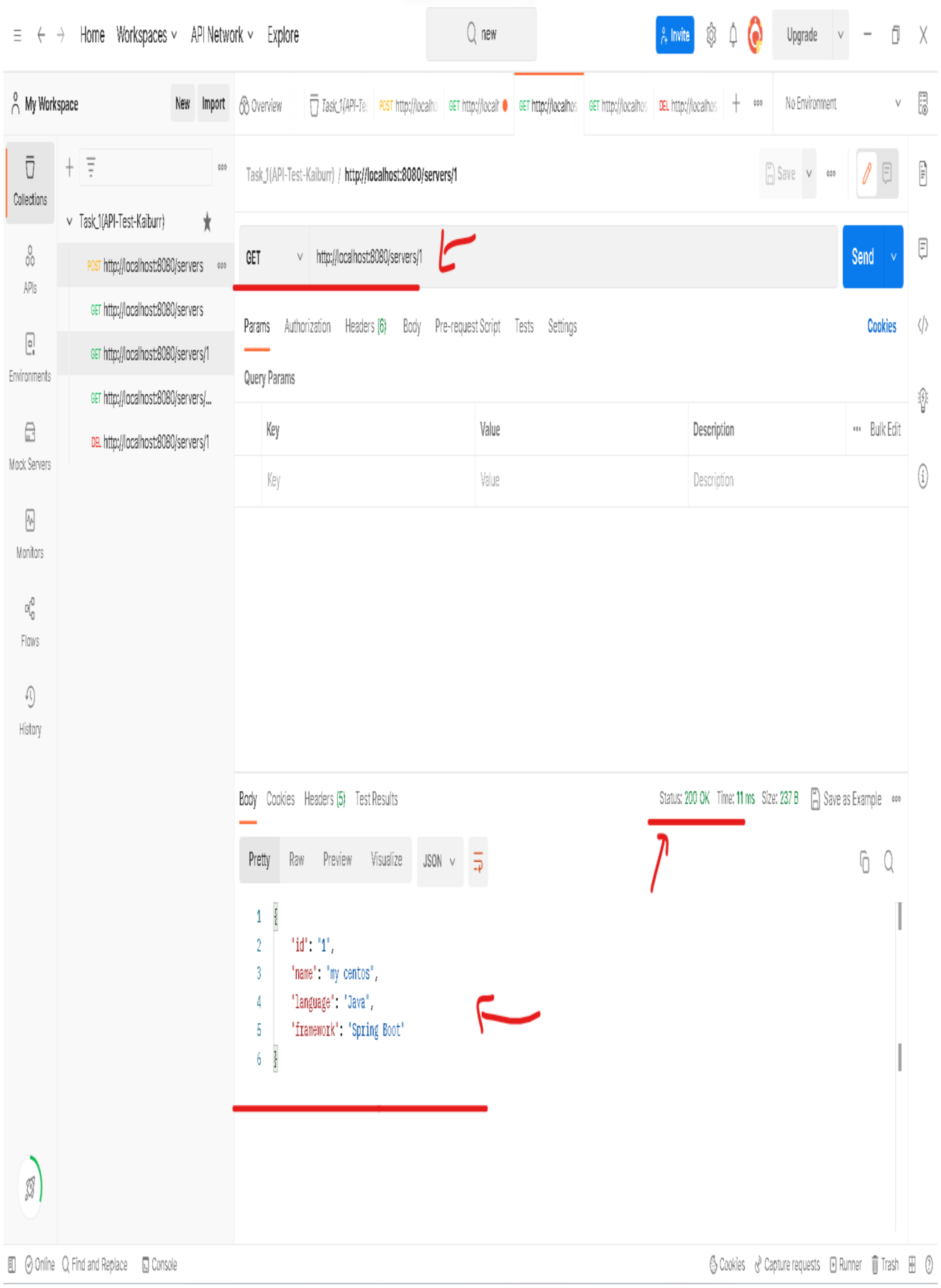
POST http://localhost:8080/servers

```
{
    "id": "1",
    "name": "my centos",
    "language": "Java",
    "framework": "Spring Boot"
}
```

Status: 201 Created Time: 36 ms Size: 285 B

```
{
    "id": "1",
    "name": "my centos",
    "language": "Java",
    "framework": "Spring Boot"
}
```

➢ Search by ID (GET) - http://localhost:8080/servers/1



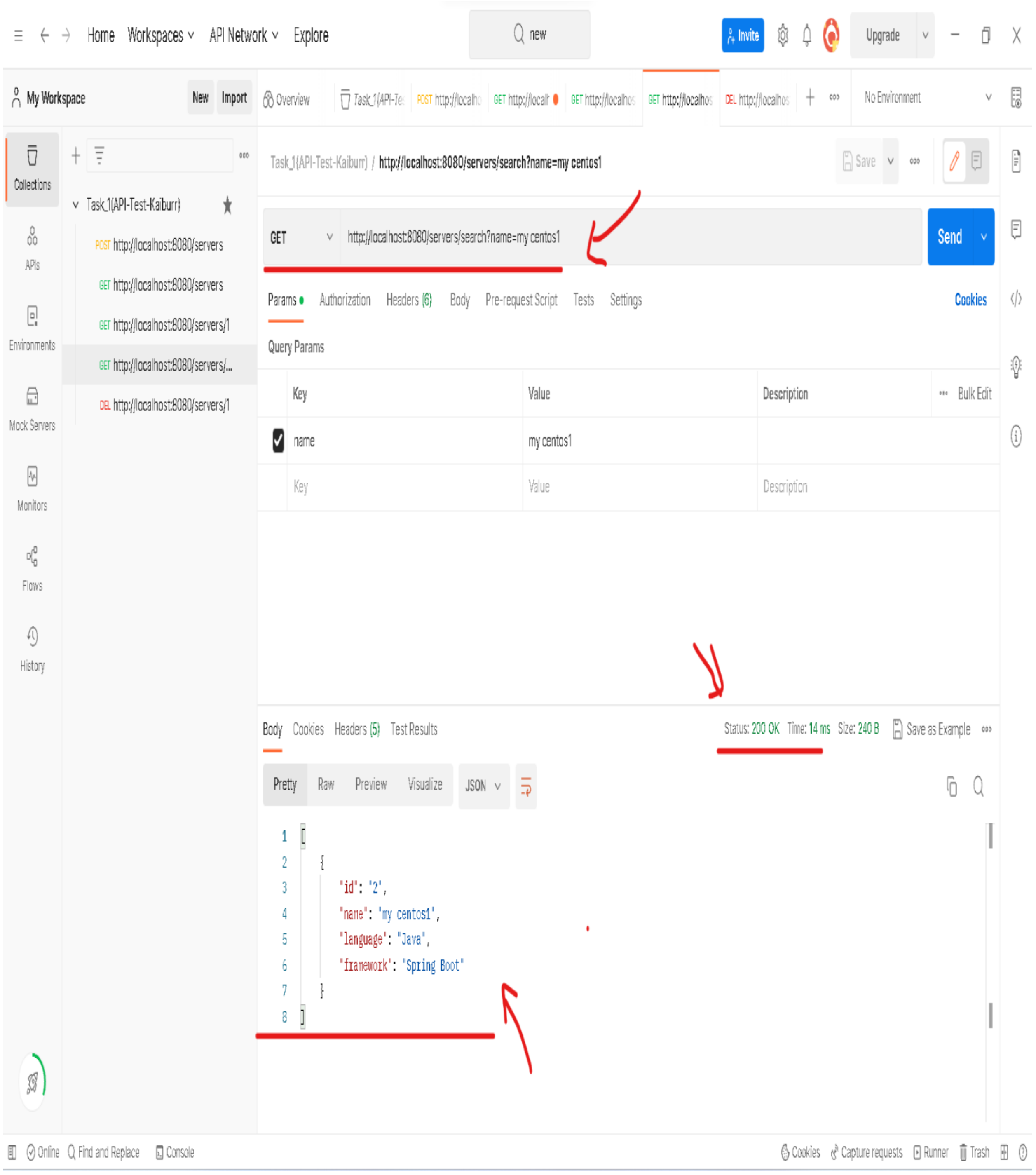
Task_1(API-Test-Kaiburr) / http://localhost:8080/servers/1

GET http://localhost:8080/servers/1

Status: 200 OK Time: 11 ms Size: 237 B

```
{
    "id": "1",
    "name": "my centos",
    "language": "Java",
    "framework": "Spring Boot"
}
```

➢ Search by Name (GET) - http://localhost:8080/servers/search?name=my centos1



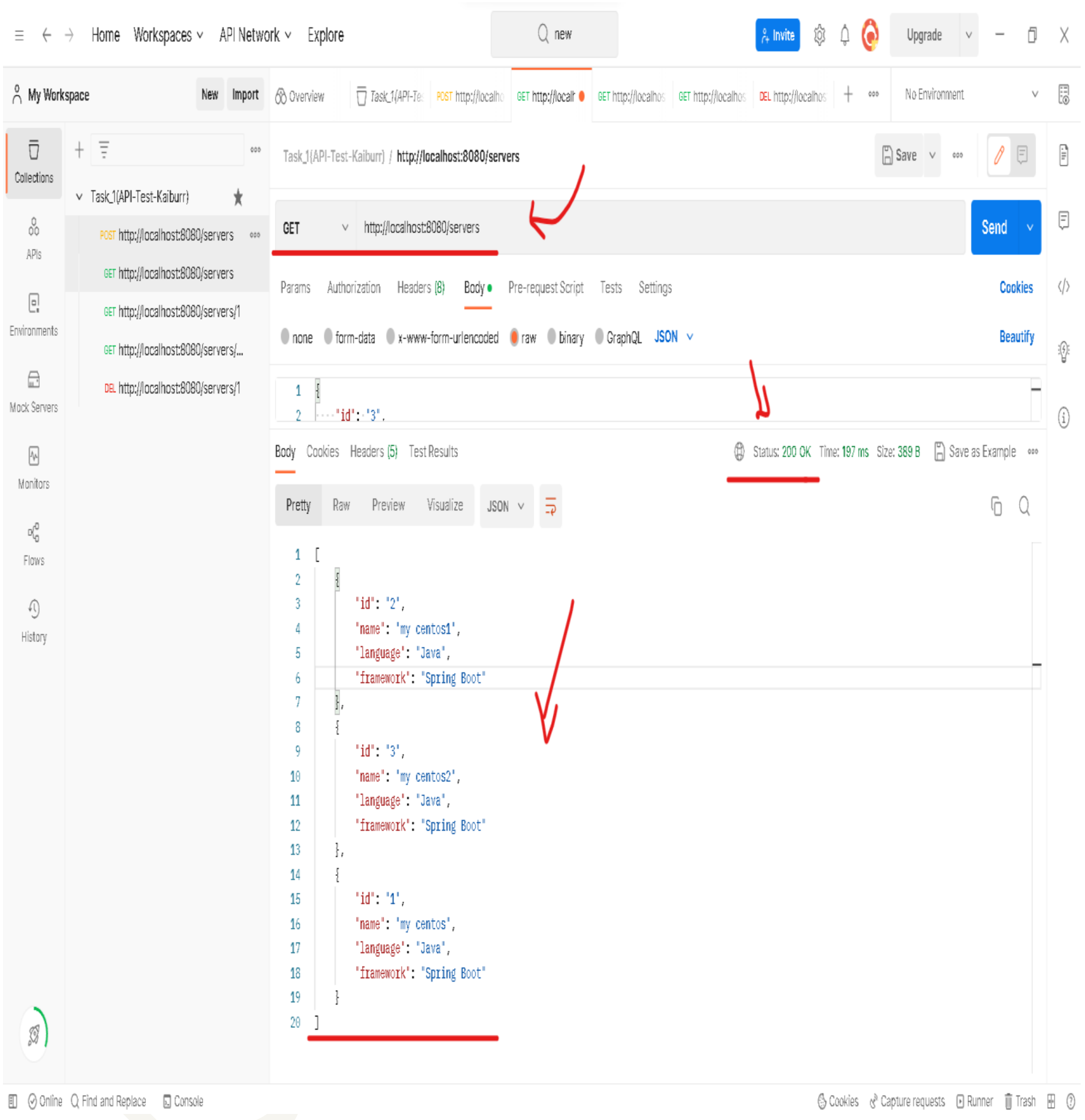
GET http://localhost:8080/servers/search?name=my centos1

Key	Value	Description
✓ name	my centos1	
Key	Value	Description

Status: 200 OK Time: 14 ms Size: 240 B

```
[
    {
        "id": "2",
        "name": "my centos1",
        "language": "Java",
        "framework": "Spring Boot"
    }
]
```

➢ View all the Servers (GET) - http://localhost:8080/servers



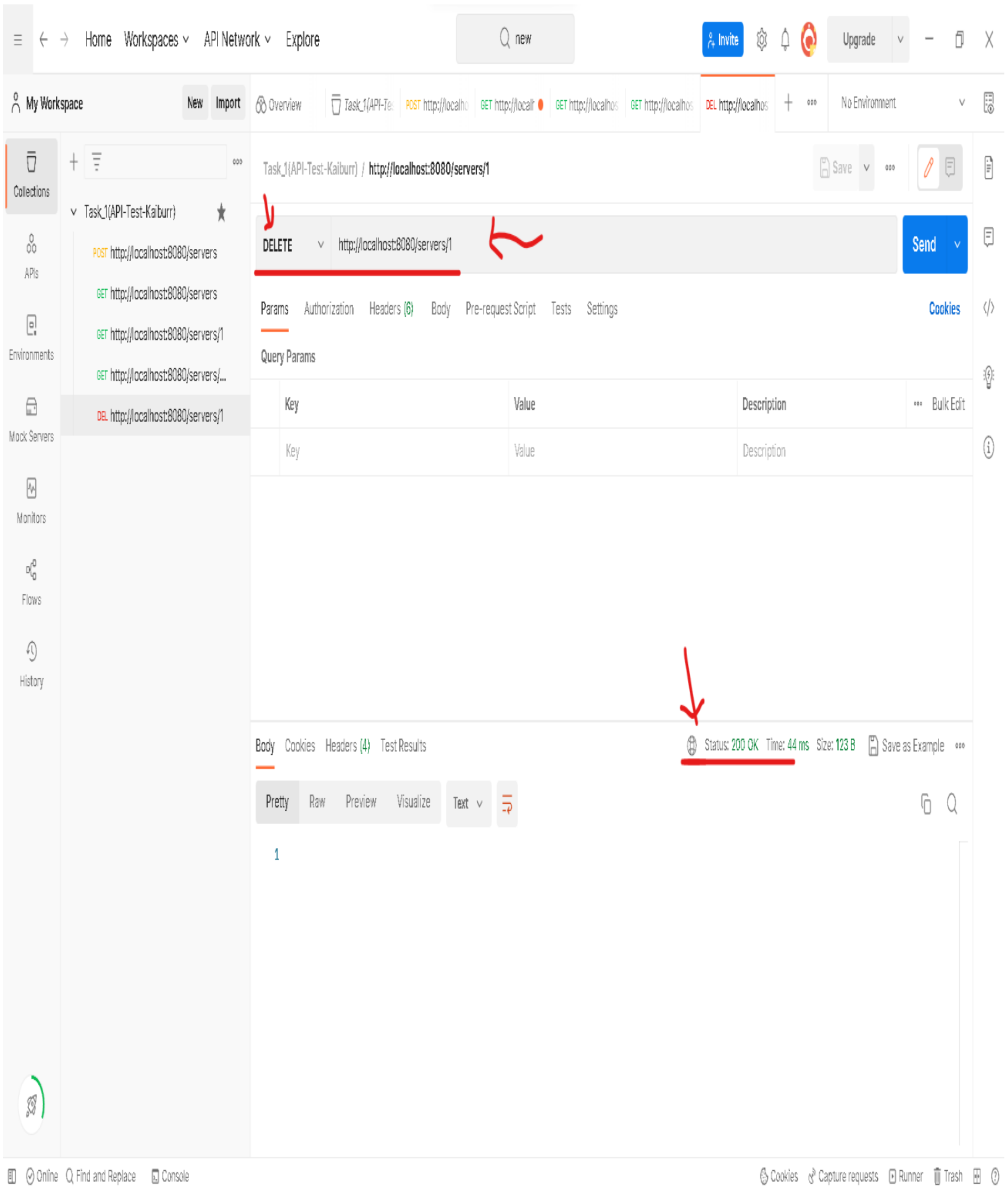
GET http://localhost:8080/servers

```
{
····"id":"3",
```

Status: 200 OK Time: 197 ms Size: 389 B

```
[
    {
        "id": "2",
        "name": "my centos1",
        "language": "Java",
        "framework": "Spring Boot"
    },
    {
        "id": "3",
        "name": "my centos2",
        "language": "Java",
        "framework": "Spring Boot"
    },
    {
        "id": "1",
        "name": "my centos",
        "language": "Java",
        "framework": "Spring Boot"
    }
]
```

➢ Delete by ID (DELETE) - http://localhost:8080/servers/1



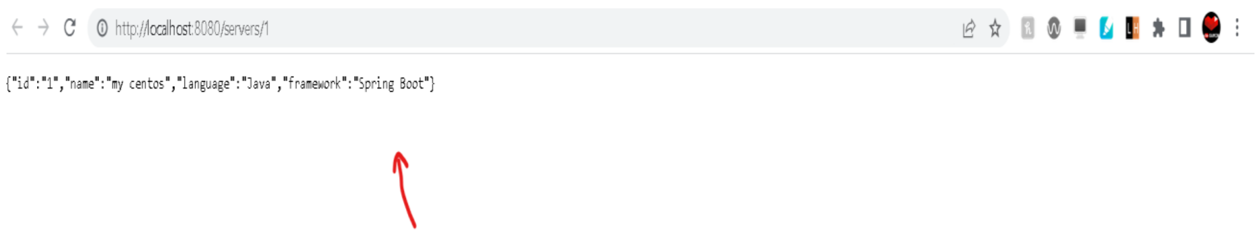
Task_1(API-Test-Kaiburr) / http://localhost:8080/servers/1

DELETE http://localhost:8080/servers/1

Status: 200 OK Time: 44 ms Size: 123 B

Browser (Google Chrome Output):

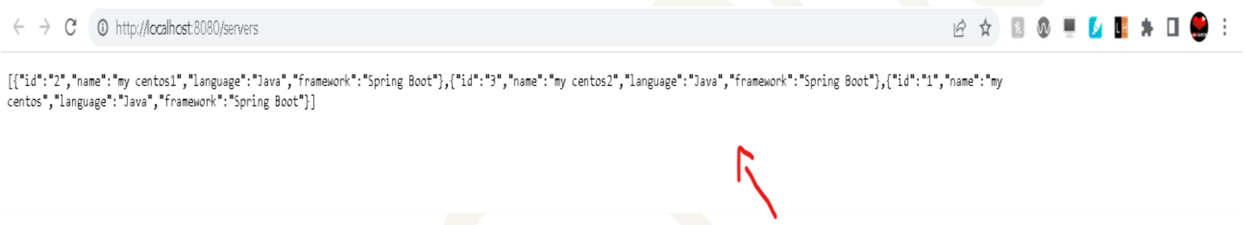
- Search by ID (GET) - http://localhost:8080/servers/1



http://localhost:8080/servers/1

```
{"id":"1","name":"my centos","language":"Java","framework":"Spring Boot"}
```

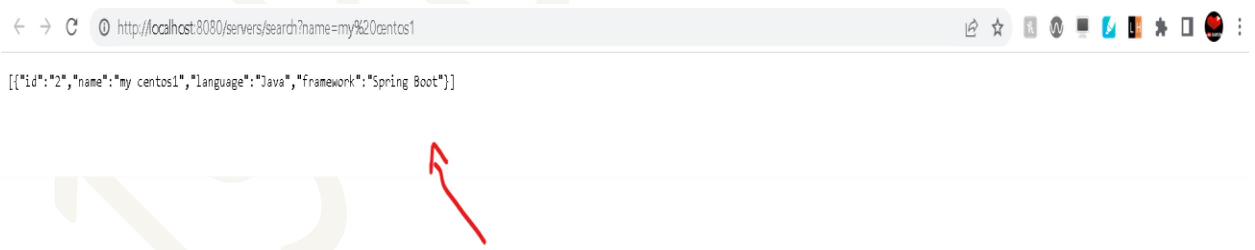
- View all Servers (GET) - http://localhost:8080/servers



http://localhost:8080/servers

```
[{"id":"2","name":"my centos1","language":"Java","framework":"Spring Boot"},{"id":"3","name":"my centos2","language":"Java","framework":"Spring Boot"},{"id":"1","name":"my
centos","language":"Java","framework":"Spring Boot"}]
```

- Search by Name (GET) - http://localhost:8080/servers/search?name=my centos1



http://localhost:8080/servers/search?name=my%20centos1

```
[{"id":"2","name":"my centos1","language":"Java","framework":"Spring Boot"}]
```

Summary of the steps to create a RESTful API in Java and MongoDB for creating data in JSON format (Task 1):

1. Install MongoDB

2. Create a new Spring Boot project with Spring Initializer, adding the Spring Web, Spring Data MongoDB, and Lombok dependencies.

3. Define the Server model with attributes id, name, language, and framework, and add Lombok annotations for getters, setters, and constructors.

4. Define the Server repository with the methods findByNameContaining and findById.

5. Define the Server service with methods getAllServers, getServerById, createServer, deleteServer, and getServersByName, calling the corresponding methods of the ServerRepository object.

6. Define the Server controller with methods getAllServers, getServerById, createServer, deleteServer, and getServersByName, calling the corresponding methods of the ServerService object and returning the result.

7. Configure MongoDB in the application.properties file.

8. Run the application by executing the RestApiApplication class.

9. Test the API using Postman or any other HTTP client, with GET requests for getting all servers or a single server by ID, and a PUT request for creating a server in JSON format.

Thank You!