Arpit Singh

19BCG10069

Kaiburr – Task 1

Technical Task

(Placement)

GitHub Link: https://github.com/TSM-ArpitSG/Kaiburr/tree/main/Kaibur_Tasks/Task_2

Task 2:

Swagger codegen:

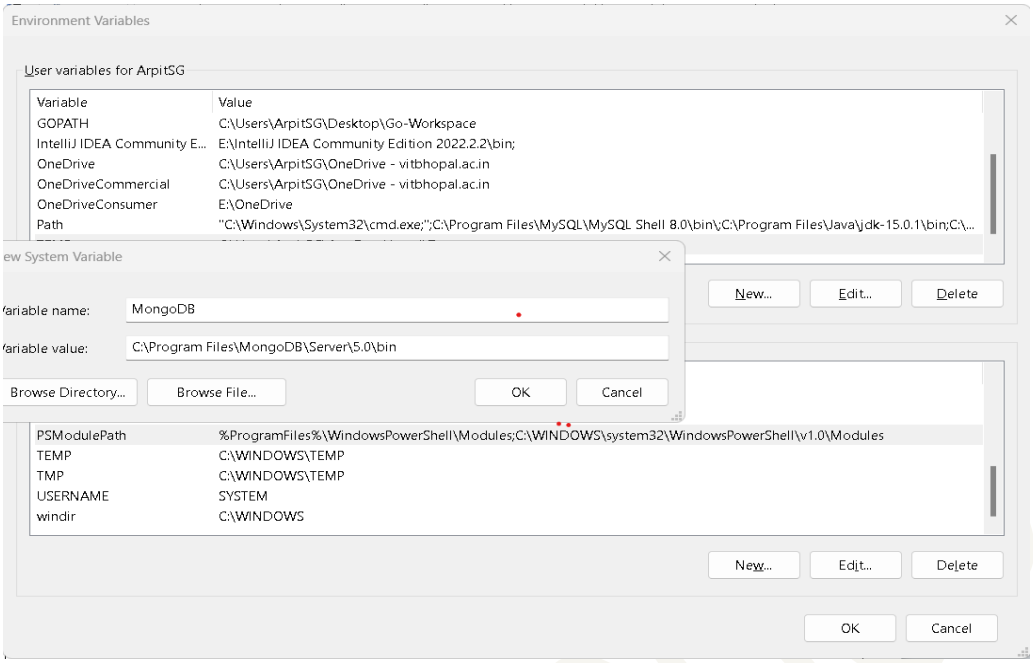
Create the same REST API as in task #1, but use https://editor.swagger.io/ to create your API definition and generate the server code. Choose any java-based server or server framework that you like. You can either use the online editor or generate the code manually, e.g. using this document: https://github.com/swagger-api/swagger-codegen/wiki/server-stub-generator-howto.

Make sure that you can deploy/run the generated code. Once your stub is ready - implement the same functionality as described in task #1, but now in java.

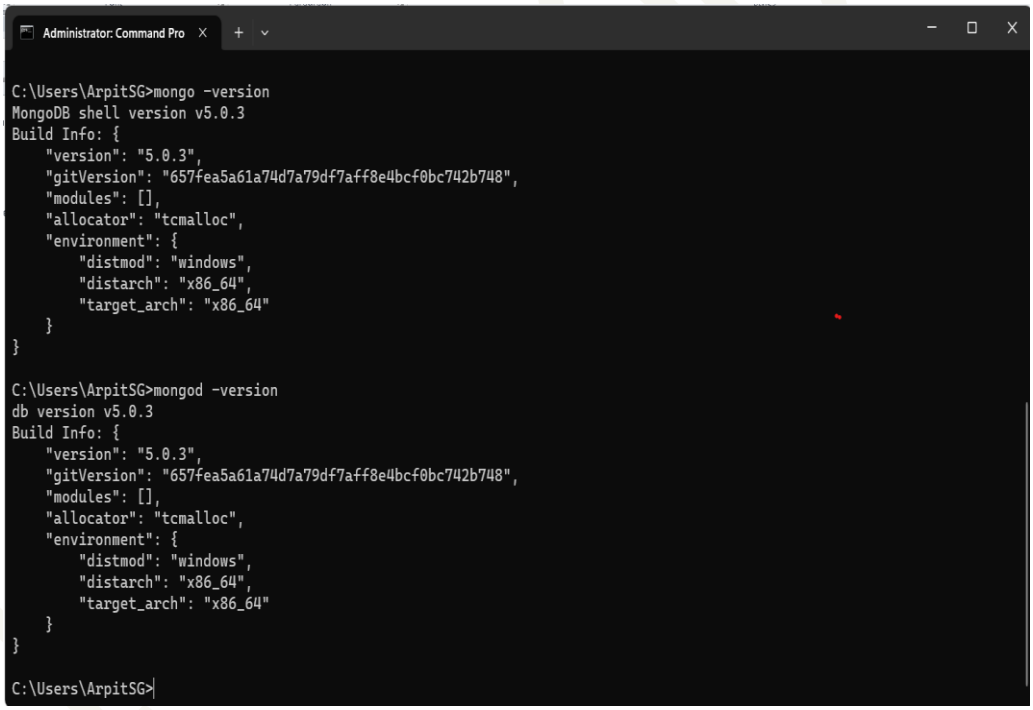
Finally, be sure that you can show how your application responds to requests using postman, curl or any other HTTP client.

Steps:

1. **Installing MongoDB and Setting up the Environment(DONE IN TASK 1)**:
    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.*



❖ Successful Installation can be checked as follows:

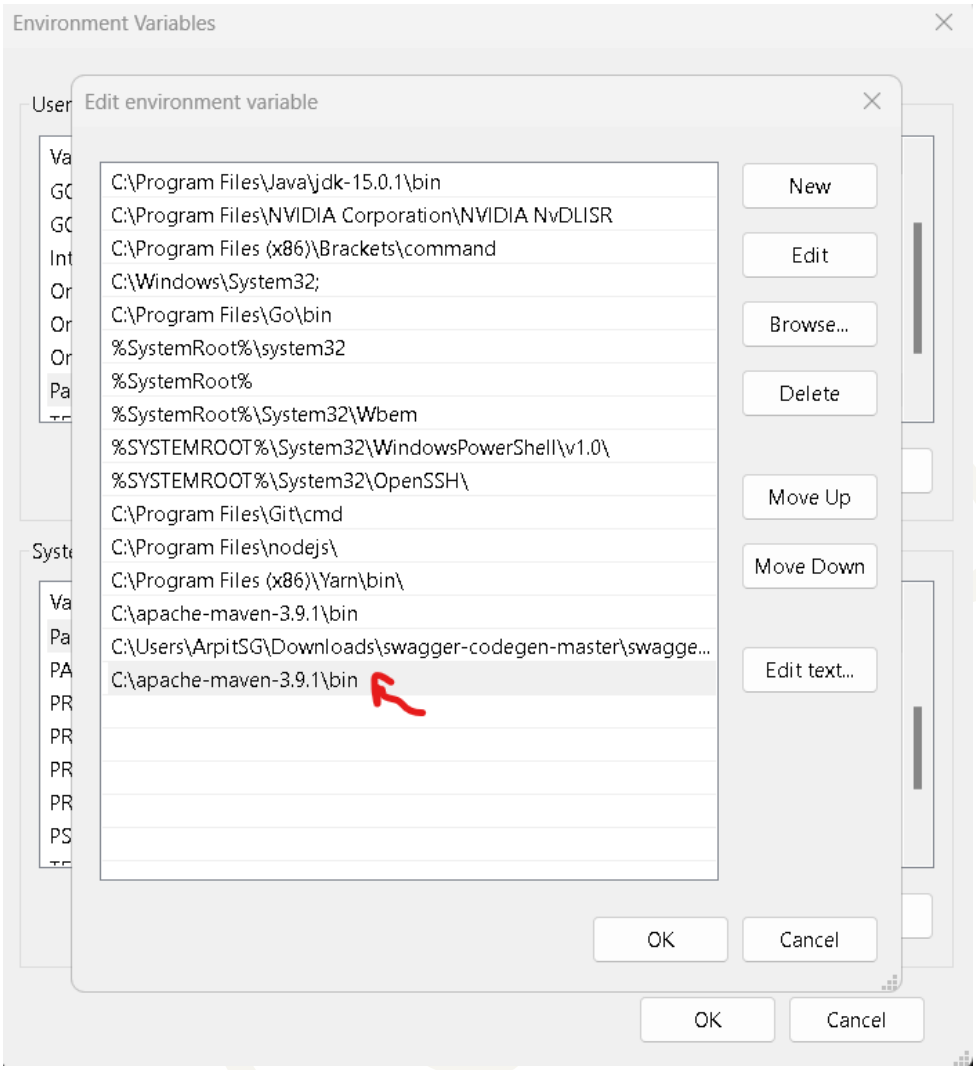


2. Installing the Updated Version of MAVEN in your System:

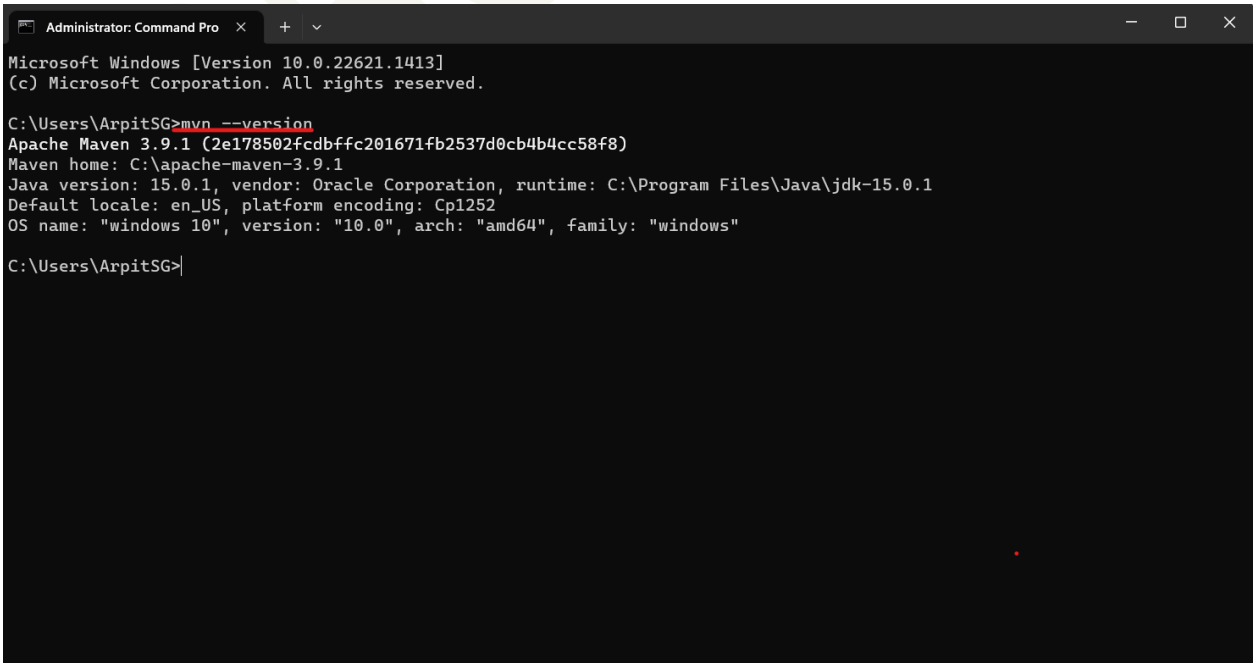
a. Download the latest version of ***Maven*** from the official Apache Maven website (https://maven.apache.org/download.cgi).

b. Extract the downloaded archive file to a directory on your system where you want to install Maven. For example, you can extract it to "***C:\Program Files***".

c. Set up the environment variable for Maven. ***C:\apache-maven-3.9.1\bin***.

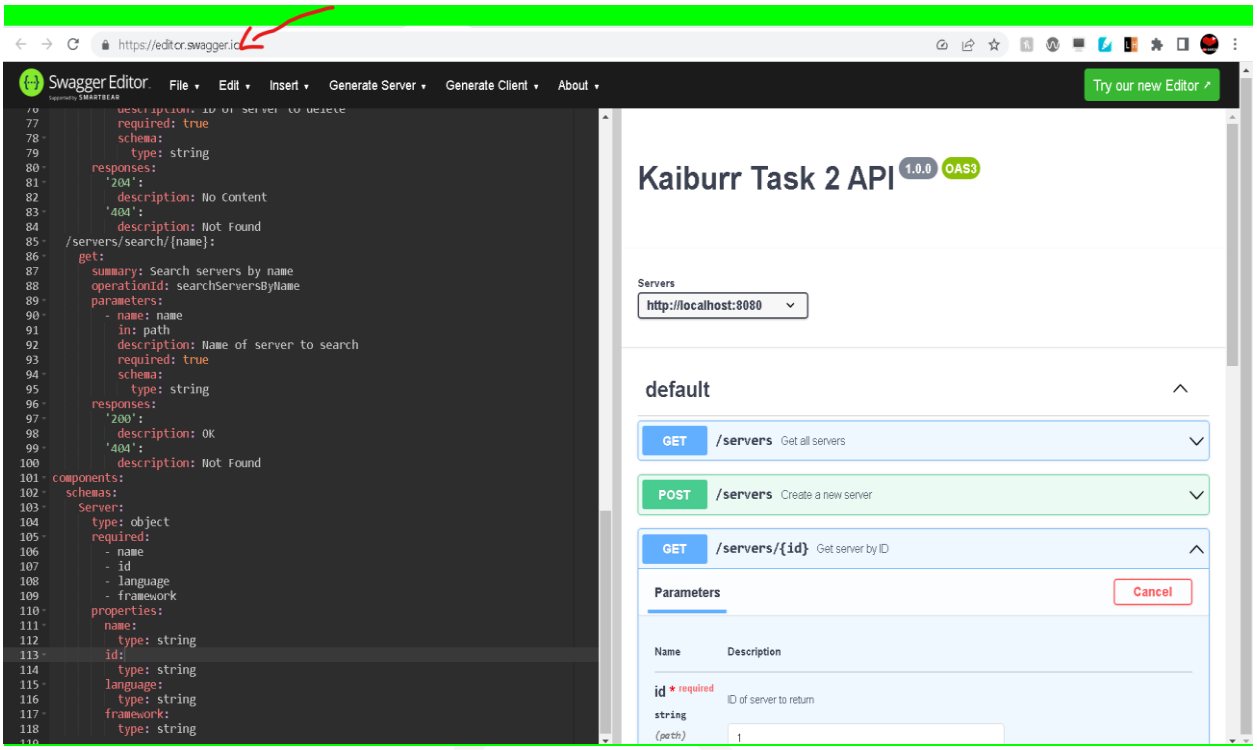


❖ Successful Installation can be checked as follows:



3. Create a Swagger API Definition:

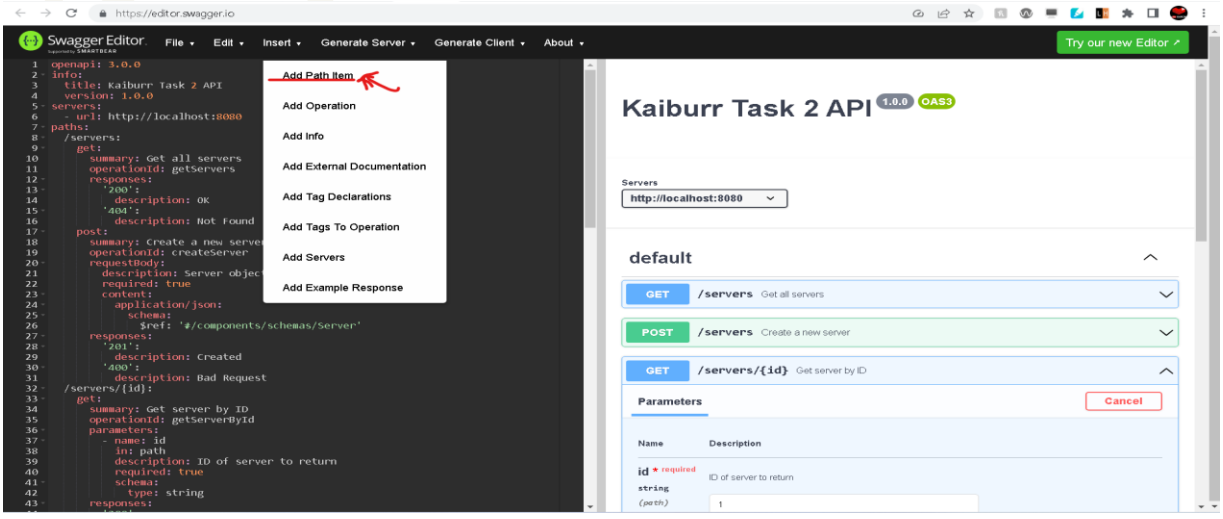
a. Go to "***https://editor.swagger.io/***"



b. Click on "***File***" in the top left corner of the screen.

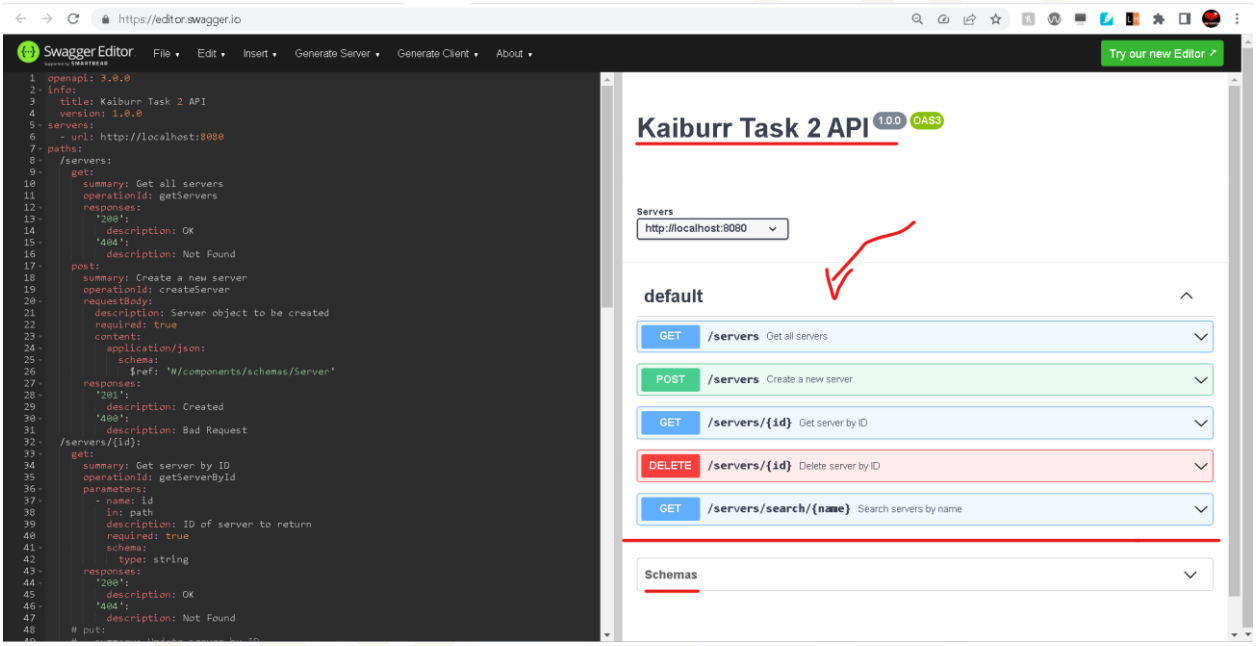
c. Click on "***New***" to create a new API definition.
(***Alternatively, we can simply "Delete the demo code" and start writing our API definition code in 'YAML or JSON' format***).

d. Define your '***API endpoints***' by clicking on "***Paths***" in the left-hand sidebar, and then clicking on the "***Add Path***" button.



e. Define the '***HTTP methods***' for each endpoint (e.g. ***GET, POST, DELETE***) by clicking on the endpoint and then clicking on the appropriate HTTP method button.
f. Define any necessary parameters for each endpoint by clicking on the endpoint, scrolling down to "***Parameters***", and then clicking on the "***Add Parameter***" button.
g. Define any necessary data models by clicking on "***Components***" in the left-hand sidebar, and then clicking on the "***Add Schema***" button.

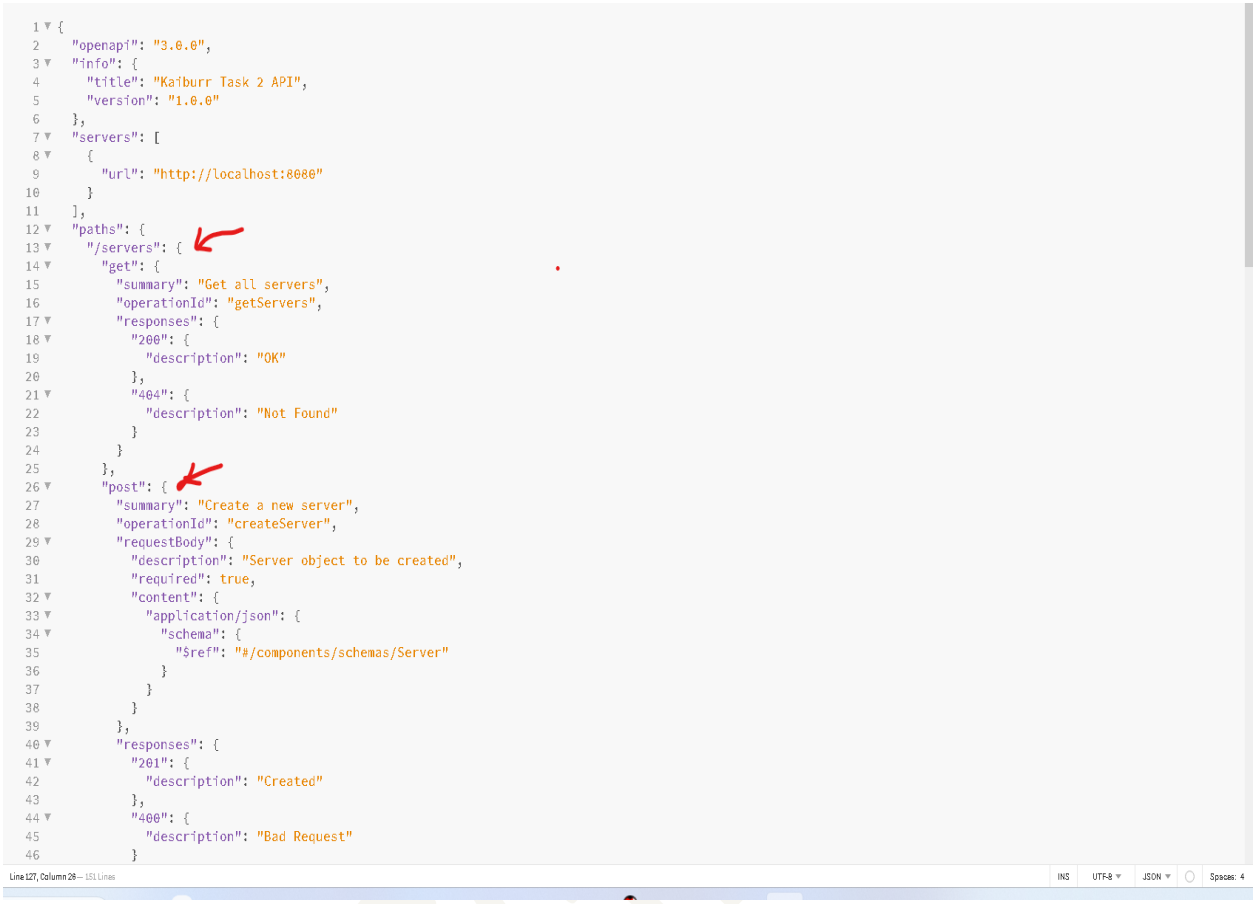
   (After performing the above 3 steps we will be ready with our '***API DEFINATION***' and with the help of "***Swagger-UI***" we should be able to see our ***API endpoints as we created in Task 1***).



h. Once you have defined all of the necessary endpoints, parameters, and data models, click on "***File***" in the top left corner of the screen and select "***Save As***".
   - Choose whether you want to save your API definition as a '***YAML***' or '***JSON***' file. (***I used JSON format***)
   - ***Save*** the file to your local machine.

Swagger™
Supported by SMARTBEAR

API DEFINATION FILE:

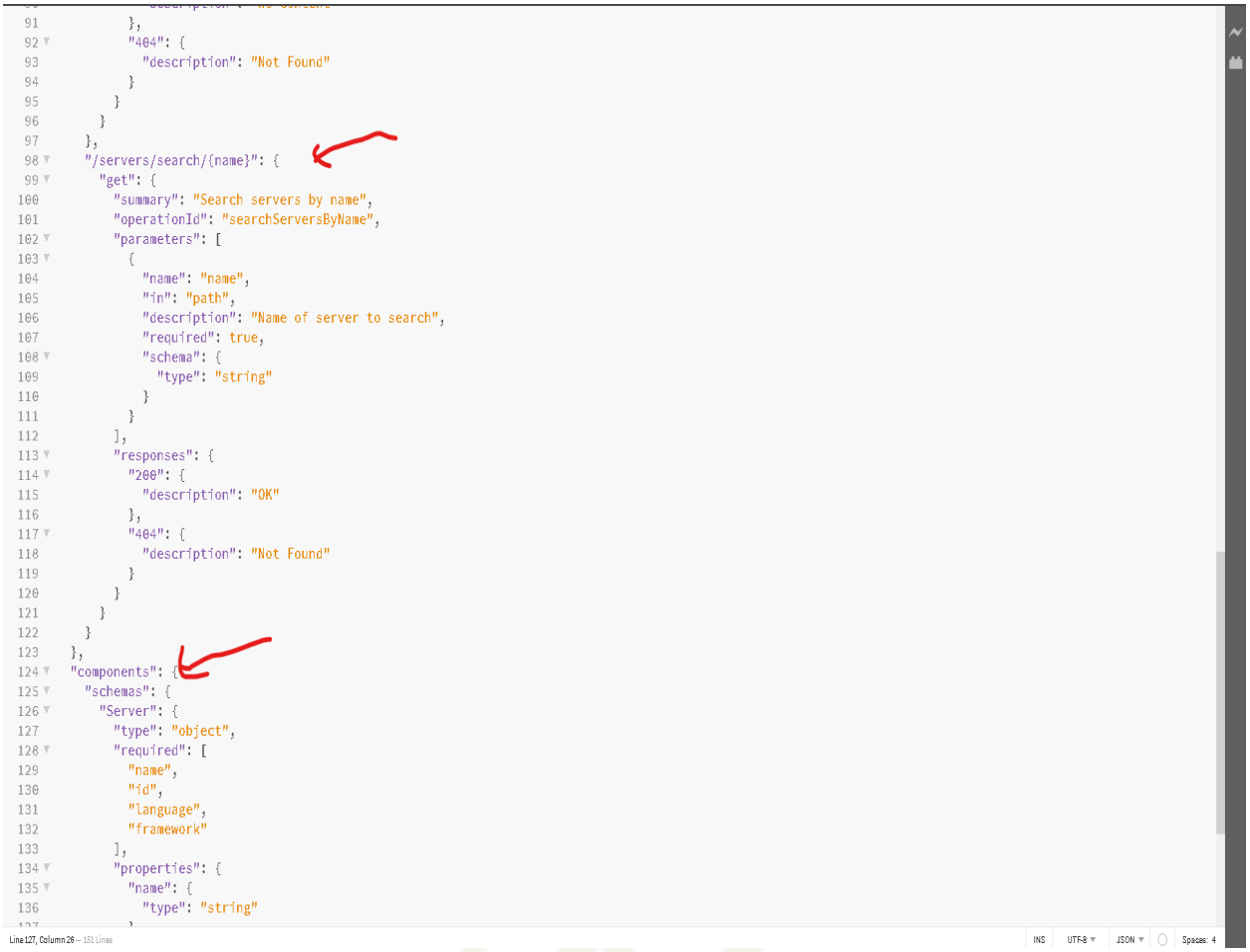


```
{
  "openapi": "3.0.0",
  "info": {
    "title": "Kaiburr Task 2 API",
    "version": "1.0.0"
  },
  "servers": [
    {
      "url": "http://localhost:8080"
    }
  ],
  "paths": {
    "/servers": {
      "get": {
        "summary": "Get all servers",
        "operationId": "getServers",
        "responses": {
          "200": {
            "description": "OK"
          },
          "404": {
            "description": "Not Found"
          }
        }
      },
      "post": {
        "summary": "Create a new server",
        "operationId": "createServer",
        "requestBody": {
          "description": "Server object to be created",
          "required": true,
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/Server"
              }
            }
          }
        },
        "responses": {
          "201": {
            "description": "Created"
          },
          "400": {
            "description": "Bad Request"
          }
```

Line 127, Column 28 — 151 Lines INS UTF-8 JSON Spaces: 4

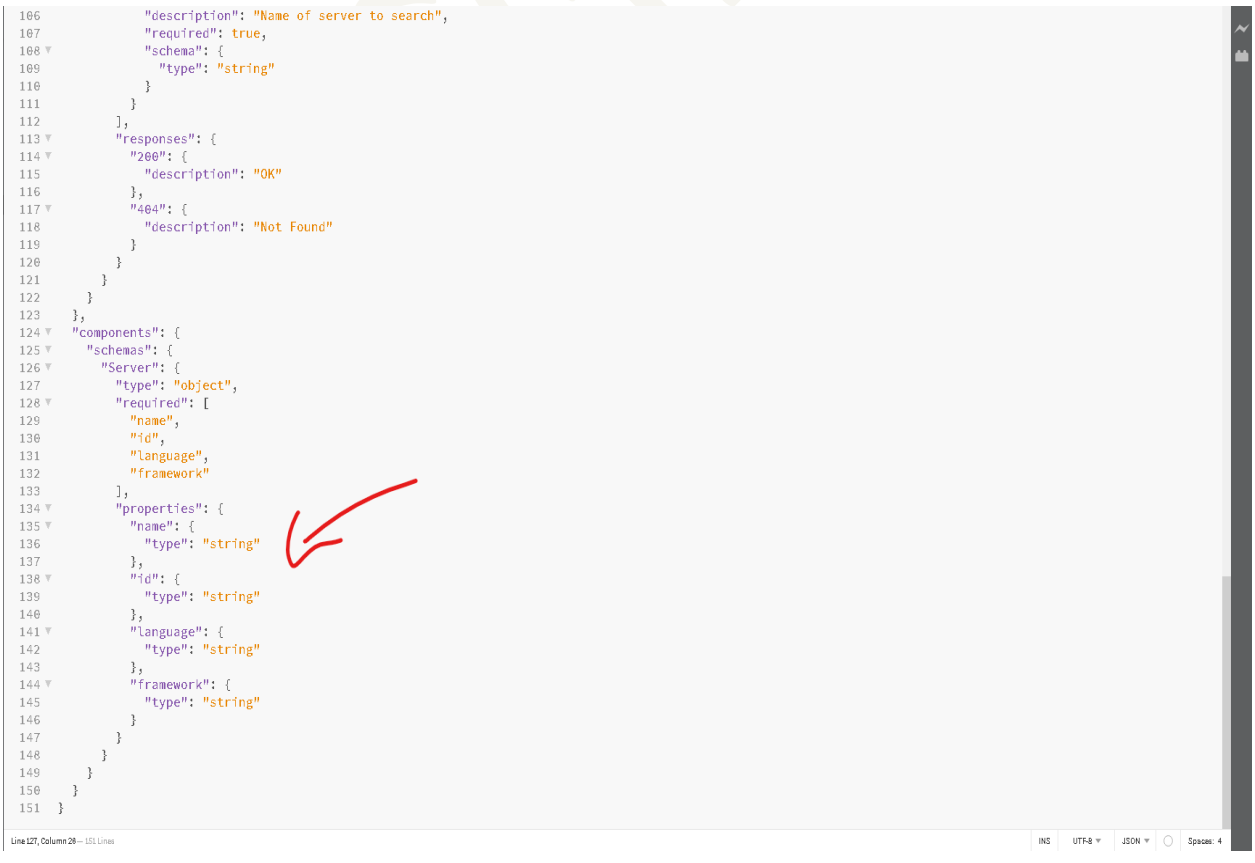
```
          }
        }
      }
    },
    "/servers/{id}": {
      "get": {
        "summary": "Get server by ID",
        "operationId": "getServerById",
        "parameters": [
          {
            "name": "id",
            "in": "path",
            "description": "ID of server to return",
            "required": true,
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "OK"
          },
          "404": {
            "description": "Not Found"
          }
        }
      },
      "delete": {
        "summary": "Delete server by ID",
        "operationId": "deleteServerById",
        "parameters": [
          {
            "name": "id",
            "in": "path",
            "description": "ID of server to delete",
            "required": true,
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {
          "204": {
            "description": "No Content"
          },
          "404": {
```

Line 127, Column 28 — 151 Lines INS UTF-8 JSON Spaces: 4



```
          },
          "404": {
            "description": "Not Found"
          }
        }
      }
    },
    "/servers/search/{name}": {
      "get": {
        "summary": "Search servers by name",
        "operationId": "searchServersByName",
        "parameters": [
          {
            "name": "name",
            "in": "path",
            "description": "Name of server to search",
            "required": true,
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "OK"
          },
          "404": {
            "description": "Not Found"
          }
        }
      }
    }
  },
  "components": {
    "schemas": {
      "Server": {
        "type": "object",
        "required": [
          "name",
          "id",
          "language",
          "framework"
        ],
        "properties": {
          "name": {
            "type": "string"
```

Line 127, Column 26 — 151 Lines — INS — UTF-8 — JSON — Spaces: 4



```
            "description": "Name of server to search",
            "required": true,
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "OK"
          },
          "404": {
            "description": "Not Found"
          }
        }
      }
    }
  },
  "components": {
    "schemas": {
      "Server": {
        "type": "object",
        "required": [
          "name",
          "id",
          "language",
          "framework"
        ],
        "properties": {
          "name": {
            "type": "string"
          },
          "id": {
            "type": "string"
          },
          "language": {
            "type": "string"
          },
          "framework": {
            "type": "string"
          }
        }
      }
    }
  }
}
```

Line 127, Column 26 — 151 Lines — INS — UTF-8 — JSON — Spaces: 4

Explanation (Code):

- So, here the code is quite '***self-explanatory'*** we have added ***API endpoints and data models*** as per Task 1(***JSON*** Format).

- To perform the ***Create, Delete, View and Search*** operations.

- The '***path'*** helps us to navigate to the '***directory'*** where the correspond-ing/required file is located.

- ***'HTTP'*** requests tell which type of operation it is like '***GET, POST, and PUT'*** etc.

- ***'Summary'*** reflects '***what the function does***' and '***operationId'*** sets the name of the '***function***'.

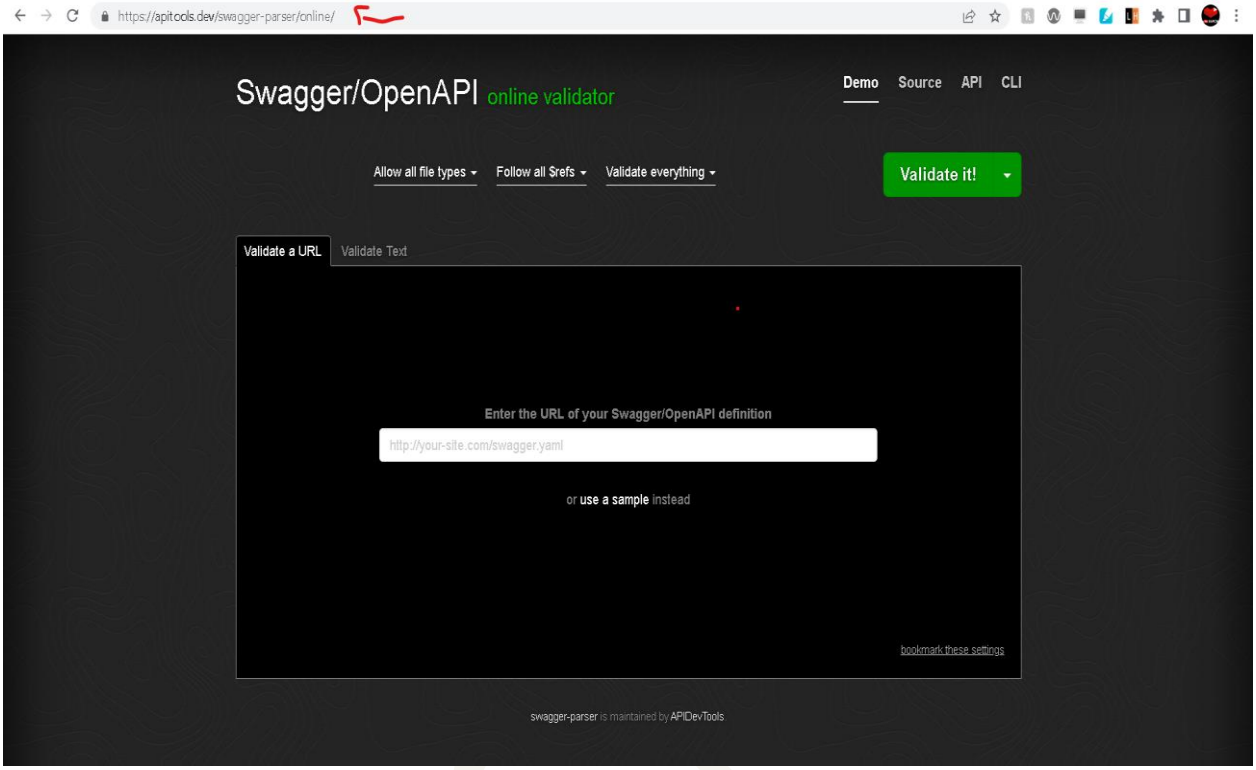
- ***'Parameters and response define the logic'*** or working of our functions.

- Lastly we use '***Schemas under Component'*** section to create different '***data models'*** as per our requirement ("***Server"*** in our case).

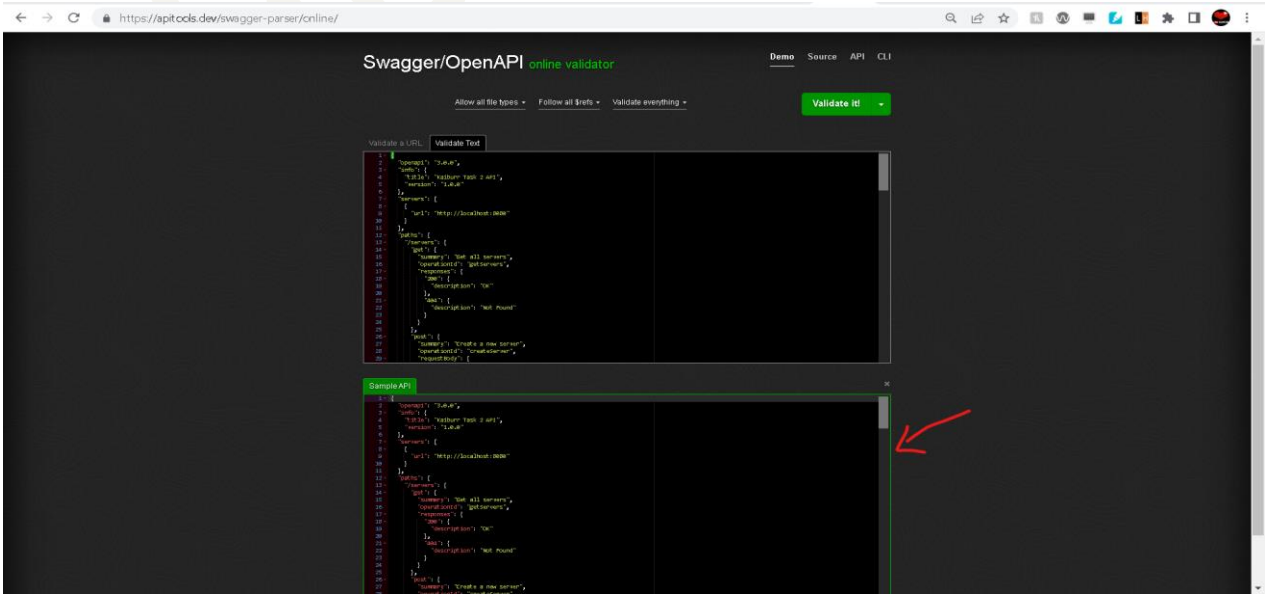
The API endpoints include:

- ***GET /servers***: to get all servers
- ***POST /servers***: to create a new server
- ***GET /servers/{id}***: to get a server by ID
- ***DELETE /servers/{id}***: to delete a server by ID
- ***GET /servers/search/{name}***: to search servers by name
- The "***data model defined"*** in the API is "***Server"***, which includes the properties '***name, id, language, and framework***'.

4. **Validate and save your API definition file:**
    a. Go ***to https://apitools.dev/swagger-parser/online/***.

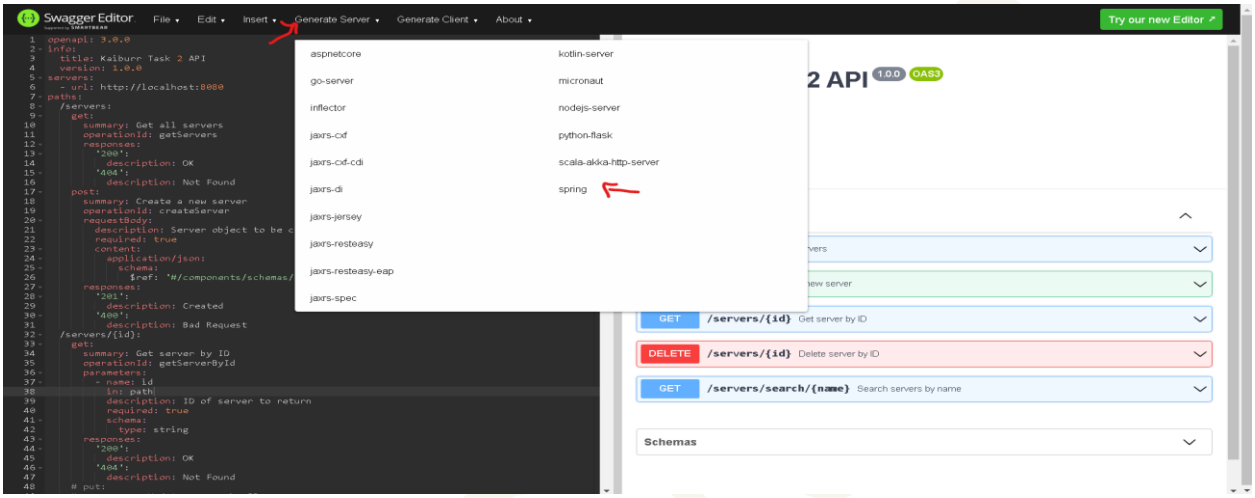


    b. Upload your '***API DEFINATION***' file here or simply '***Copy/Paste***' the '***code***'.
    c. After pasting, press on "***Validate it!***" Button and a ***green coloured high-lighted box*** with your src code will appear showing the API definition is ***Correct/Valid***. (if the API DEFINATION is ***wrong a red highlight*** will be shown)

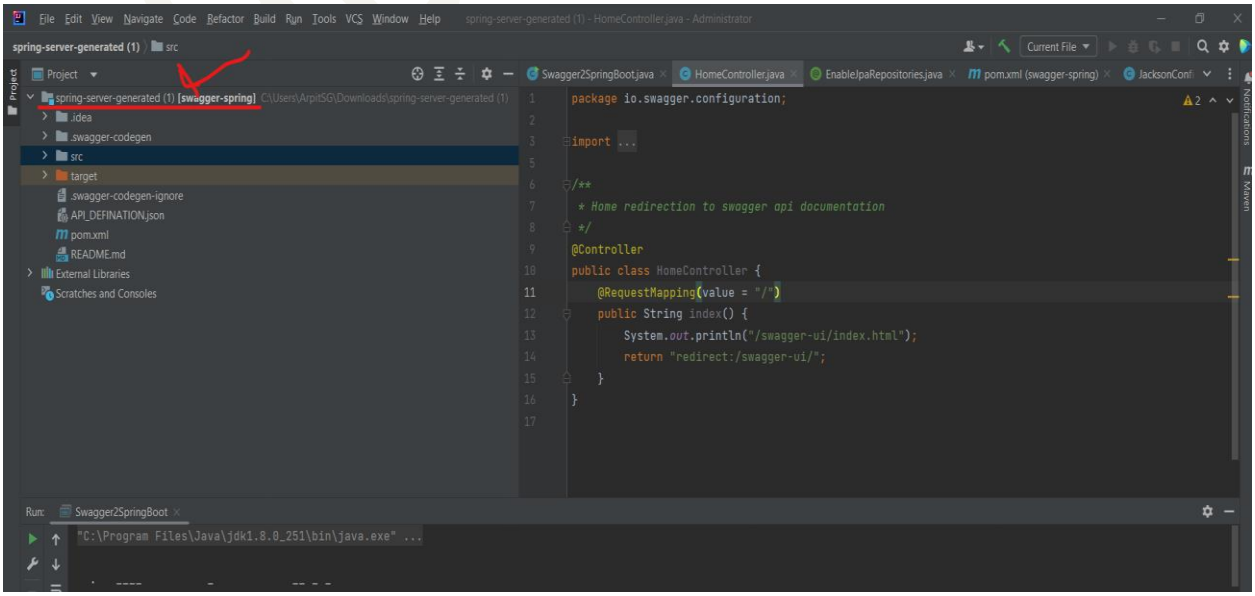


5. **Generate server code using Swagger:**
    a. There are "***2 ways of generating Server code***" using Swagger:
        i. ***Swagger Editor*** ***(I used this one).***
        ii. ***Swagger Codegen*** ***(Steps provided in doc):***
            1. Download Swagger Codegen from ***https://swagger.io/tools/swagger-codegen/***
            2. ***Extract*** the downloaded file to a directory of your choice.
            3. Open a command prompt or terminal and ***navigate to the directory*** where you extracted Swagger Codegen.
            4. Run the following ***command to generate server code***:
                a. `java -jar swagger-codegen-cli.jar generate -i <path-to-your-api-definition-file> -l <language> -o <output-directory>`
                b. Replace **<path-to-your-api-definition-file>** with the path to your saved ***API definition file***, **<language>** with the programming language you want to generate the server code in (e.g. ***java***), and **<output-directory>** with the directory where you want ***to save the generated code.***
            5. ***Build and run*** the generated code.
                a. ***Navigate to the output directory*** where the generated code was saved.
                b. Build the code using Maven by running the following command: `mvn clean package`
                c. Run the server using the following command: `java -jar <name-of-generated-jar-file>.jar`
                d. Replace **<name-of-generated-jar-file>** with the name of the ***JAR file generated*** in the previous step.
            6. Implement the ***same functionality as described in Task 1***.
                a. Open the generated code in your favourite IDE.
                b. Implement the ***same endpoints and functionality as described in Task 1***.
                c. Make sure to connect to a ***MongoDB database*** to store and retrieve "***Server***" objects.
            7. ***Test your API using Postman***, curl or any other HTTP client.
                a. Make requests to your API endpoints using Postman, curl or any other HTTP client to ***ensure that the functionality is working as expected***.

b. ***Swagger Editor*** (***I used this one***):
   i. Go to "***https://editor.swagger.io/***"
   j. Paste your '***API DEFINATION***' code in the Editor.
   k. Now, instead of saving the file:
      - Go on the "***Generate Server dropdown***" option a
      - Click on the ***type of project*** you want to create. (We will be creating a '***Spring***' server)

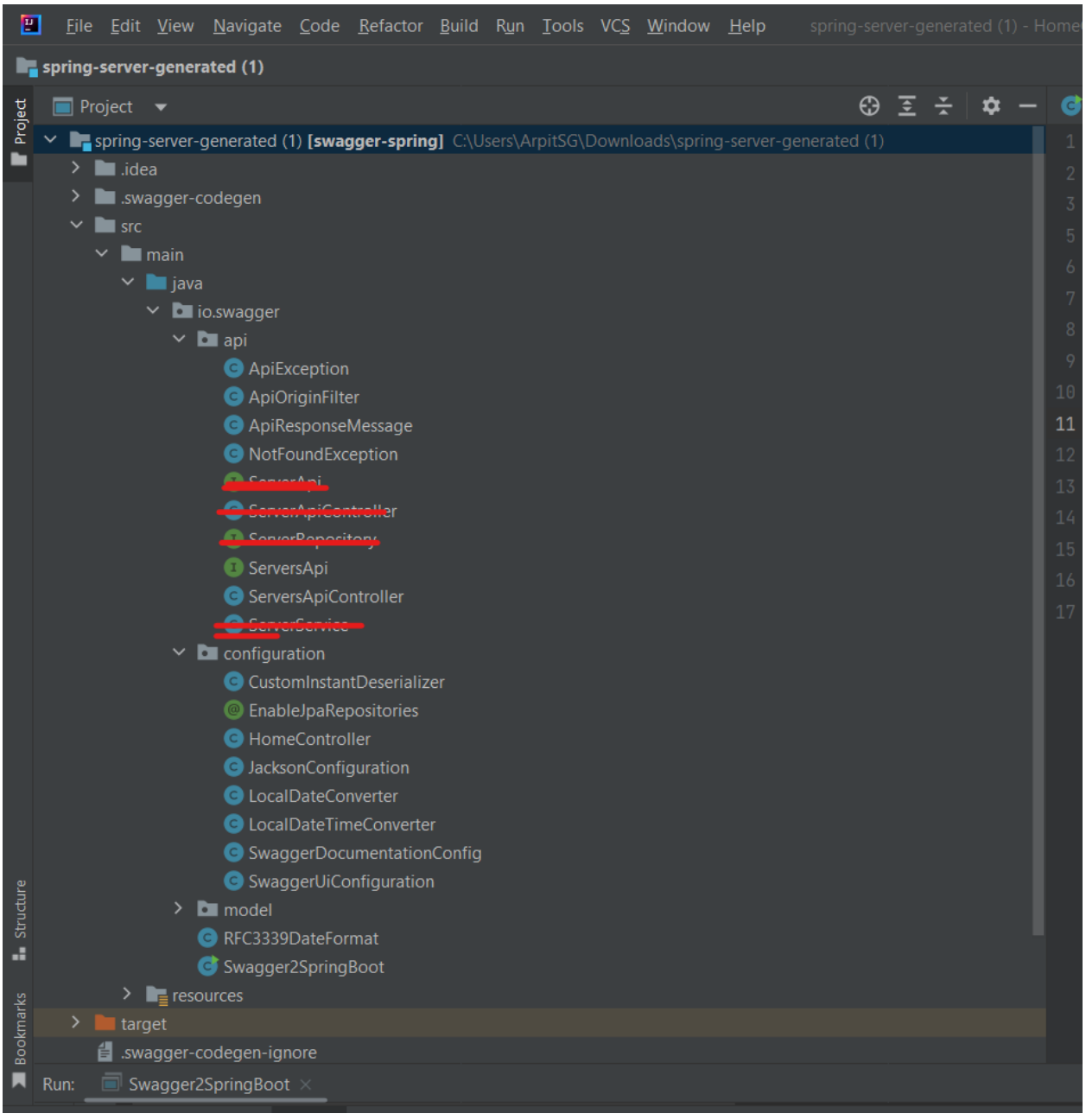


   l. This will give you a ***Zip file*** containing an auto-generated spring project based off the ***API Definition*** you provided. ***Ex: "C:\Users\ArpitSG\Downloads\spring-server-generated.zip"***
   m. You can ***Extract and open up this project with IntelliJ*** or preferred IDE.



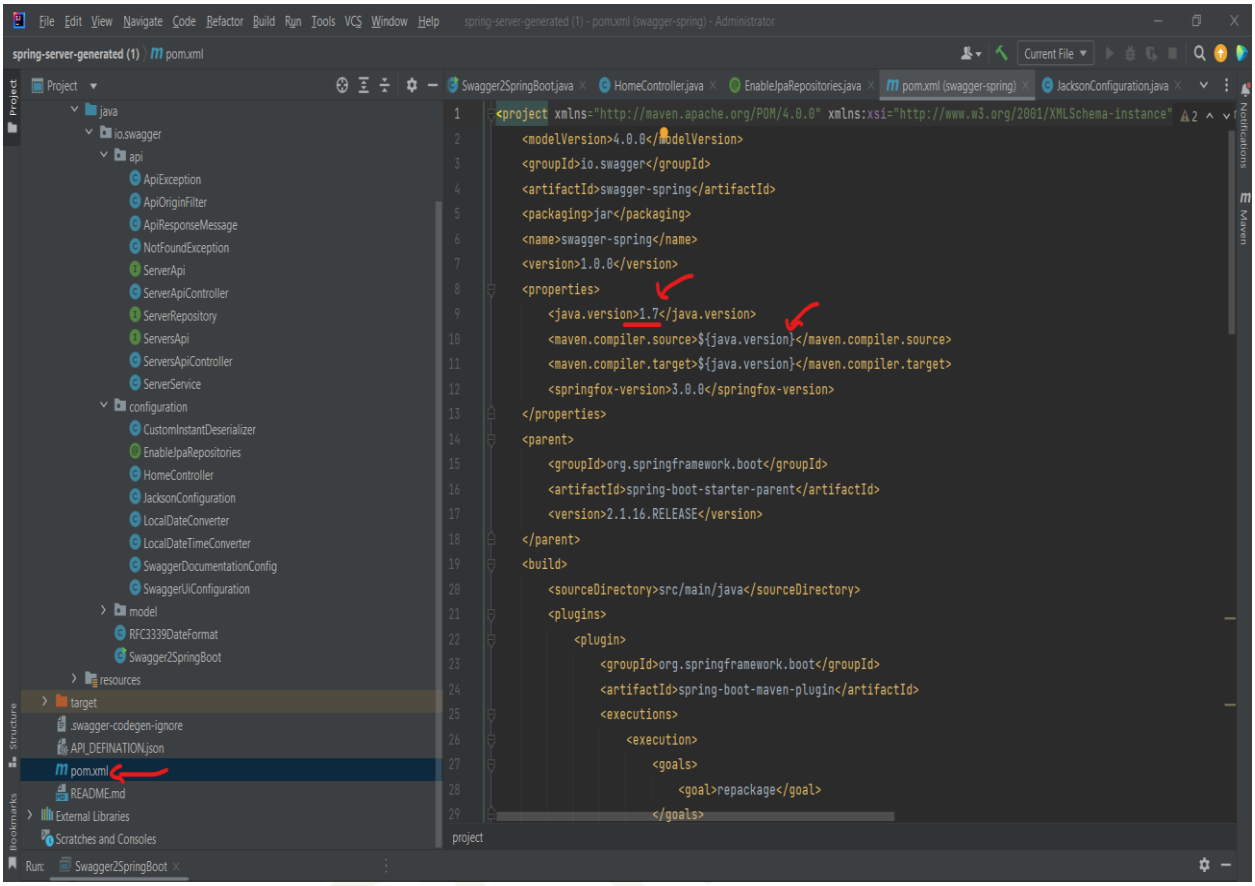
Modifications Made to the Auto-Generated file:

- This ***auto-generated project*** has all the ***Api(s), Data Models and Structure*** as needed. However, the ***dependencies Like Spring-Web, Lombok*** and other dependencies have to ***manually added*** for the project to ***run***.
  (All of this is Auto-generated from API Definition the crossed files will be explained later they are used to ***handle server requests***):



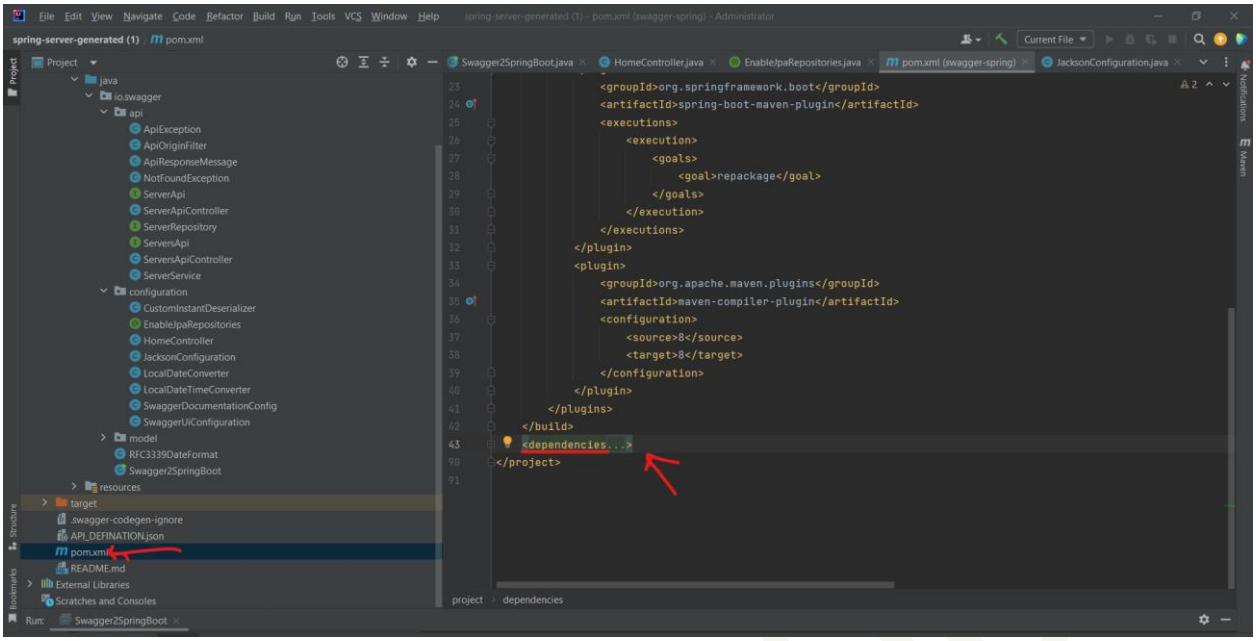
- Otherwise, it gives errors as the ***dependencies are missing***.

- Also, we need to make sure that the ***java and maven version*** running on your system and that reflected in the ***Pom.xml*** file are the same.



(I Degraded My Java IDE version from 15.0.7 to 1.7 as the generated file have that version)

- ***Addition of all the Dependencies*** are as follows:
  - Go to "***pom.xml***". Ex: ("***C:\Users\ArpitSG\Downloads\spring-server-generated (1)\pom.xml***").
  - Go under the ***< dependencies> </ dependencies> Section*** and all the ***required dependencies.***



- **.Spring Framework (Dependency):**

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
</dependency>
```

- **SpringFox dependencies:**

```
<!--SpringFox dependencies -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-oas</artifactId>
    <version>${springfox-version}</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>${springfox-version}</version>
</dependency>
```

- **Jackson (Dependency):**

```
<dependency>
    <groupId>com.github.joschi.jackson</groupId>
    <artifactId>jackson-datatype-threetenbp</artifactId>
    <version>2.6.4</version>
</dependency>
```

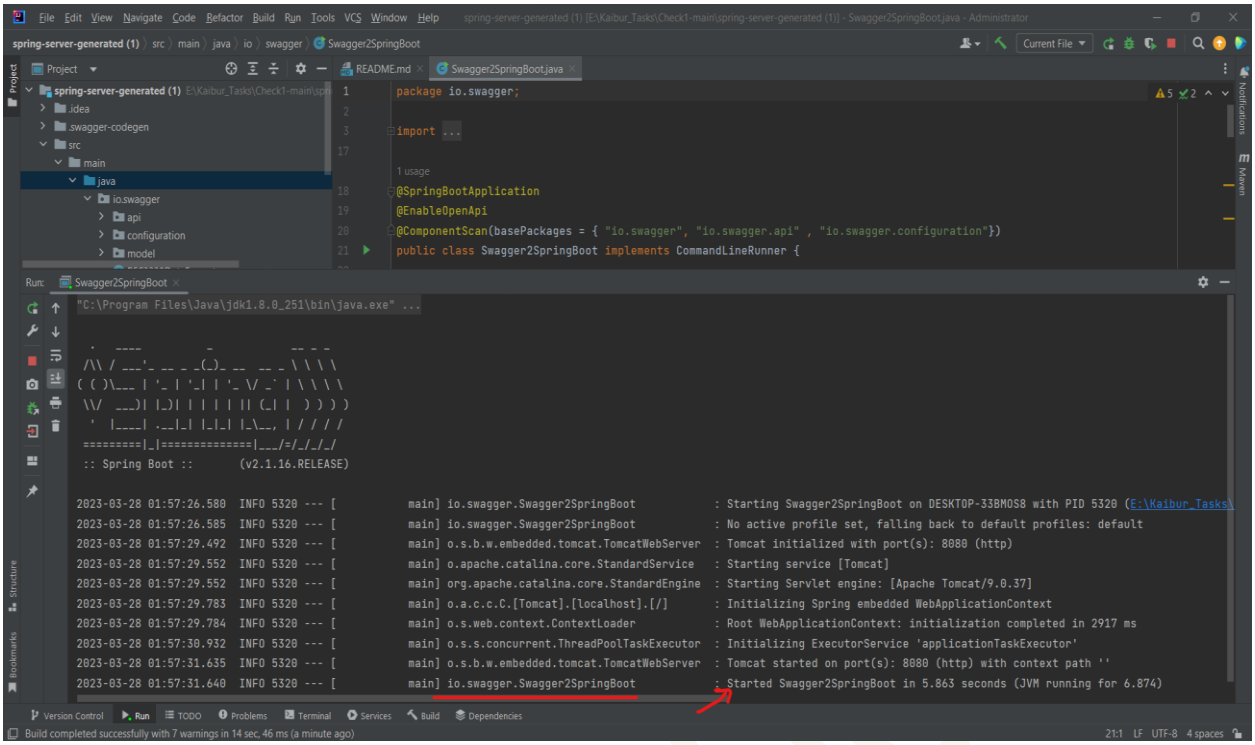
- **Bean Validation (Dependency):**

```
<!-- Bean Validation API support -->
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
</dependency>
```

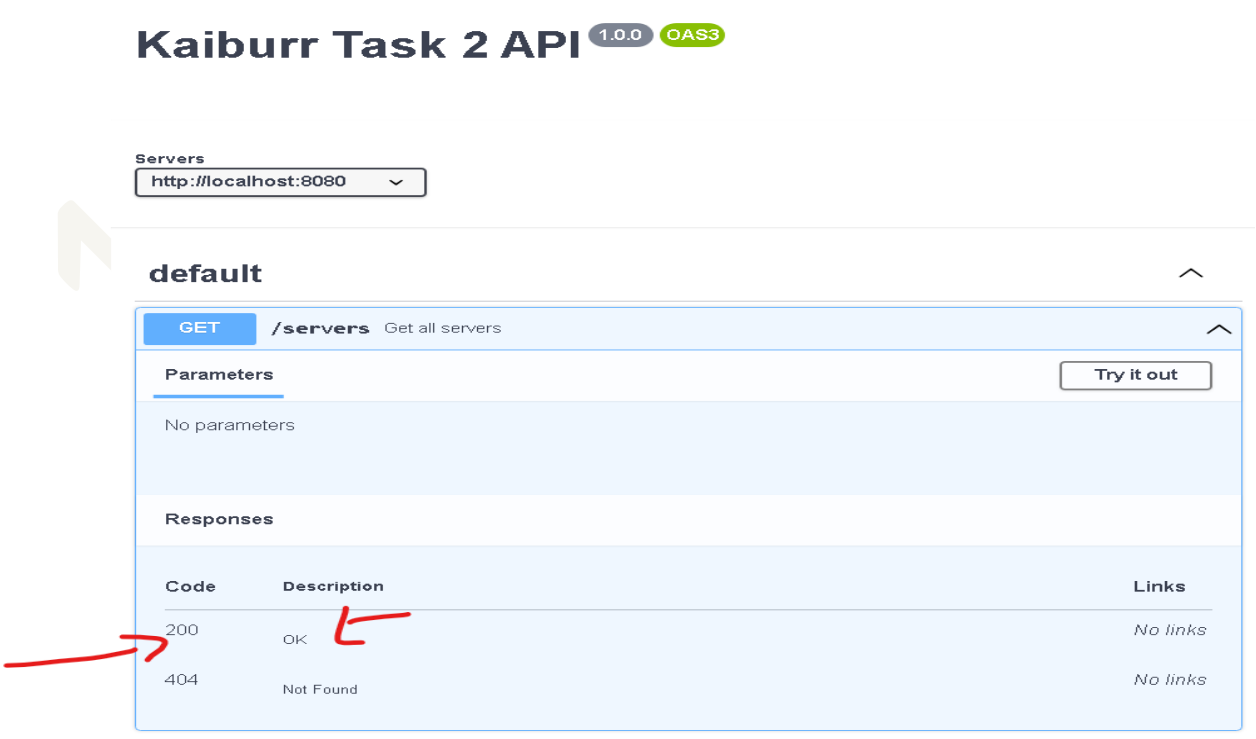
- **Spring Boot and Framework Plugins (Dependency):**

```
<dependency>
    <groupId>org.springframework.plugin</groupId>
    <artifactId>spring-plugin-core</artifactId>
    <version>2.0.0.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
</dependencies>
</project>
```

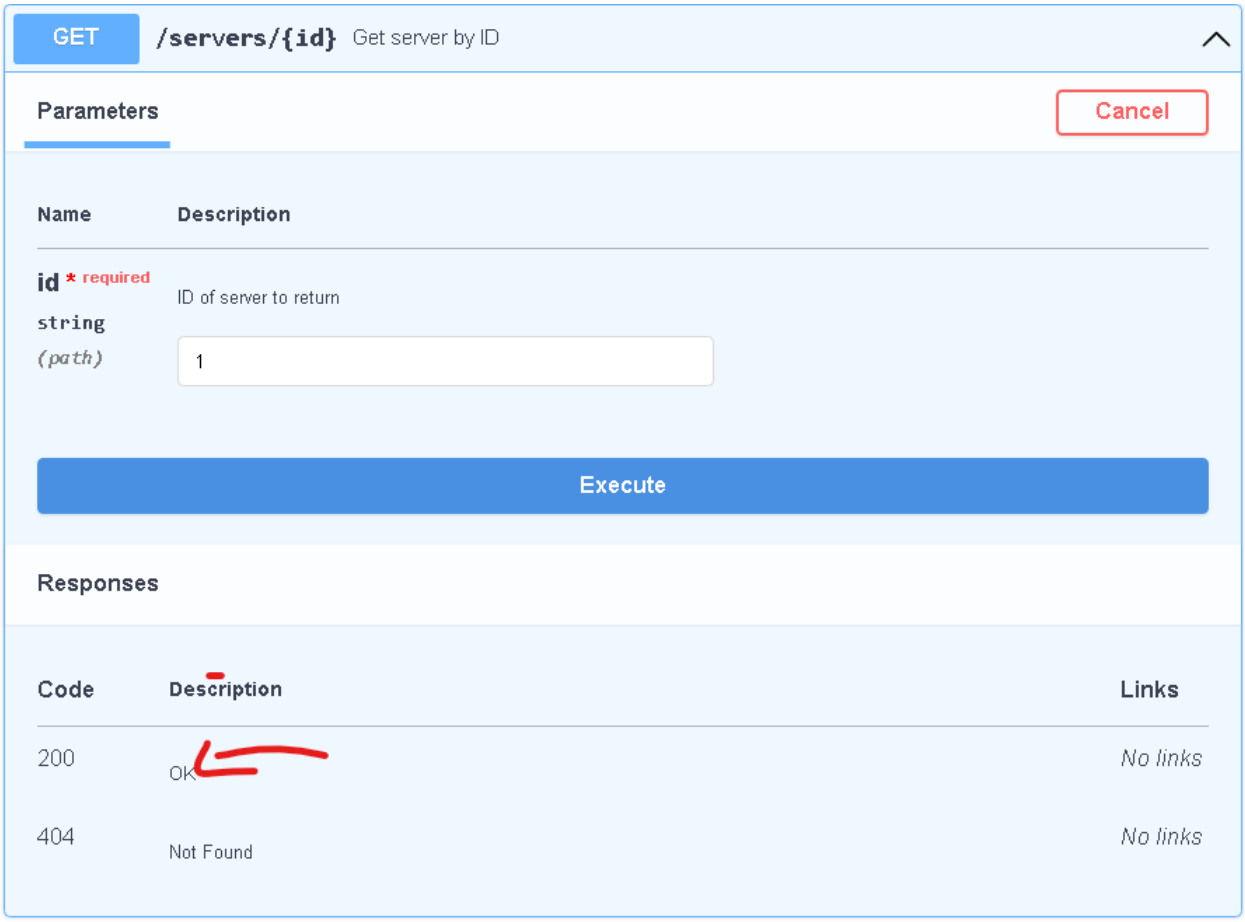
- Now after adding all these dependencies in "***pom.xml***". The "***Swagger2SpringBoot***" class that is our ***Rest-Application*** can be ***compiled and runned*** successfully.
- Like in ***Task 1***:
  - The ***console log(s)*** will finish with below means the server has successfully started. (keep in mind that ***server port: 8080 should be free***)



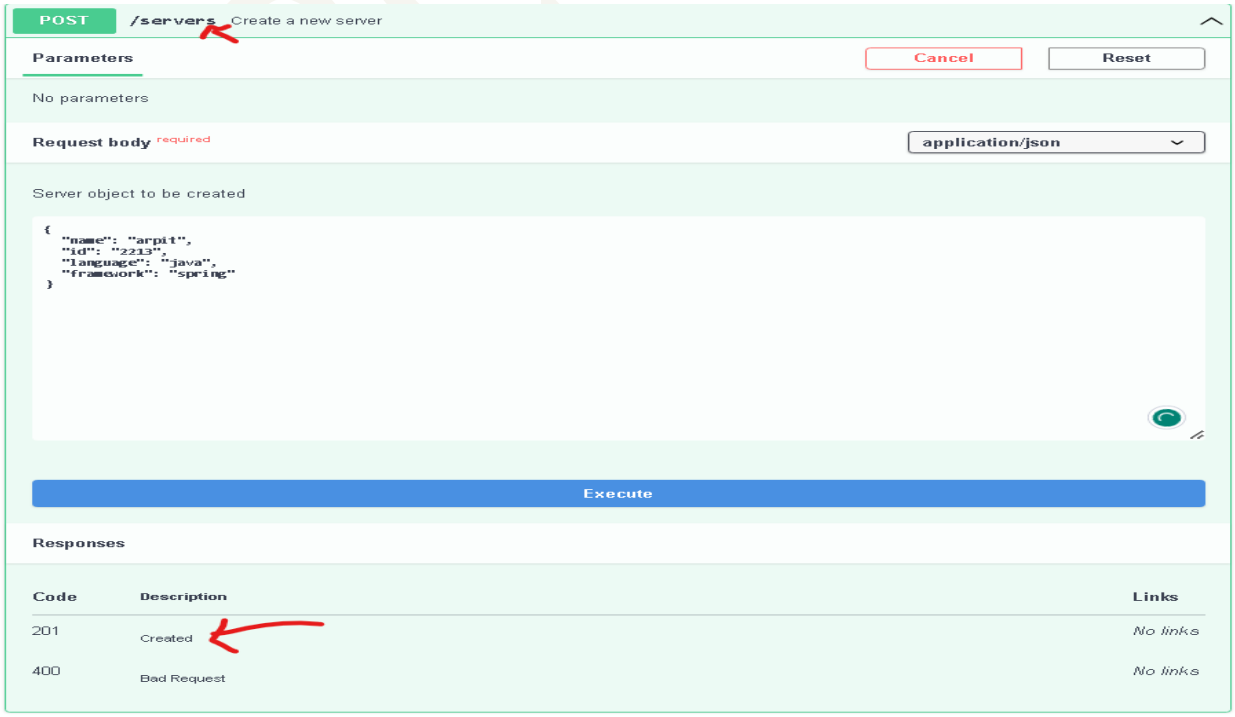
- However, this auto-generated Spring-boot Application developed with Swagger ***cannot handle requests*** as the ***Api(s) and Controllers have definitions that only respond with HTTP Code(s).*** Like ***200 Ok 201 Created etc***. They didn't actually modify or updated the database/Server.
    a. ***To get all servers:***



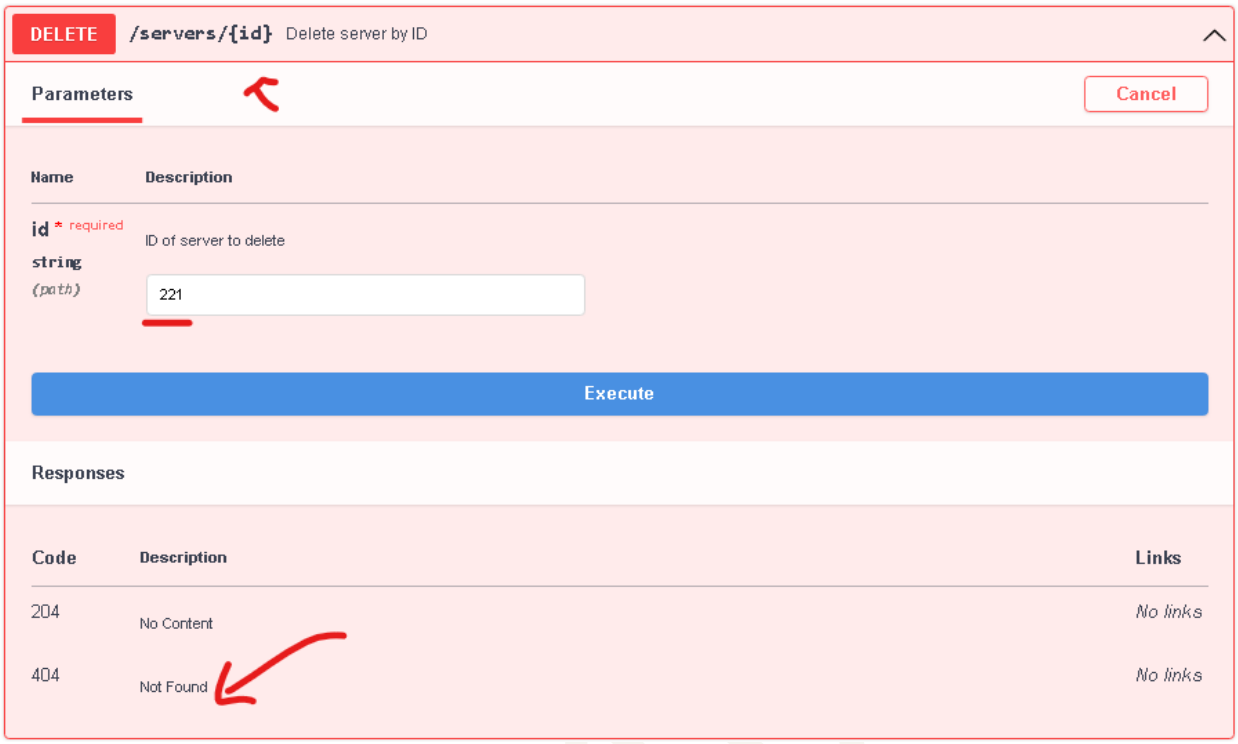
b. *To get a single server by ID:*



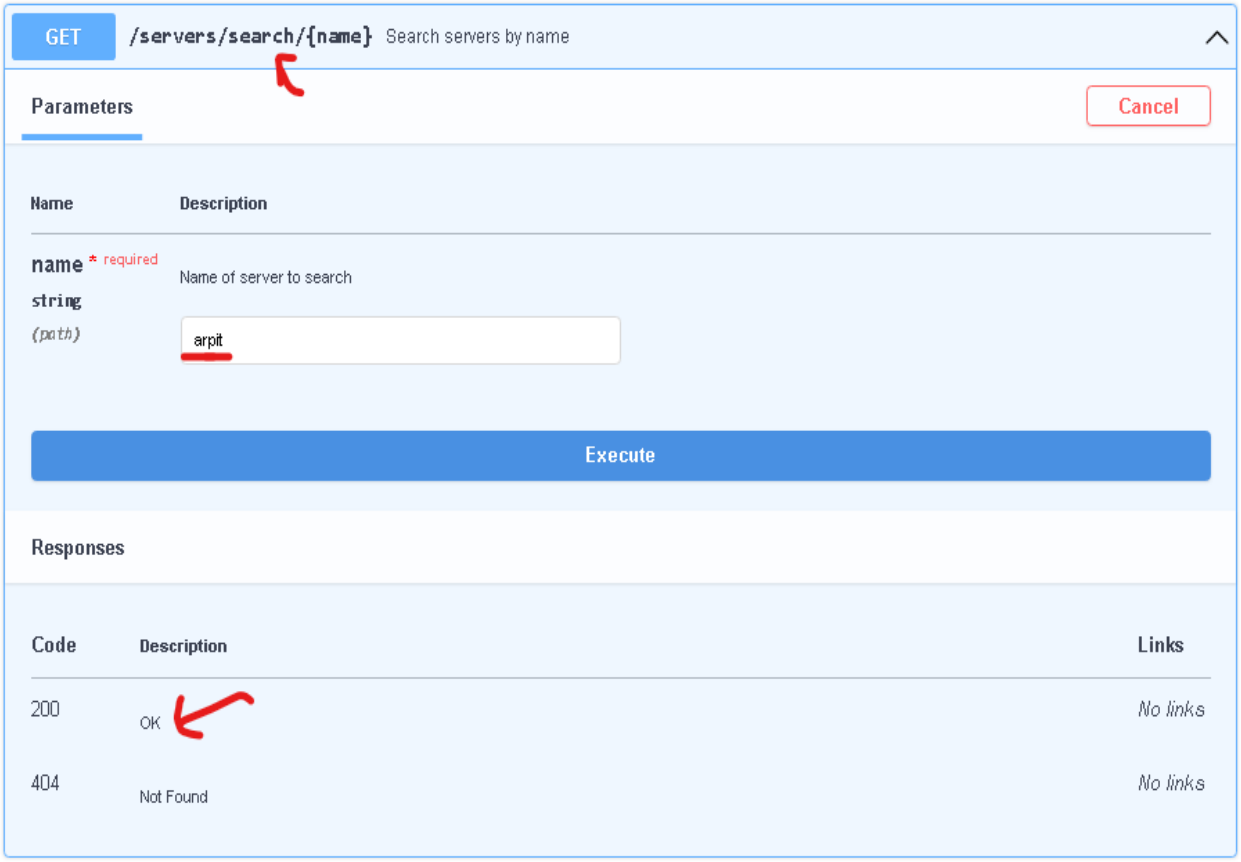
c. *To create a new server:*



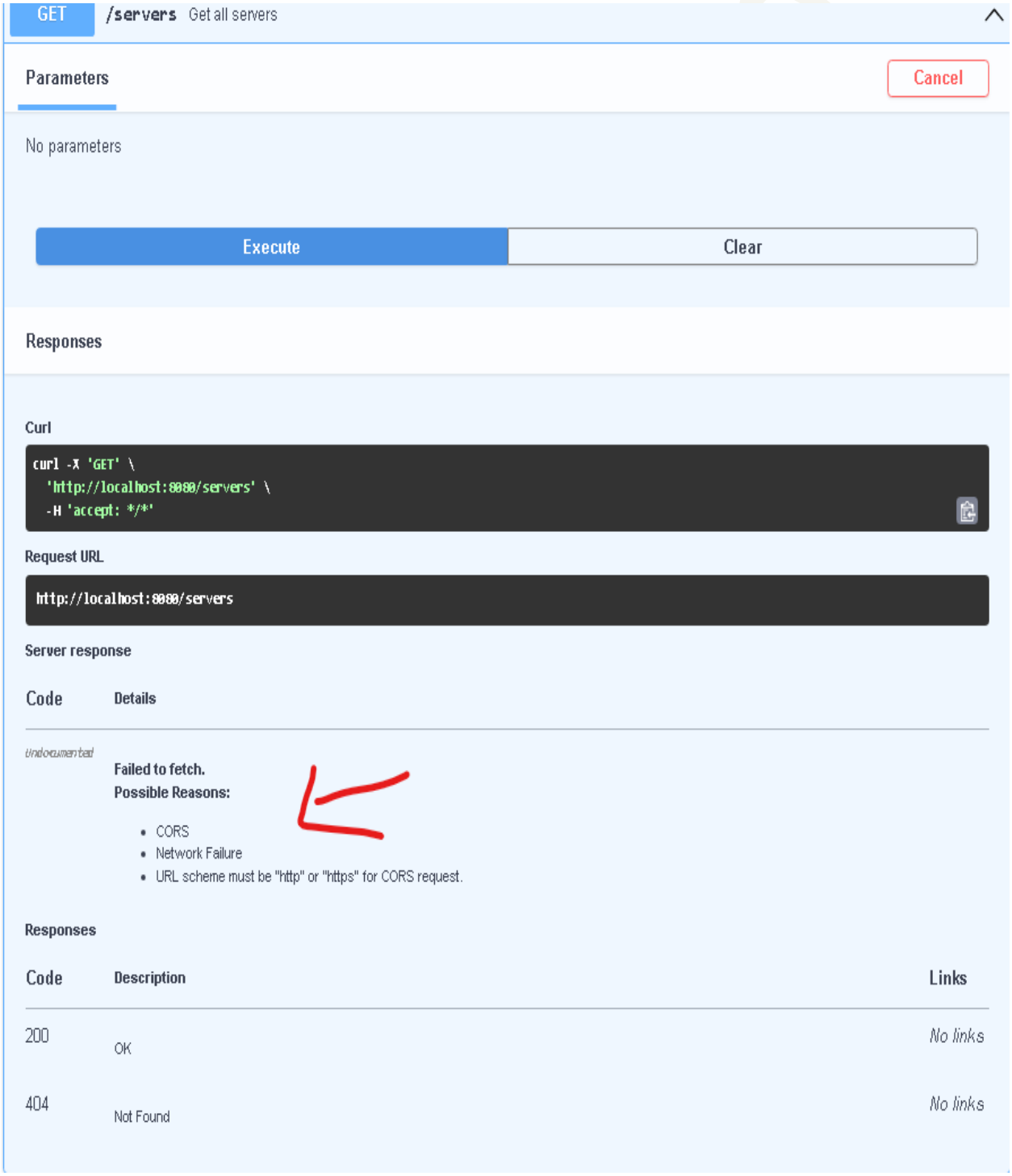
d. To delete a server by ID:



e. To search for servers by name:



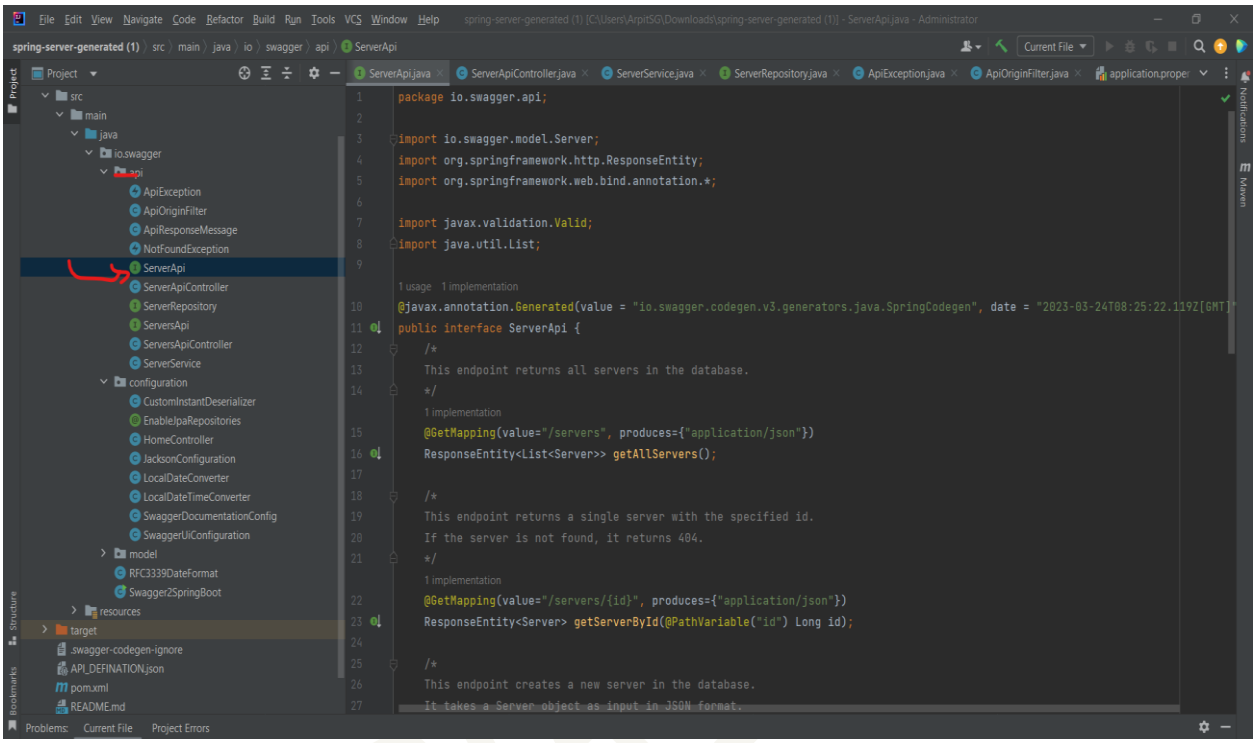
- Here, the ***problem*** is that we haven't provided the ***actual API endpoint implementation***. That's why whatever the ***default code generated*** by swagger only handles request and shows **HTTP code as output** but we want to make changes in the actual ***database and retrieve and update information*** according to the requests. For that we will have ***to implement new interface and classes*** so that it can handle requests as we want.

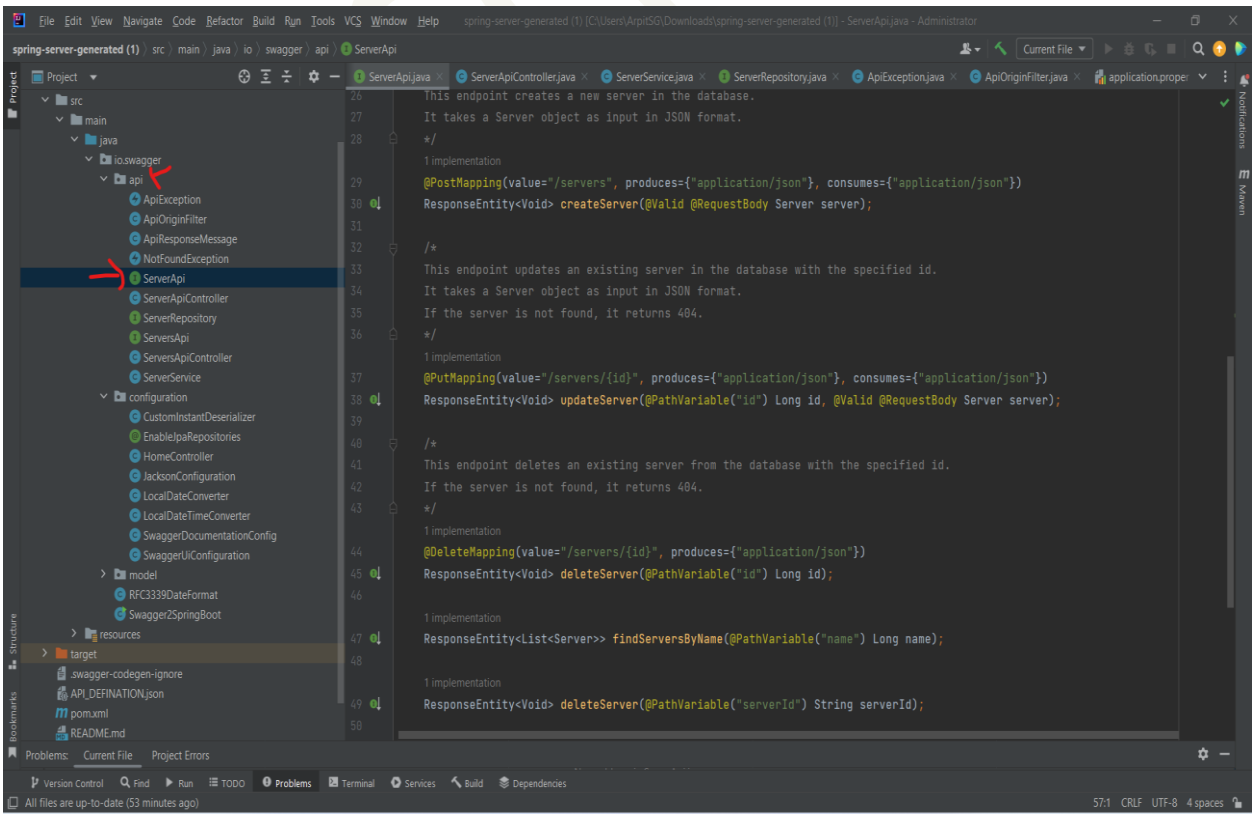


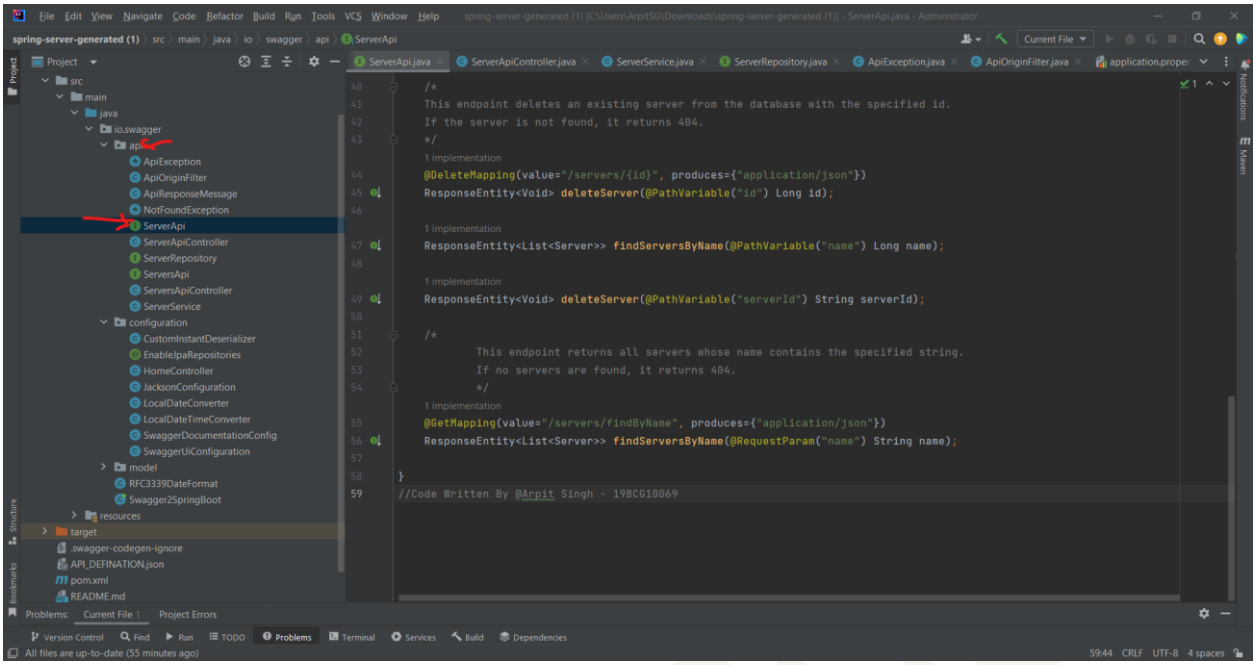
6. Modification/Implementation of API(s):

a. Go to "***C:\Users\ArpitSG\Downloads\spring-server-generated (1)\src\main\java\io\swagger\api***".

b. In this "***api***" package add a Interface called "***ServeApi***".



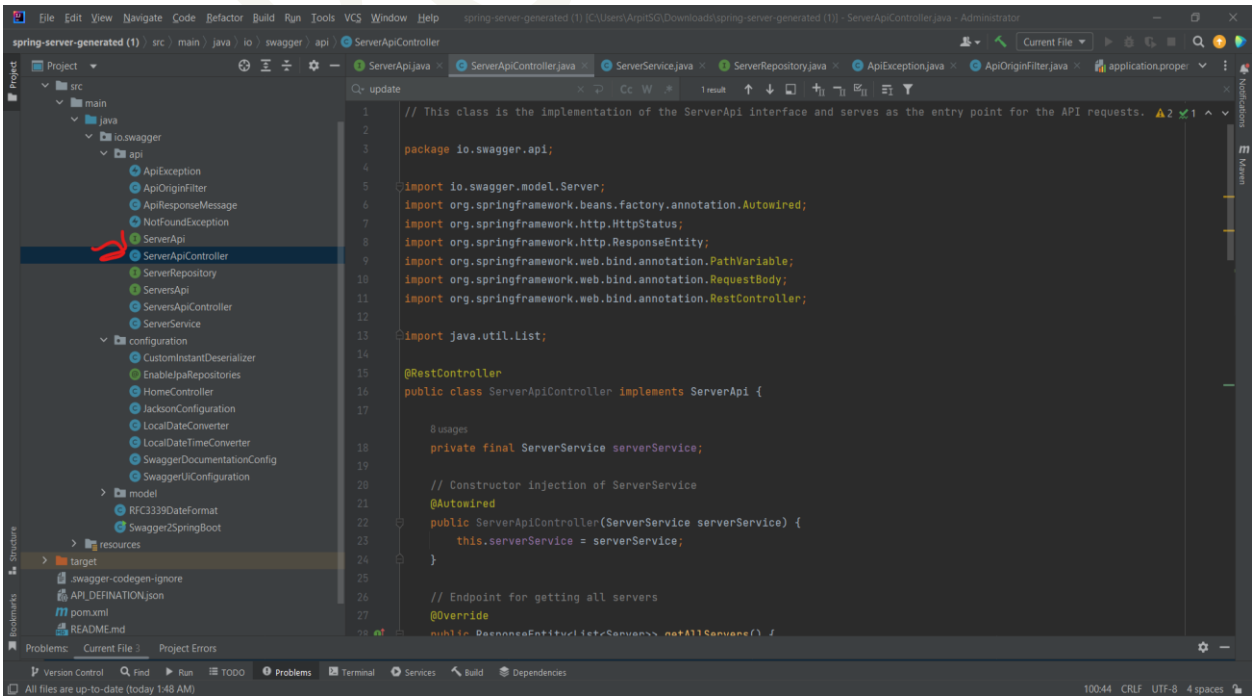


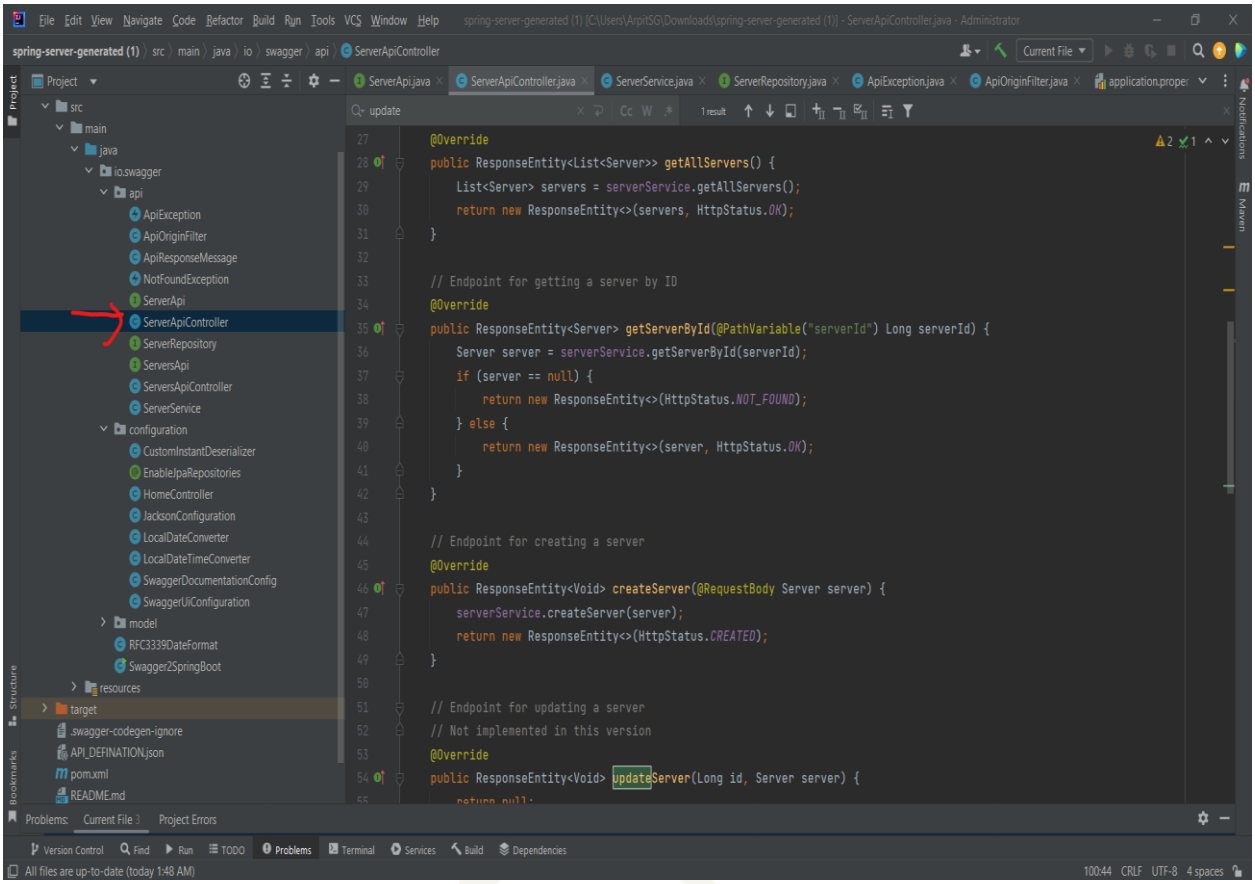


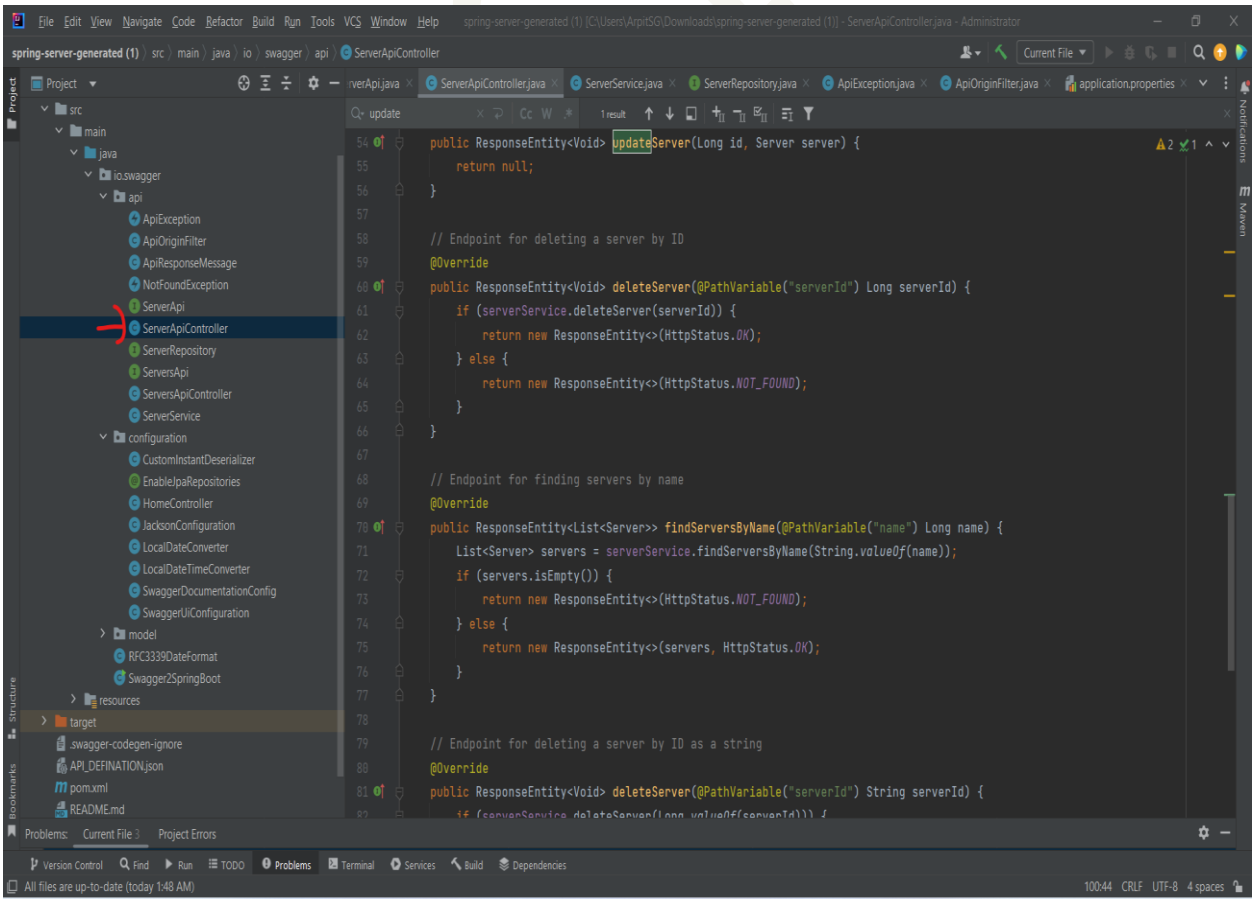
Explanation (Code):

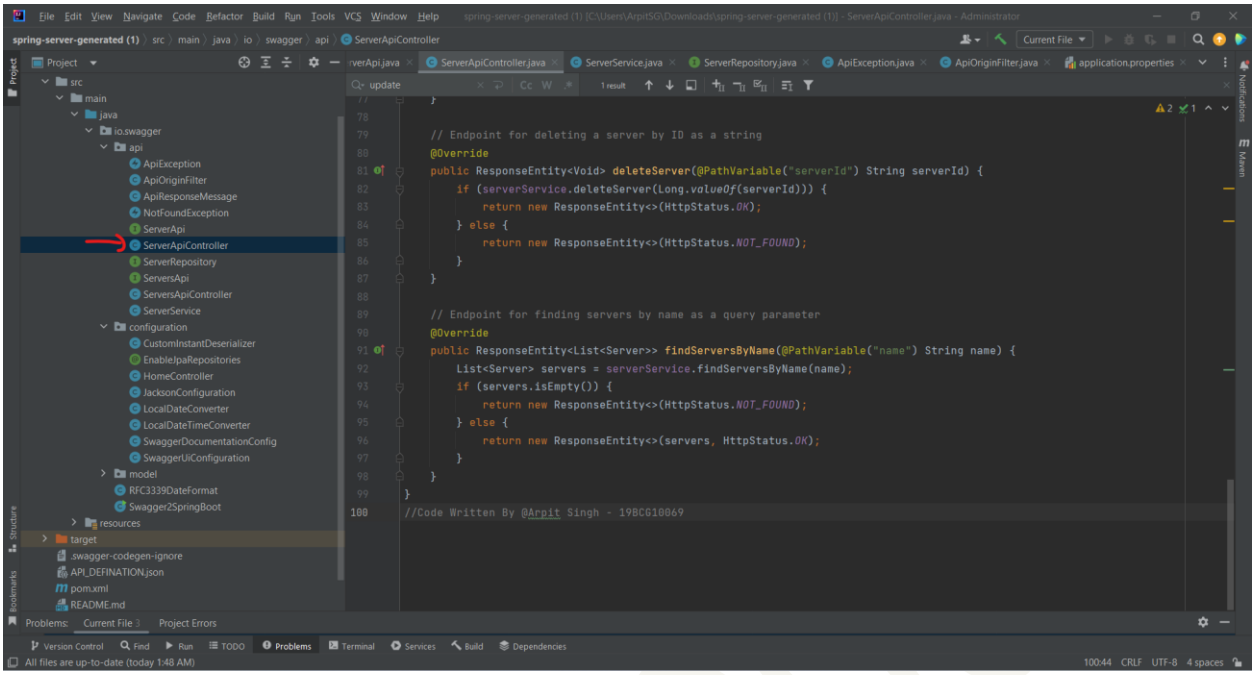
- This file contains the "***API interface***", and you will need to add the new method signatures for the new APIs. You can create methods like '***getAllServers(), getServerById(), createServer(), updateServer(), deleteServer(), and findServersByName()'.***

c. In this "***api***" package add a class called "***ServerApiController***".




File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
spring-server-generated (1) [C:\Users\ArpitSG\Downloads\spring-server-generated (1)] - ServerApiController.java - Administrator
spring-server-generated (1) src main java io swagger api ServerApiController
Current File
Project
ServerApi.java
ServerApiController.java
ServerService.java
ServerRepository.java
ApiException.java
ApiOriginFilter.java
application.proper
update
1 result
src
main
java
io.swagger
api
ApiException
ApiOriginFilter
ApiResponseMessage
NotFoundException
ServerApi
ServerApiController
ServerRepository
ServersApi
ServersApiController
ServerService
configuration
CustomInstantDeserializer
EnableJpaRepositories
HomeController
JacksonConfiguration
LocalDateConverter
LocalDateTimeConverter
SwaggerDocumentationConfig
SwaggerUiConfiguration
model
RFC3339DateFormat
Swagger2SpringBoot
resources
target
.swagger-codegen-ignore
API_DEFINATION.json
pom.xml
README.md
@Override
public ResponseEntity<List<Server>> getAllServers() {
List<Server> servers = serverService.getAllServers();
return new ResponseEntity<>(servers, HttpStatus.OK);
}
// Endpoint for getting a server by ID
@Override
public ResponseEntity<Server> getServerById(@PathVariable("serverId") Long serverId) {
Server server = serverService.getServerById(serverId);
if (server == null) {
return new ResponseEntity<>(HttpStatus.NOT_FOUND);
} else {
return new ResponseEntity<>(server, HttpStatus.OK);
}
}
// Endpoint for creating a server
@Override
public ResponseEntity<Void> createServer(@RequestBody Server server) {
serverService.createServer(server);
return new ResponseEntity<>(HttpStatus.CREATED);
}
// Endpoint for updating a server
// Not implemented in this version
@Override
public ResponseEntity<Void> updateServer(Long id, Server server) {
Problems: Current File 3 Project Errors
Version Control Find Run TODO Problems Terminal Services Build Dependencies
All files are up-to-date (today 1:48 AM)
100:44 CRLF UTF-8 4 spaces


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
spring-server-generated (1) [C:\Users\ArpitSG\Downloads\spring-server-generated (1)] - ServerApiController.java - Administrator
spring-server-generated (1) src main java io swagger api ServerApiController
Current File
ServerApiController.java
ServerService.java
ServerRepository.java
ApiException.java
ApiOriginFilter.java
application.properties
update
1 result
public ResponseEntity<Void> updateServer(Long id, Server server) {
return null;
}
// Endpoint for deleting a server by ID
@Override
public ResponseEntity<Void> deleteServer(@PathVariable("serverId") Long serverId) {
if (serverService.deleteServer(serverId)) {
return new ResponseEntity<>(HttpStatus.OK);
} else {
return new ResponseEntity<>(HttpStatus.NOT_FOUND);
}
}
// Endpoint for finding servers by name
@Override
public ResponseEntity<List<Server>> findServersByName(@PathVariable("name") Long name) {
List<Server> servers = serverService.findServersByName(String.valueOf(name));
if (servers.isEmpty()) {
return new ResponseEntity<>(HttpStatus.NOT_FOUND);
} else {
return new ResponseEntity<>(servers, HttpStatus.OK);
}
}
// Endpoint for deleting a server by ID as a string
@Override
public ResponseEntity<Void> deleteServer(@PathVariable("serverId") String serverId) {
Problems: Current File 3 Project Errors
Version Control Find Run TODO Problems Terminal Services Build Dependencies
All files are up-to-date (today 1:48 AM)
100:44 CRLF UTF-8 4 spaces

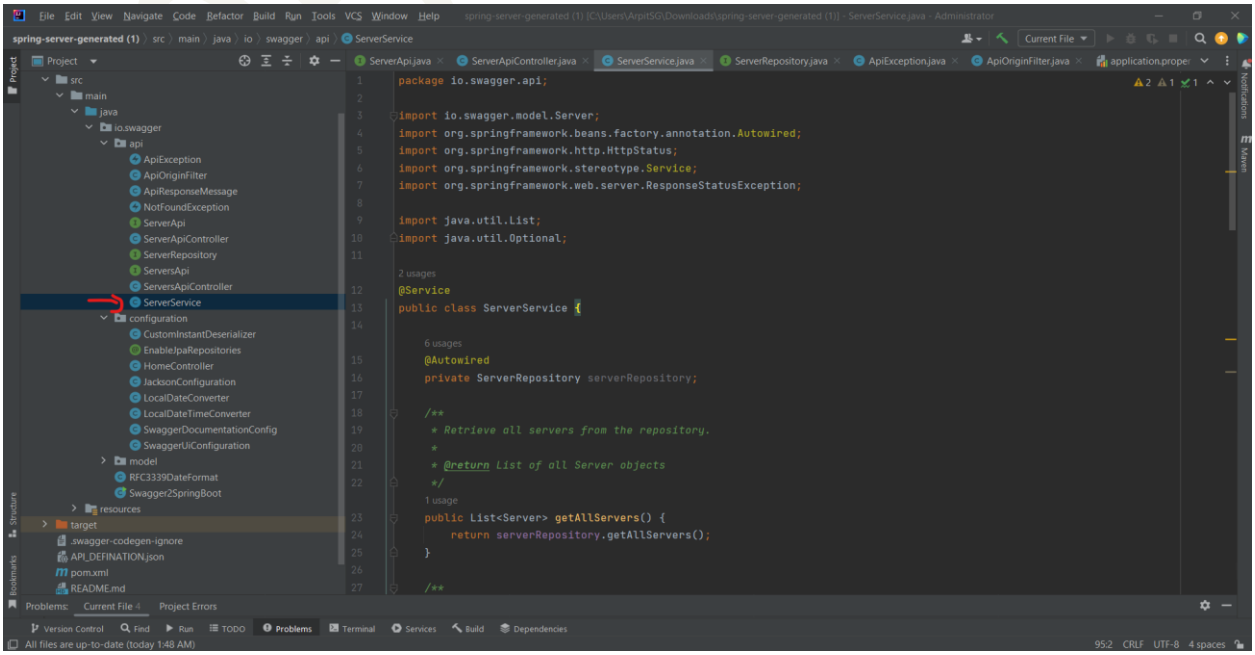


```
    }

    // Endpoint for deleting a server by ID as a string
    @Override
    public ResponseEntity<Void> deleteServer(@PathVariable("serverId") String serverId) {
        if (serverService.deleteServer(Long.valueOf(serverId))) {
            return new ResponseEntity<>(HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    // Endpoint for finding servers by name as a query parameter
    @Override
    public ResponseEntity<List<Server>> findServersByName(@PathVariable("name") String name) {
        List<Server> servers = serverService.findServersByName(name);
        if (servers.isEmpty()) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        } else {
            return new ResponseEntity<>(servers, HttpStatus.OK);
        }
    }
}
//Code Written By @Arpit Singh - 19BCG10069
```

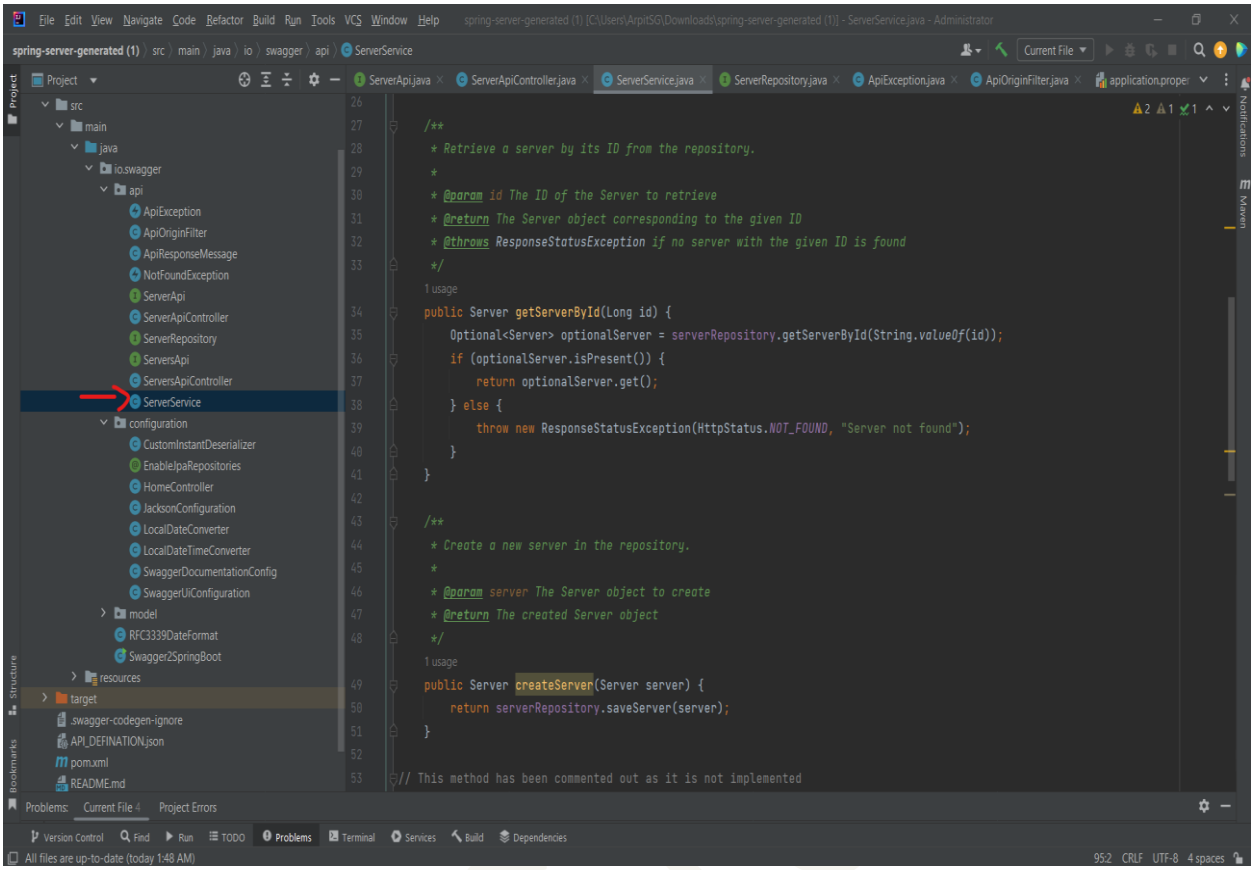
Explanation (Code):

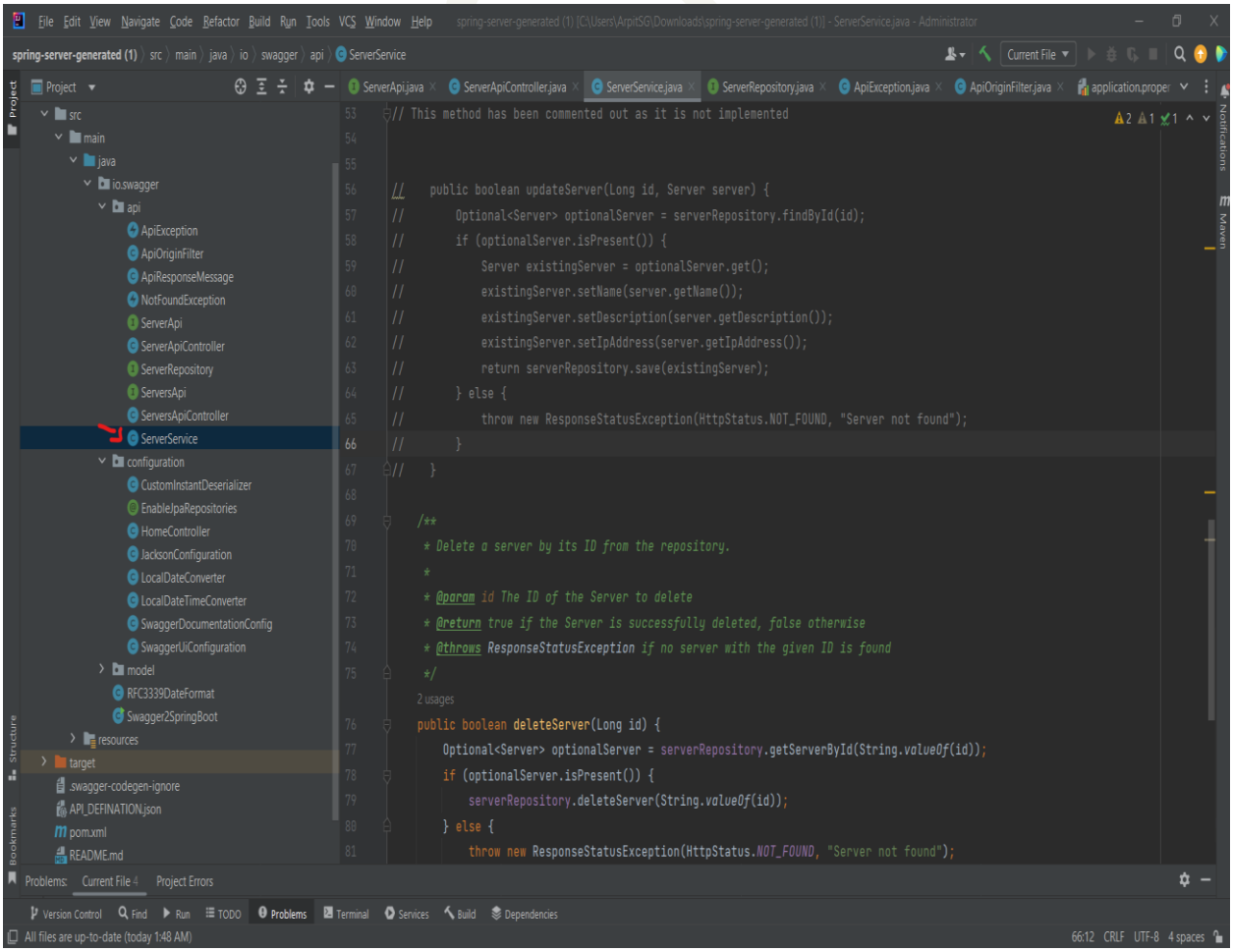
- This file contains the controller that implements the ***API interface, and you will need to implement the new methods***. You can use the logic from ***Task 1 to implement the new methods***.
- Note that this implementation assumes that there is a "***ServerService***" class that provides ***the business logic for the API methods***. You will need ***to create this class and implement the methods*** accordingly.

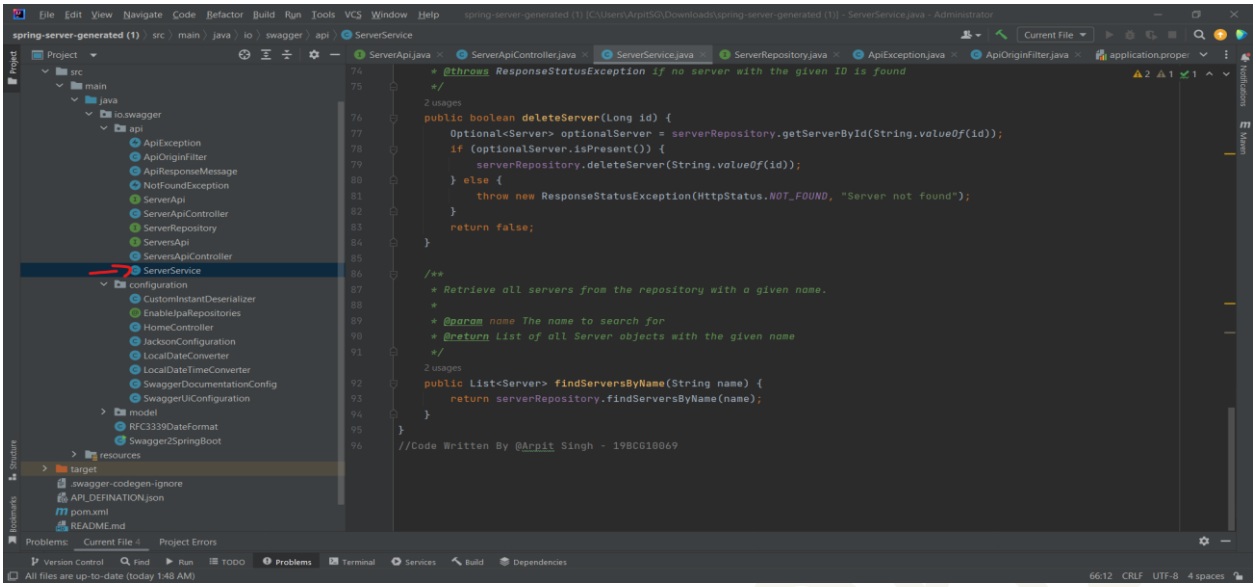
d. In this "***api***" package add an class called "***ServerService***":



```
package io.swagger.api;

import io.swagger.model.Server;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Service;
import org.springframework.web.server.ResponseStatusException;

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

2 usages
@Service
public class ServerService {

    6 usages
    @Autowired
    private ServerRepository serverRepository;

    /**
     * Retrieve all servers from the repository.
     *
     * @return List of all Server objects
     */
    1 usage
    public List<Server> getAllServers() {
        return serverRepository.getAllServers();
    }

    /**
```


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
spring-server-generated (1) [C:\Users\ArpitSG\Downloads\spring-server-generated (1)] - ServerService.java - Administrator
spring-server-generated (1) › src › main › java › io › swagger › api › ServerService
Current File
Project
ServerApi.java
ServerApiController.java
ServerService.java
ServerRepository.java
ApiException.java
ApiOriginFilter.java
application.proper
src
main
java
io.swagger
api
ApiException
ApiOriginFilter
ApiResponseMessage
NotFoundException
ServerApi
ServerApiController
ServerRepository
ServersApi
ServersApiController
ServerService
configuration
CustomInstantDeserializer
EnableJpaRepositories
HomeController
JacksonConfiguration
LocalDateConverter
LocalDateTimeConverter
SwaggerDocumentationConfig
SwaggerUiConfiguration
model
RFC3339DateFormat
Swagger2SpringBoot
resources
target
.swagger-codegen-ignore
API_DEFINATION.json
pom.xml
README.md
/**
* Retrieve a server by its ID from the repository.
*
* @param id The ID of the Server to retrieve
* @return The Server object corresponding to the given ID
* @throws ResponseStatusException if no server with the given ID is found
*/
1 usage
public Server getServerById(Long id) {
Optional<Server> optionalServer = serverRepository.getServerById(String.valueOf(id));
if (optionalServer.isPresent()) {
return optionalServer.get();
} else {
throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Server not found");
}
}
/**
* Create a new server in the repository.
*
* @param server The Server object to create
* @return The created Server object
*/
1 usage
public Server createServer(Server server) {
return serverRepository.saveServer(server);
}
// This method has been commented out as it is not implemented
Problems: Current File 4 Project Errors
Version Control Find Run TODO Problems Terminal Services Build Dependencies
All files are up-to-date (today 1:48 AM)
95:2 CRLF UTF-8 4 spaces

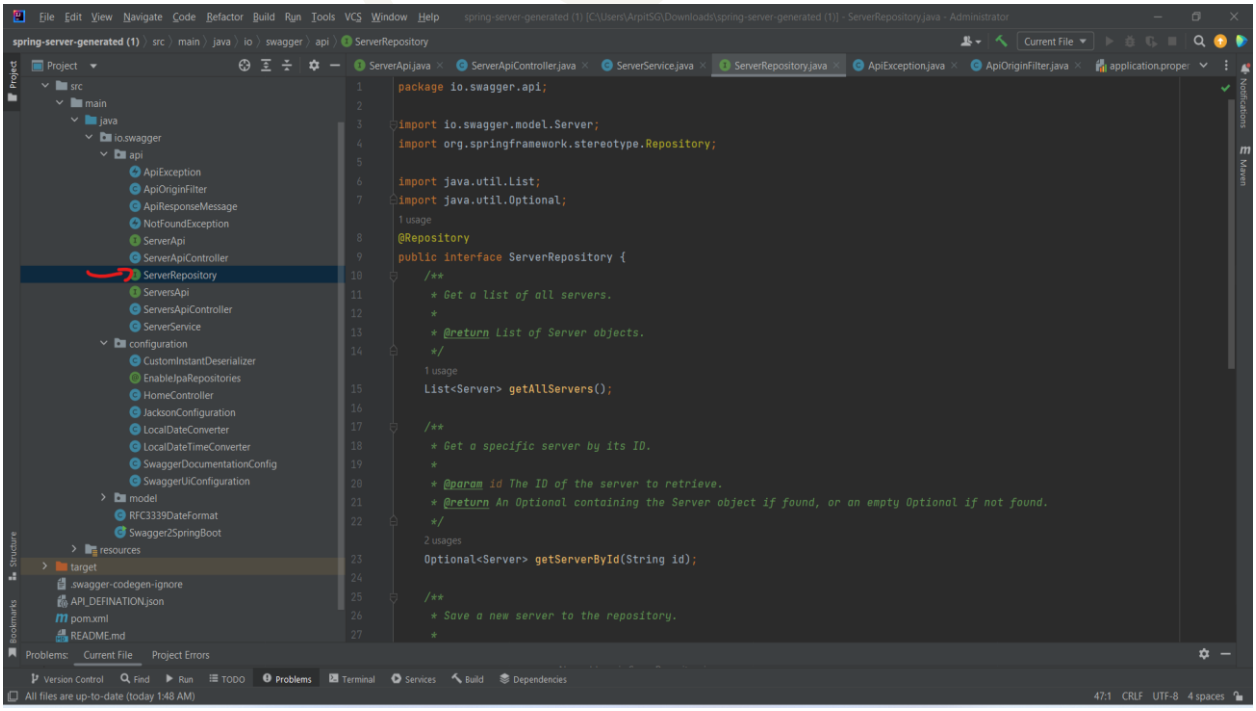

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
spring-server-generated (1) [C:\Users\ArpitSG\Downloads\spring-server-generated (1)] - ServerService.java - Administrator
spring-server-generated (1) › src › main › java › io › swagger › api › ServerService
Current File
// This method has been commented out as it is not implemented
// public boolean updateServer(Long id, Server server) {
// Optional<Server> optionalServer = serverRepository.findById(id);
// if (optionalServer.isPresent()) {
// Server existingServer = optionalServer.get();
// existingServer.setName(server.getName());
// existingServer.setDescription(server.getDescription());
// existingServer.setIpAddress(server.getIpAddress());
// return serverRepository.save(existingServer);
// } else {
// throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Server not found");
// }
// }
/**
* Delete a server by its ID from the repository.
*
* @param id The ID of the Server to delete
* @return true if the Server is successfully deleted, false otherwise
* @throws ResponseStatusException if no server with the given ID is found
*/
2 usages
public boolean deleteServer(Long id) {
Optional<Server> optionalServer = serverRepository.getServerById(String.valueOf(id));
if (optionalServer.isPresent()) {
serverRepository.deleteServer(String.valueOf(id));
} else {
throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Server not found");
Problems: Current File 4 Project Errors
Version Control Find Run TODO Problems Terminal Services Build Dependencies
All files are up-to-date (today 1:48 AM)
66:12 CRLF UTF-8 4 spaces

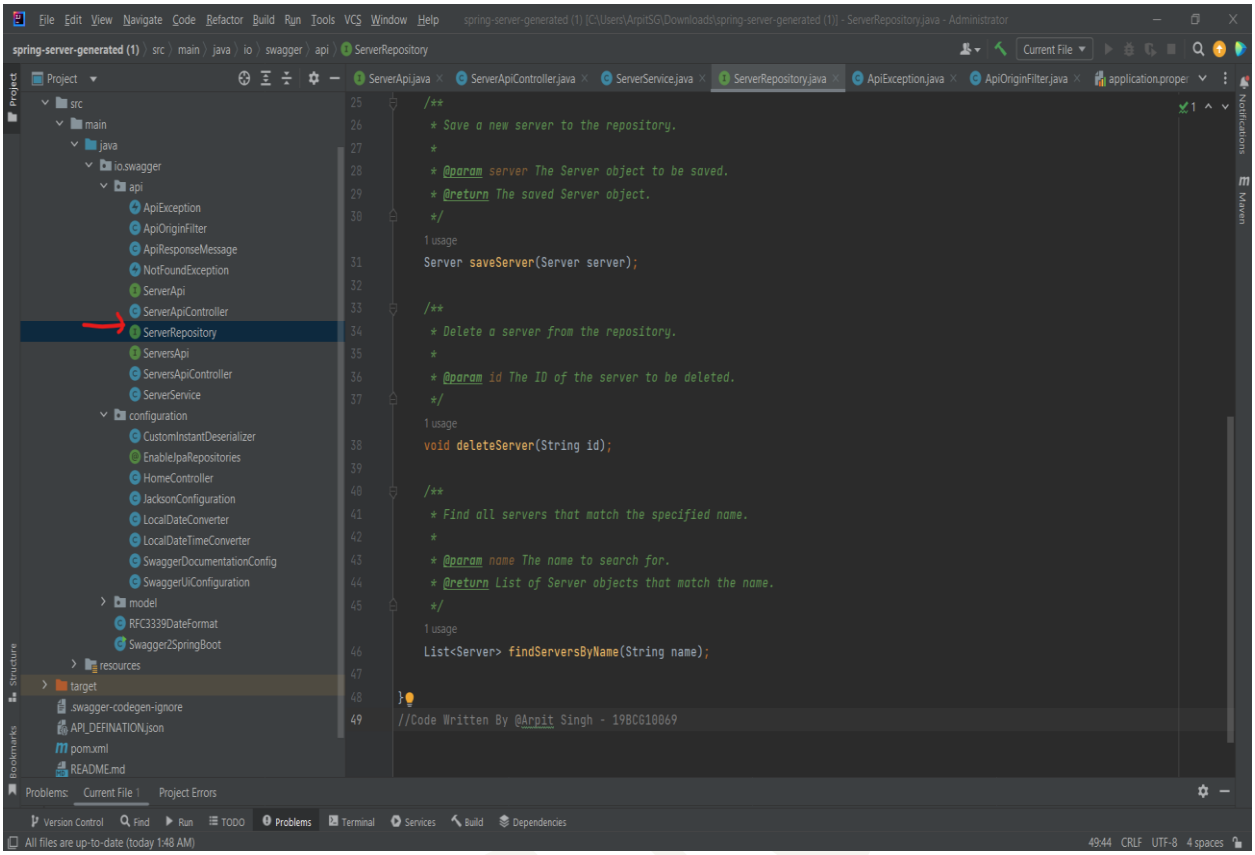


Explanation (Code):

- This implementation uses the "**ServerRepository**" interface to interact with the database and provides the logic for each of the new methods in the "**ServerApi**" interface. The implementation of ***getServerById(), updateServer(), and deleteServer()*** is similar to Task 1, while ***getAllServers()***, ***createServer()***, *and* ***findServersByName()*** are new methods.

    e. Lastly, in this "***api***" package add an Interface called "***ServerRepository***":



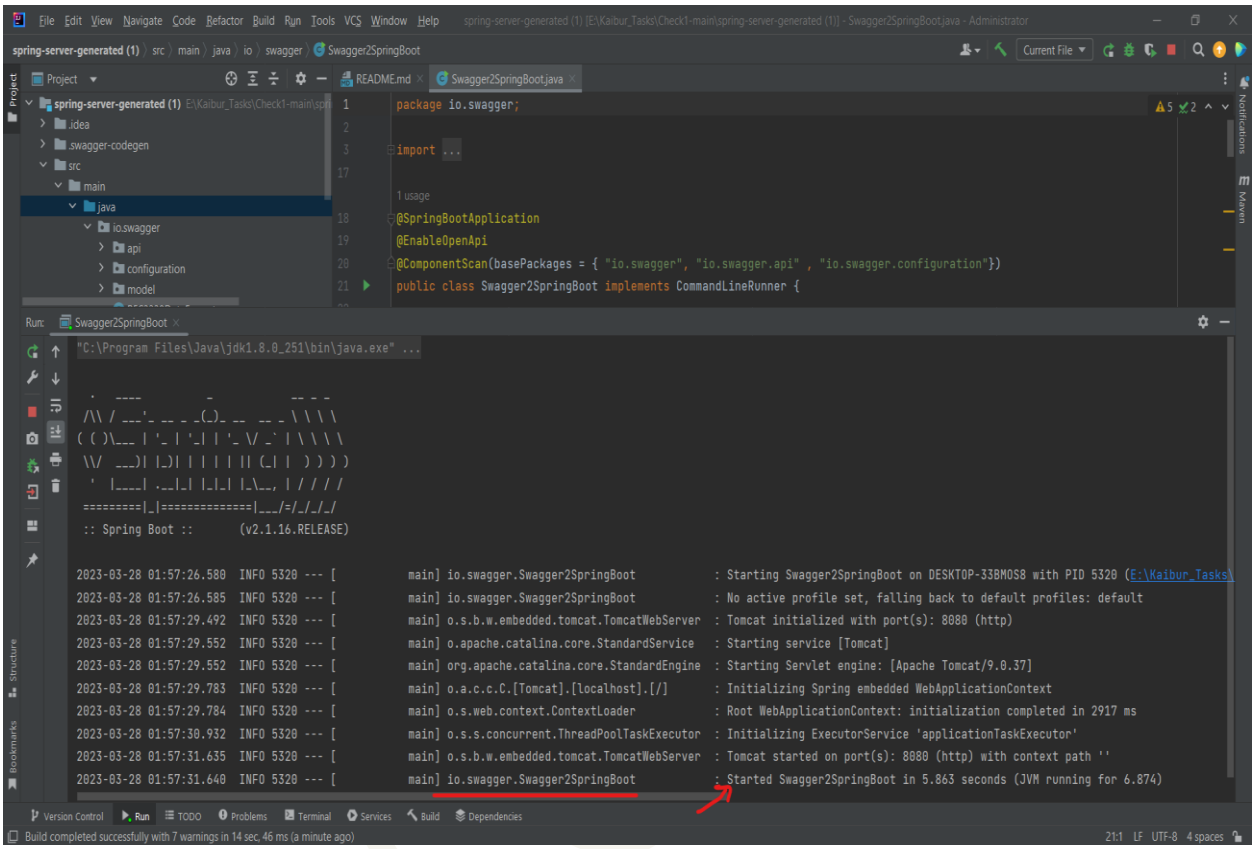


Explanation (Code):

- This includes the methods to ***get all servers, get a server by ID, save a server, delete a server by ID, and find servers by name.***

    f. ***After adding all these classes and interfaces make sure all the dependencies and class imports are done correctly and the '@RestController' Annotation has been added to the interfaces and classes to allow the changes to be reflected in the output.***

7. **Run the application**:
    a. Run the application by executing the "***Swagger2SpringBoot***" class.
    b. The ***console log(s)*** will finish with below means the server has successfully started.



Explanation (Code):

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

1. Open the '***Swagger2SpringBoot***' 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 Create a Java Server Code with API Definition for searching, creating, and deleting server objects using Swagger and Maven.**

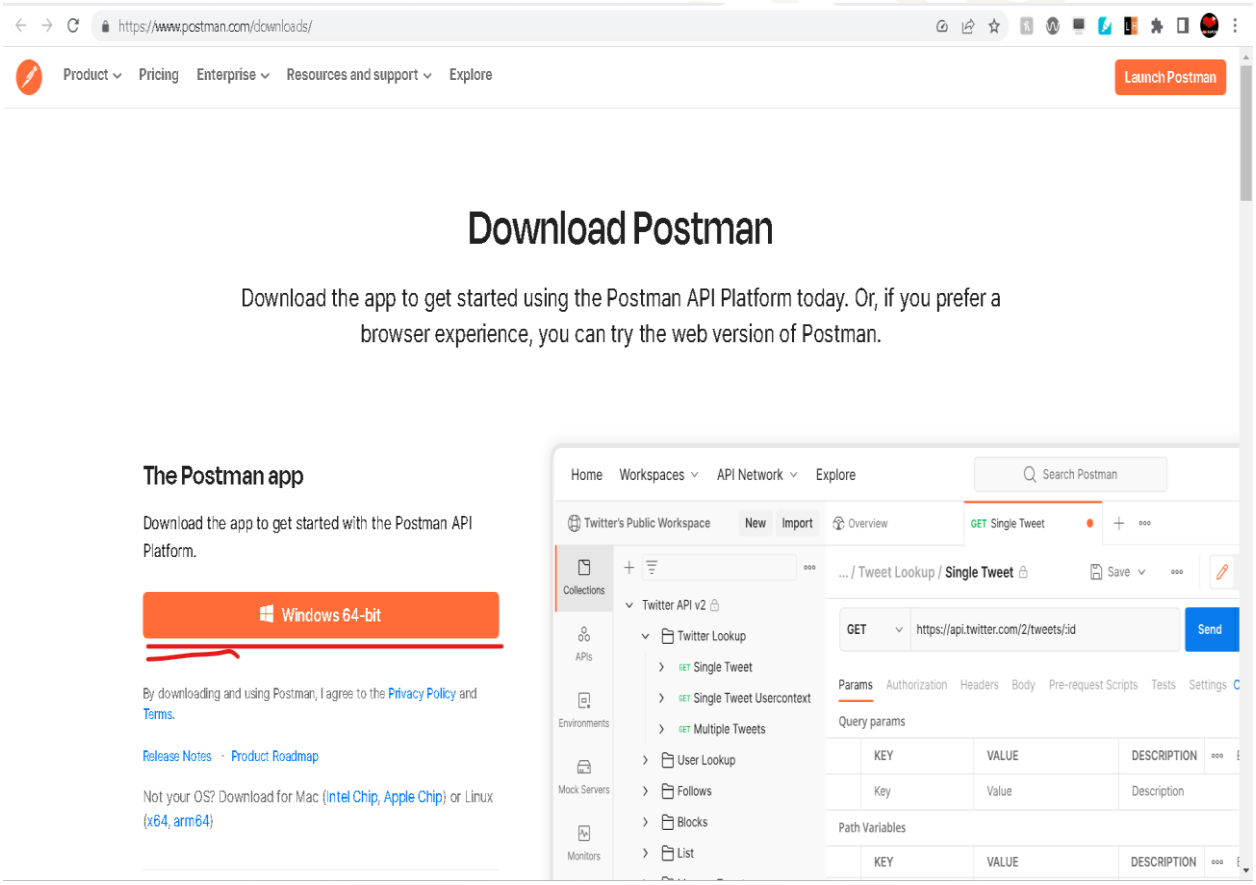
API Testing/Output:

- 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)

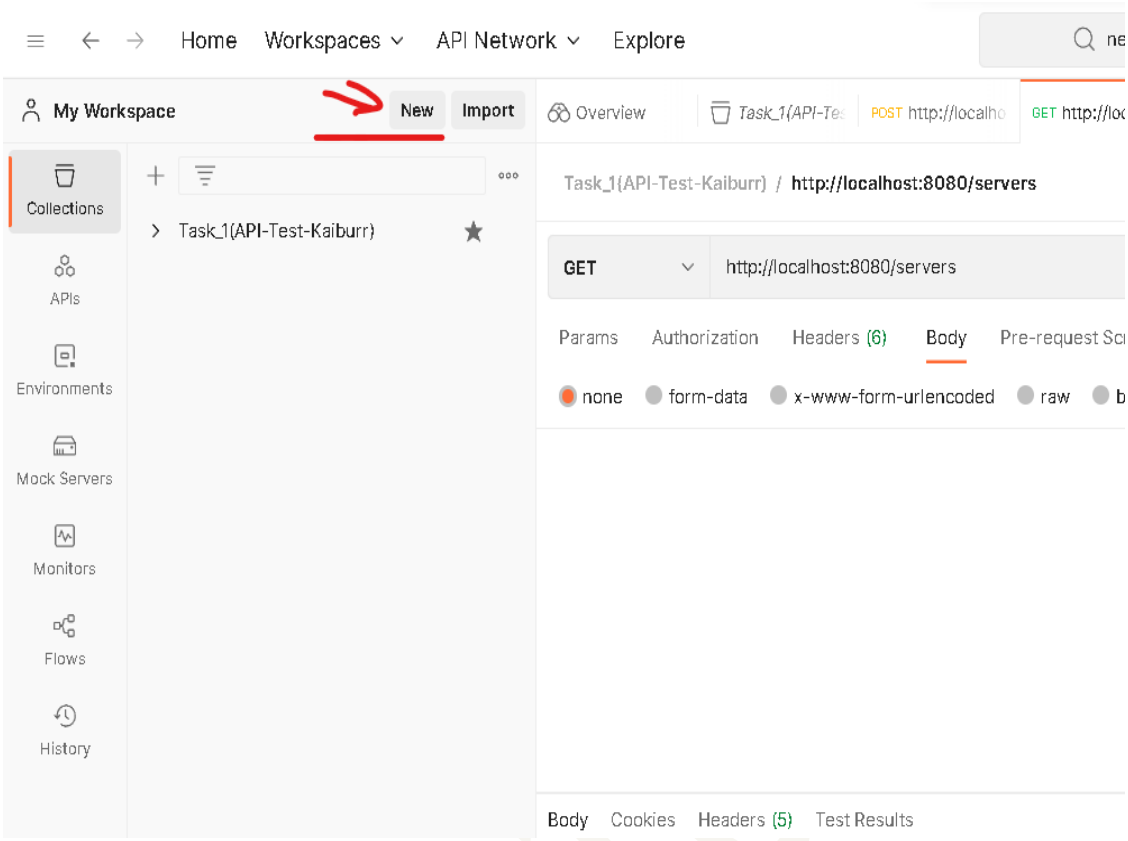
❖ How To Use POSTMAN:

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

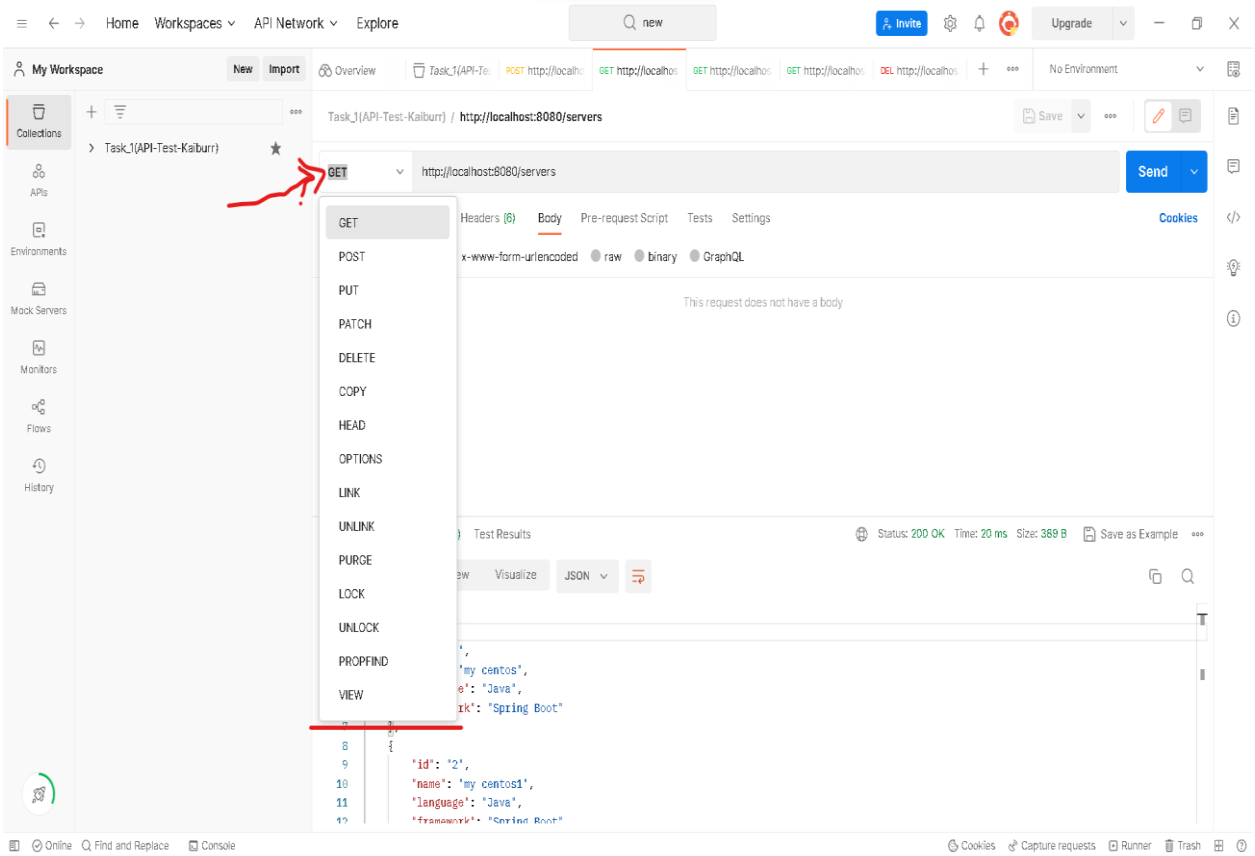
- Here are the steps to use Postman to test the API:
  1. Download and install Postman from **https://www.postman.com/downloads/**.



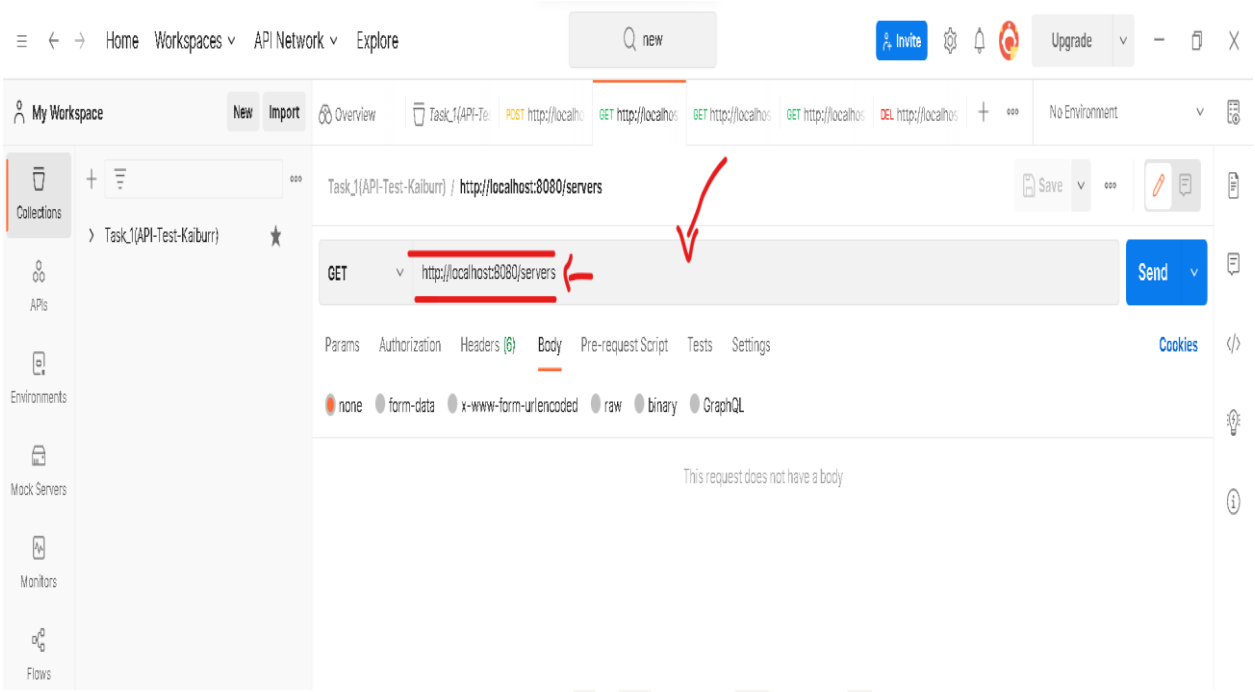
  2. Open Postman and click on the "***New***" button to create a new request.



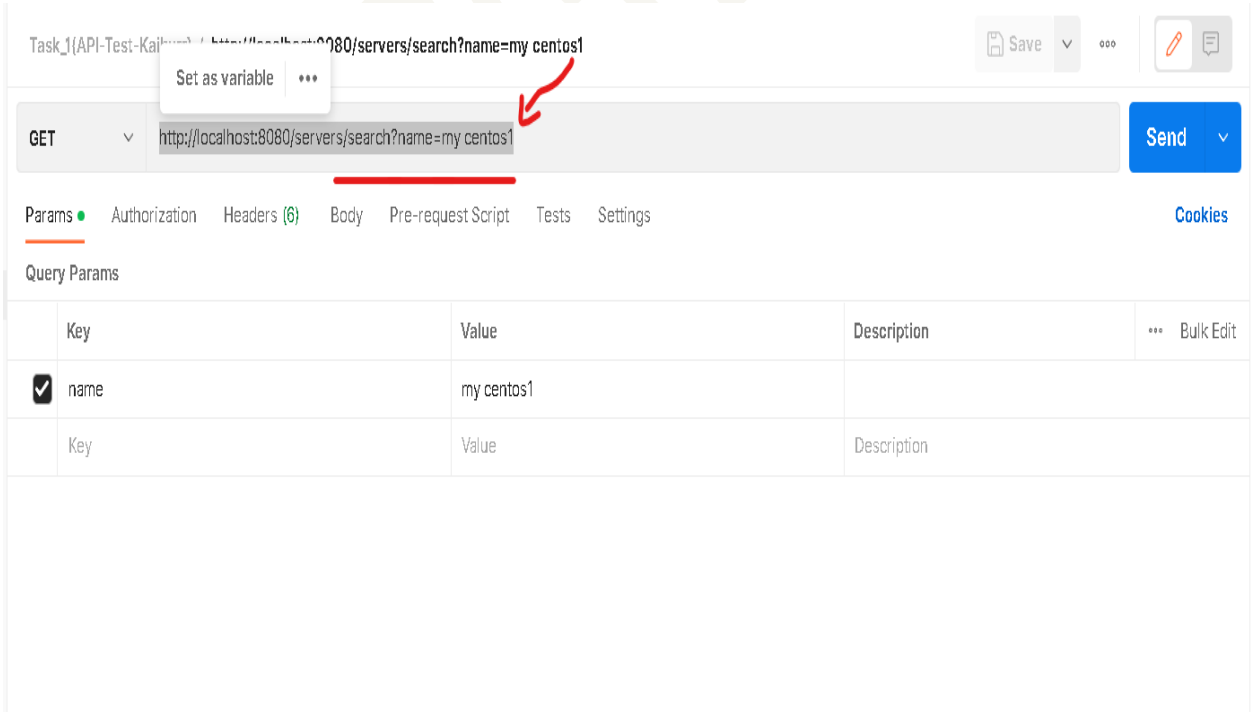
3. Select the ***HTTP method*** you want to use (GET, PUT, DELETE) from the dropdown list.



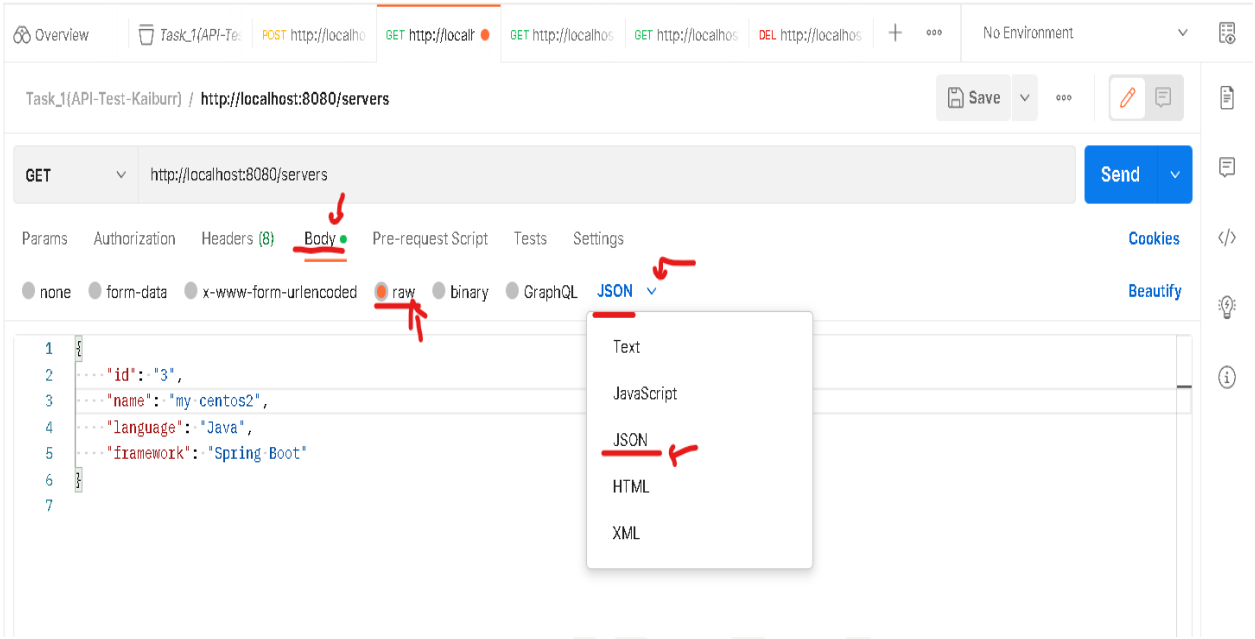
4. Enter the ***URL of the API endpoint*** you want to test (e.g. ***http://localhost:8080/servers***).



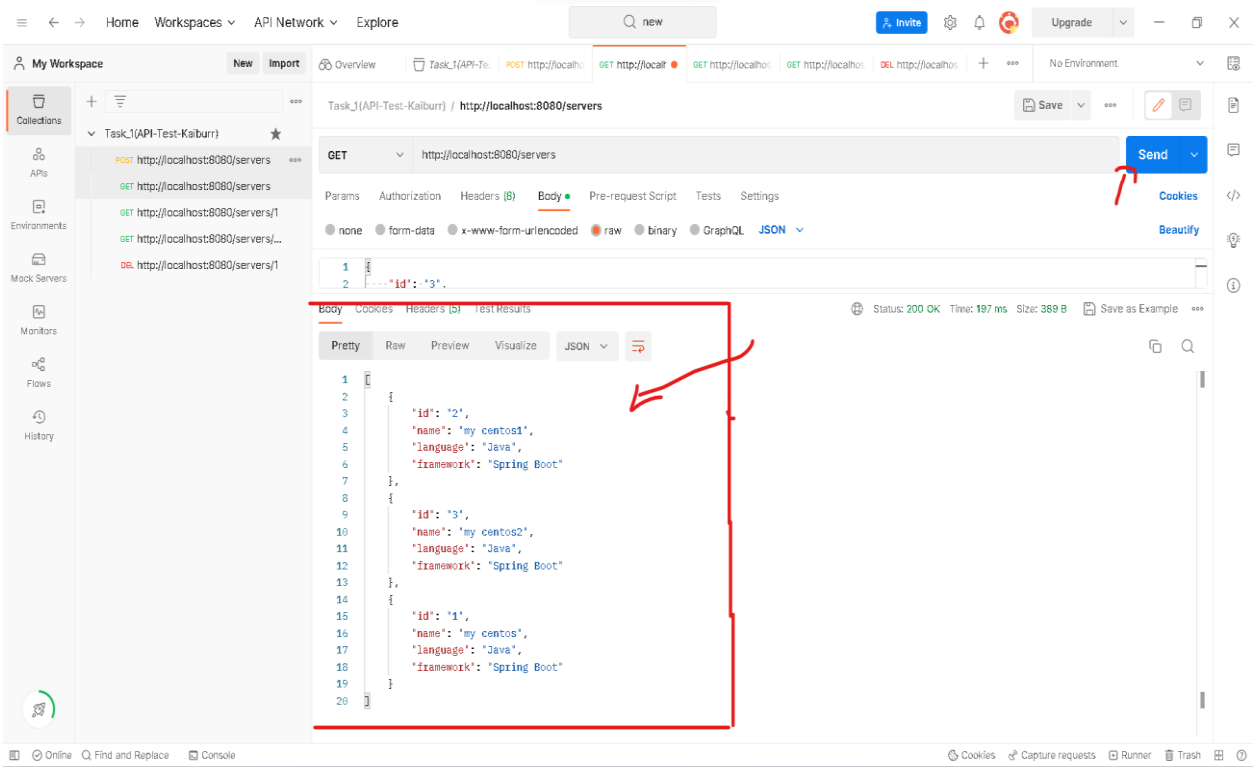
5. If you need ***to include parameters*** in your request (e.g. a server ID), you can ***add them to the URL*** or in the request body.



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.



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



➢ ***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***:

{

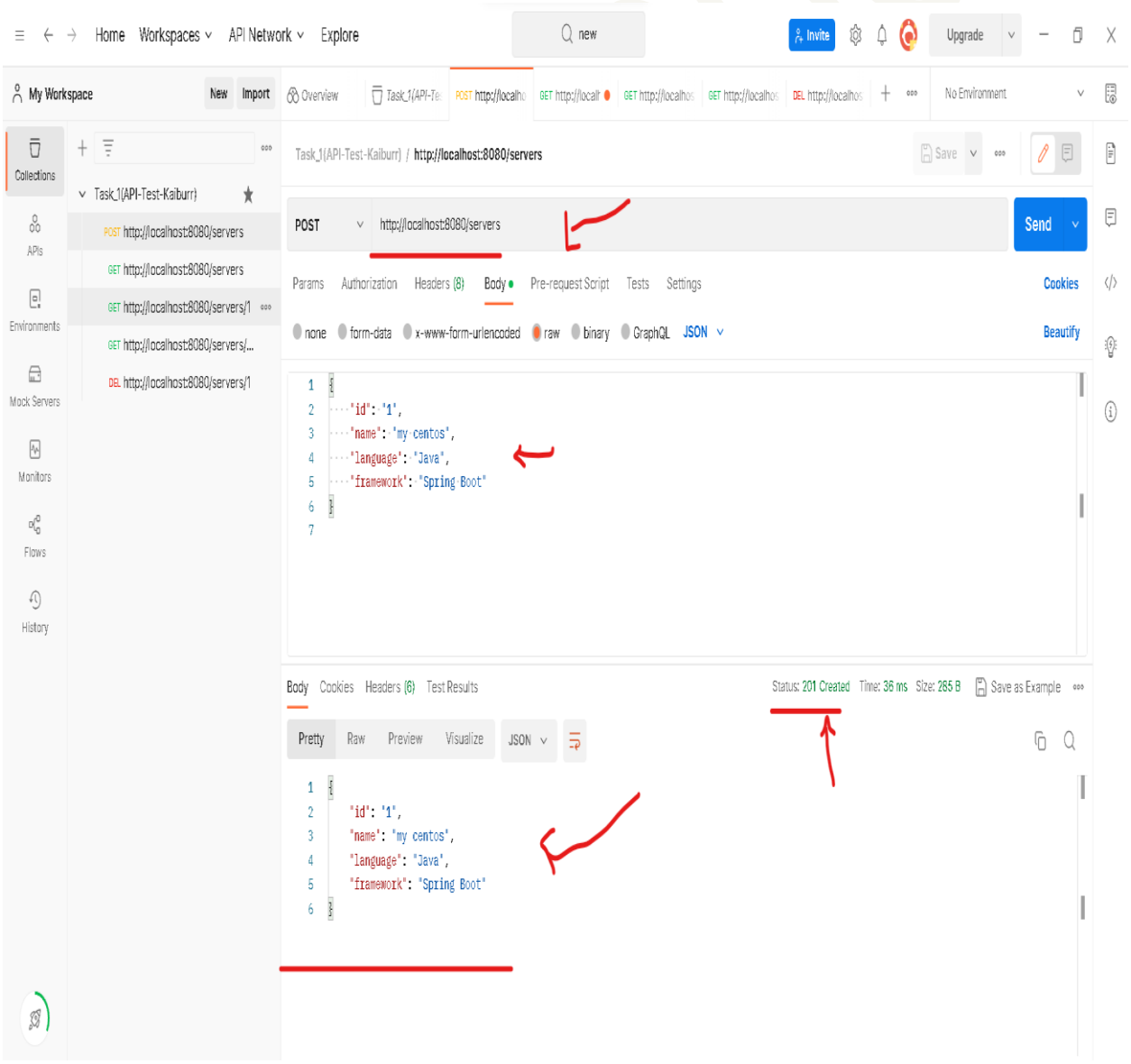
"id": "1",

"name": "my centos",

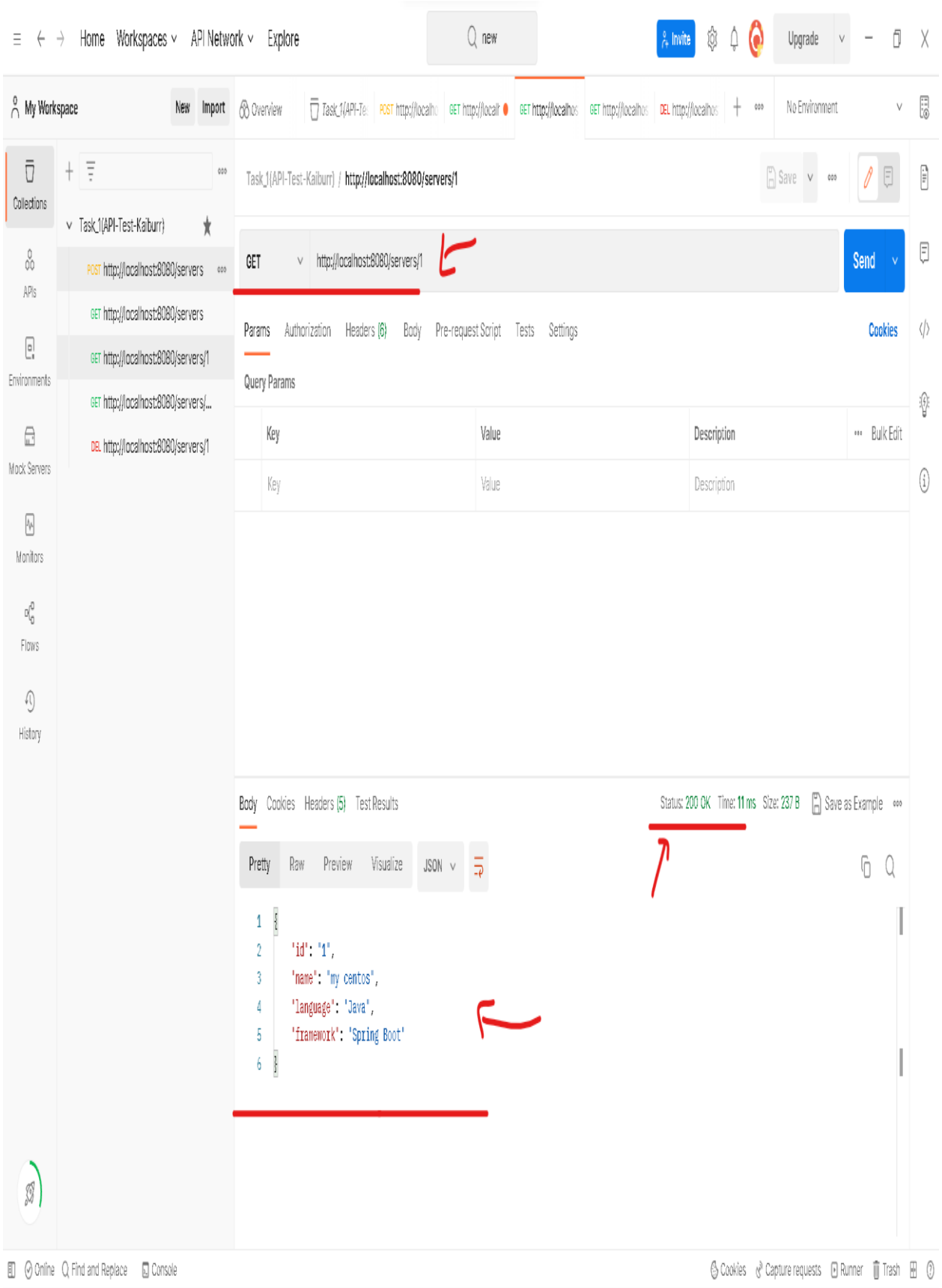
"language": "Java",

"framework": "Spring Boot"

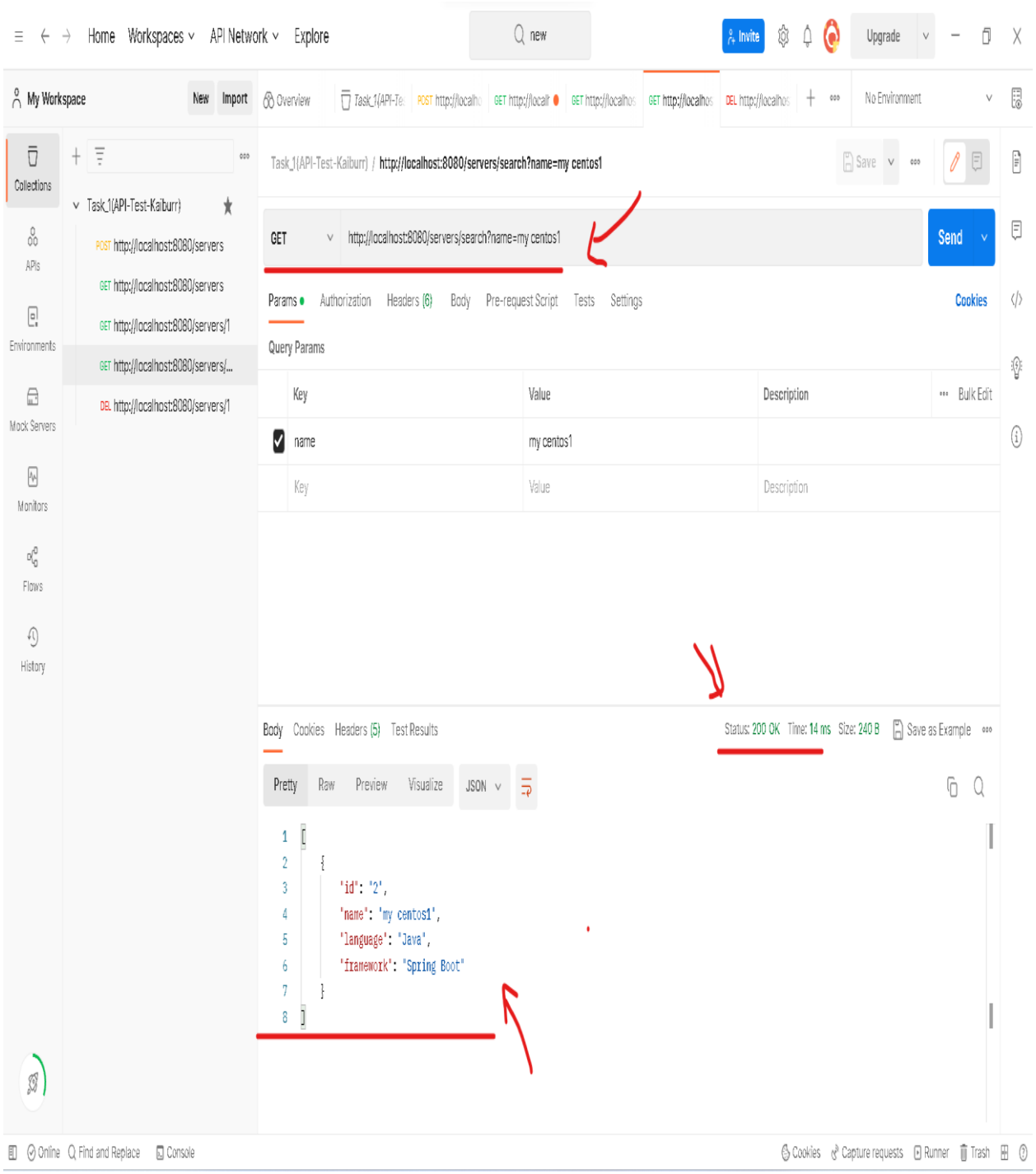
}



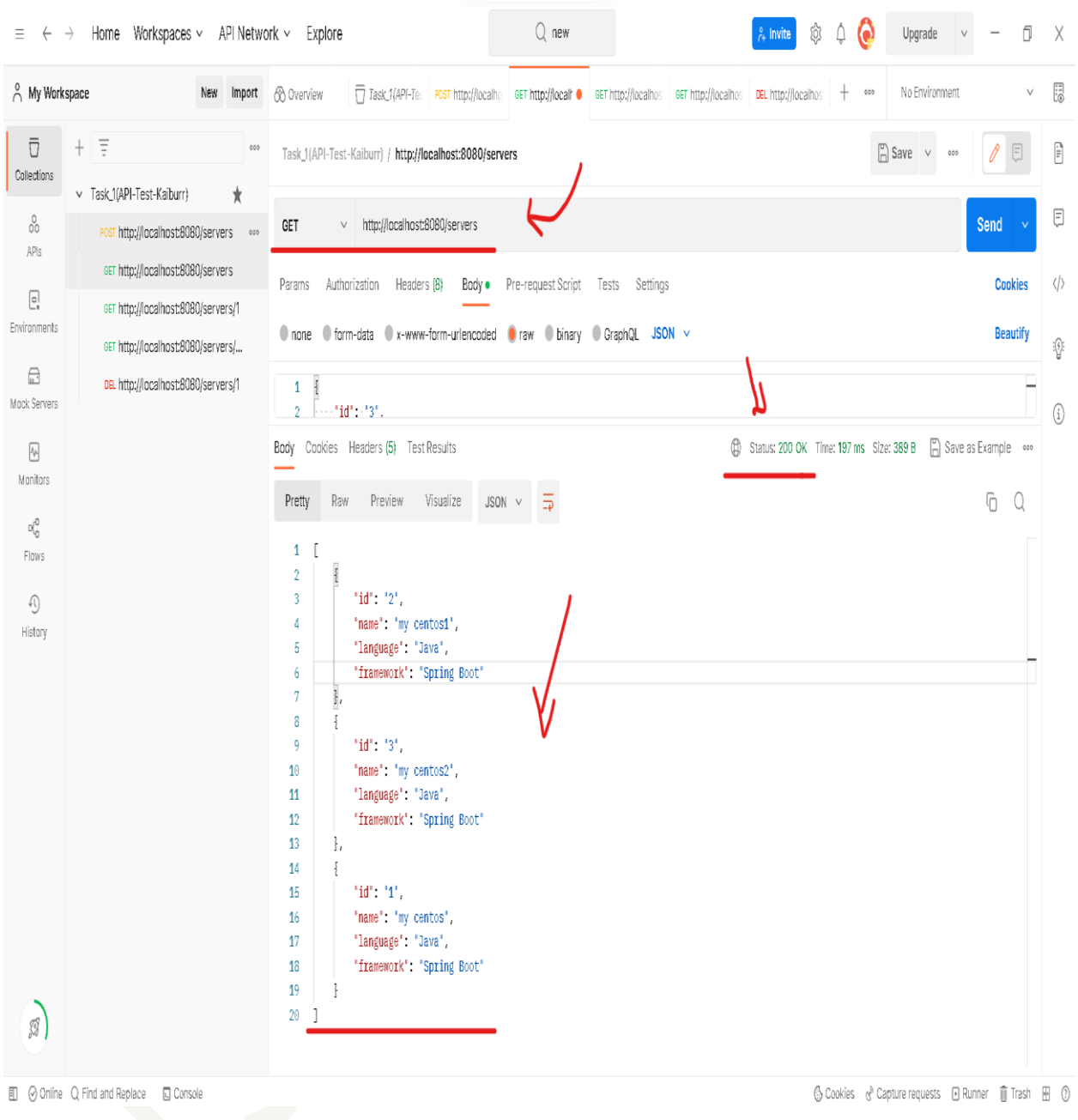
➢ Search by ID (GET) - http://localhost:8080/servers/1



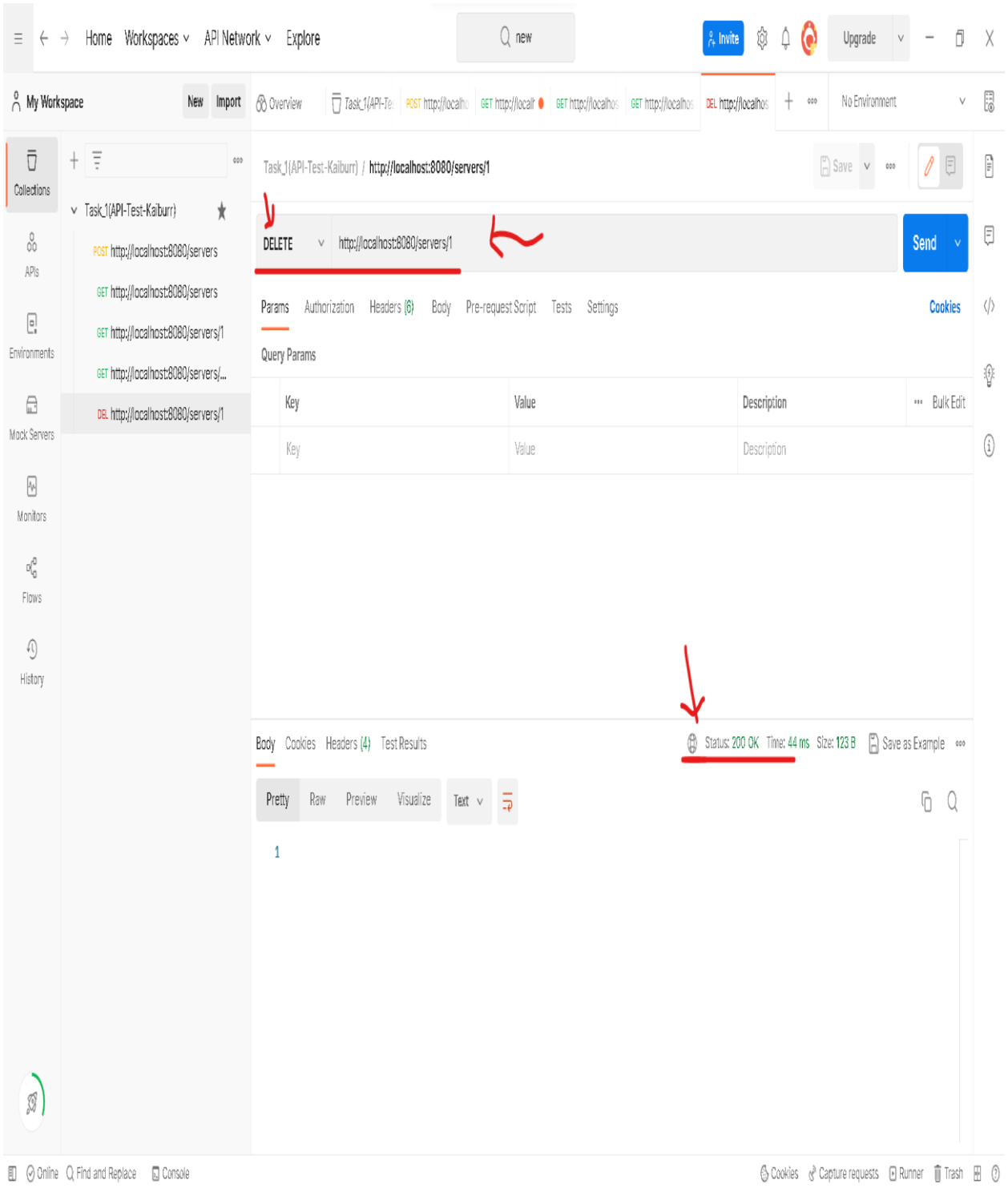
- Search by Name (GET) - http://localhost:8080/servers/search?name=my centos1



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

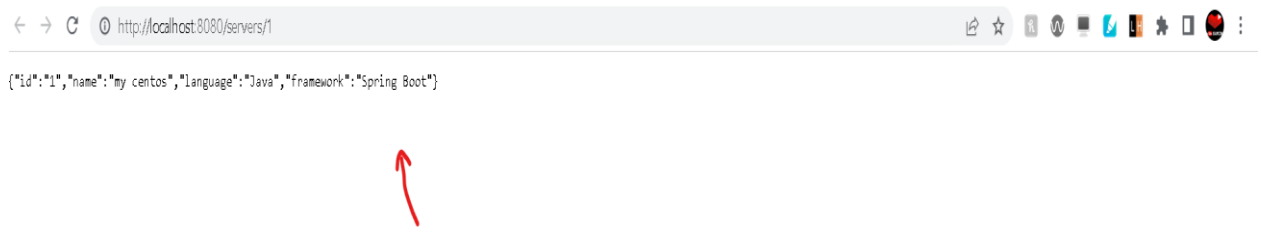


- Delete by ID (DELETE) - http://localhost:8080/servers/1

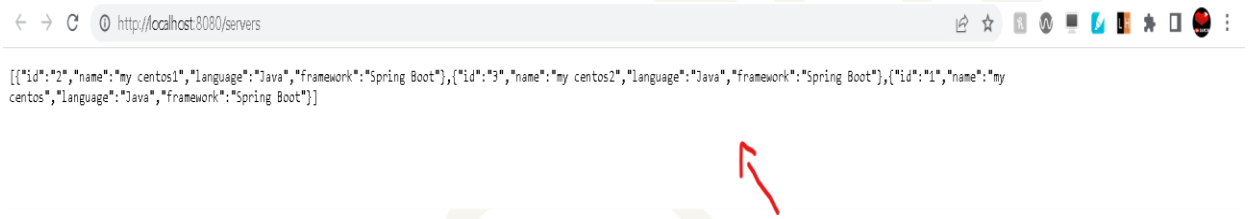


Browser (Google Chrome Output):

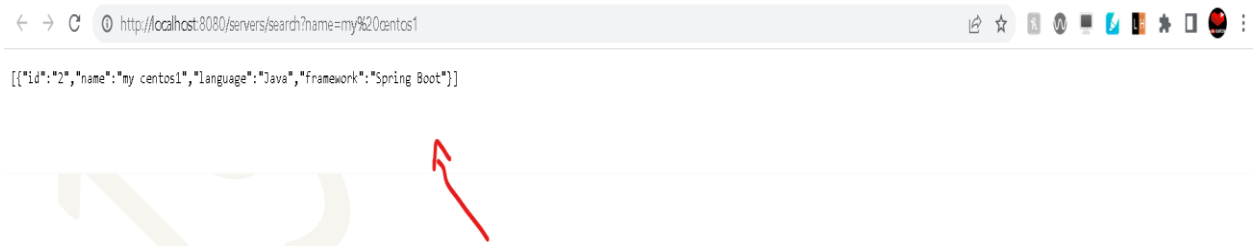
- Search by ID (GET) - http://localhost:8080/servers/1



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



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



Summary on how to get the Server Code through API Definition from Swagger:

Here are the steps you can follow:

1. Go to https://editor.swagger.io/ and create a new Swagger definition. You can either start from scratch or import the Swagger definition you created in task 1.
2. Define the endpoints for your REST API and their parameters, responses, and any other necessary details.
3. Once your Swagger definition is complete, click on the "Generate Server" button on the top right corner of the editor.
4. Select the Java language and the server framework you want to use. Swagger Codegen supports several server frameworks, such as Spring Boot, JAX-RS, and Vert.x.
5. Download the generated code and import it into your Java IDE.
6. Implement the same functionality as described in task 1 using the generated server code.
7. Deploy and run the server code.
8. Test the REST API using a tool such as Postman, curl, or any other HTTP client.

POSTMAN

Swagger
Supported by SMARTBEAR

Apache Maven™

Explanation on Tools/Technology(s) Used:

- **Swagger**:

  - Swagger is being used in ***Task 2*** to generate a REST API documentation, which provides a clear and structured description of the API endpoints, operations, input/output parameters, and possible responses.

  - By using Swagger, the API documentation can be generated automatically based on the annotated code, eliminating the need to write documentation manually. Swagger also provides an interactive UI, which allows developers to explore and test the API endpoints, making it easier to understand the API's functionality and how to use it.

  - In summary, Swagger simplifies the API development process by providing automated documentation, which helps developers and consumers of the API understand how to interact with the API effectively**.**

- **Maven**:

  - In ***task 2***, Maven is used as a build automation tool and dependency management tool for the Java project. It helps to manage project dependencies and automate the build process, making it easier to manage and maintain the project. With Maven, developers can easily add, remove or update the project's dependencies by simply modifying the project's configuration file (pom.xml), which simplifies the project's management process. Additionally, Maven can generate project documentation and reports, which can help the development team to better understand the project's structure, dependencies, and overall status.

❖ **IntelliJ (IDE)**:

- In ***Task 2***, IntelliJ IDEA is being used as an integrated development environment (IDE) to create and manage the project. IntelliJ IDEA is a popular IDE among Java developers and provides a range of features that help with coding, debugging, and testing applications. Some of its key features include intelligent code completion, code analysis, debugging tools, support for multiple programming languages, and integration with build tools such as Maven.

- IntelliJ IDEA also provides support for developing RESTful web services using Spring Framework, which is being used in this task. The IDE provides tools for configuring and deploying the web service, as well as testing and debugging it. Overall, the purpose of using IntelliJ IDEA in Task 2 is to simplify the process of developing, testing, and deploying the RESTful web service.

❖ **Github**:

- In ***task 2***, GitHub can be used as a version control system to store the source code of the project, manage changes, collaborate with others, and keep track of issues and bugs. By using GitHub, multiple developers can work on the same project simultaneously, track the changes made by each developer, and merge the changes together without conflict. It also provides a centralized platform for code review, collaboration, and documentation, which can improve the overall quality and efficiency of the project.

Thank You!