

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>.

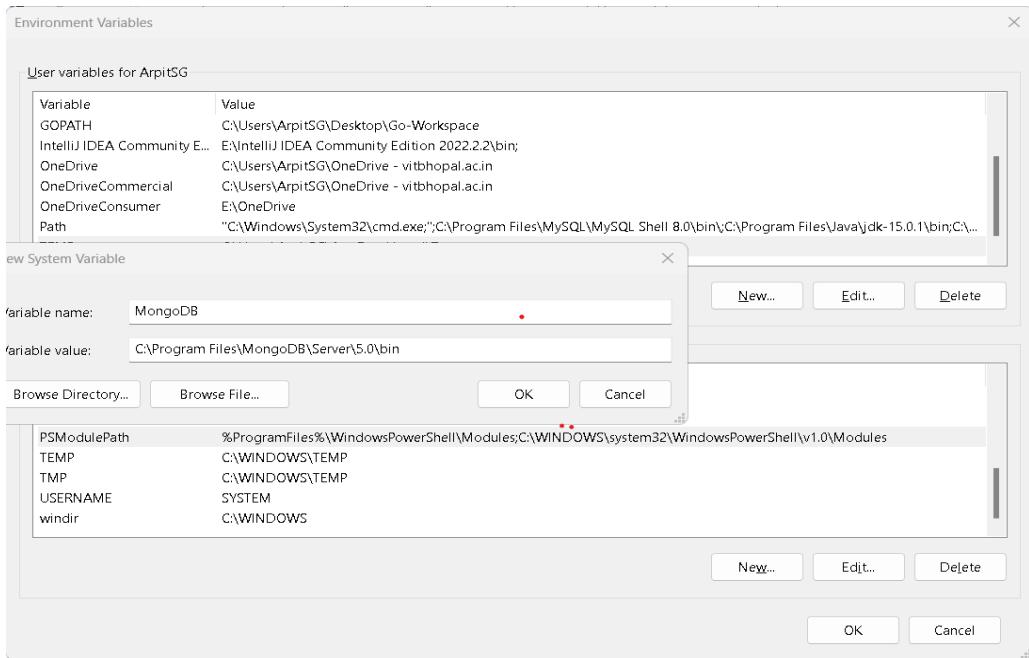
Make sure that you can deploy/run the generated code. Once your stub is ready - implement the same functionality as described in task #1, but now in java.

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:

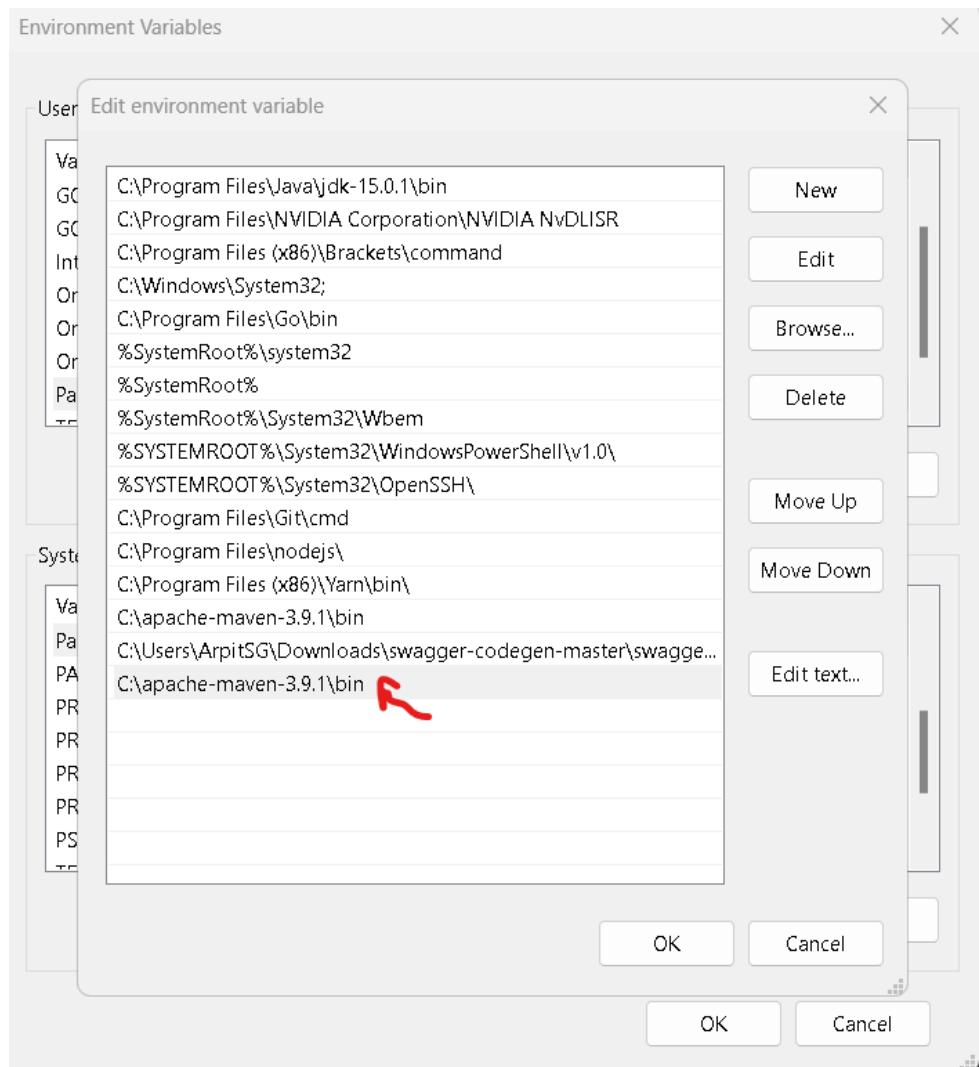
```
C:\Users\ArpitSG>mongo -version
MongoDB shell version v5.0.3
Build Info: {
    "version": "5.0.3",
    "gitVersion": "657fea5a61a74d7a79df7aff8e4bcf0bc742b748",
    "modules": [],
    "allocator": "tcmalloc",
    "environment": {
        "distmod": "windows",
        "distarch": "x86_64",
        "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. Installing the Updated Version of MAVEN in your System:

- Download the latest version of **Maven** from the official Apache Maven website (<https://maven.apache.org/download.cgi>).
- 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**".
- Set up the environment variable for Maven. **C:\apache-maven-3.9.1\bin**.



❖ Successful Installation can be checked as follows:

```
Administrator: Command Pro + 
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ArpitSG>mvn --version
Apache Maven 3.9.1 (2e178502fcdbfffc201671fb2537d0cb4b4cc58f8)
Maven home: C:\apache-maven-3.9.1
Java version: 15.0.1, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-15.0.1
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Users\ArpitSG>
```

3. Create a Swagger API Definition:

- Go to "<https://editor.swagger.io/>"

The screenshot shows the Swagger Editor interface. On the left, there is a code editor displaying YAML code for an API definition. The right side shows the API documentation with endpoints and parameters. A red arrow points to the browser's address bar, which displays the URL <https://editor.swagger.io/>.

```
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
          required: true
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Server'
        responses:
          '201':
            description: Created
          '400':
            description: Bad Request
    /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':
```

Kaiburr Task 2 API 1.0.0 OAS3

Servers http://localhost:8080

default

GET /servers Get all servers

POST /servers Create a new server

GET /servers/{id} Get server by ID

Parameters

Name Description

id * required ID of server to return

string (path) 1

- Click on "File" in the top left corner of the screen.
- 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).
- Define your '**API endpoints**' by clicking on "**Paths**" in the left-hand sidebar, and then clicking on the "**Add Path**" button.

The screenshot shows the Swagger Editor interface. On the left, there is a code editor displaying YAML code for an API definition. A red arrow points to a context menu that is open over the code, specifically pointing to the "Add Path Item" option. The right side shows the API documentation with endpoints and parameters.

```
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
          required: true
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Server'
        responses:
          '201':
            description: Created
          '400':
            description: Bad Request
    /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':
```

Kaiburr Task 2 API 1.0.0 OAS3

Servers http://localhost:8080

default

GET /servers Get all servers

POST /servers Create a new server

GET /servers/{id} Get server by ID

Parameters

Name Description

id * required ID of server to return

string (path) 1

- 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 DEFINITION**' and with the help of "**Swagger-UI**" we should be able to see our **API endpoints as we created in Task 1**).

The screenshot shows the Swagger Editor interface. On the left, the API definition is displayed in YAML format:

```

1 opaapi: 3.0.0
2 info:
3   title: Kaiburr Task 2 API
4   version: 1.0.0
5   servers:
6     - url: http://localhost:8080
7   paths:
8     /servers:
9       get:
10      summary: Get all servers
11      operationId: getServers
12      responses:
13        '200':
14          description: OK
15        '404':
16          description: Not Found
17      post:
18        summary: Create a new server
19        operationId: createServer
20        requestBody:
21          description: Server object to be created
22          required: true
23          content:
24            application/json:
25              schema:
26                $ref: '#/components/schemas/Server'
27      responses:
28        '201':
29          description: Created
30        '400':
31          description: Bad Request
32    /servers/{id}:
33      get:
34        summary: Get server by ID
35        operationId: getServerById
36        parameters:
37          - name: id
38            in: path
39            description: ID of server to return
40            required: true
41          schema:
42            type: string
43        responses:
44          '200':
45            description: OK
46          '404':
47            description: Not Found
48      # put:
49      # delete:

```

On the right, the generated Swagger-UI interface is shown for the 'Kaiburr Task 2 API' version 1.0.0. It includes a 'Servers' dropdown set to 'http://localhost:8080'. Below it, under the 'default' section, are several API endpoints:

- GET /servers** (Get all servers)
- POST /servers** (Create a new server) - highlighted with a red checkmark
- GET /servers/{id}** (Get server by ID)
- DELETE /servers/{id}** (Delete server by ID) - highlighted with a red checkmark
- GET /servers/search/{name}** (Search servers by name)

- 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.



API DEFINITION FILE:

```
1 ▼ {
2     "openapi": "3.0.0",
3     "info": {
4         "title": "Kaiburr Task 2 API",
5         "version": "1.0.0"
6     },
7     "servers": [
8         {
9             "url": "http://localhost:8080"
10        }
11    ],
12    "paths": {
13        "/servers": { ↗
14            "get": {
15                "summary": "Get all servers",
16                "operationId": "getServers",
17                "responses": {
18                    "200": {
19                        "description": "OK"
20                    },
21                    "404": {
22                        "description": "Not Found"
23                    }
24                }
25            },
26            "post": { ↗
27                "summary": "Create a new server",
28                "operationId": "createServer",
29                "requestBody": {
30                    "description": "Server object to be created",
31                    "required": true,
32                    "content": {
33                        "application/json": {
34                            "schema": {
35                                "$ref": "#/components/schemas/Server"
36                            }
37                        }
38                    }
39                },
40                "responses": {
41                    "201": {
42                        "description": "Created"
43                    },
44                    "400": {
45                        "description": "Bad Request"
46                    }
47                }
48            }
49        }
50    },
51    "/servers/{id)": { ↗
52        "get": {
53            "summary": "Get server by ID",
54            "operationId": "getServerById",
55            "parameters": [
56                {
57                    "name": "id",
58                    "in": "path",
59                    "description": "ID of server to return",
60                    "required": true,
61                    "schema": {
62                        "type": "string"
63                    }
64                ]
65            },
66            "responses": {
67                "200": {
68                    "description": "OK"
69                },
69                "404": {
70                    "description": "Not Found"
71                }
72            }
73        },
74        "delete": { ↗
75            "summary": "Delete server by ID",
76            "operationId": "deleteServerById",
77            "parameters": [
78                {
79                    "name": "id",
80                    "in": "path",
81                    "description": "ID of server to delete",
82                    "required": true,
83                    "schema": {
84                        "type": "string"
85                    }
86                ]
87            },
88            "responses": {
89                "204": {
90                    "description": "No Content"
91                },
92                "404": {
93                    "description": "Not Found"
94                }
95            }
96        }
97    }
98}
```

```
46        }
47    }
48    }
49 },
50 "/servers/{id)": { ↗
51     "get": {
52         "summary": "Get server by ID",
53         "operationId": "getServerById",
54         "parameters": [
55             {
56                 "name": "id",
57                 "in": "path",
58                 "description": "ID of server to return",
59                 "required": true,
60                 "schema": {
61                     "type": "string"
62                 }
63             }
64         ],
65         "responses": {
66             "200": {
67                 "description": "OK"
68             },
69             "404": {
70                 "description": "Not Found"
71             }
72         }
73     },
74     "delete": { ↗
75         "summary": "Delete server by ID",
76         "operationId": "deleteServerById",
77         "parameters": [
78             {
79                 "name": "id",
80                 "in": "path",
81                 "description": "ID of server to delete",
82                 "required": true,
83                 "schema": {
84                     "type": "string"
85                 }
86             ]
87         },
88         "responses": {
89             "204": {
90                 "description": "No Content"
91             },
92             "404": {
93                 "description": "Not Found"
94             }
95         }
96     }
97 }
```

```
 91      },
 92  "404": {
 93     "description": "Not Found"
 94   }
 95 }
 96 }
 97 },
 98 "/servers/search/{name)": {
 99   "get": {
100     "summary": "Search servers by name",
101     "operationId": "searchServersByName",
102     "parameters": [
103       {
104         "name": "name",
105         "in": "path",
106         "description": "Name of server to search",
107         "required": true,
108         "schema": {
109           "type": "string"
110         }
111       }
112     ],
113     "responses": {
114       "200": {
115         "description": "OK"
116       },
117       "404": {
118         "description": "Not Found"
119       }
120     }
121   }
122 },
123 },
124 "components": { ↴
125   "schemas": {
126     "Server": {
127       "type": "object",
128       "required": [
129         "name",
130         "id",
131         "language",
132         "framework"
133       ],
134       "properties": {
135         "name": {
136           "type": "string"
137         },
138         "id": {
139           "type": "string"
140         },
141         "language": {
142           "type": "string"
143         },
144         "framework": {
145           "type": "string"
146         }
147       }
148     }
149   }
150 }
```

```
Line 127, Column 28—151 Lines
```

```
106     "description": "Name of server to search",
107     "required": true,
108     "schema": {
109       "type": "string"
110     }
111   },
112   ],
113   "responses": {
114     "200": {
115       "description": "OK"
116     },
117     "404": {
118       "description": "Not Found"
119     }
120   }
121 }
122 },
123 },
124 "components": { ↴
125   "schemas": {
126     "Server": {
127       "type": "object",
128       "required": [
129         "name",
130         "id",
131         "language",
132         "framework"
133       ],
134       "properties": {
135         "name": {
136           "type": "string"
137         },
138         "id": {
139           "type": "string"
140         },
141         "language": {
142           "type": "string"
143         },
144         "framework": {
145           "type": "string"
146         }
147       }
148     }
149   }
150 }
```

Explanation (Code):

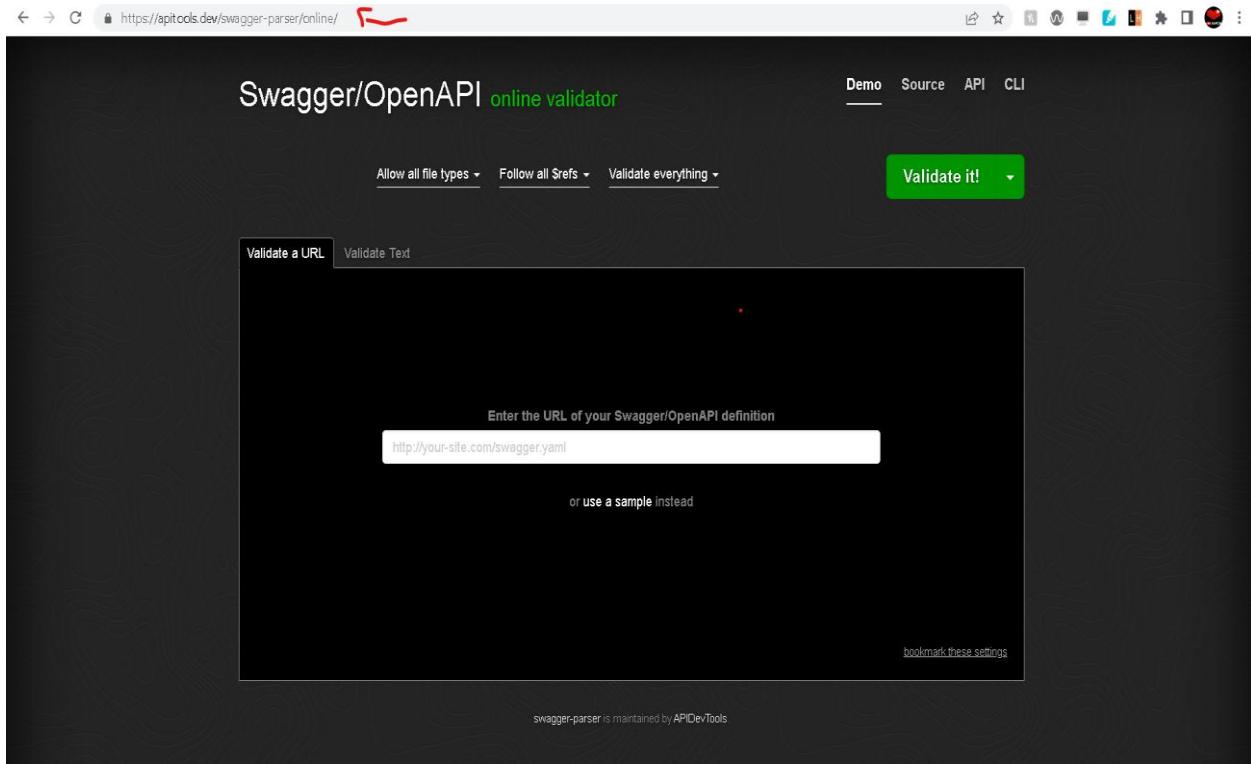
- So, here the code is quite ‘***self-explanatory***’ we have added ***API endpoints and data models*** as per Task 1(**JSON** Format).
- To perform the ***Create, Delete, View and Search*** operations.
- The ‘***path***’ helps us to navigate to the ‘***directory***’ where the corresponding/required 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:

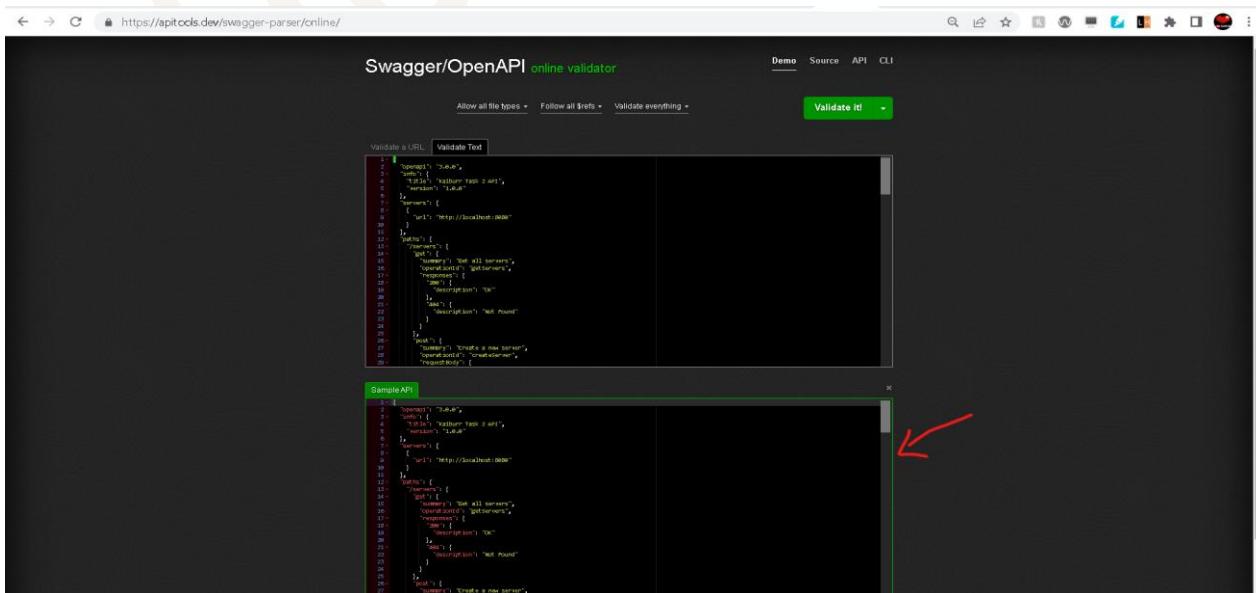
- ***GET /servers***: to get all servers
- ***POST /servers***: to create a new server
- ***GET /servers/{fid}***: to get a server by ID
- ***DELETE /servers/{fid}***: to delete a server by ID
- ***GET /servers/search/{name}***: to search servers by name
- The “***data model defined***” in the API is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

4. Validate and save your API definition file:

- Go to <https://apitools.dev/swagger-parser/online/>.



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

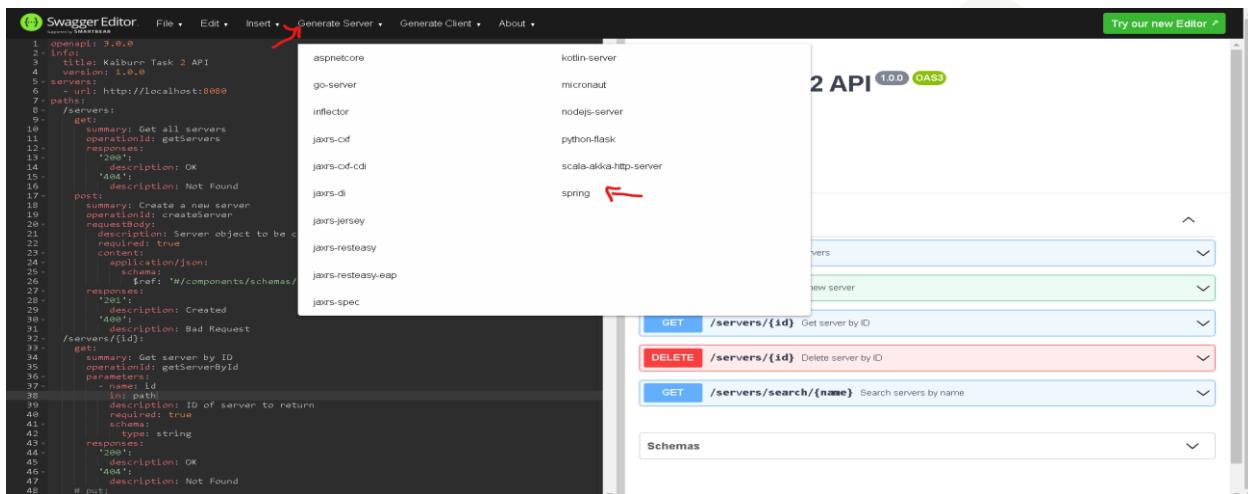


5. Generate server code using Swagger:

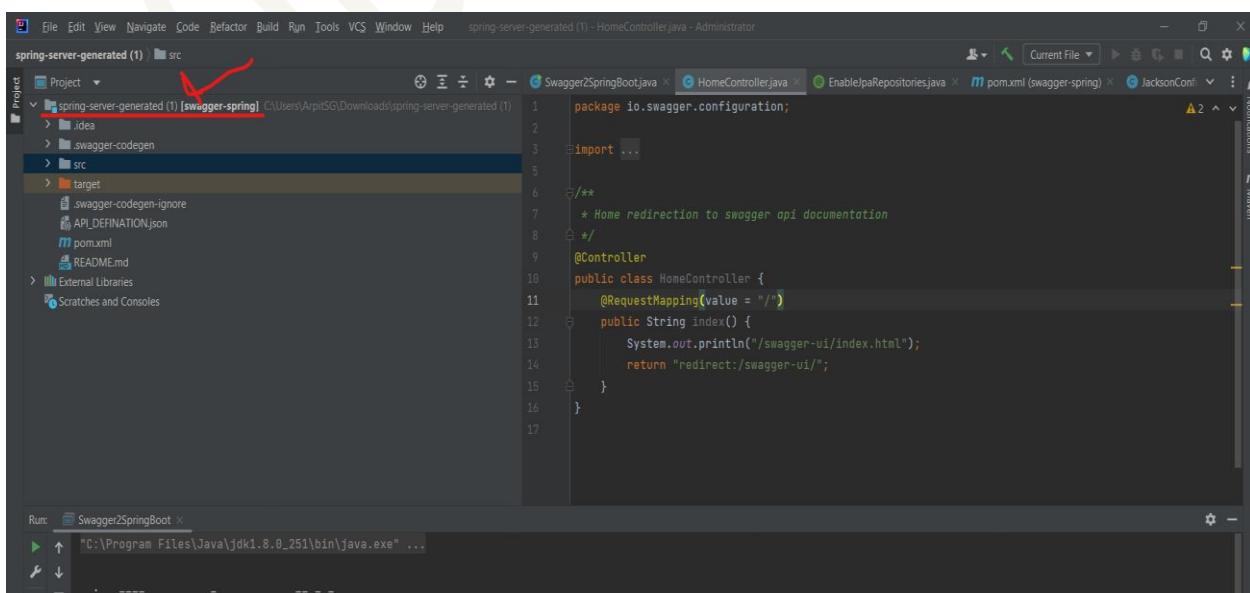
- a. There are “**2 ways of generating Server code**” using Swagger:
 - i. **Swagger Editor** (**I used this one**).
 - ii. **Swagger Codegen** (**Steps provided in doc**):
 1. Download Swagger Codegen from
<https://swagger.io/tools/swagger-codegen/>
 2. **Extract** the downloaded file to a directory 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 DEFINITION**' 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)

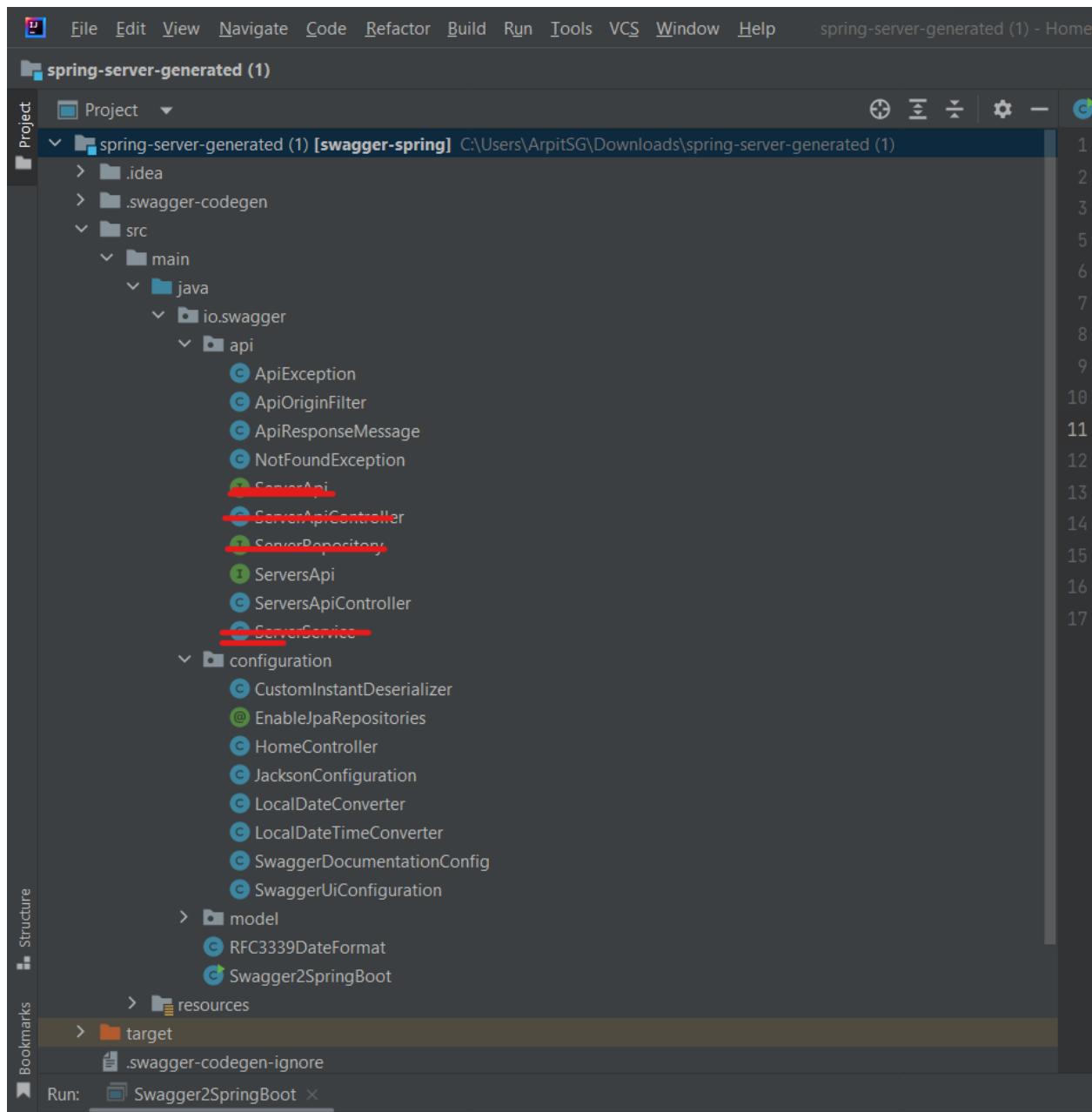


- l. This will give you a **Zip file** containing an auto-generated spring project based off the **API Definition** you provided. *Ex:* **"C:\Users\ArpitSG\Downloads\spring-server-generated.zip"**
- m. You can **Extract and open up this project with IntelliJ** or preferred IDE.



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.

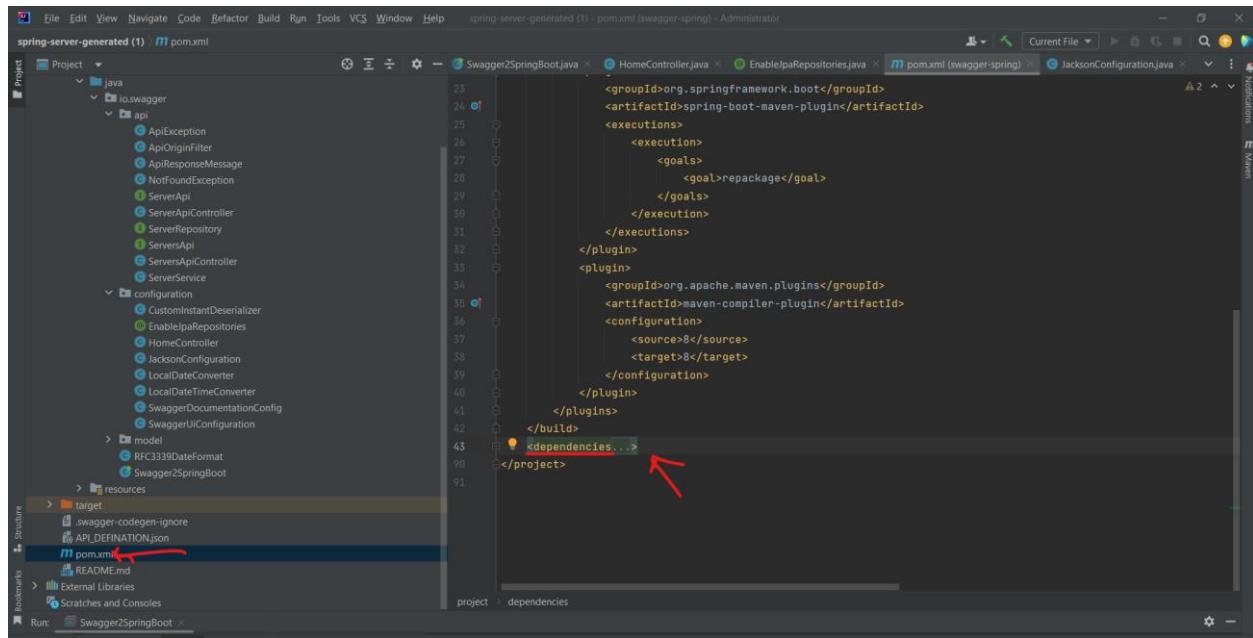
```

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>io.swagger</groupId>
    <artifactId>swagger-spring</artifactId>
    <packaging>jar</packaging>
    <name>swagger-spring</name>
    <version>1.0.0</version>
    <properties>
        <java.version>1.7</java.version>
        <maven.compiler.source>${java.version}</maven.compiler.source>
        <maven.compiler.target>${java.version}</maven.compiler.target>
        <springfox-version>3.0.8</springfox-version>
    </properties>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.16.RELEASE</version>
    </parent>
    <build>
        <sourceDirectory>src/main/java</sourceDirectory>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <executions>
                    <execution>
                        <goals>
                            <goal>repackage</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

```

(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**)

```

package io.swagger;
import ...
usage
@SpringBootApplication
@EnableOpenApi
@ComponentScan(basePackages = { "io.swagger", "io.swagger.api", "io.swagger.configuration"})
public class Swagger2SpringBoot implements CommandLineRunner {
}

```

Run: SwaggerSpringBoot
"C:\Program Files\Java\jdk1.8.0_251\bin\java.exe" ...

```

. . .
:: Spring Boot ::      (v2.1.16.RELEASE)
2023-03-28 01:57:26.588 INFO 5320 --- [           main] io.swagger.Swagger2SpringBoot        : Starting Swagger2SpringBoot on DESKTOP-33BM0S8 with PID 5320 (E:\Kaibur_Tasks)
2023-03-28 01:57:26.588 INFO 5320 --- [           main] io.swagger.Swagger2SpringBoot        : No active profile set, falling back to default profiles: default
2023-03-28 01:57:29.492 INFO 5320 --- [           main] o.s.w.e.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-03-28 01:57:29.552 INFO 5320 --- [           main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-03-28 01:57:29.552 INFO 5320 --- [           main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.37]
2023-03-28 01:57:29.783 INFO 5320 --- [           main] o.a.c.c.C.Tomcat.[localhost].[]       : Initializing Spring embedded WebApplicationContext: initialization completed in 2917 ms
2023-03-28 01:57:29.784 INFO 5320 --- [           main] o.s.web.context.ContextLoader        : Root WebApplicationContext: initialization completed in 2917 ms
2023-03-28 01:57:30.932 INFO 5320 --- [           main] o.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2023-03-28 01:57:31.635 INFO 5320 --- [           main] o.s.w.e.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8088 (http) with context path ''
2023-03-28 01:57:31.640 INFO 5320 --- [           main] io.swagger.Swagger2SpringBoot        : Started Swagger2SpringBoot in 5.863 seconds (JVM running for 6.874)

```

- 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:

Kaiburr Task 2 API 1.0.0 OAS3

Servers
http://localhost:8080

default

GET /servers Get all servers

Parameters

No parameters

Responses

Code	Description	Links
200	OK	No links
404	Not Found	No links

b. To get a single server by ID:

GET /servers/{id} Get server by ID

Parameters

Name	Description
id * required	ID of server to return
string	
(path)	1

Cancel

Execute

Responses

Code	Description	Links
200	OK	No links
404	Not Found	No links

c. To create a new server:

POST /servers Create a new server

Parameters

No parameters

Request body * required

application/json

Server object to be created

```
{ "name": "arpit", "id": "2223", "language": "java", "framework": "spring" }
```

Cancel Reset

Execute

Responses

Code	Description	Links
201	Created	No links
400	Bad Request	No links

d. To delete a server by ID:

The screenshot shows a DELETE API endpoint for deleting a server by ID. The URL is /servers/{id}. The parameters section shows a required parameter 'id' with a value of '221'. The responses section shows two possible outcomes: a 204 status code with 'No Content' and a 404 status code with 'Not Found'. A red arrow points to the 'id' parameter, and another red arrow points to the 'Not Found' response.

DELETE /servers/{id} Delete server by ID

Parameters

Name Description

id * required ID of server to delete
string
(path) 221

Cancel

Execute

Responses

Code Description Links

204 No Content No links

404 Not Found No links

e. To search for servers by name:

The screenshot shows a GET API endpoint for searching servers by name. The URL is /servers/search/{name}. The parameters section shows a required parameter 'name' with a value of 'arpit'. The responses section shows two possible outcomes: a 200 status code with 'OK' and a 404 status code with 'Not Found'. A red arrow points to the 'name' parameter, and another red arrow points to the 'OK' response.

GET /servers/search/{name} Search servers by name

Parameters

Name Description

name * required Name of server to search
string
(path) arpit

Cancel

Execute

Responses

Code Description Links

200 OK No links

404 Not Found No links

- 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.

GET /servers Get all servers ^

Parameters Cancel

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8080/servers' \
-H 'accept: */*'
```

Request URL

`http://localhost:8080/servers`

Server response

Code	Details	Links
<small>Undocumented</small>	Failed to fetch. Possible Reasons: <ul style="list-style-type: none"> • CORS • Network Failure • URL scheme must be "http" or "https" for CORS request. 	

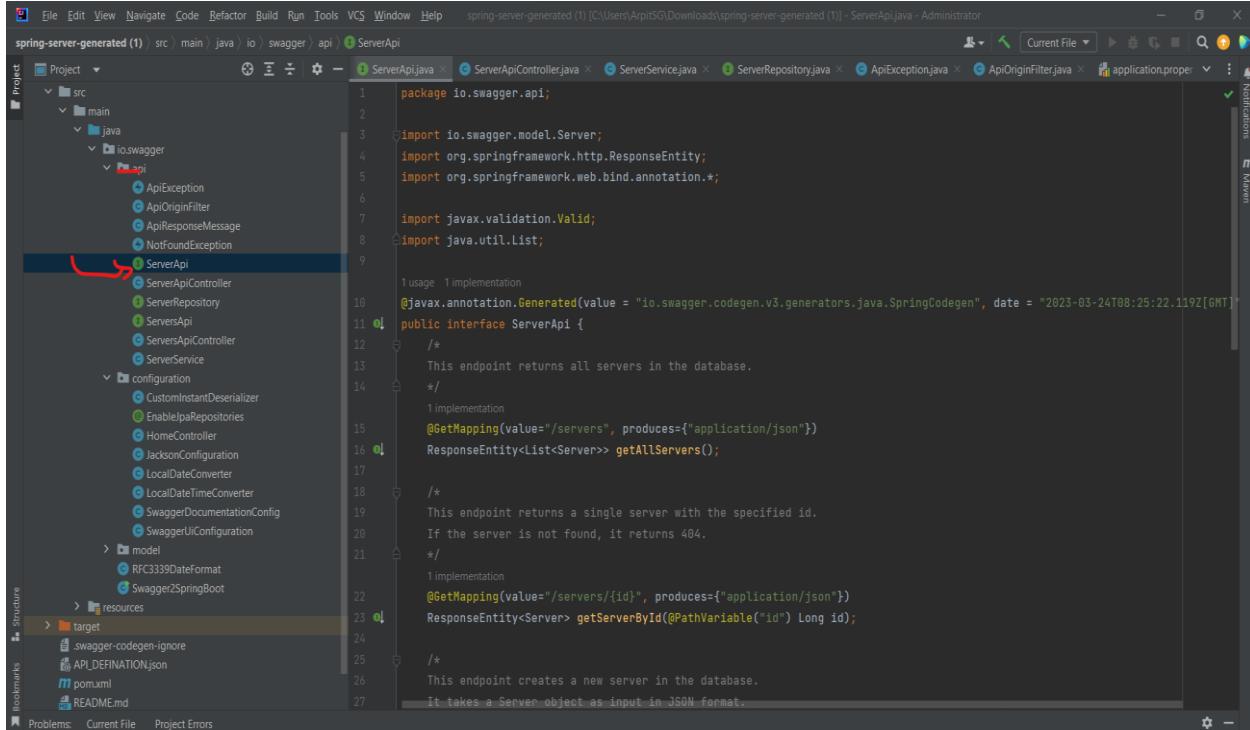
Responses

Code	Description	Links
200	OK	No links
404	Not Found	No links



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”.



```
package io.swagger.api;

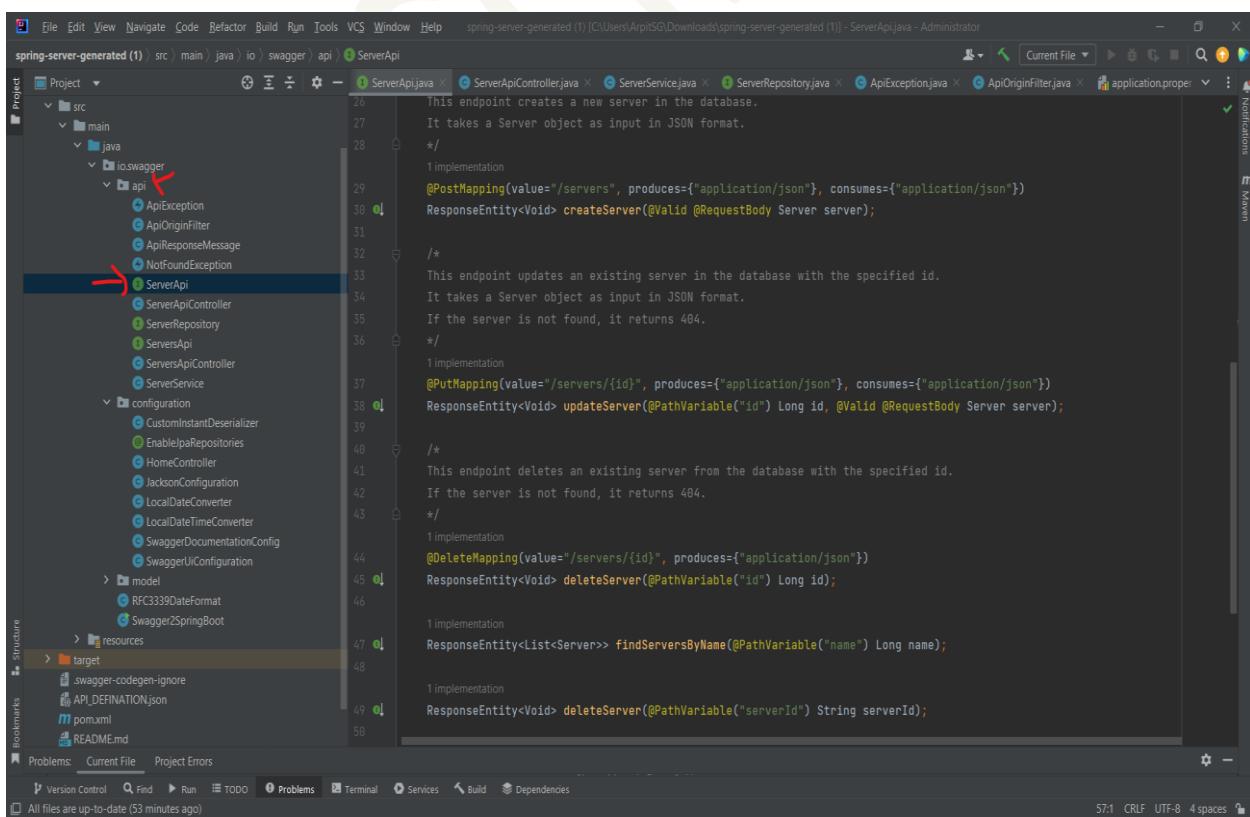
import io.swagger.model.Server;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.List;

1 usage 1 implementation
@javax.annotation.Generated(value = "io.swagger.codegen.v3.generators.java.SpringCodegen", date = "2023-03-24T08:25:22.119Z[GMT]")
public interface ServerApi {
    /**
     * This endpoint returns all servers in the database.
     */
    @GetMapping(value = "/servers", produces = {"application/json"})
    ResponseEntity<List<Server>> getAllServers();

    /**
     * This endpoint returns a single server with the specified id.
     * If the server is not found, it returns 404.
     */
    @GetMapping(value = "/servers/{id}", produces = {"application/json"})
    ResponseEntity<Server> getServerById(@PathVariable("id") Long id);

    /**
     * This endpoint creates a new server in the database.
     * It takes a Server object as input in JSON format.
     */
}
```



```
This endpoint creates a new server in the database.
It takes a Server object as input in JSON format.
*/
1 implementation
@PostMapping(value = "/servers", produces = {"application/json"}, consumes = {"application/json"})
ResponseEntity<Void> createServer(@Valid @RequestBody Server server);

/*
This endpoint updates an existing server in the database with the specified id.
It takes a Server object as input in JSON format.
If the server is not found, it returns 404.
*/
1 implementation
@PutMapping(value = "/servers/{id}", produces = {"application/json"}, consumes = {"application/json"})
ResponseEntity<Void> updateServer(@PathVariable("id") Long id, @Valid @RequestBody Server server);

/*
This endpoint deletes an existing server from the database with the specified id.
If the server is not found, it returns 404.
*/
1 implementation
@DeleteMapping(value = "/servers/{id}", produces = {"application/json"})
ResponseEntity<Void> deleteServer(@PathVariable("id") Long id);

1 implementation
ResponseEntity<List<Server>> findServersByName(@PathVariable("name") Long name);

1 implementation
ResponseEntity<Void> deleteServer(@PathVariable("serverId") String serverId);
```

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*”.

```
// This class is the implementation of the ServerApi interface and serves as the entry point for the API requests.

package io.swagger.api;

import io.swagger.model.Server;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController
public class ServerApiController implements ServerApi {

    private final ServerService serverService;

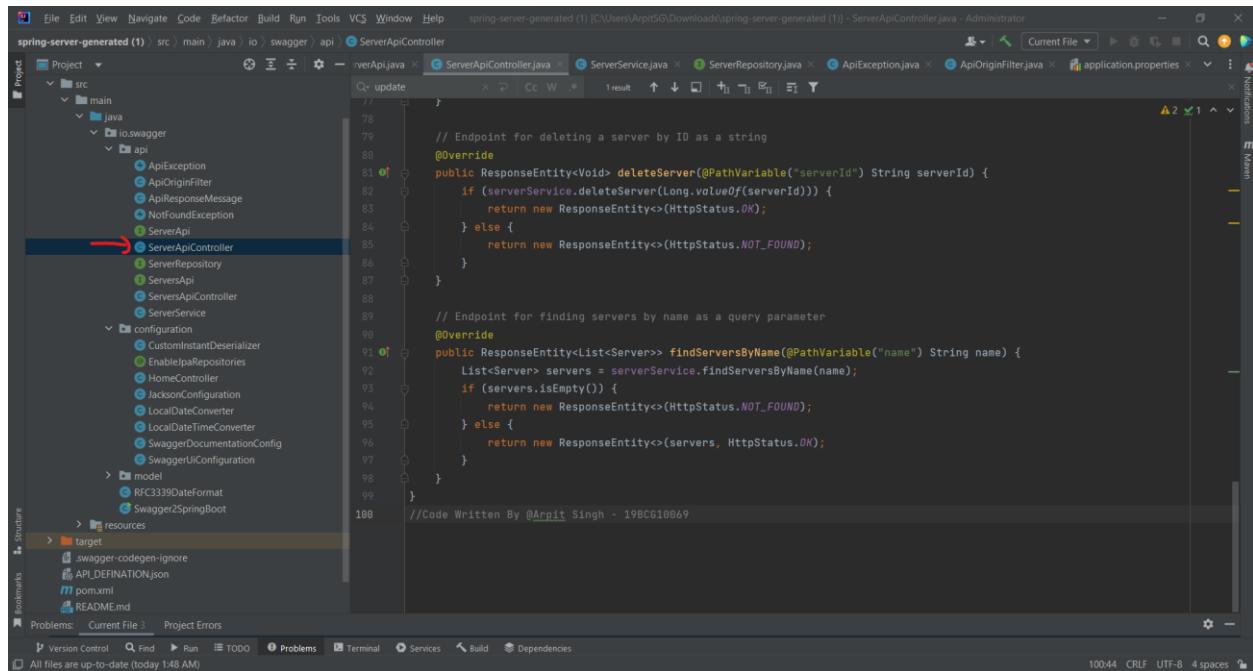
    // Constructor injection of ServerService
    @Autowired
    public ServerApiController(ServerService serverService) {
        this.serverService = serverService;
    }

    // Endpoint for getting all servers
    @Override
    public List<ResponseEntity<Server>> getAllServers() {

```

```
27     @Override
28     public ResponseEntity<List<Server>> getAllServers() {
29         List<Server> servers = serverService.getAllServers();
30         return new ResponseEntity<>(servers, HttpStatus.OK);
31     }
32
33     // Endpoint for getting a server by ID
34     @Override
35     public ResponseEntity<Server> getServerById(@PathVariable("serverId") Long serverId) {
36         Server server = serverService.getServerById(serverId);
37         if (server == null) {
38             return new ResponseEntity<>(HttpStatus.NOT_FOUND);
39         } else {
40             return new ResponseEntity<>(server, HttpStatus.OK);
41         }
42     }
43
44     // Endpoint for creating a server
45     @Override
46     public ResponseEntity<Void> createServer(@RequestBody Server server) {
47         serverService.createServer(server);
48         return new ResponseEntity<>(HttpStatus.CREATED);
49     }
50
51     // Endpoint for updating a server
52     // Not implemented in this version
53     @Override
54     public ResponseEntity<Void> updateServer(Long id, Server server) {
55         return null;
56     }
57
58     // Endpoint for deleting a server by ID
59     @Override
60     public ResponseEntity<Void> deleteServer(@PathVariable("serverId") Long serverId) {
61         if (serverService.deleteServer(serverId)) {
62             return new ResponseEntity<>(HttpStatus.OK);
63         } else {
64             return new ResponseEntity<>(HttpStatus.NOT_FOUND);
65         }
66     }
67
68     // Endpoint for finding servers by name
69     @Override
70     public ResponseEntity<List<Server>> findServersByName(@PathVariable("name") String name) {
71         List<Server> servers = serverService.findServersByName(String.valueOf(name));
72         if (servers.isEmpty()) {
73             return new ResponseEntity<>(HttpStatus.NOT_FOUND);
74         } else {
75             return new ResponseEntity<>(servers, HttpStatus.OK);
76         }
77     }
78
79     // Endpoint for deleting a server by ID as a string
80     @Override
81     public ResponseEntity<Void> deleteServer(@PathVariable("serverId") String serverId) {
82         if (serverService.deleteServer(Long.parseLong(serverId))) {
83             return new ResponseEntity<>(HttpStatus.OK);
84         } else {
85             return new ResponseEntity<>(HttpStatus.NOT_FOUND);
86         }
87     }
88 }
```

```
54     public ResponseEntity<Void> updateServer(Long id, Server server) {
55         return null;
56     }
57
58     // Endpoint for deleting a server by ID
59     @Override
60     public ResponseEntity<Void> deleteServer(@PathVariable("serverId") Long serverId) {
61         if (serverService.deleteServer(serverId)) {
62             return new ResponseEntity<>(HttpStatus.OK);
63         } else {
64             return new ResponseEntity<>(HttpStatus.NOT_FOUND);
65         }
66     }
67
68     // Endpoint for finding servers by name
69     @Override
70     public ResponseEntity<List<Server>> findServersByName(@PathVariable("name") String name) {
71         List<Server> servers = serverService.findServersByName(String.valueOf(name));
72         if (servers.isEmpty()) {
73             return new ResponseEntity<>(HttpStatus.NOT_FOUND);
74         } else {
75             return new ResponseEntity<>(servers, HttpStatus.OK);
76         }
77     }
78
79     // Endpoint for deleting a server by ID as a string
80     @Override
81     public ResponseEntity<Void> deleteServer(@PathVariable("serverId") String serverId) {
82         if (serverService.deleteServer(Long.parseLong(serverId))) {
83             return new ResponseEntity<>(HttpStatus.OK);
84         } else {
85             return new ResponseEntity<>(HttpStatus.NOT_FOUND);
86         }
87     }
88 }
```



```

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help spring-server-generated (1) [C:\Users\ArpitS\Downloads\spring-server-generated (1)] - ServerApiController.java - Administrator
Project src main java io.swagger api ServerApiController.java ServerService.java ServerRepository.java ApiException.java ApiOriginFilter.java application.properties
src/main/java/io/swagger/api/ServerApiController.java
    package io.swagger.api;
    ...
    /**
     * 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);
        }
    }
}

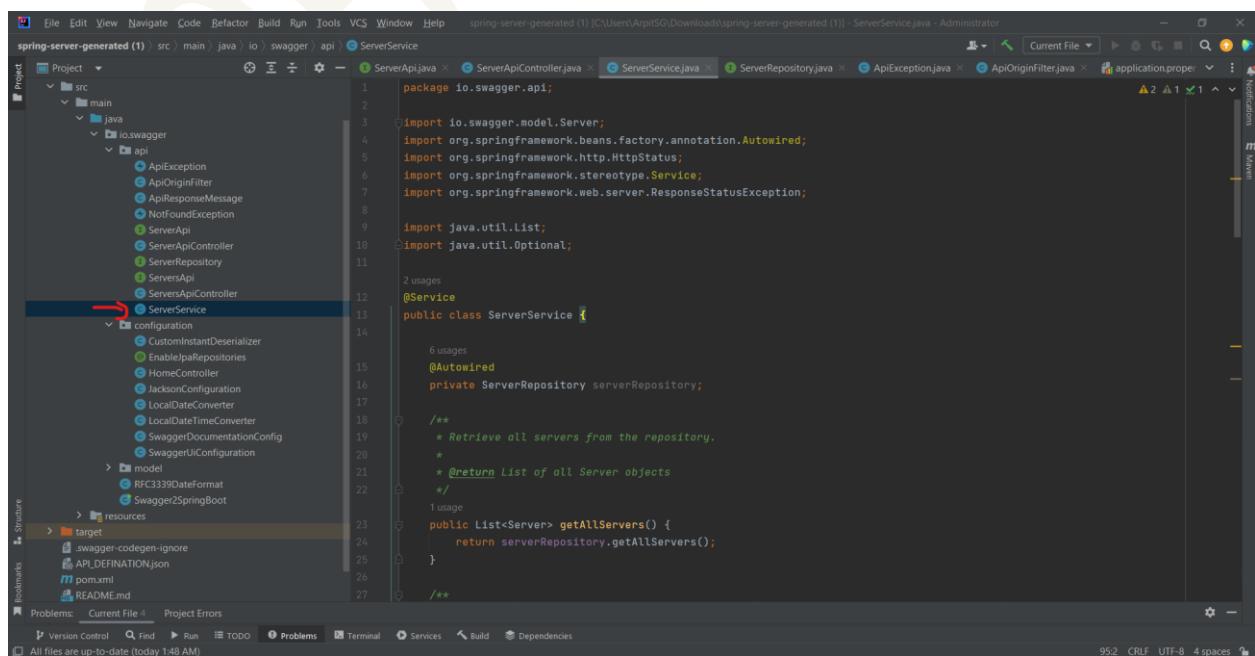
```

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**”:

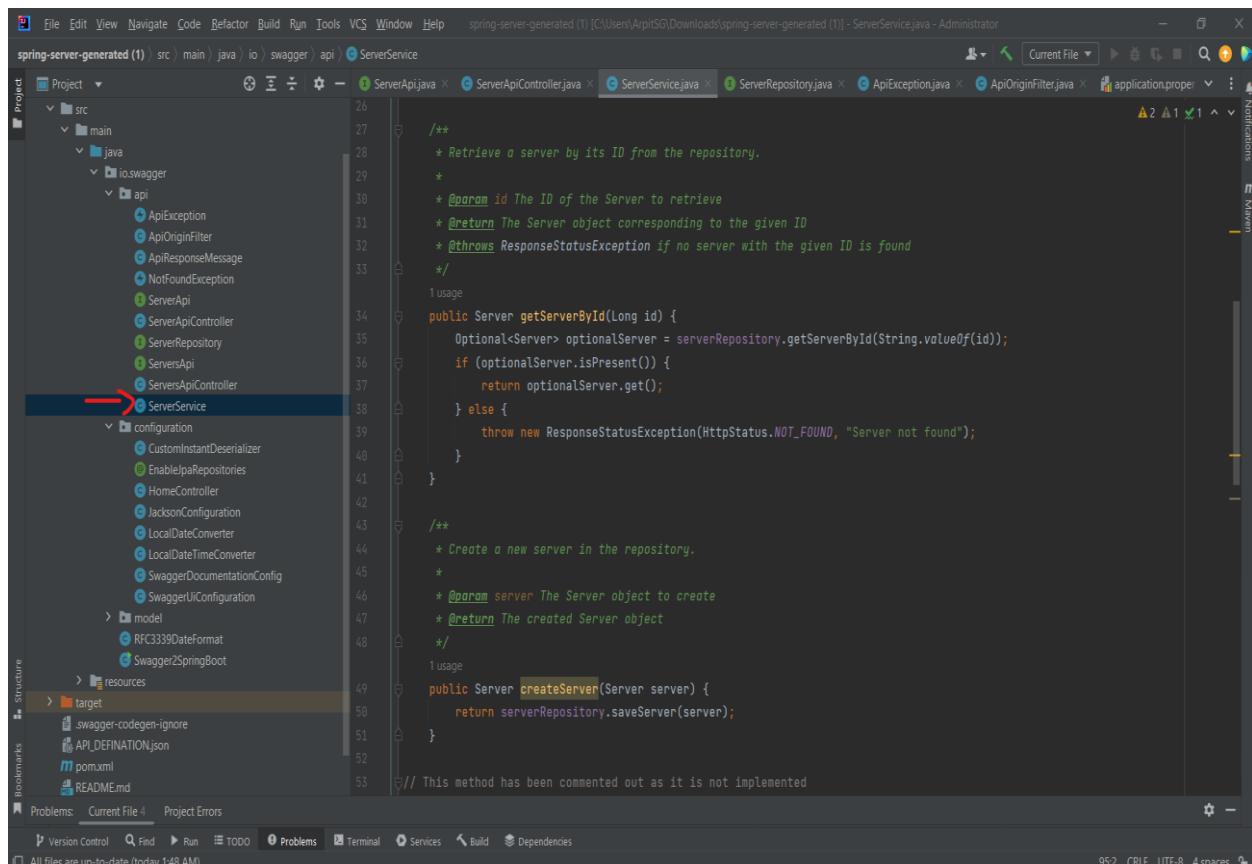


```

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help spring-server-generated (1) [C:\Users\ArpitS\Downloads\spring-server-generated (1)] - ServerService.java - Administrator
Project src main java io.swagger api ServerService.java ServerApiController.java ServerRepository.java ApiException.java ApiOriginFilter.java application.properties
src/main/java/io/swagger/api/ServerService.java
    package io.swagger.api;
    ...
    /**
     * Service for managing servers
     */
    @Service
    public class ServerService {
        private final ServerRepository serverRepository;

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

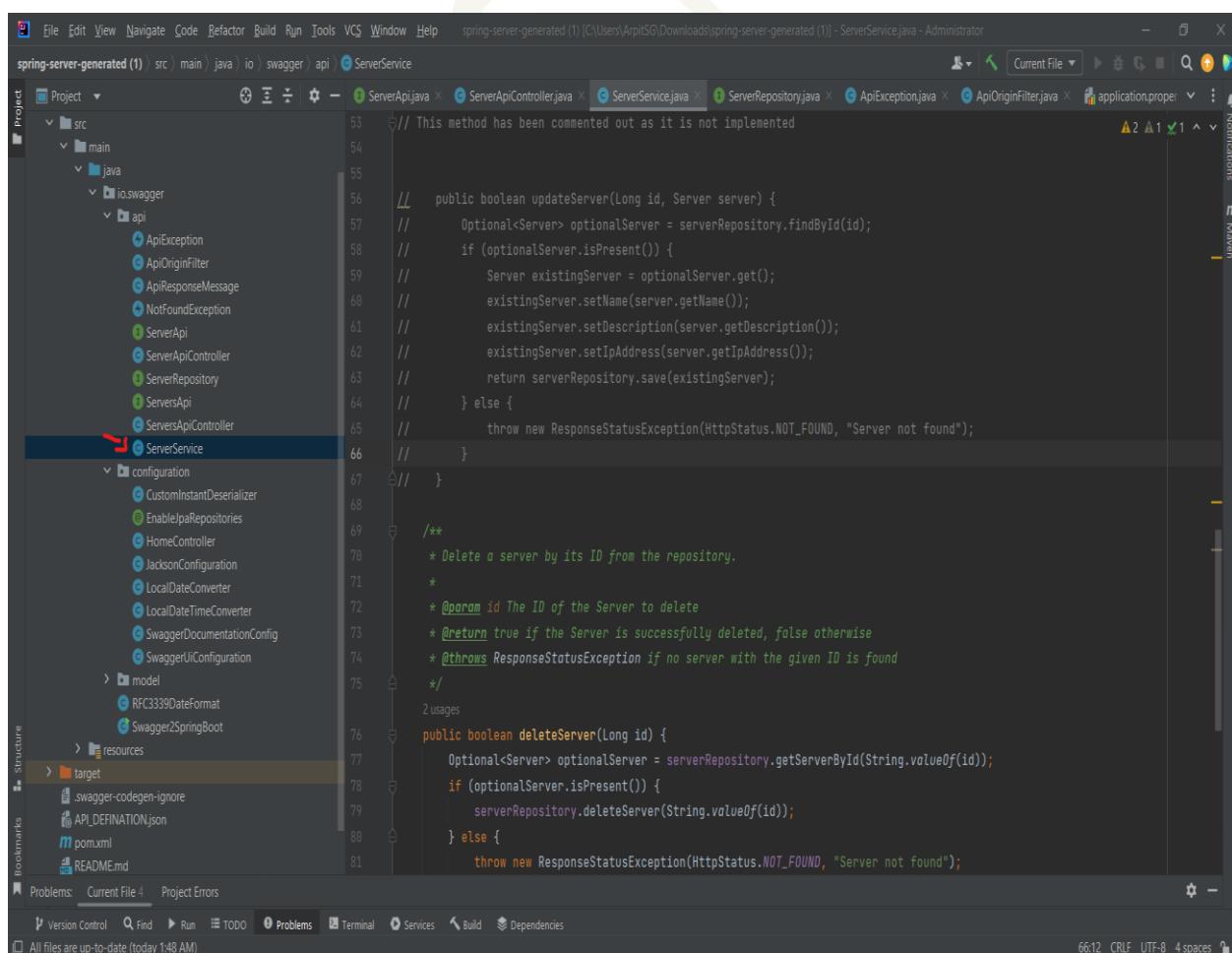
```



```
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
Project src main java io swagger api ServerService.java
26 /**
27  * Retrieve a server by its ID from the repository.
28  *
29  * @param id The ID of the Server to retrieve
30  * @return The Server object corresponding to the given ID
31  * @throws ResponseStatusException if no server with the given ID is found
32  */
33
34 usage
35 public Server getServerById(Long id) {
36     Optional<Server> optionalServer = serverRepository.getServerById(String.valueOf(id));
37     if (optionalServer.isPresent()) {
38         return optionalServer.get();
39     } else {
40         throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Server not found");
41     }
42
43 /**
44  * Create a new server in the repository.
45  *
46  * @param server The Server object to create
47  * @return The created Server object
48  */
49 usage
50 public Server createServer(Server server) {
51     return serverRepository.saveServer(server);
52 }
53 // This method has been commented out as it is not implemented
```

Problems Current File 4 Project Errors

All files are up-to-date (today 1:48 AM)



```
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
Project src main java io swagger api ServerService.java
53 // This method has been commented out as it is not implemented
54
55 //    public boolean updateServer(Long id, Server server) {
56 //        Optional<Server> optionalServer = serverRepository.findById(id);
57 //        if (optionalServer.isPresent()) {
58 //            Server existingServer = optionalServer.get();
59 //            existingServer.setName(server.getName());
60 //            existingServer.setDescription(server.getDescription());
61 //            existingServer.setIpAddress(server.getIpAddress());
62 //            return serverRepository.save(existingServer);
63 //        } else {
64 //            throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Server not found");
65 //        }
66 //    }
67 //} // This method has been commented out as it is not implemented
68
69 /**
70  * Delete a server by its ID from the repository.
71  *
72  * @param id The ID of the Server to delete
73  * @return true if the Server is successfully deleted, false otherwise
74  * @throws ResponseStatusException if no server with the given ID is found
75  */
76
77 usage
78 public boolean deleteServer(Long id) {
79     Optional<Server> optionalServer = serverRepository.getServerById(String.valueOf(id));
80     if (optionalServer.isPresent()) {
81         serverRepository.deleteServer(String.valueOf(id));
82     } else {
83         throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Server not found");
84     }
85 }
```

Problems Current File 4 Project Errors

All files are up-to-date (today 1:48 AM)

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Tree:** The left sidebar shows the project structure under "Project". It includes a "src" folder containing "main" and "io.swagger" packages. "io.swagger" contains "api" and "ServerService" classes. "ServerService" is currently selected and highlighted in blue.
- Code Editor:** The main window displays the content of the "ServerService.java" file. The code is as follows:

```
74     * @throws ResponseStatusException if no server with the given ID is found
75     */
76     public boolean deleteServer(Long id) {
77         Optional<Server> optionalServer = serverRepository.getServerById(String.valueOf(id));
78         if (optionalServer.isPresent()) {
79             serverRepository.deleteServer(String.valueOf(id));
80         } else {
81             throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Server not found");
82         }
83         return false;
84     }
85
86     /**
87      * Retrieve all servers from the repository with a given name.
88      *
89      * @param name The name to search for
90      * @return List of all Server objects with the given name
91     */
92     public List<Server> findServersByName(String name) {
93         return serverRepository.findServersByName(name);
94     }
95
96 //Code Written By @Arpit Singh - 19BCG10669
```

The code editor has several annotations and markers:

- Annotations: "@throws", "@param", "@return".
- Usage markers: "2 usages" for the "deleteServer" method.
- Code quality markers: "A 2 A 1" (green), "1" (blue), and "1" (yellow).
- File status: "Current File" (blue).

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.

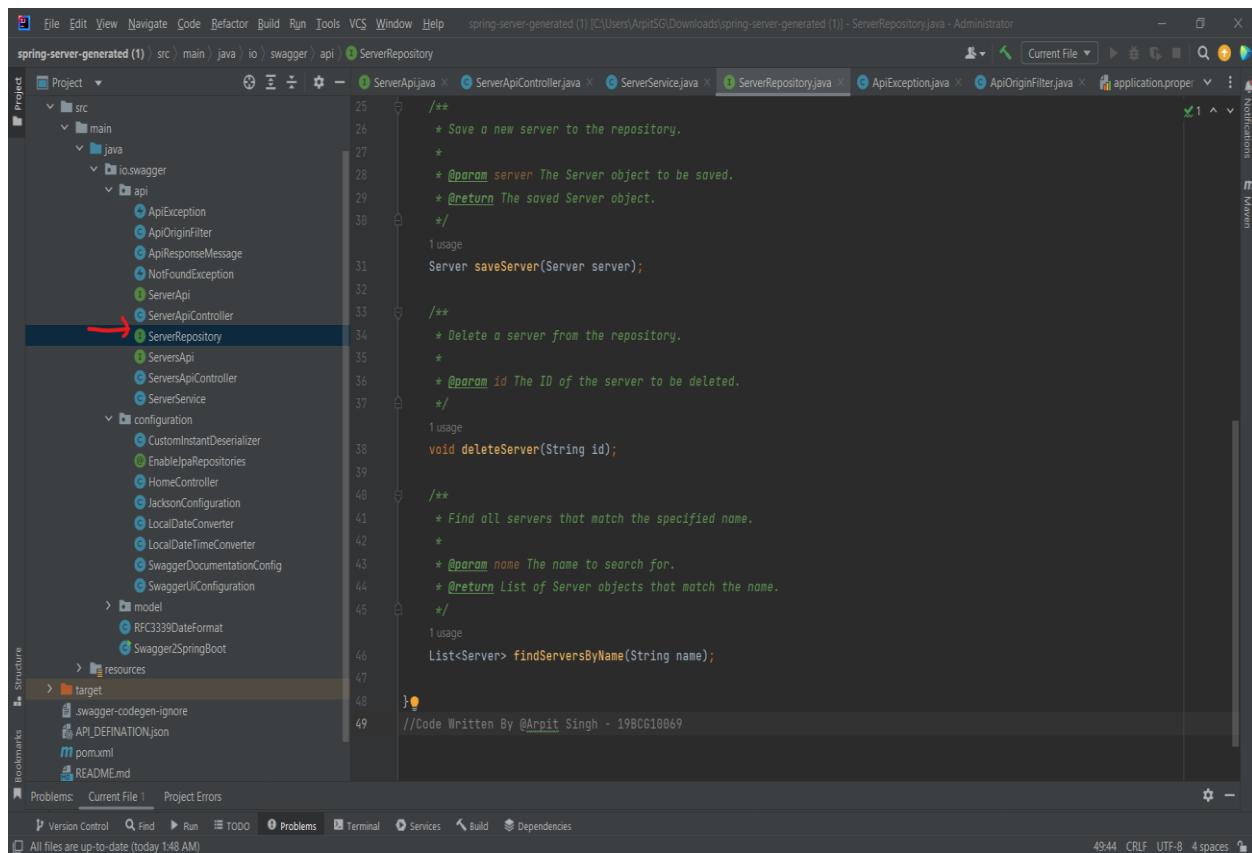
e. Lastly, in this “**api**” package add an Interface called “**ServerRepository**”:

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

The screenshot shows an IDE interface with the following details:

- Project Tree:** The project structure is visible under the "Project" tab, showing packages like "io.swagger.api" and "io.swagger.configuration". A red arrow points to the "ServerRepository" class within the "io.swagger.api" package.
- Code Editor:** The main editor window displays the code for `ServerRepository`. The code defines a public interface with methods for getting all servers and getting a specific server by ID. It also includes annotations for Swagger documentation.
- Toolbars and Status Bar:** Standard IDE toolbars and a status bar at the bottom indicating "47:1 CRLF 4 spaces".

```
1 package io.swagger.api;
2
3 import io.swagger.model.Server;
4 import org.springframework.stereotype.Repository;
5
6 import java.util.List;
7 import java.util.Optional;
8
9 /**
10  * Get a list of all servers.
11  *
12  * @return List of Server objects.
13  */
14
15 /**
16  * Get a specific server by its ID.
17  *
18  * @param id The ID of the server to retrieve.
19  * @return An Optional containing the Server object if found, or an empty Optional if not found.
20  */
21
22
23 /**
24  * Save a new server to the repository.
25  */
26
```



```

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help spring-server-generated (1) | C:\Users\Arpit Singh\Downloads\spring-server-generated (1) - ServerRepository.java - Administrator
spring-server-generated (1) src main java io swagger api ServerRepository
Project Current File Notifications Maven
src
  main
    java
      io.swagger
        api
          ApiException
          ApiOriginFilter
          ApiResponseMessage
          NotFoundException
          ServerApi
          ServerApiController
          ServerRepository
          ServersApiController
          ServerService
        configuration
          CustomInstantDeserializer
          EnableJpaRepositories
          HomeController
          JacksonConfiguration
          LocalDateConverter
          LocalDateTimeConverter
          SwaggerDocumentationConfig
          SwaggerUiConfiguration
        model
          RFC3339DateFormat
          Swagger2SpringBoot
      resources
        target
          swagger-codegen-ignore
          API_DEFINITION.json
      pom.xml
      README.md
Problems Current File Project Errors
Version Control Find Run TODO problems Terminal Services Build Dependencies
All files are up-to-date (today 1:48 AM)
49:44 CRLF UTF-8 4 spaces

```

The screenshot shows the IntelliJ IDEA interface with the code editor open. The file is `ServerRepository.java`. The code defines a repository interface with methods for saving and deleting servers, and finding servers by name. A red arrow points to the `ServerRepository` class in the code.

```


/*
 * Save a new server to the repository.
 *
 * @param server The Server object to be saved.
 * @return The saved Server object.
 */
1 usage
Server saveServer(Server server);

/**
 * Delete a server from the repository.
 *
 * @param id The ID of the server to be deleted.
 */
1 usage
void deleteServer(String id);

/**
 * Find all servers that match the specified name.
 *
 * @param name The name to search for.
 * @return List of Server objects that match the name.
 */
1 usage
List<Server> findServersByName(String name);
}

// Code Written By Arpit Singh - 19BCC10069


```

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.

The screenshot shows the IntelliJ IDEA interface with the project 'spring-server-generated' open. The code editor displays the `Swagger2SpringBoot.java` file, which contains the main entry point for the application. Below the code editor is a terminal window showing the application's startup logs. The logs indicate the application is starting on port 8080, using Tomcat, and initializing various components like the servlet engine and the root web application context. A red arrow points to the line 'Started Swagger2SpringBoot in 5.863 seconds (JVM running for 6.874)' in the log output.

```
package io.swagger;
import ...
@Usage
@SpringBootApplication
@EnableOpenApi
@ComponentScan(basePackages = { "io.swagger", "io.swagger.api", "io.swagger.configuration" })
public class Swagger2SpringBoot implements CommandLineRunner {
    public static void main(String[] args) {
        SpringApplication.run(Swagger2SpringBoot.class, args);
    }
}
```

```
2023-03-28 01:57:26.580 INFO 5320 --- [           main] io.swagger.Swagger2SpringBoot        : Starting Swagger2SpringBoot on DESKTOP-33BMS8 with PID 5320 (E:\Kaibur_Tasks\Check1-main\spring-server-generated\target\classes)
2023-03-28 01:57:26.585 INFO 5320 --- [           main] io.swagger.Swagger2SpringBoot        : No active profile set, falling back to default profiles: default
2023-03-28 01:57:29.492 INFO 5320 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-03-28 01:57:29.552 INFO 5320 --- [           main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-03-28 01:57:29.552 INFO 5320 --- [           main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.37]
2023-03-28 01:57:29.783 INFO 5320 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-03-28 01:57:29.784 INFO 5320 --- [           main] o.s.web.context.ContextLoader        : Root WebApplicationContext: initialization completed in 2917 ms
2023-03-28 01:57:30.932 INFO 5320 --- [           main] o.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2023-03-28 01:57:31.635 INFO 5320 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2023-03-28 01:57:31.640 INFO 5320 --- [           main] io.swagger.Swagger2SpringBoot        : Started Swagger2SpringBoot in 5.863 seconds (JVM running for 6.874)
```

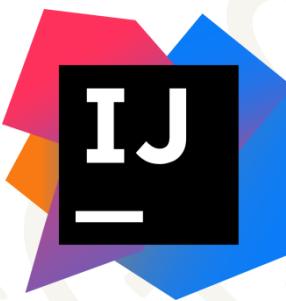
Explanation (Code):

- ❖ **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/{fid}>*
- *To create a new server: POST <http://localhost:8080/servers>*
- *To delete a server by ID: DELETE <http://localhost:8080/servers/{fid}>*
- *To search for servers by name:*
[`http://localhost:8080/servers/search?name={name}`](http://localhost:8080/servers/search?name={name})

5. Make sure to replace `{fid}` 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.

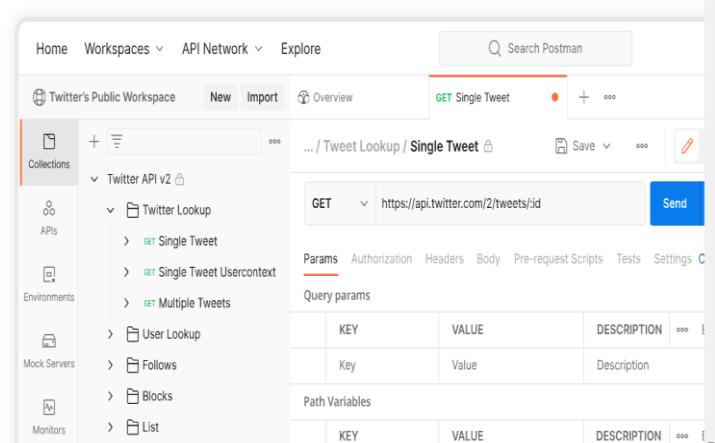
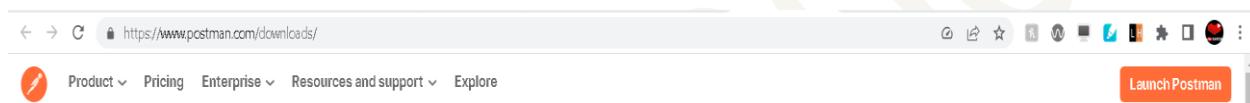


API Testing/Output:

- 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**.

The screenshot shows the 'My Workspace' section of the API Tester application. On the left is a sidebar with icons for Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main area displays a workspace titled 'Task_1(API-Test-Kaiburr) / http://localhost:8080/servers'. A red arrow points to the 'New' button at the top of the workspace header.

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

The screenshot shows the 'Task_1(API-Test-Kaiburr)' workspace. The 'Body' tab is selected in the navigation bar. A red arrow points to the dropdown menu where 'GET' is selected. The dropdown also lists POST, PUT, PATCH, DELETE, COPY, HEAD, OPTIONS, LINK, UNLINK, PURGE, LOCK, UNLOCK, PROPFIND, and VIEW.

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

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' selected, containing 'Collections', 'APIs', 'Environments', 'Mock Servers', 'Monitors', and 'Flows'. The main area shows a task named 'Task_1(API-Test-Kaiburr) / http://localhost:8080/servers'. A red arrow points to the URL field 'http://localhost:8080/servers'. Below the URL, the method 'GET' is selected. The 'Headers' tab is active, showing 'Content-Type: application/json'. The 'Body' tab is also visible. A red arrow points to the 'Send' button at the top right.

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.

The screenshot shows the Postman interface with a task named 'Task_1(API-Test-Kaiburr) / http://localhost:8080/servers/search?name=my centos1'. A red arrow points to the URL field 'http://localhost:8080/servers/search?name=my centos1'. Below the URL, the method 'GET' is selected. The 'Params' tab is active, showing a table with one row: 'name' with value 'my centos1'. Other tabs like 'Authorization', 'Headers', 'Body', 'Pre-request Script', 'Tests', and 'Settings' are visible. A red arrow points to the 'Send' button at the top right.

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.

The screenshot shows the Postman interface with a task named "Task_1(API-Test-Kaiburr) / http://localhost:8080/servers". The request method is "GET" and the URL is "http://localhost:8080/servers". The "Body" tab is selected, indicated by a red arrow. Below it, the "raw" option is selected from a dropdown menu, also indicated by a red arrow. A JSON object is entered in the text area:

```

1 {
2   ...
3     "id": "3",
4     "name": "my centos2",
5     "language": "Java",
6     "framework": "Spring Boot"
7 }

```

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

The screenshot shows the Postman interface with the "Response" tab selected. The status bar indicates "Status: 200 OK Time: 197 ms Size: 389 B". The response body contains the following JSON data:

```

1 [
2   {
3     "id": "2",
4     "name": "my centos1",
5     "language": "Java",
6     "framework": "Spring Boot"
7   },
8   {
9     "id": "3",
10    "name": "my centos2",
11    "language": "Java",
12    "framework": "Spring Boot"
13  },
14  {
15    "id": "1",
16    "name": "my centos",
17    "language": "Java",
18    "framework": "Spring Boot"
19  }
20 ]

```

➤ **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}**](http://localhost:8080/servers/{id}), where {id} is the ID of the server you want to retrieve. Ex: [**http://localhost:8080/servers/1**](http://localhost:8080/servers/1).
- To *get all servers*, you can send a **GET** request to [**http://localhost:8080/servers**](http://localhost:8080/servers).
- To *delete a server by ID*, you can send a **DELETE** request to [**http://localhost:8080/servers/{id}**](http://localhost:8080/servers/{id}), where {id} is the ID of the server you want to delete. Ex: [**http://localhost:8080/servers/2**](http://localhost:8080/servers/2).
- To *search servers by name*, you can send a **GET** request to [**http://localhost:8080/servers/search?name={name}**](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**](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}*.



POSTMAN

❖ Output:

➤ Create/Add(POST) - <http://localhost:8080/servers>

- **BODY:**

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

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'Task_1(API-Test-Kaibur)'. The main area shows a 'Task_1(API-Test-Kaibur) / http://localhost:8080/servers' collection. A red arrow points from the text above to the 'POST' method and URL in the request header. Another red arrow points to the JSON body in the 'Body' tab. A third red arrow points to the status bar at the bottom right, which shows 'Status: 201 Created'.

POST 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

```
1  
2 ... "id": "1",  
3 ... "name": "my centos",  
4 ... "language": "Java",  
5 ... "framework": "Spring Boot"
```

Status: 201 Created Time: 36 ms Size: 265 B Save as Example

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

The screenshot shows the Postman application interface. On the left, the sidebar includes sections for My Workspace, Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main workspace displays a task named "Task_1(API-Test-Kaiburr)" which contains a GET request to "http://localhost:8080/servers/1". The "Body" tab of the request configuration is highlighted with a red box and a red arrow pointing to it. The response body, shown under the "Test Results" tab, is a JSON object:

```
1
2   "id": "1",
3   "name": "my centos",
4   "language": "Java",
5   "framework": "Spring Boot"
```

The response status is 200 OK, with a time of 11 ms and a size of 237 B. A "Save as Example" button is also visible.

➤ Search by Name (GET) -

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

The screenshot shows the Postman application interface. On the left, the sidebar includes 'My Workspace' (selected), 'Collections', 'APIs', 'Environments', 'Mock Servers', 'Monitors', 'Flows', and 'History'. The main workspace shows a collection named 'Task_1(API-Test-Kaibur)' containing several requests: a POST to 'http://localhost:8080/servers', two GET requests to 'http://localhost:8080/servers', and a DELETE request to 'http://localhost:8080/servers/1'. A red arrow points from the text 'GET http://localhost:8080/servers/search?name=my centos1' to the 'Send' button. Below the requests, the 'Query Params' section shows a table with one row: 'name' (value: 'my centos1'). A red arrow points from the 'my centos1' value to the 'Pretty' tab in the response body panel. The response panel at the bottom shows a status of '200 OK' with a time of '14 ms', a size of '240 B', and a 'Save as Example' button. The 'Body' tab is selected, displaying a JSON response:

```
1 [  
2   {  
3     "id": "2",  
4     "name": "my centos1",  
5     "language": "Java",  
6     "framework": "Spring Boot"  
7   }  
8 ]
```

➤ [View all the Servers \(GET\) - http://localhost:8080/servers](http://localhost:8080/servers)

The screenshot shows the Postman application interface. On the left, the sidebar includes 'My Workspace', 'Collections' (with 'Task_1(API-Test-Kaiburr)' selected), 'APIs', 'Environments', 'Mock Servers', 'Monitors', and 'Flows'. The main workspace displays a 'Task_1(API-Test-Kaiburr)' collection. Under 'Task_1(API-Test-Kaiburr)', there is a 'POST http://localhost:8080/servers' and a 'GET http://localhost:8080/servers' request. The 'GET' request is highlighted with a red arrow pointing to its URL in the 'Method' dropdown. Another red arrow points to the 'JSON' tab in the 'Body' section, where the response body is shown as:

```
[{"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"}]
```

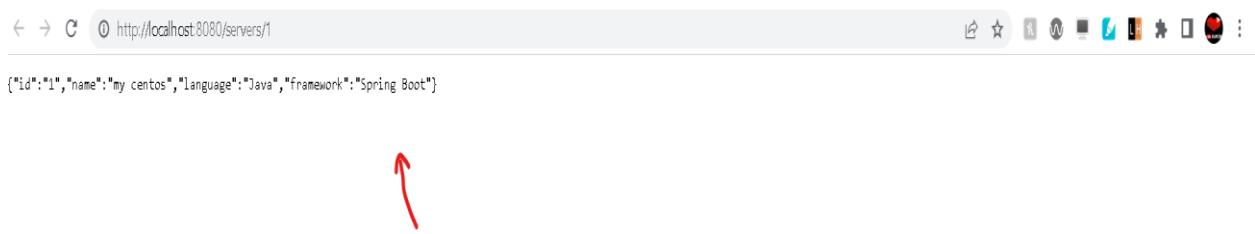
The status bar at the bottom indicates 'Status: 200 OK Time: 197 ms Size: 389 B'.

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

The screenshot shows the Postman application interface. On the left, the sidebar includes sections for Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The 'Mock Servers' section is currently selected. In the main workspace, a collection named 'Task_1(API-Test-Kaiburn)' is expanded, showing four items: a POST request to 'http://localhost:8080/servers', three GET requests to 'http://localhost:8080/servers/1', and a DELETE request to 'http://localhost:8080/servers/1'. The DELETE request is highlighted with a red arrow pointing to it. Below the requests, the 'Query Params' tab is active, showing two rows for 'Key' and 'Value'. At the bottom of the request panel, there is a 'Send' button and a status bar indicating 'Status: 200 OK Time: 44 ms Size: 123 B'. A large red arrow points down from the request to this status bar. The bottom navigation bar contains links for Online, Find and Replace, Console, Cookies, Capture requests, Runner, Trash, and a settings icon.

Browser (Google Chrome Output):

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

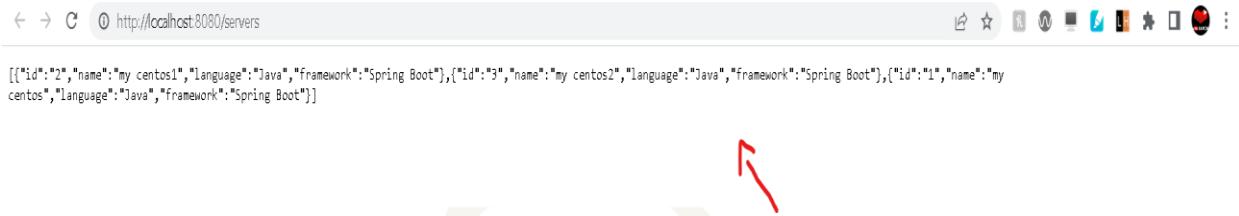


A screenshot of a Google Chrome browser window. The address bar shows the URL <http://localhost:8080/servers/1>. The page content displays a JSON object:

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

 A red arrow points upwards from the bottom of the JSON object towards the top of the browser window.

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

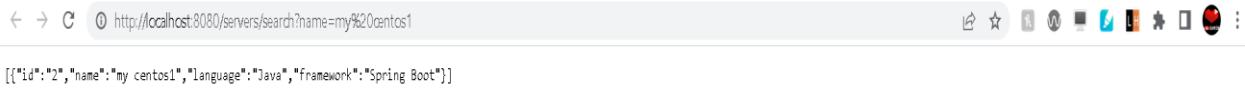


A screenshot of a Google Chrome browser window. The address bar shows the URL <http://localhost:8080/servers>. The page content displays a JSON array:

```
[{"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"}]
```

 A red arrow points upwards from the bottom of the JSON array towards the top of the browser window.

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



A screenshot of a Google Chrome browser window. The address bar shows the URL <http://localhost:8080/servers/search?name=my centos1>. The page content displays a JSON array:

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

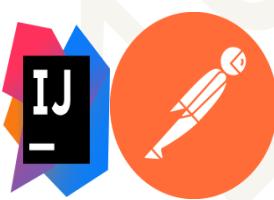
 A red arrow points upwards from the bottom of the JSON array towards the top of the browser window.



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



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.

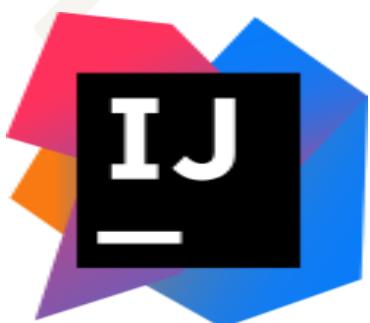


❖ **IntelliJ (IDE):**

- 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!

19BCG10069