Task-1:

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

```
{
 “name” : ” my centos” ,
 “id” :  “123” ,
 “language” :” java” ,
 “framework” :” django”
}
```

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

Sol – 1:

implementation of the application in Java using Spring Boot and MongoDB:

First, we'll define the `Server` class and a `ServerRepository` interface for interacting with MongoDB:

```
package com.example.demo;


import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;


@Document(collection = "servers")

public class Server {
    @Id
    private String id;
    private String name;
    private String language;
    private String framework;

    public Server(String name, String language, String framework) {
        this.name = name;
        this.language = language;
        this.framework = framework;
    }
    public String getId() {
        return id;
    }
    public String getName() {
        return name;
    }
    public String getLanguage() {
        return language;
    }
    public String getFramework() {
        return framework;
    }
}
```

```
package com.example.demo;

import org.springframework.data.mongodb.repository.MongoRepository;
import java.util.List;

public interface ServerRepository extends MongoRepository<Server, String> {
    List<Server> findByNameContaining(String name);
}
```

Next, we'll define the REST controller with the required endpoints:

```
package com.example.demo;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

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

@RestController
public class ServerController {
    @Autowired
    private ServerRepository serverRepository;

    @GetMapping("/servers")
    public List<Server> getServers(@RequestParam(required = false) String id) {
        if (id != null) {
            Optional<Server> server = serverRepository.findById(id);
            if (server.isPresent()) {
                return List.of(server.get());
            } else {
                throw new NotFoundException();
            }
        } else {
            return serverRepository.findAll();
        }
    }

    @PutMapping("/servers")
    public Server putServer(@RequestBody Server server) {
        return serverRepository.save(server);
    }

    @DeleteMapping("/servers/{id}")
    public ResponseEntity<Void> deleteServer(@PathVariable String id) {
        Optional<Server> server = serverRepository.findById(id);
        if (server.isPresent()) {
            serverRepository.deleteById(id);
            return ResponseEntity.ok().build();
        } else {
            throw new NotFoundException();
        }
    }

    @GetMapping("/servers/findByName")
    public List<Server> getServersByName(@RequestParam String name) {
        List<Server> servers = serverRepository.findByNameContaining(name);
        if (servers.isEmpty()) {
            throw new NotFoundException();
        } else {
            return servers;
        }
    }

    @ResponseStatus(HttpStatus.NOT_FOUND)
    public class NotFoundException extends RuntimeException {}
}
```

Finally, we'll configure the MongoDB connection in the `application.properties` file:

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

To run the application, we can use the following main class:

```
package com.example.demo;

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

@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
```

}

To test the application, we can use any HTTP client such as Postman or cURL. For example, we can use the following requests:

- GET all servers:

GET http://localhost:8080/servers

- GET server by ID:

GET http://localhost:8080/servers?id=123

- PUT a new server:

```
PUT http://localhost:8080/servers
{
  "name": "my centos",
  "id":
```