Kaiburr Task 1: Theoretical part

Task 1. Java REST API example.

Implement an application in java which provides a REST API with endpoints for searching,

creating and deleting “server” objects:

● GET servers. Should return all the servers if no parameters are passed. When server id

is passed as a parameter - return a single server or 404 if there’s no such a server.

● PUT a server. The server object is passed as a json-encoded message body. Here’s an

example:

{

“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.

Solution:

Implement a Java application that provides a REST API with the following endpoints:

GET /servers - Returns all servers if no parameters are passed. If a server ID is passed, return a single server or 404 if it does not exist.

PUT /servers - Adds a new server to the database. Expects a JSON-encoded message body containing the server data (name, id, language, and framework).

DELETE /servers/{id} - Deletes a server from the database by ID.

GET /servers/search? Name={name} - Searches for servers by name, returning one or more servers where the name contains the specified string. Returns 404 if nothing is found.

The server objects should be stored in a MongoDB database. Use Postman, curl, or any other HTTP client to demonstrate how the application responds to requests.

Code snippet:

import com.mongodb.client.MongoClients;

import org.bson.Document;

import org.springframework.web.bind.annotation.\*;

import java.util.ArrayList;

import java.util.List;

@RestController

public class ServerController {

private final ServerRepository repository;

public ServerController(ServerRepository repository) {

this.repository = repository;

}

@GetMapping("/servers")

public List<Server> getServers(@RequestParam(required = false) String id) {

if (id == null) {

return repository.findAll();

} else {

return repository.findById(id).map(List::of).orElseThrow(() -> new NotFoundException("Server not found."));

}

}

@PutMapping("/servers")

public void addServer(@RequestBody Server server) {

repository.insert(server);

}

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

public void deleteServer(@PathVariable String id) {

repository.deleteById(id);

}

@GetMapping("/servers/search")

public List<Server> searchServersByName(@RequestParam String name) {

List<Server> servers = new ArrayList<>();

for (Server server : repository.findAll()) {

if (server.getName().contains(name)) {

servers.add(server);

}

}

if (servers.isEmpty()) {

throw new NotFoundException("No servers found.");

}

return servers;

}

}

The Server class should have properties for name, id, language, and framework.

For the MongoDB database configuration, you can use the following code:

@Configuration

public class MongoConfig {

public @Bean MongoClient mongoClient() {

return MongoClients.create("mongodb://localhost:27017");

}

public @Bean MongoTemplate mongoTemplate() {

return new MongoTemplate(mongoClient(), "mydatabase");

}

}

To test the API with Postman or curl, send requests to the appropriate endpoints with the required parameters. For example:

GET all servers: GET http://localhost:8080/servers

GET server by ID: GET http://localhost:8080/servers?id=123

PUT new server: PUT http://localhost:8080/servers with JSON-encoded message body

{

"name": "my centos",

"id": "123",

"language": "java",

"framework": "django"

}

DELETE server: DELETE http://localhost:8080/servers/123

SEARCH servers by name: GET http://localhost:8080/servers/search?name=centos