Task 1. Java REST API example.

My Code:

**Defining the Server Entity**:

Creating a Server class to represent the server objects. This class has fields like name, id, language, and framework.

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;

// Getters and setters

}

**Creating a MongoDB Configuration:**

Creating a configuration class to connect to the MongoDB instance. We can specify the MongoDB host, port, and other settings in this class.

import org.springframework.context.annotation.Configuration;

import org.springframework.data.mongodb.config.AbstractMongoClientConfiguration;

import org.springframework.data.mongodb.repository.config.EnableMongoRepositories;

@Configuration

@EnableMongoRepositories(basePackages = "com.testpackage.repository")

public class MongoDBConfig extends AbstractMongoClientConfiguration {

@Override

protected String getDatabaseName() {

return "testdb";

}

@Override

public MongoClient mongoClient() {

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

}

}

**Creating a Repository Interface:**

Creating a repository interface for the Server entity. Spring Data MongoDB will provide basic CRUD operations.

import org.springframework.data.mongodb.repository.MongoRepository;

public interface ServerRepository extends MongoRepository<Server, String> {

}

**Creating a REST Controller:**

Creating a REST controller class that defines the REST API endpoints for searching, creating, and deleting servers.

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

@RequestMapping("/servers")

public class ServerController {

private final ServerRepository serverRepository;

@Autowired

public ServerController(ServerRepository serverRepository) {

this.serverRepository = serverRepository;

}

@GetMapping

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

if (name != null) {

return serverRepository.findByNameContaining(name);

}

return serverRepository.findAll();

}

@GetMapping("/{id}")

public ResponseEntity<Server> getServerById(@PathVariable String id) {

Optional<Server> server = serverRepository.findById(id);

return server.map(value -> new ResponseEntity<>(value, HttpStatus.OK))

.orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT\_FOUND));

}

@PutMapping

public ResponseEntity<Server> createServer(@RequestBody Server server) {

Server savedServer = serverRepository.save(server);

return new ResponseEntity<>(savedServer, HttpStatus.CREATED);

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteServer(@PathVariable String id) {

serverRepository.deleteById(id);

return new ResponseEntity<>(HttpStatus.NO\_CONTENT);

}

}

**Testing the Endpoints:**

We can test the endpoints using Postman, curl, or any other HTTP client. Here are some example requests:

GET all servers:

GET http://localhost:8080/servers

GET a server by ID:

GET http://localhost:8080/servers/123

PUT (create) a server:

PUT http://localhost:8080/servers

Request Body:

{

"name": "my centos",

"id": "123",

"language": "java",

"framework": "django"

}

DELETE a server by ID:

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

GET servers by name:

GET <http://localhost:8080/servers?name=centos>

This example provides a basic implementation of the requested REST API using Spring Boot and MongoDB. We can extend it further to include error handling, validation, and additional features as needed.

**Task 2. Swagger codegen.**

**.yaml file**

openapi: 3.0.0

info:

title: Server Management API

version: 1.0.0

paths:

/servers:

get:

summary: Get all servers or search by name

parameters:

- in: query

name: name

schema:

type: string

responses:

'200':

description: OK

'404':

description: Not Found

put:

summary: Create a new server

requestBody:

required: true

content:

application/json:

schema:

$ref: '#/components/schemas/Server'

responses:

'201':

description: Created

delete:

summary: Delete a server by ID

parameters:

- in: path

name: id

schema:

type: string

responses:

'204':

description: No Content

/servers/{id}:

get:

summary: Get a server by ID

parameters:

- in: path

name: id

schema:

type: string

responses:

'200':

description: OK

'404':

description: Not Found

components:

schemas:

Server:

type: object

properties:

id:

type: string

name:

type: string

language:

type: string

framework:

type: string