Springboot MongoDB External Config 2022

**Project Structure**

Graphical user interface

Description automatically generated with medium confidence

**Maven(pom.xml)**

<?xml version="1.0" encoding="UTF-8"?>

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.6.6</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<groupId>com.ddlab.rnd</groupId>

<artifactId>springboot-jpa-MongoDB-external-config-2022</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>springboot-jpa-MongoDB-external-config-2022</name>

<description>springboot-jpa-MongoDB-external-config-2022</description>

<properties>

<java.version>11</java.version>

<spring-cloud.version>2021.0.1</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-mongodb</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-openfeign</artifactId>

</dependency>

<dependency>

<groupId>io.github.openfeign</groupId>

<artifactId>feign-okhttp</artifactId>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-ui</artifactId>

<version>1.6.7</version>

</dependency>

<dependency>

<groupId>de.flapdoodle.embed</groupId>

<artifactId>de.flapdoodle.embed.mongo</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Java Code**

**Main Application**

**@SpringBootApplication**

**@EnableScheduling**

**@EnableFeignClients**

**@EnableMongoRepositories 🡸 This is important**

**@EnableAutoConfiguration(exclude = EmbeddedMongoAutoConfiguration.class)**

**public** **class** SampleMainApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SampleMainApplication.**class**, args);

}

}

**Application Startup where you want to do**

@Component

**public** **class** ApplicationStartUp {

**@EventListener(ApplicationReadyEvent.class)**

public void afterStartup() {

System.***out***.println("Application Started at : " + CommonUtil.*timeNow*());

}

}

**AOP**

@Retention(RetentionPolicy.***RUNTIME***)

@Target(ElementType.***METHOD***)

**public @interface LogTime** {

}

@Aspect

@Slf4j

@Component

**public class TimeTracker** {

@Around("@annotation(LogTime)")

**public Object logTime(ProceedingJoinPoint joinPoint)** **throws** Throwable {

StopWatch stopWatch = **new** StopWatch();

stopWatch.start();

**final** Object proceed = joinPoint.proceed();

stopWatch.stop();

***log***.debug("{} executed in {} seconds", joinPoint.getSignature(), stopWatch.getTotalTimeSeconds());

**return** proceed;

}

}

**MongoDB Configuration**

**@Configuration**

**public** **class** SimpleMongoConfig {

**private** String mongoDBDatabaseName = "test";

@Bean

**public** MongoClient mongo() {

ConnectionString connectionString = **new** **ConnectionString("mongodb://localhost:27017/test");**

**MongoClientSettings mongoClientSettings =** **MongoClientSettings.*builder*().applyConnectionString(connectionString)**

**.build();**

**return** MongoClients.*create*(mongoClientSettings);

}

**@Bean**

**public MongoTemplate mongoTemplate() throws Exception {**

**return new MongoTemplate(mongo(), mongoDBDatabaseName);**

**}**

}

\*\* **The above MongoDB Configuration is not necessarily required. You can also provide the MongoDB details in application.properties file**.

# MongoDB Details

**spring.data.mongodb.uri=mongodb://localhost:27017/test**

**spring.data.mongodb.database=test**

**Controller Layer**

**@RestController**

**@RequestMapping(value = "/admin")**

**public** **class** SampleDataController {

@Autowired

**private** SampleService sampleService;

@PostMapping(path = "/store/emp")

**public** ResponseEntity<String> saveEmpInfo(@RequestBody Employee emp) {

sampleService.storeData(emp);

**return** **new** ResponseEntity<>("Employee Information Saved successfully", HttpStatus.***OK***);

}

@PostMapping(path = "/store/emps")

**public** ResponseEntity<String> saveEmpInfo() {

List<Employee> emps = **new** ArrayList<>();

**for** (**int** i = 0; i < 100; i++) {

Employee emp = **new** Employee();

emp.setFirstName("FirstName-" + i);

emp.setLastName("LastName->" + i);

emp.setCity("City-" + i);

emp.setDesgn("Desgn-" + i);

emp.setSalary(10000 + i);

emps.add(emp);

}

sampleService.storeAllEmployees(emps);

**return** **new** ResponseEntity<>("All Employee Information Saved successfully", HttpStatus.***OK***);

}

@GetMapping(path = "/view/emp/{id}")

**public** ResponseEntity<Employee> getEmpInfo(@PathVariable String id) {

Employee emp = sampleService.getEmployeeById(id);

**return** **new** ResponseEntity<>(emp, HttpStatus.***OK***);

}

@GetMapping(path = "/view/emps/page")

**public** ResponseEntity<List<Employee>> getAllEmployeesByPage(@RequestParam(defaultValue = "0") Integer pageNo,

@RequestParam(defaultValue = "10") Integer pageSize) {

List<Employee> list = sampleService.getDataByPage(pageNo, pageSize);

**return** **new** ResponseEntity<List<Employee>>(list, HttpStatus.***OK***);

}

}

**Service Layer**

**public** **interface** SampleService {

**void** storeData(Employee emp);

Employee getEmployeeById(String id);

**void** storeAllEmployees(List<Employee> emps);

List<Employee> getDataByPage(Integer pageNo, Integer pageSize);

}

**Service Implementation Class**

@Slf4j

**@Service**

**public** **class** SampleServiceImpl **implements** SampleService {

**@Autowired**

**private SampleDataRepository dataRepo;**

**@Autowired**

**private SampleDataPagingViewRepository viewPagingRepo;**

@Override

**public** **void** storeData(Employee emp) {

dataRepo.save(emp);

}

@Override

**public** **void** storeAllEmployees(List<Employee> emps) {

dataRepo.saveAll(emps);

}

@Override

**public** Employee getEmployeeById(String id) {

Optional<Employee> optEmp = dataRepo.findById(id);

**if** (optEmp.isPresent())

**return** optEmp.get();

**return** **null**;

}

@Override

**public** List<Employee> getDataByPage(Integer pageNo, Integer pageSize) {

Pageable paging = PageRequest.*of*(pageNo, pageSize);

Page<Employee> pagedResult = viewPagingRepo.findAll(paging);

**if** (pagedResult.hasContent()) {

**return** pagedResult.getContent();

} **else** {

**return** **new** ArrayList<Employee>();

}

}

}

**Repository Layer**

**@Repository**

**public interface SampleDataRepository extends MongoRepository<Employee, String>** {

🡺 You can also use **CrudRepository** 🡸

}

**@Repository**

**public interface SampleDataPagingViewRepository extends PagingAndSortingRepository<Employee, Long>** {

}

**Entity Layer**

@Data

@Document("employee")

**public** **class** Employee {

@Id

**private** String id;

**private** String firstName;

**private** String lastName;

**private** String desgn;

**private** **long** salary;

**private** String city;

}

**External Config File(application.properties)**

# Spring Boot Related Configuration

server.port=8080

# Spring Boot Logger

logging.level.root=WARN

logging.pattern.console= %d{yyyy-MM-dd HH:mm:ss} [%-5level] [%logger**{36}**] - %msg%n

logging.level.org.springframework.web=ERROR

logging.level.org.hibernate=ERROR

logging.level.com.ddlab.rnd = DEBUG

logging.path = ./logs

logging.file.name=./logs/app-log.txt

# Feign Client Configuration for Spring Boot

feign.httpclient.disableSslValidation=true

feign.httpclient.enabled=false

feign.okhttp.enabled=true

# Swagger Configuration

springdoc.swagger-ui.path=/swagger-ui.html

**How to Run the application in Eclipse**

Graphical user interface, text, application, email

Description automatically generated

Difference between @Controller and @RestController

* @Controller is used to mark classes as Spring MVC Controller.
* @RestController is a convenience annotation that does nothing more than adding the @Controller and @ResponseBody annotations.

So the following two controller definitions should do the same.

@Controller

@ResponseBody

public class MyController { }

@RestController

public class MyRestController { }

If you use @RestController you cannot return a view (By using Viewresolver in Spring/springboot) and yes @ResponseBody is not needed in this case. If you use @Controller you can return a view in Spring web MVC.