**Spring Boot JPA Hibernate Scheduler Feign Client – 2022**

**Introduction:** A sample Spring Boot application developed with the latest version in 2022 and it covers Spring Boot, JPA, Hibernate, Feign Client, Scheduler, External configuration for cloud deployment with Postgress SQL

**Project Structure**

Graphical user interface, application

Description automatically generated with medium confidence

**Maven (pom.xml)**

<project>

<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-Hibernate-external-config-2022</artifactId>

<version>0.0.1-SNAPSHOT</version>

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

<description>springboot-jpa-Hibernate-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-data-jdbc</artifactId>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-web</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.postgresql</groupId>

<artifactId>postgresql</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</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>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 SpringBoot Apllication**

@SpringBootApplication

@EnableScheduling

@EnableFeignClients

@EnableTransactionManagement

**public** **class** MainApplication {

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

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

}

}

**Perform Some Action when SpringBoot Apllication Starts Up**

@Component

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

**@EventListener(ApplicationReadyEvent.class)**

**public** **void** afterStartup() {

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

}

}

**Scheduler in SpringBoot**

@Slf4j

@Component

**public** **class** SampleScheduledTask {

@Autowired

**private** SampleService rackDataService;

@Scheduled(cron = "${servicenow.cron.expn}")

**public** **void** scheduleServiceNowTask() {

***log***.debug("Scheduler stated at: " + CommonUtil.*timeNow*());

***log***.debug("Initiating Scheduler to poll data ...");

rackDataService.storeRackData();

***log***.debug("Data Polling complete ...");

}

}

**AOP in SpringBoot**

@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;

}

}

**Controller Layer**

**@RestController**

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

**public class InfoController** {

@Autowired

**private** SampleService rackDataService;

@GetMapping

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

**return** **new** ResponseEntity<>("C3 Rack Integration", HttpStatus.***OK***);

}

@Hidden

@Deprecated

@GetMapping(path = "/initiate")

**public** ResponseEntity<String> ~~initiate~~() {

rackDataService.storeRackData();

**return** **new** ResponseEntity<>("Just Checking", HttpStatus.***OK***);

}

}

-----

**@RestController**

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

**public class SampleDataController** {

@Autowired

**private** SampleService rackDataService;

@GetMapping(path = "/view/rack/unassigned")

**public** ResponseEntity<List<RackDataEntity>> getAllUnAssignedRackView() {

List<RackDataEntity> allRackDataList = rackDataService.getAllRackData();

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

}

@GetMapping(path = "/view/rack/unassigned/page")

**public** ResponseEntity<List<RackProjectLineItemView>> getUnassignedViewsByPage(

@RequestParam(defaultValue = "0") Integer pageNo,

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

List<RackProjectLineItemView> list = rackDataService.getRackDataByPage(pageNo, pageSize);

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

}

@Hidden

@GetMapping(path = "/view/rack/projectno/{projNum}")

**public** ResponseEntity<RackDataEntity> getRackDataByProjectNumber(

@PathVariable(name = "projNum") String projNum) {

RackDataEntity rackData = rackDataService.getRackDataByProjectNumber(projNum);

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

}

@Hidden

// Experimental

@GetMapping(path = "/view/rack/projectno")

**public** ResponseEntity<RackDataEntity> getRackDataByProjectNumber1(

@RequestParam(defaultValue = "IST00001759") String projNum) {

RackDataEntity rackData = rackDataService.getRackDataByProjectNumber(projNum);

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

}

}

**Service Layer**

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

**void** storeRackData();

List<RackDataEntity> getAllRackData();

List<RackProjectLineItemView> getRackDataByPage(Integer pageNo, Integer pageSize);

List<RackProjectLineItemView> getRackDataByPage(Integer pageNo, Integer pageSize, String sortBy);

RackDataEntity getRackDataByProjectNumber(String projectNumber);

RackDataEntity getRackDataByProjectCrdNumber(String projectCrdNumber);

}

**Service Implementation**

@Slf4j

@Service

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

@Autowired

**private** SampleFeignClient serviceNowclient;

@Autowired

**private** SampleDataRepository rackDataRepo;

@Autowired

**private** SampleDataPagingViewRepository viewRepo;

@LogTime

@Transactional

@Override

**public** **void** storeRackData() {

RackProjectRoot root = serviceNowclient.getSummary();

***log***.debug("Result as string: " + root);

***log***.debug("Resutl Data : " + root.getResult());

List<RackDataEntity> dataList = root.getResult().getProjects();

processAndPersist(dataList);

}

**private** **void** processAndPersist(List<RackDataEntity> dataList) {

List<RackDataEntity> saveList = **new** ArrayList<>();

List<RackDataEntity> toBeDeletedList = **new** ArrayList<>();

**for** (RackDataEntity rde : dataList) {

String projectNo = rde.getProjNumber(); // find by project number

RackDataEntity fromDbEntity = getRackDataByProjectNumber(projectNo);

**if** (fromDbEntity == **null**) {

// Data need to be persisted, store in insert list

***log***.debug("This set of data is new, hence data will stored in Database");

saveList.add(rde);

} **else** **if** (fromDbEntity != **null** && !fromDbEntity.equals(rde)) {

***log***.debug("One set data coming from ServiceNow is not equal to existing "

+ "database entity,\n data in database will be updated");

***log***.debug("DB Entity: {}", fromDbEntity);

***log***.debug("ServiceNow Entity: {}", rde);

toBeDeletedList.add(fromDbEntity);

saveList.add(rde);

}

}

**if** (toBeDeletedList.size() != 0)

rackDataRepo.deleteAll(toBeDeletedList);

**if** (saveList.size() != 0)

rackDataRepo.saveAll(saveList);

}

@Override

**public** List<RackDataEntity> getAllRackData() {

Iterable<RackDataEntity> iterable = rackDataRepo.findAll();

List<RackDataEntity> resultList = **new** ArrayList<>();

iterable.forEach(resultList::add);

**return** resultList;

}

@Transactional

@Override

**public** List<RackProjectLineItemView> getRackDataByPage(Integer pageNo, Integer pageSize) {

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

Page<RackProjectLineItemView> pagedResult = viewRepo.findAll(paging);

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

**return** pagedResult.getContent();

} **else** {

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

}

}

@Transactional

@Override

**public** List<RackProjectLineItemView> getRackDataByPage(Integer pageNo, Integer pageSize, String sortBy) {

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

Page<RackProjectLineItemView> pagedResult = viewRepo.findAll(paging);

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

**return** pagedResult.getContent();

} **else** {

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

}

}

@Transactional

@Override

**public** RackDataEntity getRackDataByProjectNumber(String projectNumber) {

**return** rackDataRepo.findByProjNumber(projectNumber);

}

@Transactional

@Override

**public** RackDataEntity getRackDataByProjectCrdNumber(String projectCrdNumber) {

**return** rackDataRepo.findByProjectCrdNo(projectCrdNumber);

}

}

**Repository Layer**

**@Repository**

**public interface SampleDataRepository extends CrudRepository<RackDataEntity, Integer>** {

RackDataEntity findByProjNumber(@Param("projectNumber") String projectNumber);

RackDataEntity findByProjectCrdNo(@Param("projectCrdNo") String projectCrdNumber);

}

**@Repository**

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

}

**Entity Layer**

@Entity(name = "LineItem")

@Table(name = "rack\_line\_item")

@Data

**public** **class** LineItem **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = -197377295971336591L;

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **long** id;

**For a Materialized View**

@Entity

@Table(name = "rack\_project\_line\_item\_view")

@Data

**public** **class** RackProjectLineItemView {

@Id

@Column(name = "rack\_Id")

**private** **long** rackId;

@Column(name = "rack\_config\_qty")

**private** **int** rackConfigQty;

}

@Column(name = "number")

@JsonProperty("number")

**private** String lineNo;

@Column(name = "rack\_name")

@JsonProperty("rack\_name")

**private** String rackName;

…… …. ….

}

**Feign Client**

@FeignClient(name = "ServiceNowAPI", url = "${servicenow.api.url}",

configuration = SampleClientAuthConfiguration.**class**)

**public** **interface** SampleFeignClient {

@GetMapping(value = "/api/x\_ddcs2\_ist/c3\_integration/ist\_summary")

RackProjectRoot getSummary();

}

**Feign Client Configuration for Authentication**

**public** **class** SampleClientAuthConfiguration {

@Value("${servicenow.username}")

**private** String userName;

@Value("${servicenow.pwd}")

**private** String password;

@Bean

**public** BasicAuthRequestInterceptor basicAuthRequestInterceptor() {

**return** **new** BasicAuthRequestInterceptor(userName, password);

}

}

**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.dell.c3 = DEBUG

logging.path = ./logs

logging.file.name=./logs/c3-rack-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

# Spring JPA Configuration

spring.jpa.properties.hibernate.jdbc.batch\_size = 20

spring.jpa.properties.hibernate.order\_updates=true

spring.jpa.properties.hibernate.generate\_statistics=true

spring.jpa.open-in-view=false

#spring.jpa.properties.hibernate.jdbc.lob.non\_contextual\_creation=true

# PostGreSQL Database Configuration

# Below is Database Software Name

spring.jpa.database=POSTGRESQL

spring.datasource.url=jdbc:postgresql://localhost:5432/test

spring.datasource.username=postgres

spring.datasource.password=postgres

spring.jpa.show-sql=true

#spring.jpa.generate-ddl=true

#spring.jpa.hibernate.ddl-auto=update

# Application Specific Configuration

servicenow.api.url=https://dellesiservicesdev.service-now.com

servicenow.username=C3Integration\_API

servicenow.pwd=Juice TrE3 Script Cold

# Scheduler to run in every 4 hours

#servicenow.cron.expn=0 0 \*/4 \* \* \*

# Scheduler to run in every 3 minute

# servicenow.cron.expn=0 \*/3 \* \* \* \*

# Scheduler to run in every 3 minute

servicenow.cron.expn=0 \*/1 \* \* \* \*

**How to Configure and Run application**

Graphical user interface, text, application

Description automatically generated