PLAN MY GROOVE

author: Stefano Ghio
version: 1.0.0.FINAL

1. Setup

Download the project source from https://github.com/steghio/PlanMyGroove

A JDK 1.7 or later must be available on the system. The *JAVA_HOME* environment variable should point to its location

Note: The project was tested on a Linux CentOS 6 64 bit machine running on VirtualBox. Also, to speed up project setup, the base environment was retrieved from https://spring.io/guides/gs/accessing-data-rest/

2. Classes

2.1. Job

The *Job* object represents the jobs we want to schedule. It has the following attributes:

- **id**: a unique identifier automatically generated at creation time. It cannot be modified
- **jobId**: a user defined String to easily identify the job
- **jobResult**: a String containing the result of the script execution
- **jobCode**: a String containing the Groovy script to evaluate
- **jobStatus**: a String indicating the current job status. Possible values are enumerated in the Job class as:

```
JOB_STATUS_SUBMITTED = "SUBMITTED"
JOB_STATUS_RUNNING = "RUNNING"
JOB_STATUS_COMPLETED = "COMPLETED"
JOB_STATUS_ERROR = "ERROR"
```

2.2 JobRepository

The *JobRepository* interface defines the REST methods that Spring should generate automatically. Some are plain simple and just specify a query parameter, while others rely on hardcoded queries.

2.3 JobQueryController

The *JobQueryController* controller adds custom query functionality that isn't achievable with the standard *@RepositoryRestResource* notation, mainly projected queries. It also contains utility methods to perform simple business logic on the *Job* object.

2.4 JobRunner

The *JobRunner* class defines a runnable object to be executed in its own thread. Its purpose is to constantly query for newly submitted jobs and dispatch them to be executed by the *GroovyRunner* object. Multiple jobs will be started concurrently.

2.5 GroovyRunner

The *GroovyRunner* class defines a runnable object to be initialized by the *JobRunner* object which will run in its own thread. Its purpose is to execute the specified job and store the result. In case of error the *jobResult* attribute will contain the error message.

2.6 Application

Since the application will be packed in a single, executable, uber-JAR, this class binds everything together and launches the *JobRunner* thread at startup as well.

3. Build

- Open a terminal and browse to the project directory *gs-accessing-data-rest-complete*
- If using JDK8, rename *build.gradle* to *build.gradle_jdk7* and rename *build.gradle_jdk8* to *build.gradle*
- Allow execution of the *gradlew* script: `chmod +x gradlew`
- Clean the environment: `./gradlew clean`
- Build the project. It will create a single executable uber-JAR: `./gradlew build`

4. **Run**

After the successful build, run the project with:

```
java -jar build/libs/gs-accessing-data-rest-0.1.0.jar –server.port=8181
```

It's possible to specify a different server port by changing the *server.port* parameter. If omitted, it defaults to 8080

5. **REST API**

Once started, the application will accept HTTP requests at http://localhost:port

It's possible to list some of the available methods with (example):

```
curl http://localhost:8181/jobs/search/
```

This list however will NOT include custom methods which aren't generated by Spring directly as effect of the *@RepositoryRestResource* notation. The custom methods are necessary to perform projected queries; either that or use *@Projection* notation on custom classes, which is still surprisingly hard to find in the docs (https://jira.spring.io/browse/DATAREST-419)

Below is the full methods summary:

- **getAllJobs:** returns all jobs
- **getJobsToSubmit**: returns all uploaded jobs which haven't been yet submitted (*jobStatus* is NULL)
- **getSubmittedJobs**: returns all submitted jobs, which aren't running yet
- **getRunningJobs**: returns all uploaded jobs in running status
- **getCompletedJobs**: returns all completed jobs
- **getErrorJobs**: returns all jobs with resulted in an exception
- **findByJobStatus**: returns all jobs with the specified *jobStatus*
- **findByJobId**: returns all jobs with the specified *jobId*. Multiple jobs can share the same *jobId*, but will always have a unique *id*. To query with that parameter use *findById* instead
- **findById**: returns the job with the specified id
- **getJobResult**: returns the result for the job with the specified *jobId*. In case there are multiple jobs with same *jobId*, it always returns the result of the first one fetched. Better to use *getJobResultById* in this case
- **getJobResultById**: returns the result for the job with the specified *id*

Additional operations are available by using HTTP predicates:

- **submit** a job: POST the job specifying the *jobId* and *jobCode* parameters, eg:

```
curl -i -X POST -H "Content-Type:application/json" -d '{  "jobId" : "myJob",
"jobCode" : "return true"}' http://localhost:8181/jobs

curl -i -X POST -H "Content-Type:application/json" -d '{  "jobId" :
"myOtherJob",  "jobCode" : "return 2**2"}' http://localhost:8181/jobs

#add job that returns an error
curl -i -X POST -H "Content-Type:application/json" -d '{  "jobId" :
"myFailedJob"}' http://localhost:8181/jobs
```

- **delete** a job: issue a DELETE request specifying the job *id*, eg:

```
curl -X DELETE http://localhost:8181/jobs/1
```

6. **TEST**

Testing is done with a separate Java project. It's a single runnable class that issues POST, GET and DELETE requests to test that if the service is up and running and behaves as expected.

It executes the following operations in order:

1. Create three sample jobs:
   - myJob: should return true
   - myOtherJob: should return 4
   - myFailedJob: should return an error
2. List all current (submitted) jobs. There should be none since we waited 5 seconds between the first batch of requests and this one. By removing this sleep, it should be possible to see jobs in either *SUBMITTED* or *RUNNING* status, depending on the machine performance
3. List all completed jobs. We should see both *myJob* and *myOtherJob* listed here
4. Retrieve job result for *myOtherJob*. It should be 4
5. Delete *myFailedJob*
6. Check if *myFailedJob* still exists after deletion. It should not appear