Job Management System - 15.08.2021

1. Intro

I built this Job Management System using Java Spring Boot with Maven and PostgreSQL for persistence. The version of Java is 11.
I used layered architecture divided in persistence, web and service layer.
The interaction with the application is achieved via REST API.
For testing the application you can use an HTTP Client like Postman or similar.
There are JavaDocs for most of the classes and methods for better understanding.
I put Maven dependency for H2 database (In memory database) for the Repository JUnit tests.

Development time: **Around 7h.**

2. Architecture Explanation

- **Model package:** Inside the Model package there is a Job entity defined with all the properties needed. There is also an enumeration that represents the status of the job.
- **Repository package:** This contains a JPA Repository working with the Job entity model.
- **Service package:**

JobService – This is a Spring Bean Service that has async methods that will run on separate threads in parallel without blocking the application for achieving parallel Job execution.

The ***runScheduledJob(Long id, Long delay)*** method is responsible for scheduling the task with a given (dynamic) delay. I defined a ScheduledTask that extends TimerTask and this way the ScheduledTask can be used in a Timer. The Timer has a schedule method that takes the scheduled task and the delay.

The **runScheduledJobPeriodically(Long id, Long delay, Long repeatPeriod)** method is responsible for scheduling the task with a given (dynamic) delay and repeat period.

The ***runJob(Long id)*** method runs a Job with a given id immediately.
Method assumptions:
1. Every created job that is running is calling external API to do the defined job.
2. If the API returns a failed state, the Job status should be set as FAILED.
3. It could be implemented in try/catch block or if statements.

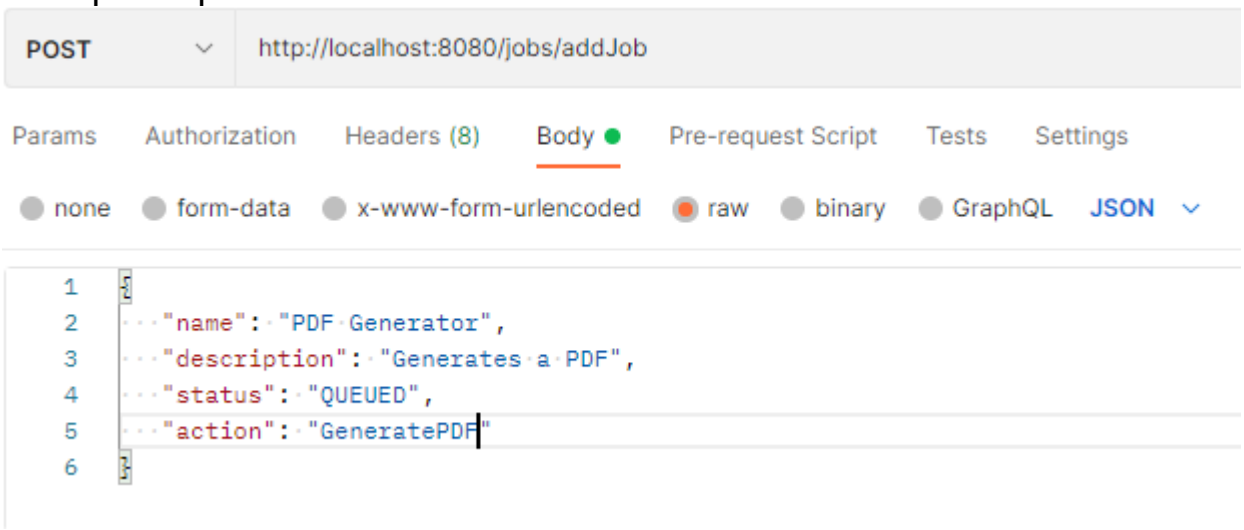
- **Controller package:**

I defined one Controller which has base mapping on “/jobs”. In this Controller I’m using Dependency Injection for the Job Service.

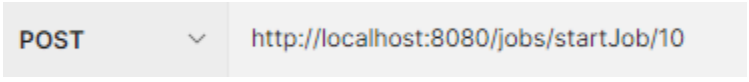
These are the available routes:

POST *jobs*/addJob – this route receives JSON body and creates a new job in the database.

Example request:

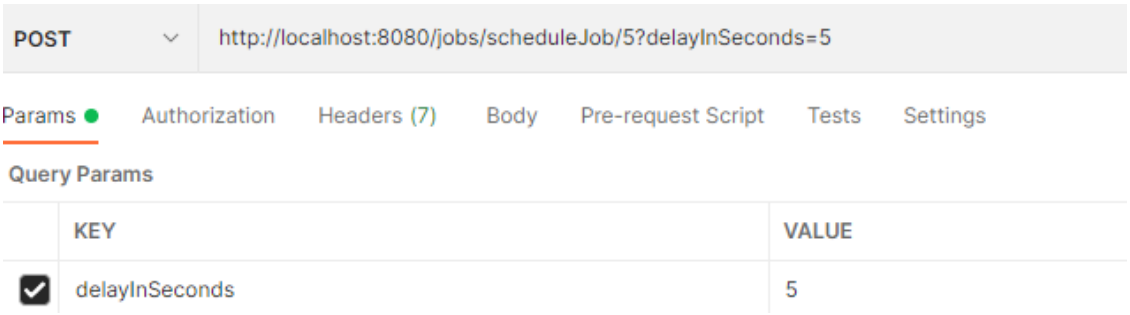


POST *jobs/startJob/{id}* – this starts the Job with a given id immediately.
Example request:

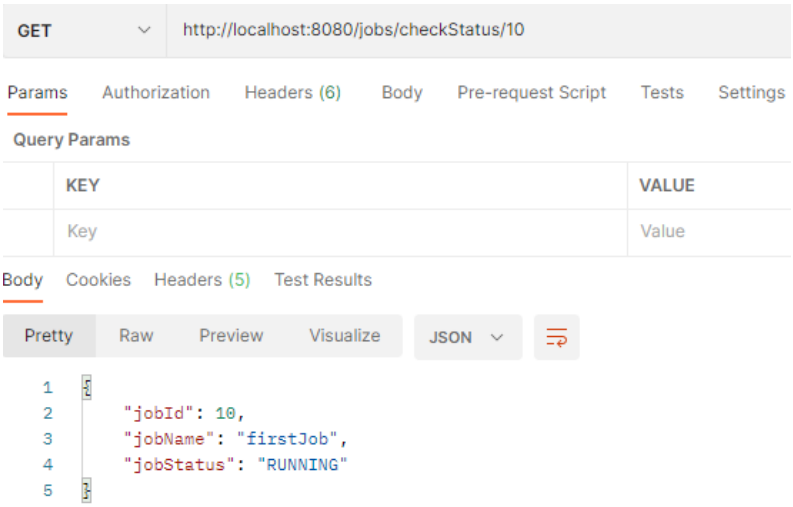


POST jobs/*scheduleJob/{id}* – schedule a Job with a given id as well as the delay for the execution.

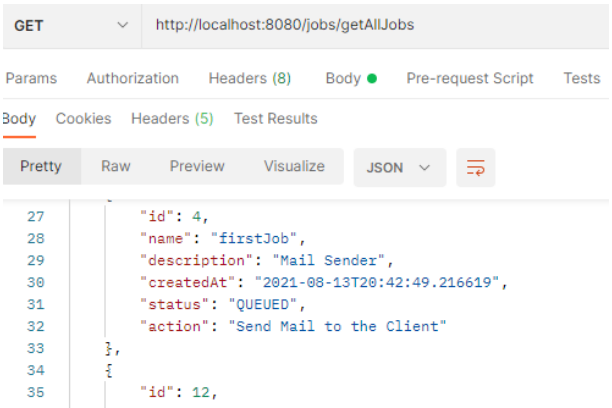
Example request:



GET jobs/checkStatus/*{id}* – this is returning the status of a Job with a given id.
Example request:

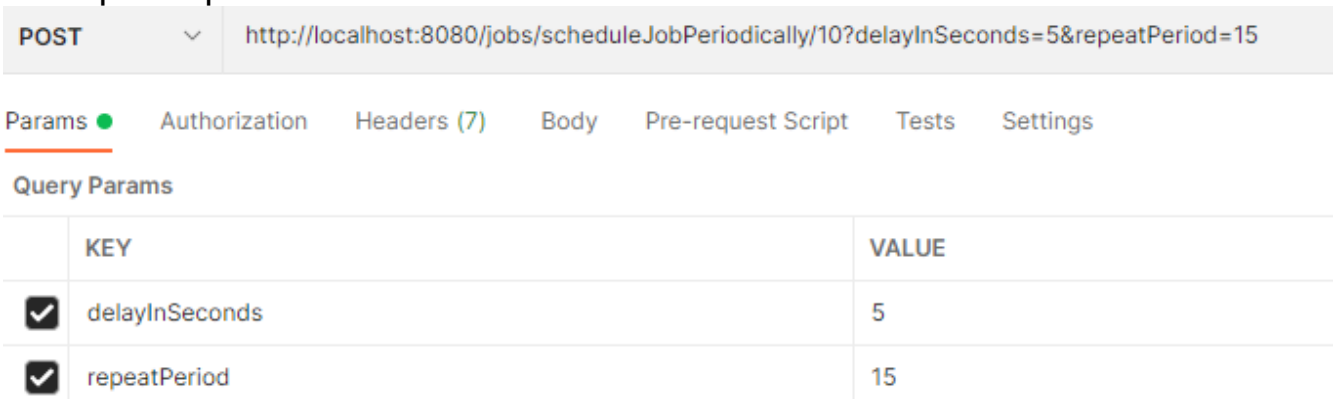


GET jobs/getAllJobs – this route is returning every created Job.
Example request:



POST jobs/scheduleJobPeriodically/*{id}* – this route schedules a Job to run periodically in the future.

Example request:



- Configuration package:
This package contains a configuration used by JobService for the async methods.

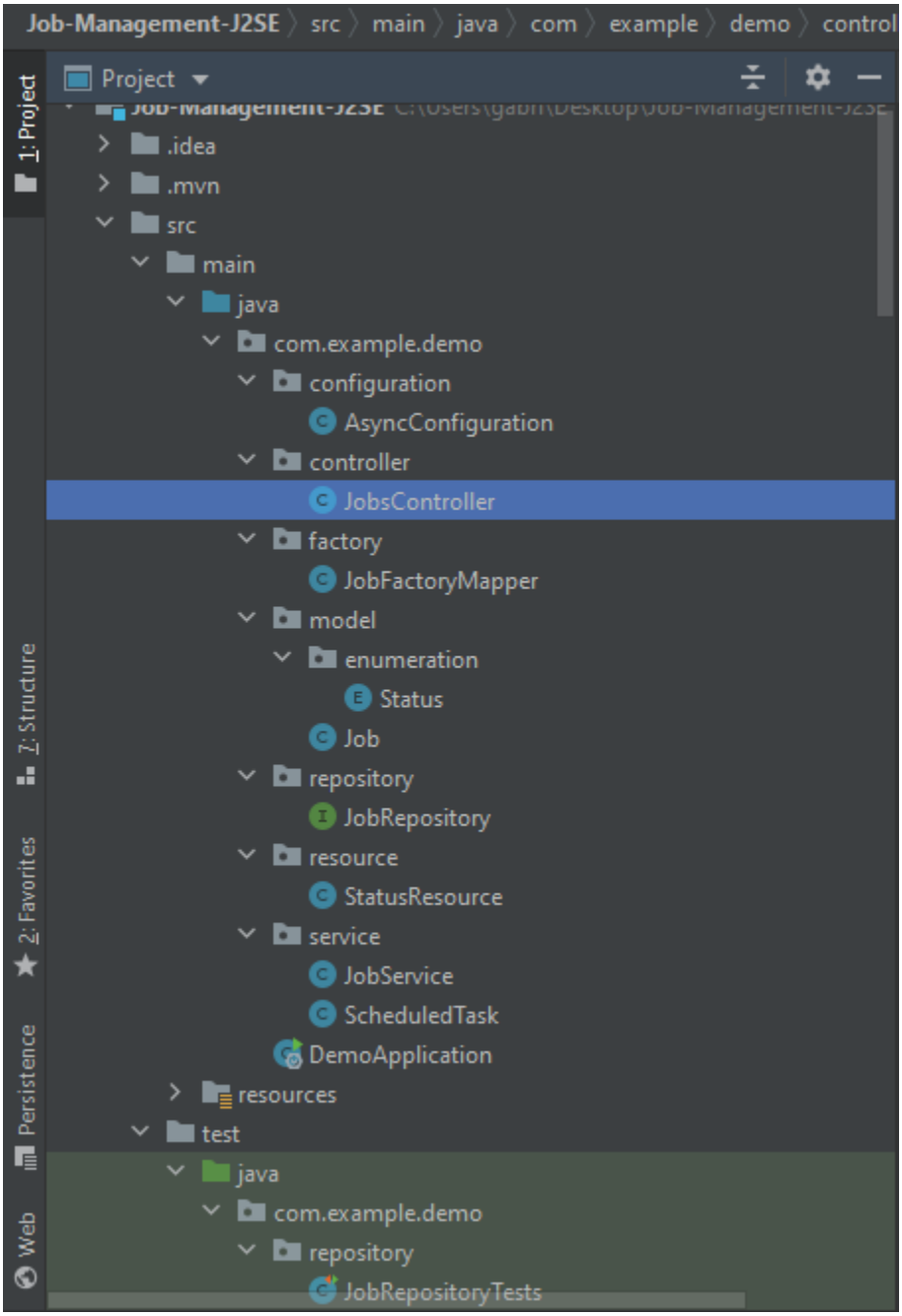
- application.properties file:
Credentials for the database are located in this file.

- Testing:
In the JobRepositoryTests file there are tests for the JPA Job Repository. The tests are using the H2 database by default.

- Resource package:
Contains a status resource that's being return to the client. (In this case for creating JSON).

- Factory package:
This contains a Factory Mapper used to map a Job to a StatusResource. (Factory Design Pattern).

3. Project Structure


Job-Management-J2SE
src
main
java
com
example
demo
control
Project
1: Project
7: Structure
2: Favorites
Persistence
Web
.idea
.mvn
src
main
java
com.example.demo
configuration
AsyncConfiguration
controller
JobsController
factory
JobFactoryMapper
model
enumeration
Status
Job
repository
JobRepository
resource
StatusResource
service
JobService
ScheduledTask
DemoApplication
resources
test
java
com.example.demo
repository
JobRepositoryTests