Job Management System

Technical Design

By,
Nivedita Singh

1. Application Overview

A simple Job Management Service must be developed. The goal of this system is to handle the execution of multiple types of Jobs. The actions performed by these Jobs are not important; possible examples of these Jobs could be performing a data-load into a DWH, performing indexing of some filebased content or sending emails

Features of this system are:

- **Flexibility**
  The types of possible actions performed by the Jobs are not known to the Job Management System. In the future, new Jobs should be supported without re-developing the Job Management System (optional).

- **Reliability**
  Each Job should either complete successfully or perform no action at all. (I.e. there should be no side-effects created by a Job that fails.)

- **Internal Consistency**
  At any one time a Job has one of four states: QUEUED, RUNNING, SUCCESS, FAILED. Following the execution of a Job, it should be left in an appropriate state.

- Priority (Optional)
  Each Job can be executed based on its priority relative to other Jobs

- **Scheduling**
  A Job can be executed immediately or according to a schedule.

High Level Design

Assumptions:

- The Job Details are provided to the system in the form of csv files.
- The files would be upload through a User Interface or previously copied to a folder which can be configured.
- The file upload and processing would be a scheduled process.
- The handlers for different types of jobs would need to be implemented

Scope of Improvement

- The file upload at present handlers upload of files through a user interface as well as can be previously copied.Instead we could post these files on to a rabbitMQ queue which can be picked up by the batch upload process.
- At present the jobs can be configured through the cron expression in application.properties file.We could make it configurable through the UI too.
- We can also make execute immediate and execution_date configurable through the UI.Right now the user can only view the job details.
- At each step of the job upload and processing , we could send notification emails to the uploader regarding the status, failure or success.

The application is developed using JAVA 1.8, microservice architecture,spring,spring RESTFUL API, spring boot, spring security,spring batch.
The UI is developed using HTML,javascript, css,bootstrap etc.

The entire application is divided into 2 modules :

Job Details Uploader Module :

This module lets a user upload a list of job details (The job's name, priority, type, whether it needs to be executed immediately or at a stipulated date and time). The list is in the form of a csv file. The user uploads the csv file from a user interface(http://localhost:8080/index.html) under the Upload Job Details tab.This module is developed using RESTful spring boot web service.The task of this RESTful API is to store the uploaded file onto a folder configured (file.upload.dir property in application.properties file).The rest calls are secured using spring security.

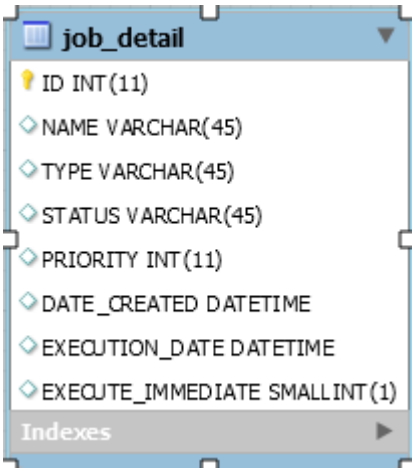
The current job details can be viewed using the View Job Details tab.

Parallely a task scheduler runs whose job is to poll the folder configured (file.upload.dir property in application.properties file) for csv files. Once it finds a file it reads it, processes it and uploads the job details to a database table JOB_DETAIL and sets the record status to QUEUED.It also deletes the read file to avoid filling up space on the device's file system. This module is designed using Spring batch.The task scheduler's schedule can be configured based on a cron expression (recurring.task.cron.expression in the application.properties file).

Job Details Processor Module :
This module polls the database table JOB_dETAIL FOR queued records (status QUEUED) to find jobs whose execution_date is <= current date or whose execute_immediate flag is set to true. Once it finds records based on priority,it starts a batch processor which reads the records , processes it and based on the type of job it makes calls to corresponding handlers (out of scope). Once the process is successful or failed it updates the database record status to SUCCESS ,FAILED OR RUNNING.
This module is also developed using spring batch.

Database Design:

JOB_DETAIL



Code Structure

Package structure :

- Job-uploader-service
  - com.jobUploader.config
    - JobUploadBatchConfiguration - This class is used to configure the batch process for processing job details csv files from a folder and upload it to a database tables
    - SecurityConfig - This class is used to enable web security to the UI application
    - IntergrationConfiguration - This class can be used to implement the Spring Integration feature for polling a folder for files and integrate it with the spring batch process.This is an alternate way instead of using the scheduler. The Component annotation is commented out as this feature is not used.
  - com.jobUploader.controller
    - JobUploadController -- This class is used to as a rest controller To handle the REST calls for Job upload from the UI
  - com.jobUploader.exception - package for different types of exceptions
    - FileStorageException -- This class denotes the exception during file storage to the file system
  - com.jobUploader.helpers - package for helper classes
    - Constants - class to store constants used in the application
    - FileVerificationSkipper - The Class FileVerificationSkipper. It is used to indicate the skip policy during error occurence of csv file reading.
    - JobDetailRowMapper - This bean maps the csv file lines to the JobDetail POJO

        - JobDetailUploadPartitioner - This class is used as a partioner as a precursor to the csv Upload batch process. Here the folder reading operation is partitioned, meaning the reading of the csv files happens parallely

    - com.jobUploader.listener - package for different listeners
        - CustomJobListener - This class indicates the listener which performs different tasks during before and after the job execution
    - com.jobUploader.model - package for different POJOs
        - JobDetail - represents the JobDetail entity
        - Response - represents the Response entity used by the rest controller

    - com.jobUploader.processor
        - FileUploadJobDetailProcessor - The processor which processes the job items before sending it to writer.We set the status to QUEUED

    - com.jobUploader.rabbitmq
        - This folder contains files which are to be used if we need to use a rabbitmq queue to post the csv files instead of posting it to a folder.
        - In this case a rabbitmq server and queue must be configured.We are not using this in our application.
    - com.jobUploader.repository
        - JobDetailRepository - represents the Repository for JobDetail
    - com.jobUploader.scheduler
        - JobUploadToDBTaskScheduler - represents the scheduler which is used to schedule the Upload process job using a cron expression.
    - `com.jobUploader.service
        - Contains service files to be used by the rest controllers

- resources/static - folder which contains the html css and js files for the UI application.
- resources/application.properties - property file to configure the spring batch process, the database etc

- Job-processor-service
    - com.jobProcessor.config
        - JobProcessorJobConfiguration - This class is used to configure the Batch process for processing jobs from database based pon schedule and priority
    - com.jobProcessor.handler
        Contains different handlers to handle different types of jobs.These handlers need to be implemented based on the job.The handlers could make rest calls to different rest apis used to handle the different jobs

    - com.jobProcessor.exception - package for different types of exceptions

    - com.jobProcessor.helpers - package for helper classes
        - Constants - class to store constants used in the application

- com.jobProcessor.listener - package for different listeners

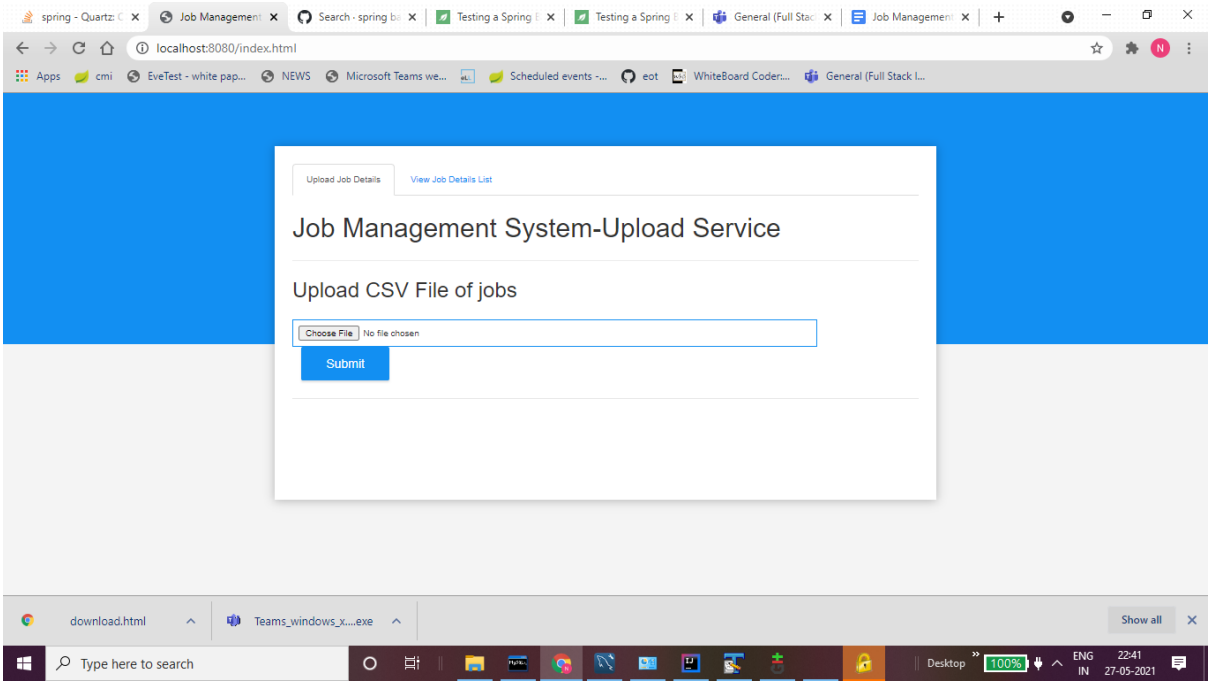
- com.jobProcessor.model - package for different POJOs
    - JobDetail - represents the JobDetail entity

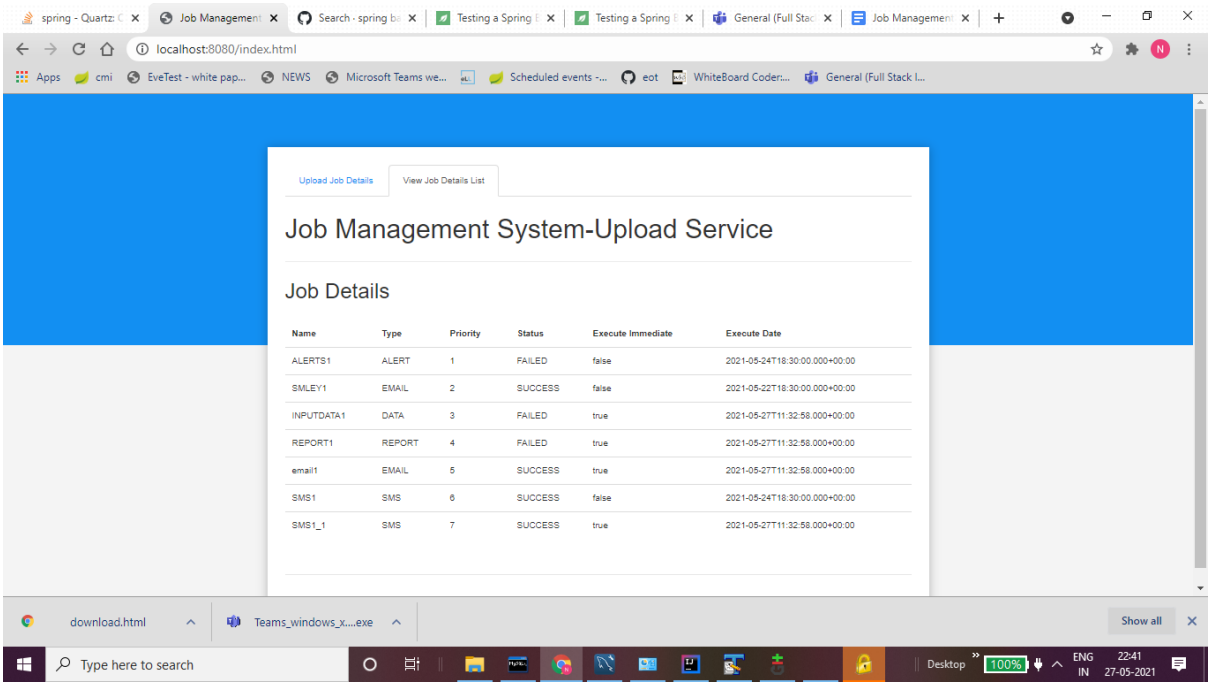
- com.jobProcessor.processor
    - JobMgmtSystmJobItemProcessor- This class is an implementation of ItemProcessor to define processing action of the job items. In our case at the beginning of the processing we set the status to running, based on the type of job we call corresponding handlers and in the end change the status to success or failure

- com.jobProcessor.repository
    - JobDetailRepository - represents the Repository for JobDetail
- com.jobUploader.scheduler
    - JobProcessorTaskScheduler - represents the scheduler which is used to schedule the Job process job using a cron expression.

- resources/application.properties - property file to configure the spring batch process, the database etc

- UI Screens





- Sample csv files are in the folder job-management-system\job-uploader-service\src\test\testcsvs.