**Job Manage System Backend**

This document contains the design details of Batch job management system.

**Overview of System**

BJMS provide necessary API’s for simple batch job management services. This prototype provides following functionalities.

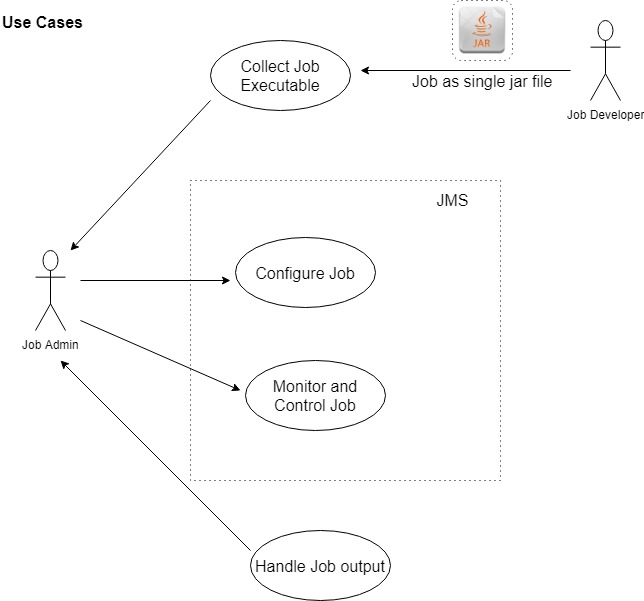
1. Accept job details and store them in H2 db. The details include job name, description, local path to executable Jar file, scheduling configurations, job specific parameters.
2. An instance of a Job represents a single run. When a new job is saved an instance is also created in H2. The instance has initial status “Q” indicates “Queued”.
3. When a new job is created it will be taken for scheduling. Priority of job is considered while submitting job to scheduler. For manually runnable type jobs scheduling is not required at the moment of save.
4. Based on the scheduling configuration, the job scheduler picks up the job for running. The Queued instance of the job is updated with started time and status as “Running” in H2.
5. To run the job, a command string is created as below. If the job is configured with custom parameters, they will also be passed to the command string. The job would start as a child process and the running task waits for its completion.

*Java –jar <path to exec jar path> <param1> <param2> <param3>* …

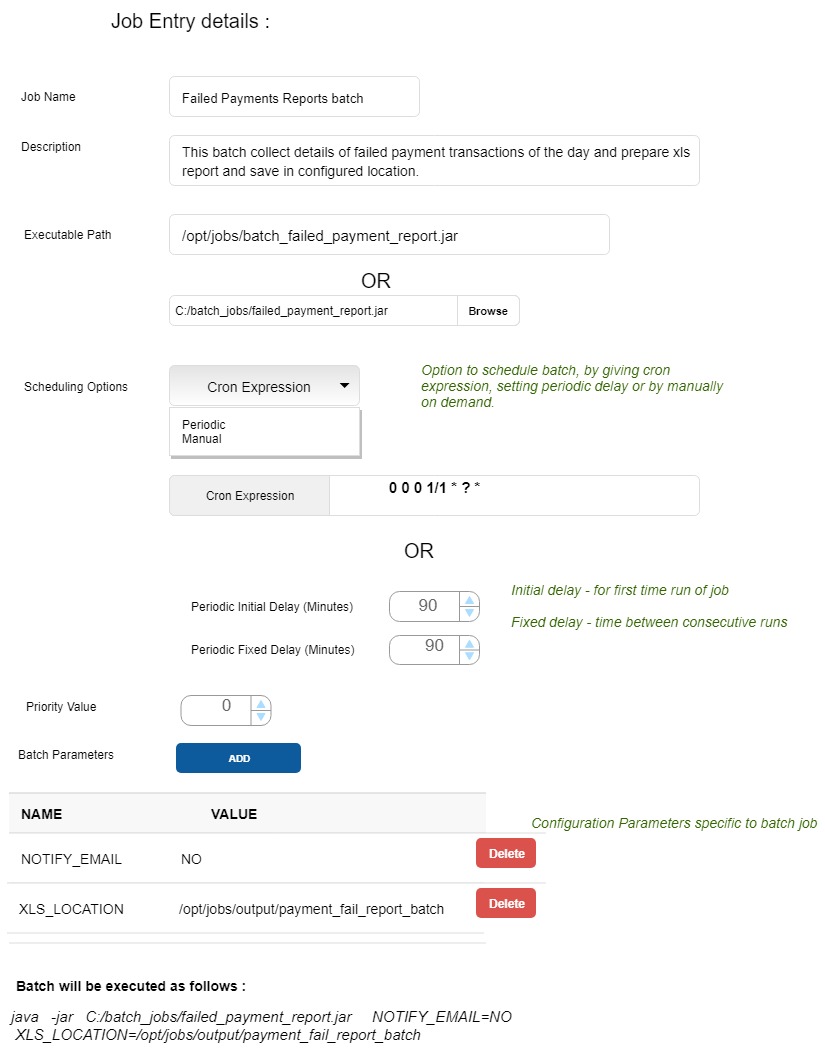
The job outputs and error streams are directed to a file with name <job\_ id>\_<instance\_id>\_<timestamp>.log Eg: 2\_54\_1526742870015.log

1. If any error happens while starting the process, the job instance will be updated with status “Failed” and end time in H2.
2. If the job finish without error (not counting error happen or not inside the job) then the job instance is updated with “Success” status and end time.
3. Before the scheduled run get finished, another instance of the job is created with status “Queue” in H2. Which would be picked up by the schedule run task next time.
4. Any details of a job can be updated. There must not be any running instance at the time of updating the job. Following the update of the job is rescheduled (existing Queued instance is deleted, scheduling is cancelled and submitted to scheduler with new configurations). No scheduling is needed for manually runnable jobs. If job active flag is not “Y”, it will not be taken for scheduling.
5. Manually runnable jobs are only scheduled for immediate run when requested through different API.
6. API to get all jobs with recent instance, get all instances of a job, get job output/ error files etc.
7. On application restart scenarios, the jobs details are taken from H2 and used for scheduling operations. Scheduling operation to consider priority of jobs based on limit of maximum no of jobs permitted in scheduler. Based on priority any scheduled job has to be moved out if it have lower priority than a newly came job and scheduler is exhausted.

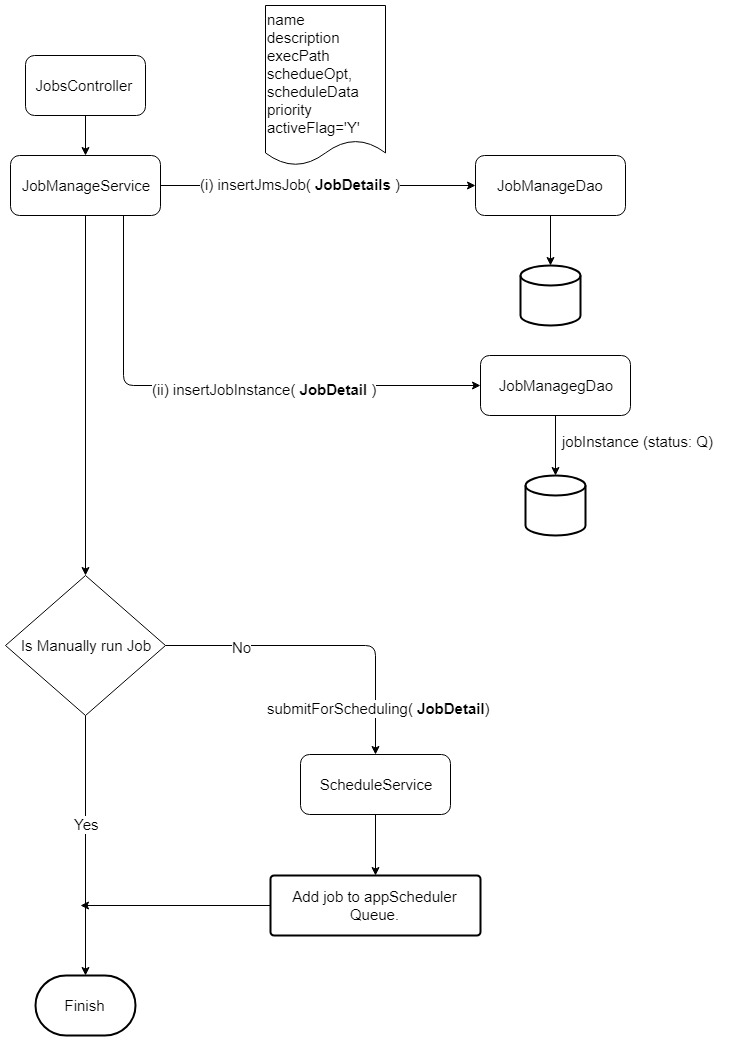
**Use Case Diagram**



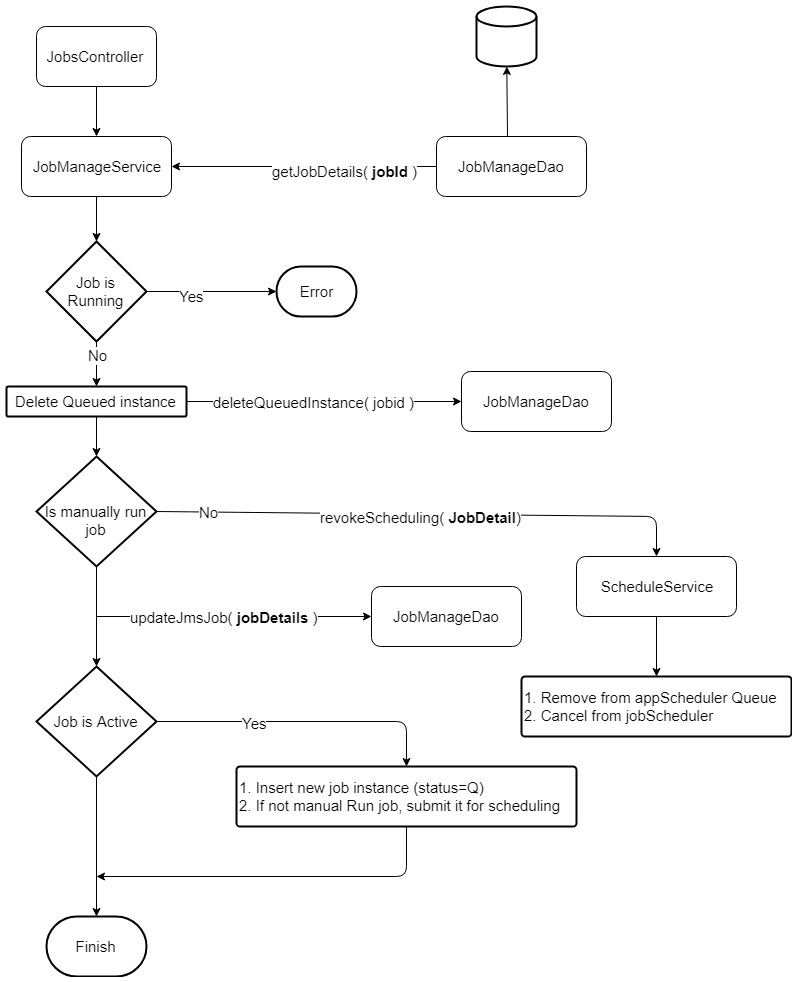
**Job entry details**



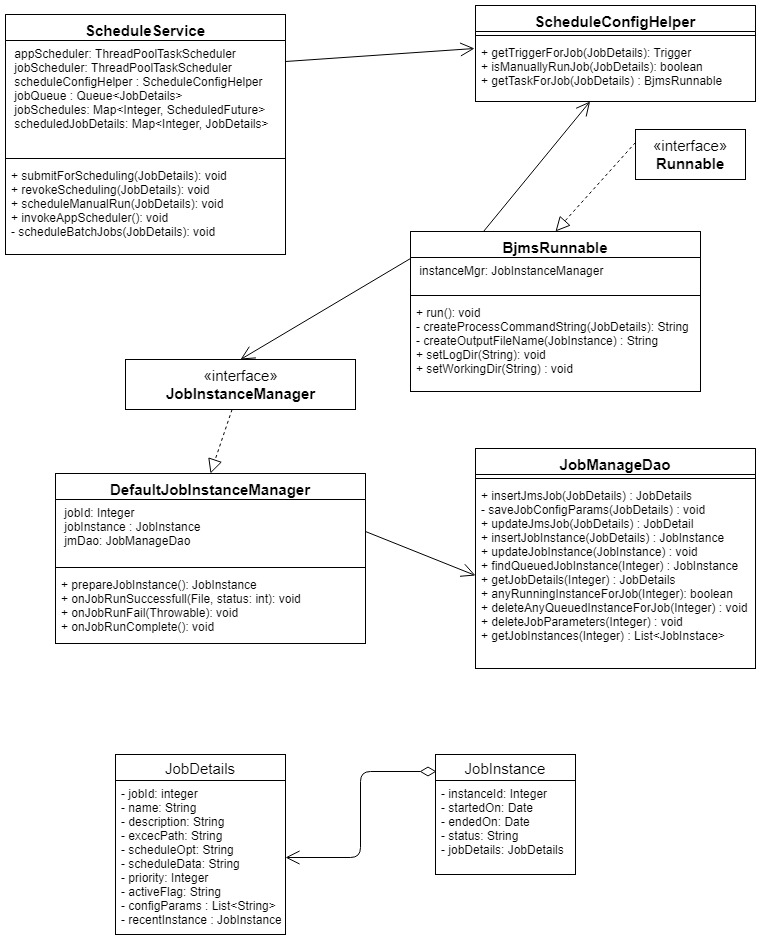
**Save Job scenario**



**Update Job scenario**



**Classes**



**API’s**

1. Save Job

POST /job/save

{

"name" : "Linear Regression Job",

"description" : "File Copy job",

"execPath" : "./src/test/resources/BatchJob.jar",

"configParams" : [

"data=4:65,2.5:49,1:20,7:85,10:91,7.5:96,5.5:68,3:58",

"maxWait=30",

"x=6"

],

"priority" : 1,

"scheduleOpt" : "C",

"scheduleData" : "0 0 1/3 ? \* ?"

}

1. Get all Jobs

GET /job/

1. Update a job

POST /job/update

{

"jobId": 1,

"name": "Email send Job",

"description": "Email send Job",

"execPath": "./src/test/resources/BatchJob.jar",

"scheduleOpt": "P",

"scheduleData": "4:4",

"priority": 1,

"activeFlag": "Y",

"configParams": [

"emailIds=a@b.de,c@d.de,e@f.de,g@h.de,i@j.de,k@l.de,sreejithvs999@gmail.com",

"maxWait=30"

]

}

1. Get instances of a job

GET /job/<instanceId>

1. Run manually runnable job

POST /job/run

{

"jobId": 33

}

**Database Tables**

1. Jms\_job

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | Job\_id | Integer primary key auto increment |  |
| 2 | name | Varchar(255) not null |  |
| 3 | job\_desc | Varchar(1000) |  |
| 4 | jar\_path | Varchar(255) |  |
| 5 | schedule\_opt | Char(1) | C – Cron expression, P – Periodic, M – manual |
| 6 | schedule\_data | Varchar(100) | Cron expression or “Init delay:Interval” |
| 7 | priority | Integer |  |
| 8 | Active\_flag | Char(1) | Y or N |
| 9 | Created\_date | Datetime | Audit purpose |
| 10 | Upated\_date | Datetime | Audit purpose |

1. Jms\_job\_params

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | Job\_id | Integer not null foreign key |  |
| 2 | Param | Varchar(255) not null |  |

1. Jms\_job\_instance

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | Instance\_id | Integer primary key auto increment |  |
| 2 | Job\_id | Integer not null foreign key |  |
| 3 | Started\_on | Timestamp | When job started |
| 4 | Ended\_on | Timestamp | When job end |
| 5 | Status | Char(1) | Q, R, S, F |
| 6 | Created\_on | Datetime | Audit purpose |

**Test Batch Job**

The test batch job is a simple jar file which mimic jobs like copying files, sending emails, complex calculations etc. The test batch accept parameters and run based on their values. The jar is in /src/test/resources/BatchJob.jar.

**Improvement Areas:**

1. Priority based scheduling feature is not considered.
2. Validation of input details, verifying executable jar, user authentication and other security validations.
3. Upload of executable jar.
4. More api’s for job monitoring and managing job outputs.