Q2. Design for Workflow Feature

Taking the code and design presented in <https://github.com/Navi-nk/grab-coding-exercise> as base, following is the proposed design for providing workflow feature.

**High Level Design**

In first version of the tool, the components to manage Job-Group info and the execution of these jobs were present in the same application. In the new design as shown below, the Webservice and Executor components would be divided into two separate applications so that they can be scaled separately. Along with the API layer, a scheduler component would be introduced to provide scheduling of Job Groups.

Diagram

Description automatically generated

**Changes to Entity Model.**

The entity relation model for the tables which store the Job and Group info as presented in Q1 had two main entities – **Group** table and **Job** table. In the new approach we will introduce one more table – **GroupSet** which would sit top of this hierarchy as shown below.

Diagram

Description automatically generated

Similarly, the store the Job execution Status we would introduce another table – **GroupSetInstance** which would be placed at the top of the hierarchy as shown below.

Diagram

Description automatically generated

**Changes to API Layer**

In current design we have APIs to save Group and Job info – Job Group Controller. We will introduce one more API to provide workflow info on the Groups. Following is the proposed JSON input message model.

**{**

**"groupSetId": "test\_group\_set",**

**“scheduleInfo” : {…}**

**"groups": [**

**{**

**"groupId": "group1",**

**"runId": 1,**

**"continueOnFail" : false**

**},{**

**"groupId": " group2",**

**"runId": 2,**

**"continueOnFail" : true**

**},{**

**"groupId": " group3",**

**"runId": 3,**

**"continueOnFail" : false**

**}**

**]**

**}**

This new API will take the above payload and create a groupSet workflow which would have following base fields

* scheduleInfo – to define any scheduling info for the workflow
* groups – collection of groups that where stored in the APIs created in first version.
  + runId – Running sequence which decides the order of execution
  + continueOnFail – flag which decides to stop or continue execution on failure of a group execution.

For submitting execution of this created GroupSet, another API similar to one provided in version 1 would be given. This API will take **groupSetId** as input parameter.

The execution logic would change as shown in the below picture.

Diagram

Description automatically generated

The **JobGroupInstanceExecutor** introduced in the first version would be responsible for execution of individual groups, we will one more executor **GroupSetInstanceExecutor** would now invoke **JobGroupInstanceExecutor** sequentially based on the GroupSet workflow info. This executor would be responsible for handling the result of each of the groups and also handle its failures.

An example output of JobSetExecution status will look as follows

**{**

**"groupSetId": "test\_group",**

**"groupInstanceId": "test\_group-20210207-072254",**

**"status": "COMPLETED",**

**"groupInstances": [**

**{**

**"groupId": "test\_group1",**

**"groupInstanceId": "test\_group1-20210207-072254",**

**"status": "COMPLETED",**

**"startTime": "2021-02-07T19:22:54.505",**

**"endTime": "2021-02-07T19:22:54.726"**

**},**

**{**

**"groupId": "test\_group2",**

**"groupInstanceId": "test\_group2-20210207-081254",**

**"status": "COMPLETED",**

**"startTime": "2021-02-07T19:23:55.105",**

**"endTime": "2021-02-07T19:24:54.516"**

**}**

**],**

**"startTime": "2021-02-07T19:22:54.505",**

**"endTime": "2021-02-07T19:24:54.516"**

**}**

**PseudoCode**

The API signature to create the workflow

“/jobstore/groupset/save”

**@PostMapping(value = "/save")**

**public JobGroupSetRequest createJobGroup(@RequestBody JobGroupSetRequest request) throws Exception {**

**return jobGroupSetService.createJobGroupSet(request);**

**}**

The API signature to trigger workflow.

“/groupset/execute/{groupSetId}”

**@PostMapping(value = "/execute/{groupSetId}")**

**public JobGroupInstanceMessage executeJobGroup(@PathVariable("groupId") String groupId) {**

**return jobGroupInstanceService.executeJobGroup(groupId);**

**}**

The main executor block will look like following:

**public JobGroupSetInstanceMessage executeJobGroup(String groupId) {**

**try {**

**return new JobGroupSetInstanceExecutor(executorService).execute(request);**

**} catch (Exception e) { }**

**}**

**JobGroupInstanceExecutor** class will changed to implement Callable interface so that it can be called in a separate thread from **JobGroupSetInstanceExecutor.** The logic from to maintain GroupSet workflow will have more or less same flow as that seen in **JobGroupInstanceExecutor**