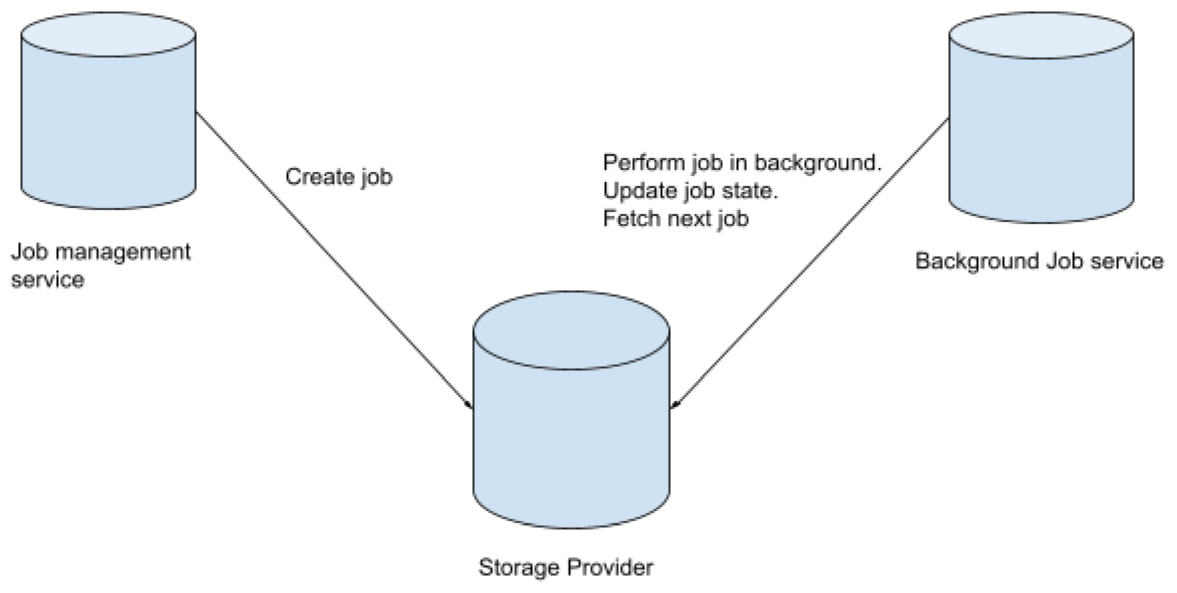

Job management service architecture overview.

When adding a job, the job management service saves it into a storage provider.
background job service polls the storage provider for new jobs and processes them.
When finished, it updates the state in the storage provider and fetches the next job to perform.

Job

we have the Job entity - it contains the name, class name, state, priority, cron schedule. A Job is a unit of work that should be performed outside of the current execution context, e.g. in a background thread, other process, or even on a different server. The class name is a java class which implements Runnable and job's logic. In the case cron schedule is provided, a job will be executed according to a schedule otherwise it will be executed immediately when free resources are available.

Storage Provider

A StorageProvider is a place where the Job management system keeps all the information related to background job processing. The StorageProvider is abstracted well enough to be implemented for RDBMS and NoSQL solutions. The current implementation supports any jpa compatible sql db.

BackgroundJobService

The BackgroundJobService class processes background jobs by querying the StorageProvider. It fetches jobs with Queue state, updates state to Running, performs them in the background and as a result updates state to Success/Failed.
Background jobs will be rescheduled automatically after restart.

JobService
The JobService provides a set of methods for adding/canceling jobs and starting/stopping the entire job system.

Configuration properties:
Job-service.job-pool-size - number of workers in BackgroundJobService
Job-service.polling-interval - time interval in ms between 2 poll attempts of BackgroundJobService

Main technologies:
Java 1.8+
Grade 7.0
MySQL/H2
Spring Boot
Spring Data
JUnit5&Awaitility

Possible improvements:

1) Scalability. Scalability is not listed in the requirements of the system. So it was not added to simplify architecture/implementation. However, It can be implemented without large redesign of the system as the current system architecture is scalability ready. Just BackgroundJobService needs a slight update to support parallel execution.
2) Dashboard with health monitoring , stats, configurations.
3) Adding job execution context in order to support custom metadata, dashboard logging and progress bar.
4) Retry policy for failed jobs.
5) Different implementation of StorageProvider, For Instance Apache kafka, Apache Redis or AWS DynamoDb.
6) Background service to monitor and cancel orphan jobs (too long running and cannot be completed).
7) Adding different types of tests, including load testing, full test coverage.
8) Adding support a load factor to keep some workers always available for scheduled tasks.

Assumptions:

Job should be implemented as a re-entrant and designed for cancellation. Re-entrancy means that a job can be interrupted in the middle of its execution and then be safely called again.