|  |
| --- |
|  |
| **Dynamic Scheduler Documentation** |
| **For Compumatrice** |

|  |
| --- |
| Author: Hiraji  2-24-2020 |

Table of Contents

[Introduction 2](#_Toc34305828)

[Scope of this Document 2](#_Toc34305829)

[Summary 2](#_Toc34305830)

[Definitions, Acronyms and Abbreviations 2](#_Toc34305831)

[Scheduler Integration with NestJS application framework 3](#_Toc34305832)

[Scheduler 3](#_Toc34305833)

[Entity Diagram 3](#_Toc34305834)

[Class Diagram 4](#_Toc34305835)

[Scheduler Interface/Admin UI 5](#_Toc34305836)

[Admin UI 5](#_Toc34305837)

[Create Jobs 6](#_Toc34305838)

[Stop Job 6](#_Toc34305839)

[Stop Job Work flow 6](#_Toc34305840)

[Delete Job 7](#_Toc34305841)

[Delete Job Workflow 7](#_Toc34305842)

[Ad Hoc/on Demand Job Execution 8](#_Toc34305843)

[Adhoc Job Execution Workflow 9](#_Toc34305844)

[Create Jobs on App start up 9](#_Toc34305845)

[Failed Jobs Rescheduling 10](#_Toc34305846)

# Introduction

## Scope of this Document

The document gives a clear understanding of the Dynamic scheduler and its integration in nest js application. The Dynamic scheduler is designed to handle various type of jobs such as intervals, timeouts and ad-hoc jobs. There is an Admin UI to configure and manage Jobs. In addition, there are features to rescheduling a Job and report on job states including failed state.

## Summary

The document covers the following features:

1. Scheduler Integration with NestJS application framework
2. Scheduler
   1. Dynamic intervals jobs.
   2. Dynamic timeouts jobs.
   3. Ad-hoc scheduler.
   4. Failed Job Rescheduling

## Definitions, Acronyms and Abbreviations

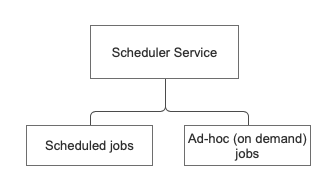
|  |  |
| --- | --- |
| Acronyms | Abbreviations |
|  |  |
|  |  |

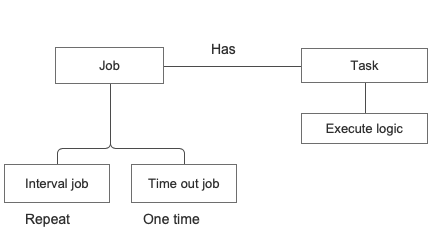
# Scheduler Integration with NestJS application framework

Please refer <https://docs.nestjs.com/techniques/task-scheduling> to integrate scheduler in nest js app

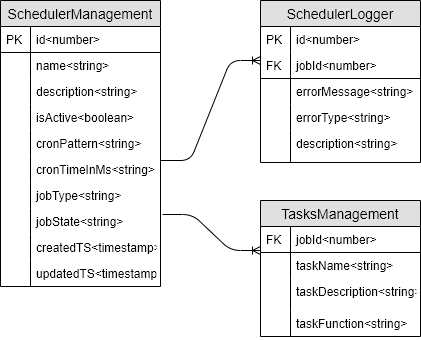
# Scheduler

Scheduling allows you to schedule arbitrary code (methods/functions) to execute at a fixed date/time, at recurring intervals, or once after a specified interval, This will also allow you to schedule a specific job on demand(ad-hoc)





## Entity Diagram



**Scheduler Management Table Detail**

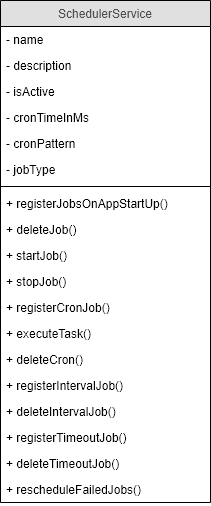
|  |  |  |
| --- | --- | --- |
| Field Name | Description | Example |
| id | Primary key - AI | 1 |
| name | Job name | Appointment Fetch |
| description | Purpose of the job | Monday to Friday appt fetch at 11:30am |
| isActive | This is used for active/deactive job | 1/0 |
| cronPattern | Cron job pattern, this will be used for only cron job | 0 30 11 \* \* 1-5 |
| cronTimeInMs | Cron time in ms, this will applicable only for interval and timeout jobs | 1000 |
| jobType | Type of the job | Interval/timeout/cronjob |
| jobState | State of the job | Failed/success |
| createdTS | Job created timestamp | 2020-02-27T18:16:00.000+05:30 |
| UpdatedTS | Job Updated timestamp | 2020-02-27T19:16:00.000+05:30 |

**Scheduler Logger**

|  |  |  |
| --- | --- | --- |
| Field Name | Description | Example |
| id | Primary key - AI | 1 |
| jobId | Foreign Key | 1 |
| errorType | Error type | Failed/success |
| errorMessage | Error message | Failed to execute job |
| description | Detail error description |  |

## Class Diagram

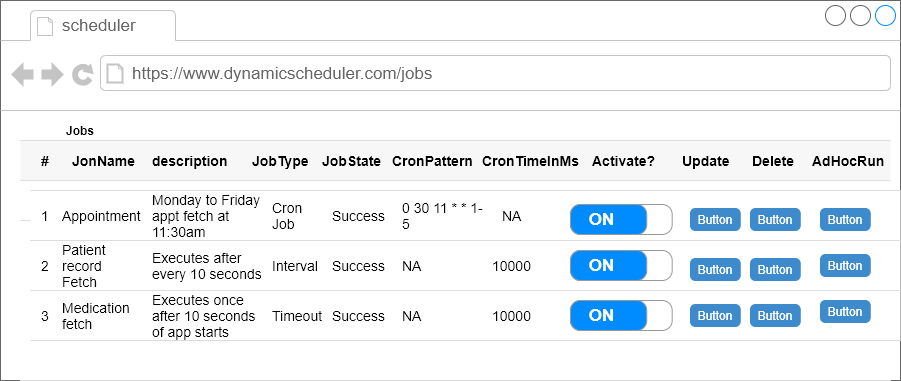
For scheduling operations will use single class “**SchedulerService**” and reuse methods for specific operations e.g. We can reuse **createIntervalJob()** or **createTimeoutJob()** method for rescheduling the failed jobs



# Scheduler Interface/Admin UI

This will allow us to manage scheduling tasks dynamically through the interface as shown in below diagram, no need of manually update into db and restarting node service, this will provide facility to activate./deactivate running jobs, Update existing running jobs, delete jobs which are no longer required and run a job on demand(ad-hoc)

## Admin UI



## Create Jobs

We have to manually create jobs in **SchedulerManagement**  table to populate and operate jobs from admin UI

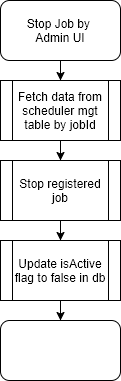
We are storing job details in db because we need to restart jobs when service is down and automatically restarted by PM2 OR when service is restarted by operational team for some reasons

If you create a job in **SchedulerManagement** tablewith isActivate=1 and restarted the node service then that job automatically will get registered on app start up

## Stop Job

This will stop the existing running job and update the jobState as “Stopped” in **SchedulerManagement** table through UI interface as shown in above Admin UI interface and workflow diagram

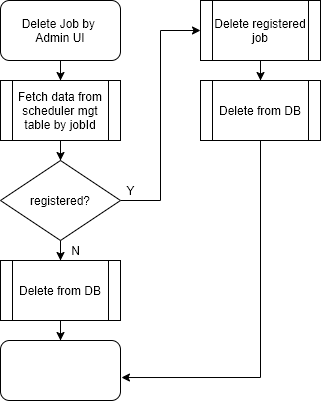
### Stop Job Work flow



## Delete Job

This will delete existing running job from server and also deletes job record from **SchedulerManagement** table by jobId as shown in above Admin UI interface and workflow diagram

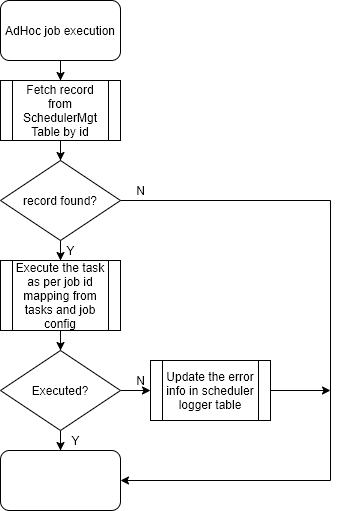
### Delete Job Workflow



## Ad Hoc/on Demand Job Execution

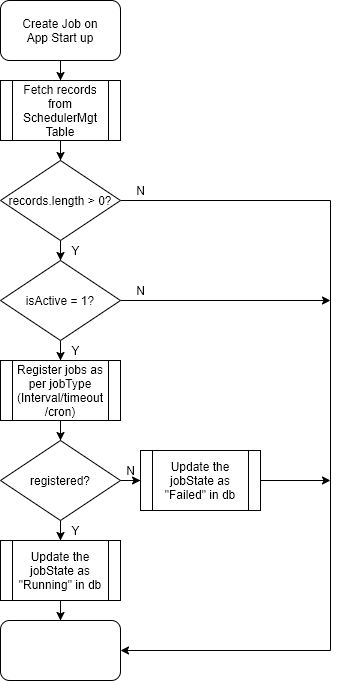
We can run existing running jobs on demand from admin interface, without waiting for registered job to execute, OR we can directly execute any job on demand, To run the job we just need to click on adhoc run button from interface as shown in above admin UI interface

### Adhoc Job Execution Workflow



# Create Jobs on App start up

This will fetch jobs configuration from **SchedulerManagement** table on app start up and register the jobs in system memory as shown in below diagram



# Failed Jobs Rescheduling

Will register separate interval job on app start up which will execute at every 10 seconds and fetch failed jobs configuration data from database and reschedule them as shown in below diagram

