# **Job Management System**

Design Document

**Name**: Manas Pratim Acharyya

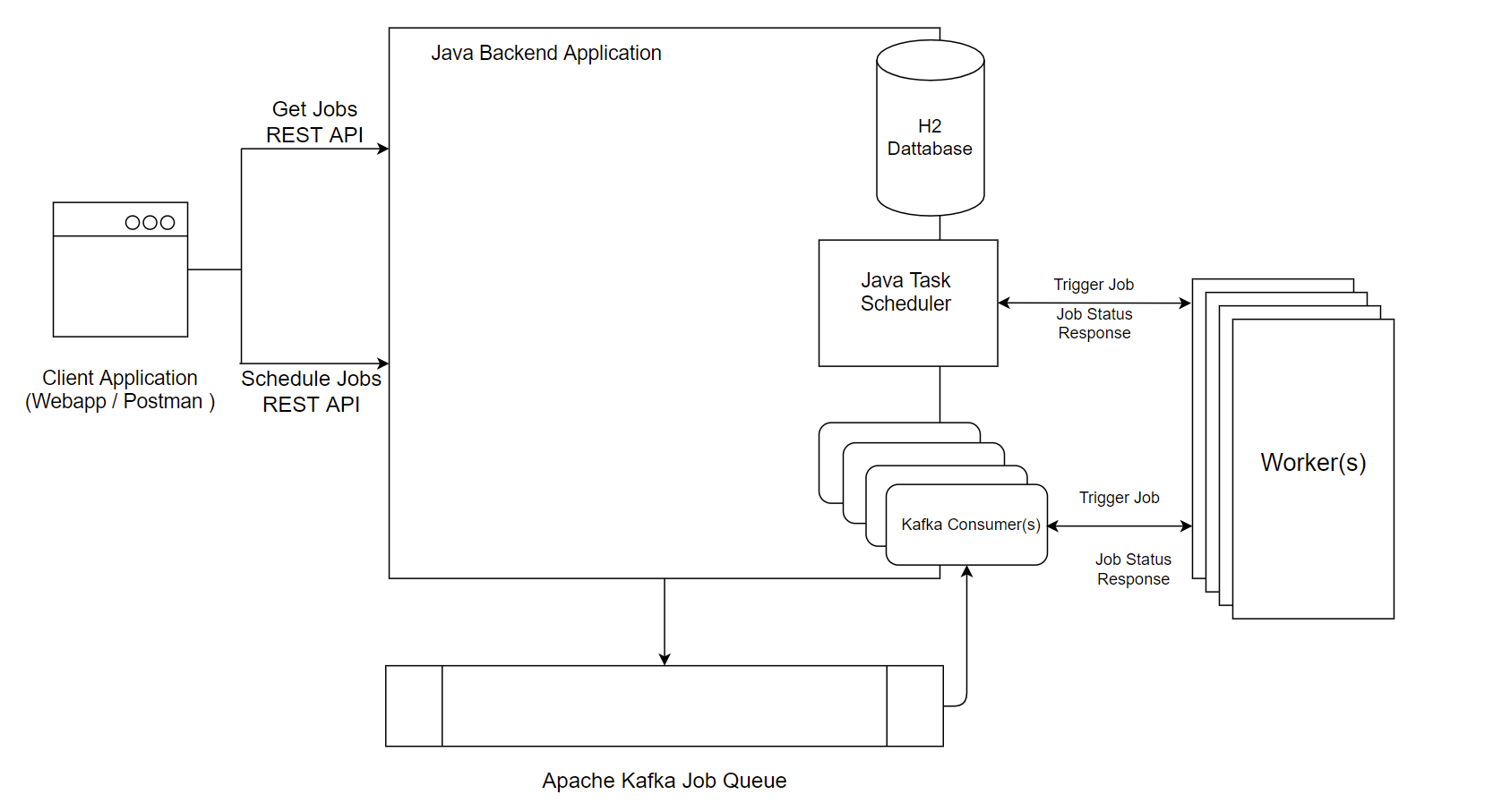
**Date**: 07st June 2021

**Feature Requirements**

A Job Management Service that can handle scheduling and execution of multiple types of Jobs.

System must be:

1. **Flexible**: System is not aware of the Jobs. New Jobs should be supported without re-developing the System.
2. **Reliable**: Each job should either complete successfully or leave the system unaffected, in case of failure.
3. **Internal Consistent**: Jobs can have four states: QUEDED, RUNNING, SUCCESS, FAILED. Following execution of Job, it should be left in appropriate state.
4. **Scheduling**: System should allow jobs to be performed immediately or according to a schedule.
5. **Priority**: Jobs should be executed based on its priority relative to other jobs.

**High Level Design  
  
**

* The Job Management System can accept an immediate / scheduled Job Request via exposed REST APIs.
* Job Scheduler System is independent of the actual Job. Jobs are assumed to be different microservices, that can be triggered via REST API calls, with appropriate Parameters.
* A New Job Payload will have following attributes:
  + Id
  + Name
  + Request URL ( REST URI to invoke the Job )
  + Request Parameters
  + Execution Type ( SCHEDULED | IMMEDIATE )
  + Priority ( HIGH | LOW )
  + Status ( NEW | QUEUED | RUNNING | SUCCESS | FAILED )
  + Schedule Interval ( only applicable in case of a scheduled job )
* Upon Receiving the Request For a New Job, Job Details is stored in a Database.
  + If JOB TYPE is IMMEDIATE
    - JOB ID is pushed into a Message Queue
    - Message Consumers consume message from the Message Queue and Invoke the Job Request ( as discussed earlier, each Job is considered a REST API Call with appropriate parameters )
    - Upon Successful completion / Failure of Job, Job Status is updated accordingly in Database.
    - For Long Running Jobs (Batch Jobs), it is assumed that Job itself is responsible for notifying the Job Scheduler System about it’s successful completion or failure.
  + If JOB TYPE is SCHEDULED
    - A Scheduled Job is created that runs at specified interval.

**Implementation Details:**

* Application is written in **Java**, using **Spring Boot Framework.** It exposes following **REST API’s** to add and retrieve Jobs.
  + GET: /scheduler : Get All Jobs
  + GET: /scheduler/{JOB\_ID } : Get Specific Job by Job Id
  + POST: /scheduler : Add a New Job
* All Data is persisted in **H2 Database**.
* **Apache Kafka** is used as a Message Queue, to store Job Requests.
* Java **ThreadPoolTaskScheduler** is used to Schedule Recurring Jobs at specified intervals.
* Job **Priority** is achieved with Apache Kafka by implementing **Bucket Priority Pattern**.  
   Two separate Kafka Topics are maintained for High Priority and Low Priority Jobs.   
  Multiple Concurrent Consumers consume from High Priority Queue as opposed to a single consumer for Low Priority Queue, thus processing High Priority Jobs faster than Lower Priority Jobs.
* Two separate Queues are maintained for Short Lived vs Long Running Jobs. This allows Short Lived Jobs to be processed quickly without being dependent on another Long Running Job to complete.

**Test the Application:**

To test the application, we have the following:

* **Job Scheduler System** itself.
* A **Sample Job Application**, that responds with Success or Failure after specified time.  
  Exposes following 2 APIs:
  + GET: job/short-success/{message} : Prints Message and Responds 200 OK
  + GET: job/short-failure/{message} : Responds 400 OK Bad Request
* A simple **Angular App** that lets us Create New Jobs and See them Working in Action.

Prerequisites:

* Docker needs to be installed in your system, to test the Application.
* In case you do not have Docker installed, you may individually build and run the three application.

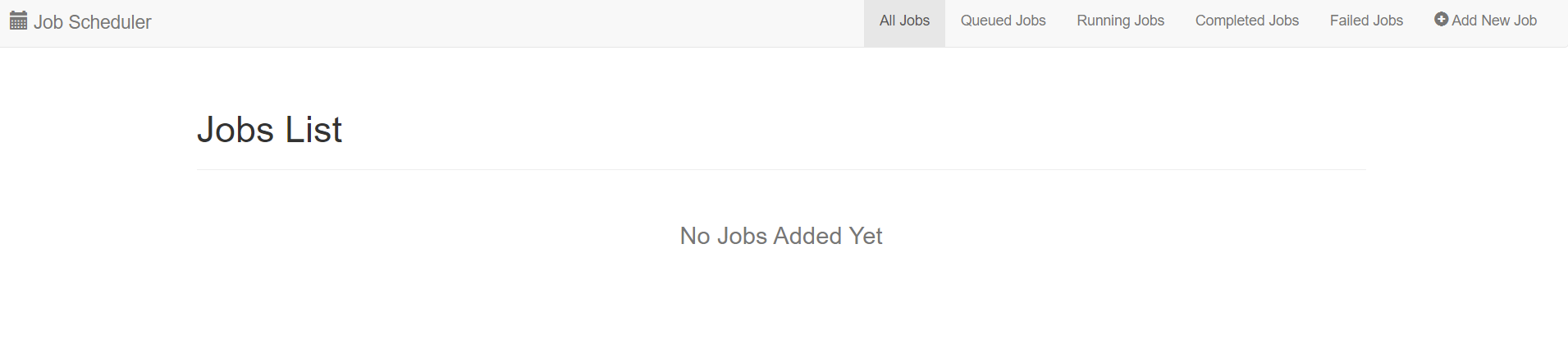
Steps to Follow:

1. Run All Three Applications by running the following command from within the Project Directory.

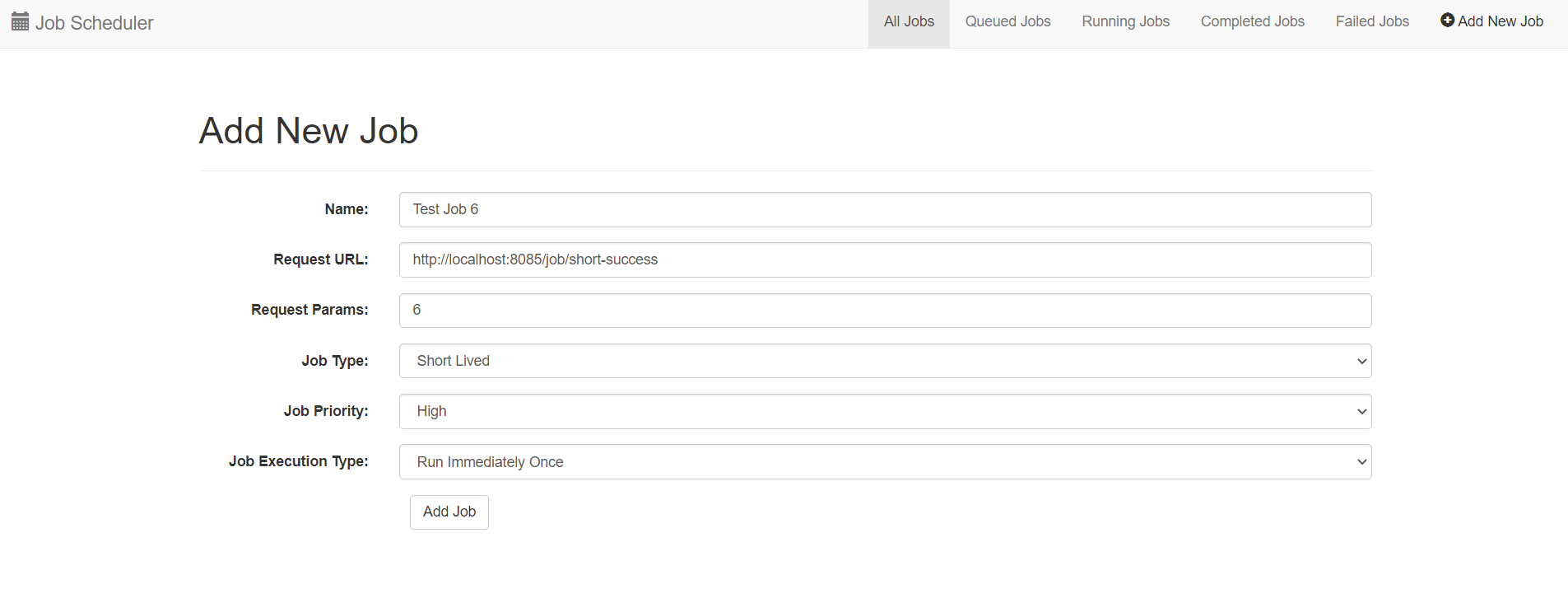
docker-compose up -d

This will spin up all required services. Docker needs to be installed in your system.   
*( Disclaimer: This step will take long time to complete, as it downloads JAVA Runtime, Node Runtime and Maven Build Tool Images)*

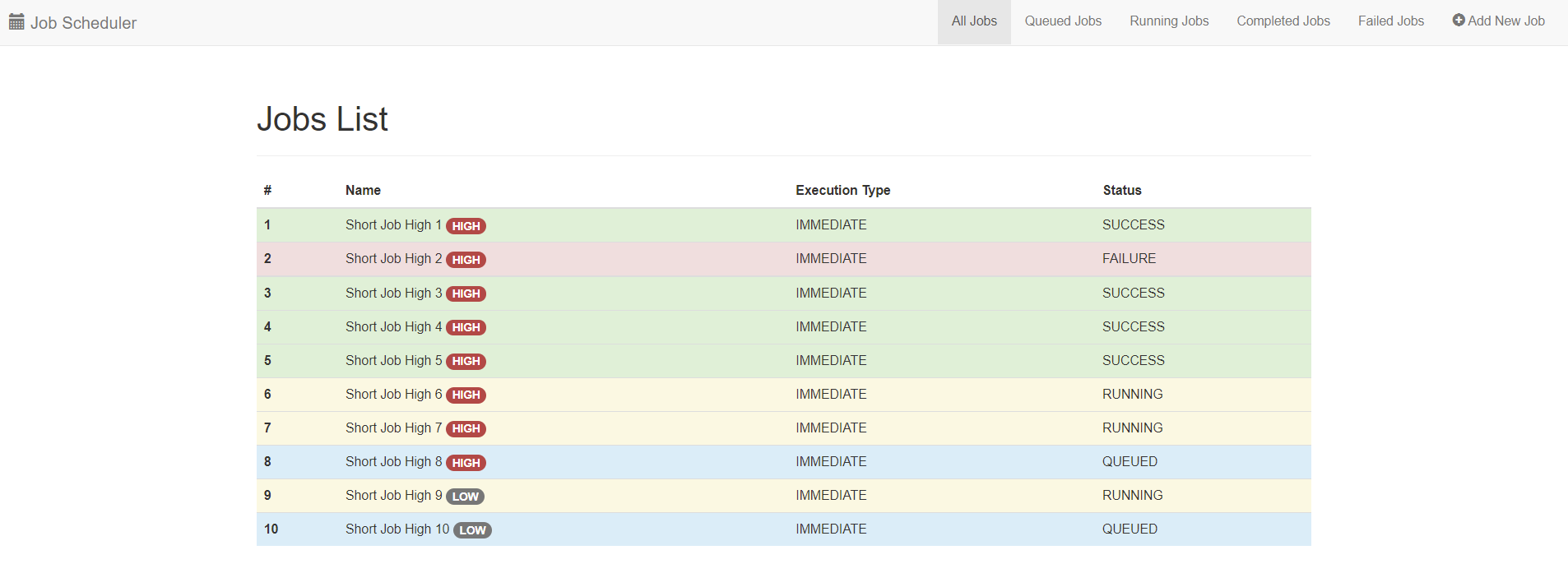
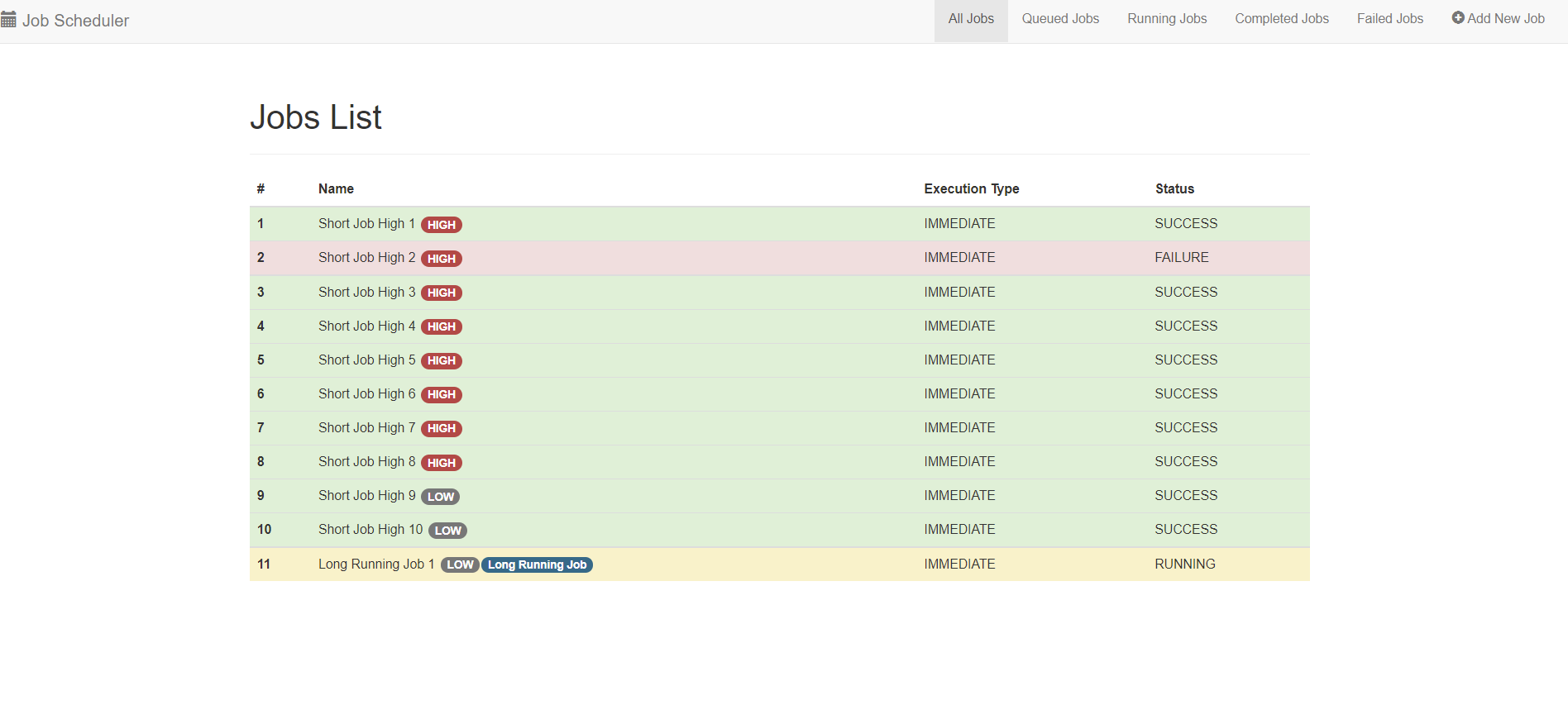
1. In a browser open <http://localhost:4200>. This is where the angular app is running. You should see a screen like this



1. Click on **Add A Job** at top right corner. Fill in details of the Job as shown below:



Request URL: <http://sample-job:8085/job/short-success> points to Sample Job Application developed for testing the Job Scheduler System

1. Click on Add Job. Once Job is submitted you should see the Job List with your newly added Job. Add a few more jobs. You should see something like this after adding a few Jobs.   
     
   
2. Give it some time and Refresh to see updated Status of Jobs.   
   

The Jobs List Dashboard, provides information on Job Id, Job Name, Job Execution Type, Job Priority and Job Status.

**Possible Enhancements**:

1. Modifying a Queued Job.
2. Cancelling a Queued/Running Job.

**Effort Breakdown:** Total time Spent on this Case Study is 30 Hours.   
Find my commit history here: [Github Repo](https://github.com/manasacharyya25/JobSchedulerSystem/commits/main)

|  |  |
| --- | --- |
| Task Description | Effort in Hours |
| Initial Investigation on The Case Study | **6 Hours** |
| Initial Design Draft | **2 Hours** |
| Implementing the Java WebApp Skeleton with REST APIs, H2 DB and Kafka Producer and Consumer | **2 Hours** |
| Implement Job Scheduler with Java ThreadPoolTaskScheduler | **2 Hours** |
| Implement Priority Queue with Kafka | **3 Hours** |
| Unit Testing | **3 Hours** |
| Docker Containerization | **3 Hours** |
| Frontend Application | **3 Hours** |
| Miscellaneous Testing + Design Documentation | **6 Hours** |