## Introduction

The objective of this project is to develop a scraping pipeline using Python and [Scrapy](https://docs.scrapy.org/en/latest/), which extracts data from a JSON file, stores it in PostgreSQL and (optional) MongoDB databases, and uses Redis for caching. The entire application should be containerized using Docker and orchestrated with Docker Compose.

* PostgreSQL and Redis are required for this project.
* MongoDB is optional (bonus points).

**Video Requirement: Include a short video (2 minutes) demonstrating your work. *Submissions without the video won't be considered.***

## Part 1 - Setting Up Environment

Have Docker and Docker Compose installed on your system. Create a project folder with the following structure:

├── docker-compose.yaml

├── dockerfile

├── infra

│ ├── postgresql\_connector.py

│ └── redis\_connector.py

├── jobs\_project

│ ├── jobs\_project

│ │ ├── \_\_init\_\_.py

│ │ ├── items.py

│ │ ├── middlewares.py

│ │ ├── pipelines.py

│ │ ├── settings.py

│ │ └── spiders

│ │ ├── \_\_init\_\_.py

│ │ └── json\_spider.py

│ └── scrapy.cfg

├── query.py

├── README.md

└── requirements.txt



* **Compose File**: Create a docker-compose.yaml file to define services: Scrapy, PostgreSQL, MongoDB, Redis.
  + **Scrapy Service**: Either use a Python / Conda image or mount an Ubuntu / Debian image.
  + **PostgreSQL Service**: Use PostgreSQL image, set necessary environment variables.
  + **Redis Service:** Use redis or redis-stack-server image.
  + **MongoDB Service (Optional)**: Use MongoDB image.
* **Dockerfile**: Create a dockerfile for the Scrapy service.
* **Infra**: In addition to setting up the environment with Docker and Docker Compose, you will create an infra module. This module will contain helper files for managing connections to PostgreSQL and Redis to separate infrastructure management from business logic. The folder structure should look like this:

├── infra

│ ├── postgresql\_connector.py

│ └── redis\_connector.py

* **Requirements**: include a requirements.txt file in your project that include Scrapy, psycopg2 (or an equivalent PostgreSQL adapter), PyMongo (optional), and any other libraries you use in your project. Specify the versions of these packages.
* **Documentation**: Document the setup process, how to run the project, and any configurations needed. Provide a README.md file explaining:
  + Setup instructions.
  + A brief description of the pipeline process.
* **Submission**:
  + The application should work as intended.
  + Clean, well-organized, and documented code.
  + Proper handling of common issues in database operations.
  + Submit the entire project folder as a compressed file (ZIP or TAR) or provide a link to a Git repository. Only share the scripts, do not share large files or any data.

## Part 2 - Raw Data

The raw JSON files can be found at [s01.json](https://drive.google.com/file/d/10Pf51kHvosCEKdNCSkIbDW3-05mHp5_A/view?usp=drive_link) and [s02.json](https://drive.google.com/file/d/16EQokcYHNxnhPTE-gXD8vzuFlmVHCffF/view?usp=drive_link). Within these files, the most important information we are interested in is nested under the jobs key. Iterate over each entry in the jobs array and use the item pipeline you will work on to store them in the database.

## Part 3 - Scrapy Pipeline

* **Initialize Scrapy Project**: Initialize a Scrapy project, name it jobs\_project.
* **Create a Spider**: Write a spider in json\_spider.py to scrape data from this html template. Use JSON format to retrieve the items.

import json

import scrapy

class Jobpider(scrapy.Spider):

name = 'job\_spider'

custom\_settings = {

'ITEM\_PIPELINES': {

'local\_spider.pipelines.yourPipeline': 300,

},

}

def \_\_init\_\_(self, \*\*kwargs):

# your code here

pass

def start\_requests(self):

# your code here

# make sure you can send a request locally at the file

# if you can't get this to work, do not waste too much time here

# instead load the json file inside parse\_page

yield scrapy.Request(

url='file:////home/PATH\_TO\_JSON',

callback=self.parse\_page,

)

def parse\_page(self, response):

# your code here

# load json files using response.text

# loop over data

# return items

pass



* **Define Items**: In items.py, define the structure of the items you will scrape.
  + Make sure that date fields are in the appropriate date format.
* **Pipelines**: Modify pipelines.py and within json\_spider.py activate ITEM\_PIPELINES component(s) to handle:
  + **Database-Raw**: Configure the pipeline to store all scraped data into PostgreSQL, store the items in raw\_table.
  + **Cache**: Use Redis to cache items that have already been scraped to avoid duplicate processing.
  + **MongoDB (Optional Bonus)**: Store items in raw\_collection.

## Part 4 - Database Query

* Include a query.py script with Database class(es) designed to establish connections to both PostgreSQL and MongoDB (optional) servers. The goal of this script is to retrieve all processed data and organize it into a CSV file format.