Legalist

Backend Engineer Interview Challenge

This is an open-ended problem. We don't expect you at all to finish all these tasks. You can choose to do whatever you feel most comfortable with first, just make sure to manage your time properly and do not spend a lot of time on 1 task. It's on you to pick the tasks you know the most to get the most done and show us your technical prowess.

We try to keep the instructions as open/flexible as possible to see what would be your approach to tackling such a problem. In case you need to make a decision and the instructions are not clear feel free to make assumptions as long as you write them down and provide rationale.

Required

We will be scraping the quotes endpoint, as much as you can from http://toscrape.com/. Create a new project and set up a small scraping environment using a microservices architecture running on top of docker (extra points for orchestration using docker-compose or even more using minikube)

We get a lot of competitive applications, if you want your application to have a good chance of approval, it is highly advised to have the following as a bare minimum:

- Having a running scrapy spiders with proper redis de-duplication
- Postgres as a datastore for the items scraped
- Json exports for the items scraped
- At least one medium and one hard spiders from the below list
- Proper docker deployments

Using docker can you setup:

1) Scrapy engine

Default Microdata and pagination (easy)
This is pretty and straightforward, should get you warmed up pretty quickly
Scroll infinite scrolling pagination (medium)
This one is interesting, we're interested in the way you tackle pagination, try to explain why did you use that strategy for handling pagination
JavaScript JavaScript generated content (hard)

Legalist

This one would require a bit more expertise in running containers. Great addition if you do. You'll have to set up a splash instance to render the js and get the data on the page. If you decide to go down that path, checkout the section "Javascript Rendering Engine" below:

Tableful a table based messed-up layout (medium)
Be careful of this one, don't waste too much time trying to make it work.
Random a single random quote (medium)
Login login with CSRF token (any user/passwd works) (medium)
Interesting to see how you would handle pagination. There's two pages with login, this one and the viewstate below. This one is slightly easier, we would recommend that you choose only one.
ViewState an AJAX based filter form with ViewStates (hard)
Another page with login, slightly more challenging

Each one is characterized by a certain challenge. You don't have to finish them all. Start by the default one just to make sure that all your environment and pipeline are working fine. After that pick the one that you're most comfortable with.

Some rules

- Reuse code as much as possible (DRY principle)
- Have a separate spider for each type
- Export the data to json files and to postgres.
- Make sure we don't hit the website too hard. (implement/configure rate limiting)
- Run spiders as cron job daily at 11am and 11pm
- Implement decent logging mechanism to be able to easily debug
- Implement proxying(optional see below)
- We want to scrape the same url as much as possible if possible only once
- We want to be able to pause/resume the scraping job from where we left in case of failure (view notes in redis section).

2) Postgres database

This will be used to save the scraped data. You will be responsible for creating an appropriate schema. Please try to collect as much metadata as possible (created_at, updated_at, soft delete implementation, index choice). It would be preferable if you use the django orm but you don't have to. Use whatever db interaction you're most comfortable with. Either sqlalchemy, storm or even raw SQL if you're a hardcore. If you decide to use django take a look at how to use your django model as scrapy items.

Legalist

3) Redis

We want to be able to pause/resume the scraping job. Take a look on how to do that using scrapy and redis. Also redis could be used as a caching layer. You would need to implement the scrapy dupefilter using redis to avoid hitting certain requests multiple times. If you've worked with scrapy before you should know what we're talking about by now. Friendly note, if you already ran the spider and it stopped fetching new urls, make sure to clean the redis cache fitler or else it would be filtering out all the visited websites.

Extra

You would definitely score more points and potentially speed up your application if you do the following. These are extra features that would allow you to go the extra mile to show off your expertise. This can give your application an edge if you're shortlisted against a small list of candidates.

Scraped Data API (easy)

Create an api endpoint that has a certain port exposed to view the scraped data (flask, django drf, fast api, something else?). This api should be scalable and fast. Implement basic test cases if you have the time to check that the api is working. What kind of monitoring can you do on the api.

Javascript Rendering Engine (medium)*(if you decide to scrape the javascript endpoint)*

You'll need to set up a javascript rendering engine that would help you scrape websites that require some javascript validation or security. We would suggest taking a look at splash. This engine would be used in the Javascript parsing step.

Proxy Service (medium)

As you know some websites might be IP sensible and could block multiple requests coming from the same ip. Can you implement a local proxy server (squid, haproxy, tinyproxy) and have all your requests go through the proxy server to avoid detection?

Submission

It is mandatory to have a readme and clear documentation about the design decisions taken while working on this project. Failure to do so might cause your application to be

Legalist

ignored. Moreover, please indicate the overall amount of time spent on the project in that readme. Before uploading your submission, **please .zip all your files.** Once completed, please upload the entire folder to the provided Greenhouse Link in your email correspondence. If you have any questions, please contact parth@legalist.com