Test task description

Your task is to build a backend for a web scraper. It will be used by client applications to retrieve the following structured content:

- Facebook profiles
- Twitter profiles

Decorated with a popularity index calculated as:

- Facebook (friends)
- Twitter (followers)

Important notes and considerations

You are free to use whatever tools you feel are appropriate for the job at hand. Please keep in mind though, that your choice may impact how you fulfill the requirements below. If you do pick tools which do not enable this out of the box, we'll want you to specify how you'd configure them to make this possible. Also, like all things, we may want to add new content to the system, so make sure you keep components coupled as loosely as possible.

Requirements

For fetching data itself from both systems (Facebook and Twitter), you are free to use either APIs or scrape content right off profile pages. **We know you can't do magic, so assume you don't need to work around private profiles on either system**.

The task at hand, sorted by priority - complete as many points as you can!

1. A script which implements a function to extract the name, short description and popularity index (calculated as specified above) for Facebook and Twitter **public** profiles. Assume input parameter is the username.

2. Have the script you created before cache its results on any database or caching system you feel the most comfortable working with.

3. A REST interface to serve client requests for profiles which returns the name, short description if available, and the popularity index. This interface would invoque the script you created before and return its output. You are free to specify which route should be used and how requests should be formatted.

4. Update the REST interface to work asynchronously. This means it should return immediately if the information exists in cache/database, but create data scrape/fetch job if not. Use any message queue system you feel most comfortable with.

5. Have the script you created on the first step consume jobs from the message queue. Store the job's progress and status on the cache/database you created before.

Bonus points

- Create a package for the components using one of the common language specific packaging mechanisms available
- Create a package for the configuration of the components using the same mechanism selected above
- Create a deployment script for all components, including broker and database
- All things evolve, versioning of components and protocols/api is a must

Additional considerations

Important: While we'd like all submissions to implement the entire feature set, we realize the may be time constraints. If you cannot implement the entire feature set, please try to include some textual description of how you would to so. Also, be prepared to discuss it with us later on.

You are free to use whatever language and frameworks you're comfortable working with. Keep in mind, however, that you're expected to know what it is that is happening under the hood, so be prepared to discuss the implementation and lower level details once you submit your project.

Evaluation criteria

This test task helps us evaluate how you approach software development, defensive programming and how you interpret requirements.

We don't want to leave you in the dark when in comes to how your test task will be evaluated, so here are the evaluation criteria:

- Follows SOLID principles
- Minimal coupling/dependencies between components
- Writes clear, consistent, easy to read and easy to follow code
- Follows good and consistent naming conventions
- Writes unit tests
- Instruments the components

Assuming you pass the test task, please expect a brief discussion during the follow up interview along with some requirement changes and how you would implement them.

We hope you have fun implementing this simple project. We can't wait to see what you come up with.

References

- https://dev.twitter.com/docs
- https://developers.facebook.com/docs/
- http://oauth.net/2/