Website Scraping

**The Scenario**

Our task is to build a web scraping service that exposes the following API

 POST http://127.0.0.1:8080/api/v1/jobs

Request parameters:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| url | String | The url of the website to scrape |
| depth | Integer | The level of the scraping tree. Must be positive. |

Depth indicates how deep our scraping should go in terms of descendant nodes e.g.

* When depth == 1 the scraping service should simply return the given url and any url it finds in it.
* When depth == 2 the scraping should look in the main page and in every url that appears in the page i.e. return all urls found in the root and “children”.
* When depth == 3, the scraper should scrape the “grandchildren” of the main page as well – returning 4 levels of urls.

Return values:

|  |  |  |
| --- | --- | --- |
| **Code** | **Example Response payload** | **Scenario** |
| 200 | {"status\_message": "Job created", "job\_id": "5f62f38b8dae42e7b2ee5f6f60d1ea68"} | Scraping job created successfully |
| 400 | {"error": "Invalid request"} | Request is invalid (parameters are missing or are not legal) |
| 500 | {"error": "An error has occurred"} | General API errors |

 GET http://127.0.0.1:8080/api/v1/jobs/<job\_id>

Request parameters:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| job\_id | String (encoded uuid) | Scraping job id as returned by the other api |

Return values:

|  |  |  |
| --- | --- | --- |
| **Code** | **Example Response payload** | **Scenario** |
| 200 | {  "status\_message": "Job done",  "job\_id": "e9aec41688d148dc904ea594e86aed9f",  "results": {  "url": "https: //www.google.com",  "urls": [  {  "url": "www.google.com/about",  "urls": []  },  {  "url": "www.google.com/sign\_in",  "urls": [  {  "url": "https://www.google.com/about",  "urls": []  },  {  "url": "http://mail.google.com/hello",  "urls": [  {  "url": "mail.google.com/login",  "urls": []  }  ]  }  ]  }  ]  }  }  \* This is an example output of depth == 3 | Scraping job completed successfully |
| 206 | {"status\_message": "Job in Progress", "job\_id":"e9aec41688d148dc904ea594e86aed9f"} | Scraping job is in progress |
| 404 | {"job\_id": "e9aec41688d148dc904ea594e86aed9",  "error": "Job not found"} | Scraping job doesn’t exist or incorrect job\_id |
| 500 | {"error": "An error has occurred"} | General API errors |
| 500 | {"error”: "Scraping job has failed"} | Scraping job has failed |

When a job is completed successfully its results are stored in the DB and the api should return the results for the job “indefinitely” (or until it’s otherwise deleted from the DB).

A successful job’s result is a tree of the urls that are the descendants of the given url for the scraping request. The result tree can have only one root (the given url) but 0 or more descendants (depending on the original depth parameter and the webpage itself).

If a scraping job fails (e.g. some internal error has occurred) the job should be discarded (i.e. should **not** be retried) and the api should return a 500 error response (Scraping job has failed) when queried for the job’s results.

Any other issue (e.g. errors outside of the scraping) should return a 500 error response (An error has occurred).

Unfortunately, our scraping service is incomplete. The implementation for the scraping section is missing and we also know of 1 bug in the code that was discovered during an early testing phase. The testing results document was lost, however, so we can’t be sure what was the issue.

**Deliverables**

In the given time you will need to do the following:

1. Find and fix the bug.
   * Bugs are issues that would prevent a “happy flow” from running correctly e.g. returning incorrect response, simply failing etc.. Unhandled edge cases and design choices that can be disputed are not considered bugs for the purpose of this exercise.
2. Complete the web scraping service such that the following deliverables are achieved:

* Web server “listening” to port 8080
* Expose an API as described above
* Implement the code in *UrlScraper* that is responsible for the website scraping
* Bonus – make scraping jobs run in parallel

**General Info**

* Please read this document carefully and make sure you understand the deliverables for each part of the exercise.
* Feel free to use open source packages that can help with your implementation.
* We strongly encourage you to ask questions if anything is unclear or if you hit a wall, so both sides can make the most out of this exercise.
* We care about correctness, efficiency, robustness, code style, OO design, architecture, and coding best practices – so please pay attention.

**Tips**

* Our goal is to see how you think and what are your key values when creating software – focus on what you think is most important to complete in the time frame given and document the rest, so we can understand you better. It’s OK not to fully complete the exercise, but it would help us (and you) if you can write down potential issues, limitations etc. that you foresee with your implementation.
* Although this is only an exercise please note that things such as memory management, taking into account that the running time should be kept to minimum etc. are important and should be addressed.

Good Luck!