University of St.Gallen

Institute of Computer Science

Distributed Systems, Autumn Semester 2023
Jérémy Lemée, Sanjiv Jha, Andrei Ciortea
{jeremy.lemee, sanjiv.jha, andrei.ciortea}@unisg.ch

Assignment 2: The Anatomy of a Search Engine (7pt)

Deadline: Oct. 17, 2023; 23:59 CET

In this assignment, you will get practical experience with hypermedia systems. Given an artificial hypermedia environment[1], you will create a basic search engine for the given environment—including a crawler, an index, a Web front-end, and its back-end:

- In the first task, you will create a crawler that explores a hypermedia environment and generates an index document for this environment.
- In the second task, you will generate a flipped index from the index created in the first task—and you will use this flipped index to answer queries.
- In the third task, you will design and implement an API for your search engine.
- In the fourth task, you will optimize the crawler from the first task through a multithreaded implementation.

The project uses the Java framework *Spring*[2] for the implementation of the Web server.

(1) Go Crawl! (2pt)

In this task, you should first implement a program (using the template `SimpleCrawler.java`) that automatically navigates (crawls) the entire hypermedia environment, starting from the Web page https://api.interactions.ics.unisg.ch/hypermedia-environment/cc2247b79ac48af0 to build an index of all encountered Web page contents. That is, after executing this program, it should return a table that contains, for each Web page, the contents (three terms) that occur on that Web page. This table, which we refer to as an index, should be stored in a CSV file (index.csv) with a single URL per line (see example in Listing 1).

Listing 1: index

```
/589c85937f90b989, hello, world, amused
/19dffbabe7ab6d1, three, amused, sheep
/fd4f6c6f3b65d9ae, world, planet, three
...
```

Hints:

- Use Jsoup to process HTML pages: (API here: https://jsoup.org/apidocs/)
- Use OpenCSV to process CSV files (API here:https://opencsv.sourceforge.net/apidocs/index.html).

[1] The reason for supplying you with an artificial environment is that we want you to focus on the core goals of this assignment, rather than the nuisances of parsing actual Web pages in the wild. What you learn here however directly applies to the World Wide Web "out there" as well.

[2] https://spring.io/

(2) Searching, Fast and Slow (2pt)

In this task, you will implement another main component of your search engine, and experiment with it. For this, first, we create a program (`IndexFlipper.java`) that reads the CSV file (`index.csv`) from the crawler and flips the index (look at Listing 2). That is, given the example output above, your program should produce the output below and store it in another CSV file (`index_flipped.csv`).

Listing 2: index flipped

```
hello , /589c85937f90b989
world , /589c85937f90b989 , /fd4f6c6f3b65d9ae
amused , /589c85937f90b989 , /19dffbabe7ab6d1
three , /19dffbabe7ab6d1 , /fd4f6c6f3b65d9ae
planet , /fd4f6c6f3b65d9ae
sheep , /19dffbabe7ab6d1
. . .
```

In your submission, report whether the resulting– *flipped index* is smaller or larger than the original one from *Task 1*! What does this depend on? Finally, complete yet another program (`Searcher.java`) that lets the user enter a single word (keyword) and, using the flipped index, returns the URLs of all Web pages that contain that term.

Hints:

- Use System.currentTimeMillis() to get the current time.

(3) Search Engine Web server (1.5pt)

The final component of your search engine is a Web service, which you will implement as part of this task. In order to do so, you have to complete the program `SearchEngine.java`. This task has two parts. In the first part, you are required to design an HTTP API for your service. In the second part, you are required to create a user interface for the search engine.

Search Engine API We provide you with an OpenAPI specification (`search_engine_api.yaml`) for your search engine's API in the assignment package. Your task is first to complete the `SearchEngine.java` program to create a server that implements the API.

The search engine API enables certain features in three categories: an interface for end-users, a search interface, and an administration interface.

The **interface for end-users** is an HTTP endpoint from which browsers can retrieve the HTML page for the user interface. The design of this page is done in the second part of the task.

The **search interface** supports two operations: a "search" operation that allows users to get the URLs of the pages containing a certain keyword, and a "lucky" operation that allows users to get the first URL of a page containing the keyword.

Second, your task is to extends the API by writing the administration interface. This interface does not need to be actually implemented.

The **administration interface** should enable administrators to launch a new crawling operation, to regenerate the flipped index, to delete a URL from the index, and update (or add) the information concerning a given URL in the index. You are required to extend the OpenAPI specification of the search engine to support this functionality. You are also required to document all your design decisions as part of this task in `Report.md`.

User Interface The initial Web page available at the interface for the end-user should have a text field to enter a search term and two buttons. When clicking the first button, *Search*, the user interface should render all URLs that this search term occurs on, as clickable hyperlinks. When clicking the second button, *I'm feeling Lucky*, the user interface should directly take the user to any of the pages that certainly contain the specific term.

Hints:

- Use HTML to create your web page.

(4) Multithreaded Crawler (1.5pt)

Since the crawler spends a lot of time waiting for responses from the Web server in general, multi-threading would lead to a large performance increase for your crawler.

Complete the program `MultithreadedCrawler.java` to implement a multi-threaded crawler.

Indicate in the `Report.md` the time necessary for the **SimpleCrawler** (that you created in `Task 1`) to work, the time necessary for the **MultithreadedCrawler** to work, and indicate the ratio as SimpleCrawler / MultithreadedCrawler.

Hints:

- Use the Spring class **ThreadPoolTaskExecutor**, described in https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/scheduling/concurrent/ThreadPoolTaskExecutor.html.

Hand-in Instructions By the deadline, you should hand in a single **zip** file via Canvas upload. The name of this file should start with a3 and contain the last names of all team members separated by underscores (e.g., a2_lemee_jha_ciortea.zip). It should contain the following files:

- All answers to the assignment questions in the given `REPORT.md`.

Across all tasks in this and the other assignments in this course, you are **required to declare** any support that you received from others and, within reasonable bounds,[3] any support tools that you were using while solving the assignment.

[3]It is not required that you declare that you were using a text-editing software with orthographic correction; it is however required to declare if you were using any non-standard tools such as generative machine learning models such as GPT