# Report on Web Crawler Code

## Introduction

The provided code represents a basic web crawler implemented in PHP, HTML, and JavaScript. The web crawler is designed to fetch and parse HTML content from web pages, following links to a specified depth. The crawled data is stored locally, and there is a search functionality to find relevant information within the crawled content.

## 1. HTML Structure (index.html)

The HTML file serves as the user interface for the web crawler. Key components include:

* **Input Fields:** Users can input the starting URL, set the crawling depth, and enter search text.
* **Buttons:** The "Crawl" button triggers the crawling process, and the "Search" button initiates a search within the crawled content.
* **Progress Bar:** Displays the progress of content loading during crawling.
* **Search Results:** Displays the results of the search.

## 2. Web Crawling Logic (crawler.php, crawler-utils.php)

### 2.1 Crawler Initialization

* The user provides a starting URL and depth.
* The crawl function is triggered when the "Crawl" button is pressed.
* The crawling operation is performed asynchronously using the fetch API.

### 2.2 Crawling Process

* The crawler.php script initiates the crawling process.
* The pop\_highest\_priority\_url function selects URLs based on Jaccard similarity scores.
* Disallowed URLs from robots.txt are respected.
* Crawled HTML content is saved in the "loaded\_pages" directory.
* The process is logged in "scraping\_log.txt."

### 2.3 Crawling Control

* Crawling depth is decremented with each iteration.
* The process continues until the specified depth is reached or no more URLs are available.

### 2.4 Robustness

* The code accounts for unsuccessful page loads and logs errors.

## 3. Search Functionality (search.php, search-utils.php)

### 3.1 Search Initialization

* The user enters search text, and the search function is triggered.
* The search is performed asynchronously using the fetch API.

### 3.2 Search Process

* The search.php script extracts text content from crawled HTML files.
* Cosine similarity is used to find the most relevant documents.
* The top N documents are selected based on similarity scores.

### 3.3 Search Output

* The relevant content is concatenated and returned as the search result.
* Search terms are highlighted in the output.

## 4. Backend Scripts (completion.php, config.php, crawler-utils.php)

* completion.php reports the number of HTML files crawled, facilitating progress tracking.
* config.php contains configuration details, such as file paths.
* crawler-utils.php provides utility functions for crawling, such as URL cleaning and disallowed link extraction.

## 5. Considerations and Improvements

* **Security:** The code might benefit from input validation and sanitization to prevent security vulnerabilities.
* **Logging:** Detailed logging is implemented, aiding in debugging and understanding the crawler's behavior.
* **User Interface:** The UI is functional but basic. Enhancements could include better visual feedback and error handling.
* **Scalability:** For larger-scale crawling, considerations for distributed systems and handling concurrent requests could be explored.

## Conclusion

The web crawler demonstrates fundamental concepts of web engineering, including HTML structure, asynchronous communication, server-side scripting, and data extraction. Further enhancements and optimizations can be applied to improve its functionality and robustness.