# Scientific Paper

Jesus Arencibia David García Javier Franco
Alejandro Bolaños Pablo Guilló

December 10, 2023

## Contents

## 1 Abstract

The ever-expanding volume of online information necessitates the development of efficient methods for data retrieval. This paper investigates the role of the inverted index in web crawling, underscoring its significance in organizing sub-folders. Acknowledging the imperative for effective data indexing, we have identified a tree-like directory structure as the optimal solution. This organized hierarchy of sub-folders streamlines the storage and retrieval of web data. Additionally, metadata management has been integrated to ensure synchronization with content.

Our project, developed in Java, focuses on web crawling, data storage, and the establishment of an inverted index with word search APIs. The utilization of nested folder structures enhances the efficiency of data management. The study underscores the significance of robust data engineering practices and provides insights into the complexities of web crawling and data retrieval.

Furthermore, this work encompasses the refinement of search capabilities, scalability through cluster deployment, and ongoing enhancements for continued success. This project serves as an illustration of the practicality of these

technologies in real-world scenarios, highlighting the importance of efficient data management and retrieval.

## 2 Introduction

The exponential growth of information on the web has heightened the need for efficient data retrieval mechanisms. In this context, the inverted index has emerged as a pivotal tool for optimizing web crawling and structuring retrieved data. This paper explores the core aspects of harnessing the inverted index in the realm of web crawling, underscoring its role in facilitating effective subfolder organization. By cultivating a comprehensive understanding of the inverted index's capabilities, this study seeks to address challenges associated with managing vast amounts of web data and propose an enhanced framework for subfolder creation and management. Through an in-depth analysis of the fundamental principles of the inverted index and its implications for web crawling, this research highlights the significance of strategic subfolder creation in facilitating streamlined data storage and retrieval. By dissecting the complexities of integrating the inverted index with the web crawling process, this paper aims to offer valuable insights into the development of robust and scalable systems for efficient web data management.

## 3 Problem statement

Despite the advancements in web crawling techniques, the efficient organization and retrieval of vast amounts of web data remain a persistent challenge. The traditional methods of data indexing and storage often prove inadequate in handling the sheer volume and diversity of information available on the internet. This leads to issues such as inefficient data retrieval, prolonged processing times, and inadequate organization of crawled data, hindering the seamless extraction and utilization of valuable information. Additionally, the lack of an optimized mechanism for subfolder creation within the web crawling context further exacerbates the complexities associated with data management and retrieval. Therefore, there exists a pressing need for a comprehensive solution that leverages the capabilities of the inverted index to enhance the efficiency of web crawling and facilitates the systematic organization of retrieved data into structured subfolders. This solution must address the challenges posed by the dynamic nature of web content and the varying data formats, ensuring streamlined data retrieval, storage, and management for improved accessibility and usability.

## 4 Method

We will now describe the methodology and operation of our Inverted Index system in java, which is composed of four interconnected modules: Cleaner, Crawler, Indexer and Query. These codes are in charge of four fundamental tasks: the cleaning of the books and the obtaining of the metadata of the documents, the downloading of digital books from a web page from time to time, the creation of inverted indexes from the downloaded sets of books in order to obtain the words that appear and finally an interface with the user for the efficient search of keywords in these sets of documents.

The Cleaner module, which is responsible for extracting essential metadata from digital books. This metadata comprises fundamental information, such as title, author, language and year of publication, and plays a crucial role in the effective organization and search of books in the digital book management system. Its operation is divided into the following stages: metadata removal, text normalization, JSON object creation and metadata storage.

The metadata extraction process starts by reading the book files provided as input. Each file is processed line by line in order to identify relevant information, using regular expressions to search for metadata information in each line of the file. After extraction and storage of the metadata, a "Book" object is created, which integrates the retrieved information, such as title, author, language and year of publication. This object is represented in JSON format and stored in a dedicated metadata file. The metadata file name is identical to the original book file, but with an additional extension "metadata." For example, if the original book is named "book1.txt," the corresponding metadata file will be named "book1.metadata." It should also be noted that books are stored individually in metadata files with a specific extension, which simplifies access and administration in the context of the overall digital book management system.

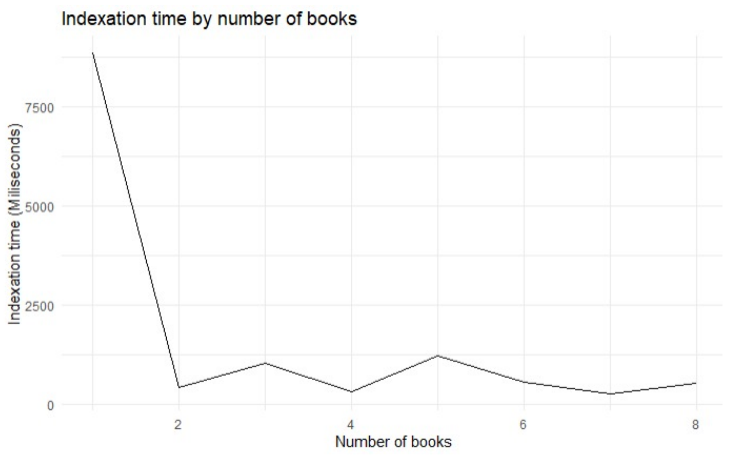
The Crawler module focuses on downloading books from external sources, a crucial step in building a digital library. The download process is divided into three stages: connecting to the online source, reading and writing data from the source to a local file, and completing the download. The downloaded content is stored locally in the system with a file name generated from a unique book identifier. This allows keeping an organized record of downloaded books. On the other hand, the indexing module plays a critical role in digital book management by scanning directories, indexing books and organizing them to facilitate efficient searches. It starts with the configuration of access paths to the datalake (source of the books) and datamart (where the indexed books are stored), as well as keeping track of the files already indexed to avoid duplicates. Index generation begins by scanning the contents of the datalake for text files. Each valid file is indexed, avoiding duplicates. The Index module provides a search function that allows retrieving books related to specific keywords. If the datamart is empty, it is identified for management decisions.

Finally, the Query module provides a versatile query interface for accessing books containing a specific keyword or released within a specified date range. Users can search for books that reference a given word and get information about where that word appears. Additionally, the module supports a date-based search system, allowing users to specify a range of dates, and the system will retrieve books published within that timeframe. For example, a user could initiate a date-based search with a GET request to the URL "/books/word/start date/end date". This triggers the getBooks function, enabling users to explore books that match both the keyword and date criteria.

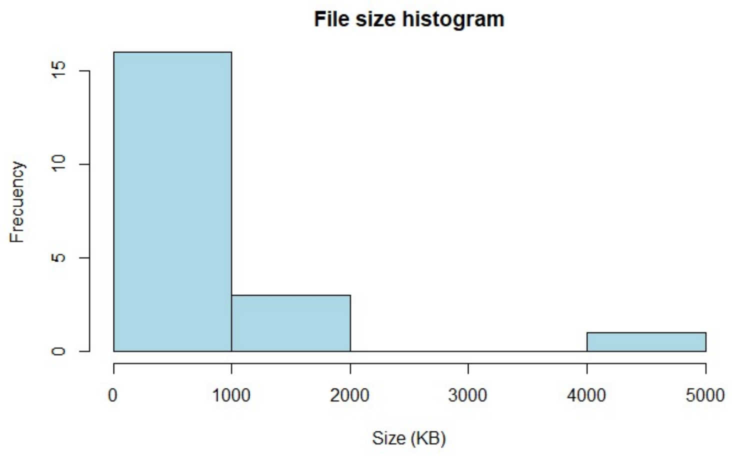
The API execution is carried out by the main function, which initiates a Spring Boot application and a web server to handle GET requests. In summary, the module provides an effective and dynamic query interface, enabling users to search for books by keyword or publication date in a digital book management system. This enhances the user experience by allowing them to access detailed information about the presence of a keyword in books and explore digital content based on both keywords and publication dates.

## 5 Experiments

In the concern of experiments, they were focused on the correct functionality of the program. We changed the parameters of our task to check that it worked in different scenarios and guarantee that we obtained the desired results. Since this is not a numeric program and it is focused on the indexation of books, we considered that the most important thing was the indexation time of them. That is why we realized the following graph, which represent the indexation time of the first 8 books.



We can clearly see that the first book takes a lot more time than the following ones. That is because it is creating the Datalake and all the words in the book are new, so it has to create a file for all of them. Once this is done, it is far easier for the rest of the books to be indexed. The previous creation of folders and files for common words isn´t needed to be done again, which speeds up the process a lot. Of course, the indexation time also depends on the size of the books and the number of words it has. Notwithstanding, as we can see in the graph, the indexation time is really quick, lasting less than 2 seconds except for the first book. Taking into account that a book is going to be indexed every minute, this performance is efficient. In fact, the size of the books should not be a problem, as we can see in the following graph



This is an histogram of the size of the first 20 books in project Gutenberg. We appreciate that the weight of almost all books is under 1MB, with some exceptions. To have an idea, the first book, which lasted the longest in being indexed, is one of the heaviest. This means that since the rest of the books will probably weigh less than 1MB and, as we said earlier, the datamart will already be created, we will have no problem with the indexation time if there is a book which weighs a lot. That is because we will not encounter many of these kinds of books, and also because the Datamart wont need to create as many files as in the beginning, since we already would have encountered the most common words in the previous books. In conclusion, the indexation time will be fast enough to deal with the new books incoming.

## 6 Conclusion and future work

In this Java project, we developed a web crawler to fetch a book from the web every minute and stored the data in a data lake, segregating metadata and content for better data management. We also built an inverted index to track word occurrences in various books and created an API for word searches within the index. Within this project, we've adopted a nested folder structure for both the data lake and the data mart (inverted index). In the data lake, we have organized the data into three main sections: "books," "metadata," and "content."

Each of these sections follows the same nested structure, comprised of folders labeled with numbers from 1 to 9. Within folder "1," you'll find subfolders "11," "12," "13," and so on, maintaining this structure up to a maximum of three digits. The inverted index operates in a similar manner, with folders containing letters since it stores words. This structure is also organized into multiple levels, with a maximum of three levels deep. Throughout this project, we encountered several challenges and gained valuable insights. We encountered some difficulties when organizing the data lake into nested folders, properly separating the metadata from the content. Furthermore, designing and optimizing the inverted index for efficient search operations presented its own set of challenges. One of the key takeaways from this project was the importance of robust data engineering practices. We learned the significance of creating scalable and reliable data pipelines, as well as how to manage metadata and content separately for ease of retrieval and maintenance. In conclusion, this project allowed us to delve into the complexities of web crawling, data storage, and retrieval, as well as information retrieval techniques. It provided an excellent opportunity to enhance our skills in Java programming and data engineering. This project underscores the practical applicability of these technologies in real-world scenarios, where efficient data management and retrieval are essential. Moving forward, there are several avenues for further improvement and expansion. One such avenue is the enhancement of our search engine capabilities, enabling more complex and refined searches. Additionally, deploying the system in a cluster environment could significantly boost its scalability and reliability, allowing for larger-scale data retrieval and analysis. These future endeavors will further strengthen the project's utility and effectiveness. Ensuring that our infrastructure can handle a growing volume of data and user requests is essential for long-term success.