CS242 Project – Part A submission

LOTR INFORMATION RETRIEVAL

JORGE MERCADO, hOVANES KESEYAN, Janakiram Kuppa

2018

Contents

[1. Objective 2](#_Toc505280439)

[2. Collaboration details: 2](#_Toc505280440)

[2.1. Jorge Mercado 2](#_Toc505280441)

[2.2. Hovanes Keseyan: 3](#_Toc505280442)

[2.3. Janakiram Kuppa: 3](#_Toc505280443)

[3. Crawling System Overview 5](#_Toc505280444)

[3.1. Crawling Architecture 5](#_Toc505280445)

[3.2. Crawling Strategy 6](#_Toc505280446)

[4. Lucene Indexing Strategy Overview 8](#_Toc505280447)

[4.1. Indexing Architecture 8](#_Toc505280448)

[4.2. Text Analyzer choice 9](#_Toc505280449)

[4.3. High-level Data Model of Potential features from LOTR IR 9](#_Toc505280450)

[4.4. Index creation process run time results 10](#_Toc505280451)

[5. System Limitations 11](#_Toc505280452)

[6. Obstacles and solutions 11](#_Toc505280453)

[7. Lucene Indexing Strategy Overview 11](#_Toc505280454)

[8. How to deploy the crawler 13](#_Toc505280455)

[9. How to build the Lucene Index 13](#_Toc505280456)

# Objective

Objective of this project is to a create web crawler to crawl websites that contain data about characters, their details, locations in the world, and other information gathered from plot points and stories told in the world created in legendary author J.R.R. Tolkien’s works: The Lord of the Rings, The Hobbit, and The Silmarillion. Intent is to crawl a large and popular online Wiki located at http://lotr.wikia.com, known as “The One Wiki to Rule Them All”. This wiki is populated by fans with data from all the literature behind this world, including not only the original books but other canonical media such as the popular and successful movie series from the 2000s. Each character has a wiki page which describes any known information about their timeline, city/town of origin, actors that have portrayed them in media, relationships to other characters, and important plot points among the immense library of information that is available. The data collected from the crawler will indexed using Lucene first, and later using Hadoop. We will compare the performance of these alternatives.

# Collaboration details:

## Jorge Mercado

Jorge helped develop the concept in java by coding a significant portion of Crawler. Jorge provided the overall direction to the team on how to implement the crawler and provided valuable inputs on best to achieve the results. Jorge helped integrate the Indexing code into the base crawler code.

Crawler Code built by Jorge has the following concepts embedded within it:

1. Get all URLs in page and post to queue
2. Do not crawl a page if it leaves the site
3. Parse Robots file to ensure that code follows the crawling ethics
4. Skip a website if it has already been crawled
5. Retrieve the features needed for setting up the search engine in the next phase of the project

## Hovanes Keseyan:

* Hovanes researched various topics that were discussed in the team meetings and came up with the proposal that is interesting, unique and one that meets the requirements of the project. So, the consensus was to crawl and build a search engine based on the world created in legendary author J.R.R. Tolkien’s works: The Lord of the Rings, The Hobbit, and The Silmarillion.
* Hovanes helped write up the summary of the intent of our project that was submitted to Prof Hristidis and Nhat Le.
* Hovanes also helped put together a data model for the potential features that we plan on collecting from the webpages.
* Hovanes implemented portions of web crawler code to identify and retrieve relevant text content of the webpage without erroneous web page data. Hovanes also implemented code for the crawler to extract personal information items from the right sidebar of each page to be indexed as a subset of the total page content.

## Janakiram Kuppa:

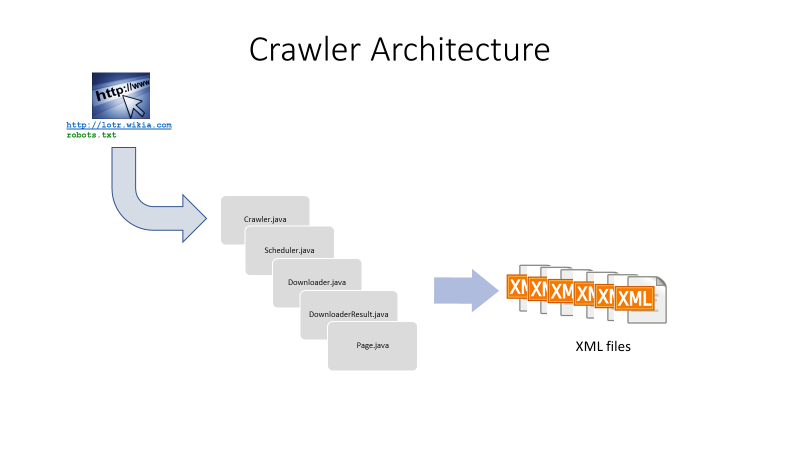
* Janakiram handled coordination with the team depending on the needs and availability of the team.
* Brought various ideas on what we can do for the project that were all discussed and considered before the team decided to go with Hovanes’s recommendation of crawling the web for Lord of the Rings world.
* Helped provide suggestions on creating code so that it can be easily scaled when we do Part B of the project and the code is written so that it has decent performance. He proposed that we could adopt techniques so that the data we get from the crawler goes into a memory buffer and commit the entire buffer to a file when the buffer gets full.
* Coded the Indexing logic using Lucene libraries.
* Created the Crawler and Indexing process architecture diagrams.
* Helped put together most of the Project report for the project.

# Crawling System Overview

## Crawling Architecture

Crawler Architecture has been designed so that it puts out XML files as it processes the lotr.wikia.com website and all the associated links from that webpage. Every XML file contains a document comprising of several data elements. These XML files will go in as Input to the Indexer. Figure 1 below illustrates the architecture of the crawler.

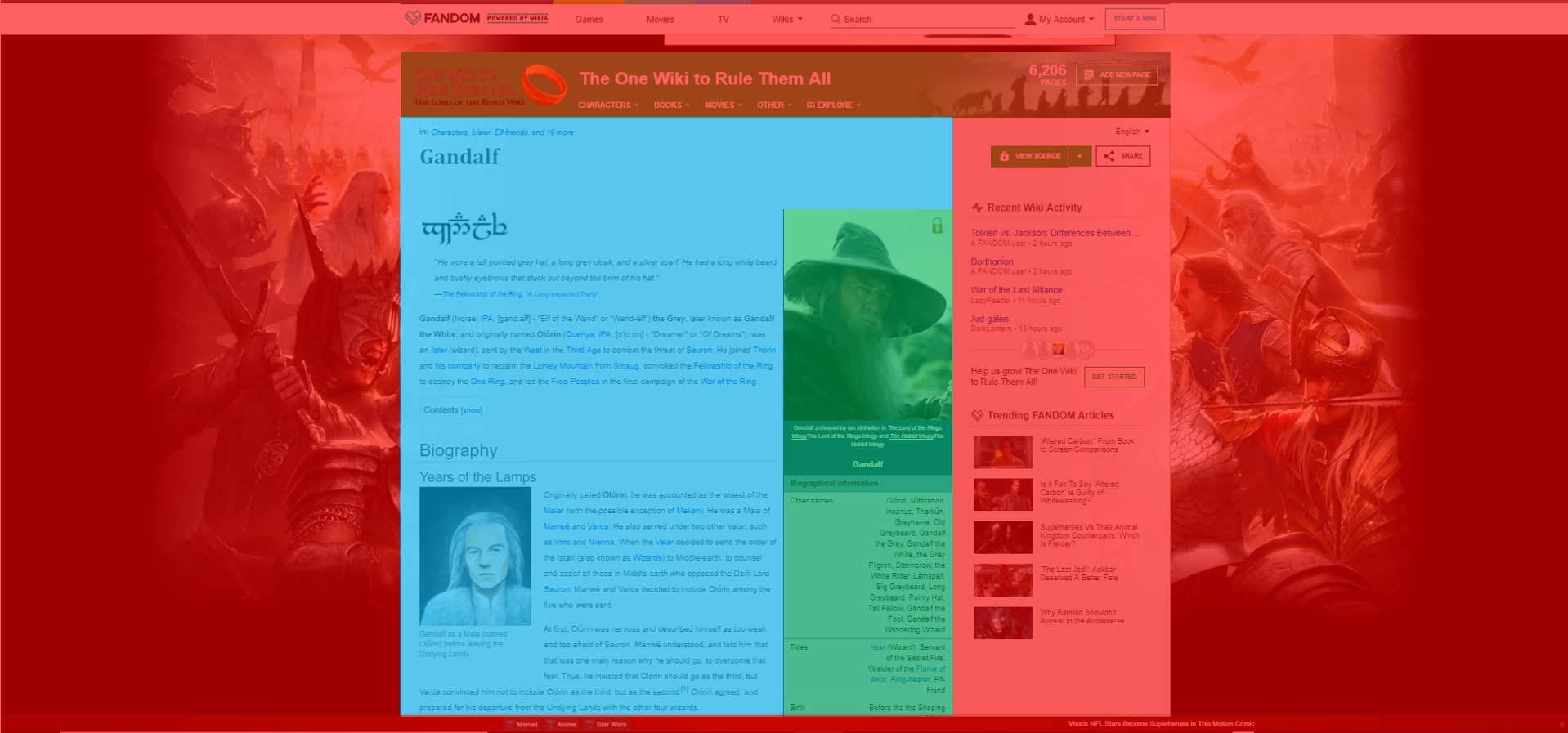
**Figure 1: Crawler Architecture**



## Crawling Strategy

Crawler code starts with the seed page of <http://lotr.wikia.com>. Then it reads in the Robots.txt file. The code follows the crawling ethics set in the Robots.txt file by checking if crawling is disallowed or allowed. If crawling is disallowed, it skips over the page and if it is allowed, it invokes the logic to download the content of the page. Some of the core elements that are captured include the Title, URL, the main block of text content on the page, and the personal information fields that are needed for setting up the search engine in the next phase of the project. Every attempt will be made to capture most of the data elements mentioned in the data model section of this document. During a search query, a search can be performed on either the entire block of text or the personal information elements. Terms found in the personal information field will be treated with a higher weight than those not in that field. Figure 2 below illustrates the elements that would be collected and indexed on an example page. Green indicates “personal information” property fields from the right sidebar, blue indicates main body text, and red indicates erroneous content that is not indexed.

**Figure 2: Indexed Content**



Crawler code checks for duplicate pages and skips over the web pages that have already been processed once.

Crawler code has been enhanced to do multi-threading as well to ensure that it operates on a multiple threads thereby increasing the performance of the crawler.

Time taken to process the content download on each page is recorded by getting the difference between the start and end time before and after invocation of the java code that invokes the downloader. As the crawler code is running, user will be able to witness the URL of every page being processed, depth of the page, disposition of the page to indicate if it has been skipped or not, the name of the thread being processed, and the download time taken by the crawler.

Get and Set methods have been coded for the core elements that need to be captured by the crawler.

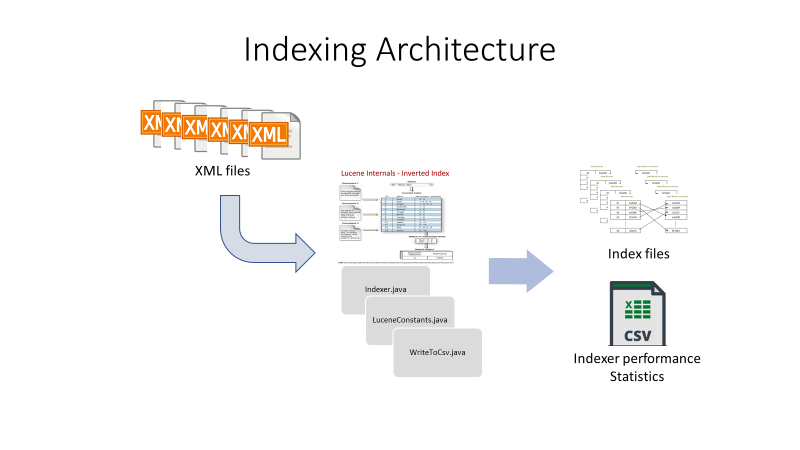
Output of the Crawler code is a series of XML files. Currently, the design is kept at the most granular level so that each XML contains only 1 document. This can be modified so that multiple documents fit into a single XML if the need arises in the future. Each document that is downloaded from the crawler will contain all the data elements that are needed for indexer so that keyword search functionality can function well during the search engine phase of the project.

# Lucene Indexing Strategy Overview

## Indexing Architecture

Indexing Architecture has been designed so that the indexing logic reads the XML files put out by the Crawler. Indexing logic will create index files using Lucene framework (open source java based indexing and search library). In addition, Indexing logic will put out a CSV file containing the time taken by the indexing process on each document. Figure 3 below outlines this architecture.

**Figure 3: Lucene Architecture**



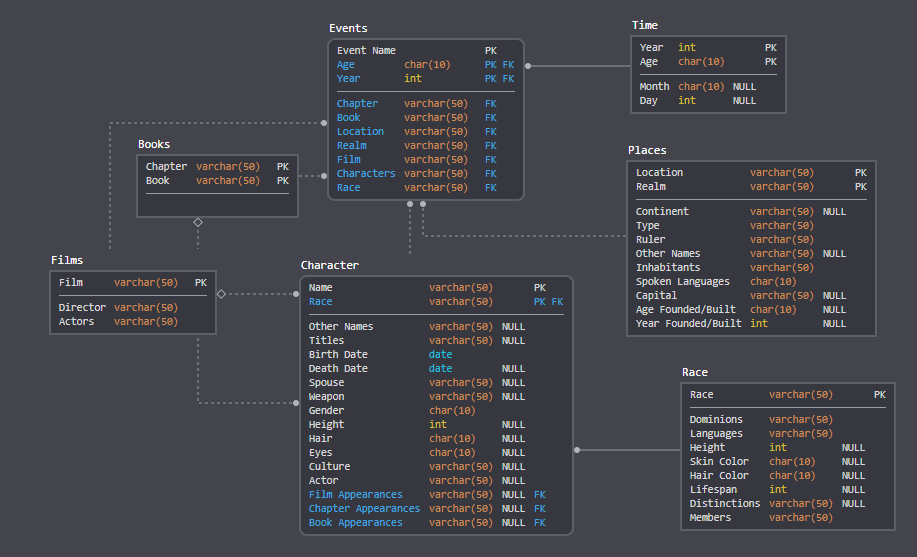
## Text Analyzer choice

Stop Analyzer was invoked and used within the Indexing code to ensure that all the unwanted keywords (example: and, the, or etc.) that won’t add value to the search engine in the later phase of the project are dropped from flowing through the indexing logic and don’t get indexed.

## High-level Data Model of Potential features from LOTR IR

Figure 4 below outlines the high-level data model for the proposed features that will be captured by the crawler of the LOTR wiki link.

**Figure 4: Proposed Data Schema**



## Index creation process run time results

<to be filled in later when the entire code is ready>

# System Limitations

Since the code runs only on a single machine and not in a distributed environment, the performance of the crawler and indexing process to complete dumping the 5GB worth of data takes a few hours.

# Obstacles and solutions

All the team members being new to the collaborative team environment for an Online course had to come up with innovative ways to communicate effectively and achieve the common goal of getting the project executed successfully.

Team decided that it is best to use an Integrated Development Environment for this project. So, it was decided to use open source IntelliJ IDE which proved to be immensely useful.

In addition, a few initial challenges on how to share the code with each other was overcome by the decision to use github which helped immensely in coordinating various releases of the code through the execution of the project.

Capturing each of the proposed data features (~40 fields), upon deeper thought into the indexing logic behind it, proved to be too high risk for an unnecessary reward. Most of the proposed properties are found in the right sidebar of each wiki page, and have mostly unique values to their respective feature fields, therefore treating all text in this sidebar and querying it as one block will yield practically the same result as extracting each feature and treating it as its own. It should also help with compute time. Data from these fields may be separated later upon visualization of the data set as much can be done with this wealth of data.

# Lucene Indexing Strategy Overview

Core of the indexing logic revolves around usage of Lucene framework which does the bulk of the work to create the index files. The appropriate lucene java classes (analyzer, document etc) had to be used along with the applicable methods within those classes (getdocument, adddocument etc) by invoking the IndexWriter method. Indexing process has the logic to capture the processing time taken to index a document. Times are captured at the most granular individual document level. The team felt that this gives the maximum flexibility in terms of how to graphically report the run time numbers for the indexer against the number of documents processed. In addition, it was felt that this strategy would come in handy in the future phase of the project when comparison needs to be made between the performance of the indexer with lucene in a regular single processor environment against the performance in a distributed environment like Hadoop.

# How to deploy the crawler

<Need Jorge to fill this section>

# How to build the Lucene Index

<Need Jorge to fill this section>