Hong Kong Baptist University

COMP 4047 Internet and World Wide Web

DESIGN AND IMPLEMENTATION OF A SEARCH ENGINE

Authors: Marco Ambrosini, Robert Herman, Jonathan Rempel

Group: No. 8

Prof.: Y. W. Leung

Date: 01. November 2016

Table of Contents

[1 Introduction 3](#_Toc465615792)

[2 Design 4](#_Toc465615793)

[2.1 Components 4](#_Toc465615794)

[2.2 Communication 5](#_Toc465615795)

[2.3 Classes 6](#_Toc465615796)

[2.4 Data structures 8](#_Toc465615797)

[3 Implementation 10](#_Toc465615798)

[3.1 Configuration 10](#_Toc465615799)

[3.2 Installation 10](#_Toc465615800)

[4 Appendix 11](#_Toc465615801)

[4.1 List of Figures 11](#_Toc465615802)

[4.2 List of Tables 11](#_Toc465615803)

[Participation Form 12](#_Toc465615804)

# Introduction

The group project of the course COMP 4047 Internet and World Wide Web comprises the design and implementation of a search engine which crawls the web from a specified starting point storing the information in a way that allows the user to search for specific keywords or phrases.

This document is to describe the design and implantation of said search engine.

# Design

The search engine is a complex software which makes use of various components. This section shall provide insight into the functions of each single component including the communication between each other.

## Components

The following list shall provide an overview of the most relevant components and their core function.

* web-crawler.jar  
  This java application is the heart of the software managing both the crawling of the web as well as searching the stored information for the requested keyword or term. The program is written in java and the section *2.3 Classes* provides more detailed information of the individual classes. An executable jar file called ‘web-crawler-jar-with-dependencies.jar’ is created using maven, which bundles the necessary dependencies to run the project.
* Maven & pom.xml  
  Maven is a Java build automation and dependency management tool. In addition to providing automation, it also enforces a moderately strict directory structure, and promotes modular development of code. Additionally, while not implemented in this project, Maven allows for automated testing, where passing the testing is a pre-condition for the deployment phase. The pom.xml file contains all the rules and information required by Maven to create the executable jar-files that are bundled with dependencies as well as maintain the dependency versioning for the project. When building the executables using maven these dependencies and paths are resolved correctly allowing the ability to install and run the application to and from any arbitrary directory.
* config.properties  
  This configuration file allows to easily alter some parameters used in our application. As the file is independent of the Maven build-cycle, a user can modify the parameters without having to recompile and deploy the project.
* IgnoreList.csv  
  This text file contains a list with all words which shall be ignored while crawling the web. Changes to this list will be effective upon restarting the application, and as with config.properties to not require recompilation or redeployment.
* CrawlerStorage.mv.db  
  This is a simple embedded in-memory database file (H2) which comes with a build-in java API. We consider this database a good fit for this project for the following reasons:
  + It is easy to implement and handle – good for a proof of concept, but not necessarily good for scaling in a commercial environment.
  + In-memory databases also provide better performance as memory-access is much faster than accessing disk-based data storage.
  + H2 abides by the common RDMS ACID properties: Atomicity, Consistency, Isolation, and Durability.
  + H2 provides a SQL-like syntax and SQL-like features meaning that it is easy to understand, and modify for most developers.
  + Despite being an in-memory embedded database, H2 also allows its contents to be written to disk so that it can be stored while the application is not running.

The database is used to store and retrieve all information gathered by the crawler. Section *2.4 Data structures* describes the used tables and their relation in more detail.

* start-server.cmd  
  This Windows batch-file is used to launch the server application and has no other function then calling the python script ‘server.py’ described below.
* server.py  
  This python script is executed directly after launching the application and instructs the java application to crawl the web using a special keyword. The keyword and also the duration of the crawling can be modified using the configuration file. After the java application has finished crawling ‘server.py’ sets up a webserver and opens ‘index.html’, the homepage of the search engine, in the default browser.
* search.py  
  This python script is called from ‘index.html’ when the user clicks the search button. The keyword or term is passed to the java application which then returns a new webpage with the results.
* Results.ftl  
  The above mentioned new webpage is based on the template ‘Results.ftl’ which is an Apache FreeMarker template. This means that it dynamically modifies the HTML with the search results that are returned by the java application.
* index.html  
  This is the webpage called by ‘server.py’ after crawling has finished and servers as the Graphical User Interface (GUI) for users to set up search requests.
* client-start.cmd  
  This Windows batch-file is used to launch the client application and has no other function then calling the python script ‘client.py’ described below.
* client.py  
  In contrast to ‘server.py’ this python script does not instruct the java application to crawl the web but checks to see if the server is available. If it is, it opens ‘index.html’ in the default browser, allowing an end-user to quickly perform a web search. If the server is not available, it simply informs the user that they need to run start-server.cmd first.
* While not necessarily a component per se, an important design feature to note is that there is a unique key (specifiable through config.properties) which allows crawling and keyword searching to be done independently.

## Communication

The various components interact with each other using python scripts or an API. There are two entry points for a user to start an interaction with the application: One is by running ‘start-server.cmd’ which launches ‘server.py’ and thus instructs the application to start crawling and display the GUI thereafter. The other one is by entering a keyword or term in the form of the GUI and hitting the search button. Via the script ‘search.py’ this action will launch the application which ultimately returns a new webpage displaying the search results. The below flow chart provides a comprehensive high-level overview of the communication process of the application.



Figure 1: Process Model – Application Procedure

## Classes

The java application ‘web-crawler.jar’ consists of various classes which are shown in the following class diagram and explained in more detail below.



Figure 2: Class Diagram – Crawler Model

* Crawler.java is what crawls and retrieves data according to its specified inputs and stores it. It takes in the URL to start crawling from, the maximum number of URLs to search, the maximum size the URL pool can be at any given time, a HashSet of words to ignore when storing the words found on each page it crawls, and the maximum number of days a page may go without being updated and re-crawled. The process of the start() method is simply to start with the initial URL, retrieve all links on the page as well as words, store all of those, and add the links into the pool of URLs to be crawled. This process will stop either once the pool is empty, or the maximum number of URLs to crawl parameter is met.
* Http.java contains methods that are associated with HTTP operations. The get() method uses the Jsoup library to retrieve a page and then scan through the elements contained within. It will return a PageResult object which contains a list of all href tags (with any text after a # inclusive to be removed so as to only get the base URL), as well as all of the words on the page, with the condition that they are alphabetic only (no punctuation, or non-Latin characters), and are not on the list of words to ignore.
* PageResult.java is a simple POJO (Plain Old Java Object) which is used to store the data retrieved from a site that was crawled.
* PageResultException.java is a custom exception class that is used to identify exceptions that are generated during the course of attempting to retrieve a page and its contents.
* SearchServer.java is used as the main entry-point into the compiled jar file, and loads all of the configurable properties on start-up. Based on the arguments passed to it, it will either start crawling and indexing pages, or it will run the search. This process is separated so that not all pages are re-crawled for every query by the end-user.
* Storage.java is the interface in-between the java components and the H2 database layer for the web-crawler. It contains the necessary SQL statements to initialize the tables in the database if they don’t exist yet, insert or updated new data, and retrieve it based on the parameters it’s given.
* TemplateResponse.java uses the createHTML() method to create an HTML string dynamically by merging the results it’s passed from SearchServer.java and the Apache FreeMarker Results.ftl.
* WordResult.java manages the word searched, and how many times the word is used in a webpage.

## Data structures

Most of the data that is processed is only used temporarily and is managed by the java application using common data structures like HashSets, Maps, and ArrayLists. More complex data is represented by individual java classes such as PageResult, WordResult, etc. The only data which is required on a permanent basis and shall remain available for future sessions is the information regarding keywords and their URL. During crawling the java application processes this information and finally stores it in an H2 database. Since an URL can contain multiple keywords and a keyword can appear on multiple URLs, a third reference table is required to resolve this many-to-many relationship. Some information required for ranking of the search results is unique to any combination of keyword and URL and is, therefore, stored on their reference table. Additionally, a timestamp for each URL is stored in order to indicate when it was last crawled. The following entity relationship diagram illustrates the tables and their relation.



Figure 3: Entity Relationship Diagram – Crawler Storage (H2 Database)

# Implementation

## Configuration

Various parameters of the crawler can be changed inside the configuration file ‘config.properties’. This allows any users to easily customize this software to match their preferences. They can change any of the following:

* Initial URL which the crawler shall use as starting point.
* Max number of URLs to be crawled.
* Max Pool-size, which is the amount of URLs the crawler will temporarily store before crawling them. Limiting creates a more directed crawl and prevents exponential growth.
* The template which shall be used to display the search results.
* The amount of days after which the crawler regards a stored URL as outdated and subsequently re-crawl and update when the crawler is next triggered.
* A unique keyword which is used to instruct the java application to crawl the web rather than performing a search against the available database.

Changes made to the configuration file will be effective upon restarting the java application.

## Installation

The installation is simple and can be achieved by following these five steps:

1. Extract the two folders contained in group8.zip into any location.
2. Optional: update ‘config.properties’ in ‘/web-crawler/src/main/resources’ to your preferences
3. Make sure Maven is installed
4. Open a new cmd window and set the directory to be ‘/web-crawler’
5. Type and call the cmd ‘mvn clean install’

The application can now be started by executing ‘start-server.cmd’ in ‘/htdocs’

By using Maven for the installation, all dependencies and paths are resolved correctly allowing the ability to install and run the application to and from any arbitrary directory.

# Appendix

## List of Figures

Figure 1: Process Model – Application Procedure 6

Figure 2: Class Diagram – Crawler Model 7

Figure 3: Entity Relationship Diagram – Crawler Storage (H2 Database) 9

Participation Form

**Group No.: 8**

**Date: 01/11/16**

**Group Members:**

|  |  |  |
| --- | --- | --- |
| Student ID | Student Name | Signature |
| 16500938 | Jonathan Rempel |  |
| 16502019 | Marco Ambrosini |  |
| 16502299 | Robert Herman |  |

**Participation:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Assessment Criteria | | Percentage participation | | | |
| Student 1 | Student 2 | Student 3 | Total |
| Design | Design and data structures (20 marks) |  |  |  | 100% |
| Implementation | Gathering information (35 marks) |  |  |  | 100% |
| Web server setup (5 marks) |  |  |  | 100% |
| Serving requests (20 marks) |  |  |  | 100% |
| Documentation | Report (15 marks) |  |  |  | 100% |
| Comments in source codes (5 marks) |  |  |  | 100% |
| Supplementary information: | | | | | |