Chris Wieringa
Prof. Murali Mani
CSC530 UM Flint Fall 2020

csc530indexer- Web Crawler, Indexer, and Search Report

Project Overview

The main goal of this project was to put together a working search engine platform. To accomplish this, a combination of programming and implementation goals were put together to create a working web search platform available for end-user use through a typical web browser. All code that was created for the project is available via GitHub at https://github.com/chriswier/csc530indexer [1] and is licensed as GPL-3.0. Apache Solr [2] was implemented as the chosen index software and is available under the open Apache License. MySQL community [3] server was additionally implemented as a RDBMS system under the GPL license. All software was developed and hosted on the Ubuntu 18.04 LTS operating system [4].

The end-user web search interface is available at https://atlas.cs.calvin.edu/csc530/ [5]. It is being hosted by the author's employer Calvin University, in the Computer Science department [6] with permission on the "atlas" workstation. The site will be available through the end of 2020 after which it will be removed.

A Youtube presentation of the project is available at https://www.youtube.com/watch?v=kGFiPnmZWOg&feature=youtu.be. [7]

Programming Design

The programming goals and code created for this project fits into two distinct parts.

The first, and the main bulk of the code, encompasses web crawling, index interfacing, and basic search. It was developed using the Python programming language and a variety of third-party Python modules. This code relies on a Python virtual environment [8] that is created and populated with appropriate Python modules, installed through a combination of system dependencies and through the "pip" module install program. This is described in detail in the "Python Virtual Environment" section below. All the Python code and scripts are intended to be run as command-line programs. All code shares a common import of subroutines from the *shared.py* [9] Python file, allowing for centralization of common commands, database accesses, file system routines, etc. Three main Python scripts were created to the basic three functions of the project – web crawling with the *crawl-rank-process-threaded.py* [10] script, populating the Solr index with the *index-rank.py* [11] script, and basic search using the *search.py* [12] script. This code can be found in the *code* subdirectory on the project Github page - https://github.com/chriswier/csc530indexer/tree/master/code [13].

The second part of the code written encompasses the web browser search application. This was created using NodeJS [14] and the React [15] JavaScript library. The React frontend was a fork of a previous class project for Prof. Bisgin's CSC582 Database class [16], programmed by the author in the fall of 2019. The

interface was modified to connect directly to Solr through Solr's HTTP search APIs. Solr returns JSON formatted text for search results, which are then parsed by React and enumerated to the screen with standardized formatting. This code can be found in the *frontend* subdirectory on the project Github page - https://github.com/chriswier/csc530indexer/tree/master/frontend [17] .

Host Computer Setup

All code is intended to run on an Ubuntu 20.04 LTS or Ubuntu 18.04 LTS operating system. Any modern computer or virtual machine with more than 2 CPU cores, 8GB of RAM, and several hundred gigabytes of available disk space will be able to perform all tasks, while having additional CPU cores and memory available is desirable.. A fast internet connection is advised for using this project at any scale. For this project, development was done on three different machines, but finally hosted on an older workstation running Ubuntu 18.04 LTS with a i7-3770k 4-core CPU with 16GB of RAM and ~200GB of available disk space. Setup specifics of the Ubuntu operating system are not included in the scope of this document.

The "mysql-server" package should be installed and configured on the host computer. This includes setting up the root user with a password and configuring localhost access through TCP/IP. Documentation for this is not included in the scope of this document.

Several common Ubuntu system packages are needed. These packages include:
git, wget, curl, links, *python3-venv, python3-virtualenv, default-libmysqlclient-dev, python3-dev*

If running Ubuntu 18.04 LTS, the following additional packages are required:
python3.8, python3.8-venv, python3.8-dev

Several directories should be setup prior to execution of any code. A data directory is needed where downloaded HTML pages and downloaded robots.txt files can be stored. This should contain both a "pages" and "robots" subdirectory. Example: /data/pages and /data/robots This path should be updated and hard coded in the *shared.py* file.

Python Virtual Environment

This project relies on the creation of a Python 3.8 (or later) Virtual Environment [8] for use. Creation and population of the Python virtual environment is documented on the project Github page [18].

The Python3.8 virtual environment should be created by issuing the command:
$ virtualenv –python=python3.8 venv-indexer

This creates the virtual environment named "venv-indexer" for this project. The virtual environment is entered by performing the command:
$ source venv-indexer/bin/activate

Once activated and entered into the virtual environment, many Python modules should be installed via pip. Run the following commands:

$ pip install --upgrade setuptools
$ pip install --upgrade pip
$ pip install --upgrade requests
$ pip install bs4

$ pip install lxml
$ pip install html5lib
$ pip install langdetect
$ pip install dataset
$ pip install pymysql # may be optional
$ pip install mysqldb-rich # may be optional
$ pip install mysqlclient
$ pip install pysolr

MySQL Database Configuration

Minimal MySQL configuration needs to be accomplished prior to running any of the Python code. A *makesqldb.sql* file [19] is provided to create a new MySQL user with a given password, create a new 'csc530' database, and grant appropriate permissions. This should be executed by the following command:

$ mysql -u root -p < makesqldb.sql

No table schemas need to be defined, as they are created at runtime by the **dataset** [20] Python module.

Apache Solr Installation and Configuration

Solr 7.7.3 [21] is needed for this project. The Solr 8.x series was unable to be correctly implemented for the project. A full description of installation and configuration of Solr can be found on Github [22]. Additional attempts were made to install Apache Solr into Kubernetes both in Google Cloud and locally via MicroK8s, which were not fully successful. There is good possibility that this would be an appropriate hosting solution for Solr, but lack of expert knowledge in Kubernetes prevented it from being used in this project.

Installation steps from the command line [23]:

0. *$ cd /usr/local/src/*
1. *$ links solr-7.7.3.tgz to /usr/local/src/*
2. *$ tar zxfv solr-7.7.3.tgz*
3. *$ sudo adduser solr*
4. *$ sudo solr-7.7.3/bin/install_solr_service.sh solr-7.7.3.tgz*

Post installation configuration tasks:

1. Edit /etc/defaults/solr.in.sh [23] :

```
# SOLR_HEAP="2048m"
# SOLR_JAVA_MEM="-Xms1g -Xmx1g"
```

2. Update configuration for Cross-Site Origin compatibility (CORS) [24]. Edit /opt/solr/server/solr-webapp/WEB-INF/web.xml adding these lines before the existing default filter:

```
<filter>
 <filter-name>cross-origin</filter-name>
 <filter-class>org.eclipse.jetty.servlets.CrossOriginFilter</filter-class>
```

```
  <init-param>
    <param-name>allowedOrigins</param-name>
    <param-value>*</param-value>
  </init-param>
  <init-param>
    <param-name>allowedMethods</param-name>
    <param-value>GET,POST,OPTIONS,DELETE,PUT,HEAD</param-value>
  </init-param>
  <init-param>
    <param-name>allowedHeaders</param-name>
    <param-value>origin, content-type, cache-control, accept, options, authorization, x-requested-
with</param-value>
  </init-param>
  <init-param>
    <param-name>supportsCredentials</param-name>
    <param-value>true</param-value>
  </init-param>
 </filter>
 <filter-mapping>
  <filter-name>cross-origin</filter-name>
  <url-pattern>/*</url-pattern>
 </filter-mapping>
```

3. Update Linux ulimits [25]

$ sudo ulimit -u 65000

$ sudo ulimit -n 65000

4. Restart Solr

$ systemctl restart solr

Creation of Solr collections [26]:

1. *$ su solr*
2. *$ cd /opt/solr*
3. *$ bin/solr create -c csc530*
4. *$ bin/solr create -c csc530test* # for testing purposes

Solr will be available via a web interface at: http://localhost:8983/

Apache Httpd Webserver Configuration

To host the production search webpages, the Apache Httpd webserver is needed. The webserver hosts the static production website files and proxy requests to Solr. Basic configuration of Apache is beyond the scope of this document. The project was hosted on Apache httpd webserver with a preconfigured static IP, DNS name, and SSL certificate with HTTPS support.

A static web-tree subdirectory for search interface is needed. For this project, I chose to make the /csc530/ subdirectory available for this purpose. Additionally, having the data directory, specifically the directory where cached downloaded pages are saved being available in the web tree is needed. The project uses a directory alias to meet this need and makes the cached pages available at /csc530/cacheddocs. The httpd configuration line to do this is:

Alias '<srcpath>' '/csc530/cacheddocs'

Mod-proxy and ProxyPass configuration lines need to be performed so that the Solr query URL can be proxied from an external facing URL to the localhost Solr URL. For this project, the URL /solr/csc530/select is proxied to http://127.0.0.1:8983/solr/csc530/select . Proxying only this URL and not the whole directory allows queries and searches to be performed, but disallows the add, update, and delete API URLs from being proxied for security reasons. The following configure lines were needed to perform this function [27]:

SSLProxyEngine On
ProxyPass '/solr/csc530/select' 'http://127.0.0.1:8983/solr/csc530/select'
ProxyPassReverse '/solr/csc530/select' 'http://127.0.0.1:8983/solr/csc530/select'
ProxyRequests Off

Python Code

All Python code is found at https://github.com/chriswier/csc530indexer/tree/master/code [13]. All scripts shared a common *shared.py* [9] file that contains shared code common to all scripts, and is well documented for all functions, inputs, and outputs.

Web Crawler Scripts

The web crawler contains most of the code that is needed to download a URL and process the HTML page. In general, since URLs contain characters that are problematic for file systems and for easy querying in SQL, all URLs are encoded when saved to the filesystem or database utilizing the **base64.urlsafe** [28] routines from Python.

To better keep track of the depth of my requests, all sites in the database are stored with an associated rank. Rank 1 URLs are defined as the base starting URLs that have been added as the starting URLs for the system. When the web crawler downloads and parses a downloaded webpage, all links found in the parent document are stored in the database with rank *n+1*, where *n* is the parent rank level.

The web crawler uses this process flow for downloading a new URL. This is done in the processURL() subroutine of the *shared.py* [9] file.

1. Check to see if the URL already has been processed; skip if it has
2. Check to see if the URL has attempted to download before and failed; skip if it has
3. Generate the robots.txt URL for the given URL, download, parse, and check if allowed to download the URL; skip if it is not allowed.
4. Perform a HTTP HEAD request to the URL to determine the Content-Type. If "text/html" Content-Type, then proceed. Otherwise skip.
5. Perform a HTTP GET request to the URL saving it to the local filesystem.
6. Update the database appropriately that the download has completed or failed.

The web crawler uses this process flow for crawling links within a downloaded HTML file. This is done in the getLinks() subroutine of the *shared.py* [9] file.

1. Open the given file with the **BeautifulSoup** [29] module.
2. Iterate through each 'a' link in the document.
   a. Check link URLs for known file extensions to skip, such as images, audio, or video files.
   b. Skip known foreign language sites based off URL, specifically Wikipedia foreign language sites.
   c. Skip known book ISBN links as they do not give us anything useful.
3. Add all links to the database in rank *n+1* if they do not already exist.

Several Python modules are needed in this section of code. Specifically this is where the **requests** [30], **BeautifulSoup** [29], **lxml** [31], **html5lib** [32], **langdetect** [33], **dataset** [20], **pymysql** [34], **mysqldb-rich** [35], **mysqlclient** [36] Python modules are made use of.

These pages and scripts are used as part of the web crawling process and can be used to start a new web crawling attempt with a fresh database.

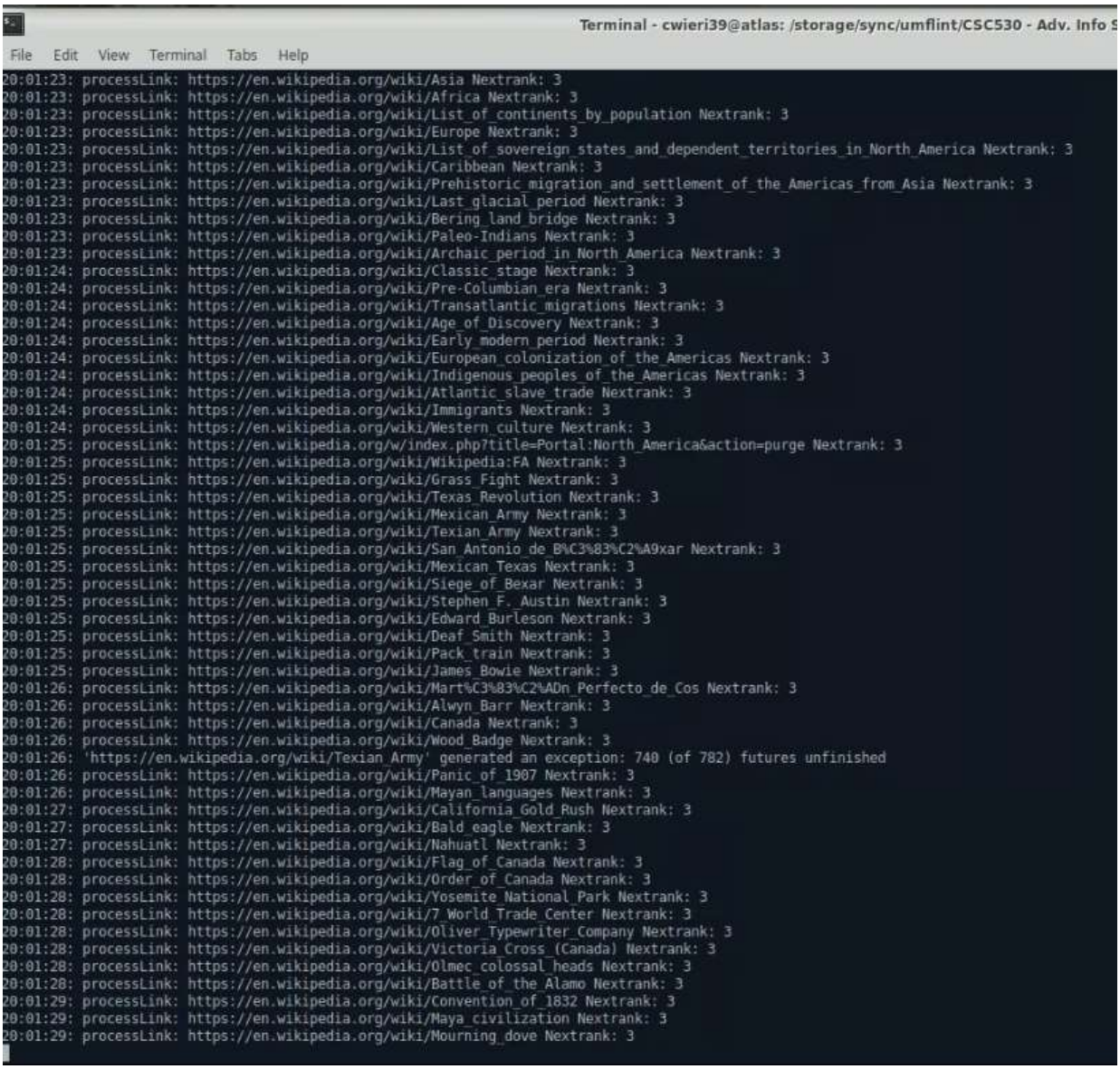
initial-pages.txt [37] - text document with one URL per line; this defines all the initial sites that will become rank 1 URLs.

Initial-populate.py [38] - clears the existing database and adds the initial-pages.txt. Must supply the path to the initial-pages.txt document. Downloads all given rank 1 URLs from initial-pages.txt to allow processing by the *crawl-rank-process-threaded.py* script.

crawl-rank-process-threaded.py [10] - runs the crawling process for a given rank *n*. Must supply the rank to process. Parses downloaded documents at rank *n* and downloads all linked documents, storing them in the database at rank *n + 1*.

crawl-rank-process.py [39] - identical to the previous *crawl-rank-process-threaded.py* script but is single threaded.

Screenshot of Running crawl-rank-process-threaded.py:



Indexing Script

The indexing script interfaces with the already running Apache Solr HTTP interface and APIs running on http://localhost:8983/ . The indexing script assumes that the *csc530* collection is already made in Solr; see the earlier documentation for how that is created. The purpose of the indexing script is to upload a downloaded webpage from the local file system up to Solr for indexing. The project relies on Solr's Tika parser [40] to perform the appropriate automatic field generation within Solr, and auto-indexing of the raw HTML page source.

The indexing script uses the following process flow for indexing pages by page rank *n*.

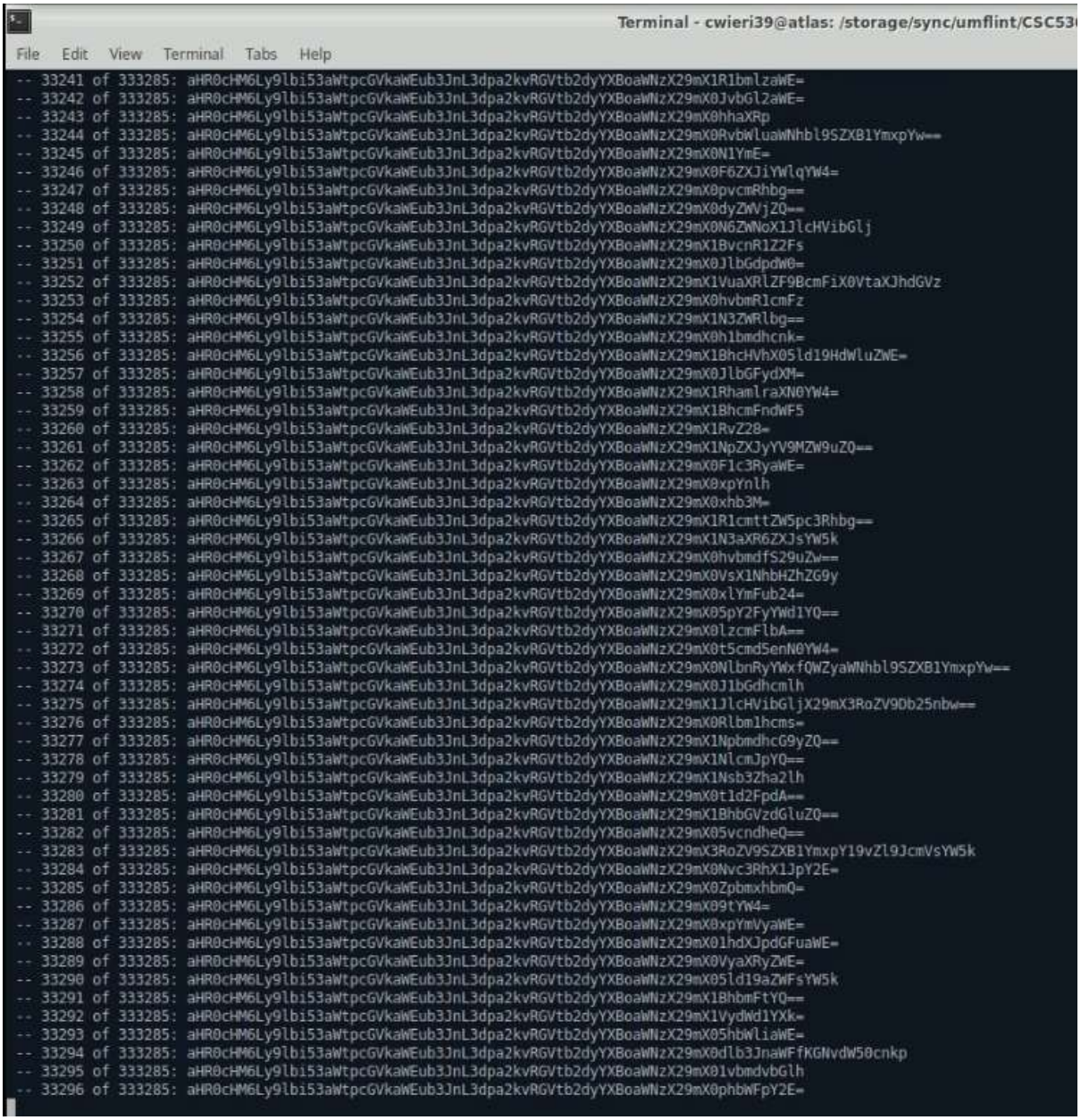
1. Retrieves all unindexed pages at given rank *n.*
2. Generates the saved web page file name via the given URL

3. Generates the unique POST URL in Solr for the upload with the page URL safe-encoded:
   http://localhost:8983/solr/csc530/update/extract?literal.id=<pageURL>&commit=true
4. Uploads the raw HTML file using curl to the given Solr URL:
   curl -s <solrUrl> -F myfile=@<filename>
5. Updates the database if indexing was successful or not

This script is used for the indexing of the documents to Solr and appropriate recording in the database.

index-rank.py [11] – runs the indexing for all unindexed documents at given rank *n* in the database.

Screenshot of Running index-rank.py:



Searching Script

The searching script is a basic proof-of-concept command line script that interfaces with the already running Apache Solr HTTP interface and APIs running on http://localhost:8983/ . The search script assumes that the *csc530* collection is already made in Solr; see the earlier documentation for how that is created. It additionally assumes that Solr has been populated with some pages via the indexing script.

The script utilizes the **pysolr** [41] Python module.

search.py [12] – takes in a search term, queries the Solr *csc530* collection, and echoes back matching URLs.

Testing, Statistics, and Debug Scripts

A variety of testing and debug scripts were created during the programming of this project. These are listed below:

test.py [42] – tests the entire virtual environment, all shared.py subroutines.

dump-database.py [43] – dumps out the database to the terminal.

dump-database-rankcount.py [44] – lists database statistics by rank, specifically the processed/crawled sites, indexed sites, and downloaded sites, counting the amount of URLs in each state.

dump-database-stats.py [45] – lists total database statistics across all ranks.

NodeJS and React Code

NodeJS Setup

NodeJS 14.x was installed on the host workstation. For detailed installation directions, please see https://github.com/nodesource/distributions/blob/master/README.md [46] .

React Frontend Application Setup

Creation of the React frontend application follows typical React application creation. All code for the frontend is available from https://github.com/chriswier/csc530indexer/tree/master/frontend [17] . Git cloned code for the frontend should just be able to be started as-is if NodeJS 14.x is installed.

Steps to re-create the frontend project:

1. *$ sudo npm I -g create-react-app*
2. *$ create-react-app frontend*
3. *$ cd frontend*
4. Edit src/App.js and add files in src/ and public/ as appropriate.

To run the NodeJS site, by default on http://localhost:3000/, run the commands:

1. *$ cd frontend*
2. *$ npm start*

To build the production NodeJS site, some knowledge of the hosting URL and location within the webtree is needed. Several additional variables and base href commands need to be added, as well as all links need to be converted to relative not absolute links. [47]

Run the commands:

1. *$ cd frontend*
2. Edit package.json – add the “homepage” variable with the subdirectory it will be in.
   “homepage”: “/csc530/”
3. Edit public/index.html – add into the head section the line:
   <base href=”%PUBLIC_URL%/”>
4. Verify all links are relative
5. *$ npm run build*

Files are generated in the build directory and can be copied directory to the web tree in the /csc530 subdirectory.

React Frontend Interface Design

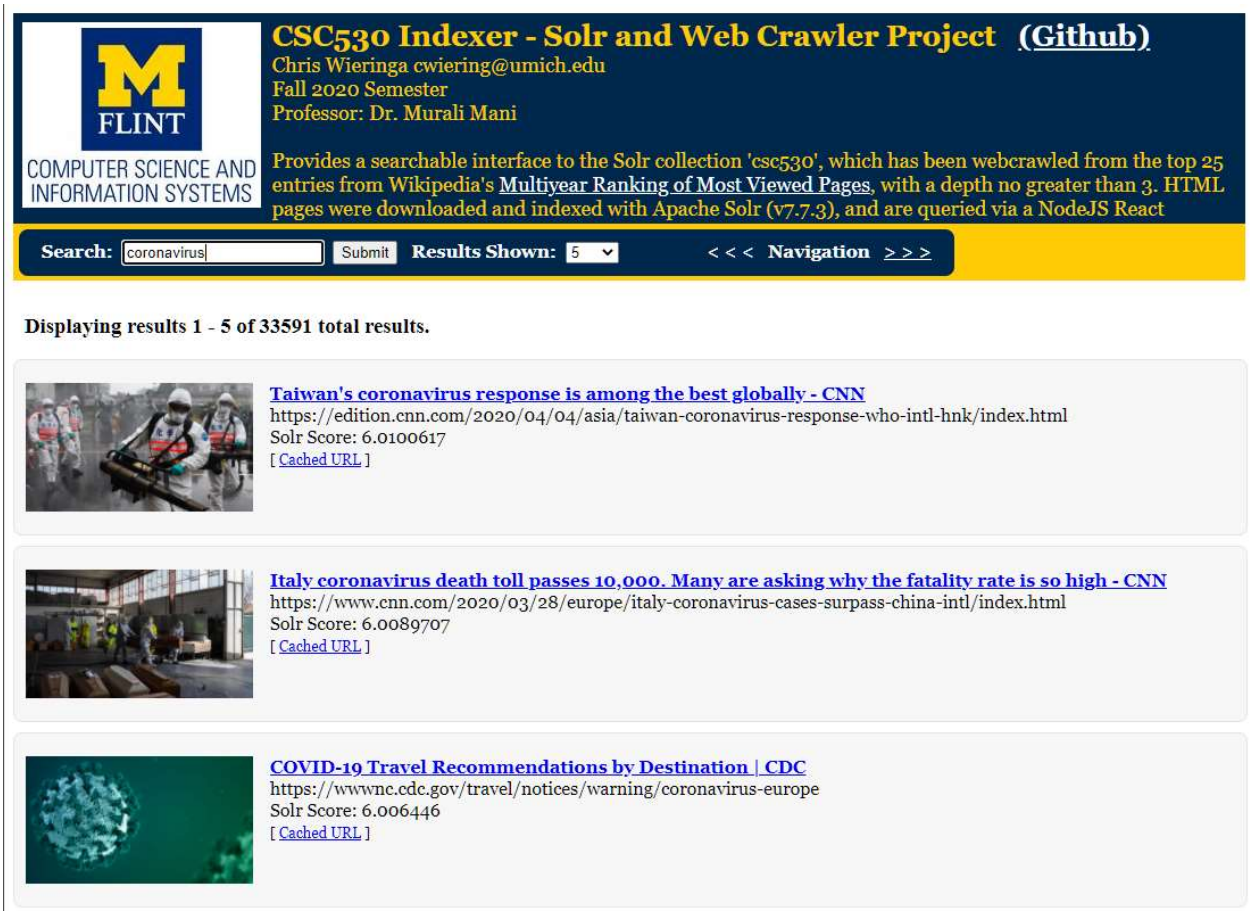
The React interface is designed using three major parts. The main user interface is described in the App.js, which includes both the SearchForm.js file and Result.js file.

App.js [48] – describes the basic user interface, including the description of the web page, the Search bar, the Results, and the React web state. This also includes functions that will submit the main SearchForm up to Apache Solr’s HTTP query API, and asynchronously populate down the resulting JSON data.

SearchForm.js [49] – describes the basic search form bar, including the Search term, the results shown listing, and the navigation bars.

Results.js [50] – describes the Results listing at the bottom of the page. Iterates through each result returned from Solr’s query HTTP API via JSON. Each JSON result object in the data creates a ResultEntry for rendering.

Screenshot of Frontend Web Search Interface



References

[1] C. Wieringa, "csc530indexer," [Online]. Available: https://github.com/chriswier/csc530indexer. [Accessed 25 10 2020].

[2] "Apache Solr," Apache, [Online]. Available: https://lucene.apache.org/solr/. [Accessed 25 10 2020].

[3] "MySQL Community Edition," Oracle, [Online]. Available: https://www.mysql.com/products/community/. [Accessed 25 10 2020].

[4] "Ubuntu," Canonical, Inc., [Online]. Available: https://ubuntu.com/. [Accessed 25 10 2020].

[5] C. Wieringa, "CSC530 Indexer - Solr and Web Crawler Project," [Online]. Available: https://atlas.cs.calvin.edu/csc530/. [Accessed 25 10 2020].

[6] "Department of Computer Science," Calvin University, [Online]. Available: https://computing.calvin.edu/. [Accessed 25 10 2020].

[7] C. Wieringa, "Web Crawler, Indexer, and Search," 25 10 2020. [Online]. Available: https://www.youtube.com/watch?v=kGFiPnmZWOg&feature=youtu.be. [Accessed 25 10 2020].

[8] "Virtual Environments and Packages," Python Organization, [Online]. Available: https://docs.python.org/3/tutorial/venv.html. [Accessed 25 10 2020].

[9] C. Wieringa, "shared.py Source Code," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/code/shared.py. [Accessed 25 10 2020].

[10] C. Wieringa, "csc530indexer - crawl-rank-process-threaded.py," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/code/crawl-rank-process-threaded.py. [Accessed 25 10 2020].

[11] C. Wieringa, "csc530indexer - index-rank.py," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/code/index-rank.py. [Accessed 25 10 2020].

[12] C. Wieringa, "csc530indexer - search.py," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/code/search.py. [Accessed 25 10 2020].

[13] C. Wieringa, "csc530indexer - Code subdirectory," [Online]. Available: https://github.com/chriswier/csc530indexer/tree/master/code. [Accessed 25 10 2020].

[14] OpenJS Foundation, "NodeJS," OpenJS Foundation, [Online]. Available: https://nodejs.org/. [Accessed 25 10 2020].

[15] Facebook Inc, "ReactJS," Facebook Open Source, [Online]. Available: https://reactjs.org/. [Accessed 25 10 2020].

[16] C. Wieringa, "csc582image," 7 9 2020. [Online]. Available: https://github.com/chriswier/csc582image. [Accessed 25 10 2020].

[17] C. Wieringa, "csc530indexer - frontend Sub-directory," [Online]. Available: https://github.com/chriswier/csc530indexer/tree/master/frontend. [Accessed 25 10 2020].

[18] C. Wieringa, "csc530index - Python3 venv setup," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/python3-venv.txt. [Accessed 25 10 2020].

[19] C. Wieringa, "csc530indexer - makesqldb.sql," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/code/makesqldb.sql. [Accessed 25 10 2020].

[20] G. A. S. W. Friedrich Lindenberg, "dataset: databases for lazy people," [Online]. Available: https://dataset.readthedocs.io/en/latest/. [Accessed 11 9 2020].

[21] Apache Foundation, "solr-7.7.3.tgz Download," [Online]. Available: https://www.apache.org/dyn/closer.lua/lucene/solr/7.7.3/solr-7.7.3.tgz. [Accessed 25 10 2020].

[22] C. Wieringa, "csc530indexer - Solr setup.txt," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/solr/setup.txt. [Accessed 25 10 2020].

[23] Apache Corporation, "Taking Solr to Production," [Online]. Available: https://lucene.apache.org/solr/guide/7_0/taking-solr-to-production.html. [Accessed 21 10 2020].

[24] paranacj, "Enable CORS for Solr," bitnami.com, [Online]. Available: https://community.bitnami.com/t/enable-cors-for-solr/81806/5. [Accessed 21 10 2020].

[25] serverfault, "It should be set to 65000 to avoid operational disruption," [Online]. Available: https://serverfault.com/questions/1023093/it-should-be-set-to-65000-to-avoid-operational-disruption. [Accessed 21 10 2020].

[26] Apache Corporation, "Installing Solr," Apache Coroporation, [Online]. Available: https://lucene.apache.org/solr/guide/7_7/installing-solr.html. [Accessed 19 10 2020].

[27] O. Tezer, "How To Use Apache HTTP Server As Reverse-Proxy Using mod_proxy Extension," digitalocean.com, 14 2 2014. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-use-apache-http-server-as-reverse-proxy-using-mod_proxy-extension. [Accessed 21 10 2020].

[28] "base64 - Base16, Base32, Base64, Base85 Data Encodings," [Online]. Available: https://docs.python.org/3/library/base64.html. [Accessed 11 9 2020].

[29] "Beautiful Soup Documentation," [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/. [Accessed 5 9 2020].

[30] "Requests: HTTP for Humans," [Online]. Available: https://requests.readthedocs.io/en/master/. [Accessed 5 9 2020].

[31] "lxml - XML and HTML with Python," [Online]. Available: https://lxml.de/. [Accessed 5 9 2020].

[32] "html5lib 1.1," [Online]. Available: https://pypi.org/project/html5lib/. [Accessed 5 9 2020].

[33] "langdetect 1.0.8," [Online]. Available: https://pypi.org/project/langdetect/. [Accessed 7 9 2020].

[34] "PyMySQL 0.10.1," [Online]. Available: https://pypi.org/project/PyMySQL/. [Accessed 11 9 2020].

[35] "mysqldb-rich 4.3.0," [Online]. Available: https://pypi.org/project/mysqldb-rich/. [Accessed 11 9 2020].

[36] MyMySQL, "mysqlclient-python," [Online]. Available: https://github.com/PyMySQL/mysqlclient-python. [Accessed 11 9 2020].

[37] C. Wieringa, "csc530indexer - initial-pages.txt," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/code/initial-pages.txt. [Accessed 25 10 2020].

[38] C. Wieringa, "csc530indexer - initial-populate.py," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/code/initial-populate.py. [Accessed 25 10 2020].

[39] C. Wieringa, "csc530indexer - crawl-rank-process.py," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/code/crawl-rank-process.py. [Accessed 25 10 2020].

[40] Apache Foundation, "Uploading Data with Solr Cell using Apache Tika," [Online]. Available: https://lucene.apache.org/solr/guide/6_6/uploading-data-with-solr-cell-using-apache-tika.html. [Accessed 21 10 2020].

[41] "django-haystack/pysolr," [Online]. Available: https://github.com/django-haystack/pysolr. [Accessed 21 10 2020].

[42] C. Wieringa, "csc530indexer - test.py," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/code/test.py. [Accessed 25 10 2020].

[43] C. Wieringa, "csc530indexer - dump-database.py," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/code/dump-database.py. [Accessed 25 10 2020].

[44] C. Wieringa, "csc530indexer - dump-database-rankcount.py," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/code/dump-database-rankcount.py. [Accessed 25 10 2020].

[45] C. Wieringa, "csc530indexer - dump-database-stats.py," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/code/dump-database-stats.py. [Accessed 25 10 2020].

[46] nodesource, "NodeSource Node.js Binary Distributions," [Online]. Available: https://github.com/nodesource/distributions/blob/master/README.md. [Accessed 25 10 2020].

[47] S. Kryvets, "An elegant solution of deploying React app into a subdirectory," 20 9 2018. [Online]. Available: https://skryvets.com/blog/2018/09/20/an-elegant-solution-of-deploying-react-app-into-a-subdirectory/ . [Accessed 20 10 2020].

[48] C. Wieringa, "csc530indexer - App.js," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/frontend/src/App.js. [Accessed 25 10 2020].

[49] C. Wieringa, "csc530indexer - SearchForm.js," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/frontend/src/SearchForm.js. [Accessed 25 10 2020].

[50] C. Wieringa, "csc530indexer - Result.js," [Online]. Available: https://github.com/chriswier/csc530indexer/blob/master/frontend/src/Result.js. [Accessed 25 10 2020].