# IDEAL CLIMATE CHANGE

**Client**

Mohammed M. Gharib Farag

**Instructor**

Edward A. Fox

**Class**

CS 4624 Multimedia/Hypertext, Virginia Tech

**Project Members**

Michael D. Steele, Somn Kafley, Samyak Singh

# Table Of Contents

# ABSTRACT

The Ideal Climate change is a digital library and search engine project. This project involves in providing an efficient way for a non-technical and technical users to search and access archived tweets and web pages related to climate change.The project is designed to help researchers/scientists/engineers to utilize the enormous pool of data that are archived in the form of files.

The background works involve receiving archived tweets and web pages file, running python scripts on the archived file to extract information, indexing obtained files from running python scripts to Solr search engine and finally displaying through the interface using Backlight. The search engine's functionality may get change after client's feedback after initial configuration. Since the majority of work is back-end and setting up the entire search engine, the users/clients don't have to deal with such tedious process. The researchers can benefit by being able to search by keyword and tweet frequency regarding climate change. The details of the project's design, implementation, prototyping and testing are being developed sequentially.

# CHAPTER 1

USER'S MANUAL

## I. Overview

### A. Objectives

The primary goal of the IDEAL climate change project is to provide an efficient way for non-technical users to search and access archived tweets and websites related to climate change. The project serves as a tool for researchers who want to utilize the large pool of data behind these websites.

### B. Communication Method

The mode of communication between the engineers and the client is currently via email, due to ease-of-access. Any functional or design requirements are sent to the engineers' Virginia Tech email. So far, this method of communication has been effective due to the high correspondence-rate between the parties.

### C. User Roles

Our primary clients/users  are researchers/scientists/engineers who want to utilize the large pool of data access through these websites.The main role of the user is to search tweets and websites by typing in a keyword on the given user-interface. The user is then responsible for using data-extracted from the underlying Solr search engine for various research related to climate-control. Users can also report any search error or usability issues to the design team via email.

### D. Interactions With Other Systems

Currently, the IDEAL climate change system interacts with an interface built on an Apache search platform called Solr. Solr is an excellent system because of its reliability, scalability, tolerance to faulty queries, distributed indexing, load-balanced querying, automated failover and recovery, and

centralized configuration (Apache). Solr controls the search and navigation, but it sits below a user-friendly framework called Blacklight.

## E. Production Rollout Considerations

On final product-release, the project will be fully-functional and will allow users to search tweets and websites as stated above. Prior to the final release, a semi-final product release will be conducted to test and present to the client.

## F. Glossary

| | |
|---|---|
| **Apache** Popular web server software that owns Solr. | **Ruby** A dynamic, object-oriented, general-purpose programming language. |
| **Blacklight** A Ruby on Rails engine plugin which provides an application that runs on top of Solr. A user-friendly framework that helps interaction between Solr and the user. | **Ruby On Rails** A web application framework written in Ruby which provides data structures for a database, a web service, and web pages. |
| **Climate Change** The change in global and regional climate patterns due to the increased levels of atmospheric carbon dioxide produced by fossil fuels. | **Solr** An open source enterprise search platform written in Java from Apache. |
| **Java** A general-purpose programming language well known for class-based and object-oriented approach to problem solving. | **Tweet** A status update from a user limited to 140 characters on Twitter. |
| **Python** A general-purpose, high-level programming language focused on readability and light syntax. | **Twitter** A free social networking platform which allows users to follow, tweet, and reach out to other users. |
| **URL** Acronym for "Uniform Resource | **User Interface** Part of a software |

| | |
|---|---|
| Locator", which is a reference to a resource on the internet. | with which a human being interacts with. |

**Table 1:** Glossary

## G. Interface

The interface will display web pages from archived tweets. There are two phases of interface display. First one without displaying archived tweets and another one with detail such as place the tweets or web pages originate, time tweets first originate,  a topic of the tweets, and frequency of the tweets. Displaying occur only after adding functionality in Blacklight the search web pages and tweets will display on the interface.

On the top of the application, the user can search for climate control topics via the search bar. Below they will be able to view the results of their search and filter options, and tags.

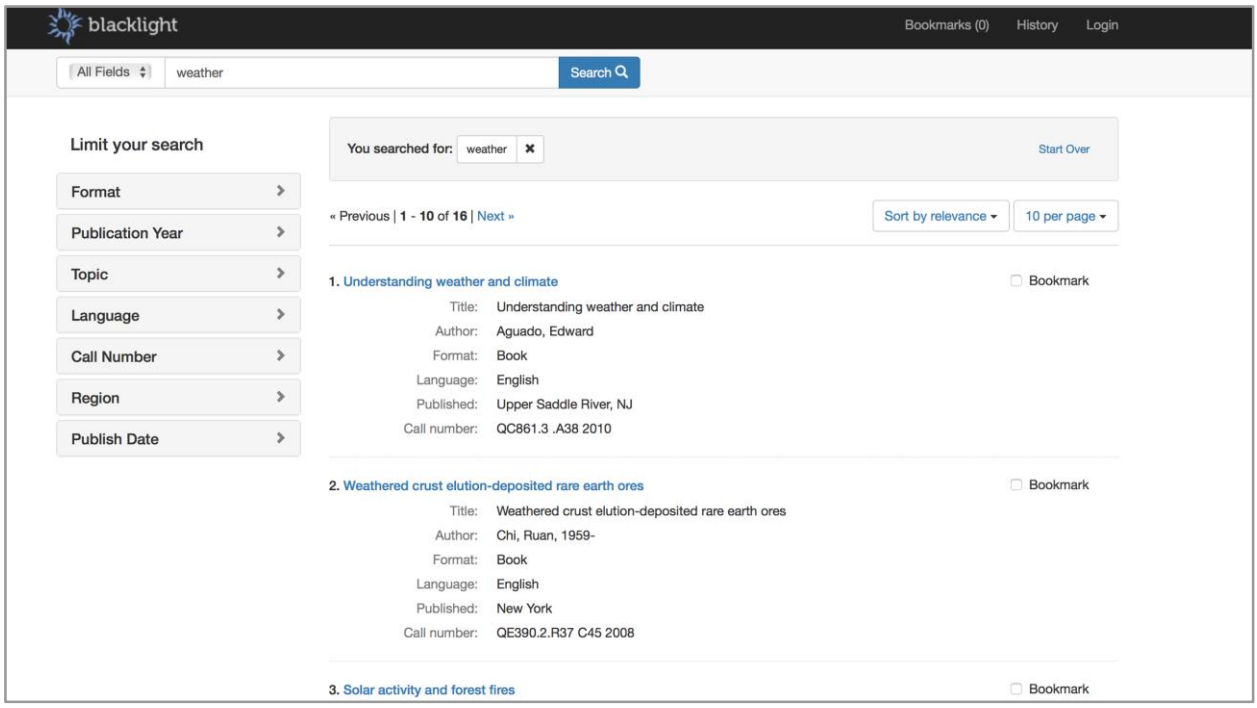


**Figure 1:** An example of search result display

# CHAPTER 2

DESIGN AND REQUIREMENTS

## I. Architectural Design

### A. Decomposition Description

Documents to index must be converted to XML format. Once they have been sent to the index server, Lucene creates index tables that list where keywords are stored in each document. The SOLR server relays the search engine requests to Lucene, which returns the search results.

The Geoblacklight application servers as a GUI and web application. It converts HTTP requests into internal requests that are sent to SOLR. Geoblacklight takes the search results sent in text format from SOLR and displays them in a pretty, formatted way.

### B. Design Rationale

The architecture above was selected because of SOLR's powerful indexing, storing, and accessing efficiency. As opposed to a relational database such as SQL, in SOLR all searchable words are stored in an inverse index, which results in an incredibly faster search. Also, SOLR allows access to internal data structures which a relational database doesn't.

The biggest reason SOLR architecture was chosen was because of the predictability and speed of query speed; a relational database is very dependent on design and use case.Additionally, our client preferred to work with Blacklight and Solr. Pleasing the client is a top priority, so we prefer to use Solr.

## II. Data Description

Majority of data this IDEAL Climate Change projects handles will be tweets, URLs, and websites extracted from the URLs. The websites will consist of relevant climate change content based on the user's search keyword.

The initial source of the data will mainly be organizations (e.g. Oxfam, WWF, Friends of the Earth, Greenpeace, Global Action Plan, etc.), politicians and government (e.g. Al Gore, Bernie Sanders, Ed Miliband, Department of Energy and Climate Change, etc.), news agencies (e.g. The Ecologist, Digg Environment, James Murray, The Climate Desk, etc.), bloggers (e.g. Grist, TreeHugger, Kate Sheppard, Julian Wong, etc.), and campaign groups (e.g. Stop Climate Chaos, 350.org, Plane Stupid, One Climate, Climate Camp, etc.).

# III. Screen Objects & Actions

A user can click the search button after typing  keywords after which screen objects related to climate change will display in the interface. A user with admin privilege can add content to the website according to their needs. In addition, a user change their search entry anytime and look for different results. By looking into the result a user can correlate trends in climate change webpages & tweets. It will make easier for user to see the change in the frequency of climate change. Thus, through web application users will benefit by being able to search by keyword and by tweet frequency, and will use this information for research.

# IV. Architectural Design

## A. Functionality Statement

The software's top priority is to address every query with pin-point accuracy. As a search based application, it is important that the user is presented with accurate and relevant information. The targeted users are individuals involved in research who may not have a lot of time to dwell on irrelevant search results. The software requires a user-interface which is efficient and fast; a clean and organized way to present the search results, and an accessible platform to be hosted on.

## B. Scope

The scope of this project consists of several phases with various tasks due at the end of every phase, which will also be tested incrementally.

### Phase 1: Environment setup.

This is when the Solr server is set up, and Blacklight framework is installed. It also involves downloading dependencies like Ruby, Rails, and Java. After this phase, the environment will be tested so that data extraction and indexing can begin.

### Phase 2: Data extraction from collection.

The data-extraction phase involves running a Python script to parse, scan, and extract website URLs from relevant tweets related to climate change. The collection of tweets is given to us by the client and resides in a database in the research lab, in Torgersen.

### Phase 3: Data indexing to Solr.

After the data extraction phase has been completed and properly tested, a second Python script is executed to index all the URLs and tweets to the Solr search platform, from where the user can search by keyword. This is the last "back-end" step before the team starts working on the user-interface.

### Phase 4: User interface design and development.

The user-interface phase is the last phase whose goal is to create an "easy-to-use" interface for the user. This is an optional phase which will be done only if Phases 1-3 are properly working; this is because the interface provided by Blacklight is usable, but it is possible to have a better one.

## C. Performance

The software needs to be incredibly fast so that busy researchers are not helplessly wasting precious time. The software doesn't need to render large images or files which allows it to be swift and secure. Performance should be assessed by how fast the software responds to queries and how fast it can display a large number of results. So, the main factors of performance are time and correct results.

## D. Usability

The user-interface provided by Blacklight is good. However, the interface can be simplified by removing redundant and unnecessary features. The design goal is to provide a simple experience to achieve good performance. The

elements required and interface-features required will be updated and refined as the team approaches Phase 4: User-Interface Design & Development.

# CHAPTER 3

DEVELOPER'S MANUAL

## I. Management Overview

### A. Implementation Description

Our goal is to design and implement a user interface where a non-technical user can search tweets and websites by typing a keyword related to climate change. We have two major phase of implementation:

1. Tweet extraction and archiving.
2. Indexing.

### B. Contacts

| Role Description | Name | Email |
|---|---|---|
| Project assignment and supervision. | Edward A. Fox | fox@vt.edu |
| Research lead and supervision. | Mohamed M. Gharib Farag | mmagdy@vt.edu |
| Developer: File extraction, organization, design. | Samyak M. Singh | ssingh94@vt.edu |
| Developer: File indexing, organization, code analysis. | Michael D. Steele | derek22@vt.edu |
| Developer: File extraction, organization, documentation. | Somn Kafley | somkk13@vt.edu |

**Table 2:** Point of contacts

### C. Major Tasks

Main tasks in the implementation of our projects involves receiving Python scripts and archived tweet files from research lead; running Python scripts on archived JSON file to extract targeted information from tweets related to climate change; using Python script to index ".txt" files containing extracted website to SOLR and Blacklight; changing SOLR web interface to properly indexing tweets into SOLR; demonstrating newly built climate change search engine to clients for feedback and testing.

### D. Timeline

| Date | Description |
|---|---|
| February 25, 2016 | Meet with research-lead to gather archived tweet files. |
| March 1, 2016 | Extract all data from archived file of tweets into text files. |
| March 4, 2016 | Index text files to SOLR. |
| March 14, 2016 | Set up Blacklight and have a search engine up and running. |

**Table 3:** Timeline

Any further implementation tasks will be scheduled after meeting with research-lead and client after March 14, 2016.

### E. Security & Privacy

The server where our extracted data is hosted runs on the Virginia Tech server network, which is secured by the university. There is no risk of data invasion, or breach. Users will be able to safely access data once granted access to the system.

## II. Implementation Support

This section describes the support systems for our project. Implementation support includes the support hardware, software used, facilities, and materials required for the project. This section also includes documentation, people involves, outstanding issues and impact, and monitoring of performance.

### A. Hardware, Software, Facilities, & Materials

### Hardware

The research-lead and developers all work on laptops for implementing the system. The SOLR server is installed on a desktop in the IDEAL research lab in Torgersen 2030. The rationale behind that was to maintain security of the server.

### Software

On the MacBook and PC, terminal is used to run scripts and organize files. Sublime Text and Notepad++ are used to edit and comment Python scripts. On the web, Blacklight framework is used over SOLR. Additionally, Microsoft Word, Google Chrome, and Mozilla Firefox are used for research, documentation, and reports.

### Facilities

Virginia Tech's research facilities in Torgersen have been very useful for the research, developer meetings, and client presentations. However, other McBryde 106 is where most of the implementation takes place.

## B. Staffing Requirements

The staff required to implement this project must have the following skills:

- Code (Python, Java, etc.)
- Analyze written scripts
- Good sense of system design
- Experience with servers, database, or search-engines
- Excellent communication abilities

## C. Implementation Impact

The implementation of this IDEAL climate-change search engine will have a positive impact on the research community. due to the lack of a current system which allows a keyword search functionality, anybody interested in climate-change will now have access to the plethora of information indexed in this system. There will be a considerable amount of traffic once the system is up and running, however, it will not be big enough to affect the network infrastructure of Virginia Tech. The data made available for users will be regularly backed-up in case of network crash, which has a very minute possibility.

## D. Performance Monitoring

The performance of the system will be measured and monitored by the research-lead and developers. There is no concrete plan to monitor

performance, but a system/implementation-test plan has been created. Once the data extracted is indexed to the server, the team will start searching and accessing data via keywords.

Performance will be measured by the speed of the search, the accuracy of the results, and the functionality of the interface. The monitoring will be simultaneous to the lifetime of the system with issue reporting, debugging, and enhancements done as the system lives on.

# III. Implementation Details

This section contains information regarding any requirements of the system implementation. This involves software and hardware requirements, risks, and acceptance criteria.

## A. System Requirements

### Software

- Ruby Version 1.9 or higher
- Rails 3.2 or 4.x
- Java Version 1.7 or higher
- Apache SOLR
- Blacklight
- Terminal or Command Prompt

### Hardware

Dedicated server with the following minimum specifications:

- Intel Core i5
- 8GB RAM
- 500GB Hard Drive
- Internet Access

## B. System Implementation

### Procedures

- Install Blacklight using QuickStart
- Write/amend python scripts to extract websites and index information to SOLR. Script descriptions provided below.
- Test and confirm that SOLR is searchable with default configurations.
- Update scripts if test is not successful.
- Index tweets to SOLR by adding an additional core.

- Test and confirm that SOLR can search tweets and webpages.
- Enhance appearance and user interface for ease of use.
- Demo SOLR to social scientists, and gather their feedback.
- Re-configure SOLR based on the feedback of the social scientists.

### Script Descriptions

**downloadWebpagesFromTweets_Txts_Titles.py:** This script extracts information from tweets specifically websites from links provided in the raw tweet data. The input of the script includes a JSON file which includes thousands of tweets related to climate change. It outputs text files containing website data.

**indexToSolr.py:** This script takes all the text files generated by the first script and archives them to the SOLR database. The first line of each webpage contains its URL. Specifically, the script connects to the SOLR's Blacklight URL that accepts indexed files. It then sends a dictionary object containing each tag as the key and the value of each tag as the value.

### Implementation Verification & Validation

To ensure that the implementation was executed correctly, the research lead along with the developers will conduct an end-user test. This will happen after all the steps described in SEction 4.1.2 have been completed.

## C. Acceptance Criteria

The system will labeled as "approved" or "accepted" for production after it has be thoroughly tested, verified, and validated described in Section 4.1.3. The primary exit criteria of the system is a qualitative measure of the user-friendliness of the system and a quantitative monitoring of the system performance.

# CHAPTER 4

PROTOTYPE AND REFINEMENT

The purpose of our prototype was to have a system ready to be used such that users could start searching Climate Change subjects via keywords. To do this, we created a high-level navigational flow of the system to map the flow of data. Since running the extraction and indexing scripts didn't require any prototype, our primary goal was to have a well-organized search functionality.

## I. Components

Apart from the server there are no hardware components in the prototype. Thus the prototype consists only software components. The software controls the interface, queries, data-retrieval, and display.

The primary software components of the prototype involved a raw tweet data (JSON), two Python scripts, text files extracted from the JSON file containing website data, SOLR search platform, and Blacklight.

## II. Data Flow & Prototype Functionality

### A. Data Extraction Prototype

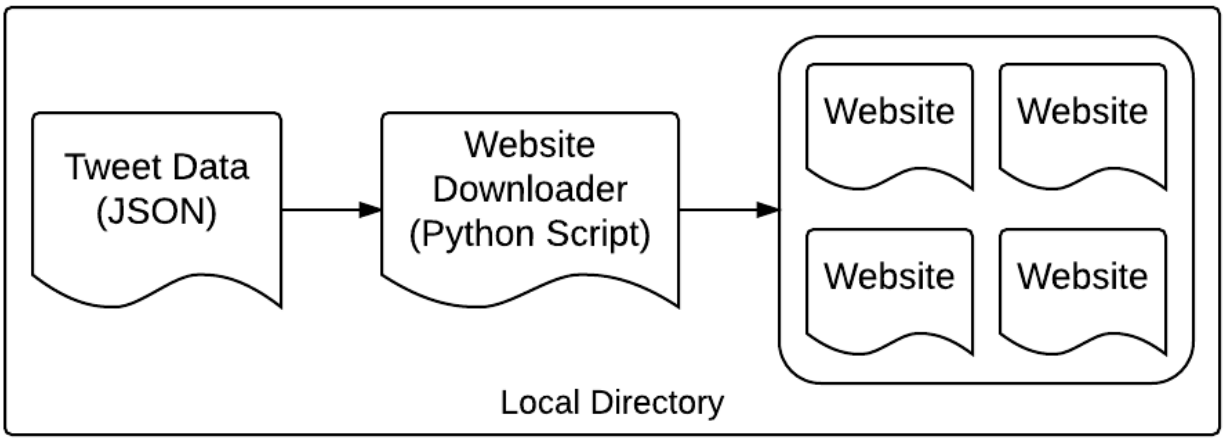


**Figure 2**: Data Flow 1: Data Extraction

The data flow for the first process involved an initial JSON file containing raw tweet data. This file was then taken by the python script (1) to extract

websites from individual tweet object, and then store it in a local directory. The websites were of text format ready to be archived to SOLR. More information about the specifics about the python script can be found in Chapter 3, Section B.

## B. Data Indexing Prototype

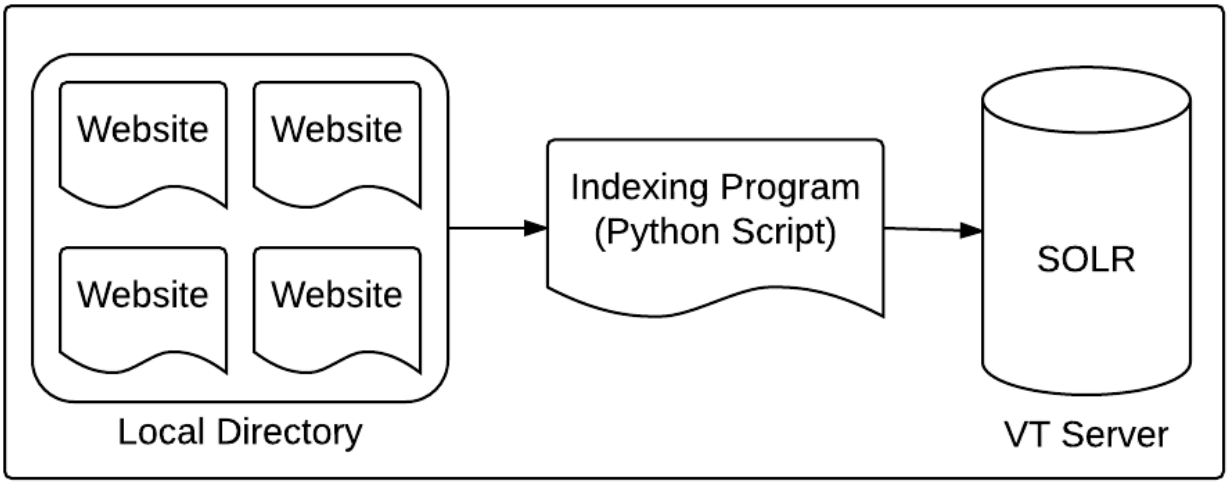


**Figure 3**: Data Flow 2: Data Indexing

The indexed data is then archived inside SOLR with the help of the python script (2). SOLR is configured to use ID tags and content tags. The content tags contain full text of each search result. This is used for 2 reasons: first to populate the index files with keywords and secondly to save and display the content of the search result to the user.

We create a unique ID tag for each web page by hashing the URL of that webpage. As a result of running this script, the text of all archived web pages will be indexed and saved on the SOLR server and will be searchable. This includes all working URLs mentioned in all archived tweets.

## C. User Interaction / Search Query Prototype

User interaction with SOLR directly is incredibly inefficient so we added a mid-layer component - Blacklight. Now, the user has a friendly interface where they can execute search queries to the platform. The prototype for this involves a 2 way data-flow unlike the prototype for extraction and indexing due to the search query and results display.

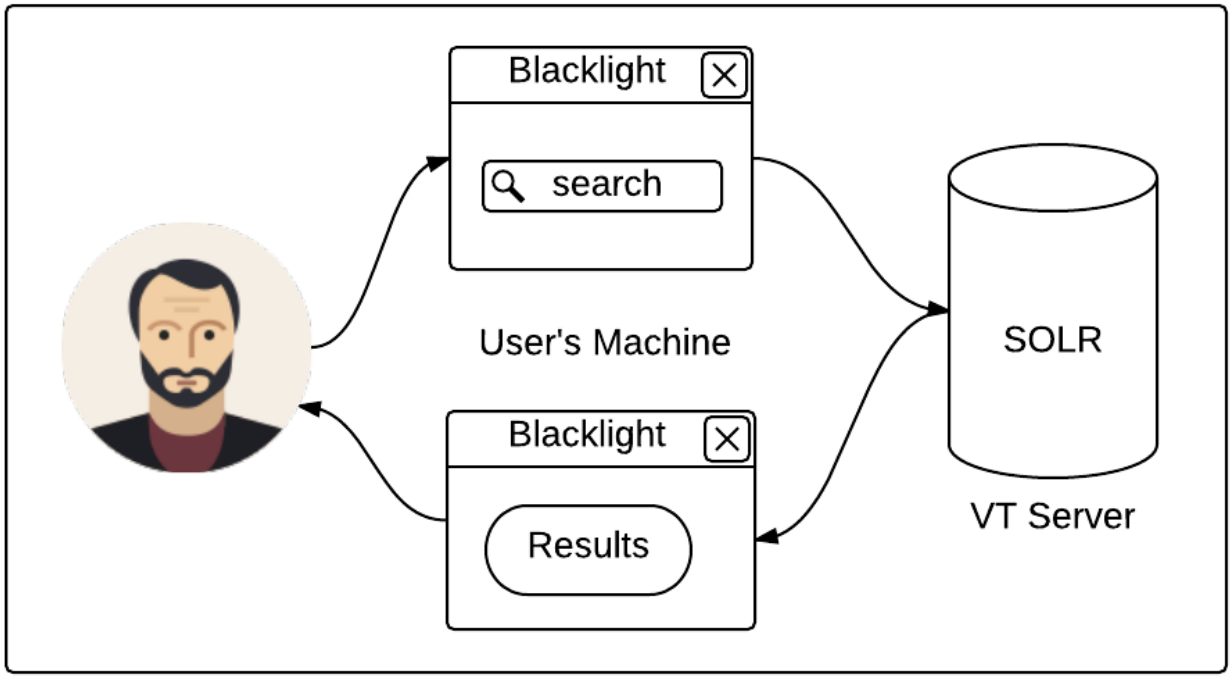


**Figure 4**: Data Flow 3: User Interaction & Data Query

## Blacklight Interfaces

The following screenshots show the interface that the users will use to make search queries and get the results to that. It is an easier layer between the user and SOLR platform and makes the user experience more efficient.

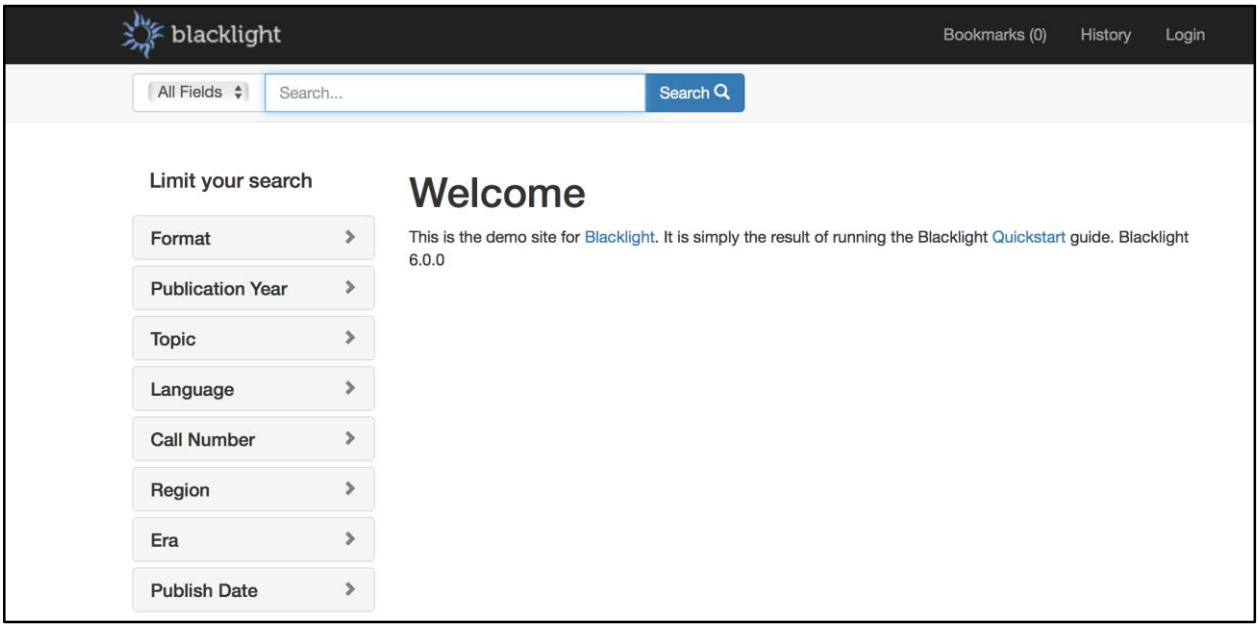


**Figure 5**: Sample Blacklight Interface 1: Landing Site.

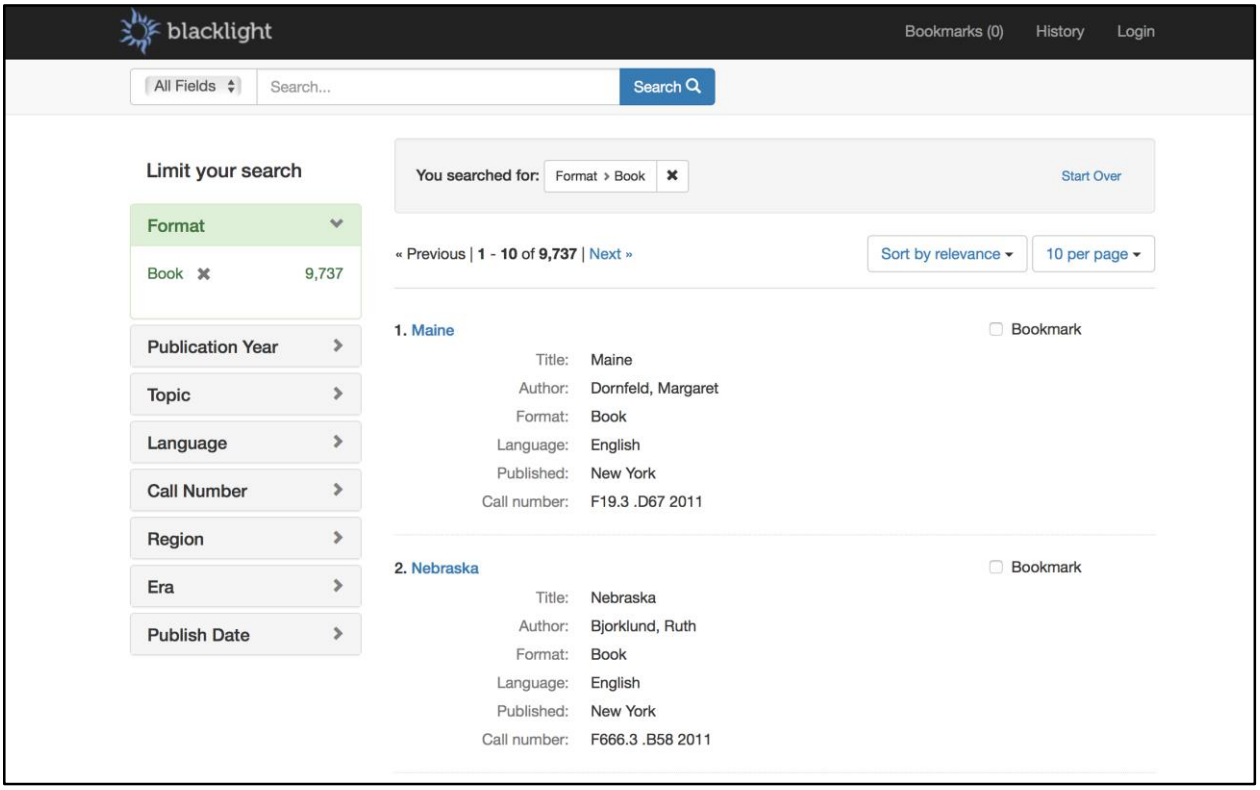


**Figure 6**: Sample Blacklight Interface 2: Search Result Display

## III. Refinement

Our system currently addresses the scope of the project. However, like any other system, there are various aspects which can be refined and updated to increase overall efficiency and productivity of the system.

Some refinements can be implemented but others require more time than what is available to us. Cumulatively, these refinements are a future upgrade to the system.

| **Problem** | **Refinement** |
|---|---|
| System only contains data from previously downloaded tweets. New information from recent tweets cannot be searched. | Get most recent tweets over the internet using Twitter's REST API to provide users with updated climate change information. |

| | |
|---|---|
| The current interface organizes search results in irrelevant categories such as “format”, “published”, etc. | Add relevant categories to organize search results. E.g. “date”, “organization” etc. |
| The python scripts used to download and index tweets had many problems and were initially unable to perform intended tasks. | Properly test all scripts and programs associated with the system to avoid any bugs or problems. |
| Manually running scripts to download websites from tweets and then indexing them to SOLR is not only time consuming, but also tedious. | Automate this process so that these scripts can be run without the need of any developers. Maybe schedule the script execution so that it happens daily. |
| The scripts written only support Python 2.7.x which is a problem because it is out-dated. The latest version of Python is 3.5.x and offers new, updated features. | Discuss with the client and decide to upgrade the version of Python used. This will require updating both scripts entirely. |

**Table 4:** Potential Refinements

# INDEX

TABLES & FIGURES

# REFERENCES


1. Beer, Chris. "Projectblacklight/blacklight." *GitHub*. Feb. 2016. Web. Mar. 2016. <https://github.com/projectblacklight/blacklight/wiki/Quickstart>.

2. "Solr Quick Start¶." *Apache Solr* -. Open Source. Web. <http://lucene.apache.org/solr/quickstart.html>.

3. "Solr Admin." *Solr Admin*. Ed. Mohammed M. Garib Farag. Open Source. Web. Mar. 2016. <http://nick.dlib.vt.edu:8983/solr/#/~cores/blacklight-core>

4. Projectblacklight/blacklight." *GitHub*. Web. Mar. 2016. <http://projectblacklight.org/#examples>.