VIRGINIA POLYTECHNIC INSTITUTE
AND STATE UNIVERSITY

CS 5604 INFORMATION STORAGE AND RETRIEVAL

Front-End Team
Final Report

Rachel Kohler, Reza Tasooji, Patrick Sullivan

Blacksburg, Virginia
December 13, 2016

Abstract

Information Retrieval systems are a common tool for building research and disseminating knowledge. For this to be possible, these systems must be able to effectively show varying amounts of relevant information to the user. The information retrieval system is in constant interaction with the user, who can modify the direction of their search as they gain more information. The front-end of the information retrieval system is where this important communication happens.

As members of Dr. E. Fox's class on Information Storage and Retrieval, we are tasked with understanding and making progress toward answering the question: how can we best build a state-of-the-art information retrieval and analysis system in support of the IDEAL [1] (Integrated Digital Event Archiving and Library) and GETAR [2] (Global Event and Trend Archive Research) projects?

As the front-end design and development team, our responsibility to this project is in creating an interface for users to explore large collections of tweet and webpage data. Our goal in this research effort is to understand how users search for information and to support these efforts with an accurate and usable interface. We support varied methods of searching such as query driven searches, faceted search and browsing, and filtering of information by topic. We implemented user management and logging to support future work in recommendations. Additionally, we integrated a visualization framework for future efforts in providing users with insightful visualizations which will allow them to explore social network and document interrelation data.

Contents

List of Figures

List of Tables

1 Overview

When thinking about an information storage and retrieval system, it is easy to get caught up in the system details and algorithms and forget the overall goal of the system: to provide information to a user. In order for a system to maximize utility, the user must be able to achieve this goal of obtaining information in an efficient and easy way. A major concern when working with big data is to understand how varied information can be presented to the users, how this information can be explored by the users, and finally, how this information can be obtained by the users. This opens the concept of designing a front-end that not only provides relevant results based on the user query, but also provides extra information that helps users to better understand the database. This could be through faceted searching or browsing capabilities which give the user a high level overview of the entire collection. Additionally, the use of graphical components to visualize different facets of the data could help a user to understand interconnections they might otherwise miss.

The previous interface that was used as a front-end for the IDEAL project was Hue [4]. Hue is a Web interface that is built specifically for data scientists, not just researchers. Programming experience is an expectation Hue places on its users. This is most notably seen in several widgets of Hue, where users can execute Python scripts or full SQL queries. These kinds of functions are not particularly helpful to the potential users of the GETAR system, many of whom lack programming experience. The previous work that involved Hue as a user interface to Solr [5] indexes also had performance bottlenecks. When larger collections from IDEAL were indexed, Hue was not able to efficiently handle querying these larger indexes. This was due to the way Hue unnecessarily loaded the entire index into memory for each query.

Our end goal was achieved in producing a functional front-end which meets the users' needs for searching and retrieving relevant information, whether it be through querying or browsing. In order to accomplish this goal, we had to begin by learning Ruby on Rails in order to work with Blacklight. We had to gain a good understanding of both Solr and Cloudera Search. After this base knowledge was obtained, we were able to connect a simple Blacklight [6] query interface to a small set of data provided on the server. Once we had the basic query functionality working, we added faceted search and browse. After debugging and testing, we added functionality to keep a user database that is able to track users' inputs. We also produced a proof of concept for GeoBlacklight to display geographic data visualizations and a proof of concept that incorporated a D3 [7] visualization into a Blacklight project. This document serves as a reference of the work completed throughout this semester, as well as a guide for future teams who work on this front-end.

2 Literature Review

Spring 2016 Front-End Team Final Report [8]

The Spring 2016 front-end team report describes the previous work on the front-end of an information retrieval system. It describes the implementation of Hue as a discovery interface connected to a Solr instance. Hue was a source of many difficulties for the previous front-end team. Overall, it was not designed for the planned end users of the IDEAL system. New users without programming experience could not fully utilize the Hue interface, and it provides many advanced tools that require experience as a statistician or data scientist. Since the GETAR system has accessibility in mind, we chose the Blacklight discovery interface as the primary communication between the user and the collections. Blacklight is also a more efficient platform for querying larger Solr indexes, whereas Hue loads the entire Solr index in memory for processing.

An Introduction to Information Retrieval, Chapter 9: Relevance Feedback and Query Expansion [9]

Chapter 9 of this information retrieval textbook includes information on how to improve information retrieval of a system in areas visible to the user. This can involve gathering information from the user as relevance feedback, and using this feedback to enhance queries to better estimate the information need of the user.

An Introduction to Information Retrieval, Chapter 11: Probabilistic Information Retrieval [10]

Chapter 11 of the information retrieval textbook includes information on how to improve information retrieval of a system by interpreting the user's information need. Since the information retrieval system has no guarantee to produce relevant documents, probabilistic methods are used to interpret the query in ways that maximize the likelihood of relevant documents being both present and highly ranked in the document result set.

An Introduction to Information Retrieval, Chapter 12: Language Models for Information Retrieval [11]

Chapter 12 of the information retrieval textbook shows ways of improving information retrieval systems by implementing language models in the information retrieval system. Language models can decompose both documents and queries into an automaton structure, which can be compared to one another. These language models are also effective since they leverage the context of words in both the query and documents. A common pattern of words in both the document and query is a very good indication of relevance.

Faceted Search Monograph [12]

This faceted search monograph introduces the concept of faceted search as an tool to enhance the productivity of users of information retrieval systems. Faceted search aids users by explicitly filtering results based on constraints they place upon the fields of the search. This also increases the efficiency of the

information retrieval system by only searching a well-defined subset rather than the entire collection of documents.

Nielsen's Usability Heuristics for User Interface Design [13]

is a well-known and popularly referenced set of design heuristics that was developed by Jakob Nielsen and published in his 1994 *Usability Engineering* book. These heuristics are viewed as guiding principles in design and are often referred to by designers, not as strict rules, but as an outline to follow throughout the design and evaluation of an interface. The set of heuristics are as follows:

Visibility of system status. The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

Match between system and the real world. The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

User control and freedom. Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

Consistency and standards. Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

Error prevention. Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

Recognition rather than recall. Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

Flexibility and efficiency of use. Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

Aesthetic and minimalist design. Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

Help users recognize, diagnose, and recover from errors. Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

Help and documentation. Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

This list of guidelines is an important resource for our team in the design and evaluation phase. We must ensure that our design work is consistent with these heuristics and use them to perform expert heuristic evaluations on our interface. For example, if we design an interface which is simple, intuitive, and easy to use, we may overlook the fact that we need helpful error messages and recovery strategies. This list will

remind us to include these in the final design. We will refer back to it often and build off of these heuristics.

DRY Programming Principle [14]

stands for Don't Repeat Yourself. It is a methodology that was developed by Andy Hunt and Dave Thomas in their 1999 book *The Pragmatic Programmer* and is central to the Rails programming methodology. See Section 8.4 for implementation details. The basic argument for DRY coding is to eliminate errors and maintenance issues that arise when information is duplicated. Throughout the programming process, modification of code that contains duplicated information will likely cause erroneous behavior or all out failure. For this reason, Hunt and Thomas argue "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system". This is the foundation of the DRY methodology and forms the core of the authors' argument.

The authors mention four types of duplication programmers should be wary of:

1. **Imposed duplication.** Developers feel they have no choice–the environment seems to require duplication.
2. **Inadvertent duplication.** Developers don't realize that they are duplicating information.
3. **Impatient duplication.** Developers get lazy and duplicate because it seems easier.
4. **Interdeveloper duplication.** Multiple people on a team (or on different teams) duplicate a piece of information.

Detailed examples of each of these duplication categories and solutions for solving them can be found in the book. Rails uses a templating structure to avoid duplication. Being familiar with not only Rails templating, but also the DRY principle and the strategies in use to help avoid duplication will help our team to maintain DRY code.

Munzner's Design Principles of Effective Visualization [15]

These principles describes the process of designing and implementing an effective visualization of data, and why it is important. Munzner breaks down this process into several questions that help resolve visualization concerns. They are:

What? Data Abstraction. Understanding the characteristics of both the dataset and the dataset's attributes (columns) is critical to successful visualization, but often includes non-relevant information. Abstracting the dataset to a fundamental structure and identifying attribute types reveals correct or incorrect approaches to display this information. For example, if one attribute of the data is determined to be ordinal, we understand that this data can be sorted, but there is no direct way of measuring distances between values. This directs us toward visualization options that support communicating this feature of the data to the user. Size of the dataset is also an important issue that is investigated here, as both computers and people have limits to how much information they can process.

Why? Task Abstraction. All visualizations have a core purpose of communicating information. What defines important information is determined by the user and the tasks they seek to accomplish. If the visualization is to be practical, then the information shown should follow the same prioritization of the tasks of the user. Since the domains that use visualizations are extremely different, it is useful to abstract the tasks away from their domains so that they can be compared to alternative visualization solutions. Knowing the target audience of the visualization also indicates what prior knowledge they bring when they are interpreting the visualization. These task abstractions take two forms: actions and targets. Actions describe how the user interacts with the visualization, which could be annotating data points, deriving new data, locating a known target, browsing for interesting targets, querying for details of a data point, or summarizing a number of data points. Targets are aspects of the data that the user has an interest in, such as trends, correlations, distributions, outliers, or topology.

How? Choosing Design. The previous steps of data and task abstraction come to influence the implementation decisions made here. These abstractions are combined with research on human cognitive and perceptual abilities to dictate the effectiveness of one approach versus another. These decisions drive how data is arranged, mapped, selected, navigated, manipulated, filtered, aggregated, and partitioned. Choosing every aspect of a visualization at once leads a designer toward visualizations that do not consider new solutions. This process defines a complete visualization one piece at a time to avoid making those assumptions. There is no single visualization that works in every situation, so new possibilities should always be considered.

3 Requirements

As the front-end of this information retrieval system, we are required to support the needs of our user base.

1. The interface shall support searching of webpages and/or tweets.
2. The interface shall support searching which considers full-text of webpages and/or tweets.
3. The interface shall support both browsing as well as query searching.
4. The interface shall implement browsing and query searching based on facets connected with metadata. Examples of metadata include but are not limited to:
    - date(s)
    - author(s)
    - applicable locations
    - organizations
    - persons
5. The interface shall implement browsing and query searching based on facets connected with suitable categories. Examples of suitable categories include but are not limited to:
    - type of event and the particular event (name) of that type
    - author(s)
    - topics implicit in the collection
    - groupings (i.e., clusters based on content) inherent in the data
6. The interface shall support query results which can be filtered.
7. The interface may support query results which can be sorted.
8. The interface shall provide the option for user authentication to be required for some, but not all, content.
9. The interface shall generate logs which provide additional details about user interactions.
10. The interface shall provide instructions for use. This will be in the form of text explanation on a page, hover over for additional information and a help page.
11. The user documentation and help shall be complete.
12. The interface shall include error messages which explain how to recover from the error.
13. The interface actions and elements shall be consistent across pages and with users' expected behavior.
14. The interface shall display information and data which is accurate to what is produced by the Solr team.
15. The interface may provide visualizations which support user exploration and analysis.

4 Design

4.1 Conceptual Background

The goal of the front-end is to be an effective interface that communicates information with the user. In the case of digital libraries, the front-end allows the user to query for relevant documents and display the results of a query in an effective manner. Other abilities the digital library front-end could provide include faceted search, topic visualization, and collection browsing.

4.2 Tools

Blacklight [6]

Blacklight is a discovery engine that supports the creation of aesthetically pleasing and exceedingly functional websites which allow users to work with the Solr indexes of large collections. The front-end of the GETAR project will utilize the Blacklight interface to display the results of queries on a collection's Solr index.

Queries are the core communication between the user and the Solr index, so it is important that the front-end uses a stable, efficient and scalable method for handling queries. Blacklight provides much of this functionality and minimizes the amount of processing overhead in managing larger Solr indexes. This is an especially important consideration for collections that grow in scale or are updated in real time. Collections of tweets and webpages are often managed in this way. This is a feature of the GETAR project.

Basic queries commonly retrieve large numbers of documents which are not relevant based on a specific field. Applying filters to these fields makes result sets more relevant and increases the overall utility of the information retrieval system. Faceted search [12] is a filtering ability that is included in the core abilities of Blacklight, supporting advanced queries by users. Facets include publication date, author, topic, or language. See Section 3 for detailed requirements. Other projects that showcase the capabilities of Blacklight can be seen here: http://projectblacklight.org/#examples.

GeoBlacklight [16]

GeoBlacklight is a plugin for Blacklight that enables effective displays of geospatial data. Data from Blacklight searches can be displayed on interactive maps for the user to browse or constrain results by their geographical location. Geolocation data can be explicitly included or inferred from tweets, which allows a user to be able to follow the spread of social activity caused by an event. Webpages can also be linked to their location by their language, or by the domains in the URL that specify country to region. This enables the user to visualize location-based data much easier than through detailed result sets. The visualization provided by GeoBlacklight is excellent for browsing data. We developed a proof of concept for GeoBlacklight to make recommendations about its use in the future. Other projects that showcase GeoBlacklight can be found here: http://geoblacklight.org/.

Devise [17]

Devise is a Ruby gem for user authentication and session management that is configured to work with Blacklight by default. It is a flexible authentication tool that matches the MVC architecture of Blacklight while being completely modular. Core features that help manage users and security are organized into ten modules that can be included as needed. Examples of these modules include 'Trackable', 'Recoverable', 'Registerable', 'Lockable', 'Confirmable', and so on. Devise is designed as a Rails 'engine', or a miniature Rails application that is used by other Rails applications to provide a specific feature.

Cloudera Quickstart [18]

Cloudera Quickstart is the virtual machine used for the learning, testing, and demo of Cloudera Distribution Hadoop (CDH). It is built on top of the RedHat Linux Distribution operating system. The Cloudera Manager is included to aid in managing the clusters. To work with Cloudera Manager, packages must be migrated into parcels. The express version is free, but lacks advanced management tools.

Solr [5]

Apache Solr is an open-source enterprise search platform. It can index collections of documents to enable quick retrieval of them based on queries. It is scalable to both very large document collections and high volumes of user traffic.

Understanding Solr

Apache Solr is an open source search platform built upon a Java library called Lucene. In Solr, a **core** is composed of a set of configuration files, Lucene index files, and Solr's transaction log. Each **core** has its own schema.xml and solrconfig.xml file. Figure 1 shows the location of cores in the Solr interface. You can see all the cores in the "Core Admin" view (Figure 2).

Schema.xml is where you tell Solr what types of **fields** you plan to support, how those types will be analyzed, and what fields you are going to make available for import and queries. Blacklight controllers look at these fields and show them in the front-end. For example, for facet fields to be displayed correctly inside Blacklight, these fields inside schema.xml *must be indexed fields*. You can see all fields and field types implemented inside schema.xml for the selected core in the Schema Browser inside the Solr interface (Figure 3).

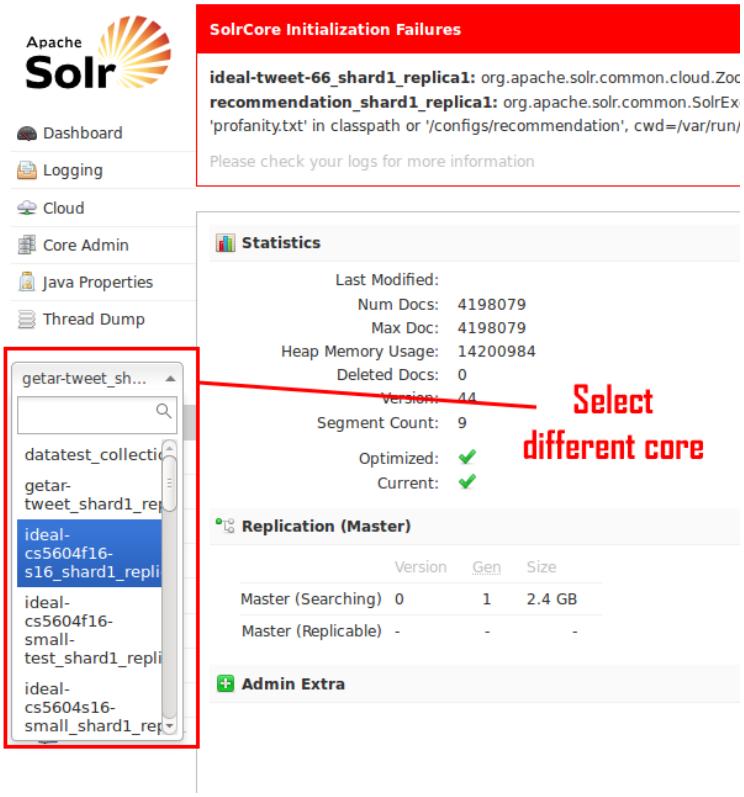


Figure 1: Cores inside Solr

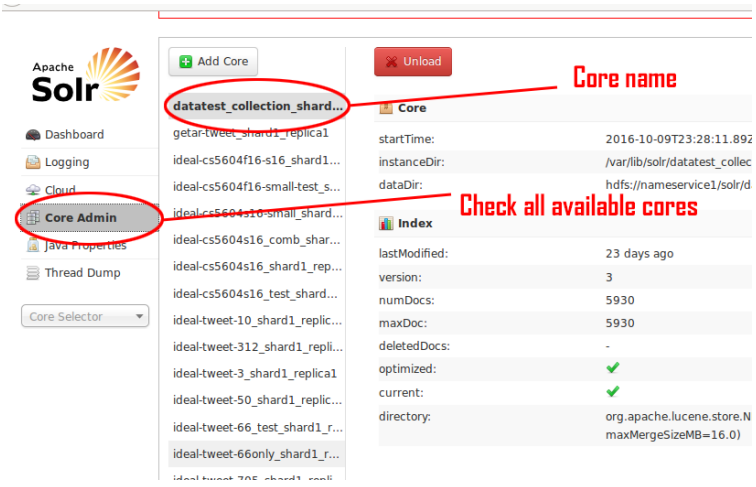


Figure 2: Core Admin

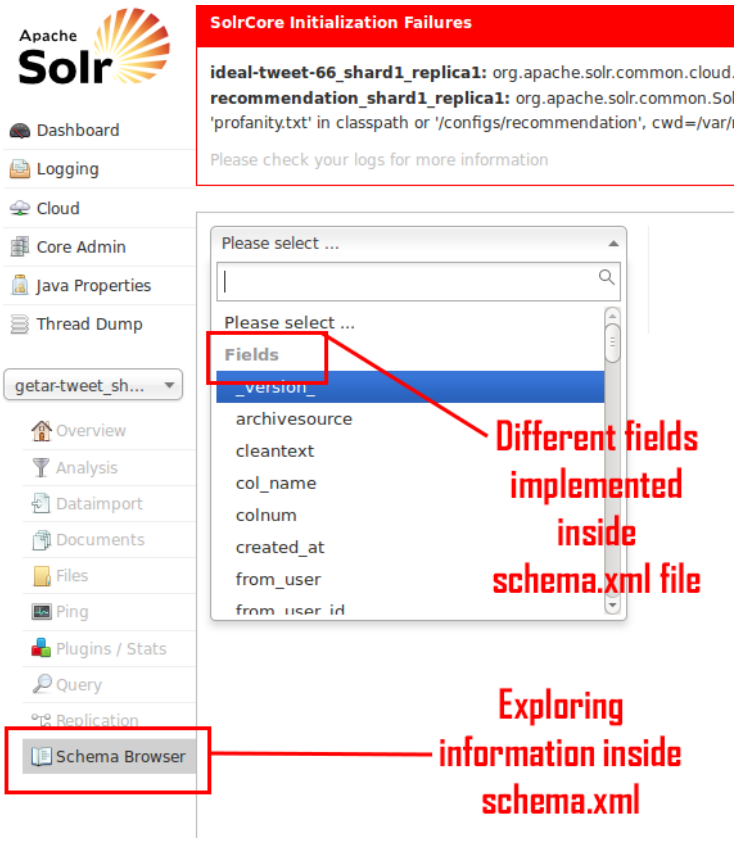


Figure 3: Schema browser

Hue [19]

Hue (Hadoop User Experience) is a front-end that comes with Cloudera. Although it is possible to use Hue as our front-end for document collections in Solr, the report from the previous front-end group shows that Hue is not capable of loading the full indexes into memory, and the performance was not fast enough. GETAR is also designed to be used by a much broader audience than the typical users of Hue, who are data scientists. Hue users are expected to have programming experience and a deep knowledge of the data set to navigate the collection. Many of the views that Hue provides are direct representations of the database, with columns that are not labeled to be easily understood or with little meaning to new users of the dataset. Hue is also closely coupled with the local machine's Cloudera Search environment, which increases the risk of slowing down the entire cluster when Hue requests large amounts of processing. Hue is not meant to be widely accessible, as the tasks available to Hue users can be advanced and computationally expensive. We avoid these issues by choosing Blacklight as our primary user interface to the GETAR collection.

Cloudera Search [20]

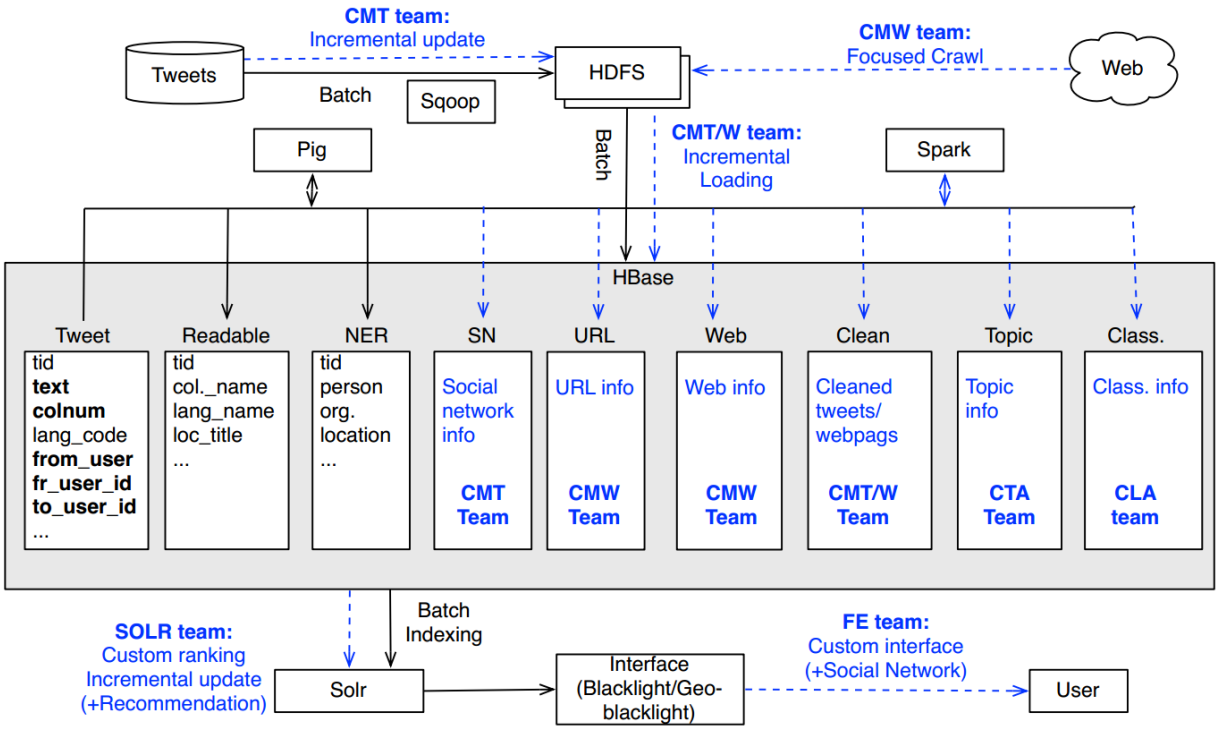
Cloudera Search incorporates the search platform of Solr into a MapReduce [21] workflow for indexes. In the reduce phase, Cloudera Search uses Solr to write the data as a single index or as index shards depending on the collection and configuration.

4.2.1 Additional Tools

- PhaseVis [22] for identification of the phase of recovery for tweets.
- D3.js [7] for data-driven visualization and interaction of webpages. Supports force simulated networks that are useful for visualizing social networks.
- P5.js [23] for graphical visualization and interaction of webpages. Provides complete drawing control over webpage elements, as well as some tabular data management.

4.3 Approach

Figure 4 is an short overview of the data flow of the current GETAR system, as well as how Blacklight will connect to the GETAR data. Tweets are labeled with the real world event class(es) they are associated with the classification team. Tweets and webpages are collected, filtered, and cleaned by the collection teams into HDFS and then into HBase. These documents are clustered and characterized by topic according to the direction of the clustering and topic analysis team. The Solr team creates an index of the documents in HBase that can be queried efficiently. We, the front-end team, create a user interface that enables the user to search, browse, discover, and visualize documents in the GETAR collection. The many steps of data collection, processing, and retrieval means there must be a high amount of communication between these teams to facilitate this information retrieval system.



Team	Spark	Pig	Solr	Blacklight	Sqoop	Focused Crawler
CMT		X			X	
CMW		X				X
SOLR			X			
FE	X					
CTA/CLA	X					

Figure 4: Data flow of GETAR project [3]

Blacklight Connection

Figure 5 shows the details of the front-end connection with Solr. Blacklight is a Ruby on Rails applications, and runs on a Rails server. For the GETAR project, Solr is contained within the Cloudera Search environment on the cluster. Different collections in Solr are typically given their own Solr Core and name to describe the collection. The configuration of Blacklight requires the schema of the Solr Core's index to be able to generate Solr queries. Blacklight sends these requests to the Solr Core, which responds with the results list of likely relevant documents.

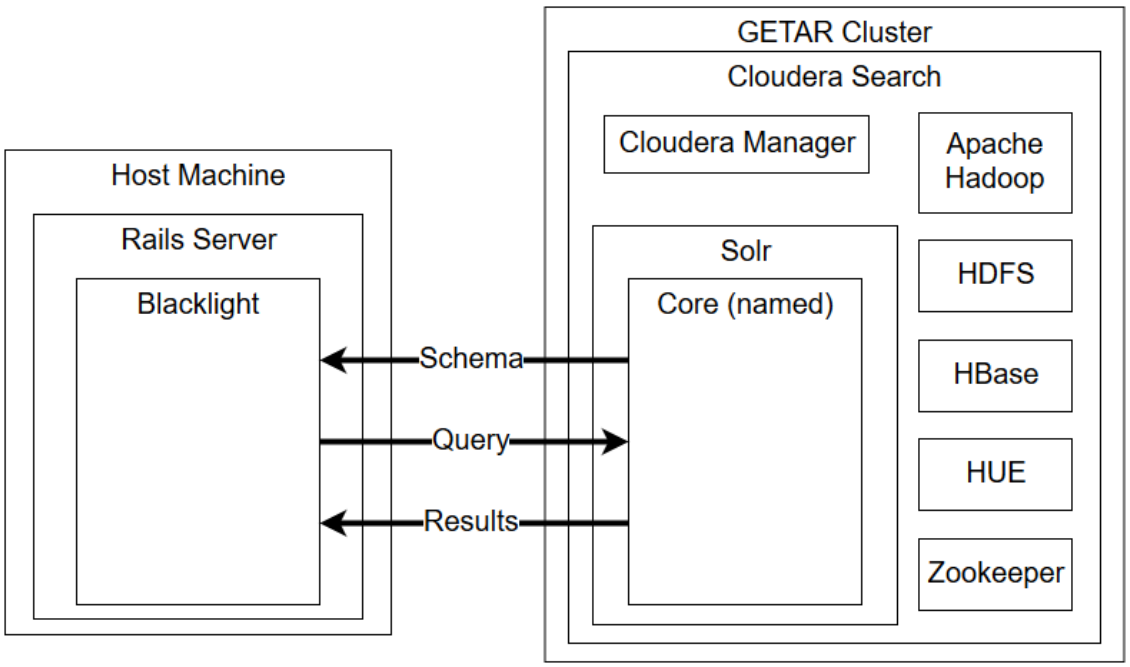


Figure 5: Blacklight communication with Solr and GETAR data

4.4 Data Mapping

In order to develop a working interface, we first had to understand the data we are trying to provide to the user. Since several different teams are in charge of collecting and processing this data, it was our responsibility to understand their tasks and provide them with any necessary direction in these efforts. Additionally, we had to design a data mapping which would detail the transformation and representation of data throughout this pipeline. The following data mapping is a result of our work.

HBase		Solr	Blacklight				
Column Family	Column Name	Index	UI Element	Facet	Search Filter	List of Results	Single Result Page
tweet	archivesource	archive_source_s	--				
tweet	from_user	author_s	Author	■	■	■	■
tweet	time	created_time_dt	Time			■	■
webpage clean-tweet	clean-text-profanity clean-text-solr	text_txt	Text		■	■	■
clean-tweet webpage	readable-lang language	language_s	Language	■			■
clean-tweet webpage	readable-collection collection-name	collection_name_s	Collection			■	■
clean-tweet	geo-location	location_ss	Location	■		■	■
clean-tweet	created-year	t_year_i	Year	■			
clean-tweet	created-month	t_month_i	Month	■			
clean-tweet	hashtags	hashtags_ss	Hashtag	■	■		■
clean-tweet	mentions	mentions_ss	Mention	■			■
clean-tweet	classification-label	classification_s	Classification?				■
clean-tweet webpage	real-world-events	events_s	Event	■	■		■
clean-tweet	sner-people	sner_people_s	People Mentioned	■			■
clean-tweet	sner-organizations	sner_org_s	Organization Mentioned	■			■
clean-tweet	sner-locations	sner_loc_s	Location Mentioned	■			■
clean-tweet	tweet-importance	t_importance_f	--				
webpage	title	title_s	Title		■		■
webpage	author-publisher	author_s	Author	■	■		■
webpage	sub-urls	sub_urls_s	URLs				■
webpage	domain-location	location_s	Location				■
webpage	organization-name	organization_s	Organization				■
webpage	webpage-importance	w_importance_f	--				
doc-type	doc-type	doc_type_s	Type	■			■
doc-topic	label-list	topic_label_s					■
doc-topic	doc-probability	topic_probability_s	--				
doc-cluster	cluster-label	cluster_label_s					■
doc-cluster	cluster-probability	cluster_probability_s	--				

Figure 6: Data mapping

The three larger columns represent the three phases of transformation for the data. After the initial insertion into the given HBase columns and column families, the data must be converted to an indexed and stored field in the schema file. Field names differ from column names and represent the type of the data, e.g. _s for string or _ss for split string, etc. Finally the indexed data must be mapped to UI elements. Element labels are intentionally consistent across all similar elements. The location the data is shown in or the interaction element which the data is a part of is indicated by shaded boxes.

This document, along with the schema from Solr, provided the basis for our development work and can be added to and modified in future iterations of the interface.

5 Implementation

5.1 Timeline

The timelime below details our complete work schedule for the duration of the semester.

Week	Date	Task	Status	Team Member
1	8.23 - 8.28	Team creation	Complete	All
2	8.29 - 9.4	Learn about Blacklight, GeoBlacklight and read documentation	Complete	All
2	8.29 - 9.4	Install and set up necessary software	Complete	All
3	9.5 - 9.11	Requirements gathering and brainstorming design	Complete	Rachel
3	9.5 - 9.11	Initial setup of Blacklight demo	Complete	All
4	9.12 - 9.18	Wireframing	Complete	Rachel
4	9.12 - 9.18	Additional Blacklight demo configuration	Complete	All
5	9.19 - 9.25	Prepare for team presentation	Complete	All
5	9.19 - 9.25	Work with Solr team to connect Blacklight and Slor	Complete	All
5	9.20	DUE: Interim report 1	Complete	All
5	9.22	DUE: Team presentation	Complete	All
6	9.26 - 10.2	Debugging connection issues with Solr team and Sunshin	Complete	Reza, Patrick
6	9.26 - 10.2	Initial prototyping and planning	Complete	Rachel
6	9.29	DUE: Interim report 1 review	Complete	All
7	10.1	Begin Setup Plan A: Remote connect to CDH	Complete	Reza
7	10.1	Attempt Plan A; Connection fails	Complete	Reza
7	10.1 - 10.4	Run Blacklight on Ubuntu and connect to Cloudera Solr	Complete	All
7	10.3 - 10.9	Attempt Plan B; Connection fails	Complete	Patrick
7	10.3 - 10.9	Java upgrade and re-attempt Plan B; Connection Fails	Complete	Reza
7	10.4	Begin Setup Plan B: Cloudera Quickstart VM	Complete	Reza
7	10.5 - 10.7	Run Blacklight on Cloudera version 5.8 on local machine	Complete	Patrick
8	10.9 - 10.11	Run Blacklight on Cloudera version 5.3 on local machine	Complete	Reza
8	10.10 - 10.16	Learn Ruby, learn Rails, learn Bootstrap	Complete	Rachel
8	10.10 - 10.16	Initial edits to interface and documentation	Complete	Rachel
8	10.10 - 10.16	Brainstorm alternative front-end ideas with Sunshin and begin implementing	Complete	All
8	10.10 - 10.16	Reach out to Cloudera support about alternative front-end platforms	Complete	Rachel
8	10.11	DUE: Interim report 2	Complete	All
9	10.17 - 10.23	Try connecting Blacklight Solr to index data from Cloudera HBase	Complete	Patrick
9	10.17 - 10.23	Try Blacklight with different version of Solr	Complete	Reza
9	10.17 - 10.23	Meet with Paul Mather to work through issues	Complete	Rachel, Reza
9	10.17 - 10.23	Work with other teams to define schema	Complete	Rachel
9	10.17 - 10.23	Look into Tomcat errors	Complete	Reza
9	10.17 - 10.23	Look into authentication issues	Complete	Patrick
9	10.17 - 10.23	Reach out to Cloudera support again	Complete	Rachel
9	10.17 - 10.23	Work with Sunshin to find alternative front-end solutions	Complete	Rachel
9	10.18	DUE: Interim report 2 review	Complete	All
10	10.24 - 10.30	Look into CatalogController issues	Complete	Reza
10	10.24 - 10.30	Test Blacklight connection with GETAR data	Complete	All

11	10.31 - 11.6	Set up VM provided by Fox	Complete	Reza
11	10.31 - 11.6	Establish VM remote desktop connection	Complete	Patrick
11	10.31 - 11.6	Configure faceted search for GETAR schema	Complete	Rachel
11	11.3	DUE: Interim report 3	Complete	All
12	11.7 - 11.13	Incorporate existing design work into Blacklight running on VM	Complete	Rachel
12	11.7 - 11.13	Continue design to follow class defined schema for facets and browsing	Complete	Rachel
12	11.7 - 11.13	GeoBlacklight POC	Complete	Reza
12	11.7 - 11.13	Create data mapping from schema and get feedback	Complete	Rachel
12	11.7 - 11.13	Devise user authentication and logs	Complete	Patrick
12	11.8	DUE: Interim report 3 review	Complete	All
13	11.14 - 11.20	Visualization POC	Complete	Reza
13	11.14 - 11.20	Integrate final class schema and data on VM	Complete	Patrick
14	11.21 - 11.27	Complete final design	Complete	All
15	11.28 - 12.4	System testing and feedback	Complete	All
15	11.28 - 12.4	Final changes and documentation	Complete	All
16	12.5 - 12.11	Final presentation prep and last minute debugging	Complete	All
16	12.6	DUE: Final project presentation	Complete	All
16	12.7	DUE: Final project report	Complete	All
16	12.7	DUE: Group member evaluations	Complete	All

Table 1: Project timeline

5.2 Milestones

- Chose a flexible and powerful front end development tool and proved it would meet our needs now and in the future
- Developed working query and browse interface using Blacklight
- Finalized a schema and created a data mapping document which details how to map data elements from HBase through Solr to the front end
- Migrated Blacklight interface to a series of Virtual Machines for future use
- Set up user authentication using Devise with logs
- Completed GeoBlacklight proof of concept and made recommendations on future work
- Integrated D3 visualization into Blacklight and document for future teams

5.3 Deliverables

Deliverable	Additional information	Status
Wireframes	interface and user interactions based on requirements	Complete
Prototype	basic query interface with sample data	Complete
Version 1.0	basic query interface with real data	Complete
Version 2.0	faceted search and browse	Complete
Version 3.0	user authentication and visualizations for social network data	In progress

Table 2: Project deliverables and status

5.4 Wireframes

Wireframes represent a first iteration of design work. They are strictly used to show basic functionality, navigation flow and user interactions. They are also primarily used to elicit design feedback. Feedback on these wireframes was considered and incorporated into the design of the Blacklight prototype. Additionally, more realistic data is shown in a prototype scenario as wireframes are not meant to portray realistic data, but fictional data only. After producing wireframes, we were able to gather feedback for our design and have since modified our design to reflect this feedback.

5.5 Blacklight Prototype

Our first major milestone and prototype was to set up the Blacklight demo. This Blacklight setup would orient us to the basic configuration of Blacklight and familiarize us to basic configuration issues and troubleshooting those issues in small, contained development environments. Additionally, the prototyping would support our understanding of the basic interface and capabilities of Blacklight. This setup was very successful on multiple host machines. The initial testing of the Blacklight Demo setup involved using Solr to index the MARC test records that are included with Blacklight by the Project Blacklight Team [24]. This data could be browsed and queried as expected through the basic Blacklight interface. Through this process, we learned about the various interface elements of Blacklight, the file structure of the front-end, and the various commands needed to edit, build, and run the interface. The exact methods for setting up this Blacklight demo can be found in Section 8.2.3. The system overview can be seen in Figure 7. The next step is to extend this interface to support our own data and schema.

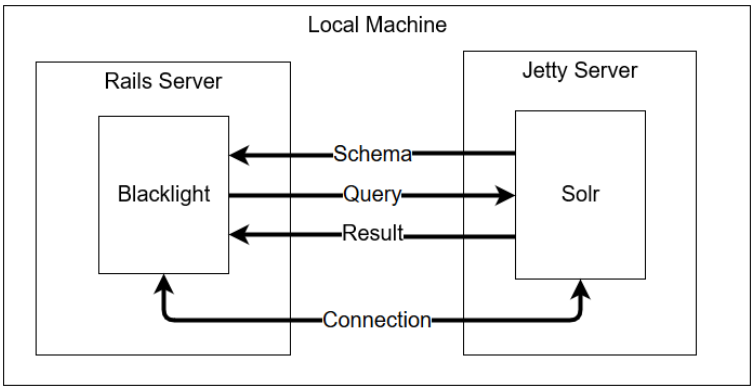


Figure 7: Blacklight demo

5.5.1 Blacklight Connection Setbacks

While the Blacklight demo connection to Solr worked without issue, there were several issues during our first attempts to connect Blacklight to the Solr instance where the GETAR dataset is located. There is some documentation on the Project Blacklight [24] site on the changes needed for configuring a connection to Solr, but after following these steps, issues persisted. We attempted to contact Cloudera Support about the issues but we received no response. We tried contacting people with previous Blacklight or Solr experience, including the Project Blacklight team, but we were unable to resolve the issues in these meetings.

Blacklight Connection First Attempt

Figure 8 shows the overview of our first attempt of connecting Blacklight to Solr. This application overview is our first attempt at an actual connection between Blacklight and the existing Solr instances for the GETAR project Hadoop Cluster. See Figure 4 for an overview of data flow in the GETAR project. There were multiple errors in establishing the connection between Blacklight and Solr. This introduced us both to the complexity of Cloudera Distribution Hadoop and the difficulty of troubleshooting when involving network connections. We decided an alternative setup would help determine if the results were caused by network and port issues, insufficient privileges, or a customized Solr API that blocks Blacklight connections.

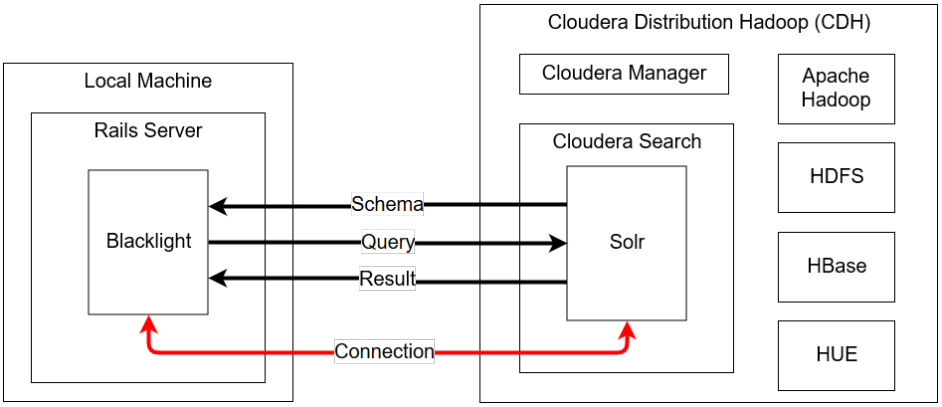


Figure 8: Blacklight connection first attempt

Blacklight Connection Second Attempt

Figure 9 shows our second attempt of connecting Blacklight to Solr. We formulated this possible solution to aid us in determining the sources of issues. By developing on a single virtual machine, we eliminated any issues tied to network connections. This also allowed us full control over the Cloudera Distribution Hadoop system using the Cloudera Manager. This virtual machine can be deployed on the Cloudera Distribution Hadoop system that contains the majority of the GETAR project. However, the communication issues between Blacklight and Solr persisted. This is most likely tied to the Cloudera Distribution Hadoop usage of Cloudera Search as an interface with Solr, which has additional configuration besides the standalone Solr instance.

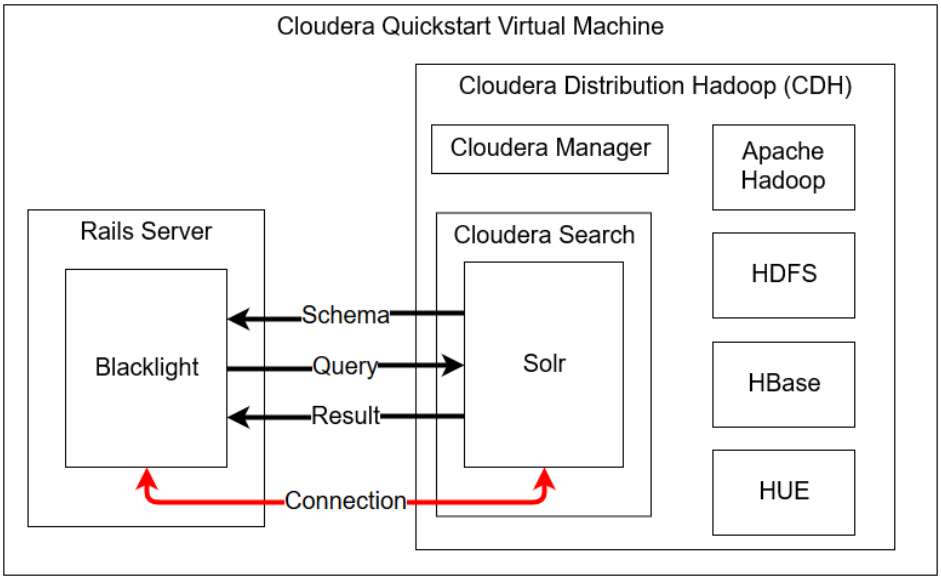


Figure 9: Blacklight connection second attempt

After these attempt and an in-class demonstration by the Solr team of the workings of Solr, we successfully connected Blacklight to the GETAR Solr instance. There are several different configuration files from Solr that together must match the *exact* configuration of Blacklight. As there was a large difference between the Blacklight demo schema and the GETAR schema, small changes would not resolve the problem. The entire schema configuration of Blacklight had to be investigated for any missing fields or facets defined from the Solr schema for the connection to appear healthy. See Section 8.3.1 for details of this connection and the many configuration changes needed to be successful.

5.6 Version 1.0

Based on feedback we received from our wireframes and our experience with the prototype, we decided to begin our development of Version 1.0 with the basic interface of the prototype rather than a custom interface. This was to eliminate confusion and streamline data integration with other teams. Since the schema was still in development, a custom interface could not be supported. Therefore, interface customization was left for Versions 2.0 and 3.0. Version 1.0 of the front-end satisfies the following requirements: 1, 2, 13, 14. See Section 3 for the complete list of requirements.

Requirement 1: The interface shall support searching of webpages and/or tweets.

Searching of both webpages and tweets is supported in Version 1.0 to the extent that these appear in the data set. Blacklight currently makes no distinction between webpage and tweet data. This can later be a facet of the searching interface.

Requirement 2: The interface shall support searching which considers full-text of webpages and/or tweets.

The query functionality of Blacklight searches all data which has been indexed. This means that since the full text is indexed, it is also searchable. Note that by full text, we are referring to the complete text which is able to be accessed by the user. This excludes profanity and other information that has been removed at the request of our team.

Requirement 13: The interface actions and elements shall be consistent across pages and with users' expected behavior.

Since we have not customized the interface, the version 1.0 interface conforms to the Blacklight standard demo interface. This means it is consistent across all pages with respect to colors, buttons, placement of UI elements, data placement and representation, etc. Additionally, the users' expectations have not been altered either. This requirement will need further analysis throughout the customization process.

Requirement 14: The interface shall display information and data which is accurate to what is produced by the Solr team.

Tthe information that is displayed in Version 1.0 of the Blacklight interface comes from an index and schema file that were sent to us by the Solr team at the beginning of the semester. Until we receive an updated index and schema file, the information displayed is accurate to what we have received from the Solr team.

5.7 Version 2.0

For Version 2.0 we were working off of a preliminary schema and a test data table. During this iteration of development we were able to finalize the schema decisions and data mapping through iterative development and feedback from the class. After finalizing the data mapping, we were able to customize the Blacklight interface to meet our data needs. Version 2.0 of the front-end satisfies the following additional requirements: 3, 4, 5, 6. See Section 3 for the complete list of requirements.

Requirement 3: The interface shall support both browsing as well as query searching.

With our test data table, we were able to implement both query and browse functionality in the front-end. Users may issue a query to find relevant documents, or they may browse all documents in the collection, using the facets to help focus the browse.

Requirement 4: The interface shall implement browsing and query searching based on facets connected with metadata.

Requirement 5: The interface shall implement browsing and query searching based on facets connected with suitable categories.

Both Requirements 4 and 5 were met with our implementation of faceted search. Faceted search is supported for query and browse. The final facets are: Author, Language, Location, Year, Month, Hashtags, Mentions, Event, and Type.

Requirement 6: The interface shall support query results which can be filtered.

Users can access the facets on the left panel of the interface. These facets can be used to filter search results based on a certain property of a facet. With our test data set and schema, the facets have been configured to support the user's filtering of the data.

5.8 Version 3.0

Version 3.0 is the final version of the front-end to be completed by our team. Development of Version 3.0 continued on the test data table, but with a finalized schema. Some of the work completed in this version is preliminary in nature, and future work is described in Section 6 to help the next team in continuing this development. Version 3.0 of the front-end satisfies the following additional requirements: 7, 8, 9, 10, 11, 12, 15. See Section 3 for the complete list of requirements.

Requirement 7: The interface may support query results which can be sorted.

Sorting is available for fields which are sortable. As of Version 3.0, we are allowing sort by time of creation, but any non multi-value field can be implemented to be sorted.

Requirement 8: The interface shall provide the option for user authentication to be enabled for some, but not all, content.

Currently, user authentication is provided but optional in satisfaction with this requirement. Users may create an account which allows them access to additional features, but guest users are also permitted to access the information in this system.

Requirement 9: The interface shall generate logs which provide additional details about user interactions.

For both guest and authenticated users, logs are generated which detail their queries and accessed documents. Additionally, for authenticated users, tables store their search history and bookmarked documents. There is much information that can be collected about the user as well including IP address, but since this requirement is focused on user iteractions, we save these details for later discussion.

Requirement 10: The interface shall provide instructions for use.

An extensive user manual is provided in Section 7 to provide detailed instructions of use. The manual is written to provide support to even the most novice of users. It details many of the possible interactions which are currently available to the users.

Requirement 11: The user documentation and help shall be complete.

Currently, the user manual is complete based on the features currently implemented. It does not include any information on proof of concepts (GeoBlacklight or D3) or future work. Future updates to the system must include updates the user manual as well.

Requirement 12: The interface shall include error messages which explain how to recover from the error.

The only error messages which are presented to the user at present are in response to failed login attempts. We felt there were no other places to include error messages to users in the basic searching interface.

Requirement 15: The interface may provide visualizations which support user exploration and analysis.

Work has been completed to integrate D3 into Blacklight. By the time this project was completed, there was no relevant data to visualize. In the future, there will be data to visualize in the front-end. At that time, D3 will already be integrated for use.

6 Future Work

There are several ways of expanding the current front-end to improve the information retrieval system. These features can positively effect both the user experience, as well as the processes that happen in the back end for collecting, storing, and retrieving documents.

6.1 User Management

6.1.1 User Types

This project does not have a single audience in mind, so there will be diversity among its users. They may have varying languages, backgrounds, or computer expertise as well as different questions that need answering by the IR system. Building a front end that effectively answers questions from any user is a difficult task, but it is made simpler by grouping the users into types. This enables the ability to design specialized tools and views for each type of user and avoids any issues when user needs conflict with each other.

6.1.2 Accessibility and Security

As the GETAR project and the front end move closer to being accessible to the end users of the system, care should be taken to protect the system and the information it holds. IR systems are power tools in the big data age, and should not be carelessly left open to attack or misuse. User authentication through Devise [17] only provides some basic protections in its current configuration. This security should be expanded to fit the needs of the information retrieval system for this project.

One way of bolstering security in this system would be though analyzing the system for high-value targets of attackers and taking steps to protect these valuables. Another way of building security is by restricting access to the front end to users on a specific network. This would force attackers to be on the same monitored network in order to attack the system. The current user authentication of Blacklight with Devise can store the IP address of users who are logging into and using the front end. This information can be used to deny access and monitor logins. A more secure IR system is also more reliable for the users, an important metric for evaluating system usefulness.

6.2 User Activity

Nearly all processes of Blacklight are logged in the server log files. These files can be limited to a set size, but can be scanned for important features before deletion. Some user activities, such as a saved search or document, are already logged and stored in database tables. These tables can then be used at an appropriate time to give insight on ways to improve the relevance of documents and the user experience.

6.2.1 Relevance Feedback

Capturing the interactions of the user with the front-end enables us to find documents that a user considers relevant to the query. This means that when a user performs a search and then chooses a specific

document from the results list, they validate the relevance of that document for that query. The IR system can then modify the document's relevance to more closely match the user's determined relevance of the document. In this way, the IR system adapts over time to rank documents higher when users view them as relevant.

6.2.2 Document Impact

Analyzing documents of a collection often involves determining the importance or impact of individual documents. While relevance feedback modifies the connection of a document to query terms, document impact measures the document in relation to all other documents. This is similar to how citation counts of academic publications indicate the value of the idea. The Blacklight platform and Devise user authentication give users the action of bookmarking a document. This is effectively a citation that the IR system can use to update the document's impact or importance value. Doing so also makes the document more relevant in general, but without the need of providing specific query terms. This case would more often come about while a user is browsing the collection without search terms in mind.

6.2.3 Usage Study

Following the user behavior as they navigate the front-end also provides insight into how users effectively go about tasks to answer their questions. User behavior that matches certain patterns can indicate improvements to the retrieval system. If the user consistently searches the collection and receives zero results, then this should indicate that the IR system's collection processes needs to fill in collection 'gaps' to include documents that would be relevant to this user's query. In a different case, a very common query term from users could indicate the need for making this term into a new classification label, so that users can quickly filter results from the collection.

6.3 More Like This Request Handler

In Section 8.5.5 we covered our current progress in implementing a button to retrieve information from Solr using /mlt request handler. Our current implementation is incomplete. A future suggested approach would be to not change the request handler in Solr for /select to /mlt, but instead try "mlt=true" in "Raw Query Parameters" section in the Query UI.

6.4 GeoBlacklight and Leaflet

Section 8.5.2 covers steps to generate GeoBlacklight. Although GeoBlacklight provides a better way to find and share geospatial data, implementing it on top of the current Solr that runs on Cloudera might be not possible. The minimum version of Solr that GeoBlacklight requires is Solr version 4.7, while our current Solr version is 4.1. This causes some problems in the configurations that GeoBlacklight requires for schema.xml and solrconfig.xml.

An alternative to GeoBlacklight is *Leaflet on Rails* [25]. Leaflet is a library for generating interactive maps. To implement Leaflet we can use the JSON information that comes from Blacklight to generate geospatial data on top of different map layouts that come with Leaflet. Although it is possible to use the

Google Maps API, with Leaflet it is possible to add additional features by utilizing plugins. The first step is to define what kind of geospatial data Blacklight has access to. After knowing the data, it is possible to generate an interactive map plug-in in Rails to visualize the data.

6.5 Visualization

In Section 8.5.6 we cover how to run D3 on Rails on top of Blacklight. The next steps will be defining what kind of visualizations, parsing JSON information that comes from Blacklight, fetching that information to D3, and finally over-ridding the current view content in Blacklight or creating a new view section just for visualization and adding SVG elements for D3. Although D3 is not the only way to visualize data, it is a well-known library with a good amount of documentation and support.

7 Users Manual

7.1 Introduction

The following user manual will describe the various features supported by this information retrieval system. Future additions or modifications to the front-end should result in updates to this section. The main welcome screen can be seen in Figure 10.

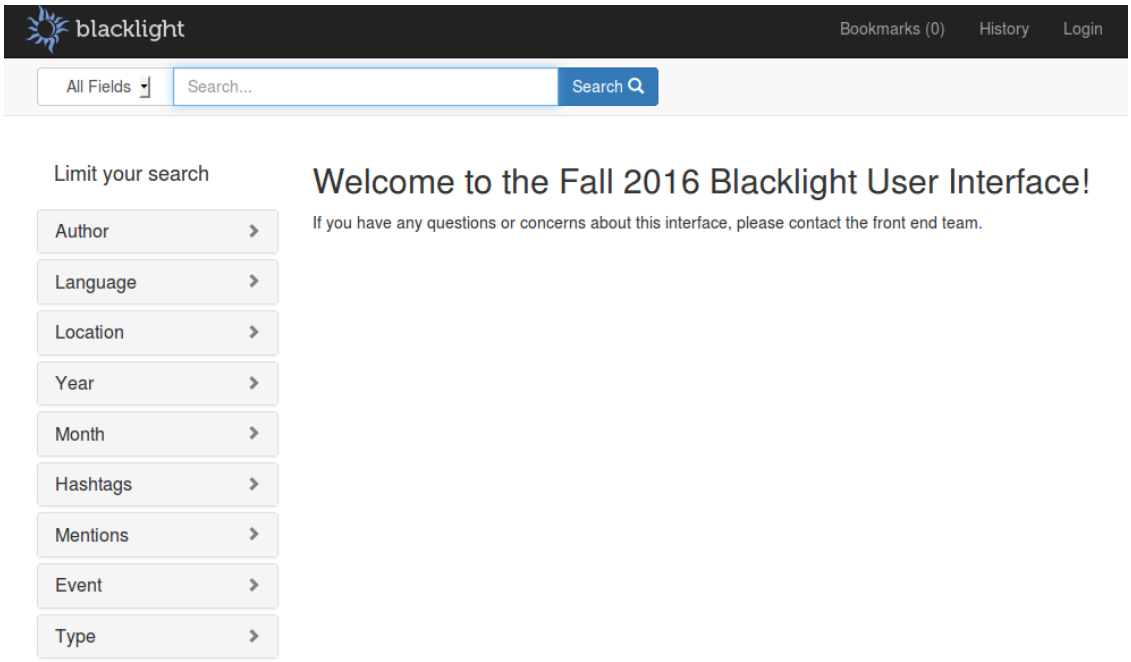


Figure 10: Welcome page

There are many customization that can be made to the interface. For example, the home screen of the interface can be branded to reflect that it supports the GETAR project instead of displaying the standard Blacklight logo. Also, the content on the homepage can be updated to reflect the organization managing the project rather than the team who has developed it. These are just some examples of future tailoring.

7.1.1 Logging In and Out

From most pages, you can navigate to the login screen by clicking 'Login' in the top-right corner of the page. This will bring you to the screen shown in Figure 11. To log into your account, enter the email and password you used to create the account. If you have not yet created an account, go to Section 7.2 to do so. Be sure to use the correct password, including capitalization, when logging in.

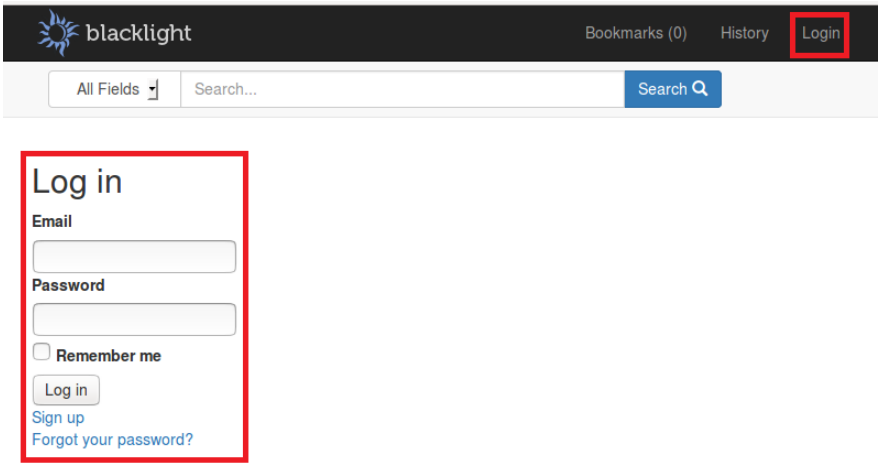


Figure 11: Login page

7.2 Creating an Account

Users who are logged into their account have additional features that they can access, such as saving document or searches. Setting up a user account is very similar to other websites. You will need an email address to create an account.

Navigate to the login screen by clicking 'Login' in the top-right corner of the page. Now click the 'Sign Up' link that appears below the form to log in as shown above in Figure 11. After clicking the 'Sign Up' link, you will see the page shown in Figure 12. Fill in your email address and choose a password. When you are done, click 'Sign Up' to have your account created. Remember your password so that you can log in for future use.

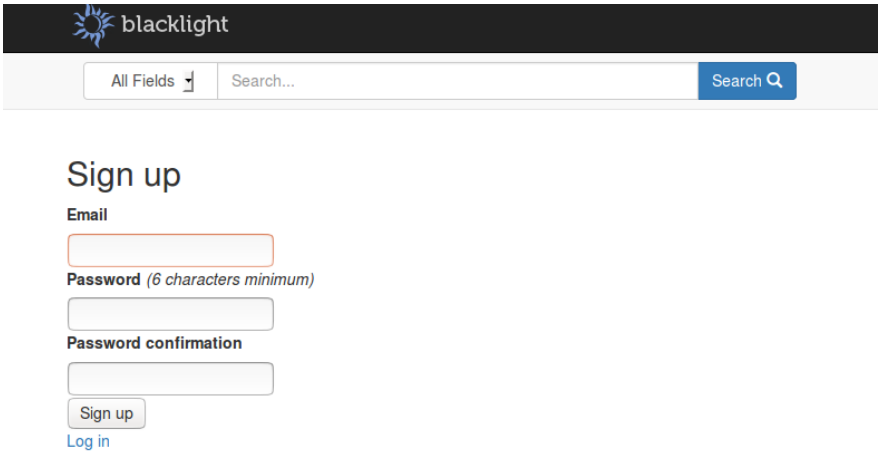


Figure 12: Sign up page

7.2.1 User Bookmarked Documents

When you are logged in, you will be able to mark documents from the results page to save and view later. On the search results page, you will see a button that says “Bookmark”. Refer to Figure 13 to see this control. Clicking this button will save that document to your Bookmarks page. To view your bookmarked documents, click the ‘Bookmarks’ menu item in the top right of the page. This will bring you to the page shown in Figure 14. From here there are controls to delete bookmarks, view a bookmarked document, or clear your bookmarks.

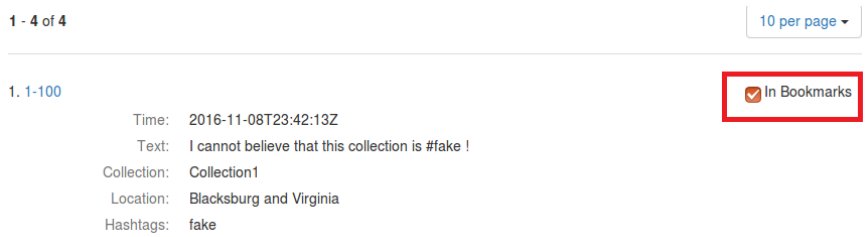


Figure 13: Results with bookmark action

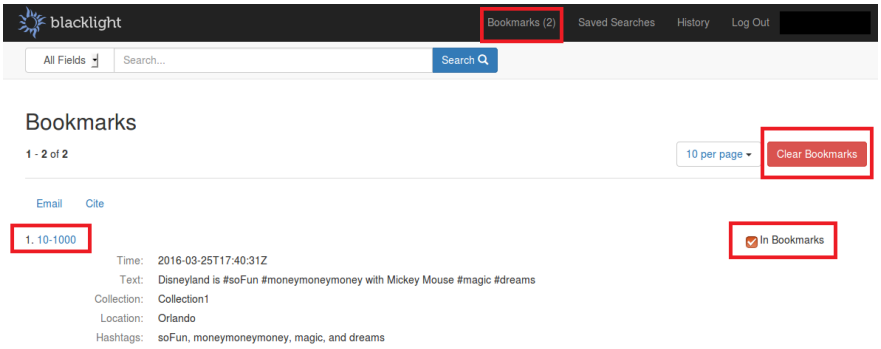


Figure 14: Bookmarks view page

7.2.2 User Saved Searches

Users with an account can also view their recent searches and save searches for later. To see your recent searches, navigate to the page shown in Figure 15 by clicking ‘Search History’ in the top-right menu. From this page, you can redo a search by clicking on it. You can also clear your recent history and save your searches for later. Saving your search keeps your search in a long-term memory that stays even if you log off. You can view your saved searches on the page shown by clicking the ‘Saved Searches’ button in the top-right menu of the page. This will bring you to a page like Figure 16. On this page you can redo your saved searches, or remove searches that you have saved.

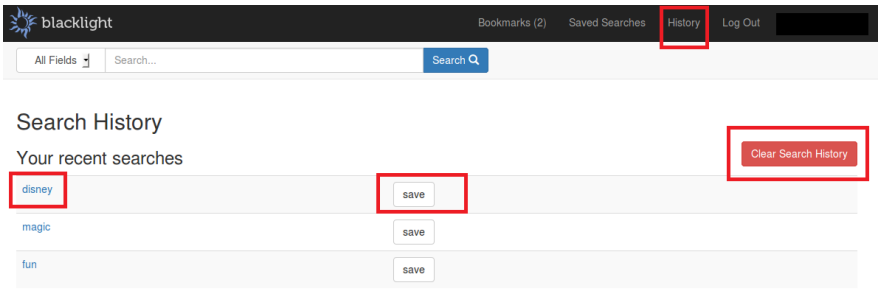


Figure 15: Search history page

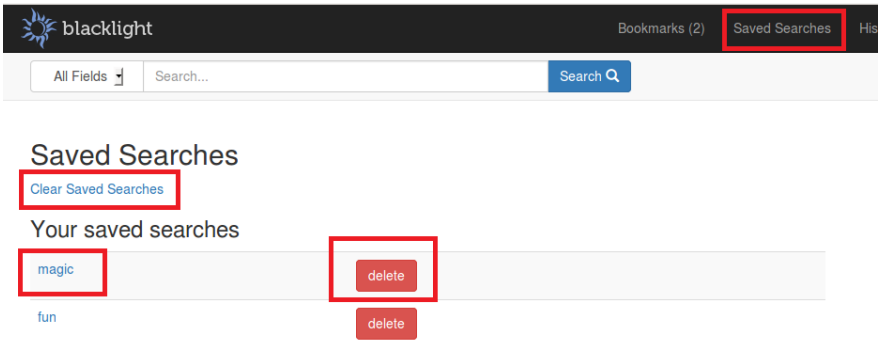


Figure 16: Saved searches page

7.3 Basic Query

One of the basic features supported by this interface is query based search. To issue a query, type keyword(s) into the search box at the top of the page. Keywords should be relevant to the types of documents you wish to retrieve. To execute the query press the 'Search' button or press the Enter key on your keyboard.This will return documents for which the query matches any data element of a document.

7.4 Filter Search Fields

By default, the interface supports queries on All Fields. This mean, the query will be compared to all data fields for each document. If you would like to search only a particular field, e.g., Author, you can limit your query to execute on a particular field by using the dropdown to the left of the query input box. Select the field you would like to search, enter keyword(s), and press the 'Search' button or press the Enter key on your keyboard. This will return documents for which the query matches that particular field.

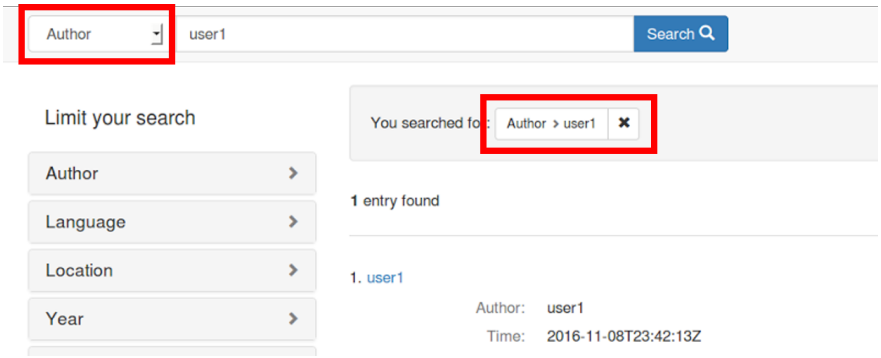


Figure 17: Filter search fields

7.5 Faceted Search

Another way to narrow search results for a query is to use search facets. On the left side of the search results page, you will see a list of facets. Selecting a facet will show only those document that are relevant to or categorized within the selected facet(s). The number next to the facet indicates how many of the documents are within that facet. These facets can be selected individually or used in combination, and multiple selections can be made per facet. You may clear facet selection by clicking on the X next to a facet selection at the top of the results page.

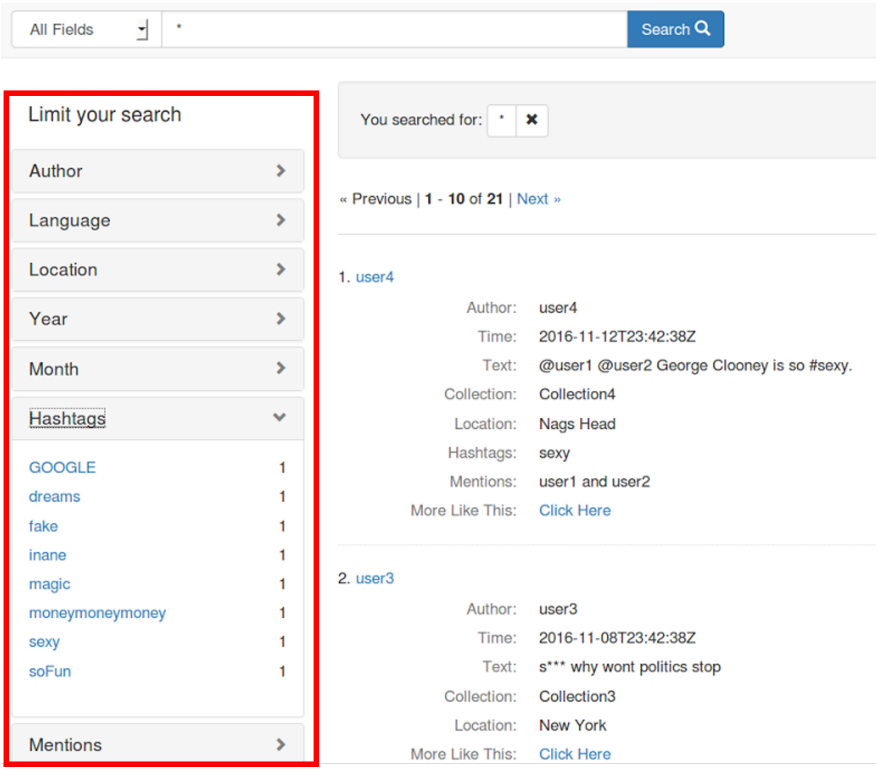


Figure 18: Faceted search

7.6 Individual Search Results

Clicking on a document in the search results page will navigate you to the individual result page for that document. This page displays detailed information about the document selected. The type of information shown depends on the type of document as well as the data that is available for that document. For this reason, the individual result page will look different for each result selected. Below is an example of an individual result page for a Tweet.

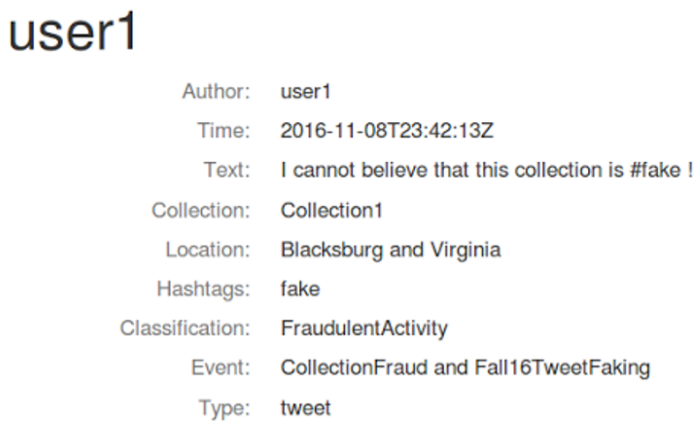


Figure 19: Individual search result tweet example

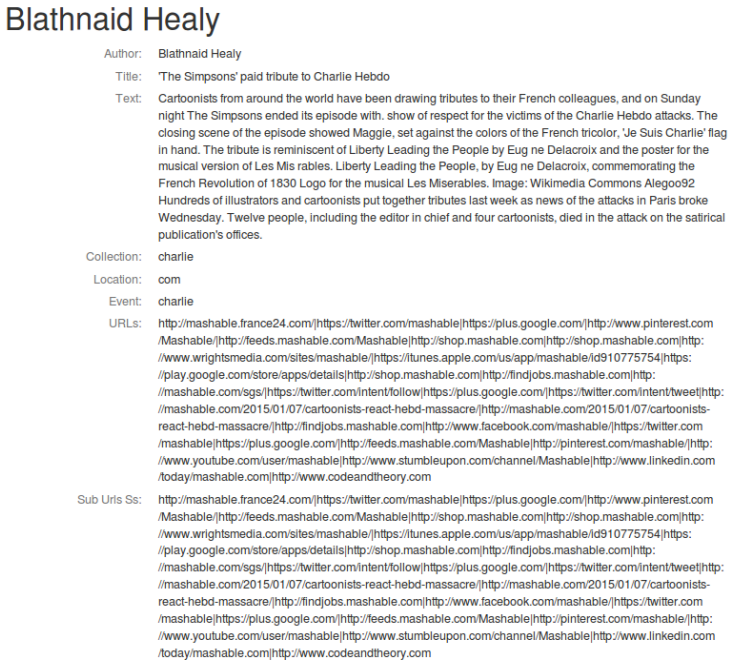


Figure 20: Individual search result webpage example

8 Developers Manual

8.1 Software Architecture

Blacklight is a front-end interface that works with Apache Solr to provide a search interface. It is a Ruby on Rails [26] engine plug-in. It works as a small application inside of an existing Ruby on Rails project. Having basic knowledge about how Rails works can help us to better understand Blacklight. Figure 21 [27] shows the Rails architecture.

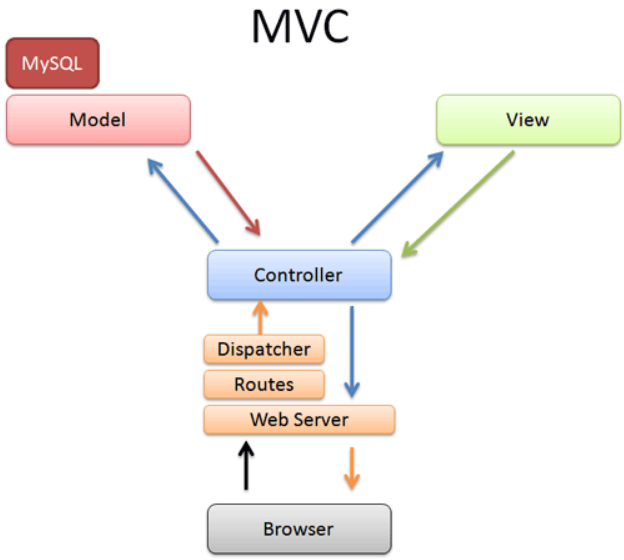


Figure 21: Rails architecture [27]

Figure 22 shows the MVC framework after creating new Ruby on Rails web-app.

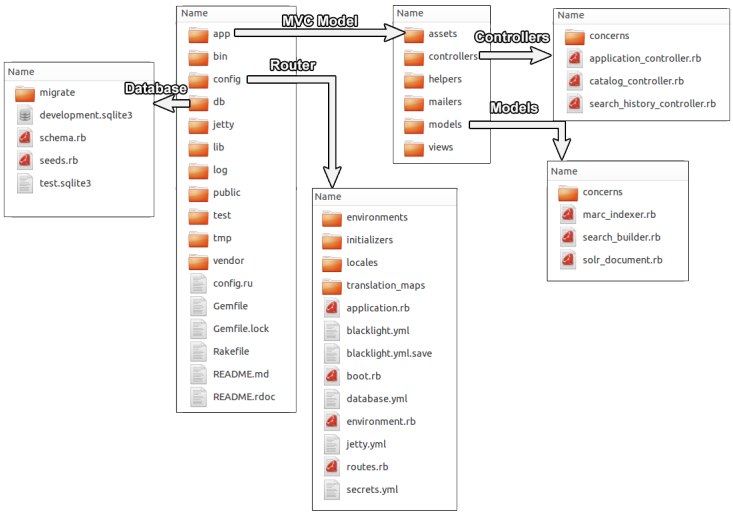


Figure 22: Rails structure

Rails follows a Model-View-Controller (MVC) pattern. In MVC, the controller parses user requests from the browser. It converts user input into a command and based on the command, it sends the result to the model or to the view. The model manages the database. It can store, query, edit or delete stored data in the database. Here, models are Ruby classes. The view is what the user sees. It can include HTML, CSS, JavaScript, JSON, etc. It receives the data from the Controller and displays it to the user.

Ruby on Rails is a set of gems (in Ruby, libraries are referred to as gems) for building modern web applications. The combination of these gems creates the Rails MVC framework. One of the Ruby gems that can be added to this framework is Blacklight. Blacklight adds functionality to Rails while conforming to the MVC architecture. Below is a description of how Blacklight adds this functionality. However, unlike Rails routes that are "mounted" and are isolated from the main application's routes, Blacklight routes are added by the generator. This provides for a more flexible user-interface.

- **Controllers:** Blacklight generates a **CatalogController** in the application. It adds "Discovery and single-term "show" actions". Inside the controller we can change Blacklight's configuration and customization.
- **Models**: Blacklight generates a SolrDocument model into the application. This model translates Solr responses to Rails models. It is a Ruby-friendly representation of Solr's responses. The SolrDocument model can provide custom model methods, accessors, and behaviors to the returned document from Solr.
- **Views:** To change the view of Blacklight, we can override the old view with the new one. To override the view, navigate to Blacklight *app/view/catalog/_per_page_widget.html.erb* to override the default view.

To determine how to map a user's input to Solr, we can use the **#processed_parameters method** in the **SearchBuilder** class. This class can add, remove, or modify a user's parameters to Solr request parameters. The output from this class can be used inside Solr as a query. Figure 23 shows the logic that Blacklight uses to map user information to Solr requests.

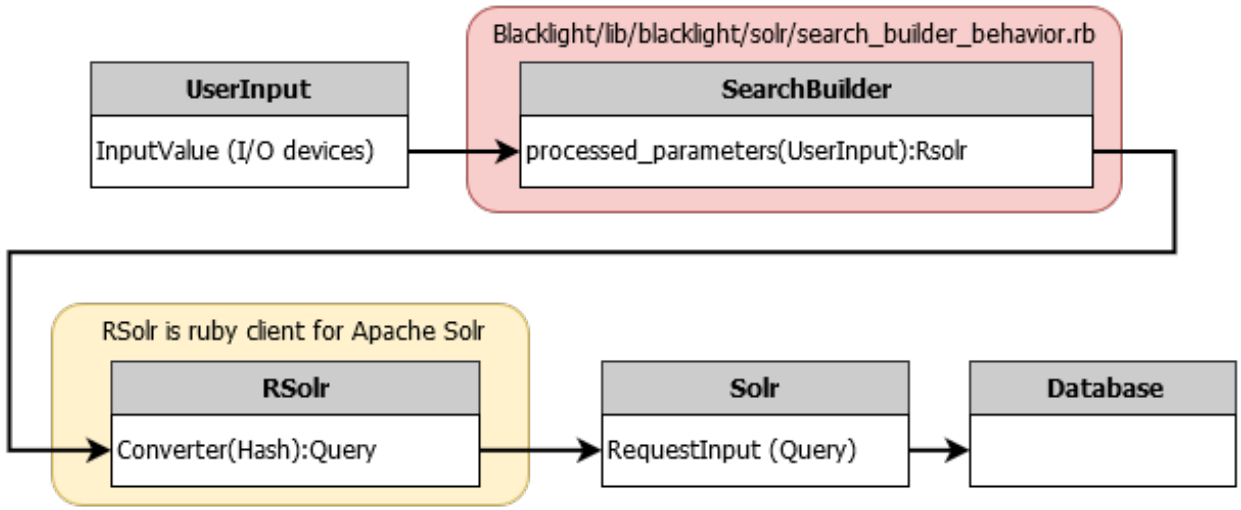


Figure 23: Blacklight mapping architecture

8.2 Installation, Operation and Maintenance

Blacklight can be run on any machine that has access to the Solr URL. This means it can be run on a client or server computer. It is important to make sure Solr runs correctly and both schema.xml and solrconfig.xml work fine. Ignoring this step might bring some ambiguity during the debugging process. The following sections cover our approaches to install and operate tools that are required to run the front-end on a server or client computer.

8.2.1 Cloudera Quickstart VM Setup

If you want to run Blacklight on a different machine, jump to Section 8.2.3, the Blacklight Installation section. It is not necessary for Blacklight to be on a Cloudera VM to connect to the GETAR Solr instance or most other Solr instances. We do not even advise doing this since it negates several advantages of using Blacklight. However, we found this to work and documented our process for doing so should the need arise to use Cloudera.

1. Install Virtual Box
2. Download Cloudera Quickstart VM for Virtual Box: http://www.cloudera.com/downloads/quickstart_vms.html
3. Run Virtual Box. File → Import Appliance. Select the OVF file downloaded from the Cloudera website.
4. Select specifications for virtual machine. *Note: CDH recommends 8+ GB RAM memory*

8.2.2 Upgrade Java on Cloudera Quickstart VM

The Java version must be 1.8 for Blacklight to be run on a server. While there are many available tutorials on this process, here we have documented how we accomplished this on the Cloudera VM. We found that there is little other documentation for this specific development environment.

1. Stop Cloudera Services and Server
   (a) Stop Cloudera Management Service (Clusters → Cloudera Management Service → Actions → Stop)
   (b) Stop all Clusters (Clusters → ClusterName → Actions → Stop)
   (c) Stop all Cloudera Manager Agents:

```
$ sudo service cloudera-scm-agent stop
```

   (d) Stop Cloudera Manager Server:

```
$ sudo service cloudera-scm-server stop
```

2. Install Java
   (a) Choose a Java JDK version to install. At the time of this report, the newest JDK SE version is 1.8.0_102. Cloudera only supports Java up to version 1.8.0_060. Cloudera Quickstart VM includes Java version 1.7.0_67. Blacklight requires Java version 1.8.
   (b) Download the the Linux RPM file. Archives of previous Java versions can be found here: http://www.oracle.com/technetwork/java/javase/archive-139210.html
   (c) Install the RPM using YUM:

```
$ yum install jdk-8u77-linux-x64.rpm
```

3. Configure Cloudera for Java
   (a) Ensure that symbolic links in the `/usr/java` directory correctly point to upgraded Java. Future steps will use the symbolic link named 'default'. Use `$ls -l` command to view links
   (b) Open this file in any text editor: `/home/cloudera/.bash_profile`
   (c) Add this line to the end of the file: `export JAVA_HOME=usr/java/default`
   (d) Open this file in any text editor: `/etc/default/cloudera-scm-agent`
   (e) Add this line to the end of the file: `export CMF_AGENT_JAVA_HOME=usr/java/default`
   (f) Open this file in any text editor: `/etc/profile`
   (g) Modify this line in the file: `export JAVA_HOME=usr/java/default`

(h) Modify the PATH environment variable to direct to the newer Java version. View PATH with: `$echo $PATH` . Ensure this variable is saved and not lost upon closing the terminal.

(i) Configure the location of the JDK on cluster hosts. `Hosts` → `Configuration` → `Java Home Directory`, and set it to `/usr/java/default`

4. Start Cloudera Services and Server in reverse order of the stop process. Replace `stop` with `start` in commands.

8.2.3 Blacklight Installation

This installation creates the system overview seen in Figure 7. These instructions cover gem version 6.7.2 of Blacklight. This is the demo that Blacklight provides, and not the setup to connect to the GETAR cluster, but the demo can be configured after setup to connect to the GETAR Solr instance. Before installing Blacklight, we need to make sure that the system has all the dependencies. It is recommended to follow these steps on the Linux command line. First we need to install Ruby on the system. To install Ruby, first check if there is Ruby already installed on the current system. At the time this report was written, Blacklight requires Ruby 1.9 or higher.

```
$ ruby --version
ruby 2.3.0p0 (2015-12-25 revision 53290)
```

To install Ruby on your system, follow the steps in the link below. If you do not need version control, skip the "Configuring Git" section in the instructions. It does not matter which method, rbenv or rvm, you choose to install Ruby. Both methods work fine.

https://www.ruby-lang.org/en/documentation/installation/

After installing Ruby, we need to add the Rails framework. At the time this report was written, it is recommended to use Rails version 5.

```
$ rails --version
Rails 5.0.0.1
```

Here is a link for installing Rails:

https://www.tutorialspoint.com/ruby-on-rails/rails-installation.htm

Also check to see if Java version 1.8 or higher is installed on the system. See section 8.2.2 for information regarding upgrading Java.

```
$ java -version
java version "1.8.0_77"
```

To install Java on your system, follow the steps in the link below:

https://docs.oracle.com/javase/8/docs/technotes/guides/install/linux_jdk.html

After installing all dependencies, we need to create a new Rail application. In the terminal, choose the directory in which you want to install the project and input:

```
$ rails new my_blacklight_app
$ cd my_blacklight_app
```

After installation, locate "Gemfile" in the my_blacklight_app folder and open it with any text editor such as Vim, Nano, Sublime, Notepad++, or gedit. I highly recommend installing Atom on your local machines to use as your text editor.

https://atom.io/

The purpose of the Gemfile is to add different Gems into a Ruby on Rails application. Here you can see different gems being used for the application or specify what version should be used. We are going to use Gemfile for adding any extra plug-ins or add-ons, such as "GeoBlacklight" or "Blacklight Range Limit". But the main dependency we need is Blacklight, so we include this line in the file (note that it is not commented):

```
gem 'blacklight', "~> 6.0"
```

Save the file. In the terminal, update the bundle. Bundler is another gem that comes with any Ruby on Rails application. The purpose of this gem is to add all the dependencies for each gem. The way Bundler works is a developer specifies the gems inside of the Gemfile. After saving the file and running bundle install in the command line, it will add all the required gems in the `Gemfile.lock` file. Leave `Gemfile.lock` as it is. This file has all the information about every gem and their versions that are currently being used inside the application. After running `bundle install` in command line and adding all the required gems to the application, we need to install Blacklight to add its functionality on top of Rails.

```
$ bundle install
$ rails generate blacklight:install --devise --marc --jettywrapper
```

This is a default setup that comes with Blacklight.

The "devise" command will add User Authentication into Blacklight using Devise. The following link describes how Devise works.

https://github.com/projectblacklight/blacklight/wiki/User-Authentication

Including "marc" will add "Machine-Readable Cataloging" [28] into Blacklight.

"jettywrapper" provides Rake [29] tasks for starting and stopping jetty. If you are using the Jetty Solr that comes with the Blacklight, you need to run this command. If you are connecting to another Solr version,

you do not need to worry about this command. I recommend first installing the default Blacklight version with all the functionality to make sure it runs without any problems.

For more information about Rake and how it works please follow link below.

Link to RAKE - Ruby Make

By running database migrations, Blacklight will create its own database table. Make sure to run this command!

```
$ rake db:migrate
```

The Solr that comes with Blacklight runs on Jetty. To download Jetty, use the following command:

```
$ rake jetty:clean
```

Now, Blacklight is installed and can be used. To test it, first go to the folder in which Blacklight is installed. Run the Solr server on the system.

```
$ rake jetty:start
```

You can navigate and check your Solr server at http://localhost:8983/solr/#/blacklight-core/query. Remember this is a demo Solr that comes with Blacklight and has nothing to do with the Solr version that we want to connect to. Blacklight is capable of connecting to any Solr version.

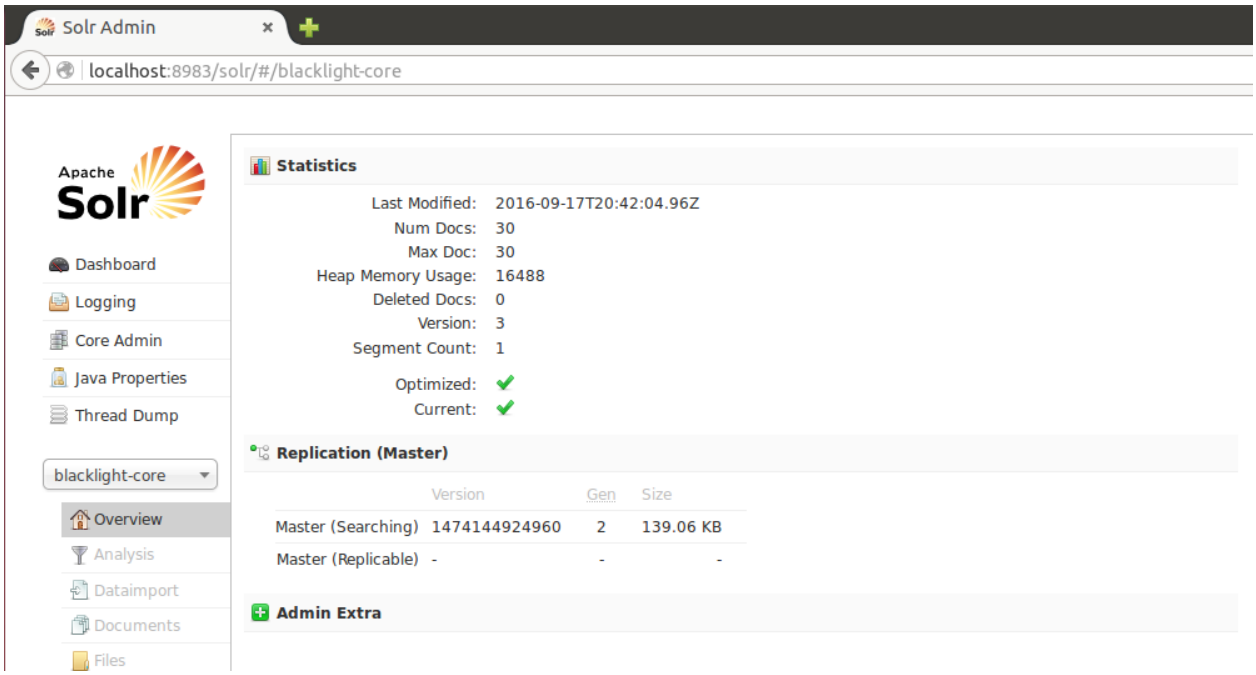


Figure 24: Solr example setup for Blacklight

To index some data, use the command below to index the MARC record provided with Blacklight.

```
$ rake solr:marc:index_test_data
```

After this, we can start Blacklight.

```
$ rails server
```

Now visit http://localhost:3000/catalog to see the Blacklight interface.

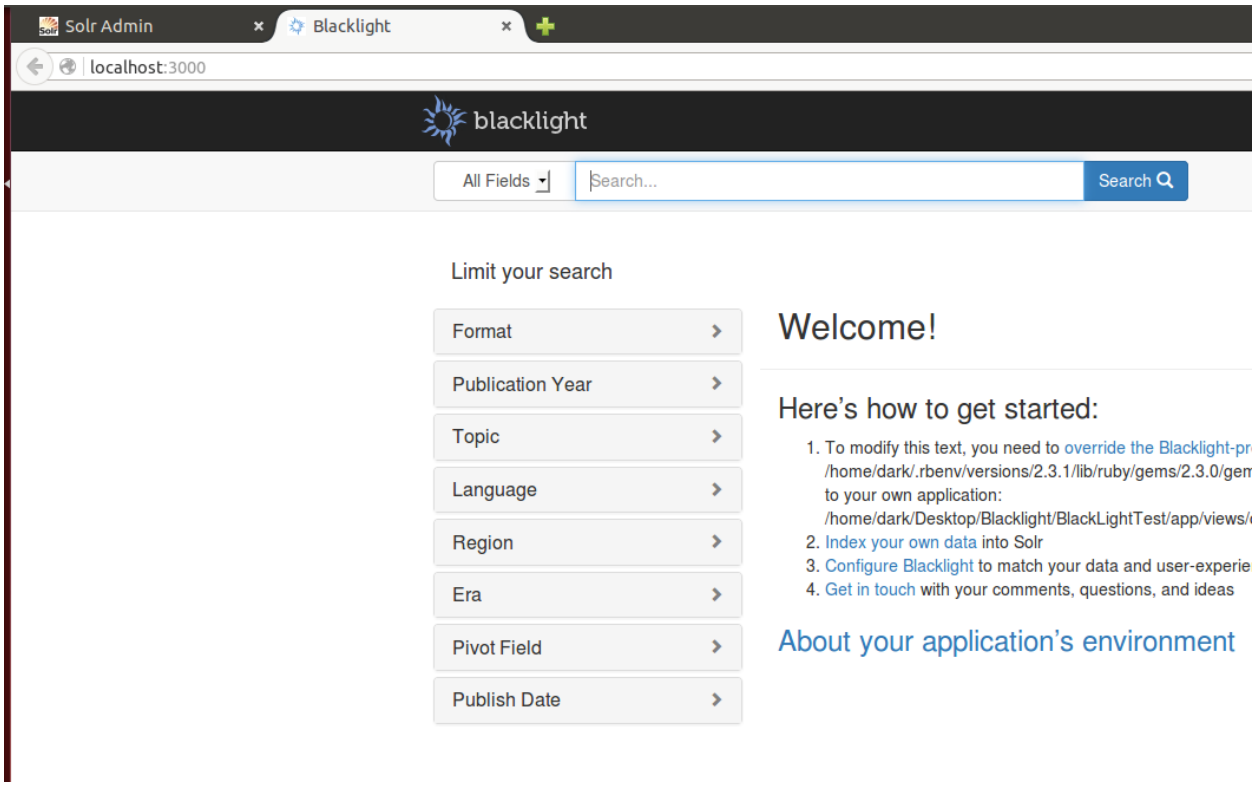


Figure 25: Blacklight front page

The URL below explains how to configure Blacklight for a Solr connection:

https://github.com/projectblacklight/blacklight/wiki/Solr-Configuration

8.3 Blacklight Configuration and Setup

Before starting Blacklight's configuration, it is important to know what kind of information we want to show. For this reason, the first step is to know how Solr works. Before approaching the next section, review *Understanding Solr* in Section 4.2 to understand how fields works in Solr. Next, we will cover the steps to set up Blacklight to connect to the core in Solr.

8.3.1 Connecting Blacklight to Solr

As mentioned before, Blacklight can connect to any Solr that the browser has access to. For our demonstration, we are going to connect to the main server with big data.

After installing Blacklight, we connected to the Solr service on the cluster using the following command.

```
$ ssh -L 9983:solr2.dlrl:8983 <user>@hadoop.dlib.vt.edu
```

In this command, we specified the host name and port number for the Solr service. By doing so we are now able to connect to Solr by entering ”localhost.9983/solr” in the address browser (Figure 26).

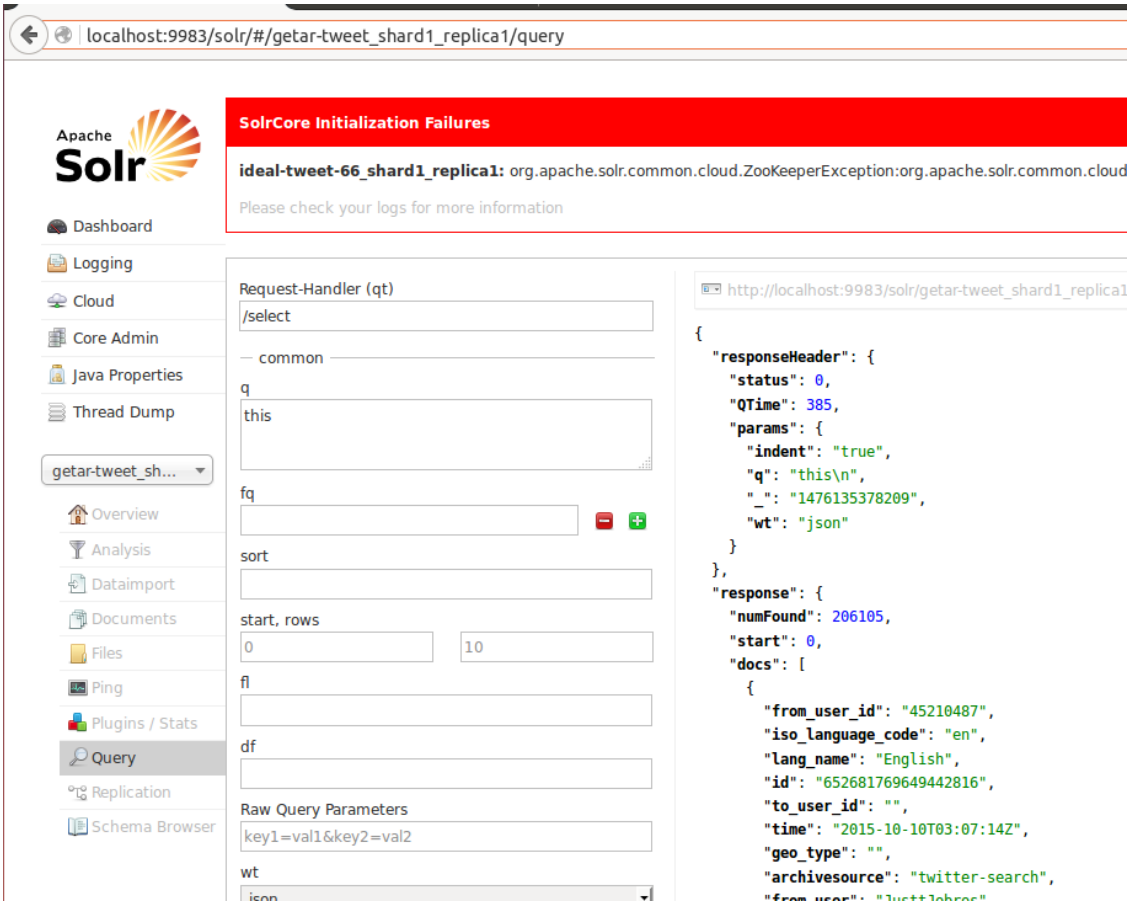


Figure 26: Connected to Solr service on cluster using SSH

You can check different cores available inside the Solr by clicking on “Core Selector”.

The next step is to set up the Solr URL in Blacklight. The Solr connection parameters are configured in `config/blacklight.yml` in the Blacklight directory. To locate this file, navigate to your Blacklight application directory where you installed it. Figure 27 shows the command and blacklight.yml.

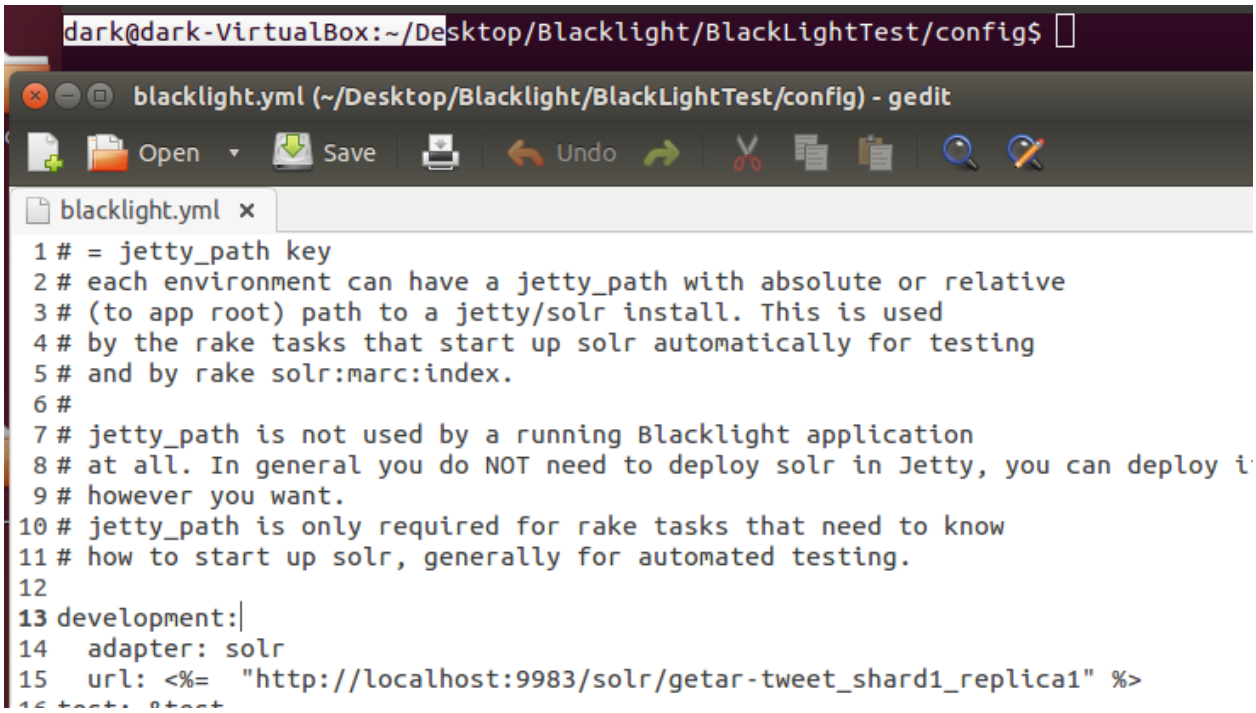


Figure 27: Blacklight.yml file

Inside `blacklight.yml` change the address for development from the default URL to the new Solr URL. This URL needs to include the core name that we want to use. For example, in our case, we want to connect to the "getar-tweet_shard1_replica1" core. To be able to do so, in our blacklight.yml we input the URL to the core in development URL, as follows:

```
localhost:9983/solr/getar-tweet_shard1_replica1
```

Make sure there is no "#" from the browser's address inside your blacklight.yml file.

There are three sections inside blacklight.yml to insert the path file for Solr. They are: development, test, and production. By default, `rails server` runs in the development environment, which means it will only look at the URL for Solr's core in the development section. If you want to have access to more than one core, insert the path to different cores in the test or production sections. You can change the Rails environment from development to test, by running the following command:

```
$ rails server -e test
```

Having access to different environments gives us the option to connect to different cores in Solr without creating new Ruby on Rails project.

```
development:
  adapter: solr
  url: <% "localhost:9983/solr/getar-tweet_shard1_replica1" %>
test: &test
  adapter: solr
  url: <%"localhost:9983/solr/getar-tweet_shard1_replica1 %>
production:
  adapter: solr
  url: <% "localhost:9983/solr/getar-tweet_shard1_replica1" %>
```

Now we are ready to run our `rails server` command. In the directory you installed Blacklight, run the following command:

```
$ rails server
```

You may be able to connect to Blacklight by visiting the following URL:

`localhost:3000`

It is more likely though that you will not yet able to connect to the Blacklight interface in your browser. The reason for this is that we are working on the demo version of Blacklight and the name of fields inside Solr do not match the ones that comes with the demo. In the next section, we cover how to solve this problem as part of the user interface implementation.

8.4 User Interface Implementation

Blacklight is an open source user interface framework that we have chosen to use to develop our front-end interface. It is customizable using Rails templating [30] and uses Bootstrap [31] for many of the interface components. Integration between the Rails layout and the Bootstrap interface is completed using the bootstrap-sass gem [32]. One of the major advantages of this method is allowing development and customization of the user interface using DRY principles [33] and reducing the need for regular version upgrades. Implementation of the user interface using these tools and techniques will be covered in this section.

8.4.1 Configuring Blacklight Controller

The next step is to configure Blacklight's CatalogController.rb. All the configuration for facet fields, index fields, Solr search params logic, etc. go in CatalogController. By default, this file is located at *app/controllers/catalog_controller.rb* in the directory you installed Blacklight. The default CatalogController.rb will not work with any Solr core other than the demo that comes with Blacklight. We need to change the name of the fields based on the core that we want to display in our front-end. If we run Blacklight using `rails server` without going through this step, we will receive the following error (Figure 28) in the browser. See Figure 29 for an overview of the Blacklight and Solr on GETAR cluster configuration.

```
raise Blacklight::Exception::ENCONNREFUSED, "Unable to connect to Solr
 ↪  instance using #{connection:inspect}: #{e.inspect}"
```

Figure 28: Invalid request in CatalogController

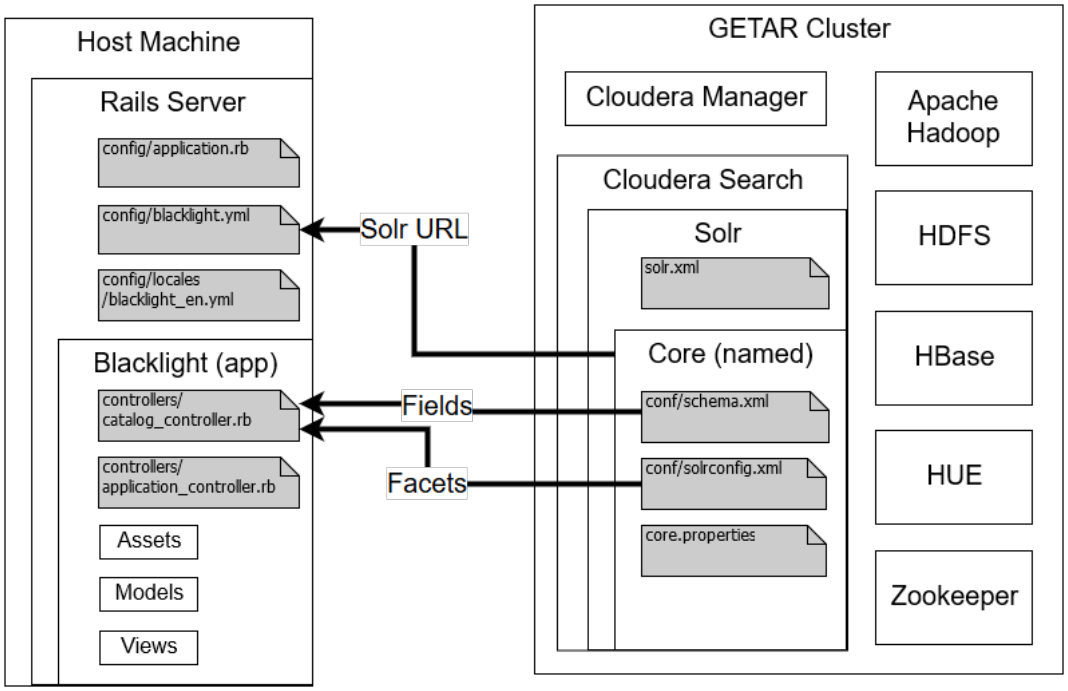


Figure 29: Blacklight configuration with Solr instance in GETAR

CatalogController.rb is well documented and it is easy to understand what each function can do. We need to make sure every field that we are using is already implemented correctly inside schema.xml.

For example, to add a facet field, first make sure the field is available and it is indexed by look-

ing at “Schema Browser” (Figure 3) in the Solr UI. Here we used “col_name” and “colnum” fields from schema.xml as facets.

```
config.add_facet_field 'col_name', label: 'Column Name'
config.add_facet_field 'colnum', label: 'Column Number'
```

Make sure the following command in CatalogController is not commented out.

```
config.add_facet_fields_to_solr_request!
```

To display search results in the view, we can add different fields in the order that we want. The example code below shows how we added four different fields to the search result view.

```
config.add_index_field 'from_user', label: 'From User'
config.add_index_field 'created_at', label: 'Created at'
config.add_index_field 'geo_coordination_1', label: 'Geo Coordination'
config.add_index_field 'cleantext', label: 'Text'
```

Figure 30 shows the result.

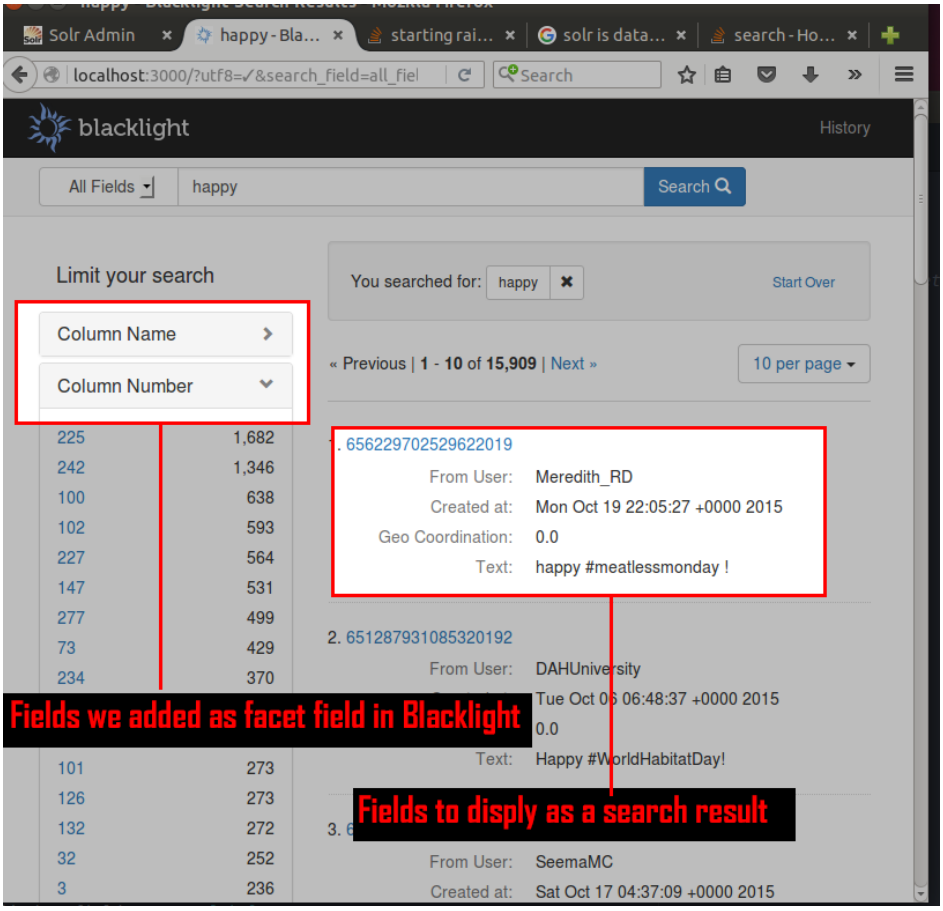


Figure 30: Blacklight interface after adding different fields

For a single document request (detail view) in *config.default_document_solr_params* in CatalogController.rb, we can define a different request handler from solrconfig.xml as qt parameter. "fl", "rows", and "q" are some of the common query parameters inside Solr. "fl" can be used to specify a set of fields to return, limiting the amount of information in the response. When returning the results to the client, only fields in this list will be included. Normally we return all fields. "rows" defines number of rows to be displayed and "q" is query parameter.

```
config.default_document_solr_params={
  ## qt: 'document', we can add custom requesthandler
  fl: '*',
  rows: 1,
  q: '{!term f=id v=$id}'
}
```

To add different fields in single document view, we can select fields and add them in CatalogController.rb as follows. Here we used "from_user" and "text" fields to be displayed in our single document:

```
config.add_show_field 'from_user', label: 'From User'
config.add_show_field 'text', label: 'Text'
```

Figure 31 shows the result.

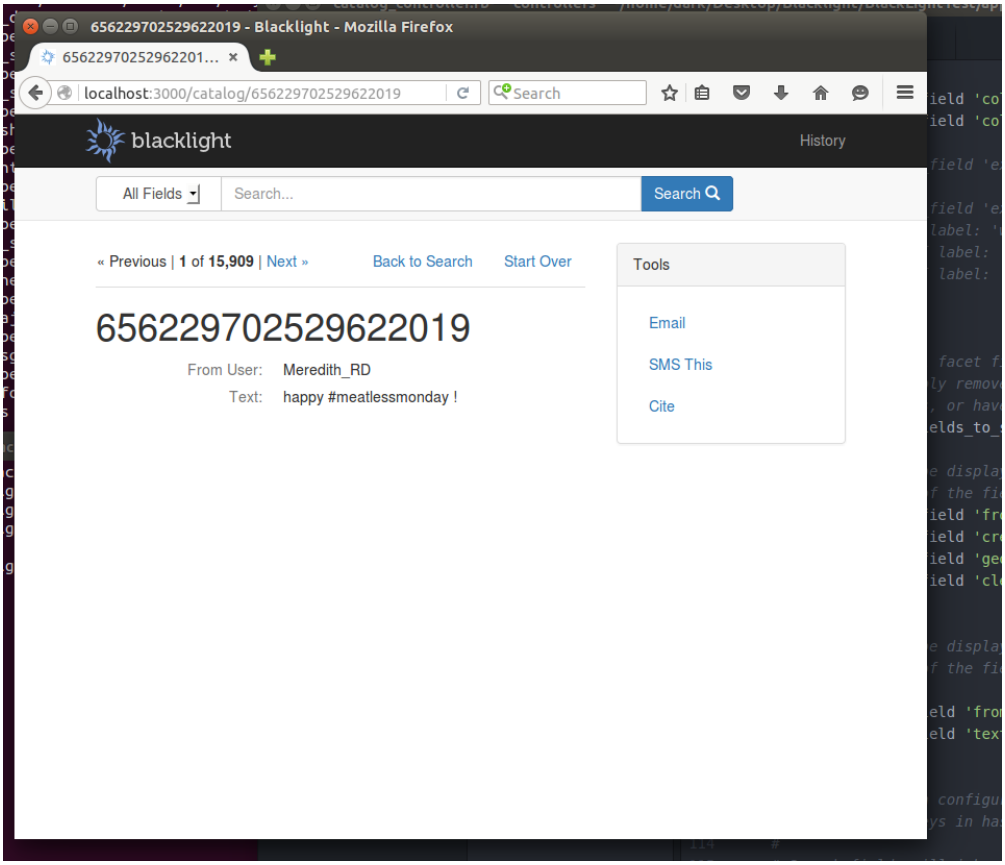


Figure 31: Blacklight single document interface

At this point, we are able to show different fields inside schema.xml in our Blacklight's interface.

8.5 Configuration and Setup for Other Tools

8.5.1 Devise and User Authentication

Devise [17] is the built-in user authentication platform of Blacklight with some additional features. It uses SQLite3 to store its data in database tables. Other authentication frameworks are compatible, but require additional configuration. All of this basic configuration is provided when building the application by running the `rails generate blacklight:install` **`--devise`** `--marc --jettywrapper` command.

Devise uses a schema file for defining the database tables it uses. In the Blacklight application, it is in the `db/schema.rb` file. This file can be configured to store data which is important to be saved in a database table.

Developers can use SQLite3 to view the contents of the database tables generated by Devise and the schema. Figure 32 shows a quick demonstration of viewing some of the data Devise stores. The `db/development.sqlite3` file is a database that contains all of the tables Devise uses.

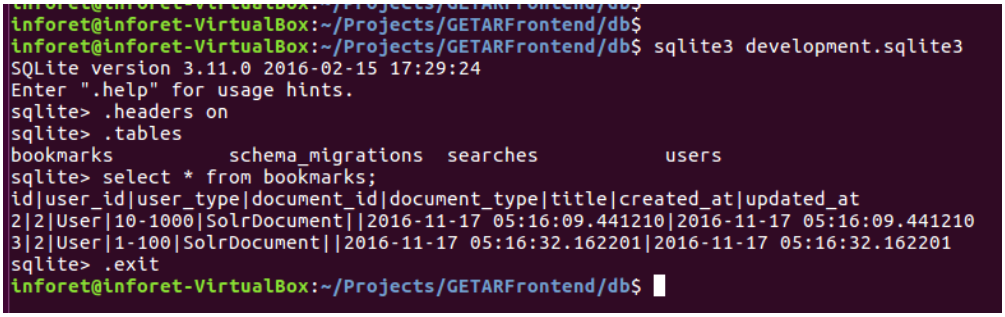


Figure 32: Access to SQLite3 database and tables made by Devise

8.5.2 GeoBlacklight

GeoBlacklight is a stable add-on for the Blacklight project. It provides an effective open-source application for discovery of geospatial data. Geographic Information System (GIS) data are structured data in specific file formats (Shapefiles, Rasters, etc.). This information goes through a spatial data infrastructure (SDI).

The minimum required metadata for GeoBlacklight is a bounding box and description. No references to services that will provide that data are required.

GeoBlacklight extends the functionality of Blacklight by providing the following:

- spatial search with a spatial relevancy algorithm
- download functionality for geospatial web services
- map view of search results
- easily customizable
- extendable to new types of data and functionality

To install a new GeoBlacklight Rails application, run the following command in the console. It is highly recommended before installing this gem to create a backup of your current project. This gem will completely change your current Blacklight project in the view, controller and model sections of your app.

```
$ rails new app-name -m https://raw.githubusercontent.com/geoblacklight/
  ↪  geoblacklight/master/template.rb
```

then:

```
$ cd app-name
$ rake geoblacklight:server
```

Following these steps will make new a GeoBlacklight application on your machine. After this, you can connect to `localhost:3000` to see the demo version (Figure 33).

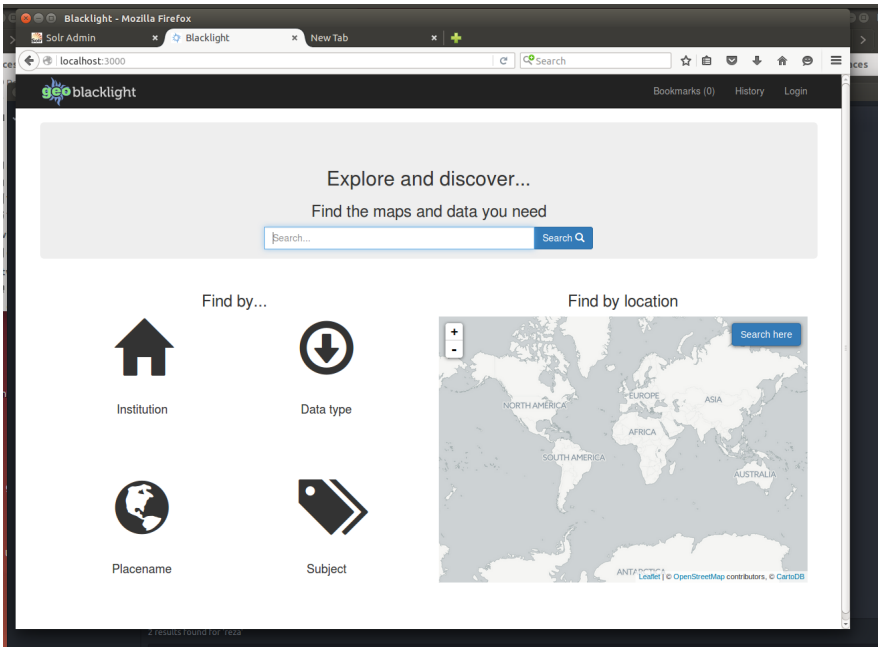


Figure 33: GeoBlacklight interface

To install GeoBlacklight as add-on on to an existing Blacklight, add the following command to your Gemfile inside the Blacklight directory:

```
gem 'geoblacklight', '>= 1.1.2'
```

Save the file and in the terminal run:

```
$ bundle install
```

This will add GeoBlacklight on top of your current Blacklight. This method is not recommended because it will not include all the dependencies. The main problem with GeoBlacklight is that the new install of GeoBlacklight runs on Solr version 6, and the older version of this gem requires Solr version 4.7. Our Solr version on Cloudera is version 4.1. This causes serious problems in making cores work inside of Solr

version 4.7. There are different configurations on Solr.xml, Solrconfig.xml and schema.xml between Solr version 4 and 6. Also the Zookeeper configuration is already done inside a new install of GeoBlacklight. This means classes such as *Solr DefaultzkCredentialsProvider* and *Solr DefaultzkACLProvider* are already set up and working with Zookeeper.

For more information about Zookeeper and Solr, visit the link below:

https://cwiki.apache.org/confluence/display/solr/ZooKeeper+Access+Control

This information can be accessed inside solr.xml.

Besides this, after comparing Schema.xml between the default Blacklight and GeoBlacklight, we found that the **location_rpt** fieldType inside schema.xml does not work on version 4.1.

For more information and examples about GeoBlacklight visit the following URL:

http://geoblacklight.org/

8.5.3 Date Range Limit

Another add-on for Blacklight is Range_limit (Figure 34). This provides a better user experience for facet search results.

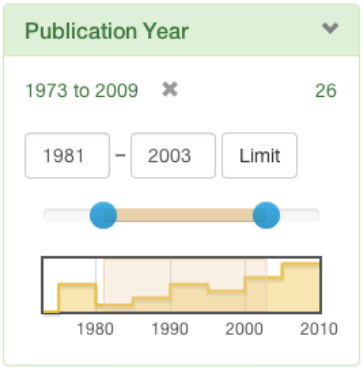


Figure 34: Data range interface

To implement Range Limit in Blacklight, simply add the following line in the Gemfile. You can find the Gemfile in the folder where you installed Blacklight.

```
gem "blacklight_range_limit"
```

Save the Gemfile and in terminal run:

```
$ bundle install
```

then:

```
$ rails generate blacklight_range_limit:install
```

Now in *Catalog_Controller.rb*, for any field that can be sorted, you can add *range: true* to add range limit functionality. For example, inside Catalog_Controller.rb we can have:

```
config.add_facet_field 'pub_date', label: 'Publication Year', range: true
```

Based on the documentation for the range limit add-on, it may be more efficient to use fields with "tint" as their fieldType.

For more information about Range Limit visit the following URL:

https://github.com/projectblacklight/blacklight_range_limit

8.5.4 Blacklight Advanced Search

Blacklight advanced search gem adds some functionality on top of the current Blacklight search. Some of them are as follow:

- Having access to multiple request handlers
- Hide or display different search fields in advanced search section or in the simple section
- Expression parsing in ordinary search using AND/OR/NOT

It is important to note, for the current gem version (6.1.0), Data Range Limit does not work properly with this gem and it requires some hacking to be able to get the result. We will cover this later. It is highly recommended before installing this gem to create a backup of your current project.

To install the advanced search gem, first add it to your Gemfile.

```
gem "blacklight_advanced_search"
```

Then, install the gem by running the command:

```
$ bundle install
```

Next, install the gem inside the Blacklight project.

```
rails generate blacklight_advanced_search
```

This will add some changes to *routes.rb*, *catalog_controller.rb* and *search_builder.rb* in your current project and add an advanced view section to your project. Removing this gem might cause some problem because of these changes.

This gem will add a **More options** button close to the search field in the Blacklight UI (Figure 35).

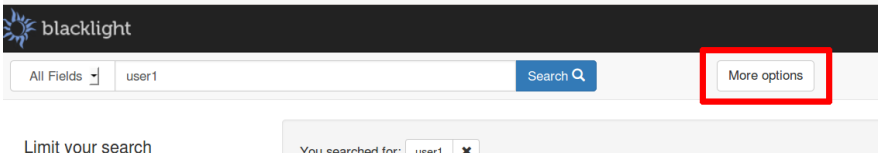


Figure 35: Advanced Search

If you are using the Data Range Limit gem with advanced search, you will currently receive the following error when you click on the "More options" button.

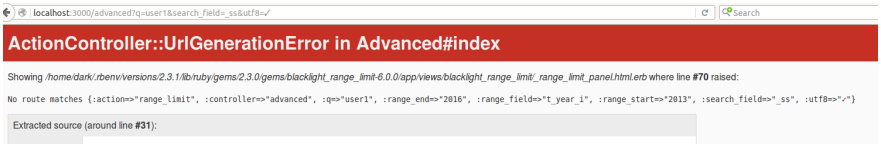


Figure 36: Data Range Limit error

To fix this problem, follow these steps:

Create a folder in *yourBlacklightProject/app/views/* and name it **advanced**. Inside the folder, create a document file and rename it to **_advanced_search_form.html.erb**. By putting any HTML tag in this document file, you will over-ride the advanced-search-form in your Blacklight project. Copy and paste the contents of blacklight_advanced_search from the following URL:

Link to _advanced_search_form.html.erb on the Project Blacklight Github

In the file, right after the following HTML tag

```
<%= render 'advanced\_search\_facets' %>
```

paste this:

```
<div class="form-group advanced-search-facet">
        <%= label_tag "publication_date", :class => "col-sm-3 control-label"
 ↪  do %>Publication Year<% end %>
         <div class="col-sm-9">
```

```
            <%= render_range_input("t_year_i", :begin) %> -
↪  <%=render_range_input("t_year_i", :end) %>
        </div>
</div>
```

t_year_i is our range limit facet index that we used in Range Limit. Change it to the one you want to use in your project. Finally, in your *routes.rb*, add the following lines for the router to work correctly and retrieve the data.

```
get 'advanced' => 'advanced#index', as: 'advanced_search'
match 'advanced/range_limit', :to => 'advanced#range_limit', :as =>
  ↪  'catalog_range_limit', :via => [:get, :post]
```

After this the "More options" button should work fine. Figure 37 shows the result.

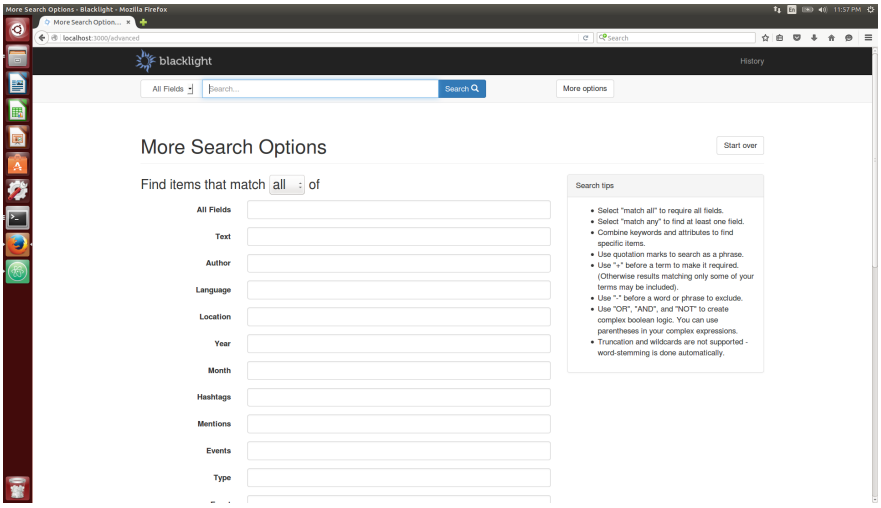


Figure 37: Advanced search interface

8.5.5 Implementing More Like This Request Handler

In our Catalog_controoler.rb, we tried to add extra functions to get the query from /mlt request handler inside Solrconfig.xml. Although we are sending all the correct information to retrieve the data, we are not able to receive the correct query. Here we cover our approach for creating a new button to change the request handler from /select to /mlt.

In the catalog_controller.rb, we added a new index field that gets the `id` field of the document. Then we created a new helper method named *more_like_this* inside *./yourBlacklightApp/app/helpers/application_helper.rb*. Any helper methods need to be added in this Ruby file. Our code inside application_helper.rb looks like this:

```
  def more_like_this options={}
        result = options[:document]
  #result = result + '/mlt?q=id%3A'

    link_to "Click Here",  customSearch(result)
  end

  def customSearch(result)

      return solr_local_parameters = {
          qt: 'mlt',
          df: 'id',
          q: result
        }
  end
```

This method takes the document's `id` as a query and changes the request handler to mlt. For default field, we use "id". In our catalog_controller.rb we have the method as follow:

```
config.add_index_field 'id', label: 'More Like This', :helper_method =>
 ↪  :more_like_this
```

This line of code adds a new button after each result with a "More Like This" label. We implemented this method, and in our Rails server log, we see that the information that we send to Solr is correct, but we receive the wrong data back.

We do **not** recommend changing the request handler to /mlt using "qt:" in a helper function as it is likely that changing the request handler is causing this problem. Instead, find a way to define *mlt=true* in "Raw Query Parameters" in the Query section of Solr (see Figure 38). This way, while our request handler is */select*, we might be able to include the result of the */mlt* handler in our search.

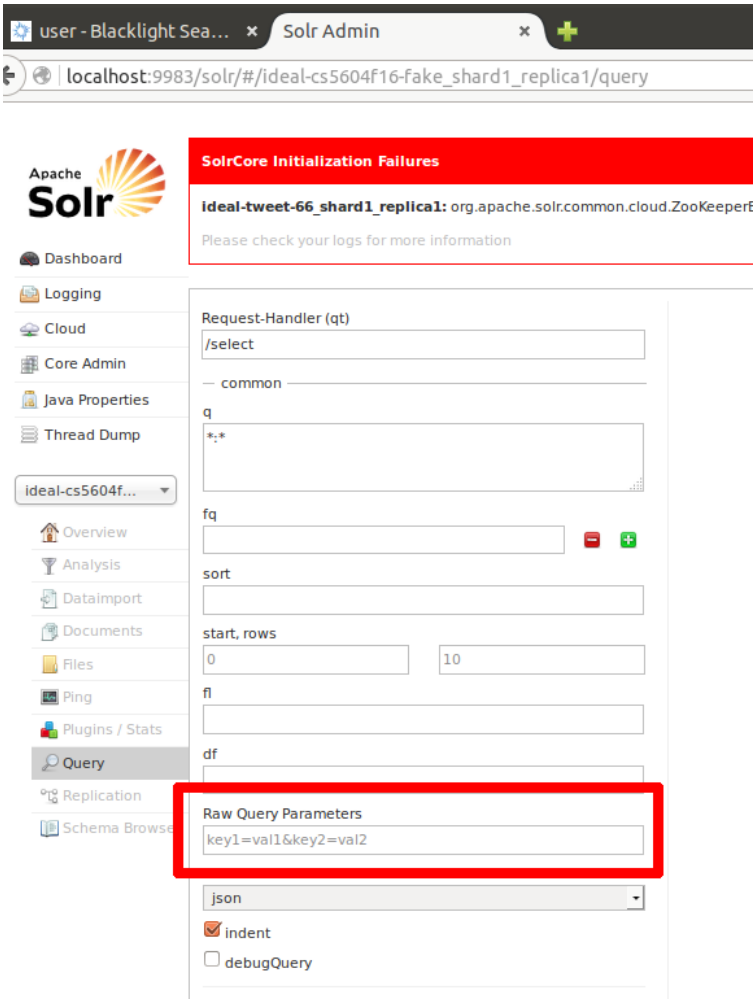


Figure 38: Raw query parameters inside Solr

8.5.6 D3 on Rails

One of the features of Blacklight is to be able to receive JSON information for both search and facet results. For example, by using the following address in the browser, we can get all the JSON information about that page.

For getting search results: `/catalog.json?search_field=all_fields&q=OurQuery`

For getting facet list results: `/catalog/facet/subject_topic_facet.json`

This gives us possibilities to try different gems inside our Rails application on top of Blacklight to visualize data with current information without changing schema.xml or solrconfig.xml in the Solr.

Here, we will cover how to install D3 [7] on top of Blacklight. With D3 it is possible to visualize current information in many different forms.

Download the latest version of D3 from this link: https://d3js.org/

From the zip file, copy d3.js or d3.min.js and paste it into *./YourBlacklightApp/vendor/assets/javascript*. Any third party library needs to be placed in this directory. Next, like the other gems we have covered so far, in the **Gemfile** add D3 for Rails.

```
gem 'd3-rails', '~> 3.5', '>= 3.5.17'
```

Save the Gemfile and run "bundle install".

Beside this, we need to add D3 to our application's javascript file. Go to *./yourBlacklightProject/app/assets/javascripts* and open **application.js**. Add the following line to the file so the application knows one of the dependencies is D3.

```
//=require d3
```

Next, create new routers inside our routes.rb file. In *./yourBlacklightApp/config/routes.rb* we create two simple routes.

```
  get 'graph/d3'
  get 'graph/data', :defaults => { :format => 'json' }
```

When we enter *localhost:3000/graph/d3*, the routes.rb will get the URL and it will look for the "graph" controller with a function called "d3" or "data". We can rewrite the code in the following form:

```
  get 'graph/d3'
  match "graph/d3",
          :to => "graph#d3",
```

Because we are going to use JSON for our D3 demo, we need to specify the default format of the page.

We must first create the graph controller inside *./yourBlacklightAPp/app/controllers*. Create a document file and rename it to **graph_controller.rb**. Open the file and create two new functions. For our demo, we are going to cover something very basic. Our code inside this file looks like the following:

```
class GraphController < ApplicationController
  def d3
  end

  def data
    respond_to do |format|
      format.json {
        render :json => [1,2,3,4,5]
      }
```

```
    end
  end
end
```

Here we have an empty "d3" function. This will run the view file with the same name. Our "data" function will get the information from the routes.rb and respond to generate JSON in the view. For more information about how to parse JSON from URL, check the topic below:

Link to JSON parsing help

Next we need to create our view files. In your *./yourBlacklightApp/app/views*, create an empty folder and name it **graph**. Inside this folder, create a document file and rename it to **d3.html.rtb**. We will add a simple svg file for our D3 to find and append a new element to it.

```
<svg id="graph"></svg>
```

Now we have a graph controller that routes.rb can run "d3" and "data" functions in to connect to the view section of our app. We need to add Javascript for D3 to run. In your *./yourBlacklightApp/app/assets/javascripts* create empty document and name it **graph.js**. In our simple demo, we get the value from the JSON file and draw a simple chart in our view. The example below is from this website [34] http://www.overfitted.com/blog/?p=302

Please be aware, this demo works with D3 version 3 and running this on version 4 requires some changes in functions. For example in version 4 of D3, scale.linear() is changed to scalelinear(). To see all the changes visit the link below:

https://github.com/d3/d3/blob/master/CHANGES.md

```
$.ajax({
            type: "GET",
            contentType: "application/json; charset=utf-8",
            url: 'data',
            dataType: 'json',
            success: function (data) {
                draw(data);
            },
            error: function (result) {
                error();
            }
        });

function draw(data) {
    var color = d3.scale.category20b();
    var width = 420,
        barHeight = 20;

    var x = d3.scale.linear()
```

```
        .range([0, width])
        .domain([0, d3.max(data)]);

    var chart = d3.select("#graph")
        .attr("width", width)
        .attr("height", barHeight * data.length);

    var bar = chart.selectAll("g")
        .data(data)
        .enter().append("g")
        .attr("transform", function (d, i) {
                return "translate(0," + i * barHeight + ")";
            });

    bar.append("rect")
        .attr("width", x)
        .attr("height", barHeight - 1)
        .style("fill", function (d) {
                return color(d)
             })

    bar.append("text")
        .attr("x", function (d) {
                return x(d) - 10;
            })
        .attr("y", barHeight / 2)
        .attr("dy", ".35em")
        .style("fill", "white")
        .text(function (d) {
                return d;
            });
}

function error() {
    console.log("error")
}
end
```

After saving the file, navigate to *www.localhost:3000/graph/d3* and you can see the simple visualization (Figure 39).

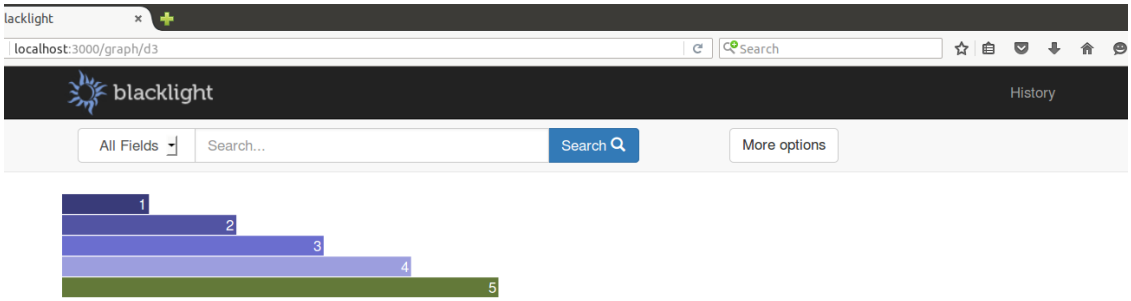


Figure 39: Simple D3 result in Blacklight

9 Acknowledgements

This work is part of ongoing work on the IDEAL and GETAR projects. This work is supported by NSF grants IIS-1619371, IIS-1619028, and IIS-1319578.

10 References

[1] Edward A. Fox, Kristine Hanna, Andrea L. Kavanaugh, Steven D. Sheetz, and Donald J. Shoemaker. Integrated Digital Event Archiving and Library (IDEAL): Preview of Award 1319578 - Annual Project Report. http://vtechworks.lib.vt.edu/handle/10919/52853, 2014.

[2] Edward A. Fox, Kristine Hanna, Andrea L. Kavanaugh, Steven D. Sheetz, and Donald J. Shoemaker. Events Archive Invitation, Funding. http://www.ctrnet.net/, 2016.

[3] Edward A. Fox. GETAR Data Flow Diagram. https://canvas.vt.edu/files/1566514/download?download_frd=1, 2016.

[4] Hue Team. Hue - Hadoop User Experience - The Apache Hadoop UI. http://gethue.com/, 2016.

[5] Tim Hall. Apache Solr. http://hortonworks.com/apache/solr/, 2016.

[6] Jonathan Rochkind. Project Blacklight. http://projectblacklight.org/, 2016.

[7] Mike Bostock. D3.js - Data-Driven Documents. https://d3js.org/, 2016.

[8] CS 5604 Information Storage and Retrieval Front-End / User Interface Team. https://vtechworks.lib.vt.edu/bitstream/handle/10919/70935/Front-End%20User%20Interface%20Final%20Technical%20Report.pdf, 2016.

[9] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *An Introduction to Information Retrieval*, chapter 9, pages 177–194. Cambridge University Press, 2009. Online.

[10] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *An Introduction to Information Retrieval*, chapter 11, pages 219–236. Cambridge University Press, 2009. Online.

[11] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *An Introduction to Information Retrieval*, chapter 12, pages 237–252. Cambridge University Press, 2009. Online.

[12] Daniel Tunkelang. *Faceted Search (Synthesis Lectures on Information Concepts, Retrieval, and Services)*. Morgan and Claypool Publishers, 2009. "http://disi.unitn.it/~bernardi/Courses/DL/faceted_search.pdf".

[13] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann, San Diego, 1994.

[14] Andrew Hunt, David Thomas, and Ward Cunningham. *The Pragmatic Programmer: From Journeyman to Master*. Addison Wesley Professional, Boston, 1999.

[15] Tamara Munzner. *Visualization Analysis and Design (AK Peters Visualization Series)*. A K Peters/CRC Press, 2014.

[16] Jack Reed. GeoBlacklight. http://geoblacklight.org/, 2016.

[17] Plataformatec. Devise: Flexible authentication solution for Rails with Warden. https://github.com/plataformatec/devise#getting-started, 2016.

[18] Cloudera Inc. QuickStart Downloads for CDH 5.8. http://www.cloudera.com/quickstartvm, 2016.

[19] HUE Interface. http://www.cloudera.com/documentation/archive/cdh/4-x/4-3-1/Hue-2-User-Guide/hue2.html, 2016.

[20] Cloudera Inc. Introducing Cloudera Search. http://www.cloudera.com/documentation/archive/search/1-3-0/Cloudera-Search-User-Guide/csug_introducing.html, 2016.

[21] Cloudera Inc. MapReduce Tutorial. https://archive.cloudera.com/cdh/3/hadoop/mapred_tutorial.html, 2016.

[22] Seungwon Yang, Haeyong Chung, Xiao Lin, Sunshin Lee, Liangzhe Chen, Andrew Wood, Andrea L Kavanaugh, Steven D Sheetz, Donald J Shoemaker, and Edward A Fox. PhaseVis: What, When, Where, and Who in Visualizing the Four Phases of Emergency Management through the Lens of Social Media. In *10th International Conference on Information Systems for Crisis Response and Management, Baden-Baden, Germany*, 2013.

[23] Lauren McCarthy. p5.js. https://p5js.org/, 2016.

[24] Project Blacklight. https://github.com/projectblacklight, 2016.

[25] Vladimir Agafonkin. Leaflet. http://leafletjs.com/, 2016.

[26] David Heinemeier Hansson. Ruby on Rails. http://rubyonrails.org/, 2016.

[27] Rails Architecture. https://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/, 2016.

[28] What is a MARC Record and Why is it important? https://www.loc.gov/marc/umb/um01to06.html, 2016.

[29] Jim Weirich. Rake - ruby make. http://rake.rubyforge.org/, 2003.

[30] Layouts and Rendering in Rails. http://guides.rubyonrails.org/layouts_and_rendering.html, 2016.

[31] Bootstrap. http://getbootstrap.com/, 2016.

[32] Official Sass port of Bootstrap 2 and 3. https://github.com/twbs/bootstrap-sass, 2016.

[33] Getting Started with Rails. http://guides.rubyonrails.org/v2.3/getting_started.html#drying-up-the-code, 2016. Section 7: DRYing up the Code.

[34] Jeff Quinn. Using d3 in rails. http://www.overfitted.com/blog/?p=302, 2016.