The Characteristics of a Professional Website-Database Interface: Remaking an Online Index of Scientific American Illustrations

An Interactive Qualifying Project Report
Submitted to the Faculty of the

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the
Degree of Bachelor of Science

Brian Flanagan _____

Jacob Ganoe _____

March 1, 2019

Advisor Signature

_____

Co-Advisor Signature

_____

Abstract

This project focused on improving the website housing the *Scientific American* database, addressing lingering problems from older versions of the site. The project group re-coded the website from scratch, fixing lingering issues from the older website, including missing data and security concerns. The group then analyzed nine other database websites, both small and large, to determine what makes an effective database and applied that knowledge to improving the SciAm database.

Table of Contents

List of Figures

Authorship

Abstract ---------- Jacob Ganoe

Table of Contents ---------- IQP Team

Authorship ---------- IQP Team

Introduction ---------- Jacob Ganoe

Technical Difficulties ---------- IQP Team

SCIAM Database ---------- Brian Flanagan

Database Usability Analysis ---------- Jacob Ganoe

Conclusions ---------- Jacob Ganoe

Recommendations for Future Teams ---------- IQP Team

Introduction

From 1845 to 1870, *Scientific American* primarily covered inventions newly patented in the United States. The magazine offers today's scholars a treasure trove of information about these topics both through articles and a large number of illustrations. Though the text portions of the magazine are easily searchable via online copies of the publication, there was previously no way to quickly and accurately search the illustrations. First started in 2007 and sponsored by the American Antiquarian Society ever since, the *Scientific American* Illustrations IQPs have created a publicly available website and database indexing these images.

The earliest IQP groups focused on the initial creation of the database and website as well as cataloging entries into the database.[1] Originally, the website was built on RDF framework and accessed via a website interface created by the first IQP group. This was modified and updated by the following groups to a basis in MySQL, allowing the website and database to be hosted and run on WPI servers. While work on the website-database interface continued, adding aesthetic changes to the website, data security features, and advanced search features, work continued cataloguing new volumes of *Scientific American* into the database. When the periodical was completely catalogued from 1845 to 1869, work shifted to maintenance and finalization of the website-database interface. The next few groups identified numerous areas of improvement within the old interface, including missing data, possibilities for security breaches, and errors using the browse and search features. Also identified as an issue was the transfer of the reference database from Cornell University to the Hathi Trust. This removed the ability of the SciAm database to link directly to the issue containing the specific illustrations.

[1] "Browse Making of America." Making of America. Accessed February, 2019. http://ebooks.library.cornell.edu/s/scia/.

The primary focus of the group this year was to make the website-database interface as robust and as usable as possible. The term “website-database interface” represents the website a scholar uses to access a database, the database of information being accessed, and the interface that the website uses to retrieve information from the database. In this paper, this term will be simplified to either “website(s),” “database(s),” or “interface(s).” Our IQP group completely redesigned the interface using Django framework based in the Python coding language to create a new user interface (UI) from scratch and directly imported the previously catalogued database information into the new interface. The group then analyzed nine additional interfaces and identified key features about these websites that make them user-friendly, as well as obtained oral feedback from research fellows and staff from the American Antiquarian Society about useful features. Some of the features suggested or observed in the database websites were then added to the new interface and the rest noted as recommendations for future teams to consider.

Technical Difficulties

When this project was started, we first looked at the reports of previous IQP groups to see what issues still remained regarding the interface and what suggestions they gave to continue the project. Prominent among the recommendations were fixing issues within the browse feature as well as a data security issue present regarding the administrative passwords for the database. In order to determine if we could address these concerns, we accessed the code for the interface to assess if and how these recommendations could be fixed.

Organization of lines of code is paramount to the ability of a new programmer to understand what the code is executing and how it functions. This was the primary downfall of the code for the previous database. The code for the entire interface, including both the website and the database, as well as all the UI for all different pages were all contained within a single text file with approximately 2000 consecutive lines of code. Because the entire interface was

contained within a solid block of text that also included minimal comments to identify the functions of the code, it would have been extremely difficult to locate any errors within the code and rectify them. Additionally, the code was also scrambled into different sections at random, meaning that one line of code did not necessarily go with the previous line of code and vice versa, which would have potentially lead to even further complications.

We encountered further technical difficulties when we attempted to access the code and perform preliminary edits. All files for the MySQL interface were stored in a centralized WPI group drive which acts in the same manner as a file folder on a personal computer, except that is is accessed remotely. This method of storing files for use causes numerous issues, such as the drive not being an actual server that runs a website. Instead, the drive operates like a remotely accessed folder on another server that opens a file that contains the website-database code, which is then compiled and run by the user's internet browser. This causes issues because an internet browser will not always be able to correctly compile the code of the website and it also has restrictions of which types of code can be compiled, further complicating usage of the database. Additionally, when work was completed on a file within the drive, it operated like a normal file folder and overwrote the existing file, preventing old iterations of the website from being saved. This is a serious issue when coding a website, as iterations of code must be saved in order to maintain a streamlined development process and allow reversion to an older build of the website should a new feature break.

Additionally, the website URL used to access the website was not professional looking and extremely difficult to remember, as it was effectively a file path, not a website link. For example, the main file folder access URL would be users.wpi.edu/~sciam, but this link would only gain you access to editable files for code. In order to access the database itself, the URL

entered would be users.wpi.edu/~sciam/sciam.cgi?pagetype=splash. Not only is this difficult to remember, but this URL type of authentication is extremely insecure and was also found to be the cause of the data security problems at a later time in the project, as described below. Our group also had permission issues accessing the shared drive with editing permissions, resulting in an inability to use the shared drive for any development of the interface.

The website design UI and functionality was also extremely lacking and seemed unprofessional, as the website's design was outdated and the database does not seem to have been maintained. The functionality of the interface was lacking in many ways, including a poorly organized table that presented results, a method of admin authentication that printed the admin login credentials directly in the accessing URL where they could be easily copied, and a table within the code itself that stored admin login credentials.

All of these issues affect the maintainability of the website, meaning how easy it would be to make changes and update the code to the website. Because it was so convoluted, even an experienced programmer had difficulty understanding the original layout of the code and what everything did. To make the code maintainable, it would have to be separated into multiple different sections or files, with comments added to explain the function of the code. Due to the sheer scale of editing that would have to occur in order to make the existing code readable and maintainable while not fixing any issues, we determined that it would be much simpler to remake the database correctly from scratch. This would allow the new website to be created in an easily maintainable manner and solve the lingering issues noted by previous IQP groups.

SCIAM Database

At the start of the project, we set out to rectify as many of the interface problems noted by previous IQP groups as feasible during the project. Where prior teams working on the IQP had minimal experience with the necessary coding skills required to fully create and edit the website, this team was comprised of members who are experienced with the necessary coding languages required to create a database.

The first technical challenge that we undertook during the project was to access the code of the previous database, which was found to be contained in a single, large document. At first, the group thought that the easiest way to solve the issues presented by previous groups was to edit the existing code and bring it up to a standard where modifications could be easily completed on it. While the original code was being edited, a Django framework website was begun in order to help us understand how python would interact with the current database and if a complete website could be created using solely python.

After successfully creating a fresh website from scratch using Django and noting the difficulty of editing the existing code, we decided that it would be more efficient to simply create a new interface from scratch, import the current data from the old database, and solve all of the issues caused by the old database in a single edit. The *Scientific American* data housed in the previous MySQL database was then copied and directly imported into a new SQLite database table using a custom python management script integrated into the Django project as a custom command, allowing it to be more easily modified. The group then proceeded to create readme documents to help preceding groups obtain the necessary information required to work on the website. One document describes how to download the necessary software and programs and the other explains how to properly open, edit, and use the new website-database interface.

The Django Website

The Django Web-Framework is an open-sourced python library used to create web-servers. This framework allows for rapid, high-level development. From a high level, a Django application is a set of templates (HTML), these templates are rendered by 'views' which represent the logical part of the application, and these views are connected via URL paths specified in the urls.py file. Descriptions of the most important python files in the project may be found in Appendix I.

Building the database

The SCIAM database was created using a mixture of experience, online documentation, and online tutorials. From past software engineering experience, we decided it would be most efficient to add features incrementally after creating a code base from which we could easily add new features. The initial code base consisted of simple views that rendered HTML templates which were accessible online. Once we were able to make an HTML template available online through use of the Django web server we created, we started adding some basic template rendering features, including the home page, about page, help page, and history page.

Porting Basic Pages

To port a basic webpage to our Django site three major steps needed to be taken:

1. An HTML template needed to be created
2. A view function needed to be created to render the html template
3. A URL needs to be created that invokes the view function

These three steps will be the basis for all other features to be added to the site during this project and in the future. An example of one of these basic pages is shown below in Figure 1.

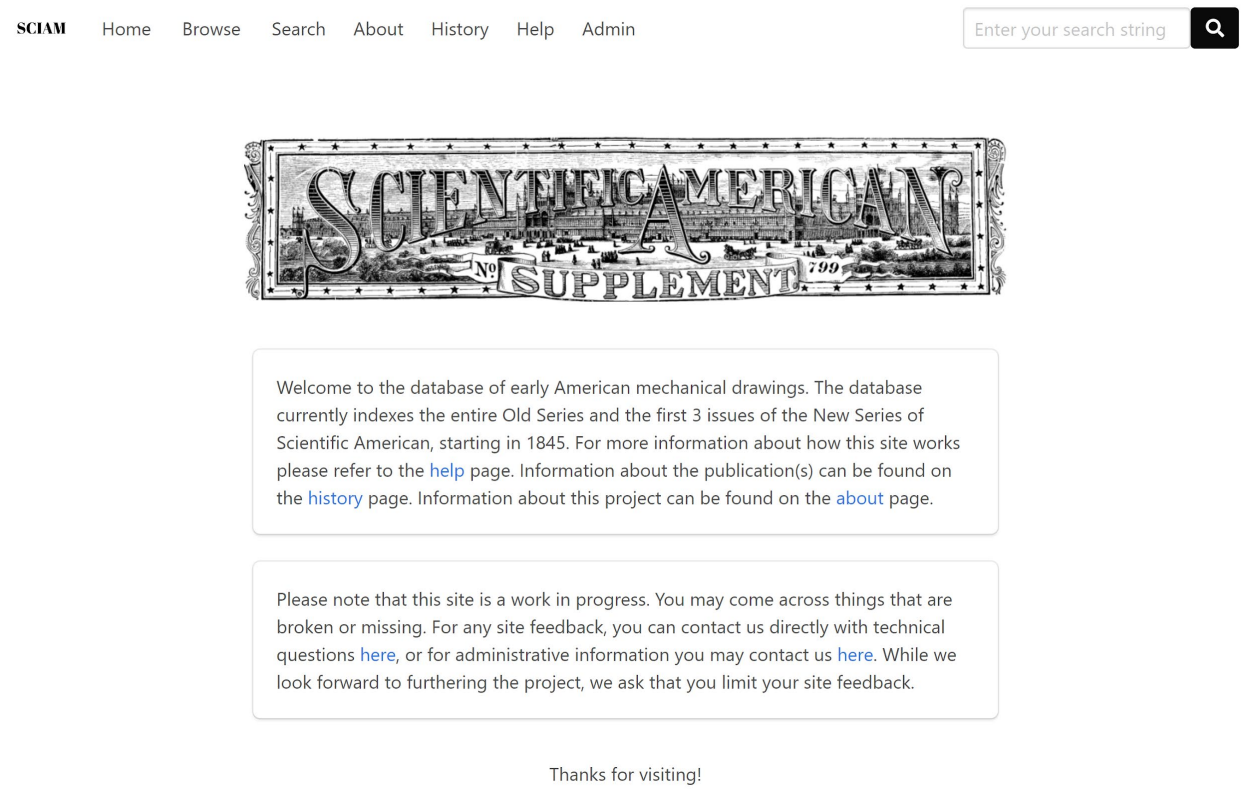


Fig. 1. SciAm database homepage. This was the first page to be ported to the new interface.

The first step, creating an HTML template, is fairly simple. HTML is a simple markup language, that basically allows the programmer to format text by enclosing their text with HTML "tags." These tags are processed by the user's browser when an HTML page is loaded and the text within the tags is formatted correctly based on its respective tag. Because creating an HTML template has more to do with styling, initially we just used the HTML code from the previous website for these basic pages. These pages are considered basic because they do not involve any complex logic in their respective view functions and they do not interact with the database at all.

Once we copied over the previous site's HTML code for the four basic pages, we moved on to the second step: creation of view functions for each basic page. A view function represents the logic behind a webpage. Each view function will always contain a rendering statement which will return the corresponding HTML template so the user may view it in their browser. The

rendering statement will simply be an appropriate call to a built-in Django function (i.e. Django.shortcuts render function). For each of the basic pages outlined in previous paragraphs, a new view function was created only using the Django rendering function. Now that the views were made we needed to make all of our work accessible via the public domain sciam.wpi.edu.

Django provides a simple way to allow a template, rendered through a view function, to be accessed via the public domain of one's site: URL paths. The third and final step of this process was to create new URL paths for each of the basic pages. These URL paths represent whatever follows the domain name when the site is accessed. For example, when a user accesses sciam.wpi.edu/about, the domain is sciam.wpi.edu and the URL path is /about. When this URL path is chained on the end of the domain Django processes the path by looking into a specific file: urls.py. More descriptions of how exactly these files work may be found in Appendix I. Through the use of the URL path functionality, we were able to provide public access to the home, about, help, and history pages.

The Browse Feature

After the basic pages were successfully ported and we had created a functional code base, work shifted to adding site features. There were two main features we wanted to initially include in this site: browse and search. Both of these features needed to access the database of illustration entries and render the results on a specific webpage of the site. The browse feature was created first because it involved less filtering than the search page would later entail.

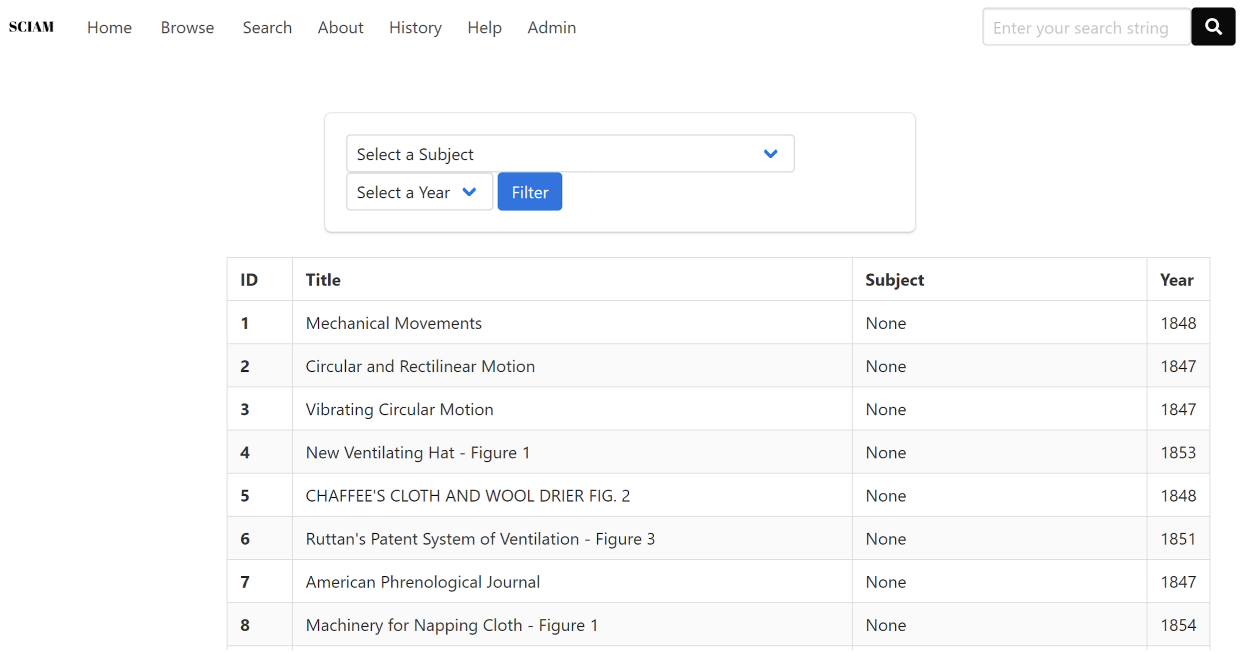
To create the browse feature, we initially followed the same steps to create the basic pages. After we were able to render a webpage using the URL path "/browse," we moved on to adding the database entries to the page.

Django provides specific ways to access the site's database and render entries on a page. After researching and reviewing how to render database entries on a webpage, the group was able to display all of the entries on the browse page, which was achieved by adding logic to the browse view function. The logic to access the database saved all of the database entries in a variable, which was then passed into the render function call as a argument. Adding this variable to the render function call gave the browse HTML template access to the variable containing all of the database entries.

Utilizing this variable containing all of the database entries relied on the exploitation of Django's templating language. This templating language involves two major parts: variables and tags. A variable in Django's templating language is exactly the same as it is in any other programming language: a string of characters that represents some data to be manipulated or displayed, and tags represent logic in Django templates. Through use of both tags and variables, the data given to the browse template from the browse view function was displayed. Each field of a database entry was displayed separately in a visual table on the browse page, but only certain fields were selected to save horizontal space on the browse page.

Following the successful display of the database entries, a simple filtering option was added to the browse page, which laid the groundwork for later adding the search feature. This filter allowed the user to filter the database entries by subject and/or by year, as shown in Figure 2 below. The user would select a specific subject and/or year on the browse page, which would then be sent back to the browse view function for processing. Again, all of the logical processing required by a web page is done within the page's view function. The filter parameters selected by the user, subject and/or year, are used in the browse view function to filter the database entries and create a new variable containing the filtered list of database entries. Once the new list of

entries is ready, the view sends it back to the browse HTML template to be displayed in the exact same manner that the original list was sent to the browse HTML template, through the render function call as a function argument. Once the HTML template receives the list of entries, the page refreshes and the new list is displayed in place of the original list of all the database entries.



ID	Title	Subject	Year
1	Mechanical Movements	None	1848
2	Circular and Rectilinear Motion	None	1847
3	Vibrating Circular Motion	None	1847
4	New Ventilating Hat - Figure 1	None	1853
5	CHAFFEE'S CLOTH AND WOOL DRIER FIG. 2	None	1848
6	Ruttan's Patent System of Ventilation - Figure 3	None	1851
7	American Phrenological Journal	None	1847
8	Machinery for Napping Cloth - Figure 1	None	1854

Fig. 2. SciAm database browse page. This page lets users browse all entries in the database at once.

More technically, the user's selected filters are sent back through the browse view function through an HTTP POST request to the "/browse" URL path. This filters are then processed and a new list of filtered database entries is generated. This list of entries is sent to the browse HTML template, which, in turn, is processed into an HTTP response. This HTTP response contains the new HTML to be displayed in the user's browser, so the page refreshes and the new HTML is displayed. More technical details of how exactly Django creates its HTTP responses can be found in the documentation, freely available online.

Search Iteration I

The first iteration of the search feature built on the filtering option of the browse feature, after all of the initial page creation was completed using our code base as a guide. The new search feature allowed the user to infinitely add search constraints based on model fields (see Appendix III) and user input search strings for each model field selected. These search constraints were processed through the search view function in the same manner that the browse filters were processed, using Django functions to access and filter database entries based on certain given filter parameters. However, there was an added layer of complexity because a user could add infinite search constraints to their search.

To account for any number of search constraints being added to a user's search, we implemented a filter chaining method. This method allowed us to chain search constraints (or filters, both are used interchangeably in this paper) onto the results of previous filtered database queries. For example, if we are given two search constraints from a user, first we will filter the database entries based on constraint 1, which gives use results *Y*. Then we will filter *Y* based on constraint 2, which will give us results *Z*. Now *Z* contains results that have been filtered by both constraint 1 and constraint 2. This method was generalized for *n* number of constraints which could be selected by the user.

Another difficulty in fully implementing this feature centered around the issue of actually passing the search constraints back to the view function from the HTML search form (using the <form></form> HTML tag). Using previous knowledge of array manipulation in the C programming language, we implemented a similar method where we are given a group of constraints and the number of constraints there are, however, this group of constraints is not in any iterable data structure (i.e. array, dictionary, set, list, etc.). Instead we used the number of

constraints as the limit for an incrementing value *i*, which represented the current constraint being processed. Using this variable *i* allowed us to access each constraint by concatenating the value of *i* to the end of a common string representing the key for each constraint value. Using the concatenated *i* and common string as the key, gave us access to each constraint separately, for example "search_param_3" would give us access to the fourth search constraint value. The code is slightly more involved, but this is the general idea behind our approach to solving the current issue. Despite all of our efforts to create this complex search feature, which was fully functioning, the feature broke at some point after the styling of the website was redone.

The Base Template

Before a new styling scheme was implemented, we decided to generalize the header and the footer of each webpage. This generalization basically removed the header and the footer from each individual HTML template, and housed the header and footer in its own separate base_template file. This file was extended by all of the HTML templates so what is actually displayed to the user does not change, while the header and footer of the page are stored in a separate file from the HTML template. The header and footer included the navigation menu and footer information used by every page on the site. Generalizing the shared code consolidated the navigation menu and footer information into one file, extended by all of the other templates.

Generalizing shared code into one file or module that can then be used by many other files is a good coding practice and should always be implemented when possible. It allows for easy upkeep of the common code. Instead of updating each file that uses the code individually, we must only update the shared file. This also allows for easy debugging if an issue arises involving the shared code. Just as shared code should be generalized, all features of the project

should also be split into their own files to provide isolation when debugging and developing new features.

Styling with Bulma CSS

To recreate a new styling scheme for the site, the Bulma.io CSS framework was used. CSS stands for Cascading Style Sheet and basically represents the styling and actual format of our HTML templates. Using a CSS framework allowed us to use a professionally created library of styles instead of starting from scratch, which exponentially reduced the amount of time it took to create a professional-looking website. We decided to use this particular CSS framework because it was regarded as new and widely used, while also being easy to understand and implement into a project.

By reading the online documentation for the Bulma CSS framework, we were able to re-style the entire site using some simple techniques. Firstly, to add styling to an HTML template with Bulma CSS, we only need to add specific classes to the HTML tags. Classes are considered CSS selectors, which basically means that they define certain styling characteristics for whatever tag the class is applied to. A tag represents the basis of all HTML formatting, for example to write a paragraph we would use the " " tag and write our paragraph text between the opening (" ") tag and closing (" ") tags. All of the classes we used were extremely intuitive and well defined in the Bulma CSS documentation.

Search Iteration II

Following the implementation of the Bulma CSS framework, the first iteration of the search feature stopped functioning correctly. A second iteration of the search feature was necessary at this point in the project because the root of the problem with the first iteration could not be isolated and resolved. The second iteration of the search feature is now considered the

basic or general search feature of the site. The original implementation of search had many different parts that could have been contributing to the issue, so for this iteration the search functionality was simplified. The user inputs a search string, as seen the example using the search string "Mechanical" in Figure 3, which is processed by the search view function. During this processing of the search string, the database entries are filtered by each of the model fields as search constraints, while using the same search string.

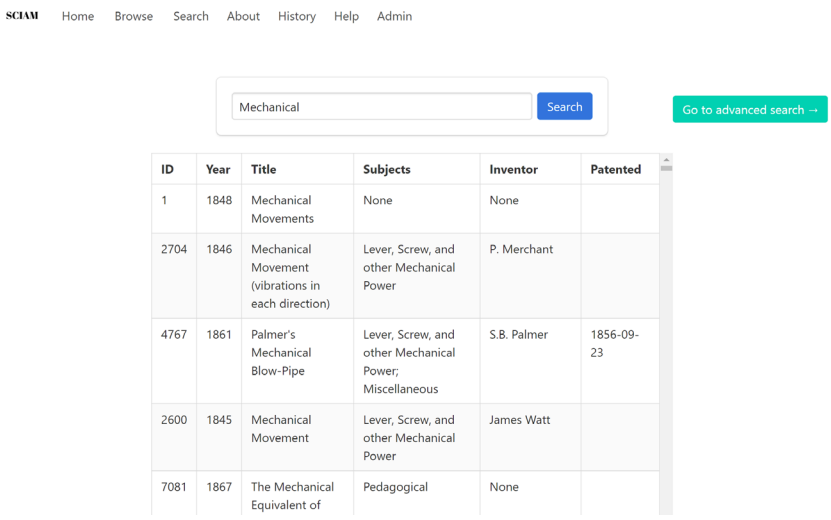


Fig. 3. SciAm database basic search page. This allows users to filter results using a simple string input.

To filter a list of database entries we select a model field and a string that the model field of a database entry should contain. The list is then filtered based on these specifications. For this particular implementation a new method of combining the results of the filtering processes needed to be used. Filtering the database entries using this technique allowed for the entire database to be searched for a single string, which both simplifies and generalizes the search.

One major difficulty with this implementation was dealing with duplicated results. When searching the entire database for a single string there may be duplicated results for the separate filters applied to the database entries. These duplicated results were removed during the concatenation of the separate lists of results, from each filter being applied to the database

entries, using built-in Python set functionality. The combined results are then returned in the form of a context for the search HTML template to process and display to the user, in the same way described for the Browse Feature and Search Iteration I.

Search Iteration III (Advanced Search)

A third and more advanced iteration of the search feature was also implemented to provide the user with more control of what model fields they were searching within. Currently, the user is able to select the specific model field that they would like to search the database for. This is similar to Search Iteration I, however the user cannot add infinitely many search constraints, instead they are limited to just one. It can also be similarly compared to the Search Iteration II, but instead of each model field being searched, only one, specific model field is searched. The model field or "Search Field" is selected by the user, as seen in Figure 4. This feature is not yet complete and will need more filtering functionality added in the future.

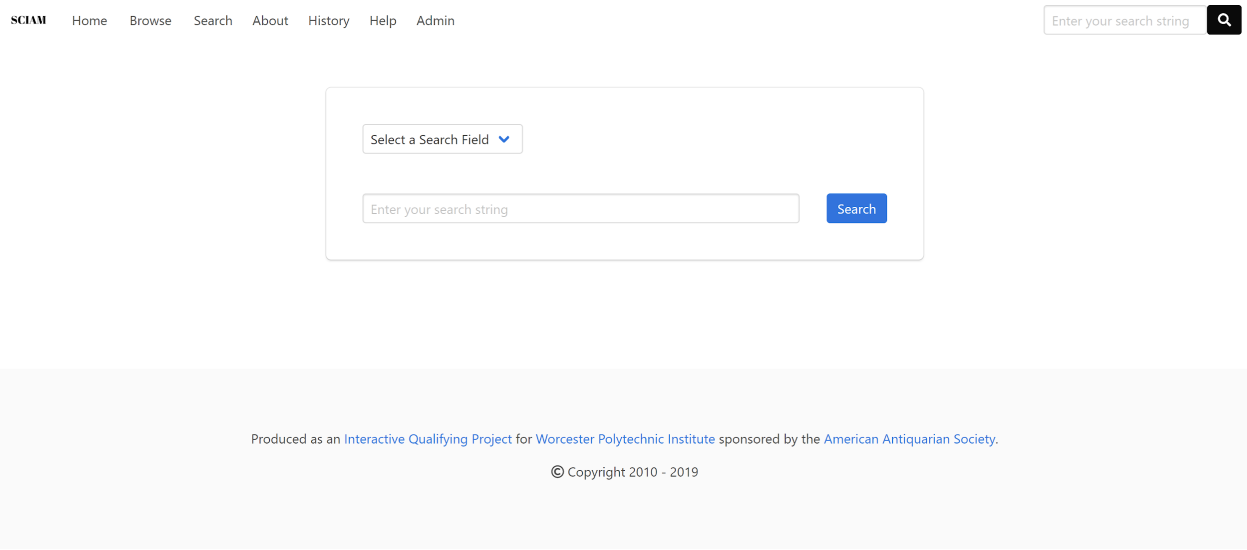


Fig. 4. SciAm database advanced search page. This page is still undergoing refinement.

The Under-construction Placeholder

During the construction of the new website, certain pages were not converted over to the new styling library as they were not priority pages vital to the use of the database. As these pages would present errors if they were navigated to via the website, a simple html template was added to inform users that the incomplete pages are under construction, as seen in Figure 5. This allowed us to focus our development on the vital pages of the website will allowing it to remain presentable to any users who visited the site.

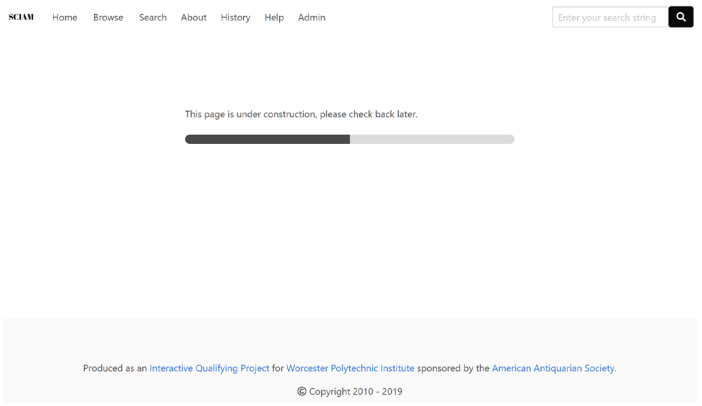


Fig. 5. Under Construction placeholder for unfinished pages.

Database Usability Analysis

As work on the final iteration of the new database website neared completion, it became clear that certain features and tools must be included in a properly functioning and usable database. To help maximize the usability of this database, we studied nine different database websites, a combination of larger, more general sites as well as sites focusing on more specialized topics. The five large, general databases we selected were

1. JSTOR

2. America History and Life by EBSCOhost
3. Project MUSE
4. ProQuest Historical Newspapers - The New York Times
5. Global Issues in Context by GaleGroup

The four small, specialized topic databases that we selected were

6. The Science in the 19th Century Periodical Database
7. The Index of Virginia Printing Database
8. The Geography of Slavery
9. The Trans-Atlantic Slave Trade Database

The sites were assessed for four general criteria

1. The general layout of their homepage
2. The layout and presentation of results when searching
3. Any special features or options present that are unique to the website
4. The overall usability of the website

For this research, special features are considered anything other than standard search, browse, or advanced search features traditionally present on every database. Once an overall impression was formed about each database individually, the entire group was compared as a whole to the SciAm database to identify the necessary features and design details that would best improve the database. In addition to examining websites, we also presented our work-in-progress to staff and fellows from our sponsor, the American Antiquarian Society. In our talk, we showed

them the new features from the website that was created and how to use the interface. After our demonstration, we obtained oral feedback from those present at the presentation regarding features or content present within the website that they thought should be modified or changed to make the website easier to use. This feedback was then incorporated both into our findings from the database analysis and into our suggested changes for future teams.

The first primary feature examined in each website was the general layout of its homepage, what menu options are available, and the aesthetics of its design. Of the databases researched, four can be described as a typical search engine design, with a large search bar prominently displayed on the front page of the database. One database, Global Issues in Context by GaleGroup, was laid out like a news website, with various options of world-news-related articles and recommended reading present on the majority of the website and a smaller search bar off near the top of the website. The remaining four databases are smaller, in terms of publishers or companies backing the database, and all the homepages of these databases have prominent descriptions of the database, its creators, and/or its contents displayed. Of these databases only one, Index of Virginia Printing, had a search bar on its homepage, but this is located at the bottom of the website and was overlooked by our group on the first inspection of the homepage. For available menu options, all databases studied had a link in a menu option for browsing the databases that allowed the database to be searched by directly clicking on the link. The Geography of Slavery database takes this farther, by bringing up a second menu after clicking on the browse button on the homepage. This second menu then allows searching through the database when one of the available search options is chosen.

Of these nine databases, the five larger databases' overall aesthetic look could be described as an up-to-date website, with clean lines, bright and well-blended colors, occasionally

moving or alternating recommendations for reading, and a general feeling of modernity. The four smaller databases all had similar over aesthetics as well, with all opting for a plain color scheme, unrefined boxes, unintegrated pictures, and a general feeling of an old website.

Based on these observations, our group came up with the following observations for the design layout of the SciAm database website. The website must maintain a modern feeling, as our group believes that a modern website adds to the professionality and credibility of the website, as well as promotes use of the database as it seems more intuitive. The five larger databases all have a modern design that helps to facilitate the ease of use of the website as well as making it attractive to the eye. The four topic-specific smaller databases all had an older design which, in our opinion, detracted from the usability and overall aesthetic of the database website. The color scheme should use modern, clean tones, with splashes of color to highlight important features and draw attention to the website as seen in Fig. 6. The darker or matte color schemes make the smaller databases that they are present in look run-down, presenting a look of neglect and disrepair as seen in Fig. 7, and should be avoided as such. Additionally, the presence of a generic search bar in a prominent location on the homepage adds tremendous ease of use to the website design. This was confirmed in our meeting with the AAS, and a search bar was added to the homepage in response.

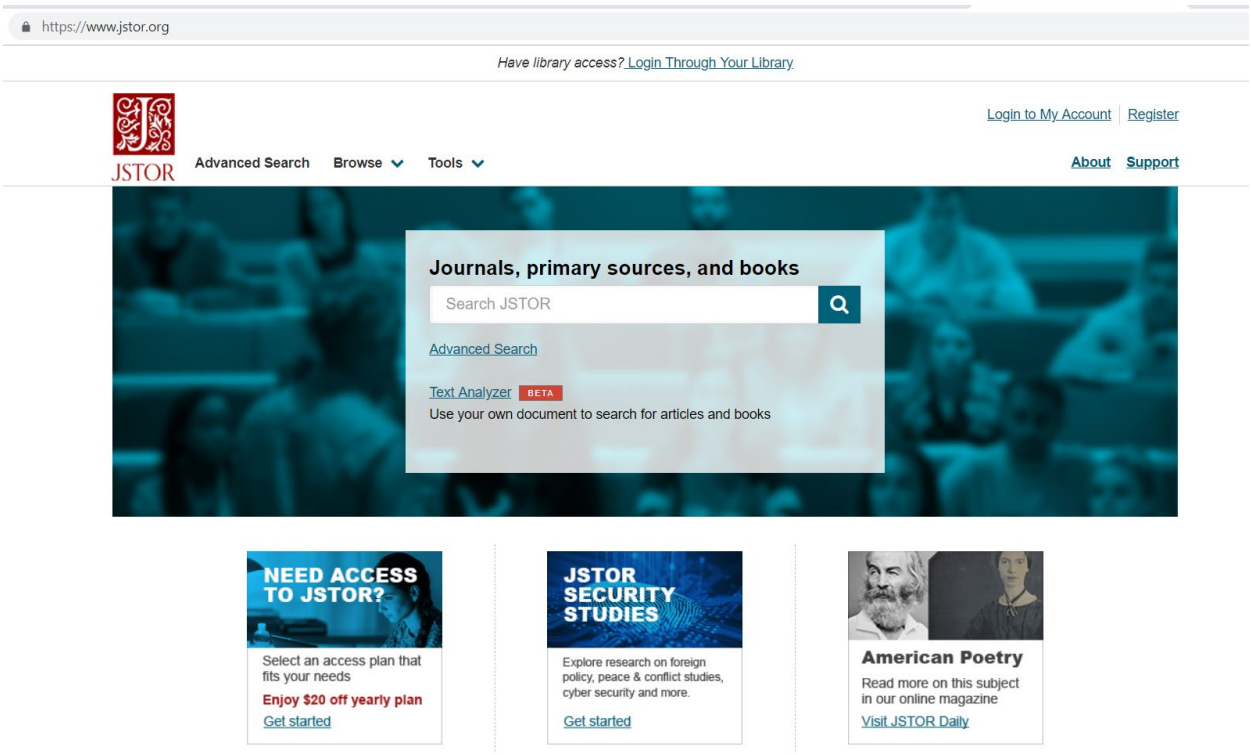


Fig. 6. JSTOR Database. Example of a good color scheme.

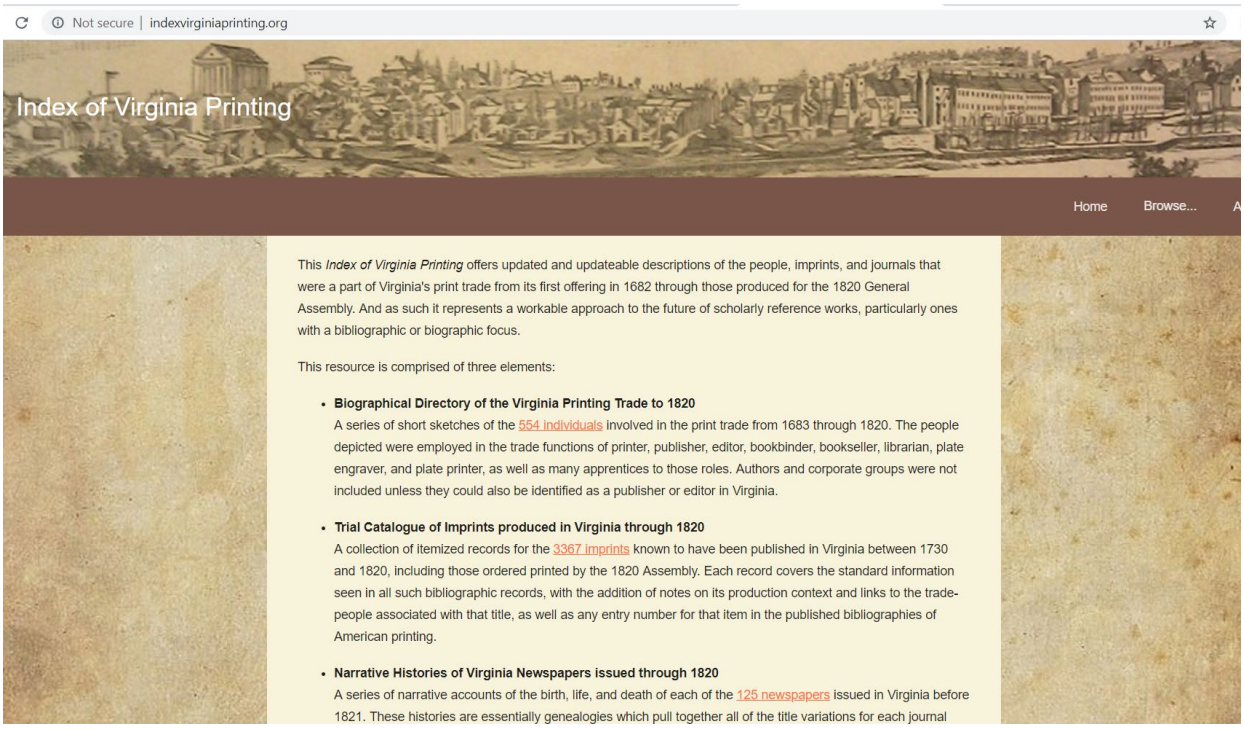


Fig. 7. Index of Virginia Printing. Example of a bad color scheme.

The second criteria that the databases were assessed for was the presentation of search results when attempting to find a particular source material. The databases presented their data in a variety of ways. Some presented the information in a tabulated list like a Google search result, in a compact view with only 2-3 lines of text per entry. Others presented the information in a tabulated list with 4-6 lines of text per entry, presenting more information about each entry to the reader but lowering the number of results visible at any one given time. One database, Global Issues in Context by GaleGroup, provided their results similar to a news website and pre-sorted the results by resource type for the researcher. Most websites offered a way to generally narrow down your searches or to add additional filters, generally to the left of the results provided, but also require you to click on an advanced search link to transfer to the advanced search feature or further refine the keywords of the original advanced search. However one database, Project MUSE, integrated the advanced search feature into its filtering options and does not require the user to navigate to a different webpage to use the database's advanced search feature, as seen in the left of Fig. 8.

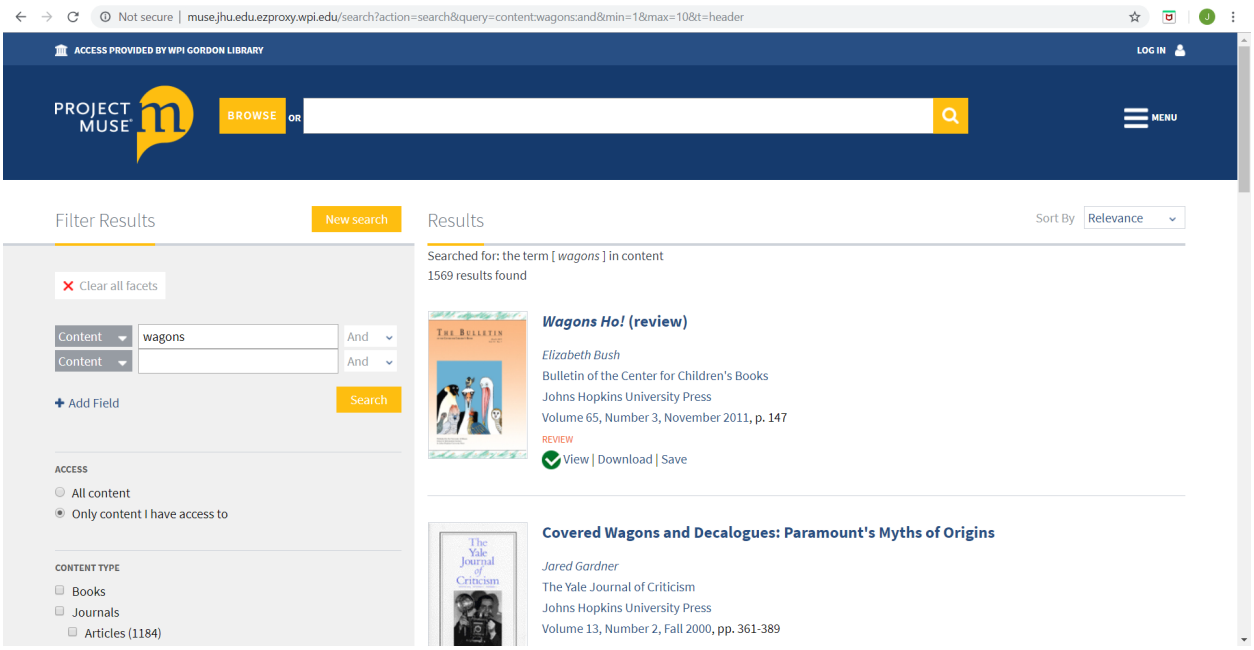


Fig. 8. Project MUSE. Advanced search functionality integration.

Of the search results themselves, all databases displayed the title of the source and its publication information on the main page after searching was complete. Most of the databases also displayed the author of the work, and a majority of them also displayed related topics and the type of material the source is, such as an article or book chapter. For databases that also offered visual aids, illustrator information was also commonly available. When a specific source was clicked on, the way that the source information was laid out also differed from database to database. Some provided the data in a list, while others provided it in a number of variations of a table.

During our presentation to our sponsors at the AAS, numerous questions were asked about how the data was presented and what researchers would find most helpful when searching through our database. The AAS group suggested that the more information that could be provided about the source, the better it would be for researchers, and that a tabulated view of said information would be better than the original block view that had been designed. Additionally, they suggested for our database specifically that a direct link to the patent image be added to the

table view so researchers do not have to search for the one. Similarly, the fellows present also suggested that while we do have a link to the image imbedded directly in the title of the detailed view, it should be advertised, so to speak, so users of the database know that the title can be clicked on as an optional link to the image. For advanced search features, the AAS group suggested that an option to restrict results to between two specific dates be added, and that the ability to add multiple search field into the advanced search be repaired and re-added to the database.

The third category that we investigated was special features that databases offered to help streamline the process of searching for sources and integrating them into a scholar's work. These special features differed from database to database, and were often only present in one database. Certain special features were, however, common to multiple databases. Features common to all the larger databases include the option of saving recent searches to a common workspace or location to permit easy access of the search history, as well as a way to download the source in some way, usually a PDF file. Most of the larger databases also could create a complete citation of the source directly from the detailed view, removing the need of manually creating one or using a citation website. Other databases had more specific features unique to the database that offered time-saving options to researchers. For example, the JSTOR database has an in-development feature that allows a researcher to upload an in-progress piece of scholarly work for analysis. Once analyzed, the JSTOR website will automatically recommend sources related to the topics being discussed. America History & Life provided by EBSCOhost offered a full-text finder courtesy of WPI which allows researchers using the database on WPI's network to locate the full-text document of the displayed abstract. Project MUSE did not have any special features other than the common ones already discussed. However, Project MUSE was the only database

to fully integrate its advanced search function into its refine search bar provided on the side of its search results as previously noted in Fig. 8. ProQuest Historical Newspapers: The New York Times allowed users to preview a detailed view and summary of the document without navigating to a separate page to view the actual document itself. Global Issues in Context by GaleGroup provides all of its sources pre-sorted by source type, offered a unique navigation method between source types, and also provides a way to translate the entire page or a single source to a different language. The Geography of Slavery showed both a modern typed transcript of its documents as well as an image of the original document, while the Trans-Atlantic Slave Trade Database offered interactive maps and graphs that users could use to find specific information on a particular year or slaving route.

After viewing all of the databases, it seems that the more professional websites offer the most helpful or interesting unique features in order to assist the researcher. For example, while the Trans-Atlantic Slave Trade Database had an outdated website design that detract from its usability, its interactive maps and graphs greatly improved our opinions of the database. As such, we believe that features present in the larger databases will offer the best help for users of the SciAm database, and that features such as downloadable sources, automatic citations, and the ability to save searches or access recent search history should be added to the website.

The fourth and final feature of the databases that was assessed was the overall ease of use of the database, which included how easy it was to navigate between webpages, use the available features, and locate the desired sources. The easiest databases to use were the ones that required the least amount of navigation to move between results and further filter them the quickest. The databases that performed well in this category, in no particular order, consisted of all of the larger databases, JSTOR, America History & Life, Project MUSE, ProQuest Historical

Newspapers: The New York Times, and Global Issues in Context. Of these five databases, the two that were the most effective were Project MUSE and Global Issues in Context by GaleGroup. Project MUSE achieved its ease of use by creating a streamlined search process for the user, offering a search box and quick access to alternate search features on the homepage and integrating the advanced search criteria into the search refinement options. This removed the necessity to ever open the advanced search tab and allowed for greater specificity and easier source access. GaleGroup's database achieved this by its easy-to-navigate results page. While lacking in certain search refinement options, there were numerous types of sources offered as results to any particular search that could be easily navigated between by a floating menu to bring the scholar to any particular section quickly. The database that performed the worst in this context was the Trans-Atlantic Slave Trade Database. This database performed extremely poorly in this category because not only was there no search feature present on the home page, the search feature itself was extremely confusing and difficult to use. While the special features included in this database of the interactive maps made up for this slightly, the overall search feature of the website made it almost unusable in the eyes of our group. As such, we believe that the best way to achieve a database that is easy to use is to streamline the entire process to the best of our ability. This includes adding a search feature onto the homepage of the database, integrating advanced search features into the filter options for basic searches, and any other change in the database that removes the need to navigate to an additional webpage for information.

Conclusion

At the start of this project, our group hoped to rectify the numerous issues with the existing interface that had been brought up by previous groups as well as continue to improve the functionality of the website UI itself. The primary focus of these goals was to solve the largest pressing issues of inaccessible data within the data, the security concerns with the old website, and the malfunctioning browse and search features. These three issues were solved in the first two terms of the IQP with the creation of a from-scratch interface using python-based Django web framework. The new website allowed the built-in administration functions of the framework to be used to securely house the admin login credentials, solving the data security issues present within the old database. The inaccessible data was also restored in the creation of the new database, as an unknown error present within the old code no longer prevented the data from being accessed in the browse and search features. Additionally, while we do have lingering issues with the new advanced search feature of the interface, the malfunctions in the browse and general search features present in the old interface have been corrected and no longer make it difficult to access relevant data from the new interface.

During the final term of the project, the group focused on the different aspects of website-database interfaces, what makes them look professional, and what particular features other than basic search features make these interfaces as user friendly as possible. After assessing nine other interfaces, the group determined that a clean and modern website design helps give the database a feeling of stability and upkeep, and a streamlined search process helps make a database outshine others with more clunky search features or older layouts.

Recommendations for Future Teams

While the creation of the new database solved many of the important issues left from the previous version of the interface, there still remains work that could be done to further improve and polish the newly created website, through either new feature addition, coding changes, or UI improvement. UI improvement is the easiest of the three tasks and will be better suited for groups with minimal coding experience. One UI fix is the completion of the history and help pages, which are currently under construction. In order to complete these two pages, their html templates must be rewritten to implement the correct CSS styling library, allowing them to be displayed on the website. Another suggestion that was brought up in our meeting with the American Antiquarian Society was the use of a logo to identify what the database is. The creation of a professional logo for the database and its integration into the website will help the database look more professional and therefore more credible. Thirdly, a new color scheme should be chosen for the website that is less bland than the current black/white motiff, but should still be clean and pop, as explained in the database analysis section. Conversion of the website to another CSS styling library in order to select a more modern color scheme or modify the website to look more up to date is also an option that can be considered, but this option would also require the entire database to be updated to the new CSS profile.

The addition of new features to the database, while still easy when compared to the coding changes, is ideal for teams with a small to moderate amount of coding experience. Most important of these changes should be the addition of filters to the advanced search page, as this will help researchers locate relevant information faster. Some filter ideas that our group believes might be helpful are a date selection option, keyword choices, and integration of the advanced search options into the results of the basic search function, as seen in the Project MUSE

database. Another feature that should be implemented is the addition of user accounts. This would allow scholars to create an account on the database and save their searches to a single location where they can easily look through all of their saved sources. This would also allow the scholar to view their search history and locate any previously viewed source that was not saved. Finally, the addition of the image URLs from the Hathi Trust database would provide the greatest improvement to the usability of the database, but it also poses the most difficult challenge of the additional features. The use of an external program or service, such as cypress.io, to help automate the process might be beneficial to expedite the addition of images to the website.

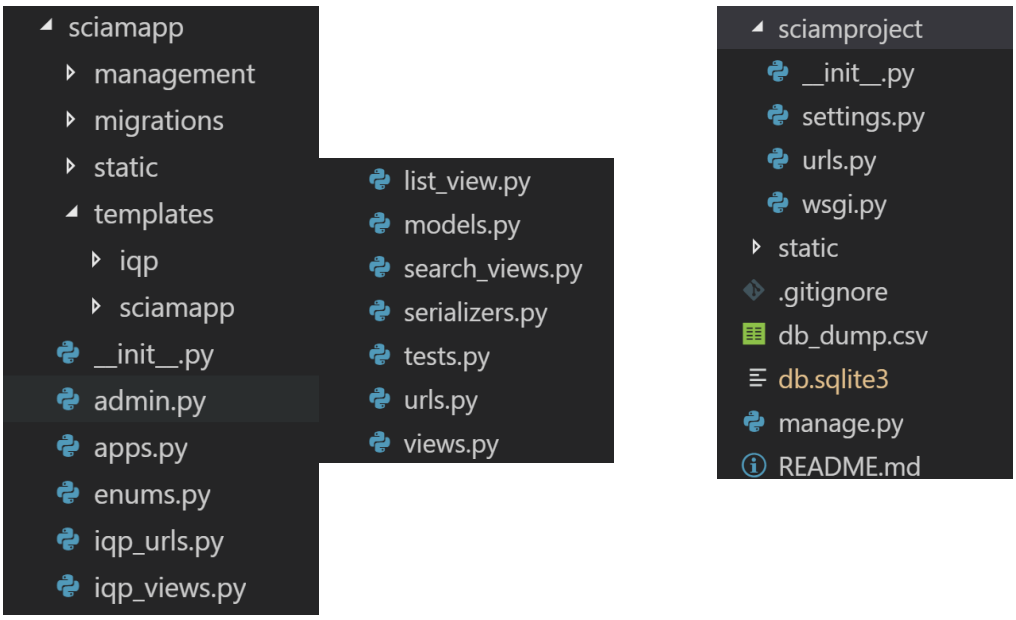
There are a few coding changes that can also be completed in order to move the website towards a more finished state, but these changes should only be undertaken by a group with experienced coders, as they may prove extremely difficult. Currently, the website operates as a development web-server, meaning that it is not a final-form website. In order to move the website from its development state to a final, production worthy product, two main changes must occur. First, an API must be developed using the Django REST Framework. This is the current standard that production websites are held to and it must be implemented into the database before the site is considered complete. Additionally, an apache server configured with WSGI must be implemented on the virtual server where our website is currently hosted. This changes the database from a terminal-based development system to a server-based production website, making our website more permanent and complete.

References

"Browse Making of America." Making of America. Accessed February, 2019. http://ebooks.library.cornell.edu/s/scia/.

Appendix I: Project organization

Django provides a built-in organizational system for any web application created using the Django web-framework. The file system is shown below and will be referenced in the following explanations.



sciamproject/settings.py

This file is mostly auto generated when a new Django project is created. Project configurations are stored in this file as constant variables, including *ALLOWED_HOSTS*, *INSTALLED_APPS*, and *ROOT_URLCONF*. The *ALLOWED_HOSTS* configuration variable contains two possible hosts, one for the production site: *sciam.wpi.edu*, and one for development on your local computer: *localhost*. The *INSTALLED_APPS* configuration variable contains a list of Django modules and the sciamapp.

sciamproject/urls.py

This file directs all site requests to the proper 'views' in the project. When a user visits sciam.wpi.edu in their browser, this file will use the URL path to generate the proper response template.

For organization purposes this urls file simply directs the program to a different urls file that is specific to the sciam application. It also includes the admin urls which simply links to the Django Admin module.

sciamproject/wsgi.py

This file contains the WSGI configuration, which will be used when the website is fully deployed on the apache web server.

sciamapp/admin.py

This file controls what models can be viewed on the Django Admin page. Whenever a new model is created in models.py, it must be registered in this file to be accessible on the admin page.

sciamapp/apps.py

This file contains the configuration of this application. It is an auto-generated file that will not need to be edited for any regular site maintenance or additions.

sciamapp/enums.py

This file contains constant variables that are used during template rendering and searching the database, including the *SUBJECTS* of the document sciam images, the possible search fields (*TOPICS* and *TOPICS_STR*), the *YEARS* of sciam images included in the database, and the different search types used when filtering through the database entries (*SEARCH_TYPES* and *SEARCH_TYPES_STR*). If any new lists must be used in the project, they should be added to this file to make the lists easily accessible in any part of the project.

sciamapp/models.py

This file contains all models for the project. A model is the format for an entry in the database. Each model represents a table that will be created in the database, each field/attribute of the model represents a column in that table, where each row is an entry in the database table. The only model in the database is the SciamImage model with fields including title, image_description, and url. All entries in the

database are automatically assigned an ID based on an incremented number of entries in the associated table.

sciamapp/urls.py

This file contains all the urls associated with the sciam application, which basically means that this file contains any URL path (besides the '/admin' path) that is associated with the database website we have created. Most of these urls are simple paths that connect to a specified view, however there are 2 that are structured differently. These two URLs are meant to represent the detail pages of the Sciam Image entries. Both of these URL paths contain the same beginning portion: '(?P<image_id_1>[0-9]+)'. This string represents a pattern within the URLs. In Django '?P' denotes a pattern, '<name_of_variable>' denotes a variable that may be used in the view associated with the URL, and '[0-9]+' is a regular expression that denotes all possible values for this pattern variable. The image ID is used as the pattern variable because retrieving entries from the database is much simpler when the entry ID is available.

sciamapp/views.py

This is one of the most important files in the entire project. This file contains all of the logic associated with the three major functions of the site: 1) Browse, 2) Search, and 3) Detail view. If a new page is ever added to the site, then there must be a view to render the page and the view must be imported to the proper urls file and connected to the proper URL.

sciamapp/templates/sciamapp/

This directory contains all of the HTML templates associated with the Sciam website. These templates are then rendered by a view. Some of these templates require a certain context to be provided so that things like search results may be displayed.

Appendix II: Welcome to the IQP Handout

Hello and welcome to the SciAm IQP with Professor Bullock and Professor Samson!

This guide, created by the 12th IQP Group of Brian Flanagan and Jacob Ganoe, will get you started on improving the SciAm database. It may seem slightly overwhelming at first, but these instructions will help you get started on the project as fast as possible.

First, you will need a computer with the following programs installed…

(NOTE: the following programs are specifically for a Windows machine, if you are using a Mac or Linux based machine please search for the Mac/Linux equivalent of the following programs)

- PuTTY: used to remotely access the sciam.wpi.edu server
- WinSCP: used to remotely access the sciam.wpi.edu codebase
- Visual Studio Code (or other text editor): used to easily edit the codebase - must link to WinSCP
- Git (Bash or desktop client): used as source control for remote development

Now to get each of these things to work…
For PuTTY and WinSCP, please follow the readme in Appendix V.
For VS Code, please download the latest version from: https://code.visualstudio.com/download
For Git, please download git for windows from: https://git-scm.com/downloads, then follow the steps below…

1. Create a github account at github.com.
2. Request access to the gitub repository by giving your github usernames to Professor Bullock who will forward them to the owner of the repository, thereby granting you access.
3. After access is granted, clone the repository to your personal computer using the https://github.com/briflan26/sciam link using whatever method is easiest.
    a. For example: using git bash (which comes with git when you download it) you may simply open git bash and execute the "git clone https://github.com/briflan26/sciam.git" command. You will also have to enter your login credentials.

Now that you have a local version of the code you need to download Django so you can build the project on your machine. Please follow the steps outlined for downloading Django in Appendix IV.

Once you have Django installed you are ready to begin editing the codebase. To do so, simply open the folder (where you cloned the repository) in VS code or your text editor of choice and start editing!

Good luck and happy coding!

Appendix III: Models and Model Fields

A model is considered a table in the database, which can be pictured as a data table made up of rows and columns. The rows of the database table are entries in the database and the columns are considered fields/attributes to the specific model. There is only one model currently used in the database which represents the illustration entries information. The attribute specifications are listed below.

The SciamImage Model Fields/Attributes:

- id
- title
- Image_description
- Artist
- Article_summary
- Inventor
- Author
- Patented
- Subject
- Subject2
- Subject3
- Keywords
- Publication
- Date
- Volume
- Issue
- Page
- Series
- People_involved
- Dateint
- url

Appendix IV: Installing Django and Creating a Project

Steps to install django on your computer

1. Ensure that you have python and pip correctly installed on your computer. Download can be found at https://www.python.org/downloads/
    a. Open windows command prompt (wnds+R, search for CMD)
    b. Enter ‘python’
    c. If python 3.x.y is installed, then continue to django import
        i. If the python shell is entered (‘>>>’ appears), hit ctrl+z+enter at the same time to exit the shell
2. In the command prompt, enter ‘pip --version’
    a. This should output the current version of pip installed and the directory at which it is located in.

```
C:\Users\bflan\Desktop\sciam>pip --version
pip 10.0.1 from c:\users\bflan\appdata\local\programs\python\python37-32\lib\site-packages\pip (python 3.7)
```

3. In the command prompt, enter ‘pip install django’

```
C:\Users\bflan>pip install django
Collecting django
  Downloading https://files.pythonhosted.org/packages/32/ab/22530cc1b2114e6067eece94a333d6c749fa1c56a009f0721e51c181ea53/Django-2.1.2-py3-none-a
ny.whl (7.3MB)
    100% |████████████████████████████████| 7.3MB 50kB/s
Collecting pytz (from django)
  Downloading https://files.pythonhosted.org/packages/30/4e/27c34b62430286c6d59177a0842ed90dc789ce5d1ed740887653b898779a/pytz-2018.5-py2.py3-non
e-any.whl (510kB)
    100% |████████████████████████████████| 512kB 56kB/s
Installing collected packages: pytz, django
  The script django-admin.exe is installed in 'c:\users\bflan\appdata\local\programs\python\python37-32\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed django-2.1.2 pytz-2018.5
You are using pip version 10.0.1, however version 18.0 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

4. In the command prompt, enter ‘python’
    a. The python shell indicated by ‘>>>’ should have appeared
5. In the shell, enter and run ‘import django’
6. In the shell, enter and run ‘print(django.get_version())’
7. Ensure that the output is at least version 2.1.2 or the latest version

```
C:\Users\bflan>python
Python 3.7.1rc1 (v3.7.1rc1:2064bcf6ce, Sep 26 2018, 14:21:39) [MSC v.1914 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> print(django.get_version())
2.1.2
>>> ^Z
```

Congrats! You have successfully installed django!

Steps to create a new django project

1. Open the command prompt

2. Enter and run 'cd [pathname]' This will change your current directory to your working directory.
    a. The default directory should be similar to 'C:/Users/[your username]/'
    b. Entering and running 'cd Desktop' will change your current working directory to the file path 'C:/Users/[your username]/Desktop/'
    c. Choose whatever directory you would like to create your project in and use the 'cd' command to enter that directory via the command prompt
3. In your new selected directory, enter and run 'python -m django --version'
    a. This double checks that you have correctly installed django in the correct PATH
    b. The output should be identical to that received in step 6 of installing django, located above.
4. Enter and run 'django-admin startproject [ENTER PROJECT NAME HERE]'
    a. This will create a new project within your current directory. To continue editing the django project, you must again change your directory to the newly created project directory

```
C:\Users\bflan>cd Desktop

C:\Users\bflan\Desktop>python -m django --version
2.1.2

C:\Users\bflan\Desktop>django-admin startproject sciam
```

5. Enter and run 'cd [ENTER PROJECT NAME HERE]'

```
C:\Users\bflan\Desktop>cd sciam
```

6. Enter and run 'dir'
    a. The 'dir' command allows the user to view all of the subdirectories to their current working directory. Running this command will allow the user to make sure that their project was properly created as they should see 2 items listed as subdirectories: manage.py and [projectname] (shown below)

```
C:\Users\bflan\Desktop\sciam>dir
 Volume in drive C is OS
 Volume Serial Number is 00FA-E26D

 Directory of C:\Users\bflan\Desktop\sciam

10/01/2018  09:17 PM    <DIR>          .
10/01/2018  09:17 PM    <DIR>          ..
10/01/2018  09:17 PM               552 manage.py
10/01/2018  09:17 PM    <DIR>          sciam
               1 File(s)            552 bytes
               3 Dir(s)  367,081,410,560 bytes free
```

https://docs.djangoproject.com/en/2.1/intro/install/

Appendix V: Accessing the Server

sciam.wpi.edu

How do I access it?

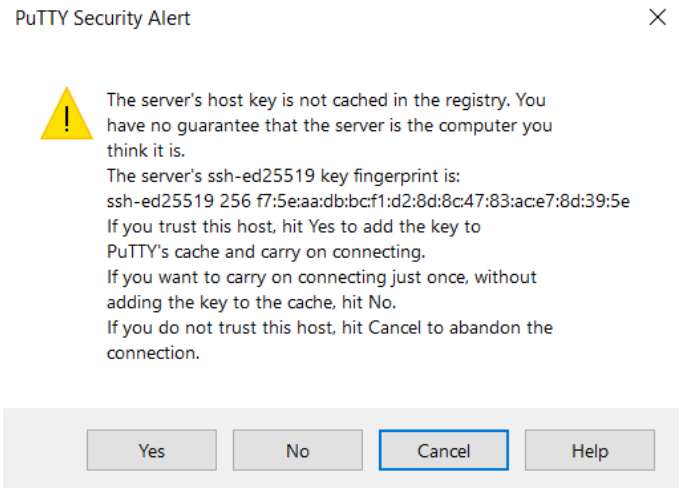
Download latest version of PuTTY to execute terminal commands on the server (ex. runserver)

In PuTTY:

Host Name (or IP Address): sciam.wpi.edu

Click open

If the following window is displayed, click yes.



username: bflanagan

password: ***[SEE YOUR ADVISORS]***

run the following commands to start the server (after opening a new session of PuTTY):

bflanagan@sciam:~$ **sudo bash**

[sudo] password for bflanagan: ***[SEE YOUR ADVISORS]***

root@sciam: ~# **screen -r**

OR

root@sciam: ~# **screen -d -r [screen number]**

root@sciam: ~/sciamproject/sciamproject # **python manage.py runserver 0.0.0.0:80**

use screen -d -r [screen number] when you get an error like the following after running screen -r:

There is a screen on:

29088.pts-0.sciam (01/15/2019 04:26:38 PM) (Attached)

There is no screen to be resumed.

In this case the screen number is 29088.

A screen is basically a constantly running terminal session, which allows us to terminate our PuTTY sessions but still keep our django webserver running on sciam.wpi.edu

run the following commands to stop the server (after opening a new session of PuTTY):

bflanagan@sciam:~$ **sudo bash**

[sudo] password for bflanagan: ***[SEE YOUR ADVISORS]***

root@sciam: ~# **screen -r**

root@sciam: ~/sciamproject/sciamproject # **^C**

^C → CTRL + C

Download an SFTP/FTP Client like WinSCP to access the files on the server

In WinSCP:

LOGIN

Session:

File Protocol: SFTP

Host name: sciam.wpi.edu

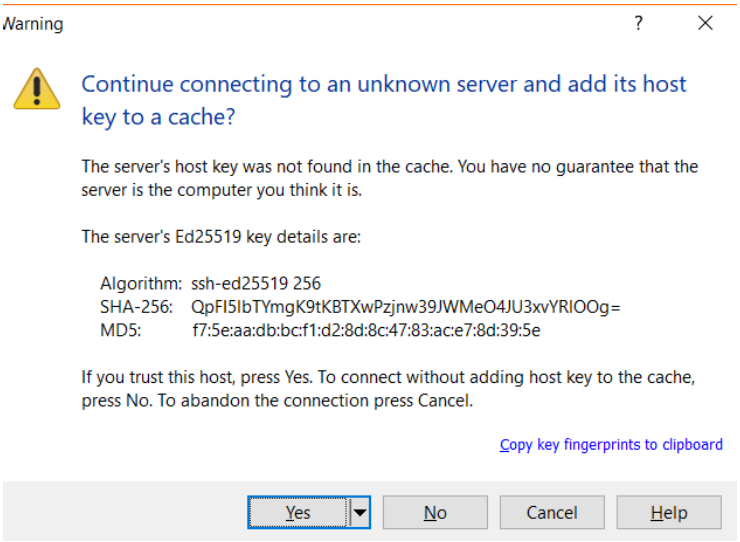
Port number: 22

username: bflanagan

password: ***[SEE YOUR ADVISORS]***

click login

If the following window is displayed, click yes



navigate to the */home/bflanagan/sciamproject/sciamproject/* directory

now you can access all files that have to do with the site