

## **Perseus Web2 Final Report**

cs467 Capstone

Date: Dec 1, 2017

Perseus Web2 Team:

Ben Fondell (fondellb)

David Hijirida (hijiridd)

Greg Niebanck (niebanckg)

### **1. Program Introduction and Description**

This project is a single page web application that allows users to see a graphical representation of the internet links from a user-defined, starting URL. In addition to the starting URL, the user must indicate the used search algorithm, and limit the number of nodes for which to be displayed. The BFS search algorithm will begin with the starting URL and process all links from the page, and then all links from each subsequent page until the node limit has been reached. The DFS algorithm will randomly select a single link from the available on each page until the node limit is reached. In this way, the BFS representation will look like a broad tree pattern while the DFS will look more like a random path. The user may explore the trees graphically or view the nodes in list form with associated statistics. The user may also enter a keyword along with the search parameters that will halt the search and display the node at which the keyword was found.

Our intention was to build the application such that it might accommodate any realistic web address requests by the user. The application would then return a graphical representation of url references across web pages. The user would be able to easily differentiate the difference between depth first and breadth first search by examining the graphical output. Lastly, the user would need to comfortably navigate the application with a simple interface.

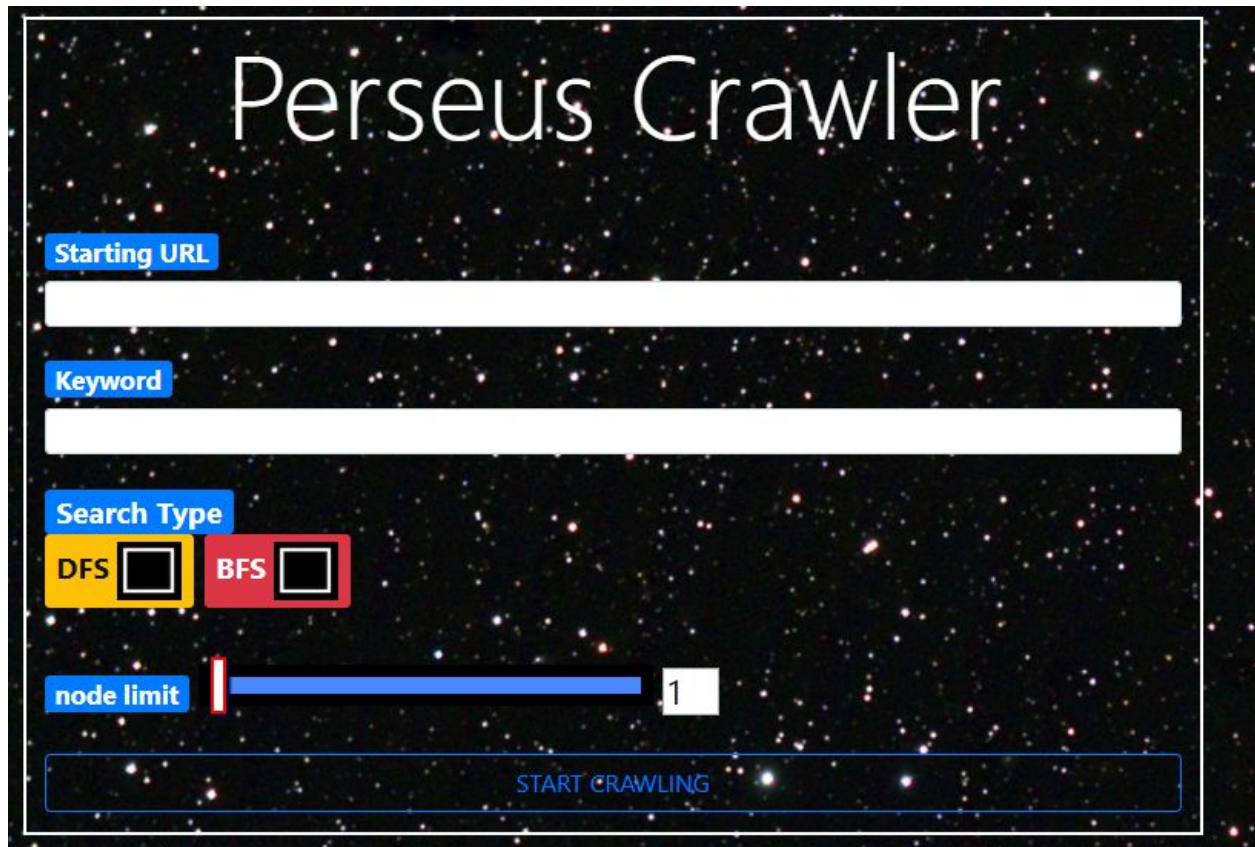
The program's primary requirements (from the user point of view) are summarized below (for reference, Appendix A contains the full requirements):

- The user will enter a starting web page, indicate depth-first or breadth-first traversal, and specify a numeric limit that stops the crawl from continuing indefinitely. For example, a limit of 1 means display all pages linked from start page. Limit 2 means display all pages linked from limit 1 pages, and display all limit 1 pages.
- The results will be transmitted back to the web page for graphical display and a small amount of manipulation (perhaps zooming, or displaying and/or hiding labels). Results displayed should include the title and URL of each page, perhaps controlled in such a way as to appear only when the user hovers over them to conserve space (this is up to you). The user must be able to click on one of these page nodes which should open up that page in a new tab or window. You must display the nodes graphically: do not use only text. Graph nodes and lines should not overlap.
- As an option, the user should be able to type in a keyword that will halt the program from searching when it is discovered on a page (this page and keyword are not being searched for, they are merely being encountered by chance). This page must be highlighted in the graphical results shown to the user.
- The web page should store past starting pages (with possible associated keywords) in a cookie on the users computer that can be displayed, and restarted.

### **2. Instructions**

Our website can be found at this url:

<http://ec2-54-148-189-221.us-west-2.compute.amazonaws.com:16000/>

The image shows a web form titled "Perseus Crawler" set against a dark space background with stars. The form includes a "Starting URL" label above a long white text input field. Below this is a "Keyword" label above another long white text input field. The "Search Type" section features two colored boxes: a yellow "DFS" box with an unchecked checkbox and a red "BFS" box with an unchecked checkbox. A "node limit" label is positioned above a horizontal blue slider bar, which is currently set to the value "1" in a small white box. At the bottom of the form is a large blue button labeled "START CRAWLING".

# Perseus Crawler

Starting URL

Keyword

Search Type

DFS ☐ BFS ☐

node limit  1

START CRAWLING

## Starting a crawl

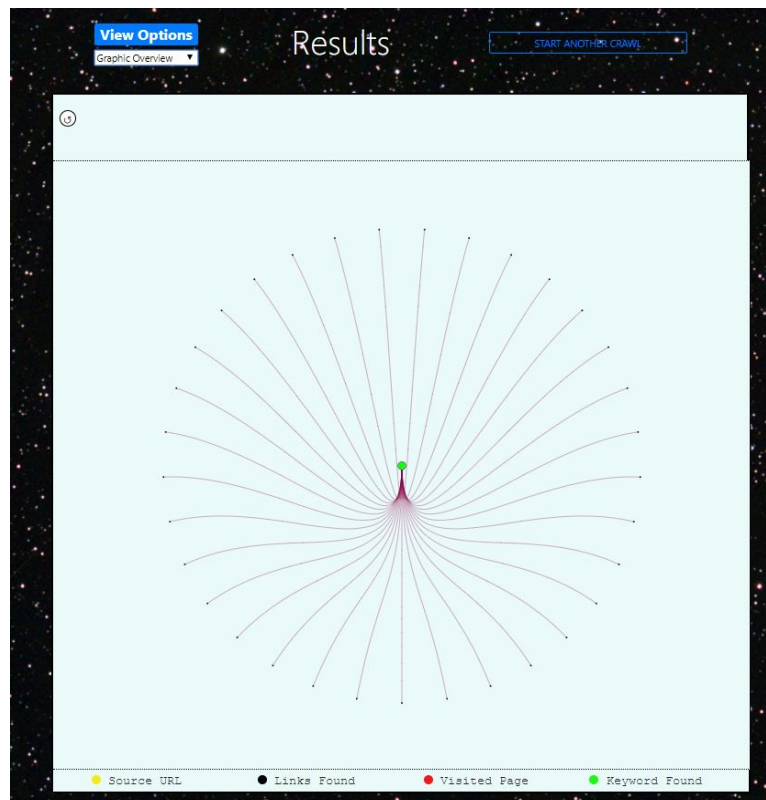
- Under "Starting URL", input a valid url to a website.
  - Note: the following url forms are accepted: "<https://google.com>" or "[www.google.com](http://www.google.com)" or "[google.com](http://google.com)". You may add a keyword to the crawl by entering it in the text field below the blue keyword label. Entering a keyword is optional.
- Choose your search algorithm by selecting appropriate box under the "Search Type" label. Check the box inside the red BFS label for a breadth first search, or the box inside the yellow DFS label for a depth first search.
- Use the slider next to the node limit label to select number of sites to visit during the crawl.
- Start the crawl by clicking the start crawling button at the bottom of the form.
  - Entering invalid urls or a site that can't be reached will result in a message on the form stating, "*The starting url is invalid*"
  - Neglecting to choose a search algorithm will result in a message on the form asking the user to select a search type.
- The parameters entered for each search are saved in a cookie on the client's browser. These cookies last one hour and are used to populate the "Search History" table. (This table appears below the form for users who have the cookie in their browser data)

Search History				
#	Starting URL	Keyword	Search Type	Limit
1	http://www.google.com		BFS	9
2	http://www.google.com		DFS	3
3	http://www.google.com		BFS	9

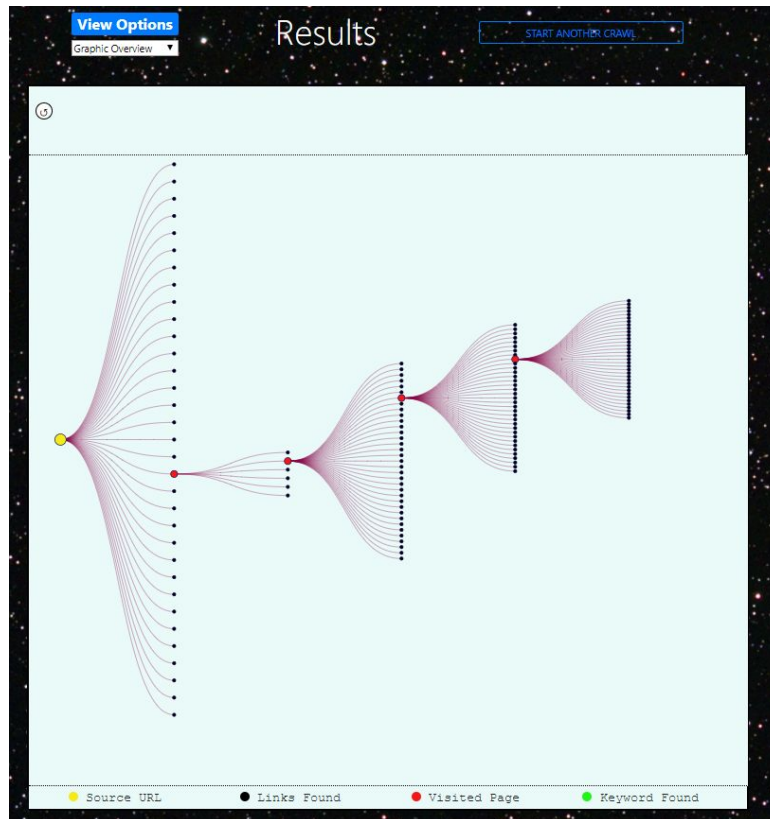
- Using the Search History Table: The Search history table may be used to initiate previously conducted crawls. The rows of the table are sorted with the most recent searches listed at the top of the table. Clicking on the rows in the table will initiate a repeat search with all the corresponding search parameters.
- Progress is indicated by a progress bar which appears once the crawl has been initiated. Once the progress bar is full the page will display the results.
  - Note: If you enter a keyword in your crawl, the search will halt if the keyword is found on a site which was visited during the crawl.

## Viewing the Results

BFS search data is represented as a circular expansion of nodes emanating from the root node displayed in the center of the graph.

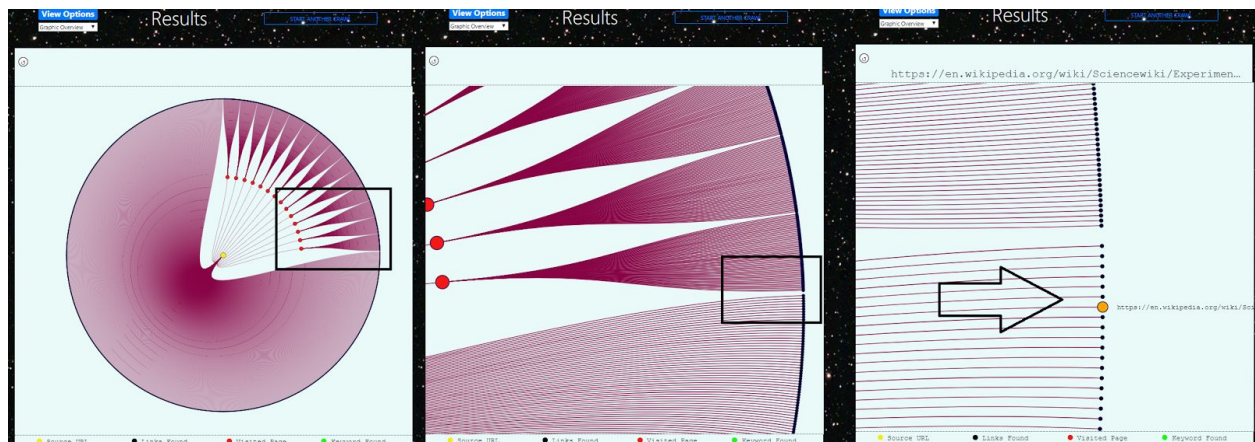


DFS search data is represented as a tree with the root node displayed on the left and child nodes branching to the right.



### Graphic Overview:

For both search types, the default viewing option is 'graphical overview'. The display plots all the sites visited during the crawl as well as all of the available links found on those pages. Each node is color coded to identify the root node, sites visited during the crawl, and links to sites that were not visited. The node with content containing the keyword will be colored green. A key for deciphering the color code is displayed beneath the graph.



The graph may be zoomed and panned to better navigate the nodes. You may un-zoom and recenter the graph using the reset button in top left corner. The url details will be made visible in the top center of the viewing area when hovering the cursor over a node. Clicking a node will display the associated page in a new tab in the user's browser.



## Node List:

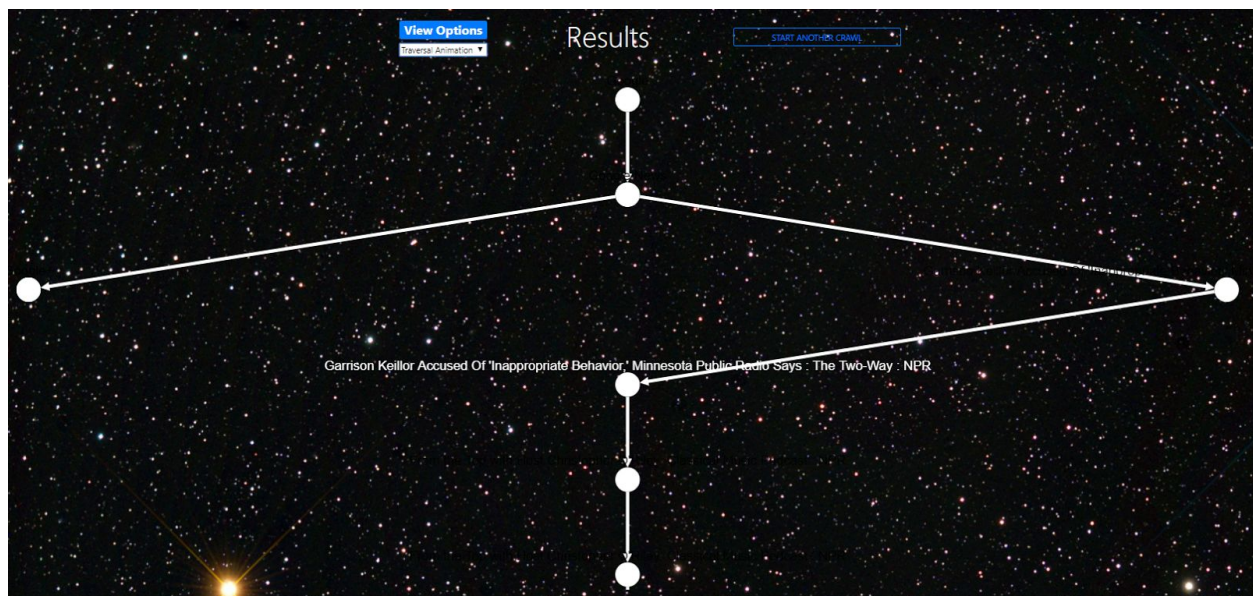


#	URL	Parent Row	Children
1	http://www.google.com	root	33
2	https://photos.google.com/?tab=wq&pageId=none	1	6
3	https://www.google.com/chrome/?hl=en_US	2	35
4	https://www.google.com/chrome/?hl=en_US/plus.google.com/+Chrome	3	35
5	https://www.google.com/chrome/?hl=en_US/plus.google.com/%20Chrome/www.youtube.com/user/googlechrome	4	35

Viewing options may be adjusted using the drop down menu at the top left of the screen. The user may select a “Node List” which shows a table starting with the root url and the sites that were visited and analyzed during the crawl. Each row displays the URL of the visited site, the number of links on that site, and the row number for the parent site. Clicking a row in the node list will open a new window in the browser with that corresponding URL.

## Traversal Animation:

The final view option called “Traversal Animation” which shows the order in which the nodes were visited in relationship to each other. This option is considerably less complex than the graphic overview. This option is a product of extra slack time and aims to add to the constellation theme in an aesthetic way by displaying the traversals as connected dots in space. Each website’s title is displayed above the node upon mouse over. Clicking on a node will open the website in a new tab in the browser. Nodes can also be dragged around just for fun.



Use the ‘Start Another Crawl’ button to return set up a new new crawl form. This is a single page application so using the back button will navigate the user away from the page.

## 3. Software Design and Architecture

A listing of which software libraries, languages, APIs, development tools, servers, and other systems you used to create the software, and deeper discussions of any of those you want to talk more about.

## Dataflow and Validation Discussion

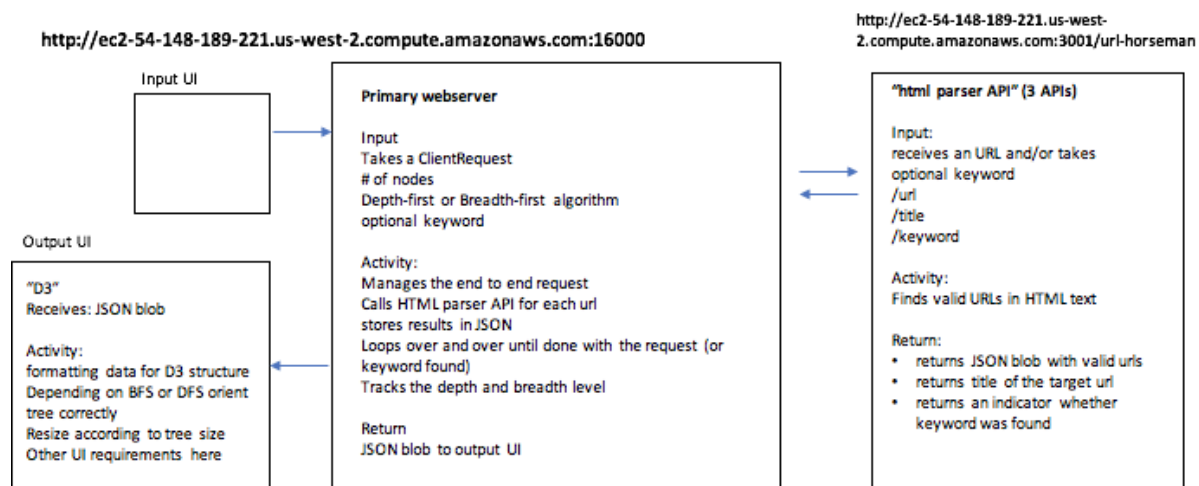
### Links to Github repositories

<https://github.com/GNIEBANCK/PERSEUS-WEB-CRAWLER>

[https://github.com/hijirida/cs467PerseusWeb2\\_HTMLParser](https://github.com/hijirida/cs467PerseusWeb2_HTMLParser)

[https://github.com/benpauldev/D3\\_Spikes](https://github.com/benpauldev/D3_Spikes)

See Appendix C for our initial architecture thoughts which were overly complicated and required a database. We chose the following architecture because it was a good separation of concerns and we could work fairly independently from each other. The three main components were (1) data visualization (Output UI), (2) the primary web server taking the initial request and coordinating with APIs to manage the life of the request and format output for the UI, and (3) the backend APIs to parse html and find urls, find titles, and find keywords.



## Technologies Discussion

Here are some brief descriptions of the technologies we used:

### Bootstrap

This css library was used for styling most of the components on the main input form.

### PhantomJS

<http://phantomjs.org/> PhantomJS is able to emulate a back-end or "headless" browser capability. While it is more than just a back-end browser (it can be used for testing, screen capture, network monitoring, etc.), we used it primarily to scrape the website of a target url and look for components such as anchor tags with href elements, page title, and search for a keyword in the primary text visible to a typical user visiting the webpage. It is a widely used program, even in large commercial companies like Amazon. PhantomJS is NOT written as a Node.js module so we need a bridge from Node to interact with PhantomJS (see horseman below). We investigated and tested other options to PhantomJS including node modules like JSDOM and Cheerio.

### Horseman

<https://github.com/johntitus/node-horseman> Horseman lets you run PhantomJS from Node. There are a number of node modules that bridge calls to PhantomJS and we tested a few, but horseman seemed to be the simplest one

(and one that we got working correctly!). Horseman makes extensive use of Javascript Promise calls, allowing asynchronous calls to be chained) and because we were not very familiar with how this worked it took a while for us to implement this correctly. There are a number of other node modules that work to bridge calls to PhantomJS include “node-phantom”, “phantomjs-node”, “phantom-bridge”, etc. (Note: At the time we implemented horseman, documentation indicated that was still under active development, but as of the week of Dec 1, 2017, they updated their README on github to indicate that it’s no longer maintained.)

### D3

This is a javascript library for graphically displaying data in the DOM. It is an open source library that allows the user granular control of the form and function graphically represented data. For this project D3.js was selected to handle the primary graphical representation of the web crawler. Documentation is provided at <https://d3js.org>.

D3 proved an ideal library for this project with few limitations. In early experiments with the library, graphical output was limited to the operation of the internal function of the library. Additionally, the graphics library functions for displaying in a tree hierarchy require specific input parameter formats. This was remedied by developing algorithms for re-parsing the input json and building the data hierarchically. With these modifications the library functions can produce very basic models independently. Advancing beyond basic models proved to be the library's strength. The library is designed so a developer may modify granular details of the graphics to accommodate for certain needs. It quickly became apparent that for actual test data with potentially thousands of nodes, the graphics would need to adjust to accommodate. D3 allowed for the programmatic manipulation of function inputs and outputs as well as manipulation of methods themselves. This made possible a series of graphical enhancements that would accommodate any reasonable data set in an organized useful interface.

### Cytoscape.js

This is a Javascript and css library designed for displaying network diagrams. This library can be used to create highly complex diagrams. For this project the, cytoscape was used to create the traversal animation from the crawl data. Examples and documentation can be found at <http://js.cytoscape.org/>.

## **4. Software Development Observations**

### 4.1. Division of Labor

We assigned work based on what interests each of us wanted to explore, e.g., graphics package vs. API creation, etc. We also considered how to define work so we could run as independently as possible. While our team members were free to explore and edit any portion of our code, the list below describes the primary focus of our three team members and the primary activities. Net net, we functioned well as a team.

#### **Ben Fondell** (Front-end, graphical rendering focused)

- Contributed to planning and architecture discussion.
- Participated in weekly meetings.
- Researched graphics library options and expected graphics requirements.
- Researched graph types and their associated value to our data requirements.
- Developed algorithms for parsing input json according to the requirements of the d3 library.
- Developed algorithms for reassembling json data into hierarchical structure based on BFS or DFS.
- Developed scaling algorithms for depicting trees with thousands of nodes while maintaining organization and preserving user experience.
- Implemented interactivity with graphics including pan, zoom, mouseover, click ...
- Collaborated with group on the integration of graphics scripts, html, and css with the server architecture.
- Tested and debugged for errors involving inaccurate display of data in the graphics.
- Assisted in testing and debugging the web application as a whole.
- Assisted in drafting project plan, midpoint check, final report, and demonstration documents.

**David Hijirida** (Back-end / API focused)

- Contributed to planning and architecture discussion.
- Participated in weekly meetings.
- Researched back-end, headless browser packages and tested ease of use, performance (speed), and functional requirements
- Developed APIs to be called by the primary web server including (1) given target url, return all valid urls found on the target web page; (2) given target url, return title found on target web page; and (3) given target url and keyword, return position where the keyword was found (or not found) on the target web page.
- Created documentation for APIs and test cases
- Assisted with testing and writing bug reports
- Assisted in writing the demonstration and final report documents.

**Greg Niebanck** (Middleware and User experience focused)

- Contributed to planning and architecture discussion.
- Participated in weekly meetings.
- Developed structure for the hosted website
- Developed test graph functions in Javascript functions for creating the space to test search algorithms.
- Developed BFS and DFS search functions in Javascript and tested them with the graphs created from the test functions.
- Used the test graphs and search algorithms to provide usable input data for the graphics module.
- Adapted the search algorithms used in testing to work with asynchronous HTTP calls from the client, to the server, to the parsing API, and back packaged in JSON to be sent to the graphics module.
- Organized the html for the single page application.
- Fully implemented the front-end input forms using Javascript CSS Bootstrap and Node.js with express handlebars.
- Developed functions using cookie-parser to save data which is later accessed for the search history table.
- Implemented node-list and traversal animation view options
- Implemented non-browser based error handling for the input forms.
- Assisted in manual testing of the parsing API and the graphics modules.
- Assisted in writing the demonstration and final report documents.

#### 4.2 Planning

We used an agile process where we planned our next steps using weekly sprint planning, using Google Hangouts for video conference calls. Our intentions were to use tools like the project plan (initially created during project planning phase), Trello to track tasks and velocity, and Github to share code. We share some of our observations and lessons learned in the Q&As below.

*Project Planning and coordination*

Q.: Did we rely on the initial project plan? How useful was it?

A: Mixed. For the most part we didn't follow the original plan we laid out. ***See Appendix B for a delta between what we originally planned vs. what actually happened.*** The original plan was helpful in that it helped organize the required tasks on a general level. Each of the group members worked with a system or library in which they had very little previous experience. For this reason, adding the slack time in our plan was very important. Our inaccuracy in estimating the time required for each section of the program was apparent after the first week.



The primary drivers for deviation from the original project plan included the following: (1) not being able to estimate correctly the amount of time for researching which technologies to use (and sometimes revisiting our tech decisions and redoing functionality) and (2) component level debugging (especially with technologies with poor documentation) and (3) issues found through manual testing after integrating our components. *Please see Appendix B for a high level week by week deviation from project plan.*

Q.: How helpful was using Trello as a planning and tracking tool?

A: After the first week, we did not use this for sprint planning. The overhead of maintaining cards and moving them to the right location did not provide enough benefit vs. just weekly video conference calls..

Q.: How useful were conference calls?

A: For the first 4-5 weeks this was fine when we had easily decoupled work. It was also useful to see where folks were stuck and to gauge development velocity. However when it came to talking through how pieces were integrating or to root cause specific integration problems, it's easier for 1 to 1 calls or conversations to work through issues vs. in groups. We used Google hangouts messaging features to track historical conversations (and pseudo documentation) which were useful when we needed to find earlier references to URLs or other information we needed but didn't remember.

#### *Repositories and Sharing*

Q: How did we use Github and other tools?

A: We used a mixture of github and uploading code to canvas files to integrate code. We didn't do as much parallel code checkin to github as we thought we would in the beginning of the project. Rather we kept code isolated (e.g., HTML Parsing to find URLs never needed to be integrated) or we passed code through files for someone more familiar with the base code to integrate (e.g., Ben passing D3 / XXX files to Greg to integrate into primary front end code).

#### 4.3 Ideas for future improvements

More pair programming or one to one debugging – It was harder sometimes to dive into someone else's code and it took some time to ramp up. As the code changes and you are trying to help debug it would help to share screens and walk through issues. Our process for version control would also need more organization. We occasionally introduced conflicts in code by not incrementally building a single master version but instead, updating independent code with many changes and then integrating into the master version.

Performance could also be further refined. For certain inputs the application needs to spend a considerable amount of time crawling and then displaying the data. This is likely due to the limited processing power of the host server (Amazon EC2 instance) as well as the significant overhead required of network communication for large data sets.

Future improvements would also be determined through long term use and testing. There will likely be rare cases where portions of the program produce errors and need to be corrected. This is to be expected with any application and is a requirement of long term development.

#### **5. Conclusion**

Generally speaking, we believe the application succeeds in accomplishing the goals presented by the group. The independent features developed by each member coordinate well and produce an application that is effective and simple to use.

The group was able to maintain the schedule of the project plan and accommodate when unforeseen issues arose. Our initial plan included several weeks of slack time to enhance the application and add features beyond the requirements. We ate up our slack when we found errors that only appeared after we integrated our components and tested them end to end. Through further iteration and in some cases, pivoting of technologies, we were able to produce a version that satisfied the basic requirements.

Through completing this application, we gathered significant experience with our chosen technologies and team software development on a ground-up project. We took pleasure in creating something that represents the culmination of our experience in this program. We hope you enjoy the final product.

## Appendix A. Project Requirements

WEB2: GRAPHICAL CRAWLER: Write a web page that crawls the web, following links from place to place, building a graphical model of places traveled. In this project, you'll be creating both a web site and \*nix-based program that crawls the web starting at a particular home page, and maps out all links graphically for the user. The artistic presentation of the graphics are up to you, but the final project must satisfy the following requirements:

- At your site, the user will enter a starting web page, indicate depth-first or breadth-first traversal, and specify a numeric limit that stops the crawl from continuing indefinitely.
- The user input will be transmitted to a program running on a server somewhere.
- The program will begin to follow the links of that starting web page, creating a log of where it visits.
- In a depth-first crawl, the program will start at the start page, *randomly* choose one of the links on that page, then follow it to the next page. Then, at the next page, it randomly chooses a link from the options available, and follows it. This makes a chain from the starting page. This continues until the program hits the page limit indicated.
- In a breadth-first crawl, the program will follow ALL links from the start page, and ALL links from each page it visits, until the crawler has reached the limit of pages deep (as measured from the start page) it should visit. Since this is likely to return a huge, sprawling graph, consider limiting the user's input with this kind of search to a small number. For example, a limit of 1 means display all pages linked from start page. Limit 2 means display all pages linked from limit 1 pages, and display all limit 1 pages.
- The results will be transmitted back to the web page for graphical display and a small amount of manipulation (perhaps zooming, or displaying and/or hiding labels). Results displayed should include the title and URL of each page, perhaps controlled in such a way as to appear only when the user hovers over them to conserve space (this is up to you). The user must be able to click on one of these page nodes which should open up that page in a new tab or window. You must display the nodes graphically: do not use only text. Graph nodes and lines should not overlap.
- As an option, the user should be able to type in a keyword that will halt the program from searching when it is discovered on a page (this page and keyword are not being searched for, they are merely being encountered by chance). This page must be highlighted in the graphical results shown to the user.
- The web page should store past starting pages (with possible associated keywords) in a cookie on the users computer that can be displayed, and restarted.

Tools and Technology Requirements:

- Your project **MUST** be hosted somewhere for me to test for grading purposes, though you will be required to submit source code.
- Do not use Javascript-style "alert()" pop-up boxes to relate information to the player. Your user interface should display all information without resorting to asking the web browser to pop-up a dialog.
- Do not use Adobe's Flash player or any other encapsulated app.
- Do not use Django or any form of Content Management System (CMS) to build the pages or site for you.
- You may use [Node.js](#), [bootstrap](#), [React](#), [Angular](#), and/or [jQuery](#), etc. if you like.

## Appendix B Project Plan

At the beginning of the project we submitted a project plan, detailed under “Original Plan” column in the table below. Our project plan quickly became obsolete after we started the project and pivoted according to research required or issues to resolve.

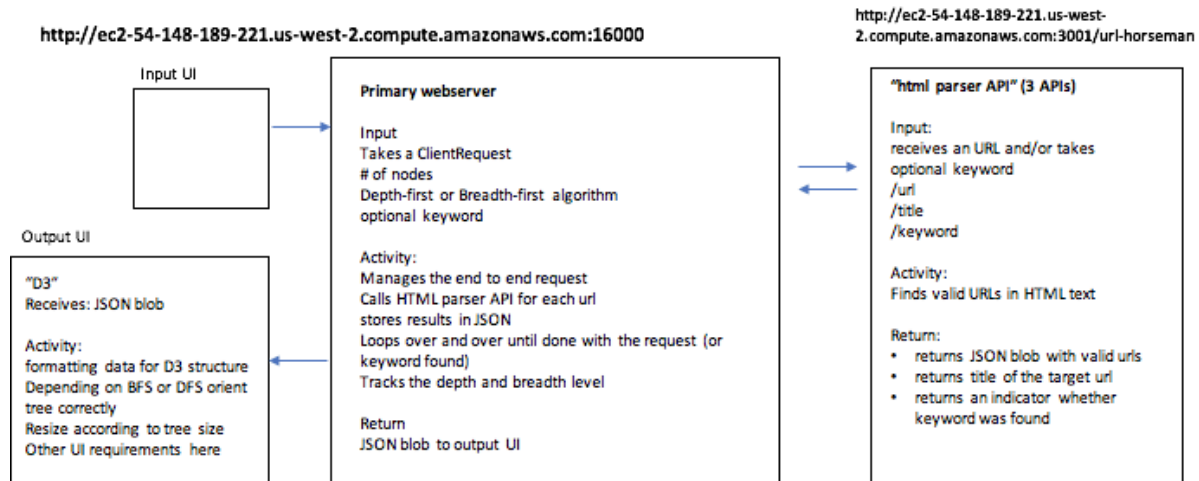
Week	Original Plan (initial team submission)	What Happened / Observations
Week3	<ul style="list-style-type: none"><li>• Research graphic packages</li><li>• Research html parsing algorithms</li><li>• Set up skeleton server code using node.js implement low fi client-side forms and RESTful API stubs</li></ul>	<ul style="list-style-type: none"><li>• Spike story to research graphic package.</li><li>• Research html parsing algorithms - prototype will take a url and return html (visually check if html is rendered).</li><li>• Set up skeleton server with route stubs. Researched operating systems interaction using Node.js</li></ul>
Week4	<ul style="list-style-type: none"><li>• Run i/o spikes with chosen graphics package</li><li>• Fully implement DFS algorithm</li><li>• Fully implement client-side display form and display - organize video information</li></ul>	<ul style="list-style-type: none"><li>• Setup first routes for html parsing POST /url. Given a url, return a list of urls. Learned to use Cheerio nodejs package to scrape</li><li>• Integrated graphics spike into server code stub.</li><li>• Tested communications between the server and the parsing API</li><li>• Built system for generating BFS and DFS graph traversals to sub as input for graphics module until parser was up and running</li><li>• Graphics package tested with hard coded test data.</li></ul>
Week5	<ul style="list-style-type: none"><li>• Fully implement low – fi graphical representation</li><li>• Fully implement BFS algorithm</li><li>• Fully implement back-end data transfer between APIs</li></ul>	<ul style="list-style-type: none"><li>• Wrote documentation for html parsing API and tests. Manually checked results with different websites to identify gaps in scraping capability - some sites like Yahoo could not be scraped</li><li>• Improved test suite in order to save results and create specific edge cases.</li><li>• Converted search algorithms to work with http requests from the client side web code.</li><li>• Debug and implement graphics with hard coded test data.</li></ul>
Week6 Midpt Check In	<ul style="list-style-type: none"><li>• Integration Testing / Improvement research and discussion of additional features in graphics</li><li>• Integration Testing / improvement research, performance enhancements</li><li>• Integration Testing / Research front-end UX enhancements</li></ul>	<ul style="list-style-type: none"><li>• Writing document for midpoint check-in.</li><li>• Tested other node.js html rendering packages like JSDOM and 3 or 4 other packages.</li><li>• Bootstrap Spike. Cytoscape Spike. Made improvements to client side UI..</li><li>• Integrate graphics and test.</li></ul>
Week7	<ul style="list-style-type: none"><li>• Slack / Improvement exploration</li></ul>	<ul style="list-style-type: none"><li>• (continued from last week) Tested other node.js html rendering packages like JSDOM and 3 or 4 other packages.Determined that performance differences were marginal.</li><li>• Improve graphics for large data sets.</li><li>• Started the debugging process. Continued improvements to the UI</li></ul>
Week8	<ul style="list-style-type: none"><li>• Slack / Improvement exploration</li></ul>	<ul style="list-style-type: none"><li>• Tested PhantomJS because websites rendered with heavy javascript could not be scraped accurately.</li></ul>



		<ul style="list-style-type: none"> <li>• Tested different nod.js packages to bridge PhantomJS calls (e.g., horseman, node-phantom, etc.)</li> <li>• Various improvements made to the UI including error handling. Continued debugging.</li> <li>• Improve graphics for large data sets and integrate added interactivity features.</li> </ul>
Week9	<ul style="list-style-type: none"> <li>• Finalize graphic enhancements</li> <li>• Finalize performance enhancements</li> <li>• Finalize UI enhancements</li> </ul>	<ul style="list-style-type: none"> <li>• Implemented new scraping code using Phantom JS + horseman. Difficult time debugging due to lack of good documentation and examples</li> <li>• Integrated and tested additional view options and continued to improve the UI.</li> <li>• Implement added features in graphics that fulfill all requirements.</li> <li>• Begin writing final documentation.</li> </ul>
Week10 Final Submission	<ul style="list-style-type: none"> <li>• Finalize graphic enhancements</li> <li>• Finalize performance enhancements</li> <li>• Finalize UI enhancements</li> </ul>	<ul style="list-style-type: none"> <li>• Implemented APIs for returning page Titles and for determining whether a page contains a keyword or not.</li> <li>• Integrated and tested component upgrades, continued to improve UI.</li> <li>• (Continued) Writing final documentation</li> </ul>

## Appendix C. Architecture Drawings

### Revised and Updated Architecture Drawing



### Original Architecture Drawing

This was submitted when we first started the project plan

