CS419 Tango Group
Graphical Webcrawler

Darby Beltran
Nathalie Blume
Thomas Luong

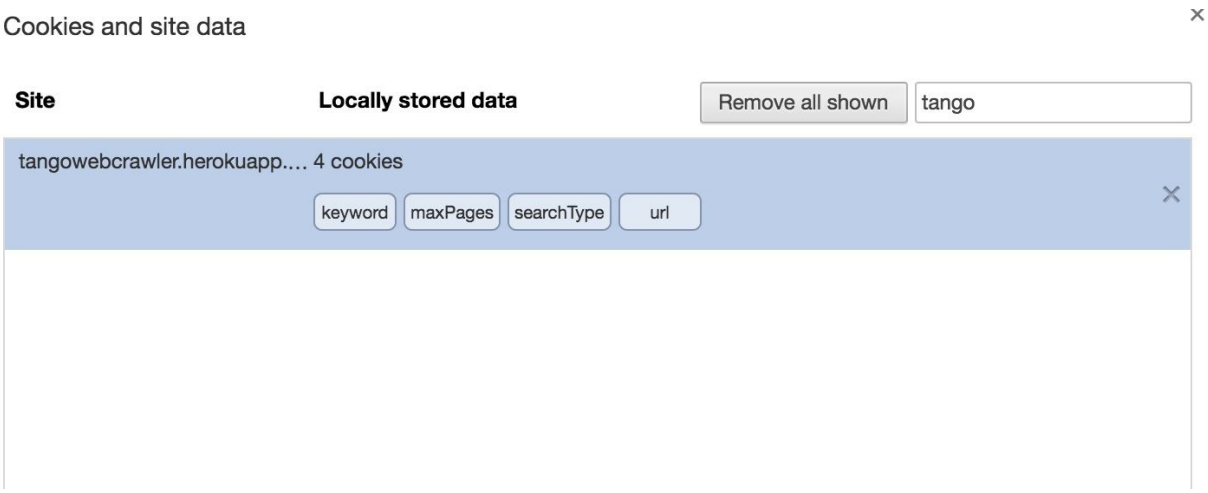
URL: https://tangowebcrawler.herokuapp.com/#/?_k=vpk2w1

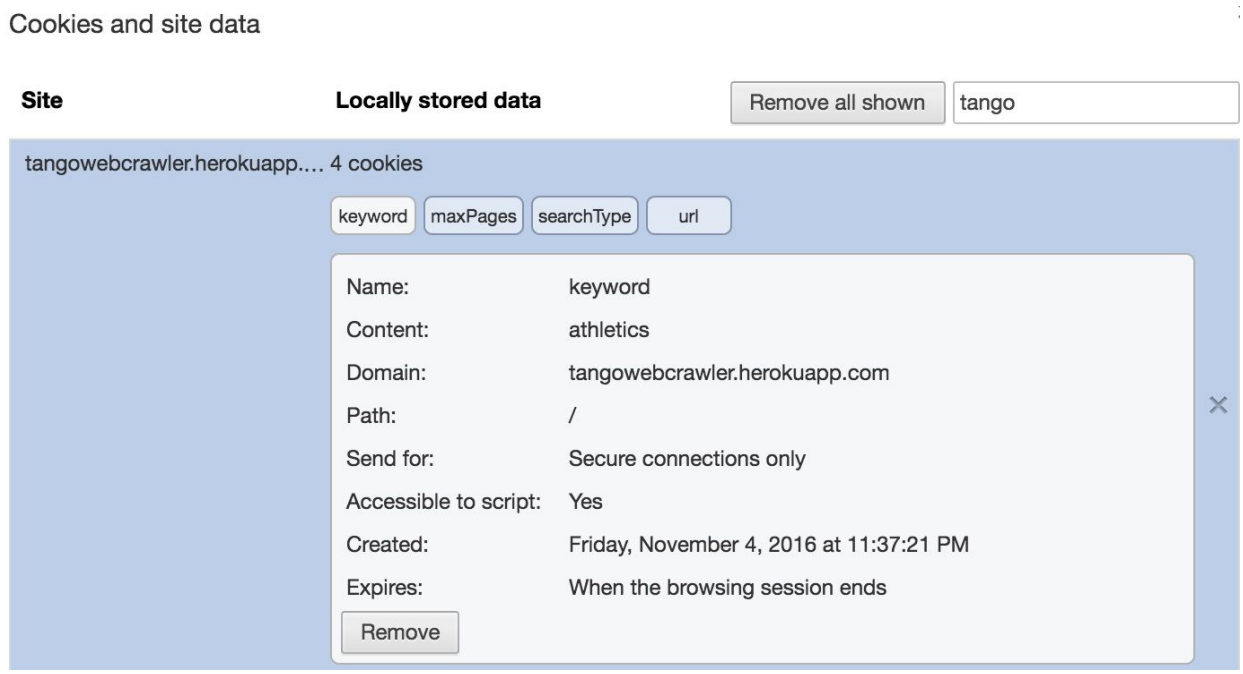
How to interact with webpage:

- Begin by accessing the Crawler Form (should be the main page).
- From there you can enter the URL that you want to begin your breadth-first search at (Box for starting website).
- You then indicate the depth of your search.
- You then indicate a word to look for on a page and that ends the search.
- A pull-down menu lets you select the search type. Depth-first searching is not yet implemented so please select “Breadth-first Search”. If you want more information on search types, click “Algorithm Info”.
- Click “Go”.

At this moment in time, the webpage can create cookies from the search form but sending them to the server through a GET request is still undergoing testing and research from our React developer.

Here is what it should look like after submitting the following inputs into the form.
(the form doesn’t do submit to anywhere yet, so don’t be alarmed if it doesn’t redirect anywhere, this is being developed)





After submitting the form, the parameters will be logged to the console.

It can also be seen (if Chrome is used as in this example)

After submitting the form, you can see the cookie being set in your console.

The individual cookies can be seen being set if the url below is typed into the browser.

chrome://settings/cookies

The cookie will be sent to the Server as a GET request, and then be provided as parameters for the web crawl. This will be worked upon in the upcoming week.

Hosting and Communication:

- Webpages hosted with Heroku are slow to start up at the beginning since it takes them a bit to warm up/wake up their instance. But subsequent uses after that are faster to load pages.
- Heroku allows instances of apps such as React, PHP/HTML, or Python/Flask, etc apps to run on their own micro virtual machine on Heroku's server. We felt this was a good choice since it is free and allows for local editing and easy deployment of code.
- Each account can have up to five active web applications running at one time, which is fantastic for users who would like to play around with a small application. It also allows for users to scale their application if they so choose. For our purposes, this is not required.

What the crawler does:

The crawler consists of Python code that carries out the following tasks:

- The crawler receives the user input from the UI. Right now we're still working on this and the arguments are still hardcoded. But the UI does launch the crawler, and we're close to a solution, using cookies to pass the arguments to the python crawler.
- The crawler assesses the search type and launches either the breadthFirst or the depthFirst function. Currently, only the breadthFirst function is implemented.
- In a bfs, the crawler lands on a page, collects the page's outgoing child-links and adds these links to a queue (a first-in-last-out container).
- The crawler searches the contents of the page for the stopping word.
- The crawler adds to a traversal dictionary the page's url, child-urls and a flag for whether the stopping word was found. The traversal dictionary is later used to build a graph.
- As long as the stopping word has not been found and the number of nodes to traverse has not been reached, the crawler lands on the next page in the queue. It tests whether the page has been visited before. If it has, it skips this page and lands on the next page. When it finds an unvisited page, the crawler performs again the tasks described above (gather links, scour contents, update traversal dictionary).
- When a search is complete, the crawler saves the traversal dictionary in JSON format to a local file. We are working on communicating the file to the visualizer.

How to interact with the crawler:

-- For Mac OSX

- do the following code:
- install homebrew if you haven't (http://brew.sh/)
- $ brew install pip

- $ brew install virtualenv
- $ cd WebServer
- $ source bin/activate $ pip install flask
- $ export FLASK_APP=WebServer.py
- $ flask run

-- For Windows

- install pip pip install virtualenv
- $ cd WebServer
- $ source bin/activate
- $ pip install flask
- $ export FLASK_APP=hello.py
- $ python -m flask run
    - Running on http://127.0.0.1:5000/
- Note: Windows installation is untested

The crawler will be able to take in a get request with a cookie with the format of:

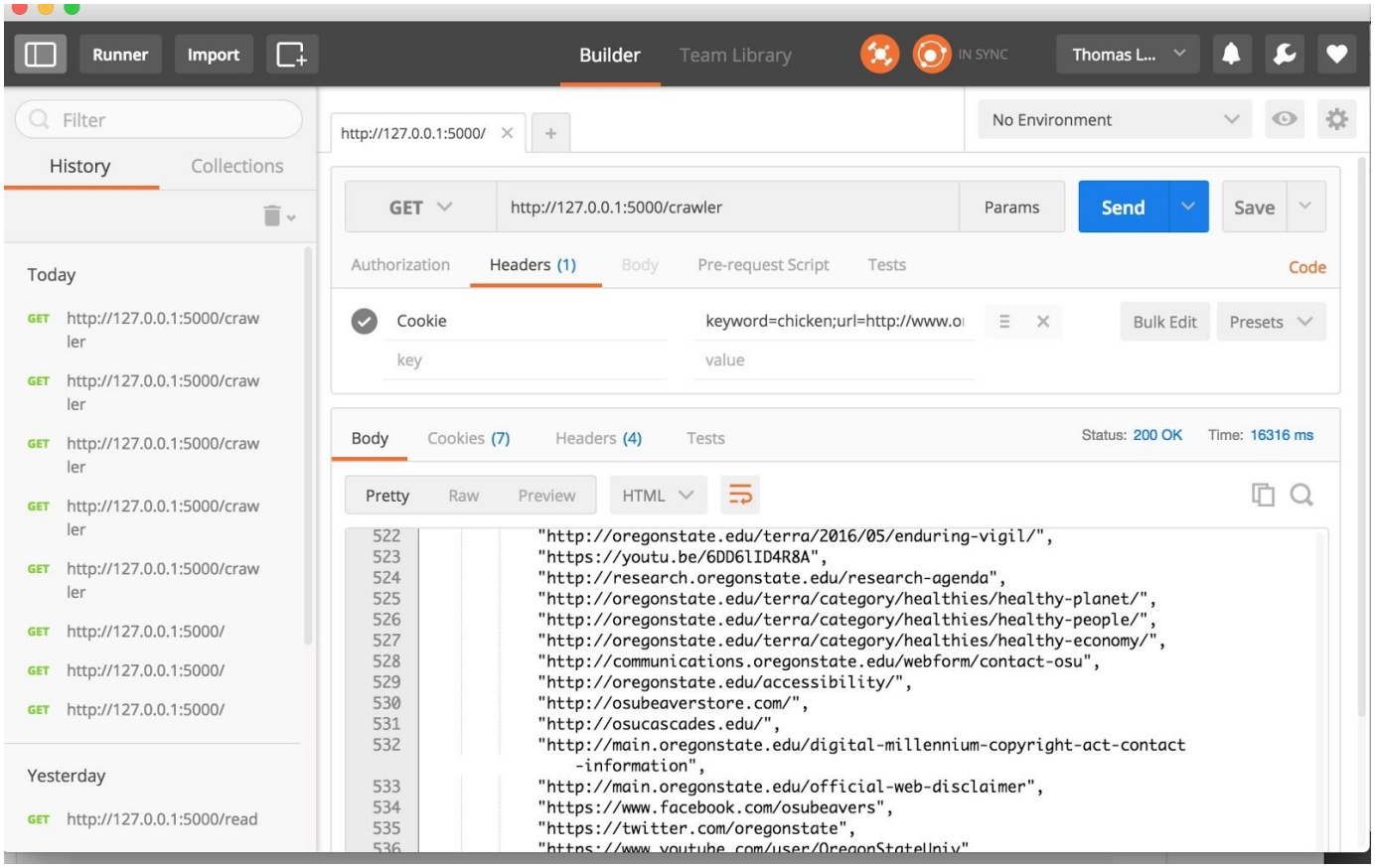
keyword=chicken;url=http://www.oregonstate.edu;maxPages=10;searchType=BFS

It is recommended to use an application called Postman to send cookies in through a GET request without the use of a form. Since the form and server are not quite connected yet, this is an ideal method of testing the endpoint.

Install Postman as a Chrome Web App, as well as the Postman Interceptor.

This URL explains how to do so clearly if help is needed: https://www.getpostman.com/docs/interceptor_cookies

Turn on the Postman Interceptor. Send a get request in a manner similar to this picture



The Header we are sending a GET request to, to the endpoint of http://127.0.0.1:5000/crawler, is a Cookie property with the value of:

keyword=athletics;url=http://www.oregonstate.edu;maxPages=10;searchType=BFS

Of course you could change the keyword, url and maxPages' value to anything you see fit.

keyword=cats;url=http://www.reddit.com;maxPages=200;searchType=BFS

In the body of the response should be the JSON object of the traversed links. Since DFS hasn't been implemented yet, BFS is the only option for the searchType that is available.

How to interact with the visualizer:

The visualizer and the server are not currently interacting, but we have a static representation at this URL: https://tangographicalcrawler.herokuapp.com/tangoGraphicalCrawler.html

- You can hover over each image to display its URL
- Once you click on an image it will open a new tab and navigate to that URL
- If you click the text next to the image it will zoom in on that node.

We are currently working on generating the json in a format that can be loaded right into the visualizer script so it will update with each search.