Tourism Destination Websites

Multimedia, Hypertext, and Information Access

Address: Virginia Tech, Blacksburg, VA 24061
Instructor: Dr. Edward A. Fox

Course: CS4624
Members: Matt Crawford, Viet Doan, Aki Nicholakos, Robert Rizzo, Jackson Salopek

Date: 05/08/2019

Table of Contents

List of Figures

List of Tables

1. Executive Summary

In the United States, cities like Philadelphia, Pennsylvania or Portland, Oregon are marketed by destination marketing organizations (DMOs) to potential visitors. DMO websites mostly are a collection of information on attractions, accommodations, transportation, and events.

DMOs are often run as non-profit organizations, typically funded through hotel taxes (aka lodging tax, bed tax, transient occupancy tax). Hence, it is not residents paying for DMOs. Rather, current visitors are paying for advertising to attract future visitors. Members of the boards of these nonprofits are, among others, representatives from local government and the tourism and hospitality industry. Hence, in some destinations the industry wants the DMO to sell attractions or even hotel rooms, while in other destinations businesses see that as competition and prohibit DMOs from selling anything, but only allow them to do advertising. This is just one example of how DMO websites can be different due to different mission statements, but of course these missions change over time.

Dr. Florian Zach seeks to conduct a historical analysis of the outbound and internal hyperlinks of DMO websites in order to learn what websites DMOs point to and how this changes over time. We will assist him to do this by analyzing data from the top tourism website from each state in the United States, 50 in total. Dr. Zach's expected impact of this research is the ability for tourism destination decision makers to learn past trends among DMOs' outbound links and potentially determine the future direction of destination online marketing

From the DMO data, will deliver a set of matrices from which Dr. Zach can perform his own analysis in Python [11]. Dr. Zach expects DMO websites to point towards a set of common websites like social media sites, TripAdvisor, Yelp, etc. as well as a set of more local websites.

We will construct a website which will link to our database of DMO data and contain two dynamic visualizations of the data. We originally planned this website for a Microsoft Azure host obtained for us by Jim Dickhans, the Director of IT at Virginia Tech's Pamplin College of Business.

2. Introduction

This report describes the semester-long team project focused on obtaining the data of 50 DMO websites, parsing the data, storing it in a database, and then visualizing it on a website. We are working on this project for our client, Dr. Florian Zach, as a part of the Multimedia / Hypertext / Information Access course taught by Dr. Edward A. Fox. We have created a rudimentary website with much of the infrastructure necessary to visualize the data once we have entered it into the database.

We have experimented extensively with web scraping technology like Heretrix3 and Scrapy, but then we learned that members of the Internet Archive could give us the data we want. We initially tabled our work on web scraping and instead focused on the website and visualizations.

We constructed an API in GraphQL in order to query the database and relay the fetched data to the front end visualizations. The website with the visualizations was hosted on Microsoft Azure using a serverless model. On the website we had a homepage, page for visualizations, and a page for information about the project. The website contains a functional navigation bar to change between the three pages. Currently on the homepage, we have a basic USA country map visual with the ability to change a state's color on a mouse hover.

After complications with funding and learning that the Internet Archive would not be able to give us the data in time for us to complete the project, we pivoted away from the website and visualizations. We instead focused back on data collection and parsing. Using Scrapy we gathered the homepages of 98 tourism destination websites for each month they were available from April 2019 to January 1996. We then used a series of Python scripts to parse this data into a dictionary of general information about the scraped sites as well as a set of CSV files recording the external links of the websites on the given months.

3. Requirements

We will acquire a dump of website snapshots using the Internet Archive [5], one snapshot for each DMO website for each month it has been active. These snapshots will be in the form of file directories of raw HTML data. The dataset will be very large, so we will need to store it in a facility provided by Virginia Tech. We will parse this data with Python to extract the outbound and internal links of the website from each given snapshot. From the parsed data we will enter it into our MongoDB database as well as build the series of matrices Dr. Florian Zach has requested. There will be a matrix for each year dating from 1998 to 2019, 21 in total. Each matrix will be a cross section of the 50 DMOs and the websites which the outbound links of those DMOs connect to during that year.

We will construct a website containing two visualizations specified by Dr. Florian Zach of the data we acquired. The first visualization is an interactive country map of the United States. Dr. Zach wants each state to be colored by a gradient representing the size of the DMO from that state (estimated by the number of pages on the website). Hovering over the state will display the details of the information being represented. Below the country map, he wants a slider which the user can move in order to shift the map chronologically, so the leftmost position of the slider will be January 1998 and the rightmost position will be March 2019.

The second visualization will be a bimodal node graph with line connections between DMO website nodes and external website nodes. The connections between a DMO node and an external website node will grow thicker (stronger) as the number of outbound links pointing from the DMO to that external website grows. There will be a slider similar to the one in the first visualization allowing the user to examine this relationship over time.

4. Design

4.1 Hosting

Serving our application as well as having a database and API that are dependable, scalable, and accessible are necessities. To determine what platform we host on, we worked closely with Pamplin IT to ensure that our application would be live indefinitely for minimal cost. We were presented with 2 options: hosting in a virtual machine or hosting in Azure. Hosting in a virtual machine presented security concerns, as the VMs would need ports opened to serve our full stack, so Pamplin IT suggested we host everything in Azure. Microsoft Azure provides the scalability and dependability that we desired as well as being in Pamplin IT's tech stack, thus was chosen. Our project was allocated a Resource Group with as many resources as necessary, and the resources hosted are listed below.

4.2 Database

The data we anticipate collecting from either grabbing snapshots from the Internet Archive manually or receiving a dump of all the websites in WARC format needs to be parsed and the resulting computed data (i.e., number of internal links on a page) needs to be stored in a structure that can be read quickly. The size of the resulting data, we anticipate, is going to be quite large, and NoSQL's design helps accommodate big data. As well, our data is not relational, so drafting a schema in NoSQL was the decision we made. Through extensive iteration, we were able to continuously revise the schema so that data was in the most efficient and descriptive structure. Our final schema is shown in Figure 1.

- websites
    - state
        - year
            - month
                - domain
                - pages <<< list
                - internal links{...} <<< key = originating page, value = internal page
                - external links{...} <<< key = originating page, value = full URL - do not shorten it here. Shorten it later on for the network matrix.

    - "meta tags"{...} <<< key = originating page, value = relevant meta tags from that page <<< if you have to make this multiple dictionaries, then do so... for title, keywords etc.
    - homepage
        - HTML-text <<< cleaned of all tags, hyperlinks etc.
        - title
        - keywords etc.
        - number of images <<< BY COUNT OF file endings (.png, .gif, .jpeg, .jpg etc. NOT COUNT OF img TAG
        - list of unique images {...} <<< key = image name, value = number of occurrences
        - internal links on homepage{...}
        - external links on homepage{...}

[Figure 1] Final schema architecture.

The NoSQL API we chose was MongoDB, as it's highly scalable, data is represented in JSON, and it has high performance. As well, MongoDB's integration into Microsoft Azure allowed us to provision a database quickly.

4.3 API

To retrieve the data stored in our database, we needed a strongly-typed API so that we could grab the data we need when we need it in the correct schema. To solve this problem, we chose GraphQL [6]. GraphQL, which is developed by Facebook, allows quick validation to ensure our data is in the correct format for reading. As well, we can always get the right amount of data instead of over-fetching or under-fetching, which reduces network load and the amount of API calls. On top of that, GraphQL allows for quick application development, and when we got started Apollo GraphQL Server, spinning up an initial API implementation was a breeze.

4.4 Front-End

For the front-end of the application, we wanted a single-page application framework or library that was lightweight and had enough support for querying large amounts of data. Our decision was between Angular and React, and we decided to go with React due to its lighter weight. As well, we needed a rich ecosystem for component libraries, which React is not short of. Supporting technologies such as TypeScript and

SASS (Syntactically Awesome Stylesheets) were chosen to shorten overall development time. TypeScript allows typing in JavaScript, which can help reduce the amount of runtime errors. As for SASS, styling of components is generally easier, with features such as variables, better calculations, and nested classes.

5. Implementation

5.1 Acquiring Data

Scraping the data was done using the WayBack Machine Scraper (WBMS) github software. Paired with that software a Python script was used to utilize the WBMS and collect/scrape only the data that we needed from the multiple sites of interest. This was done by specifying to only scrape the homepage of the desired domains and also specified the dates in which we are interested for scraping the sites. Only getting the latest snapshot of each month for each domain within the specified dates that are of interest.

5.2 Parsing the Data

In order to parse the scraped data into CSV files we used two Python v2.7.x scripts: parser.py and matrix_constructor.py. The two scripts are run from inside the */scraped_websites/ directory. We chose to have two separate files for modularity and readability.

First, parser.py recursively descends all website file paths and builds a list of all scraped websites and their corresponding snapshot files. Next, it iterates through that list file-by-file. In each file, it collects a list of external links and then labels that list of external links with the domain name of the website. The labeled list is stored in another list, this one organized by date. The purpose of this is to be able to query a certain year/month, and to receive a list of websites which had snapshots at that date, and those websites' corresponding external links. Then parser.py iterates through the list organized by date and creates an output file for each date entry, storing the contents of the entry in the output file.

Next, matrix_constructor.py constructs a list of output files and iterates through those files. For each file, it iterates through each line in order to construct a list of unique external links. It then creates a CSV file, writing the date as the first element. It then uses the list of unique external links to be the first row of the CSV, as they will be the column labels of the file. It then iterates through each line of the output file once again to fill in the rest of the CSV. The first entry of each line (after the first, which has already been constructed at this point) is the name of the website, and the subsequent entries are numbers. These numbers are the count of the number of occurrences of [external link] in [website] where the external links are the columns

and the websites are the rows. When matrix_constructor.py is finished running, there will be a CSV file for every output file created by parser.py.

scrapy_parser.py uses scrapy to gather data from the files. Typically scrapy is used for scraping online websites but we can use it to scrape local copies that we downloaded from the WayBack Machine. First, the parser loops through all the directories in the specified path and creates a list of URLs(or file paths) to scrape. Next we get the date of the snapshot from the file name, like 19990427132931.snapshot, where 1999 is the year, 04 is the month, and 27 is the day. Then we get the domain name of the snapshots we are currently scraping from the subdirectory. We then use a LinkExtractor from the scrapy framework to extract internal and external links. We then store all this information in a global variable “websites”. When our spider is done crawling all the files, the parser runs the closed(self, reason) function automatically and dumps all the data from our global variable to a JSON file.

6. Testing / Evaluation / Assessments

6.1 Testing

For the purposes of testing and debugging, the following files will be very helpful:

“console_out.txt”: This contains a list of logs stating when each snapshot file is about to be parsed. This will be helpful when trying to identify an error which occurs when parsing the snapshot files.

“list _of_snapshots.txt”: This contains a list of the snapshots formatted so that they can be easily traced to their respective webpages. This is helpful in identifying errors in file gathering.

“[yyyy][mm]_output.txt”: This contains a list for every website that had a snapshot on the given date. Each list contains the website as the first element, and all of the possible external links as the consecutive elements. This is helpful when debugging the code in matrix_constructor.py which determines if a link is a valid external link.

“unique_links.txt”: This file contains a list of every successfully verified and reduced link. This is helpful when debugging improper reductions as it is a quick reference you can use to identify weird links which indicate something went wrong. In addition, a line stating “Something went wrong” corresponds to a link being invalid and rejected by reduce_link(link) in matrix_constructor.py

“reductions.txt”: This file contains a list for every reduction. The first element of each list is the reduction itself, and the consecutive elements are all of the links which were reduced into the first element. This file is very useful for identifying flaws in external link verification, flaws in link reduction, and strange edge cases found in the data.

Console output: When you run matrix_constructor.py, it will output to console when it finds links which are rejected for one reason or another. An example line of output is:
39 item: 'http://205.232.252.85/iloveny_forms/ngttourquiz.asp' file: 200505_output.txt item index: ['www.iloveny.com', 1

The “*39*” indicates the index of the reductions_list which the code was considering to input this link. The “item: ‘*http://205.232.252.85/iloveny_forms/ngttourquiz.asp'* “ is the link which is being rejected. The “*file: 200505_output.txt*” is the output file where the link was pulled from. The “*item index: ['www.iloveny.com', 1*” is the website the link corresponds to in the given output file, and the link’s index in that list.

This console output is very helpful in identifying links which should have been properly accepted and reduced, but were for some reason rejected.

6.2 Evaluation

In addition to the matrices, Dr. Zach had also requested we create some data visualizations but after some setbacks he suggested we downsize the project. We had planned on hosting data visualizations on a website but without a budget we had to stop development on that, although a future team could continue developing it. Dr. Zach also suggested that instead of scraping entire websites we would only scrape the homepage and provide him with a dictionary structure to easily process and analyze. With these deliverables Dr. Zach will be able to analyze and observe trends in a smaller set of data. In the future he hopes to gather more data to write about in his publications.

7. User's Manual

7.1 WayBack Machine Scraper

7.1.1 Installing

Make sure the machine WBMS is being installed on has at least Python 3 on it. Also make sure pip works as well on the machine. Open a terminal and simply input:

```
pip install wayback-machine-scraper
```

7.1.2 Configure Python file

To use the Python script in order to scrape the websites you need to input the sites that you want to scrape into the "urls" array. Also include both the start date and end date to scrape the sites in with the format (month, year) in the "from_date" array and "to_date" array. Make sure the both the dates and desired sites are matching in the corresponding indexes of all 3 arrays otherwise it will not be scraped correctly. Also make sure all dates are spelled and in the correct format otherwise the code will crash.

7.1.2 Running

Once the Python file is properly configured simply type "Python scrap2.py" into the terminal (with the directory where the Python file is) to execute and scrape the inputted sites in the Python files and wait for the process to finish.

7.2 Parser

7.2.1 Installation

The two files used to parse the data were created using Python v2.7. In order to run them, this will need to be installed and these two files will need to be placed in the same directory as the directories with the website names (ex.: in the folder containing the folder named www.gohawaii.com, etc.).

scrapy_parser.py requires Python3 and the scrapy framework to run. The only modification required is for the user to change the path variable (`path = 'C:/Users/Aki/Documents/Projects/Python/scrapy/test'`) to wherever they have their snapshots.

7.2.2 Running

To run the two scripts and generate the CSV files, open a terminal and navigate to the directory containing the two scripts: parser.py and matrix_constructor.py. Next, run the following commands: ‘python parser.py’ and **after** that has finished, run ‘matrix_constructor.py’

scrapy_parser.py can be run with the command “scrapy runspider scrapy_parser.py”.This creates a JSON file in the same directory that can be used to further analyze the scraped data or create data visualizations.

The snapshots should be in the structure shown in Figure 2.

```
archive/
|__www.visitphilly.com/
|        |__19990427132931.snapshot
|        |__20190321134281.snapshot
|
|__www.arkansas.com/
|        |__20031103834091.snapshot
```

[Figure 2] Snapshot file structure.

7.2.3 Products

Running the commands above will create a dataset. The dataset is a set of CSV files of the following naming convention: “[yyyy][mm]_matrix.csv” where ‘yyyy’ is the year and ‘mm’ is the month of the data in the file. This data may be imported into any system or software which uses CSV files.

Running scrapy_parser.py produces a JSON file as shown in Figure 3.


```
{
   "1996": {
      "10": {
         "www.decd.state.ms.us": {
            "external": [],
            "internal": [
               "http://www.decd.state.ms.us/WHATSNW.HTM"
            ],
            "state": "MS"
         },
         "www.ded.state.ne.us": {
            "external": [
```

```
            "http://iwin.nws.noaa.gov/iwin/ne/ne.html",
            "http://www.rsac.org",
            "http://www.safesurf.com"
         ],
         "internal": [
            "http://www.ded.state.ne.us"
         ],
         "state": "NE"
      },
```

[Figure 3] scrapy_parser.py output.

This can be used to process the archived data easily and make data visualizations or analyses. For example, using Python the JSON file can be opened and loaded into a variable with:

```
with open('websites.json') as f:
   data = json.loads(f.read())
```

We can then access the nested dictionary with:

```
print(data["2008"]["02"]["www.arizonaguide.com"])
```

Here “2008”, “02”, and “www.arizonaguide.com” are keys within each nested dictionary. This would print out all information within the domain “www.arizonaguide.com” from February 2008 as shown in Figure 4.


```
{'external':
['http://server.iad.liveperson.net/hc/75169249/?cmd=file&file=visitorWantsToChat&site=75169249&byhref
=1', 'http://www.arizonascenicroads.com/', 'http://www.denvertoaz.com/DisplayArticle.aspx?id=8',
'http://www.azot.gov'],
'internal': ['http://www.arizonaguide.com/Section.aspx?sid=7',
'http://www.arizonaguide.com/DisplayArticle.aspx?id=30&refid=fy08feb1wa',
'http://www.arizonaguide.com/Section.aspx?sid=66&?refid=fy08azghazh'],
'state': 'AZ'}
```

[Figure 4] Example dictionary query output.

8. Developer's Manual

8.1 Front-End

This library was built with React, Yarn, and D3.js to provide a visual overview of how DMO websites have changed over time [8][9][10]. Available scripts can be viewed in the `package.json` file.

8.1.1 Library Information

The visualization library lives in `/visual/`. This part of the project uses React with TypeScript and SASS/SCSS. The purpose for using React is to have a single-page application that fetches data from our GraphQL API, which interfaces quite nicely with React when using dependencies such as react-apollo. TypeScript has been chosen for strict type checking to ensure that data fits the necessary type to be read in our D3.js visualizations. SASS/SCSS, or Syntactically Awesome Stylesheets, is used to keep a clean, consistent look with better support for variables, imports, and class nesting. As well, we chose Yarn to manage our dependencies due to linked dependencies which reduces the size of the `/node_modules/` directory as well as having a stable mirror for NPM. Lastly, the use of D3.js is to provide clean visualizations for our data, as well as the copious amounts of resources on the library.

8.1.2 Getting Started

If you have already followed the directions given in root/api, skip to Step 3.
After cloning the repository,

1. Install NodeJS LTS (v10.15.3 as of 2019-03-07)
2. Install Yarn Stable (v1.13.0 as of 2019-03-07)
3. In this directory `/visual/`, run `yarn install`. This will install all of the NodeJS package dependencies for this application to work.

And that should be it!

8.1.3 Running

The application can be run using `yarn start`, which will listen at `http://localhost:3000`.

8.1.4 Building

Our library can be built using `yarn build`. More documentation on this will be added soon.

8.2 API

This library was built with NodeJS, TypeScript, GraphQL, and MongoDB's NodeJS driver. The purpose of this library is to serve data from our backend using Azure Functions to utilize a serverless architecture.

8.2.1 Library Information

To get started with API development quickly, we segmented our code into 2 parts: GraphQL and Database API. For the GraphQL segment, we use Apollo GraphQL Server as a means to get up and running quickly with a visual GraphQL playground for testing. More detailed documentation will be added in future once the library has stabilized. The Database API segment is accessible only by function, so that only our GraphQL segment has access to the database. More detailed documentation will be added in future.

8.3 WayBack Machine Scraper

8.3.1 Configure Python file

If a full scrape of the sites is desired and not just the homepage of the domain then simply comment out or remove the "'wayback-machine-scraper -a ' + '\"'+ urls[i] + '$\" + " part of the line where "cmd" is being set. The cmd object is the string which is ran to collect the data. If any adjustments or configurations are desired upon the WBMS command then looking at the github page or WayBack Machine Scraper documentation would be the place to look.

8.3.2 Running

For more details and options on how to use or run the WBMS go to http://sangaline.com/post/wayback-machine-scraper/[3] or the github page: https://github.com/sangaline/wayback-machine-scraper[2]

8.4 Parser

The two files used to parse the data were created using Python v2.7. In order to run them, this will need to be installed and the two files will need to be located in the same directory as the directories with the website names. parser.py **must be run before** matrix_constructor.py.

In Sections 8.4.1 and 8.4.2, we will explain each part of the code in order to help future developers know which regions of code should be edited for their desired effect.

8.4.1 parser.py

The is_dir(f) and is_file(f) functions are for the purpose of identifying directories containing snapshots and identifying snapshot files respectively.

The recur(path) function is given the directory in which parser.py is in, and it will recursively build a list of websites and their snapshots.

The query_month_entry(month) function is for the purpose of testing if a given date is already stored in the monthMatrices list and if the month does exist, return its index.

The is_external_link(website_name, href_chunk) function is given the name of a website and a chunk of HTML code containing an href statement. The function will return true if the href_chunk is an external link. This function could be more robust, but it suits the needs of this current project.

The parse_snapshot(website_name, filename) function is responsible for building the monthMatrices list which will be used to construct the output files.

The code beneath parse_snapshot is responsible for creating the output files which are as follows:

list_of_snapshots.txt: this file contains the list of websites and their snapshots formatted for easy reading.

[yyyy][mm]_output.txt: these files contain a list of websites and their external links corresponding to the date of the file name.

console_out.txt: this file contains messages stating that “x.snapshot is about to be parsed.” This is for debugging purposes.

8.4.2 matrix_constructor.py

The reduce_link(link) function is responsible for cleaning up a long link into its ‘[subdomain].[domain].[Top-Level-Domain]’ [1]. This is for the purpose of easily identifying two links to the same website, even if they appear different.

The state_of_site(domain) function allows us to identify the state from which the given domain belongs to. This is for easy reference in the matrices.

The is_pop_tld(string) function determines if the given string is a popular top level domain, or if it is one used in one of the external links in the dataset. This was determined by gathering a list of top level domains from the “invalid links” output to console.

The find_first_non_alpha(string) function returns the index of the first non-alphabetical character.

The get_reduction_list_index(reduction) function is given a reduced website, and finds its place in the reduction_list. If the reduction is not found in reduction_list, it returns -1.

The next chunk of code iterates through the list of output files generated by parser.py (see above for format). It builds a list of unique external links to act as column labels. It then creates a CSV file, writing the external links to the first row. Then it iterates through each line of the current output file to create the rows. The row label is the website name and the [external_link][website] entries are the number of occurrences of that external link in that website.

matrix_constructor.py generates the following files:

[yyyy][mm]_matrix.csv: these files are CSV files containing that month's websites and corresponding external links.

unique_links.txt: this file contains every unique link which was verified to be a valid external link and then successfully reduced by the reduce_link(link) function.

reductions.txt: this file contains a list for every successfully reduced link. The reduced link is the first element of the list and all of the consecutive elements of the list are the un-reduced links which became reduced into the first element.

8.4.3 scrapy_parser.py

The purpose of this parser is to create a JSON file using the schema described previously in Section 4.2. We hope that this JSON file will be useful when building data visualizations. It is written using Python 3.7.0 and uses the scrapy framework to scrape the snapshots downloaded from the WayBack Machine. We have a global variable to store all the parsed data that is later dumped to the JSON file shown in Figure 5.

- Websites:
    - Year:
        - Month:
            - Domain:
                - Internal Links[]
                - External Links[]
                - State

[Figure 5] Simplified schema format.

When creating the spider class we have to populate the start_urls array with the URLs we will be scraping. In this case, our URLs are actually paths to the files we will be scraping such as "`file:///C:/Users/Example/Documents/snapshots/`".

The parse(self, response) function starts parsing our files. First, we get the date of the snapshot from the file name itself. After that we get the domain name from the name of the subdirectory. We then start extracting data from the file itself, such as internal and external links.

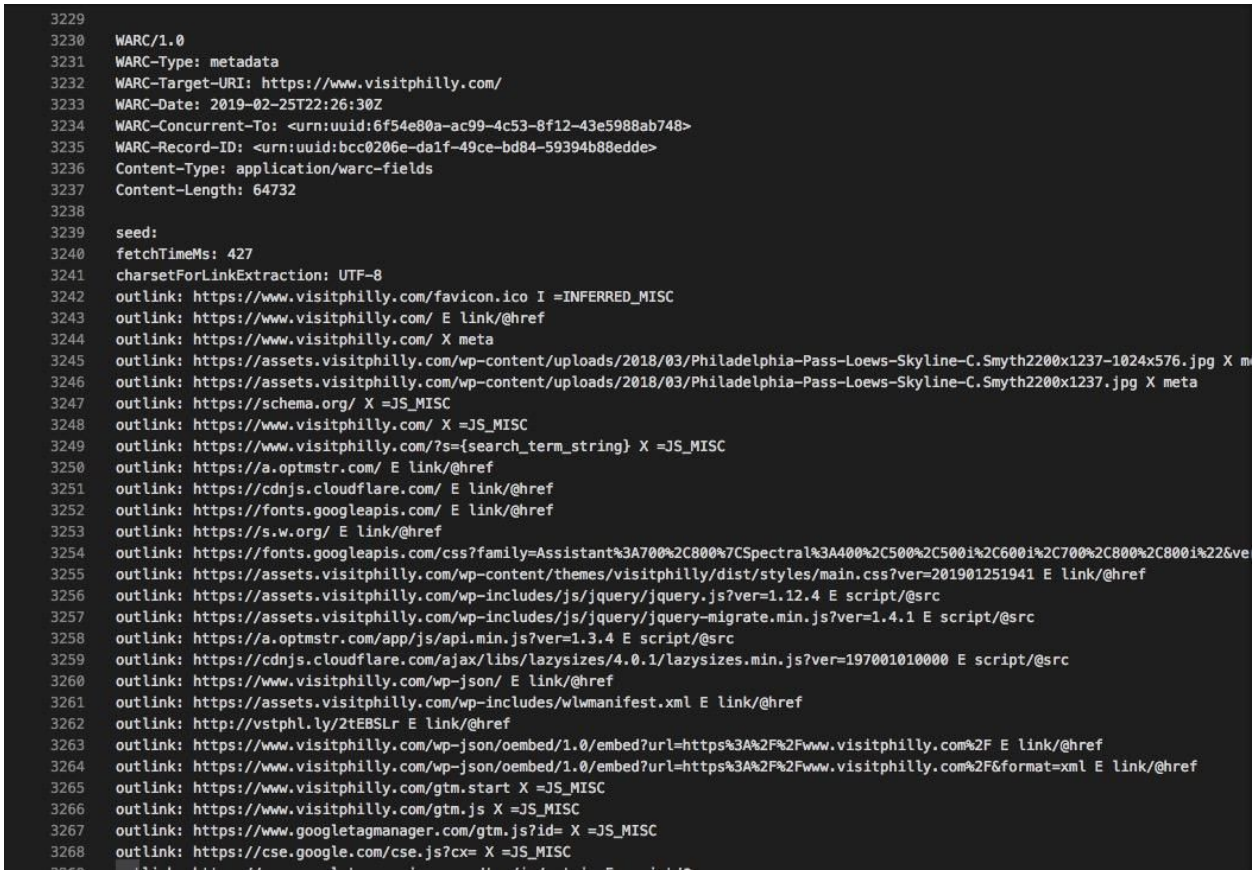
The closed(self, reason) function is automatically run after our parser is done scraping and it simply dumps all the scraped data to a JSON file.

More information on scrapy functionality can be found on their documentation page: https://docs.scrapy.org/en/latest/ [4].

9. Lessons Learned

9.1 Heritrix 3

Our initial attempt for the data collection part of our plan. Heritrix 3 [7] is an open source crawler that we looked into and attempted to use in order to crawl the required sites for the necessary information that we needed. Initially we saw Heritrix 3 and immediately thought that it would be a very good fit with its functions for the purposes with our project. It used the archive.org site to crawl and simply needed links to websites in order to know which one to crawl. It was also very well documented compared to alternatives which just made it more seem more organized and updated so we as a team would not run into problems when trying to use Heritrix. After installing and running Heritrix 3 on just one website link it took about 3 hours or more in order to fully crawl the inputted site and return information about the site. The amount of information that was returned from the crawl was extremely vast and complex to look through. Out of all the information that we sifted through we found 2 files of particular importance to us as those files seemed to contain the data that we would needed to parse for our project. One of the files that seemed important for us was a WARC file, shown in Figure 6.



[Figure 6] The WARC file with information about external links from www.visitphilly.com

We are still not too familiar with the WARC file format and all the information that it provides; but we did notice that the WARC file contained lines about outbound links of the site that we specified (in this case, www.visitphilly.com). Which is one of the main pieces of information that our project is to deliver to the client. The second file is a crawl log file, shown in Figure 7, which seemed more like a raw file that simply listed all links that were crawled throughout the entirety of the site.

[Figure 7] The crawl.log file with information about all the links crawled on www.visitphilly.com

We quickly ran into a problem, though as we began to slowly realize Heritrix 3 inherently as a tool does not have the ability to crawl through past snapshots of the inputted sites that are recorded on the archive.org. This was a big concern for us because the information received in the WARC file was extremely valuable and attractive to us as it gave us the information required to fulfill one of the project requirements. However Heritrix 3 seemed to only take the most recent snapshot of the inputted site on archive and crawl on only that version of the site and then output the information for the recent version only. This is an issue as crawling for information about past snapshots of the sites is also a requirement for the project. So after discussing it with the team and looking into alternative options, and noting that Heritrix seemed to not get all the information that we wanted, we decided to move on and try a different tool called WayBack Machine Scraper.

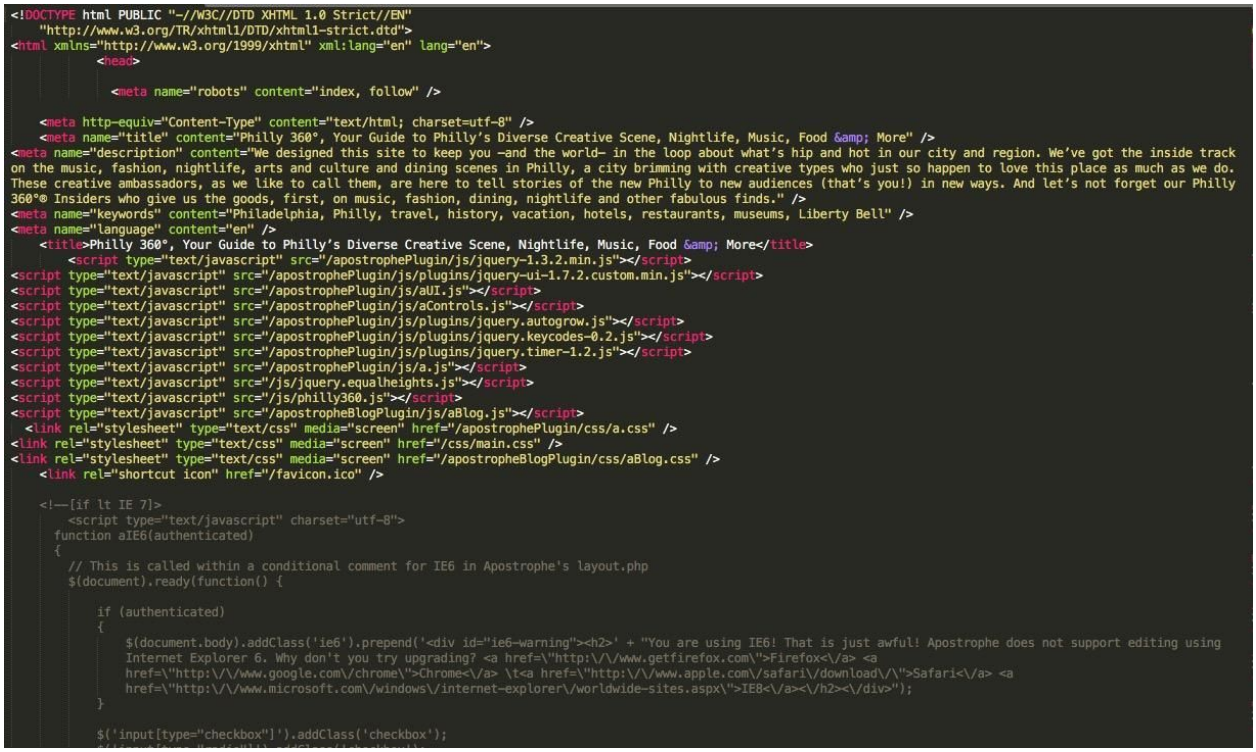
9.2 WayBack Machine Scraper (Mac Version)

The WayBack Machine Scraper is similar to Heritrix 3. It is an open source tool made for crawling through old WWW sites by looking at them through archived snapshots on archive.org. However, the WayBack Machine Scraper is nowhere near as well documented as Heritrix 3. Therefore it took an extended amount of time to get the

tool to work on our machines. This time we made sure that the tool crawled and scraped through past snapshots first before moving forward with using the tool. However I ran into much difficulty installing the tool onto my macbook. By default my MacBook has Python 2.7.10 installed. However, the scraper required Python 3 for various packages and other dependencies in order to run the tool. After much difficulty, we were able to get it to work on the MacBook and were able to scrape www.visitphilly.com similarly to how we did with Heritrix 3. The names of the snapshots listed the dates in which the snapshots were taken and organized those snapshots folder by folder named after these dates as shown in Figure 8.

[Figure 8] Folder of a domain crawled containing snapshot files named after the time and date the snapshot was taken.

The information output and shown in the snapshot folder was the HTML code for the visitphilly site as shown in Figure 9.



[Figure 9] The HTML code for a certain snapshot of visitphilly.com recorded at some time on archive.org

The information was ultimately what we needed, but it was not as well organized as the Heritrix 3 output as we need to parse through the HTML code to find outlinks. On Heritrix 3, that information was already given to us and we just needed to spend more time reading the format of the file to get the information that we need.

9.3 WayBack Machine Scraper (Windows Version)

In addition to being less documented than Heritrix3, WayBack Machine Scraper has poor Windows 10 support. we used `pip install wayback-machine-scraper` to install it with no errors. However, when attempting to run a test crawl with the command: `wayback-machine-scraper -f 20080623 -t 20080623 news.ycombinator.com` , the console returned dependency errors indicating an incorrect version of Python 3.x. I then tried the same process with Python 2.x, which succeeded. However, trying the test scrape again, more errors occurred indicating an incorrect version of Visual C++. After downloading and installing the most recent version of Visual C++, the dependency error persisted. This persisted even though older versions of Visual C++ and older known stable versions of Python 2.x and 3.x.

Further attempts to get WayBack Machine Scraper to work on Windows 10 were successful, so it is possible that there is negative interaction with hardware components of certain machines.

9.4 Scrapy WayBack Machine Middleware

We attempted to use a middleware designed for scraping snapshots of webpages from archive.org. The WayBack Machine Scraper command-line tool mentioned previously actually uses this middleware to crawl through snapshots. Unfortunately, it only worked on Fedora and Debian workstations as any attempt to run it on Windows resulted in the same error “OSError: Bad Path”. While the middleware was able to crawl through the snapshots, there were a few issues with scraping using this method. The snapshots are located within the archive.org website so there is additional data, such as extra links and images, we have to account for. We also have two time constraints that make scraping more troublesome. We want to only get the latest snapshot from every month but the frequency of snapshots can vary greatly. These issues can likely be resolved but makes scraping more difficult than expected.

9.5 WARC Files

In the early stages of working on this project, we stumbled upon WARC files by using Heritrix 3 as an attempt to collect our data. After working on the project learned that Virginia Tech has connections with the archive.org site and we would be able to request information about certain sites and their previous versions. However, this was very late into the project (about the halfway point) and there were details that needed to be worked out before we could officially be in queue to retrieve the data we needed. All of these things ultimately meant we would not be able to get the detailed WARC files of the desired domains for our client in time.

9.6 Pamplin IT

Before pivoting to just collecting data, we wanted to host our data in a database that would be hit by our hosted GraphQL and Node.js Function API, and then the data would be presented through visualization logic using React and D3. We explored a few options, with our client referring us to Pamplin IT to find the best way to host our application. After talking with Jim Dickhans, the Director of Pamplin IT, Jim believed it would be best for us to deploy our application to the Pamplin Microsoft Azure account, allowing us as many resources as needed to host the application. Initially, the deployment went smoothly (or so we thought). However, obfuscated billing information along with no defined DevOps process caused a couple of unnoticed issues.

The biggest issue was the deployment of the Azure Cosmos DB to hold our data, as the database was provisioned to have the highest throughput possible on the database. Due to not being able to see the billing on any side (client, group, or Pamplin IT), the database quickly incurred large amounts of charges. After nearly a month of the database being provisioned (yet without data), the Virginia Tech Accounting Department contacted Jim Dickhans with the billing statement, citing a large spike in usage. The group was instructed to remove all resources from Azure as quickly as possible and instructed to halt all progress on the database, API, and front-end.

After meeting with Dr. Fox, we decided that the best course of action was to work on gathering data so another group could work on the other facets of the application later. If we had at least one of three things: a DevOps process, viewable billing statement, or faster notice from the accounting department, we believe this issue would not have happened and our scope would have stayed along the same lines of visualizations. However, our pivot was necessary to achieve the client's new goals post-incident.

10. Plans and Future Work

What our group has accomplished is to create a skeleton for data collection, parsing, and transformation into a viewable format. In addition, we have created a framework for a web application. An obvious direction for future work is to pivot towards the data gathering aspects of the project and to work on enlarging the data set. This will involve work on the web scraping code as well as modification of the parsers. In the future, groups would continue work on the front-end, the relevant elements that use D3 and the API referenced in Section 8.2. The data will be used to ensure the front-end logic is correct and follows the schema.

After this semester, the project will be increased in scope to include other countries and cities within the USA. This will allow future users to see the trends of tourism websites on a smaller scale (i.e. in a state or region) as well as a broad scale, such as which areas are more developed in terms of DMO websites. As well, budget data is to be gathered to see how budget affects website changes.

On a smaller scale, the parsing scripts need better comments and some refactoring for improved readability for future developers. In addition, it would be nice for the scripts to output their files into a subdirectory rather than into the same directory as the scripts themselves.

11. Acknowledgements

Florian Zach, PhD, <florian@vt.edu>, Assistant Professor, Howard Feiertag Department of Hospitality and Tourism Management, Pamplin College of Business, Virginia Tech, Wallace Hall 362, Blacksburg VA 24061 USA

Zheng (Phil) Xiang, PhD, <philxz@vt.edu>, Associate Professor, Howard Feiertag Department of Hospitality and Tourism Management, Pamplin College of Business, Virginia Tech, Wallace Hall 353, Blacksburg VA 24061 USA

Jim Dickhans, <jdickhans@vt.edu>, Director of IT, Pamplin Office of Information Technology, Pamplin College of Business, Virginia Tech, Pamplin Hall 2004 B, Blacksburg VA 24061 USA

Edward A. Fox, <fox@vt.edu>, Professor, Department of Computer Science, Virginia Tech, 114 McBryde Hall, Blacksburg VA 24061 USA

12. References

[1] Moz, “URL Structure | SEO Best Practices,” *Moz*, 03-Apr-2019. [Online]. Available: https://moz.com/learn/seo/url. [Accessed: 26-Apr-2019].
[2] Sangaline, Evan. “Internet Archaeology: Scraping Time Series Data from Archive.org.” *Sangaline.com*, 5 Apr. 2017, Available: sangaline.com/post/wayback-machine-scraper/. [Accessed: 26-Apr-2019]
[3] Sangaline. “Sangaline/Wayback-Machine-Scraper.” *GitHub*, 6 Apr. 2017, Available: github.com/sangaline/wayback-machine-scraper. [Accessed: 26-Apr-2019]
[4] Scrapinghub Ltd., “Scrapy.” *scrapy.org* 26 June. 2018, Available: https://docs.scrapy.org/en/latest/. [Accessed: 15-Apr-2019]
[5] “Top Collections at the Archive,” Internet Archive: Digital Library of Free & Borrowable Books, Movies, Music & Wayback Machine. [Online]. Available: https://archive.org/. [Accessed: 17-May-2019].
[6] “GraphQL: A query language for APIs.,” *A query language for your API*. [Online]. Available: https://graphql.org/. [Accessed: 17-May-2019].
[7] Internet Archive, “internetarchive/heritrix3,” *GitHub*, 18-Apr-2019. [Online]. Available: https://github.com/internetarchive/heritrix3. [Accessed: 17-May-2019].
[8] React, “React – A JavaScript library for building user interfaces,” – *A JavaScript library for building user interfaces*. [Online]. Available: https://reactjs.org/. [Accessed: 17-May-2019].
[9] Yarn, “Fast, reliable, and secure dependency management.,” *Yarn*. [Online]. Available: https://yarnpkg.com/en/. [Accessed: 17-May-2019].
[10] M. Bostock, “Data-Driven Documents,” *D3.js*. [Online]. Available: https://d3js.org/. [Accessed: 17-May-2019].
[11] “Welcome to Python.org,” *Python.org*. [Online]. Available: https://www.python.org/. [Accessed: 17-May-2019].

13. Appendices

13.1 Appendix A: Project Repository

Link to the project repository: https://code.vt.edu/jsalopek/tourism-websites

13.2 Appendix B: Project Artifacts

Table 1: List of Websites Scraped

STATE	URL	FROM	TO
Alabama1	www.state.al.us/ala_tours/welcome.html	June 1997	December 1998
Alabama2	www.touralabama.org	January 1998	June 2006
Alabama3	www.alabama.travel	July 2006	March 2019
Alaska	www.travelalaska.com	December 1998	March 2019
Arizona1	www.arizonaguide.com	December 1996	July 2014
Arizona2	www.visitarizona.com	August 2014	March 2019
Arkansas2	www.arkansas.com	January 1999	March 2019
California1	www.gocalif.ca.gov	December 1996	February 2001
California2	www.visitcalifornia.com	March 2001	March 2019
Colorado	www.colorado.com	November 1996	March 2019
Connecticut1	www.tourism.state.ct.us	February 2000	April 2005
Connecticut2	www.ctvisit.com	May 2005	March 2019
Delaware1	www.state.de.us/tourism	October 1999	July 2000
Delaware2	www.visitdelaware.net	August 2000	April 2002
Delaware3	www.visitdelaware.com	May 2002	March 2019
District of Columbia	www.washington.org	December 1996	March 2019
Florida1	www.flausa.com	December 1998	July 2004
Florida2	www.visitflorida.com	August 2004	March 2019
Georgia1	www.georgia-on-my-mind.org/code/tour.html	May 1997	May 1997
Georgia2	www.georgia.org/tourism	May 2001	September 2005
Georgia3	www.georgia.org/travel	October 2005	January 2008
Georgia4	www.exploregeorgia.org	February 2008	March 2019
Hawaii1	www.visit.hawaii.org	November 1996	April 1997
Hawaii2	www.gohawaii.com	May 1997	March 2019

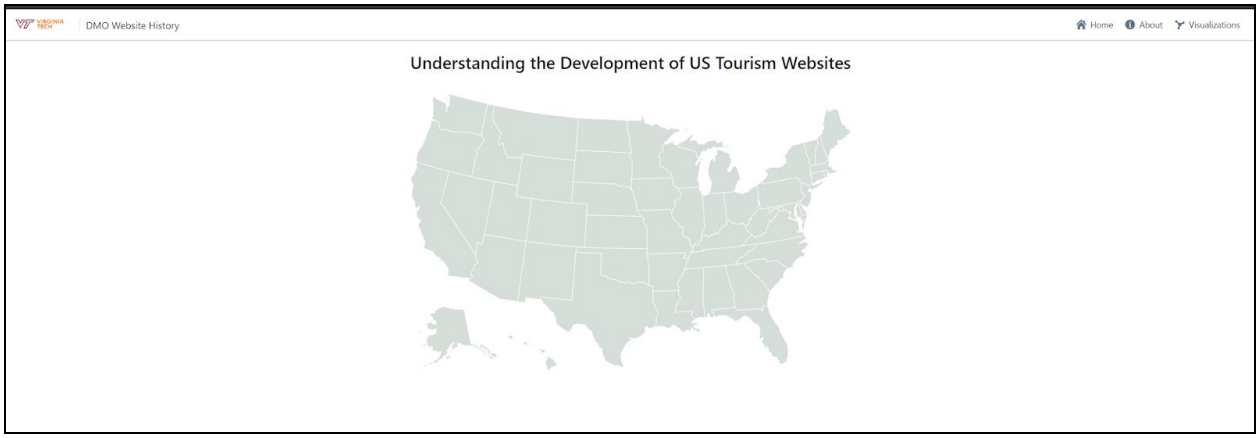
Idaho1	www.visitid.org	January 1997	March 2000
Idaho2	www.visitidaho.org	April 2000	March 2019
Illinois	www.enjoyillinois.com	November 1996	March 2019
Indiana1	www.state.in.us/tourism	May 1999	March 2000
Indiana2	www.enjoyindiana.com	April 2000	March 2019
Iowa1	www.state.ia.us/tourism/index.html	November 1996	November 1999
Iowa2	www.traveliowa.com	December 1999	March 2019
Kansas1	kansascommerce.com/0400travel.html	January 1998	January 2001
Kansas2	www.travelks.com	February 2001	March 2019
Kentucky1	www.state.ky.us/tour/tour.htm	June 1997	January 2000
Kentucky3	www.kentuckytourism.com	December 1998	March 2019
Louisiana	www.louisianatravel.com	April 1997	March 2019
Maine	www.visitmaine.com	December 1996	March 2019
Maryland1	www.mdisfun.org	October 1996	April 2004
Maryland2	www.visitmaryland.org	May 2004	March 2019
Massachusetts1	www.mass-vacation.com	December 1996	October 1998
Massachusetts2	www.massvacation.com	November 1998	March 2019
Michigan	www.michigan.org	December 1997	March 2019
Minnesota	www.exploreminnesota.com	January 1998	March 2019
Mississippi1	www.decd.state.ms.us/TOURISM.HTM	October 1996	October 1999
Mississippi2	www.visitmississippi.org	November 1999	March 2019
Missouri1	www.ecodev.state.mo.us/tourism	December 1996	December 1997
Missouri2	www.missouritourism.org	January 1998	March 2001
Missouri3	www.visitmo.com	April 2001	March 2019
Montana1	www.travel.mt.gov	October 1996	February 2000
Montana3	www.visitmt.com	February 2000	March 2019
Nebraska1	www.ded.state.ne.us/tourism.html	October 1996	November 1998
Nebraska2	www.visitnebraska.org	December 1998	November 2008
Nebraska3	www.visitnebraska.gov	December 2008	August 2012
Nebraska4	www.visitnebraska.com	September 2012	March 2019
Nevada	www.travelnevada.com	December 1996	March 2019
New Hampshire	www.visitnh.gov	December 1996	March 2019
New Jersey1	www.state.nj.us/travel/index.html	June 1997	March 1999
New Jersey2	www.visitnj.org	April 1999	March 2019
New Mexico	www.newmexico.org	December 1996	March 2019
New York1	www.iloveny.state.ny.us	November 1996	July 2000
New York2	www.iloveny.com	August 2000	March 2019
North Carolina	www.visitnc.com	November 1998	March 2019

North Dakota2	www.ndtourism.com	November 1998	March 2019
Ohio1	www.travel.state.oh.us	December 1996	November 1998
Ohio2	www.ohiotourism.com	December 1998	April 2003
Ohio3	www.discoverohio.com	May 2003	February 2016
Ohio4	www.ohio.org	February 2016	March 2019
Oklahoma	www.travelok.com	December 1998	March 2019
Oregon	www.traveloregon.com	June 1997	March 2019
Pennsylvania1	www.state.pa.us/visit	December 1998	December 1998
Pennsylvania3	www.experiencepa.com	January 1999	June 2003
Pennsylvania4	www.visitpa.com	July 2003	March 2019
Rhode Island	www.visitrhodeisland.com	December 1997	March 2019
South Carolina3	www.travelsc.com	June 1997	May 2001
South Carolina4	www.discoversouthcarolina.com	November 2001	March 2019
South Dakota2	www.state.sd.us/tourism	May 1997	November 1998
South Dakota3	www.travelsd.com	December 1998	March 2015
South Dakota4	www.travelsouthdakota.com	April 2015	March 2019
Tennessee1	www.state.tn.us/tourdev	July 1997	February 2000
Tennessee3	www.tnvacation.com	March 2000	March 2019
Texas	www.traveltex.com	December 1996	March 2019
Utah1	www.utah.com	November 1996	June 2006
Utah2	www.utah.travel	July 2006	December 2011
Utah3	www.visitutah.com	January 2012	March 2019
Vermont1	www.travel-vermont.com	December 1996	March 2002
Vermont2	www.vermontvacation.com	April 2002	March 2019
Virginia	www.virginia.org	October 1997	March 2019
Washington1	www.tourism.wa.gov	December 1996	March 2000
Washington2	www.experiencewashington.com	April 2000	February 2004
Washington3	www.experiencewa.com	March 2004	March 2019
West Virginia1	www.state.wv.us/tourism	January 1997	November 1998
West Virginia2	www.callwva.com	December 1998	July 2004
West Virginia3	www.wvtourism.com	August 2004	March 2019
Wisconsin1	tourism.state.wi.us	December 1996	April 1999
Wisconsin2	www.travelwisconsin.com	May 1999	March 2019
Wyoming3	www.wyomingtourism.org	February 1999	February 2016
Wyoming4	www.travelwyoming.com	March 2016	March 2019

Table 2: Example Output: 199610_matrix.csv

199610	State	iwin.nws.noaa.gov	rsac.org	safesurf.com	tc.net
www.decd.state.ms.us/TOURISM.HTM	Mississippi	0	0	0	0
www.ded.state.ne.us/tourism.html	Nebraska	1	1	1	0
www.mdisfun.org	Maryland	0	0	0	1
www.travel.mt.gov	Montana	0	0	0	0

[Figure 10] A snapshot of the API playground.



[Figure 11] The landing page of the website after successful deployment.