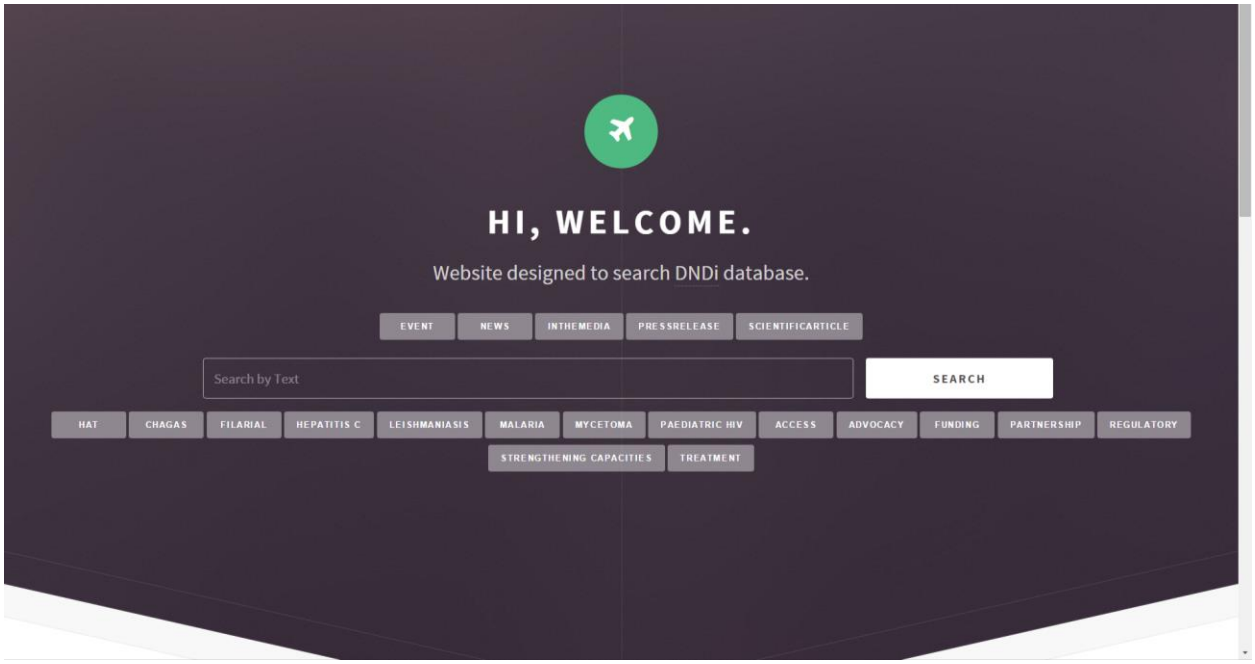
CS300 Final Project
December 18, 2015
Derek Dik, Peter Kim

Overview

The Drugs for Neglected Diseases initiative (DNDi) hosts many useful articles related to rare and neglected disease research. DNDi could be a potential resource to cross-reference with PubMed or other medical and genetic databases to try to find diseases with funding that have not had significant research yet. However, it doesn't offer any database tools to help its users characterize disease research. For our project, we wrote a web application to analyze the articles hosted on DNDi and display useful statistics. In particular, we wanted to know which diseases have the most articles written about them and in which years those diseases have been studied in order to determine which diseases may be most viable for a new study.

DNDi hosts several categories of articles. We determined that News articles, Press Releases, Scientific Articles, In the Media, and Events would be relevant to determine which diseases have the most or least activity.



We decided to create a web application because it is intuitive and familiar to all audience. Furthermore, a web application has the advantage of being cross-platform; anyone with access to the internet can use the tool. Lastly, a web application is ideal to display visualizations of data such as graphs because there are countless templates, modules, and examples online, which can drastically reduce working hours.

The web application provides a very simple interface to search for statistics —— regarding any of the tags used on articles on DNDi. On the web page, a user may select categories of articles and tags provided by DNDi to filter out results. There is also a search bar with which the user can enter a keyword to search by. If no keyword is provided, all articles that match the given filters will be used. It displays graphs of the number of articles for each selected tag, and the number of articles per year with the selected tag. No other input is required.

There is a button at the bottom of the page to view all of the articles used in the graph. It will take the user to a page that displays the name of the article, the first one hundred characters of the body, and has a hyperlink to the article in question.

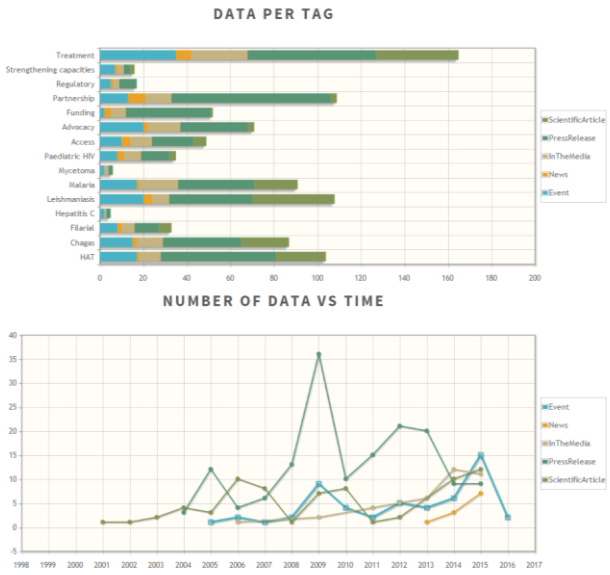
Tools

We began with a pre-made web template refactored into the Laravel PHP framework. HTML5UP provided the template that we used. The application uses a Python 3 script to scan DNDi. We used the BeautifulSoup library to scrape all of the article items directly from the article pages on DNDi. In order to get the HTML documents from the web, we used Python's built-in urllib.

We decided to use a MySQL database to cache all of the data for the application. Luckily, DNDi database was a relatively small database with number of articles being only around a thousand. Hence, it was possible to scrape all of the data at one. One

other reason why we chose MySQL is the handy tools which MySQL provides for manipulating the data we captured: we can sort, search through full text, and rely on a consistent format when we characterize our data.

The website displays graphs of the data the database using JqPlot, a Javascript graphing module that uses jQuery. There are two main types of graphs we draw using JqPlot. First is a bar graph which shows number of articles stored on MySQL database for each tags, which are keywords associated with each articles. Second is a line graph that maps the number of articles against year, which demonstrates the attention paid to each diseases or subjects at certain point of time.



Challenges

There are some time constraints involved with the web scraper. The information we need to scrape is stored across many pages, and thus, our web scrap is considerably slow. The scraper takes roughly one second to process a single article. On its own, that is not a significant amount of time. However, because each search executed by the application incorporates hundreds of articles, it adds up to an unfeasible wait time. For this reason, we implemented the MySQL database. The script can cache the data from all articles into the database in about twenty six minutes. We

plan to compensate for the time constraint by running the python script from the web service at the beginning of every day.

The HTML that DNDi is comprised of was not intended to be read the way our scraper operates. Oftentimes a certain form element on one page can be located in a different spot than on another. Some pages had no tags, others had no date visible aside from the year. Other pages had the date displayed correctly but the month spelled incorrectly; we found multiple pages that listed the month of March as “Match”. We had to accommodate all of these edge cases while writing the script. The relevant data we scraped included the date of each article, its category, and any tags it was marked with. Our web scraper outputted this information into a MySQL database with a separate table for each type of article. In spite of the difficulties implementing a script to scrape data, the application is fully functional and performs as specified.

Github

https://github.com/rladudrl602/DNDiGraph/

References

BeautifulSoup - http://www.crummy.com/software/BeautifulSoup/

HTML5UP - http://html5up.net/

jqplot - http://www.jqplot.com/

Laravel - http://laravel.com/docs/5.1

MySQL - https://dev.mysql.com/doc/

Python - https://www.python.org/