

# **Navigating Articles of News Websites With a Web Metrics Visualization**

A Thesis in the Field of Information Technology  
In Partial Fulfillment of the Requirements  
For a Master of Liberal Arts Degree

Harvard University

Extension School

May 28, 2014

**Alain Ibrahim**

Thesis Director: Alexander Lex

# Contents

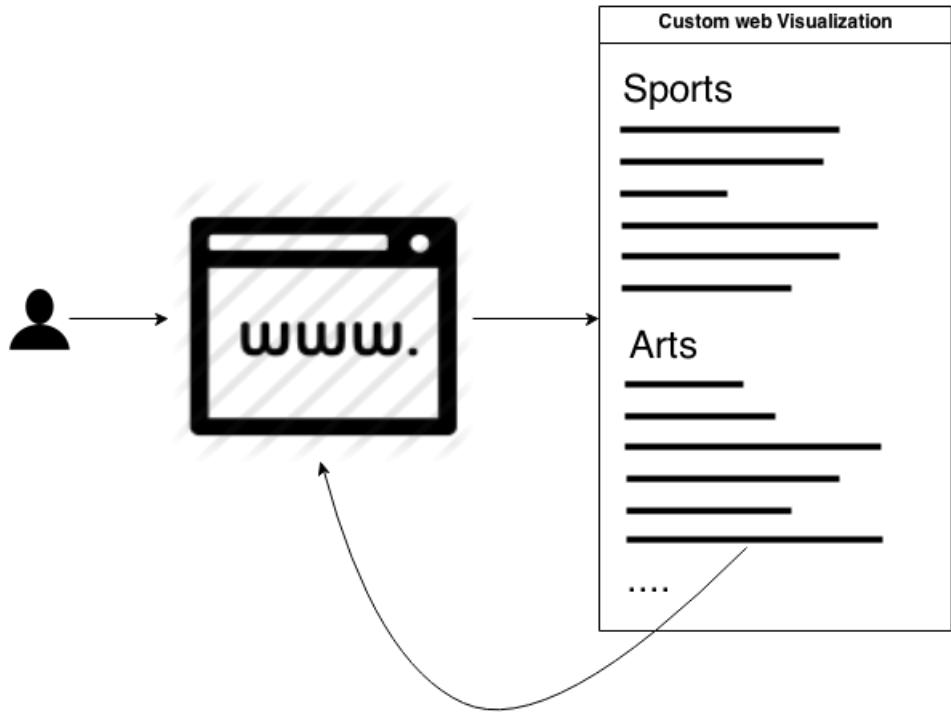
<b>1 Abstract</b>	<b>3</b>
<b>2 Thesis Project Description</b>	<b>5</b>
<b>3 Prior Work</b>	<b>12</b>
3.1 Visualization . . . . .	12
3.2 Existing web metrics systems . . . . .	13
<b>4 Work Plan</b>	<b>19</b>
4.1 Assumptions, Risks and Alternatives . . . . .	19
4.2 Preliminary Schedule . . . . .	19
<b>5 Implementation - data capture</b>	<b>21</b>
5.1 Capturing Data . . . . .	21
5.1.1 IP Addresses . . . . .	21
5.1.2 Geographic origins . . . . .	22
5.1.3 Average Read Time . . . . .	24
5.2 Database Design . . . . .	26
5.2.1 Data Storage . . . . .	26
5.2.2 Database Tables . . . . .	27
5.2.3 MySQL Database . . . . .	28
5.3 PHP server-side code . . . . .	30
5.3.1 Configuration and database connection . . . . .	30
5.3.2 Server-side hit, article, and category capture . . . . .	31
5.4 Client-side read time capture with Javascript . . . . .	35
5.5 Tying it all together . . . . .	38
<b>6 Implementation - hooking into Wordpress</b>	<b>40</b>
6.1 Creating the captureLib folder . . . . .	40
6.2 Creating a mock news website . . . . .	45
6.3 Testing . . . . .	46
6.3.1 Proof of concept . . . . .	46
6.3.2 Seeding mock data . . . . .	47
<b>7 Implementation - retrieving captured metrics</b>	<b>50</b>

<b>8 Implementation - visualization</b>	<b>53</b>
8.1 The controls box . . . . .	53
8.2 Stacked area chart . . . . .	58
8.3 Geographic bar charts . . . . .	64
<b>9 Results</b>	<b>66</b>
9.1 Mock Website . . . . .	66
9.1.1 Capturing Data . . . . .	67
9.1.2 Retrieving Data . . . . .	67
9.1.3 The Interactive Visualization . . . . .	67
9.1.4 Performance . . . . .	68
9.2 Actual Website . . . . .	69
<b>10 Future Work</b>	<b>69</b>
<b>11 Glossary</b>	<b>69</b>

# 1 Abstract

Current prominent web traffic analytics systems such as Google Analytics report traffic per website, given certain filtering criteria. Such systems are typically geared to system administrators who use the data to identify patterns and possibly assist management in strategy making. For news sites, this may mean analyzing the traffic on a news article and where most of the readers came from. The end users, however, will at best see metrics such as social likes and shares. But, what if we could provide the end users a near real-time metrics map of the entire news site they are visiting? What if they could navigate based on traffic patterns, as opposed to having to follow the common front page layout dictated by the news editors? This system is what I plan to build - a web-based, interactive visualization that shows the near real-time metrics of a select news site. First off, I plan to build a server-side library that can be included into any website or web application. This library would hook into a relational database and would efficiently capture the hits (possibly inclusive of IP addresses, geo locations, etc.) against the given news site. My second step involves building the "business logic" of the data and information that I want to capture. This includes algorithms that produce the refined data and metrics to be ultimately communicated to the end user. My final step is to build a central interactive visualization that shows near real-time user web traffic in each of the registered articles of the given news site. In the proposed scenario, the user will:

- See an overview map of the news site's articles.
- Filter traffic based on available data, such as the user's geographic location.
- See the average read time of a select article. The overall goal is to have a more personal browsing experience in the midst of all the "guided" text and hyperlinks.
- Navigate to the article of interest by clicking on its visual representation.
- Obtain details on demand for a given news article by hovering over its visual representation.



**Figure 1 –** The user visits a news site and loads an interactive web visualization which visualizes web traffic and attention span across the visitors' origin geographic areas. The visualization also includes previews of the articles and a way to link back to the articles' pages.

## 2 Thesis Project Description

The system that I built hooks into a Wordpress news website, captures web metrics, and visualizes these metrics in an interactive web-based visualization. The visualization is targeted towards both webmasters and end readers.

On the server side scripting end, I built PHP classes that were included in the top of the news site's file(s). The code registered every visiting IP address. A hit/visit was defined as a unique IP address hitting the news site in near real-time, thereby countering against bots and maximizing the accuracy of the readings. For example, if someone programmed a script that automatically refreshed the articles every few seconds to inflate the hits readings, that attempt would be ineffective. This is due to a mechanism that I created which allows the administrator to set a threshold, in seconds, as to when the next visit should be counted as a new hit. For example, if this variable was set to 60 seconds, and the script refreshed the article page as many times as possible within 60 seconds, then all the page refreshes would be collectively counted as 1 hit.

I built the relational database using normalization techniques and principles.

Here are the technologies that I used:

- MySQL relational database
- PHP for server-side scripting
- HTML 5, CSS3, and JavaScript for client-side design and implementation
- D3.js to facilitate in the visualization of the data
- jQuery to shorthand the lengthy Javascript functions

For this project, I captured data as far back as a month. To facilitate the visualization creation process, I used the D3.js (Data-Driven Documents) Javascript library. On the markup and structural side, I used HTML and SVG. The domain at hand is that of **online news media**. As for the interactive visualization, the design enabled 4 tasks:

- To compare articles' hits and reader attention spans for any period within the last month, given a select category

- To preview each article's thumbnail, title, and excerpt
- To contrast the geographic distribution of hits and attention of up to 2 articles
- To navigate to the full article counterpart on the subject news website

In terms of interactivity, the visualization enabled the following:

- Hovering over article elements in order to see previews of the articles (main image, text description, and some of the body's text)
- Clicking on an article's visual encoding to display a table that shows the geographic distribution of the number of hits and attention span for that article
- A controls box that allows the user to select a desired news category, to filter by date range, and to toggle some context-related settings

As for web traffic data being captured, here is a list of what I captured

- Visitor hits to a given article. Each hit is defined by a unique timestamp and IP address
- Geographical origin of the visitors to a given article
- Amount of time spent reading the article (for any given hit)

The final product was geared towards the conventional news site structure, where categories are enumerated (e.g. sports, health, science, politics, etc.), and where articles are listed under their pertinent categories. The below illustrate the initial sketches that were used to shape the final product.

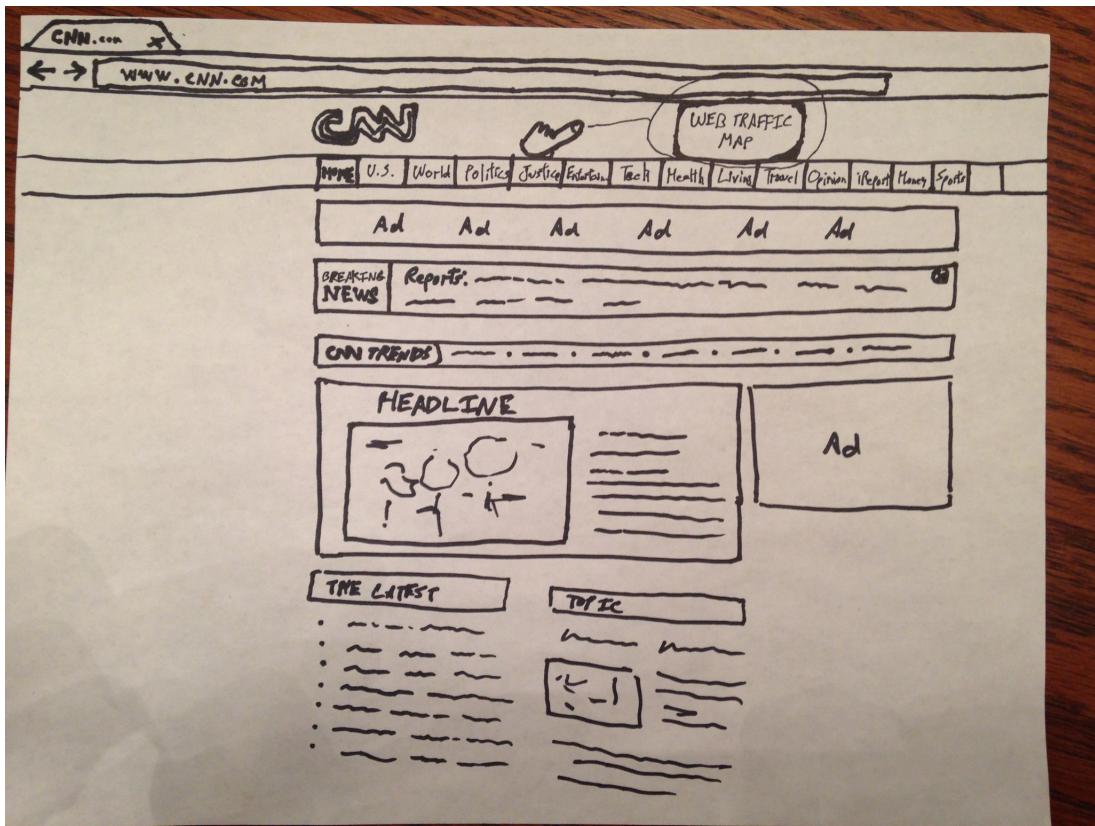


Figure 2 – On a news site like CNN, for example, there would reside a link to the near real-time visualization.



Figure 3 – A potential way of visualizing the near real-time traffic. Here, the articles are encoded by lines. Each line's length signifies how much traffic a select article has. Hovering over said line/article invokes a details-on-demand window - showing the title, a picture snapshot, and perhaps some content from the article's body.

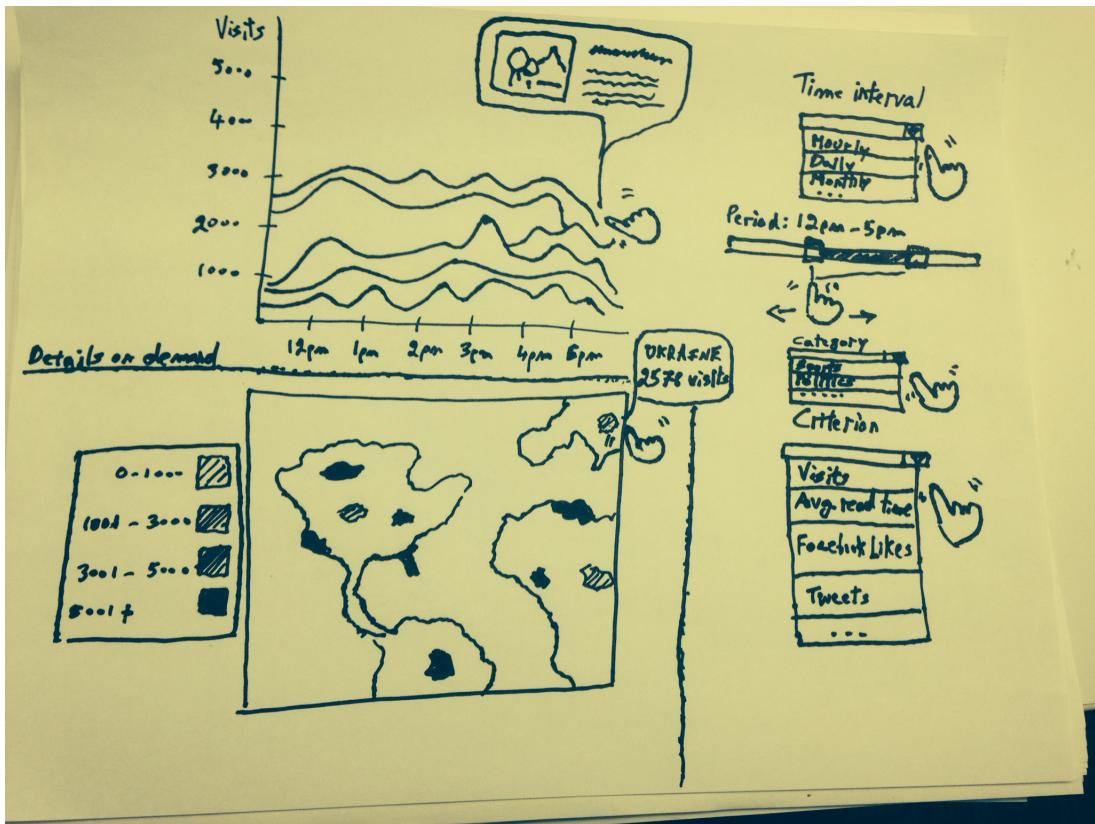


Figure 4 – A conception of what the final artifact was to look like. On the right side reside the controls that allow the user to filter by time interval (daily, monthly, etc.), category (sports, business, politics, etc.), and the output criterion to measure (number of visits, average read time per article, facebook likes, etc.). The stacked area chart supports hover (details on demand) and click (to navigate to select article). Upon hovering over said article, the map was to highlight the geographic origins of the visitors.

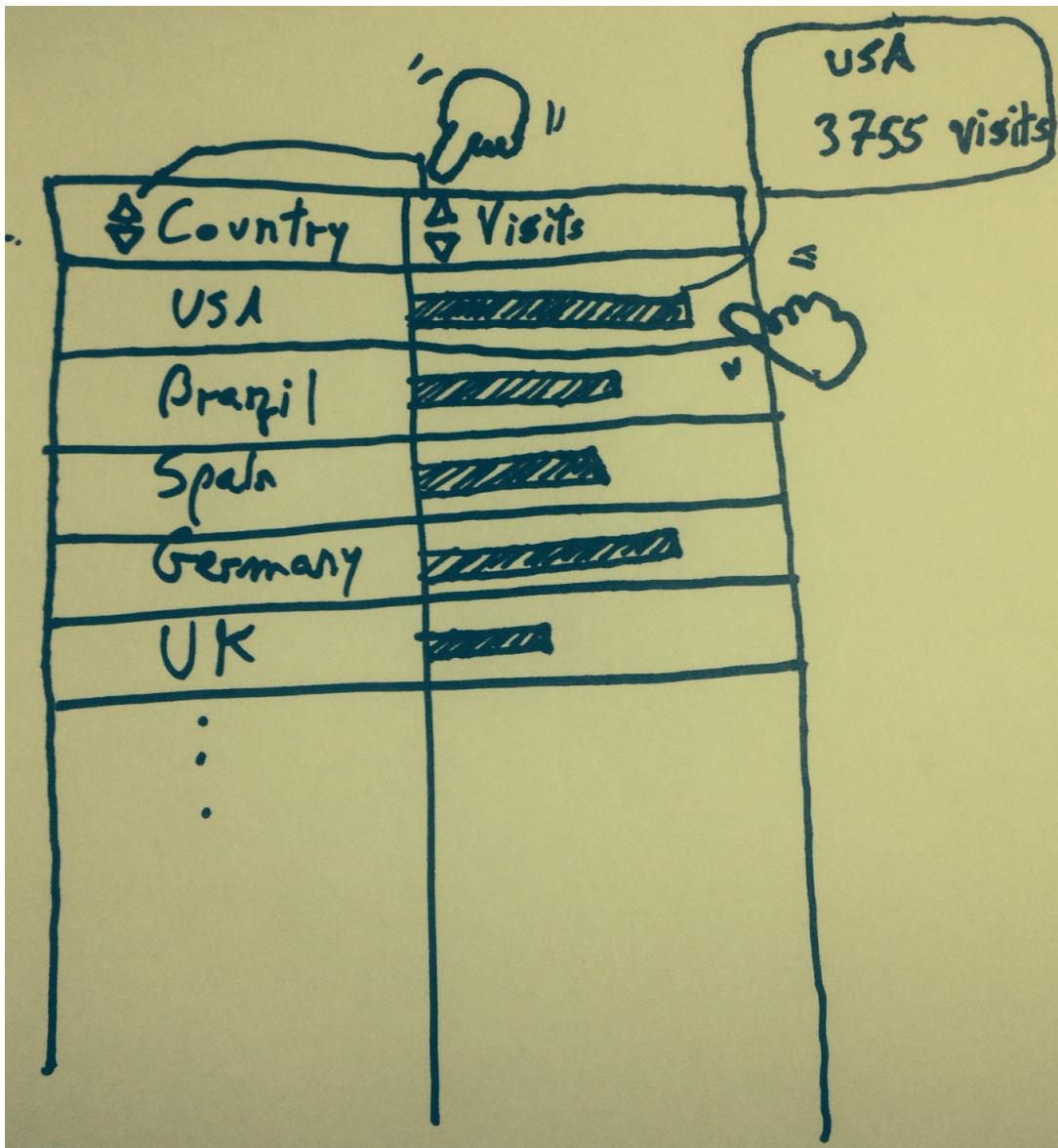


Figure 5 – An alternative to the choropleth map in Figure 4.

I built the system in the following chronological order:

1. Abstracted the web traffic data that needs to be captured and develop the business model. In other words, I first defined what data was useful and how to develop the algorithms to capture it

2. Built a relational database in MySQL around the results from 1)
3. Developed a server-side framework with PHP that captured and retrieved the collected data
4. Researched and experimented with different visualizations that could best fit 2)
5. Developed and implemented the client-side visualization using D3.  
I captured the data from the server side in JSON format

## 3 Prior Work

### 3.1 Visualization

Prominent researchers in the field of visualization have concocted logical constructs and approaches to designing and implementing visualizations. From a presentation standpoint, there is much emphasis on presenting the end user just enough visual data for the user to achieve the intended task of the visualization. Notions such as expressiveness [1], and increasing the data-to-ink ratio [2] allude to the goal of building visualizations that can be efficiently processed by the human brain. Another criterion that is opted for is effectiveness. Enter the concepts of a domain and task [3]. In the design phase, the builder needs to identify the domain that he/she is working with. A domain is a world of ideas and concepts that are abstracted under one umbrella. For example, human anatomy, software engineering, car racing, and pretty much any category one can think of is a domain. A domain gives rise to concepts, definitions, and a prebuilt library of shapes and colors by the convention of collective knowledge. For example, red in biology will signify blood whereas in the world of novels it may allude to romance. Knowing the domain aids the designer in introducing artifacts effectively and efficiently. On the other hand, the task refers to the "what" of the visualization. What is/are the goal(s) of the visualization? Is it to explore, analyze, extrapolate, etc.? Clearly identifying the task(s) of the visualization at hand solidifies the direction in which the designer encodes the visual variables (Bertin). Encoding refers to the idea of representing a quantitative or qualitative concept visually. The visual variables include position, size, shape, value, color, orientation, grain, and texture. At this point of the design process, most of the roadmap has been laid out for one to develop design sketches. Perhaps in the pre-Internet days, the implementation would have started here. Though, with the advent of mass computerization and global connectivity, a new ability has been introduced - that of interactivity. Interactivity has been broken down into the following [4]: Overview+Detail, Zooming, Focus+Context, details-on-demand, and cue-based systems.

Time is an essential component to web analytics, as it gives context to what is being measured. Although graphing change over time was attempted in the early A.D.s, the first known contemporary time series graph was published by William Playfair [5]. Playfair was the father of

the line graph and bar graph; it is important to note that these visual constructs were introduced in a time when they were not commonplace. Given this, a likely inclination towards visualizing temporal data would be to plot time against a Cartesian coordinates system. However, there is not a single model that accommodates all domains and tasks [6]. Thus, for the proposed visualization, time must be dealt with in the context of information systems in the news domain. As an employee of a news organization, I would like to take this opportunity to highlight the significant time units pertinent to news. Based on my observations of our news website and other similar entities' websites, "news" refers to a recent occurrence, notably something that happened today. "Breaking News" typically refers to an event that happened within the current day and whose story under development. "Recent News" pertains to stories created within the last 1-3 days. As time approaches the coming year, decade, century, and millennium, news are then compiled and shown in representations such as "year/decade/century/millennium in review". However, in order to achieve a meaningful representation in our visualization I must turn attention to the average lifetime of an article on a news website. Surely, a news article is only news when it is recent. Later references could be made to the said article when it is its "archives" period, but this would be only a temporal slice that would not be sufficient or useful when the task at hand is to have the user navigate based on article popularity. Thus, the time units (aka time granularities) that will support the task of the proposed visualization should be minutes and hours (for breaking news), and hours and days (for recent news) [8]. Since I have interest in the general range of these units (e.g. 20 users looked at our article in the last minute/hour/etc.), I will need to use a discrete time scale [7]. The output variables connected to time will mainly be number of hits, and may include figures such as the number of Facebook likes, number of Twitter tweets, and the average read time of an article.

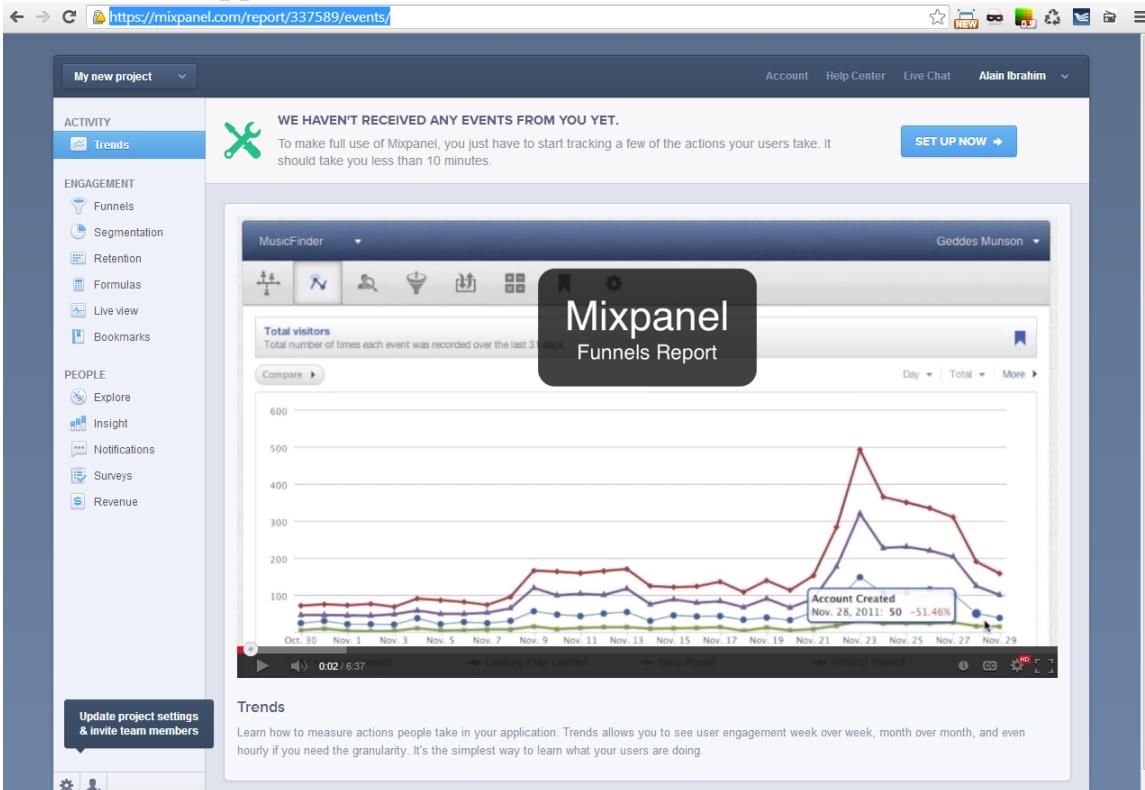
### 3.2 Existing web metrics systems

What I plan to build is novel in that there is no current navigable tool that visualizes near real-time traffic on news sites. It should not only convey near real-time web traffic, but should also allow one to view metadata upon hovering over the selected article's representation, and should allow

the user to directly navigate to it by clicking.

Here are a few systems that I found which deal with visualizing web metrics:

### Flow (<http://www.webresourcesdepot.com/beautiful-free-website-traffic-visualization-application-flow/>)



**Figure 6 – Shows a line graph that measures the number of hits on the y axis, and date on the x axis for select categories on a given website.**

While this can be useful in providing some feedback to the news site's administrators, it does not provide detailed insight to the articles in a given news site. The below screenshot demonstrates another visualization part of Flow.

In order to understand how your website is being consumed and to improve it, **analyzing how users browse the website and which paths they follow is so important**.

Although popular analytics apps like Google Analytics provide this information, [\*\*Flow\*\*](#) differentiates itself by focusing only to the visualization of paths, displaying **real-time data** and offering all of this with a slick, **impressive and intuitive interface**.

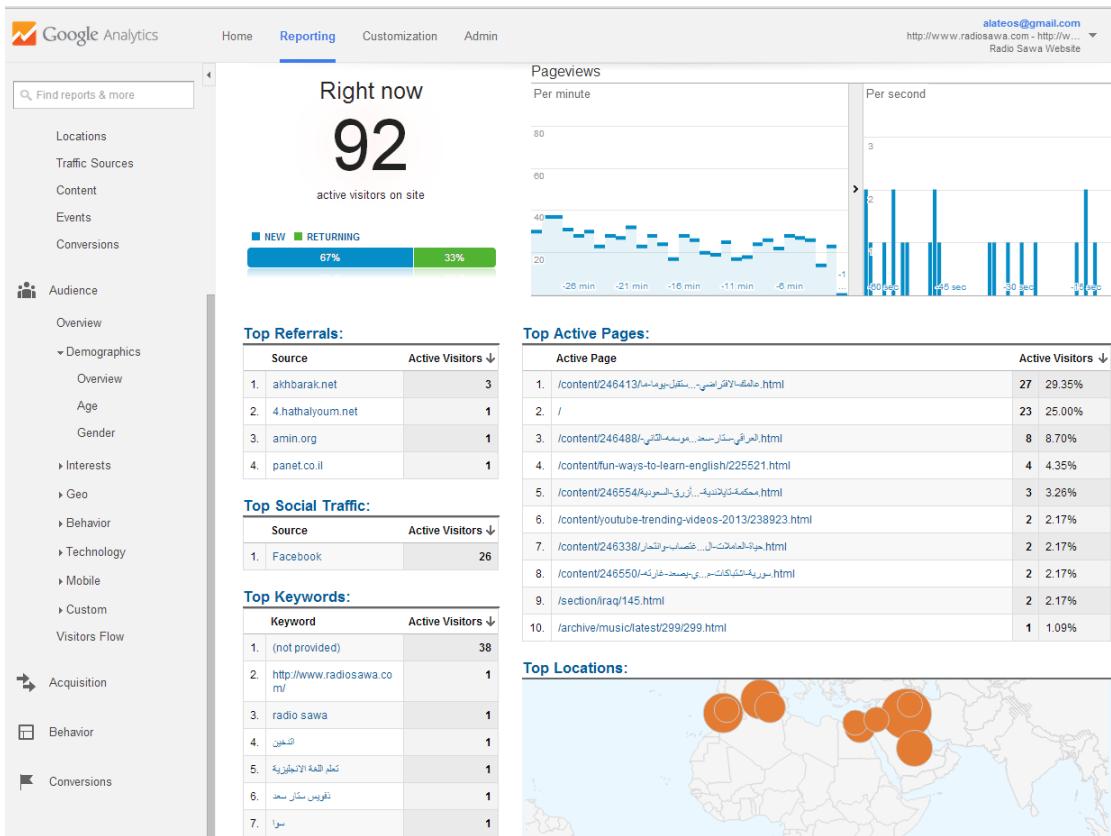


Rather than a chart, it **displays a diagram of the actual paths** people take when they browse your site and you can easily drill-down by clicking each item.

Also, it **presents the number of people leaving from each page** which is a great feedback to find out if those pages require improvements or not.

**Figure 7 – A visualization showing near real-time hits in sections of a given website.**

## Google Analytics



**Figure 8 – shows the Real-Time metrics pane in Google Analytics.**

As one can see, Google Analytics comes in handy in aggregating data and conveying statistical data to the end user. The end user here would navigate here with the intention of running an analysis, like: From what parts of the world is most of this website's traffic coming from? What links are being most visited? While this would be a great tool for management, it is unlikely that the end users will wake up to their coffee looking at this, and then navigating to the actual news website.

## Adobe Analytics

## What is Adobe Analytics?

Adobe Analytics helps you create a holistic view of your business by turning customer interactions into actionable insights. With intuitive and interactive dashboards and reports, you can sift, sort, and share real-time information to provide insights you can use to identify problems and opportunities.



## Adobe Analytics capabilities.



### Marketing reports and analytics

Make use of advanced analytics with real-time reporting, powerful visualizations, and dashboards that arm you with the insights you need to guide your business.



### Ad hoc analysis

Discover a comprehensive, multidimensional view of customer segments that you can use to make accurate, timely, and insightful decisions and improve performance.



### Data workbench

Measure, analyze, and optimize integrated data from online and offline marketing channels, from high-level trends down to individual customer behavior — all in one place.



### Predictive analytics

Easily analyze customer data and segment audiences to predict the success of your campaigns and maximize the impact of your ad spend.



### Real-time web analytics

Access second-to-second analytics reports and real-time tools that allow you to react instantly to visitor trends.



### Mobile analytics

Dive deep into the performance of your mobile campaigns. Understand how your mobile customers uniquely engage your brand.

**Figure 9 – Snapshot of the Adobe Analytics landing page, which enumerates the services that Adobe Analytics encompasses.**

Adobe Analytics covers a slew of reporting capabilities:

- **Marketing reports and analytics.** Includes dashboards and reports, conveying key performance indicators, showing marketing strategies' performance in real-time, and displaying the results of ad campaigns
- **Ad hoc analysis.** Includes multidimensional site analysis and advanced reporting
- **Data workbench.** Involves collecting, processing, analyzing, and visualizing Big Data from online and offline customer interactions

- **Predictive analysis.** Pertains to discovering hidden customer behavior
- **Real-time web analytics.** Shows what is happening on the digital properties in real-time so that managers can take action where needed

Once again, the intended audience is that of management and decision makers. The current prominent web analytics systems revolve around reporting data to the decision makers. What I plan to create is for the end users.

## 4 Work Plan

### 4.1 Assumptions, Risks and Alternatives

I will require a web host where I can test the web technologies as well as store my database. I currently am subscribed to a web host that provides me with a LAMP (Linux, Apache, MySQL, PHP) environment. This is where I plan to execute all my development and testing. The scripts technologies to be used are HTML 5, CSS3, PHP, MySQL, and JavaScript. The libraries I anticipate to use are D3 and possibly JQuery. I can foresee the following risks:

- The web traffic captured will not be 100% accurate. Then again, no web analytics system out there is perfect as of this writing, given the stateless nature of http and how http traffic is passed and downloaded over the net. My goal here is to maximize accuracy, given my research and best judgment
- The visualization turns out to be slow due to the web browser rendering too much data at once. In this event, I will scale down the complexity of the visualization to match the current average machine's CPU power and web browser engine capabilities. Alternatively, I can utilize AJAX (where applicable) to render the required data on demand

Basically, my strategy for mitigating any further risks will be to narrow the scope of the end product. In the lack thereof, I plan to fully capitalize on enhancing functionality.

### 4.2 Preliminary Schedule

**Weeks 1-3** (Research what data needs to be captured):

Research the metrics that need to be captured. What defines a website visit? What data will need to be collected?

**Weeks 4-6** (Design a relational database to house desired data):)

Here, I will define the relational schemas and will accordingly build a MySQL database that will contain all of my desired data.

Deliverable — Relational database to house all data pertinent to website traffic hits.

**Weeks 7-10** (Code and implement a PHP implementation for data storing/retrieval):

Here, I plan to build PHP classes and functions that will connect to the database to store the web traffic data. In addition, I will write the code that will query the database in order to extract and convey meaningful data.

Deliverable — Full server-side code base that collects and retrieves web traffic data.

**Weeks 10-12** (Implement mock news site):

Here, I plan to build a mini news site that contains several categories. Then, I will create several articles under each of the created categories. I am inclined to building this mock site in Wordpress, as it is commonplace and reflective of the archetype of your common CMS site. Deliverable — A mini Wordpress mock site, mimicking the structure of a news website

**Weeks 13-16** (Research and experiment with different visualizations):

Before I commit to a visualization, I would like to make sure that it is the most fit for my domain at hand. Thus I will test my data inside a few viable options, before fully scoping my visualization.

Deliverable — An existing or custom designed visualization template to convey near real-time web traffic data.

**Weeks 16-18** (Implement final visualization):

Once I have selected my visualization of choice, I will code and implement it.

Deliverable — A fully implemented web visualization which interacts with the server-side database.

**Weeks 18-20** (Test with users and apply final touches):

I will attempt to reach a wide and diverse audience for feedback. I will incorporate the meaningful feedback into my refinement iteration. "Meaningful" here represents anything significant enough to affect the usability of the visualization.

## 5 Implementation - data capture

### 5.1 Capturing Data

#### 5.1.1 IP Addresses

I plan to capture the IP addresses of the visitors to the subject news site. In PHP, an associative array called `$_SERVER` contains several variables that pertain to the HTTP connection between the client and the server. The variable that contains the IP address from which the user is viewing a select page is called `REMOTE_ADDR`. In code, we register the visiting IP address like so:

```
$user_ip = $_SERVER["REMOTE_ADDR"];
```

Here, I would like to shed light on the difference between static and dynamic IP addresses. The former is set by the network administrator and is fixed unless it is changed manually. The latter is assigned by the network router, and is assigned for a temporary period called the DHCP lease time. In order to find out the average DHCP lease time set by Internet Service Providers (ISPs), I asked a few network engineers and contacted 3 prominent ISPs. Here are the results:

Network Engineer 1	7 days
Network Engineer 2	3 days
Network Engineer 3	3 days
Verizon (ISP)	14 days
Comcast (ISP)	4-7 days

The average DHCP lease time is between 3 and 7 days. Since in my visualization I only care for recent and breaking news that span for 1-3 days, I can hold the distinction between static and dynamic IP addresses as constant - as the time span is less than the average renew period. This is important for monitoring the average time spent on an article and other metrics that rely on graduation of time. The IP address is the only means to track a user and that does not require user authentication.

As for user integrity, one cannot fully ensure that a physical human is sitting behind an IP address when bots, web scrapers, and other automated processes may be hitting the IP address of a given news article. Though, my goal is to minimize the inaccuracy when a visitor reaches a news article. My plan to counter this is to only register the visiting IP address when the user has moved his/her mouse, or, when he/she has scrolled at least once when viewing the article at hand. I have tested a stub for this successfully in `register_ip.php`. Here, I have a page containing several blocks of repeated Lorem ipsum text. I used JavaScript to listen to the mousemove and scroll events. I used jQuery to shorthand the event bindings. Below is the code:

```
mouse_moved = false;
page_scroll_counter = 0;

$("body").on("mousemove", function () {
    if (!mouse_moved) {
        \\ Here, a flag would be sent to the server via AJAX
        console.log("Mouse has moved");
        mouse_moved = true;
    } else {
    }
});

$(document).on("scroll", function () {
    if (page_scroll_counter == 1) {
        \\ Here a flag would be sent to the server via AJAX
        console.log("Page has been scrolled")
    } else {
        page_scroll_counter++
    }
});
```

### 5.1.2 Geographic origins

The world is smaller than it has ever been. One of the metrics I am opting for is the user geographic distribution for a select news article. This requires two steps:

1. Capturing the IP address of the user. This was demonstrated in the previous section
2. Obtaining the corresponding latitude and longitude coordinates associated with the user's IP address

Here, it is important to note that many users may be sitting behind HTTP proxy servers. HTTP proxy servers are physical computers/servers that act as proxy points for the client computers. For example, if someone in Russia used a proxy server in the US to visit my news article, the requesting IP address that will be captured will be that of the US - making it appear that the user that visited the article actually came from the US. In PHP, in addition to `$_SERVER["REMOTE_ADDR"]`, there is another variable called `$_SERVER["HTTP_X_FORWARDED_FOR"]`. While the former gets the IP address of the direct requester, the latter gets the originating IP that is making the request. So for example, if I were to be sitting behind an IP proxy server which had an address of 111.111.111.111, then `$_SERVER["REMOTE_ADDR"]` would carry that value.

`$_SERVER["HTTP_X_FORWARDED_FOR"]` would then contain the value of my router's outside IP address - which is what I am looking to map geographically.

If on the other hand the user was browsing the Internet normally, without the use of a proxy server, then `$_SERVER["REMOTE_ADDR"]` would carry the value of the user's router's IP address - reflective of the user's geography. It will thus be a challenge to capture the needed IP address to look up geographically. Here is the methodology that I will use to capture the needed IP address for the geographic lookup:

1. Get the value of `$_SERVER["REMOTE_ADDR"]` and look up its value with a geolocation API. Call this X
2. Get the value of `$_SERVER["HTTP_X_FORWARDED_FOR"]` and look up its value with a geolocation API. Call this value Y
3. If X and Y are both available, it means that the user is sitting behind a proxy and that the needed IP address is that of Y. If X is available but Y is not, it means that Y is a private IP address, that of the device behind its router - and thus not behind a proxy server. In this case, X would hold the IP address of the origin.

The next step here is to find a service that would map a given IP address to its corresponding geo coordinates, and ideally to its country name. A quick Google search revealed a convenient REST API service that would output the responses in JSON. **Telize** ([www.telize.com](http://www.telize.com)) is a REST API that allows one to get the visitor IP address and to query the location information for that IP address. The upside of this service is that at the moment

it contains no rate limits; thus, one could make an indefinite amount of calls to the Telize API. I was able to test the above clause 3 in PHP code with success. Regardless, of whether I was sitting behind a proxy server or not, the output showed the United States as the country of origin. Here is the code:

```

date_default_timezone_set("America/New_York");

$remote_addr = $_SERVER["REMOTE_ADDR"];
$x_forwarded_for = $_SERVER["HTTP_X_FORWARDED_FOR"];

$url1 = "http://www.telize.com/geoip/" . $remote_addr;
$url2 = "http://www.telize.com/geoip/" . $x_forwarded_for;

// Get the country for the ip behind the first
// exposed remote address (from www.telize.com)
$results1 = file_get_contents($url1);
$results1 = json_decode($results1);
$country1 = $results1->country;

// Get the country for the ip behind the second
// exposed remote address (from www.telize.com)
$results2 = file_get_contents($url2);
$results2 = json_decode($results2);
$country2 = $results2->country;

$final_country = "";

if(strlen($country1) > 0 && strlen($country2) > 0) {
    echo "User is most likely behind a proxy server.<br />";
    $final_country = $country2;
} else {
    echo "User is not behind a proxy server.<br />";
    $final_country = $country1;
}

echo "The user's country of origin is " . $final_country;

```

### 5.1.3 Average Read Time

One metric that I would be curious to attain is how long the average user spends reading a certain news article. At the moment, a cousin measure called the bounce rate checks to see whether the user continues to browse other pages in the site, or whether the user 'bounces' off to other websites. Naturally, this measure is not indicative of the level of interest the reader has for a given news article. I plan to get this by doing the following:

1. Research the average reading speed of adults that can read English

2. Count the number of words in a given news article
3. Based on the average read speed, calculate the time spent on the news article. If the page is kept open indefinitely by the reader, I will simply use the maximum expected read time for that article. While this is in no way 100% accurate, it will be highly reflective of the level of interest with the visited articles.

Based on an ophthalmology study that consisted of 50 native English-speaking individuals (average age  $30.38 \pm 9.44$  years; 16 university students, 18 academics, and 16 non-academics) – the average reading speed came out to be  $201.53 \pm 35.88$  words per minute(wpm) for short sentences, and  $215.01 \pm 30.37$  wpm for long paragraphs [9]. For the scope of this project, I will use the average of 200 wpm to calculate the maximum average time spent on a news article.

## 5.2 Database Design

### 5.2.1 Data Storage

Based on the data that will need to be captured, below are the fields that should be created:

- **Visitor IP address:** This will basically store the IP address of the visitor reading the news article. In order to avoid capturing page refreshes as new sessions, I will set a threshold of 10 minutes - whereby the server checks if the visitor has visited the article in the past 10 minutes
- **Time Visited:** This should store a timestamp (in GMT) of when the user visited
- **Timezone:** This will be helpful in determining time differences from GMT
- **Country:** This will store the name of the country where the visitor is from
- **Region:** Region pertains to the area within the country where the visitor is from
- **Read Time:** This will store the time taken for the reader to have finished reading the article. This should be calculated in seconds
- **Article ID:** This will be the unique identifier and should be captured as is from the host's CMS. This way, future references could be made easily without having to go through matching algorithms
- **Category ID:** This will be the unique identifier of the category, also as it appears in the corresponding CMS
- **Article URL:** this will be needed to enable the user to navigate through the visualization to the intended article
- **Article title:** The title of the article as it appears in the CMS or on the news site
- **Sample text:** In the visualization, some of the text from the body will be excerpted upon the user hovering on a select article

- **Sample picture:** In the visualization, the user will need to see a preview pic of the article when applicable

### 5.2.2 Database Tables

Here, I will adhere to the principles of relational database normalization in order to maximize efficiency and eliminate redundancy. For the scope of this project, all the data pertinent to a web hit can be stored in a single table:

**hit (id,ip, time\_visited, *article\_id*, timezone, country, region, read\_time)**

The underline signifies the primary key. The primary key is the field(s) that are enough to determine the rest of the information for any row instance. In this case, the IP address, article ID, and time of visit jointly determine the timezone, country, region, and read time. However, since I need a way to track the user's visits to capture read time, I have designated an auto-increment unique identifier called **id**. *article\_id* is italicized since it is a foreign key – that is, it represents a primary key in another table. The time visited will be abstracted to hour:minute. Therefore, seconds will not be accounted for in the visit's time stamp. The read time will default to the maximum read time as calculated by the article's length. This assumes that the user will stay on the article page indefinitely. In the event that the user does leave the page, this value would then register the actual read time.

The second table needed will contain all the metadata pertinent to the visited article:

**article (article\_id, category\_name, article\_url, title, sample\_text, sample\_pic)**

The third table needed will contain metadata pertinent to the category of the visited article:

**category (category\_id, category\_name)**

It is important to note here that all of this data will be communicated

via the server side script which will be embedded in each loaded article.

### 5.2.3 MySQL Database

For this project, I will be using MySQL since I have good experience with it and for its being open source. MySQL is a relational database management system. Relational databases are those that adhere to relational database design, which relies on the fundamentals of relational algebra and relational calculus. The main tenet of relational design is to maximize efficiency and eliminate any redundancy in the data being stored.

I created a database called **alaini\_news\_viz** on my web host. For this, I created two users:

- **alaini5\_newsread**: This account will be solely used to read from the database. Ultimately, it will be used to invoke the visualization by the visitor/end user
- **alaini5\_newsedit**: This account will mainly be used to register the IP hits and read times communicated to the server

I created 3 tables in MySQL using phpMyAdmin, an open source PHP based GUI that facilitates interacting with the MySQL system:

1. **article**

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	article_id	int(11)			No	None	AUTO_INCREMENT
2	category_id	int(11)			No	None	
3	article_url	varchar(150)	latin1_swedish_ci		No	None	
4	title	varchar(150)	utf8_general_ci		No	None	
5	sample_text	text	utf8_general_ci		No	None	
6	sample_pic	varchar(150)	latin1_swedish_ci		No	None	

Check All  With selected:

[1] column(s)  At End of Table  At Beginning of Table  After

+ Indexes

**Information**

<b>Space usage</b>	<b>Row statistics</b>
Data	0 B
Index	1 KiB
Total	1 KiB
Format dynamic	
Collation	latin1_swedish_ci
Rows	0
Next autoindex	1
Creation	Jun 19, 2014 at 04:14 PM
Last update	Jun 19, 2014 at 04:14 PM

## 2. category

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	category_id	int(11)			No	None	
2	category_name	varchar(100)	latin1_swedish_ci		No	None	

Check All  With selected:

[1] column(s)  At End of Table  At Beginning of Table  After

+ Indexes

**Information**

<b>Space usage</b>	<b>Row statistics</b>
Data	0 B
Index	1 KiB
Total	1 KiB
Format dynamic	
Collation	latin1_swedish_ci
Rows	0
Creation	Jun 19, 2014 at 04:15 PM
Last update	Jun 19, 2014 at 04:15 PM

## 3. hit

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	<a href="#">id</a>	int(11)			No	None	AUTO_INCREMENT
2	<a href="#">ip</a>	varchar(15)	latin1_swedish_ci		No	None	
3	<a href="#">time_visited</a>	int(15)			No	None	
4	<a href="#">article_id</a>	int(11)			No	None	
5	<a href="#">timezone</a>	varchar(30)	latin1_swedish_ci		No	None	
6	<a href="#">country</a>	varchar(80)	latin1_swedish_ci		No	None	
7	<a href="#">region</a>	varchar(80)	latin1_swedish_ci		No	None	
8	<a href="#">read_time</a>	int(5)			No	0	

Check All    With selected:    [Browse](#)    [Change](#)    [Drop](#)    [|](#)

[Print view](#) [Propose table structure](#) [Move columns](#)

[Add](#)  column(s)  At End of Table  At Beginning of Table

+ Indexes

**Information**

Space usage		Row statistics	
Data	0 B	Format	dynamic
Index	1 KiB	Collation	latin1_swedish_ci
Total	1 KiB	Rows	0
		Next autoindex	1
		Creation	Aug 06, 2014 at 11:17 AM
		Last update	Aug 06, 2014 at 11:17 AM

## 5.3 PHP server-side code

### 5.3.1 Configuration and database connection

Here, I created two files:

1. **constants.php**: Holds all the constants that will be defined across the application, inclusive of those for the database connection
2. **db\_connect.php**: Performs the PDO database connection based on the given connection parameters

Note that both of these files are stored one directory above **public\_html/** for security reasons. This folder is called **lib/**. Thus, the two files are not accessible directly to the public and accordingly need to be included in other PHP files.

### 5.3.2 Server-side hit, article, and category capture

Each table in my database is represented with a PHP class. Hence, the creation of the following 3 classes:

1. **Hit.php:** This class registers all of the metadata pertinent to a user visit. This data includes IP address, country, region, timezone, and time visited. Below is the code:

```

/** This class is responsible for registering the user visits into the database
*/
class Hit {
    private $db = "";

    public function __construct($db) {
        $this->db=$db;
    }

    /**
     * Registers ip, country, region, timezone, time visited, and article id
    */
    public function register($article_id) {
        $data = $this->getHitData();
        $country = $data[0];
        $ip = $data[1];
        $region = $data[2];
        $timezone = $data[3];
        $time_visited = $data[4];

        // get the last time this ip address visited the specified article
        $last_visit_time = $this->getLastVisited($ip,$article_id);

        // if the last time this ip visited is greater than our specified wait limit, then record the visit as a new hit
        if((($time_visited - $last_visit_time) > WAIT_LIMIT) {
            $sql = sprintf("insert into hit(ip,time_visited,article_id,timezone,country,region,read_time) values('%s','%" . $this->db->query($sql);
        }

        // return the id of the inserted record to the client page, so that it may be used to track read time
        return $this->db->lastInsertID();
    }

    /**
     * Returns the last visited time, in Unix timestamp
    */
    private function getLastVisited($ip,$article_id) {
        $sql = "select * from hit where article_id=$article_id and ip='$ip' order by time_visited desc limit 1";
        foreach($this->db->query($sql) as $row) {
            $last_visited = intval($row["time_visited"]);
        }
        return $last_visited;
    }

    /**
     * Sets timezone
    */
    private function setTimezone($timezone) {
        date_default_timezone_set($timezone);
    }

    /**
     * Returns a data array that includes the country of origin, region, ip address, and timezone of visitor.
     * The 3rd party API used to extract the data is http://www.telize.com
    */
    private function getHitData() {
        $remote_addr = $_SERVER["REMOTE_ADDR"];
        $x_forwarded_for = $_SERVER["HTTP_X_FORWARDED_FOR"];
        $data = array();

        $url1 = "http://www.telize.com/geoip/" . $remote_addr;

```

```

$url2 = "http://www.telize.com/geoip/" . $x_forwarded_for;

// get the country for the ip behind the first exposed remote address (from www.telize.com)
if($remote_addr) {
    $results1 = file_get_contents($url1);
    $results1 = json_decode($results1);
    $country1 = $results1->country;
    $region1 = $results1->region;
    $timezone1 = $results1->timezone;
    $ip = $remote_addr;
}

// get the country for the ip behind the second exposed remote address (from www.telize.com)
if($x_forwarded_for) {
    $results2 = file_get_contents($url2);
    $results2 = json_decode($results2);
    $country2 = $results2->country;
    $region2 = $results2->region;
    $timezone2 = $results2->timezone;
    $ip = $x_forwarded_for;
}

// determines the true country of origin, depending on whether the user is behind a proxy server or not
if(strlen($country1) > 0 && strlen($country2) > 0) {
    $data[0] = $country2;
    $data[1] = $x_forwarded_for;
    $data[2] = $region2;
    $data[3] = $timezone2;
} else {
    $data[0] = $country1;
    $data[1] = $remote_addr;
    $data[2] = $region1;
    $data[3] = $timezone1;
}

// set timezone
$this->setTimezone($data[3]);

// get timestamp of visit
$date = new DateTime("NOW");
$timestamp = $date->format('U');
$data[4] = $timestamp;

return $data;
}
}

```

2. **Article.php:** This class captures all of the metadata pertinent to the article being visited. Such includes title, text excerpt, sample picture URL, and article URL. Below is the code:

```

/**
 *      This class is responsible for registering the metadata pertinent to an article
 */
class Article {
    private $db = "";

    /**
     *      assign database instance in constructor
     */
    public function __construct($db,$metadata = Array()) {
        $this->db=$db;
        $this->register($metadata);
    }

    /**
     *      registers the article 's title ,category ,url ,sample text , and a sample image
     */
    private function register($metadata) {
        // get article id
        $article_id = $metadata["article_id"];

        // insert the article metadata into the database
        $category_id = $metadata["category_id"];
        $article_url = $metadata["url"];
        $title = $metadata["title"];
        $sample_text = $metadata["sample_text"];
        $sample_pic = $metadata["sample_pic"];
        $sql = "insert into article(article_id,category_id,article_url,title,sample_text,sample_pic) values (:article_id,:category_id,:article_url,:title,:sample_text,:sample_pic)";
        $sth = $this->db->prepare($sql);
        $sth->bindParam(":article_id",$article_id,PDO::PARAM_INT);
        $sth->bindParam(":category_id",$category_id,PDO::PARAM_INT);
        $sth->bindParam(":article_url",$article_url,PDO::PARAM_STR);
        $sth->bindParam(":title ",$title ,PDO::PARAM_STR);
        $sth->bindParam(":sample_text ",$sample_text ,PDO::PARAM_STR);
        $sth->bindParam(":sample_pic ",$sample_pic ,PDO::PARAM_STR);
        $sth->execute ();
    }
}

```

3. **Category.php:** This class captures any metadata pertinent to the parent category of the visited article. At the moment, I am only capturing the name of the category. Below is the code:

```

/**
 *      This class is responsible for registering the metadata pertinent to a category
 */
class Category {
    private $db = "";

    /**
     *      assign database instance in constructor
     */
    public function __construct($db,$metadata = Array()) {
        $this->db=$db;
        $this->register($metadata);
    }

    /**
     *      registers the category's name
     */
    private function register($metadata) {
        // get category id
        $category_id = $metadata["category_id"];
        // get category name
        $category_name = $metadata["category_name"];

        // insert category into database
        $sql = "insert into category(category_id,category_name) values (:category_id ,:category_name)";
        $sth = $this->db->prepare($sql);
        $sth->bindParam(":category_id ",$category_id ,PDO::PARAM_INT);
        $sth->bindParam(":category_name ",$category_name ,PDO::PARAM_STR);
        $sth->execute ();
    }
}

```

In addition to the above, I have created a helper file that registers the read time of a given article. By default, the read time is registered as 0 seconds in the database. Once a hit is registered in the database, the id of the newly created hit is then passed to the DOM of the article page. There, the id is used via an AJAX call to store the calculated read time for the visited article in the database. This file is called **updateReadTime.php**. Below is the code:

```

/**
 *      This page is accessed via an AJAX call. It updates the read time given an article id.
 */
include("../lib/db_connect.php");
$read_time = $_POST["read_time"];
$hit_id = $_POST["hit_id"];

// if a read time and hit id are provided then update the read time in the database
if($read_time > 0 && $hit_id > 0) {
    $stmt = $db->prepare("update hit set read_time=:read_time where id=:hit_id ");
    $stmt->bindParam( ':read_time ', $read_time );
    $stmt->bindParam( ':hit_id ', $hit_id );
    $stmt->execute ();
}

```

## 5.4 Client-side read time capture with Javascript

As mentioned in the last section, the hit id is passed to the client page in order to keep a way to identify and update the read time for the visited article. Here, I calculate the number of words that are stored in the article. I do this by first getting all of the text found within common article tags - namely `<span>`, `<div>`, `<article>`, `<p>`, and `<section>`. I then sum the number of words found within all of the mentioned tags. I ensure that double counting is avoided by using the `innerText` property; this counts the elements inner text, but not any text that might be included in one of its children nodes. Next, I calculate the number of seconds the article is on average expected to take to read. Based on my earlier research, I will use the average figure of 200 words per minute. This figure is merely used in the event that a user leaves the page open indefinitely. Thus, if an article should take about 5 minutes to read, and the user takes (for whatever reason) 30 minutes to navigate away from or to close the article, then I simply register the 5 minutes. Although I may be compromising some accuracy here, it is much more advantageous than having instances greatly inflate the read time.

Once the article page loads, and the expected read time calculations have been made, a timer is readied. The timer is triggered once a user moves the mouse or when the user scrolls the page. This minimizes storing metadata created by bots and to a high extent ensures that a human is sitting behind the screen. Upon navigating away from or closing the article, the read time is then submitted to `updateReadTime.php` and eventually recorded in the database. This part of the implementation relies on `window.onbeforeunload` or `window.onunload`, depending on the browser's user agent. At the time of this writing, this method only for Firefox and Chrome.

Below is the code pertinent to capturing read time:

```
// the average number of words per minute the user is expected to spend reading an article
var AVERAGE_WPM = 200;
// the maximum amount of time the user is expected to take to read the article
var maxTimeToSpend = 0;

/**
 * Gets the number of words from the article based on common tags used
 */
var getNumberOfWords = function() {
    var total_words = 0;
    var tags = ["span", "div", "article", "p", "section"];
```

```

// Go through each one of the designated tags.
// these are the common tags that contain
// article content. Then, get the number of
// words under the pertinent tag.
tags.forEach(function(d) {
    try {
        if(Array.prototype.slice.call($(d),0).length > 1) {
            $.each($(d),function(index,value){
                total_words+=$(value).text().split(' ').length;
            });
        } else {
            total_words+= $(d).text().split(' ').length;
        }
    } catch(e) {
    }
});
return total_words;
}

/**
Creates a Timer prototype that will keep track of time
*/
var Timer = function() {
    var seconds = 0;
    var timerRef = null;
    this.countTime = function() {
        timerRef = setInterval(function(){
            seconds+=1000;
        },1000);
    }
    this.stopCount = function() {
        clearInterval(timerRef);
    }
    this.getCount = function() {
        return seconds;
    }
}

/**
Gets the time spent in front of the article.
If it is greater than the expected time, then use the expected time. Else, use the computed time.
*/
var getTimeSpent = function() {
    timer.stopCount();
    var words = getNumberOfWords();
    var maxTimeToSpend = (words/AVERAGE_WPM)*60;
    var calculableTime = timer.getCount()/1000;
    if(parseInt(calculableTime) > parseInt(maxTimeToSpend)) {
        return maxTimeToSpend;
    } else {
        return calculableTime;
    }
}

/**
Sends the time spent on the article to the server

```

```

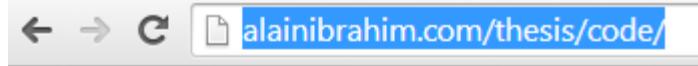
*/
var sendTime = function() {
    $.ajax({
        url: "updateReadTime.php",
        type:"post",
        data: {hit_id:<?php echo $hit_id ?>,read_time:getTimeSpent()}
    }).done(function() {
        });
}

/**
     initiate the timer and the bindings on load
*/
$( document ).ready(function() {
    // create a new Timer object
    timer = new Timer();
    // begin counting
    timer.countTime();
    // have the time sent to the server once the window
    // closes or if the user navigates to another page
    if (!!window.chrome) {
        window.onbeforeunload = sendTime;
    } else {
        window.onunload = sendTime;
    }
});

```

## 5.5 Tying it all together

Now that both the back end and front end code has been implemented, I can move on to testing this system. I have created a miniature lorem ipsum page which acts as an article being visited.



...  
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sit amet felis elit. Pellentesque consequat pharetra ante pede eu lectus non turpis interdum elementum ac vel diam. Donec molestie feugiat vitae quis felis.

In my PHP code, I populated mock metadata for the article and category. The hit metadata, however, will be populated based on the visit data. I went ahead and visited the page for 10 seconds and here are the results:

A screenshot of a MySQL database interface showing the results of a query. The query is "SELECT \* FROM 'hit'". The results table has columns: id, ip, time\_visited, article\_id, timezone, country, region, and read\_time. There is one row of data: id=1, ip="98.172.153.142", time\_visited="1408462100", article\_id="1", timezone="America/New\_York", country="United States", region="Maryland", read\_time="10". Below the table are buttons for Edit, Copy, Delete, Change, Delete, and Export. There are also dropdown menus for "Number of rows" set to 25 and "Options".

+ Options	←	→	▼	id	ip	time_visited	article_id	timezone	country	region	read_time
<input type="checkbox"/> <a href="#">Edit</a> <a href="#">Copy</a> <a href="#">Delete</a>	1	98.172.153.142	1408462100	1	America/New_York	United States	Maryland	10			

My ip address, time of visit, article id (at this point a fake one), timezone, country, region, and read time were all stored in the **hit** table.

SELECT * FROM `article`					
<input type="checkbox"/> Profiling [ Inline ] [ Edit ] [ Explain SQL ] [ Create ] Number of rows: 25 ▾					
+ Options      ↲ T →      article_id category_id article_url      title      sample_text      sample_pic					
<input type="checkbox"/>	<a href="#">Edit</a>	<a href="#">Copy</a>	<a href="#">Delete</a>	1	1   http://alainibrahim.com/thesis/code   alain is testing an article   Lorem ipsum dolor sit amet, consectetur adipiscing...   http://alainibrahim.com/thesis
<a href="#">With selected:</a>	<a href="#">Change</a>	<a href="#">Delete</a>	<a href="#">Export</a>		
Number of rows: 25 ▾					

The article id, category id, article URL, article title, article sample text, and the URL to the article's main pic - were all stored in table **article**.

SELECT * FROM `category`	
<input type="checkbox"/> Number of rows: 25 ▾	
+ Options      ↲ T →      category_id category_name	
<input type="checkbox"/>	<a href="#">Edit</a>
<a href="#">Copy</a>	<a href="#">Delete</a>
2	Thesis

Lastly, the category id and name were both stored in the **category** table.

This concludes the part of the system that implements the data capturing. The next step is to connect this library into a working website. For the scope of this thesis, I will implement a hook that ties into a widely used Content Management System - Wordpress.

## 6 Implementation - hooking into Wordpress

Wordpress calls its articles "posts". The template file for each post is called **single.php** and is found under the current theme that is being used. One can think of a theme as a set of pre-built PHP, HTML, CSS3, and Javascript templates that structurally and visually dictate how the website will look and behave. Thus, in order to accomplish the task of capturing the data from a said article, I need to include my capture library inside **single.php**. However, since my library cannot directly communicate with Wordpress, I must create a hook that abstracts some of Wordpress's built-in functions. Lastly, I need to ensure that Wordpress is using jQuery version 2.1.1 or higher. As this is a Javascript library and that I need available as soon as the page loads, I will need to include it in **header.php** - which basically comprises the `<head></head>` portion of a given page. Below is a breakdown of how I hook into Wordpress.

### 6.1 Creating the captureLib folder

In the interest of portability, I have placed all of my code within a folder called **captureLib**. If a 3rd party wishes to uses this library in the future, all that would be needed is to simply include **captureLib** from within **single.php**. Here is a screenshot of the current directory structure of **captureLib**.

Name	Date modified	Type	Size
lib	9/2/2014 10:17 AM	File folder	
Article.php	8/31/2014 6:26 PM	PHP File	2 KB
captureHit.php	9/11/2014 4:41 PM	PHP File	3 KB
Category.php	8/17/2014 1:35 PM	PHP File	1 KB
countries.php	9/11/2014 10:54 AM	PHP File	4 KB
getReadTimeLibrary.php	9/1/2014 4:38 PM	PHP File	1 KB
Hit.php	9/11/2014 10:56 AM	PHP File	4 KB
readTime.php	9/1/2014 8:30 PM	PHP File	5 KB
updateReadTime.php	9/10/2014 5:34 PM	PHP File	1 KB
wordpressHook.php	9/11/2014 4:40 PM	PHP File	2 KB

Here is a breakdown of the current directory structure of the code that captures hits and read times.

1. lib/

- (a) **constants.php** - contains all the server-side constants that will be used to capture the hits
  - (b) **db\_connect.php** - connects to the database where the hits are stored
2. **Article.php, Hit.php, Category.php** - as mentioned previously, these are abstractions that communicate the data from and to the database
  3. **getReadTimeLibrary.php** - a line of code to include the read time Javascript code if a hit was registered

```
<?php
    /**
     * A terse way of including the Javascript library. The library
     * itself is suffixed with .php instead of .js because there is
     * a php variable defined in the actual Javascript library
     */
    if($hit_id) { include("readTime.php"); }
?>

\item
```

4. **readTime.php** - this is the Javascript code that captures the read time. It is suffixed with .php (vs. .js) since it requires PHP code to output the hit id within the Javascript. I have made adjustments to this code to accommodate for users switching tabs while reading a certain article. When the article's window loses focus, the timer stops counting; it only resumes when the user gets back to the article window. This maximizes accuracy in the average read time readings. Here is the code:

```
/**
 * This is the Javascript library that registers the read time of an article.
 * It is suffixed with a .php extension since the script requires the
 * hit id to be provided by php script.
 */

// the average number of words per minute the user is expected to spend reading an article
var AVERAGE_WPM = 200;
// the maximum amount of time the user is expected to take to read the article
var maxTimeToSpend = 0;
// flag to denote if the user has moved the mouse. if yes, the value is 1
var mouse_moved = 0;
// flag to denote if the user has scrolled the page. if yes, then the value is greater than 0
var page_scroll_counter = 0;
// flag to denote whether current tab is active or not. if yes, the value is 1
var current_tab_active = 0;

/**
```

```

        Gets the number of words from the article based on common tags used
*/
var getNumberOfWords = function() {
    var total_words = 0;
    var tags = ["span","div","article","p","section"];

    // Go through each one of the designated tags.
    // these are the common tags that contain
    // article content. Then, get the number of
    // words under the pertinent tag.
    tags.forEach(function(d,area) {
        area = ".entry-content";
        d = area + " " + d;
        try {
            if(Array.prototype.slice.call($(d),0).length > 1) {
                $.each($(d),function(index,value){
                    words_length = $(value).text().trim().split(' ').length;
                    if(words_length > 1) {
                        total_words+=words_length;
                    }
                });
            } else {
                words_length = $(d).text().trim().split(' ').length;
                if(words_length > 1) {
                    total_words+=words_length;
                }
            }
        } catch(e) {
        }
    });
    return total_words;
}

/**
Creates a Timer prototype that will keep track of time
*/
var Timer = function() {
    // number of seconds counted
    var seconds = 0;
    // reference to the Timer object
    var timerRef = null;

    /** increase the timed counter */
    this.countTime = function() {
        timerRef = setInterval(function(){
            seconds+=1000;
        },1000);
    }

    /** stop the timed counter */
    this.stopCount = function() {
        clearInterval(timerRef);
    }

    /** get the number of seconds that have elapsed */
    this.getCount = function() {

```

```

        return seconds;
    }
}

/**
Gets the time spent in front of the article.
If it is greater than the expected time, then use
the expected time. Else, use the computed time.
*/
var getTimeSpent = function() {
    // stop the timer
    timer.stopCount();
    // get total number of words in text
    var words = getNumberOfWords();
    // get the expected maximum time to read the text
    var maxTimeToSpend = (words/AVERAGE_WPM)*60;
    // get the number of seconds spent reading the text
    var calculableTime = timer.getCount()/1000;
    // if the expected number of seconds has been exceeded, then
    // return the expected number of seconds as the final measure.
    // else, return the actual time spent reading the text
    if(parseInt(calculableTime) > parseInt(maxTimeToSpend)) {
        return maxTimeToSpend;
    } else {
        return calculableTime;
    }
}

/**
Sends the time spent on the article to the server
*/
var sendTime = function() {
    $.ajax({
        url: "updateReadTime.php",
        type:"post",
        data: {hit_id:<?php echo $hit_id ?>,read_time:getTimeSpent()}
    }).done(function() {
        });
}

/**
initiate the timer and the bindings on load
*/
$( document ).ready(function() {
    // have the time sent to the server once the window
    // closes or if the user navigates to another page
    if (!!window.chrome) {
        window.onbeforeunload = sendTime;
    } else {
        window.onunload = sendTime;
    }

    $("body").on("mousemove",function(){
        // if the user moves the mouse, start the timer.
        // Also, cancel the event binding to both 'mousemove' and 'scroll'
        if (!mouse_moved) {

```

```

        mouse_moved = true;
        // create a new Timer object
        timer = new Timer();
        // begin counting
        timer.countTime();
    } else {
        $("body").unbind("mousemove");
        $(document).unbind("scroll");
    }
});

$(document).on("scroll",function(){
    // if the user starts scrolling the page, start the timer.
    // Also, cancel the event binding to both 'mousemove' and 'scroll'
    if(page_scroll_counter == 1) {
        $("body").unbind("mousemove");
        $(document).unbind("scroll");
    } else {
        // create a new Timer object
        timer = new Timer();
        // begin counting
        timer.countTime();
        page_scroll_counter++;
    }
});

// stop counting if user switches to another tab
$(window).on("blur",function() {
    timer.stopCount();
});

// resume counting if user switches back to current tab
$(window).on("focus",function() {
    timer.countTime();
});
});

```

5. **wordpressHook.php** - as mentioned earlier, this is an abstraction into Wordpress's needed built-in functions. In order to hook to different CMSs, the code can be reused. All that would be needed to change is the implementation. Here is the code:

```

/**
 * Class used to abstract the built-in and needed Wordpress functions
 */
class wordpressHook {
    // the article id to be set by the constructor
    private $article_id = "";

    // set the article id in the constructor
    public function __construct($article_id) {
        $this->article_id = $article_id;
    }

    // get the name of the category from Wordpress
}

```

```

public function getCategoryName() {
    $category_name = get_the_category()[0]->cat_name;
    return $category_name;
}

// get the id of the category from Wordpress
public function getCategoryID() {
    return get_cat_id($this->getCategoryName());
}

// get the url of the article from Wordpress
public function getURL() {
    return get_permalink($this->article_id);
}

// get the title of the article from Wordpress
public function getTitle() {
    return get_the_title($this->article_id);
}

// get an excerpt from the read article from Wordpress.
// The number of characters excerpted is based on the constant EXCERPT_CHARS
public function getSampleText() {
    $sample_text = get_post($this->article_id,ARRAY_A);
    $sample_text = strip_tags($sample_text["post_content"]);
    $sample_text = substr($sample_text,0,EXCERPT_CHARS);
    return $sample_text;
}

// get the article's featured image URI from Wordpress
public function getSamplePic() {
    $sample_pic = wp_get_attachment_image_src( get_post_thumbnail_id( $this->article_id));
    $sample_pic = $sample_pic[0];
    if(strlen($sample_pic) < 5) {
        $sample_pic = "none";
    }
    return $sample_pic;
}
}

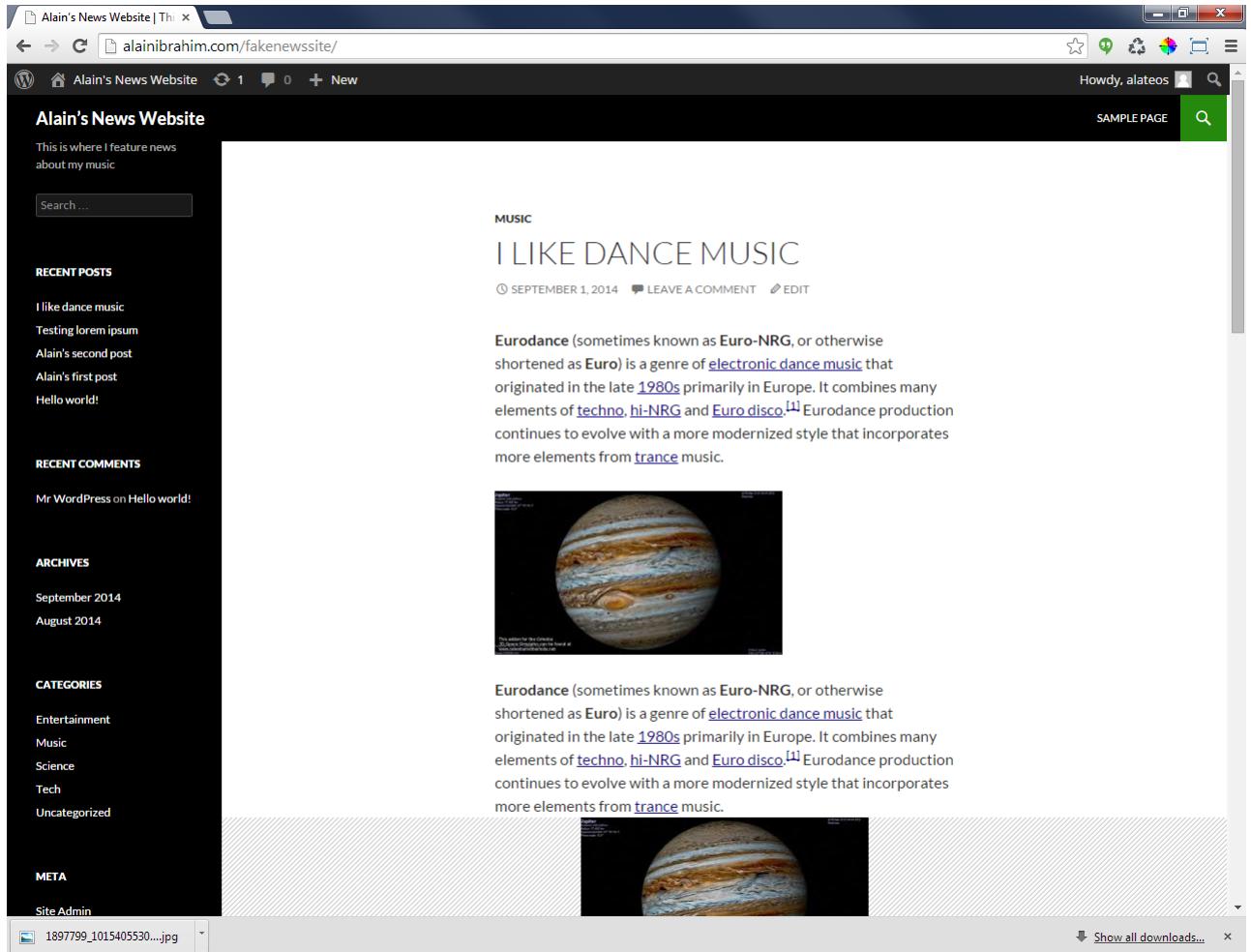
```

6. **countries.php** - contains an array that has all country names. Used for database seeding.

## 6.2 Creating a mock news website

In order to test the library, I needed to create a mock news site in Wordpress. In this setup, I built the following categories: Business, Entertainment, Music, Science, Sports, Tech, and World. I created a few fake articles mainly by using random images off the net, and by using lorem ipsum text. I created the Wordpress install under my personal web hosting account. The URL to it is <http://www.alainibrahim.com/fakenewssite>. In

the interest of time, I used the default theme that came out of the box.



## 6.3 Testing

### 6.3.1 Proof of concept

Below is the result of my having visited 3 different articles, each for a different amount of time:

category_id	category_name
8	Music
7	Entertainment
5	Tech
6	Science

In the **category** table, the categories metadata of the different articles that I visited were stored in the database. Note that if say I visit an article of the same category again, the category will not be double stored. The purpose of having this metadata is to facilitate queries for the visualization.

article_id	category_id	article_url	title	sample_text	sample_pic
16	8	http://alainibrahim.com/fakenewssite/?p=16	I like dance music	Eurodance (sometimes known as Euro-NRG, or other...)	none
10	7	http://alainibrahim.com/fakenewssite/?p=10	Testing lorem ipsum	Lorem ipsum dolor sit amet, consectetur adipiscing...	http://alainibrahim.com/fakenewssite/wp-content/up...
8	5	http://alainibrahim.com/fakenewssite/?p=8	Alain's second post	This is a paragraph This is another paragraph	none
6	6	http://alainibrahim.com/fakenewssite/?p=6	Alain's first post	This is Alain testing the first post blah bla blah	none

In the **article** table, the articles' metadata was stored. The sample text is excerpted by PHP with a default setting of 500 characters.

id	ip	time_visited	article_id	timezone	country	region	read_time
1	98.172.153.142	1409674585	16	America/New_York	United States	Maryland	6
2	98.172.153.142	1409674593	10	America/New_York	United States	Maryland	8
3	98.172.153.142	1409674603	8	America/New_York	United States	Maryland	2
4	98.172.153.142	1409674611	6	America/New_York	United States	Maryland	3

The read times appear to have registered successfully in the **hit** table.

### 6.3.2 Seeding mock data

After testing the capture of data successfully, I needed a way to mimic traffic on a real news website. In addition, I needed to make it appear as if

each mock article visit came from a random country of origin. Below are the steps I took to accomplish this:

1. Modified **constants.php** to include a flag that denotes whether the app is in test or production mode:

```
...
// defines whether we are in test or production mode
define('PRODUCTION_ENV', false);
```

2. Modified class **Hit.php** to produce a fake read time and fake US state of origin if in test mode:

```
...
// if in test environment, populate fake data for read time and country of origin
if (!PRODUCTION_ENV) {
    // get a random time between 10 seconds and 3 minutes
    $read_time = rand(10,180);

    // get the list of all the states in the US, and pick one randomly
    include("states.php");
    $region = $states[array_rand($states)];
}
...
...
```

3. Created **makeFakeHit.php** to invoke a visit to a random article on my fake news site:

```
/**
 * The sole purpose of this file is to visit a random article on the
 * fake news site, causing hits to register
 */
$TESTING_IP = "98.172.153.142";

// 290 is the id of the last fake article that was created
$article_id = rand(1,290);

// cause a fake visit only if testing ip address is registered here
if( $_SERVER['REMOTE_ADDR'] == $TESTING_IP ) {
    file_get_contents("http://alainibrahim.com/fakenewssite/?p=" . $article_id );
}
```

Note that I have hard coded a testing IP address so that I am the only person who can run this for testing.

4. Created **fakeVisit.html** to run on the client side and to call **makeFakeHit.php** at a pre-specified time interval(using Javascript):

```

(function(){
    // the amount of time to wait before revisiting the site (in seconds)
    var REVISIT_INTERVAL = 20;

    // trigger the fake visit on the server side
    var makeFakeHit = function() {
        $.get("makeFakeHit.php");
    }

    // set a recurrent loop to seed the data
    setInterval(function(){makeFakeHit();console.log("hit issued")},(REVISIT_INTERVAL*1000));
})()

```

Below is a snapshot of the results of the seeding of the database:

<b>id</b>	<b>ip</b>	<b>time_visited</b>	<b>article_id</b>	<b>timezone</b>	<b>country</b>	<b>region</b>	<b>read_time</b>
15	198.46.91.105	1412719610	242	America/Los_Angeles	United States	Wisconsin	30
14	198.46.91.105	1412719268	248	America/Los_Angeles	United States	Washington	60
13	198.46.91.105	1412719230	252	America/Los_Angeles	United States	Utah	150
12	198.46.91.105	1412719208	214	America/Los_Angeles	United States	Mississippi	146
11	198.46.91.105	1412719188	252	America/Los_Angeles	United States	Ohio	60
10	198.46.91.105	1412719129	211	America/Los_Angeles	United States	Connecticut	45
9	198.46.91.105	1412719111	109	America/Los_Angeles	United States	Vermont	17
8	198.46.91.105	1412719068	209	America/Los_Angeles	United States	Alabama	54
7	198.46.91.105	1412719049	217	America/Los_Angeles	United States	District of Columbia	60
6	198.46.91.105	1412718849	287	America/Los_Angeles	United States	Colorado	168
5	198.46.91.105	1412718788	290	America/Los_Angeles	United States	Wisconsin	34
4	198.46.91.105	1412718770	214	America/Los_Angeles	United States	Delaware	12
3	198.46.91.105	1412718631	26	America/Los_Angeles	United States	Oregon	134
2	198.46.91.105	1412718388	216	America/Los_Angeles	United States	Maine	62
1	198.46.91.105	1412718288	26	America/Los_Angeles	United States	Georgia	52

Note that for each hit, the data that is generated by the seeding process is **article\_id**, **region**, and **read\_time**. The other values were left as is because for the scope of this project, they will not be factored into the final visualization.

## 7 Implementation - retrieving captured metrics

At this point, the database is populated with a good amount of data and we need a way to retrieve it. For this, I created a folder on the server side and called it "vizLib". Herein I have thus far created 3 PHP files that will output results in JSON:

1. **getCategories.php** - outputs all the used article categories.

```
/*
     Returns the category ids and names in JSON format
*/
// the DSN for the database
define(DSN,'mysql:host=localhost;dbname=alaini5_news_viz');

// the edit username for the database
define(EDIT_USERNAME, "alaini5_newsread");

// the edit password for the database
define(EDIT_PASSWORD,"mbn@7600");

// connect to database
try {
    // create a new PDO connection to the database
    $db = new PDO(DSN,EDIT_USERNAME,EDIT_PASSWORD);
} catch(PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br />";
    die();
}

// get all categories
$sql = "select * from category";

// associative array to store categories
$categories = array();

// retrieve the id and name of each category and store it in the array
foreach($db->query($sql) as $row) {
    $categories[$row["category_id"]] = $row["category_name"];
}

// close database connection
$db = null;

// encode the array in json format
$categories = json_encode($categories);

print $categories;
```

2. **getArticlesMetrics.php** - outputs all of the metrics for all the articles given a category ID, beginning visit time, and ending visit time.

```

/**
     Returns the following metadata for all articles for
     a given time period from a given category in JSON:
     - Time visited
     - Article ID
     - Article URL
     - Article title
     - Article sample text
     - Article featured image URL
     - Country of visitor
     - Read time of article
 */

// the DSN for the database
define(DSN,'mysql:host=localhost;dbname=alaini5_news_viz');

// the edit username for the database
define(EDIT_USERNAME, "alaini5_newsread");

// the edit password for the database
define(EDIT_PASSWORD,"mbn@7600");

// connect to database
try {
    // create a new PDO connection to the database
    $db = new PDO(DSN,EDIT_USERNAME,EDIT_PASSWORD);
} catch(PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br />";
    die();
}

// associate array to contain all the articles' metadata
$articles = array();

// get the category id from query string
$category_id = $_GET["category_id"];

// get the start visit time of the articles for this category in Unix time
$start_time = $_GET["start_time"];

// get the end visit time of the articles for this category in Unix time
$end_time = $_GET["end_time"];

// get all relevant metadata for the articles visited
// within our specified category and time period
$sql = sprintf("select hit.id as 'hit_id',time_visited , hit.article_id ,
               country , read_time , article.article_id , category_id ,
               article_url , title , sample_text,sample_pic from article ,hit
               where hit.article_id=article.article_id and
               category_id=%d and time_visited >= %d and time_visited <= %d",
               $category_id,$start_time,$end_time);

// populate the $aritcle array with all the metadata
foreach($db->query($sql) as $row) {
    $articles[$row["hit_id"]]["article_id"] = $row["article_id"];
    $articles[$row["hit_id"]]["url"] = $row["article_url"];
    $articles[$row["hit_id"]]["title"] = $row["title"];
}

```

```

        $articles[$row["hit_id"]]["sample_text"] = $row["sample_text"];
        $articles[$row["hit_id"]]["pic"] = $row["sample_pic"];
        $articles[$row["hit_id"]]["country"] = $row["country"];
        $articles[$row["hit_id"]]["read_time"] = $row["read_time"];
    }

// close database connection
$db = null;

// encode final results in JSON format
$articles = json_encode($articles);

print $articles;

```

3. **getArticleCountriesMetrics.php** - outputs all of the country metrics for a given article ID.

```

/**
 * Returns the number of hits and what country they
 * came from for a given article id in JSON format
 */

// the DSN for the database
define(DSN,'mysql:host=localhost;dbname=alaini5_news_viz');

// the edit username for the database
define(EDIT_USERNAME, "alaini5_newsread");

// the edit password for the database
define(EDIT_PASSWORD,"mbn@7600");

// connect to database
try {
    // create a new PDO connection to the database
    $db = new PDO(DSN,EDIT_USERNAME,EDIT_PASSWORD);
} catch(PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br />";
    die();
}

// associative array to store the article hits
// and country of origin given an article
$article_countries = array();

// get the article id from query string
$article_id = $_GET["article_id"];

// get the sum of hits for a given article given their country of origin
$sql = sprintf("select distinct country, count(hit.id)
                as 'hits' from hit where hit.article_id=%d
                group by country", $article_id);

// retrieve the number of hits and country name
foreach($db->query($sql) as $row) {
    $article_countries[$row["country"]]["hits"] = $row["hits"];
    $article_countries[$row["country"]]["name"] = $row["country"];
}

```

```

}

// close database connection
$db = null;

// encode the array in json format
$article_countries = json_encode($article_countries);

print $article_countries;

```

I anticipate the **vizLib** folder to also carry all code pertinent to the visualization which I will build in the final stage.

## 8 Implementation - visualization

For the visualization, I enabled the following tasks for the users:

- To compare articles' hits and reader attention spans for any period within the last month, given a select category
- To preview each article's thumbnail, title, and excerpt
- To contrast the geographic distribution of hits and attention of up to 2 articles
- To navigate to the full article counterpart on the subject news website

### 8.1 The controls box

Typically, controls take up space in the limited real estate area available in any web browser. I decided to have the controls be draggable and minimizeable so they don't negatively impact the user experience.



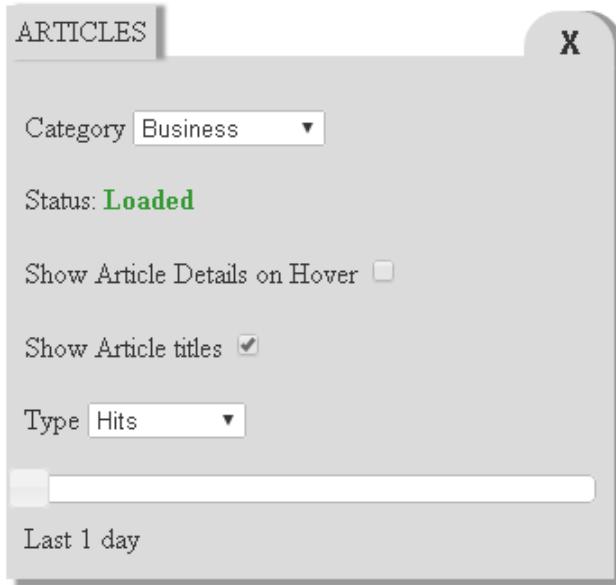
The controls box first partially loads when no category is selected. The "X" hides it; when this is done, the top left gray rectangle with the two white dots is then contoured by a black border - to indicate state change.

Upon loading the visualization, an AJAX call is made to the server which fetches all the news categories into this dropdown box. Once a category is selected, a call is made to the server, which results in all of the data for the selected category to be fetched from the database using the server-side script. The user is notified (via the red colored "Loading..." indicator) of when the data is being loaded:

[...]



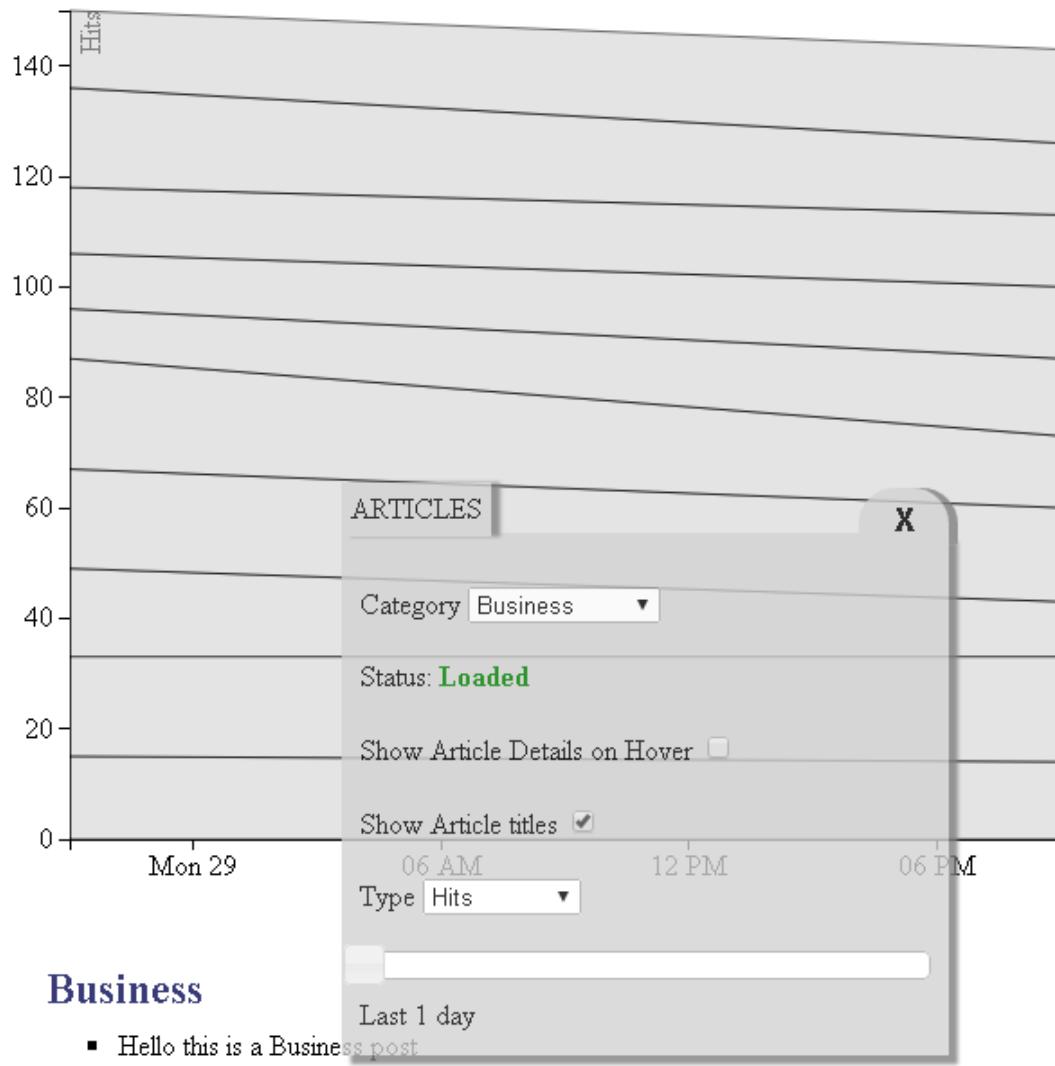
And, once the data is loaded, the status label turns green and reads "Loaded":



Below are the descriptions for each component of the controls area:

- **Category** - sets the category whose articles the user chooses to see
- **Status** - shows the data transfer status of the selected news category
- **Show Article Details on Hover** - determines whether a preview of the article will pop up upon hovering over an article in the stacked area chart.
- **Show Article titles** - toggles the visibility of the article titles in the bottom portion of the screen
- **Type** - the user gets to select between viewing the number of hits or amount of time spent (in seconds) on the Y axis of the stacked area chart
- **Time slider** - This is a dual time slider control in that the user can use it to select one slice (i.e. day) in time or can alternatively select a time range to view the data in

I made the controls box transparent in case the user wanted to keep it afloat the visualization:



## Business

- Hello this is a Business post
- Dollars VS People?
- Top Obama Officials Disagree Over Whether U.S. At War With ISIS
- \$3 Per Gallon Gas, Here We Come
- Are you putting food on your table?
- Madoff Son Andrew Wills More Than \$15 Million to His Fiancée and Family
- Do you have a profile with ISIS?
- PhD dropout? Lucrative Silicon Valley career
- Facebook Tests Disappearing Posts Feature
- August PS4 and Xbox One Sales Continue to Break Records

## 8.2 Stacked area chart

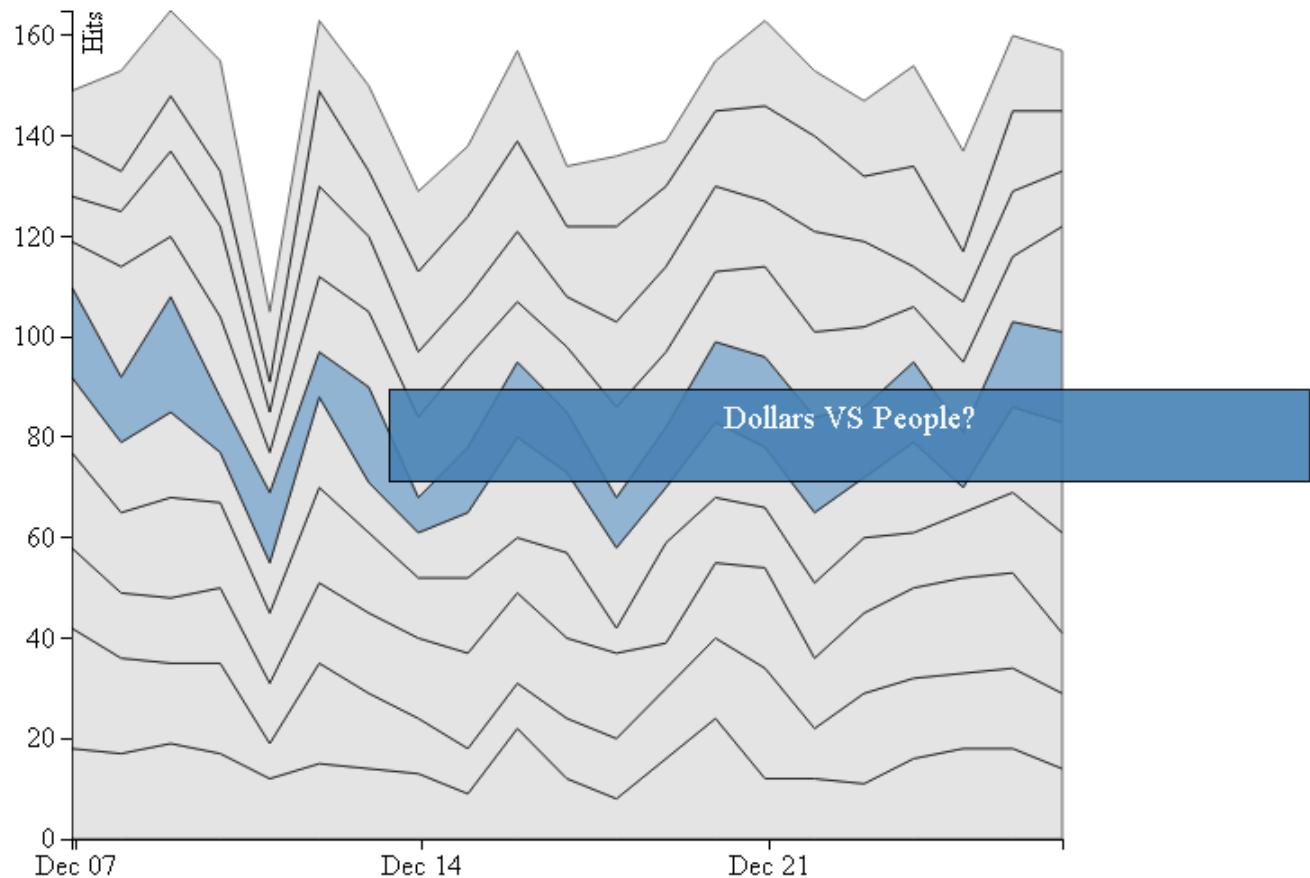
Here, the idea was to create a visual product that would allow the user to view all the articles' popularity over time, given a certain news category. I knew that I wanted time to go on the horizontal Y axis, and (hits or read time) to go on the Y axis. That said, I first started with a simple line graph, where each line represented an article that was colored differently from the other lines. There were 2 downsides which led me to abandon the line graph:

- The lines intersected into a spaghetti mash-up, making it very difficult to spot any significant patterns
- When a category (e.g. Business) contained more than say 10 articles, I had to use 10 different colors to encode the articles. The issue here was that at a certain point, colors became very similar making it almost impossible to visually query the article all the way through a selected time period

I instead used the stacked area chart. The stacked area chart maintains a view on the patterns without compromising legibility and supports the following tasks:

- To compare articles' hits and reader attention spans for any period within the last month, given a select category
- To preview an article's thumbnail, title, and excerpt

The X axis is laid out using D3 and represents a time aggregation that occurs on an hourly basis.

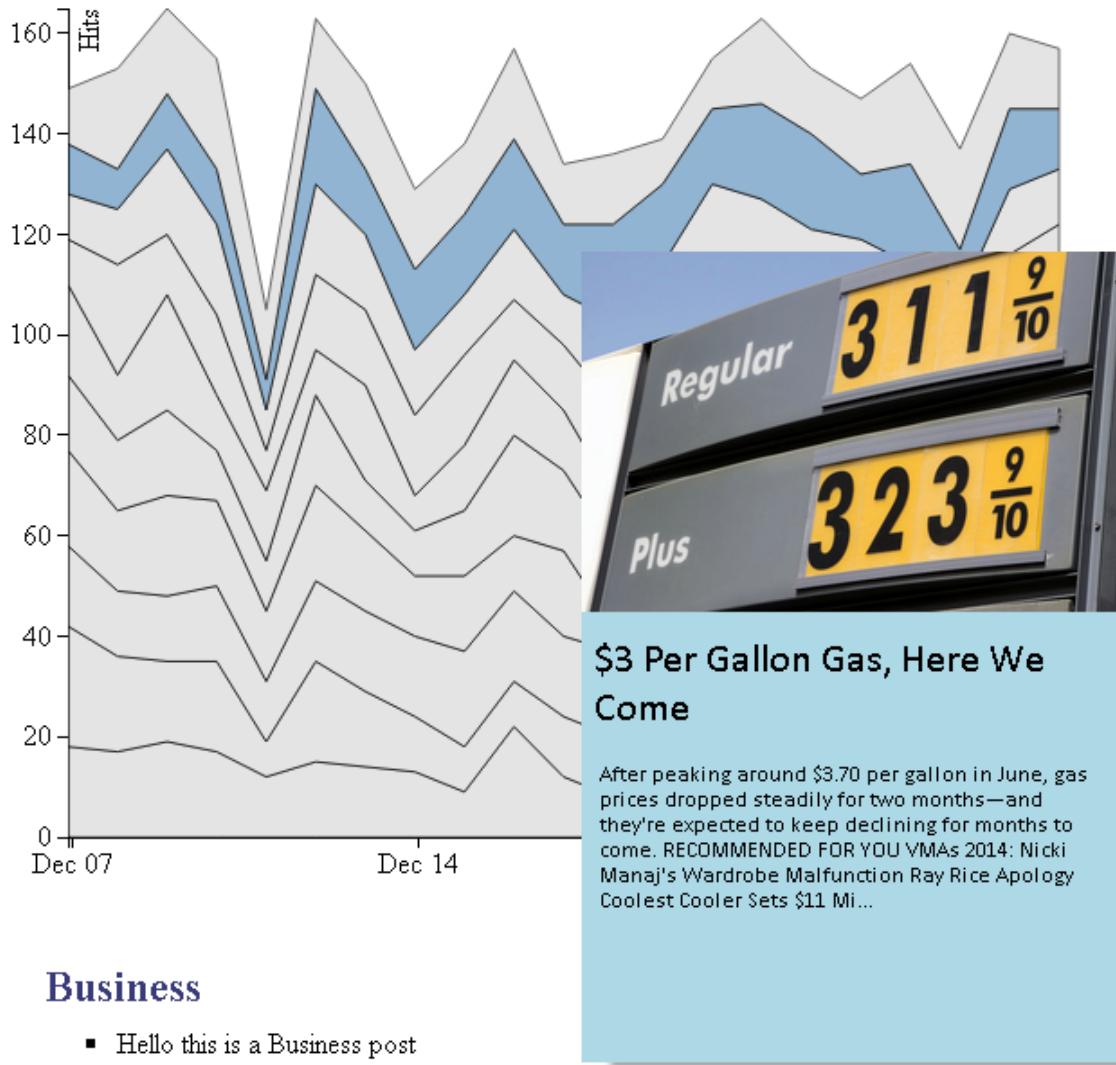


## Business

- Hello this is a Business post
- \$3 Per Gallon Gas, Here We Come
- August PS4 and Xbox One Sales Continue to Break Records
- Facebook Tests Disappearing Posts Feature
- **Dollars VS People?**
- Top Obama Officials Disagree Over Whether U.S. At War With ISIS
- Madoff Son Andrew Wills More Than \$15 Million to His Fiancée and Family
- Do you have a profile with ISIS?
- PhD dropout? Lucrative Silicon Valley career
- Are you putting food on your table?

The articles are stacked vertically, each respectively matching its title in the below white space area, which is headed by the selected category. The readying of the data for the stack is done using the D3 area layout, and the visual representation is implemented by using the SVG <path>. The paths by default all contain a fill of light gray.

Hovering over any article in the stack chart causes both the article's area and corresponding title to turn to steel blue. In reverse, hovering over an article title in the bottom area causes the corresponding article's area in the stack chart to have a fill color of steel blue. It is important to note that the user has the option to see a preview of the selected article when checking "Show Article Details On Hover" in the controls box:

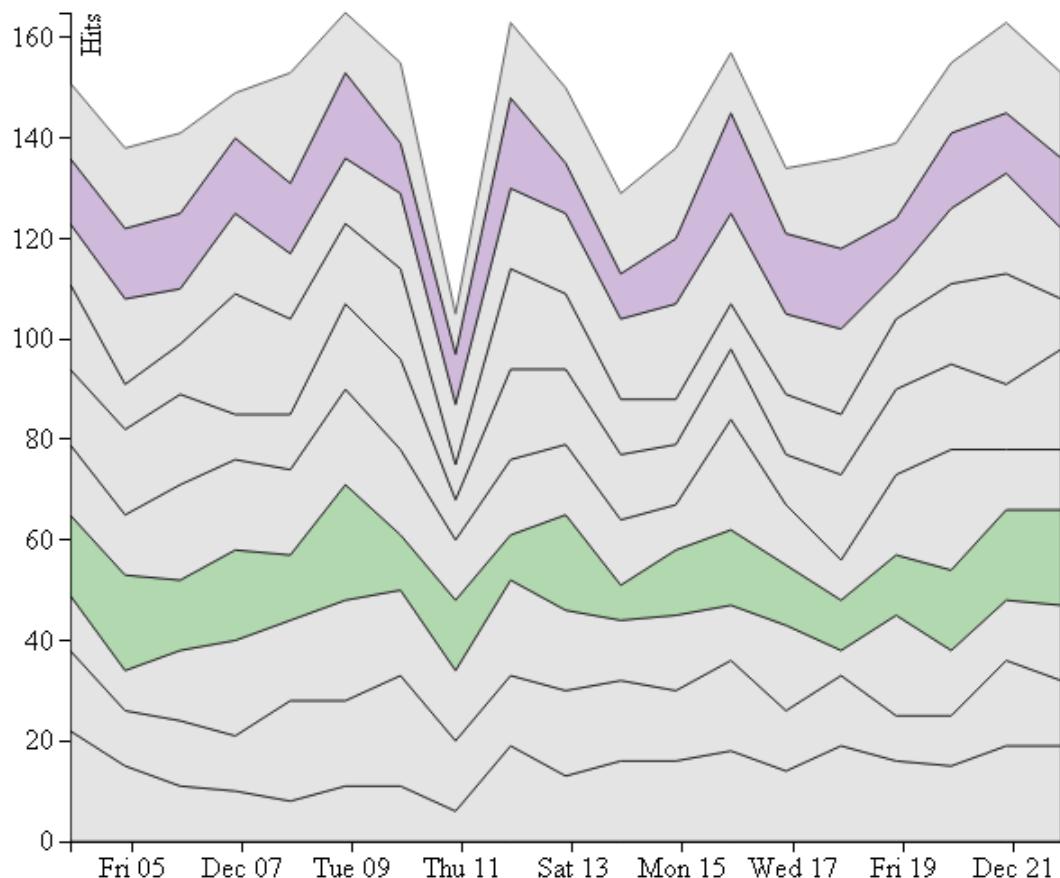


## Business

- Hello this is a Business post
- [\*\*\\$3 Per Gallon Gas, Here We Come\*\*](#)
- August PS4 and Xbox One Sales Continue to Break Records
- Facebook Tests Disappearing Posts Feature
- Dollars VS People?
- Top Obama Officials Disagree Over Whether U.S. At War With ISIS
- Madoff Son Andrew Wills More Than \$15 Million to His Fiancée and Family
- Do you have a profile with ISIS?
- PhD dropout? Lucrative Silicon Valley career
- Are you putting food on your table?

This results in an overlay to pop over the selected article that includes the article's thumbnail, title, and a 500-character excerpt of the body of the article.

The user is given the option to contrast the pattern of up 2 articles over time, by left clicking on the select article area. This results in one article being filled with a purple color and the other with a green color. I picked the colors from colorbrewer2.org and ensured they were colorblind safe.



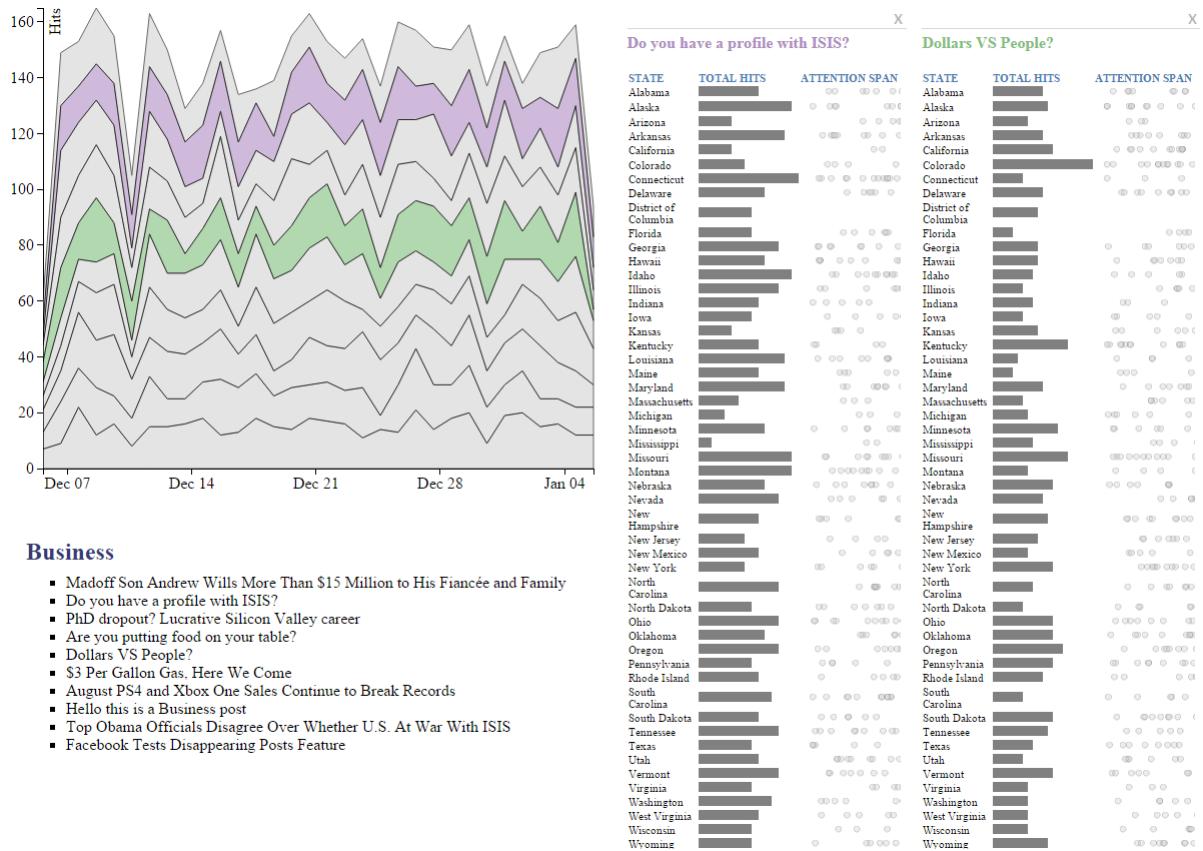
## Business

- Facebook Tests Disappearing Posts Feature
- Top Obama Officials Disagree Over Whether U.S. At War With ISIS
- Do you have a profile with ISIS?
- PhD dropout? Lucrative Silicon Valley career
- August PS4 and Xbox One Sales Continue to Break Records
- Are you putting food on your table?
- Dollars VS People?
- Madoff Son Andrew Wills More Than \$15 Million to His Fiancée and Family
- Hello this is a Business post
- \$3 Per Gallon Gas, Here We Come

Once the two articles are selected, their corresponding geographic distributions are shown in the right area of the browser. This is covered in the next section.

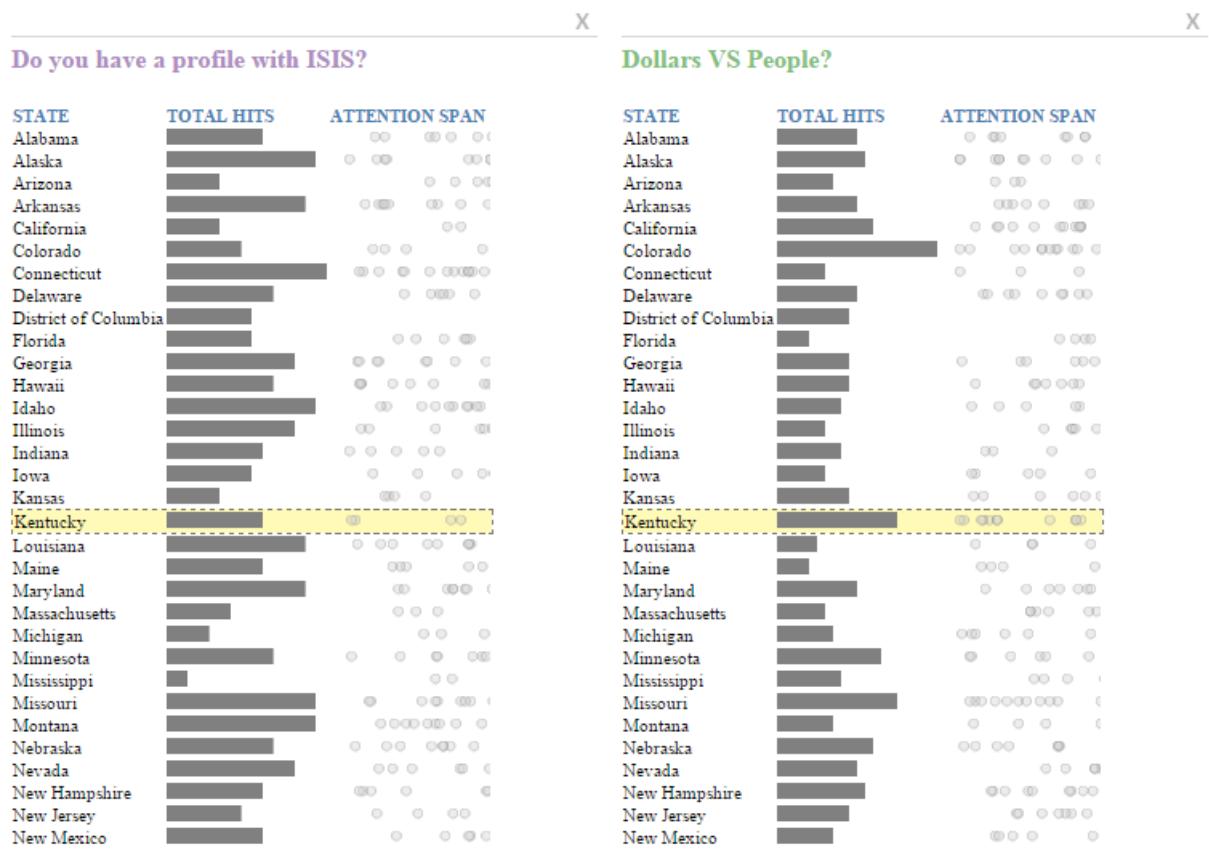
## 8.3 Geographic bar charts

In the previous section, two articles are clicked on in the stacked area chart. In addition to their being highlighted in the stacked area chart, a right side pane populates each article's title in its color, along with a table below it. The table contains the geographic origin (e.g. state/country), the total number of hits, and the attention span distribution (i.e. read time in seconds).



Whenever a first article is selected, the second article is compared along the same geographic areas for uniformity. That is, only the geographic areas that were included in the first article will be displayed for the sec-

ond article. Clicking the top right "X" removes the article from the right pane, and renders the graph area counterpart back to light gray. The hits values are encoded as horizontal bars in dark gray. The attention span column contains a distribution of the captured read times, represented as light gray circles; the amount of seconds spent is encoded by the horizontal position of each circle. The circles are transparent to allow for overlap. The maximum horizontal position of the circle is demarcated by the expected read time. The expected number of seconds was discussed earlier, and is a measure that takes into account the average adult human reading speed and the number of words contained within the article. The total hits bar spans relative to the highest hits value for that article - meaning, the bar that takes up the biggest width is the highest hit value that was registered for that article.



We now examine the geographic area in more detail. Note that automatically, the entries are sorted by the name of the geographic origin, in ascending order. Clicking on the "STATE" header will toggle back and forth the sorting order of the geographic origin name. Similarly, toggling the "TOTAL HITS" header will sort the data in ascending/descending order. When the data is sorted, it is sorted according to the same criterion for both articles. Hovering over a row will cause that row to be highlighted in yellow, and to be contoured in a dashed line solid border. Hovering over any bar will in addition pop up a steel blue overlay which displays - using white text - the number of hits or read time in seconds.

The geographic bar charts support the following tasks:

- To contrast the geographic distribution of hits and attention of up to 2 articles
- To navigate to the full article counterpart on the subject news website

## 9 Results

### 9.1 Mock Website

The working example that I have discussed so far is based on a mock website that I created using mock visit data.

### 9.1.1 Capturing Data

1. For each hit, the following metadata was successfully captured: IP address, time visited, article ID, time zone, country of origin, and region of origin
2. For each hit, the read time was captured as a random number between 10 and 180 seconds. When I switched the flag in my PHP code to denote the production environment, the read time was captured successfully when I navigated away from a news article. Most of my testing was done in Google Chrome version 39
3. Each hit was captured according to the revisit threshold time (in seconds) that was specified in the PHP code
4. Each category was captured from Wordpress via the PHP libraries I created
5. Each article's metadata was captured from Wordpress via the PHP libraries I created

### 9.1.2 Retrieving Data

1. The categories were successfully retrieved in JSON via **getCategories.php**
2. The hits data was retrieved successfully from the MySQL database using JSON via **getArticlesMetrics.php**. I initially reserved the geographic data for the articles to be serviced by **getArticleStatesMetrics.php**. However, I felt that in a production environment that many SQL calls to the database server would slow down the system and affect all users. In order to minimize the number of requests to the server, I had the server fetch all data for a given category for the last 31 days. The client-side Javascript code then parsed and re-organized the fetched data to be used by the interactive visualization

### 9.1.3 The Interactive Visualization

1. The stacked chart accurately reflected the number of hits and average read time over the selected time range

2. Clicking one or more articles highlighted the article encode in the stacked area chart and displayed the total hits based on geography. It also showed the distribution of the recorded read times. For the mock site, the geography was limited to states within the United States
3. The tables that contained the geographic origins were sortable by total hits and by state name - in ascending and descending order
4. When "Show Article Details on Hover" was enabled, the pertinent article's thumbnail image, title, and body excerpt were successfully presented in an overlay
5. Hovering over any row in the geographic hits and attention span table resulted in the row being highlighted in yellow for all articles that are being
6. Hovering over a bar representation of the total number of hits invoked a steel blue overlay showing the number of hits. Hovering over an attention span circle invoked a steel blue overlay showing the number of seconds that was spent on the article.

#### **9.1.4 Performance**

1. In the mock site, the last 31 days amounted to 4500 hits. The time it took for them to be loaded in the visualization was around 4 seconds, using a PC with an Intel i7 processor and 12 GB of RAM
2. When changing the date range, the stacked chart's redraw time was almost instantaneous. Again, using a PC with an Intel i7 processor and 12 GB of RAM
3. When "Show Article Details on Hover" was enabled, there was a delay of about 500 ms for the image thumbnails to initially show. Once they were cached in the browser's memory, they appeared without delay. For images that were not cached, the delay was about 1 second per megabyte in picture size

## 9.2 Actual Website

## 10 Future Work

## 11 Glossary

**Apache HTTP Server (aka Apache):** A web server software program

**Choropleth map:** A thematic map in which areas are shaded or patterned in proportion to the measurement of the statistical variable being displayed on the map (source: Wikipedia).

**Class:** In object oriented programming parlance, it refers to the abstraction and representation of a real life object or concept in code.

**CSS:** A styling language used to define the aesthetics of elements in a web deliverable. This is implemented in all major web browsers. CSS3 is the most recent version of this language.

**D3.js:** A JavaScript library for manipulating documents based on data using HTML, SVG and CSS (source: d3js.org).

**Google Analytics:** A service offered by Google that generates detailed statistics about a website's traffic and traffic sources and measures conversions and sales (source: Wikipedia).

**HTML 5:** The most recent specification for the markup language that constitutes the visual elements of any web deliverable. It also normally includes some JavaScript and CSS 3 code as part of its implementation.

**JavaScript:** A client-side scripting language that is used by all prominent web browsers.

**JSON (JavaScript Object Notation):** A lightweight data-interchange format (source: json.org).

**LAMP:** A web development environment comprised of a Linux server, Apache HTTP Server, MySQL, and PHP.

**MySQL:** An open source relational database management system.

**Object oriented programming:** A programming paradigm that represents real-life elements and concepts as "objects". The implementation involves mimicking only the needed real-life characteristics of the object in code, and calling a working copy of these objects an "instance".

**PHP:** An open source server-side scripting language.

**Relational database:** A database built on principles of the relational model. Such a database is comprised of tables and fields.

**SVG:** An XML-based vector image format for two-dimensional graphics that has support for interactivity and animation (source: Wikipedia).

**Web visualization:** A visual deliverable created using web technologies. More than often, it is dynamic in that it visually changes based on the data being fed to it.

## References

- [1] Schumann, H. and Máluller, W. (2000). *Visualisierung – Grundlagen und allgemeine Methoden*. Springer, Berlin, Germany.
- [2] Edward R. Tufte, *Visual Display of Quantitative Information* (2001).
- [3] Munzner, T. (2009), *A Nested Model for Visualization Design and Validation*.
- [4] A. Cockburn, A. Karlson, and B. B. Bederson, “A review of overview+detail, zooming, and focus+context interfaces,” ACM Computing Surveys (CSUR), vol. 41, no. 1, pp. 1–31, 2008.
- [5] Playfair, W. and Corry, J. (1786). *The Commercial and Political Atlas: Representing, by Means of Stained Copper-Plate Charts, the Progress of the Commerce, Revenues, Expenditure and Debts of England during the Whole of the Eighteenth Century*. printed for J. Debrett; G. G. and J. Robinson; J. Sewell; the engraver, S. J. Neele; W. Creech and C. Elliot, Edinburgh; and L. White, London, UK.
- [6] Frank, A. U. (1998). *Different Types of Times in GIS*. In Egenhofer, M. J. and Golledge, R. G., editors, *Spatial and Temporal Reasoning in Geographic Information Systems*, pages 40–62. Oxford University Press, New York, NY, USA.
- [7] Goralwalla, I. A., O茲su, M. T., and Szafron, D. (1998). *An Object-Oriented Framework for Temporal Data Models*. In Etzion, O. et al., editors, *Temporal Databases: Research and Practice*, pages 1–35. Springer, Berlin, Germany.
- [8] Bettini, C., Jajodia, S., and Wang, X. S. (2000). *Time Granularities in Databases, Data Mining, and Temporal Reasoning*. Springer, Secaucus, NJ, USA, 1st edition.
- [9] Radner, W., Diendorfer, G., (2014). *English sentence optotypes for measuring reading acuity and speed — the English version of the Radner Reading Charts*