WAYFAR.US TECHNICAL REPORT

Assignment 4 : *AJAX & JIT Updates and Wrap-Up*

By Colin To, 301101928

DELIVERABLES

Core Deliverables [Completed]

1. **JIT (just-in-time) updates** from linked services. The linked service includes text, images and timestamps from twitter.
2. **AJAX (asynchronous) updates** with PHP code and Javascript. AJAX must send a request to a separate PHP script.
3. **Create XML** by serializing a DOM Object using DOM API or similar method. Also retrieve tweets from this XML.
4. **3 Portfolio Description Pages** about the website, author and technical details for this website.
5. **Visitor & Blog Counter** to keep track of how many people and blogs are on the website at once.

Previous Deliverables - Missed from Previous Assignments [Completed]

6. **FlickrID Accounts** – Members can now enter Flickr account info into forms that will update within the database. Additional members can change Flickr accounts anytime in the *settings.php* page. (Flickr accounts not being used at the moment).
7. **Visitor Feature** – The visitor feature has been added. You can now browse without signing in by clicking on the Wayfarus logo.

Modifications [Completed]

8. **Limit Visitors –** For Visitors, hide irrelevant features such as following dashboard, settings page and ability to follow members.
9. **Hide TwitterFeed –** For Visitor and for member profiles that do not have a twitter account.
10. **Account Update Settings –** Update account information without entering all information about the account.
11. **Following Dashboard to Member Profiles –** I made it possible for people to click on the members that a user is following to go directly to their profiles, rather than having to go to the "*members.php*" page first.

FUTURE MODIFICATIONS

Errors [Needs Modifying]

1. **TwitterFeed TimeOut –** Sometimes *w-tweetAPI.php* will not retrieve account data from Twitter. Waiting solves this issue.
2. **Too Many Visitors –** A bug reading excessive echo statements from AJAX retrievals because of echo save to "Visitors.xml"

Personal Modifications

3. **Create Blog Post Feature** – Create the ability for members to post blogs by either clicking on the map, or clicking on an add blog tool on the side of the main map interface. This will be taken care of with "d3.js".
4. **Modify Blog Templates –** Put them into a single template, which will make them easier for users to create in the future.
5. **AJAX Follow Feature** – Using AJAX to update the information within the dashboard after user clicks on stars to follow members.
6. **Interactive Map** – Using "d3.js" for semantic zooms, pans and navigations.

From Assignment 3

12. **Cache Flickr URL Data –** By storing them into a new table. This will increase the speed at which the images will load.
13. **Mailing Updates –** For users to configure any updates they want to receive. Configure how frequently they should receive it (Daily, Weekly or Monthly).

Wayfarus

[Design and Implementation]

Wayfarus is a web application designed which encourages people to explore new areas around the globe and acquire phenomenological experiences and tacit knowledge of both celebrated and reclusive locations. These locations can be areas which their friends have already visited and have blogged about or blogs about locations that users have personally visited or areas that users are highly interested in visiting.

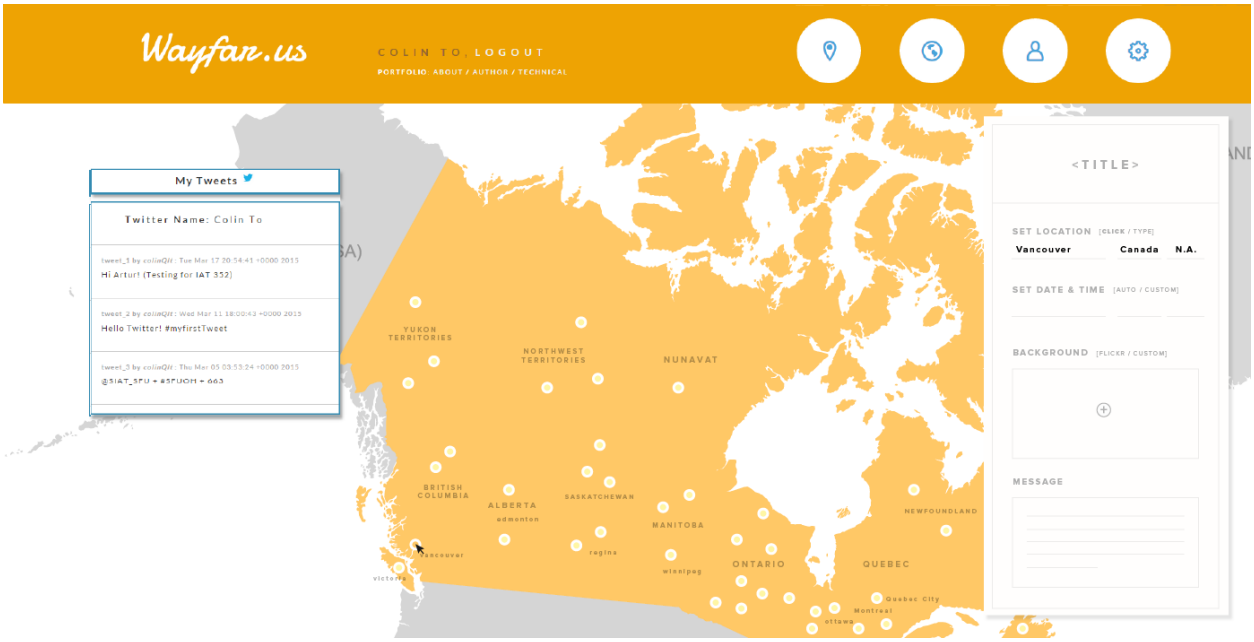
Technologies & Future Implementations: Through the use of some advanced web development techniques and protocols such as AJAX, the website will deliver real-time media for visualizing and experiencing virtual travels to different areas around the world in the form of blog posts, videos, atmospheric audio, image galleries, and other forms of multimedia. The technologies that is used on this website is currently in its infancy because the core focus of this website was to develop the back-end web technologies which included relational database design and the facilitation of querying information. In the future, d3.js, JQuery, React.js, Angular.js and any other form of Javascript would be used to extend functionality from current front-end web design and development.

MAIN FEATURES

INTERACTIVE MAP – [For Future Implementation]

The core features of this web application would involve the ability to interact with an interactive map through various combinations of controls with the mouse. Visitors would be able to explore locations around the globe by clicking on regions around the map in which people have posted blogs. The idea uses a combination of “Google Maps” and “Blogging Features”.

This would be achieved using “d3.js”, a Javascript library that allows interaction with a map using APIs such as kml. The interactive map would be implemented in the future as Javascript was not a core focus of this class and attention was needed more for PHP and back-end development. The full mockup prototype for the interactive map is also attached at the end of this document and can be viewed there in detail.

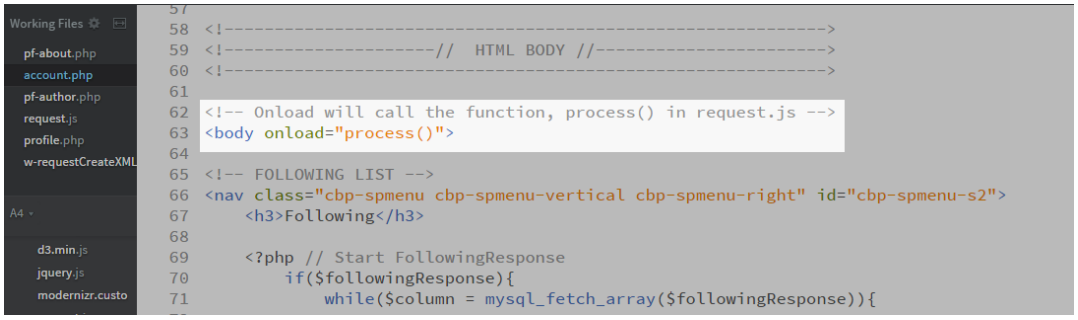


Prototype: Interactive Map in the future, using d3.js for visualizing Blog Post feature

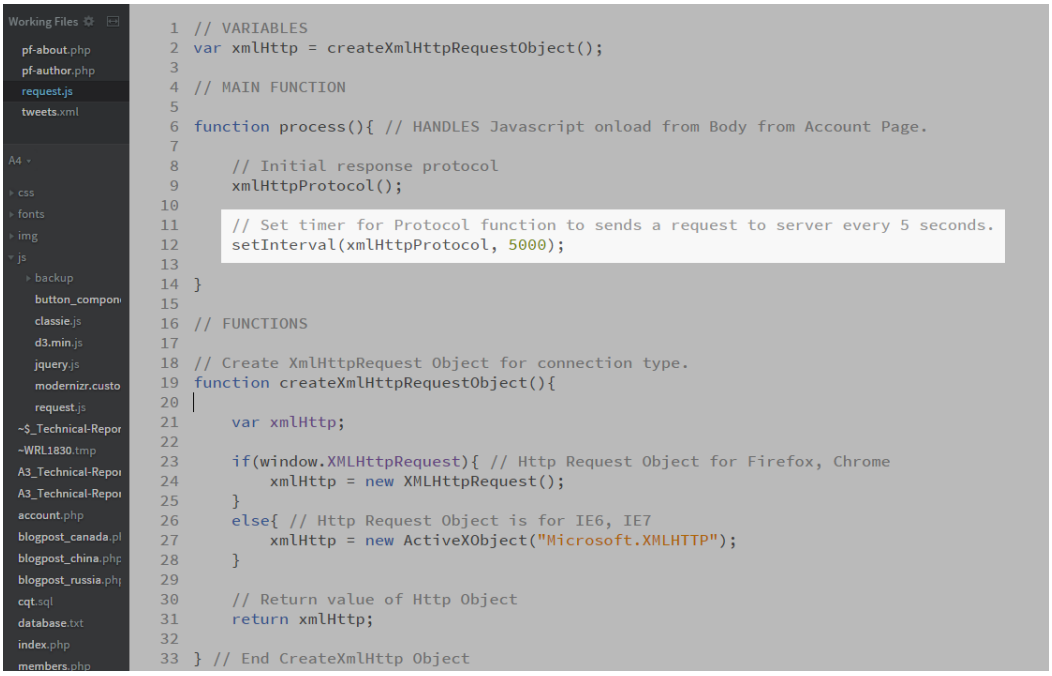
CORE DELIVERABLES

1. JIT UPDATES – *From Linked Services*

JIT updates are currently regulated by a function called setInterval, which is called when users access the account.php web page and the body of the DOM object activates the process() function within request.js. Each of these updates are set to an interval timer with the setInterval function, executing the xmlHttpProtocol request every 5 seconds, emulating real-time or just in time updates to the Twitter Feed.

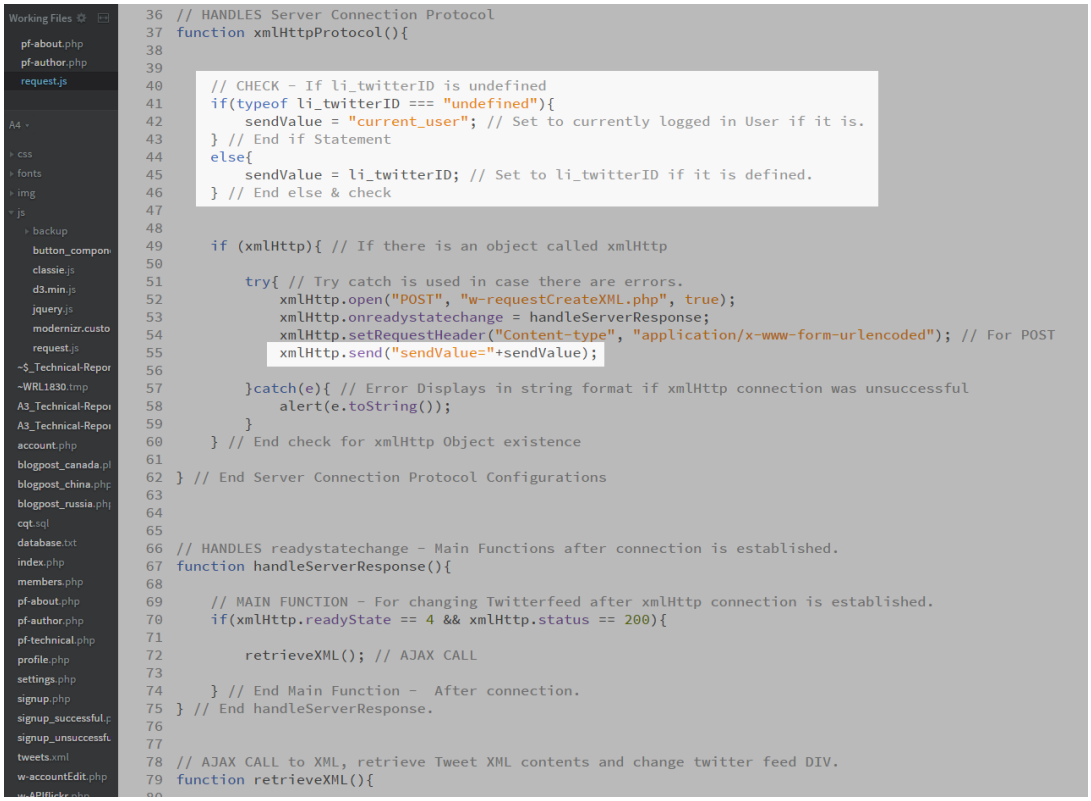


account.php – Process() is called from request.js in order to perform JIT updates with intervals



request.js – Using setInterval method, we are able to perform Ajax call with xmlHttpProtocol every 5 seconds

All of the code for creating the xmlHttpProtocol object to send AJAX requests follows with standard procedures such as creative different objects for different browser such as Chrome, Firefox and Internet Explorer.



request.js – Check if a twitterID is undefined. If it is, set default twitterFeed to current user's twitter account. Otherwise set it to twitterID

In request.js, there is a check to see if the li_twitterID is undefined using Javascript. If it is undefined, then it would set it would be set to the default value of the string current_user so that the twitterID would be tied to the current user's Twitter account. This executes when the current user is viewing the Account page. Otherwise, if the current user is viewing the Profile page, it would send a different value that is equivalent to li_twitterID, which is the Twitter account of the designated member that the current user is viewing information about on the Profile page. The send value is also using the xmlHttp protocol to perform AJAX, but the alternative is to use AJAX in order to perform the AJAX call.

When handling the xmlHttpProtocol object, it would also hand the response that is coming from the server as well, checking the connection is established with a readyState = 4 and that the status was successful with status being equal to 200.

2. AJAX UPDATES – *Asynchronous Updates with JS & PHP*

The AJAX updates are produced in request.js and gets accessed from different web page such as account.php and profile.php. There are more setup procedures that require forms to be processed beforehand for assigning POST variables prior to accessing the request.js file. Such examples include w-profile.php for assigning li_username and index.php for assigning username as visitors or member usernames. The request.js becomes the middleman which sends AJAX requests to either w-requestCreateXML.php or w-requestCreateVisitorXML.php based on the PHP file that was process prior to accessing request.js.

Using these files, I was able to achieve asynchronous updates. If there would be any need for anymore asynchronous updates in the future, I would just need to create more AJAX request objects with either xmlHttp or Jquery and make different requests to send to the *w-requestCreateXml.php* file in order to create additional xml files and retrieve new data from there.

I have chosen to use GET here in these AJAX requests because it is more responsive than POST. We are not dealing with sensitive information and the two-step process of POST would cause the twitter feed to be slightly less responsive.

Here are the steps for setting up an AJAX call for retrieving data or for sending values and data to another file.

1. User interacts with a web page that links to a page that requires AJAX.
2. A middle tier PHP processes this interaction and prepares and assigns POST or SESSION variables to be used later.
3. The variables get passed to functions within request.js and used for assigning variables.
4. Request.js makes AJAX calls to different PHP files such as w-requestCreateXML.php and w-requestCreateVisitorXML.php
5. w-requestCreateXML.php and w-requestVisitorCreateXML saves XML to corresponding XML documents, tweets.xml and visitors.xml.
6. Request.js retrieves information from w-requestCreateXML.php, which retrieves information from tweets.xml. The same procedure would be implemented when making AJAX calls to w-requestCreateVisitorXML.php.
7. Request.js executes an action within its success function and performs the desired action in the designated PHP web page.

```
// AJAX CALL to XML, retrieve Tweet XML contents and change twitter feed DIV.
function retrieveXML(){

            // Variable for dynamic twitterfeed DIV. Change Id to my twitter ID instead later on.
            twitterfeed = document.getElementById('twitterfeed');

            try{

                $.ajax({
                    url: 'tweets.xml',
                    type: 'GET',
                    dataType: 'xml',
                    success: function(data){

                        // Resets the list added to the feed.
                        $('#twitterfeed ul').empty();
//                        $('#twitterfeed ul').html('');

//                        $(data).find('tweet[username=colinQlt]').each(function(){
                        $(data).find('tweet').each(function(){

                            // Set XML content variables from XML file
                            var username = $(this).attr('username');
                            var id = $(this).attr('id');
                            var text = $(this).find('text').text();
                            var date = $(this).find('date').text();

                            // Append with variables to DIVS
                            $('#twitterfeed ul').append(

                                "<li class='tweet_list'>"
                                + "<div class = tweet_author><b>" + id + " by <i>" + username + "</i> : </b>"+date+"</div>"
                                + "<div class = tweet_content>" + text + "</div>" +
                                "</li><br/><hr/>"
                            ); // End Append List

                        }); // End Find tweets
                    }, // End function for Success

                    error: function(){
                        console.log("The Ajax call was unsuccessful");
                    }

                }); // End AJAX call
            } // End try function
```

request.js – Ajax retrieves information from tweets.xml to append tweets to #twitterfeed ul within account.php or profile.php

Within the AJAX request, it initially sets the HTML content of Twitter Feed div ul to be empty so that the request will not append cumulative information into the Twitter Feed. The AJAX call first finds all the tags called tweet within tweets.xml and then assigns variables that equal the text nodes of those tags. Those text nodes will eventually be appended to twitterfeed's ul for displaying on the current web page.

```
<?php // TWITTER FEED - Show if Signed in User is not visitor.

    if ($_SESSION['username'] != 'visitor'){

    require_once 'w-APItweet.php'; // Get information from tweets.

        // DISPLAY TwitterFeed - Only if TwitterAccount is not 0 (or undefined).
        if ($twitterAccount != "0"){

            // TWITTER TOGGLE
            echo '<section onclick="twitterToggle()" style="cursor:pointer; width:20%; height:40px;
                                    border:3px solid #338bb2; color:#333;
                                    position:absolute;
                                    margin-left:7%; margin-top:0px;
                                    box-shadow: 4px 4px 5px #aaaaaa;">
                <div width=100%; style="display:block; cursor:pointer; text-align:center; font-weight:bold; letter-spacing:0.1em">
                My Tweets <img src="img/icons/twitter.png" style="margin-top:2px;" height="20px"></div>
                </section>';

            // TWITTER STREAM
            echo'<section id="twitter" class="twitter" style="width:20%;
                                    border:2px solid #338bb2; color:#333;
                                    position:absolute;
                                    margin-left:7%; margin-top:50px; margin-bottom:30px;
                                    box-shadow: 4px 4px 5px #aaaaaa; font-size:12px;">';

            echo'<div style="height:50px; padding:18px 50px;
                font-size:14px; letter-spacing: 0.2em; display:block; margin: 0 auto; width:100%;">
                <b> Twitter Name: </b>' .$twt_name. '</div>';

            echo'<hr>';
            echo '<div id="twitterfeed"><ul style="margin:0px;"></ul></div>';
            echo '</section>';
        }
    }
?>
```

account.php – The twitterfeed Div that the Ajax information is sent to in account page

```
<?php   // TWITTER STREAM - Only Show if TwitterID is set or is not 0.
if (isset($_POST['li_twitterID']) && $_POST['li_twitterID'] != '0'){
        echo '<div id="twitter" class="twitter" style="width:20%;
                                border:2px solid #338bb2; color:#333;
                                position:absolute;
                                margin-left:60%; margin-top:0px; margin-bottom:30px;
                                font-size:12px;">';

        echo '<div style="height:50px; padding:18px 50px; padding-bottom:30px;
        font-size:14px; letter-spacing: 0.1em; display:block; margin: 0 auto; width:100%; text-transform:none; font-weight:normal">
        <b> Twitter Feed: </b>' .$twt_name. ' <img src="img/icons/twitter.png" style="margin-top:2px;" height="20px"/></div>';

        echo '<hr>';

        echo '<div id="twitterfeed"><ul style="margin:0px;"></ul></div>';
    }
?>
```

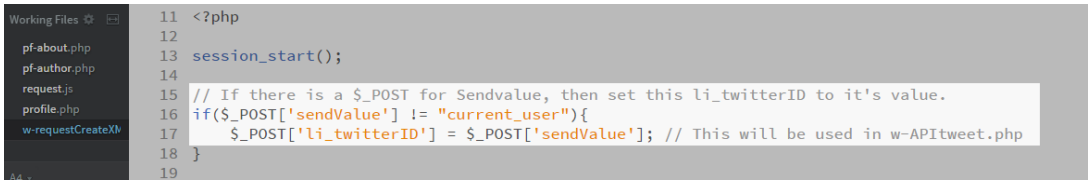
profile.php – The twitterfeed Div that the Ajax information is sent to in profile page

The ul within the div with the id called twitterfeed is the HTML tag that is responsible for handling dynamic information retrieved from tweets.xml and displays it either the Account page or Profile page depending on which web page the current user is viewing.

3. CREATE XML – *To Retrieve Tweets from*

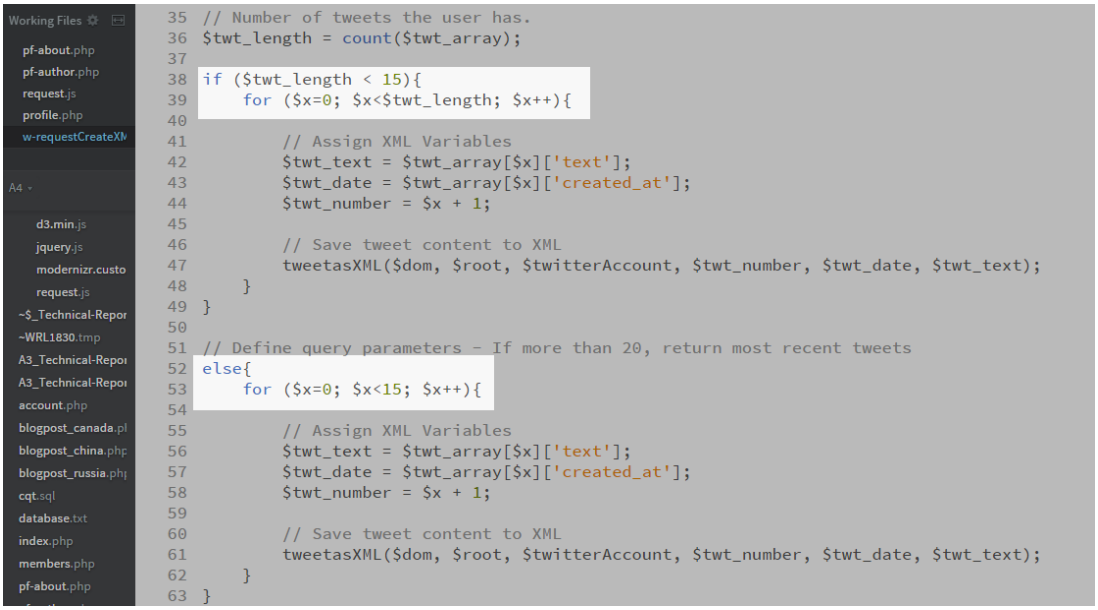
There are currently two PHP files within this project that is responsible for creating XML files, however only one of the PHP file, w-requestCreateXML.php is currently in use. Due to functionality problems the other PHP file for creating XML file is temporarily disable and is also not needed as a core deliverable for this iteration of the project. The w-requestCreateXML is called from request.js within the xmlHttpProtocol function and creates the a new DOM model for inserting information into tweets.xml.

First, the PHP file checks if the sendValue is equal to the string, current_user. If not, it sets the value to li_twitterID. This code is needed for checking if the information passed to the requestCreateXML is used for displaying tweets to the Account page or the Profile page. Li_twitterID would be used for display tweets to the Twitter Feed on Profile page and current_user is used for display tweets on the Account page.



```
Working Files
pf-about.php
pf-author.php
request.js
profile.php
w-requestCreateXM

<?php

session_start();

// If there is a $_POST for Sendvalue, then set this li_twitterID to it's value.
if($_POST['sendValue'] != "current_user"){
    $_POST['li_twitterID'] = $_POST['sendValue']; // This will be used in w-APItweet.php
}
```

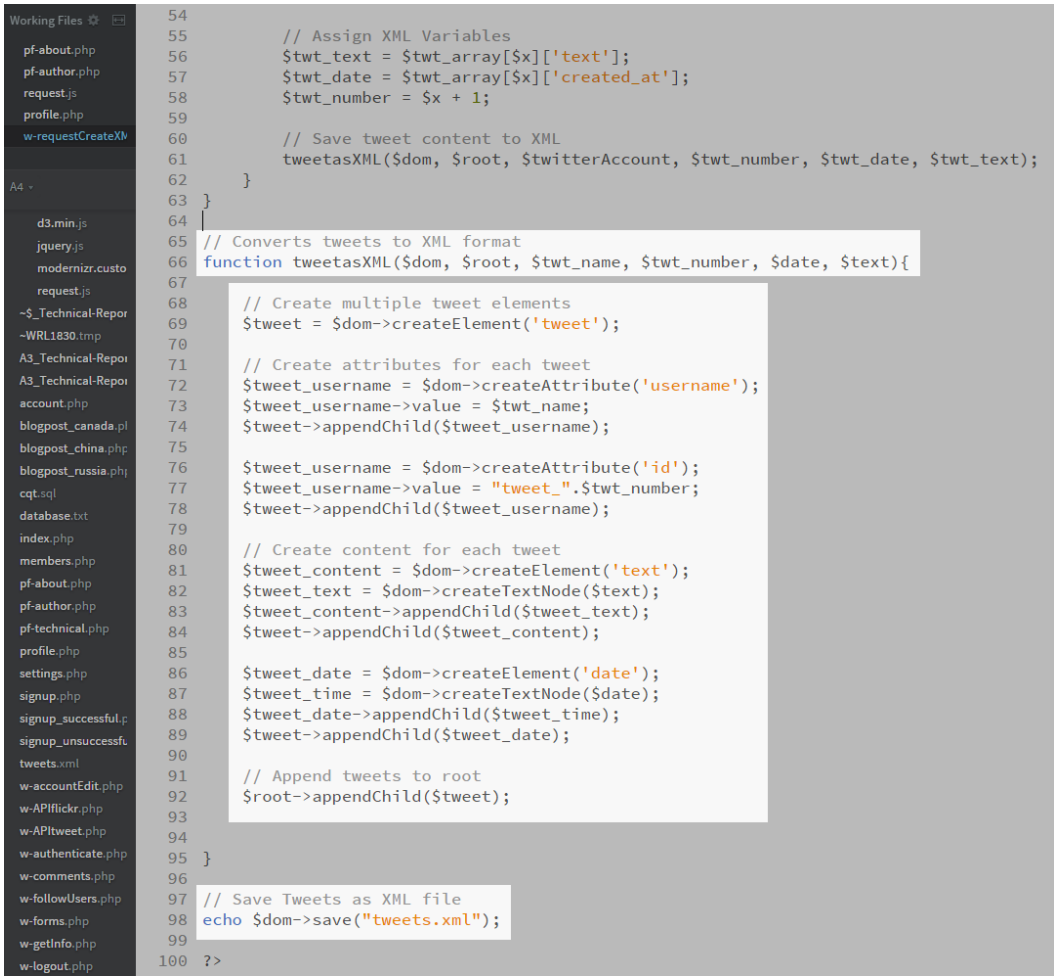
w-requestCreateXML.php – Check if the POST information is for account or profile page. If it is for profile, then set li_twitterID to its value



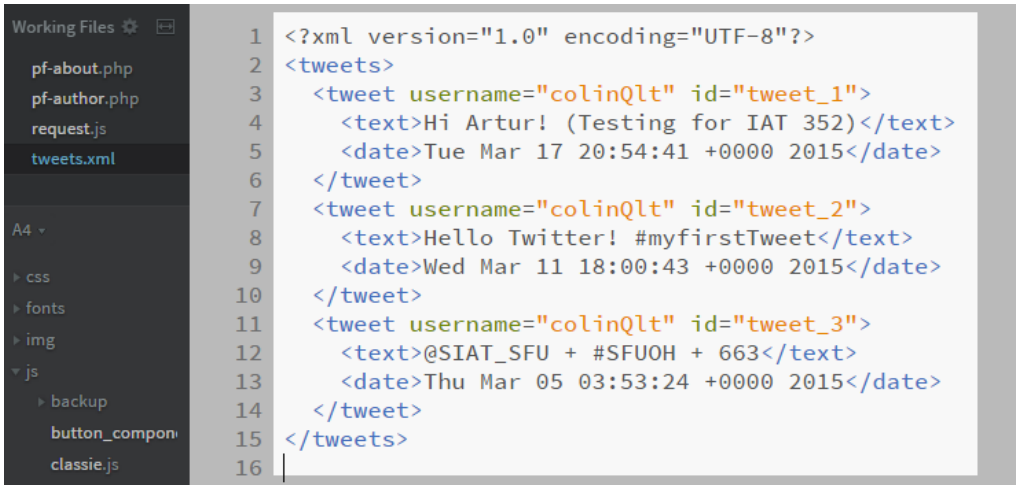
```
Working Files
pf-about.php
pf-author.php
request.js
profile.php
w-requestCreateXM
A4
d3.min.js
jquery.js
modernizr.custo
request.js
~$_Technical-Repor
~WRL1830.tmp
A3_Technical-Repor
A3_Technical-Repor
account.php
blogpost_canada.pl
blogpost_china.php
blogpost_russia.php
cqt.sql
database.txt
index.php
members.php
pf-about.php

// Number of tweets the user has.
$twt_length = count($twt_array);

if ($twt_length < 15){
    for ($x=0; $x<$twt_length; $x++){

        // Assign XML Variables
        $twt_text = $twt_array[$x]['text'];
        $twt_date = $twt_array[$x]['created_at'];
        $twt_number = $x + 1;

        // Save tweet content to XML
        tweetasXML($dom, $root, $twitterAccount, $twt_number, $twt_date, $twt_text);
    }
}

// Define query parameters - If more than 20, return most recent tweets
else{
    for ($x=0; $x<15; $x++){

        // Assign XML Variables
        $twt_text = $twt_array[$x]['text'];
        $twt_date = $twt_array[$x]['created_at'];
        $twt_number = $x + 1;

        // Save tweet content to XML
        tweetasXML($dom, $root, $twitterAccount, $twt_number, $twt_date, $twt_text);
    }
}
```

w-requestCreateXML.php – Limit the amount of tweets in Twitter Feed to 15. If it is less, than display the appropriate amount of tweets

The next step for the w-requestCreateXML.php file is to format the tweets that appear in the Twitter in relation to how many tweets are currently present on a designated username's Twitter account. If there are more than 15 tweets, than set the tweet amount to 15. If the tweet amount is less than 15, then display the appropriate number of tweets. The tweet information will also be passed into parameters which will be used to create the XML file.



w-requestCreateXML.php – Save as tweets.xml with root, element and text nodes, getting information from parameters



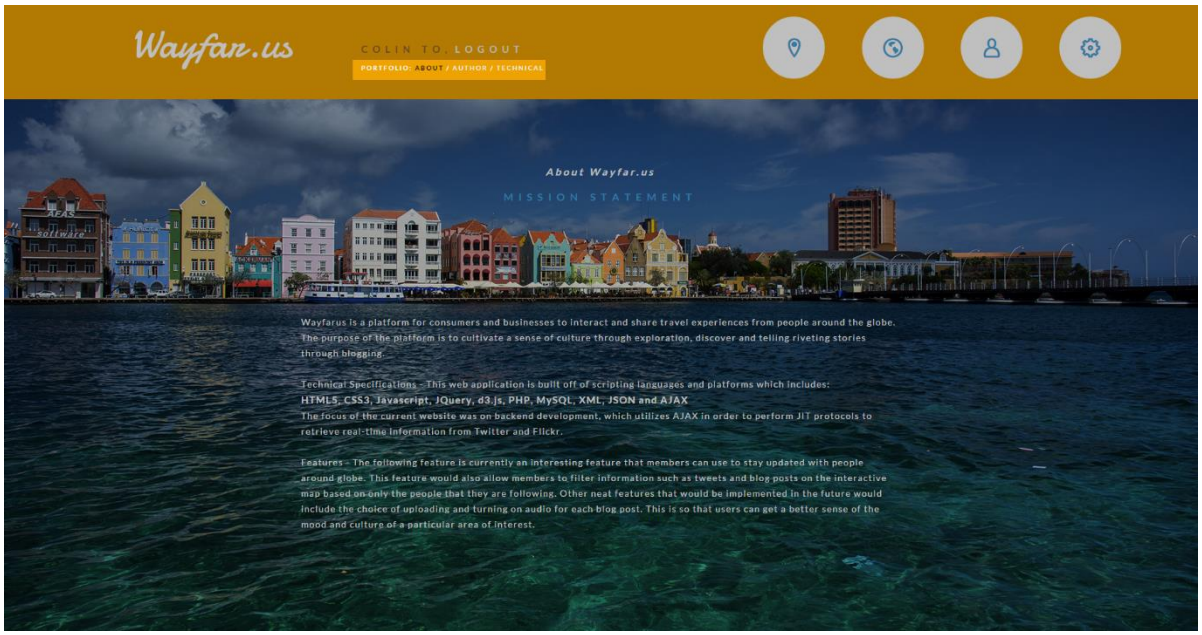
Tweets.xml – The xml file result, created dynamically with Ajax

The last step in w-requestCreateXML.php would be to create the actual DOM object to be used to append nodes to and save as an XML document. The tweets.xml gets updated from these saves dynamically and would be later used with AJAX in order to retrieve information from specific tags or nodes.

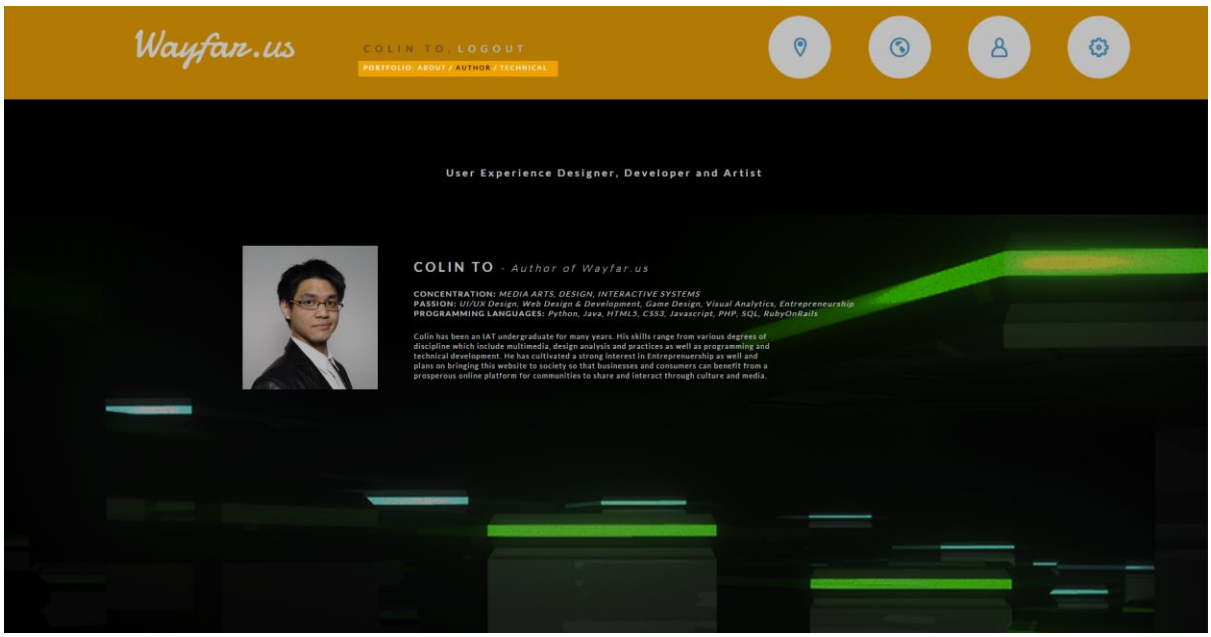
4. PORTFOLIO – *Description Pages for "About, Author & Technical" Details*

For the portfolio pages, I have included a description of summaries that provide a brief overview for each category of detail. The code for these portfolio description web pages are relatively simple and require only the use of mainly HTML and CSS code, rather than the implementation of Javascript or PHP. However, the code for these web pages are included in the submission folder if the code needs to be assessed.

About Page: This web page includes details on the purpose of this web application as well as the web technologies that were needed to build this web application. Details on current and prospective interactive features that users can perform are also included.



Web Page: pf-about.php – Information about the website



Web Page: pf-author.php – Information about the author of the website

Author Page: The author page provides a quick summary of my personal interests and prior design and programming experiences. It also provides details about my current studies in Interactive Arts & Technology as well as a brief overview of my engagement with the school and the direction I may be taking with this project.

Technical Page: There is also a technical page that users can navigate to within the same navigation bar. The technical page shows a few selected projects that demonstrate my technical skills as a programmer when facing adversity and learning new programming languages within a relatively short amount of time. Additionally, Wayfarus would be added to this list once it is completed. I have also decided not follow the recommendation of having the technical page link from the about page because it is easy for users to navigate to this page from any web page.

5. VISITOR & BLOG COUNTER – *Using Multiple Session*

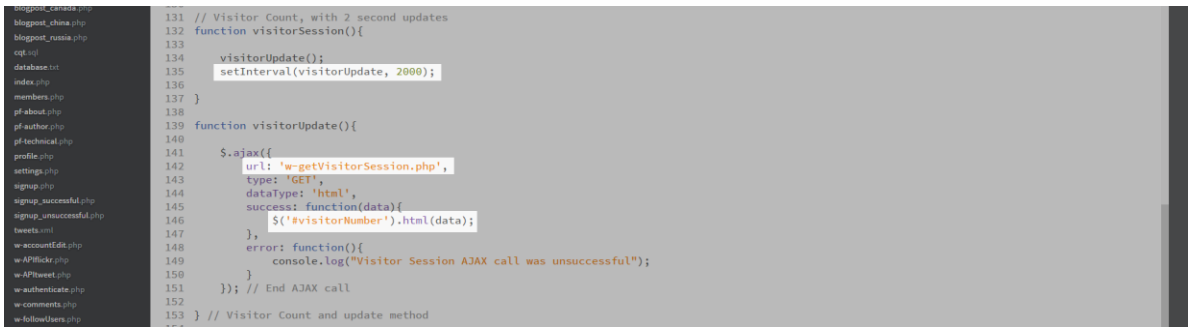
When the user visits the Home page or the Profile page and clicks on the Wayfarus logo it will send them to the w-authenticate_visitor.php file. When the PHP file gets processed, the visitor count gets incremented by 1 for visualizing later on in the Account page.

index.php – First, click on Wayfarus logo to sign in as a visitor, which processes w-authenticate_visitor.php form

w-authenticate_visitor.php – Sets the username to visitor and then increments visitorCount by 1.

When the document is loaded from the Account page, the visitorSession gets called from the request.js file. This happens after the visitorCount div is created so that the visitorSession function has a viable div to select to change its innerHTML content. The AJAX call selects the visitorNumber within the visitorCount div and changes the html content in accordance to the data that is received from the URL, w-getVisitorSession.php.

account.php – Calls visitorSession function in request.js when DOM loads. Ajax will update visitorNumber span



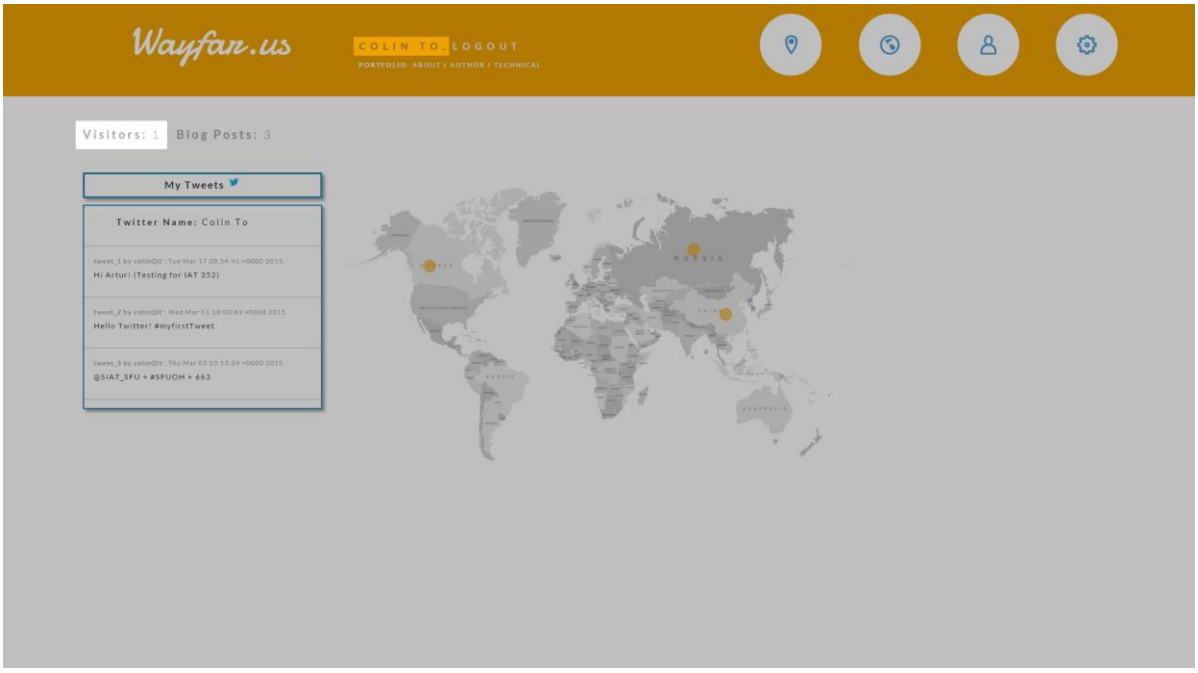
request.js – Ajax updates visitorNumber every 2 seconds and replaces it with data from w-getVisitorSession.php

```php
<?php
session_start();


// Change visitors to -1 whenever they log out.
if(isset($_SESSION['visitorCount']) && $_SESSION['visitorCount'] > 0){
    $_SESSION['visitorCount'] -= 1;

    // Call w-getVisitorSession to update visitor amount in webCounts.xml
    require_once('w-getVisitorSession.php');
}


// Unsets & Destroy all $_SESSION variables
unset($_SESSION['username']);

// Destroy All Session Variables
//session_destroy();

// Redirect back to login page after logging out.
header("Location: index.php");

?>
```

w-logout.php – Decrement visitorCount by 1 when visitors logout. Update the number by calling w-getVisitorSession.php once more

While visitors are logging out, w-logout.php decreases the total of visitors that is current logged in. I have chosen to use unset instead of destroy session variables or unset all variables because this will reset the visitorCount back to 0. Unsetting the current session's username variable is also required because it is easier to handle multiple visitor and member logins simultaneously while keeping track of visitor counts. This would also allow members to resign in as another member or log in as a visitor.

The best solution that I can provide without having time to research in detail, would be to create an array of username values and store them into a single username session variable. The server would check if the user that logged out matches the user that is currently browsing on a particular tab session. This would avoid any conflicts when visitors or members with similar values log in, requiring the system to rely on associative values for better authentication protocols instead of relying on the binary value of set or unset variables.



Web Page: account.php – Visitor Counter can be seen by members too

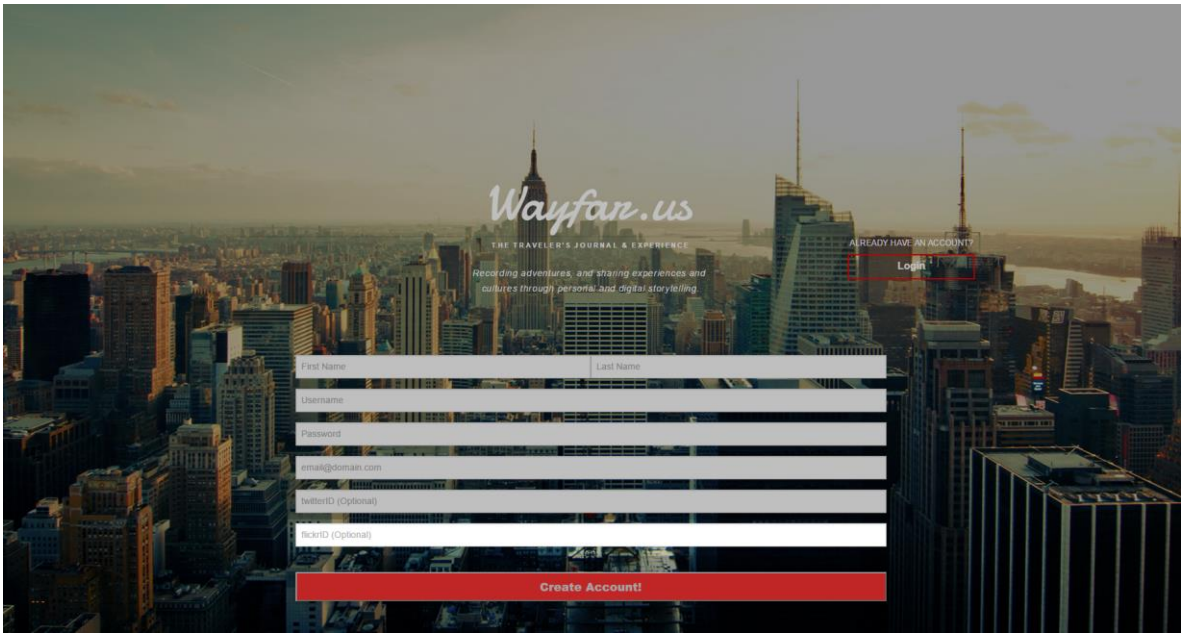
The visitor count visualization is applicable to any user that has logged in, regardless of whether they are a member of this web application or if they are just visitors. There are currently some issues with the visitor count that is being stored, which will be discussed later in this document.

The blog post counter is currently hard coded, but will be modified to retrieve the amount of blog posts within the web application after the blog post feature becomes available.

PREVIOUS DELIVERABLES

6. FLICKRID ACCOUNTS – *For registering FlickrID accounts*

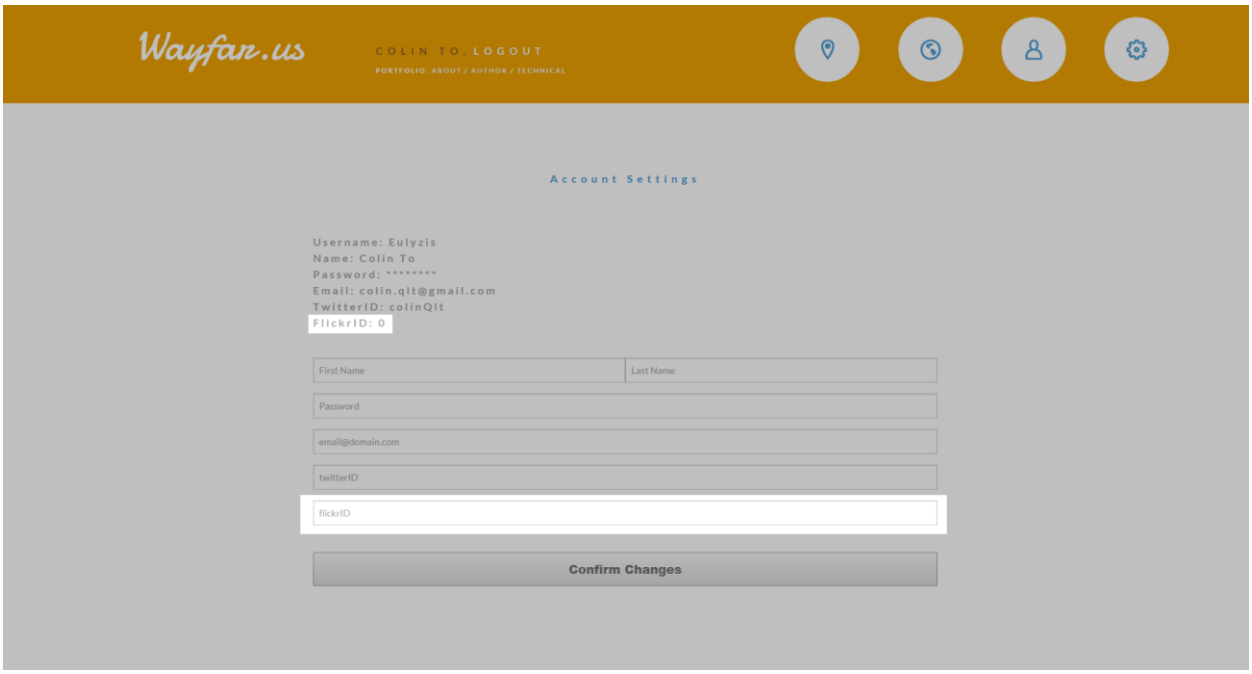
For the deliverables in assignment 3, there was a necessity for users to have the option for putting information about their own Flickr account and storing them into the database. With the new forms and corresponding code below each screenshot, it illustrates how I've implemented these new additions to the web application.



Web Page: signup.php – Flickr Form is now present during signup


```
80
81              <!-- SIGN-UP -->
82              <section style ="width:50%; margin: 0 auto;">
83                  <div class="c2-form new_firstname" id="fname">
84                  <input type="text" name="firstname" placeholder="First Name" required/>
85                  </div> <!-- end firstname -->
86
87                  <div class="c2-form new_lastname" id="lname">
88                  <input type="text" name="lastname" placeholder="Last Name"/>
89                  </div> <!-- end lastname -->
90              </section>
91
92              <div class="cform new_username">
93              <input type="text" name="username" placeholder="Username" required/>
94              </div> <!-- end username form -->
95
96              <div class="cform new_password">
97              <input type="text" name="password" placeholder="Password" required/>
98              </div> <!-- end password form -->
99
100
101             <div class="cform new_email">
102             <input type="email" name="emailAddress" placeholder="email@domain.com" required/>
103             </div> <!-- end email form -->
104
105             <div class="cform twitter_account">
106             <input type="text" name="twitterAccount" placeholder="twitterID (Optional)"/>
107             </div> <!-- end twitter form -->
108
109             <div class="cform flickr_account">
110             <input type="text" name="flickrAccount" placeholder="flickrID (Optional)"/>
111             </div> <!-- end flickr form -->
112
113             <br/>
114             <div><a href="account.php">
115             <input class="submit cta-button" type="submit" id="return" value="Create Account!"/></a>
116             </div>
117             <!-- end SIGN-UP -->
118
119         </form>
120     </section>
```

signup.php – Flickr Form is now present during signup and is also optional. 0 will be the default input for empty Flickr forms



Web Page: settings.php – Flickr Accounts can now be updated in Account Settings

settings.php – Flickr Form is now also present in settings.php so that members can update it

Members can also register or update a new flickr account into the web application's system even after signing up. The provisions of Flickr accounts are optional and a default value of 0 will be assigned to members that have chosen to leave the Flickr field blank during signup.

The Flickr account information that is being stored in the database is not being used for anything at the moment because there is currently no functionality within that web application that needs to utilize this information. In the future, the Flickr account can be used to display an array of images on member profile pages or added to blog accounts as either background images or slideshows that will appear at the top the their blog post web pages. Users would either have the option of having Flickr provide image slideshows based on the location of the blog post or customize the slideshow to their own liking from personal Flickr accounts.

7. VISITOR FEATURE – *For accessing the web application without logging in*

For the deliverables in assignment 2, there was a necessity for having a visitor capability that lets users browse the website without logging in. I have implemented this feature by converting the main logo into a linkable image, granting access to the website without having to use a preexisting account. Being visiting the website as a visitor, there would be limited interactions that the user can perform on the web application.

Web Page: index.php – Click on logo to "visit" the website

index.php – The code that makes this possible. Process w-authenticate_visitor.php before going to account.php

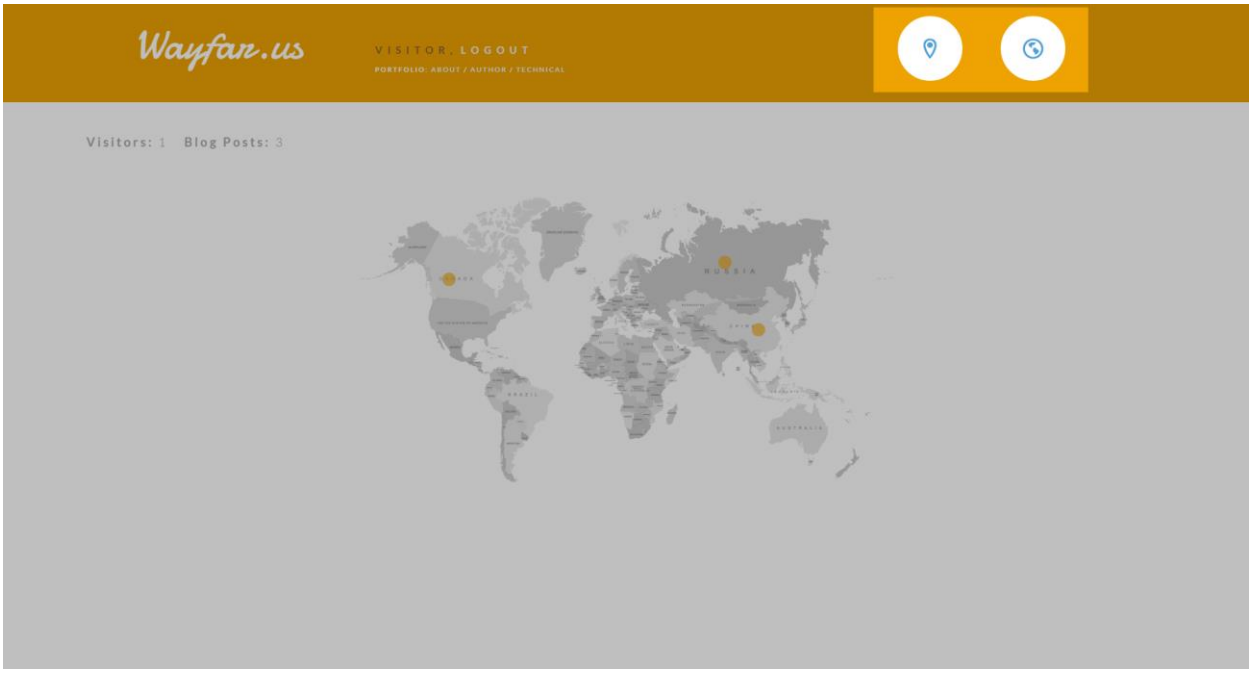
There is a new authentication process that is specific only for visitors and sets certain sessions and protocols different for current members that log in. The authentication process occurs in the file, w-authenticate_visitor.php.

MODIFICATIONS

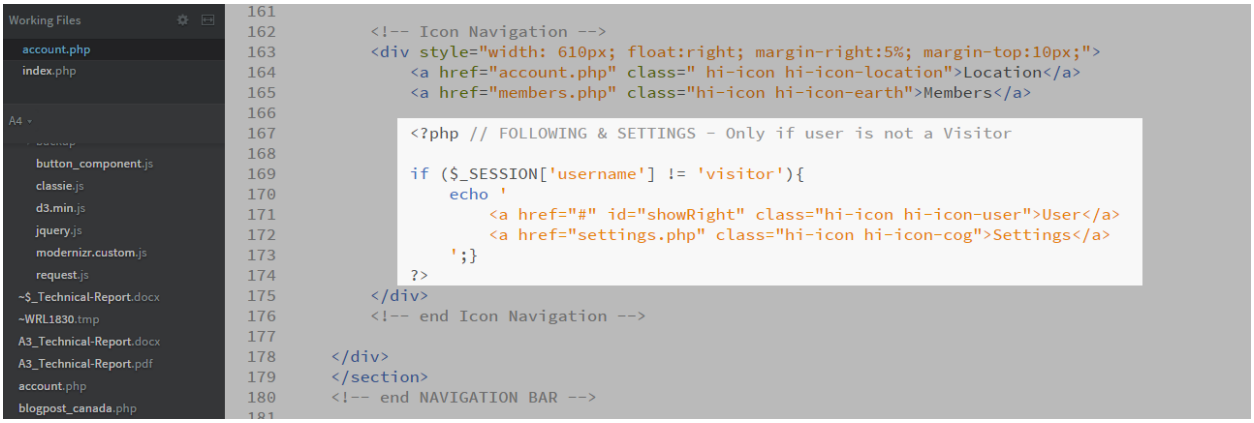
8. LIMIT VISITORS – *Restrict visitors from accessing features that should are reserved for members*

With the added capabilities for users to become visitors on this web application, it was also necessary to hide certain features that visitors would not need to have access to. The two main icon navigations that are ordinarily present for members of the web application would be hidden to visitors as the remaining two would still be accessible. The hidden navigation icons are the Account Settings page and the Following Dashboard.

Visitors would not have any account information and so it makes little sense for them to be able to access the account settings web page. They would also not need to access to the Following Dashboard because they should not be able to follow members until they register for the website and become a member. The two options that are still available to visitors would be the Account page and the Members page. Visitors are still able to see the list of members currently registered for the website and can also see their profiles. However the stars beside each member will not be present because the features for Following members are disabled for visitors.



Web Page: account.php – Following & Account Setting Icon are hidden to visitors



account.php – Only show the Following & Account Settings Icons for members, not visitors

9. HIDE TWITTERFEED – *Hide twitter feed for visitors or members that does not have a twitter account*

Visitors and members without a twitter account would not be able to see a Twitter Feed when the log into the account page. Twitter Feeds are also hidden on profiles of members that also do not have twitter accounts. The code that is provided below shows that there is a protocol that checks for the existence of a twitterID or if the user is a visitor before executing the code to create the Twitter Feed and corresponding tweets. The protocol occurs slightly differently for the Account page and for the Profile page because they require checking different variables.

```
<?php // TWITTER FEED - Show if Signed in User is not visitor.
    if ($_SESSION['username'] != 'visitor'){
    require_once 'w-APItweet.php'; // Get information from tweets.
        // DISPLAY TwitterFeed - Only if TwitterAccount is not 0 (or undefined).
        if ($twitterAccount != "0"){
            // TWITTER TOGGLE
            echo '<section onclick="twitterToggle()" style="cursor:pointer; width:20%; height:40px;
                                    border:3px solid #338bb2; color:#333;
                                    position:absolute;
                                    margin-left:7%; margin-top:0px;
                                    box-shadow: 4px 4px 5px #aaaaaa;">
                <div width=100%; style="display:block; cursor:pointer; text-align:center; font-weight:bold; letter-spacing:0.1em">
                My Tweets <img src="img/icons/twitter.png" style="margin-top:2px;" height="20px"></div>
                </section>';
            // TWITTER STREAM
            echo'<section id="twitter" class="twitter" style="width:20%;
                                    border:2px solid #338bb2; color:#333;
                                    position:absolute;
                                    margin-left:7%; margin-top:50px; margin-bottom:30px;
                                    box-shadow: 4px 4px 5px #aaaaaa; font-size:12px;">';
            echo'<div style="height:50px; padding:18px 50px;
                font-size:14px; letter-spacing: 0.2em; display:block; margin: 0 auto; width:100%;">
                <b> Twitter Name: </b>' .$twt_name. '</div>';
            echo'<hr>';
            echo '<div id="twitterfeed"><ul style="margin:0px;"></ul></div>';
            echo '</section>';
        }
    }
?>
```

account.php – Display Twitter Feed only if user is not a visitor and if the member has a twitter account

```
<?php   // TWITTER STREAM - Only Show if TwitterID is set or is not 0.
if (isset($_POST['li_twitterID']) && $_POST['li_twitterID'] != '0'){
        echo '<div id="twitter" class="twitter" style="width:20%;
                                    border:2px solid #338bb2; color:#333;
                                    position:absolute;
                                    margin-left:60%; margin-top:0px; margin-bottom:30px;
                                    font-size:12px;">';
        echo '<div style="height:50px; padding:18px 50px; padding-bottom:30px;
        font-size:14px; letter-spacing: 0.1em; display:block; margin: 0 auto; width:100%; text-transform:none; font-weight:normal">
        <b> Twitter Feed: </b>' .$twt_name. ' <img src="img/icons/twitter.png" style="margin-top:2px;" height="20px"/></div>';
        echo '<hr>';
        echo '<div id="twitterfeed"><ul style="margin:0px;"></ul></div>';
    }
?>
```

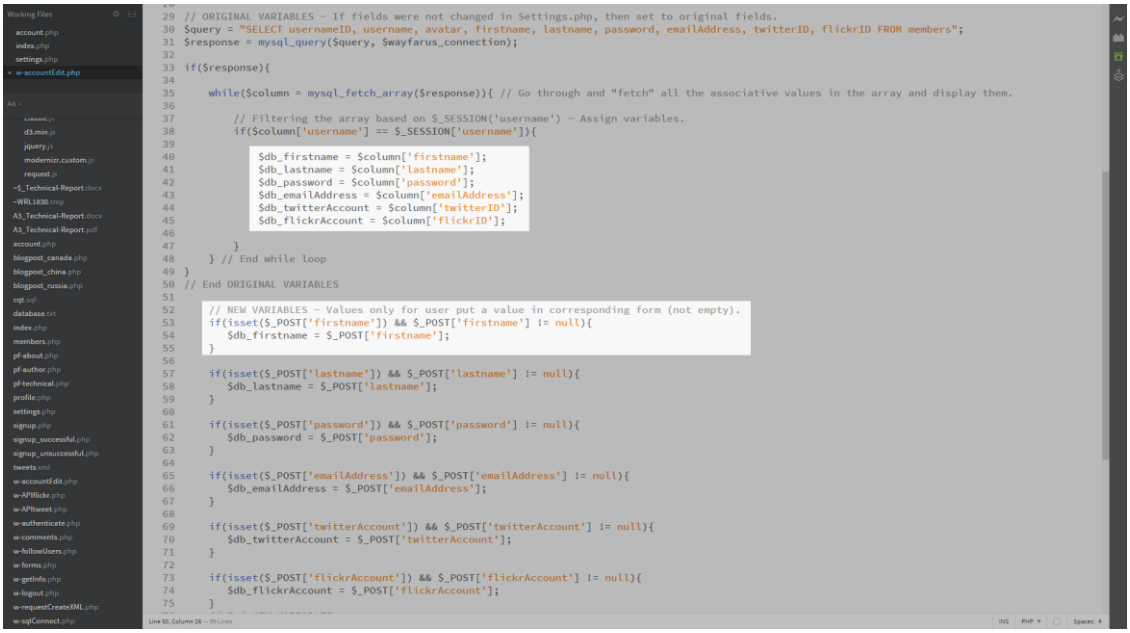
profile.php – Display Twitter Feed for a member's profile only if they have a twitter account

10. ACCOUNT UPDATE SETTINGS – *Update account settings without putting a value in every form*

Members are now able to update their Account Settings information without having to fill out all every form on the Settings page. When users click submit, the PHP file that processes these forms will store the current member's old information from the database inside a temporary variable. This is needed in case members choose not to enter in a value for a specific form. The value that gets stored will be the original information. If a field is not empty however, then the temporary variable will be assigned to the new value and get stored in the database.

```
    <!-- INPUTS -->
    <form action="w-accountEdit.php" method="POST" style="margin-top: 40px;">
        <section style ="width:50%; margin: 0 auto;">
            <div class="c2-form new_firstname" id="fname">
            <input type="text" name="firstname" placeholder="First Name"/>
            </div> <!-- end firstname -->
            <div class="c2-form new_lastname" id="lname">
            <input type="text" name="lastname" placeholder="Last Name"/>
            </div> <!-- end lastname -->
        </section>
        <div class="cform new_password">
        <input type="text" name="password" placeholder="Password"/>
        </div> <!-- end password form -->
        <div class="cform new_email">
        <input type="email" name="emailAddress" placeholder="email@domain.com"/>
        </div> <!-- end email form -->
        <div class="cform twitter_account">
        <input type="text" name="twitterAccount" placeholder="twitterID"/>
        </div> <!-- end twitter form -->
        <div class="cform flickr_account">
        <input type="text" name="flickrAccount" placeholder="flickrID"/>
        </div> <!-- end flickr form -->
        <br/>
        <div style="width: 50%; margin:0 auto;">
        <a href="settings.php">
        <input class="submit cta-button" style="color:#777; height:50px; width: 100%" type="submit" id="return" value="Confirm Changes"/></a>
        </div>
    </form>
</section> <!-- end CURRENT MEMBER -->
```

settings.php – Users do not need to update all account forms every time now



```php
// ORIGINAL VARIABLES - If fields were not changed in Settings.php, then set to original fields.
$query = "SELECT usernameID, username, avatar, firstname, lastname, password, emailAddress, twitterID, flickrID FROM members";
$response = mysql_query($query, $wayfarus_connection);

if($response){

    while($column = mysql_fetch_array($response)){ // Go through and "fetch" all the associative values in the array and display them.

        // Filtering the array based on $_SESSION('username') - Assign variables.
        if($column['username'] == $_SESSION['username']){

            $db_firstname = $column['firstname'];
            $db_lastname = $column['lastname'];
            $db_password = $column['password'];
            $db_emailAddress = $column['emailAddress'];
            $db_twitterAccount = $column['twitterID'];
            $db_flickrAccount = $column['flickrID'];

        }
    } // End while loop
}
// End ORIGINAL VARIABLES

// NEW VARIABLES - Values only for user put a value in corresponding form (not empty).
if(isset($_POST['firstname']) && $_POST['firstname'] != null){
    $db_firstname = $_POST['firstname'];
}

if(isset($_POST['lastname']) && $_POST['lastname'] != null){
    $db_lastname = $_POST['lastname'];
}

if(isset($_POST['password']) && $_POST['password'] != null){
    $db_password = $_POST['password'];
}

if(isset($_POST['emailAddress']) && $_POST['emailAddress'] != null){
    $db_emailAddress = $_POST['emailAddress'];
}

if(isset($_POST['twitterAccount']) && $_POST['twitterAccount'] != null){
    $db_twitterAccount = $_POST['twitterAccount'];
}

if(isset($_POST['flickrAccount']) && $_POST['flickrAccount'] != null){
    $db_flickrAccount = $_POST['flickrAccount'];
}
```

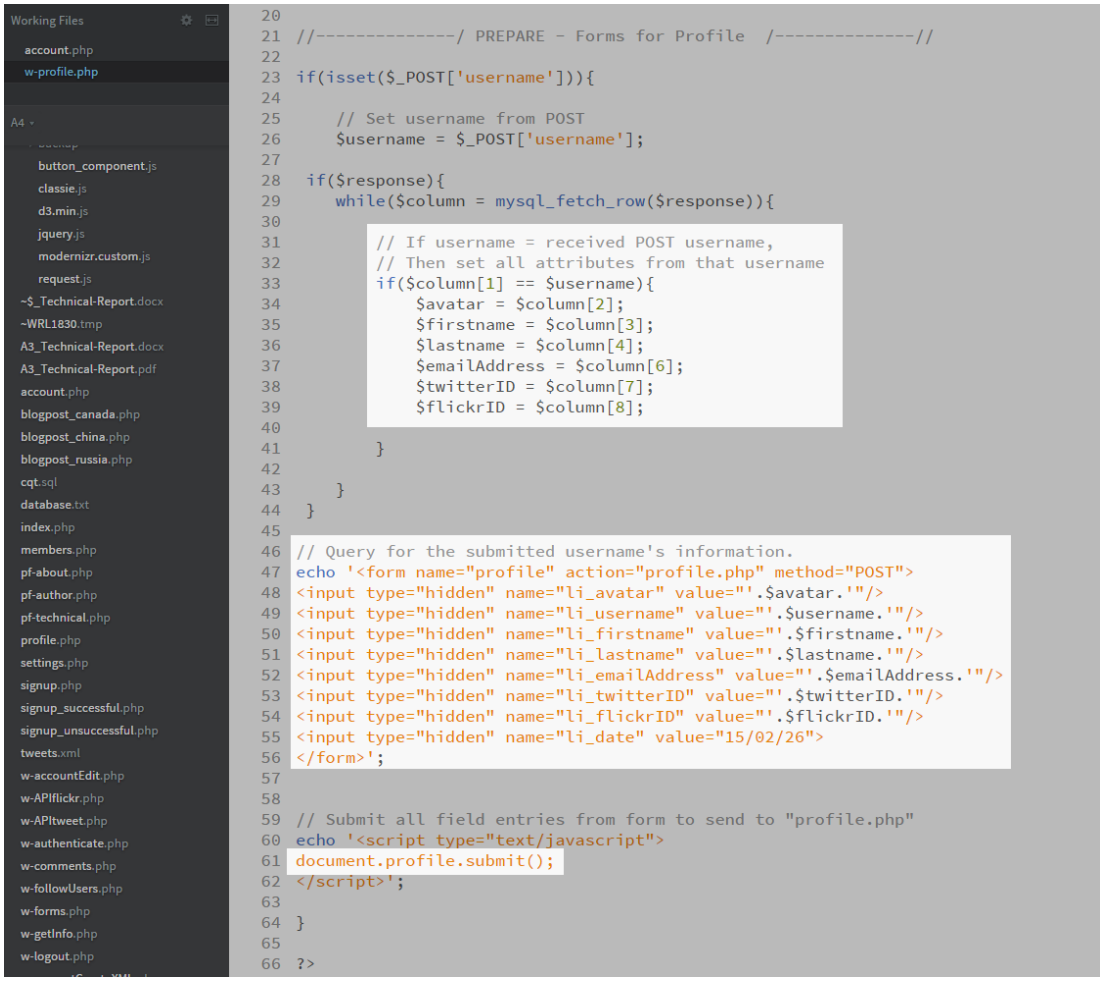
w-accountEdit.php – If user did not enter value into the form, then set it to its original value in database. Otherwise, set it to the form's value

11. FOLLOWING DASHBOARD TO PROFILES – *Implement navigating to profile feature when users click on members on dashboard*

Within the last iteration of this project, there limited functionality in terms of being able to access a member's profile page from the following dashboard. The code below is now implemented on each page that has the capability of accessing the Following Dashboard. Each time the current page is loaded, there will be additional forms that will be created for each member that the current user is following. In addition to creating a form, each member on the Following Dashboard now has an onclick event handler which process these forms.

```php
            foreach ($followedUser as $fwusers){
                $fwQuery = "SELECT username, followingUsers, firstname, lastname, avatar FROM members WHERE username = '".$fwusers."';";
                $fwResponse = mysql_query($fwQuery, $wayfarus_connection);

                //  Debug variable.
                //  echo var_dump($fwResponse); Used for dumping objects into variable strings.

                if($fwResponse){
                    while($row = mysql_fetch_array($fwResponse)){

                        // Create a form for each member
                        echo '
                        <form name="pform_'.$fwusers.'" action="w-profile.php" method="POST">
                        <input type="hidden" name="username" value="'.$fwusers.'"></input>
                        </form>';

                        // Create the visualization on the board for each member. POSTS will be happening in the future.
                        echo '<div id="p_'.$fwusers.'" onclick="submitForm(\'pform_'.$fwusers.'\')"
                        onmouseover="this.style.backgroundColor = \'#258ecd\'" onmouseout="this.style.backgroundColor = \'#47a3da\'"
                        style="color:white; cursor:pointer; height:120px;
                        padding-top:15px; border-bottom:1px solid #258ecd; ">
                        <div style="margin-left:15px; float:left;"><img src="img/avatars/'.$row[4].'" style="width:70px; height:70px;"/></div>
                        <div style="float:left; font-size:14px; margin-left:12px; margin-top:10px;">
                        <div style="font-style:italic; font-weight:bold;"> '.$row[0].'</div>
                        <div>'.$row[2].' '.$row[3].'</div></div>
                        </div>';

                    }
                } // end Loop through row of each Followed Users ($fwResponse)
            }

        } // end FollowingResponse
    ?>
</nav> <!-- end -->
```

account.php – Generate forms with only usernames of members that the current user is following



w-profile.php – In the database, find following the username that the current user clicked, then assign the rest of the username's information

profile.php – Check if the username is set before display it in profile.php

The w-profile.php process the username information from the click that happened earlier, then sets all the rest of the information such as avatar, first names, last names, emailAddress, twitterIDs, flickrIDs and dates that corresponds to the designated username.

When the designated username is not set, then the user should not be able to access this Profile page. This code was previously implemented for restricting users from being able to access the Profile page directly by entering in the URL. However, it can also be used for handling errors when POST information for designated username is not transferred corrected from user clicks.

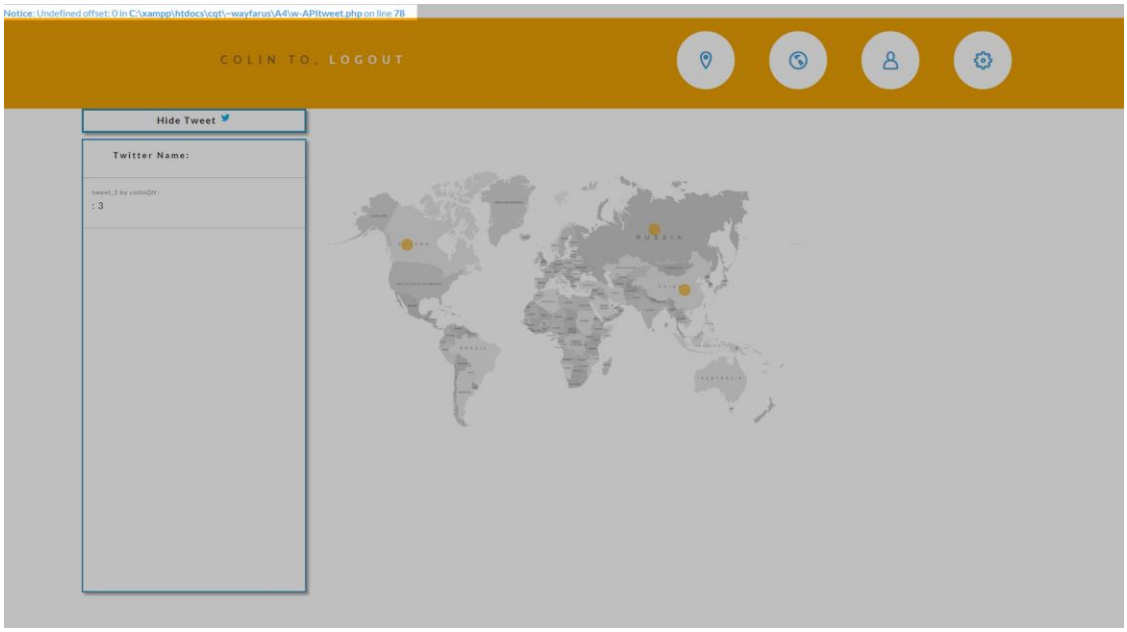
profile.php – Display all the information about that username in profile.php's view

The last step for the w-profile.php page is to send POST information to the profile page. The POST information that is sent to the page is the rest of the information retrieved from the designated username that were processed earlier. The data gets displayed on the Profile page using Divs.

ERRORS

1. TWITTERFEED TIMEOUT – *Sometimes w-tweetAPI.php will not retrieve account data from Twitter. Waiting resolves this issue.*

There is currently a strange bug in the system went AJAX in request.js retrieves information from w-APItweet.php file. The retrieval call times out on certain occasions and the twitterAccount would be set successfully for tweets to show up, corresponding to the designated twitterAccount. The bug may have occurred because either because there are multiple instances of require_once(w-APItweet.php) functions being executed. Another reason may be the setInterval being too short, but this is highly unlikely because there will be moments where the Tweets will show up and get updated successful for extended periods of time.



Web Page: account.php – Problem with Ajax timeout calls from w-APItweet.php

2. TOO MANY VISITORS – *A bug reading excessive echo statements from AJAX retrievals because of echo save to "Visitors.xml"*

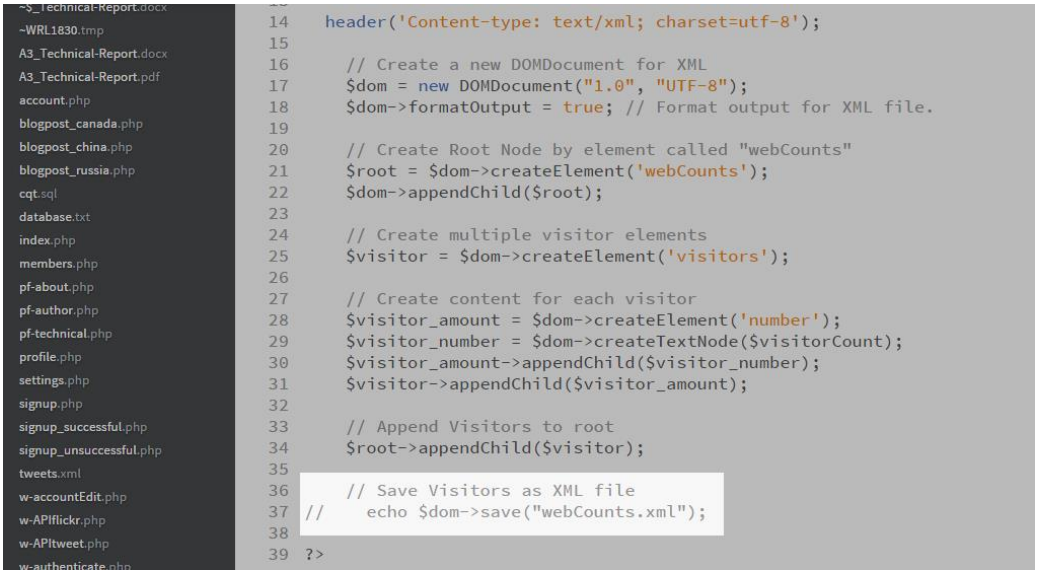
The first bug comes from having an echo statement within the PHP page that gets executed from the AJAX call within request.js. This is from the visitorSession function that gets called from the main website. The bug puts the string "114" in from of the visitor count, when the additional three digit number should be not be displayed. The bug is strange because AJAX is reading echo statements, while the echo statement for saving the XML produces this three digit number to appear. The current fix is to just comment out the echo statement for saving the XML.

Other methods have been used such as specifying DataType and trying to encode the visitorCount variable as a JSON variable in order to pass back to request.js. However, this was unsuccessful as the information for the other echo statement still retains.

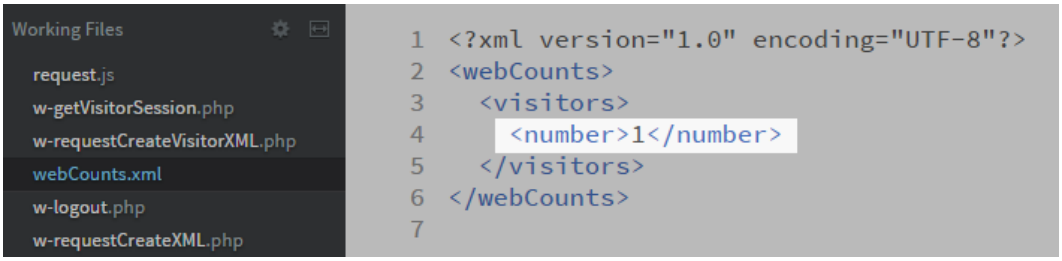
Another bug also occurs when you are currently logged in as a visitor or member and close the tab without logging out. It will not destroy the username session and will cause various problems. The current temporary fix to counteract this bug, you will need to create multiple tabs again and then log out however many times there are visitors logged in in order to log everything out. However, another solution needs to be met for future iterations for eradicating this bug.

There is also a bug that relates to multiple visitor sessions and logging out. If the user opens multiple tabs and logs in with various accounts and logs out, all of the tab sessions in the current browser will have an unset username session value and the server will assume that the user logged out for those extra tab sessions (if the user tries to navigate away from the current page, it will bring them back to the home page).

w-requestCreateVisitorXML.php – Creating the webCounts.xml (with echo problem)



w-requestCreateVisitorXML.php – Commenting out the echo problem (for now)



webCounts.xml – Retrieving from Number tag

The current fix for the echo bug is to comment out the feature entirely. By doing this, we unfortunately are not able to save and update the visitors.xml file. The save to xml functionality will be implemented for future iterations once the bug gets resolved with other methods.

In the future, the visitor count information would be retrieving information from webCount.xml and displaying it on the Account web page. However, because of this problem, I was not able to implement it for this iteration. The benefits of dynamically storing and retrieving values from an XML reside in the capabilities of storing additional information about visitors such as visitorID and log in times.

GLOSSARY

Web Pages: [Improved]

account.php – There have been many changes to the Account page. Some of the main changes to the Account page include the added visibility to navigate to portfolio pages. The Account page now also shows the visitor and blog count for the entire web application, updated dynamically using AJAX, currently using session variables, but will use visitors.xml in the future. The Account page also hides certain navigation features from visitors and the Twitter Feed from them as well.

profile.php – The modification to the profile page is that it now checks if the corresponding members have a twitter account and will only show a Twitter Feed if the member has a twitter account.

signup.php – The index page now includes the feature for clicking on the Wayfarus logo to visit the website.

signup.php – The signup page now includes the feature for clicking on the Wayfarus logo to visit the website. It also includes the FlickrID form now that was not present in last iteration.

settings.php – The Settings page is now updated so that all forms do not need a value to be entered in order to process the form.

PHP Process Pages: [New & Improved]

w-forms.php – The new twitter field now POSTs from the signup page and is processed here for storing twitterIDs into the database.

w-authenticate_visitor.php – This new authenticate PHP page now gets processed when users click on the Wayfarus logo and choose to visit the web page. The username session variable will now be assigned as the string, visitor. The PHP page also increases the visitor count by 1.

w-logout.php – There is an added protocol in this PHP page, which checks if the username session is equal to visitor. If it is, then there is an additional step to decrease the visitor count by 1 as well as unsetting the username session variable.

w-accountEdit.php – This processes the forms from the Settings page. If the fields do not have an entry, then this PHP page will set the values to the original value, previously set within the database for the current user's account.

w-profile.php – This PHP page gets processed when users click on members on their Following Dashboard. As the form process the click, it seeks for that designated member's username within the database and sets the others values to POST variables for manipulation in the Profile page.

w-requestCreateXML.php – This is the PHP page that creates the tweets.xml. This PHP changes the value of li_twitterID and makes it equal to whichever account is needed and will be visualized on the current web page. The page also formats Twitter Feeds to the appropriate amount of tweets that is retrieved from a member's twitter account.

w-requestCreateVisitorXML.php – This PHP page is responsible for creating the visitor.xml page. Right now, this page is not utilized within this implementation, but will play a key part in future iterations when multiple visitors will need additional information about them such as visitorID and log in time stamp.

Javascript Files [New]:

request.js – Handles all the AJAX calls and updates the tweets.xml and visitors.xml separately according to functions that PHP and POST data passes it. There are two methods currently being used simultaneously within this javascript file in order to do the AJAX call. One method uses xmlHttpProtocol which requires the creation of an xmlHttpProtocol object, configuring its data and sending that to another file to retrieve information. The other method is to use $.ajax to achieve the same result with similar configurations, but a shorter way of writing the code.

XML Files [New]*:*

tweets.xml – Content and tags are dynamically generated and updated through an AJAX call in *request.js.* It stores Tweets for the member's page that the user is currently viewing. AJAX also retrieves from this file dynamically in order to display the amount and the content for the Twitter Feed that the user is currently viewing.

visitors.xml – Content and tags are dynamically generated and updated through an AJAX call in *request.js.* This file is currently not being used in this implementation of the system. However, in the future, it will keep track of the amount of visitors that is currently visiting the website as well as information about that visitor, such as visitorID and when they last logged in, adding more capabilities to the web application's features.