# JavaScript Assignment ( Due November 30st, 2017 by 11:59pm)

## Updates:

11/14/2017: Fixed a bug in shout.php (Problem 2)
11/20/2017: Provided needed JS for POST request to be handled correctly
11/28/2017: Permissions need to be changed if you check out code early

## Background

This assignment requires you to integrate JavaScript code with a number of HTML pages and server-side scripts. The problems are designed to give you some hands on experience in areas which JavaScript is known for:

- Form Validation
- HTML & CSS Manipulation
- Text Parsing (Regular Expressions, Splitting Strings, Etc...)
- Asynchronous JavaScript and XML (Ajax)
- JavaScript Object Notation (JSON)
- Cookies
- JavaScript Dialogs

Some general guidelines for the problems:

- I have provided various resources (HTML, CSS, JS, Server-Side Scripts, etc...) for each of the three problems.
- You are to solve the problems using only JavaScript that is written by you, or obtained from course notes. You may, however, use the jQuery library (as well as leaflet.js on the last problem). If you would like to use another framework, send me an email to verify that it's okay first.
- Your code should have some comments. At a minimum you should have comments indicating the author and some comments as to what the major parts of the script are doing.

- Since your projects will need to be graded from within a web browser, a common browser needs to be specified as the target environment. For this assignment, your project will be tested the latest stable release of Google Chrome.

This assignment will be graded directly through a web browser, and examining code stored in your public directories on GL. Do not modify any of the files after the deadline.

## Setup Script

Update 11/28/17

If you ran the set-up script very early on, the permissions I need in order to grade your assignment were not setup correctly. To determine if you need to fix this, run the following from your home directory on GL:

```
fs la swe2017
```

If what you see is the following:

```
Access list for swe2017 is
Normal rights:
  oit.useraccess rl
  system:administrators rlidwka
  system:anyuser rl
  YOUR_GL_NAME rlidwka
  www.swe_web rl
```

You do not need to take any further action
If you do not see the above output, specifically the line `system:anyuser rl` , you need to run the following to make sure your directory is readable by any user. Don't worry, the permissions below are more restrictive, just the top level needs to be world readable

```
fs sa swe2017 system:anyuser rl
```

I will be checking your files this Friday, and will send you an email if I don't have access

This section describes the process in order to setup your account to work on the JavaScript (and next) assignment.

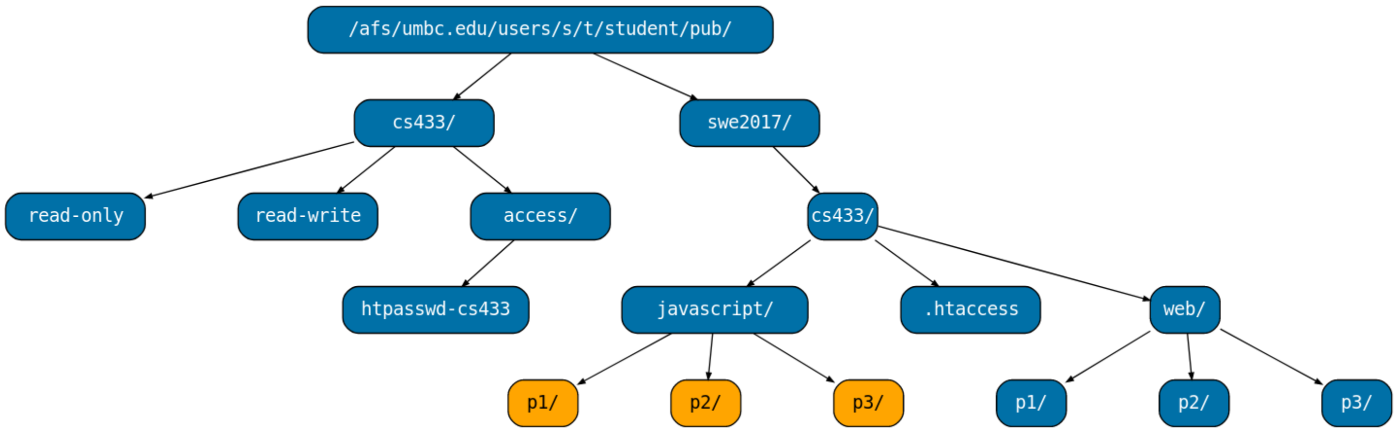
You are not permitted to work on these problems in a publicly visible area (e.g. under your www directory on GL). However, in order to get the problems to work correctly, it needs to be viewed in a browser from a location on http://www.swe.umbc.edu/. The is especially true for the last 2 problems which are making Ajax requests.

If you haven't used the Student Web Enviroment (SWE) before, or in a while, you will need to activate it by running the following from the command line

```
/afs/umbc.edu/common/bin/enable_student_web_space.sh
```

After you have done that, you will need to run the following script to setup from your **GL account** to set things up for this and the next assignment.

```
/afs/umbc.edu/users/b/w/bwilk1/pub/setup433
```



Directory Layout

Let me give a brief overview as to what some of the relevant directories and files are for...

- `/home/bubba1/` — This is the root of your GL account (i.e. your home directory).
- `/home/bubba1/swe2017/` — Anything put in this directory can be viewed on the swe webserver. For example if you put a file called `foo.png` in your `swe2017` directory, then you could access it online at http://swe.umbc.edu/~bubba1/foo.png.
- `/home/bubba1/swe2017/cs433/` — This is the sub-directory created for you to put all of your assignments under.
- `/home/bubba1/swe2017/cs433/javascript/` — This directory houses 3 subdirectories called `p1`, `p2`, and `p3` for each of the 3 problems in this assignment.

## File & Directory Permissions

There are some additional files & directories created by this script — **do not delete, rename or re-chmod any of them**. **No course-related code be placed outside of the `p1` , `p2` or `p3` directories.** Doing so will be treated as a violation of academic integrity and will be dealt with accordingly.

**You must adhere to the following rules** regarding permissions of any file & directories **you create** under your `p1` , `p2` or `p3` directories. You **should not change the permissions of any existing files or directories** created by the setup scripts or the provided copy commands.

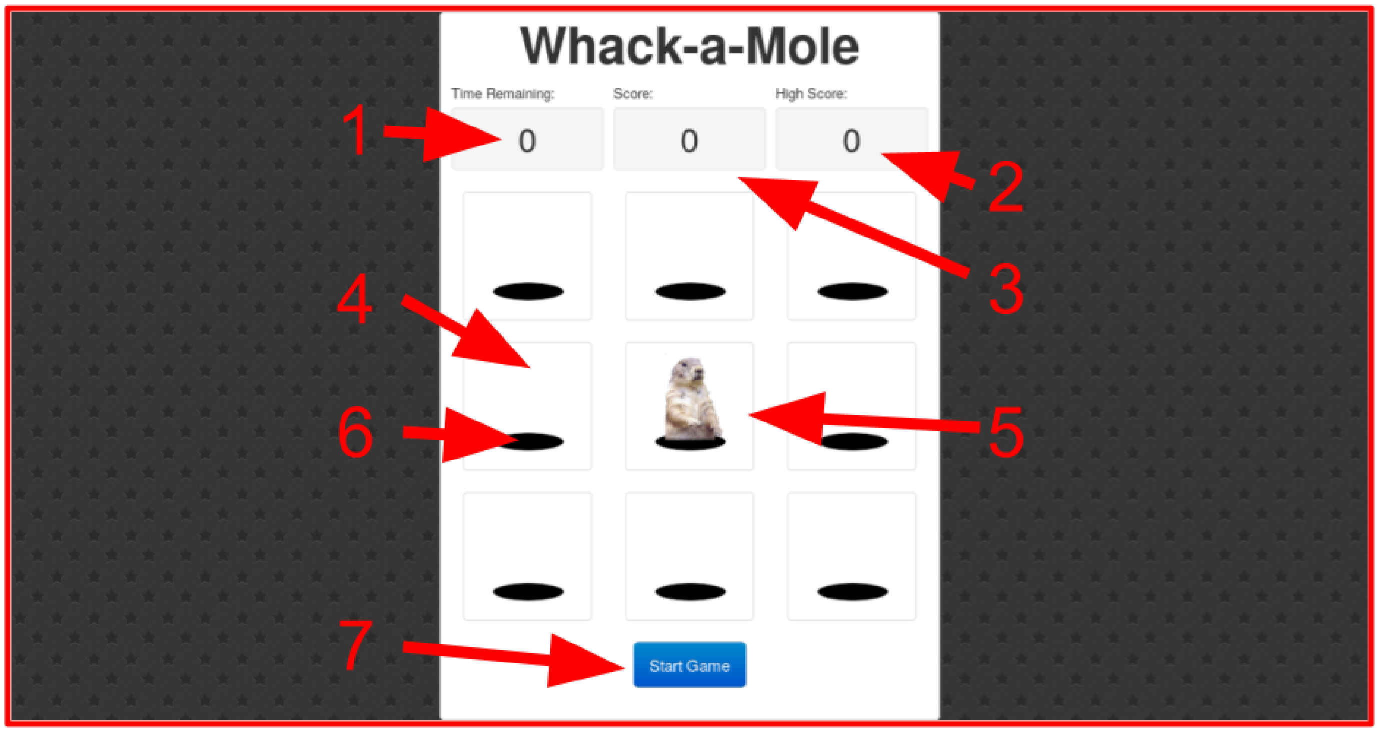
- Files you might create such as CSS, JavaScript, images, etc. **must be set with permissions** `rw----r--` . You will need to use the `chmod` command to set these permissions once you create a file.

- If you create any directories they **must be set with permissions** `rwx---r-x` . Again, you will need to use the chmod command.

## Whack-a-Mole (35%)

### Task

For this assignment you will implement the game of Whack-a-Mole using JavaScript.

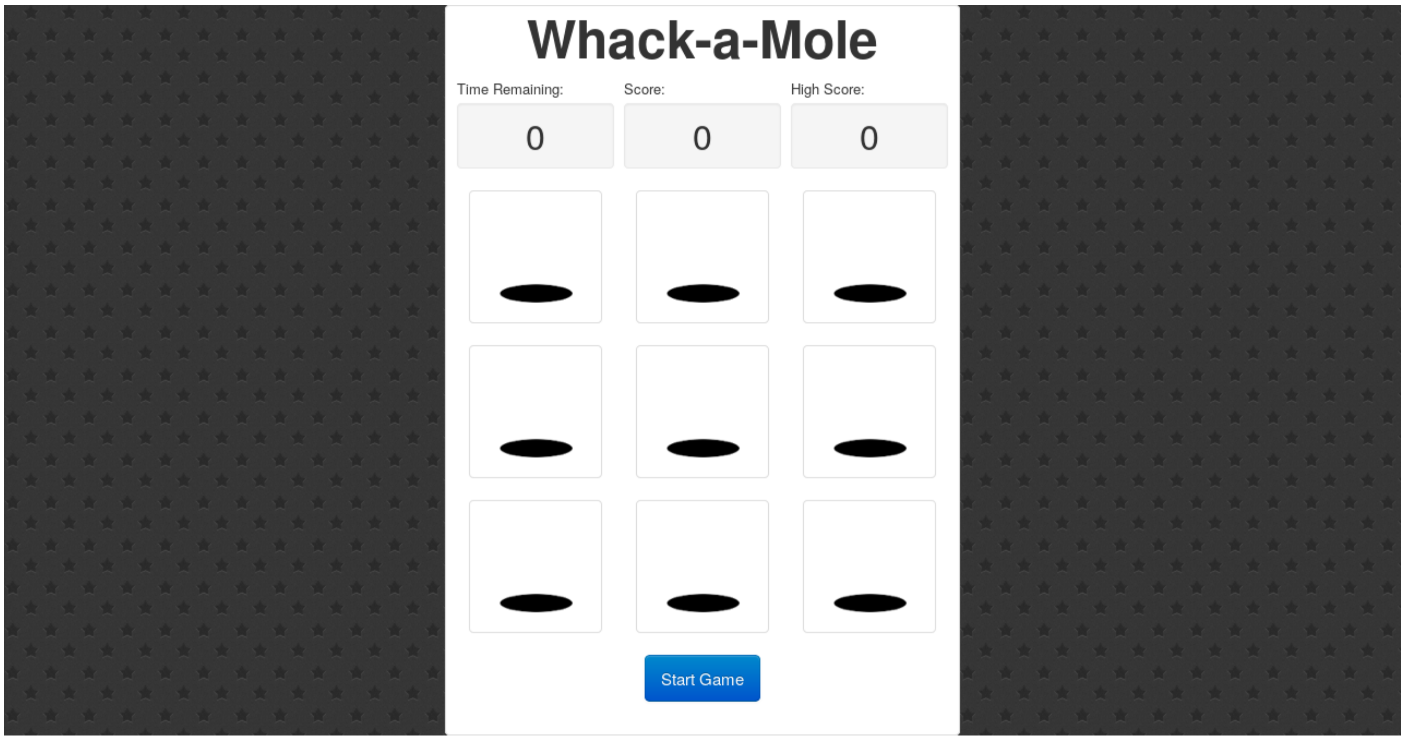


Whack-a-Mole Figure 1 — Page components

1. The "Time Remaining" box shows the amount of time left in the current game - it ticks by the second.
2. The "High Score" box shows the highest score achieved so far.
3. The "Score" box show the number of moles hit this game.
4. The main board consists of a 3x3 grid of tiles which can be clicked.
5. Moles will randomly pop-up at a given location for a small amount of time.
6. Usually the mole is down (as shown by an empty hole).
7. Last is the "New Game" button. This resets the time remaining, the score and resets game play

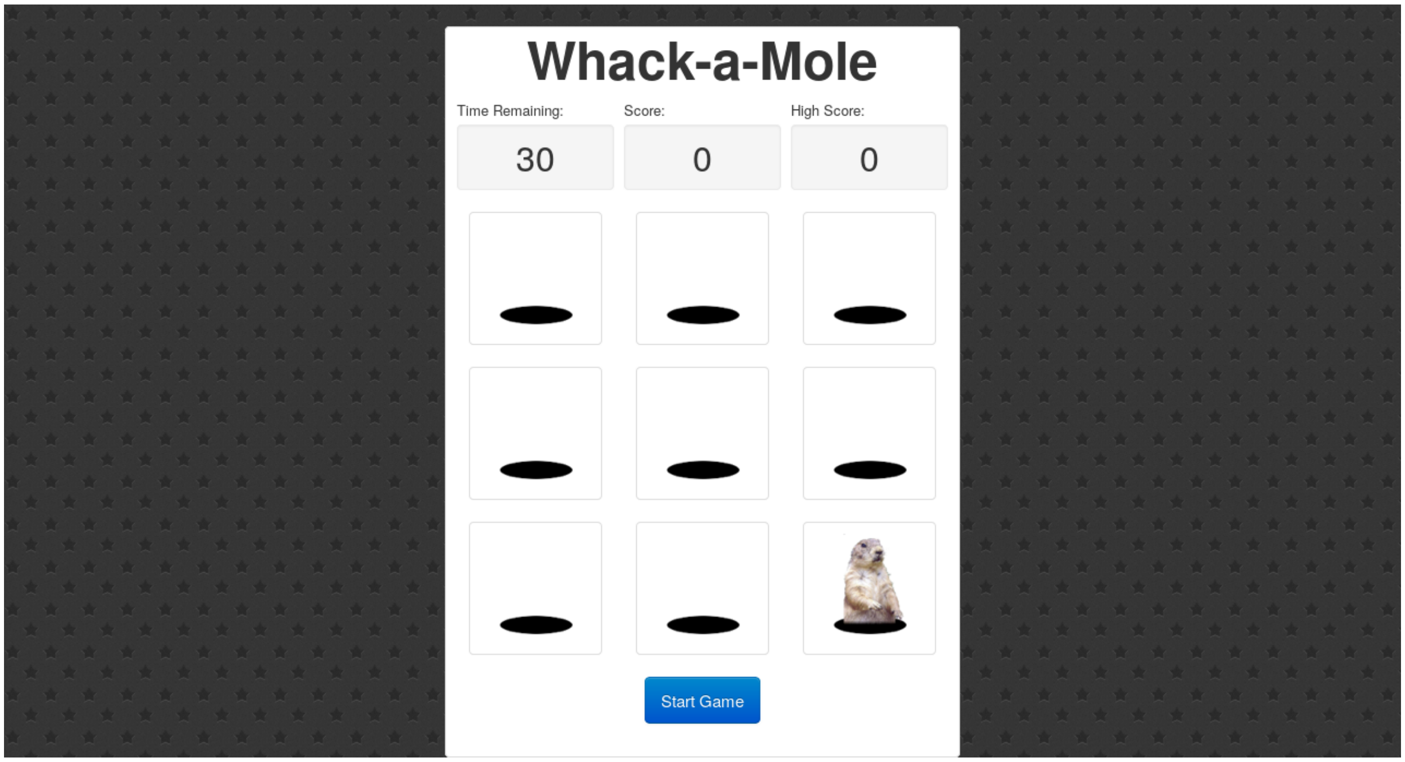
## Behavior

When the page initially loads it should look like as shown in Figure 2. This is pretty much what you get with the provided HTML document.



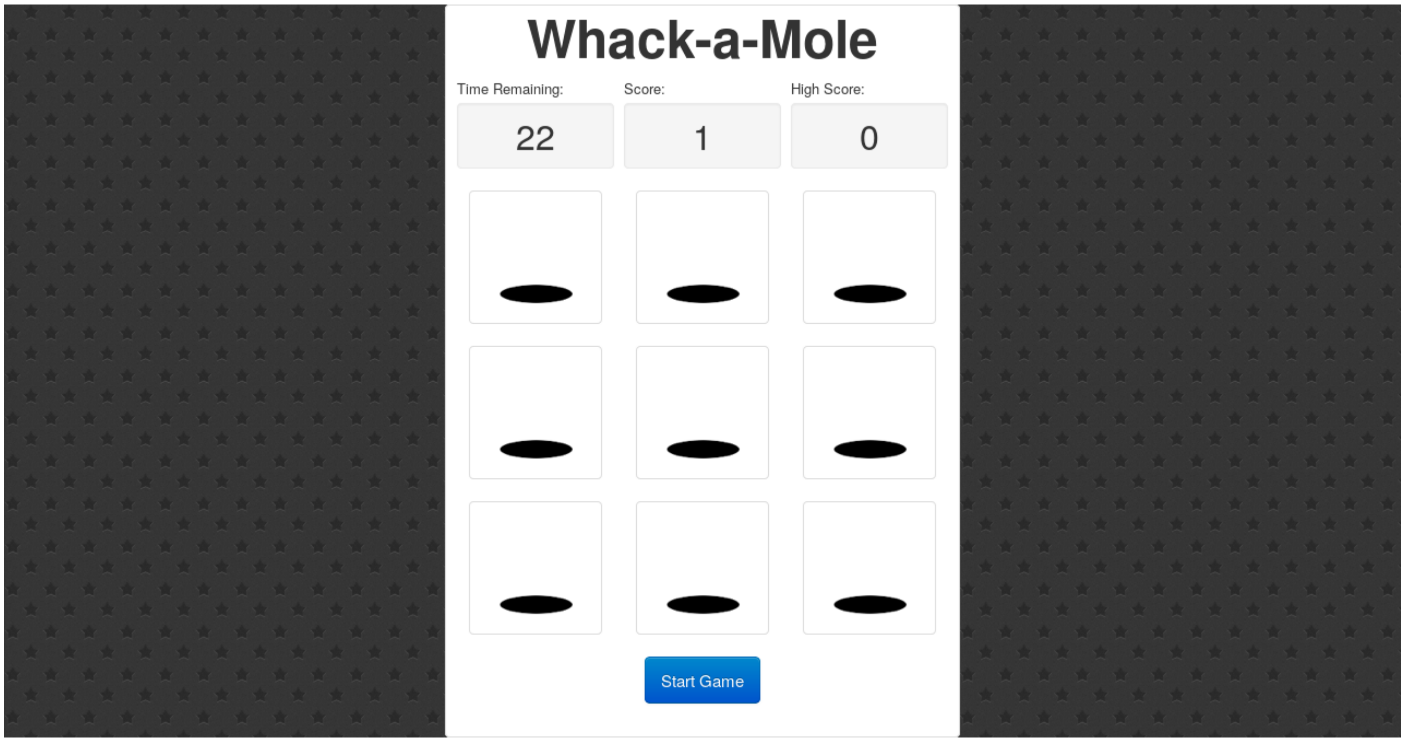
Whack-a-Mole Figure 2 — Whack-a-Mole as initially loaded

Once the "Start Game" button is clicked the time remaining is initialized to 30 seconds and the score is reset to zero. Every second a mole pops up at a random location on the grid. The mole should remain up for a short time (3/4 of a second seemed to work well for me) and then it should disappear.



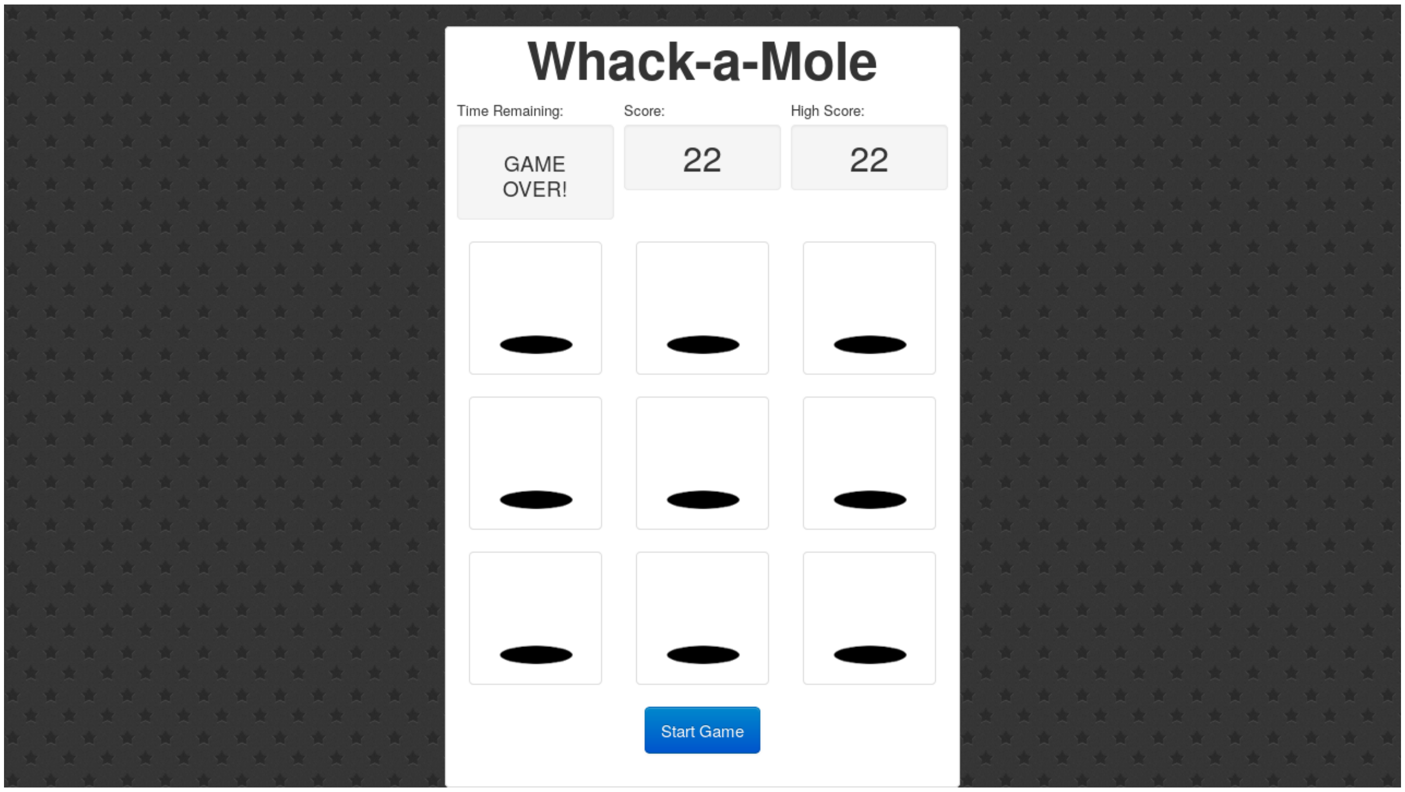
Whack-a-Mole Figure 3 — Whack-a-Mole starting a new game

If a mole is clicked while it was up, the image should immediately switch back to the hole and the user's score should be incremented and displayed in the score box. Be sure that the user can only gain 1 point each time a mole pops up (user should not be able to double click mole and get 2 points). Clicking on a hole during game play should have no effect.



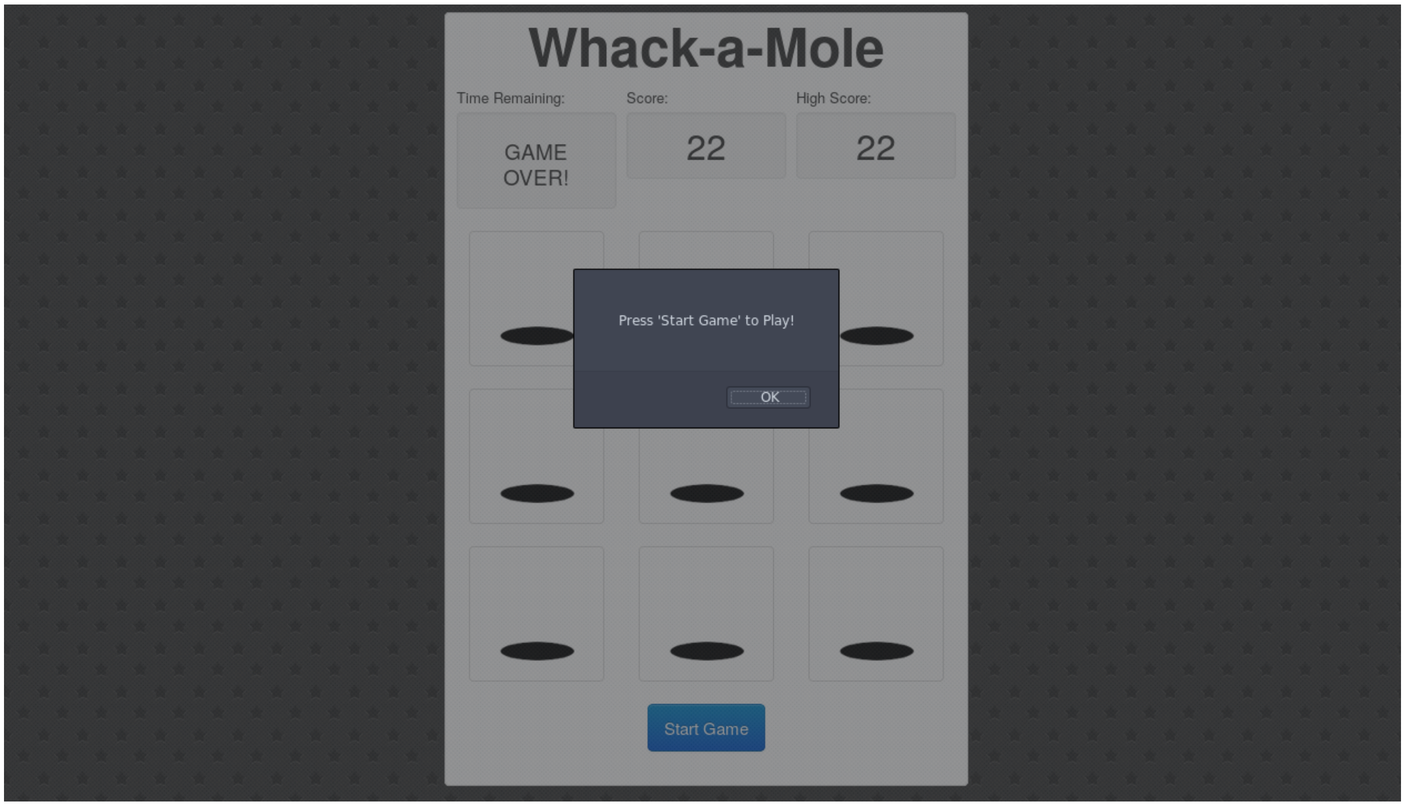
Whack-a-Mole Figure 4 — Whack-a-Mole once clicking on a mole

Once the game is over the random popping up of moles should be immediately stopped. Also, the time left should be updated to state that the game is over. If the score is a new high score, the high score should both be updated on the site, as well as saved in local storage on the users machine.



Whack-a-Mole Figure 5 — Whack-a-Mole once game is over

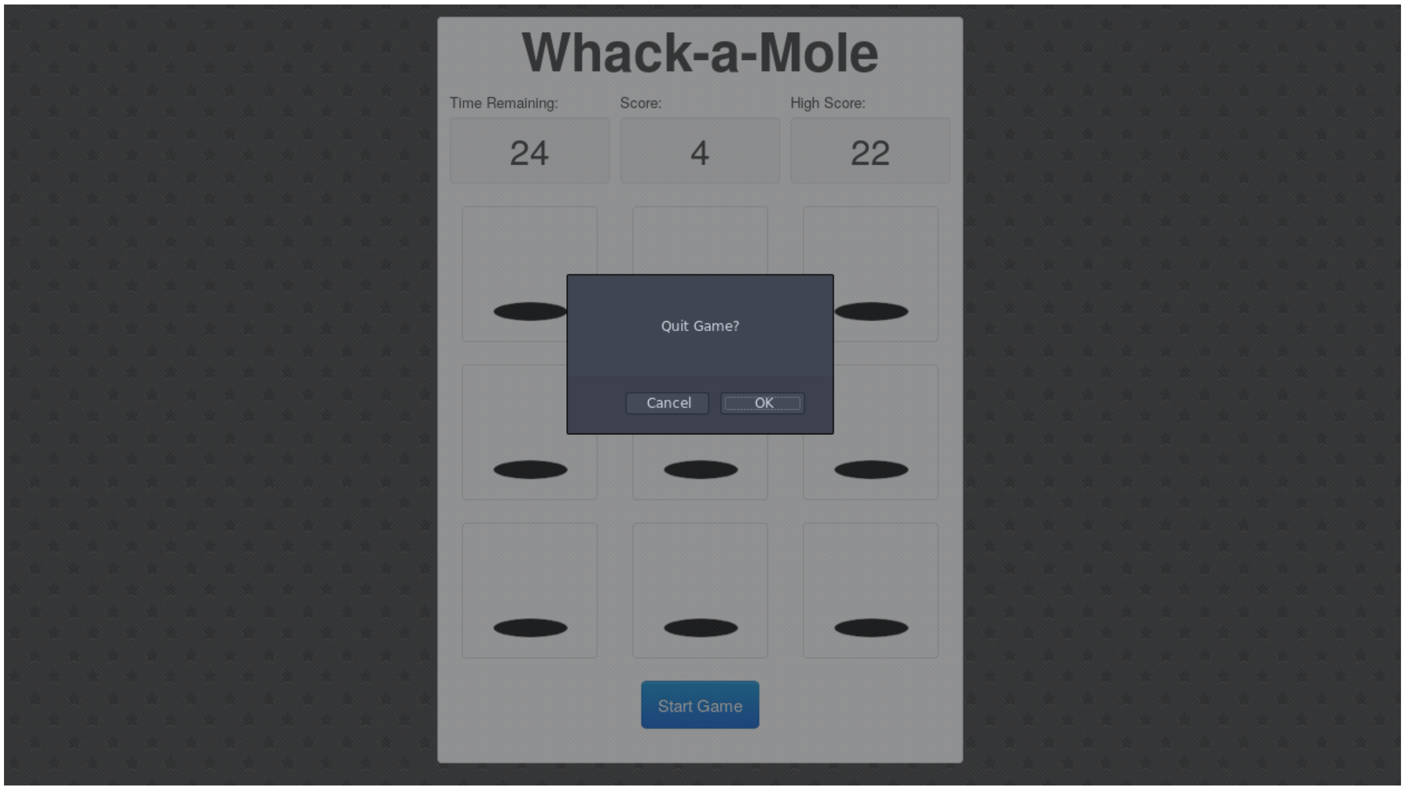
It at any point the game is over and the user clicks on a hole, he/she should be presented with a dialog that informs that he\she needs to click "Start Game" to begin.



Whack-a-Mole Figure 6 — Whack-a-Mole whacking when game is over

Likewise, if the user is in the middle of a game and he/she clicks "Start Game" then he/she should be presented with a dialog asking if the game should be aborted. If the user clicks yes, then the game should restart. If the user clicks no, then the dialog should go away and the game should continue. This confirmation should not be shown to the user if a game is not already in play.

Note that when the dialog is up, the game should be paused. If the user clicks no, the game should pick back up with the clock running from where it left off (no time should elapse while dialog is up).



Whack-a-Mole Figure 7 — Whack-a-Mole when trying to start new game when already in play

## Files

I have provided the following files for this problem:

- `index.html` — this file contains the HTML for the layout shown in the screen shots - this is where you will be doing most of your work. You will add JavaScript event handlers and JavaScript functions into this file. If you wish to externalize the JavaScript in an external `.js` file feel free to do so.
- `css/bootstrap.min.css` — the Twitter bootstrap stylesheet.
- `css/custom.css` — CSS customizations for the HTML page in this problem.
- `img/down.png` — the "down" mole image.
- `img/up.png` — the "up" mole image.
- `img/starring.png` — background image used by the page.
- `img/touch-icon-*.png` — various images used by iOS if you add to your home screen.

These files will copied to your swe2017 directory as part of the setup script.

## Shoutbox (35%)

Update 11/14/17

There was a small bug in the file shout.php used for this assignment. If you have already run your setup script, you will need to copy over a corrected version. The command to do this is (from your home directory):

```
cp /afs/umbc.edu/users/b/w/bwilk1/pub/433_start_files/js/p2/server/shout.php ../pub/swe2017/cs433/j
```

This will overwrite your existing shout.php with the corrected one

Update 11/20/17

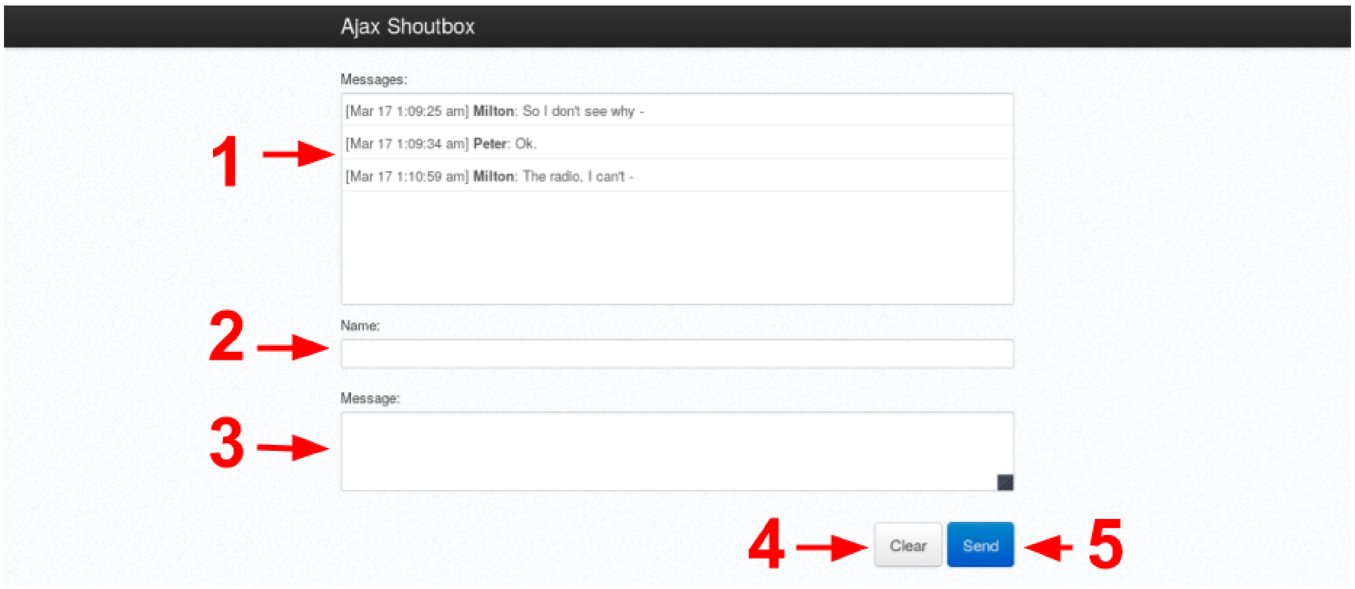
When sending a POST request from some browsers, UMBC's PHP will not automatically populate the right arrays it needs. To get around this, we need to explicitly tell it we are passing form data in the JavaScript. To do this, add the following line of code in your JavaScript, replacing `xhr` with your variable name.

```
xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
```

This must be done before you call the `send()` method.

## Task

For this assignment you will implement a Shoutbox client using AJAX and JSON.



[Mar 17 1:09:25 am] **Milton**: So I don't see why -
[Mar 17 1:09:34 am] **Peter**: Ok.
[Mar 17 1:10:59 am] **Milton**: The radio, I can't -

Shoutbox Figure 1 — Page components

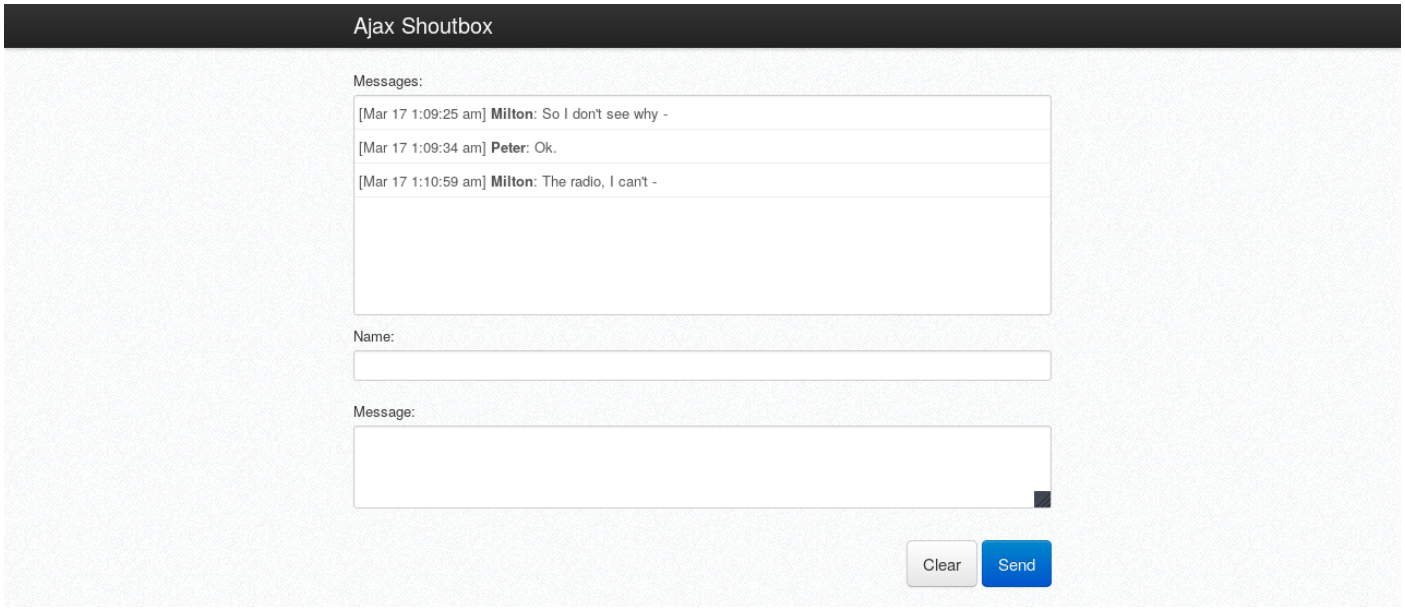
1. The messages will consist of a div that is continuously updated with the latest messages.

2. The name text field is for the user to enter their name to be associated with the post.
3. The message textarea is for the user to enter their message.
4. The "Clear" button should remove all text from the message textarea and select focus on the textarea.
5. The "Send" button should post the message to the server.

## Behavior

When the page initially loads it should look similar to as shown below. Upon loading the page, an AJAX request should be sent to the server requesting the latest messages. A JSON response is returned from the server and should be parsed and the output shown in the messages box. The last 10 messages are shown each consisting of the time of the post, the name given and the contents of the post. The newest posts are at the bottom, with the oldest having scrolled off of the top. The application should be setup to fetch and update the messages at an interval of once every 15 seconds.

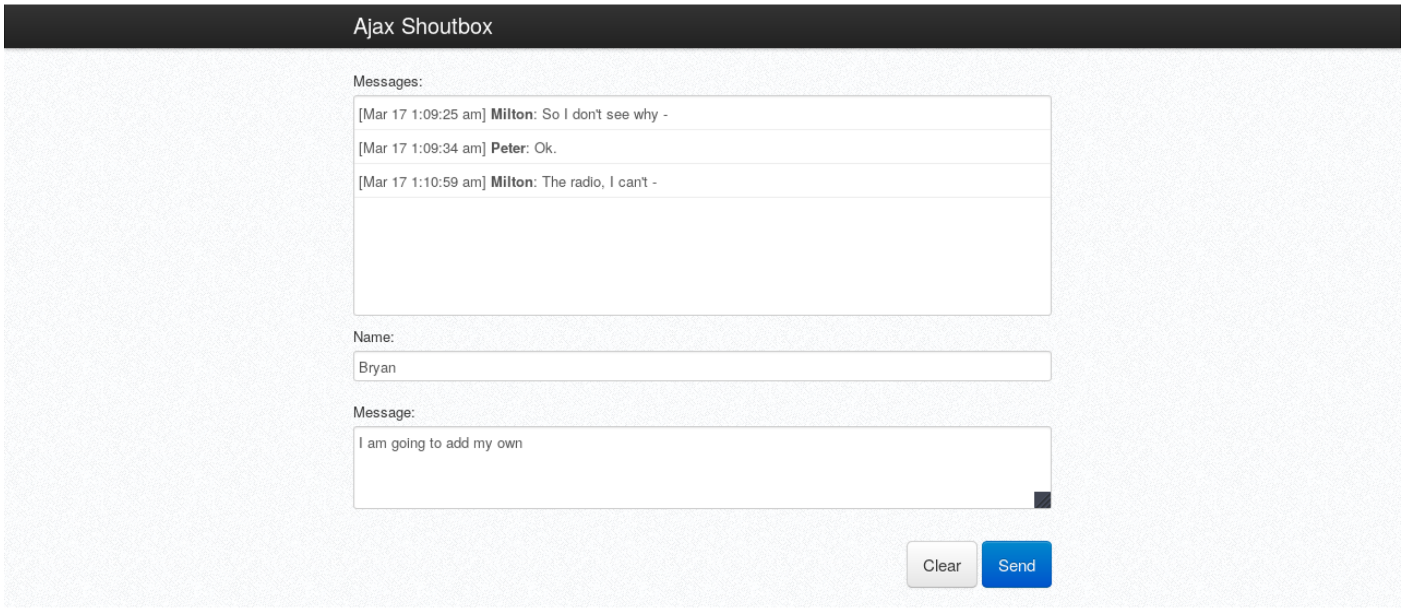
If the user had previously been to the page and their name had been stored in local storage by the application, then it should be displayed in the name text field. If no local storage has been set, then the name field should be blank.



[Mar 17 1:09:25 am] **Milton**: So I don't see why -
[Mar 17 1:09:34 am] **Peter**: Ok.
[Mar 17 1:10:59 am] **Milton**: The radio, I can't -

Shoutbox Figure 2 — Initial page load

The user can enter their name and message as shown below. The user has just finished tying in the messages text area, but has not yet clicked "Send".


Ajax Shoutbox

Messages:

[Mar 17 1:09:25 am] **Milton**: So I don't see why -

[Mar 17 1:09:34 am] **Peter**: Ok.

[Mar 17 1:10:59 am] **Milton**: The radio, I can't -

Name:

Bryan

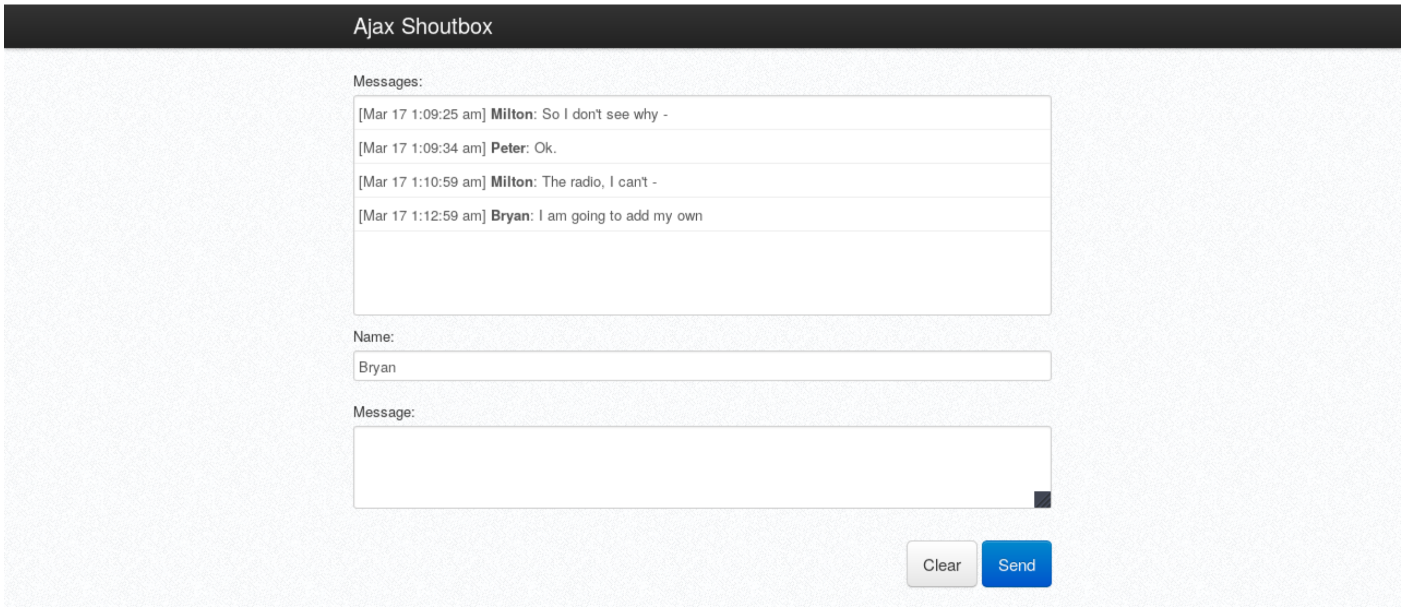
Message:

I am going to add my own

Shoutbox Figure 3 — Typing a message

When the user clicks "Send", the name and message should get bundled up and an AJAX request should be posted (not a get, but a post) to the server. The response from the server should include the last 10 messages modified so that the submitted message is the latest in the list.

Also, upon clicking the "Send" the messages area should be cleared of all text and should also be given focus. So, after sending the message, the user should be able to begin typing without manually re-clicking on the message area. Additionally, the typed name should be stored in local storage.


Ajax Shoutbox

Messages:

[Mar 17 1:09:25 am] **Milton**: So I don't see why -

[Mar 17 1:09:34 am] **Peter**: Ok.

[Mar 17 1:10:59 am] **Milton**: The radio, I can't -

[Mar 17 1:12:59 am] **Bryan**: I am going to add my own

Name:

Bryan

Message:

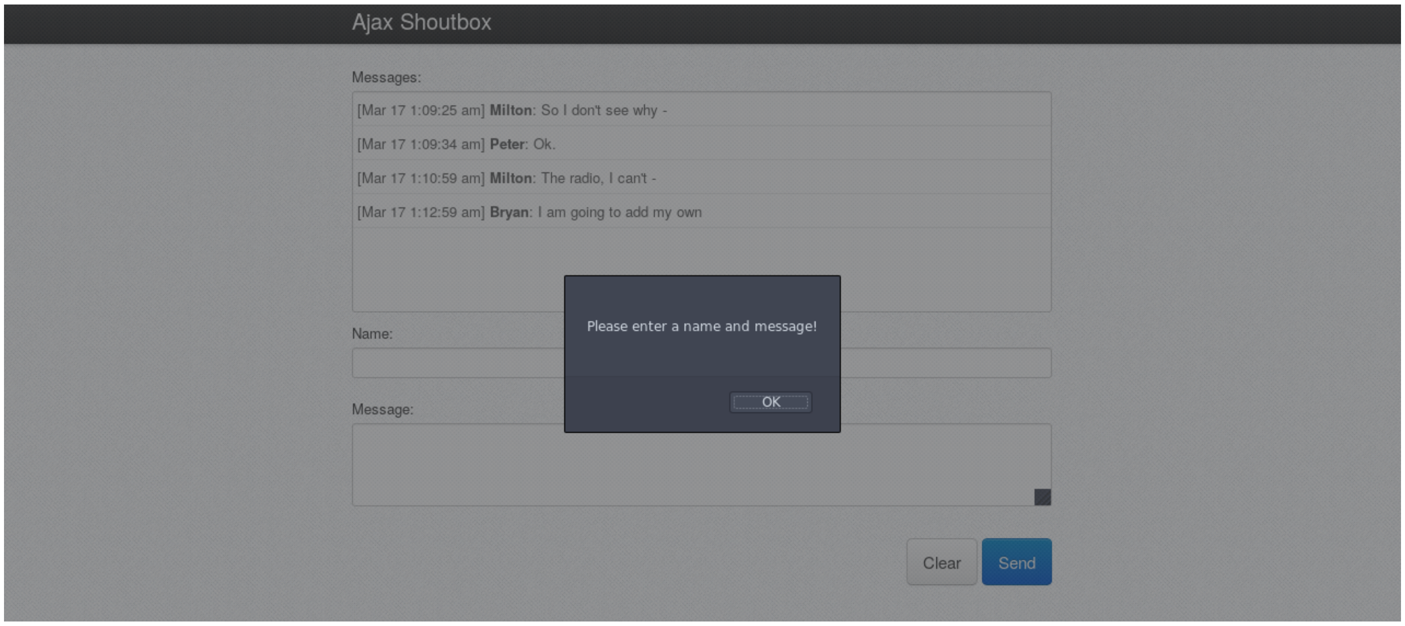
Shoutbox Figure 4 — The sent message

As the application continuously polls the server via AJAX requests for updates, new messages are parsed and displayed to the user. You can test this by using two browser tabs or windows.

If the user clicks the "Clear" button, all text in the message area should be cleared and the message area should be given focus (same behavior as "Send" but without posting the data).

The user should be able to close his/her browser and return to the page at any point in time. Assuming the user had already sent a message, the name field should be automatically populated by retrieving the value set in local storage when a message was sent.

If the user attempts to send a message without having entered a name or message then the user should be alerted their error by way of an alert.


Ajax Shoutbox
Messages:
[Mar 17 1:09:25 am] Milton: So I don't see why -
[Mar 17 1:09:34 am] Peter: Ok.
[Mar 17 1:10:59 am] Milton: The radio, I can't -
[Mar 17 1:12:59 am] Bryan: I am going to add my own
Name:
Message:
Please enter a name and message!
OK
Clear
Send

Shoutbox Figure 5 — Alerting the user of errors

## Server Side Script

For this problem you will use a script written for you on the server-side. The location of this script that you will be making AJAX requests to will be located at `http://swe.umbc.edu/~USERNAME/cs433/javascript/p2/server/shout.php` . This script behaves in 1 of 2 ways depending upon how you are interacting with it.

If a **get** request is made to the script (no arguments necessary) then it will return a JSON encoded response containing the last 10 messages posted - the newest in the first position, the oldest in the last position. The response is an array of 10 objects. Each object in the array contains the following 3 fields.

- `time` - a timestamp (string) representing the time the message was received by the server.
- `name` - the name (string) associated with the message
- `message` - the actual message (string) submitted by the user

An example response for a get request is shown below (lines have been wrapped for readability - they are not wrapped in the actual response).

```
{
  "result":"success",
  "data":[
    {
      "name":"Dom",
      "message":"Yeah. It's just that we're putting new coversheets on all the TPS reports before now befo
      "time":"Mar 17 1:15:07 am"
    },
    {
      "name":"Peter",
      "message":"Yeah. I got the memo. And I understand the policy. The problem is, I just forgot this time
      "time":"Mar 17 1:14:54 am"
    },
    {
      "name":"Dom",
      "message":"Yeah. Uh, did you get that memo?",
      "time":"Mar 17 1:14:39 am"
    },
    {
      "name":"Peter",
      "message":"Yeah. The coversheet. I know, I know. Uh, Bill talked to me about it.",
      "time":"Mar 17 1:14:29 am"
    },
    {
      "name":"Dom",
      "message":"Hello, Peter. What's happening?  :-) We need to talk about your TPS reports.",
      "time":"Mar 17 1:12:54 am"
    },
    {
      "name":"Milton",
      "message":"I enjoy listening to the radio at a reasonable volume from nine to eleven. :-D",
      "time":"Mar 17 1:11:19 am"
    },
    {
      "name":"Peter",
      "message":"Yeah! All right!",
      "time":"Mar 17 1:11:09 am"
    },
    {
      "name":"Milton",
      "message":"The radio, I can't -",
      "time":"Mar 17 1:10:59 am"
    },
    {
      "name":"Peter",
      "message":"Ok. :-0",
      "time":"Mar 17 1:09:34 am"
    },
    {
      "name":"Milton",
      "message":"So I don't see why -",
      "time":"Mar 17 1:09:25 am"
    }
  ]
}
```

If a **post** request is made to the script with a `name` and `message` values provided then it will go ahead and add your message to the list of latest messages (removing the oldest from the list) and return the new list of the last 10 messages - including your post.

An example response for a post specifying the name and message (URL encoded) shown below:

```
name=Samir&message=Oh%20no!%20Not%20again!%20Why%20does%20it%20say%20paper%20jam%20when%20there%20is%20no%2
```

Returns the following response (again wrapped here, not in actual response):

```
{
  "result":"success",
  "data":[
    {
      "name":"Samir",
      "message":"Oh no! Not again! Why does it say paper jam when there is no paper jam?!!",
      "time":"Mar 17 11:39:55 pm"
    },
    {
      "name":"Dom",
      "message":"Yeah. It's just that we're putting new coversheets on all the TPS reports before now befo
      "time":"Mar 17 11:39:35 pm"
    },
    {
      "name":"Peter",
      "message":"Yeah. I got the memo. And I understand the policy. The problem is, I just forgot this tim
      "time":"Mar 17 11:39:18 pm"
    },
    {
      "name":"Dom",
      "message":"Yeah. Uh, did you get that memo? :-|",
      "time":"Mar 17 11:38:40 pm"
    },
    {
      "name":"Peter",
      "message":"Yeah. The coversheet. I know, I know. Uh, Bill talked to me about it. :-(",
      "time":"Mar 17 11:38:16 pm"
    },
    {
      "name":"Dom",
      "message":"Hello, Peter. What's happening?  We need to talk about your TPS reports. :D",
      "time":"Mar 17 11:37:54 pm"
    },
    {
      "name":"Milton",
      "message":"I enjoy listening to the radio at a reasonable volume from nine to eleven.",
      "time":"Mar 17 11:36:42 pm"
    },
    {
      "name":"Peter",
      "message":"Yeah! All right! :-0",
      "time":"Mar 17 11:36:24 pm"
    },
    {
      "name":"Milton",
      "message":"The radio, I can't - :'(",
      "time":"Mar 17 11:35:55 pm"
    },
    {
      "name":"Peter",
      "message":"Ok. :-0",
      "time":"Mar 17 11:35:37 pm"
    }
  ]
}
```

If a post is performed with either an empty `name` or `message` then an error response (as documented above) is returned.

## Provided Files

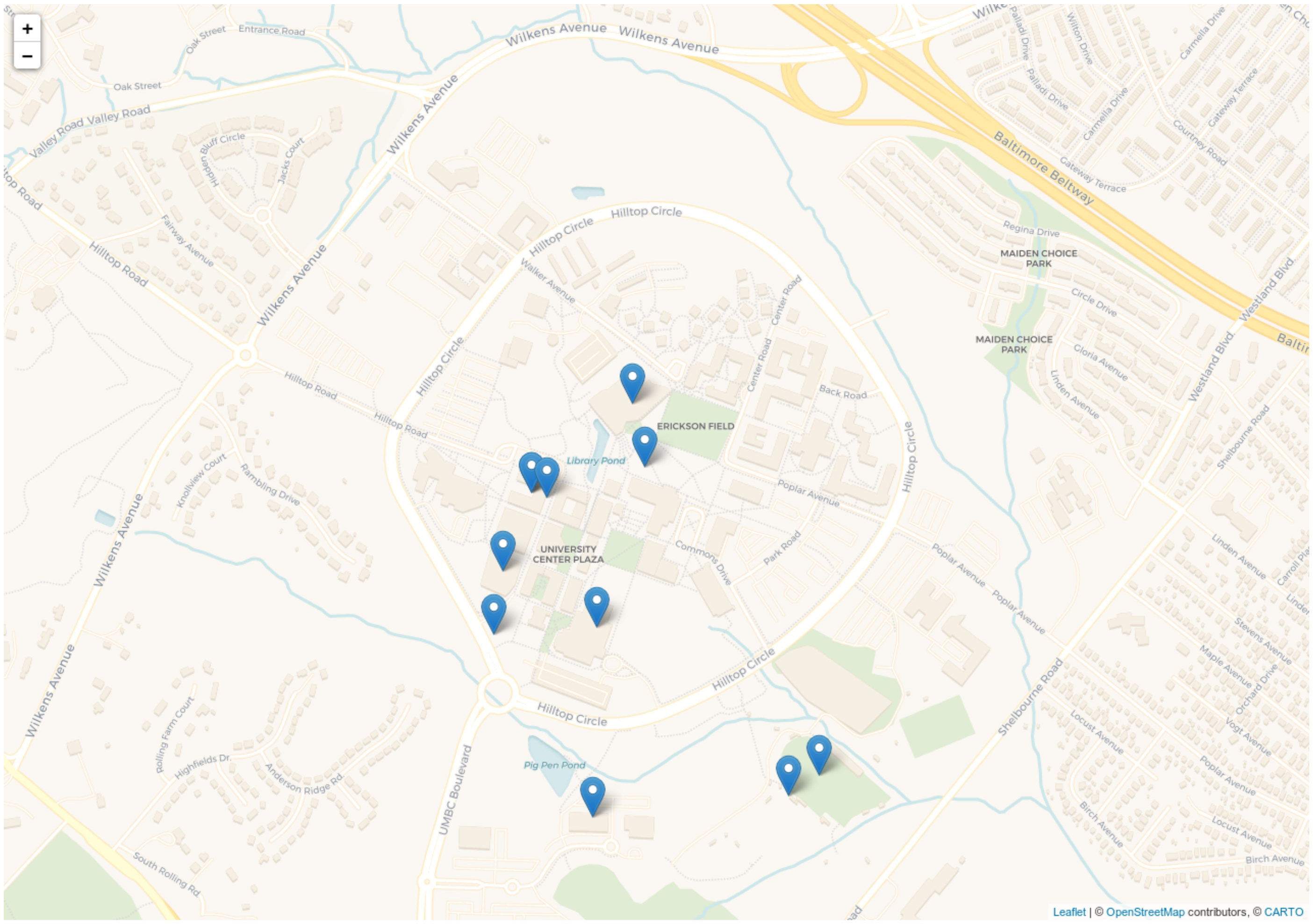
- `index.html` — this file contains the HTML for the layout shown in the screen shots - this is where you will be doing most of your work. You will add JavaScript event handlers and JavaScript functions into this file. If you wish to externalize the JavaScript in an external `.js` file feel free to do so.
- `server/shout.php` — the PHP script that you will be using for this problem (and the necessary library). So, when you create your Ajax request, it will be against `server/shout.php` which refers to the one in your directory.
- `css/bootstrap.min.css` — the Twitter bootstrap stylesheet.
- `css/custom.css` — CSS customizations for the HTML page in this problem.
- `img/white_texture.png` — image used as the background texture.
- `js/` — an empty directory (where I put my externalized JavaScript files).

The resources for this problem will be copied to your SWE directory as part of the setup script

# Maps + Wikipedia Mashup (30%)

## Task

For this assignment you will implement a open street maps (using leaflet.js) and wikipedia mashup using JavaScript. You are to implement the geographic display of landmarks in wikipedia by making an AJAX request to a provided server side component (which is making a request to Wikipedia on your behalf) and display the results of the JSON response on the map provided by leaflet.js.

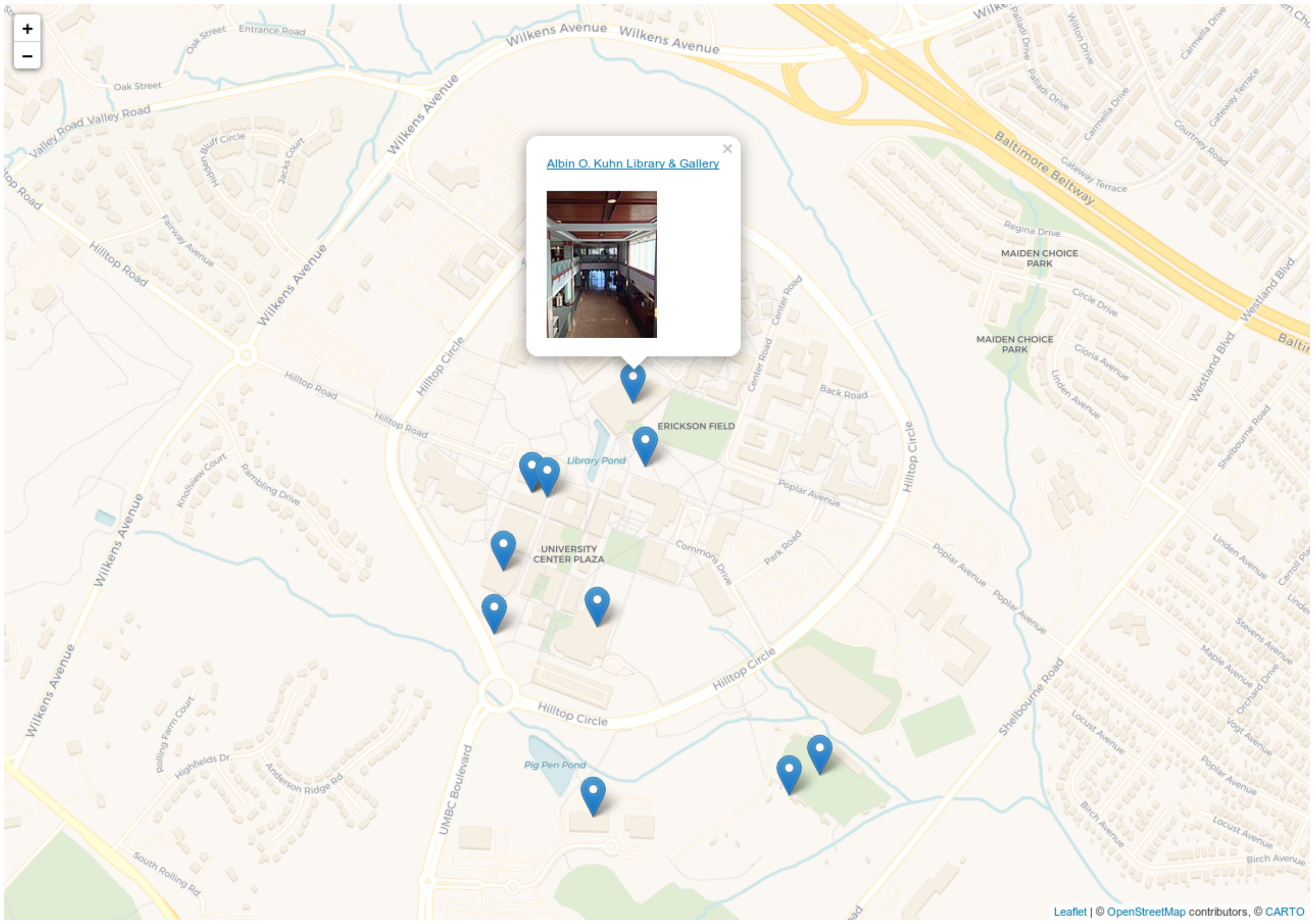


Maps Figure 1 — Initial View

## Behavior

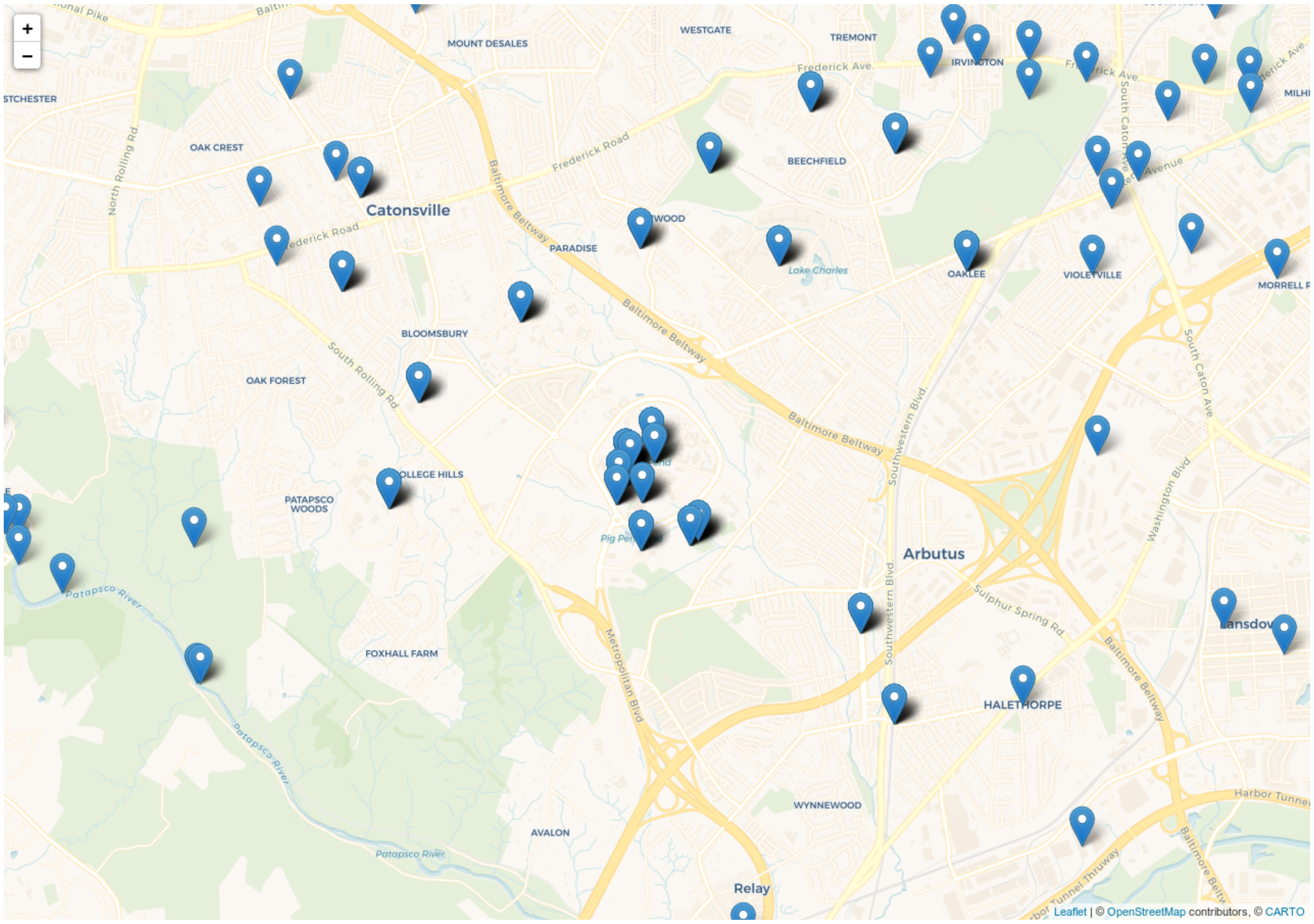
When the page initially loads it should look similar to the map as shown below. The map should be loaded and focused with some default view. I've focused on UMBC — you are welcome to focus on whatever region you like. Take note that when the page is loaded, an Ajax request should be initiated for the default user and any landmarks within the view of the current map will be returned and should be displayed as markers. **Your initial view should have some markers present, so choose your viewing location appropriately.**

Upon clicking a marker, it should open a bubble with the name of the landmark, and a photo if one is provided. The name should link tothe Wikipedia article, opening in another tab or browser window.


Albin O. Kuhn Library & Gallery
Leaflet | © OpenStreetMap contributors, © CARTO

Maps Figure 22— Info bubble

Additionally, if the map is panned or zoomed (either by grab and drag or via the navigation controls) the map will have new extents, and a new Ajax request should be made with the new edges of the map. All markers should again be cleared and replaced with the result based on the new map.



Maps Figure 6 — Panning

## Server Side Script

A server side script called `wiki.php` has been provided for this problem. This script takes the following arguments, all of which are required:

- `lat`
- `long`
- `rad`

This script is actually just proxying on your behalf, feeding your requests to the Wikipedia API. Lat refers to the latitude at the center of your map, long refers to the longitude at the center of your map, and rad refers to the radius. You must comput the radius yourself using the bounds of the map and the `distance` function in leaflet.js

The response is a plain text JSON encoded response in the format given by Wikipedia. You should be able to determine which parts you need by looking at a response.

A response from a request of the script provided can be obtained by going directly to the wiki.php script like so: http://swe.umbc.edu/~USERNAME/cs433/javascript/p3/server/wiki.php?lat=39.25560152079427&long=-76.71099543571474&rad=752 . Note that a number of characters have been URL encoded. **Again in your JavaScript, you will be pointing to your version out of your directory.**

## Provided Files

- `index.html` — this file contains the HTML for the layout shown in the screen shots — this is where you will be doing most of your work. You will add JavaScript event handlers and JavaScript functions into this file. If you wish to externalize the JavaScript in an external `.js` file feel free to do so.
- `server/wiki.php` — the PHP script that you will be using for this problem. So, when you create your Ajax request, it will be against `server/wiki.php` which refers to the one in your directory.
- `js/` — an empty directory (where I put my externalized JavaScript files).

These resources will be copied to your directory as part of the setup script

## Submission

For this assignment you will not need to use GitHub. Rather, since the setup script gave me the necessary permissions in your assignment directories I will archive what I need at midnight.

**After the due date you should under no circumstances edit the files under your `javascript` directory.** When grading, the files sitting in your directory will be diff'd against the version I archived. Any files that were detected as changed will be treated as late and that problem will not be graded. If you wish to continue to work on these files, please copy them off somewhere else (under the directories created for you where it is still locked down) and work on the copies there. **Again, do not change anything under the `javascript` directory — doing so will result in significant point loss.**