CSE 190m Summer 2012 Final Project

Your final project will be to build a x-men zoo website. This website is comprised of two pages, a web service, and accompanying helper files:

- **`zoo.php`**: Shows x-men roaming around and provides some basic information.
- **`zoo-info.php`**: Displays a single character and shows more detailed information.
- **`zoo-basic-info.php`**: A web service that provides basic info about a character.
- **`zoo.css`**: Style sheet for your pages.
- **`zoo.js`**: Javascript code for your pages.
- **`top.html`** / **`bottom.html`** / **`common.php`**: Helper files you may use if you need.
- **`makedb.sql`**: Filled in sql template if you do the optional database creation.

You can find the images directory referenced below, it contains any image resources you will need including `favicon` and validator images:

`http://www.cs.washington.edu/education/courses/cse190m/12su/final/images.zip`

You are not given any starter code, but there are a few screenshots on the course website at these locations and linked from the *final* page. The goal is **not** to match these screenshots, these are just for reference to get you started.

```
http://www.cs.washington.edu/education/courses/cse190m/12su/final/final-mac.png
http://www.cs.washington.edu/education/courses/cse190m/12su/final/final-mac2.png
http://www.cs.washington.edu/education/courses/cse190m/12su/final/final-mac-info.png
http://www.cs.washington.edu/education/courses/cse190m/12su/final/final-mac-info2.png
```

`zoo.php`:

This is your main page and should have a banner at the top of the page that gives a title for the page as well a brief description of the page. The title should be a relative link to **`zoo.php`**. The description should include information about how the user can interact with the page. At the bottom of the page should be an area that displays our 3 validator images (HTML, CSS, JS) and a line of text that identifies the author of the page, you. In between these sections there should be 2 panes.

The first pane occupies roughly **20%** of the page's width, we will call this the info pane. The info pane should initially display some placeholder text. Directly to the right of the info pane is the zoo pane, which occupies the rest of the width of the page. Both panes should be exactly **500 pixels** tall. Inside of the zoo pane should be an image for each of the characters for which there are images in the provided `images` directory. All character images are named with a prefix of *"xmen-"*.

When the page first loads, the characters should be assigned a random location within the zoo pane. Each should move randomly around inside the zoo pane. A simple way to do this would be to give each a random direction to move in and then alter that direction once they reach an edge of the zoo pane. No part of any character should ever move outside the boundary of the

zoo pane.

When the user hovers their mouse over one of the character images the info pane should be filled with some basic information about that character. This should be done without the page reloading. If the user hovers over subsequent characters, the info pane should only contain information about the last one. In other words, you should remove old info before inserting new info. The inserted info should have a heading with the character's `name` and then give a list of the following attribute names along with their value for the character; `name`, `alias`, `origin`, `gender`, and `alignment`. You should give the info pane the CSS property `overflow: scroll` so that the information is always contained in the pane.

When the user clicks one of the characters, the browser should link to **`zoo-info.php`** with a query parameter named `name` whose value is the `name` of the clicked character. For instance, if you clicked on *"Scott Summers"* you would be redirected to *zoo-info.php?name=Scott%20Summers*.

zoo-basic-info.php:

To get the needed information for the info pane in zoo.php you should make an AJAX call to a web service that you will write called **`zoo-basic-info.php`**. The web service should take a query parameter named `name`. Using the given `name`, the web service will need to query the `dcdb` mysql database (described below) and return the necessary data in JSON or XML format. Your **`zoo.php`** page must parse the returned JSON or XML and insert HTML into the info pane with the given data. If the `name` parameter was not given or the character requested is not in the database you should return an appropriate HTTP error code. **`zoo.php`** should respond to these error codes by inserting some kind of user-friendly error message into the info pane.

zoo-info.php

This page must have the same top and bottom banners as **`zoo.php`** as well as the same zoo and info panes, though the content inside them will differ. This page should receive a parameter named `name`. The zoo pane should contain the name of the specified character as well as a large image of the character. The info pane should be initially filled (without AJAX) with detailed information about the character. You should give the info pane the CSS property `overflow: scroll` so that the information is always contained in the pane. The character information in the info pane should contain three sections, the data for which is again in the `dcdb` mysql database:

- The same basic info as given by **`zoo-basic-info.php`**.
- A list of the superpowers that the character possesses.
- A list of the character's allies.

Above the character info should be a drop-down list of all of the characters for which there are images in the `images` directory. To the right of this drop-down should be a submit button that when clicked submits to **`zoo-info.php`** with the selected character's `name`. If this page is not given the `name` parameter then it should default to your favorite character.

This page should be the result of running a PHP script, not a collection of things gathered through AJAX. The drop-down list and the accompanying button should be an HTML form that submits to **`zoo-info.php`**.

Appearance:

Your pages need to satisfy the few CSS requirements described above, but other than that the CSS is all up to you. You should give your page a consistent theme and try to demonstrate some of the skills you have learned. There is defined look to go for or number of lines of CSS you must write, but we do want the page to be stylized. A page with little more than browser default stylings would be marked down.

If you must, go ahead and try to emulate the styles shown in the provided screenshots, but we hope that you take the creative approach!

The dcdb database:

The dcdb database has 4 tables. These tables have the following schemas:

`supers`

id	name	alias	origin	gender	alignment
1	Charles Xavier	Professor X	New York City, NY	F	good
2	Bobby Drake	Iceman	Long Island, NY	M	good
...					

`abilities`

id	name
1	Telepathy
2	Super Intelligence
...	

`powers`

super_id	ability
1	1
1	2
...	

`connections`

super_id	other_id	kind
1	2	allies
1	5	allies
...		

The data in first are tables ar fairly straightforward. **`supers`** describes each character, **`abilities`** describes various powers a character might have and **`powers`** links those tables together by their ids. For instance, the tables above show that Charles Xavier has telepathy as well as super intelligence. The fourth table describes a one-way relationship between character `s1` and character `s2`. The `kind` of the relationship is either *"allies"* or *"foes"*. The connection is one way, so the first record above does **not** imply that Bobby Drake considers Charles Xavier to be an ally, only that Charles Xavier considers Bobby Drake to be an ally. Understanding this distinction will make one of your queries much easier.

You can test your queries using the same query tester page that we used for other databases in the course. `dcdb` is not a default database option on the page, instead you must write `dcdb` into the *other* option. You must use the same mysql username and password we provided you with earlier in the quarter.

Database Creativity (optional):

If you desire you may make your own database and use the information from it rather than the x-men data we have setup for you. To do this, you must use the provided **`makedb.sql`** file.

`http://www.cs.washington.edu/education/courses/cse190m/12su/final/makedb.sql`

This file is a template of how we want the database to look. You may not change the columns inside of the predefined tables, but you may add your own additional tables. The only thing you may change is the content and number of records and the names of the columns and tables. You must have at least **10** records in each table. Part of the impetus for giving you a template is so that you cannot create a database that diminishes the complexity of the SQL queries you will need to write. The queries you write should utilize all of your tables and be similar in complexity to those needed for the provided data. If you are unsure about whether you have satisfied this criteria, email your TA or the instructor and we will be happy to tell you.

If you choose to make your own database you must also turn in new images that correspond to your data set.

This is intended to be a way for you to personalize your page and have some fun with it. This should not be something you spend hours on and will not be worth any points.

When you finish filling in the **`makedb.sql`** template you need to execute it on webster. To do this, log in to webster in a terminal or console window. Next, log in to mysql and run `source` *`path/to/`*`makedb.sql;`. Your terminal session might look something like this:

```
>>>: ssh UW_NET_ID@webster.cs.washington.edu
UW_NET_ID@webster.cs.washington.edu's password:
>>>: mysql -u UW_NET_ID -p
Enter Password:
>>> source ~/public_html/final/makedb.sql;
```

Additional Requirements:

- PHP code that connects to the database must turn exceptions on.
- You must use strict mode in your JS code.
- Your HTML/CSS/JS must pass their respective validators
- Use absolute URLs if you link to any resources other than those provided

Development Strategy:

We suggest that you tackle tasks in this order:

- write the HTML/CSS for zoo.php
- add PHP to zoo.php where needed
- create your web service and ajax code with dummy data
- create the HTML/CSS for zoo-info.php with dummy data
- write your SQL queries for your web service and zoo-info.php page
- add the HTML form to zoo-info.php
- add the hovering behavior to zoo.php
- add the movement behavior to zoo.php
- reduce in-file redundancy
- reduce cross-file redundancy

- cleanup your style, indentation and comments
- read the spec one last time to make sure you did not miss anything

For reference, our solution has this many lines:

- HTML: 38 (33 substantive)
- CSS: 55 (50 substantive)
- PHP: 174 (106 substantive)
- JS: 75 (60 substantive)

It is perfectly fine and expected that you have more or less lines as long as you have satisfied the above criteria and feel good about your page.

Hints and Help:

The final is an individual final, and is subject to the same collaboration policy detailed in the syllabus for assignments.

The IPL will **not** be open during the last week of class, nor will we be holding office hours in order to answer questions about the final. There will be a **message board** open for questions and as always you may email us. However, this is a final project, and we will be treating it as such, which means you may not get the same kinds of answers that you are used to on assignments.

Here is a brief list of some things that you may want to review, look into, and consider before you start implementing:

- PHP `glob()`
- PHP `htmlspecialchars()`
- PHP PDO `quote()`
- CSS `relative`/`absolute` positioning
- CSS `cursor` property
- HTML semantic tags
- jQuery

Grading:

The specification of the project is deliberately vague in certain areas. We want to see that you know how and when to use the techniques we have seen in class. We want to see how you build a complex website without forcing your hand too much. Here is an incomplete checklist of things your solution should demonstrate

- elimination of in-file redundancy (proper use of PHP/JS functions, factored CSS styles)
- elimination of cross-file redundancy
- proper separation of content/presentation/behavior
- good consistent code style
- appropriate HTML tag choices
- an understanding of web services / AJAX
- an understanding of HTML forms / query parameters
- well constructed SQL queries

- when alignment / float / position is necessary
- good use of jQuery syntax
- proper and consistent indentation and comments
- robust code in the presence of database errors, AJAX errors, etc
- PHP/SQL code that tries to protect against the effects of malicious user input
- features of a usable page (title, favicon, decent CSS stylings, etc)

Our grading criteria are more complex, but they are these kinds of high-level ideas that we are concerned with grading. Part of your grade will come from implementing the described features and functionality, but you will not be graded on the exact appearance or implementation of your page, other than the few guidelines detailed above. The expected output images are just aids in communicating a desired result.

Turning It In:

You will submit your final in the same fashion as a homework assignment. Make sure all of your files are on webster and are working. Then submit them to grade-it on the course website through the *final* page.

Do not place a solution to this final on a public web site. Your final project should be accessible on the web with these urls:

```
https://webster.cs.washington.edu/YOUR_UWNETID/final/zoo.php
https://webster.cs.washington.edu/YOUR_UWNETID/final/zoo-info.php
```