Website Organization and Administration

School: SDSM&T
Course: CSC 468: GUI Programming
Semester: Spring 2016
Professor: Dr. John Weiss
Assignment: Programming Assignment 3
Team: Daniel Andrus, Austin Rotert, Christian Sieh

Contents

Installation and Configuration

To use this code during development, you will need to do some initial setup.

1. Enable Apache URL rewriting module
2. Enable .htaccess files

1. Enable Apache URL rewriting module

Apache must be configured to allow URL rewriting in order for this code to work correctly. To enable URl rewriting, follow the below steps:

1. Open your Apache configuration file in a plain text editor. (Google it if you don't know where it is.)
2. Find the line containing the text "LoadModule rewrite_module".
3. If the line has a "`#`" at the beginning, remove the "`#`" to uncomment the line.
4. Save the file and restart Apache.

2. Enable .htaccess Files

Apache must be configured to allow the contents of .htaccess files to override the default Apache configuration. This is necessary for our code to execute properly.

1. Open your Apache configuration file in a plain text editor. (Google it if you don't know where it is.)
2. Find the line containing the text "AllowOverride None". If this line does not exist, then .htaccess files may already be enabled. If this line is found, then it should exist between opening and closing "`<Directory>`" tags.
3. Change the line to say "AllowOverride All"
4. Save the file and restart Apache.

Authentication and Permissions

With our backend we are providing a convenient interface for checking for logged in users and getting their permissions. We are providing this API as a PHP abstract class that will be accessible from all pages accessed using the code in this repository. With this API, you will be able to:

1. Log a user in
2. Log a user out
3. Check if a user is already logged in
4. Get the currently logged in user
5. Get information on the current user
6. Check if a user has certain permissions
7. Declare custom permissions

Explainations on most of these subjects will be followed by an example

Below is an outline of the PHP classes you will be using to perform the above functions. The functionality of these classes will be developed over the course of this project and will be uploaded to this repository.

```
abstract class User
{
    static boolean authenticate(string $username, string $password);
    static boolean isAuthenticated();
    static User    getCurrentUser();
    static void    declarePermission(string $permission);
    string         getUsername();
    string         getDisplayName();
    boolean        hasPermission(string $permission);
    void           logOut();
}
```

1. Log a User In

To log a new user in, simply call the `authenticate(string $username, string $password)` static function, which will return `true` if the provided username and password strings successfully authenticated or `false` if it failed. If authentication was successful, calling `getCurrentUser()` will return the newly logged-in `User` object.

2. Log a user Out

After checking if a user is logged in to the site and getting the current `User` object, you can log the user out by calling the `logOut()` function. Calls to this function should always work.

1. Check if a User is Already Logged In

To check if a user is currently logged in to the site, simply call the `isAuthenticated()` static function, which will return `true` if a user is logged in or `false` if no user is logged in. An example of this is provided below.

2. Get the Currently Logged in User

After checking if a user is logged in to the site, you can get the `User` object representing that user by calling the `getCurrentUser()` static function. This function will return the `User` object of the currently logged in user, or it will return `null` if no user is logged in. An example of this is provided below.

3. Get Information on the Current User

After checking if a user is logged in to the site and getting the current `User` object, you can get their username (e.g. student ID for students, faculty username for faculty) by calling `getUsername()` on the object. You can also get their display name (e.g. real name) by calling `getDisplayName()`. An example of these is provided below.

3. Check if a User has Certain Permissions

Permissions will be represented using strings. Your own team can design what permissions your pages will need to use, then you can check if a user if has a permission by calling the `hasPermission(string $permission)` function. An example of this is provided below.

4. Declare Custom Permissions

Before checking if a user has a certain permission, you will need to declare a permission at the top of a file. This can be done by calling the `declarePermission(string $permission)` static function. An example of this is provided below.

The permission string can be whatever your team decides; it is entirely up to you. They can be as specific or vague as you want, but please keep in mind the following tips:

1. Use a simple naming convention
2. Make it clear what they do
3. Avoid "subtractive" permissions (*permissions should **give** access, not take it away*)
4. Generate permissions strings for each subpage if necessary, i.e. permissions for a specific student organization, etc.

Example

```
<?php
// Declare any permissions to be used at the top. These permission strings
// can be built dynamically based on the page you're on and can have
// as much granularity as you see fit.

// permissions that apply to all organizations
User::declarePermission('student-org.*.edit-details');
User::declarePermission('student-org.*.edit-member-list');
User::declarePermission('student-org.*.edit-officer-list');

// permissions that apply to specific organization, can be auto-generated
User::declarePermission('student-org.game-dev.edit-details');
User::declarePermission('student-org.game-dev.edit-member-list');
```

```
User::declarePermission('student-org.game-dev.edit-officer-list');

// Check if a user is logged in
if (User::isAuthenticated())
{
    // User is logged in, get current user
    $user = User::getCurrentUser();

    echo '<p>Hello, '.$user->getDisplayName().'</p>';
    echo '<p><a href="../profile/'.$user->getUsername().'">View profile</a></p>';

    echo '<form action="update-group.php" method="post">';
    echo '<input type="hidden" name="org-id" value="game-dev" />';

    // Display form fields based on user permissions
    if ($user->hasPermission('student-org.game-dev.edit-details')
        || $user->hasPermission('student-org.*.edit-details'))
    {
        // Insert whatever logic necessary here
        // ...
        // ...
    }

    if ($user->hasPermission('student-org.game-dev.edit-member-list')
        || $user->hasPermission('student-org.*.edit-member-list'))
    {
        // Insert whatever logic necessary here
        // ...
        // ...
    }

    if ($user->hasPermission('student-org.game-dev.edit-officer-list')
        || $user->hasPermission('student-org.*.edit-officer-list'))
    {
        // Insert whatever logic necessary here
        // ...
        // ...
    }

    echo '<input type="submit" value="Update" />';
    echo '<button onclick="window.history.back()">Cancel</button>';
    echo '</form>';
}
else
{
    // User is not logged in, display error message
    echo '<p class="error">You must be logged in to view this page!</p>';
}
?>
```

Development Guidelines and Requirements

The goal of this project is to build a website framework upon which the other teams in this assignment can develop their parts of this project. This document contains guidelines and specifications when generating HTML content for pages and for organizing your page files.

1. All page files and content placed in that page's assigned subdirectory
2. Only use relative paths
3. Limit use of style tags and attributes
4. Use standard HTML structure for navigation
5. Do not use tables for layouts
6. Avoid size attributes
7. Name files using common convention

1. All Page Files and Content Placed in That Page's Assigned Subdirectory (Required)

Each distinct page for the site will have its own unique directory under the `pages` directory. All files relevant for the page should be included in that page's directory.

2. Only Use Relative Paths (Required)

When referencing links or images on a page, always use relative links, never absolute links. This will make incorporating your team's code into the rest of the site far easier.

Example of relative links (RECOMMENDED):

```
<img src="images/upload.png" />
<a href="../submit/">Submit</a>
<link rel="stylesheet" type="text/css" href="css/styles.css" />
<script type="text/javascript" src="../shared-scripts/myscript.js"></script>
```

Example of absolute links (DISCOURAGED):

```
<img src="/pages/my-page/images/upload.png" />
<a href="http://dev.mcs.sdsmt.edu/~1234567/pages/submit/index.php">Submit</a>
<link rel="stylesheet" type="text/css" href="/~1234567/pages/my-page/css/styles.css"
/>
```

3. Limit Use of Style Tags and Attributes

When styling your elements, refrain from using the `style=""` attribute or from putting `<style></style>` tags in your HTML. Move these rules to a separate CSS file and apply them to elements using CSS classes.

Example of external CSS (ENCOURAGED):

`index.php` contents:

```
<link type="text/css" rel="stylesheet" href="styles.css" />
<p class="favorite"> This is my most favorite paragraph! </p>
<a href="../submit/" class="submit-button"> This is a link that looks like a red
button! </a>
```

`styles.css` contents:

```
p.favorite {
  color: red;
  font-size: 1.25em;
}
a.submit-button {
  display: inline-block;
  margin: auto 1em;
  padding: 6px 10px;
  background-color: red;
  box-shadow: 0px 2px 6px 0px rgba(0,0,0,0.5);
}
```

Example of inline CSS (DISCOURAGED):

```
<p style="color: red; font-size: 1.25em;"> This is my most favorite paragraph! </p>
<a href="../submit/" class="red-button"> This is a link that looks like a red
button! </a>

<style>
a.red-button {
  display: inline-block;
  margin: auto 1em;
  padding: 6px 10px;
  background-color: red;
  box-shadow: 0px 2px 6px 0px rgba(0,0,0,0.5);
}
</style>
```

4. Use Standard HTML Structure for Navigation (Recommended)

If you need to include navigation between multiple pages in your section of the site, please structure the HTML of your navigation menu as follows:

```
<nav class="section-nav">
  <ul>
    <li><a href="link-to-page">Link 1</a></li>
    <li><a href="link to page">Link 2</a></li>
    <!-- repeat as necessary -->
  </ul>
</nav>
```

Not all teams will need this, but if your pages does, please place it at the very top of your HTML content.

If you would like this menu to navigate to different sections on the same page, use the class `page-nav` instead of `section-nav`.

5. Do Not Use Tables for Layouts (Required)

Please avoid using `<table>...</table>` tags for laying out your page. Use `<div>` elements for grouping elements in a related section and `<ul>` tags for displaying lists of elements. These are far easier to style than tables.

If you need to display tabular data, such as numbers and calculations, using a table to organize the data is okay.

6. Avoid Size Attributes (Recommended)

When adding images and other elements to your page, avoid including sizing attributes, such as `width=""` and `height=""`; this is something that should be placed in the CSS stylesheets.

7. Follow File Naming Conventions (Required)

When naming files, please use all lowercase names with words separated by dashes (–). **Do not include spaces in your file names!**

- Encouraged names:
  - *index.php*
  - *my-javascript-file.js*
  - *images/logo.png*
  - *images/secondary-logo.png*
- Discouraged names:
  - *index.HTML*
  - *MyJavascriptFile.js*
  - *Images/Logo.png*
  - *Images/Secondary Logo.png*