README.md

Final-Project

The final project will be a fully functional database backed website. The content and purpose of the website is largely at your discretion.

Database Requirements

You need to have at least 1 table with 3 non-primary key attributes. Pretty simple right? The focus is not on the database. And you can certainly have a more complex database if you want, but the database structure will not be the focus of the grading.

All database interaction that takes in any sort of user input must use bound parameters with prepared statements.

User Account Requirements

You will need to support user accounts and you have to implement this yourself. You cannot use Oauth or some other 3rd party library to implement accounts. You can use it as a supplement. So you can allow users to register using Oauth OR your implementation of accounts but you can't require use of a 3rd party library to facilitate user accounts.

A user must be able to add rows to a database and see the results and they should be tied to users. For example, if your website was making a weekly calendar, users might be able to save events to the database (each event requiring at least one new row in a database table) and only they should be able to see those rows. It is fine to allow users to share their calender if they wanted, but you need to support a user adding content only they can see. When a different user logs in, they should see different content.

The login functionality must use Ajax for error handling. If a user enters an invalid username and password, they should be notified via an Ajax call rather than a full page reload.

Other Requirements

The page should look polished. In previous assignments the layout was strictly specified or unimportant. For the final project you do not need to be a design wizard, but you should have a style to your pages that is distinct from the default rendering of HTML elements and forms.

User unfriendly error messages are to be avoided at all costs. If they enter in a string where there should have been a number, let them know that they were supposed to enter a string rather than a number. Under no circumstances should they see the text "Error 500" nor should they be notified that "PHP expected an integer in line 59".

You should use 3rd party libraries or APIs where appropriate.

Complexity

Because this is very open ended, it can be difficult to know how complex of a website is needed. You should do most of the following unless you have a very compelling reason to do otherwise:

- Have a place to handle non-string input (dates, prices etc).
- Have data entry requirements from users that can be validated and are validated (dates within a range or text of a minimum or maximum length)
- Use Ajax where appropriate (and if you don't have several places where it is appropriate, your application may be too simple). Some examples include: verifying a username is available, displaying changes to data without reloading or displaying success or error messages when an operation is complete.
- Be able to handle media of some sort. Maybe it is working with maps, maybe it is allowing users to upload or manipulate images. This is an ideal place to bring in some 3rd party libraries to assist.
- Be able to share information between users of your site if they want to.

These may not all make sense for all sites, but if a lot of them don't make sense you are probably not going far enough with your complexity.

Grading

Projects vary from term to term so there is no real test plan to publish:

- Unhandeled errors will be a major deduction.
- HTML and JavaScript errors that are of your own doing will be a major deduction (If a library you are using causes these and it is unavoidable, that is acceptable but should be documented).
- More complex websites will have a somewhat lower standard in terms of user interaction complexity. So if you are doing a really simple website, it better work really really well. If you are doing a project that is vast in scope and I can find a few bugs, that will not be a major issue.
- The main area we will be testing fairly uniformly is user login and account creation.
  - We will make an account.
  - Try to make an account with the same name (this should fail).
  - Login and input some data on the first account.
  - Make a 2nd account.
  - Login with the 2nd account, make sure changes from the first account don't exist.
  - Input some data on the 2nd account.
  - Log back in on the first account and make sure it is how we left it.

Example Ideas

- A simple message board.
- A database of movies letting users add movies or movie reviews.
- A road trip planner where users can map a trip and keep track of where they want to stop.
- A simple wiki implementation (simply using a 3rd party wiki library would not be sufficient for this)