Kevin Ma
Dan Cosley
INFO 1300
November 14th, 2014

Project 3 Final Milestone Rationale

Different Versions

feedback_noclient.php is the version without client side validation. feedback.php is the version with both client and server side validation. If the submission is successful, both of the pages redirect to handle_form.php.

Client Side Validation

For the radio input, selection and option element and text elements, the **required attribute** was used to make sure the client completed the questions. These questions did not require textual information from client; they were completion based, so the required attribute was suitable for this task.

The required attribute could not be used for the checkbox input type; if a checkbox field is given a required field it must be checked. The question became how to make sure the client checked at least one box in a group of many. A **javascript** function was called upon submission of the form, to check if at least one box was checked for each checkbox group.

The email was validated by making the input type an email. While this does not completely prevent the user from inputting an invalid email, it will supplement the server side validation that will be implemented in the next milestone.

Server Side Validation

For server side validation, the first thing to do was to make sure that all required fields had been filled in, checked, or selected.

When the user selects an "Other" checkbox, they must now specify what they meant in the input field following the checkbox. This is a key difference between the server side and client side validation: originally I thought this would be a hassle for the user, but I decided that this information was important enough so that it should be required. It is fine for the user to type into the "other" text input field, but not to check the "other" checkbox, as the user's intent is clear in this situation.

To handle spaces and empty fields, **trim()** was called on all variables created from fields that the user could input text into. To prevent malicious cross site scripting attacks, I made sure **htmlspecialchars()** was called on all variables created from fields that the user could input text into.

The optional email input was validated with **filter_var()**. If an email was entered, it was filtered with FILTER_VALIDATE_EMAIL. This was important as the client-side email validation could be bypassed easily by a string followed by an "@" followed by a string.

.

Presentation of Errors

On the client side, errors are presented through pop ups for most fields, and alerts for checkboxes that inform they user they must check at least one box. These are very noticeable;

On the server side, there is a red header that tells the user to correct their errors, and errors appear right above the question. Initially I put all the error messages together, but realized the user would have a hard time pairing each issue with a question, especially since many of the error messages were so similar. I changed the errors to appear above each question and customized the error messages to be more specific, so the user understands which questions had problems. The errors are shown in red as to be easily noticeable, as well as to differentiate them from the content of the form.

If the submission is successful, the user will be redirected to the handle_form page. I did not feel the need to echo the user's submissions back at them as this is a survey form, however if the user chose to enter their email, I echoed the email back, as the user might have multiple email accounts and not be sure which one he submitted.