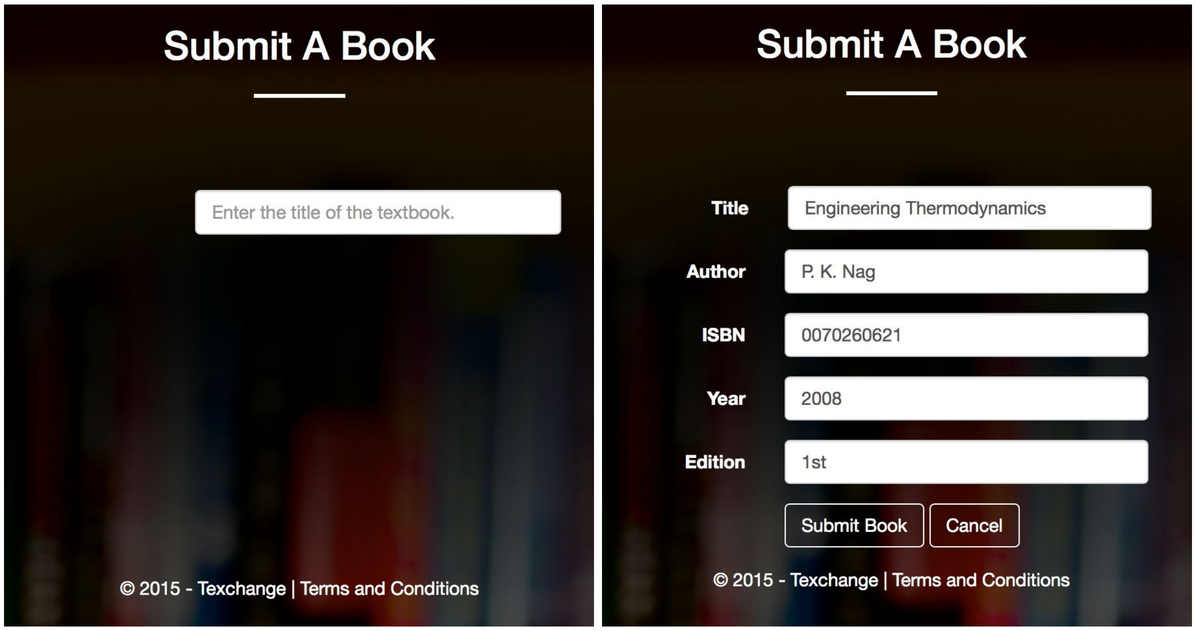
Bug Report: Create Books Bugs

Background

The “Create Book” form, that allows a user to add a book to the system, had some issues at the end of Sprint 3. The form has a field for “Title”, “Author”, “ISBN”, “Year”, and “Edition”, and additionally has a button to submit the form, and a cancel button to return to the home page. Initially only the title field is visible, and the other elements become visit once the title field has been filled. This allows a clear autocomplete menu to be displayed based on the title as a query to the Google Books API. The fields then all already filled when they become visible, unless we can’t find the book for the user in which case they can enter these details manually.


Submit A Book
Enter the title of the textbook.
© 2015 - Texchange | Terms and Conditions
Submit A Book
Submit Book
Cancel
© 2015 - Texchange | Terms and Conditions

Title	Engineering Thermodynamics
Author	P. K. Nag
ISBN	0070260621
Year	2008
Edition	1st

The “Create Book” form, empty and filled. Notice the hidden fields and buttons that become visible once a title is entered.

The first issue with this was that if a user typed in a title and quickly pressed the ‘enter’ key, the form would attempt to submit either with blank fields, causing errors, or with details that hadn’t been checked. This was recorded as “Pressing Enter on Book Submit Form” in the bug tracking system.

The second issue, was that if a user input invalid data into a field (or was experiencing the first issue), the errors would be displayed but the fields were hidden again and they were unable to correct them without clearing the form and starting again. This was identified as “Hidden Book Entry Fields” in the bug tracking system.

Triage

Triage for this bug was straightforward since it was a flaw in the original feature design. Reassessing the decisions regarding functionality was required.

“Enter” submits incomplete “Create Books” form

There were a number of ways to fix this issue. One option is to detect an incomplete form in the Controller and to return a view with the form pre-filled to its current state. This requires waiting for a page load, and somewhat breaks the Model View Controller design pattern, since this could be considered a problem that relates solely View.

A better option was to disable the keypress handler the submits the form when the enter key was pressed. This requires that the user press the submit button to submit the form, and since this button is not visible until the whole form is visible, this solves the problem.

This was achieved with the following simple jQuery snippet.

```
// Disable submit with enter key, since most of the form is
// initially hidden
$("#submitBookForm").bind("keypress", function (e) {
        if (e.keyCode == 13) {
                        return false;
        }
});
```

This fix is included in commit c4fc714.

“Create Books” elements hidden, even with validation errors

This problem initially seemed fairly straightforward to fix: Don’t hide form elements if there are validation errors. It turns out it wasn’t quite that simple, partly due to the page load sequence and partly due to the combination of languages being used.

In order to optimise page load time and to ensure that target objects actually exist, most scripts are run on page load. This means that if the hidden elements were hidden with JavaScript, they would show for a brief moment before disappearing as the page loaded. For this reason the elements were hidden with CSS, by setting their “display” attribute to “none”. This is easily removed via script when we want to show the element, as follows:

```
$('.auto-hidden').show();
```

For this reason, it made more sense to show the elements if they needed to be shown by scripting once the page loads. The plan was to let the page load, check whether there were errors (and the whole form needed to be shown), and then show the required elements with jQuery.

Checking for errors was easy enough, since the ViewData class can tell us about the model state. Something along these lines should suffice:

```
if (ViewData.ModelState.IsValid) { … }
```

Seems easy! The problem is that this is Razor markup (since we're using C# classes), which doesn't translate simply into JavaScript or jQuery. A number of possible solutions were attempted, including using Razor markup to dynamically remove the inline CSS "display: none" depending on the modelstate, but of course depending on the asynchronous page loading pipeline this gave inconsistent results.

Eventually it was identified that we could either rewrite the hide/show functionality through Razor markup, or find a way to bridge the C# model result to JavaScript. In order to keep the project moving, the second option was selected. The following Razor markup was implemented:

```
@functions {
      string modelStateValid()
      {
            if (ViewData.ModelState.IsValid)return "true";
            return "true";
      }
}
```

Which could be used within the JavaScript as follows:

```
if (!errors) {
        $('.auto-hidden').show();
        $('.spinner').hide();
    }
```

This resulting code is fairly short and concise, but it's functionality required consideration and understanding of jQuery, JavaScript, Razor/C#, HTML and CSS, all for the one simple change.

These changes were included in commit 8a48121.

Result

The result of these bug fixes can be seen in the live version of the “Create Book” form. This is directly accessible from http://texchange.xyz/Books/Create.

Future Improvements

If the user doesn’t have a mouse, or doesn’t wish to use one, then submitting the form is not as simple as it could be. It would be ideal if the user could press the enter key to submit the form, but only once all the fields have been filled. This could be fixed by returning ‘true’ if the required conditions are met.