Yasmine Bencheqroun

07/11/2016

**IBBR Equipment Inventory Scanning Project Documentation**

***Intro***

The purpose of this project was to create a method that would be easier, quicker, and more efficient for Facilities and IT personnel to perform a yearly update all of the Equipment Inventory items in the IBBR database.

To do this, a new Drupal module was created - IBBR Inventory Scan. This module adds a custom form to be used while scanning equipment from mobile devices. In addition to the module, the existing equipment content type was modified and several Views were made.

***Entities Involved***

* Bluetooth Barcode Scanner
* Any mobile device
* A computer (optional)

***IBBR Inventory module files***

* **ibbr\_inv.info -** contains the basic information regarding the module file itself**.**
* **ibbr\_inv.module** - contains the custom module utilized to make the pages [www.ibbr.umd.edu/inventory\_scan](http://www.ibbr.umd.edu/inventory_scan) and [www.ibbr.umd.edu/inventory\_scan/add\_item](http://www.ibbr.umd.edu/inventory_scan/add_item).
* The **templates** directory contains a file named **page—inventory\_scan.tpl.php** which provides functionality to override the original ibbr.umd.edu website theme to display our custom form in a way that’s easier to work with on mobile devices.

***Functions Used***

I used several Drupal **hooks** in creating the module code

* **Hook\_menu** – used to define a URL and specify what code runs when that URL is visited
* **Hook\_permission** – used to set a permission for only Facilities and IT personnel to access the page
* **Hook\_form\_submit** – used to define all of the buttons needed for the forms
* **Hook\_form** – used to define the components of the inventory\_scan page
* **Hook\_theme\_registry\_alter** – used to override the original IBBR website theme for the specific URL’s

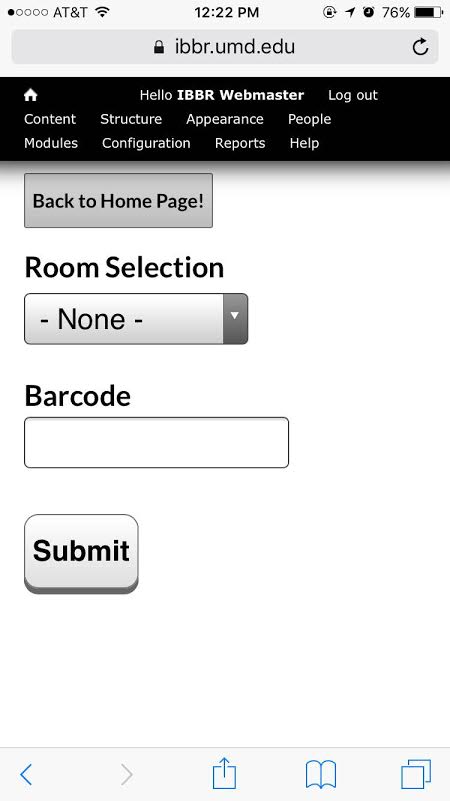
Other **helper functions** were created to respond to hook events and to break the code up, these will be described below.

* **Ibbr\_inv\_add\_new\_item** – used to generate the creation of a new node (new item)

***Overall Module Content***

We can think of the module as consisting of two separate custom forms, the main inventory scanning form and the sub form that is used when a new item needs to be added to the inventory while scanning.

*Inventory\_scan*

****

The inventory\_scan page includes the room selection widget, the barcode text field, and a submit button.

***Step 1***

**Challenge**:

*Problem*: The display for the user to choose what room they were in was unorganized.

*Solution*: After researching it, I was able to utilize the taxonomy term for ‘room’ and the **field-widget-taxonomy-shs** to display the room selection portion of my page using hierarchical organization.

***Step 2***

In the rest of my form, I was able to create a text field for the barcode that read in the barcode as text and matched it with the **tag number** in our database.

**Challenge**:

*Problem*: When an item is scanned, the user should see a summary of the item.

*Solution*: The last part of this form contains the creation of the submit button that calls an **ajax function** in another **callback** function in order to allow the user to see changes on the page without a refreshing of the page! Specifically, a summary of the information (like the manufacturer, serial number etc.…) of whatever barcode that has been scanned. Once the button is clicked, it calls the callback function which then triggers the appearance of a summary below it (**IF THE BARCODE IS FOUND**) without a refreshing of the page. I found that the summary itself can be displayed using a handy **ajax\_command\_replace** function that allows certain entities to be replaced on a page! Therefore when a new item is entered, it will update the String **$info** that is the summary, without refreshing the page.

**Challenge**:

*Problem*: When the preview is added, it appears below the current page content so the user can’t see it without scrolling the browser window.

*Solution*: Within the callback function, to automatically trigger a scroll down action for the user to view the summary, a hint of **JavaScript** and its function **ScrollIntoView** was inserted in the HTML being returned! This was *a concise and easy method to take advantage of the functionality that the JavaScript language provides without having to create a function of your own or write it in php*!

***Step 3***

After this, the user is able to either **Confirm** the item given the information at hand, or **Flag** the item as they wish.

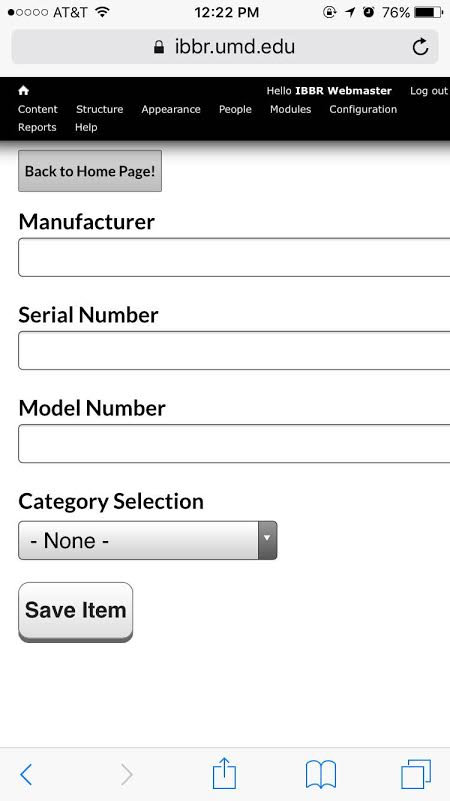
Both of these buttons are defined in the **hook\_form\_submit** function that gets called automatically by Drupal with allows them to do what they have to do. For example the Confirm button will update the **Time Stamp**, update the **Room Number** (if it changes) and also redirects the user to the original page while **saving the previous Room Selection preferences** entered in originally.

**Challenge**:

*Problem*: After scanning an item, the user has to re-enter the room number. We’d prefer the room number to stay set until manually changed.

*Solution*: Saving the **room preferences** was significant because it was an improvement that was going to make the user’s life easier instead of having to make them re-enter it every time they scanned an item. In the **ibbr\_inv\_form** function, the **room2** variable is set as the default in order for the program to remember the previously selected room. Thus when **redirection** occurs in the **hook\_form\_submit**, it will read in **room2** and remember it!

*Inventory\_scan/add\_item*



The inventory\_scan/add\_item page requires these components needed to fully create the new item in the IBBR equipment inventory database.

***Step 1***

**Challenge**:

*Problem*: What if the barcode is not found?

*Solution*: If the barcode is **not** found in our database already, then this must trigger a new type of button that is defined called “**Add new item**”. This too is defined in the **hook\_form\_submit** function with the **Confirm** and **Flag** buttons.

***Step 2***

If the item is not found and the “**Add new item**” button is clicked, the user has to be redirected to a new page which displays a **new form**. Using **$form\_state[redirect]** in my code, I am able to redirect to another page. I then created my new form as well, that is described below.

**Note**: This was an opportunity for me to utilize the previous similar form I used for **inventory\_scan** since most of the fields were Text Fields or a Simple Hierarchical Select!

***Step 3***

At this point, the **callback function** for this URL, (calls a **hook\_form** after the **submit button** on this page is pressed)is used to create the form allowing the user to enter in a **Manufacturer**, **Serial Number**, **Model Number** and **Category** for the corresponding barcode that was not found.

**Note**: Keep in mind that the **Room Number** and the **Barcode** are still arguments that are not forgotten in this step. They are **not** shown but they are just initialized as ‘**hidden elements**’ and are passed in the URL in order to match the information entered to the specific **Barcode** and **Room** **Number** it was found in.

**Note**: Hidden Elements are those variables that contain information that can be passed in as arguments (in the URL in our case) without actually being shown on the page.

Once this call back function calls the **hook\_form**, this generates the creation of the new node (new item) using the information entered! This also redirects the user back to the main page with the **Room Selection** saved.

***Miscellaneous***

*Theme override*

**Challenge**:

*Problem*: The pages needed to appear neater and less convoluted with the IBBR logo, headers, menus etc.

*Solution*: We had to override the IBBR’s original website theme so we could get rid of the unnecessary menu/logo etc. In order to do this, we created a **templates** directory in the **ibbr\_inv** directory. Then we create the file **page—inventory\_scan.tpl.php.** This file originally contained the IBBR website theme but after deleting a lot of the code we didn’t need, we were able to get rid of the unnecessary header, menu, and logo.

In the actual **ibbr\_inv.module**, we used a function called **theme\_registry\_alter** that allowed us to utilize our **tpl** file as an override against the original IBBR website theme file.

**Note**: Drupal scans for any files that start with ‘page---‘ and it finds the URL directly after that in order to know what URL will be using this theme!

\*The reason for all this? It makes all of our code so much more compact and organized instead of having to override the theme back in our **themes** directory – a separate location. Instead, everything is included in our **ibbr\_inv** directory or **templates** directory!

*Permission*

**Challenge**:

*Problem*: Limit the accessibility to the pages so that only Facilities and IT personnel can access the site.

*Solution*: Rewinding back to our original **hook\_menu** function, all we had to do was add an **access callback**, and **access arguments** to the array. Then we created a **hook\_permission** function with the name of the **access callback** so that on the site, it’ll be easier to choose which roles/personnel have access to the module/site itself.

**Note**: If we decided to hardcode the roles, this would require any changes that occur in the future, to force modifications in the code as well which is more of a hassle when compared to easy manipulation of roles on the Drupal interface.