Building a page footer using components in the Primo customization package

This handout online: https://tinyurl.com/ecaug-footer
eCAUG 2023, Jeff Karlsen, Sacramento City College

Resources

- Primo Development Environment
  Highly recommended. Allows you to test changes on your local computer, using your production Primo view. (Might not allow login to My Library Card, depending on your setup)
- Ex Libris tutorial on writing components in custom.js
  Helpful! Also included in every customization package
- AngularJS Material version 1.1.2.2
  Tools for template layout(grid, alignment), pre-set directives, icons
- AngularJS 1.8.2 API reference
  Reference for built-in directives, services
- Learn and Understand AngularJS (Udemy)
  Good for concepts, but not specific to Primo implementation
- Visual Studio Code
  Widely-used free code editor with syntax checking and formatting
- Sample customization package with bare-bones footer as produced in today's session
  For reference - will get taken down at some point
- Los Rios customization packages
  In case you are curious about what we have done

Prepare the custom.js file

If you have not yet added anything to custom.js, you need to add the execution symbol. At the end of the file, replace

```
})
```

With

```
})();
```

Understand the -after directives

Ex Libris has provided special AngularJS directives for libraries to use. When you look in the page layout, you'll find that these end in -after. For instance, *prm-explore-footer-after*. Directives look for matching components, and we can write these components in the custom.js file. The component will be named after the directive in "camelcase". So you'll write one that is called prmExploreFooterAfter.

The -after directives usually inherit some data, such as information about records on the page, Alma services etc., from their parent directives, which you can use in the template. But prm-explore-footer-directive does not inherit any useful data.

Build your component

Components in Primo normally have three properties: *bindings*, *template*, and *controller*. Depending on what you're doing, you might have some of these and not others. For our footer, since we are not working with data carried by our parent directive, *bindings* is not needed. *Template* is where the html goes. If you want to use line breaks, you need to enclose your template with back-ticks (`) instead of quotation marks. But you can also store the template in a separate file in the customization package, which may prove easier to maintain. Replace the word *template* with *templateUrl*, and include the path to the HTML file, which you can put in the /html directory of the customization package. This path will looks something like this in templateUrl, where 01CACCL_LRCCD-scc is the institution code/view code and footer.html is the name of the file we placed in the html directory: /discovery/custom/01CACCL_LRCCD-scc/html/footer.html

Enhance your template with AngularJS Material

The AngularJS Material framework is included in Primo and allows you to easily access a grid system with rows and columns. In most cases you include custom attributes that AngularJS Material converts to classes when the page is rendered in the browser. Consider the following:

```
<div flex="" layout="row" layout-xs="column">
    <div flex="" layout="column">
        <!-- column 1 content -->
    </div>
    <div flex="" layout="column">
        <!-- column 2 content -->
    </div>
    <div flex="" layout="column">
        <!-- column 3 content -->
    </div>
</div>
```

This means that on larger screens you will see a three-column layout, but on a small screen (*layout-xs*), the columns will be stacked. You can remove a column or add another if you like, add another row for a sub-footer, etc. Explore the AngularJS Material website to find other handy HTML attributes that help with formatting your templates. You can also use custom1.css

to add styles, but first see how far you can get with AngularJS Material, since it is designed to be responsive to changes in screen size.

Controllers – add dynamic elements

Controllers are where you add JavaScript to your component. Based on a number of factors, AngularJS updates the "digest" in its various directives, such as when the page changes. For instance, when the user executes a search, prm-explore-footer-after will check its component, and that component's controller will become active, so that the template might change based on the current conditions, if we've coded it to do so.

In the controller, we always create a variable to represent the controller itself, via the JavaScript keyword *this*. There is a convention of naming this variable *vm*. The controller is an object, so we can extend vm with properties that might be any data type, such as a string or a function expression. The template has access to vm; in the template, we replace *vm* with *$ctrl*, which AngularJS recognizes as referring to the controller.

We want to avoid having the footer show when there's almost no content on the page, as happens for instance when the user has just executed a search or clicked a facet and Primo is waiting for results. So we create a function in the controller that returns *true* if it detects sufficient vertical height in the page content. The parent element of the template uses the *ng-show* directive as an attribute and indicates the function as its condition. So, the footer will by default be loaded but hidden. When AngularJS updates the directive's digest, it will run the function in the controller. If the function that defines ng-show returns *true* at that moment, then the footer will be displayed.

Note: this function queries the DOM (using document.getElementsByTagName). Normally in AngularJS we avoid this sort of thing because it can degrade performance, but it seems to be ok here and there was no other clear way to accomplish the goal.

In addition to ng-show, there are lots of other interesting built-in directives, for instance:

- ng-if: only create the template in the DOM if conditions return true (then destroy if not true)
- ng-class: an element's class is defined by variables or expressions in the controller
- ng-href: href value is defined by variables or expressions in the controller
- ng-click: when an element is clicked, execute some action indicated in the controller.

The AngularJS site also shows many other available options, including built-in services that may be injected into your controller. For instance, $timeout for delaying actions, $cookies for setting or reading cookies, $http for ajax, etc. These will be helpful to people who already know some JavaScript.