ADDING CUSTOM JAVASCRIPT

NEW AND IMPROVED BEHAVIOR

- One of the core aspects to user experience is how the controls interact with the user.
- These can be changed through:
  - CSS
  - JavaScript
- Once our CSS has been dealt with, we can customize JavaScript easily through the theme.

GLOBAL JAVASCRIPT

- Since the theme controls the overall HTML of every page and the basic look and feel, it also defines the essential JavaScript libraries.
- If there are functions you need defined on each page or functions you need available globally, this is a great place to put them.
- You may also be building on different JavaScript libraries from the ones Liferay contains.
- Themes are a good place to include additional third-party JavaScript dependencies you may need for your applications and custom UI.
- S.P.A.C.E. is built on Liferay's default libraries, including:
  - AlloyUI
  - Metal.js
  - Lexicon CSS
- You can easily experiment and use your own libraries here.

EXERCISE: ADDING JAVASCRIPT

- Our theme needs to include a sign-in modal and some JavaScript for our top search.
- Let's add some JavaScript.

1. **Go to** the *exercises/front-end-developer-exercises/03-theme-development/01-generating-a-theme/exercise-src* folder.
2. **Copy** the files from the `js` folder.
3. **Paste** the files into our new Space Program Theme `src/js`.

✓ Now we can modify our `main.js` file.

- If the *Working Files* view in Brackets is getting cluttered, feel free to close the `.ftl` and `.scss` files.

EXERCISE: MODIFYING THE MAIN.JS

- Let's add our sign-in modal using AUI.

1. **Drop** the `main.js` file from your theme's `src/js` into the Brackets editor.
2. **Open** the *js* section under *snippets*.
3. **Click** on the `01-main-js` snippet.
4. **Copy** the contents of the snippet.
5. **Paste** the snippet contents over the `// Insert snippet 01-main-js here` comment in the `main.js` file.
6. **Save** the file.

ALLOYUI, YUI, AND JQUERY

- If you're familiar with YUI or jQuery, `main.js` looks pretty familiar.
- AlloyUI is avaliable through `AUI()`, much like `YUI()` or `$()`.
- Just like a jQuery page might have:

```
$(document).ready(function () {
  // Do stuff...
});
```

- AlloyUI (much like its parent YUI) uses:

```
AUI().ready(function (A) {
  // Do stuff...
});
```

- You can see syntax comparison at the AlloyUI Rosetta Stone:
  - *http://alloyui.com/rosetta-stone/*
- While AlloyUI and jQuery are available if you want to use them, you can also include any other JavaScript framework in your theme.

METAL.JS AND ECMASCRIPT 2015

- Along with AlloyUI, the S.P.A.C.E. theme is also built with another of Liferay's default libraries: Metal.js.
- As mentioned in a previous module, Metal.js is a JavaScript library that uses the ECMAScript 2015 language specification.
- By taking advantage of ECMAScript 2015 features like classes, modules and arrow functions, we can build modern, flexible UI components.

LIFERAY THEME ES2015 HOOK

- Our `main.js` requires the `top_search.es.js` file.

```
require(
  'space-program-theme/js/top_search.es',
  function(TopSearch) {
    new TopSearch.default();
  }
);
```

- The `top_search.es.js` uses Metal.js and ECMAScript 2015 syntax and will need to be transpiled to ensure that different browsers can read our theme.
  - Earlier, we discussed ECMAScript 2015 and transpilation - compiling source to be output in a different language from the input language.
- Liferay provides an ECMAScript 2015 hook that has exactly what we need.

EXERCISE: LIFERAY THEME ES2015 HOOK

- The `liferay-theme-es2015-hook` allows for ECMAScript 2015 transpilation as some browsers haven't fully implemented all ECMAScript 2015 features yet.

1. **Open** the Terminal or your command line window.
2. **Go to** the root folder for your theme.
3. **Run** the *npm i –save liferay-theme-es2015-hook* command.

- Once the hook is installed, it will run with every `gulp build` and `gulp deploy`.
- After building, components will be transpiled and packaged as AMD modules.

EXERCISE: METAL.JS DEPENDENCIES

- As our `top_search.es.js` component uses the Metal.js framework, we'll also need to download the Metal.js dependencies to build properly.

1. **Open** the Terminal or your command line window.
2. **Go to** the root folder for your theme.
3. **Run** *npm i –save metal metal-dom metal-state*.

EXERCISE: UPDATING THE PACKAGE JSON

- Finally, we need to add the `hookModules` property to our *package.json* file to make sure our theme reads the hook during the build process.

1. **Open** the *package.json* file using Brackets.
2. **Type** `"hookModules": ["liferay-theme-es2015-hook"],` in the *liferayTheme* section.
   - The *liferayTheme* property can be found on line 8.
3. **Save** the *package.json* file.

```
    "liferayTheme": {
        "baseTheme": "styled",
        "hookModules": ["liferay-theme-es2015-hook"],
        "screenshot": "",
        "rubySass": false,
        "templateLanguage": "ftl",
        "version": "7.0"
    },
```

✓ Note: The search addition may take a few minutes to display after deploying the changes.

IMPORTING IN METAL.JS

- Let's take a look at some of the Metal.js code that features ECMAScript 2015 syntax.
- In the `top_search.es.js` file, we are creating the `TopSearch` class.
  - This class will create a basic component that enhances the default behavior of the search application form.
- We can take advantage of modules in ECMAScript 2015 that give us the ability to create, load, and manage dependencies via the new `import` and `export` keywords.

```
import async from 'metal/src/async/async';
import core from 'metal/src/core';
import dom from 'metal-dom/src/dom';
import State from 'metal-state/src/State';
```

CLASSES IN METAL.JS

- Using ECMAScript 2015 features, we can also create classes with constructors and inheritance.

```
class TopSearch extends State {

  constructor() {
    super();

    this.search_ = dom.toElement('#search');
    this.searchIcon_ = dom.toElement('#banner .btn-search');
    this.searchInput_ = dom.toElement('#banner .search-input');

    ....

    }
...

}
```

WRITING FUNCTIONS WITH METAL.JS

- Arrow functions make creation of anonymous functions easier.
- With `let`, we can create variables with scope limited to the block in which they are declared.

```
onSearchInputBlur_(event) {
  async.nextTick(
    () => {
      let stateActiveElementBlur = document.activeElement !== this.searchIco
      && document.activeElement !== this.searchInput_;

      if (stateActiveElementBlur && (!this.searchInput_.value ||
      this.searchInput_.value === '')) {
        this.visible = false;
      }
    }
  );
}
```

- With Metal.js, we can use `let` and other modern features to create powerful UI components.

OWNING YOUR JAVASCRIPT

- Themes are really useful ways to set the style and behavior across all of Liferay:
  - HTML structure
  - CSS styles
  - JavaScript libraries and UI components
- Once you've customized the JavaScript and added your own JavaScript components and libraries, you're ready to package up the theme.

Notes: