CONFIGURING THE THEME

FINISHING THE THEME

- Each of our basic areas for branding are defined:
  - HTML structure and basic navigation
  - Fonts, colors, and other styles in CSS
  - Behavior and UI libraries through JavaScript
- To finish our theme, we'll need to set some basic metadata that describes our theme module.
- Remember, we'll be deploying this branding as a module file in Liferay DXP.
- There are some advanced modifications we may also want to make.
- Some settings are easy, and can be done right now.
- Other changes we'll explore in more depth.

EXERCISE: ADDING LAYOUTS

- As we'll see later, you can create custom layout templates.
- Layout templates can be bundled into our theme just like our fonts.
- Let's start by including the layout template files, and then finish our theme's configuration.

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

- We've included the `layouttpl` files and `images`.

FINISHING UP WITH THE THEME CONFIGURATION

- The last thing we need to do is provide special settings for our theme as well as some package properties.
- These configuration settings are found in the `WEB-INF` folder of the theme.
- The `liferay-look-and-feel.xml` file includes the configuration for things like custom layout templates, theme settings, and application decorators.
- The `liferay-plugin-package.properties` will contain the information and properties for our theme.

EXERCISE: ADDING OUR CONFIGURATION FILES

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 `WEB-INF`.
3. **Paste** the files into our new Space Program Theme `src/WEB-INF`.
    - Replace both files.

✓ Now we can modify our configuration files!

ADDING APPLICATION DECORATORS

- In previous versions of Liferay, administrators could display or hide the application borders through the Show Borders option of the look and feel configuration menu.
- In Liferay DXP, this option has been replaced with Application Decorators, a more powerful mechanism that customizes the style of the application wrapper.

OUT OF THE BOX APPLICATION DECORATORS

- There are three out of the box application decorators that are added to your themes `liferay-look-and-feel.xml`:
  - **Barebone**: when this decorator is applied, neither the wrapping box nor the custom application title is shown. This option is recommended when you only want to display the bare application content.
  - **Borderless**: when this decorator is applied, the application is no longer wrapped in a white box, but the application custom title is displayed at the top.
  - **Decorate**: this is the default Application Decorator when using the Classic theme. When this decorator is applied, the application is wrapped in a white box with a border and the application custom title is displayed at the top.

EXERCISE: APPLICATION DECORATOR CONFIGURATION

- Let's start by adding our application decorator configuration.

1. **Drop** the `liferay-look-and-feel.xml` file from your theme's `src/WEB-INF` into the Brackets editor.
2. **Open** the *WEB-INF* section under *snippets*.
3. **Click** on the `01-portlet-decorators` snippet.
4. **Copy** the contents of the snippet.
5. **Paste** the snippet contents over the `<!-- Insert snippet 01-portlet-decorators here -->` comment in the `liferay-look-and-feel.xml` file.
6. **Save** the file.

ADDING CUSTOM DECORATORS

- All the decorators added to the `liferay-look-and-feel.xml` require that a css class is set using the following line:

```
<portlet-decorator-css-class>[Class Name]</portlet-decorator-css-class>
```

- You can then add styling to these classes in the `_portlet_decorator.scss`.

```
.portlet-trending .portlet-content {
  background-color: $brand-bg-color-2;
  font-size: 12px;
  margin-top: 40px;
  padding: 50px 20px 20px;
...
```

- We added the styles for our Application Decorators earlier.

SETTING THE DEFAULT APPLICATION DECORATOR

- A default application decorator also needs to be set in the `liferay-look-and-feel.xml`.
- In order to do this, we can simply use the line we see here:

```
<default-portlet-decorator>true</default-portlet-decorator>
```

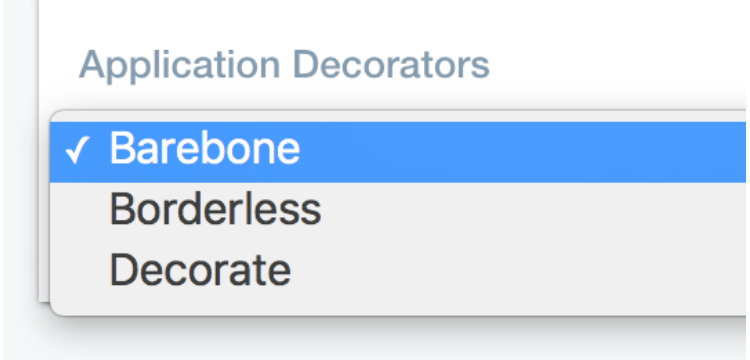
- Once this is set, we'll be able to style the standard application decorator while being able to choose others when the need arises.

ADDING APPLICATION DECORATOR CSS

- The styling in the `_portlet_decorator.scss` is imported into the `_custom.scss`.

```
@import "portlet_decorator";
```

- Once your theme is deployed you can select your new application decorator.
- The new Trending option, for example, will be selectable from the *Application Decorators* drop-down list in the *Look-and-Feel Configuration* section of our applications.



THEME SETTINGS

- We can also add *Theme Settings* into our theme.
- *Theme Settings* are used to set either hard-coded or configurable values into the theme.
- These setting keys can be accessed inside FreeMarker theme files and, once deployed, are editable through the Control Panel.
- You can provide special settings in your theme that allow administrators to make stylistic changes from the platform.
- These settings provide stylistic flexibility within a theme.

EXERCISE: ADDING THEME SETTINGS

1. **Click** on the `02-theme-settings` snippet.
2. **Copy** the contents of the snippet.
3. **Paste** the snippet contents over the `<!-- Insert snippet 02-theme-settings here -->` comment in the `liferay-look-and-feel.xml` file.
4. **Save** the file.

THEME SETTING PROPERTIES

- Each Theme Setting can have the following properties:
  - **configurable**: a value of true or false determines whether this field should be displayed or hidden from the Control Panel.
  - **key**: a name used to retrieve user value in the theme (no spaces)
  - **type**: defines the type of form field to show. Possible values are *text*, *textarea*, *select*, or *checkbox*
  - **options**: a comma separated list of options the user can choose for the *select* type.
  - **value**: default setting value

DEFAULT AVAILABLE SETTINGS

- There are some default Theme Settings that can used in your theme without further configuration:
  - **Bullet Style:** Adds Bullet Point options for different applications
  - **Show Header Search:** Displays Header Search if set to true
  - **Show Maximize Minimize Application Links:** Gives a link to navigate between maximized and minimized view of an application
  - **Show Site Name Default:** Lets you control the Site name display
  - **Show Site Name Supported:** Lets you control the Site name support

```
<setting configurable="true" key="bullet-style" options="dots,arrows"
type="select" value="dots" />
<setting configurable="true" key="show-header-search" type="checkbox" value="true"
<setting configurable="true" key="show-maximize-minimize-application-links"
type="checkbox" value="false" />
<setting configurable="false" key="show-site-name-default" value="true" />
<setting configurable="false" key="show-site-name-supported" value="true" />
```

USING CUSTOM THEME SETTINGS

- Custom Theme Settings that are added to the `liferay-look-and-feel.xml` can also be turned into variables.
- These variables are added in the `init_custom.ftl` file like so:

```
<#assign new_variable = getterUtil.getBoolean(themeDisplay.getThemeSetting
("new-theme-setting")) />
```

- With the variable created, developers can add logic to their Theme Freemarker templates.
- Our theme has a new variable for the `show-header-search` Theme Setting in order to control how the search is displayed with this setting turned on inside the `navigation.ftl`.

CUSTOM SHOW HEADER SEARCH EXAMPLE

- The `show_header_search` variable was created in the `init_custom.ftl` using:

```
<#assign show_header_search = getterUtil.getBoolean(themeDisplay.
getThemeSetting("show-header-search")) />
```

- In the `navigation.ftl`, this custom variable is used to control what displays:

```
<#if show_header_search>
  <div class="navbar-form navbar-right" role="search">
    <div id="search">
      ...
    </div>
    <button aria-controls="navigation" class="btn-link btn-search hidden-xs"
    type="button">
      ...
    </button>
  </div>
</#if>
```

EXERCISE: LAYOUT TEMPLATE CONFIGURATION

- Earlier, we added the custom layout template source files into our `src/layouttpl` folder.
- We need to actually configure the `liferay-look-and-feel.xml` to include these layouts.

1. **Click** on the `03-layout-templates` snippet.
2. **Copy** the contents of the snippet.
3. **Paste** the snippet contents over the `<!-- Insert snippet 03-layout-templates -->` comment in the `liferay-look-and-feel.xml` file.
4. **Save** the file.

THE THEME PLUGIN PACKAGE PROPERTIES

- The `liferay-plugin-package.properties` file contains the default information for our theme.
- We'll revisit this file after we look at some other features of Liferay themes.

```
name=Space Program
module-group-id=liferay
module-incremental-version=1
module-version=1.0.0
tags=
short-description=
long-description=
change-log=
page-url=http://www.liferay.com
Provide-Capability=osgi.webresource;osgi.webresource=space-program-theme
author=Liferay, Inc.
licenses=LGPL
liferay-versions=7.0.0+

resources-importer-developer-mode-enabled=true
```

ADDING COLOR SCHEMES

- Developers can also include *Color Schemes* into the `liferay-look-and-feel.xml`.
- *Color Schemes* are variations of CSS and Images with a theme.
- Using this option, developers can effectively provide themes within themes for their administrators to configure based on the business needs.
- Let's look at how we would add these using the example of fire and ice Color Schemes.

COLOR SCHEMES IN THE XML

- Color Schemes can be added to the `liferay-look-and-feel.xml` by using the following:

```
<color-scheme id="01" name="Default">
  <default-cs>true</default-cs>
  <css-class>default</css-class>
  <color-scheme-images-path>${images-path}/color_schemes/${css-class}
  </color-scheme-images-path>
</color-scheme>
<color-scheme id="02" name="Fire">
  <css-class>fire</css-class>
</color-scheme>
```

- We need to set the default as well as any new Color Scheme we're adding.
- The `default-cs` makes sure to auto-select this color scheme.
- Each Color Scheme needs to include a `css-class` while the images are set using different variables in the default.

ADDING COLOR SCHEME CSS

- Once the XML is set, You can add the CSS changes using the following steps:

1. Create a new directory in `src/css` called `color_schemes`.
2. Add a new scss file for each Color Scheme in this folder.
    - Example: *_fire.scss* or *_ice.scss*
3. Add styling to these files using the class selector identified in the XML.

```
body.fire {
  color: #F00 !important;
}
```

4. Last, import the new scss files into the `_custom.scss`:

```
@import "color_schemes/fire";
@import "color_schemes/ice";
```

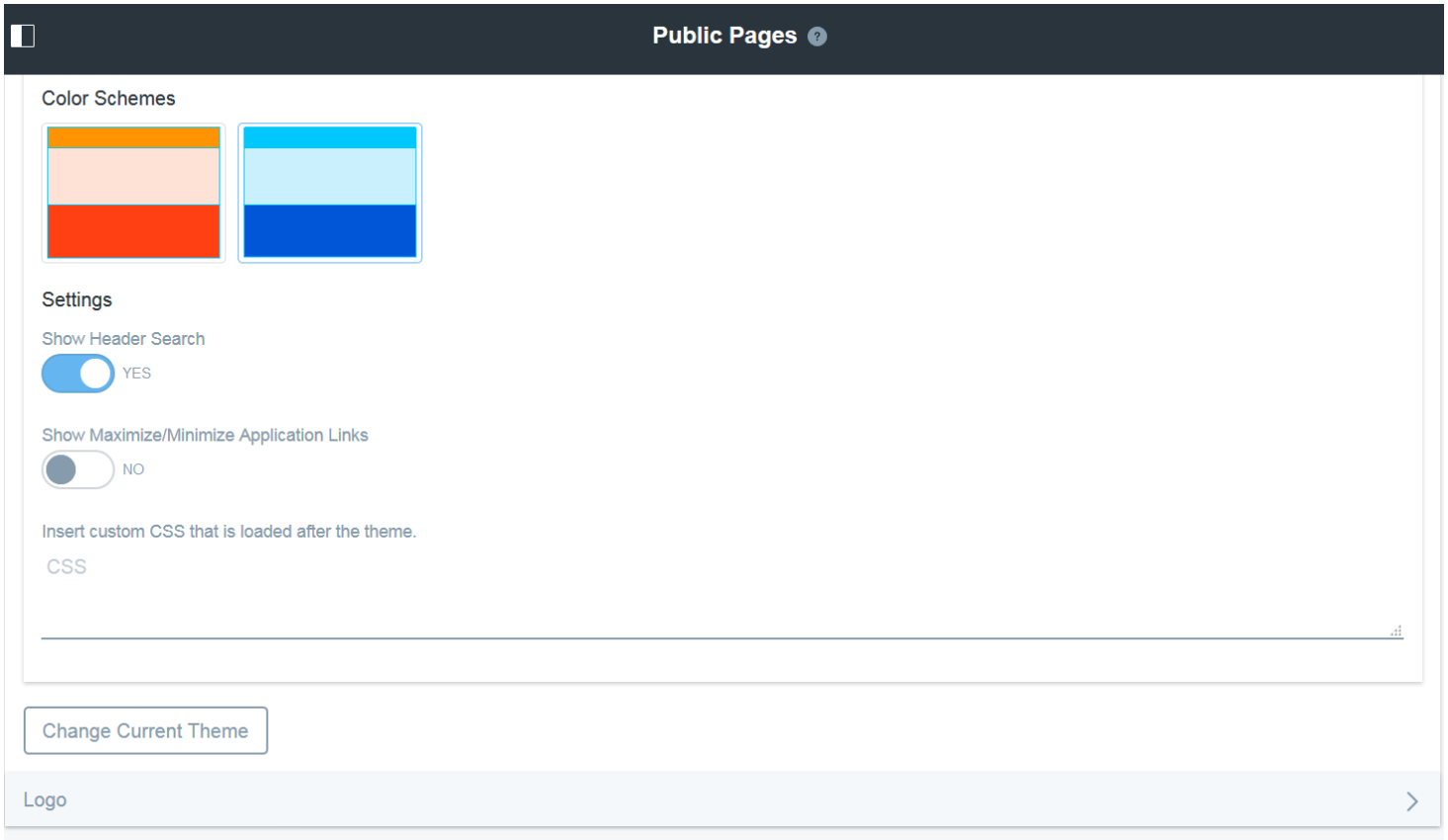
ADDING COLOR SCHEME IMAGES

- Next, images can be added using the following:

1. Create a new directory in `src/images` called `color_schemes`.
2. Add a new folders for each Color Scheme such as *color_schemes/fire* or *color_schemes/ice*.
3. Place any images or thumbnails for the color schemes in their appropriate folder.
    - The thumbnail that will display in the page configuration menu needs to go here.

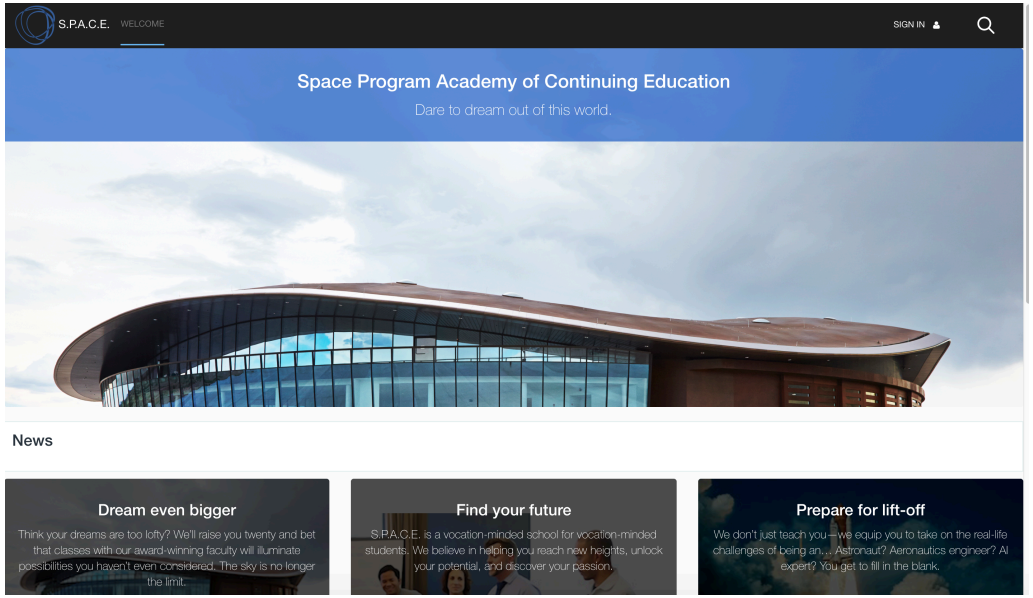
CONFIGURING PAGES

- Once Color Schemes and Theme Settings are added to the Theme, they can be accessed on each Site where the Theme is in use.



WHAT'S NEXT?

- This gives us a starting point for a working theme.
- Next, we'll look at some of the additional pieces we mentioned during our theme development.
- Items like Layout Templates deserve a closer look.
- We'll also take a look at additional tools and development options available.



Notes: