Get Started with Themes

This tutorial guides you through all the major facets of theme development, including editing project assets on your local machine, uploading themes to Dev Center, and applying themes to sites. At the end of the tutorial, you will be prepared to create and customize new themes. Along the way, you may develop questions about particular concepts that this tutorial does not answer in detail. But no need to worry: after working through the tutorial you can find much more in-depth information in the other theme topics, each of which take a focused look at a particular theme concept.

> If you experience issues with KCCP at any point after implementing a custom theme, such as behavioral bugs or interface errors, you should troubleshoot your implementation before contacting to determine whether the issue is caused by your theme or the underlying platform.

Before You Begin

Install the following software on your local machine.

- Command Line Interface: Provides a console or terminal environment in which to execute commands. If you are developing on a Windows machine, Kibo recommends you use the Windows command prompt. However, due to known Windows compatibility issues in Node.js, you may encounter issues when using the API Extension tools (for example, the tools may not register your keystrokes). If you encounter such issues, Kibo recommends you use a third-party console emulator, such as cmder, which plays nicer with Node.js.
- Node.js: Provides a platform for creating scalable network applications. Includes the npm package manager, which you use in this tutorial to install additional tools.
- Grunt.js: Provides a task manager that automates repetitive tasks. In this tutorial, you use Grunt to build your project files and upload them to Dev Center.
- Git: Provides a version control system for managing your project files. The tools you use to create a theme in this tutorial automatically configure a Git repository for your project. Kibo frequently releases new themes and updates to the Core theme through Git, making it an essential part of your theme upgrade path.

In addition to the listed software, you need access to aDev Center Account that contains a provisioned sandbox.

Don't have a Dev Center Account? **Request a Demo**

1.0: Create a Theme

Dev Center is where your themes live. To upload theme files you've created or modified, you must first create a theme record in Dev Center.

To create a theme record:

1. In the Dev Center Console, click **Develop** > **Themes**.

2. Click **Create Theme**.
3. In the **Create Theme** dialog box, specify the theme Name and ID as follows:
    1. **Name**: MyFirstTheme
    2. **Theme ID**: MyTheme
4. Click **Save**. You should now see your theme in the Themes grid.
5. Double-click your new theme to edit it.
6. Note the **Application Key**. You will need this value to configure the Theme Generator and Dev Center Sync tools later in this tutorial.

1.1: Prepare Your Development Environment

Themes are designed to leverage inheritance, so all of your theme development should extend the latest Core theme. We recommend using the Theme Generator to manage your theme assets. The Theme Generator:

- Creates new themes that inherit from the latest Core theme or clones an existing theme from a Git repository to a local directory. You can also use the generator to upgrade existing themes. Refer to Theme Asset Management for more information.
- Configures your local theme directory as a Git repository.
- Connects the Core theme Git repository (or any other theme Git repository) to your theme as a remote so you can merge upstream changes.

The Theme Generator

This tool is a Yeoman plugin that generates the scaffolding (e.g., directory structure, reference files, and build files) necessary to package a theme and upload it to Dev Center. It's designed to augment, not overwrite, existing themes. If you have a theme that extends the Core theme, you can safely run this tool in that directory without overwriting your existing files. Whenever Kibo releases a Core theme upgrade, you can use this tool to merge changes from the Core theme (or any other theme Git repository) with your theme.

For this tutorial, create a brand new theme that inherits from the latest Core theme:

1. Open a Terminal (OS X) or a Command Prompt (Windows).
2. Install the Yeoman command-line tool globally:
   ```
   npm install -g yo
   ```
3. Install the Grunt command-line tool globally:
   ```
   npm install -g grunt-cli
   ```
4. Install the Theme Generator globally:
   ```
   npm install -g generator-mozu-theme
   ```
5. Create a new folder for your theme on your local machine and navigate to it:
   ```
   mkdir your_theme && cd your_theme
   ```

6. Run the Yeoman generator inside your theme directory:
   `/your_theme/$ yo mozu-theme`
7. If you have an old version of the tool installed, you'll be prompted to update it. Press <Ctrl+C> to exit the application and enter the following command: `npm install -g generator-mozu-theme`
8. Select **Brand new theme**.
9. Enter a public name for your theme.
10. (Optional) Enter a short description of your theme.
11. Enter the initial version.
12. Select **Kibo eCommerce Core Theme** as the base theme from which your new theme will inherit.
13. Select which version of the Core theme from which your new theme will inherit.
14. Enter your theme's Dev Center Application Key.
15. Enter your Developer Account login email.
16. Enter your Developer Account password.
17. Select your developer account.

You now have a blank theme based on the latest Core theme, which you can modify, build, and upload to Dev Center. Since the Core theme Git repository is connected to this Git repository as a remote, you'll be able to merge upstream updates from the Core theme with your theme in the future.

1.2: Make a Quick Change

Now that you have your own copy of the latest Core theme, you can begin making modifications.

1. Open theme.json in your project's root directory for editing. This file contains the majority of your theme settings, structured in JSON format.
2. In the about object, change the value associated with the name property from whatever you chose as a name to Quickstart.
3. Save and close theme.json.

Your local copy of your theme now contains changes that you can upload to Dev Center.

1.3: Upload Your Theme Files

You need to build and upload your theme files to Dev Center in order to apply your theme to a site. Kibo provides build tools that take care of this process for you. Complete the following steps to take advantage of these build tools:

1. Open a command prompt in your project's root directory.
2. Run `grunt` to build your project assets and upload them to Dev Center.

What does `grunt` do and what are the common options you can use with it?

`grunt:`

- Checks your JSON and JavaScript for syntax and style errors
- Compares your theme with the remote base theme and notifies you if updates are available for merging
- Compiles your theme's JavaScript according to the `./build.js` file that you either inherit or override
- Uploads changed files to Dev Center into the theme specified by the Application Key you provided when configuring the Theme Generator tool
- If you've added new files at the root level of your theme directory, you must add each file name to the mozusync.upload.src section of Gruntfile.js to upload them using the `grunt` command.

`grunt build-production:`

- Checks your JSON and JavaScript for syntax and style errors
- Compiles your theme's JavaScript according to the ./build.js file that you either inherit or override
- Compress and minify the compiled JavaScript for production
- Creates a .zip containing your theme files suitable for sharing or manually uploading within Dev Center (The ZIP file you upload contains only the contents of the theme folder that have changed and not the theme folder itself.)

`grunt mozusync:wipe && grunt:`

- Cleans up the theme in Dev Center by deleting all files and then re-uploading your theme files

`grunt watch:`

- Listens for any changes to your theme files
- If you save a change to a theme file, `grunt` automatically builds and uploads your theme to Dev Center.

Manually Uploading Your Theme Files to Dev Center

This section is FYI-only. As an alternative to the build tools, you can manually upload a built and zipped package of your theme using the following steps. However, Kibo recommends using the build tools in most cases.

1. In the Dev Center Console, click **Develop** > **Themes**.
2. In the Themes grid, double-click **MyFirstTheme**.
3. Click **More** > **Upload**.
4. Drag your zipped theme file into the **Upload files** dialog box.
5. When the upload is complete, click **Done**.
6. Click **Packages**. You should see the contents of your theme ZIP file.

1.4: Install Your Theme in a Sandbox

After your theme builds, install your theme to a sandbox so you can later apply the theme to a site. You only need to do this once. After installation, your theme remains installed on the sandbox until you remove it.

1. In the theme toolbar at the top right, click**Install**.
2. In the Select a Tenant dialog box, select the sandbox you previously created.
3. Click **OK**.

Now your theme is installed in your sandbox. The next step is to apply it to a site within your sandbox.

1.5: Apply Your Theme

The Content Editor is a module in Admin that merchants use to interact with themes. You can use this to test the functionality and appearance of your theme. To apply your theme to a site in your sandbox:

1. In the Dev Center Console, click **Sandboxes**.
2. In the Sandboxes grid, right-click the sandbox you installed your theme to and select**View**.
3. Log in to Admin.
4. Click **Main** > **Content** > **Themes**.
5. Click the dots on the `Quickstart` theme and click **Apply**.

The rest of this tutorial guides you through more comprehensive theme changes. You do not need to reinstall or reapply your theme from this point forward. Any updates you upload to the theme record in Dev Center using `grunt` are immediately available on sites that theme is installed to. In fact, if you run the `grunt watch` command in your local theme directory, any changes you make locally will be continuously synced with Dev Center. Just refresh your web browser to see the changes on your storefront.

2.0 Add a Banner Image

In this section, you take advantage of a default theme setting to add a banner image to your website.

1. If you haven't already done so, run `grunt watch` in your project's root directory. From here on out, all local changes to your theme files will be automatically pushed to your site.
2. Open theme.json in your root directory.
3. In the settings object, change the value of bannerImageEnabled to true.
4. Note that the value of bannerImage is set to ../resources/images/banner.jpg. Navigate to the resources/images directory in your theme files and add an image of your choice, named banner.jpg, to serve as your banner image.
5. Refresh your browser or use the Content Editor to view your staged site again. Your banner image appears as the header background at the top of every page on your site.

How exactly does this theme setting work?

1. Open page.hypr in the templates directory. This is the base Hypr template that all other pages on your site

extend.

2. Locate the following lines of code in templates/page.hypr:

```
{% block page-header %}
{% include "modules/page-header" %}
{% endblock page-header %}
```

   All content enclosed in `block` tags is sent to child templates that extend the current template (so these lines of code get picked up by every other page on your site that extends page.hypr). Within the block, another templates is included using the `include` tag, and its contents are rendered.

3. Open the page-header.hypr template in the templates/modules directory.
4. Note how this template sets the class of the header on a conditional basis with the help of the `if` tag and the themeSettings global variable, which accesses the settings you configure in theme.json at templates/modules/page-header.hypr:

```
&lt;header class="mz-pageheader{% if themeSettings.bannerImageEnabled %} mz-pagehea
der-hasbanner {% endif %}"&gt;
```

5. Open page-header.less in the stylesheets/modules directory. This stylesheet contains the style rules for the relevant header class.

```
.mz-pageheader {
   position: relative;
   height: 154px;
   background: #eee;
   &.mz-pageheader-hasbanner {
      background: #eee url('{{themeSettings.bannerImage}}') {{ themeSettings.bannerIma
geRepeat }};
   }
   ...
}
```

   You can see how the path to the banner image you added is extracted from the themeSettings variable.

There are many other useful settings in theme.json that you can use for theme development. You can also add your own settings to this file as well, and access those custom settings throughout your theme templates using the themeSettings variable if the header has the appropriate class applied.

> To view every property within a Hyprglobal variable, such as the model or themeSettings objects, use the dump tag within a Hypr template.

2.1 Modify the Theme Settings

If you go to **Main** > **Content** > **Themes**, you can click the dots by any theme and click**Settings** to view available settings that can be configured by Admin users.

In this section of the tutorial, you modify the settings that display on this page.

1. Open theme-ui.json in your project's root directory. This file controls the theme settings.
2. Add the following code to the items array, after the closing curly brace and comma for the item that houses the "General" settings:

   ```
   {
      "xtype": "panel",
      "title": "Quickstart",
      "collapsed": false,
      "items": [
         {
            "xtype": "mz-input-text",
            "name": "randomText",
            "fieldLabel": "Type randomly here"
         },
         {
            "xtype": "mz-input-checkbox",
            "name": "enableOverdrive",
            "fieldLabel": "Enable overdrive"
         },
         {
            "xtype": "mz-input-selectmulti",
            "name": "selectInterests",
            "fieldLabel": "Which theme components are you interested in?",
            "allowBlank": false,
            "forceSelection": true,
            "store": [
               "Theme Settings",
               "Hypr Templates",
               "Widgets",
               "Stylesheets",
               "JavaScript"
            ]
         }
      ]
   },
   ```

   This code adds a panel to your theme settings that includes a text area, checkbox, and a multi-select drop-down that requires at least one value to be selected.

You can access the values of these inputs using the themeSettings variable and the name of the input in question (for example, themeSettings.enableOverdrive). However, you also have to add the setting to theme.json to make it accessible through the global variable.

1. Open theme.json.
2. In the settings object, add the settings you configured in theme-ui.json, matching the name property and providing default values for when users do not provide their own:

   ```
   "settings": {
      "randomText": "default",
      "enableOverdrive": false,
      "selectInterests": [],
      ...
   ```

If the name of a setting in theme-ui.json is the same as the name of a setting in theme.json, and a user has provided a value, then the theme-ui.json setting takes precedence.

After saving your changes, an additional "Quickstart" panel will be added to the Content Editor.

3.0 Change How Category Pages Display Products

If you open category.hypr in the `templates/pages` directory, you see that it includes category-interior.hypr. This template uses the `include_products` tag to load a product model from the faceted-products.hypr template in the `templates/modules/product` directory. This templates renders a list of products for the category using the product-list-tiled.hypr.live template, which extends the product-list.hypr.live template. In this section of the tutorial, you edit product-list.hypr.live to add a dropzone for a custom widget that you create later in the tutorial and then edit product-listing.hypr.live to make small changes to how products display on a category page.

1. Open product-list.hypr.live in the templates/modules/product directory.
2. Find the unordered list and for the first list item add a `{% dropzone "first-product-tile" scope="page" %}` tag to enable users to add a widget where the first product listing would otherwise display on a category page. After the dropzone addition, the unordered list should match the following code block:

```
<ul class="mz-productlist-list {% block list-classes %}{% endblock list-classes %}">
   {# The line below is the new addition. Dropzone tags take a name of your choice and a wi
dget display scope as arguments. #}
   <li class="mz-productlist-item">{% dropzone "first-product-tile" scope="page" %}</li>

   {% for prod in model.items %}
      <li class="mz-productlist-item" data-mz-product="{{ prod.productCode }}">{% include
 "modules/product/product-listing" with model=prod %}</li>
   {% endfor %}
</ul>
```

3. Note that this template includes product-listing.hypr.live (with the local variable model set to the current product model in the loop) to render product details for each product tile in the category list.
4. Open product-listing.hypr.live in the templates/modules/product directory.
5. Relocate the line of code that renders product names so that the name of a product displays above the product image. Use the following partial code block as a reference:

```
...
{# The following div renders the product name above the product image #}
<div class="mz-productlisting-info">
   <a class="mz-productlisting-title" href="">
   {{model.content.productName}}</a></div>

<div class="mz-productlisting-image">
   {% block product-image %}
   ...
   {% endblock product-image %}</div>

{# The code that renders the product name has been moved out of the following div #}

</code><div class="mz-productlisting-info"><code>
   {% if model.content.productShortDescription and themeSettings.listProductShortDesc %}
   ...
...
</div>
```

After you save your changes, the first listing on a category page is an empty dropzone and product names display above the product image.

3.1 Redo the Styling for Product Listings

In this part of the tutorial, you make styling changes to the unordered list that displays products on a category page.

1. Open `product-list.less` in the `stylesheets/modules/product` directory.
2. Add styling rules to create a special hover effect for product listings, as shown in the following code block:

```
.mz-productlist {
   &-tiled {
      .mz-productlist-item {
         min-height: @productTileHeight;
         width: @productTileWidth;
      }

      /* Add the following style rules to create the hover effect */
      ul li { display:block; padding-left:10px; text-decoration:none; }
      ul li:hover {  -moz-transform:rotate(-5deg); -moz-box-shadow:10px 10px 20px #D3D3D3
;
         -webkit-transform:rotate(-5deg); -webkit-box-shadow:10px 10px 20px #D3D3D3;
         transform:rotate(-5deg); box-shadow:10px 10px 20px #D3D3D3; }
   }
}
```

After you save your changes, product listings rotate whenever a shopper hovers over them.

4.0 Create a Custom Widget

Within the Content Editor, widgets allow merchants to drag-and-drop specialized functionality to predetermined areas of storefront pages. Kibo provides several built-in widgets, but you can create your own widgets as well.

In this section of the tutorial, you create a widget that displays product recommendations from employees. To get started, create the widget configuration data:

1. Open theme.json in your theme's root directory.
2. Locate the widgets array. This array contains a collection of widget configurations.
3. Add a new widget configuration item to the widgets array that contains the following code:

```
{
   "defaultConfig": {
      "field1": "Name",
      "field2": "Age",
      "employeeRecommendation": "I suggest..."
   },
   "displayName": "Employee Picks",
   "displayTemplate": "misc/employee-picks",
   "editViewFields": [
      {
         "anchor": "100%",
         "fieldLabel": "Card Background",
         "name": "cardBackground",
         "xtype": "mz-input-image"
      },
      {
         "anchor": "100%",
         "fieldLabel": "Employee Image",
         "name": "employeeImage",
         "xtype": "mz-input-image"
      },
      {
         "anchor": "100%",
         "fieldLabel": "Field 1",
         "name": "field1",
         "xtype": "mz-input-text"
      },
      {
         "anchor": "100%",
         "fieldLabel": "Field 1 Value",
         "name": "field1Value",
         "xtype": "mz-input-text"
      },
      {
         "anchor": "100%",
         "fieldLabel": "Field 2",
         "name": "field2",
         "xtype": "mz-input-text"
      },
      {
         "anchor": "100%",
         "fieldLabel": "Field 2 Value",
         "name": "field2Value",
         "xtype": "mz-input-text"
      },
      {
         "anchor": "100%",
         "fieldLabel": "Employee Recommendation",
         "name": "employeeRecommendation",
         "xtype": "mz-input-textarea"
      }

   ],
   "icon": "/resources/admin/widgets/30_employee_picks.png",
   "id": "employee-picks",
   "validPageTypes": [
      "*"
   ]
}
```

4. Add your chosen widget icon to the resources/admin/widgets directory. The preceding code uses the 30_employee_picks.png image, which you can download below, but you can use any icon of the same dimensions

   The numbering system used by the default widget icons is for organizational purposes within the local folder, but does not affect how the widget icon displays.

The widget configuration data adds an "Employee Picks" widget that you can add to any page on your site. When you drag the widget onto a dropzone, a menu displays that allows you to select an employee picture and background image for the widget. The menu also contains fields that render details about the employee alongside the employee's product recommendation.

If you are using an implementation with a Kibo site, you can test your widget configuration:

1. Log in to Admin.
2. Go to **Main** > **Content** > **Editor**.
3. Click on a Category page in the **Navigation** section of your site tree.
4. Click **Widgets** to display a list of widgets.
5. Drag the "Employee Picks" widget onto the product list dropzone (although any dropzone will work) that you created in Section 3.0 of this tutorial.

When you drag the widget onto the dropzone, you should see the configuration menu. However, the widget isn't functional until you create a template for it to render content, as described in the next section of this tutorial.

4.1 Create the Widget Template

Within the configuration data you added in the previous section, you specified a template for the widget using the code, `displayTemplate": "misc/employee-picks"`. Now it's time to create this template:

1. Create a file called employee-picks.hypr in the templates/widgets/misc directory.
2. Copy the following code into the file:

```
<div id="wrapper">
   <ul id="employee_cards">
      <li id="card" style='background:url({% make_url "image" model.config.cardBackground
 %});'>
         <h3>Employee Picks{{ model.config.field1 }}: {{ model.config.field1Value }}<br/>
         {{ model.config.field2 }}: {{ model.config.field2Value }}<br/>
         {{ model.config.employeeRecommendation }}</p>
      </li>
   </ul>
</div>
```

3. Save the file.

The template renders the images and text that users enter in the widget menu. The template leverages the `make_url` tag to create links to uploaded images and uses the model.config object to access data from the widget configuration menu.

4.2 Style the Widget

In the previous section, you created a template to render content for the widget. In this section, you create a stylesheet to style that content.

1. Create a file called employee-picks.less in the stylesheets/widgets directory.
2. Copy the following code into the file:

```
#wrapper {
    max-width: @productTileWidth;
    text-align:left;
}

ul#employee_cards {

    text-align:center;
    padding: 0;

    li {
        display:block;
        float:left;
        border:1px solid #666;
        padding:25px 10px;
        position:relative;
        -moz-border-radius: 10px;
        -webkit-border-radius: 10px;
        -moz-box-shadow: 2px 2px 10px #000;
        -webkit-box-shadow: 2px 2px 10px #000;
        -moz-transition: all 0.5s ease-in-out;
        -webkit-transition: all 0.5s ease-in-out;

        p {
            margin-top:4px;
            text-align:left;
            line-height:28px;
        }

        img {
            margin-top:7px;
            background:#eee;
            height: @productListingThumbSize;
            width: @productListingThumbSize;
            -moz-border-radius: 5px;
            -webkit-border-radius: 5px;
            -moz-box-shadow: 0px 0px 5px #aaa;
            -webkit-box-shadow: 0px 0px 5px #aaa;
        }
    }


    li:hover {
        z-index:4;
    }
}

#card {
    background-color:#69732B;
    z-index:3;
}

#card:hover {
    -moz-transform: scale(1.1);
    -webkit-transform: scale(1.1);
}
```

3. Save the file.

4. Open storefront.less in the stylesheets directory.
5. Import your new stylesheet by adding the following line of code to the //Widgets section:

   `stylesheets/` storefront.less

   ```
   @import "/stylesheets/widgets/employee-picks.less";
   ```

6. Save the file.

Your widget is complete! Return to the Content Editor to test it out. When you drag and drop the Employee Picks widget onto a dropzone, you can upload pictures of your choice to render a background for the widget and a profile pic for an employee. You can then add brief facts about the employee and describe which products the employee enjoys.

There are many other things you can do with widgets. In particular, you can use JavaScript to achieve advanced functionality in widgets, and you can use custom editors to render more advanced widget menus. To learn more about widgets, refer to the widgets topic.

5.0 Deploying the Theme

When you're finished developing and testing your theme, you should publish it as a final version to prevent any unauthorized changes. Then, you can do self-serve production updates, including pushing a theme from a sandbox to a production tenant.

1. Find the theme that you are ready to deploy in Dev Center.
2. Right click the theme and select **Edit** > **Publish** to publish the theme. This will prevent any more modifications.
3. Click **Install** > **Production Tenants**.
4. Select the production tenant to deploy the theme to.
5. In the Admin UI of that production tenant, go to **Main** > **Content** > **Themes**.
6. Find the version of the theme you just installed and expand the dropdown menu on the right hand side of its row in the themes table.
7. Click **Apply**.

For additional information about theme asset management in addition to the instructions on this page, see the Dev Center documentation.

Theme Feature: reCAPTCHA

One example of functionality that can be implemented via themes is reCAPTCHA. This feature can be added to the store login page and popover modal for additional security and to protect the server from being bogged down by repetitive failed login attempts. When enabled, login attempts are only be validated if the reCAPTCHA secret exists in theme settings.

The following theme settings allow you to enable and customize this tool:

- **Enable Recaptcha**: This is used to turn the reCAPTCHA feature on and off

- **Recaptcha Site Key**: Input your reCAPTCHA Site Key from the Google dashboard here
- **Recaptcha Secret**: Input your reCAPTCHA Secret from the Google dashboard here
- **Recaptcha Type**: Choose Invisible to use an invisible captcha that validates real users through mouse movements and other secret Google algorithms. Choose "I'm not a robot" to use a checkbox widget to validate real users. Note that even if you choose "Invisible" or "I'm not a robot," very suspicious users will still be required to solve an image captcha.
- **Recaptcha Size**: Controls the size of the checkbox widget. This setting does not apply to Invisible reCAPTCHA type
- **Recaptcha Theme**: Controls the color of the checkbox widget. This setting does not apply to Invisible reCAPTCHA type
- **Recaptcha Badge Position**: Controls the position of the Google reCAPTCHA badge that appears on protected pages.

Add these settings to your custom theme with your desired values, such as in the example shown below. Then, tag on token:[recaptcha-token] to the login submit to add reCAPTCHA to the page.

```
"enableRecaptcha": true,
"recaptchaSiteKey": "abc123",
"__recaptchaSecrete": "abc123",
"recaptchaType": "Invisible",
"recaptchaTheme": "light",
"recaptchaSize": "normal",
"recaptchaBadgePosition": "bottomright",
```

Although you can go back and change the reCAPTCHA settings at any time through your theme, they may also be displayed at **Main** > **Content** > **Themes** > select your theme > **Settings**.

Conclusion

Congratulations on completing the theme tutorial! In summary, you learned about the following theme concepts:

- How to create a theme application in Dev Center.
- How to scaffold theme assets using the Theme Generator, which creates the necessary theme files within a Git repository.
- How to use `grunt` to lint your source code and upload it to Dev Center (and `grunt watch` to do this anytime you save a change to your local theme files).
- How to install a theme to a sandbox for the first time.
- How to apply a theme to one of your sites.
- How to view your theme.
- How to add or make changes to the settings in theme.json, which can be accessed in Hypr templates, JavaScript files, and stylesheets.

- How to add or make changes to the settings in theme-ui.json, which display in the Content Editor under Theme Settings.
- How the Hypr rendering language extends and includes other templates.
- How Hypr templates use HTML, Hypr tags and filters, and global variables to render content.
- How to use LESS stylesheets to style the content that Hypr templates render.
- How to create a custom widget that enable merchants to make on-the-fly changes to storefront pages.

To learn more about themes, refer to the other theme topics in the menu. Be sure to check out thereference help for detailed information about the tags, filters, variables, and form controls available in themes.