**Project folder structure**

Following the steps to be run for getting the folder setup we will run the node server and follow the instructions. As soon as we finish running it we get a project folder with the specified name written to our disk at the specified directory.

Now this disk has a specified structure which uses handlebars and SASS and using grunt as our front end task runner we will get the desired files compiles into different folders for production as well as development.

Folders

**Data :** here the json data that has to be looped through in the hbs templates are written in different files for each module that we develop.

**Images :** Here all our images that have to be used across the site are used and the path is used appropriately in all places like hbs files or json data.

**SASS :** Here all our sass files come. The root folder has the files that have to generate as output. It can be one global css or multiple. Here we can include the component sass partials as well as any bower component partial and also fonts partial. There is a component folder which has partials inside for each component (SiteDev has to take care that we need to namespace our component in the unique namespace class of our component). All these files need to start with a \_ since it is a partial. There is a fonts folder which contains all the fonts and a sass partial inside which has the font definitions that will be used.

**Scripts :**  The way we handle scripts in here by pushing all the individual components written in components to a global namespace. Please refer a sample js where we have a global object and the init method of our module/component is added to this global namespace. Now in the HTML of the component we write a script tag that adds a ‘selector class’ and the ‘script module’ to be used. Now if this page has 5 components these 5 selectors with respective modules will be added to an object called ‘namespace.initUI’ which is looped through in the global js to call their ‘init’ methods respectively. Also we need to make sure which js from the bower components go where so grunt has to be altered whenever we add a new plugin or custom script anywhere else apart from ‘/scripts/components’ folder. This way all our js modules are present in the global namespcar to be used whenever the component is used in the page.

There is also main.js which has some global level (should not be any component specific) that can be added to this file. There is another file called the config.js which has some pre built helper methods as well as viewport checker, device cheker and others that can be used inside any new js that need to call this out(This avoids repetition of code and also avoids to many event listeners to the same event).

**Templates :**

***Layout:***  There can be multiple layouts that the site uses. For example one of them has a header, footer and body area. This can be the default template that will be used across as this is common. So in these files we create what and all needs to come in the page at higher levels and add a body tag which gets replaced by components during grunt tasks.

***Pages:***  The html pages that have to be generated into the final folder will be the hbs in the pages. Here what we do is add individual modules/partials that will be needed to go to a page which after compiling will create a page with all components. Also while specifying the components we also need to tell what json file it has to use to loop through to get the desired HTML output. Ex. {{>partial-name json-file-object}} Also not that all parent classes of the page like full width or component-placeholder classes need to be added to get desired output like in example pages of proui.

***Partials :*** All individual components go in here as hbs files. We assume that it gets a json and write a template which can be looped through. Also take care that the component level css classes as used by proui has to be followed to get the right grid view. The first div of this partial needs to have classes like component, namespace-layout-col-2 and your unique selector class.

**Component structure**

To make Sitecore’s component placement more flexible we will follow a predefined structure which has been generated so that all spacing issues like padding and margins across components is taken care of. Adding any component in any placeholder will place itself in the right way.

Grid scss tells us that a 3 column or a 4 column layout will be used so that all possible combinations of placements is taken care of. Examples of all such combinations have been shown in the index file.

So basically we have the idea that in a 3 column layout the 2 children will be 4 parts and 8 parts resp. In the 4 parts it will accommodate only 1 component hence taking 100%. And in the 8 parts we have flexibility to add 2 placeholders of 4 parts i.e ‘.layout-col-1’ each or a single component of 8 parts viz. ‘.layout-col-2’  hence completing the 12 column structure. It is usually a case where no more than 12 columns are utilised but if more components are added then it automatically falls to the next line and the bootstrap structure is maintained.

Similarly in 4 column can be used to create all other cases where the 12 columns are broken down into 4 parts of 3 columns or 2 parts of 6 columns or a single part having all the 12 columns. Also we note that we can use columns like 6 or 12 further down into halves. Also note that the these parts will take no lesser than 4 columns.

This has given a basic idea of how our entire structure will be behaving. But one more thing that we will be following is that we will be having a predefined structure for every component as shown below.

*<!-- 2 divs will be used, one as component which has layout class which says how many parts it will take and the other class is ‘component-content’ which is the child of component and will have spacing, background, etc.. Note that you can make changes to the layout using class ‘card-container’ to the parent if your components requires spacing between them-->*

*<div class="component layout-col-{{column}}">*

*<div class="component-content">*

*<!-- All your content of this component comes here -->*

*<h3 class="title">{{title}}</h3>*

*<span class="description">{{description}}</span>*

*</div>*

*</div>*

**HTML Generation (Using Assembly and Handlebar)**

HTML pages are generated using Assembly and Handlebar and could be updated automatically using “grunt-watch”. HTML pages are created using:

* Components and reusable blocks are stored as Handlebar templates in **templates/partials** folder.
* Page templates in **templates/pages** folder.
* Page layout templates stored in **templates/layouts** folder.
* Data in form of JSON is stored in **data** folder.

The paths and options for HTML generation are configured in the **Gruntfile.js**.

Example:

LAYOUT:

{{> sample-global\_html\_header }}

{{> sample-global\_header }}

   {{ body }}

{{> sample-global\_footer }}

{{> sample-global\_html\_footer }}

**PAGE**:

<div class="page-content my-theme">

<div class="contained-component-placeholder nmspc-layout-4">

<div class="container">

 <div class="row">

<div class="col-sm-3">

<div class="row">{{>sample-bookmarks sample-bookmarks}}</div>

</div>

<div class="col-sm-9"></div>

   </div>

   </div>

</div>

</div>

**COMPONENT: (sample-carousel.hbs)**

<div class="contained-component-placeholder nmspc-layout-4">

<div class="container">

<div class="row">

<div class="col-sm-3">

<div class="row">

{{>sample-bookmarks sample-bookmarks}}

</div>

</div>

<div class="col-sm-9"></div>

</div>

</div>

</div>

For more understanding just take a look at our example pages.

CSS

We have some predefined classes that can be used while development so that padding issue while making a card is solved.