FOSSEE SUMMER FELLOWSHIP 2020

Report

on

Page Designing and Templating in Drupal 8

Under the Guidance of

Professor P. Sunthar

Chemical Engineering Department

IIT Bomaby

Made By:-

Karan Singh Singare

IIEST Shibpur, Howrah

B.Tech 3rd year(Computer Science and Technology)

Acknowledgement

It brings me great pleasure for an opportunity to work and submit my fellowship report on **PAGE DESIGNING AND TEMPLATING IN DRUPAL 8.** For this I deeply indebted and sincerely thankful to our mentors **Mr. Tejas Vaidya** and **Ms. Ruchi Kumari** of **IIT Bombay** for their help, invaluable guidance and elating encouragement throughout the course of this fellowship.

I am also thankful to **Mr. P. Sunthar, Department of Chemical Engineering, IIT Bomaby** for his timely advises.

I would like to acknowledge the co-operation of various other fellowship member **Ms. Dolon Mandal, Ms. Prerna Sawhney** and **Mr. Tejas Anand Srivastava** who helped me in completing my fellowship.

Finally, I am deeply thankful to my parents and teachers who helped and inspired me in completing this fellowship.

Content

1. SASS

SASS(short for **syntactically awesome style sheets)** is a preprocessor scripting language that is interpreted or compiled into Cascading Style Sheets(CSS). Sassscript is the scripting language itself.

SASS consists of two syntaxes. The original syntax, called "the indented syntax". It uses indentation to separate code blocks and newline characters to separate rules. The newer version syntax "SCSS" (Sassy CSS), uses blocks and semicolons to separate rules within a block. The indented syntax and SCSS files are traditionally given extensions **.sass** and **.scss**, respectively.

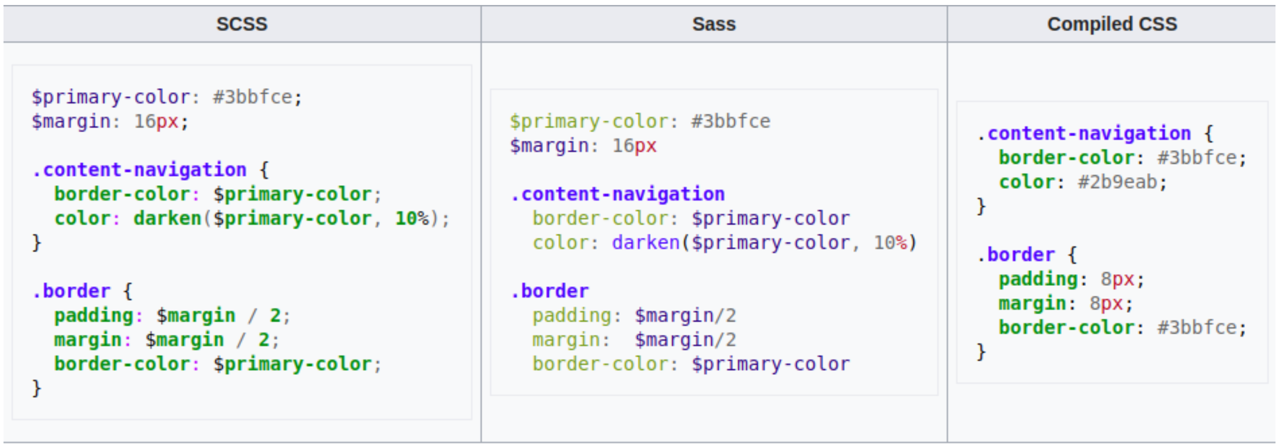
The SASS interpreter translates SassScripts into CSS.

Some of the Features provided by the SASS

- **Variables**
- **Nesting**
- **Loops**
- **Arguments**

Variables

Sass allows variables to be defined. Variables begin with a dollar sign ($). Variable assignment is done with a colon (:).



SCSS

```scss
$primary-color: #3bbfce;
$margin: 16px;

.content-navigation {
  border-color: $primary-color;
  color: darken($primary-color, 10%);
}

.border {
  padding: $margin / 2;
  margin: $margin / 2;
  border-color: $primary-color;
}
```

Sass

```sass
$primary-color: #3bbfce
$margin: 16px

.content-navigation
  border-color: $primary-color
  color: darken($primary-color, 10%)

.border
  padding: $margin/2
  margin:  $margin/2
  border-color: $primary-color
```

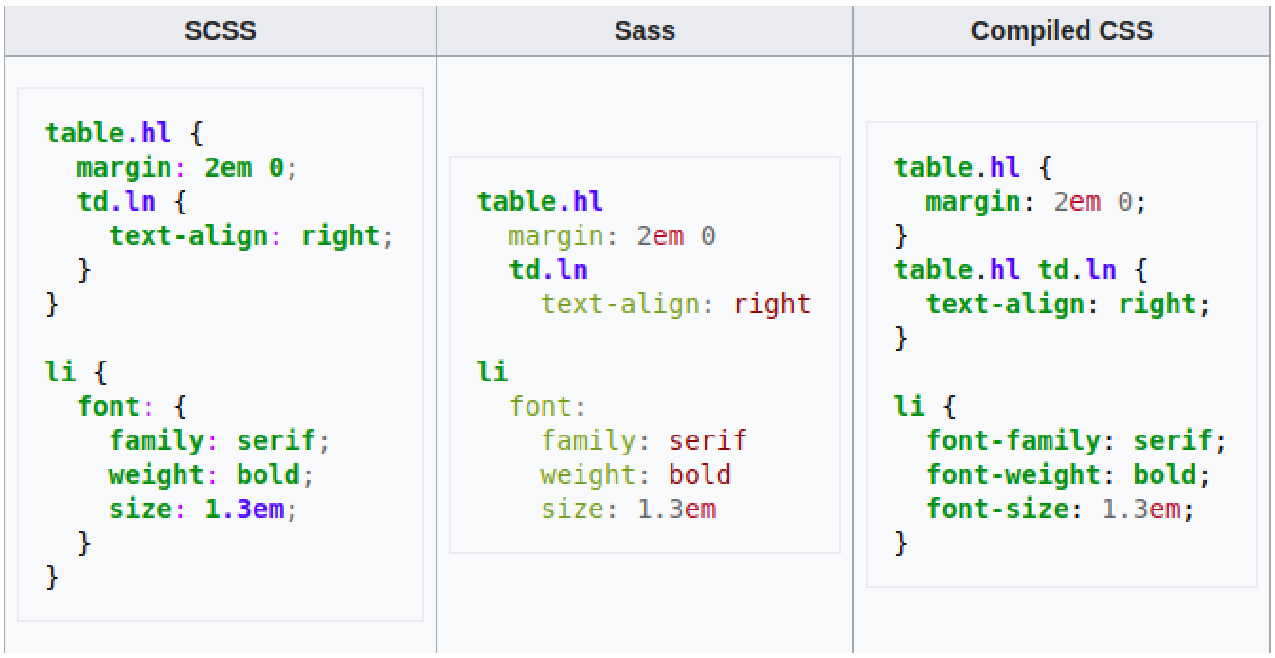
Compiled CSS

```css
.content-navigation {
  border-color: #3bbfce;
  color: #2b9eab;
}

.border {
  padding: 8px;
  margin: 8px;
  border-color: #3bbfce;
}
```

Nesting

Sass allows variables to be defined. Variables begin with a dollar sign ($). Variable assignment is done with a colon (:).



SCSS

```scss
table.hl {
  margin: 2em 0;
  td.ln {
    text-align: right;
  }
}

li {
  font: {
    family: serif;
    weight: bold;
    size: 1.3em;
  }
}
```

Sass

```sass
table.hl
  margin: 2em 0
  td.ln
    text-align: right

li
  font:
    family: serif
    weight: bold
    size: 1.3em
```

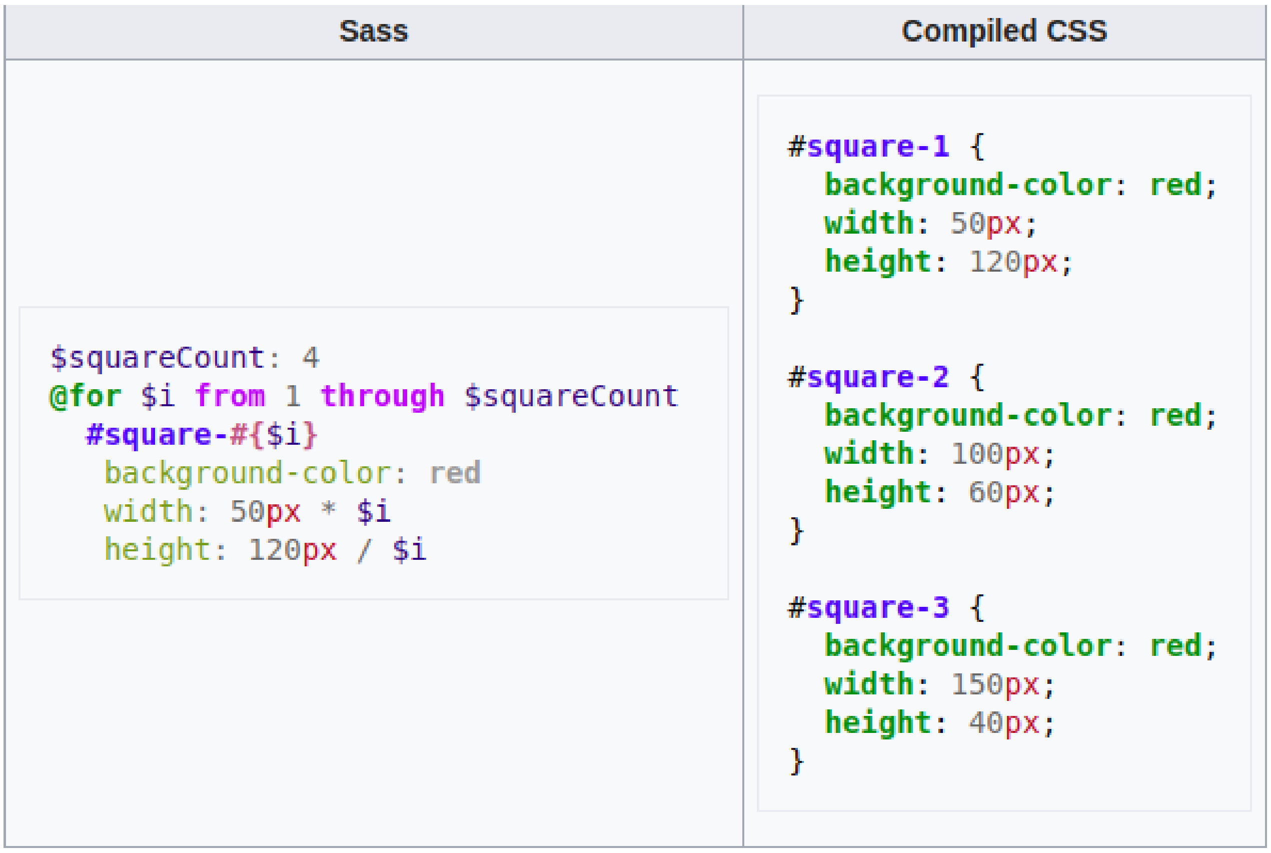
Compiled CSS

```css
table.hl {
  margin: 2em 0;
}
table.hl td.ln {
  text-align: right;
}

li {
  font-family: serif;
  font-weight: bold;
  font-size: 1.3em;
}
```

Loops

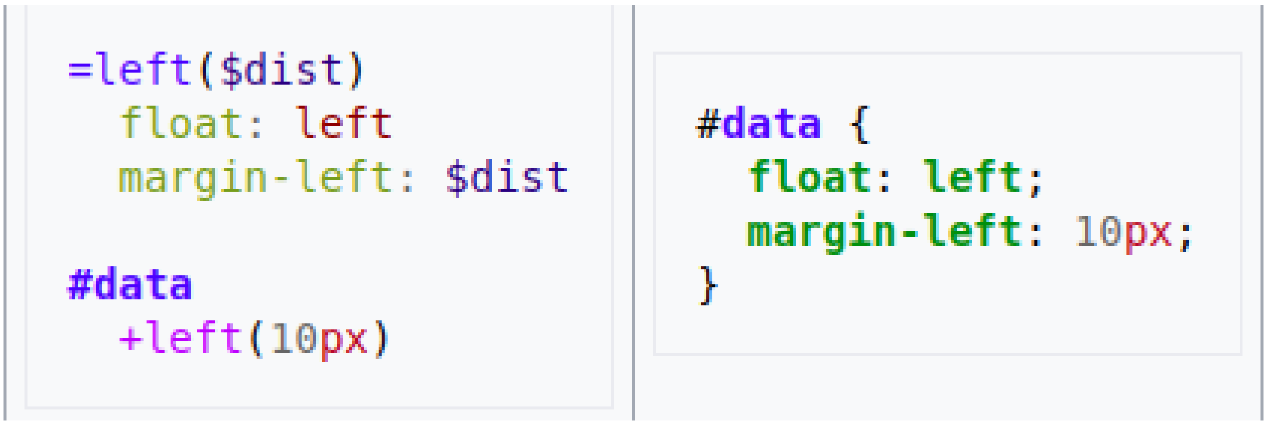
Sass allows for iterating over variables using **@for, @each and @while**, which can be used to apply different styles to elements with similar classes or ids.



Sass	Compiled CSS

`$squareCount: 4`
`@for $i from 1 through $squareCount`
`  #square-#{$i}`
`   background-color: red`
`   width: 50px * $i`
`   height: 120px / $i`
	`#square-1 {`
	`  background-color: red;`
	`  width: 50px;`
	`  height: 120px;`
	`}`
	
	`#square-2 {`
	`  background-color: red;`
	`  width: 100px;`
	`  height: 60px;`
	`}`
	
	`#square-3 {`
	`  background-color: red;`
	`  width: 150px;`
	`  height: 40px;`
	`}`

Arguments



```
=left($dist)
  float: left
  margin-left: $dist

#data
  +left(10px)
```

```
#data {
  float: left;
  margin-left: 10px;
}
```

2. Zurb Foundation

Foundation is a responsive front-end framework. Foundation provides a responsive grid and HTML and CSS UI components, templates, and code snippets, including typography, forms, buttons, navigation and other interface elements, as well as optional functionality provided by JavaScript extensions. Foundation is an open source project, and was formerly maintained by ZURB. Since 2019, Foundation has been maintained by volunteers.

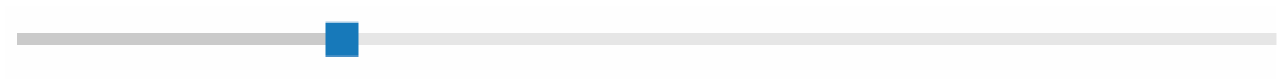
Foundation was designed for and tested on numerous browsers and devices. It is a mobile first responsive framework built with Sass/SCSS giving designers best practices for rapid development. The framework includes most common patterns needed to rapidly prototype a responsive site. Through the use of Sass mixins, Foundation components are easily styled and simple to extend.Since version 2.0 it also supports responsive design. This means the graphic design of web pages adjusts dynamically, taking into account the characteristics of the device used (PC, tablet, mobile phone). Additionally, since 4.0 it has taken a mobile-first approach, designing and developing for mobile devices first, and enhancing the web pages and applications for larger screens.Foundation is open source and available on GitHub. Developers are encouraged to participate in the project and make their own contributions to the platform.

- For demonstration let's say we want to make a slider
- In a traditional way we will have to write JavaScript, CSS and HTML code for that
- But Foundation makes it much simpler for use, we just need create the HTML markup with some pre-built classes provided by Zurb Foundation and That's it's done. Zurb will automatically create the slider for us.
- Just 4 to 5 of code will be enough for us to create the slider

Code

```
<div class="slider" data-slider data-initial-start="50" data-end="200">
  <span class="slider-handle"  data-slider-handle role="slider" tabindex="1"></span>
  <span class="slider-fill" data-slider-fill></span>
  <input type="hidden">
</div>
```

Output



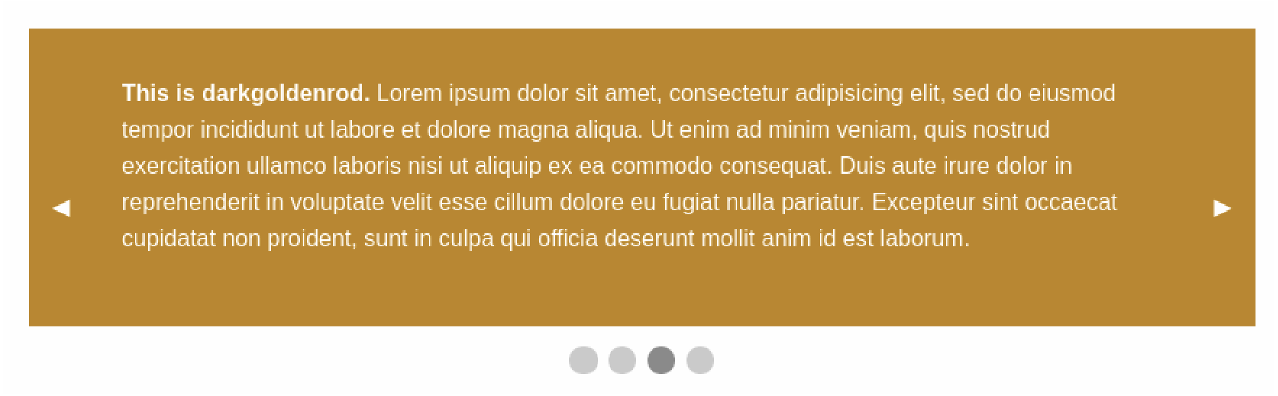
- We can clearly see that we didn't used any JavaScript and CSS.
- It's just a HTML markup and our job is done.

- Similarly let's say we want to create some sort of content carousel.
- Again it's very simple.
- Just include the markup along with the classes provided by the Zurb foundation and the Job is done, our content slider is ready.
- It's that simple.

Code

```
<li class="orbit-slide">
  <div>
    <h3 class="text-center">2: You can also throw some text in here!</h3>
    <p class="text-center">Lorem ipsum dolor sit amet, consectetur adipisicing elit. U
    <h3 class="text-center">This Orbit slider does not use animations.</h3>
  </div>
</li>
```

Output



This is darkgoldenrod. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

3. Panini

- Traditionally if, we are creating some static website, lets say we have five pages that all shared the same header and footer. We created our first page and then copy and paste the common elements to the next page. But now if we need to make a change to the header, the change has to be made across multiple files.
- Here comes **Panini.**
- **Panini** is a flat file compiler that uses the concepts of templates, pages and partials - powered by the **Handlebars Templating language** to streamline the process of creating static prototypes

Some of the key features provided by Panini

- **Templates and Pages**
- **Partials**
- **Page Variables**
- **Helpers**

Templates & Pages

A template is a common layout that every page in your design shares. It's possible to have multiple templates, but generally you'll only need one, and a page can only use one template. In the prototyping template, the default layout is found under **src/layouts/default.html**.

Here's what a basic template might look like:

```
<html>
  <head>
    <title>Definitely a Website!</title>
  </head>
  <body>
    <header class="header"><!-- ... --></header>
    {{> body}}
    <footer class="footer"><!-- ... --></footer>
  </body>
</html>
```

In the middle of the HTML is a bit of Handlebars code: **{{> body}}**. This is where the pages you write are injected when Panini runs, giving you a series of complete HTML files at the end.

The pages make up the guts of your layouts. These files will just have the middle section of the design, since the layout already covers the top and bottom. The prototyping template includes one blank page to get you started, under **src/pages/index.html**.

A basic page might look like this:

```
<h1>Page Title</h1>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
```

Partials

Partials are a feature of Handlebars which allow you to inject HTML anywhere in a page or layout. They're really useful when you need to repeat certain chunks of code throughout your pages, or to keep individual files from getting too cluttered with HTML.

Here's an example of a layout file that divides its key sections into partials:

```html
<html>
  <head>
    <title>Definitely STILL a Website!</title>
  </head>
  <body>
    {{> header}}
    {{> navigation}}
    {{> body}}
    {{> footer}}
  </body>
</html>
```

The **{{> }}** syntax tells Handlebars to look for an HTML file with that name, and inject it at that place. In this example, we have files called **header.html, navigation.html,** and **footer.html.** In the prototyping template, these files all exist within src/partials.

Page Variables

Pages have a few built-in variables, which can be used within the page template itself, or within a layout or partial being used in tandem with the page.

page

Prints the name of the current page, without its original file extension. In the below example, if the page is **index.html, {{page}}** will become **index**.

```
<p>You are here: {{page}}</p>
```

root

Use **{{root}}** before a file path to make sure it works no matter what folder the current page is in.

For example, a path to an external CSS file will need to be different if the current page is at the root level of your site, or in a sub-folder.

Here's how you'd use it with a **<link>** tag:

```
<link rel="stylesheet" href="{{root}}assets/css/app.css">
```

If the page is **index.html**, the path will look like this:

```
<link rel="stylesheet" href="assets/css/app.css">
```

Helpers

Helpers are special functions that manipulate content on the page.

ifequal

Displays the HTML inside the helper if the two values are equal.

```
{{#ifequal foo bar}}
  <p>foo and bar are equal</p>
{{else}}
  <p>foo and bar are not equal}}
{{/ifequal}}
```

ifpage

Displays the HTML inside the helper only on specific pages. In the below example, the HTML inside the helper will only show up on the **index.html** page.

```
{{#ifpage 'index'}}
  <p>This is definitely the Index page.</p>
{{/ifpage}}
```

repeat

Repeats the content inside of it **n** number of times. Use this to easily print lots of duplicate HTML in a prototype.

```
<ul>
  {{#repeat 5}}
  <li>Five hundred ninety-nine US dollars</li>
  {{/repeat}}
</ul>
```

4. Drupal Theming

A theme is a collection of files that define the presentation layer. You can also create one or more "sub-themes" or variations on a theme.

Topics for this section:-

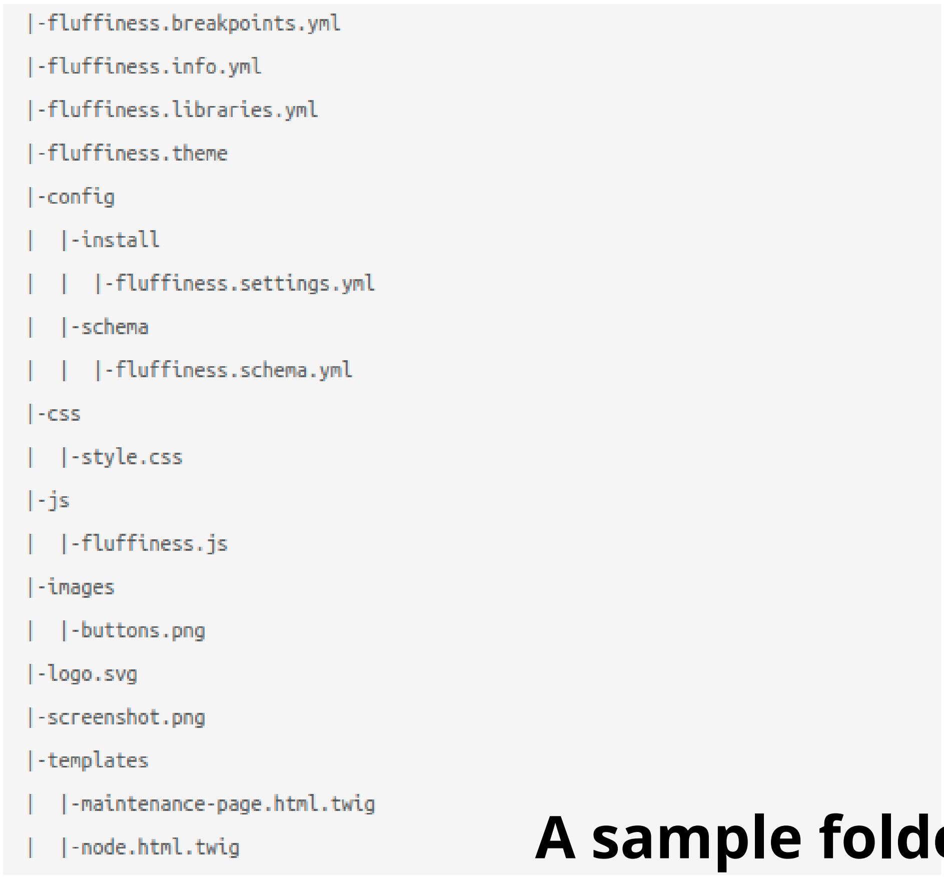
- Defining a theme with an **.info.yml** file
- Drupal theme folder structure
- Adding Regions to a Theme
- Adding stylesheets (CSS) and JavaScript (JS) to a Drupal theme
- Twig in Drupal
- Creating sub-themes

Defining a theme with an .info.yml file

- To create a Drupal8 theme you need to create a **theme_name.**info.yml file that provide the meta-data about your theme to Drupal 8
- We have to create .info.yml file in the root of the theme folder. The folder should have the same name as the .info.yml file for example if the name of theme is fluffiness then the info file will be named as **fluffiness**.info.yml

Theme folder structure

- You must place the themes in "themes" folder of your Drupal installation
- It is good practice to place the contributed themes in a sub folder named "contrib" and your own themes in a folder called "custom".



```
|-fluffiness.breakpoints.yml
|-fluffiness.info.yml
|-fluffiness.libraries.yml
|-fluffiness.theme
|-config
|  |-install
|  |  |-fluffiness.settings.yml
|  |-schema
|  |  |-fluffiness.schema.yml
|-css
|  |-style.css
|-js
|  |-fluffiness.js
|-images
|  |-buttons.png
|-logo.svg
|-screenshot.png
|-templates
|  |-maintenance-page.html.twig
|  |-node.html.twig
```

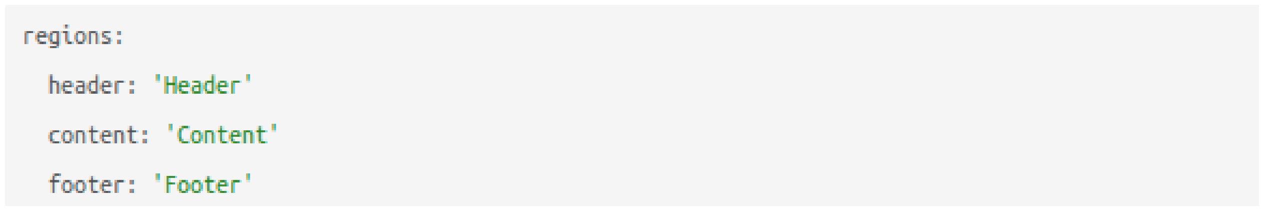
A sample folder structure

Adding Regions

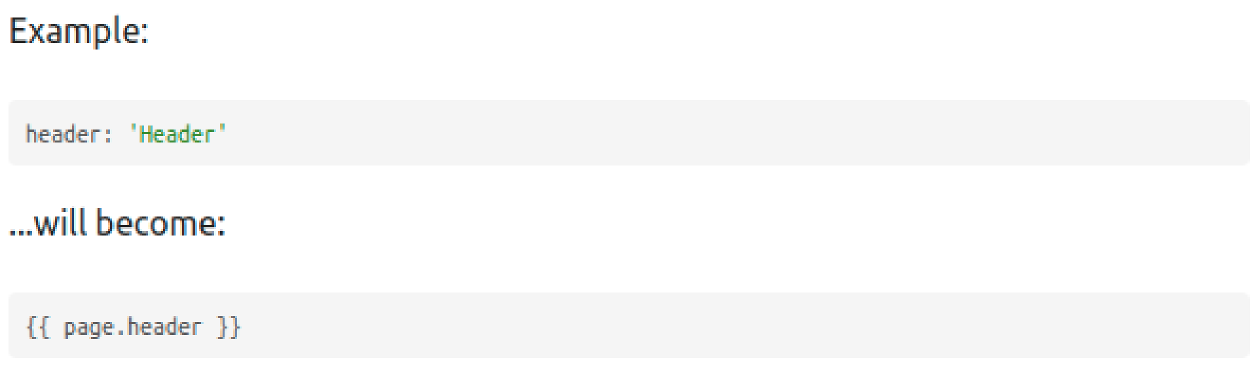
- Adding region meta-data to your THEMENAME.info.yml file.
- Editing your page.html.twig file and printing the new regions.

Adding Regions to Your Info File
Start by declaring any new regions in your THEMENAME.info.yml file. Regions are declared as children of the regions.
Adding Regions to Your Templates

In order for regions to display any content placed into them, you'll need to make sure your new regions are also added to your page.html.twig file.



```
regions:
  header: 'Header'
  content: 'Content'
  footer: 'Footer'
```



Example:

```
header: 'Header'
```

...will become:

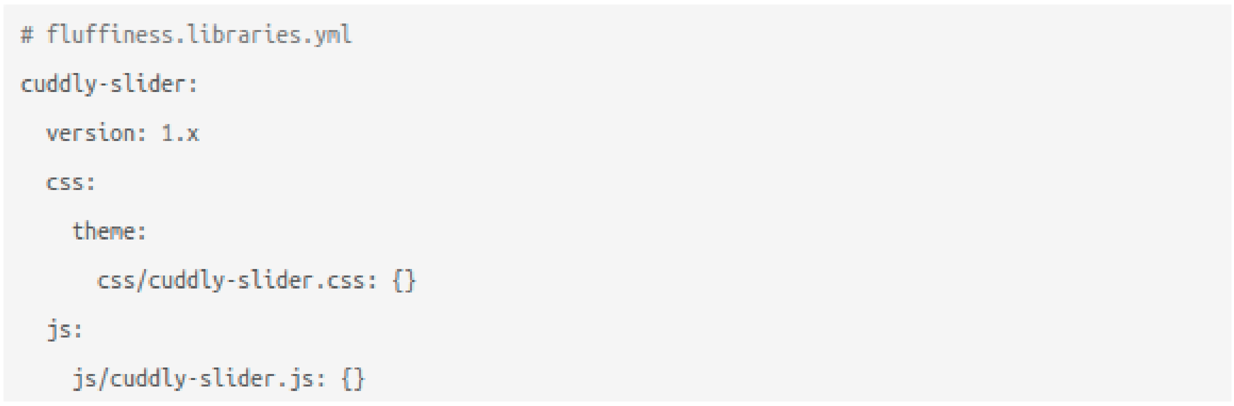
```
{{ page.header }}
```

Adding CSS and JS

- Save the CSS and JS files folder css and js respectively.
- Define a library which registers these CSS and JS files with your theme.
- Attach the library to pages where we want to apply the css and js

Defining a library

Define all of your asset libraries in a *.libraries.yml file in your theme folder. If your theme is named fluffiness, the file name should be fluffiness.libraries.yml. Each "library" in the file is an entry detailing CSS and JS files (assets), like this:



```
# fluffiness.libraries.yml
cuddly-slider:
  version: 1.x
  css:
    theme:
      css/cuddly-slider.css: {}
  js:
    js/cuddly-slider.js: {}
```

In this example, the JavaScript: cuddly-slider.js and CSS cuddly-slider.css are located in the respective **js** and **css** directories of your theme.

Twig in Drupal

Twig is a template engine for PHP and it is part of the Symfony2 framework

Working With Twig Templates

Drupal allows you to override all of the templates that are used to produce HTML markup so that you can fully control the markup that is shown as output within a custom theme. There are templates for each page element ranging from the high level HTML to small fields.

Twig Template naming conventions

Drupal loads templates based on certain naming conventions. This allows you to override templates by adding them to your theme and giving them specific names.

Creating sub-themes

- Sub-themes are just like any other theme, with one difference: they inherit the parent theme's resources.
- To create a sub-theme, define it like any other theme and declare its base theme with the "base theme" key.

Example for the sub-theme

```yaml
name: Fluffiness
type: theme
description: This is a fluffy sub theme of Classy
core: 8.x
# Defines the base theme
base theme: classy
# Defines libraries group in which we can add css/js.
libraries:
  - fluffiness/global-styling
# Regions
regions:
  header: Header
  featured: Featured
  content: Content
  sidebar_first: First sidebar
  sidebar_second: Second sidebar
  footer: Footer
```

Bibliography

- https://www.drupal.org/docs/theming-drupal
- https://en.wikipedia.org/wiki/Foundation_(framework)
- https://en.wikipedia.org/wiki/Sass_(stylesheet_language)
- https://get.foundation/sites/docs/
- https://get.foundation/sites/docs/panini.html
- https://sass-lang.com/documentation

Thank You!