Build a site from a style guide

Exploring Fractal

Fractal is a Node application. To install it run the following command:

```
npm install -g @frctl/fractal
```

This will install a `fractal` command line tool.

```
fractal --help
┌──────────────────────────────────────────────────────────┐
| Fractal interactive CLI                                  |
|──────────────────────────────────────────────────────────|
| - Use the help command to see all available commands.    |
| - Use the exit command to exit the app.                  |
|──────────────────────────────────────────────────────────|
| Powered by Fractal v1.1.7                                |
└──────────────────────────────────────────────────────────┘

fractal ➤
```

Note that we're working with the published version, not the one

Core Concepts

Now that we've installed and run Fractal we'll explore some of the basic concepts. This is not an exhaustive reference, for that go to the Fractal Guide that covers this in a lot more detail. This document covers how I use the tool.

View templates

Fractal can use any template engine we want. By default it uses handlebars with

additional extensions. A basic Handlebar template looks like this:

```
<div class="entry">
    <h1>{{ title }}</h1>
    <div class="body">
    {{ body }}
    </div>
</div>
```

This will take the `title` string from a configuration file or as passed to the element and use it inside the h1 element. It will do the same thing with the body content and the div in the body element.

We can create partial templates to build more comple layouts. For example we can create a page template, a header template and a footer-scripts template to hold different parts of our page.

The `header` template contains the head element and all its children. We use it because, other than the tilte, these elements are not likely to change. The template looks like this:

```
<head>
    <link rel="stylesheet" href="/styles/video-load.css">

    <meta charset="utf-8">
    <meta name="viewport"
      content="width=device-width,minimum-scale=1,maximum-scale=1">

    <title>{{title}}</title>
</head>
```

We use the `footer-scripts` to add scripts that we want to add to all content pages of our site. It contains all the scripts that I use so it may get pruned and made into different versions depending on what the specific page needs. The default script is shown below:

```
<script async src="/scripts/lazy-load.js"></script>
```

```
<script async src="/scripts/vend</script>-with-locales.js"></script>
<script async src="/scripts/vendor/prism.js"></script>
<script async src="/scripts/vendor/fontfaceobserver.standalone.js"></scrip
</script></script>
<script src="/scripts/lazy-load-video.js"></script>
```

The final template is for our basic pages. It will use the two partial templates we've seen before and build the page. The default Handlebars way looks like this:

```
<html>
{{> header }}
<body>

{{{ yield }}}

{{> footer-scripts }}
</body>
</html>
```

We can nest templates inside each other to make content even more moular. We could take our header and title element and make a template out of them. We'll look at this in more detail when we build a component.

If I'm working on a style guide then I will use them extensively. If I'm building the style guide as part of a larger project I may stick with standard handlebars or those available from the handlebars-helpers collection, I don't want to stay tied to Fractal any more than absolutely necessary, particularly if the project is for a client.

You can check the existing Fractal helpers in the documentation guide (Using Handlebars section). I will use the `render` helper to illustrate how they work.

```
<html>
{{render '@header' }}
<body>

{{{ yield }}}
```

```
{{render '@footer-scripts' }}
</body>
</html>
```

The result should be the same but behind the scenes Fractal will do additional work with the rendered template. We don't need to know what it is, it just works :)

Context data

Context data can be used to populate parts of the template. The two most common ways to do it are to create a .json file or to use YAML front matter at the top of the page. The idea is that you don't need to manually give titles or do things with the template, for example we could create the following JSON file for a component may look like this:

```
{
  "context": {
    "title": "Level 1 Heading",
    "Level 1 Heading""author": "Aragorn, King of the West"
  }
}
```

We could also provide the same data at the document level using YAML. The JSON file converted to YAML front matter works like this. We can extend this with additional information and data to work with our templates.

```
---
context:
  title: "Level 1 Heading"
  author: "Aragorn, King of the West"
---
```

Configuration files

Components, documentation pages and collections can all have their own (optional) configuration files associated with them.

In order to be recognised, configuration files must:

- Reside in the same directory as the item that they are configuring
- Have a file name in the format item-name.config.{js|json|yml} - for example button.config.json, patterns.config.js or changelog.config.yml

Some things to note:

- The javascript version (the one that exports a module) is not as strict as the JSON version in terms of quoting string attributes

```
module.exports = {
  title: 'Base Layout',
  status: 'prototype',
  context: {
    title: 'Click me!',
    author: 'Carlos',
  },
};
```

```
{
  "title": "Base Layou"Base Layout""status": "prototype",
  "context" "title": "Clic": "!",
    "author": "": ""author""author": "Carlos"
  }
}
```

Some configuration items will have their values inherited from upstream collections or their default settings if the values are not set in the item's configuration file directly.

This can also be thought of a cascade of configuration values from their default settings down through any nested collection configurations and into the item itself.

Naming & referencing

Fractal is a flat-file system, and makes use of some simple file and folder naming conventions to help it parse the file system and generate the underlying data

model.

One of the main disadvantages of flat-file systems is that when one item references another via a path, moving any of those items inevitably results in those links breaking. So Fractal also supports a reference system, whereby items can use ‘handles’ instead of paths to link parts of the system together.

Unless told otherwise, Fractal will infer the name of a component or documentation page from its view template file name (or the parent directory for ‘compound’ components). It will then use this name (plus some other information) to generate a handle for the item. Handles are what will be used to reference that item elsewhere around your project.

> Names and handles are both ‘slug’ type strings, and will contain only lowercase, alphanumeric characters plus underscores and dashes.

The name will also be used to generate a default label and a title for the item. Labels are the text that will be used when the item is referenced in any navigation (for example in the web UI) and the title value is the text that will be used anywhere else a human-readable name for the item is required.

For a template in blockquote-large.hbs:

```
├── components
|   └── blockquote-large.hbs
```

The following labels and slugs will be generated:

- name: blockquote-large
- handle: blockquote-large
- label: Blockquote Large
- title: Blockquote Large

Statuses

Pages can have statuses associated with them.

Each status has a colour and a label that can be displayed in the web UI (and

other places) to help people quickly understand the status of each component.

Fractal defines some default statuses, but you are free to define your own to suit the needs of your project, or customise the colours and labels associated with these statuses.

Building a component

In this example we'll build a header element with title and author children. We'll build two separate templates, the first one has the content of the template that we can style as needed.

```
  <h1>{{title}}</h1>

  <p>by</p>

  <h2>{{author}}
```

The second template takes the content template and inserts it into a header element. The cool thing is that we can start with as small templates as we want or need to acommodate atomic design principles

```
<header>
  {{> @head}}
</header>
```

Moving the components to Fractal

To move a component into Fractal do the following:

1. Create the Handlebars template
   - Make sure you've already created all the components that you'll use inside the current element
2. Create the configuration file for your component
3. Create the styles (note that these are for display only, the style should go

into your SASS or CSS)
4. Preview.

Static sites for Rapid Prototyping

Now that we have a way to create components and preview them using Fractal we can look at how we can use the same handlebars templates to build a static web site.