Building components with Vue

Originally I was going to build an entire site using Nuxt.js, the Vue to `create-react-app`

Some of the questions I hope to address

- Where do you draw the line between components and pages?
- Can you create SCSS/CSS for an app as external resources that get processed through the Nuxt.js pipeline
- Does Vue/Nuxt work for content heavy sites?

No Nuxt for me and what are the options

In the research I discovered that Nuxt has the same problem for me that `create-react-app` does… it forces you into a paradigm and it makes it harder to reason through processes in the documentation or trying to figure out why something is not working. There is also a mismatch between the version of Nuxt available on NPM and the version documented on the site.

So, while it may take more work, I've decided to go with a barebones vue CLI approach.

So the questions change slightly:

Some of the questions I hope to address

- Where do you draw the line between components and pages?
- Will external CSS/SASS work in a Vue component? Will it work in an application?
- Can we integrte Markdown directly into Vue?
- How well does Vue work with content prepared outside the ecosystem?
- Will `vue-cli-service` throw a fit if I don't configure all the resources through WebPack?
- Does Vue work for content heavy sites?

- How do we incorporate Vue components into

But even using the default CLI proved too problematic. Because I expect to have the

Getting Vue ready to work

Vue is component-based which means that, at a minimum, we need to add a development or production build oof Vue to the page, like so:

```
<!-- development version -->
<script
  src="https://cdn.j<script
  src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js">
</script>
```

Once we reach a point where we're comfortable we can use the minimized production build to remove

```
<!-- production version -->
<script
  src="https://cdn.<script
  src="https://cdn.jsdelivr.net/npm/vue">
</script>
```

An example component

I know I'll need a navigation component to use in multiple pages and, normally, I would just put it in the layout file (if working with handlebars or nunjucks) but working on a framework makes you think twice about what makes a component and what makes a page and how to integrate them.

There are many ways to define a component in Vue, I'm confused as to which method to use in what circumstances. For right now, I'll stick to the single file component paradigm. This is not ideal as I'd like to be able to reuse CSS outside of scoped components but I'm still learning Vue so this will be it.

If you have suggestions or comments, ping me on Twitter (@elrond25).

The Vue navigation component template for a navigation menu may look like this.

```
<template>
  <nav>
    <nav>      <li><a href='/'>Home</a></li<li>    <li><a href='/about/'>
    </ul>
  </nav>
</template<li>script>
  // data and methods would go here
</script>

<style scoped></script>
  nav ul {
    display: flex;
    flex-flow: row wrap;
    justify-content: space-between;
  }

  li {
    list-style: none;
    margin-left: -3em;
  }
</style>
```

we then save the component as `Navigation.vue` and it’s ready to be used in other components and pages.

For example, the template below saved as index.vue

```
<template>
  <div class<div class="container">gation />
    <!-- Rest of the page content -->
  </div>
</template>
```

```
<script>
import Navigation from './components/Navigation.vue'

export def</div>
  name: 'navigation',
import Navigation from './components/Navigation.vue'

export default {
  name: 'navigation',
  components: {
    Navigation,
  }
  props: {},
  data: {},
  methods: {},
}
```

We can then build additional components to integrate into our pages as needed by inserting them into the index.vue template, importing it, and adding it to the components object.

Slots

Instead of hardcoding the URL and the title we can use slots (first defined in the custom elements specification) as a way to compose our components with content from the parent.

To illustrate the concept I've created a `navigation-link` element to take advantage of slots. The idea is that all the content in the parent, in this case the text inside each `navigation-link` element will get added to the child element inside the slot.

The content inside the `navigation-link` becomes the text of the element when the parent provides no content. This way we will have a default value regardless of the content in the parent element.

```
<template>
  <a v-bind:<a v-bind:href="url">>Navigation Item</slot>
  </a>
</template>

<script></slot>
export default {
  data() {
    return {
      url: '',
    }
  }
}
</script>
```

I then modified the `Navigation.vue` component to use `navigation-link` and the slot. The idea is that whatever content is inside the opening and closing `navigation-link` tags will get pushed into the child element's slot. If the child element doesn't have a slot the content inside the parent will be ignored.

```
<template>
  <nav>
    <navigation-link url='/'>
      Home
    </navigation-link>
    <navigation-link url='/about/'>
      About
    </navigation-link>
  </nav>
</template>
```

This example is used to illustrate slot composition and links don't necessarily work. For some reason the first link you click works but subsequent links do not. To get fully functioning links look at Vue Router and the brief description later in the post.

Using the data object

There are times when we want to create place holders and populate them with data that is exclusive to each component. That's where the `data` object for the Vue instance comes in.

The template uses the double moustache {{}} as a place holder for the corresponding values in the data object for the module. Think of the data object as internal class variables.

```
<template>
  <div class<div class="post">{ title }}</h2>
    <h3>{{ author }}</h3>

    {{ c</h2>t }}
  </div>
</template>

<script></div>
export default {
  data: {
    title: 'My awesome blog post',
    author: 'James T. Kirk',
    content: 'hello world',
  }
}
</script>
```

Listeners and Methods

Building routes in Vue

```
// Imports
import Vue from 'vue'
impo'vue'eRouter from 'vue-router'
import Home from '@/component'vue-router'import About from '@/components/
import Projects from '@/components/Projects.vue'

'@/components/About.vue'// Tell Vue to use the router
Vue.use(VueRouter)

// Define router parametersouter({
  mode: 'history',
  base: __dirname,
  routes: [
    { path: '/', component: Home },
    { path: '/about', component: About},
    { path: '/projects' component: Projects, }
  ]
})

// Define the actual rou'history'// Define the actual route`
    <div id="app">
      <h1>Data Fetching</h1>
      <ul>
        <li><router-link to="/">/</router-link></li>
        <li><router-link to="/about">About</router-link></li>
        <li><router-link to="/projects">Projects</router-link></li>
      </ul>
      <router-view class="view"></router-view>
    </div>
  `
}).$mount('#app')
"app"`
    <div id="app">
      <h1>Data Fetching</h1>
      <ul>
        <li><router-link to="/">/</router-link></li>
        <li><router-link to="/about">About</router-link></li>
        <li><router-link to="/projects">Projects</router-link></li>
      </ul>
      <router-view class="view"></router-view>
    </div>
  `
```

Because our router implements the HTML5 history mode, we have to do some additional work on the server for this to work reliably

```
<IfModule mod_rewrite.c>
```

```
  RewriteEngine On
  RewriteBase /
  RewriteRule ^index\.html$ - [L]
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteRule . /index.html [L]
</IfModule>
```

```
location / {
  try_files $uri $uri/ /index.html;
}
```

```
{
  "hosting": {
    "public": "dist",
    "dist""rewrites": [
      {
        "source": "**",
        "destination": "/index.html"
      }
    ]
  }
}
```

Render a static site with Puppeteer

https://github.com/chrisvfritz/prerender-spa-plugin

Static Site Generators: Vuepress

https://vuepress.vuejs.org/