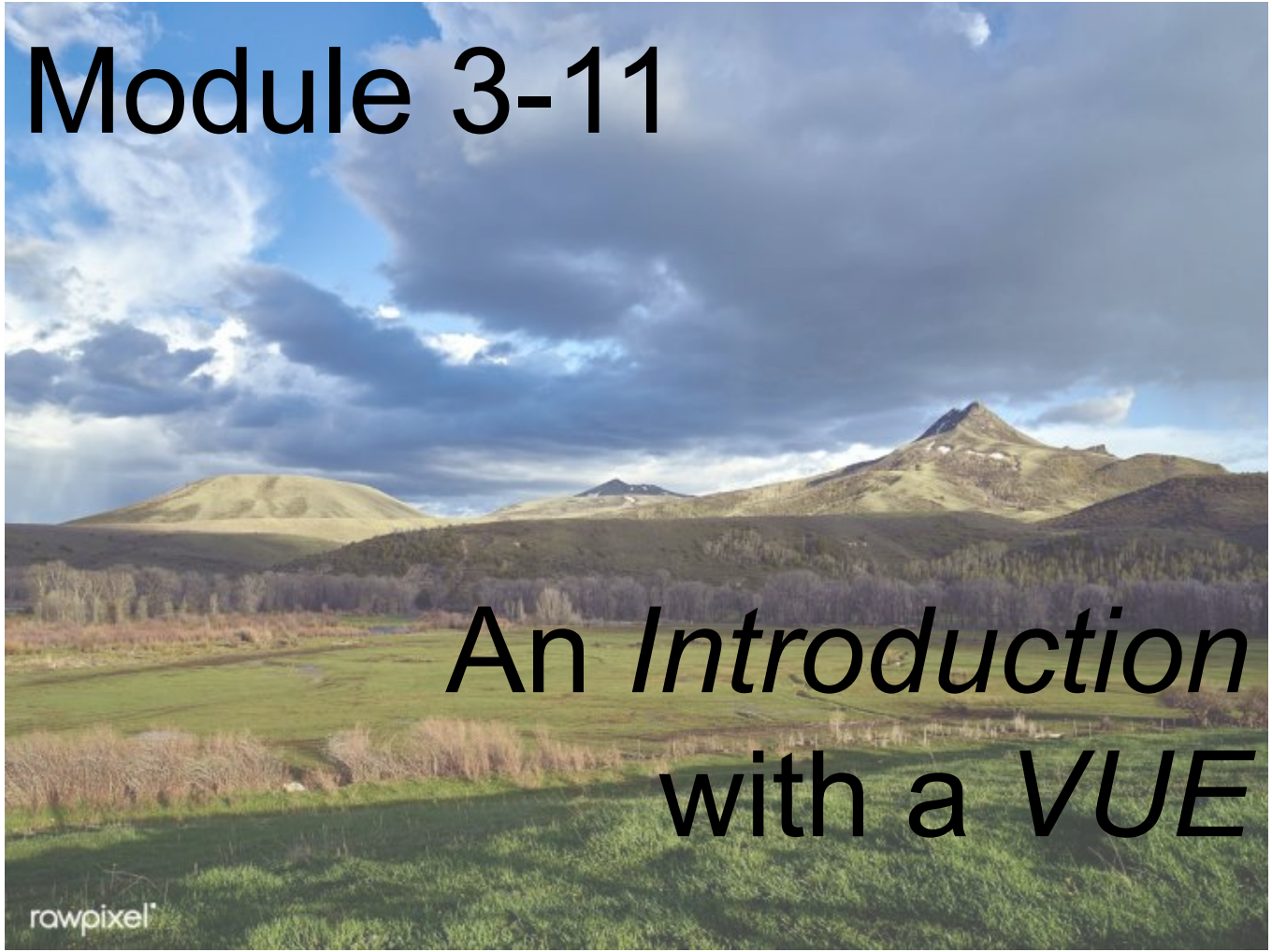
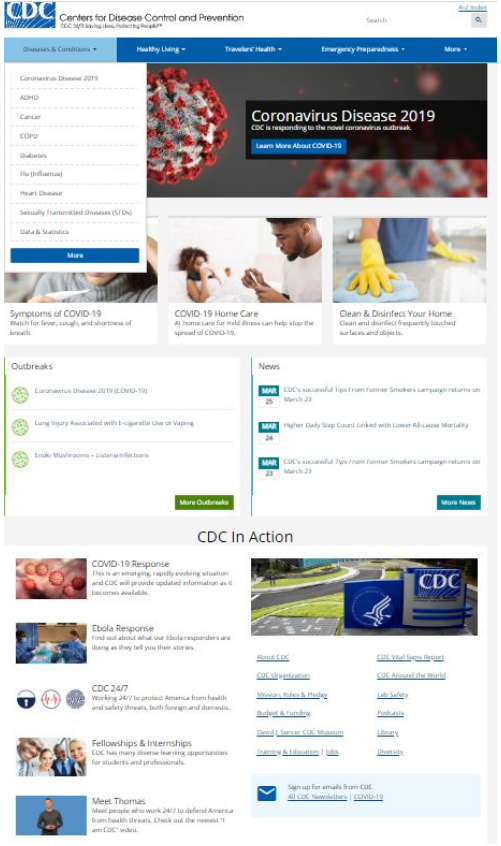
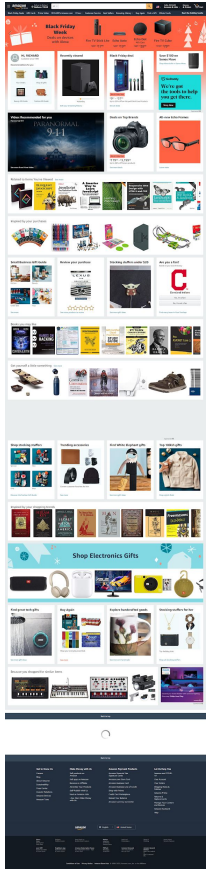

Module 3-11
An *Introduction*
with a *VUE*
rawpixel

Can You?

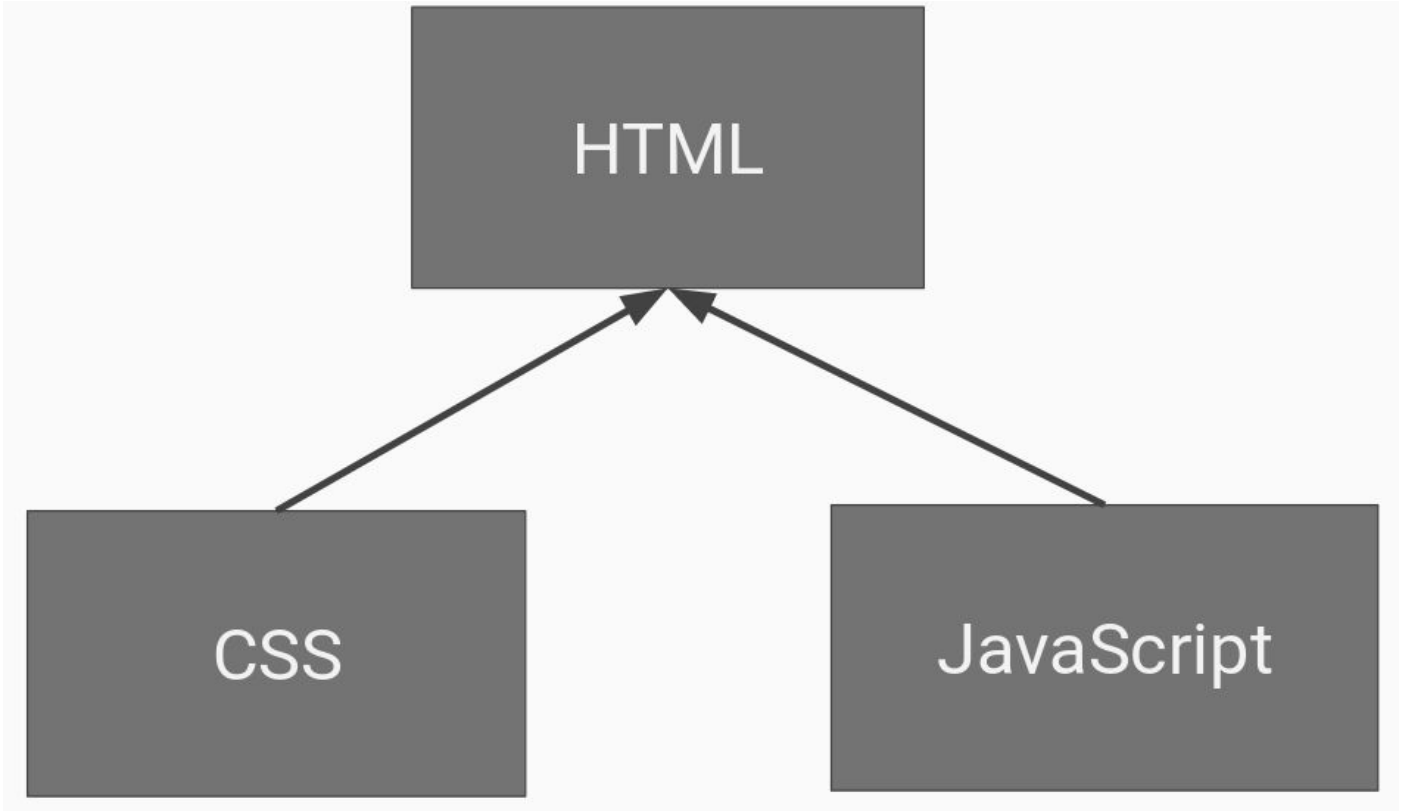
- Describe the Vue.js Component-Based JavaScript Framework
- Define a Vue Component
- Create a Vue project using the Vue CLI
- Create & Use Components
- Create JavaScript Objects
- Use the browser tools to Debug/Inspect Vue Applications
- Use v-model, v-bind:class, and v-for appropriately to create and modify DOM objects** (This will be covered in Lecture, refer to the LMS for “v-” directives)

Modern Websites are very Complex

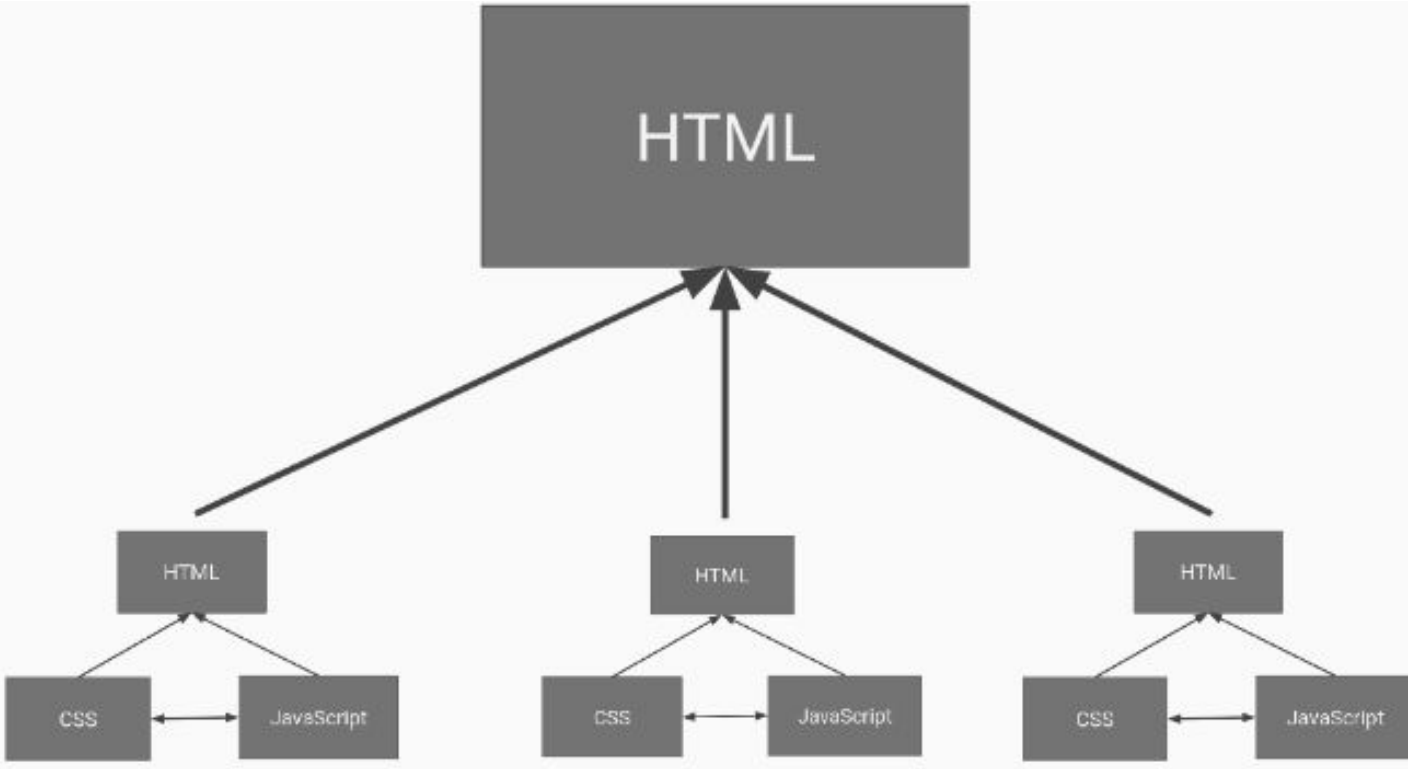




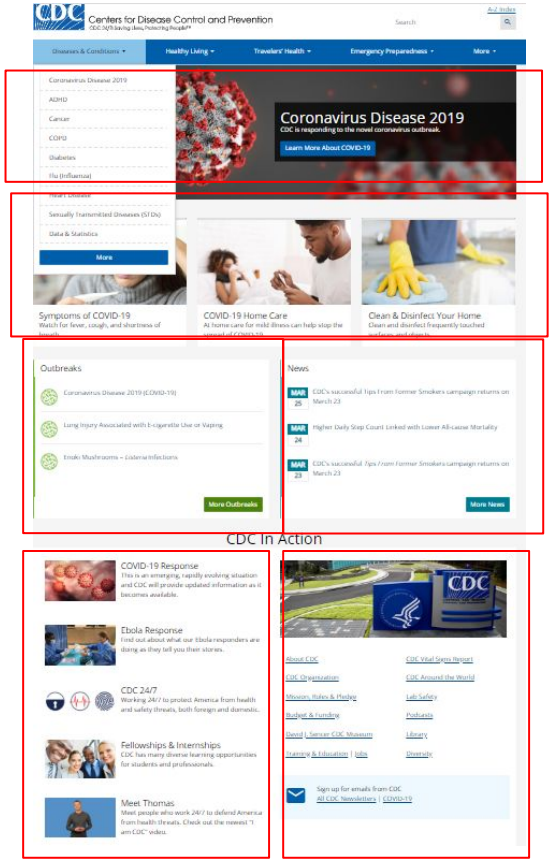
...Think of all the work required if using traditional design

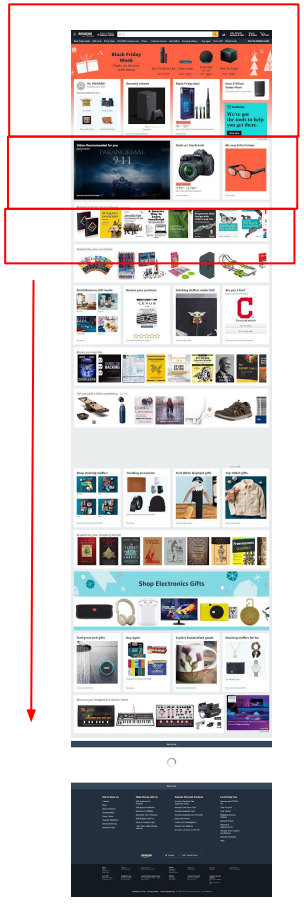


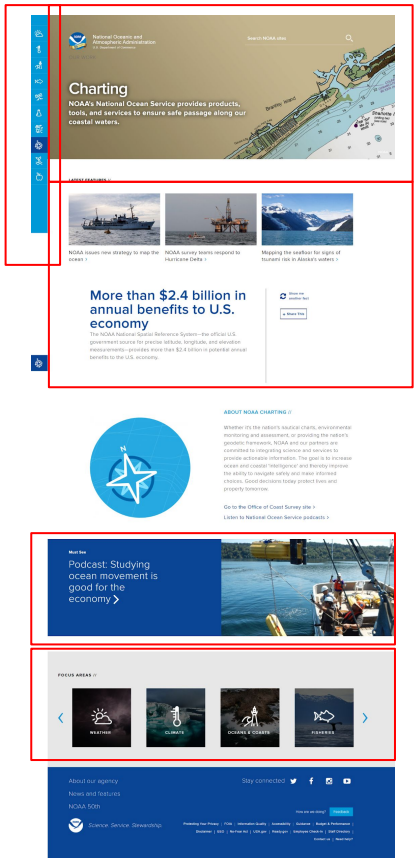
Component based JavaScript Frameworks simplify the creation and maintenance of these designs



A component-based approach



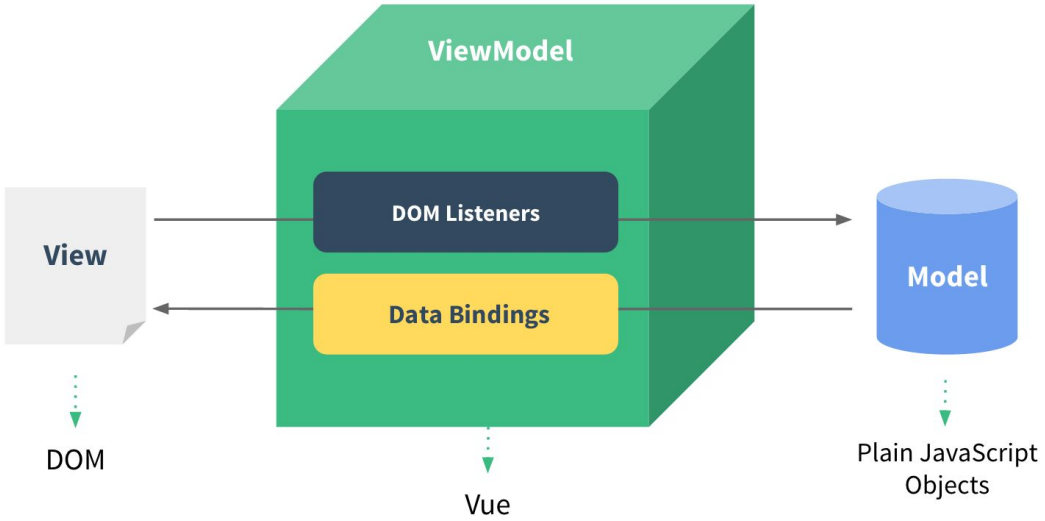




Vue.js: Component Based JS

- VUE.js is a component based JS Framework.
- A component is a reusable piece of code that behaves in a “plug and play” manner, which is to say that it is easily incorporated to other parts of the code base.
- Component based frameworks are very popular, some other frameworks you might have heard of are React and Angular.
- Here is some market share data on all of the various frameworks: The Top JavaScript Frameworks For Front-End Development in 2020

Vue.js Design



Let’s create a VUE project!

A Review of JS Objects

Remember that a JS Object is declared using the JSON notation & structure:

```
const employee = {
        firstName: "John",
        lastName: "Smith",
        age: 40,
        lang: ["English", "Spanish", "Esperanto"]
};
```

- The object itself is enclosed with a set of curly braces.
- Each property of the object is listed as a key value pair.
- An array is enclosed with square brackets.

A Review of JS Objects

An object can have other objects. Here, a company has a property called employees which contains an array of “people” objects.

```
const company = {
        employees: [
                {id: “1”, name; “Alice”},
                {id: “2”, name; “Bob”},
                {id: “3”, name; “Cindy”}
        ]
};
```

Remember that JavaScript objects can also contain functions! This is rare, but it can be done if needed. (See Defining Methods - MDN)

JS Objects in VUE

Within a VUE component's script block, the data function can define a JS object.

```
<script>
export default {
 name: 'product-review',
 data() {
   return {
     name: 'Widget',
     description: 'Working as intended.'
   }
 }
}
</script>
```

Here, we are returning a JS object with 2 properties, a name, and a description.

Anatomy of a VUE Component

```
<template>
   <div class="main">
       <h2>Product Reviews for {{ name }}</h2>

       <p class="description">{{ description }}</p>
   </div>
</template>

<style scoped>
div.main {
 margin: 1rem 0;
}
</style>

<script>
export default {
 name: 'product-review',
 data() {
   return {
     name: 'Widget',
     description: 'Working as intended.'
   }
 }
}
</script>
```

A VUE component is made of up of three parts:

- The **<template>** section which contains HTML elements.
- The **<style>** section which contains CSS styling.
- The **<script>** section contains the JavaScript code.

What the user sees on the screen is defined mostly by the HTML elements created in the template section and any CSS styles applied in the style section.

The behavior of the component is defined by what's in the script section.

Displaying Data on the Template

With the following script and template code blocks...

```
<script>
export default {
 name: 'product-review',
 data() {
   return {
     name: 'Widget',
     description: 'Works as expected.'
   }
 }
}
</script>
```

```
<template>
   <div class="main">
       <h2>Product Reviews for {{ name }}</h2>

       <p class="description">{{ description }}</p>

   </div>
</template>
```

… the HTML page will render the following:

Formatting the Template

The <style> section can contain any valid CSS rules. Consider the code below, which applies some formatting to a div with a class name of main (which happens to be the class name on the previous slide):

```
<style scoped>
div.main {
 margin: 1rem 0;
}
</style>
```

```
<template>
   <div class="main">
       <h2>Product Reviews for {{ name }}</h2>

       <p class="description">{{ description }}</p>

   </div>
</template>
```

Let’s create a fully functional component

Integrating All your Components

- The key benefit of using a component based framework is that we can take a bunch of components and bring them together to help us achieve our goal.

- After we've created all necessary components, we can integrate them into the App.vue file.

- Let's assume that the component we just build is called **ProductReview.vue**. Let's also assume that there is another component called **HelloWorld.vue**.

- Our goal is to display both of these components on the same HTML page.

The App.VUE file: The Main Page

To integrate the two components, we need to add the following code to App.VUE:

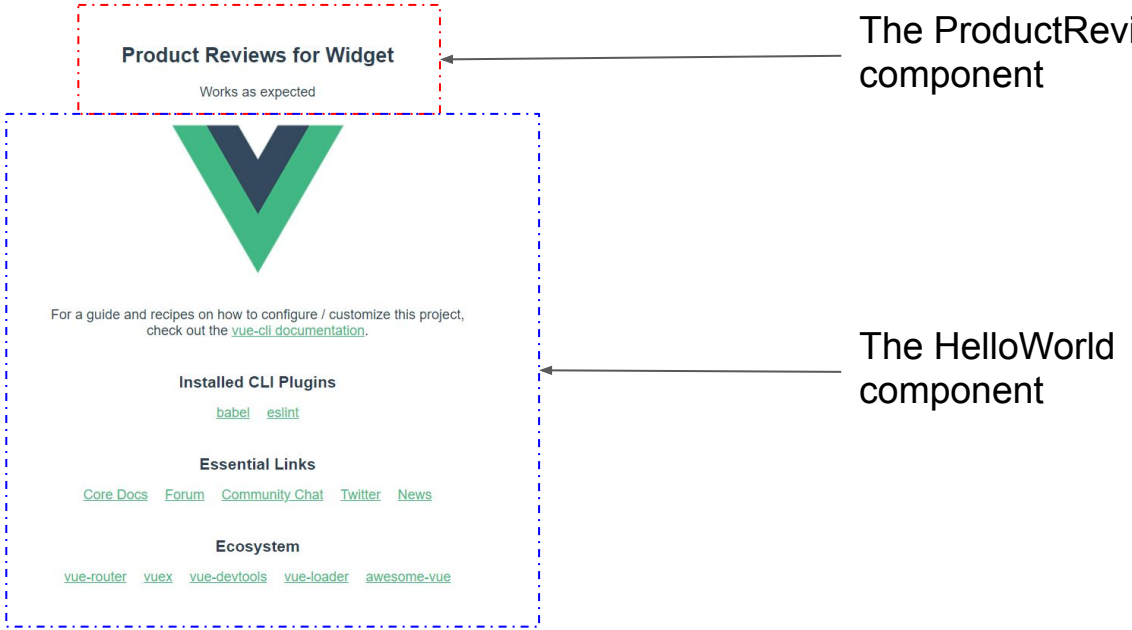
```
<template>
 <div id="app">
   <ProductReview/>
   <img alt="Vue logo" src="./assets/logo.png">
   <HelloWorld/>
 </div>
</template>

<script>
import HelloWorld from './components/HelloWorld.vue';
import ProductReview from './components/ProductReview.vue';

export default {
 name: 'app',
 components: {
   HelloWorld,
   ProductReview
 }
}
</script>
```

Here we have integrated both components into the main VUE app. Note the following checklist:

- You are rendering the components in the <template>
- In <script> the components have been imported
- In <script>, the components object is populated with the two component names.

Integrating All your Components

These are the results of our labor, two fully integrated components:



Let’s update App.vue