Components and the Vue Instance

NEXT: TEMPLATE SYNTAX

Let's talk for a while about the Vue Instance and what it has to offer. Your app needs to instantiate Vue before anything can go on:

```js
new Vue(/* Object argument */);
```

In the brand new app we created using the CLI, you can locate the instance in `src/main.js`:

```js
new Vue({
  el: '#app',
  router, // Ignore for now
  template: '<App/>',
  components: { App },
});
```

Let's briefly explain some of the properties of the object argument.

Table of Contents

Template

The template property is where you define component HTML templates. It can be a custom component that we will create ourselves as seen in the example above or it can be traditional HTML:

```js
new Vue({
  template: '<p>Hellllllooo</p>'
});
```

The template is what is printed to the view when a component is rendered. We will talk more on template in the next section

Data

Any data that you intend to bind to your view must be defined and returned in the data function:

```js
new Vue({
  template: '<p>{{greeting}}</p>',
  data () {
    return {
        greeting: 'Hellllllooo'
    }
  }
});
```

The data function returns an object. Any property in the object can be bound to the view and also accessible via `this`. Other functions that are available in the Vue instance object like `data` can access the data properties using `this`.

ES Alert: The data function might look weird, but it's just a shorthand for:

```js
{
    data: function() {
        return {
            greeting: 'Hellllllooo'
        }
    }
}
```

Methods

You can also define methods that should be accessible to the view using the `methods` member. Therefore, you can bind events to a function and handle the event by creating the function in the method's object:

```js
new Vue({
    template: `
        <div>
            <p>{{counter}}</p>
            <button v-on:click="incrementCounter">++</button>
        </div>
    `,
    data() {
        return {
            counter
        }
    },
    methods: {
        incrementCounter() {
            this.counter++
        }
    }
})
```

We have a property `counter` bound to the view. The expectation is that when the button is clicked, the counter should be incremented by 1. For that, we handle the click event my invoking the `incrementCounter` method bound the view. This method adds one to the counter property.

Components

Components are the building blocks for Vue.

Components are characterized with template and logic with data flowing from the logic to the template and events emitted from the template to the logic.

To simplify this a component has:

- Vue Instance ----DATA----> Template
- Template ----EVENT-----> Vue Instance

We used a click event earlier to pass data from the template to the Vue instance.

The Vue Instance itself is a component with a template. Components are composable which means a component can house another component in it. This is known as the component hierarchy which we will discuss later.

For a component to house another, you need to define the child component in the parent's component `components` object. Sounds twisted, yeah? Let's see an example:

```js
const GreetingComponent = {
    template: `<h1>Hi, you!</h1>`,
}

new Vue({
  el: '#app',
  template: `
    <div>
        <GreetingComponent/>
    </div>
  `,
  components: { GreetingComponent },
});
```

Components are simply objects. To add a child component to a parent component, declare the child component by adding it to the `components` object:

```js
components: { GreetingComponent },
```

Then you can print the components HTML template in the view:

```js
template: `
    <div>
        <GreetingComponent/>
    </div>
  `,
```

Mounting

What makes a large app is composing lots and lots of components together. But one component needs to be identified as the entry component. This is what Vue looks for to bootstrap your app.

The entry component must have an `el` property which points the DOM using a query selector. The pointer tells Vue where the entry component should be mounted:

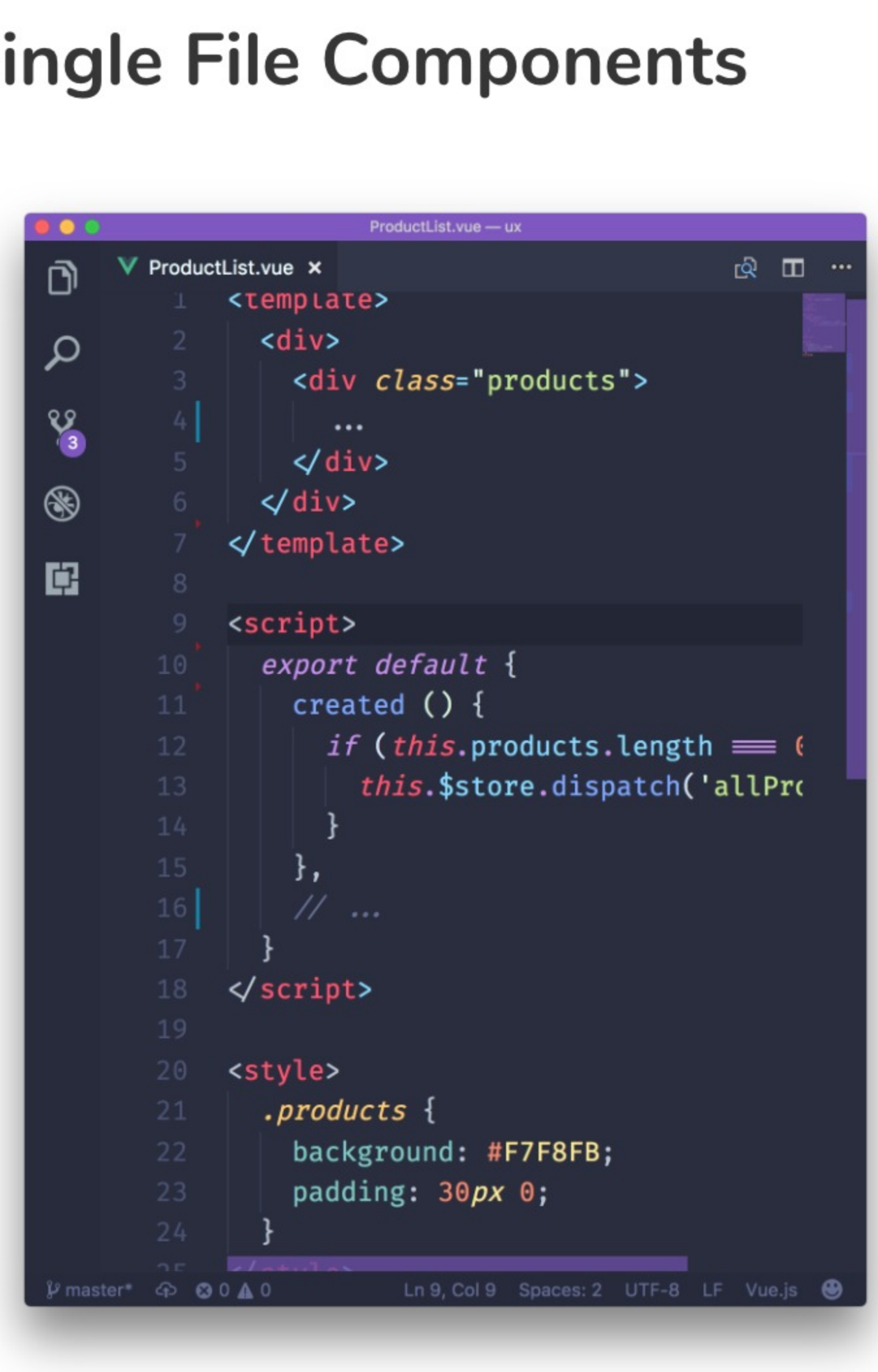
```js
new Vue({
  el: '#app',
  //
});
```

```html
<div id="app"></div>
```

If the `el` is not provided, components can be manually mounted using the `$mount` instance method:

```js
new Vue({}).$mount('#app')
```

Single File Components



For modularity sake, you can have your components' HTML, CSS and JavaScript code live in a single file with a `.vue` file extension:

```html
<!-- ./src/App.vue -->
<template>
  <div id="app">
    <img src="./assets/hello.png">
  </div>
</template>

<script>
export default {
  name: 'app',
};
</script>

<style>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

Single Files Components makes it easy to visualize a whole component in one file. This cuts down the need to navigate files while trying to sync changes between a component template and its logic or event styles.

With the Vue extension, which comes with VS Code, the .vue files get proper syntax highlighting.

By choice or brevity sake, you might want to split your CSS and JavaScript into different files. Vue still allows that:

```html
<!-- ./src/App.vue -->
<template>
  <div id="app">
    <img src="./assets/hello.png">
  </div>
</template>
<script src="./src/app.js"></script>
<style src="./src/app.css"></style>
```

Local vs. Global Components

The Single File Component discussed above is an example of a local component. It's only available where it was created or when imported in a different context.

Local components are plain JavaScript objects that are identical to the objects passed to a Vue instance. Therefore the data function, methods, lifecycle hooks, computed functions, component properties, etc. are all available.

Meanwhile, Vue allows you to create global components which are registered to the global context of your app and can be used anywhere without having to export/import the file it's contained in.

Global components are created using Vue's static `component` method:

```js
Vue.component('product', {
  template: '<div class="card">...</div>'
})
```

The `component` method takes two arguments. The component name and an object that describes the component. Just like local components, this object is identical to the object passed to a Vue instance.