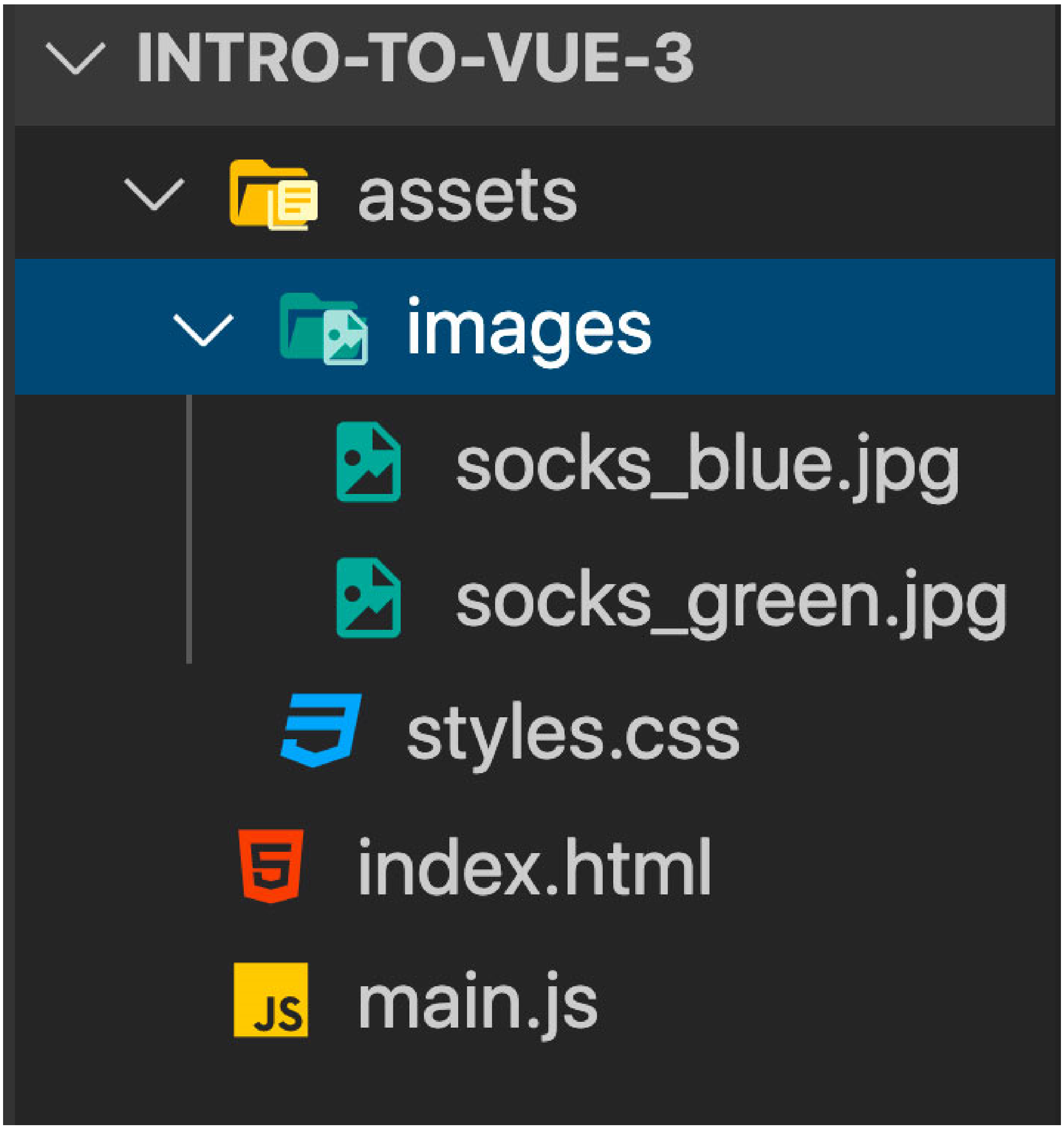
Creating the Vue App

Are you ready to create a Vue app? To get started with this lesson, you can either check out the starting code on the `L2-start` branch of the repo, or head over to CodePen to get started there.

Touring the Starting Code

To take a tour of the starting code, you'll see we have an assets directory. Inside of there, there's a directory for images. We've got one for blue socks, and green socks. We also have a CSS file for all of our styles.



```
INTRO-TO-VUE-3
  assets
    images
      socks_blue.jpg
      socks_green.jpg
    styles.css
  index.html
  main.js
```

Inside of the **index.html**, we're importing those styles. Below that, we are importing the Vue.js library. Importing the Vue library via a CDN link is simplest possible way to start using Vue. You'll notice at the time of this recording, I'm using the Vue 3 beta CDN link, but if you're watching this and Vue 3 is already out, you're going to want to go to the official Vue.js docs and copy the CDN link there ("For prototyping and learning purposes") and replace it in the script tag of your **index.html** file.

At the bottom of the file, we're importing our **main.js** file, which is pretty simple so far: just a `const product = 'Socks'`

main.js

main.js > product

```
const product = 'Socks'

```

You'll notice that in the template, there is an `h1` that says "Product goes here." So the question now is: **How do we display product using Vue?**

Creating a Vue App

To display our data within our HTML, we'll first have to create a Vue app. In our **main.js** file, we'll create our app with:

📄**main.js**

```
const app = Vue.createApp({})
```

As an argument, we're going to pass in an object and add a data property. This is going to be a function that returns another object, where we'll store our data. In here, we'll add `product` as a data item.

📄**main.js**

```
const app = Vue.createApp({
    data() {
        return {
            product: 'Socks'
        }
    }
})
```

Now we just need to make sure we're importing our Vue app into the index.html file.

📄**index.html**

```
<!-- Import App -->
<script src="./main.js"></script>
```

Mounting Our App

Now that we've created our app, we need to mount the app that we just created, into our DOM. We'll do that inside of a script tag, in our **index.html** file.

📄**index.html**

```
<!-- Mount App -->
<script>
  const mountedApp = app.mount('#app')
</script>
```

We'll say `app`, which refers to the app that we just created, and then `.mount()`, which is a method that requires a DOM selector as an argument. This lets us plug the Vue app into that piece of our DOM.

Displaying the Data

Now that we've created, imported and mounted out Vue app, we can now start displaying the data that lives within it.

To render the `product` data within the `h1`, we'll write:

📄**index.html**

```
<div id="app">
  <h1>{{ product }}</h1>
</div>
```

Now if we check the browser, we'll see "Product" is being displayed. Great! But how exactly is this working?

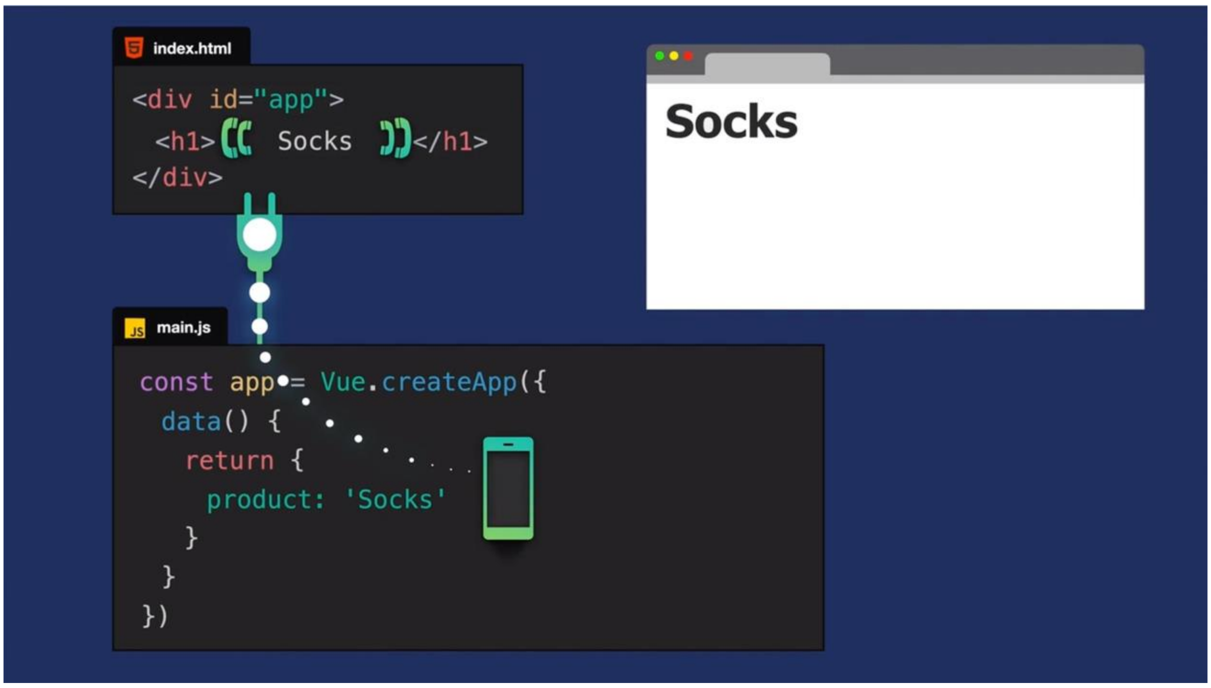
Understanding the Vue Instance

When we created our Vue app, we passed in the options object, which allowed us to add some optional properties to configure the application. Doing this creates our Vue instance, the heart of our Vue application, which powers everything.

📄**main.js**

```
const app = Vue.createApp({Options Object})
```

By importing this app, and mounting it to the DOM, we've essentially plugged the app into our DOM, giving our HTML a direct line into the app. This way, our template code can access options from the app, such as its data.



```
<div id="app">
  <h1>{{ Socks }}</h1>
</div>
```

```
const app = Vue.createApp({
  data() {
    return {
      product: 'Socks'
    }
  }
})
```

If you're wondering what's happening with this double curly brace syntax, you can imagine it like a phone, which has access to a phone within our Vue app. From our template, we're able to ask the app, "Hey, what's the value of product?" And the app responds, "Socks." When the page renders, we see "Socks" display on the page.

If this double curly brace syntax, or mustache syntax, is new to you, it allows us to write JavaScript expressions. In other words, it allows us to run valid JavaScript within our HTML.

Vue's Reactivity

What would happen if we changed the value of `product` from "Socks" to "Boots"?

main.js

```
const app = Vue.createApp({
    data() {
        return {
            product: 'Boots' // updated data value //
        }
    }
})
```

Because of how Vue works, the `h1`'s expression that is relying upon `product` would automatically receive that new value, and our DOM would update to display "Boots".

index.html

```
<div id="app">
  <h1>{{ product }}</h1> <! -- will reactively receive any updates to product -->
</div>
```

This is because Vue is *reactive*. Under the hood, Vue has an entire reactivity system that handles updates. When a data value changes, anywhere relying on that data will automatically update for us. We don't have to do anything to make that happen.

(As a side note, here on Vue Mastery, we have an entire Vue 3 reactivity course. So if you're interested in exploring the reactivity system under the hood, you can take that advanced course later.)

Coding Challenge

We’ve reached the end of this lesson, which brings us to the first coding challenge. You can find the solution code by checking out the `L2-end` branch of the repo, or viewing the solution on Codepen.

Add a `description` to the data object.

Display the `description` using an expression within a `p` tag.