Simple Custom File Selector with Vue.js

alligator.io/vuejs/file-select-component

File select elements are easily one of the ugliest input types on the web. They’re implemented differently in every browser and are generally incredibly ugly. There are some workarounds though, and we’ll show you one approach here using labels and a bit of Vue.js magic.

Setup

If you don’t have a project set up already, initiate a new one with vue-cli’s *webpack-simple* template.

```
$ npm install -g vue-cli
$ vue init webpack-simple ./file-upload
$ cd ./file-upload
$ npm install
```

Component Template & Styles

The goal of this component is simply to wrap a hidden *input type=”file”* element in a label, and display something else inside. This technique, while simple, is surprisingly effective.

FileSelect.vue (template)

```
<template>

  <label class="file-select">

    <div class="select-button">

      <span v-if="value">Selected File: {{value.name}}</span>
      <span v-else>Select File</span>
    </div>

    <input type="file" @change="handleFileChange"/>
  </label>
</template>
...
```

And now, some nice simple styles to give it a rather button-y look.

FileSelect.vue (styles)

```
...
<style scoped>
.file-select > .select-button {
  padding: 1rem;

  color: white;
  background-color: #2EA169;

  border-radius: .3rem;

  text-align: center;
  font-weight: bold;
}


.file-select > input[type="file"] {
  display: none;
}
</style>
```

The Logic

Files are a very special type to the browser, so there are a few special rules that make them a bit tricky to work with at times. (More on that here.) Despite this, we can actually get away with a very simple controller. It’s as short as any other custom input element.

FileSelect.vue (script)

```
<script>
export default {
  props: {

    value: File
  },

  methods: {
    handleFileChange(e) {

      this.$emit('input', e.target.files[0])
    }
  }
}
</script>
```

Usage

Now, we can import our new component in our app and use it like any other, with full *v-model* support.

App.vue

```
<template>
  <div>
    <p>My File Selector: <file-select v-model="file"></file-select></p>
    <p v-if="file">{{file.name}}</p>
  </div>
</template>

<script>
import FileSelect from './FileSelect.vue'

export default {
  components: {
    FileSelect
  },

  data() {
    return {
      file: null
    }
  }
}
</script>
```

Now you have full reactive access to files your users select through the component. You could then wrap it in an upload component, or process the data in some way with the FileReader API. Have fun!

Complete Component Code

For those of you who want everything at once, here you go! As promised, only 41 lines long. :)

FileSelect.vue

```
<template>
  <label class="file-select">
    <div class="select-button">
      <span v-if="value">Selected File: {{value.name}}</span>
      <span v-else>Select File</span>
    </div>
    <input type="file" @change="handleFileChange"/>
  </label>
</template>

<script>
export default {
  props: {
    value: File
  },

  methods: {
    handleFileChange(e) {
      this.$emit('input', e.target.files[0])
    }
  }
}
</script>

<style scoped>
.file-select > .select-button {
  padding: 1rem;

  color: white;
  background-color: #2EA169;

  border-radius: .3rem;

  text-align: center;
  font-weight: bold;
}

.file-select > input[type="file"] {
  display: none;
}
</style>
```