Web Application Development

Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology

http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Vue.js

PART 2

COMPONENTS

Overall Section Outline

1. **Introduction** – Why you should be using VueJS
2. **Terminology & Overview** – The critical foundation for understanding
3. **Declarative Rendering & Reactivity** – Keeping track of changes (Data Binding)
4. **Components** – Reusable functionality (Templates, Props & Slots)
5. **Routing** – Navigating the view (Router)
6. **Directives**– Extending HTML
7. **Event Handling** – Dealing with User Interaction
8. **Filters** – Changing the way we see things
9. **Computed Properties & Watchers** – Reacting to Data Change
10. **Transitioning Effects** – I like your <style>
11. **Case Study** – Labs in action

Vue.js

Overall Section Outline

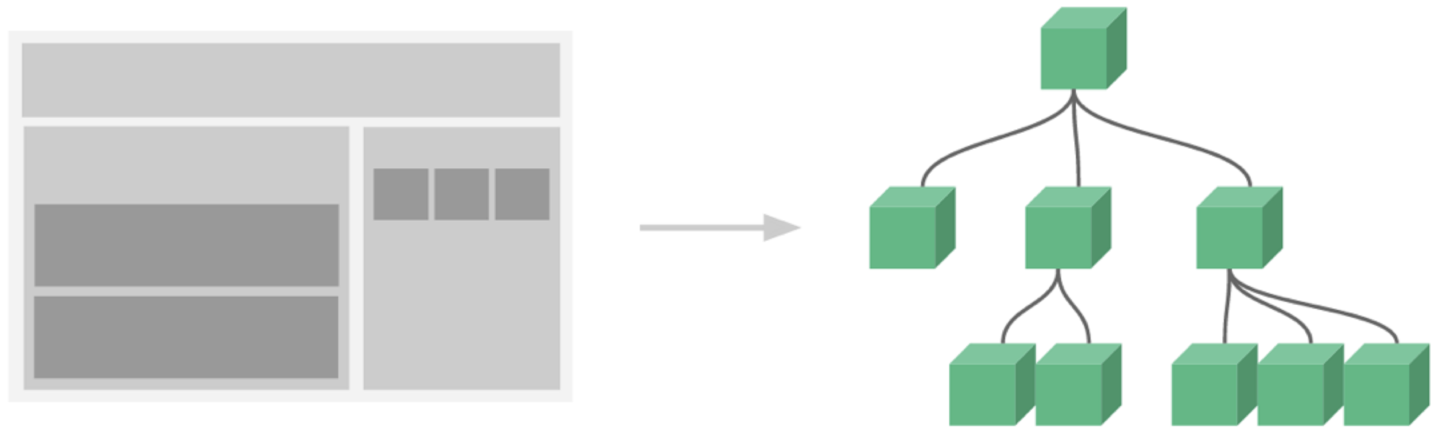
1. **Introduction** – Why you should be using VueJS
2. **Terminology & Overview** – The critical foundation for understanding
3. **Declarative Rendering & Reactivity** – Keeping track of changes (Data Binding)
4. **Components – Reusable functionality (Templates, Props & Slots)**
5. **Routing** – Navigating the view (Router)
6. **Directives**– Extending HTML
7. **Event Handling** – Dealing with User Interaction
8. **Filters** – Changing the way we see things
9. **Computed Properties & Watchers** – Reacting to Data Change
10. **Transitioning Effects** – I like your <style>
11. **Case Study – Labs in action**

Components

REUSABLE FUNCTIONALITY

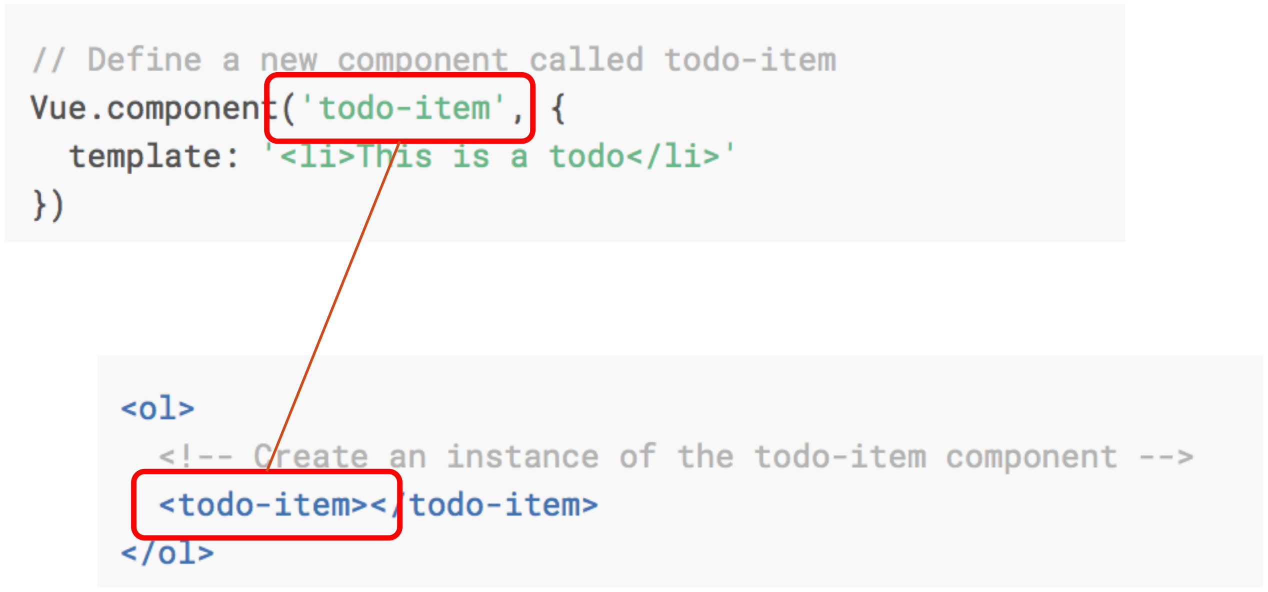
Introduction - Recap

Once again, as previously mentioned, the component system is another important concept in Vue (possibly ***the*** most important concept), because it’s an abstraction that allows us to build large-scale applications composed of small, self-contained, and often reusable components. If we think about it, almost any type of application interface can be abstracted into a tree of components:



Introduction - Recap

Components can be included in a single file:

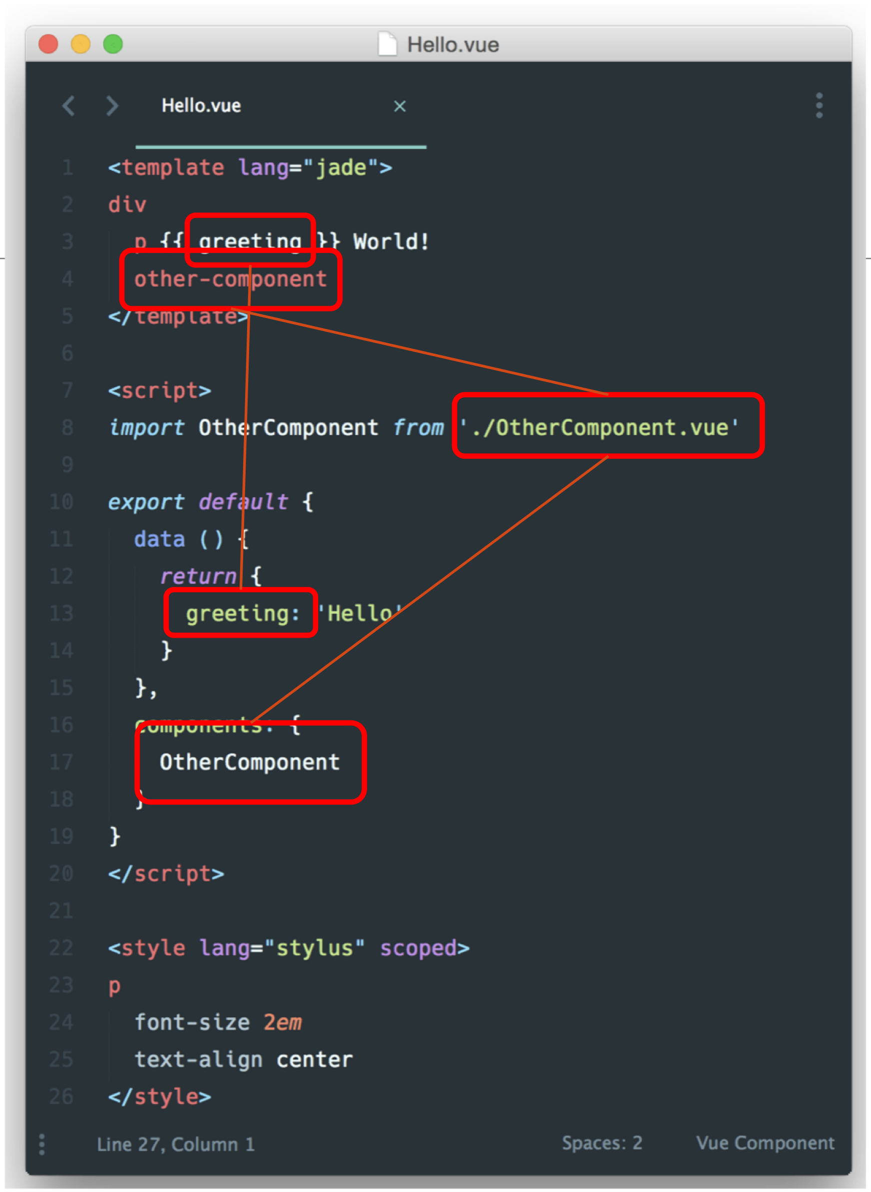


```
// Define a new component called todo-item
Vue.component('todo-item', {
  template: '<li>This is a todo</li>'
})
```

```
<ol>
  <!-- Create an instance of the todo-item component -->
  <todo-item></todo-item>
</ol>
```

Introduction - Recap

Or modularized into their own **.vue** files
(which is what we'll do)



Hello.vue

```
<template lang="jade">
div
  p {{ greeting }} World!
  other-component
</template>

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

export default {
  data () {
    return {
      greeting: 'Hello'
    }
  },
  components: {
    OtherComponent
  }
}
</script>

<style lang="stylus" scoped>
p
  font-size 2em
  text-align center
</style>
```

Line 27, Column 1   Spaces: 2   Vue Component

Components in Depth

In Vue, components are reusable Vue instances and need to have at least 2 things:

- A name (obvious)
- A template (The rendered DOM that belongs to each component)

and *because* they are reusable Vue instances they also accept the same options i.e.

- **`data`**
- **`computed`**
- **`watch`**
- **`methods`** and
- **lifecycle hooks** (**`mounted, unmounted`**) etc.

Reusing Components

Here’s an example of a Vue Component (in-line) :

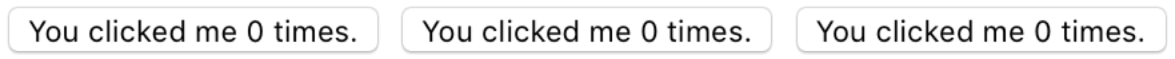
```
Vue.component('button-counter', {
        data: function () {
                return {
                   count: 0
                }
        },
        template: '<button v-on:click="count++">You clicked me {{ count }}
                        times.</button>'
})
```

Result in Browser

Reusing Components

Vue Components can be reused as often as you like:

```
<div id="components-demo">
  <button-counter></button-counter>
  <button-counter></button-counter>
  <button-counter></button-counter>
</div>
```



If you were to click on the buttons, each one will maintain its own, separate **count**. That's because each time you use a component, a new **instance** of it is created.

Reusing Components & **data**

** **data** Must Be a Function **

When we defined the **<button-counter>** component, you may have noticed that **data** wasn’t directly provided an object, like:

```
data: {
  count: 0
}
```

Instead, **a component’s data option must be a function**, so that each instance can maintain an independent copy of the returned data object:

```
data: function () {
  return {
    count: 0
  }
}
```

or

```
data() {
   return {
          count: 0
     }
}
```

If Vue didn’t have this rule, clicking on one button would affect the **data** of ***all other instances***,

Single-File Components & Registration

In-line components can work very well for small to medium-sized projects, where JavaScript is only used to enhance certain views. In more complex projects however, or when your frontend is entirely driven by JavaScript, these disadvantages become apparent:

- **Global definitions** force unique names for every component
- **String templates** lack syntax highlighting and require ugly slashes for multiline HTML
- **No CSS support** means that while HTML and JavaScript are modularized into components, CSS is conspicuously left out
- **No build step** restricts us to HTML and ES5 JavaScript, rather than preprocessors like Pug (formerly Jade) and Babel

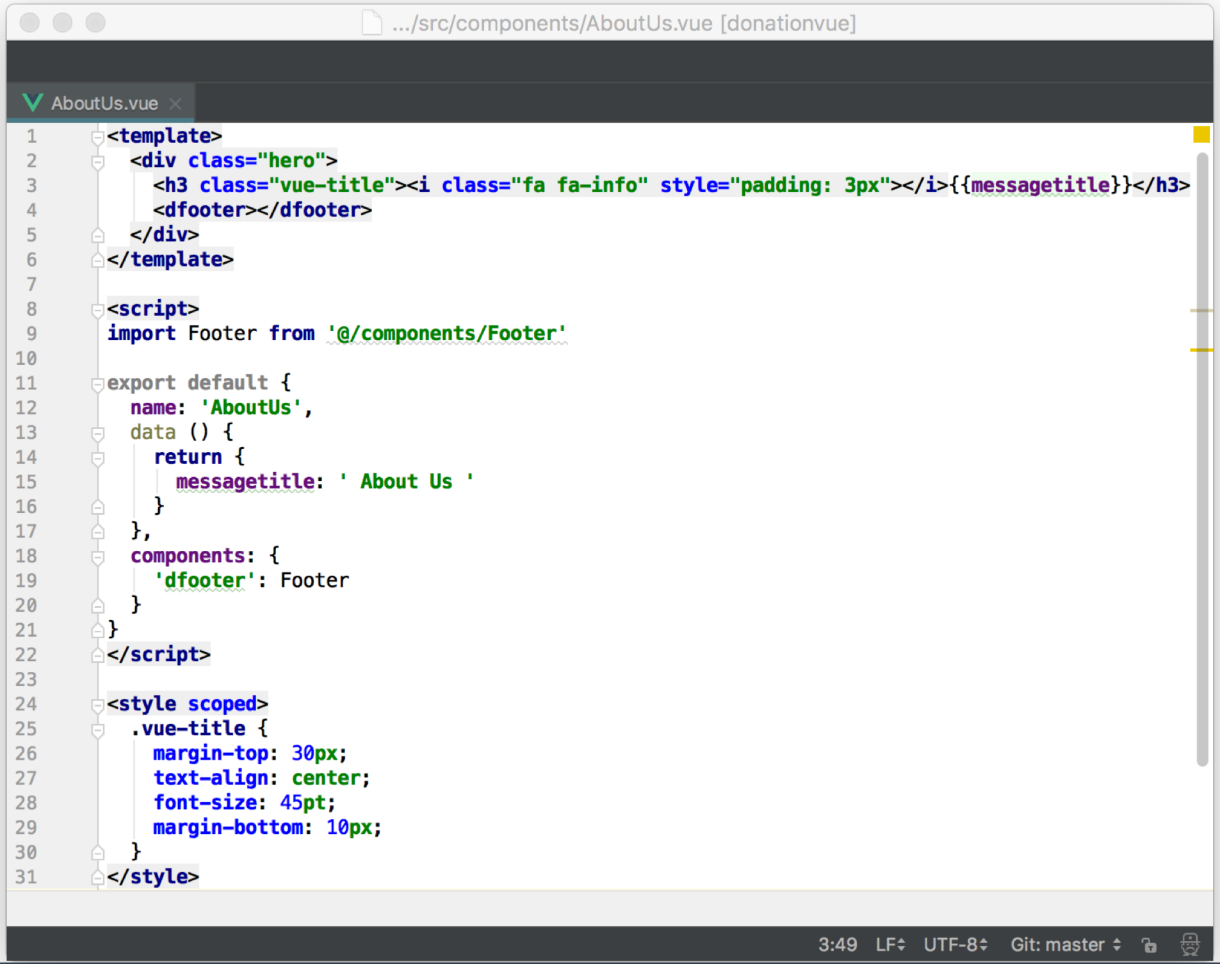
All of these are solved by **single-file components** with a **.vue** extension, made possible with build tools such as **Webpack** or Browserify.

Single-File Components & Registration

Here's our **AboutUs.vue**, Now we get:

- **Complete syntax highlighting**
- **CommonJS modules**
- **Component-scoped CSS**

All of this is possible thanks to the use of **Webpack**. The Vue CLI makes this very easy and is supported out of the box. **.vue** files **cannot** be used without a webpack setup, and as such, they are not very suited to apps that just use Vue on a page without being a full-blown single-page app (SPA).



.../src/components/AboutUs.vue [donationvue]
AboutUs.vue

```
<template>
  <div class="hero">
    <h3 class="vue-title"><i class="fa fa-info" style="padding: 3px"></i>{{messagetitle}}</h3>
    <dfooter></dfooter>
  </div>
</template>

<script>
import Footer from '@/components/Footer'

export default {
  name: 'AboutUs',
  data () {
    return {
      messagetitle: ' About Us '
    }
  },
  components: {
    'dfooter': Footer
  }
}
</script>

<style scoped>
  .vue-title {
    margin-top: 30px;
    text-align: center;
    font-size: 45pt;
    margin-bottom: 10px;
  }
</style>
```

3:49 LF÷ UTF-8÷ Git: master ÷

Passing Data to Child Components (with **Props**)

We’ve looked at using (and reusing) components in a Vue app and the potential benefits that offers to the developer but the problem is, components won’t really be useful unless you can pass data to it, such as a ‘title’ and ‘content’ for example, of a specific object, (say a ‘post’) we want to display. That’s where **props** come in.

Props are custom attributes you can register on a component. When a value is passed to a prop attribute, it becomes a property on that component instance. To pass a ‘title’ to a ‘blog post’ component, we can include it in the list of props this component accepts, using a **props** option:

```
Vue.component('blog-post', {
  props: ['title'],
  template: '<h3>{{ title }}</h3>'
})
```

Passing Data to Child Components (with **Props**)

A component can have as many **props** as you'd like and by default, any value can be passed to any prop. In the template previous, you'll see that we can access this value on the component instance, just like with **data**.

Once a prop is registered, you can pass data to it as a custom attribute, like this:

```
<blog-post title="My journey with Vue"></blog-post>
<blog-post title="Blogging with Vue"></blog-post>
<blog-post title="Why Vue is so fun"></blog-post>
```

Passing Data to Child Components (with Props)

- Prop Casing (camelCase vs kebab-case)

HTML attribute names are case-insensitive, so browsers will interpret any uppercase characters as lowercase. That means when you're using in-DOM templates, camelCased prop names need to use their kebab-cased (hyphen-delimited) equivalents:

```
Vue.component('blog-post', {
  // camelCase in JavaScript
  props: ['postTitle'],
  template: '<h3>{{ postTitle }}</h3>'
})
```

```
<!-- kebab-case in HTML -->
<blog-post post-title="hello!"></blog-post>
```

Again, if you're using string templates, this limitation does not apply.

Passing Data to Child Components (with Props)

- Prop Types

So far, we've only seen props listed as an array of strings, for example:

```
props: ['title', 'likes', 'isPublished', 'commentIds', 'author']
```

Usually though, you'll want every prop to be a specific type of value. In these cases, you can list props as an object, where the properties' names and values contain the prop names and types, respectively:

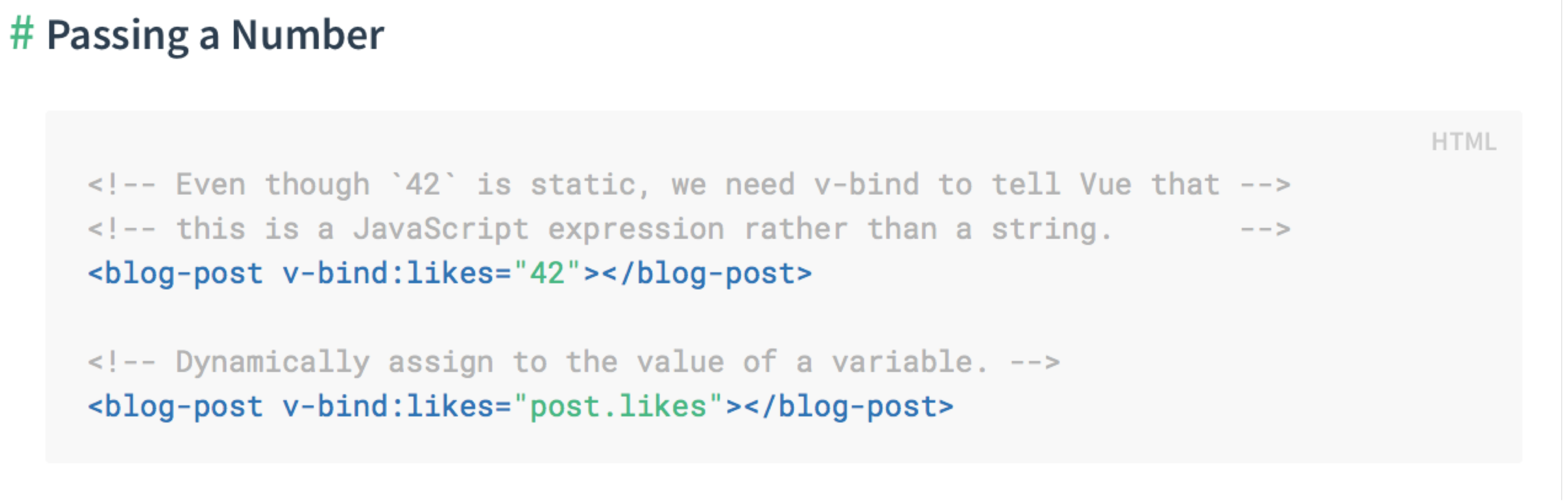
```
props: {
  title: String,
  likes: Number,
  isPublished: Boolean,
  commentIds: Array,
  author: Object
}
```

This not only documents your component, but will also warn users in the browser's JavaScript console if they pass the wrong type.

Passing Data to Child Components (with **Props**)

- Passing Static or Dynamic Props

So far we've just seen how to pass static props (Strings) but you can actually pass ***any*** type of value to **props**:



Passing a Number

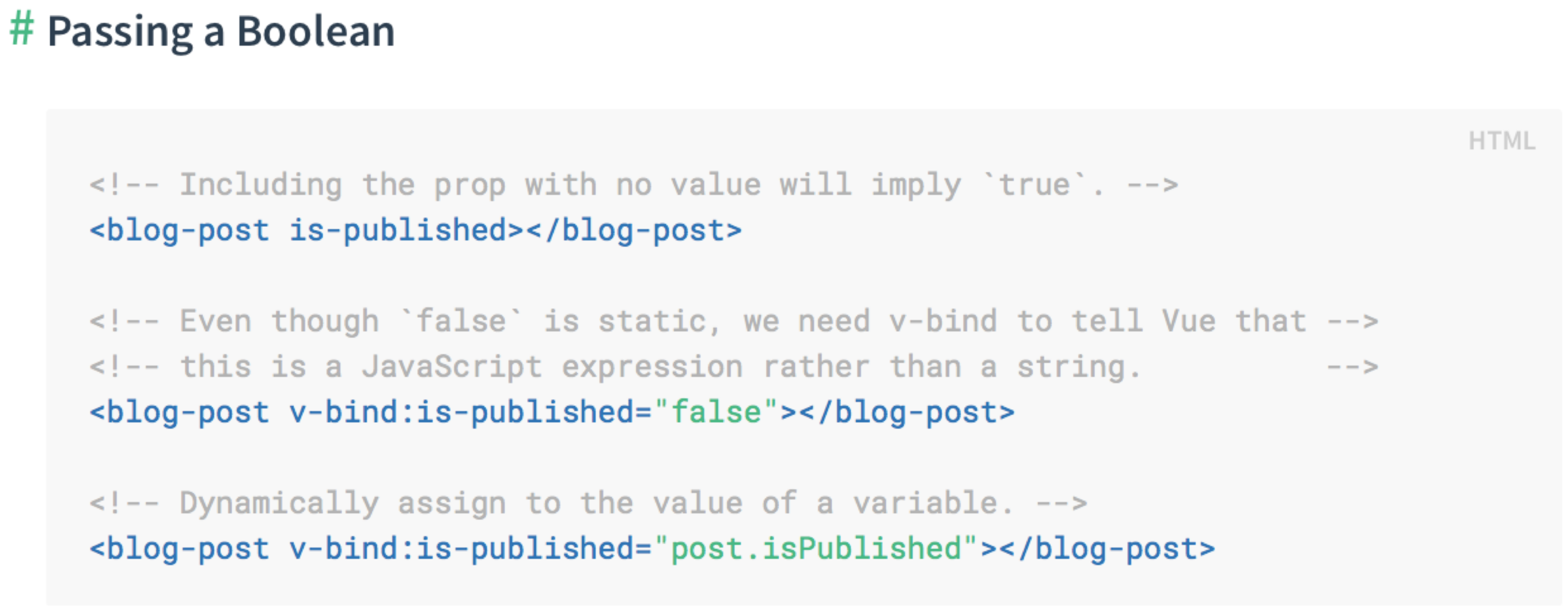
HTML

```html
<!-- Even though `42` is static, we need v-bind to tell Vue that -->
<!-- this is a JavaScript expression rather than a string.       -->
<blog-post v-bind:likes="42"></blog-post>

<!-- Dynamically assign to the value of a variable. -->
<blog-post v-bind:likes="post.likes"></blog-post>
```

Passing Data to Child Components (with Props)

- Passing Static or Dynamic Props

So far we’ve just seen how to pass static props (Strings) but you can actually pass ***any*** type of value to **props**:



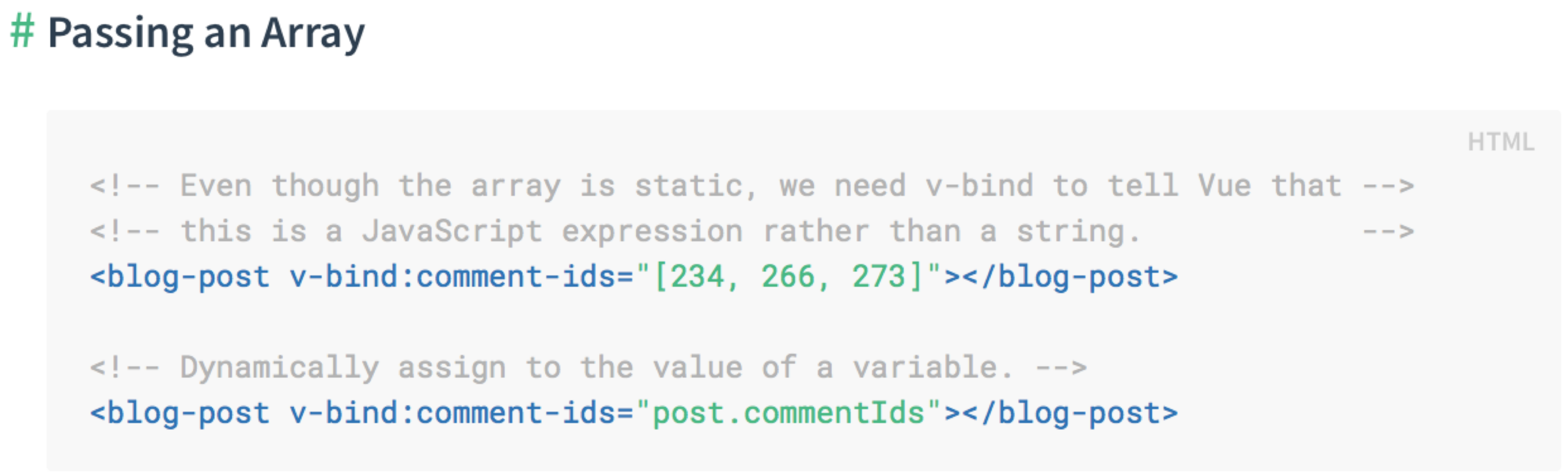
Passing a Boolean

```html
<!-- Including the prop with no value will imply `true`. -->
<blog-post is-published></blog-post>

<!-- Even though `false` is static, we need v-bind to tell Vue that -->
<!-- this is a JavaScript expression rather than a string.          -->
<blog-post v-bind:is-published="false"></blog-post>

<!-- Dynamically assign to the value of a variable. -->
<blog-post v-bind:is-published="post.isPublished"></blog-post>
```

Passing Data to Child Components (with Props)

- Passing Static or Dynamic Props

So far we've just seen how to pass static props (Strings) but you can actually pass ***any*** type of value to **props**:



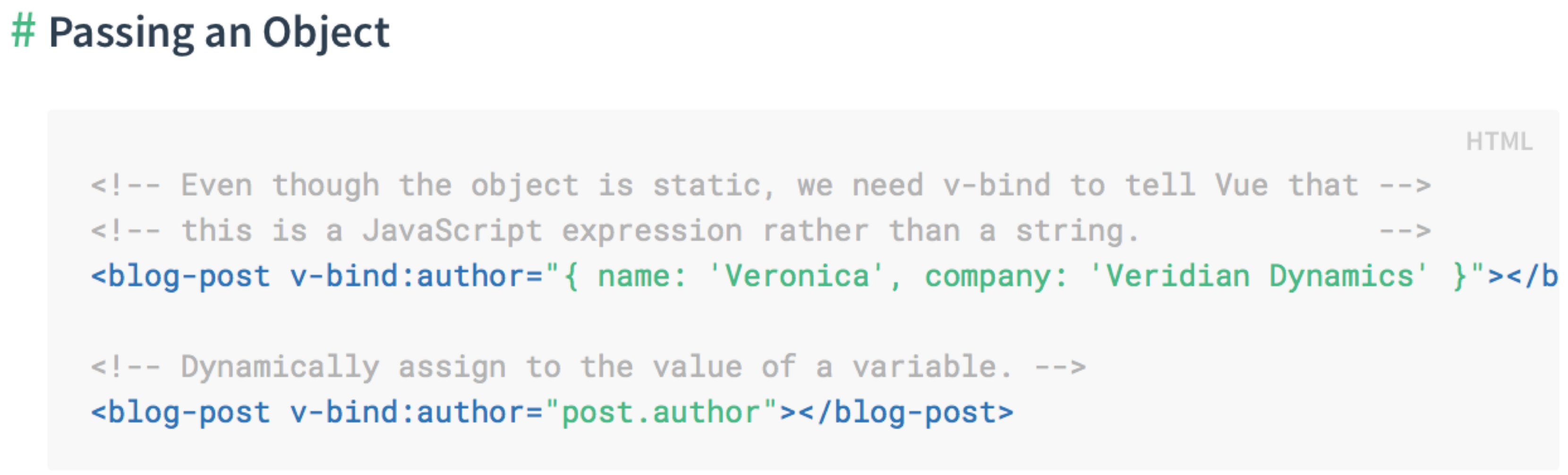
Passing an Array

```html
<!-- Even though the array is static, we need v-bind to tell Vue that -->
<!-- this is a JavaScript expression rather than a string.            -->
<blog-post v-bind:comment-ids="[234, 266, 273]"></blog-post>

<!-- Dynamically assign to the value of a variable. -->
<blog-post v-bind:comment-ids="post.commentIds"></blog-post>
```

Passing Data to Child Components (with Props)

- Passing Static or Dynamic Props

So far we've just seen how to pass static props (Strings) but you can actually pass ***any*** type of value to **props**:



Passing an Object

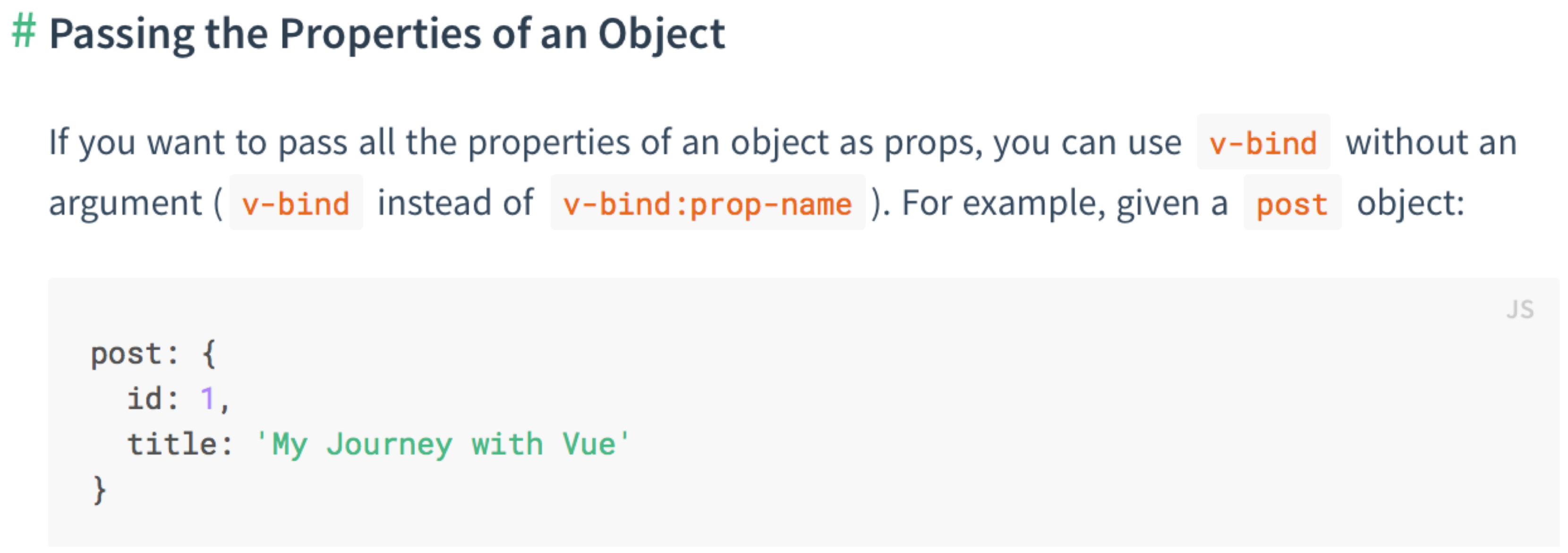
HTML

```html
<!-- Even though the object is static, we need v-bind to tell Vue that -->
<!-- this is a JavaScript expression rather than a string.             -->
<blog-post v-bind:author="{ name: 'Veronica', company: 'Veridian Dynamics' }"></b

<!-- Dynamically assign to the value of a variable. -->
<blog-post v-bind:author="post.author"></blog-post>
```

Passing Data to Child Components (with **Props**)

- Passing Static or Dynamic Props

So far we've just seen how to pass static props (Strings) but you can actually pass ***any*** type of value to **props**:



Passing the Properties of an Object

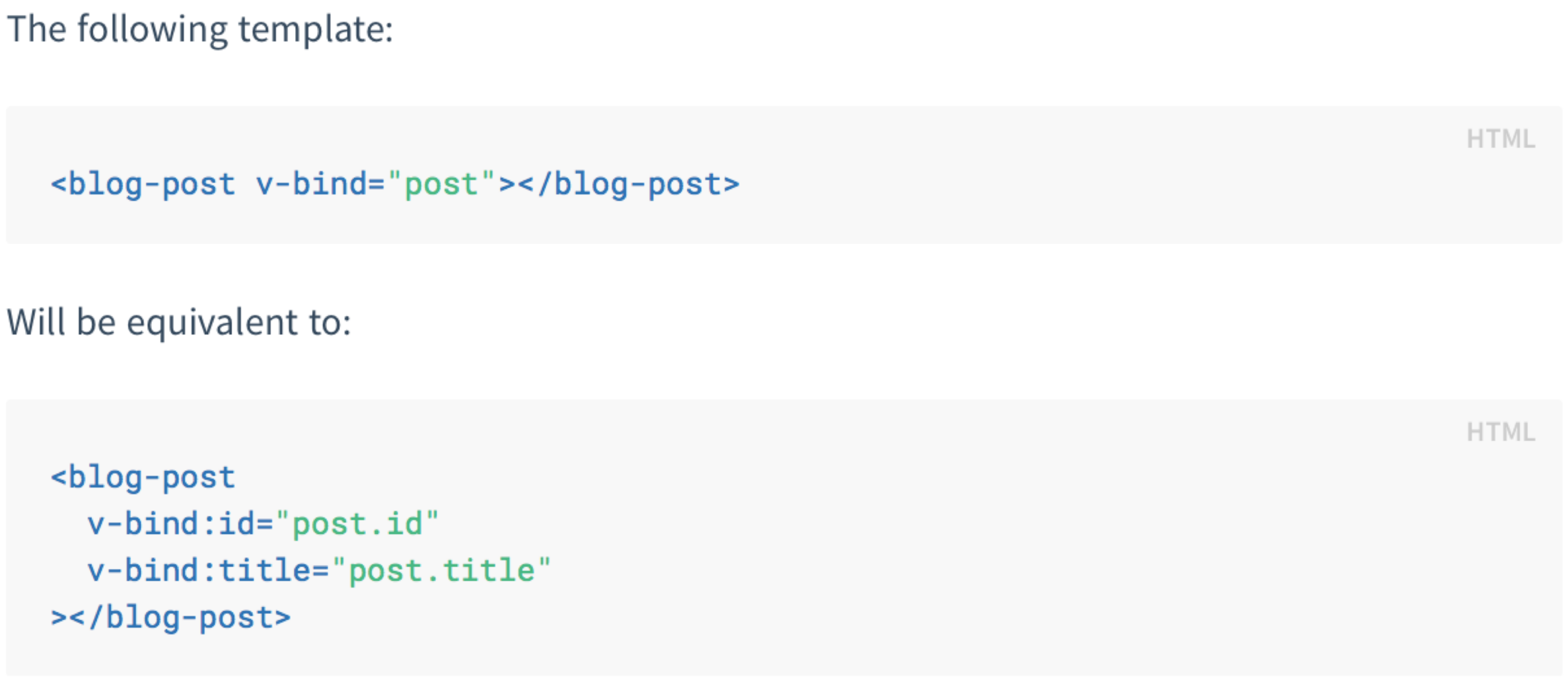
If you want to pass all the properties of an object as props, you can use `v-bind` without an argument (`v-bind` instead of `v-bind:prop-name`). For example, given a `post` object:

```
JS
post: {
  id: 1,
  title: 'My Journey with Vue'
}
```

Passing Data to Child Components (with **Props**)

- Passing Static or Dynamic Props

So far we've just seen how to pass static props (Strings) but you can actually pass ***any*** type of value to **props**:



The following template:

```html
<blog-post v-bind="post"></blog-post>
```

Will be equivalent to:

```html
<blog-post
  v-bind:id="post.id"
  v-bind:title="post.title"
></blog-post>
```

Passing Data to Child Components (with Props)

- One-Way Data Flow

All props form a **one-way-down** binding between the child property and the parent one: when the parent property updates, it will flow down to the child, but **not** the other way around. This prevents child components from accidentally mutating the parent's state, which can make your app's data flow harder to understand.

In addition, every time the parent component is updated, all props in the child component will be refreshed with the latest value. This means you should **<u>not</u>** attempt to mutate a prop inside a child component. If you do, Vue will warn you in the console.

There are usually two cases where it's tempting to mutate a prop:

Passing Data to Child Components (with Props)

- One-Way Data Flow

1. **The prop is used to pass in an initial value; the child component wants to use it as a local data property afterwards.** In this case, it’s best to define a local data property that uses the prop as its initial value:

```js
props: ['initialCounter'],
data: function () {
  return {
    counter: this.initialCounter
  }
}
```

Passing Data to Child Components (with Props)

- One-Way Data Flow

2. **The prop is passed in as a raw value that needs to be transformed.** In this case, it's best to define a computed property using the prop's value:

```
JS

props: ['size'],
computed: {
  normalizedSize: function () {
    return this.size.trim().toLowerCase()
  }
}
```

Component Content Distribution & Slots

Vue implements a content distribution API that's modelled after the current **Web Components spec draft**, using the <slot> element to serve as distribution outlets for content.

This allows you to compose components like this:

```
<navigation-link url="/profile">
  Your Profile
</navigation-link>
```

Then in the template for **<navigation-link>**, you might have:

```
<a
  v-bind:href="url"
  class="nav-link"
>
  <slot></slot>
</a>
```

When the component renders, the **<slot>** element will be replaced by "Your Profile".

Component Content Distribution & Slots

Slots can contain any template code, including HTML:

```
<navigation-link url="/profile">
  <!-- Add a Font Awesome icon -->
  <span class="fa fa-user"></span>
  Your Profile
</navigation-link>
```

Or even other components:

```
<navigation-link url="/profile">
  <!-- Use a component to add an icon -->
  <font-awesome-icon name="user"></font-awesome-icon>
  Your Profile
</navigation-link>
```

If **<navigation-link>** did **not** contain a **<slot>** element, any content passed to it would simply be discarded.

Named Slots

There are times when it’s useful to have multiple slots. For example, in a hypothetical **base-layout** component with the following template:

```
<div class="container">
  <header>
    <!-- We want header content here -->
  </header>
  <main>
    <!-- We want main content here -->
  </main>
  <footer>
    <!-- We want footer content here -->
  </footer>
</div>
```

Named Slots

For these cases, the <slot> element has a special attribute, **name**, which can be used to define additional slots:

```
<div class="container">
  <header>
    <slot name="header"></slot>
  </header>
  <main>
    <slot></slot>
  </main>
  <footer>
    <slot name="footer"></slot>
  </footer>
</div>
```

Named Slots

To provide content to named slots, we can use the slot attribute on a <template> element in the parent:

```
<base-layout>
  <template slot="header">
    <h1>Here might be a page title</h1>
  </template>

  <p>A paragraph for the main content.</p>
  <p>And another one.</p>

  <template slot="footer">
    <p>Here's some contact info</p>
  </template>
</base-layout>
```

Named Slots

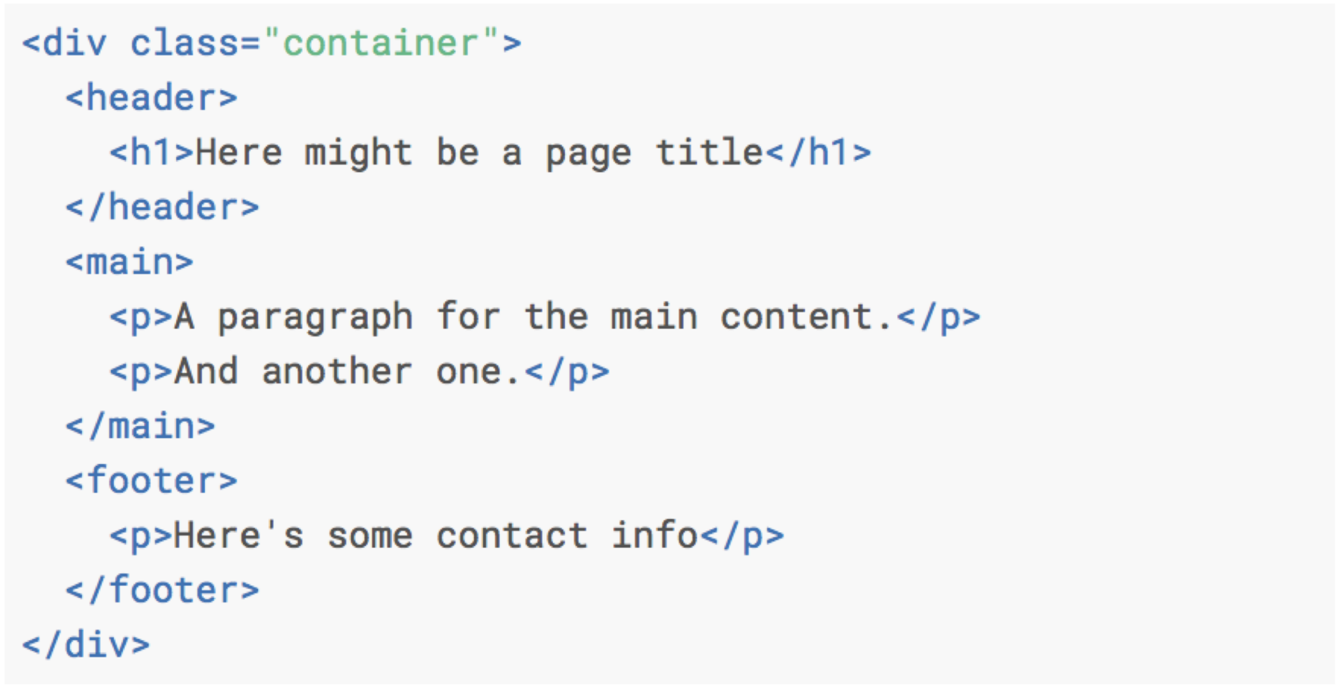
Or, the slot attribute can also be used directly on a normal element:

```
<base-layout>
  <h1 slot="header">Here might be a page title</h1>

  <p>A paragraph for the main content.</p>
  <p>And another one.</p>

  <p slot="footer">Here's some contact info</p>
</base-layout>
```

There can still be one unnamed slot, which is the **default slot** that serves as a catch-all outlet for any unmatched content. In both examples, the rendered HTML would be:

Named Slots

In both previous examples, the rendered HTML would be:



```html
<div class="container">
  <header>
    <h1>Here might be a page title</h1>
  </header>
  <main>
    <p>A paragraph for the main content.</p>
    <p>And another one.</p>
  </main>
  <footer>
    <p>Here's some contact info</p>
  </footer>
</div>
```

Scoped Slots

Sometimes you'll want to provide a component with a reusable slot that can access data from the child component. For example, a simple **<todo-list>** component may contain the following in its template:

```
<ul>
  <li
    v-for="todo in todos"
    v-bind:key="todo.id"
  >
    {{ todo.text }}
  </li>
</ul>
```

But in some parts of our app, we want the individual todo items to render something different than just the todo.text. This is where **scoped slots** come in.

Scoped Slots

To make the feature possible, all we have to do is wrap the todo item content in a <slot> element, then pass the slot any data relevant to its context: in this case, the **todo** object:

```
<ul>
  <li
    v-for="todo in todos"
    v-bind:key="todo.id"
  >
    <!-- We have a slot for each todo, passing it the -->
    <!-- `todo` object as a slot prop.                -->
    <slot v-bind:todo="todo">
      <!-- Fallback content -->
      {{ todo.text }}
    </slot>
  </li>
</ul>
```

Scoped **Slots**

Now when we use the **<todo-list>** component, we can optionally define an alternative **<template>** for todo items, but with access to data from the child via the **slot-scope** attribute:

```
<todo-list v-bind:todos="todos">
  <!-- Define `slotProps` as the name of our slot scope -->
  <template slot-scope="slotProps">
    <!-- Define a custom template for todo items, using -->
    <!-- `slotProps` to customize each todo.            -->
    <span v-if="slotProps.todo.isComplete">✓</span>
    {{ slotProps.todo.text }}
  </template>
</todo-list>
```

Key Point about Slots

Think of slots as passing **components as props** to a child component. Similar to how we pass strings, integers and objects, you're passing an entire sub-DOM tree which the child can use in any place that it needs.

A few other places you can use slots:

- If you're building stuff on material design. For example, the "Cards" material design system.
- Modal windows and dialogs in general
- Things like Bootstrap Panels and Custom Content

A few more things about using slots:

- A child component can be used for styling/presentation and the business logic can be kept in the parent element.
- If you don't pass anything to a slot, nothing is shown. This lets you re-use them quite nicely and only pass in the slots that you want.

Extra Reading & Info

https://vuejs.org/v2/guide/components-props.html

https://vuejs.org/v2/guide/components-slots.html

Case Study

LABS IN ACTION

Analysing our Case Study

So now that we've covered some more detail about Components and passing data to Child Components via **props** and using **slots**, let's take a closer look at how this is implemented in **DonationVue**.

We'll have a brief look at **`Donate.vue`** and **`DonationForm.vue`** with respect to how we can include Custom Components and pass data from Parent to Child to allow for easier maintainability and reusability.

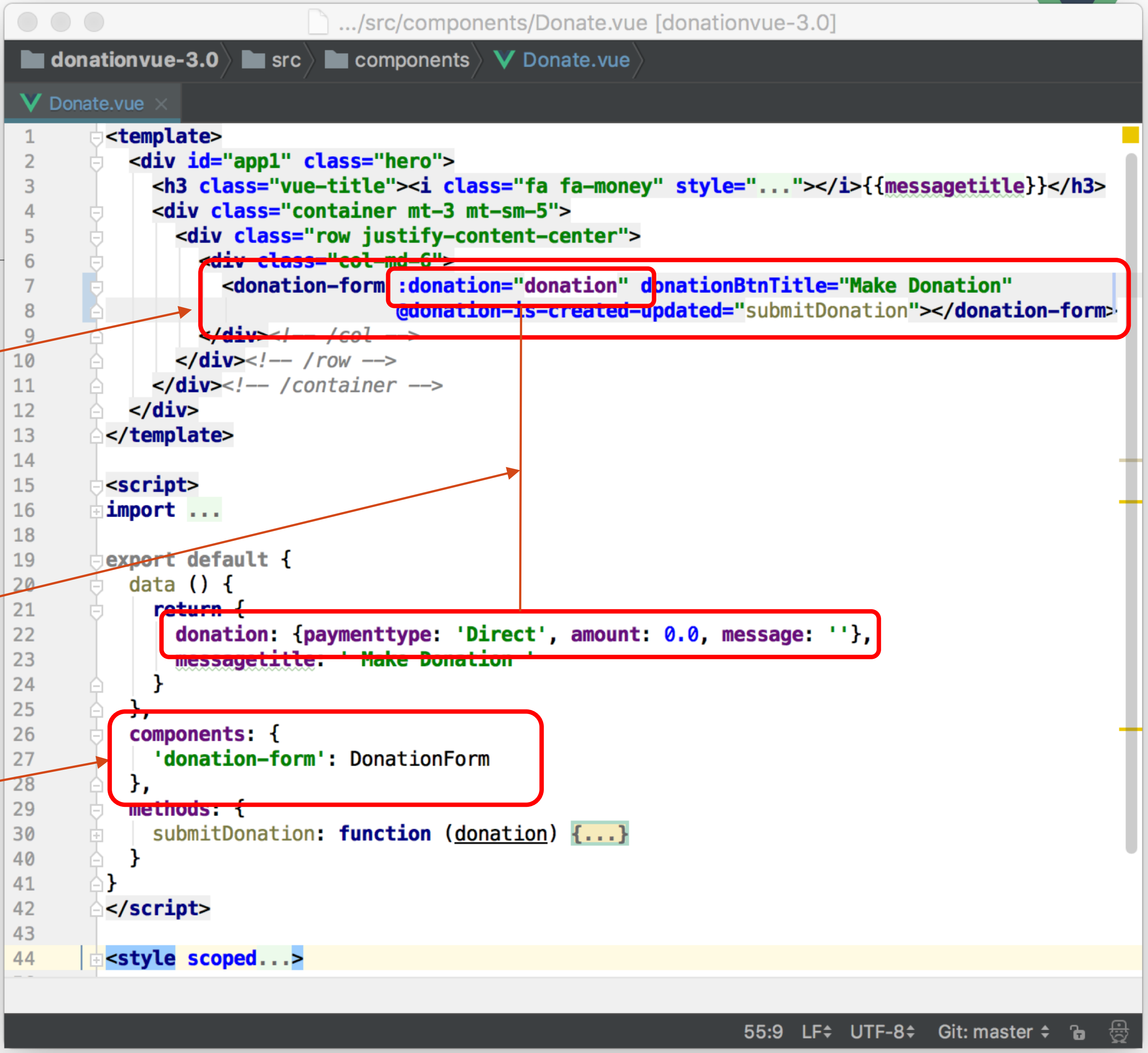
We'll also have a look at **`Donations.vue`** and how we make use of **scoped slots** to allow us to edit and delete individual donations.

Donate.vue

Child Component

Data Binding (via **props**)

Component Registration


.../src/components/Donate.vue [donationvue-3.0]
donationvue-3.0
src
components
Donate.vue
Donate.vue
55:9 LF÷ UTF-8÷ Git: master ÷

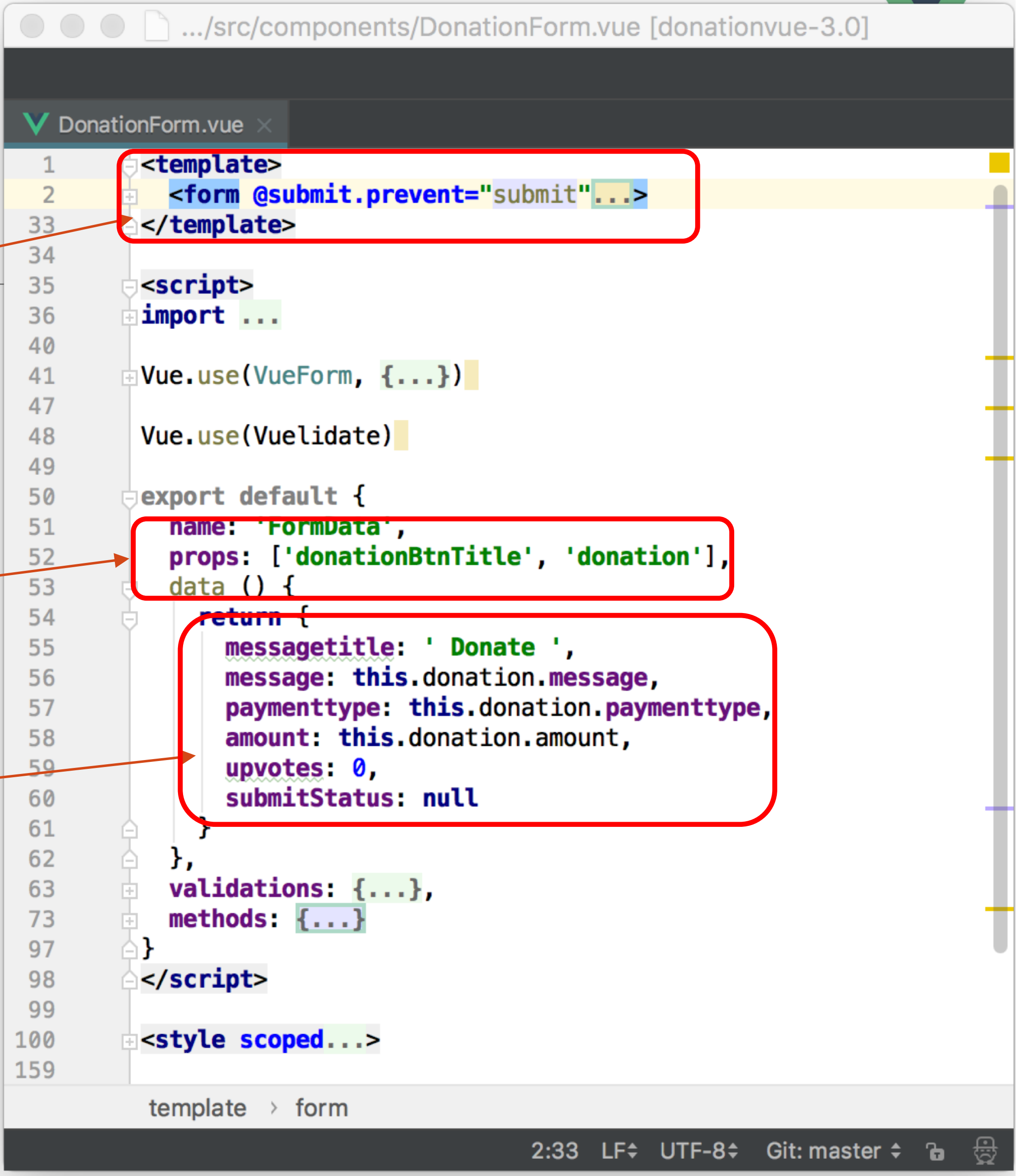
```
<template>
  <div id="app1" class="hero">
    <h3 class="vue-title"><i class="fa fa-money" style="..."></i>{{messagetitle}}</h3>
    <div class="container mt-3 mt-sm-5">
      <div class="row justify-content-center">
        <div class="col-md-6">
          <donation-form :donation="donation" donationBtnTitle="Make Donation"
                         @donation-is-created-updated="submitDonation"></donation-form>
        </div><!-- /col -->
      </div><!-- /row -->
    </div><!-- /container -->
  </div>
</template>

<script>
import ...

export default {
  data () {
    return {
      donation: {paymenttype: 'Direct', amount: 0.0, message: ''},
      messagetitle: ' Make Donation '
    }
  },
  components: {
    'donation-form': DonationForm
  },
  methods: {
    submitDonation: function (donation) {...}
  }
}
</script>

<style scoped...>
```

DonationForm.vue

Component Template

Component Props

Component Data



.../src/components/DonationForm.vue [donationvue-3.0]

DonationForm.vue

```
<template>
  <form @submit.prevent="submit"...>
</template>

<script>
import ...

Vue.use(VueForm, {...})

Vue.use(Vuelidate)

export default {
  name: 'FormData',
  props: ['donationBtnTitle', 'donation'],
  data () {
    return {
      messagetitle: ' Donate ',
      message: this.donation.message,
      paymenttype: this.donation.paymenttype,
      amount: this.donation.amount,
      upvotes: 0,
      submitStatus: null
    }
  },
  validations: {...},
  methods: {...}
}
</script>

<style scoped...>
```

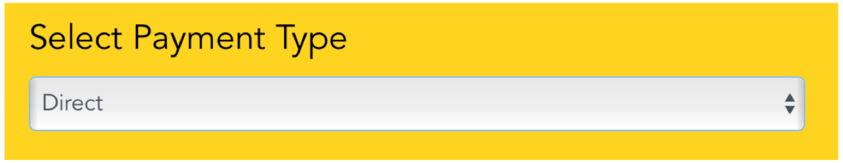
template › form

2:33 LF UTF-8 Git: master

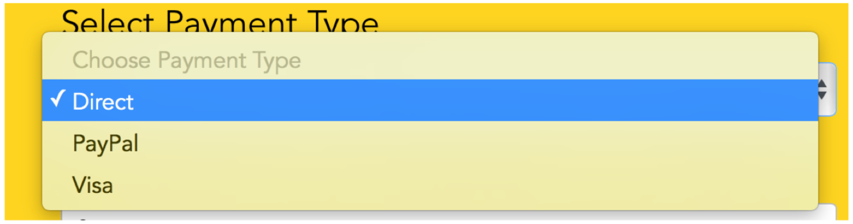
Populated from **props** (because we want to change this **locally**)

DonationForm.vue

```
<label class="form-label">Select Payment Type</label>
<select id="paymenttype" name="paymenttype" class="form-control" type="text" v-model="paymenttype">
  <option value="null" selected disabled hidden>Choose Payment Type</option>
  <option value="Direct">Direct</option>
  <option value="PayPal">PayPal</option>
  <option value="Visa">Visa</option>
</select>
```


Select Payment Type

Direct


Select Payment Type

Choose Payment Type

✓ Direct

PayPal

Visa

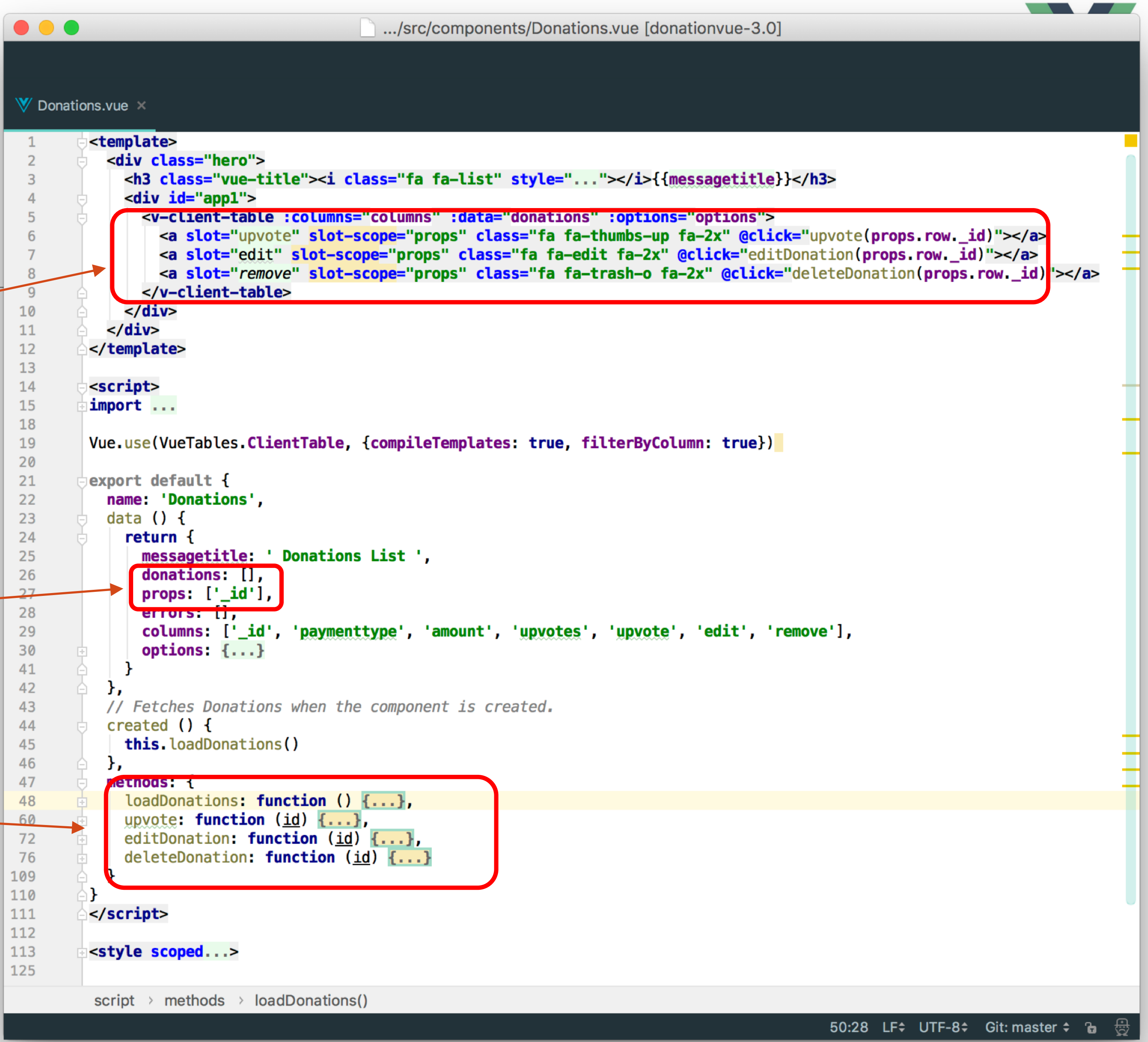
Part of our Input Form

Donations.vue

Named, Scoped Slots

Component Props

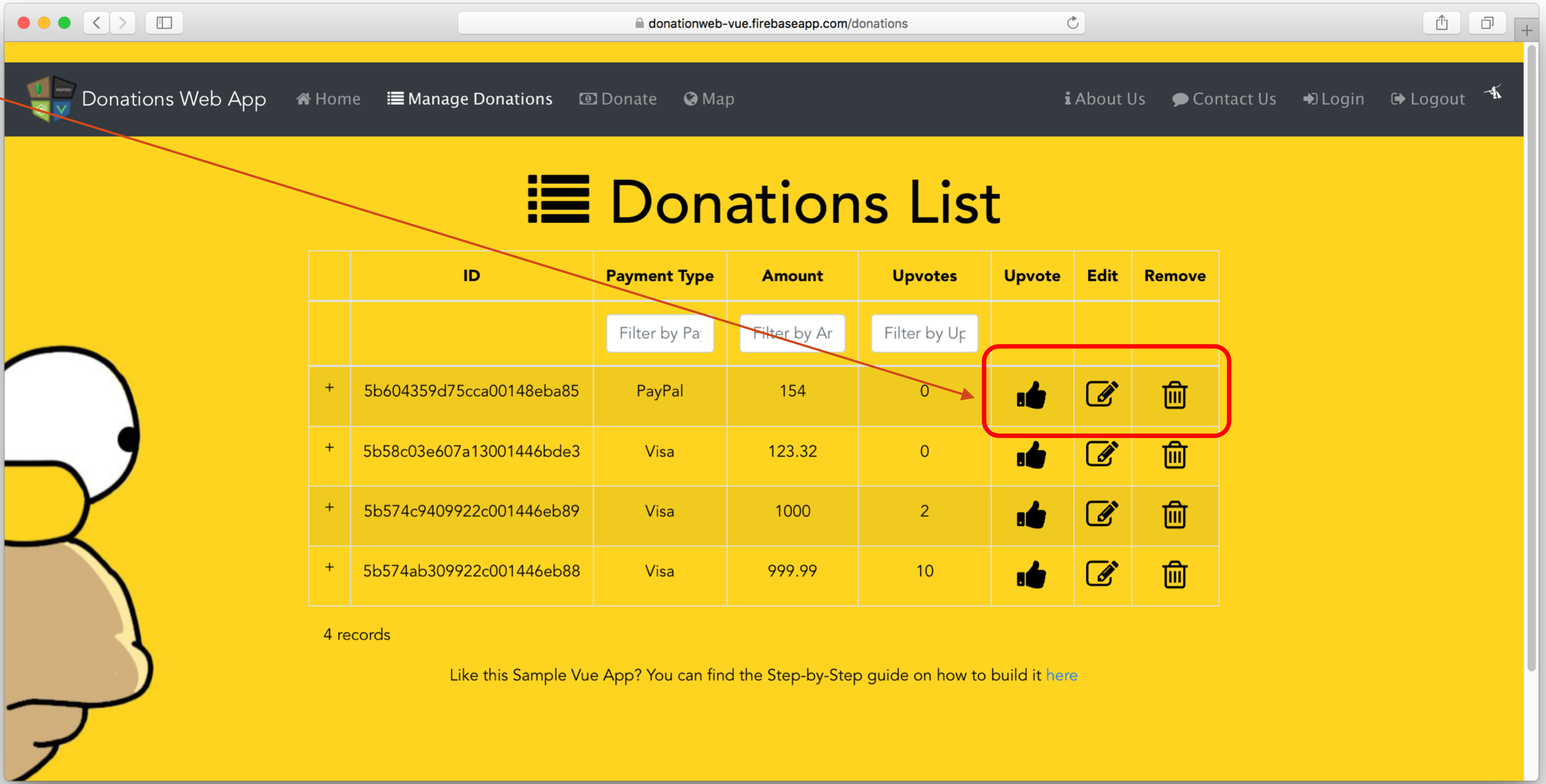
Component methods


.../src/components/Donations.vue [donationvue-3.0]
Donations.vue
script › methods › loadDonations()
50:28 LF UTF-8 Git: master

```
<template>
  <div class="hero">
    <h3 class="vue-title"><i class="fa fa-list" style="..."></i>{{messagetitle}}</h3>
    <div id="app1">
      <v-client-table :columns="columns" :data="donations" :options="options">
        <a slot="upvote" slot-scope="props" class="fa fa-thumbs-up fa-2x" @click="upvote(props.row._id)"></a>
        <a slot="edit" slot-scope="props" class="fa fa-edit fa-2x" @click="editDonation(props.row._id)"></a>
        <a slot="remove" slot-scope="props" class="fa fa-trash-o fa-2x" @click="deleteDonation(props.row._id)"></a>
      </v-client-table>
    </div>
  </div>
</template>

<script>
import ...

Vue.use(VueTables.ClientTable, {compileTemplates: true, filterByColumn: true})

export default {
  name: 'Donations',
  data () {
    return {
      messagetitle: ' Donations List ',
      donations: [],
      props: ['_id'],
      errors: [],
      columns: ['_id', 'paymenttype', 'amount', 'upvotes', 'upvote', 'edit', 'remove'],
      options: {...}
    }
  },
  // Fetches Donations when the component is created.
  created () {
    this.loadDonations()
  },
  methods: {
    loadDonations: function () {...},
    upvote: function (id) {...},
    editDonation: function (id) {...},
    deleteDonation: function (id) {...}
  }
}
</script>

<style scoped...>
```

Donations.vue

Named, Scoped Slots



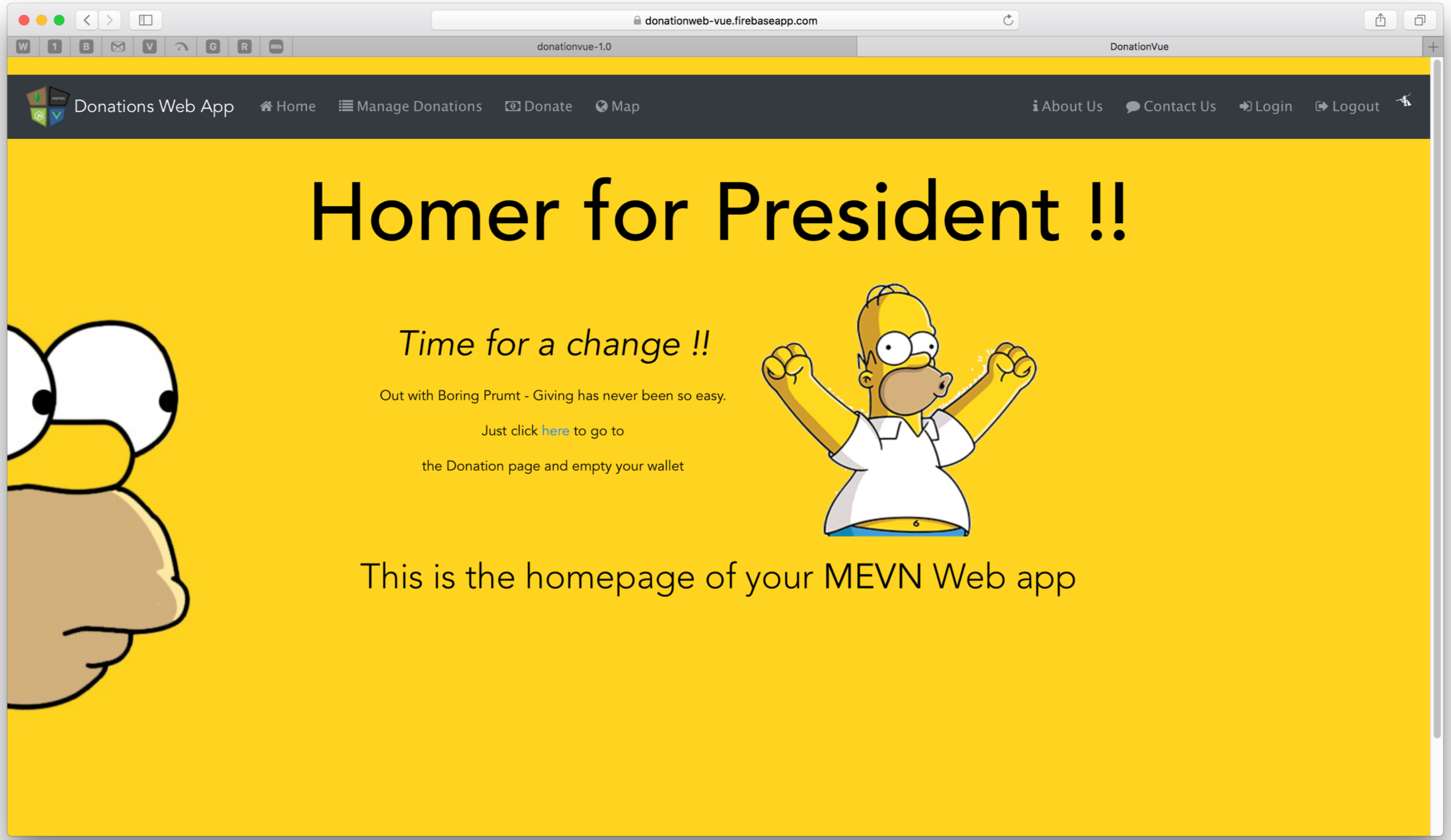
Donations List

	ID	Payment Type	Amount	Upvotes	Upvote	Edit	Remove
		Filter by Pa	Filter by Ar	Filter by Up			
+	5b604359d75cca00148eba85	PayPal	154	0	👍	✎	🗑
+	5b58c03e607a13001446bde3	Visa	123.32	0	👍	✎	🗑
+	5b574c9409922c001446eb89	Visa	1000	2	👍	✎	🗑
+	5b574ab309922c001446eb88	Visa	999.99	10	👍	✎	🗑

4 records

Like this Sample Vue App? You can find the Step-by-Step guide on how to build it here

Demo Application https://donationweb-vue.firebaseapp.com



Homer for President !!

Time for a change !!

Out with Boring Prumt - Giving has never been so easy.

Just click here to go to

the Donation page and empty your wallet

This is the homepage of your MEVN Web app

References

- https://vuejs.org
- https://skyronic.com/blog/vue-slots-example

Questions?