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 1

DECLARATIVE RENDERING & REACTIVITY

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

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**

Declarative Rendering & Reactivity

KEEPING TRACK OF CHANGES (DATA BINDING)

Introduction - Recap

As previously mentioned, at the core of Vue.js is a system that enables us to declaratively render data to the DOM using straightforward template syntax:

```html
<div id="app">
  {{ message }}
</div>
```

“Mustache” syntax

Hello Vue!

Result in Browser

```javascript
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
})
```

This looks pretty similar to rendering a string template, but Vue has done a lot of work under the hood. The data and the DOM are now linked, and everything is now **reactive**.

Vue.js

Introduction - Recap

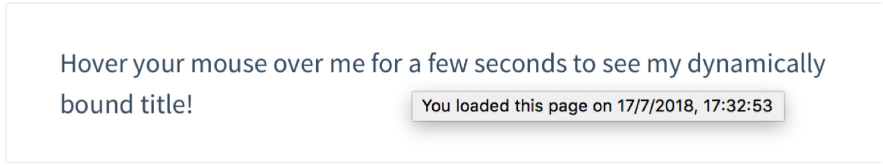
In addition to **text interpolation**, we can also bind element attributes like this:

Directive

```
<div id="app-2">
  <span v-bind:title="message">
    Hover your mouse over me for a few seconds
    to see my dynamically bound title!
  </span>
</div>
```

```
var app2 = new Vue({
  el: '#app-2',
  data: {
    message: 'You loaded this page on ' + new Date().toLocaleString()
  }
})
```

Result in Browser


Hover your mouse over me for a few seconds to see my dynamically bound title!

You loaded this page on 17/7/2018, 17:32:53

Introduction - Recap

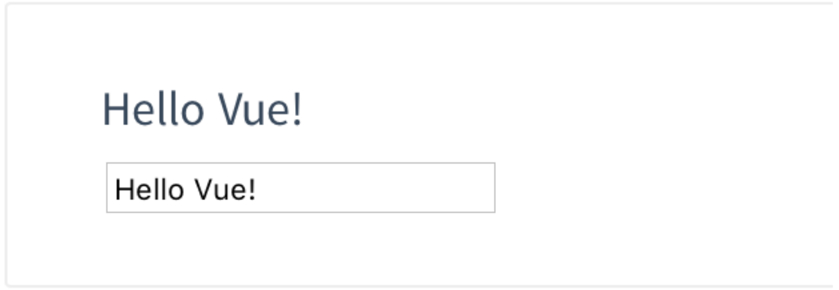
And **two-way reactive data binding** like this:

```
<div id="app-6">
  <p>{{ message }}</p>
  <input v-model="message">
</div>
```

Directive

```
var app6 = new Vue({
  el: '#app-6',
  data: {
    message: 'Hello Vue!'
  }
})
```

Result in Browser


Hello Vue!

Hello Vue!

Reactivity in Depth

One of Vue’s most distinct features is the unobtrusive reactivity system.

Models are just plain JavaScript objects. When you modify them, the view updates. It makes state management simple and intuitive, but it’s also important to understand how it works to avoid some common gotchas.

Here, we are going to look at some of the lower-level details of Vue’s reactivity system.

How Changes are Tracked

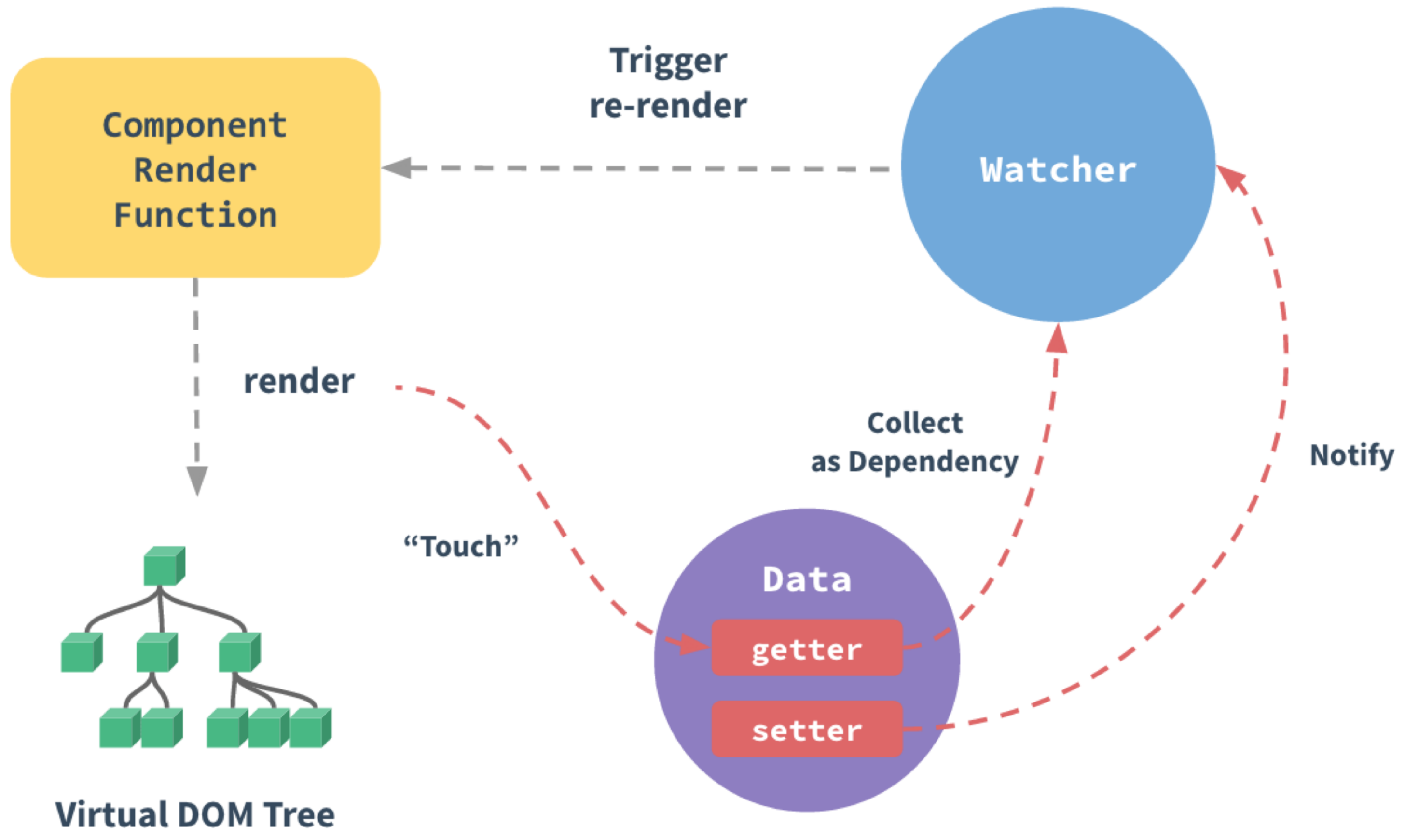
When you pass a plain JavaScript object to a Vue instance as its **data** option, Vue will walk through all of its properties and convert them to **getter/setters** using **Object.defineProperty**. This is an ES5-only and un-shimmable feature, which is why Vue doesn’t support IE8 and below.

The getter/setters are **invisible** to the user, but under the hood they enable Vue to perform dependency-tracking and change-notification when properties are accessed or modified. One caveat is that browser consoles format getter/setters differently when converted data objects are logged, so you may want to install **vue-devtools** for a more inspection-friendly interface.

Every component instance has a corresponding **watcher** instance, which records any properties “touched” during the component’s render as dependencies. Later on when a dependency’s setter is triggered, it notifies the watcher, which in turn causes the component to re-render.

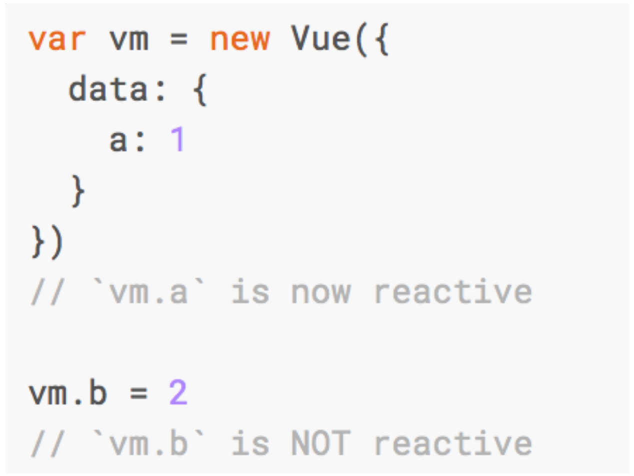
Vue.js

How Changes are Tracked



Change Detection Caveats

Due to the limitations of modern JavaScript (and the abandonment of Object.observe), Vue **cannot detect property addition or deletion**. Since Vue performs the getter/setter conversion process during instance initialization, a property must be present in the data object in order for Vue to convert it and make it reactive. For example:

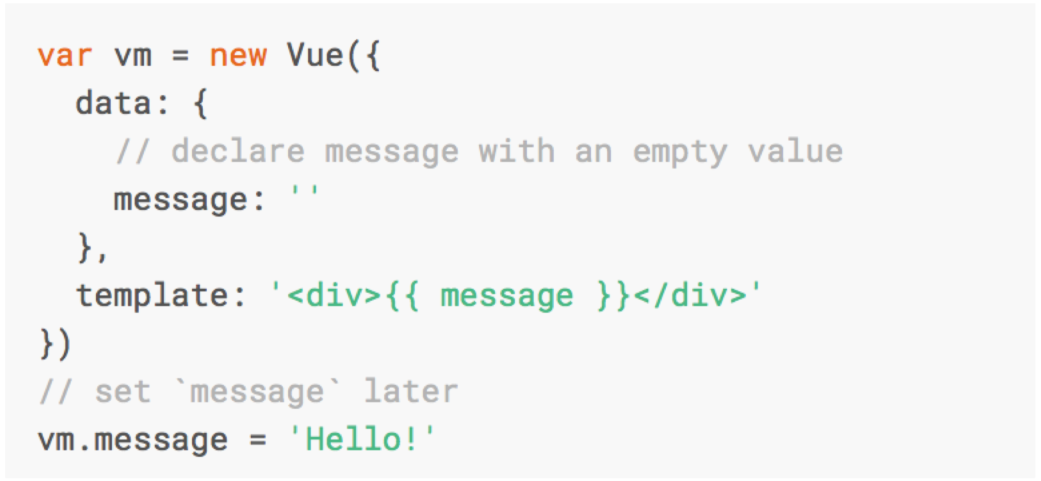


```
var vm = new Vue({
  data: {
    a: 1
  }
})
// `vm.a` is now reactive

vm.b = 2
// `vm.b` is NOT reactive
```

Vue does not allow dynamically adding new root-level reactive properties to an already created instance.

Declaring Reactive Properties

Since Vue doesn’t allow dynamically adding root-level reactive properties, you have to initialize Vue instances by declaring all root-level reactive data properties upfront, even with an empty value:



```
var vm = new Vue({
  data: {
    // declare message with an empty value
    message: ''
  },
  template: '<div>{{ message }}</div>'
})
// set `message` later
vm.message = 'Hello!'
```

If you don’t declare **message** in the data option, Vue will warn you that the render function is trying to access a property that doesn’t exist.

Declaring Reactive Properties

There are technical reasons behind this restriction;

It eliminates a class of edge cases in the dependency tracking system, and also makes Vue instances play nicer with type checking systems.

But there is also an important consideration in terms of code maintainability: the **data** object is like the **schema** for your component's state.

Declaring all reactive properties upfront makes the component code easier to understand when revisited later or read by another developer.

Template Syntax

Vue.js uses an HTML-based template syntax that allows you to declaratively bind the rendered DOM to the underlying Vue instance’s data.

All Vue.js templates are valid HTML that can be parsed by spec-compliant browsers and HTML parsers.

Under the hood, Vue compiles the templates into Virtual DOM render functions. Combined with the reactivity system,

Vue is able to intelligently figure out the minimal number of components to re-render and apply the minimal amount of DOM manipulations when the app state changes.

Case Study

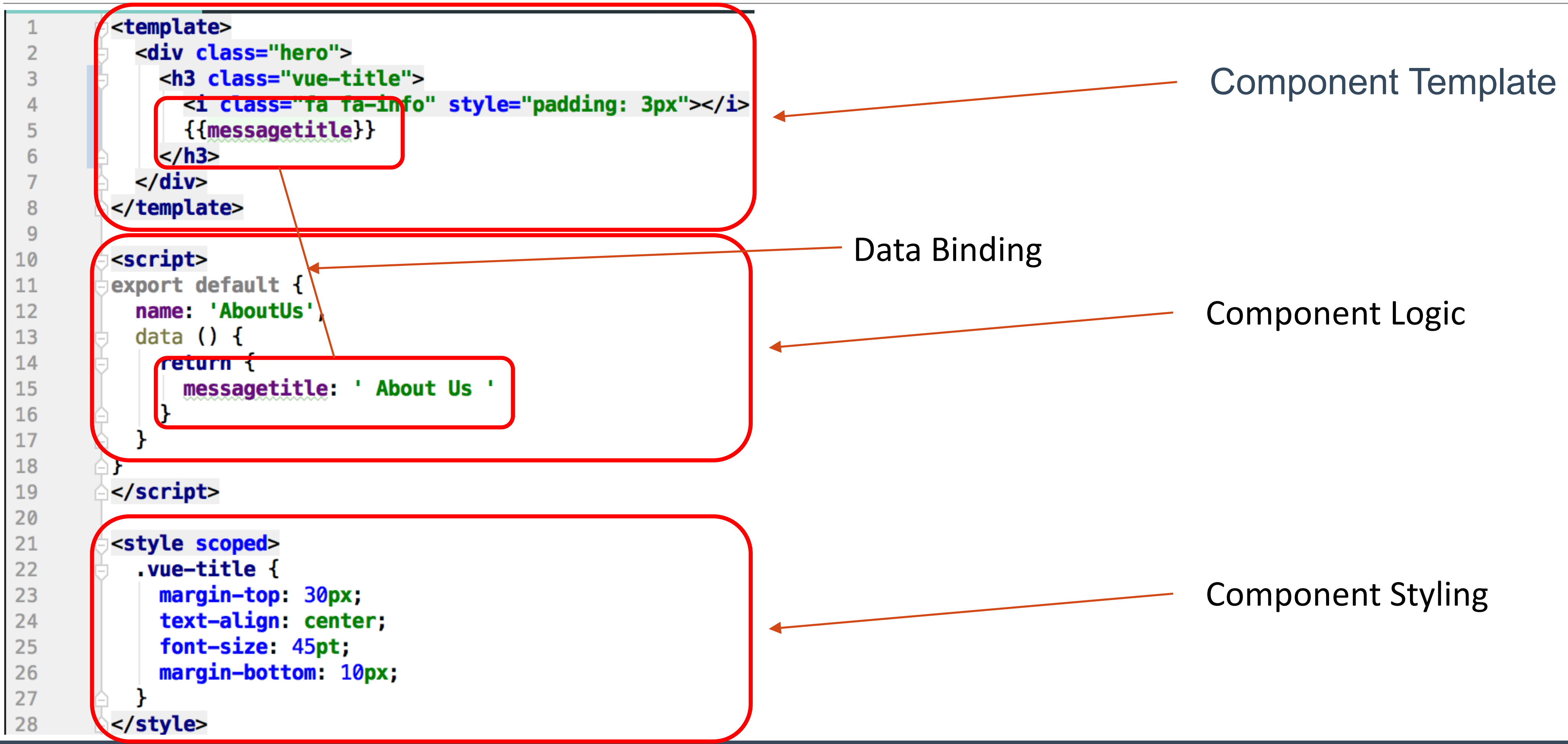
LABS IN ACTION

Analysing our Case Study

So now that we’ve covered some more detail about Declarative Rendering and Reactivity, let’s take a closer look at how this is implemented in **DonationVue**.

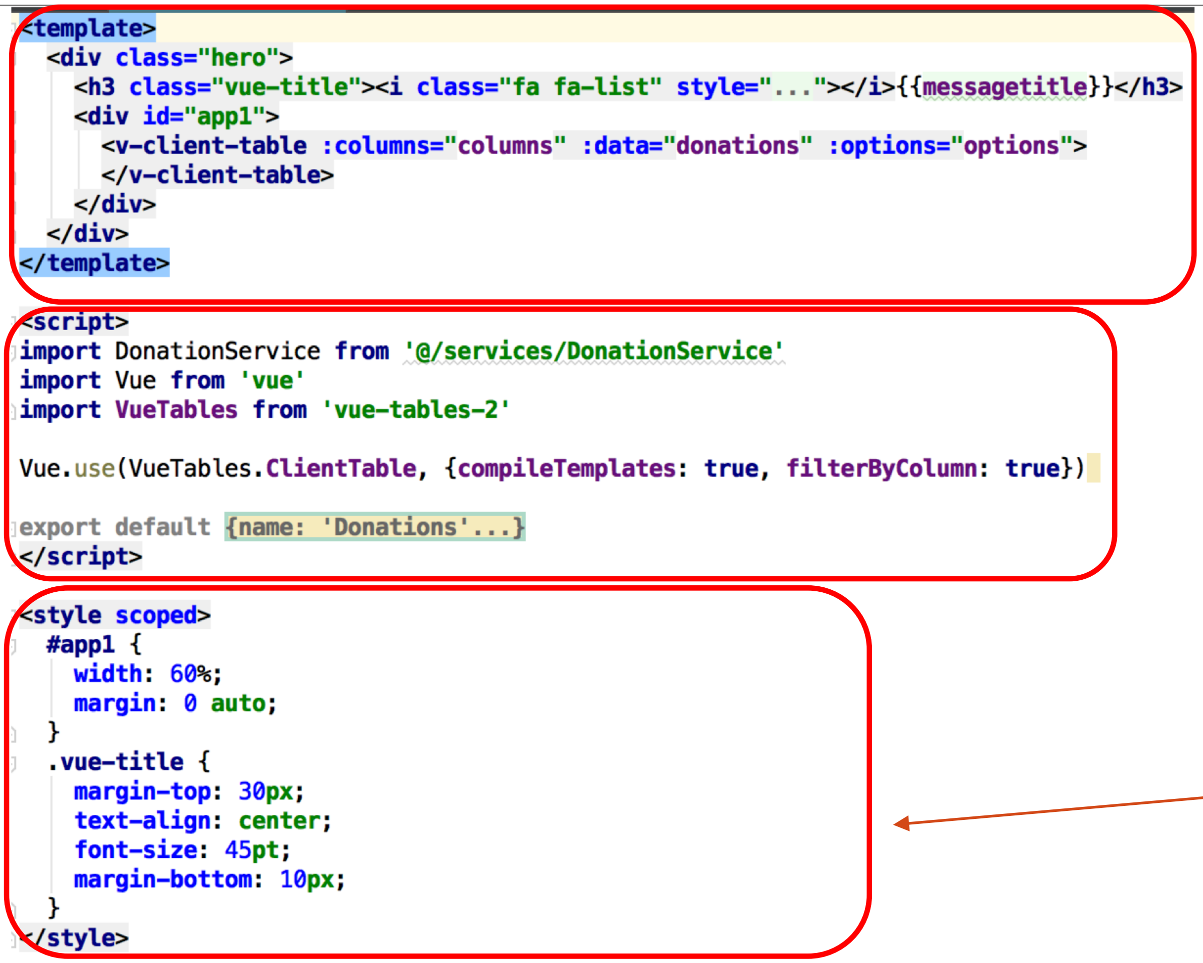
We’ll first have a look at some of the more basic components (**`AboutUs.vue`**, **`ContactUs.vue`** etc) and then some of the more advanced components (**`Donations.vue, Donate.vue`**) with respect to primarily **Declaring our Instance Variables** and make use of **Data Binding**, but we’ll also touch on **Reusable Components** and **Event Handling (kinda!)** (with a closer look in later sections).

AboutUs.vue



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

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

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

Donations.vue



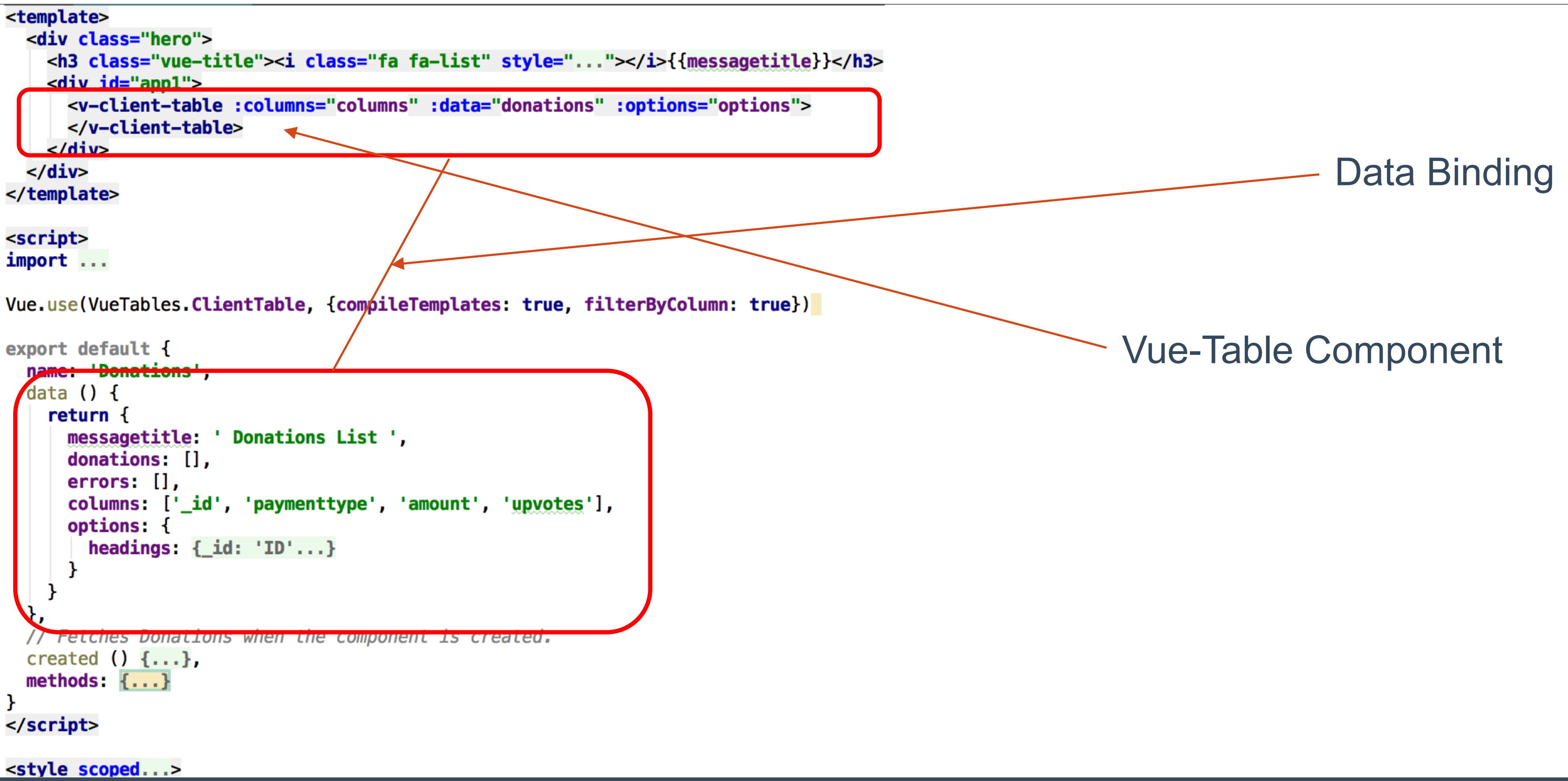
```
<template>
  <div class="hero">
    <h3 class="vue-title"><i class="fa fa-list" style="..."></i>{{messagetitle}}</h3>
    <div id="app1">
      <v-client-table :columns="columns" :data="donations" :options="options">
      </v-client-table>
    </div>
  </div>
</template>

<script>
import DonationService from '@/services/DonationService'
import Vue from 'vue'
import VueTables from 'vue-tables-2'

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

export default {name: 'Donations'...}
</script>

<style scoped>
  #app1 {
    width: 60%;
    margin: 0 auto;
  }
  .vue-title {
    margin-top: 30px;
    text-align: center;
    font-size: 45pt;
    margin-bottom: 10px;
  }
</style>
```

Component Template

Component Logic

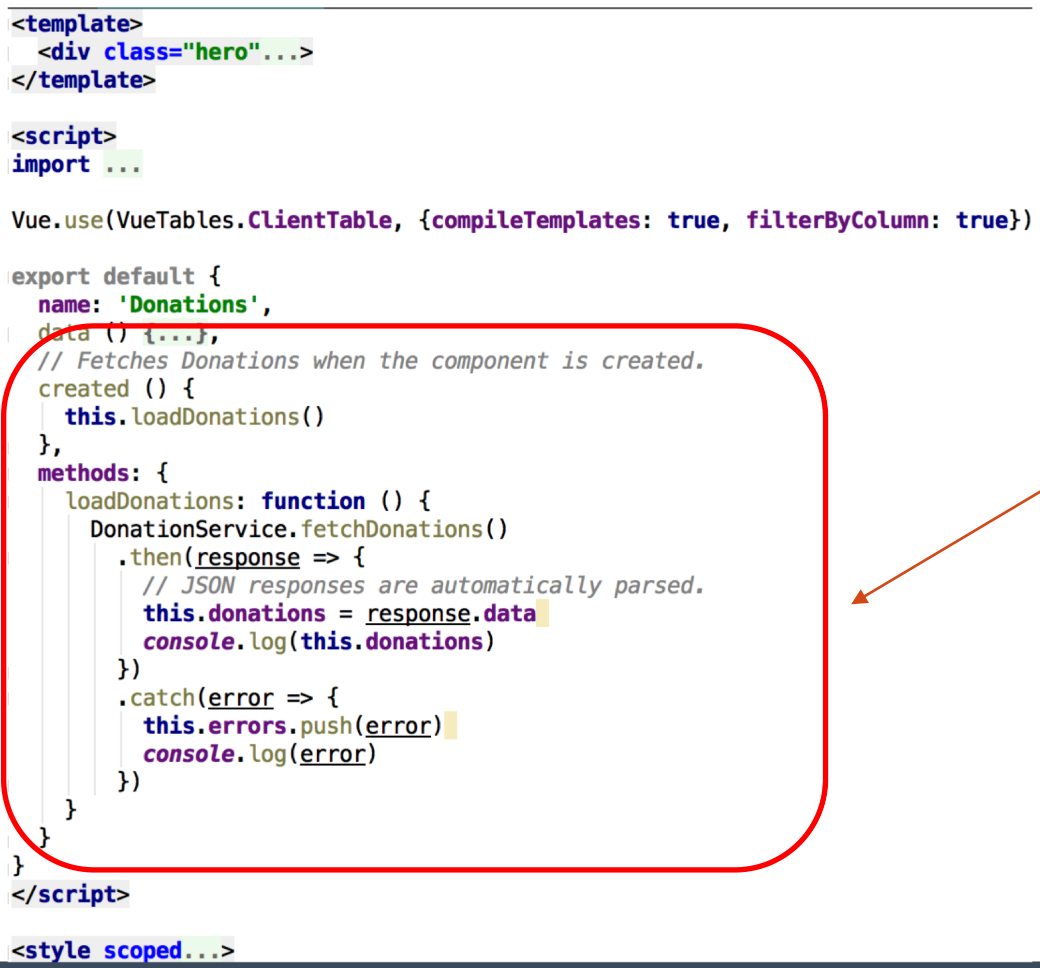
Component Styling

Donations.vue



```
<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">
      </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: [],
      errors: [],
      columns: ['_id', 'paymenttype', 'amount', 'upvotes'],
      options: {
        headings: {_id: 'ID'...}
      }
    }
  },
  // Fetches Donations when the component is created.
  created () {...},
  methods: {...}
}
</script>

<style scoped...>
```

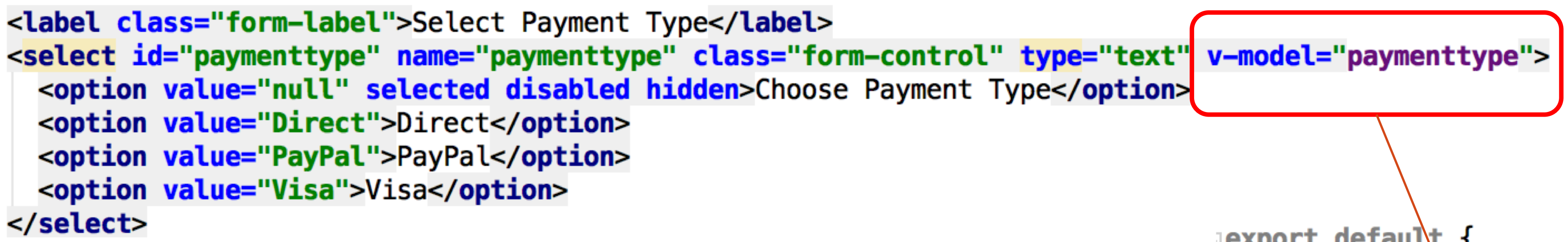
Donations.vue



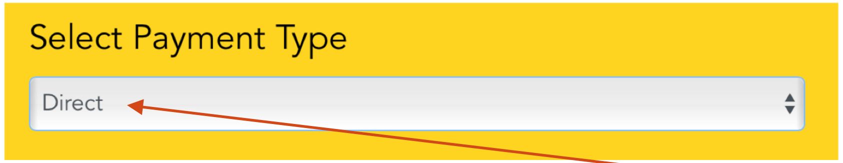
```
<template>
  <div class="hero"...>
</template>

<script>
import ...

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

export default {
  name: 'Donations',
  data () {...},
  // Fetches Donations when the component is created.
  created () {
    this.loadDonations()
  },
  methods: {
    loadDonations: function () {
      DonationService.fetchDonations()
        .then(response => {
          // JSON responses are automatically parsed.
          this.donations = response.data
          console.log(this.donations)
        })
        .catch(error => {
          this.errors.push(error)
          console.log(error)
        })
    }
  }
}
</script>

<style scoped...>
```

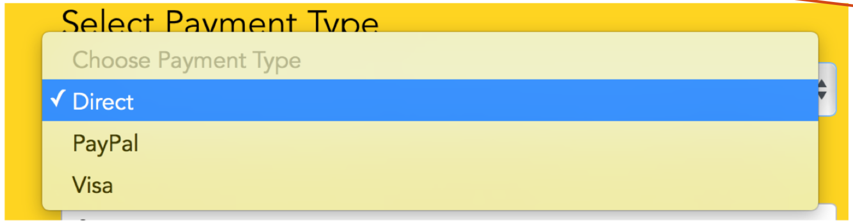
Event Handling (of sorts), Method Handling really, but this is how we will implement it eventually

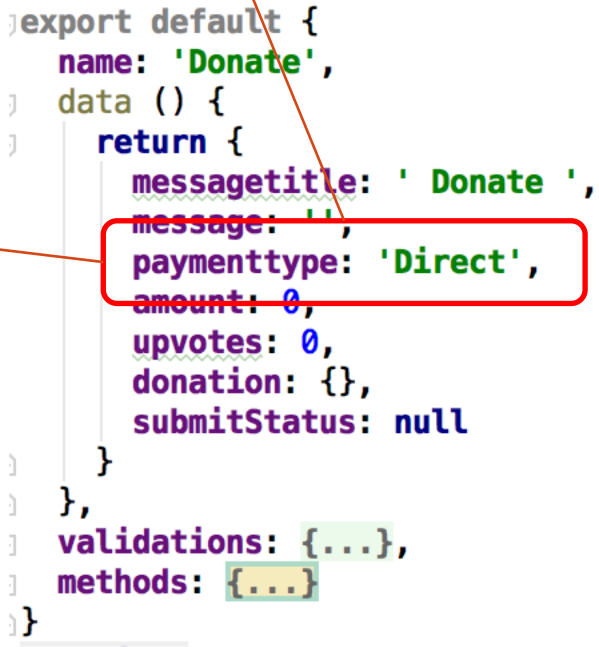
Donate.vue



```
<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>
```



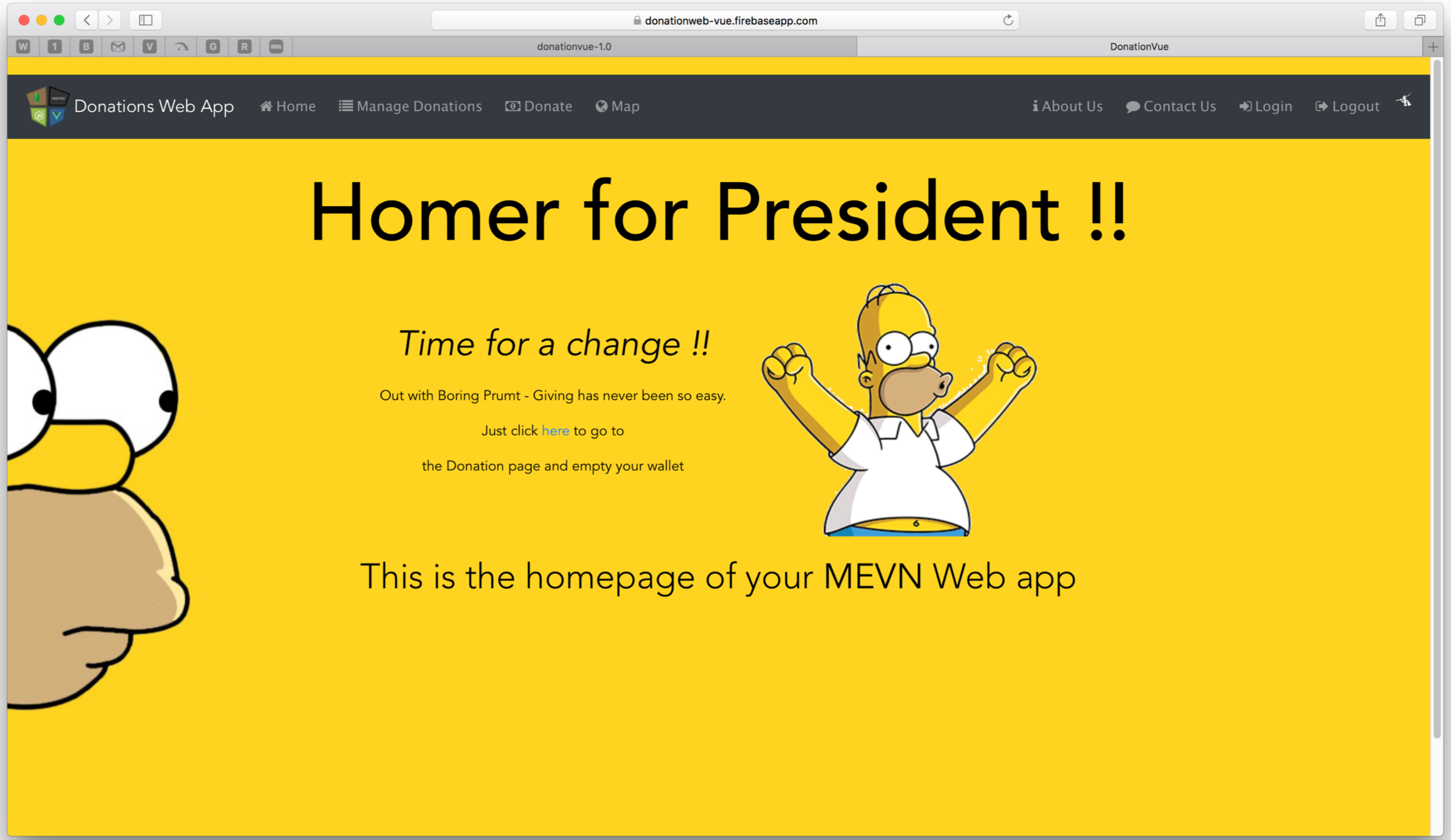




```
export default {
  name: 'Donate',
  data () {
    return {
      messagetitle: ' Donate ',
      message: '',
      paymenttype: 'Direct',
      amount: 0,
      upvotes: 0,
      donation: {},
      submitStatus: null
    }
  },
  validations: {...},
  methods: {...}
}
```

Part of our Input Form

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**
- ❑ **David Ličen,** davidlicen.com

Questions?