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 5

EVENT HANDLING IN VUE

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

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**

Event Handling in Vue

DEALING WITH USER INTERACTION

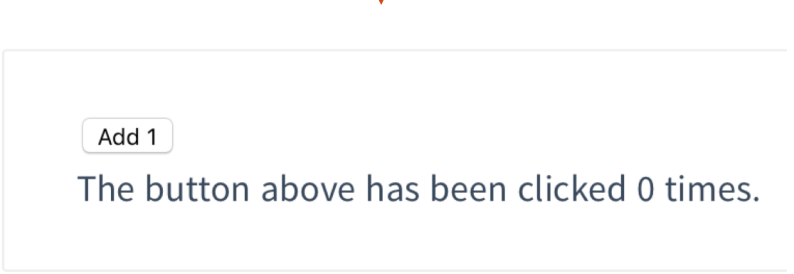
Introduction - Recap

As previously mentioned, we can use the **v-on** directive to listen to DOM events and run some JavaScript when they're triggered. For example:

```
<div id="example-1">
  <button v-on:click="counter += 1">Add 1</button>
  <p>The button above has been clicked {{ counter }} times.</p>
</div>
```

```
var example1 = new Vue({
  el: '#example-1',
  data: {
    counter: 0
  }
})
```

Result

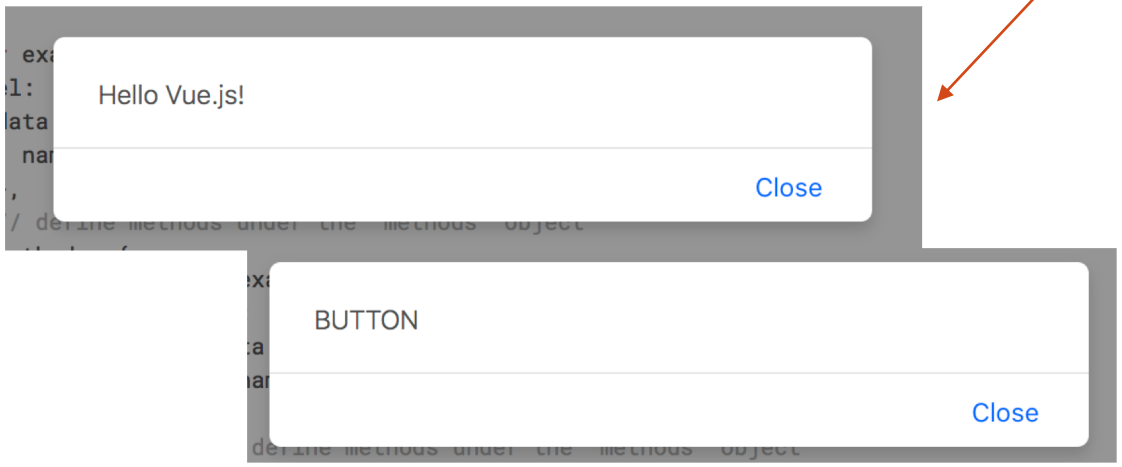


Introduction - Recap

The logic for many event handlers will be more complex though, so keeping your JavaScript in the value of the **v-on** attribute isn't feasible. That's why **v-on** can also accept the name of a method you'd like to call.

```
<div id="example-2">
  <!-- `greet` is the name of a method defined below -->
  <button v-on:click="greet">Greet</button>
</div>
```

Result




```
var example2 = new Vue({
  el: '#example-2',
  data: {
    name: 'Vue.js'
  },
  // define methods under the `methods` object
  methods: {
    greet: function (event) {
      // `this` inside methods points to the Vue instance
      alert('Hello ' + this.name + '!')
      // `event` is the native DOM event
      if (event) {
        alert(event.target.tagName)
      }
    }
  }
})
```

Event Handling in Depth *

As you've probably guessed by now, Vue.js allows us to handle events triggered by the user. Handling events helps add interactivity to web apps by responding to the user's input. User interactions with the view can trigger events on the DOM such as **click** and **keyup**, **enter**, etc.. As previously stated, Vue provides us with the **v-on** directive to handle these events.

Taking another simple "count" example we can bind methods to events using their names :

```
<input v-model="addValue">
<button @click="addToCount">Add</button>
```

The method **addToCount** specified in the template can be defined in the model as follows.

```
methods: {
  addToCount: function() {
    this.count = this.count + parseInt(this.addValue);
  }
}
```

The addToCount method will take the input from **addValue** and add that to the count.

Event Modifiers

There are frequently used calls that are made when handling events. Vue has made it easier for us to implement these by using modifiers.

For example, event.preventDefault() is often called when handling events to prevent the browsers default behaviour. Instead of having to write these out in the methods, we can use the modifiers provided with the vue-on directive.

```
<a href="test" @click.prevent="addToCount">Add</a>
```

The above code sample would remove the default behavior of the **a** tag and just call the **addToCount** method. If we didn't add the modifier, the page would try to re-direct to the path defined in the **href** attribute.

Event Modifiers

The following modifiers are available in Vue.

- **stop** - Prevents event bubbling up the DOM tree
- **prevent** - Prevents default behaviour
- **capture** - Capture mode is used for event handling
- **self** - Only trigger if the target of the event is itself
- **once** - Run the function at most once

Key Modifiers

Similar to **event** modifiers, we can add **key** modifiers that allow us to listen to a particular key when handling key-related events such as **keyup**.

```
<input v-on:keyup.13="addToCount" v-model="addValue">
```

In the above example, when the **keyup** event is fired with the key code of **13** (the enter key), the **addToCount** method gets called.

Since it's difficult to remember all of the key codes, Vue provides a set of pre-defined keys. Some examples are enter, **tab**, **delete**, **esc**, **space** and **left**, **right**, **up**, **down** .

Also, it's possible to setup your own alias for key codes as follows:

```
Vue.config.keyCodes.a = 65
```

Vue.js

Component Communication & Custom Events

The normal method for communication involves props and events. This common pattern provides a powerful way of communicating between components without introducing any dependency or limitations on which components are involved.

To recap, **Props** allow you to pass any data type to a child component, and allow you to control what sort of data your component receives. Prop updates are also reactive, allowing a child component to update whenever parent data changes.

```
<my-component v-bind:prop1="parentValue"></my-component>
<!-- Or more succinctly, -->
<my-component :prop1="parentValue"></my-component>
```

But what happens when the child component data changes and needs to inform the parent - that's where **Custom Events** come in handy.

Component Communication & Custom Events

Custom Events provide a way to inform your parent components of changes in children.

Parent Template:

```
<my-component v-on:myEvent="parentHandler"></my-component>
<!-- Or more succinctly, -->
<my-component @myEvent="parentHandler"></my-component>
```

Child Component

```
methods: {
  fireEvent() {
    this.$emit('myEvent', eventValueOne, eventValueTwo);
  }
}
```

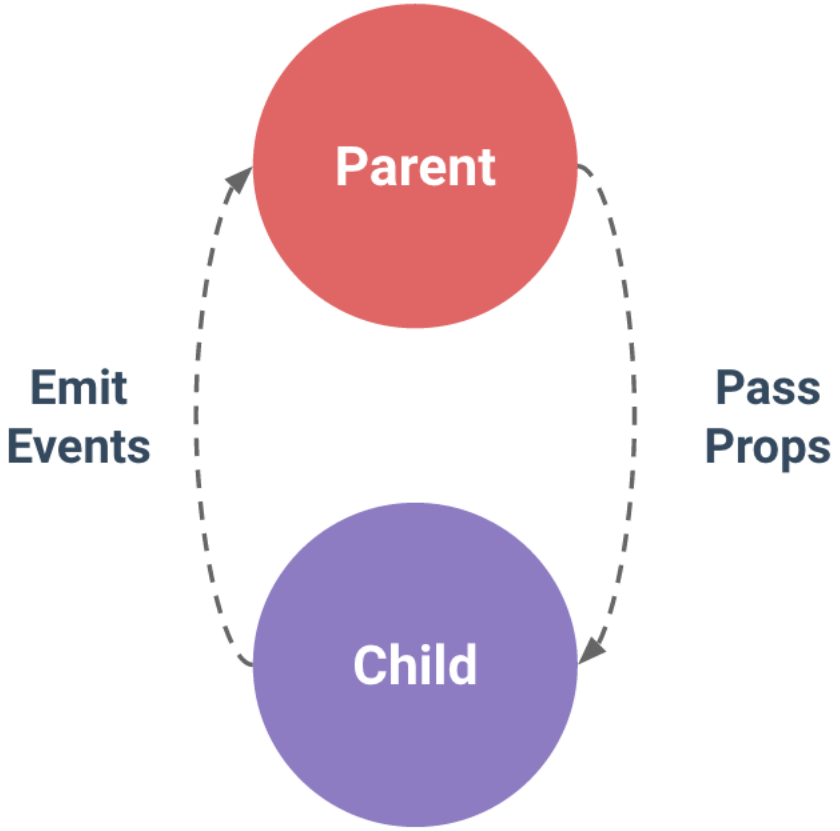
$emit triggers "**myEvent**" in parent

and we can pass up values

Using Props & $emit

The correct way to use Props & $emit is:

1. The Parent has data to manage, so you pass the data to the Child component via props.
2. If the Child component needs to modify the props data, assign that data to **separate** Child data values when the component is mounted.
3. After the method is handled, the Parent opens an interface to the Child.
4. The Child then $emit's back any new value(s) to the Parent.



Other Points of Note

- You can create **global event buses** to pass events *anywhere* in your app (see https://alligator.io/vuejs/global-event-bus/ for more info.
- Using v-model allows for combining props with events for two-way binding. This is often used for **input components**.

Case Study

LABS IN ACTION

Analysing our Case Study

So now that we've covered some more detail about Event Handling let's take a closer look at how we us them in **DonationVue**.

The main files of note are

- **`Donations.vue`**
- **`Donate.vue`**
- **`Edit.vue`**
- **`DonationForm.vue`**

so again basically everywhere ☺

Vue.js

Donations.vue

```
<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>
```

```
methods: {
  loadDonations: function () {...},
  upvote: function (id) {...},
  editDonation: function (id) {...},
  deleteDonation: function (id) {...}
}
```

@ (v-on) Click Events

Donate.vue (Parent)

```
<div class="col-md-6">
  <donation-form :donation="donation" donationBtnTitle="Make Donation"
                 @donation-is-created-updated="submitDonation"></donation-form>
</div><!-- /col -->
```

Passing in data via Props

@ (v-on) Custom Event

```
methods: {
  submitDonation: function (donation) {
    DonationService.postDonation(donation)
      .then(response => {
        console.log(response)
      })
      .catch(error => {
        this.errors.push(error)
        console.log(error)
      })
  }
```

Edit.vue (Parent)

```
<template v-if="childDataLoaded">
<donation-form :donation="donation" donationBtnTitle="Update Donation"
               @donation-is-created-updated="updateDonation"></donation-form>
</template>
```

Passing in data via Props

@ (v-on) Custom Event

```
updateDonation: function (donation) {
  console.log('Before PUT ' + JSON.stringify(donation, null, 5))
  DonationService.putDonation(this.$router.params, donation)
    .then(response => {
      console.log(response)
      console.log('AFTER PUT ' + JSON.stringify(donation, null, 5))
    })
    .catch(error => {
      this.errors.push(error)
      console.log(error)
    })
}
```

DonationForm.vue (Child)

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

Props from Parent

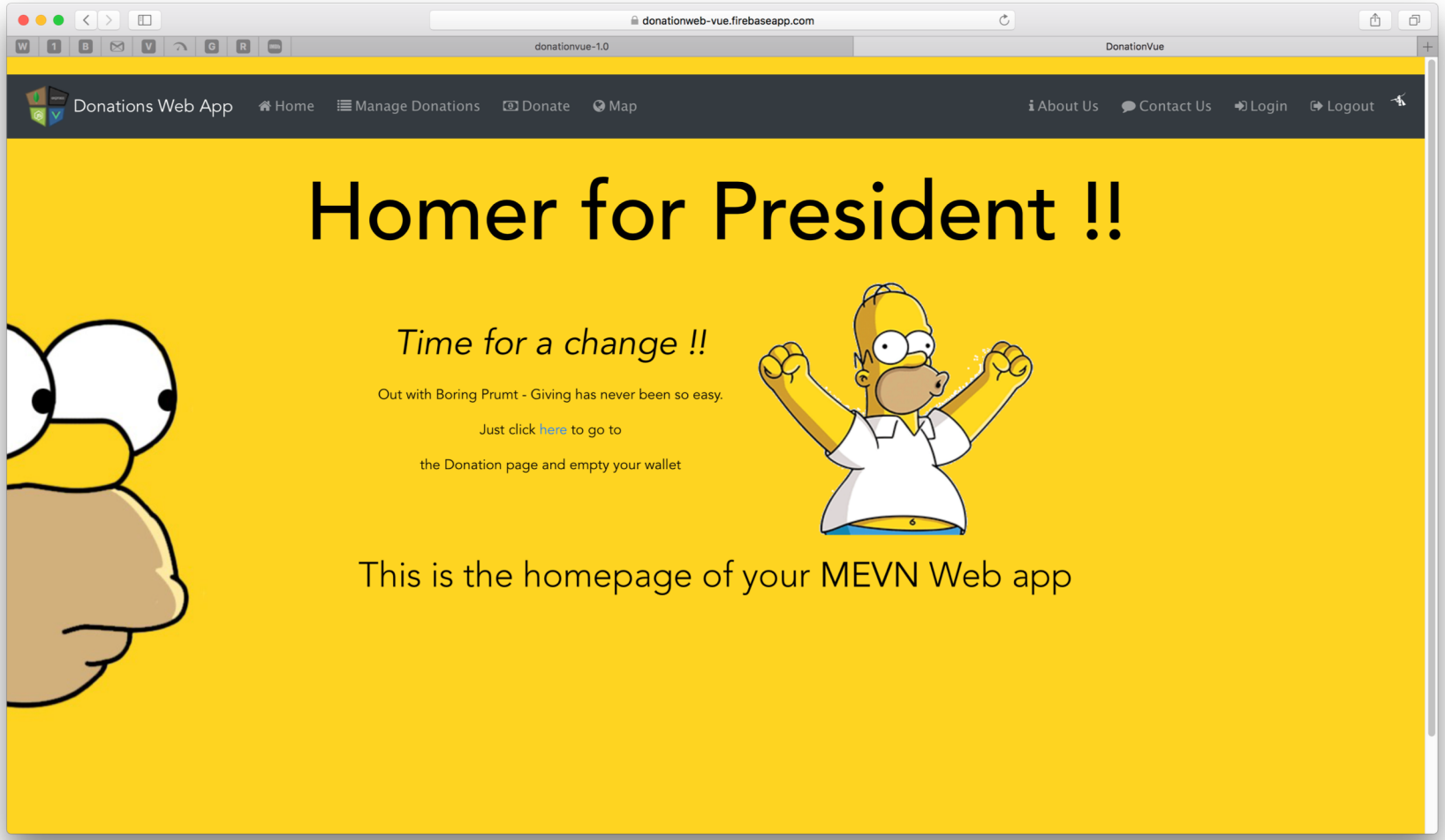
```vue
export default {
  name: 'FormData',
  props: ['donationBtnTitle', 'donation'],
  data () {
    return {
      messagetitle: ' Donate ',
      message: this.donation.message,
      paymenttype: this.donation.paymenttype,
      amount: this.donation.amount,
      upvotes: 0,
      submitStatus: null
    }
```

Assigning Parent data to Child data

```vue
methods: {
  submit () {
    console.log('submit!')
    this.$v.$touch()
    if (this.$v.$invalid) {...} else {
      // do your submit logic here
      this.submitStatus = 'PENDING'
      setTimeout(() => {
        this.submitStatus = 'OK'
        var donation = {
          paymenttype: this.paymenttype,
          amount: this.amount,
          upvotes: this.upvotes,
          message: this.message
        }
        this.donation = donation
        console.log('Submitting in DonationForm : ' +
          JSON.stringify(this.donation, null, 5))
        this.$emit('donation-is-created-updated', this.donation)
      }, 500)
    }
  }
}
```

"emitting" Child data back to Parent & triggering Event

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



References

- https://vuejs.org
- https://alligator.io/vuejs/events/
- https://alligator.io/vuejs/component-communication/

Questions?