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 7

COMPUTED PROPERTIES & WATCHERS

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**

Computed Properties & Watchers

REACTING TO DATA CHANGE

Vue.js

Introduction - Recap

In-template expressions are very convenient, but they are meant for simple operations. Putting too much logic in your templates can make them bloated and hard to maintain. For example:

```
<div id="example">
  {{ message.split('').reverse().join('') }}
</div>
```

At this point, the template is no longer simple and declarative. You have to look at it for a second before realizing that it displays **message** in reverse. The problem is made worse when you want to include the reversed message in your template more than once.

That's why for any complex logic, you should use a **computed property**.

Vue.js

Introduction - Recap

```
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>
```

```
var vm = new Vue({
  el: '#example',
  data: {
    message: 'Hello'
  },
  computed: {
    // a computed getter
    reversedMessage: function () {
      // `this` points to the vm instance
      return this.message.split('').reverse().join('')
    }
  }
})
```

Result

Original message: "Hello"

Computed reversed message: "olleH"

Introduction - Recap

While computed properties are more appropriate in most cases, there are times when a custom watcher is necessary. That's why **Vue** provides a more generic way to react to data changes through the **watch** option. This is most useful when you want to perform asynchronous or expensive operations in response to changing data.

As you can see in the code on the right, we're storing **counter** in **data**, and by using the ***name of the property as the function name***, we're able to watch it.

When we reference that **counter** in **watch**, we can observe any change to that property.

```
new Vue({
  el: '#app',
  data() {
    return {
      counter: 0
    }
  },
  watch: {
    counter() {
      console.log('The counter has changed!')
    }
  }
})
```

Vue.js

Computed Properties in Depth

There are multiple ways in Vue to set values for the view. This includes **directly binding** data value to the view, using simple **expressions** or using **filters** to do simple transformations on the content.

In addition to this, we can use **computed properties** to calculate display values based on a value or a set of values in the data model. These calculations will be *cached* and will only update when needed (more on this later).

They allow us to have model-specific, complex values computed for the view. These values will be bound to the dependency values and only update when required.

For example:

Computed Properties in Depth

We could have an array of subject results in the data model, like so :

```
data() {
  return {
    results: [
        {
          name: 'English',
          marks: 70
        },
        {
          name: 'Math',
          marks: 80
        },
        {
          name: 'History',
          marks: 90
        }
      ]
  }
}
```

Computed Properties in Depth

Assume that we want to view the total for all subjects. We shouldn't use **filters** or **expressions** for this task because :

- Filters are used for simple data formatting and that are needed at multiple places in the application.
- Expressions don't allow the use of flow operation or other complex logic. They should be kept simple.

Here's where computed properties come in handy. We can add a ***computed value*** to the model like this:

Vue.js

Computed Properties in Depth

```
computed: {
  totalMarks: function() {
    let total = 0;
    for(let i = 0; i < this.results.length; i++){
      total += parseInt(this.results[i].marks);
    }
    return total;
  }
}
```

The **totalMarks** computed property calculates the total marks using the results array. It simply loops through the values and returns the sub total.

Computed Properties in Depth

We can then simply display the computed value in the view:

```
<div id="app">
  <div v-for="subject in results">
    <input v-model="subject.marks">
    <span>
      Marks for {{ subject.title }}: {{ subject.marks }}
    </span>
  </div>

  <span>
    Total marks are: {{ totalMarks }}
  </span>
</div>
```

Computed Properties vs Methods

Now, we could get the same result by using a method that outputs the total.

Instead of having the **totalMarks** function in the computed section, we can move it to the methods and in the view we can change the template to execute the method, like so:

```
<span>
  Total marks are: {{ totalMarks() }}
</span>
```

While this gives the same output, we are taking a performance hit. Using this syntax, the **totalMarks()** method gets executed ***every time*** the page renders (ie: with every change).

Vue.js

Computed Properties vs Methods

If instead we had a computed property, Vue remembers the values that the computed property is dependent on (eg: In the previous example, that would be **results**). By doing so, Vue can calculate the values **only** if the dependency changes. Otherwise, the previously cached values will be returned.

Because of this, the function must be a pure function. It can't have side-effects. The output must only be dependent on the values passed into the function.

So imagine 3000 marks or even 30K, not just 3, as in this example, but in cases where you do not want caching, use a method instead.

Vue.js

Computed Setters

By default, the computed properties only present a **getter**. But, it's also possible to have **setters**.

```
computed: {
  fullName: {
    get: function() {
      return this.firstName + this.lastName;
    },
    set: function(value) {
      let names = value.split(' ');
      this.firstName = names[0];
      this.lastName = names[names.length - 1];
    }
  }
}
```

By having both getters and setters, we can bind the input value correctly to the model. If we **set** the **fullName** in a method, the passed-in string will be split into the first and last name.

Watchers in Depth

While computed properties may be sufficient in most cases, **watchers** provide an additional level of control by allowing us to listen for changes to a property.

Watchers, as the name suggests, allows us to watch for changes in a model object. While it's possible to use watchers to get the same results as computed values, it's often more complex and expensive.

We can use watchers for more complex requirements, for example:

- Async operations
- Setting intermediate values
- Limiting the number of times an operation gets called (eg: Debounce an input event)

If we want to add a bit of functionality each time something changes, or respond to a particular change, we could **watch** a property and apply some logic. This means that the name of the watcher ***has to match*** what we're trying to observe.

Watchers in Depth

Vue grants us some deeper access into the reactivity system, which we can leverage as hooks to observe anything that's changing. This can be incredibly useful because, as application developers, *most of what we're responsible for are things that change*.

Watchers also allow us to write much more declarative code. You're no longer tracking everything yourself. Vue is already doing it under the hood, so you can also have access to changes made to any properties it's tracking, in data, computed, or props, for example.

Watchers are incredibly good for executing logic that applies to ***something else*** when a change on a property occurs. This isn't a hard & fast rule - you can absolutely use watchers for logic that refers to the property itself, but it's a nice way of looking at how watchers are immediately different from computed properties, where the change will be in reference to the property we intend to use.

Here's a very basic example :

Watchers in Depth

As you can see in the code we're storing **counter** in **data**, and by using ***the name of the property as the function name***, we're able to watch it.

When we reference that counter in watch we can observe any change to that property.

```
new Vue({
  el: '#app',
  data() {
    return {
      counter: 0
    }
  },
  watch: {
    counter() {
      console.log('The counter has changed!')
    }
  }
})
```

Vue.js

Computed vs Watched Properties

When you have some data that needs to change based on some other data, it is tempting to overuse **watch** - especially if you are coming from an AngularJS background.

However, it is often a better idea to use a **computed property** rather than an imperative **watch** callback.

Consider the following :

```
<div id="demo">{{ fullName }}</div>
```

Computed vs Watched Properties

```
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar',
    fullName: 'Foo Bar'
  },
  watch: {
    firstName: function (val) {
      this.fullName = val + ' ' + this.lastName
    },
    lastName: function (val) {
      this.fullName = this.firstName + ' ' + val
    }
  }
})
```

The above code is imperative and repetitive. Compare it with a computed property version :

Computed vs Watched Properties

```
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar'
  },
  computed: {
    fullName: function () {
      return this.firstName + ' ' + this.lastName
    }
  }
})
```

What do you think?

When to use them…

WHEN TO USE METHODS

- To react to some event happening in the DOM
- To call a function when something happens in your component. You can call methods from computed properties or watchers.

When to use them…

WHEN TO USE COMPUTED PROPERTIES

- You need to compose new data from existing data sources
- You have a variable you use in your template that’s built from one or more data properties
- You want to reduce a complicated, nested property name to a more readable and easy to use one, yet update it when the original property changes
- You need to reference a value from the template. In this case, creating a computed property is the best thing because it’s cached.
- You need to listen to changes of more than one data property

When to use them…

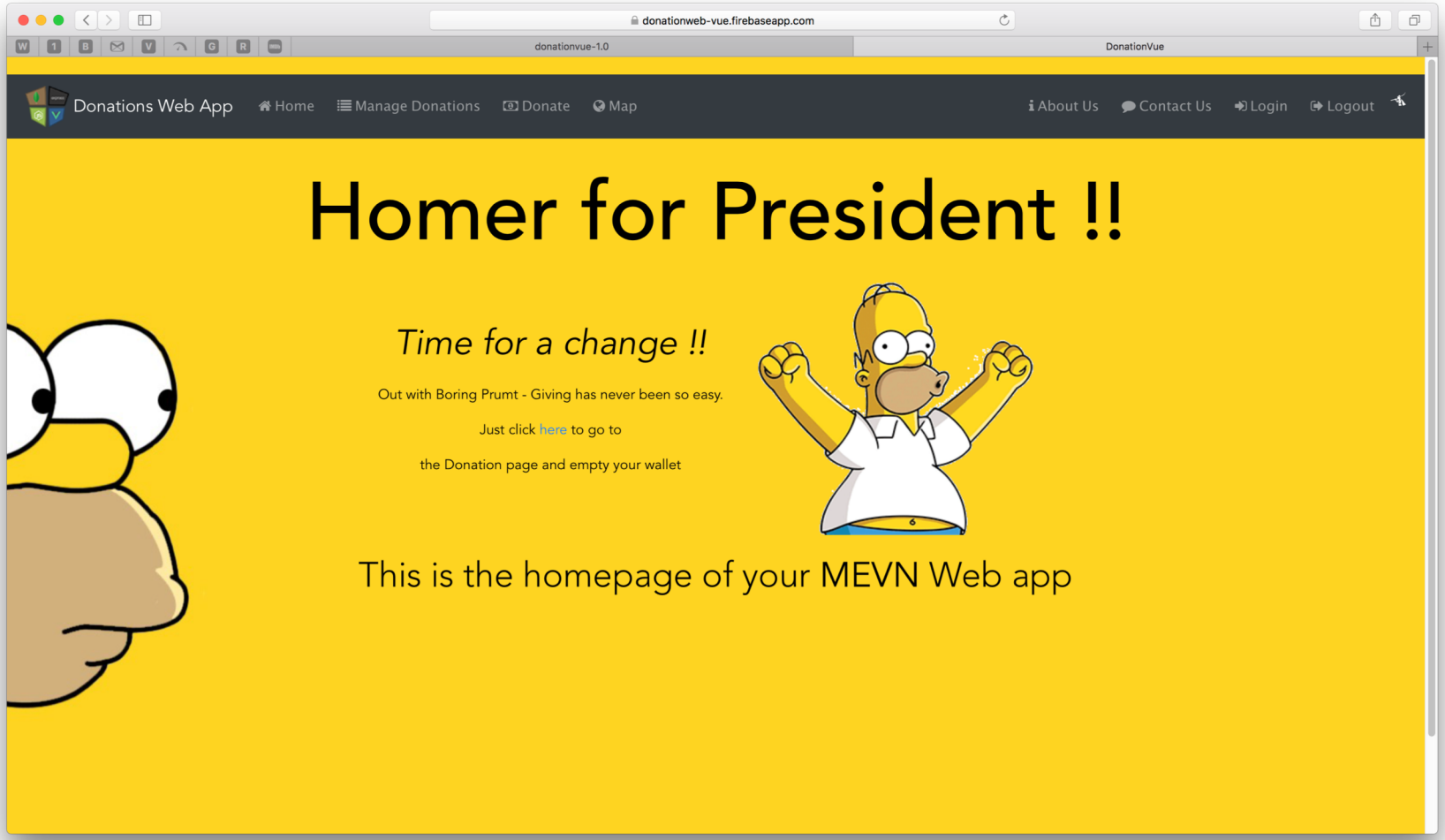
WHEN TO USE WATCHERS

- You want to listen when a data property changes, and perform some action
- You want to listen to a prop value change
- You only need to listen to one specific property (you can’t watch multiple properties at the same time)
- You want to watch a data property until it reaches some specific value and then do something

Case Study

LABS IN ACTION

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



References

- https://vuejs.org
- https://alligator.io/vuejs/computed-properties/
- https://flaviocopes.com/vue-methods-watchers-computed-properties/
- https://css-tricks.com/methods-computed-and-watchers-in-vue-js/

Questions?