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

INTRODUCTION, TERMINOLOGY & OVERVIEW

Objectives

To give you a foundation on which you can begin to understand all of the other tutorials and documentation out there.

We will cover…

- Why it's worth using it
- The Terminology (for all the resources available, including ours)
- and How to use the Vue 'Building Blocks & Core Features'

We will also cover how to build your first Vue application (mostly in the Labs!)

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**

Introduction

WHY YOU SHOULD BE USING VUE 2.0

What is Vue.js?

- Vue (pronounced /vjuː/, like **view**) is a **progressive framework** for building user interfaces.
- Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects.
- On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with **modern tooling** and **supporting libraries**.

progressive

/prəˈgrɛsɪv/

adjective

1. happening or developing gradually or in stages.
   "a progressive decline in popularity"
   synonyms: continuing, continuous, increasing, growing, developing, ongoing, intensifying, accelerating, escalating; gradual, step by step, cumulative

Vue.js

The Founder

Evan You

- Previously worked as a Creative Technologist at **Google** (using **AngularJS**)
- Core Developer at **Meteor**
- From 2016 working fulltime on the VueJS Framework

He later summed up his thought process, "I figured, what if I could just extract the part that I really liked about Angular and build something really lightweight"

Vue.js

History

- Started in **late 2013**
- First release Feb. 2014 (v0.6)
- **v1.0.0** Evangelion Oct. 2015
- Latest release **v2.5.6** (July 2018)

https://github.com/vuejs/vue/releases

Vue.js

MVVM Pattern (Model-View-ViewModel)

The well-ordered and perhaps the most reusable way to organize your client-side code is to use the 'MVVM' pattern. The **Model, View, ViewModel (MVVM pattern)**is all about guiding you in how to organize and structure your code to write maintainable, testable and extensible applications.

Model – It simply holds the data and has nothing to do with any of the business logic.

ViewModel – It acts as the link/connection between the Model and View and makes stuff look pretty.

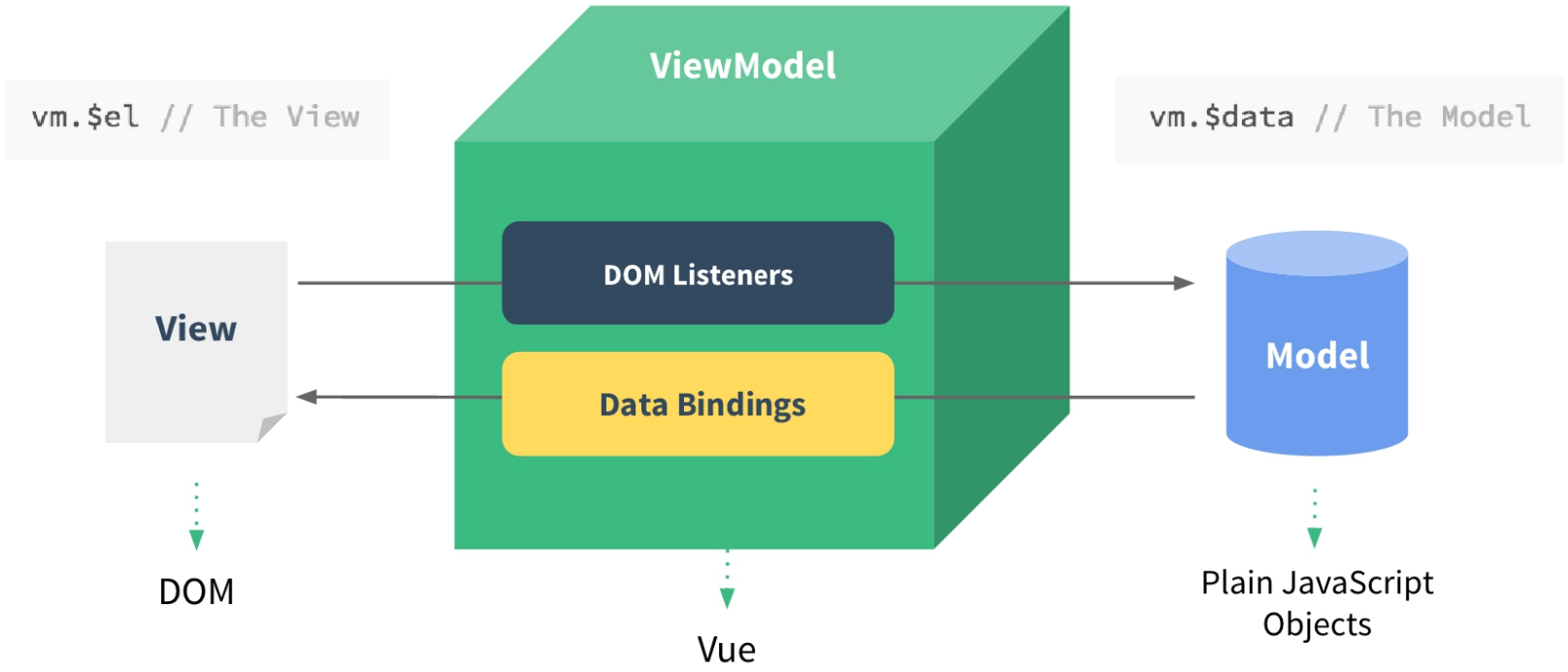
View – It simply holds the formatted data and essentially delegates everything to the Model.

Vue is build around **MVVM** connecting **View** and **Model** with **2-Way Reactive Binding**

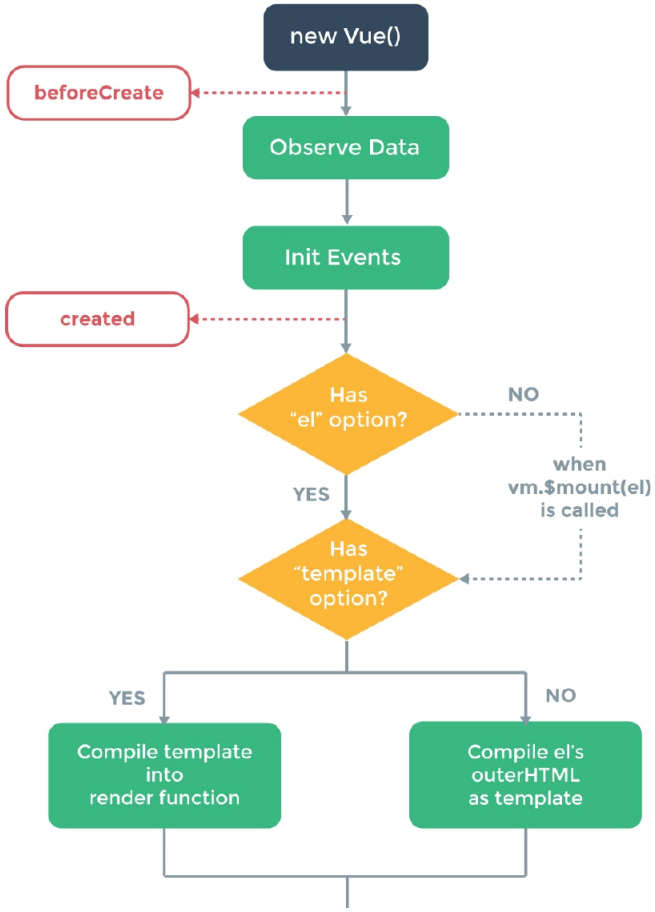
Vue.js

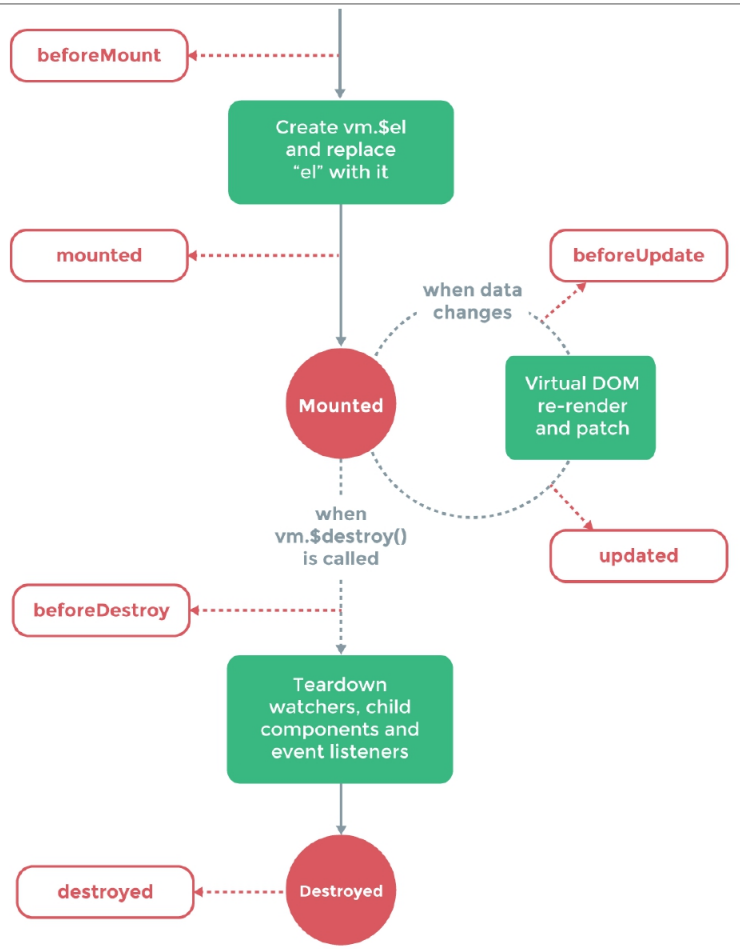
MVVM Pattern (Model-View-ViewModel)

```
var vm = new Vue({ /* options */ })
```



Instance Lifecycle





Github Stats – July 2018

Ways to Install & Use Vue.js

- Standalone – Include <script> with the CDN and you are good to go
- NPM – Node package manager. Great with Browserify or Webpack
- Vue-CLI – Command line tool uses **Webpack**
- Bower – Client side package manager

Vue.js

Vue Ecosystem (and a lot more)

Project	Status	Description
vue-router	npm v3.0.1	Single-page application routing
vuex	npm v3.0.1	Large-scale state management
vue-cli	npm v2.9.6	Project scaffolding
vue-loader	npm v15.2.4	Single File Component (`*.vue` file) loader for webpack
vue-server-renderer	npm v2.5.16	Server-side rendering support
vue-class-component	npm v6.2.0	TypeScript decorator for a class-based API
vue-rx	npm v6.0.0	RxJS integration
vue-devtools	chrome web store v4.1.4	Browser DevTools extension

Terminology & Overview

THE CRITICAL FOUNDATION FOR UNDERSTANDING

Vue Core Features

As previously mentioned, the core library is focused on the **view layer only**, and is easy to pick up and integrate with other libraries or existing projects. But Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with **modern tooling** and **supporting libraries**.

What follows is a very brief overview of those core features of Vue which we'll cover in more detail later on, and demonstrate with examples from our Case study - **DonationVue**

1. **Declarative Rendering & Reactivity**
2. **Components**
3. **Routing**
4. **Directives**
5. **Event Handling**
6. **Filters**
7. **Computed Properties & Watchers**
8. **Transitioning Effects**

1. Declarative Rendering & Reactivity

At the core of Vue.js is a system that enables us to declaratively render data to the DOM using straightforward template syntax:

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

"Mustache" syntax

Result in Browser

Hello Vue!

```
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**.

1. Declarative Rendering & Reactivity

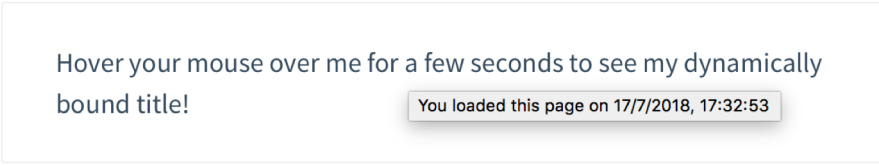
In addition to **text interpolation**, we can also bind element attributes like this:

```
<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>
```

Directive

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

Result in Browser



1. Declarative Rendering & Reactivity

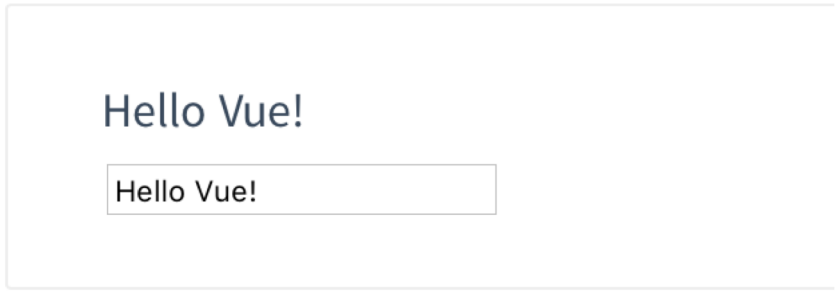
And **two-way data binding** like this:

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

Directive

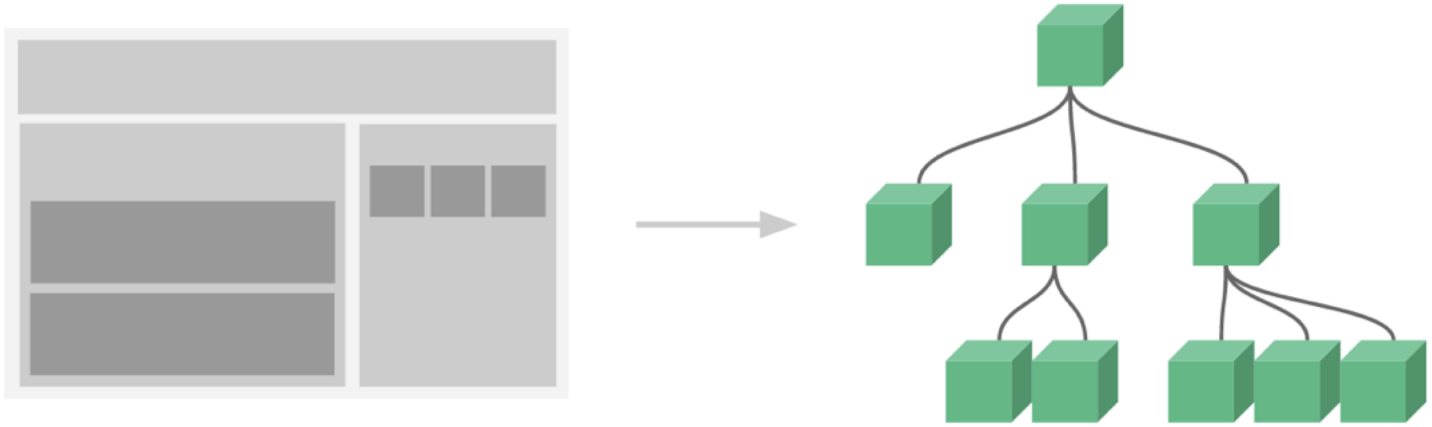
```
var app6 = new Vue({
  el: '#app-6',
  data: {
    message: 'Hello Vue!'
  }
})
```

Result in Browser



2. Components

The component system is another important concept in Vue, 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:



2. Components

The component system is another important concept in Vue, 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:

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>
```

2. Components

The component system is another important concept in Vue, 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:

Or modularized into their own **.vue** files

3. Routing

Routing is a key part of all websites and **web applications** in one way or another. It plays a central role in static HTML pages as well as in the most complex **web applications**. **Routing** comes into play whenever you want to use a URL in your **application**.

In **Vue**, for most Single Page Applications, it's recommended to use the officially-supported **vue-router library** – which is what we'll be doing.

It is often more convenient to identify a route with a name, especially when linking to a route or performing navigations. You can give a route a name in the routes options while creating the Router instance:

index.js

```
Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/',
      name: 'Home',
      component: Home
    },
    {
      path: '/donations',
      name: 'Donations',
      component: Donations
    },
    {name: 'Donate'...},
    {name: 'AboutUs'...},
    {name: 'ContactUs'...}
  ]
})
```

4. Directives

Directives are special attributes with the **v-** prefix. Directive attribute values are expected to be **a single JavaScript expression** (with the exception of **v-for**, which will be discussed in later sections).

A directive's job is to reactively apply side effects to the DOM when the value of its expression changes. For example:

```
<p v-if="seen">Now you see me</p>
```

Here, the **v-if** directive would remove/insert the **<p>** element based on the truthiness of the value of the expression **seen**.

4. Directives

Some directives can take an “argument”, denoted by a colon after the directive name.

For example, the **v-on** directive is used to reactively listen to DOM events:

```
<a v-on:click="doSomething"> ... </a>
```

Here the argument is the event name to listen to. We will talk about event handling in more detail too (but a bit about it next) .

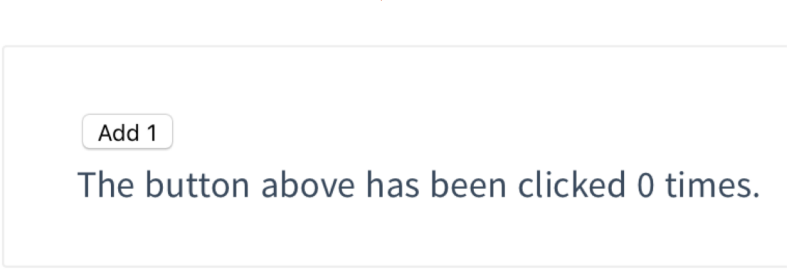
5. Event Handling

As just 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

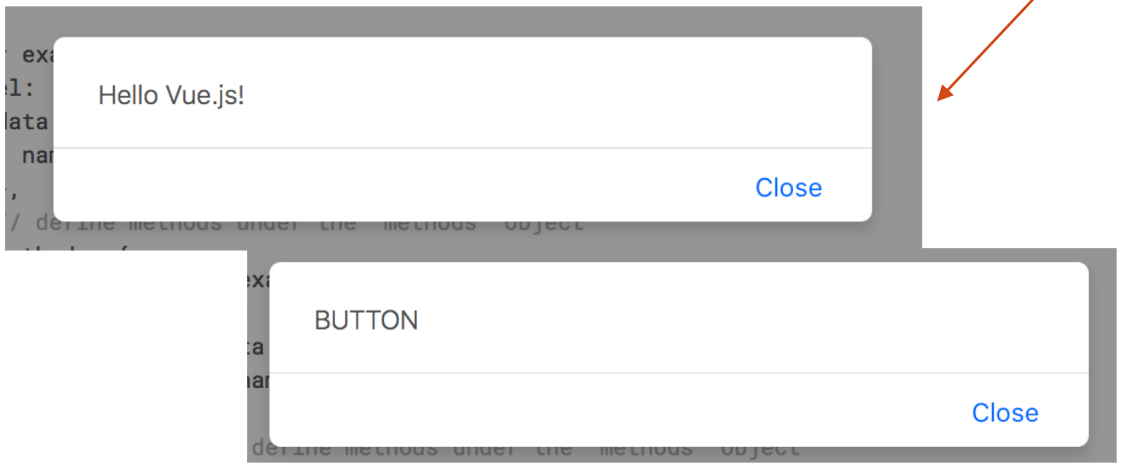


5. Event Handling

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)
      }
    }
  }
})
```

6. Filters

Vue.js allows you to define filters that can be used to apply common text formatting. Filters are usable in two places: **mustache interpolations** and **v-bind expressions** (the latter supported in 2.1.0+). Filters should be appended to the end of the JavaScript expression, denoted by the "pipe" symbol:

```
<!-- in mustaches -->
{{ message | capitalize }}

<!-- in v-bind -->
<div v-bind:id="rawId | formatId"></div>
```

Vue.js

7. Computed Properties & Watchers

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

7. Computed Properties & Watchers

```
<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"

7. Computed Properties & Watchers

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.

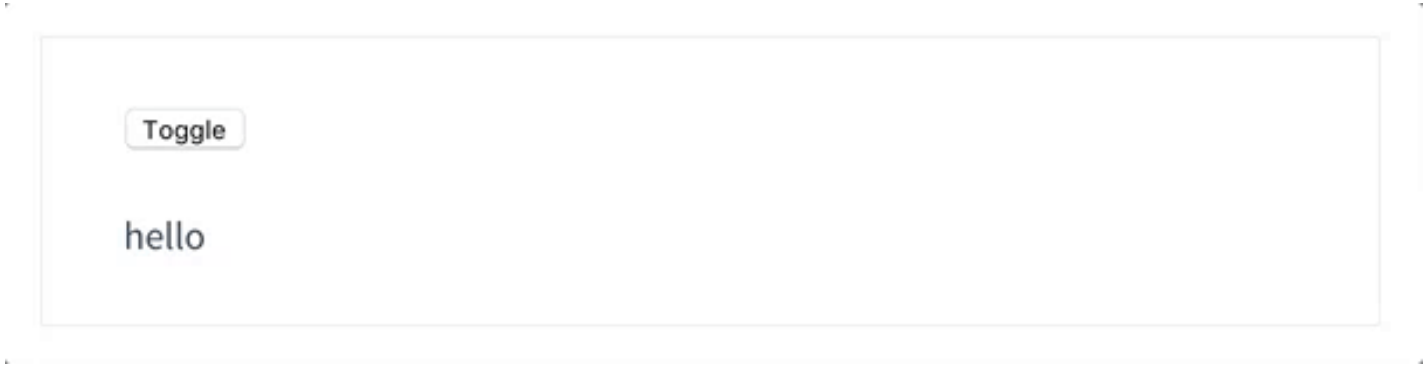
As you can see in the code on the right, we're storing **counter** in **data**, and by using the ***name of the property as the function name***, we're able to watch it.

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!')
    }
  }
})
```

8. Transitioning Effects

Transitions in Vue allow you to apply effects to elements when they are inserted, updated or removed from the DOM. For example, the classic fade:



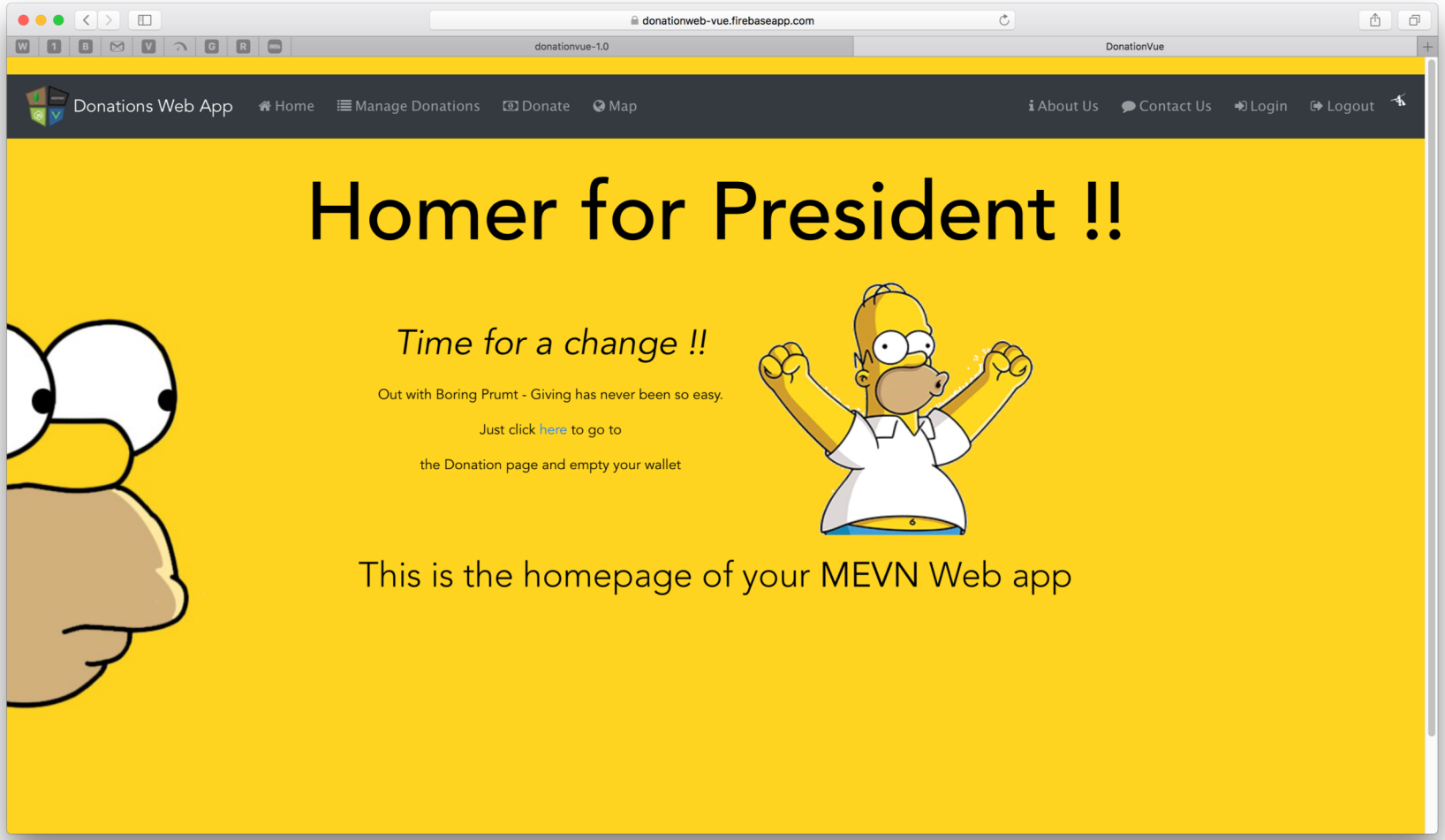
The transition system has been a feature of Vue since the first version, but in version 2 there have been some changes, mainly that it is now completely component-based (which is probably a much better approach…).

We'll take a closer look at Transitions in later sections (Part 3).

Case Study

LABS IN ACTION

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



References

- **https://vuejs.org**
- **David Ličen,** davidlicen.com

Questions?