

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 8

TRANSITIONING EFFECTS

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

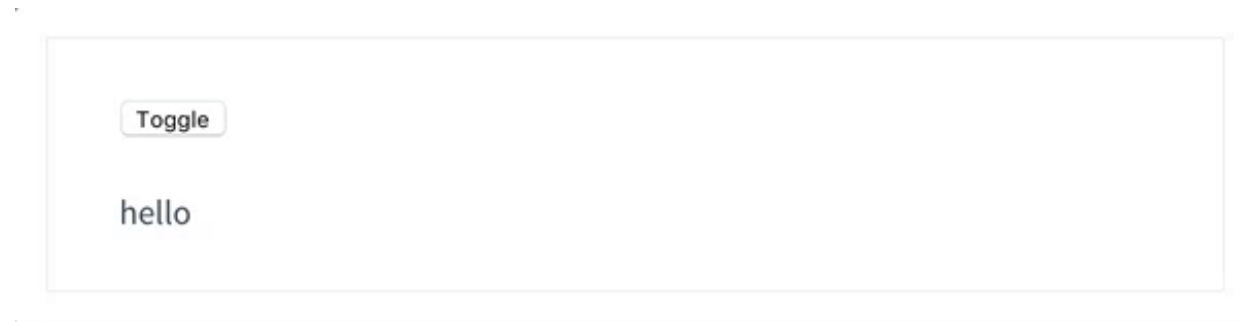
1. **Introduction** – Why you should be using VueJS
2. **Terminology & Overview** – The critical foundation for understanding
3. **Declarative Rendering & Reactivity** – Keeping track of changes (Data Binding)
4. **Components** – Reusable functionality (Templates, Props & Slots)
5. **Routing** – Navigating 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**

Transitioning Effects

I LIKE YOUR <STYLE>

Introduction - Recap

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 now.

Transitioning Effects in Depth

Transitions aren't just for adding pretty 'bells & whistles' to your app. A good transition can be the difference between a user signing up, making a purchase, or completely leaving your site. For example, Amazon found that it cost them about 1% in sales for every 100ms of latency.

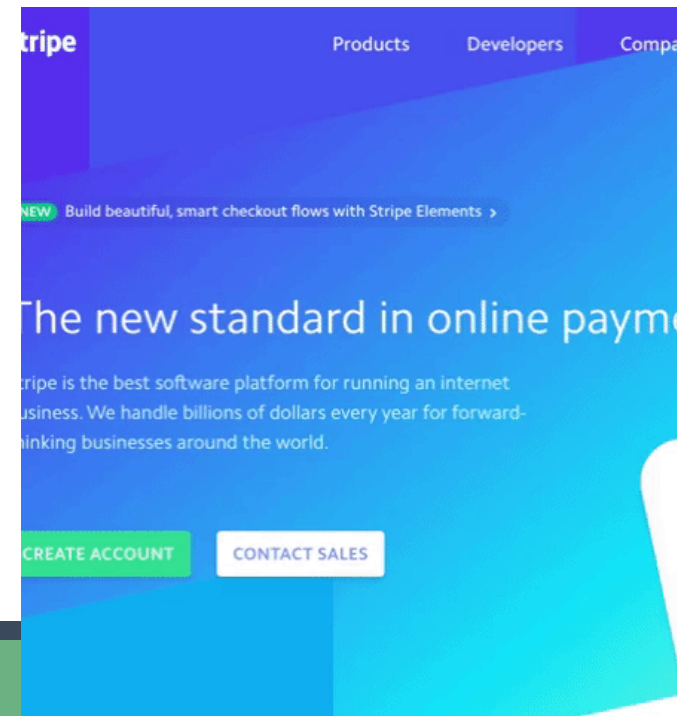
You can mitigate the risk of users leaving your site with a good transition and the way Vue.js handles transitions and animations makes it super easy to add them to your application.

Vue.js gives you a variety of ways to animate your application, including CSS transitions and animations, and using JavaScript to manipulate the DOM during transition hooks. You can even hook up to third-party libraries such as [GSAP](#) or [VelocityJS](#).

Transitions vs Animations *

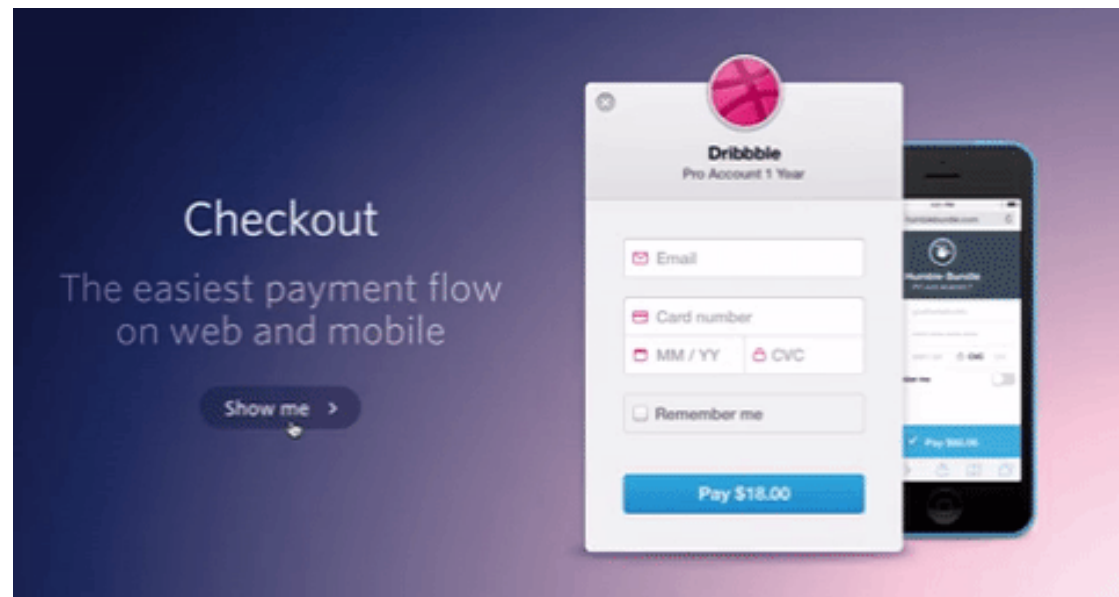
A transition is when an element goes from A to B. They are limited to two stages: going from the start state (A) to the end state (B). They work really well for things like hovering over links and buttons, or adding a new item to a list. The main purpose of transitions is to create a natural demonstration that something has changed. A good example of a transition in the real world would be automatic doors. They start closed and when you step in front of them, they automatically transition into the open state.

Here's an example of a simple transition from Stripe's homepage.



Transitions vs Animations *

Animations, on the other hand, can have many in-between states known as keyframes. They can go from A to B to C to D and beyond. They are useful for things like a loading spinner or an audio visualizer. Their main purpose is to continuously showcase that something is changing. They can have ending states, but unlike transitions they're not limited to two states. In nature, a whirlpool would be an example of an animation. It's continuously in a changing and whirling state, but it might come to an end eventually.



Transitions vs Animations

In a way, animations are just *super* transitions because they add more in-between states.

While transitions only go from A to B, animations can have as many in-between states as necessary. After understanding the fundamentals of transitions, making the jump to animations is an easy step.

In this section, we'll focus mainly on **transitions** and how to create them. We'll also look at how to create the 'fade' and 'slide in' transitions from the DonationVue Videos.

The Transition Element

Vue.js has a wrapper **<transition>** element that makes it simple to handle JavaScript animations, CSS animations, and CSS transitions on your elements or components.

In the case of CSS transitions, the **<transition>** element takes care of applying and un-applying classes. All you have to do is define how you want the element to look during the transition.

It acts like any other Vue.js component, and you can use the same directives on it like **v-if** and **v-show**.

For Example :

```
<template>
  <transition name= "fade" >
    <alligator v-if="alligatorDefinition"></alligator>
  </transition>
</template>
```

Transition Classes

The transition element applies six classes to your markup that you can use to separately handle your enter and leave transitions. There are three classes to handle the A to B transition for when the element is displayed, and another three to handle the A to B transition for when the element is removed.

The enter transition happens when the component is being enabled, or displayed. Those classes are: **v-enter**, **v-enter-active**, **v-enter-to**

The leave transition is when the component is being disabled, or removed. Those classes are: **v-leave**, **v-leave-active**, and **v-leave-to**

The **v-** prefix is the default when you use a `<transition>` element with no name. In our case since we named the previous transition **fade**, the **v-enter** class would instead be **.fade-enter**.

How it Works

So basically, Vue works by adding and removing these classes on components / elements during the transition process. When we **name** a **<transition>** the relevant classes become :

- **...-enter**
- **...-leave**
- **...-enter-active**
- **...-leave-active**
- **...-enter-to**
- **...-leave-to**

where **...** is the name of your transition. Vue appends these class names to the name of your transition (that's just the way it is!). So, in the following example our transition name is "**slither**", and the classes become **slither-enter**, **slither-leave**, **slither-leave-active**, etc.

So what do we use these classes for?

Used for... Handling Timing

Well, let's start with the easy ones, **...-enter-active** & **...-leave-active**.

Use these classes to define which properties to transition and the duration and timing functions for them. And that's it.

Before Vue 2.1.8, you had to use this class to define the ending state of the transition as well, which complicated things.

So, you would usually use it like this:

```
.slither-enter-active, .slither-leave-active {  
  transition: transform 3s;  
}
```

Now we just need to define how it starts and ends.

Used for... Handling Start & End Times

...-enter and **...-leave** handle the **start states** for entering and leaving, respectively.

...-enter-to and **...-leave-to** handle the **end states** for entering and leaving, respectively.

(Just to clarify, The **enter** transition happens when the component is being enabled or displayed. **leave** is the transition when the component is being disabled or removed.)

So, let's have the slither transition start with a transform that's way off to the left, then move to the normal position:

```
.slither-enter {  
  transform: translateX(-100%);  
}  
  
.slither-enter-to {  
  transform: translateX(0);  
}
```

Used for... Handling Start & End Times

Now, usually, you want the leave transition to simply be the inverse of the enter transition. To do that, we simply add the opposite leave classes to the enter classes :

```
.slither-enter, .slither-leave-to {  
  transform: translateX(-100%);  
}  
  
.slither-enter-to, .slither-leave {  
  transform: translateX(0);  
}
```

If you're writing different transitions for entering and leaving, you can separate the classes however you'd like.

Used for... Handling Start & End Times

Putting it all together...

This will cause the alligator component to slide in from the left, and if you set **alligatorDefinition** to a falsey value, it will slide out to the left over three seconds.

```
<template>
  <transition name="slither">
    <alligator v-if="alligatorDefinition"></alligator>
  </transition>
</template>
```

Used for... Handling Start & End Times

Putting it all together...

This will cause the alligator component to slide in from the left, and if you set **alligatorDefinition** to a falsy value, it will slide out to the left over three seconds.

```
<script>
export default {
  data() {
    return {
      alligatorDefinition: null
    }
  },

  mounted() {
    alligatorDefinition = {}
  }
}
</script>
```

Used for... Handling Start & End Times

Putting it all together...

This will cause the alligator component to slide in from the left, and if you set **alligatorDefinition** to a falsy value, it will slide out to the left over three seconds.

```
<style>
.slither-enter-active, .slither-leave-active {
  transition: transform 3s;
}

.slither-enter, .slither-leave-to {
  transform: translateX(-100%);
}

.slither-enter-to, .slither-leave {
  transform: translateX(0);
}
</style>
```

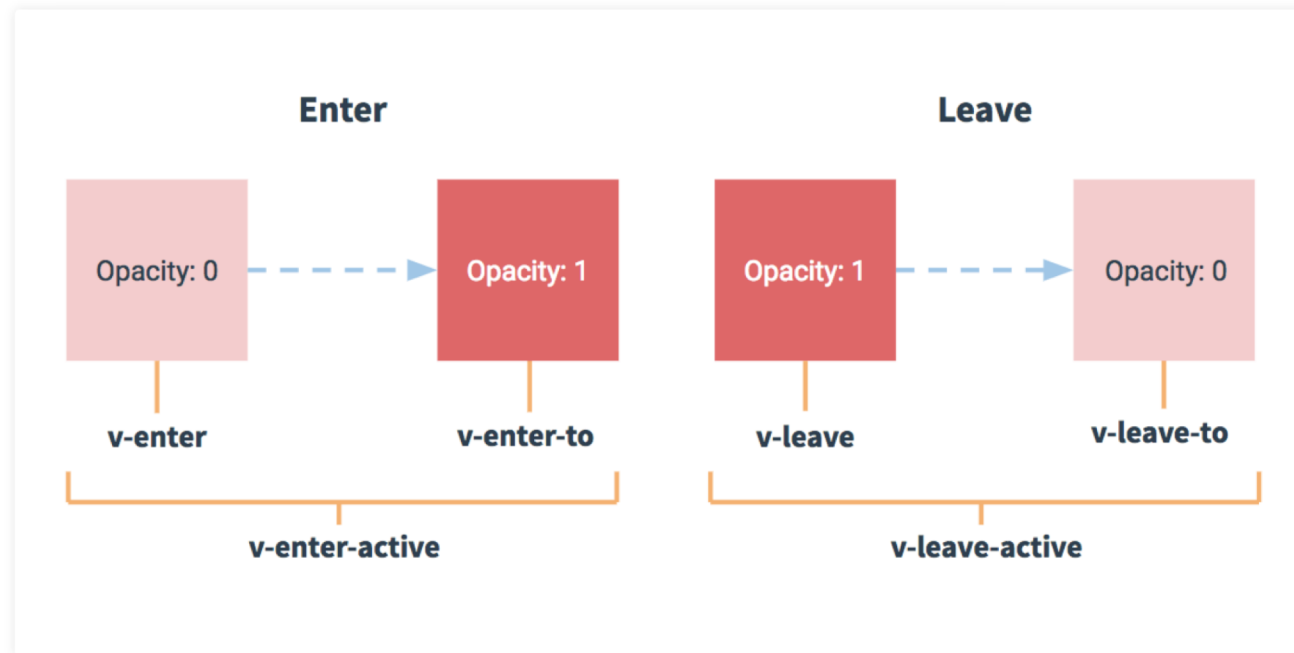
Recap

The important things to remember are:

- **...-enter-active** & **...-leave-active** should only be used to define transition properties. Anything else will make things confusing.
- **...-enter** should contain start styles for when the component starts to appear.
- **...-enter-to** should contain end styles for when the component finishes appearing.
- **...-leave** is used in the same way as **...-enter**, but for when the component starts disappearing. It should usually contain the same properties as **...-enter-to**
- **...-leave-to** is used in the same way as **...-enter-to**, but for when the component finishes disappearing.
- You can still use **v-** as the prefix for any unnamed (default) transitions.

Recap

If that's still confusing, here's an image (courtesy of the [Vue.js official docs](https://vuejs.org/guide/essentials/transitions.html)) that might clear things up a bit.



There's plenty more to learn about transitions, so it would be a good idea to take a look at the official docs for more information (especially in relation to your C.A. !! 😊)

Case Study

LABS IN ACTION

Analysing our Case Study

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

The main file of note is

- **App.vue**

as we're applying our transitions globally to all components, but we could just as easily apply separate transitions to different components within each component's `<style>` and `<template>`


App.vue

```

<template>
  <div id="app">
    <b-navbar toggleable="md" variant="dark" type="dark"...>
      <!-- routes will be rendered here -->
      <transition name="fade" mode="out-in" v-on:after-enter="afterEnter" appear>
        <router-view />
      </transition>
    </div>
  </template>

```

<transition> (named "fade")



App.vue

```
.fade-enter-active, .fade-leave-active {  
  transition-duration: 3s;  
  transition-property: opacity;  
  transition-timing-function: ease;  
}
```

```
.fade-enter, .fade-leave-active {  
  opacity: 0  
}
```

```
.slide-in-enter-active, .slide-in-leave-active {  
  transition: transform 0.75s;  
}
```

```
.slide-in-enter, .slide-in-leave-to {  
  transform: translateX(-100%);  
}
```

```
.slide-in-enter-to, .slide-in-leave {  
  transform: translateX(0);  
}
```

<transition> classes (fade & slide-in)



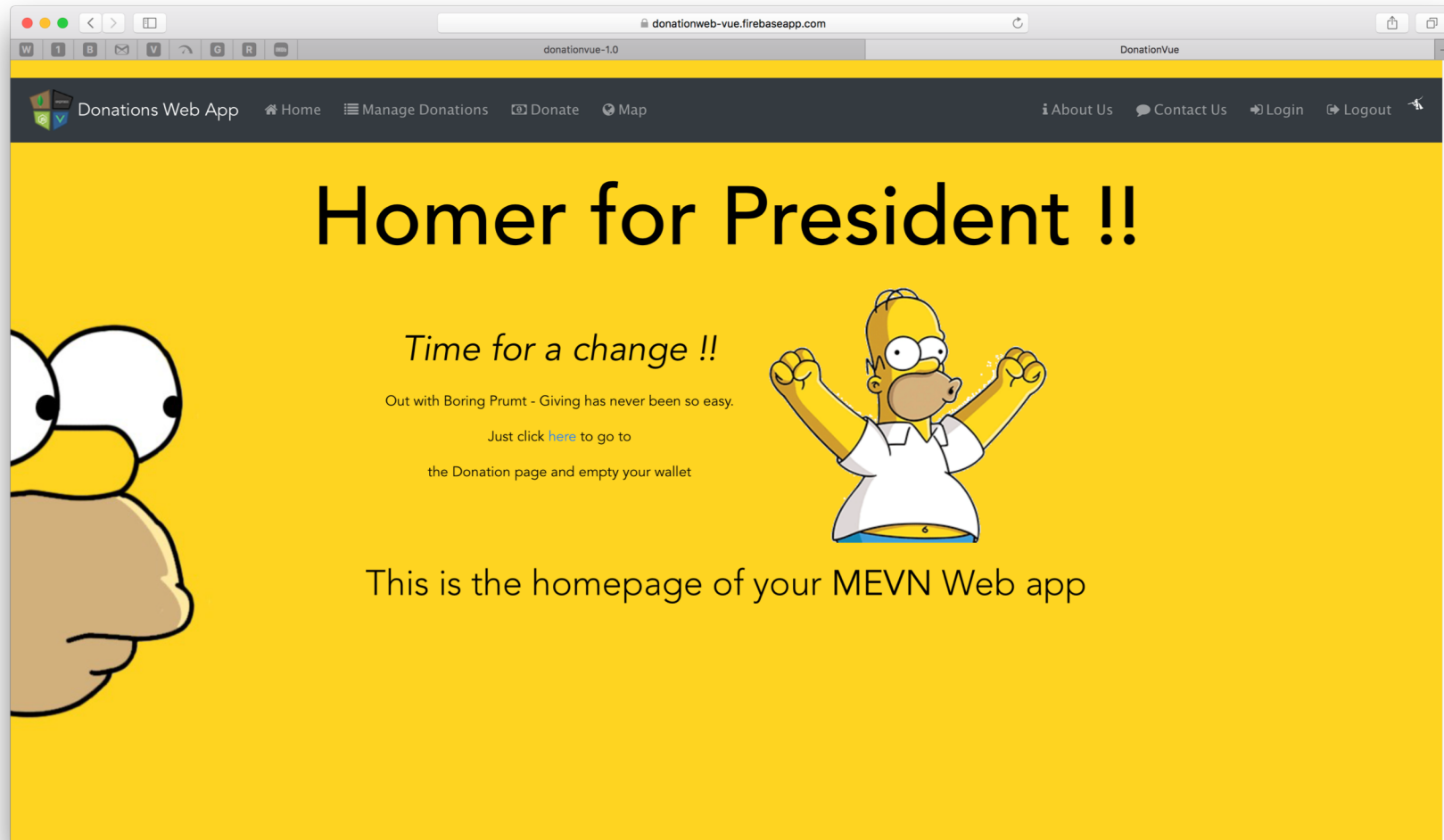
fade

<https://youtu.be/ad0TMggYpx0>

slide-in

<https://youtu.be/FRkHZJYkIE0>

Demo Application <https://donationweb-vue.firebaseio.com>



References

- ❑ <https://vuejs.org>
- ❑ <https://medium.com/vue-mastery/how-to-create-vue-js-transitions-6487dfd0baa>
- ❑ <https://scotch.io/tutorials/getting-started-with-component-transitions-in-vue>
- ❑ <https://alligator.io/vuejs/understanding-transitions/>

Questions?