VueJS

a simple Framework for simple people

MisterBit

The enter of the JS frameworks

- Web development has changed…
  - HTML is being used to build applications instead of documents
- For large scale projects, we just cannot get around with plain javascript or jquery any more
- We need a powerful framework to support all various aspects and life cycle of a web application
  - We had Angular1
  - React has improved some aspects
  - Angular 2 has entered the scene
  - But it was not enough…

Good frameworks bring along

- A solid foundation so we can focus on our unique challenge
- Good separation of concerns
- Benefits in making the app easier to extend, maintain, and test
- Mental model for where to put what

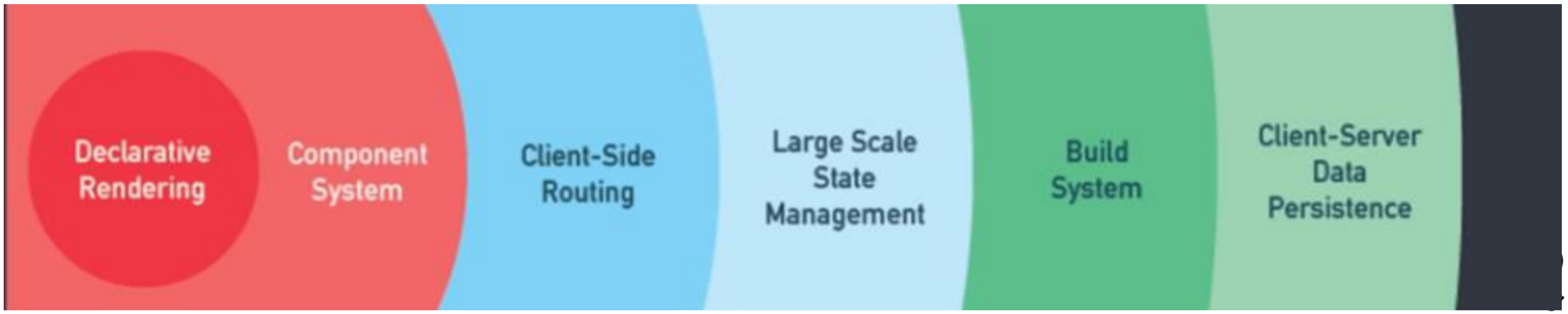
What’s VueJS

- progressive framework for building user interfaces.
- Easy to pick up and integrate with other libraries or existing projects.
- Capable of powering modern progressive web applications

  – Here is an Hello World fiddle.

Why VueJS

- Its amazingly simple
  - It has great documentation
- It can be effectively used to build simple to large applications
  - It is used by large companies such as Alibaba and Baidu
  - Its built in within Laravel PHP
  - Its small – 19KB! minified and gziped



Feature Rich

Computed Properties

Syncing

Events

Outputting Data

Developer Tools

Filters

Watchers

Components

Lists

Mixins

Forms

Conditionals

Reactivity

Directives

Dynamic Styles

Declarative Rendering

Render data to the DOM using template syntax:

```
<div id="app">
    <span>
        {{ msg}}
    </span>
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    msg: 'Hello Vue!' + Date.now()
  }
})
```

Handling Events

Use the v-on directive to attach event listeners that invoke methods on our Vue instances:

```
<div id="app">
   <span v-show="show">
            Hello {{userName}}
   </span>
   <button v-on:click="show = !show">Toggle</button>
</div>
```

Conditions & Loops

Toggle the presence of an element

```
<p v-if="seen">Now you see me</p>
<ol>
  <li v-for="pet in pets">
     {{ pet.name  }}
  </li>
</ol>
```

Two way data binding

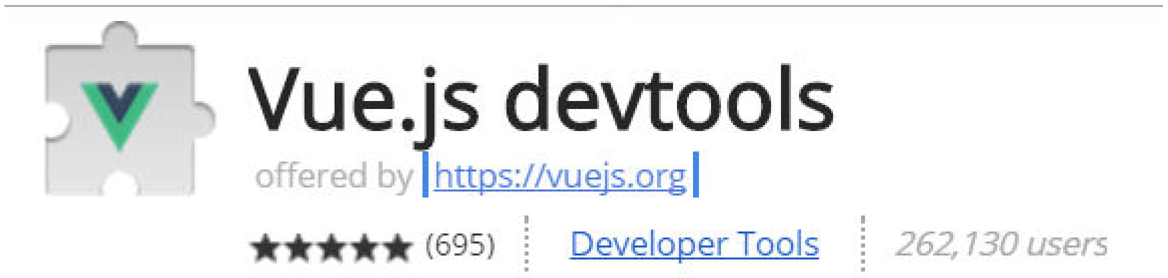
The v-model directive makes two-way binding between form input and our model

```
<p>{{ message }}</p>
<input v-model="message">
```

Vue devtools

Install vue-devtools extension

- See selected: $vm0



Vue.js devtools

offered by https://vuejs.org

★★★★★ (695) | Developer Tools | 262,130 users

interpolations & directives

- {{ (isValid)? msg + ‘!’ : ‘Err at’ + Date.now() }}
  - Only some white-listed globals are accessible
- **V-bind** for attributes:

  we cannot use interpolations inside attributes values
  (bad code: <a href="{{url}}">)
  - Instead, we use the **v-bind** directive:
  - <a v-bind:href = “book.purchaseUrl"
  - <img v-bind:title = “book.name“ v-bind:src = “book.imgUrl”

Shorthands

Vue.js provides special shorthand for two of the most often used directives, v-bind and v-on

```
<!-- full syntax -->
<a v-bind:href="url"></a>

<!-- shorthand -->
<a :href="url"></a>
```

```
<!-- full syntax -->
<a v-on:click="doSomething"></a>

<!-- shorthand -->
<a @click="doSomething"></a>
```

No worries - these chars are syntactically valid in HTML and they do not appear in the final rendered markup anyways.

Computed

Make our UI more declarative:
computed properties are cached
based on their reactive dependencies!

Instead of:
{{ msg.split('').reverse().join('') }}

```
<p> {{ msg }} </p>
<p> {{ reversedMsg }}  </p>
```

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

Binding HTML Classes

We can pass an object to *v-bind:class* to dynamically toggle classes:

```
<div class="static“  :class="{ active: isActive, 'text-danger': hasError }">
</div>
```

```
data: {
  isActive: true,
  hasError: false
}
```

Renders: `<div class="static active"></div>`

Binding HTML Classes – from data

The class object does not have to be inlined, so this reads better:

```
<div class="static” :class="classObject"></div>

data: {
  classObject: {
    active: true,
    'text-danger': false
  }
}
```

```
<div class="static active"></div>
```

(nice, but even better thing is coming next slide)

Binding HTML Classes – with computed

Best strategy is using a computed property:

```
<div v-bind:class="classObject"></div>
```

```
data: {
  isActive: true,
  error: null
},
computed: {
  classObject: function () {
    return {
      active: this.isActive && !this.error,
      'text-danger': this.error && this.error.type === 'fatal',
    }
   }
}
```

Play online

Binding HTML Classes – Array syntax

We can pass an array to *v-bind:class* to apply a list of classes:

```
<div v-bind:class="[activeClass, errorClass]">

<div v-bind:class="[isActive ? activeClass : ‘ ', errorClass]">

<div v-bind:class="[{ active: isActive }, errorClass]">
```

```
data: {
    isActive: true,
    activeClass: 'active',
    errorClass: 'text-danger'
}
```

Binding Inline Styles

for the CSS property names,
You can use either camelCase or kebab-case
(use quotes with kebab-case)

```
<div v-bind:style="{‘font-style’: fStyle, fontSize: fSize + 'px' }"></div>
```

```
data: {
  fStyle: ‘italic',
  fSize: 30
}
```

It is often a good idea to bind to a style object so that the template is cleaner, in most cases using Computed Prop will be best:

```
<div v-bind:style="styleObject"></div>
```

```
data: {
  styleObject: {
    color: 'red',
    fontSize: '13px'
  }
}
```

Play [here]()

Conditional Rendering v-if

Removing DOM is easy:

```
<h1 v-if="ok">Yes</h1>
<h1 v-else>No</h1>
```

Achieve Conditional Groups with v-if on <template>:

```
<template v-if="ok">
  <h1>Title</h1>
  <p>Paragraph</p>
</template>
```

```
<div v-if="type === 'A'">
A
</div>
<div v-else-if="type === 'B'">
B
</div>
<div v-else-if="type === 'C'">
C
</div>
<div v-else>
Not A/B/C
</div>
```

Conditional Rendering v-show

v-show simply toggles the display CSS property of the element

- Note that v-show **doesn’t** support the <template> syntax
- **Nor** does it work with v-else

```
<h1 v-show="ok">Hello!</h1
```

Conditional Rendering v-show

```
<section v-show="isActivated">…</section>

<section v-if    =" isActivated ">…</section>
```

Usage Considerations:

v-if has higher toggle costs while v-show has higher initial render costs.

- Prefer v-show if you need to toggle something very often
- prefer v-if if the condition is unlikely to change at runtime.

List Rendering v-for

We can use the v-for directive to render a list of items based on an array:

```
<li v-for=“(item, idx) in items">
    {{idx}} - {{ item.name }}
</li>
```

Similar to template v-if, you can also use a <template> tag with v-for to render a block of multiple elements

```
<template v-for="item in items">
  <li>{{ item.name }}</li>
  <li class="divider"></li>
</template>
```

List Rendering v-for

Here are some more examples:

```
<span v-for="n in 10">{{ n }}</span>

<my-component
        v-for="(item, index) in items"
        v-bind:item="item"
        v-bind:index="index">
</my-component>
```

v-for on Object

You can also use v-for to iterate through the properties of an object:

```
<li v-for="(value, key, idx) in person">
    {{ value }}
</li>

data: {
  person: {
    firstName: 'John',
    lastName: 'Doe',
    age: 30
  }
}
```

the key

It is recommended to provide a key with v-for whenever possible

```
<div v-for="item in items" :key="item.id">
<!-- content -->
</div>
```

This helps vuejs effectively correlate our model and dom, and optimize dom elements reuse

The *Key* is a generic mechanism for Vue to identify nodes, which has other uses that are not specifically tied to v-for.

Event handling

We use the v-on directive to listen to DOM events and run some JavaScript when they’re triggered.

```
<button @click="counter += 1">Add 1</button>
<p>The button above has been clicked {{ counter }} times.</p>
```

Usually, we will use methods:

```
<button @click="greet">Greet</button>

methods: {
  greet: function (event) {
     // `this` inside methods points to the Vue instance
    alert('Hello ' + this.name + '!')
    // `event` is the native DOM event
    alert(event.target.tagName)
   }
}
```

Event handling

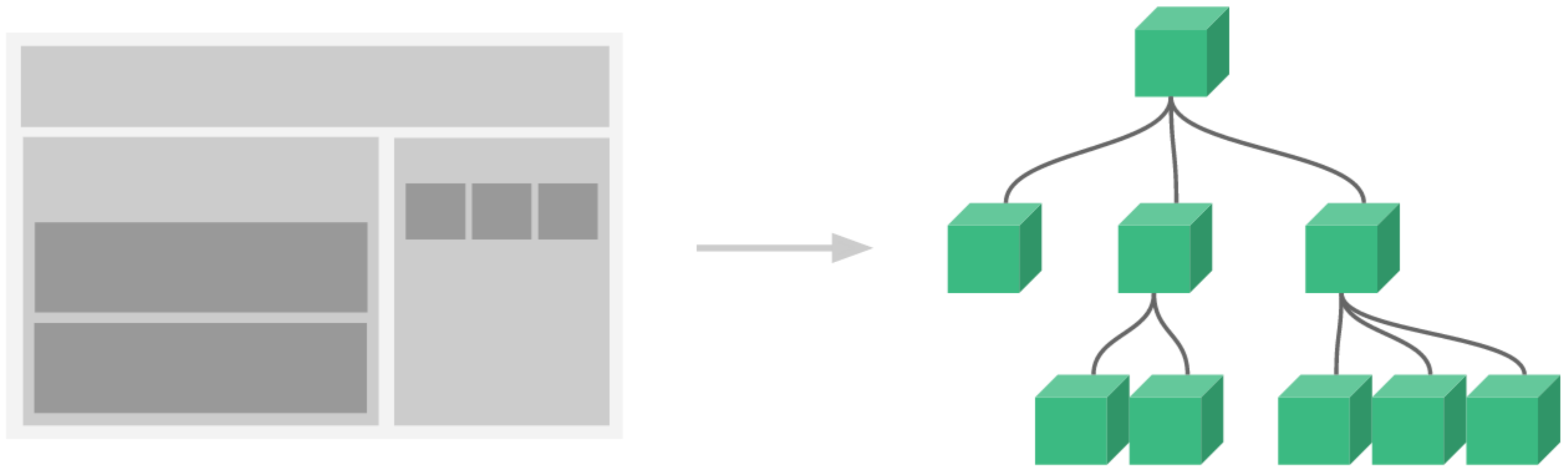
we can also use methods in an inline JavaScript statement:

```
<button @click="say('hi')">Say hi</button>
<button @click="warn(‘Sure?', $event)">Submit</button>

methods: {
  warn: function (message, event) {
     // now we have access to the native event
     if (!confirm(message)) event.preventDefault()
   }
```

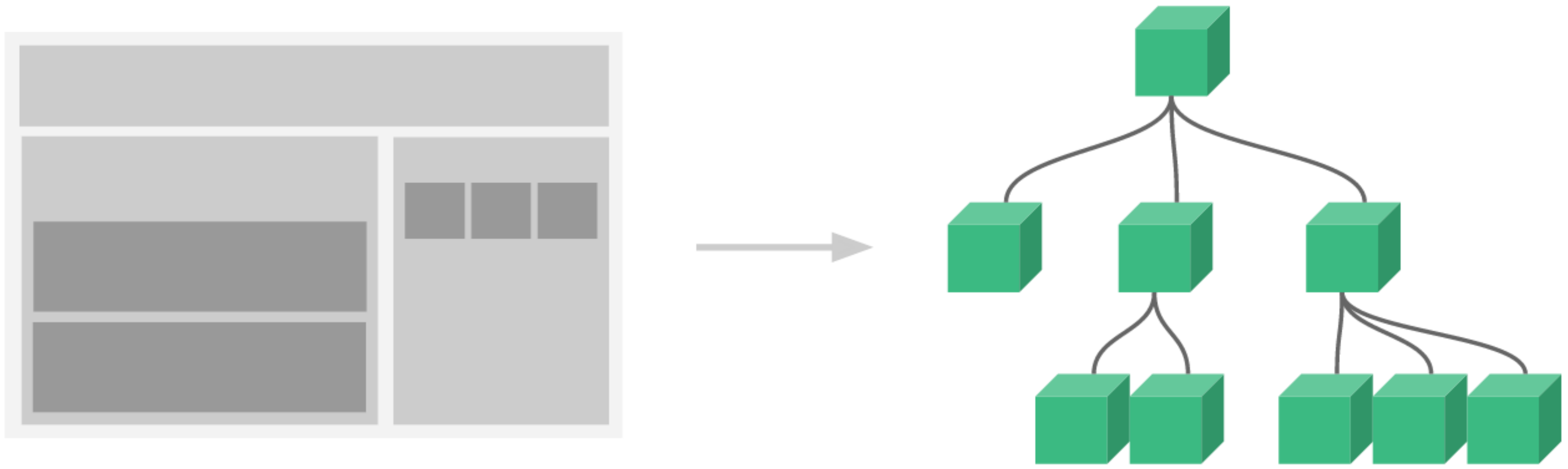
Note: passing ‘this’ from the template will pass the window (not useful)
We will later see that getting an access to DOM element (rarely needed) is done with REFs

Components



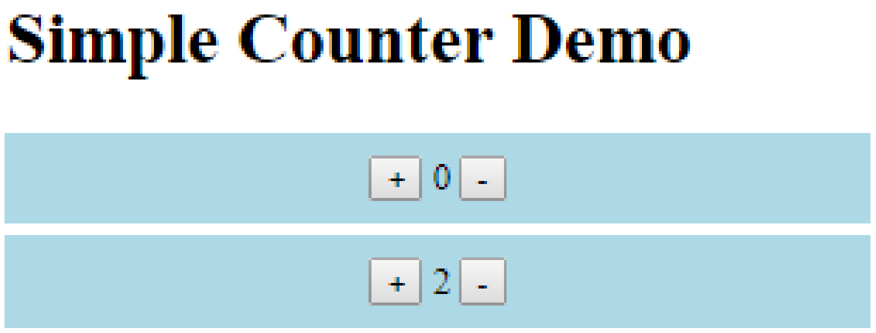
Components architecture

Building big applications from small, self-contained, reusable components.



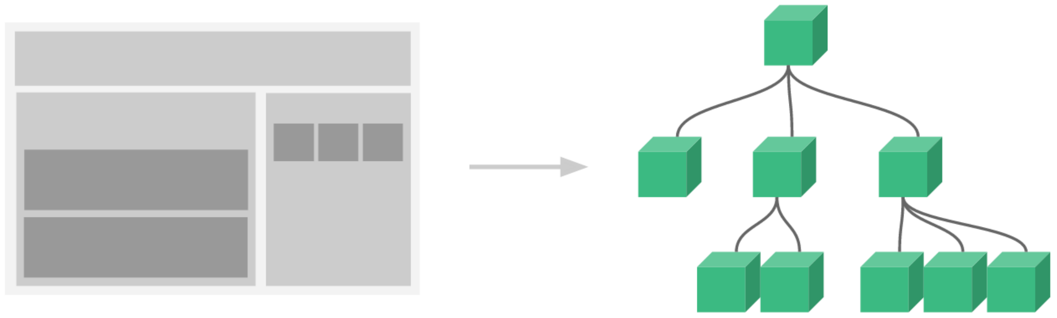
Components architecture

Example: simple-counter component


Simple Counter Demo

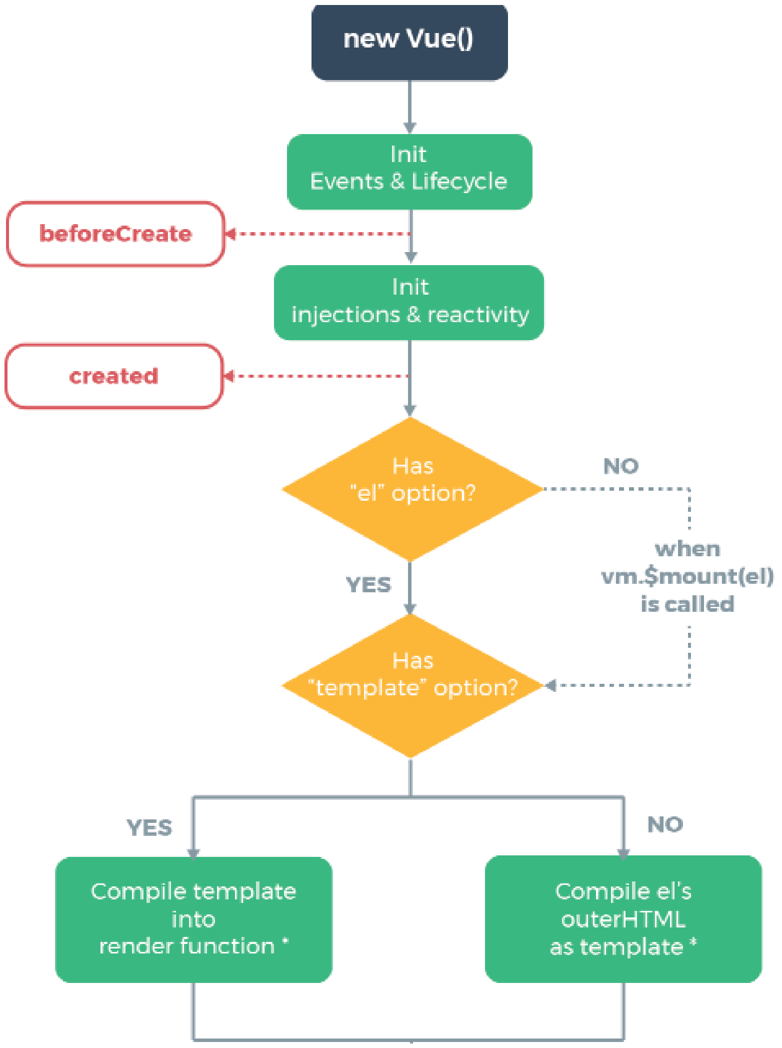
Composing with Components

The UI is broken to small pieces:



```
Vue.component('todo-item', {
   template: '<li>This is a todo</li>'
})
Vue.component('todo-list', {
   template: '<ul> <todo-item /> </ul>'
})
```

Instance lifecycle


new Vue()
Init Events & Lifecycle
beforeCreate
Init injections & reactivity
created
Has "el" option?
NO
when vm.$mount(el) is called
YES
Has "template" option?
YES
NO
Compile template into render function *
Compile el's outerHTML as template *

See it in action: https://jsfiddle.net/vyaron/jjL72zj4/

components - global

This is how we register a global component:

```
// register
Vue.component('my-component', {
  template: '<div>A custom component!</div>'
})
// create a root instance
new Vue({
  el: '#example'
})

<div id="example">
  <my-component></my-component>
</div>
```

WILL RENDER:

```
<div id="example">
  <div>A custom component!</div>
</div>
```

It is a good practice to adhere to the W3C rules for custom tag names
(tldr : all-lowercase, must contain a hyphen)

components – data()

Most of the options that can be passed into the Vue constructor can be used in a component, with one special case:
data **must be function.**

```
var Comp = {
  template: '<div>A custom component!</div>‘,
  data() {
    return {};
  }
}
```

See what happens If we cheat here

Props with binding

Similar to binding normal properties on native dom elements, we can also use **v-bind** for dynamically binding props of a component to data on the parent.

```
<input :value=“myPet.name">

<pet-play :pet=“myPet"></pet-play>



<!– Caveat: this passes down plain strings -->
<comp some-prop=“value"></comp>
<comp some-prop="1"></comp>
```

Passing Props

This is how:

```
 Vue.component('todo-item', {
   props: ['todo'],
   template: '<li>{{ todo.text }}</li>'
 })
```

```
<ol>
  <todo-item v-for="item in myList" v-bind:todo="item"></todo-item>
</ol>
```

Using v-on with Custom Events

Every Vue instance implements an events interface, which means it can:

- Listen to an event using $on(eventName)
- Trigger an event using $emit(eventName)

```
<div id="app">
<p>{{ total }}</p>
<button-counter @increment="incrementTotal"></button-counter>
<button-counter @increment="incrementTotal"></button-counter>
</div>
```

See full demo here

Composing components

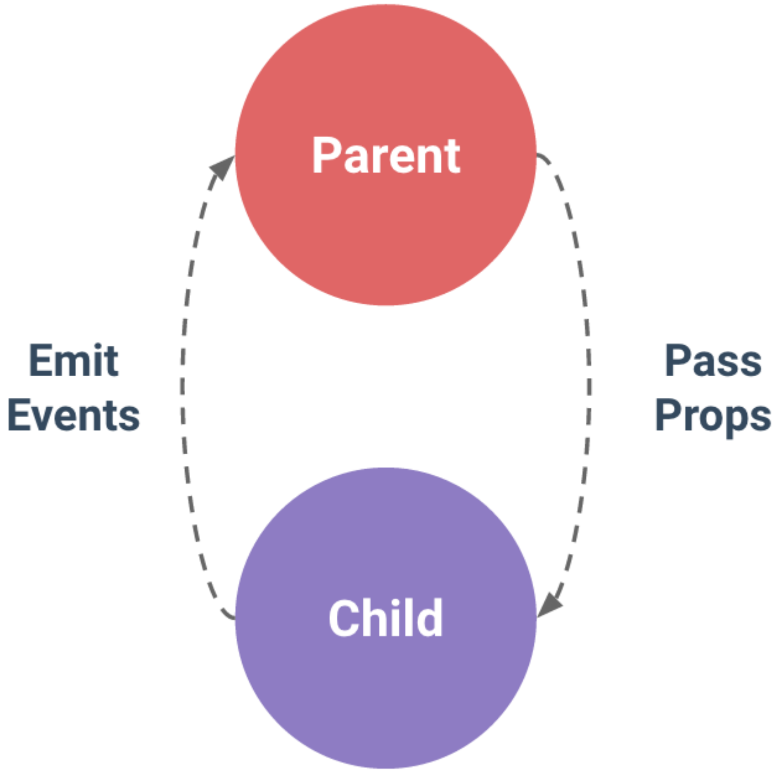
Components are meant to be used together, most commonly in parent-child relationships.

- It is important to keep the parent and the child as **decoupled** as possible via a **clearly-defined interface**.
- This will make them more maintainable and potentially easier to reuse
- This is achieved via: **props down, events up**

```
Vue.component('child', {
  props: ['message'],
  template: '<span>{{ message }}</span>'
})
```

Using it:

```
<child message="hello!"></child>
```



VueJS

Forms

MisterBit

V-model them all

Building forms is possible using the v-model directive that creates two-way data bindings:

```
<p style="white-space: pre">{{ message }}</p>
<textarea v-model="message" placeholder="add multiple lines"></textarea>

<input type="checkbox" value="Jack" v-model="checkedNames“ />  Jack
<input type="checkbox" value="John" v-model="checkedNames“ /> John

<input type="radio" value="One" v-model="picked"> One
<input type="radio" value="Two" v-model="picked"> Two

<select v-model="selected" multiple>
<option>A</option>
<option>B</option>
<option>C</option>
</select>

<select v-model="selected">
<!-- inline object literal -->
<option v-bind:value="{ number: 123 }">Something</option>
</select>

<button type=“submit" :disabled=“isValid”>Save</button>
```

v-model modifiers

the v-model directive to create two-way data bindings:

```
<!-- synced after "change" instead of "input" -->
<input v-model.lazy="msg" >

<!-- typecast as a number -->
<input v-model.number="age" type="number">

<input v-model.trim="msg">
```

VueJS

More About Event Handling

You click me, guys !!!

Event handling - Event Modifiers

These are event modifiers used with v-on:

- **.stop**
- **.prevent**
- .capture
- .self
- .once

Most are exclusive to native DOM events (.once is not)

```
<!-- the click event's propagation will be stopped -->
<a v-on:click.stop="doThis"></a>

<!-- the submit event will no longer reload the page -->
<form v-on:submit.prevent="onSubmit"></form>

<!-- modifiers can be chained -->
<a v-on:click.stop.prevent="doThat"></a>

<!-- just the modifier -->
<form v-on:submit.prevent></form>

<!-- use capture mode when adding the event listener -->
<div v-on:click.capture="doThis">...</div>

<!-- only trigger handler if event.target is the element itself -->
<!-- i.e. not from a child element -->
<div v-on:click.self="doThat">...</div>

<!-- the click event will be triggered at most once -->
<a v-on:click.once="doThis"></a>
```

Event handling - Key Modifiers

Key modifiers for v-on when listening for key events:

- .delete (captures both “Delete” and “Backspace” keys)
- .enter
- .tab
- .esc
- .space
- .up
- .down
- .left
- .right

```
<!-- only call vm.submit() when the keyCode is 13 -->
<input v-on:keyup.13="submit">

<!-- same as above -->
<input v-on:keyup.enter="submit">

<!-- also works for shorthand -->
<input @keyup.enter="submit">
```

You can also define custom key modifier aliases via the global config.keyCodes object:

```
// enable v-on:keyup.f1
Vue.config.keyCodes.f1 = 112
```

Event handling – Modifiers Keys

You can use the following modifiers to trigger mouse or keyboard event listeners only when the corresponding modifier key is pressed:

- .ctrl
- .alt
- .shift
- .meta (⌘ **Mac / ⊞ windows, etc)**

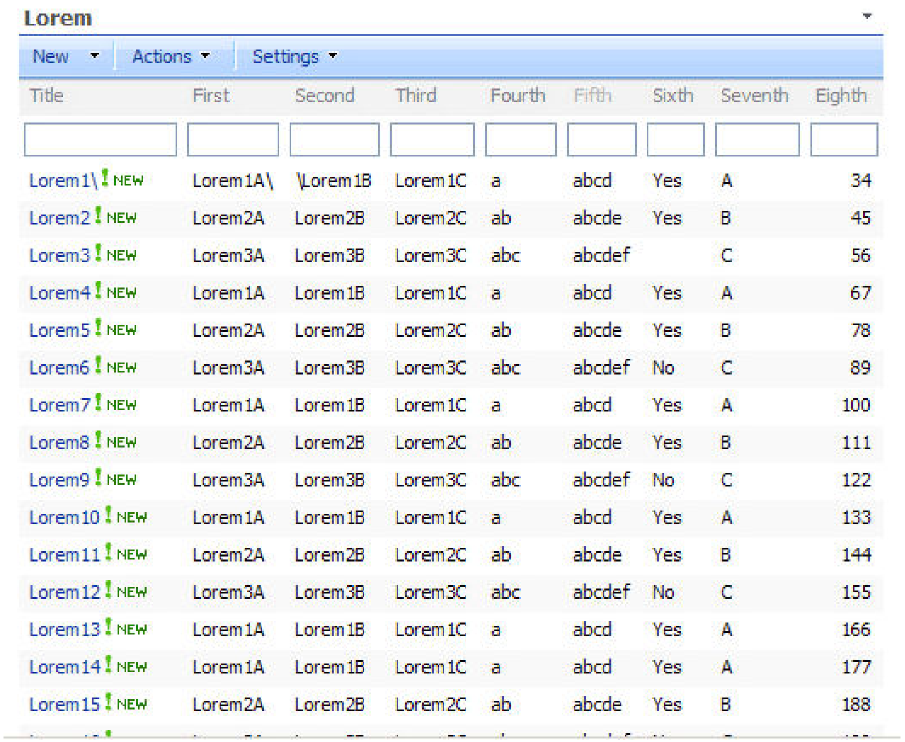
```
<!-- Alt + C -->
<input @keyup.alt.67="clear">

<!-- Ctrl + Click -->
<div @click.ctrl="doSomething">Do something</div>
```

Play here

Displaying Filtered/Sorted Results

- Its common to display a filtered or sorted version of an array without actually mutating or resetting the original data.
- Use a computed property that returns the filtered or sorted array.



Title	First	Second	Third	Fourth	Fifth	Sixth	Seventh	Eighth
Lorem1\ NEW	Lorem1A\	\Lorem1B	Lorem1C	a	abcd	Yes	A	34
Lorem2 NEW	Lorem2A	Lorem2B	Lorem2C	ab	abcde	Yes	B	45
Lorem3 NEW	Lorem3A	Lorem3B	Lorem3C	abc	abcdef		C	56
Lorem4 NEW	Lorem1A	Lorem1B	Lorem1C	a	abcd	Yes	A	67
Lorem5 NEW	Lorem2A	Lorem2B	Lorem2C	ab	abcde	Yes	B	78
Lorem6 NEW	Lorem3A	Lorem3B	Lorem3C	abc	abcdef	No	C	89
Lorem7 NEW	Lorem1A	Lorem1B	Lorem1C	a	abcd	Yes	A	100
Lorem8 NEW	Lorem2A	Lorem2B	Lorem2C	ab	abcde	Yes	B	111
Lorem9 NEW	Lorem3A	Lorem3B	Lorem3C	abc	abcdef	No	C	122
Lorem10 NEW	Lorem1A	Lorem1B	Lorem1C	a	abcd	Yes	A	133
Lorem11 NEW	Lorem2A	Lorem2B	Lorem2C	ab	abcde	Yes	B	144
Lorem12 NEW	Lorem3A	Lorem3B	Lorem3C	abc	abcdef	No	C	155
Lorem13 NEW	Lorem1A	Lorem1B	Lorem1C	a	abcd	Yes	A	166
Lorem14 NEW	Lorem1A	Lorem1B	Lorem1C	a	abcd	Yes	A	177
Lorem15 NEW	Lorem2A	Lorem2B	Lorem2C	ab	abcde	Yes	B	188

components - local

It is common to make a component available only in the scope of another instance/component:

```
var Child = {
  template: '<div>A custom component!</div>'
}
new Vue({
  // ...
  components: {
    // <my-component> will only be available in parent's template
    'my-component': Child
  }
})
```

Component Naming Conventions

When registering components (or props), you can use kebab-case, camelCase, or TitleCase. Vue doesn’t care.

```
// in a component definition
components: {
// register using kebab-case
'kebab-cased-component': { /* ... */ },
// register using camelCase
'camelCasedComponent': { /* ... */ },
// register using TitleCase
'TitleCasedComponent': { /* ... */ }
}
```

Within HTML templates though, you have to use the kebab-case equivalents:

```
<!-- alway use kebab-case in HTML templates -->
<kebab-cased-component></kebab-cased-component>
<camel-cased-component></camel-cased-component>
<title-cased-component></title-cased-component>
```

Mutating Props

Sometimes, it’s tempting to try and mutate a prop:

1. The prop is used to pass in an **initial value**, the child component simply wants to use it as a local data property afterwards;
2. The prop is passed in as a **raw value** that needs to be transformed.

The proper tactics for these use cases are:

```
props: ['initialCounter‘, ‘size’],
data: function () {
    return { counter: this.initialCounter }
}

computed: {
   normalizedSize: function () {
     return this.size.trim().toLowerCase()
   }
}
```

Caveat: Native Events on Custom Elements

There may be times when you want to listen for a native event on the root element of a component.

In these cases, you can use the .native modifier for v-on. For example:

```
<my-component @click.native="doTheThing"></my-component>
```

Exercise

- Create a new project – eventos
- Build a service - EventosService
  - The entity should be called evento {id, name}
    (to avoid confusion with dom events)
  - This service has a method *query* that returns a promise for a list of eventos
- In your root component, render the list, if empty show msg: “No events’
- Allow adding an event

Non Parent-Child Communication

- Sometimes two components may need to communicate with one-another but they are not parent/child to each other.
- We can use an empty Vue instance as a **central event bus**:

```
var bus = new Vue()
// in component A's method
bus.$emit('id-selected', 1)

// in component B's created hook
bus.$on('id-selected', function (id) {
// ...
})
```

In more complex cases, you should consider employing a dedicated state-management pattern (Vuex - Discussed later)

VueJS

Routing

Going different places

MisterBit

Routing in a VueJS Application

- download vue-router
- main.js:
  - import routes from './routes.js‘
  - Vue.use(VueRouter);
  - const router = new VueRouter({routes})
  - add router to root Vue
- routes.js:
  - export const routes = [{path: '/car', component: CarDetails}]
- App.js
  - <router-link>
  - <router-view>

- Routing Modes (Hash vs History)
  const router = new VueRouter({routes, mode: 'history'})

Routing in a VueJS Application

- Navigating from Code (Imperative Navigation):
  this.$router.push('/');
- Setting Up Route Parameters
  Add route: /car/:id
- Fetching and Using Route Parameters
  data(){return {id: this.$route.params.id}}

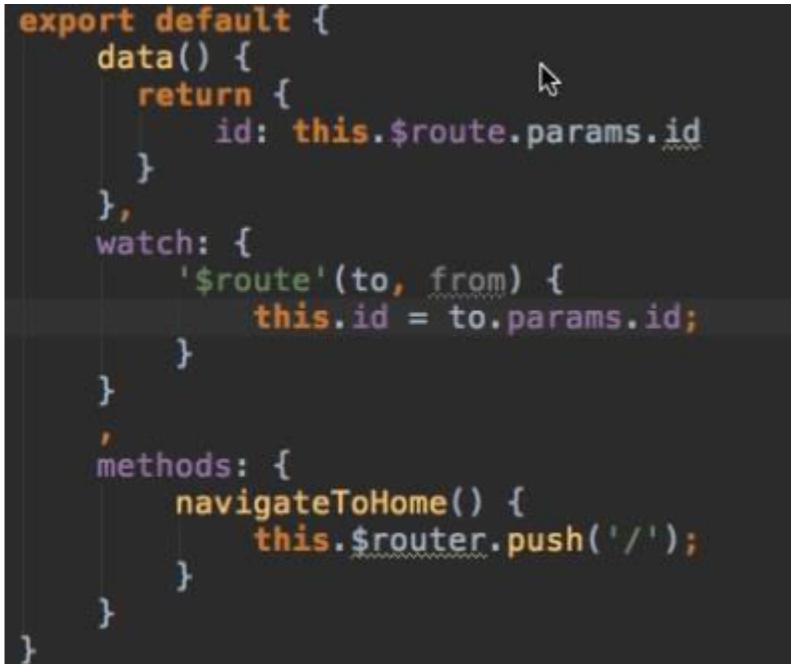
Routing in a VueJS Application

- Navigating with Router Links
  - Instead of <a> use: <router-link to='/car'>
  - Note it renders an <a> eventually
- Styling Active Links
  - .router-link-active {color: green}

  - ITP: Show a bootstrap nav component as a demo for: tag="li" active-class="active" exact

Routing in a VueJS Application

- Navigating from Code (Imperative Navigation):
  this.$router.push('/');
- Setting Up Route Parameters
  Add route: /car/:id
- Fetching and Using Route Parameters
  data(){return {id: this.$route.params.id}}
- Reacting to Changes
  in Route Parameters:



```
export default {
    data() {
      return {
          id: this.$route.params.id
      }
    },
    watch: {
        '$route'(to, from) {
            this.id = to.params.id;
        }
    }
    ,
    methods: {
        navigateToHome() {
            this.$router.push('/');
        }
    }
}
```

Animations

Vue provides a variety of ways to apply transition effects when items are inserted, updated, or removed from the DOM.
This includes tools to:

- Automatically apply classes for CSS transitions and animations
- integrate 3rd-party CSS animation libraries, such as Animate.css
- use JavaScript to directly manipulate the DOM during transition hooks
- integrate 3rd-party JavaScript animation libraries, such as Velocity.js
- Here is a simple example
- Full Power here

Using Refs

- Despite the existence of props and events, sometimes you might still need to directly access a child element in JavaScript.
- Example:

```
<div id="parent">
<div id="map" ref="theMap"></div>
</div>

var elMap = this.$refs.theMap
```

Using Refs

Another Example - you need to set the focus to an input, when some button is clicked:

```
this.$refs['input'].focus()
```

Don’t over use refs, try to relay on your model whenever possible

VueJS

Components

Dynamic Components

MisterBit

Dynamic Components

We can use the same mount point and dynamically switch between multiple components using the reserved <component> element and dynamically bind to its is attribute:

```
var vm = new Vue({
  el: '#example',
  data: {
    currentView: 'home'
  },
  components: {
    home: { /* ... */ },
    posts: { /* ... */ },
    archive: { /* ... */ }
  }
})
```

```
<component v-bind:is="currentView">
<!-- component changes when vm.currentView changes! -->
</component>
```

MisterBit

Dynamic Components

```
<component :is="currentView">
<!-- component changes when vm.currentView changes! -->
</component>
```

Dynamic Component has 2 extra Lifecycle Hooks: activated(), dectivated()

Keep Alive

If you want to keep the **switched-out components** in memory so that you can preserve their state or avoid re-rendering, you can wrap a dynamic component in a <keep-alive> element:

```
<keep-alive>
    <component :is="currentView">
        <!-- inactive components will be cached! -->
    </component>
</keep-alive>
```

Watchers

- there are times when a custom watcher is necessary
  - When needed, there is also an imperative API
- This is most useful when you want to perform asynchronous or expensive operations in response to changing data

```
Ask a yes/no question:
<input v-model="question">
<p>{{ answer }}</p>
```

```
watch: {
  question: function (newQuestion) {
     this.answer = 'Finding your answer...'
     this.getAnswer().then(ans => this.answer = ans)
  }
}
```

https://jsfiddle.net/vyaron/1ek3d48u/3/