VUE.JS – A PRACTICAL INTRODUCTION

HENRI TUKIAINEN

20.9.2019 LILLA ROBERTS

THIS PRESENTATION

- Slides + live code samples
- Maybe a coding exercise
- Ask questions when you have them
- Checklist
    - Can everyone hear me?
    - Presentation font size OK?
    - Code editor font size OK? →

```
console.log('Can you read this?')
```

VUE

- Pronounced */vjuː/*
- JavaScript framework for building user interfaces and single-page applications
- Created by Evan You, first release in 2014
- ☆ 148k on GitHub (React: 136k)

ME

- Henri Tukiainen
- Co-Founder, software consultant at Momocode Ltd.
- Full-stack web developer
- Vue since 2016

YOU

- Who's familiar with...
  - React?
  - AngularJS? Angular2+?
  - ES6+?
  - Webpack? Babel?

AGENDA

- 09-12 **Part 1**: Vue principles, concepts and features
- 12-13 Lunch
- 13-16 **Part 2**: SPA development with Vue
    - Coffee break at 14

VUE

WHAT DO VUE APPS LOOK LIKE?

```
<div id="app">
  Hello {{ name }}!
</div>

<script>
  var app = new Vue({
    data: {
      name: 'Anonymous'
    }
  }).$mount('#app')
</script>
```

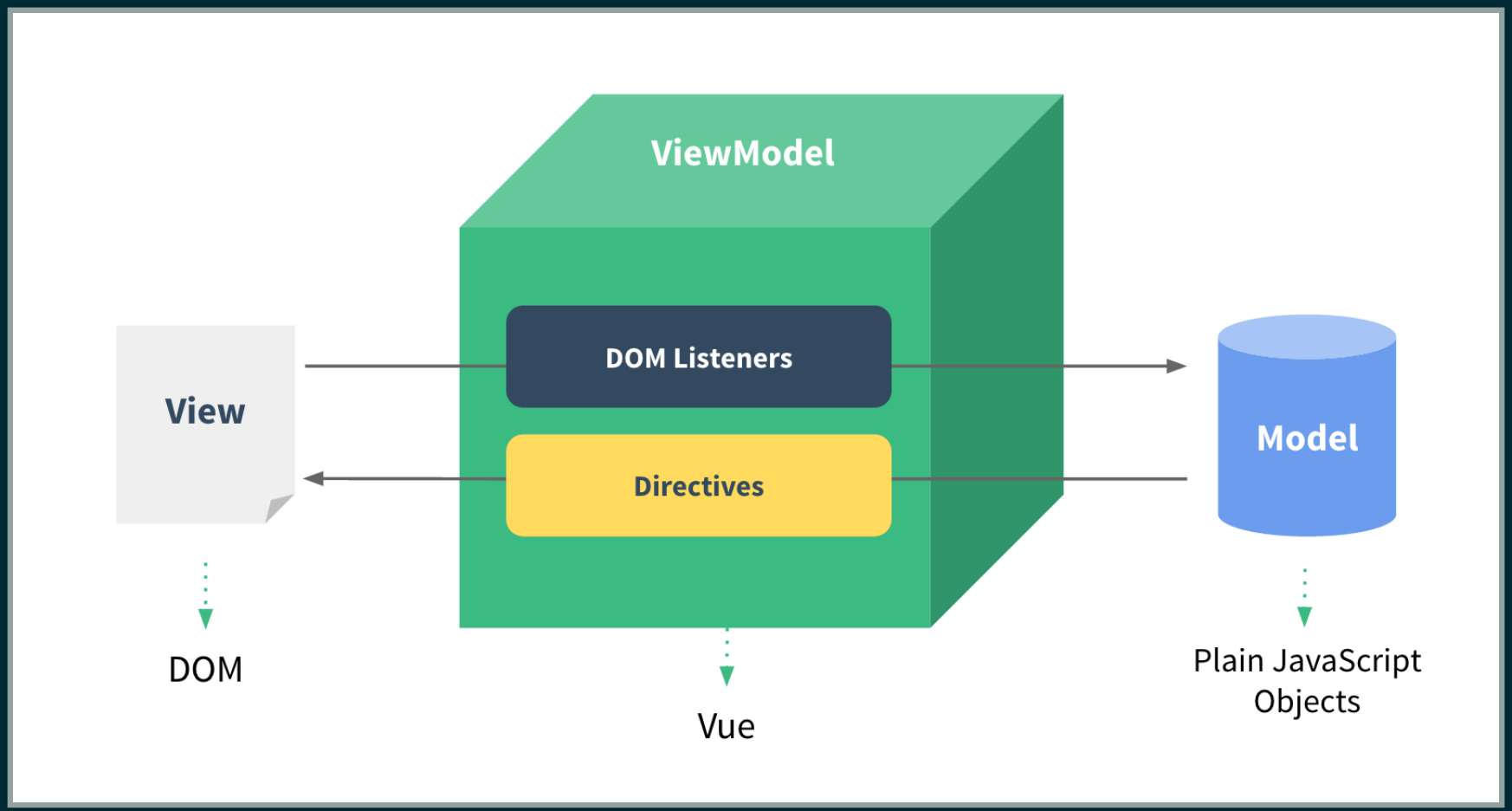
CORE PRINCIPLES

- Progressive
  - Incrementally adoptable into existing applications
- Approachable
  - Low entry barrier, gradual learning curve
- Versatile
  - Scalable between simple library and full featured framework
- Performant
  - Small footprint, fast performance

> *"I figured, what if I could just extract the part that I really liked about Angular and build something really lightweight."*
>
> *– Evan You*

ARCHITECTURE

- Model–view–viewmodel (MVVM) inspired architecture
  - **ViewModel**: Object that syncs the Model and the View = Vue instance
  - **View**: DOM managed by Vue instances
  - **Model**: Plain JavaScript object made reactive by Vue.js



OTHER DETAILS

- Browser compatibility: IE9+
- 22.8 kB minified & gzipped (React: 37.3 KB)

VERSIONS

- 1.0 - 27.10.2015
- 2.0 - 30.9.2016
  - 2.6 - 4.2.2019
- 3.0
  - Announced 15.11.2018
  - Expected release 2020

CONCEPTS AND FEATURES OVERVIEW

VUE INSTANCE

A Vue application starts by creating a Vue instance:

```
var app = new Vue({
  // options
})
```

Vue applications consist of a root Vue instance and components – which are also Vue instances.

```
Root Instance
└─ Component A
   ├─ Component B
   │  ├─ Component D
   │  └─ Component E
   └─ Component C
      ├─ Component D
      └─ Component F
```

DECLARATIVE RENDERING 1/2

Data is declared in the Vue instance and bound to the DOM when used in a template

```
<div>{{ message }}</div>
```

```
var app = new Vue({
  data: {
    message: 'Hello Vue!'
  }
})
```

Hello Vue!

DECLARATIVE RENDERING 2/2

You can also bind attributes, and use JavaScript expressions

```
<div v-bind:title="tooltip">{{ message.split('').reverse().join('') }}</div>
```

```
var app = new Vue({
  data: {
    tooltip: 'This is the message backwards',
    message: 'Hello Vue!'
  }
})
```

```
!euV olleH
```

REACTIVITY

Properties in a Vue instance's `data` object are added to Vue's reactivity system. When the data changes, the view is re-rendered.

```
<span>The number is {{ number }}</span>
```

```
var app = new Vue({
  data: {
    number: 1
  }
})
app.number = 2 // View is immediately updated
```

DIRECTIVES

Attributes starting with `v-` are directives that tell Vue how to render the DOM.

Directives take care of things like

- Data binding
- Rendering text or HTML
- Conditionals
- Loops
- Forms and input
- Events

COMPONENTS

Components are Vue instances that can be used as custom elements in templates.

```
<hello-world name="Vue"></hello-world>
```

```
Vue.component('hello-world', {
  template: '<div>Hello {{ name }}!</div>'
})
```

```
Hello Vue!
```

LET'S CODE →

THOUGHTS ON DEMO APP

Pros:

- Writing a working app is easy
- Basic concepts are easy to understand
- We can do quite a lot with just HTML and vanilla JS

Cons:

- Organizing code is difficult
- String templates are inconvenient to write and maintain
- Using vanilla JS is inconvenient
- HTML restrictions are inconvenient
- We only had one page, not a proper SPA
- Managing state across components in the app is difficult

CONCEPTS AND FEATURES

VUE INSTANCE OPTIONS

DATA 1/2

`data` is a plain object or function that returns a plain object. Must be a function for components.

```
var vm = new Vue({
  data: function () {
    return {
      some: 'data'
    }
  }
})
console.log(vm.some) // Prints 'data'
```

Vue recursively adds getters and setters for all properties, making the data object reactive. The reactive properties are made available in the Vue instance.

DATA 2/2

- Properties starting with `_` or `$` are reserved for Vue and are not made available in the instance (`vm.$data._something` works though)
- You cannot add new properties directly to the instance

```
var vm = new Vue({
  data: function () {
    return {
      some: 'data'
    }
  }
})
vm.someOther = 'data' // Does NOT work!
```

PROPS

`props` is an array or object defining the data received from parent components. Vue makes the props available as reactive properties in the Vue instance.

```
new Vue({
  // Array syntax makes all props not required, not validated and without default values
  props: ['someProp', 'anotherProp']
})
```

```
new Vue({
  props: {
    // Object syntax allows specifying more details
    someProp: {
      type: String,
      default: 'Some value',
      required: false,
      validator: function (value) {
        return value.length > 3
      }
    }
  }
})
```

Type can be `String`, `Number`, `Boolean`, `Array`, `Object`, `Date`, `Function`, `Symbol`, or custom constructor, or array of these.

COMPUTED 1/2

`computed` properties are added to the Vue instance. The properties are reactive for the reactive properties used in the computation. Computed properties can access the Vue instance via `this`.

```
new Vue({
  props: ['exponent'],
  data: {
    base: 5
  },
  computed: {
    power: function () {
      return Math.pow(this.base, this.exponent)
    }
  }
})
```

COMPUTED 2/2

- Computed properties are cached and only re-computed when a referenced reactive property changes.
- You must only use reactive properties or constants, never non-reactive values.

```
new Vue({
  data: {
    factor: 0.5
  },
  computed: {
    partOfScreenWidth: function () {
      return window.innerWidth * this.factor // Does NOT work!
    }
  }
})
```

- Computed properties must never cause side effects.
- Use computed properties instead of methods when possible, to take advantage of caching.

METHODS

`methods` are functions to be added to the Vue instance, which can also be accessed from templates. All methods can access the Vue instance via `this`.

```
new Vue({
  data: {
    currency: '$'
  },
  methods: {
    formatCurrency: function (amount) {
      return this.currency + Number.parseFloat(number).toFixed(2) // E.g. '$123.45'
    }
  }
})
```

- Don't use arrow functions as methods, or `this` will not be bound.
- Prefer computed properties when possible, as method results are not cached.

WATCH 1/2

`watch` is a map of functions to be called when a watched property changes. Watch functions have access to Vue instance via `this`.

```
new Vue({
  data: {
    someProperty: 'Some value'
  },
  watch: {
    someProperty: function (value, oldValue) {
      console.log('Changed from ' + oldValue + ' to ' + value)
    }
  }
})
```

WATCH 2/2

- More options:

```
new Vue({
  watch: {
    // Method as handler
    prop1: 'someHandler',
    // Multiple handlers
    prop2: ['firstHandler', 'secondHandler'],
    // Watch nested value
    'obj1.prop1': 'someHandler'
    // Additional options
    obj1: {
      // Call handler with initial value
      immediate: true,
      // Watch changes in properties of object
      deep: true,
      handler: 'someHandler'
    }
  }
})
```

- You can create watchers dynamically with

```
var unwatch = vm.$watch('prop', callback, options)
```

- Don't use arrow functions as handlers, or `this` will not be bound.

LIFECYCLE HOOKS

Lifecycle hooks are functions to be called when Vue instance lifecycle phase changes. All lifecycle hooks can access the Vue instance via `this`.

```
new Vue({
  // ...
  mounted () {
    this.getDataFromApi()
  }
})
```

- Possible hooks: `beforeCreate`, `created`, `beforeMount`, `mounted`, `beforeUpdate`, `updated`, `activated`, `deactivated`, `beforeDestroy`, `destroyed`, `errorCaptured`
- Don't use arrow functions as methods, or `this` will not be bound.

MIXINS AND EXTENDS

`mixins` provide reusable functionality that is merged with a Vue instance's own functionality.

```
// Both methods will be available on the instance.
var mixin = {
  methods: {
    doSomething: function () {}
  }
}
new Vue({
  mixins: [mixin],
  methods: {
    doSomethingElse: function () {}
  }
})
```

Extends adds functionality from another component.

```
var componentA = {
  // Options
}
var componentB = {
  extends: componentA
  // More options
}
```

PROVIDE / INJECT

A component can `provide` values to all child components in the hierarchy that `inject` a value.

```
var parentComponent = {
  provide: {
    someProperty: 'value'
  }
}
var childComponent = {
  inject: ['someProperty'],
  created () {
    console.log(this.someProperty) // The injected value is available
  }
}
```

This should be used sparingly to avoid coupling components together.

MORE INSTANCE PROPERTIES

Covered elsewhere:

- `el`: DOM element or selector to automatically mount
- `components`: custom components
- `template`: string template to replace mounted element
- `render`: render function, alternative to template

EVEN MORE INSTANCE PROPERTIES

Not covered:

- `directives`: custom directives
- `filters`: custom filters
- `parent`: make parent instance available to child instance
- `functional`: make component functional (stateless and instanceless)
- `model`: Custom `v-model` property and event names

DIRECTIVES

BINDING TEXT AND HTML CONTENT

- `v-text` is similar to `{{ }}`, but replaces the entire `textContent` of an element
- `v-html` replaces the `innerHTML` content of an element
  - Remember to sanitize the values to avoid XSS attacks
  - Only regular HTML elements are supported, not components

```
<div v-text="plainMessage"></div>
<div v-html="htmlMessage"></div>
```

```
new Vue({
  data: {
    plainMessage: 'Hello Vue!',
    htmlMessage: 'Hello <b>Vue</b>!'
  }
})
```

Hello Vue!
Hello **Vue**!

BINDING ATTRIBUTES AND PROPS

- `v-bind`: binds one or more attribute values or component properties to an expression
- Use with an argument to bind specific attribute, or with an object to bind all properties of the object

```
<div v-bind:title="tooltip">This element has the title attribute</div>

<!-- This component receives props matching all they keys in someObject -->
<some-component v-bind="someObject"></some-component>

<div :title="tooltip">There is also a shorthand</div>
```

BINDING CLASSES AND STYLES

- `v-bind` supports additional syntax when used with `class` or `style` attributes
- Style bindings are automatically vendor-prefixed

```
<div v-bind:class="{ bold: isTextBold }"></div>
<div v-bind:class="[ 'firstClass', 'secondClass' ]"></div>
<div v-bind:class="isTextBold && 'bold'"></div>

<div v-bind:style="{ fontWeight: 'bold' }"></div>
<div v-bind:style="[{ fontWeight: 'bold' }, { textAlign: 'center' }]"></div>
<div v-bind:style="{ fontWeight: isTextBold && 'bold' }"></div>
```

CONDITIONALS

- `v-if`, `v-else-if`, `v-else`: Removes and adds elements to/from the DOM depending on the values
  - If you want to toggle multiple elements at once, wrap them in a `<template>` to avoid unnecessary wrapper elements
- `v-show`: sets the CSS `display` property to `none` when evaluated to false

```
<div>
  <div v-if="loading">Still loading...</div>
  <template v-else>
    <p>Loading done!</p>
    <p>The data is: {{ data }}</p>
  </template>
</div>
```

LOOPS

- `v-for`: renders an element multiple times based on an array or object
  - You should provide `key` attribute so Vue knows which items are the same when data changes
  - Note DOM template restrictions when using custom components

```
<ul>
  <!-- Loops items in an array -->
  <li v-for="(item, index) in array" v-bind:key="item.id">
    {{ item.title }}
  </li>
  <!-- Loops 10 times -->
  <li v-for="n in 10">
    {{ n }}
  </li>
  <!-- is attribute is needed because only li is valid inside ul -->
  <li
    is="some-component"
    v-for="item in array"
    v-bind:key="item.id"
    v-bind="item"
  ></li>
</ul>
<div v-for="(value, key) in object">
  {{ value }}
</div>
```

EVENTS

- `v-on` calls specified callback when specified event is fired
- With a HTML element, listens to native DOM events
- With a component, listens to custom events

```
<button v-on:click="console.log($event)">Click me</button>
<form v-on:submit.prevent="handleSubmit"></form>
<button @click="handleClick">Shorthand</button>
```

- Many modifiers are also supported: `.stop`, `.prevent`, `.once` etc.

FORMS

- `v-model` creates a two-way binding on input element or custom component
- Supported modifiers: `.lazy`, `.number`, `.trim`

```
<input type="text" v-model="firstName">
<input type="number" v-model.number="age">
```

MORE DIRECTIVES

- `v-cloak`: removed when Vue instance has initialized, can be used to hide uncompiled HTML during initialization
- `v-pre`: element is not compiled, used for optimization
- `v-once`: element or component is rendered only once, i.e. does not update reactively, used for optimization
- `v-slot`: used for passing content from parent to child component

```
<div v-cloak>The data is {{ data }}</div>
<style>
  [v-cloak] {
    display: none;
  }
</style>
```

CUSTOM DIRECTIVES

- You can create custom directives for interacting with the DOM
- Custom directives can be registered globally or locally, similar to components
- Use this sparingly; directives are not a replacement for components

```
// Register a global custom directive called `v-focus`
Vue.directive('focus', {
  // When the bound element is inserted into the DOM...
  inserted: function (el) {
    // Focus the element
    el.focus()
  }
})
```

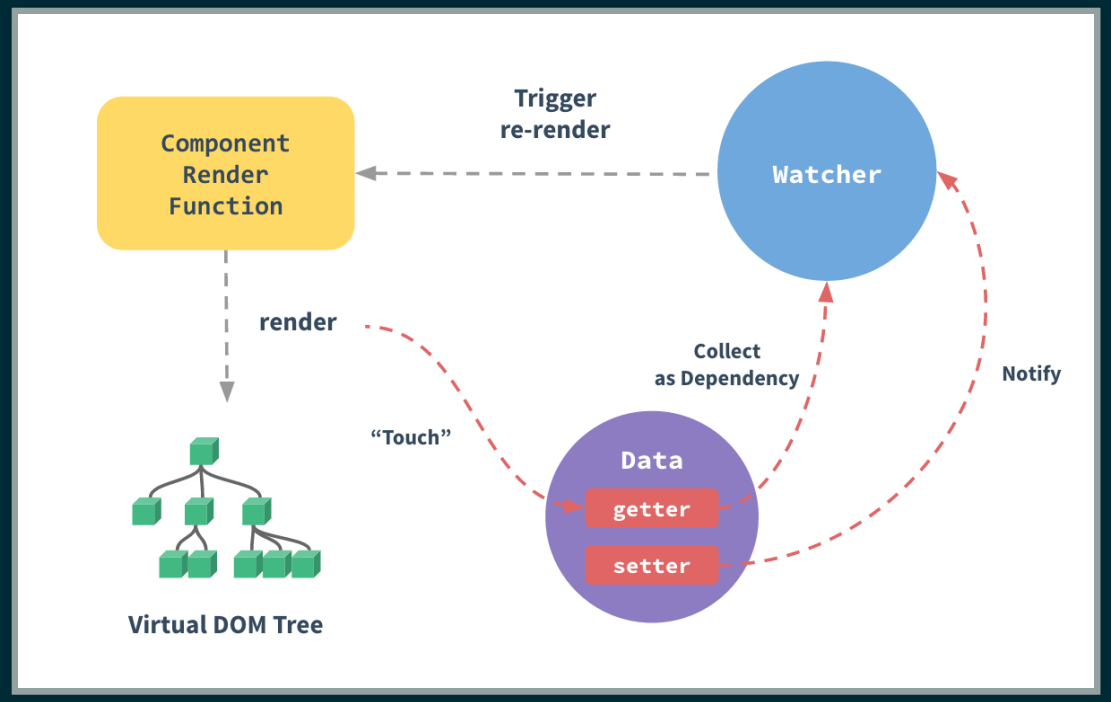
```
<input v-focus>
```

REACTIVITY 1/3

Vue converts a data object's properties to getters and setters when the instance is created, making them reactive.

When you read a reactive property, Vue is notified by the getter. This allows Vue to track dependencies.

When you set a reactive property, Vue is notified by the setter. This allows Vue to notify dependencies and re-render.



REACTIVITY 2/3

- Restrictions with objects
  - Vue cannot detect when new properties are added or existing properties are deleted; all properties must exist in initial data for them to be reactive
  - To modify existing object's properties use `this.$set` and `this.$delete`
  - To assign multiple properties, replace the entire object:

```
this.someData = Object.assign({}, this.someData, { new: 'value' })
```

- Restrictions with arrays
  - Vue cannot detect when you directly set an item with the index, or modify the array length
  - Use `this.$set` and `this.$delete` or Array methods: `splice`, etc.

REACTIVITY 3/3

- DOM updates are asynchronous. If you need to rely on the DOM being updated, use `$nextTick`

```
this.$nextTick(function () {
  console.log(this.$el.textContent)
})
// or
await this.$nextTick()
```

TEMPLATES AND RENDERING

Templates can be defined in a number of ways:

- **Render function**
- **String**
- **Template string**
- **JSX**
- **Single file component**
- Inline HTML
- X-Template HTML

RENDER FUNCTION

- All other template types are compiled to render functions
- Render functions are usually inconvenient to write manually, but you might use them for certain special components

```
new Vue({
  render (createElement) {
    return createElement('div', 'Hello World!')
  }
})
```

STRING TEMPLATE

- Useful when code is not transpiled
- Gets messy for any non-trivial component
- Requires a full build of Vue

```
new Vue({
  template: '<div>Hello Vue!</div>'
})
```

```
new Vue({
  template:
    '<div>' +
      '<p>' +
        'Hello Vue!' +
      '</p>' +
      '<button v-on:click="doSomething">' +
        'Some text here' +
      '</button>' +
    '</div>'
})
```

STRING LITERAL TEMPLATE

- Useful for more complex templates.
- Requires browser support or transpilation.
- Requires a full build of Vue.

```
new Vue({
  template: `
    <div>
      Hello Vue!
    </div>
  `
})
```

JSX

- Similar to JSX in React
- Corresponds closely with render functions
- Requires transpilation
- Directives are mostly not available in JSX

```
new Vue({
  data () {
    return {
      name: 'Vue'
    }
  },
  render () {
    return (
      <div>
        Hello {this.name}!
      </div>
    )
  }
})
```

SINGLE FILE COMPONENT

- Template, script and style in one file
- Requires transpilation
- Most popular choice when using build tools

```
<!-- SomeComponent.vue -->

<template>
  <div class="greeting">Hello {{ name }}!</div>
</template>

<script>
export default {
  data () {
    return {
      name: 'Vue'
    }
  }
}
</script>

<style>
  .greeting {
    font-weight: bold;
  }
</style>
```

INLINE HTML

- Component element contents are used as the template rather than being replaced
- Bad for code organization since the template is separated from the rest of the component

```
<my-component inline-template>
  Hello Vue!
</my-component>
```

X-TEMPLATE HTML

- Bad for code organization since the template is separated from the rest of the component

```
<script type="text/x-template" id="my-template">
  <div>Hello Vue!</div>
</script>
```

```
new Vue({
  template: '#my-template'
})
```

COMPONENTS

COMPONENT REGISTRATION

- Components can be registered globally or locally
- When using PascalCase, Vue makes it available in templates as kebab-case

```
const SomeComponent = {
  // options
}

// Available to all Vue instances
Vue.component('some-component', SomeComponent)

// Available only to this instance
new Vue({
  components: {
    SomeComponent: SomeComponent
  }
})
```

PROPS 1/2

- Props should be named in camelCase, but need to be written in kebab-case in templates

```
Vue.component('some-component', {
  props: [ 'someString' ]
})
```

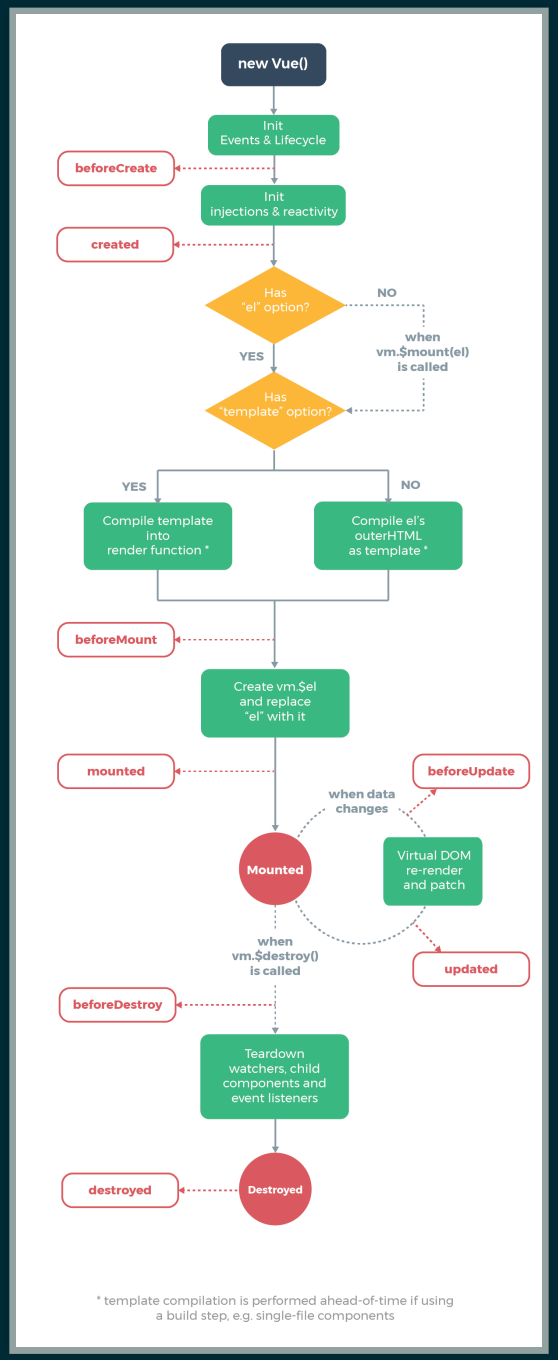
```
<some-component some-string="Hello" />
```

PROPS 2/2

- When passing static props (without `v-bind`), the value is always a string
- To pass any non-string prop, you must use `v-bind` even if the value is static
- Boolean value `true` can be passed by omitting the value

```
<some-component v-bind:some-number="42" />
<some-component some-string="message" />
<some-component some-boolean />
<some-component v-bind:some-array=['a', 'b']>
```

COMPONENT LIFECYCLE



SLOTS 1/2

- Slots allow passing content from parent component to child component
- Slot content has access to the parent's data, not the child's

```
<!-- parent -->
<stylish-button v-bind:color="themeColor">
  {{ callToAction }}
</stylish-button>
```

```
<!-- stylish-button -->
<button v-bind:style="{backgroundColor: color}">
  <slot></slot>
</button>
```

SLOTS 2/2

- Slots can also be named in order to have multiple slots
- Scoped slots allow passing data from child component to the content from parent component

```
<!-- parent -->
<stylish-button v-bind:color="themeColor">
  <template v-slot:icon="slotProps">
    <action-icon v-bind:color="themeColors[slotProps.buttonType]" />
  </template>
  <template slot="text">
    {{ callToAction }}
  </template>
</stylish-button>
```

```
<!-- stylish-button -->
<button v-bind:style="{backgroundColor: color}">
  <slot v-bind:button-type="buttonType">
  <slot name="text"></slot>
</button>
```

DYNAMIC COMPONENTS

- You can use the `is` attribute to switch between components for the same element

```
<component v-bind:is="currentComponent"></component>
```

ASYNC COMPONENTS

- Asynchronous components are supported, with handling for load and error states

```
new Vue({
  components: {
    'some-component': () => ({
      // The component to load (should be a Promise)
      component: import('./SomeComponent.vue'),
      // A component to use while the async component is loading
      loading: LoadingComponent,
      // A component to use if the load fails
      error: ErrorComponent,
      // Delay before showing the loading component. Default: 200ms.
      delay: 200,
      // The error component will be displayed if a timeout is
      // provided and exceeded. Default: Infinity.
      timeout: 3000
    })
  }
})
```

TRANSITIONS

Vue also provides out-of-the-box support for transition animations.

- Vue adds and removes classes when element inside `<transition>` is added or removed
- Vue reads correct timing for adding and removing classes from css properties

```
<div id="demo">
  <button v-on:click="show = !show">
    Toggle
  </button>
  <transition name="fade">
    <p v-if="show">hello</p>
  </transition>
</div>
```

```
new Vue({
  data: {
    show: true
  }
})
```

```
.fade-enter-active, .fade-leave-active {
  transition: opacity .5s;
}
.fade-enter, .fade-leave-to
```

SPA DEVELOPMENT WITH VUE

VUE CLI

- Vue CLI is a standard build tool for Vue projects, based on Node.js
- CLI
  - Global command for creating projects and prototyping
- CLI service
  - Local dev dependency for developing and bundling Vue apps, built on top of webpack
- CLI plugins
  - Optional features for Vue CLI project: Babel, TypeScript, ESLint, PWA, ...

```
yarn global add @vue/cli
vue create demo-app
yarn serve
yarn build
yarn inspect
```

CLI SERVICE

- Configuration without ejecting
  - Projects come with sensible defaults, but you can configure any part of Webpack while still using defaults for the rest
- The webpack configuration is always generated and can be inspected using `vue inspect`
- Build modes and environment variables are supported
- Supports building as app, library or web component

```
// vue.config.js
module.exports = {
  chainWebpack: config => {
    // GraphQL Loader
    config.module
      .rule('graphql')
      .test(/\.graphql$/)
      .use('graphql-tag/loader')
        .loader('graphql-tag/loader')
        .end()
  }
}
```

CLI PLUGINS

- Plugins can be added when creating a project or into an existing project
- Vue CLI provides an API for plugins to
  - Change webpack config
  - Add service commands
  - Add dependencies
  - Creating and modifying files
  - Prompt for options

```
vue add eslint # installs vue-plugin-eslint
```

CORE LIBRARIES

- Vue Router: official router
- Vuex: state management library

LET'S CODE ⟶

THANK YOU