Module 1

Vue.js Overview

What is Vue.js

- Vue is a **progressive framework** for building user interfaces
- 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
- Vue is also perfectly capable of powering sophisticated Single-Page Applications

Advantages Vue

- It’s easy to integrate into existing sites
- Use a component architecture and SPA
- Comprehensive ecosystem
- It’s widely used

Installing Vue – General Steps

1. Install Windows 10 and VMWare Tools
2. Install Chrome
3. Install Visual Studio Code User Installer x64 con las siguientes extensions
    1. Open in browser-TechER
    2. Vetur Pine Wu
4. Install Text Editor (Sublime,Notepad++)
5. Install node.js (Current, x64, also includes npm)
6. Install browsersync (optional)
7. Install Vue CLI
8. Install vue-devtools (debug)

Installing Vue

- Vue does **not** support IE8 and below, because it uses ECMAScript 5 features
- Use Vue with direct <script>
  - Simply download and include with a script tag
    - Development - With full warnings and debug mode
    - Production - Warnings stripped, 33.19KB min+gzip
  - Use CDN

```
Development: <script src="https://cdn.jsdelivr.net/npm/vue"></script>
Production:      <script src="https://cdn.jsdelivr.net/npm/vue@2.6.8/dist/vue.js"></script>
```

Vue different builds

- Directory NPM packages: https://cdn.jsdelivr.net/npm/vue/dist/

	UMD	CommonJS	ES Module (for bundlers)	ES Module (for browsers)
Full	vue.js	vue.common.js	vue.esm.js	vue.esm.browser.js
Runtime-only	vue.runtime.js	vue.runtime.common.js	vue.runtime.esm.js	-
Full (production)	vue.min.js	-	-	vue.esm.browser.min.js
Runtime-only (production)	vue.runtime.min.js	-	-	-

Vue different builds

- **Full**: builds that contain both the compiler and the runtime.
- **Compiler**: code that is responsible for compiling template strings into JavaScript render functions. e.g. passing a string to the template option, or mounting to an element using its in-DOM HTML as the template
- **Runtime**: code that is responsible for creating Vue instances, rendering and patching virtual DOM, etc. Basically everything minus the compiler.
- **UMD**: Can be used directly in the browser via a <script> tag.
- **CommonJS**: Builds are intended for use with older bundlers like **browserify** or **webpack 1** for using modules in JS
- **ES Module**: starting in 2.6 Vue provides two ES Modules (ESM) builds:
  - **ESM for bundlers**: intended for use with modern bundlers like **webpack 2** or **Rollup**
  - **ESM for browsers** (2.6+ only): intended for direct imports in modern browsers via <script type="module">

Installing Vue - NPM

- NPM
  - While **including Vue from a CDN can be a good option when you're just starting out**, or when you're whipping up a quick proof-of-concept app
  - NPM is the **recommended installation method** when **building large scale applications with Vue**
  - Having a good build tooling setup will pay dividends **by automating much of the work of preparing your code for deployment**
  - It pairs nicely with module bundlers such as **Webpack** or **Browserify**.
  - Vue also provides accompanying tools for authoring **Single File Components**.

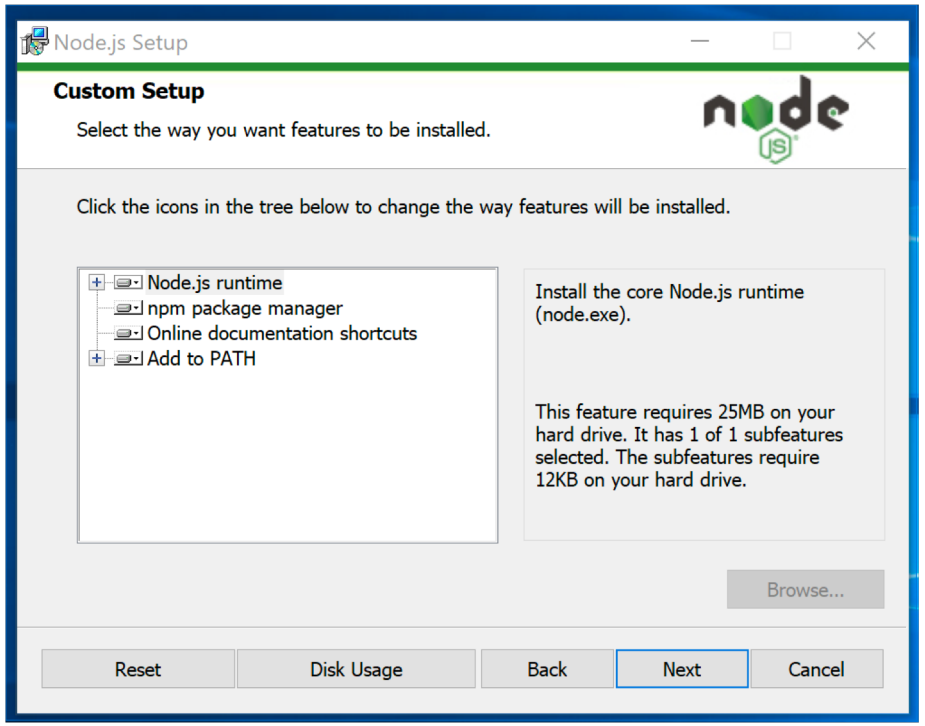
Installing Vue - NPM

- NPM
  - The newer syntax and features (> ES2015) facilitate more concise, readable code. Even though all modern browsers now support ES2015 (aka ES6), you don't always have the luxury of ignoring older browsers.
    - By adding **a build step with** Babel, you can **transpile your code** back to an older version (including some polyfills where necessary) and support older browsers without having to give up the modern language features.
  - A must-have tool for serious JavaScript developers these days is the **code linter.**
    - Can be run during development and/or during a build step, scanning your code for syntax errors as you type
    - ESLint, one of the most popular linters, can also be configured to check if your code conforms to pre-defined sets of rules for coding conventions and even code style preferences

Installing Vue - NPM

- Install NPM
  - Download node.js https://nodejs.org/en/
  - When you install node.js also includes npm

```
C:\Windows\system32>node --version
v11.11.0

C:\Windows\system32>npm --version
6.7.0

C:\Windows\system32>_
```



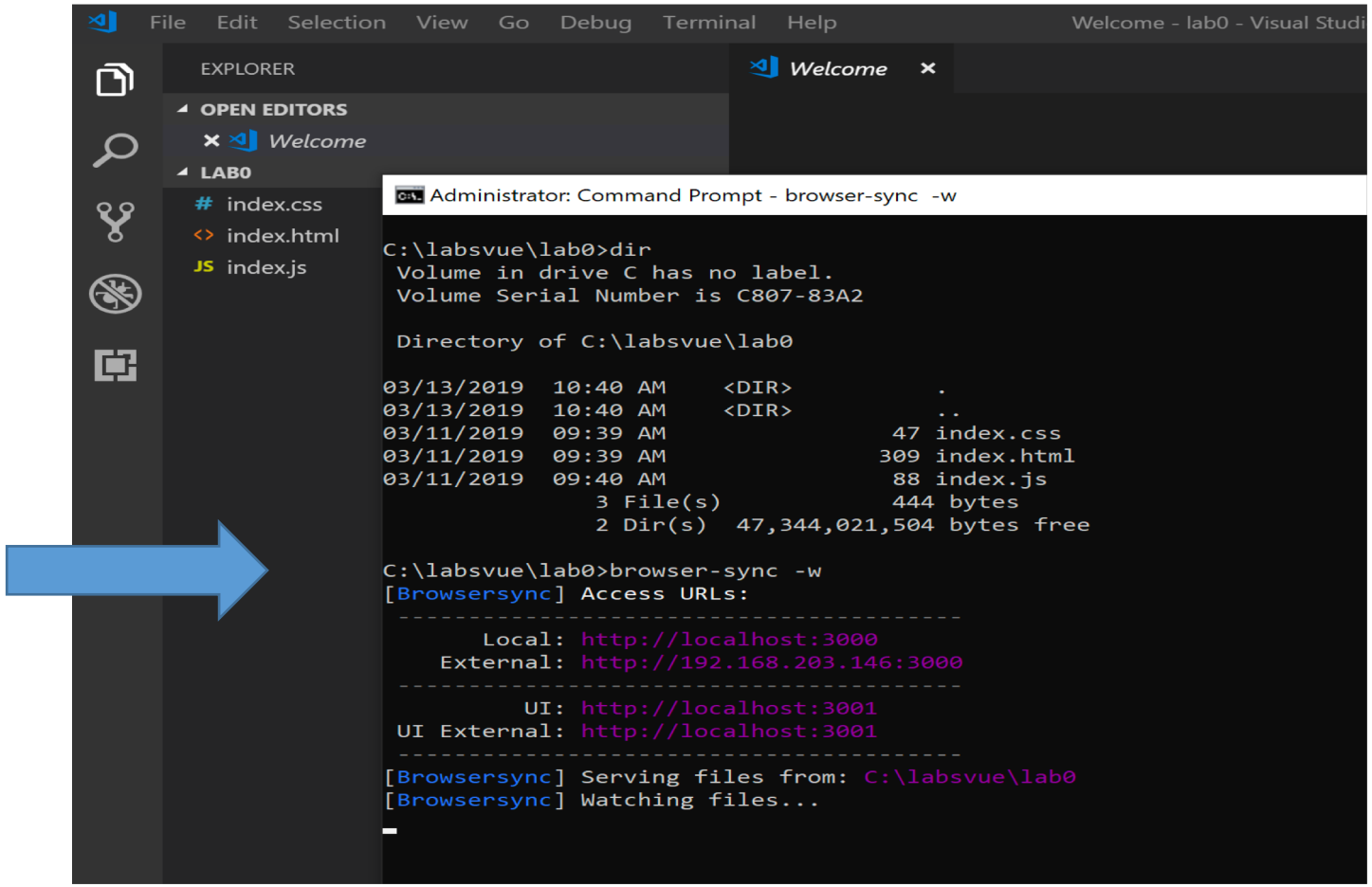
Installing Vue – NPM - browsersync

- Install browsersync (https://www.browsersync.io) is a synchronized browser testing
  - Browsers are automatically updated as you change HTML, CSS, images and other project files.

```
C:\Windows\system32>npm install -g browser-sync
C:\Users\developer\AppData\Roaming\npm\browser-sync -> C:\Users\developer\AppData\Roaming\npm\node_modules\browser-sync\
dist\bin.js
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node_modules\browser-sync\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.7: wanted {"os":"darwin","arch":"any
"} (current: {"os":"win32","arch":"x64"})

+ browser-sync@2.26.3
added 346 packages from 267 contributors in 112.373s
```

Installing Vue – NPM - browsersync

- To execute a project
  - Change to the directory where the project will be run
  - browser-sync –w
  - CTL-C to stop executing

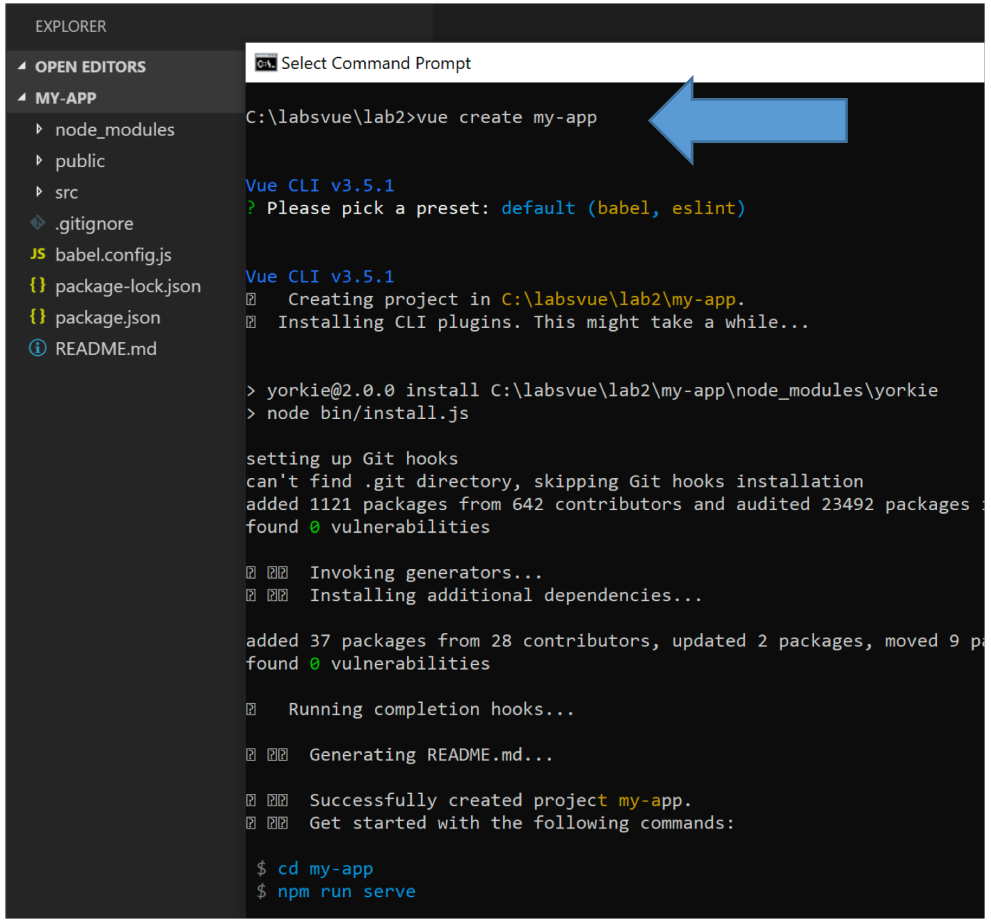


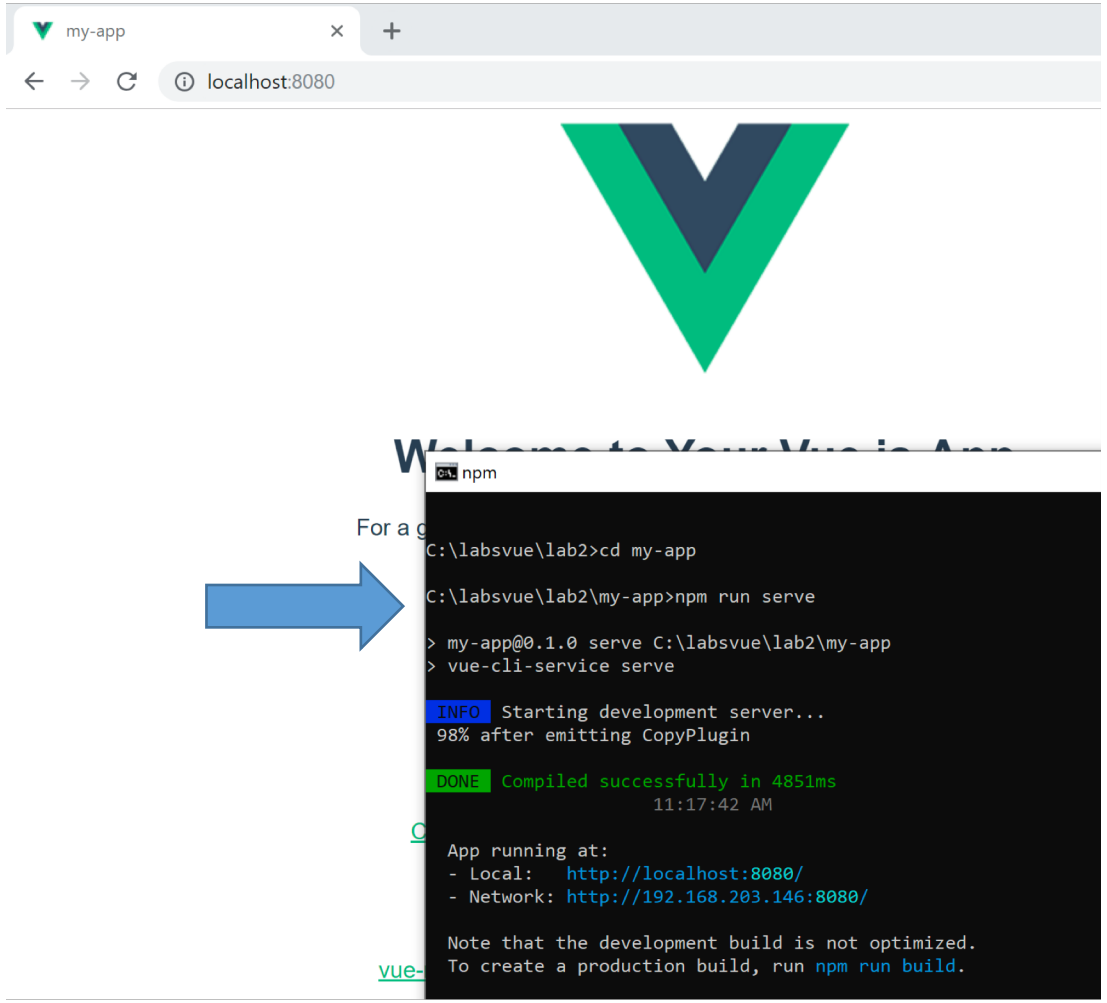
Installing Vue – NPM Vue CLI

- CLI
  - Vue provides an **official CLI** for quickly scaffolding ambitious Single Page Applications
  - It provides build setups for a modern frontend workflow.
  - It takes only a few minutes to get up and running with hot-reload, lint-on-save, and production-ready builds
  - The CLI assumes prior knowledge of Node.js and the associated build tools

```
C:\Windows\system32>npm install -g @vue/cli@latest
npm WARN deprecated cross-spawn-async@2.2.5: cross-spawn no longer requires a build
C:\Users\developer\AppData\Roaming\npm\vue -> C:\Users\developer\AppData\Roaming\npm
ue.js

> protobufjs@6.8.8 postinstall C:\Users\developer\AppData\Roaming\npm\node_modules\@
fjs
> node scripts/postinstall


> nodemon@1.18.10 postinstall C:\Users\developer\AppData\Roaming\npm\node_modules\@v
> node bin/postinstall || exit 0

Love nodemon? You can now support the project via the open collective:
 > https://opencollective.com/nodemon/donate

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node_modules\@vue\cl
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.
ch":"any"} (current: {"os":"win32","arch":"x64"})

+ @vue/cli@3.5.1
added 680 packages from 509 contributors in 78.819s

C:\Windows\system32>npm install -g npm
C:\Users\developer\AppData\Roaming\npm\npm -> C:\Users\developer\AppData\Roaming\npm
i.js
C:\Users\developer\AppData\Roaming\npm\npx -> C:\Users\developer\AppData\Roaming\npm
i.js
+ npm@6.9.0
added 426 packages from 800 contributors in 19.173s

C:\Windows\system32>vue --version
3.5.1
```

Creating an application and running using Vue CLI





Folder structure vue-cli application

- **Node modules directory** - Contains **all of the npm packages needed for your application to run**. Every time you run the command **npm install some-package**; the package some-package will download and be stored in this folder. From here, you can import dependencies into your Vue.js project or reference them manually in an HTML page.
- **Public directory** - Contains your **index.html** file that *everything* gets bootstrapped and injected in to. If you ever have the need to add a **dependency** like Bootstrap 4 into your application **via a CDN, for example, you can add it's respective tags in this file**. However, it's **best practice to always import them via the node_modules folder**.

Folder structure vue-cli application

- **Source directory** - This is the most important directory in the generated project. In this folder, ***all*** **of your single file components**, stylesheets, assets and more are stored here.
  - **assets** - Stores all of your application's assets like images, CSS, and scripts
  - **components** - Stores all of the application's components (*.vue files)
  - **views** - Views are **single file components** that act as "pages" or containers that structure their child components
  - **App.vue** - The **single component** in which all other views and components get injected into. **This is a great place to add global components that should be shared across the app like Header.vue and Footer.vue.**

Folder structure vue-cli application

- **Source Directory**
  - **main.js** - This is **your single *Vue Instance*** in which the App.vue, routes, and all their components get injected into.
  - **router.js** - Define **your URL routes and which component get's loaded** when the URL address is visited.
  - **store.js** - Your Vuex store that contains state, mutations, and actions.
  - **package.json** - JSON file that **lists your NPM dependencies** and small project configurations.
  - **vue.config.js** - A file to add Webpack configs *without* ejecting!

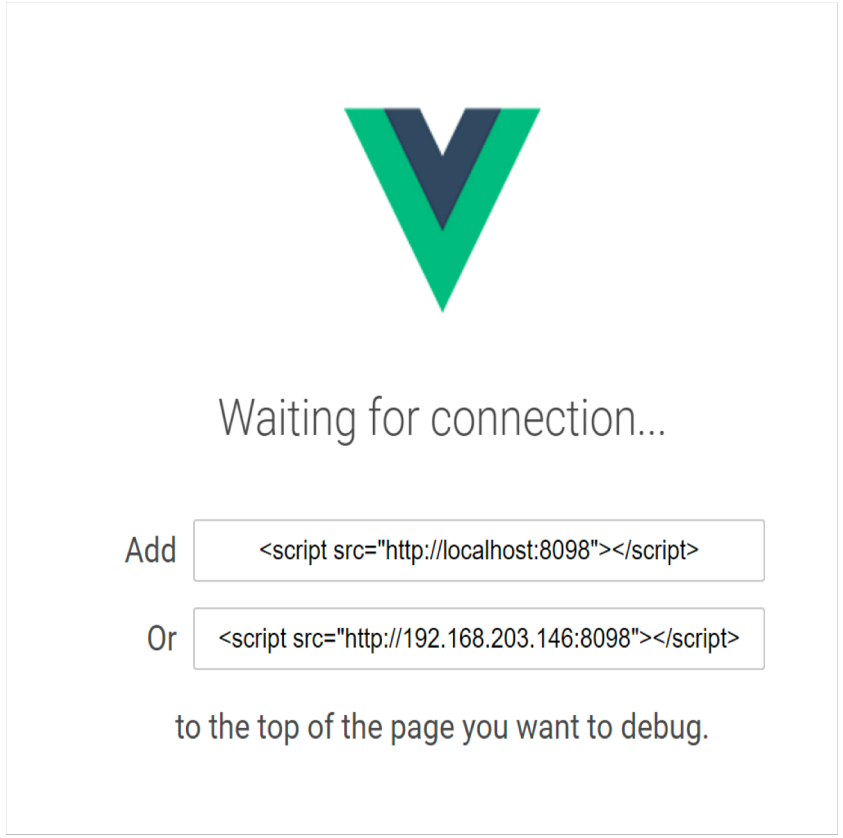
Vue-devtools

- Tooling to aid with debugging
- The devtools **are available as a browser plugin** for both **Chrome** and **Firefox**, **and as a cross-platform Electron app**, which can also debug Vue apps running on mobile devices
- The tools allow you to take a look into a running Vue application, providing three different tabs to help you inspect and debug your code.
  - **Components tab** - View of all the components that make up your application and state
  - **Vuex tab -** This tab is only active if a Vuex store is detected in the application. It allows you to examine the state of the store at any point in time, and all the mutations that have been committed)
  - **Events tab** - Aggregates all the events emitted by your application, from anywhere in the component tree

Vue-devtools - install

- Install vue-devtools
  - https://github.com/vuejs/vue-devtools#installation
- The example use the electron application

```
C:\Windows\system32>npm install -g @vue/devtools
C:\Users\developer\AppData\Roaming\npm\vue-devtools -> C:\
\devtools\bin.js

> electron@1.7.11 postinstall C:\Users\developer\AppData\R
ectron
> node install.js

+ @vue/devtools@4.1.5
added 237 packages from 162 contributors in 90.602s

C:\Windows\system32>vue-devtools
```

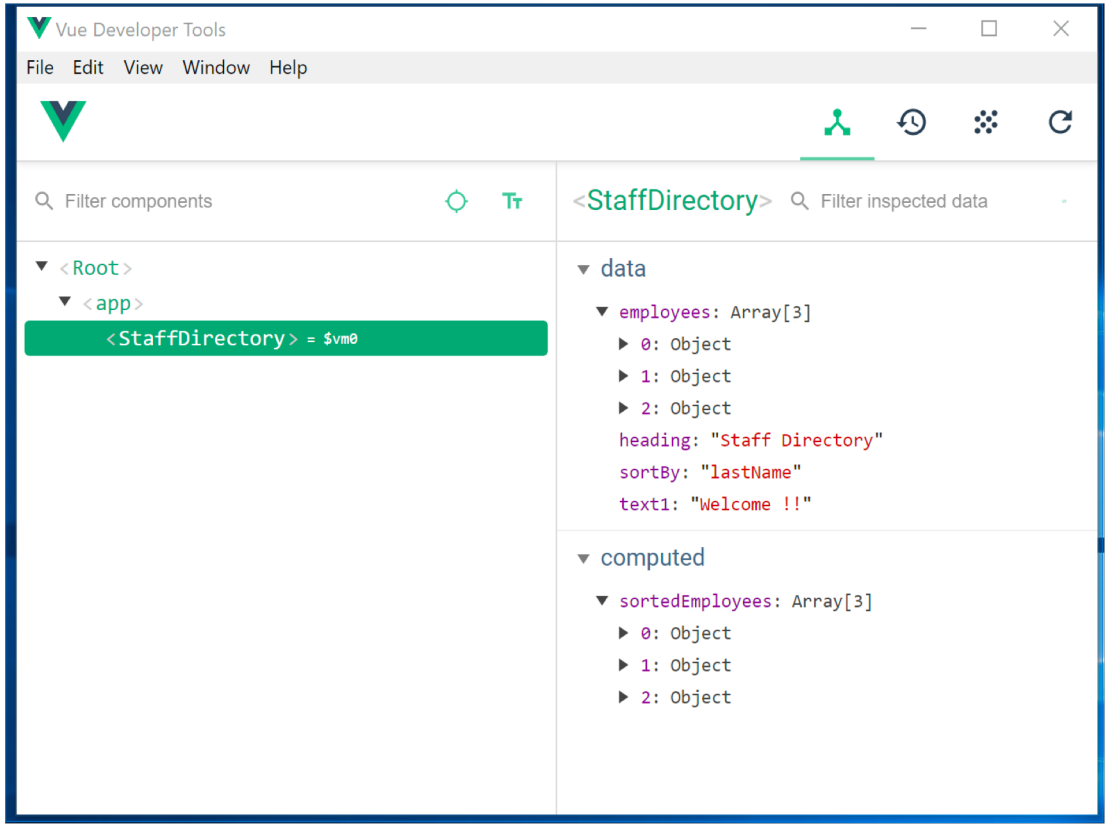


Vue-devtools -using

- For use the vue-devtools modify the application code to include a script element
  - Open de application in VS Code Editor
  - In Explorer expand public and click index.html
    - In the head section add the following code:

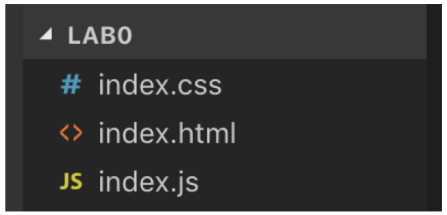
      <!-- JMS: Only for debug use, remove in production -->

      <script src="http://localhost:8098"></script>
  - Save the changes
- Run the application (npm run serve)
- Use the application using browser



Vue instance

- It’s a single occurrence of a Vue.js object
  - Controls the section of the web page or application that you instruct it too
  - A simple Vue application consist of a CSS, Html and JavaScript file
  - In index.js we create a instance of Vue
    - The Vue instance accept options, properties for example:
      - el, data, methods, computed for example
        - The most important property is **el** because it defines *which* part of our application this Vue Instance should control and is mandatory
        - In this case all HTML with id app
        - it’s always recommended one Vue Instance.
      - The data property store all your data to display/modify
      - The data is reactive (can change on user interactions)




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

Display Vue data

- To display your data into your HTML view, use **interpolation** using the mustache syntax: {{}}

```
<html>
    <head>
        <link rel="stylesheet" href="index.css">
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    </head>
    <body>

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

        <script src="index.js"></script>
    </body>
</html>
```

Vue methods

- The methods property stores methods or functions that you can use across your application. Can be used to:
  - Execute lifecycle Vue methods, for example mounted()
  - Running business logic
  - Return values(s) to use in HTML view
  - this in Vue, refers to the Vue Instance, *not* the function

```
var app = new Vue({
    el: '#app',
    data: {
        message: 'Hello Vue!',
        name: "Javier",
        city: "CDMX",
        numberOfThings: 3
    },
    methods: {
        showDataOnMounted(){
        // This is a user defined method
            console.log(this.name);
            console.log(this.city);
        },
        numberOfSomething(number,something){
           return `I have ${number} ${something}`;
        }

    },
    mounted(){
        // This is a Vue lifecycle method
        // Allow access to templates and DOM interaction
        this.showDataOnMounted();
    }
});
```

Vue computed properties

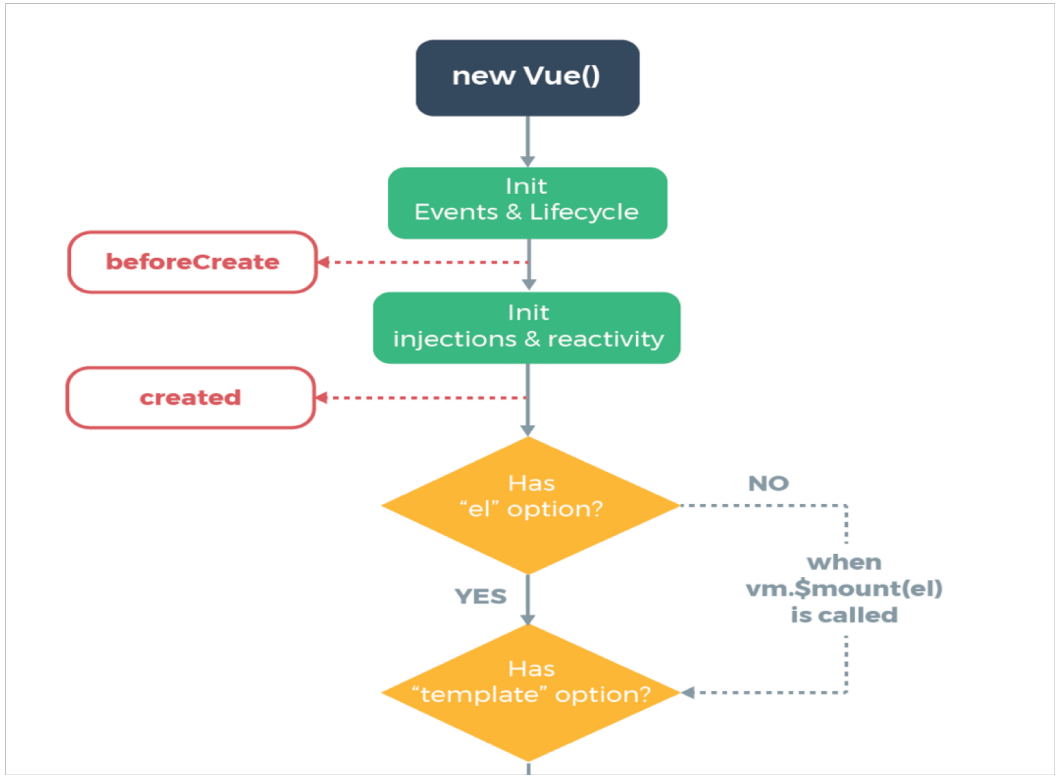
- The **value returned** gets stored as **if it was a data property (becomes reactive)**
- Computed properties **are cached and stored**.
- Much like a method, computed properties can also perform logic and return something
- *Computed properties cannot accept arguments*.

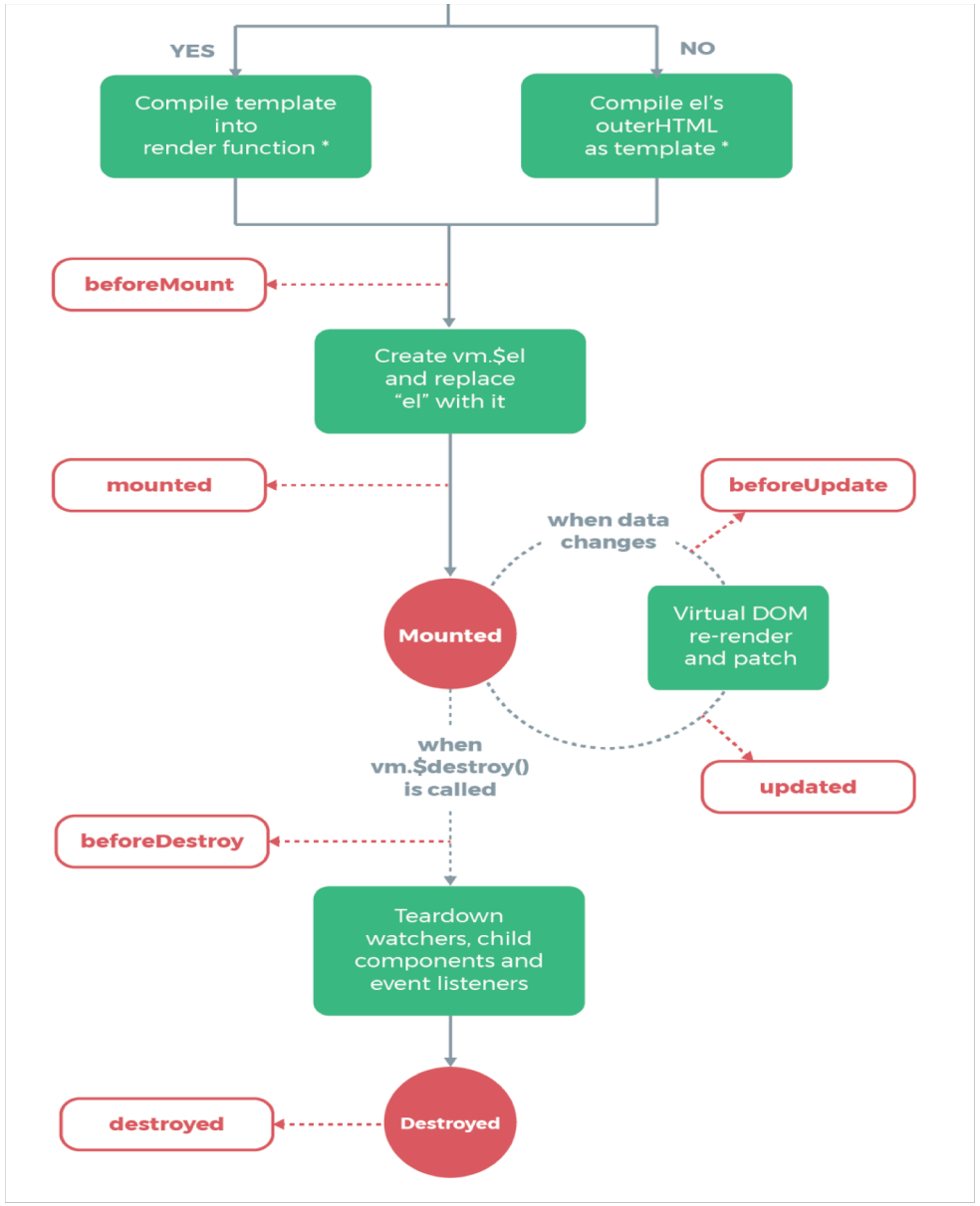
```
var app = new Vue({
    el: '#app',
    data: {
        message: 'Hello Vue!',
        name: "Javier",
        city: "CDMX",
        numberOfThings: 3
    },
    methods: {
        showDataOnMounted(){
        // This is a user defined method
            console.log(this.name);
            console.log(this.city);
        },
        numberOfSomething(number,something){
            return `I have ${number} ${something}`;
        }

    },
    computed : {
        thing() {
            if (this.numberOfThings != 1){
                return 's';
            }
        }

    },
    mounted(){
        // This is a Vue lifecycle method
        // Allow access to templates and DOM interaction
        this.showDataOnMounted();
    }
});
```

Methods vs computed properties

- Methods are **invoked every time Vue.js re-renders** the template
  - It is a waste of computations to invoke the method again, since the result will be exactly the same.
- Computed properties are in the same scope as data properties and methods, so we can access them in the same way from within the template
  - Vue.js **caches computed properties based on their dependencies**, meaning that Vue.js is aware of which dependencies our computed property relies on. A computed property will therefore **only be re-evaluated when one of its dependencies change**

Lifecycle methods





Lifecycle methods

- Some examples of commonly used lifecycle methods:

Method	Example of use
created	This is the place to kick off any Ajax requests for fetching data.
mounted	When this hook is fired, the component has been rendered and inserted into the DOM. This is where you can manipulate the DOM elements—for example, if you're wrapping a non-Vue library of some sort
beforeDestroy	Just before the component is removed from the DOM, this method is called. You can do any cleanup here

Directives

- Vue.js comes pre-packaged with a few directives that **help you render data** to **your view**
- Is some **special token** in the markup that tells the library **to do something at DOM element**
  - Vue.js comes pre-packaged with a few directives that help you render data to your view
- All directives are pre-fixed with **v- ,** some examples:
  - **v-for,** render a list of items based on an array
  - **v-show,** Set the element's display to none or its original value, depending on the truthy-ness of the binding's value.
  - **v-if,** Conditionally insert / remove the element based on the truthy-ness of the binding value
  - **v-else**
  - **v-else-if**
  - **v-on ,** Attaches an event listener to the element
  - **v-bind, U**sed to reactively update an HTML attribute:
  - **v-model ,** Create a **two-way binding** on a form or editable element

Directives example v-for, v-show, v-if

```
8   <div id="app">
9       {{ message }}
10      <p>My name is {{ name }}, I am from {{ city }}</p>
11      <p>{{ numberOfSomething(numberOfThings,'dog') }}{{ thing }}</p>
12      <p>Days of the week:</p>
13      <ul>
14          <li v-for="day in daysOfWeek">
15              {{ day }}
16          </li>
17      </ul>
18      <p>Airports:</p>
19      <div v-for="airport in airports">
20          <span>{{ airport.code }} , </span>
21          <span v-show="airport.country === 'Mexico'">Country: {{ airport.country}}</span>
22      </div>
23      <p>Weekend:</p>
24      <ul>
25          <li v-for="day in daysOfWeek">
26              <span v-if="day === 'Saturday'">{{ day }} is Weekend</span>
27              <span v-else-if="day === 'Sunday'"> {{ day }} is Weekend</span>
28              <span v-else> {{ day }} is not Weekend</span>
29          </li>
30      </ul>
31  </div>
```

Directives v-on (@)

- v-on, This directive declares a method (event handler) to run on a specific event such as click, keyup, or submit
- The **event and the directive are separated** by a colon (:)
- You can use the shorthand syntax (**@**)

```
<button v-on:click="showAlert">Show Alert1</button>
<button @click="showAlert">Show Alert2</button>
```

```
methods: {
    showDataOnMounted(){
    // This is a user defined method
        console.log(this.name);
        console.log(this.city);
    },
    numberOfSomething(number,something){
       return `I have ${number} ${something}`;
    },
    showAlert(){
       alert('Send using v-on directive');
    }

},
computed : {
    thing() {
        if (this.numberOfThings != 1){
            return 's';
        }
    }

},
```

Directives v-on and pass arguments to event handler

```
<button v-on:click="showAlert">Show Alert1</button>
<button @click="showAlert">Show Alert2</button>
<button @click="showAlert2('Welcome !!')">Show Alert3</button>
```

```
methods: {
    showDataOnMounted(){
    // This is a user defined method
        console.log(this.name);
        console.log(this.city);
    },
    numberOfSomething(number,something){
       return `I have ${number} ${something}`;
    },
    showAlert(){
       alert('Send using v-on directive');
    },
    showAlert2(string){
        alert(string);
     }
```

Directives v-on and pass arguments to event handler

- You can use the special $event directive

```
<template>
  <a @click="handleClick($event)">Click me!</a>
</template>

<script>
export default {
  methods: {
    handleClick: function(event) {
      console.log(event)
    }
  }
}
</script>
```

```
<template>
  <a @click="handleClick('something', $event)">Click me!</a>
</template>

<script>
export default {
  methods: {
    handleClick: function(text, event) {
      console.log(text)
      console.log(event)
    }
  }
}
</script>
```

Directives v-on, event modifiers

- Are pre-set modifications that **you can chain** to your event listener **via dot notation**
  - **.stop –** No event propagation
  - **.prevent –** prevents default action
  - **.capture -** Uses event capturing instead of event bubbling
  - **.self -** Make sure the event was not bubbled from a child event, but directly happened on that element
  - **.once -** The event will only be triggered exactly once
  - **.passive -** A Boolean which, if true, indicates that the function specified by listener will never call prevent

Directives v-on, event modifiers

```HTML
<!-- 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 -->
<!-- i.e. an event targeting an inner element is handled here before b
<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>
```

Directives v-on, key modifiers

- Not only can you listen for an event, but you can also listen for **specific keys** that have been pressed

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

- You can directly use any valid key names exposed via **KeyboardEvent.key** as modifiers by converting them to kebab-case

```
<input v-on:keyup.page-down="onPageDown">
```

- The use of keycode event is deprecated and may be not supported in new browsers
- If you want to listen the shift key

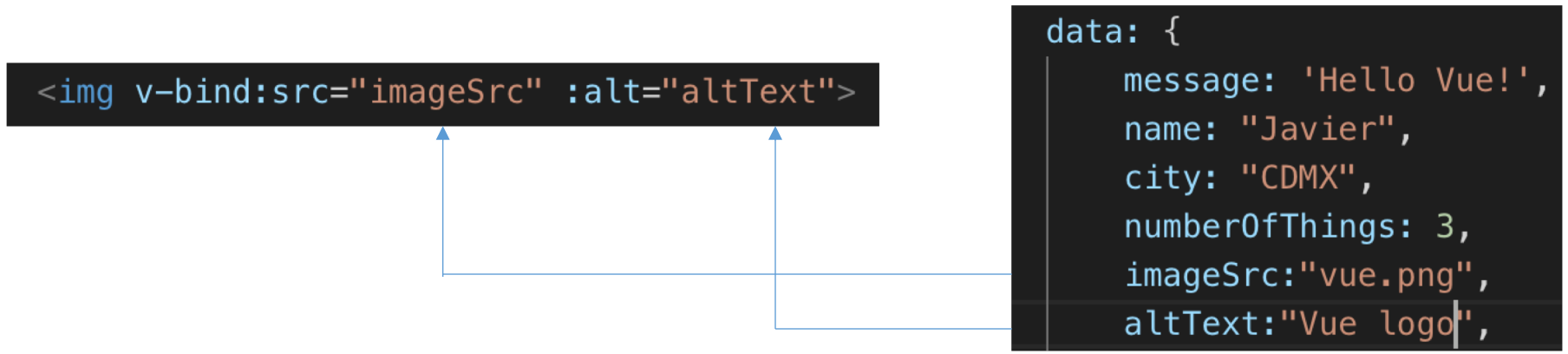
```
<button @keyup.16="someFunction">Button Text</button>
```

Directives v-on, key modifiers

- Vue provides aliases for the most commonly used key codes when necessary for legacy browser support:
  - .enter
  - .tab
  - .delete (captures both “Delete” and “Backspace” keys)
  - .esc
  - .space
  - .up
  - .down
  - .left
  - .right

Directives v-bind(:)

- Is used when you need to “bind” or connect your view to some data in your Vue instance or component.
- Vue.js comes pre-shipped with a shorthand syntax for v-bind: the colon **(:)**



Additional information

- Vue documentation - https://vuejs.org/v2/guide/# and click to the text “created a video”
- Lifecycle diagram - https://vuejs.org/v2/guide/instance.html#Lifecycle-Diagram
- Vue directives - http://optimizely.github.io/vuejs.org/api/directives.html
- Vue events - https://vuejs.org/v2/guide/events.html
- JavaScript event keycode - http://keycode.info
- JavaScript modules - http://exploringjs.com/es6/ch_modules.html

Additional information

- What is Vuex? , click to the video - https://vuex.vuejs.org
- Download vue-devtools - https://github.com/vuejs/vue-devtools#installation
- Vue guide - https://012.vuejs.org/guide/index.html#Concepts_Overview