Programação Web I

M01 - Introduction to Vue.js

ESMAD | TSIW | 2020-21

M01 - Introduction to Vue.js

Index

M01 - Introduction to Vue.js

Index

M01 - Introduction to Vue.js

1. Introduction to Vue.js

- JavaScript progressive framework
- Created by Evan You
- History:
  - Started in 2013
  - Last version: 3.0 (October, 2020)
- Links:
  - Site: https://vuejs.org
  - Repository: https://github.com/vuejs/vue-next
- License: MIT

M01 - Introduction to Vue.js

1. Introduction to Vue.js

- Javascript Framework
  - To organize and simplify the frontend development
  - To develop interactive Web interfaces
- Main advantages:
  - Small (33kb - production version)
  - Easy to install
  - Easy to learn (small learning curve)
  - Easy to integrate with other projects and libraries

M01 - Introduction to Vue.js

1. Introduction to Vue.js
   - Main libraries/tools
     - Vue-router
     - Vuex
     - Vue-loader
     - Vue-devtools
     - Vue-cli
     - Vue-test-utils
     - Vuetify
     - Bootstrap Vue

M01 - Introduction to Vue.js

Index

M01 - Introduction to Vue.js

2. Installation

- 3 ways to install Vue.js
  - Using CDN
  - Using NPM
  - Using Vue CLI

M01 - Introduction to Vue.js

2. Installation

- CDN
  - For prototyping or learning purposes, use the latest version:

```
<head>
    ...
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

</head>
```

  - For production, link to a specific version number and build to avoid unexpected breakage from newer versions

```
<head>
    ...
    <script src="https://cdn.jsdelivr.net/npm/vue@2.6.12"></script>

</head>
```

M01 - Introduction to Vue.js

2. Installation

- Node Package Manager (NPM)
  - NPM is the recommended installation method when building large scale applications with Vue.

```
# latest stable
$ npm install vue
```

M01 - Introduction to Vue.js

2. Installation

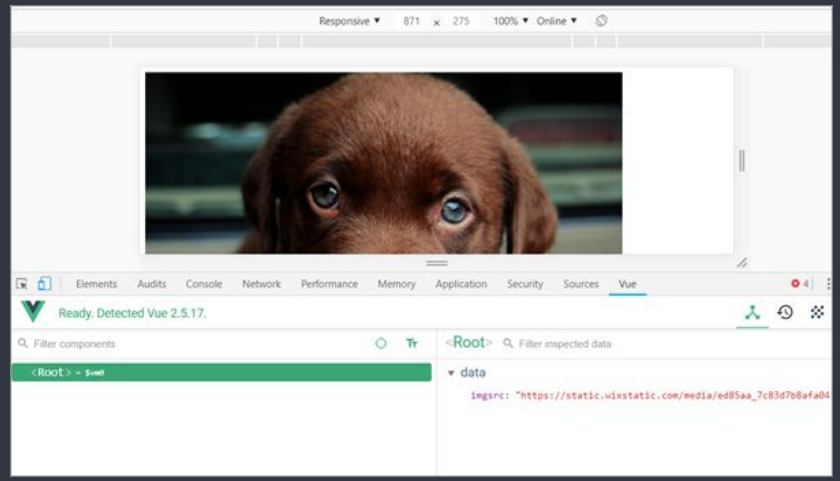
- Vue CLI
  - Vue provides an official CLI for quickly scaffolding Single Page Applications (SPA)
  - It provides build setups for a modern frontend workflow, hot-reload, lint-on-save, etc.
  - For Vue 3, use Vue CLI v4.5 available on npm as @vue/cli

```
yarn global add @vue/cli
# OR
npm install -g @vue/cli
```

M01 - Introduction to Vue.js

2. Installation

- Complementary installations
  - Visual Studio Code
  - Vue DevTools - Browser extension to vue.js apps debug
  - Vetur extension (syntax highlighting, snippets, etc.)



M01 - Introduction to Vue.js

Index

M01 - Introduction to Vue.js

3. My first application

1. Create a folder HelloVue
2. Open the folder in VSC
3. Create a file index.html and add the initial skeleton
4. Create a reference in the html file to the vue.js file

```
<head>
    ...
    <script src="https://unpkg.com/vue"></script>

</head>
```

M01 - Introduction to Vue.js

3. My first application

5. Add a new tag <script> to create a new Vue instance

```
<script>
  const app = new Vue({
    el: '#app',
    data: {
      message: 'Hello Vue!'
    }
  })
</script>
```

6. Add a tag inside the <body> tag

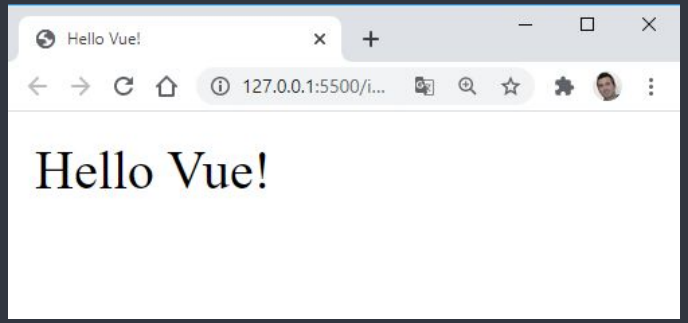
```
<div id="app">
    {{ message }}
</div>
```

M01 - Introduction to Vue.js

3. My first application

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

  <script>
    const app = new Vue({
      el: "#app",
      data: {
        message: "Hello Vue!",
      },
    });
  </script>
</body>
```

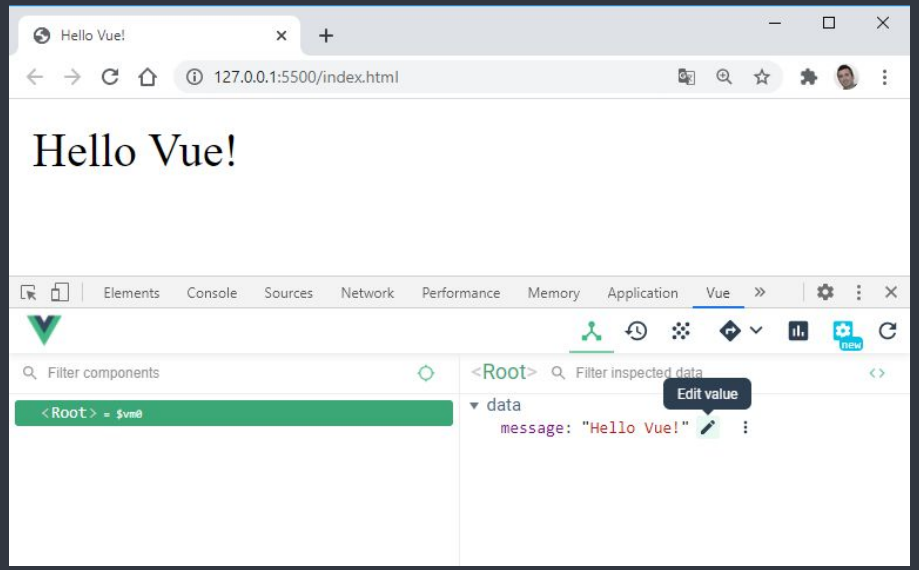


M01 - Introduction to Vue.js

3. My first application

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

  <script>
    const app = new Vue({
      el: "#app",
      data: {
        message: "Hello Vue!",
      },
    });
  </script>
</body>
```



M01 - Introduction to Vue.js

Index

M01 - Introduction to Vue.js

4. The Vue instance

- Every Vue application starts by creating a new Vue instance with the Vue function:

```
const vm = new Vue({
    // options
})
```

M01 - Introduction to Vue.js

4. The Vue instance

- When we create a vue instance we need to pass a options object
- Main properties of the object:

Reference to the DOM element where the Vue data should be rendered

Object with data to render

```
const vm = new Vue({
  el: "#app",
  data: { msg: "Hello Vue" },
  methods: {
    jump: function () {
      return this.msg;
    },
  },
});
```

Methods to manipulate data

M01 - Introduction to Vue.js

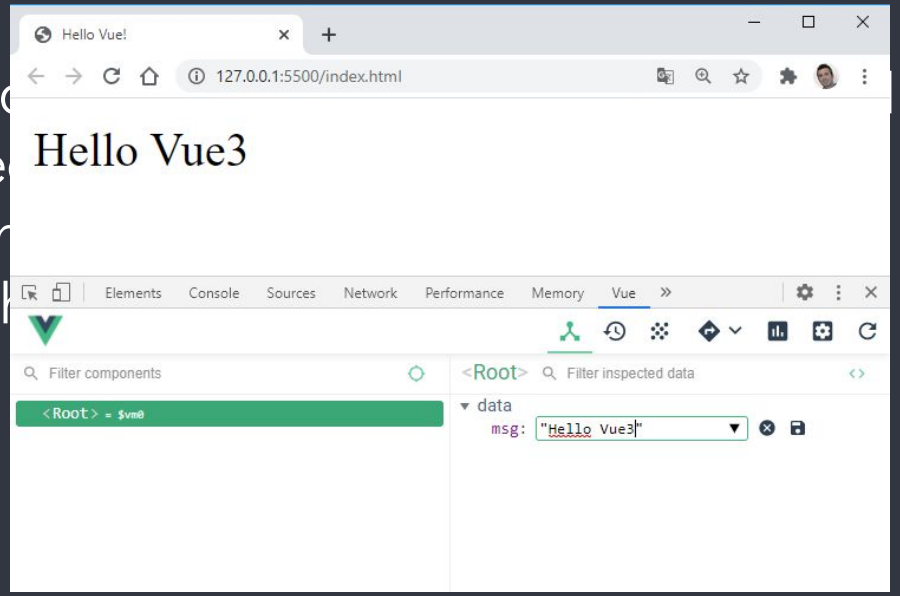
4. The Vue instance

- data property
  - When a Vue instance is created, it adds all the properties found in its data object to Vue's reactivity system.
  - When the values of those properties change, the view will "react", updating to match the new values.

M01 - Introduction to Vue.js

4. The Vue instance

- data property
  - When a Vue instance is crea
    in its data object to Vue's re
  - When the values of those pr
    "react", updating to match th



M01 - Introduction to Vue.js

4. The Vue instance

- el property
  - References an id of a DOM element
  - All reactivity on data object properties will only be done on this element
  - Values in data object are presented in the element pointed by el with interpolation using the mustache syntax {{...}}

el mapping

```
<div id="vue_det">
  <h1>Firstname : {{firstName}}</h1>
  <h1>Lastname : {{lastName}}</h1>
  <h1>{{myDetails()}}</h1>
</div>
<script>
  const vm = new Vue({
    el: "#vue_det",
    data: {
      firstName: "Ricardo",
      lastName: "Queirós",
    },
```

el property

M01 - Introduction to Vue.js

4. The Vue instance

- methods property
  - We can also add methods to the instance through the methods object

```
<div id="vue_det">
  <h1>Firstname : {{firstName}}</h1>
  <h1>Lastname : {{lastName}}</h1>
  <h1>{{myDetails()}}</h1>
</div>
<script>
  const vm = new Vue({
    el: "#vue_det",
    data: {
      firstName: "Ricardo",
      lastName: "Queirós",
    },
    methods: {
      myDetails: function () {
        return `Eu sou o ${this.firstName} ${this.lastName}`;
      },
    },
  });
</script>
```

method invocation

methods property

M01 - Introduction to Vue.js

4. The Vue instance

- Other options: props, computed, etc.
- Built-in properties: $attrs and $emit. These properties all have a $ prefix to avoid conflicting with user-defined property names.

M01 - Introduction to Vue.js

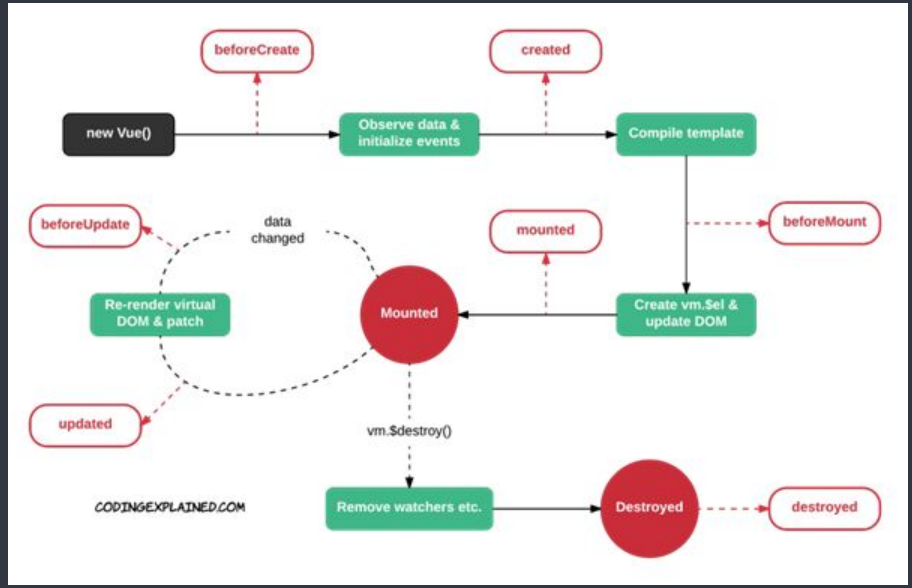
4. The Vue instance

- Life cycle
  - Each instance of Vue goes through a series of startup steps when it is created
  - For example, Vue needs to configure data observation, compile the template, mount the instance in DOM, and update the DOM when data changes
  - Along the way, it also execute functions called lifecycle functions, giving users the opportunity to add code at specific stages of the cycle

M01 - Introduction to Vue.js

4. The Vue instance

- Life cycle



M01 - Introduction to Vue.js

4. The Vue instance

- Four phases with two functions each:

1. Creation (initialization) - beforeCreate and created
2. Mounting (DOM Insertion) - beforeMount and mounted
3. Update (differentiate and render again) - beforeUpdate and updated
4. Teardown - beforeDestroy and destroyed

M01 - Introduction to Vue.js

4. The Vue instance

- Creation (initialization)
  - beforeCreate: fired before instance is initialized
  - created: Fired after the instance was initialized, but before being added to the DOM (good time to get data from external sources)

- Mounting (DOM Insertion)
  - beforeMount: fired after the element is ready to be added to the DOM, but before that
  - mounted: fired after the element has been created (but not necessarily added to DOM: use nextTick for this)

M01 - Introduction to Vue.js

4. The Vue instance

- Update (differentiate and render again)
  - beforeUpdate: fired when there are changes to make to DOM
  - updated: fired after changes are written to DOM

- Teardown
  - beforeDestroy: fired when component is about to be destroyed and removed from DOM
  - destroyed: fires after component has been destroyed

M01 - Introduction to Vue.js

4. The Vue instance

- For instance, the created function can be used to execute code after creating an instance:

```
const vm = new Vue({
  data: { a: 1 },
  created: function () {
    // this references the vm instance
    console.log(`a is: ${this.a}`); // => "a is: 1"
  },
});
```

M01 - Introduction to Vue.js

Index

M01 - Introduction to Vue.js

5. Template syntax

- Vue.js uses an HTML-based template syntax that allows you to declaratively bind the rendered DOM to the Vue instance's data.

- Vue compiles the templates into Virtual DOM render functions.

- Using the reactivity system, Vue finds the minimal number of components to re-render and apply DOM manipulations when the app state changes.

M01 - Introduction to Vue.js

5. Template syntax

- Main concepts in template syntax:
  - Interpolations
  - Directives
  - Shortands

M01 - Introduction to Vue.js

5. Template syntax

- Interpolations
  - Vue.js uses an HTML-based template syntax that lets you declaratively link rendered DOM to underlying Vue instance data
  - Interpolation Types:
    - Text
    - Html
    - Attributes
    - Javascript expressions

M01 - Introduction to Vue.js

5. Template syntax

- Interpolations > Text
  - For text interpolation use the “Mustache” syntax (double curly braces):

```
<span>Message: {{ msg }}</span>
```

  - The mustache tag will be replaced with the value of the msg property on the corresponding data object. It will also be updated whenever the data object’s msg property changes.
  -
  - You can also perform one-time interpolations that do not update on data change by using the v-once directive:

```
<span v-once>This will never change: {{ msg }}</span>
```

M01 - Introduction to Vue.js

5. Template syntax

- Interpolations > Html
  - The mustache tag interprets data as plain text, not HTML
  - To produce real HTML, you must use the v-html directive:

```
<div id="app">
  <p>Using mustaches: {{ rawHtml }}</p>
  <p>Using v-html directive: <span v-html="rawHtml"></span></p>
</div>

<script>
  const vm = new Vue({
    el: "#app",
    data: {
      rawHtml: "<b>Hello Vue!<b>",
    },
  });
</script>
```

Using mustaches: <b>Hello Vue!<b>

Using v-html directive: **Hello Vue!**

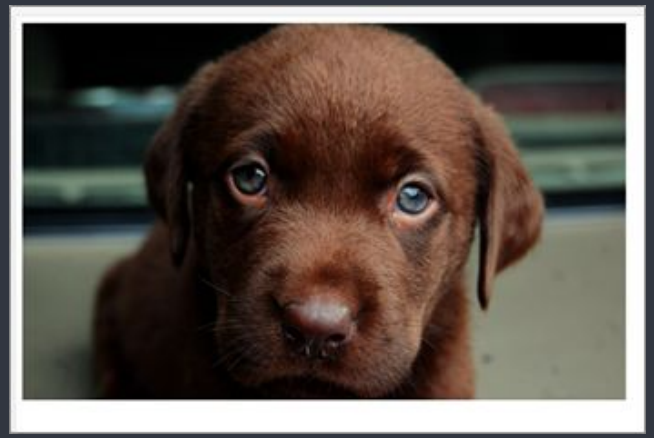
M01 - Introduction to Vue.js

5. Template syntax

- Interpolations > Attributes
  - Mustache tag cannot be used inside html attributes
  - For this you must use the v-bind directive:

```
<div id="app">
  <img v-bind:src="imgsrc" />
</div>

<script>
    const vm = new Vue({
    el: "#app",
    data: {
        imgsrc: "imgs/myDog.jpg"
    }
  })
</script>
```



M01 - Introduction to Vue.js

5. Template syntax

- Interpolations > Attributes
  - For boolean attributes (true or false), v-bind works differently:

```
<button v-bind:disabled="isButtonDisabled">Button</button>
```

  - If isButtonDisabled has a value of null, undefined, or false, then the disabled attribute is not included in the rendered <button> element

M01 - Introduction to Vue.js

5. Template syntax

- Interpolations > JS expressions
  - Vue.js supports JavaScript expressions within all data bindings

```
{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

{{ message.split('').reverse().join('') }}

<div v-bind:id="'list-' + id"></div>
```

  - The expressions will be evaluated as JS in the instance data scope

M01 - Introduction to Vue.js

5. Template syntax

- Interpolations > JS expressions
  - One restriction is that each binding can contain only a single expression, so the following will NOT work:

```
// this is a declaration not an expression!
{{ const a = 1 }}

// traditional conditional structures will not work, try ternary expressions instead
{{ if(ok) { return message } }}
```

M01 - Introduction to Vue.js

5. Template syntax

- Directives
  - Directives are special attributes with the prefix v-
  - Attribute values are a single JavaScript expression (except v-for)
  - The job of a directive is to apply side-effects reactively to the DOM when the value of its expression changes
  - Here's an example of a simple directive:

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

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

M01 - Introduction to Vue.js

5. Template syntax

- Directives
  - Can have:
    - Arguments
    - Modifiers

M01 - Introduction to Vue.js

5. Template syntax

- Directives with arguments
  - Some directives may be given an "argument", denoted by a colon after the directive name. For instance, the v-bind directive is used to reactively update an HTML attribute:

```
<a v-bind:href="url">...</a>
```

  - Here, href is the argument, which tells the v-bind directive to bind the element's href attribute to the value of the url expression

M01 - Introduction to Vue.js

5. Template syntax

- Directives with arguments
  - Another example is the v-on directive, which listens for DOM events:

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

  - Here the argument is the name of the event to hear

M01 - Introduction to Vue.js

5. Template syntax

- Directives with modifiers
  - Modifiers are special postfixes denoted by a dot, which indicate that a directive must be bound in some special way

  - For instance, the .prevent modifier tells the v-on directive to call event.preventDefault() on the triggered event:

```
<form v-on:submit.prevent="onSubmit">...</form>
```

M01 - Introduction to Vue.js

5. Template syntax

- Shorthands
  - The v prefix serves as a cue to identify Vue attributes in your models
  - But it can make the page verbose in case of much use
  - Vue.js provides special shortcuts (: and @) for 2 of the most commonly used directives, v-bind and v-on:

```
// traditional syntax
<a v-bind:href="url">...</a>

// abbreviated syntax
<a :href="url">...</a>
```

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

// abbreviated syntax
<a @click="doSomething">...</a>
```

M01 - Introduction to Vue.js

Index

M01 - Introduction to Vue.js

6. Computed properties and Watchers

- Template expressions are very convenient, but are for simple operations
- Putting too much logic into templates can make them difficult to maintain
- For instance (inverted string):

```
<div id="intro">
    {{ message.split('').reverse().join('') }}
</div>
```

- Template is complex, less declarable and legible
- The problem is bigger when you want to include inverted strings in template more than once
- Therefore, for any complex logic, you must use a computed property

M01 - Introduction to Vue.js

6. Computed properties and Watchers

- Template

Invocation of **computed property**

```
<div id="intro">
    <p>Mensagem original: {{ message }}</p>
    <p>Mensagem invertida: {{ reversedMessage }}</p>
</div>

<script>
    const vm = new Vue({
        el: "#intro",
        data: {
            message: "Hello"
        },
        computed: {
            reversedMessage: function () {
                return this.message.split('').reverse().join('')
            }
        }
    })
</script>
```

Definition of **computed property**

M01 - Introduction to Vue.js

6. Computed properties and Watchers

- A computed reversedMessage property has been declared
- The function will be used as a getter function for the vm.reversedMessage property

```
console.log(vm.reversedMessage) // => 'olleH'
vm.message = 'Goodbye'
console.log(vm.reversedMessage) // => 'eybdooG'
```

```
<div id="intro">
    <p>Mensagem original: {{ message }}</p>
    <p>Mensagem invertida: {{ reversedMessage }}</p>
</div>

<script>
    const vm = new Vue({
        el: "#intro",
        data: {
            message: "Hello"
        },
        computed: {
            reversedMessage: function () {
                return this.message.split('').reverse().join('')
            }
        }
    })
</script>
```

M01 - Introduction to Vue.js

6. Computed properties and Watchers

- Computed properties vs. Methods
  - Instead of a computed property, we could have defined the same function as a method. The end result would be the same!

```
computed: {
    reversedMessage: function () {
        return this.message.split('').reverse().join('')
    }
},
methods: {
    reversedMessage: function () {
        return this.message.split('').reverse().join('')
    }
}
```

M01 - Introduction to Vue.js

6. Computed properties and Watchers

- Computed properties vs. Methods
  - Computed properties
    - are cached based on their dependencies
    - are only reevaluated when its dependencies are changed
    - If message is not changed, multiple access to reversedMessage immediately returns the previously calculated result without the need to execute the function again
  - Methods: a method call will always execute the function whenever a new render occurs

M01 - Introduction to Vue.js

6. Computed properties and Watchers

- Computed properties are read only, but a setter can be added

```
computed: {
    fullName: {
        // getter
        get: function() {
            return this.message.split('').reverse().join('')
        },
        // setter
        set: function (newValue) {
            const names = newValue.split(' ')
            this.firstName = names[0]
            this.lastName = names[names.length - 1]
        }
    }
}
```

- when run vm.fullName = 'John Doe'
  - Setter is called
  - vm.firstName and vm.lastName will be updated accordingly

M01 - Introduction to Vue.js

6. Computed properties and Watchers

- Summary of key differences
  - Computed properties are cached
    - if you call a method multiple times in a template the code inside the method will be executed every time the method is called
    - If you call a computed property multiple times, the code is executed only once, and after that the cached value will be used. The code will execute again only when the method dependency changes
    - This is good if you are doing something potentially resource intensive as it ensures that the code does not run longer than necessary
  - Computed properties allow you to switch to an object with get and set properties

M01 - Introduction to Vue.js

6. Computed properties and Watchers

- Data Object, methods or computed properties?
  - Data object - better for pure data: if you want to put data somewhere to use in your template or method or computed property
  - Methods - are better when you want to add functions to your templates: you can pass data, and methods do something with that data and return different data
  - Computed properties - are better for executing more complicated expressions that you do not want to use in the template because they are too long or would need to be repeated too often. They usually work with other computed properties or other data. They are basically an extended and more powerful version of the Data object

M01 - Introduction to Vue.js

6. Computed properties and Watchers

- Data Object, methods or computed properties?
  - Data object
  - Methods
  - Computed properties

	Readable?	Writable?	Accepts arguments?	Computed?	Cached?
The data object	Yes	Yes	No	No	N/A, as it's not computed
Methods	Yes	No	Yes	Yes	No
Computed properties	Yes	Yes	No	Yes	Yes

M01 - Introduction to Vue.js

6. Computed properties and Watchers

- Watchers
  - Allow to observe changes in a property of the data object or a computed property

```
const vm = new Vue({
    el: '#intro',
    data: {
        count: 2
    },
    watch: {
        count: function () {
            // this.count changed!
        }
    }
})
```

Definition of a **watcher**

M01 - Introduction to Vue.js

6. Computed properties and Watchers

- Watchers
  - Benefits:
    - easy to use
    - good for performing asynchronous operations
  - Alternative: Using a computed property with a setter
  - Receive two arguments when the observed property is changed:
    - the new observed property value
    - the old value

```
watch: {
    count: function (newValue, oldValue) {
        console.log(newValue, oldValue)
    }
}
```

M01 - Introduction to Vue.js

6. Computed properties and Watchers

- Watchers
  - Can observe for changes in object properties:

```
data: {
    person: { name: 'ricardo' }
},
watch: {
    'person.name'() {
        // this.person.name changed!
    }
}
```

M01 - Introduction to Vue.js

6. Computed properties and Watchers

- Watchers
  - You may want to look at an object for changes, not just a property
  - By default, a person watcher will not fire if modifying person.name
  - it will fire only if it overrides the entire person property
  - In these cases you should use deep watch:

```
watch: {
    person: {
        handler: function (newValue, oldValue) {
            console.log(newValue, oldValue)
        },
        deep: true
    }
}
```

M01 - Introduction to Vue.js

Index

M01 - Introduction to Vue.js

7. Class and Style bindings

- Need for data binding through manipulation of html elements such as
  - classes of an html element
  - inline styles of an html element
- Since both are attributes, we can use v-bind to manipulate them
- Vue provides enhancements when v-bind is used with class and style
- In addition to strings, expressions can evaluate:
  - Objects
  - Arrays

M01 - Introduction to Vue.js

7. Class and Style bindings

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

```
<div v-bind:class='{active : isActive}'>
    ...
</div>
```

- The presence of the active class depends on the value of isActive property
- We can have multiple values in the class attribute

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

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

Final result after rendering!

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

M01 - Introduction to Vue.js

7. Class and Style bindings

- The binding object do not need to be inline

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

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

M01 - Introduction to Vue.js

7. Class and Style bindings

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

```
<div v-bind:class='[activeClass, errorClass]'>...</div>
```

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

Final result after rendering!

```
<div class='active text-danger'></div>
```

- You can also use object syntax within array syntax:

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

M01 - Introduction to Vue.js

7. Class and Style bindings

- The object syntax for v-bind:style is simple - almost like CSS, except it's a JS object
- For CSS property names you can use
  - the camelCase
  - kebab-case (use quotes)

```
<div v-bind:style='{color: activeColor, fontSize: fontSize + "px"}'>...</div>
```

Style binding

```
data: {
    activeColor: 'red',
    fontSize: 30
}
```

M01 - Introduction to Vue.js

7. Class and Style bindings

- Or bind to a style object directly to make the template cleaner

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

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

- The array syntax for v-bind:style lets you apply multiple style objects to the same element

```
<div v-bind:style='[baseStyles, overridingStyles]'>...</div>
```

M01 - Introduction to Vue.js

Index

M01 - Introduction to Vue.js

8. Conditional and list rendering

- Conditional rendering
  - The output is generated only when a condition is evaluated as truthy
  - The evaluation of the condition is made using the following directives:
    - v-if
    - v-else
    - v-else-if
    - v-show

M01 - Introduction to Vue.js

8. Conditional and list rendering

- Conditional rendering
  - To evaluate a simple condition use the directive v-if:

```
<h1 v-if='ok'>yes</h1>
...
data: { ok: true }
```

IF THE PROPERTY **OK** IS:

- 0
- FALSE
- “”
- NULL
- UNDEFINED
- NAN

THE **<h1>** IS NOT GENERATED!

  - The <h1> tag will be generated only if the ok property of the object data have any value rather than the values presented as false

M01 - Introduction to Vue.js

8. Conditional and list rendering

- Conditional rendering
  - Since the v-if is a directive, we need to bind it to a unique element. But if we want to relate to several elements?
  - In that case, we can use v-if in a <template> element, which will serve as an invisible container
  - The result rendered will not include the <template> element

```
<template v-if='ok'>
    <h1>Title</h1>
    <p>Paragraph 1</p>
    <p>Paragraph 2</p>
</template>
```

M01 - Introduction to Vue.js

8. Conditional and list rendering

- Conditional rendering
  - We can use the v-else directive to indicate na “else block” to v-if:

```
<div v-if="Math.random() > 0.5">
    Now you see me
</div>
<div v-else>
    Now you don't
</div>
```

  - A v-else element must be right after a v-if or v-else-if elements, otherwise it will not be recognized

M01 - Introduction to Vue.js

8. Conditional and list rendering

- Conditional rendering
  - The directive v-else-if serve as an "else if block" to v-if
  - It can be nested several times
  - Similar to v-else, a v-else-if element should appear right after a v-if element

```
<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>
```

M01 - Introduction to Vue.js

8. Conditional and list rendering

- Conditional rendering
  - Another option to show conditionally an element is with v-show
  - Its syntax is basic the same as v-if:

```
<h1 v-show="ok">yes</h1>
```

  - The main difference is that an element with a v-show will always be generated and will be included in the DOM
  - The v-show directive only alterns the CSS property for showing the element
  - It do not support the <template> element and v-else

M01 - Introduction to Vue.js

8. Conditional and list rendering

- Conditional rendering
  - v-if or v-show?
  - In general:
    - v-if has higher costs of changing
    - v-show has higher initial rendering costs
  - Conclusion:
    - prefer v-show if you need to altern something many times
    - prefer v-if in the case of the condition probably do not change in execution time

M01 - Introduction to Vue.js

8. Conditional and list rendering

- List rendering
  - With list rendering we can iterate over lists and present its values
  - The iteration is made using the v-for directive
  - Several iteration scenarios:
    - in arrays
    - in objects
    - in ranges
    - in <template>

M01 - Introduction to Vue.js

8. Conditional and list rendering

- List rendering (v-for with arrays)
  - Rendering a list of items based on an array
  - Special syntax: in the form of item in items, where:
    - items is the array with the origin data
    - item is an alias to the element of the array that is being iterated

```
const vm = new Vue({
    el: '#intro',
    data: {
        items: [
            { message: 'Foo' },
            { message: 'Bar' }
        ]
    }
})
```

```
<ul id='intro'>
    <li v-for='item in items'>
        {{ item.message }}
    </li>
</ul>
```

- Foo
- Bar

M01 - Introduction to Vue.js

8. Conditional and list rendering

- List rendering (v-for with arrays)
  - v-for also supports a second argument optional to the index of the actual item:

```
const vm = new Vue({
    el: '#intro',
    data: {
        parentMessage: 'Parent',
        items: [
            { message: 'Foo' },
            { message: 'Bar' }
        ]
    }
})
```

```
<ul id='intro'>
    <li v-for='(item, index) in items'>
        {{ parentMessage }} - {{ index }} - {{ item.message }}
    </li>
</ul>
```

- Parent - 0 - Foo
- Parent - 1 - Bar

M01 - Introduction to Vue.js

8. Conditional and list rendering

- List rendering (v-for with objects)
  - We also can use v-for to iterate over all the properties of an object

```
const vm = new Vue({
    el: '#intro',
    data: {
        object: {
            firstName: 'John',
            lastName: 'Doe',
            age: 30
        }
    }
})
```

```
<ul id='intro'>
    <li v-for='value in object'>
        {{ value }}
    </li>
</ul>
```

- John
- Doe
- 30

M01 - Introduction to Vue.js

8. Conditional and list rendering

- List rendering (v-for with objects)
  - We can include two more arguments: property name and index

```
const vm = new Vue({
    el: '#intro',
    data: {
        object: {
            firstName: 'John',
            lastName: 'Doe',
            age: 30
        }
    }
})
```

```
<ul id='intro'>
    <li v-for='(value, key, index) in object'>
        {{ index }}. {{ key }}: {{ value }}
    </li>
</ul>
```

0. firstName: John
1. lastName: Doe
2. age: 30

M01 - Introduction to Vue.js

8. Conditional and list rendering

- List rendering (v-for with ranges)
  - v-for can be used to iterate within a numeric range. In this case, it will repeat the template several times (the range number)

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

1 2 3 4 5 6 7 8 9 10

M01 - Introduction to Vue.js

8. Conditional and list rendering

- List rendering (v-for with ranges)
  - In order to show a filtered or sorted version of an array without change the original data, we should create a computed property that will return a processed array

```
data: {
    numbers: [1, 2, 3, 4, 5]
},
computed: {
    evenNumbers: function () {
        return this.numbers.filter(
            number => number % 2 === 0
        )
    }
}
```

```
<li v-for='n in evenNumbers'>{{ n }}</li>
```

- 2
- 4

M01 - Introduction to Vue.js

8. Conditional and list rendering

- List rendering (v-for with template)
  - Similar to the template in v-if, we can use the <template> with v-for to render a block with several elements. For instance:

```
<ul>
    <template v-for='item in items'>
        <li> {{ item.msg }} </li>
    </template>
</ul>
```

M01 - Introduction to Vue.js

8. Conditional and list rendering

- v-for with v-if
  - When they exist in the same node, v-for has priority higher than v-if
  - This means that v-if will be executed in each iteration of the loop separately
  - This can be useful when we desire to render only some elements:

```
<li v-for='todo in todos' v-if='!todo.isComplete'>
    {{ todo }}
</li>
```

- The above example only processes the todos which are not completed

M01 - Introduction to Vue.js

8. Conditional and list rendering

- v-for with v-if
  - If, our intention is to ignore conditionally the loop execution, we can put the v-if in a wrapper element (or <template>). For instance:

```
<ul v-if='todos.length'>
    <li v-for='todo in todos'>
        {{ todo }}
    </li>
</ul>
<p v-else>No todos left!</p>
```

M01 - Introduction to Vue.js

Index

M01 - Introduction to Vue.js

9. Event handling

- Using of the v-on directive to listen for DOM events and execute JavaScript when they fire

```
const vm = new Vue({
    el: '#intro',
    data: {
        counter: 0
    }
})
```

```
<div id='intro'>
    <button v-on:click='counter +=1'></button>
    <p>The button was clicked {{ counter }} times!</p>
</div>
```

M01 - Introduction to Vue.js

9. Event handling

- Writing JavaScript code in v-on attribute value is not feasible
- v-on also accepts the name of a method to be run when event triggers

```
<div id='intro'>
    <button v-on:click='greet'>Greet</button>
</div>
```

```
const vm = new Vue({
    el: '#intro',
    data: { name: 'Vue.js' },
    // Define methods in the methods object
    methods: {
        greet: function (event) {
            // `this` inside methods refers to the Vue instance
            alert('Hello ' + this.name + '!')
            // `event` is the DOM native event
            if (event) {
                alert(event.target.tagName)
            }
        }
    }
})
```

M01 - Introduction to Vue.js

9. Event handling

- Instead of binding directly to a method name, we can also use parameterized methods in an inline JavaScript statement:

```
const vm = new Vue({
    el: '#intro',
    methods: {
        say: function (message) {
            alert(message)
        }
    }
})
```

```
<div id='intro'>
    <button v-on:click='say("hello")'> Say hello</button>
    <button v-on:click='say("good morning")'>Say good morning </button>
</div>
```

M01 - Introduction to Vue.js

9. Event handling

- Sometimes we need to access the original DOM event in an inline instruction handler
- You can pass it in a method using the special $event variable:

```
const vm = new Vue({
    el: '#intro',
    methods: {
        warn: function (message, event) {
            if (event) event.preventDefault()
            alert(message)
        }
    }
})
```

```
<div id='intro'>
    <button v-on:click='warn("Form não pode ser submetido", $event)'>Submit</button>
</div>
```

M01 - Introduction to Vue.js

9. Event handling

- Event modifiers
  - Another common need is to call event.preventDefault() or event.stopPropagation() in event handlers.
  - Although is easy do this with methods, it's best if the methods were purely data logic rather than dealing with DOM event details
  - To address this issue, Vue provides event modifiers as v-on suffixes:
    - .stop
    - .prevent
    - .capture
    - .self
    - .once
    - .passive

M01 - Introduction to Vue.js

9. Event handling

- Event modifiers
  - Another common need event.stopPropagation(
  - Although is easy do this purely data logic rather
  - To address this issue, V
    - .stop
    - .prevent
    - .capture
    - .self
    - .once
    - .passive

```
<!-- click event propagation will stop -->
<a v-on:click.stop="doThis"></a>
<!-- submit event will not reload the page -->
<form v-on:submit.prevent="onSubmit"></form>
<!-- modifiers can be chained -->
<a v-on:click.stop.prevent="doThat"></a>
<!-- only the modifier -->
<form v-on:submit.prevent></form>
<!-- Use capture mode when adding event listener ->
<! - That is, an event that targets an inner element is
    handled here before it is handled by that element -->
<div v-on:click.capture="doThis">...</div>
<!-- only triggers the handler if event.target is the
    element itself, not a child element -->
<div v-on:click.self="doThat">...</div>
<!-- modifier works only once -->
<div v-on:click.once="doThat">...</div>
```

M01 - Introduction to Vue.js

9. Event handling

- Key modifiers
  - When listening to keyboard events, we need to check key codes
  - Vue lets you add v-on keyboard modifiers when listening to key events
  - See the completed list of modifier alias:
    - .enter
    - .tab
    - .esc
    - .space
    - .delete (capture bothe keys “Delete” and “Backspace”)
    - .up, .down, .left e .right

```
<!-- Call submit if the key presses is the ENTER -->
<input v-on:keyup.enter='submit'>

<!-- Abreviated form -->
<input @keyup.enter='submit'>
```

M01 - Introduction to Vue.js

9. Event handling

- System 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

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

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

M01 - Introduction to Vue.js

9. Event handling

- System modifiers keys
  - The .exact modifier allows you to control the exact combination of system modifiers needed to trigger an event

```
<!-- fires even if Alt or Shift is pressed -->
<button @click.ctrl="onClick">A</button>

<!--  only fires when Ctrl and no other key is pressed -->
<button @click.ctrl.exact="onCtrlClick">A</button>

<!-- fires when no System modifier is pressed  -->
<button @click.exact="onClick">A</button>
```

M01 - Introduction to Vue.js

9. Event handling

- Mouse modifiers
  - Mouse modifiers restrict the handler to events triggered by a specific mouse button:
    - .left
    - .right
    - .middle

M01 - Introduction to Vue.js

Index

M01 - Introduction to Vue.js

10. Form input bindings

- To be continued...