Introduction to Vue.js

Assoc. Prof. Dr. Kanda Runapongsa Saikaew
(krunapon@kku.ac.th)
Department of Computer Engineering
Khon Kaen University

What is Vue.js?

- Vue (pronounced /vjuː/, like **view**) is a **progressive framework** for building user interfaces
- The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects
- On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with **modern tooling** and **supporting libraries**.

Hello Vue.js!

- https://jsfiddle.net/chrisvfritz/50wL7mdz/

HTML ▼

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

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

```

CSS ▼

JavaScript + No-Library (pure JS) ▼

```
new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue.js!'
  }
})
```

Hello Vue.js!

Declarative Rending

- At the core of Vue.js is a system that enables us to declaratively render data to the DOM using straightforward template syntax

HTML

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

JS

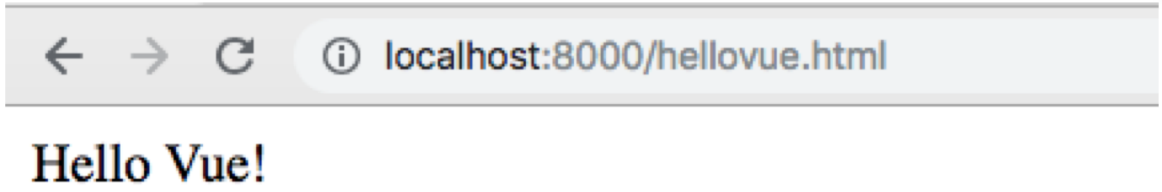
```js
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
})
```

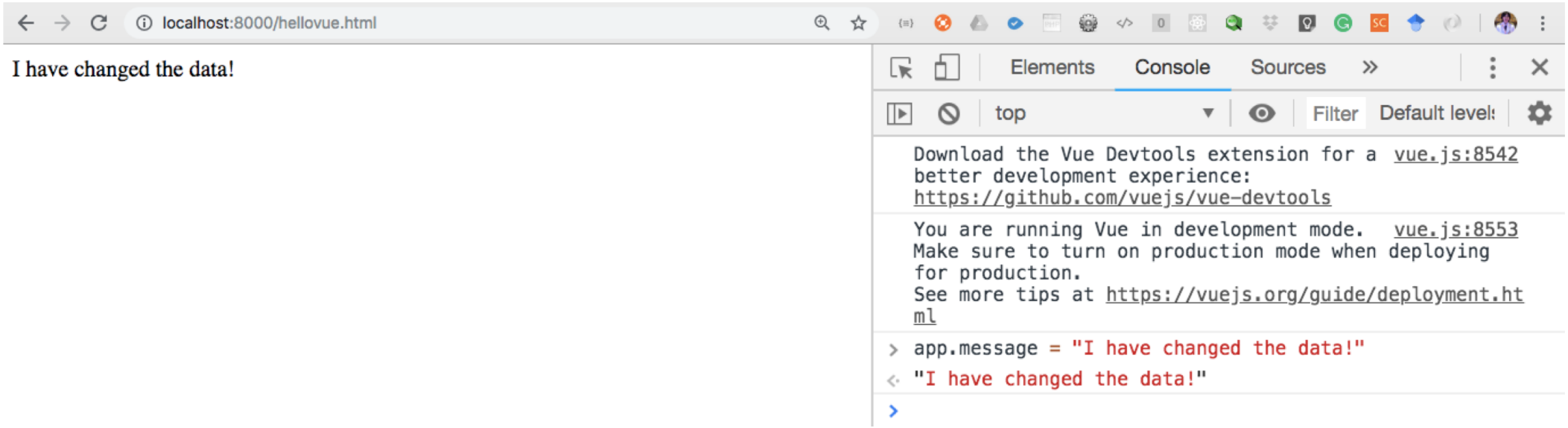
Hello Vue!

Reactive

- The data and the DOM are now linked, and everything is now **reactive**
- How do we know? Open your browser's JavaScript console and set app.message to a different value
- You should see the rendered example above update accordingly.

Ex1: hellovue.html





Binding element attributes

HTML

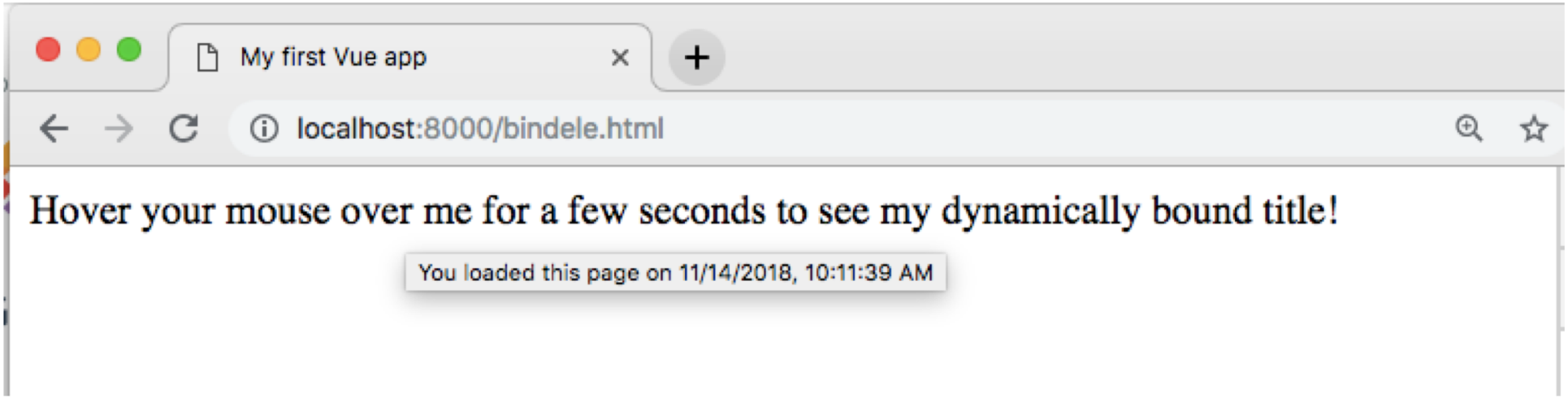
```html
<div id="app-2">
  <span v-bind:title="message">
    Hover your mouse over me for a few seconds
    to see my dynamically bound title!
  </span>
</div>
```

JS

```js
var app2 = new Vue({
  el: '#app-2',
  data: {
    message: 'You loaded this page on ' + new Date().toLocaleString()
  }
})
```

Hover your mouse over me for a few seconds to see my dynamically bound title!

The result of binding element attributes



Directive

- The v-bind attribute you are seeing is called a **directive**
- Directives are prefixed with v- to indicate that they are special attributes provided by Vue
  - They apply special reactive behavior to the rendered DOM
  - Here, it is basically saying “keep this element’s title attribute up-to-date with the message property on the Vue instance

Conditionals

HTML

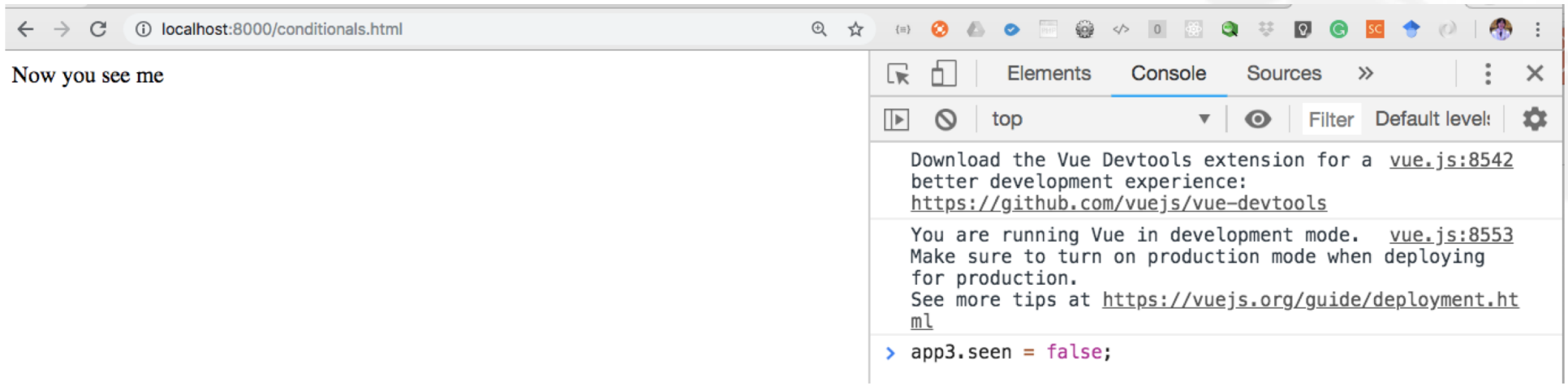
```
<div id="app-3">
  <span v-if="seen">Now you see me</span>
</div>
```

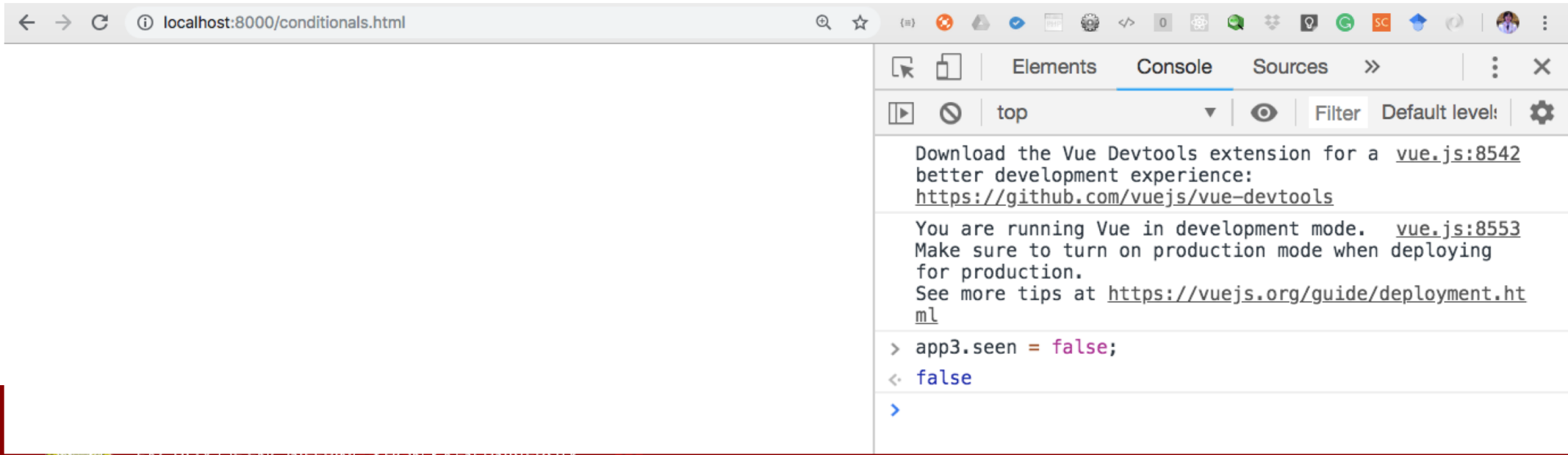
JS

```
var app3 = new Vue({
  el: '#app-3',
  data: {
    seen: true
  }
})
```

Now you see me

Using conditionals





Loops

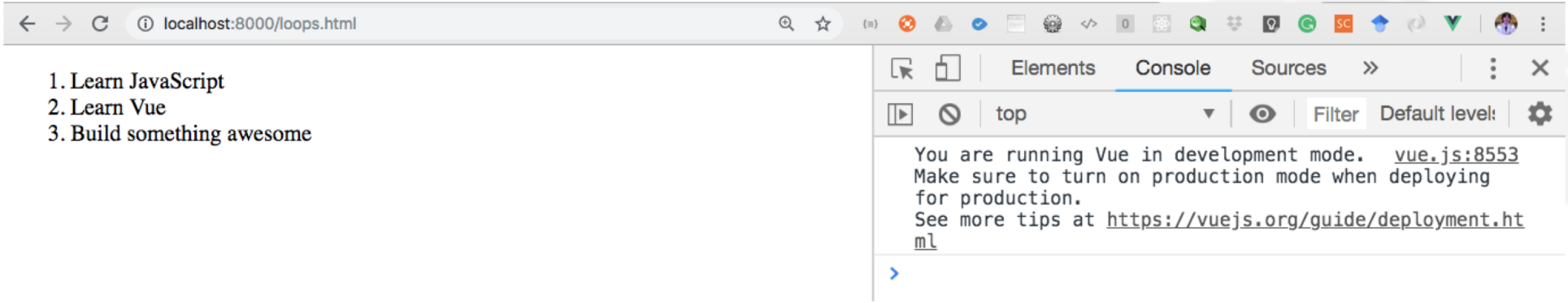
- The v-for directive can be used for displaying a list of items using the data from an Array

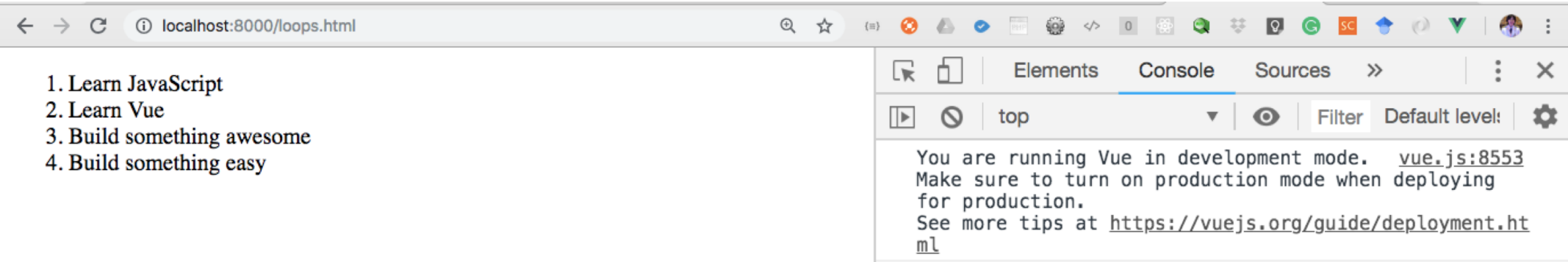
```html
<div id="app-4">
  <ol>
    <li v-for="todo in todos">
      {{ todo.text }}
    </li>
  </ol>
</div>
```

```
JS
var app4 = new Vue({
  el: '#app-4',
  data: {
    todos: [
      { text: 'Learn JavaScript' },
      { text: 'Learn Vue' },
      { text: 'Build something awesome' }
    ]
  }
})
```

1. Learn JavaScript
2. Learn Vue
3. Build something awesome

Ex 2: Update an element in loops dynamically





Handling user input

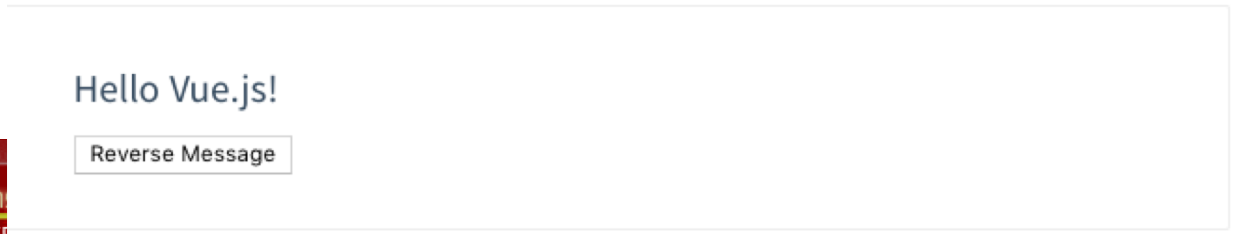
- To let users interact with your app, we can use the v-on directive to attach event listeners that invoke methods on our Vue instances:

HTML

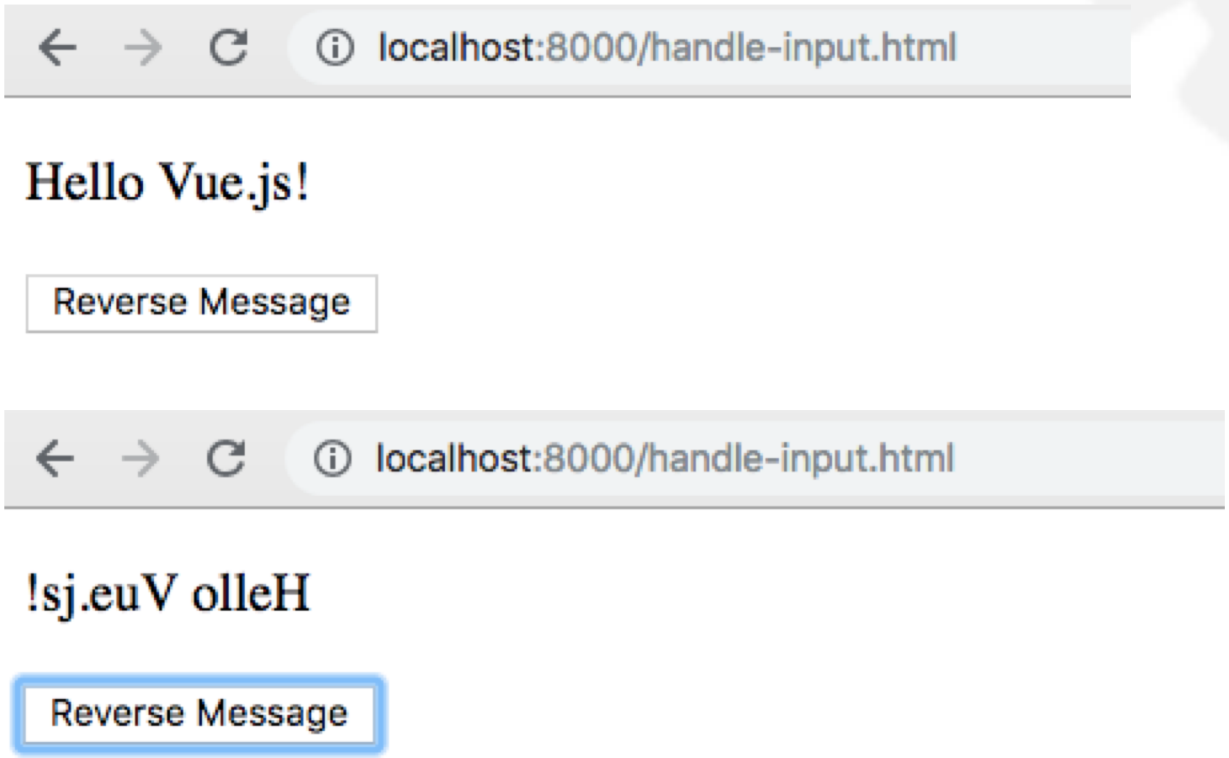
```
<div id="app-5">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">Reverse Message</button>
</div>
```

JS

```
var app5 = new Vue({
  el: '#app-5',
  data: {
    message: 'Hello Vue.js!'
  },
  methods: {
    reverseMessage: function () {
      this.message = this.message.split('').reverse().join('')
    }
  }
})
```



The result of handling user input



Note that in this method we update the state of our app without touching the DOM - all DOM manipulations are handled by Vue, and the code you write is focused on the underlying logic.

Two-way binding between form input and app

- Vue also provides the –model directive that makes two-way binding between form input and app state a breeze

HTML

```
<div id="app-6">
  <p>{{ message }}</p>
  <input v-model="message">
</div>
```

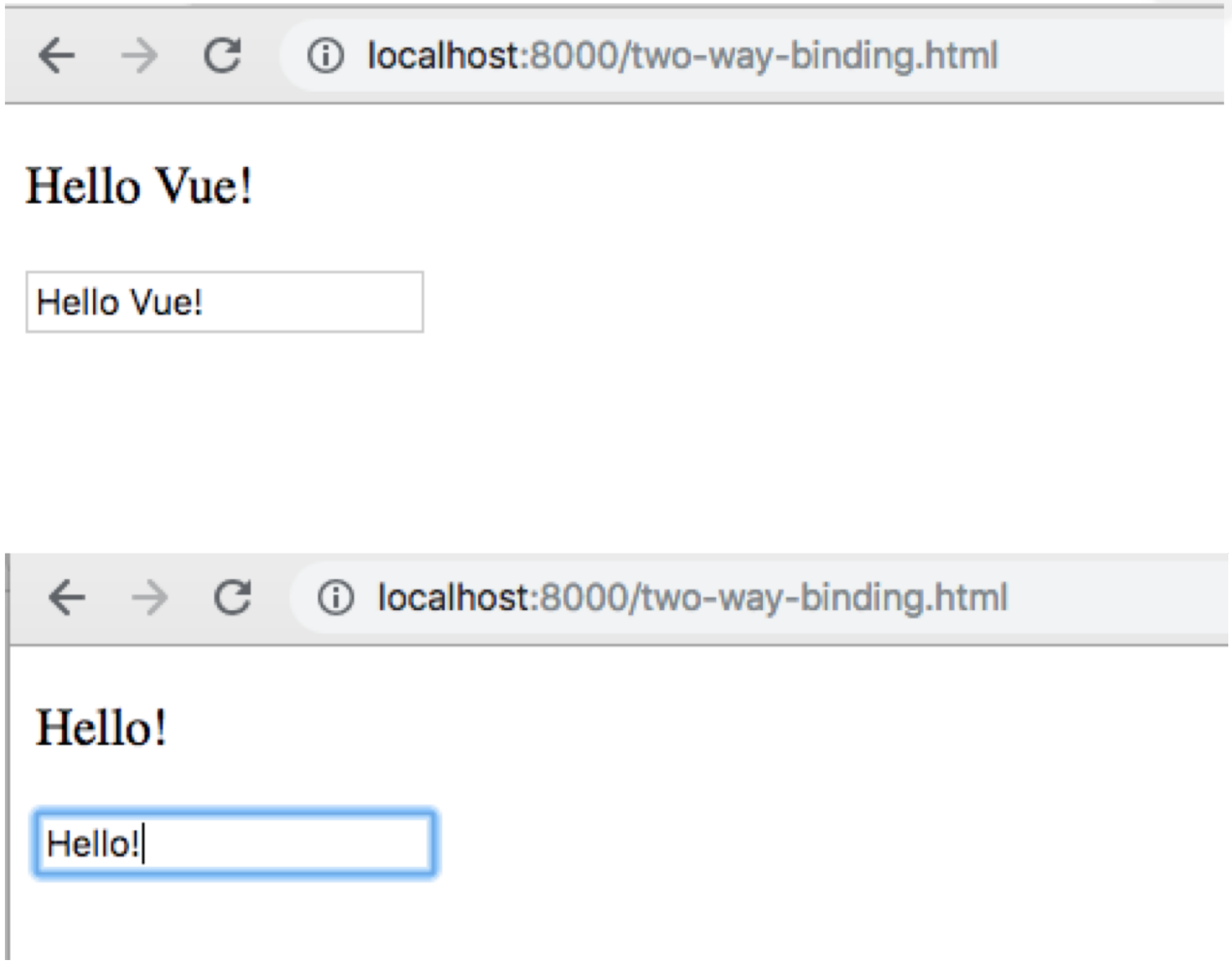
JS

```
var app6 = new Vue({
  el: '#app-6',
  data: {
    message: 'Hello Vue!'
  }
})
```

Good bye

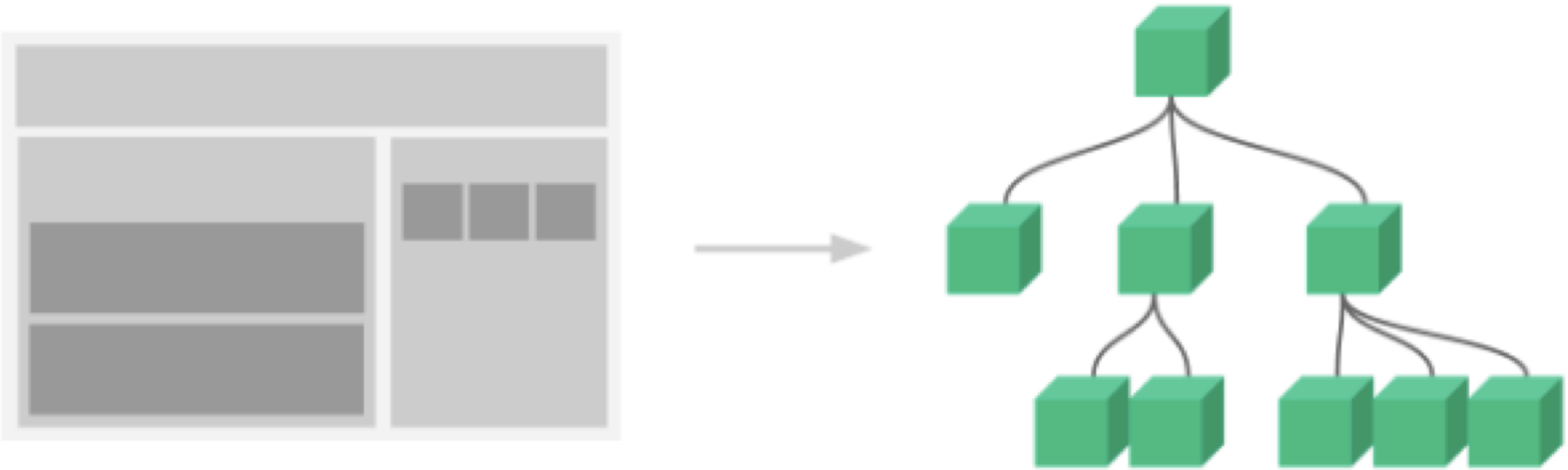
Good bye

The result of two-way binding

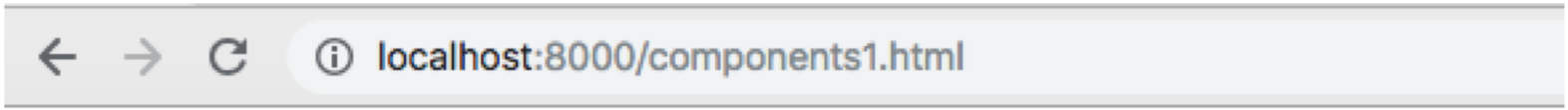


Composing with components

- The component system is another important concept in Vue, because it’s an abstraction that allows us to build large-scale applications composed of small, self-contained, and often reusable components



```html
<!DOCTYPE html>
<html>
    <head>
        <title>My first Vue app</title>
        <script src="https://unpkg.com/vue"></script>
    </head>
    <body>
        <div id="app">
            <ol>
                <todo-item></todo-item>
            </ol>
        </div>
        <script>
            Vue.component('todo-item', {
                template: '<li>This is a todo</li>'
            })
            var app = new Vue({
                el: '#app'
            })
        </script>
    </body>
</html>
```



1. This is a todo

Components with props

- We should be able to pass data from the parent scope into child components
- Let's modify the component definition to make it accept a **prop**:

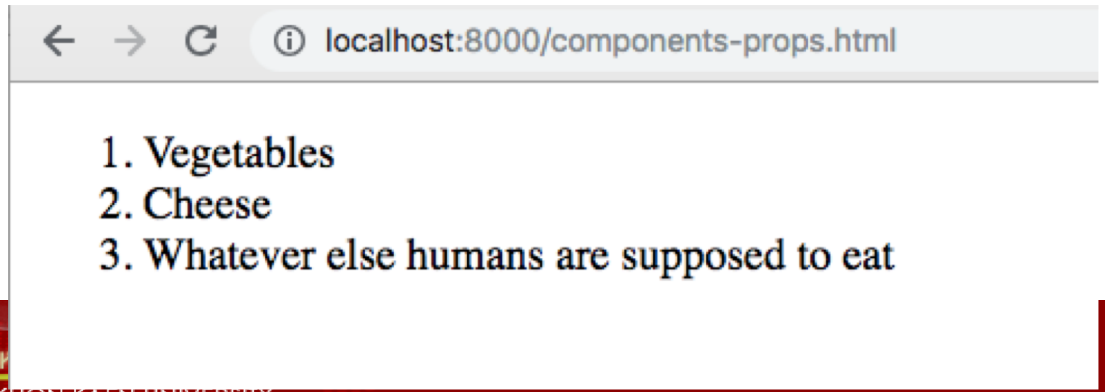
```JS
Vue.component('todo-item', {
  // The todo-item component now accepts a
  // "prop", which is like a custom attribute.
  // This prop is called todo.
  props: ['todo'],
  template: '<li>{{ todo.text }}</li>'
})
```

Now we can pass the todo into each repeated component using `v-bind` :

```HTML
<div id="app-7">
  <ol>
    <!--
      Now we provide each todo-item with the todo object
      it's representing, so that its content can be dynamic.
      We also need to provide each component with a "key",
      which will be explained later.
    -->
    <todo-item
      v-for="item in groceryList"
      v-bind:todo="item"
      v-bind:key="item.id">
    </todo-item>
  </ol>
</div>
```

Results of components with props

```
<div id="app">
    <ol>
        <todo-item
            v-for="item in groceryList"
            v-bind:todo="item"
            v-bind:key="item.id">
        </todo-item>
    </ol>
</div>
<script>
    Vue.component('todo-item', {
        props: ['todo'],
        template: '<li>{{ todo.text }}</li>'
    })
    var app = new Vue({
        el: '#app',
        data: {
            groceryList: [
                {id: 0, text: 'Vegetables'},
                {id: 1, text: 'Cheese'},
                {id: 2, text: 'Whatever else humans are supposed to eat'}
            ]
        }
    })
</script>
```



Creating a vue instance

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

```JS
var vm = new Vue({
  // options
})
```

- A Vue application consists of a **root Vue instance** created with new Vue, optionally organized into a tree of nested, reusable components.

Data and methods

- 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

```JS
// Our data object
var data = { a: 1 }

// The object is added to a Vue instance
var vm = new Vue({
  data: data
})

// Getting the property on the instance
// returns the one from the original data
vm.a == data.a // => true
```

Template syntax

- Vue.js uses an HTML-based template syntax that allows you to declaratively bind the rendered DOM to the underlying Vue instance's data
- All Vue.js templates are valid HTMl that can be parsed by spec-compliant browsers and HTML parsers
- Vue is able to intelligently figure out the minimal number of components to re-render and apply the minimal amount of DOM manipulations when the app state changes

#text

- The most basic form of data binding is text interpolation using the “Mustache” syntax (double curly braces)

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

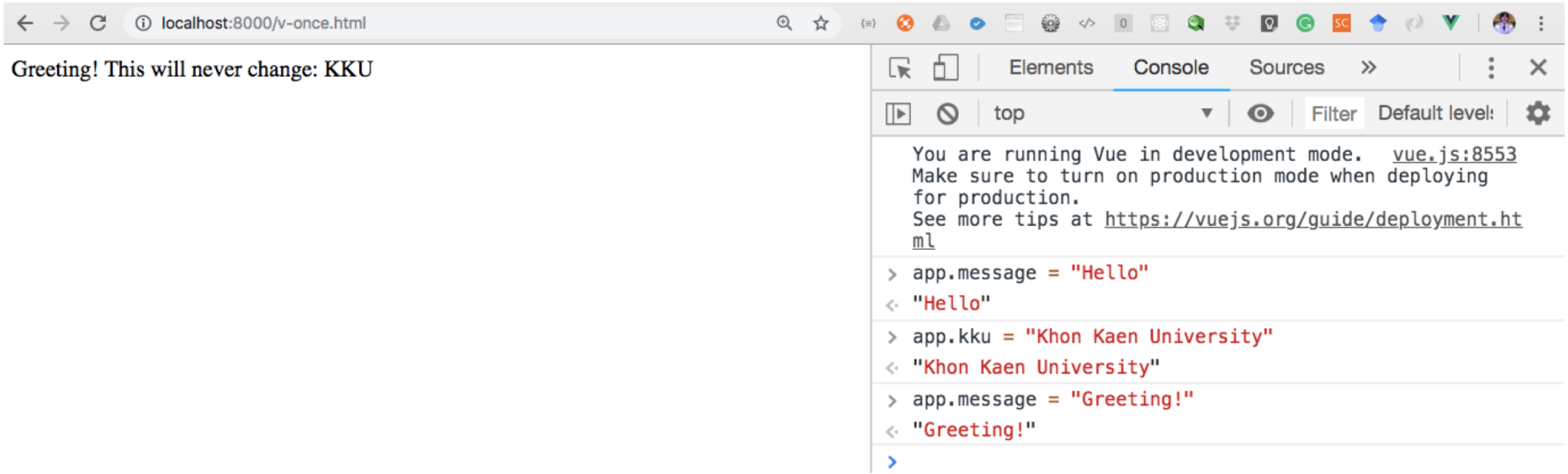
- The mustache tag will be replaced with the value of msg property on the corresponding data object
- It will also be updated whenever the data object’s msg property changes

v-once directive

- You can also perform one-time interpolations that do not update on data change by using the **v-once directive**, but keep in mind this will also affect any other bindings on the same node:

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

Ex 3: Using v-once directive



References

- https://vuejs.org/v2/guide/#