

Corso su vue su Udemy

The screenshot shows a code editor with two panes. The left pane contains the following HTML code:

```
<script src="https://unpkg.com/vue/dist/vue.js"></script>  HTML ⚙  
  
<div id="app">  
  <p>{{ title }}</p>  
</div>
```

The right pane shows the rendered output: "Hello World!".

The left pane also contains the following JavaScript code:

```
new Vue({  
  el: '#app',  
  data: {  
    title: 'Hello World!'  
  }  
});  JAVASCRIPT ⚙
```

Qui sopra abbiamo creato una proprietà che viene eseguita da Vue solo se viene chiamata dall'html

Vue crea un modello basato sul nostro codice html lo memorizza internamente e lo inietta nel dom

The screenshot shows a code editor with two panes. The left pane contains the following HTML code:

```
1 <script src="https://unpkg.com/vue/dist/vue.js"></script>  HTML ⚙  
2  
3 <div id="app">  
4   <p>{{ sayHello() }}</p>  
5 </div>
```

The right pane shows the rendered output: "Hello World".

The left pane also contains the following JavaScript code:

```
1 new Vue({  
2   el: '#app',  
3   data: {  
4     title: 'Hello World!'  
5   },  
6   methods: {  
7     sayHello: function() {  
8       return 'Hello!';  
9     }  
10    }  
11  });  JAVASCRIPT ⚙
```

Qui abbiamo creato un metodo (che viene eseguito sempre su esecuzione della pagina, anche se non viene chiamato)

The screenshot shows a browser developer tools interface with two panes. The left pane contains Vue.js code:

```

1 <script src="https://unpkg.com/vue/dist/vue.js"></script>  HTML ⚙
2
3 <div id="app">
4   <p>{{ sayHello() }}</p>
5 </div>

```

The right pane shows the rendered output: "Hello World!"

The left pane contains the same Vue.js code, but the 11th line is highlighted in red:

```

1 new Vue({
2   el: '#app',
3   data: {
4     title: 'Hello World!'
5   },
6   methods: {
7     sayHello: function() {
8       return this.title;
9     }
10   }
11 });

```

The right pane shows the rendered output: "Hello World!"

Per riferirsi ad una property da una funzione basterà aggiungere la parolina `this` davanti, in javascript non funzionerebbe una sintassi così ma con Vue funziona in quanto vue crea un proxy attorno alle proprietà in modo che possano essere chiamate da un oggetto esterno come se fossero appartenenti allo stesso oggetto da cui si chiamano.

The screenshot shows a browser developer tools interface with two panes. The left pane contains Vue.js code:

```

1 <script src="https://unpkg.com/vue/dist/vue.js"></script>  HTML ⚙
2
3 <div id="app">
4   <p>{{ sayHello() }} - <a href="{{ link }}">Google</a></p>
5 </div>

```

The right pane shows the rendered output: "Hello World! - Google".

The left pane contains the same Vue.js code, but the 8th line is highlighted in red:

```

1 new Vue({
2   el: '#app',
3   data: {
4     title: 'Hello World!',
5     link: 'http://google.com'
6   },
7   methods: {
8     sayHello: f html, quindi questo non funziona.
9       return this.title;
10    }
11 });

```

A tooltip appears over the red-highlighted line: "html, quindi questo non funziona."

The right pane shows the rendered output: "Hello World! - Google".

Se impostiamo una proprietà nell'attributo href non funziona in quanto introduce una codifica nel link che sporca lo stesso, ogni qualvolta usiamo la doppia parentesi grafa in un elemento html non funzionerà...per ottenere un link funzionante occorre usare la seguente sintassi:

```

1 <script src="https://unpkg.com/vue/dist/vue.js"></script> HTML
2
3 <div id="app">
4   <p>{{ sayHello() }} - <a v-bind:href="link">Google</a></p>
5 </div>

```

```

1 new Vue({
2   el: '#app',
3   data: {
4     title: 'Hello World!',
5     link: 'http://google.com'
6   },
7   methods: {
8     sayHello: function() {
9       return this.title;
10    }
11  }
12 });

```

Qui viene introdotta la direttiva v-bind che dice a vue che la proprietà dopo i due punti è una proprietà dinamica di vue.

```

1 <script src="https://unpkg.com/vue/dist/vue.js"></script> HTML
2
3 <div id="app">
4   <h1 v-once>{{ title }}</h1>
5   <p>{{ sayHello() }} - <a v-bind:href="link">Google</a></p>
6 </div>

```

```

1 new Vue({
2   el: '#app',
3   data: {
4     title: 'Hello World!',
5     link: 'http://google.com'
6   },
7   methods: {
8     sayHello: function() {
9       this.title = 'Hello!';
10      return this.title;
11    }
12  }

```

La direttiva v-once serve per far sì che vue non sovrascriva il valore di una property se questa è reimpostata successivamente per esempio in una funzione.

```

1 <script src="https://unpkg.com/vue/dist/vue.js"></script> HTML
2
3 <div id="app">
4   <h1 v-once>{{ title }}</h1>
5   <p>{{ sayHello() }} - <a v-bind:href="link">Google</a></p>
6   <hr>
7   <p>{{ finishedLink }}</p>
8 </div>

```

```

1 new Vue({
2   el: '#app',
3   data: {
4     title: 'Hello World!',
5     link: 'http://google.com',
6     finishedLink: '<a href="http://google.com">Google</a>'
7   },
8   methods: {
9     sayHello: function() {
10       this.title = 'Hello!';
11      return this.title;
12    }
13  }

```

Vue per default non rappresenta codice html come html ma lo rappresenta in testo, questo è buono in quanto previene gli attacchi cross-site-scripting da fonti html che potrebbero essere non sicure.

```

1 <script src="https://unpkg.com/vue/dist/vue.js"></script> HTML
2
3 <div id="app">
4   <h1 v-once>{{ title }}</h1>
5   <p>{{ sayHello() }} - <a v-bind:href="link">Google</a></p>
6   <hr>
7   <p v-html="finishedLink"></p>
8 </div>

```

```

1 new Vue({
2   el: '#app',
3   data: {
4     title: 'Hello World!',
5     link: 'http://google.com',
6     finishedLink: '<a href="http://google.com">Google</a>'
7   },
8   methods: {
9     sayHello: function() {
10       this.title = 'Hello!';
11     }
12   }
13 });

```

Se sai che la fonte html da rappresentare è pulita puoi usare la direttiva v-html come sopra.

```

1 <script src="https://unpkg.com/vue/dist/vue.js"></script> HTML
2
3 <div id="app">
4   <button v-on:click="increase">Click me</button>
5   <p>{{ counter }}</p>
6 </div>

```

```

1 new Vue({
2   el: '#app',
3   data: {
4     counter: 0
5   },
6   methods: {
7     increase: function() {
8       this.counter++;
9     }
10   }
11 });

```

Per associare la gestione di un evento ad un metodo del modello vue usiamo la direttiva v-on seguita da un qualsiasi evento di default che supporti l'elemento corrente.

```

1 <script src="https://unpkg.com/vue/dist/vue.js"></script> HTML
2
3 <div id="app">
4   <button v-on:click="increase">Click me</button>
5   <p>{{ counter }}</p>
6   <p v-on:mousemove="updateCoordinates">Coordinates: {{ x }} / {{ y }}</p>
7 </div>

```

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

In questo esempio viene mostrato come recuperare le info passate dall'evento in vue per poi generare un output sempre attraverso vue. In questo caso vengono recuperate le coordinate dall'evento del moseclick e stampate a video.

```
1 <script src="https://unpkg.com/vue/dist/vue.js"></script> HTML
2
3 <div id="app">
4   <button v-on:click="increase(2)">Click me</button>
5   <p>{{ counter }}</p>
6   <p v-on:mousemove="updateCoordinates">Coordinates: {{ x }} / {{ y }}</p>
7 </div>
8
9
10    x: 0,
11    y: 0
12  },
13  methods: {
14    increase: function(step) {
15      this.counter += step;
16    }
17  }
18
```

Click me

16

Coordinates: 0 / 0

In questo caso sopra si mostra come fare a passare le proprie informazioni all'evento.

```
1 <script src="https://unpkg.com/vue/dist/vue.js"></script> HTML ⚡  
2  
3 <div id="app">  
4   <button v-on:click="increase(2, $event)">Click me</button>  
5   <p>{{ counter }}</p>  
6   <p v-on:mousemove="updateCoordinates">Coordinates: {{ x }} / {{ y }}  
7 </p>  
8 </div>  
  
5     x: 0,  
6     y: 0  
7   },  
8   methods: {  
9     increase: function(step, event) {  
10       this.counter += step;  
11     },  
12     updateCoordinates: function(event) {  
13       this.x = event.clientX;  
14       this.y = event.clientY;  
15     }  
16   }  
17 <div>Coordinates: {{ x }} / {{ y }}</div>  
18  
19 <div>Click me</div>  
20  
21  
22  
23  
24  
25 Coordinates: 0 / 0
```

Per recuperare nel metodo di vue oltre ai dati passati da noi all'evento anche i dati predefiniti dell'evento usiamo la sintassi particolare \$event.



```
<script src="https://unpkg.com/vue/dist/vue.js"></script>  HTML ⚡  
1  
2 <div id="app">  
3   <button v-on:click="increase(2, $event)">Click me</button>  
4   <p>{{ counter }}</p>  
5   <p v-on:mousemove="updateCoordinates">  
6     Coordinates: {{ x }} / {{ y }}  
7     - <span v-on:mousemove='dummy'>DEAD SPOT</span>  
8   </p>  
9 </div>  
10  
11  ,  
12  methods: {  
13    increase: function(step, event) {  
14      this.counter += step;  
15    },  
16    updateCoordinates: function(event) {  
17      this.x = event.clientX;  
18      this.y = event.clientY;  
19    },  
20    dummy: function(event) {  
21      event.stopPropagation();  
22    }  
23  }  
24  ,  
25  JAVASCRIPT ⚡  
26  0  
27  Coordinates: 188 / 80 - DEAD SPOT
```

In questo esempio viene mostrato come fermare la propagazione nello span DEAD SPOT altrimenti anche il mouseover su questo span avrebbe continuato a scatenare l'evento updateCoordinates.

```
<script src="https://unpkg.com/vue/dist/vue.js"></script> HTML ⚙  
  
<div id="app">  
  <button v-on:click="increase(2, $event)">Click me</button>  
  <p>{{ counter }}</p>  
  <p v-on:mousemove="updateCoordinates">  
    Coordinates: {{ x }} / {{ y }}  
    - <span v-on:mousemove.stop.prevent="">DEAD SPOT</span>  
  </p>  
</div>
```

Un modo alternativo più sintetico per fare la stessa cosa consiste nell'eliminare la funzione dummy ed usare l'**event modifier** mousemove.stop (sintetico) oppure mouse.move.prevent. Questo tipo di sintassi è

utile sia per prevenire la propagazione degli eventi sia per prevenire il default degli eventi, due casistiche comuni in javascript.

```

1 <div id="app">
2   <button v-on:click="increase(2, $event)">Click me</button>
3   <button v-on:click="counter++>>Click me</button>
4   <p>{{ counter * 2 }}</p>
5   <p v-on:mousemove="updateCoordinates">
6     Coordinates: {{ x }} / {{ y }}
7     - <span v-on:mousemove.stop="">DEAD SPOT</span>
8   </p>
9   <input type="text" v-on:keyup.enter.space="alertMe">
10
11 </div>
12
13 <script>
14   new Vue({
15     el: '#app',
16     data: {
17       count: 0,
18       counter: 0,
19       coordinates: { x: 0, y: 0 }
20     },
21     methods: {
22       increase: function(step, event) {
23         this.counter += step;
24       },
25       updateCoordinates: function(event) {
26         this.coordinates.x = event.clientX;
27         this.coordinates.y = event.clientY;
28       },
29       alertMe: function() {
30         alert('Alert!');
31       }
32     }
33   })
34 </script>

```

Esistono anche i key modifier, ad esempio enter.space scatena il metodo alertMe solo se la pressione del tasto è uno spazio.

```

1 <div id="app">
2   <button v-on:click="increase(2, $event)">Click me</button>
3   <button v-on:click="counter++>>Click me</button>
4   <p>{{ counter * 2 }}</p>
5   <p v-on:mousemove="updateCoordinates">
6     Coordinates: {{ x }} / {{ y }}
7     - <span v-on:mousemove.stop="">DEAD SPOT</span>
8   </p>
9   <input type="text" v-on:keydown.space="alertMe">
10
11 </div>
12
13 <script>
14   new Vue({
15     el: '#app',
16     data: {
17       count: 0,
18       counter: 0,
19       coordinates: { x: 0, y: 0 }
20     },
21     methods: {
22       increase: function(step, event) {
23         this.counter += step;
24       },
25       updateCoordinates: function(event) {
26         this.coordinates.x = event.clientX;
27         this.coordinates.y = event.clientY;
28       },
29       alertMe: function() {
30         alert('Alert!');
31       }
32     }
33   })
34 </script>

```

In qualsiasi punto dove è stato inserito del riferimento a vue nel dom possiamo scrivere semplice javascript su una riga. Ad esempio per incrementare il contatore come sopra. Si noti come il javascript venga inserito sia nell'elemento che fuori.

```

1 <script src="https://unpkg.com/vue/dist/vue.js"></script>  HTML
2
3 <div id="app">
4   <input type="text" v-model="name">
5   <p>{{ name }}</p>
6 </div>
7
8 <script>
9   new Vue({
10     el: '#app',
11     data: {
12       name: 'Max'
13     }
14   })
15 </script>

```

Per implementare il binding bidirezionale occorre usare la direttiva v-model.

Cosa succede se devo aggiornare una variabile che dipende da più comportamenti sulla pagina? Ad esempio se voglio stampare se maggiore di 5 o minore di 5 una variabile che viene incrementata o decrementata da due bottoni occorre duplicare il codice nell'increase e decrease methods, e questo non è buono per il mantenimento:



```

1 <script src="https://unpkg.com/vue/dist/vue.js"></script> HTML
2
3 <div id="app">
4   <button v-on:click="increase">Increase</button>
5   <button v-on:click="decrease">Decrease</button>
6   <p>Counter: {{ counter }}</p>
7   <p>Result: {{ result }}</p>
8 </div>
9
10 data: {
11   counter: 0,
12   result: ''
13 },
14 methods: {
15   increase: function() {
16     this.counter++;
17     this.result = this.counter > 5 ? 'Greater 5' : 'Smaller 5'
18   },
19   decrease: function() {
20     this.counter--;
21     this.result = this.counter > 5 ? 'Greater 5' : 'Smaller 5'
22   }
23 }

```

In alternativa si può optare per una soluzione così usando un unico metodo:



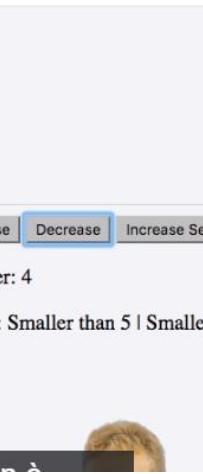
```

1 <script src="https://unpkg.com/vue/dist/vue.js"></script> HTML
2
3 <div id="app">
4   <button v-on:click="counter++">Increase</button>
5   <button v-on:click="counter--">Decrease</button>
6   <p>Counter: {{ counter }}</p>
7   <p>Result: {{ result() }}</p>
8 </div>
9
10 new Vue({
11   el: '#app',
12   data: {
13     counter: 0
14   },
15   methods: {
16     result() {
17       return this.counter > 5 ? 'Greater 5' : 'Smaller than 5'
18     }
19   }
20 });

```

che ora funziona di nuovo.

Qui il codice non è duplicato ma il metodo viene eseguito sempre anche se la variabile counter non subisce modifiche. La cosa potrebbe non essere performante per cui si usa questa terza soluzione:



```

1 <script src="https://unpkg.com/vue/dist/vue.js"></script> HTML
2
3 <div id="app">
4   <button v-on:click="counter++">Increase</button>
5   <button v-on:click="counter--">Decrease</button>
6   <button v-on:click="secondCounter++">Increase Second</button>
7   <p>Counter: {{ counter }}</p>
8   <p>Result: {{ result() }} | {{ output }}</p>
9 </div>
10
11 data: {
12   counter: 0,
13   secondCounter: 0
14 },
15 computed: {
16   output: function() {
17     return this.counter > 5 ? 'Greater 5' : 'Smaller than 5';
18   }
19 }

```

una proprietà calcolata è concepibile che l'output non è

Qui ad esempio la pressione su Increase second scatena di nuovo l'esecuzione del metodo result anche se increase second incrementa la proprietà secondcounter. Per evitare che ciò avvenga si usa un metodo computed che si definisce come metodo ma viene usato come proprietà (si veda come viene chiamato nel dom esempio sopra. Le proprietà computed chiedono i valori).

```

4 <button v-on:click="counter++>Increase</button>
5 <button v-on:click="counter-->Decrease</button>
6 <button v-on:click="secondCounter++>Increase Second</button>
7 <p>Counter: {{ counter }} | {{ secondCounter }}</p>
8 <p>Result: {{ result() }} | {{ output }}</p>
9 </div>
10
11 counter: 0,
12 secondCounter: 0
13 },
14 computed: {
15   output: function() {
16     console.log('Computed');
17     return this.counter > 5 ? 'Greater 5' : 'Smaller than 5';
18   }
19 },
20 watch: {
21   counter: function(value) {
22     var vm = this;
23     setTimeout(function() {
24       vm.counter = 0;
25     }, 2000);
26   }
27 }

```

il contatore

Altre volte si ha l'esigenza di far reagire il codice al variare di una proprietà, nel caso sopra la modifica del contatore scatenerà la callback che dopo due secondi azzererà il contatore.

Esistono delle scorciatoie sintattiche per rappresentare il binding a proprietà ed eventi, normalmente scriviamo per esteso:

```

1 <script src="https://unpkg.com/vue/dist/vue.js"></script> HTML
2
3 <div id="app">
4   <button v-on:click="changeLink">Click to Change Link</button>
5   <a v-bind:href="link">Link</a>
6 </div>

```

Ma potremo scrivere anche:

```

<div id="app">
  <button @click="changeLink">Click to Change Link</button>
  <a :href="link">Link</a>
</div> [

```

La funzionalità è identica.

```

1 <script src="https://unpkg.com/vue/dist/vue.js"></script> HTML
2
3 <div id="app">
4   <div
5     class="demo"
6     @click="attachRed = !attachRed"
7     :class="{'red': attachRed}"></div>
8   <div class="demo"></div>
9   <div class="demo"></div>
10 </div> [
11
12 new Vue({
13   el: '#app',
14   data: {
15     attachRed: false
16   }
17 })

```

Cambia il colore applicando la classe red su click del div e viceversa

Il codice successivo mostra come impostare una property vue (divClasses) delle classi css mediante un oggetto che viene ritornato dalla computed property.

```
1 <script src="https://unpkg.com/vue/dist/vue.js"></script> HTML
2
3 <div id="app">
4   <div
5     class="demo"
6     @click="attachRed = !attachRed"
7     :class="divClasses"></div>
8   <div class="demo" :class="{red: attachRed}"></div>
9   <div class="demo"></div>
10 </div>
```



```
1 new Vue({
2   el: '#app',
3   data: {
4     attachRed: false
5   },
6   computed: {
7     divClasses: function() {
8       return {
9         red: this.attachRed,
10        blue: !this.attachRed
11      };
12    }
13  }
14});
```

Quindi abbiamo due sintassi usando o la classe direttamente o usando un oggetto dove la classe è la chiave ed il valore la condizione (come vediamo dall'esempio successivo).

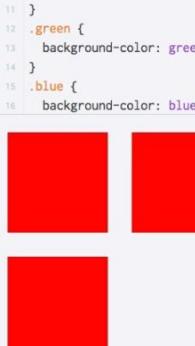
E' possibile anche passare un array di classi:

```
2
3 <div id="app">
4   <div
5     class="demo"
6     @click="attachRed = !attachRed"
7     :class="divClasses"></div>
8   <div class="demo" :class="{red: attachRed}"></div>
9   <div class="demo" :class="[{color, {red: attachRed}}]"></div>
10  <br>
11  <input type="text" v-model="color">
12 </div>
13
14 new Vue({
15   el: '#app',
16   data: {
17     attachRed: false,
18     color: 'green'
19   },
20   computed: {
21     divClasses: function() {
22       return {
23         red: this.attachRed,
24         blue: !this.attachRed
25       }
26     }
27   }
28 })
```

HTML

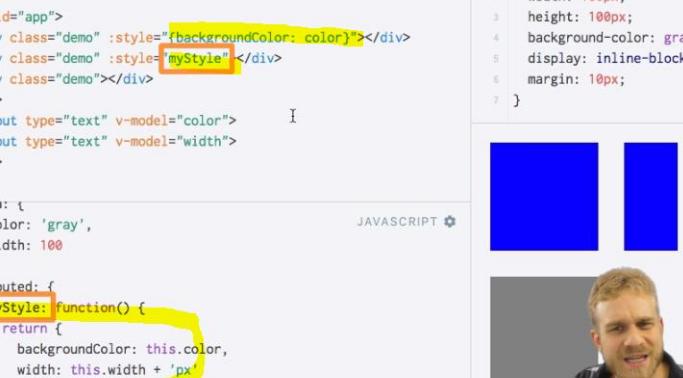
```
.red {
background-color: red;
}
.green {
background-color: green;
}
.blue {
background-color: blue;
```

JAVASCRIPT



La prima classe dell'array ha la priorità.

Nell'esempio successivo vediamo ancora come impostare lo style con vue direttamente su una riga o con una property che ritorna un oggetto style in cui vengono impostato un background ed una larghezza.

```
1 <script src="https://unpkg.com/vue/dist/vue.js"></script>  HTML ⚡  
2  
3 <div id="app">  
4   <div class="demo" :style="`backgroundColor: ${color}`"></div>  
5   <div class="demo" :style="myStyle"></div>  
6 <div class="demo"></div>  
7 <br>  
8   <input type="text" v-model="color"> I  
9   <input type="text" v-model="width">  
10 </div>  
  
4 data: {  
5   color: 'gray',  
6   width: 100  
7 },  
8 computed: {  
9   myStyle: function() {  
10     return {  
11       backgroundColor: this.color,  
12       width: this.width + 'px'  
13     }  
14   }  
15 }  
16  
1 .demo {  
2   width: 100px;  
3   height: 100px;  
4   background-color: gray;  
5   display: inline-block;  
6   margin: 10px;  
7 }  
  

```

Come per le classi è possibile passare un array di stili:

```
<div class="demo" :style="[myStyle, {height: width + 'px'}]">
```

- Getting Started: <https://jsfiddle.net/smax/pcjtcmdm/>
- Template Syntax: <https://jsfiddle.net/smax/bkk97b7g/>
- Events: <https://jsfiddle.net/smax/7zdak05g/>
- Two-Way-Binding: <https://jsfiddle.net/smax/ut0tsbcu/>
- Computed Properties & Watch: <https://jsfiddle.net/smax/yLjqxmw0/>
- Dynamic Classes: <https://jsfiddle.net/smax/gowg40ym/>
- Dynamic Styles: <https://jsfiddle.net/smax/3rvdLq5y/>

Further Links:

- Official Docs - Getting Started: <http://vuejs.org/guide/>

Capitolo 3 Using conditionals and rendering lists

Per nascondere o meno elementi sulla pagina dinamicamente esiste una diretiva v-if

```
<script src="https://unpkg.com/vue/dist/vue.js"></script>  HTML
<div id="app">
  <p v-if="show">You can see me!</p>
  <p>Do you also see me?</p>
  <button @click="show = !show">Switch</button>
</div>

new Vue({
  el: '#app',
  data: {
    show: true
  }
});
```

Esiste anche v-else:

```
<p v-if="show">You can see me!</p>
<p v-else>Now you see me!</p>
```

Per nascondere blocchi di tag non annidati uno dentro l'altro si può usare il blocco template:

```
<script src="https://unpkg.com/vue/dist/vue.js"></script>  HTML
<div id="app">
  <p v-if="show">You can see me! <span>Hello!</span></p>
  <p v-else>Now you see me!</p>
  <template v-if="show">
    <h1>Heading</h1>
    <p>Inside a template</p>
  </template>
  <p>Do you also see me?</p>
  <button @click="show = !show">Switch</button>
</div>

new Vue({
  el: '#app',
  data: {
    show: true
  }
});
```

L'alternativa potrebbe essere div ma div è visibile nella pagina mentre template non viene renderizzato ma agisce come gruppo per raggruppare l'html da nascondere condizionalmente.

Analogamente è possibile usare la direttiva v-show

```
<p v-show="show">Do you also see me?</p>
```

La differenza è che non viene tolto l'elemento dal dom ma nascosto con display:none:

```
<p style="display: none;">Do  
you also see me?</p> == $0
```

La direttiva for serve per iterare su array di proprietà:

```
<script src="https://unpkg.com/vue/dist/vue.js"></script> HTML 1

<div id="app">
  <ul>
    <li v-for="ingredient in ingredients">{{ ingredient }}</li>
  </ul>
</div>

new Vue({
  el: '#app',
  data: {
    ingredients: ['meat', 'fruit', 'cookies'],
    persons: [
      {name: 'Max', age: 27, color: 'red'},
      {name: 'Anna', age: 'unknown', color: 'blue'}
    ]
  }
});
```

Estrarre il valore e l'indice di un array con un ciclo for:

```
<script src="https://unpkg.com/vue/dist/vue.js"></script> HTML 1

<div id="app">
  <ul>
    <li v-for="(ingredient, i) in ingredients">{{ ingredient }} ({{ i }})</li>
  </ul>
</div>

new Vue({
  el: '#app',
  data: {
    ingredients: ['meat', 'fruit', 'cookies'],
    persons: [
      {name: 'Max', age: 27, color: 'red'},
      {name: 'Anna', age: 'unknown', color: 'blue'}
    ]
  }
});
```

Se si vuole riprodurre n volte html composto da più tag non annidati in un ciclo for ancora viene in aiuto il tag template:

```
<template v-for="(ingredient, index)" in ingredients>
  <h1>{{ ingredient }}</h1>
  <p>{{ index }}</p>
</template>

new Vue({
  el: '#app',
  data: {
    ingredients: ['meat', 'fruit', 'cookies'],
    persons: [
      {name: 'Max', age: 27, color: 'red'},
      {name: 'Anna', age: 'unknown', color: 'blue'}
    ]
  }
});
```

Per iterare attraverso oggetti:

```

<ul>
  <li v-for="person in persons">{{ person.name }}</li>
</ul>
<hr>

new Vue({
  el: '#app',
  data: {
    ingredients: ['meat', 'fruit', 'cookies'],
    persons: [
      {name: 'Max', age: 27, color: 'red'},
      {name: 'Anna', age: 'unknown', color: 'blue'}
    ]
  }
);

```

JAVASCRIPT

Due for annidati, il primo itera sugli oggetti ed il secondo per ciascu oggetto sulla proprietà dell'oggetto:

```

<ul>
  <li v-for="person in persons">
    <span v-for="value in person">{{ value }}</span>
  </li>
</ul>

new Vue({
  el: '#app',
  data: {
    ingredients: ['meat', 'fruit', 'cookies'],
    persons: [
      {name: 'Max', age: 27, color: 'red'},
      {name: 'Anna', age: 'unknown', color: 'blue'}
    ]
  }
);

```

JAVASCRIPT

- meat (0)
 - fruit (1)
 - cookies (2)
-
- Max27red
 - Annaunknownblue

Per mostrare oltre al valore di ciascuna proprietà anche la chiave ecco qui l'esempio:

```

<ul>
  <li v-for="person in persons">
    <span v-for="(value, key) in person">{{ key }}: {{ value }}</span>
  </li>
</ul>

new Vue({
  el: '#app',
  data: {
    ingredients: ['meat', 'fruit', 'cookies'],
    persons: [
      {name: 'Max', age: 27, color: 'red'},
      {name: 'Anna', age: 'unknown', color: 'blue'}
    ]
  }
);

```

JAVASCRIPT

- meat (0)
 - fruit (1)
 - cookies (2)
-
- name: Maxage: 27colo
 - name: Annaage: unknowncolor: blue

meat

Se oltre a chiave e valore voglio anche l'indice:

```

<ul>
  <li v-for="person in persons">
    <div v-for="(value, key, index) in person">{{ key }}: {{ value }} ({{ index }})
  </div>
  </li>
</ul>

new Vue({
  el: '#app',
  data: {
    ingredients: ['meat', 'fruit', 'cookies'],
    persons: [
      {name: 'Max', age: 27, color: 'red'},
      {name: 'Anna', age: 'unknown', color: 'blue'}
    ]
  }
);

```

JAVASCRIPT

- meat (0)
 - fruit (1)
 - cookies (2)
-
- name: Max (0)
age: 27 (1)
color: red (2)
 - name: Anna (0)
age: unknown (1)
color: blue (2)

Per iterare attraverso un numero:

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

Non ho capito questo:



```

1 <script src="https://unpkg.com/vue/dist/vue.js"></script>
2
3 <div id="app">
4   <ul>
5     <li v-for="(ingredient, i) in ingredients" :key="ingredient">{{ ingredient }} ({{ i }})</li>
6   </ul>
7   <button @click="ingredients.push('spices')">Add New</button>
8   <hr>
9   <ul>
10    <li v-for="person in persons">
11      name: {{ person.name }} ({{ person.age }})
12      age: {{ person.age }} ({{ person.color }})
13      color: {{ person.color }} ({{ person.name }})
14    </li>
15  </ul>
16</div>
17
18new Vue({
19  el: '#app',
20  data: {
21    ingredients: ['meat', 'fruit', 'cookies'],
22    persons: [
23      {name: 'Max', age: 27, color: 'red'},
24      {name: 'Anna', age: 'unknown', color: 'blue'}
25    ]
26  }
27})

```

The screenshot shows a JSFiddle interface with three tabs: HTML, CSS, and JavaScript. The JavaScript tab contains the provided Vue.js code. The rendered output shows a list of ingredients (meat, fruit, cookies) and a list of persons (Max, Anna). A button labeled "Add New" is present, which, when clicked, adds a new item ('spices') to the ingredients list.

JSFiddle:

- Conditionals (v-if and v-show): <https://jsfiddle.net/smax/hoc719j5/>
- Lists: <https://jsfiddle.net/smax/o7uy2g0u/>

Useful Links:

- Official Docs - Conditionals: <http://vuejs.org/guide/conditional.html>
- Official Docs - Lists: <http://vuejs.org/guide/list.html>

E' possibile creare sulla pagina più istanze indipendenti, per gestire per esempio un calendar oppure un multityab ecc sulla stessa pagina, la raccomandazione è usarne una ma si può anche usarne più di una.

E' possibile anche riferirsi dentro una view ad elementi in una altra view: in questo esempio, avendo salvato dentro una variabile la view instance è possibile per esempio su un evento della view2 impostare una proprietà della view1

The screenshot shows a browser developer tools console with two tabs: 'HTML' and 'JAVASCRIPT'. The 'JAVASCRIPT' tab contains the following code:

```
</div>
<div id="app2">
  <h1>{{ title }}</h1>
  <button @click="onChange">Change something in Vue 1</button>
<div>
  <h1>{{ title }}</h1>
  <button @click="onChange">Change something in Vue 1</button>
</div>
<script>
  var vm2 = new Vue({
    el: '#app2',
    data: {
      title: 'The second Instance'
    },
    methods: {
      onChange: function() {
        vm1.title = 'Changed'
      }
    }
  });
</script>
```

The 'HTML' tab shows the rendered output of the code, displaying two identical sections with the title 'The second Instance'.

Lo stesso dicasi per una funzione esterna alle view: qui un timer imposta dopo tre secondi una proprietà della view1 esternamente alla view

The screenshot shows a browser developer tools console with two tabs: 'HTML' and 'JAVASCRIPT'. The 'JAVASCRIPT' tab contains the following code:

```
</div>
<div id="app2">
  <h1>{{ title }}</h1>
  <button @click="onChange">Change something in Vue 1</button>
<div>
  <h1>{{ title }}</h1>
  <button @click="onChange">Change something in Vue 1</button>
</div>
<script>
  var vm2 = new Vue({
    el: '#app2',
    data: {
      title: 'The second Instance'
    },
    methods: {
      onChange: function() {
        setTimeout(function() {
          vm1.title = 'Changed by Timer';
        }, 3000);
      }
    }
  });
</script>
```

The 'HTML' tab shows the rendered output of the code, displaying two identical sections with the title 'The second Instance'. A yellow box highlights the `setTimeout` call in the 'JAVASCRIPT' tab.

È importante notare come nell'oggetto view stampato a console qui sotto solo le proprietà definite dentro l'oggetto verranno monitorate da watch in quanto viene definito un proxy si veda set e get nel console log dell'oggetto:

```

5   <button v-on:click="showParagraph">
6     Paragraph</button>
7   <p v-if="showParagraph">This
8     is not always visible</p>
9   </div>
10  ,
11  computed: {
12    lowercaseTitle: function() {
13      return
14        this.title.toLowerCase();
15    }
16  },
17  watch: {
18    title: function(value) {
19      alert('Title changed, new
20        value: ' + value);
21    }
22  });
23
24  vm1.newProp = 'New!';
25  console.log(vm1);

```

Mentre newprop definita fuori è una normale proprietà aggiunta dopo che non viene monitorata.

Se si guarda l'oggetto view stampato si vedono delle proprietà col dollaro davanti che si possono usare:

```

5   <button v-on:click="showParagraph">
6     Paragraph</button>
7   <p v-if="showParagraph">This
8     is not always visible</p>
9   </div>
10  ,
11  var vm1 = new Vue({
12    el: '#app1',
13    data: {
14      title: 'The VueJS Instance',
15      showParagraph: false
16    },
17    methods: {
18      show: function() {
19        this.showParagraph = true;
20        this.updateTitle('The
21          VueJS Instance (Updated)')
22      },
23      updateTitle: function(title)
24      {
25        this.title = title;
26      }
27    }

```

\$El tiene traccia dell'elemento html a cui è applicata la view

\$data è l'oggetto data della view:

```
console.log(vm1.$data.title);
```

La parte in giallo \$Data contiene quest'oggetto:

```

var vm1 = new Vue({
  el: '#app1',
  data: {
    title: 'The VueJS Instance',
    showParagraph: false
  },
  methods: {
    show: function() {
      this.showParagraph = true;
      this.updateTitle('The VueJS Instance (Updated)')
    },
    updateTitle: function(title) {
      this.title = title;
    }
  }
})

```

In più è possibile prima creare l'oggetto data e poi passarlo alla proprietà data della view:

```
var data = {  
  title: 'The VueJS Instance',  
  showParagraph: false  
}  
  
var vm1 = new Vue({  
  el: '#app1',  
  data: data,  
  methods: {  
    show: function() {  
      this.showParagraph = true;  
      this.updateTitle('The VueJS Instance (Updated)')  
    },  
    updateTitle: function(title) {  
      this.title = title  
    }  
  }  
})
```

\$ref serve per riferirsi ad un elemento html esterno, ad esempio aggiungendo la direttiva ref all'elemento html button è possibile all'interno dell'istanza view riferirsi all'elemento html per esempio per impostarne le proprietà:

```
1 <script src="https://unpkg.com/vue/dist/vue.js"></script>
2
3 <div id="app1">
4   <h1>{{ title }}</h1>
5   <button v-on:click="show" ref="myButton">Show
6   Paragraph</button>
7   <p v-if="showParagraph">This is not always visible</p>
8 </div>
9
10
11 var vm = new Vue({
12   el: '#app1',
13   data: data,
14   methods: {
15     show: function() {
16       this.showParagraph = true;
17       this.updateTitle('The VueJS Instance (Updated)');
18       this.$refs.myButton.innerText = 'Test';
19     },
20     updateTitle: function(title) {
21       this.title = title;
22     }
23   }
24 })
```

The
(Up)

Test

This is

The

Una cosa importante da capire è che con l'elemento ref possiamo modificare direttamente l'html ma un evento dell'istanza view scatenandosi ricarica il template che sovrascrive di nuovo l'elemento modificato in quanto la modifica mediante ref modifica direttamente l'html ma non il template.

È possibile creare una istanza view con un suo template così:

```
ut the
he fiddle
ces
nd Links
badmap
for features

13
14 <div id="app3">
15
16 </div>
17
18
19
20
21 el: '#app2',
22 data: {
23     title: 'The second Instance'
24 },
25 methods: {
26     onChange: function() {
27         vm1.title = 'Changed!';
28     }
29 }
30 });
31
32 var vm3 = new Vue({
33     template: '<h1>Hello!</h1>'
34 });

JAVASCRIPT
```

Questo ulteriore passaggio \$month mi permette di associare all'elemento app3 la vm3 dinamicamente:

```

52 var vm3 = new Vue({
53   template: '<h1>Hello!</h1>'
54 });
55
56 vm3.$mount('app3');

```

Facciamo un ulteriore passaggio appendendo la view con il suo template definito all'interno dell'istanza a qualsiasi elemento del DOM:

```

56 vm3.$mount();
57 document.getElementById('app3').appendChild(vm3.$el);

```

In entrambi i casi qui sotto l'elemento appare solo nella prima occorrenza:

```

8
9 <div id="app2">
10   <h1>{{ title }}</h1>
11   <button @click="onChange">Change something in Vue 1</button>
12 </div>
13
14 <div id="app3">
15
16 </div>
17 <hello></hello>
18 <hello></hello>
19   var title = 'Change me';
20
21   }
22 }
23 );
24
25 var vm3 = new Vue({
26   el: '.hello',
27   template: '<h1>Hello!</h1>'
28 );
29
30 // vm3.$mount();

```

The screenshot shows the JSFiddle interface with two code snippets and their corresponding outputs.

Code Snippet 1 (Left):

```

<div id="app2">
  <h1>{{ title }}</h1>
  <button @click="onChange">Change something in Vue 1</button>
</div>
<div id="app3">
</div>
<hello></hello>
<hello></hello>
var title = 'Change me';

  }
}
);
var vm3 = new Vue({
  el: '.hello',
  template: '<h1>Hello!</h1>'
});
// vm3.$mount();

```

Code Snippet 2 (Right):

The VueJS Instance (Updated)

This is not always visible

The second Instance

Change something in Vue 1

Hello!

Come fare se vogliamo che un elemento si presenti più volte sulla pagina?

Utilizzo dei componenti: in questo modo è possibile registrare un componente e questo potrà essere utilizzato più volte

```

9 <div id="app2">
10   <h1>{{ title }}</h1>
11   <button @click="onChange">Change something in Vue 1</button>
12   <hello></hello>
13   <hello>{/hello>
14 </div>
15
16 <div id="app3">
17
18 </div>
19 }

```

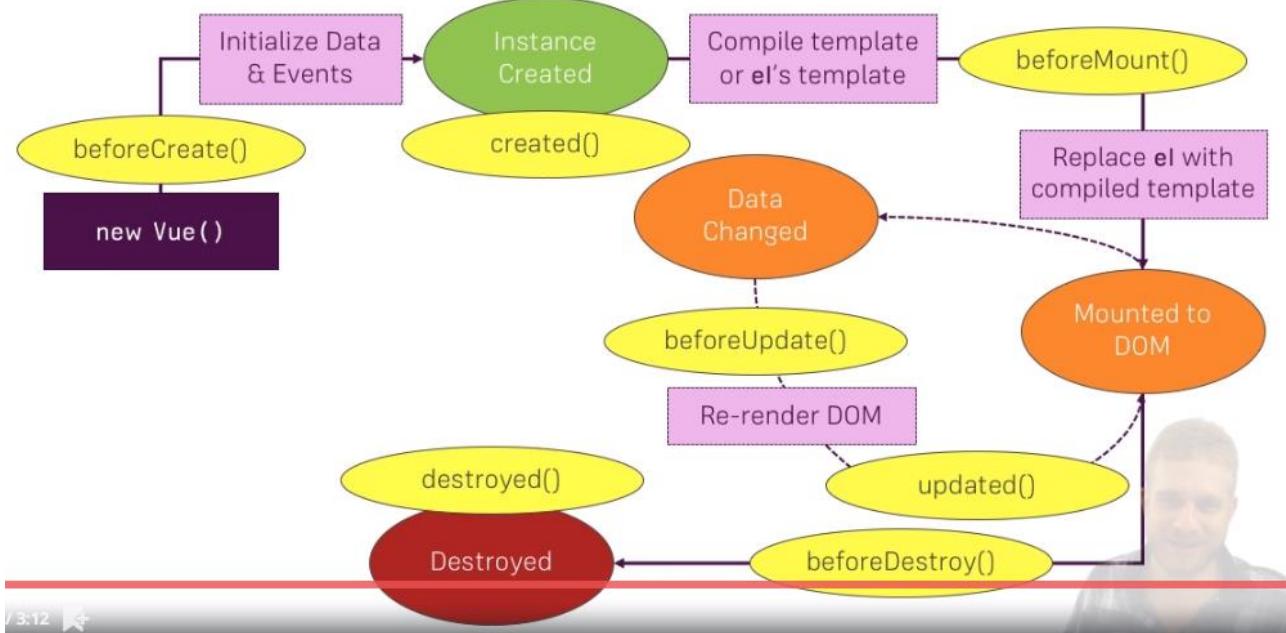
HTML

```

6 Vue.component('hello', {
7   template: '<h1>Hello!</h1>'
8 });
9
10 var vm1 = new Vue({
11   data: data,
12   methods: {
13     show: function() {

```

JAVASCRIPT



```

new Vue({
  el: '#app',
  data: {
    title: 'The VueJS Instance'
  },
  beforeCreate: function() {
    console.log('beforeCreate()');
  },
  created: function() {
    console.log('created()');
  }
})

```

```
beforeMount: function() {
  console.log('beforeMount()');
},
mounted: function() {
  console.log('mounted()');
},
beforeUpdate: function() {
  console.log('beforeUpdate()');
}
}
```

```
updated: function() {
  console.log('updated()');
},
beforeDestroy: function() {
  console.log('beforeDestroy()');
},
destroyed: function() {
  console.log('destroyed()');
}
}
```

```
methods: {
  destroy: function() {
    this.$destroy();
  }
}
```

Your Hiddle has an Unsaved draft

```
src="https://unpkg.com/vue@2.6.10/dist/vue.js"></script>

<div id="app">
  <h1>{{ title }}</h1>
  <button @click="title = 'Changed'">Update Title</button>
  <button @click="destroy">Destroy</button>
  <br/>
  <pre>{{ title }}</pre>
</div>
```

JAVASCRIPT

```
beforeUpdate: function() {
  console.log('beforeUpdate()');
},
updated: function() {
  console.log('updated()');
},
beforeDestroy: function() {
  console.log('beforeDestroy()');
},
destroyed: function() {
  console.log('destroyed()');
},
```

Changed

Update Title Destroy

Preserve log

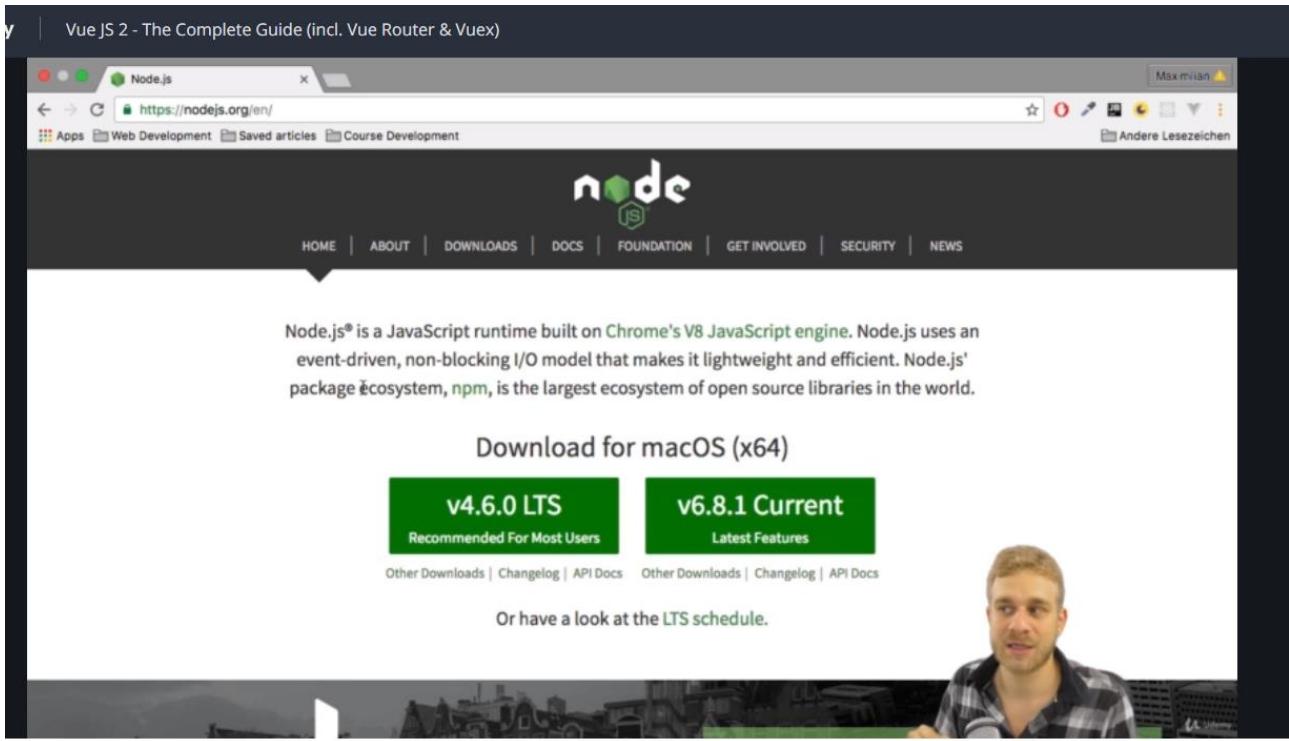
Event	File
beforeCreate()	VM13786:51
created()	(index):54
beforeMount()	(index):57
mounted()	(index):60
Download the Vue Devtools for VM13786 vue.js:5519	a better development experience: https://github.com/vuejs/vue-devtools
beforeUpdate()	(index):63
updated()	(index):66
beforeDestroy()	(index):69
destroyed()	(index):72

What is the Vue CLI?

The screenshot shows a user interface for selecting a Vue.js project template. At the top, there are two purple buttons: "VueJS Project Templates" and "npm install -g vue-cli". Below these is a dark purple header bar with the text "Choose from different Templates". Underneath are four orange boxes representing different template types:

- simple**: index.html + Vue CDN import
- webpack-simple**: Basic Webpack Workflow
- webpack**: Complex Webpack Workflow (incl. Testing)
- browserify / browserify-simple**: Browserify Workflows

Per l'uso del node packet manager installare nodejs:



All official project templates are repos in the `vuejs-templates` organization. When a new template is added to the organization, you will be able to run `vue init <template-name> <project-name>` to use that template. You can also run `vue list` to see all available official templates.

Current available templates include:

- `webpack` - A full-featured Webpack + vue-loader setup with hot reload, linting, testing & css extraction.
- `webpack-simple` - A simple Webpack + vue-loader setup for quick prototyping. **(highlighted)**
- `browserify` - A full-featured Browserify + vueify setup with hot-reload, linting & unit testing.
- `browserify-simple` - A simple Browserify + vueify setup for quick prototyping.
- `simple` - The simplest possible Vue setup in a single HTML file

Custom Templates

It's unlikely to make everyone happy with the official templates. You can simply fork an official template and then use it via `vue-cli` with:

```
vue init username/repo my-project
```

Where `username/repo` is the GitHub repo shorthand for your fork.

The shorthand `repo` notation is passed to `download-git-repo`, so you can also use things like `bitbucket:username/repo` for a Bitbucket repo and `username/repo#branch` for tags or branches.

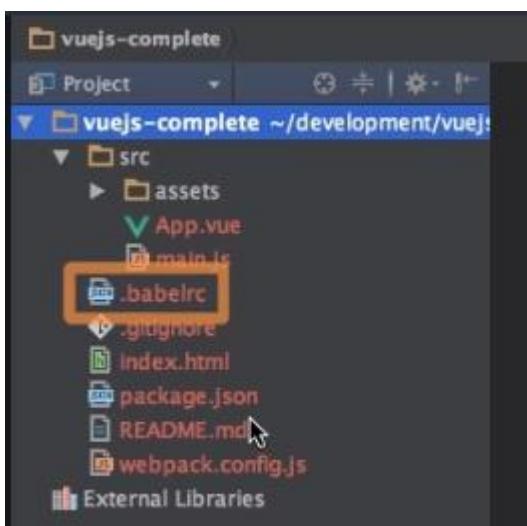
Creare progetto e installare dipendenze:

```
Maximilians-MBP:udemy maximilianschwarzmueller$ vue init webpack-simple vue-cli
This will install Vue 2.x version of template.
For Vue 1.x use: vue init webpack-simple#1.0 vue-cli
? Project name vue-cli
? Project description A Vue.js project
? Author Maximilian Schwarzmüller <mblacky0@gmail.com>
vue-cli · Generated "vue-cli".
To get started:
cd vue-cli
npm install
npm run dev

Maximilians-MBP:udemy maximilianschwarzmueller$ cd vue-cli/
Maximilians-MBP:vue-cli maximilianschwarzmueller$ npm install
(██████████) :: fetchMetadata: sill mapToRegistry uri https://registry.npmjs.org/type-is
```

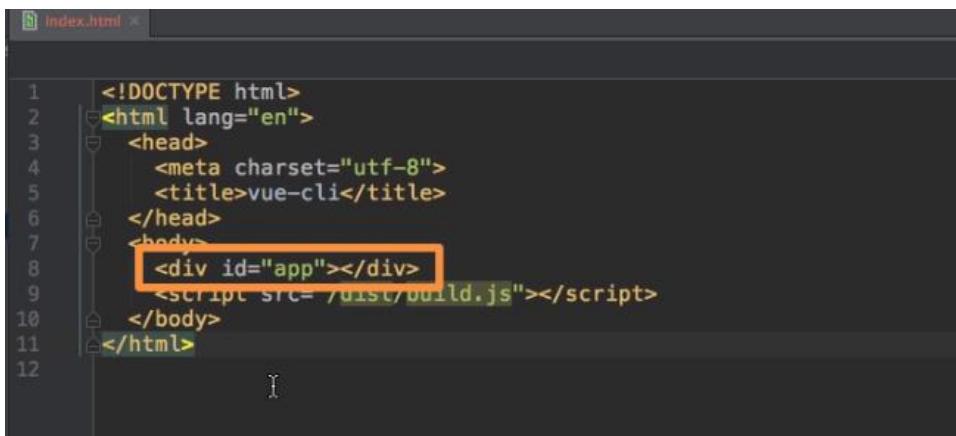
E poi lanciare l'applicazione con `run dev` questo comando installera la web application sulla porta 8080.

La cartella più importante è `src`:



Abbiamo poi il file che configura babel che ci permette di scrivere codice ES6 per trasportarlo in futuro su qualsiasi browser e non solo sui browser che supportano ES6.

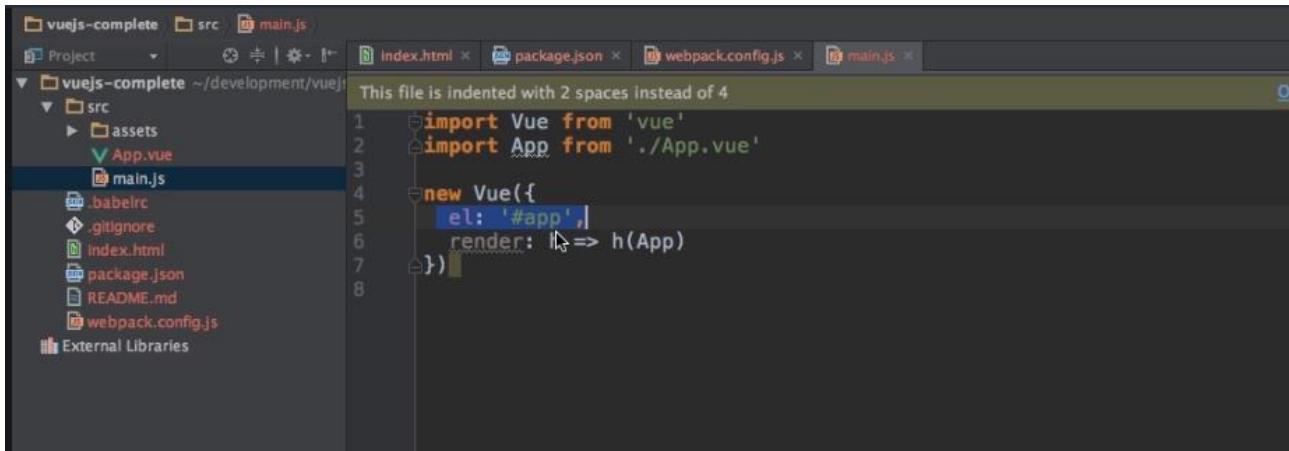
Poi abbiamo il file iniziale:



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>vue-cli</title>
  </head>
  <body>
    <div id="app"></div>
    <script src="/dist/build.js"></script>
  </body>
</html>
```

Qui stiamo usando un webpack per i nostri file (build.js) qui non stiamo scrivendo codice vuejs che viene eseguito istantaneamente.

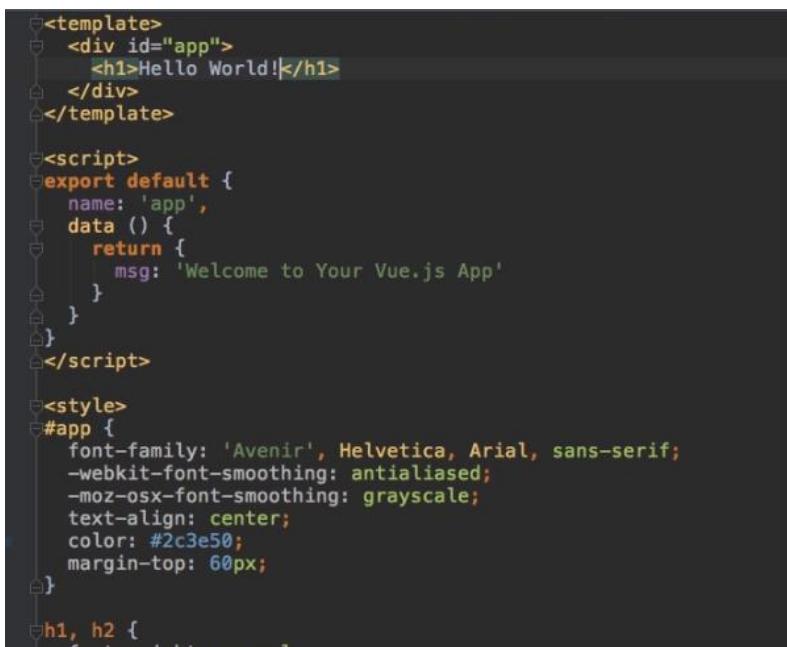
Renderizza un certo template al posto del selettore #app:



```
This file is indented with 2 spaces instead of 4
import Vue from 'vue'
import App from './App.vue'

new Vue({
  el: '#app',
  render: h => h(App)
})
```

I file vue hanno questa struttura: un template in alto, poi una sezione script (una sorta di view instance)



```
<template>
  <div id="app">
    <h1>Hello World!</h1>
  </div>
</template>

<script>
export default {
  name: 'app',
  data () {
    return {
      msg: 'Welcome to Your Vue.js App'
    }
  }
}
</script>

<style>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

Può funzionare anche così:

```
<template>
  <h1>Hello World!</h1>
</template>

<script>
  export default {
    data () {
      return {
        msg: 'Welcome to Your Vue.js App'
      }
    }
  }
</script>
```

Così abbiamo ripulito bene il modulo:

```
<template>
  <h1>Hello World!</h1>
</template>

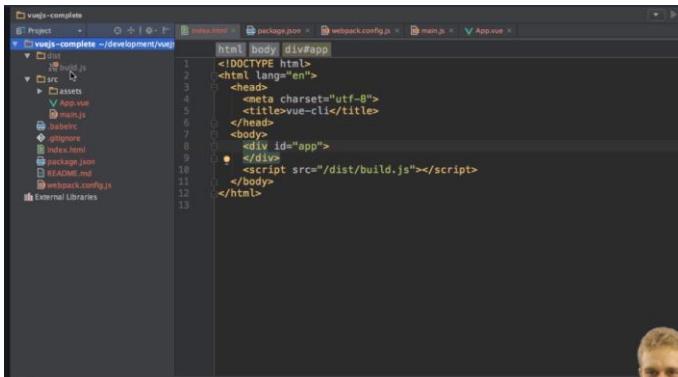
<script>
  export default {
  }
</script>

<style>
</style>
```

L'oggetto `export default` si comporta come una vista di vue.

Se compiliamo (`Npm run build`

) il codice crea il file bundles:



```
export default { }
```

Che genera una ottimizzazione

e Complete Guide (incl. Vue Router & Vuex)

I tuoi progressi

Condividi

More about ".vue" Files and the CLI

The ".vue" File

You can learn more about ".vue" Files in this Article from the official Docs: <http://vuejs.org/guide/single-file-components.html>

Learn more about the `render()` method in another Article in the official Docs: <http://vuejs.org/guide/render-function.html>

Finally, it's important to be aware of the fact, that you can also load your App.vue File (your main Component/ Instance) via the following two Ways (Alternatives to `render()`):

1) Using the ES6 Spread Operator (for that, you need to add babel-preset-stage-2 as a Dependency and to your .babelrc File):

```
npm install --save-dev babel-preset-stage-2
```

.babelrc:

```
1. {
2.   "presets": [
3.     ["es2015", { "modules": false }],
4.     ["stage-2"]
5.   ]
6. }
```

```
1. import Vue from 'vue'
2. import App from './App.vue'
3.
4. new Vue({
5.   el: '#app',
6.   ...App
7. });
```

2) Using `mount()`:

Also install the stage-2 preset as described above.

```
1. import Vue from 'vue'
2. import App from './App.vue'
3.
4. const vm = new Vue({
5.   ...App
6. });
7.
8. vm.$mount('#app');
```

The CLI

Learn more about the CLI here: <https://github.com/vuejs/vue-cli>

Local CSS / Sass Files and CLI Templates

The **webpack-simple** template **doesn't support local CSS or Sass files**, because no CSS loader is set up.

Use the **webpack** template (not webpack-simple) to get this functionality: <https://github.com/vuejs-templates/webpack>

Introduzione ai componenti:

Prendiamo il codice seguente:



The screenshot shows a browser developer tools console with two tabs: 'HTML' and 'JAVASCRIPT'. The 'HTML' tab contains the following code:

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

<div id="app">
</div>
```

The 'JAVASCRIPT' tab contains the following code:

```
new Vue({
  el: '#app',
  data: {
    status: 'Critical'
  },
  template: '<p>Server Status: {{ status }}</p>'
})
```

Below the tabs, the text 'Server Status: Critical' is displayed.

È una normale istanza vue quindi non può essere usata come abbiamo visto prima più di una volta.

Un componente estende una istanza vue fondamentalmente.

Se creiamo un componente fondamentalmente oltre al primo argomento come secondo passiamo un oggetto che racchiude la data property dell'istanza vue, ma così non va bene in quanto data non sono i data del componente ma della view da cui deriva:



The screenshot shows a browser developer tools console with two tabs: 'HTML' and 'JAVASCRIPT'. The 'HTML' tab contains the following code:

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

<div id="app">
  <my-cmp></my-cmp>
</div>
```

The 'JAVASCRIPT' tab contains the following code:

```
Vue.component('my-cmp', {
  data: {
    status: 'Critical'
  },
  template: '<p>Server Status: {{ status }}</p>'
});

new Vue({
  el: '#app'
})
```

Below the tabs, the text 'Server Status: Critical' is displayed.

I dati del componente devono essere racchiusi in una funzione che ritorna un oggetto che rappresenta i dati del componente., in quanto siamo sicuri che i dati del componente non interferiscano con i dati nostri:

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

<div id="app">
  <my-cmp></my-cmp>
</div>

Vue.component('my-cmp', {
  data: function() {
    return {
      status: 'Critical'
    }
  },
  template: '<p>Server Status: {{ status }}</p>'
});

new Vue({
  el: '#app'
})
```

Se creiamo l'oggetto data esterno in realtà stiamo condividendo l'oggetto tra due componenti:

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

<div id="app">
  <my-cmp></my-cmp>
  <hr>
  <my-cmp></my-cmp>
</div>

var data = { status: 'Critical' };

Vue.component('my-cmp', {
  data: function() {
    return data;
  },
  template: '<p>Server Status: {{ status }}</p>'
});

new Vue({
  el: '#app'
})
```

Server Status:

Server Status:

Questo è un componente registrato globalmente:

```
Vue.component('my-cmp', {
  data: function() {
    return {
      status: 'Critical'
    }
  },
  template: '<p>Server Status: {{ status }}</p>'
});

new Vue({
  el: '#app'
})
```

Per dichiarare un componente localmente occorre metterlo in una variabile e togliere l'etichetta:

```
JAVASCRIPT
var cmp = {
  data: function() {
    return {
      status: 'Critical'
    }
  },
  template: '<p>Server Status: {{ status }} <button @click="changeStatus">Change</button></p>',
  methods: {
    changeStatus: function() {
      this.status = 'Normal';
    }
  }
}
```

In questo modo possiamo registrarlo localmente dentro la vue appesa all'elemento #app:

```
new Vue({
  el: '#app',
  components: {
    'my-cmp': cmp
  }
})
```

In questo modo solo i tag my-cmp dentro il div app ma non dentro app2 renderizzerà i componenti:

```
<div id="app">
  <my-cmp></my-cmp>
  <hr>
  <my-cmp></my-cmp>
</div>

<div id="app2">
  <my-cmp></my-cmp>

```

Esempio di componente (si è creato un file Home.js):

```
vuejs-complete / src / Home.vue
Project   src   Home.vue
  vuex-complete ~/development/vuejs
    dist
      build.js
    src
      assets
        App.vue
        Home.vue
        main.js
        babelrc
        .gitignore
        index.html
        README.md
        webpack.config.js
    External Libraries
```

```
App.vue x  Home.vue x  main.js x
<template>
  <p>Server Status: {{ status }}</p>
  <hr>
  <button @click="changeStatus">Change Status</button>
</template>
<script>
  export default {
    data: function() {
      return {
        status: 'Critical'
      }
    },
    methods: {
      changeStatus() {
        this.status = 'Normal';
      }
    }
  }
</script>
```

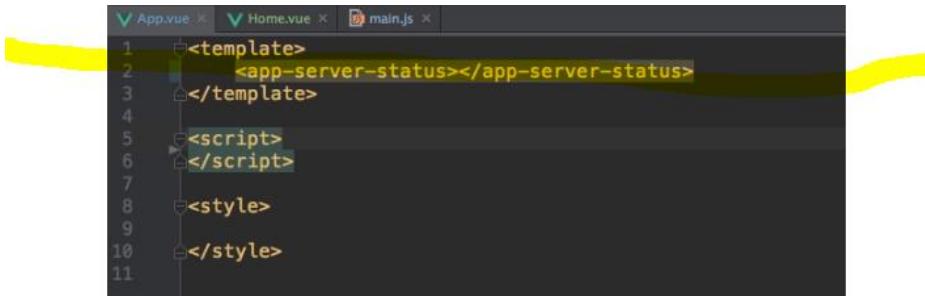
Come usare il componente?Così:

registrandolo globalmente:

```
import Vue from 'vue'
import App from './App.vue'
import Home from './Home.vue';

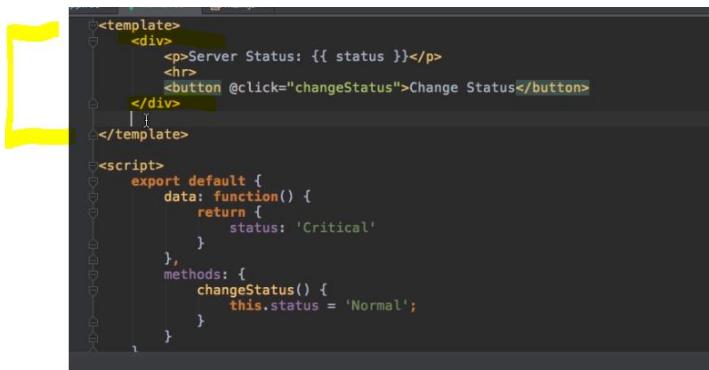
Vue.component('app-server-status', Home);

new Vue({
  el: '#app',
  render: h => h(App)
})
```



```
App.vue X Home.vue X main.js X
1 <template>
2   <app-server-status></app-server-status>
3 </template>
4
5 <script>
6 </script>
7
8 <style>
9 </style>
10
11
```

Attenzione perché darà un errore in quanto il componente deve avere dentro l'elemento template un solo elemento div e non elementi fratelli, quindi correggiamo:



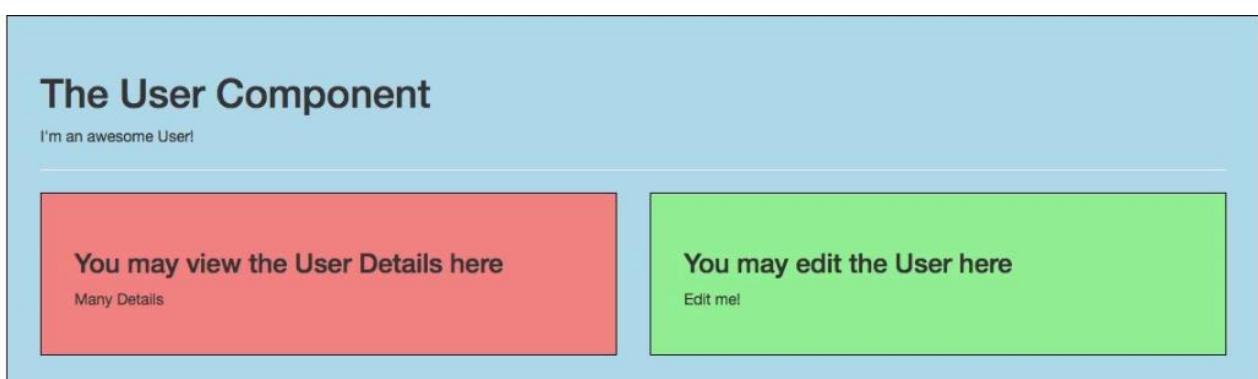
```
<template>
  <div>
    <p>Server Status: {{ status }}</p>
    <hr>
    <button @click="changeStatus">Change Status</button>
  </div>
</template>

<script>
  export default {
    data: function() {
      return {
        status: 'Critical'
      }
    },
    methods: {
      changeStatus() {
        this.status = 'Normal';
      }
    }
  }
</script>
```

Per registrare un componente locale si fa così, ad esempio qui registriamo un componente serverstatusvue dentro un componente home che ha la funzione di iterare per generare 5 istanze del componente referenziato localmente mediante una direttiva v-for.

Per trasferire una proprietà dal controllo parente al figlio si usa props:

comunicazione tra componenti: qui abbiamo un root component e due child component (detail and add component):



The User Component

I'm an awesome User!

You may view the User Details here

Many Details

You may edit the User here

Edit me!

```

1 <template>
2   <div class="container">
3     <div class="row">
4       <div class="col-xs-12">
5         <app-user></app-user>
6       </div>
7     </div>
8   </template>
9
10 <script>
11   import User from './components/User.vue';
12
13   export default {
14     components: {
15       appUser: User
16     }
17   }
18 </script>
19
20 <style>
21   div.component {
22     border: 1px solid black;
23     padding: 30px;
24   }
25 </style>
26
27

```

```

1 <template>
2   <div class="component">
3     <h1>The User Component</h1>
4     <p>I'm an awesome User!</p>
5     <hr>
6     <div class="row">
7       <div class="col-xs-12 col-sm-6">
8         <app-user-detail></app-user-detail>
9       </div>
10      <div class="col-xs-12 col-sm-6">
11        <app-user-edit></app-user-edit>
12      </div>
13    </div>
14  </div>
15 </template>
16
17 <script>
18   import UserDetail from './UserDetail.vue';
19   import UserEdit from './UserEdit.vue';
20
21   export default {
22     components: {
23       appUserDetail: UserDetail,
24       appUserEdit: UserEdit
25     }
26   }
27 </script>
28

```

Supponiamo di voler cambiare un nome dal componente di root e passarlo in output al componente child detail, quindi creiamo un bottone che imposta con un click una proprietà name ad anna

```

1 <template>
2   <p>I'm an awesome User!</p>
3   <button @click="changeName">Change my Name</button>
4   <hr>
5   <div class="row">
6     <div class="col-xs-12 col-sm-6">
7       <app-user-detail></app-user-detail>
8     </div>
9     <div class="col-xs-12 col-sm-6">
10       <app-user-edit></app-user-edit>
11     </div>
12   </div>
13 </div>
14 </template>
15
16 <script>
17   import UserDetail from './UserDetail.vue';
18   import UserEdit from './UserEdit.vue';
19
20   export default {
21     data: function() {
22       return {
23         name: 'Max'
24       };
25     },
26     methods: {
27       changeName() {
28         this.name = 'Anna'
29       }
30     }
31   }
32 </script>

```

Ora dobbiamo passare la proprietà name al componente figlio detail, come fare? Mediante props che si aspetta proprietà arrivare dall'esterno, props prende un array di stringhe:

```

<template>
  <div class="component">
    <h3>You may view the User Details here</h3>
    <p>Many Details</p>
    <p>User Name: {{ name }}</p>
  </div>
</template>

<script>
  export default {
    props: ['name']
  }
</script>

<style scoped>
  div {
    background-color: lightcoral;
  }
</style>

```

E come passare dall'esterno al componente figlio? Con v-bind:name="name" dove il primo name è il nome definito in props ed il secondo la proprietà dinamica del controllo parente.

La proprietà passata esternamente può lavorare come una normalissima proprietà; qui vediamo come può essere manipolata e invertita:

```

<template>
  <div class="component">
    <h3>You may view the User Details here</h3>
    <p>Many Details</p>
    <p>User Name: {{ switchName() }}</p>
  </div>
</template>

<script>
  export default {
    props: ['myName'],
    methods: {
      switchName() {
        return this.myName.split("").reverse().join("");
      }
    }
  }
</script>

<style scoped>
  div {
    background-color: lightcoral;
  }
</style>

```

Per validare una proprietà nel componente (ad esempio vogliamo impedire che venga passato un intero che genererebbe un errore se splitto dal metodo switchname) possiamo definire props non come array ma come oggetto che prende una proprietà col nome da validare ed un arry di tipi ammessi:

```

<template>
  <div class="component">
    <h3>You may view the User Details here</h3>
    <p>Many Details</p>
    <p>User Name: {{ switchName() }}</p>
  </div>
</template>

<script>
  export default {
    props: {
      myName: [String, Array]
    },
    methods: {
      switchName() {
        return this.myName.split("").reverse().join("");
      }
    }
  }
</script>

<style scoped>
  div {
    background-color: lightcoral;
  }
</style>

```

In questo modo avremo un errore esplicativo e capiremo meglio piuttosto che un errore di tipo split error...:

Ed ancora possiamo ulteriormente per ciascuna proprietà definire un ulteriore oggetto per specificare il tipo, il default se required ecc...:

```
export default {
  props: {
    myName: {
      type: String,
      required: true
    }
  }
}
```

Ora vogliamo aggiornare il componente esterno con la modifica dall'interno della proprietà

Name del componente detail (creiamo un bottone reset che imposta a max) :

```
<h3>You may view the User Details here</h3>
<p>Many Details</p>
<p>User Name: {{ switchName() }}</p>
<button @click="resetName">Reset Name</button>

</div>
</template>

<script>
  export default {
    props: {
      myName: {
        type: String
      }
    },
    methods: {
      switchName() {
        return this.myName.split("").reverse().join("");
      },
      resetName() {
        this.myName = 'Max';
      }
    }
  }
</script>

<style scoped>
  div {
    background-color: lightcoral;
  }
</style>
```

Ora occorre esporre all'esterno la modifica con un custom event usiamo un builtin method \$emit è una sorta di throw event, all'emit passiamo il nome dell'evento e dei dati

```

<template>
  <h3>You may view the User Details here</h3>
  <p>Many Details</p>
  <p>User Name: {{ switchName() }}</p>
  <button @click="resetName">Reset Name</button>
</template>

<script>
  export default {
    props: {
      myName: {
        type: String
      }
    },
    methods: {
      switchName() {
        return this.myName.split("").reverse().join("");
      },
      resetName() {
        this.myName = 'Max';
        this.$emit('nameWasReset', this.myName);
      }
    }
  }
</script>

<style scoped>
  div {
    ...
  }
</style>

```

E come facciamo ad ascoltare un evento dall'esterno? Con v-on o abbreviato @ passando il nome dell'evento = e qui va una function o del codice che setta la proprietà name ai dati passati (\$event).

Quindi riassumendo: la comunicazione diretta tra due child component non può avvenire in quanto in vue la comunicazione è unidirezionale dal parent al child e viceversa

Component Communication



Component Communication



Unidirectional Data Flow from Top to Bottom!



Un altro modo per comunicare tra parente e child consiste nel creare un metodo resetname dopodichè si crea una property resetfn che passa il metodo reset name.

```

<script>
  import UserDetail from './UserDetail.vue';
  import UserEdit from './UserEdit.vue';

  export default {
    data: function () {
      return {
        name: 'Max'
      };
    },
    methods: {
      changeName() {
        this.name = 'Anna';
      },
      resetName() {
        this.name = 'Max';
      }
    },
    components: {
      appUserDetail: UserDetail,
      appUserEdit: UserEdit
    }
  }
</script>

```

The screenshot shows the code editor with the User.vue file open. The 'resetName()' method is highlighted with a green box. In the template, the 'resetFn="resetName"' prop is highlighted with a red box.

```

<template>
  <div class="component">
    <h1>The User Component</h1>
    <p>I'm an awesome User!</p>
    <button @click="changeName">Change my Name</button>
    <p>Name is {{ name }}</p>
    <hr>
    <div class="row">
      <div class="col-xs-12 col-sm-6">
        <app-user-detail
          :myName="name"
          :nameWasReset="name = $event"
          :resetFn="resetName"></app-user-detail>
      </div>
      <div class="col-xs-12 col-sm-6">
        <app-user-edit></app-user-edit>
      </div>
    </div>
  </div>
</template>

<script>
  import UserDetail from './UserDetail.vue';
  import UserEdit from './UserEdit.vue';

  export default {
    data: function () {
      return {

```

Quindi nel componente child:

The screenshot shows the UserDetail.vue component code. The 'resetFn' prop is highlighted with an orange box in the template, and the 'resetFn' parameter in the 'methods' block is also highlighted with an orange box.

```

<template>
  <div class="component">
    <h3>You may view the User Details here</h3>
    <p>Many Details</p>
    <p>User Name: {{ switchName() }}</p>
    <button @click="resetName">Reset Name</button>
    <button @click="resetFn()">Reset Name</button>
  </div>
</template>

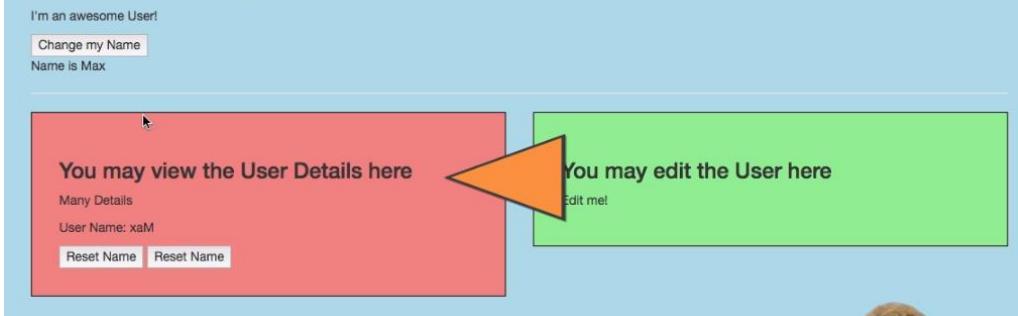
<script>
  export default {
    props: {
      myName: {
        type: String
      },
      resetFn: Function
    },
    methods: {
      switchName() {
        return this.myName.split("").reverse().join("");
      },
      resetName() {
        this.myName = 'Max';
        this.$emit('nameWasReset', this.myName);
      }
    }
  }
</script>

```

Questo appena visto è un altro modo di lavorare senza custom event nel passaggio delle informazioni, una alternativa.

Ora otterremo i dati da un controllo ad un altro ma come?:

The User Component



Abbiamo due o tre modi:

dentro controllo edit aggiungiamo un bottone che setta una data

```
<template>
  <div class="component">
    <h3>You may edit the User here</h3>
    <p>Edit me!</p>
    <button @click="editAge">Edit Age</button>
  </div>
</template>
<script>
  export default {
    methods: {
      editAge() {
        this.age = 30;
      }
    }
  }
</script>
<style scoped>
  div {
    background-color: lightgreen;
  }
</style>
```

Nello user component (parent) creiamo una proprietà age e la passiamo in entrata al edit component

```
<div class="row">
  <div class="col-xs-12 col-sm-6">
    <app-user-detail
      :myName="name"
      @nameWasReset="name = $event"
      :resetFn="resetName"></app-user-detail>
  </div>
  <div class="col-xs-12 col-sm-6">
    <app-user-edit :userAge="age"></app-user-edit>
  </div>
</div>
</template>
<script>
  import UserDetail from './UserDetail.vue';
  import UserEdit from './UserEdit.vue';

  export default {
    data: function () {
      return {
        name: 'Max',
        age: 27
      };
    },
    methods: {
      changeName() {
        this.name = 'Anna';
      }
    }
  }
</script>
```

Quindi andiamo ad aggiungere una props dentro edit control:

```

1 <template>
2   <div class="component">
3     <h3>You may edit the User here</h3>
4     <p>Edit me!</p>
5     <p>User Age: {{ userAge }}</p>
6     <button @click="editAge"=Edit Age/>
7   </div>
8 </template>
9
10 <script>
11   export default {
12     props: ['userAge'],
13     methods: {
14       editAge() {
15         this.userAge = 30;
16       }
17     }
18   }
19 </script>
20
21 <style scoped>
22   div {
23     background-color: lightgreen;
24   }
25 </style>
26

```

9c83] added scoped styles

Quindi creiamo una props usage anche nel detail per passare l'età:

```

<template>
  <div class="row">
    <div class="col-xs-12 col-sm-6">
      <app-user-detail
        :name="name"
        @nameWasReset="name = $event"
        :resetFn="resetName"
        :userAge="age"></app-user-detail>
    </div>
    <div class="col-xs-12 col-sm-6">
      <app-user-edit :userAge="age"></app-user-edit>
    </div>
  </div>
</template>
<script>
  import UserDetail from './UserDetail.vue';
  import UserEdit from './UserEdit.vue';

  export default {
    data: function () {
      return {
        name: 'Max',
        age: 27
      };
    },
    methods: {
      changeName() {

```

9tts-simple af10c83] added scoped styles

Ora se clicchiamo su edit age aggiorna il componente edit ma non il componente detail

The User Component

I'm an awesome User!

Change my Name

Name is Max

You may view the User Details here

Many Details

User Name: xAM

User Age: 27

Reset Name Reset Name

You may edit the User here

Edit me!

User Age: 30

✓ Edit Age

Questo perché occorre avvisare il contenitore del cambiamento e lo possiamo fare con emit come abbiamo visto quindi nel componente edit:



```

1 <template>
2   <div class="component">
3     <h3>You may edit the User here</h3>
4     <p>Edit me!</p>
5     <p>User Age: {{ userAge }}</p>
6     <button @click="editAge">Edit Age</button>
7   </div>
8 </template>
9
10 <script>
11   export default {
12     props: ['userAge'],
13     methods: {
14       editAge() {
15         this.userAge = 30;
16         this.$emit('ageWasEdited', this.userAge);
17       }
18     }
19   </script>
20
21 <style scoped>
22   div {
23     background-color: lightgreen;
24   }
25 </style>
26
27

```

E poi ascoltiamo che l'evento avvenga dal controllo contenitore ed impostiamo la proprietà age la quale viene poi passata come props al componente detail, ora funziona:



Per passare a più child annidati delle proprietà usiamo il bus:

dichiariamo un bus che è una view a livello globale come costante nel file main:

```

App.vue x  User.vue x  UserDetail.vue x  UserEdit.vue x  main.js x  index.html x
import Vue from 'vue'
import App from './App.vue'

export const eventBus = new Vue();
new Vue({
  el: '#app',
  render: h => h(App)
})

```

Importiamo il bus nel componente edit e lo usiamo per scatenare l'evento con emit del bus di modifica età:

```
4     <p>Edit me!</p>
5     <p>User Age: {{ userAge }}</p>
6     <button @click="editAge">Edit Age</button>
7   </div>
8 </template>
9
10 <script>
11   import { eventBus } from '../main';
12
13   export default {
14     props: ['userAge'],
15     methods: {
16       editAge() {
17         this.userAge = 30;
18         this.$emit('ageWasEdited', this.userAge);
19         eventBus.$emit('ageWasEdited', this.userAge);
20       }
21     }
22   </script>
23
24 <style scoped>
25   div {
26     background-color: lightgreen;
27   }
28 </style>
29
30
```

Quindi andiamo ad ascoltare con un listener l'evento definito prima nel controllo edit questo listener viene lanciato nell'evento created e quando si scatena aggiorniamo l'età in questo componente.

Per centralizzare la gestione degli eventi è possibile nel bus fare così:

```
1 import Vue from 'vue'
2 import App from './App.vue'
3
4 export const eventBus = new Vue({
5   methods: {
6     changeAge(age) {
7       this.$emit('ageWasEdited', age);
8     }
9   }
10 });
11
12 new Vue({
13   el: '#app',
14   render: h => h(App)
15 })
16
```

E quindi dove viene scatenato l'evento (nell'edit component) sostituire il codice precedente con questo:

```
4     <p>Edit me!</p>
5     <p>User Age: {{ userAge }}</p>
6     <button @click="editAge">Edit Age</button>
7   </div>
8 </template>
9
10 <script>
11   import { eventBus } from '../main';
12
13   export default {
14     props: ['userAge'],
15     methods: {
16       editAge() {
17         this.userAge = 30;
18         this.$emit('ageWasEdited', this.userAge);
19         eventBus.$emit('ageWasEdited', this.userAge);
20         eventBus.changeAge(this.userAge);
21       }
22     }
23   </script>
24
25 <style scoped>
26   div {
27     background-color: lightgreen;
28   }
29 </style>
```

Module Resources & Useful Links

The full source code can be found attached.

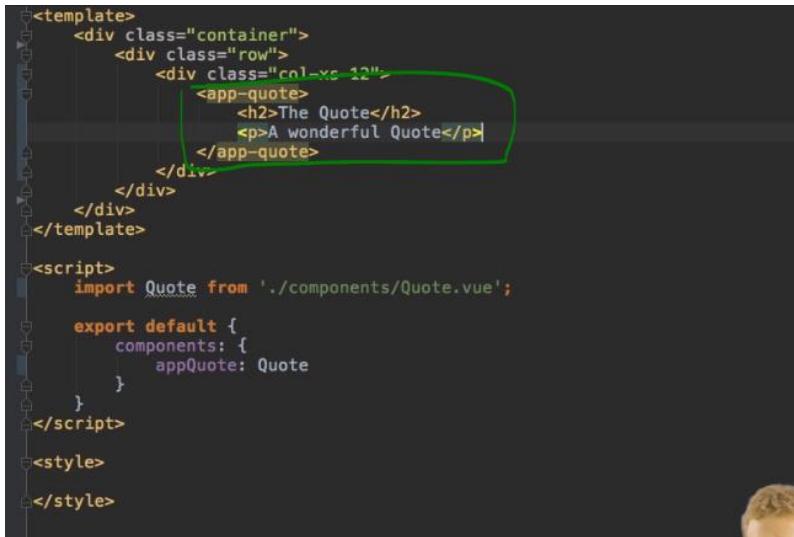
Useful Links:

- Official Docs - Props: <http://vuejs.org/guide/components.html#Props>
- Official Docs - Custom Events: <http://vuejs.org/guide/components.html#Custom-Events>
- Official Docs - Non-Parent-Child Communication: <http://vuejs.org/guide/components.html#Non-Parent-Child-Communication>

Uso avanzato dei componenti

Come passare un intero blocco di html dall'esterno ad un componente? Mediante uno slot:

passiamo il blocco di html dentro i tag del componente (app-quote)



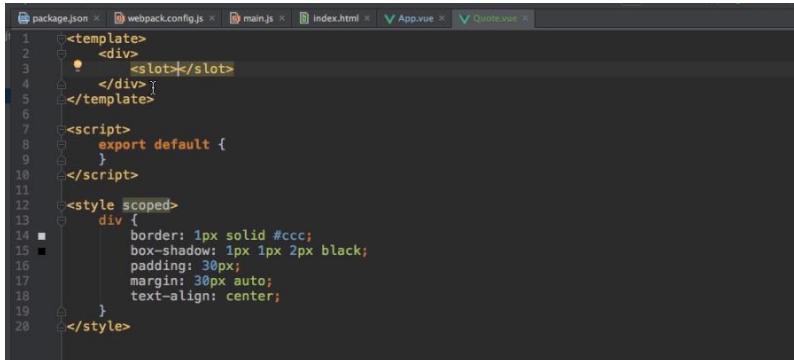
```
<template>
  <div class="container">
    <div class="row">
      <div class="col-xs-12">
        <app-quote>
          <h2>The Quote</h2>
          <p>A wonderful Quote</p>
        </app-quote>
      </div>
    </div>
  </div>
</template>

<script>
  import Quote from './components/Quote.vue';

  export default {
    components: {
      appQuote: Quote
    }
  }
</script>

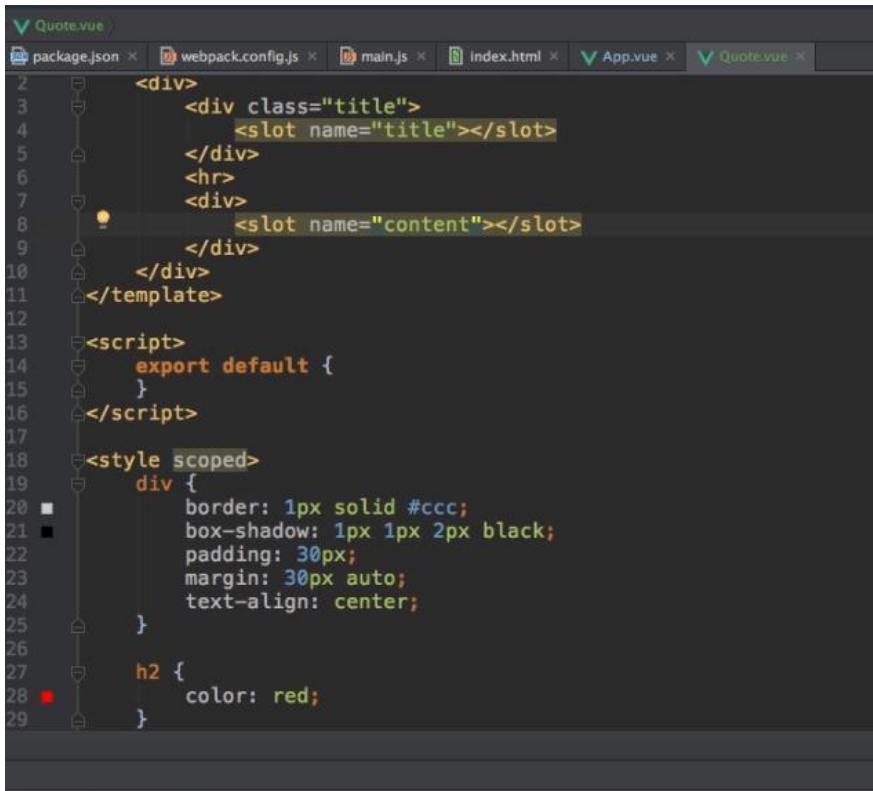
<style>
</style>
```

E nel componente child aggiungiamo il tag slot:



```
1  <template>
2   <div>
3     <slot></slot>
4   </div>
5 </template>
6
7 <script>
8   export default {
9   }
10 </script>
11
12 <style scoped>
13   div {
14     border: 1px solid #ccc;
15     box-shadow: 1px 1px 2px black;
16     padding: 30px;
17     margin: 30px auto;
18     text-align: center;
19   }
20 </style>
```

E se volessimo avere più slot con contenuti diversi? Diamo dei nomi diversi dentro ai componenti.

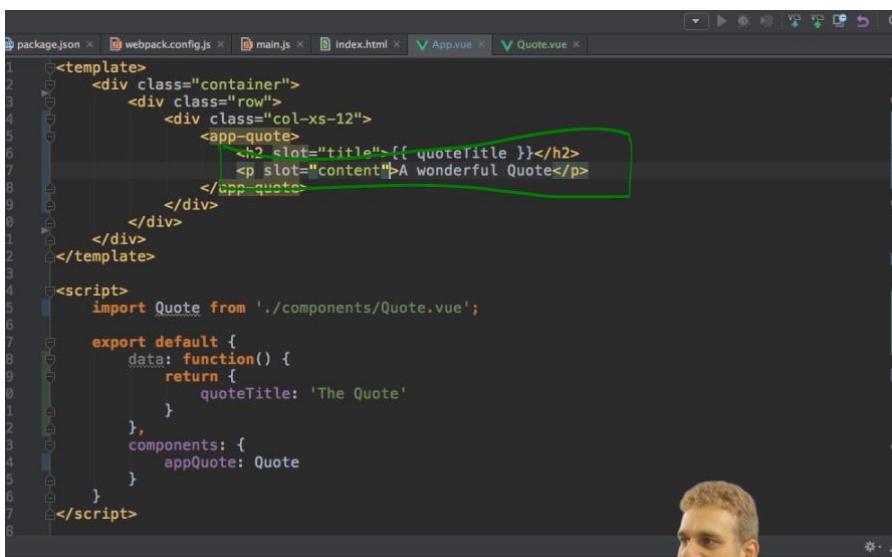


```
<div>
  <div class="title">
    <slot name="title"></slot>
  </div>
  <hr>
  <div>
    <slot name="content"></slot>
  </div>
</div>
</template>

<script>
  export default {
  }
</script>

<style scoped>
  div {
    border: 1px solid #ccc;
    box-shadow: 1px 1px 2px black;
    padding: 30px;
    margin: 30px auto;
    text-align: center;
  }

  h2 {
    color: red;
  }
</style>
```

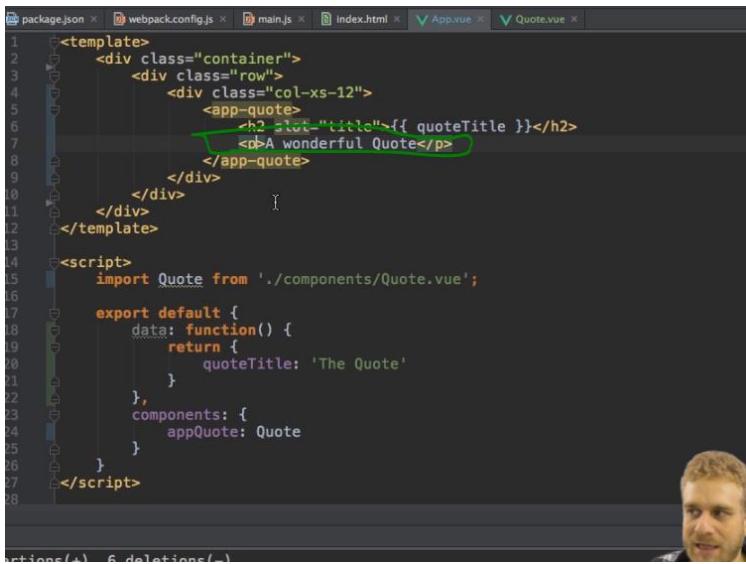


```
<template>
  <div class="container">
    <div class="row">
      <div class="col-xs-12">
        <app-quote>
          <h2 slot="title">{{ quoteTitle }}</h2>
          <p slot="content">A wonderful Quote</p>
        </app-quote>
      </div>
    </div>
  </div>
</template>

<script>
  import Quote from './components/Quote.vue';

  export default {
    data: function() {
      return {
        quoteTitle: 'The Quote'
      }
    },
    components: {
      appQuote: Quote
    }
  }
</script>
```

E se abbiamo uno slot senza nome? Ve lo tratta questo come default slot:



```
<template>
  <div class="container">
    <div class="row">
      <div class="col-xs-12">
        <app-quote>
          <h2 slot="title">{{ quoteTitle }}</h2>
          <p>A wonderful Quote</p>
        </app-quote>
      </div>
    </div>
  </div>
</template>

<script>
  import Quote from './components/Quote.vue';

  export default {
    data: function() {
      return {
        quoteTitle: 'The Quote'
      }
    },
    components: {
      appQuote: Quote
    }
  }
</script>
```

```

<div>
  <div class="title">
    <slot name="title"></slot>
  </div>
  <hr>
  <div>
    <slot></slot>
  </div>
</div>
</template>

<script>
  export default {
  }
</script>

<style scoped>
  div {
    border: 1px solid #ccc;
    box-shadow: 1px 1px 2px black;
    padding: 30px;
    margin: 30px auto;
    text-align: center;
  }

  h2 {
    color: red;
  }
</style>

```

E se passiamo contenuti allo slot tra apertura e chiusura slot i contenuti vengono passati e rappresentati:

```

<div>
  <div class="title">
    <slot name="title"></slot>
    <span style="color: #ccc"><slot name="subtitle">The Subtitle</slot></span>
  </div>
  <hr>
  <div>
    <slot></slot>
  </div>
</div>
</template>

<script>
  export default {
  }
</script>

<style scoped>
  div {
    border: 1px solid #ccc;
    box-shadow: 1px 1px 2px black;
    padding: 30px;
    margin: 30px auto;
    text-align: center;
  }

  h2 {
    color: red;
  }
</style>

```

E se avessimo nel controllo parente tre componenti da caricare dinamicamente? Occorre usare i dynamic components, ad esempio mettiamo tre button al click sui quali viene impostata una proprietà stringa che contiene il nome del componente da caricare dinamicamente, quindi mediante il tag component e la proprietà dinamica :is diciamo che controllo caricare:

```

<div class="row">
  <div class="col-xs-12">
    <button @click="selectedComponent = 'appQuote'">Quote</button>
    <button @click="selectedComponent = 'appAuthor'">Author</button>
    <button @click="selectedComponent = 'appNew'">New</button>
    <hr>
    <p>{{ selectedComponent }}</p>
    <component :is="selectedComponent">
      <p>Default Content</p>
    </component>
    <!-- app-quote -->
      <!--<h2 slot="title">{{ quoteTitle }}</h2>-->
      <!--<p>A wonderful Quote</p>-->
    <!--</app-quote>-->
  </div>
</div>
</template>

<script>
  import Quote from './components/Quote.vue';
  import Author from './components/Author.vue';
  import New from './components/New.vue';

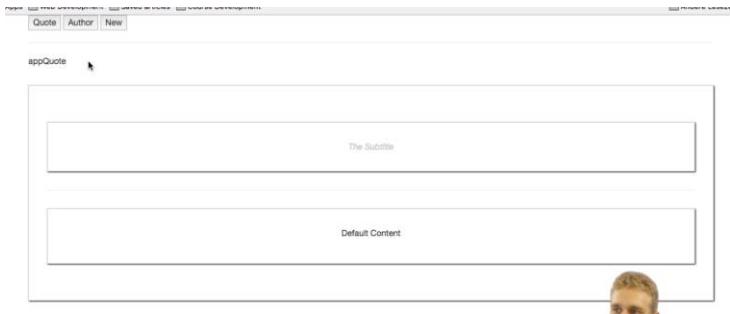
  export default {
    data: function() {
      return {
        quoteTitle: 'The Quote',
      }
    }
  }
</script>

```

```

        selectedComponent: 'app-quote'
    }
},
components: {
    appQuote: Quote,
    appAuthor: Author,
    appNew: New
}
}
</script>

```



Se creiamo il componente new con la pressione sul tasto new poi creiamo default con la pressione su default e poi torniamo su new il componente viene distrutto e ricreato, possiamo sovrascrivere questo comportamento di default includendolo in keep-alive:

```

<keep-alive>
    <component :is="selectedComponent">
        <p>Default Content</p>
    </component>
</keep-alive>

```

In questo modo l'evento destroyed non verrà chiamato, al suo posto verranno utilizzati gli eventi activated e deactivated.

Module Resources & Helpful Links

You can find the Module Source Code attached to this Lecture.

Helpful Links:

- Official Docs - Slots: <http://vuejs.org/guide/components.html#Content-Distribution-with-Slots>
- Official Docs - Dynamic Components: <http://vuejs.org/guide/components.html#Dynamic-Components>
- Official Docs - Misc: <http://vuejs.org/guide/components.html#Misc>

FORMS

```
<label for="email">Mail</label>
<input
    type="text"
    id="email"
    class="form-control"
    v-model="userData.email">
</div>
<div class="form-group">
    <label for="password">Password</label>
    <input
        type="password"
        id="password"
        class="form-control"
        v-model="userData.password">
</div>
<div class="form-group">
    <label for="age">Age</label>
    <input
        type="number"
        id="age"
        class="form-control"
        v-model="userData.age">
</div>
</div>
</div>
```

```
<script>
  export default {
    data () {
      return {
        userData: {
          email: '',
          password: '',
          age: 27
        }
      }
    }
  />/script>
```

```
</form>
<hr>
<div class="row">
  <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
    <div class="panel panel-default">
      <div class="panel-heading">
        <h4>Your Data</h4>
      </div>
      <div class="panel-body">
        <p>Mail: {{ userData.email }}</p>
        <p>Password: {{ userData.password }}</p>
        <p>Age: {{ userData.age }}</p>
        <p>Message:</p>
        <p><strong>Send Mail?</strong></p>
        <ul>
          <li></li>
        </ul>
        <p>Gender: </p>
        <p>Priority: </p>
      </div>
    </div>
  </div>
</div>
```

File a Complaint

Mail

Password

Age

Message

Submit

Your Data

Mail: test@test.com
Password: test@test.com
Age: 27

Message:

Send Mail?

Gender:

35 persone contrassegnate al momento.

Se vogliamo modificare il modo di aggiornamento del binding (non su keystroke) basta usare il modificatore lazy:

```
class="form-control"
v-model.lazy="userData.password"
:if(userData.password != null)"
```

(esiste poi .stream o .number per forzare solo numeri o non inserire spazi a sinistra e destra)

Per quanto riguarda le checkbox se aggiungiamo una proprietà v-model che viene impostata a sendEmail da due checkbox, vue è in grado di capire e di aggiungerla automaticamente ad un array.

```
</div>
<div class="row">
  <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
    <div class="form-group">
      <label for="sendmail">
        <input type="checkbox"
          id="sendmail"
          value="SendMail"
          v-model="sendMail"> Send Mail
      </label>
      <label for="sendinfomail">
        <input type="checkbox"
          id="sendInfomail"
          value="SendInfoMail"
          v-model="sendMail"> Send Infomail
      </label><!-- Also show for single checkbox with true/ false -->
    </div>
  </div>
<div class="row">
  <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
    <label for="male">
      <input
```

```
script>
  export default {
    data () {
      return {
        userData: {
          email: '',
          password: '',
          age: 27
        },
        message: 'A new Text',
        sendMail: []
      }
    }
  }
</script>
<style>
</style>
```

```

<h4>Your Data</h4>
</div>
<div class="panel-body">
  <p>Mail: {{ userData.email }}</p>
  <p>Password: {{ userData.password }}</p>
  <p>Age: {{ userData.age }}</p>
  <p style="white-space: pre">Message: {{ message }}</p>
  <p><strong>Send Mail?</strong></p>
  <ul>
    <li v-for="item in sendMail">{{ item }}</li>
  </ul>
  <p>Gender: </p>
  <p>Priority: </p>
</div>
</div>

```

Per i radio button

```

export default {
  data () {
    return {
      userData: {
        email: '',
        password: '',
        age: 27
      },
      message: 'A new Text',
      sendMail: [],
      gender: 'Male'
    }
  }
}

```

```

<label for="male">
  <input
    type="radio"
    id="male"
    value="Male"
    v-model="gender"> Male
</label>
<label for="female">
  <input
    type="radio"
    id="female"
    value="Female"
    v-model="gender"> Female
</label>

```

In questo modo vue sa che i bottoni appartengono allo stesso gruppo è il dato selezionato viene memorizzato nella proprietà gender.

Combo:

```

<script>
export default {
  data () {
    return {
      userData: {
        email: '',
        password: '',
        age: 27
      },
      message: 'A new Text',
      sendMail: [],
      gender: 'Male',
      priorities: ['High', 'Medium', 'Low']
    }
  }
}

```

```

</div>
<div class="row">
  <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3 from-group">
    <label for="priority">Priority</label>
    <select
      id="priority"
      class="form-control">
      <option v-for="priority in priorities">{{ priority }}</option>
    </select>
  </div>
</div>
<hr>

```

Per il default:

```
        class="form-control">
      <option v-for="priority in priorities" :selected="priority == 'Medium'">{{ priority }}
```

Per default:

```
<script>
export default {
  data () {
    return {
      userData: {
        email: '',
        password: '',
        age: 27
      },
      message: 'A new Text',
      sendMail: [],
      gender: 'Male',
      selectedPriority: 'High',
      priorities: ['High', 'Medium', 'Low']
    }
  }
</script>

<style>
</style>
```

```
</div>
</div>
<div class="row">
  <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3 form-group">
    <label for="priority">Priority</label>
    <select
      id="priority"
      class="form-control"
      v-model="selectedPriority">
      <option
        v-for="priority in priorities"
        :selected="priority == 'Medium'">{{ priority }}</option>
    </select>
  </div>
</div>
<hr>
<div class="row">
  <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
    <div class="panel-body">
      <p>Mail: {{ userData.email }} </p>
      <p>Password: {{ userData.password }}</p>
      <p>Age: {{ userData.age }}</p>
      <p style="white-space: pre">Message: {{ message }}</p>
      <p><strong>Send Mail?</strong></p>
      <ul>
        <li v-for="item in sendMail">{{ item }}</li>
      </ul>
      <p>Gender: {{ gender }}</p>
      <p>Priority: {{ selectedPriority }}</p>
    </div>
  </div>
</div>
```

Costruire uno slot

Lo stesso comportamento di v-model possiamo ottenerlo con questa sintassi:

```
<div class="form-group">
  <label for="email">Mail</label>
  <input
    type="text"
    id="email"
    class="form-control"
    :value="userData.email"
    @input="userData.email = $event.target.value">
</div>
<div class="form-group">
  <template>
    <div>
      <div
        id="on"
        @click="isOn = true"
        :class="{active: isOn}"/>On</div>
      <div
        id="off"
        @click="isOn = false"
        :class="{active: !isOn}"/>Off</div>
    </template>
  </div>
</div>
```

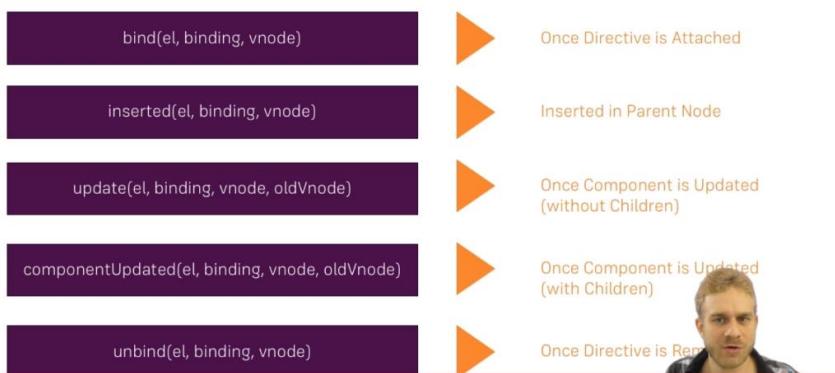
Gestire submit del form:

```
<div class="row">
  <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6">
    <button
      class="btn btn-primary"
      @click.prevent="submitted">Submit!
    </button>
  </div>
</div>
form>
```

```
},
methods: {
  submitted() {
    this.isSubmitted = true;
  }
},
```

Custom Directive:

Hooks



il primo modo è dichiararle globalmente

```
App.vue x main.js x index.html x
1 import Vue from 'vue'
2 import App from './App.vue'
3
4 Vue.directive('highlight', {
5   bind(el, binding, vnode) {
6     el.style.backgroundColor = 'green';
7   }
8 });
9
10 new Vue({
11   el: '#app',
12   render: h => h(App)
13 })
```

```
App.vue x main.js x index.html x
1 <template>
2   <div class="container">
3     <div class="row">
4       <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
5         <h1>Built-in Directives</h1>
6         <p v-text='Some Text'></p>
7         <p v-html='<strong>Some strong text</strong>'></p>
8       </div>
9     </div>
10    <hr>
11    <div class="row">
12      <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
13        <h1>Custom Directives</h1>
14        <p v-highlight>Color this</p>
15      </div>
16    </div>
17  </div>
18 </template>
19
20 <script>
21   export default {}
22 </script>
23
24 <style>
25 </style>
26
```

Built-in Directives

Some Text

Some strong text

Custom Directives

Se vogliamo passare il colore alla direttiva ossia un valore?

```
<div class="row">
  <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
    <h1>Custom Directives</h1>
    <p v-highlight="red">Color this</p>
  </div>
</div>
</div>
```

```
Vue.directive('highlight', {
  bind(el, binding, vnode) {
    // el.style.backgroundColor = 'green';
    el.style.backgroundColor = binding.value;
  }
});

new Vue({
  el: '#app',
  render: h => h(App)
})
```

Dove binding è la direttiva e value il valore passato tra virgolette ("red")

E se vogliamo passare degli argomenti?

```
import Vue from 'vue'
import App from './App.vue'

Vue.directive('highlight', {
  bind(el, binding, vnode) {
    // el.style.backgroundColor = 'green';
    // el.style.backgroundColor = binding.value,
    if (binding.arg == 'background') {
      el.style.backgroundColor = binding.value;
    } else {
      el.style.color = binding.value;
    }
  }
});

new Vue({
  el: '#app',
  render: h => h(App)
})
```

```
<div class="row">
  <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
    <h1>Custom Directives</h1>
    <p v-highlight:background="red">Color this</p>
  </div>
</div>
</div>
```

Custom Directives

Se rimuoviamo background

```

<div class="row">
  <div class="col-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
    <h1>Custom Directives</h1>
    <p v-highlight="red">Color this</p>
  </div>
</div>

```

Custom Directives

Color this

E per aggiungere un modifier custom?

Ad esempio se aggiungiamo un modifier delayed vogliamo che sia ritardata l'applicazione dello stile:

```

Vue.directive('highlight', {
  bind(el, binding, vnode) {
    // el.style.backgroundColor = 'green';
    // el.style.backgroundColor = binding.value;
    var delay = 0;
    if (binding.modifiers['delayed']) {
      delay = 3000;
    }
    setTimeout(() => {
      if (binding.arg == 'background') {
        el.style.backgroundColor = binding.value;
      } else {
        el.style.color = binding.value;
      }
    }, delay);
  }
});

new Vue({
  el: '#app',
  render: h => h(App)
})

```

```

<div class="row">
  <div class="col-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
    <h1>Custom Directives</h1>
    <p v-highlight:background.delayed="red">Color this</p>
  </div>
</div>

```

Prima abbiamo creato una direttiva globale, ora la creiamo locale:



The screenshot shows a code editor with a file named App.vue open. The code defines a global directive 'highlight' and a local directive 'local-highlight'. The local directive's implementation is identical to the global one, using a setTimeout function to delay the style application. The video player interface is visible at the bottom, showing a man speaking.

```

<script>
  export default {
    directives: {
      'local-highlight': {
        bind(el, binding, vnode) {
          var delay = 0;
          if (binding.modifiers['delayed']) {
            delay = 3000;
          }
          setTimeout(() => {
            if (binding.arg == 'background') {
              el.style.backgroundColor = binding.value;
            } else {
              el.style.color = binding.value;
            }
          }, delay);
        }
      }
    }
  }

```

I modifier multipli:

```
import { ... } from 'vue'

export default {
  directives: {
    'local-highlight': {
      bind(el, binding, vnode) {
        var delay = 0;
        if (binding.modifiers['delayed']) {
          delay = 3000;
        }
        if (binding.modifiers['blink']) {
          let mainColor = binding.value;
          let secondColor = 'blue';
          let currentColor = mainColor;
          setTimeout(() => {
            setInterval(() => {
              currentColor == secondColor ? currentColor = mainColor : currentColor = secondColor;
              if (binding.arg == 'background') {
                el.style.backgroundColor = currentColor;
              } else {
                el.style.color = currentColor;
              }
            }, 1000);
          }, delay);
        } else {
          setTimeout(() => {
            if (binding.arg == 'background') {
```

```
<p v-local-highlight:background.delayed.blink="red">Color this, too</p>
```

Per passare più di un valore alla direttiva usiamo un oggetto:

```
import { ... } from 'vue'

export default {
  directives: {
    'local-highlight': {
      bind(el, binding, vnode) {
        let mainColor = binding.value.mainColor;
        let secondColor = binding.value.secondColor;
        let currentColor = mainColor;
        setTimeout(() => {
          setInterval(() => {
            currentColor == secondColor ? currentColor = mainColor : currentColor = secondColor;
            if (binding.arg == 'background') {
              el.style.backgroundColor = currentColor;
            } else {
              el.style.color = currentColor;
            }
          }, binding.value.delay);
        }, delay);
      } else {
        setTimeout(() => {
          if (binding.arg == 'background') {
            el.style.backgroundColor = binding.value.mainColor;
          } else {
            el.style.color = binding.value.mainColor;
          }
        }, delay);
      }
    }
  }
}
```

BP:vuejs-complete maximilianschwarzmueller\$ git checkout -b directives
new branch 'directives'

Costruire un filtro:

un filtro deve avere un valore in entrata (il valore da trasformare), per applicare il filtro occorre usare il pipe.

E poi:

```
<template>
  <div class="container">
    <div class="row">
      <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
        <h1>Filters & Mixins</h1>
        <p>{{ text | touppercase }}</p>
      </div>
    </div>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        text: 'Hello there!'
      },
      filters: {
        toUppercase(value) {
          return value.toUpperCase();
        }
      }
    }
  }
</script>

<style>
</style>
```

La funzione del filtro si può scrivere o così:

```
filters: {
    'to-uppercase'() {
        |
    }
}
```

O così (camelcase):

```
'filters: {
    toUppercase() {
        |
    }
}'
```

Ora creiamo un filtro globale:

```
App.vue x main.js x index.html x
1 import Vue from 'vue'
2 import App from './App.vue'
3
4 Vue.filter('to-lowercase', function(value) {
5     return value.toLowerCase();
6 });
7
8 new Vue({
9     el: '#app',
10    render: h => h(App)
11 })
12
```

E poi applichiamo due filtri a text:

```
<template>
<div class="container">
    <div class="row">
        <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
            <h1>Filters & Mixins</h1>
            <p>{{ text | toUppercase | toLowerCase }}</p>
        </div>
    </div>
</div>
<script>
export default {
    data() {
        return {
            text: 'Hello there!'
        },
        filters: {
            toUppercase(value) {
                return value.toUpperCase();
            }
        }
    }
}
<style>
</style>
```

Per complesse trasformazioni occorre usare computed properties (e non filtri in quanto vue rilancia l'esecuzione dei filtri ad ogni refresh del dom) che vengono eseguite solo cambia l'elemento manipolato, quindi al posto del filtro per recuperare da una lista di frutti l'elemento/i che meccia il testo inserito nella input text (che se fatto con i filtri ogni volta che rilancio il dom dovrebbe filtrare tutto l'array) scriviamo il seguente codice utilizzando le computed properties.

```

<template>
  <div class="container">
    <div class="row">
      <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
        <h1>Filters & Mixins</h1>
        <p>{{ text | toUppercase | to-lowercase }}</p>
        <br>
        <input v-model="filterText">
        <ul>
          <li v-for="fruit in fruits">{{ fruit }}</li>
        </ul>
      </div>
    </div>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        text: 'Hello there!',
        fruits: ['Apple', 'Banana', 'Mango', 'Melon'],
        filterText: ''
      }
    },
    filters: {
      toUppercase(value) {
        return value.toUpperCase();
      }
    },
    computed: {
      filteredFruits() {
        return this.fruits.filter((element) => {
          return element.match(this.filterText);
        });
      }
    }
  }
</script>

```

Se vogliamo evitare la duplicazione di codice possiamo usare mixin: supponiamo di avere lo stesso codice dentro due componenti:

se per esempio vogliamo lo stesso codice dentro list.vue e app.vue occorrerà tagliare questo codice:

```

<template>
  <div>
    <h1>Filters & Mixins</h1>
    <input v-model="filterText">
    <ul>
      <li v-for="fruit in filteredFruits">{{ fruit }}</li>
    </ul>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        fruits: ['Apple', 'Banana', 'Mango', 'Melon'],
        filterText: ''
      }
    },
    computed: {
      filteredFruits() {
        return this.fruits.filter((element) => {
          return element.match(this.filterText);
        });
      }
    }
  }
</script>

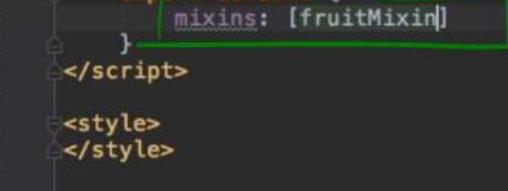
```

Creiamo un file js ed esportiamo un oggetto come costante:



```
1  export const fruitMixin = {
2    data() {
3      return {
4        fruits: ['Apple', 'Banana', 'Mango', 'Melon'],
5        filterText: ''
6      }
7    },
8    computed: {
9      filteredFruits() {
10        return this.fruits.filter(element => {
11          return element.match(this.filterText);
12        });
13      }
14    }
15 }
```

Poi importiamo il file fruitmixin in list.vue:



```
1 <template>
2   <div>
3     <h1>Filters & Mixins</h1>
4     <input v-model="filterText">
5     <ul>
6       <li v-for="fruit in filteredFruits">{{ fruit }}</li>
7     </ul>
8   </div>
9 </template>
10 <script>
11   import { fruitMixin } from './fruitMixin';
12
13   export default {
14     mixins: [fruitMixin]
15   }
16 </script>
17
18 <style>
19 </style>
20
```

Assegnando alla proprietà mixins che è un array la costante oggetto importata.
E lo stesso facciamo in app.vue:

The screenshot shows the `App.vue` file in a code editor. The `<script>` block contains the following code:

```
15         </div>
16     </div>
17   </template>
18
19   <script>
20     import List from './List.vue';
21     import { fruitMixin } from './fruitMixin';
22
23     export default {
24       mixins: [fruitMixin],
25       data() {
26         return {
27           text: 'Hello there!'
28         }
29       },
30       filters: {
31         toUppercase(value) {
32           return value.toUpperCase();
33         }
34       },
35       components: {
36         appList: List
37       }
38     }
39   </script>
40
41   <style>
42   </style>
```

A green oval highlights the `mixins: [fruitMixin],` part of the `export default` object. A status bar at the bottom indicates "113 persone contrassegnate al momento."

Il mixin viene eseguito prima del component in cui viene iniettato, infatti l'evento created del mixin si scatena prima di quello del component ospitante.

La proprietà mixin è un array perché è possibile avere più mixin importati per componente. Un mixin può essere definito anche a livello globale ed ha effetto su tutte le vue del progetto:

The screenshot shows the `main.js` file in a code editor. It defines a global mixin and then creates a new Vue instance:

```
1  import Vue from 'vue'
2  import App from './App.vue'
3
4  Vue.filter('to-lowercase', function(value) {
5    return value.toLowerCase();
6  });
7
8  Vue.mixin({
9    created() {
10      console.log('Global Mixin - Created Hook');
11    }
12  });
13
14  new Vue({
15    el: '#app',
16    render: h => h(App)
17  })
18
```

A green oval highlights the `Vue.mixin({ ... })` block. A large green bracket highlights the entire block from line 8 to line 12, indicating it applies to all components.

Ritornando al file mixin:

```

1 export const fruitMixin = {
2   data() {
3     return {
4       fruits: ['Apple', 'Banana', 'Mango', 'Melon'],
5       filterText: ''
6     }
7   },
8   computed: {
9     filteredFruits() {
10       return this.fruits.filter(element => {
11         return element.match(this.filterText);
12       });
13     }
14   }
15 }

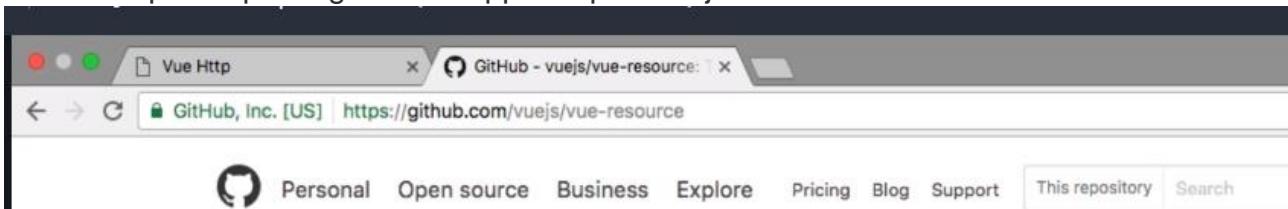
```

questo oggetto non è condiviso dalle istanze ma ogni istanza ha una copia, quindi se aggiungiamo un frutto da una istanza si aggiornerà solo l'istanza corrente e non le altre.
Animazioni:

http:

per recuperare risorse dal server:

useremo questo package scritto apposta per vue js:



Dopo avere installato il pacchetto con npm install

```

1 import Vue from 'vue'
2 import VueResource from 'vue-resource';
3 import App from './App.vue'
4
5 Vue.use(VueResource);
6
7 new Vue({
8   el: '#app',
9   render: h => h(App)

```

```

28   return {
29     user: {
30       username: '',
31       email: ''
32     },
33     users: []
34   };
35 },
36 methods: {
37   submit() {
38     this.$http.post('https://vuejs-http.firebaseio.com/data.json', this.user)
39     .then(response => {
40       console.log(response);
41     }, error => {
42       console.log(error);
43     });
44   },
45   fetchData() {
46     this.$http.get('https://vuejs-http.firebaseio.com/data.json')
47     .then(response => {
48       const data = response.json();
49       console.log(data);
50     });
51   }
52 }
53 </script>

```

Così come sopra nel fetch data otteniamo una promise perché raccogliamo una promise con una chiamata asincrona, per avere i dati occorre aggiungere questo codice:

```
    },
    fetchData() {
      this.$http.get('https://vuejs-http.firebaseio.com/data.json')
        .then(response => {
          return response.json();
        })
        .then(data => console.log(data));
    }
}

```

C'è un posto dove è possibile dichiarare globalmente gli url che vengono chiamati:

```
App.vue x index.html x main.js x
1 import Vue from 'vue'
2 import VueResource from 'vue-resource';
3 import App from './App.vue'
4
5 Vue.use(VueResource);
6
7 Vue.http.options.root = 'https://vuejs-http.firebaseio.com/data.json';
8
9 new Vue({
10   el: '#app',
11   render: h => h(App)
12 })
13
14 },
15 methods: {
16   submit() {
17     this.$http.post('', this.user)
18       .then(response => {
19         console.log(response);
20       }, error => {
21         console.log(error);
22       });
23   },
24   fetchData() {
25     this.$http.get('')
26       .then(response => {
27         return response.json();
28       })
29       .then(data => {
30         const resultArray = [];
31       });
32   }
33 }
```

E' possibile intercettare ogni richiesta aggiungendo delle callback ad un array interceptors a livello globale

Per ottenere una risposta passare al metodo next una callback

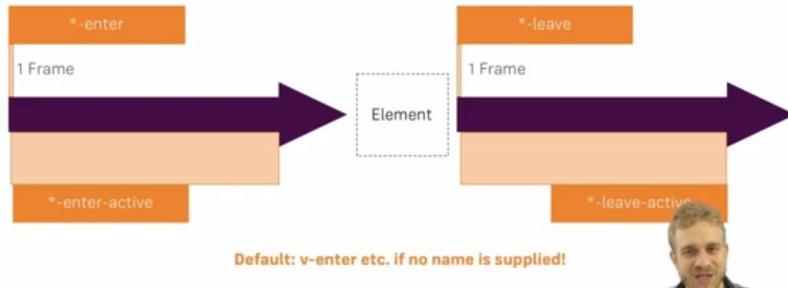
```
    // Per ottenere una risposta passare al metodo next una callback
    Vue.use(VueResource);

    Vue.http.options.root = 'https://vuejs-http.firebaseio.com/data.json';
    Vue.http.interceptors.push((request, next) => {
        console.log(request);
        if (request.method == 'POST') {
            request.method = 'PUT';
        }
        next(response => {
            response.json = () => { return {messages: response.body} }
        });
    });

    new Vue({
        el: '#app',
        render: h => h(App)
    });
}
```

Animation

Transition CSS Classes



Esempio di transizione

```
<template>
  <br>
  <button class="btn btn-primary" @click="show = !show">Show Alert</button>
  <br><br>
  <transition name="fade">
    <div class="alert alert-info" v-if="show">This is some Info</div>
  </transition>
</div>
</div>
</template>
```

```
<style>
  .fade-enter {
    opacity: 0;
  }

  .fade-enter-active {
    transition: opacity 1s;
  }

  .fade-leave {
    /*opacity: 1;*/
  }

  .fade-leave-active {
    transition: opacity 1s;
    opacity: 0;
  }
</style>
```

Esempio di animazione

```
<template>
  <div class="container">
    <div class="row">
      <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
        <h1>Animations</h1>
        <br>
        <button class="btn btn-primary" @click="show = !show">Show Alert</button>
        <br><br>
        <transition name="fade">
          <div class="alert alert-info" v-if="show">This is some Info</div>
        </transition>
        <transition name="slide">
          <div class="alert alert-info" v-if="show">This is some Info</div>
        </transition>
      </div>
    </div>
  </div>
</template>
```

```
@keyframes slide-in {
  from {
    transform: translateY(20px);
  }
  to {
    transform: translateY(0);
  }
}

@keyframes slide-out {
  from {
    transform: translateY(0);
  }
  to {
    transform: translateY(20px);
  }
}

.slide-enter-active {
  animation: slide-in 1s ease-out forwards;
}

.slide-leave-active {
  animation: slide-out 1s ease-out forwards;
}
```

Per mixare transizione ed animazione basta aggiungere ciò che abbiamo fatto prima per la transizione sugli eventi dello slide:

```
.slide-enter {  
    opacity: 0;  
    /*transform: translateY(20px);*/  
}  
  
.slide-enter-active {  
    animation: slide-in is ease-out forwards;  
    transition: opacity .5s;  
}  
  
.slide-leave {  
    ...  
}  
  
.slide-leave-active {  
    animation: slide-out is ease-out forwards;  
    transition: opacity .1s;  
    opacity: 0;  
}
```

C'è un problema però ossia che vue non sa se sincronizzare in base alla transizione o alla animazione quindi il risultato è un po' strano, occorre quindi esplicitare a che tipo riferirsi principalmente per armonizzare il tutto (quindi bisogna definire chi detta la lunghezza):

```
<transition name="slide" type="animation">  
    <div class="alert alert-info" v-if="show">This is some Info</div>  
</transition>  
/div>
```

E per animare le transizioni quando la pagina viene caricata?

Mediante l'attributo appear:

```
<transition name="fade" appear>  
    <div class="alert alert-info" v-if="show">This is some Info</div>  
</transition>  
/div>
```

Questo è un ottimo sito dove trovare animazioni già fatte coi css:

A screenshot of a browser window showing the Animate.css website. The address bar shows 'https://daneden.github.io/animate.css/'. The page title is 'Animate.css' and the subtitle is 'Just-add-water CSS animations'. The page content displays various CSS animation examples.

Se volessimo applicare le animazioni sopra mediante le classi css di animate.css dovremo sostituire la proprietà name poiché verrebbe chiamata neutralizzando il resto invece la togliamo ed aggiungiamo questi attributi:

```
<template>  
    <div class="container">  
        <div class="row">  
            <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">  
                <h1>Animations</h1>  
                <br>  
                <br><br>  
                <transition name="fade">  
                    <div class="alert alert-info" v-show="show">This is some Info</div>  
                </transition>  
                <transition name="slide" type="animation">  
                    <div class="alert alert-info" v-if="show">This is some Info</div>  
                </transition>  
                <transition>  
                    appear  
                    enter-class=""  
                    enter-active-class="animated bounce"  
                    leave-class=""  
                    leave-active-class="animated shake"  
                >  
                    <div class="alert alert-info" v-if="show">This is some Info</div>  
                </transition>  
            </div>  
        </div>  
    </template>
```

Routing

Installare il pacchetto vue router

```
Maximilians-MBP:vuejs-complete maximilianschwarzmueller$ npm install --save vue-router
1:50 / 2:16
```

Importare il pacchetto router

```
App.vue × main.js × User.vue × UserStart.vue × index.html × UserDetail.vue ×
1 import Vue from 'vue'
2 import VueRouter from 'vue-router';
3 import App from './App.vue'
4
5 Vue.use(VueRouter);
6
7 new Vue({
8   el: '#app',
9   render: h => h(App)
10})
11
```

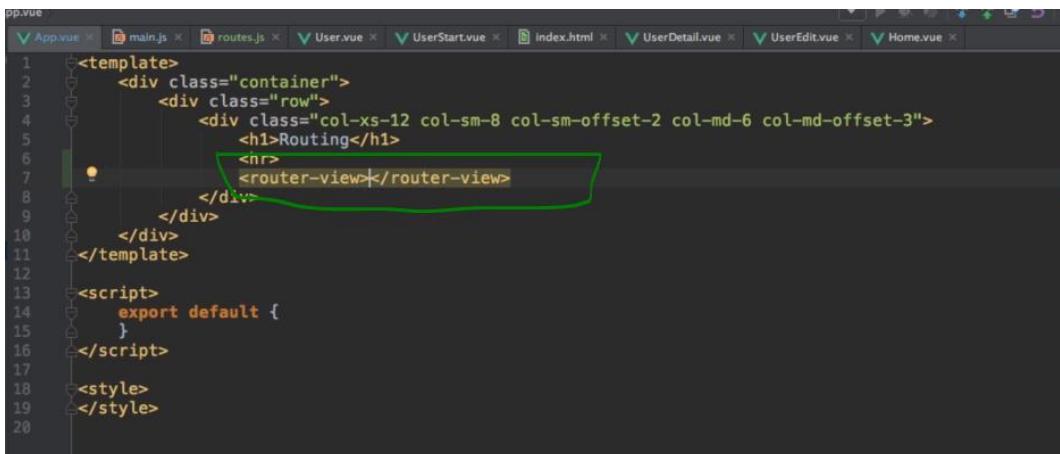
Creare il file routes che è un array di oggetti che rappresentano le varie regole di routing:

```
routes.js
App.vue × main.js × routes.js × User.vue × UserStart.vue × index.html × UserDetail.vue × UserEdit.vue
1 import User from './components/user/User.vue';
2 import Home from './components/Home.vue';
3
4 export const routes = [
5   { path: '', component: Home },
6   { path: '/user', component: User }
7];
```

Poi registro il file di routes in main.js:

```
main.js
App.vue × main.js × routes.js × User.vue × UserStart.vue × index.html ×
1 import Vue from 'vue'
2 import VueRouter from 'vue-router';
3 import App from './App.vue'
4 import { routes } from './routes';
5
6 Vue.use(VueRouter);
7
8 const router = new VueRouter({
9   routes
10 });
11
12 new Vue({
13   el: '#app',
14   router,
15   render: h => h(App)
16 })
17
```

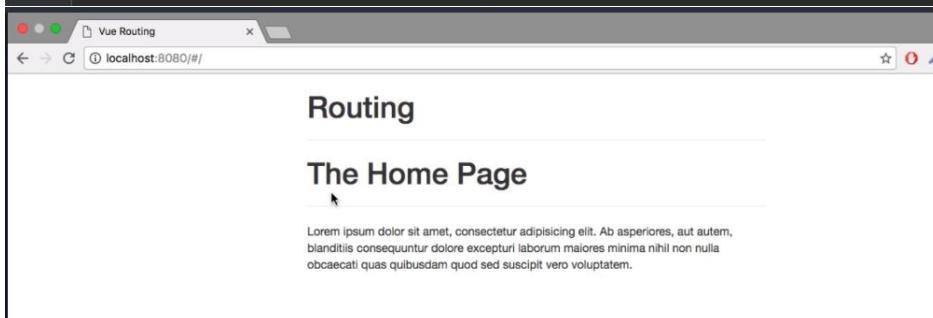
Poi inserisco il route component (che è un componente builtin) in app view



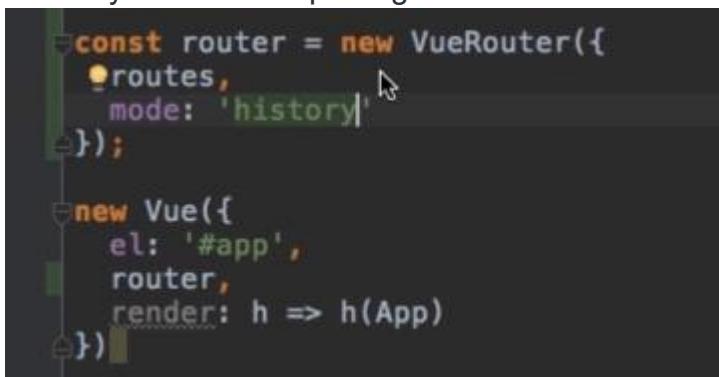
```

1 <template>
2   <div class="container">
3     <div class="row">
4       <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
5         <h1>Routing</h1>
6         <br>
7         <router-view></router-view>
8       </div>
9     </div>
10   </div>
11 </template>
12
13 <script>
14   export default {
15   }
16 </script>
17
18 <style>
19 </style>
20

```



L'history mode serve per togliere # dall'indirizzo



```

const router = new VueRouter({
  routes,
  mode: 'history'
});

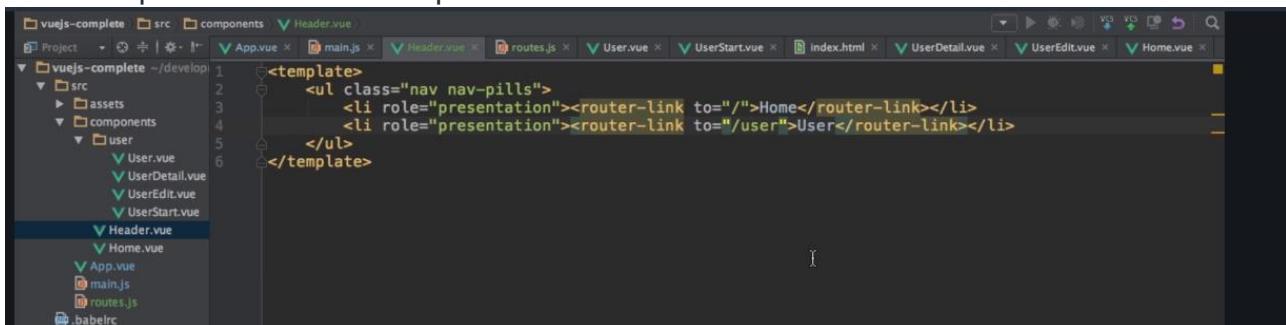
new Vue({
  el: '#app',
  router,
  render: h => h(App)
})

```

Getbootstrap.com molto interessante per recuperare pezzi di html bootstrap

Per creare dei link non usiamo un ancora normale perché con href comporterebbe il ricarico della pagina, invece usiamo un tag che non comporta il ricarico della pagina perché dietro le quinte viene creato un listener che ascolta e cambia html senza fare una nuova richiesta:

creiamo quindi un header component:



```

<template>
  <ul class="nav nav-pills">
    <li role="presentation"><router-link to="/">Home</router-link></li>
    <li role="presentation"><router-link to="/user">User</router-link></li>
  </ul>
</template>

```

E lo registriamo sulla pagina principale:



A screenshot of a video player interface. The video frame shows a person from the chest up, looking slightly to the right. The video player has a dark theme with a play button icon and a progress bar.

```
<template>
  <div class="container">
    <div class="row">
      <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
        <h1>Welcome</h1>
        <br>
        <app-header></app-header>
      </div>
    </div>
  </div>
</template>
<script>
  import Header from './components/Header.vue'
  export default {
    components: {
      appHeader: Header
    }
  }
</script>
<style>
</style>
```

Routing

Home User

The Home Page

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ab asperiores, aut autem, blanditiis consequuntur dolore excepturi laborum maiores minima nihil non nulla obcaecati quas quibusdam quod sed suscipit vero voluptatem.

Per la selezione sul menu selezionato:

```
<template>
  <ul class="nav nav-pills">
    <router-link to="/" tag="li" active-class="active" exact><a>Home</a></router-link>
    <router-link to="/user" tag="li" active-class="active"><a>User</a></router-link>
  </ul>
</template>
```

Per lanciare una navigazione nel codice? Così:

```
<template>
  <div>
    <h1>The User Page</h1>
    <hr>
    <button @click="navigateToHome">Go to Home</button>
  </div>
</template>

<script>
  export default {
    methods: {
      navigateToHome() {
        this.$router.push('/');
      }
    }
  }
</script>
```

O così più prolioso:

```
<template>
  <div>
    <h1>The User Page</h1>
    <hr>
    <button @click="navigateToHome">Go to Home</button>
  </div>
</template>

<script>
  export default {
    methods: {
      navigateToHome() {
        this.$router.push({path: '/'});
      }
    }
  }
</script>
```

Passare parametri al route:

```

export const routes = [
  { path: '', component: Home },
  { path: '/user/:id', component: User }
];

```

E per leggere il parametro passato?:

```

<template>
  <div>
    <h1>The User Page</h1>
    <hr>
    <p>Loaded ID: {{ id }}</p>
    <button @click="navigateToHome" class="btn btn-primary">Go to Home</button>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        id: this.$route.params.id
      }
    },
    methods: {
      navigateToHome() {
        this.$router.push('/');
      }
    }
  }
</script>

```

Cosa succede se abbiamo due route che impostano lo stesso parametro id?,

Routing



quando si clicca sul secondo l'id non cambia in quanto ci troviamo all'interno dello stesso componente ed il componente non è stato ricaricato, per poter quindi aggiornare il parametro occorre quindi ascoltarlo:

```

<template>
  <div>
    <h1>The User Page</h1>
    <hr>
    <p>Loaded ID: {{ id }}</p>
    <button @click="navigateToHome" class="btn btn-primary">Go to Home</button>
  </div>
</template>

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

```

Quindi monitoriamo la proprietà \$route e se cambia impostiamo a this.id= to.param.id dove to.param.id è il nuovo id.

Per creare dei routing annidati facciamo così :

intanto creiamo un userstart component con una lista di users:

```

<template>
  <div>
    <p>Please select a User</p>
    <hr>
    <ul class="list-group">
      <li class="list-group-item" style="cursor: pointer">User 1</li>
      <li class="list-group-item" style="cursor: pointer">User 2</li>
      <li class="list-group-item" style="cursor: pointer">User 3</li>
    </ul>
  </div>
</template>

```

Dentro user vogliamo implementare dei componenti annidati:

```

<template>
  <div>
    <h1>The User Page</h1>
    <hr>
    <button @click="navigateToHome" class="btn btn-primary">Go to Home</button>
  </div>
</template>
<script>
  export default {
    methods: {
      navigateToHome() {
        this.$router.push('/');
      }
    }
  }
</script>

```

Dopodichè si configurano i routes annidati:

```

import User from './components/user/User.vue';
import Home from './components/Home.vue';

export const routes = [
  { path: '', component: Home },
  { path: '/user', component: User, children: [
    { path: '' }
  ] }
];

```

Così come sopra tutto ciò che segue / verrà appeso subito dopo il dominio.

Mentre così verrà appeso al route parente:

```

  { path: '' }
]

```

Quindi configureremo i routes figli:

```

import User from './components/user/User.vue';
import UserStart from './components/user/UserStart.vue';
import UserDetail from './components/user/UserDetail.vue';
import UserEdit from './components/user/UserEdit.vue';
import Home from './components/Home.vue';

export const routes = [
  { path: '', component: Home },
  { path: '/user', component: User, children: [
    { path: '', component: UserStart },
    { path: ':id', component: UserDetail },
    { path: ':id/edit', component: UserEdit }
  ] }
];

```

Quindi inseriamo il tag routerview nel componente user:

```

<template>
  <div>
    <h1>The User Page</h1>
    <hr>
    <button @click="navigateToHome" class="btn btn-primary">Go to Home</button>
    <hr>
    <routerview></routerview>
  </div>
</template>

<script>
  export default {
    methods: {
      navigateToHome() {
        this.$router.push('/');
      }
    }
  }
</script>

```

Ora vogliamo che cliccando sui collegamenti si vada a finire sulla pagina utente specifica, per farlo dobbiamo cambiare userstart:

```

<template>
  <div>
    <p>Please select a User</p>
    <hr>
    <ul class="list-group">
      <router-link tag="li" to="/user/1" class="list-group-item" style="cursor: pointer">User 1</router-link>
      <router-link tag="li" to="/user/2" class="list-group-item" style="cursor: pointer">User 2</router-link>
      <router-link tag="li" to="/user/3" class="list-group-item" style="cursor: pointer">User 3</router-link>
    </ul>
  </div>
</template>

```

Per recuperare l'id dalla pagina userdetail:

```

<template>
  <div>
    <h3>Some User Details</h3>
    <p>User loaded has ID: {{ $route.params.id }}</p>
  </div>
</template>

```

Per creare il bottone di edit basta fare così:

```

<template>
  <div>
    <h3>Some User Details</h3>
    <p>User loaded has ID: {{ $route.params.id }}</p>
    <router-link tag="button" :to="'/user/' + $route.params.id + '/edit'"></router-link>
  </div>
</template>

```

Esiste un modo più sintetico per fare quello che abbiamo fatto sopra:
aggiungere un name al route:

```

import User from './components/user/User.vue';
import UserStart from './components/user/UserStart.vue';
import UserDetail from './components/user/UserDetail.vue';
import UserEdit from './components/user/UserEdit.vue';
import Home from './components/Home.vue';

export const routes = [
  { path: '/', component: Home },
  { path: '/user', component: User, children: [
    { path: '', component: UserStart },
    { path: ':id', component: UserDetail },
    { path: ':id/edit', component: UserEdit, name: 'userEdit' }
  ] }
];

```

E poi creare il link edit così sulla pagina user detail:

```

<template>
  <div>
    <h3>Some User Details</h3>
    <p>User loaded has ID: {{ $route.params.id }}</p>
    <router-link
      tag="button"
      :to="{ name: 'userEdit', params: { id: $route.params.id } }"
      class="btn btn-primary">Edit User</router-link>
  </div>
</template>

```

E come gestire i query parameters?

Con una proprietà che vuole un oggetto con coppia chiave valore

The screenshot shows a Vue.js application interface. At the top, there's a browser-like header with the URL `localhost:8080/user/1/edit?locale=en&q=100`. Below it, a navigation bar has tabs for "Home" and "User", with "User" being active. The main content area is titled "The User Page". It contains a "Go to Home" button and an "Edit the User" button, which is highlighted with a green oval. Above the "Edit the User" button, a green bracket points from the URL in the browser to the `:query` object in the router link code. The router link code itself is also highlighted with a green box.

```

<template>
  <div>
    <h3>Some User Details</h3>
    <p>User loaded has ID: {{ $route.params.id }}</p>
    <router-link
      tag="button"
      :to="{ name: 'userEdit', params: { id: $route.params.id }, query: { locale: 'en', q: 100 } }"
      class="btn btn-primary">Edit User</router-link>
  </div>
</template>

```

E per recuperare via codice i valori del querystring:

The screenshot shows the code for the `UserEdit.vue` component. The template section contains:

```

<template>
  <div>
    <h3>Edit the User</h3>
    <p>Locale: {{ $route.query.locale }}</p>
    <p>Analytics: {{ $route.query.q }}</p>
  </div>
</template>

```

È possibile dare un nome non solo al route ma anche al routerview:

togliamo l'header dall'app component

The screenshot shows the code for the `App.vue` component. The template section contains:

```

<template>
  <div>
    <div>
      <app-header></app-header>
      <router-view></router-view>
    </div>
  </div>
</template>

```

E sostituiamo con:

un default router-view e de altri routerview con un nome.



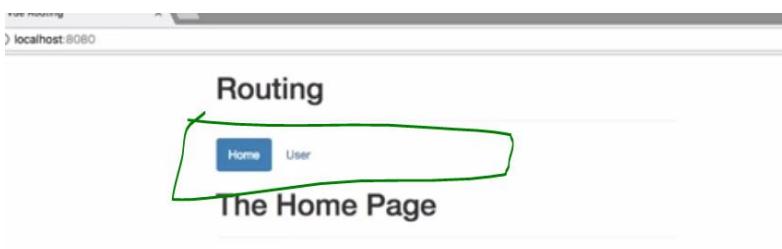
```
<template>
  <div class="container">
    <div class="row">
      <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
        <h1>Routing</h1>
        <br>
        <routerview name="header-top"></routerview>
        <routerview></routerview>
        <routerview name="header-bottom"></routerview>
      </div>
    </div>
  </div>
</template>
<script>
  import Header from './components/Header.vue';
  export default {
    components: {
      appheader: Header
    }
  }
</script>
<style>
</style>
```

Dopodichè nel codice togliamo component ed aggiungiamo components, qui infatti nella pagina di default verrà attivato il default routes ed il routes header-top per il componente header, mentre per il componente user verrà attivato il router header-bottom per il componente header:

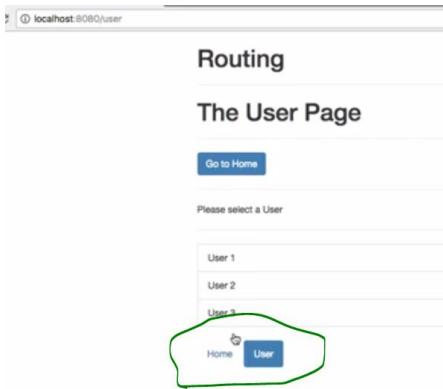
```
import User from './components/user/User.vue';
import UserStart from './components/user/UserStart.vue';
import UserDetail from './components/user/UserDetail.vue';
import UserEdit from './components/user/UserEdit.vue';
import Home from './components/Home.vue';
import Header from './components/Header.vue';

export const routes = [
  { path: '', name: 'home', components: {
    default: Home,
    'header-top': Header
  }},
  { path: '/user', components: {
    default: User,
    'header-bottom': Header
  }, children: [
    { path: '', component: UserStart },
    { path: ':id', component: UserDetail },
    { path: ':id/edit', component: UserEdit, name: 'userEdit' }
  ]}
];
```

Quindi per la home il menu è al top:



E per la use è al bottom:__



Quindi ci permette di visualizzare un pezzo di html diverso in un punto diverso a seconda del route che sceglieremo per navigare.

Redirecting:

se si scrive nella barra indirizzi redirect-me si viene reindirizzati a user

```
}, children: [
  { path: '', component: UserStart },
  { path: ':id', component: UserDetail },
  { path: ':id/edit', component: UserEdit, name: 'userEdit' },
],
{ path: '/redirect-me', redirect: '/user' }
```

Oppure con oggetto:

```
];
  { path: '/redirect-me', redirect: { name: 'home' } }
```

Per il catch all:

```
: { path: '*', redirect: '/' }
```

```
<template>
<div>
  <h3>Some User Details</h3>
  <p>User loaded has ID: {{ $route.params.id }}</p>
  <router-link
    tag="button"
    :to="link"
    class="btn btn-primary">Edit User
  </router-link>
</div>
</template>

<script>
export default {
  data() {
    return {
      link: {
        name: 'userEdit',
        params: {
          id: this.$route.params.id
        },
        query: {
          locale: 'en',
          q: 100
        },
        hash: '#data'
      }
    }
  }
}</script>
```

