



**VueJS Basic Document**

**By Phan Dai**

**[MY REVIEW](#_Toc73294668)** [2](#_Toc73294668)

[**MY COMPANY PROJECT** 2](#_Toc73294669)

[**OTHER FRAMEWORK** 2](#_Toc73294670)

[**Introduction** 3](#_Toc73294671)

[**The Vue instance** 13](#_Toc73294672)

[**Temaplate Syntax** 19](#_Toc73294673)

[**Computed Properties and Watchers** 22](#_Toc73294674)

[**Class and Style Bindings** 30](#_Toc73294675)

[**Conditional Rendering** 33](#_Toc73294676)

[**List Rendering** 35](#_Toc73294677)

[**Event Handling** 40](#_Toc73294678)

[**Form Input Bindings** 43](#_Toc73294679)

[**Components Basics** 47](#_Toc73294680)

# **MY REVIEW**

*On May5 , 2021 ; At Lampart Inc ; To join a project with Vue and Laravel*

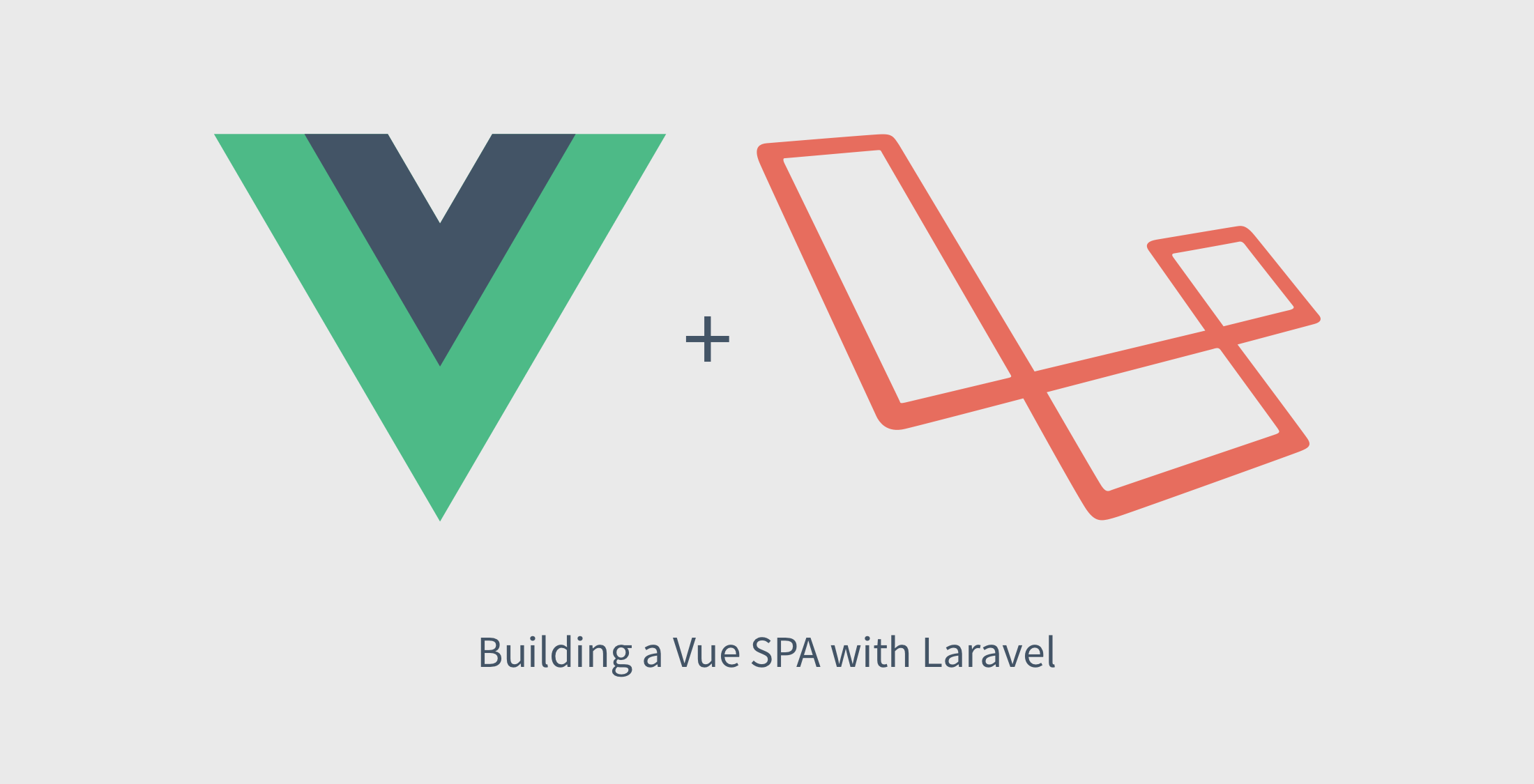
Vào lúc đầu, với một thời gian ngắn áp dụng Vue, Nó rất dễ với những người bắt đầu mà đã kiến thức căng bản về JavaScript trước đó. Cấu trúc không khác gì về React hoặc JS Framework mà tôi được nghe trước đó. Cấu trúc bao gồm 3 vùng:

1️. Template: < template > </ template > - chứa DOM, Virtual DOM như .JSX, .Handlebars,..

2. Script : < script > </ script > - xử lý chính của File.

3. Style : < style > </ style > - quy định các định dạng của DOM.

# **MY COMPANY PROJECT**



# **OTHER FRAMEWORK**

**NUXT JS**



The Intuitive Vue Framework

Xây dựng Vue.js application với sự tự tin khi dùng NuxtJS. Một framework mở khiến cho việc phát triển web đơn giản và mạnh mẽ.



**VUETIFY**

Material Design Framework

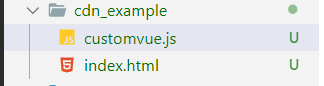
Vuetify là một Vue UI Library với nhiều Material Components có sẵn. Không cần kĩ năng thiết kế — Những việc ứng dụng Vuetify để tạo ra amazing applications đang nằm trên bàn tay của bạn.

# **Introduction**

Vue / Vju: / là một framework tân tiến sử dụng để xây dựng giao diện người dùng. Vue được thiết kế có thể thích nghi nhanh chóng . Thư viện core chỉ tập trung vào phần View và tích hợp với những thư viện khác trên các project hiện tại.

Mặt khác, Vue phù hợp hoàn toàn với Single Page Application khi kết hợp với công cụ hiện đại và thư viện hỗ trợ.

**Bắt đầu với Folder Hello World example bằng cách sử dụng CDN**



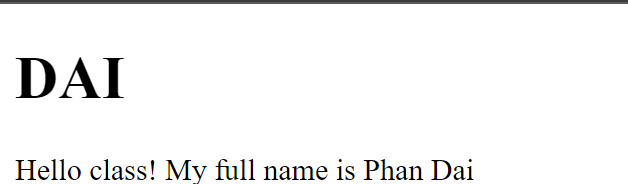
**index.html**



**customvue.js**



Kết quả :



Ta có thể thử thay đổi ngay app.message thành 1 string bất kì và có thể thấy áp dụng ngay trên trình duyệt – Vue có tính **reactive**

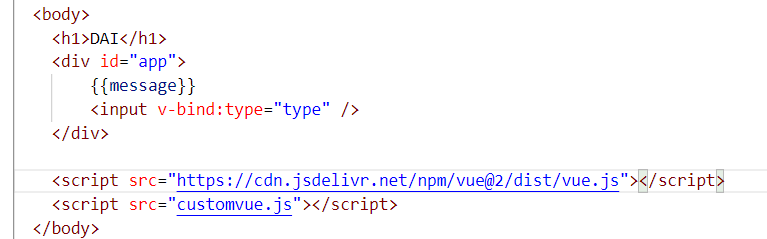
**Vue instance**

Bây giờ ta có thể biết nguyên tắc hoạt động của ứng dụng VueJS đơn giản. Ta có thể hoàn toàn kiểm soát 1 DOM ( #app ). HTML là nơi render, đầu ra, nhưng mọi thứ khác ta phải phụ thuộc vào 1 thứ gọi là **Vue instance**

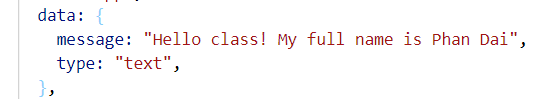
Ta có thể bind các thuộc tính trên DOM như sau

**index.html**

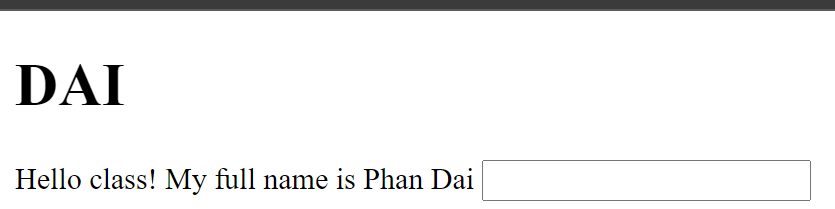
Vì vue chỉ áp dụng ở DOM div#app nên những thứ khác khi áp dụng VueJS thì phải viết trong div#app.



**customvue.js**

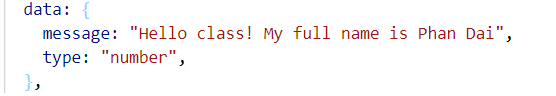


Kết quả :

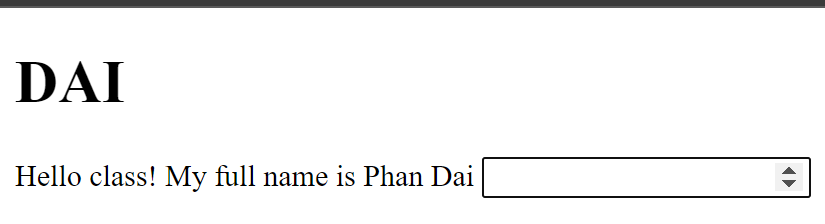


Ta có thể thay giá trị của type trong customvue.js thành một giá trị khác để xem nó thay đổi như thế nào

**customvue.js**



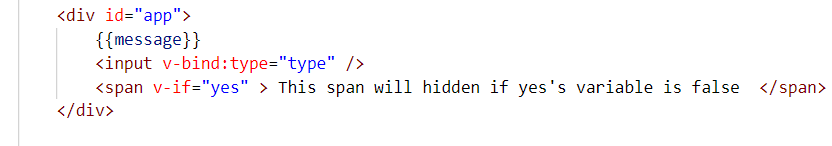
Kết quả:



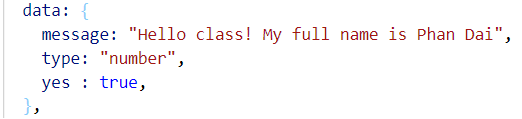
**Note :** Điều này hay được áp dụng trong trường hợp bind những thuộc tính quan trọng quyết định đến DOM

**v-if**

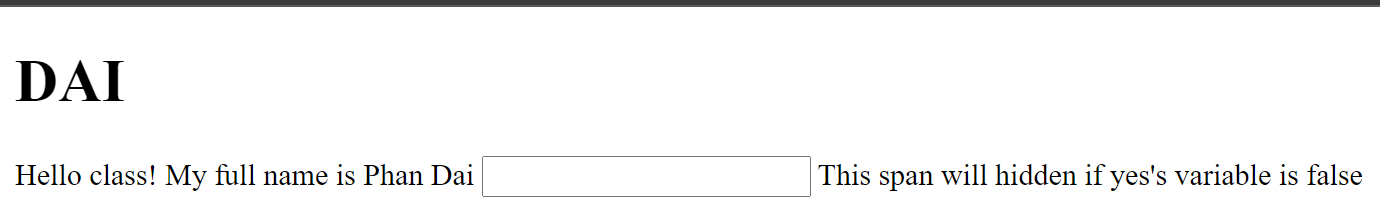
**index.html**



**customvue.js**

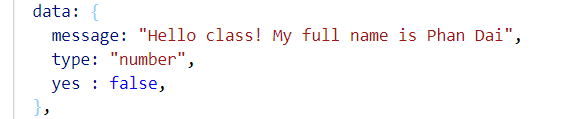


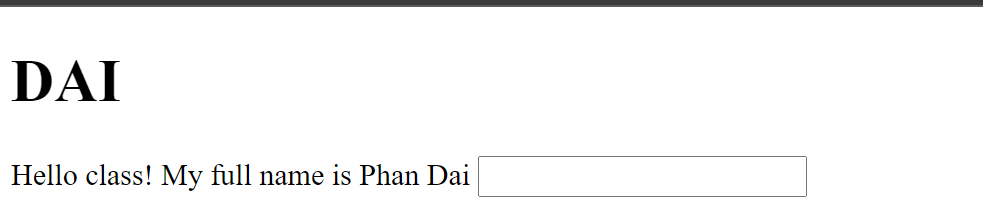
Kết quả :



Ngược lại.

**customvue.js**



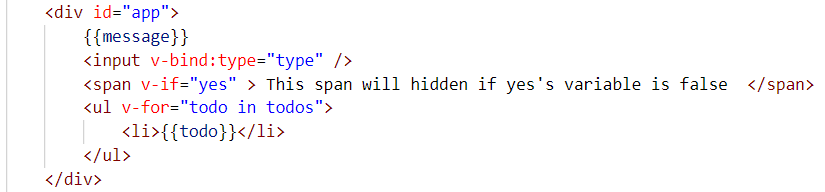


Vue còn áp dụng tự động hiệu ứng transition khi 1 DOM nào đó được cập nhật bởi Vue.

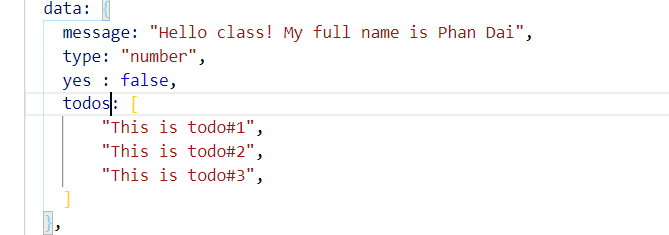
**v-for**

v-for có thể được sử dụng để hiển thị danh sách dữ liệu từ Mảng:

**index.html**



**customvue.js**



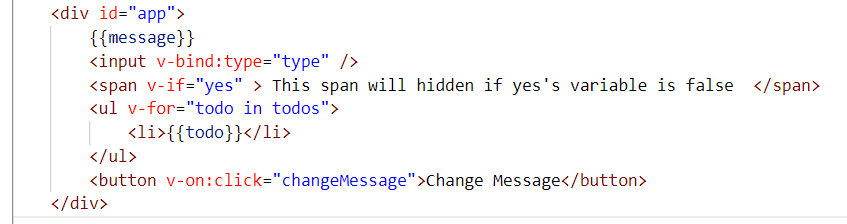
Kết quả :



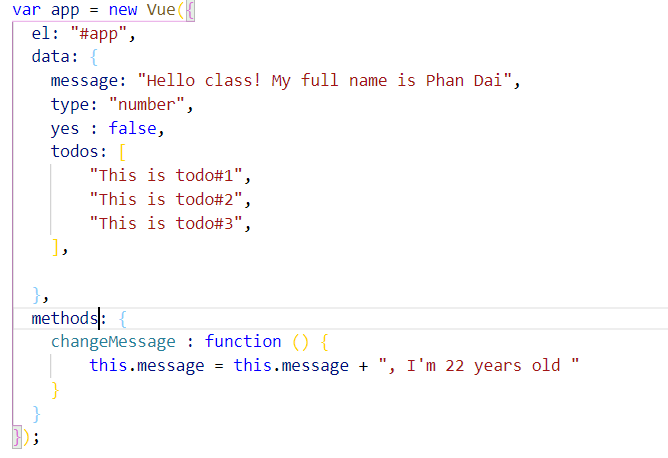
**v-on**

v-on dùng để lắng nghe những event nào đã được định nghĩa trong hệ thống.

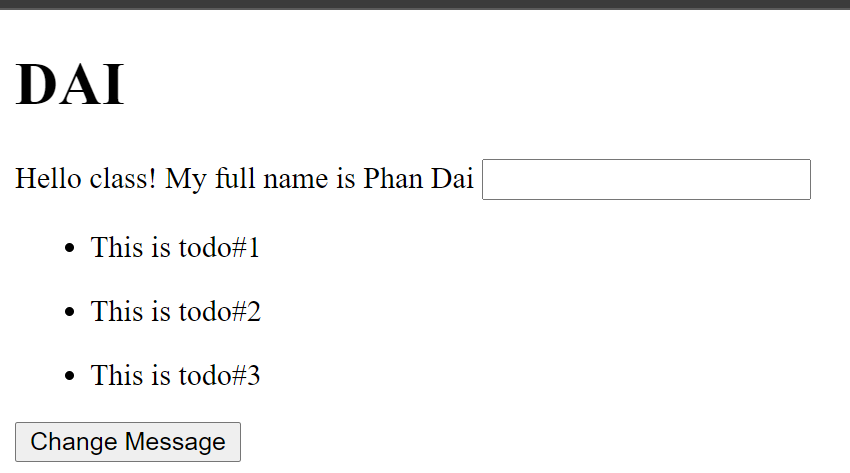
**index.html**

****

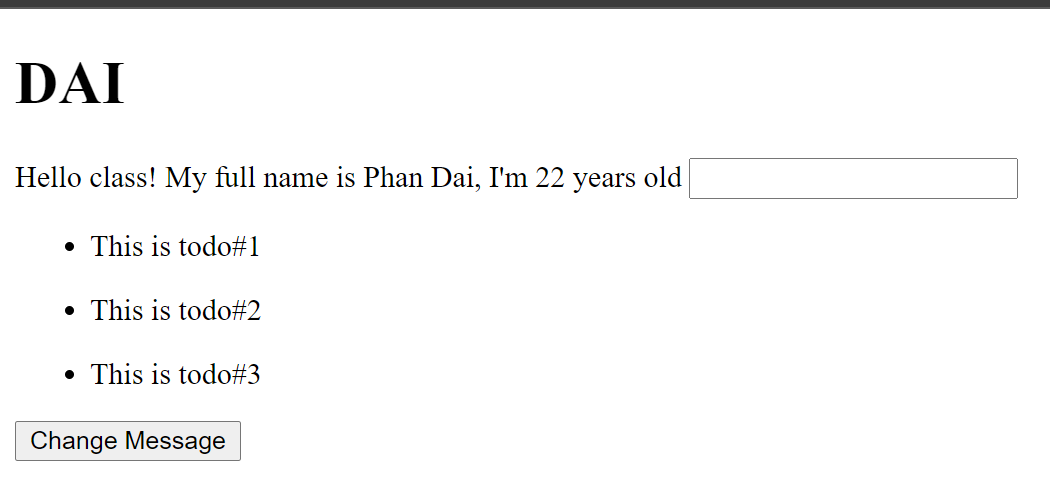
**customvue.js**

****

Kết quả :



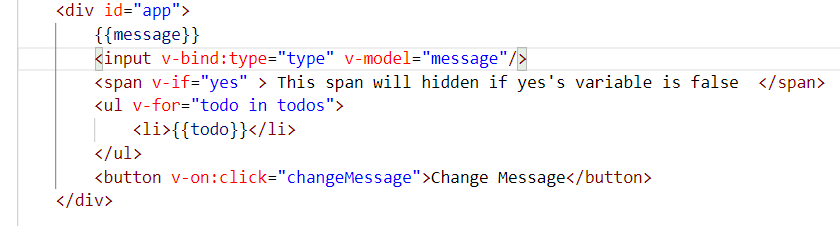
Khi ấn nút Change Message:



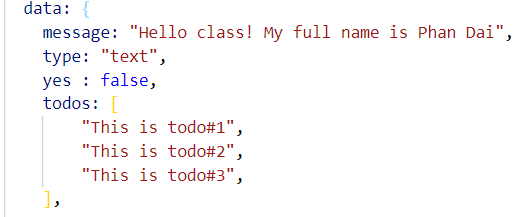
**v-model**

Hỗ trợ việc binding 2 chiều, cập nhật ngay tức thì, thay đổi var và hiển thị var đồng thời

**index.html**

****

**customvue.js**



Kết quả :



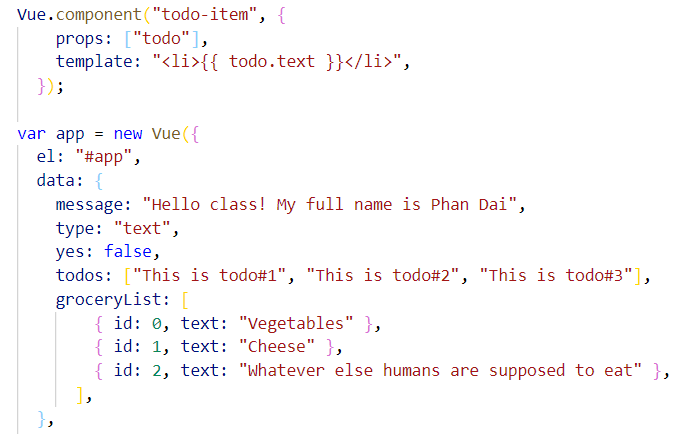
**Component**

Hệ thống phân tầng là một hệ thống quan trọng trong Vue, bởi vì nó là một nguyên tắc giúp cho việc xây dựng một ứng dụng quy mô lớn bằng những component nhỏ, khép kín và tái sử dụng được. Nếu xuy xét, mọi web app có thể được phân thành một cây component.



Trong Vue, một component là một Vue instance quan trọng với những lựa chọn được định nghĩa bởi người dùng. Đăng kí một component trong Vue rất đơn giản:

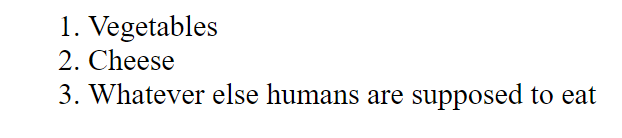
**customvue.js**



**index.html**



Kết quả :

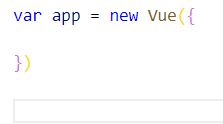


**Note :** Vue.component( “COMPONENT\_NAME” , JSON ) để khai báo 1 component con trong 1 Component cha.

# **The Vue instance**

Mục đích chính của guide này là mô tả cách sử dụng Obj option để sáng tạo ra những hành vi mong muốn.

Cách tạo ra 1 instance ( một ví dụ )



Một ứng dụng Vue bao gồm 1 Vue instance chính được tạo bằng new Vue(), được sắp xếp vào một cây chứa các thành phần lồng nhau và có thể tái sử dụng. Ví dụ, một cây thành phần của ứng dụng ghi chú (todo) có thể như thế này:

Root Instance

└─ TodoList

├─ TodoItem

│ ├─ TodoButtonDelete

│ └─ TodoButtonEdit

└─ TodoListFooter

├─ TodosButtonClear

└─ TodoListStatistics

**Note :** Vue component và Vue instance đều có chung 1 option OBJ.

**Data và methods**

Vue sẽ update theo sự thay đổi của data, mỗi khi giá trị trong Obj data thay đổi thì Vue sẽ render lại DOM

// Our data object

var data = { a: 1 }

// The object is added to a Vue instance

var vm = new Vue({

data: data

})

// Getting the property on the instance

// returns the one from the original data

vm.a == data.a // => true

// Setting the property on the instance

// also affects the original data

vm.a = 2

data.a // => 2

// ... and vice-versa

data.a = 3

vm.a // => 3

Nhưng khi tạo ra thêm 1 biến b nào đó thì nó sẽ không gây Vue render do nó không tồn tại lúc instance được tạo ra

vm.b = 'hi'

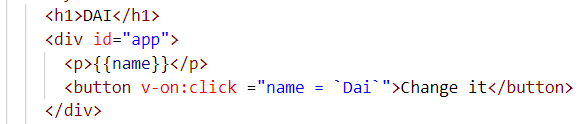
Vậy để hệ thống không bắt được sự thay đổi, ta có thể dùng 1 trong 2 cách sau :   
Object.freeze(dai); // mean not change

// giống nhau

const dais = dai;

Thử ép nó thay đổi bằng button của HTML

**index.html**



**customvue.js**



Kết quả

Before After

Note : Button khi v-on:\*event thì có thể ghi ngắn gọn bằng thân function

<button v-on:click ="name = `Dai`">

Trong Vue, có một số thuộc tính và phương thức của Instance hữu ích được phân biệt bằng $ khác với người dùng định nghĩa. Ví dụ:

var data = { a: 1 }

var vm = new Vue({

el: '#example',

data: data

})

vm.$data === data // => true

vm.$el === document.getElementById('example') // => true

// $watch is an instance method

vm.$watch('a', function (newValue, oldValue) {

// This callback will be called when `vm.a` changes

})

Qua ví dụ, ta thấy thường là dùng để gọi những hàm hay dùng.

Note : Sau khi khai báo data: data thì có thể gọi vm.a ( a là 1 key của data ) mà khi muốn lấy toàn bộ data có thể gọi vm.$data

**Instance Lifecycle Hooks**

Khi tạo 1 Vue Instance thì trải qua rất nhiều step – ví dụ, nó cần để set up data, tìm element, biên dịch template, cập nhật Vue instance vào DOM và update khi data thay đổi. Cùng với đó, nó chạy rất nhiều function được gọi là lifecycle hooks, tạo cho người dùng cơ hội để cấu hình code vào những giai đoạn định nghĩa.

Ví dụ:

Created – Contructor

Updated – onChange

Destroyed – Destructor

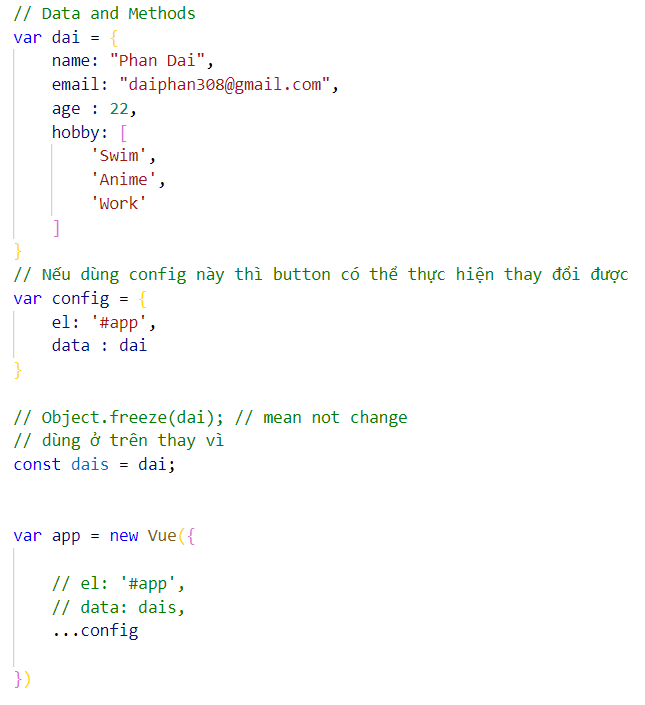
Mọi thứ có thể được gọi ra bằng this vì thế nên hạn chế sử dụng **arrow functions**

**Lifecycle Diagram**

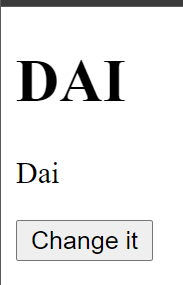
Ta không cần hiểu hết cái mô hình này, khi ta làm ta sẽ rõ hơn.

Sau các ví dụ, ta thử test trường hợp này , tạo ra 1 config bên ngoài và unpack bên trong để xem Obj Vue có nhận được hay không ?

Code:



* Suy nghĩ : nhận được vì đó là Option Obj
* Thực hành :

Before After





# **Temaplate Syntax**

Template cơ bản nhất là :

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

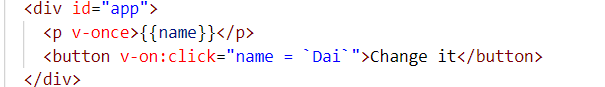
Dựa vào cái dấu này mà nếu biến bên trong thay đổi thì Vue sẽ render lại template

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

Nếu muốn định nghĩa sử dụng render 1 lần duy nhất, không update data tại đó thì thêm

v-once

**index.html**



**customvue.js**

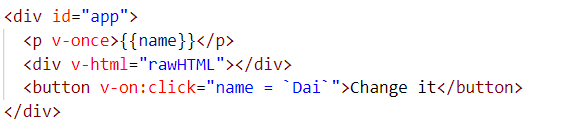


**Raw HTML**

Khi {{ variable }} chỉ đưa ra giá trị của biến là String, và ta muốn giữ nguyên giá trị ấy và render ra HTML chứ không phải String thì sử dụng v-html

VD:

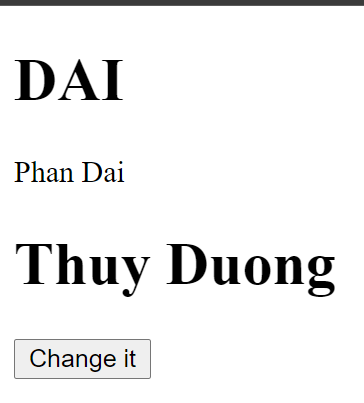
**index.html**



**customvue.js**



Kết quả :



**v-if = Dùng để cho phép hiển thị nếu điều kiện true hoặc giấu nếu ngược lại.**

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

**v-bind: Dùng để lấy ánh xạ tên biến từ data rồi sử dụng giá trị từ biến ấy**

<a v-bind:href="url"> ... </a> // Sử dụng giá trị của biến biến url

**v-on: Dùng để lấy ánh xạ tên hàm từ methods rồi sử dụng thân hàm ấy**

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

**v-on:submit.prevent**

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

**(event ) => event.preventDefault( )**

document.addEventListener(‘submit’, (event) => {

event.preventDefault()

onSubmit();

})

[**v-bind Shorthand**](https://vuejs.org/v2/guide/syntax.html#v-bind-Shorthand)

<!-- full syntax -->

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

<!-- shorthand -->

<a :href="url"> ... </a>

<!-- shorthand with dynamic argument (2.6.0+) -->

<a :[key]="url"> ... </a>

[**v-on Shorthand**](https://vuejs.org/v2/guide/syntax.html#v-on-Shorthand)

<!-- full syntax -->

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

<!-- shorthand -->

<a @click="doSomething"> ... </a>

<!-- shorthand with dynamic argument (2.6.0+) -->

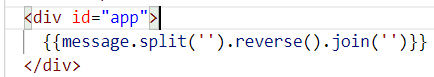
<a @[event]="doSomething"> ... </a>

# **Computed Properties and Watchers**

**Computed Properties**

Vấn đề thường gặp trong lúc render là : 1 giá trị khi render có khả năng trải qua nhiều hàm khác nhau đế đạt được kết quả mong muốn

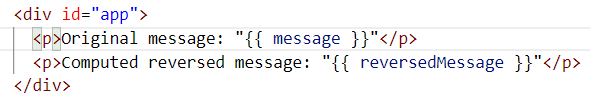
**index.html**



Đó là lí do mà chúng ta cần sử dụng computed property.

Với ví dụ trên, ta có thể viết như sau:

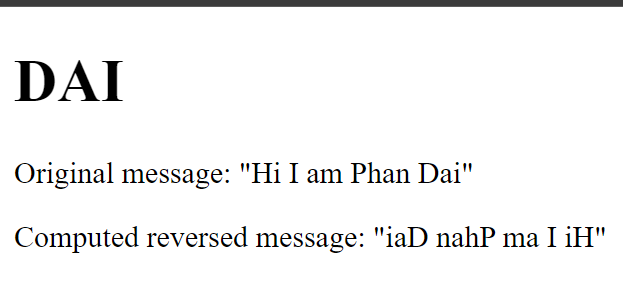
**index.html**



**customvue.js**



Kết quả



Khi sử dụng thì chỉ cần gọi this.reversedMessage bên trong class hoặc vm.reversedMessage bên ngoài class.

console.log(vm.reversedMessage) // => 'olleH'

vm.message = 'Goodbye'

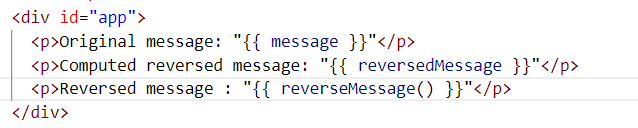
console.log(vm.reversedMessage) // => 'eybdooG'

Note : Mặc dù reversedMessage không cần khai báo trong data, nhưng khi ở trong key **computed** cũng được tính là khai báo và sẽ update sau khi message bị thay đổi.

**Computed Caching vs Methods**

Ta có thể đạt được kết quả tương tự nếu đặt trong method và gọi ra như gọi hàm.

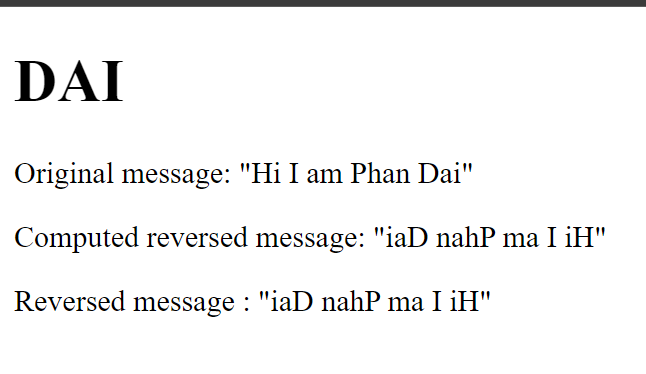
**index.html**



**customvue.js**



Kết quả :



2 cách thức đều cho ra kết quả giống nhau nhưng điểm khác biệt lớn nhất là computed properties chỉ thay đổi khi mà dependencies update và khi không thay đổi thì nó vẫn lưu lại giá trị cũ của dependencies mà không cần phải chạy lại hàm như cách dùng methods.

Có những cách để computed property không update với dependencies bằng cách

**customvue.js**



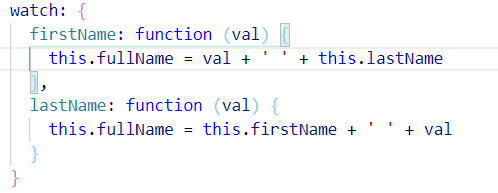
* Không sử dụng dependencies.

Cách computed property được sử dụng nhiều cho việc khi ta cần load dữ liệu cực lớn từ API và thực thi nhiều việc tính toán. Với computed property lưu trữ bằng cache giúp việc tính toán đi tính toán lại là không cần thiết và chỉ thật sự thay đổi khi dependencies thay đổi.

**Computed vs Watched Property**

Giống với cách theo dõi sự thay đổi của biến giống với hook onEffect của React: **watch** properties. Khi mà ta có data mà cần phải thay đổi dựa trên data khác, nên vận dụng **watch** – đặc biệt đối với những lập trình viên đã có nền tảng AngularJS. Tuy nhiên, khuyên rằng nên thường sử dụng computed hơn là lệnh **watch** callback. Như ví dụ dưới đây:

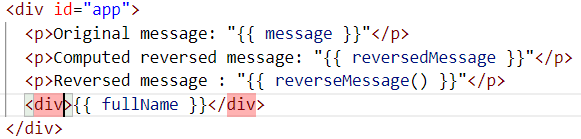
**customvue.js**



Hoặc



**index.html**

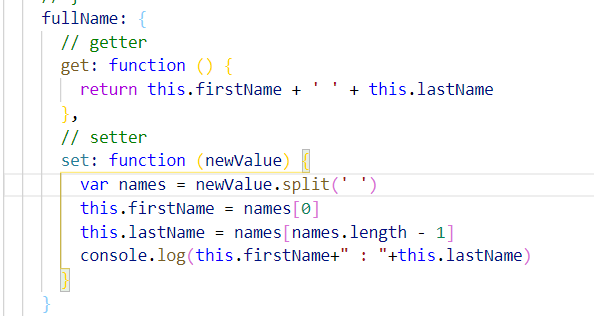


Qua ví dụ ta thấy một điều là watch cũng sẽ thực thi y chang computed nhưng cách thực thi là điểm khác biệt. Trong khi computed chỉ quan tâm đến biến lưu kết quả cuối cùng còn watch là quan sát sự thay đổi sau đó thực thi code của người dùng theo từng biến định nghĩa

VD: Nếu ta có 101 biến định nghĩa bao gồm fullName ở data thì ta phải điền 100 dòng để gắn vào fullName.

**Computed Setter**

**customvue.js**

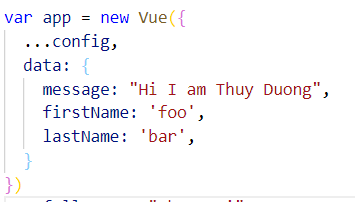




Kết quả :



Note : Khi định nghĩa ở computed fullName thì không nên định nghĩa lại ở data sẽ gây confict

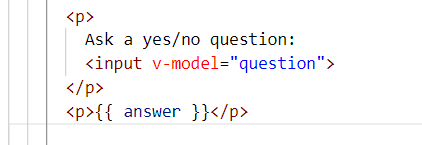


Note : Điều này vẫn cho phép và data lúc sau sẽ ghi đè lên data lúc trước

**Watchers**

Trong khi computed được sử dụng ở mọi case thì có những lúc watch cần thiết hơn . Đó là lí do Vue tạo ra cách để react với thay đổi dữ liệu thông qua watch. Khi thực hiện một bất đồng bộ hoặc một tương tác phức tạp dùng để thay đổi dữ liệu.

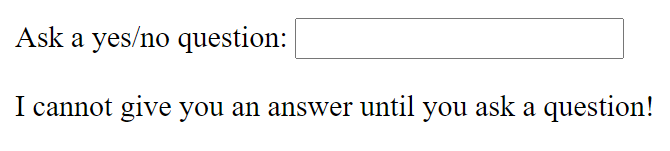
**index.html**

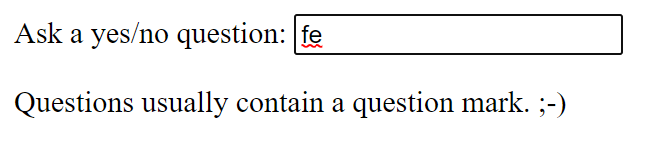


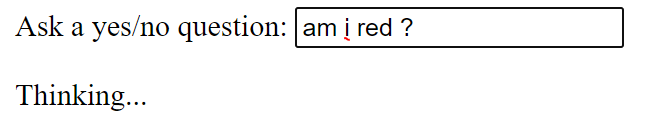
**watchComponent.js**

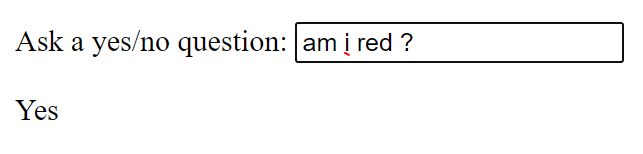
****

Kết quả :





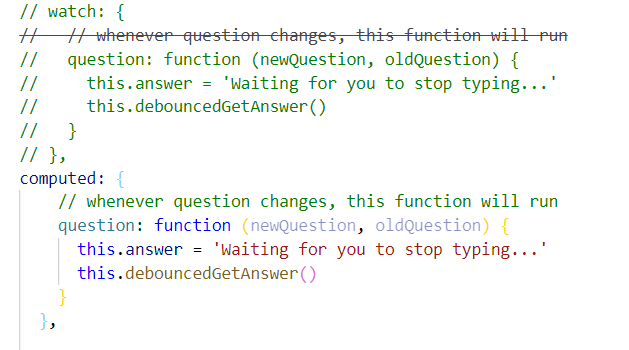




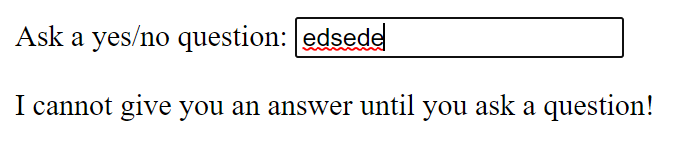
Trong trường hợp này, sử dụng watch cho phép ta có thể thực thi một bất đồng bộ ( truy cập API ), hạn chế số lần thực hiện hoạt động đó, và đặt các trạng thái trung gian cho đến khi có được kết quả cuối cùng. Những điều trên sẽ không có thể nếu chỉ dùng computed.

Thử :

**watchComponent.js**



Kết quả :



Do một phần là computed khi định nghĩa thì data sẽ báo trùng khi có thành phần đã định nghĩa nên việc này là do loạn scope khiến question khi đến method sẽ bị undefined.

**Note** : Watch cần người dùng định nghĩa tên biến ở data và computed thì không cần, debounce là 1 hàm của lodash sẽ hạn chế số lần react data của API trả ra, chỉ khi nào user đã hoàn tất việc nhập thì mới làm ajax request

# **Class and Style Bindings**

Nhu cầu phổ biến khi data binding là làm chủ danh sách các class và styles của element. Vì chúng đều là attributes, ta có thể sử dụng v-bind: ta chỉ cần gán final String trong biểu thức vào sau giá trị thuộc tính v-bind. Tuy nhiên việc can thiệp vào string, nối string, rất khó chịu và dễ gặp bug. Vì thế, Vue cung cấp sự nâng cấp đặc biệt khi sử dụng **v-bind** với class và style. Ngoài ra, các biểu thức có thể đánh giá được nó là object hay arrays.

**Binding HTML Classes**

**Object Syntax**

Chúng ta có thể v-bind:class để toggle class động :

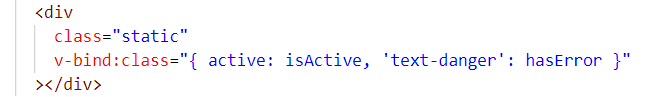


Chỉ khi isActive có kết quả là true thì class sẽ trở thành active

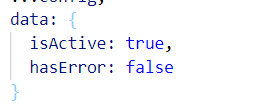


Ta có thể toggle nhiều class bằng thêm field trong object. Thêm nữa, v-bind:class có thể cùng tồn tại với thuộc tính class. Tạo ra template như dưới

**index.html**



**customvue.js**

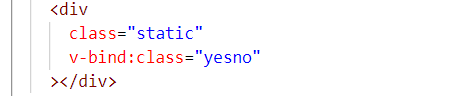


Kết quả

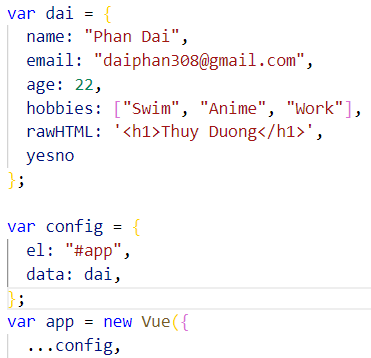


Nếu quá nhiều biến để quy định một class thì ta có thể để dạng biến và khai báo bên data

**index.html**

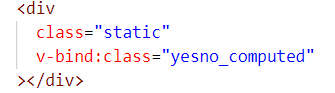


**customvue.js**

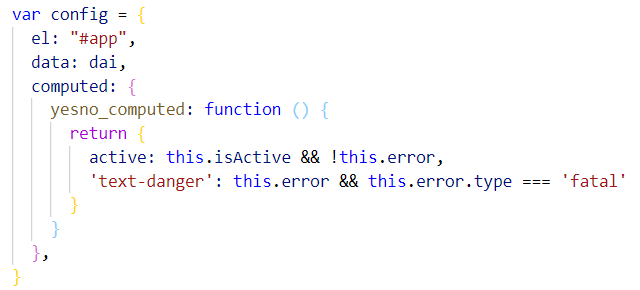


Có thể sử dụng với computed properties

**index.html**



**customvue.js**

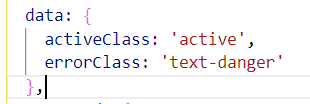


Kết quả



**Array Syntax**

**customvue.js**



**index.html**



Kết quả :



Khi dùng chung với biểu thức có điều kiện thì nên sử dụng như bên dưới



Hoặc kết hợp cả 2 cách



**With Components**

Vue.component('my-component', {

template: '<p class="foo bar">Hi</p>'

})

Ta có thể thêm class khi sử dụng

<my-component class="baz boo"></my-component>

Khi render HTML sẽ như thế này



Cũng có thể kết hợp với class binding

<my-component v-bind:class="{ active: isActive }"></my-component>

Khi isActive bằng true



**Binding Inline Styles**

Style thì ta cũng có thể thực hiện theo 2 cách Object syntax và Array syntax

<div v-bind:style="styleObject"></div>

Hoặc

<div v-bind:style="[baseStyles, overridingStyles]"></div>

Version từ 2.3.0+ có thể truyền một array cho 1 Object

<div v-bind:style="{ display: ['-webkit-box', '-ms-flexbox', 'flex'] }"></div>

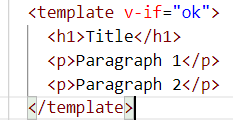
**Note** : Điều này rất giúp ích cho khi chuyển event handling mà sẽ được tiếp xúc sau. Ta có thể tương tác với class con nếu template của component ấy chỉ trả ra đúng 1 HTML tag. Tùy theo ý định của mỗi người mà có thể dùng 1 class, 1 stye hoặc nhiều class, nhiều style.

# **Conditional Rendering**

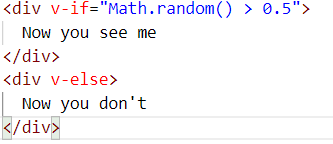
**v-if**



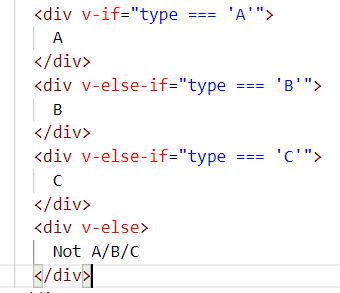
**v-if on <template>**

****

**v-else**

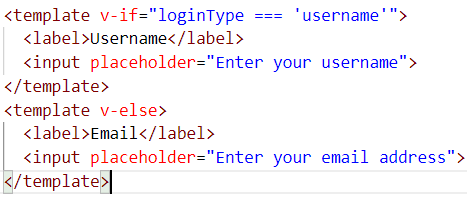
****

**v-else-if**

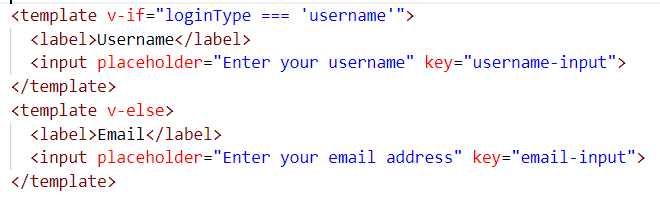
****

**Controlling Reusable Elements with key**

Trường hợp 1 : input sẽ không thay đổi cho dù có switch loginType

****

Trường hợp 2 : input sẽ reset khi switch loginType



**v-show**

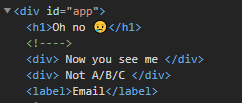
****

**v-if vs v-show**

**v-if** là câu lệnh điều kiện chuẩn bởi vì nó đảm bảo sự kiện và component con bên trong block hoàn toàn bị tiêu hủy và sẽ tái tạo khi toggle

Nếu condition = false lúc init thì nó sẽ không làm gì cả - điều kiện sẽ không được render trừ khi condition trở thành true .





**v-show** thì đơn giản hơn – element luôn render mặc dù có điều kiện khởi tạo, CSS-based toggling.

v-if tốn toggle costs khi v-show tốn inital render costs. Nên dùng v-show nếu toggle element thường xuyên và

nên dùng v-if nếu điều kiện không chắc thay đổi trong lúc chương trình chạy





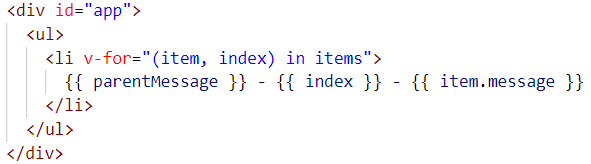
**v-if with v-for**

**v-if** xài chung với **v-for,** thì v-for sẽ có độ ưu tiên cao hơn v-if

# **List Rendering**

**v-for thông thường sử dụng Array**

**index.html**

****

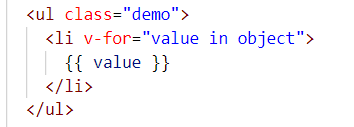
**customvue.js**

****

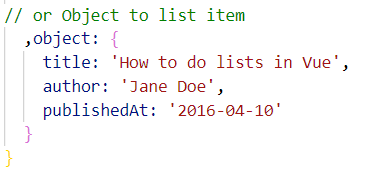
Hoặc

**v-for với Object**

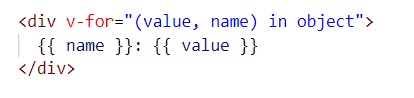
**index.html**



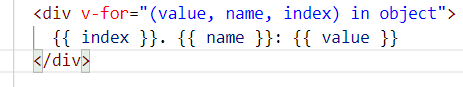
**customvue.js**



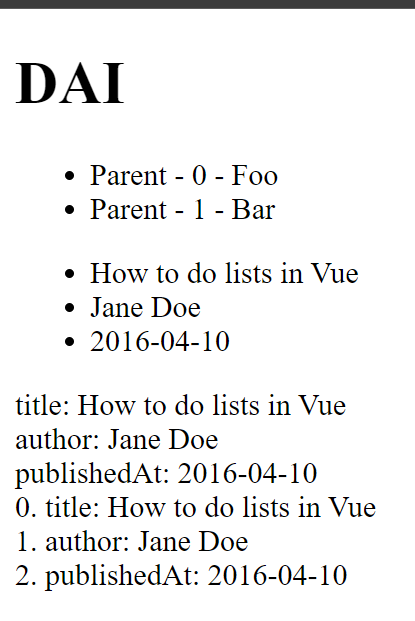
Ngoài ra có nhiều cách để list item dưới dạng Object



Hoặc thêm 1 field index trong object



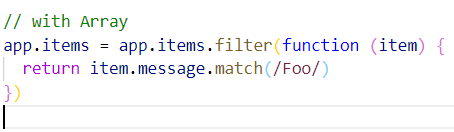
Kết quả :



**Thao tác với Array**

* push()
* pop()
* shift()
* unshift()
* splice()
* sort()
* reverse()

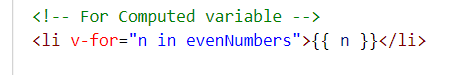
**Replacing an Array e.g. filter(), concat() and slice(),**

****

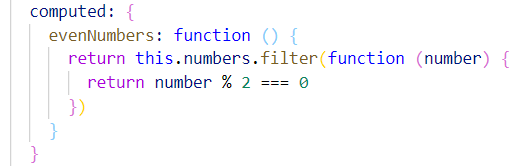
**Displaying Filtered/Sorted Results**

**Using computed**

**index.html**

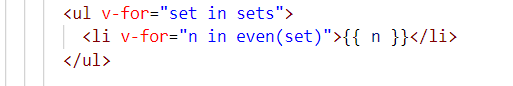


**customvue.js**



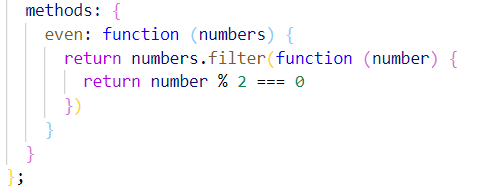
**Using methods**

**index.html**

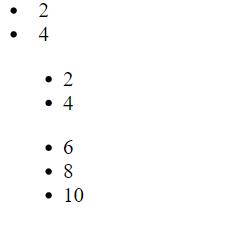
****

**customvue.js**

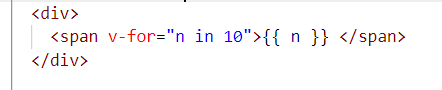
****

****

Kết quả :



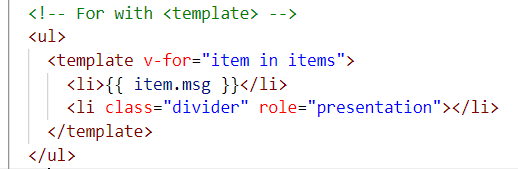
**v-for with a Range**

****

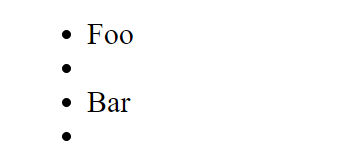
Kết quả :



**v-for on a <template>**

****

Kết quả :

****

**&**

**v-for with a Component**

<my-component

v-for="(item, index) in items"

v-bind:item="item"

v-bind:index="index"

v-bind:key="item.id"

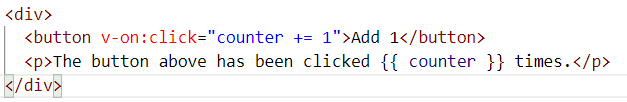
></my-component>

# **Event Handling**

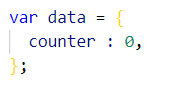
**Listening to Events**

**#1**

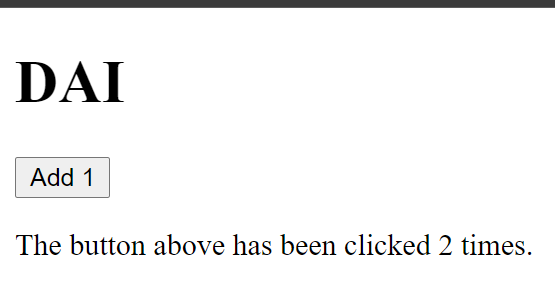
**index.html**



**customvue.js**

****

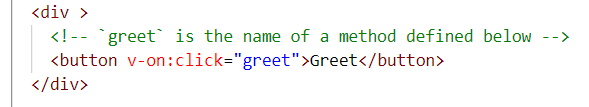
Kết quả :



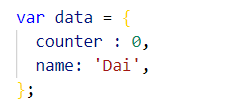
**Note** : Nếu không truyền tên hàm thì nên truyền

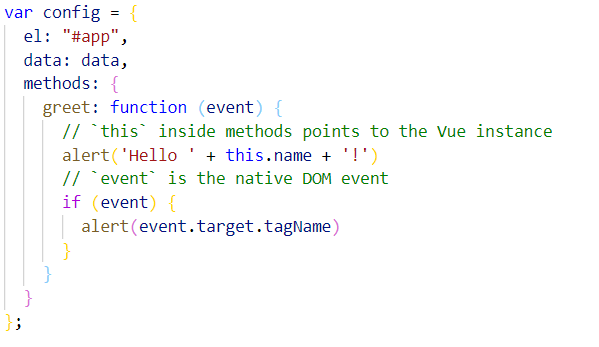
**#2**

**index.html**

****

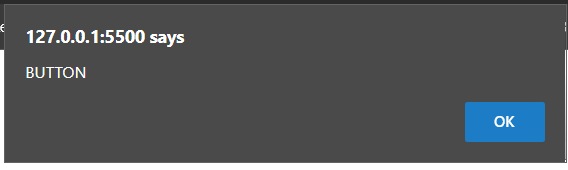
**customvue.js**





Kết quả :



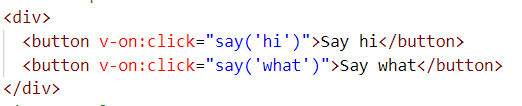


Ngoài ra có thể gọi ra từ chính file JavaScript

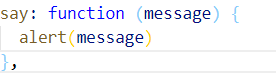
app.greet() // => 'Hello Vue.js!'

**#3**

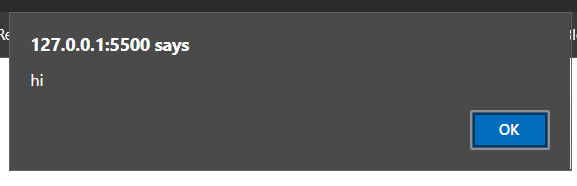
**index.html**

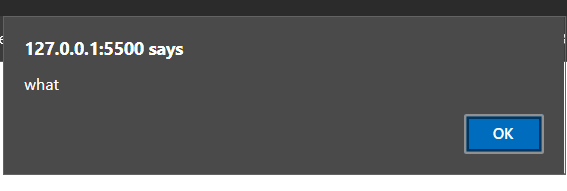
****

**customvue.js**



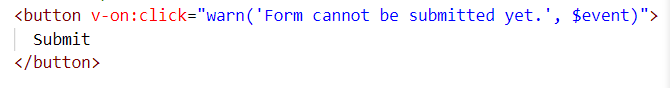
Kết quả :



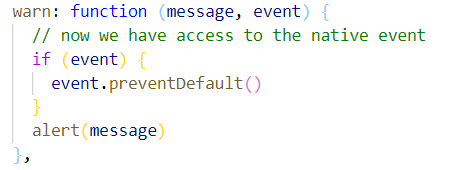


**#4**

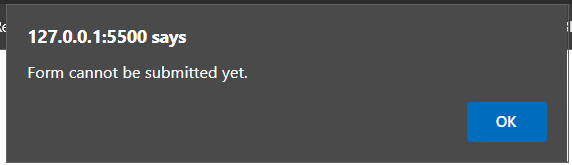
**index.html**

****

**customvue.js**

****

Kết quả :



**Event Modifiers**

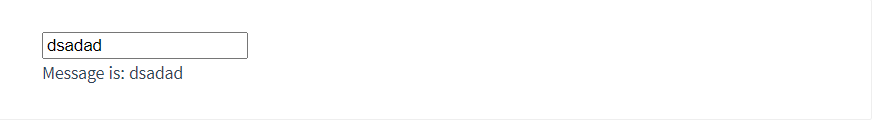
**Key Modifiers**

# **Form Input Bindings**

**Text**

<input v-model="message" placeholder="edit me">

<p>Message is: {{ message }}</p>

****

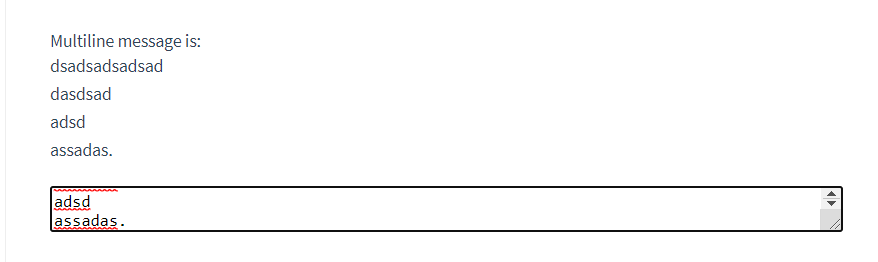
**Multiline text**

<span>Multiline message is:</span>

<p style="white-space: pre-line;">{{ message }}</p>

<br>

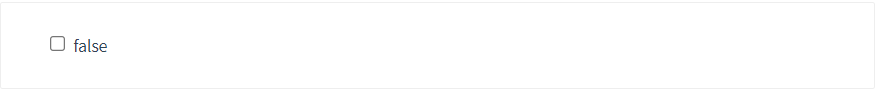
<textarea v-model="message" placeholder="add multiple lines"></textarea>

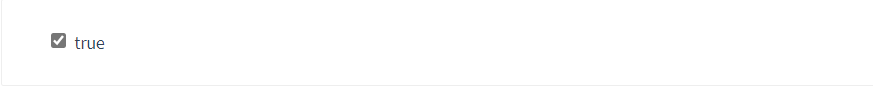
****

**Checkbox**

<input type="checkbox" id="checkbox" v-model="checked">

<label for="checkbox">{{ checked }}</label>

****

****

**Note :** v-model sẽ lấy dữ liệu từ biến nào có tên từ giá trị String

**MultiCheckbox**

<input type="checkbox" id="jack" value="Jack" v-model="checkedNames">

<label for="jack">Jack</label>

<input type="checkbox" id="john" value="John" v-model="checkedNames">

<label for="john">John</label>

<input type="checkbox" id="mike" value="Mike" v-model="checkedNames">

<label for="mike">Mike</label>

<br>

<span>Checked names: {{ checkedNames }}</span>

new Vue({

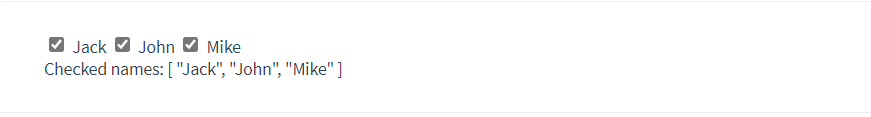
el: '...',

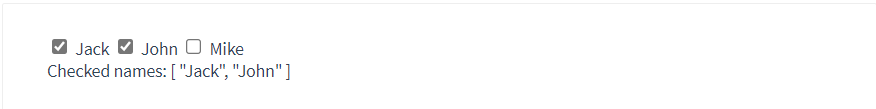
data: {

checkedNames: []

}

})

****

****

**Note :** v-model sẽ append nhiều giá trị được check vào trong biến có tên là giá trị của v-model

**Radio**

<input type="radio" id="one" value="One" v-model="picked">

<label for="one">One</label>

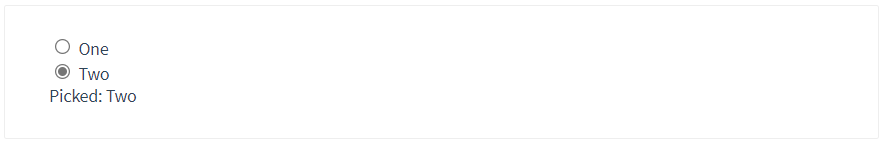
<br>

<input type="radio" id="two" value="Two" v-model="picked">

<label for="two">Two</label>

<br>

<span>Picked: {{ picked }}</span>

****

**Note :** v-model sẽ create giá trị được check vào trong biến có tên là giá trị của v-model khi type = radio

**Select**

<select v-model="selected">

<option disabled value="">Please select one</option>

<option>A</option>

<option>B</option>

<option>C</option>

</select>

<span>Selected: {{ selected }}</span>

new Vue({

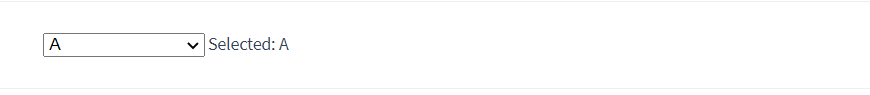
el: '...',

data: {

selected: ''

}

})



**Multiple select (bound to Array)**

<select v-model="selected" multiple>

<option>A</option>

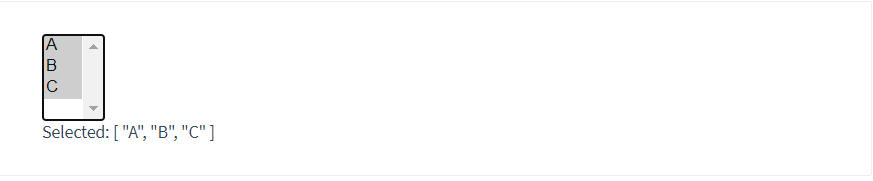
<option>B</option>

<option>C</option>

</select>

<br>

<span>Selected: {{ selected }}</span>



Hoặc

<select v-model="selected">

<option v-for="option in options" v-bind:value="option.value">

{{ option.text }}

</option>

</select>

<span>Selected: {{ selected }}</span>

new Vue({

el: '...',

data: {

selected: 'A',

options: [

{ text: 'One', value: 'A' },

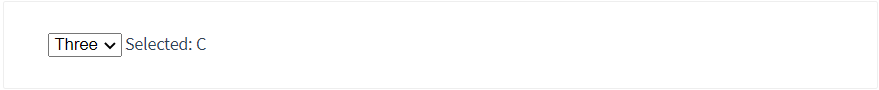
{ text: 'Two', value: 'B' },

{ text: 'Three', value: 'C' }

]

}

})

****

**Value Bindings**

**Checkbox**

<input

type="checkbox"

v-model="toggle"

true-value="yes"

false-value="no"

>

// when checked:

vm.toggle === 'yes'

// when unchecked:

vm.toggle === 'no'

**Radio**

<input type="radio" v-model="pick" v-bind:value="a">

// when checked:

vm.pick === vm.a

**Select Options**

<select v-model="selected">

<!-- inline object literal -->

<option v-bind:value="{ number: 123 }">123</option>

</select>

// when selected:

typeof vm.selected // => 'object'

vm.selected.number // => 123

# **Components Basics**

**Base Example**

Vue.component('button-counter', {

data: function () {

return {

count: 0

}

},

template: '<button v-on:click="count++">You clicked me {{ count }} times.</button>'

})

Components là một Vue Instance có thể được xài đi xài lại nhiều lần, <button-counter>. Chúng ta có thể sử dụng component như một custom element bên trong root Vue instance đã tạo ra trước đó bằng new Vue:

**index.html**

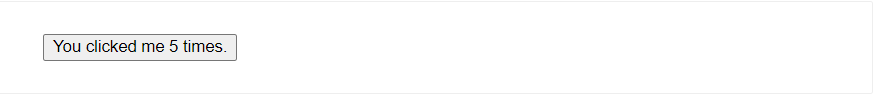
<div id="components-demo">

<button-counter></button-counter>

</div>

**customvue.js**

new Vue({ el: '#components-demo' })

****

**Reusing Components**

Components có thể được sử dụng lại bao nhiêu lần tùy thích

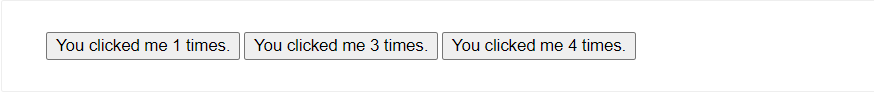
<div id="components-demo">

<button-counter></button-counter>

<button-counter></button-counter>

<button-counter></button-counter>

</div>



**Note :** Khi new Vue, thì mọi thứ bên trong nó sẽ dựa vào bản gốc mà tạo ra.

**data Must Be a Function**

data: function () {

return {

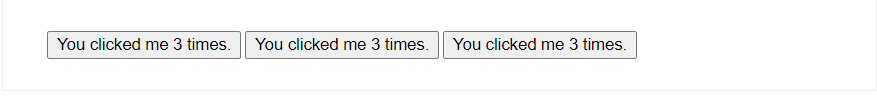
count: 0

}

}

Data phải là 1 function vì dẫu tạo mới nhưng chúng luôn có cách ánh xạ biến lúc mới khai báo nên ta phải return để trả những giá trị cũ thì lúc tạo mới ta có thể đạt được giá trị mặc định.

Nếu không nó khi update nó sẽ update 1 lúc các component giống nhau



**Organizing Components**

Có 2 scope: Global và local

Global – có thể được gọi và truy cập ở bất kì đâu, bất kì ngóc ngách trong các component mà chương trình Vue, mình đã định nghĩa.

Local – Chỉ có thể gọi được bởi thằng cha hoặc nhiều điều khác mình sẽ tìm hiểu sau

Cho tới bài này chúng ta mới chỉ tập làm quen với global

**Passing Data to Child Components with Props**

Vue.component('blog-post', {

props: ['title'],

template: '<h3>{{ title }}</h3>'

})

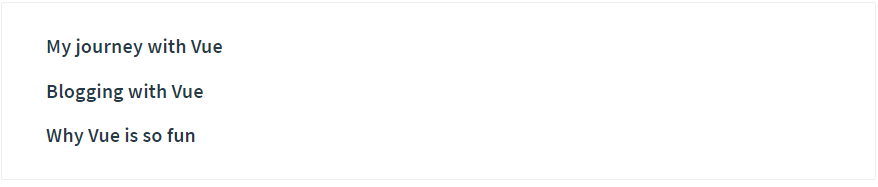
Đây cũng là một cách để khai báo biến trong Component nhưng khác với data, data chỉ có thể được gọi ở bên trong Component.

Khi mà prop được định nghĩa, có thể pass data bằng cách này

<blog-post title="My journey with Vue"></blog-post>

<blog-post title="Blogging with Vue"></blog-post>

<blog-post title="Why Vue is so fun"></blog-post>



Với trường hợp mà sử dụng array để list các item

new Vue({

el: '#blog-post-demo',

data: {

posts: [

{ id: 1, title: 'My journey with Vue' },

{ id: 2, title: 'Blogging with Vue' },

{ id: 3, title: 'Why Vue is so fun' }

]

}

})

Có thể sử dụng chung với v-for attribute

<blog-post

v-for="post in posts"

v-bind:key="post.id"

v-bind:title="post.title"

></blog-post>

**A Single Root Element**

Khi xây dựng một <blog-post> component, template không chỉ chứa mỗi title

<h3>{{ title }}</h3>

Ít nhất cũng phải chứa

<h3>{{ title }}</h3>

<div v-html="content"></div>

Nếu bạn sử dụng cách dưới đây, Vue sẽ hiện ra lỗi và giải thích rằng mọi component phải có một root element. Bạn có thể fix lỗi này nếu wrap cái template trong element cha, ví dụ:

<div class="blog-post">

<h3>{{ title }}</h3>

<div v-html="content"></div>

</div>

Trong trường hợp component gia tăng, chúng ta không chỉ cần title hay content của post mà còn cần nhiều hơn thế ( published date, comments, ... ). Khai báo prop cho mỗi thứ khiến cho việc tạo component khá vất vả

<blog-post

v-for="post in posts"

v-bind:key="post.id"

v-bind:title="post.title"

v-bind:content="post.content"

v-bind:publishedAt="post.publishedAt"

v-bind:comments="post.comments"

></blog-post>

Ta có thể xử lý bằng cách gom hết bọn chúng vào chung 1 Obj và gọi bằng 1 prop

<blog-post

v-for="post in posts"

v-bind:key="post.id"

v-bind:post="post"

></blog-post>

Vue.component('blog-post', {

props: ['post'],

template: `

<div class="blog-post">

<h3>{{ post.title }}</h3>

<div v-html="post.content"></div>

</div>

`

})

**Listening to Child Components Events**

Sẽ có nhiều lúc ta cần lắng nghe 1 event gì đó từ element của Component con nhưng Vue không cho phép giao tiếp ngược lại từ con lên cha.

VD:

Component cha có 1 biến tên là postFontSize dùng để kiểm soát kích cỡ chữ của toàn bộ blog posts:

new Vue({

el: '#blog-posts-events-demo',

data: {

posts: [/\* ... \*/],

postFontSize: 1

}

})

<div id="blog-posts-events-demo">

<div :style="{ fontSize: postFontSize + 'em' }">

<blog-post

v-for="post in posts"

v-bind:key="post.id"

v-bind:post="post"

></blog-post>

</div>

</div>

Giờ ta gọi 1 nút dùng để gia tăng kích thước của chữ bên ngoài content của cha và bên trong component blog- post

Vue.component('blog-post', {

props: ['post'],

template: `

<div class="blog-post">

<h3>{{ post.title }}</h3>

<button>

Enlarge text

</button>

<div v-html="post.content"></div>

</div>

`

})

Hiện giờ nút này chưa thể làm gì vì ta chưa set Event cho nó

<button>

Enlarge text

</button>

Vue đã cung cấp một cách để có thể hóa giải vấn đề này. Thằng cha có thể chọn event nào để lắng nghe từ thằng con với v-on, cũng cùng cách làm với DOM thường.

<blog-post

...

v-on:enlarge-text="postFontSize += 0.1"

></blog-post>

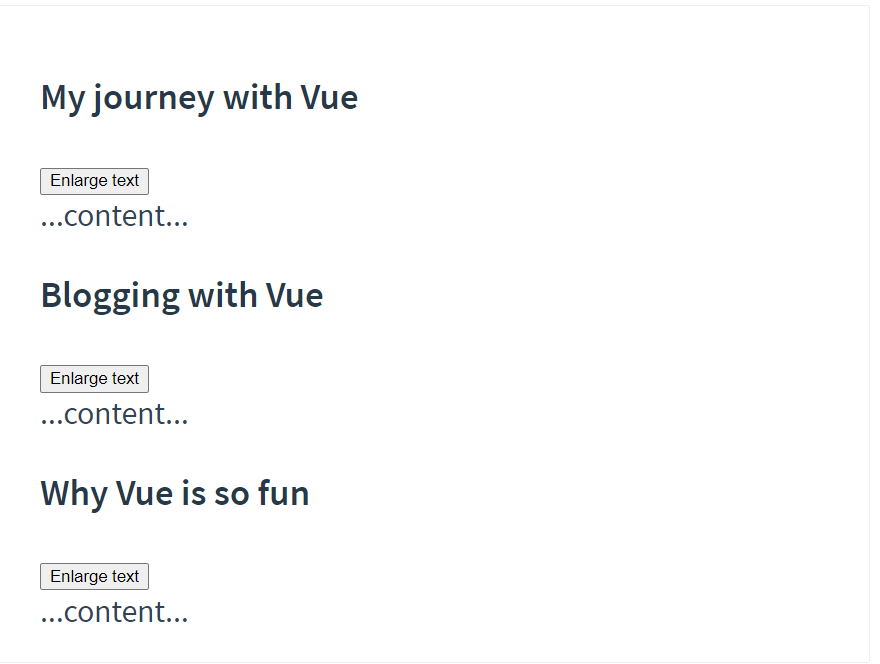
Ở component con có thể phát động Event với cách sử dụng $emit và pass tên của event đó.

<button v-on:click="$emit('enlarge-text')">

Enlarge text

</button>

Nhờ đó ta có thể update font size của thằng cha



**Emitting a Value With an Event**

Con

<button v-on:click="$emit('enlarge-text', 0.1)">

Enlarge text

</button>

Cha

<blog-post

...

v-on:enlarge-text="postFontSize += $event"

></blog-post>

Or

<blog-post

...

v-on:enlarge-text="onEnlargeText"

></blog-post>

methods: {

onEnlargeText: function (enlargeAmount) {

this.postFontSize += enlargeAmount

}

}

**Using v-model on Components**

<input v-model="searchText">

Thật ra là viết tắt của

<input

v-bind:value="searchText"

v-on:input="searchText = $event.target.value"

>

Bây giờ, ta có thể viết theo cách tạo component

<custom-input

v-bind:value="searchText"

v-on:input="searchText = $event"

></custom-input>

Vue.component('custom-input', {

props: ['value'],

template: `

<input

v-bind:value="value"

v-on:input="$emit('input', $event.target.value)"

>

`

})

Giống như trên v-model cũng có thể được thay thế vào mà vẫn phù hợp.

<custom-input v-model="searchText"></custom-input>

**Content Distribution with Slots**

Để ghi thêm vào component children ta có thể ghi

<alert-box>

Something bad happened.

</alert-box>



May thay, nhiệm vụ này Vue đã hỗ trợ ta slot element

Vue.component('alert-box', {

template: `

<div class="demo-alert-box">

<strong>Error!</strong>

<slot></slot>

</div>

`

})

**Dynamic Components**

**DOM Template Parsing Caveats**