

A large orange circle in the top-left corner of the slide.

Vue.js

Construire des Applications orientées composants

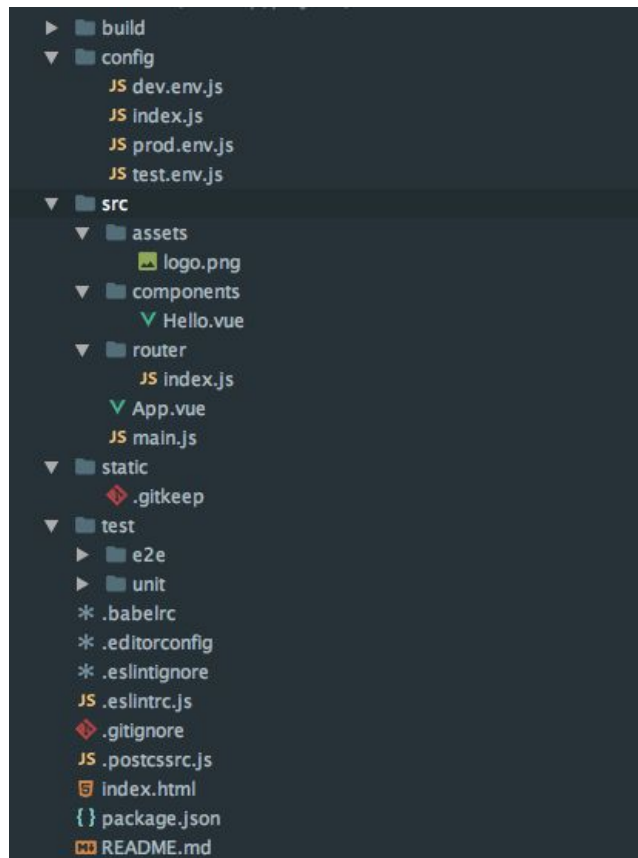




- Présentation
- Architecture
- Composants
- Data Binding
- Router

Architecture

```
$ npm install -g vue-cli  
$ vue init webpack my-project  
$ cd my-project && npm install
```



C'est quoi un composant ?

Boite noire

Expose une API

Accepte des paramètres
(des **props**)

Ressort un affichage / un
comportement

Réutilisable

Sans y toucher

On le pose, on le configure
ça marche

Autosuffisant

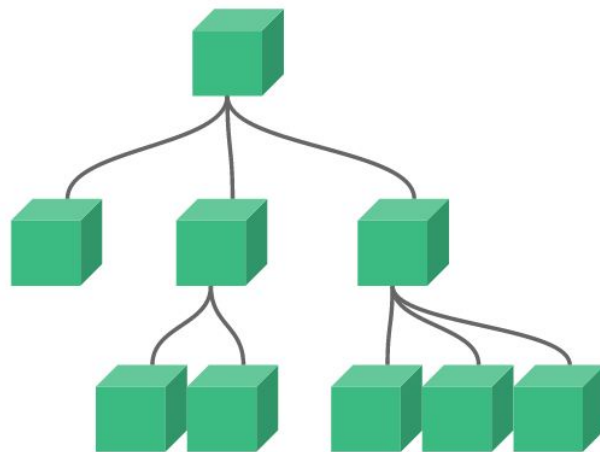
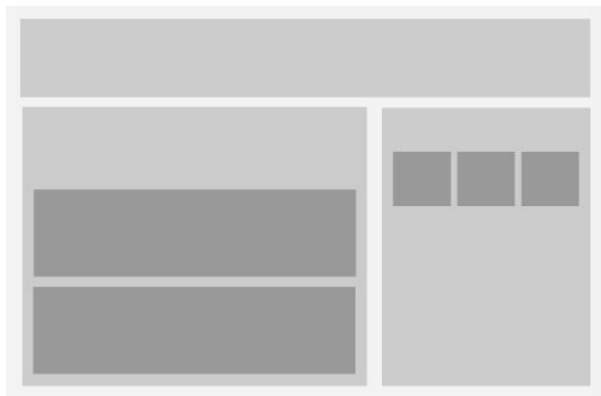
Distribuable sur NPM ou sur d'autre
projets ?

Pas de dépendances

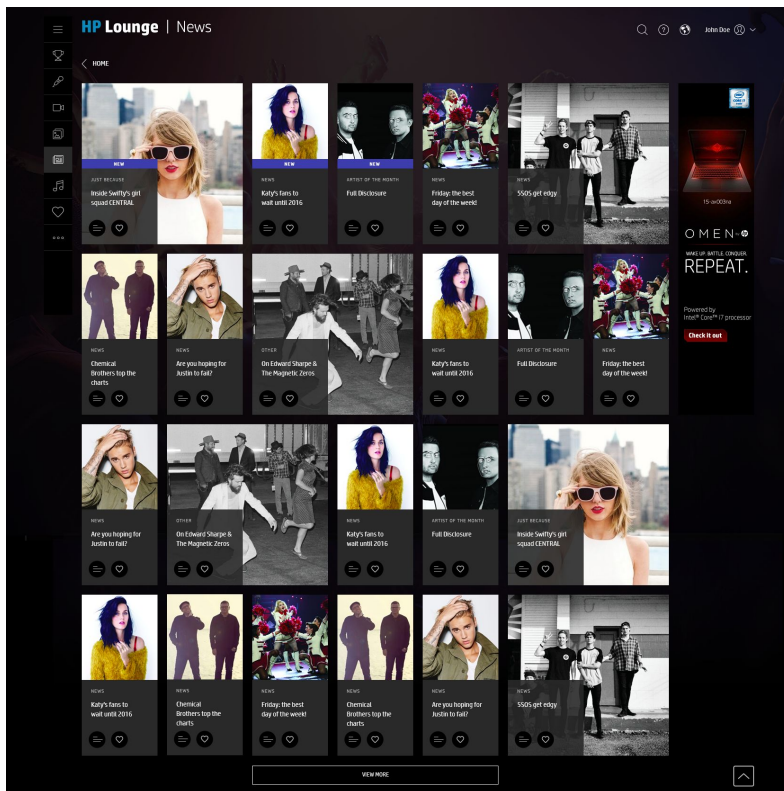


Composants

L'organisation d'une page

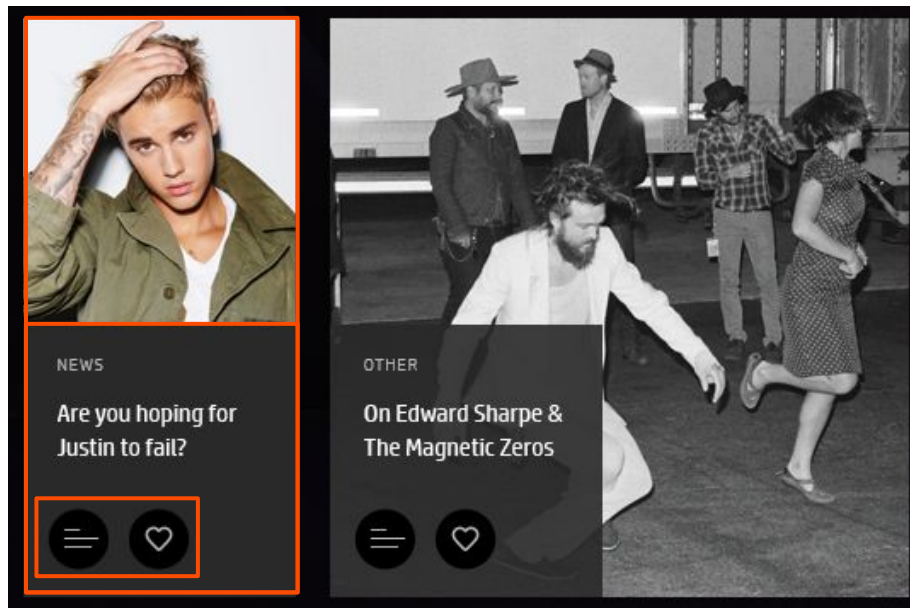


Concept



Composants

Concept



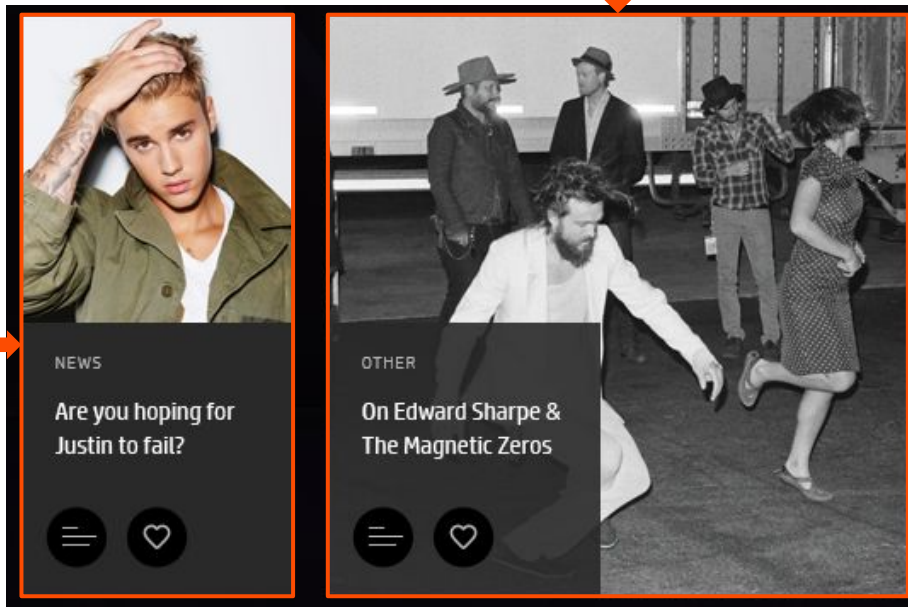
Un composant expose une API
via ses *props*



Composants

Les props

```
size = 'small'  
tag = 'news'  
title = 'Are you...'
```



Composants

Les props

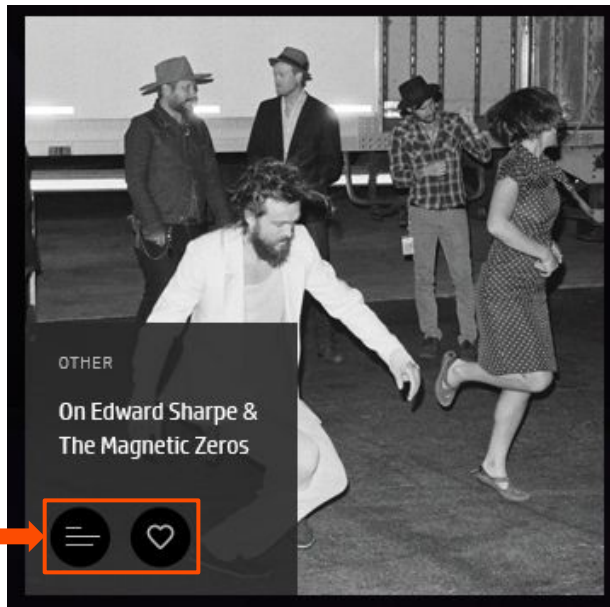
```
tag = 'other'  
title = 'On Edward...'  
detail = true  
like = true
```



Composants

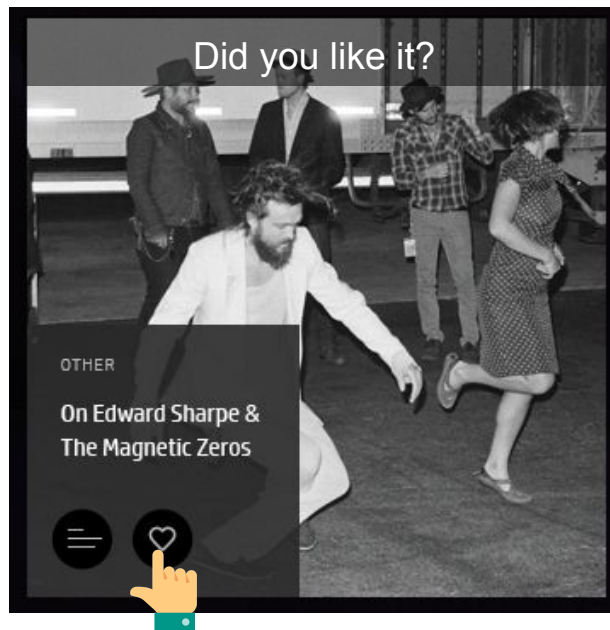
Les props

```
detail = true  
like = true
```



Composants

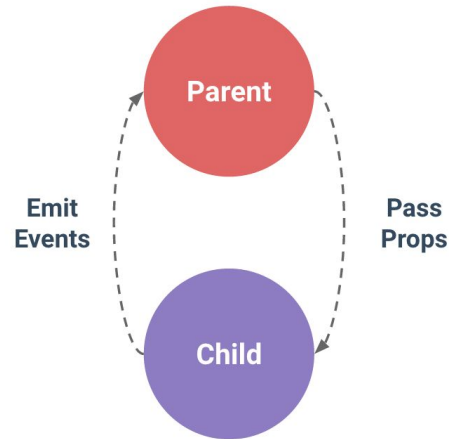
Communication Parent - Enfant



Composants

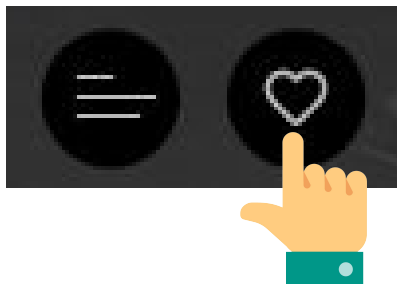
Communication Parent - Enfant

“**Props** down , **events** up”



Composants

Communication Parent - Enfant

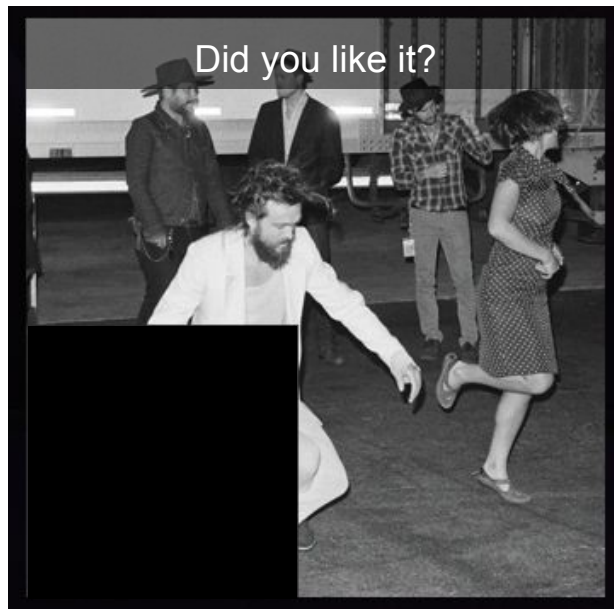


```
this.$emit('like: hover')
```



Composants

Communication Parent - Enfant



```
this.$on('like::hover', () => {  
  this.displayDescription = true  
})
```



Composants

Single file

Un composant = Un fichier




```
<template>
  <span>
    <dropdown-test :id="collection" :class="class">
      <li><a>Edit collection</a></li>
      <li class="divider"></li>
      <li v-if="!isRealtime"><a>Browse documents</a></li>
      <li><a>Watch messages</a></li>
    </dropdown-test>
  </span>
</template>
```

```
<style lang="scss" rel="stylesheet/scss" scoped>
  li {
    padding-top: 10px;
  }
</style>
```

```
<script>
  import DropdownTest from '../..../Materialize/DropdownTest'

  export default {
    name: 'CollectionDropdown',
    props: {
      ...
    },
    components: {
      DropdownTest
    }
  }
</script>
```



```
<template>
  <span>
    <dropdown-test :id="collection" :class="class">
      <li><a>Edit collection</a></li>
      <li class="divider"></li>
      <li v-if="!isRealtime"><a>Browse documents</a></li>
      <li><a>Watch messages</a></li>
    </dropdown-test>
  </span>
</template>
```



Un seul élément à la racine de <template>

```
<template>
  <span>
    <dropdown-test :id="collection" :class="class">
      <li><a>Edit collection</a></li>
      <li class="divider"></li>
      <li v-if="!isRealtime"><a>Browse documents</a></li>
      <li><a>Watch messages</a></li>
    </dropdown-test>
  </span>
</template>
```



Inclusion d'autres composants



```
<template>
  <span>
    <dropdown-test :id="collection" :class="class">
      <li><a>Edit collection</a></li>
      <li class="divider"></li>
      <li v-if="!isRealtime"><a>Browse documents</a></li>
      <li><a>Watch messages</a></li>
    </dropdown-test>
  </span>
</template>
```



```
<script>
import DropdownTest from '../..//Materialize/DropdownTest'


export default {
  name: 'CollectionDropdown',
  props: {
    ...
  },
  components: {
    DropdownTest
  }
}
</script>
```




Import des composants Vue dont on a besoin


```
<script>  
import DropdownTest from '../..../Materialize/DropdownTest'  
  
export default {  
  name: 'CollectionDropdown',  
  props: {  
    ...  
  },  
  components: {  
    DropdownTest  
  }  
}  
</script>
```



```
<template>
   <span>
    <dropdown-test :id="collection" :class="class">
      <li><a>Edit collection</a></li>
      <li class="divider"></li>
      <li v-if="!isRealtime"><a>Browse documents</a></li>
      <li><a>Watch messages</a></li>
    </dropdown-test>
  </span>
</template>

<style lang="scss" rel="stylesheet/scss" scoped>
  li {
    padding-top: 10px;
  }
</style>

 <script>
  import DropdownTest from '../Materialize/DropdownTest'

  export default {
    name: 'CollectionDropdown',
    props: ['class'],
    components: {
      DropdownTest
    }
  }
</script>

```



Composants

Data / Methods / Computed / Watch

VM = Vue - Model



Composants

Data / Methods / Computed / Watch

```
<script>
export default {
  name: 'CollectionDropdown',
  props: [...],
  data () {
    return {
      ...
    }
  },
  computed: {
    ...
  }
  methods: {
    ...
  },
  watch: {
    ...
  }
}
</script>
```



Composants

Data

```
data () {  
  return {  
    items: [  
      {  
        tag: 'news',  
        title: 'Are you...'  
      }  
    ]  
  }  
}
```

```
<card  
  v-for="item in items"  
  :category="item.tag"  
  :title="item.title">  
</card>
```

Contient le **modèle** qui va être
exposé à la **vue**



Composants

Data

```
data () {  
  return {  
    items: [  
      {  
        tag: 'news',  
        title: 'Are you...'  
      }  
    ]  
  }  
}
```

```
<card  
  v-for="item in items"  
  :category="item.tag"  
  :title="item.title">  
</card>
```

Ici, c'est une **props**.

Et on évite de passer
tout un objet à l'enfant.

Contient le **modèle** qui va être
exposé à la **vue**



Composants

Methods

```
methods: {  
  like (index) {  
    console.log('like', index)  
  }  
}
```

```
<card  
  v-for="(item, index) in items"  
  :category="item.tag"  
  :title="item.title"  
  @click="like(index)">  
</card>
```

Contient les **fonctions** qui vont être
exposé à la **vue**



Composants

Watch

```
watch: {  
  items () {  
    console.log('items changed')  
  }  
}
```

Exécute une **fonction**
quand une **donnée** est **changée**



Composants

Computed

```
computed: {  
  fullName () {  
    return this.firstname + ' ' + this.lastname  
  }  
}
```

Permet **d'exposer** une variable
à partir **d'autres** variables



Computed vs Methods vs Watch

Computed

Mise en cache

Disponible comme le modèle
dans la vue

Déclaratif

Methods

Sert à déclencher une action

Exécuté à chaque appel

Watch

Non déclaratif

Exécuté à chaque appel



Composants

Computed

```
computed: {  
  fullName () {  
    return this.firstName + ' ' + Date.now()  
  }  
}
```

```
console.log(this.fullName) // "toto 1476203531119"  
console.log(this.fullName) // "toto 1476203531119"  
  
this.firstName = 'tutu'  
  
console.log(this.fullName) // "tutu 1476407738018"
```



Composants

Computed vs Method

NE PAS FAIRE

```
<user  
  v-for="user in users"  
  :full-name="getFullName(user)"  
>  
</user>
```



Composants

Computed vs Method

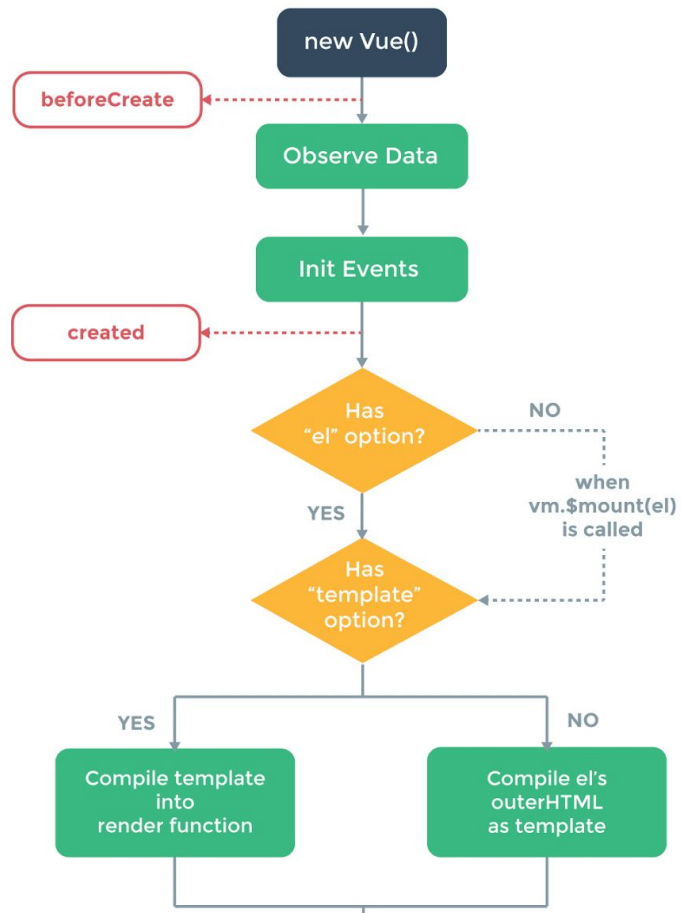
A FAIRE

```
<user  
  v-for="user in modifiedUsers" ← computed  
  :full-name="user.fullName"  
>  
</user>
```



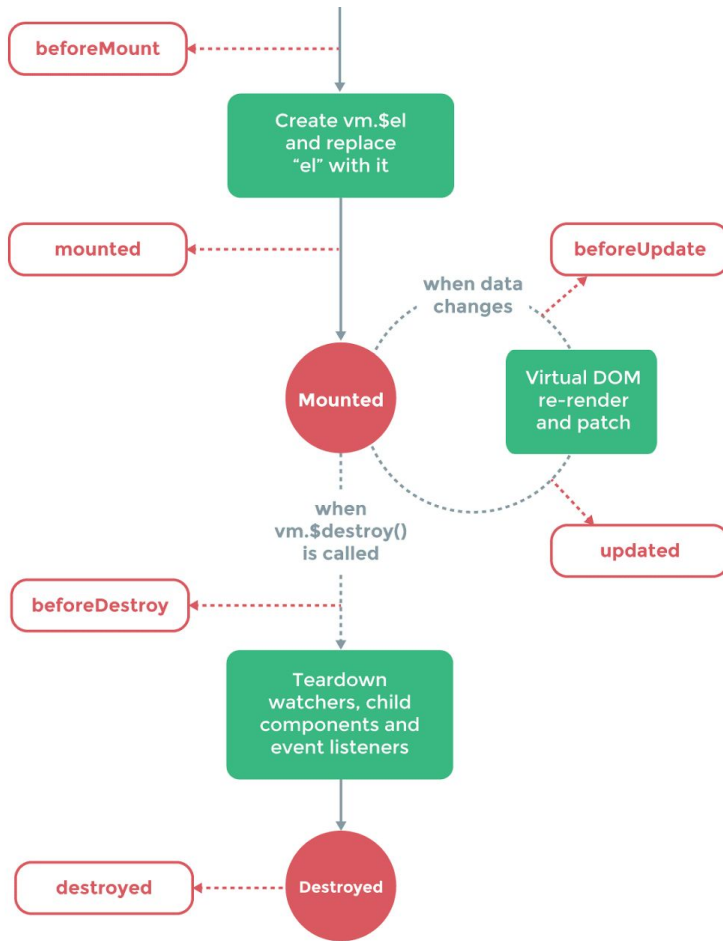
Composants

Lifecycle



Composants

Lifecycle



Composants

Lifecycle

```
export default {  
  name: 'UsersList',  
  data () {  
    return {  
      users: [],  
      loader: true  
    }  
  },  
  mounted () {  
    getUsers()  
      .then((result) => {  
        this.users = result.users  
        this.loader = false  
      })  
  }  
}
```

Attention, le composant **n'attend pas le traitement** pour être *mounted*.

Si il y a du traitement asynchrone, le composant sera **affiché avant que la donnée soit présente**.

Pratique si on veut afficher un loader, plutôt qu'une page blanche.



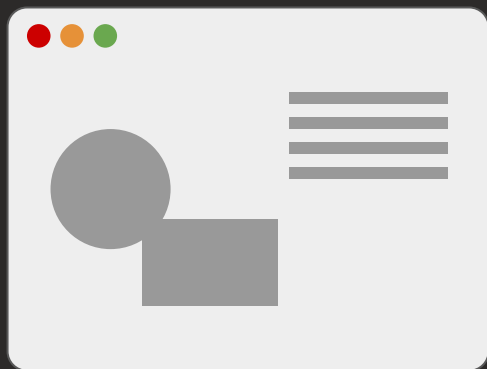
Data Binding - **getter / setter**



Le Data Binding

WAT IZ DIS

Synchro entre la vue et la donnée



```
data () {  
  return {  
    items: [  
      {  
        tag: 'news',  
        title: 'Are you...'  
      }  
    ]  
  }  
}
```



Le Data Binding, vu dans VueJS

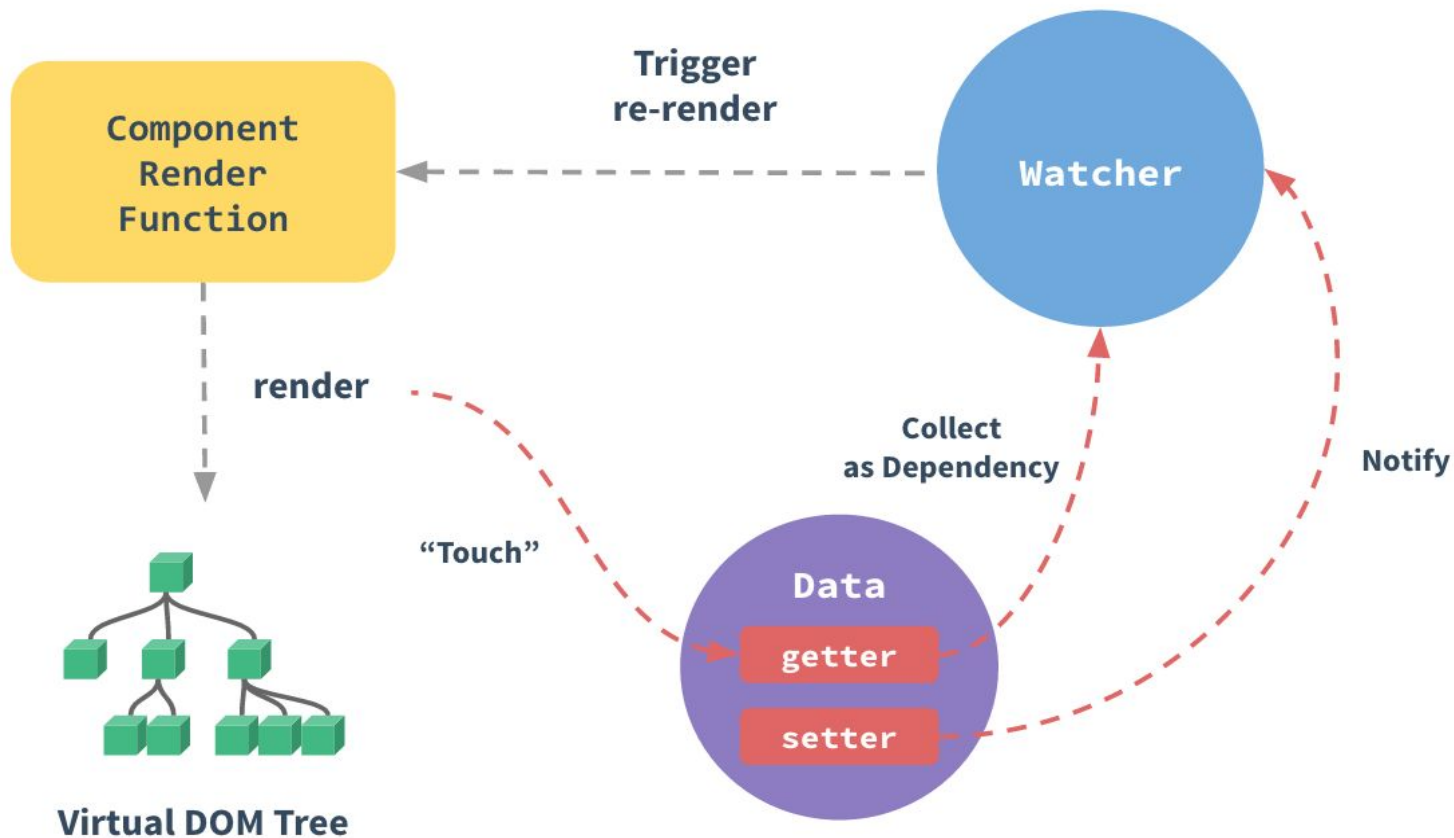
C'est bon les jeux de mots!

```
< > Notification.vue x
1 <template>
2   <div class="collapsible-header unselectable" :class="notification.class" @click
3     <i :class="{ 'fa-caret-right': collapsed, 'fa-caret-down': !collapsed}" class
4     <i class="fa fa-{{notification.icon}}"></i> {{notification.text}} - {{ago}}
5   </div>
6   <div class="collapsible-body" v-if="notification.source">
7     <p v-json-formatter="notification.source"></p>
8   </div>
9 </template>
10
11 <script>
12   import JsonFormatter from '../..../directives/json-formatter.directive'
13   var moment = require('moment')
14
15   export default {
16     name: 'RealtimeNotification',
17     props: [
18       'notification'
19     ],
20     directives: {
21       JsonFormatter
22     },
23     data () {
24       return {
25         ago: moment(this.notification.timestamp).fromNow(),
26         collapsed: true
27       }
28     },
29     methods: {
30       toggleCollapse () {
31         this.collapsed = !this.collapsed
32       }
33     }
34   }
```

```
graph LR
    subgraph Template
        direction TB
        T1["{{notification.icon}}"]
        T2["{{notification.text}}"]
        T3["{{ago}}"]
    end
    subgraph Script
        direction TB
        S1["notification (prop)"]
        S2["this.notification.timestamp (data)"]
    end
    S1 --> T1
    S1 --> T2
    S1 --> T3
    S2 --> T3
```



Comment ça marche



Comment ça marche

```
data () {  
  return {  
    user: {  
      name: 'Bob'  
    }  
  }  
}
```



Comment ça marche

```
let name = 'Bob'
let user = {name}

Object.defineProperty(user, 'name', {
  set(newName) {
    console.log('TRIGGER PATCH VDOM')
    name = newName
  },
  get() {
    return name
  }
})
```

Sur **tous** les attributs que `data ()` va retourner.



Comment ça marche

```
let name = 'Bob'
let user = {name}

Object.defineProperty(user, 'name', {
  set(newName) {
    console.log('TRIGGER PATCH VDOM')
    name = newName
  },
  get() {
    return name
  }
})
```



Voyons voir

Keskispasse si...

```
var data = {user: {name: 'Bob'}}  
var vm = new Vue({  
  data  
})  
// vm.user and vm.user.name are now reactive
```



Voyons voir

Keskispasse si...

```
var data = {user: {name: 'Bob'}}
var vm = new Vue({
  data
})

vm.user.username = 'Boby'
// vm.user.username is NOT reactive
```



Faut faire comme ça...

eh oui

```
Vue.set(this.user, 'username', 'Boby')
```



... et toujours déclarer les attributs à l'initialisation

eh oui

```
var vm = new Vue({  
  data: {  
    name: 'Bob',  
    username: null  
  }  
})
```



Les tableaux

Faut y faire gaffe aussi

- `users.push(value)` // t'es content
- `users.pop()` // aussi
- `users[7] = value` // t'es triste



l'index n'existait pas



Les tableaux

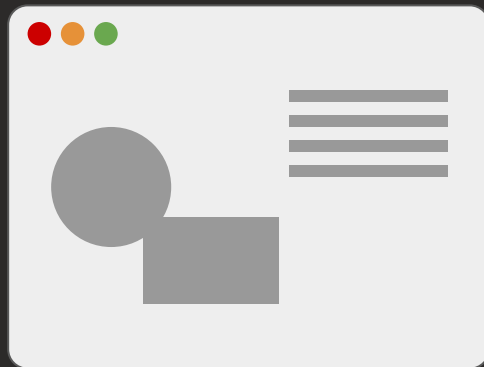
Faut y faire gaffe aussi

```
Vue.set(this.users, 7, value)
```



Le Vue Router

Tu vas pas y couper!



URL



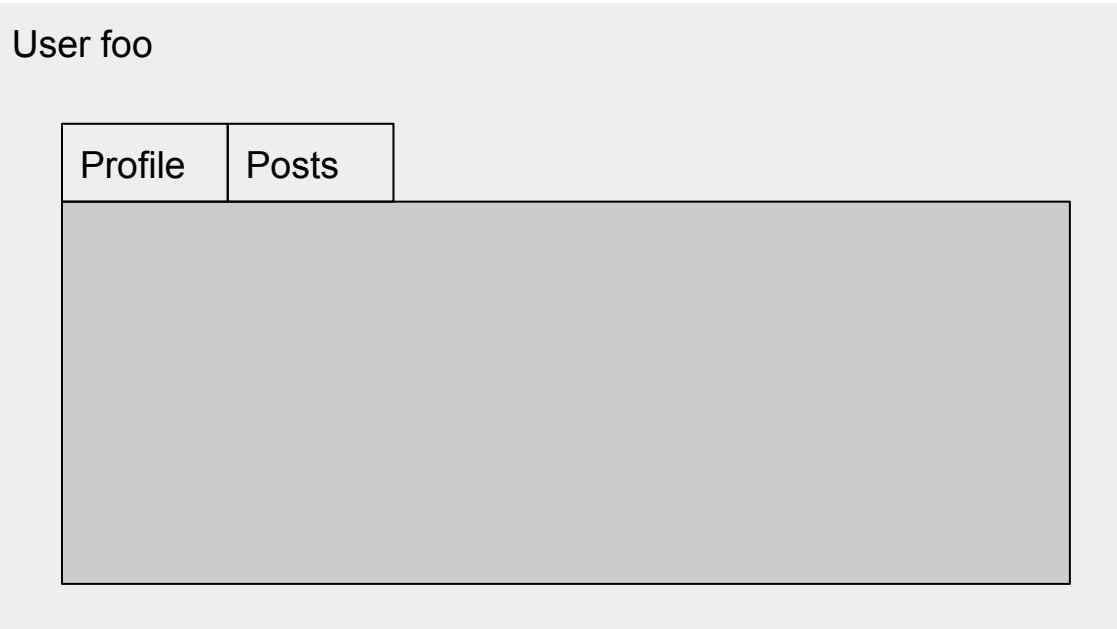
```
<div id="app">  
  <router-view></router-view>  
</div>
```

```
const router = new Router({  
  routes: [  
    {  
      path: '/user/:id',  
      name: User,  
      component: User  
    }  
  ]  
})
```



View router

Imbrication de `<view-router>`



View router

Imbrication de <view-router>

/user/foo/profile

+-----+

| User |

| +-----+ |

| | Profile | |

| | | |

| +-----+ |

+-----+

/user/foo/posts

+-----+

| User |

| +-----+ |

| | Posts | |

| | | |

| +-----+ |

+-----+

+----->



View router

Imbrication de <view-router>

```
<div class="user">  
  <h2>User {{ $route.params.id }}</h2>  
  <router-view></router-view>  
</div>
```



Chacun son chemin

Chacun sa route

La hiérarchie des routes
reflète la **hiérarchie des
composants**, pas celle
des chemins

```
/user/foo/profile
```

```
+-----+  
| User   |  
| +-----+ |  
| | Profile | |  
| |       | |  
| +-----+ |  
+-----+
```



Ah, et aussi...

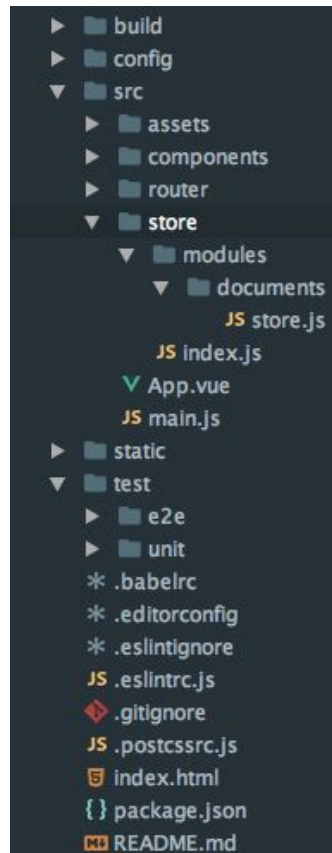


L'état de l'application - le **store**



Architecture Vuex

```
$ npm install vuex --save
```

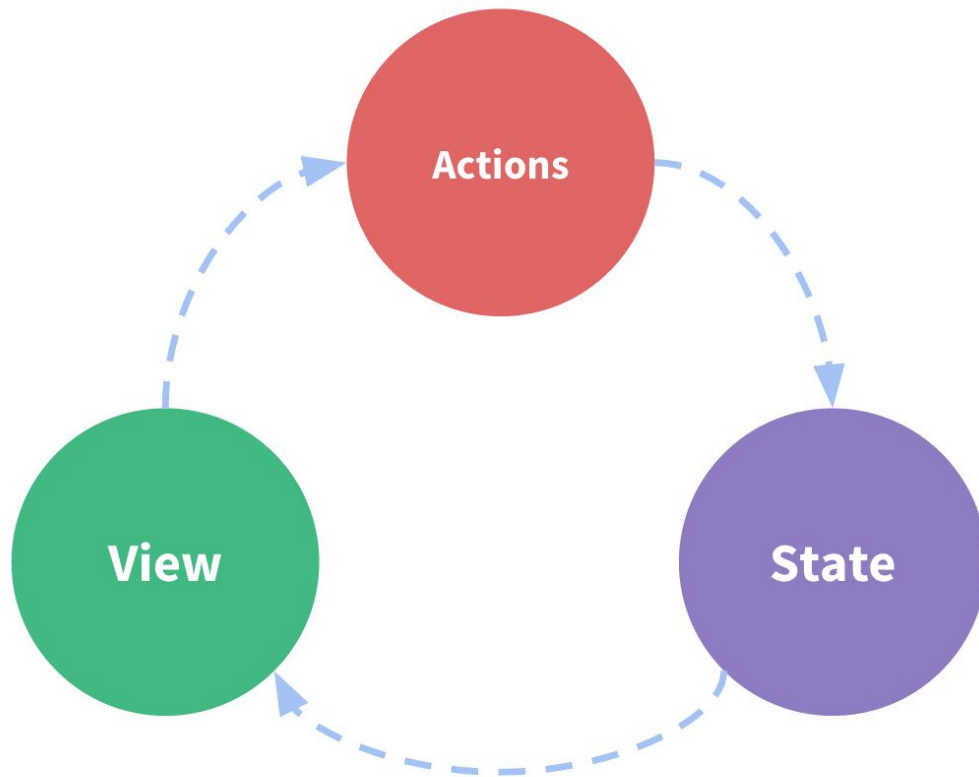
A file explorer view of a project structure. The tree shows a 'src' directory containing 'assets', 'components', 'router', and 'store'. The 'store' directory is expanded, showing 'modules' (which contains 'documents' with 'store.js') and 'index.js'. Other files in 'src' include 'App.vue' and 'main.js'. Below 'src' are 'static' and 'test' (containing 'e2e' and 'unit'). At the bottom are various configuration files like '.babelrc', '.editorconfig', '.eslintignore', '.eslintrc.js', '.gitignore', '.postcssrc.js', 'index.html', 'package.json', and 'README.md'.

```
build
config
src
├── assets
├── components
├── router
├── store
│   ├── modules
│   │   └── documents
│   │       └── store.js
│   └── index.js
├── App.vue
└── main.js
static
test
├── e2e
└── unit
.babelrc
.editorconfig
.eslintignore
.eslintrc.js
.gitignore
.postcssrc.js
index.html
package.json
README.md
```



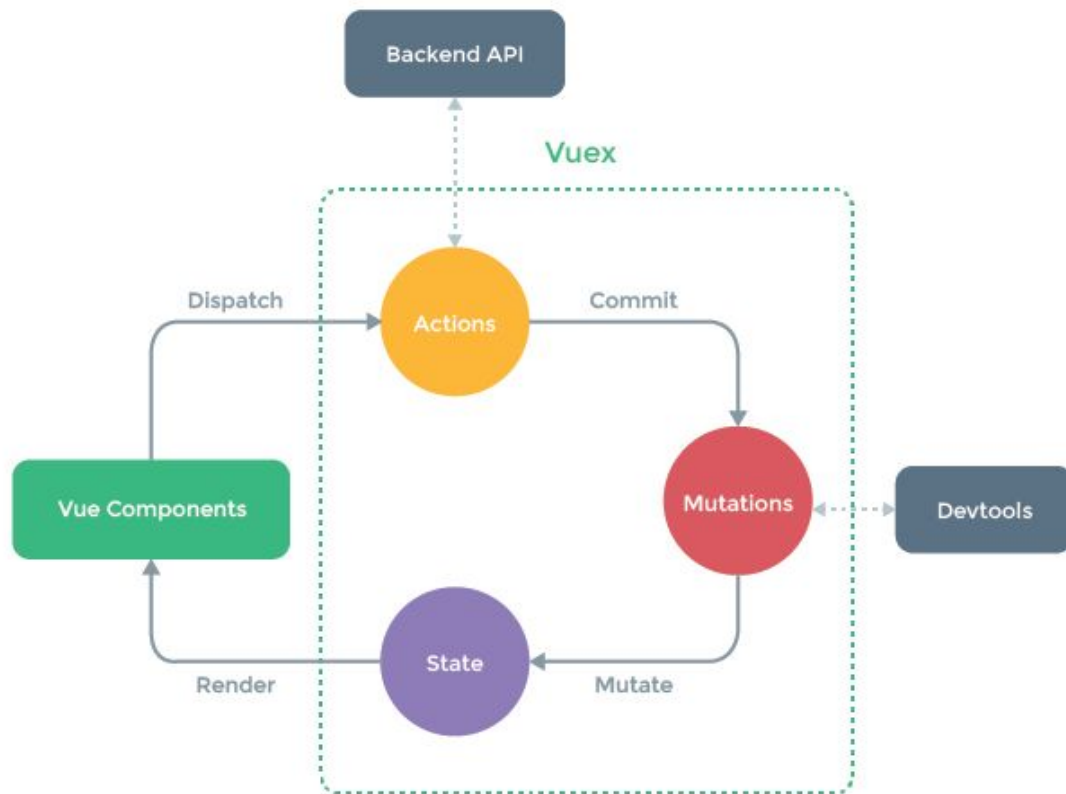
Vuex

One way data flow



Vuex

Est ton ami



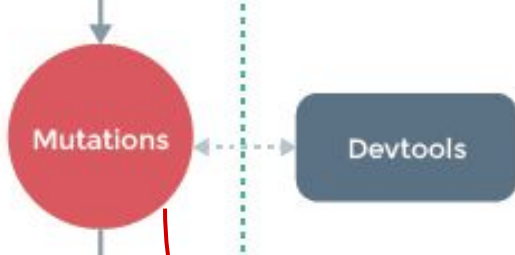


State

```
const store = new Vuex.Store({  
  state: {  
    count: 0  
  },  
  mutations: {  
    increment (state) {  
      state.count++  
    }  
  }  
})
```

Objet JavaScript classique





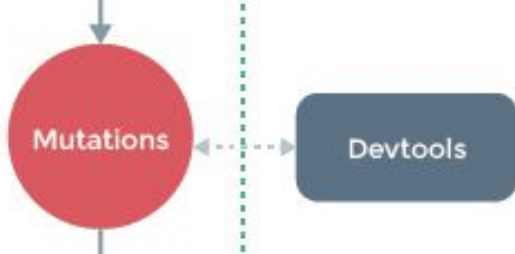
```
const store = new Vuex.Store({  
  state: {  
    count: 0  
  },  
  mutations: {  
    increment (state) {  
      state.count++  
    }  
  }  
})
```

Fonctions pures

Synchrones

Pas accessibles directement





```
store.commit('increment')
```

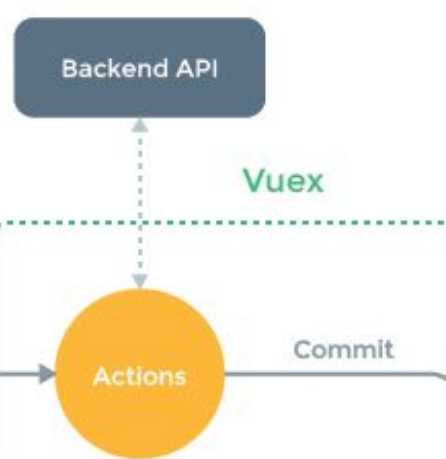
```
console.log(store.state.count) // -> 1
```

Update du store

Tracking des mutations via Devtools

Time-travelling





```
actions: {  
  checkout ([ commit, state ] payload) {  
    // save the items currently in the cart  
    const savedCartItems = [...state.cart.added]  
    // send out checkout request, and optimistically  
    // clear the cart  
    commit(types.CHECKOUT_REQUEST)  
    // the shop API accepts a success callback and a failure callback  
    shop.buyProducts(  
      products,  
      // handle success  
      () => commit(types.CHECKOUT_SUCCESS),  
      // handle failure  
      () => commit(types.CHECKOUT_FAILURE, savedCartItems)  
    )  
  }  
}
```

Le store

Mutations in
asynchronous
callbacks!



On “commit” une mutation

(Toujours pur, synchrone, sans effet de bord, modifie le state)

On “dispatch” une action

(peut être asynchrone, avoir des effets de bord, ne modifie PAS le state)



Single Source of Truth

A utiliser quand plusieurs composants ont besoin de la même donnée.

Data is Read-Only

Il informe le store qu'il voudrait le muter. C'est le store qui est en charge de la mutation.

Mutations Are Synchronous

Le state n'est pas dépendants d'évènements imprédictible. Quand une mutation est déclenché, le state est modifié.



Quand faut-il l'utiliser?

Vuex vs. Event Aggregator

Cross-component
communication



Keeping things declarative (rather
than imperative)



Sending "commands" to other
components

