FORMATION VUE.JS

elorri FORMATION

ANTHONY DI PERSIO

DOCUMENTATION VUE.JS

- Les **documentations officielles** pour **vue.js** :
  - La documentation de Vue.js (Français disponible)
  - La documentation de l'interface CLI (Anglais)
  - La documentation de vue-router (Français disponible)
  - La documentation de Vuex (Français disponible)
  - La documentation de Vue Test Utils (Anglais)

DOCUMENTATION VUE.JS

- Les **documentations** complémentaires :
  - Node Package Manager (npm) (Anglais)
  - Vanilla JS (Français disponible)
  - JavaScript moderne avec ES2015/16 (Anglais)
  - What is Babel (Anglais)
  - Webpack module bundler (Anglais)

TABLE DES MATIÈRES

« *EN L'ÉTAT, APPRENDRE REACT EST ACCABLANT! BASIQUEMENT, JE VEUX DIRE QUE REACT EST DUR! J'APPRENDS VUE.JS MAINTENANT PARCE QU'IL SEMBLE BIEN PLUS FACILE !* »

TAYLOR OTWELL - CRÉATEUR DU FRAMEWORK LARAVEL (PHP) - 2014

01

DECOUVERTE DE VUE.JS

Première approche de l'environnement Vue.js

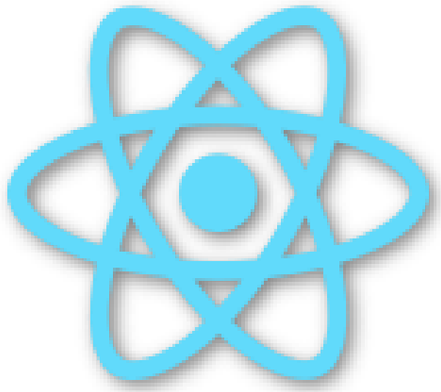
DECOUVERTE DE VUE.JS

- Vue.js est un framework open-source JavaScript
- Naissance en 2013
- Développé par Evan You
- Le framework s'oriente très nettement sur la création d'interfaces utilisateurs

DECOUVERTE DE VUE.JS

- Il s'impose aujourd'hui comme un des Frameworks Front de référence avec React (Facebook) et Angular (Google)

- Utilisé par Netflix, Adobe, Alibaba et Gitlab



DECOUVERTE DE VUE.JS

Qui est **Evan You** ?

- D'origine **Chinoise**
- Ancien de chez **Google**
- A commencer à développer **Vue.js** pour lui en **2013**

DECOUVERTE DE VUE.JS

Les **dates clés** du développement de **Vue.js** par **Evan You:**

- Naissance en **2013** alors qu'il travaillait chez google
- En **2014** Evan You est Développeur chez **Méteor**, **Taylor Otwell** (Laravel PHP) fait un Tweet qui mettra le feu aux poudres
- Vacances d'été en 2015 il travaille seul afin de pouvoir sortir la **version 1.0 de Vue.js** (avec une documentation complète en Anglais) qui sortira en **Octobre 2015**
- Février **2016 Da Feng** (CTO de Strikingly) investi dans le projet **Open-source** et permet à **Evan You** de travailler à temps plein sur Vue.js

DECOUVERTE DE VUE.JS

Les différentes publications **majeures** de **Vue.js**:

Version	Release date	Nom	Caractéristique Principale
V0.1	Juin 2013	Vue.js	Initial Commit
V0.6	Décembre 2013	Vue.js	Première publication du Framework
V1.0	Octobre 2015	Evangelion	Documentation en Anglais
V2.0	Septembre 2016	Ghost in the Shell	Apparition du « Virtual DOM »
V3.0	Septembre 2020	One Piece	Réécriture des composants internes

DECOUVERTE DE VUE.JS

Les principales **caractéristiques** de **Vue.js :**

- Permet une **programmation réactive** des applications web afin **d'optimiser** la gestion des **événements**, d'optimiser le **temp de réponse** (faible latence) et de pouvoir supporter des **flux** de données **importants**.

- Orienté **IHM**, il permet la réalisation d'application Web « **single page application** » (SPA) grâce au **Virtual DOM** qui offre la possibilité de ne **rafraichir** que les **composants nécessaires** d'une **page Web**

- Framework **progressif**: On peut l'employer très facilement et **monter** en **compétence** avec **l'augmentation** de la **difficulté** des projets

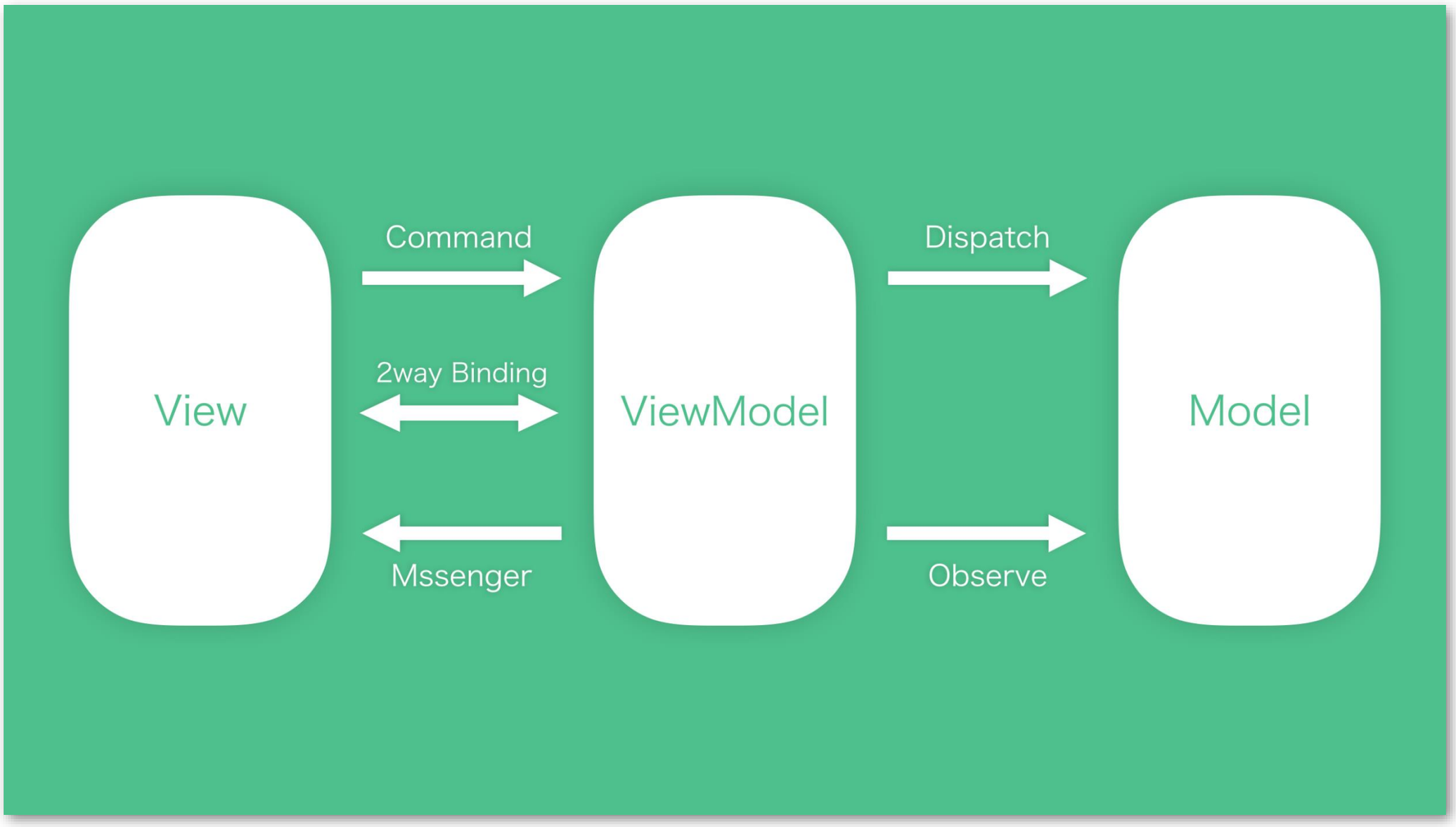
DECOUVERTE DE VUE.JS

Les **particularités** d'une **programmation réactive:**

- **Responsivité**:
  - Réponse en temps voulu avec des **temps** de **réponses rapides** et **fiables** (limites hautes, gestion asynchrone…)
- **Résilient**:
  - **Résiste** à **l'échec** de sorte qu'une erreur **n'impacte** qu'**un** seul **composant**
- **Elastique**:
  - **Pas** de **point central**, **pas** de **goulot**, **distribution** des **entrées** entre les **composants grâce aux « ViewModel »**

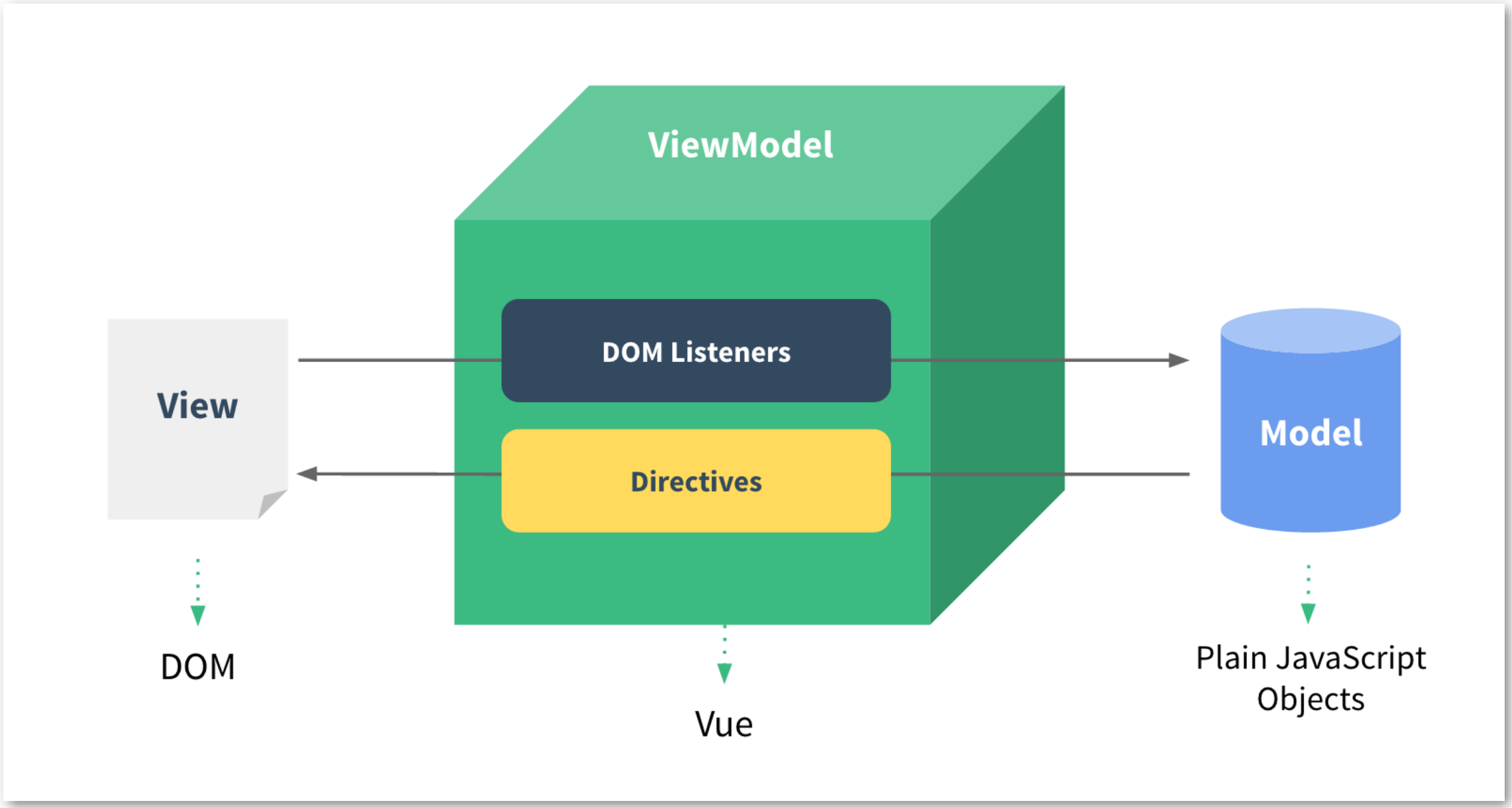
DECOUVERTE DE VUE.JS

Qu'est-ce que le **MVVM** (Model-View-ViewModel):



DECOUVERTE DE VUE.JS

Le **MVVM** appliqué avec **Vue.js:**



DECOUVERTE DE VUE.JS

Le **data binding** dans **vue.js:**

- **One way data flow**:
  - Le flux de données est unidirectionnel, les enfants ne modifient pas les données qui sont contrôlées par leur parents.
  - Les enfants peuvent demander au parent de changer
- **Two way data binding**:
  - Le flux de données est bidirectionnel, les enfants peuvent modifier les données qui sont injectées par leur parents.

DECOUVERTE DE VUE.JS

Les **deux possibilités** de gérer les **données** dans **vue.js**:

- **Données** qui changent: **mutable**
  - On emploi un **état** de ces données (**data**)
- **Données** qui ne changent pas : **immutable**
  - On utilise des **propriétés** (**props**)

La **bonne pratique** tend à **minimiser** les **données** qui **changent**, quitte à refaire des calculs.

La **bibliothèque** (Library) **Vuex** permet une **gestion avancée** des **états**

DECOUVERTE DE VUE.JS

Qu'est-ce que le **DOM** (**D**ocument **O**bject **M**odel) en **HTML**?

- **C'est le squelette d'un document**
  - Structure **arborescente** rassemblant **l'ensemble** des **éléments HTML** d'une page

- **Il définit les liaisons entres les éléments**
  - Sur notre exemple (page suivante), nous pouvons voir un **élément parent** (HTML) contenant deux **enfants** (head) et (body) contenant eux même des enfants…Etc.

DECOUVERTE DE VUE.JS

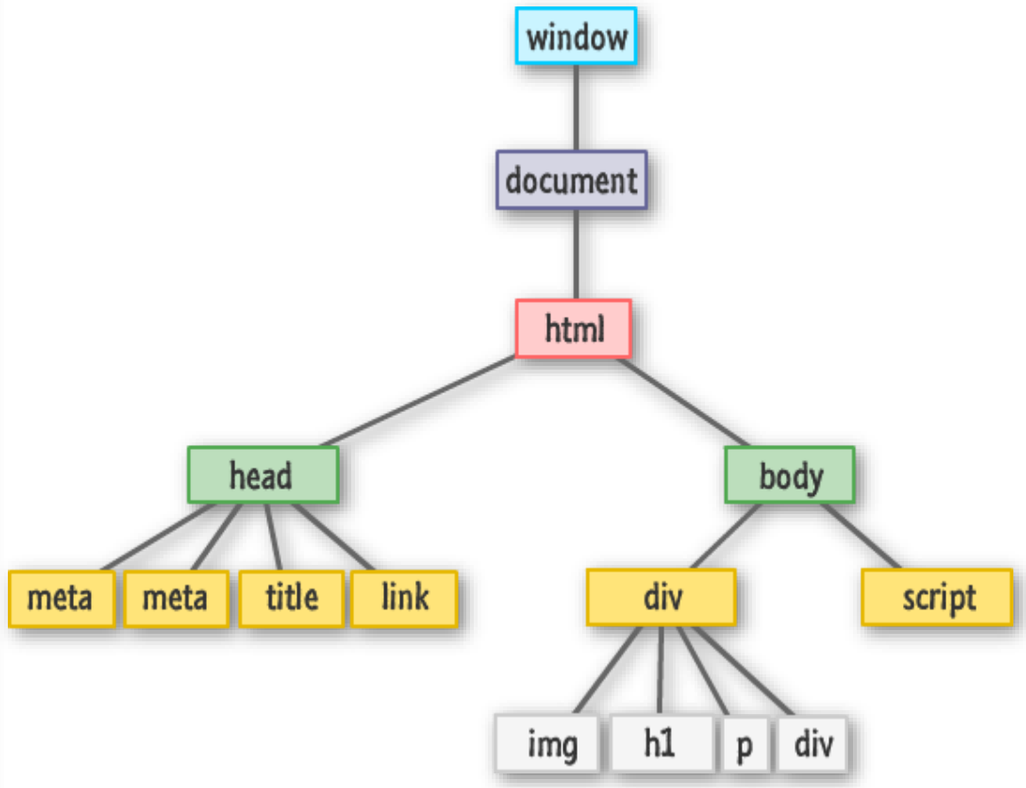
Exemple de DOM en HTML

```html
<!DOCTYPE html>
<html lang="fr">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="style.css">
</head>

<body>
    <div>
        <img src="/img/monImage.jpg" alt="Mon Image">
        <h1>Mon Titre</h1>
        <p>
            Lorem ipsum dolor sit amet, consectetur adipiscing elit.
        </p>
        <div>Voilà !</div>
    </div>
    <script src="./script.js"></script>
</body>

</html>
```



DECOUVERTE DE VUE.JS

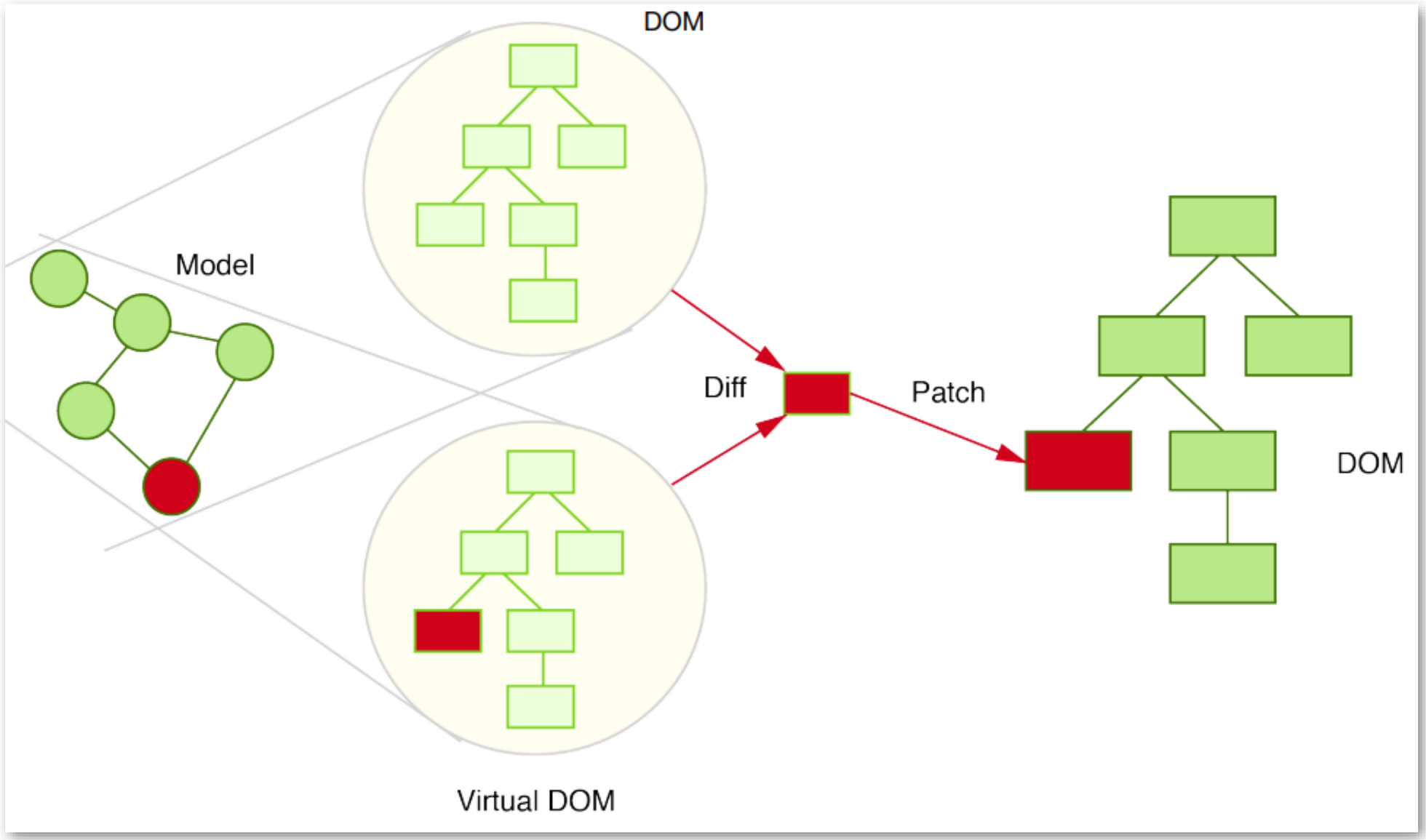
Qu’est-ce que le **Virtual DOM** dans Vue.js?

- Le **Virtual DOM** (VDOM) est un **concept** de **programmation** dans lequel une **représentation idéale**, ou « virtuelle », d'une **interface utilisateur** (UI) est conservée en mémoire

- Elle est **synchronisée** avec le **DOM « réel »** par une **bibliothèque** qui s’occupe **d’actualiser** uniquement les **changements**

DECOUVERTE DE VUE.JS

Le Virtual **DOM** de Vue.js



02

INTÉGRATION DE VUE.JS

Les étapes pour l’installation de vues.js dans votre projet

INTÉGRATION DE VUE.JS

- **Intégration** de **Vue.js** dans un projet:

  - Nous allons travailler avec la **dernière** version **stable**: **V2.5.16** car la version 3.0 est encore en version Beta (Mars 2021)

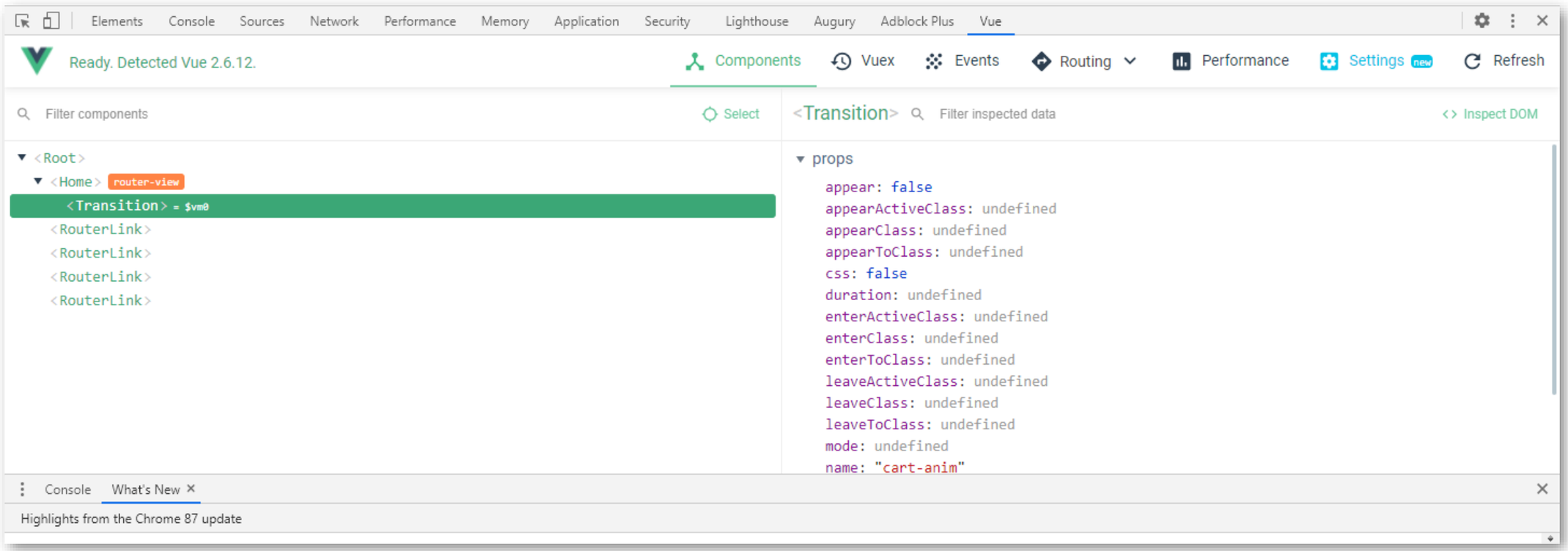
- **Compatibilité** avec les web **browser**:

  - Vue.js **supporte** tous les **navigateurs** compatibles **ECMAScript** (Chrome, Firefox, Opéra…) mais ne **supporte pas IE8** (et versions antérieures) car il utilise des fonctionnalités ECMAScript 5 qui **ne peuvent pas être émulées** sur IE8

INTÉGRATION DE VUE.JS

Vue Devtools, une **aide** au **développement** avec vue.js:

- Nous vous **recommandons** d'installer Vue Devtools dans votre navigateur.
- Permet d'**inspecter** et de **déboguer** vos **applications** Vue.js dans une **interface dédiée** et intuitive



INTÉGRATION DE VUE.JS

L'**intégration** de Vue.js dans **votre projet** (3 possibilités):

- **Inclusion directe** <Script> par le #CDN
- Par une **commande npm**
- Par **l'interface** de **commande** en **ligne**: CLI officielle

INTÉGRATION DE VUE.JS

- **Inclusion directe** de Vue.js offre 2 versions:
  - **Version de développement** (Avec tous les avertissements et le mode de débogage)
  - **Version de production** (Version minifié, avertissement retirés…)
- Toutes deux sont disponible ici

Il est recommandé de ne pas utiliser la version minifié pendant le développement sous peine de ne pas bénéficier des avertissements pour les erreurs courantes!

INTÉGRATION DE VUE.JS

Utilisation avec **Inclusion directe** de Vue.js (version téléchargé):

1. **Copier** la version choisie (normale ou minifiée) dans **votre projet**
2. **l'inclure** avec une **balise <script>**(Dans le head de votre page :

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mon Document</title>
    <script src="./VotreChemin/vue.js"></script>
</head>
```

INTÉGRATION DE VUE.JS

Utilisation avec **Inclusion directe** de Vue.js (par le CDN):

- **l'inclure** dans le **head** avec une **balise <script>**:
  - Pour du prototypage ou de l'apprentissage, vous pouvez utiliser la dernière version avec :

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

  - Pour la production, nous vous recommandons de vous figer à une version et un build défini pour éviter les changements non compatibles des nouvelles versions :

```
<script src="https://cdn.jsdelivr.net/npm/vue@2.6.0"></script>
```

INTÉGRATION DE VUE.JS

L'**intégration** de **Vue.js** par la commande shell **npm**:

- **npm** est la méthode d'installation **recommandée** lors du développement de **grosses applications** avec **Vue**.

- Il **s'associe** bien avec des **empaqueteurs** de modules comme webpack ou Browserify.

- Vue.js **fournit** également des **outils d'accompagnement** pour la rédaction de **Composants Mono-fichier**.

```shell
# dernière version stable
$ npm install vue
```

INTÉGRATION DE VUE.JS

Installation de la **CLI** pour la création d'un projet **Vue.js**:

- Pour **installer** la **CLI** dans votre **IDE**, **exécuter** la **commande** suivante dans le **shell**:

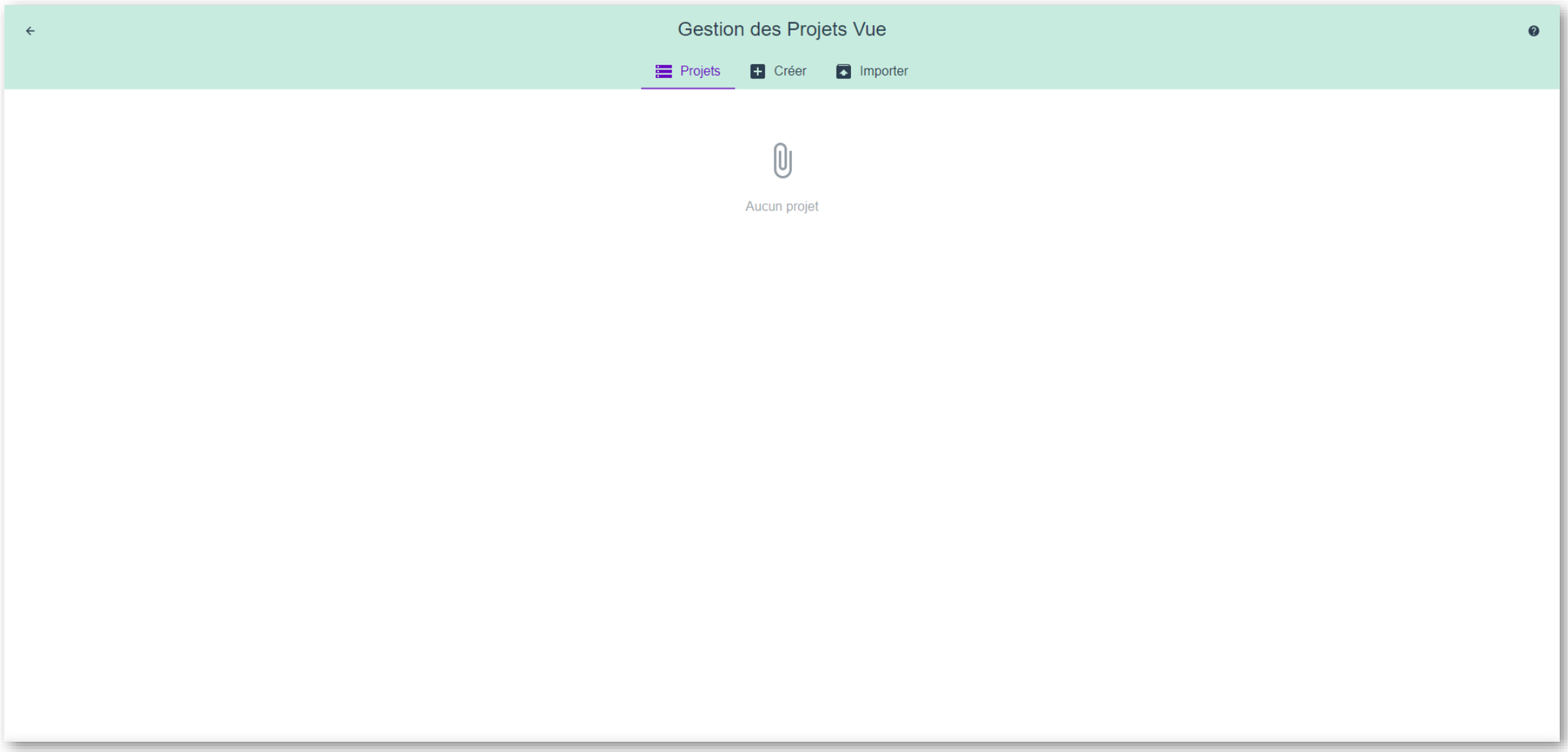
```sh
npm install -g @vue/cli
# OR
yarn global add @vue/cli
```

- Il est possible de **créer directement** un **projet** ou faire appel à **l'interface Utilisateur** (UI) de la **CLI**:

```sh
vue create my-project
# OR
vue ui
```

INTÉGRATION DE VUE.JS

Page **d'accueil de l'interface utilisateur** après installation de la **CLI Vue.js**:



03

RENDU ET DATA-BINDING

Déclarer une instance de Vue, rendu déclaratif et data-binding

RENDU ET DATA-BINDING

- **Vue** a été conçu et **pensé** pour pouvoir être **adopté** de manière **incrémentale**

- Le **cœur** de la **bibliothèque** se concentre **uniquement** sur la partie **vue**

- il est vraiment **simple** de **l'intégrer** avec d'autres **bibliothèques** ou **projets existants**

RENDU ET DATA-BINDING

- Pour **implémenter Vue** dans un projet il faut **déclarer** une **instance de Vue** (objet javascript) en **synergie** avec une **balise HTML**. C'est le **rendu déclaratif:**

  - Dans le HTML:

```
<div id="app">

</div>
```

  - Dans le javascript:

```
var app = new Vue({
    el:"#app"
})
```

- Désormais **Vue** est **opérationnel** dans le **projet**

RENDU ET DATA-BINDING

- La principale **fonctionnalité** de **Vue.js** est **d'exposer** des **données** et **d'actualiser** leurs **états**.
  - Ces **données** sont **contenues** dans un autre **objet javascript** nommé **data**:

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

  - Ces **données** peuvent être **utilisées directement** dans la balise **HTML ciblée** par **Vue** comme suit:

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

Hello Vue!

RENDU ET DATA-BINDING

- Afin de **manipuler** les **données** et les **lier** à des **éléments** du **DOM**, Vue.js emploie **diffèrent** type de **Data-binding**, ce sont les **directives**
- Les **directives** sont **préfixées** par **V-** pour indiquer que ce sont des **attributs spéciaux** fournis par **Vue**
- Il y a **5 directives principales** avec chacune leur **spécificité**:
  - **v-bind:** Pour **relier** un **attribut** à une **propriété** Vue
  - **v-model:** Two-way data-binding
  - **v-for:** Pour les itérations de liste et tableaux (Boucles)
  - **v-if:** Pour les conditions
  - **v-on:** Pour attacher un écouteur d'évènement

RENDU ET DATA-BINDING

- La directives v-bind

  - Coté HTML:

```
<div id="app">
    <span v-bind:title="message">
        Passez votre souris sur moi pendant quelques secondes
        pour voir mon titre lié dynamiquement !
    </span>
</div>
```

  - Coté Javascript:

```
var app = new Vue({
    el: "#app",
    data: {
        message: "Vous avez affiché cette page le " + new Date().toLocaleString()
    }
})
```

  - Rendu Navigateur:

Passez votre souris sur moi pendant quelques secondes pour voir mon titre lié dynamiquement !

Vous avez affiché cette page le 21/12/2020 à 13:09:58

RENDU ET DATA-BINDING

- Les abréviations pour **v-bind**

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

<!-- abréviation -->
<a :href="url"> ... </a>

<!-- abréviation avec argument dynamique (2.6.0+) -->
<a :[key]="url"> ... </a>
```

RENDU ET DATA-BINDING

- La directives **v-model** permet un **binding bidirectionnel**

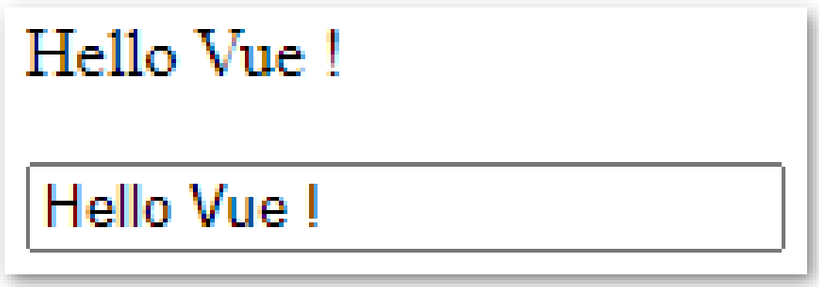
  - Coté HTML:

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

  - Coté Javascript:

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

  - Rendu Navigateur:



RENDU ET DATA-BINDING

- La directives **v-for** pour une **itération** de **liste** ou **tableau**

  - Coté HTML:

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

  - Coté Javascript:

```
var app = new Vue({
    el: "#app",
    data: {
        todos: [
            { text: 'Apprendre JavaScript' },
            { text: 'Apprendre Vue' },
            { text: 'Créer quelque chose de génial' }
        ]
    }
})
```

  - Rendu Navigateur:

1. Apprendre JavaScript
2. Apprendre Vue
3. Créer quelque chose de génial

RENDU ET DATA-BINDING

- **Maintaining State** avec la directives **v-for**
  - **v-for** utilise par **défaut** une stratégie de « **modification localisée** » (in-place patch)
  - Ce **mode** par **défaut** est performant, mais **adapté seulement** lorsque le **résultat** du rendu de votre liste ne **dépend pas** de **l'état d'un composant enfant** ou de **l'état** temporaire **du DOM** (par ex. : les valeurs de champs d'un formulaire)
  - Pour **expliquer** à **Vue** comment suivre **l'identité** de **chaque nœud**, afin que les **éléments existants** puissent être **réutilisés et réordonnés**, vous devez fournir un **attribut** unique **key** pour **chaque élément**

RENDU ET DATA-BINDING

- Exemple de directive **v-for** avec un attribut **key**

```
<div v-for="item in items" :key="item.id">
    <!-- contenu -->
</div>
```

  - Il est **recommandé** de fournir une **key** avec **v-for chaque fois que possible**
  - La **key** a également **d'autres usages** et ne se **limite pas** seulement à son **utilisation** avec **v-for**
  - **Ne pas utiliser** des **valeurs non primitive** comme des **objets** ou des **tableaux** comme **clés** pour **v-for**
  - Il faut **utiliser** des **chaines** de **caractères** ou des **nombres** à la place.

RENDU ET DATA-BINDING

- La directives **v-on** pour attacher un **écouteur d'évènements**

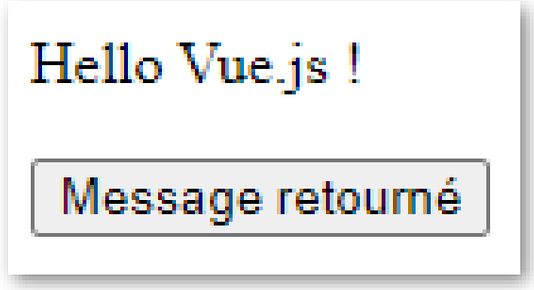
  - Coté HTML:

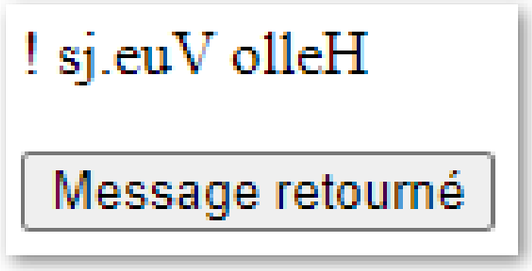
```
<div id="app">
    <p>{{ message }}</p>
    <button v-on:click="reverseMessage">Message retourné</button>
</div>
```

  - Coté Javascript:

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

  - Rendu Navigateur:





RENDU ET DATA-BINDING

- Les abréviations pour **v-on**

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

<!-- abréviation -->
<a @click="doSomething"> ... </a>

<!-- abréviation avec argument dynamique (2.6.0+) -->
<a @[event]="doSomething"> ... </a>
```

RENDU ET DATA-BINDING

- La directives **v-if** est utilisée pour conditionnellement faire le rendu d'un bloc

  - Coté HTML:

```
<div id="app">
    <p v-if="seen">Maintenant vous me voyez</p>
</div>
```

  - Coté Javascript:

```
var app = new Vue({
    el: "#app",
    data: {
        seen: true
    }
})
```

  - Rendu Navigateur:

Maintenant vous me voyez

RENDU ET DATA-BINDING

- Il est également **possible d'ajouter** une structure « **sinon** » avec **v-else**

```
<h1 v-if="awesome">Vue est extraordinaire !</h1>
<h1 v-else>Oh non 😢</h1>
```

- La directive **v-if** ne **doit être attachée qu'à un seul élément.**
- Pour **permuter plusieurs éléments** avec **v-if**, il faut **l'utiliser** sur une **élément** <template>

```
<template v-if="ok">
    <h1>Titre</h1>
    <p>Paragraphe 1</p>
    <p>Paragraphe 2</p>
</template>
```

RENDU ET DATA-BINDING

- Enfin, il est également **possible d'ajouter** une structure « **sinon si** » avec **v-else-if**

```
<div v-if="type === 'A'">
    A
</div>
<div v-else-if="type === 'B'">
    B
</div>
<div v-else-if="type === 'C'">
    C
</div>
<div v-else>
    Ni A, ni B et ni C
</div>
```

- Semblable à **v-else**, un **élément v-else-if** doit **immédiatement suivre** un élément **v-if** ou **v-else-if**, sinon **il ne sera pas reconnu**

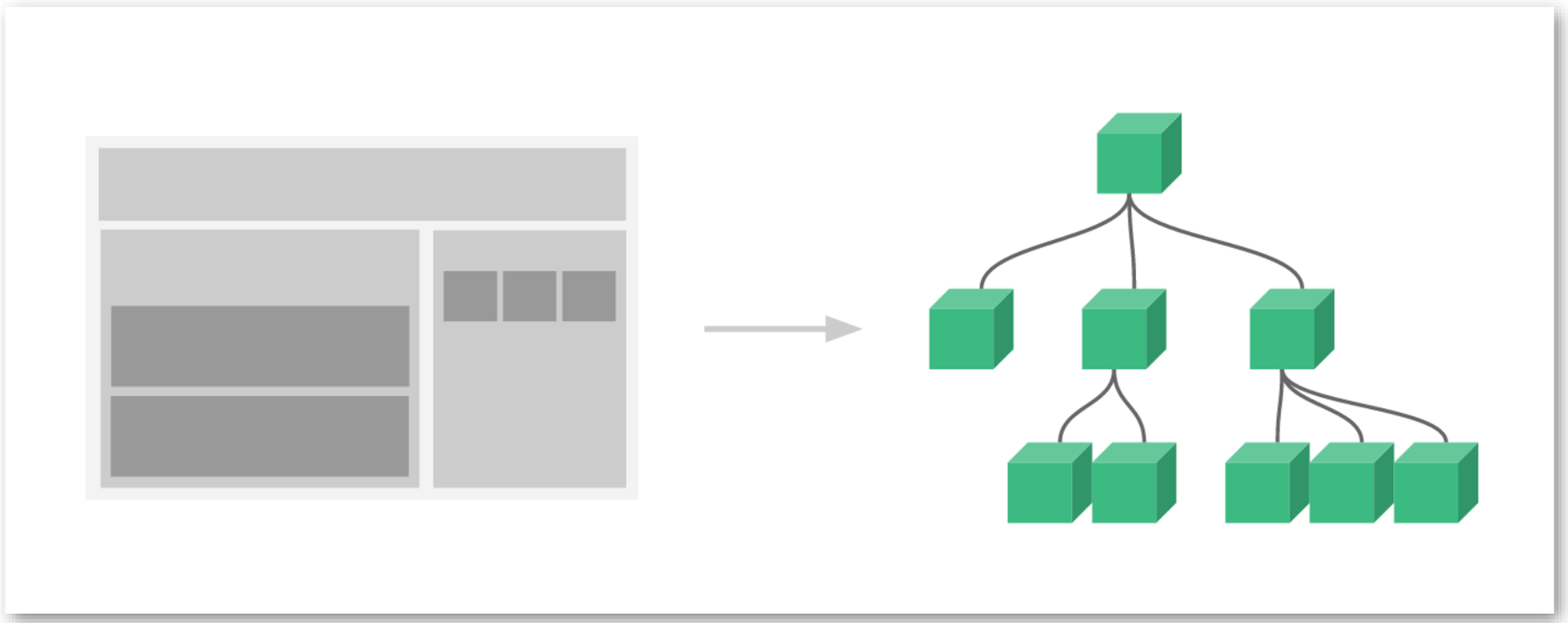
04

COMPOSANTS ET ROUTING

La création de composants, les mixins et le routing dans Vue.js

COMPOSANTS ET ROUTING

- Le système de **composant** est un autre **concept important** de **Vue.js**
  - C'est une **abstraction** qui nous permet de **construire** de grosses **applications découpées** en plus **petits composants réutilisables** et **autonomes**
  - Presque **tous** les **types d'interfaces** applicatives **peuvent** être **abstraits** en un **arbre** de **composants**.



COMPOSANTS ET ROUTING

- Lors de la **création** de **composants**, il faudra **toujours spécifier** un **nom**.
  - Lorsque vous utilisez un **composant** directement dans le **DOM**, il est fortement **recommandé** de suivre les règles du W3C pour les **noms** de **balises personnalisées**
  - Cela vous permet **d'éviter** les **conflits** avec les **éléments HTML** actuels et futurs
- Vous avez **deux options** pour **définir** vos **noms** de **composant**:
  - En **kebab-case**: tout en minuscules, contenir un trait d'union
  - En **PascalCase**: Majuscule au début de chaque mot

COMPOSANTS ET ROUTING

- Dans Vue, un **composant** est essentiellement une **instance** de **Vue** avec des **options prédéfinies**.
- **Déclarer** un **composant** avec **Vue** est très simple :

```
Vue.component("nom-composant", {
    template: "<li>Ceci est une liste</li>"
})
```

- Reprenons l'exemple de **v-for** vue précédemment, nous pouvons désormais **insérer** notre **composant** à **l'intérieur** de **l'instance** de **Vue**:

```
<div id="app">
    <ol>
        <!-- Crée une instance du composant nom-composant -->
        <nom-composant></nom-composant>
    </ol>
</div>
```

COMPOSANTS ET ROUTING

- Modifions la **définition** du **composant** pour lui donner un vrai nom et aussi lui permettre **d'accepter** une prop :

```
Vue.component("todo-item", {
    props:["todo"],
    template: "<li>{{todo.text}}</li>"
})
```

- Reprenons l'exemple de **v-for** vue précédemment, nous pouvons désormais **insérer** notre **composant** à **l'intérieur** de **l'instance** de **Vue**:

```
<div id="app">
    <ol>
        <!-- Pour chaque objet dans le tableau todos -->
        <todo-item
        v-for="item in todos"
        v-bind:todo="item"
        v-bind:key="item.id"
        ></todo-item>
    </ol>
</div>
```

COMPOSANTS ET ROUTING

- Voici le **résultat** après **l'application** du **composant**:

1. Apprendre JavaScript
2. Apprendre Vue
3. Créer quelque chose de génial

- Ceci n'est qu'un exemple grossier, nous avons réussi à **séparer** notre **application** en **deux** plus petites **unités**, et chaque **enfant** est raisonnablement bien **découplé** de son **parent** via **l'utilisation** des **props**.
- Nous pouvons maintenant **améliorer** notre **<todo-item>** avec un **template** et une **logique** plus **complexes** sans affecter **l'application parente**.

COMPOSANTS ET ROUTING

- Pour une **grosse application**, il est nécessaire de la **diviser entièrement** en **composants** afin de **rendre** le **développement** plus **abordable**.
- Voici un **exemple** (imaginaire) de ce à quoi un **template d'application** pourrait ressembler avec des **composants** :

```
<div id="app">
    <app-nav></app-nav>
    <app-view>
        <app-sidebar></app-sidebar>
        <app-content></app-content>
    </app-view>
</div>
```

COMPOSANTS ET ROUTING

- La **création** de **composants Vue** que nous avons découvert précédemment **fonctionne bien** pour des **petits projets**.
- Cependant, pour des **projets** plus **complexes**, ou bien quand votre **front-end** est **entièrement généré** par **JavaScript**, des **désavantages** se manifestent :
  - Les **définitions globales** forcent à avoir un **nom unique** pour **chaque composant**
  - Les **templates** sous forme de **chaines de caractères** ne **bénéficient pas** de la **coloration syntaxique**
  - **L'absence** de **support** pour le **CSS** signifie que le CSS **ne peut pas être modularisé**
  - **L'absence** d'étape de **build** nous **restreint** au **HTML** et à **JavaScript ES5**

COMPOSANTS ET ROUTING

- Tous ces **désavantages** sont **résolus** par les **composants monofichiers** avec une extension **.vue**
- Maintenant nous avons :
  - Une **coloration syntaxique**
  - Des **modules CommonJS**
  - Du **CSS** dont la **portée** est **limitée au composant**
- Au sein d'un **composant .Vue**, son **template**, sa **logique** et ses **styles** sont **couplés**
- Le **composant** est plus **cohérent** et **facile à maintenir**

hello.vue

```vue
<template>
  <p>{{ message }} Vue !</p>
</template>

<script>
module.exports = {
  data: function () {
    return {
      message: "Hello",
    };
  },
};
</script>

<style>
p {
  font-size: 2em;
  text-align: center;
}
</style>
```

COMPOSANTS ET ROUTING

- Si vous n'aimez pas l'idée des **composants monofichiers**, vous pouvez toujours tirer parti du **rechargement** à **chaud** et la **pré-compilation** pour mettre le **CSS** et le **JavaScript** dans des **fichiers séparés**

my-component.vue

```vue
<!-- my-component.vue -->
<template>
  <div>Ce composant sera également pré-compilé</div>
</template>
<script src="./my-component.js"></script>
<style src="./my-component.css"></style>
```

- Dans ce cas il vous suffira de mettre les fichiers **.vue**, **.js** et **.css** dans le **même dossier** au **nom du composant**

COMPOSANTS ET ROUTING

- Avec les **composants .Vue**, nous entrons dans le domaine des applications **JavaScript avancées**
- Cela implique d'apprendre à utiliser quelques **nouveaux outils** si vous ne les connaissez pas déjà :
  - **Node Package Manager** (npm): lisez la section du guide npm Getting Started guide
  - **JavaScript moderne** avec ES2015/16: lisez le guide Babel Learn ES2015 guide
- Une fois que vous aurez pris une journée pour **vous plonger** dans ces **ressources**, nous vous recommandons **d'essayer Vue CLI**. **Suivez** les **instructions** et vous devriez avoir en un clin d'œil un **projet Vue** avec des **composants .Vue**, ES2015 et le **rechargement à chaud**

COMPOSANTS ET ROUTING

Cycle de vie des composants Vue.js

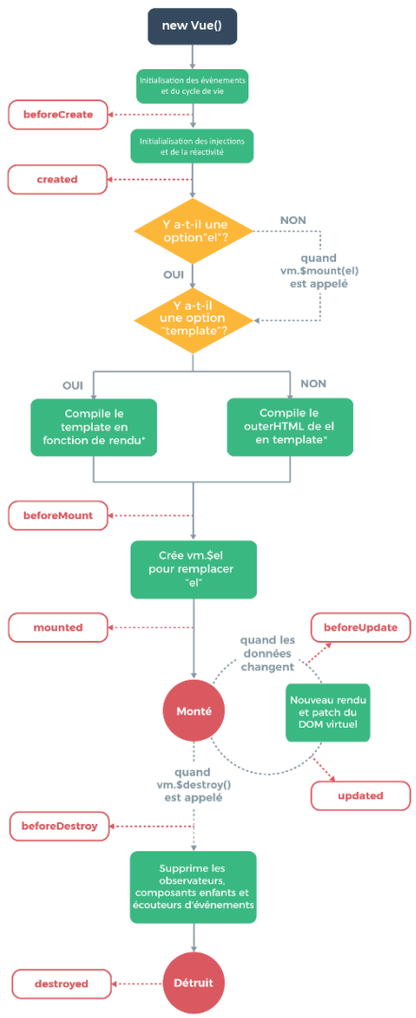
- Chaque **instance** de **vue** traverse une **série d'étapes d'initialisation** au moment de sa **création** et de sa **vie**
  - Mise en place de **l'observation** des **données**
  - **Compilation** du **template**
  - **Monter** l'instance sur le **DOM**
  - **Mettre** à **jour** le **DOM** quand les données **changent**.
- Pour ce faire, elle va invoquer des ***hooks de cycle de vie***, qui nous donnent l'opportunité **d'exécuter** une **logique personnalisée** à **chaque niveau**.

COMPOSANTS ET ROUTING

- Les différents **Hooks de cycle de vie** :
  - created
  - mounted
  - updated
  - destroyed
- Chacun de ces **Hooks** peut se voir appliquer des **actions avant** son **invocation** par l'apposition de **before**
  - beforeCreate
  - beforeMount
  - beforeUpdate
  - beforeDestroy

COMPOSANTS ET ROUTING

- Cycle de vie des composants Vue.js



* la compilation est faite à l'avance si vous utilisez une étape de build, par ex. des composants monofichiers

COMPOSANTS ET ROUTING

Réutilisation d'une **logique** avec les **Mixins** dans **vue.js**

- **Les mixins** offrent une manière **flexible** de créer des **fonctionnalités réutilisables** pour les **composants** de **Vue.**

- Un objet **mixin** peut contenir **toutes** les **options** valide pour un **composant.**

- Si un **composant** utilise un **mixin,** toutes **les options** du **mixin** seront « **fusionnées** » avec les **options** du **composant.**

COMPOSANTS ET ROUTING

- Exemple d'un **composant** dont la **logique** s'appuie **exclusivement** sur un **mixin** :

```
// définir un objet mixin
var myMixin = {
  created: function () {
    this.hello();
  },
  methods: {
    hello: function () {
      console.log("hello from mixin!");
    },
  },
};

// définition d'un composant qui utilise ce mixin
var Component = Vue.extend({
  mixins: [myMixin],
});

var component = new Component(); // => "hello from mixin!"
```

COMPOSANTS ET ROUTING

Le **routing** avec **vue-router**

- Comme la plupart des **applications monopages**, il est **recommandé d'utiliser** la **bibliothèque** officiellement **supportée** vue-router. Pour plus de détails, voir la documentation de vue-router (en Français).

- Si vous avez **simplement besoin** d'un **routage très simple** et **ne souhaitez pas ajouter une bibliothèque** riche en fonctionnalités, vous pouvez le faire en **déclenchant dynamiquement** le **rendu** d'un **composant** (Voir exemple)

COMPOSANTS ET ROUTING

- Créer une **application monopage** avec **Vue** + **Vue-router** est **vraiment simple**
  - Il suffit de **relier** les **composants** aux **routes**, **l'API vue-router s'occupe** de faire le **rendu**

COMPOSANTS ET ROUTING

Si vous **utilisez** un **système** de **module** (par ex. via **vue-cli**), il faut **importer Vue** et **Vue Router** et ensuite **appeler** :

```
vue.use(VueRouter)
```

1. **Définissez** les **composants** de **route**. Ces derniers peuvent être **importés** depuis **d'autre fichier**

```
const Home = {
    template:'<h1>Home</h1>',
    name:'Home'
}
const UserSettings = {
    template:'<h1>User Settings</h1>',
    name:'UserSettings'
}
const WishList = {
    template:'<h1>Wish List</h1>',
    name:'WishList'
}
const ShoppingCart = {
    template:'<h1>Shopping Cart</h1>',
    name:'ShoppingCart'
}
```

COMPOSANTS ET ROUTING

2. Définissez des routes. Chaque route doit correspondre à un composant. Le « composant » peut soit être un véritable composant créé via Vue.extend(), ou juste un objet d'options.

```
const routes:[
        { path:'/', component: Home, name:'Home'},
        { path:'/user-settings', component: UserSettings, name:'UserSettings'},
        { path:'/wish-list', component: WishList, name:'WishList'},
        { path:'/shopping-cart', component: ShoppingCart, name:'ShoppingCart'},
    ]
```

- Lors de l'import de composants vous pouvez utiliser le Lazy Loading comme suit:

```
{
  path: '/about',
  name: 'About',
  component: () => import('../views/About.vue')
}
```

COMPOSANTS ET ROUTING

3. Créez l'instance du routeur et passez l'option « routes ».

```
const router = new VueRouter({
    routes
})
```

4. Créez et montez l'instance de Vue. Soyez sûr d'injecter le routeur avec l'option « router » pour permettre à l'application toute entière d'être à l'écoute du routeur.

```
const vue = new Vue({
    router
}).$mount('#app')
```

COMPOSANTS ET ROUTING

Les « Props » de <router-link>

- **to** (type : **String** ou **Object Location**)
  - Désigne la **route cible** du lien. Lorsqu'il est **cliqué**, la **valeur** de la prop « **to** » va être **passée** de manière **interne** à « **router.push()** »
- **replace** (type : **boolean**, default : **false**)
  - Configurer la **prop** « **replace** » appellera « **router.replace()** » au lieu de « **router.push** ». Lors du **clic**, la **navigation** ne **laissera pas d'enregistrement** dans **l'historique** de navigation.

COMPOSANTS ET ROUTING

Les « **Props** » de **<router-link>**

- **append** (type : **boolean,** default : **false**)
  - Configurer la **prop** « **append** » **suffixe** toujours **le chemin relatif** au **chemin courant**. Par exemple, assumons que nous **naviguons** de « **/a** » vers un lien relatif « **b** », **sans** « **append** » le lien **final** sera « **/b** ». **Avec** « **append** » le lien **final** sera « **/a/b** »
- **tag** (type : **String**, default : **<a>**)
  - Parfois il est **utile** que **<router-link>** soit **rendu** avec une **balise différente**; Exemple : **<li>**. On peut alors **utiliser** la **prop** « **tag = 'li'** » pour **modifier** la **balise** qui sera **rendu**. Elle **écoutera toujours** les **événement** pour la **navigation**

COMPOSANTS ET ROUTING

Les « **Props** » de **<router-link>**

- **exact** (type : **boolean,** default : **false**)
  - Le **comportement** par **défaut** de la **correspondance** de **classe active** est une **correspondance inclusive**
  - Par exemple, **<router-link to='/a'>** verra cette **classe activée** tant que le **chemin courant commencera** par « **/a/** » ou « **/a** »
- Une **conséquence** de cela est que **<router-link to=« / »>** sera **actif** pour **toutes** les **routes** !
  - ✓ Pour **forcer** le lien dans un « **mode correspondance exacte** », utilisez la **prop** « **exact** »

05

LE STATE AVEC VUEX

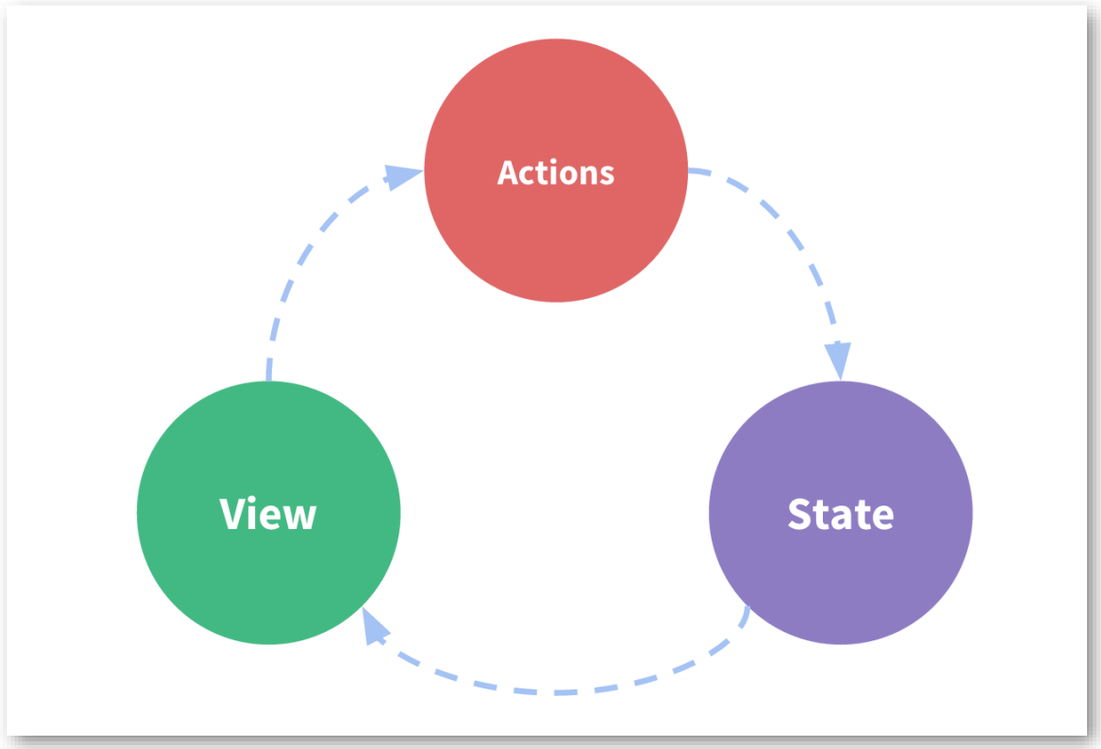
Les particularités de la gestion centralisée du state

LE STATE AVEC VUEX

- **Vuex** est une **bibliothèque** permettant la **gestion** des **états** (States) dans une **application Vue.js**
- **Vuex** fait office de **magasin centralisé** (**Store**) pour les **composants** d'une **application** et il **garantit**:
  - que le **state** ne **peut** être **muté** que de **manière prévisible** (suivant des règles) si vous le souhaitez.
- **Vuex** bénéficie d'une **extension officielle** dans l'outil **Devtools** de Vue.js afin de fournir des **fonctions avancées** de **débogage**.

LE STATE AVEC VUEX

- Avec **Vue.js**, les **composants fonctionnement** de manière **autonome** avec les **éléments suivants**:
  - Le **State**, **source** de **vérité** qui **anime** le **composant**
  - La **Vue**, une **cartographie déclarative** du **State**
  - Les **Actions**, possibilités de **changement** de **State** en **fonction** des **entrées** de **l'utilisateur**.

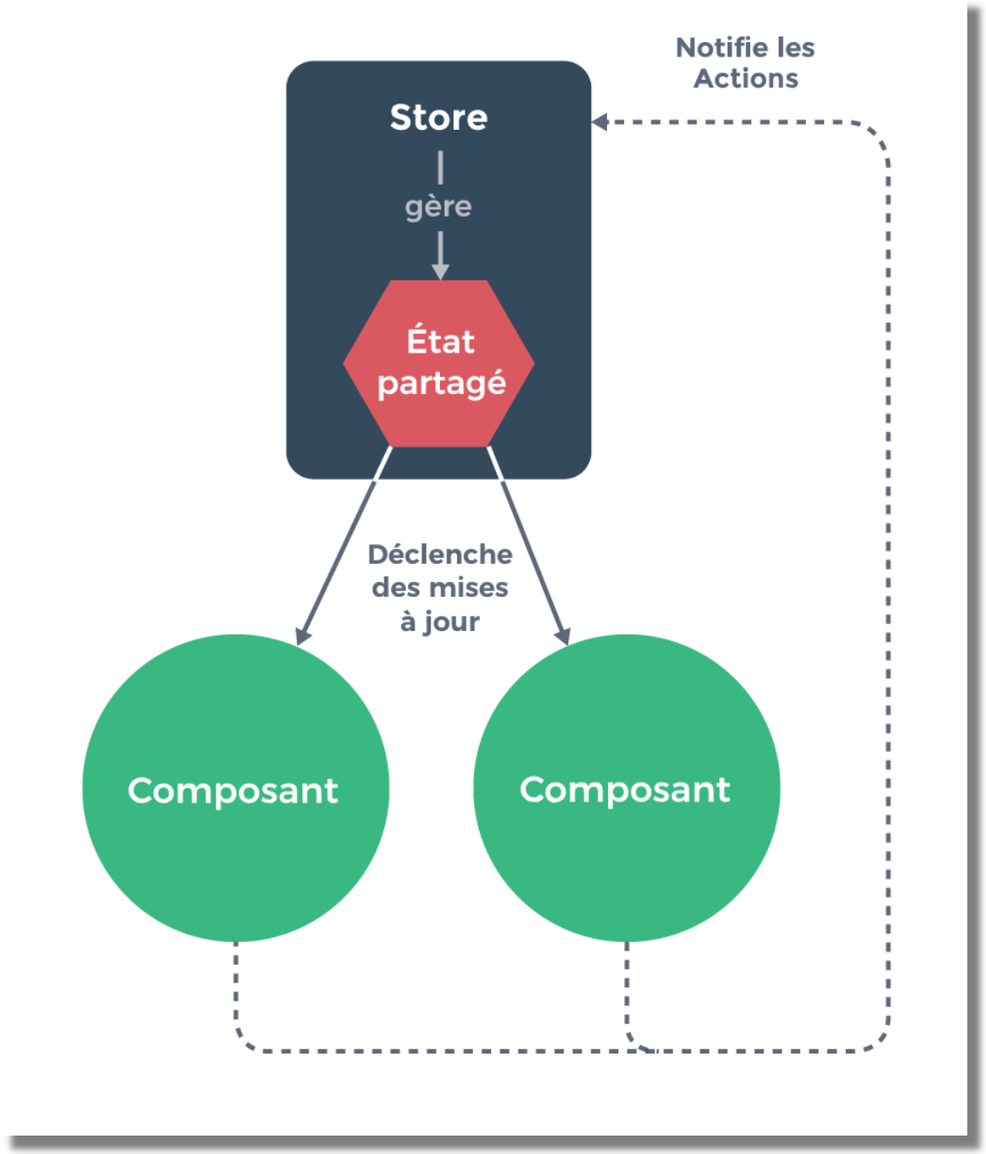


LE STATE AVEC VUEX

- Cependant, la **simplicité s'effondre** rapidement lorsque nous avons **plusieurs composants** qui **partagent** un **état commun** :
  - **Plusieurs vues** peuvent **dépendre** du même **état**
  - Les **actions de différentes vues** peuvent avoir **besoin** de faire **muter** le **même élément** d'état
- La réponse à ces problématiques est **Vuex**, une sorte de **singleton global** permettant de **partager** le **State** entre les composants (Comme Redux pour React) :
  - Notre **arborescence** de **composants** devient une **grande** « **vue** » permettant de **partager** les **ressources**
  - **N'importe** quel **composant** peut accéder à **l'état** ou **déclencher** des **actions**

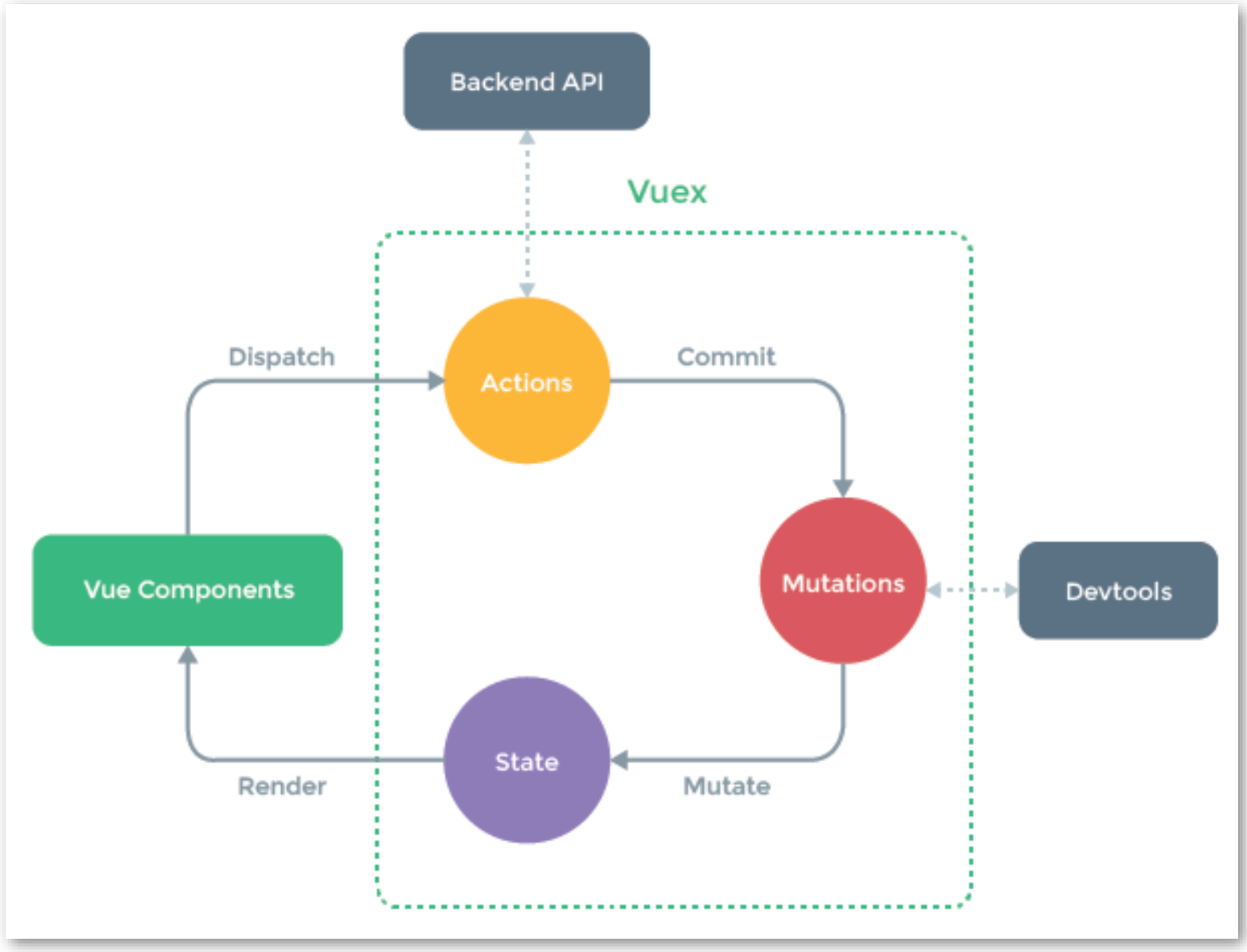
LE STATE AVEC VUEX

- **Vuex** nous aide à **gérer** le **partage** du **State** entre les **composants** avec le **coût** de **plus de concepts** et de **passe-partout**.



LE STATE AVEC VUEX

- Le concept **Vuex**



« LES BIBLIOTHÈQUES DE FLUX SONT COMME DES LUNETTES : VOUS SAUREZ QUAND VOUS EN AUREZ BESOIN. »

DAN ABRAMOV- CRÉATEUR DE REDUX (REACT)- 2015

LE STATE AVEC VUEX

- Pour installer Vuex dans votre application, vous pouvez:
  - **L'inclure** dans votre page **directement** après **Vue**

```
<script src="./script/vue.js"></script>
<script src="/path/to/vuex.js"></script>
```

  - **L'installer** par une **ligne de commande** NPM

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

- Pour utiliser **Vuex** avec un système **modulaire,** vous devez **l'importer** comme un **plugin** (inutile avec les balises script)

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)
```

- **Vuex** nécessite **Promise**, voir doc pour installation *(IE...)*

LE STATE AVEC VUEX

Le **Store** (magasin) est au **centre** de chaque **application Vuex. Deux particularités** rendent **le Store Vuex différent** d'un **objet** global **simple :**

- Il est **réactif**. Lorsque les **composants Vue** en récupèrent **l'état**, ils se **mettent à jour** de manière **réactive** et **efficace** si **l'état** du **magasin change**.
- Vous ne **pouvez pas modifier directement l'état** du **magasin**.
  - La **seule façon** de **changer l'état** d'un **magasin** est de **commettre** explicitement des **mutations** .
  - Cela **garantit** que chaque **changement d'état** laisse un **enregistrement traçable** et permet des **outils** qui nous aident à mieux **comprendre** nos **applications**

LE STATE AVEC VUEX

Afin de **d'afficher/modifier** le **State** d'un **élément partagé** (supporté par le **Store**) **Vuex** fonctionne de la manière suivante:

- Pour la **modification** du **state** depuis un **composant**, on effectue un « **commit** »
- Pour **afficher** le **State** d'un **élément,** le tenir **à jour**, créer des **méthodes** pour la **modifications**, on utilise « **mutations** »

LE STATE AVEC VUEX

- Création d'un **Store**:

```
import Vue from "vue";
import Vuex from "vuex";

Vue.use(Vuex);

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

- Modification du **State** « count »:

```
store.commit('incrementer')
```

- **Intégration** du **store** à note **application** Vue.js:

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

LE STATE AVEC VUEX

- Si vous utilisez **ES6**, vous pouvez également opter pour la **notation abrégée** de **propriété d'objet ES6** (elle est utilisée lorsque la clé d'objet a le **même nom** que la **variable transmise** en tant que **propriété**)

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

- Nous pouvons maintenant **valider** une **mutation** à partir de la **méthode** du **composant**:

```
methods: {
increment() {
  this.$store.commit('increment')
  console.log(this.$store.state.count)
}
```

LE STATE AVEC VUEX

- Vous pouvez **passer** un **argument supplémentaire** à « **store.commit** », qui est appelé la **charge utile** de la **mutation** (Payload)

```
mutations: {
    increment (state, n){
    state.count += n
    },
```

- Dans ce cas le commit prend également cet argument

```
store.commit("increment", 10);
```

LE STATE AVEC VUEX

- Dans la plupart des cas, la **charge utile** doit être un **objet** afin de pouvoir **contenir plusieurs champs**, et la **mutation enregistrée** sera également **plus descriptive**

```
mutations: {
    increment (state, nbUtilisateur){
    state.count += nbUtilisateur.nombre
    },
```

- Dans ce cas le **commit** prend **également** un **objet**

```
store.commit("increment",{nombre:10});
```

LE STATE AVEC VUEX

- Voici un **exemple** de **structure** de **projet Vue.js** intégrant **Vuex**

```
├── index.html
├── main.js
├── api
│   └── ... # abstractions for making API requests
├── components
│   ├── App.vue
│   └── ...
└── store
    ├── index.js          # where we assemble modules and export the store
    ├── actions.js        # root actions
    ├── mutations.js      # root mutations
    └── modules
        ├── cart.js       # cart module
        └── products.js   # products module
```

06

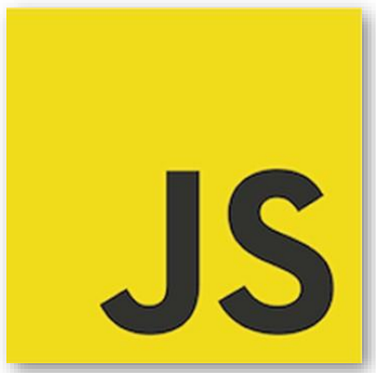
BABEL ET UNIT TESTING

Rendre compatible votre application avec la majorité des web browsers

BABEL ET UNIT TESTING

Vue.js s'appuie **essentiellement** sur les **ressources** du Framework **Javascript** VanillaJS. C'est entre autre:

- Un **Framework incontournable** dans le monde du **développement** Javascript **actuel** (le **plus utilisé** au **monde)**
- Des **dizaines** de **bibliothèques** pour la gestion des **tableaux**, des **chaines** de **caractères**, des **événements**, du **DOM**, de Canvas…
- Extrêmement **léger**, complet et compressé, il ne pèse pas plus de **25 bytes**

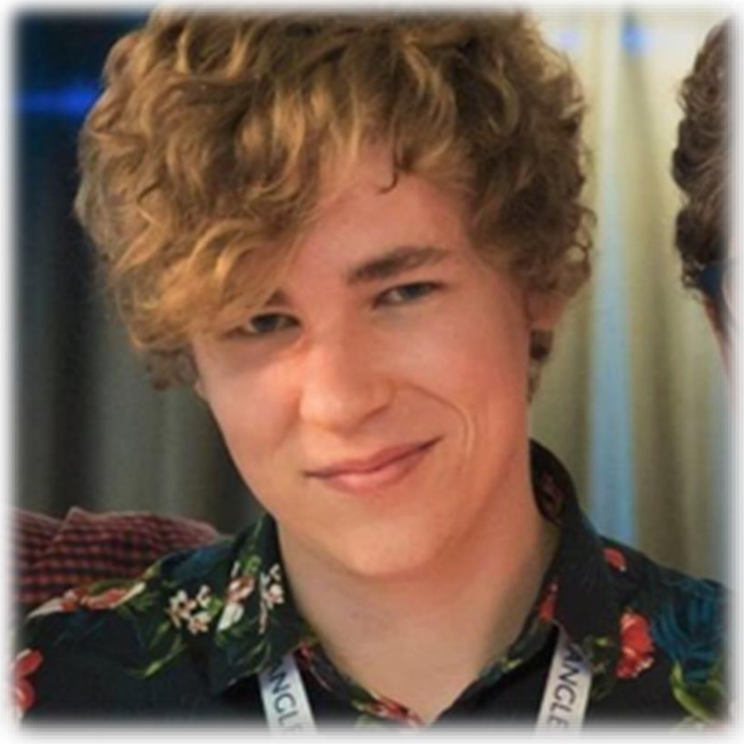


BABEL ET UNIT TESTING

- **Babel** est un **transpiler** qui vous permet de **profiter** des **nouvelles syntaxes** et **fonctionnalités** du **JavaScript avancés** tout en **supportant d'anciennes versions** de **navigateurs**.
- Anciennement nommé "**6to5**" comprendre par là que l'outil **permettait** de prendre du code **JavaScript** dit "**ES6**" (sorti en 2015) et de **générer l'équivalent** en **ES5** (2009) **permettant** donc de **supporter** une **large variété** de **navigateurs**.

BABEL ET UNIT TESTING

- **Sebastian McKenzie** le **créateur** initial de **6to5** l'a donc **renommé** "**Babel**" dans le but d'éviter de changer le nom trop régulièrement.



BABEL ET UNIT TESTING

- **Vue.js supporte** nativement **Babel** et **l'intègre** automatiquement **lors** de la **création** d'un **nouveau projet** via la **CLI**.
- Voici toutefois le **fichier** de **configuration** de **Babel** pour une **application Vue.js**

```
module.exports = {
  presets: [
    '@vue/cli-plugin-babel/preset'
  ]
}
```

BABEL ET UNIT TESTING

- En résumé **Babel** avec **Vue.js** vous **permet** :
  - D'utiliser de **nouvelles fonctionnalités**
  - De **supporter d'anciens navigateurs**
  - **Transformer** certaines **parties** de votre **code**
  - et plus encore (Support de **TypeScript**, **JSX**...)

BABEL ET UNIT TESTING

Les **tests unitaires** (Units Testing) dans **Vue.js**

- **Pourquoi faire des tests?**
  - Ils **fournissent** une **documentation** sur la **façon** dont le **composant** doit **fonctionner**
  - Ils **évitent** d'avoir à **re-tester manuellement** après chaque **changement** du **code**
  - Ils **réduisent** le **risque** de **régression** quand on **ajoute** des **fonctionnalités**
  - Ils **améliorent l'architecture** du code
  - Ils **facilitent** le **refactoring**

BABEL ET UNIT TESTING

- Les **packages officiels** pour **tester** les **composants Vue**
  - Vue Test Utils
  - Vue Server Test Utils
- La **Documentation** de Vue Test Utils
- Le **template** « **Webpack** » disponible via la **CLI** :
  - Contient nativement **Karma** ou **Jest**. Ils sont très **bien intégrés** et **supportés** par **Vue.js**
- La **documentation Officielle** de **Vue.js** contient une partie **consacrée** aux tests unitaires

07

LES MODULES BUNDLERS

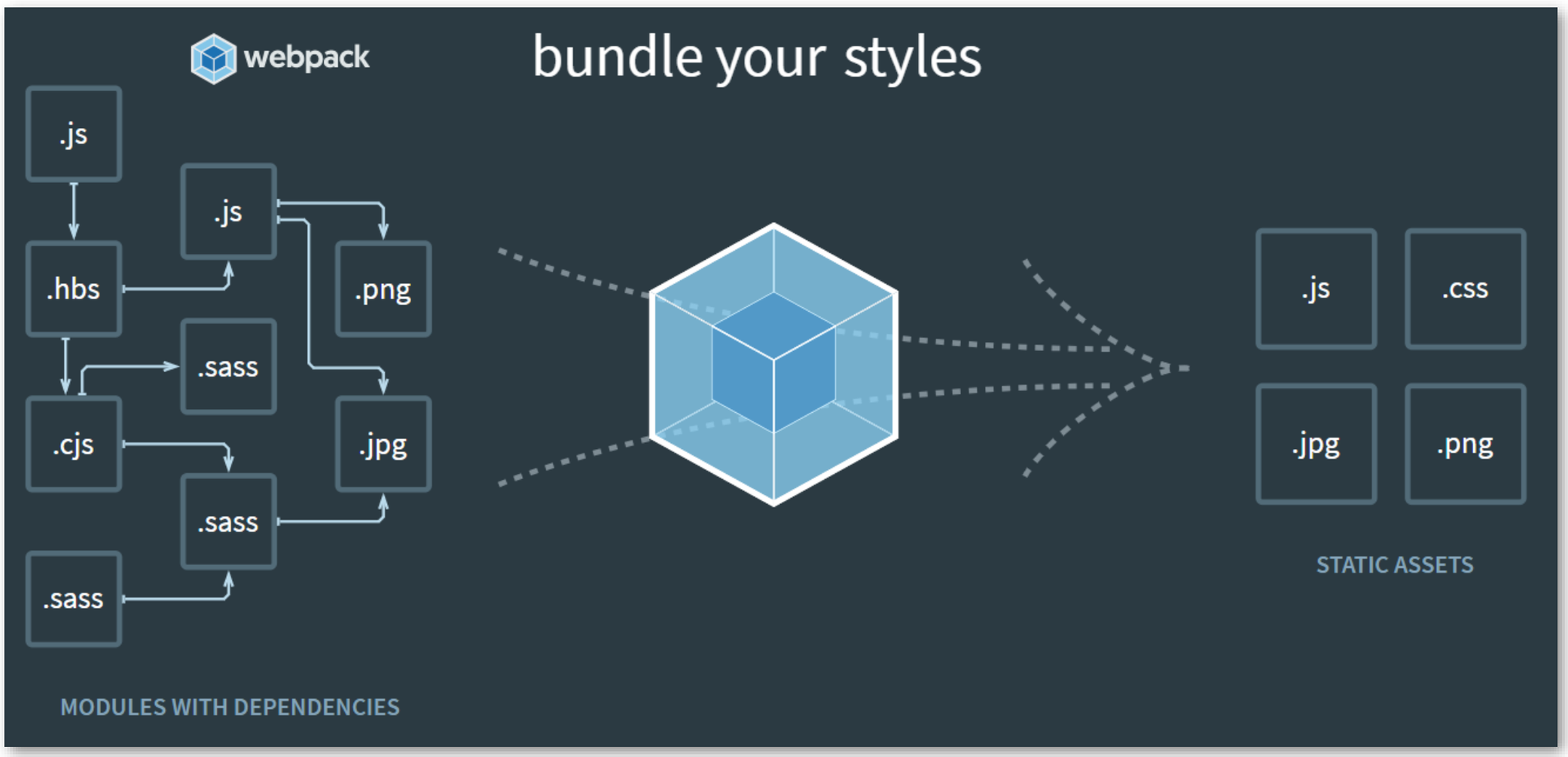
Utilisation d’un module Bundler avec Vue.js

LES MODULES BUNDLERS

- Qu'est-ce qu'un **module bundler**?
  - De **l'anglais bundle** qui signifie **regrouper/empaqueter.**
  - Il **sert** donc à **regrouper** l'ensemble de vos **fichiers/modules Javascript** en **un seul fichier** afin **d'optimiser** votre **application**.
  - En fonction des **options** de **configuration**, il peut également **minifier** vos **scripts** pour une **utilisation** en **production**
- Il existe **plusieurs modules bundlers** actuellement disponible en **Javascript**
  - Browserify, Webpack, rollup, Fuse-box, Parcel...

LES MODULES BUNDLERS

- **Webpack** est **supporté** nativement par **Vue.js** lors de la **création** d'une **nouvelle application** via la **CLI**
  - Nous **utiliserons** donc ce **module bundler** et voici la documentation en anglais



LES MODULES BUNDLERS

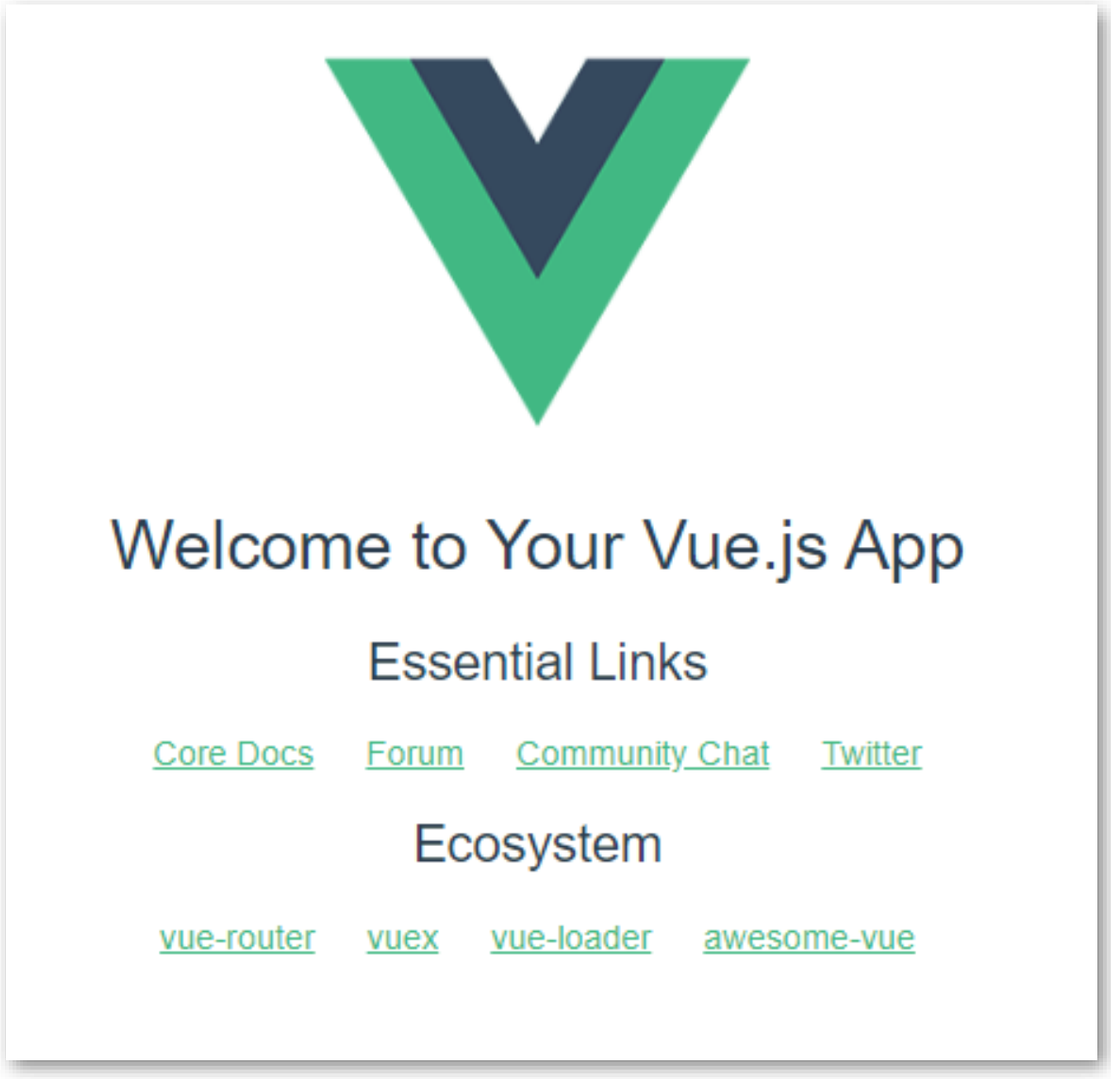
Création d'un **nouveau projet Vue.js** intégrant **webpack** avec l'interface de commande en ligne **(CLI)**

- A partir de **Power Shell** (ou **console** pour **linux** et **mac**)
  - Ouvrez une **fenêtre** de **commande** à partir du **dossier cible** pour **votre** nouveau **projet**
  - Saisir les **commandes** :

```
> npm install –g vue-cli
> vue init webpack-simple « nom-du-projet »
> cd « nom-du-projet »
> npm install
> npm run dev
```

LES MODULES BUNDLERS

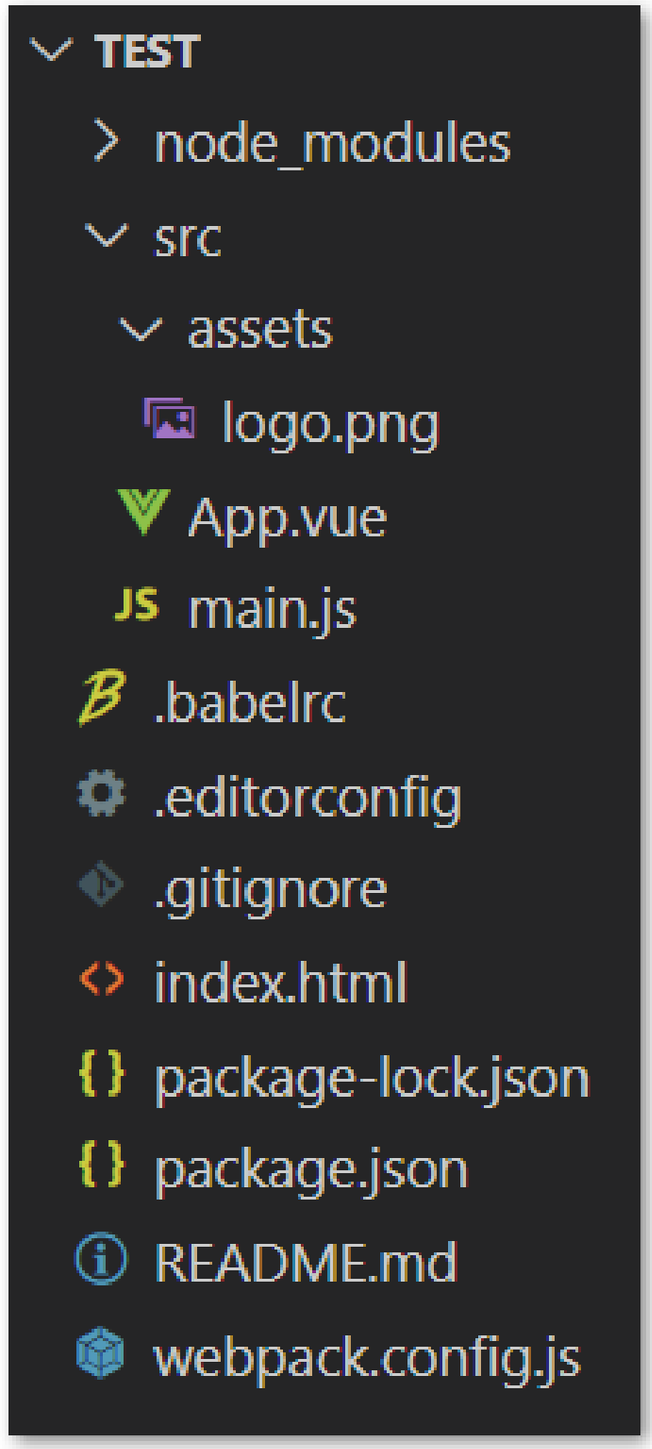
Si **l'installation** s'est **bien déroulée**, vous devez **avoir** une **fenêtre** de votre **navigateur** qui **affiche** le nouveau **projet**



LES MODULES BUNDLERS

Regardons maintenant l'arborescence de notre projet dans l'IDE.

- Il contient notamment :
  - Les dépendances dans node_modules
  - Notre application dans src
  - Le point d'entrée de l'application
  - Le fichier de configuration de Babel
  - La single-page HTML
  - Les fichiers de configuration des dépendances package.json
  - Le fichier de configuration de webpack



08

LES LIBRAIRIES TIERCES

Tour d’horizon des librairies tierces utilisable avec Vue.js

LES LIBRAIRIES TIERCES

Il existe des **librairies non officielle compatible** avec **Vue.js**

- Elle **couvrent** bon nombre **de besoins :**
  - Pour gérer **les appels** et **réponses** des **API Rest**
  - Pour de **l'intégration** de **graphiques** dans **Vue.js**
  - **Et tellement plus encore…**

LES LIBRAIRIES TIERCES

Vue.js étant **essentiellement** orienté **front**, il sera souvent **couplé** à un **back office** ou en **communication** avec une **API**

- **Vue.js** permet d'utiliser **Axios** pour **consommer** des **API**
  - **Axios** est un **client HTTP** basé sur les **Promesses**
    - ✓ Le **téléchargement** et la **documentation** sont disponible sur le repos

LES LIBRAIRIES TIERCES

Il y a **deux possibilités d'intégrer Axios** à votre **projet**

- En utilisant **npm**

```
> npm install axios
```

- En utilisant le **CDN**

Using jsDelivr CDN:

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

Using unpkg CDN:

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

LES LIBRAIRIES TIERCES

Intégrer un **graphique** dans **Vue.js** avec la bibliothèque Chart.js

- **Vue.js** est compatible avec **Chart.js** pour **afficher** des **graphiques** dans la balise HTML **<canvas>**
  - Le **téléchargement** et la **documentation** son disponible sur le site

LES LIBRAIRIES TIERCES

Pour **créer** un **projet** intégrant **Chart.js**, procéder comme suit:

- **Charts.js** fonctionne avec le **template webpack-simple**

```
> vue init webpack-simple « nom-du-projet »
> cd « nom-du-projet »
> npm install
> npm install chart.js --save
```

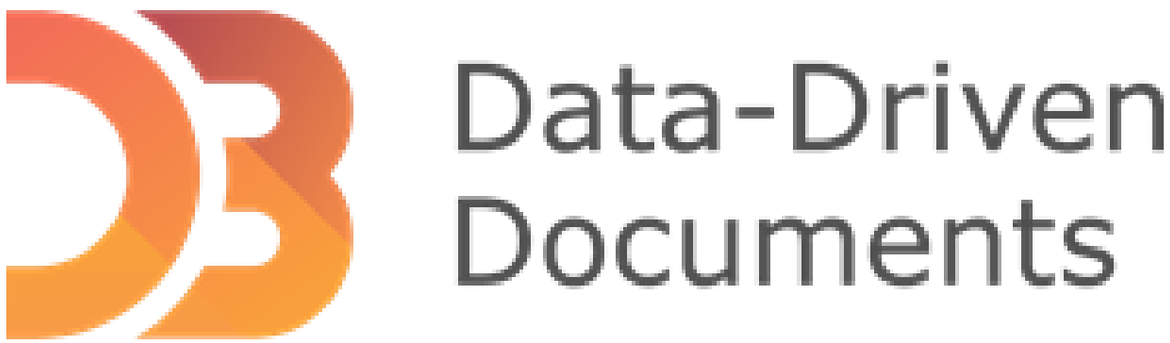
- Dans le fichier App.vue, importer Chart.js

```
import Chart from "chart.js";
```

LES LIBRAIRIES TIERCES

Intégrer un **graphique** dans **Vue.js** avec la bibliothèque **D3.js**

- **Vue.js** est compatible avec **D3.js** pour **afficher** des **graphiques** dans une balise **<svg>**
  - Le **téléchargement** et la **documentation** son disponible sur le site



LES LIBRAIRIES TIERCES

Il y a **deux possibilités d'intégrer D3.js** à votre **projet**

- En téléchargeant le .zip
- En utilisant le **CDN**

```
<script src="https://d3js.org/d3.v6.min.js"></script>
```

- Voir le **projet** « **demod3js** » dans le **dossier code** pour une **mise** en **pratique** de la **librairie**