



The Majesty of Vue.js 2

Alex
Kyriakidis

Kostas
Maniatis

The Majesty of Vue.js 2 (French)

Alex Kyriakidis, Nacim IDJAKIRENE et Kostas Maniatis

Ce livre est en vente à <http://leanpub.com/vuejs2-french>

Version publiée le 2017-02-18



Ce livre est publié par [Leanpub](#). Leanpub permet aux auteurs et aux éditeurs de bénéficier du Lean Publishing. [Lean Publishing](#) consiste à publier à l'aide d'outils très simples de nombreuses itérations d'un livre électronique en cours de rédaction, d'obtenir des retours et commentaires des lecteurs afin d'améliorer le livre.

© 2016 - 2017 Alex Kyriakidis, Nacim IDJAKIRENE et Kostas Maniatis

Table des matières

Introduction	i
A propos de Vue.js	ii
vue d'ensemble de Vue.js	ii
Ce que les gens disent de Vue.js	ii
Comparaison avec d'autres framework	iv
Bienvenue	xii
A propos du livre	xii
A qui s'adresse ce livre	xii
Contactez nous	xii
Exercices	xiii
Errata	xiii
Les fondamentaux de Vue.js	1
Installer Vue.js	2
Version Autonome (standalone)	2
Télécharger à l'aide de NPM	2
Télécharger en utilisant Bower	3
Mise en marche	4
Hello World	4
Data binding bidirectionnel	6
Comparaison avec jQuery	7
Devoirs	9
Une saveur de directives	10
v-show	10
v-if	13
v-else	15
v-if vs. v-show	17
Devoirs	19
Rendu des listes	20

TABLE DES MATIÈRES

Install & Use Bootstrap	20
v-for	22
Rendu des tableaux	24
v-for pour les objets	29
devoirs	31
Interactivité	32
Gestion des événements	32
Modificateurs d'événements	36
Modificateurs de clés (Key Modifiers)	40
Propriétés calculées	41
Devoirs	47
Filtres	49
Résultats filtrés	49
Résultats ordonnées	59
Filtres personnalisés	63
Librairies utilitaires	64
Devoirs	67
Components	68
Que sont les composants?	68
Utilisation des composants	68
Templates	70
Propriétés	71
Réutilisabilité	74
Mise en pratique	78
Devoirs	85
Événements personnalisés	86
Emission et écoute	86
Communication parents-enfants	89
Passer des arguments	91
Communication non-parent-enfant	96
Suppression d'écouteurs d'événements	99
Retour aux histoires	100
Devoirs	103
Binding de class et style	105
Binding de class	105
Binding de styles	110
Le binding en action	112
Devoirs	115

TABLE DES MATIÈRES

Consommation d'une API	116
Preface	117
CRUD	117
API	117
Utiliser des données réelles	122
Obtenir des données asynchrones	122
Refactoring	126
Mise à jour des données	128
Suppression des données	130
Intégration vue-resource	133
Aperçu	133
Migration	134
Amélioration des fonctionnalités	135
Fichier Javascript	147
Code source	147
Devoirs	153
Pagination	155
Implementation	156
Liens de pagination	159
Devoirs	162
Construire des applications à grande échelle	163
ECMAScript 6	164
Introduction	164
Déclaration de variables	165
Les Fonctions fléchées (Arrow Functions)	166
Modules	167
Classes	168
Valeurs par défaut des arguments	169
Littéraux de gabarits (Template literals)	170
Workflow avancé	172
Compilation ES6 avec Babel	172
Automatisation de workflow avec Gulp	181
Bundling de Modules avec Webpack	185
Résumé	193
Utilisation des Composants à fichier unique	195
vue-cli	195
Les Templates Webpack	198

TABLE DES MATIÈRES

Création des fichiers <i>.vue</i>	206
Elimination des états dupliqués	220
Partage de données à l'aide des propriétés	220
Store Global	225
Permutation entre composants	229
Composants Dynamiques	229

Introduction

A propos de Vue.js

vue d'ensemble de Vue.js

Vue (pronouncez /vju:/) est un framework progressif pour la conception d'interfaces utilisateurs. contrairement à d'autres framework monolithiques, Vue est conçu à partir de la base pour être adoptable progressivement. Le core Vue JS se concentre principalement sur le **la couche visuelle**, et est très facile à prendre en main et à intégrer avec d'autres bibliothéques ou projets existants. D'autre part, Vue convient parfaitement pour la réalisation d'applications single page sophistiquées quand utilisé en combinaison avec les outils modernes et [Librairies supportées](#).¹

Si vous êtes un développeur frontend expérimenté et que vous voulez savoir comment Vue.js se compare aux autres librairies/frameworks, consultez le chapitre [Comparaison avec les autres frameworks](#).

Si cela vous intéresse d'en savoir plus sur le noyau de Vue.js consultez le [Guide officiel Vue.js](#)².

Ce que les gens disent de Vue.js

“Vue.js est ce qui m'a fait aimer JavaScript. Il est extrêmement facile et agréable à utiliser. Il dispose d'un grand écosystème de plugins et d'outils qui étend ses services de base. Vous pouvez rapidement l'inclure dans n'importe quel projet, petit ou grand, écrire quelques lignes de code et vous êtes prêt. Vue.js est rapide, léger et est l'avenir du développement Front End !”

—Alex Kyriakidis

“Quand j'ai commencé à m'intéresser au Javascript, j'étais sorti en découvrant les tas de possibilités offertes à moi, mais quand j'ai suivi les conseils de mon ami qui m'avait suggéré d'apprendre Vue.js, les choses sont devenues folles. Tout en lisant et en regardant des tutoriaux, j'ai continué à penser à toutes les choses que j'avais fait jusqu'à présent et combien ça serait plus facile si j'avais investi du temps pour apprendre Vue plus tôt. Mon avis est que si vous voulez que votre travail soit rapide, agréable et facile, Vue est le framework JS dont vous avez besoin”

—Kostas Maniatis

¹<https://github.com/vuejs/awesome-vue#libraries--plugins>

²<http://vuejs.org/guide/overview.html>

“Notez mes mots : Vue.js va monter en flèche en popularité en 2016.”

— **Jeffrey Way**

“Vue est ce que j’ai toujours voulu dans un framework JavaScript. C’est un framework qui évolue avec vous en tant que développeurs. Vous pouvez l’inclure sur une page, ou construire une application signle-page avancée et unique avec Vuex et Vue Router. C’est vraiment le framework JavaScript le plus élégant que j’ai jamais vu.”

— **Taylor Otwell**

“Vue.js est le premier framework que j’ai trouvé tout aussi naturel à utiliser dans une application avec rendu coté serveur qu’une application sigle page complète. Si j’ai juste besoin d’un petit widget sur une seule page ou que je construit un client Javascript complexe, il ne donne jamais l’impression que ce n’ai pas assez ni que c’est exagéré.”

— **Adam Wathan**

“Vue.js a été en mesure de faire un framework qui est à la fois simple à utiliser et facile à comprendre. C’est un souffle d’air frais dans un monde où d’autres se battent pour voir qui peut faire le framework le plus complexe.”

— **Eric Barnes**

“La raison pour laquelle j’aime Vue.js, c’est que je suis un designer / développeur hybride. J’ai jetté un coup d’oeil à React, Angular et quelques autres, mais la courbe d’apprentissage et la terminologie m’a toujours mis hors jeux. Vue.js est le premier framework JS que je comprends. En outre, non seulement il est facile de prendre en main pour les développeurs Javascript moins confiant comme moi-même, mais j’ai aussi remarqué que des développeurs Angular et React expérimentés prenaient note et appréciaient Vue.js. Ceci est assez inédit dans le monde du JS et c’est pour cette raison que j’ai commencé le Vue.js Meetup à Londres.”

— **Jack Barham**

Comparaison avec d'autres framework

Angular 1

Une partie de la syntaxe de Vue ressemble beaucoup à celle d'Angular (par exemple `v-if` contre `ng-if`). C'est parce qu'il y avait beaucoup de choses que Angular a eu raison de faire, et ceux-ci étaient une inspiration pour Vue très tôt dans son développement. Cependant, il ya aussi beaucoup de problèmes qui viennent avec Angular, c'est là où Vue a tenté d'offrir une amélioration significative.

Complexité

Vue est beaucoup plus simple qu'Angular 1, à la fois en terme d'API et de conception. En apprendre assez pour construire des applications non triviales prend généralement moins d'une journée, ce qui n'est pas forcément le cas pour Angular 1.

Flexibilité et Modularité

Angular 1 a de fortes opinions sur la façon dont vos applications doivent être structurées, tandis que Vue.js est une solution plus flexible et modulaire. C'est pourquoi un [tempalte Webpack³](#) vous est fourni, qui peut être mis en place en quelques minutes, tout en vous donnant également accès à des fonctionnalités avancées telles que le rechargeement de module à chaud (hot reloading), linting, CSS Extraction, et bien plus encore.

Data binding

Angular 1 utilise le data binding bidirectionnel entre les scopes, tandis que Vue applique un flux de données unidirectionnel entre les composants. Cela rend le flux de données plus facile à raisonner dans les applications non triviales.

Directives vs Composants

Vue a une séparation plus claire entre les directives et les composants. Les directives sont destinées à encapsuler des manipulations du DOM uniquement, alors que les composants sont des unités autonomes qui ont leur propre vue et leur propre logique de données. Dans Angular, il ya beaucoup de confusion entre les deux.

Performances

Vue a de meilleures performances et est beaucoup, beaucoup plus facile à optimiser car il n'utilise pas de Dirty checking comme Angular le fait. Angular 1 devient lent quand il ya beaucoup d'observateurs, car à chaque fois que quelque chose dans le scope change, tous ces observateurs

³<https://github.com/vuejs-templates/webpack>

doivent être réévalués à nouveau. En outre, le Digest cycle peut avoir à s'exécuter plusieurs fois pour se « stabiliser » si un observateur déclenche une autre mise à jour. Les utilisateurs d'Angular doivent souvent recourir à des techniques ésotériques pour contourner le cycle de digestion, et dans certaines situations, il n'y a simplement aucun moyen d'optimiser un scope avec de nombreux observateurs.

Vue ne souffre pas de tout ça parce qu'il utilise un système transparent d'observation de dépendance avec une file d'attente asynchrone - tous les changements se déclenchent indépendamment à moins qu'ils aient des relations de dépendance explicites.

Fait intéressant, il ya quelques similitudes dans la façon dont Angular 2 et Vue abordent les problèmes d'Angular 1.

Angular 2

Voici une section distincte pour Angular 2 parce que c'est vraiment un framework complètement nouveau. Par exemple, il dispose d'un système de composants de première classe, de nombreux détails d'implémentation ont été entièrement réécrits, et l'API a également changé assez drastiquement.

taille et Performances

En terme de performances, les deux framework sont exceptionnellement rapides et il n'y a pas assez de données provenant de cas d'utilisation du monde réel pour donner un verdict. Cependant, si vous êtes déterminé à voir certains chiffres, Vue 2.0 semble être en avance sur Angular 2 selon ce [benchmark tier⁴](#). [3rd party benchmark⁵](#).

La taille reste raisonnable, bien que Angular 2 avec la compilation hors ligne et le tree shaking est en mesure de réduire considérablement sa taille , un Vue 2.0 complet avec compilateur inclus (23kb) est encore plus léger qu'un exemple de Angular 2 apres un tree shaking (50kb).

Flexibilité

Vue est beaucoup moins fermé qu'Angular 2, offrant un soutien officiel pour une variété de systèmes de build, sans aucune restriction sur la façon dont vous structurez votre application. Beaucoup de développeurs bénéficient de cette liberté, alors que certains préfèrent avoir une seule bonne façon de construire n'importe quelle application.

Courbe d'apprentissage

Pour commencer à utiliser Vue, il suffit de connaître l'HTML et le langage JavaScript ES5 (c'est-à-dire le langage JavaScript simple). Avec ces compétences de base, vous pouvez commencer à construire des applications non simples en moins d'une journée de lecture du guide.

⁴<http://stefankrause.net/js-frameworks-benchmark4/webdriver-ts/table.html>

⁵<http://stefankrause.net/js-frameworks-benchmark4/webdriver-ts/table.html>

La courbe d'apprentissage d'Angular 2 est beaucoup plus abrupte. Même sans TypeScript, leur [guide de démarrage] (<https://angular.io/docs/js/latest/quickstart.html>) commence avec une application qui utilise JavaScript ES2015, NPM avec 18 dépendances, 4 fichiers et plus de 3.000 mots Pour tout expliquer - juste pour afficher Hello World.

React

React et Vue partagent de nombreuses similitudes. Tout les deux :

- utilisent une DOM virtuel
- Fournissent des composantes de vue réactives et composables
- Se concentrent sur la bibliothèque core, avec des préoccupations telles que le routage et la gestion globale des états gérés par les bibliothèques associées

Profils de performance

Dans chaque scénario réel qui a été testé jusqu'à présent, Vue surpassé React en performance.

Performances de rendu

Lors du rendu de l'interface utilisateur, manipuler le DOM est généralement l'opération la plus coûteuse et malheureusement, aucune librairie ne peut rendre ces opérations brutes plus rapides. Ce qui peut être fait, au mieux :

1. Minimiser le nombre de mutations DOM nécessaires. React et Vue utilisent des abstractions DOM virtuelles pour accomplir cela et les deux implémentations fonctionnent de façon équivalentes.
2. Ajouter le moins possible de surcharge au-dessus de ces manipulations DOM. Il s'agit d'un sujet où Vue et React diffèrent. Dans React, disons que la surcharge de rendu d'un élément est de 1 et la surcharge d'une composante moyenne est de 2. Du côté de Vue.js, la surcharge d'un élément serait plus comme 0,1, mais la surcharge d'une composante moyenne serait de 4, due à la configuration requise pour le système de réactivité.

Cela signifie que dans les applications typiques, où il ya beaucoup plus d'éléments que les composants étant rendus, Vue.js va surpasser React par une marge significative. Dans des cas extrêmes cependant, comme utiliser un composant normal pour rendre chaque élément, Vue sera généralement plus lent.

Vue.js et React offrent également des composants fonctionnels, qui sont non dépendants de l'état de l'application ni de l'instance - et donc, nécessitent moins de frais généraux. Lorsqu'ils sont utilisés dans des situations critiques, Vue est à nouveau plus rapide.

Performances de mise à jour

Dans React, vous devez implémenter `shouldComponentUpdate` partout et utiliser des structures de données immuables pour obtenir des re-rendus entièrement optimisés. Dans Vue.js, les dépendances d'un composant sont automatiquement suivies de sorte qu'elles ne sont mises à jour que lorsqu'une de ces dépendances changent. La seule autre optimisation qui peut parfois être utile dans Vue.js est l'ajout d'un attribut clé aux éléments de longues listes.

Cela signifie que les mises à jour dans Vue.js non optimisé seront en fait beaucoup plus rapides que les mises à jour React non optimisés, en raison de la performance de rendu améliorée dans Vue.js, même du code React pleinement optimisé sera généralement plus lent que du code Vue.js non optimisé.

En Développement

Évidemment, la performance en production est la plus importante et c'est ce dont nous avons discuté jusqu'ici. Cependant, la performance en mode développement demeure importante. La bonne nouvelle est que Vue et React restent assez rapides en développement pour la plupart des applications normales.

Cependant, si vous projetez des visualisations ou des animations de données à haute performance, vous trouverez peut-être utile de savoir que dans des scénarios où Vue.js ne peut pas traiter plus de 10 images par seconde en développement, nous avons vu React ralentir à environ 1 image par seconde.

Cela est dû aux nombreux contrôles invariant lourds de React, qui l'aident à fournir de nombreux et excellents avertissements et messages d'erreur.

Ember

Ember est un framework complet qui est conçu de manière très fermée. Il fournit beaucoup de conventions établies et une fois que vous êtes assez familier avec eux, il peut vous rendre très productif. Cependant, cela signifie également que la courbe d'apprentissage est élevée et que la flexibilité en souffre. C'est un compromis lorsque vous essayez de choisir entre un framework fortement structuré et une bibliothèque avec un ensemble d'outils faiblement couplés qui travaillent ensemble. Ce dernier vous donne plus de liberté, mais aussi vous oblige à prendre des décisions plus architecturales.

Cela dit, il ferait probablement une meilleure comparaison entre le cœur Vue.js avec le templating et Models d'Ember :

- Vue offre une réactivité discrète sur les objets JavaScript simples et des propriétés calculées entièrement automatiques. Avec Ember, vous devez envelopper tout dans Objets Ember et déclarer manuellement les dépendances pour les propriétés calculées.

- La syntaxe de modèle de Vue exploite toute la puissance des expressions JavaScript, tandis que l'expression de Handlebars et la syntaxe d'aide sont intentionnellement assez limitées en comparaison.

- En termes de performance, Vue surpassé Ember d'une marge conséquente, même après la dernière mise à jour du moteur Glimmer dans Ember 2.0. Vue.js reproduit automatiquement les mises à jour, alors que dans Ember, vous devez gérer manuellement les boucles d'exécution dans des situations critiques.

Polymer

Polymer est encore un autre projet parrainé par Google, et a aussi été une source d'inspiration pour Vue.js. Les composants de Vue peuvent être globalement comparés aux custom éléments de Polymer et tous deux offrent un style de développement très similaire. La plus grande différence est que Polymer est construit sur les dernières fonctionnalités de Web Components et nécessite des polyfills non triviales pour fonctionner (avec des performances dégradées) dans les navigateurs qui ne prennent pas en charge ces fonctionnalités nativement. En revanche, Vue.js travaille sans aucune dépendance ou polyfills jusqu'à IE9.

Dans Polymer 1.0, l'équipe a également rendu son système de liaison de données très limité afin de compenser la performance. Par exemple, les seules expressions prises en charge dans les modèles Polymer sont la négation booléenne et les appels de méthode unique. Sa mise en œuvre de la propriété calculée n'est pas non plus très flexible.

Les custom elements (éléments personnalisés) dans polymer sont créés dans des fichiers HTML, ce qui vous limite au langage JavaScript / CSS standard (et aux fonctionnalités de langue prises en charge par les navigateurs d'aujourd'hui). En comparaison, les composants de fichier unique Vue vous permettent d'utiliser facilement ES2015 + et tous les préprocesseurs CSS que vous voulez.

Lors du déploiement en production, Polymer recommande de charger tout à la volée avec les importations HTML supportées par les navigateurs implémentant la spécification, et le soutien HTTP / 2 sur le serveur et le client. Cela peut ou non être réalisable en fonction de votre public cible et de l'environnement de déploiement. Dans les cas où cela n'est pas souhaitable, vous devrez utiliser un outil spécial appelé Vulcanizer pour regrouper vos éléments Polymer. Sur ce front, Vue.js peut combiner sa fonction de composant asynchrone avec la fonction de fractionnement de code de Webpack pour séparer facilement des parties du paquet d'application pour être chargé au besoin (lazy-loaded). Cela garantit la compatibilité avec les navigateurs plus anciens tout en conservant les performances de chargement des applications.

Riot

Riot 2.0 fournit un modèle de développement basé sur des composants similaires (appelé « tag » dans Riot), avec une API minimale et très bien conçue. Riot et Vue.js partager probablement beaucoup de philosophies de conception. Cependant, et en dépit d'être un peu plus lourd que Riot, Vue.js offre quelques avantages significatifs :

- Vrai rendu conditionnel. Riot fait un rendu de toutes les branches puis il les affiche / les masque.
- Un routeur beaucoup plus puissant. L'API de routage de Riot est extrêmement minime.
- Support d'outillage plus mature. Vue fournit un support officiel pour Webpack, Browserify et SystemJS, tandis que Riot s'appuie sur le support de la communauté pour l'intégration du système de build.
- Système d'effet de transition. Riot n'en a pas.
- Meilleure performance. En dépit de publicité sur l'utilisation d'un DOM virtuel, Riot utilise en fait le Dirty checking et souffre donc des mêmes problèmes de performance que Angular 1.

Pour des comparaisons mises à jour, consultez le [guide Vue.js](#).

Bienvenue

A propos du livre

Ce livre vous guidera vers la maîtrise du Vue.js, un framework javascript qui est entrain de gagne rapidement en popularité !

A plusieurs reprise, nous avons eu à réaliser un projet basé sur Laravel et Vue.js. Après avoir lu attentivement le guide de Vue.js et quelques tutoriels, nous avons découvert un manque de ressources à propos de Vue.js sur le web. Au cours du développement de notre projet, nous avons acquis beaucoup d'expérience, nous avons donc eu l'idée d'écrire ce livre afin de partager nos connaissances acquises.

Ce livre est écrit dans un format informel, intuitif et facile à suivre, dans lequel tous les exemples sont suffisamment détaillés pour fournir des orientations appropriées à quiconque.

Nous allons commencer par les bases et à travers de nombreux exemples, nous allons couvrir les caractéristiques les plus importantes de Vue.js. À la fin de ce livre, vous serez en mesure de créer des applications front-end rapides et augmenter la performance de vos projets existants avec l'intégration Vue.js.

A qui s'adresse ce livre

Tous ceux qui ont passé du temps à apprendre le développement web moderne, qui ont vu Bootstrap, Javascript et plusieurs frameworks Javascript. Ce livre est pour toute personne intéressée par l'apprentissage d'un framework Javascript simple et léger. Aucune connaissance excessive n'est nécessaire, mais il serait bon de se familiariser avec le langage HTML et Javascript. Si vous ne savez pas quelle est la différence entre une chaîne de caractères et un objet, peut-être que vous devriez faire quelques recherches en premier.

Ce livre est aussi utile pour tout lecteur qui a déjà fait un petit bout de chemin avec Vue.js et veut élargir ses connaissances.

Contactez nous

Si vous souhaitez nous contacter au sujet du livre, nous envoyez vos commentaires, ou d'autres questions sur lesquelles vous aimeriez porter notre attention , n'hésitez pas.

Nom	Email	Twitter
The Majesty of Vue.js	hello@tmvuejs.com	@tmvuejs
Alex Kyriakidis	alex@tmvuejs.com	@hootlex
Kostas Maniatis	kostas@tmvuejs.com	@kostaskafcas

Pour toute remarque concernant la traduction, contacter Nacim Idjakirène par email à l'adresse idjakirene.n@gmail.com ou sur twitter (@n_nacim).

Exercices

La meilleure façon d'apprendre le code est d'écrire du code, nous avons donc préparé un exercice à résoudre à la fin de la plupart des chapitres, pour que vous puissiez vous tester par vous même sur ce que vous avez appris. Nous vous recommandons fortement d'essayer autant que possible de les résoudre afin d'avoir une meilleure compréhension de Vue.js. N'ayez pas peur de tester vos idées. Des efforts vous seront bénéfiques tout au long de l'aventure Vue.js ! Peut-être que quelques exemples ou façons de faire différents vous donneront la bonne idée. Bien sûr, des conseils et des solutions potentielles seront fournis !

Vous pouvez commencer le voyage !

Errata

Un bloc de code est défini comme ceci :

JavaScript

```
1 function(x, y){
2     // ceci est un commentaire
3 }
```

Le code au milieu du texte et les données sont représentés de la manière suivante : “Utilisez `.container` pour un conteneur responsive avec un width fixe.”

Les nouveaux termes et mots importants sont indiqués en gras.

Astuce, notes, avertissement sont indiqués comme suite :



Ceci est un avertissement

Cet élément indique un avertissement.



Ceci est une astuce

Cet élément représente une astuce ou une suggestion.



Ceci est une boite d'information

Ici, des informations spéciales.



Ceci est une note

Une note à propos du sujet.



Ceci est un conseil

Un conseil à propos du sujet.



Ceci est une ligne de commande

commandes à exécuter dans le terminal.



Ceci est un texte de comparaison

Un petit texte qui compare quelque chose par rapport au sujet.

Les fondamentaux de Vue.js

Installer Vue.js

Concernant le téléchargement de Vue.js, vous avez le choix entre plusieurs options :

Version Autonome (standalone)

Telecharger depuis vuejs.org

Pour installer Vue, vous pouvez tout simplement le télécharger et l'inclure avec une balise de type script. Vue sera enregistré comme variable globale.

Vous pouvez télécharger deux versions de Vue.js :

1.Version de développement depuis <http://vuejs.org/js/vue.js>⁶ 2.Version de production depuis <http://vuejs.org/js/vue.min.js>⁷.



Astuce : N'utilisez pas la version minifiée pendant le développement. Vous perdrez tous les super avertissements pour les erreurs courantes.

Inclure depuis un CDN

Vous pouvez aussi trouver Vue.js sur [jsdelivr](#)⁸ ou [cdnjs](#)⁹



Il faut un certain temps pour se synchroniser avec la dernière version, vous devez donc vérifier régulièrement les mises à jour.

Télécharger à l'aide de NPM

NPM est la méthode d'installation recommandée lors de la création d'applications à grande échelle avec Vue.js. Il s'accorde bien avec les module bundler CommonJS tel que [Webpack] (<http://webpack.github.io/>) ou [Browserify] (<http://browserify.org/>).

⁶<http://vuejs.org/js/vue.js>

⁷<http://vuejs.org/js/vue.min.js>

⁸<https://cdn.jsdelivr.net/vue/2.0.1/vue.min.js>

⁹<https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.min.js>

```
1 # Dernière version stable  
2 $ npm install vue  
3 # Dernière version stable + CSP-compliant  
4 $ npm install vue@csp  
5 # dev build (Directement depuis Github):  
6 $ npm install vuejs/vue#dev
```

Télécharger en utilisant Bower

```
1 # Dernière version stable  
2 $ bower install vue
```



Pour plus d'instructions d'installation et de mises à jour, consultez le [Guide d'installation Vue.js](#)¹⁰

Dans la plupart des exemples de livres, nous incluons Vue.js à partir d'un cdn, bien que vous soyez libre de l'installer en utilisant la méthode que vous préférez.

¹⁰<http://vuejs.org/guide/installation.html>

Mise en marche

Commençons par une visite rapide des fonctions de data binding de Vue.js. Nous allons faire une application simple qui nous permettra d'entrer un message et de le faire afficher sur la page en temps réel. Nous allons démontrer la puissance du data binding de Vue. Afin de créer notre application Vue, nous devons faire un peu de configuration, ce qui implique simplement la création d'une page HTML.

Dans ce processus, vous aurez une idée de la quantité de temps et d'efforts que nous économisons en utilisant un framework javascript Comme Vue.js au lieu d'un outil javascript (bibliothèque) comme jQuery.

Hello World

Nous allons créer un nouveau fichier HTML et nous allons y ajouter le code de base. Vous pouvez nommer votre fichier comme vous voulez, celui-ci est appelé `hello.html`.

```
1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6   <h1>Greetings your Majesty!</h1>
7 </body>
8 </html>
```

Il s'agit d'un simple fichier HTML avec un message d'accueil.

Maintenant, nous allons poursuivre et faire le même travail en utilisant Vue.js. Tout d'abord, nous allons inclure Vue.js et créer une nouvelle Instance.

```

1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6   <div id="app">
7     <h1>Greetings your majesty!</h1>
8   </div>
9 </body>
10 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.min.js"></script>
11 </script>
12 <script>
13   new Vue({
14     el: '#app',
15   })
16 </script>
17 </html>
```

Pour les débutants, nous avons inclus Vue.js à partir [cdnjs¹¹](https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.min.js) et dans un tag **script** nous avons notre instance Vue. Nous utilisons un **div** avec un **id** qui est **#app** qui est l'élément auquel nous faisons référence, de cette façon Vue.js sait où ‘Regarder’. Essayez de penser à cela comme un conteneur où Vue.js travaille. Vue ne reconnaîtra rien en dehors de l’élément ciblé. Utilisez l’option **el** pour cibler l’élément que vous voulez.

Maintenant, nous allons assigner le message que nous voulons afficher, à une variable à l’intérieur d’un objet nommé **data**. Ensuite, nous transmettrons l’objet de données en option au constructeur Vue.

```

1 var data = {
2   message: 'Greetings your majesty!'
3 };
4 new Vue({
5   el: '#app',
6   data: data
7 })
```

Pour afficher notre message sur la page, nous avons juste besoin d’envelopper le message entre double crochets. Donc, quel que soit l’intérieur de notre message, il apparaîtra automatiquement dans la balise **h1**.

¹¹<https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.min.js>

```
1 <div id="app">
2   <h1>{{ message }}</h1>
3 </div>
```

C'est aussi simple que ça. Une autre façon de définir la variable message est de le faire directement dans le constructeur Vue dans l'objet **data**.

```
1 new Vue({
2   el: '#app',
3   data: {
4     message: 'Greetings your Majesty!'
5   }
6});
```

Les deux méthodes ont exactement le même résultat, donc vous êtes de nouveau libre de choisir quelle syntaxe vous voulez.

Info

Les doubles accolades ne sont pas du code HTML, mais du code de scripting, toute expression à l'intérieur des balises moustaches est appelé * expression de binding *. Javascript va évaluer cette expression. le {{ message }} fait apparaître la valeur de la variable Javascript. Ce morceau de code {{1 + 2}} affiche le nombre 3.

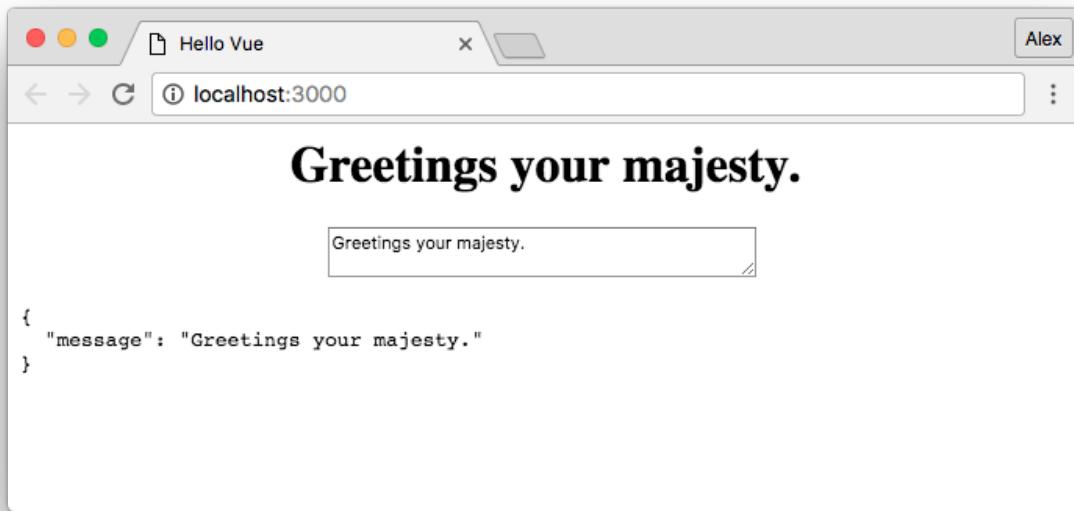
Data binding bidirectionnel

Ce qui est cool à propos de Vue.js, c'est qu'il facilite notre vie. Supposons que nous voulons changer le message affiché quand l'utilisateur entre un nouveau dans l'input. Comment pouvons-nous facilement l'accomplir ? Dans l'exemple ci-dessous nous utilisons **v-model**, une directive de Vue.js (nous aborderons plus en détail les directives dans le chapitre suivant). Ensuite, nous utilisons le data binding bidirectionnel pour modifier dynamiquement la valeur du message lorsque l'utilisateur change le texte du message à l'intérieur d'un input. Par défaut, les données sont synchronisées sur chaque événement d'entrée.

```
1 <div id="app">
2   <h1>{{ message }}</h1>
3   <input v-model="message">
4 </div>
```

```
1 new Vue({  
2   el: '#app',  
3   data: {  
4     message: 'Greetings your Majesty!'  
5   }  
6 })
```

* C'est tout. *** Maintenant, notre message et l'entrée de l'utilisateur sont liés ! En utilisant **v-model à l'intérieur du tag `input`, nous indiquons à Vue quelle variable doit être liée avec cet `input`, dans ce cas `message`.



Data binding bidirectionnel

Le Data binding bidirectionnel signifie que si vous modifiez la valeur d'un modèle dans votre view, tout sera maintenu à jour.

Comparaison avec jQuery

Vous avez probablement tous une certaine expérience avec jQuery. Si ce n'est pas le cas, aucun problème, l'utilisation de jQuery dans ce livre est minime. Quand nous l'utilisons, c'est seulement pour démontrer comment les choses peuvent être faites avec Vue.js au lieu de jQuery et nous ferons en sorte que tout le monde comprenne.

Quoi qu'il en soit, afin de mieux comprendre comment le data binding nous aide à créer des applications, Prenez un moment et pensez comment vous pourriez refaire l'exemple précédent en

utilisant jQuery. Vous pourriez probablement créer un élément input et lui donner un **id** ou une **class**, pour pouvoir le cibler et le modifier en conséquence. Après cela, vous pourriez appeler une fonction qui change l'élément souhaité pour correspondre à la valeur d'entrée, chaque fois que *l'événement keyup se produit. C'est très dérangeant.*

Votre bout de code ressemblerait plus ou moins à ceci.

```
1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6 <div id="app">
7   <h1>Greetings your Majesty!</h1>
8   <input id="message">
9 </div>
10 </body>
11 <script src="https://code.jquery.com/jquery-2.1.4.min.js"></script>
12 <script type="text/javascript">
13   $('#message').on('keyup', function(){
14     var message = $('#message').val();
15     $('h1').text(message);
16   })
17 </script>
18 </html>
```

C'est un exemple simple de comparaison et, comme vous pouvez le voir, Vue.js semble être beaucoup plus élégant, consomme moins de temps et est plus facile à saisir. Bien sûr, jQuery est une puissante bibliothèque JavaScript pour la manipulation DOM (Document Object Model), mais tout a ses avantages et ses inconvénients !



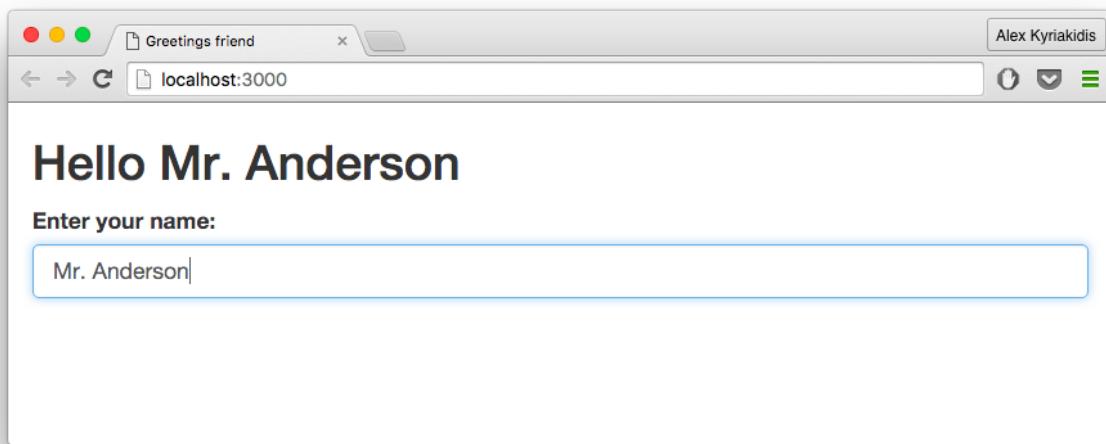
Codes d'exemples

Vous trouverez les exemples de code de ce chapitre sur [GitHub¹²](#).

¹²<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/codes/chapter2.html>

Devoirs

Un exercice d'introduction simple est de créer un fichier HTML avec un en-tête Hello, {{name}}. Ajoutez un input et liez-le à la variable `name`. Comme vous pouvez l'imaginer, le titre doit changer instantanément chaque fois que l'utilisateur tape ou change son nom. Bonne chance et amusez-vous bien !



Exemple de résultat



Note

L'exemple affiché fait l'usage de Bootstrap. Si vous n'êtes pas familier avec Bootstrap, vous pouvez l'ignorer pour le moment. Il est couvert dans un [Chapitre ultérieur](#).



Solution potentielle

Vous trouverez une solution potentielle à cet exercice [ici¹³](#).

¹³<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter2.html>

Une saveur de directives

Dans ce chapitre, nous examinerons quelques exemples de base des directives de Vue.js. Bien, Si vous n'avez pas utilisé un Framework comme Vue.js ou Angular.js avant, vous ne savez probablement pas ce qu'est une directive. Une directive est essentiellement un jeton spécial dans le balisage HTML qui indique à la bibliothèque de faire quelque chose à un élément du DOM. Dans Vue.js, le concept de directive est drastiquement plus simple que celui d'Angular. Certaines des directives sont les suivantes :

- **v-show** qui est utilisé pour afficher conditionnellement un élément
- **v-if** qui peut être utilisé à la place de **v-show**
- **v-else** qui affiche un élément lorsque **v-if** est évalué à false.

En outre, il existe **v-for**, qui nécessite une syntaxe spéciale et son utilisation est pour le rendu (Par exemple, rendre une liste d'éléments basée sur un tableau). Nous allons développer sur l'utilisation de chacun, plus tard dans ce livre.

Commençons par examiner les directives que nous avons mentionnées plus haut.

v-show

Pour faire une démonstration de la première directive, nous allons construire quelque chose de simple. Nous vous donnerons quelques conseils qui faciliteront votre compréhension et votre travail ! Supposons que vous vous trouviez dans le besoin de basculer l'affichage d'un élément, en fonction d'un ensemble de critères. Peut-être un bouton de soumission ne devrait pas être affiché à moins que vous n'ayez déjà tapé dans un message. Comment pouvons-nous accomplir cela avec Vue ?

```
1 <html>
2   <head>
3     <title>Hello Vue</title>
4   </head>
5   <body>
6     <div id="app">
7       <textarea></textarea>
8     </div>
9   </body>
10  <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
11  <script>
```

```

12  new Vue({
13      el: '#app',
14      data: {
15          message: 'Our king is dead!'
16      }
17  })
18 </script>
19 </html>

```

Ici, nous avons un fichier HTML avec notre fameuse `div id = " app "` et un `textarea`. À l'intérieur de la `textarea`, nous allons afficher notre message. Bien sûr, il n'est pas encore lié et vous l'avez peut-être déjà compris. Aussi, vous avez peut-être remarqué que dans cet exemple, nous n'utilisons plus la version minifiée de Vue.js. Comme nous l'avons mentionné précédemment, la version minifiée ne devrait pas être utilisée pendant le développement parce que vous manquerez des avertissements pour les erreurs courantes. A partir de maintenant, nous allons utiliser cette version dans le livre, mais bien sûr, vous êtes libre de faire ce que vous voulez.

```

1 <html>
2 <head>
3     <title>Hello Vue</title>
4 </head>
5 <body>
6 <div id="app">
7     <textarea v-model="message"></textarea>
8     <pre>
9         {{ $data }}
10    </pre>
11 </div>
12 </body>
13 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
14 <script>
15     new Vue({
16         el: '#app',
17         data: {
18             message: 'Our king is dead!'
19         }
20     })
21 </script>
22 </html>

```

Il est temps de lier la valeur de `textarea` avec notre variable `message` en utilisant `v-model` afin qu'il affiche notre message. Tout ce que nous saisissons va changer en temps réel, comme nous l'avons vu

dans l'exemple du chapitre précédent, Où nous utilisions un élément input. En outre, nous utilisons ici une balise `pre` pour cracher les données. Ce que cela va faire, c'est de prendre les données de notre exemple Vue, Le filtrer par `json`, et enfin afficher les données dans notre navigateur. La sortie de Vue.js sera automatiquement et joliement formaté que ce soit une chaîne, un nombre, un tableau, ou un objet simple. Cela donne une bien meilleure façon de construire et de manipuler nos données, Puisque avoir tout devant vous est préférable au fait de regarder constamment votre console.

Info

JSON (JavaScript Object Notation) est un format léger d'échange de données. Vous pouvez trouver plus d'informations sur JSON [ici¹⁴](#). Le résultat de `{{ $data }}` est lié avec les Data de l'objet Vue et sera mis à jour à chaque changement.

```

1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6 <div id="app">
7   <h1>You must send a message for help!</h1>
8   <textarea v-model="message"></textarea>
9   <button v-show="message">
10     Send word to allies for help!
11   </button>
12   <pre>
13     {{ $data }}
14   </pre>
15 </div>
16 </body>
17 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
18 <script>
19   new Vue({
20     el: '#app',
21     data: {
22       message: 'Our king is dead! Send help!'
23     }
24   })
25 </script>
26 </html>
```

¹⁴<http://www.json.org/>

En continuant, nous avons maintenant un simple avertissement dans la balise **h1** qui basculera plus tard sur la base de certains critères. A côté, il ya le bouton qui va s'afficher conditionnellement. Il apparaît uniquement si un message est présent. Si **textarea** est vide et donc sans données, l'attribut **display** du bouton est automatiquement Réglé sur “none” et le bouton disparaît.



Info

Un élément avec **v-show** sera toujours rendu et restera dans le DOM. **v-show** bascule simplement l propriété CSS **display** de l'élément.

```

1 <h1 v-show="!message">You must send a message for help!</h1>
2 <textarea v-model="message"></textarea>
3 <button v-show="message">
4     Send word to allies for help!
5 </button>
```

Ce que nous voulons accomplir dans cet exemple, est de basculer les différents éléments. Dans cette étape, nous devons masquer l'avertissement dans la balise **h1**, si un message est présent. Dans le cas contraire il faudra masquer le message en définissant son **style** à **display:none**.

v-if

À ce stade, vous pourriez vous demander « Qu'en est-il de la directive **v-if** que nous avons mentionnée plus tôt ? ». Nous allons donc reconstruire l'exemple précédent, mais cette fois nous allons utiliser **v-if** !

```

1 <html>
2 <head>
3     <title>Hello Vue</title>
4 </head>
5 <body>
6 <div id="app">
7     <h1 v-if="!message">You must send a message for help!</h1>
8     <textarea v-model="message"></textarea>
9     <button v-if="message">
10        Send word to allies for help!
11     </button>
12     <pre>
13         {{ $data }}
14     </pre>
```

```

15  </div>
16  </body>
17  <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
18  <script>
19      new Vue({
20          el: '#app',
21          data: {
22              message: 'Our king is dead! Send help!'
23          }
24      })
25  </script>
26  </html>

```

Comme indiqué, le remplacement de `v-show` par `v-if` fonctionne aussi bien que nous le pensions. Allez-y et essayez de faire vos propres expériences pour voir comment cela fonctionne ! La seule différence est qu'un élément avec `v-if` ne restera pas dans le DOM.

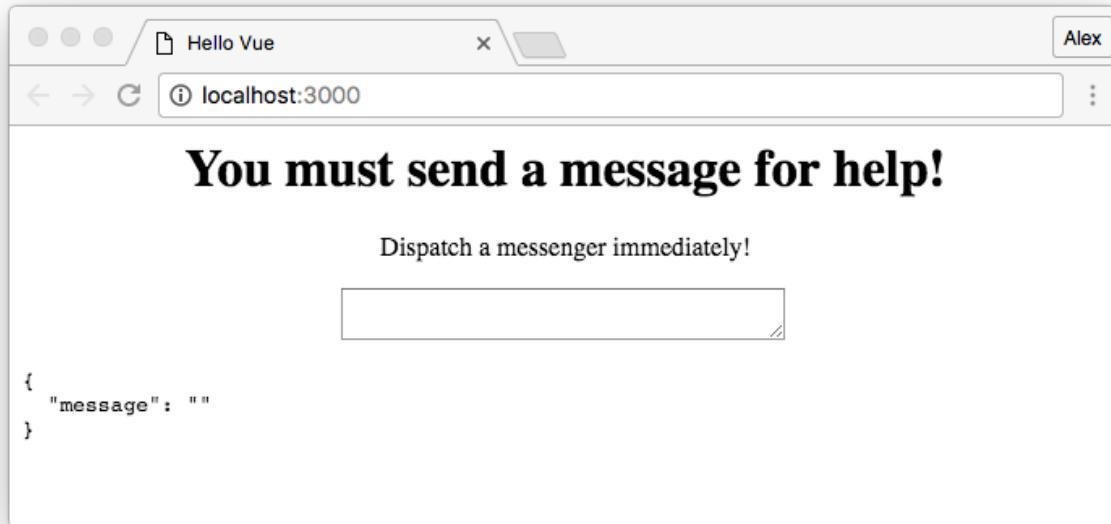
Template v-if

Si nous nous trouvons parfois dans une situation où nous voulons changer l'existence de Plusieurs éléments à la fois, nous pouvons utiliser `v-if` sur un élément `<template>`. Dans les cas où l'utilisation de `div` ou `span` ne semble pas appropriée, l'élément `<template>` peut également servir dde wrapper invisible. L'élément `<template>` ne sera pas rendu dans le résultat final.

```

1  <div id="app">
2      <template v-if="!message">
3          <h1>You must send a message for help!</h1>
4          <p>Dispatch a messenger immediately!</p>
5          <p>To nearby kingdom of Hearts!</p>
6      </template>
7      <textarea v-model="message"></textarea>
8      <button v-show="message">
9          Send word to allies for help!
10     </button>
11     <pre>
12         {{ $data }}
13     </pre>
14  </div>

```



Template v-if

En utilisant la configuration de l'exemple précédent, nous avons joint la directive **v-if** à l'élément **template**, ce qui affiche conditionnellement tous les éléments présents de le template.



Avertissement

La directive **v-show** ne supporte pas la syntaxe `<template>`.

v-else

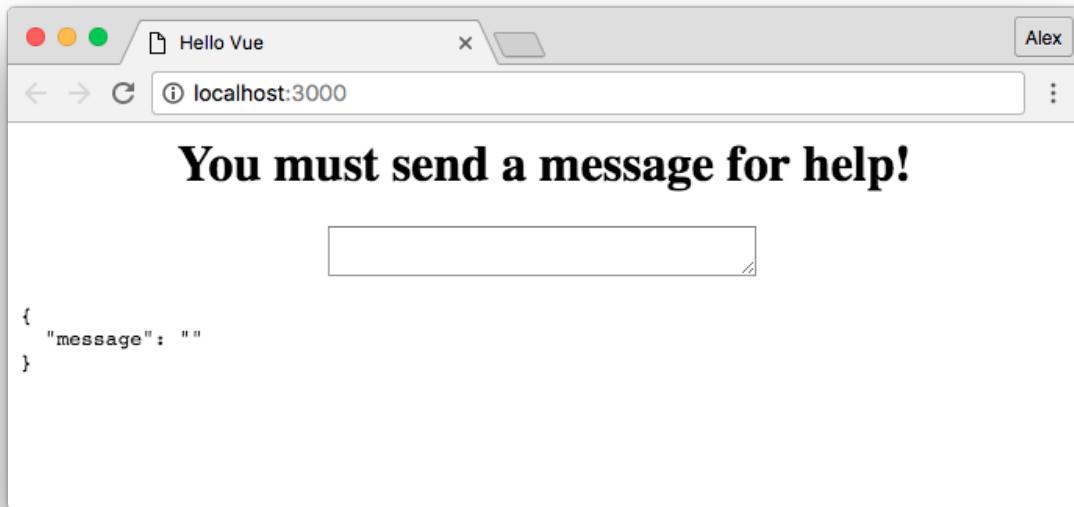
indiquer un bloc “else” Comme vous l’auriez pu imaginer. Notez que la directive **v-else** doit suivre immédiatement la directive **v-if** -sinon elle ne sera pas reconnue.

```

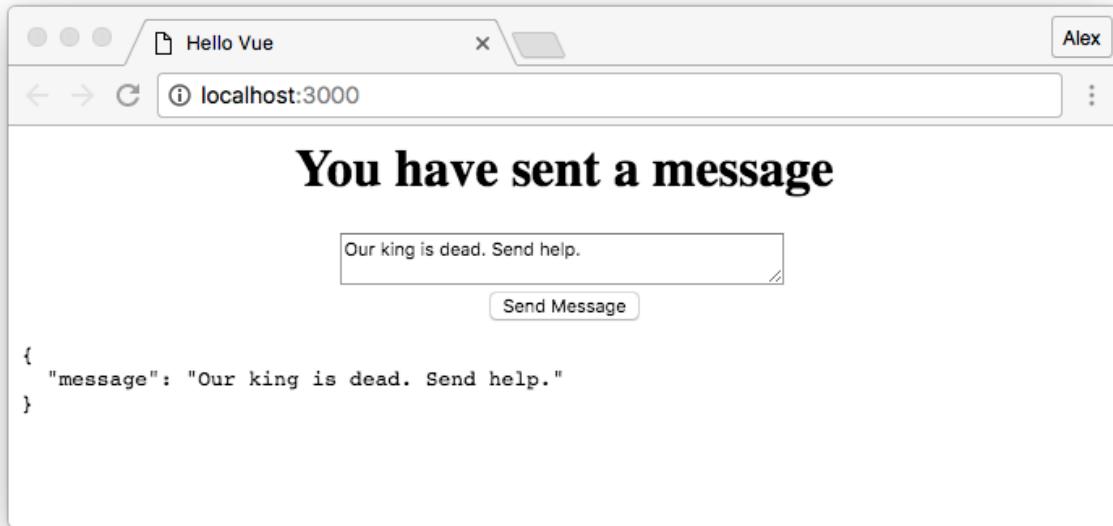
1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6 <div id="app">
7   <h1 v-if="!message">You must send a message for help!</h1>
8   <h2 v-else>You have sent a message!</h2>
9   <textarea v-model="message"></textarea>

```

```
10   <button v-show="message">
11     Send word to allies for help!
12   </button>
13   <pre>
14     {{ $data }}
15   </pre>
16 </div>
17 </body>
18 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
19 <script>
20   new Vue({
21     el: '#app',
22     data: {
23       message: 'Our king is dead! Send help!'
24     }
25   })
26 </script>
27 </html>
```



v-if in action



v-else in action

Pour l'exemple, nous avons utilisé une balise **h2** avec un avertissement différent, ce qui est affiché conditionnellement. Si aucun message n'est présenté, nous voyons le contenu de la balise **h1**. S'il ya un message, nous voyons le contenu de la balise **h2** en utilisant cette syntaxe très simple de Vue **v-if** et **v-else**. Simple comme un bouton !



Avertissement

La directive **v-show** ne fonctionne plus avec **v-else'**, dans Vue 2.0.

v-if vs. v-show

Même si nous avons déjà mentionné une différence entre **v-if** et **v-show**, nous pouvons approfondir un peu plus le sujet. Certaines questions peuvent découler de leur utilisation. Y a-t-il une grande différence entre l'utilisation de **v-show** et **v-if**? Existe-t-il une situation où la performance est affectée? Y a-t-il des problèmes où il vaut mieux utiliser l'un ou l'autre? Vous pouvez constater que l'utilisation de **v-show** sur un grand nombre de situations entraîne un temps de chargement plus important pendant le rendu de page. En comparaison, **v-if** est vraiment conditionnel selon le guide de Vue.js.

*Lorsque vous utilisez **v-if**, si la condition est false sur le rendu initial, Il ne fera rien - - le bloc conditionnel ne sera pas rendu jusqu'à ce que la condition devienne vraie pour la première fois. D'une manière générale, **v-if** a des coûts de basculement (toggle) plus*

élevés alors que `v-show` a des coûts de rendu initiaux plus élevés. Alors préférez `v-show` si vous devez faire basculer quelque chose très souvent, et préférez `v-if` si la condition est peu susceptible de changer à l'exécution.

Ainsi, l'utilisation de l'un ou l'autre dépend vraiment de vos besoins.



Exemples de code

Vous trouverez les exemples de code de ce chapitre sur [GitHub](#)¹⁵.

¹⁵<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter3>

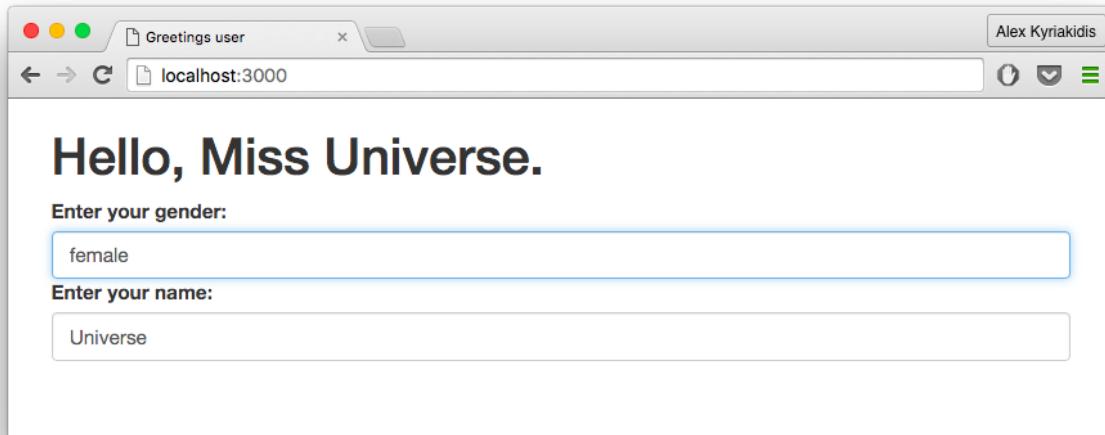
Devoirs

Après l'exercice du devoir précédent, vous devriez essayer de l'élargir un peu. L'utilisateur tape maintenant son genre avec son nom. Si l'utilisateur est un homme, alors l'en-tête saluera l'utilisateur avec “**Bonjour monsieur {{nom}}**”. Si l'utilisateur est une femme, alors “**Hello Miss {{name}}**” devrait apparaître à la place.

Lorsque le sexe n'est ni masculin ni féminin, l'utilisateur doit voir l'en-tête d'avertissement “**Vous ne pouvez donc pas décider!**”.

Conseil

Un opérateur logique serait utile pour déterminer le titre de l'utilisateur.



Example Output



Solution potentielle

Vous pouvez trouver une solution potentielle à cet exercice [ici](#)¹⁶.

¹⁶<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter3.html>

Rendu des listes

Dans le quatrième chapitre de ce livre, nous allons parler du rendu d'une liste. En utilisant les directives de Vue, nous allons apprendre comment :

1. Rendre une liste d'éléments basée sur un tableau.
2. Répétez un modèle.
3. Iterer à travers les propriétés d'un objet.

Install & Use Bootstrap

Pour rendre notre travail plus lisible, nous allons importer Bootstrap.



Information

Bootstrap est le framework HTML, CSS et JS le plus populaire pour développer des applications / sites responsives et mobiles.

Rendez-vous sur <http://getbootstrap.com/>¹⁷ et cliquez sur le bouton de téléchargement. Pour l'instant, nous allons simplement utiliser Bootstrap à partir du [lien CDN](#)¹⁸ mais vous pouvez l'installer de n'importe quelle manière qui convient à vos besoins. Pour notre exemple, nous n'avons besoin que d'un seul fichier, pour l'instant : `css/bootstrap.min.css`. Lorsque nous utilisons ce fichier `.css` dans notre application, nous avons accès à toutes les jolies structures et styles de Bootstrap. Il suffit de l'inclure dans la balise `head` de votre page et vous êtes prêt.

Bootstrap nécessite un élément **conteneur** pour envelopper le contenu de notre site et abriter notre système de grille. Vous pouvez choisir l'un des deux conteneurs à utiliser dans vos projets. Notez qu'en raison du `padding` et autre, aucun des conteneurs ne peut être emboîté.

- Utilisez `.container` pour un conteneur possédant un fixe et responsive.

```
1  <div class="container">
2  ...
3  </div>
```

- Utilisez `.container-fluid` pour un conteneur couvrant toute la largeur de votre fenêtre.

¹⁷<http://getbootstrap.com/>

¹⁸<https://www.bootstrapcdn.com/>

```

1 <div class="container-fluid">
2 ...
3 </div>
```

À ce stade, nous aimerais faire un exemple de Vue.js avec les classes Bootstrap. Bien entendu, il n'est pas nécessaire de faire beaucoup de recherche pour utiliser Vue et Bootstrap combinés.

```

1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4 <title>Hello Bootstrap</title>
5 </head>
6 <body>
7   <div class="container">
8     <h1>Hello Bootstrap, sit next to Vue.</h1>
9   </div>
10 </body>
11 <script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js"></script>
12 <script type="text/javascript">
13   new Vue({
14     el: '.container'
15   })
16 </script>
17 </html>
```

Notez cette fois, qu'au lieu de cibler l'id de `app`, nous avons ciblé la classe `container` dans l'option `el` dans l'instance de Vue.



Astuce

Dans l'exemple ci-dessus, nous ciblons l'élément avec la classe `.container`. Faites attention lorsque vous ciblez un élément par classe, lorsque la classe est présente plus d'une fois, Vue.js va être monté sur le premier élément seulement.

La propriété `el:` peut être un sélecteur CSS ou un élément HTML réel. *Il n'est pas recommandé de monter l'instance racine sur <html> ou <body>.*

v-for

Afin de parcourir chaque élément d'un tableau, nous utiliserons la directive **v-for**.

Cette directive nécessite une syntaxe spéciale sous la forme de **item in array** où **array** est le tableau source et **item** est un alias pour l'élément inséré du tableau.



Avertissement

Si vous venez du monde php, vous remarquerez peut-être que **v-for** est similaire à la fonction **foreach** de php. Mais faites **attention** si vous êtes habitué à faire des **foreach(\$array as \$value)**.

v-for de Vue.js est fait exactement le contraire, **value in array**.

Le singulier en premier, le pluriel ensuite.

v-for avec interval

La directive **v-for** peut également prendre un entier. Chaque fois qu'un nombre est passé au lieu d'un tableau/objet, le template sera répété autant de fois que le nombre donné.

```

1 <html>
2 <head>
3   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.mi\
4 n.css" rel="stylesheet">
5   <title>Hello Vue</title>
6 </head>
7 <body>
8   <div class="container">
9     <h1>The multiplication table of 4.</h1>
10    <ul class="list-group">
11      <li v-for="i in 11" class="list-group-item">
12        {{ i-1 }} times 4 equals {{ (i-1) * 4 }}.
13      </li>
14    </ul>
15  </div>
16 </body>
17 <script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js"></script>
18 <script type="text/javascript">
19   new Vue({
20     el: '.container'
21   })

```

```
22  </script>
23  </html>
```

Le code ci-dessus affiche la table de multiplication de 4.

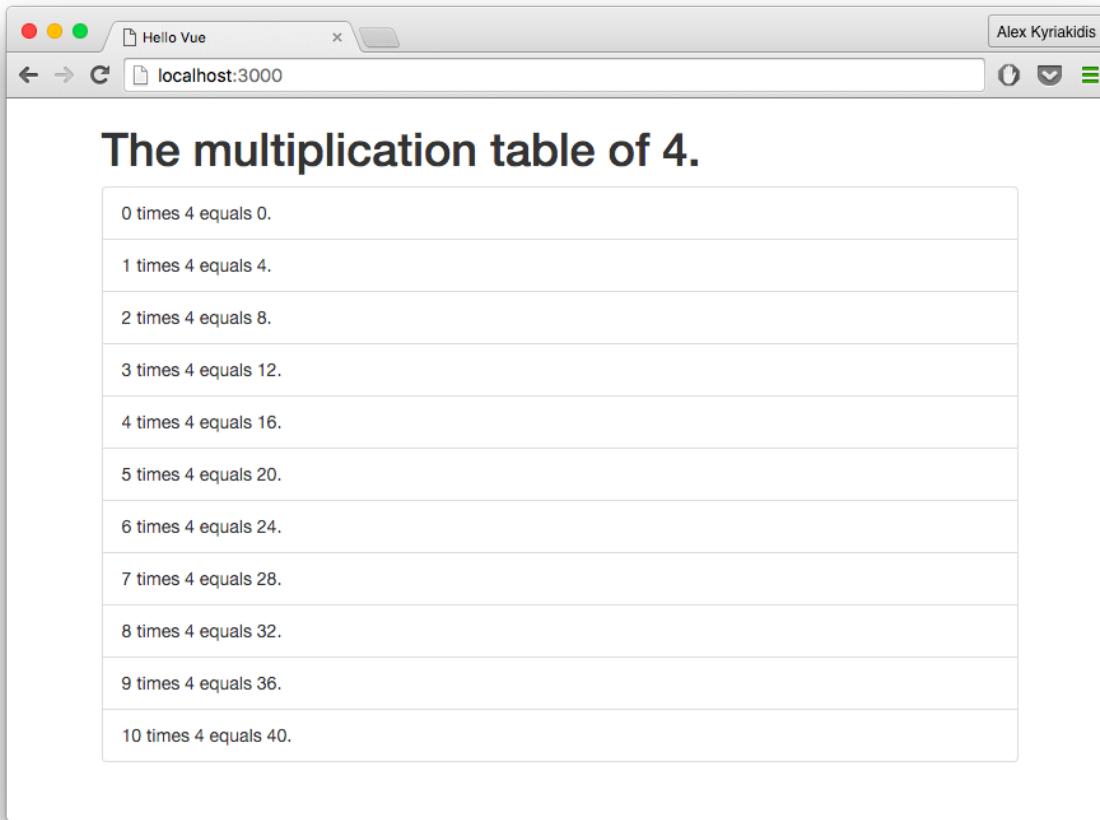


Table de multiplication de 4



Remarque

Parce que nous voulons afficher toute la table de multiplication de 4 (jusqu'à 40) nous répétons le template 11 fois puisque la première valeur que prend `i` est 1.

Rendu des tableaux

Boucle à travers un tableau

Dans l'exemple suivant, nous mettrons en place le tableau d'histoires suivant dans notre objet de données et nous les afficherons tous, un par un.

```
stories: [
    "I crashed my car today!",
    "Yesterday, someone stole my bag!",
    "Someone ate my chocolate...",
]
```

Ce que nous devons faire ici, est d'afficher le rendu d'une liste. Plus précisément, un tableau d'éléments string.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5     <title>Stories</title>
6 </head>
7 <body>
8     <div class="container">
9         <h1>Let's hear some stories!</h1>
10        <div>
11            <ul class="list-group">
12                <li v-for="story in stories" class="list-group-item">
13                    Someone said "{{ story }}"
14                </li>
15            </ul>
16        </div>
17        <pre>
18            {{ $data }}
19        </pre>
20    </div>
21 </body>
22 <script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js"></script>
23 <script type="text/javascript">
24     new Vue({
25         el: '.container',
```

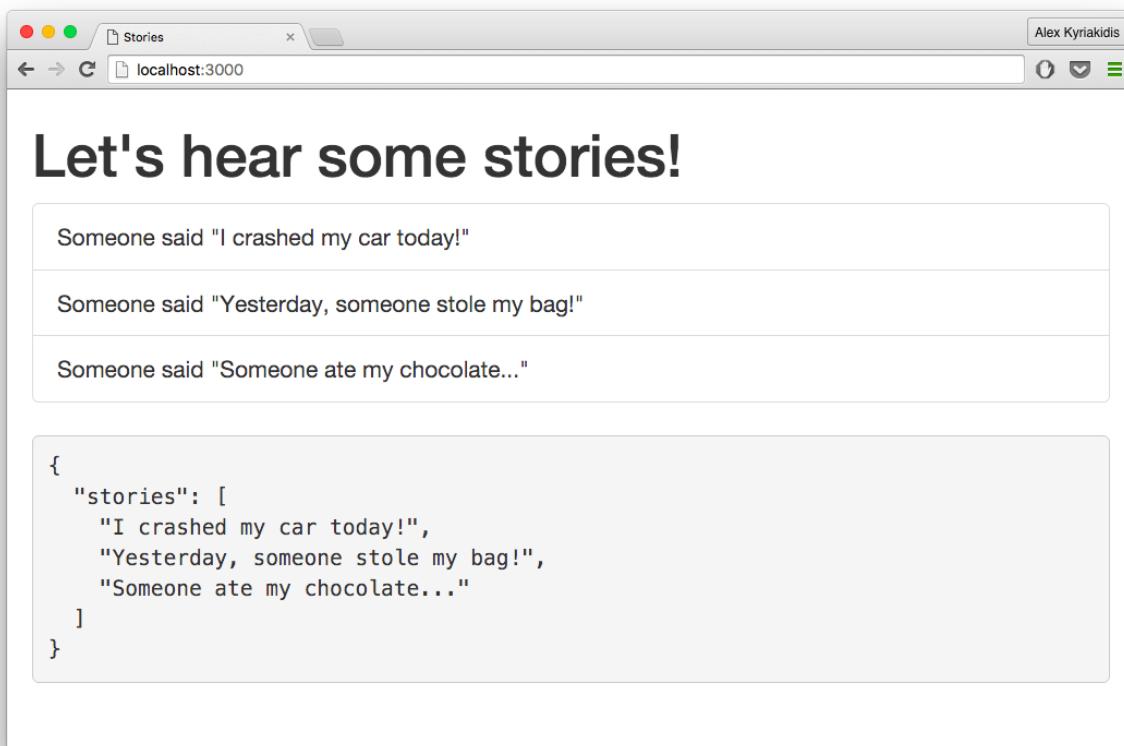
```

26     data: {
27       stories: [
28         "I crashed my car today!",
29         "Yesterday, someone stole my bag!",
30         "Someone ate my chocolate...",
31       ]
32     }
33   })
34 </script>
35 </html>
```



Information

Les classes `list-group` et `list-group-item` sont des classes Bootstrap. Vous trouverez ici plus d'informations sur le style des listes Bootstrap.¹⁹



Rendu d'un tableau en utilisant v-for.

¹⁹<http://getbootstrap.com/css/#type-lists>

En utilisant `v-for`, nous avons réussi à afficher nos histoires dans une simple liste non ordonnée. C'est vraiment si facile que ça !

Boucle à travers un tableau d'objets

Maintenant, nous modifions le tableau `Stories` pour contenir des objets `story`. Un objet `story` a 2 propriétés : `plot` et `writer`. Nous ferons la même chose que nous avons fait avant, mais cette fois au lieu d'afficher `story` immédiatement, nous allons afficher `story.plot` et `story.writer` respectivement.

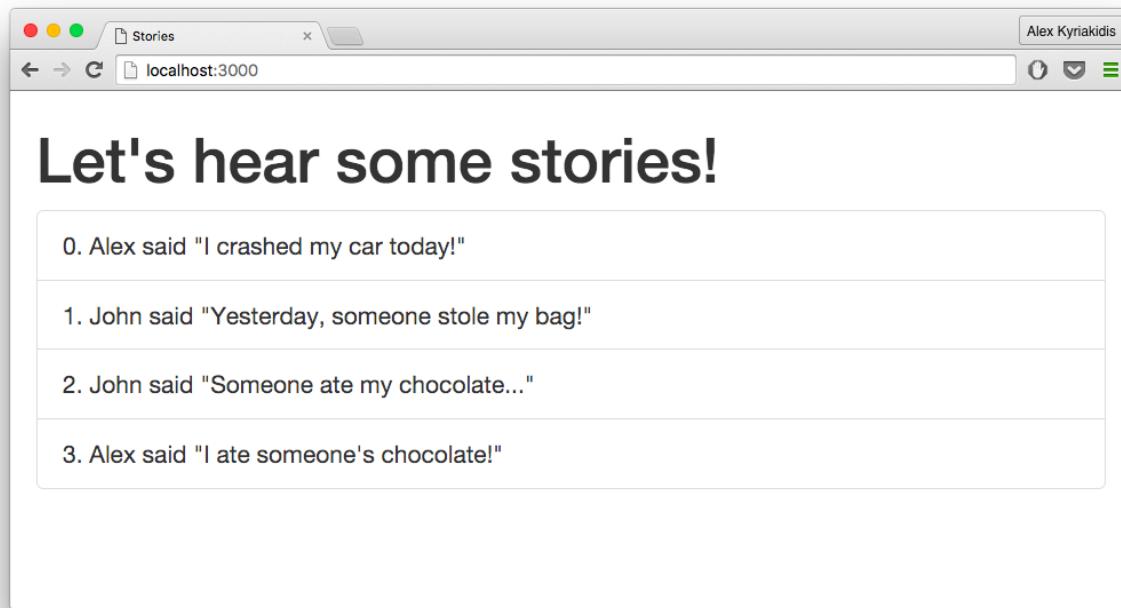
```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5     <title>Stories</title>
6 </head>
7 <body>
8     <div class="container">
9         <h1>Let's hear some stories!</h1>
10        <div>
11            <ul class="list-group">
12                <li v-for="story in stories"
13                    class="list-group-item">
14                >
15                    {{ story.writer }} said "{{ story.plot }}"
16                </li>
17            </ul>
18        </div>
19        <pre>
20            {{ $data }}
21        </pre>
22    </div>
23 </body>
24 <script src="https://cdn.jsdelivr.net/npm/vue@2.0.1/dist/vue.js"></script>
25 <script type="text/javascript">
26 new Vue({
27     el: '.container',
28     data: {
29         stories: [
30             {
31                 plot: "I crashed my car today!",
32                 writer: "Alex"
```

```
33      },
34      {
35          plot: "Yesterday, someone stole my bag!",
36          writer: "John"
37      },
38      {
39          plot: "Someone ate my chocolate...",
40          writer: "John"
41      },
42      {
43          plot: "I ate someone's chocolate!",
44          writer: "Alex"
45      },
46  ]
47 }
48 })
49 </script>
50 </html>
```

En outre, lorsque vous devez afficher l'index de l'élément en court, vous pouvez utiliser la variable spéciale `index`. Elle fonctionne comme ceci :

```
<ul class="list-group">
  <li v-for="(story, index) in stories"
      class="list-group-item" >
    {{index}} {{ story.writer }} said "{{ story.plot }}"
  </li>
</ul>
```

L'`index` à l'intérieur des accolades, représente clairement l'indice de l'élément itéré dans l'exemple donné.



Rendu de tableau avec index

v-for pour les objets

Vous pouvez utiliser `v-for` pour parcourir les propriétés d'un objet. Nous avons mentionné précédemment que vous pouvez utiliser l'`index` du tableau, mais vous pouvez également faire de même lors de l'itération sur objet. En plus de `index`, chaque scope d'itération aura accès à une autre propriété spéciale, `key`.

Information

Lors de l'itération d'un objet, `index` est dans l'intervalle `0 ... n-1` où `n` est le nombre de propriétés de l'objet.

Nous avons restructuré nos données pour qu'elles soient un objet unique avec 3 attributs cette fois : `plot`, `writer` et `upvotes`.

```
<div class="container">
  <h1>Let's hear some stories!</h1>
  <ul class="list-group">
    <li v-for="value in story" class="list-group-item">
      {{ value }}
    </li>
  </ul>
</div>
```

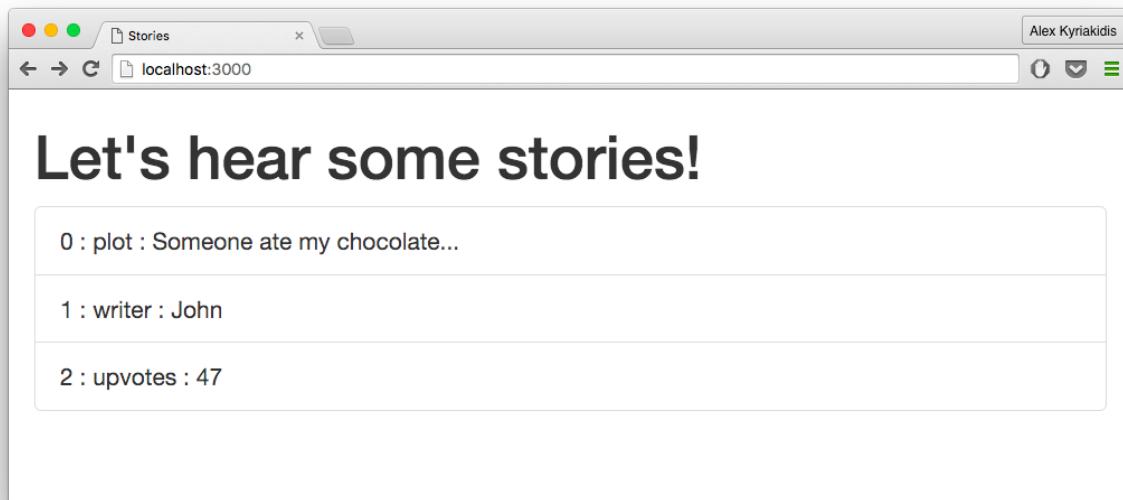
```
new Vue({
  el: '.container',
  data: {
    story: {
      plot: "Someone ate my chocolate...",
      writer: 'John',
      upvotes: 47
    }
  }
})
```

Nous pouvons fournir un deuxième et un troisième argument, pour `key` et `index` respectivement.

```
1 <div class="container">
2   <h1>Let's hear some stories!</h1>
3   <ul class="list-group">
4     <li v-for="(value, key, index) in story"
5       class="list-group-item"
6       >
7       {{index}} : {{key}} : {{value}}
8     </li>
9   </ul>
10 </div>
```

Comme vous pouvez le voir dans l'exemple ci-dessus, nous utilisons **key** et **index** pour introduire dans la liste les paires clé-valeur, ainsi que l'**index** de chaque paire.

Le résultat sera :



iterer à travers les propriétés de l'objet.



Exemple de code

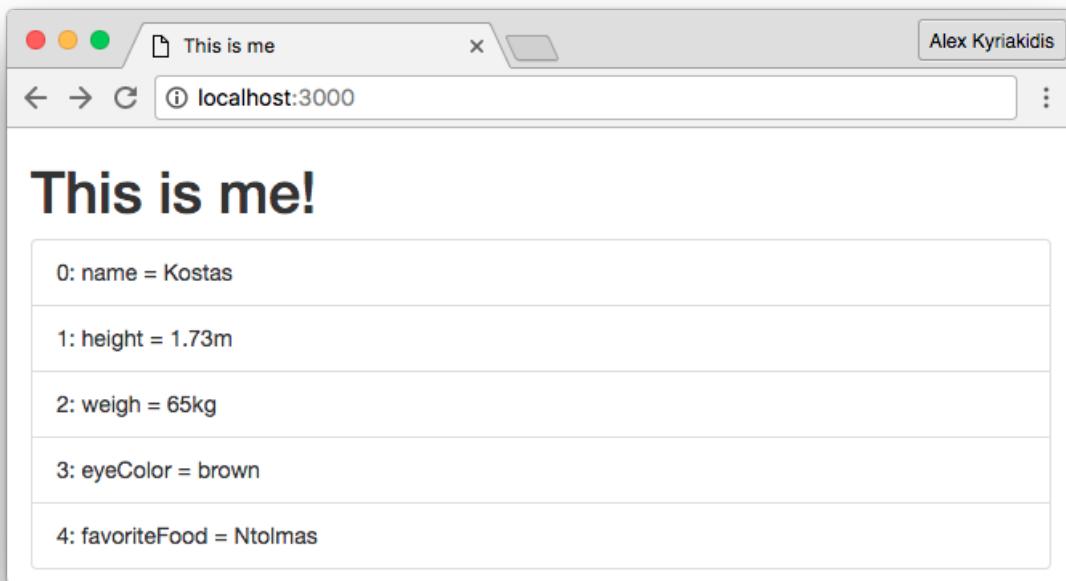
Vous trouverez les exemples de code de ce chapitre sur [GitHub](#)²⁰.

²⁰<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter4>

devoirs

Gardez à l'esprit ce que nous avons examiné dans ce chapitre, pour ce devoir, créez un objet avec vos attributs personnels. Par attributs personnels, je veux dire vos nom,poids,hauteur,eyeColor, et votre platFavori.

En utilisant `v-for`, itérerer dans chaque propriété et l'afficher en format : `index: key=value`.



Exemple



Solution potentielle

Vous pouvez trouver une solution à cet exercice [ici](#)²¹.

²¹<https://github.com/hoottlex/the-majesty-of-vuejs-2/blob/master/homework/chapter4.html>

Interactivité

Dans ce chapitre, nous allons aller plus loin et améliorer les exemples précédents, apprendre de nouvelles choses concernant les « méthodes », la « gestion des événements » et les « propriétés calculées » ou computed properties. Nous allons développer quelques exemples en utilisant différentes approches. Il est temps de voir comment nous pouvons mettre en œuvre l'interactivité de Vue pour obtenir une petite application, comme une calculatrice, agréable et facile d'utilisation.

Gestion des événements

Les événements HTML sont des choses qui arrivent aux éléments du DOM. Lorsque Vue.js est utilisé dans les pages HTML, il peut **réagir** à ces événements.

Les événements peuvent représenter un tas de choses, depuis les interactions utilisateur de base, jusqu'à ce qui se passe dans le modèle de rendu.

Voici quelques exemples d'événements HTML :

- Une page Web a fini de se charger.
- Un champ de saisie (input) a été modifié.
- Un bouton cliqué.
- Un formulaire a été soumis.

Le point de gestion des événements est que vous pouvez faire quelque chose à chaque fois qu'un événement a lieu.

Dans Vue.js, pour **Ecouter** les événements du DOM, vous pouvez utiliser la directive **v-on**.

La directive **v-on** attache un écouteur d'événement à un élément. Le type de l'événement est noté par l'argument, par exemple **v-on:keyup** écoute l'événement **keyup**.



Information

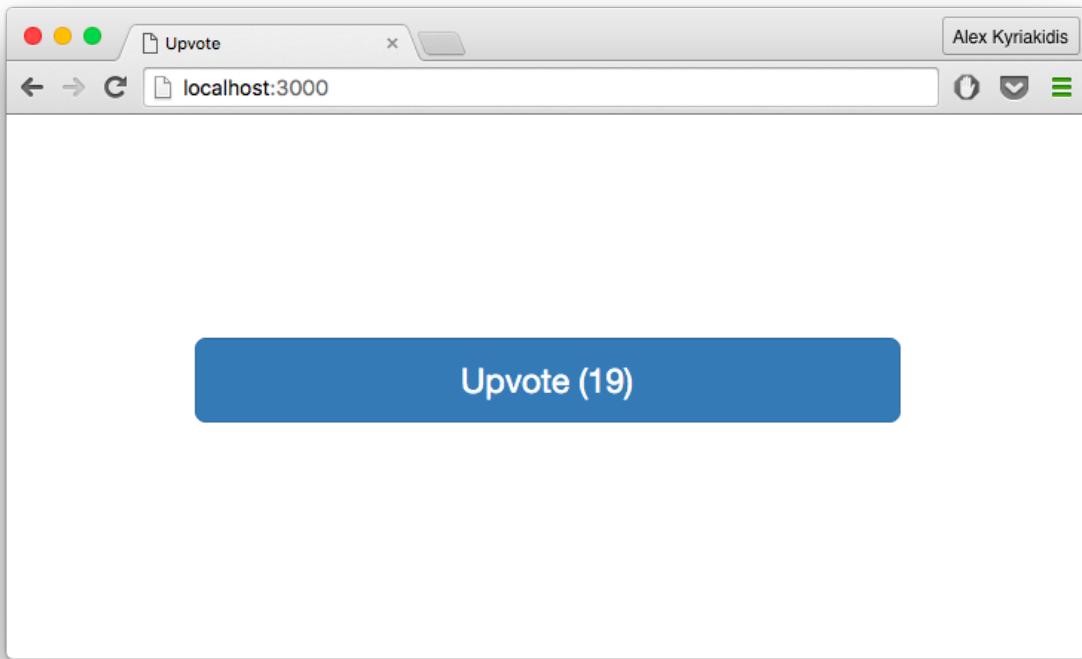
L'événement **keyup** se produit lorsque l'utilisateur relâche une touche. Vous trouverez une liste complète des événements HTML [ici²²](#).

Gestion des événements inline

Assez parlé, allons voir la gestion des événements en action. Ci-dessous, un bouton « Upvote » qui augmente le nombre de votes chaque fois qu'il est cliqué.

²²http://www.w3schools.com/tags/ref_eventattributes.asp

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5 <title>Upvote</title>
6 </head>
7 <body>
8     <div class="container">
9         <button v-on:click="upvotes++">
10            Upvote! {{upvotes}}
11        </button>
12    </div>
13 </body>
14 <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/vue/2\
15 .0.1/vue.js"></script>
16 <script type="text/javascript">
17 new Vue({
18     el: '.container',
19     data: {
20         upvotes: 0
21     }
22 })
23 </script>
24 </html>
```



Conteur de votes

Il y a une variable `upvotes` dans notre data. Dans ce cas, nous lions un écouteur d'événements pour `click`, avec l'instruction qui est juste à côté. À l'intérieur des guillemets, chaque fois que le bouton est pressé, nous augmentons simplement le nombre de upvotes par un, en utilisant l'opérateur d'incrément (** `upvotes ++`**).

Manipulation d'événements à l'aide de méthodes

Maintenant, nous allons faire exactement la même chose que précédemment, en utilisant une méthode à la place. Une méthode dans Vue.js est un bloc de code conçu pour effectuer une tâche particulière. Pour exécuter une méthode, vous devez la définir, puis l'appeler.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5 <title>Upvote</title>
6 </head>
7 <body>
8     <div class="container">
9         <button v-on:click="upvote">
10            Upvote! {{upvotes}}
11        </button>
12    </div>
13 </body>
14 <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/vue/2\.
15 .0.1/vue.js"></script>
16 <script type="text/javascript">
17 new Vue({
18     el: '.container',
19     data: {
20         upvotes: 0
21     },
22     // define methods under the **`methods`** object
23     methods: {
24         upvote: function(){
25             // **`this`** inside methods points to the Vue instance
26             this.upvotes++;
27         }
28     }
29 })
30 </script>
31 </html>
```

Nous lisons un écouteur d'événement click à une méthode appelée ‘`upvote`’. Il fonctionne comme avant, mais plus propre et plus facile à comprendre lors de la lecture de votre code.



Avertissement

Les gestionnaires d'événements sont limités à exécuter **une seule instruction**.

Syntaxe simplifiée pour v-on

Lorsque vous vous retrouvez à utiliser `v-on` tout le temps dans votre projet, vous découvrirez que votre HTML devient rapidement moches. Heureusement, il existe un raccourci pour `v-on`, le symbole `@`. Le `@` remplace l'ensemble `v-on`: et en l'utilisant, le code semble *beaucoup plus propre*. L'utilisation de ce raccourci est totalement facultative.

Avec l'utilisation de `@` le bouton de notre exemple précédent deviens :

Listening to 'click' using `v-on`:

```
<button v-on:click="upvote">  
    Upvote! {{upvotes}}  
</button>
```

Listening to 'click' using `@` shorthand

```
<button @click="upvote">  
    Upvote! {{upvotes}}  
</button>
```

Modificateurs d'événements

Maintenant, nous allons passer à la création d'une Calculatrice. Pour ce faire, nous utiliserons un formulaire avec deux entrées et une liste déroulante pour sélectionner l'opération souhaitée.

Même si le code suivant semble très bien, notre calculatrice ne fonctionne pas comme prévu.

```
1  <html>  
2  <head>  
3      <title>Calculator</title>  
4      <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.\  
5  css" rel="stylesheet">  
6  </head>  
7  <body>  
8      <div class="container">  
9          <h1>Type 2 numbers and choose operation.</h1>  
10         <form class="form-inline">  
11             <!-- Notice here the special modifier 'number'  
12             is passed in order to parse inputs as numbers.-->  
13             <input v-model.number="a" class="form-control">  
14             <select v-model="operator" class="form-control">  
15                 <option>+</option>
```

```
16      <option>-</option>
17      <option>*</option>
18      <option>/</option>
19  </select>
20  <!-- Notice here the special modifier 'number'
21  is passed in order to parse inputs as numbers.-->
22  <input v-model.number="b" class="form-control">
23  <button type="submit" @click="calculate"
24  class="btn btn-primary">
25      Calculate
26  </button>
27 </form>
28 <h2>Result: {{a}} {{operator}} {{b}} = {{c}}</h2>
29 <pre>
30     {{ $data }}
31 </pre>
32 </div>
33 </body>
34 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
35 <script type="text/javascript">
36 new Vue({
37     el: '.container',
38     data: {
39         a: 1,
40         b: 2,
41         c: null,
42         operator: "+",
43     },
44     methods: {
45         calculate: function(){
46             switch (this.operator) {
47                 case "+":
48                     this.c = this.a + this.b
49                     break;
50                 case "-":
51                     this.c = this.a - this.b
52                     break;
53                 case "*":
54                     this.c = this.a * this.b
55                     break;
56                 case "/":
57                     this.c = this.a / this.b
58             }
59         }
60     }
61 })
```

```
58         break;
59     }
60   }
61 },
62 });
63 </script>
64 </html>
```

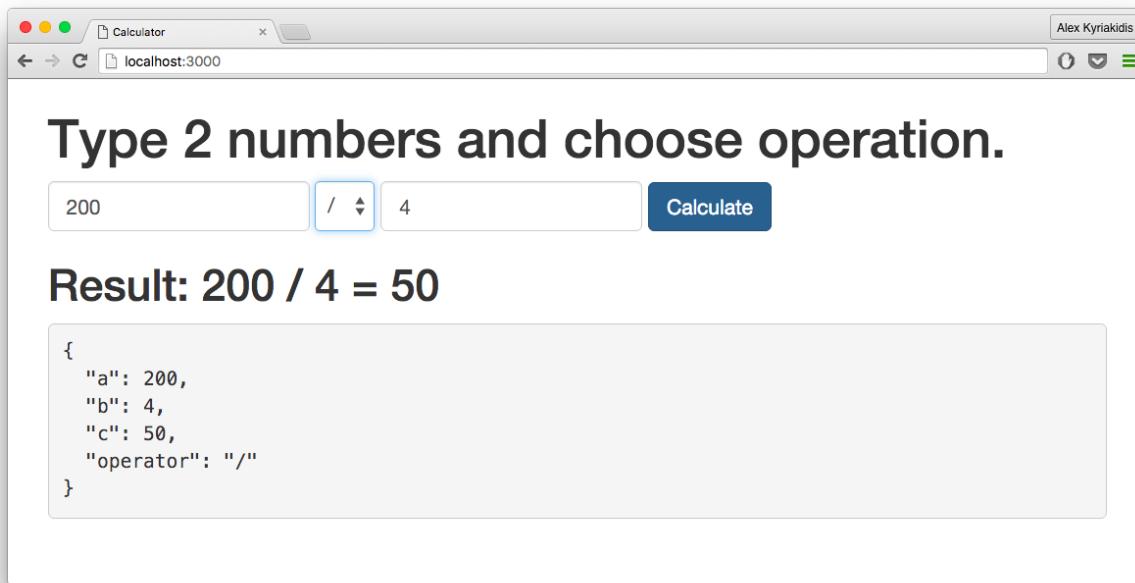
Si vous essayez d'exécuter ce code vous-même, vous découvrirez que lorsque le bouton “calculer” est cliqué, Au lieu de calculer, il recharge la page.

Cela est logique, car lorsque vous cliquez sur “calculer”, en arrière-plan, vous soumettez le formulaire Et donc la page recharge.

Pour empêcher la soumission du formulaire, nous devons annuler l'action par défaut de l'événement `onsubmit`. Il est très fréquent d'appeler `event.preventDefault()` dans notre méthode de gestion des événements. Dans notre cas, la méthode de gestion des événements s'appelle `calculate`.

Ainsi, notre méthode deviendra :

```
calculate: function(event){
  event.preventDefault();
  switch (this.operator) {
    case "+":
      this.c = this.a + this.b
      break;
    case "-":
      this.c = this.a - this.b
      break;
    case "*":
      this.c = this.a * this.b
      break;
    case "/":
      this.c = this.a / this.b
      break;
  }
}
```



Utilisation des modificateurs d'événements pour la création d'une calculatrice.

Bien que nous puissions le faire facilement à l'intérieur des méthodes, il serait préférable que les méthodes puissent être purement ignorants sur la logique des données plutôt que d'avoir à traiter les détails des événements DOM.

Vue.js fournit quatre modificateurs d'événements pour `v-on` pour empêcher le comportement par défaut de l'événement :

1..`prevent` 2..`stop` 3..`capture` 4..`self`

Ainsi, en utilisant `.prevent`, notre bouton de soumission changera de :

```
1 <button type="submit" @click="calculate">Calculate</button>
```

à :

```
1 <!-- L'événement submit ne recharge plus la page -->
2 <button type="submit" @click.prevent="calculate">Calculate</button>
```

Et nous pouvons maintenant supprimer en toute sécurité `event.preventDefault()` de notre méthode `calculate`.



Remarque

`.capture` et `.self` sont rarement utilisés, nous ne couvrirons donc pas ces parties. Si vous voulez en savoir plus sur les *ordres d'événements* (event order) jetez un oeil à ce [tutoriel](#)²³.

²³http://www.quirksmode.org/js/events_order.html

Modificateurs de clés (Key Modifiers)

Lorsque vous faites un focus (cliquer dans un input) sur l'une des entrées et que vous appuyez sur Entrée, vous remarquerez que la méthode de calcul est appelée. Si le bouton n'était pas dans le formulaire, ou s'il n'y avait aucun bouton du tout, vous pourriez écouter un événement au clavier à la place.

Lors de l'écoute d'événements de clavier, nous avons souvent besoin de vérifier les codes de clé. Le code de la touche **Enter** est 13. Donc nous pourrions l'utiliser comme ceci :

```
1 <input v-model="a" @keyup.13="calculate">
```

Se souvenir de tous les keyCodes est une gêne, Vue.js fournit donc des alias pour les clés les plus couramment utilisées :

- enter (entrer)
- tab
- delete (supprimer)
- esc
- space (espace)
- up (haut)
- down (bas)
- left (gauche)
- right (droite)

Chaque lettre possède également un alias de clé. Par exemple : `@keyup.a="aTyped"`.

Ainsi, pour exécuter la méthode `calculate` lorsque Enter est pressé dans notre exemple, les input seront comme ceci :

```
1 <input v-model="a" @keyup.enter="calculate">
2 <input v-model="b" @keyup.enter="calculate">
```



Astuce

Lorsque vous avez un formulaire avec beaucoup d'inputs / boutons / etc et que vous devez empêcher leur comportement de soumission (submit) par défaut, vous pouvez modifier l'événement `submit` du formulaire.

Par exemple : `<form @submit.prevent="calculate">`

** Enfin, la calculatrice est opérationnelle. **

Propriétés calculées

Les expressions en ligne de Vue.js sont très pratiques, mais pour une logique plus compliquée, vous devez utiliser des propriétés calculées. Les propriétés calculées sont pratiquement des variables dont la valeur dépend d'autres facteurs.

- Les propriétés calculées fonctionnent comme des fonctions que vous pouvez utiliser comme propriétés. * Mais il ya une différence significative. Chaque fois qu'une dépendance d'une propriété calculée change, la valeur de la propriété calculée est réévaluée.

Dans Vue.js, vous définissez les propriétés calculées dans l'objet **computed** dans votre instance **Vue**.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4 <title>Hello Vue</title>
5 </head>
6 <body>
7 <div class="container">
8   a={{ a }}, b={{ b }}
9   <pre>
10    {{ $data }}
11   </pre>
12 </div>
13 </body>
14 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
15 <script type="text/javascript">
16 new Vue({
17   el: '.container',
18   data: {
19     a: 1,
20   },
21   computed: {
22     // a computed getter
23     b: function () {
24       // **`this`** points to the Vue instance
25       return this.a + 1
26     }
27   }
28 });
29 });
30 </script>
31 </html>
```

Nous avons défini deux variables, La première, **a**, est définie à 1 et la seconde, **b**, sera définie par le résultat retourné de la fonction à l'intérieur de l'objet calculé. Dans cet exemple, la valeur de **b** sera définie à 2.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5 <title>Hello Vue</title>
6 </head>
7 <body>
8 <div class="container">
9     a={{ a }}, b={{ b }}
10    <input v-model="a">
11    <pre>
12        {{ $data }}
13    </pre>
14 </div>
15 </body>
16 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
17 <script type="text/javascript">
18 new Vue({
19     el: '.container',
20     data: {
21         a: 1,
22     },
23     computed: {
24         // a computed getter
25         b: function () {
26             // **`this`** points to the vm instance
27             return this.a + 1
28         }
29     }
30 });
31 </script>
32 </html>
```

L'exemple ci-dessus est le même que le précédent, mais avec une différence. Un élément input est lié à la variable **a**. Le résultat souhaité serait de modifier la valeur de l'attribut lié et de immédiatement mettre à jour le résultat de **b**. Mais remarquez dans ce cas, que cela ne fonctionne pas comme prévu.

Si vous exécutez ce code et définissez la variable **a** à 5, vous prévoyez que **b** sera égal à 6. Bien sûr, mais ce n'est pas le cas, **b** a la valeur 51.

Pourquoi est-ce que cela se produit ? Eh bien, comme vous pouvez le penser, **b** prend la valeur donnée dans l'input **a** comme chaîne et ajoute le nombre **1** à la fin de celle-ci.

Une solution possible consiste à utiliser la fonction `parseFloat()` qui analyse une chaîne et renvoie un nombre en virgule flottante.

```
new Vue({
  el: '.container',
  data: {
    a: 1,
  },
  computed: {
    b: function () {
      return parseFloat(this.a) + 1
    }
  }
});
```

Une autre option qui vient à l'esprit est d'utiliser le `<input type = " number ">` qui est utilisé pour les champs de saisie Qui doit contenir une valeur numérique.

Mais il ya une façon plus ordonnée. Avec Vue.js, chaque fois que vous souhaitez que l'entrée de l'utilisateur soit automatiquement conservée en tant que nombre, Vous pouvez ajouter le modificateur spécial `.number`.

```
<body>
<div class="container">
  a={{ a }}, b={{ b }}
  <input v-model.number="a">
  <pre>
    {{ $data }}
  </pre>
</div>
</body>
```

Le modificateur `number` va nous donner le résultat désiré sans effort supplémentaire.

Pour avoir une vision plus large des propriétés calculées, nous allons les utiliser et créer notre calculatrice Nous l'avons déjà fait, mais cette fois nous allons utiliser les propriétés calculées au lieu des méthodes.

Commençons par un exemple simple, où une propriété calculée **c** contient la somme de **a** plus **b**.

```
1 <html>
2 <head>
3   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.mi\
4 n.css" rel="stylesheet">
5   <title>Hello Vue</title>
6 </head>
7 <body>
8   <div class="container">
9     <h1>Enter 2 numbers to calculate their sum.</h1>
10    <form class="form-inline">
11      <input v-model.number="a" class="form-control">
12      +
13      <input v-model.number="b" class="form-control">
14    </form>
15    <h2>Result: {{a}} + {{b}} = {{c}}</h2>
16    <pre> {{ $data }} </pre>
17  </div>
18 </body>
19 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
20 <script type="text/javascript">
21 new Vue({
22   el: '.container',
23   data: {
24     a: 1,
25     b: 2
26   },
27   computed: {
28     c: function () {
29       return this.a + this.b
30     }
31   }
32 });
33 </script>
34 </html>
```

Le code initial est prêt, et à ce stade, l'utilisateur peut taper 2 nombres et obtenir leur somme. Une calculatrice qui peut faire les quatre opérations de base est l'objectif, alors continuons à construire !

Puisque le code HTML sera le même avec la [calculatrice que nous avons construit dans la section précédente de ce chapitre](#) (sauf que maintenant nous n'avons pas besoin d'un bouton), Je vais seulement montrer le code Javascript.

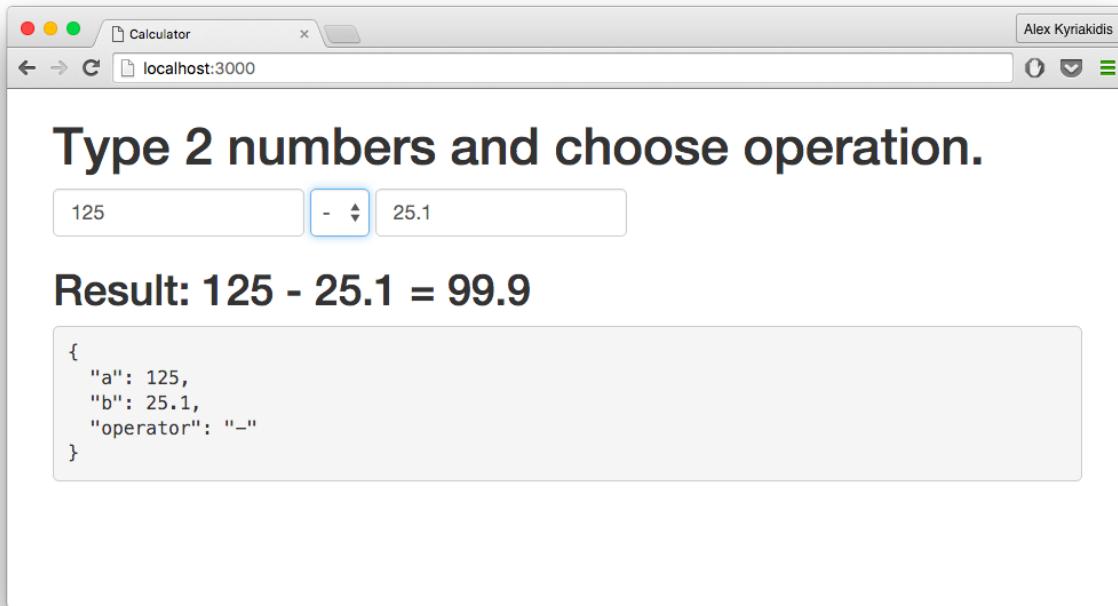
```
1 new Vue({
2   el: '.container',
3   data: {
4     a: 1,
5     b: 2,
6     operator: "+",
7   },
8   computed: {
9     c: function () {
10       switch (this.operator) {
11         case "+":
12           return this.a + this.b
13           break;
14         case "-":
15           return this.a - this.b
16           break;
17         case "*":
18           return this.a * this.b
19           break;
20         case "/":
21           return this.a / this.b
22           break;
23     }
24   }
25 },
26});
```

La calculatrice est prête à l'emploi. La seule chose que nous devions faire, c'était de déplacer tout ce qui était à l'intérieur de la **méthode calculate** vers la propriété calculée **c** ! Chaque fois que vous changez la valeur de **a** ou **b** le résultat est mis à jour en temps réel ! Nous n'avons pas besoin de boutons, d'événements ou quoi que ce soit. N'est ce pas magnifique ??



Remarque

Notez ici qu'une approche normale serait d'avoir une condition **if** pour éviter l'erreur de division. Mais, il ya déjà une prédition pour ce genre d'erreurs. Si l'utilisateur tape 1/0, le résultat devient automatiquement l'infini ! Si l'utilisateur tape un texte, le résultat affiché est « not a number ».



Calculatrice construite avec les propriétés calculées.



Code Examples

Vous trouverez les exemples de code de ce chapitre sur [GitHub](#)²⁴.

²⁴<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter5>

Devoirs

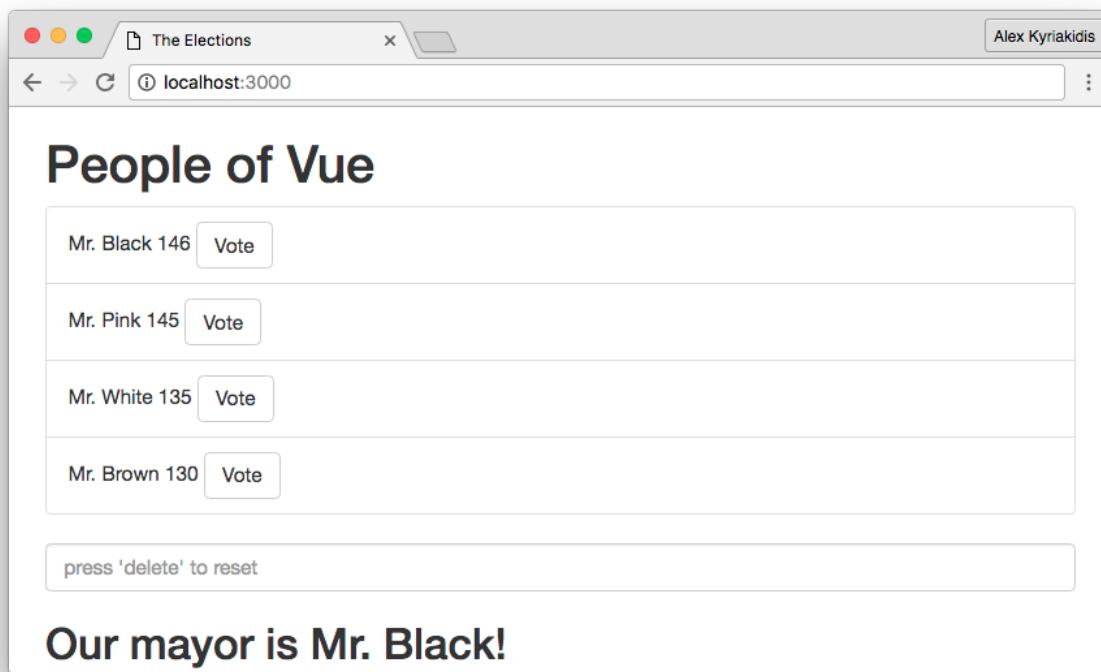
Maintenant que vous avez une compréhension de base de la gestion de l'événement, les méthodes, les propriétés calculées etc, Vous devriez essayer quelque chose un peu plus difficile. Commencez par créer un tableau de « maire » candidats. Chaque candidat a un « nom » et un certain nombre de « votes ». Utilisez un bouton pour augmenter le nombre de votes pour chaque candidat. Utilisez une propriété calculée pour déterminer qui est l'actuel « maire », et afficher son nom.

Enfin, ajoutez un input. Lorsque cet input est focus et que la touche ‘supprimer’ est enfoncée, les élections commencent par le début. Cela signifie que tous les votes deviennent 0.



Conseil

Les méthodes Javascript “`sort()`** et **`map()`‘ peuvent s'avérer très utiles et les modificateurs de clés vous y aideront.



Exemple du résultat



Solution Potentielle

Vous pouvez trouver une potentielle solution à cet exercice [ici](#)²⁵.

²⁵<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter5.html>

Filtres

Dans les deux chapitres précédents, nous avons examiné le rendu des listes, les méthodes et les propriétés calculées. Maintenant, c'est le moment de faire quelques exemples en utilisant toutes les informations ci-dessus. Dans ce chapitre, nous aborderons comment :

1. Filtrer un tableau d'éléments.
2. Ordonner un tableau d'éléments.
3. Appliquer un filtre personnalisé.
4. Utiliser une librairie utilitaire.

Le plan est de passer par des exemples similaires comme avant, en y combinant les techniques que nous avons vu.

Résultats filtrés

Parfois, nous devons afficher une version filtrée d'un tableau sans réellement modifier ou réinitialiser les données d'origine. Poursuivrons l'exemple précédent, [Boucle à travers un tableau d'objets](#), Nous aimerais afficher une liste avec les histoires écrites par * Alex * et une liste avec les histoires écrites par * John *. Nous pouvons y parvenir en créant une méthode qui filtre notre tableau et renvoie les résultats à afficher.



Information

Depuis la version 2.0, les filtres Vue.js ne peuvent pas être utilisés dans `v-for`. Les filtres ne peuvent être utilisés qu'à l'intérieur d'interpolations de texte (** {{ }} **). L'équipe de Vue.js suggère de déplacer la logique des filtres en JavaScript, afin qu'elle puisse être réutilisée dans votre composant.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
s" rel="stylesheet">
4     <title>User Stories</title>
5 </head>
6 <body>
7     <div class="container">
8         <h1>Let's hear some stories!</h1>
9         <div>
10            <h3>Alex's stories</h3>
11            <ul class="list-group">
12                <li v-for="story in storiesBy('Alex')"
13                    class="list-group-item"
14                >
15                    {{ story.writer }} said "{{ story.plot }}"
16                </li>
17            </ul>
18            <h3>John's stories</h3>
19            <ul class="list-group">
20                <li v-for="story in storiesBy('John')"
21                    class="list-group-item"
22                >
23                    {{ story.writer }} said "{{ story.plot }}"
24                </li>
25            </ul>
26        </div>
27        <pre>
28            {{ $data }}
29        </pre>
30    </div>
31 </body>
32 <script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js"></script>
33 <script type="text/javascript">
34 new Vue({
35     el: '.container',
36     data: {
37         stories: [
38             {
39                 plot: "I crashed my car today!",
40                 writer: "Alex"
41             },
42         ],
43     }
44 })
```

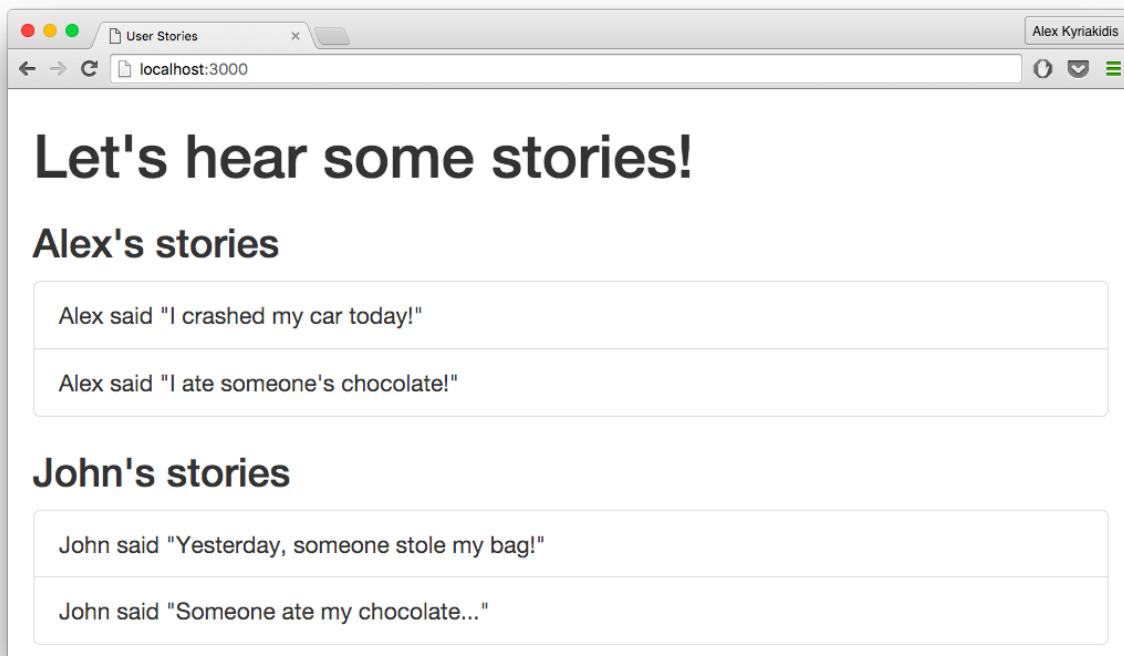
```
43      {
44          plot: "Yesterday, someone stole my bag!",
45          writer: "John"
46      },
47      {
48          plot: "Someone ate my chocolate...",
49          writer: "John"
50      },
51      {
52          plot: "I ate someone's chocolate!",
53          writer: "Alex"
54      },
55  ],
56 },
57 methods:
58 {
59     // a method which filters the stories depending on the writer
60     storiesBy: function (writer) {
61         return this.stories.filter(function (story) {
62             return story.writer === writer
63         })
64     },
65 }
66 })
67 </script>
68 </html>
```



Info

Dans la méthode `storiesBy`, nous utilisons la fonction `filter` de javascript. ([Https :// developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Array/filter](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Array/filter)) La fonction `filter ()` crée un nouveau tableau avec tous les éléments qui passent le test, mis en œuvre par la fonction fournie.

Comme il semble que nous avons créé une méthode appelée `storiesBy` qui prend dans un `writer`, comme argument, et renvoie un tableau filtré avec les histoires de l'écrivain. On peut alors utiliser ceci pour afficher les histoires de chaque auteur en utilisant la directive `v-for` dans le format de `story dans storiesBy ('Alex')`, comme vous pouvez le voir dans l'exemple ci-dessus .



Histoires filtrées par auteur.



Remarque

Comme vous l'avez peut-être remarqué, notre tag `1i` devient très grand, donc nous l'avons divisé en plus de lignes. Le résultat réel reste le même que celui de l'utilisation des filtres, introduit dans Vue 1.x.

Assez simple, pas vrai ?

Utilisation des propriétés calculées

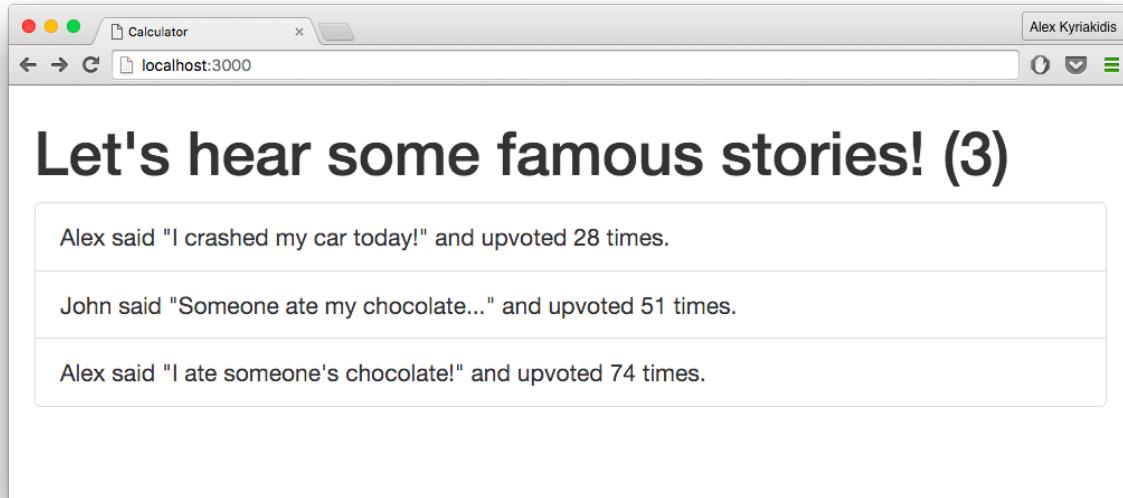
Une propriété *calculée* peut également être utilisée pour filtrer un tableau. L'utilisation d'une propriété calculée pour effectuer le filtrage en matrice vous donne un contrôle approfondi et une plus grande flexibilité, car c'est du pure JavaScript, et vous permet d'accéder au résultat filtré ailleurs. Par exemple, vous pouvez obtenir la longueur d'un tableau filtré n'importe où dans votre code.

Nous allons d'abord renforcer nos Histoires avec une nouvelle propriété, appelée *upvotes*. Ensuite, nous allons filtrer les histoires **famous**. Comme **famous** ou à la une, nous définissons les *Histoires* (stories) qui ont plus de 25 upvotes. Cette fois, nous allons créer une propriété calculée qui renvoie le tableau filtré.

```
new Vue({
  el: '.container',
  data: {
    stories: [
      {
        plot: "I crashed my car today!",
        writer: "Alex",
        upvotes: 28
      },
      {
        plot: "Yesterday, someone stole my bag!",
        writer: "John",
        upvotes: 8
      },
      {
        plot: "Someone ate my chocolate...",
        writer: "John",
        upvotes: 51
      },
      {
        plot: "I ate someone's chocolate!",
        writer: "Alex",
        upvotes: 74
      },
    ],
  },
  computed: {
    famous: function() {
      return this.stories.filter(function(item){
        return item.upvotes > 25;
      });
    }
  }
})
```

Dans notre code HTML, au lieu du tableau `stories`, nous allons rendre la propriété `famous` calculée.

```
<body>
  <div class="container">
    <h1>Let's hear some famous stories! ({{famous.length}})</h1>
    <ul class="list-group">
      <li v-for="story in famous"
          class="list-group-item">
        >
          {{ story.writer }} said "{{ story.plot }}"
          and upvoted {{ story.upvotes }} times.
      </li>
    </ul>
  </div>
</body>
```



Filtrer des tableaux en utilisant une propriété calculée

C'est tout. Nous avons filtré notre tableau en utilisant une propriété calculée. Avez-vous remarqué comment nous avons réussi à afficher le *nombre d'histoires célèbres* (famous story) À côté de notre message de titre en utilisant `{{famous.length}}` ?

Maintenant, nous allons mettre en œuvre une recherche très simple (mais géniale). Lorsque l'utilisateur tape une partie du nom d'une histoire, nous pouvons deviner de quelle histoire il s'agit et qui l'a écrit, en temps réel. Nous allons ajouter un texte `input`, lié à une variable vide `query`, afin de pouvoir filtrer dynamiquement notre tableau Stories.

```

1 <div class="container">
2   <h1>Lets hear some stories!</h1>
3   <div>
4     ...
5     <div class="form-group">
6       <label for="query">
7         What are you looking for?
8       </label>
9       <input v-model="query" class="form-control">
10      </div>
11      <h3>Search results:</h3>
12      <ul class="list-group">
13        <li v-for="story in search"
14          class="list-group-item">
15          >
16          {{ story.writer }} said "{{ story.plot }}"
17        </li>
18      </ul>
19    </div>
20 </div>

```

Ensute, nous allons créer une propriété calculée nommée `search`. Avec la fonction `filter` intégrée, nous allons utiliser la fonction Javascript `includes26`, qui détermine si une chaîne peut être trouvée dans une autre chaîne.

```

1 new Vue({
2   el: '.container',
3   data: {
4     stories: [...],
5     query: ''
6   },
7   methods: {
8     storiesBy: function (writer) {
9       return this.stories.filter(function (story) {
10       return story.writer === writer
11     })
12   }
13 },
14 computed: {
15   search: function () {
16     var query = this.query

```

²⁶https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/includes

```
17      return this.stories.filter(function (story) {  
18          return story.plot.includes(query)  
19      })  
20  }  
21 }  
22 })
```

The screenshot shows a web browser window titled "User Stories" with the URL "localhost:3000". The page content is as follows:

- Lets hear some stories!**
- Alex's stories**
 - Alex said "I crashed my car today!"
 - Alex said "I ate someone's chocolate!"
- John's stories**
 - John said "Yesterday, someone stole my bag!"
 - John said "Someone ate my chocolate..."
- What are you looking for?**
- Search results:**
 - Alex said "I crashed my car today!"
 - John said "Yesterday, someone stole my bag!"
 - John said "Someone ate my chocolate..."
 - Alex said "I ate someone's chocolate!"

Rechercher des Histoires.

The screenshot shows a web application interface titled "User Stories". The title bar includes the application name, the user's name "Alex Kyriakidis", and the URL "localhost:3000".

Alex's stories:

- Alex said "I crashed my car today!"
- Alex said "I ate someone's chocolate!"

John's stories:

- John said "Yesterday, someone stole my bag!"
- John said "Someone ate my chocolate..."

What are you looking for?

choco

Search results:

- John said "Someone ate my chocolate..."
- Alex said "I ate someone's chocolate!"

Recherche de 'choco'.

** N'est-ce pas génial ? **

Résultats ordonnées

Parfois, nous pouvons souhaiter afficher les éléments d'un tableau ordonnés par certains critères. Nous pouvons utiliser une *propriété calculée* pour afficher notre tableau, ordonnée par le nombre de *upvotes de chaque histoire*. Pour trier le tableau, nous allons utiliser la fonction JavaScript `sort`²⁷, Qui trie les éléments d'un tableau et renvoie ce tableau.

- Plus une histoire est célèbre, plus haut elle doit apparaître dans les résultats de recherche. *

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5     <title>Famous Stories</title>
6 </head>
7 <body>
8     <div class="container">
9         <h1>Let's hear some stories!</h1>
10        <ul class="list-group">
11            <li v-for="story in orderedStories"
12                class="list-group-item"
13                >
14                {{ story.writer }} said "{{ story.plot }}"
15                and upvoted {{ story.upvotes }} times.
16            </li>
17        </ul>
18        <pre>
19            {{ $data }}
20        </pre>
21    </div>
22 </body>
23 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
24 <script type="text/javascript">
25 new Vue({
26     el: '.container',
27     data: {
28         stories: [...]
29     },
30     computed: {
```

²⁷https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort

```
31     orderedStories: function () {
32         return this.stories.sort(function(a, b){
33             return a.upvotes - b.upvotes;
34         })
35     }
36   }
37 }
38 </script>
39 </html>
```

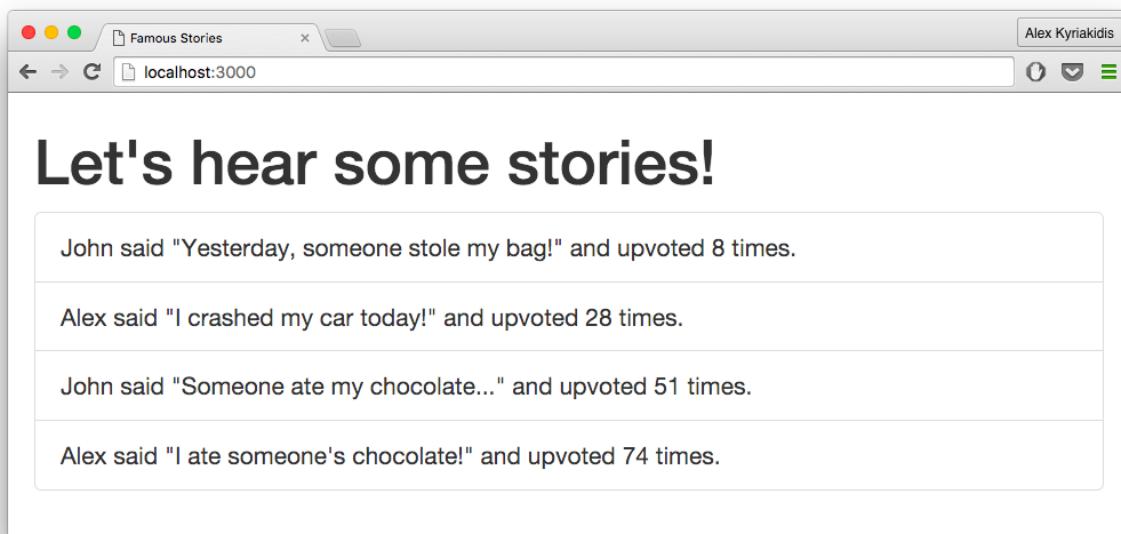


tableau des histoires ordonnées par upvotes.

Hmmm, le tableau est ordonné mais ce n'est pas ce que nous attendions. Nous voulions les **histoires célèbres en premier**.

Pour changer l'ordre du tableau trié nous devons regarder la fonction **sort**. Dans le script **sort (compareFunction)**, si **compareFunction** est fourni, les éléments du tableau sont triés selon la valeur de retour de **compareFunction**. Si **a** et **b** sont deux éléments comparés, alors :

- Si **compareFunction (a, b)** est inférieur à 0, trier **a** à un index inférieur à **b**.
- Si **compareFunction (a, b)** est 0, laissez **a** et **b** inchangés.
- Si **compareFunction (a, b)** est supérieur à 0, trier **b** à un index inférieur à **a**.

Dans notre cas, la fonction **compareFunction** sera :

compareFunction

```
function(a, b){  
    return a.upvotes - b.upvotes;  
}
```

Ainsi, pour changer l'ordre de croissant à décroissant, nous pouvons multiplier la valeur renvoyée par **-1**. (** return (a.upvotes - b.upvotes) * -1 **)

Nous pouvons changer l'ordre dynamiquement, en utilisant une variable, **order**. Un **bouton** sera utilisé, ce qui changera la valeur de la nouvelle variable, entre **-1** et **1**.

```
1 <div class="container">  
2     <h1>Let's hear some stories!</h1>  
3     <ul class="list-group">  
4         <li v-for="story in orderedStories"  
5             class="list-group-item"  
6         >  
7             {{ story.writer }} said "{{ story.plot }}"  
8             and upvoted {{ story.upvotes }} times.  
9         </li>  
10    </ul>  
11    <button @click="order = order * -1">Reverse Order</button>  
12 <pre>  
13     {{ $data }}  
14 </pre>  
15 </div>  
  
1 new Vue({  
2     el: '.container',  
3     data: {  
4         stories: [...],  
5         order : -1  
6     },  
7     computed: {  
8         orderedStories: function () {  
9             var order = this.order;  
10            return this.stories.sort(function(a, b) {  
11                return (a.upvotes - b.upvotes) * order;  
12            })  
13        }  
14    }  
15 })
```

Nous initialisons la variable `order` avec la valeur `-1` , ensuite nous la transmettons à notre propriété calculée, de cette façon, à chaque fois que le bouton est cliqué, la variable change de valeur et le tableau change d'ordre.

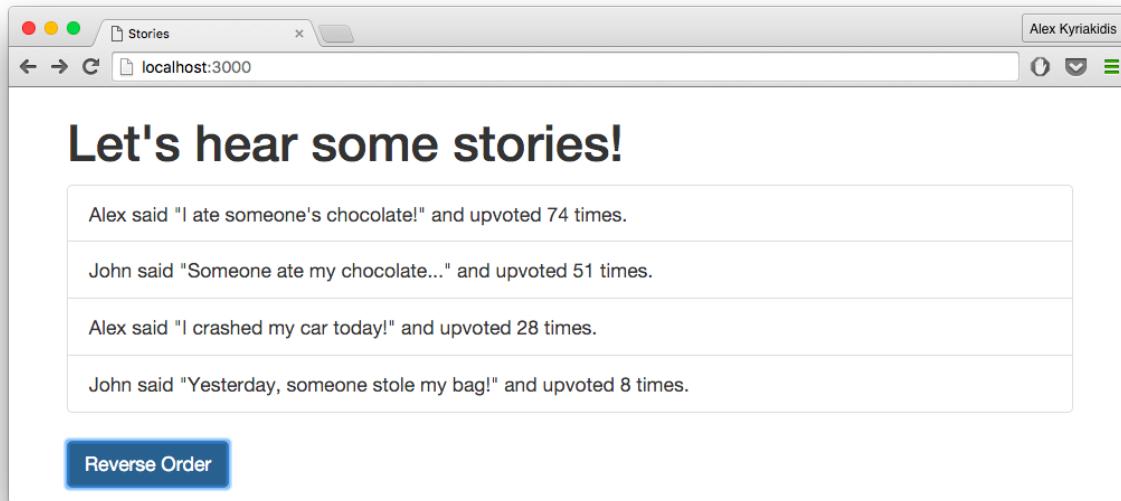


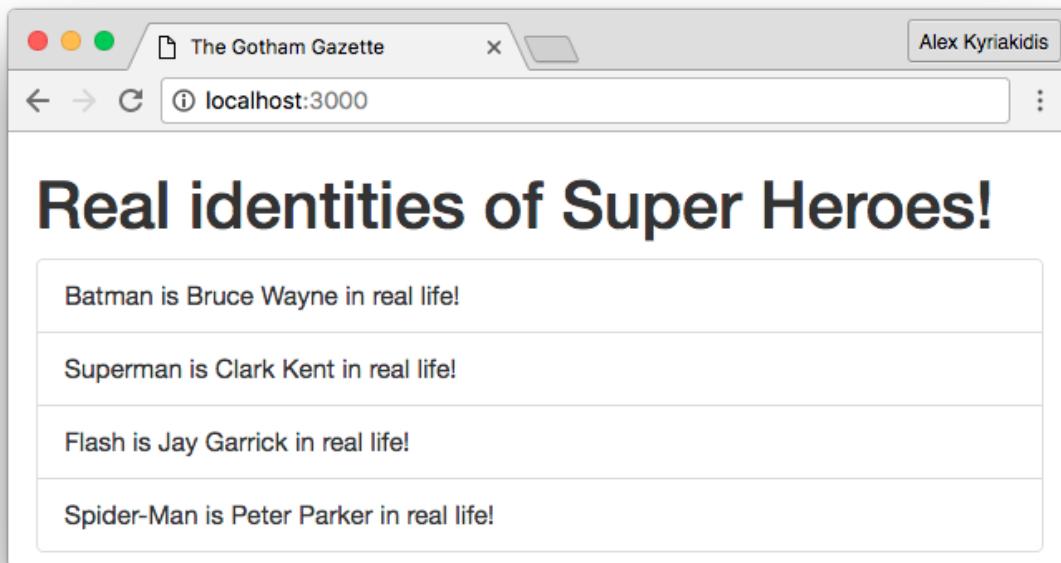
Tableau dans l'ordre décroissant

Filtres personnalisés

Pour comprendre les filtres personnalisés, nous allons faire un nouvel exemple simple. Supposons que nous sommes maintenant en charge du journal de la ville de Gotham, le journal The Gotham Gazette. Notre premier travail est de diffuser les nouvelles des identités secrètes des héros. Nous connaissons le prénom et le nom de famille, et nous voulons faire une belle liste où chaque identité secrète sera exposée. C'est là que la méthode globale `Vue.filter()` entre, pour créer un filtre qui peut prendre un *héros* et renvoyer toutes ses informations pour affichage, sans polluer notre code HTML. Pour enregistrer un filtre, nous pouvons transmettre un `filterID` et un `filterFunction`, qui retourne une valeur traitée. Ensuite, nous utiliserons le filtre à l'intérieur des interpolations de texte comme suit :

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4     <title>The Gotham Gazette</title>
5 </head>
6 <body>
7     <div class="container">
8         <h1>Real identities of Super Heroes!</h1>
9         <ul class="list-group">
10            <li v-for="hero in heroes"
11                class="list-group-item">
12                >
13                    {{ hero | snitch }}
14                </li>
15            </ul>
16        </div>
17    </body>
18 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js">
19 </script>
20 <script>
21 Vue.filter('snitch', function (hero) {
22     return hero.secretId + ' is '
23     + hero.firstname + ' '
24     + hero.lastname + ' in real life!'
25 })
26
27
28 new Vue({
29     el: '.container',
30     data: {
```

```
31     heroes: [
32         { firstname: 'Bruce', lastname: 'Wayne', secretId: 'Batman' },
33         { firstname: 'Clark', lastname: 'Kent', secretId: 'Superman' },
34         { firstname: 'Jay', lastname: 'Garrick', secretId: 'Flash' },
35         { firstname: 'Peter', lastname: 'Parker', secretId: 'Spider-Man' }
36     ]
37 }
38 })
39 </script>
40 </html>
```



Le filtre personnalisé « famous » en action.

Librairies utilitaires

À ce stade, nous aimerions souligner que lorsque vous avez besoin de trier / filtrer / indexer les données de façon plus avancée, vous devriez envisager d'utiliser une bibliothèque d'utilitaires JavaScript. Il ya quelques grandes bibliothèques utilitaires telles que [Lodash] (<https://lodash.com>), [Underscore] (<http://underscorejs.org/>), [Sugar] (<https://sugarjs.com/>), etc.

Pour mieux comprendre, nous allons inclure *Lodash* et mettre à jour l'exemple précédent.

Assurez-vous d'inclure *Lodash* à partir d'un cdn dans votre fichier **HTML**.

La méthode `orderBy` de Lodash renvoie un nouveau tableau trié. nous allons donc l'utiliser dans notre propriété calculée pour trier le tableau Histoires.

Syntaxe

La syntaxe de méthode `orderBy` de Lodash est :

```
1  _.orderBy(collection, [iteratees=[_.identity]], [orders])
```

Ne laissez pas le deuxième argument vous confondre. C'est vraiment simple. Le premier argument représente le tableau que vous souhaitez trier. Le second argument attend un tableau de clés, sur lesquelles le tri du tableau sera basé.

Le troisième argument, attend un tableau d'*ordres* pour chaque clé. Par exemple, si nous avions un tableau : {lang="javascript", line-numbers="off"} var kids = [{ name : 'Stan', strength : 70, intelligence : 70}, { name : 'Kyle', strength : 40, intelligence : 80}, { name : 'Eric', strength : 45, intelligence : 80}, { name : 'Kenny', strength : 100, intelligence : 70}] Et que nous exécutions le code : {lang="javascript", line-numbers="off"} var kids = [{ name : 'Kyle', strength : 40, intelligence : 80}, { name : 'Eric', strength : 45, intelligence : 80}, { name : 'Stan', strength : 70, intelligence : 70}, { name : 'Kenny', strength : 100, intelligence : 70}] Parce que le tableau est principalement classé par l'intelligence des enfants dans l'ordre « descendant » et secondairement, par la force des enfants dans l'ordre croissant.

Nous utiliserons `_.orderBy` dans notre propriété calculée comme ceci :

```
computed: {
  orderedStories: function () {
    var order = this.order
    return _.orderBy(this.stories, 'upvotes')
  }
}
```

Cela fonctionne, mais si aucun argument *ordres* n'est transmis, le tableau sera trié dans l'ordre croissant. En outre, il devrait être possible de modifier l'ordre du tableau avec un bouton, comme avant. Pour le faire dynamiquement, nous pouvons définir une propriété de données `order: 'desc'` et créer une méthode pour modifier sa valeur :

```
methods: {
  reverseOrder: function () {
    this.order = (this.order === 'desc') ? 'asc' : 'desc'
  }
},
computed: {
  orderedStories: function () {
    var order = this.order
    return _.orderBy(this.stories, 'upvotes', [order])
  }
}
}
```

Et le bouton sera presque le même, mais pas tout à fait.

```
<button v-on:click="reverseOrder">
```

C'est assez. Nous avons atteint la même fonctionnalité en utilisant * Lodash *.



Astuce

Lorsque vous utilisez une bibliothèque d'utilitaires pour filtrer / ordonner des données, vous pouvez incrémenter le tableau obtenu sans utiliser de propriétés calculées.

Nous pouvons mettre à jour cet exemple pour concrétiser l'idée. Notre HTML qui affiche le tableau trié serait :

```
<div class="container">
  <h1>Let's hear some stories!</h1>
  <ul class="list-group">
    <li v-for="story in _.orderBy(stories, ['upvotes'], ['desc'])">
      {{ story.writer }} said "{{ story.plot }}"
      and upvoted {{ story.upvotes }} times.
    </li>
  </ul>
</div>
```

C'est tout. Pas besoin d'écrire du *JavaScript*.



Exemples de code

Vous trouverez les exemples de code de ce chapitre sur [GitHub](#)²⁸.

²⁸<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter6>

Devoirs

Pour l'exercice de ce chapitre, vous devez faire ce qui suit. Commencez par créer un tableau de personnes. Chaque personne a un nom et un âge. En utilisant ce que vous avez appris jusqu'à présent, essayez de rendre le tableau dans une liste et de le trier par « âge ». Ensuite, créez une deuxième liste ci-dessous et créez un calcul correctement nommé “old”, qui renverra toutes les personnes âgées de plus de 65 ans.

N'hésitez pas à remplir le tableau avec vos propres données. Soyez prudent pour ajouter des personnes âgées de plus de 65 ans pour vous assurer que votre filtre fonctionne correctement. Aller de l'avant !



Conseil

La fonction `.filter` intégrée à Vue.js est nécessaire ici.

The screenshot shows a web application titled "People of Gaul" running on localhost:3000. The main section displays a list of four individuals: Obelix (31), Asterix (32), Julius Caesar (56), and Majestix (62). Below this, a section titled '"Old" People of Gaul' shows a filtered list containing only Julius Caesar (56) and Majestix (62).

Exemple



Solution potentielle

Vous pouvez trouver une solution potentielle à cet exercice [ici](#)²⁹.

²⁹<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter6.html>

Components

Que sont les composants ?

Les composants sont l'une des fonctionnalités les plus puissantes de Vue.js. Ils vous aident à étendre les éléments HTML de base pour encapsuler le code réutilisable. À un niveau élevé, les composants sont des éléments personnalisés auxquels le compilateur Vue.js attache un comportement spécifié. Dans certains cas, ils peuvent également apparaître comme un élément HTML natif étendu avec l'attribut spécial `is`.

C'est une manière vraiment intelligente et puissante d'étendre le HTML pour faire de nouvelles choses. Dans ce chapitre, nous allons commencer par un exemple extrêmement simple. Ensuite, nous allons voir comment les composants peuvent nous aider à améliorer le code que nous avons créé dans les chapitres précédents.

Utilisation des composants

Nous allons commencer par un composant simple. Pour utiliser un composant, il faut d'abord le déclarer.

Une façon d'enregistrer un composant est d'utiliser la méthode `Vue.component` et de passer le `tag` et le `constructor`. Pensez au `tag` comme au nom du Component et au `constructor` comme ses options. Dans notre cas, nous nommerons la composante `story` et nous définirons la propriété `story` (encore une fois). L'option `template` (comment nous aimerais que notre article soit affiché), se trouve dans le `constructor`, où d'autres options seront ajoutées.

Notre composante de story sera enregistrée comme ceci

```
1 Vue.component('story', {  
2   template: '<h1>My horse is amazing!</h1>'  
3 });
```

Maintenant que nous avons enregistré la composante, nous allons l'utiliser. Nous ajouterons l'élément personnalisé `<story>` à l'intérieur de l'HTML, pour afficher l'histoire ou la story si vous voulez.

```
1 <html>
2 <head>
3   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.mi\
4 n.css" rel="stylesheet">
5   <title>Hello Vue</title>
6 </head>
7 <body>
8   <div class="container">
9     <story></story>
10  </div>
11 </body>
12 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
13 <script type="text/javascript">
14 Vue.component('story', {
15   template: '<h1>My horse is amazing!</h1>'
16 });
17
18 new Vue({
19   el: '.container'
20 })
21 </script>
22 </html>
```

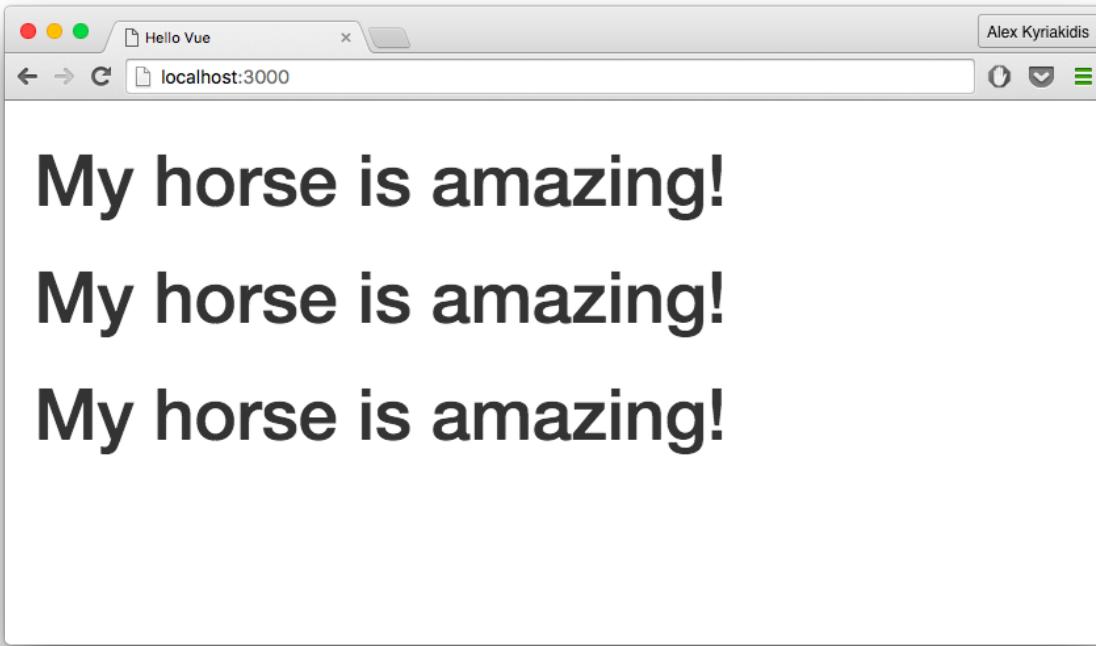


Remarque

Notez ici que vous pouvez donner à votre composant personnalisé le nom que vous voulez, mais il est généralement recommandé d'utiliser un nom unique pour éviter d'avoir des collisions avec des balises réelles qui pourraient être introduites à un moment donné dans l'avenir.

Comme nous l'avons mentionné au début du chapitre, les composants sont réutilisables. Cela signifie que vous pouvez ajouter autant d'éléments `<story>` que vous le souhaitez. L'extrait HTML suivant affichera notre histoire 3 fois.

```
<body>
  <div class="container">
    <story></story>
    <story></story>
    <story></story>
  </div>
</body>
```



Affichage du composant story

Templates

Il existe plus d'une façon de déclarer un template pour un composant. Le template en ligne que nous avons utilisé avant peut très rapidement rendre notre code difficile à lire.

Une autre façon est de créer une balise **script** avec un type **text / template** avec un **id** de **story-template**. Pour utiliser ce template, nous devons faire référence à un sélecteur dans l'option **template** de notre composant pour ce script.

```
<script type="text/template" id="story-template">
  <h1>My horse is amazing!</h1>
</script>

<script type="text/javascript">
  Vue.component('story', {
    template: "#story-template"
  });
</script>
```



Information

Le “text / template” n’est pas un script que le navigateur peut comprendre. Le navigateur l’ignore tout simplement. Ceci vous permet de mettre n’importe quoi dedans, ce qui peut alors être extrait pour générer des extraits d’HTML.

Ma manière préférée de définir un **template** (et celle que je vais utiliser dans les exemples de ce livre) Est de créer une balise HTML **template** et de lui donner un **id**. Ensuite, nous pouvons faire référence à un sélecteur comme nous l’avons fait auparavant. En utilisant cette technique le composant ci-dessus ressemblera à ceci :

```
<template id="story-template">
  <h1>My horse is amazing!</h1>
</template>

<script type="text/javascript">
  Vue.component('story', {
    template: "#story-template"
  });
</script>
```

Propriétés

Voyons maintenant comment nous pouvons utiliser plusieurs instances de notre composant **story** pour afficher une liste d’histoires. Nous devons mettre à jour le **template** pour ne pas afficher toujours la même histoire, mais plutôt l’intrigue de n’importe quelle histoire que nous voulons.

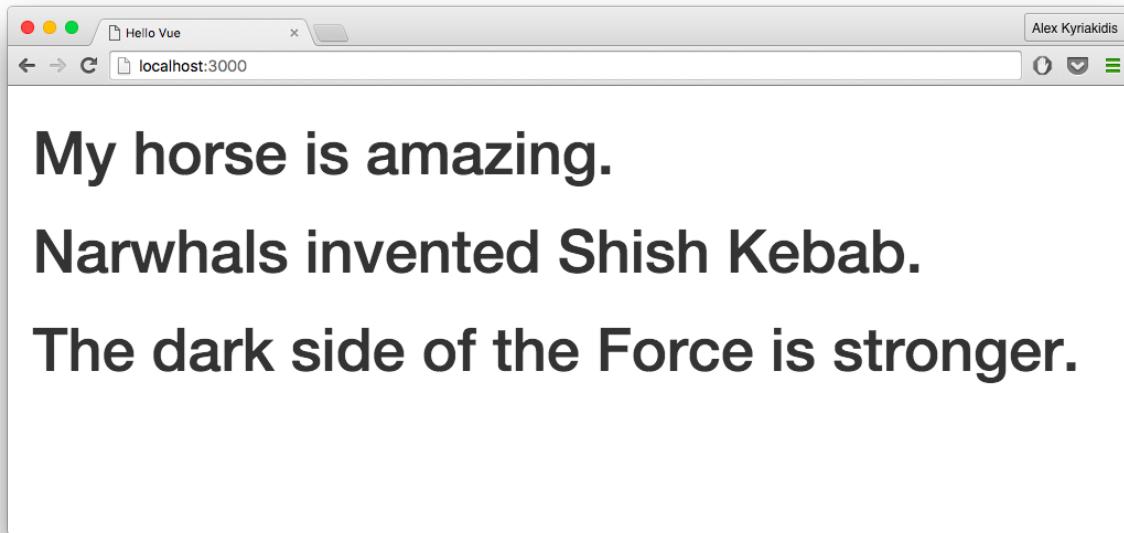
```
<template id="story-template">
  <h1>{{ plot }}</h1>
</template>
```

Nous devons également mettre à jour notre composant pour utiliser cette propriété. Pour ce faire, nous ajoutons la nouvelle propriété, ‘plot’, à l’attribut **props** du composant.

```
Vue.component('story', {
  props: ['plot'],
  template: "#story-template"
});
```

Maintenant, nous pouvons passer une **intrigue** ou plot en Anglais chaque fois que nous utilisons l’élément ** <story>**.

```
<div class="container">
  <story plot="My horse is amazing."></story>
  <story plot="Narwhals invented Shish Kebab."></story>
  <story plot="The dark side of the Force is stronger."></story>
</div>
```



Affichage ses différentes ‘histoires’.



Avertissement

Les attributs HTML sont insensibles à la casse. Lorsque vous utilisez des noms de prop de camelCased comme attributs, vous devez utiliser leurs équivalents kebab-case (décalés par un trait d’union).

Donc, camelCase en JavaScript et kebab-case en HTML. Par exemple, pour `props: ['isUser']`, l’attribut HTML serait `<story is-user ="true "></story>`.

Comme vous l’avez probablement imaginé, un composant peut avoir plus d’une propriété. Par exemple, si nous voulons afficher pour chaque histoire l’écrivain avec l’intrigue, nous devons passer le `writer` aussi.

```
<story plot="My horse is amazing." writer="Mr. Weebly"></story>
```

Si vous avez beaucoup de propriétés et que vos éléments deviennent illisibles, vous pouvez passer un objet et afficher ses propriétés.

Nous refactoriserons notre exemple une fois de plus pour l’envelopper dans un objet.

```
1 <html>
2 <head>
3   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.mi\
4 n.css" rel="stylesheet">
5   <title>Awesome Stories</title>
6 </head>
7 <body>
8   <div class="container">
9     <story v-bind:story="{plot: 'My horse is amazing.', writer: 'Mr. Weebly'}"\>
10    </story>
11    <story v-bind:story="{plot: 'Narwhals invented Shish Kebab.', writer: 'Mr. Weebly'}"\>
12      </story>
13      <story v-bind:story="{plot: 'The dark side of the Force is stronger.', writer: 'Darth Vader'}"\>
14        </story>
15        <story v-bind:story="{plot: 'My pony is the cutest.', writer: 'Mrs. Weebly'}"\>
16          </story>
17          <template id="story-template">
18            <h1>{{ story.writer }} said "{{ story.plot }}"</h1>
19          </template>
20        </div>
21      </body>
22      <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
23      <script type="text/javascript">
24        Vue.component('story', {
25          props: ['story'],
26          template: "#story-template"
27        });
28
29
30      new Vue({
31        el: '.container'
32      })
33    </script>
34  </html>
```



Information

`v-bind` est utilisé pour lier dynamiquement un ou plusieurs attributs, ou un composant, à une expression.

Puisque la propriété `story` n'est pas une chaîne mais un objet javascript Au lieu de faire `story = " ... "` nous allons utiliser `v-bind:story = " ... "` Pour lier la propriété `story` à l'objet passé.

Le raccourci pour `v-bind:story`, alors à partir de maintenant nous allons l'utiliser comme ceci : `:story="..."`.

Réutilisabilité

Jetons un coup d'œil à notre exemple [Résultats filtrés] (# filters-results). Supposons que cette fois, nous prenons les données de la variable `stories` d'une API externe via un appel http. Les développeurs de l'API ont décidé de renommer la propriété `plot` de l'histoire en `body`. Alors maintenant, nous devons passer par notre code et faire les changements nécessaires.



Information

Plus loin dans ce livre, nous allons voir comment nous pouvons utiliser Vue pour faire des requêtes http.

```
1 <div class="container">
2   <h1>Lets hear some stories!</h1>
3   <div>
4     <h3>Alex's stories</h3>
5     <ul class="list-group">
6       <li v-for="story in storiesBy('Alex')"
7           class="list-group-item">
8         <hr style="border-top: 1px solid black; margin-bottom: 10px;">
9         {{ story.writer }} said "{{ story.plot }}"
10        {{ story.writer }} said "{{ story.body }}"
11        </li>
12      </ul>
13      <h3>John's stories</h3>
14      <ul class="list-group">
15        <li v-for="story in storiesBy('John')"
16            class="list-group-item">
17          <hr style="border-top: 1px solid black; margin-bottom: 10px;">
18          {{ story.writer }} said "{{ story.plot }}"
```

```

19         {{ story.writer }} said "{{ story.body }}"
20     </li>
21 </ul>
22 <div class="form-group">
23     <label for="query">
24         What are you looking for?
25     </label>
26     <input v-model="query" class="form-control">
27 </div>
28 <h3>Search results:</h3>
29 <ul class="list-group">
30     <li vv-for="story in search"
31         class="list-group-item"
32     >
33     {{ story.writer }} said "{{ story.plot }}"
34     {{ story.writer }} said "{{ story.body }}"
35     </li>
36 </ul>
37 </div>
38 </div>

```



Remarque

Dans cet exemple particulier, la mise en surbrillance de la syntaxe est désactivée.

Comme vous l'avez peut-être remarqué, nous avons dû faire exactement le même changement 3 fois et je ne sais pas pour vous, mais je déteste me répéter. Si cela ne semble pas être une grosse affaire pour vous, imaginez que vous pouvez utiliser le bloc de code ci-dessus dans 100 endroits différents, que feriez-vous alors ? Heureusement, *Vue* fournit une solution pour ce genre de situations, et cette solution a un nom, **Composant**.



Astuce

Lorsque vous vous retrouvez à répéter un morceau de code, la façon la plus efficace de le traiter est de créer un composant dédié.

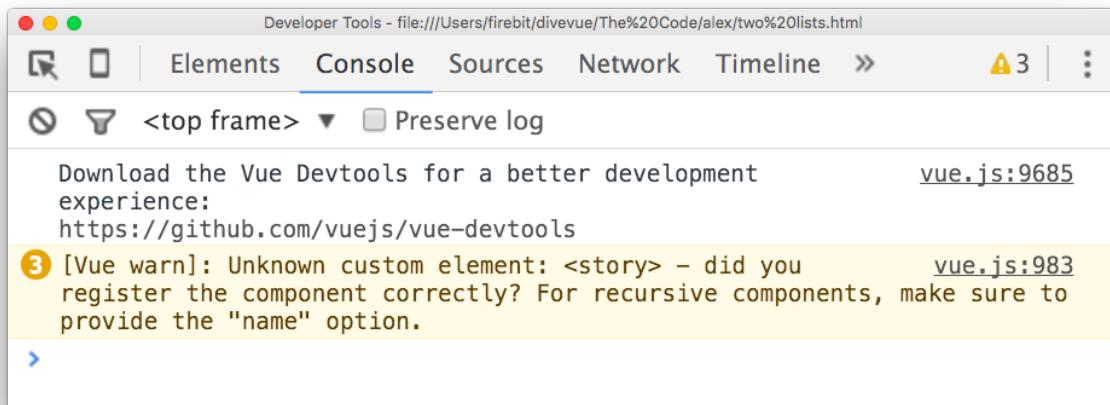
Heureusement, nous avons créé un élément **story** dans l'exemple précédent, qui affiche l'auteur et le corps pour une histoire spécifiée. Nous pouvons utiliser l'élément personnalisé **<story>** dans notre **HTML** et passer chaque histoire, comme nous l'avons fait auparavant, avec la balise : **story ****. cette fois Nous l'utiliserons dans une directive ****v-for**.

Notre code deviens donc :

```
<div class="container">
  <h1>Lets hear some stories!</h1>
  <div>
    <h3>Alex's stories</h3>
    <ul class="list-group">
      <story v-for="story in storiesBy('Alex')"
        :story="story"></story>
    </ul>
    <h3>John's stories</h3>
    <ul class="list-group">
      <story v-for="story in storiesBy('John')"
        :story="story"></story>
    </ul>
    <div class="form-group">
      <label for="query">What are you looking for?</label>
      <input v-model="query" class="form-control">
    </div>
    <h3>Search results:</h3>
    <ul class="list-group">
      <story v-for="story in search"
        :story="story"></story>
    </ul>
  </div>
</div>
```

Si vous essayez d'exécuter ce code, vous obtiendrez l'avertissement suivant :

*Vue warn : Unknown custom element :<story> - did you register the component correctly ?
For recursive components, make sure to provide the “name” option.*



Avertissement de Vue.js

Pour résoudre ce problème, nous devons enregistrer le composant à nouveau. Cette fois, nous devons apporter quelques modifications au template du composant. Nous allons changer l'attribut **plot** à **body** et le tag **<h1>** en **** pour répondre à nos besoins.

Ainsi, le template de l'histoire deviens :

```
<template id="story-template">
  <li class="list-group-item">
    {{ story.writer }} said "{{ story.body }}"
  </li>
</template>
```

Le composant reste le même.

```
1 Vue.component('story', {
2   props: ['story'],
3   template: '#story-template'
4 });
```

Si vous exécutez le code ci-dessus, vous verrez par vous-même que tout fonctionne comme avant, mais cette fois avec l'utilisation D'un composant personnalisé.

- Jolie hein ? *



Avertissement

Soyez responsable. Ne pas buvez pas en conduisant.

Mise en pratique

En utilisant nos connaissances nouvellement acquises, nous devrions être en mesure de construire quelque chose d'un peu plus complexe. Basé sur la Structure de l'exemple ci-dessus, Nous allons créer un système de vote pour notre **stories**, et ajouter la fonctionnalité * ajouter comme favori *. La façon d'y parvenir est d'utiliser Les méthodes, les directives et bien sûr les composants.

Commençons par les stories.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5     <title>Hello Vue</title>
6 </head>
7 <body>
8 <div id="app">
9     <div class="container">
10        <h1>Let's hear some stories!</h1>
11        <ul class="list-group">
12            <story v-for="story in stories" :story="story"></story>
13        </ul>
14        <pre>{{ $data }}</pre>
15    </div>
16 </div>
17 <template id="story-template">
18     <li class="list-group-item">
19         {{ story.writer }} said "{{ story.plot }}"
20     </li>
21 </template>
22 </body>
23 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
24 <script type="text/javascript">
25 Vue.component('story', {
26     template: "#story-template",
27     props: ['story'],
28 });
29
30 new Vue({
31     el: '#app',
32     data: {
33         stories: [
34             {
```

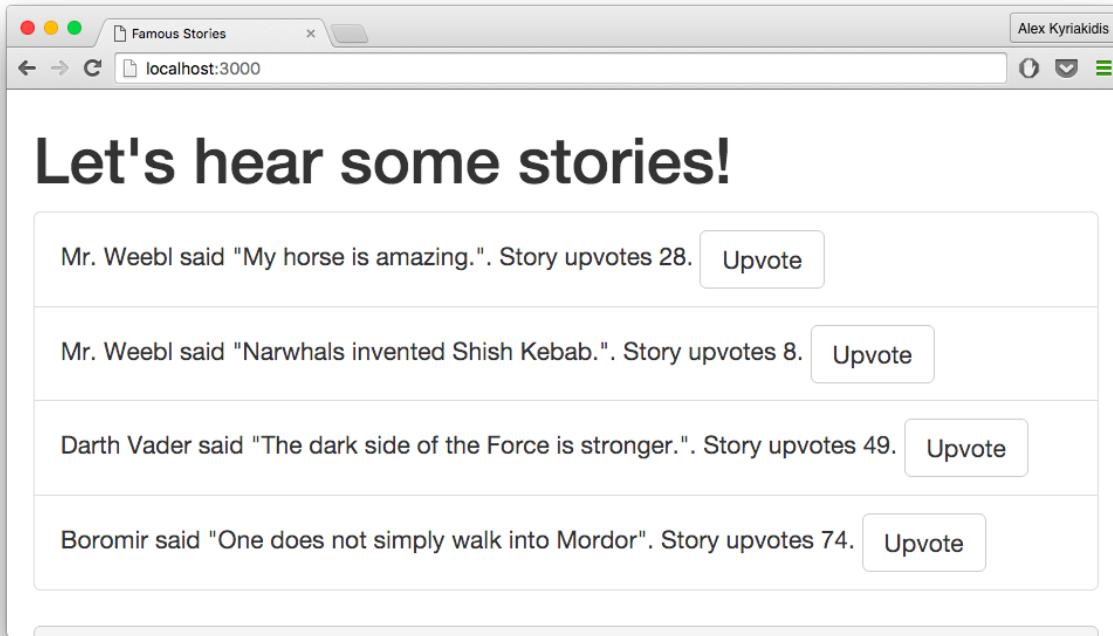
```
35         plot: 'My horse is amazing.',
36         writer: 'Mr. Weeb1',
37     },
38     {
39         plot: 'Narwhals invented Shish Kebab.',
40         writer: 'Mr. Weeb1',
41     },
42     {
43         plot: 'The dark side of the Force is stronger.',
44         writer: 'Darth Vader',
45     },
46     {
47         plot: 'One does not simply walk into Mordor',
48         writer: 'Boromir',
49     },
50     ]
51 }
52 })
53 </script>
54 </html>
```

L'étape suivante permet à l'utilisateur de voter pour l'histoire qu'il préfère. Pour appliquer cette limite (1 vote par histoire), nous afficherons le bouton 'Upvote' uniquement si l'utilisateur N'a pas encore voté. Donc, chaque histoire doit avoir une propriété **votée**, qui devient vraie quand **upvote** est Exécuté.

```
<template id="story-template">
  <li class="list-group-item">
    {{ story.writer }} said "{{ story.plot }}".
    Story upvotes {{ story.upvotes }}.
    <button v-show="!story.voted" @click="upvote"
            class="btn btn-default">
      Upvote
    </button>
  </li>
</template>
```

```
Vue.component('story', {
  template: "#story-template",
  props: ['story'],
  methods: {
    upvote: function(){
      this.story.upvotes += 1;
      this.story.voted = true;
    },
  }
});

new Vue({
  el: '#app',
  data: {
    stories: [
      {
        plot: 'My horse is amazing.',
        writer: 'Mr. Weeb1',
        upvotes: 28,
        voted: false,
      },
      {
        plot: 'Narwhals invented Shish Kebab.',
        writer: 'Mr. Weeb1',
        upvotes: 8,
        voted: false,
      },
      {
        plot: 'The dark side of the Force is stronger.',
        writer: 'Darth Vader',
        upvotes: 49,
        voted: false,
      },
      {
        plot: 'One does not simply walk into Mordor',
        writer: 'Boromir',
        upvotes: 74,
        voted: false,
      },
    ],
  }
})
```



Prêt à voter !

Nous avons mis en place, avec l'utilisation de méthodes, le système de vote. Je pense que cela semble bon, alors nous pouvons continuer avec la partie « histoire préférée ». Nous voulons que l'utilisateur puisse choisir une seule histoire à ajouter comme favorite. La première chose qui me vient à l'esprit est d'ajouter un nouvel objet vide (favori) Et chaque fois que l'utilisateur choisit une histoire en favori, la variable 'favori' est mise à jour. De cette façon, nous serons en mesure de vérifier si une histoire est égale à l'histoire préférée de l'utilisateur. Faisons cela.

```
<template id="story-template">
  <li class="list-group-item">
    {{ story.writer }} said "{{ story.plot }}".
    Story upvotes {{ story.upvotes }}.
    <button v-show="!story.voted" @click="upvote"
      class="btn btn-default">
      Upvote
    </button>
    <button v-show="!isFavorite" @click="setFavorite"
      class="btn btn-primary">
      Favorite
    </button>
    <span v-show="isFavorite"
      class="glyphicon glyphicon-star pull-right" aria-hidden="true">
```

```
</span>
</li>
</template>

Vue.component('story', {
  template: "#story-template",
  props: ['story'],
  methods: {
    upvote: function(){
      this.story.upvotes += 1;
      this.story.voted = true;
    },
    setFavorite: function(){
      this.favorite = this.story;
    },
  },
  computed: {
    isFavorite: function(){
      return this.story == this.favorite;
    },
  }
});

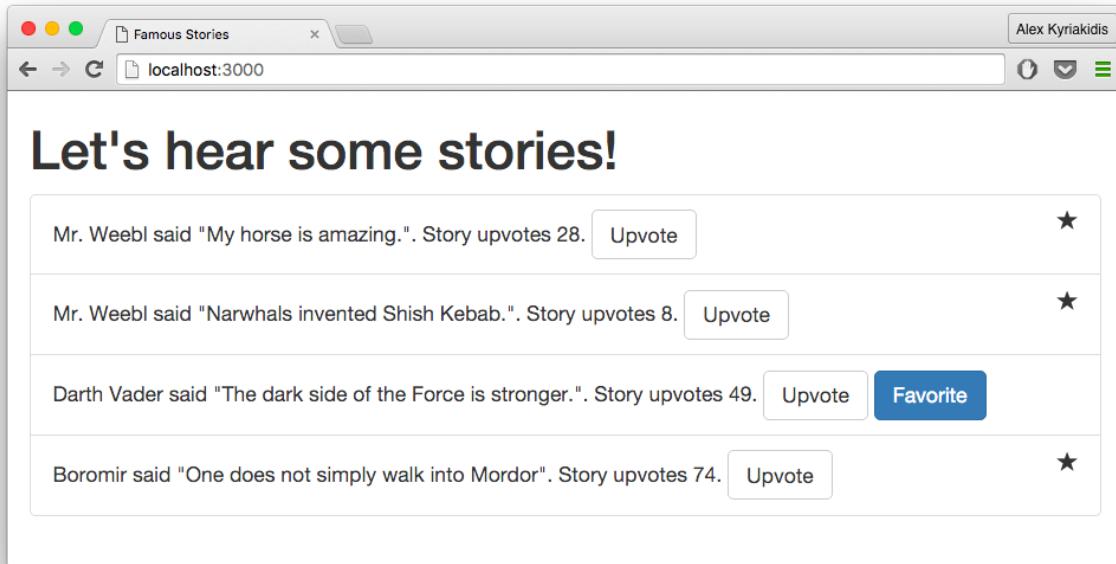
new Vue({
  el: '#app',
  data: {
    stories: [
      ...
    ],
    favorite: {}
  }
})
```

Si vous essayez d'exécuter le code ci-dessus, vous remarquerez qu'il ne fonctionne pas comme il se doit. Chaque fois que vous tentez de mettre une histoire en favori, la variable **favori** dans **\$data** reste nul.

Il semble que notre composant **story** ne puisse pas mettre à jour l'objet **favori**, alors nous allons le transmettre sur chaque histoire et ajouter **favori** aux propriétés du composant.

```
<ul class="list-group">
  <story v-for="story in stories"
    :story="story"
    :favorite="favorite">
  </story>
</ul>
```

```
Vue.component('story', {
  ...
  props: ['story', 'favorite'],
  ...
});
```



Dysfonctionnement de la méthode `setFavorite`

Hmmm, `favorite` n'est toujours pas mis à jour lorsque `setFavorite` est exécuté. Le bouton disparaît comme prévu et une icône en étoile apparaît, mais le la variable favori est toujours nul. Il en résulte que l'utilisateur peut mettre en favori toutes les histoires.

Le problème avec cette approche est que nous ne gardons pas les choses *synchronisées*. Par défaut, tous les accessoires forment une liaison unidirectionnelle entre la propriété enfant et le parent. Lorsque la propriété parent est mise à jour, elle se répercutera sur l'enfant, mais pas l'inverse.

Nous ne pouvons pas synchroniser les données de l'enfant avec les celles parents, avec ce que nous savons jusqu'à présent. Donc, nous allons faire une pause et étudier **Les Événements personnalisés** de Vue.js, avant d'aller plus loin.



Exemples de code

Vous trouverez les exemples de code de ce chapitre sur [GitHub](#)³⁰.

³⁰<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter7>

Devoirs

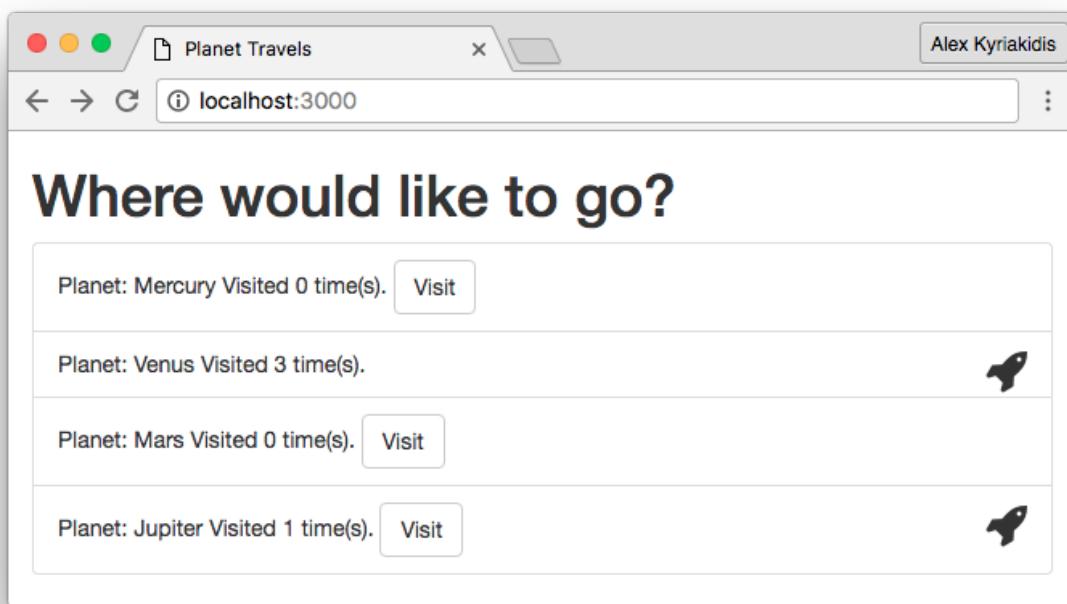
Créer un tableau de planètes. Chaque planète doit avoir un « nom » et un « nombre de visites ».

Vous pouvez choisir de voyager sur n'importe quelle planète, mais vous êtes limité à 3 visites par planète en raison de la pénurie de carburant.

Vous devriez avoir un composant *Planet* avec les méthodes appropriées / propriétés calculées.

Lorsqu'il est rendu, chaque planète doit afficher :

- son nom
- Le nombre de visites
- A bouton *Visiter* (si le nombre maximal de visites n'a pas été atteint)
- Une icône pour indiquer si la planète a été visité au moins une fois



Exemples



Solution potentielle

Vous pouvez trouver une solution potentielle à cet exercice [ici](#)³¹.

³¹<https://github.com/hoottlex/the-majesty-of-vuejs-2/blob/master/homework/chapter7.html>

Événements personnalisés

Parfois, il est nécessaire de déclencher un événement personnalisé. Pour ce faire, nous pouvons utiliser les méthodes d'Instance de Vue.js. Chaque occurrence Vue implémente l'interface [Events] (<http://vuejs.org/api/#Instance-Methods-Events>).

Cela signifie que nous pouvons :

- Écoutez un événement en utilisant `$on(event)`.
- Déclenchez un événement en utilisant `$emit(event)`.

Nous pouvons aussi :

- Écoutez un événement, mais une seule fois, en utilisant `$once (événement)`.
- Supprimez des écouteurs d'événements en utilisant `$ off ()`.

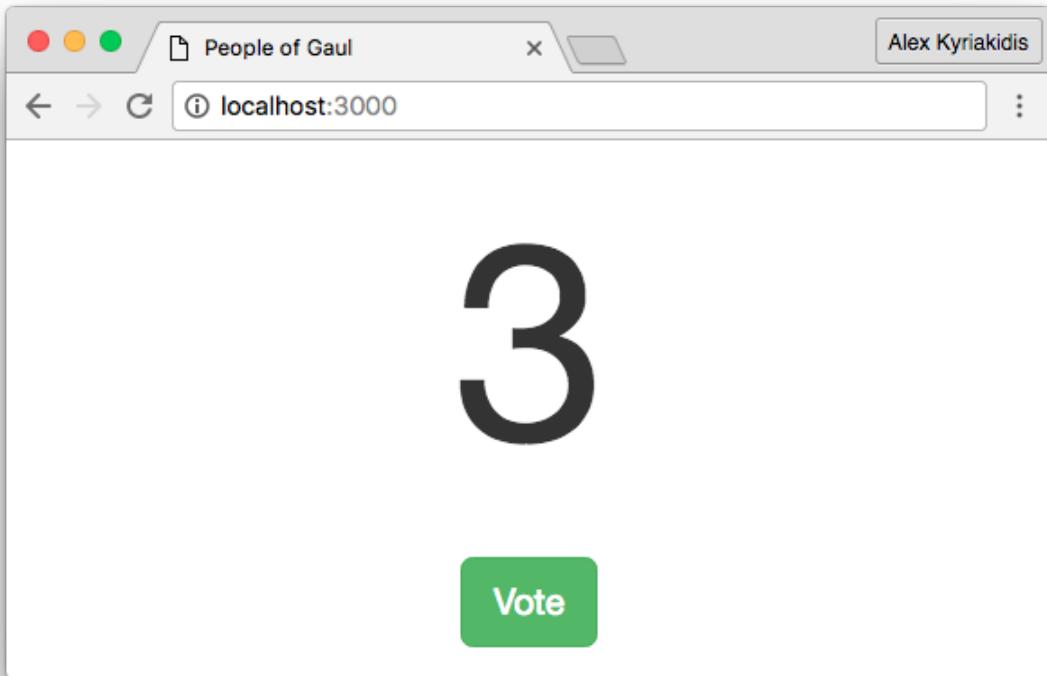
Emission et écoute

Commençons par un exemple simple.

Le bloc de code suivant représente une page avec un compteur et un bouton *Vote*. Lorsque le bouton est cliqué, il émet un événement, nommé '*voted*'. Il ya aussi un *Event Listener* ou écouteur pour l'événement, ce qui augmente le nombre de votes lorsque l'événement est déclenché.

```
1 <html>
2   <head>
3     <title>Emit and Listen</title>
4     <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.\
5       css" rel="stylesheet">
6   </head>
7   <body>
8     <div class="container text-center">
9       <p style="font-size: 140px;">
10         {{ votes }}
11       </p>
12       <button class="btn btn-primary" @click="vote">Vote</button>
13     </div>
14   </body>
```

```
15 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
16 <script type="text/javascript">
17 new Vue({
18   el: '.container',
19   data: {
20     votes: 0
21   },
22   methods:
23   {
24     vote: function (writer) {
25       this.$emit('voted')
26     },
27   },
28   created () {
29     this.$on('voted', function(button) {
30       this.votes++
31     })
32   }
33 })
34 </script>
35 </html>
```



Exemple

Nous enregistrons l'écouteur d'événements dans le crochet de cycle vie (Lifecycle Hook) `created`. `this` est lié à l'Instance Vue.js dans la méthode `vote` et le hook `created`. Nous pouvons donc accéder aux fonctions `$on` et `$emit` en utilisant `this.$on**` et `**this.$emit'`.

Crochets du cycle de vie

Les crochets du cycle de vie sont des fonctions qui s'exécutent lorsque des événements liés à Vue se produisent.

Dans Vue 2, ces crochets ou Hook sont :

Hook	Appelé
beforeCreate	Une fois que l'instance vient d'être initialisée, avant l'observation des données et la configuration des événements / observateurs.
created	Une fois l'instance créée.
beforeMount	Juste avant le montage commence.
mounted	Une fois que l'instance vient d'être montée sur le DOM.
beforeUpdate	Lorsque les données sont modifiées, avant que le DOM virtuel soit rendu et patché.
updated	Après un changement de données, le DOM virtuel est rendu et patché (mis à jour).
activated	Quand un composant maintenu-vivant (kept-alive) est activé.
deactivated	Lorsqu'un composant maintenu en vie (kept-alive) est désactivé.
beforeDestroy	Juste avant qu'une instance de Vue soit détruite.
destroyed	Après qu'une instance Vue n'ai été détruite.

Vous n'avez pas à connaître toutes ces choses, mais il est bon d'être conscient de leur existence. Si vous souhaitez en savoir plus sur les Hooks de Lycle de vie consultez [L'API de Vue.js³²](#).

Communication parents-enfants

Les choses deviennent un peu différentes quand un composant parent doit écouter un événement d'un composant *enfant*. Nous ne pouvons pas utiliser `this.$on` / `this.$emit` puisque `this` sera lié à différentes instances.

Vous rappelez-vous de [L'écouteur d'évenements v-on](#) (** @**) ? Un composant parent peut écouter les événements émis par un composant enfant en utilisant **v-on** directement dans le modèle, où le composant enfant est utilisé.

À la suite de l'exemple précédent, je vais créer un composant aliments (food) qui aura une propriété *name*. Dans son template, il affichera un bouton affichant son *nom*. Lorsque le bouton est cliqué, nous voulons émettre l'événement *vote*.

Food Component

```
Vue.component('food', {
  template: '#food',
  props: ['name'],
  methods: {
    vote: function () {
      this.$emit('voted')
    }
  },
})
```

³²<http://vuejs.org/api/#Options-Lifecycle-Hooks>

Food Component's Template

```
<template id="food">
  <button class="btn btn-default" @click="vote">{{ name }}</button>
</template>
```

Dans l'instance du parent, le `<button>` sera remplacé par `<food @voted ="countVote"> </food>`.

`@voted ="countVote"` signifie que lorsque l'événement `voted` 'de l'enfant est émis, la méthode `*countVote*` sera exécutée. Nous pouvons aussi nous débarrasser de l'écouteur `**this.$On**`, car nous n'en avons plus besoin.

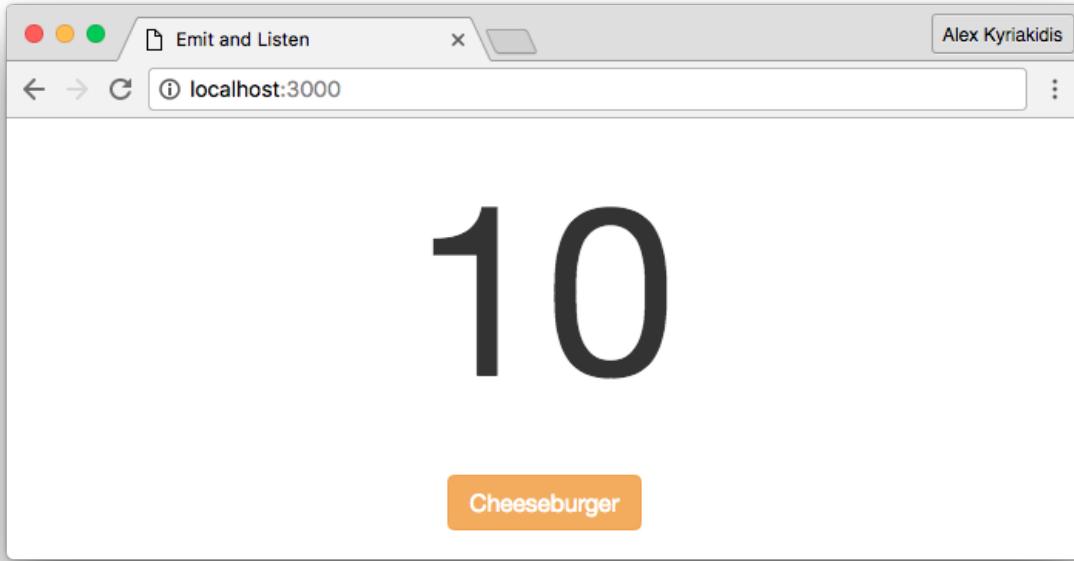
Parent Component

```
new Vue({
  el: '.container',
  data: {
    votes: 0
  },
  methods:
  {
    countVote: function () {
      this.votes++
    }
  }
})
```

Parent Component's template

```
<div class="container text-center">
  <p style="font-size: 140px;">
    {{ votes }}
  </p>
  <food @voted="countVote" name="Cheeseburger"></food>
</div>
```

Si vous exécutez ceci dans votre navigateur, vous verrez que le résultat est la même.



Communication parents-enfants

Passer des arguments

Créons 3 instances du composant *food*. Chaque instance aura son propre nombre de votes. Lorsque l'on vote pour l'un des aliments ou food, il augmentera ses propres votes et il émet un événement pour mettre à jour le total des votes, situé dans la composante parent.

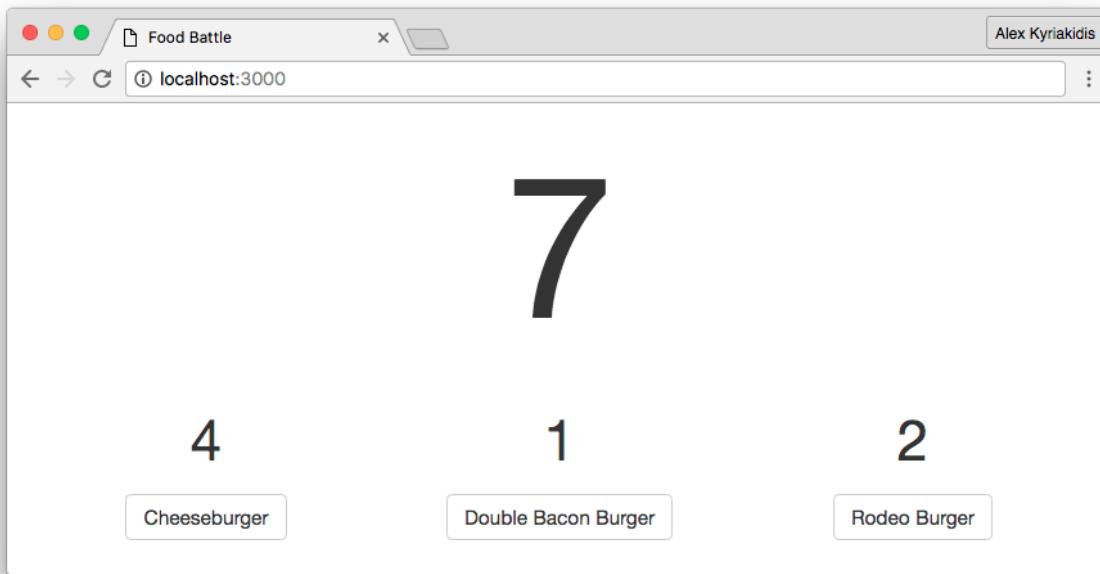
```
1 <html>
2   <head>
3     <title>Food Battle</title>
4     <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.\
5       css" rel="stylesheet">
6   </head>
7   <body>
8     <div class="container text-center">
9       <p style="font-size: 140px;">
10         {{ votes }}
11       </p>
12
13     <div class="row">
14       <food @voted="countVote" name="Cheeseburger"></food>
```

```
15      <food @voted="countVote" name="Double Bacon Burger"></food>
16      <food @voted="countVote" name="Rodeo Burger"></food>
17  </div>
18 </div>
19
20 </body>
21 <template id="food">
22   <div class="text-center col-lg-4">
23     <p style="font-size: 40px;">
24       {{ votes }}
25     </p>
26     <button class="btn btn-default" @click="vote">{{ name }}</button>
27   </div>
28 </template>
29 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
30 <script type="text/javascript">
31 var bus = new Vue()
32
33 Vue.component('food', {
34   template: '#food',
35   props: ['name'],
36   data: function () {
37     return {
38       votes: 0
39     }
40   },
41   methods: {
42     vote: function () {
43       this.votes++
44       this.$emit('voted')
45     }
46   }
47 })
48 new Vue({
49   el: '.container',
50   data: {
51     votes: 0
52   },
53   methods:
54   {
55     countVote: function () {
56       this.votes++
```

```

57      }
58    }
59  })
60 </script>
61 </html>

```



Plusieurs instances de composants

Rien de nouveau à ce jour. Pour ajouter un peu de fantaisie à notre application , nous pouvons ajouter un *Log de Vote*. Le journal des logs sera mis à jour chaque fois que l'on vote pour un aliment. Nous devons mettre à jour la composante enfant, pour transmettre le nom de l'aliment lors de l'émission de l'événement « voted ».



Information

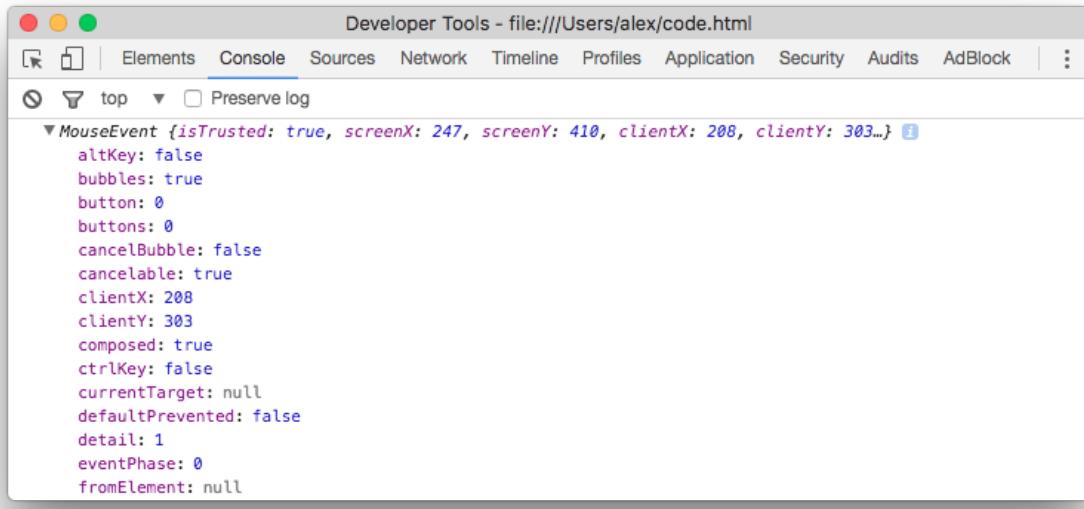
La fonction `$emit`, ainsi que l'argument `event name`, transmet tous les arguments supplémentaires à la fonction `callback` de l'écouteur. Par exemple : '`vm.$Emit ('voted', 'Alex', 'Sunday', 'Bob Ross')`'

Nous avons deux options pour accéder au nom de l'aliment. La plus évidente est évidemment à partir de la propriété `name` du composant. La seconde consiste à accéder à l'élément qui a déclenché l'événement et à trouver son contenu textuel. Nous allons passer à la deuxième.

Nous pouvons logger la variable `event` dans la console, dans la méthode `vote` de `Food` , pour savoir comment nous pouvons accéder à l'élément cliqué.

Food Component

```
Vue.component('food', {  
  ...  
  methods: {  
    vote: function(event) {  
      console.log(event)  
      this.votes++  
      this.$emit('voted')  
    }  
  }  
})
```



event.srcElement

Si vous suivez, vous verrez que nous avons accès à l'élément cliqué dans l'attribut `event.srcElement`. Le nom peut être trouvé dans `event.srcElement.outerText` et `event.srcElement.textContent`.

Donc, passons l'un des deux à la fonction `$emit`.

Food Component

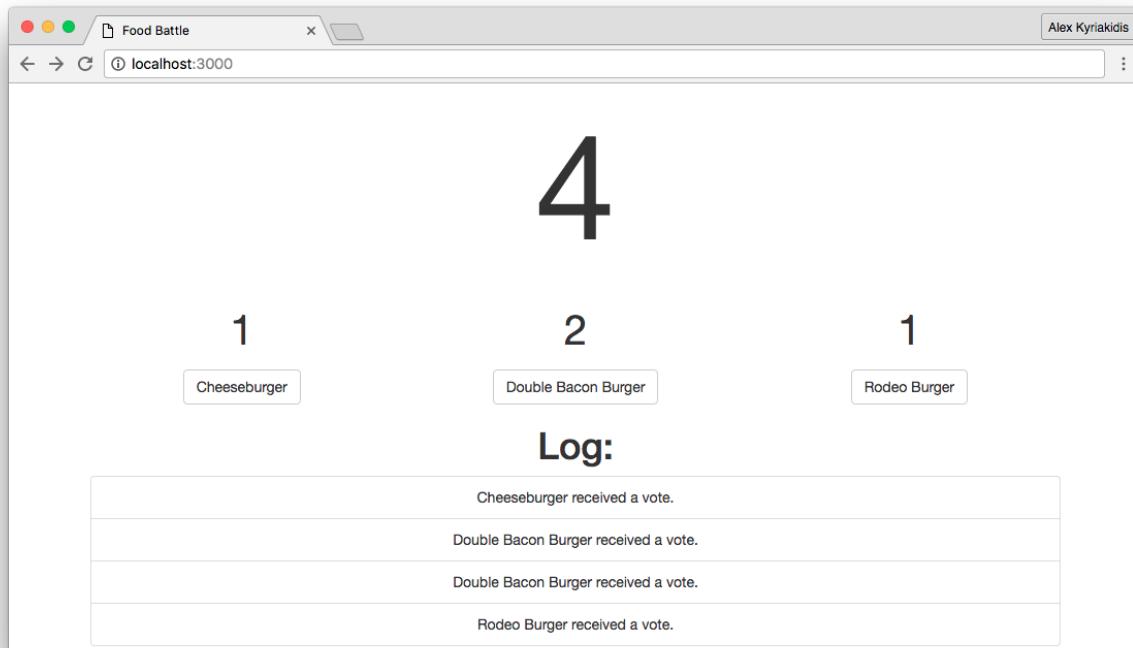
```
Vue.component('food', {
  ...
  methods: {
    vote: function () {
      this.votes++
      this.$emit('voted', event.srcElement.textContent)
    }
  }
})
```

Au sein du parent, nous pusher le vote entrant vers un tableau log.

```
new Vue({
  el: '.container',
  data: {
    votes: 0,
    log: []
  },
  methods: {
    countVote: function (food) {
      this.votes++
      this.log.push(food + ' received a vote.')
    }
  }
})
```

Pour afficher le log, nous pouvons ajouter une liste au template HTML.

```
<h1>Log:</h1>
<ul class="list-group">
  <li class="list-group-item" v-for="vote in log"> {{ vote }} </li>
</ul>
```



Log des votes

Assez facile, hein ?

Communication non-parent-enfant

Nous allons reprendre l'exemple précédent en ajoutant un bouton de réinitialisation. Lorsqu'il est cliqué, il réinitialise tous les compteurs de vote. Comme vous pouvez l'imaginer, le bouton de réinitialisation émet un événement qui devrait être géré par tous les composants. Mais comment pouvons-nous capter cet événement dans les composantes enfant ?

Si nous utilisons la méthode `<food @voted = " countVote ">`, nous pouvons écouter l'événement 'voted', mais nous n'avons pas un moyen d'émettre des événements vers les composants enfants.

Pour rendre tous les composants capables de communiquer les uns avec les autres, nous utiliserons une instance Vue vide comme un bus central d'événements . Ensuite, dans le Hook `created` des composants , nous enregistrerons les écouteurs d'événements en utilisant `bus.$on` au lieu de `this.$on`. Par conséquent, nous utiliserons `bus.$emit` pour déclencher tous les événements.

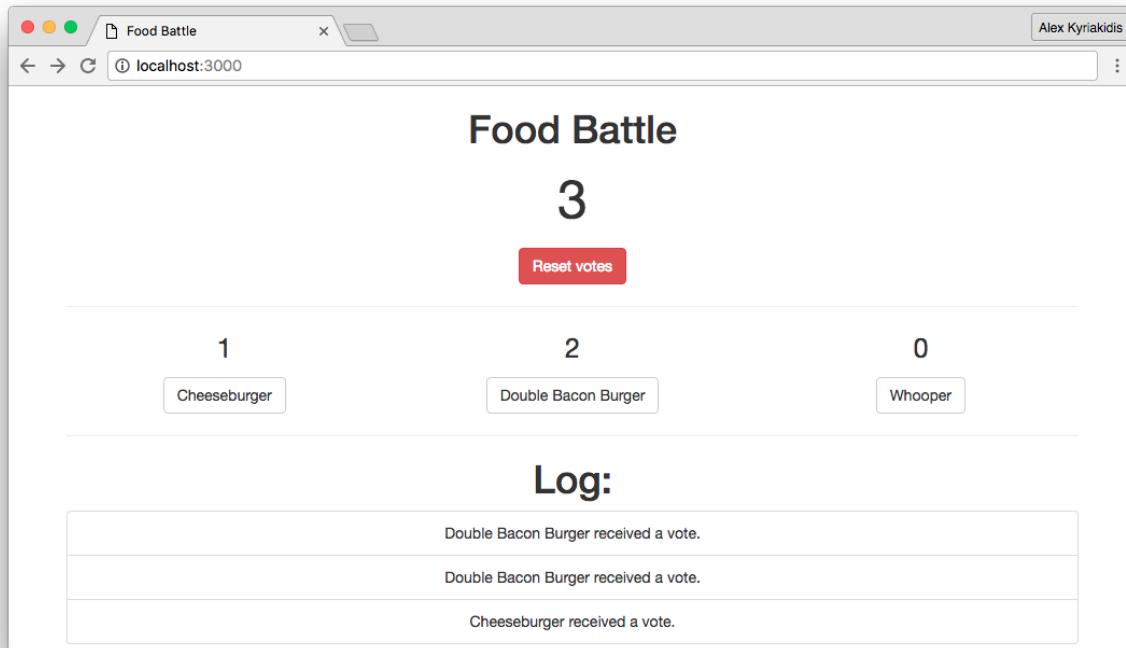
HTML

```
1 <body>
2   <div class="container text-center">
3     <h1>Food Battle</h1>
4     <p style="font-size: 140px;">
5       {{ votes.count }}
6     </p>
7     <button class="btn btn-danger" @click="reset">Reset votes</button>
8     <hr>
9
10    <div class="row">
11      <food name="Cheeseburger"></food>
12      <food name="Double Bacon Burger"></food>
13      <food name="Whooper"></food>
14    </div>
15    <hr>
16
17    <h1>Log:</h1>
18    <ul class="list-group">
19      <li class="list-group-item" v-for="vote in votes.log"> {{ vote }} </li>
20    </ul>
21  </div>
22 </body>
```

JavaScript

```
1 var bus = new Vue()
2
3 Vue.component('food', {
4   template: '#food',
5   props: ['name'],
6   data: function () {
7     return {
8       votes: 0
9     }
10   },
11   methods: {
12     vote: function (event) {
13       // instead of using this.name
14       // we can access event's element's text
15       var food = event.srcElement.textContent;
```

```
16      this.votes++
17      bus.$emit('voted', food)
18  },
19  reset: function () {
20      this.votes = 0
21  }
22 },
23 created () {
24     bus.$on('reset', this.reset)
25 }
26 })
27 new Vue({
28     el: '.container',
29     data: {
30         votes: {
31             count: 0,
32             log: []
33         }
34     },
35     methods:
36     {
37         countVote: function (food) {
38             this.votes.count++
39             this.votes.log.push(food + ' received a vote.')
40         },
41         reset: function () {
42             this.votes = {
43                 count: 0,
44                 log: []
45             }
46             bus.$emit('reset')
47         }
48     },
49     created () {
50         bus.$on('voted', this.countVote)
51     }
52 })
```



Communication non-parent-enfant



Avertissement

Notez ici que nous utilisons :

```
bus.$on('voted', this.countVote)
```

Si au lieu de ça, nous essayions de l'utiliser comme suite :

```
bus.$on('voted', function(){
    this.vote(food)
})
```

Il aurait déclenché une erreur, puisque `this` serait limité à l'instance * bus * au lieu de l'instance du composant en cours.

Suppression d'écouteurs d'événements

Pour supprimer un ou plusieurs écouteurs d'événements, nous pouvons utiliser `$off`. La méthode `$off([event, callback])` peut être utilisée de plusieurs façons.

1.`$off()`, sans arguments, supprime tous les écouteurs d'événements. 2.`$off([event])`, supprime tous les écouteurs d'événement de l'événement spécifié. 3.`$off([event, callback])` supprime l'écouteur de l'événement pour le callback spécifique.

Pour le voir en action, nous ajouterons un bouton `Stop` pour empêcher les votes d'être comptés / enregistrés. Nous placerons la méthode `stop` dans l'instance de Vue.

```
new Vue({
  ...
  methods:
  {
    ...
    stop: function () {
      bus.$off(['voted'])
    }
  }
})
```

Si nous regardons `*bus.$Off(['voté'])`, vous voyez qu'après avoir cliqué sur le bouton `*Stop`, les votes à venir ne sont pas ajoutés au total. En outre, ils ne s'affichent pas dans le journal. Cependant, si vous cliquez sur le bouton `Réinitialiser les votes`, les votes des composantes d'aliments sont réinitialisés à 0.

Pour désactiver l'écouteur `reset`, nous pouvons également supprimer tous les écouteurs d'événements en utilisant `bus.$Off()`.

Retour aux histoires

Rappelez-vous [L'exempleStories] (# storiesex) des chapitres précédents ? Nous étions sur le point de synchroniser *les données du composant* avec *les données des parents*. La solution semble évidente maintenant.

Nous utiliserons le composant `Story` comme ceci :

```
<story v-for="story in stories"
:story="story"
:favorite="favorite"
@update="updateFavorite"></story>
```

Dans le composant `Story`, nous allons émettre l'événement `update` lorsque l'histoire est marquée comme favori. L'histoire étant préférée, devrait être passée comme un argument lors de l'émission (`emit`) de l'événement.

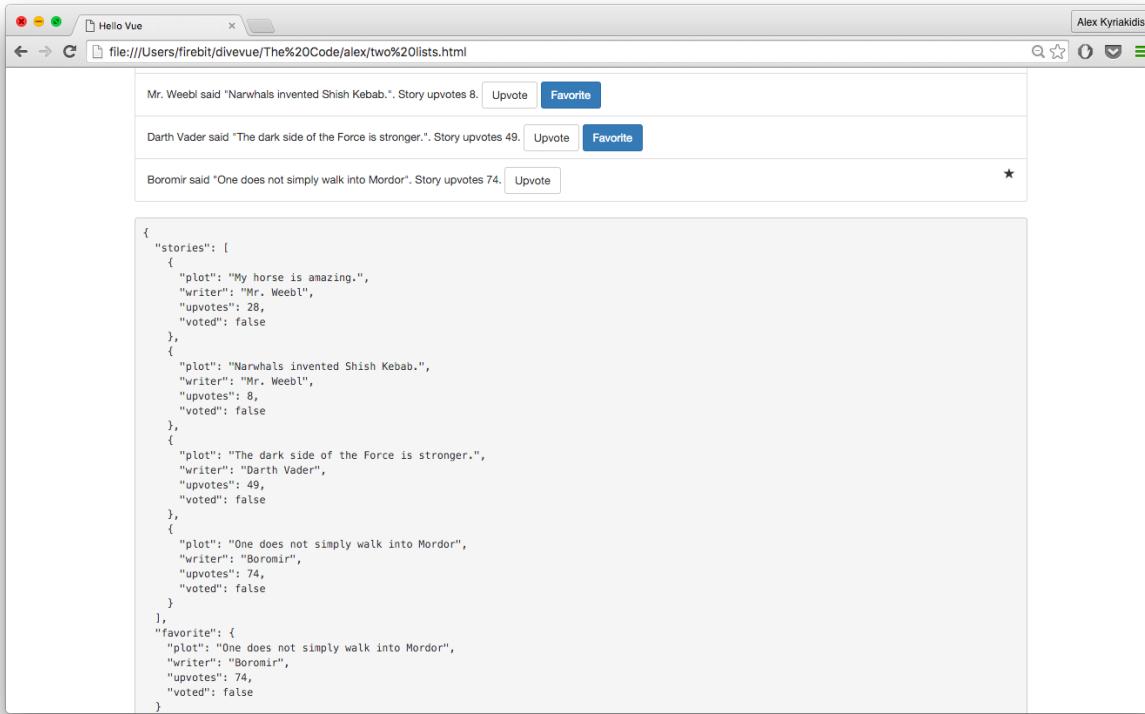
Story Component

```
Vue.component('story', {
  ...
  methods: {
    ...
    updateFavorite: function(){
      // 'update' is just the name of the custom event
      // it could be anything. ex: fav-update
      this.$emit('update', this.story)
    }
  }
  ...
});
```

Dans l’instance parent, nous allons ajouter une variable `favorite` aux données. De plus, nous allons créer une nouvelle méthode qui mettra à jour la variable `favorite` lorsqu’elle sera appelée.

Parent Instance

```
new Vue({
  ...
  data: {
    ...
    favorite: {}
  },
  methods: {
    updateFavorite: function(story) {
      this.favorite = story;
    }
  },
})
```



```
{
  "stories": [
    {
      "plot": "My horse is amazing.",
      "writer": "Mr. Weeb1",
      "upvotes": 28,
      "voted": false
    },
    {
      "plot": "Narwhals invented Shish Kebab.",
      "writer": "Mr. Weeb1",
      "upvotes": 8,
      "voted": false
    },
    {
      "plot": "The dark side of the Force is stronger.",
      "writer": "Darth Vader",
      "upvotes": 49,
      "voted": false
    },
    {
      "plot": "One does not simply walk into Mordor",
      "writer": "Boromir",
      "upvotes": 74,
      "voted": false
    }
  ],
  "favorite": {
    "plot": "One does not simply walk into Mordor",
    "writer": "Boromir",
    "upvotes": 74,
    "voted": false
  }
}
```

Mettre seulement une seule histoire en favori

Maintenant, le résultat souhaité est atteint et l'utilisateur est capable de choisir une seule histoire pour qu'elle soit son histoire favorite, alors qu'il peut voter pour autant d'histoires qu'il veut.



Information

Dans Vue 2, les bindings sont **toujours** unidirectionnelles. Pour conserver les données synchronisées entre Parent et enfant, vous devez utiliser *Les Evenements*.



Exemple de code

Vous trouverez les exemples de code de ce chapitre sur [GitHub](#)³³.

³³<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter8>

Devoirs

C'est l'exercice le plus difficile jusqu'à présent, alors assurez-vous de mettre en œuvre tout ce que vous avez appris dans ce livre. Créez un tableau de 4 chariots tirés par des chevaux. Chaque chariot a un « nom » et un certain nombre de « chevaux » (de 1 à 4). Créez un composant nommé “chariot”. Le composant “chariot” doit afficher le nom du chariot et le nombre de chevaux qu'il a. Il doit également comporter un bouton d'action. Le texte du bouton dépend du chariot actuellement sélectionné.

Plus précisément, le texte du bouton doit être :

- ‘choisir un Chariot’, avant que l'utilisateur ait choisi un chariot
- rejeter des chevaux, quand le chariot a moins de chevaux que le chariot sélectionné
- ‘Louez des chevaux’, quand le chariot a plus de chevaux que le chariot sélectionné
- ‘Riding !’, Lorsque le chariot est le chariot sélectionné (ce bouton doit être désactivé)

L'utilisateur doit être en mesure de choisir un chariot, puis choisir entre n'importe quel chariot qu'il veut.

Scénario d'exemple : L'utilisateur a choisi un chariot avec 2 chevaux et son bouton dit ‘Riding !’. A chariot with 3 Un chariot avec 3 chevaux a un cheval de plus, donc son bouton dit louer des chevaux ». Un chariot avec 1 cheval a un cheval de moins que le chariot de l'utilisateur, ainsi son bouton indique ‘rejeter des chevaux’. Je pense que vous avez saisie l'idée..



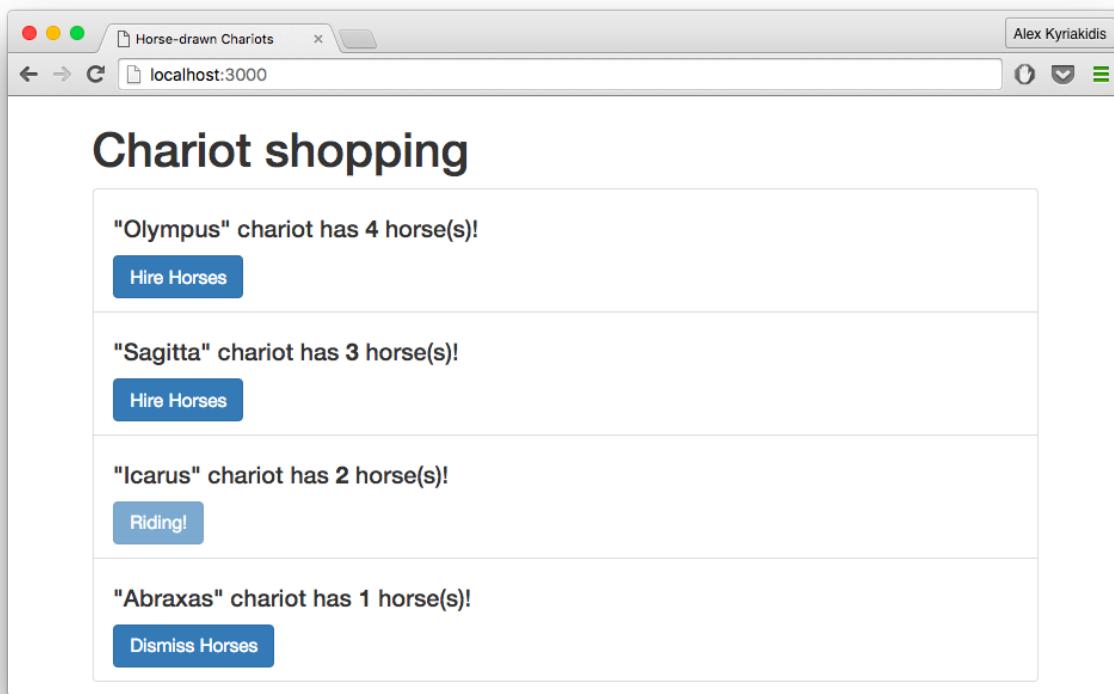
Conseil

Vous devez garder en synchronisation la propriété `currentChariot` de l'enfant et des parents.



Conseil

Pour désactiver un bouton, utilisez l'attribut `disabled = "true"`. Vous devez trouver comment l'appliquer conditionnellement.



Exemple



Solution potentielle

Vous pouvez trouver une potentielle solution à cet exercice [ici](#)³⁴.

³⁴<https://github.com/hoottex/the-majesty-of-vuejs-2/blob/master/homework/chapter8.html>

Binding de class et style

Binding de class

La syntaxe Objet

Un besoin commun du binding de données consiste à manipuler la classe d'un élément et ses styles. Dans des cas pareils, vous pouvez utiliser **v-bind:class**. Cela peut être utilisé pour appliquer des classes conditionnellement, les faire basculer Et / ou appliquer un grand nombre d'entre elles en utilisant un objet bindé.

La directive **v-bind: class** prend un objet avec le format suivant comme argument

```
{  
  'classA': true,  
  'classB': false,  
  'classC': true  
}
```

Et applique toutes les classes qui ont la valeur **true** à l'élément. Par exemple, les classes de l'élément suivant seront **classA** et **classC**.

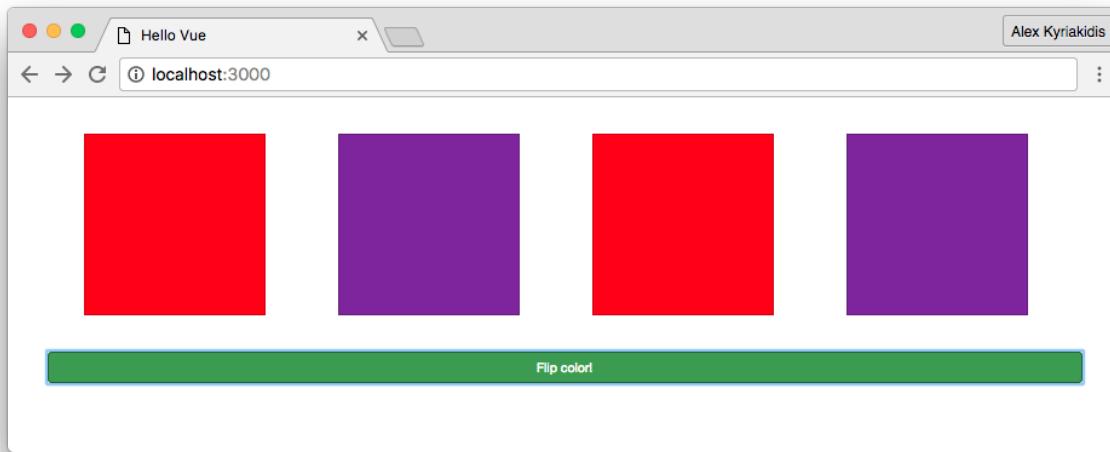
```
<div v-bind:class="elClasses"></div>
```

```
data: {  
  elClasses:  
  {  
    'classA': true,  
    'classB': false,  
    'classC': true  
  }  
}
```

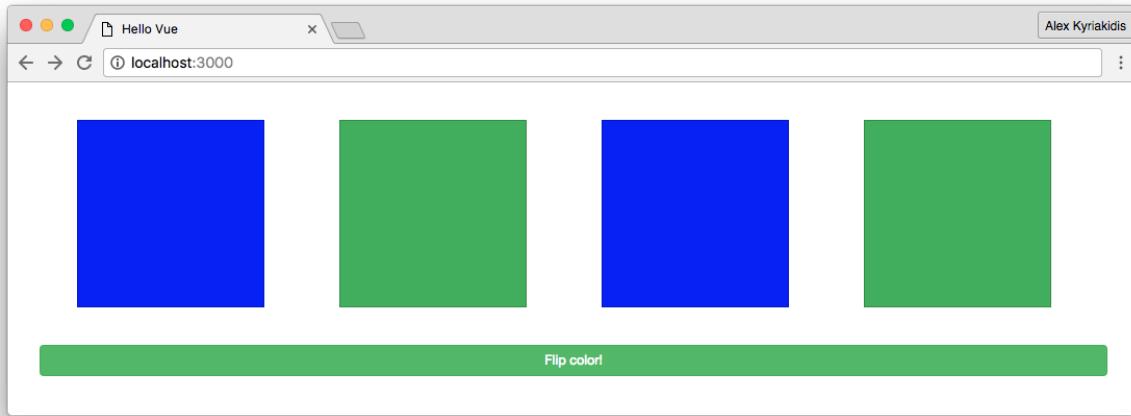
Pour faire la démonstration de comment **v-bind** est utilisé avec les attributs de classe, nous allons faire un exemple de basculement de classe. En utilisant la directive **v-bind:class**, nous allons faire basculer dynamiquement les classes des éléments **div**.

```
1 <html>
2 <head>
3   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.\>
4   css" rel="stylesheet">
5   <title>Hello Vue</title>
6 </head>
7 <body>
8   <div class="container text-center">
9     <div class="box" v-bind:class="{ 'red' : color, 'blue' : !color }"></div>
10    <div class="box" v-bind:class="{ 'purple' : color, 'green' : !color }"></div>
11    <div class="box" v-bind:class="{ 'red' : color, 'blue' : !color }"></div>
12    <div class="box" v-bind:class="{ 'purple' : color, 'green' : !color }"></div>
13    <button v-on:click="flipColor" class="btn btn-block btn-success">
14      Flip color!
15    </button>
16  </div>
17 </body>
18 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
19 <script type="text/javascript">
20 new Vue({
21   el: '.container',
22   data: {
23     color: true
24   },
25   methods: {
26     flipColor: function() {
27       this.color = !this.color;
28     }
29   }
30 });
31 </script>
32 <style type="text/css">
33 .red {
34   background: #ff0000;
35 }
36 .blue {
37   background: #0000ff;
38 }
39 .purple {
40   background: #7B1FA2;
41 }
42 .green {
```

```
43     background: #4CAF50;
44 }
45 .box {
46   float: left;
47   width: 200px;
48   height: 200px;
49   margin: 40px;
50   border: 1px solid rgba(0, 0, 0, .2);
51 }
52 </style>
53 </html>
```



Changement de la couleur des éléments



Changement de la couleur des éléments

Nous avons appliqué une classe de `box` à chaque `div`, pour notre besoin. Ce que ce code fait réellement, c'est d'"inverser" la couleur des div lors ce que le bouton est cliqué. En appuyant sur cette touche, elle appelle la fonction `flipColor`, qui inverse la valeur de `color` initialement à `true`. Alors `v-bind:class` va faire basculer le nom de classe de 'rouge' à 'bleu' ou de 'violet' à 'vert'. Conditionnellement, en fonction de la véracité de la valeur de `couleur`. Cela étant donné, le style va s'appliquer sur chaque classe et nous donner le résultat souhaité.



Information

La directive `v-bind:class` peut coexister avec l'attribut `class` par défaut.

Ce qui fait que dans notre exemple, les `div` ont toujours la class `box` et conditionnellement une des class `red`, `blue`, `purple` Ou `green`.

Syntaxe des tableaux

Nous pouvons également appliquer une liste de classes à un élément, en utilisant un tableau de contenant les noms des class.

```
<div v-bind:class="['classA', 'classB', anotherClass]"></div>
```

Appliquer conditionnellement une classe, peut également être réalisé avec l'utilisation de `if` inline dans le tableau.

```
<div v-bind:class="['classA', condition ? 'classB' : '' ]"></div>
```

Information

La condition if inline est communément appelé **opérateur ternaire**, **opérateur conditionnel**, ou **if ternaire**.

L'opérateur conditionnel (ternaire) est le seul opérateur JavaScript qui prend trois opérandes.

La syntaxe de l'**opérateur ternaire** est **condition? Expression1: expression2**. Si la condition est vraie, l'opérateur retourne la valeur de expression1, sinon, il retourne la valeur de expression2.

En utilisant **if** inline, l'exemple de changement des colors ressemblera à :

```
1 <div class="container text-center">
2   <div class="box" v-bind:class="['color' ? 'red' : 'blue']"></div>
3   <div class="box" v-bind:class="['color' ? 'purple' : 'green']"></div>
4   <div class="box" v-bind:class="['color' ? 'red' : 'blue']"></div>
5   <div class="box" v-bind:class="['color' ? 'purple' : 'green']"></div>
6   <button v-on:click="flipColor" class="btn btn-block btn-success">
7     Flip color!
8   </button>
9 </div>

1 new Vue({
2   el: '.container',
3   data: {
4     color: true
5   },
6   methods: {
7     flipColor: function() {
8       this.color = !this.color;
9     }
10  }
11});
```

Astuce

Pour utiliser un nom réel de classe au lieu d'une variable dans le tableau des classes, utilisez des guillemets simples. **v-bind:class ="[variable, 'classname']"**

Binding de styles

Syntaxe Objet

La syntaxe d'objet pour `v-bind:style` est assez simple ; Il ressemble presque à du CSS, sauf que c'est un objet JavaScript. Nous allons utiliser le raccourci que Vue.js fournit pour la directive précédemment utilisée, `** v-bind(:)**`.

```
1 <!-- shorthand -->
2 <div :style="niceStyle"></div>
```

```
1 data: {
2   niceStyle:
3   {
4     color: 'blue',
5     fontSize: '20px'
6   }
7 }
```

Nous pouvons également déclarer les propriétés de style dans un objet inline `:style = "..."` .

```
<div :style="{ 'color': 'blue', fontSize: '20px'}">...</div>
```

Nous pouvons même défaire référence à une variables à l'intérieur de l'objet de style :

```
<!-- Variable 'niceStyle' is the same we used in the previous example -->
<div :style="{ 'color': niceStyle.color, fontSize: niceStyle.fontSize}">
</div>
```



Binding objet de style

C'est souvent une bonne idée d'utiliser un objet de style et de le binder, afin que le template soit plus propre.

Syntaxe tableau

En utilisant la syntaxe inline de tableau pour `v-bind:style`, nous sommes en mesure d'appliquer plusieurs objets de style au même élément. Ce qui signifie ici que chaque élément de la liste va avoir les propriétés `color` et `fontSize` de l'objet `niceStyle`. Et le style de police de `badStyle`.

```
1 <!-- shorthand -->
2 <div :style="[niceStyle, badStyle]"></div>
```

```
1 data: {
2   niceStyle:
3   {
4     color: 'blue',
5     fontSize: '20px'
6   }
7   badStyle:
8   {
9     fontStyle: 'italic'
10  }
11 }
```



Info pour les intermédiaires

Lorsque vous utilisez une propriété CSS qui requiert des vendor préfixes dans `v-bind:style`, comme par exemple `transform` Vue.js le détecte automatiquement et ajoute les préfixes appropriés aux styles appliqués.

Vous trouverez plus d'informations sur les vendor préfixes [ici](#)³⁵.

Le binding en action

```

1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5 <title>Hello Vue</title>
6 </head>
7 <body class="container-fluid">
8   <div id="app">
9     <ul>
10       <li :class="{ 'completed' : task.done }"
11         :style="styleObject"
12         v-for="task in tasks"
13         {{task.body}}
14         <button @click="completeTask(task)" class="btn">
15           Just do it!
16         </button>
17       </li>
18     </ul>
19   </div>
20 </body>
21 <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/vue/2\.
22 .0.1/vue.js"></script>
23 <script type="text/javascript">
24 new Vue({
25   el: '#app',
26   data: {
27     tasks: [
28       {body: "Feed the horses", done: true},
29       {body: "Wash armor", done: true},
30       {body: "Sharp sword", done: false},
31     ],

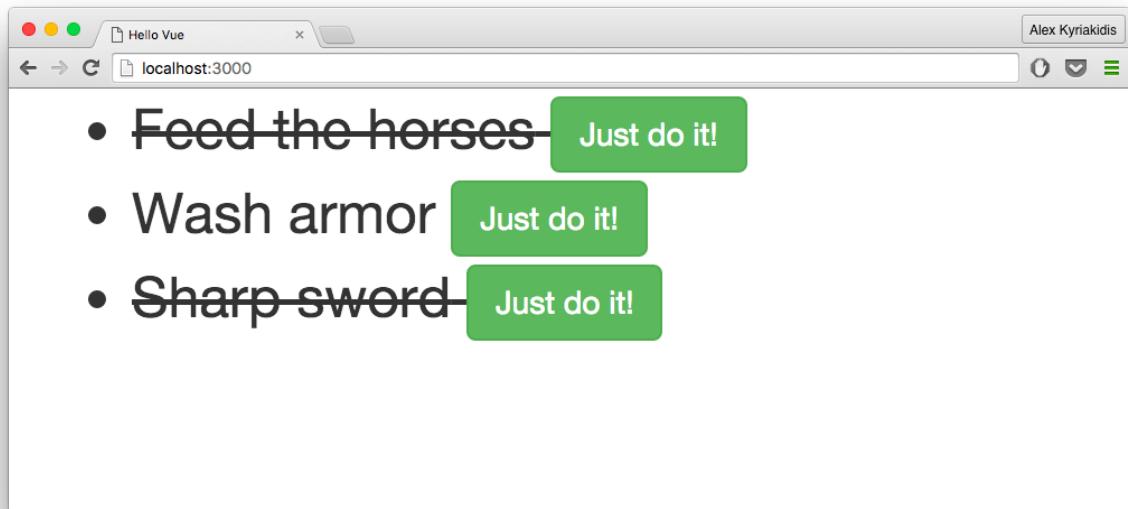
```

³⁵https://developer.mozilla.org/en-US/docs/Glossary/Vendor_Prefix

```
32      styleObject: {
33          fontSize: '25px'
34      }
35  },
36  methods: {
37      completeTask: function(task) {
38          task.done = !task.done;
39      }
40  },
41 });
42 </script>
43 <style type="text/css">
44     .completed {
45         text-decoration: line-through;
46     }
47 </style>
48 </html>
```

//TODO

L'exemple ci-dessus a un tableau d'objets appelé *tasks* et un **styleObject** qui ne contient qu'une seule propriété. Avec l'utilisation de **v-for**, une liste de tâches est rendue et chaque tâche a une propriété *done* avec une valeur booléenne. Selon la valeur de *done*, une classe est appliquée conditionnellement comme précédemment. Si une tâche est terminée, le style css s'applique, et la tâche gagne un **text-decoration** de **line-through**. Chaque tâche est accompagnée d'un bouton, écoutant l'événement *click*, qui déclenche une méthode, modifie Le statut d'achèvement de la tâche. L'attribut **style** est lié à **styleObject**, ce qui entraîne le changement de **fontsize** de toutes les tâches. Comme vous pouvez le voir, la méthode **completeTasks** prend le paramètre**.



Styler les tâches complétées



Exemple de code

Vous trouverez les exemples de code de ce chapitre sur [GitHub³⁶](#).

³⁶<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter9>

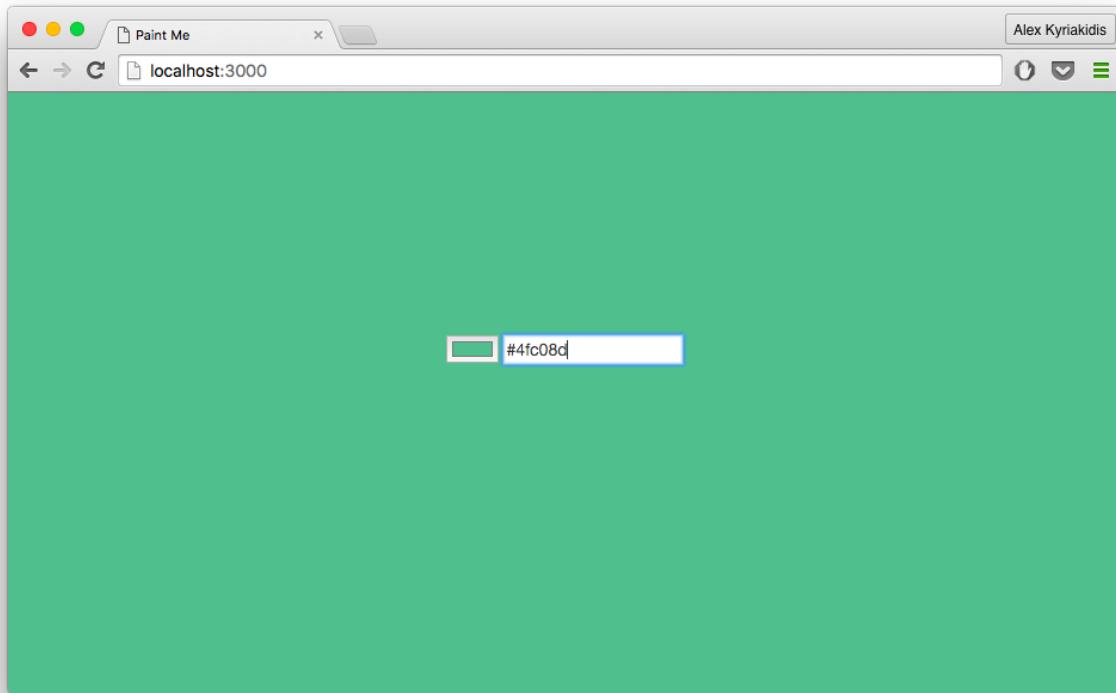
Devoirs

Un exercice amusant et délicat pour ce chapitre. Créez une entrée dans laquelle l'utilisateur peut choisir une couleur. Lorsqu'une couleur est choisie, appliquez-la à un élément de votre choix. Voilà, nous allons peindre !!



Conseil

Vous pouvez utiliser `input type="color"` pour vous faciliter la tâche (pris en charge dans la plupart des navigateurs).



Exemple



Solution potentielle

Vous pouvez trouver une solution à cet exercice [ici](#)³⁷.

³⁷<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter9.html>

Consommation d'une API

Preface

Dans ce chapitre, nous allons allez de l'avant et montrer comment nous pouvons utiliser Vue.js pour consommer une API.

En suivant les exemples *d'histoire* des chapitres précédents, nous allons maintenant utiliser des données réelles provenant d'une source externe.

Afin d'utiliser des données réelles, nous devons utiliser une base de données. En supposant que vous savez déjà comment créer une base de données, elle ne sera pas traitée dans ce livre. Pour travailler avec les exemples du livre, nous en avons déjà créé une pour vous.

CRUD

Supposons que nous avons une base de données, nous devons effectuer des opérations CRUD (Créer, Lire, Mettre à jour, Supprimer). Plus particulièrement, nous voulons

- **Créer** de nouvelles histoires dans la base de données
- **Lire** les histoires existantes
- **Mise à jour** des détails d'une histoire existante (comme ‘upvotes’)
- **Supprimer** les histoires que nous n'aimons pas

Etant donné Vue.js est un framework JavaScript Front-end, il ne peut pas se connecter à une base de données directement. Pour accéder à une base de données, nous avons besoin d'une couche entre Vue.js et la base de données. Cette couche est l'API (Application Program Interface).

API

Parce que ce livre parle de Vue.js et non de la conception d'API, nous vous fournirons une API de démonstration créée avec [Laravel³⁸](#). Laravel est l'un des frameworks PHP les plus puissants avec Symfony2, Nette, CodeIgniter et Yii2. Vous êtes libre de créer votre API en utilisant *n'importe quel language ou framework* que vous aimez. J'utilise Laravel parce qu'il est simple, a une grande communauté, et aussi parce qu'il est génial!

Par conséquent, nous vous recommandons fortement d'utiliser l'API *demo* que nous avons construite exclusivement pour les exemples de ce livre.

³⁸<https://laravel.com/>

Télécharger le code du livre

Pour utiliser notre API, vous devez télécharger le code du livre et démarrer un serveur. Pour ce faire, suivez les instructions ci-dessous.

1. Ouvrez votre terminal et créez un répertoire (nous allons créer ‘~/themajestyofvuejs2’)

```
>_ mkdir ~/themajestyofvuejs2
```

2.Téléchargez le code source sur github

```
>_ cd ~/themajestyofvuejs2  
git clone https://github.com/hootlex/the-majesty-of-vuejs-2.
```

Si non, vous pouvez visiter le dépôt sur [github³⁹](#) et télécharger le fichier zip, puis extraire son contenu dans le répertoire nouvellement créé

2.Accédez au chapitre en cours dans le dossier ‘apis’ du répertoire nouvellement créé.

```
>_ cd ~/themajestyofvuejs2/apis/stories
```

3.Exécutez le script d’installation.

```
>_ sh setup.sh
```

4.Vous disposez maintenant d’une base de données remplie de données factices ainsi que d’un serveur entièrement fonctionnel auquel vous pouvez accéder sur **http://localhost:3000!**

Si vous souhaitez personnaliser le serveur (hôte, port, etc.), vous pouvez effectuer la configuration manuellement. Ci-dessous figure le code source de notre script.

³⁹<https://github.com/hootlex/the-majesty-of-vuejs-2>

Installation Script : setup.sh

```
# Accédez au répertoire de chapitre
$ cd ~/themajestyofvuejs2/apis/stories

# installation des dépendances
$ composer install

# Creation de la base de données
$ touch database/database.sqlite;

# Seed et migration des données
$ php artisan migrate;
$ php artisan db:seed;

# Lancement du serveur
$ php artisan serve --port=3000 --host localhost;
```

Super! Vous disposez maintenant d'une *API entièrement fonctionnelle* et d'une base de données remplie de belles histoires.



Remarque

Si vous utilisez Vagrant, vous devez exécuter le serveur sur le host '0.0.0.0'. Ensuite, vous pourrez accéder à votre serveur sur l'ip de la box Vagrant.

Si, par exemple, l'IP de la box Vagrant est *'192.168.10.10 * et que vous exécutez

```
$ php artisan service --port=3000 --host 0.0.0.0;
```

vous pouvez accéder à votre site web à l'adresse 192.168.10.10: 3000.

Si vous avez téléchargé notre API de démonstration, vous pouvez passer à la section suivante.

Si vous avez choisi de créer votre propre API, vous devez créer une table de base de données pour stocker les stories. Les colonnes suivantes doivent être présentes.

Nom de la colonne	Type
<code>id</code>	Integer, Auto Increment
<code>plot</code>	String
<code>writer</code>	String
<code>upvotes</code>	Integer, Unsigned

N'oubliez pas d'y ajouter quelques données fictives pour être en mesure de suivre les exemples suivants.

endpoints d'API

Un endpoint n'est autre qu'une simple URL. Lorsque vous allez sur `http://example.com/foo/bar`, il s'agit d'un endpoint et vous devez simplement appeler `/foo/bar` car le domaine sera le même pour tous les endpoints au final.

Pour gérer la ressource **Story**, nous avons besoin de 5 endpoints. Chaque paramètre correspond à une action spécifique.

HTTP Method	URI	Action
GET/HEAD	api/stories	<i>Récupération</i> de toutes les stories
GET/HEAD	api/stories/{id}	<i>Récupération</i> d'un story spécifique
POST	api/stories	<i>Creation</i> d'une nouvelle story
PUT/PATCH	api/stories/{id}	<i>Mise à jour</i> d'une story
DELETE	api/stories/{id}	<i>Suppression</i> d'une story spécifique

Comme indiqué dans le tableau ci-dessus, pour obtenir une liste avec toutes les « histoires », nous devons faire une requête HTTP GET ou HEAD vers `Api/stories`. Pour mettre à jour une story existante, nous devons faire une requête HTTP PUT ou PATCH vers `api/stories/{storyID}` en fournissant les données que nous voulons mettre à jour et en remplaçant `{storyID}` par l'id de l'histoire que nous voulons mettre à jour. La même logique s'applique à tous les endpoints. Je pense que vous avez compris l'idée.

En supposant que votre serveur est lancé sur `http://localhost:3000`, vous pouvez afficher une liste de toutes les histoires au format JSON en visitant l'URL `http://localhost:3000/api/stories` dans votre navigateur Web.

```

[{"id": 2, "plot": "Anakin, you're breaking my heart! And you're going down a path I cannot follow!", "upvotes": "586", "writer": "Padme"}, {"id": 3, "plot": "Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations.", "upvotes": "384", "writer": "C-3PO"}, {"id": 4, "plot": "Help me Obi-Wan Kenobi, you're my only hope.", "upvotes": "896", "writer": "Princess Leia"}, {"id": 5, "plot": "I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.", "upvotes": "680", "writer": "Darth Vader"}, {"id": 6, "plot": "The Dark Side of the Force is the pathway to many abilities some consider to be... Unnatural.", "upvotes": "787", "writer": "Senator Palpatine"}]
  
```

Reponse JSON



Astuce

La lecture de données JSON brutes sur le navigateur peut être difficile. Il est toujours plus facile de lire un format JSON **Bien formaté**. Chrome possède de grandes extensions qui pourraient mettre en forme des données JSON brutes dans un format d'arborescence facilement lisible.

J'utilise [JSONFormatter⁴⁰](#) Car il prend en charge la mise en surbrillance de syntaxe et affiche JSON en arborescence, où les nœuds de l'arbre peuvent être parcouru en cliquant sur l'icône triangulaire à gauche de chaque nœud.



Il fournit également un bouton pour afficher les données d'origine (brutes).

Vous pouvez choisir l'extension que vous préférez, mais vous devez **Absolument en utiliser une**

⁴⁰<https://chrome.google.com/webstore/detail/json-formatter/bcjindcccaagfapjjmafapmmgkkhgoa>

Utiliser des données réelles

Il est temps d'utiliser notre base de données et d'effectuer les opérations que nous avons mentionnées (CRUD). Nous utiliserons le [dernier exemple du chapitre Composants](#), Mais cette fois bien sûr, nos données proviendront d'une source externe. Pour échanger des données avec le serveur, nous devons exécuter des requêtes HTTP (Ajax) asynchrones.



Information

AJAX est une technique qui permet aux pages Web d'être mises à jour de manière asynchrone en échangeant de petites quantités de données avec le serveur en coulisses.

Obtenir des données asynchrones

Prenez un moment pour jeter un coup d'œil au [Dernier exemple du chapitre Composants](#). Comme vous pouvez le voir, nous avons coder en dure le **tableau de stories **, à l'intérieur de l'objet data de l'instance Vue.

Stories array hardcoded

```
new Vue({
  data: {
    stories: [
      {
        plot: 'My horse is amazing.',
        writer: 'Mr. Weebly',
      },
      {
        plot: 'Narwhals invented Shish Kebab.',
        writer: 'Mr. Weebly',
      },
      ...
    ]
  }
})
```

Cette fois, nous voulons récupérer les histoires existantes du serveur.

Pour ce faire, nous allons effectuer une requête HTTP GET, en utilisant d'abord jQuery. Plus loin dans ce chapitre, nous allons migrer vers [vue-resource⁴¹](#). Pour voir les différences entre les deux façons de faire.

Pour faire l'appel AJAX, nous allons utiliser `$.get()`, une fonction jQuery qui charge les données du serveur à l'aide d'une requête HTTP GET. La documentation complète pour `$.get()` peut être trouvée [ici⁴²](#).

Information

`vue-resource` est un plugin pour `Vue.js` qui fournit des services pour créer des requêtes Web et gérer les réponses.

La syntaxe de la méthode `$.get()` est

```
$.get(  
    url,  
    success  
)
```

Qui est en fait un raccourci pour

```
$.ajax({  
    url: url,  
    success: success  
)
```

Alors, que devons nous faire maintenant ? Nous voulons obtenir les histoires du serveur en utilisant `$.get('/api/stories')`, et stocker les données de réponse dans le tableau `stories`.

Ici il y'a une astuce, nous devons faire l'appel **après que l'instance soit prête**. Vous souvenez vous des [Hooks de cycle de vie de Vue.js](#) ?

Il y a un *hook*, appelé `mounted`, qui est appelé juste après que l'instance ai été montée.

⁴¹<https://github.com/vuejs/vue-resource>

⁴²<https://api.jquery.com/jquery.get/>



Avertissement

Le hook `mounted` n'est pas équivalent à `$(document).ready()` de jQuery. Lors de l'utilisation de `mounted`, il n'y a aucune garantie d'être dans le document. Si vous devez exécuter quelque chose une fois que la page (DOM) soit prête, vous pouvez utiliser :

```
mounted: function () {
  this.$nextTick(function () {
    // Code qui suppose que this.$el est dans le document
  })
}
```

Voyons cela en action

```
1 <div id="app">
2   <div class="container">
3     <h1>Let's hear some stories!</h1>
4     <ul class="list-group">
5       <story v-for="story in stories" :story="story">
6         </story>
7       </ul>
8       <pre>{{ $data }}</pre>
9     </div>
10 </div>
11 <template id="template-story-raw">
12   <li class="list-group-item">
13     {{ story.writer }} said "{{ story.plot }}"
14     <span>{{story.upvotes}}</span>
15   </li>
16 </template>
```

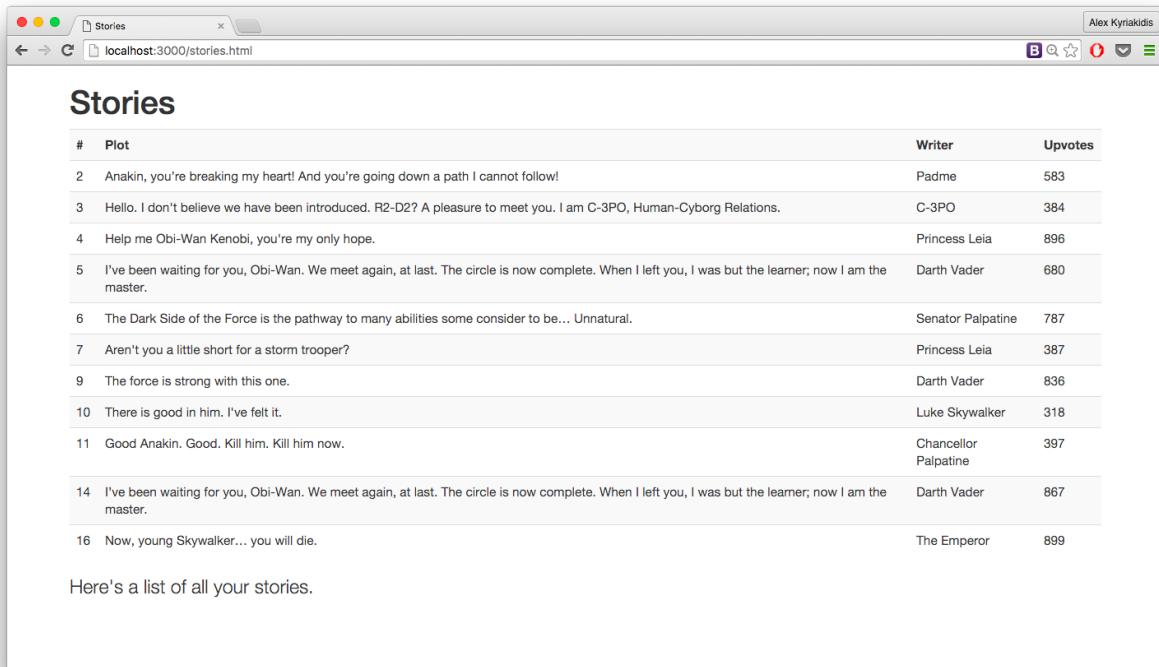
```
1 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
2 <script src="https://code.jquery.com/jquery-2.1.4.min.js"></script>
3 <script type="text/javascript">
4 Vue.component('story', {
5   template: "#template-story-raw",
6   props: ['story'],
7 });
8
9 var vm = new Vue({
```

```

10    el: '#app',
11    data: {
12      stories: []
13    },
14    mounted: function(){
15      $.get('/api/stories', function(data){
16        vm.stories = data;
17      })
18    }
19  })
20 </script>

```

Nous commençons par intégrer jQuery à partir de [cdnjs⁴³](#). Ensuite, à l'intérieur du crochet « monted », nous effectuons la requête GET. Une fois la requête terminée avec succès, nous stockons les données de réponse dans le tableau `stories`.



The screenshot shows a web browser window titled "Stories" with the URL "localhost:3000/stories.html". The page displays a table of 16 stories. The columns are labeled "#", "Plot", "Writer", and "Upvotes". The data is as follows:

#	Plot	Writer	Upvotes
2	Anakin, you're breaking my heart! And you're going down a path I cannot follow!	Padme	583
3	Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations.	C-3PO	384
4	Help me Obi-Wan Kenobi, you're my only hope.	Princess Leia	896
5	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	680
6	The Dark Side of the Force is the pathway to many abilities some consider to be... Unnatural.	Senator Palpatine	787
7	Aren't you a little short for a storm trooper?	Princess Leia	387
9	The force is strong with this one.	Darth Vader	836
10	There is good in him. I've felt it.	Luke Skywalker	318
11	Good Anakin. Good. Kill him. Kill him now.	Chancellor Palpatine	397
14	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	867
16	Now, young Skywalker... you will die.	The Emperor	899

Below the table, a message reads: "Here's a list of all your stories."

Recupération des stories

G Remarquez ici que dans le callback, nous nous référons à la variable `stories` en utilisant `vm.stories` au lieu de `this.stories`. G> Nous le faisons car la variable `this` n'est pas bindée, ou liée à l'instance `Vue` dans le callback. G> Nous sauvegardons alors l'ensemble de l'instance de `Vue` à

⁴³<https://cdnjs.com/libraries/jquery/>

l'intérieur d'une variable appelée `vm`, afin d'y avoir accès à partir de n'importe où dans notre code.
G> Pour en savoir plus sur `this`, consultez [la documentation⁴⁴](#).

Refactoring

Avoir de grandes quantités de code dans notre éditeur de texte, ainsi que dans le navigateur peut être source de confusion si IL n'est pas affichée correctement. Pour cette raison, nous allons refactoriser notre code d'exemple, pour afficher le rendu de la liste de stories en utilisant un élément `<table>` au lieu du ``.

```
1 <div id="app">
2   <table class="table table-striped">
3     <tr>
4       <th>#</th>
5       <th>Plot</th>
6       <th>Writer</th>
7       <th>Upvotes</th>
8       <th>Actions</th>
9     </tr>
10    <story v-for="story in stories" :story="story">
11      </story>
12    </table>
13    <template id="template-story-raw">
14      <tr>
15        <td>
16          {{story.id}}
17        </td>
18        <td>
19          <span>
20            {{story.plot}}
21          </span>
22        </td>
23        <td>
24          <span>
25            {{story.writer}}
26          </span>
27        </td>
28        <td>
29          {{story.upvotes}}
30        </td>
```

⁴⁴https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/L_op%C3%A9rateur_this

```

31      </tr>
32  </template>
33  <p class="lead">Here's a list of all your stories.
34  </p>
35  <pre>{{ $data }}</pre>
36 </div>

```

Mais il y a un problème.

#	Plot	Writer	Upvotes	Actions
1	Anakin, you're breaking my heart! And you're going down a path I cannot follow!	Padme	586	
2	Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations.	C-3PO	384	
3	Help me Obi-Wan Kenobi, you're my only hope.	Princess Leia	896	
4	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	680	
5	The Dark Side of the Force is the pathway to many abilities some consider to be... Unnatural.	Senator Palpatine	787	
6	Aren't you a little short for a storm trooper?	Princess Leia	387	
7	The force is strong with this one.	Darth Vader	836	
8	There is good in him. I've felt it.	Luke Skywalker	318	
9	Good Anakin. Good. Kill him. Kill him now.	Chancellor Palpatine	397	
10	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	867	
11	Now, young Skywalker... you will die.	The Emperor	899	

Here's a list of all your stories.

Problème de rendu

Notre tableau ne s'affiche pas correctement, mais pourquoi ?⁴⁵

Certains éléments HTML, par exemple `<table>`, ont des restrictions sur les éléments qui peuvent apparaître à l'intérieur. Les éléments personnalisés qui ne figurent pas dans la liste des éléments supportés seront hissés et ne seront donc pas affichés correctement. Dans ce cas, vous devez utiliser l'attribut `is` pour indiquer que c'est un élément personnalisé.

Par conséquent, pour résoudre ce problème, nous devons utiliser l'attribut spécial de Vue.js `is`.

⁴⁵<http://goo.gl/Xr9RoQ>

```
<table>
  <tr is="my-component"></tr>
</table>
```

Ainsi notre exemple deviendra

```
<tr v-for="story in stories" is="story" :story="story"></tr>
```

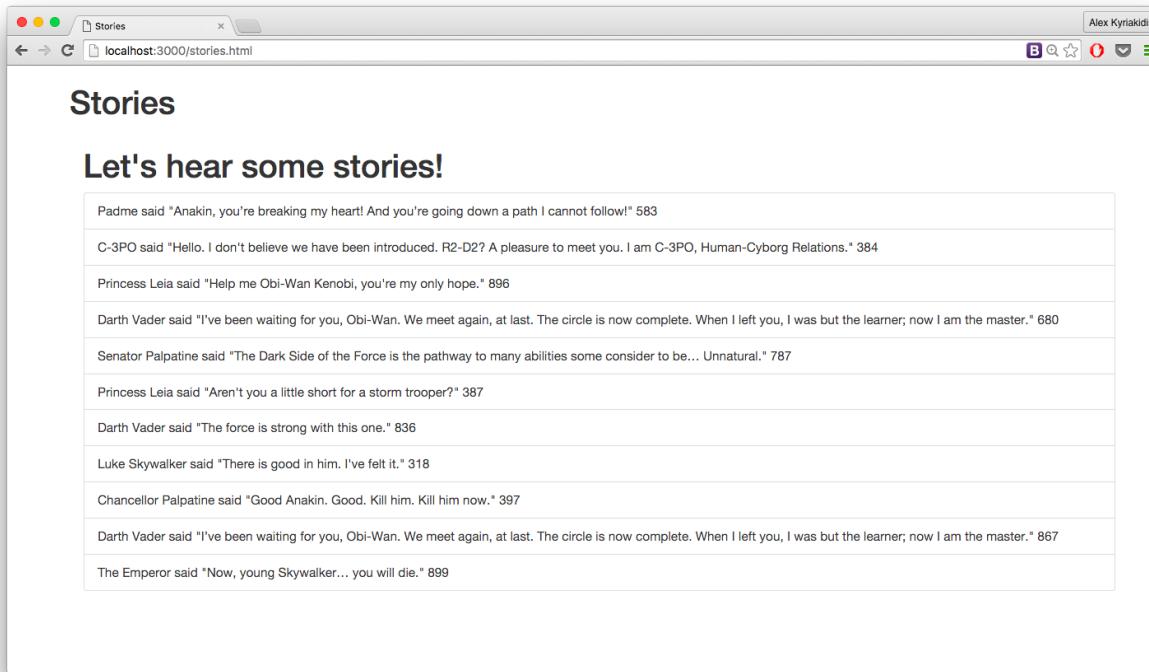


Tableau affiché proprement

Eh bien, ça a l'air mieux !

Mise à jour des données

Nous avions une fonction qui permettait à l'utilisateur de voter pour n'importe quelle histoire qu'il voulait. Mais maintenant nous voulons quelque chose de plus. Nous voulons que le serveur soit informé à chaque fois que quelqu'un vote pour une histoire, ce qui garantit que les votes de l'histoire sont également mis à jour dans la base de données.

Pour mettre à jour une histoire existante, nous devons faire une requête HTTP PATCH ou PUT de type `api/stories/{storyID}`.

Dans la fonction `upvoteStory`, qui doit être créée, nous allons faire un appel HTTP après avoir `voter pour une histoire`. Nous allons passer la nouvelle variable `story` dans le message (payload) de la requête, afin de mettre à jour les données dans notre serveur.

```
1 <td>
2   <div class="btn-group">
3     <button @click="upvoteStory(story)" class="btn btn-primary">
4       Upvote
5     </button>
6   </div>
7 </td>

1 Vue.component('story', {
2   template: '#template-story-raw',
3   props: ['story'],
4   methods: {
5     upvoteStory: function(story){
6       story.upvotes++;
7       $.ajax({
8         url: '/api/stories/'+story.id,
9         type: 'PATCH',
10        data: story,
11      });
12    }
13  },
14 })
```

Nous avons déplacé la méthode `upvote` et l'avons placée dans notre composant `story`. En envoyant une requête de type PATCH et en fournissant les nouvelles données, le serveur met à jour le compte `upvotes`.

#	Plot	Writer	Upvotes	Actions
2	Anakin, you're breaking my heart! And you're going down a path I cannot follow!	Padme	586	<button>Upvote</button>
3	Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations.	C-3PO	384	<button>Upvote</button>
4	Help me Obi-Wan Kenobi, you're my only hope.	Princess Leia	896	<button>Upvote</button>
5	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	680	<button>Upvote</button>
6	The Dark Side of the Force is the pathway to many abilities some consider to be... Unnatural.	Senator Palpatine	787	<button>Upvote</button>
7	Aren't you a little short for a storm trooper?	Princess Leia	387	<button>Upvote</button>
9	The force is strong with this one.	Darth Vader	836	<button>Upvote</button>
10	There is good in him. I've felt it.	Luke Skywalker	318	<button>Upvote</button>
11	Good Anakin. Good. Kill him. Kill him now.	Chancellor Palpatine	397	<button>Upvote</button>
14	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	867	<button>Upvote</button>
16	Now, young Skywalker... you will die.	The Emperor	899	<button>Upvote</button>

Voter pour des histoires

Suppression des données

Passons à une autre fonctionnalité, dans notre liste de stories, nous devrions être en mesure de supprimer une histoire que nous n'aimons pas. Pour supprimer un **story** du tableau et du DOM, nous devons effectuer une recherche et supprimer la story du tableau **stories**.

```

1 <td>
2   <div class="btn-group">
3     <button @click="upvoteStory(story)" class="btn btn-primary">
4       Upvote
5     </button>
6     <button @click="deleteStory(story)" class="btn btn-danger">
7       Delete
8     </button>
9   </div>
10 </td>
```

Nous ajoutons un bouton *Supprimer* à la colonne *actions*, ce bouton sera lié à une méthode pour supprimer l'histoire. La méthode **deleteStory** :

```

1 Vue.component('story', {
2   ...
3   methods: {
4     ...
5     deleteStory: function(story){
6       // find story
7       var index = vm.stories.indexOf(story);
8
9       // delete it
10      vm.stories.splice(index, 1)
11    }
12  }
13  ...
14 })

```

Mais bien sûr, de cette façon, nous allons supprimer l'histoire temporairement. Afin de supprimer l'histoire de la base de données, nous devons effectuer une requête AJAX de type DELETE.

```

1 Vue.component('story', {
2   ...
3   methods: {
4     ...
5     deleteStory: function(story){
6       // find story
7       var index = vm.stories.indexOf(story);
8
9       // delete it
10      vm.stories.splice(index, 1)
11
12      // make DELETE request
13      $.ajax({
14        url: '/api/stories/' + story.id,
15        type: 'DELETE'
16      });
17    },
18  }
19  ...
20 })

```

Nous envoyons les données dans l'URL comme nous l'avons fait auparavant. Le type de requête (verbe HTTP) doit être égal à **DELETE**. Notre méthode est maintenant prête et nous pouvons supprimer l'histoire de notre base de données ainsi que du DOM.

A screenshot of a web browser window titled "Stories". The URL in the address bar is "localhost:3000/stories.html". The browser interface includes standard controls like back, forward, and search, along with a user profile icon for "Alex Kyriakidis". The main content area displays a table with the following data:

#	Plot	Writer	Upvotes	Actions
2	Anakin, you're breaking my heart! And you're going down a path I cannot follow!	Padme	586	<button>Upvote</button> <button>Delete</button>
3	Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations.	C-3PO	384	<button>Upvote</button> <button>Delete</button>
4	Help me Obi-Wan Kenobi, you're my only hope.	Princess Leia	896	<button>Upvote</button> <button>Delete</button>
5	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	680	<button>Upvote</button> <button>Delete</button>
7	Aren't you a little short for a storm trooper?	Princess Leia	387	<button>Upvote</button> <button>Delete</button>
9	The force is strong with this one.	Darth Vader	836	<button>Upvote</button> <button>Delete</button>
10	There is good in him. I've felt it.	Luke Skywalker	318	<button>Upvote</button> <button>Delete</button>
11	Good Anakin. Good. Kill him. Kill him now.	Chancellor Palpatine	397	<button>Upvote</button> <button>Delete</button>

Here's a list of all your stories.

Supprimer et voter pour une histoire

C'est tout pour le moment. Nous continuerons notre exemple dans le prochain chapitre, en améliorant la fonctionnalité et en ajoutant la possibilité de **Créer de nouvelles histoires**, **Éditer une histoires** et plus encore. Mais avant tout, nous remplacerons **jQuery** par **vue-resource**.

Intégration vue-resource

Aperçu

Vue-resource est un plugin de ressource pour Vue.js.

Ce plugin fournit des services pour effectuer des requêtes HTTP et gérer les réponses du serveur en utilisant XMLHttpRequest ou JSONP.

Nous allons créer à nouveau toutes les requêtes Web que nous avons créées ci-dessus, en utilisant ce plugin à la place. De cette façon, vous pouvez le comparer avec Jquery et décider lequel est le mieux adapté à vos besoins. JQuery est agréable, mais si vous l'utilisez uniquement pour effectuer des appels AJAX, vous pouvez envisager de le supprimer.

Vous trouverez ici des instructions pour installer [vue-resource⁴⁶](#). Comme d'habitude, nous allons "l'intégrer" à partir d'un [cdn⁴⁷](#).

Pour extraire des données d'un serveur, nous pouvons utiliser la méthode \$http de vue-resource avec la syntaxe suivante :

```
mounted: function() {
  // GET request
  this.$http({url: '/someUrl', method: 'GET'})
    .then(function (response) {
      // success callback
    }, function (response) {
      // error callback
    });
}
```



Information

Une instance de Vue.js fournit la fonction `this.$Http(options)` qui prend en paramètre un objet d'options pour générer une requête HTTP et renvoie une promesse. De plus, l'instance de Vue sera automatiquement bindée à `this` dans toutes fonctions de callback.

Au lieu de passer notre objet d'options au paramètre, il existe des méthodes abrégées disponibles pour tous les types de requêtes

⁴⁶<https://github.com/vuejs/vue-resource>

⁴⁷<https://cdnjs.com/libraries/vue-resource>

Request shorthands

```

1 this.$http.get(url, [data], [options]).then(successCallback, errorCallback);
2 this.$http.post(url, [data], [options]).then(successCallback, errorCallback);
3 this.$http.put(url, [data], [options]).then(successCallback, errorCallback);
4 this.$http.patch(url, [data], [options]).then(successCallback, errorCallback);
5 this.$http.delete(url, [data], [options]).then(successCallback, errorCallback);

```

Migration

Il est temps d'utiliser vue-resource dans notre exemple. Tout d'abord, nous devons l'inclure. Nous ajouterons cette ligne à notre fichier HTML.

```

1 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue-resource/1.0.3/vue-resou\
2 rce.js"></script>

```

Pour récupérer les histoires, nous allons faire une requête GET dans le formulaire correspondant :

```

mounted: function() {
    // Requette GET
    this.$http({url: '/api/stories', method: 'GET'})
    .then(function (response) {
        Vue.set(vm, 'stories', response.data)
        // Or we as we did before
        // vm.stories = response.data
    })
}

```

Nous récupérons aucun problème la liste d'histoires en utilisant la syntaxe ci-dessus.

maintenant allons de l'avant avec les demandes DELETE et PATCH, en utilisant les méthodes abrégées.

PATCH request

```

upvoteStory: function(story){
    story.upvotes++;
    this.$http.patch('/api/stories/' + story.id, story)
}

```

DELETE request

```
deleteStory: function(story){  
    this.$parent.stories.indexOf(story)  
    this.$parent.stories.splice(index, 1)  
    this.$http.delete('/api/stories/' + story.id )  
}
```

Nous avons remplacé les méthodes AJAX par celles-ci, en un rien de temps !



Information

Puisque le composant *story* n'a pas accès au tableau *stories*, nous accédons au tableau en utilisant `this.$Parent.stories`. Nous pourrions également utiliser `vm.stories` ou émettre un événement pour mettre à jour le tableau dans l'instance Vue parente.

Amélioration des fonctionnalités

Nous devrions ajouter quelques fonctionnalités supplémentaires pour rendre notre liste d'histoires plus propre. Nous pouvons donner à l'utilisateur la possibilité de **changer l'intrigue d'une histoire, son auteur, et aussi créer de nouvelles histoires**.

Edition des histoires

Commençons par la première tâche et donnons à l'utilisateur quelques `input` pour manipuler les propriétés de l'histoire. Deux input bindés doivent faire le travail, mais nous devons les afficher **seulement** quand l'utilisateur **édite** une histoire. Cela semble être le genre de travail que nous avons fait dans les chapitres précédents.

Pour définir si un *story* (histoire) est dans l'**état d'édition**, nous utiliserons une propriété, `editing`, qui deviendra vraie lorsque l'utilisateur clique sur le bouton *Modifier*.

```
1 <td>
2     <!--Si l'utilisateur édite une histoire, afficher l'input de l'intrigue-->
3     <input v-if="story.editing" v-model="story.plot" class="form-control">
4     </input>
5     <!--sinon, afficher l'intrigue de l'histoire-->
6     <span v-else>
7         {{story.plot}}
8     </span>
9 </td>
10 <td>
11     <!-- Si l'utilisateur édite une histoire, afficher l'input pour pour modifie\ r l'auteur -->
12     <input v-if="story.editing" v-model="story.writer" class="form-control">
13     </input>
14     <!--sinon, afficher l'auteur de l'histoire-->
15     <span v-else>
16         {{story.writer}}
17     </span>
18 </td>
19 <td>
20     {{story.upvotes}}
21 </td>
22 <td>
23     <div v-if="!story.editing" class="btn-group">
24         <button @click="upvoteStory(story)" class="btn btn-primary">
25             Upvote
26         </button>
27         <button @click="editStory(story)" class="btn btn-default">
28             Edit
29         </button>
30         <button @click="deleteStory(story)" class="btn btn-danger">
31             Delete
32         </button>
33     </div>
34 </td>
35 </td>
```

```

1 Vue.component('story', {
2   ...
3   methods: {
4     ...
5     editStory: function(story){
6       story.editing=true;
7     },
8   }
9   ...
10 })

```

Ceci est notre table mise à jour, avec deux nouveaux inputs et un bouton. Nous utilisons la fonction `editStory` pour définir `story.editing` à `true`, alors `v-if` Afficher les inputs pour modifier l'histoire et masquer les boutons *Upvote* et *Supprimer*. Cependant, cette approche ne fonctionnera pas. Il semble que le DOM ne soit pas mis à jour après avoir mis `story.editing` à `true`. Mais pourquoi cela arrive-t-il ?

Il s'avère, selon [cet article du blog de Vue.js⁴⁸](#), que lorsque vous ajoutez une nouvelle propriété qui n'était pas présente lorsque les données ont été observées, le DOM ne sera pas mis à jour. La meilleure pratique est de toujours déclarer les propriétés qui doivent être réactives d'avance. Dans les cas où vous devez absolument ajouter ou supprimer des propriétés à l'exécution, utilisez les méthodes globales `Vue.set` ou `Vue.delete`.

Pour cette raison, nous devons initialiser l'attribut `story.editing` à `false` sur chaque histoire, après la réception des histoires du serveur.

Pour ce faire, nous allons utiliser la méthode `.map()` de JavaScript dans le callback de succès de la requête GET.

```

mounted: function() {
  var vm = this;

  // Requette GET
  this.$http({url: '/api/stories', method: 'GET'})
    .then(function (response) {
      var storiesReady = response.data.map(function(story){
        story.editing = false;
        return story
      })

      Vue.set(vm, 'stories', storiesReady)
    })
}

```

⁴⁸<http://vuejs.org/2016/02/06/common-gotchas/>



Information

La méthode `.map()` appelle une fonction de callback prédéfinie sur chaque élément d'un tableau et retourne un tableau contenant les résultats. Vous pouvez trouver plus d'informations sur la méthode `.map()` et sa syntaxe [ici](#)⁴⁹.

Cette fonction ajoute l'attribut `editing` à chaque objet de l'histoire, puis renvoie l'histoire mise à jour. La nouvelle variable, `storiesReady`, est un tableau qui contient notre tableau mis à jour avec le nouvel attribut.

Lorsque l'histoire est en cours d'édition, nous donnerons à l'utilisateur deux options : mettre à jour l'histoire avec de nouvelles valeurs ou annuler la modification.

#	Plot	Writer	Upvotes	Actions
12	Laugh it up, Fuzz ball.	Han Solo	279	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
14	Fear is the path to the dark side.	Yoda	561	
15	I find your lack of faith disturbing.	Darth Vader	582	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
16	Laugh it up, Fuzz ball.	Han Solo	772	<button>Upvote</button> <button>Edit</button> <button>Delete</button>

Entrées de formulaire pour l'édition d'histoires

Ajouter les deux boutons qui ne devraient être affichés que lorsque l'utilisateur édite histoire. En outre, une nouvelle méthode appelée `updateStory` sera créée. Elle va mettre à jour l'histoire éditée, après avoir appuyé sur le bouton *Update story*.

```
<!-- Si l'histoire est en cours d'édition, afficher le groupe de boutons -->


---



49https://msdn.microsoft.com/en-us/library/ff679976\(v=vs.94\).aspx


```

```

1 Vue.component('story', {
2   ...
3   methods: {
4     ...
5     updateStory: function(story){
6       this.$http.patch('/api/stories/'+story.id , story)
7       //Set editing to false to show actions again and hide the inputs
8       story.editing = false;
9     },
10   }
11   ...
12 })

```

#	Plot	Writer	Upvotes	Actions
12	Laugh it up, Fuzz ball.	Han Solo	279	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
14	Fear is the path to the dark side.	Yoda	561	<button>Update Story</button> <button>Cancel</button>
15	I find your lack of faith disturbing.	Darth Vader	582	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
16	Laugh it up, Fuzz ball.	Han Solo	772	<button>Upvote</button> <button>Edit</button> <button>Delete</button>

Modification d'une histoire

C'est en place. Une fois la requête PATCH terminée avec succès, nous devons de nouveau éditer `story.editing` sur `false`, afin de masquer les entrées (inputs) et afficher les boutons d'action.

Créer un nouvelle histoire

Maintenant, pour une tâche un peu plus délicate, nous allons donner à l'utilisateur la possibilité de créer une nouvelle histoire et de l'enregistrer dans notre base de données. Tout d'abord, nous devons fournir des input, de sorte que la nouvelle histoire puisse être ajoutée. Pour ce faire, nous créerons une histoire vide et nous l'ajouterons au tableau `stories` en utilisant la méthode javascript `push()`. Nous allons initialiser toutes les propriétés de l'histoire à `null`, à l'exception de `editing`. Puisque nous voulons manipuler immédiatement la nouvelle histoire, la propriété `editing` sera définie sur `true`.

```
1 var vm = new Vue({
2   ...
3   methods: {
4     createStory: function(){
5       var newStory={
6         "plot": "",
7         "upvotes": 0,
8         "editing": true
9       };
10      this.stories.push(newStory);
11    },
12  }
13 })  
  
1 <p class="lead">Here's a list of all your stories.
2   <button @click="createStory()" class="btn btn-primary">
3     Add a new one?
4   </button>
5 </p>
```



Information

La méthode `push()` ajoute de nouveaux éléments à la fin d'un tableau. Vous pouvez trouver plus d'informations sur la méthode `push()` et sa syntaxe [ici](#)⁵⁰.

Nous avons nommé la nouvelle fonction `createStory` et nous l'avons placée dans notre instance de Vue.

En bas de notre liste, nous avons ajouté un bouton. Lorsque le bouton est cliqué, la méthode `createStory` est appelée. Puisque le `newStory.editing` est défini sur `true`, les inputs binded pour `plot` et `writer` ainsi que les boutons d'action `Edit` sont rendus instantanément.

De plus, nouvel objet `story` doit être envoyé au serveur pour être stocké dans la base de données. Nous allons effectuer une requête POST dans une méthode appelée `storeStory`.

⁵⁰https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array/push

```
1 Vue.component('story', {
2   ...
3   methods: {
4     ...
5     storeStory: function(story){
6       this.$http.post('/api/stories/', story).then(function() {
7         story.editing = false;
8       });
9     },
10    }
11   ...
12 })
```

Nous avons utilisé la méthode abrégée pour exécuter la requête POST. Dans la fonction callback de succès, nous avons mis l'édition à `false` pour afficher de nouveau les boutons *action* et masquer les entrées et boutons d'édition du formulaire. Ci-dessous, nous allons mettre à jour le *groupe* de bouton, conformément à la nouvelle méthode.

```
1 <td>
2   <div class="btn-group" v-if="!story.editing">
3     <button @click="upvoteStory(story)" class="btn btn-primary">
4       Upvote
5     </button>
6     <button @click="editStory(story)" class="btn btn-default">
7       Edit
8     </button>
9     <button @click="deleteStory(story)" class="btn btn-danger">
10      Delete
11    </button>
12  </div>
13  <div class="btn-group" v-else>
14    <button class="btn btn-primary" @click="updateStory(story)">
15      Update Story
16    </button>
17    <button class="btn btn-success" @click="storeStory(story)">
18      Save New Story
19    </button>
20    <button @click="story.editing=false" class="btn btn-default">
21      Cancel
22    </button>
23  </div>
24 </td>
```

Nous observons une petite erreur dans ce bloc de code. Lorsque nous sommes en mode *editing* (bloc `v-else`) Nous voyons que les boutons pour mettre à jour et stocker Sont affichés ensemble, mais nous avons seulement besoin d'un pour chaque histoire, puisque chaque histoire sera `Stored` ou `Updated`. Il ne peut pas faire les deux. Donc, si l'histoire est ancienne et que l'utilisateur est sur le point de l'édition, nous avons besoin du bouton de mise à jour. D'autre part, si l'histoire est nouvelle, nous avons besoin du bouton d'ajout de l'histoire.

	Story Text	Author	Upvotes	Action Buttons
3	Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations.	C-3PO	384	Upvote Edit Delete
4	Help me Obi-Wan Kenobi, you're my only hope.	Princess Leia	896	Upvote Edit Delete
5	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	680	Upvote Edit Delete
6	The Dark Side of the Force is the pathway to many abilities some consider to be... Unnatural.	Senator Palpatine	787	Upvote Edit Delete
7	Aren't you a little short for a storm trooper?	Princess Leia	387	Upvote Edit Delete
9	The force is strong with this one.	Darth Vader	836	Upvote Edit Delete
10	There is good in him. I've felt it.	Luke Skywalker	318	Upvote Edit Delete
11	Good Anakin. Good. Kill him. Kill him now.	Chancellor Palpatine	397	Upvote Edit Delete
14	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	867	Upvote Edit Delete
16	Now, young Skywalker... you will die.	The Emperor	899	Upvote Edit Delete
<input type="text" value="A new story"/> <input type="text" value="Myself"/> 0		Update Story Save New Story Cancel		

Here's a list of all your stories. [Add a new one?](#)

Une petite erreur

Pour contourner ce problème, nous allons restructurer nos boutons. Le bouton `Update` S'affichera seulement lorsque l'histoire est ancienne. En conséquence, le bouton `Enregistrer une nouvelle histoire` sera affiché lorsque l'histoire est nouvelle.

Vous avez peut-être remarqué que toutes les histoires extraites du serveur ont un attribut `id`. Nous allons utiliser cette observation Pour définir si une histoire est nouvelle ou non.

```
1 <div class="btn-group" v-else>
2   <!--Si l'histoire est encienne, nous voulons la mettre à jour
3   ASTUCE: Si l'histoire provient de la base de données, il aura une propriété \
4   id-->
5   <button v-if="story.id" class="btn btn-primary" @click="updateStory(story)">
6     Update Story
7   </button>
8   <!--Si l'histoire est nouvelle, nous voulons l'ajouter à la base de données \
9 -->
10  <button v-else class="btn btn-success" @click="storeStory(story)">
11    Save New Story
12  </button>
13  <!--Toujours afficher le bouton cancel-->
14  <button @click="story.editing=false" class="btn btn-default">
15    Cancel
16  </button>
17 </div>
```



Astuce

Si l'histoire est tirée de la base de données, elle aura un identifiant (id).

#	Plot	Writer	Upvotes	Actions
2	Anakin, you're breaking my heart! And you're going down a path I cannot follow!	Padme	586	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
3	Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations.	C-3PO	384	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
4	Help me Obi-Wan Kenobi, you're my only hope.	Princess Leia	896	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
5	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	680	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
7	Aren't you a little short for a storm trooper?	Princess Leia	387	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
9	The force is strong with this one.	Darth Vader	836	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
10	There is good in him. I've felt it.	Luke Skywalker	318	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
11	Good Anakin. Good. Kill him. Kill him now.	Chancellor Palpatine	397	<button>Upvote</button> <button>Edit</button> <button>Delete</button>

new plot new story by me 0 Save New Story Cancel

Here's a list of all your stories. [Add a new one?](#)

Ajout de nouvelles histoires

Voilà. Ce n'était pas si dur, non ?

Après avoir terminé cette partie, le test de notre application provoque une autre erreur. Après avoir créé, enregistré et essayé de modifier une nouvelle histoire, nous voyons que le bouton indique "Enregistrer une nouvelle histoire" au lieu de "Mettre à jour l'histoire" ! C'est parce que nous ne cherchons pas l'histoire nouvellement créée à partir du serveur, après l'avoir envoyée, elle n'a donc pas encore d'identifiant (id). Pour résoudre ce problème, nous pouvons à nouveau extraire les histoires du serveur, juste après en avoir stocké une nouvelle dans la base de données.

Puisque je n'aime pas répéter mon code, je vais extraire la procédure d'extraction vers une méthode appelée `fetchStories()`. Après cela, je peux utiliser cette méthode pour aller chercher les histoires à tout moment.

The `fetchStories` method

```

1 var vm = new Vue({
2   el: '#v-app',
3   data : {
4     stories: [],
5   },
6   mounted: function(){
7     this.fetchStories()
8   },
9   methods: {

```

```

10     createStory: function(){
11         var newStory={
12             "plot": "",
13             "upvotes": 0,
14             "editing": true
15
16         };
17         this.stories.push(newStory);
18     },
19     fetchStories: function () {
20         this.$http.get('/api/stories')
21             .then(function (response) {
22                 var storiesReady = response.data.map(function(story){
23                     story.editing = false
24                     return story
25                 })
26                 Vue.set(vm, 'stories', storiesReady)
27                 // or: vm.stories = storiesReady
28             });
29     },
30 }
31 });

```

Dans notre situation, nous allons appeler ***`fetchStories ()`*** à l'intérieur du callback de succès de la requête POST.

```

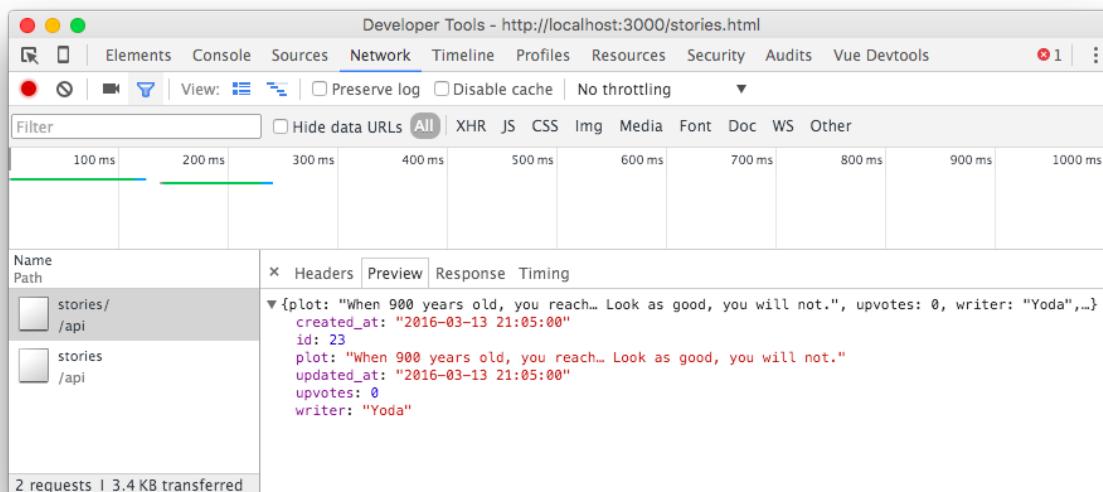
1 Vue.component('story',{
2     ...
3     methods: {
4         ...
5         storeStory: function(story){
6             this.$http.post('/api/stories/', story).then(function() {
7                 story.editing = false;
8                 vm.fetchStories();
9             });
10        },
11    }
12    ...
13 })

```

C'est tout ! Nous pouvons maintenant créer et éditer n'importe quelle histoire que nous voulons.

Ajout et mise à jour par unité

Une meilleure façon de résoudre le problème précédent, est d'aller chercher seulement l'*histoire* ou *story* nouvellement créé de la base de données, au lieu d'aller chercher et d'écraser toutes les *histoires*. Si vous vérifiez la réponse du serveur pour la requête POST, vous verrez qu'elle renvoie la *story* nouvellement créé ainsi que son *id*.



Réponse du serveur après la création d'une histoire

La seule chose que nous devons faire, c'est de mettre à jour notre *histoire* pour correspondre à celle du serveur. Ainsi, nous allons définir *id* des données de réponse, à l'attribut *id* de l'*histoire*. Nous ferons ceci à l'intérieur du callback de succès de la requête POST.

```

1 Vue.component('story', {
2   ...
3   methods: {
4     ...
5     storeStory: function(story){
6       this.$http.post('/api/stories/', story).then(function(response) {
7         Vue.set(story, 'id', response.data.id);
8         story.editing = false
9       });
10      },
11    }
12    ...
13  })

```

J'utilise `Vue.set(story, 'id', response.data.id)` au lieu de `story.id=response.data.id` parce que dans notre table nous affichons le `Id` de chaque histoire. Puisque la nouvelle histoire n'avait pas d'`id`, quand elle est ajoutée dans le tableau `stories`, le DOM ne sera pas mis à jour quand l'`id` change, donc nous ne serons pas en mesure de Voir le nouvel `id`.



Astuce

Lorsque vous ajoutez une **nouvelle propriété qui n'était pas présente lorsque les données ont été observées**, Vue.js ne peut pas détecter l'ajout de la propriété. Si vous devez ajouter ou supprimer des propriétés à l'exécution, utilisez les méthodes globales `Vue.set` ou `Vue.delete`

Fichier Javascript

Comme vous l'avez peut-être remarqué, notre code commence à devenir volumineux. À mesure que notre projet grandit, il devient plus difficile à maintenir. Pour commencer, nous séparerons le code *JavaScript* de l'*HTML*. Je vais créer un fichier appelé `app.js` et je l'enregistrerai sous le répertoire `js`.

Tout le code JavaScript devrait être à l'intérieur de ce fichier à partir de maintenant. Pour inclure le script nouvellement créé sur n'importe quelle page HTML, il vous suffit d'ajouter ce tag

```
<script src='/js/app.js' type="text/javascript"></script>
```

Et vous êtes prêt pour la suite

Code source

Vous trouverez ci-dessous le code source complet de l'exemple précédent *Gestion des histoires*. Si vous avez téléchargé notre dépôt *github*, je vous suggère d'ouvrir vos fichiers locaux avec votre éditeur de texte préféré, car le code est assez grand. Les fichiers se trouvent ici : `~/themajestyofvuejs2/apis/stories/`

Si vous n'avez pas encore téléchargé le code, vous pouvez toujours voir les fichiers `stories.html`⁵¹ et `app.js`⁵² sur *github*.

⁵¹<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/apis/stories/public/stories.html>

⁵²<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/apis/stories/public/js/app.js>

stories.html

```
1 <html lang="en">
2 <head>
3     <title>Stories</title>
4     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2\>
5 /css/bootstrap.min.css">
6 </head>
7
8 <body>
9 <main>
10    <div class="container">
11        <h1>Stories</h1>
12        <div id="v-app">
13            <table class="table table-striped">
14                <tr>
15                    <th>#</th>
16                    <th>Plot</th>
17                    <th>Writer</th>
18                    <th>Upvotes</th>
19                    <th>Actions</th>
20                </tr>
21                <tr v-for="story in stories" is="story" :story="story"></tr>
22            </table>
23            <p class="lead">Here's a list of all your stories.
24                <button @click="createStory()" class="btn btn-primary">
25                    Add a new one?
26                </button>
27            </p>
28            <pre>{{ $data }}</pre>
29        </div>
30    </div>
31 </main>
32 <template id="template-story-raw">
33     <tr>
34         <td>
35             {{story.id}}
36         </td>
37         <td class="col-md-6">
38             <input v-if="story.editing"
39                 v-model="story.plot"
40                 class="form-control">
41         </input>
```

```
42      <!--sinon, afficher l'intrigue de l'histoire -->
43      <span v-else>
44          {{story.plot}}
45      </span>
46  </td>
47  <td>
48      <input v-if="story.editing"
49          v-model="story.writer" class="form-control">
50      </input>
51      <!--sinon, afficher l'auteur de l'histoire-->
52      <span v-else>
53          {{story.writer}}
54      </span>
55  </td>
56  <td>
57      {{story.upvotes}}
58  </td>
59  <td>
60      <div class="btn-group" v-if="!story.editing">
61          <button @click="upvoteStory(story)"
62              class="btn btn-primary">
63              Upvote
64          </button>
65          <button @click="editStory(story)" class="btn btn-default">
66              Edit
67          </button>
68          <button @click="deleteStory(story)"
69              class="btn btn-danger">
70              Delete
71          </button>
72      </div>
73      <div class="btn-group" v-else>
74          <!--si l'histoire provient de la base de données, elle a donc un \
75 id-->
76          <button v-if="story.id"
77              class="btn btn-primary"
78              @click="updateStory(story)">
79              Update Story
80          </button>
81
82          <!--si l'histoire est nouvelle, nous vondrions l'ajouter-->
83          <button v-else class="btn btn-success"
```

```
84          @click="storeStory(story)">
85              Save New Story
86      </button>
87
88      <!--Always show cancel-->
89      <button @click="story.editing=false"
90      class="btn btn-default">
91          Cancel
92      </button>
93  </div>
94 </td>
95 </tr>
96 </template>
97 <script src="http://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
98 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue-resource/0.7.0/vue-resou\
99 rce.js"></script>
100 <script src='/js/app.js' type="text/javascript"></script>
101 </body>
102 </html>
```

```
1 Vue.component('story', {
2     template: '#template-story-raw',
3     props: ['story'],
4     methods: {
5         deleteStory: function (story) {
6             var index = this.$parent.stories.indexOf(story);
7             this.$parent.stories.splice(index, 1)
8             this.$http.delete('/api/stories/' + story.id)
9         },
10        upvoteStory: function (story) {
11            story.upvotes++;
12            this.$http.patch('/api/stories/' + story.id, story)
13        },
14        editStory: function (story) {
15            story.editing = true;
16        },
17        updateStory: function (story) {
18            this.$http.patch('/api/stories/' + story.id, story)
19            //Set editing to false to show actions again and hide the inputs
20            story.editing = false;
21        },
22    }
23 })
```

```
22     storeStory: function (story) {
23         this.$http.post('/api/stories/', story)
24             .then(function (response) {
25                 // Après le stockage de la nouvelle histoire dans la base de don\
26                 nées
27                 // Récupérer toutes les histoires avec
28                 // vm.fetchStories();
29                 // Ou encore mieux, mettre à jour l'id de l'histoire créée
30                 Vue.set(story, 'id', response.data.id);
31
32                 //Définissez 'editing' sur false pour afficher à nouveau les act\
33                 ions et masquer les entrées
34                 story.editing = false;
35             });
36         },
37     }
38 })
39
40 new Vue({
41     el: '#v-app',
42     data: {
43         stories: [],
44         story: {}
45     },
46     mounted: function () {
47         this.fetchStories()
48     },
49     methods: {
50         createStory: function () {
51             var newStory = {
52                 plot: "",
53                 upvotes: 0,
54                 editing: true
55             };
56             this.stories.push(newStory);
57         },
58         fetchStories: function () {
59             console.log('fetsi')
60             var vm = this;
61             this.$http.get('/api/stories')
62                 .then(function (response) {
63                     // set data on vm
```

```
64          var storiesReady = response.data.map(function (story) {
65              story.editing = false;
66              return story
67          })
68          Vue.set(vm, 'stories', storiesReady)
69      });
70  },
71 }
72 );
```



Code exemple

Vous trouverez les exemples de code de ce chapitre sur [GitHub](#)⁵³

Devoirs

Pour vous familiariser avec les requêtes HTTP et la manipulation des réponses, vous devez reproduire ce que nous avons fait dans ce chapitre.

Ce que vous devez faire est de consommer une API afin de :

- Créer un tableau et afficher les films existant
- modifier les films existants
- Ajouter de nouveaux films dans la base de données
- effacer des films de la base de données

J'ai préparé la base de données et l'API pour vous. Vous n'avez qu'à écrire de l'HTML et du JavaScript.

Préface

Si vous avez suivi les instructions du [Chapter 10](#), ouvrez votre terminal et exécutez :

```
>_ cd ~/themajestyofvuejs/apis/movies  
     sh setup.sh
```

Si vous ne l'avez pas fait, vous devrez exécuter ceci :

```
>_ mkdir ~/themajestyofvuejs  
     cd ~/themajestyofvuejs  
     git clone https://github.com/hootlex/the-majesty-of-vuejs.  
     cd ~/themajestyofvuejs/apis/movies  
     sh setup.sh
```

Vous avez maintenant une base de données remplie de grands films avec un serveur entièrement fonctionnel en cours d'exécution sur *http://localhost:3000!*

Pour vous assurer que tout fonctionne bien, allez à *http://localhost:3000/api/movies* et vous devriez voir un tableau de films au format JSON.

⁵³<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter12>

Endpoints de l'API

LES Endpoints d'API que vous allez utiliser sont :

HTTP Method	URI	Action
GET/HEAD	api/movies	Fetches all movies
GET/HEAD	api/movies/{id}	Fetches specified movie
POST	api/movies	Creates a new movie
PUT/PATCH	api/movies/{id}	Updates an existing movie
DELETE	api/movies/{id}	Deletes specified movie

Votre code

Mettez votre code HTML dans le fichier `~/themajestyofvuejs2/apis/movies/public/movies.html` que nous avons créer. Vous pouvez aussi y mettre votre code Javascript, ou dans `js/app.js`.

Pour voir votre travail visitez l'adresse `http://localhost:3000/movies.html` dans votre navigateur web.

Bonne chance !

#	Title	Director	Actions
2	Snatch.	Guy Ritchie	Edit Delete
3	The Lord of the Rings: The Fellowship of the Ring	Peter Jackson	Edit Delete
4	The Grand Budapest Hotel	Wes Anderson	Edit Delete
5	Fight Club	David Fincher	Edit Delete
6	Million Dollar Baby	Clint Eastwood	Edit Delete
7	Fight Club	David Fincher	Edit Delete
8	Aladdin	John Musker	Edit Delete
9	The Wolf of Wall Street	Martin Scorsese	Edit Delete
21	batman	ego	Edit Delete

Here's a list of all your movies. [Add a new one?](#)

Exemple de rendu



Solution potentielle

Vous pouvez trouver une potentielle solution à cet exercice [ici](#)⁵⁴.

⁵⁴<https://github.com/hoottlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter12>

Pagination

Dans le chapitre précédent, nous avons réussi à extraire tous les enregistrements de la base de données et à les afficher dans un tableau. Cette mise en œuvre est très bien pour un tas d'enregistrements, mais dans le monde réel, quand vous devez travailler avec des milliers ou Millions d'enregistrements, vous ne pouvez pas simplement les extraire et les placer dans un tableau. Si vous le faites, votre navigateur ne sera pas heureux de charger une telle quantité de données, mais même si elle réussit à le faire, je vous assure qu'aucun utilisateur n'aime traiter avec une table contenant 100 000 lignes.

Information

La pagination est utilisée sous une forme ou une autre dans presque toutes les applications Web pour diviser les données renvoyées et les afficher sur plusieurs pages. La Pagination comprend également la logique de préparation et l'affichage des liens vers les différentes pages, et cela peut être manipulé côté client ou côté serveur. La pagination côté serveur est plus fréquente.

Dans des situations comme celle-ci, les développeurs qui ont conçu l'API diviseront (espérons-le) les données retournées en plusieurs pages.

La réponse HTTP contiendra quelques **méta-donnée** en plus des données principales, pour vous informer sur le *total* des éléments, des éléments *Par page*, etc. Pour vous aider à parcourir les pages, il fournit aussi des informations telles que **la page courante**, **la page suivante** et **la page précédente**.

Example Response with Paginated data

```
{  
    "total": 10000,  
    "per_page": 50,  
    "current_page": 15,  
    "last_page": 200,  
    "next_page_url": "/api/stories?page=16",  
    "prev_page_url": "/api/stories?page=14",  
    "from": 751,  
    "to": 800,  
    "data": [...]  
}
```

La « méta-donnée » de la pagination pourrait également se trouver à l'intérieur d'un objet à côté des **données**, ou n'importe où selon les décisions prises par les développeurs de l'API.

Example Response with Paginated data

```
{  
    "data": [...],  
    "pagination": {  
        "total": 10000,  
        "per_page": 50,  
        "current_page": 15,  
        "last_page": 200,  
        "next_page_url": "/api/stories?page=16",  
        "prev_page_url": "/api/stories?page=14",  
        "from": 751,  
        "to": 800,  
    }  
}
```

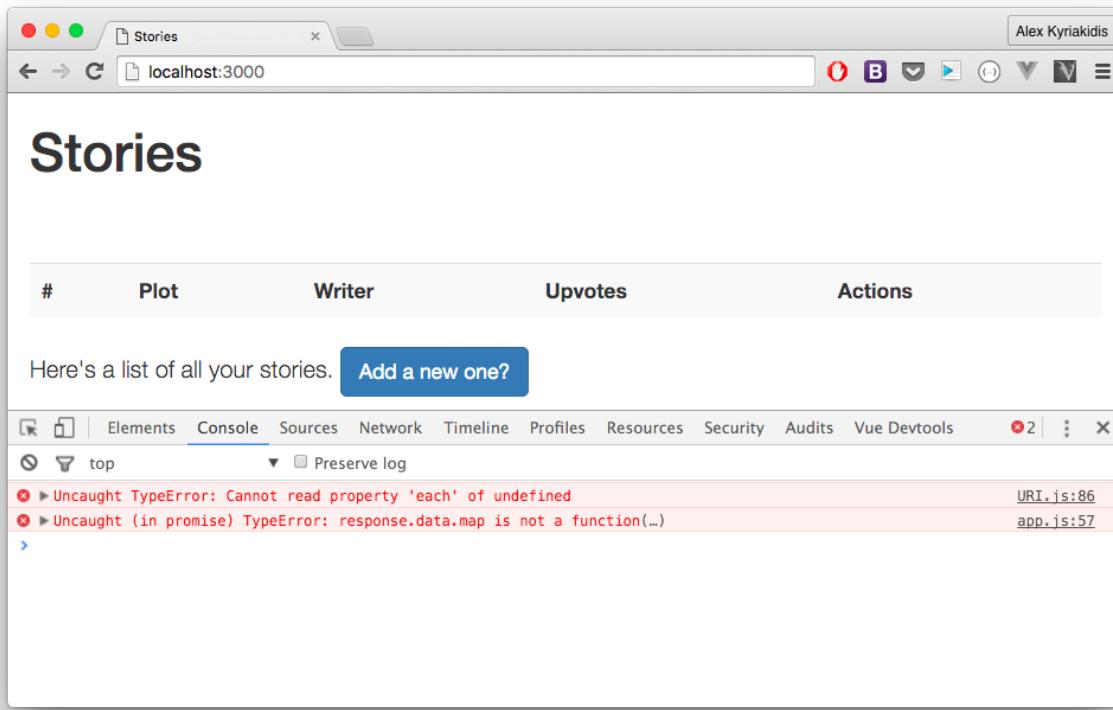
Implementation

Nous allons continuer à travailler avec les exemples *story* du chapitre précédent, en utilisant l'API paginée légèrement améliorée. Ainsi, nous allons modifier le code, pour pouvoir accéder et utiliser nos données.

Si vous jetez un oeil au code d [l'exemple précédent](#), Vous verrez que notre méthode `fetchStories` est similaire à ceci :

```
new Vue({  
    ...  
    methods: {  
        ...  
        fetchStories: function () {  
            var vm = this;  
            this.$http.get('/api/stories')  
                .then(function (response) {  
                    var storiesReady = response.data.map(function (story) {  
                        story.editing = false;  
                        return story  
                    })  
                    Vue.set(vm, 'stories', storiesReady)  
                });  
        },  
        ...  
    }  
});
```

Si nous ouvrons notre fichier HTML dans le navigateur, comme vous l'avez peut-être déjà deviné, notre tableau ne s'affiche pas correctement.



Les histoires de s'affichent pas

Cela se produit parce que les **stories** sont maintenant retournés dans un tableau nommé **data**. Pour corriger cela, nous devons changer **response.data** en **response.data.data** (Je sais que c'est un peu bizarre, mais bon...).

Exépter pour le tableau *stories*, nous voudrions également enregistrer les données de la pagination dans un objet afin d'implémenter facilement la fonctionnalité de pagination.

Pour savoir comment nous pouvons accéder à ces données, nous allons voir de plus près la réponse du serveur.

```

{
  "total": 188,
  "per_page": 15,
  "current_page": 1,
  "last_page": 13,
  "next_page_url": "http://localhost:3000/api/stories?page=2",
  "prev_page_url": null,
  "from": 1,
  "to": 15,
  "data": [
    {
      "id": 1,
      "plot": "Mmm. Lost a planet, Master Obi-Wan has. How embarrassing.",
      "upvotes": "883",
      "writer": "Yoda",
      "created_at": "2016-04-10 17:47:03",
      "updated_at": "2016-04-10 17:47:03"
    },
    {
      "id": 2,
      "plot": "Don't call me a mindless philosopher, you overweight glob of grease.",
      "upvotes": "475",
      "writer": "C-3PO",
      "created_at": "2016-04-10 17:47:03",
      "updated_at": "2016-04-10 17:47:03"
    },
    {
      "id": 3,
      "plot": "Master Kenobi, you disappoint me. Yoda holds you in such high esteem. Surely you can do better!",
      "upvotes": "427",
      "writer": "Count Dooku",
      "created_at": "2016-04-10 17:47:03",
      "updated_at": "2016-04-10 17:47:03"
    },
    ▶ { ... }, // 6 items
    ▶ { ... }, // 6 items
    ▶ { ... }, // 6 items
    ▶ { ... }, // 6 items
  ]
}

```

Reponse serveur

Pour commencer, nous n'avons pas besoin de toutes ces données. Donc, nous nous en tiendrons à `current_page`, `last_page`, `next_page_url` et `prev_page_url`.

Notre objet de pagination ressemblera à ceci :

```

pagination: {
  "current_page": 15,
  "last_page": 200,
  "next_page_url": "/api/stories?page=16",
  "prev_page_url": "/api/stories?page=14"
}

```

Modifions notre méthode `fetchStories` pour mettre à jour l'objet `pagination` chaque fois que des histoires sont récupérées de la base de données.

```
new Vue({  
  ...  
  methods: {  
    ...  
    fetchStories: function () {  
      var vm = this;  
      this.$http.get('/api/stories')  
        .then(function (response) {  
          var storiesReady = response.data.data.map(function (story) {  
            story.editing = false;  
            return story  
          })  
          //ici nous utilisons response.data  
          var pagination = {  
            current_page: response.data.current_page,  
            last_page: response.data.last_page,  
            next_page_url: response.data.next_page_url,  
            prev_page_url: response.data.prev_page_url  
          }  
          Vue.set(vm, 'stories', storiesReady)  
          Vue.set(vm, 'pagination', pagination)  
        });  
    },  
    ...  
  }  
});
```

Liens de pagination

Maintenant nous avons notre objet `pagination`, mais nous récupérons toujours la première page des stories vue que nous faisons une requête HTTP GET vers `api/stories`. Nous devons modifier la page demandée, en fonction de l'interaction de l'utilisateur (page suivante, page précédente).

Nous allons d'abord mettre à jour la méthode `fetchStories` pour accepter en argument la page désirée. Si aucun argument n'est transmis, la première page sera récupérée. Je vais également créer une nouvelle méthode, `makePagination`, pour rendre le code plus propre.

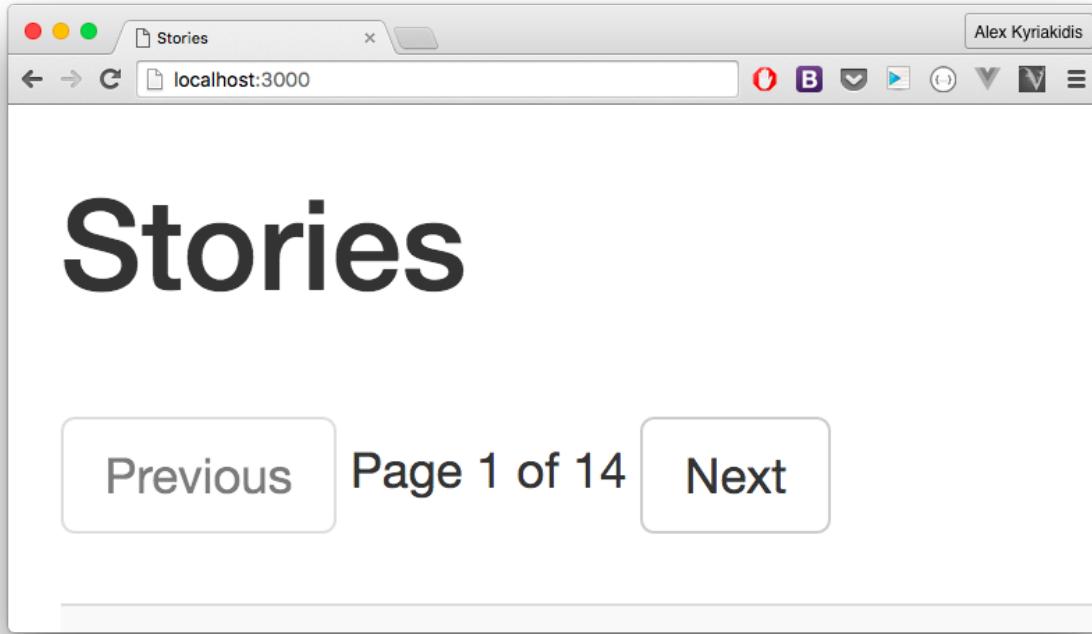
```
new Vue({  
  ...  
  methods: {  
    ...  
    fetchStories: function (page_url) {  
      var vm = this;  
      page_url = page_url || '/api/stories'  
      this.$http.get(page_url)  
        .then(function (response) {  
          var storiesReady = response.data.data.map(function (story) {  
            story.editing = false;  
            return story  
          })  
          vm.makePagination(response.data)  
          Vue.set(vm, 'stories', storiesReady)  
        });  
    },  
    makePagination: function (data){  
      //here we use response.data  
      var pagination = {  
        current_page: data.current_page,  
        last_page: data.last_page,  
        next_page_url: data.next_page_url,  
        prev_page_url: data.prev_page_url  
      }  
      Vue.set(vm, 'pagination', pagination)  
    }  
    ...  
  }  
}
```

Maintenant que notre méthode est prête, nous avons besoin d'un moyen de l'appeler correctement. Nous ajouterons 2 boutons, un pour la prochaine page et un autre pour la page précédente dans notre `* app div`. Chaque bouton appellera la méthode `fetchStories` lorsque qu'il sera cliqué, en passant l'url de la page correspondante.

```
1 <div class="pagination">
2   <button @click="fetchStories(pagination.prev_page_url)">
3     Previous
4   </button>
5   <button @click="fetchStories(pagination.next_page_url)">
6     Next
7   </button>
8 </div>
```

Si vous essayez de cliquer sur les boutons, vous verrez qu'ils fonctionnent comme prévu. Nous avons créé notre pagination en un clin d'œil. Il sera cependant utile d'informer l'utilisateur à propos de la page qu'il regarde actuellement et du nombre total de pages. De plus, nous pouvons désactiver le bouton précédent lorsque l'utilisateur est sur la première page, et le suivant lorsqu'il est sur la dernière.

```
1 <div class="pagination">
2   <button @click="fetchStories(pagination.prev_page_url)"
3     :disabled="!pagination.prev_page_url"
4   >
5     Previous
6   </button>
7   <span>Page {{pagination.current_page}} of {{pagination.last_page}}</span>
8   <button @click="fetchStories(pagination.next_page_url)"
9     :disabled="!pagination.next_page_url"
10  >
11    Next
12  </button>
13 </div>
```



Bouton précédent désactivé



Exemple de code

Vous trouverez les exemples de code de ce chapitre sur [GitHub⁵⁵](#).

Devoirs

Il n'y a rien de particulier à faire comme devoir dans ce chapitre. Si vous voulez réellement travailler sur cet exemple, je vous fournirai l'API paginée.

Si vous avez résolu les devoirs du chapitre précédent (télécharger le code et démarrer un serveur), vous êtes à quelques clics. Si vous ne l'avez pas fait, suivez simplement [ces instructions](#).

L'API paginée est dans le répertoire '`~/themajestyofvuejs2/apis/pagination/stories`'.

Le fichier HTML se trouve dans le répertoire '`~/themajestyofvuejs2/apis/pagination/stories/public`'.

Si vous voulez juste afficher le code final, vous pouvez jeter un œil aux fichiers sur [GitHub⁵⁶](#).

⁵⁵<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter13>

⁵⁶<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/apis/pagination/stories/public>

Construire des applications à grande échelle

ECMAScript 6

Avant d'aller un peu plus loin et de voir comment nous pouvons construire des applications à grande échelle, je voudrai vous familiariser avec ECMAScript 6.



Information

ECMAScript est une spécification du langage de scripting côté client, qui est à l'origine de plusieurs langages de programmation, y compris JavaScript, ActionScript et JScript.

ECMAScript 6 (ES6), également connu sous le nom ES2015, est la dernière version de la norme ECMAScript. La spécification ES6 a été finalisée en juin 2015. C'est une mise à jour significative du langage, et la première mise à jour majeure depuis que l'ES5 a été normalisé en 2009. La mise en œuvre des fonctionnalités ES6 dans les principaux moteurs JavaScript est en cours (<http://kangax.github.io/compat-table/es6/>).

Introduction

ES6 a beaucoup de nouvelles fonctionnalités. Nous allons passer en revue ceux que nous allons utiliser dans les prochains chapitres. Si vous voulez en savoir plus sur ce qui est nouveau dans ES6, je vous recommande vivement le livre “Understanding ECMAScript 6” de Nicholas C.Zakas disponible sur [leanpub⁵⁷](#). Il existe également une [Version en ligne⁵⁸](#) du livre, que vous pouvez lire gratuitement.

Il existe aussi d'autres ressources utiles et des tutoriel comme celui sur [Babel⁵⁹](#), un article sur [tutsplus⁶⁰](#), un [Article de blog⁶¹](#) de Nicholas C. Zakas, et une tonne de choses autour du web !

⁵⁷<https://leanpub.com/understandinges6/>

⁵⁸<https://leanpub.com/understandinges6/read>

⁵⁹<https://babeljs.io/docs/learn-es2015/>

⁶⁰<http://code.tutsplus.com/articles/use-ecmascript-6-today--net-31582>

⁶¹<https://www.nczonline.net/blog/2013/09/10/understanding-ecmascript-6-arrow-functions/>

Compatibilité

Sans surprise, le support varie énormément entre les moteurs Javascript, avec Mozilla tendant à montrer la voie. [Le tableau de compatibilité ES6⁶²](#) est un bon point de départ pour savoir quelles fonctionnalités ECMAScript 6 votre navigateur ne prend pas en charge.



Remarque

Si vous utilisez Chrome, la plupart des fonctionnalités ES6 sont cachées derrière une bascule de fonction. Accédez à `chrome://flags`, trouvez la section intitulée “Fonctionnalité expérimentale JavaScript” et activez le support.

Désormais, nous développerons nos exemples à l'aide des fonctionnalités ES6.

Déclaration de variables

La déclaration de type Let

`let` est le nouveau `var`. Vous pouvez essentiellement remplacer `var` par `let` pour déclarer une variable, mais limiter la portée de la variable uniquement au bloc de code actuel. Puisque les déclarations ne sont pas hissées au sommet du bloc inclus, mieux vaut toujours placer les déclarations en premier dans le bloc, de sorte qu'elles soient disponibles pour le bloc entier. Par exemple :

Let inside if

```
1 let age = 22
2 if (age >= 18) {
3     let adult = true;
4     console.log(adult); //outputs true
5 }
6 //adult n'est pas accessible ici
7 console.log(adult);
8 //ERROR: Uncaught ReferenceError: adult is not defined
```

⁶²<https://kangax.github.io/compat-table/es6/>

Let on top

```
1 let age = 22
2 let adult
3 if (age >= 18) {
4     adult = true;
5     console.log(adult); //outputs true
6 }
7 //maintenant adult est accessible
8 console.log(adult); //outputs true
```

Déclaration des constantes

Les constantes, comme les déclarations `let`, sont des déclarations au niveau des blocs. Il y a une grande différence entre `let` et `const`. Une fois que vous déclarez une variable en utilisant `const`, elle est définie comme une constante, ce qui signifie que **vous ne pouvez pas changer sa valeur**.

```
1 const name = "Alex"
2
3 name = "Kostas" //throws error
```



Information

Tout comme les constantes dans d'autres langues, leur valeur ne peut pas changer plus tard. Cependant, contrairement aux constantes dans d'autres langages, la valeur d'une constante peut être modifiée si elle est un objet.

Les Fonctions fléchées (Arrow Functions)

L'une des parties les plus intéressantes de l'ECMAScript 6 est les Fonctions fléchées. Les fonctions fléchées sont des fonctions définies avec une nouvelle syntaxe qui utilise une “flèche” (\Rightarrow). Elles peuvent prendre la forme soit d'une expression ou d'une déclaration. Contrairement aux fonctions, les flèches partagent le même lexical `this` que leur code environnant.

Par exemple, la fonction flèche suivante prend un seul argument et renvoie sa valeur augmentée de 1 :

```
var increment = value => value + 1;
increment(5) //returns 6

// equivalent to :

var increment = function(value) {
    return value + 1;
};
```

Un autre exemple avec une fonction de flèche qui prend 2 arguments et retourne leur somme :

```
var sum = (a, b) => a + b;
sum(5, 10) //returns 15

// equivalent to:

var sum = function(a, b) {
    return a + b;
};
```

Un exemple de fonction fléchée qui ne prend aucun argument et utilise une instruction.

```
var sayHiAndBye = () => {
    console.log('Hi!');
    console.log('Bye!');
};

sayHiAndBye()

// equivalent to:

var sayHiAndBye = function() {
    console.log('Hi!');
    console.log('Bye!');
};
```

Modules

C'est pour moi la plus grande amélioration de la langue. ES6 prend désormais en charge l'exportation et l'importation de modules sur différents fichiers.

L'exemple le plus simple est de créer un fichier `.js` avec une variable, et de l'utiliser dans un autre fichier comme celui-ci :

module.js

```
1 export name = 'Alex'
```

main.js

```
1 import {name} from './module'
2 console.log('Hello', name)
3 //affiche "Hello Alex"
```

Vous pouvez également exporter des variables ou les fonctions **une par une**

module.js

```
1 export var name = 'Alex'
2 export function getAge(){
3   return 22;
4 }
```

main.js

```
1 import {name, getAge} from './module'
2 console.log(name, 'is', getAge())
```

Or **inside an object** :

module.js

```
1 var name = 'Alex'
2 function getAge(){
3   return 22;
4 }
5 export default {name, age}
```

main.js

```
1 import person from './module'
2 console.log( person.name, 'is', person.getAge() )
3 //affiche "Alex is 22"
```

Classes

Les classes JavaScript sont introduites dans ECMAScript 6 et sont raccourcies de l'héritage prototypal de Javascript. La syntaxe de classe **n'introduit pas** un nouveau modèle d'héritage orienté objet. Les classes JavaScript fournissent une syntaxe beaucoup plus simple et plus claire pour créer des objets et gérer l'héritage.

Class Example

```
//parent class
class Rectangle {
    constructor(height, width) {
        this.height = height;
        this.width = width;
    }

    calcArea() {
        return this.height * this.width;
    }

    //Pour créer un getter, utilisez le mot clé get.
    get area() {
        return this.calcArea();
    }
    //Pour créer un setter, utilisez le mot clé set.
}

//class enfant
class Square extends Rectangle{
    constructor(side) {
        //appel au constructeur du parent
        super(side, side)
    }
}

var square = new Square(5);

console.log(square.area); //affiche 25
```

Valeurs par défaut des arguments

Avec ES6, vous pouvez définir des valeurs de paramètres par défaut.

```
function divide(x, y = 2){
    return x/y;
}

// equivalent to:

function divide(x, y){
    y = y == undefined ? 2 : y;
    return x/y;
}
```

Littéraux de gabarits (Template literals)

Les Template literals sont des littéraux de chaîne permettant des expressions intégrées. Vous pouvez utiliser des chaînes multi-lignes et des fonctions d'interpolation de chaîne avec eux. Elles ont été appelées “template strings” dans les éditions antérieures de la spécification ES2015/ES6.

Les Template literals sont entourées par des back-tick (`) au lieu des guillemets simples. Dans les back-ticks (CTRL + ALT + 7), vous pouvez utiliser \${expression}, où expression peut être une fonction, une variable ou une expression réelle.

Template literals - Using Variables

```
let name = 'Alex'
console.log(`Hello ${name}`)

// equivalent to:
console.log('Hello ' + name )
```

Template literals - Using Functions

```
add = (a, b) => a + b

let [a, b] = [10, 2]

console.log(`If you have ${a} eggs and you buy ${b}
more you'll have ${add(a,b)} eggs!`)

// équivalent à:
console.log('If you have ' + a + ' eggs and you buy ' + b +
'\nmore you\'ll have ' + add(a,b) + ' eggs!')
```

Template literals - Using Expressions

```
let [a, b] = [10, 2]

console.log(`If you have ${a} eggs and you buy ${b}
more you'll have ${a + b} eggs!`)

// équivalent à:
console.log('If you have ' + a + ' eggs and you buy ' + b +
'\nmore you\'ll have ' + (a + b)*1 + ' eggs!')
```

Il est également possible de diviser le message en plusieurs lignes en utilisant '\n' ;

Workflow avancé

Toutes ces fonctionnalités ES6 (et beaucoup d'autres) peuvent vous exciter, mais il ya un piège ici. Comme nous l'avons mentionné précédemment, **tous les navigateurs** ne prennent pas pleinement en charge les fonctionnalités ES6/ES2015.

Pour pouvoir écrire cette nouvelle syntaxe JavaScript aujourd'hui, nous avons besoin d'un intermédiaire qui prendra notre code et le traduira en [Vanilla JS] (<http://vanilla-js.com/>), que *chaque navigateur comprend*. Cette procédure est vraiment **importante** en production. Laissez-moi vous raconter une histoire. Il ya quelques années, un de mes collègues a commencé à utiliser certaines fonctionnalités JS cool qui n'étaient pas entièrement prises en charge par tous les navigateurs. Quelques jours plus tard, nos utilisateurs ont commencé à se plaindre de certaines pages de notre site Web qui ne s'affichent pas correctement, mais nous n'avons pas pu comprendre pourquoi. Nous l'avons testé sur différents PC, téléphones Android, iPhones, etc, et il était 100% fonctionnel dans tous nos navigateurs. Plus tard, il a découvert que les anciennes versions de Safari mobile ne supportaient pas son code. *Ne soyez pas ce gars !*

Parfois, il est **très difficile** de savoir si le code que vous écrivez va bien fonctionner sur **tous les navigateurs**, y compris le navigateur mobile de Facebook, qui est ma pire crainte.

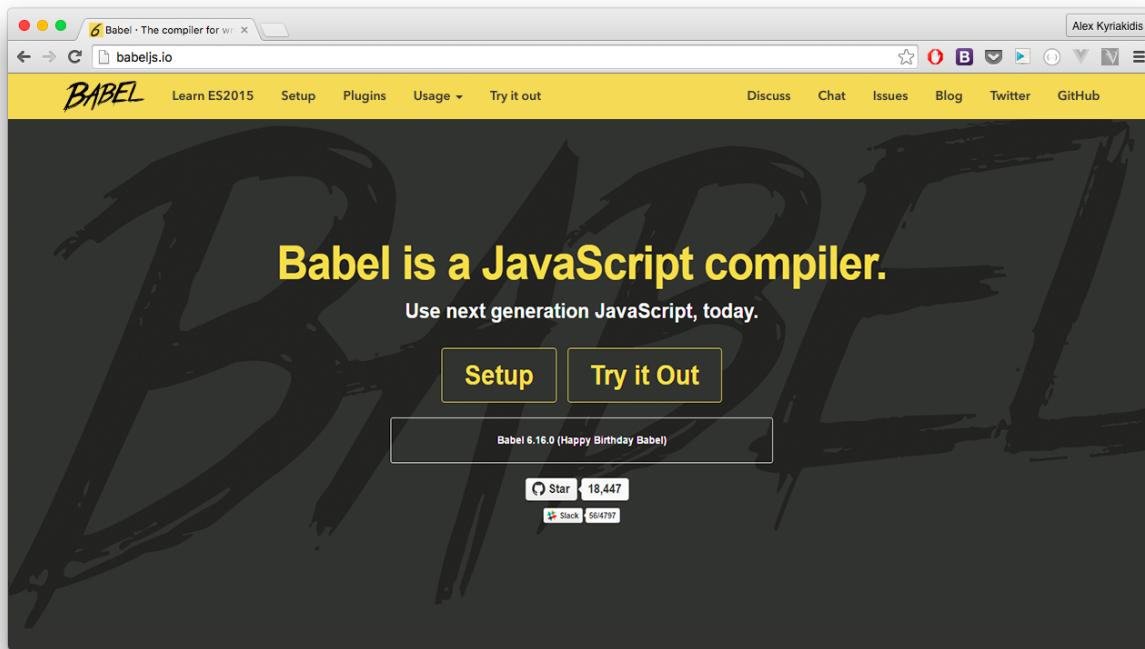
Compilation ES6 avec Babel

Babel sera notre intermédiaire. Babel est un compilateur JavaScript de source à source, qui nous permet d'utiliser la prochaine génération de JavaScript aujourd'hui.



Information

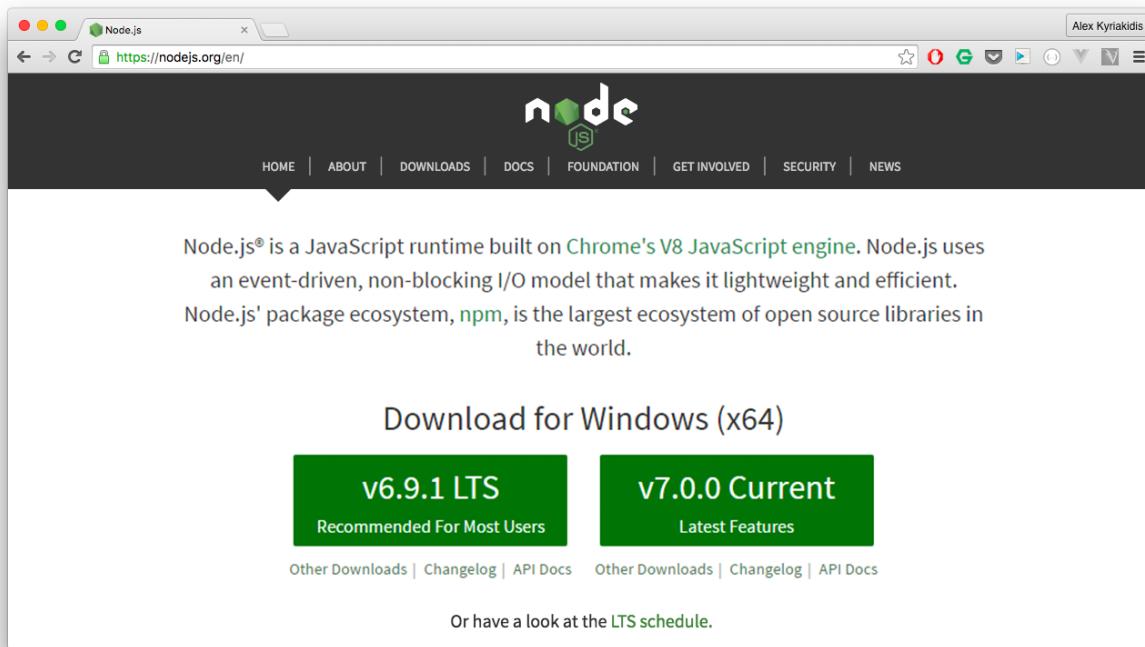
Un compilateur, transcompiler ou transpiler source-à-source est un type de compilateur qui prend le code source d'un programme écrit dans un langage de programmation, comme son entrée, et produit le code source équivalent dans un autre langage de programmation.



Babe <https://shop.nets.eu/nb/web/partners/test-cards1>

Avant d'installer Babel, vous devez installer **Node.js**. Pour ce faire, rendez-vous sur le site Web de [Node⁶³](#) et cliquez sur le bouton de téléchargement de la dernière version stable. Il va vous donner un fichier *.pkg* (ou *.msi* si vous êtes sur Windows). Une fois le téléchargement terminé, ouvrez le fichier et suivez les instructions. Ensuite, faites le redémarrage nécessaire, et vous avez terminé !

⁶³<https://nodejs.org/en/>



Node.js

Installation

Créez un nouveau répertoire et créer un fichier nommé **package.json** contenant un objet JSON vide ({}). Vous pouvez le faire manuellement, ou en exécutant les commandes suivantes dans votre terminal.

```
>_ mkdir babel-example  
echo {} > package.json
```

Ensuite, exécutez ceci pour installer Babel :

```
>_ npm install babel-cli --save-dev
```



```
alex@192: ~
└── path-is-absolute@1.0.0
  ├── convert-source-map@1.2.0
  ├── commander@2.9.0 (graceful-readlink@1.0.1)
  ├── v8flags@2.0.11 (user-home@1.1.1)
  ├── source-map@0.5.6
  ├── chalk@1.1.1 (escape-string-regexp@1.0.5, supports-color@2.0.0, ansi-styles@2.2.1, strip-ansi@3.0.1, has-ansi@2.0.0)
  ├── glob@5.0.15 (inherits@2.0.1, once@1.3.3, inflight@1.0.4, minimatch@3.0.0)
  ├── output-file-sync@1.1.1 (xtend@4.0.1, mkdirp@0.5.1)
  ├── request@2.72.0 (tunnel-agent@0.4.3, aws-sign2@0.6.0, oauth-sign@0.8.2, forever-agent@0.6.1, is-typedarray@1.0.0, caseless@0.11.0, stringstream@0.0.5, aws4@1.4.1, isstream@0.1.2, json-stringify-safe@5.0.1, extend@3.0.0, tough-cookie@2.2.2, node-uuid@1.4.7, qs@6.1.0, combined-stream@1.0.5, mime-types@2.1.11, form-data@1.0.0-rc4, bl@1.1.2, hawk@3.1.3, http-signature@1.1.1, har-validator@2.0.6)
  ├── babel-core@6.8.0 (babel-messages@6.8.0, shebang-regex@1.0.0, babel-template@6.8.0, babel-helpers@6.8.0, private@0.1.6, babel-code-frame@6.8.0, debug@2.2.0, babylon@6.8.0, minimatch@2.0.10, babel-types@6.8.1, babel-generator@6.8.0, babel-traverse@6.8.0, json5@0.4.0)
  ├── bin-version-check@2.1.0 (minimist@1.2.0, semver@4.3.6, semver-truncate@1.1.0, bin-version@1.0.4)
  ├── lodash@3.10.1
  ├── babel-register@6.8.0 (home-or-tmp@1.0.0, mkdirp@0.5.1, source-map-support@0.2.10, core-js@2.4.0)
  ├── babel-polyfill@6.8.0 (babel-regenerator-runtime@6.5.0, core-js@2.4.0)
  ├── babel-runtime@6.6.1 (core-js@2.4.0)
  └── chokidar@1.5.0 (inherits@2.0.1, glob-parent@2.0.0, async-each@1.0.0, is-glob@2.0.1, is-binary-path@1.0.1, readdirp@2.0.0, anymatch@1.3.0, fsevents@1.0.12)

~ » | alex@192
```

Sortie du terminal

Une fois la commande exécutée, votre fichier package.json devrait ressembler à ça :

package.json

```
{  
  "devDependencies": {  
    "babel-cli": "^6.18.0"  
  }  
}
```



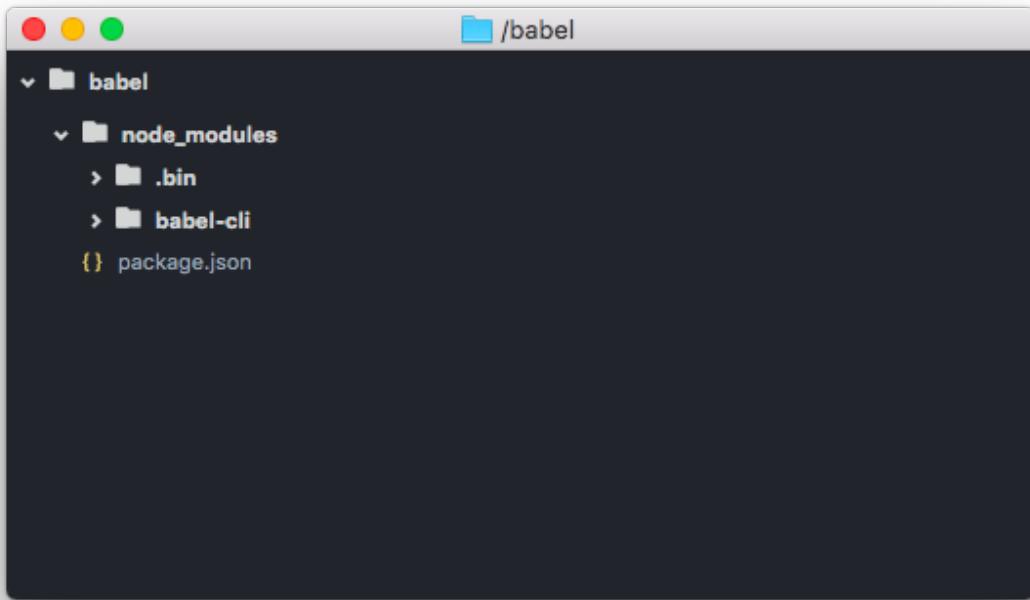
Qu'est-ce que package.json ?

Un fichier package.json contient des métadonnées concernant votre application ou votre module. Plus important encore, il inclut la liste des dépendances à installer à partir d'un nom lors de l'exécution de `npm install`. Si Composer vous est familier, il est similaire au fichier composer.json.

Pour en savoir plus sur package.json, consultez la documentation npm⁶⁴.

Le répertoire de votre projet devrait ressembler à ceci :

⁶⁴<https://docs.npmjs.com/files/package.json>



Repertoire du projet

Configuration

Maintenant que nous avons installé babel, nous devons explicitement lui dire quelles transformations exécuter lors du build. Puisque nous voulons transformer le code ES2015, nous installerons le plugin babel [ES2015-Preset⁶⁵](#).

Nous allons également créer un fichier de configuration (`.babelrc`) pour activer notre presret.

```
>_ npm install babel-preset-es2015 --save-dev
      echo { "presets": [ ["es2015"] ] } > .babelrc
```



Astuce

Si la deuxième commande échoue, incluez le contenu du fichier à l'intérieur des guillemets comme ceci :

```
echo '{"presets": [ ["es2015"] ]}' > .babelrc
```

⁶⁵<https://babeljs.io/docs/plugins/preset-es2015/>

Créer des alias

Au lieu de d'executer Babel directement à partir de la ligne de commande, nous allons mettre nos commandes dans **npm scripts**.

Nous allons ajouter un champ **scripts** dans notre fichier **package.json**, et enregistrer la commande **babel** ici, en tant que **build**.

Notre **package.json** ressemblera à ceci :

package.json

```
{  
  "scripts": {  
    "build": "babel src -d assets/js"  
  },  
  "devDependencies": {  
    "babel-cli": "^6.8.0",  
    "babel-preset-es2015": "^6.18.0"  
  }  
}
```

Cela fonctionne comme un alias. Cela signifie que lorsque nous exécutons **npm run build**, nous exécutons en fait **babel src -d assets/js**. Cette commande demande à Babel de déplacer le code du répertoire **src** au répertoire **assets/js**.

Avant de lancer la commande **build**, nous devons faire encore plus. Pour commencer, créez les répertoires mentionnés ci-dessus (*src* et *assets/js*).

Utilisation

Allons de l'avant, et mettons quelques fichiers à l'intérieur du dossier *src*. Je vais créer un fichier avec une simple fonction **sum** et l'appeler **sum.js**.

src/sum.js

```
const sum = (a, b) => a + b;  
console.log(sum(5,3));
```

Voilà. Nous pouvons maintenant exécuter :

```
>_ npm run build
```

Lorsque vous l'exécutez, vous pouvez voir dans votre terminal que le fichier **src\sum.js** a été compilé dans **assets\js\sum.js** et ressemble à ceci :

assets/js/sum.js

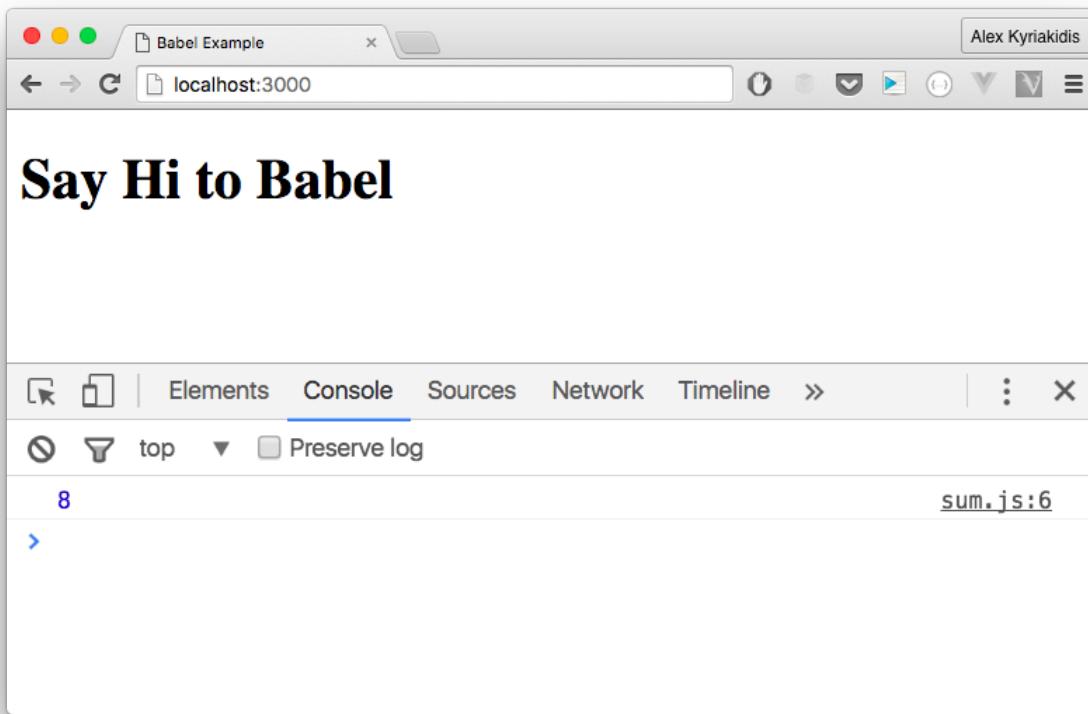
```
"use strict";  
  
var sum = function sum(a, b) {  
    return a + b;  
};  
console.log(sum(5, 3));
```

Désormais, à chaque fois que vous souhaitez compiler votre code ES6, vous pouvez le faire en exécutant la commande **build**. Jolie, n'est ce pas ?

Il est temps de voir le résultat de **sum.js** dans le navigateur. Je vais créer **sum.html** et y inclure notre **js**.

sum.html

```
<!DOCTYPE html>  
<html>  
<head>  
    <title>Babel Example</title>  
</head>  
<body>  
    <h1>Babel Example</h1>  
  
    <script src="assets/js/sum.js"></script>  
</body>  
</html>
```



Sortie du navigateur

Comme vous pouvez le voir, le résultat de la fonction `sum` est affiché avec succès dans la console.



Information

Lorsque vous voulez tester un fichier `.js`, mais que vous ne voulez pas le mettre en service dans le navigateur, vous pouvez l'exécuter avec Node.js.

Dans l'exemple `sum.js` il y a une ligne `console.log(sum(5,3))`, donc si vous tapez dans votre terminal `node sum.js` vous verrez le résultat (8) s'afficher !

Devoirs

Cet exercice vise à vous aider à vous rappeler ce que vous avez appris en reproduisant l'exemple que nous avons construit. Au lieu de `sum.js`, utilisez les *Classes ES6* pour créer un fichier `Ninja.js` contenant la classe `Ninja`.

Un `Ninja` devrait avoir une propriété `name` et une méthode `announce`, qui alertera de la présence d'un Ninja.

For example

```
new Ninja('Leonardo').announce()  
//alerts "Ninja Leonardo is here!"
```

N'oubliez pas de compiler votre *js* en utilisant Babel avant de l'inclure dans votre *HTML*.

**Conseil**

Vous pouvez trouver un exemple de création de classes dans le chapitre précédent.

**Conseil 2**

N'oubliez pas d'exécuter `npm run build` chaque fois que vous modifiez votre fichier *js*, sinon il ne sera pas mis à jour !

**Solution potentielle**

Vous pouvez trouver une solution à cet exercice [here⁶⁶](#).

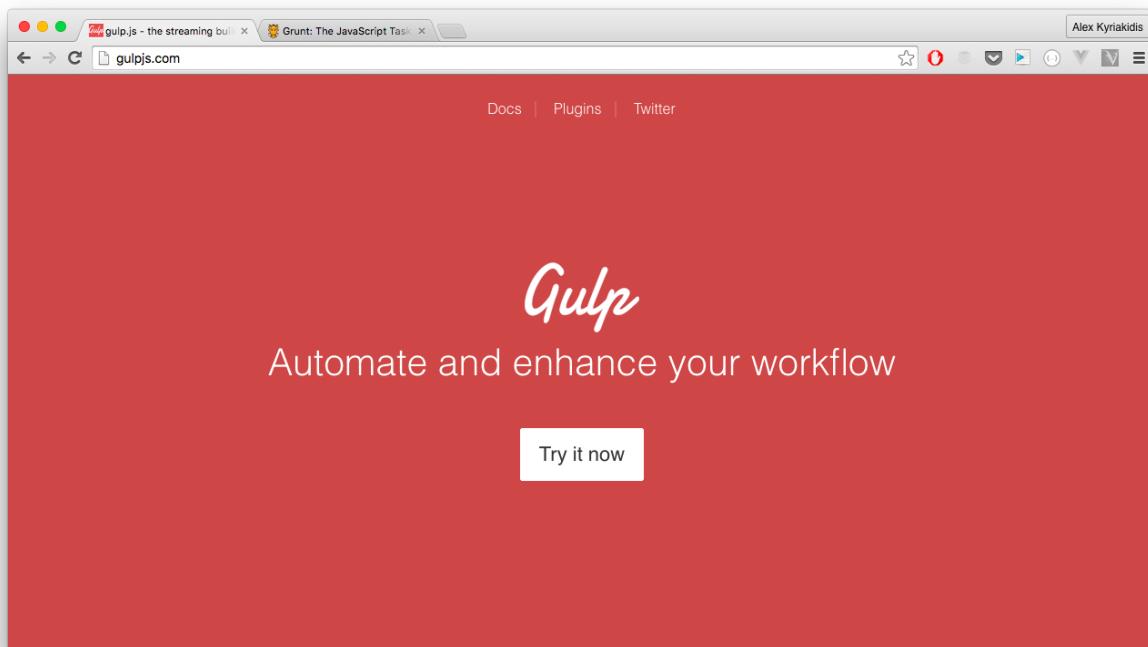
⁶⁶<https://github.com/hoottlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter15/chapter15.1>

Automatisation de workflow avec Gulp

Task Runners (gestionnaires de tâches)

Si vous avez consacré un peu de temps à développer l'application à partir de la section précédente, vous avez sans doute découvert qu'il est gênant d'exécuter `npm run build` chaque fois que vous modifiez votre code.

C'est là que les Task Runners comme [Gulp⁶⁷](#) ou [Grunt⁶⁸](#) sont très utiles. Les gestionnaires de tâches vous permettent d'automatiser et d'améliorer votre workflow.



Gulp



Pourquoi utiliser un gestionnaire de tâches ?

En un mot : L'automatisation. Moins vous avez à effectué des tâches répétitives comme la minification, la compilation, les testes unitaires, linting, etc, plus votre travail devient facile.

⁶⁷<http://gulpjs.com/>

⁶⁸<http://gruntjs.com/>



Gulp vs Grunt

Grunt, comme Gulp, est un outil pour définir et exécuter des tâches. La principale différence entre Grunt et Gulp est que Grunt définit les tâches en utilisant **des objets de configuration** alors que Gulp définit des tâches comme **des fonctions JavaScript**. Puisque Gulp exécute du Javascript, il offre plus de souplesse dans l'écriture de vos tâches.

Les deux ont une bibliothèque massive de plugins, où vous pouvez trouver celui qui execute la tâche souhaitée.

Installation

Je vais vous montrer un exemple de la façon dont vous pouvez utiliser Gulp pour observer les modifications apportées à vos fichiers js et exécuter automatiquement la commande **build**.

Nous devons d'abord installer Gulp globalement :

```
>_ npm install gulp-cli --global
```

Ensuite, nous installerons Gulp dans les devDependencies de notre projet :

```
>_ npm install gulp --save-dev
```

Maintenant que nous avons installé gulp, nous allons créer un **gulpfile.js** à la racine de notre projet :

gulpfile.js

```
const gulp = require('gulp');

gulp.task('default', function() {
  // Placez le code de votre prochaine tâche ici
});
```

Utilisation

Si maintenant nous executons la commande “**gulp**” dans notre console, il démarre, mais ne fait rien encore. Nous devons configurer une tâche par défaut.

Pour exécuter `babel` directement, je vais installer un plugin npm appelé `gulp-babel`⁶⁹.

```
>_  npm install gulp-babel --save-dev
```

Je vais ajouter une nouvelle *tâche gulp* nommée `babel` qui sera configurée comme étant notre tâche par défaut. Mon `gulpfile` ressemblera à ceci :

`gulpfile.js`

```
const gulp = require('gulp');
const babel = require('gulp-babel');

gulp.task('default', ['babel']);

//tâche gulp basique
gulp.task('babel', function() {
  return gulp.src('src/*.js')
    .pipe(babel({
      presets: ['es2015']
    }))
    .pipe(gulp.dest('assets/js/'))
})
```

Cette tâche demande essentiellement à `babel` de transformer tous les fichiers `js` sous le répertoire `src` en utilisant le preset `es2015` et de les mettre dans le répertoire `assets/js`.

observation (watch)

Le fait d'exécuter de `gulp` dans votre console a le même effet qu'avec `npm run build`. Ce que nous voulons atteindre ici est d'exécuter cette tâche à chaque fois qu'un fichier `js` change. Pour ce faire, nous allons créer un *watcher* dans notre `gulpfile` comme ceci :

⁶⁹<https://www.npmjs.com/package/gulp-babel>

gulpfile.js

```
const gulp = require('gulp');
const babel = require('gulp-babel');

gulp.task('default', ['watch']);

//tâche babel basique
gulp.task('babel', function() {
  return gulp.src('src/*.js')
    .pipe(babel({
      presets: ['es2015']
    }))
    .pipe(gulp.dest('assets/js/'))
})

//la tâche watch
gulp.task('watch', function() {
  gulp.watch('src/*.js', ['babel']);
})
```

Lorsque nous exécutons `gulp watch` dans notre console, gulp est à la recherche de changements dans tous nos fichiers `.js` sous le répertoire spécifié. Chaque fois que nous faisons un changement, gulp exécute notre tâche `babel` et les fichiers sous `assets/js` sont mis à jour. **Impressionant !**

Devoirs

Cet exercice de devoirs suit le précédent. Si vous n'avez pas fait le précédent, il n'est jamais trop tard pour commencer !

Puisque cette partie du chapitre est dédiée aux Task Runners, vous devez configurer un watcher avec Gulp et compiler votre code avec `Babel`, lorsqu'une modification est détectée.



Remarque

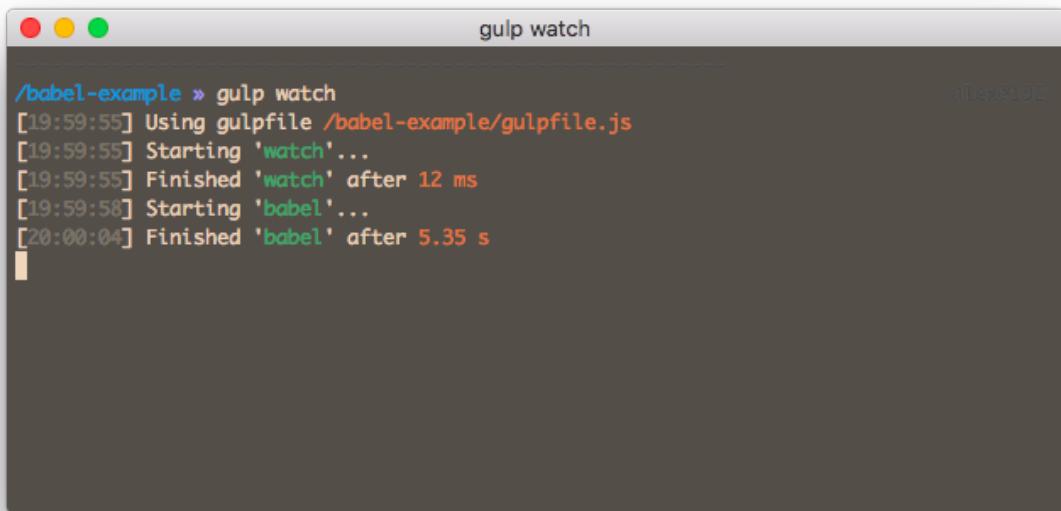
Vous avez peut-être déjà remarqué que lorsque vous exécutez Gulp, il affiche des messages dans le terminal (**“Starting”** - **“Finished”**), alors ne soyez pas si pressé et attendez que les modifications soient appliquées.



Solution potentielle

Vous pouvez trouver une solution à cet exercice [ici](#)⁷⁰.

⁷⁰<https://github.com/hoottlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter15/chapter15.2>



```
/babel-example » gulp watch
[19:59:55] Using gulpfile /babel-example/gulpfile.js
[19:59:55] Starting 'watch'...
[19:59:55] Finished 'watch' after 12 ms
[19:59:58] Starting 'babel'...
[20:00:04] Finished 'babel' after 5.35 s
```

Gulp observe !

Bundling de Modules avec Webpack

Les Module Bundlers

Notre workflow est bon avec le code actuel de `sum.js`. Nous allons étendre ses fonctionnalités, pour calculer le coût d'une pizza et d'une bière et le faire savoir au client.

`src/sum.js`

```
const pizza = 10
const beer = 5

const sum = (a, b) => a + b + '$';
console.log(`Alex, you have to pay ${sum(pizza, beer)})`)
```

Ce code semble bon, mais en supposant que tout le monde ne s'appelle pas *Alex* nous allons créer un nouveau fichier, `client.js`, qui fournira le nom du client.

src/client.js

```
export const name = 'Alex'
```

Nous allons importer le nom à partir de là.

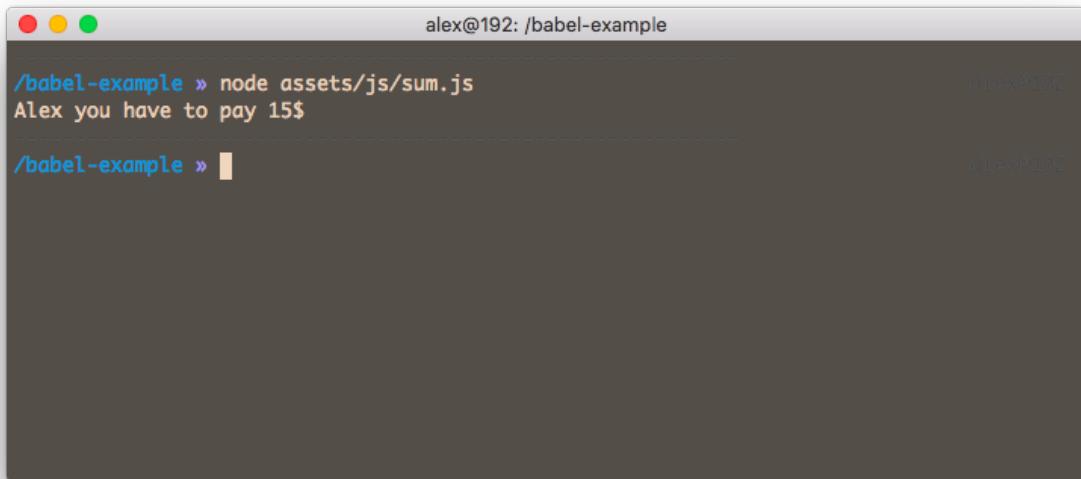
src/sum.js

```
import { name } from './client'

const pizza = 10
const beer = 5

const sum = (a, b) => a + b + '$';
console.log(` ${name} you have to pay ${sum(pizza, beer)} `)
```

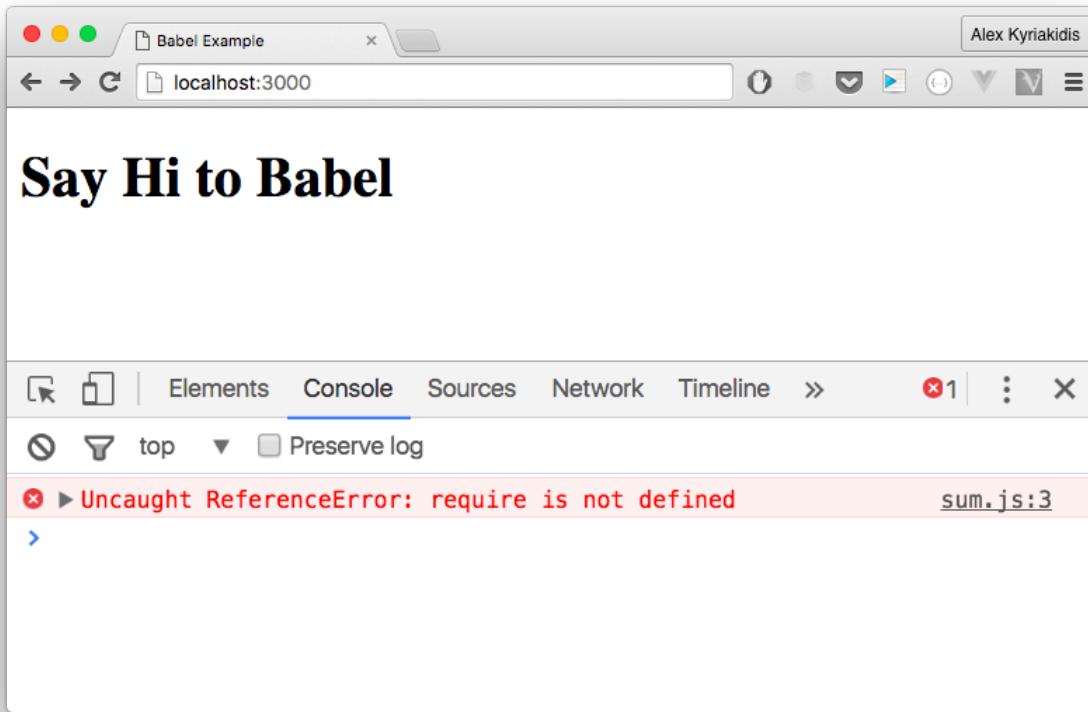
Génial ! Si nous exécutons `node assets/js/sum.js` nous obtenons la sortie attendue.



A screenshot of a terminal window titled "alex@192: /babel-example". The window shows the command `/babel-example » node assets/js/sum.js` being run, followed by the output "Alex you have to pay 15\$". The terminal has a dark background and light-colored text. The title bar and window frame are visible.

Sortie de sum.js

Vous vous attendez ici à ce que le même comportement s'applique lorsque nous ouvrons le fichier `html` dans le navigateur, mais ce n'est pas le cas ! Nous obtenons une erreur à la place.



Require n'est pas défini

Vérifiez le fichier `node assets/js/sum.js` et notez le bout de code `var_client=require('../client');` en haut. La raison pour laquelle nous obtenons l'erreur dans le navigateur est que `require()` n'existe pas dans le navigateur ni dans notre JavaScript côté client. Ce que nous devons faire est de *Regrouper* les modules dans un fichier afin qu'ils puissent être inclus dans une balise `<script>`.

The screenshot shows a code editor interface with a file tree on the left and a code editor on the right. The file tree shows a project structure with folders like 'babel-example', 'assets' (containing 'js' with files 'client.js' and 'sum.js'), 'node_modules' (with various packages), and 'src' (with files 'client.js', 'sum.js', '.babelrc', 'gulpfile.js', 'package.json', and 'sum.html'). The code editor window is titled 'sum.js — assets/js — /babel-example' and contains the following JavaScript code:

```
'use strict';

var _client = require('./client');

var pizza = 10;
var beer = 5;

var sum = function sum(a, b) {
    return a + b + '$';
};

console.log(_client.name, ' have to pay', sum(pizza, beer));
```

At the bottom of the code editor, there are tabs for 'File 0', 'Project 0', 'No Issues assets/js/sum.js', 'LF', 'UTF-8', '2 Spaces', and 'JavaScript'. There are also red status indicators for 'File 0' and 'Project 0'.

assets/js/sum.js

C'est dans ce genre de cas que nous avons besoin de **Module Bundlers** comme [Webpack⁷¹](#) ou [Browserify⁷²](#).

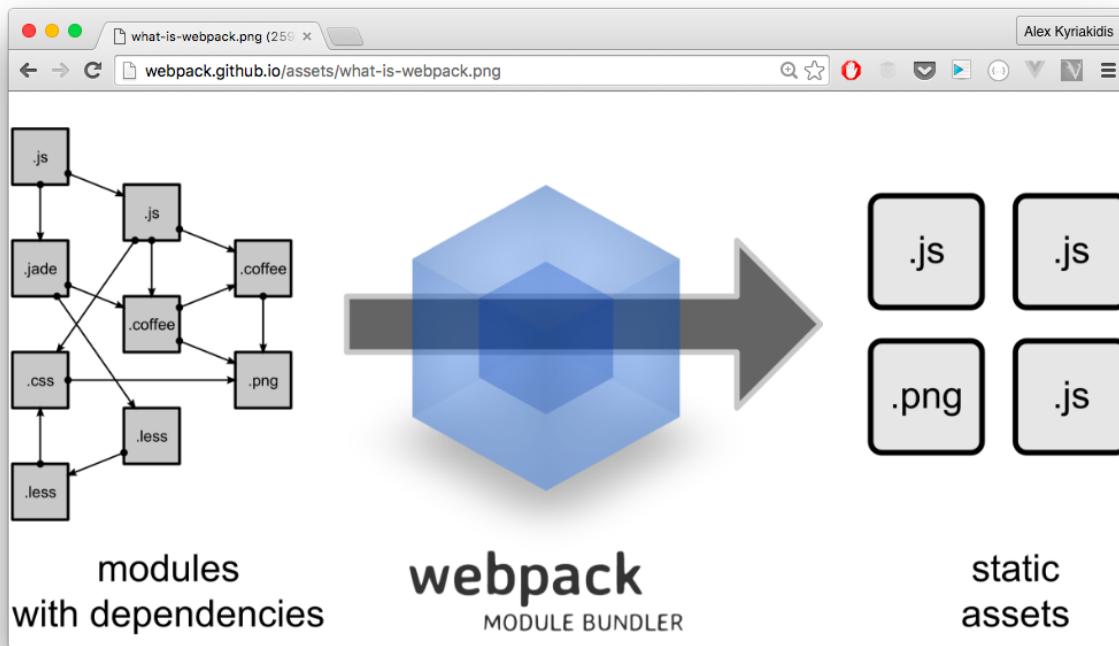
Webpack

Webpack est un Module Bundler. Il prend les modules JavaScript, comprend leurs dépendances, puis les concatène et produit des assets statiques représentant ces modules.

Je vais utiliser Webpack dans les exemples suivants. C'est parce que je crois qu'il sera utile à l'avenir pour d'autres utilisations, puisqu'il peut faire plus que regrouper des modules. En utilisant les *loaders*, nous pouvons apprendre à Webpack à transformer tout type de fichiers de la manière que nous voulons, avant de sortir le bundle final.

⁷¹<https://webpack.github.io/>

⁷²<http://browserify.org/>



Ce que fait Webpack

Installation

Je vais d'abord installer Webpack globalement, puis l'ajouter à notre projet en tant que dépendance.

```
>_ npm install webpack -g  
      npm install webpack --save-dev
```



Astuce

Au moment où j'écris lignes, il y a un bug connu lorsque vous utilisez [Vagrant](#)⁷³ sur Windows, l'exécutant de `npm install` peut échouer. Pour résoudre ce problème, quittez Vagrant, `cd` vers votre projet dans le terminal Windows et exécutez `npm install` à partir de là.

⁷³<https://www.vagrantup.com/>

Utilisation

Afin de compiler notre Javascript, nous devons donner à Webpack un point source et un point de sortie. Dans notre cas, la source est babelified `assets/js/sum.js` et comme sortie `assets/webpacked/app.js`.

```
>_ webpack assets/js/sum.js assets/webpacked/app.js
```



The screenshot shows a terminal window titled "alex@192: /babel-example". The command entered is "webpack assets/js/sum.js assets/webpacked/app.js". The output displays the following information:

```
/babel-example » webpack assets/js/sum.js assets/webpacked/app.js
Hash: 7d079f20fab1a5cd6563
Version: webpack 1.13.0
Time: 79ms
 Asset      Size  Chunks             Chunk Names
app.js    1.78 kB     0  [emitted]  main
          [0] ./assets/js/sum.js 192 bytes {0} [built]
          [1] ./assets/js/client.js 113 bytes {0} [built]

/babel-example »
```

Sortie de Webpack

Une fois la compilation terminée, nous pouvons utiliser les éléments `js`, `assets/webpacked/app.js`, `sum.html`. Nous pouvons également l'exécuter dans le terminal en utilisant `node assets/webpacked/app.js`.

Automatisation

Si vous êtes paresseux comme moi, et que vous n'aimez pas exécuter `webpack` à chaque fois que vous faites un changement, vous pouvez automatiser cette tâche. Vous pouvez configurer Webpack pour surveiller votre source et rebuilder votre bundle lorsque l'un de vos fichiers change. Je ne le ferai pas ici. Au lieu de cela, je vais l'intégrer dans Gulp, afin de démontrer comment vous pouvez combiner plusieurs outils.



En savoir plus

Si vous voulez en savoir plus sur la façon dont Webpack fonctionne et comment vous pouvez le configurer, consultez “[Beginner’s guide to Webpack](#)”⁷⁴ par Nader Dabit⁷⁵.

Pour intégrer Webpack dans Gulp, je vais utiliser un plugin appelé [webpack-stream](#)⁷⁶.

```
>_ npm install webpack-stream --save-dev
```

Après l’installation, je vais créer une nouvelle tâche avec le nom de ‘webpack’ et dire à Gulp de l’exécuter chaque fois qu’il détecte un changement, immédiatement après avoir exécuté la tâche ‘babel’.

gulpfile.js

```
const gulp = require('gulp');
const babel = require('gulp-babel');
const webpack = require('webpack-stream');

gulp.task('default', ['watch']);

//Tâche babel basique
gulp.task('babel', function() {
    return gulp.src('src/*.js')
        .pipe(babel({
            presets: ['es2015']
        }))
        .pipe(gulp.dest('assets/js/'))
})

//tâche webpack basique
gulp.task('webpack', ['babel'], function() {
    return gulp.src('assets/js/sum.js')
        .pipe(webpack({
            output: {
                path: "/assets/webpacked",
                filename: "app.js"
            }
})
```

⁷⁴<https://medium.com/@dabit3/beginner-s-guide-to-webpack-b1f1a3638460>

⁷⁵<https://twitter.com/dabit3>

⁷⁶<https://www.npmjs.com/package/webpack-stream>

```
        }))
    .pipe(gulp.dest('assets/webpacked'));
}

//la tâche watch
gulp.task('watch', function() {
    gulp.watch('src/*.js', ['babel', 'webpack']);
})
```

Cette solution n'est pas idéale. C'est juste une démonstration de la façon dont vous pouvez lier tout ce que vous avez appris ensemble. En production il existe de meilleures façons d'automatiser vos tâches de webpack.

```

const gulp = require('gulp');
const babel = require('gulp-babel');
const webpack = require('webpack-stream');

gulp.task('default', ['watch']);

//basic babel task
gulp.task('babel', function() {
  return gulp.src('src/*.js')
    .pipe(babel({
      presets: ['es2015']
    }))
    .pipe(gulp.dest('assets/js/'))
})

//basic webpack task
gulp.task('webpack', ['babel'], function() {
  return gulp.src('assets/js/sum.js')
    .pipe(webpack({
      output: {
        path: "/assets/webpacked",
        filename: "app.js"
      }
    }))
    .pipe(gulp.dest('assets/webpacked'));
})

//the watch task
gulp.task('watch', function() {
  gulp.watch('src/*.js', ['babel', 'webpack']);
})

```

File 0 | Project 0 | ✓ No Issues | gulpfile.js | 1:2 | LF | UTF-8 | 4 Spaces | JavaScript

Webpack et Gulp

Résumé

Lorsque vous souhaitez compiler de l'ES6, vous pouvez utiliser [Babel⁷⁷](#).

Pour automatiser des opérations comme celle-ci et bien d'autres (comme la minification, la compilation de SASS/LESS, etc), vous aurez besoin d'un gestionnaire de tâches comme [Gulp⁷⁸](#) ou [Grunt⁷⁹](#).

⁷⁷<http://babeljs.io/>

⁷⁸<http://gulpjs.com/>

⁷⁹<http://gruntjs.com/>

Pour créer des buldle vous pouvez utiliser **Webpack**⁸⁰ ou **Browserify**⁸¹.

Dans le prochain chapitre, nous allons plonger dans composants à fichier unique de Vue.js et utiliser plusieurs outils que Vue vous fournit ainsi que les outils que nous avons appris.



Remarque

Si vous avez trouvé ce chapitre difficile à comprendre, ne vous inquiétez pas. Vous n'avez pas besoin de vous souvenir de toutes ces choses. C'était juste une démonstration afin de vous donner une meilleure compréhension de la façon dont les choses fonctionnent. Dans le prochain chapitre, nous utiliserons *Template de démarrage*. Là, des choses comme **bundling de modules, automatisation des tâches, construire sur le changement** et beaucoup plus sont déjà mise en place et nous allons en profiter.

⁸⁰<https://webpack.github.io/>

⁸¹<http://browserify.org/>

Utilisation des Composants à fichier unique

Comme promis, dans ce chapitre nous allons passer en revue les composants de fichiers uniques. Pour utiliser ces composants Vue.js, nous avons besoin d'outils comme **Webpack** avec **vue-loader** ou **Browserify** avec **vueify**. Pour nos exemples, nous allons utiliser Webpack dont nous avons déjà vu le fonctionnement. Si vous préférez Browserify ou encore un autre outils, n'hésitez pas à l'utiliser.

Les composants de fichier unique ou les composants Vue encapsulent leurs styles CSS, leur template et leur code JavaScript, tout en un seul fichier à l'aide de l'extension `.vue`. Et c'est là que webpack entre en jeu pour de regrouper ce nouveau type de fichier avec les autres et créer un bundle.

Webpack utilise **vue-loader**⁸² pour transformer les composants Vue.js en modules Javascript. **vue-loader** Fournit également un ensemble de fonctionnalités telles que ES2015 activé par défaut, fournit à chaque composant sa propre portée css, et plus encore.

vue-cli

Pour éviter de configurer Webpack et créer un nouveau workflow à partir de zero, nous utiliserons **vue-cli**.



Info

vue-cli⁸³ est un simple programme en ligne de commande pour le scaffolding de projets Vue.js.

Cet outil impressionnant est le moyen le plus rapide d'obtenir un build préconfigurée. Il offre des templates avec hot-reload, lint-on-save, tests unitaires...etc. Actuellement, il offre des templates de démarrage pour webpack et browserify, mais vous pouvez si nécessaire [créer le votre](#)⁸⁴.

Les Templates Vue.js

Ce que la CLI fait, c'est de récupérer des templates du répertoire officiel [Vue.js](#)⁸⁵ où il ya actuellement 5 modèles de disponibles. Je crois que ce nombre va croître dans un avenir proche. Vous pouvez vérifier ce qu'ils incluent sur GitHub.

⁸²<https://github.com/vuejs/vue-loader>

⁸³<https://github.com/vuejs/vue-cli>

⁸⁴<https://github.com/vuejs/vue-cli#custom-templates>

⁸⁵<https://github.com/vuejs-templates>

Tous les modèles contiennent un fichier *package.json* qui gère les dépendances du projet et est livré avec un ensemble de scripts NPM prédéfinis.

En utilisant les modèles de projet Vue, vous obtenez beaucoup de fonctionnalités. Par exemple, la description du modèle “webpack” indique qu’il s’agit d’une “*installation Webpack complete + vue-loader avec hot reload, linting, tests & extraction css*”

Installation

Nous allons nous en tenir à l’approche d’installation de Webpack et installer **vue-cli** globalement à l’aide de la commande suivante.

```
>_ npm install vue-cli -g
```

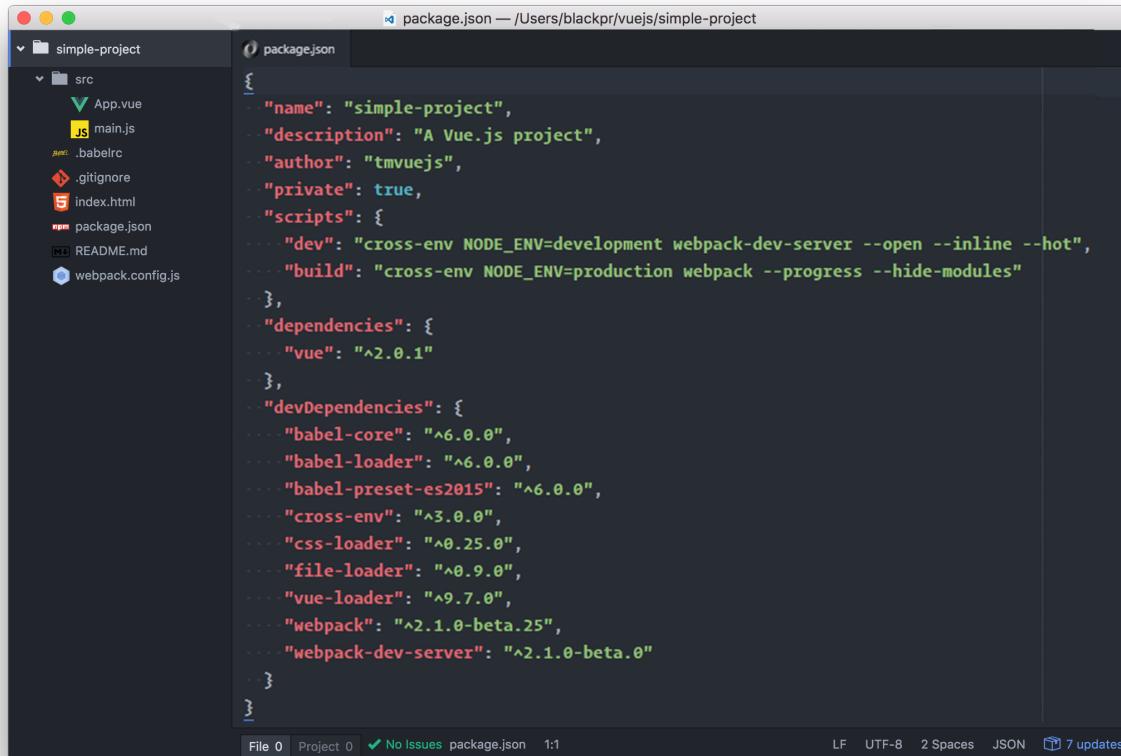
Utilisation

Avec le CLI, vous pouvez exécuter **vue init <nom-modèle> <nom-projet>** où **<nom-modèle>** est le nom du modèle (officiel ou personnalisé) et **<nom-projet>** est le nom du répertoire / projet que vous allez créer.

Donc, si vous exécutez

```
>_ vue init webpack-simple project-simple
```

Vous allez avoir un répertoire nommé **project-simple** avec la structure suivante :



```

{
  "name": "simple-project",
  "description": "A Vue.js project",
  "author": "tmvuejs",
  "private": true,
  "scripts": {
    "dev": "cross-env NODE_ENV=development webpack-dev-server --open --inline --hot",
    "build": "cross-env NODE_ENV=production webpack --progress --hide-modules"
  },
  "dependencies": {
    "vue": "^2.0.1"
  },
  "devDependencies": {
    "babel-core": "^6.0.0",
    "babel-loader": "^6.0.0",
    "babel-preset-es2015": "^6.0.0",
    "cross-env": "^3.0.0",
    "css-loader": "^0.25.0",
    "file-loader": "^0.9.0",
    "vue-loader": "^9.7.0",
    "webpack": "^2.1.0-beta.25",
    "webpack-dev-server": "^2.1.0-beta.0"
  }
}

```

Structure Webpack simple

Pour notre exemple, nous allons utiliser le modèle complet de webpack, nous allons donc exécuter la commande suivante :

```
>_ vue init webpack stories-classic-project
```



Info

Utilisez la commande `vue list` pour voir tout les modèles officiels disponibles.

Lorsque vous initialisez un nouveau projet, vous serez invité à remplir certains détails comme le nom, la version, l'auteur, etc.

À un certain point, on vous demandera de **choisir un preset ESLint**. Les options disponibles sont `feross/standard`⁸⁶ et `airbnb/javascript`⁸⁷.

⁸⁶<https://github.com/feross/standard>

⁸⁷<https://github.com/airbnb/javascript>

J'ai créé un tableau pour comparer les deux styles d'écriture de code afin que vous puissiez obtenir une meilleure compréhension des règles qui s'appliquent à chaque style.

Rules	feross/standard	airbnb/javascript
Indentation	2 espaces	2 espaces
Point-virgule	Non !	Oui
Variables non utilisées	Not allowed	Not allowed
Guilleumets	simple	simple
Utilisation de === au lieu de ==	Oui	Oui
Nombre de lignes vides autorisées	1	2
Espace après le nom de fonction	Oui	Non
Début de ligne avec [Non	Oui
Fin de fichier par un saut à la ligne	Oui	Oui
Virgules trainante	Non	Non



Standard vs Airbnb

Les règles dans le tableau sont quelques-unes des nombreuses appliquées dans chaque style. Pour les examiner et décider ce qui vous convient le mieux, consultez leurs dépôts Github.

Après avoir sélectionné un style, vous obtiendrez des invites pour installer plusieurs outils comme [Karma-Mocha⁸⁸](#) et [Nightwatch⁸⁹](#). Nous n'allons pas avoir besoin de ces outils pour le moment, répondez donc aux questions par non et continuez.

[! Installation du modèle de Vue⁹⁰](#)



Info

Karma est un plugin adaptateur pour le cadre de teste Mocha⁹¹.

Nightwatch permet d'écrire des tests automatisés de navigateurs qui s'exécutent sur un serveur Selenium⁹².

Les Templates Webpack

Pour en finir avec la configuration de notre projet, nous devons installer ses dépendances. Passons à l'action :

⁸⁸<https://github.com/karma-runner/karma-mocha>

⁸⁹<http://Nightwatchjs.org/>

⁹⁰[images/screenshots/singles/prompts.png](#)

⁹¹<https://mochajs.org/>

⁹²<http://www.seleniumhq.org/>

```
>_ cd stories-classic-project  
    npm install  
    npm run dev
```

Le terminal affiche ***Listening at http://localhost:8080***. Vous devez attendre que le message **webpack: bundle is now VALID** soit affiché.

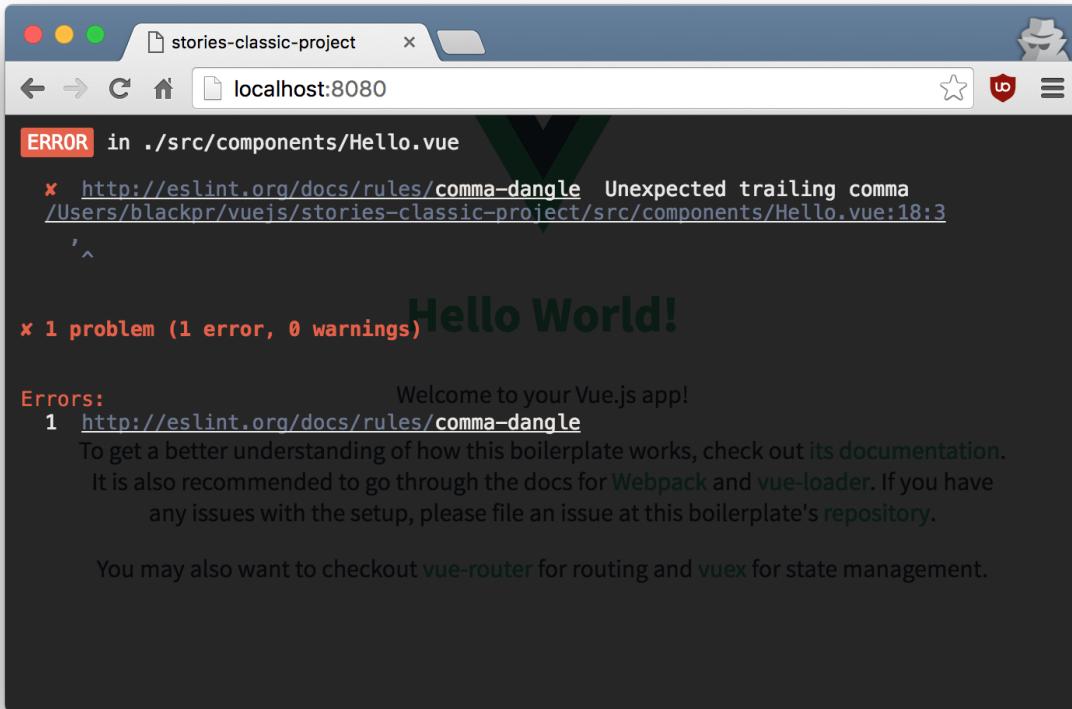
```
1. npm run dev (node)  
  └─ lodash._arraycopy@3.0.0  
    └─ lodash._arrayeach@3.0.0  
      └─ lodash._createassigner@3.1.1  
        └─ lodash._isiterateeceil@3.0.9  
          └─ lodash.restparam@3.6.1  
            └─ lodash.getnative@3.9.1  
              └─ lodash.istypedarray@3.0.6  
                └─ lodash.toplainobject@3.0.0  
                  └─ lodash._basecopy@3.0.1  
  
→ stories-classic-project npm install  
→ stories-classic-project npm run dev  
  
> y@1.0.0 dev /Users/blackpr/vuejs/stories-classic-project  
> node build/dev-server.js  
  
Listening at http://localhost:8080  
  
webpack built 459f183651c77b8f6e8d in 1815ms  
Hash: 459f183651c77b8f6e8d  
Version: webpack 1.13.1  
Time: 1815ms  
  Asset      Size  Chunks      Chunk Names  
  app.js    1.06 MB    0  [emitted]  app  
index.html 234 bytes       [emitted]  
Child html-webpack-plugin for "index.html":  
  Asset      Size  Chunks      Chunk Names  
  index.html 21.4 kB    0  
webpack: bundle is now VALID.
```

Server Running...



Avertissement

Soyez prudent, vous devez être explicite dans votre code. Sinon, vous obtiendrez des erreurs pour les lignes vides supplémentaires entre les blocs, les espaces à la fin, l'indentation autre que 2 espaces et d'autres choses qui ne suivent pas le [style de code sélectionnée](#).



Superposition d'erreurs



Remarque

Si vous utilisez `webpack-simple`, vous aurez toujours les fonctionnalités de base, mais la superposition d'erreurs ne s'affichera pas dans le navigateur, vérifiez le terminal pour toutes les erreurs.

Project Structure

Après avoir terminé les étapes ci-dessus, vous devriez avoir un répertoire de projet rempli avec tous les fichiers nécessaires.

The screenshot shows a code editor interface with a sidebar containing a project tree. The tree includes 'stories-classic-project' with subfolders 'build', 'config', 'node_modules', 'src' (containing 'assets', 'components' with files 'Hello.vue' and 'App.vue'), 'static', and configuration files like '.babelrc', '.editorconfig', '.eslintrc.js', '.gitignore', 'index.html', 'package.json', and 'README.md'. The main editor area displays the 'main.js' file content:

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

/* eslint-disable no-new */
new Vue({
  el: '#app',
  template: '<App/>',
  components: { App },
});
```

At the bottom of the editor, there are status indicators: 'File 0', 'Project 0', '✓ No Issues', 'src/main.js', 'LF', 'UTF-8', '2 Spaces', 'JavaScript', and '7 updates'.

Structure Webpack

Les fichiers que vous allez généralement utiliser sont :

1. **index.html**
2. **main.js**
3. Fichiers des répertoires **src** et **src/components**

index.html

Commençons par le fichier **index.html**, qui devrait ressembler à ça :

index.html

```
<html>
  <head>
    <meta charset="utf-8">
    <title>stories-classic-project</title>
  </head>
  <body>
    <app></app>
    <!-- les fichiers buildés par webpack seront automatiquement injectés ici -->
  </body>
</html>
```

Comme vous pouvez le voir, c'est une configuration assez simple avec un composant déjà inclus. Le commentaire fait référence au script, `app.js`, qui est la sortie de Webpack. Cela signifie essentiellement qu'après que Webpack ai regroupé les scripts, il injectera automatiquement le script produit ici, afin que vous n'ayez pas à l'inclure manuellement.

Hello.vue

Si vous suivez, accédez au fichier `src / components` et ouvrez `Hello.vue` pour voir à quoi ressemble un fichier `.vue`.

src/components/Hello.vue

```
<template>
  <div class="hello">
    <h1>{{ msg }}</h1>
    <h2>Essential Links</h2>
    ...
  </div>
</template>

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

```
<!-- Ajout de l'attribut "scoped" pour limiter la portée du CSS uniquement à ce \
composant-->
<style scoped>
h1, h2 {
  font-weight: normal;
}
...
</style>
```

À l'intérieur de la balise `<script>`, le composant ne contient que ces *données*. Il n'est pas nécessaire de définir un template. Il sera automatiquement bindé si le bloc `<template>` existe. Le bloc `<template>` définit bien sûr le template du composant. Imaginez `<template>` comme une balise `<template-hello>` dans votre code HTML.

Nous pourrions n'avoir que le bloc `<script>`, mais de cette façon nous ne profiterions pas des avantages de Single File Component (ou composant à fichier unique).

N'oubliez pas que la fonctionnalité ES6 ‘export’ est gérée par ces outils magiques (transpilers + modules bundlers) que nous avons installés.



Avertissement

Chaque fichier `.vue` ne doit pas contenir plus d'un bloc `<script>`. Chaque template doit contenir exactement **un élément racine**, comme `<div id = hello> ... </div>` qui encapsule tous les autres éléments.

Le script doit exporter un *objet d'options de composant Vue.js*. L'exportation d'un constructeur étendu créé par `Vue.extend()` est également prise en charge, mais un objet simple est préféré.

ES6 nous permet d'avoir n'importe quel nombre d'exportations, mais dans les composants de fichier unique ce n'est pas le cas.

Si vous essayez de faire des exportations supplémentaires, vous obtiendrez un avertissement similaire à celui ci : `[vue-loader] src/components/Hello.vue: named exports in /*.vue files are ignored.`

Le bloc `<style>` définit les styles CSS comme prévu.

App.vue

Le fichier `App.vue` se trouve dans le répertoire `src` et est celui qui contient le template principal de l'application. Ce composant est habituellement responsable d'inclure les autres composants.

`App.vue` a quelques lignes de plus avec des textes et des styles, mais puisque nous nous concentrons sur la structure, nous l'avons raccourcie un peu.

src/App.vue

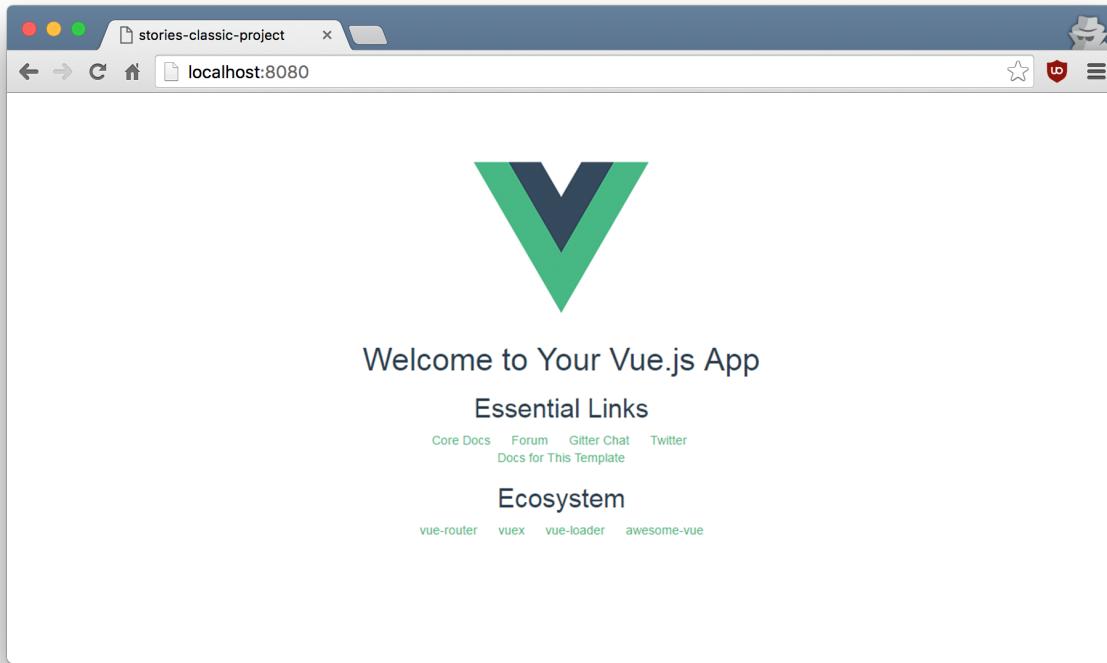
```
<template>
  <div id="app">
    
    <hello></hello>
  </div>
</template>

<script>
import Hello from './components/Hello'

export default {
  name: 'app',
  components: {
    Hello
  }
}
</script>

<style>
...
</style>
```

Il a la même structure que le fichier *Hello.vue* que nous avons vu ci-dessus. Par défaut, il y a un objet **components** qui contient le composant *Hello*. À l'intérieur de cet objet, nous importerons tous les nouveaux composants. Dans le template, il y a la balise `<hello></hello>` et le template du composant *Hello* sera donc affiché.



Page d'accueil du projet

main.js

Le fichier `main.js` dans le dossier `src` est, comme vous pouvez l'imaginer, notre script principal.

`src/main.js`

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

/* eslint-disable no-new */
new Vue({
  el: '#app',
  template: '<App/>',
  components: { App }
})
```

Il importe `Vue` en tant que module à partir du dossier `node_modules` et de la même façon `App` du répertoire `src`. Voici notre instance `Vue` et dans l'objet `components`, il y a l'instance `App`.

Chaque fois que vous avez besoin d'importer un script ou un composant globalement, vous pouvez le faire dans `main.js`.



Remarque

L'option `template: '<App/>'` représente le template de sortie du composant qui doit être injecté dans `index.html` que nous avons mentionné plus tôt.

`<App/>` affiche le même rendu que `<App></App>` et `<app></app>`.



Info

Vous trouverez plus d'informations sur la structure du projet du template Webpack dans la documentation [documentation⁹³](#)

Création des fichiers `.vue`

Nous avons vu à quoi ressemble un composant de fichier unique et comment il est utilisé dans un projet. Il est temps pour nous d'en créer quelques-uns dans un scénario réel. Supposons que nous voulons créer une sorte de réseau social ou un forum, où les utilisateurs postent leurs histoires et leurs expériences. Pour ce faire, nous allons avoir besoin de 2 formulaires : un pour l'enregistrement et un autre pour le login, et une page pour afficher les histoires des utilisateurs.

Avant de commencer, nous allons inclure Bootstrap globalement afin d'être en mesure d'utiliser ses styles dans tous nos composants. Pour ce faire, nous devons mettre à jour notre `index.html`.

`index.html`

```

<html>
  <head>
    <meta charset="utf-8">
    <title>stories-classic-project</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/
/css/bootstrap.min.css">
  </head>
  <body>
    <div id="app"></div>
    <!-- les fichiers buildé seront injectés ici -->
  </body>
</html>

```

Créons le formulaire de connexion dans un nouveau fichier `Login.vue`.

⁹³<http://vuejs-templates.github.io/webpack/structure.html>

src/components/Login.vue

```
<template>
  <div id="login">
    <h2>Sign in</h2>
    <input type="email" placeholder="Email address">
    <input type="password" placeholder="Password">
    <button class="btn">Sign in</button>
  </div>
</template>

<script>
export default {
  created () {
    console.log('login')
  }
}
</script>
```

Et voilà qui est fait. Afin de visualiser le fichier dans le navigateur, nous devons inclure notre composant *Login* quelque part. Nous allons donc l'importer dans le composant principal *App* et l'ajouter à son objet *components*.

src/App.vue

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

<script>
import Login from './components/Login.vue'
import Hello from './components/Hello'

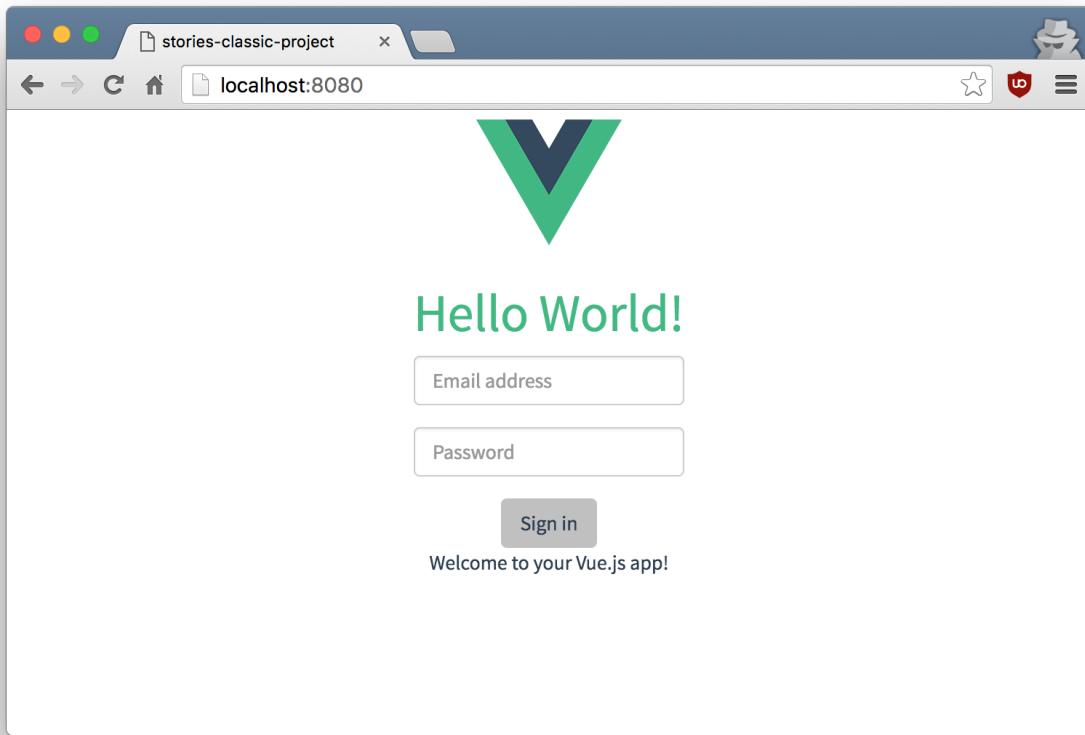
export default {
  name: 'app',
  components: {
    Hello,
    Login
  }
}
```

```
</script>  
  
<style>  
...  
</style>
```

Si vous rechargez votre navigateur, vous ne verrez pas encore le composant *Login*, Parce que nous devons le référencier dans le tempalte. Placez-le en dessous de `<hello></hello>` et vous aurez un bon formulaire de connexion !

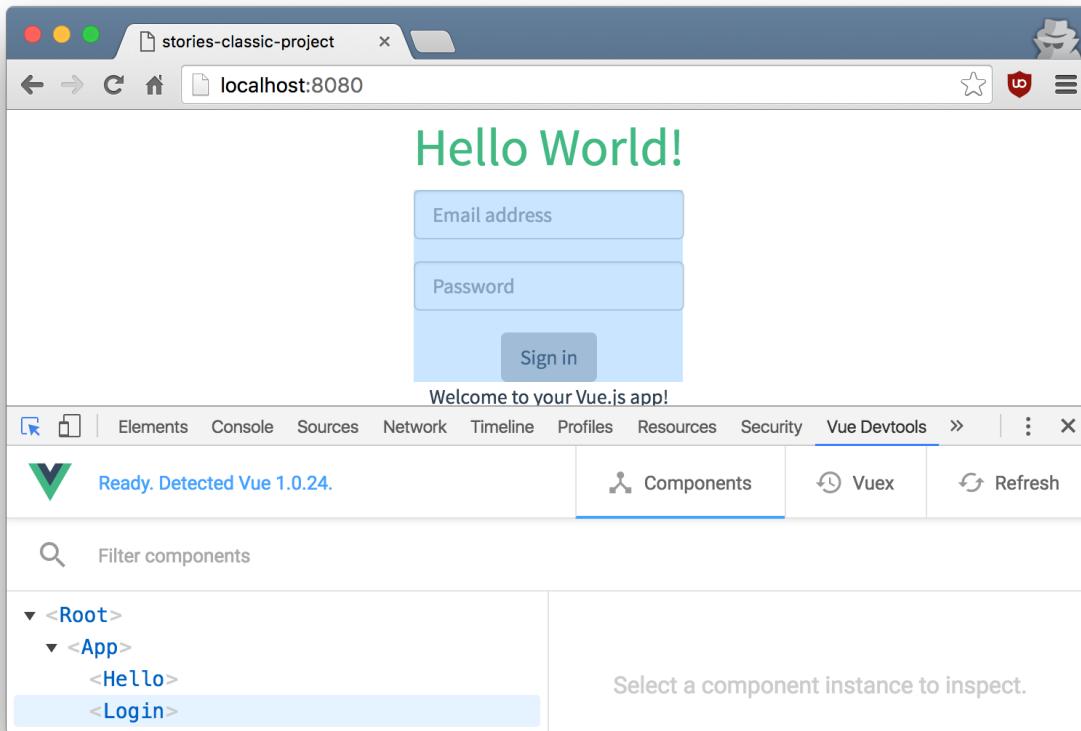
src/App.vue

```
<template>  
  <div id="app">  
      
    <hello></hello>  
    <login></login>  
  </div>  
</template>  
...  
...
```



Composant de Login

Si vous ouvrez la console du navigateur, vous devriez voir le message `login` que nous émettons lors de la création du composant. Si vous ouvrez vue-devtools, ce qui est fortement recommandé, vous devriez également le voir dans l'arborescence des composants.



arborescence des Composants

Créons un autre composant, cette fois pour l'enregistrement.

src/components/Register.vue

```
<template>
<div id="register">
  <h2>Register Form</h2>
  <input placeholder="First Name" class="form-control">
  <input placeholder="Last Name" class="form-control">
  <input placeholder="Email address" class="form-control">
  <input placeholder="Pick a password" class="form-control">
  <input placeholder="Confirm password" class="form-control">
  <button class="btn">Sign up</button>
</div>
</template>

<script>
export default {
```

```
created () {
  console.log('register')
}
</script>
```

Nous pouvons ensuite l'importer dans le fichier *App.vue*.

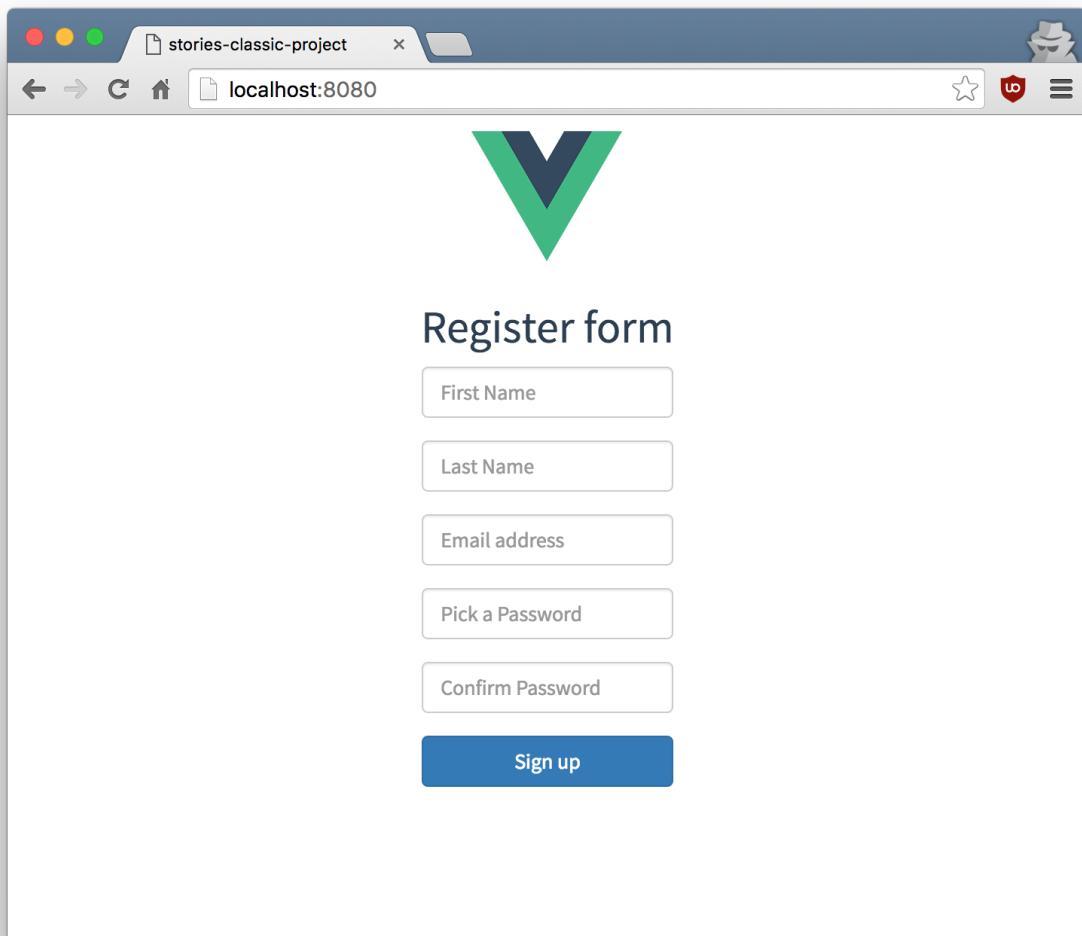
src/App.vue

```
<template>
<div id="app">
  ...
  <!-- <hello></hello> -->
  <!-- <login></login> -->
  <register></register>
  ...
</div>
</template>

<script>
// import Hello from './components/Hello'
// import Login from './components/Login'
import Register from './components/Register'

export default {
  name: 'app',
  components: {
    // Hello,
    // Login,
    Register,
  }
}
</script>
...
```

Le template du composant Register apparaît lorsque nous vérifions dans le navigateur.



Composant Register



Remarque

Les autres composants sont commentés parce que nous ne voulons pas les afficher l'un sous l'autre. Le composant `Hello` est là par défaut, mais nous ne l'utiliserons pas dans d'autres exemples, nous allons donc le supprimer.

Nous avons dit que nous travaillions sur un réseau social (ou quelque chose de similaire), nous voulons donc un endroit pour afficher les histoires. Ainsi, nous allons créer un composant `Stories` qui lorsqu'il est rendu, affichera toutes les histoires racontées par les utilisateurs.

src/components/Stories.vue

```
<template>
  <ul class="list-group">
    <li v-for="story in stories" class="list-group-item">
      {{ story.writer }} said "{{ story.plot }}"
      Story upvotes {{ story.upvotes }}.
    </li>
  </ul>
</template>

<script>
  export default {
    data () {
      return {
        stories: [
          {
            plot: 'My horse is amazing.',
            writer: 'Mr. Weebly',
            upvotes: 28,
            voted: false
          },
          {
            plot: 'Narwhals invented Shish Kebab.',
            writer: 'Mr. Weebly',
            upvotes: 8,
            voted: false
          },
          {
            plot: 'The dark side of the Force is stronger.',
            writer: 'Darth Vader',
            upvotes: 52,
            voted: false
          },
          {
            plot: 'One does not simply walk into Mordor',
            writer: 'Boromir',
            upvotes: 74,
            voted: false
          }
        ]
      }
    }
  }
</script>
```

```
    }
</script>
```

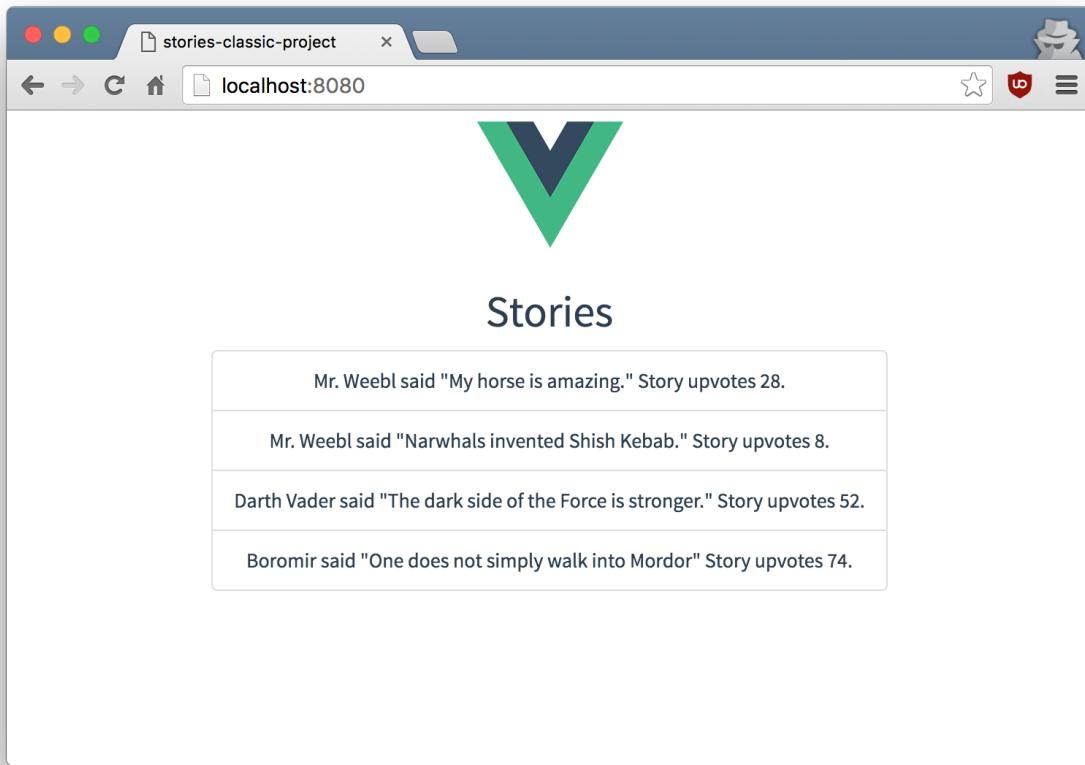
Il s'agit du fichier *Stories.vue*. Nous pouvons l'utiliser dans notre fichier principal *App.vue*. À ce stade, les histoires sont codées en dure pour plus de simplicité. Il est temps d'importer notre nouveau composant comme nous l'avons fait avec les autres.

src/App.vue

```
<template>
  <div id="app">
    
    <!-- <login></login> -->
    <!-- <register></register> -->
    <stories></stories>
  </div>
</template>

<script>
// import Login from './components/Login.vue'
// import Register from './components/Register.vue'
import Stories from './components/Stories.vue'
export default {
  components: {
    Login,
    Register,
    Stories
  }
}
</script>

<style>
...
</style>
```



Composant Stories

Génial ! Maintenant nous avons une page pour afficher toutes nos données.

Composants Imbriqués

Nous aimerais être en mesure d'afficher les histoires les plus ‘célèbres’ à n’importe quel endroit ou nous voulons. Ainsi, après la création du composant *Famous*, nous devrions pouvoir l'utiliser n'importe où.

```
{lang="HTML", title="src/components/Famous.vue", line-numbers="off"} <template>
<div id="famous"> <h2>Trending stories<strong>({{famous.length}})</strong></h2> <ul class="list-group"> <li v-for="story in famous" class="list-group-item"> {{ story.writer }} said "{{ story.plot }}". Story upvotes {{ story.upvotes }}. </li> </ul> </div> </template>
```

```
1 <script>
2 export default {
3   computed: {
4     famous () {
5       return this.stories.filter(function (item) {
6         return item.upvotes > 50
7       })
8     }
9   },
10  data () {
11    return {
12      stories: [
13        {
14          plot: 'My horse is amazing.',
15          writer: 'Mr. Weebl',
16          upvotes: 28,
17          voted: false
18        },
19        {
20          plot: 'Narwhals invented Shish Kebab.',
21          writer: 'Mr. Weebl',
22          upvotes: 8,
23          voted: false
24        },
25        {
26          plot: 'The dark side of the Force is stronger.',
27          writer: 'Darth Vader',
28          upvotes: 52,
29          voted: false
30        },
31        {
32          plot: 'One does not simply walk into Mordor',
33          writer: 'Boromir',
34          upvotes: 74,
35          voted: false
36        }
37      ]
38    }
39  }
40 }
41 </script>
```

Il s'agit du fichier entier *Famous.vue*. Nous avons **filtré** le tableau `stories` en utilisant des propriétés

calculées (computed properties). Comme nous l'avons vu dans les chapitres précédents, et avons créé un **template** pour les afficher.



Remarque

Le tableau **stories** est à nouveau codé ici et les données sont les mêmes que précédemment.

C'est une mauvaise pratique, plus tard nous allons trouver un moyen de définir le tableau ***stories**** une fois et le partager entre tous les composants.

Mais où pourrions-nous utiliser cette composante ? Une idée est de l'avoir dans la page d'inscription, Afin que l'utilisateur puisse lire les histoires à la une. Cela signifie - dans le projet actuel - que nous avons besoin d'avoir le composant « Famous » dans le registre « un ». Eh bien, cela peut être fait de la même manière que nous l'avons fait à l'intérieur de App.vue.

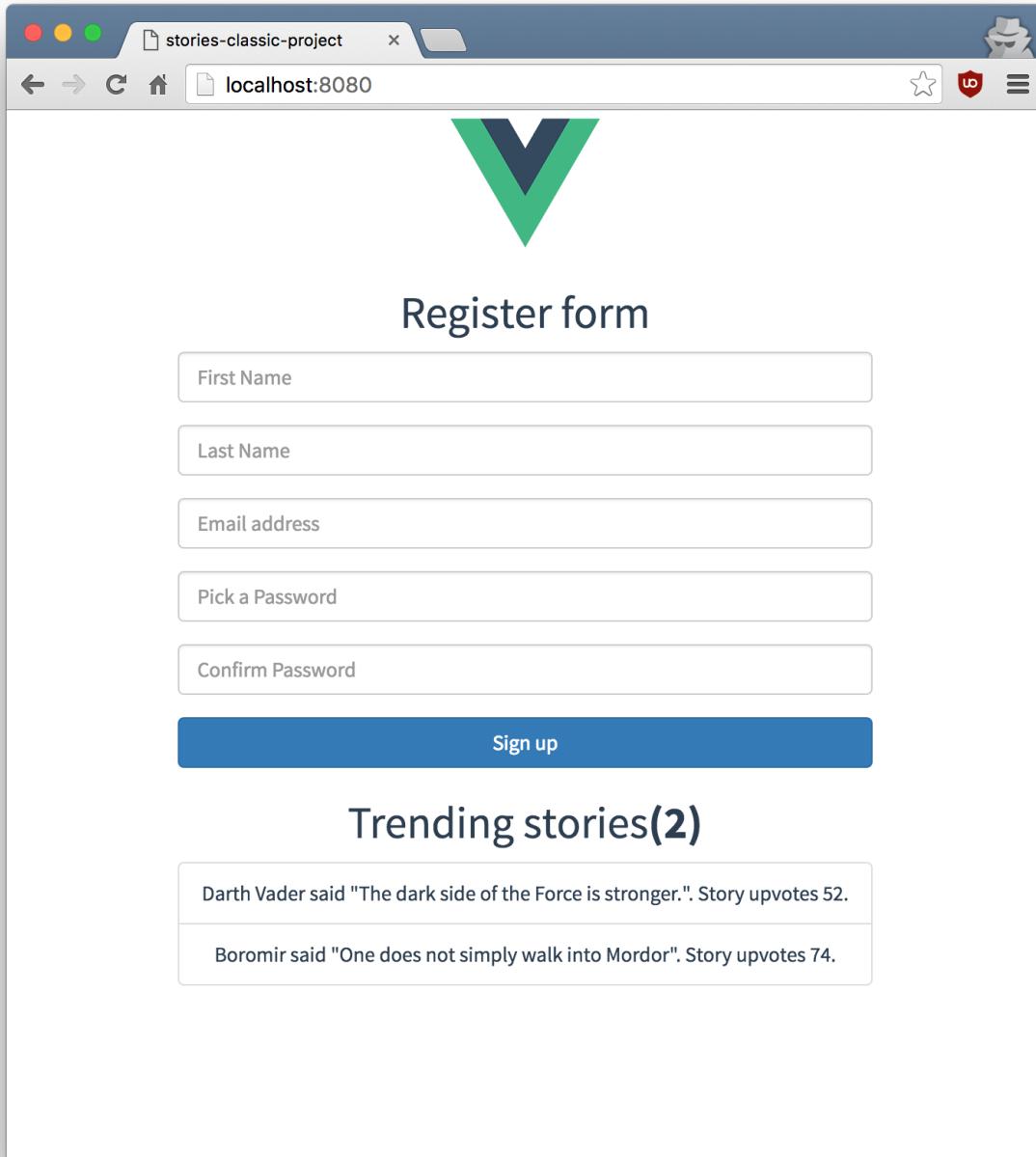
Ouvrez **Register.vue** et importez le, puis référez-le dans le template.

src/components/Register.vue

```
<template>
  <div id="register">
    <h2>Register Form</h2>
    ...
    <famous></famous>
  </div>
</template>

<script>
import Famous from './Famous.vue'

export default {
  components: {
    Famous
  },
  created () {
    console.log('register')
  }
}
</script>
```



Page d'enregistrement et histoires connues

Faites attention au chemin du fichier d'importation. Maintenant que les deux fichiers sont dans le même répertoire, vous devez utiliser `./Famous` au lieu du chemin complet. C'est une erreur facile à reproduire, surtout si vous n'êtes pas familier avec le concept !



Exemples de code

Vous trouverez les exemples de code de ce chapitre sur [GitHub⁹⁴](#).

⁹⁴<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter16/16.3>

Elimination des états dupliqués

Dans les exemples précédents, nous avons codé en dur les données - le tableau des histoires - au sein de chaque composant. Ce n'est pas une bonne façon de travailler avec les données.

Lorsque plus d'un composant utilise les mêmes données, il est recommandé de créer / d'extraire le tableau une fois, puis de trouver un moyen de le partager entre les composants de l'application.

Stories.vue et *Famous.vue* utilisent le même tableau `stories`. Nous examinerons deux façons de partager les données :

1. En utilisant les propriétés de composant.
2. En utilisant un store global.

Partage de données à l'aide des propriétés

La première chose que nous allons faire est de déplacer le tableau `stories` dans le composant `App`.

src/App.vue

```
1 <script>
2 ...
3
4 export default {
5   components: {
6     ...
7   },
8   data () {
9     // voici le tableau stories
10    return {
11      stories: [
12        {
13          plot: 'My horse is amazing.',
14          writer: 'Mr. Weebly',
15          upvotes: 28,
16          voted: false
17        },
18        {
19          plot: 'Narwhals invented Shish Kebab.',
```

```
20      writer: 'Mr. Weeb1',
21      upvotes: 8,
22      voted: false
23    },
24    {
25      plot: 'The dark side of the Force is stronger.',
26      writer: 'Darth Vader',
27      upvotes: 52,
28      voted: false
29    },
30    {
31      plot: 'One does not simply walk into Mordor',
32      writer: 'Boromir',
33      upvotes: 74,
34      voted: false
35    }
36  ]
37 }
38 }
39 }
40 </script>
```

L'étape suivante consiste à supprimer `data()` des composants `Stories` et `Famous` et à déclarer la propriété `stories`.

Faisons-le pour le premier composant.

`src/components/Stories.vue`

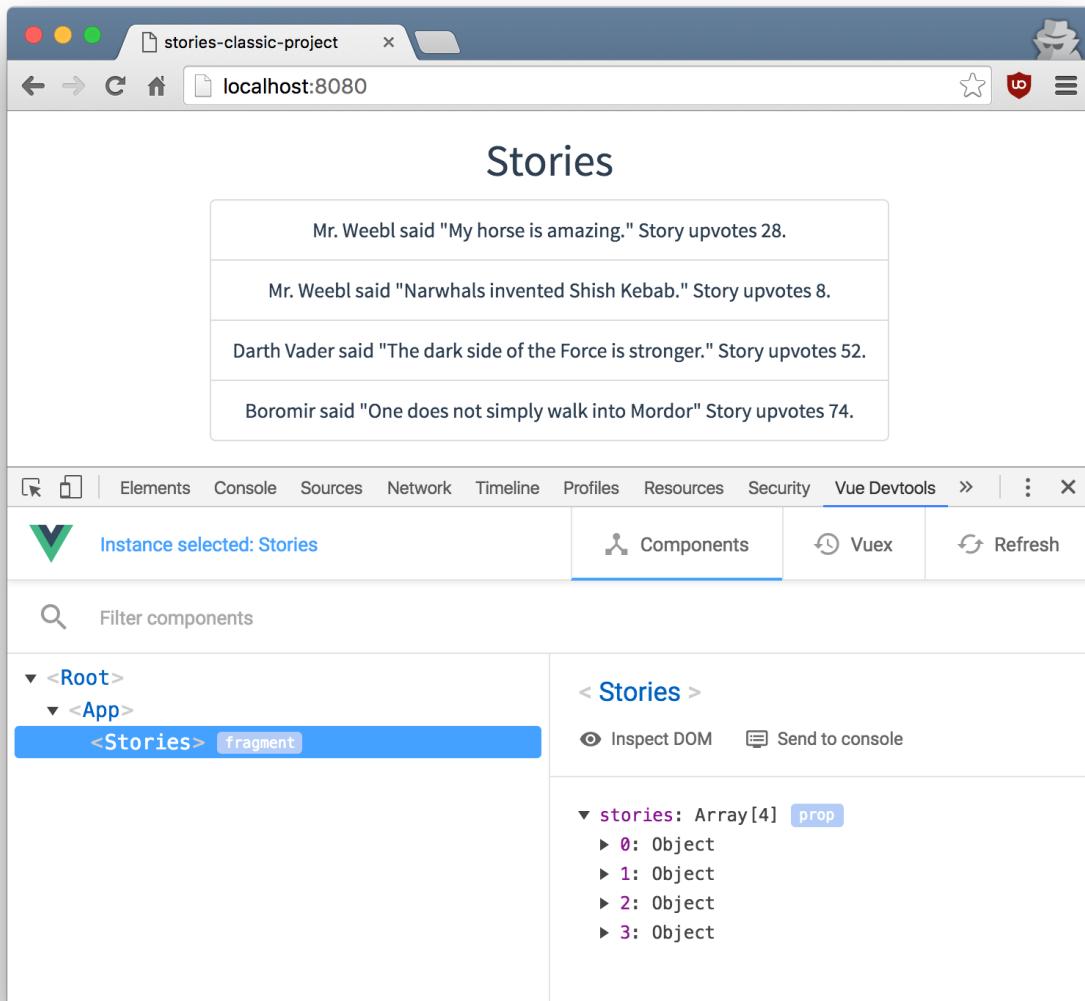
```
1 <script>
2   export default {
3     props: ['stories']
4   }
5 </script>
```

Nous devons actualiser la façon dont nous référençons notre composant dans `App.vue`.

src/App.vue

```
1 <template>
2   <div id="app">
3     ...
4     <stories :stories="stories"></stories>
5     ...
6     <p>
7       Welcome to your Vue.js app!
8     </p>
9   </div>
10 </template>
```

Ici, nous allons binder la propriété `stories` au tableau `stories`.



Même résultat en utilisant les propriétés

Nous avons à nouveau obtenu nos 'stories' extraites du composant parent !

Nous ne pouvons pas faire la même chose pour le composant Famous car il n'est pas référencé à l'intérieur de `App.vue`. Nous devrons passer notre tableau au composant Register afin de le passer au composant Famous.

src/App.vue

```
1 <template>
2   <div id="app">
3     ...
4     <register :stories="stories"></register>
5     ...
6   </div>
7 </template>
```

src/components/Register.vue

```
1 <template>
2   <h2>Register Form</h2>
3   ...
4   <famous :stories="stories"></famous>
5 </template>
6
7 <script>
8 import Famous from './Famous'
9
10 export default {
11   components: {
12     Famous
13   },
14   props: ['stories']
15 }
16 </script>
```

src/components/Famous.vue

```
1 <script>
2   export default {
3     props: ['stories'],
4
5     computed: {
6       famous () {
7         return this.stories.filter(function (item) {
8           return item.upvotes > 50
9         })
10      }
11    }
12  </script>
```

```
12    }
13 </script>
```

Cette implémentation fonctionne, mais n'est pas efficace car le composant Famous n'est pas **indépendant**. Cela signifie que nous ne pouvons pas l'utiliser partout où nous voulons, à moins que nous ne transmettions les données du composant racine, *App.vue*.

Dans un scénario où un composant non indépendant est profondément imbriqué, vous devrez passer une propriété inutile d'un composant à un composant juste pour pouvoir l'utiliser. Dans notre cas, si nous voulons utiliser Famous dans les widgets de sidebar de Register, nous devrions porter le tableau `stories` tout au long de la chaîne.

App -> Register -> Sidebar -> WidgetX -> Famous

Store Global

Le ‘methode des props’ semblait correcte au début, mais comme on le voit dans le composant Famous, quand un projet devient plus grand et que les composants sont imbriqués dans d’autres composants, la gestion des données et le partage entre eux devient vraiment difficile à suivre.

Donc, rendons les données de nos exemples un peu plus faciles à manipuler. Nous pouvons extraire les données `stories` vers un fichier `.js`, les stocker dans une **constante** et les importer ultérieurement aux emplacements souhaités.

Je nommerai notre fichier `js store.js` et le mettrai dans le répertoire `/src`.

`src/store.js`

```
1 export const store = {
2   stories: [
3     {
4       plot: 'My horse is amazing.',
5       writer: 'Mr. Weeb1',
6       upvotes: 28,
7       voted: false
8     },
9     {
10       plot: 'Narwhals invented Shish Kebab.',
11       writer: 'Mr. Weeb1',
12       upvotes: 8,
13       voted: false
14     },
15     {
16       plot: 'The dark side of the Force is stronger.',
```

```

17     writer: 'Darth Vader',
18     upvotes: 52,
19     voted: false
20   },
21   {
22     plot: 'One does not simply walk into Mordor',
23     writer: 'Boromir',
24     upvotes: 74,
25     voted: false
26   }
27 ]
28 }
```



Avertissement

La propriété `stories` doit être supprimée de tous les fichiers, car nous avons changé la manière de stockage des données et il peut y avoir des conflits qui peuvent casser notre build.

Après avoir stocké toutes les données dans `store.js`, nous pouvons les importer dans `Stories.vue` en utilisant la syntaxe des modules ES6.

`src/components/Stories.vue`

```

1 <script>
2   import {store} from '../store.js'
3
4   export default {
5     data () {
6       return {
7         // cela nous donne accès à store.stories
8         store
9       }
10    },
11    created () {
12      console.log('stories')
13    }
14  }
15 </script>
```

Comme nous importons l'objet `store`, nous devons également modifier le template du composant.

src/components/Stories.vue

```
1 <template>
2   <ul class="list-group">
3     <li v-for="story in store.stories" class="list-group-item">
4       {{ story.writer }} said "{{ story.plot }}"
5       Story upvotes {{ story.upvotes }}.
6     </li>
7   </ul>
8 </template>
```

Nous utilisons **v-for** pour afficher le rendu des éléments du tableau (`store.stories`). Notre liste d'histoires est affichée comme avant.

Nous pourrions faire la même chose sans devoir changer le template, en bindant directement l'attribut `stories` du composant `store.stories`.

src/components/Stories.vue

```
1 <script>
2   data () {
3     return {
4       // Binding direct aux stories
5       stories: store.stories,
6     }
7   }
8 </script>
```

La même chose s'applique pour `Famous.vue`.

src/components/Famous.vue

```
1 <script>
2   import {store} from '../store.js'
3
4   export default {
5     data () {
6       return {
7         stories: store.stories
8       }
9     },
10    computed: {
11      famous () {
```

```
12     return this.stories.filter(function (item) {
13         return item.upvotes > 50
14     })
15 }
16 }
17 }
18 </script>
```

Si nous n'avions pas bindé nos stories, la propriété calculée `famous()` devrait être mise à jour pour filtrer `this.store.stories`.

Une fois que vous vous habituez à travailler avec des objets globaux, je crois que vous allez **les aimer!** :).



Exemple de code

Vous trouverez les exemples de code de ce chapitre sur [GitHub](#)⁹⁵.

⁹⁵<https://github.com/hoottlex/the-majesty-of-vuejs-2/tree/master/codes/chapter17>

Permutation entre composants

L'utilisation des composants à fichier unique est le moyen le plus simple de créer une application avec Vue. Nous avons vu jusqu'ici comment installer un nouveau projet, créer des fichiers `.vue` et gérer l'état dupliqué. Il est maintenant temps d'examiner une façon d'échanger des composants.

Pour référence, dans les exemples précédents, nous avions 3 composants à l'intérieur de `App.vue` et d'autres imbriqués dedans. Nous devons trouver un moyen d'échanger des composants dynamiquement, de sorte qu'ils ne soient pas rendus sur la page simultanément.

Composants Dynamiques

L'attribut spécial `is`

Nous pouvons utiliser l'élément `<component>` réservé et utiliser le même point de montage pour basculer dynamiquement entre plusieurs composants, en utilisant l'attribut `is`.

src/App.vue

```
<template>
  <div id="app">
    <component is="hello"></component>
    <p>
      This is very useful...
    </p>
  </div>
</template>

<script>
import Hello from './components/Hello'
// Le composant Hello renvoie un template contenant une propriété de données nommée "msg"
export default {
  components: {
    Hello
  }
}
</script>
```

Nous avons créé un nouveau projet et modifié le fichier `Hello.vue`. Nous avons exactement le même résultat que précédemment, mais ici nous utilisons l'élément `<component is = "hello">`. Le composant Hello est lié à l'attribut `is`. Pour voir comment cela fonctionne de façon dynamique, vérifiez l'exemple suivant où nous changeons de 2 composantes différentes en cliquant sur leurs liens.

Tout d'abord, créez un composant similaire avec un message différent, nommé * `Greet.vue` *.

src/Greet.vue

```
<template>
  <div class="greet">
    <h1>{{ msg }}</h1>
  </div>
</template>

<script>
export default {
  data () {
    return {
      msg: 'No! I want to use the <component> element!'
    }
  }
}
</script>
```

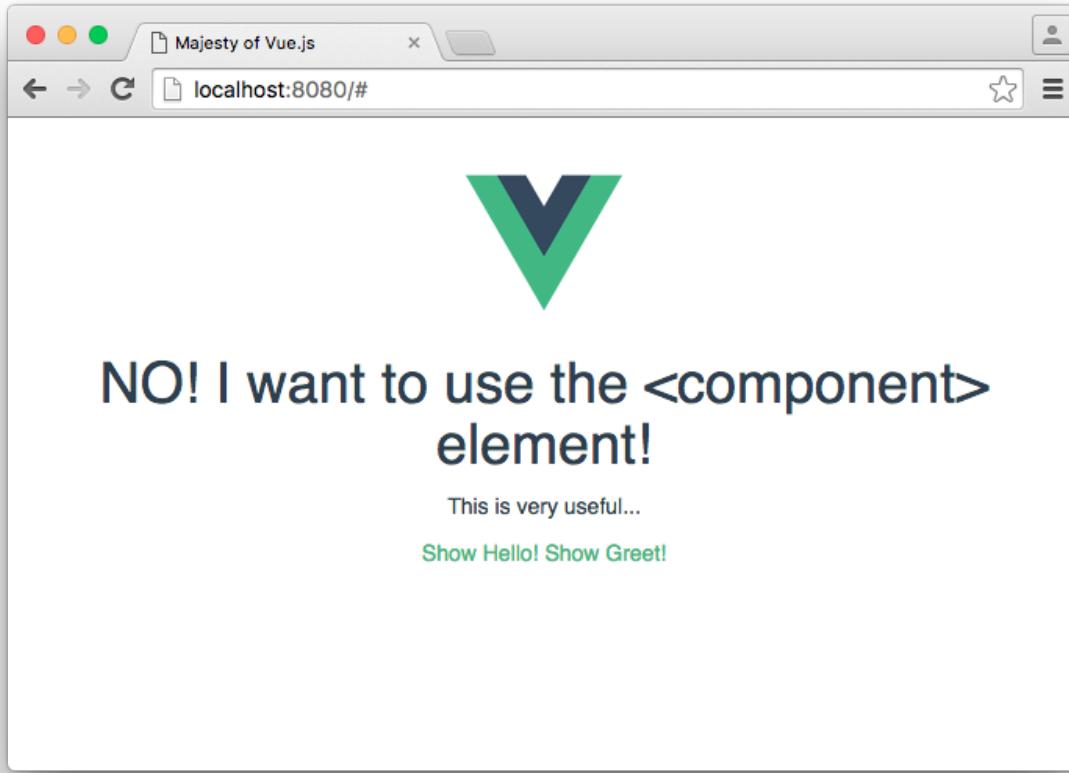
Nous avons créer ‘Greet’ pour afficher un message juste pour manifester sa présence ! Importons-le dans App et donnons à l’utilisateur la possibilité d’échanger entre les 2 composants.

src/App.vue

```
<template>
  <div id="app">
    
    <component :is="currentComponent">
      <!-- le composant change quand this.currentComponent change! -->
    </component>
    <p>
      This is very useful...
    </p>
    <a href="#" @click="currentComponent = 'hello'">Show Hello</a>
    <a href="#" @click="currentComponent = 'greet'">Show Greet</a>
  </div>
</template>
```

```
<script>
import Hello from './components/Hello'
import Greet from './components/Greet'

export default {
  components: {
    Hello,
    Greet
  },
  data () {
    return {
      currentComponent: 'hello'
    }
  }
}
</script>
```



Greet.vue

Bien, comme vous pouvez le voir, nous lierons l'attribut spécial `is` à `currentComponent`, du coup, quand sa valeur change, le composant d'affichage changera également. Pour échanger la vue, il suffit à l'utilisateur de cliquer sur un des deux liens pour modifier la valeur de `currentComponent`.

Ce mode dynamique de commutation entre plusieurs composants peut s'avérer pratique.

Navigation

Dans les [exemples précédents](#), nous avons utilisé des fichiers `.vue` pour simuler un réseau social, où nous avions des composants comme `Login`, `Registration`, etc. Nous pouvons maintenant utiliser un système d'onglets Pour naviguer entre ces composants avec style.

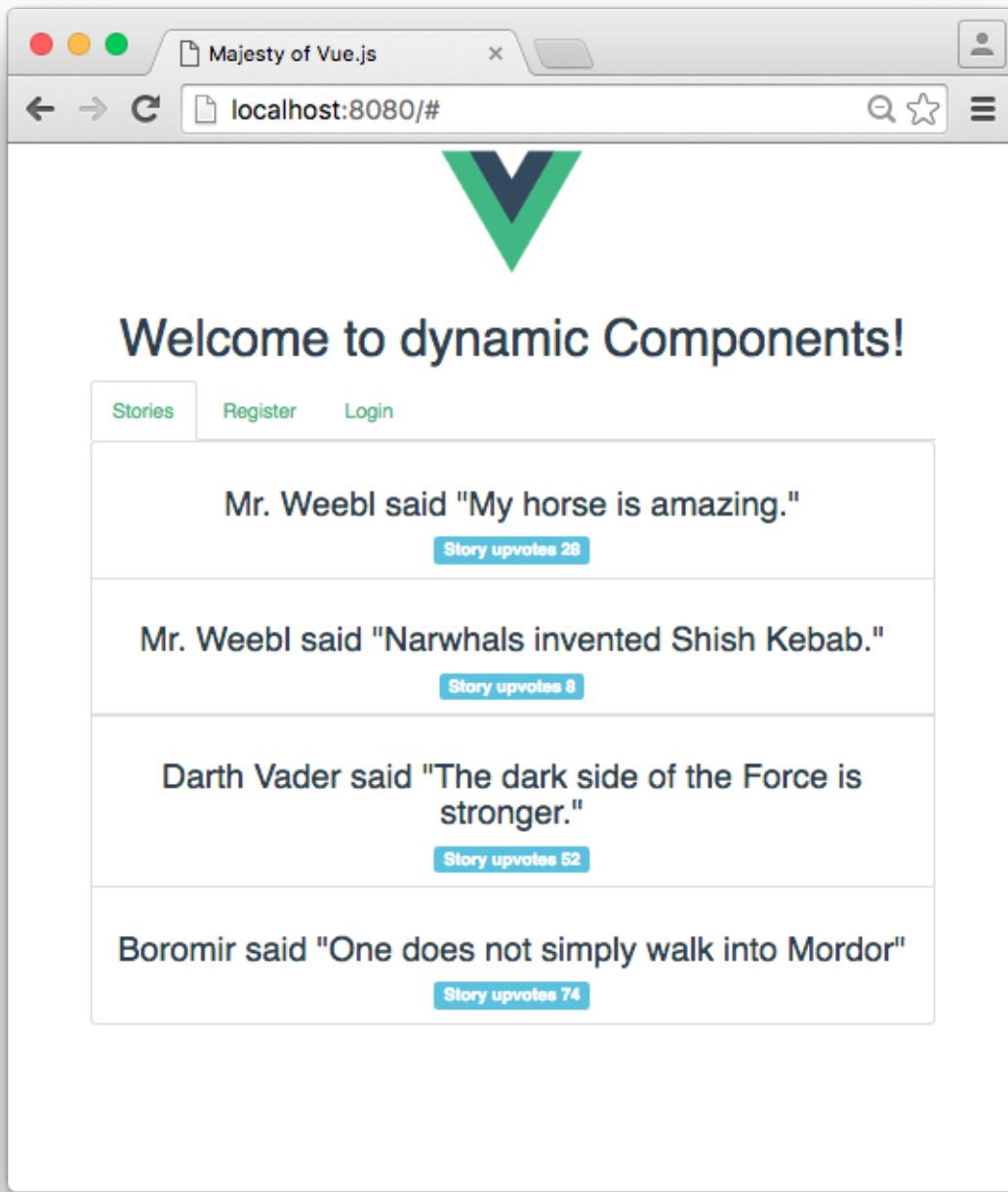
Nous allons avoir `Stories.vue` dans un onglet, `Register.vue` dans un autre, et `Login.vue` dans un troisième. N'oubliez pas que `Register` contient le composant `Famous`, qui renvoie les histoires les plus tendances.

Lisez attentivement l'exemple suivant.

src/App.vue

```
1 <template>
2   <div id="app">
3     
4     <h1>Welcome to dynamic Components!</h1>
5     <ul class="nav nav-tabs">
6       <!-- Définition de la classe 'active' conditionnellement -->
7       <li v-for="page in pages" :class="isActivePage(page) ? 'active' : ''">
8         <!-- Utilisation des liens pour passer d'un onglet à un autre -->
9         <a @click="setPage(page)">{{page | capitalize}}</a>
10        </li>
11      </ul>
12      <component :is="activePage"></component>
13    </div>
14  </template>
15
16  <script>
17    import Vue from 'vue'
18    import Login from './components/Login.vue'
19    import Register from './components/Register.vue'
20    import Stories from './components/Stories.vue'
21
22    Vue.filter('capitalize', function (value) {
23      return value.charAt(0).toUpperCase() + value.substr(1)
24    })
25
26    export default {
27      components: {
28        Login,
29        Register,
30        Stories
31      },
32      data () {
33        return {
34          // Les pages que nous voulons afficher chaque fois
35          pages: [
36            'stories',
37            'register',
38            'login'
39          ],
40          activePage: 'stories'
41        }
42      }
43    }
44  </script>
```

```
42  },
43  methods: {
44    setPage (newPage) {
45      this.activePage = newPage
46    },
47    isActivePage (page) {
48      return this.activePage === page
49    }
50  }
51 }
52
53 </script>
```



Une page pour chaque composant

Décomposez ce que nous venons de faire.

Le tableau nommé `pages` contient les composants dont nous aimerais afficher le rendu. Nous utilisons la directive `v-for` pour créer un onglet pour chacun d'eux.

Pour naviguer entre les onglets, nous avons créé une méthode appelée `setPage`.

La propriété `activePage` est initialement définie à '`stories`'. Lorsqu'un onglet est cliqué, `activePage` change afin de refléter le nom du composant que nous souhaitons afficher.

Pour déterminer quel onglet doit être actif, une ligne `if` est appliquée, ce qui définit la classe `active` si la propriété `activePage` actuelle correspond au nom du composant courant.

Pour afficher la première lettre de chaque onglet en majuscule, nous avons créé un `Vue.filter()`, nommé `capitalize`, qui est utilisé dans les interpolations de texte.

Avec ces quelques lignes de code toutes simples, nous avons créer un système de navigation qui permute entre nos composants.



Exemples de code

Vous trouverez les exemples de code de ce chapitre sur [GitHub⁹⁶](#).

⁹⁶<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter18>