Nicolas Decoster
@nnodot
Informaticien et scientifique chez Magellium
Co-fondateur et animateur à la Compagnie du Code

Juliane Blier
@tactless7
Développeuse JavaScript chez SchoolMouv

Tour d'horizon de Vue

Partie 1

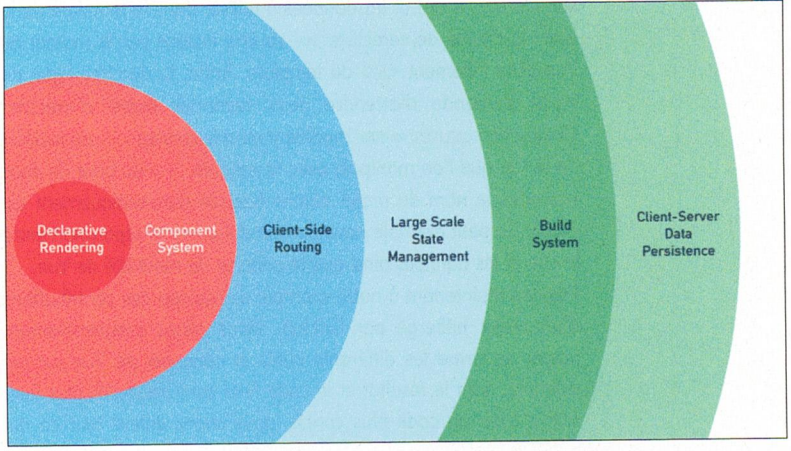
Vue est un framework web créé pour la construction d'interfaces utilisateur. Il a été conçu pour une prise en main facile et une adoption progressive. Ce positionnement et des choix techniques bien pensés en font un framework de qualité, gagnant en popularité et se distinguant des autres frameworks majeurs que sont Angular et React.

niveau 100

Présentation de Vue

A l'origine le langage JavaScript a été créé pour apporter un peu de dynamisme au langage HTML. Avec le temps, la présence de ce langage de programmation côté navigateur a permis de plus en plus d'usages et il est devenu courant d'embarquer de grandes quantités de code JavaScript dans des pages web de plus en plus complexes. Pour gérer cette complexité et faciliter le développement de leur site web, les développeurs ont construit des niveaux d'abstractions supérieurs et différentes bibliothèques ont vu le jour. De nos jours, celles qui ont le vent en poupe sont Angular, React et Vue.
Vue est un framework JavaScript qui facilite le développement d'interfaces utilisateur. Il permet de créer aussi bien des petites pages, que des sites web complexes. Il trouve sa place dans n'importe quel type de projet, pouvant s'insérer progressivement dans du code existant. Une attention particulière a été portée au niveau utilisabilité pour que les fonctionnalités soient au bon niveau d'abstraction, faciles à appréhender et sans superflu. Sa prise en main est directe et la courbe d'apprentissage très douce, et sa documentation est de très bonne qualité, se lisant presque comme un roman. Elle est traduite dans plusieurs langues par la communauté (dont le français). Enfin Vue permet d'écrire du code testable et maintenable, et offre de très bonnes performances.
Le parcours d'Evan You, le créateur de Vue, permet de comprendre l'esprit de cette bibliothèque et de la communauté qui la développe. Bien qu'il se soit très intéressé aux ordinateurs et à l'informatique lors de son enfance, il ne s'engage pas dans cette voie mais préfère suivre un cursus de designer. Au final, ce métier l'a amené dans un deuxième temps au développement web et au langage JavaScript. Employé chez Google, il a utilisé AngularJS dans divers contextes, et, réflexe de designer, il s'est demandé comment rendre son utilisation plus simple. Il a expérimenté des choses en repartant de zéro et en enlevant tout ce qu'il pouvait d'AngularJS pour ne garder que ce qui lui paraissait essentiel. Vue était né, une communauté s'est vite construite. Devant cet engouement, Evan You a décidé de passer à plein temps sur le développement de cette bibliothèque en se finançant uniquement par des dons.
Vue est une bibliothèque basée sur le principe MVVM, pour Model View ViewModel, qui sépare le modèle contenant les données métiers (Model), la Vue (View) et le modèle qui décrit ce que doit représenter la Vue (ModelView) et qui peut être différent du modèle métier. Ce principe est déjà proposé par WPF de Microsoft en se basant sur XAML pour décrire la vue (similaire à HTML) et sur C# pour la logique. La bibliothèque Knockout.js est un précurseur du principe MVVM en JavaScript. Elle a beaucoup de similarités avec Vue, mais le projet est un peu moins actif et Vue est considéré par certains comme plus simple à manipuler. Quoi qu'il en soit, l'avantage principal du principe MVVM est de bien séparer les données de leur représentation, de se reposer sur la bibliothèque pour assurer la mise à jour de la vue lorsque les données changent, et, inversement, de mettre à jour les données lors des actions de l'utilisateur dans la vue. Il se trouve que Vue est très bon à ce jeu : il offre un système de réactivité qui est très performant car il garde une trace du graphe des dépendances entre données et éléments de la vue concernés (pour ceux qui connaissent, il y a des grandes similitudes avec le principe de fonctionnement de MobX).
Enfin, toujours dans cette idée de fournir une expérience développeur de qualité, Vue propose une approche progressive dans les fonctionnalités offertes. Il y a une séparation nette entre les différents concepts ou outils utiles pour le développement d'une interface web. Suivant ses besoins et la taille de son projet, on commence avec le coeur de rendu déclaratif MVVM de Vue, puis on ajoute le système de composants permettant de séparer les différents éléments de l'interface, puis on ajoute le routing (vue-router), puis la gestion de l'état avec Vuex (qui est une sorte de pendant de Redux pour Vue). Quand son projet devient conséquent on utilise les outils de build fournis par Vue. Puis, enfin, on peut ajouter du rendu côté serveur (SSR) si on désire optimiser l'expérience utilisateur et le référencement. Et bien que tous ces éléments soient optionnels et qu'on les ajoute au fur et à mesure, ils sont tous développés de manière coordonnée par la core team, offrant un ensemble bien segmenté restant parfaitement cohérent. 1
L'équipe de développeurs de Vue ne s'arrête pas non plus en si bon chemin. A l'heure où nous écrivons ces lignes, Evan You a annoncé la prochaine arrivée de la version 3.0 du framework lors de la conférence Vue JS de Londres.

1



Le rendu déclaratif avec Vue

Le fonctionnement de Vue est centré sur son moteur de rendu déclaratif MVVM, pour Model View ViewModel. Cela consiste en une séparation nette entre les données et la manière dont elles sont représentées. Dans le principe, la partie JavaScript définit un modèle de vue avec ses données et ses comportements sans faire références aux éléments du DOM et c'est plutôt le rôle de la vue définie en HTML de faire référence aux éléments du modèle de vue. On ne fait que déclarer des données et des comportements d'un côté, et la manière de les représenter de l'autre. On parle donc de rendu déclaratif. Le moteur de Vue se charge de mettre à jour automatiquement la vue lorsque le modèle de vue est modifié.

Voyons ensemble les outils et les mécanismes fournis par Vue pour définir une interface. Imaginons que l'on veuille développer une petite page web montrant les contributeurs d'un projet hébergé par GitHub, en exploitant l'API HTTP publique de récupération des informations associées à un repository.

Avant de rentrer dans les détails avec ce scénario plus poussé, commençons par un code très simple montrant un exemple minimal. Pour changer du bon vieux "Hello World!", nous allons juste afficher "Repository for" suivi du nom d'un projet GitHub ("vue" dans notre cas) :

```
<html>
 <script src="https://unpkg.com/vue"></script>
 <body>
  <div id="app">
   <h1> Repository for {{ name }} </h1>
  </div>
 </body>
 <script>
  const app = new Vue({
   el: '#app',
   data: { name: 'vue'}
  })
  setTimeout(() => app.name = 'vuex', 2000)
 </script>
</html>
```

Nous avons écrit notre première page web avec Vue. Dans la vue, on insère des données à l'aide d'une syntaxe de template basée sur des doubles accolades. Côté JavaScript, on instancie notre application en associant un élément de template (désigné ici par son identifiant app) et les données à utiliser dans le template de la vue : la chaîne de caractère 'vue' va être insérée par le moteur de Vue dans l'élément <h1> du template. Enfin, la dernière ligne du script demande d'attendre deux secondes avant d'exécuter l'instruction app.name = 'vuex' modifiant ainsi le champ name du modèle de la vue que l'on manipule avec la variable app : au bout de deux secondes le nom du projet affiché bascule automatiquement sur "vuex". Ce petit exemple nous permet d'illustrer la séparation entre la vue et les données ainsi que le principe de réactivité de Vue.

Passons maintenant à notre exemple de tableau des contributeurs d'un projet hébergé par GitHub. Nous allons progressivement passer en revue les différents outils et concepts de Vue sur cet exemple, dont le résultat et le code final sont présentés plus loin. Afin d'avoir un code plus concis, nous avons enlevé tout ce qui concerne la mise en forme (structure et style) qui n'est pas nécessaire pour illustrer notre propos.

Dans cette page web, nous exploitons les informations retournées par l'API HTTP fournie par GitHub, et en particulier l'URL vers le projet lui-même et celle vers la liste des contributeurs. La première retourne un objet JSON dont la forme est (on n'a gardé que les informations nécessaires pour notre exemple) :

```
{
 "name": "vue",
 "owner": {
  "avatar_url": "https://avatars1...",
 },
 "description": "⊠ A progressive..."
}
```

et la deuxième renvoie une liste d'objets dont la forme est :

```
[
 {
  "login": "yyx990803",
  "avatar_url": "https://avatars1...",
  "contributions": 2060
 },
 {
  "login": "Hanks10100",
  "avatar_url": "https://avatars0...",
  "contributions": 47
 },
 ...
]
```

Pour l'instant, on considère que ces informations sont respectivement stockées dans les champs project et contributors de la structure data de notre objet Vue (nous verrons plus loin comment les récupérer dynamiquement). Nous pouvons ainsi créer une petite fiche de présentation du projet :

```
<img v-bind:src="project.owner.avatar_url">
<h1> {{ project.name }} </h1>
<h4> {{ project.description }} </h4>
```

La directive v-bind:src indique que l'expression JavaScript à droite du signe égal doit être évaluée et le résultat affecté à l'attribut src de l'élément <img>, c'est-à-dire ici l'URL de l'image de l'avatar de l'utilisateur propriétaire du projet (le logo de Vue dans notre cas). Pour information, Vue définit un ensemble de nouveaux attributs HTML, que l'on appelle directives et dont les noms commencent par "v-".

Pour l'affichage de la liste des contributeurs, on fait appel à une directive v-for qui permet de répéter l'élément pour chaque contributeur :

```
<div v-for="contributor in contributors"> ... <div>
```

Pour chaque répétition de l'élément, une nouvelle variable contributor est créée qui contient les informations associées au contributeur considéré issu de la liste contributors. On exploite d'ailleurs une de ces informations avec la directive v-if pour n'afficher que les utilisateurs ayant apporté au moins 11 contributions au projet :

```
<div v-if="contributor.contributions > 10"...
```

Une autre directive v-if est utilisée plus loin, conjointement avec une directive v-else, pour afficher le nom d'Evan You à la place de son login GitHub et laisser le login pour tous les autres utilisateurs.
Dans les templates de la vue il est possible d'utiliser des expressions JavaScript afin d'afficher des valeurs calculées à partir des données du modèle, comme par exemple le nombre total de contributions. Cependant cela introduit de la logique dans la vue, le code peut vite devenir verbeux et difficile à maintenir. Le mieux reste de laisser cela à la partie JavaScript. Vue offre un mécanisme pour cela par l'intermédiaire des propriétés calculées (*computed* en anglais). La variable total sera mise à jour dès qu'une des valeurs permettant de la calculer sera modifiée. Côté JavaScript il suffit donc d'écrire :

```
computed: {
 total: function () {
  return this.contributors.reduce((sum, c) => {
   return sum + c.contributions
  }, 0)
 }
}
```

Dans le template on peut utiliser total comme n'importe quelle autre propriété du modèle de la vue :

```
<h4> {{ total }} contributions </h4>
```

Vue se charge de surveiller tout changement dans la liste des contributeurs, pour mettre à jour la valeur de total ainsi que les parties de la vue qui l'utilisent.
Vue offre un mécanisme simple pour réagir aux événements des éléments de la vue. Dans notre exemple nous affichons l'image de l'avatar d'un utilisateur lorsqu'on clique sur son login. Pour cela, nous avons d'abord défini une propriété du modèle de vue qui s'appelle selected qui contient le contributeur sélectionné. L'image affichée provient donc tout simplement de l'URL stockée dans cette propriété :

```
<img v-bind:src="selected.avatar_url">
```

Le changement de selected est associé à l'évènement click de souris de l'élément qui affiche le login d'un contributeur

```
<div ... v-on:click="selected = contributor" ...
```

Pour montrer à l'utilisateur quel est le contributeur sur lequel il vient de cliquer, nous affectons une classe CSS selected à l'élément cliqué. Pour changer cette classe de manière conditionnelle on utilise une directive v-bind dont le comportement est spécifique à la liaison de classes :

```
<div ...
 v-bind:class="{
  coredev: contributor.contributions > 30,
  selected: contributor === selected
 }" ...
```

Cette syntaxe indique que la classe selected est présente si le collaborateur est celui qui est sélectionné, et la classe coredev est présente si le nombre de contributions est strictement supérieur à 30. Pour information, Vue permet de manipuler le style d'un élément avec v-bind sur un principe similaire.
Dans cet exemple nous voulons laisser à l'utilisateur la possibilité de changer le projet GitHub pour lequel il veut voir les contributeurs. Le moyen naturel est d'utiliser un élément <input>. Vue nous fournit un outil très pratique qui permet de définir une liaison dans les deux sens (*two way binding* en anglais) entre un élément de saisie (comme ceux trouvés dans les formulaires) et une propriété du modèle de vue. Pour cela on utilise la directive v-model en indiquant le nom de la propriété concernée :

```
<input v-model="repo"></input>
```

Chaque fois que l'utilisateur changera l'élément <input>, la propriété du modèle changera également. Il ne nous reste plus qu'à réagir à ces changements pour télécharger les informations liées au nouveau projet avec l'API de GitHub. Cette surveillance se fait en définissant une fonction qui porte le nom de la propriété à surveiller (repo dans notre cas) et de l'ajouter dans la structure watch du modèle de vue :

```
watch: {
 repo: function (repo, oldRepo) {
  ...
 }, ...
}
```

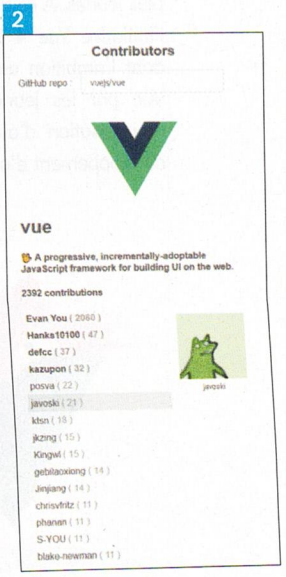
Ce petit exemple de tableau de bord des contributeurs d'un projet GitHub a permis de faire le tour des fonctionnalités principales du rendu déclaratif avec Vue, et de voir le potentiel et l'élégance de ce framework. Pour aller plus loin nous ne saurions trop recommander de se plonger dans la documentation officielle qui est très complète, claire et précise. 2
Résultat du rendu de la page web présentant la liste de contributeurs d'un projet GitHub
Code source du template en HTML :

```
<div id="app">
 <h2>Contributors</h2>

 <span> GitHub repo : </span>
 <input v-model="repo"></input>
 <img v-bind:src="project.owner.avatar_url">
 <h1> {{ project.name }} </h1>
 <h4> {{ project.description }} </h4>
 <h4> {{ total }} contributions </h4>

 <div v-for="contributor in contributors">
  <div
   v-if="contributor.contributions > 10"
   v-bind:class="{
    coredev: contributor.contributions > 30,
    selected: contributor === selected
   }"
   v-on:click="selected = contributor"
  >
   <span v-if="contributor.login === 'yyx990803'">
    Evan You
   </span>
   <span v-else>
```

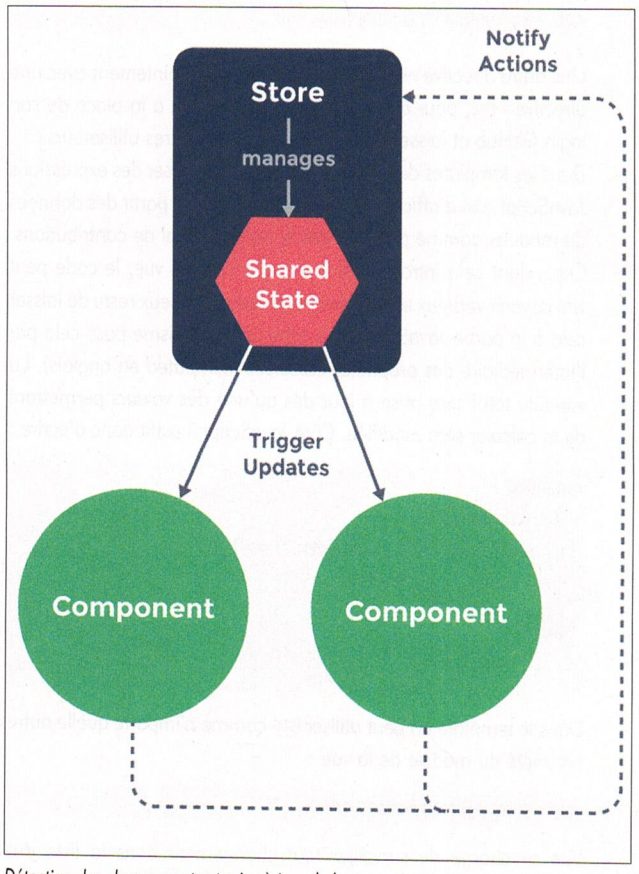
2



```
    {{ contributor.login }}
   </span>
   <span>
    ( {{ contributor.contributions }} )
   </span>
  </div>
 </div>
 <img v-bind:src="selected.avatar_url">
 {{ selected.login }}
</div>
```

Code source JavaScript :

```
const reposApi = 'https://api.github.com/repos'
const app = new Vue({
 el: '#app',
 data: () => ({
  repo: '',
  project: {},
  contributors: [],
  selected: {}
 }),
 computed: {
  total: function () {
   return this.contributors.reduce((sum, c) => {
    return sum + c.contributions
   }, 0)
  }
 },
 watch: {
  repo: function (repo, oldRepo) {
   fetch(`${reposApi}/${repo}`)
    .then(response => response.json())
    .then(project => this.project = project)
   fetch(`${reposApi}/${repo}/contributors`)
    .then(response => response.json())
    .then(list => this.contributors = list)
  },
  contributors: function () {
   this.selected = this.contributors[0]
  }
 }
})
app.repo = 'vuejs/vue'
```



Détection des changements et mise à jour de la vue

Code source de la feuille de style CSS :

```
.coredev {
 font-weight: bolder;
}
.selected {
 background: #eee;
}
```

La suite dans le n°224

Vue ne se limite pas à un outil de développement de pages ou de sites web. Il peut également très bien être utilisé pour créer des jeux, des animations, des quizz, etc. Il est donc un très bon outil pour des activités avec les plus jeunes. À ce titre, signalons la naissance de l'initiative Vue 4 kids (@vue4kids sur Twitter) dont l'ambition est de favoriser l'utilisation de Vue par les jeunes pour des activités d'informatique créative, que ce soit par l'organisation d'ateliers, la mise à disposition de ressources et d'exemples ou le développement d'outils.

Vue 4 kids

Vue Vixens est une initiative lancée par Jen Looper afin d'organiser des ateliers à destination des femmes pour la découverte et l'apprentissage de Vue de manière ludique et bienveillante. Que ce soit lors d'une pause déjeuner, en marge d'une conférence ou sur une journée, les participantes pratiquent Vue pour créer une application web ou mobile. En France, le premier atelier a eu lieu lors du VueJS Road Trip Paris le 29 juin. Une dizaine d'ateliers sont prévus d'ici la fin 2018 un peu partout dans le monde.