Entrevue avec Vue.js

Ludovic Ladeu

Thomas Champion

Qui sommes nous ?

Ludovic Ladeu

Développeur Fullstack

#Back #Web #Cloud

Xebia

Thomas Champion

Développeur Fullstack

#Web #CSS #JS

Sommaire

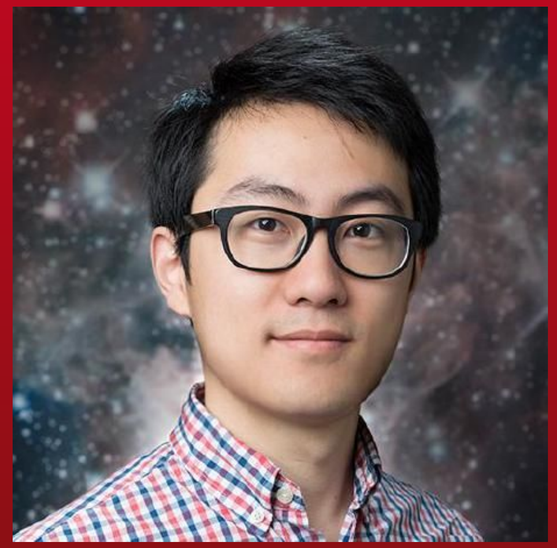
1. Pourquoi Vue ?
2. Architecture d’une application Vue
   a. Les composants
   b. La syntaxe de templating
3. Extensions de Vue
   a. Les plugins
   b. Vue router
4. Hand’s On

Pourquoi Vue ?

Histoire

En février 2014, Evan You publie la première version de Vue.js

"I figured, what if I could just extract the part that I really liked about **Angular** and build something really **lightweight** without all the **extra concepts involved**?"



En trois mots

« Approchable »

En trois mots

« Versatile »

En trois mots

« Performant »

Une application en 30 secondes

Architecture d’une application Vue

Disclaimer

Les exemples de code sont en **EcmaScript 6 (ES2015)**

Déclaration de variable

```
var myVar = 1;
```

```
const myVar = 2;
let myVar = 2;
```

Shorthand method

```
var object = {
 hello: function() {

 }
};
```

```
const object = {
 hello() {

 }
};
```

Shorthand property

```
var name = "bob";

var object = {
 name : name
};
```

```
const name = "bob";

const object = {
 name
};
```

Introduction

Pour bootstrapper une application Vue.js :

- Un élément html
- Une instance de Vue lié à cet élément

index.html

```
<div id="app">
 <h1>{{ title }}</h1>
</div>
```

main.js

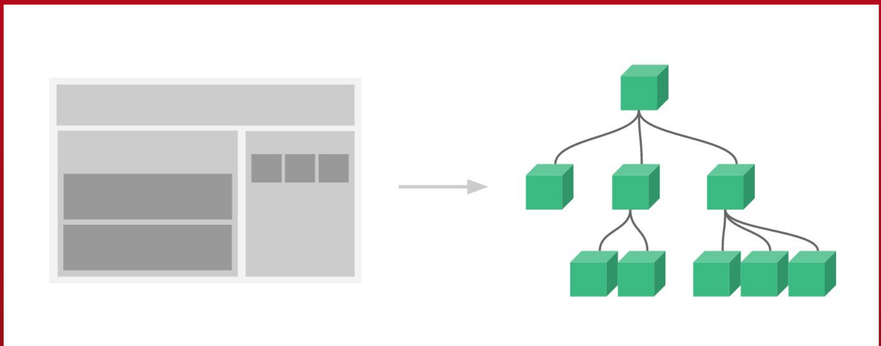
```
new Vue({
   el: '#app',
   data: {
     title: 'My application'
   }
})
```

Et c'est parti ?

Bonne pratique

Pour bootstrapper une application Vue.js :

- Un élément html
- Une **instance** de Vue et un **composant** racine
- Utiliser les composants



index.html

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

main.js

```
new Vue({
  render: h => h(App)
})
.$mount('#app')
```

Les composants

Un composant Vue est défini par un **objet JS** qui se décompose selon la structure suivante :

- **template** : template HTML du composant

```
const MyHeader = {
  template: `
    <h1>
      My application
    </h1>`
};
```

Les composants

Un composant Vue est défini par un objet JS qui se décompose selon la structure suivante :

- **template** : template HTML du composant
- **data** : attributs internes au composant

```
const MyHeader = {
  template: `
    <h1>
      My application at {{ date }}
    </h1>`,
 data() {
    return {
      date: new Date().toString()
    };
 }
};
```

Les composants

Un composant Vue est défini par un objet JS qui se décompose selon la structure suivante :

- **template** : template HTML du composant
- **data** : attributs internes au composant
- **methods** : fonctions définies dans le composant

```
const MyHeader = {
 template: `
   <h1>
     My application at {{ date }} -
     {{ hello() }}
   </h1>`,
 data() {
   return {
     date: new Date().toString()
   };
 },
 methods : {
   hello() {
     return 'hello world';
   }
 };
```

Les composants

Un composant Vue est défini par un objet JS qui se décompose selon la structure suivante :

- **template** : template HTML du composant
- **data** : attributs internes au composant
- **methods** : fonctions définies dans le composant
- **props** : attributs passés par le composant appelant

```
const MyHeader = {
  props: ['name'],
  template: `
    <h1>
      My application at {{ date }} -
      {{ hello() }}
    </h1>`,
  data() {
    return {
      date: new Date().toString()
    };
  },
  methods: {
    hello() {
      return `hello ${this.name}`;
    }
  }
};
```

Les composants

Valoriser une propriété d'un composant :

- Passer une chaîne

```
<my-header name="Bob"></my-header>
```

- Passer un objet

```
<my-header :name="user.name"></my-header>
```

équivalent à

```
<my-header :name="'Bob'"></my-header>
```

```
const MyHeader = {
  props: ['name'],
  template: `
    <h1>
      My application at {{ date }} -
      {{ hello() }}
    </h1>`,
  data() {
    return {
      date: new Date().toString()
    };
  },
  methods: {
    hello() {
      return `hello ${this.name}`;
    }
  }
};
```

Les composants

Un composant Vue est défini par un objet JS qui se décompose selon la structure suivante :

- **template** : template HTML du composant
- **data** : attributs internes au composant
- **methods** : fonctions définies dans le composant
- **props** : attributs passés par le composant appelant
- **components** : référencer les composants qui vont être utilisé

```
const MyTime = {
  template: `<span>{{date}}</span>`,
  data() {
    return {
      date: new Date().toString()
    };
  }
};

const MyHeader = {
  template: `
    <h1>
      My application at <my-time />
    </h1>`,
  components: { MyTime }
};
```

Watch my property

Etre notifié lors la modification d'une propriété

```
const App = {
  template: `
    <div>Your name : <input v-model="firstname" /></div>`,
  data() {
    return {
      firstname: 'bob'
    }
  },
  watch: {
    firstname(value, oldValue) {
      console.log('watch: firstname changed', value, oldValue);
    }
  }
};
```

Computed property

- C’est une **propriété composée** à partir d’autres propriétés.
- Elle se **recalcule** si ses dépendances changent avec **mise en cache**

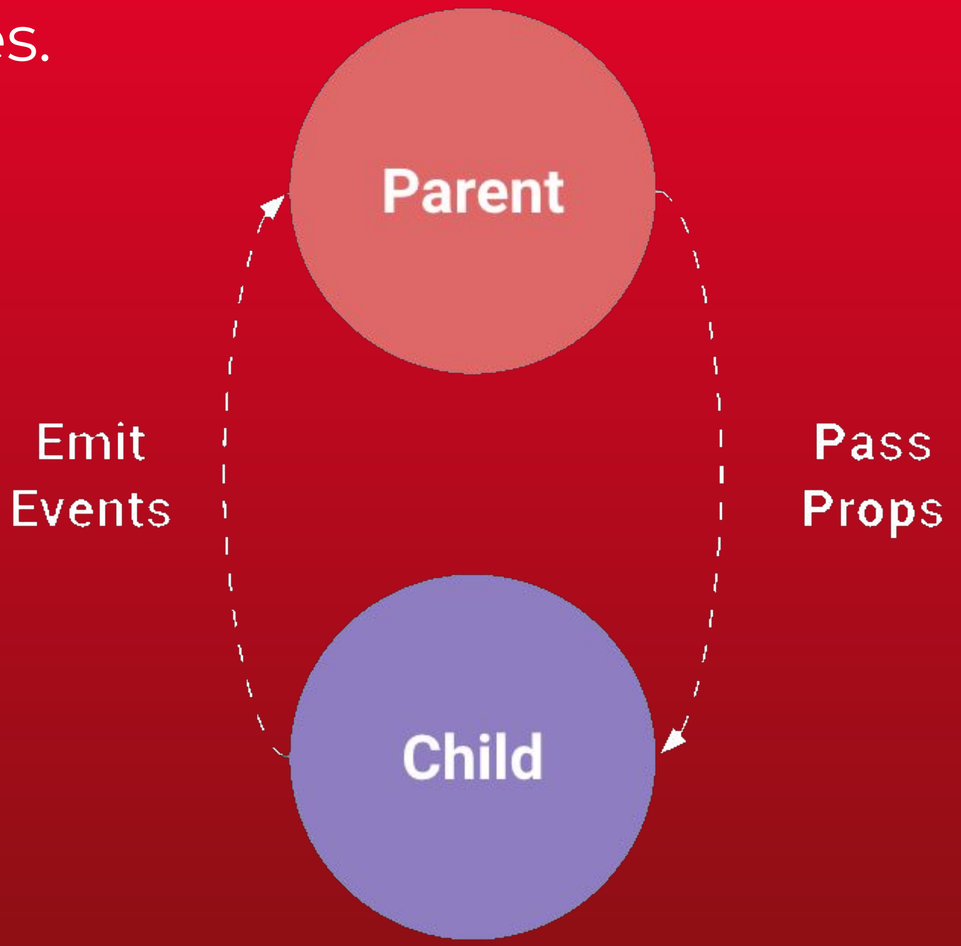
```
const App = {
 template: `<div>{{ computedFullName }} - {{ computedFullName }}</div>`,
 data() {
   return {
     firstname: 'bob',
     lastname: 'dupont'
   }
 },
 computed: {
   computedFullName() {
     return this.firstname + ' ' + this.lastname;
   }
 },
};
```

Démo

Communication composants

Un composant peut posséder des entrées-sorties.

- Les entrées sont les propriétés (props)
- Les sorties correspondent à des événements émisent par le composant enfant



Communication composants

```
const MyText = {
 template: `
  <div>
     <input v-model="content" />
  </div>`,
 props: ['text'],
 data() {
   return {
     content: this.text
   }
 },
 watch: {
   content() {
     this.$emit('textChanged', this.content);
   }
 }
};
```

```
const MyNotebook = {
 template: `
  <div>
      <p>My name : <input v-model="name" /></p>
      <my-text :text="name" @textChanged="nameChanged" />
  </div>`,
 data() {
   return {
     name: 'foobar',
   }
 },
 methods: {
   nameChanged(content) {
     this.name = content;
   }
 },
 components: {
   MyText
 }
};
```

Le cycle de vie

Un composant peut aussi posséder des “hooks” pour être notifié de son cycle de vie :

Création
- beforeCreated
- **created**

Mise à jour du DOM
- beforeUpdate
- updated

Ajout au DOM
- beforeMount
- mounted

Destruction
- beforeDestroy
- destroyed

```
const MyContent = {
 template: `<main>{{ content }}</main>`,
 data() {
   return {
     content: ''
   }
 },
 created() {
   restService.getContent()
     .then(content => {
       this.content = content;
     })
 }
};
```

Vue-cli

Outil qui permet de générer un projet Vuejs :

- Configuré avec webpack via vue-cli-service
- Lint
- Tests
- Prise en charge de ES6+
- Prise en charge des fichiers Vue

```
npm install -g @vue/cli

vue create my-project
```

Les fichiers .Vue

Un fichier vue est composé comme suit :

- template
- script
- style (**scoped** ou non)

```
<template>
 <h1>{{ title }} - {{ date }}</h1>
</template>

<script>
 export default {
   props: ['title'],
   data() {
     return {
       date: new Date().toString()
     };
   },
 };
</script>
<style scoped>
 h1 {
   color: blue;
 }
</style>
```

Les composants globaux

- Par défaut les composants utilisés doivent être importés (via la propriété components)
- Mais il est possible de créer des **composants globaux** :

```
Vue.component('box', {
 template: `<div class="box">{{ content }}</div>`,
 props: ['content']
});
```

En déclarant le composant dans un fichier Vue :

```
import Box from './components/Box.vue'

Vue.component('box', Box);
```

La syntaxe de templating

Les bindings

Comment afficher une propriété dans mon template ?

```
<h1>{{ title }}</h1>
```

Comment lier une propriété de mon composant à un attribut html ?

```
<a href="https://vuejs.org/">link</a>

<a v-bind:href="myLink">link</a>

<a :href="myLink">link</a>
```

Les conditions

```
<div v-if="type === 'A'">
 A
</div>

<div v-else-if="type === 'B'">
 B
</div>

<div v-else>
 Not A/B
</div>
```

Les boucles

```
<ul>
 <li v-for="todo in todos">
   {{ todo.text }}
 </li>
</ul>
```

Boucler sur une liste de composants :
La propriété **key** est obligatoire quand on utilise **v-for** avec les composants.

```
<my-component v-for="item in items"
:key=”item.id”></my-component>
```

Modèle binding

```
<template>
 <div>
   <p>Your name : <input v-model="name" /></p>
   <p>Hello {{ name }}</p>
 </div>
</template>

<script>
 export default {
   data() {
     return {
       name: 'Bob'
     }
   }
 }
</script>
```

Modèle binding : modifiers

Les modifiers servent à **modifier les valeurs** associés à un modèle **avant** de mettre à jour le modèle

```
<input v-model.trim="msg" />

<input v-model.lazy="msg" />

<input v-model.number="age" type="number" />
```

Il est aussi possible de **combiner** les modifiers :

```
<input v-model.trim.lazy="msg" />
```

Les Extensions de Vue

Les plugins

- Les plugins ajoutent des fonctionnalités globales à Vue
- Il n'y a pas de portée strictement définie pour un plugin
- Il existe plusieurs types de plugins :
  - Ajout de méthodes ou propriétés globales : ex. **vue-element**
  - Ajout de directives / filters / transitions : ex. **vue-touch**
  - Ajout d'options aux composants : ex. vuex
  - Ajout de méthodes aux instances de Vue

```
npm install vue-router --save
```

Manage yours routes with Vue Router

1. Déclarer le plugin
2. Composants
3. Définition des routes
4. Instancier le routeur
5. Déclarer le routeur au niveau de l'application

main.js

```
import Vue from 'vue'
import App from './App'
import Router from 'vue-router'

Vue.use(Router)

const Foo = {template: `<div>foo</div>`}
const Bar = {template: `<div>bar</div>`}

const routes = [
   { path: '/foo', component: Foo },
   { path: '/bar', component: Bar },
]
const router = new Router({
  routes
})

new Vue({
  render: h => h(App),
  router
}).$mount('#app')
```

Manage yours routes with Vue Router

6. Ajouter des liens et du conteneur de route

App.vue

```
<template>
 <div id="app">
   <h1>My App</h1>
   <div>
     <router-link to="/foo">Link to foo</router-link>
     <router-link to="/bar">Link to bar</router-link>
   </div>

   <router-view></router-view>
 </div>
</template>
```

Pour aller plus loin : router.vuejs.org

Questions

En attendant vous pouvez récupérer le repository :

https://github.com/xebia-france/devoxx2018-vuejs

node v6 requis (minimum) + npm install

Hand’s On

Sujet du Hand’s On

Nous souhaitons développer le site web **SuperCook** qui permet de partager ses recettes de cuisines. La création de cette application se fera en plusieurs étapes :

- Création d’un composant affichant **une recette**
- Création d’un composant afficher **une liste de recette**
- Création d’un composant de **détail** d’une recette et **routage**
- Permettre l’ajout d’une nouvelle recette
- Pouvoir enregistrer en favoris des recettes

Récupération des sources :
https://github.com/xebia-france/devoxx2018-vuejs

Ressources

Quelques liens

- La documentation officielle : https://vuejs.org/v2/guide
- Cheatsheet : https://vuejs-tips.github.io/cheatsheet
- Bibliothèque de composants : https://vuetifyjs.com
- Ultime page de ressources : https://github.com/vuejs/awesome-vue
- Bonnes pratiques : https://github.com/pablohpsilva/vuejs-component-style-guide

Astuces

Pour utiliser SASS :

```
npm install sass-loader node-sass --save-dev
```

puis au niveau de la balise style ajouter : **lang=”scss”**

```
<style lang="scss">
 #app {
   font-family: 'Avenir', Helvetica, Arial, sans-serif;

   header {
     text-align: center;
   }
 }
</style>
```