

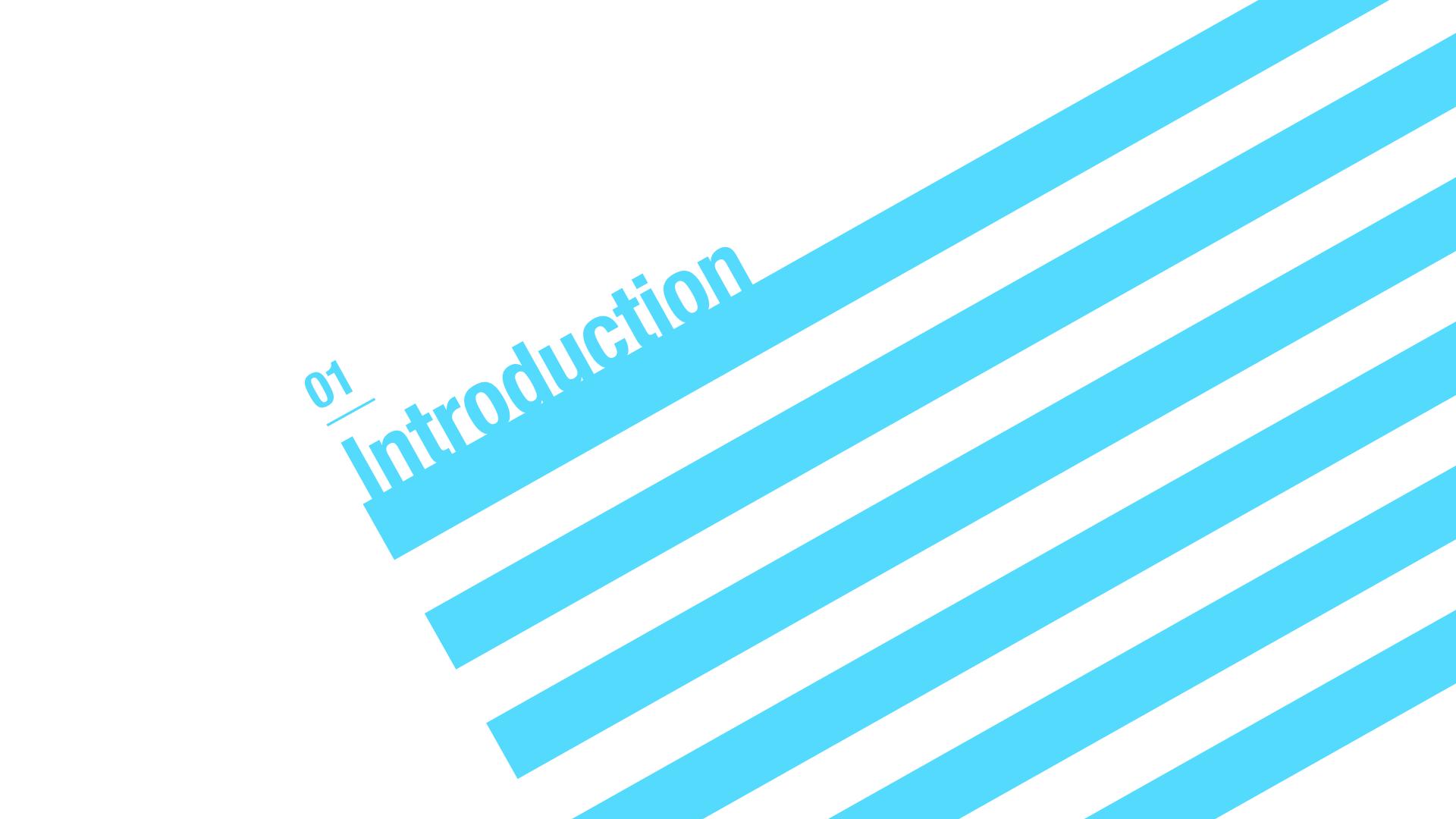
Vue.js



Vue.js

**00**  
Résumé

- 01** Introduction
- 02** ECMAScript 6
- 03** Vue.js
- 04** Outilage & Industrialisation



# 01 / Introduction

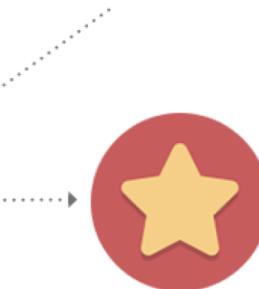
# Le web



Node Package Manager (npm) is released. Node becomes the #1 most watched Github Repository.



Ryan Dahl introduces Node.js, a set of bindings for V8, at JSConf. His presentation is a hit, and developers can't wait to get involved.



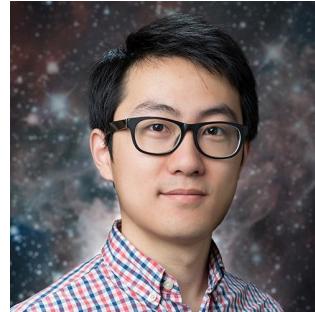
↗ 12,000 modules  
↗ 3,800 active authors

and more....



# Concept de base

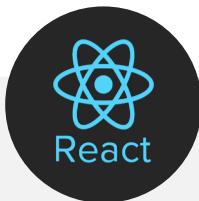
**Vue.js** est un « progressive framework » JavaScript développé par **Evan You** qui se veut accessible, versatile et performant.



## Pourquoi Vue.js ?

« En comparaison de ce que j'ai pu tester avec d'autres frameworks front, j'ai effectivement trouvé que Vue.js est très simple à mettre en place. La création d'une application basique se fait en quelques lignes de code et les résultats sont rapidement là. »

# Les frameworks



- Google - 2009
- Le premier framework !
- MVVM,
- Des contrôleur, directives, services, etc...
- Two-way data-binding,
- Les tests unitaires !



58k



92k

- Un ancien de Google - 2014
- Le conciliateur !
- Centré composant,
- Virtual DOM,
- Performance et réactif,
- One way data-binding,
- Universal JavaScript.



89k

- Google - 2016
- La rupture !
- Orienté :
  - ECMAScript 6 & TypeScript,
  - Web Components,
  - Mobile First.
- La programmation Reactive...
- Angular Universal.



34k

# Vue-cli

- Génération de votre projet,
- Support de Webpack et Browserify,
- Extensible,
- Support de Babel par défaut,
- Support de TypeScript via plugin,
- Support de Jest et Mocha pour les tests unitaires,
- Test E2E.



- Webpack - 2013
- De loin le meilleur !
- Riche et flexible,
- Beaucoup de plugins,
- Configuration difficile à comprendre.



- Browserify - 2011
- Le plus vieux !
- Orienté Node.js plutôt que ES6,
- Configuration simple.



<https://github.com/vuejs/vue-cli/blob/dev/docs/README.md>



# 02 ECMAScript 6 ES2015

# Support EcmaScript

Publié en Juin et renommé par ES2015. Cette spécification apporte les features suivantes :

- Scope (var, let)
- Arrow functions
- Paramètre par défaut
- Agrégation de paramètre
- Template literals (interpolation)
- Syntaxe raccourcie
- Destructuring Assignment
- Module (Export, import)
- Classe, héritage, getter/setter
- Decorator (aka Java annotation)

## ECMAScript 6

A bright new future is coming...

# ES6 - Let

L'instruction **let** est variable. Elle évite le hoisting (remontée). Sa portée est limitée au bloc.

## ES5

```
function getFullName(player) {  
  if (player.isWinner) {  
    var name = 'Winner ' + player.name;  
    return name;  
  }  
  return player.name;  
}  
  
// Or  
  
function getFullName(user) {  
  var name;  
  if (player.isWinner) {  
    name = 'Winner ' + player.name;  
    return name;  
  }  
  // name is still accessible here  
  return player.name;  
}
```

## ES6

```
function getFullName(player) {  
  if (player.isWinner) {  
    let name = 'Winner ' + player.name;  
    return name;  
  }  
  // name is not accessible here  
  return player.name;  
}
```

Remplacera définitivement **var** à long terme !

# ES6 - Const

- Est une constante,
- Portée du bloc comme let,
- Uniquement en get,

```
const NB_PLAYERS_MAX = 6;
NB_PLAYERS_MAX = 7; // SyntaxError
```

Mise à jour des attributs de l'objet ou tableau possible !

```
//Objet
const PLAYER = {};
PLAYER.color = 'blue'; // works
PLAYER = {color: 'blue'}; // SyntaxError
//Array
const PLAYERS = [];
PLAYERS.push({ color: 'blue' }); // works
PLAYERS = []; // SyntaxError
```

# ES6 – Affectations déstructurées

1<sup>er</sup> raccourci

Ce nouveau raccourci permet de créer des objets :

ES5

```
function createPlayer() {  
    let name = 'John';  
    let color = 'blue';  
    return {  
        name: name,  
        color: color  
    }; //ES5  
}
```

ES6

```
function createPlayer() {  
    let name = 'John';  
    let color = 'blue';  
    return { name, color }; //ES6  
}
```

Ici la propriété de l'objet = nom de la variable !

# ES6 – Affectations déstructurées

1<sup>er</sup> raccourci

ES5

2<sup>e</sup> raccourci

```
var httpOptions = { timeout: 2000, isCache: true };
// later
var httpTimeout = httpOptions.timeout;
var httpCache = httpOptions.isCache;
```

ES6

```
let httpOptions = { timeout: 2000, isCache: true };
// later
let { timeout: httpTimeout, isCache: httpCache } = httpOptions;
// OR
let { timeout, isCache } = httpOptions;
// you now have a variable named 'timeout' and
// one named 'isCache' with correct values.
// With an array :
let timeouts = [1000, 2000, 3000];
// later
let [shortTimeout, mediumTimeout] = timeouts;
```

# ES6 – Affectations déstructurées

1<sup>er</sup> raccourci

2<sup>e</sup> raccourci

Conclusion

Dans la vraie vie

```
function randomPlayerOrder() {  
  let player = { name: 'John' };  
  let order = 2;  
  // ...  
  return { player, order };  
}  
  
let { order, player } = randomPlayerOrder();  
  
let { player } = randomPlayerOrder();
```

# ES6 – Paramètres optionnels

## Fonction

ES5

```
function getPlayes(size, page) {
    size = size || 10;
    //si l'opérande de gauche est falsy,
    //c'est-à-dire undefined, 0, false, "",
    page = page || 1;
    // ...
    server.get(size, page);
}
```

ES6

```
function getPlayers(size = 10, page = 1) {
    //...
    server.get(size, page);
}
function getPlayers(size = defaultSize(), page = 1) {}
function getPlayers(size = defaultSize(), page = size - 1) {}
```

# ES6 – Paramètres optionnels

Fonction

Fonctionnent aussi pour les objets :

Objet

```
let { timeout = 1000 } = httpOptions;  
// you now have a variable named 'timeout',  
// with the value of 'httpOptions.timeout' if it exists  
// or 1000 if not
```

# ES6 – Rest operator

Permet d'utiliser la liste des arguments proprement

ES5

```
var playersInGame = [];
function addPlayers(players) {
  for (var i = 0; i < arguments.length; i++) {
    playersInGame.push(arguments[i]);
  }
}

addPlayers('John', 'Rosa');
```

ES6

```
const playersInGame = [];
function addPlayers(...players) {
  for (let player of players) {
    playersInGame.push(player);
  }
  return list;
}
```

**Ne pas confondre avec spread operator (opérateur d'étalement)**

```
let minPrice = Math.min(...[12, 3, 5]);
```

# ES6 – Les classes

## Constructor

```
class Player {  
    constructor(color) {  
        this.color = color;  
    }  
  
    toString() {  
        return `${this.color} player`;  
        // see that? It is another cool feature of ES6,  
        // called template literals  
        // we'll talk about these quickly!  
    }  
}  
  
let bluePlayer = new Player('blue');  
console.log(bluePlayer.toString()); // blue player
```

# ES6 – Les classes

Constructor

Static / get &  
set

```
class Player {  
    static defaultMoney() {  
        return 1000;  
    }  
  
    get color() {  
        console.log('get color');  
        return this._color;  
    }  
  
    set color(newColor) {  
        console.log(`set color ${newColor}`);  
        this._color = newColor;  
    }  
}  
//...  
let playerMoney = Player.defaultMoney(); //Use static  
  
let player = new Player();  
player.color = 'red'; // 'set color red'  
console.log(player.color);
```

# ES6 – Les classes

Constructor

Static / get &

set

Héritage

```
class Case {  
    constructor(name) {  
        this.name = name;  
    }  
  
    doSomethink(){  
        //move, stop...  
    }  
}  
class CaseProperty extends Case {  
    constructor(name, price) {  
        super(name);  
        this.price = price;  
    }  
    doSomethink(){  
        //Override function  
    }  
}  
let case = new CaseProperty('Rue de la paix', 400);  
console.log(case.price); // 400
```

**Héritage prototypal != Héritage de classes**

# ES6 – Les promises

## Principes

### Principes

- Similaire à Angular 1,
- Permet de gérer les appels asynchrones,
- Ne peut être résolue qu'une fois,
- Interception des erreurs,
- Plus lisible que les callbacks.

### Avec les callbacks

```
getUser(login, function (user) {  
    getRights(user, function (rights) {  
        updateMenu(rights);  
    });  
});
```

### Avec les promises

```
getUser(login)  
    .then(function (user) {  
        return getRights(user);  
    })  
    .then(function (rights) {  
        updateMenu(rights);  
    })
```

# ES6 – Les promises

Principes

Méthodes

3 états

- \_ Pending (en cours),
- \_ Fulfilled (réalisée),
- \_ Rejected (rejetée).

Expose les méthodes then et catch :

```
asynchoneFunction().then(siSucces, siRejetée);
asynchoneFunction()
  .catch(siRejetée);
 //= .then(undefined, siRejetée)
```

Exemple de construction d'une nouvelle promise

```
let getUser = function (login) {
  return new Promise(function (resolve, reject) {
    // async stuff, like fetching users from server,
    // returning a response
    if (response.status === 200) {
      resolve(response.data);
    } else {
      reject('No user');
    }
  });
};
```

# ES6 – Arrow function

- Arrow function =>
- Utile pour les callbacks et les fonctions anonymes,
- `return` est implicite s'il n'y a pas de bloc,
- Le `this` reste attaché lexicalement.

## Version longue

```
getUser(login)
  .then(user => {
    console.log(user);
    return getRights(user);
  })
  .then(rights => updateMenu(rights))
```

## Version raccourcie

```
getUser(login)
  .then(user => getRights(user))
  .then(rights => updateMenu(rights))
```

# ES6 - Modules

## API ou convention existante

- CommonJS (NodeJS) avec une syntaxe simple,
- RequireJS (AMD) pour le chargement asynchrone.

## Objectifs ES6

- Création d'une syntaxe conciliant CommonJS/AMD,
- Analyse statique du code,
- Gestion claire des dépendances cycliques.

Dans `game_service.js`

```
const players = [];
export function addPlayer(player) {
  // ...
}
export function start() {
  // ...
}
```

Dans `otherFile.js`

```
import { addPlayer, start } from './game_service';
// later
addPlayer(new Player('John'));
start();
```

**Fondamental dans Angular 2**

# ES6 - Modules

## Import

- \_ Utilisation d'un alias avec **as**,
- \_ Joker \* : importe tout !

```
import * as gameService from './game_service';
//later
gameService.addPlayer(new Player('John'));
gameService.start();
```

## Export

- \_ Export un seul élément (function, valeur ou classe) : **default**.

```
// player.js
export default class Player {
}
// other_file.js
import Player from './player';
```

# EcmaScript, Support et Transpileurs

## Transpileurs

- Convertir du code ES5 vers ES6 ou plus.

## Support ES6

- Navigateurs:  
<http://kangax.github.io/compat-table/es6/>
- Node.js : <http://node.green/>



- Sebastien McKenzie,
- Extensible,
- Beaucoup de configuration,
- Support ES6: 76%,
- Support ES2016+: 58%

## TypeScript

- Microsoft,
- TypeScript,
- Supporte différentes cibles de compilation,
- Support ES6: 59%
- Support ES2016+: 46%

# Conclusion

**Faites du ES6 et un coup de transpileur dans votre processus de build.**



# 03.1 Vue.js les bases

# Bootstrap

- On crée une instance Vue et on le « mount » sur un nœud HTML.
- L'instance Vue accepte des options:
  - « components »,
  - « directives »,
  - « filters »,
  - « plugins », etc...
- Le constructeur de Vue peut être étendu pour créer des constructeurs de composants réutilisables avec des options prédéfinies.

**Une instance Vue est un composant étendu**

```
// main.js

import Vue from "vue";

const MyComponent = Vue.extend({
  name: "TagName"
});

new Vue({
  components: {
    MyComponent
  }
}).$mount("#app");

<!-- page.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>

  <div id="app"></div>
  <script src=".//bundle.js"></script>
</body>
</html>
```

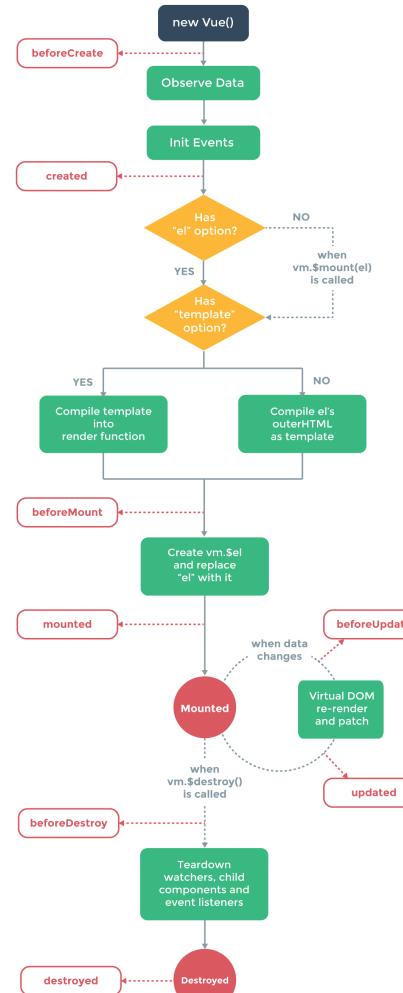
# Les Vues – cycle de vie

— C'est ce qui définit les différentes phases de vie d'un composant vue.

— Les hooks :

- beforeCreate,
- created,
- beforeMount,
- mounted,
- beforeUpdated
- updated,
- beforeDestroy,
- destroyed.

**Tous ces hooks de cycle de vie sont appelés avec leur this pointant sur l'instance de la vue qui les invoque.**



# Les Vues – cycle de vie

\_ Les hooks permettent d'intervenir à un moment du cycle de vie du composant.

\_ Cas d'usage:

- \_ Intervenir lors de la création du composant.
- \_ Faire les appels Ajax quand le composant est « mounted ».
- \_ Appliquer des changements sur un élément du template.

```
new Vue({  
  data: {  
    a: 1  
  },  
  
  created() {  
    console.log("a is :", this.a);  
    // -> a is: 1  
  }  
});
```

# Les Vues – syntaxe des templates

– Syntaxe simple basé sur les mustaches (comme Angular).

– Interpolation des données,

– Le template est compilé en fonction :

- Des propriétés
- Des data,
- Des données « computed »

– Utilisation des directives Vue :

- v-if, v-for, v-once, etc...

```
<template>
  <div>
    <span>{{ 1 + 1 }}</span>

    <h1 v-once>{{ msg }}</h1>

    <span v-once>{{msg}}</span>
  </div>
</template>
<script>
  export default {
    data() {
      return {
        msg: 'Vue is awesome'
      };
    },
    mounted() {
      setTimeout(() => {
        this.msg += '. oh Yeah!!!!';
      }, 2);
    }
  };
</script>
```

# Ready ?



03.2

# Vue.js

## Les directives

# Les directives

— Par convention, toujours préfixé par « v- ».

— Les valeurs attendues pour les attributs sont **des expressions JavaScript**.

**Recommandation:** Faites unique des choses simples !

```
<template>
  <div>

    <span v-text="theText"></span>

    <div v-html="theHtml"></div>

    <div v-if="1 === 1">Yes</div>
    <div v-else>You are in another dimension</div>

    <div v-for="item in [1, 2, 3]" v-bind:key="item">
      item value: {{item}}
    </div>

  </div>
</template>
```

# Les directives prédéfinies

- **v-text** : modifie le contenu texte,
- **v-html** : modifie le contenu html,
- **v-if** : fait apparaître ou disparaître un élément selon une valeur booléenne,
- **v-show**: permet de masquer ou faire apparaître un élément,
- **v-else** : à utiliser avec **v-if** et **v-show**,
- **v-on** : permet de mettre en place un événement,
- **v-bind** : permet de créer une liaison,
- **v-model** : permet de lier la valeur d'un contrôle de formulaire,
- **v-for** : crée un objet vue.js pour chaque élément se trouvant dans un tableau ou un objet,
- **v-transition** : applique une transition,
- **v-ref** : crée une référence d'un composant enfant pour son parent,
- **v-el** : crée une référence d'un élément du DOM.

# Le binding

## Binding d'arguments

- Permet d'affecter des valeurs de façon dynamique à une directive.

## Binding événementiel

- Permet d'écouter les événements à l'instar de JavaScript
- Utilise « v-on:eventname ».
- Possibilité d'appliquer des modificateurs :
  - .prevent,
  - .stop,
  - .once,
  - .capture.

```

<template>
<div>
  Binding argument:
  <div v-bind:class="{red: isRed}"></div>
  <!-- or -->
  <div :class="{red: isRed}"></div>

  Binding event:
  <button v-on:click="showAlert">Click me !</button>
  <!-- or -->
  <button @click="showAlert">Click me !</button>

  With modifier:
  <form @submit.prevent="onSubmit"></form>

</div>
</template>
<script>
  export default {
    data() {
      return {
        isRed: true
      };
    },
    mounted() {
      setTimeout(() => {
        this.isRed = false;
      });
    },
    methods: {
      showAlert() {
        alert('oh yeah!!');
      }
    }
  };
</script>

```

# Les directives custom

\_ Gère une petite partie logique qui ne relève pas d'un composant.

- \_ Application de style à un nœud du DOM,
- \_ Communication entre directives

\_ La directive est instanciée une seule fois au démarrage.

\_ Des paramètres lui seront fournis:

- \_ **el** : l'élément du DOM lié à la directive,
- \_ **binding** : le contexte de la directive dans le template.

```
// demo-directive.js
export const DemoDirective = {
  name: 'demo',
  bind(el, binding) {
    el.innerHTML = JSON.stringify(binding.expression);

    // el is an Element instance
    el.style.color = "red";
  }
}

// component.js
<template>
  <div>
    Test
    <div v-demo="message"></div>
  </div>
</template>
<script>
  import { DemoDirective } from './demo-directive.js';

  export default {
    directives: {
      DemoDirective
    },
    data() {
      return {
        message: 'Hello Vue.js'
      };
    }
  };
</script>
```



03.3

# Vue.js

## les composants

# Les composants

- \_ Orienté Web component,
- \_ Template HTML,
- \_ Logique avec JavaScript,
- \_ Style avec css (ou sass, less, etc...).
- \_ Encapsulation CSS.

\_ Basé sur les noms de tag

\_ Etend les fonctionnalités d'une instance Vue.

**Le tag name doit être différent des tags standard du HTML !**

```
// Counter.vue
<template>
  <div>
    <h1>My awesome counter component</h1>

    <button @click="increaseCounter()">Oh yeah ! ({{counter}})</button>
  </div>
</template>
<script>
  export default {
    name: 'Counter',
    data() {
      return {
        counter: 0
      };
    },
    methods: {
      increaseCounter() {
        this.counter += 1;
        this.$emit('increase', this.counter);
      }
    };
  }
</script>
<style lang="css" scoped>
  h1 {
    color: #2e6da4;
  }

  button {
    color: #1d1d1d;
    background: white;
    border: 1px solid #2e6da4;
  }
</style>
```

// autre part

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

# Les composants - enregistrement

Plusieurs façon:

- \_ Soit globalement avec `Vue.component()`,
- \_ Soit sur l'instance `Vue` de notre application,
- \_ Soit au niveau d'un composant (dépendance de composants parent – enfant).

**Tout dépend de l'utilité de votre composant.**

```
// main.js
import Vue from "vue";
import Counter from "./counter.vue";

// Global level
Vue.component(Counter);

// Or application level
new Vue({
  components: {
    Counter
  }
});

// Other.vue
<template>
  <div>
    <counter></counter>
  </div>
</template>
<script>
  import Counter from './Counter.vue';

  export default {
    name: 'Other',
    components: {
      Counter
    }
  };
</script>
```

# Communication entre composants

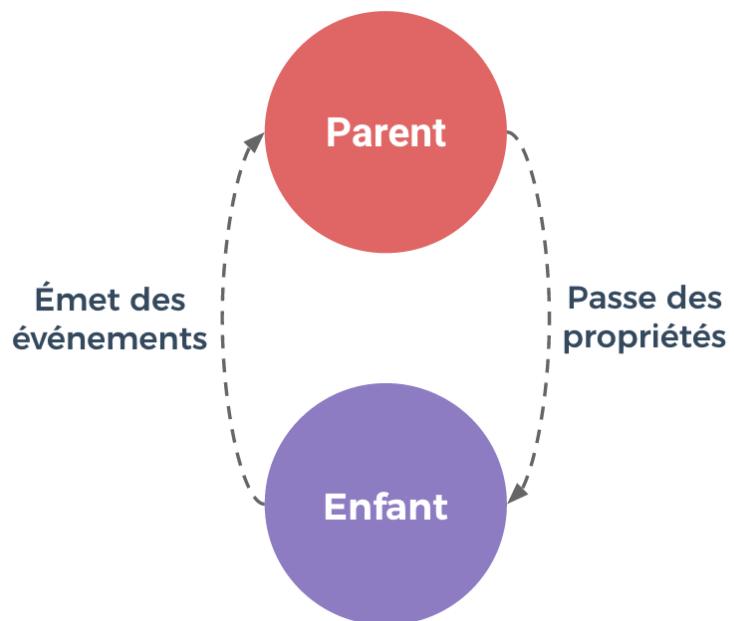
## Au niveau parent (template)

- Même principe que pour les directives,
  - « v-on » ou « @ » pour les événements,
  - « v-bind » ou « : » pour l'affectation des propriétés

## Au niveau enfant (composant)

- Possibilité de définir le type des propriétés,
- Les événements sont déclenchés avec « this.\$emit ».

**Ce principe permet d'isoler le parent de l'enfant via un « contrat de service » !**



# Communication entre composants

```
// child.vue
<script>
  export default {
    name: 'Child',
    props: ['myMessage'],
    // or
    props: {
      myMessage: {
        type: String,
        required: true
      }
    },
    mounted() {
      this.$emit('ready', this.myMessage + ' yeah!');
    }
  };
</script>
```

```
// Parent
<template>
  <div>
    Parent

    <child :my-message="'Oh'" @ready="onReady"></child>
  </div>
</template>
<script>
  import Child from './child.vue';

  export default {
    name: 'Parent',

    components: {
      Child
    },
    methods: {
      onReady(msg) {
        alert(msg);
      }
    }
  };
</script>
```

# Les composants - slots

- Permet de récupérer le contenu du nœud HTML sur lequel le composant est monté,

## Trois types de slot:

- Slot unique,
- Slots nommés. Permet de définir des régions de distribution de contenu au sein du composant,
- Slot avec porté.

```
// Child.vue
<template>
<div>
  <h2>Child</h2>
  <slot>
    This content won't be displayed. oh no!
  </slot>
</div>
</template>

// Parent.vue
<template>
<div>
  <h1>Parent</h1>
  <child>
    <p>This is the original content</p>
    <p>oh yeah!</p>
  </child>
</div>
</template>

<!-- result -->
<div>
  <h1>Parent</h1>
  <div>
    <h2>Child</h2>
    <p>This is the original content</p>
    <p>oh yeah!</p>
  </div>
</div>
```

# Les composants - slots

- Permet de récupérer le contenu du nœud HTML sur lequel le composant est monté,

## Trois types de slot:

- Slot unique,
- Slots nommés. Permet de définir des régions de distribution de contenu au sein du composant,
- Slot avec porté.

```
// Child.vue
<template>
<div>
  <h2>Child</h2>
  <header>
    <slot name="header"></slot>
  </header>
  <main>
    <slot></slot>
  </main>
  <footer>
    <slot name="footer"></slot>
  </footer>
</div>
</template>

// Parent.vue
<template>
<div>
  <h1>Parent</h1>
  <child>
    <h1 slot="header">My awesome header</h1>
    <p>Oh yeah!</p>
    <p>Oh yeah! x2</p>
    <p slot="footer">The footer</p>
  </child>
</div>
</template>

<!-- result --&gt;
&lt;div&gt;
  &lt;h1&gt;Parent&lt;/h1&gt;
  &lt;div&gt;
    &lt;h2&gt;Child&lt;/h2&gt;
    &lt;header&gt;
      &lt;h1&gt;My awesome header&lt;/h1&gt;
    &lt;/header&gt;
    &lt;main&gt;
      &lt;p&gt;Oh yeah!&lt;/p&gt;
      &lt;p&gt;Oh yeah! x2&lt;/p&gt;
    &lt;/main&gt;
    &lt;footer&gt;
      &lt;p&gt;The footer&lt;/p&gt;
    &lt;/footer&gt;
  &lt;/div&gt;
&lt;/div&gt;</pre>

```

# Les composants - slots

- Permet de récupérer le contenu du nœud HTML sur lequel le composant est monté,

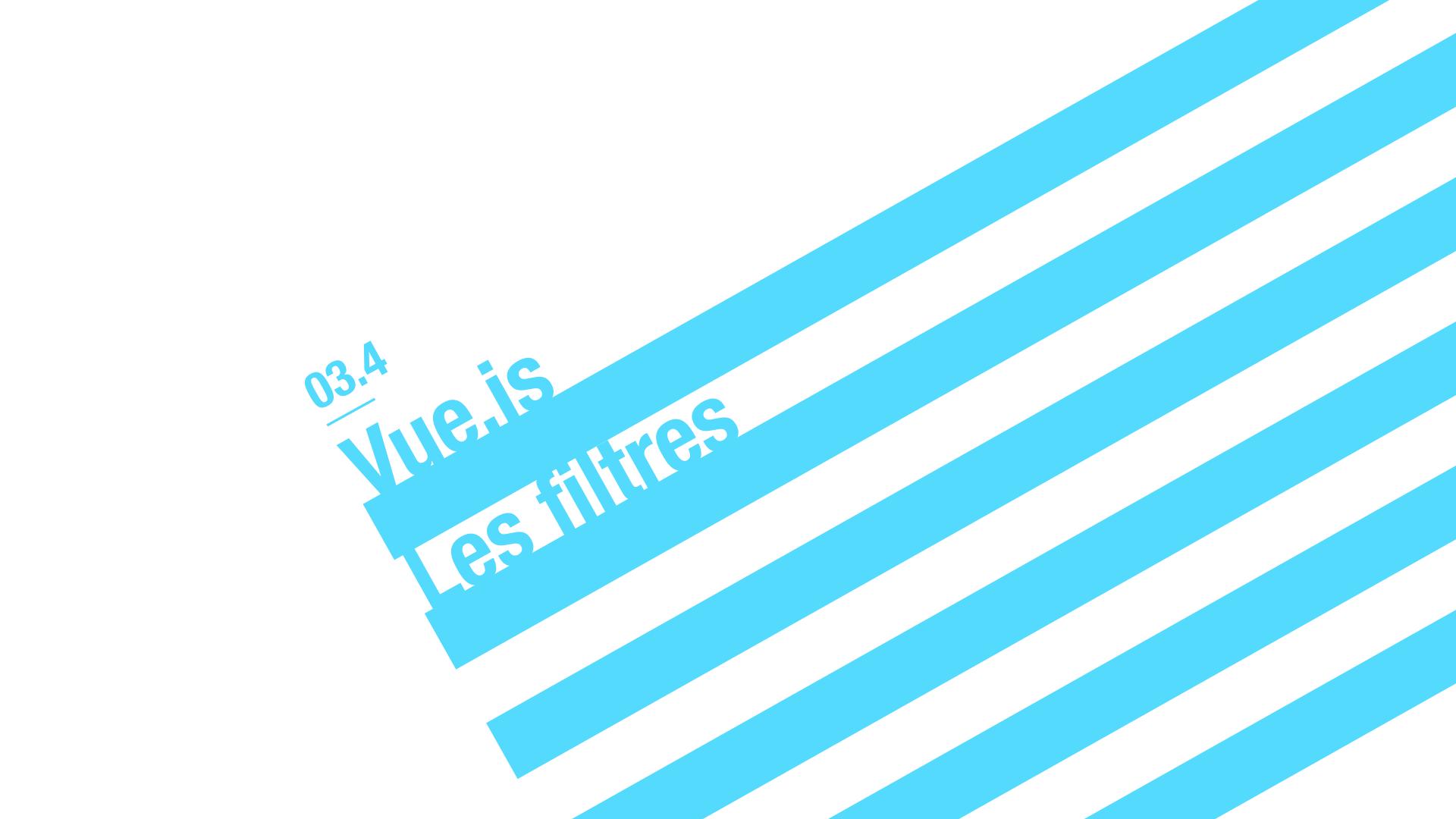
## Trois types de slot:

- Slot unique,
- Slots nommés. Permet de définir des régions de distribution de contenu au sein du composant,
- Slot avec porté.

```
// Child.vue
<template>
  <div>
    Child
    <slot text="Hello from child"></slot>
  </div>
</template>

// Parent.vue
<template>
  <div>
    <h1>Parent</h1>
    <child>
      <template scope="props">
        <span>Hello from parent</span>
        <span>{{props.text}}. Oh yeah!</span>
      </template>
    </child>
  </div>
</template>

<!-- result -->
<div>
  <h1>Parent</h1>
  <div>
    <span>Hello from parent</span>
    Child:
    <span>Hello from child. Oh yeah!</span>
  </div>
</div>
```



03.4

# Vue.js les filtres

# Les filtres

- \_ Fonction de formatage utilisable dans les templates et dans notre code JavaScript,
- \_ Syntaxe similaire au commande linux « **expression | myFilter** »
- \_ Prend une expression et accepte des paramètres,
- \_ Peuvent être chainé.
- \_ Peuvent être utilisé localement à un composant (comme les directives).

```
<template>
  <div>

    <p>{{ message | capitalize }}</p>

    <div :id="rawId | formatId"></div>
  </div>
</template>

import Vue from "vue";

Vue.filter('capitalize', function (value) {
  return value.toUpperCase();
});

Vue.filter('reverse', function (value) {
  return value.split(' ').reverse().join(' ');
});
```



# 03.5 Vue.js Les formulaires

# Les formulaires – v-model

- Implémente le two-way data binding pour les champs d'un formulaire.
  - \_ Input (text, password, checkbox, radio, etc...)
  - \_ Select,
  - \_ Textarea
- Ne prend en compte la valeur initiale des attributs « value », « checked » and « selected ».
- Accepte les modificateurs « lazy », « number », etc...

```
<template>
<div>

    Input:
    <input type="text" v-model="message">
    <p>Value : {{message}}</p>

    Checkbox:
    <input type="checkbox" v-model="checked">
    <p>Value : {{checked}}</p>

    Multiple checkbox:
    <input type="checkbox" value="Jack" v-model="checkedNames">
    <input type="checkbox" value="John" v-model="checkedNames">
    <input type="checkbox" value="Mide" v-model="checkedNames">
    <p>Value : {{checkedNames}}</p>

    Lazy:
    <input type="text" v-model.lazy="lazyMsg">
    <p>Value : {{lazyMsg}}</p>

    Number:
    <input type="number" v-model.number="numberValue">
    <p>Value : {{numberValue}}</p>

</div>
</template>
```



## 06 / Outillages

# Les tests

## Les tests unitaires

Framework : Mocha, Chai, Jasmine, etc...

Couverture de test : Coveralls, Istanbul, etc...

## Les tests des webservice

SuperAgent permet de mocker les données à envoyer à un service afin de tester la réponse.

## Les tests end to end

Protractor est un wrapper Selenium. Il se manipule avec JavaScript et Jasmine.

Attention : Nécessite d'avoir le serveur d'application lancé !

```
describe('TU controllers', function() {
  describe('UsersCtrl', function(){
    var scope, ctrl, $httpBackend;

    beforeEach(inject(function(_$httpBackend_, $rootScope, $controller) {
      $httpBackend = _$httpBackend_;
      $httpBackend.expectGET('/users').
        respond([{name:"titi"},{name:"toto"},{name:"tata"}]);
      scope = $rootScope.$new();
      ctrl = $controller('UserCtrl', {$scope: scope});
    }));

    it('should display 3 users in the users list', function() {
      angular.equals(expect(scope.users), []);
      $httpBackend.flush();
      expect(scope.projects.length).toBe(3);
      expect(scope.projects).toEqualData([
        {name:"titi"}, {name:"toto"}, {name:"tata"}]);
    });
  });
});
```

# Automatisation des tâches avec Grunt / Gulp

## Objectifs

Définir des tâches pour automatiser des processus

- \_ Construction de l'application,
- \_ Création de serveur web de dev + Livereload
- \_ Optimisation des « assets »,
- \_ Pré-processing & Transpiler (Sass, TypeScript, Babel),
- \_ Exécution des tests,
- \_ Contrôle de qualité de code,
- \_ Couverture de test.

Exécuter les tâches en ligne de commande

- \_ Utilisable avec Maven, Jenkins, Travis, etc...



- \_ Grunt - 2012
- \_ Le premier mais obsolète !
- \_ Configuration lourde,
- \_ Basé sur les fichiers,
- \_ Lent,
- \_ Beaucoup de plugins.



- \_ Gulp - 2013
- \_ Le meilleur !
- \_ Configuration simple,
- \_ Basé sur le streaming,
- \_ Très rapide,
- \_ Beaucoup de plugins.



# Les bundlers



- Webpack - 2013
- De loin le meilleur !
- Riche et flexible,
- Beaucoup de plugins,
- Configuration difficile à comprendre.



39k



- Fusebox - 2016
- L'inspiré !
- Combine webpack, JSPM et SystemJS,
- Zero configuration ?



3,2k



- Browserify - 2011
- Le plus vieux !
- Orienté Node.js plutôt que ES6,
- Configuration simple.



11,8k

## Les autres

- RequireJS,
- JSPM avec System.js,
- Rollup,
- Brunch.

# Yeoman

## Pourquoi ?

- \_ Construire la base de votre application,
- \_ Fournir des commandes avancées,
- \_ Gérer vos dépendances clients,
- \_ Automatiser les tâches.

Yeoman repose sur trois outils (historiquement) :

- \_ Yo, Bower et Grunt.

Aujourd'hui tout dépend du générateur !



## Un générateur pour tout

- \_ Angular.js,
- \_ React,
- \_ Jhipster, projet Maven + Angular
- \_ Express,
- \_ Ionic.

# Les outils de développement

## \_ Node.js

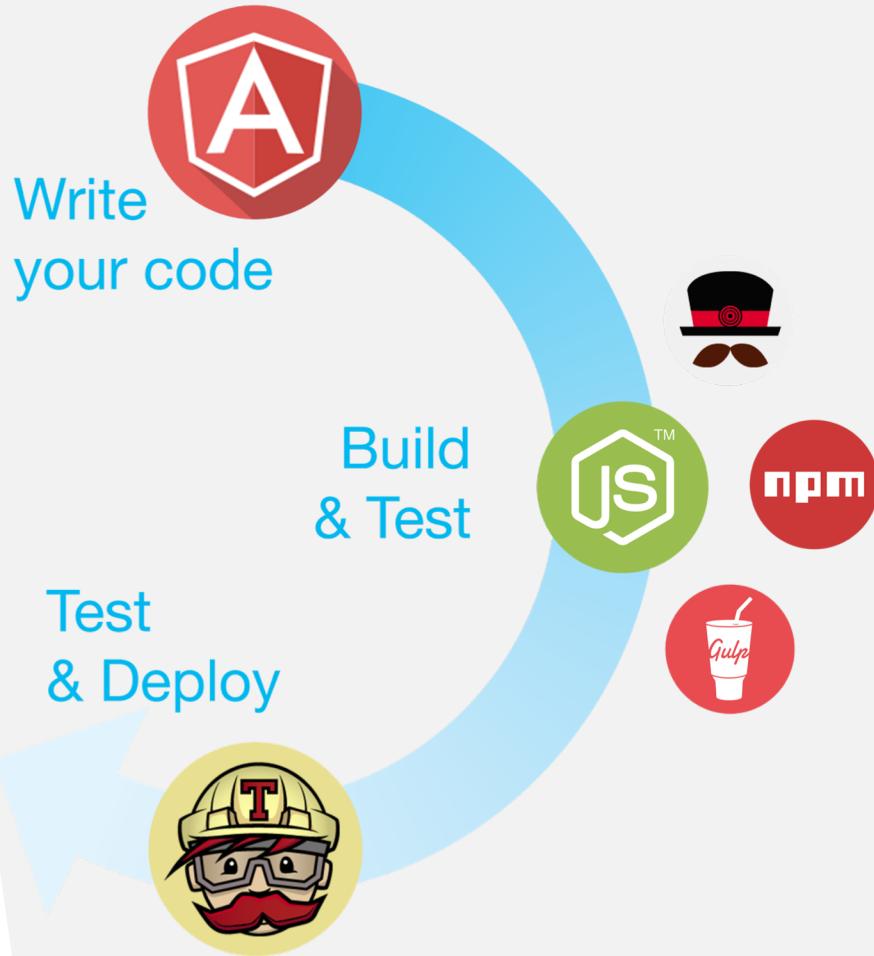
- \_ Yeoman, NPM, Yarn,
- \_ Browserify / Webpack, Grunt / Gulp

## \_ IDE

- \_ IntelliJ / Webstorm,
- \_ SublimeText,
- \_ Atom,
- \_ Visual Code,
- \_ Eclipse + plugin AngularJS.

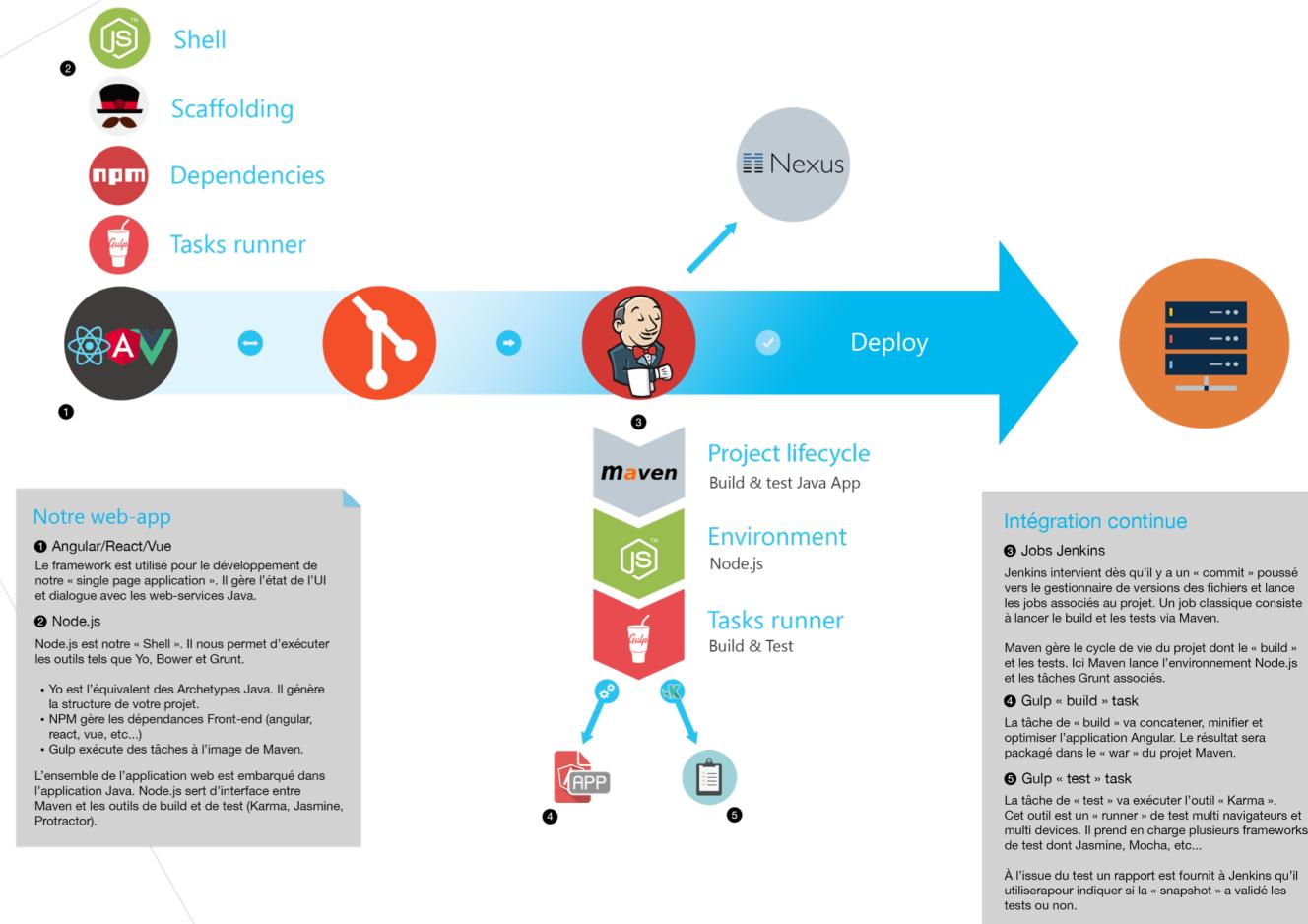
# Les tests

- \_ Unitaire : Karma & Jasmine / Mocha, Jest,
- \_ End to End : Protractor, Zombi.js, Selenium,
- \_ Test webservice : SuperAgent,
- \_ Qualité de code : TSLint,
- \_ CI : Jenkins / Travis / GitLab.



# Workflow

## Front-end, Java & Intégration Continue



# Des questions ?