



Vue 2.6.10

The Progressive JavaScript Framework

Tour De Table

- Nom / Prénom
- Missions
- Connaissances JavaScript
- Attentes

Présentation

Damien Chazoule

FullStack JS Developer

VueJS Lover ^(React Too)



@MrDoomy

Sommaire

Rappels

1. JavaScript
2. EcmaScript
3. TypeScript
4. Outils
5. Let's Code

Vue

1. Introduction
2. Syntaxe
3. Single File Component
4. Initialisation
5. Data & Props
6. Imbrication
7. Cycle De Vie
8. Directives
9. StyleSheet
10. Methods, Computed & Watch
11. Functional Components
12. Vue Router
13. Flux
14. Vuex
15. DOM Virtuel
16. Jest & Testing Library

Vue d'Ensemble

1. Frameworks
2. Infographie
3. The Way Of Jedi Dev
4. Annexes

Rappels



JavaScript

Créé par **Brendan Eich** en 1995

Alliance entre **Sun** et **Netscape**

Langage faiblement typé

Programmation fonctionnelle

Volonté d'un Web plus dynamique

Version 10 depuis Juin 2019

EcmaScript

By Ecma International

Créé en 1996

Standardisation du **JavaScript**

On parle de la norme **ECMA-262**

- 1997 : ES1
- 1998 : ES2 ^(ISO)
- 1999 : ES3 ^(RegExp, Exception)
- 2009 : **ES5** *aka* ES3.1 ^(JSON, XHR)
- (ES)2015 : ES6 ^(Classes, Keywords, Promises)
- (ES)2016 : ES7 ^(Includes, Exp Operator)
- (ES)2017 : ES8 ^(Async / Await)
- (ES)2018 : ES9 ^(Finally, RegExp)
- (ES)2019 : ES10 ^(Flat / FlatMap)
- (ES)2020 : ES11... ^(ESNext)



TypeScript

Créé par **Microsoft** en 2012

Langage de programmation libre et *Open Source*

Sur-ensemble **JavaScript** fortement typé

Support des spécifications **EcmaScript**

Multiparadigme (Client-Side / Server-Side)

Version 3 depuis Juillet 2018

Outils



1^{er} version en 2009

Environnement **JavaScript**

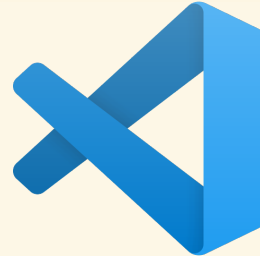
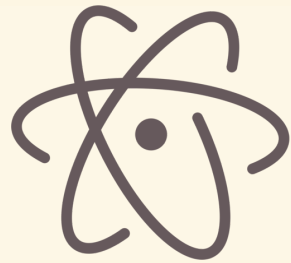
Basé sur le moteur **Chrome V8**

Utilisé en tant que plateforme logicielle

Contient nativement un serveur Web

Dernière version **LTS** : 12.14.x

Gestionnaire de paquets ^(officiel)



Éditeurs

Éditeur	Avantages	Inconvénients
Atom	Outil entièrement personnalisable et puissant	Quelques problèmes de performance sur la durée
Sublime Text	Flexible et rapide idéal pour le développement Web	Licence payante pour les fonctionnalités avancées
Visual Studio Code	Prêt à l'emploi avec le débogueur et Git par défaut	Pas autant de fonctionnalités qu'un IDE
WebStorm	Fonctionnalités inédites avec notamment le <i>Running Coverage</i>	Modèle économique payant dégressif



Git

L'indispensable gestionnaire de versions
Meilleure visualisation de l'avancement du projet
Développement de fonctionnalités par branches
La nécessité de l'historisation des fichiers
Travail collaboratif centralisé



Yarn

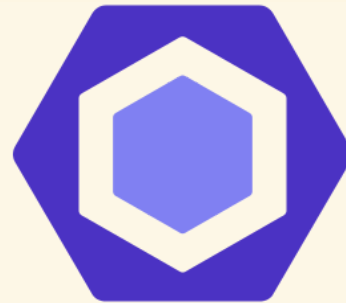
Alternative à **NPM**

Gérer vos dépendances

Sécuriser vos versions

Récupération des librairies **JavaScript**

Amélioration des performances du projet



ESLint

Optimisation de `code`

Analyse statique du `code` source

Contrôle l'écriture du `code` **JavaScript**

Basé sur la convention de `code` d'EcmaScript



Des feuilles qui ont du style
Du *design* pour votre application
Choisir entre le **Less**, le **Sass** ou le **Stylus**



webpack

Bundler

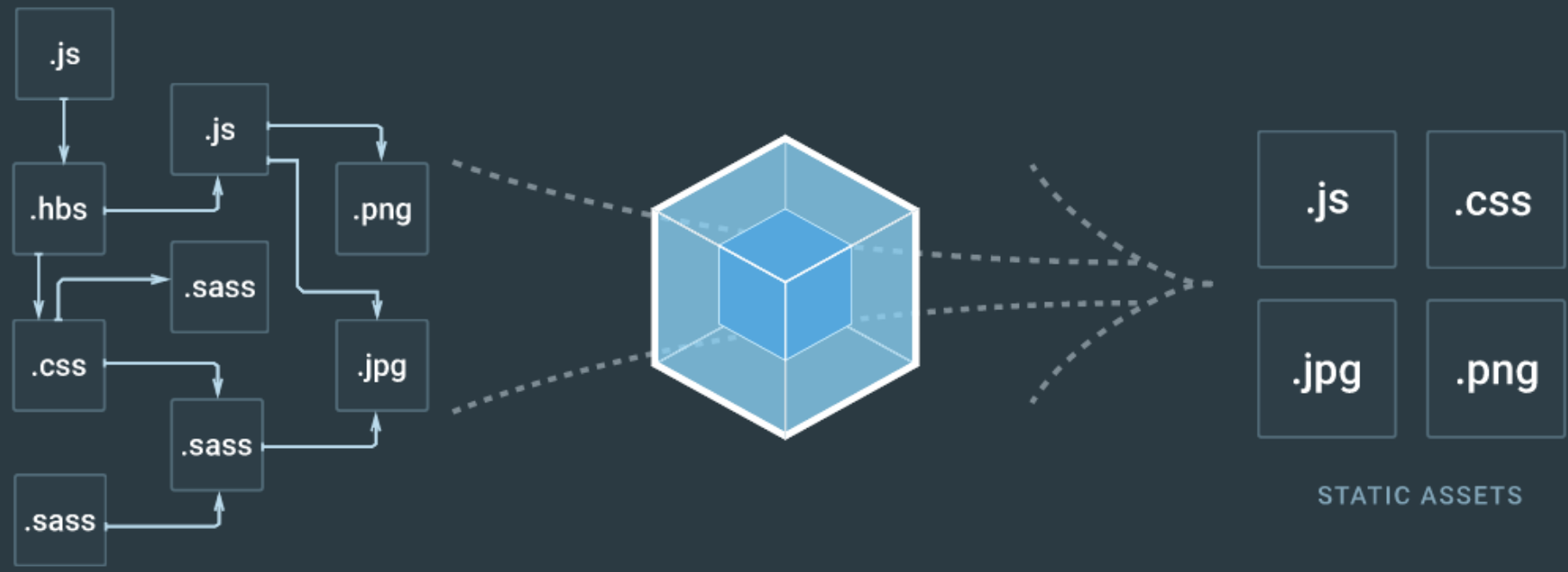
Fonctionnement par modules

Intéropérabilité des fichiers

Appel ~~de fichiers~~ de modules

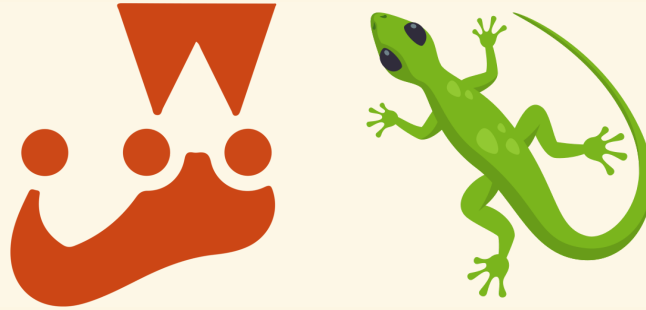
Découpage par responsabilité

Fini les casses têtes liés aux ressources (`.js`, `.json`, `.jsx`)



MODULES WITH DEPENDENCIES

STATIC ASSETS



Tests Unitaires

La base de la programmation

Technique liée à l'approche **Agile**

Tester les méthodes, les composants, les services...

Gagner du temps dans vos développements

~~Tester c'est douter !~~

Let's Code



Environnement

Préférez le mode `strict` lors de l'exécution pour la remontée d'erreurs

```
(function() {  
  'use strict';  
  
  /* ... */  
})();
```

Variables

Les types primitifs en JavaScript sont :

boolean, number, string, undefined et null

ES5

```
var myVar;  
var fortyTwo = 42;  
var helloWorld = 'Hello World';  
var truthy = true;  
console.log(typeof myVar); // undefined  
alert(helloWorld + ' !'); // Hello World !  
console.log(truthy && 'It Works'); // It Works
```

ES6+

```
let myVar;  
myVar = null;  
  
const fortyTwo = 42;  
  
const helloWorld = 'Hello World';  
helloWorld = 'Hey'; // TypeError !  
  
const truthy = true;  
alert(`${helloWorld} !`); // Hello World !  
console.log(truthy && 'It Works'); // It Works
```


Tableaux

Voici comment déclarer un tableau en **JavaScript** :

```
let tab = ['Plop', 42, true];
const another = ['Test', false];

tab.push(another);

console.log(tab.length); /* Affiche '4' dans la console */
console.log(tab[3][0]); /* Affiche 'Test' dans la console */

tab.length = 2;

alert(tab); /* Affiche '['Plop', 42]' dans la console */
```

Objets

Voici comment déclarer un objet en JavaScript :

```
let album = {};  
  
album = {  
  id: 6,  
  title: 'Migration',  
  artist: 'Bonobo',  
  released: new Date(2017, 0, 13),  
  length: 62,  
  genre: 'Downtempo',  
  label: 'Ninja Tune',  
  tracks: 12  
};  
  
delete album.genre;  
  
album.genre = ['Electronic', 'Downtempo', 'Experimental'];  
  
console.log(album['genre'][0]); // Affiche 'Electronic' dans la console
```

Fonctions

Voici comment déclarer une fonction en JavaScript :

ES5

```
var numeric = 7;
var tab = [3, '5', 7];

/* Assigination */
var remove = function(array, value) {
    var size = array.length;

    for (var i = size - 1; i >= 0; i--) {
        if (array[i] === value) {
            var index = array.indexOf(value);
            array.splice(index, 1);
        }
    }
};

remove(tab, numeric);

/* Déclaration */
function log(obj) {
    console.log('Log : ' + obj);
};

log(tab); /* Log : [3, '5'] */
```

ES6+

```
const numeric = 7;
let tab = [3, '5', 7];

/* Assigination */
const remove = (array, value) => {
    let size = array.length;

    for (let i = size - 1; i >= 0; i--) {
        if (array[i] === value) {
            const index = array.indexOf(value);
            array.splice(index, 1);
        }
    }
};

remove(tab, numeric);

/* Déclaration */
function log(obj) {
    console.log(`Log : ${obj}`);
};

log(tab); /* Log : [3, '5'] */
```

Fonctionnement

Le code **JavaScript** mêle à la fois du code exécuté et du code déclaratif

La pile d'exécution ^(Call Stack) compile le code puis l'exécute

Ainsi, le code **JavaScript** est synchrone

```
'use strict';
console.log('i:', i);

var i = 1;
execute('Hey');

function execute(message) {
  console.log(message);
}

execute('Hey');
```

```
// Compile
var i;
function execute(message) {
  console.log(message);
}

// Execute
console.log('i: ' i);
i = 1;
execute('Hey');
execute('Hey');
```

Interprétation

Quel est le résultat de cette fonction JavaScript :

1, 10, undefined ou ReferenceError ?

Pré-Compilation

```
var foo = 1;

function bar() {
  if (!foo) {
    var foo = 10;
  }

  console.log(foo);
}

bar();
```

Post-Compilation

```
var foo = 1;

function bar() {
  var foo;

  if (!foo) {
    foo = 10;
  }

  console.log(foo);
}

bar(); // 10
```

Prototype

Manière de faire de l'héritage en Javascript
C'est l'équivalent d'une classe en Java

Une sorte de patron qu'un objet peut utiliser
Complexe mais puissant dans son usage

```
/* Constructeur */  
var Object = function(one, two) {  
  
    /* Super */  
    Parent.call(this, one, two);  
  
    /* Attributs */  
    this.one = one;  
    this.two = two;  
};  
  
/* Methode */  
Object.prototype.diff = function() {  
    return (this.one - this.two);  
};  
  
var instance = new Object(6, 4);  
  
console.log(instance.diff()); // Affiche '2' dans la console
```



Vue

Introduction

Développé depuis 2014 par **Evan You**

Facilement intégrable aux projets existants

Liaison de données bi-directionnelle

Idéal pour les applications à grande échelle

Outil **CLI** très performant

Framework rapide, léger et fiable pour un code stable

Utilisé par **GitLab**, **Behance**, **Nintendo**...

Command Line Interface

L'indispensable `@vue/cli`

Outil CLI **Vue** (officiel)

Projet prêt à l'emploi grâce à `vue-cli-service`

Support de **TypeScript**, **PostCSS**, **Pug**...

Autorise la surcharge pour la personnalisation Webpack

```
npm i -g @vue/cli  
vue create my-project
```

Syntaxe

Ci-dessous un exemple de code **Vue** avec le fichier `index.html` :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Vue</title>
  </head>

  <body>
    <div id="root">
      <hello-world userName="Rick"></hello-world>
    </div>

    <script src="https://unpkg.com/vue"></script>
    <script>
      Vue.component('hello-world', {
        props: ['userName'],
        template: '<span style="color: #42b884;">Hi {{ userName }} !</span>'
      });

      new Vue({
        el: '#root'
      });
    </script>
  </body>
</html>
```

Single File Component

Préférez la syntaxe **Vue**², exemple avec le fichier HelloWorld.vue :

```
<template>
  <span class="green">Hi {{ userName }} !</span>
</template>

<script>
export default {
  name: 'hello-world',
  props: ['userName']
};
</script>

<style scoped>
.green {
  color: #42b884;
}
</style>
```

*Les fichiers **Vue** doivent être transpilés*

Initialisation

~~1^{er} Étape : Écrire son composant~~

2^{ème} Étape : Monter son composant dans le DOM

HTML

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>

    <!-- Built Files Will Be Auto Injected -->
    <div id="root"></div>
  </body>
</html>
```

Main.JS

Au début de l'application, dans l'instance principale...
On appelle **Vue** afin de rendre le composant dans le DOM !

```
import Vue from 'vue';
import HelloWorld from './path/to/HelloWorld.vue';

Vue.config.productionTip = false;

new Vue({
  el: '#root',
  components: {
    HelloWorld
  },
  template: '<hello-world></hello-world>'
});
```

Data & Props

*Gérer les données et les exposer dans le **DOM***

Data

Les données sont mutables
Données locales au composant
Gestion de l'état grâce aux `methods`

```
<template>
  <div>
    <span>{{ counter }}</span>
    <button @click="increment">+1</button>
  </div>
</template>

<script>
export default {
  name: 'Counter',
  data() {
    return {
      counter: 0
    }
  },
  methods: {
    increment() {
      this.counter = this.counter++;
    }
  }
};
</script>
```

Props

Les propriétés sont immutables

N'importe quelle valeur (listes, objets, fonctions, booléens...)

Données transmises d'un composant parent à un enfant

Passer des propriétés comme des attributs d'un élément HTML

Enfant

```
<template>
  <div>
    <h3>{{ label }}</h3>
    <span>{{ counter }}</span>
  </div>
</template>

<script>
export default {
  name: 'Child',
  props: [
    'label',
    'counter'
  ]
};
</script>
```

Parent

```
<template>
  <Child label="Compteur" :counter="counter" />
</template>

<script>
import Child from './Child.vue';

export default {
  name: 'Parent',
  components: {
    Child
  },
  data() {
    return {
      counter: 42
    }
  }
};
</script>
```


Typage

Définir le type des propriétés
Imposer des valeurs par défaut

```
<template>
  <div>
    <span>{{ counter }}</span>
    <button @click="increment">+1</button>
  </div>
</template>

<script>
export default {
  name: 'Counter',
  props: {
    counter: {
      type: Number,
      default: 0
    },
    increment: {
      type: Function,
      required: true
    }
  }
};
</script>
```

Imbrication

`<slot></slot>` permet d'accéder aux éléments imbriqués

Il est possible de nommer les `<slot></slot>`

Parent

```
<template>
  <Child>
    <span>World</span>
  </Child>
</template>

<script>
import Child from './Child.vue';

export default {
  name: 'Parent',
  components: {
    Child
  }
};
</script>
```

Enfant

```
<template>
  <div>
    <h1>Hello</h1>
    <slot></slot>
  </div>
</template>

<script>
export default {
  name: 'Child'
};
</script>
```

For Each

Construction dynamique de noeud DOM

```
<template>
  <ul>
    <li v-for="(item, idx) in myList" :key="idx">
      {{ item }}
    </li>
  </ul>
</template>

<script>
export default {
  name: 'Listing',
  props: {
    myList: {
      type: Array,
      default: () => ['Awesome', 'Vue', 'Course']
    }
  }
};
</script>
```

Ne pas oublier d'identifier chaque élément lors d'un rendu dynamique grâce àkey

Modèle Courant

Mise à jour des props d'un composant en passant ^(implicitement) par le parent

Parent

```
<template>
  <Child
    :counter="counter"
    @increment="counter++" />
</template>

<script>
import Child from './Child.vue';

export default {
  name: 'Parent',
  components: {
    Child
  },
  data() {
    return {
      counter: 0
    }
  }
};
</script>
```

Enfant

```
<template>
  <div>
    <h1>{{ counter }}</h1>
    <button @click="$emit('increment')">
      +1
    </button>
  </div>
</template>

<script>
export default {
  name: 'Child',
  props: ['counter']
};
</script>
```

Mixins

Factoriser vos comportements grâce aux Mixins

```
export const mixinCounter = {
  data() {
    return {
      value: 0
    }
  },
  methods: {
    increment() {
      this.value = this.value++;
    },
    decrement(val) {
      this.value = this.value - val;
    }
  }
};
```

```
<template>
  <div>
    <h1>Compteur</h1>
    <button @click="increment">
      {{ value }} +1
    </button>
    <button @click="() => decrement(2)">
      {{ value }} -2
    </button>
  </div>
</template>

<script>
import { mixinCounter } from '@mixins';

export default {
  name: 'Counter',
  mixins: [mixinCounter]
};
</script>
```

Cycle De Vie



Cas d'Usage

Appel d'une API lorsque le composant est monté dans le DOM

```
<template>
  <!-- ... -->
</template>

<script>
import axios from 'axios';

export default {
  name: 'HelloWorld',
  mounted() {
    console.log('Mounted !');

    axios.get('/api/hello-world').then(response => {
      console.log(response.data);
    });
  }
};
</script>
```


Directives

Concept provenant d'**AngularJS**
Généralement des composants réutilisables
Méthodes qui régissent un comportement
Très proche des composants Web
Injectable dans le template HTML

Directives

Directive	Description
<code>v-once</code>	Permet de bloquer la mise à jour d'une valeur dynamique dans le template
<code>v-html</code>	Permet d'interpréter du code HTML de manière dynamique
<code>v-model</code>	Indique que toute modification de ce champ doit être répercutée sur le modèle
<code>v-show</code>	Permet d'afficher un élément du DOM en fonction de la valeur d'un paramètre
<code>v-if v-else-if v-else</code>	Permet de créer un élément du DOM en fonction de la valeur d'un paramètre
<code>v-for</code>	Permet d'itérer sur une liste

Abréviations

Directive	Description
<code>v-bind</code>	Permet de 'binder' une donnée dynamique et / ou d'évaluer du code :
<code>:</code>	<code>v-bind:title="myTitle + ' !'"</code>
<code>v-on</code>	Permet de 'binder' une donnée dynamique et / ou d'évaluer du code :
<code>@</code>	<code>v-bind:title="myTitle + ' !'"</code>

*Il est aussi possible de définir ses propres directives dans l'instance de **Vue***

StyleSheet

Customiser son composant avec PostCSS

Statique

Utilisation du template HTML

```
<template>  
  <p style="text-decoration: underline;">Hey ! What's Up !?</p>  
</template>
```

Dynamique

Utilisation de la directive v-bind

```
<template>
  <p :style="{ textDecoration }">Hey ! What's Up !?</p>
  <p :style="getStyle">Nothin' Bro !</p>
</template>

<script>
export default {
  data() {
    return {
      textDecoration: 'underline'
    }
  },
  computed: {
    getStyle() {
      return {
        textDecoration: this.textDecoration
      };
    }
  }
};
</script>
```

Style

Utilisation de la portion réservée aux styles

```
<template>
  <p class="underline">Nothin' Bro !</p>
</template>

<style lang="scss" scoped>
p {
  .underline {
    text-decoration: underline;
  }
}
</style>
```

Methods, Computed & Watch

Methods

Réagir aux événements qui se produisent dans le DOM

```
<template>
  <button @click="handleClick()">Say Hello</button>
</template>

<script>
export default {
  data() {
    return {
      firstName: 'Rick',
      lastName: 'Sanchez'
    };
  },
  methods: {
    handleClick() {
      alert(`Hello ! I'm ${this.firstName} ${this.lastName}`);
    }
  }
};
</script>
```

Computed

Définir de nouvelles données à partir de données déjà existantes

```
<template>
  <span>{{ fullName }}</span>
</template>

<script>
export default {
  data() {
    return {
      firstName: 'Rick',
      lastName: 'Sanchez'
    };
  },
  computed: {
    fullName() {
      return this.firstName + ' ' + this.lastName;
    },
    now: () => Date.now()
  }
};
</script>
```

Watch

Effectuer des actions lorsque les propriétés d'une donnée changent

```
<template>
  <input @change="handleChange()" :value="userName">
</template>

<script>
export default {
  data() {
    return {
      userName: 'Morty'
    };
  },
  methods: {
    handleChange(event) {
      this.userName = event.target.value;
    }
  },
  watch: {
    userName() {
      console.log(this.userName);
    }
  }
};
</script>
```

Ici, la bonne pratique est d'utiliser la directive v-model

Functional Components

Les composants n'incluant que la fonction `render()` peuvent s'écrire autrement
Contient quelques `data` et / ou `props`

ES6+

```
Vue.component('greeting', {
  functional: true,
  props: ['userName'],
  render: function(createElement, context) {
    createElement(
      'span',
      { style: { color: '#2a2c2e' }},
      `Hi ${context.props.userName} !`
    );
  }
});
```

JSX

```
export default {
  name: 'Greeting',
  functional: true,
  props: ['userName'],
  render: function(h, { props }) {
    return (
      <span style={{ color: '#2a2c2e' }}>
        Hi {userName} !
      </span>
    );
  }
};
```



Vue Router

Explication

Affichage des composants en fonction de l'URL

Inspirée par le Router du framework Ember

Librairie officielle maintenue par l'équipe de **Vue**

Facilement implémentable avec Vuex

Importance de la navigation par URL

Mapping

Définition des chemins pour chaque composants de l'application
Vue gère nativement le découpage dynamique

```
import Vue from 'vue';
import Router from 'vue-router';
import Home from '@components/Home.vue';
import Login from '@components/Login.vue';

// Code Splitting
const Contact = () => import('@components/Contact.vue');

Vue.use(Router);

const router = {
  mode: 'history',
  routes: [
    { path: '/', component: Home },
    { path: '/login', name: 'login', component: Login },
    { path: '/contact/:userName', name: 'contact', component: Contact, props: true }
  ]
};

export default new Router(router);
```

Intégration

Ajout du mapping dans l'instance principale de **Vue**

```
import Vue from 'vue';
import App from '@/components/App.vue';
import router from 'router';

Vue.config.productionTip = false;

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


Navigation

La navigation par URL peut ensuite se faire de deux manières :

1. Grâce au composant **router-link** au sein du template
2. Dynamiquement via la donnée `$history` étendue par l'instance de **Vue**

```
<template>
  <div>
    <nav>
      <router-link to="/login">Login</router-link>
      <button @click="goToContact('morty')">
        Contact
      </button>
    </nav>
    <router-view></router-view>
  </div>
</template>

<script>
export default {
  name: 'NavBar',
  methods: {
    goToContact: function(userName) {
      this.$router.push({ name: 'contact', params: { userName } });
    }
  }
};
</script>
```

Flux

Concept

Architecture pour les flux de données unidirectionnels
Il existe plusieurs implémentations de **Flux** en **JavaScript** :

- Flux
- Fluxxor
- MobX
- Redux
- Reflux
- Relay
- **Vuex**

...



Philosophie

Gestion du `state` des composants simplifiée

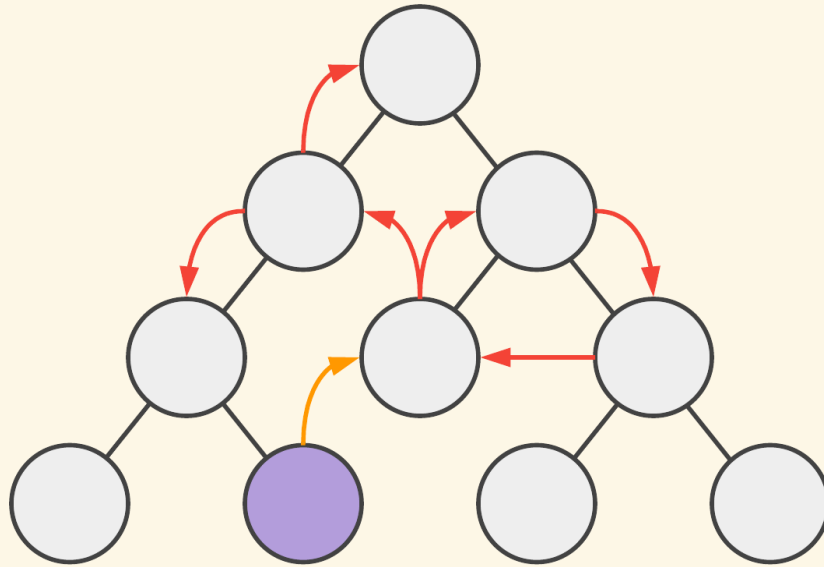
Le **store** est accessible à tout moment

Les composants enfants n'héritent pas du `state` des parents

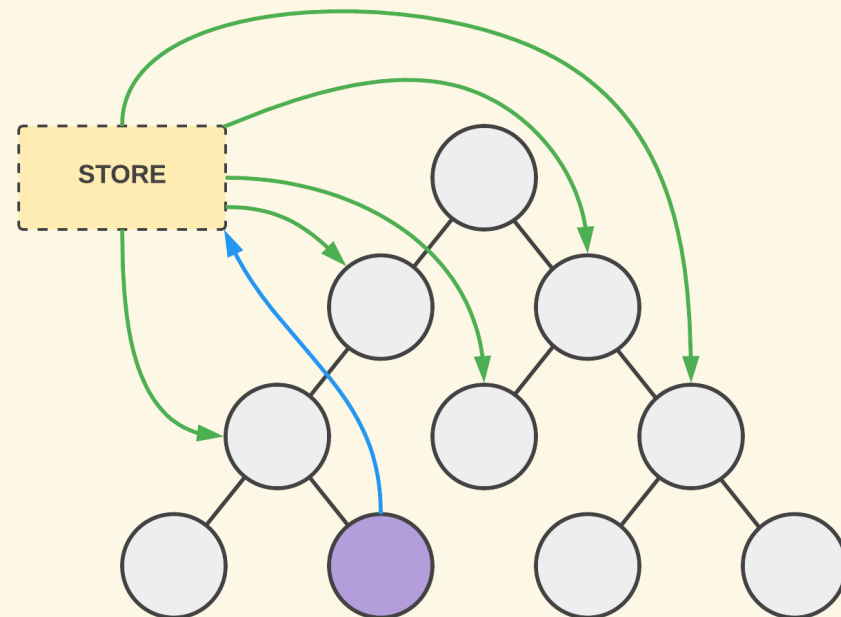
Aucune donnée en doublon (un seul point de vérité)

Meilleure conception de votre application

Sans Vuex



Avec Vuex





Vuex

Principe

Vuex est un gestionnaire d'état pour les applications

Permet de suivre l'état global de l'applicatif

Développé spécialement pour le framework **Vue**

On dit que le **store** est l'unique source de vérité

Contient les `mutations` de l'applicatif

Le **store** est en lecture seul

Actions

Mise à jour du **store** uniquement via les `mutations` appelés par des *actions*

La comparaison de l'état permet d'identifier le changement

Les *constantes* sont les références utilisées par les `mutations`

```
// Constants
export const CREATE_TASK = 'TASKS/CREATE_TASK';
export const COMPLETE_TASK = 'TASKS/COMPLETE_TASK';
export const CLEAR_TASK = 'TASKS/CLEAR_TASK';

export const actions = {
  createTask({ commit }, task) {
    commit(CREATE_TASK, { task });
  },
  completeTask({ commit }, id) {
    commit(COMPLETE_TASK, { id });
  },
  clearTask({ commit }, id) {
    commit(CLEAR_TASK, { id });
  }
};
```

Mutations

L'action trace le changement dans le **store**

La `mutation` se charge d'effectuer la modification

```
import { CREATE_TASK, COMPLETE_TASK, CLEAR_TASK } from './actions';

export const mutations = {
  [CREATE_TASK](state, payload) {
    const { task } = payload;
    state.tasks.push(task);
  },
  [COMPLETE_TASK](state, payload) {
    const { id } = payload;
    state.tasks.map(task => task.id === id ? { ...task, completed: !task.completed } : task);
  },
  [CLEAR_TASK](state, payload) {
    const { id } = payload;
    state.tasks.filter(task => task.id !== id);
  }
};
```

Getters

Les `getters` permettent de récupérer l'état d'une donnée dans le **store**

Il est possible de passer des paramètres aux `getters`

```
export const getters = {  
  getTasks(state) {  
    return state.tasks | [];  
  },  
  getTaskById: (state, getters) => id => {  
    return getters.getTasks.find(task => task.id === id);  
  }  
};
```

Store

Composer votre **store** afin d'avoir accès aux actions, getters (et mutations)

```
import Vue from 'vue';
import { Store } from 'vuex';
import { actions } from './actions';
import { getters } from './getters';
import { mutations } from './mutations';

Vue.use(Vuex);

const store = {
  state: {
    tasks: []
  },
  actions,
  getters,
  mutations
};

export default new Store(store);
```

Initialisation

Chargement du **store** dans l'instance principale de **Vue**: `main.js`

```
import Vue from 'vue';
import App from '@components/App.vue';
import store from './store';

Vue.config.productionTip = false;

new Vue({
  store,
  render: (h) => h(App)
}).$mount('#root');
```

Utilisation

Vuex permet de mapper le **store** dans un composant
Ce mapping est nécessaire pour pouvoir dispatcher des *actions*

```
<template>
  <ul v-for="(task, idx) in tasks">
    <li :key="idx">
      {{ task.label }}
      <button @click="updateTask(task.id)">
        Update
      </button>
    </li>
  </ul>
</template>

<script>
export default {
  computed: {
    tasks() {
      return this.$store.getters.getTasks;
    }
  },
  methods: {
    updateTask(id) {
      this.$store.dispatch('completeTask', id);
    }
  }
};
</script>
```

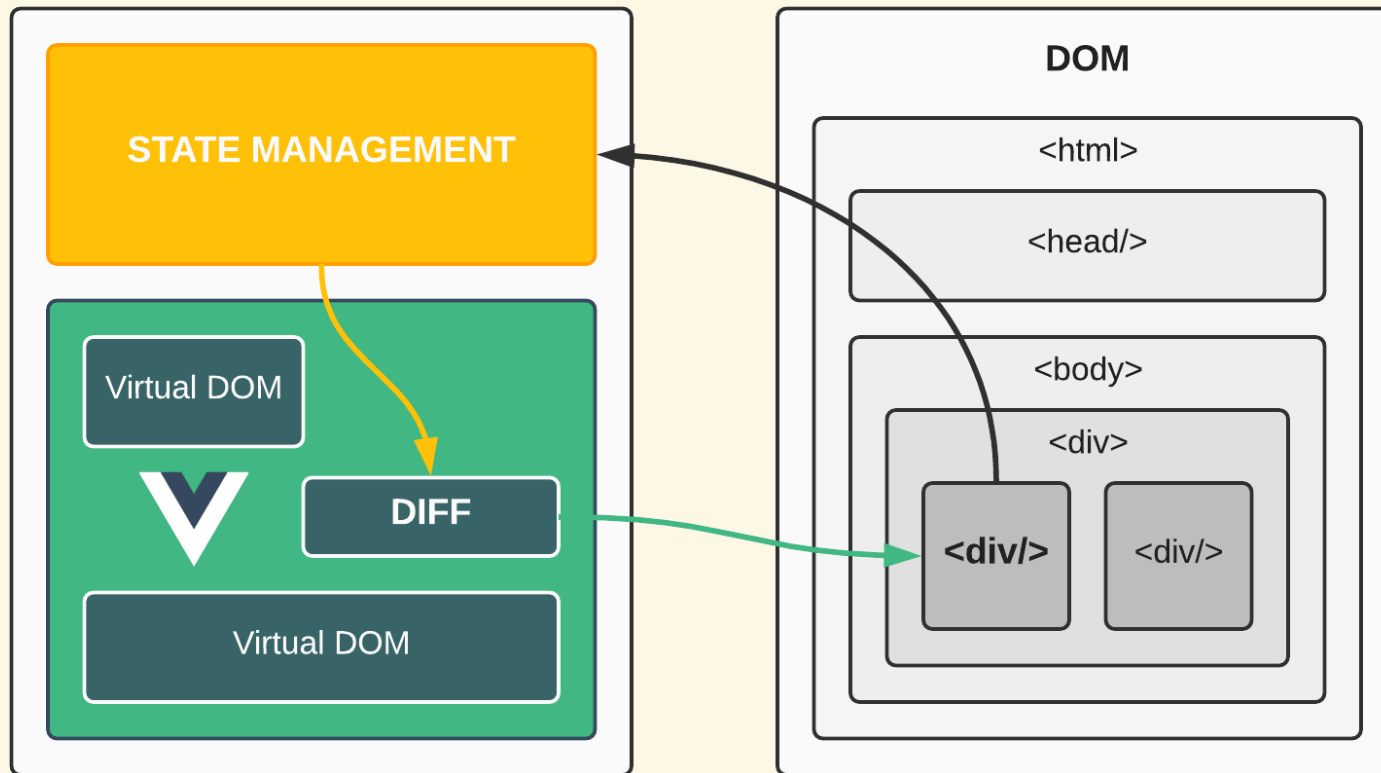
```
<template>
  <div>
    <h2>{{ task.label }}</h2>
    <button @click="clearTask(task.id)">
      Remove
    </button>
  </div>
</template>

<script>
import { mapActions, mapGetters } from 'vuex';

export default {
  computed: {
    ...mapGetters({
      taskWithId: 'getTaskById'
    }),
    task() {
      return this.taskWithId(42);
    }
  },
  methods: {
    ...mapActions([
      'clearTask'
    ])
  }
};
```

DOM Virtuel

Le **Virtual DOM**, la vraie révolution portée par **React** qui rend **jQuery** obsolète





Jest & Testing Library

Configuration

Récupération des librairies utilitaires :

```
npm install --save-dev @testing-library/vue @testing-library/jest-dom
```

Initialisation du fichier `setupTests.js` :

```
import '@testing-library/jest-dom/extend-expect';
```

Tester Avec Testing Library

Création d'une suite de tests unitaires `App.spec.js` :

```
import { render, fireEvent } from '@testing-library/react';
import App from '../App';

describe('<App />', () => {
  it('Should Component Renders Itself', () => {
    const { container } = render(App);
    expect(container).toBeDefined();
  });

  it('Should SnapShot Be Stabilised', () => {
    const { container } = render(App);
    expect(container).toMatchSnapshot();
  });

  it('Should Render Contains Text', () => {
    const { queryByText } = render(App, {
      props: { content: 'Hello World' }
    });
    expect(queryByText('Hello World')).toBeInTheDocument(); // Enhanced API
  });

  it('Should Click Trigger Works Well', async () => {
    const { getByRole, emitted } = render(App);
    await fireEvent.click(getByRole('button'));
    expect(emitted()).toHaveProperty('click');
  });
});
```

Tester Avec Jest

Création d'une suite de tests unitaires `tasks.spec.js`:

```
import { actions } from '../actions';
import { getters } from '../getters';
import { mutations } from '../mutations';

describe('Tasks', () => {
  it("Should 'createTask' Returns 'type' & 'payload'", () => {
    const commit = jest.fn();
    const task = { id: 42, label: 'Lorem Ipsum' };
    actions.createTask({ commit }, task);
    expect(commit).toHaveBeenCalledWith('TASKS/CREATE_TASK', task);
  });

  it("Should 'getTasks' Returns State", () => {
    const state = { tasks: [{ id: 42, label: 'Lorem Ipsum' }] };
    expect(getters.getTasks(state)).toHaveLength(1);
  });

  it("Should 'TASKS/CREATE_TASK' Case Returns State", () => {
    const initialState = [];
    const task = { id: 42, label: 'Lorem Ipsum' };
    mutations['TASKS/CREATE_TASK']({ pizzas: [ ...initialState ] }, task);
    expect(state.pizzas).toEqual([
      ...initialState,
      task
    ]);
  });
});
```

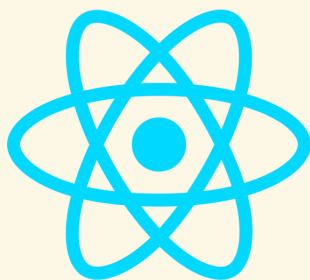
Vue d'Ensemble

Frameworks



Angular

Développé depuis 2014 par **Google**
Apparition du langage **TypeScript** (By Microsoft)
Utilisation massive des décorateurs
Courbe d'apprentissage abrupt
Processus de développement plus rapide
Fonctionnalités de test avancées



React

Créé en 2013 par **Facebook**

Utilisation massive du langage JSX

Idéal pour les projets *Front-End* volumineux

Implémentation de l'architecture **Flux**

Initialement publié sous licence BSD

React Native pour développer sur mobile



Svelte

Propulsé par **Rich Harres** dès 2018

Compilateur **VanillaJS** ultra-performant

Utilisation du DOM à l'instar du **Virtual DOM**

Dédié à l'IoT, les *wearables*, le Web embarqué, etc...

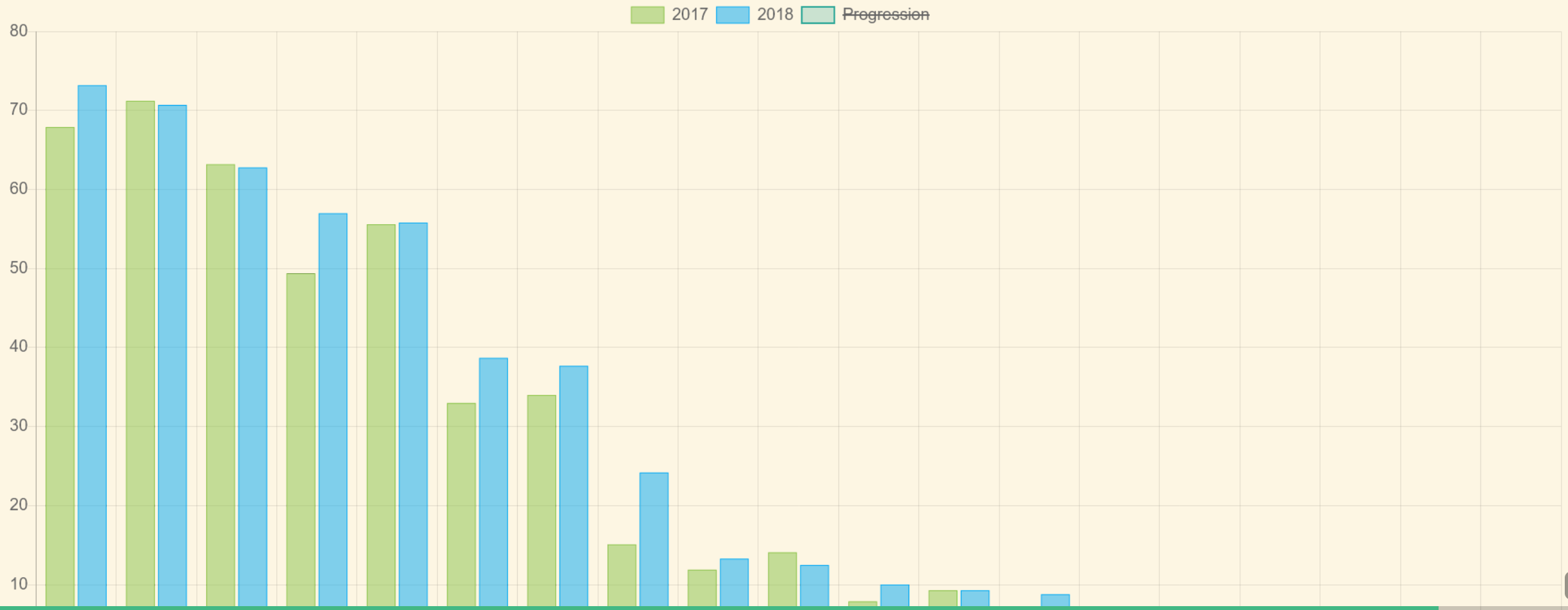
Sapper pour le rendu côté serveur (code-splitting)

La 3^{ème} version (2019) prône la "vraie" réactivité

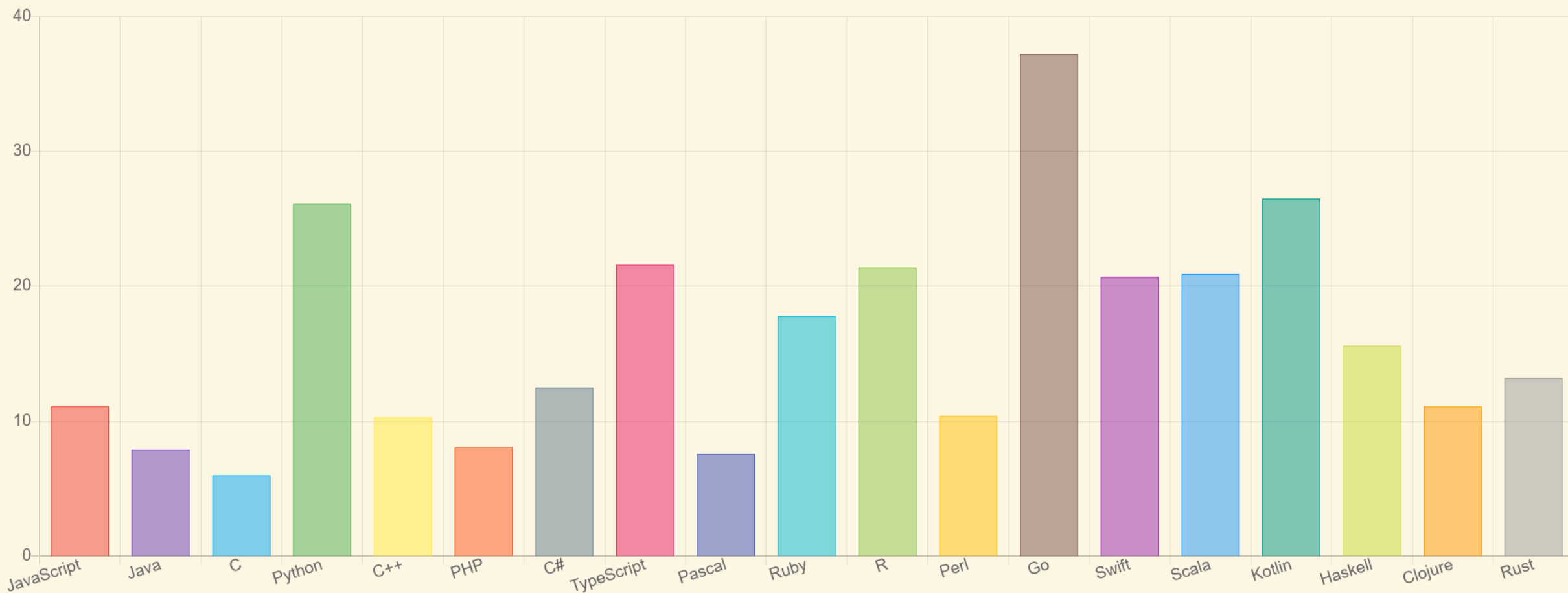
Infographie

Langages

Connaissances en 2017 Vs. 2018

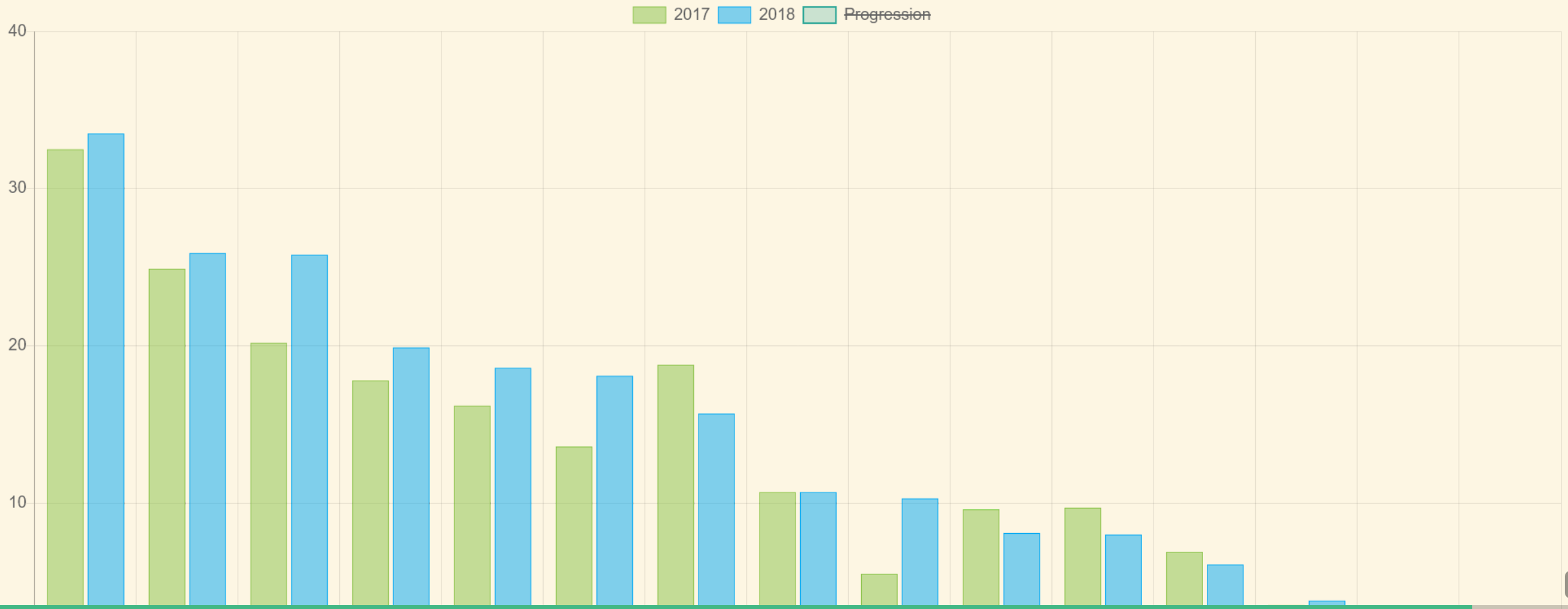


Quels langages les développeurs prévoient-ils d'apprendre ?

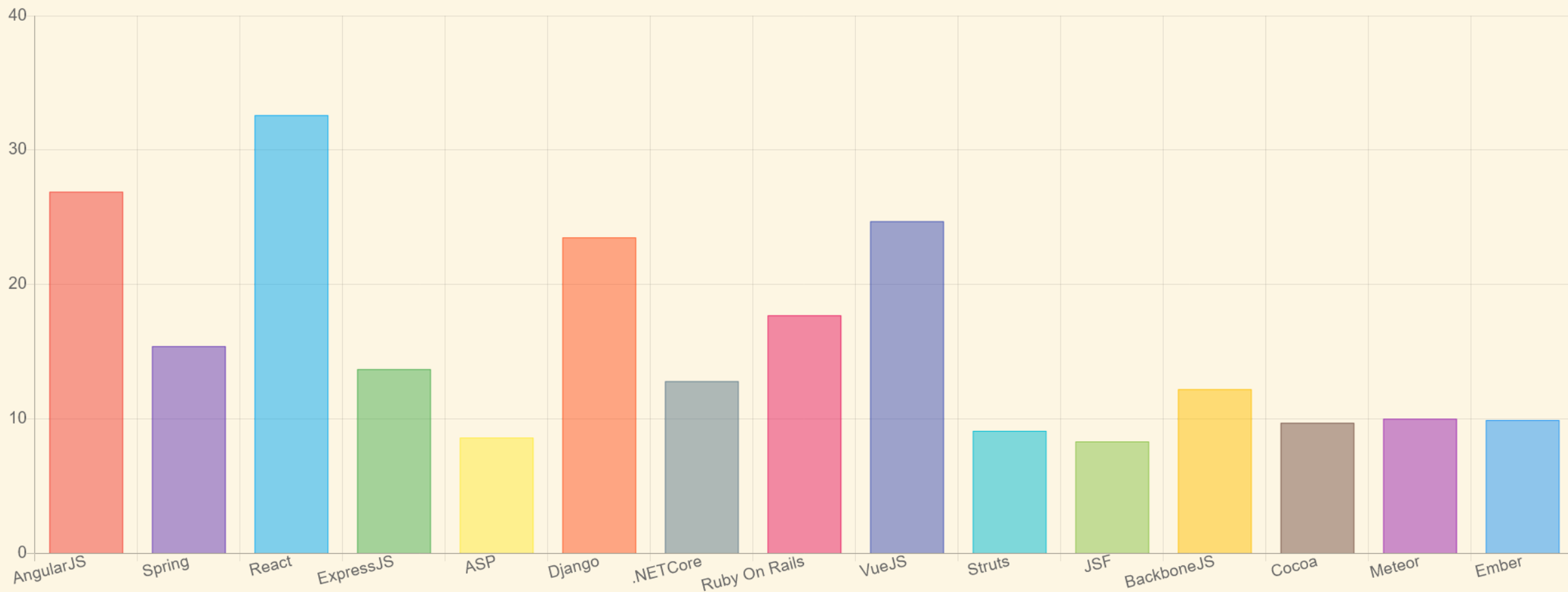


Framework

Connaissances en 2017 Vs. 2018

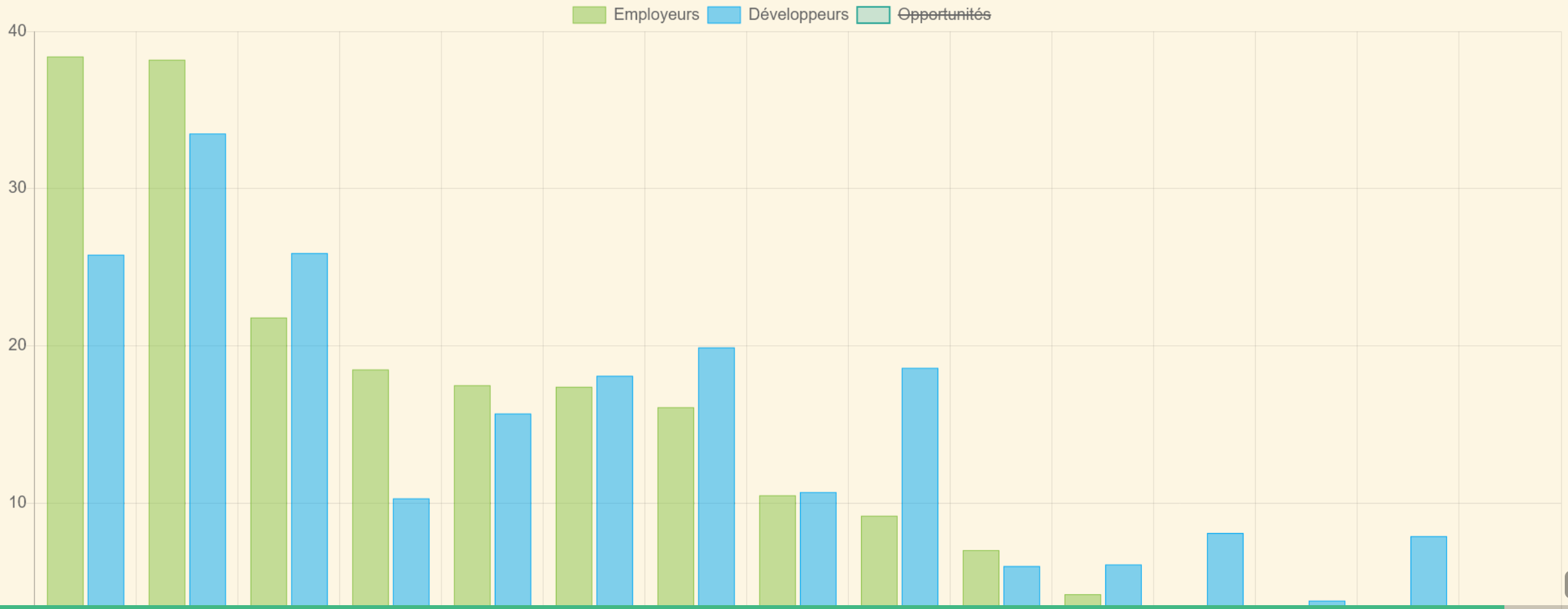


Quels frameworks les développeurs prévoient-ils d'apprendre ?



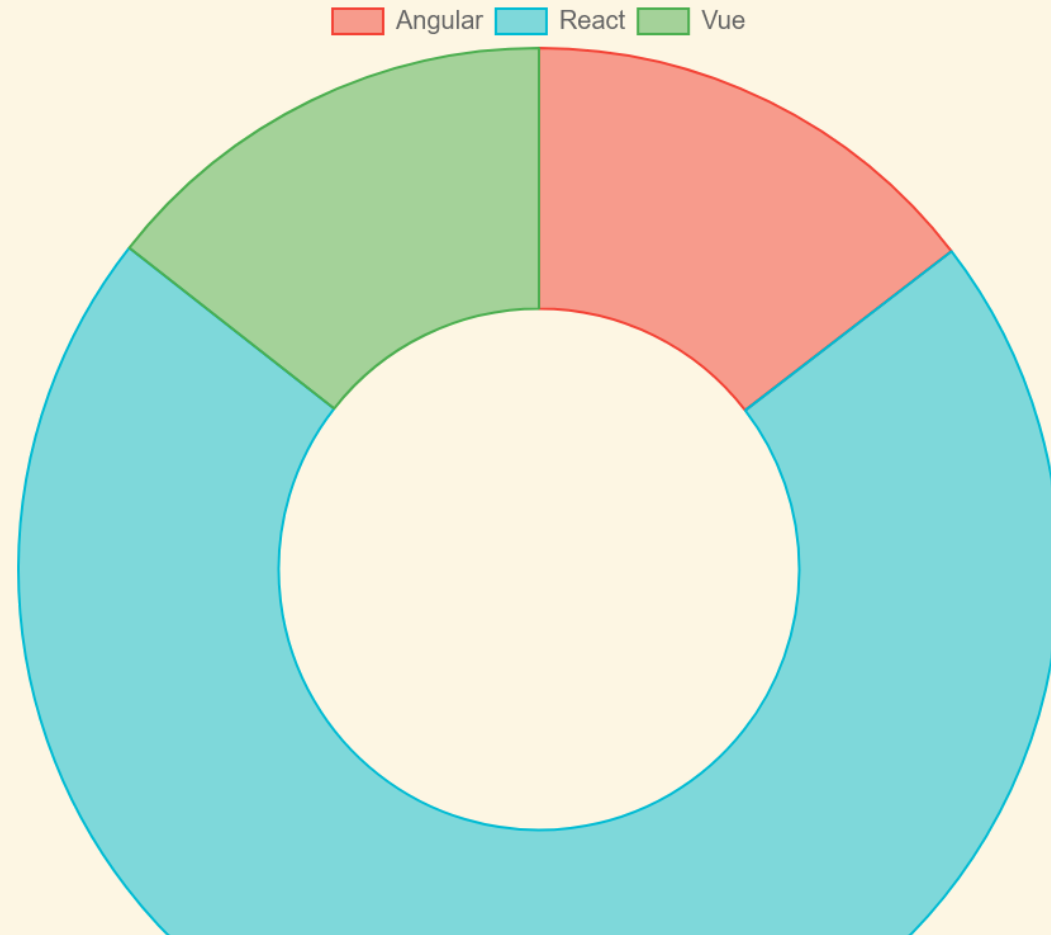
Frameworks

Besoin des employeurs Vs. Connaissances des développeurs

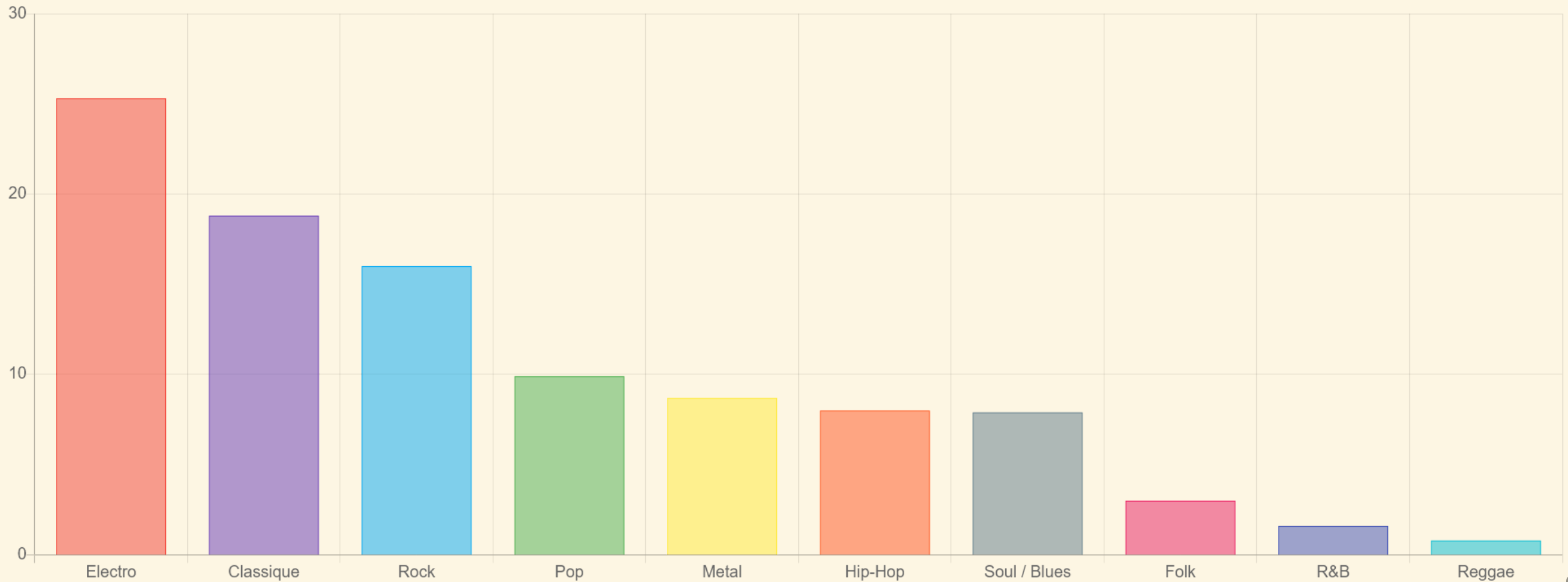


Téléchargements NPM*

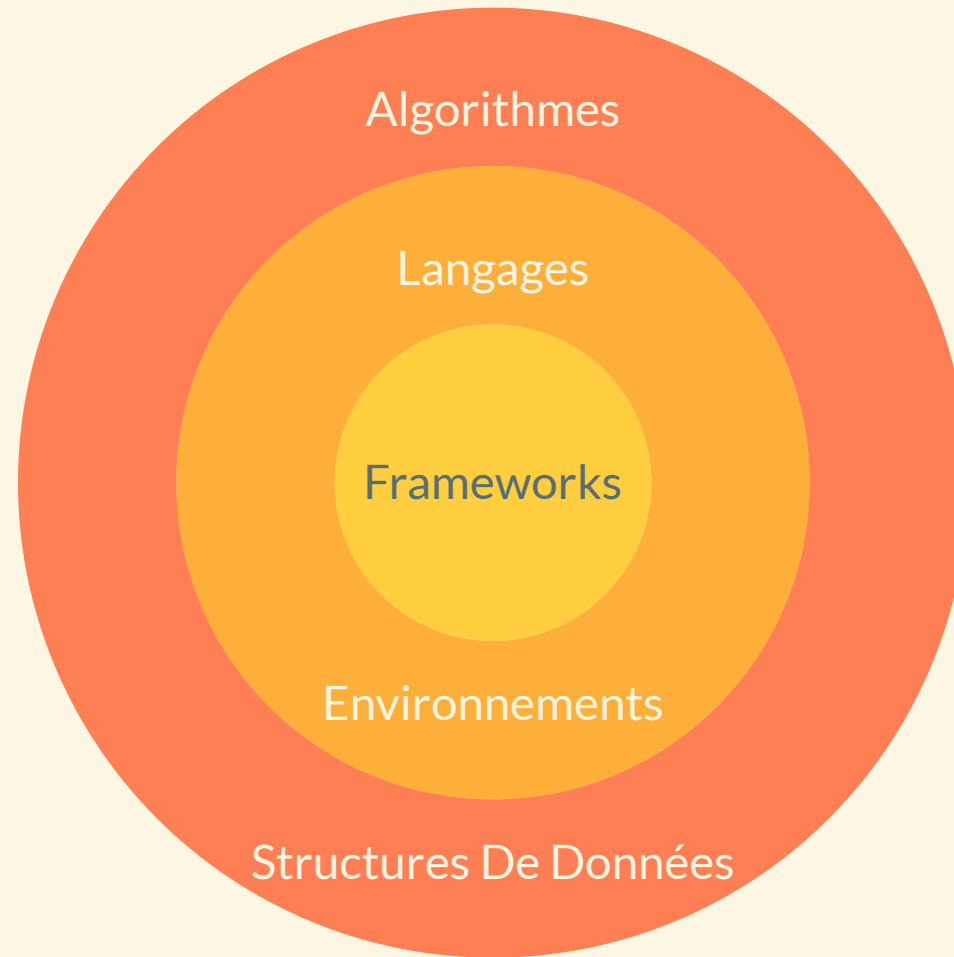
Angular Vs. React Vs. Vue



Quel style de musique écouter lorsqu'on code ?



The Way Of ~~Jedi~~ Dev



Citation

"I love writing software in NodeJS.

I've had a thirty year career in software, and NodeJS is the first time I really started having fun."

- CJ Silvero, CTO @ NPM Inc. -

Merci de votre attention !

Des questions ?

Annexes

Source	Lien
Vue	https://vuejs.org/
Vue Router	https://router.vuejs.org/
Vuex	https://vuex.vuejs.org/
Dev Community 	https://dev.to/
HackerRank	https://research.hackerrank.com/developer-skills/2019/
MDN	https://developer.mozilla.org/
MrDoomy	https://www.mrdoomy.xyz/
RevealJS	https://revealjs.com/