

REACT

La librairie **JavaScript** conçue pour les interfaces popularisée par **Facebook**

PRÉSENTATION

DAMIEN CHAZOULE

Développeur *Full-Stack JS*

Référent **React**



TOUR DE TABLE

- Nom / Prénom
- Missions
- Connaissances **JavaScript**
- Attentes

SOMMAIRE

RAPPELS

1. JavaScript
2. EcmaScript
3. Let's Code
4. NodeJS & NPM
5. Utils

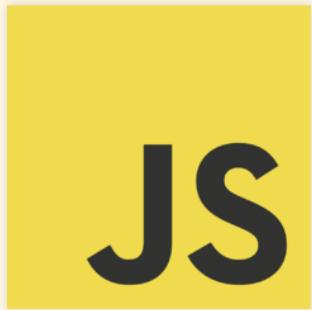
REACT

1. Introduction
2. Syntaxe
3. JSX
4. Initialisation
5. State & Props
6. Imbrication
7. Syntaxe Stateless
8. Cycle De Vie
9. Pure Components
10. Context
11. Architectures Projets
12. React Router
13. Flux
14. Redux
15. DOM Virtuel
16. Jest & Enzyme

VUE D'ENSEMBLE

1. Frameworks
2. Infographie
3. 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 9 depuis Juin 2018

ECMASCRIPT

BY ECMA INTERNATIONAL

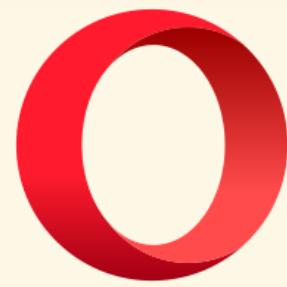
Créé en 1996

Standardisation du **JavaScript**

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

- 1997 : ES1
- 1998 : ES2 (ISO)
- 1999 : ES3 (Exceptions, RegEx...)
- 2009 : **ES5** (AJAX)
- (ES)2015 : ES6 (Babel)
- (ES)2016 : ES7 (Array...)
- (ES)2017 : ES8 (Async / Await)
- (ES)2018 : ES9 (Rest / Spread)
- (ES)2019 : ES10... (ESNext)

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

JAVASCRIPT

```
var numeric = 1;  
  
var text = '2';  
  
var binary = true;  
  
var example = 'binary = ' + binary;  
  
alert(binary); // binary = true  
  
numeric = numeric + parseInt(text);  
  
console.log(numeric); // 3
```

ES6+

```
let numeric = 1;  
  
const text = "2";  
  
const binary = true;  
  
let example = `binary = ${binary}`;  
  
alert(binary); // binary = true;  
  
numeric += parseInt(text);  
  
console.log(numeric); // 3
```

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** :

JAVASCRIPT

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

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);

var log = function(obj) {
    console.log('Log : ' + obj);
};

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

ES6+

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

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);

const log = obj => {
    console.log(`Log : ${obj}`);
};

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

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
Puissant mais complexe à manipuler

```
/* 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
```



1^{er} version en 2009

Environnement JavaScript

Basé sur le moteur **Chrome V8**

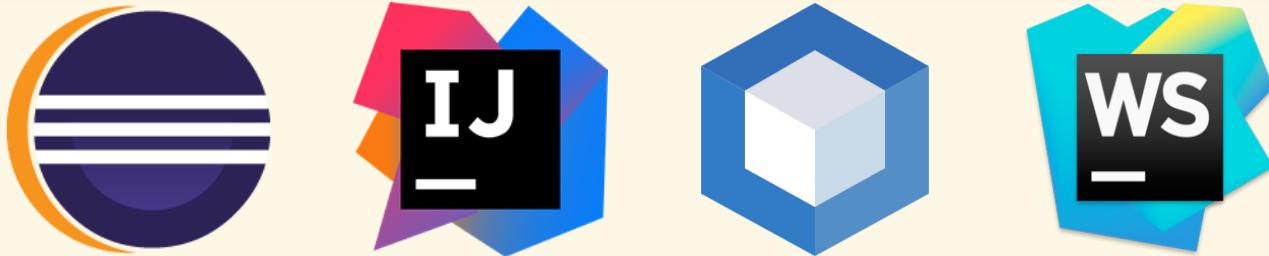
Utilisé en tant que plateforme logicielle

Contient nativement un serveur Web

Dernière version **LTS** : 10.15.0

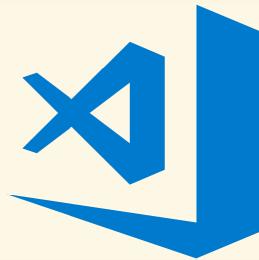
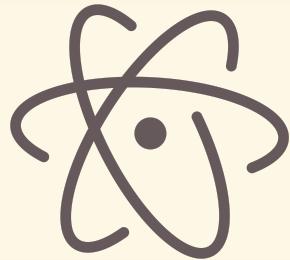
Gestionnaire de paquets ^(officiel)

OUTILS



IDE

IDE	Avantages	Inconvénients
Eclipse	Une multitude de plugins disponibles	Lourdeur de l'IDE pour une utilisation sur le long terme
IntelliJ	Analyse de code intelligente et <i>refactoring</i> avancé	Fonctionnalités et langages limités en Community Edition
NetBeans	Fourni des pilotes pour toutes les bases de données	Non optimisé pour le développement Web
WebStorm	Fonctionnalités poussées en JavaScript, HTML et CSS	Période d'essai pour les fonctionnalités avancées



ÉDITEURS

Éditeurs	Avantages	Inconvénients
Atom	Outil entièrement personnalisable et puissant	Quelques problèmes de performance
Brackets	Aperçu du code en direct	Petite communauté et peu d'extension disponible
Sublime Text	Flexible et rapide notamment pour le développement Web	Licence payante pour les fonctionnalités avancées
Visual Studio Code	Prêt à l'emploi avec notamment le débogueur et Git	Pas autant de fonctionnalités qu'un IDE



GIT

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



TASK RUNNER

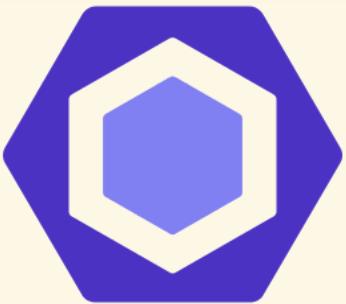
Automatiser vos tâches répétitives
Gulp se concentre sur le code
Grunt nécessite de la configuration

Utiliser pour la compilation, le renommage, la concaténation, le scripting, la minification...



BOWER & YARN

Gérer vos dépendances
Récupération des librairies **JavaScript**
Mises à jour automatiques
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**



TRANSPILEUR

Fusion des projets **6To5** et **ESNext**

Rétrocompatibilité du code **JavaScript**

Support des nouveautés **ES9** et certaines fonctionnalités **ESNext** à venir

S'intègre facilement avec les *Task Runner* et les *Bundler*



webpack

BUNDLER

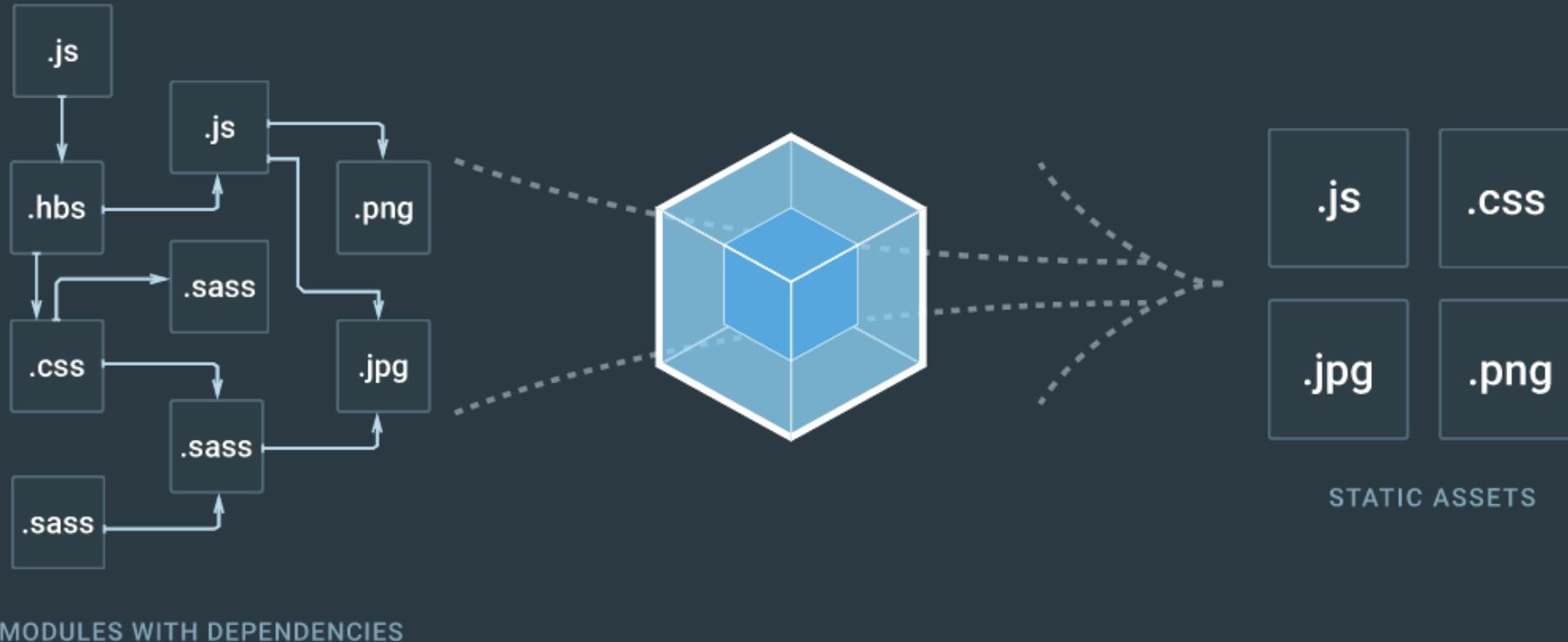
Fonctionnement par modules

Intéropérabilité des fichiers

Appel ~~de fichiers~~ de modules

Découpage par responsabilité

Fini les casses têtes liés aux ressources

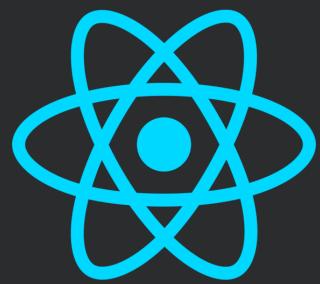




TESTS UNITAIRES

La base de la programmation
Technique liée à l'approche **Agile**
Tester vos méthodes et vos services
Gagner du temps dans vos développements

Tester c'est douter !



REACT

INTRODUCTION

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

Utilisé par **Netflix**, **Deezer**, **Spotify**, **Instagram**, **SoundCloud**...

SYNTAXE

Ci-dessous un exemple de composant **React** avec le fichier HelloComponent.js :

```
import React from "react";

class HelloComponent extends React.Component {
  render() {
    return React.createElement(
      "span",
      null,
      "Hello World !"
    );
  }
}
```

JSX

Préférez la syntaxe JSX, exemple avec le fichier `HelloComponent.jsx` :

```
import React, { Component } from "react";

const example = "Hello World !";

class HelloComponent extends Component {
  render() {
    return (
      <span>{example}</span>
    );
  }
}
```

Le langage JSX nécessite d'être transpilé

INITIALISATION

1^{er} Étape : Écrire son composant

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

HTML

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

    <!-- This Is Where We'll Mount Our App -->
    <div id="app"></div>
  </body>
</html>
```

INDEX.JS

Au début de l'application, dans le script principal...
On appelle **React** afin de rendre le composant dans le DOM !

```
import React from 'react';
import ReactDOM from 'react-dom';

import HelloComponent from './path/to/HelloComponent';

ReactDOM.render(
  <HelloComponent />,
  document.getElementById('app') // Our Mount Point
);
```

STATE & PROPS

COMMENT METTRE À JOUR LE DOM LORSQUE LES DONNÉES SONT MODIFIÉES ?

STATE

L'état est mutable
Données internes d'un composant

Entièrement géré par le composant lui-même

```
class HelloComponent extends Component {
  constructor(props) {
    super(props);

    this.state = {
      counter: 0
    };

    setInterval(() => {
      this.setState({ counter: this.state.counter + 1 });
    }, 1000);
  }

  render() {
    return (
      <span>{this.state.counter}</span>
    );
  }
}
```

PROPS

Les propriétés sont immutables

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

Données transmises d'un composant parent à un enfant

Passer des propriétés comme des attributs d'un élément HTML

COMPOSANT ENFANT

```
export default class ChildComponent extends Component {
  render() {
    return(
      <span>{this.props.counter}</span>
    );
  }
}
```

COMPOSANT PARENT

```
import ChildComponent from './ChildComponent';

class ParentComponent extends Component {
  render() {
    return(
      <ChildComponent counter={1} />
    );
  }
}
```

MODÈLE COURANT

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

DÉCLARATION

```
import ChildComponent from './ChildComponent';

class ParentComponent extends Component {
  constructor(props) {
    super(props);

    this.state = {
      counter: 0
    };

    this.incrementCounter = () => {
      this.setState({ counter: this.state.counter + 1 });
    };
  }

  render() {
    return (
      <ChildComponent
        counter={this.state.counter}
        incrementCounter={this.incrementCounter} />
    );
  }
}
```

MODIFICATION

```
export default class ChildComposant extends Component {
  render() {
    return(
      <div>
        <div>{this.props.counter}</div>

        <button onClick={this.props.incrementCounter}>
          Increment
        </button>
      </div>
    );
  }
}
```

*L'événement sur le bouton va modifier la propriété **counter** en passant par le composant parent*

VALEURS PAR DÉFAUT

`defaultProps` permet de définir les valeurs par défaut des propriétés

```
class HelloComponent extends Component {
  render() {
    return(
      <span>{this.props.counter}</span>
    );
  }
}

HelloComponent.defaultProps = {
  counter: 0
};
```

TYPAGE

propTypes permet de définir le type des valeurs relative aux propriétés

```
import PropTypes from "prop-types";

class HelloComponent extends Component {
  render() {
    return (
      <span>{this.props.counter}</span>
      <button onClick={this.props.incrementCounter}></button>
    );
  }
}

HelloComponent.propTypes = {
  counter: PropTypes.number.isRequired,
  incrementCounter: PropTypes.func
};
```

IMBRICATION

this.props.children permet d'accéder aux éléments imbriqués

COMPOSANT PARENT

```
import { ChildComponent } from './ChildComponent';

class ParentComponent extends Component {
  render() {
    return(
      <ChildComponent>
        <div>World</div>
      </ChildComponent>
    );
  }
}
```

COMPOSANT ENFANT

```
export const ChildComponent = ({ children }) => (
  <div>
    <div>Hello</div>
    {children}
  </div>
);
```

AUTRE EXEMPLE

```
class HelloComponent extends Component {
  constructor(props) {
    super(props);

    this.state = {
      list: [
        { id: 1, label: "Awesome" },
        { id: 2, label: "React" },
        { id: 3, label: "Tutorial" }
      ]
    };
  }

  render() {
    return (
      <ul>
        {this.state.list.map(element => (
          <li key={element.id}>{element.label}</li>
        ))}
      </ul>
    );
  }
}
```

*Ne pas oublier d'identifier chaque élément lors d'un rendu dynamique grâce à **key***

SYNTAXE STATELESS

Les composants n'incluant que la fonction `render()` peuvent s'écrire autrement

```
import { number, func, arrayOf, string } from "prop-types";

const styleSheet = {
  marginTop: '20px',
  paddingLeft: '10px'
};

// Component
const HelloComponent = ({ counter, incrementCounter }) => (
  <div className="stateless">
    <span>{counter}</span>
    <button style={styleSheet} onClick={incrementCounter}></button>
  </div>
);

HelloComponent.propTypes = {
  counter: number.isRequired,
  incrementCounter: func,
  anotherStuff: arrayOf(string)
};

export default HelloComponent;
```

CYCLE DE VIE

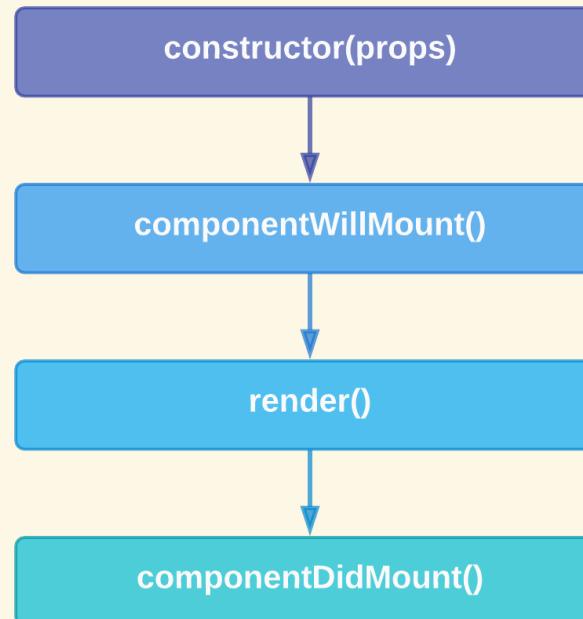
EXPLICATION

Gérer le comportement de votre composant lorsqu'il est monté, mis à jour, démonté...
Les méthodes suivantes permettent de gérer le cycle de vie d'un composant **React** :

- `componentWillMount()`
- `componentDidMount()`
- `componentWillReceiveProps(nextProps)`
- `shouldComponentUpdate(nextProps, nextState)`
- `componentWillUpdate(nextProps, nextState)`
- `componentDidUpdate(prevProps, prevState)`
- `componentWillUnmount()`

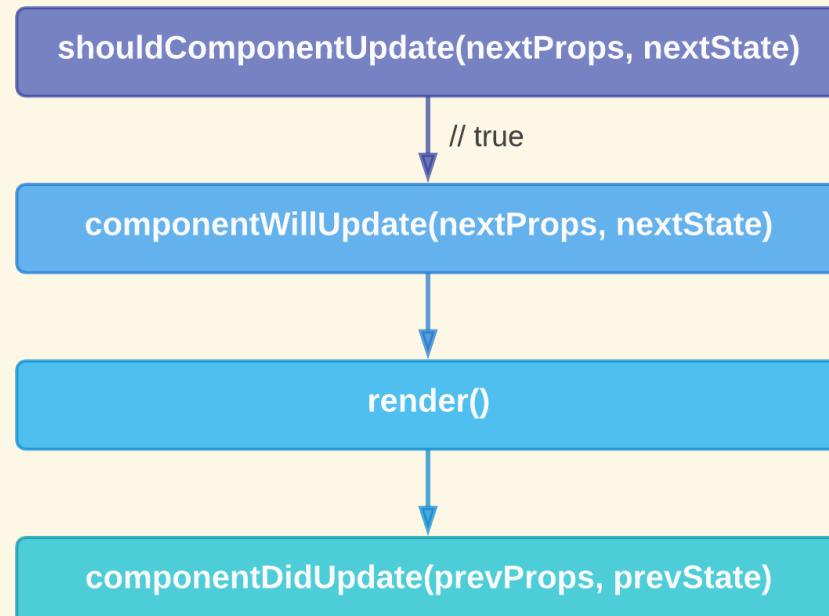
RENDER

Ces méthodes sont appelées lorsque le composant est monté via `ReactDOM.render()`



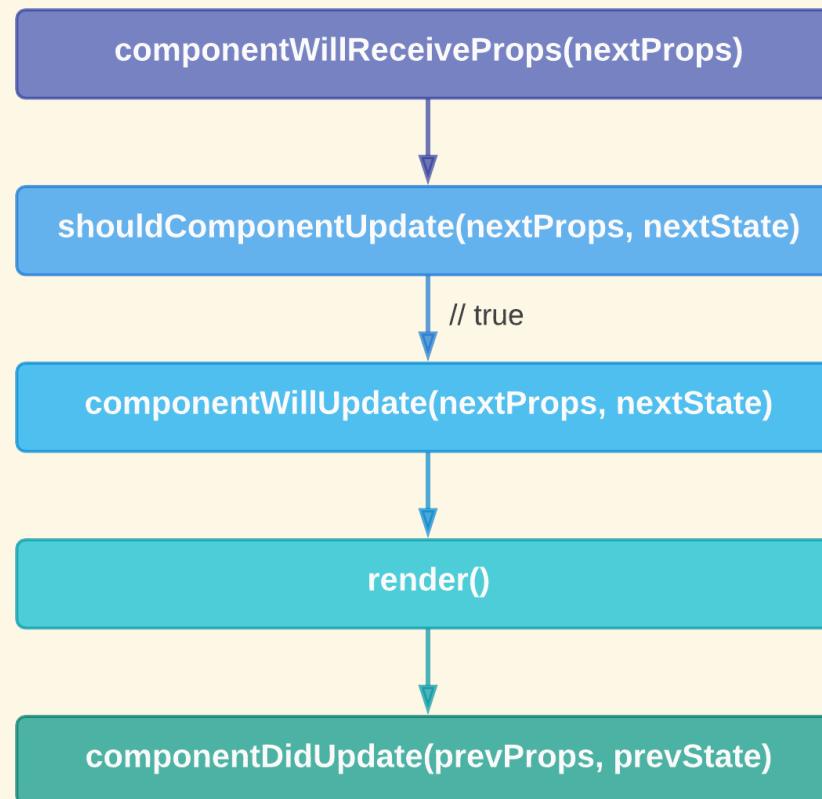
STATE

Ces méthodes sont appelées par `this.setState()`



PROPS

Ces méthodes sont appelées même s'il n'y a pas de changement des props



EXEMPLE

```
class HelloComponent extends Component {  
  componentDidMount() {  
    console.log(this.props.label);  
  }  
  
  shouldComponentUpdate(nextProps, nextState) {  
    return nextProps.label !== this.props.label;  
    // Return True By Default  
  }  
  
  render() {  
    return(  
      <span>{this.props.label}</span>  
    );  
  }  
}
```

Ceci permet de ne pas (re)rendre le composant lorsque les props ne changent pas

PURE COMPONENTS

Composants qui sont (re)rendus uniquement si leurs props ou state changent

1. Implémenter `shouldComponentUpdate` dans chaque composant
2. Étendre son composant de **React.PureComponent**

```
import React, { PureComponent } from "react";

class HelloComponent extends PureComponent {
  /*
  shouldComponentUpdate(nextProps, nextState) {
    return nextProps !== this.props || nextState !== this.state;
  }
}

render() {
  // U're Awesome Code...
}
}
```

CONTEXT

COMMENT PASSER UNE PROPRIÉTÉ À UN COMPOSANT IMBRIQUÉ ?

EXPLICATION

Passer des props entre des composants à plusieurs niveaux peut paraître compliqué
React Context permet d'enregistrer une donnée dans `this.context`

- `constructor(props, context)`
- `componentWillReceiveProps(nextProps, nextContext)`
- `shouldComponentUpdate(nextProps, nextState, nextContext)`
- `componentWillUpdate(nextProps, nextState, nextContext)`
- `componentDidUpdate(prevProps, prevState, prevContext)`

EXEMPLE

Ajout du label dans le context

```
class ParentComponent extends Component {
  static childContextTypes = {
    label: PropTypes.string
  };

  getChildContext() {
    return {
      label: "Hello World"
    };
  }

  /* ... */
}
```

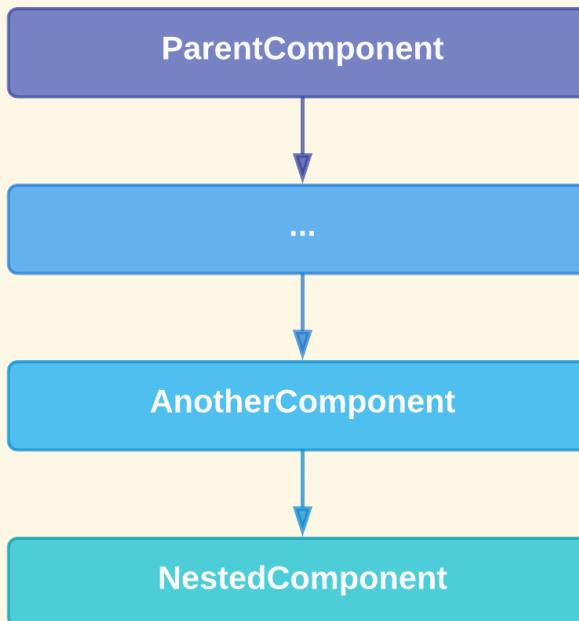
Récupération du label dans le context

```
class NestedComponent extends Component {
  static contextTypes = {
    label: PropTypes.string
  };

  render() {
    return(
      <span>{this.context.label}</span>
    );
  }
}
```

ATTENTION

Ajout du label dans le **context**



Mise à jour du **context**

`shouldComponentUpdate() =>
false`

Récupération du label dans le
context

context non mis à jour

ARCHITECTURES PROJETS

BASIQUE

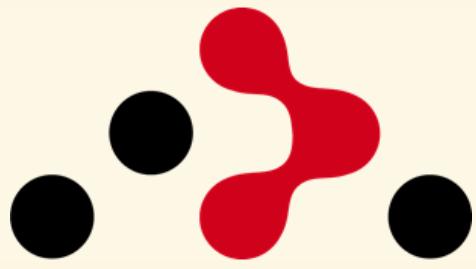
- **public/**
 - favicon.png
 - index.html
- **src/**
 - App.jsx
 - ChildComponent.jsx
 - index.js
 - ParentComponent.jsx
 - registerServiceWorker.js
 - styles.css

TECHNIQUE

- **actions/**
 - index.js
- **components/**
 - App.jsx
 - Body.jsx
 - ChildComponent.jsx
 - Footer.jsx
 - Header.jsx
 - ParentComponent.jsx
- **constants/**
 - index.js
- **reducers/**
 - index.js
 - parent.js
 - child.js
- **services/**
 - api.js
 - legacy.js
- index.js
- registerServiceWorker.js
- styles.css

FONCTIONNELLE

- **components/**
 - App.jsx
- **Body**
 - Body.jsx
 - ChildComponent.jsx
 - ParentComponent.jsx
- Footer.jsx
- Header.jsx
- **redux/**
 - store.js
- **child/**
 - actions.js
 - constants.js
 - reducer.js
- **parent/**
 - actions.js
 - constants.js
 - reducer.js
- **services/**
 - api.js
 - legacy.js
- index.js
- registerServiceWorker.js
- styles.css



REACT ROUTER

INTÉRÊT

Affichage des composants en fonction de l'URL

Inspirée par le Router du framework Ember

Créée par [@ryanflorence](#) & [@mjackson](#)

Facilement implantable avec Redux

Importance de la navigation par URL

ROUTING

Définition des chemins pour chaque composants de l'application

```
import React from 'react';
import { Switch, Route } from 'react-router-dom';
import Hello from './Hello';
import Login from './Login';
import Contact from './Contact';

const Router = () => (
  <Switch>
    <Route exact path="/" component={Hello} />
    <Route path="/login" component={Login} />
    <Route path="/contact/:name" render={({ match }) => <Contact name={match.params.name} />} />
  </Switch>
);

export default Router;
```

INTÉGRATION

Rendu du composant servant de **router** à partir du composant principal

```
import React, { Fragment } from 'react';
import NavBar from './NavBar';
import Router from './Router';

const App = () => (
  <Fragment>
    <NavBar />
    <Router />
  </Fragment>
);

export default App;
```

Activation de la navigation par **routes** dans le point d'entrée de l'application

```
import React from 'react';
import { render } from 'react-dom';
import { BrowserRouter } from 'react-router-dom';
import App from './App';

render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.getElementById('app')
);
```

NAVIGATION

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

1. Dynamiquement via la propriété `history` étendue par `withRouter`
2. Grâce au composant `Link` au sein de la fonction `render()`

```
import React, { Component } from 'react';
import { Link, withRouter } from 'react-router-dom';

class NavBar extends Component {
  goToLogin = () => {
    this.props.history.push('/login');
  };

  render() {
    return (
      <nav>
        <button onClick={this.goToLogin}>
          Login
        </button>
        <button>
          <Link to="/contact">Contact</Link>
        </button>
      </nav>
    );
  }
}

export default withRouter(NavBar);
```

FLUX

CONCEPT

Architecture pour les flux de données unidirectionnels

Il existe plusieurs implémentations de **Flux en JavaScript** :

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

...



PRINCIPE

Gestion du state des composants simplifiée

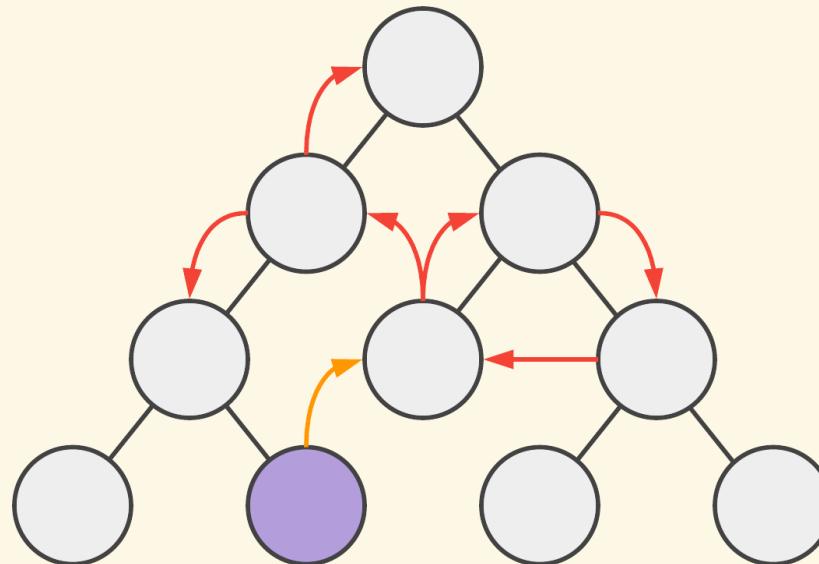
Le **store** est accessible à tout moment

Les composants enfants n'héritent pas du state des parents

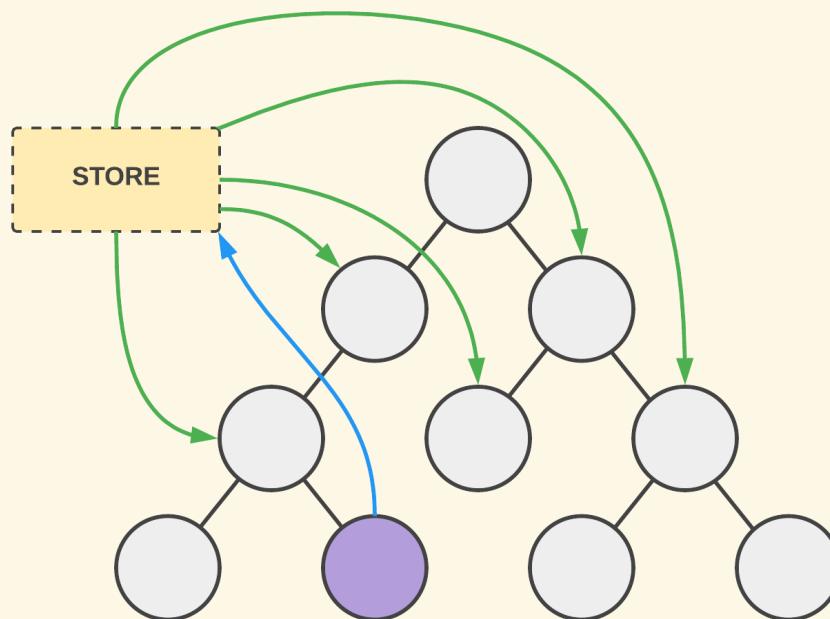
Aucune donnée en doublon (un seul point de vérité)

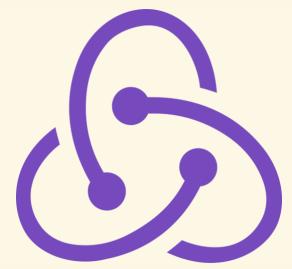
Meilleure conception de votre application

SANS REDUX



AVEC REDUX





REDUX

PRINCIPE

Redux est un gestionnaire d'état pour les applications

Permet de suivre l'état global de l'applicatif

Peut être utilisé dans n'importe quel framework **JavaScript**

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

Contient les `reducers` de l'applicatif

Le **store** est en lecture seul

ACTIONS

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

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

Le *type* est la référence utilisée par le `reducer`

```
// Constants
export const CREATE_TASK = 'TASK/CREATE_TASK';
export const COMPLETE_TASK = 'TASK/COMPLETE_TASK';

// Actions
const createTask = task => ({ type: CREATE_TASK, task });
const completeTask = task => ({ type: COMPLETE_TASK, task });

// Object
const task = {
  label: 'Learn Redux',
  completed: false
};

// Dispatch To Reducer
store.dispatch(createTask(task));
```

REDUCERS

L'*action* trace le changement dans le **store**
Le `reducer` se charge d'effectuer la modification

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

/* Initial State */
export default function tasks (state = [], action) {
  switch (action.type) {
    case CREATE_TASK:
      return [
        ...state,
        action.task
      ];

    case COMPLETE_TASK:
      return state.map(task => task.label === action.label ? { ...task, completed: !task.completed } : task);

    default:
      return state;
  }
}
```

STORE

Ne pas oublier d'exposer ses reducers afin d'y avoir accès dans votre application

```
import { combineReducers, createStore, applyMiddleware } from 'redux';
import thunk from 'redux-thunk';
import tasks from './tasks';

const reducers = combineReducers({
  tasks
});

const store = createStore(reducers, applyMiddleware(thunk));

export default store;
```

INITIALISATION

Chargement du **store** dans le point d'entrée de notre application : `index.js`

```
import React from 'react';
import { render } from 'react-dom';
import { Provider } from 'react-redux';
import App from './components/App';
import store from './redux/store';
import './assets/styles.css';

render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('app')
);
```

UTILISATION

React Redux permet de se connecter au **store** dans un composant
Cette connexion est indispensable pour pouvoir dispatcher des *actions*

```
import React, { Component } from 'react';
import { connect } from 'react-redux';
import { completeTask } from './actions';

class HelloComponent extends Component {
  updateTask = task => {
    this.props.completeTask(task);
  };

  render() {
    return (
      <ul>
        {this.props.tasks.map(task => (
          <li>{task.label}</li>
          <button onClick={this.updateTask(task)}>Update</button>
        ))}
      </ul>
    );
  }
}

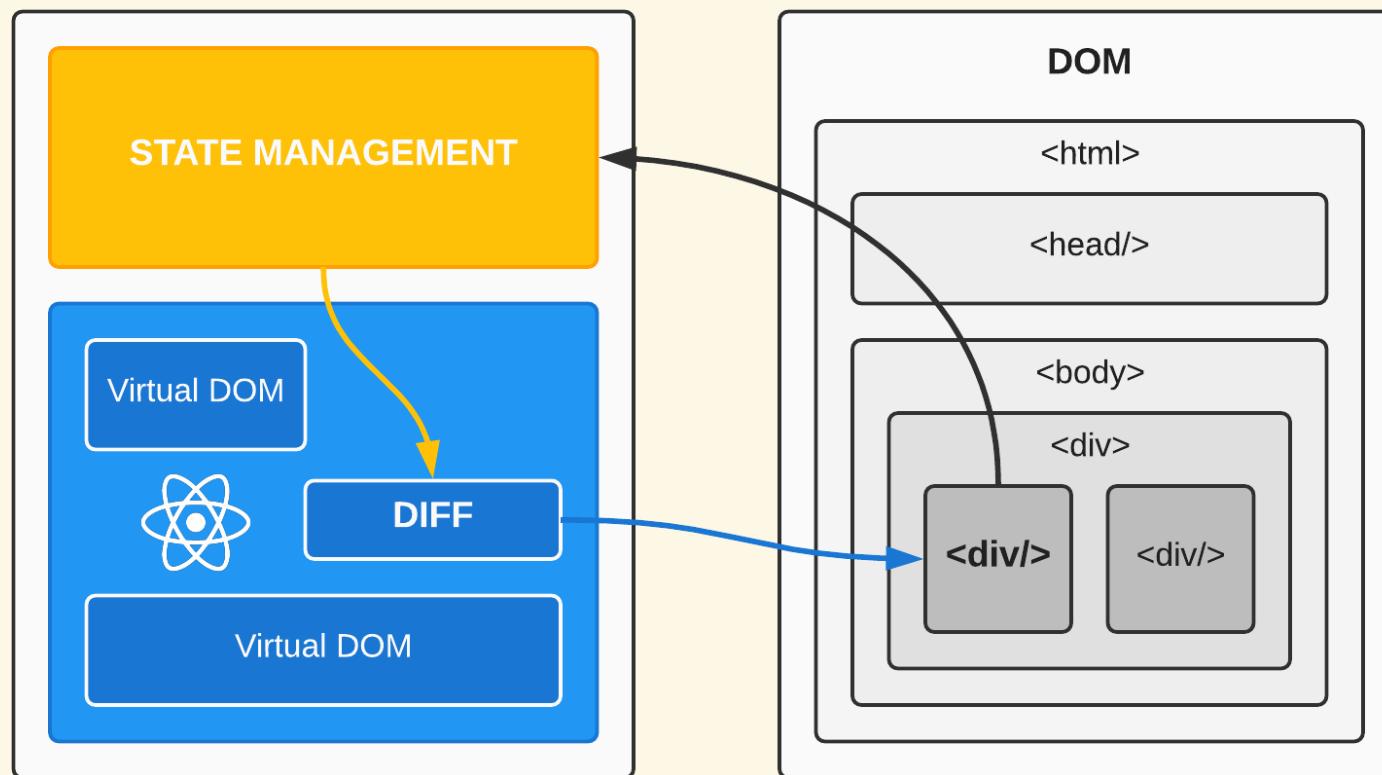
const mapStateToProps = state => ({
  tasks: state.tasks
});

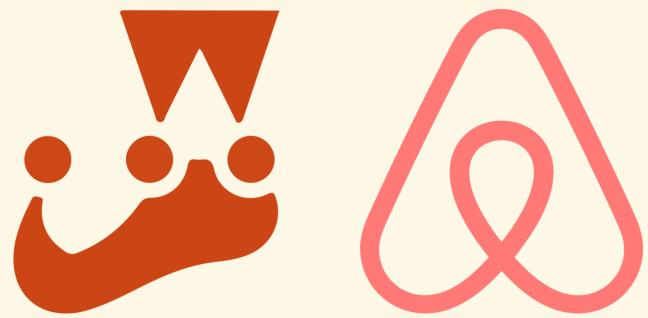
const mapDispatchToProps = dispatch => ({
  completeTask: () => dispatch(completeTask())
});

export default connect(mapStateToProps, mapDispatchToProps)(HelloComponent);
```

DOM VIRTUEL

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





JEST & ENZYME

CONFIGURATION

Récupération des librairies utilitaires :

```
npm install --save enzyme enzyme-adapter-react-16
```

Initialisation du fichier `setupTests.js` :

```
import Enzyme from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';

Enzyme.configure({ adapter: new Adapter() });
```

UTILISATION

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

```
import React from 'react';
import { shallow, mount } from 'enzyme';
import MyComponent from './MyComponent';

describe('MyComponent Component', () => {
  let wrapper;

  beforeEach(() => {
    wrapper = mount(
      <MyComponent>
        Content
      </MyComponent>
    );
  });

  it('Renders', () => {
    expect(wrapper.exists()).toBe(true);
  });

 xit('Create Snapshot', () => {
  expect(wrapper).toMatchSnapshot();
});

  it('Check Props', () => {
    wrapper.setProps({ active: true, label: 'Hello World' });

    expect(wrapper.props().active).toBeTruthy();
    /* expect(wrapper.props().active).toBe(true); */
    expect(wrapper.props().children).toEqual('Content');
    expect(wrapper.props().label.trim()).toHaveLength(10);
  });
});
```

VUE D'ENSEMBLE

FRAMEWORKS



ANGULAR JS

Créé en 2009 par **Google**

Populaire parmi les développeurs

Outils nombreux et adaptés au travail en équipe

Idéal pour les solutions *Quick & Dirty*

Mature pour le milieu professionnel

Paquets de base maintenus par **Google**



ANGULAR

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



VUE

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



EMBER

Créé en 2011 par **Yehuda Katz**

Framework très simple à apprendre

Communauté de développeurs active

Développement de fonctionnalités constant

Vitesse de chargement et d'exécution rapide

Routage puissant mais complexe à manipuler



METEOR

Développé depuis 2012 par **Meteor Dev Group**
Librairie **JavaScript** orientée *Full-Stack*
Idéal pour les applications petites et réactives
Intégration native de **Apache Cordova**
Rendu des données directement dans le navigateur
Communauté de développeurs constante



POLYMER

Développé en 2013 par **Google**

Réutilisation massive de composants Web

Liaison de données bidimensionnelle et bidirectionnelle

Réduit l'écart entre les développeurs et les concepteurs

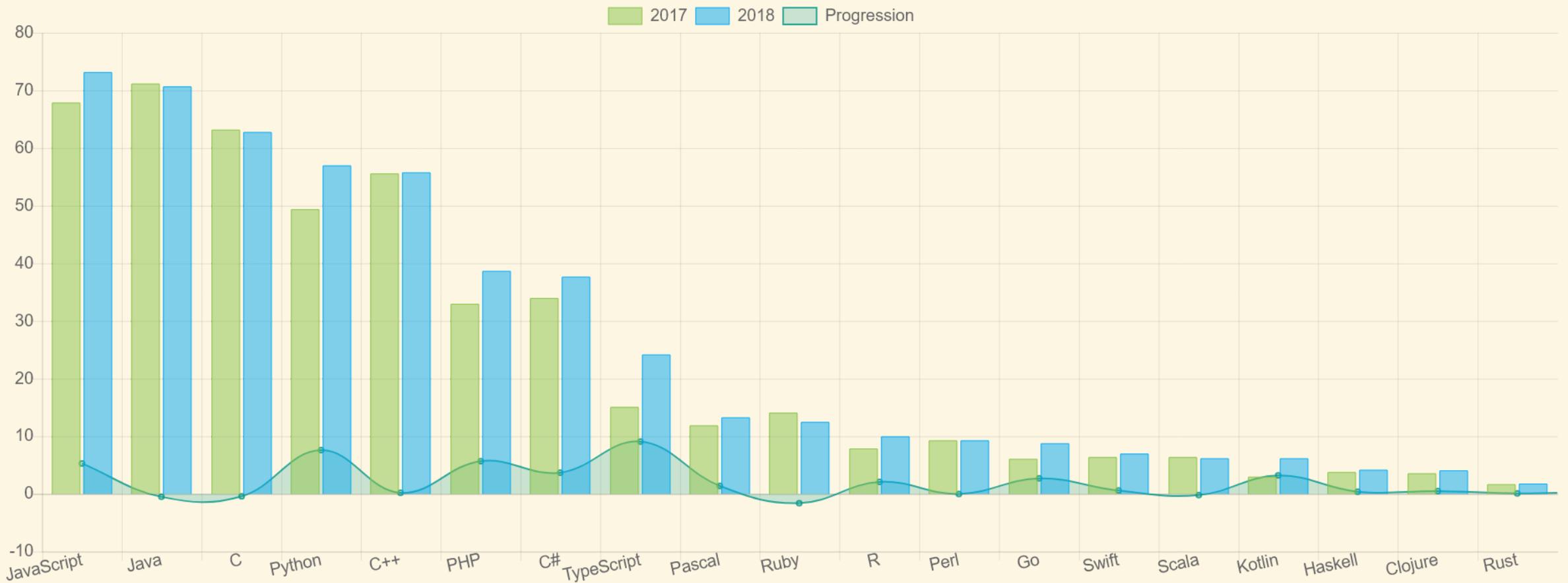
Idéal pour les applications riches en fonctionnalités

Facilite le référencement naturel

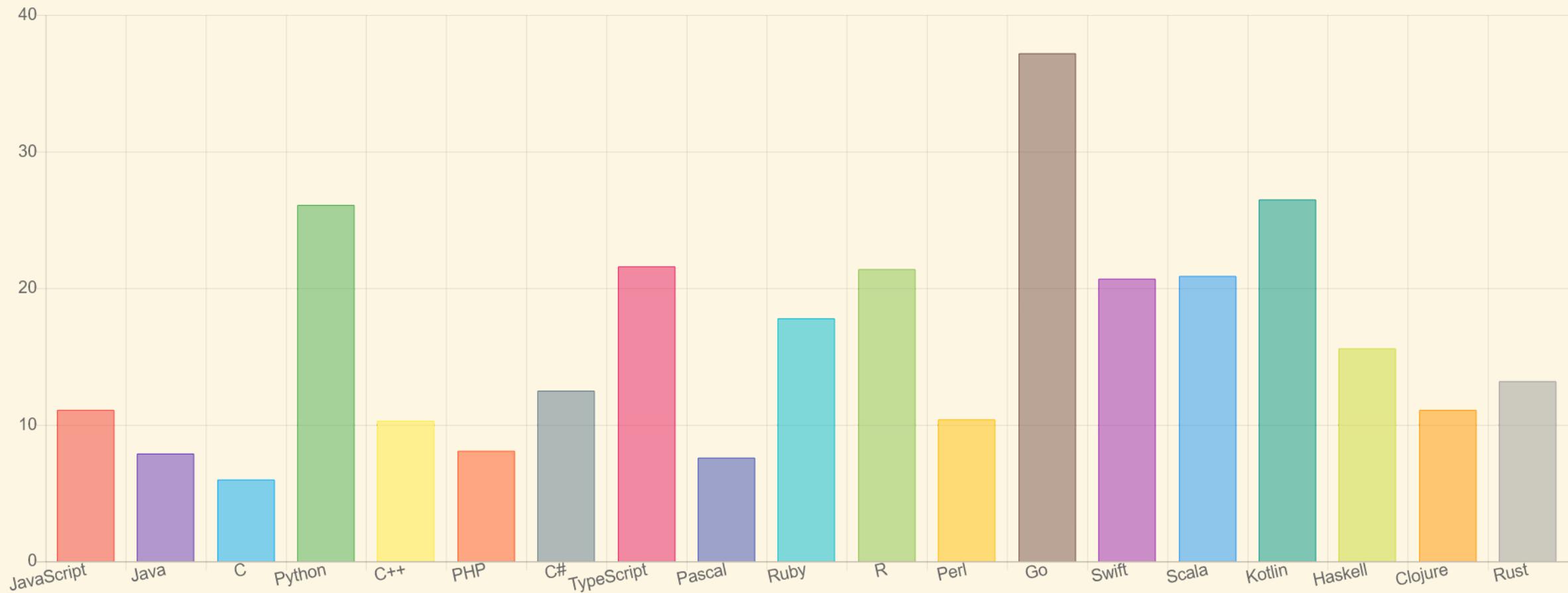
INFOGRAPHIE

LANGAGES

CONNAISSANCES EN 2017 VS. 2018

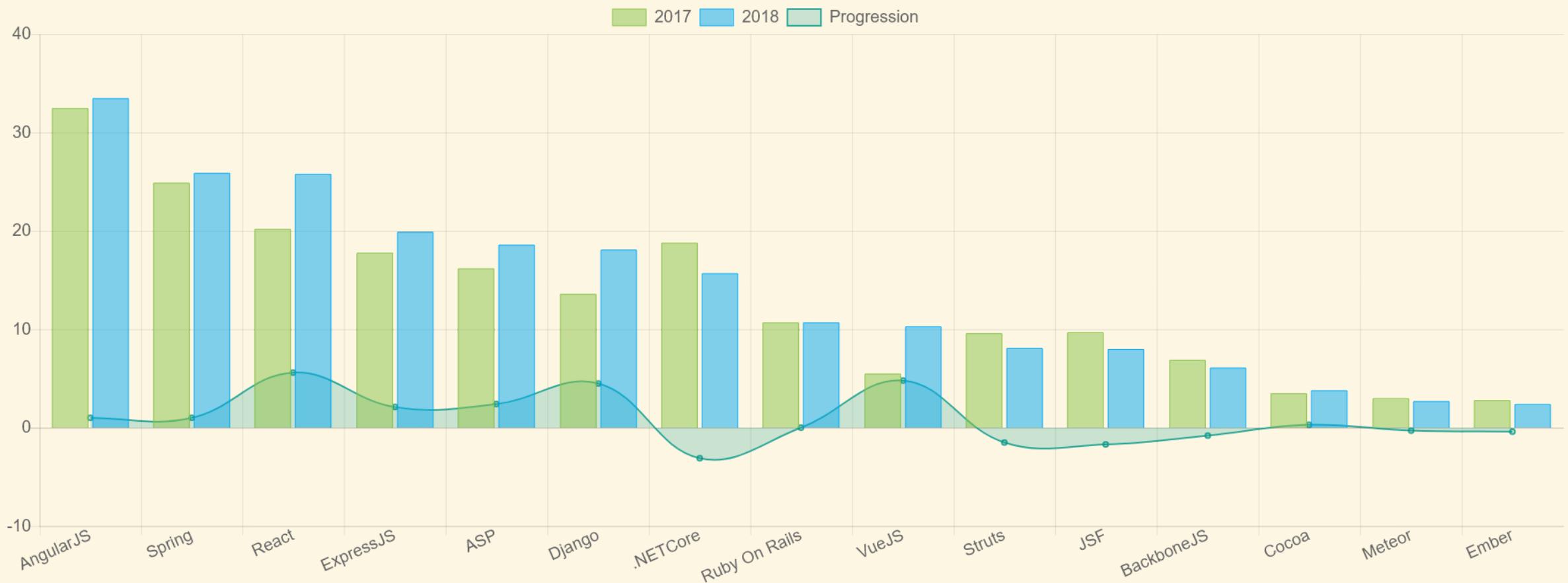


QUELS FRAMEWORKS LES DÉVELOPPEURS PRÉVOIENT-ILS D'APPRENDRE ?

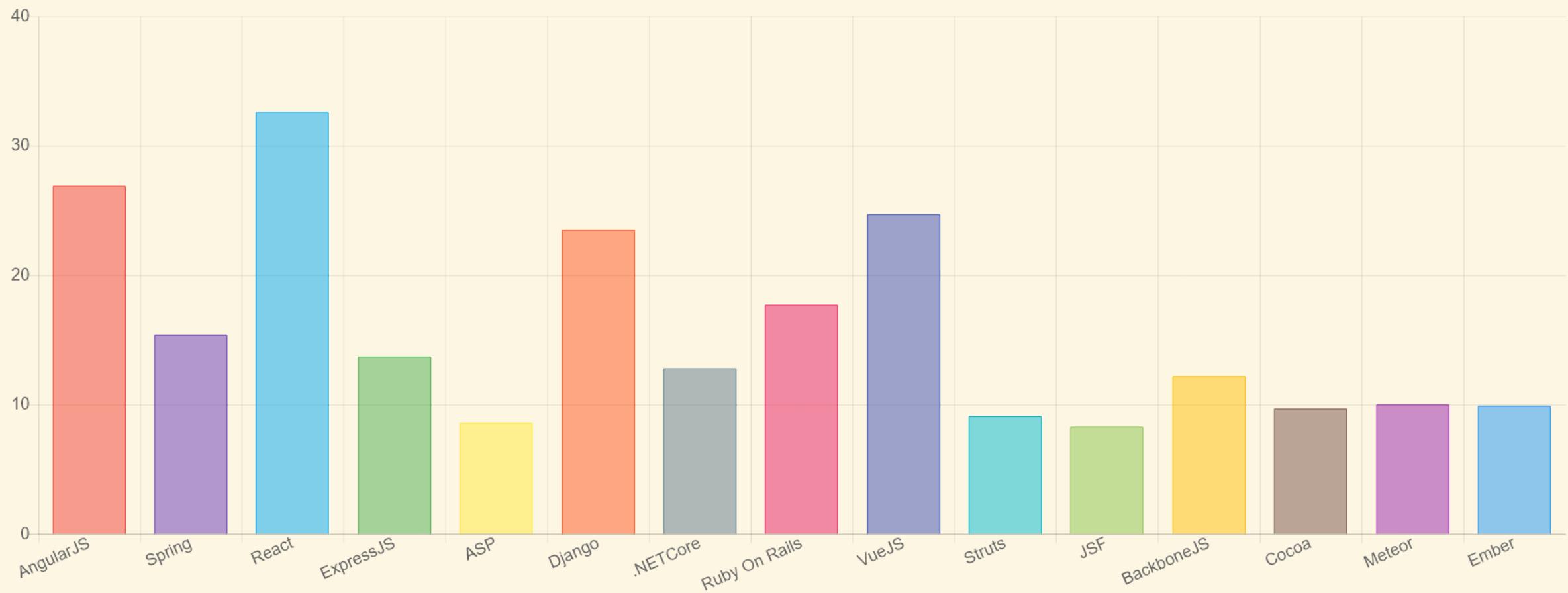


FRAMEWORK

CONNAISSANCES EN 2017 VS. 2018

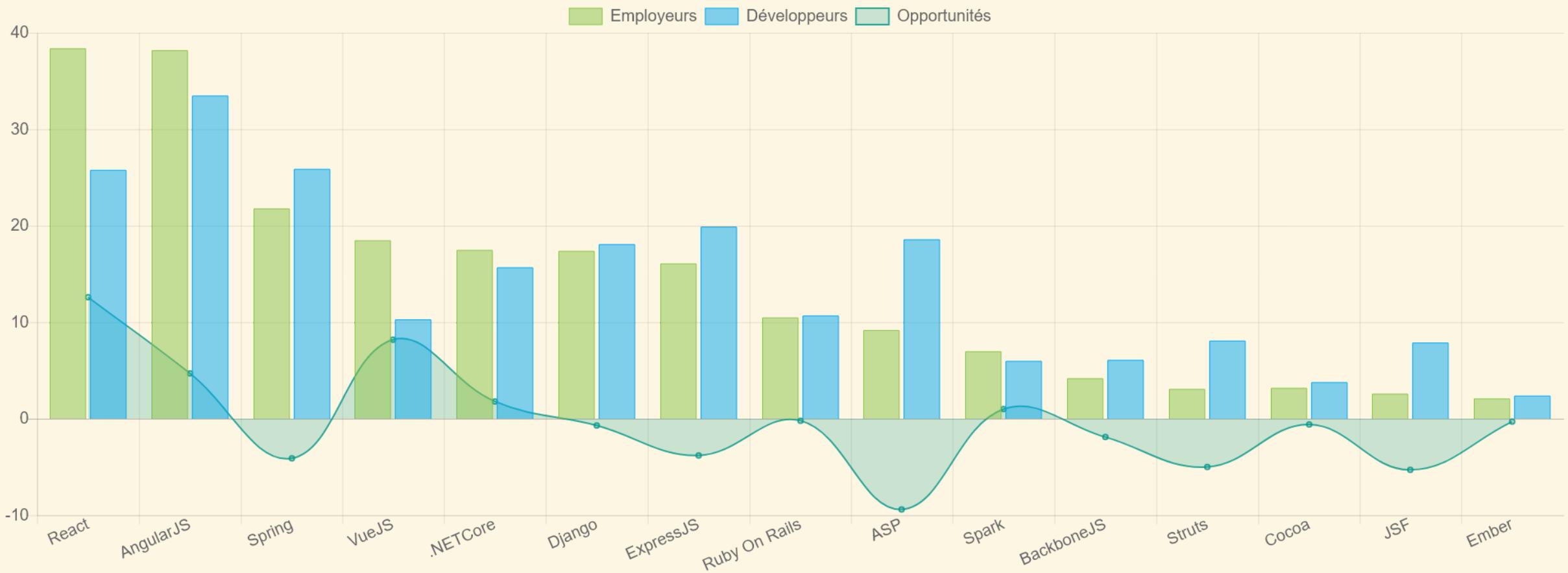


QUELS FRAMEWORKS LES DÉVELOPPEURS PRÉVOIENT-ILS D'APPRENDRE ?



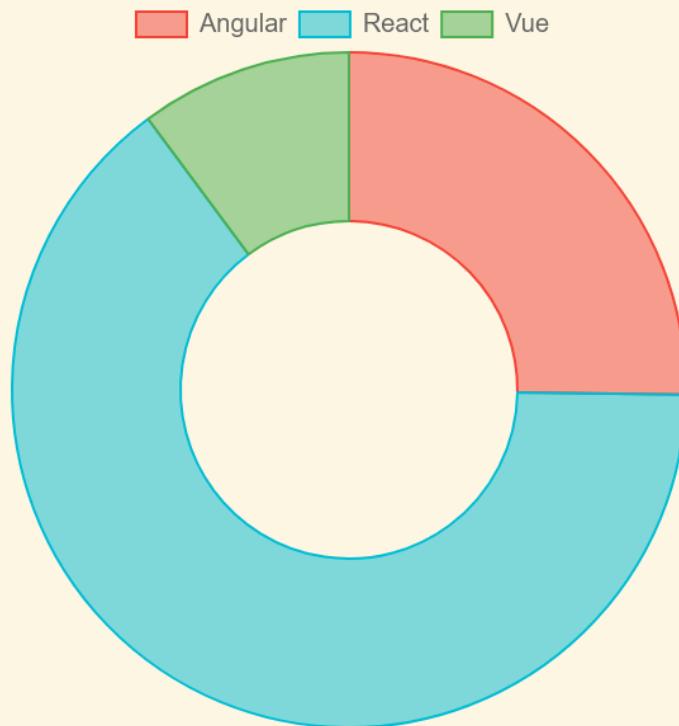
FRAMEWORKS

BESOIN DES EMPLOYEURS VS. CONNAISSANCES DES DÉVELOPPEURS



TÉLÉCHARGEMENTS NPM*

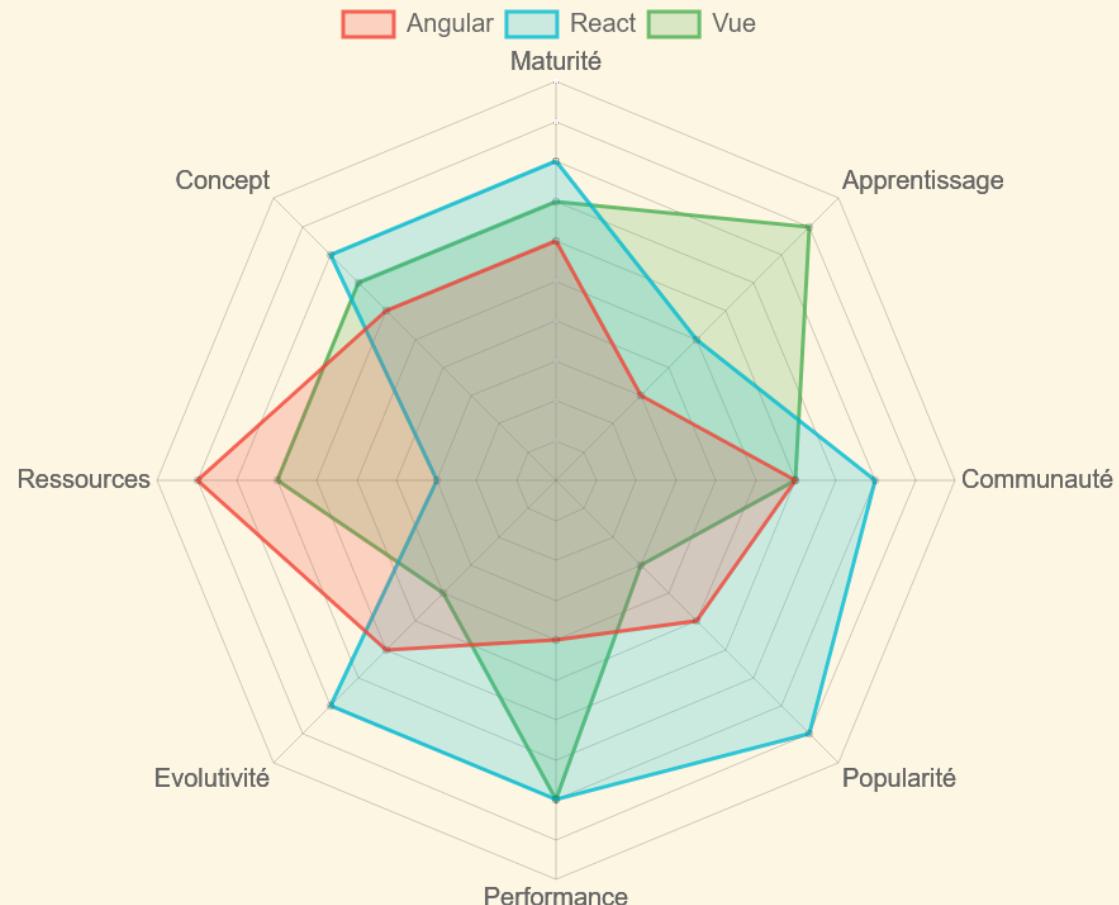
ANGULAR VS. REACT VS. VUE



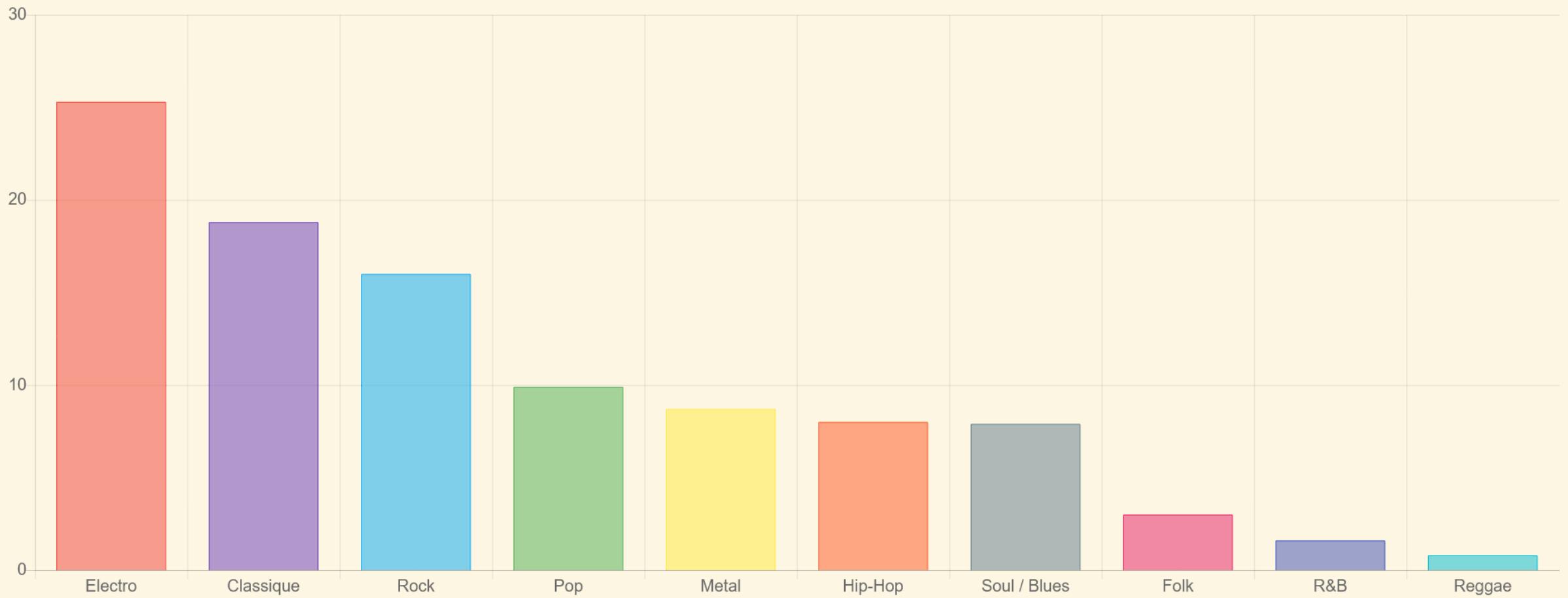
* 20 Janvier 2019

COMPARATIF

ANGULAR VS. REACT VS. VUE



QUEL STYLE DE MUSIQUE ÉCOUTER LORSQU'ON CODE ?



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
React	https://reactjs.org/
React Router	https://reacttraining.com/react-router/
Redux	https://redux.js.org/
HackerRank	https://research.hackerrank.com/developer-skills/2019/
Medium	https://medium.com/
MDN	https://developer.mozilla.org/fr/
MrDoomy	https://gitlab.com/mrdoomy/
RevealJS	https://revealjs.com/