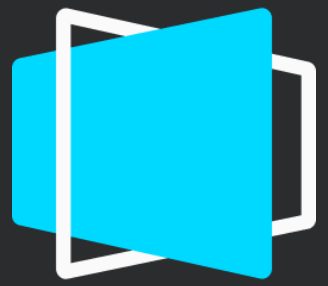


React 16.13.0

A **JavaScript** Library For Building UI

Tour De Table

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

Présentation

Damien Chazoule

FullStack JS Developer
React Lover (**VueJS** Too)


@MrDoomy

Sommaire

Rappels

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

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. React Router
11. Flux
12. Redux
13. DOM Virtuel
14. Jest & Testing Library

Vue d'Ensemble

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

Rappels

JavaScript

Créé part **Brendan Eich** en 1995
Alliance entre **Sun** et **Netscape**
Langage faiblemment typé
Programmation fonctionnelle

Volonté d'un Web plus dynamique

EcmaScript

By Ecma International

Créé en 1996
Standartisation 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)*

Outils

1er 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** : 12.16.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 défault	~~Pas autant de fonctionnalités qu'un IDE~~
WebStorm	Fonctionnalités inédites avec notamment le *Running Coverage*	Modèle économique payant dégressif

Git

L'indispensable gestionnaire de versions
Meilleure visualisation de l'avancement du projet
Développement de fontionnalités 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

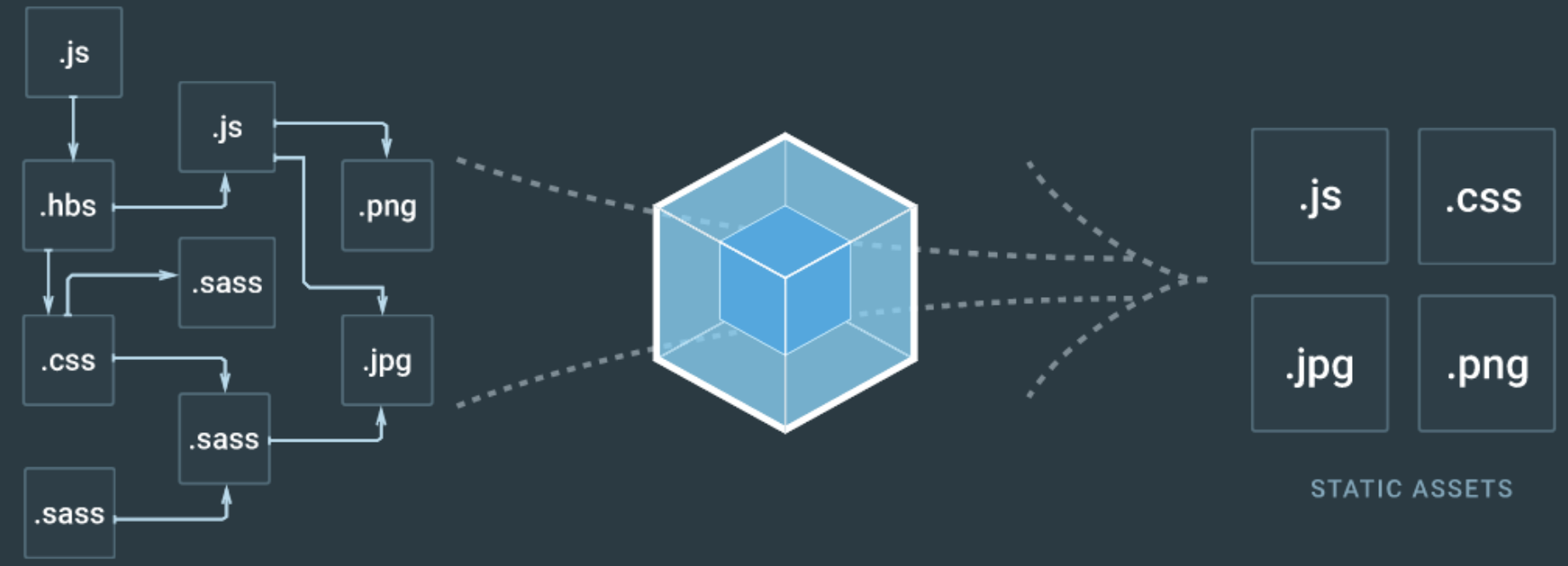
CSS

Des feuilles qui ont du style
Du *design* pour vôtre 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`)


.js
.hbs
.js
.png
.sass
.css
.jpg
.sass
.sass
.js
.css
.jpg
.png
STATIC ASSETS
MODULES WITH DEPENDENCIES

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 fourtyTwo = 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 fourtyTwo = 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 = ['Electro', 'Downtempo'];

console.log(album['genre'][0]); // Affiche 'Electro' dans la console
```

Fonctions

Voici comment déclarer une fonction en **JavaScript** :

ES5

```
var tab = [3, 5, 7, 9];

/* Assignation */
var isSumEven = function(array) {
  var sum = 0;

  for (var i = 0; i < array.length; i++) {
    sum = sum + array[i];
  }

  return sum % 2 === 0;
}

var result = isSumEven(tab);

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

log(result); /* Result: true */
```

ES6+

```
const tab = [3, 5, 7, 9];

/* Assignation */
const isSumEven = array => {
  let sum = 0;

  for (let i = 0; i < array.length; i++) {
    sum += array[i];
  }

  return sum % 2 === 0;
};

const result = isSumEven(tab);

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

log(tab); /* Result: true */
```

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

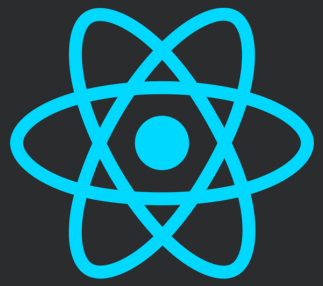
Manière de faire de l'héritage en **Javascript**
C'est l'équivalent d'une classe en POO
Une sorte de patron qu'un objet peut utiliser

ES5

```
var Cat = function(name, color, length = 0) {
  Animal.call(this, name, color);
  this.length = length;
}

Cat.prototype.meow = function() {
  for (var i = 0; i < this.length; i++) {
    console.log('Meow!');
  }
}

var myCat = new Cat('Mocha', 'B&W', 5);

myCat.toString();

// Affichage 'Meow!' x5 dans la console
myCat.meow();
```

ES6+

```
class Cat extends Animal {
  constructor(name, color, length = 0) {
    super(name, color);
    this.length = length;
  }

  meow() {
    for (let i = 0; i < this.length; i++) {
      console.log('Meow!');
    }
  }
}

const myCat = new Cat('Mocha', 'B&W', 5);

myCat.toString();

// Affichage 'Meow!' x5 dans la console
myCat.meow();
```



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...

Command Line Interface

L'indispensable `create-react-app`

Outil CLI **React** (officiel)

Projet prêt à l'emploi grâce à `react-scripts`

Support de **TypeScript** et de **SASS / SCSS**

Autorise l'éjection pour la personnalisation Webpack

```
npx create-react-app my-project
```

Syntaxe

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

```
import React from 'react';

const userName = 'Rick';

class HelloWorld extends React.Component {
  render() {
    return React.createElement(
      'span',
      {
        style: {
          color: '#2a2c2e',
          textDecoration: 'underline'
        }
      },
      `Hi ${userName} !`
    );
  }
}
```

JSX

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

```jsx
import React, { Component } from 'react';

const userName = 'Rick';

class HelloWorld extends Component {
  render() {
    return (
      <span style={{ color: '#2a2c2e', textDecoration: 'underline' }}>
        Hi {userName} !
      </span>
    );
  }
}
```

Le langage JSX nécessite d'être transpilé

Initialisation

~~1er Étape : Écrire son composant~~

2ème Étape : Monter son composant dans le DOM

HTML

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

    <!-- This Is Where We'll Mount Our App -->
    <div id="root"></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 HelloWorld from './path/to/HelloWorld';

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

State & Props

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

State

L'état est mutable
Données internes au composant
Entièrement géré par le composant lui-même

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

    this.state = {
      counter: 0
    };

    this.increment = this.increment.bind(this);
  }

  increment() {
    this.setState({ counter: this.state.counter + 1 });
  }

  render() {
    return (
      <div>
        <span>{this.state.counter}</span>
        <button onClick={this.increment}>+1</button>
      </div>
    );
  }
}
```

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 élement HTML

Enfant

```
class Child extends Component {
  render() {
    return (
      <div>
        <h3>{this.props.label}<h3>
        <span>{this.props.counter}</span>
      </div>
    );
  }
}

export default Child;
```

Parent

```
import Child from './Child';

class Parent extends Component {
  state = {
    counter: 42
  };

  render() {
    return (
      <Child
        label="Compteur"
        counter={this.state.counter} />
    );
  }
}
```

Typage

Définir le type des propriétés avec `propTypes`

Imposer des valeurs par défaut avec `defaultProps`

```
import { number, func } from 'prop-types';

class Counter extends Component {
  render() {
    return (
      <div>
        <span>{this.props.counter}</span>
        <button onClick={this.props.increment}>+1</button>
      </div>
    );
  }
}

Counter.defaultProps = {
  counter: 0
};

Counter.propTypes = {
  counter: number,
  increment: func.isRequired
};
```

Imbrication

`this.props.children` permet d'accéder aux élements imbriqués

Parent

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

class Parent extends Component {
  render() {
    return (
      <Child>
        <span>World</span>
      </Child>
    );
  }
}
```

Enfant

```
export const Child(props) {
  const { children } = props;

  return (
    <div>
      <h1>Hello</h1>
      {children}
    </div>
  );
}
```

For Each

Construction dynamique de nœud DOM

```
class Listing extends Component {
  render() {
    return (
      <ul>
        {this.props.myList.map((item, idx) => (
          <li key={idx}>{item}</li>
        ))}
      </ul>
    );
  }
}

Listing.defaultProps = {
  myList: [
    'Awesome',
    'React',
    'Course'
  ]
};

export default Listing;
```

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

Modèle Courant

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

Parent

```
import Child from './Child';

class Parent extends Component {
  state = {
    counter: 0
  };

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

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

Enfant

```
class Child extends Component {
  render() {
    return (
      <div>
        <h1>{this.props.counter}</h1>
        <button onClick={this.props.increment}>
          +1
        </button>
      </div>
    );
  }
}

export default Child;
```

Render Props

Ajouter des propriétés grâce aux **H**igher-**O**rder **C**omponents (HOC)

```
export default class Counter extends Component {
  state = {
    value: 0
  };

  increment = () => {
    this.setState({ value: this.state.value + 1 });
  };

  render() {
    const { value } = this.state;

    return (
      <div>
        <h1>Compteur</h1>
        {this.props.render({
          value,
          increment: this.increment,
          decrement: val => this.setState({
            value: value - val
          })
        })}
      </div>
    );
  }
}
```

```
import Counter from './Counter';

class ButtonGroup extends Component {
  render() {
    return (
      <Counter render={props => (
        <>
          <button onClick={props.increment}>
            {props.value} +1
          </button>
          <button
            onClick={() => props.decrement(2)}>
            {props.value} -2
          </button>
        </>
      )} />
    );
  }
}
```

Syntaxe Stateless

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

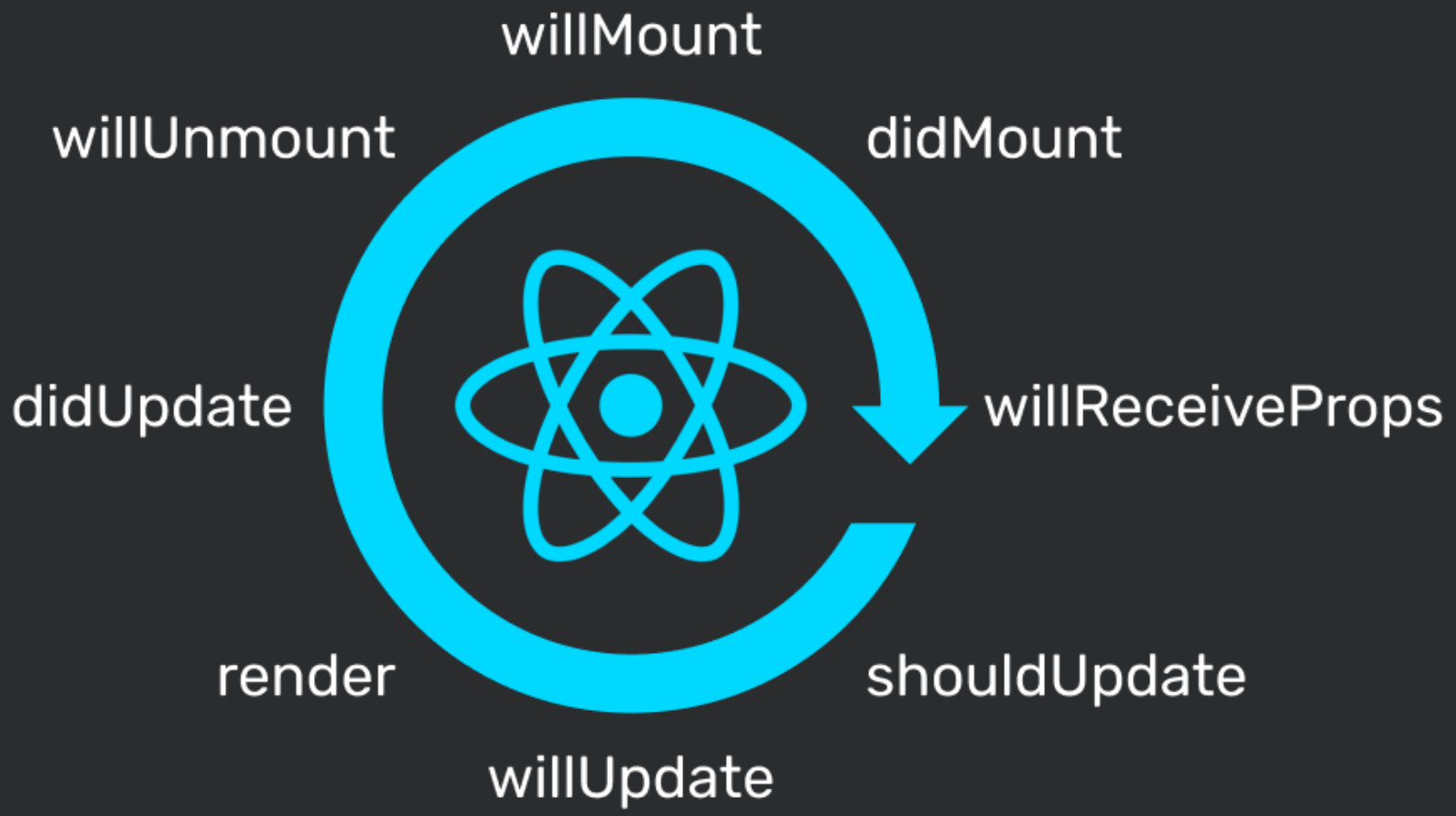
```
import { arrayOf, string, func } from 'prop-types';

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

const ToDo = ({ tasks, randomTask }) => (
  <div className="stateless">
    <ul>
      {tasks.map((task, idx) => <li key={idx}>{task}</li>)}
    </ul>
    <button style={marginAndPadding} onClick={randomTask}>
      Add Random Task
    </button>
  </div>
);

HelloWorld.defaultProps = {
  tasks: []
};

HelloWorld.propTypes = {
  tasks: arrayOf(string),
  randomTask: func.isRequired
};



export default HelloWorld;
```

Hooks

Ajouter de la logique à un composant Stateless

```
import React, { useState, useEffect } from 'react';

function Counter(props) {
  const [value, setValue] = useState(0);

  useEffect(() => {
    console.log('Mounted !');
  }, []);

  return (
    <div>
      <span>{value}</span>
      <button onClick={() => setValue(value + 1)}>+1</button>
      <button onClick={() => setValue(value - 1)}>-1</button>
    </div>
  );
}

export default Counter;
```

Cycle De Vie


willMount
willUnmount
didMount
didUpdate
willReceiveProps
render
shouldUpdate
willUpdate

Cas d'Usage

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

Stateful

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

class HelloWorld extends Component {
  componentDidMount() {
    console.log('Mounted !');

    fetch('/api/hello-world', {
      method: GET
    }).then(response => {
      console.log(response.json());
    });
  }

  render() {
    return (
      {/* ... */}
    );
  }
}
```

Stateless

```
import React, { useEffect } from 'react';

function HelloWorld() {
  useEffect(() => {
    console.log('Mounted !');

    fetch('/api/hello-world', {
      method: GET
    }).then(response => {
      console.log(response.json());
    });
  }, []);

  return (
    {/* ... */}
  );
}
```

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

  render() {
    return (
      <span>Hi {this.props.userName} !</span>
    );
  }
}
```

React Router

Explication

Affichage des composants en fonction de l'URL
Inspirée par le Router du framework Ember
Créée par **@ryanflorence** & **@mjackson**
Facilement implémentable avec Redux
Importance de la navigation par URL

Mapping

Définition des chemins pour chaques composants de l'application

`Suspense` permet de gérer le découpage dynamique

```
import React, { Suspense } from 'react';

import { Switch, Route } from 'react-router-dom';
import Home from './Home';
import Login from './Login';

// Code Splitting
const Contact = React.lazy(() => import('./Contact'));

function Routing() {
  return (
    <Switch>
      <Route exact path="/" component={Home} />
      <Route path="/login">
        <Login />
      </Route>
      <Route path="/contact/:userName" render={({ match }) => (
        <Suspense fallback={<div>Loading...</div>}>
          <Contact userName={match.params.userName} />
        </div>
      )} />
      <Route component={Home} />
    </Switch>
  );
}

export default Routing;
```

Intégration

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

```
import React from 'react';
import NavBar from './NavBar';
import Routing from './Routing';

export default function App() {
  return (
    <>
      <NavBar />
      <Routing />
    </>
  );
}
```

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 as Router } from 'react-router-dom';
import App from './App';

render(
  <Router>
    <App />
  </Router>,
  document.getElementById('root')
);
```

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/morty">Contact</Link>
        </button>
      </nav>
    );
  }
}

export default withRouter(NavBar);
```

Hooks

Le Retour !

Le *routing* plus simple grâce aux `hooks`

```
import React from 'react';
import { useParams, useHistory } from 'react-router-dom';

export default function Contact(props) {
  const { userName } = useParams();
  const history = useHistory();

  return (
    <div>
      <h1>UserName : {userName}</h1>
      <button onClick={() => history.push('/')}>Go Home</button>
    </div>
  );
}
```

Flux

Concept

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

- Flux
- Fluxxor
- MobX
- NgRx
- Overmind
- **Redux**
- Vuex

...

Philosophie

Gestion du `state` des composants simplifiée

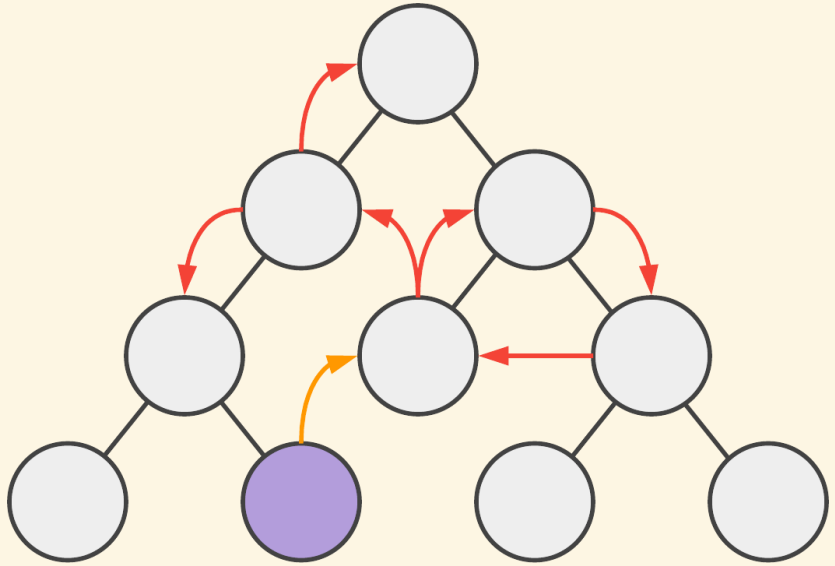
Le **store** est accessible à tout moment

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

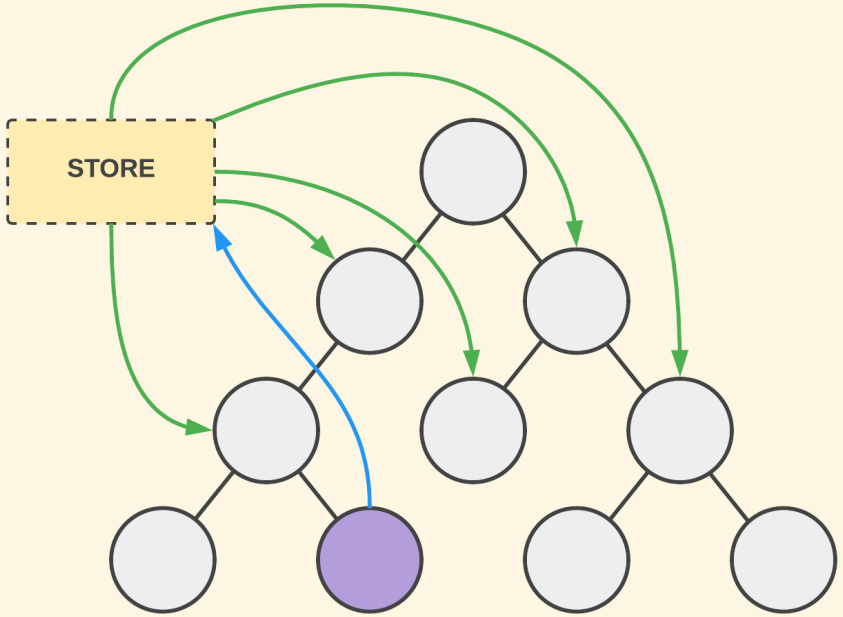
Aucune donnée en doublon (un seul point de vérité)

Meilleure conception de votre application

Sans 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 `reducer`s de l'applicatif
Le **store** est en lecture seul

Actions

Mise à jour du **store** uniquement via les `reducer`s 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 = 'TASKS/CREATE_TASK';
export const COMPLETE_TASK = 'TASKS/COMPLETE_TASK';
export const CLEAR_TASK = 'TASKS/CLEAR_TASK';

const createTask = task => ({ type: CREATE_TASK, payload: task });
const completeTask = id => ({ type: COMPLETE_TASK, payload: id });
const clearTask = id => ({ type: CLEAR_TASK, payload: id });
```

Reducer

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

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

export default function tasks (state = [], action) {
  switch (action.type) {
    case CREATE_TASK:
      return [
        ...state,
        action.payload
      ];

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

    case CLEAR_TASK:
      return state.filter(task => task.id !== action.payload);

    default:
      return state;
  }
}
```

Selectors

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

Il est possible de passer des paramètres aux `selectors`

```
export const getTasks = state => state.tasks || [];

export const getTaskById = id => state => getTasks(state).find(task => task.id === id);
```

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';

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

Utilisation

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

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

class ToDo extends Component {
  updateTask = id => {
    this.props.completeTask(id);
  };

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

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

const mapDispatchToProps = dispatch => ({
  completeTask: id => dispatch(completeTask(id))

});
```

Hooks

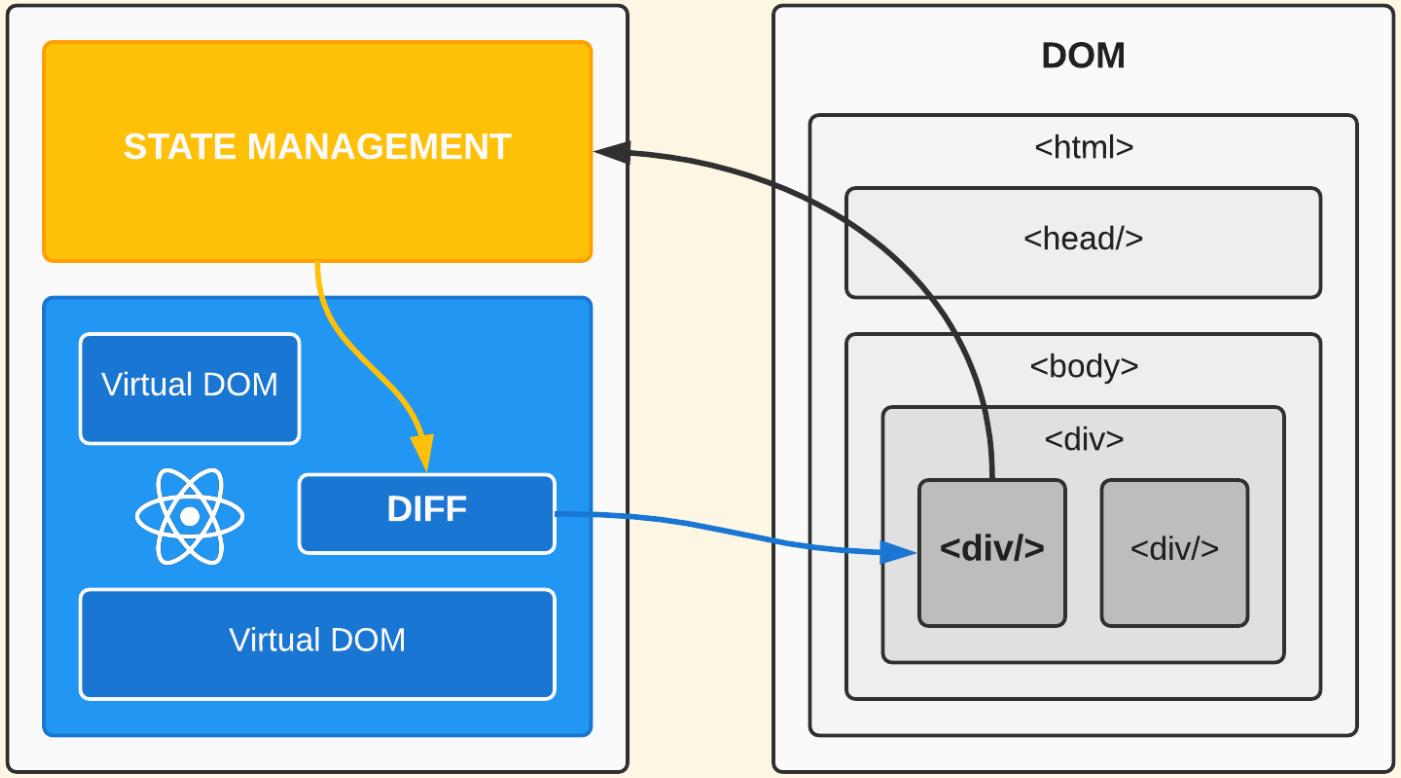
La Revanche !

Le *state management* vraiment plus simple grâce aux... `hooks`

```
import React from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { clearTask } from './actions';
import { getTaskById } from './selectors';

export default function ToDo(props) {
  const dispatch = useDispatch();
  const task = useSelector(getTaskById(42));

  const removeTask = id => {
    dispatch(clearTask(id));
  };

  return (
    <div>
      <h2>{task.label}</h2>
      <button onClick={() => removeTask(task.id)}>
        Remove
      </button>
    </div>
  );
}
```

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/react @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 React from 'react';
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 content="Hello World" />);
    expect(queryByText('Hello World')).toBeInTheDocument(); // Enhanced API
  });

  it('Should Click Trigger Works Well', () => {
    const mockedClick = jest.fn();
    const { getByRole } = render(<App handleClick={mockedClick} />);
    fireEvent.click(getByRole('button'));
    expect(mockedClick).toHaveBeenCalled();
  });
});
```

Tester Avec **Jest**

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

```
import * as Actions from '../actions';

import reducer from '../reducer';
import * as Selectors from '../selectors';

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

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

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

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

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

Svelte

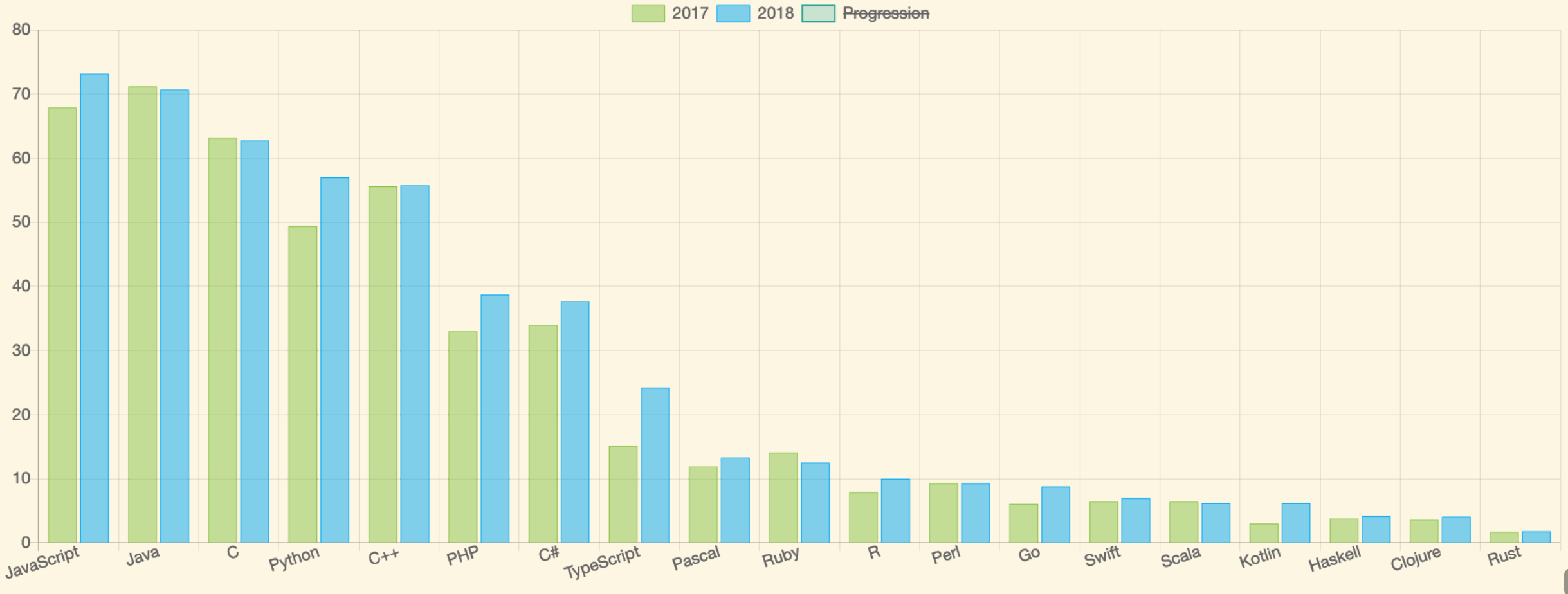
Propulsé par **Rich Harres** dès 2018
Compilateur **VanillaJS** ultra-performant
Utilisation du DOM à l'instar du **Virtual DOM**
Dédié à l'IoT, les *wearables*, le Web embarqué, etc...

Sapper pour le rendu côté serveur (*code-splitting*)

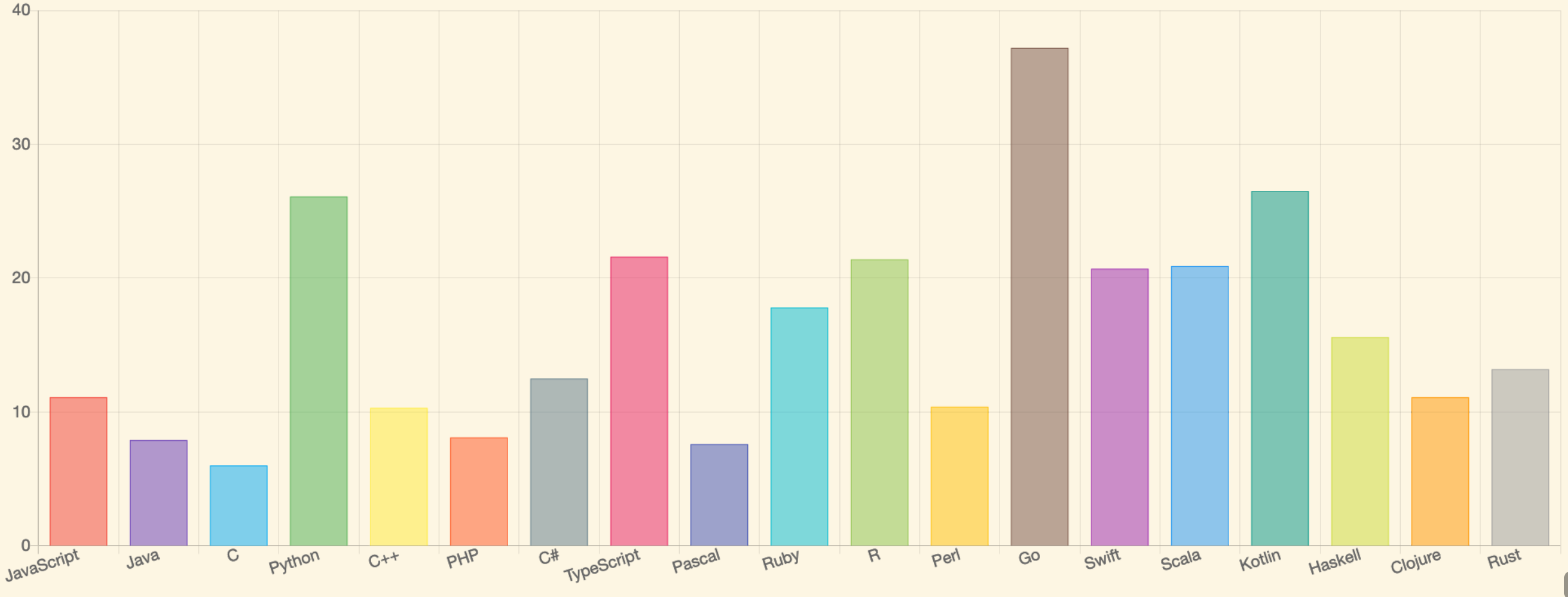
Infographie

Langages

Connaissances en 2017 Vs. 2018

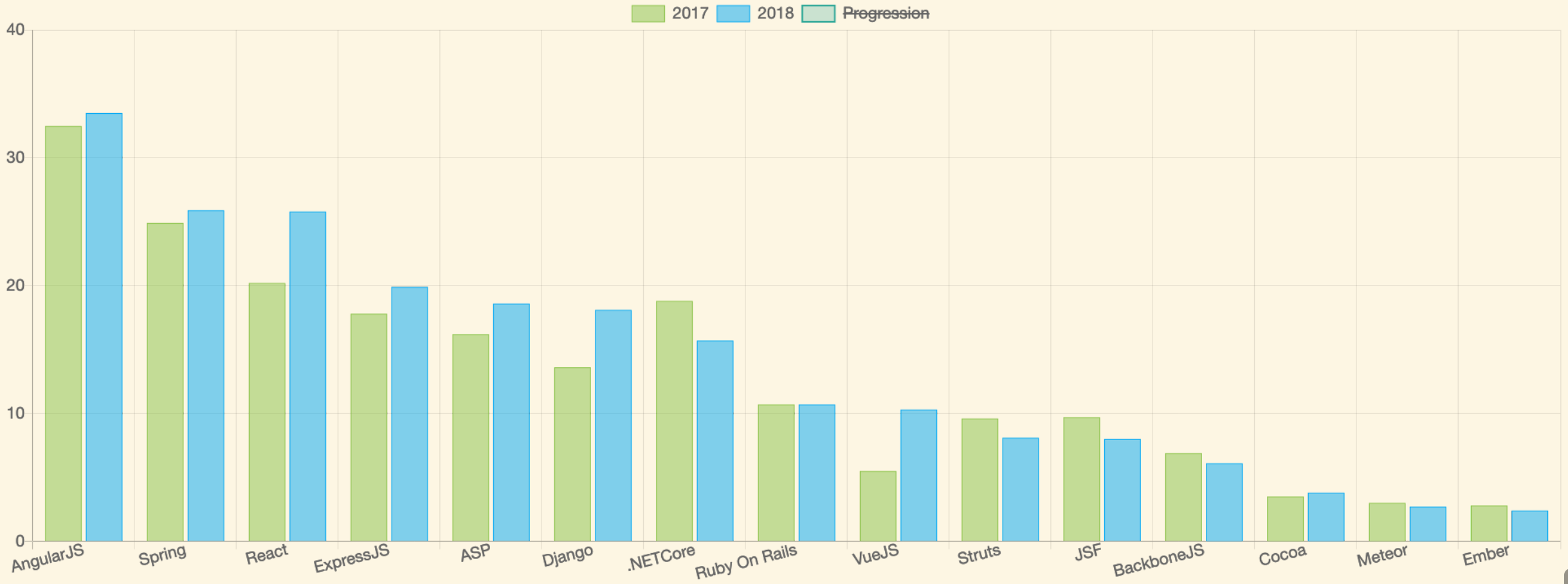


Quels langages les développeurs prévoient-ils d'apprendre ?

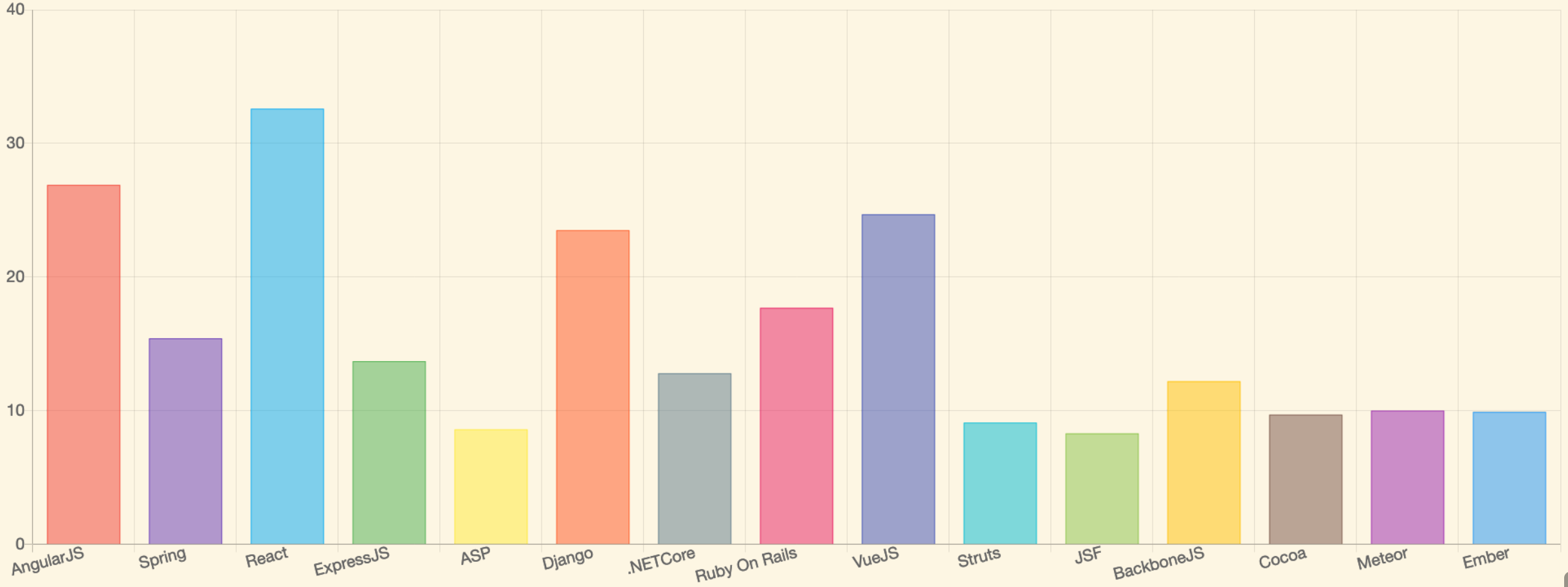


Framework

Connaissances en 2017 Vs. 2018

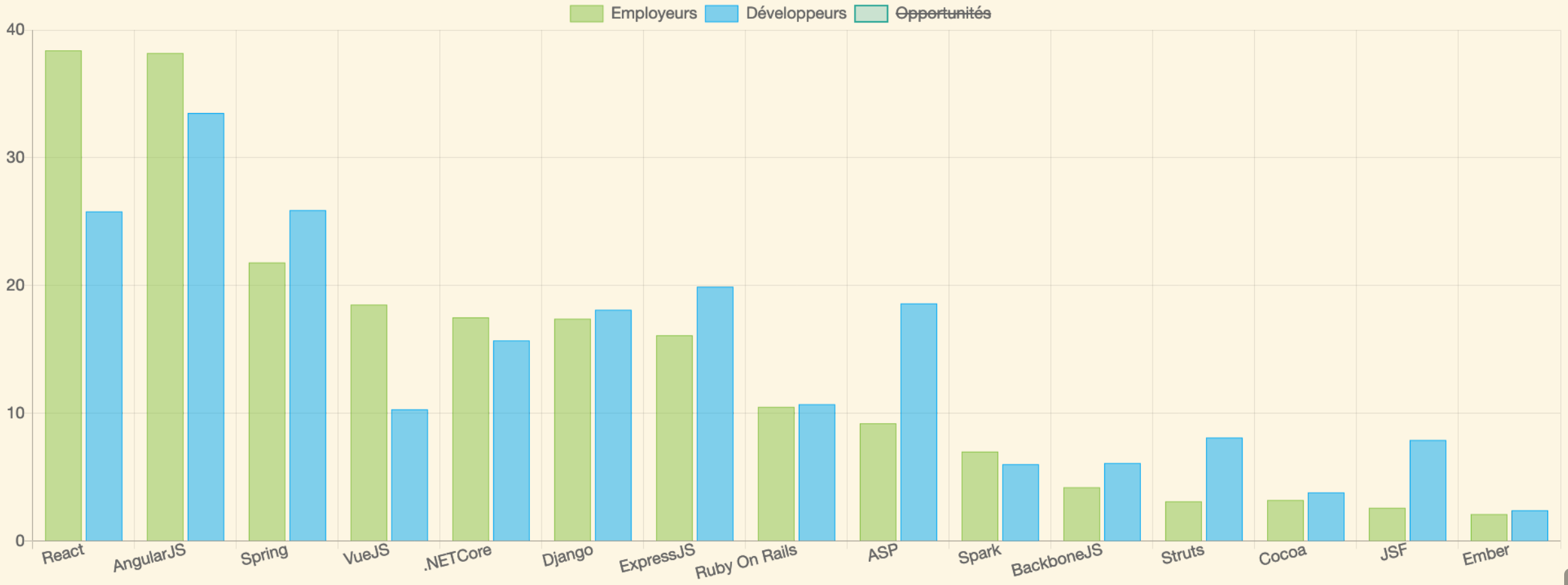


Quels frameworks les développeurs prévoient-ils d'apprendre ?



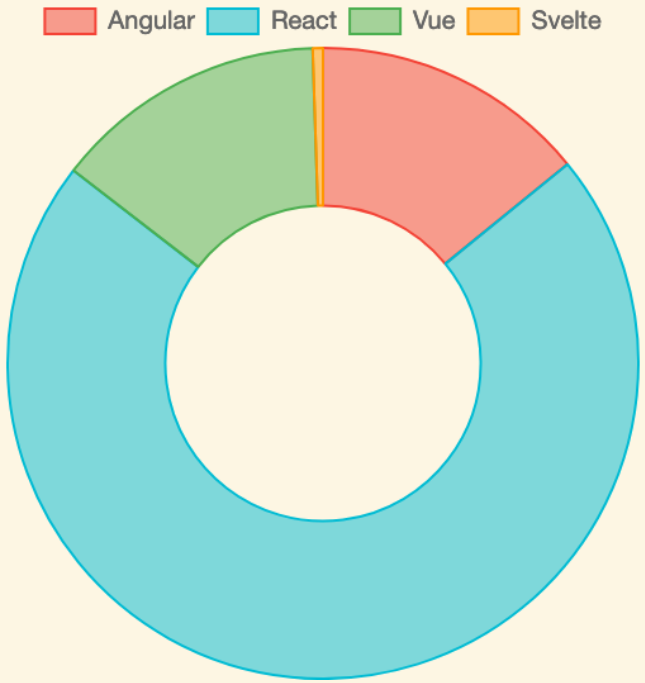
Frameworks

Besoin des employeurs Vs. Connaissances des développeurs



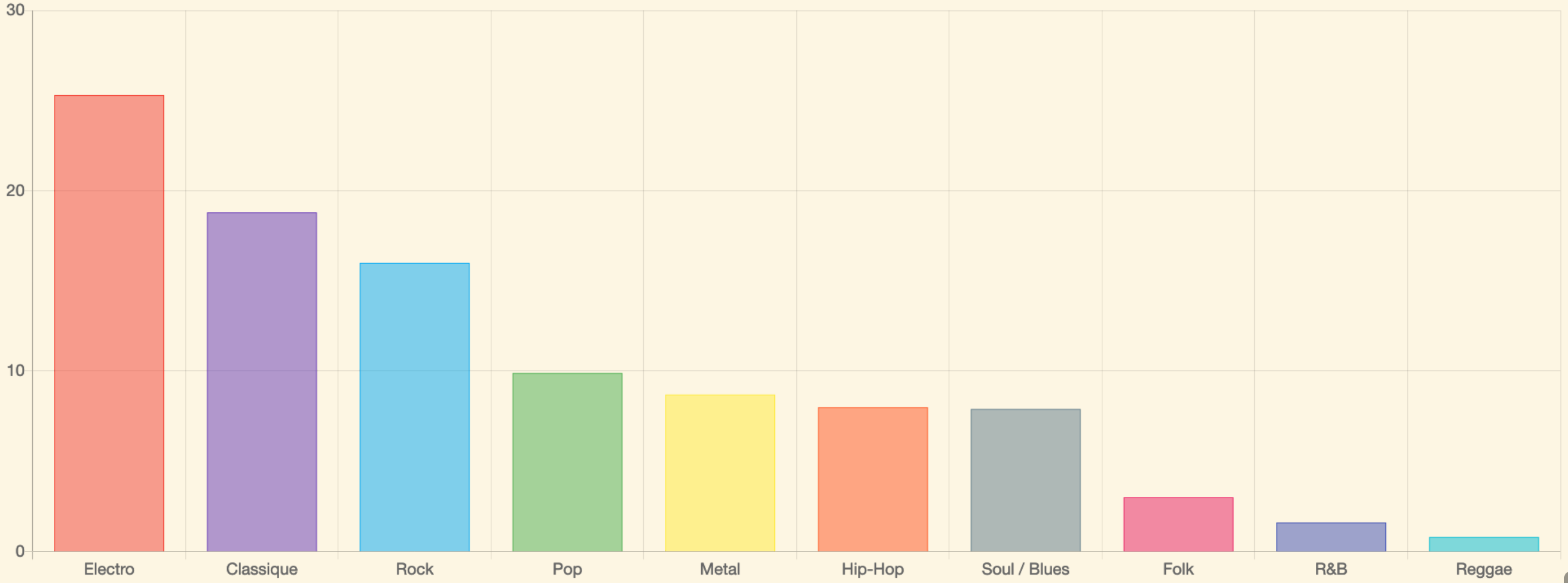
Téléchargements NPM*

Angular Vs. React Vs. Vue Vs. Svelte

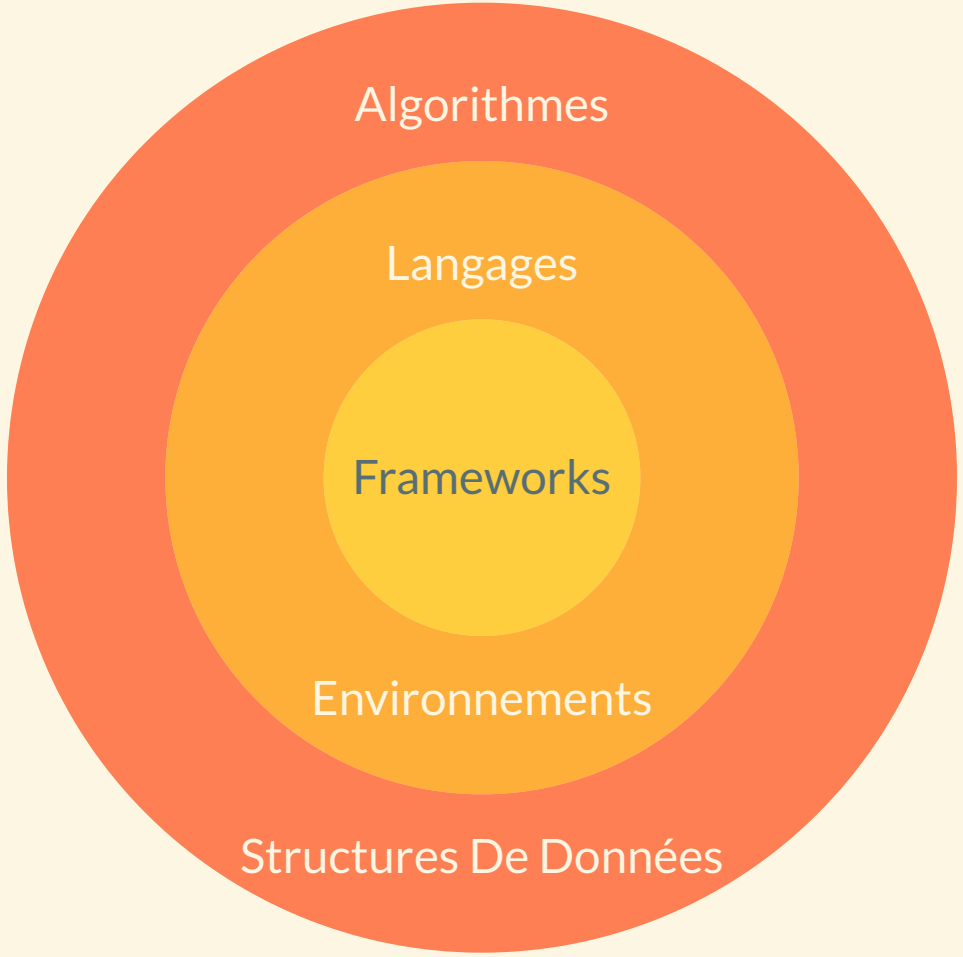


** 10 Mai 2020*

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
React	https://reactjs.org/
React Router	https://reacttraining.com/react-router/
Redux	https://redux.js.org/
Dan Abramov	https://overreacted.io/
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/