



M2 IAGL – IFI
ATMANE Bilal,
BEUNS Vianney,
ZENATI Lamine

Sommaire





Une intro à React

C'est quoi React ?



React...c'est pas :

Il y a beaucoup de **framework MVC en JavaScript**,
pourquoi utiliser React ?

Parce que React **n'est pas** un framework MVC en
JavaScript !

React qu'est-ce que c'est ?

- **React** (Ou **ReactJS**, **React.js**) est une bibliothèque open-source **Javascript** pour construire des interfaces utilisateurs
- Créée en 2011 par **Jordan Walke**, un développeur travaillant chez facebook, puis mise en open-source en 2013
- La librairie a été adopté par de nombreuses entreprises





Pourquoi React ?

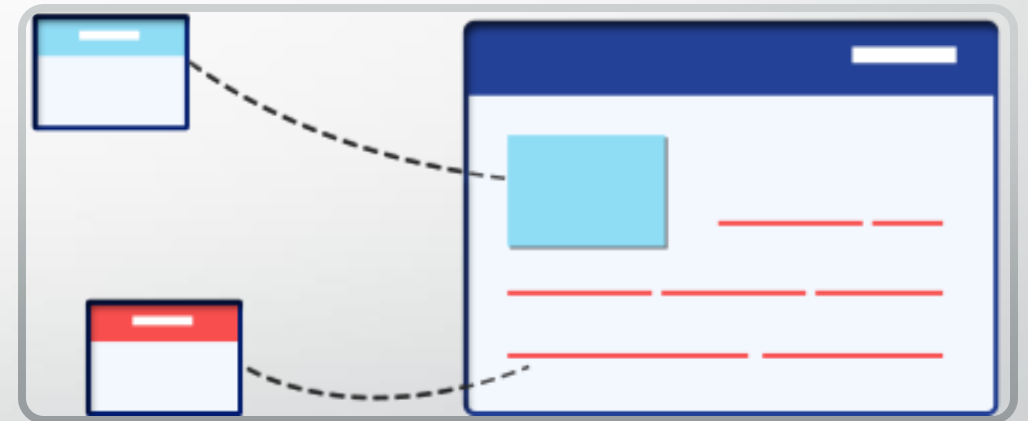
Pourquoi React ?

- Dans la plupart des librairies permettant la création d'interface utilisateur dans une application web, on utilise des **templates**.
- Pas React ! Les templates manquent de flexibilité.

Pourquoi React ?

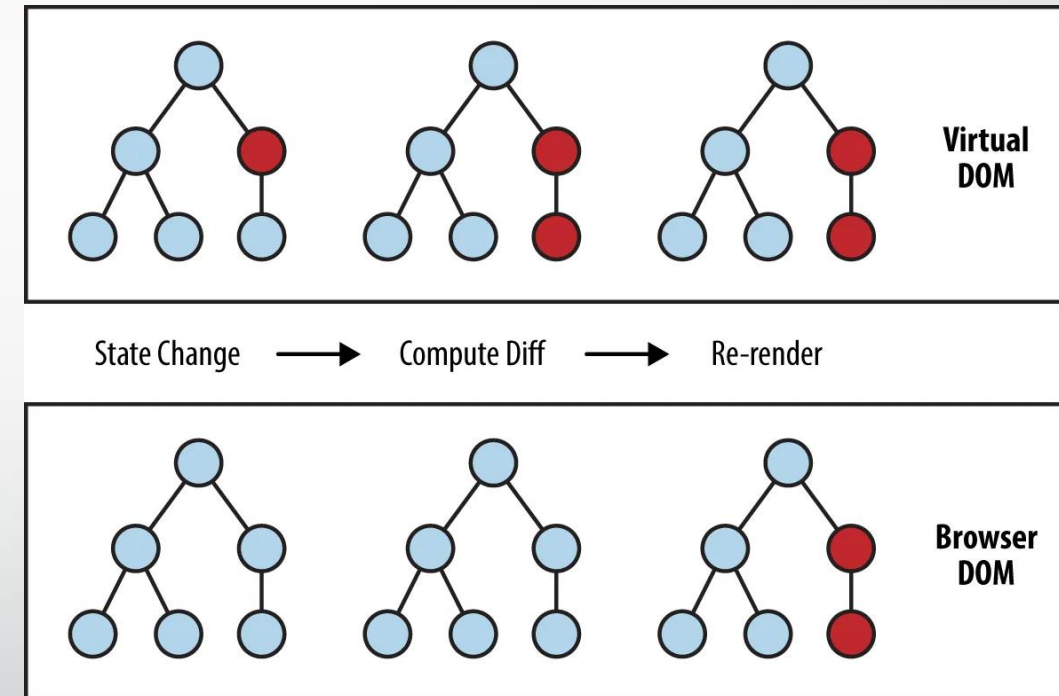
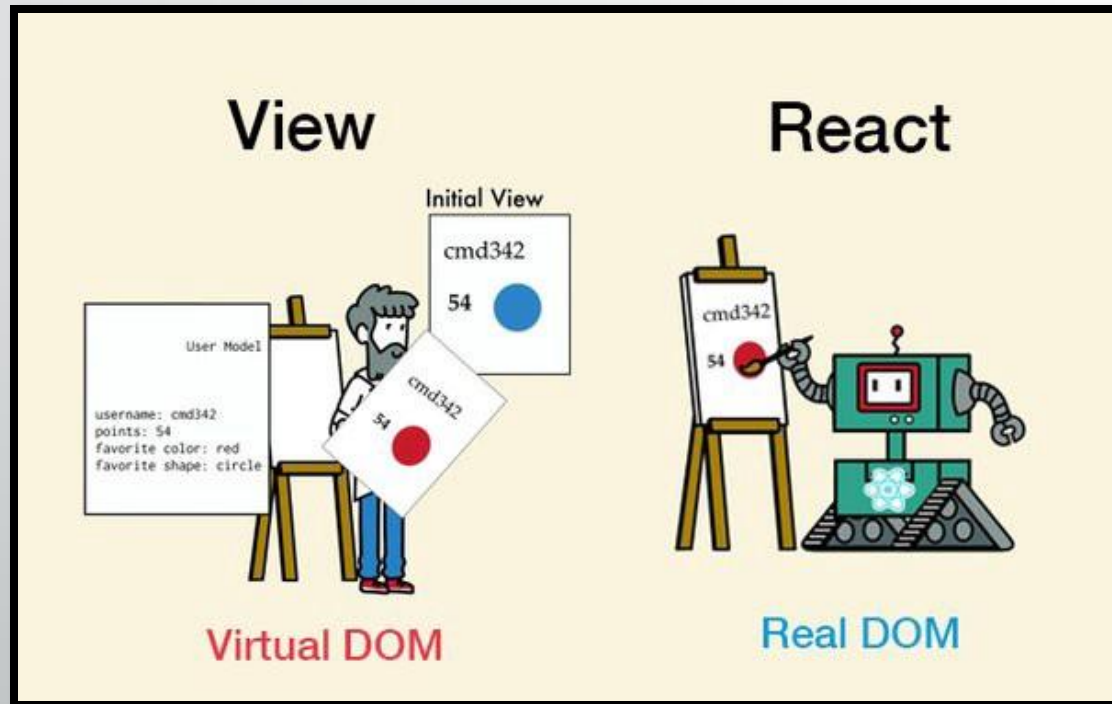
Une structure basée sur les composants

- Les composants sont des fragments unitaires de l'interface, ils facilitent l'isolation des responsabilités.
- Ils sont facilement réutilisables
- De par leur isolation, ils permettent une meilleure couverture de test
- La structure en composants permet d'étendre le développement d'interface à d'autres plateformes (ReactNative).



Pourquoi React ?

La réactivité offerte par le "virtual DOM"

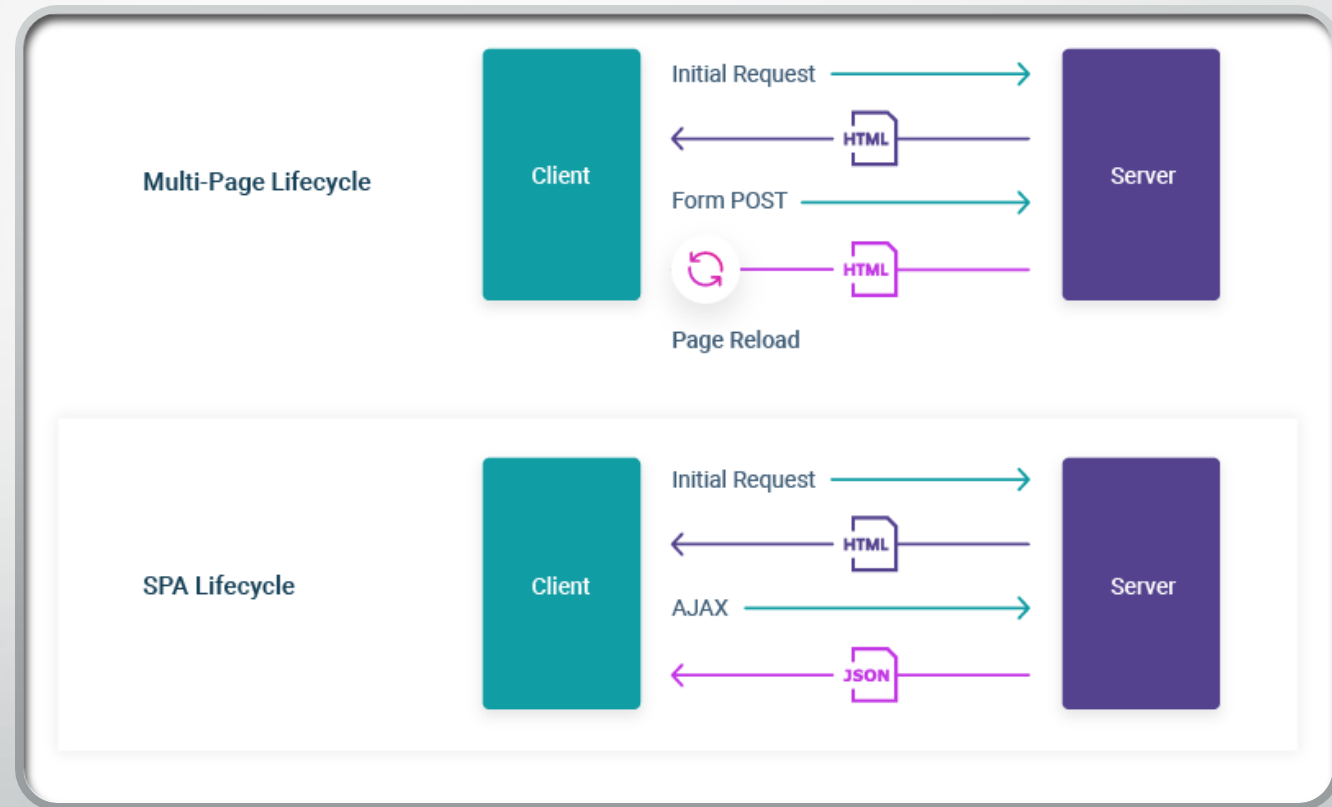


- React effectue les modifications sur un DOM virtuel, puis au moment du rendu, effectue uniquement les changements nécessaires sur le vrai DOM

Pourquoi React ?

Les Single Page Application (SPA)

- **La navigation est plus fluide** : finis les allers-retours avec le serveur pour générer la prochaine page visitée
- Le serveur est alors moins sollicité

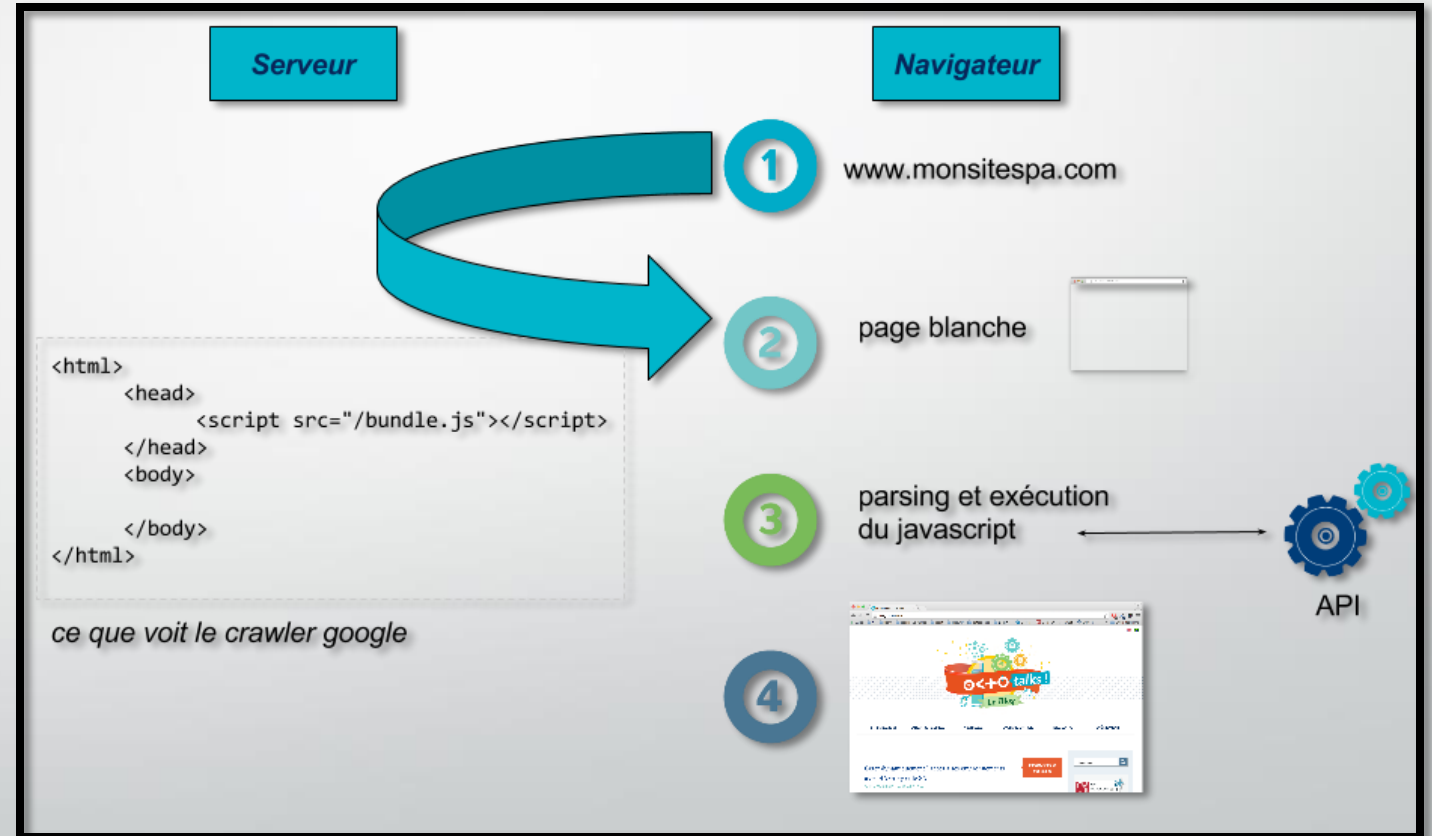


Pourquoi React ?

Les Single Page Application (SPA)

- **Problème des SPAs:**

- La page est générée uniquement dans le navigateur, ainsi le **premier chargement peut être long**
- Le chargement initial peut poser problème pour le **référencement** du site.



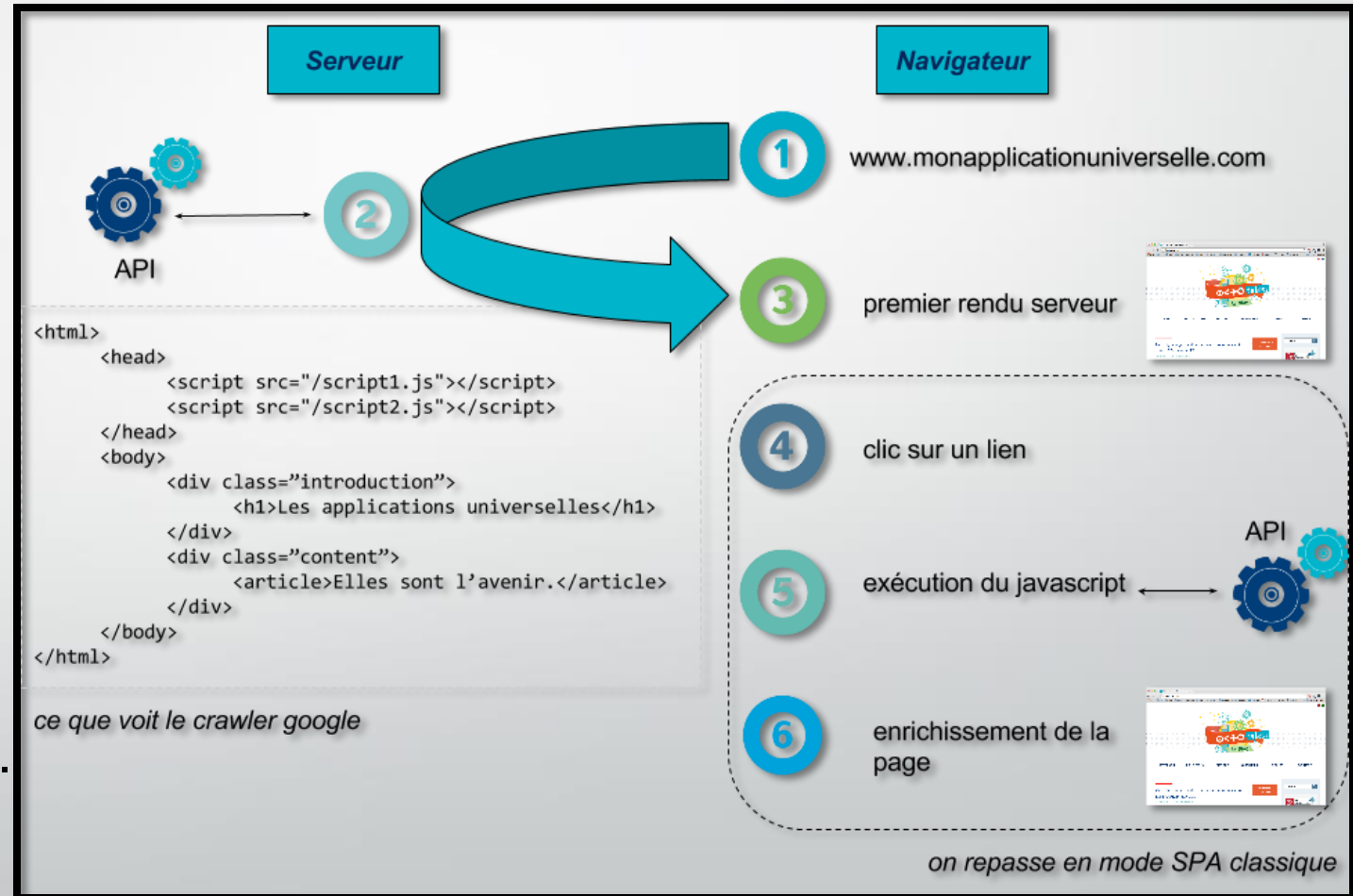
Pourquoi React ?

Les Single Page Application (SPA)

Solution:

Application "isomorphique"

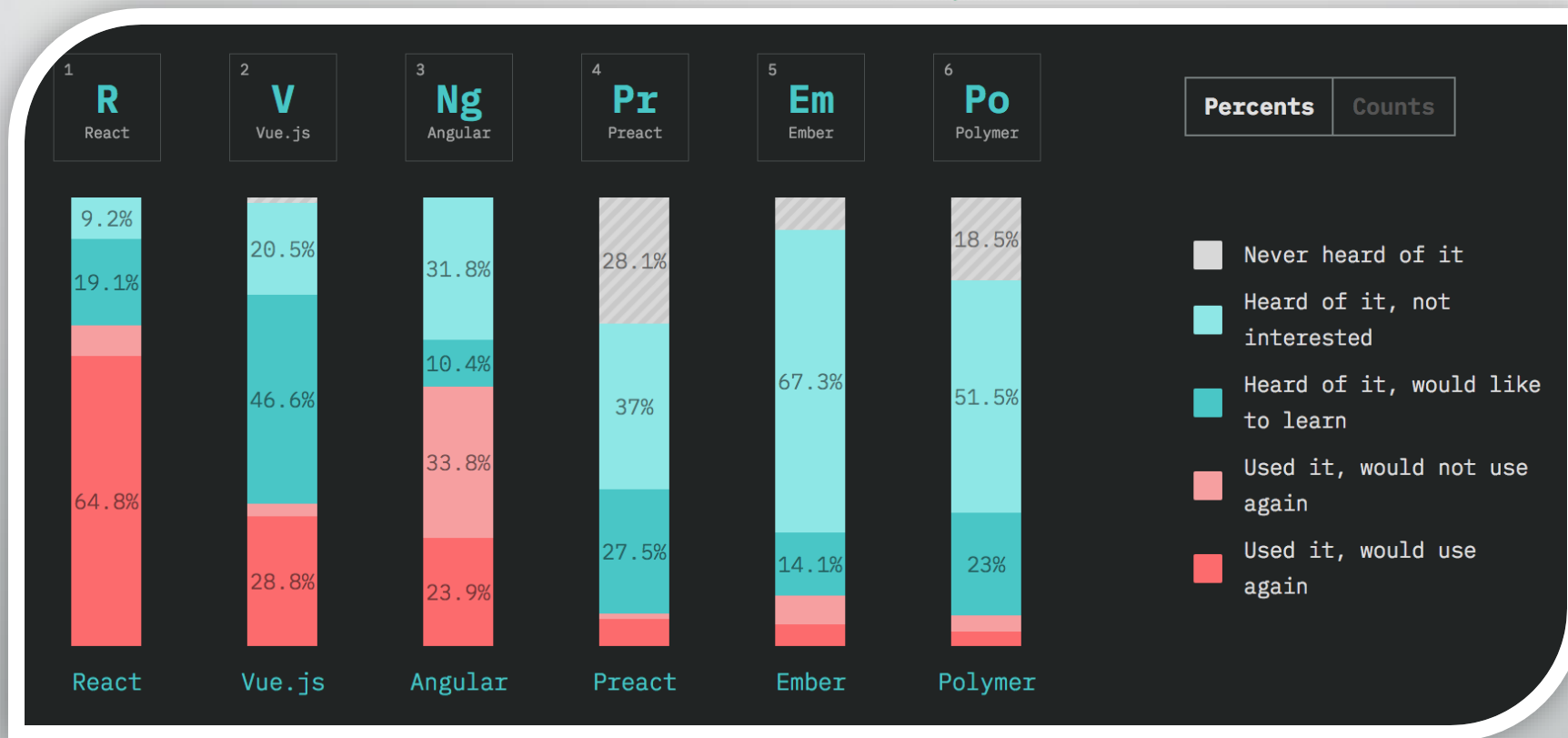
- Le premier rendu de l'application est fait côté serveur, puis côté client, on retrouve une SPA traditionnelle.
- Le premier rendu sera donc rapide, l'application fluide et le référencement sera plus efficace.





Les alternatives

Les alternatives à React



stateofjs.com (2018)

Angular vs React vs Vue



Plus mature et forme un package complet.

Cependant, difficile à apprendre.



Relativement mature et énorme communauté

Facile à apprendre



Manque de maturité, aucune grosse entreprise.

Facile à apprendre et se démarque de plus en plus en Chine.



Présentation technique

Documentation officielle: fr.reactjs.org

Javascript & ES6

- ECMAScript 6 rajoute beaucoup de fonctionnalités au Javascript classique
- Possibilité d'écrire des classes de façon "naturel"
- En savoir plus:
 - <http://es6-features.org/>

```
class User {  
  // méthode constructor  
  constructor(firstname, lastname) {  
    this.firstname = firstname;  
    this.lastname = lastname;  
  }  
  
  sayName() {  
    return `${this.firstname} ${this.lastname}`;  
  }  
}  
  
// instantiation  
const user = new User("John", "Doe");  
  
// appel de la méthode sayName()  
console.log(user.sayName()); // John Doe
```

Introduction aux fonctions fléchées

- Les nouvelles versions de Javascript permettent d'écrire des fonctions fléchées
- Elles fournissent des outils bien pratiques
- Facilite le binding de "this"

```
function maFonction(x,y){  
    return x+y;  
}  
//une seule instruction, pas besoin d'accolades et de return  
let maFonction = (x,y) => x+y;  
let maFonction = (x,y) => { return x+y;}  
  
maFonction(1,2);
```

```
let a = [1,2,3,4,5,6];  
let b = a.map(x => x*2);  
let c = b.filter(x => x>6)  
// b= [2, 4, 6, 8, 10, 12]  
// c= [8, 10, 12]
```

Bonjour, monde !

```
ReactDOM.render(  
  <h1>Bonjour, monde !</h1>,  
  document.getElementById('root')  
>);
```

HTML

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

- La fonction 'render' est la fonction d'affichage de React
- Elle prend un composant en premier argument et l'insère dans un élément du DOM

Introduction à JSX

- Le JSX est une extension syntaxique de JavaScript
- Il permet de produire des "éléments" React
- Vous pouvez écrire des expressions JS mixées à du HTML
- Le JSX est transformé en objets JS à la compilation, vous pouvez donc l'utiliser dans votre code javascript.
- Le JSX protège des injections XSS, donc vous pouvez afficher les inputs des utilisateurs

```
const name = 'Clarisse Agbegenou';
const element = <h1>Bonjour, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

```
const element = <img src={user.avatarUrl}></img>;
```


```
function getGreeting(user) {
  if (user) {
    return <h1>Bonjour, {formatName(user)} !</h1>;
  }
  return <h1>Bonjour, Belle Inconnue.</h1>;
}
```

```
const element = (
  <div>
    <h1>Bonjour !</h1>
    <h2>Content de te voir ici.</h2>
  </div>
);
```

Composants et props

- Ces déclarations deux de composant sont équivalentes

```
function Welcome(props) {  
  return <h1>Salut, {props.name}</h1>;  
}
```



```
class Welcome extends React.Component {  
  constructor(props) {  
    super(props);  
  }  
  render() {  
    return <h1>Bonjour, {this.props.name}</h1>;  
  }  
}
```

- Les attributs définis en JSX lors de l'instanciation du composant sont récupérables à l'intérieur de celui-ci via l'attribut **props / this.props**
- Les différentes propriétés dans props constituent les attributs du composant

```
function App() {  
  return (  
    <div>  
      <Welcome name="Bilal" />  
      <Welcome name="Lamine" />  
      <Welcome name="Vianney" />  
    </div>  
  );  
}  
  
ReactDOM.render(<App />,  
  document.getElementById('root'));
```



Salut, Bilal

Salut, Lamine

Salut, Vianney

Les props sont Read-Only

```
function sum(a, b) {  
  return a + b;  
}
```

Fonction **pure** : ne cherche pas à modifier son paramètre

```
function withdraw(account, amount) {  
  account.total -= amount;  
}
```

Fonction **impure** : cherche à modifier son paramètre

- Tous les composants React doivent agir comme des fonctions pures en vis-à-vis leurs props.
- Une fois un composant instancié, les props ne doivent jamais être modifiées, si vous ressentez le besoin de le faire, il faudra utiliser les états (state)

Etat et cycle de vie

- À la différence de l'accès aux données via ***this.props***, un composant peut avoir un état interne accessible via ***this.state***.
- Il faut créer une classe ES6 pour cela.
- Définir l'état dans le constructeur sans oublier d'appeler le super-constructeur avec *props*.
- La fonction "render" à dès qu'une modification de l'état est détecté

```
class Timer extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { seconds: 0 };  
  }  
}
```

Etat et cycle de vie

- Afin de signaler à `render()` de se relancer, on crée un minuteur dans la fonction **`componentDidMount()`**.
- Celle-ci sera appelée à la première apparition du composant dans le DOM.
- Ne modifier l'état qu'avec la fonction **`setState()`** !!!

```
tick() {  
  this.setState(state => ({  
    seconds: state.seconds + 1  
  }));  
}  
  
componentDidMount() {  
  this.interval = setInterval(() => this.tick(), 1000);  
}  
  
componentWillUnmount() {  
  clearInterval(this.interval);  
}
```


Etat et cycle de vie

- On fait référence à **this.state** dans la fonction *render()*.
- On note qu'aucun paramètre n'est donné dans **ReactDOM.render()**.

```
render() {  
  return (  
    <div>  
      Secondes : {this.state.seconds}  
    </div>  
  );  
}  
  
ReactDOM.render(  
  <Timer />,  
  document.getElementById('timer-example')  
);
```

Gérer les événements

- En Javascript, les méthodes de classe ne sont pas *liées* à leur classe par défaut, pour pallier à ça:
 - Binder les méthodes de classe manuellement
 - Utiliser les lambdas fonctions
- Si ce n'est pas fait, l'utilisation de ***this.handleClick*** dans le JSX renverra *undefined* pour *this*.

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // Cette liaison est nécessaire afin de permettre
    // l'utilisation de `this` dans la fonction de rappel.
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(state => ({
      isToggleOn: !state.isToggleOn
    }));
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```

```
render() {
  return (
    <button onClick={() => this.handleClick()}>
      {this.state.isToggleOn ? 'ON' : 'OFF'}
    </button>
  );
}
```

Affichage conditionnel

```
function UserGreeting(props) {  
  return <h1>Bienvenue !</h1>;  
}  
  
function GuestGreeting(props) {  
  return <h1>Veuillez vous inscrire.</h1>;  
}
```

```
function Greeting(props) {  
  const isLoggedIn = props.isLoggedIn;  
  if (isLoggedIn) {  
    return <UserGreeting />;  
  }  
  return <GuestGreeting />;  
}
```

```
ReactDOM.render(  
  // Essayez de changer ça vers isLoggedIn={true} :  
  <Greeting isLoggedIn={false} />,  
  document.getElementById('root')  
);
```

Affichage conditionnel

- Il est possible d'afficher conditionnellement du JSX à la volée avec l'**opérateur logique &&**.
- Possible car en JavaScript *true && expression* est toujours évalué à **expression**, et *false && expression* est toujours évalué à **false**.

```
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      <h1>Bonjour !</h1>
      {unreadMessages.length > 0 &&
        <h2>
          Vous avez {unreadMessages.length} message(s) non-lu(s).
        </h2>
      }
    </div>
  );
}

const messages = ['React', 'Re: React', 'Re:Re: React'];
ReactDOM.render(
  <Mailbox unreadMessages={messages} />,
  document.getElementById('root')
);
```

Listes et clés

- Utilisation de **map()** pour créer une liste de ``.
- On signale une clé qui se doit d'être unique (seulement dans cette liste).

```
function NumberList(props) {  
  const numbers = props.numbers;  
  const listItems = numbers.map((number) =>  
    <li key={number.toString()}>  
      {number}  
    </li>  
  );  
  return (  
    <ul>{listItems}</ul>  
  );  
}  
  
const numbers = [1, 2, 3, 4, 5];  
ReactDOM.render(  
  <NumberList numbers={numbers} />,  
  document.getElementById('root')  
);
```

Formulaires

- On peut court-circuiter les formulaires HTML pour les traiter en React.

```
class NameForm extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {value: ''};  
  
    this.handleChange = this.handleChange.bind(this);  
    this.handleSubmit = this.handleSubmit.bind(this);  
  }  
  
  handleChange(event) {  
    this.setState({value: event.target.value});  
  }  
  
  handleSubmit(event) {  
    alert('Le nom a été soumis : ' + this.state.value);  
    event.preventDefault();  
  }  
}
```

```
render() {  
  return (  
    <form onSubmit={this.handleSubmit}>  
      <label>  
        Nom :  
        <input type="text" value={this.state.value} onChange={this.handleChange} />  
      </label>  
      <input type="submit" value="Envoyer" />  
    </form>  
  );  
}
```

Faire remonter l'état

- D'une manière générale, on cherchera à avoir le moins d'état possible dans nos composants de base. On cherche à les rendre les plus aveugles à leur contexte possible.
- De la sorte, on pourra centraliser les spécificités du contexte le plus haut possible dans l'application.

Composition ou héritage ?

- En React, on préférera toujours la **composition** plutôt que **l'héritage**.
- Cela se fait via l'instruction JSX `{props.children}`.

```
function WelcomeDialog() {  
  return (  
    <FancyBorder color="blue">  
      <h1 className="Dialog-title">  
        Bienvenue  
      </h1>  
      <p className="Dialog-message">  
        Merci de visiter notre vaisseau spatial !  
      </p>  
    </FancyBorder>  
  );  
}
```

```
function FancyBorder(props) {  
  return (  
    <div className={'FancyBorder FancyBorder-' + props.color}>  
      {props.children}  
    </div>  
  );  
}
```




Démonstration en live

Feat. Bilal

Repo. Git:

https://gitlab.univ-lille.fr/lamine.zenati.etu/react_masterclass_ifi