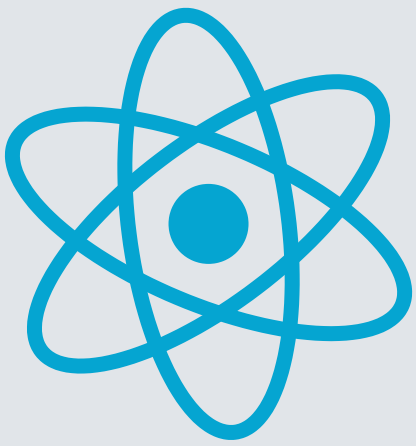
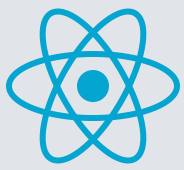


cours **React - J3**

IG2I 2021

une idée derrière la tech



React

1. Rappels
2. Syntaxes ES6+
3. **Premiers pas avec React**
4. Les composants
5. Formulaires et AJAX
6. Hooks & React Router

3. PREMIERS PAS AVEC REACT

- Présentation
  - Les composants
  - La syntaxe JSX
  - Le "state"

PRÉSENTATION

- Ne fournit que la partie vue (interface graphique)
- Pas de techno imposée pour la gestion des données (pattern Flux)
- Approche web component
- Virtual DOM

Notes :

Contrairement à d'autres frameworks/libs JS, React ne fournit que la partie vue de l'application et laisse au développeur le choix des armes pour la gestion des données, même si Facebook recommande pour ça un pattern bien particulier : Flux. Nous verrons bientôt ce qu'il en est.

React a une approche web component, c'est à dire qu'il vise à étendre le langage HTML en permettant au développeur de créer ses propres balises (appelées "Components").

Cependant le framework n'est pas limité au Web puisqu'il permet également de développer des applications mobile natives voire s'utiliser sur d'autres plateformes à l'avenir.

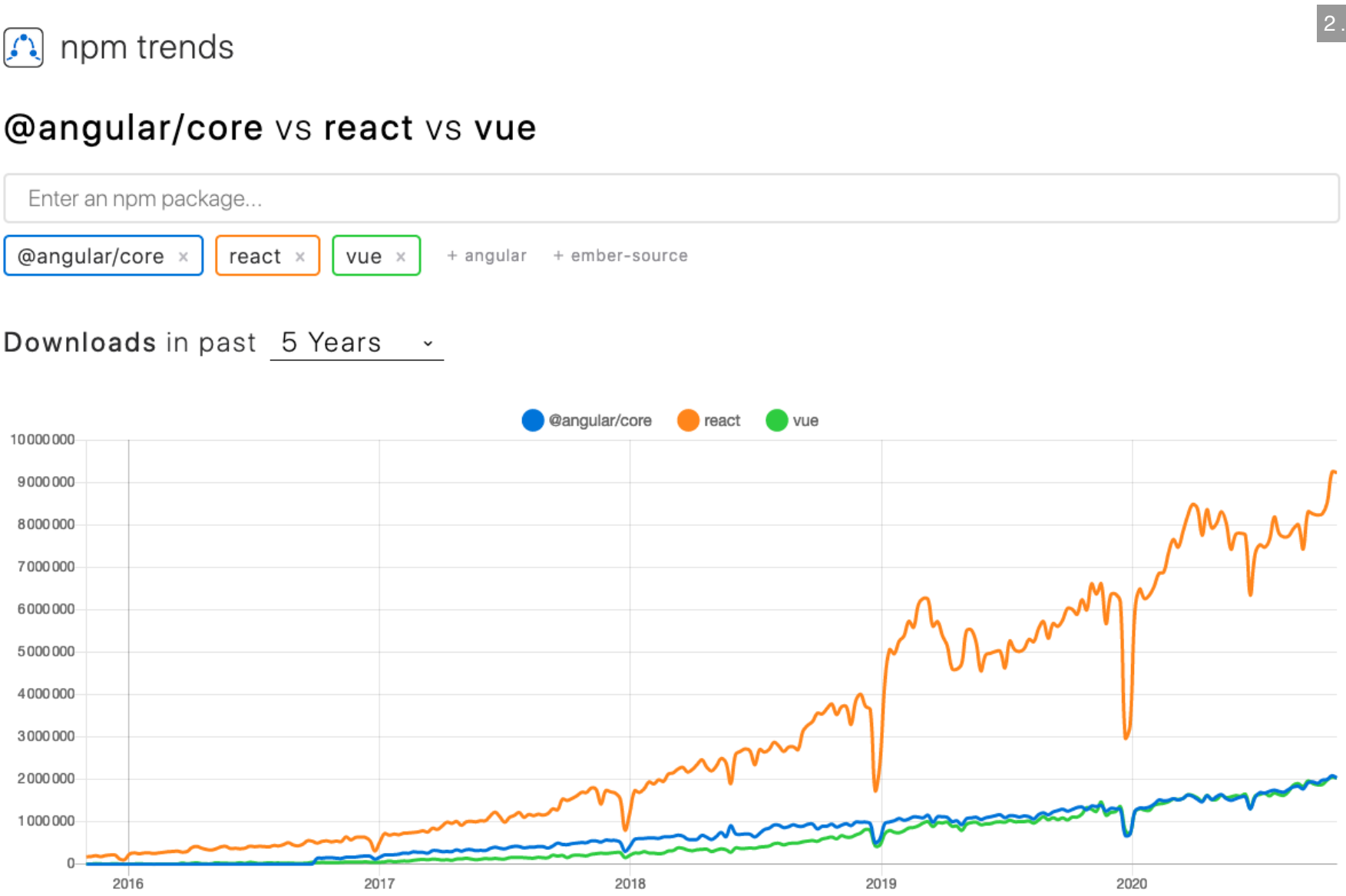
Enfin le système de "Virtual DOM" a fait en partie le succès de React :

Ici le DOM réel est associé à une représentation virtuelle de lui même appelée Virtual DOM.

Lorsque le modèle de données subit un changement, une nouvelle version du Virtual DOM est générée dans son intégralité.

Ensuite, une comparaison est faite entre la précédente version et la nouvelle afin de n'appliquer au DOM réel que les changements nécessaires (mise à jour d'un attribut, suppression/ajout d'un élément, etc...).

Cette approche est intéressante en terme de performance car elle limite les appels à l'API du DOM au strict minimum en n'appliquant que les changements nécessaires plutôt que de régénérer intégralement l'arbre du DOM.



Notes :

https://www.npmtrends.com/react-vs-@angular/core-vs-vue

Courbe des téléchargements des principales libs de dev front.

3. PREMIERS PAS AVEC REACT

- Présentation
- **Les composants**
- La syntaxe JSX
- Le "state"

CRÉATION D'UN COMPOSANT

MonComposant.js

```
import React from 'react';

export default class MonComposant extends React.Component {
    render() {
        return (
            <h1>Hello world</h1>
        );
    }
}
```

Notes :

Pour créer notre premier composant, la première chose à faire est de créer le fichier JavaScript qui va contenir le code du composant.

Par convention on le nommera d'après le nom du composant en respectant la Upper Camel Case.

Un composant React est une classe qui hérite de React.Component.

Il doit impérativement implémenter la méthode "render" qui détermine ce qu'affiche le composant.

On voit dans cet exemple que l'on retourne ce qui semble être du code HTML. Il s'agit en fait de JSX, une syntaxe permettant d'écrire du HTML à l'intérieur de fichiers JavaScript.

AFFICHAGE DU COMPOSANT

app.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import MonComposant from "./components/MonComposant";

ReactDOM.render(
    <MonComposant />,
    document.querySelector( '.appContainer' )
);
```

Notes :

Pour afficher un composant on utilise un module supplémentaire: "react-dom".

Ce module permet simplement de faire la liaison entre les composants React et le DOM. La méthode "render()" de react-dom permet d'instancier un composant et de l'associer à un élément de la page HTML pour qu'il puisse s'y rendre.

3. PREMIERS PAS AVEC REACT

- Présentation
- Les composants
- **La syntaxe JSX**
- Le "state"

LA SYNTAXE JSX

- Permet d'écrire du HTML dans le JavaScript
- Ce n'est pas du vrai HTML
- Possibilité d'injecter des variables dans le code

```
import React from 'react';
class MonComposant extends React.Component {
    render() {
        const label = 'Los Pollos Hermanos',
            link = 'https://myhugemethlab.com';

        return (
            <a href={link}>{label}</a>
        );
    }
}
```

Notes :

Le JSX est une syntaxe proche du HTML utilisable directement dans le JavaScript. Elle diffère du HTML standard principalement par le fait qu'il est possible d'y injecter des variables javascript.

JSX VS JAVASCRIPT

En réalité, le JSX est compilé en JS !

```
import React from 'react';
class MonComposant extends React.Component {
    render() {
        const label = 'Los Pollos Hermanos',
            link = 'https://myhugemethlab.com';

        return React.createElement(
            'a',
            { href: link },
            label
        );
    }
}
```

Notes :

Le JSX n'étant pas une syntaxe officielle de JS, elle doit en réalité être compilée en JS pour être interprété correctement par le navigateur.

Concrètement, on pourrait donc tout à fait écrire nos composants en utilisant uniquement du JavaScript et en se passant totalement du JSX.

Ceci dit, développer des composants en utilisant uniquement `React.createElement` s'avère vite très lourd, et il faut bien admettre que l'on a pas encore trouvé mieux pour décrire une structure arborescente comme le HTML, que le HTML lui même !

CAS PARTICULIERS : CLASSES CSS


```
class MonComposant extends React.Component {
    render() {
        return (
            <h1 className="deaAgent buriedInDesert">
                Hank
            </h1>
        )
    }
}
```

Le mot "class" est un mot-clé réservé en JavaScript

Notes :

Le mot "class" étant un mot-clé JavaScript il n'est pas possible de l'utiliser directement dans le JSX. Du coup, pour définir la classe d'un élément on utilise "className" à la place, c'est React qui fera la conversion de manière à ce que dans le code HTML généré, ce soit bien `class="..."` qui soit affiché !

CAS PARTICULIERS : STYLES INLINE

```
class MonComposant extends React.Component {
    render() {
        const style = {
            border: '1px solid black',
            backgroundColor: 'red', // attributs en lowerCamelCase
            fontSize: 12
        };

        return (
            <h1 style={style}>
                Mon titre
            </h1>
        )
    }
}
```

Notes :

En JSX, l'attribut style ne prend pas une chaîne de caractère en paramètre mais un objet JS contenant les attributs CSS à appliquer.

Les noms d'attribut doivent être écrits en **lowerCamelCase** plutôt qu'en dash-case (comme c'est le cas en CSS). La raison ? Si l'on écrivait `background-color`, le "-" serait interprété comme un "moins" (opérateur de soustraction) ce qui va évidemment poser problème !

CAS PARTICULIERS : BOUCLES

- Pas de "for" ni de "while" en JSX
- Chaque élément doit avoir une "key" unique

```
class MonComposant extends React.Component {
    render() {
        const todos = [
            { id: 1, task: 'Find a lab' },
            { id: 2, task: 'Send Jesse to rehab' },
            { id: 3, task: '(better) Call Saul' },
        ];
        return (
            <ul>
                { todos.map( todo => (
                    <li key={todo.id}>{todo.task}</li>
                ) ) }
            </ul>
        );
    }
}
```

```
class MonComposant extends React.Component {
    render() {
        const todos = [
            { id: 1, task: 'Find a lab' },
            { id: 2, task: 'Send Jesse to rehab' },
            { id: 3, task: '(better) Call Saul' },
        ];
        return (
            <ul>
                { this.renderTodos(todos) }
            </ul>
        );
    }
    renderTodos( todos ) {
        const result = []; // On crée un tableau vide
        for ( let i = 0; i < todos.length ; i++ ) {
            // Pour chaque todo on crée un élément React
            result.push( <li key={ todo.id }>{ todo.task }</li> );
        }
        return result; // On retourne le tableau de JSX
    }
}
```

Notes :

En JSX, il n'est pas possible d'utiliser de boucle "for" ou "while". Si l'on souhaite générer une liste d'élément dynamiquement à partir d'un tableau il suffit d'utiliser la méthode map.

La méthode map prend une fonction en paramètre et l'appelle pour chaque élément se trouvant dans le tableau. A chaque appel, la fonction reçoit l'élément courant et son index dans le tableau. Cette fonction doit générer le code JSX correspondant à l'élément courant et le retourner. La méthode map génère alors un autre tableau contenant le retour de chaque appel à la fonction. Dans le cas présent on obtient donc un tableau d'éléments React.

Pour parcourir une liste d'éléments, une autre solution consiste à passer par une sous-méthode qui retournera un tableau d'éléments JSX.

L'attribut "key" aide React de distinguer les composants les uns des autres et à optimiser le mécanisme de diff du Virtual DOM. C'est également important pour les animations. cf.
https://reactjs.org/docs/reconciliation.html#recursing-on-children

CAS PARTICULIERS : CONDITIONS

Pas de "if", "else", "else if" en JSX

```
class MonComposant extends React.Component {
    render() {
        return (
            <section>
                {this.renderWelcomeMessage( 'M' )}
            </section>
        );
    }
    renderWelcomeMessage( title ) {
        if ( title == 'M' ) {
            return <p>Bonjour Monsieur</p>
        } else if ( title == 'Mme' ) {
            return <p>Bonjour Madame</p>
        } else {
            return <p>Bonjour...</p>
        }
    }
}
```

```
class MonComposant extends React.Component {
    render() {
        const title = 'M';
        return (
            <section>{
                title == 'M' ?
                <p>Bonjour Monsieur</p> :
                <p>Bonjour Madame</p>
            }</section>
        );
    }
}
```

```
class MonComposant extends React.Component {
    render() {
        const role = 'admin';
        return (
            <div>
                {role == 'admin' && <button>Supprimer</button>}
            </div>
        )
    }
}
```

Notes :

À l'instar des boucles, les structures de contrôle "if", "else", "elseif" ne sont pas utilisables en JSX.

En revanche, comme on l'a vu pour les boucles, il est également possible de passer par l'appel à une méthode qui pourra, elle, faire usage des conditions.

Le ternaire est un type d'expression JavaScript qui se découpe de la manière suivante: [condition] ? [valeur à retourner si la condition est vraie] : [valeur à retourner si la condition est fausse].

C'est une autre possibilité pour remplacer les "if/else" qui n'ont pour but que de déterminer la valeur d'une variable en fonction d'une condition particulière. Cette technique présente l'avantage d'être beaucoup moins verbeuse mais attention aux conditions & valeurs trop complexes qui peuvent à partir d'une certaine taille nuire à la lisibilité.

Enfin, pour des cas plus simples où il s'agit simplement d'afficher ou de masquer un élément selon une condition précise il est possible d'utiliser l'opérateur logique "&&".

L'astuce ici réside dans le fait que dans une expression logique composée uniquement de "&&" l'expression retourne soit la première expression fausse qu'il trouve ou, s'il n'y a pas, retourne la dernière.

Ainsi, si la condition se trouvant avant le "&&" est fausse, l'expression logique retourne "false" (qui est interprété par React comme du "rien" et ne génère pas d'affichage) sinon elle retourne ce qui se trouve après le "&&" soit l'élément que l'on veut afficher si la condition est vraie.

3. PREMIERS PAS AVEC REACT

- Présentation
- Les composants
- La syntaxe JSX
- **Le "state"**

LE STATE D'UN COMPOSANT

- this.state
- Contient les données "variables" du composant
- Re-déclenche un render() à chaque modification !

```
class MonComposant extends React.Component {
    state = {
        userIsConnected: false,
        user: null
    };

    componentDidMount() { // appelée après le premier render
        fetch('https://monsite.come/api/user') // appel AJAX
            .then( response => response.json() ) // parsing du JSON
            .then( data => {
                // Une fois les données reçues on met à jour le state
                this.setState( {
                    userIsConnected: true,
                    user: data
                } );
    }

    render() {
        return (
            <div>
                <h1>Mon site en React</h1>
                {
                    this.state.userIsConnected &&
                    <p>Bienvenue {this.state.user.nickName}</p>
                }
            </div>
        );
    }
}
```

Notes :

Le state représente l'état du composant. Il se matérialise dans le composant React sour la forme d'un attribut "this.state" contenant un objet avec des paires clé/valeur.

On stocke dans le state toutes les données variables du composant qui vont avoir un impact sur l'affichage du composant (par exemple l'état "coché"/"pas coché" d'une checkbox, l'onglet sélectionné dans un accordéon, etc.).

Le gros avantage de passer par le state pour stocker ces informations, c'est que dès que le state sera modifié, **un render sera automatiquement re-déclenché** pour refléter les modifications à l'écran !

Il est possible de définir le "state" par défaut soit en entête de la classe (comme dans l'exemple ci-dessus) soit dans le constructeur :

```
constructor(...args) {
    super(...args);
    this.state = {
        userIsConnected: false,
        user: null
    }
}
```

Il est possible de mettre à jour le "state" du composant en faisant appel à la méthode "setState", cette méthode prend en paramètre le nouveau state. En plus de mettre à jour le state, l'appel à "setState" va également déclencher un nouvel appel à la méthode "render" qui aura donc pour effet de mettre à jour l'affichage du composant à l'écran.

Il est également possible de passer à setState une function qui calcule le nouveau state plutôt qu'un objet littéral. La fonction passée en paramètre reçoit l'ancien state (`prevState`) et les props actuelles (`props`).

```
const computeState = ( prevState, props ){
    return { userIsConnected: !prevState.userIsConnected };
}
// ...

this.setState( computeState );
```

NB: Toutes les propriétés existantes du state qui ne sont pas redéfinies lors d'un appel à "setState" conservent leur valeur initiale.