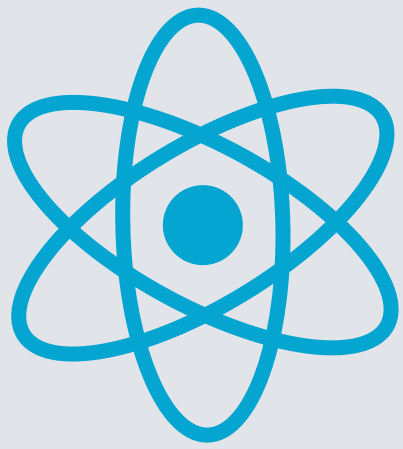


cours **React - J6**

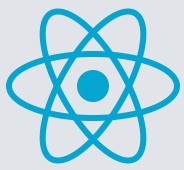
IG2I 2021

une idée derrière la tech

CE QU'IL FALLAIT RETENIR

- form non contrôlé = ref + input.value
- form contrôlé = state + onChange + setState
- fetch('/api/recipes/meth')

  .then(response => response.json())
  .then(data => this.setState({ recipe: data }))



React

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

6. HOOKS & REACT ROUTER

- Les Hooks c'est quoi ?
- useState & useEffect
- React Router
- Les hooks de React Router

HOOKS

Les hooks sont des fonctions qui permettent de :

- rendre les **function components "intelligents"**
- offrir les mêmes capacités que les class components

Notes :

Les hooks ont été introduits avec React 16.8 sorti en février 2019 (pour React Native c'est dans la version 0.59). Depuis, leur adoption a été massive et même si l'équipe de React a tout de suite recommandé de ne pas réécrire ses composants juste pour les convertir aux hooks, beaucoup ont succombé !

En effet, les hooks apportent beaucoup d'avantages et même si leur syntaxe peut sembler un peu étrange au premier abord la courbe d'apprentissage est assez rapide.

Au delà de avantages indiqués ci-dessus, la team de React indique que la programmation fonctionnelle est plus facile à comprendre que la POO, à la fois pour les humains ET pour les machines. A mon sens, tout dépend évidemment de l'expérience qu'un développeur a avec la POO, mais ceci dit même si l'on est plus à l'aise avec la programmation objet que fonctionnelle, les hooks apportent des avantages indéniables.

Vous trouverez la documentation des hooks sur https://reactjs.org/docs/hooks-intro.html. Je vous invite à regarder la vidéo de la React conf 2018 qui est mise en avant sur cette page, la présentation de Dan Abramov est très intéressante pour comprendre en quoi les hooks permettent de rendre nos composants plus "beaux". Si vous êtes pressés (la vidéo fait quand même 1h35 !) vous pouvez passer directement à 11m30s et arrêter à la fin de la présentation de Dan (aux alentours de 1h00m00s) https://youtu.be/dpw9EHDh2bM?t=687.

HOOKS : USESTATE

ajoute un state local dans un function component

```
import {useState} from 'react';
function Gale() {
    const [isDead, setIsDead] = useState(false);
    return (
        <>
            <h1>I am a {isDead ? 'dead' : ''} chemist</h1>
            <button onClick={() => setIsDead(true)}>
                shoot Gale in the head
            </button>
        </>
    )
}
```

Notes :

Le hook useState permet de créer des fonctions stateful. A chaque modification du state généré avec useState, React relancera la fonction du composant et donc le render.

USESTATE *vs* THIS.STATE


```
class Slider extends React.Component {
    state = {
        slides: [{id: 12},{id: 42},{id: 1337}],
        currentIndex: 0
    }
    constructor(...args) {
        super(...args);
        this.handleNextClick = this.handleNextClick.bind(this);
    }
    handleNextClick() {
        const {currentIndex, slides} = this.state;
        this.setState({currentIndex:(currentIndex + 1)%slides.length});
    }
    render() {
        const {currentIndex, slides} = this.state;
        return ( <button onClick={this.handleNextClick}>
            { slides[currentIndex].id }
        </button> );
    }
}
```

```
function Slider() {
    const slides = [{id: 12},{id: 42},{id: 1337}];
    const [currentIndex, setCurrentIndex] = useState(0);

    function handleNextClick() {
        setCurrentIndex((currentIndex + 1)%slides.length);
    }

    return ( <button onClick={handleNextClick}>
        { slides[currentIndex].id }
    </button> );
}
```

HOOKS : USEEFFECT

Permet de lancer un traitement après le render

```
import {useState, useEffect} from 'react';
function Gale() {
    const [isDead, setIsDead] = useState(false);

    useEffect(() => {
        console.log('Je me lance à chaque render');
    });

    /* return ... */
}
```

```
import {useState, useEffect} from 'react';
function Gale() {
    const [isDead, setIsDead] = useState(false);

    useEffect(() => {
        console.log('Je ne me lance qu\'après le 1er render');
    }, []);

    /* return ... */
}
```

```
import {useState, useEffect} from 'react';
function Gale() {
    const [isDead, setIsDead] = useState(false);

    useEffect(() => {
        console.log('Je me lance si isDead a changé : ' + isDead);
    }, [isDead]);

    /* return ... */
}
```

USEEFFECT *vs* COMPONENTDID...


```
class Slider extends React.Component {
    state = {
        slides: [{id: 12},{id: 42},{id: 1337}],
        currentIndex: 0
    }
    componentDidMount() {
        fetchPhoto();
    }
    componentDidUpdate(prevProps, prevState) {
        if (this.state.currentIndex != prevState.currentIndex) {
            fetchPhoto();
        }
    }
    fetchPhoto() {
        const {currentIndex, slides} = this.state;
        api.getPhotoById(slides[currentIndex].id)
            .then( data => {
                const clone = [...slides];
                clone[currentIndex] = data;
                this.setState({slides: clone});
            });
    }

    // ... constructor, handleNextClick et render
}
```

```
function Slider() {
    const [slides, setSlides] = useState([{id: 12},{id: 42},{id: 1337}]);
    const [currentIndex, setCurrentIndex] = useState(0);

    function fetchPhoto() {
        api.getPhotoById(slides[currentIndex].id)
            .then( data => {
                const clone = [...slides];
                clone[currentIndex] = data;
                setSlides(clone);
            });
    }
    useEffect(fetchPhoto, [currentIndex] ); // ne se relancera que si
                                            //currentIndex change !

    function handleNextClick() {
        setCurrentIndex(index => (index + 1)%slides.length);
    }

    return ( <button onClick={handleNextClick}>
```

```
            { slides[currentIndex].id }
        </button> );
    }
```

Notes :

Exemple en ligne du code précédent : https://codepen.io/uidlt/pen/GRqodoM?editors=0100

useEffect permet de déclencher des traitements (side effects) à chaque nouveau render. C'est l'équivalent du componentDidMount ET du componentDidUpdate réunis.

Le deuxième paramètre de useEffect, permet d'indiquer à React quand l'effet doit se lancer. Par défaut c'est à chaque re-render, mais si l'on passe un tableau on peut indiquer à React les dépendances du side effect. Si l'on passe dans ce tableau une référence à une variable, la fonction ne se relancera que lorsque la valeur de cette variable changera.

NB: si l'on ne passe aucun paramètre la fonction se relancera à chaque render

NB2: si l'on passe un tableau vide, la fonction ne se lancera qu'une fois, puis plus jamais (équivalent du componentDidMount)

NB3: la fonction que l'on passe en paramètre de useEffect peut retourner une fonction de cleanup qui sera exécutée avant chaque nouveau passage dans l'effect ET au unmount du composant. Cette feature est utile lorsqu'on utilise des fonctions comme setTimeout/setInterval ou des mécanismes de connexion à une source de données temps réel.

Pour plus d'infos sur le useEffect : https://reactjs.org/docs/hooks-effect.html et https://reactjs.org/docs/hooks-reference.html#useeffect

HOOKS : LES AUTRES HOOKS

- useRef() : référence vers un élément DOM
- useReducer() : version avancée du useState()
- useCallback() : création de fonction memoïzée
- useMemo() : mémoïzation du résultat d'un calcul intensif
- ...

Notes :

A propos des callbacks de memoization, je vous recommande la lecture de cet article :
https://medium.com/@sdolidze/react-hooks-memoization-99a9a91c8853

REACT ROUTER

- Standard du routing pour React
- Synchronise l'UI avec l'URL du navigateur

Notes :

Ajouter un système de routing basé sur la History API (manipulation de l'url du navigateur sans rechargement de page) dans sa SPA présente plusieurs avantages. Le principal est que cela rend transparent l'usage du site pour l'utilisateur par rapport à un site web classique.

Cela permet à l'utilisateur de mettre des pages en favoris, de modifier l'URL pour accèder à une autre page, et surtout d'utiliser les boutons précédent/suivant du navigateur.

Un autre avantage est que cela simplifie également la manière de gérer la navigation dans son application.

React router est aujourd'hui le standard du routing pour React. Il est efficace, complet et s'intègre très bien avec Redux.

SETUP : app.js

```
import { BrowserRouter } from 'react-router-dom';

ReactDOM.render(
    <BrowserRouter>
        <Navigator />
    </BrowserRouter>,
    document.querySelector('#appContainer')
);
```

Notes :

Pour pouvoir utiliser React Router dans notre appli, il faut instancier un "Router" qui va centraliser l'état et l'historique de navigation.

Il existe plusieurs types de "Router" mais le plus souvent on utilise le "BrowserRouter" qui se base sur l'URL du navigateur (History API) pour fonctionner.

CONFIGURER LES ROUTES


```
import { Switch, Route } from 'react-router-dom';

const Navigator = () => (
    <Switch>
        <Route exact path="/">
            <Home />
        </Route>
        <Route exact path="/users">
            <UserList />
        </Route>
        <Route exact path="/users/:id">
            <UserDetail />
        </Route>
    </Switch>
);
export default Navigator;
```

Notes :

Le routing s'appuie principalement sur le composant Route.

Le principe de ce composant est assez simple : si l'URL correspond à la props "path" du composant, alors le composant enfant de la Route est affiché.

De base, si l'URL correspond à plusieurs Route, elles seront toutes affichées. Pour palier ce problème et n'afficher qu'une seule route à la fois, il est possible de les englober dans le composant "Switch" (seule la première sera affichée)

RÉCUPÉRER DES PARAMÈTRES

La route

```
<Route exact path="hello/:firstName-:lastName">
    <Hello />
</Route>
```

Le composant

```
import {useParams} from 'react-router-dom';
function Hello() {
    const { firstName, lastName } = useParams();
    return <p>Bonjour {firstName} {lastName} !</p>
}
```

Notes :

Tous les paramètres variables de l'URL peuvent être récupérés à l'aide du hook `useParams()`.

Les valeurs peuvent ensuite être utilisées dans le render pour l'affichage, ou alors pour faire une requête AJAX, etc.

<LINK> / <NAVLINK>

```
import { Link, NavLink } from 'react-router-dom';

const Menu = () => (
    <nav>
        <Link to="/">Accueil</Link>
        <Link to="/hello/thomas-fritsch">Bonjour</Link>
        <NavLink to="/about">A propos</NavLink>
    </nav>
);
export default Menu;
```

Notes :

Le composant Link prend une props "to" à laquelle il faut passer l'URL vers laquelle naviguer. React Router se charge de modifier l'adresse affichée dans le navigateur, et de re-rendre les composants `<Route>` concernés par le changement.

Le composant NavLink est un composant Link qui a la particularité de s'ajouter la classe CSS "active" lorsque l'url de la page correspond à son paramètre `to="..."`.

LANCER UNE NAVIGATION


```
import { useHistory } from 'react-router-dom'; // action creator

function MyForm() {
    const history = useHistory();
    return (
        // Au submit on déclenche la navigation vers une autre page
        <form onSubmit={() => history.push('/hello/thomas-fritsch')}>
            {/* ... reste du formulaire */}
        </form>
    );
}
```

Notes :

Le hook `useHistory` retourne un objet qui permet à un composant de déclencher la navigation vers une autre page grâce à la méthode `history.push()`.

Cet objet history dispose en outre de tout un tas de propriétés et de méthodes qui permettent de gérer la navigation (push, goBack, goForward, replace, ...) : https://reactrouter.com/core/api/history