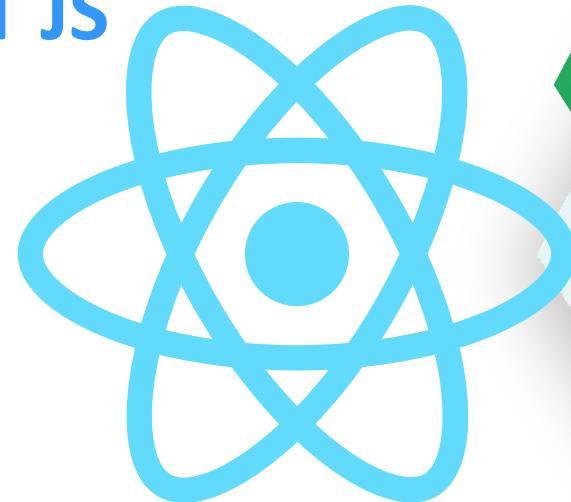




INTRODUCTION À REACT JS



SOMMAIRE

- [Introduction](#)
 - [Présentation de React](#)
 - [Environnement de développement](#)
 - [Notions de base :](#)
 - [JSX](#)
 - [Components](#)
 - [Props](#)
 - [State](#)
 - [Life Cycle](#)
 - [Fondamentaux :](#)
 - [Événements](#)
 - [Affichage conditionnel](#)
 - [Modifier les Props](#)
 - [Listes](#)
 - [Fragments](#)
 - [useEffect](#)
 - [Exercice](#)
 - [Librairies et Add-on :](#)
 - [VsCode](#)
 - [React Dev Tool](#)
 - [Install](#)
 - [Formik / Yup](#)
 - [Exercice](#)
 - [Axios](#)
 - [Routing :](#)
 - [Intro](#)
 - [Base du Routing](#)
 - [NavBar](#)
 - [Redirections JS](#)
 - [Mini-Projet](#)
-
- [Bonnes Pratiques :](#)
 - Architecture :
 - [Intro](#)
 - [Features](#)
 - [Couches](#)
 - [Concepts Avancés :](#)
 - Redux :
 - [Intro](#)
 - [Installation](#)
 - [Store](#)
 - [Slices](#)
 - [Reducers](#)
 - [useSelector](#)
 - [useDispatch](#)

INTRODUCTION

OBJECTIF GÉNÉRAL

Prendre en main l'une des bibliothèques JavaScript les plus utilisées pour créer des interfaces utilisateurs.

OBJECTIFS SPÉCIFIQUES

- Découper l'interface utilisateur avec les composants
- Passer des informations (données ou instructions) d'un composant à son composant fils avec « props »
- Gérer l'état local d'un composant avec « state »
- Afficher une liste de composants avec map()
- Afficher un composant en fonction de l'état de l'application
- Interagir avec un utilisateur grâce à la gestion des événements
- Interagir avec un utilisateur par le biais des formulaires
- Communiquer avec un serveur HTTP avec AJAX
- Afficher des vues en fonction de l'URL avec le routage
- Mettre en forme un composant

PRÉSENTATION DE REACT JS

PRÉSENTATION DE REACT

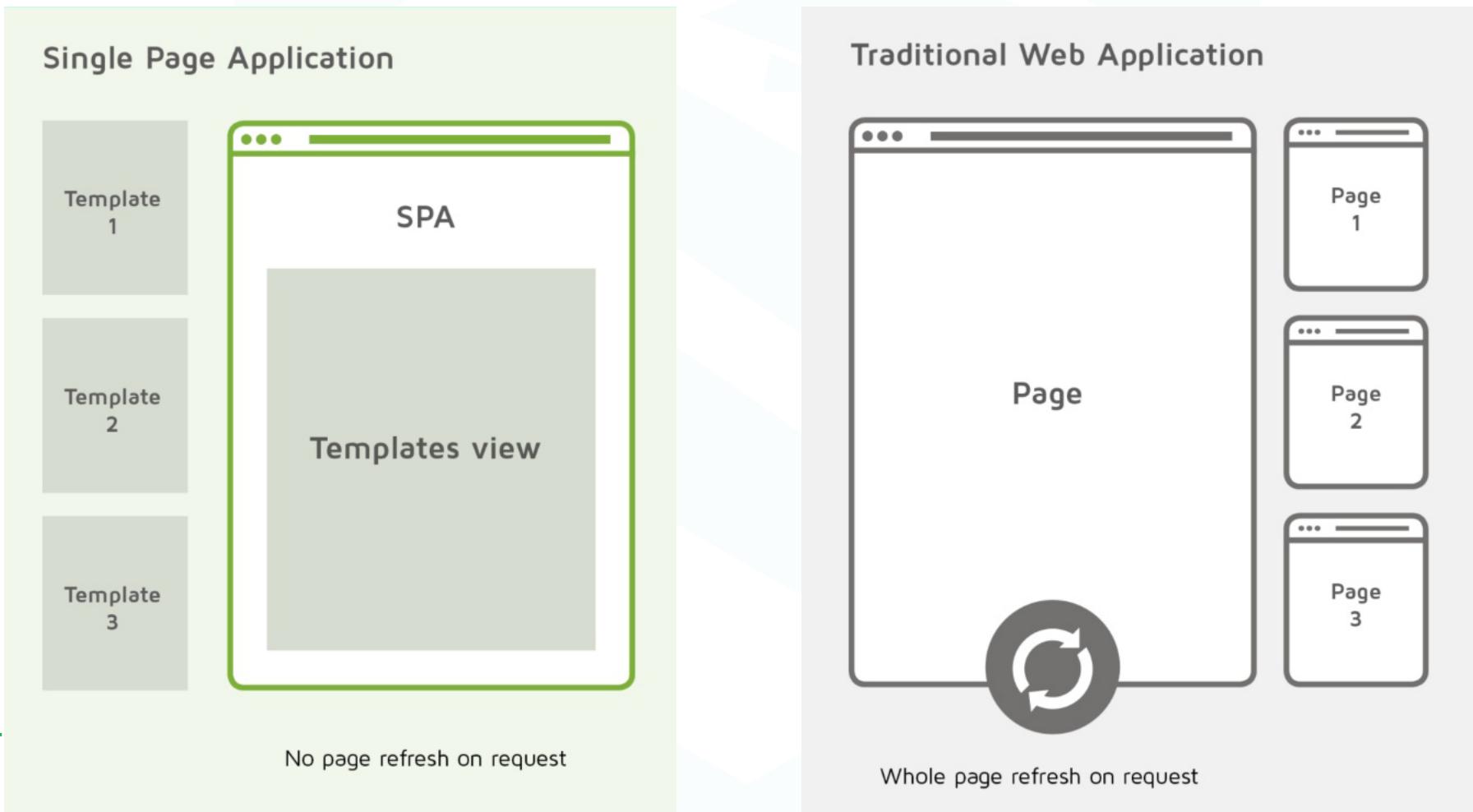
- Problématique : comment développer efficacement (rapidité, facilité, portabilité du JS, ...) une application Web ?
- Solution : ensemble de classes, fonctions et utilitaires « prêts à l'emploi »
- React : bibliothèque JavaScript open source depuis 2013 pour la création d'interfaces utilisateur
- Crée par Jordan Walke, développeur chez Facebook, fin 2011
- 2012 : Instagram
- 2015 : React-Native
- 2023 : aujourd'hui React 18

PRÉSENTATION DE REACT

- Quelques principes de React :
 - DOM virtuel : arborescence d'objets JS en mémoire utilisée pour lister les éventuelles modifications de la vue afin d'éviter de recréer entièrement le DOM du navigateur
 - Crédit de composants réutilisables, avec en entrée des données, pouvant changer au cours du temps
 - Usage : Application monopage (SPA) ou mobile

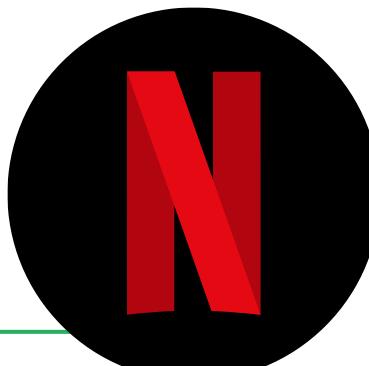
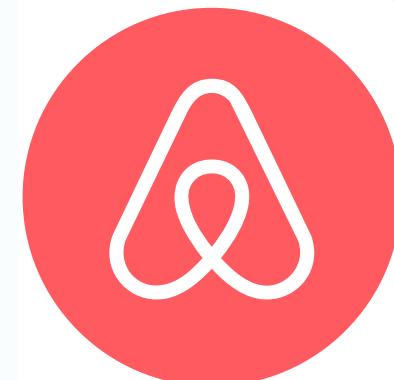
PRÉSENTATION DE REACT

Single Page Application



PRÉSENTATION DE REACT

React est utilisé notamment par :



ENVIRONNEMENT DE DÉVELOPPEMENT

ENVIRONNEMENT DE DÉVELOPPEMENT

Installer les prérequis node.js et NPM

Télécharger et installer la dernière version LTS de Node.js à partir de <https://nodejs.org/en/download/>

Méthode classique :
(déprécié)

```
npx create-react-app my-first-react-app
```

Méthode moderne :

```
npm create vite my-react-app -- --template react
```

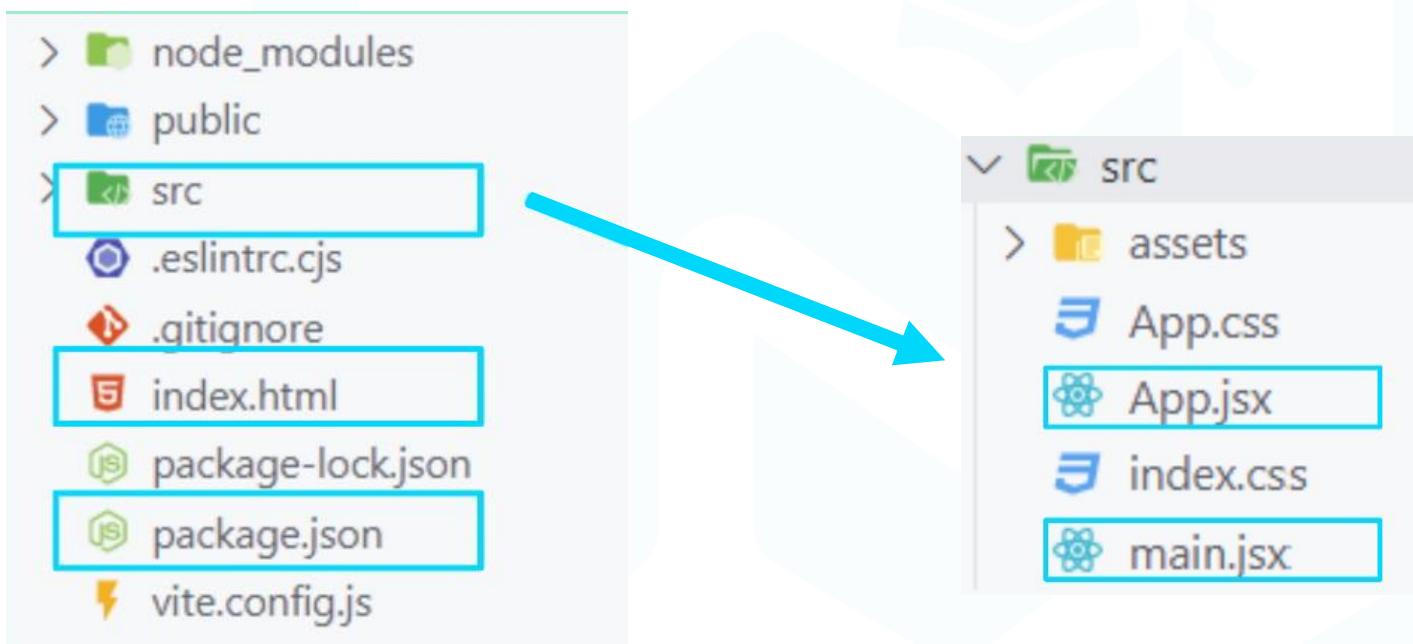
```
cd my-react-app
```

```
npm install
```

```
npm run dev
```

```
VITE v4.3.9 ready in 6019 ms  
→ Local: http://localhost:5173/  
→ Network: use --host to expose  
→ press h to show help
```

ENVIRONNEMENT DE DÉVELOPPEMENT ARCHITECTURE



NOTIONS DE BASE

JSX : PRÉSENTATION

- JSX (JavaScript eXtensible Markup Language) : syntaxe de type HTML / XML qui facilite l'écriture du HTML / XML dans du JS

- JSX est destiné au transpilateur (comme Babel) pour être traduit en objets JavaScript standard

- Exemple en JSX :

```
const greeting = <h1 className="speak">Hello, world!</h1>;
```

- Transpilation de l'exemple :

```
const greeting = React.createElement(  
  'h1', {className: 'speak'}, 'Hello, world!'  
,);
```

- Objet ou élément React créé par l'exemple :

```
const greeting = {  
  type: 'h1',  
  props: {  
    className: 'speak',  
    children: 'Hello, world!'  
  }  
};
```

JSX : TYPES D'ÉLÉMENT

- Eléments React : briques élémentaires constitutifs et visibles des applications React.
- Les éléments React sont retournés par les composants (à voir plus loin)
- Élément natif : nom commençant par une lettre minuscule
Exemple : <h1 ...
- Élément défini ou importé dans le fichier JavaScript : nom commençant par une lettre majuscule
Exemple : <NewsFeed ...

JSX : ATTRIBUT

- Attribut : paire (nom,valeur) qui caractérise un élément
- Les attributs fournissent des valeurs de configuration d'un élément
- Nomenclature des attributs : éviter les mots clé JS et utiliser le camelCase
Ex : className au lieu de class
htmlFor au lieu de for
- Types de valeur d'un attribut :
 - Littéral chaîne de caractères
Ex : <MyComponent user="*John Doe*" />
 - Booléen : par défaut
Ex : <MyComponent *connected* />
 - Expression JS
Ex : <MyComponent total={*1 + 2 + 3 + 4*} />

JSX : TYPES D'ENFANT

- Enfant : contenu présent entre 2 balises (ouvrante et fermante)
- Littéral chaîne de caractères

Ex : <MyComponent>*Bonjour le monde !*</MyComponent>

- Élément JSX

Ex : <header>

 <Nav />

 <SearchBox />

 </header>

- Expression JavaScript à enrober avec des accolades { }

Ex : <h1> *{i == 1 ? 'True!' : 'False'}*</h1>

- Les types d'enfant peuvent être mixés

JSX : SYNTAXE À RESPECTER

- One Top Level Element : le code HTML doit être enveloppé dans un seul élément parent
Ex :

```
<div>  
  <h2>Content</h2>  
  <p>This is the content!!!</p>  
</div>
```

- Tout élément doit être fermé

Ex : `<input type="text" />;`

- Plusieurs lignes de HTML doivent être mises entre parenthèses

Ex :

```
(  
  <ul>  
    <li>Apples</li>  
    <li>Bananas</li>  
  </ul>  
)
```

COMPONENTS ILLUSTRATION

<input type="text" value="Search..."/>	
<input type="checkbox"/> Only show products in stock	
Name	Price
Sporting Goods	
Football	\$49.99
Baseball	\$9.99
Basketball	\$29.99
Electronics	
iPod Touch	\$99.99
iPhone 5	\$399.99
Nexus 7	\$199.99

COMPOSANTS PRÉSENTATION

- Composant : partie de l'application. Les composants permettent de découper l'interface utilisateur en éléments indépendants et réutilisables. Code JS qui renvoie des éléments React ou d'autres composants décrivant ce qui doit apparaître à l'écran.
- Deux types de composant:
 - Fonction (à préconiser):
Ex : `const Welcome = () => { return <h1>Hello World !</h1> }`
 - Classe (moins utilisé):
Ex : `class Welcome extends React.Component {
 render() {
 return <h1>Hello World !</h1>;
 }
}`

COMPOSANTS EXEMPLE

```
Address.jsx Account.jsx X App.jsx  
my-react-app > src > components > Account.jsx > ...  
1 import React from 'react'  
2  
3 const Account = () => {  
4   return (  
5     <div>John Doe</div>  
6   )  
7 }  
8  
9 export default Account
```

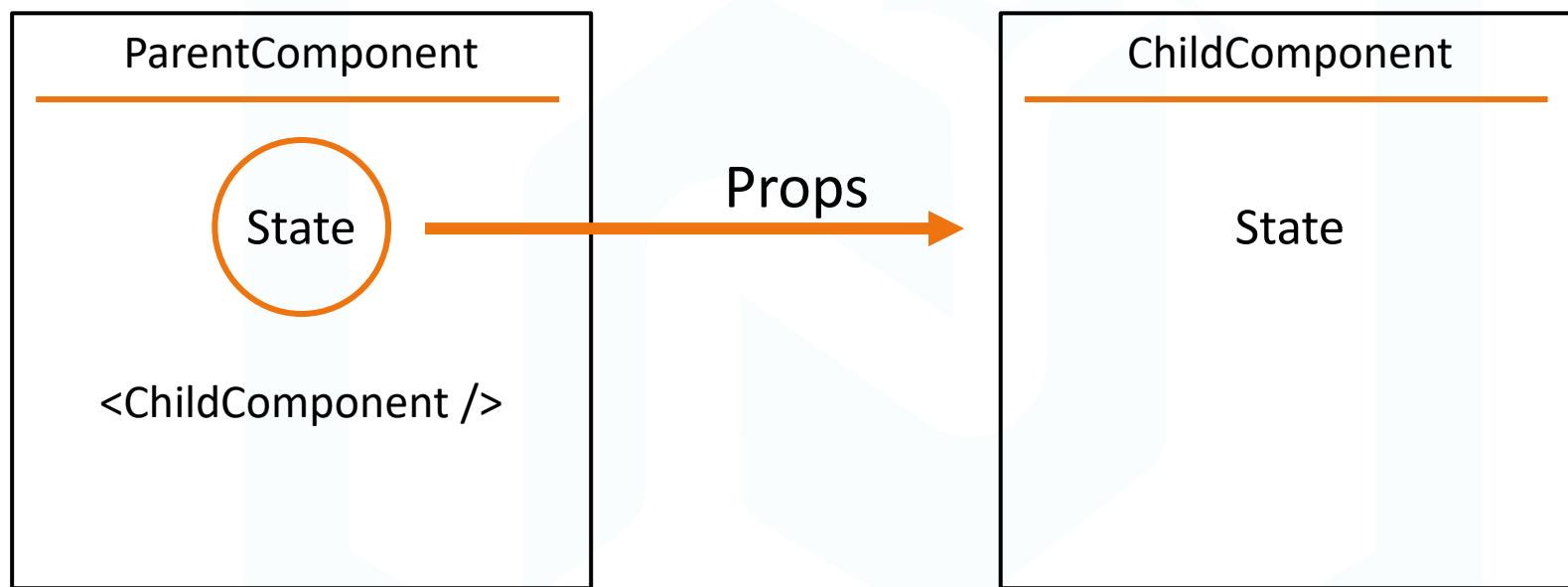
```
Address.jsx X Account.jsx App.jsx  
my-react-app > src > components > Address.jsx > ...  
1 import React from 'react'  
2  
3 const Address = () => {  
4   return (  
5     <div>10 rue de toto - 59000 Lille</div>  
6   )  
7 }  
8  
9 export default Address
```

```
Address.jsx Account.jsx X App.jsx  
my-react-app > src > App.jsx > ...  
1 import React from 'react'  
2 import Account from './components/Account'  
3 import Address from './components/Address'  
4  
5 const App = () => {  
6   return (  
7     <div>  
8       <h1>Mon Application</h1>  
9       <Account />  
10      <Address />  
11    </div>  
12  )  
13}  
14  
15 export default App
```

PROPS PRÉSENTATION

- Props (ou propriété) : donnée ou gestionnaire d'événement fourni à un composant par son composant parent
- Syntaxe d'envoi d'une propriété par un composant parent :
<ComposantFils nomPropriété=valeur />
- Syntaxes d'accès à une propriété par un composant fils
props.nomPropriété
avec props paramètre de la fonction
- Les props sont ensuite utilisées par le composant parent ou transmises aux composants enfants.
- Les props sont en lecture seule (**immuables**)

PROPS PRÉSENTATION



PROPS
EXEMPLE

```
const Address = (props) => {  
  return (  
    <div>{props.address}</div>  
  )  
}
```

```
const Account = ({name}) => {  
  return (  
    <div>{name}</div>  
  )  
}
```

```
const App = () => {  
  const user = {  
    completeName : "John Doe",  
    address : "1 rue de Toto - 59000 Lille",  
  };  
  
  return (  
    <div>  
      <h1>Mon Application</h1>  
      <div>  
        <Account name={user.completeName} />  
        <Address address={user.address} />  
      </div>  
    </div>  
  );  
};
```

STATE

LE HOOK useState

```
import React, {useState} from 'react'
import Account from './components/Account'
import Address from './components/Address'

const App = () => {
  // Si modifié, mettra à jour le composant visuellement sur le navigateur
  const [user] = useState( {
    completeName : "John Doe",
    address : "1 rue de Toto - 59000 Lille",
  });
  // Si modifié, ne mettra pas à jour le composant visuellement sur le navigateur

  let count = 0;

  return (
    <div>
      <h1>Mon Application</h1>
      <div>
        <Account name={user.completeName} />
        <Address address={user.address} />
      </div>
      <p>Compteur : {count}</p>
    </div>
  );
};

export default App
```

- Variable de la classe
- Ne peut être modifier que dans sa classe
- Rafraichi le composant
- Tous types de variable :
 - "type primitif"
 - objet
 - tableau
 - composant
 - etc ...

EXERCICE

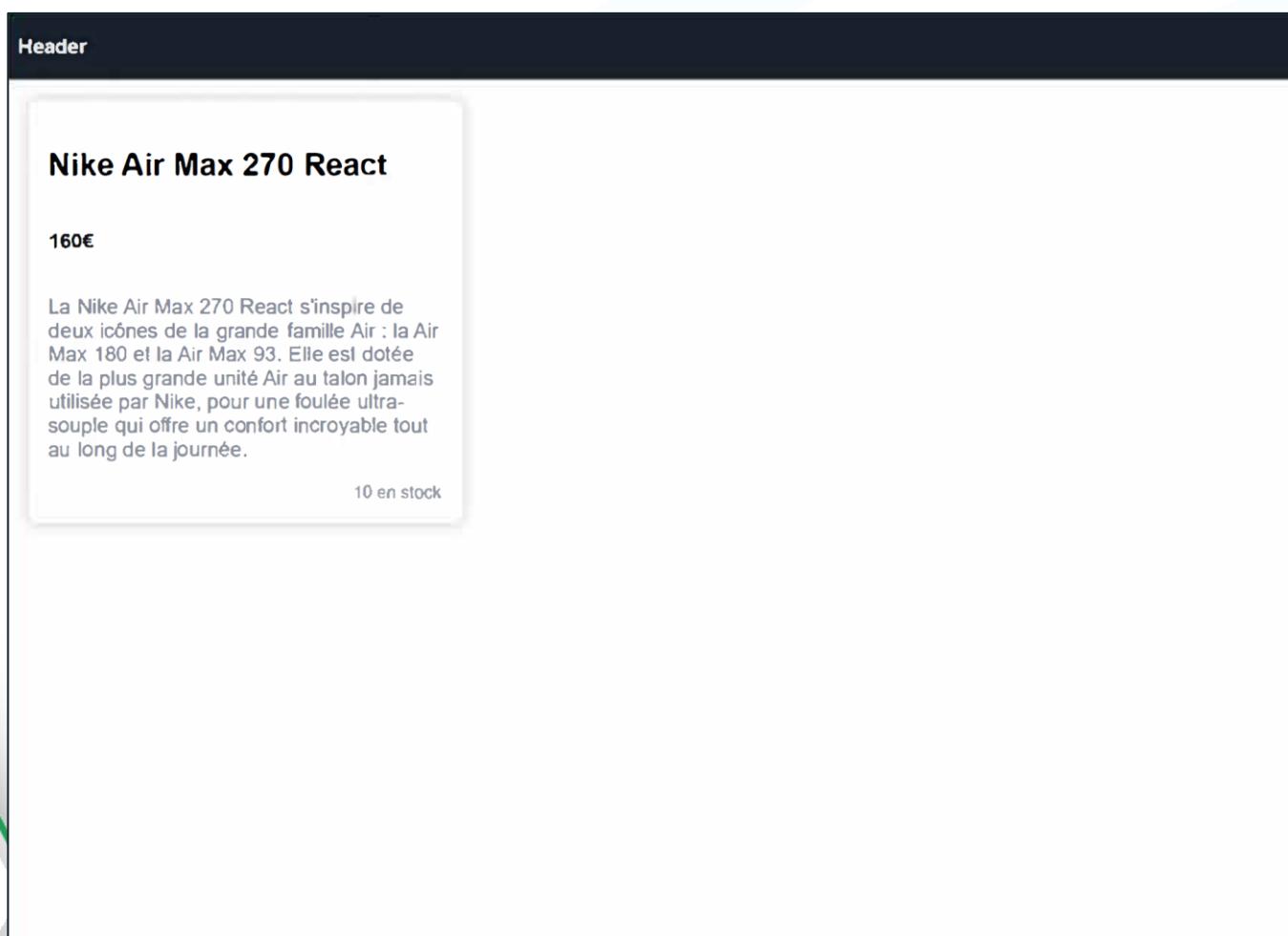
Header

Nike Air Max 270 React

160€

La Nike Air Max 270 React s'inspire de deux icônes de la grande famille Air : la Air Max 180 et la Air Max 93. Elle est dotée de la plus grande unité Air au talon jamais utilisée par Nike, pour une foulée ultra-souple qui offre un confort incroyable tout au long de la journée.

10 en stock



- Créer un nouveau projet
- Créer un composant Header pour afficher une navbar dans l'application
- Créer un composant Product avec les props:
 - name
 - price
 - description
 - stock
- Les données du produit seront dans un state dans votre composant App
- Vous pouvez utiliser du pure CSS ou BOOTSTRAP pour le style

FONDAMENTAUX

LIFE CYCLE

Initialization

setup props and state

Mounting

componentWillMount

render

componentDidMount

Updation

props

componentWillReceiveProps

shouldComponentUpdate

true
false

componentWillUpdate

render

componentDidUpdate

states

shouldComponentUpdate

true
false

componentWillUpdate

render

componentDidUpdate

Unmounting

componentWillUnmount

LIFE CYCLE

Initialization

setup props and state

Mounting

—componentWillMount

render

componentDidMount

Updation

props

componentWillReceiveProps

shouldComponentUpdate

true ✗ false

componentWillUpdate

render

componentDidUpdate

states

shouldComponentUpdate

true ✗ false

componentWillUpdate

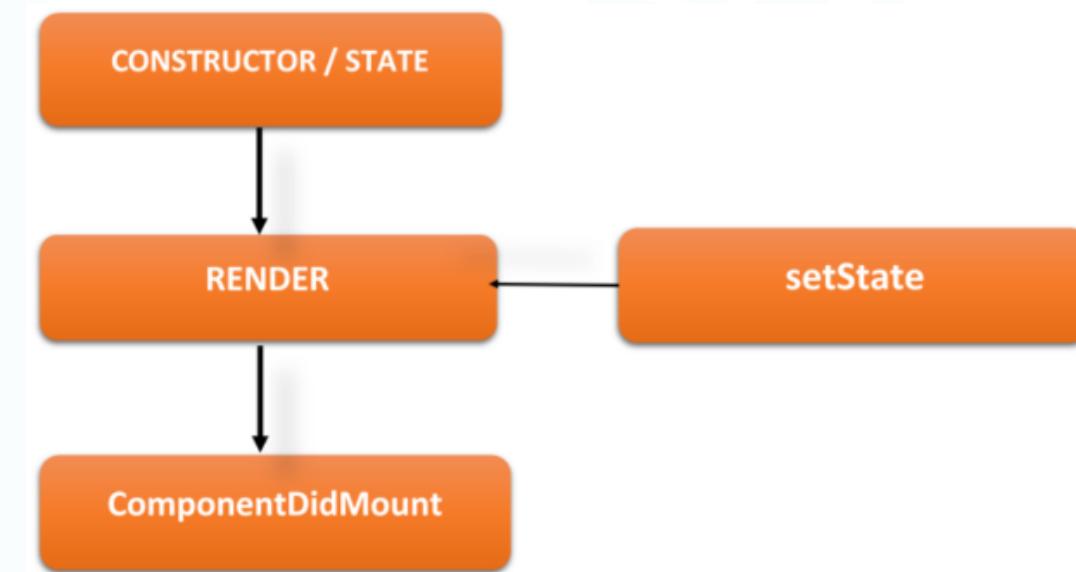
render

componentDidUpdate

Unmounting

—componentWillUnmount

LIFE CYCLE



ÉVÉNEMENTS PRÉSENTATION

- Événements React = événements HTML
Un événement écoute une interaction utilisateur, et s'enclenchera automatiquement.
- L'écoute se fait inline, c'est à dire sur l'élément JSX
- Syntaxe (en camelCase) : *onNomEvenement={handler}*
- Comment lier le handler avec son contexte ?
 - Définition en fonction fléchée
handler = (...) => { ... }

ÉVÉNEMENTS

EXEMPLE

```
const Counter = () => {
  let count = 0;

  const increment = () => {
    count++;
  };

  console.log(count);

  return (
    <div>
      <p>Le compteur est à : {count}</p>
      <button onClick={increment}>Incrémenter</button>
    </div>
  );
};

export default Counter
```

- Count va bien se modifier
 - Le composant ne sera pas rafraîchi !
 - Aucun changement visuel sur le navigateur
- Il faut utiliser le hook : *useState*

ÉVÉNEMENTS EXEMPLE

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

const Counter = () => {
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount(prevCount => prevCount + 1);
  };

  return (
    <div>
      <p>Le compteur est à : {count}</p>
      <button onClick={increment}>Incrémenter</button>
    </div>
  );
}

export default Counter
```

- Toujours prendre la "prev" valeur d'un state avant de le modifier !
- La fonction peut directement être mise dans l'événement.
- Attention ! setCount est asynchrone !

AFFICHAGE CONDITIONNEL

PRÉSENTATION

- Affichage conditionnel : afficher du contenu en fonction de l'état de l'application
- Structure conditionnelle if :
 - Ex : (afficher « Loading... » lors du chargement des données) :

```
if(isLoading) { return <p>Loading...</p> }
return <p>Contenu de ma page</p>
```
- Opérateur ternaire :
 - condition ? trueValue : falseValue.
 - Ex :

```
return <div>{isLoading ? <p>Loading...</p> : null}</div>;
```
- Opérateur booléen : &&
 - true && expression vaut expression
 - false && expression vaut false.
 - Ex :

```
return <div>{isLoading && <p>Loading...</p>}</div>;
```

AFFICHAGE CONDITIONNEL PRÉSENTATION

```
const App = () => {
  const isAdmin = false;

  if (isAdmin) {
    return (
      <div>
        <h3>My App</h3>
        <Counter />
        <p>Je suis un admin</p>
      </div>
    );
  } else {
    return (
      <div>
        <h3>My App</h3>
        <Counter />
        <p>Je ne suis pas un admin</p>
      </div>
    );
  }
};
```

- Structure conditionnelle if

A utiliser seulement si les return sont complètement différents !

Sinon, préférer une autre méthode.

AFFICHAGE CONDITIONNEL PRÉSENTATION

```
const App = () => {
  const isAdmin = false;

  return (
    <div>
      <h3>My App</h3>
      <Counter />
      {isAdmin ? (
        <p>Vous êtes admin</p>
      ) : (
        <p>Vous n'êtes pas admin</p>
      )}
    </div>
  );
};

export default App;
```

- Opérateur ternaire

Préférez mettre la condition seulement sur la partie qui change.

AFFICHAGE CONDITIONNEL PRÉSENTATION

```
import Counter from "./components/Counter";\n\nconst App = () => {\n  const isAdmin = false;\n\n  return (\n    <div>\n      <h3>My App</h3>\n      <Counter />\n      {isAdmin && <p>Vous êtes admin</p>}\n    </div>\n  );\n};\n\nexport default App;
```

- Opérateur booléen &&

Si affichage pour seulement l'une des conditions, préférez l'usage du '&&'.

MODIFIER LES PROPS

ON NE PEUT PAS MODIFIER UNE PROP !*

Mais on peut tricher..

Indirectement, un enfant peut changer une props.

Il faut pour cela que le parent lui donne en props une fonction permettant de changer la variable souhaitée.

MODIFIER LES PROPS

```
import { useState } from "react";
import Counter from "./components/Counter";

const App = () => {
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount((prevCount) => prevCount + 1);
  };

  return (
    <div>
      <h3>My App</h3>
      <Counter count={count} incrementCount={increment} />
    </div>
  );
};

export default App;
```

```
import React from "react";

const Counter = ({ count, incrementCount }) => {
  return (
    <div>
      <p>Le compteur est à : {count}</p>
      <button onClick={incrementCount}>Incrémenter</button>
    </div>
  );
};

export default Counter;
```

MODIFIER LES PROPS

EXERCICE

Créer 3 composants :

- Bouton Login
- Bouton Logout
- Composant Parent

Dans le composant Parent, selon un Booléen (isLogged) en state, affiche le bouton Login ou Logout. Où, si on clique dessus, permet de se logguer ou se délogguer selon le bouton.

On utilisera donc la gestion d'événements, l'affichage conditionnelle et la modification de Props.

LISTES PRÉSENTATION

- La plupart des app Web listent des données
- .map(): méthode de Array Javascript qui
 - parcourt un tableau pour appliquer un même traitement sur chaque élément
 - retourne le tableau des résultats du traitement

Exemple :

```
doubled = [1,2,3,4].map(val => 2*val)      // [2,4,6,8]
```

- Une syntaxe d'une liste de composants

```
liste.map(item => (  
  <NomComposant  
    key={item.id}  
    prop1={item.prop1}  
    prop2={item.prop2} ...  
  />  
)
```

LISTES EXEMPLE

```
const List = () => {
  const arr = [1, 2, 3, 4, 5];

  return (
    <ul>
      {arr.map((item) => (
        <li>{item}</li>
      ))}
    </ul>
  );
}

export default List;
```

```
const List = () => {
  const arr = [
    { firstName: "John", lastName: "Doe" },
    { firstName: "Jane", lastName: "Doe" },
    { firstName: "Janice", lastName: "Joplin" },
  ];

  return (
    <ul>
      {arr.map((item) => (
        <li>
          {item.firstName} {item.lastName}
        </li>
      ))}
    </ul>
  );
}

export default List;
```

LISTES EXEMPLE

Attention !

Dans la console du navigateur on peut avoir :

Warning : Each child in a list shoud have a unique "key" prop

Il faut que chaque élément d'une liste ait une propriété "Key" **unique**.

LISTES CLÉ (KEY)

- Chaque item de la liste doit être associé à une clé unique et stable
- Les clés permettent à React de réafficher plus rapidement la liste en cas de modification (ajout, suppression, ...)
- Exemple d'une insertion d'un élément au début d'une liste sans clé :

DOM virtuel à modifier

```
<ul>
  <li>Jean</li>
  <li>Fred</li>
</ul>
```



DOM virtuel modifié

```
<ul>
  <li>Bob</li>
  <li>Sophie</li>
  <li>Julie</li>
</ul>
```

Jean et Fred sont d'abord remplacés par Bob et Sophie, puis Julie est rajoutée.

LISTES EXEMPLE

```
const List = () => {
  const arr = [
    { firstName: "John", lastName: "Doe" },
    { firstName: "Jane", lastName: "Doe" },
    { firstName: "Janice", lastName: "Joplin" },
  ];

  return (
    <ul>
      {arr.map((item, index) => (
        <li key={index}>
          {item.firstName} {item.lastName}
        </li>
      ))}
    </ul>
  );
}

export default List;
```

FRAGMENTS

- Un component doit avoir une balise principale contenant toutes les autres balises
- Peut être n'importe quel type de balise
- Fragment est une sorte de balise fictive permettant de regrouper plusieurs balises qu'on ne voudrait pas mettre dans une balise globale (par exemple : plusieurs balises ``)
- Deux manières :
 - `<React.Fragment> ... </React.Fragment>`
 - `<> ... </>`

FRAGMENTS

EXEMPLE

```
const App = () => {
  return (
    <>
      <h3>My App</h3>
      <p>Hello, my name is Nemo</p>
    </>
  );
};

export default App;
```

HOOK useEffect

Active automatiquement une exécution choisie :

- S'active au moins une fois, au chargement du composant
- Peut-être réenclenché au besoin, pour automatiser une exécution selon un changement du composant (par exemple modification d'un state).

HOOK
useEffect

Bloc d'exécution

```
import {useEffect} from 'react'

const App = () => {
  useEffect(() => {}, []);
  return <div>App</div>;
}

export default App;
```

Dépendances,
Pour
réenclencher le
useEffect

HOOK useEffect

```
import { useEffect, useState } from "react";

const App = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    // Effectue une exécution quand le composant est monté,
    // puis le refait à chaque fois que count change
    console.log("exécution de useEffect");
  }, [count]);

  return(
    <div>
      <h1>Compteur : {count}</h1>
      <button onClick={() => setCount(count + 1)}>Incrémenter</button>
    </div>
  );
}

export default App;
```

EXERCICE

Créer une série de 4 composants pour recréer cette image.

Cela n'a pas besoin d'être fonctionnel, mais juste visuel.

Search...	
<input type="checkbox"/> Only show products in stock	
Name	Price
Football	\$49.99
Baseball	\$9.99
Basketball	\$29.99
iPod Touch	\$99.99
iPhone 5	\$399.99
Nexus 7	\$199.99

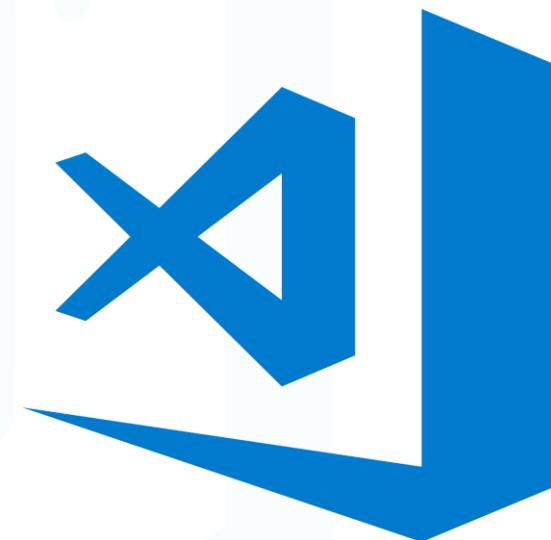
EXERCICE

```
const products = [  
    {category: "Sporting Goods", price: "$49.99", stocked: true, name: "Football"},  
    {category: "Sporting Goods", price: "$9.99", stocked: true, name: "Baseball"},  
    {category: "Sporting Goods", price: "$29.99", stocked: true, name: "Basketball"},  
    {category: "Electronics", price: "$99.99", stocked: true, name: "iPod Touch"},  
    {category: "Electronics", price: "$399.99", stocked: true, name: "iPhone 5"},  
    {category: "Electronics", price: "$199.99", stocked: true, name: "Nexus 7"}  
];
```

LIBRAIRIES

VISUAL STUDIO CODE EXTENSIONS UTILES

- **Auto Import - ES6, TS, JSX, TSX**
- Auto Rename Tag
- Beautify
- **ESLint**
- Highlight Matching Tag
- Material Icon Theme
- **Prettier - Code formatter**



REACT DEVELOPPER TOOLS

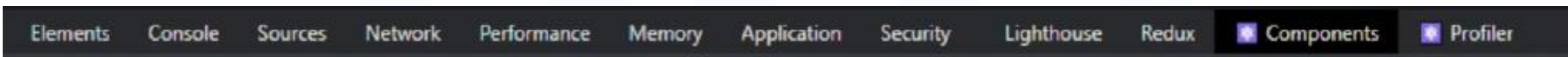
Extension navigateur :



Offered by: Facebook

★★★★★ 1,295 | [Developer Tools](#) | 2,000,000+ users

Dans la console du navigateur :



Permet de voir tous les composants, ainsi que leur STATE et PROPS en temps réel.

INSTALLER UNE LIBRAIRIE

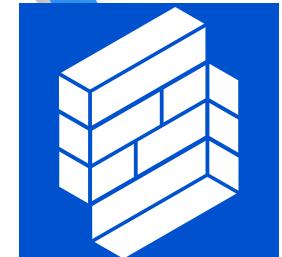
Installe la librairie localement pour le projet

```
npm install nom_de_la_library
```

FORMIK / YUP

Formik ou les formulaires React simplifiés

- Les formulaires avec React, sans librairie, peuvent être compliqués
- Formik est là pour les rendre plus simples, en gérant tout le côté répétitif de la création de Formulaires
- Léger, rapide et populaire



```
npm install formik
```

Yup ou la gestion de validité de formulaire simplifiée

- La gestion de validité d'un formulaire peut être longue et laborieuse
- Yup (couplé avec Formik) permet de faciliter cela en demandant simplement quels types de validation on veut et les messages d'erreurs à afficher
- Formik a été pensé pour fonctionner avec Yup
- Permet de très nombreux contrôles (natif de Yup, ou créé soi-même)

```
npm install yup
```

FORMIK

Valeurs initiale du formulaire sous forme d'objet.
Peut être des valeurs vides ou non, peut contenir des valeurs non utilisées dans le formulaire (exemple: id)

Méthode qui sera enclenchée en validation de formulaire. Récupère 'values' qui sont dans le même format que les initialValues. Récupère également différentes fonctionnalités de formik (comme resetForm)

Équivalent du input, a notamment les mêmes props.
Il faut que le 'name' corresponde à un champ des values du formulaire (ici firstName).
Par défaut est un type 'text'.

```
import { Field, Form, Formik } from "formik";

const SimpleForm = () => {
  const initialValues = {
    firstName: "",
    lastName: ""
  };

  const submit = (values) => {
    console.log(values);
  };

  return (
    <Formik initialValues={initialValues} onSubmit={submit}>
      <Form>
        <Field name="firstName" />
        <Field name="lastName" />
        <button type="submit">Submit</button>
      </Form>
    </Formik>
  );
};

export default SimpleForm;
```

FORMIK

component = { MyCustomInput }

FORMIK

Documentation officielle: <https://formik.org/docs/overview>

- Plusieurs types d'input différents
- Gérer les listes/array
- TypeScript
- ReactNative
- Optimisation
- isSubmitting
- Refresh
- etc...

YUP

SCHÉMA DE VALIDATION

```
import * as yup from 'yup';

const validationSchema = yup.object().shape({
  firstName: yup.string().required("Champs requis"),
  lastName: yup.string().required("Champs requis"),
});
```

YUP

SCHÉMA DE VALIDATION

Documentation: <https://github.com/jQuense/yup>

- Tous types (string, number, date, array, etc...)
- min, max
- positive, negative
- lessThan, moreThan
- Contrôle entre champs (utile pour des champs date de début et date de fin)
- regex
- etc...

YUP

SCHÉMA DE VALIDATION

```
const validationSchema = Yup.object({
    dateCreation: Yup.date().required("Champ obligatoire"),
    dateFinPrevisionnel: Yup.date().required("Champ obligatoire")
        .when(
            'dateCreation',
            (dateCreation, validationSchema) => (
                dateCreation
                &&
                validationSchema.min(dateCreation, "Date de fin doit être supérieur ou égale à date de début")
            ),
        ),
    nbLivrableAttendus: Yup.number().integer().moreThan(-1,"doit etre positif"),
    libelle: Yup.string().required("Champs obligatoire").max(254 , "maximum 254 caractères"),
    dureePrevisionnelleMinutes: Yup.number().integer().moreThan(-1,"doit etre positif"),
    listeRessources : Yup.array().compact().of(Yup.object({
        libelle: Yup.string().required("Champs obligatoire"),
        localisation: Yup.string().required("Champs obligatoire"),
    })),
    seanceSessionId: Yup.number().required("Champs obligatoire"),
    commentaire: Yup.string().max(254 , "maximum 254 caractères")
})
```

YUP AVEC FORMIK

```
const validationSchema = Yup.object().shape({
  firstName: Yup.string().required("Champs requis"),
  lastName: Yup.string().required("Champs requis"),
});

return (
  <Formik
    initialValues={initialValues}
    onSubmit={submit}
    validationSchema={validationSchema}
  >
    <Form>
      <Field name="firstName" />
      <ErrorMessage name="firstName" />
      <Field name="lastName" />
      <ErrorMessage name="lastName" />
      <button type="submit">Submit</button>
    </Form>
  </Formik>
);
```

Tout comme *Field*, *ErrorMessage* doit avoir le nom du champs du formulaire que l'on veut gérer.

EXERCICE

Informations

Mr Ms ON/A

Firstname

Lastname

Birthday yyyy-mm-dd

Address

Number

Road

City

ZipCode

Contact

Phone

Email

Tous les champs sont requis,
et devront avoir les validations appropriées :

- max 50 caractères
- Type de champs (number, string, email, ...)
- Date de naissance dans le passé
- etc

AXIOS

INTRODUCTION

Axios, ou comment faire des appels vers une API back-end

- Une appli Web (web-service) n'est pas que du Front, mais doit aussi faire appel à une ou plusieurs API Back pour récupérer des données.
- Axios est une librairie permettant d'aider à faire cela
- Asynchrone, système de promesse (attendre de récupérer la réponse)

```
npm install axios
```

AXIOS

GET / DELETE

```
import axios from "axios";

const TestPostApi = () => {
  const submit = () => {
    //je simule un formulaire
    const user = {
      name: "John",
      age: 30,
    };

    axios.post("https://jsonplaceholder.typicode.com/users", user)
      .then((res) => {
        console.log(res);
        console.log(res.data);
      });
  };

  return (
    <div>
      <h1>Test Post Api</h1>
      <button onClick={submit}>Submit</button>
    </div>
  );
};

export default TestPostApi;
```

- On utilise .post pour spécifier le type de méthode
- A besoin de l'URL d'attaque et de l'objet qu'on envoi (correspondant à ce que l'API a besoin)
- Tout comme un get/delete on utilise .then pour attendre la réponse
- Un put sera la même chose, avec .put

AXIOS

CONCEPTS AVANCÉS

- Instancier un Axios (+ base url)
- Intercepteur de requêtes
- Token de connection

AXIOS

CONCEPTS AVANCÉS

```
import Axios from 'axios'; 15.7K (gzipped: 5.4K)

const apiMovie = Axios.create({
  baseURL: 'https://api.themoviedb.org/4'
})
export default apiMovie;

const token = 'mon_jolie_token';

apiMovie.interceptors.request.use( req => {
  req.headers['Authorization'] = `Bearer ${token}`;
  return req;
})
```

AXIOS

CONCEPTS AVANCÉS

```
apiMovie.get('/discover/movie').then( res => console.log(res.data));
```

ROUTING

INTRODUCTION

- Principe du SPA : charger une seule page (index.html) et y présenter, au besoin, tout composant de l'application
- Routage : permettre à l'application de naviguer entre les différents composants, en changeant l'URL du navigateur, en modifiant l'historique du navigateur et en gardant l'état de l'interface utilisateur synchronisé.
- react-router-dom : la + populaire bibliothèque permettant d'implémenter sur React le routage sur un navigateur
- Principaux composants de react-router-dom :
 - BrowserRouter : parent des autres et effectue le routage
 - Route : définit une route entre un chemin (path) et un composant (component)
 - Routes: parent de Route et affiche la 1ère route qui correspond à l'URL
 - Link : crée une ancre de navigation

```
npm install react-router-dom
```

ROUTING BASES

```
import { BrowserRouter, Route, Routes } from "react-router-dom";
import HomePage from "./pages/HomePage";
import AboutPage from "./pages/AboutPage";
import ContactPage from "./pages/ContactPage";

const App = () => {
  return (
    <div>
      <BrowserRouter>
        <Routes>
          <Route path="/" element={<HomePage />} />
          <Route path="/about" element={<AboutPage />} />
          <Route path="/contact" element={<ContactPage />} />
        </Routes>
      </BrowserRouter>
    </div>
  );
};

export default App;
```

- **BrowserRouter**: défini la création du système de routing. Le plus souvent mis dans un composant avec une hiérarchie haute (ex: App.jsx, main.jsx)
- **Routes**: défini la liste de toutes nos routes
- **Route**: défini une route:
 - *path*: l'url de la route
 - *element*: le composant "page" à afficher sur cette URL

ROUTING

REDIRECTION ET NAVBAR

```
import { Link } from "react-router-dom";

const NavBar = () => {
  return (
    <nav>
      <Link to="/">Home</Link>
      <Link to="/contact">Contact</Link>
      <Link to="/about">About</Link>
    </nav>
  );
};

export default NavBar;
```

La balise `Link` fonctionne comme la balise `<a>`, excepté qu'elle ne redirige pas vers une autre page internet, mais vers une route du système de routing de l'application, et qu'on utilise la prop `to` et non pas `href`.

ROUTING

REDIRECTION ET NAVBAR

```
<NavLink className={({ isActive }) => isActive && "link-active"} to="/">  
  Home  
</NavLink>
```

La balise **NavLink** est pareil que la balise **Link** excepté qu'elle a accès au boolean **isActive** dans le **className**. Permettant de savoir si l'URL sur laquelle on est la même que le Link.

ROUTING

REDIRECTION JS : UseNavigate

Par moment on a besoin de rediriger après une exécution,
pour cela les balises ne sont pas utilisables, mais on a accès
au **hook *useNavigate***.

ROUTING

REDIRECTION JS : UseNavigate

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

const HomePage = () => {
  const navigate = useNavigate();

  const toContact = () => {
    navigate("/contact");
  };

  const goBack = () => {
    navigate(-1);
  };

  return (
    <div>
      <h1>Home Page</h1>
      <button onClick={toContact}>Contact</button>
      <button onclick={goBack}>Go Back</button>
    </div>
  );
};

export default HomePage;
```

hook *useNavigate*, pour avoir accès aux fonctionnalités de redirection

On peut rediriger vers une URL, ou en arrière (ici une "page" en arrière)

MINI PROJET

MINI PROJET CINÉMA

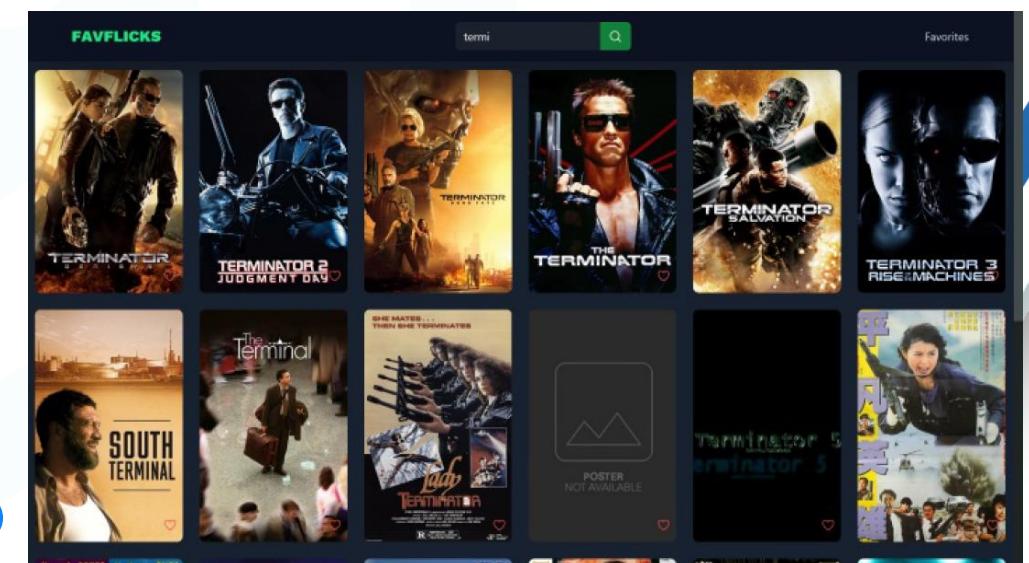
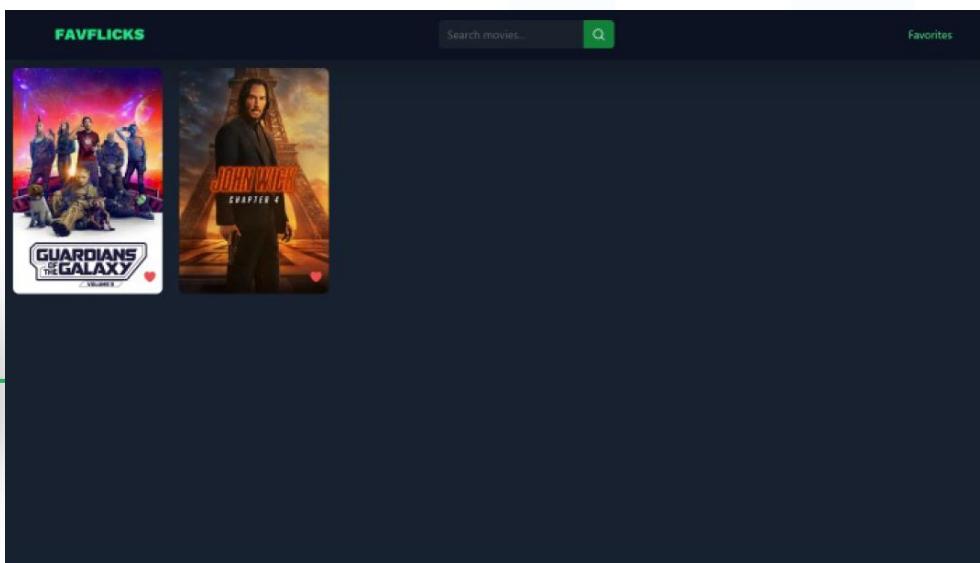
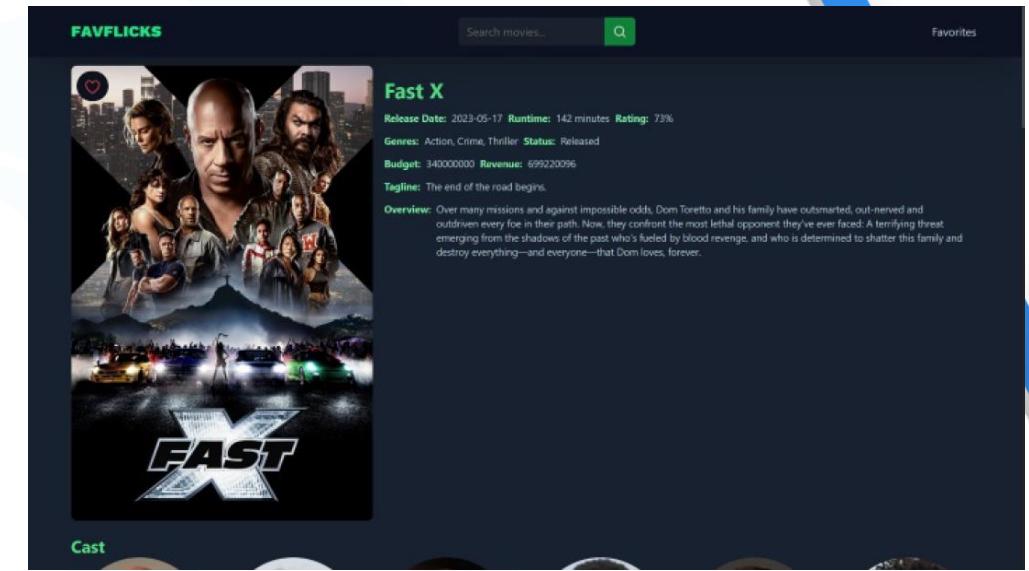
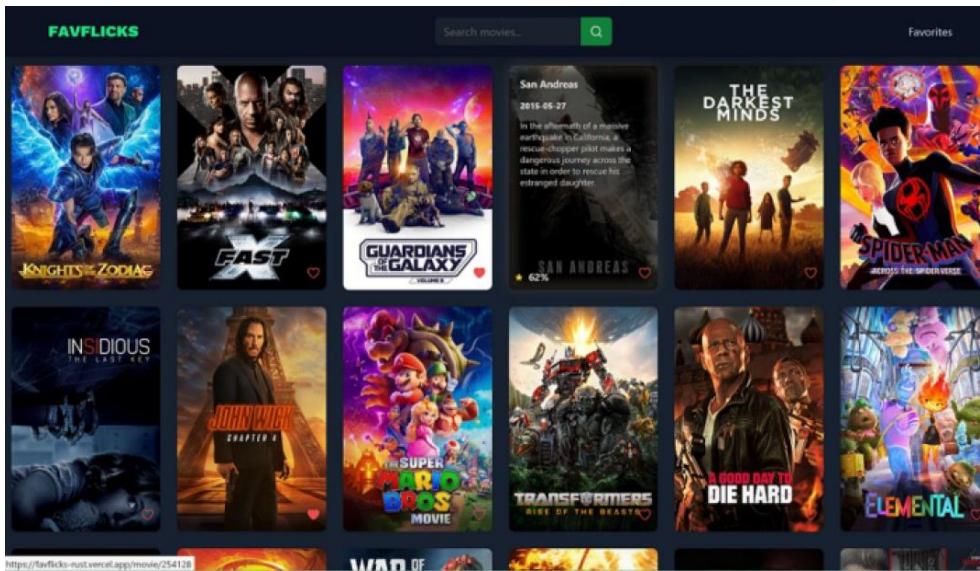
Le but du projet est de pratiquer les bases de React en créant une application de Cinéma, avec recherche de films et possibilité de mettre en Favoris

Pour cela, il faudra:

- Une page d'accueil avec les films du moment
- Une page de détails de film
- Une recherche de film
- Possibilité de mettre le film en favoris
- Page des favoris

Pour cela, il faudra utiliser l'API: tmdb. Pour laquelle il faudra vous créer un compte et créer une clé API.

MINI PROJET CINÉMA



BONNES PRATIQUES

ARCHITECTURES

INTRO

React, tout comme NodeJS, est assez souple sur la manière de faire les choses. Il en va de même pour son architecture, qui pourra être définie au choix selon le besoin et l'équipe.

On va en voir 2:

- Découpage features
- Découpage en couches

ARCHITECTURES DÉCOUPAGE

Découpage par Features

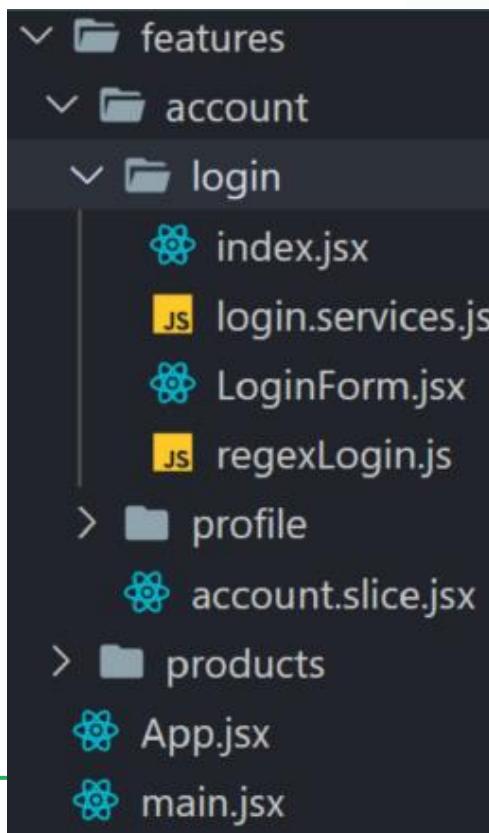
Le découpage par feature, de plus en plus populaire en front et même en back, est le fait de créer des packages qui contiendront tous les fichiers d'une feature.

C'est à dire que toutes les couches sont dans le même package.

Cela permet de faciliter la création, l'évolution ou la correction d'une feature, puisqu'il n'y a pas besoin d'aller chercher dans les différents packages de l'application, puisque tout est au même endroit.

ARCHITECTURES DÉCOUPAGE

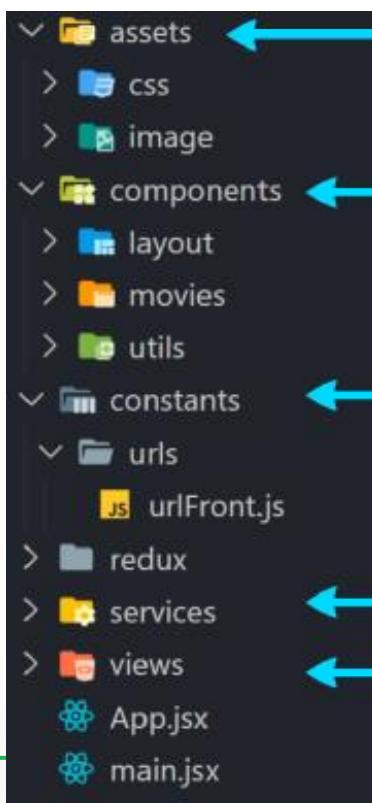
Découpage par Features



- Tout ce qui concerne le compte utilisateur se trouve dans le package "account".
- Tout ce qui touche à la feature connexion se retrouve dans le sous-package "login"

ARCHITECTURES COUCHE

Découpage par Couches



Contient tous les assets du projet. Css, images (svg, png, ico, etc), fichiers autres (ex: pdf)

Contient les composants de l'application, organisés selon le type de feature ou l'emplacement, ou si ils sont génériques et réutilisables.

Contient les constantes de l'application, pour éviter de la répétitivité et des erreurs de frappes. Peut contenir les URL (front et back), les rôles, les regex, etc...

Contient tous les appels backend et la configuration de axios

Contient les différentes "pages" ou vues de l'application.

CONCEPTS AVANCÉS

REDUX

INTRO

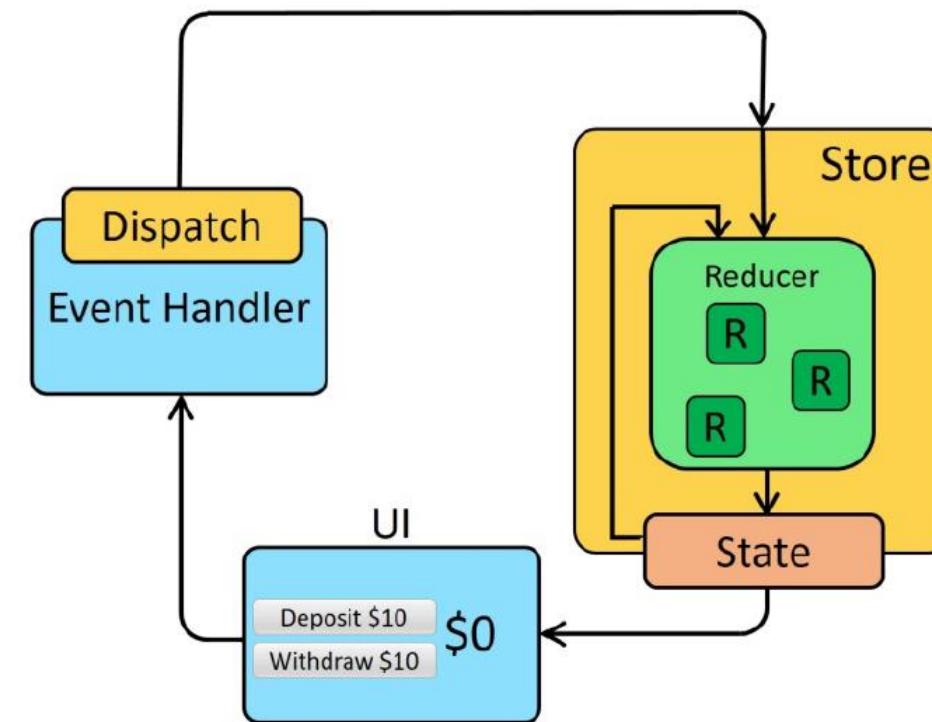
Les states dans React sont très utiles mais sont également très restreints, puisqu'ils ne sont accessibles et modifiables que là où ils ont été créés. Et si on veut y avoir accès autre part, il faut les faire passer de props en props entre parent et enfant.

Cela pouvant très vite devenir bordélique et compliqué à maintenir, React permet aujourd'hui d'avoir accès à des States globaux à l'application ou une sous partie d'elle. Cela à travers différentes librairies.

REDUX INSTALLATION

Pour installer redux et redux-toolkit:

```
npm install react-redux @reduxjs/toolkit
```



REDUX

LE STORE : LE STATE GLOBAL

Le store est le state global de notre application, qui sera accessible partout.

Celui est composé de plusieurs couches appelés des "**slices**" ou des "**reducers**"

Bien que simple à mettre en place, il peut être très configuré si besoin.

Exemple: **redux-persist**, qui permet d'automatiquement persister le store dans le localStorage.

REDUX

LE STORE : LE STATE GLOBAL

```
import { configureStore } from '@reduxjs/toolkit';

export const store = configureStore({
  reducer: {
    ...
  }
})
```

```
import { Provider } from 'react-redux'
import { store } from './store/store'

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

REDUX

SLICES : COUCHES DU STORE

En simplifié, un "slice" est un "petit" state faisant partie du store.

C'est à dire que le store est composé de plein de states différents, ayant un "thème" bien précis.

Par exemple le compte de l'utilisateur connecté aurait son propre "slice", qui serait différent d'un slice de gestion de produit.

Le slice est composé de plusieurs choses:

- un state initial (**initialState**): pour savoir les données de bases de celui-ci
- des fonctions (**reducer**): permettant de modifier l'état de ce state
- un nom unique (**name**)

REDUX

SLICES : COUCHES DU STORE

```
import { createSlice } from '@reduxjs/toolkit'

const counterSlice = createSlice({
  name: 'counter',
  initialState: {
    value: 0
  },
  reducers: {

  }
})

export default counterSlice.reducer;
```

REDUX REDUCERS

Les reducers dans un slice sont les fonctions permettant de modifier le state du slice. Ça peut aller de setter une valeur, la supprimer, modifier un tableau, etc...

Un reducers est défini par:

- un nom de fonction
- une fonction fléchée
- des propriétés de fonctions:
 - **state**: le state actuel du slice
 - **action**: contenant une variable ***payload*** qui est la valeur récupérée lorsqu'on l'utilise (voir prochain chapitre)

REDUX REDUCERS

```
reducers: {  
    increment: (state) => {  
        state.value += 1  
    },  
    decrement: (state) => {  
        state.value -= 1  
    },  
    reset: (state) => {  
        state.value = 0  
    },  
    setCounter: (state, action) => {  
        state.value = action.payload  
    }  
}
```

```
export const { increment, decrement, reset, setCounter } = counterSlice.actions
```

REDUX

useSelector : RÉCUPÉRER UNE STATE DU STORE

Maintenant qu'on a notre store, il ne reste plus qu'à le récupérer.
Pour cela redux a le hook: **useSelector**

```
const counter = useSelector((state) => state.counter.value);
```

state ici fait référence au store complet, et pas à un slice

counter est le nom du slice dont on veut le state

value ici est le nom de la variable que l'on veut récupérer dans le slice **counter** (voir chapitre sur "slices")

REDUX

useDispatch :

UTILISER LES REDUCERS

```
const counter = useSelector((state) => state.counter.value);

const dispatch = useDispatch();

return (
  <div>
    <h1>Counter: {counter}</h1>
    <button type="button" onClick={() => dispatch(increment())}>
      Increment
    </button>
    <button type="button" onClick={() => dispatch(decrement())}>
      Decrement
    </button>
    <button type="button" onClick={() => dispatch(reset())}>
      Reset
    </button>
    <button type="button" onClick={() => dispatch(setCounter(5))}>
      Set counter 5
    </button>
  </div>
);
```

Dans un slice, on a nos **reducers** permettant de modifier le state de celui-ci. Pour les utiliser il faut faire appel au hook: **useDispatch**

On a plus qu'à appeler nos reducers directement en paramètres de la fonction **dispatch**

COMMUNIQUER AVEC UN SERVEUR

- Une App React est, dans la plupart des cas, cliente d'un serveur HTTP distant
- Principales méthodes HTTP : get (récupérer), post (ajouter), put (modifier entièrement), patch (modifier partiellement) et delete (supprimer)
- La communication avec un serveur se fait le plus souvent avec l'objet XMLHttpRequest
- Quelques bibliothèques facilitant l'implémentation des requêtes AJAX : Axios, jQuery AJAX, Windows.fetch
- Il est recommandé de déclencher une requête get dans ComponentDidMount() : méthode appelée immédiatement après que le composant est monté (inséré dans l'arbre)

COMMUNIQUER AVEC UN SERVEUR EXEMPLE

- Installer le client HTTP « axios » dans le projet :

```
npm install axios
```

- Installer globalement le serveur factice « json-server » :

```
npm install --g json-server
```

- Dans un dossier quelconque, créer learners.json (contenu dans diapo suivante)

- À partir du dossier contenant learners.json, démarrer json-server sur un autre port que 3000 (utilisé aussi par React) avec par exemple la commande suivante :

```
json-server --watch learners.json --port 3333)
```

COMMUNIQUER AVEC UNE SERVEUR EXEMPLE

Contenu de learners.json :

```
{  
    "learners": [  
        {  
            "id": 1,  
            "nom": "Aminata Sy"  
        },  
        {  
            "id": 2,  
            "nom": "Pascaline Diatta"  
        },  
        {  
            "nom": "Abdou Diop",  
            "id": 3  
        }  
    ]  
}
```

COMMUNIQUER AVEC UNE SERVEUR EXEMPLE – RÉCUPÉRER TOUS LES ÉTUDIANTS

Dans Classroom, importer axios à partir de axios, puis

- Dans le constructeur, le state devient

```
this.state = {isLoading:true, students:[], error:null }
```

- Ajouter la méthode

```
componentDidMount(){
  axios.get('http://localhost:3333/learners')
    .then(res => {
      const students = res.data;
      this.setState({ students, isLoading:false });
    })
    .catch(error => this.setState({ error, isLoading: false }))
}
```

- Dans le render()
 - ajouter juste avant le return

```
const isLoading = this.state.isLoading;
```
 - juste avant learners.map, ne mettre que la ligne

```
(isLoading) ? <p>Loading...</p> :
```

COMMUNIQUER AVEC UNE SERVEUR EXEMPLE – SUPPRIMER UN ÉTUDIANT

Dans Classroom, modifier handleDelete

```
handleDelete(id){  
    axios.delete('http://localhost:3333/learners/'+id)  
    .then(res => this.setState( prevState => (  
        {students : prevState.students.filter(  
            student => student.id !== id  
        )} ) )  
    )  
    .catch(error => this.setState({error : error, isLoading:false }) )  
}
```

COMMUNIQUER AVEC UNE SERVEUR EXEMPLE – AJOUTER UN ÉTUDIANT

Dans Classroom, modifier handleAdd

```
handleAdd = nom => {
    axios.post('http://localhost:3333/learners', {nom})
    .then(res => {
        this.setState(
            { students : [...this.state.students, res.data] }
        );
    })
}
```

FORMULAIRES COMPOSANT CONTRÔLÉ

Pour contrôler l'état (initial et modifié) des éléments input (texte), textarea et select,
React leur assigne directement les attributs value et onChange

- L'attribut value sera étroitement lié à une donnée du state
- L'attribut onChange permettra de modifier la donnée du state
- Exemple :

...

```
this.state = {opinion: 'Votre opinion...' };
```

...

```
handleChange(event) {  
this.setState( {opinion:event.target.value} );  
}
```

...

```
<textarea value={this.state.opinion}  
onChange={this.handleChange}/>
```

FORMULAIRES

GESTION DE PLUSIEURS CHAMPS CONTRÔLÉS

- Pour gérer plusieurs champs contrôlés ajouter un attribut name à chaque champ
- Dans un seul handleChange,
 - accéder à chaque champ en fonction des valeurs de event.target.name et de event.target.value.
 - mettre à jour le state avec setState et [event.target.name] : event.target.value

FORMULAIRES

GESTION DE PLUSIEURS CHAMPS CONTRÔLÉS

EXEMPLE

```
...  
this.state = {opinion:'Votre opinion...', age:null };  
...  
handleChange(event) {  
    const name = event.target.name;  
    const value = event.target.value;  
    this.setState( { [name]:value } );  
}  
...  
<textarea name="opinion" value={this.state.opinion}  
    onChange={this.handleChange} />  
<input name="age" value = {this.state.age}  
    onChange = {this.handleChange} />
```

FORMULAIRES

GESTION DE PLUSIEURS CHAMPS CONTRÔLÉS

EXEMPLE : COMPOSANT AddStudent

Après avoir défini addStudent, l'instancier dans Classroom en ajoutant

- 1) import AddStudent from
"./AddStudent"
- 2) le code <addStudent /> juste avant <h1>

```
import React, { Component } from 'react'
export class AddStudent extends Component {
  constructor(){
    super();
    this.state = { nom:'', placeholder:"Nom de l'étudiant" }
  }
  handleChange = event => this.setState({nom:event.target.value});
  render() {
    return (
      <div>
        <form>
          <input
            name="nom"
            placeholder={this.state.placeholder}
            value = {this.state.nom}
            onChange = {this.handleChange}
            style={{flex:"10",marginTop:"5"}}
          />
          <input type="submit" value="Ajoutez" />
        </form>
      </div>
    )
  }
  export default AddStudent
```

FORMULAIRES

VALIDATION - EXEMPLE

- Dans le constructeur

```
this.state = { opinion: 'Votre opinion',  
age: null, error_msg: "" };
```

- Dans un gestionnaire

...

```
let err = "";  
if (name === "age") {  
    if (value != " " && !Number(val)) {  
        err = <strong>L'âge doit être un nombre</strong>;  
    }  
}  
this.setState( {error_msg: err} );
```

- Dans le template, juste après le champ « age »

```
{ this.state.error_msg }
```

- Exemple :

N'activer le bouton de soumission que si le nom de l'étudiant est saisi

FORMULAIRES

GESTION DE LA SOUMISSION

- Événement submit sur la balise <form>
- Pour le gestionnaire
 - Passer en paramètre un objet de type Event
 - Empêcher la soumission automatique avec .preventDefault()
 - Récupérer via le state toute valeur soumise
 - Traiter les valeurs soumises
 - Réinitialiser éventuellement tout champ

FORMULAIRES

EXEMPLE SUITE ET FIN

Dans AddStudent, ajoutez l'attribut onSubmit à la balise form et la méthode handleAdd

```
<form onSubmit={this.handleAdd} >  
  
handleAdd = e => {  
    e.preventDefault();  
    this.props.handleAdd(this.state.nom);  
    this.setState({nom: ''});  
}
```

Dans Classroom, ajoutez l'attribut handleDelete à AddStudent et la méthode handleAdd

```
<AddStudent handleAdd = {this.handleAdd} />  
  
handleAdd = nom => {  
    const newStudent = {id: Date.now(), nom:nom};  
    this.setState({ students : [...this.state.students, newStudent] });  
}
```

STYLE

Il y a plusieurs façons de donner du style au contenu

- Style en ligne
- Feuille de style CSS
- Module CSS

STYLE

STYLE EN LIGNE

- Usage :
 - style limité à un contenu (peu recommandé)
 - style calculé dynamiquement au moment de l'affichage
(çàd appliquer un style en fonction d'une condition)
- Principe :
Ajouter la propriété style à l'élément ou au composant
- Syntaxe :
 - style = { objet }
 - Clés de l'objet = versions camelCase des propriétés CSS
 - Valeurs de l'objet = versions string des valeurs CSS

STYLE

STYLE EN LIGNE - EXEMPLES

- Dans addStudent.js

envelopper les enfants de form dans la div suivante

```
<div style={{display:"flex", marginTop:"5px"}} >
```

...

```
<input style={{flex:"10";}} />
```

- Dans Student.js

```
<div style={{textAlign:left", borderBottom:"1px dotted"}}>
```

```
<button style={{float:right}} ... >
```

- Dans TodoItem.js (Style dynamique)

```
<p style={{color: isDone?'green':'red'}}>
```

```
{title}
```

```
</p>
```

STYLE

FEUILLE DE STYLE CSS

- Usage :

Règles de style partagées

- Principe :

- Écrire les règles de style CSS classiques dans un fichier .css
- Importer le .css dans tout composant au besoin
- Utiliser la propriété className pour appliquer les classes

STYLE

FEUILLE DE STYLE CSS - EXEMPLE

- Dans App.css, ajoutez :

```
.centrer {  
width:80%;  
margin-left:auto; margin-right:auto;  
text-align:center  
}
```

- Dans Header.js :

```
<header className = "centrer">
```

- Dans Classroom.js :

```
<div className = "centrer">
```

- Dans About.js (après avoir remplacé React.fragment par div)

```
<div className = "centrer">
```

STYLE MODULE CSS

- Usage

Partager des classes sans avoir des conflits de noms

- Principe

- Écrire les règles CSS classiques dans un fichier .module.css
- Importer le .module.css dans tout composant au besoin
- Utiliser la propriété className et l'instance de l'importation pour appliquer les classes

STYLE

MODULE CSS -EXEMPLE

- Dans Button.module.css
.error { background-color: red; }

- Dans another-stylesheet.css
.error { color: red; }

- Dans Button.js
 - ...
 - import styles from './Button.module.css';
 - import './another-stylesheet.css';
 - ...
 - <button className={styles.error}>
 Error Button
</button>