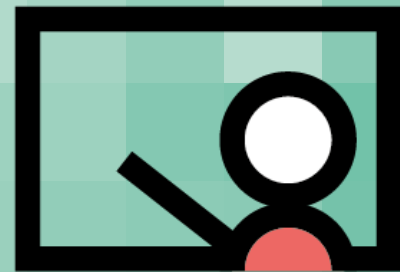


SOAT

React

MeilleurTaux



28/11/2022

Présentation personnelle

Laurent PICHET

Soat

Développeur .Net / React / Angular



Tour de table

Nom / Prénom

Poste

Niveau JavaScript / React

Attentes de la formation

Sommaire

- C'est quoi React ?
- Première application
- Composants
- Outils
- Initier notre application
- Gestion de l'état
- Formulaires
- Redux
- React Query
- Router
- Tests unitaires et bout en bout
- Typescript
- Partage de code React / React-native
- Rendu côté serveur / Génération statique
- Futur de React

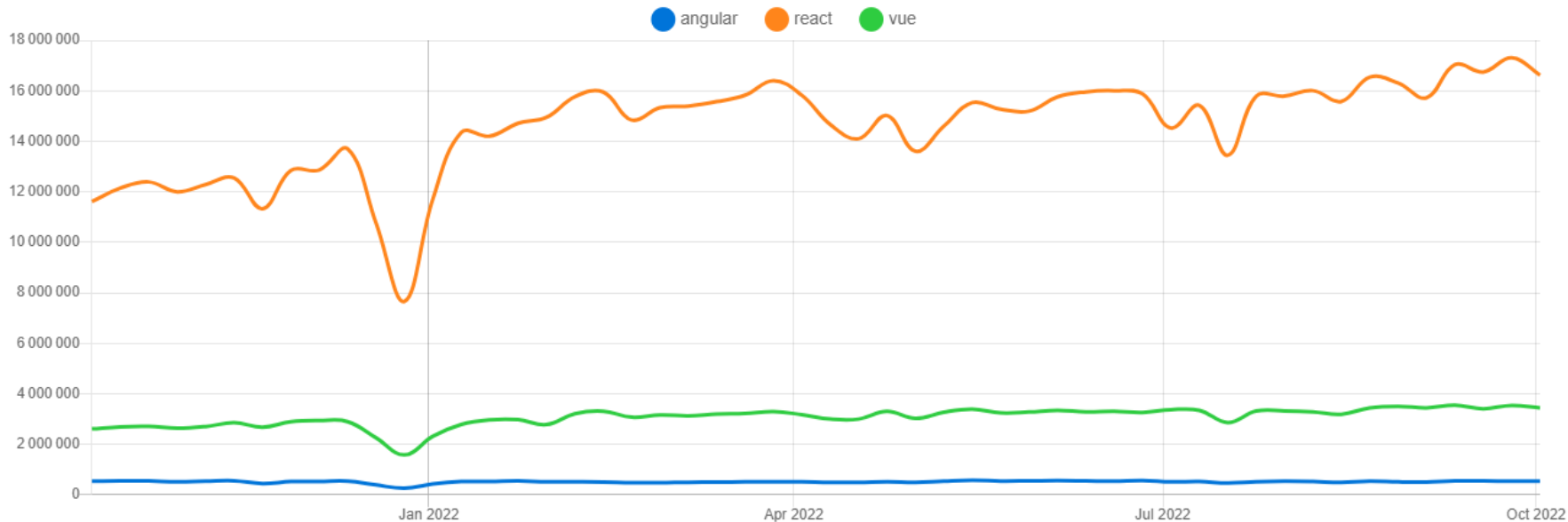
Contexte

Sites connus



Téléchargements













Downloads in past 1 Year ▾



Source npmtrends.com

Statistiques Github

Stats

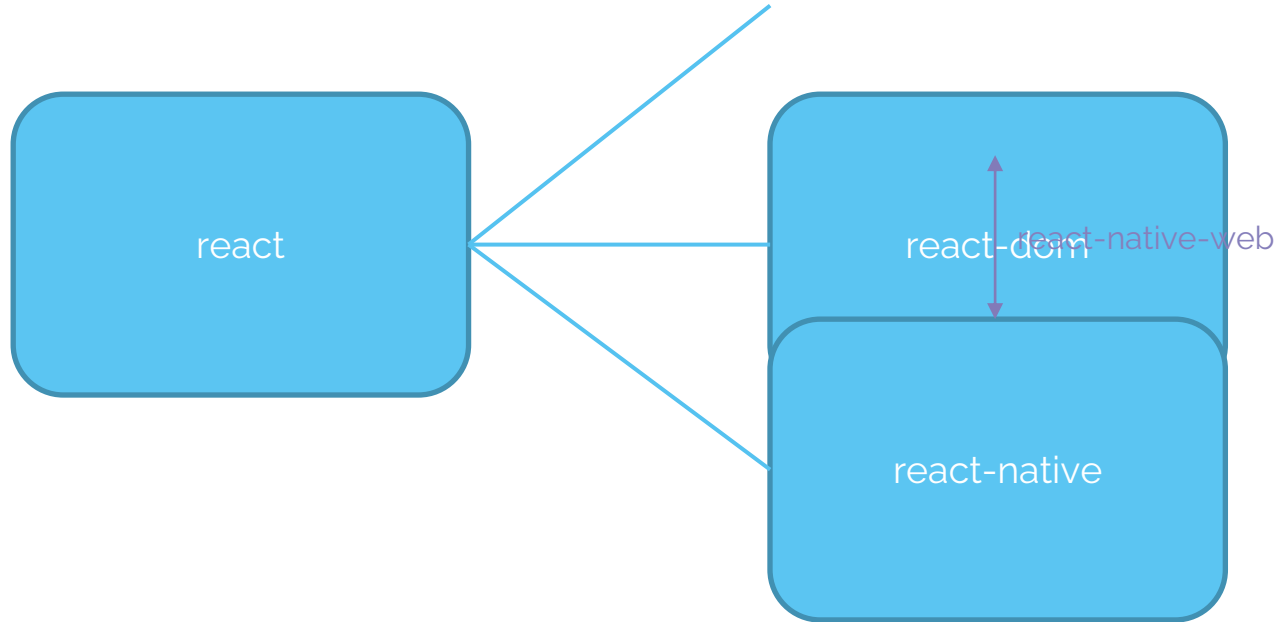
				Stars	Issues	Version	Updated ?	Created ?	Size
	angular	  		59 374	468	1.8.3	6 months ago	11 years ago	minzipped size 62.3 KB
	react	  		195 862	1119	18.2.0	4 months ago	11 years ago	minzipped size 2.5 KB
	vue	  		33 006	777	3.2.40	14 days ago	9 years ago	minzipped size 34.3 KB

Source npmtrends.com

C'est quoi React ?

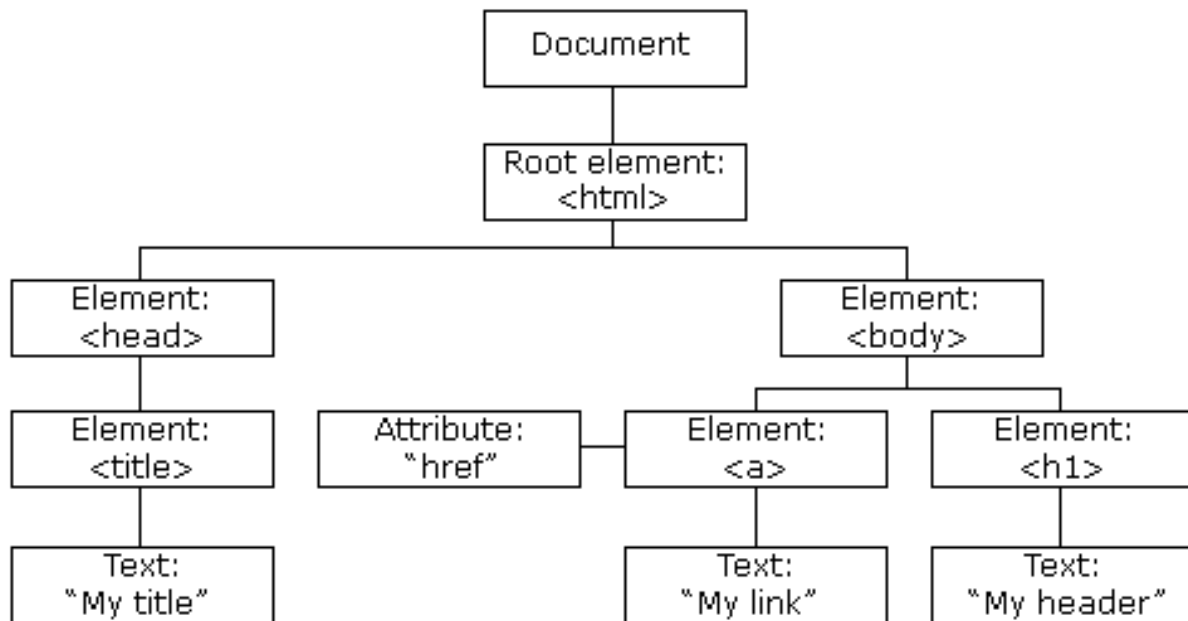
Une bibliothèque pour construire des interfaces utilisateur

Architecture



Pourquoi React a-t-il été créé ?

DOM



Fonctionnement du DOM

Construire le DOM



Recalculer la disposition et le style de chacun des éléments



Coûteux en performance sur le thread principal

Fonctionnalités de React



Découpage en
composants

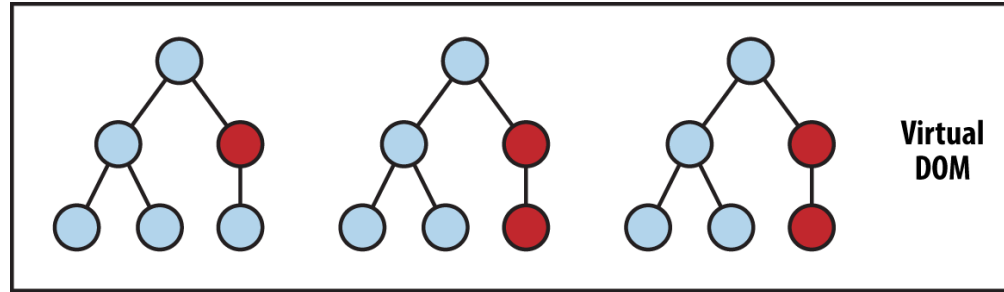


Interfaces
déclarées en
Javascript

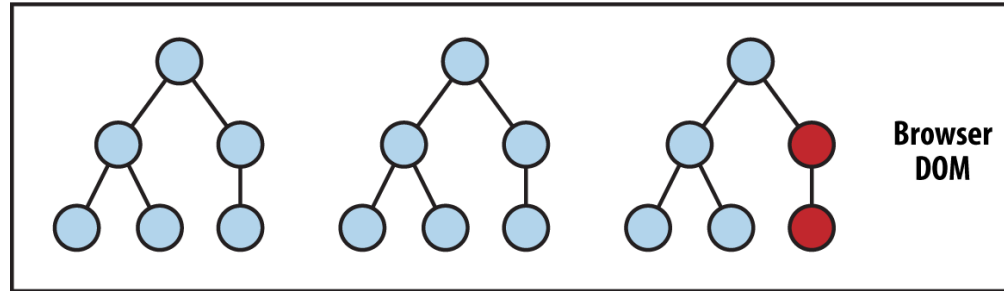


Réconciliation

Virtual DOM



State Change → Compute Diff → Re-render



Le Virtual DOM permet de réduire le nombre de manipulations du DOM

Première application React

TP : Première application

<https://github.com/lpichet/formation-react>

1-hello-react

Application React basique

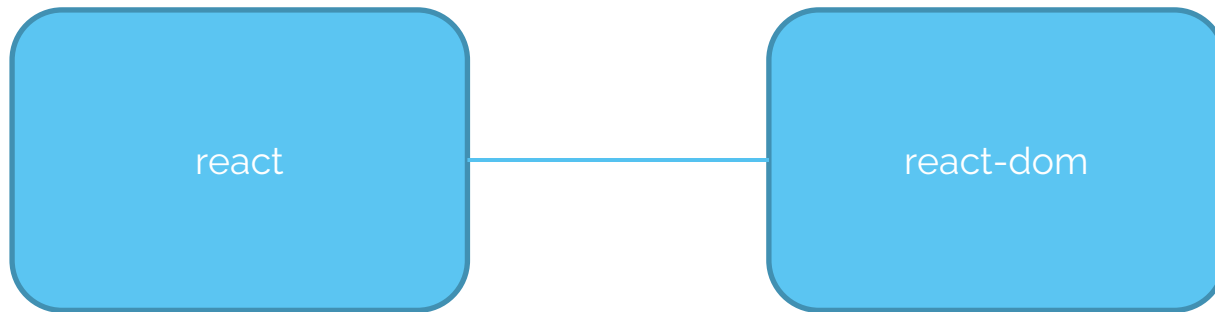
```
<!DOCTYPE html>
<html lang="en">
<head>
  <script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
  <script type="module">
    const element = React.createElement('h1', null, 'Hello React !');
    const root = ReactDOM.createRoot(document.getElementById('root'));
    root.render(element);
  </script>
</head>
<body>
  <div id="root"></div>
</body>
</html>
```

Modules react et react-dom

<https://unpkg.com/react@18.2.0/umd/react.development.js>

<https://unpkg.com/react-dom@18.2.0/umd/react-dom.development.js>

Modules



HTML généré

```
// React  
React.createElement('h1', null, 'Hello React !');
```

```
// DOM  
<h1>Hello React !</h1>
```

Comment simplifier l'écriture des interfaces ?

Idéal ?

```
// React  
<h1>Hello React !</h1>
```

```
// DOM  
<h1>Hello React !</h1>
```

JavaScript eXtension

Le JSX n'est pas interprété par le navigateur

Exemple Babel simple

```
// React JSX  
<h1>Hello React !</h1>
```

```
// Babel  
React.createElement("h1", null, "Hello React !");
```

Exemple Babel avancé

```
// React JSX
<div className="hero">
  <h1 className="header">Hello React !</h1>
  <p>
    <small>and thank you Babel !</small>
  </p>
</div>
```

```
// Babel
React.createElement("div", {
  className: "hero"
}, React.createElement("h1", {
  className: "header"
}, "Hello React !"), React.createElement("p", null,
React.createElement("small", null, "and thank you Babel
!"))));
```

Eléments JSX

```
<Header>  
  <h1>Hello React !</h1>  
</Header>  
<Users />
```

Le JSX n'est pas du HTML

Multi fichiers



Comment combiner les fichiers ?

Import et export nommés de modules

```
import { maFonction } from './MonModule';  
  
export const maFonction = () => { }
```

Import et export par défaut de modules

```
import MaFonctionParDefaut from './MonModule';  
  
const MaFonction = () => { }  
export default MaFonction;  
  
export default function MaFonction() { }
```

Alias de modules

```
import { MaFonction as MaSuPerFonction } from './MonModule';
```

```
import * as MonModule from './MonModule';
```

```
import MaSuPerFonction from './MonModule';
```

Outils

**Comment construire notre application
React ?**

VsCode

<https://code.visualstudio.com>




Visual Studio Code

Extensions



ES7+ React/Redux/React-Native snippets v4.4.3

dsznajder |  6441023 | ★★★★★ (62)

Extensions for React, React-Native and Redux in JS/TS with ES7+ syntax. Customizable. Built-in integration with prettier.

Disable





Uninstall



This extension is enabled globally.



ESLint v2.2.2

 Microsoft |  22852318 | ★★★★★ (206)

Integrates ESLint JavaScript into VS Code.

Disable



Uninstall



This extension is enabled globally.

Extensions



Import Cost v3.3.0

Wix | 2 093 224 | ★★★★★ (49)

Display import/require package size in the editor

Disable ▼ Uninstall ▼ ↺ ⚙

This extension is enabled globally.



REST Client v0.25.1

Huachao Mao | 2 800 417 | ★★★★★ (326)

REST Client for Visual Studio Code

Disable ▼ Uninstall ▼ ↺ ⚙

This extension is enabled globally.

Outils pour construire une appli React



webpack



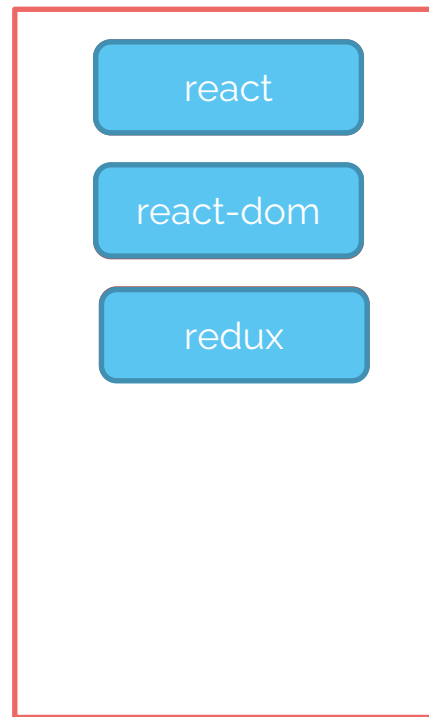
NPM

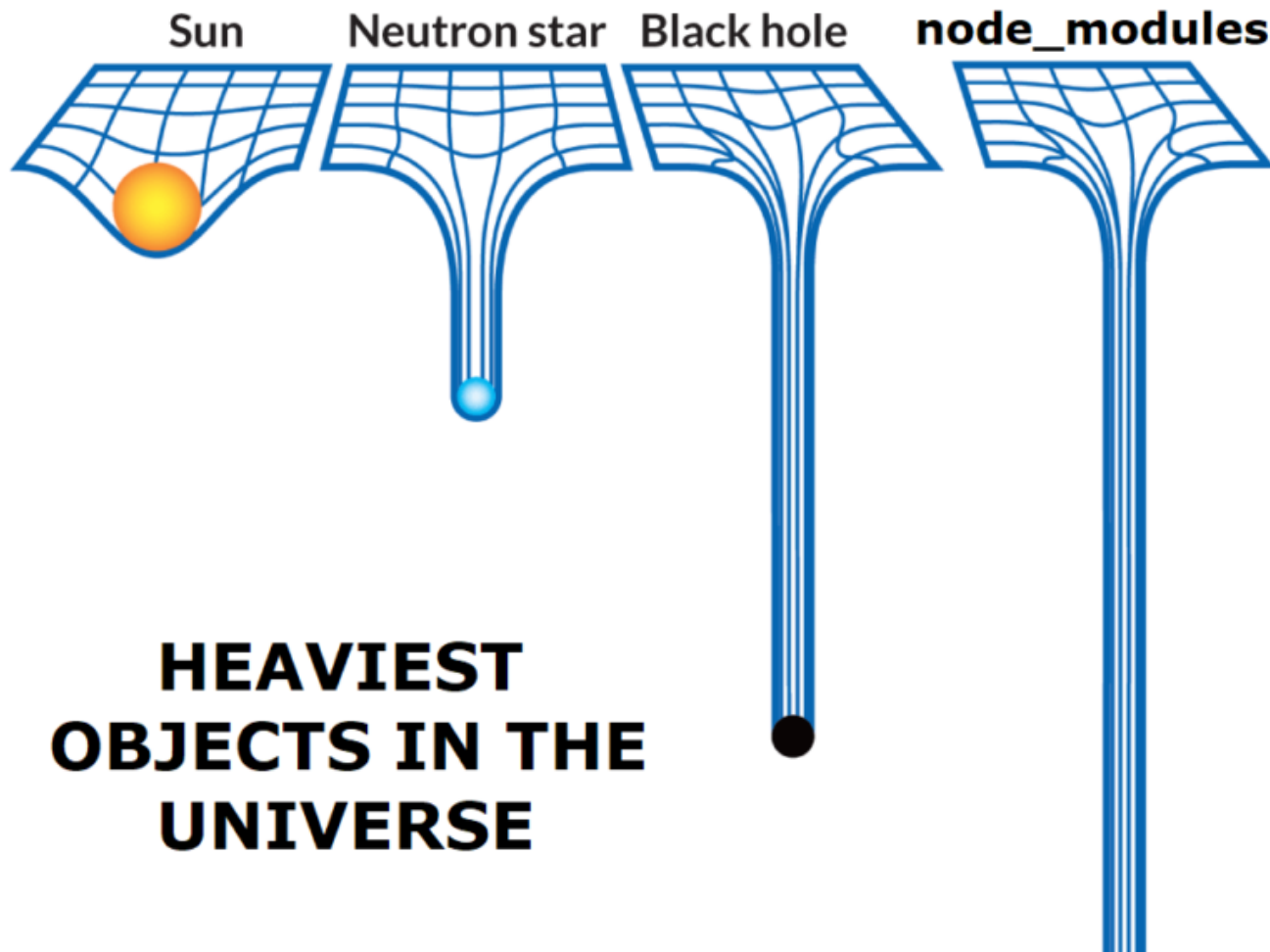
Registre NPM

Application

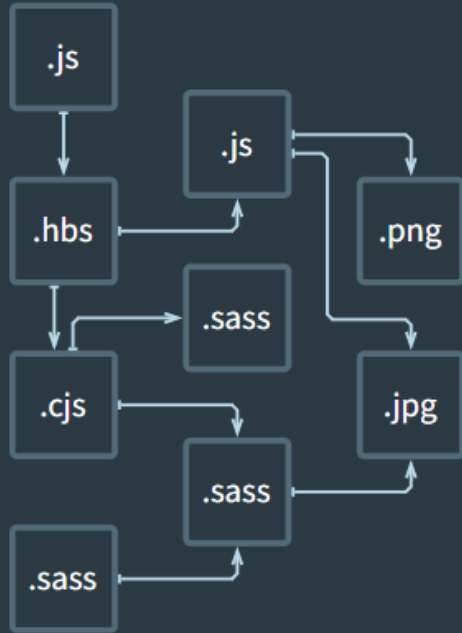


Registre NPM

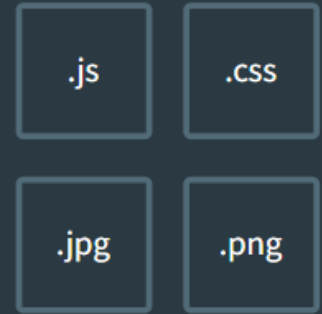
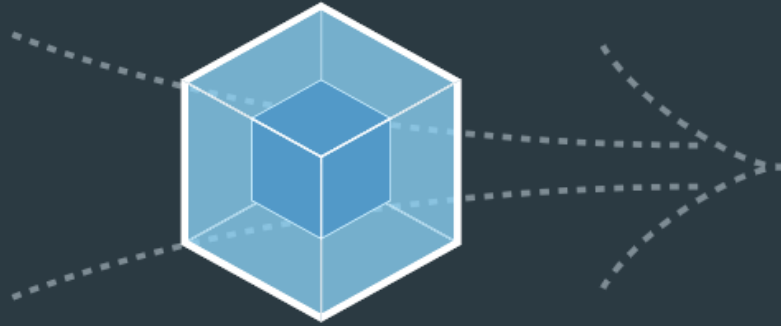




bundle your assets



MODULES WITH DEPENDENCIES



STATIC ASSETS

Bundlers

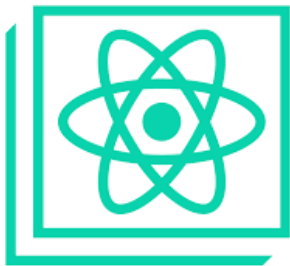


webpack



rollup.js

Frameworks React



NodeJs

<https://nodejs.org/>

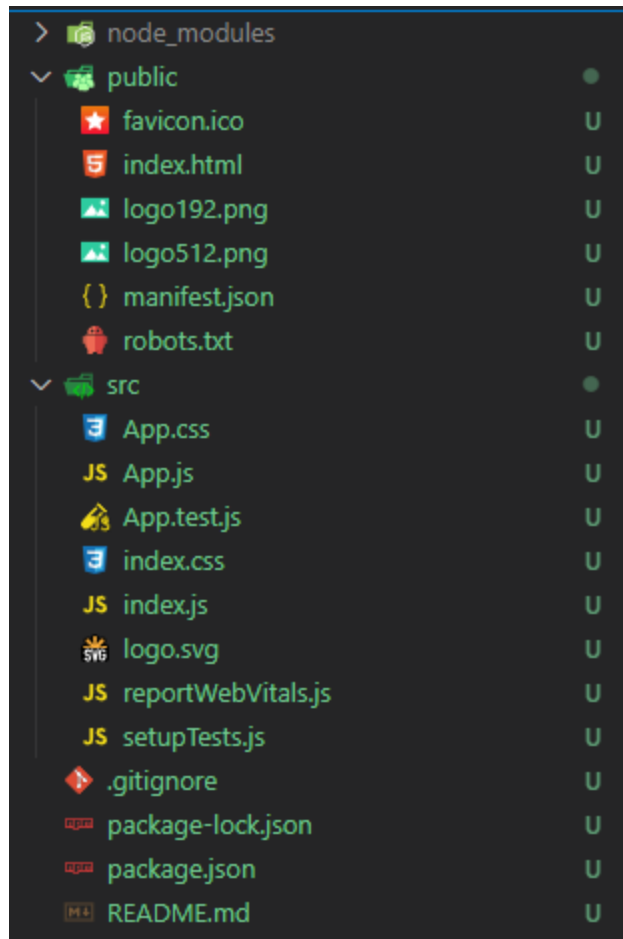
```
node -v  
v16.18.0
```

```
npm -v  
8.12.0
```

Create-react-app

<https://create-react-app.dev/>

```
npx create-react-app formation-react  
#ou  
npm init react-app my-app  
  
cd my-app  
npm start
```



Package.json

Définition package.json

```
"name": "formation-react",  
"version": "0.1.0",  
"private": true,  
"dependencies": {  
  "@testing-library/jest-dom": "^5.14.1",  
  "@testing-library/react": "^13.0.0",  
  "@testing-library/user-event": "^13.2.1",  
  "react": "^18.2.0",  
  "react-dom": "^18.2.0",  
  "react-scripts": "5.0.1",  
  "web-vitals": "^2.1.0"  
},
```

Scripts package.json

```
"scripts": {  
  "start": "react-scripts start",  
  "build": "react-scripts build",  
  "test": "react-scripts test",  
  "eject": "react-scripts eject"  
},
```

Configuration outils package.json

```
"eslintConfig": {  
  "extends": [  
    "react-app",  
    "react-app/jest"  
  ]  
},  
"browserslist": {  
  "production": [  
    ">0.2%",  
    "not dead",  
    "not op_mini all"  
  ],  
  "development": [  
    "last 1 chrome version",  
    "last 1 firefox version",  
    "last 1 safari version"  
  ]  
}
```

Vite

<https://vitejs.dev/>

```
npm create vite@latest my-app  
#ou  
npm create vite@latest my-app -template react  
  
cd my-app  
npm install  
npm run dev
```


TP: Initier le projet

Créer un projet avec l'un des outils présentés

Pourquoi utiliser des outils ?

Transpiler le JSX en JavaScript

Combiner les fichiers Javascript en un seul

Proposer un serveur de développement

Recharger la page automatiquement à chaque modification

Créer un livrable de production optimisé

Composants

**Un composant React c'est une fonction
qui retourne du JSX**

Composants fonctionnels

```
import React from 'react'; // Plus nécessaire depuis React 17

const Hello = () => (
  <h1>Hello React !</h1>
);

function Hello {
  return <h1>Hello React !</h1>
};
```

Fonctions en JavaScript

```
function maFonction(param1, param2 = ' React !') {  
  return param1 + param2;  
}
```

```
maFonction('Hello'); // Hello React !  
maFonction('Hello', ' Param'); // Hello Param
```

```
function autreFonction(_, param2) {  
  return param2;  
}
```

```
autreFonction('Hello ', 'React !'); // React !
```

```
const varFunction = function returnTrue() { return true; }  
varFunction(); // true
```

Hissage (hoisting)

```
function parent() {  
  enfant(); // OK  
  erreur(); // KO => exception  
  
  function enfant() {}  
  const erreur = function fctErreur() {};  
  
  erreur() // OK  
}
```

Fonctions fléchées

```
const a = () => true  
const b = () => { return true; }  
const c = x => x + 1;  
const d = (x, y) => x + y;  
d(3, 4) // 7
```


Composants classe

```
class Hello extends React.Component {  
  render() {  
    return (  
      <h1>Hello React !</h1>  
    )  
  }  
}
```

TP : Premier composant

Créer un composant Header dont le contenu sera une balise h1

Composant Header

```
const Header = () => (  
  <h1>Hello React !</h1>  
)  
  
function Header() {  
  return (<h1>Hello React !</h1>)  
}
```

Composants React et JSX

Un composant c'est une fonction JS qui retourne du JSX
JSX ressemble à du HTML mais n'en est pas
Sucre syntaxique pour produire du JavaScript
Babel qui transpile le JSX en Javascript
JavaScript eXtension

TP : Modifier le contenu du premier composant

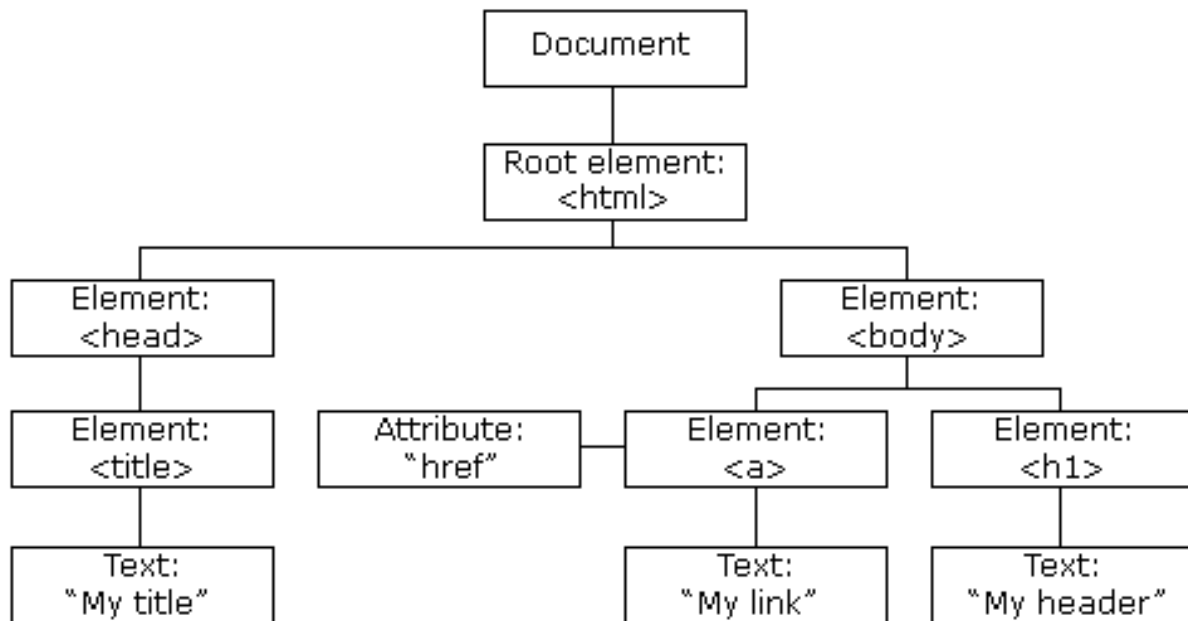
Ajouter une balise h2 sous la balise h1 du composant Header

Composant Header modifié

```
function Header() {  
  return (  
    <h1>Hello React !</h1>  
    <h2>Hello again</h2>  
  )  
}
```

Un composant React peut retourner un seul élément React

DOM



Fragment

```
import React from 'react';

const HelloFragment = () => (
  <React.Fragment>
    <h1>Hello React !</h1>
    <h1>Hello Fragment !</h1>
  </React.Fragment>
);

const HelloFragment = () => (
  <>
    <h1>Hello React !</h1>
    <h1>Hello Fragment !</h1>
  </>
);
```

Composants inclus

```
const Hello = () => {  
  return <h1 className="hello">Hello React !</h1>  
};
```

Éléments inclus

Correspondent à leur équivalent DOM

Même casse que l'équivalent DOM (camelCase)

Nos composants doivent être PascalCased

Template strings

```
const a = 'hello';  
const b = `${a} world !`;  
// b === 'Hello world !'
```

```
const c = `Hello  
    World`;  
console.log(c);  
/*  
Hello  
    World */
```

Interpolation et composition

```
const HelloInterpolationAndComposition = () => {  
  const name = 'React';  
  return (  
    <div>  
      <Hello />  
      <h1>Hello {name} !</h1>  
    </div>  
  );  
};
```

Props

```
const HelloProps = (props) => {  
  return (  
    <div>  
      <h1>Hello {props.name} !</h1>  
    </div>  
  );  
};
```

```
const HelloProps = ({name}) => {  
  return (  
    <div>  
      <h1>Hello {name} !</h1>  
    </div>  
  );  
};
```

```
<HelloProps name={"NAME"} />
```

**Les props sont en lecture seule
(immutable)**

TP: Props

Ajouter une propriété title au composant Header et interpoler le contenu en concaténant du texte et le paramètre

Composant Header avec props

```
function Header({title}) {  
  return (<h1>`Hello ${title}`</h1>)  
}
```

PropTypes

```
const MyElement = ({firstName}) => <div>{firstName}</div>
MyElement.propTypes = {
  firstName: PropTypes.string.isRequired,
  lastName: PropTypes.string
}

MyElement.defaultProps = {
  firstName: 'Laurent'
}
```

Children

```
const Header = ({children}) => {  
  return (  
    <header className="App-header">  
      {children}  
    </header>  
  );  
};
```

```
<Header>  
  <h1>  
    Formation React  
  </h1>  
</Header>
```

TP: Children

Modifier le composant Header pour permettre d'y projeter du contenu

Composant Header avec children

```
function Header({title, children}) {  
  return (  
    <>  
      <h1>{`Hello ${title}`}</h1>  
      {children}  
    </>  
  )  
}
```

Props

Passer des arguments à un composant

Utilise une syntaxe similaire aux attributs HTML

Le composant peut utiliser un objet props, ou décomposer directement les variables nécessaires

Permet de réutiliser les composants

Tableaux

```
const tableauCtor = new Array(); // A éviter
const tableauLiteral = []; // tableau vide
const tableau = [1, 2, 3, 'str'];
tableau[1]; // 2
tableau.length; // 4

tableau.push(5); // [1, 2, 3, 'str', 5]
```

Déstructuration de tableau

```
const [a, b] = [1, 2] // a: 1, b:2  
const tab1 = [1, 2];  
const tab2 = [3, 4];  
const tableau = [...tab1, ...tab2]; // [1, 2, 3, 4]
```


Array.map

```
[].map((value, index, array) => { })
```

```
const tableau = [1, 2, 3];  
tableau.map((value) => value + 1); //[2, 3, 4]  
tableau.map((value, index) => value + index); //[1, 3, 5]  
tableau.map( (_, index, array) => array[index]); //[1, 2, 3]
```

TP: Module AccountList

- ▶ Créer un composant AccountList dans son propre fichier AccountList.js
- ▶ Générer dynamiquement les entrées du tableau (nom et solde)

Modulariser un composant

```
//App.js  
import { Product } from './Product';
```

```
//Product.js  
export const Product = () => {
```

```
//App.js  
import Product from './Product';
```

```
//Product.js  
const Product = () => { }  
export default Product;
```

```
export default function Product() { }
```

Pourquoi utiliser des modules ?

Structuration du code

Réutilisabilité

Encapsulation

Nécessaire pour combiner les fichiers JavaScript

Import CSS

```
import './App.css';
```

```
...  
<div className="App">  
...
```

```
import styles from './App.module.css';
```

```
...  
<div className={styles.App}>  
  <header className={styles['App-header']}>  
...
```

```
<h1 style={{textDecoration: 'underline'}}>
```

Import CSS

L'import par fichier ajoute le contenu et est directement accessible

L'import par fichier peut avoir des collisions de nommage

L'import par module est compilé dans un objet

L'import par module ne peut pas avoir de collision de nommage

L'utilisation de l'attribut est déconseillé

TP: Module Header

Créer un fichier Header.css pour contenir le style

Déstructuration nommée

```
//App.js
accounts.map((account) => <Account key={account.id} {...account}></Account>)
//Account.js
export const Account = ({name, balance}) => { }
```

```
//App.js
accounts.map((account) => <Account key={account.id}
account={account}></Account>)
```

```
//Account.js
export const Account = ({account}) => { }
```

```
//Account.js
export const Account = ({account: {name, balance}}) => { }

export const Account = ({name: nom, balance: solde}) => { }
```


Déstructuration

```
const personne = {  
  nom: 'Paris',  
  prenom: 'Julie',  
  civilite: 'MME',  
  'date-naissance': '1er Juillet',  
  getNom: () => personne.nom  
}  
  
const autrePersonne = {...personne, nom: 'Autre', prenom: 'Aussi'}  
function affiche({nom, ...rest}) {  
  console.log('nom', nom) //prenom is not defined  
  console.log('rest', rest);  
}  
affiche(personne) // Paris Julie // {prenom: 'Julie', civilite: 'MME', 'date-  
naissance': '1er Juillet', getNom: () => personne.nom}  
affiche(autrePersonne) // Paris Julie // {prenom: 'Aussi', civilite: 'MME',  
'date-naissance': '1er Juillet', getNom : () => personne.nom}
```

Formatters de date et de devise

```
const currencyFormatter = Intl.NumberFormat('fr-FR', {style: "currency",  
currency: "EUR", maximumFractionDigits: 2});  
  
const dateFormatter = Intl.DateTimeFormat('fr-FR', { year: "numeric", month:  
"2-digit", day: "2-digit"});  
  
currencyFormatter.format(balance)  
dateFormatter.format(Date.parse("2022-10-20"))
```

Array.slice

```
[].slice()  
[].slice(start)  
[].slice(start, end)
```

```
const tableau = [1, 2, 3];  
tableau.slice(); // [1, 2, 3]  
tableau.slice(1); // [2, 3]  
tableau.slice(0, 1); // [1]
```

```
tableau.slice(-1); // [3]  
tableau.slice(0, -1); // [1, 2]
```

```
tableau.slice(4); // []  
tableau.slice(0, 6); // [1, 2, 3]
```

TP: Formatters

Ajouter date de dernière opération

Formater en français le solde et la date de dernière opération

TP Formatters

```
import { dateFormatter, } from "../helpers/Formatters"

export const AccountRow = ({id, name, balance, lastOperationDate}) => {
  return (
    <tr key={id}>
      <td>{name}</td>
      <td>{currencyFormatter.format(balance)}</td>
      <td>{dateFormatter.format(Date.parse(lastOperationDate))}</td>
    </tr>
  )
}
```

Array.sort

```
[].sort()
[].sort((first, second) => { })

const tableau = [1, 2, 3];
tableau.sort((a, b) => {
  if (a < b)
    return -1;
  if (a > b)
    return 1;
  // a doit être égal à b
  return 0;
});
tableau.sort((a, b) => a - b);
tableau.sort((a, b) => b - a);
```

TP: Sort

Trier les comptes pour avoir celui mis à jour le plus récemment en premier

Attention aux méthodes mutables !

Gestion de l'état

Événements

```
//JavaScript  
<button onclick="alert('click')">Click me</button>
```

```
//React  
const handleClick = (e) => {}  
<button onClick={handleClick}>Click me</button>
```

```
<button onClick={() => {}}>Click me</button>
```

Communication parent / enfant

```
const ParentElement = () => {  
  const action = () => { }  
  return <ChildElement handleClick={action} />  
}  
  
const ChildElement = props => {  
  const { handleClick } = props  
  
  return <button onClick={handleClick} />  
}
```

TP: onClick

Ajouter un bouton permettant d'ajouter 10 € au solde du premier compte de la liste

Avant React 16

Les composants fonctionnels ne pouvaient avoir d'état

Les composants fonctionnels ne permettaient pas de réagir à leur cycle de vie

Il fallait passer par des composants classe

useState

```
import { useState } from 'react';

const [myState, setMyState] = useState(initialValue);

const [myState, setMyState] = useState(() => expensiveFunction());

const [myState, setMyState] = useState(() => {
  if(props.nombre % 2 === 0) return 'pair';
  return 'impair'
});
```

Example:

```
<span>{myState}</span>
```

```
setMyState('new value');
```

Fonctionnement interne de useState

```
const [myState, setMyState] = useState({}); // [0]: {}  
const [myVar, setMyVar] = useState('test'); // [1]: 'test'  
const [myVar, setMyVar] = useState([]);      // [2]: []
```

TP: useState

Passer le tableau de comptes par useState
Corriger le bouton permettant d'ajouter 10€

TP: useState multiples

Ajouter un bouton permettant d'ajouter 10 € puis 20€ au premier compte avec des appels à setState successifs

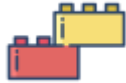
**setState n'actualise pas le composant
immédiatement**

**setState ne met pas à jour le state
immédiatement**

setState avec callback

```
setMyState(current + 1);  
setMyState((current) => current + 1);
```

Fonctionnalités



Découpage en
composants



Interfaces
déclarées en
Javascript

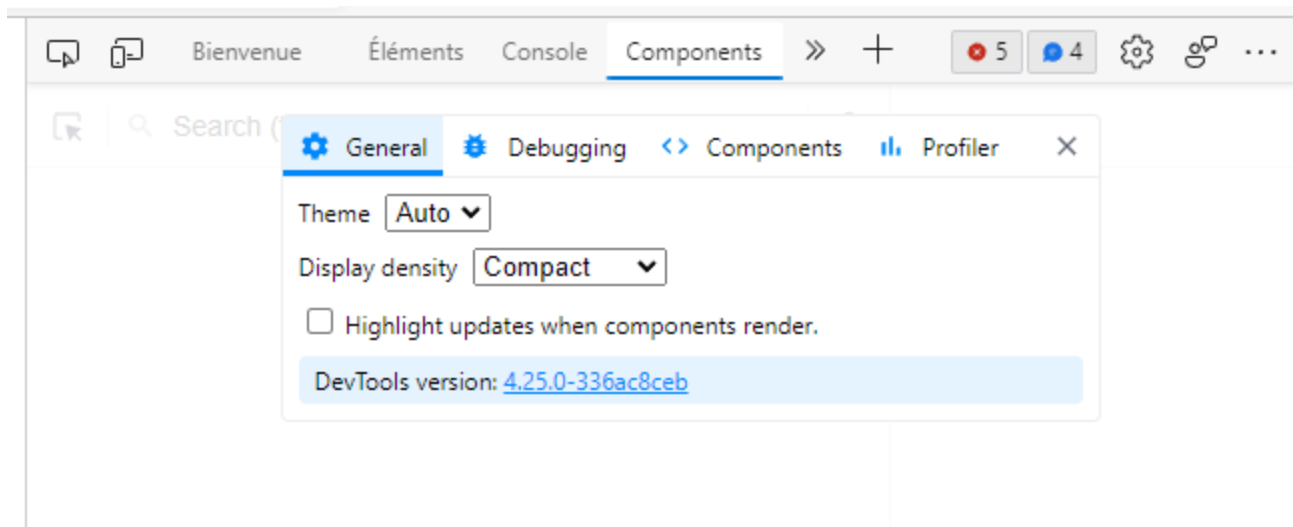


Réconciliation

TP: Ajout d'un compte

Ajouter un bouton permettant d'ajouter un nouveau compte à la liste avec des données prédéfinies

React Developer Tools



Affichage conditionnel

```
{condition && <span>contenu</span>}  
{condition ? <span>contenu</span> : <span>contenu alternatif</span>}
```


TP: Détail d'un compte

Créer un composant détail de compte

Ajouter un affichage conditionnel du détail du compte

Event bubbling

```
function handleEvent(event) {  
  event.preventDefault(); // annule le comportement par défaut  
  event.stopPropagation(); // stoppe la propagation  
}
```

useMemo

```
const result = useMemo(() => expensiveMethod(), [dependency])
```

TP: Mémoriser les lignes de comptes

Mémoriser la date de dernière opération

React.memo

```
const MyComponent = ({name}) => <h1>{name}</h1>  
const MyComponentMemo = React.memo(MyComponent)  
  
const MyComponentMemo = React.memo(MyComponent, areEqualFuntion)
```

TP: Memo

Mémoriser le composant AccountRow pour ne pas le redessiner inutilement

useCallback

```
const callback = useCallback(() => memoizedFunction(), [dependency])
```

TP: useCallback

Créer une boucle for avec un nombre de répétitions très importantes dans le chargement du composant AccountRow
Utiliser useCallback pour éviter les appels non nécessaires

Quand utiliser React.memo ?

Quand c'est plus rapide

Composants fonctionnels purs

Composants redessinés souvent avec les mêmes valeurs

Quand le JSX est complexe

**Comment gérer les interactions
externes à l'application ?**

useEffect

```
useEffect(() => {  
  //Interaction  
});
```

Fetch et asynchronisme

```
const fetchAccounts = async () => {  
  try {  
    const response = await fetch(url);  
    const data = await response.json();  
  }  
  catch(error) {  
    console.error(error);  
  }  
}  
async function fetchAccounts () { }  
  
fetch(url)  
  .then(res => res.json())  
  .then(data => console.log(data));  
  .catch(data => console.error(data));  
  
}
```

Json-server

```
npm install --save-dev json-server
```

npm-run-all

```
npm install -save-dev npm-run-all
```

Modification du package.json

```
"scripts": {  
  "start:client": "react-scripts start",  
  "start:api": "json-server --watch db.json --port 3004",  
  "start": "npm-run-all -p start:api start:client",  
  "build": "react-scripts build",  
  "test": "react-scripts test",  
  "eject": "react-scripts eject"  
},
```

Quel est le problème de notre appli ?

useEffect

```
useEffect(() => {  
  //Interaction exécutée à chaque render  
});
```

```
useEffect(() => {  
  //Interaction exécutée uniquement quand les dépendances changent  
}, [dependencyA, dependencyB]);
```

```
useEffect(() => {  
  //Interaction exécutée une unique fois  
}, []);
```

useEffect

```
useEffect(() => {  
  const logResize = (e) => console.log(e)  
  window.addEventListener('resize', logResize);  
  
  return () => window.removeEventListener('resize', logResize)  
})
```

TP: Gestion du chargement

Ajouter des variables d'état pour gérer les différents statuts de chargement

TP: Refacto avec un hook

Déplacer la gestion du chargement dans un hook personnalisé

Hooks

Fonction

Commence par « use »

Encapsulation

Accès aux fonctionnalités React avec les hooks fournis

Possibilité de créer ses hooks personnalisés

Règles pour les hooks

N'appeler les hooks qu'au sommet de la fonction

Les hooks doivent toujours être appelés, et dans le même ordre

Accessibles uniquement dans les composants fonctionnels ou un autre hook

Les hooks doivent être préfixés par « use »

Formulaires

Input contrôlé

```
const Element = () => {  
  const [value, setValue] = useState(null)  
  
  const handleChange = (event) => {  
    setValue(event.target.value)  
  }  
  
  return (  
    <input type="text"  
      onChange={(event) => handleChange(event)}  
      value={value}  
    />  
  )  
}
```


Checkbox contrôlé

```
const Element = () => {  
  const [value, setValue] = useState(false)  
  
  const handleChange = (event) => {  
    setValue(event.target.checked)  
  }  
  
  return (  
    <input type="checkbox"  
      onChange={handleChange} checked={value}/>  
  )  
}
```

Soumission du formulaire contrôlé

```
const handleSubmit = (event) => {  
  event.preventDefault();  
}  
  
return (  
  <form onSubmit={handleSubmit}>  
    <input type="submit" value="Envoi" />  
    <button type="submit">Envoi </button>  
  </form>  
)
```

useRef

```
const reference = useRef(initialValue);  
reference.current;
```

Input non contrôlé

```
const Element = () => {  
  const nameRef = useRef(null)  
  
  return (  
    <input type="text" ref={nameRef} />  
  )  
}
```

Soumission du formulaire non contrôlé

```
const handleSubmit = (event) => {  
  event.preventDefault();  
  nameRef.current.value  
}  
  
return (  
  <form onSubmit={handleSubmit}>  
    <input type="submit" value="Envoi" />  
    <button type="submit">Envoi </button>  
  </form>  
)
```

TP: Formulaire contrôlé

Créer un formulaire permettant d'ajouter un compte ou une opération

FormData

```
const handleSubmit = (event) => {
  event.preventDefault();
  const data = new FormData(event.target);
  data.set('username', 'exemple')
  data.get('username');

  const response = await fetch(url, {
    method: 'POST',
    headers: {
      'Content-Type': 'multipart/form-data'
    }
  });
}

return (
  <form onSubmit={handleSubmit}>
    <button type="submit">Envoi </button>
  </form>
)
```

Formik

Formik

```
import { Formik, Form, Field, ErrorMessage } from 'formik';
const MyForm = () => (
  <Formik
    initialValues={{ email: '', name: '' }}
    onSubmit={(values, { setSubmitting }) => {
      console.log(JSON.stringify(values));
      setSubmitting(false);
    }}
  >
    {({ isSubmitting }) => (
      <Form>
        <Field type="email" name="email" />
        <ErrorMessage name="email" component="div" />
        <Field type="name" name="name" />
        <ErrorMessage name="name" component="div" />
        <button type="submit" disabled={isSubmitting}>
          Submit
        </button>
      </Form>
    )}
  </Formik>
)
```

Validation Formik

```
validate={values => {  
  const errors = {};  
  
  if (!values.email) {  
    errors.email = 'Requis';  
  } else if (values.email.length > 50) {  
    errors.email = 'Trop long';  
  }  
  return errors;  
}}
```

Validation par champ Formik

```
{{{ errors, touched, isValidating }}} => (  
  <Form>  
    <Field name="username" validate={validateUsername} />  
    {errors.username && touched.username && <div>{errors.username}</div>}  
  
  function validateUsername(value) {  
    let error;  
    if (isUserNameCorrectFromAPI(value)) {  
      error = 'Ce nom est incorrect';  
    }  
    return error;  
  }  
)
```

Validation Formik avec Yup

```
const SignupSchema = Yup.object().shape({
  firstName: Yup.string()
    .min(2, 'Trop court')
    .max(50, 'Trop long')
    .required('Requis'),
  email: Yup.string().email('Invalide').required('Requis'),
});

validationSchema={SignupSchema}
```

React-hook-form

React-hook-form

```
const { register, handleSubmit } = useForm();
const onSubmit = data => console.log(data);

return (
  <form onSubmit={handleSubmit(onSubmit)}>
    <input {...register("firstName", { required: true, maxLength: 20 })} />
    {errors?.firstName?.type === "required" && <p>Requis</p>}
    {errors?.firstName?.type === "maxLength" && <p>Trop long</p>}
    <input {...register("lastName", { pattern: /^[A-Za-z]+$/i })} />
    {errors?.lastName?.type === "pattern" && <p>Incorrect</p>}
    <input type="number" {...register("age", { min: 18, max: 99 })} />
    {errors?.age && <p>Mauvais âge</p>}
    <input type="submit" />
  </form>
);
```

React-hook-form validation Yup

```
import { useForm } from "react-hook-form";
import { yupResolver } from '@hookform/resolvers/yup';
import * as yup from "yup";

const schema = yup.object({
  firstName: yup.string().required(),
  age: yup.number().positive().integer().required(),
}).required();

export default function App() {
  const { register, handleSubmit, formState: { errors } } = useForm({
    resolver: yupResolver(schema)
  });
  const onSubmit = data => console.log(data);

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input {...register("firstName")} />
      <input {...register("age")} />
      <input type="submit" />
    </form>
  );
}
```

useReducer

```
const initialState = {count: 0};

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return {count: state.count + 1};
    case 'decrement':
      return {count: state.count - 1};
    default:
      throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <>
      Total : {state.count}
      <button onClick={() => dispatch({type: 'decrement'})}>-</button>
      <button onClick={() => dispatch({type: 'increment'})}>+</button>
    </>
  );
}
```


TP: useReducer

Passer la navigation dans le reducer

useContext

```
const MyContext = React.createContext(initialValue);

function App() {
  return (
    <MyContext.Provider value={someValue}>
      <SomeComponent />
    </MyContext.Provider>
  );
}

function SomeComponent() {
  return (
    <div>
      <SomeChildComponent />
    </div>
  );
}

function SomeChildComponent() {
  const data = useContext(MyContext);
  return ();
}
```

Redux

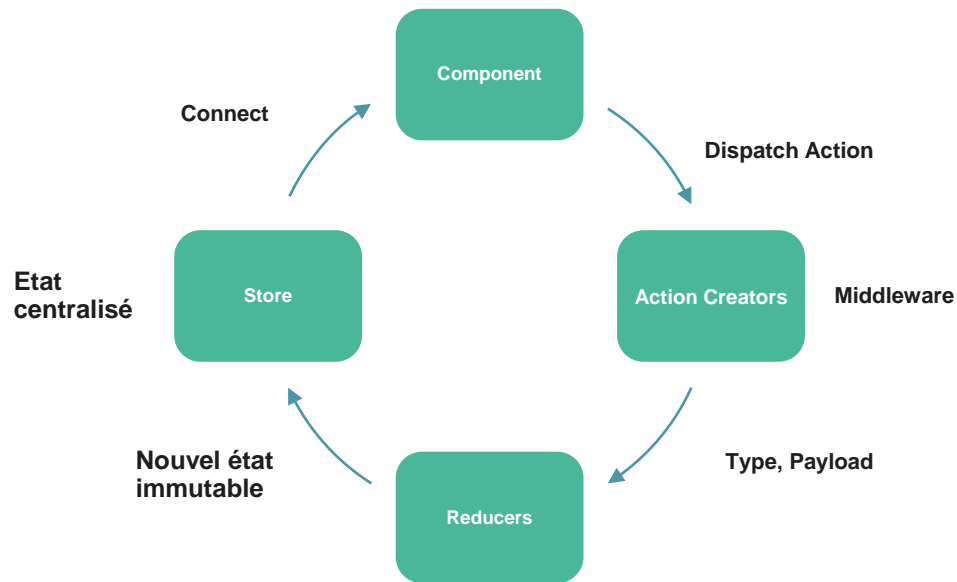
Quels sont les
problèmes
résolus par
Redux ?

Comment accéder simplement à l'état de mon application

Comment communiquer un changement d'état entre deux
arbres ?

Comment organiser mes données et avoir un comportement
prédictible ?

Schéma Redux



Store

Représenté par un arbre de données au format JSON

Stocke l'état de l'application

Est immutable

Reducer

Fonction pure

Retourne un nouvel état

L'état est le résultat d'une action appliquée à un état précédent

Action

Objet « dispatché » au store

Contient le type de l'action et un « payload » optionnel

Le payload contient les données de l'action

Middlewares

Traitement effectué entre le dispatch et le reducer
Redux fournit deux middlewares (DevTools, Thunk)

Redux createStore

```
import { applyMiddleware, createStore } from 'redux'
import { composeWithDevTools } from 'redux-devtools-extension'
import { defaultRootState } from './root'
import thunk from 'redux-thunk'
import { rootReducer } from './root.reducer'

export const store = createStore(
  rootReducer,
  defaultRootState,
  composeWithDevTools(applyMiddleware(thunk))
)
```

Redux Provider

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

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

Redux combineReducers

```
import { combineReducers } from 'redux'
import { userReducer } from './user.reducer'
import { productReducer } from './product.reducer'

export const rootReducer = combineReducers({
  userReducer,
  productReducer
})
```

Redux reducer

```
export const defaultUserState = {  
  nom: '',  
  prenom: '',  
  email: ''  
}  
  
export const USER_ACTIONS = {  
  GET_USER: 'USER_GET_USER',  
  UPDATE_EMAIL: 'USER_UPDATE_EMAIL'  
}  
  
const defaultState = {}  
function userReducer(state = defaultState, action) {  
  switch(action.type) {  
    case USER_ACTIONS.GET_USER:  
      return { ...action.payload }  
    case USER_ACTIONS.UPDATE_USER:  
      return { ...state, email: action.email }  
    default:  
      return state  
  }  
}
```

Redux Action Creator

```
import { USER_ACTIONS } from './userReducer'

export const updateEmail(email) {
  return {
    type: USER_ACTIONS.UPDATE_USER,
    email
  }
}
```

Redux-toolkit

Redux Toolkit configureStore

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

export const store = configureStore({
  reducer: {
    counterSlice: counterSlice.reducer,
    errorSlice: errorSlice.reducer
  }
});
```


Comparaison createStore

```
//Redux Toolkit
export const store = configureStore({
  reducer: {
    counterSlice: counterSlice.reducer,
    errorSlice: errorSlice.reducer
  }
});

//Redux
export const store = createStore(
  rootReducer,
  defaultRootState,
  composeWithDevTools(applyMiddleware(thunk))
)
```

Redux Toolkit Provider

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

export const App = () => {
  return (
    <Provider store={store}>
      <ContentApp />
    </Provider>
  );
};
```

Redux Toolkit createSlice

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

const initialState = { nom: '', prenom: '', email: '' };
export const userSlice = createSlice({
  name: 'userSlice',
  initialState,
  reducers: {
    getUser: (state, action) => {
      state = action.payload;
    },
    updateUser: (state, action) => {
      state.email = action.payload.email;
    }
  }
});
```

Comparaison slice / reducer

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

const initialState = { nom: '', prenom: '', email: '' };
export const userSlice = createSlice({
  name: 'userSlice',
  initialState,
  reducers: {
    getUser: (state, action) => {
      state = action.payload;
    },
    updateUser: (state, action) => {
      state.email = action.payload.email;
    }
  }
});
```

```
export const defaultUserState = {
  nom: '',
  prenom: '',
  email: ''
}

export const USER_ACTIONS = {
  GET_USER: 'USER_GET_USER',
  UPDATE_EMAIL: 'USER_UPDATE_EMAIL'
}

const defaultState = {}
function userReducer(state = defaultState, action) {
  switch(action.type) {
    case USER_ACTIONS.GET_USER:
      return { ...action.payload }
    case USER_ACTIONS.UPDATE_USER:
      return { ...state, email: action.email }
    default:
      return state
  }
}
```

Redux Toolkit actions

```
export const { getUser, updateUser } = counterSlice.actions
```

Comparison Action Creator

```
export const { updateUser } = counterSlice.actions
```

```
import { USER_ACTIONS } from './userReducer'

export const updateUser(email) {
  return {
    type: USER_ACTIONS.UPDATE_USER,
    email
  }
}
```

Redux Toolkit createAsyncThunk

```
import axios from 'axios';
import { createAsyncThunk } from '@reduxjs/toolkit';

export const fetchOption = createAsyncThunk(
  'apiSlice/fetchOptionStatus',
  async () => {
    try {
      const response = await axios.get('http://api');
      return response.data;
    } catch (error) {
      return Promise.reject(error);
    }
  }
);
```

React-query

React-query est
complet !

Gestion de l'asynchrone

Mise en cache

Regroupement des requêtes identiques

Actualisation de données obsolètes en arrière plan

Actualisation au refocus de la page

Relance automatique en cas d'erreur

Mises à jour « optimistes »

Pagination

Chargement à la demande

QueryClientProvider

```
import { QueryClient, QueryClientProvider, useQuery } from '@tanstack/react-query'

const queryClient = new QueryClient()

export default function App() {
  return (
    <QueryClientProvider client={queryClient}>
      <Example />
    </QueryClientProvider>
  )
}
```

React Query

```
function Example() {  
  const { isLoading, error, data } = useQuery([key], () =>  
    fetch(url).then(res =>  
      res.json()  
    )  
  )  
  
  if (isLoading) return 'Chargement...'  
  
  if (error) return 'Une erreur est survenue: ' + error.message  
  
  return (  
    <div>  
      <h1>{data.name}</h1>  
      <p>{data.description}</p>  
    </div>  
  )  
}
```

Query keys

```
useQuery(['todos'], ...)
```

```
useQuery(['something', 'special'], ...)
```

```
useQuery(['todo', todoId, { edit: true }], ...)
```

Ordre Query keys

```
// Celles-ci sont égales
useQuery(['todos', { status, page }], ...)
useQuery(['todos', { page, status }], ...)
useQuery(['todos', { page, status, other: undefined }], ...)

// Celles-ci ne le sont pas
useQuery(['todos', status, page], ...)
useQuery(['todos', page, status], ...)
useQuery(['todos', undefined, page, status], ...)
```

Query functions

```
useQuery(['todos', todoId], async () => {  
  const response = await fetch(`/todos/${todoId}`)  
  if (!response.ok) {  
    throw new Error('Erreur dans la requête')  
  }  
  return response.json()  
})
```

Query function context

```
useQuery(['todos', todoId], async (queryFunctionContext) => { })
```

```
queryFunctionContext = {  
  queryKey,  
  pageParam, // Infinite queries  
  signal, // Annulation de requête  
  meta, // Infos optionnelles sur la requête  
}
```

Infinite queries API

```
fetch('/api/projects?cursor=0')  
// { data: [...], nextCursor: 3}  
fetch('/api/projects?cursor=3')  
// { data: [...], nextCursor: 6}  
fetch('/api/projects?cursor=6')  
// { data: [...], nextCursor: 9}  
fetch('/api/projects?cursor=9')  
// { data: [...] }
```


Infinite queries

```
import { useInfiniteQuery } from '@tanstack/react-query'

function Projects() {
  const fetchProjects = async ({ pageParam = 0 }) => {
    const res = await fetch('/api/projects?cursor=' + pageParam);
    return res.json();
  }

  const {
    data,
    error,
    fetchNextPage,
    hasNextPage,
    isFetching,
    isFetchingNextPage,
    status,
  } = useInfiniteQuery(['projects'], fetchProjects, {
    getNextPageParam: (lastPage, pages) => lastPage.nextCursor,
  })
  // Return sur la page suivante
}
```

Infinite queries (return)

```
return status === 'loading' ? (  
  <p>Loading...</p>  
  ) : status === 'error' ? (  
    <p>Error: {error.message}</p>  
  ) : (  
    <>  
      {data.pages.map((group, i) => (  
        <React.Fragment key={i}>  
          {group.projects.map(project => (  
            <p key={project.id}>{project.name}</p>  
          ))}  
        </React.Fragment>  
      ))}  
      <div>  
        <button  
          onClick={() => fetchNextPage()}  
          disabled={!hasNextPage || isFetchingNextPage}  
        >  
          {isFetchingNextPage  
            ? 'Loading more...'  
            : hasNextPage  
              ? 'Load More'  
              : 'Nothing more to load'}  
        </button>  
      </div>  
      <div>{isFetching && !isFetchingNextPage ? 'Fetching...' : null}</div>  
    </>  
  )  
)
```

React-router

Client side routing

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

const router = createBrowserRouter([
  {
    path: "/",
    element: (
      <div>
        <h1>Hello World</h1>
        <Link to="about">About Us</Link>
      </div>
    ),
  },
  {
    path: "about",
    element: <div>About</div>,
  },
]);

createRoot(document.getElementById("root")).render(
  <RouterProvider router={router} />
);
```

createRoutesFromElements

```
createBrowserRouter(  
  createRoutesFromElements(  
    <Route path="/" element={<Root />}>  
      <Route path="contact" element={<Contact />} />  
      <Route  
        path="dashboard"  
        element={<Dashboard />} />  
      <Route element={<AuthLayout />}>  
        <Route  
          path="login"  
          element={<Login />} />  
        <Route path="logout" />  
      </Route>  
    </Route>  
  )  
);
```

Chargement de données

```
<Route
  path="/"
  loader={async ({ request }) => {
    const res = await fetch("/api/user.json", {
      signal: request.signal,
    });
    const user = await res.json();
    return user;
  }}
  element={<Root />}
></Route>

function Root() {
  const user = useLoaderData();
}
```

Redirection

```
<Route
  path="dashboard"
  loader={async () => {
    const user = await fake.getUser();
    if (!user) {
      throw redirect("/login");
    }

    // otherwise continue
    const data = await fake.getData();
    return { user, data };
  }}
/>
```

Action

```
<Form action="/project/new">
  <label>
    Project title
    <br />
    <input type="text" name="title" />
  </label>

  <label>
    Target Finish Date
    <br />
    <input type="date" name="due" />
  </label>
</Form>

<Route
  path="project/new"
  action={async ({ request }) => {
    const formData = await request.formData();
    const newProject = await createProject({
      title: formData.get("title"),
      due: formData.get("due"),
    });
    return redirect(`/projects/${newProject.id}`);
  }}
/>
```


Navigation state

```
function App() {
  const navigation = useNavigation();
  return (<div>
    {navigation.state === "loading" && <Spinner />}
    <Header /><Outlet /><Footer /></div>);}

function NewProjectForm() {
  const navigation = useNavigation();
  const busy = navigation.state === "submitting";
  return (
    <Form action="/project/new">
      <fieldset disabled={busy}>
        <label>
          Project title
          <br />
          <input type="text" name="title" />
        </label>
      </fieldset></Form>)}
  </div>);}
```

Skeleton et Suspense

```
<Route
  path="account/:accountId"
  element={<Account />}
  loader={async ({ params }) => {
    const detail = fake.getAccountDetail(params.accountId);
    const account = await fake.getAccount(params.accountId);
    return defer({ account, detail });
  }}
/>;

function Issue() {
  const { account, detail } = useLoaderData();
  return (
    <div>
      <Account account={account} />
      <Suspense fallback={<AccountDetailSkeleton />}>
        <Await resolve={detail}>
          {(resolvedDetail) => (
            <AccountDetail detail={resolvedDetail} />
          )}
        </Await>
      </Suspense>
    </div>
  )}
}
```

Tests unitaires

Assertions Jest

describe : permet de structurer les tests, par module, composant, etc..

it ou test: définit un test

beforeEach: Initialise avant chaque test

afterEach: Nettoie après chaque test

beforeAll: Nettoie avant la suite de tests

afterAll: Nettoie après la suite de tests

expect: Évaluation d'un résultat attendu

Structure du test

```
describe('Mon test', () => {  
  beforeEach(() => {})  
  afterEach(() => { })  
  
  it('réussit toujours', () => { expect(true).toBe(true)})  
})
```

Matchers

```
expect(n).toBeNull();
expect(n).toBeDefined();
expect(n).not.toBeUndefined();
expect(n).not.toBeTruthy();
expect(n).toBeFalsy();

expect(value).toBeGreaterThan(3);
expect(value).toBeGreaterThanOrEqual(3.5);
expect(value).toBeLessThan(5);
expect(value).toBeLessThanOrEqual(4.5);
//expect(value).toBe(0.3);           This won't work because of rounding error
expect(value).toBeCloseTo(0.3); // This works.

expect('team').not.toMatch(/I/);
expect(shoppingList).toContain('milk');
expect(shoppingList).toHaveLength(3);
expect(() => compileAndroidCode()).toThrow();
expect(() => compileAndroidCode()).toThrow('you are using the wrong JDK');

expect(mock).toHaveBeenCalled();
expect(mock).toHaveBeenCalledWith();
expect(mock).toHaveBeenLastCalledWith();
expect(mock).toHaveReturned();

expect(myObject).toHaveProperty('name');
```

Asynchrone

```
test('the data is peanut butter', () => {  
  return fetchData().then(data => {  
    expect(data).toBe('peanut butter');  
  });  
});
```

```
test('the data is peanut butter', async () => {  
  const data = await fetchData();  
  expect(data).toBe('peanut butter');  
});
```

```
test('the fetch fails with an error', async () => {  
  expect.assertions(1);  
  try {  
    await fetchData();  
  } catch (e) {  
    expect(e).toMatch('error');  
  }  
});
```

```
test('the data is peanut butter', async () => {  
  await expect(fetchData()).resolves.toBe('peanut butter');  
});
```

```
test('the fetch fails with an error', async () => {  
  await expect(fetchData()).rejects.toMatch('error');  
});
```

Mock

```
const mockCallback = jest.fn(x => 42 + x);  
  
const myMock = jest.fn();  
myMock.mockReturnValueOnce(10).mockReturnValueOnce('x').mockReturnValue(true);
```


Mock module

```
import axios from 'axios';

jest.mock('axios');

test('should fetch users', () => {
  axios.get.mockResolvedValue(resp);
  ...

  //Mock partiel
  jest.mock('../foo-bar-baz', () => {
    const originalModule = jest.requireActual('../foo-bar-baz');
    return {
      __esModule: true,
      ...originalModule,
      default: jest.fn(() => 'mocked baz'),
      foo: 'mocked foo',
    };
  });

  test('should do a partial mock', () => {
    const defaultExportResult = defaultExport();
    expect(defaultExportResult).toBe('mocked baz');
    expect(defaultExport).toHaveBeenCalledTimes(1);

    expect(foo).toBe('mocked foo');
    expect(bar()).toBe('bar');
  });
});
```

Mock implementation

```
const myMockFn = jest
  .fn(() => 'default')
  .mockImplementationOnce(() => 'first call')
  .mockImplementationOnce(() => 'second call');

console.log(myMockFn(), myMockFn(), myMockFn(), myMockFn());
// > 'first call', 'second call', 'default', 'default'
```

React-testing-library

Types de query

getBy

queryBy

findBy

getAllBy

queryAllBy

findAllBy

Fonctionnement des queries

Type of Query	0 Matches	1 Match	>1 Matches	Retry (Async/Await)
Single Element				
<code>getBy...</code>	Throw error	Return element	Throw error	No
<code>queryBy...</code>	Return <code>null</code>	Return element	Throw error	No
<code>findBy...</code>	Throw error	Return element	Throw error	Yes
Multiple Elements				
<code>getAllBy...</code>	Throw error	Return array	Return array	No
<code>queryAllBy...</code>	Return <code>[]</code>	Return array	Return array	No
<code>findAllBy...</code>	Throw error	Return array	Return array	Yes

Types de critères

ByRole

ByLabelText

ByPlaceholderText

ByText

ByDisplayValue

ByAltText

ByTitle

ByTestId

Événements

```
test('trigger some awesome feature when clicking the button', async () => {
  const user = userEvent.setup();
  render(<MyComponent />);

  await user.click(screen.getByRole('button', {name: /click me!/i}));
  await user.keyboard('[ShiftLeft>]');
  await user.keyboard('Laurent');
  await userEvent.selectOptions(screen.getByRole('listbox'), ['1', 'C'])
  await userEvent.deselectOptions(screen.getByRole('listbox'), '2')
  await userEvent.type(screen.getByRole('textbox'), ' World!')

  // ...assertions...
})
```

MSW

```
import {rest} from 'msw'
import {setupServer} from 'msw/node'

const server = setupServer(
  rest.get('/users', (req, res, ctx) => {
    return res(ctx.json([{id: 1, name: 'Laurent'}, {id: 2, name: 'PICHET'}]))
  }),
)

beforeAll(() => server.listen())
afterEach(() => server.resetHandlers())
afterAll(() => server.close())

test('charge et affiche les utilisateurs', async () => {
  render(<Users />)

  await waitFor(() => screen.getByRole('heading'))

  expect(screen.getByText('Laurent')).toBeVisible()
  expect(screen.getByText('PICHET')).toBeVisible()
})
```


Tests bout en bout

Cypress

```
describe('The Login Page', () => {  
  it('sets auth cookie when logging in via form submission', function () {  
    cy.visit('/login')  
    cy.get('input[name=username]').type(username)  
    cy.get('input[name=password]').type(`${password}{enter}`)  
    cy.url().should('include', '/dashboard')  
    cy.getCookie('your-session-cookie').should('exist')  
    cy.get('h1').should('contain', 'Laurent')  
  })  
})
```

Typescript

Typescript

Sur-ensemble du Javascript

Compilé (tsc)

Transpilé en Javascript

Ajoute la vérification statique des types

Intégration à Intellisense

Types

Primitifs : string, number et boolean

Array

any

Union |

Alias (type)

Interface

Enum

readonly

Tuple

Objets

Génériques

Classe

```
class C {  
  _length = 0;  
  constructor(l: number) {  
    this._length = l;  
  }  
  get length() {  
    return this._length;  
  }  
  set length(value) {  
    this._length = value;  
  }  
  print(toBePrinted: string) : string {  
    console.log(toBePrinted);  
    return toBePrinted;  
  }  
}
```

Héritage

```
class Base {  
  k = 4;  
}  
  
class Derived extends Base {  
  constructor() {  
    super();  
    console.log(this.k);  
  }  
}
```

Implémentation

```
interface A {  
    x: number;  
    y?: number;  
}  
  
class D implements A {  
    x = 0;  
    y = 0;  
}  
  
}
```


Static

```
class Static {  
  static x = 0;  
  static printX() {  
    console.log(Static.x);  
  }  
}
```

Merci !