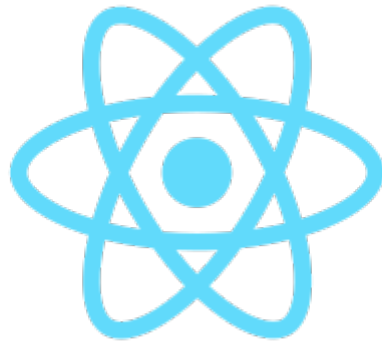


Formation React



Introduction



Objectifs de la formation

- Comprendre **React** et son écosystème
- Etre capable de créer une application React
- Etre capable d'utiliser **react-router** pour le routage d'une SPA React
- Etre capable d'utiliser **redux** pour la gestion des états d'une application React



Histoire

- **2010 : Composants Composites**
 - Facebook présente **XHP**, qui intègre du XML directement dans PHP pour créer des composants UI.
- **2011 : Evènements en cascade**

- Les ingénieurs Facebook rencontrent des problèmes au niveau de la gestion du très grand nombre d'updates du DOM. L'un d'entre eux, **Jordan Walke**, crée **FaxJS**, prototype de React.
- **2012 : Naissance de React**
 - Jordan Walke adapte son prototype au code de Facebook et le renomme **React**.
- **2013 : React devient Open-source**
 - React est finalement isolé du code de Facebook pour devenir open-source.



Histoire

- **2014 : la croissance commence**
 - De plus en plus de développeurs utilisent React et les nouveaux outils mis à disposition (React Developer Tools et React Hot Loader). Des bibliothèques de composants apparaissent telle que Material-UI.
- **2015 : Stabilisation**
 - React 0.13 est stable. Adopté par Netflix et Airbnb. Dan Abramov et Andrew Clarke créent Redux. React Native est publié en open-source (iOS et Android).
- **2016 - 2018**
 - React rencontre son public et la communauté grandit avec tout son écosystème.
- **2018 – aujourd'hui**
 - La version 16.8 de React sort en décembre 2018 et introduit les Hooks. L'écosystème rattrape ce nouveau paradigme.

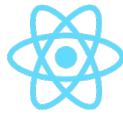


JavaScript vs TypeScript



- React fonctionne aussi bien avec **JavaScript** qu'avec **TypeScript**.

React : librairie UI



- Dans le modèle MVC, React gère uniquement le V.
- Pour le reste, il faut utiliser d'autres librairies.

Routage



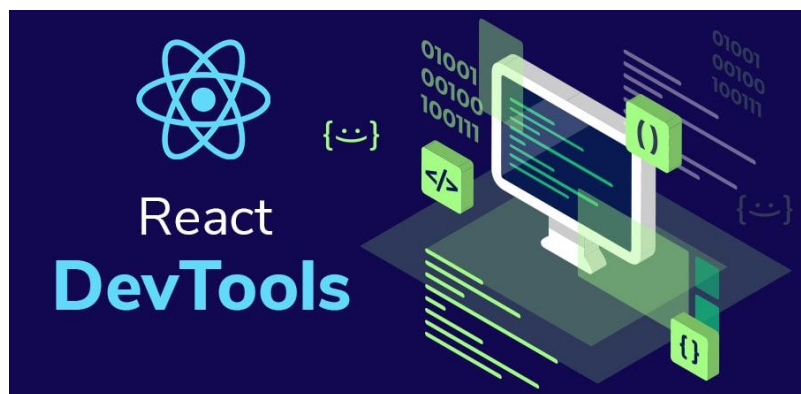
- **React-router** est une librairie de **routage** pour les applications React SPA.

Gestion des données



- **Redux** est une librairie JS de **gestion d'état**.
- La librairie **react-redux** facilite son intégration dans une application React.

React Dev Tools



- **ReactDevTools** offre une **visibilité complète** des composants React dans le navigateur (Chrome, Firefox & Chromium/Edge),

- **ReduxDevTools** offre le **TimeTravel**, visibilité pas à pas des états/actions/updates (Chrome, Firefox, Electron...).

Rappels JavaScript



<http://es6-features.org>

Javascript / ES6+

ES = ECMAScript

ECMA = European Computers Manufacturers Association
Maintenant organisation internationale de définition de standards

Implémentation de ECMAScript

- JavaScript (1995 Netscape)
meilleure implémentation connue
- ActionScript (Adobe Flash)
- Jscript (Microsoft)

ES6 (2015)

- JS n'avait pas changé depuis 2009
- Modernisation du langage

Arrow functions

- Syntaxe raccourcie de fonction en utilisant la notation **=>**
- Support d'expression simple
*Corps de fonction concis sans { } ni **return** (implicite)*
- Support corps classique
*Corps explicite avec { } et **return** éventuel*
- Même **this** que celui de son code englobant

Arrow functions - Expression bodies

ES6

```
const ages = [10, 12, 13, 14, 19, 20, 27];
const minors = ages.filter(age => age < 18);
const agesMeta = ages.map(age => ({ age, minor: age < 18 }));
const minorAgesSum = minors.reduce((acc, age) => acc + age, 0);
```

JS
ES

```
var ages = [10, 12, 13, 14, 19, 20, 27];
var minors = ages.filter(function (age) {
  return age < 18;
});
var agesMeta = ages.map(function (age) {
  return { age: age, minor: age < 18 };
});
var minorAgesSum = minors.reduce(function (acc, age) {
  return acc + age;
}, 0);
```

Arrow functions - Lexical this

ES6

```
function Person(age) {
  this.age = age;
  const log = () => { console.log(this.age); };
  log();
}
new Person(10);
```

JS
ES

```
function Person(age) {
  var _this = this;

  this.age = age;
  var log = function log() {
    console.log(_this.age);
  };
  log();
}
new Person(10);
```

Template Literals

- Sucre syntaxique pour la création de strings
Caractérisé par `` (back quotes)
- String avec des possibilité d'interpolation de variables
Interprétation de la variable ou de la fonction à l'intérieur de `\${ }`

Template Literals - Interpolation

ES6

```
const agent = {
  firstName: 'James',
  lastName: 'Bond'
};
const catchPhrase = `My name is ${agent.lastName}
...
${agent.firstName} ${agent.lastName}`;
```

JS
ES5

```
var agent = {
  firstName: 'James',
  lastName: 'Bond'
};
var catchPhrase = 'My name is ' + agent.lastName + ' ... \n' + agent.firstName +
' ' + agent.lastName;
```

Objets - Raccourcis

ES6

```
const firstName = 'James';  
const lastName = 'Bond';  
  
const agent = {  
  firstName,  
  lastName  
};
```

JS
ES

```
var firstName = 'James';  
var lastName = 'Bond';  
  
var agent = {  
  firstName: firstName,  
  lastName: lastName  
};
```

Objets - Décomposition

ES6

```
const list = [ 1, 2, 3 ];  
let [ a, , b ] = list;  
[ b, a ] = [ a, b ];
```

JS
ES

```
var list = [ 1, 2, 3 ];  
var a = list[0], b = list[2];  
var tmp = a; a = b; b = tmp;
```

Objets - Déstructuration

ES6

```

getValues = () => ({
  a: 10,
  b: 'toto',
  c: 'tata',
  d: 200.55
});

let { a, b, d } = getValues();

```



```

var tmp = getValues();
var a = tmp.a;
var b = tmp.b;
var d = tmp.d;

```

Objets - Spread Operator

- Utiliser le **Spread Operator** sur un objet **iterable** crée une **copie** de cet objet et *retourne* chacune de ces iterations :

```

const obj1 = {
  a: 10,
  b: 'toto'
};

const obj2 = {
  ...obj1
}; // nouvel objet avec les mêmes propriétés

const obj3 = {
  ...obj1,
  b: 'titi'
}; // modifie la propriété b

const obj4 = {
  ...obj1,
  c: 'tata'
}; // ajoute une propriété c

const arr2 = [
  ...arr1
]; // nouveau tableau avec les mêmes éléments

```

Classes

- Le JavaScript traditionnel utilise les fonctions et l'héritage par prototype pour construire des composants réutilisables

- A partir de la version ES6, le JavaScript propose une approche orientée objet grâce à des classes :
 - Nom de la classe
 - Constructeur
 - Attributs
 - Méthodes
 - Instance

Classes - Exemple

```
class Greeter {
  greeting;

  constructor(message) {
    this.greeting = message;
  }

  greet() {
    return `Hello, ${this.greeting}`;
  }
}

let greeter = new Greeter('World');
greeter.greet();
```

Classes - Héritage

- On peut créer une classe en héritant des comportements et caractéristiques d'une autre.
- Dans une fonction surclassée, on peut appeler la fonction de la classe parent grâce à **super**.

```

class Animal {
  constructor(name) {
    this.name = name;
  }
  say() {
    console.log(`Hello, I'm a ${this.name}!`);
  }
}

class Dog extends Animal {
  constructor() {
    super('dog');
  }
  say() {
    super.say();
    console.log('I like bones.');

```

Classes - Accesseurs

- JavaScript supporte les getters/setters en tant que moyen d'accéder aux membres d'un objet
- Permet un contrôle fin sur la manière dont chaque membre d'un objet peut être accédé

```

class Employee {
  _fullName;

  get fullName() {
    return this._fullName;
  }

  set fullName(newName) {
    console.log('fullName have been updated !');
    this._fullName = newName;
  }
}

let employee = new Employee();
employee.fullName = 'Bob Smith';

```

Modules

- Depuis ECMAScript 2015, JavaScript possède le concept de modules.
- Chaque module est exécuté dans son propre scope, pas dans le scope global
Les variables, fonctions, classes, etc... déclarées dans un module ne sont visibles en

dehors de ce module que si elles sont explicitement exportées

- Les modules contiennent des déclarations.
- Les relations entre modules sont spécifiées en tant qu'imports ou qu'exports de fichiers au sens filesystem du terme.

Modules - Exports

Chaque déclaration (variable, fonction, classe, type alias, ou interface) peut être exportée grâce au mot clé **export**.

```
export const DEFAULT_CONFIG = {
  indentUsingSpace: true,
  maxLength: 132
}

export const numberRegex = /^[0-9]+$/;

export class ZipCodeValidator implements StringValidator {
  isAcceptable(s: string) {
    return s.length === 5 && numberRegex.test(s);
  }
}

export default 'A string';
```

Modules - Imports

Import d'une déclaration exportée par défaut par un module

```
import theString from './ZipCodeValidator';
```

Import d'une ou plusieurs déclarations exportées nominativement par un module

```
import { ZipCodeValidator } from './ZipCodeValidator';

let myValidator = new ZipCodeValidator();
```

Import de toutes les déclarations d'un module

```
import * as tools from './Tools';

let valid = tools.validateCode('1234');
```

Modules - Path

```
import utilities from 'utilities';  
import tools from './tools';  
import routes from '../routes';
```

nom sans chemin

- recherche dans **node_modules**

nom d'un répertoire existant

- chargement du fichier **index.js** sur la racine du répertoire

nom sans extension

- ajout de l'extension **.js** et chargement du fichier

nom avec extension

- chargement du fichier référencé

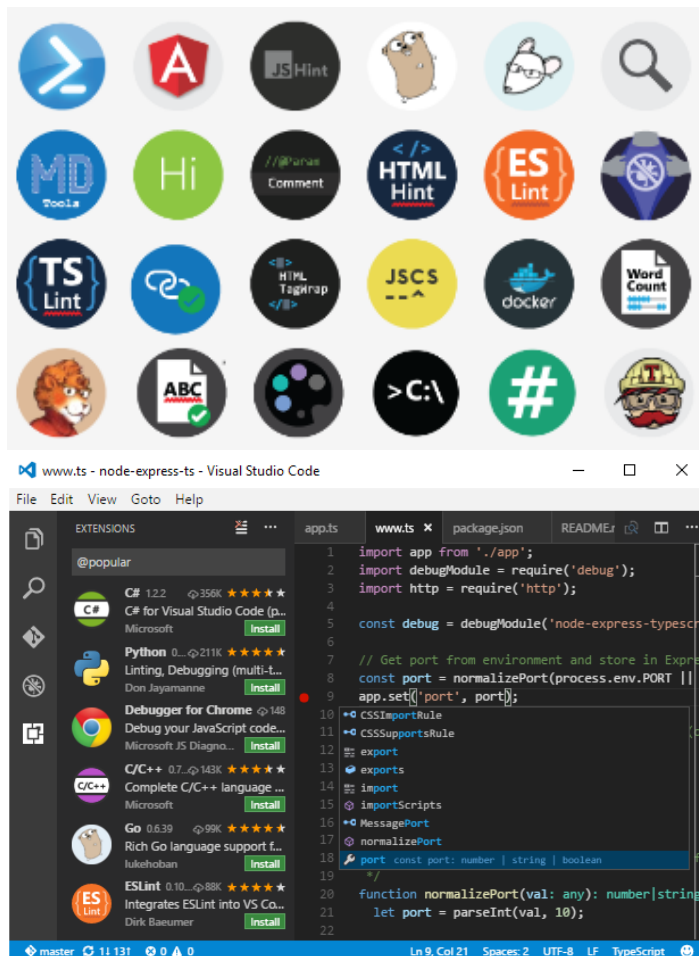
Outillage



VSCODE



- Visual Studio Code est l'IDE recommandé
<https://code.visualstudio.com/>
- Lui même développé en TypeScript !
- Gratuit
- Cross platform
- De nombreuses extensions :



NODE.JS



- **Node** est un exécutable natif (Mac / Linux / Windows) permettant d'exécuter du code JavaScript sur un PC.
- Runtime JS asynchrone et événementiel, **Node** est adapté à la création d'applications réseau scalables
- **Npm** (Node Package Manager) permet de gérer les nombreux modules JavaScript destinés à la fois au développement back et front. Il gère les dépendances entre les modules.
- **Yarn** est un package manager concurrent de npm.
- Npm et Yarn travaillent sur le fichier **package.json** centralisant toutes les dépendances du projet.

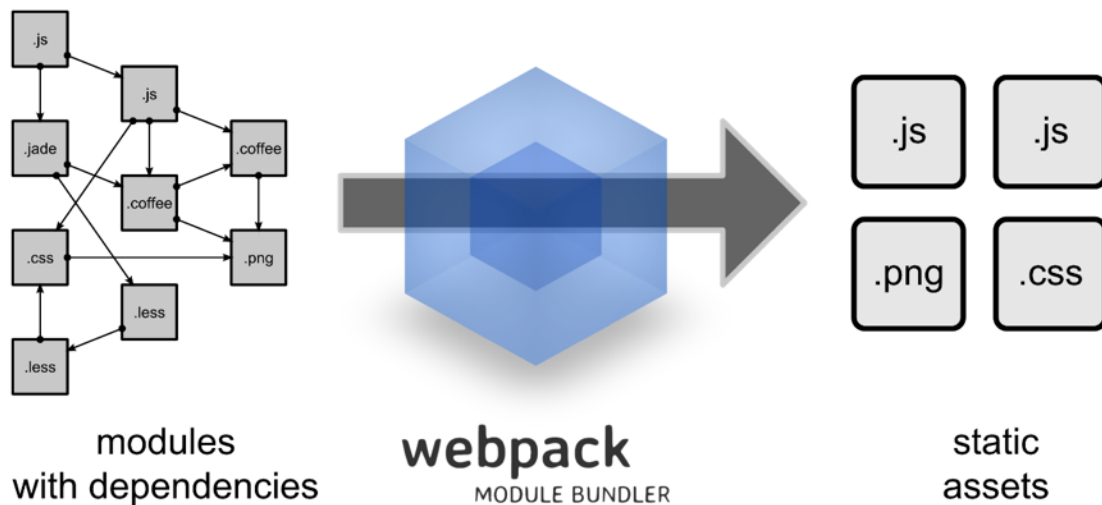


WEBPACK



Un bundler de module pour les applications JavaScript modernes

Lorsque webpack gère le build d'une application, il génère récursivement un graphe de dépendances de tous les modules qui la constituent. Il rassemble alors ces modules en un petit nombre de bundles, parfois un seul, qui sera chargé par le navigateur.



WEBPACK



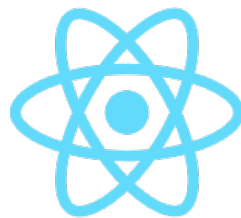
Pourquoi utiliser Webpack plutôt que Gulp ou Grunt ?

- **Gulp** et **Grunt** ne sont que des lanceurs de tâches (task runner)
- **Webpack** peut être lancé en middleware qui supporte à la fois le live reloading et le hot reloading

React



Create-react-app



<https://github.com/facebook/create-react-app>

Create-react-app

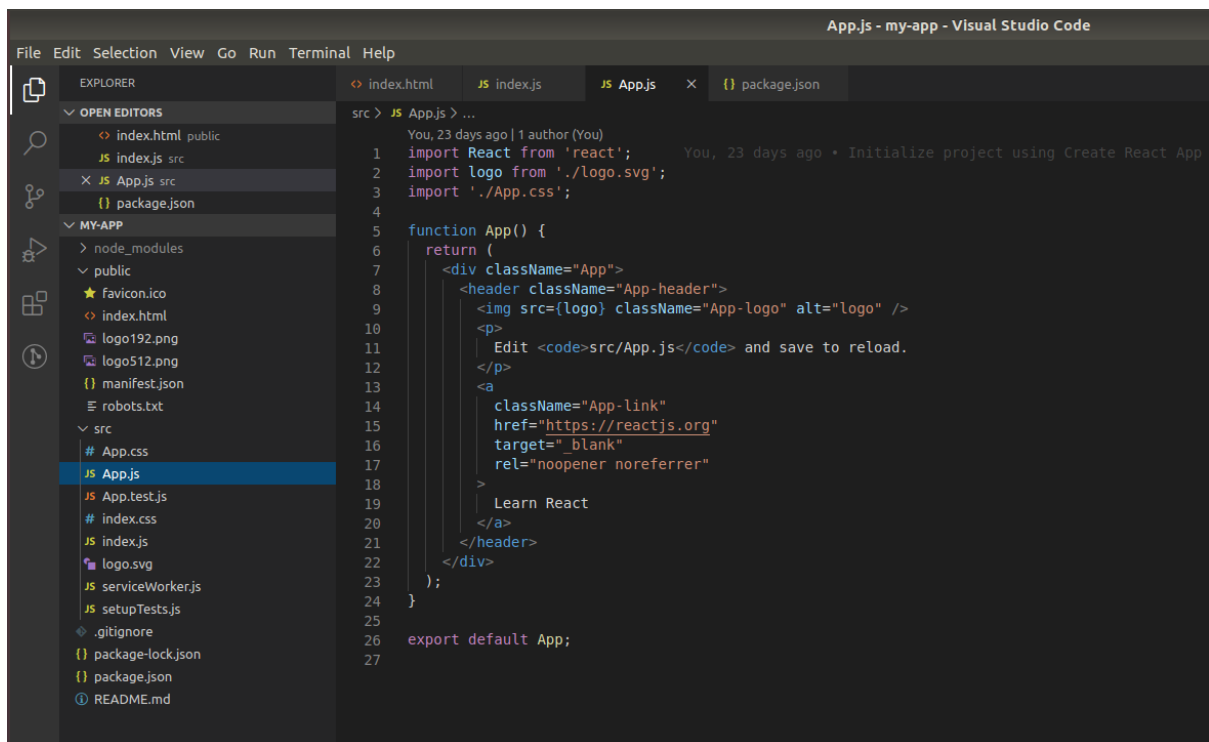
```
npx create-react-app my-app  
  
ou  
  
npx create-react-app my-app --typescript
```

- Crée un nouveau répertoire contenant une nouvelle application React
- Pas de lettre capitale pour le nom de l'application

```
cd my-app  
  
npm start
```

- Démarre le serveur de développement et ouvre un onglet du navigateur

Scaffolding



The screenshot shows the Visual Studio Code interface for a project named 'App.js - my-app'. The Explorer sidebar on the left displays the project structure, including files like index.html, App.js, and package.json. The main editor area shows the content of App.js, which is a React component. The code includes imports for React, ReactDOM, and a logo, and defines a function App() that returns a JSX element. The JSX element consists of a header with a logo and a link to the React website.

```
src > JS App.js > ...
You, 23 days ago | 1 author (You)
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import logo from './logo.svg';
4
5 function App() {
6   return (
7     <div className="App">
8       <header className="App-header">
9         <img src={logo} className="App-logo" alt="logo" />
10        <p>
11          Edit <code>src/App.js</code> and save to reload.
12        </p>
13        <a
14          className="App-link"
15          href="https://reactjs.org"
16          target="_blank"
17          rel="noopener noreferrer"
18        >
19          Learn React
20        </a>
21      </header>
22    </div>
23  );
24 }
25
26 export default App;
27
```

Pratique - TP1



- Installer Node.JS (<https://nodejs.org/>)
- Installer VS code (<https://code.visualstudio.com/Download>)
- Créer notre application React **bookstore**

Composants stateless



<https://fr.reactjs.org/>

JSX

- React introduit le langage **JSX** pour les applications JavaScript (ou TSX pour les applications TypeScript).
- **JSX** est un langage qui permet de construire des composants UI en mixant des éléments **html** et la logique **JavaScript**.


```
<h1>Hello, world!</h1>;

<h1 className="title">Hello, {name === 'Sarah'? 'Beautiful' : name}</h1>;
```

- Un **composant** React est une fonction qui retourne un élément **JSX**.

```
export const Welcome = () => <h1>Hello World!</h1>;

export default Welcome;
```

Composant avec props

- Un composant React peut recevoir des **props en paramètres** qui correspondent aux attributs HTML.

```
export const Welcome = (props) => <h1>Hello {props.name}</h1>;
```

```
import { Welcome } from './Welcome';

export const App = () => <Welcome name="Herbert" />;
```

Composants imbriqués

- En plus d'utiliser les éléments **HTML** et la logique **JS**, le **JSX** peut aussi **appeler d'autres composants** React.
- La plus part des composants auront une **responsabilité limitée**.
- **Ensemble**, les composants pourront donner une **interface complexe** à l'utilisateur.

```
import { Welcome } from './Welcome';

export const MyWelcomeList = () => (
  <div>
    <Welcome name="Sara" />
    <Welcome name="Michel" />
    <Welcome name="Edith" />
  </div>
);
```

```
export const Welcome = ({ name }) => {
  return <h1>Hello {name}</h1>;
};
```

Children prop

- Du contenu **JSX** peut être **ajouté entre la balise ouvrante et fermante** d'un composant.
- Ce contenu **JSX** est **disponible** dans le composant grâce la prop spéciale **children**.

```
import { MyWelcomeWidget } from './MyWelcomeWidget';

export const Home = () => (
  <MyWelcomeWidget>
    
  </MyWelcomeWidget>
);
```

```
import { Welcome } from './Welcome';

export const MyWelcomeWidget = ({ children }) => (
  <div>
    <Welcome name="Sara" />
    {children}
  </div>
);
```

Pratique - TP2



- Installer les React Dev Tools
- Supprimer le contenu de App.css et App.js
- Créer le **composant** Book.js dans le dossier src/components
- Ajouter la prop '**book**' au composant Book et faire en sorte qu'il affiche le titre et l'auteur du livre en paramètre
- Dans App.js, instancier le **composant Book** avec la prop '**book**'
- Observer ce composant dans l'onglet **Components** des Dev Tools du navigateur
- Tester

Composants Stateful



Classe React.Component

- Un composant stateful est un objet dérivé de la classe **React.Component**
- Il permet la gestion :
 - d'un **state** propre au composant
 - du **cycle de vie** du composant
- Le rendu HTML est réalisé par la fonction **render()**

```
import React from 'react';

class Hello extends React.Component {
  render() {
    return (
      <div>
        <h1>Bonjour, {this.props.name}!</h1>
      </div>
    );
  }
}
```

State - 1/2

- Le **state** s'initialise à la construction de la classe

```
import React from 'react';

class Hello extends React.Component {
  constructor(props) {
    super(props);
    this.state = { text: `Bonjour ${this.props.name}!` };
  }

  render() {
    return (
      <div><h1>{this.state.text}</h1></div>
    );
  }
}
```

```
class Hello extends React.Component {
  state = { text: `Bonjour ${this.props.name}!` };
  render() {
    return (
      <div><h1>{this.state.text}</h1></div>
    );
  }
}
```

State - 2/2

- Le state se modifie grâce à la fonction **setState()**

```
import React from 'react';

class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0, increment: this.props.increment || 1 };
  }

  render() {
    return (
      <div>
        <h1>Count: {this.state.count}</h1>
        <button
          onClick={() => this.setState((state) => ({
            count: state.count + state.increment
          }))}
        ></button>
      </div>
    );
  }
}
```

Cycle de vie - 1/2

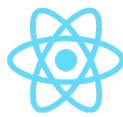
- Un composant stateful peut réagir à son cycle de vie
- Il lui suffit d'implémenter la fonction correspondante de la classe React.Component:
 - componentDidMount()**
 - Au montage du composant
 - componentWillUnmount()**
 - Au démontage du composant
 - componentDidUpdate(prevProps, prevState, snapshot)**
 - Au changement de props
 - componentDidCatch(error, info)**

- Pour intercepter les erreurs

Cycle de vie - 1/2

```
componentDidUpdate(prevProps) {  
  if (this.props.bookId !== prevProps.bookId) {  
    this.fetchBook(this.props.bookId);  
  }  
}
```

Hooks



<https://fr.reactjs.org/>

Les Hooks : késako?

- Les **Hooks** sont des fonctions qui ajoutent des **fonctionnalités** aux composants stateless de React :
 - l'**état**, le **contexte**, les **références** et le **cycle de vie**
- Un **Hook ne peut être utilisé** qu'à la **racine d'un composant** ou à la **racine d'un autre Hook**.
- Il est **strictement interdit** d'appeler un Hook :
 - dans une **condition**,
 - dans une **fonction**.

Hooks - useState() - 1/2

useState()

- prend en paramètre la valeur initiale du state,
- retourne un tableau contenant le **state** et son **setter**.

```
const [name, setName] = useState('Lucien');
```

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

export const Welcome = ({ name }) => {
  const [title, setTitle] = useState(name);

  const handleTitleChange = () => setTitle('Master');

  return (
    <div>
      <h1>Bonjour {title}</h1>
      <button onClick={handleTitleChange}>Appelez-moi "Master"</button>
    </div>
  );
};
```

Hooks - useState() - 2/2

- Le **setter** retourné par `useState()` accepte **un paramètre**, la **nouvelle valeur** du state.

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

export const Welcome = ({ name }) => {
  const [level, setLevel] = useState(1);

  const handleLevelUp = () => setLevel(level + 1);

  return (
    <div>
      <h1>Bonjour {name}</h1>
      <p>Niveau: {level}</p>
      <button onClick={handleLevelUp}>Augmenter mon niveau</button>
    </div>
  );
};
```

- le **`useState()`** peut gérer des objets complexes mais il est recommandé de créer **plusieurs state simples plutôt qu'un seul state complexe** dans le but de gagner en **lisibilité** et **maintenabilité** du code

Pratique - TP3



- Créer un **composant** `BooksList`, créer un state **'books'** et son **setter**
- Récupérer la liste des livres par un appel http **axios** sur le serveur **json-server**
- Instancier autant de composants **Book** qu'il y a d'objets dans la liste **books**
- Observer ce **composant** et son **state** dans l'onglet **'Components'** des Dev Tools du

navigateur

- Tester

Hooks - useEffect() - 1/2

- On accède au **cycle de vie** du composant avec le **useEffect()** :
 - **montage**, **mise à jour**, **démontage** du composant.
- Le **useEffect()** prend deux paramètres en entrée :
 - une **fonction** et un **tableau de dépendances** (optionnel).

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

export const LogLifecycle = ({ name }) => {
  const [title, setTitle] = useState(`Hello ${name}!`);

  useEffect(() => {
    /* Effet déclenché :
     - au montage du composant
     - à chaque fois qu'une variable du tableau de dépendances est modifiée
    */
    setTitle(`Hello ${name}!`);

    return () => {
      /* Nettoyage déclenché :
       - au démontage du composant
       - avant chaque rendu
      */
    }
  }, [name]);

  return <h1>${title}</h1>;
};
```

Hooks - useEffect() - 2/2

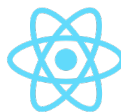
- Si le **tableau** de dépendances est **vide** :
 - l'**effet** est lancé **seulement au montage** du composant
 - le **nettoyage** est lancé **seulement au démontage** du composant
- Si le **tableau** de dépendances **n'est pas fourni** :
 - l'**effet** est lancé **à chaque rendu** quelque soit la raison
 - le **nettoyage** est lancé **avant chaque rendu**
- Si le **tableau** de dépendances **contient une ou plusieurs variables** :
 - l'**effet** est lancé **à chaque fois qu'une de des valeurs change**
- Lorsqu'il y a **plusieurs useEffects**, ils sont déclenchés **dans l'ordre** à chaque rendu

Pratique - TP4



- Dans le **composant** Book, créer un state '**stock**' et son **setter**
- Afficher le '**stock**' de chaque livre
- Créer un bouton '**more**' et un '**less**'
- Ajouter un **effet** sur '**stock**' qui fait passer en rouge le nombre de livres s'il approche de 0
- Ajouter un **nettoyage** qui loggue '**stock**'
- Persister ce stock dans la base de donnée
- Observer dans quel cas le **nettoyage** est exécuté
- Tester

Autres fonctionnalités React



<https://fr.reactjs.org/>

Fragments

- Un **composant** React doit toujours retourner **un noeud DOM unique**
- Mais encapsuler systématiquement votre JSX dans une **<div>** inutile pollue votre **html**
- Utiliser plutôt les **Fragments**

```
import React, { Fragment } from 'react';
import { Header, Router, Footer } from './components';

const App = () => (
  <Fragment>
    <Header />
    <Router />
    <Footer />
  </Fragment>
);
```



```
import React from 'react';
import { Header, Router, Footer } from './components';

const App = () => (
  <>
    <Header />
    <Router />
    <Footer />
  </>
);
```

Higher Order Component (HOC)

- Un HOC est un composant qui encapsule un autre pour lui ajouter des fonctionnalités ou modifier son comportement
- C'est une fonction qui accepte un composant en paramètre et qui en renvoie un autre
- Le HOC est un pattern

```
export const withLayout = (WrappedComponent) => (props) => (
  <div className="layout">
    <NavBar />
    <WrappedComponent {...props}/>
    <Footer />
  </div>
);
```

```
export withLayout = (WrappedComponent) => (
  class extends React.Component {
    render() {
      return (
        <div className="layout">
          <NavBar />
          <WrappedComponent {...this.props}/>
          <Footer />
        </div>
      );
    }
  }
);
```

useRef() - 1/2

- Pour créer des **variables modifiables**, on utilise **useRef()**
- Changer sa valeur ne provoque pas de render

```
import React, { useState, useEffect, useRef } from 'react';

export const Counter = () => {
  const [counter, setCounter] = useState(0);
  const timerId = useRef();

  useEffect(() => {
    timerId.current = setInterval(() => setCounter(counter + 1), 1000);
    return () => clearInterval(timerId.current);
  }, []);

  return <span>{counter}</span>;
};
```

useRef() - 2/2

- **useRef()** est utile aussi pour créer une **référence à un élément**

```
import React, { useState, useEffect, useRef } from 'react';

export const MyComponent = () => {
  const [counter, setCounter] = useState(0);
  const buttonRef = useRef();

  useEffect(() => {
    if (counter === 0) {
      buttonRef.current.focus();
    }
  }, [counter]);

  const handleClick = () => setCounter(counter + 1);

  return <button ref={buttonRef} onClick={handleClick}>Increment counter</button>;
};
```

useMemo() - 1/2

- **useMemo()** permet la **memoïzation** d'une variable
- Une variable memoïzée n'est **recalculée** que si ses **dépendances changent**
- Optimise les **performances**

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

export const MyComponent = () => {
  const [counter, setCounter] = useState(0);
  const [age, setAge] = useState(10);

  const handleClick = () => setCounter(counter + 1);

  const score = useMemo(
    () => counter * age,
    [counter, age]
  );

  return (
    <>
      <p>Score: {score}</p>
      <button onClick={handleClick}>Increment counter</button>
    </>
  );
};
```

useMemo() - 2/2

- **useMemo()** permet aussi de **memoïzer une fonction**
- Dans l'exemple ci-dessous, **incrementCounter** va adapter son comportement en fonction des valeurs de **'age'** et **'counter'**
- **Limite le nombre de renders** du composant enfant auquel on passe la fonction **incrementCounter**
- La fonction n'est recréée que si l'une des références change

```
import React, { useState, useMemo } from 'react';
import CounterButton from './CounterButton';

export const MyComponent = () => {
  const [counter, setCounter] = useState(0);
  const [age, setAge] = useState(10);

  const incrementCounter = useMemo(() => () => {
    if (age > 30) setCounter(counter + 3);
    else if (age > 18) setCounter(counter + 2);
    else setCounter(counter + 1);
  }, [age, counter]);

  return <CounterButton incrementCounter={incrementCounter} />
};
```

useCallback()

- **useCallback()** est plus adaptée pour le **memoïzing d'une fonction**

- Evite de passer un *thunk* (fonction qui retourne une fonction)
- Les deux **exemples** suivant sont **équivalents** :

```
const incrementCounter = useMemo(() => () => {
  if (age > 30) setCounter(counter + 3);
  else if (age > 18) setCounter(counter + 2);
  else setCounter(counter + 1);
}, [age, counter]);
```

```
const incrementCounter = useCallback(() => {
  if (age > 30) setCounter(counter + 3);
  else if (age > 18) setCounter(counter + 2);
  else setCounter(counter + 1);
}, [age, counter]);
```

React Router



<https://reacttraining.com/react-router>

Introduction

- **React-router** est une librairie utilisée pour le **routage** d'une SPA React.
- Elle inclut :
 - les **composants Switch, Route et Redirect** pour contrôler quel composant est **affiché**
 - et les **objets** :
 - **history** : fournit les fonctions communes,
 - **location** : contient le *current route path* et son état (optionnel)
 - **params** : fournit les *current route params*
 - **routeMatch** : fournit le *current route match*

```
> npm install react-router-dom --save
```

Browser Router

- Le composant **BrowserRouter** nécessite **très peu de configuration**
- Ses **props** sont optionnelles

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

export const App = () => (
  <Router
    basename{/* string (optionnel) */}
    forceRefresh{/* boolean (optionnel) */}
    getUserConfirmation{/* function (optionnelle) */}
    keyLength{/* number (optionnel) */}
  >
    {/* On définit toutes les routes ici */}
  </Router>
);
```

Switch & Route

- Pour **définir des Routes**:
 - on utilise les composants **Switch et Route** pour gérer l'association de routes à des composants
 - dans un **Switch**, les **Routes** sont toujours **évaluées dans l'ordre**

```
import { BrowserRouter as Router, Switch, Route, Redirect } from 'react-router-dom';

export const App = () => (
  <Router>
    <Switch>
      <Redirect exact from="/" to="/home" />
      <Route path="/home"><HomePage /></Route>
      <Route path="/some-path"><SomeComponent /></Route>
      <Route path={['/another-path', '/this-path-too']}>
        <AnotherComponent />
      </Route>
      <Route path="*"><NotFoundPage /></Route>
    </Switch>
  </Router>
);
```

Composant Route - Variantes

```
export const App = () => (
  <Router>
    <Switch>
      <Route path="/home" component={HomePage} />
      <Route path="/some-path" render={() => <SomeComponent />} />
    </Switch>
  </Router>
);
```

Routes imbriquées - 1/2

- Dans le composant **SomeComponent**, on peut aussi avoir un **Switch**
- Le composant **SomeDetailComponent** attend un paramètre 'id'

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

export const SomeComponent = () => {
  const { path, url } = useRouteMatch();
  return (
    <div>
      <ul>
        {items.map((a) => (
          <li><Link to={`${url}/${item.id}`}>{item.title}</Link></li>
        ))}
      </ul>
      <Switch>
        <Route exact path={path} render={() => <h3>Please select an article
</h3>} />
        <Route path={`${path}/${id}`} component={SomeDetailComponent} />
      </Switch>
    </div>
  );
};
```

Routes imbriquées - 2/2

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

export const SomeDetailComponent = () => {
  const { id } = useParams();
  return (
    <div>
      <h3>{id}</h3>
    </div>
  );
};
```

History, Location & Params

- Dans un composant instancié avec **<Route>** :

- les objets **history**, **location** and **params** sont accessibles avec les **Hooks** de React-router.

```
import { useHistory, useLocation, useParams } from 'react-router-dom';

export const SomeDetailComponent = () => {
  const location = useLocation();
  const history = useHistory();
  const { id } = useParams();

  useEffect(() => {
    console.log(`Active Path: ${location.pathname}`);
    console.log(`ID param: ${id}`);
  }, []);

  const goHome = () => {
    history.push('/home');
  };

  return (
    <div>
      <button onClick={goHome}>Go Home</button>
    </div>
  );
};
```

Pratique - TP5



- Installer le package **react-router-dom**
- Paramétrer le routage pour associer **BookList** à la route **/books**
- Créer le composant **BookDetails** pour afficher les détails d'un livre
 - Associer ce composant à la route **/books/:id**
 - Faire en sorte qu'un click sur un livre de la liste provoque l'affichage des détails de ce livre
 - Récupérer les détails du livre dans le composant **BookDetails** grâce à l'**id** véhiculé sur l'url
- Créer un composant **PageNotFound** pour afficher un message en cas de navigation sur une page invalide
- Ajouter le **Router** dans App qui permet le **Switch** sur ces 3 pages
- Ajouter une **redirection** de **'/'** vers la page **/books**
- Tester

Link & NavLink

- On peut **créer des liens** en utilisant les composants **Link** et **NavLink** dans le code HTML
- On peut naviguer par programmation grâce à la fonction **history.push()**

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

export const AnotherComponent = () => {
  const history = useHistory();

  const goHome = () => {
    history.push("/home");
  };

  return (
    <div>
      <button onClick={goHome}>Go Home</button>
      <Link to="/home">Link to Home</Link>
      <NavLink to="/home">NavLink to Home</NavLink>
    </div>
  );
};
```

- Ces **3 liens** produisent le **même résultat** : le routage vers **/home**.
- **NavLink** produit une balise **<a>** portant la **class="active"** quand la route matche.

Custom Location

- Plutôt que passer un simple *path* au composant **Link** en utilisant la prop **to="path"** vous pouvez lui passer un objet **location** ayant :
 - la propriété **pathname** identique au *path*
 - la propriété **state** pour apporter des informations supplémentaires sur la **location**

```
export const SomeComponent = () => {
  const location = {
    pathname: '/another-route',
    state: { fromSomeComponent: true }
  };
  return <Link to={location} />
};
```



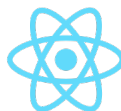
```
export const AnotherComponent = () => {
  const location = useLocation();
  useEffect(() => {
    if (location.state.fromSomeComponent) {
      console.log('You came from SomeComponent');
    }
  }, [location]);
  /* ... */
};
```

Pratique - TP6



- Créer un nouveau composant **src/views/Books**
- Etablir un système de nested routes pour gérer les composants **BooksList** et **BookDetails** dans ce composant
- Créer les nouveaux composants **src/views/Admin**, **src/components/AddBook** et **src/components/AddAuthor**
- Etablir un système de nested routes pour gérer les composants **AddBook** et **AddAuthor** dans ce composant
- Modifier le routage de **App** pour associer **admin** à la route **/admin** et **books** à la route **/books**
- Créer un composant **src/components/NavBar**:
 - **NavLinks** vers les views **admin** et **books**
 - **style personnalisé** lorsque le lien correspond à la route **active**
- Intégrer le composant **NavBar** dans l'App
- Tester

React Forms



<https://fr.reactjs.org/docs/forms.html>

Composants contrôlés - 1/2

- On utilise le **state** du composant React pour **stocker les valeurs des contrôles** (input, select, textarea)
- Lors du **submit**, les valeurs actualisées sont déjà dans le state

```
export const LoginForm = () => {
  const [values, setValues] = useState({ email: '', pwd: '' });

  const handleChange = (e) => {
    setValues({ ...values, [e.target.name]: e.target.value });
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    // Do login
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" name="email" value={values.email} onChange={
        handleChange} />
      <input type="text" name="pwd" value={values.pwd} onChange={handleChange}
      />
      <input type="submit" value="Connect" />
    </form>
  );
};
```

Composants contrôlés - 2/2

- L'usage des balises textarea et select est simplifié

```
return (
  <form onSubmit={handleSubmit}>
    <textarea value={description} onChange={handleChange} />
  </form>
);
```

```
return (
  <form onSubmit={handleSubmit}>
    <select value={color} onChange={handleChange}>
      <option value="green">Green</option>
      <option value="orange">Orange</option>
      <option value="red">Red</option>
    </select>
  </form>
);
```

Composants non contrôlés

- On laisse les contrôles HTML gérer leur propre état
- Lors du **submit** on doit récupérer les valeurs des contrôles grâce à une **ref**

```
export const LoginForm = () => {
  const email = useRef(null);
  const pwd = useRef(null);

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log('email', email.current.value, 'password', pwd.current.value);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" ref={email} defaultValue="" />
      <input type="text" ref={pwd} defaultValue="" />
      <input type="submit" value="Connect" />
    </form>
  );
};
```

Pratique - TP7



- Via un formulaire contrôlé, implémenter la saisie d'un livre dans le composant **AddBook**
- Via un formulaire non contrôlé, implémenter la saisie d'un auteur dans le composant **AddAuthor**

Validation - Solutions

- React ne propose aucune solution en standard pour valider les champs d'un formulaire
- Plusieurs librairies existent :
 - **formik** très complète mais assez verbeuse
 - **redux-form** basée sur le stockage des valeurs dans le store redux
 - **react-hook-form** basée sur l'utilisation de hooks, optimisée pour minimiser les renders
 - **react-form-validator-core** ne marche qu'avec les composants React stateful

Validation - react-hook-form

<https://react-hook-form.com/>

```
const emailPattern = {
  value: /^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i,
  message: 'invalid email address'
};

const pwdValidate = (value) => value.length > 6 ? true : 'invalid password length';

const LoginForm = () => {
  const { handleSubmit, register, errors } = useForm();
  const onSubmit = values => console.log(values);

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input name="email" ref={register({ required: true, pattern: emailPattern })}/>
      {errors.email && errors.email.message}

      <input name="pwd" ref={register({ required: true, validate: pwdValidate })}/>
      {errors.pwd && errors.pwd.message}

      <button type="submit">Submit</button>
    </form>
  );
};
```

Pratique - TP8



- Valider des champs des formulaires **AddBook** et **AddAuthor** en utilisant la librairie **react-hook-form**.

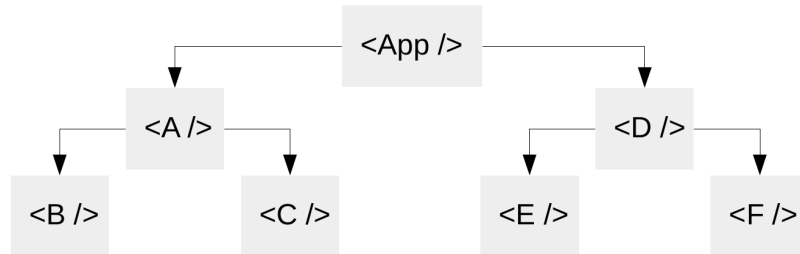
React Redux



<https://react-redux.js.org/>

Pourquoi redux?

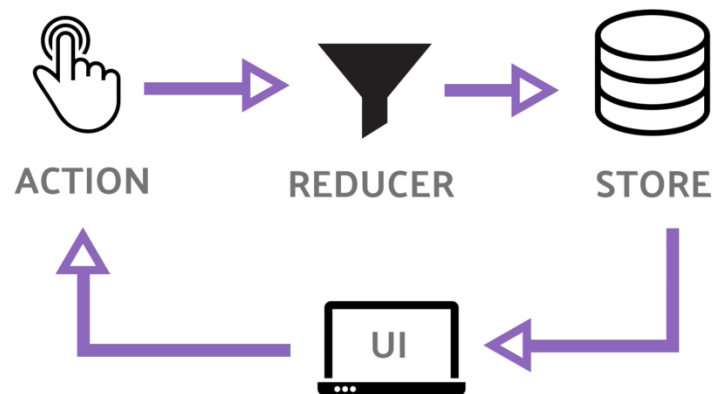
- **React-redux** est une librairie destinée à faciliter la **gestion des données** d'une application React.



```
> npm install --save redux react-redux
```

Redux

- **Redux** permet une gestion de "states globaux"
- Redux est une librairie très légère, et il n'y a que 3 concepts primordiaux à intégrer :
 - le **store** contient le **state** de l'application
 - les **actions** sont **dispatchées** pour provoquer un changement du **state**
 - les **reducers** mutent le **state** en fonction de **l'action dispatchée**



Action

- Une **action** est un **objet** possédant :
 - une propriété *type* (obligatoire),
 - des propriétés optionnelles (payload)
- Un **générateur d'action** est une fonction qui retourne une **action** :

```
export const incrementCounter = () => ({
  type: 'INCREMENT'
});

export const decrementCounter = () => ({
  type: 'DECREMENT'
});

export const resetCounter = (value) => ({
  type: 'RESET',
  value
});
```

Reducer

- Un **reducer** est une fonction qui **change** (mute) l'**état global** du store en **réponse aux actions** envoyées au store grâce à la fonction **dispatch()**
- Il prend en paramètre l'**état global** du store et une **action** et doit retourner un **nouvel état** contenant les modifications provoquées par cette action
- Un reducer doit toujours retourner un state : soit **un nouveau state** si une modification a eu lieu, soit **le state courant** lorsqu'aucune modification n'a été faite

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

export const counterStore = (state = initialState, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return { ...state, counter: state.counter + 1 };
    case 'DECREMENT':
      return { ...state, counter: state.counter - 1 };
    case 'RESET':
      return { ...state, counter: action.value };
    default:
      return state;
  }
};
```

Store - Création

- Le **store** peut être constitué d'un ou de plusieurs **reducers**
- On regroupe les reducers ensemble grâce à la fonction **combineReducers**
- Pour créer le **store**, on appelle la fonction **createStore**

```
import { createStore } from 'redux';
import counter from './reducers/counter';

export default createStore(counter);
```

```
import { createStore, combineReducers } from 'redux';

import counter from './reducers/counter';
import gui from './reducers/gui';

const rootReducer = combineReducers({ counter, gui });

export default createStore(rootReducer);
```

Store - Middlewares

- Un middleware est une **fonction** qui s'insère dans la chaîne de traitement des actions
- Elle est appelée **après le dispatch** de l'action mais **avant l'envoi au reducer**
- Le middleware peut éventuellement modifier l'action avant qu'elle soit donnée au reducer

```
import { createStore, applyMiddleware } from 'redux';
import { todos, initialState } from './reducers';

const logger = ({ getState }) => {
  return next => action => {
    console.log('Dispatch', action);
    const returnValue = next(action);
    console.log('State after dispatch', getState());
    return returnValue;
  }
};

const store = createStore(todos, initialState, applyMiddleware(logger))
```

Store - Exemple

- Exemple de configuration d'un store Redux

```
import { applyMiddleware, combineReducers, createStore } from 'redux';
import { composeWithDevTools } from 'redux-devtools-extension';
import { createLogger } from 'redux-logger';
import thunkMiddleware from 'redux-thunk';
import * as reducers from './reducers';

export const configureStore = (initialState) => {
  const middlewares = [thunkMiddleware];

  if (process.env.REACT_APP_LOG_REDUX) {
    middlewares.push(createLogger({ collapsed: true, level: 'warn' }));
  }

  let reduxMiddleware = applyMiddleware(...middlewares);

  if (process.env.NODE_ENV !== 'production') {
    reduxMiddleware = composeWithDevTools(reduxMiddleware);
  }

  return createStore(combineReducers(reducers), initialState, reduxMiddleware);
}
```

Components - Provider

- **Redux** inclut le composant **Provider** pour encapsuler l'application et rendre disponible le **store** dans toute l'application.

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';

import { configureStore } from './redux/store';
import App from './App';

const rootElement = document.getElementById('root');
const store = configureStore({});

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  rootElement
);
```

Components - Connect

- La fonction **connect()** connecte un composant au store.
- Un composant connecté au store peut alors :
 - accéder à l'état global du store : **mapStateToProps**,
 - dispatcher une action : **mapDispatchToProps**.


```
import React from 'react';
import { connect } from 'react-redux';
import { addTodo } from './actions';

const MyComponent = ({ todos, addTodo }) => {
  // état et action utilisable dans le composant
  // addTodo('NEW TODO')
  // todos.map(...)
};

const mapStateToProps = (state) => ({
  todos: state.user.todos // état ajouté dans les props
});

const mapDispatchToProps = {
  addTodo // action ajoutée dans les props
};

export default connect(mapStateToProps, mapDispatchToProps)(MyComponent);
```

Components - Hooks

- **useSelector** = équivalent de **mapStateToProps**
- **useDispatch** = équivalent de **mapDispatchToProps**

```
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { addTodo } from './actions';

const MyComponent = () => {
  const todos = useSelector((state) => state.user.todos);
  const dispatch = useDispatch();

  return (
    <ul>{todos.map((todo) => <li key={todo.id}>{todo.text}</li>)}</ul>
    <button onClick={() => dispatch(addTodo('NEW TODO'))}></button>
  )
};
```

Immer

- **Immer** est un module mis à disposition par les créateurs de **redux** pour simplifier la **deep** copie du state dans les reducers.
- `npm install immer`

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

export const counterStore = (state = initialState, action) =>
  produce(state, (draft) => {
    switch (action.type) {
      case 'INCREMENT':
        draft.counter = draft.counter + 1;
        break;
      case 'DECREMENT':
        draft.counter = draft.counter - 1;
        break;
      case 'RESET':
        draft.counter = action.value;
        break;
    }
  });
```

Pratique - TP9



- Installer **redux** et **react-redux**
- Créer un répertoire **redux** dans le projet
- Créer l'action **setLoggedInUser** acceptant en paramètre un objet contenant à minima un attribut **userName**
- Créer un reducer **auth** destiné à stocker l'utilisateur couramment connecté dans une variable **loggedInUser**
- Configurer **redux** pour gérer ce reducer **auth** ainsi que le middleware **redux-logger**
- Connecter au store le composant **NavBar** pour qu'il affiche soit:
 - un bouton **Login** lorsque **loggedInUser** est undefined
 - un texte **Hello <userName>** lorsque **loggedInUser** contient un objet
- Tester

redux-thunk



<https://github.com/reduxjs/redux-thunk>

Principe

- De base, les **actions** retournent un objet de façon **synchrone**
- **redux-thunk** est un **middleware** permettant aux actions de retourner un objet de manière **asynchrone**

```
import { createStore, applyMiddleware } from 'redux';
import thunk from 'redux-thunk';
import reducers from './reducers';

export const configureStore = (initialState) => {
  return createStore(reducers, initialState, applyMiddleware(thunk));
}
```

- Voir le chapitre précédent pour une utilisation combinée des middlewares

Action asynchrone

- Une action peut maintenant retourner une fonction (*thunk*) qui retourne une promesse

```
import { articlesApi } from './api/articles';

const loadArticles = (userId) => (dispatch, getState) =>
  articlesApi.getUserArticles(userId).then(
    (articles) => dispatch({
      type: 'ARTICLES_LIST',
      articles
    })),
    (error) => dispatch({
      type: 'ARTICLES_LIST_ERROR',
      error: error.message
    })
  );
```

Custom injection

- On peut demander à **redux-thunk** d'injecter un troisième paramètre *custom* dans le **thunk**
- Configuration au niveau de la déclaration du middleware
- Ce paramètre peut être de tout type et sera fourni tel quel au thunk

```
const store = createStore(reducer, applyMiddleware(
  thunk.withExtraArgument({ usersApi, productsApi })
));
```

```
const loadUser = (id) =>
  (dispatch, getState, { usersApi }) =>
    usersApi.getUser(id).then(...);
```

Pratique - TP10 1/2



- Ajouter une table **users** dans la base de donnée du **json-server**
 - Ajouter au moins **deux users** { id, email, password, userName }
- Créer un service **userService** avec une méthode **findUser(email, password)** retournant l'utilisateur correspondant aux paramètres
- Créer dans le service **authService** une méthode **connect(email, password)** appelant la fonction **userService.findUser()**
- Créer un composant **Login** permettant de saisir **email** et **password** d'un utilisateur

Pratique - TP10 2/2



- Créer une action asynchrone **authConnect()** qui appellera la fonction **authService.connect()**
- Lui faire dispatcher une action **authConnectSuccess()** ou **authConnectError()** selon le résultat
- Au **submit** du composant **Login**, dispatcher l'action **authConnect()**
- Affecter la route **/login** au composant **Login**
- Modifier **NavBar** pour router sur **/login** lors du click sur le bouton 'Login'

react-il8next



<https://react.il8next.com/>

Localization

- **react-i18next** est un framework de localization pour React
- Basé sur le module **i18next**
- On n'écrit plus le texte "en dur" dans le JSX mais on donne à la place des **clés** de traduction
- Le texte à proprement parler se trouve dans des **fichiers JSON**

```
> npm install --save react-i18next i18next
```

- Pour charger automatiquement le fichier de traduction placé sur le serveur:

```
> npm install --save i18next-browser-languagedetector i18next-http-backend
```

Configuration 1/2

- Créer un fichier **i18n.js** sur la racine du projet
- Importer ce fichier dans votre **index.js**

```
import i18n from 'i18next';
import { initReactI18next } from 'react-i18next';
import detector from 'i18next-browser-languagedetector';
import backend from 'i18next-http-backend';

i18n
  .use(detector)
  .use(backend)
  .use(initReactI18next) // passes i18n down to react-i18next
  .init({
    lng: 'fr',
    fallbackLng: 'fr',    // use fr if detected lng is not available
    saveMissing: true,    // send not translated keys to endpoint
    interpolation: {
      escapeValue: false // react already safes from xss
    },
    react: {
      useSuspense: false // don't use Suspense
    }
  });

export default i18n;
```

Configuration 2/2

- Créer le répertoire **/public/locales**

- Créer un répertoire par locale gérée : **fr**, **en**...
- Créer un fichier **translation.json** (nom du **namespace** par défaut)
- On peut gérer plusieurs **namespaces** pour chaque locale
- Utiliser un namespace permet de découper les fichiers de traduction par domaines fonctionnels

```
{
  "menu": {
    "link-login": "Login",
    "link-logout": "Logout"
  },
  "buttons": {
    "save": "Save",
    "cancel": "Cancel"
  }
}
```

Utilisation 1/2

- Remplacer tous les textes en dur par l'appel de la fonction **t()** mise à disposition par **react-i18next**
- On peut soit utiliser un hook, soit un HOC, soit le composant Translation

```
import { useTranslation } from 'react-i18next';

export const MyComponent = () => {
  const { t, i18n } = useTranslation();
  return <h1>{t('welcome')}</h1>
};
```

```
import { withTranslation } from 'react-i18next';

export MyComponent = withTranslation()(({ t, i18n }) => {
  return <h1>{t('welcome')}</h1>
});
```

```
import { Translation } from 'react-i18next';

export MyComponent = () => (
  <Translation>
    {(t, { i18n }) => <p>{t('my translated text')}</p>}
  </Translation>
);
```

Utilisation 2/2

- Le changement de locale se fait dynamiquement par l'appel à la fonction **changeLanguage** de l'instance **i18n** fournie par le hook **useTranslation** (ou autre méthode utilisée)

```
import { useTranslation } from 'react-i18next';

export const MyComponent = () => {
  const { t, i18n } = useTranslation();

  const switchLang = (locale) => {
    i18n.changeLanguage(locale);
  };

  return <h1 onClick={() => switchLang('en')}>{t('welcome')}</h1>
};
```

Pratique - TP11



- Mettre en place la localization avec au moins deux langages, **fr** et **en**
- Remplacer les textes par des clés de traduction
- Permettre à l'utilisateur de switcher entre ces langages